UNIVERSITÀ DEGLI STUDI DI MESSINA

Dipartimento BIOMORF

DOTTORATO DI RICERCA IN
BIOINGEGNERIA APPLICATA ALLE SCIENZE MEDICHE

CICLO XXXVII
ING-INF/05

**Compute Continuum in Bioengineering:
Enhanced Motion Capture
and Real-Time Data Processing
on Cloud-Edge Infrastructures**

Candidato
Ing. Luca D'Agati

Supervisore
Ch.mo Prof. Ing. Giovanni Merlino

Coordinatore del corso:
Ch.mo Prof. Michele Gaeta

Anno Accademico 2023/2024

Abstract

The constant evolution of cloud-edge computing, correlated with advancements in bioengineering technologies, has opened new opportunities for real-time data processing and analysis in critical applications. This dissertation explores the integration of motion capture (MoCap) systems with Internet of Things (IoT) architectures, leveraging cloud-edge platforms such as OpenStack and Stack4Things to address scalability and efficiency challenges across domains like healthcare and smart cities.

At the core of this research is the development of MocapMe, a framework designed to optimize motion capture and analysis through markerless technologies powered by deep learning models. This framework can allow a wide range of applications, including clinical rehabilitation, sports performance analysis, and real-time animation, showing notable accuracy and system responsiveness improvements. Moreover, the MocapMe integration within a Compute Continuum architecture aims to minimize latency, promoting real-time feedback for critical motion capture tasks in medical and sports contexts.

Enabling technologies such as LoRaWAN and peer-to-peer (P2P) networking are investigated to provide robust communication in distributed systems, even in resource-constrained environments. Moreover, the dissertation explores the dynamic virtualization and management of IoT resources through the I/Ocloud paradigm, ensuring seamless scalability and efficient data processing across cloud and edge environments.

The research also delves into the Compute Continuum prospect of integrating IoT, cloud, and edge computing in diverse applications. Case studies cover a range of use cases, from smart city infrastructures to healthcare scenarios, including sit-to-stand (STS) analysis, demonstrating the practical advantages of the developed technologies in these domains.

Contents

Acronyms **10**

1 Introduction **14**
1.1 Research Context . . . 14
1.2 Research Objectives . . . 14
1.3 Thesis Structure . . . 15

2 The Compute Continuum: Integration of IoT, Cloud, and Edge Computing **17**
2.1 Introduction to the Compute Continuum . . . 17
2.2 IoT and Cloud Computing . . . 17
2.3 Cloud Computing . . . 18
2.3.1 Definition . . . 18
2.3.2 Cloud Services . . . 18
2.3.3 Resource Provisioning in the Compute Continuum . . . 19
2.3.4 Types of Cloud Models . . . 19
2.4 IoT Cloudification in the Compute Continuum . . . 20
2.4.1 Motivation . . . 20
2.4.2 The Device-Centric Approach . . . 21
2.4.3 I/Ocloud: A Multi-Tenant IoT Solution . . . 22
2.5 Enabling Technologies . . . 25
2.5.1 OpenStack . . . 25
2.5.2 Stack4Things . . . 26
2.5.3 I/Ocloud Use Case: Software-Defined Cities . . . 28

3 Cloud-Edge Architectures: Foundations and Applications **31**
3.1 The Role of Horizontally Scalable Architectures in IoT Systems . . . 31
3.1.1 Challenges in Scaling IoT Architectures . . . 31
3.1.2 Horizontal Scalability in Smart Cities . . . 32
3.2 Enabling Technologies for Scalable Architectures . . . 32
3.2.1 I/Ocloud: Virtualized IoT Resource Management . . . 32
3.2.2 LoRaWAN: Enabling Mobility in IoT Monitoring . . . 32
3.2.3 Peer-to-Peer Networking: Enabling Decentralized Scalability . . . 33
3.2.4 Edge Computing: Enhancing Real-Time Responsiveness . . . 33
3.3 Building Towards Practical Applications . . . 33

3.3.1 Real-Time Data Processing in Bioengineering . . . 34
3.3.2 Telemedicine and Remote Health Monitoring . . . 34
3.4 Stacking Technologies for Horizontally Scalable Architectures . . . 34
3.5 Connecting Foundations to Applications . . . 35

4 Case Studies and Applications 36
4.1 Communication Systems and IoT Connectivity . . . 36
4.1.1 Examples of PAN, WAN, and Roaming Communication Technologies . . . 36
4.1.2 LoRaWAN Roaming Solutions Without Network Server Dependencies . . . 37
4.1.3 API-Driven LoRaWAN Roaming for Device Mobility in IoT Networks . . . 42
4.1.4 IoT Diagnostics in Energy Substations via BLE and Cloud Integration . . . 50
4.1.5 Meshing IoT Networks with WebRTC-Based P2P Overlay Networks . . . 54
4.2 Smart Cities and Urban Infrastructure . . . 60
4.2.1 IoT/Cloud-powered Green Mobility Solutions in Smart Cities . . . 60
4.2.2 IPA-enabled IoT Resource Management in Smart Environments . . . 65
4.2.3 Edge-Driven Data Management in Smart Cities . . . 71
4.2.4 Function-as-a-Service (FaaS) on top of I/Ocloud . . . 75
4.2.5 Self-Conscious Cyber-Physical Systems for Sustainable Energy Management in Communities . . . 82
4.2.6 Innovative Urban Intelligence Framework with OpenStack and IoT Integration 88
4.3 Emergency Response and Civil Protection . . . 93
4.3.1 Integrating Cloud Computing with UAV Autopilot Systems for Coordinated Mission Efficiency . . . 93
4.3.2 Decentralized Emergency Manufacturing and Supply Chains Using 3D Printing and Blockchain . . . 98
4.4 Healthcare: Additive Manufacturing and Telemedicine . . . 102
4.4.1 Torque Force Applied Evaluation in Dental Implant Employing IoT Device . 102
4.4.2 Web of Things Architecture for Remote Healthcare Monitoring . . . 108
4.4.3 Motion Capture-Based Design of Hip Prostheses . . . 112

5 MocapMe Framework 118
5.1 Motion Capture: Techniques and Applications . . . 118
5.2 Edge-Centric Optimization of Motion Capture: A MocapMe Integration Perspective . . . 120
5.3 Enabling Architectures for ML-based Bioengineering: LoRa, P2P Networking, and I/Ocloud . . . 120
5.3.1 LoRa as an Enabling Technology . . . 120
5.3.2 Peer-to-Peer (P2P) Communication . . . 121
5.3.3 I/Ocloud as a Supporting Infrastructure . . . 121
5.4 MocapMe: Machine Learning for Markerless Motion Tracking . . . 122
5.4.1 Methodology . . . 123
5.4.2 Data Collection and Preparation . . . 123
5.4.3 Model Architecture . . . 123
5.4.4 Training Strategy . . . 124
5.4.5 Results and Evaluation . . . 125

5.4.6 Deployment and Modular Design . . . 125
5.4.7 Computational Efficiency and Precision . . . 126
5.4.8 Lessons Learned . . . 127
5.4.9 Deterministic Real-time Performance in MocapMe . . . 128
5.5 Integration of Motion Capture in Edge Computing . . . 131
5.5.1 3D Motion Capture with MocapMe: Integration with Cloud-Edge Computing 132
5.5.2 Graphical User Interface (GUI) . . . 136

6 Conclusions and Future Developments 139
6.1 Key Findings and Contributions . . . 140
6.2 Future Challenges and Opportunities . . . 140
6.2.1 Enhancing Performance and Scalability . . . 141
6.2.2 Security and Ethical Considerations . . . 141
6.3 Implications for Bioengineering . . . 141
6.4 Future Directions . . . 142
6.4.1 Advanced AI and Machine Learning Integration . . . 142
6.4.2 Blockchain for Secure Data Management . . . 142
6.4.3 Scalability and Interoperability Solutions . . . 142
6.4.4 Ethical and Regulatory Frameworks . . . 143
6.5 Concluding Remarks . . . 143

A Appendix A: Cloud Processing for Motion Capture Framework 144
A.0.1 Overview of Cloud-Based Processing . . . 144
A.0.2 Keypoint Extraction . . . 144
A.0.3 Data Filtering and Smoothing . . . 145
A.0.4 Angle Calculation for Biomechanical Analysis . . . 146
A.0.5 Integration with Cloud Storage and APIs . . . 146

B Appendix: Python Code for Synchronous Video Acquisition and Cloud Upload 148
B.0.1 Code Explanation . . . 151

C Appendix: Python Code for Camera Calibration and 3D Reconstruction 152
C.0.1 Code Explanation . . . 156

List of Figures

2.1 Cloud-based IoT architectures: (a) IoT devices as a data source for the Cloud, (b) IoT as a remote interface to Cloud-based applications, and (c) IoT as an extension of Cloud resources. . . . 21
2.2 I/Ocloud instances with attached I/O resources. . . . 24
2.3 OpenStack architecture. . . . 26
2.4 Stack4Things subsystems. . . . 27
2.5 Stack4Things architecture overview. . . . 28
2.6 Stack4Things Cloud-side architecture. . . . 29
2.7 Lightning-Rod architecture. . . . 30
2.8 The Software-Defined City paradigm. . . . 30

4.1 NetworkServer-agnostic LoRaWAN Roaming Architecture . . . 38
4.2 Passive roaming: setup procedure. Source: [42] . . . 39
4.3 Passive roaming: teardown procedure, initiated by the . Source: [42] . . . 39
4.4 Proposed ABP-based data packet roaming (without SLA setup). Source: [42] . . . 40
4.5 Proposed ABP-based data packet roaming (including SLA setup).Source: [42] . . . 40
4.6 Proposed architecture for decoupling GWs from NSs to support dynamic and flexible deployments. Source: [21] . . . 43
4.7 Proposed ABP (Activation By Personalization)-based data packet roaming using the API-driven Gateway Bridge service. Source: [21] . . . 44
4.8 Test Network Topology for Experimental Validation. Source: [21] . . . 46
4.9 Latency at the Chirpstack Server for a Single Roaming Device (No Home Network Devices). Source: [21] . . . 47
4.10 Latency at the Chirpstack Server for Two Roaming Devices and Two Home Network Devices. Source: [21] . . . 47
4.11 Average Packet Latency for Roaming and Non-Roaming Devices in Simulated Scenarios. Source: [21] . . . 48
4.12 Latency Distribution Over Time for Roaming and Non-Roaming Devices in Simulated Scenarios. Source: [21] . . . 49
4.13 Sequence diagram showing interactions between the Cloud (Stack4Things), Substation (Arancino board), and Android Smartphone. . . . 50
4.14 Scenario 1: Continuous Cloud Connectivity for Remote Monitoring . . . 51
4.15 Scenario 2: On-Demand Cloud Connectivity for On-Site Diagnostics . . . 51
4.16 BLE Service and Characteristic Structure. . . . 52
4.17 BLE Communication Sequence between Arancino and Mobile Device . . . 53

4.18 BLE GATT Server Structure on Arancino Board. Source: [23] 54
4.19 P2P connection between two IoT devices using STUN and signaling agents. 55
4.20 Network configuration with three IoT devices and a virtual switch using STUN signaling. 56
4.21 Node-side architecture of the S4T framework. 57
4.22 Complex network architecture with six IoT devices and virtual switches. 58
4.23 Message exchange comparison between centralized and P2P overlay networks. . . . 59
4.24 Architecture of the IoT/Cloud-powered infrastructure integrating OpenStack and Stack4Things (S4T) for managing IoT nodes across a smart city environment. Source: [20] 61
4.25 System architecture for EasyTaxi, showing the integration of predictive analytics and IoT data for optimizing taxi positioning. Source: [20] 62
4.26 Linear regression model comparison for predicting taxi bookings. The degree-1 model (green) demonstrates the best performance. Source: [20] 63
4.27 Network topology for the PlaceHolder service showing integration with the IoT and blockchain networks. Source: [20] 63
4.28 Network topology for the Weather Station service, showing the integration of IoT devices with cloud resources for real-time air quality monitoring. Source: [20] . . . 64
4.29 Stack4Things cloud-side architecture, illustrating the interactions between IoTronic and Lightning-Rod via WebSockets and WAMP for managing IoT resources. Source: [60] 66
4.30 High-level architecture of an SDI/O-enabled infrastructure. Source: [60] 67
4.31 Overview of the EIPA architecture in an accommodation facility. Source: [60] . . . 68
4.32 Resource assignment workflow in an accommodation facility. Source: [60] 69
4.33 User interaction with the environment mediated by the Virtual Connector. Source: [60] 69
4.34 Elapsed time as a function of device number involved. Source: [60] 70
4.35 Flow Shaping and Plugin Injection average values. Source: [60] 70
4.36 The OpenStack/Elastic Stack-powered Smart City system architecture. Source: [7] 71
4.37 Elastic Stack (ELK) architecture. Source: [7] 73
4.38 Kibana dashboard showing smart lighting poles in the Lorenteggio area. Source: [7] 73
4.39 Real-time graph showing electromagnetic field intensity in the Lorenteggio district. Source: [7] 74
4.40 The area of the Lorenteggio district being monitored. Source: [7] 74
4.41 System architecture for FaaS in Edge/Fog computing environments. Source: [41] . . 76
4.42 Runtime creation and function execution workflows. Source: [41] 78
4.43 Runtime instantiation workflow. Source: [41] 79
4.44 A function execution workflow. 80
4.45 Response times for function execution under different concurrency configurations. Source: [41] 80
4.46 Failure rates for function execution under different concurrency levels. Source: [41] . 81
4.47 Performance comparison between traditional HTTP server and FaaS platform. Source: [41] 81

4.48 System architecture of the Renewable Energy Community (REC). ECEM: Energy Consumption Estimation Module, EPREM: Energy Production Estimation Module, TEANS: Threshold Evaluator and Notification System. Source: [16] 83
4.49 Energy estimation values for both consumption and production. The cloud ensures that consumption aligns with available production. Source: [16] 84
4.50 Gap seasonality of a real estate unit by hour, month, and week of the year. 87
4.51 The OpenStack-based Urban Intelligence Architecture. Source: [61] 89
4.52 Instantiation of a new application-level workflow operated by an Urban Data Scientist. Source: [61] 91
4.53 Architectural integration of PX4-Avoidance with I/O cloud, illustrating the interaction between UAV onboard systems and cloud computing services. Source: [19] 94
4.54 Subdivision of a generic mission path into two segments (V1 and V2), demonstrating the path-splitting strategy. Source: [19] 96
4.55 Cumulative distance vs. cumulative mission time, illustrating the efficiency gains through path-splitting. Source: [19] 96
4.56 System architecture of the Air Factories 2.0 platform, integrating Ethereum and Hyperledger Fabric blockchains. Source: [22] 98
4.57 Proof of Print (PoP) process in Air Factories 2.0, ensuring the quality and authenticity of the printed products. Source: [22] 100
4.58 Voting process for new product proposals in Air Factories 2.0. Source: [22] 101
4.59 Wiring configuration of the load cell, HX711 ADC, and Arduino microcontroller, optimized for signal integrity and minimal noise. Source: [45] 103
4.60 Orthogonal projection view of the 3D-printed torque wrench, designed using Siemens NX software. Source: [45] 106
4.61 Loads and boundary conditions applied during Finite Element Analysis (FEA) of the torque wrench. Source: [45] 106
4.62 Deformation along the z-axis under maximum applied load, showing a maximum deflection of 1 mm. Source: [45] 107
4.63 IoTronic and Designate integration design for cloud-side IoT management. Source: [18] 108
4.64 S4T Lightning-Rod architecture for node-side operations and interaction with cloud-based IoTronic subsystem. Source: [18] 109
4.65 Telemedicine workflow utilizing the Stack4Things WoT routing mechanism. Source: [18] 110
4.66 Keypoints detected by OpenPose during motion capture, enabling the analysis of joint kinematics. Source: [46] 113
4.67 Knee angle calculation using vector geometry based on motion capture data. Source: [46] 114
4.68 Parametric multibody model used for simulating gait dynamics and force transmission through the hip joint. Source: [46] 114
4.69 Mesh refinement process during FEM analysis to achieve stress distribution accuracy. Source: [46] 115
4.70 Optimization process for shape refinement of the prosthesis. Source: [46] 116
4.71 Lattice optimization applied to reduce prosthesis weight. Source: [46] 116

5.1 COCO Pose Model estimating 25 key points. Source: [3] . . . 119
5.2 Schematic representation of distances and angles used during video acquisition. . . . 123
5.3 Training and validation loss during the training process, illustrating convergence. Source: [44] . . . 125
5.4 Processing time comparison between OpenPose and DLC. Source: [44] . . . 126
5.5 Confidence of selected keypoints for OpenPose and DLC-based MocapME. Source: [44] . . . 127
5.6 Stability of ankle and foot keypoints for OpenPose and MocapME (DLC-based). Source: [44] . . . 127
5.7 FPS Ratio (Processed/Original) vs. Video. The red dashed line represents the real-time threshold (FPS ratio = 1). . . . 129
5.8 Computation Time per Second of Video vs. Resolution (Total Pixels). . . . 130
5.9 Computation Time per Frame per Pixel vs. Video. . . . 131
5.10 FPS Processed vs. FPS Original. . . . 132
5.11 3D camera setup with Logitech C920 cameras and rigid frame. . . . 133
5.12 Aruco board used during the camera calibration process. . . . 134
5.13 Architecture . . . 136
5.14 Sequence diagram of the MocapMe workflow . . . 137
5.15 The MocapMe Graphical User Interface. The main window (left) offers options for calibration, video processing, and analysis, while the calibration window (right) allows users to set the destination folder, select cameras, and capture calibration images. . . . 137

List of Tables

2.1 Management responsibilities across different Cloud models: On-Premises, IaaS, PaaS, and SaaS. 19

4.1 ChirpStack Passive Roaming 41
4.2 Proposed Roaming Solution 41
4.3 Hardware Configuration for Experimental Setup 45
4.4 BLE Advertising Data Packet Structure. 52
4.5 Comparison of message propagation times between centralized and P2P overlay networks. 59
4.6 Performance metrics for energy consumption (BiLSTM) and production (LSTM) models. 85
4.7 Comparative Analysis of Mission Completion Times 95
4.8 Printer Scoring Criteria for Job Allocation 101
4.9 Mechanical properties of AISI 316L used in FEM analysis. 115

5.1 Architecture of ResNet34 employed in the study, comparing it with deeper ResNet models. 124
5.2 Training results of the DeepLabCut model. 125

Listings

4.1 Example hypermedia API response enabling dynamic resource discovery 111
A.1 Keypoint Extraction 144
A.2 Data Filtering and Smoothing 145
A.3 Angle Calculation 146
A.4 Upload Processed Data to Cloud Storage 147
B.1 Synchronous Video Acquisition 148
C.1 Camera Calibration and 3D Reconstruction 152

Acronyms

IoT Internet of Things

IaaS Infrastructure-as-a-Service

PaaS Platform-as-a-Service

SaaS Software-as-a-Service

NIST National Institute of Standards and Technology

XaaS Everything-as-a-Service

VM Virtual Machine

VN Virtual Node

WAN Wide Area Network

SBC Single-Board Computer

NAT Network Address Translation

OS Operating System

FUSE Filesystem in User-space

WAMP Web Application Messaging Protocol

P2P Peer-to-Peer

LoRaWAN Long Range Wide Area Network

PAN Personal Area Network

BLE Bluetooth Low Energy

NFC Near Field Communication

NS Network Server

LPWAN Low-Power Wide-Area Network

API Application Programming Interface

WebRTC Web Real-Time Communication

IPA Intelligent Personal Assistant

SDI/O Software-Defined I/O

S4T Stack4Things

ELK Elastic Stack

FaaS Function-as-a-Service

DCPS Distributed Cyber-Physical System

UAV Unmanned Aerial Vehicle

WoT Web of Things

REST Representational State Transfer

HATEOAS Hypermedia as the Engine of Application State

AM Additive Manufacturing

ML Machine Learning

FEM Finite Element Method

PoP Proof-of-Print

IPFS InterPlanetary File System

AFT Air Factories Tokens

ERC-20 standard token Ethereum

ECEM Energy Consumption Estimation Module

EPREM Energy Production Estimation Module

TEANS Threshold Evaluator and Notification System

LSTM Long Short-Term Memory

BiLSTM Bidirectional Long Short-Term Memory

RMSE Root Mean Square Error

MAE Mean Absolute Error

GAP Global Active Power

DC Direct Current

AC Alternating Current

GATT Generic Attribute Profile

MAC Media Access Control

UUID Universally Unique Identifier

RPC Remote Procedure Call

UI User Interface

CLI Command Line Interface

MQTT Message Queuing Telemetry Transport

ACL Access Control List

GW Gateway

ED End-Device

AI Artificial Intelligence

JSON JavaScript Object Notation

NoSQL Not Only SQL

Docker containerizzazione

Kibana parte dell'ELK Stack

DLC DeepLabCut

STS Sit-to-Stand

OP OpenPose

CNN Convolutional Neural Network

RGB Red, Green, Blue

SGD Stochastic Gradient Descent

MSE Mean Squared Error

GPU Graphics Processing Unit

ResNet34 Residual Network 34-layer

FLOPs Floating Point Operations

CPS Cyber-Physical Systems

DBaaS Database-as-a-Service

FDM Fused Deposition Modeling

ADC Analog-to-Digital Converter

PGA Programmable Gain Amplifier

I2C Inter-Integrated Circuit

GPIO General Purpose Input/Output

FEA Finite Element Analysis

PLA Polylactic Acid

RSSI Received Signal Strength Indicator

SNR Signal-to-Noise Ratio

Tx/Rx Transmission/Reception

STUN Session Traversal Utilities for NAT

ICE Interactive Connectivity Establishment

TURN Traversal Using Relays around NAT

NV Network Virtualization

SDN Software-Defined Networking

VLAN Virtual Local Area Network

VXLAN Virtual Extensible Local Area Network

OVS Open vSwitch

QoS Quality of Service

DNS Domain Name System

DNSaaS DNS as a Service

LR Lightning-Rod

HTTPS Hypertext Transfer Protocol Secure

X.509 certificati X.509

NGINX reverse proxy

CA Certificate Authority

Chapter 1

Introduction

1.1 Research Context

In recent years, advancements in cloud-edge computing and the growth of the IoT have converged to create scalable, efficient solutions across multiple domains, including smart cities, healthcare, and emergency contexts. This dissertation examines how these technologies, also applied to motion capture (MoCap) systems, can be leveraged to provide real-time data processing and enhanced performance for complex bioengineering and smart environments applications [20, 41, 44]. The perspective is integrating these technologies into decentralized architectures based on OpenStack and Stack4Things and exploring their impact on application domains.

The transition from centralized cloud platforms to decentralized edge computing enables more responsive systems, particularly in environments requiring real-time analysis and low-latency communication. This shift is mainly required in bioengineering applications, where real-time feedback from sensors and MoCap systems can significantly improve outcomes in clinical settings, sports performance, and emergency response [18]. By employing cloud-edge architectures, systems can process large volumes of data closer to the source, reducing the dependence on centralized cloud resources and allowing for better scalability, reliability, and system efficiency.

In this context, MoCap technologies, initially developed for applications like animation and biomechanics, have evolved significantly with the advent of machine learning and deep learning frameworks. Markerless MoCap, powered by tools such as OpenPose and DeepLabCut, has reduced the need for traditional tracking markers, making it more suitable for a broader range of applications, including medical diagnostics, rehabilitation, and human-computer interaction [44]. Integrating MoCap with cloud-edge platforms is promising in delivering high-precision motion analysis in real-time, directly at the edge, enhancing the usability and accuracy of these systems in both controlled and uncontrolled environments.

1.2 Research Objectives

This dissertation aims to address several key research objectives that lie at the intersection of cloud-edge computing, communication technologies, and MoCap:

- **Investigate integrating cloud-edge platforms with MoCap systems:** The research explores how cloud and edge architectures, such as OpenStack and Stack4Things, can be

combined with MoCap systems to enhance real-time data processing, reduce latency, and improve scalability.

- **Developing a markerless MoCap system:** The goal is to create a MoCap system that leverages deep learning approaches, i.e., Resnet models, to track human movement accurately without requiring physical markers, optimizing system performance for clinical and human movement-related use cases.
- **Investigating the benefits of cloud-edge architectures for real-time applications:** This objective focuses on exhibiting the applicable advantages of cloud-edge systems, highlighting their role in real-time feedback, scalability, and computational efficiency in distributed MoCap networks.
- **Applying these technologies in real-world use cases:** This objective involves evaluating the developed system's practical applications in real-world scenarios, such as smart city applications, emergency management, biomechanics, and telemedicine. The analysis will demonstrate the impact of cloud-edge architectures in enhancing performance in heterogeneous environments.

1.3 Thesis Structure

The thesis is organized as follows, covering the core areas of cloud-edge integration, IoT, MoCap, and practical applications:

- **Chapter 2 - The Compute Continuum: Integration of IoT, Cloud, and Edge Computing:** This chapter introduces the Compute Continuum, a paradigm that integrates cloud, edge, and IoT technologies for distributed computing. It explores the architectural frameworks that enable scalable and flexible data processing.
- **Chapter 3 - Cloud-Edge Architectures for Bioengineering Applications:** This chapter examines the design of cloud-edge architectures that are optimized for IoT mobility and low-power, long-range communication technologies such as Long Range Wide Area Network (LoRaWAN). It covers resource management, system scalability, and the integration of edge computing to support applications.
- **Chapter 4 - Case Studies and Applications:** This chapter presents real-world applications of the developed systems. It includes case studies on using cloud-edge architectures in smart cities, energy management, emergency response, mobility, and healthcare, focusing on their impact on operational efficiency and scalability.
- **Chapter 5 - MocapMe at the Edge:** The fifth chapter explores the MocapMe framework, a key contribution of this research, integrating MoCap systems with edge computing. The chapter details the technical implementation, methodologies, and outcomes of using MocapMe, focusing on its real-time, markerless motion-tracking capabilities.
- **Chapter 6 - Conclusions and Future Developments:** This chapter provides a comprehensive summary of the key contributions of the thesis, including integrating cloud-edge

computing with MoCap technologies. It also discusses potential future developments, such as extended Artificial Intelligence (AI)-driven motion analysis, the use of blockchain for data security, and challenges with ethical and privacy considerations. Additionally, the chapter outlines future research directions, focusing on enhancing performance, scalability, and regulatory compliance in healthcare applications.

Chapter 2

The Compute Continuum: Integration of IoT, Cloud, and Edge Computing

2.1 Introduction to the Compute Continuum

The concept of the Compute Continuum denotes an evolution in computing infrastructure. It blurs the traditional boundaries between centralized Cloud computing and distributed Edge and IoT resources. This paradigm envisions seamlessly integrating Cloud, Edge, and IoT environments. It enables computational tasks and data processing across a spectrum of resources, from centralized data centers to devices located at the network's periphery. This chapter explores the role of the Compute Continuum in integrating IoT, Cloud, and Edge computing to address the growing demands of modern applications, particularly in managing vast amounts of data generated by IoT devices.

2.2 IoT and Cloud Computing

The exponential growth of IoT devices, projected to reach billions, propels the domain of Big Data, where centralized processing, analytics, and storage are essential [52]. The quick adoption of IoT services introduces significant challenges in storing, processing, and accessing large volumes of data. In this context, the Cloud computing paradigm [17, 35] plays a primary role in enhancing the effectiveness of IoT by providing essential facilities and services. Integrating Cloud platforms with IoT acts as a catalyst, offering numerous data management and processing advantages. IoT devices with sensing capabilities upload collected data about their surrounding environments to the Cloud, serving as input for intelligent monitoring and actuation systems. This IoT-Cloud integration aims to transform IoT data into actionable insights, driving cost-effective services and applications.

As service-oriented computing trends continue to expand, mainly through the Everything-as-a-Service (XaaS) model [2], various solutions have emerged that adapt the "as-a-Service" paradigm to IoT environments. However, many approaches consider the Cloud merely as an extended application domain, acting as a data sink where IoT-generated data is stored and retrieved following a data-centric approach [56, 47, 8]. While such solutions offer extensive resources for processing IoT data, they are limited in scope, as users cannot customize the business logic running on IoT

devices. Consequently, the stored data in the Cloud needs to be more used, and the data-centric approach needs to facilitate user-initiated interactions with actuators.

2.3 Cloud Computing

This section delves into the complexities of Cloud and IoT integration within the Compute Continuum. It introduces the Cloud computing paradigm, its various services, and implementations. Then, it presents a perspective that combines IoT deeply within the Cloud infrastructure, enabling users to share IoT resources by virtualizing the nodes hosting these resources (e.g., sensors and actuators).

2.3.1 Definition

The Cloud computing paradigm significantly expands computing, storage, and networking capabilities for Cloud-based applications. According to the National Institute of Standards and Technology (NIST), Cloud computing is defined as a model that enables ubiquitous, convenient, and on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider involvement [17]. This flexibility supports the pay-as-you-go billing model [62], which has played a crucial role in the widespread adoption of Cloud services. By offering convenient access to remote resources and data management services, Cloud computing enables users to be charged only for the resources they consume. Main IT companies such as Google, IBM, Microsoft, and Amazon operate large data centers to host user applications and services, solidifying the importance of Cloud computing within the Compute Continuum.

2.3.2 Cloud Services

The Cloud offers various services to cater to the various needs of application developers. The three most recognized Cloud offerings are Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). Cloud users can select a service based on the level of control they require over the infrastructure.

In the IaaS model, users have low-level access to IT infrastructure, including processing, storage, and networking resources [25]. IaaS users can configure their instances, often as standalone virtual machines (VMs), regarding hardware and software. Specifically, control over the instance allows Cloud consumers to customize hardware configurations, such as the number of CPU cores, RAM capacity, and storage capacity. Additionally, users are responsible for managing the system-level software [65].

In the PaaS model, users have less control over the infrastructure compared to IaaS. PaaS users delegate the management of hardware and software infrastructure to the Cloud provider, who delivers hardware and ready-to-use software tools over the Internet, typically needed for application development [4]. The PaaS model lets users focus on the application's business logic without worrying about software or hardware configurations.

In the SaaS model, the Cloud provider offers an entire application stack, which runs entirely in the provider's data center. Users delegate all configuration and management tasks to the provider. Using the SaaS model, users only need to log in and use the service through a specific application (e.g., a web browser). Examples of SaaS include Gmail, Dropbox, and Microsoft 365.

On Premises	IaaS	PaaS	SaaS
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
O/S	O/S	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Managed by the user Managed by the Cloud provider

Table 2.1: Management responsibilities across different Cloud models: On-Premises, IaaS, PaaS, and SaaS.

Table 2.1 illustrates the three Cloud services and their relationship with the underlying infrastructure. It also highlights the infrastructure management responsibilities in the three Cloud offerings (IaaS, PaaS, and SaaS).

2.3.3 Resource Provisioning in the Compute Continuum

Resource provisioning is a key feature of the Compute Continuum. Given the unpredictable and changing demands of Cloud, Edge, and IoT environments, static resource allocation can lead to performance issues due to either over-provisioning or under-provisioning [58]. The Compute Continuum's core concept is based on flexibly provisioning resources according to demand. To optimize resource usage across the continuum, providers use virtualization technologies and efficient provisioning systems to manage the hardware and software configurations of their data centers, edge nodes, and IoT devices. Additionally, since estimating the usage of applications and services hosted within the continuum is challenging, providers adopt the pay-as-you-go billing model [30] alongside demand-driven resource provisioning.

2.3.4 Types of Cloud Models

Cloud deployments within the Compute Continuum can be categorized into four types: private Clouds, community Clouds, public Clouds, and hybrid Clouds [17].

2.3.4.1 Private Cloud

A private Cloud provides a dedicated proprietary environment for a single business entity. Like other Cloud computing environments, it offers extensive virtualized computing resources through

physical components stored on-premises or in a vendor's data center. One of the primary benefits of deploying a private Cloud is the high degree of control it offers to the organization and the enhanced privacy it ensures.

2.3.4.2 Community Cloud

The Community Cloud, which is less typically used, involves sharing a Cloud data center among several organizations with similar security and confidentiality requirements. It can be compared to a shared private Cloud.

2.3.4.3 Public Cloud

A public Cloud is a service accessible to everyone via the Internet. This service can consist of provisioning resources, such as storage (e.g., Dropbox), computing power (e.g., Amazon EC2), or even applications (e.g., Customer Relationship Management tools). Public Cloud providers benefit from enormous storage and computing capacities, allowing them to serve all users simultaneously. However, the services public Cloud platforms provide may only partially meet all user needs due to limited adaptability. Moreover, using a public Cloud can be economical as no upfront investment is required, and users only pay for their consumption.

2.3.4.4 Hybrid Cloud

The hybrid Cloud combines private and public Clouds. Given companies' continuously evolving and increasingly complex information systems needs, the hybrid Cloud enables the distribution of resources and a precise definition of each Cloud's roles within the overall information system process. A hybrid Cloud allows users to benefit from the security of a private Cloud for storing sensitive data while leveraging the flexibility and scalability of a public Cloud for dynamic resource allocation.

2.4 IoT Cloudification in the Compute Continuum

2.4.1 Motivation

The increasing interest in the IoT arises from the widespread availability of devices with sensing and actuating capabilities that serve as programmable gateways to the physical world. Generally, most approaches to fully harness the potential of the IoT ecosystem rely on adopting the Cloud paradigm. However, as previously discussed, many of these solutions fall into the category of data-centric solutions [51, 56, 47], where the only allowed operations involve data manipulation (see Figure 2.1(a)). In this management design, IoT devices are often considered simple data sources or, at most, non-reconfigurable bidirectional remote interfaces [10].

To gain complete control over an IoT infrastructure and enable reprogramming capabilities, users may choose vertical solutions to deploy and manage their infrastructure (see Figure 2.1(b)). However, such solutions do not allow application developers to share IoT infrastructure, necessitating that each user sets up their infrastructure. This limitation restricts the widespread adoption of

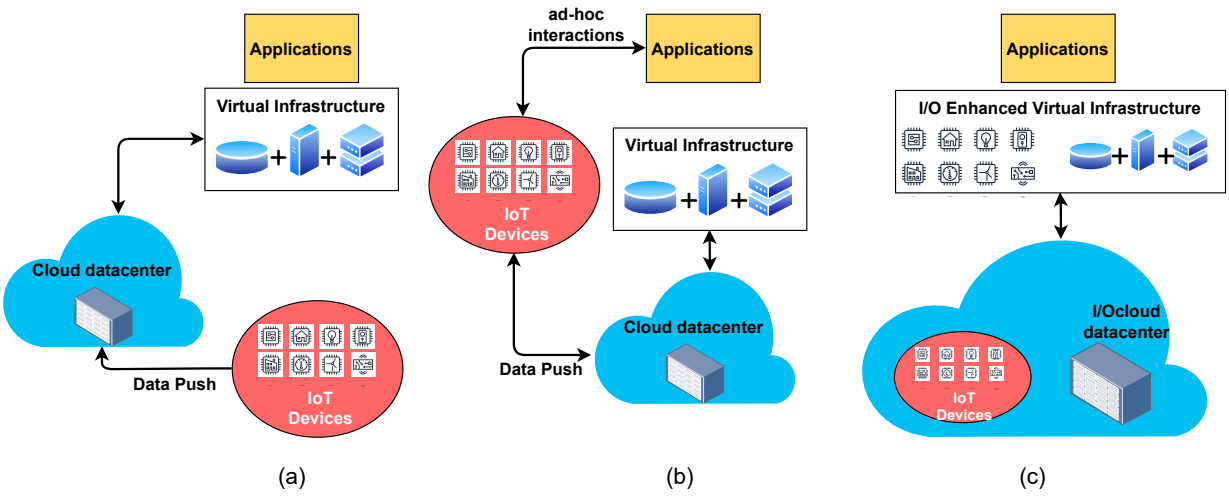


Figure 2.1: Cloud-based IoT architectures: (a) IoT devices as a data source for the Cloud, (b) IoT as a remote interface to Cloud-based applications, and (c) IoT as an extension of Cloud resources.

IoT applications, as the financial fee required for IoT infrastructure is often substantial. Additionally, obtaining authorization to deploy IoT nodes in public domains for large-scale deployments can be challenging. Beyond the limitation of sharing IoT infrastructure, data-centric solutions involve transmitting all generated data to a data center. For instance, an IoT sensing deployment with numerous sensors producing data at high rates can incur significant operational costs in terms of bandwidth, storage, and processing [32]. In such scenarios, processing the data at the edge and only transmitting preprocessed information to the Cloud may be beneficial, thereby reducing bandwidth and storage usage. Furthermore, edge data processing is essential for time-sensitive applications that cannot accept delays introduced by relying on a distant Cloud.

Challenging the conventional view of the relationship between the Cloud and IoT is now possible. In the context of enabling multi-tenant IoT infrastructure within the Compute Continuum, this approach aims to extend the Cloud paradigm by adapting Cloud-enabled analogies to the IoT infrastructure. This adaptation involves viewing IoT as a natural extension of the data center, as illustrated in Figure 2.1(c). Doing so makes it possible to pool a diverse range of geographically distributed devices as infrastructure resources alongside standard Cloud facilities such as computing, storage, and networking capabilities.

2.4.2 The Device-Centric Approach

Creating a public, multi-tenant IoT infrastructure that integrates edge computing when needed and functions as an extension of cloud deployments requires addressing the challenges intrinsic in IoT/Cloud integration. A promising solution lies in adopting a device-centric approach, where computation is carried closer to the data, reducing communication overhead. This approach is particularly critical when managing large volumes of incoming data in sensing-related tasks. Unlike the traditional data-centric model, which is heavily reliant on Cloud-based systems, the device-centric paradigm focuses on providing users with actual sensing and actuation resources at the network edge, even if virtual, rather than merely delivering the data generated [26].

While the data-centric approach can be effective in particular scenarios, the device-centric approach offers several advantages [26]:

- **Decentralized control:** Distributed policies can be established on sensors and actuators through customization features, allowing users to deploy personalized software on sensing/actuation entities.
- **Onboard data prefiltering and processing:** By employing edge computing, data generated by sensors and actuators can be filtered and/or preprocessed directly on IoT devices, thereby reducing latency in decision-making while improving user privacy.
- **Reduced data transfers:** Enabling edge computing on IoT devices allows for direct communication between users and sensing/actuation devices, requiring only a single data transfer. In contrast, data-centric approaches require at least two transfers, as data is stored in a database and then made available to users.
- **Composition and repurposing:** Users can implement customized logic on IoT nodes, enabling them to aggregate, organize, and/or repurpose sensing resources.
- **Enhanced security:** The device-centric approach enhances security and privacy in IoT by moving processing tasks between the Cloud and devices as needed, depending on the required level of security and device capabilities.
- **Information dissemination:** Data is transmitted through the distributed sensing infrastructure, allowing the implementation of distributed data delivery algorithms to optimize data transfer.

To implement the device-centric view in conjunction with multi-tenancy capabilities in IoT, a set of functionalities in the areas of sensor and actuator virtualization is required. These functionalities should make virtual sensing resources available as endpoints, enabling registration, enumeration, and interaction.

The following sections introduce an overview of the I/Ocloud approach [10, 5], which aims to provide standardized and generic programming capabilities on top of IoT resources, regardless of the underlying infrastructure configurations. Additionally, the approach leverages the unique characteristics of an IoT-enhanced distributed data center, such as the availability of edge nodes, which can be used as computing infrastructure for data preprocessing.

2.4.3 I/Ocloud: A Multi-Tenant IoT Solution

2.4.3.1 Types of IoT Nodes

An IoT resource is a connected entity (e.g., a sensor or actuator) that can be exported and connected without necessarily being programmable. Examples of IoT resources include sensors connected to General-Purpose Input/Output (GPIO) pins of a Single-Board Computer (SBC), accelerometers in smartphones, optical heart rate sensors in smartwatches, and wireless sensor nodes.

Thus, an IoT node can be defined as any computing entity capable of hosting physical IoT resources (e.g., sensors and actuators) while running user-defined logic. Typically, such nodes are

commercialized with limited computing and storage capabilities. Moreover, they are often deployed at the network edge and operate behind networking middleboxes, such as Network Address Translators (NATs) and/or firewalls.

2.4.3.2 Virtual IoT Entities

The I/Ocloud approach aims to achieve seamless integration between the Cloud and IoT by providing distributed IoT resources (i.e., sensors and actuators) hosted on edge-deployed nodes as virtualized Cloud resources. This approach can be viewed as an extension of the Cloud data center, referred to as I/Ocloud [5]. A Cloud user can interact with remote IoT resources as if they were Cloud-based, without depending on ad-hoc or application-level Application Programming Interfaces (APIs). One of the critical functions of I/Ocloud is ensuring that IoT deployments are engaged as active elements of the Cloud infrastructure while preserving their characteristics (e.g., sensing capability). The core concept of Cloud/IoT integration redefines virtualization to include IoT nodes' hosted resources, referred to as I/O virtualization (virtIO).

The I/Ocloud approach extends the concept of virtualization to the IoT world by abstracting IoT resources and providing them as virtual resources. An I/O resource can be considered an instance of a developer-friendly interface for the I/O primitives of its physical counterpart. This abstraction can contain all I/O resources of an IoT node or only a subset. Additionally, I/O virtualization can logically group IoT resources from different nodes into a single logical entity.

The I/Ocloud abstraction approach employs file system virtualization to grant low-level access to IoT resources. This choice is intentional, as many modern IoT nodes, such as Raspberry Pi and Arduino, employ the GPIO pseudo file system to communicate with the physical pins of the boards. This I/O virtualization can also expand to node virtualization in IoT. The I/Ocloud instance functions as a virtual representation of a physical IoT node, complete with its physical pins. This virtual representation can host user-defined logic and interact with remote physical IoT resources. In technical terms, an I/Ocloud instance is an isolated environment with a user-defined file system, representing a clone of the remote physical IoT resources within the file system hierarchy of the virtual IoT node.

The virtualization approach involves a layered architecture where, at the foundational level, sensors and actuators are associated with physical IoT devices. Above the Linux-based Operating System (OS) on these IoT nodes, a pseudo file system (sysfs) interface facilitates I/O operations between the physical pins and the system. The I/O hypervisor plays an essential role in this process by exposing a virtualized sysfs (i.e., the /sys filesystem) that is user-space-defined for each virtual IoT node created. This sysfs virtualization effectively promotes the availability of I/O operations to the corresponding virtual IoT node.

Additionally, the I/Ocloud virtualization approach can be deployed within the Cloud by making remote transducers accessible on virtual IoT nodes hosted in the Cloud data center. Through inter-hypervisor communication, sensors on the physical IoT node are made available to the virtual IoT instance hosted on the Cloud computing node. In this design, the user-space file system leverages Filesystem in User-space (FUSE) [38] over Remote Procedure Calls (RPCs) to ensure that remote interactions with physical IoT resources are executed efficiently.

When considering typical Cloud facilities, the I/Ocloud approach can be deployed using either plain VMs or Virtual Nodes (VNs). The difference between these instances lies in the instance flavor, as in standard Cloud deployments. Once a VM or VN is instantiated, virtualized I/O

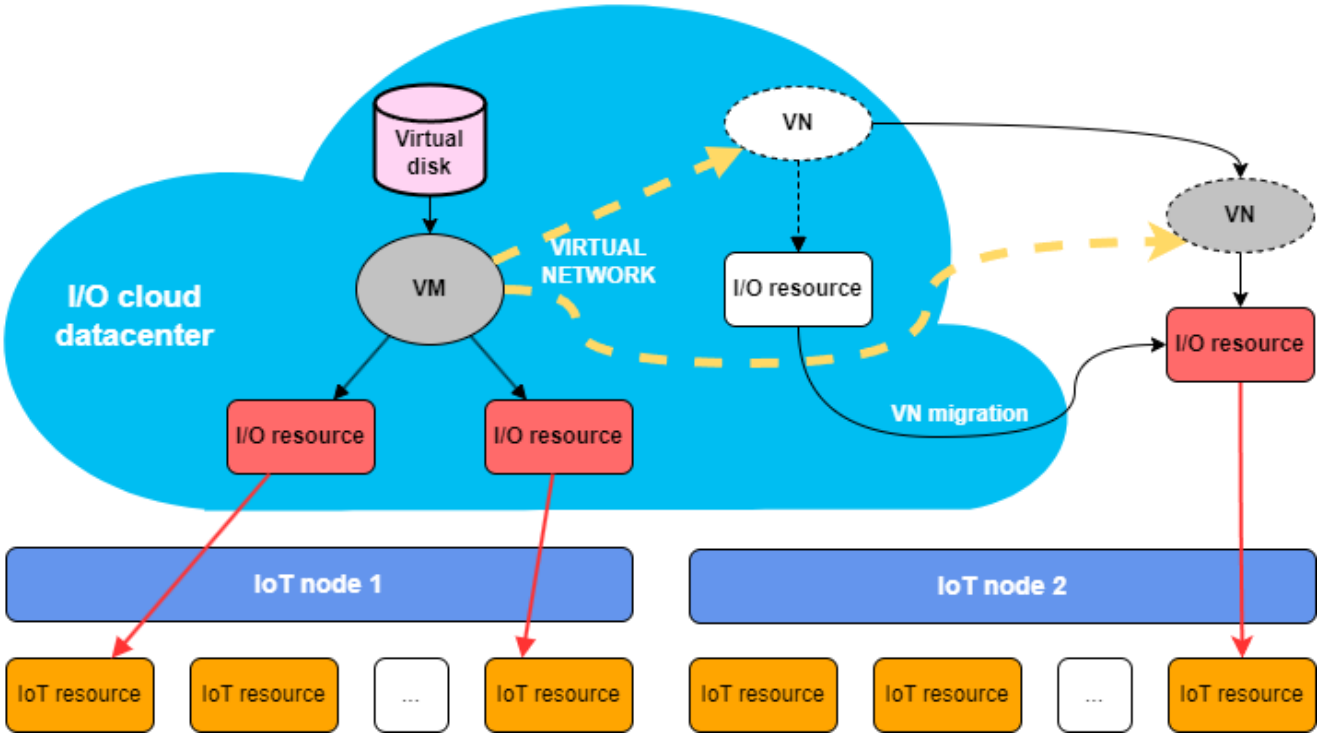


Figure 2.2: I/Ocloud instances with attached I/O resources.

resources can be attached to the instance as though they were physically connected, regardless of the configuration of the physical nodes hosting them. Consequently, an IoT application developer can use their VM or VN as if it were a physical IoT node with sensors and actuators attached to its physical pins (see Figure 2.2).

2.4.3.3 I/Ocloud Virtualization at the Network Edge

The I/Ocloud virtualization at the edge becomes apparent when recognizing that I/O resources serve as interfaces to the physical world, while distributed IoT nodes hosting these resources can provide a pool of distributed computing nodes at the network edge.

Shifting computation to the network's periphery involves decentralizing business logic, allowing it to be scheduled on IoT nodes when possible. This approach often requires using the Cloud primarily to facilitate communication at levels below the application, mainly when network constraints limit direct interaction between nodes.

Regarding VN management, a VN may be instantiated as an isolated and portable environment (e.g., a lightweight container) either in the data center or on a remote IoT node. Initially, a VN may be deployed on the I/Ocloud data center and then migrated, as needed, to other infrastructure (e.g., to the edge) to meet specific requirements such as latency constraints (see Figure 2.2). Conversely, a VN may be instantiated on an edge-based physical IoT node and offloaded to the Cloud if more computing resources are required. In addition to virtualizing IoT nodes and their physical resources, the I/Ocloud approach extends to network virtualization. This is crucial to fully integrating IoT with the Cloud and overcoming networking obstacles in IoT deployments. The

I/Ocloud framework enables users to instantiate personalized networking topologies among any combination of VMs and VNs, spanning both the data center and Wide Area Networks (WANs) when VNs are deployed at the network edge (see yellow dashed lines in Figure 2.2). The networking solution should also support bare-metal IoT nodes, particularly when users own IoT devices and wish to include them in their deployment.

Based on the discussions surrounding the I/Ocloud approach, several significant benefits emerge, including:

- **Decoupling IoT infrastructure from underlying networking configurations:** This approach splits the IoT infrastructure from the complexities of network management, allowing users to focus on their business logic.
- **Enhancing the concept of IoT infrastructure as Code (IaC):** The approach supports the Infrastructure as Code (IaC) model, enabling developers to manage IoT infrastructure programmatically.
- **Enabling edge computing to meet specific application requirements:** Edge computing is vital for applications that require low-latency processing and quick response times.
- **Providing low-level abstraction of IoT nodes and resources:** This abstraction is essential for ensuring application code portability across different IoT deployments.
- **Enabling high-granularity interaction with IoT resources:** Thanks to the pseudo file system, users can interact with IoT resources at a detailed level.
- **Overcoming networking barriers in IoT deployments:** The approach provides solutions to networking challenges that often emerge in large-scale IoT deployments.

2.5 Enabling Technologies

2.5.1 OpenStack

OpenStack is an open-source platform consisting of software tools designed for building and managing Cloud computing infrastructures. It serves as a cornerstone for infrastructure Cloud solutions in most commercial, in-house, and hybrid deployments and is a fully open-source ecosystem of tools and frameworks. Currently, OpenStack enables the management of virtualized computing and storage resources, attaching to the infrastructure Cloud paradigm.

Figure 2.3 illustrates the conceptual architecture of OpenStack, depicting its components as boxes and the services they provide to other components through arrows. Nova, the compute resource management subsystem, lies at the core of OpenStack and is responsible for provisioning VMs, supported by various subsystems that offer both core (e.g., networking via Neutron) and optional services (e.g., block storage via Cinder). Horizon acts as the dashboard, providing a User Interface (UI) through a web-based platform or a command-line interface for Cloud end users. The metering and billing subsystem, Ceilometer, is closely integrated with Nova and other middleware components, functioning within the OpenStack ecosystem. As the central component, Nova dictates the hierarchy among participating devices, including their roles and interaction

policies. Moreover, Nova requires a Cloud controller, a machine that centrally manages one or more compute nodes, typically providing component-specific services (e.g., computing) by leveraging a hypervisor for resource-sharing and workload-multiplexing.

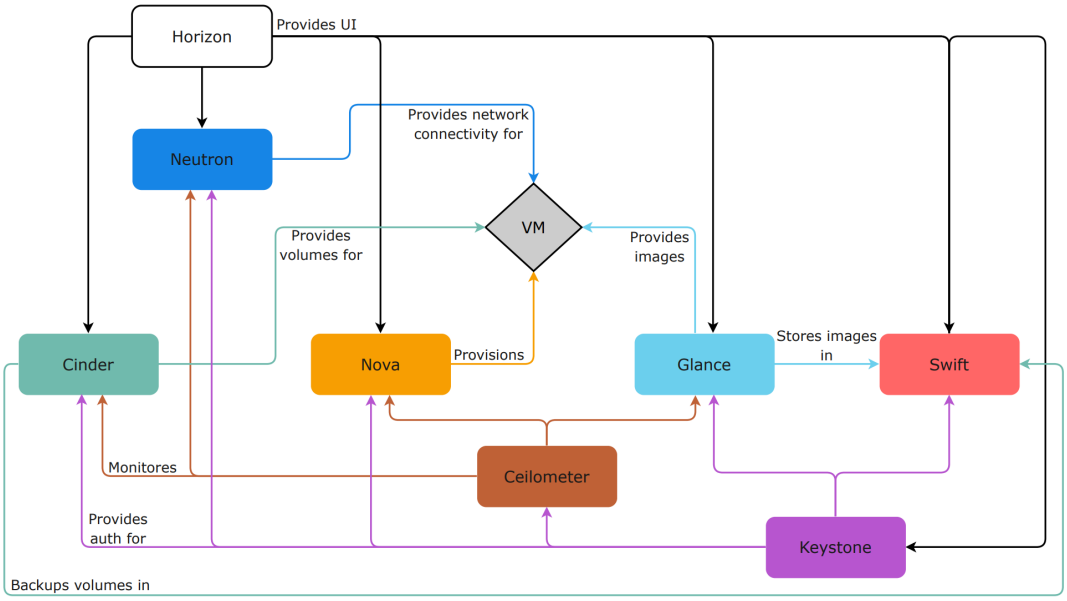


Figure 2.3: OpenStack architecture.

2.5.2 Stack4Things

Built on the OpenStack open-source project to realize the I/Ocloud concept discussed earlier, the Stack4Things (S4T) middleware represents a first step to extend the OpenStack ecosystem to support the management of IoT deployments [38]. The project employs an implementation-driven approach to incorporate capabilities that enable IoT infrastructure to join an edge-extended IaaS Cloud. The middleware offers infrastructure-enabling facilities to manage instances at the network edge.

As shown in Figure 2.4, the same conceptual architecture used in Figure 2.3 illustrates the subsystems that comprise the S4T middleware, with a focus on core components. A new subsystem, IoTronic, is introduced to provision and configure IoT nodes that host sensing and actuation resources. A diamond-shaped box in place of a VM represents an IoT node hosting transducers, with corresponding interactions described as text along the arrows. Within the S4T architecture, Neutron has been enriched to provide network connectivity for both IoT nodes deployed at the network edge and virtual IoT nodes instantiated using the Zun subsystem. Additionally, to expose edge-based IoT node resources (virtual or physical) as Web resources, the OpenStack Designate subsystem associates publicly resolvable domain names with distributed IoT nodes, even when deployed within IPv4 masquerade networks.

Furthermore, the capabilities of the I/Ocloud framework are extended to enable developers to use Serverless-like interactions as interfaces to the nodes' hosted resources (e.g., sensors and actuators) in specific situations.

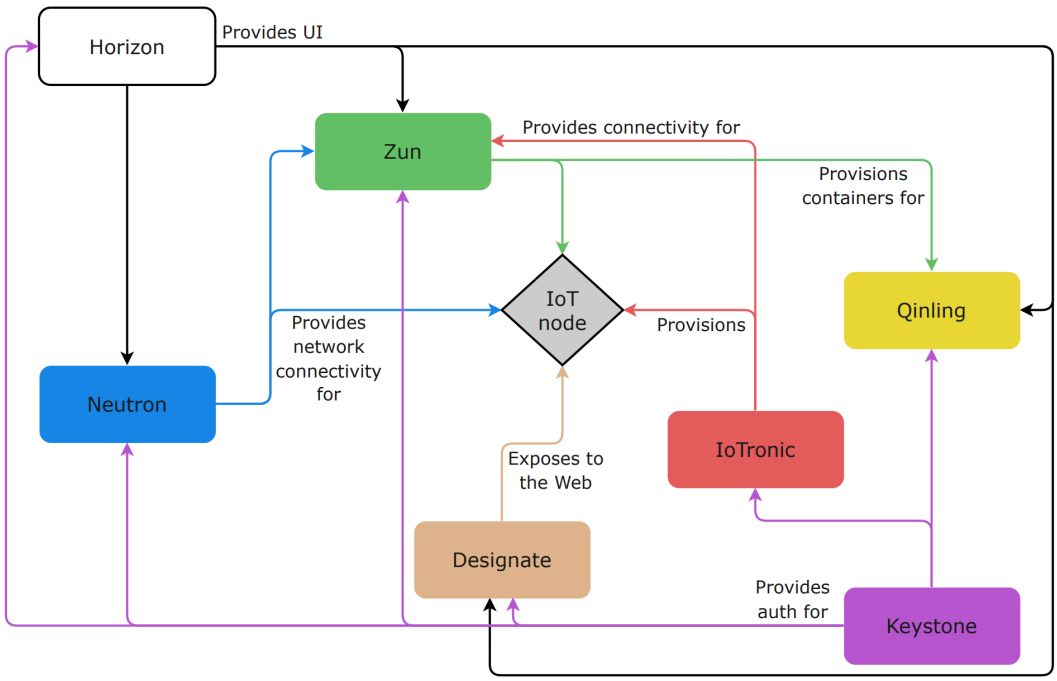


Figure 2.4: Stack4Things subsystems.

Figure 2.5 provides a high-level technical overview of the S4T deployment, distributed between a data center and numerous edge IoT nodes. For the hardware setup of the managed nodes, relatively smart (embedded) devices capable of hosting a minimal Linux distribution (e.g., OpenWRT) were intentionally chosen, such as Single-Board Computers (SBCs) like Arduino, Raspberry Pi, and Arancino, all of which are powered by microprocessors (MPUs). This hardware configuration allows the IoT nodes to host Linux-based tools and runtime environments, particularly Python and Node.js, which the S4T node agent Lightning-Rod (LR) requires. This agent bridges the remote IoT nodes to the Cloud infrastructure where the S4T IoTronic service is deployed. IoTronic follows the standard design of OpenStack services, as illustrated in Figure 2.6, which depicts the Cloud-side architecture of S4T.

The interconnection between IoTronic and LR is established via a full-duplex message channel that forwards commands from the Cloud to the IoT nodes. Technically, this interconnection is built using the Web Application Messaging Protocol (WAMP), as shown by the violet arrows in Figures 2.5, 2.6, and 2.7. WAMP is an open standard WebSocket subprotocol designed to provide publish/subscribe (pub/sub) as well as Remote Procedure Call (RPC) patterns. IoTronic enables services forwarding through the Cloud (green arrows in Figures 2.5, 2.6, and 2.7). Specifically, users and administrators can access services (e.g., SSH) on remote devices, regardless of their physical networking configuration. The Cloud-side component of this forwarding mechanism is the S4T IoTronic WebSocket (WS) tunnel agent, which acts as a "wrapper" in control of the WS server to which devices connect using the S4T wstunnel libraries (see Figure 2.7). This reverse tunneling mechanism uses WebSocket to provide service forwarding [27].

At the core of the IoTronic subsystem is the conductor, which manages the local database that stores metadata about the nodes (see Figure 2.6). The S4T API server exposes a set of RESTful (Representational State Transfer) APIs facilitating various interactions with remote IoT nodes

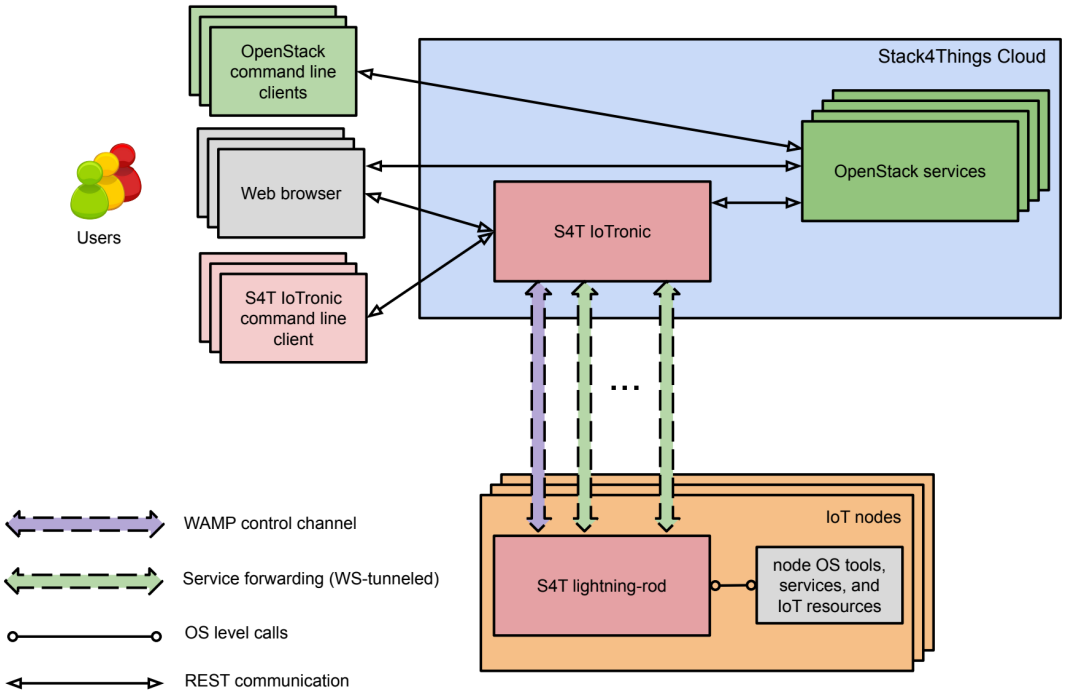


Figure 2.5: Stack4Things architecture overview.

through the IoTronic Command-Line Interface (CLI) or a Web-based Graphical User Interface (GUI). For this purpose, the OpenStack Horizon dashboard has been customized with an S4T panel that exposes the services provided by IoTronic.

2.5.3 I/Ocloud Use Case: Software-Defined Cities

A Smart City [59] represents an ecosystem of infrastructure and services that aims to connect society, government, and technology to enhance services such as smart mobility and smart environments. This holistic perspective necessitates a comprehensive approach to integrating technologies and services, providing broad or global solutions to urban challenges. In this context, a scalable architecture is required to reuse, multiplex, and share technologies and services across an urban scale. The objective is to establish a homogeneous ecosystem where multiple applications can scale to a metropolitan dimension, thereby supporting an open and shared Information and Communication Technologies (ICT) infrastructure composed of sensing, actuation, network, processing, and storage resources.

Managing heterogeneous and complex socio-technical systems on the scale of entire cities requires an overarching approach that addresses all related issues comprehensively. Specifically, the goal is to provide a uniform representation of connected smart objects by abstracting, grouping, and managing them as a unified ecosystem. This ecosystem should be configurable, customizable, and contextualized according to high-level application requirements. Simultaneously, a management layer capable of controlling the ecosystem's dynamics is needed to map these high-level requirements to lower-level ones, implementing and enforcing specific policies to satisfy them.

A suitable solution is to adopt a software-defined approach, where the control plane uses the basic mechanisms provided by smart city objects at the data plane to implement policies related

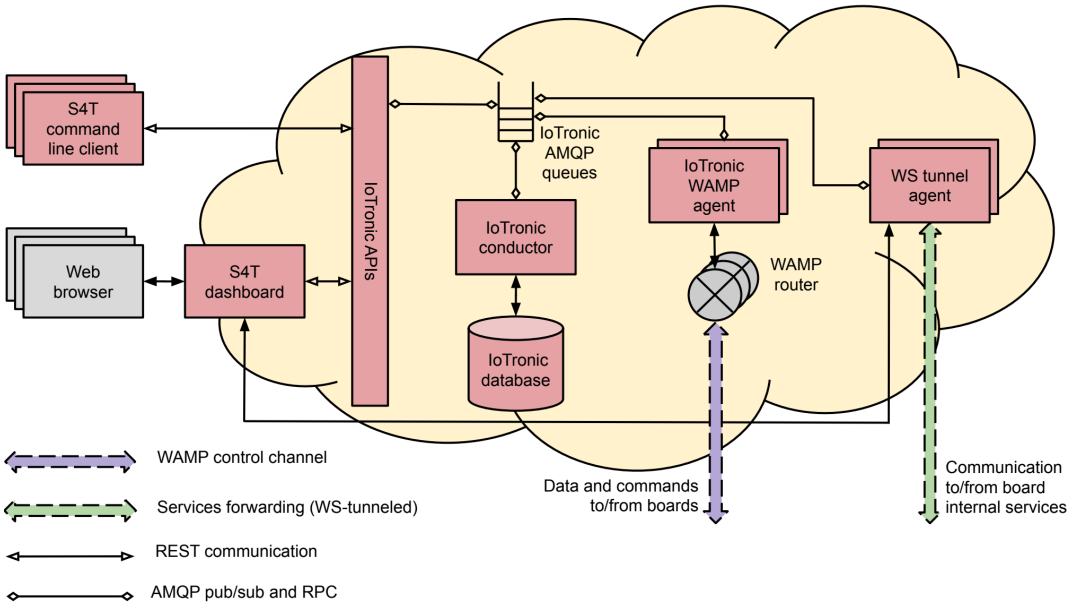


Figure 2.6: Stack4Things Cloud-side architecture.

to application and end-user requirements. This concept leads to the notion of Software-Defined Cities (SDCs) [43]. In this approach, the data plane comprises geographically distributed IoT devices interacting with the physical environment through sensors and actuators. Conversely, the control plane consists of Cloud-hosted (or edge-based) virtual boards, as illustrated in Figure 2.8.

In the Software-Defined City (SDC) paradigm, the control logic is decoupled from the physical devices, allowing for a more flexible and programmable infrastructure. The virtual boards in the control plane manage the underlying physical systems by implementing specific policies related to applications and end-user requirements. These virtual boards can dynamically configure and control the physical IoT devices distributed throughout the city, creating a cohesive and responsive urban environment.

The SDC approach enables cities to efficiently manage and scale their services by abstracting the physical infrastructure and exposing it through programmable interfaces. This paradigm also promotes the integration of new technologies and applications, ensuring that cities can adapt to evolving needs and challenges. The software-defined approach allows for centralized management and automation of various city services, thereby enhancing the efficiency and effectiveness of urban operations.

Moreover, the SDC concept emphasizes the importance of edge computing in managing large-scale IoT deployments. By processing data closer to the source, edge computing reduces latency and bandwidth consumption, enabling citizens to access real-time services. The combination of edge computing and centralized control in the SDC framework ensures that cities can meet the demands of modern applications while maintaining a high level of responsiveness and adaptability.

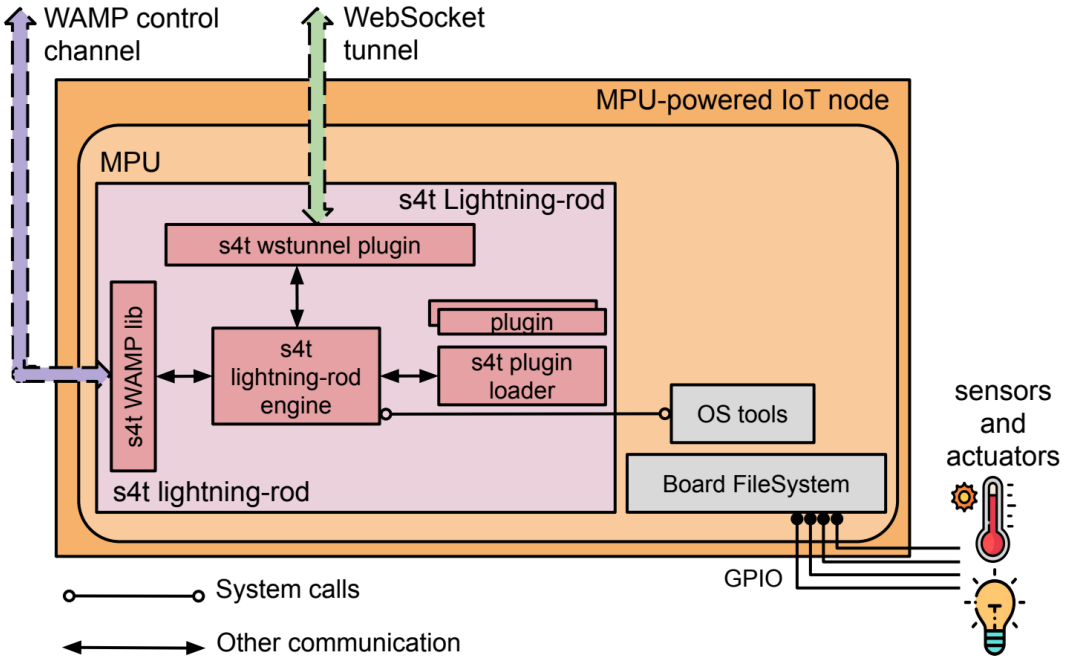


Figure 2.7: Lightning-Rod architecture.

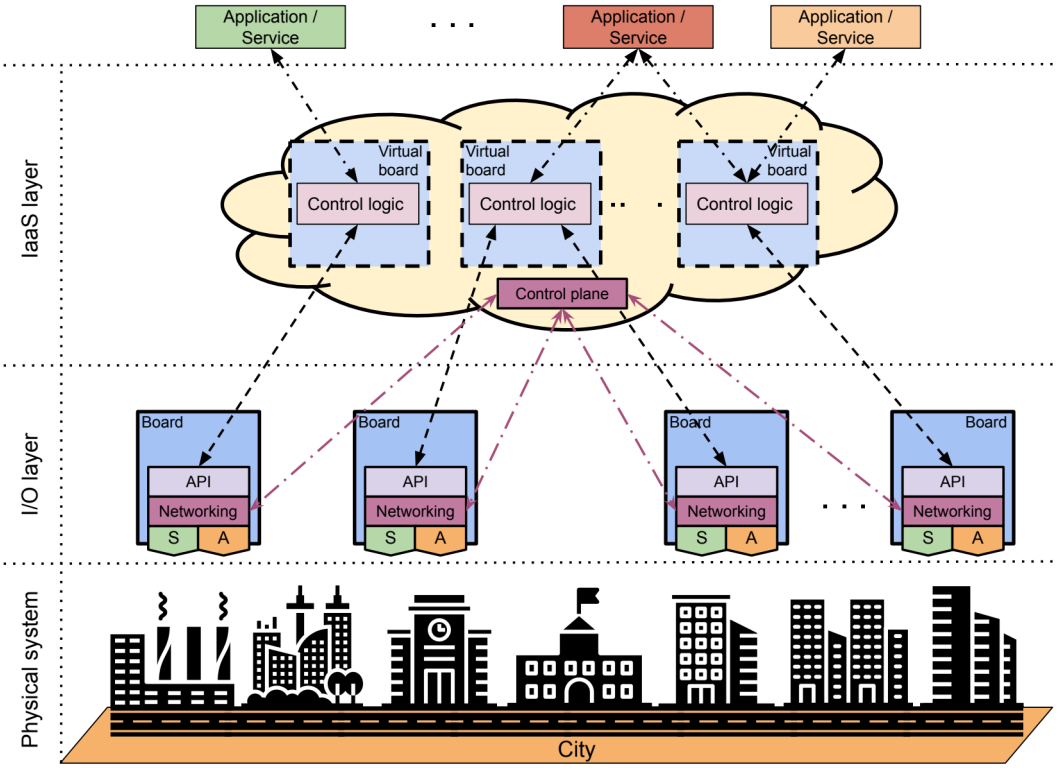


Figure 2.8: The Software-Defined City paradigm.

Chapter 3

Cloud-Edge Architectures: Foundations and Applications

The rapid evolution of cloud-edge computing has significantly impacted various domains, particularly in managing large-scale IoT systems and advancing bioengineering applications. Horizontally scalable architectures, characterized by their ability to distribute workloads across multiple nodes and dynamically adjust to increasing demands, are critical to supporting these advancements. This chapter connects the foundational concepts introduced in Chapter 2 with the applied research explored in Chapter 4, offering a detailed examination of the technologies and strategies that underpin scalable cloud-edge systems.

Additionally, this chapter aims to establish a solid foundation for understanding how these architectures enable the integration of IoT and bioengineering applications, setting the stage for the detailed case studies presented in the next chapter. By exploring critical technologies such as LoRaWAN, Peer-to-Peer (P2P) networking, edge computing, and I/Ocloud, the chapter will highlight their contributions to creating scalable, resilient, and efficient systems capable of meeting the complex needs of modern applications.

3.1 The Role of Horizontally Scalable Architectures in IoT Systems

3.1.1 Challenges in Scaling IoT Architectures

The exponential growth of IoT devices presents significant challenges to traditional cloud-centric architectures, which often struggle to manage the high volume and diversity of data generated by these devices. In typical IoT deployments, data from sensors and devices is transmitted to centralized cloud servers for processing and analysis. However, as the number of connected devices increases, this approach can result in bottlenecks, increased latency, and higher operational costs due to extensive data transmission and centralized processing power.

Horizontally scalable architectures address these challenges by enabling distributed processing across multiple edge nodes, reducing the load on centralized servers, and enhancing overall system efficiency. This is particularly advantageous in scenarios where real-time processing and low-latency responses are essential, such as in smart cities, healthcare, and bioengineering systems.

3.1.2 Horizontal Scalability in Smart Cities

Smart cities, which rely on complex networks of interconnected devices to monitor and manage urban services, exemplify the need for horizontally scalable architectures. These cities leverage vast networks to manage transportation, energy, public safety, and healthcare systems. Horizontally scalable designs allow these systems to adapt to the continuous growth of urban populations and the increasing demand for smarter, more responsive services.

For example, in a smart city, sensors generate real-time data on traffic patterns, environmental conditions, and energy usage. A horizontally scalable architecture enables this data to be processed locally at the edge, allowing fast decisions to optimize traffic flow, reduce energy consumption, or respond to emergencies. The cloud provides resources for long-term data storage, advanced analytics, and coordination between various city services.

Chapter 4 will present case studies demonstrating how horizontally scalable architectures are implemented in smart cities, focusing on integrating IoT devices, edge computing, and cloud resources to create robust, adaptive urban infrastructures.

3.2 Enabling Technologies for Scalable Architectures

3.2.1 I/Ocloud: Virtualized IoT Resource Management

I/Ocloud is a paradigm that enhances cloud-edge architectures' scalability, flexibility, and efficiency, particularly in large-scale IoT systems. I/Ocloud enables seamless integration between cloud and edge computing environments by virtualizing IoT resources, such as sensors and actuators, into dynamic shareable entities. This approach is essential for building horizontally scalable systems that manage complex, data-intensive applications in smart cities, healthcare, and bioengineering.

A key feature of I/Ocloud is *IoT resource virtualization*, which abstracts physical IoT devices into virtual resources that can be accessed, reconfigured, and shared across multiple users and applications. Traditional IoT deployments often require dedicated devices for specific applications, leading to inefficient resource utilization and higher costs. I/Ocloud addresses these limitations by creating a virtual layer where IoT resources can be managed as cloud-like entities, enabling efficient resource sharing and reducing operational overhead. This virtualization also allows IoT resources to be provisioned dynamically, ensuring that system resources are allocated according to real-time demand.

Additionally, I/Ocloud integrates *edge computing* to reduce latency and enhance real-time data processing. In cloud-centric models, data from IoT devices is typically transmitted to centralized cloud servers for processing, leading to significant delays. I/Ocloud addresses this challenge by processing data at the network edge, closer to the source, thereby improving system responsiveness. This capability is vital in smart cities and healthcare, where real-time decisions, such as adjusting traffic flows or monitoring patient health, are critical. I/Ocloud supports a flexible and scalable architecture by distributing computational tasks between cloud and edge nodes.

3.2.2 LoRaWAN: Enabling Mobility in IoT Monitoring

LoRaWAN (Long Range Wide Area Network) is another technology that supports scalable architectures, particularly in IoT networks requiring extensive geographical coverage. LoRaWAN's

long-range, low-power characteristics make it ideal for connecting many mobile devices, ensuring continuous data collection and monitoring even as devices move between locations.

In healthcare, LoRaWAN could be critical for remote patient monitoring, particularly for individuals with chronic conditions or those recovering from surgery. Its low power consumption allows wearable health devices to operate for extended periods without frequent charging. By decentralizing data collection and enabling local processing at the edge, LoRaWAN enhances the scalability and resilience of IoT monitoring infrastructures.

Chapter 4 will explore specific applications of LoRaWAN, principally in mobility, focusing on its role in supporting scalable and efficient monitoring systems.

3.2.3 Peer-to-Peer Networking: Enabling Decentralized Scalability

P2P networking is another crucial technology that enhances the scalability and resilience of cloud-edge architectures by enabling decentralized data processing and communication. Unlike traditional client-server models, where data passes through centralized servers, P2P networks allow devices to communicate directly, reducing latency and improving fault tolerance.

P2P networking is particularly valuable in scenarios requiring decentralized control, such as emergency response situations where centralized infrastructure may be compromised. In healthcare, P2P networking enables the creation of decentralized networks of medical devices and sensors that communicate directly to share data and coordinate responses. This decentralized structure enhances the system's ability to scale horizontally, as new devices can be added without overloading central servers. Moreover, P2P networking improves resilience by ensuring systems can continue functioning even when network parts are disrupted.

The next chapter will examine how P2P networking has been studied and implemented in IoT device communication scenarios, where decentralized control and fault tolerance are critical.

3.2.4 Edge Computing: Enhancing Real-Time Responsiveness

Edge computing is a foundational component of scalable architectures, providing the computational power to process data at or near its source. By offloading data processing tasks from centralized cloud servers to edge nodes, edge computing reduces latency and improves real-time responsiveness, which is critical in applications such as autonomous vehicles, industrial automation, and telemedicine.

Edge computing complements cloud computing by enabling the distribution of computational tasks across multiple nodes, ensuring system scalability even as data volumes increase. This hybrid approach leverages the strengths of both cloud and edge computing, offering a flexible and scalable solution for processing large amounts of data in real-time.

3.3 Building Towards Practical Applications

The technologies and architectures discussed in this chapter provide a basis for developing advanced applications that can be easily employed in the bioengineering domain. Cloud-edge systems are well-suited for managing the complex, data-intensive tasks associated with bioengineering, such as real-time motion capture, remote health monitoring, and personalized medicine.

3.3.1 Real-Time Data Processing in Bioengineering

Bioengineering applications, such as clinical rehabilitation and sports science, often involve collecting and analyzing large amounts of data in real-time. Motion capture systems, for instance, generate detailed data on patient movements, which must be processed quickly to provide immediate feedback to both patients and clinicians. Horizontally scalable architectures enable these systems to operate efficiently, even as the number of users or the complexity of data increases.

Edge computing plays a crucial role in these applications by processing data locally, reducing the latency involved in transmitting data to the cloud. This real-time feedback is essential in scenarios such as clinical rehabilitation, where immediate adjustments to patient treatment plans can lead to better outcomes.

Chapter 5 will explore specific motion capture studies and their applications.

3.3.2 Telemedicine and Remote Health Monitoring

As mentioned previously, an area where horizontally scalable architectures have significant potential is in telemedicine and remote health monitoring. As healthcare systems increasingly rely on remote monitoring and telemedicine to manage patient care, the ability to scale these platforms becomes essential. This is particularly true when many patients must be monitored simultaneously, or the system must adapt to changing patient needs in real-time.

Edge computing and decentralized data processing enable telemedicine platforms to deliver more responsive and personalized care, ensuring that critical health data is processed quickly and interventions can be made promptly. Additionally, the integration of LoRaWAN and P2P networking enhances the scalability and resilience of these platforms, ensuring they operate effectively in resource-constrained environments.

3.4 Stacking Technologies for Horizontally Scalable Architectures

The enabling technologies discussed in this chapter (LoRaWAN, P2P networking, edge computing, and I/Ocloud) offer complementary capabilities that contribute to the scalability, resilience, and efficiency of scalable cloud-edge systems.

- *LoRaWAN*, with its long-range, low-power communication, is particularly well-suited for mobility-focused applications where continuous monitoring over large geographical areas is necessary.

- *P2P networking* enhances decentralized control, allowing systems to function even when parts of the network are compromised. This makes it ideal for emergency response and healthcare scenarios, where fault tolerance is critical.

- *Edge computing* reduces latency by bringing data processing closer to the source, making it essential for real-time applications such as telemedicine, autonomous vehicles, and industrial automation.

- *I/Ocloud* supports dynamic resource allocation and seamless integration between cloud and edge environments, making it highly effective for managing complex, large-scale IoT deployments in smart cities and healthcare.

Each of these technologies offers distinct strengths, and their combination enables the development of scalable and adaptable systems capable of addressing the demands of modern applications.

3.5 Connecting Foundations to Applications

As explored, horizontally scalable cloud-edge architectures offer significant advantages in taming the complexity and scale of modern IoT and bioengineering applications. By investigating the enabling technologies and strategies that underpin these systems, a solid foundation has been established for understanding their practical applications.

In the next chapter, the focus will shift from theory to practice, with specific case studies illustrating the real-world implementation of these architectures. The role of scalable architectures in real-time urban traffic management, telemedicine, and bioengineering will be explored, demonstrating how the approaches discussed in this chapter are driving innovation across diverse domains.

Chapter 4

Case Studies and Applications

This chapter investigates the application of advanced technologies across various domains, focusing on communication systems, smart cities, energy management, emergency response, and healthcare. Integrating these technologies improves system efficiency and scalability, paving the way for sustainable and resilient infrastructures. The chapter begins by examining the foundational role of communication systems in IoT connectivity, followed by specific case studies that illustrate the practical applications of these innovations.

4.1 Communication Systems and IoT Connectivity

The wide adoption of smart city initiatives and large-scale IoT projects is linked to the network's ability to accommodate various devices with diverse communication requirements. From Personal Area Networks (PANs) to Wide Area Networks (WANs) and the incorporation of sophisticated roaming functionalities, these systems ensure service continuity even amidst device mobility. The subsequent sections delve into specific case studies and research advancements demonstrating these communication technologies' practical applications and benefits.

4.1.1 Examples of PAN, WAN, and Roaming Communication Technologies

Personal Area Networks (PANs) are employed for connecting devices within proximity, typically within a few meters. Technologies such as Bluetooth Low Energy (BLE), Zigbee, and Near Field Communication (NFC) are essential to applications like wearable devices, smart home systems, and on-site diagnostics. BLE, in particular, is noted for its energy efficiency and ability to connect many devices within a confined area, making it crucial for low-power, short-range communication scenarios.

Wide Area Networks (WANs) enable communication over extended geographical distances, essential for IoT applications requiring extensive coverage. Technologies like Long Range Wide Area Network (LoRaWAN) and Sigfox exemplify WANs in the IoT domain. These technologies offer long-range communication with minimal power consumption, which is ideal for environmental monitoring, smart metering, and large-scale agricultural deployments. LoRaWAN, in particular, is widely adopted in smart city implementations for its ability to connect many devices across

expansive areas.

Roaming Technologies: As IoT devices increasingly require mobility, roaming technologies have become vital in maintaining seamless connectivity across different network domains. LoRaWAN, while providing basic roaming capabilities, often faces limitations due to the need for predefined agreements between network servers (NSs). One research area investigated proposes infrastructure-centric, network server-agnostic approaches that leverage smart contracts based on blockchain and third-party gateways to enable dynamic, on-demand roaming. These solutions can improve the flexibility and scalability of IoT deployments in smart cities and other dynamic environments.

4.1.2 LoRaWAN Roaming Solutions Without Network Server Dependencies

The need for scalable and flexible communication technologies becomes crucial as IoT networks expand, especially in smart city environments. LoRaWAN, prominent technology in Low-Power Wide-Area Networks (LPWAN), addresses this need but is often constrained by limitations in roaming capabilities. The proposed research has introduced an infrastructure-centric, network server-agnostic approach to LoRaWAN roaming to overcome these challenges. This approach employs smart contracts and third-party gateways to facilitate dynamic, on-demand roaming without requiring predefined agreements between network servers, thereby significantly enhancing the flexibility and scalability of LoRaWAN in dynamic environments [42].

4.1.2.1 LoRaWAN Roaming Challenges and Infrastructure-Centric Solution

LoRaWAN is a well-established LPWAN standard, widely used in smart cities, agriculture, and Industry 4.0 sectors. Despite its adoption, a significant challenge remains the lack of effective roaming mechanisms that support seamless mobility across different network operators. LoRaWAN specifications, versions 1.0 and 1.1, offer basic roaming features but are limited and often inadequate for real-world, dynamic environments.

Version 1.0 lacks comprehensive roaming support, making it challenging for End-Devices (EDs) to move between different network coverage areas. Although version 1.1 introduces more advanced roaming functionalities and improved security, its deployment has been slowed by compatibility issues and limited certification. As a result, most implementations still rely on version 1.0, which does not fully support backward-compatible roaming [11].

The primary limitation of existing solutions lies in their reliance on predefined agreements between Network Servers (NSs), which imposes operational overhead and restricts flexibility. A scalable and flexible roaming solution is essential in dynamic environments where EDs frequently transition across network domains. Current static agreements do not meet the needs of such dynamic IoT deployments.

To overcome these limitations, [42] propose an infrastructure-centric architecture that decouples Gateway (GW) operation from Network Servers. This approach is based on three key mechanisms:

1. **Decoupling Gateways from Network Servers:** GWs can operate independently of any specific NS, enabling them to forward packets to any NS on demand. This flexibility is crucial for managing network load and handling End Device (ED) mobility in real-time, without relying on static connections.

2. **On-Demand Gateway-to-Network Connections:** Connections between GWs and NSs are dynamically established based on traffic conditions, making the network adaptable to changes in ED location and traffic patterns.

3. **Dynamic Agreement Management via Smart Contracts (SCs):** SCs automate the negotiation and enforcement of Service Level Agreements (SLAs) between network operators, eliminating the need for predefined agreements. This allows for real-time negotiation of roaming terms based on current network conditions.

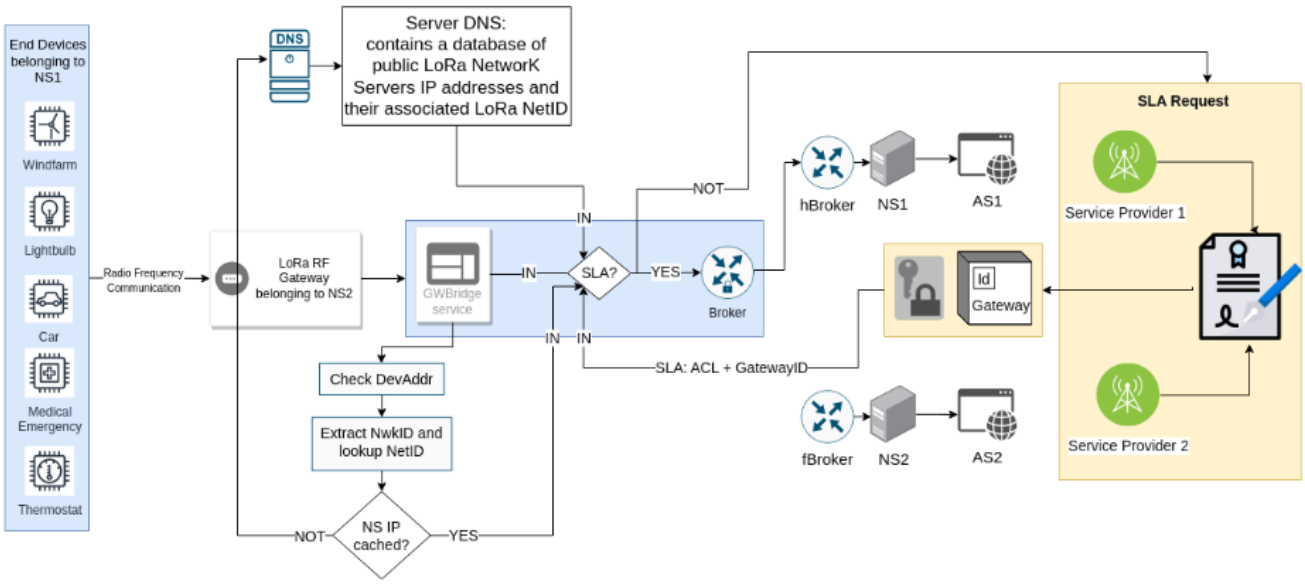


Figure 4.1: NetworkServer-agnostic LoRaWAN Roaming Architecture

This architecture addresses the limitations of current roaming solutions and provides a scalable framework for LoRaWAN deployments, especially in environments with high mobility.

4.1.2.2 Technical Implementation

The proposed architecture illustrated in Figure 4.1 builds upon the concept of passive roaming, which is supported by both LoRaWAN versions 1.0 and 1.1. In traditional passive roaming, an ED remains connected to its home NS (hNS) while transmitting data through a foreign NS (fNS) that controls the GW receiving the ED's packets. The setup and teardown processes of passive roaming are illustrated in Figures 4.2 and 4.3.

While passive roaming has its advantages, it suffers from limitations, such as the need for predefined agreements between NSs and centralized control by the hNS, which reduce network scalability and flexibility.

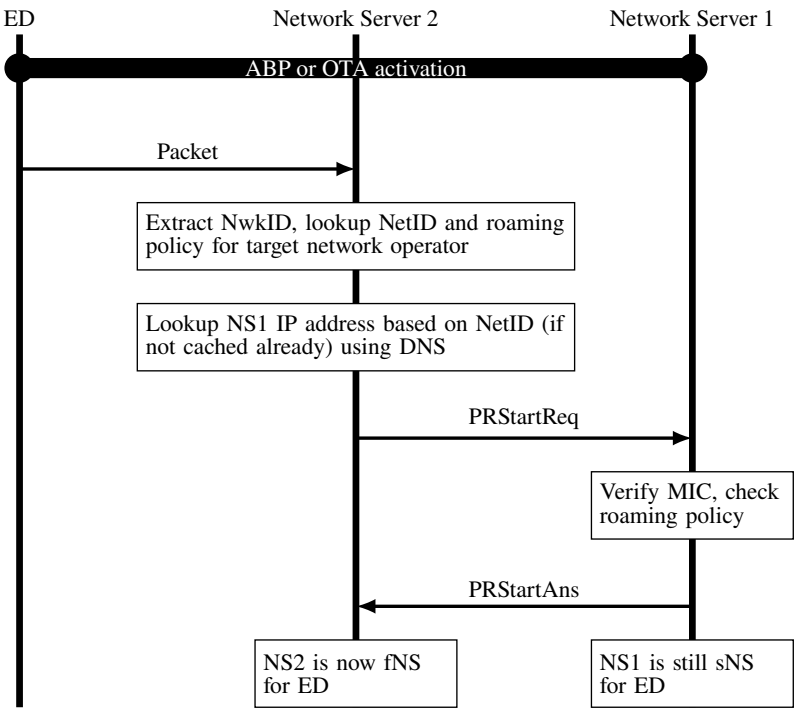


Figure 4.2: Passive roaming: setup procedure. Source: [42]

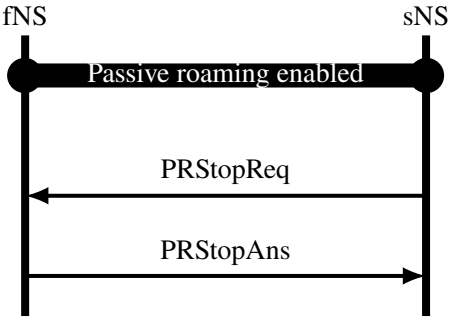


Figure 4.3: Passive roaming: teardown procedure, initiated by the . Source: [42]

To address these issues, the proposed architecture eliminates direct interaction between NSs by introducing SCs to dynamically manage SLAs between network operators. Unlike traditional passive roaming, the solution operates on top of services running on the GW, which are responsible for decoding incoming LoRa packets and forwarding them to the appropriate Serving Network Server SNS as depicted in Figure 4.4.

When a foreign Bridge (fBridge) receives a LoRa packet, it decodes the packet to extract the DevAddr of the ED. Using this DevAddr, the Network ID (NwkID) and Network Server ID (NetID) associated with the ED can be derived. If the IP address of the SNS is not cached, a Domain Name System (DNS) request is generated using the resolved NetID as a parameter. Once the SNS IP address is retrieved, the GW verifies if an active SLA authorizes the transmission. If an SLA is in place, the packet is forwarded to the appropriate SNS.

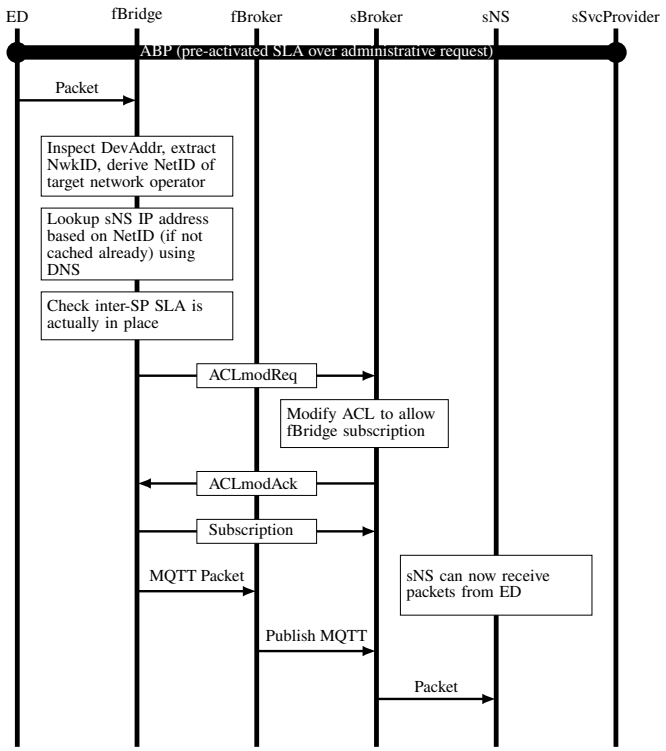


Figure 4.4: Proposed ABP-based data packet roaming (without SLA setup). Source: [42]

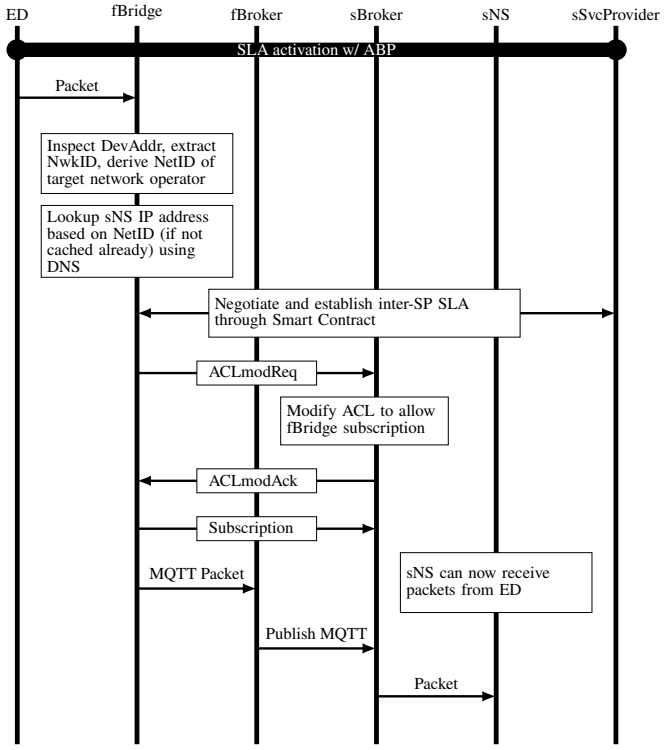


Figure 4.5: Proposed ABP-based data packet roaming (including SLA setup).Source: [42]

If an active SLA does not exist, the GW can initiate an Access Control List modification request Access Control List modification request (ACLmodReq) to the serving Broker (sBroker) to request access. This request can also be automatically negotiated and established through a Smart Contract with the Service Provider, as shown in Figure 4.5. Upon receiving the ACLmodAck, the fBridge subscribes to the sBroker, allowing the forwarding of mangled Message Queuing Telemetry Transport (MQTT) packets to the SNS.

4.1.2.3 Experimental Evaluation and Performance Analysis

A proof-of-concept implementation of the proposed architecture was developed and tested using the ChirpStack open-source LoRaWAN network server stack [15]. The testbed included simulated EDs and GWs interacting with the network server to evaluate packet travel times and processing overhead.

Two experimental setups were employed: the first tested the standard LoRaWAN passive roaming mechanism, while the second implemented the proposed network-agnostic roaming architecture. The results, presented in Tables 4.1 and 4.2, show that the proposed solution significantly reduces packet travel times for networks with fewer than 250 EDs. However, as the number of EDs increases, the processing overhead introduced by the GW becomes more pronounced, leading to increased packet travel times.

Table 4.1: ChirpStack Passive Roaming

Number of EDs	ED to NS Delay (ms)	Confidence Interval (95%)
10	251.79	± 3.23
20	270.94	± 4.76
50	268.63	± 2.72
100	274.88	± 2.33
250	277.09	± 1.56
500	298.78	± 1.71
1000	342.97	± 2.24

Table 4.2: Proposed Roaming Solution

Number of EDs	ED to NS Delay (ms)	Confidence Interval (95%)	GW Handling Overhead (ms)	Confidence Interval (95%)
10	110.17	± 2.37	92.32	± 1.83
20	128.86	± 7.87	87.03	± 2.55
50	174.99	± 11.98	86.88	± 3.89
100	224.22	± 13.09	78.49	± 2.81
250	254.69	± 9.08	68.63	± 1.44
500	428.41	± 12.38	62.36	± 1.02
1000	1493.3	± 29.69	55.53	± 0.44

4.1.2.4 Lessons Learned

- The proposed infrastructure-centric, network-server-agnostic LoRaWAN roaming architecture offers significant improvements over existing solutions by enabling dynamic and flexible roaming without requiring predefined agreements between NSs.
- The use of SCs to manage SLAs and the decoupling of GWs from specific NSs make the architecture scalable and adaptable to various deployment scenarios.
- Future work will focus on optimizing the architecture's performance in high-density environments, particularly where the number of EDs exceeds 250.
- Additionally, real-world trials are planned in collaboration with industry partners to further validate the architecture's applicability in large-scale IoT deployments.

4.1.3 API-Driven LoRaWAN Roaming for Device Mobility in IoT Networks

Building on the need for improved roaming capabilities in IoT networks, as highlighted in previous sections, in [21] is proposed an evolved solution for enabling seamless remote roaming within LoRaWAN networks. This solution leverages an API-driven gateway bridge service that decouples gateway operations from network servers, thereby enhancing mobility support and interoperability across different network domains. These improvements are particularly beneficial for large-scale IoT deployments in urban environments, where device mobility is frequent and scalability is a key concern.

While LoRaWAN version 1.1 introduced basic roaming functionalities, its adoption has been slow due to certification delays and the persistence of legacy infrastructures based on version 1.0. In response, the proposed architecture builds on version 1.0, providing enhanced roaming capabilities without significant infrastructure changes.

4.1.3.1 Proposed Architecture

The proposed architecture illustrated in Figure 4.6 addresses the limitations of LoRaWAN version 1.0 by introducing a flexible and scalable framework that supports seamless device mobility. The architecture comprises three core components: the Gateway Abstraction Layer, the API-driven Gateway Bridge Service, and the Local Database for Roaming Management.

4.1.3.2 Gateway Abstraction Layer

The Gateway Abstraction Layer (GAL) acts as a decoupling mechanism that separates the physical GWs from the NSs, enabling GWs to communicate dynamically with multiple NSs. This decoupling is achieved by abstracting the role of the GW in the network, allowing it to forward packets to different NSs based on real-time network conditions and traffic demands. The primary advantage of this approach is the ability to support seamless device mobility without requiring manual reconfiguration or predefined network agreements.

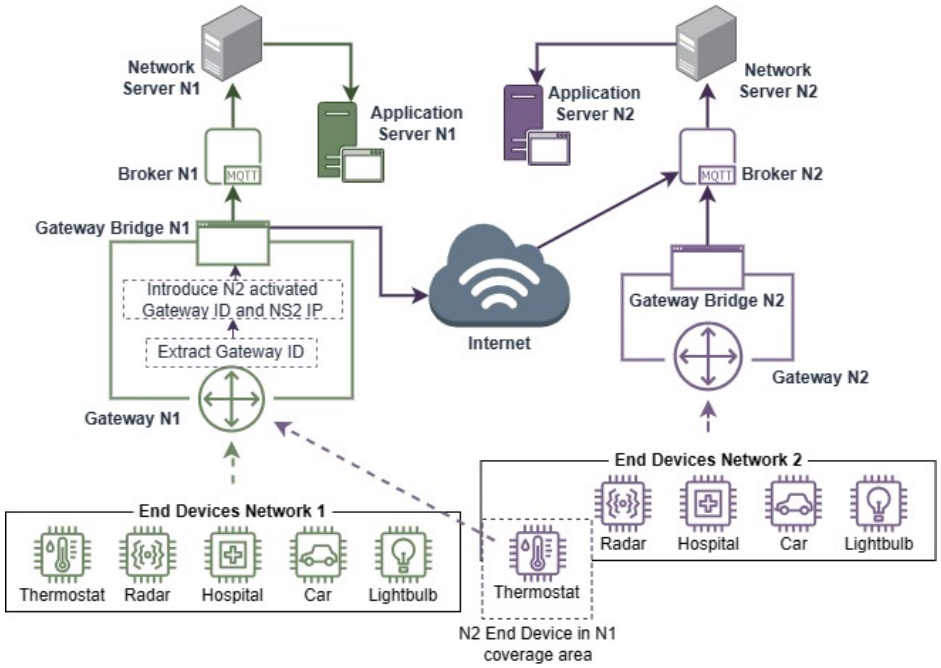


Figure 4.6: Proposed architecture for decoupling GWs from NSs to support dynamic and flexible deployments. Source: [21]

The GAL operates by intercepting packets at the GW level and routing them to the appropriate NS based on device-specific identifiers such as DevAddr and NetID. Upon receiving a packet, the GAL decodes the DevAddr to determine the NS responsible for the device. This process is entirely transparent to the end devices, which continue operating as if connected to their home network, ensuring uninterrupted service during mobility events.

4.1.3.3 API-Driven Gateway Bridge Service

A main innovation of the proposed architecture is the API-driven Gateway Bridge service. This service enhances the traditional GW Bridge by integrating external interaction capabilities through APIs, enabling dynamic activation and management of roaming functionalities. Unlike passive roaming, which relies on complex interactions between NSs, our approach eliminates such interactions. It allows the GW to autonomously manage incoming LoRa packets and forward them to the appropriate serving network server sNS based on device and network identifiers. This API-driven approach ensures that gateways can adapt to changing real-time network conditions, facilitating seamless integration with external systems and services.

The API-driven Gateway Bridge service is fully integrated into the ChirpStack architecture, extending the standard functionalities of the GW Bridge. Through well-defined APIs, external systems can interact with the GW Bridge to dynamically handle packet forwarding from unrecognized GWs to the correct NSs, using identifiers such as the NetID and DevAddr. This architecture eliminates the need for pre-established network agreements, offering a more flexible and scalable solution for network deployment and growth.

As illustrated in Figure 4.7, the system can automatically initiate an ACLmodReq via an API if an active SLA does not exist. This request updates permissions and enables communication

with the sBroker. Once the request is acknowledged, the fBridge can forward the modified MQTT packets to the SNS. This real-time API-driven management allows network operators to activate and manage roaming functionalities remotely, adjusting configurations dynamically based on network conditions, ensuring low-latency, scalable operations without the limitations of traditional passive roaming.

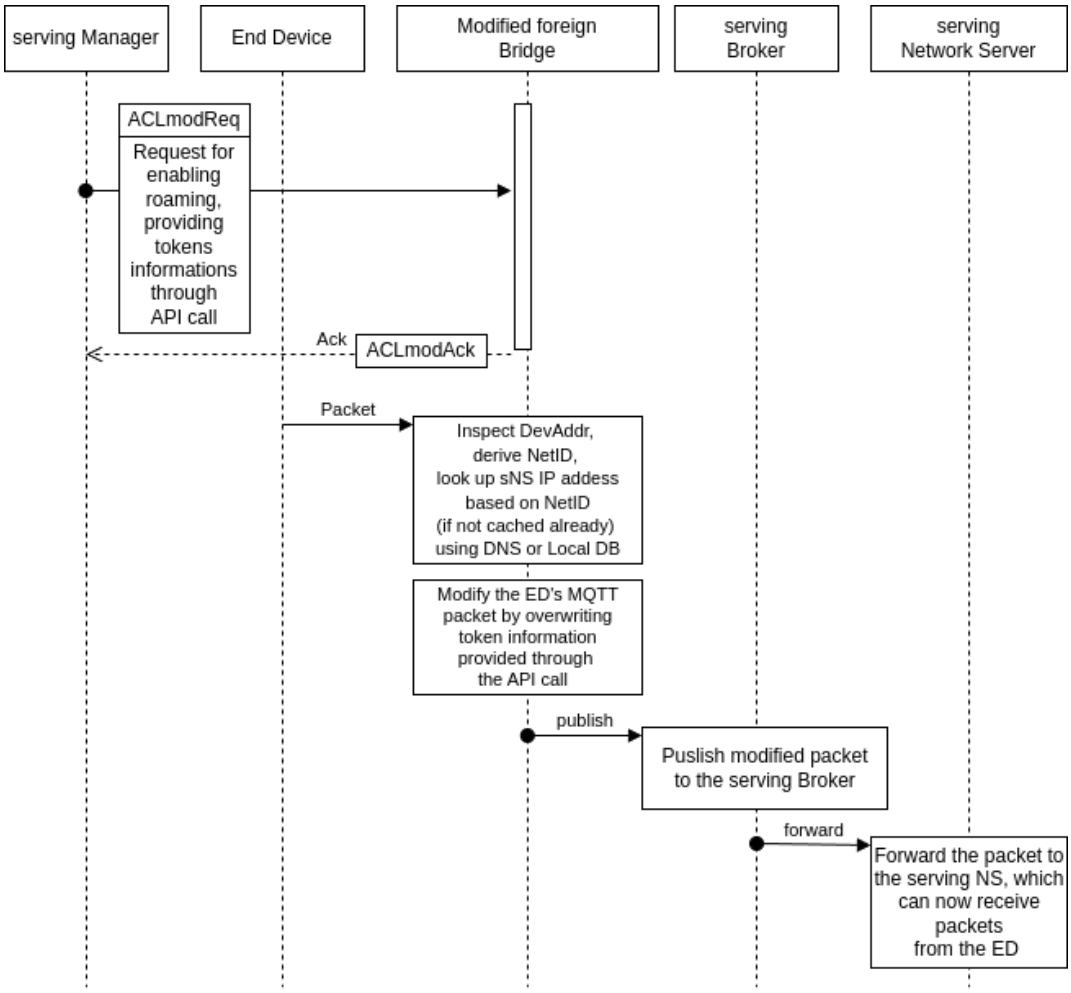


Figure 4.7: Proposed ABP (Activation By Personalization)-based data packet roaming using the API-driven Gateway Bridge service. Source: [21]

4.1.3.4 Local Database for Roaming Management

The architecture incorporates a local database within the GW infrastructure to ensure the reliability and continuity of the roaming process. This database stores essential information, including device identifiers (e.g., DevAddr, NetID) and network configurations, enabling the system to maintain communication during network transitions. The local database is pivotal in managing the roaming process, ensuring that packets are routed correctly even when devices move between different network regions.

In addition to managing device information, the local database is responsible for caching network-related data, reducing the latency associated with DNS queries and other network discov-

ery processes. By maintaining a local cache of network information, the system can quickly resolve the appropriate NS for packet forwarding, thereby minimizing the impact of network transitions on end-device communication.

4.1.3.5 Implementation Details

The proposed architecture was implemented using the ChirpStack open-source LoRaWAN network server stack. The implementation involved modifying the ChirpStack GW Bridge to support API-driven interactions and integrating the local database for roaming management. The system was deployed in a controlled test environment consisting of two separate LoRaWAN networks, each configured with its own ChirpStack NS and Application Server (AS).

4.1.3.6 Hardware Configuration

The test environment was built using a combination of embedded systems, single-board computers, and virtual machines. The specific hardware components used in the experimental setup are detailed in Table 4.3.

Table 4.3: Hardware Configuration for Experimental Setup

Device	**Processor**	**Clock Speed**	**RAM**
Arduino MKR 1310	Arm Cortex-M0 32-bit SAMD21	32.768 kHz	32KB SRAM
Raspberry Pi 4 B	64-bit quad-core Cortex-A72	1.5 GHz	8 GB
PC	Intel Core i7-11800H	2.3 GHz	32 GB

Network 1 was instantiated on a Raspberry Pi 4 model B [28] equipped with 8 GB of RAM and configured with ChirpStack's AS and NS. A RAK 5146 LoRa module [14] and a Pi-Hat adapter were used to set up the LoRaWAN GW. Network 2 was deployed on a virtual machine running Ubuntu 22.04.1, serving as a static network with its own ChirpStack NS and AS instances.

4.1.3.7 Software Implementation

The API-driven Gateway Bridge service was developed as an extension to the existing ChirpStack GW Bridge software. This implementation involved the addition of API endpoints that allow external systems to interact with the GW Bridge, enabling dynamic configuration of roaming functionalities. The local database was implemented using TinyDB, a lightweight NoSQL database that runs locally on the Raspberry Pi.

The key features of the implementation include:

- **Dynamic Packet Routing**: The GW Bridge decodes incoming packets to extract the DevAddr and NetID, determining the appropriate NS for forwarding based on the information stored in the local database.
- **API Integration**: External systems can configure the GW Bridge through API calls, enabling the activation of roaming functionalities and the management of network configurations in real-time.

- **Local Caching**: The local database stores network information, reducing the latency associated with DNS queries and ensuring reliable communication during network transitions.

4.1.3.8 Experimental Validation

The experimental validation of the proposed architecture was conducted in a controlled laboratory environment. The primary objectives of the validation were to evaluate the system's performance in terms of latency, packet loss, and reliability, particularly in scenarios involving device mobility and roaming.

4.1.3.9 Test Scenarios

The experimental tests were created to simulate different scenarios involving roaming and non-roaming devices. The test setup included home network devices connected to Network 1 and roaming devices connected to Network 2. The tests were carried out under various network loads, with different numbers of devices transmitting data at fixed intervals. The testing of the roaming scenarios was done using the network topology as shown in Figure 4.8.

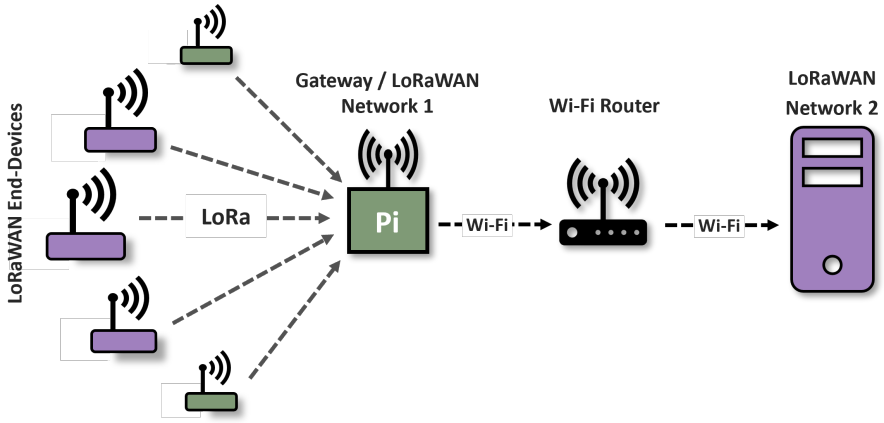


Figure 4.8: Test Network Topology for Experimental Validation. Source: [21]

The primary metrics evaluated during the tests included:

- **Latency**: The time taken for a packet to travel from the end device to the AS, measured at the ChirpStack NS.
- **Packet Loss**: The number of packets lost during transmission, measured at both the GW and NS levels.
- **Signal Quality**: The Received Signal Strength Indicator (RSSI) and Signal-to-Noise Ratio (SNR) for each packet, measured at the GW.

4.1.3.10 Results and Analysis

The experimental results demonstrated that the proposed architecture could handle increased device mobility without significant degradation in performance. The average latency ranged between

360 ms and 400 ms, depending on the network load and the number of active devices. Packet loss was minimal, with only a few isolated incidents observed during high-load scenarios.

Figures 4.9 to 4.10 illustrate the latency measurements for various test scenarios involving different numbers of roaming and non-roaming devices. The results indicate that the system can maintain stable performance even as the number of devices increases.

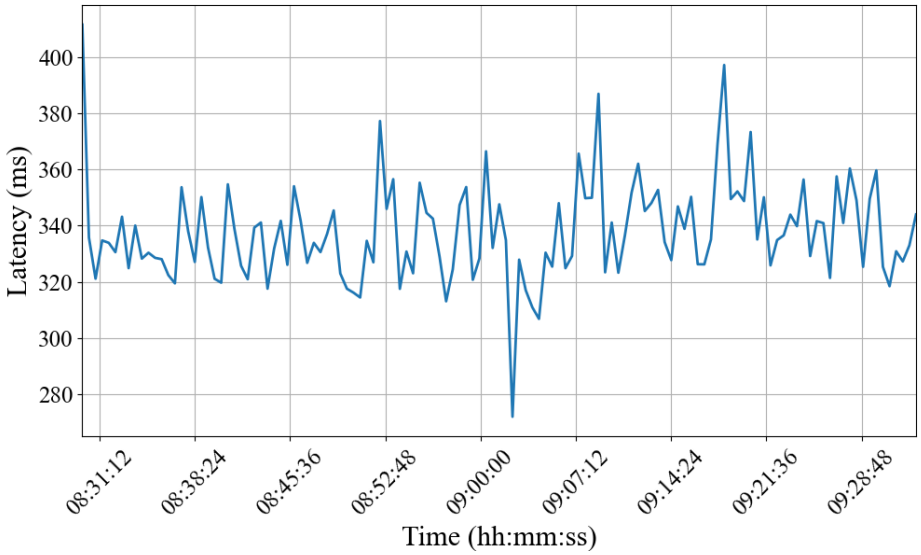


Figure 4.9: Latency at the Chirpstack Server for a Single Roaming Device (No Home Network Devices). Source: [21]

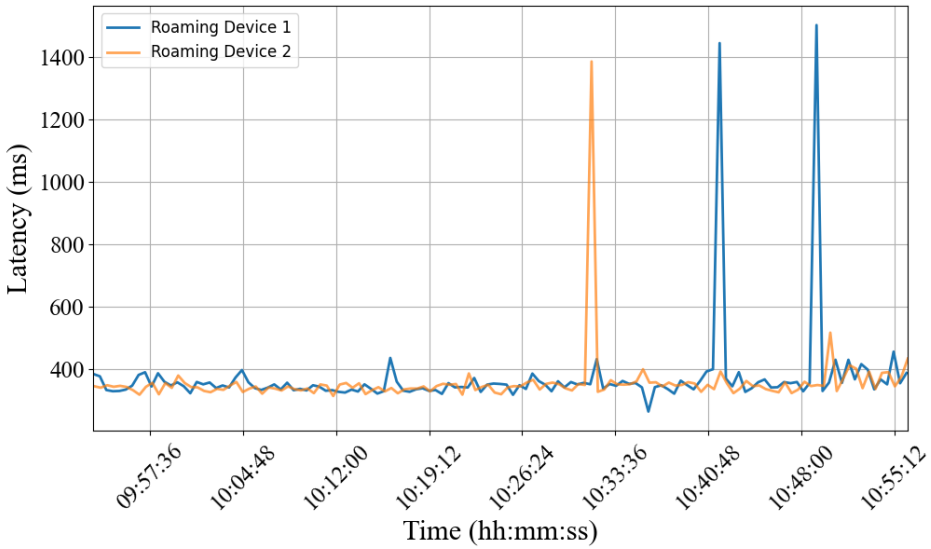


Figure 4.10: Latency at the Chirpstack Server for Two Roaming Devices and Two Home Network Devices. Source: [21]

Additionally, the RSSI and SNR values were consistent across different test scenarios, indicating that the roaming process did not adversely affect signal quality.

4.1.3.11 Scalability and Performance Analysis

To evaluate the scalability of the proposed architecture, extensive simulations were conducted using the LWN-Simulator [55], a widely used tool for simulating LoRaWAN networks. The simulations assessed the system's performance under varying network loads, focusing on latency, packet loss, and signal quality.

4.1.3.12 Simulation Setup

The simulations were conducted using three virtual machines (VMs) configured to represent different network components: *NetworkServer1*, *NetworkServer2*, and the GW Bridge. The VMs were connected via a simulated network environment with a fixed latency of 50 ms, introduced using the `tc-netem` tool. The simulations included scenarios with both stationary and roaming devices, with the number of devices ranging from 10 to 100.

Each VM was configured to simulate the respective network component, and the performance metrics were recorded for different scenarios. The LWN-Simulator provided a controlled environment to assess the impact of various factors, such as device density, mobility patterns, and network load on the system's performance.

4.1.3.13 Simulation Results

The simulation results demonstrated that the proposed architecture could scale effectively, maintaining low latency and high reliability even as the network load increased. The average latency for roaming devices was slightly higher than for non-roaming devices, but the difference was minimal and within acceptable limits for typical IoT applications.

Figures 4.11 and 4.12 present the average latency and latency distribution trends for different simulation scenarios. The results indicate that the system can handle increased network load without significant degradation in performance.

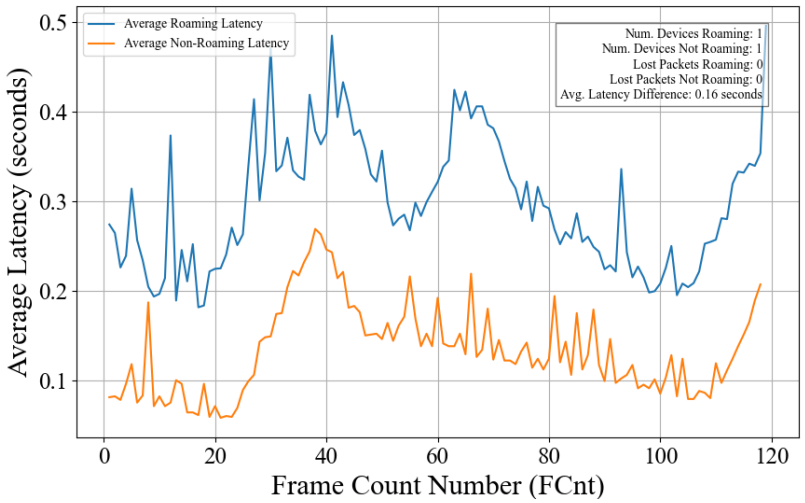


Figure 4.11: Average Packet Latency for Roaming and Non-Roaming Devices in Simulated Scenarios. Source: [21]

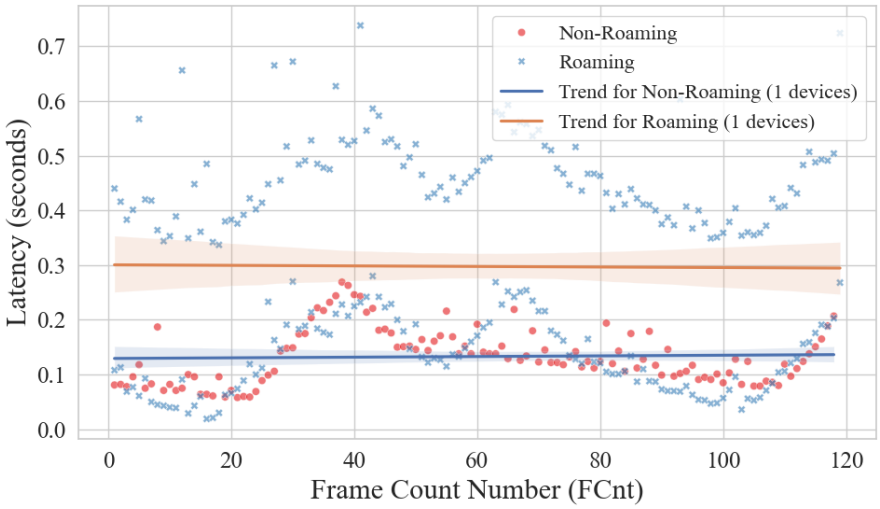


Figure 4.12: Latency Distribution Over Time for Roaming and Non-Roaming Devices in Simulated Scenarios. Source: [21]

In the simulations, even as the number of devices increased to 100, the architecture maintained a stable average latency with minimal packet loss. The scalability tests also revealed that the system could efficiently handle high-density scenarios with multiple roaming devices without compromising performance. This scalability is critical for future IoT deployments, where the number of connected devices is expected to grow exponentially.

Moreover, the simulation results highlighted the proposed architecture's robustness in managing network transitions and maintaining consistent communication quality across different mobility scenarios. The ability to dynamically route packets and manage network configurations in real-time ensures that the system can adapt to varying network conditions, making it suitable for deployment in large-scale IoT environments.

4.1.3.14 Lessons Learned

The experimental results confirm that the proposed architecture successfully addresses the limitations of LoRaWAN version 1.0, particularly in supporting seamless device mobility. Key technical takeaways include:

- The decoupling of GWs from NSs, combined with the GAL, ensures that devices can maintain connectivity while moving between different network regions without manual reconfiguration.
- The adaptive protocol dynamically adjusts the GW-to-NS connection based on real-time traffic conditions, optimizing the use of network resources and minimizing latency.
- The API-driven approach simplifies the deployment of roaming functionalities, allowing for on-demand activation and integration with external systems without needing pre-negotiated agreements between network operators.

4.1.4 IoT Diagnostics in Energy Substations via BLE and Cloud Integration

In the context of critical infrastructure, such as energy substations, ensuring reliable and continuous communication is paramount for maintaining operational efficiency and preventing failures. On-site diagnostics play a key role in this process, enabling timely interventions and facilitating preventive maintenance. A BLE-enabled diagnostic system for energy substations has been developed, integrating IoT devices with a cloud-controlled infrastructure to enhance communication and data flow. This system supports predictive maintenance, reduces downtime, and improves operational performance using IoT and cloud technologies. The following section provides a comprehensive analysis of the system's architecture, implementation, and preliminary results, highlighting the impact of these technologies on enhancing communication and infrastructure management [23].

4.1.4.1 System Architecture

The proposed system architecture is designed to enable efficient monitoring and maintenance of energy substations. It integrates Bluetooth Low Energy (BLE) technology with IoT devices and a cloud-controlled infrastructure to facilitate on-site diagnostics and remote monitoring. The architecture comprises three main components: the Stack4Things (S4T) cloud platform, Arancino embedded boards deployed at the substations, and Android-based smart devices equipped with BLE capabilities.

The sequence diagram in Figure 4.13 illustrates the overall system flow. The Stack4Things platform, based on OpenStack, serves as the central cloud infrastructure that manages the IoT devices deployed at the substations. The Arancino boards, which are equipped with various sensors and actuators, are responsible for collecting data from the substation and transmitting it to the cloud. The Android smartphones act as mobile agents, allowing authorized personnel to perform on-site diagnostics and maintenance by connecting to the Arancino boards through BLE.

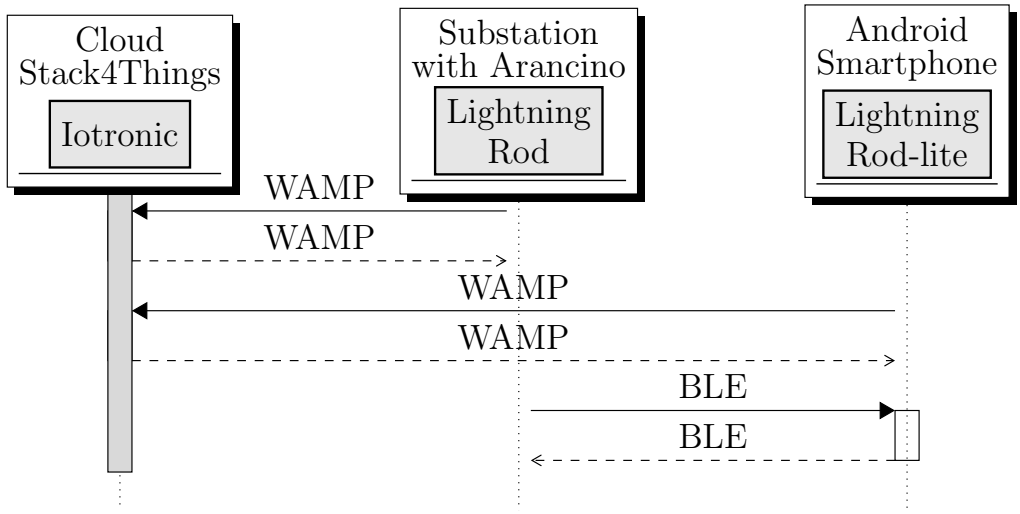


Figure 4.13: Sequence diagram showing interactions between the Cloud (Stack4Things), Substation (Arancino board), and Android Smartphone.

4.1.4.2 Operational Modes

The architecture supports two operational modes: continuous cloud connectivity and on-demand connectivity, depending on the availability of network resources and operational requirements.

In the continuous connectivity mode, as depicted in Figure 4.14, the substation remains connected to the cloud at all times, enabling real-time monitoring and control. In contrast, the on-demand connectivity mode (Figure 4.15) allows the substation to connect to the cloud only when required, reducing dependency on constant network availability.

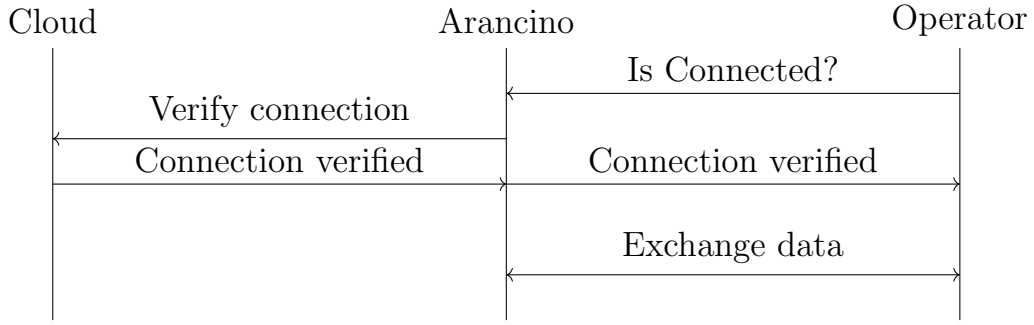


Figure 4.14: Scenario 1: Continuous Cloud Connectivity for Remote Monitoring

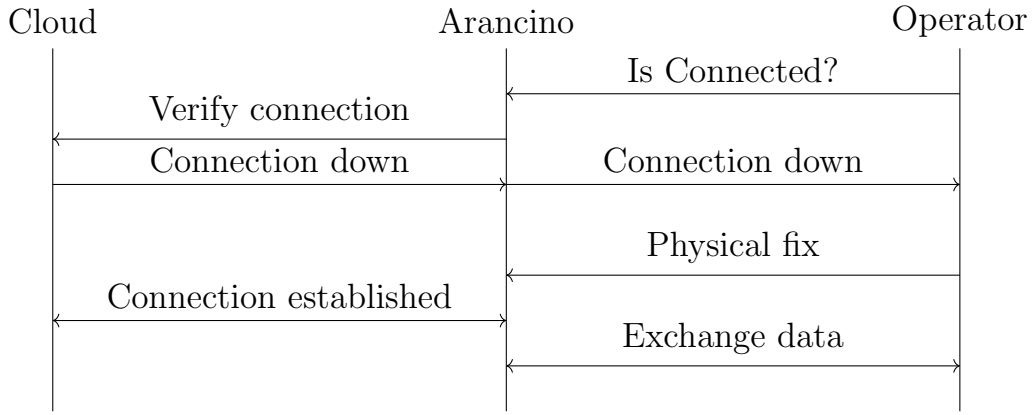


Figure 4.15: Scenario 2: On-Demand Cloud Connectivity for On-Site Diagnostics

4.1.4.3 BLE-Enabled Diagnostics and Communication Protocol

The BLE-enabled diagnostic system facilitates on-site troubleshooting by enabling mobile devices to connect directly to the IoT devices at the substations. The BLE connection provides a low-power, short-range communication channel between the smartphone and the Arancino board. This connection allows real-time data exchange, enabling the smartphone to retrieve metrics from the IoT devices and send commands to control the substation's operations.

BLE technology, operating at 2.4 GHz, is utilized for this purpose. BLE operates under the Generic Attribute Profile (GATT), which organizes device data exchange into Services and Characteristics. A Service is a collection of related attributes, while a Characteristic defines the actual

data value being exchanged. GATT also provides Descriptors, which offer additional information about the Characteristics, such as formatting, ranges, and units.

Figure 4.16 illustrates the structure of a BLE Service and its Characteristics.

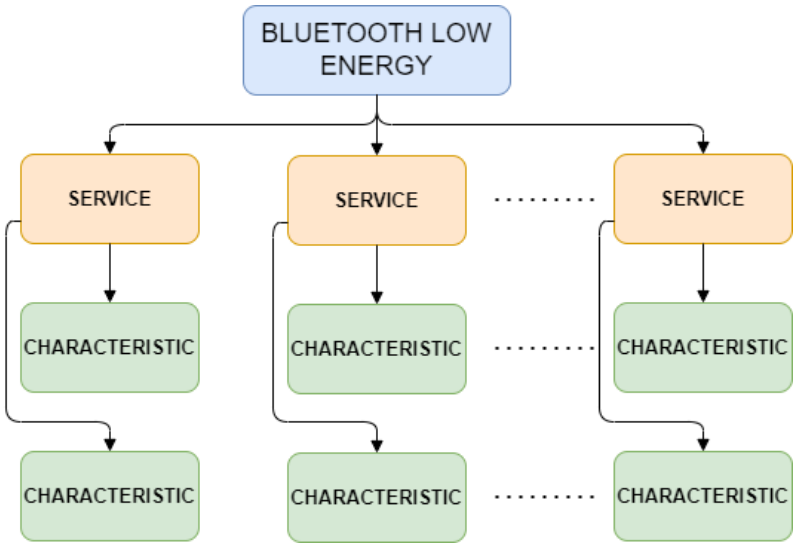


Figure 4.16: BLE Service and Characteristic Structure.

The BLE connection between the smartphone and the Arancino board is established following a sequence that includes scanning for BLE signals, establishing a connection, and then exchanging data through the GATT protocol. The BLE Advertising Data Packet, shown in Table 4.4, contains information critical to establishing this connection, such as the device's Media Access Control (MAC) address, Universally Unique Identifiers (UUIDs), and flags indicating the device's capabilities and status.

AT	AAT	Advertiser Address	Advertising Data (payload)
1B	1B	6 Byte	Max 31 Byte

Table 4.4: BLE Advertising Data Packet Structure.

Once the connection is established, the GATT server on the Arancino board manages the data exchange. This includes providing the list of available Services and Characteristics, which the smartphone can query to retrieve data points or control specific functions. The GATT server also allows the subscription to notifications or indications, enabling the smartphone to receive updates when specific data values change.

The overall BLE communication process is depicted in the diagram in Figure 4.17, displaying the interaction between the smartphone and the Arancino board during the BLE communication sequence. This diagram illustrates the typical data flow during on-site diagnostics, from scanning and connection establishment to the exchange of Services and Characteristics.

The BLE GATT server structure on the Arancino board is crucial for managing the services and characteristics that enable on-site diagnostics and control operations. Figure 4.18 illustrates the BLE GATT server, which supports various services and characteristics tailored to the needs of

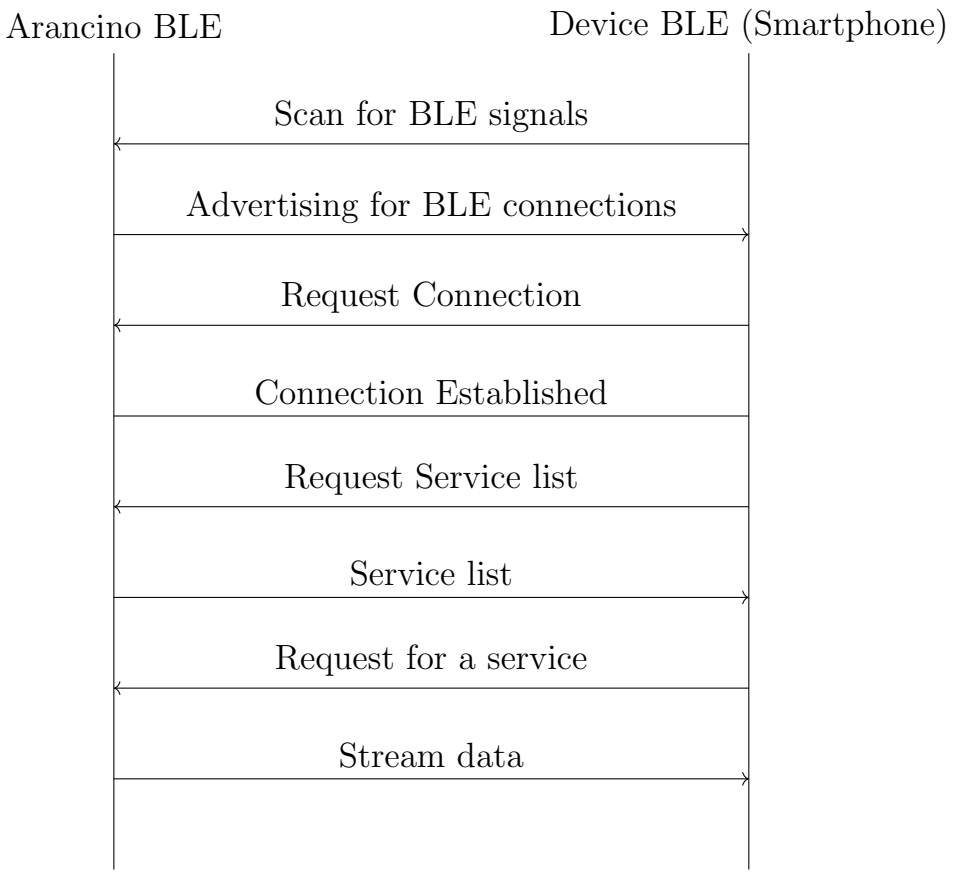


Figure 4.17: BLE Communication Sequence between Arancino and Mobile Device

the substation. These include sensor data monitoring, actuator control, and diagnostic information retrieval.

4.1.4.4 Lessons Learned

The deployment and testing of the BLE-enabled diagnostic system provided valuable insights:

1. BLE Range and Stability: The BLE connection, while reliable for short-range communication, was limited to approximately 50 meters. Physical obstructions and interference affected signal stability. A mesh network approach or signal repeaters could improve performance in larger installations.

2. Data Synchronization: Synchronizing data collected during BLE diagnostics with cloud-based records posed challenges. Developing more robust conflict resolution mechanisms and synchronization protocols is essential to prevent data discrepancies.

3. Power Efficiency: BLE proved to be energy-efficient, significantly reducing power consumption compared to continuous cloud connectivity. However, further optimizations are required to maximize the lifespan of Arancino boards in remote installations, especially in off-grid settings.

4. Usability of Mobile Diagnostics: Field tests indicated that the mobile diagnostic app was effective but required User Interface (UI) improvements. Enhancing the user experience through clearer workflows and error messages would improve efficiency for on-site personnel.

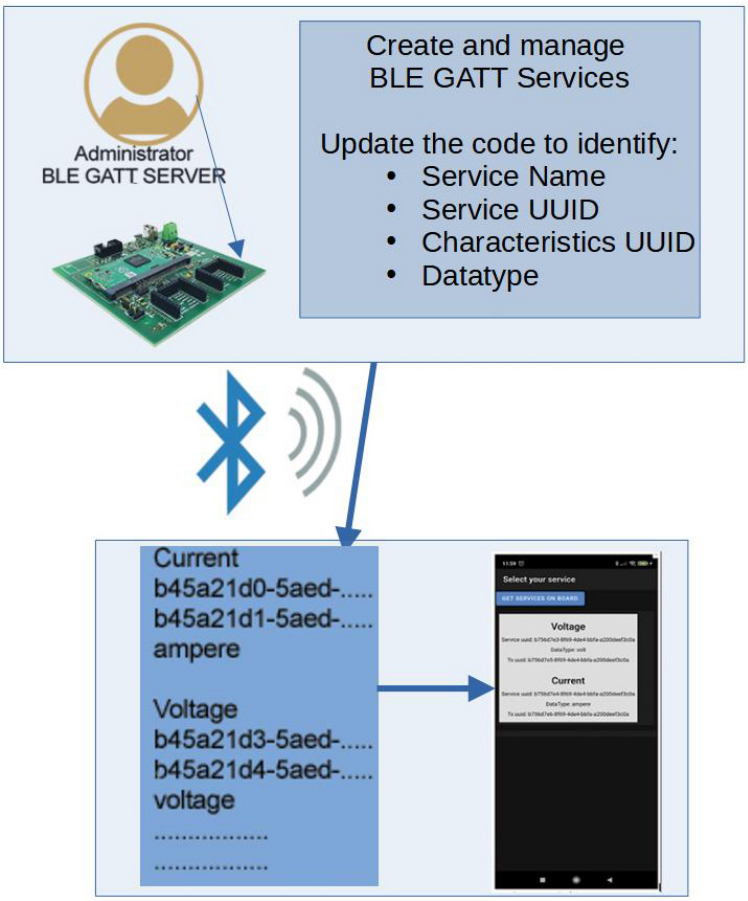


Figure 4.18: BLE GATT Server Structure on Arancino Board. Source: [23]

These lessons will guide future system enhancements, particularly in BLE performance, data management, and user interface improvements.

4.1.5 Meshing IoT Networks with WebRTC-Based P2P Overlay Networks

To further augment the communication of IoT infrastructures, another research activity has explored the use of peer-to-peer (P2P) overlay networks based on Web Real-Time Communication (WebRTC) technology. This approach addresses the challenges of cloud-centric IoT systems by enhancing scalability, reducing latency, and improving security. The deployment of WebRTC-based P2P overlay networks at the edge provides a decentralized solution well-suited for real-time applications in dynamic IoT environments [18].

4.1.5.1 System Architecture

The proposed system architecture leverages Web Real-Time Communication (WebRTC) technology to deploy peer-to-peer (P2P) overlay networks at the edge of IoT infrastructures. The primary

motivation for adopting this decentralized architecture is to address the inherent limitations of traditional cloud-centric IoT systems, such as high latency and single points of failure.

The architecture comprises two main components: the cloud-side infrastructure and the edge-side nodes. The cloud side is responsible for orchestrating the deployment of IoT devices and managing the network topology, while the edge-side nodes, or IoT boards, handle local data processing and direct communication with other nodes in the network.

As shown in Figure 4.19, the WebRTC-based system enables direct communication between IoT devices using STUN and signaling agents, thus avoiding the reliance on centralized servers for data exchange.

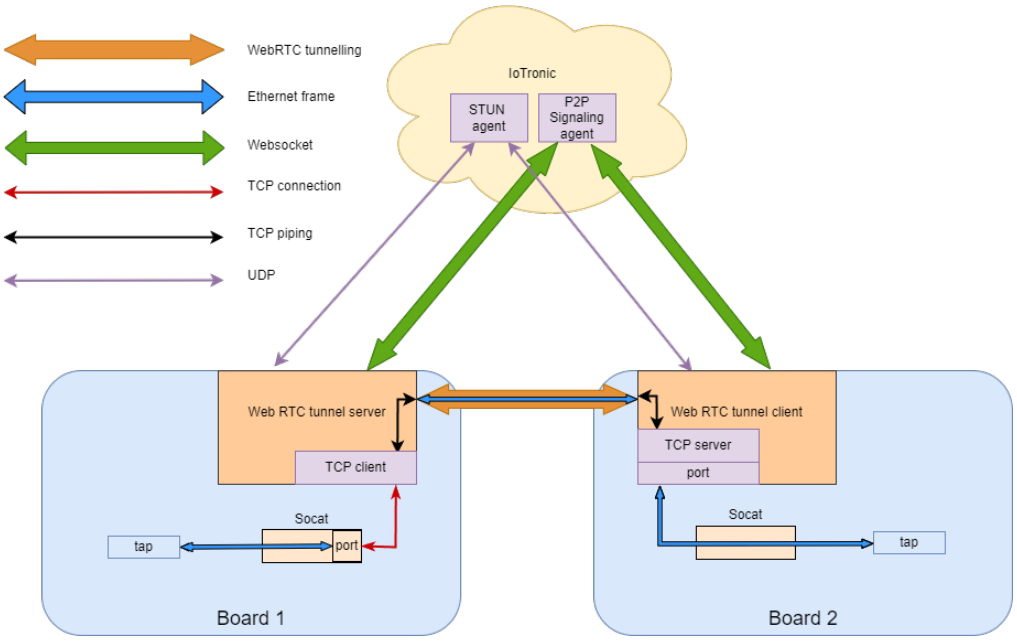


Figure 4.19: P2P connection between two IoT devices using STUN and signaling agents.

4.1.5.2 Network Virtualization

Network virtualization (NV) techniques are integrated into the Stack4Things (S4T) framework, which is built on the OpenStack ecosystem to facilitate seamless communication between IoT devices. NV allows the creation of logical networks that operate independently of the underlying physical infrastructure. This abstraction enables multiple virtual networks to coexist on a single physical network, enhancing flexibility, scalability, and security.

Integrating OpenStack Neutron with the S4T framework provides advanced networking capabilities such as routing, firewalling, and load balancing. These functionalities enable dynamic management of virtual networks across geographically distributed IoT devices. The key NV technologies employed include:

- **LinuxBridge:** A virtual switch within the Linux kernel that supports Virtual Local Area Network (VLAN) creation, providing basic NV capabilities.

- **Virtual Extensible Local Area Network (VXLAN):** A protocol encapsulates Layer 2 frames within Layer 3 UDP packets, creating large-scale virtual networks across data centers.
- **Open vSwitch (OVS):** An open-source virtual switch that supports advanced features such as traffic control, Quality of Service (QoS), and integration with Software-Defined Networking (SDN).

The network virtualization layer abstracts the physical network, allowing IoT devices to be managed as virtual entities. This enables the deployment of IoT applications across cloud and edge environments, ensuring consistent performance in real-time data processing and low-latency communication.

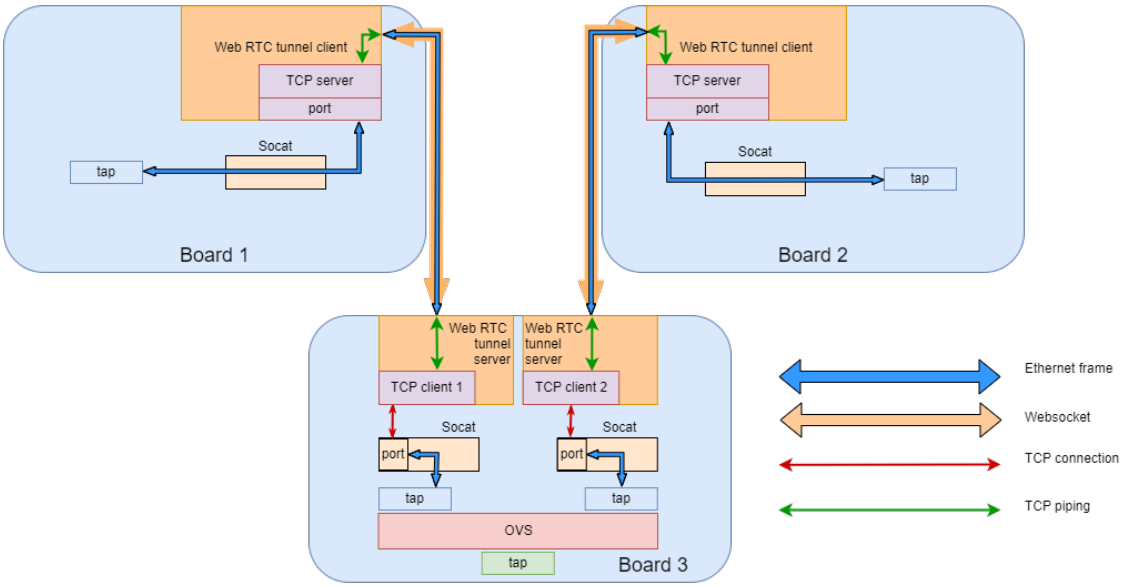


Figure 4.20: Network configuration with three IoT devices and a virtual switch using STUN signaling.

4.1.5.3 WebRTC-Based P2P Overlay Networks

The proposed architecture's core is using WebRTC to establish P2P overlay networks among IoT devices. WebRTC is a suite of protocols and APIs that enable direct communication between devices over the Internet without needing an intermediary server. This technology offers significant advantages regarding latency reduction, scalability, and security.

In a WebRTC-based network, each IoT device communicates directly with peers using STUN and ICE techniques to traverse Network Address Translation (NAT) and firewalls. Figure 4.20 illustrates a more complex topology where three IoT devices are connected, with one acting as a hub through Open vSwitch (OVS).

4.1.5.4 NAT Traversal Techniques

Network Address Translation (NAT) and firewall traversal are critical challenges for P2P communication. The following WebRTC components are employed to overcome these challenges:

- **STUN (Session Traversal Utilities for NAT):** A protocol that allows clients to discover their public IP address and port, helping navigate NAT restrictions. STUN servers provide the necessary information to establish direct P2P connections.
- **ICE (Interactive Connectivity Establishment)**: ICE is a framework WebRTC uses to gather and prioritize multiple candidate addresses to establish a connection. These include local, STUN, and TURN candidates, allowing for the best possible connectivity path.
- **TURN (Traversal Using Relays around NAT):** TURN servers relay data between nodes when direct P2P communication is not possible, typically in cases of symmetric NAT.
- **NAT Hole Punching:** This technique establishes P2P connections by creating "holes" in the NAT, enabling direct communication. However, symmetric NAT often requires the use of TURN servers for connectivity.

These techniques enable reliable communication between IoT devices in complex network environments.

4.1.5.5 S4T Framework Integration

The Stack4Things (S4T) framework provides a robust foundation for managing IoT device lifecycles and orchestrating network configurations. It consists of two primary components:

- **IoTronic:** The server-side component manages IoT resources in the cloud. It handles device registration, configuration, and network orchestration, enabling the dynamic addition and removal of IoT devices.
- **Lightning-Rod:** The client-side component is implemented on IoT devices. It acts as a bridge between the device and the cloud, facilitating secure communication and remote management. Lightning-Rod also supports various plugins, such as `wstunnel` for WebSocket-based communication and `WebRTC` for P2P connections, see Figure 4.21.

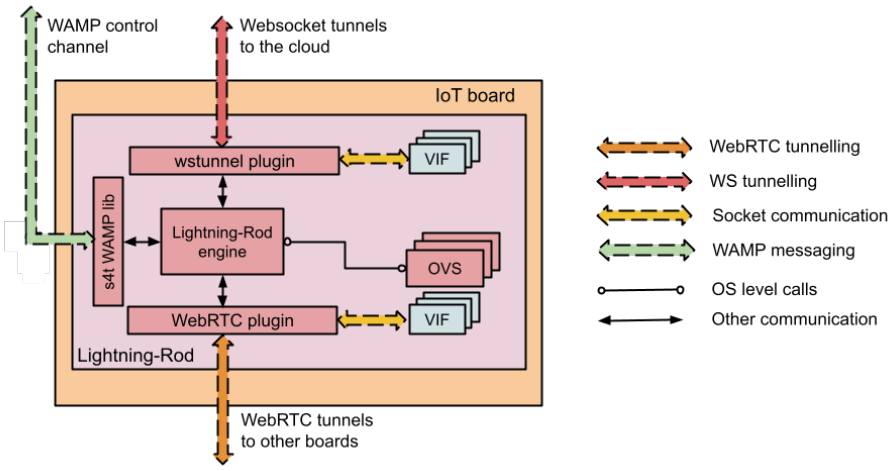


Figure 4.21: Node-side architecture of the S4T framework.

Integrating WebRTC into the S4T framework allows IoT devices to establish direct P2P connections for data exchange, bypassing centralized cloud servers. This approach reduces latency, enhances scalability, and improves the overall reliability of the IoT network.

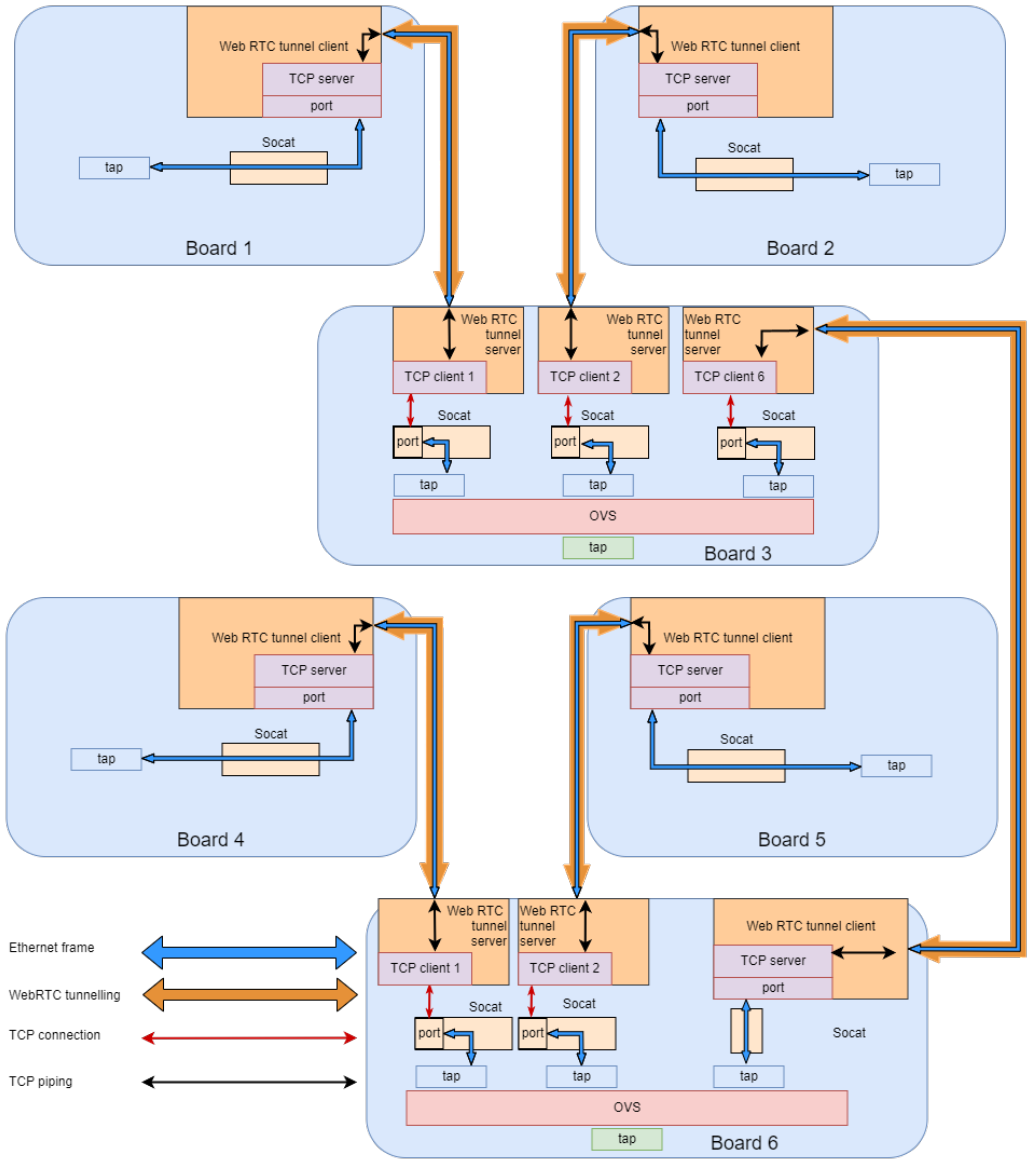


Figure 4.22: Complex network architecture with six IoT devices and virtual switches.

4.1.5.6 Experimental Setup and Evaluation

Several experiments involving different network topologies have been conducted to evaluate the proposed architecture's performance. These experiments measured the propagation time of messages across various configurations, comparing the performance of WebRTC-based P2P overlay networks with traditional cloud-based communication.

4.1.5.6.1 Two-Device Connection Scenario Two IoT devices are connected via a P2P overlay network in the simplest configuration, as shown in Figure 4.19. The devices first connect to a STUN server to obtain their public IP addresses and ports. Then, they establish a direct connection using P2P signaling, facilitating the exchange of Session Description Protocol (SDP) messages and ICE candidates.

4.1.5.6.2 Three-Device Connection Scenario In a more complex scenario, a third device is introduced, with one of the devices acting as a hub, as depicted in Figure 4.20. Open vSwitch (OVS) is used to manage data flow between the devices, ensuring optimal network performance.

4.1.5.6.3 Complex Network Architecture A network topology with six IoT devices was tested to evaluate further scalability, two of which acted as hubs (Figure 4.22). The results showed that the WebRTC-based P2P overlay network maintained low latency and efficient communication, even as the network size increased.

Table 4.5: Comparison of message propagation times between centralized and P2P overlay networks.

Network Configuration	Centralized (ms)	P2P Overlay (ms)
2 Devices	15.2	8.1
3 Devices	22.5	11.3
6 Devices	45.6	19.7

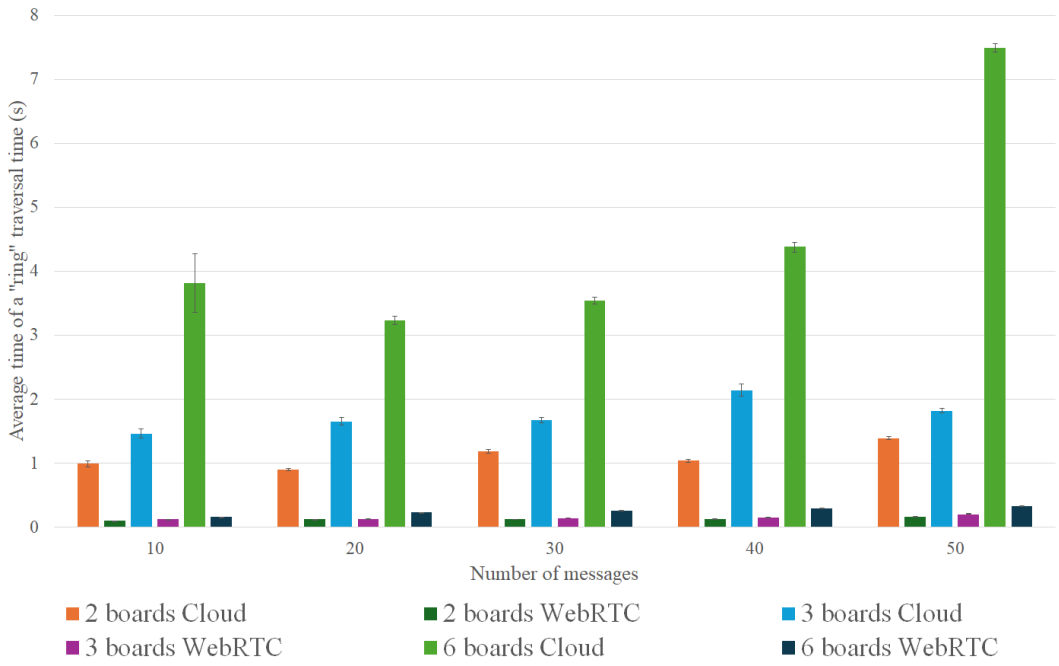


Figure 4.23: Message exchange comparison between centralized and P2P overlay networks.

4.1.5.7 Results

The experimental results, summarized in Table 4.5, demonstrate significant performance improvements with the WebRTC-based P2P overlay network compared to traditional cloud-based communication. The P2P network consistently exhibited lower message propagation times, especially in configurations with larger devices. Moreover, Figure 4.23 illustrates the average "ping" traversal time across different network configurations, highlighting the impact of overlay networks on performance. The data clearly show that WebRTC-based P2P overlay networks significantly outperform traditional cloud-based solutions, especially as the number of messages increases.

4.1.5.8 Lessons Learned

The integration of WebRTC into the S4T framework has yielded several important insights:

1. Scalability of WebRTC: The P2P overlay network demonstrated low latency even in more complex topologies, confirming its scalability for larger IoT networks.

2. Network Virtualization Efficiency: Using VXLAN and OVS within the S4T framework enhanced flexibility and scalability, though further optimization of traffic management is required to maximize performance.

3. NAT Traversal: While STUN and ICE handle most NAT traversal issues, symmetric NAT requires TURN servers, which add latency and complexity to the system.

4. Latency Reduction: The P2P overlay network consistently exhibited lower latency than cloud-based communication, making it suitable for real-time applications.

These findings will guide future work to optimize network management and explore predictive analytics for IoT networks.

4.2 Smart Cities and Urban Infrastructure

Communication technologies provide the backbone for efficient IoT connectivity across various domains. These advancements are now being applied to tackle broader challenges in urban infrastructure. The following sections will explore how IoT and cloud-based systems are key enablers in addressing these challenges, particularly in the context of sustainable and efficient mobility solutions for smart cities.

The evolution of smart cities marks a significant shift in urban planning, where the integration of advanced technologies aims to improve quality of life, enhance resource efficiency, and deliver innovative services. Technologies such as IoT, cloud computing, and artificial intelligence play a pivotal role in managing urban infrastructure and addressing the complexities of modern urbanization. The following sections delve into the infrastructure of smart cities, highlighting how these technologies contribute to creating sustainable, adaptive, and highly efficient urban environments.

4.2.1 IoT/Cloud-powered Green Mobility Solutions in Smart Cities

Mobility is a fundamental component of smart cities, with sustainable transportation solutions playing a crucial role in reducing environmental impact and enhancing urban life. Integrating IoT and cloud computing into mobility services enables real-time data collection and analysis, facilitating efficient public transportation, intelligent parking systems, and pollution monitoring. This

subsection discusses the architecture and applications of an IoT/cloud-powered mobility system designed to optimize urban transport and reduce carbon emissions [20].

4.2.1.1 System Architecture: Integrating IoT and Cloud for Smart City Solutions

The core of this research is the development of a multilevel IoT/Cloud infrastructure designed to address urban mobility and environmental sustainability challenges. The architecture integrates IoT devices with cloud computing, allowing real-time data processing and decision-making across a distributed network of sensors and smart devices.

The architecture is based on OpenStack, a widely adopted cloud computing platform that facilitates creating and managing virtual machines (VMs) and networks. By extending OpenStack's capabilities through the Stack4Things (S4T) framework, the system allows seamless integration of geographically distributed IoT nodes into the cloud infrastructure. This integration enables the collection, transmission, and processing of data from various urban sensors, essential for managing mobility services, detecting available parking slots, and monitoring air quality in real-time [38, 26, 6].

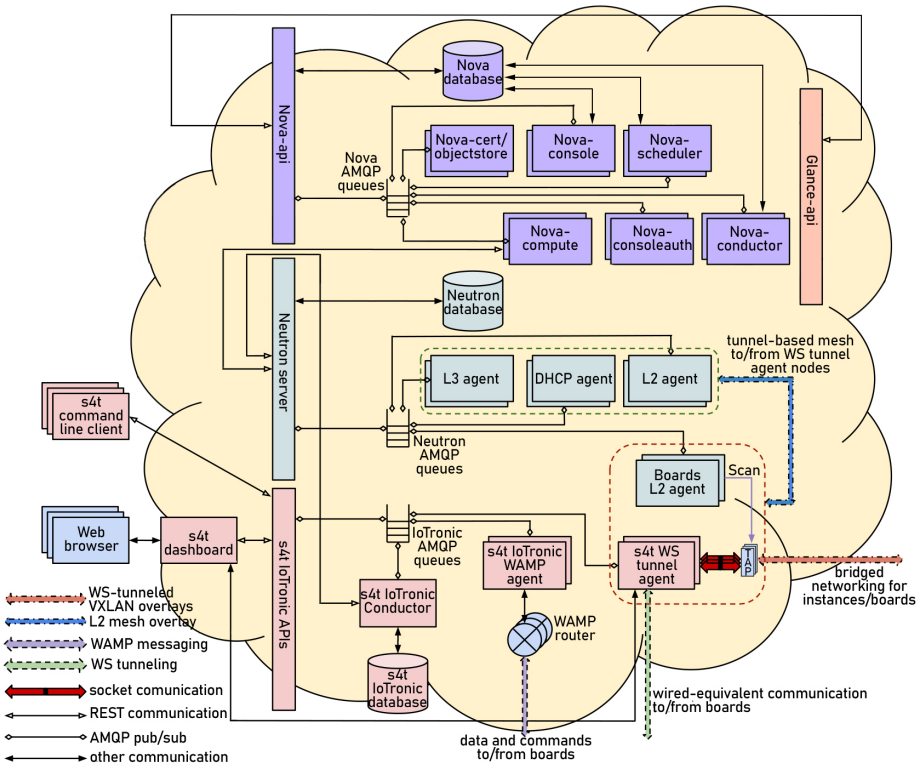


Figure 4.24: Architecture of the IoT/Cloud-powered infrastructure integrating OpenStack and Stack4Things (S4T) for managing IoT nodes across a smart city environment. Source: [20]

Figure 4.24 illustrates integrating IoT nodes and cloud resources. Using WebSocket with reverse tunneling mechanisms ensures reliable communication between distributed IoT nodes and the cloud, overcoming challenges posed by network address translation (NAT) and firewalls.

Key components of the system include:

- **S4T IoTronic Service:** Manages IoT devices as cloud resources and enables interaction with OpenStack subsystems.
- **Lightning-Rod Agents:** Deployed on IoT nodes, these agents manage local resources and communicate with the IoTronic service.
- **OpenStack Cloud Resources:** Provides the computational and networking infrastructure for hosting VMs and managing cloud services.

4.2.1.2 Use Cases: Practical Implementations of the IoT/Cloud Framework

The IoT/Cloud infrastructure developed in this research supports several practical use cases, demonstrating its applicability in real-world smart city scenarios. The following sections describe these use cases in detail, with corresponding data and architectural diagrams.

EasyTaxi: Optimizing Urban Mobility through Predictive Analytics

The EasyTaxi service utilizes distributed IoT technologies and machine learning to predict the optimal positioning of taxis based on historical booking data. This service reduces unnecessary driving and idle time, minimizes emissions, and improves overall efficiency.

Figure 4.25 shows the system's architecture. The data processing pipeline involves:

- **Data Collection:** Booking data is collected and stored in a No-SQL database.
- **Predictive Analytics:** A Linear Regression model is used to predict taxi demand across different zones based on historical data.
- **User Interaction:** A web-based interface allows users to book taxis and view predicted taxi availability in real-time.

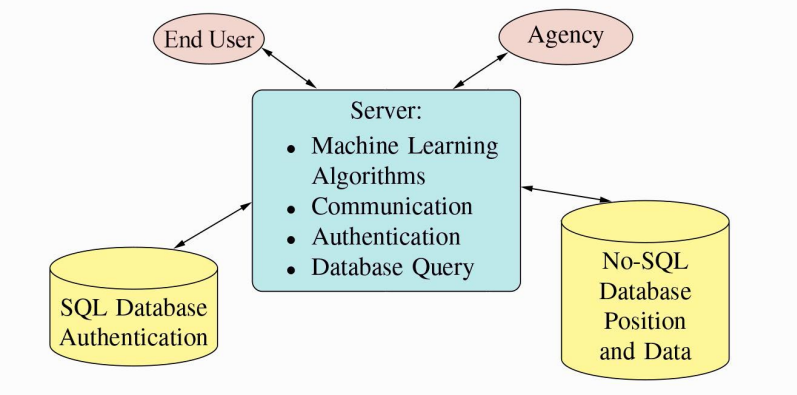


Figure 4.25: System architecture for EasyTaxi, showing the integration of predictive analytics and IoT data for optimizing taxi positioning. Source: [20]

The predictive model was validated using historical booking data from New York City's Yellow Taxi service. Figure 4.26 shows the performance of different linear regression models, with the degree-1 model proving to be the most effective for predicting future bookings.

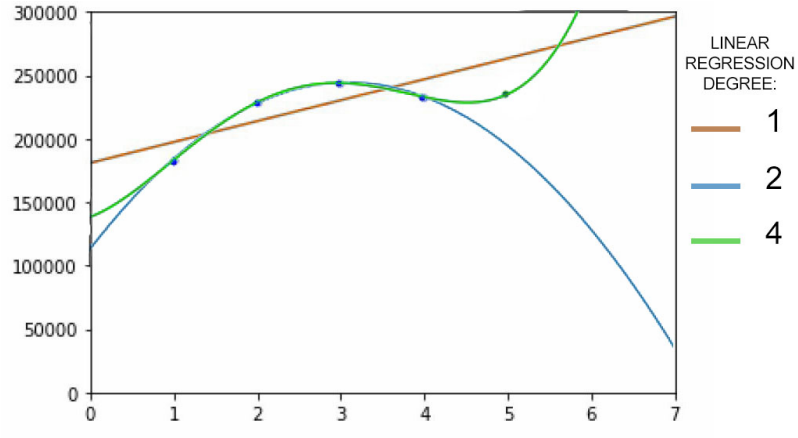


Figure 4.26: Linear regression model comparison for predicting taxi bookings. The degree-1 model (green) demonstrates the best performance. Source: [20]

PlaceHolder: Enhancing Parking Efficiency in Urban Areas

The PlaceHolder service addresses parking challenges by providing real-time information on available parking slots. The system uses crowdsourced data from IoT devices and integrates with existing urban infrastructure, offering a comprehensive parking management solution.

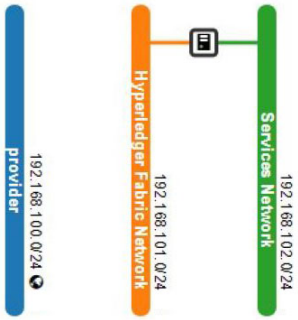


Figure 4.27: Network topology for the PlaceHolder service showing integration with the IoT and blockchain networks. Source: [20]

The architecture of PlaceHolder leverages blockchain technology (Hyperledger Fabric) to ensure data integrity. The following functions are implemented as part of the blockchain-based parking management system:

- **AddParkingZone:** Creates new parking zones.
- **BookSlotInParkingZone:** Reserves a parking slot in a specific zone.
- **FreeSlotInParkingZone:** Frees up a parking slot when it becomes available.
- **GetAllParkingZones:** Retrieves all parking zones and their statuses.

A RESTful API allows the system to interact with the database and blockchain, ensuring that parking data is consistently updated and available to users.

Figure 4.27 shows the network topology used by the PlaceHolder service, illustrating the integration of IoT and blockchain technologies.

Weather Stations: Real-Time Air Quality Monitoring and Alerting System

Using distributed sensors, the Weather Stations service monitors air quality in real-time across different urban areas. This system not only collects data on pollutants but also integrates blockchain technology to ensure the authenticity of the data. The service provides real-time alerts when pollution levels exceed predefined thresholds, enabling authorities to act immediately.

Figure 4.28 illustrates the network topology for the Weather Stations service. Data generated by the sensors is processed and stored in a cloud database, with the results visualized through a Grafana dashboard. The system also includes an alert mechanism that uses a Telegram bot to notify users when air quality levels are critical.

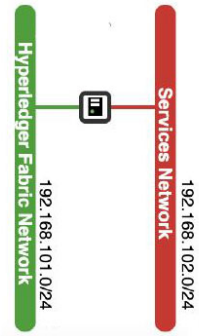


Figure 4.28: Network topology for the Weather Station service, showing the integration of IoT devices with cloud resources for real-time air quality monitoring. Source: [20]

4.2.1.3 Lessons Learned

The design and implementation of the IoT/Cloud infrastructure revealed several important lessons:

1. Scalability: The P2P overlay network demonstrated the system's ability to scale effectively while maintaining low-latency communication. This scalability is particularly critical in managing large-scale IoT deployments across urban environments.

2. Blockchain Integration: The integration of Hyperledger Fabric for services like parking management ensured data integrity and security. However, the added complexity of managing blockchain transactions introduced some latency. Optimizing blockchain protocols for IoT environments will be essential for future scalability.

3. Predictive Analytics: The EasyTaxi use case successfully reduced emissions and improved operational efficiency by optimizing taxi positioning. The use of predictive analytics in urban mobility services proved valuable, and future work could explore more advanced machine learning models.

4. Modularity and Flexibility: The modular architecture, leveraging Docker containers and microservices, allowing for flexible system deployment. This design should be further optimized for easier maintenance and the addition of new services in the future.

These lessons will guide the continued development of the IoT/Cloud architecture for more complex smart city applications, focusing on enhancing system scalability, security, and real-time performance.

4.2.2 IPA-enabled IoT Resource Management in Smart Environments

Building on the technologies that enable efficient and sustainable urban infrastructures, the rapid evolution of smart cities necessitates more advanced methods for dynamically managing IoT resources. This is particularly important in scenarios where device configuration and reconfiguration must be performed quickly and without physical intervention.

In this context, integrating Intelligent Personal Assistants (IPAs), such as Amazon Alexa and Google Home, offers a promising solution for managing IoT devices in temporary urban environments like hotel rooms. Users can configure and control IoT devices with minimal effort through voice commands or mobile applications by leveraging technologies such as Software-Defined I/O (SDI/O) and the S4T platform. This approach enhances IoT systems' flexibility and ease of deployment, providing a seamless and secure interface between residents or visitors and the urban infrastructure. Such innovations improve the user experience in smart cities and demonstrate the potential for creating responsive, user-centered urban environments [60].

4.2.2.1 Integration of Intelligent Personal Assistants (IPAs) with IoT Devices

The integration of voice-controlled Intelligent Personal Assistants (IPAs), such as Google Home, Amazon Alexa, Apple Siri, and Microsoft Cortana, into daily life has been significantly accelerated by advances in Artificial Intelligence (AI) and Machine Learning (ML). These devices have become indispensable for managing various tasks, from retrieving information to controlling home automation systems. However, configuring IPAs to interact with Internet of Things (IoT) devices often involves navigating complex manufacturer-specific interfaces and ensuring compatibility across various platforms, a challenge that is both time-consuming and error-prone [24, 48].

This paper addresses these challenges by introducing a novel Software-Defined Input/Output (SDI/O) approach, which abstracts the underlying IoT infrastructure from user-facing applications. This abstraction simplifies IoT device management and enhances the user experience, particularly in environments like hotel rooms, where user access is temporary and dynamic. The SDI/O paradigm is implemented using the Stack4Things (S4T) middleware, an OpenStack-based IoT platform that orchestrates data flows and controls access to IoT devices while ensuring both security and ease of use.

The proposed system, "Everywhere IPA" (EIPA), automates the configuration and management of IoT devices in dynamic environments, reducing the overhead associated with manual configuration and enhancing security. By leveraging the S4T framework and SDI/O concepts, EIPA facilitates seamless interactions between IPAs and IoT devices in scenarios such as smart hotels, where IoT resources must be frequently reconfigured for different users.

Intelligent Personal Assistants (IPAs) have gained widespread adoption in various domains, including personalized recommendation systems and healthcare, where they are often integrated with IoT devices. Applications like smart home systems and wearable devices rely on IPAs for natural language processing and speech recognition, providing users with an intuitive interface for controlling devices. However, the computational complexity of these tasks, including speech and

image recognition, requires high-performance computing resources typically provided by cloud-based systems [1, 29].

Stack4Things (S4T) is an OpenStack-based platform designed to extend cloud computing capabilities to IoT devices, which are often resource-constrained. S4T consists of two primary components: the cloud-side IoTronic service and the device-side Lightning-Rod (LR) agent. IoTronic manages IoT devices as cloud resources, allowing dynamic orchestration and real-time control. At the same time, LR agents operate on IoT devices to facilitate communication with the cloud via WebSockets and the Web Application Messaging Protocol (WAMP). This architecture enables the seamless integration of geographically distributed IoT devices, from microcontroller-based systems like Raspberry Pi to more powerful edge servers, into the cloud ecosystem (see Fig. 4.29) [53].

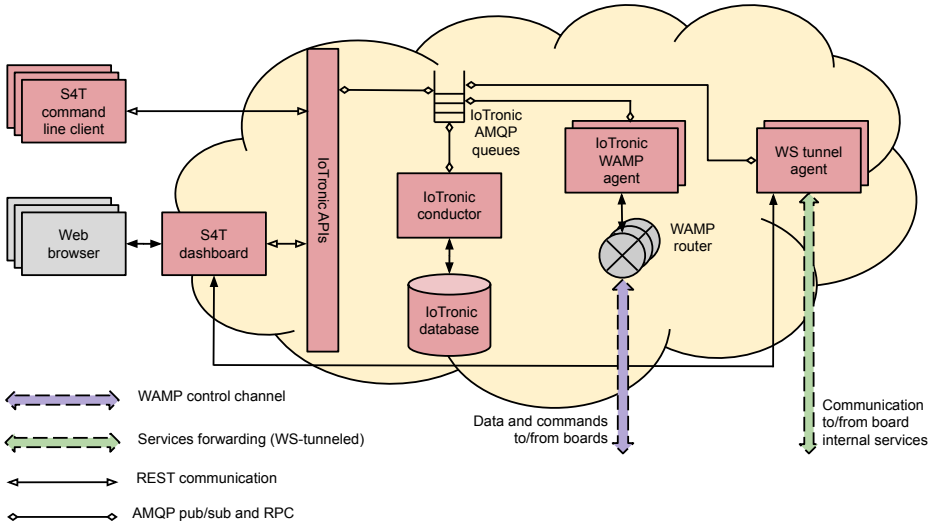


Figure 4.29: Stack4Things cloud-side architecture, illustrating the interactions between IoTronic and Lightning-Rod via WebSockets and WAMP for managing IoT resources. Source: [60]

S4T leverages the SDI/O paradigm to decouple the control and data planes, which allows for dynamic and secure management of IoT devices. This separation is crucial for environments like hotels, where different users require temporary access to IoT devices. The SDI/O paradigm enhances flexibility, enabling users to interact with IoT devices through IPAs without reconfiguring the entire IoT infrastructure each time a new user arrives.

4.2.2.2 Software-Defined Input/Output (SDI/O) Architecture

The core of the EIPA solution is the Software-Defined Input/Output (SDI/O) architecture, which facilitates the separation of control and data flows within IoT networks. Initially designed for telecommunications and later applied to data networks, SDI/O allows for more flexible management of complex infrastructures by decoupling the underlying hardware from the application logic [31]. In the context of IoT, SDI/O leverages Fog and Edge computing to dynamically manage data flows and control access to IoT devices, ensuring that IPAs can interact with these devices securely and efficiently.

Figure 4.30 illustrates the high-level architecture of the SDI/O-enabled infrastructure. The system is designed to enable seamless communication between the IPA and the IoT environment

by abstracting the complexity of the underlying IoT devices. This abstraction allows IPAs to issue commands and receive data from IoT devices without requiring deep knowledge of the specific devices or their configurations.

SDI/O provides the following key functionalities:

- **Device Virtualization:** IoT devices are virtualized as software-defined entities, allowing for dynamic reconfiguration and simplified interaction with IPAs.
- **Dynamic Resource Management:** The infrastructure dynamically allocates and configures IoT resources based on user requirements, reducing manual intervention and ensuring optimal performance.
- **Secure Access Control:** SDI/O incorporates security mechanisms that manage user access to IoT devices, ensuring only authorized users can interact with the environment.

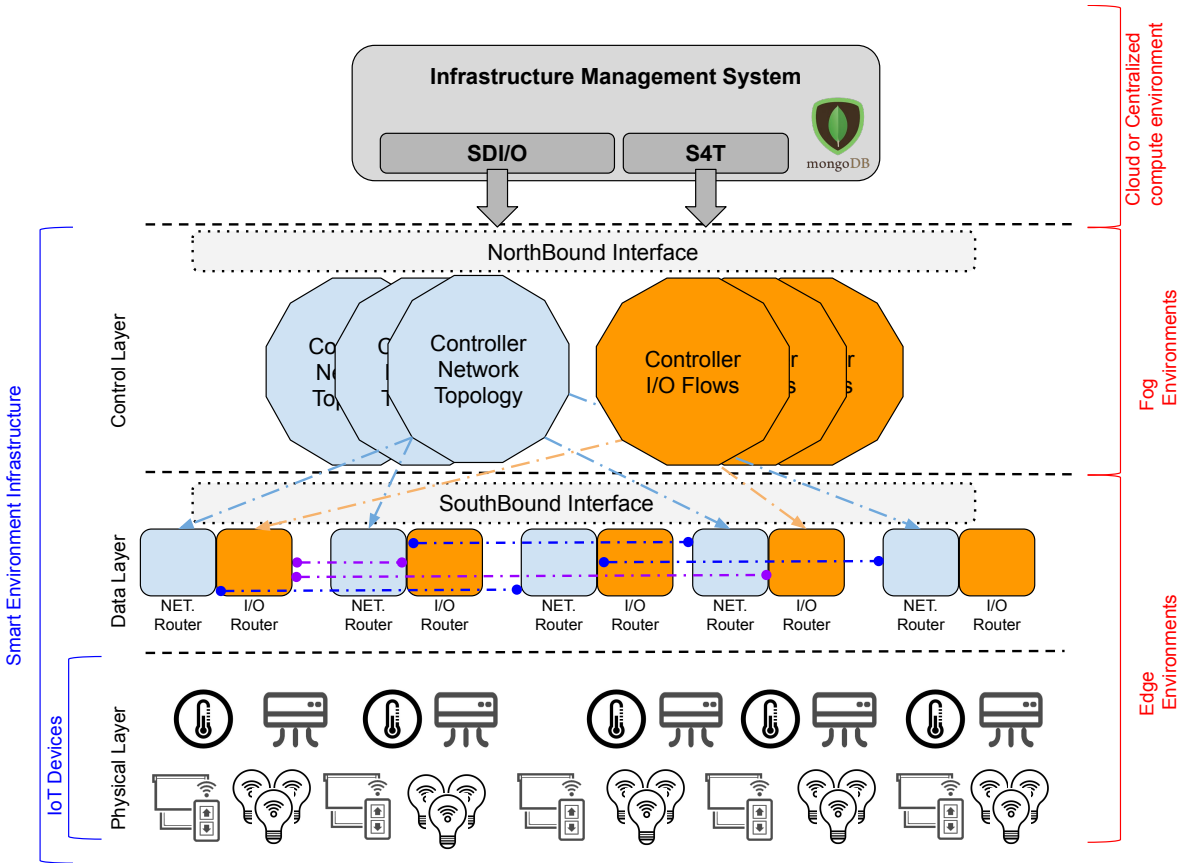


Figure 4.30: High-level architecture of an SDI/O-enabled infrastructure. Source: [60]

4.2.2.3 Everywhere IPA (EIPA): Solution Overview

The Everywhere IPA (EIPA) solution is designed to automate the configuration and management of IoT resources in dynamic environments such as hotels. EIPA leverages the SDI/O architecture to create an abstraction layer between IPAs and the underlying IoT infrastructure, simplifying

user interaction and enhancing security. EIPA comprises four key components: the SDI/O infrastructure, a Device Descriptor Creator, a Virtual Connector, and a custom IPA application.

Figure 4.31 shows the general architecture of the EIPA system in a hotel room environment. The SDI/O infrastructure enables the dynamic configuration of IoT devices, while the Device Descriptor Creator generates JavaScript Object Notation (JSON)-based descriptors that define the IoT resources. The Virtual Connector acts as an intermediary, translating IPA commands into device-specific actions, allowing users to control the environment seamlessly through their IPAs.

The two main workflows in EIPA are resource assignment and user-device interaction:

- **Resource Assignment:** This process involves generating the device descriptors, setting up the Virtual Connector, and assigning IoT resources to the user's IPA. Once completed, the custom IPA application registers the IoT devices, allowing seamless interaction.
- **User-Device Interaction:** This workflow ensures that user commands issued via the IPA are translated into appropriate device-specific actions, enabling real-time control of the IoT environment.

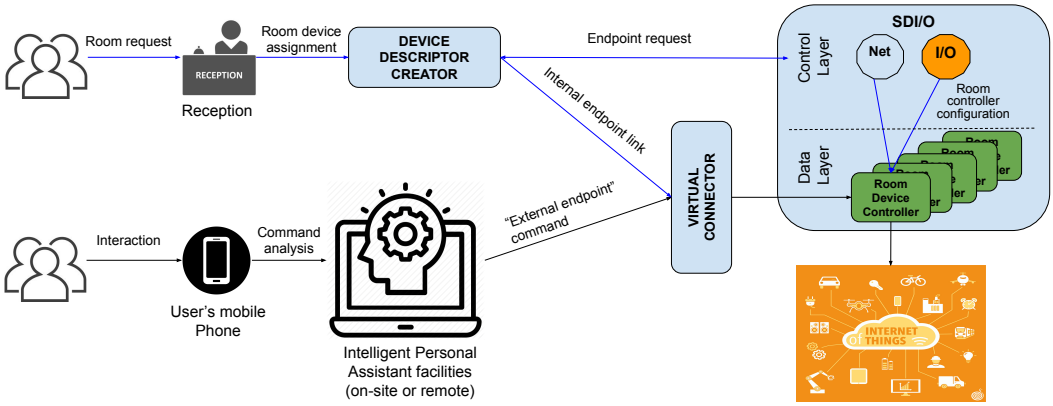


Figure 4.31: Overview of the EIPA architecture in an accommodation facility. Source: [60]

4.2.2.4 Solution Workflows: Resource Assignment and User-Device Interaction

The resource assignment workflow is illustrated in Fig. 4.32, which shows the steps in generating the resource descriptor and configuring the Virtual Connector. Once the resources are assigned, the user can interact with IoT devices via their IPA, with the Virtual Connector handling the translation of commands.

User-device interaction is depicted in Fig. 4.33, which shows how user commands are processed by the Virtual Connector and executed on the IoT devices. This process ensures that users can control the environment without understanding the underlying IoT infrastructure.

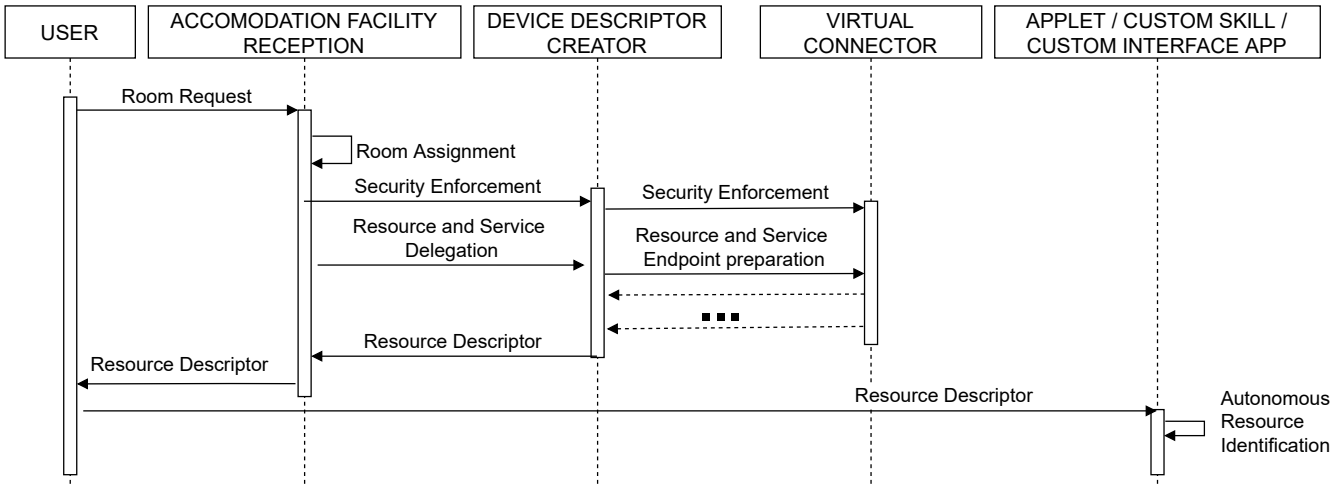


Figure 4.32: Resource assignment workflow in an accommodation facility. Source: [60]

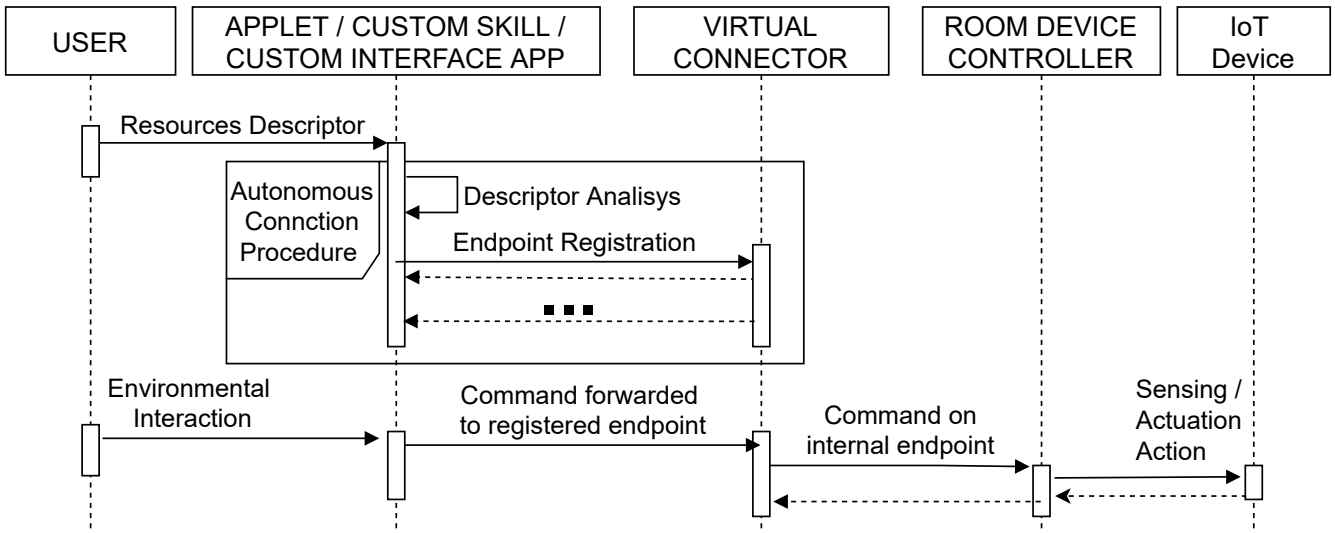


Figure 4.33: User interaction with the environment mediated by the Virtual Connector. Source: [60]

4.2.2.5 Qualitative Evaluation and Scalability Analysis

To evaluate the scalability and performance of EIPA, we conducted several experiments in environments with varying numbers of IoT devices. As shown in Fig. 4.34, configuration time increases with the number of devices, with the most time-consuming operations being the setup of I/O flows and the injection of plugins into the Virtual Connector.

Statistical analysis, shown in Fig. 4.35, demonstrates the system's ability to scale efficiently, with average flow shaping and plugin injection times remaining within acceptable limits as the complexity of the environment increases.

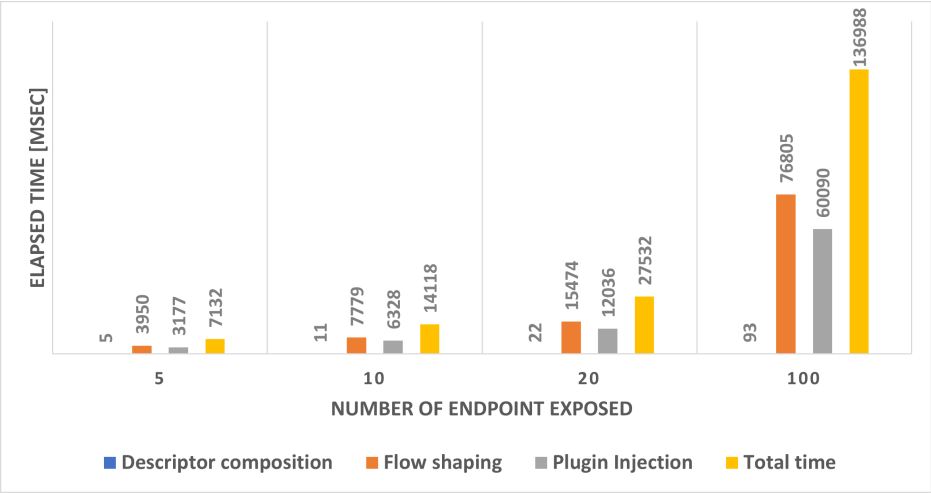


Figure 4.34: Elapsed time as a function of device number involved. Source: [60]

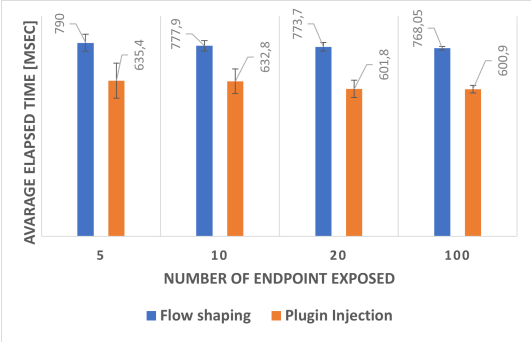


Figure 4.35: Flow Shaping and Plugin Injection average values. Source: [60]

4.2.2.6 Lessons Learned

The development and deployment of EIPA have yielded several key insights:

1. Scalability: The SDI/O paradigm enabled the EIPA system to scale efficiently across different environments, even in large-scale IoT deployments. Optimization of Virtual Connector setup times will be crucial for future iterations.

2. Device Abstraction: By abstracting IoT devices as software-defined entities, EIPA allows users to interact with these devices through IPAs without requiring detailed knowledge of the underlying infrastructure, enhancing user experience.

3. Security and Access Control: The SDI/O-based architecture provides robust security mechanisms, ensuring that only authorized users can interact with IoT devices. Further research is needed to explore security in multi-user scenarios.

4. Flexibility in Dynamic Environments: The EIPA solution proved highly flexible, adapting to changes in the environment by dynamically reconfiguring IoT resources based on user needs. This flexibility is particularly valuable in temporary access environments such as hotels.

4.2.3 Edge-Driven Data Management in Smart Cities

As the integration of Intelligent Personal Assistants and dynamic IoT management systems enhances the user experience in temporary urban environments, the scalability of such solutions becomes critical for larger and more permanent urban infrastructures. Efficient data management and processing are fundamental in ensuring the smooth operation of smart city systems.

In this regard, the edge computing infrastructure implemented in Milan's Lorenteggio district is an example. Employing OpenStack and IoTronic, integrated with the Elastic Stack (ELK) for real-time data handling, this system manages IoT resources, data collection, and processing across various applications. This section explores the technical aspects of this architecture, focusing on how it supports smart city applications such as smart lighting, environmental monitoring, and traffic management. [7].

4.2.3.1 System Architecture and Design

The architecture deployed in the Lorenteggio district leverages the integration of edge computing with cloud-based services to optimize the management of IoT infrastructure. This system is orchestrated through Stack4Things (S4T), a middleware designed to facilitate the coordination of IoT nodes at the network edge. The Elastic Stack (ELK) further enhances the system, which enables robust data processing and visualization capabilities. Figure 4.36 illustrates the overall architecture, which balances computational workloads between edge and cloud resources.

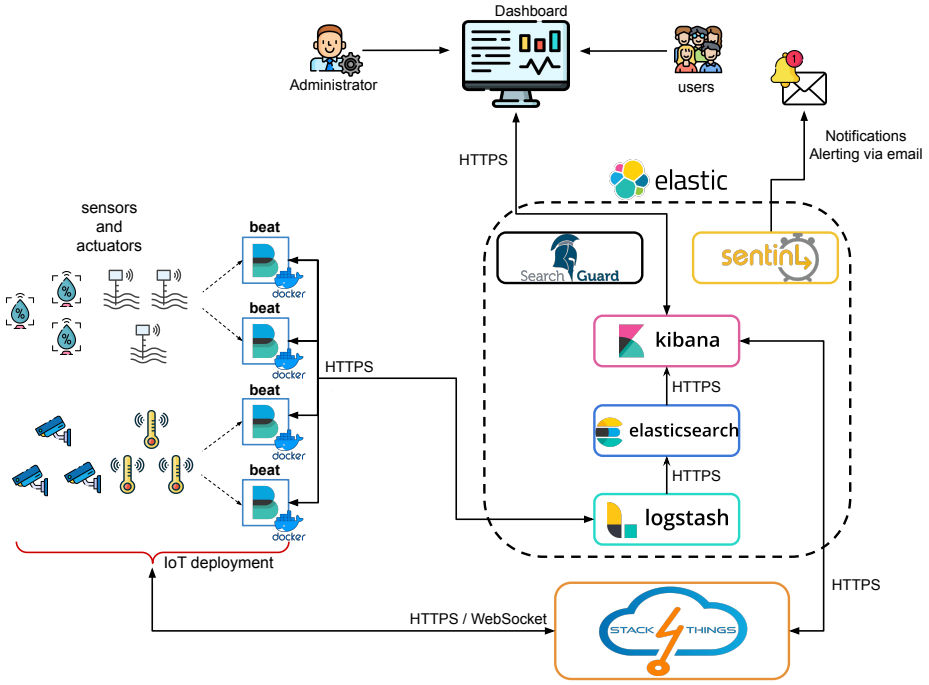


Figure 4.36: The OpenStack/Elastic Stack-powered Smart City system architecture. Source: [7]

The system is designed to offload processing tasks to edge nodes, which minimizes the need for continuous data transmission to cloud servers. Edge devices are equipped with Docker containers,

where lightweight agents (Beats) collect and preprocess sensor data before transmitting it to the cloud for more extensive analysis. This design improves latency and optimizes bandwidth usage.

4.2.3.2 Infrastructure Management with Stack4Things

The management of IoT devices at the edge of the network is governed by Stack4Things (S4T), a middleware platform that provides critical functionalities to ensure efficient control over distributed IoT devices:

- **Remote Access and Control:** S4T allows administrators to access IoT nodes remotely, independent of their network configuration. This feature is essential for smart city environments where IoT devices are widely distributed and may face varying network conditions.
- **Dynamic Reprogramming:** S4T supports the dynamic reprogramming of IoT nodes through the transmission of custom code via Web Application Messaging Protocol (WAMP) Remote Procedure Calls (RPCs). This capability ensures IoT devices can quickly adapt to new operational needs without physical intervention.
- **Containerized Deployment:** Edge devices run Docker containers managed by S4T. These containers host ELK Beats, which collects and preprocesses data. Containerization enhances scalability and security by isolating individual processes and simplifying updates and maintenance.

4.2.3.3 Data Collection and Processing with Elastic Stack

The Elastic Stack (ELK) is the backbone of the system's data processing pipeline. It consists of several components, each optimized for specific tasks in data handling:

- **Elasticsearch:** This component indexes and queries large volumes of data in real-time. Elasticsearch's ability to perform complex queries and filtering based on parameters such as geolocation and sensor type is essential for smart city operations.
- **Logstash:** As the central data processor, Logstash ingests data from multiple sources, transforms it into structured formats, and forwards it to Elasticsearch for indexing. This transformation allows for efficient data querying and storage.
- **Beats:** Lightweight agents known as Beats are deployed on edge devices. They handle the collection and forwarding of sensor data to Logstash or Elasticsearch. Each Beat operates within its own Docker container, ensuring modularity and efficiency.
- **Kibana:** Kibana provides a powerful data visualization and analysis interface. Through Kibana, administrators can monitor real-time data and create customized dashboards to track the performance of the smart city infrastructure.

Figure 4.37 illustrates the architecture of the ELK stack, emphasizing the interaction between the edge devices and centralized cloud components.

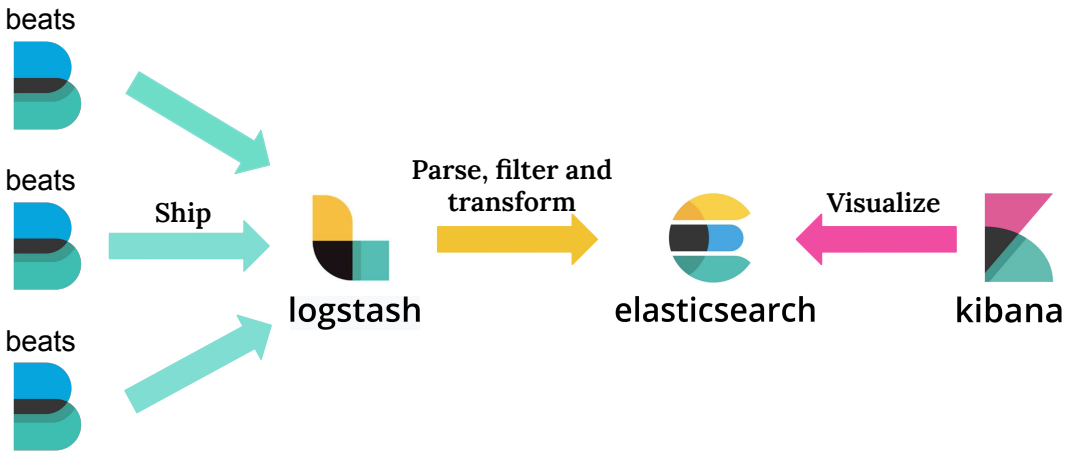


Figure 4.37: Elastic Stack (ELK) architecture. Source: [7]

4.2.3.4 Real-Time Data Processing and Applications

The system's ability to process data in real-time is critical for applications such as smart lighting, environmental monitoring, and traffic management. Below are specific use cases implemented in the Lorenteggio district:

Smart Lighting: The smart lighting infrastructure is monitored using Kibana dashboards, as shown in Figure 4.38. The dashboard provides insights into each lighting pole's power consumption, temperature, and operational status, enabling energy optimization and real-time operational adjustments.

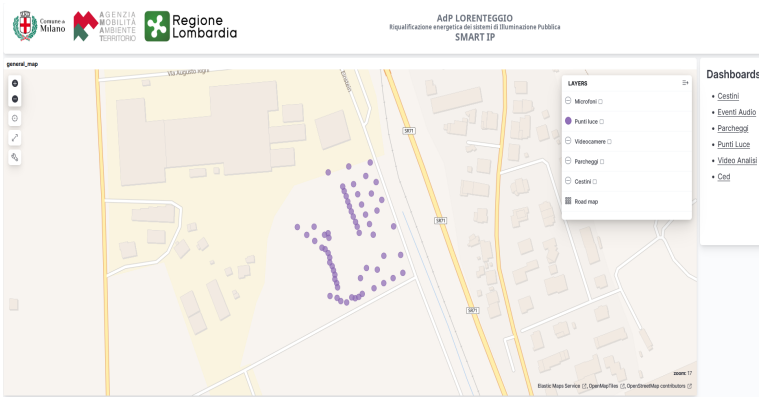


Figure 4.38: Kibana dashboard showing smart lighting poles in the Lorenteggio area. Source: [7]

Environmental Monitoring: The system collects real-time environmental data, including air quality and electromagnetic field intensity. Figure 4.39 illustrates monitoring electromagnetic field levels, providing compliance verification with safety regulations.

Traffic Management: Although not depicted in figures, the system can collect and process traffic data in real-time. This data is used to dynamically adjust traffic signals and provide optimized routing suggestions to citizens, reducing congestion and improving traffic flow.

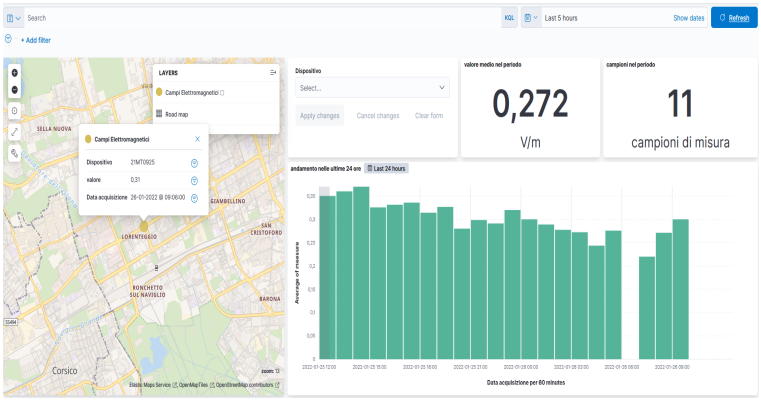


Figure 4.39: Real-time graph showing electromagnetic field intensity in the Lorenteggio district. Source: [7]

Figure 4.40: The area of the Lorenteggio district being monitored. Source: [7]

4.2.3.5 Lessons Learned

The implementation of the ELK-powered IoT architecture in the Lorenteggio district has provided several key insights into the management of smart city infrastructures:

1. Scalability: The modular nature of Docker containers combined with Elasticsearch's distributed architecture ensures that the system can scale efficiently as more IoT devices are added to the network. This scalability is crucial for expanding the smart city system across larger geographic areas.

2. Real-Time Processing: The integration of ELK with edge computing has demonstrated that real-time data processing can significantly improve operational efficiency. Applications like smart lighting and environmental monitoring benefited from immediate feedback and dynamic adjustments based on real-time data.

3. Security and Privacy: Ensuring the security of IoT systems in distributed networks is paramount. Incorporating Search Guard for secure communication and authentication between system components has proven effective in maintaining data integrity and protecting sensitive information.

4. Data Integrity and Synchronization: Synchronizing data between the edge and cloud in real-time was achieved through timestamping and synchronization protocols, ensuring that data is consistent across all components and no information is lost or duplicated during transmission.

Future research will focus on expanding the system to incorporate more IoT devices and investigating the application of predictive analytics for proactive system management and optimization in urban environments.

4.2.4 Function-as-a-Service (FaaS) on top of I/Ocloud

Building on the edge computing infrastructure implemented in Milan's Lorenteggio district, efficient data management and resource allocation are essential for the smooth operation of smart city applications. However, as the scale of IoT deployments continues to grow, more advanced computing paradigms are required to handle the increasing data and ensure low-latency services.

As smart cities evolve, integrating paradigms like Edge and Fog computing becomes crucial for managing the expanding number of IoT devices and the data they generate. These paradigms process data closer to its source, reducing latency and improving the efficiency of urban services. The Function-as-a-Service (FaaS) model offers a flexible and scalable solution for managing and orchestrating distributed IoT resources within this context.

The following section presents an advanced architecture integrating the FaaS model into Edge and Fog computing environments. This system extends the OpenStack ecosystem by incorporating edge-level function execution, leveraging components such as Zun, Qinling, and IoTronic to provide an efficient, scalable, and secure solution for IoT environments.

4.2.4.1 System Architecture

The architecture, depicted in Figure 4.41, consists of two primary layers: the *Cloud Layer*, responsible for service orchestration and function management, and the *Edge/Fog Layer*, where IoT devices execute containerized functions. The cloud side manages function scheduling, resource allocation, and the orchestration of distributed IoT nodes. Meanwhile, the edge side consists of IoT devices that execute functions within a resource-constrained environment.

The **Cloud Layer** is responsible for centralized management and high-level coordination of IoT resources. It ensures resource scheduling and allocation through Qinling and oversees the orchestration of containers through Zun. This layer also integrates IoTronic for secure and efficient communication between the cloud and edge devices, establishing WebSocket tunnels to handle devices behind Network address translations (NATs) or firewalls.

The **Edge/Fog Layer** consists of IoT devices that run containerized functions near the data producers. These devices operate under constraints, including limited computational power, memory, and network capabilities. By distributing computing tasks to the edge, the architecture reduces latency and minimizes data transmission to the cloud, optimizing network usage. This layer executes functions on edge devices in real-time, offloading tasks from the cloud to improve response times.

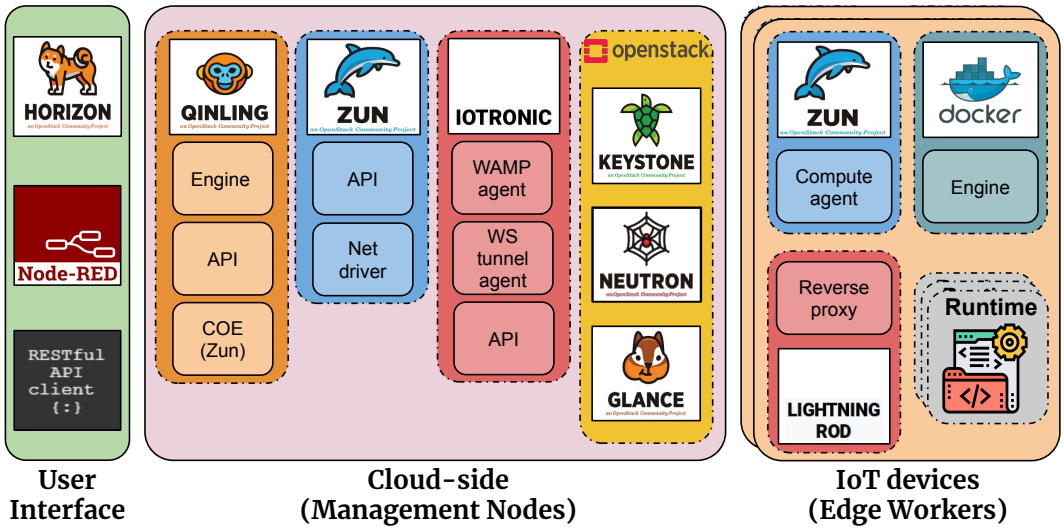


Figure 4.41: System architecture for FaaS in Edge/Fog computing environments. Source: [41]

The key components of the system are:

- **Qinling**: This OpenStack component provides FaaS capabilities, managing the life cycle of functions from creation to execution and termination. It abstracts the underlying infrastructure, offering developers a seamless interface for deploying and managing functions across a distributed environment. Qinling integrates with Zun for container orchestration, ensuring scalable and flexible function execution by dynamically allocating resources based on demand and system load.
- **Zun**: Zun acts as the container orchestration engine within OpenStack and is responsible for creating, deploying, and managing containers at the edge. It provides key services such as container scheduling, network configuration, and resource management, enabling containers to run on resource-constrained IoT devices. Zun coordinates the placement and scaling of containers to ensure optimal resource utilization and performance.
- **IoTronic**: IoTronic is a middleware that facilitates seamless communication between the cloud and IoT devices. It overcomes common network limitations like NAT traversal using WebSocket-based tunnels, allowing IoT services to be securely exposed to the cloud. This component enables real-time interaction between the cloud and distributed IoT nodes, supporting efficient deployment and management of edge-based services.

Integrating these components creates a distributed FaaS environment capable of dynamically deploying containerized functions at the network edge. This approach significantly reduces latency by executing tasks closer to the data sources, improving scalability by dynamically managing resources at the edge, and optimizing resource allocation to ensure efficient operation in IoT infrastructures with limited processing power.

4.2.4.2 Container Management and Networking with Zun and IoTronic

Zun and IoTronic manage container orchestration and networking for edge environments, where IoT devices typically reside behind NATs or firewalls. Zun is responsible for container deployment and

lifecycle management, including instantiating runtimes for function execution. IoTronic handles networking, establishing WebSocket tunnels to ensure communication between the cloud and IoT devices.

The container lifecycle begins when Zun initiates the creation of a containerized runtime on a designated IoT device. This process includes generating a unique runtime ID and exposing the container to the cloud through IoTronic's networking layer. IoTronic's WebSocket-based architecture allows seamless interaction between the cloud and edge, ensuring secure communication between distributed nodes and the cloud management layer. The container is accessible via a public IP and port on the cloud side, allowing function execution requests to be routed efficiently to the correct container on the edge.

Zun employs a custom scheduling algorithm that includes the `HostnameFilter`, which allows containers to be deployed on specific IoT nodes based on hardware characteristics, resource availability, and other constraints. This ensures the functions are executed on the most appropriate device, optimizing resource usage and performance. Additionally, IoTronic provides a reverse proxy mechanism that maps the public IP and port on the cloud to the local network of the IoT device, ensuring secure and efficient communication between the cloud and edge layers. This approach guarantees that even devices behind firewalls or NATs are reachable, maintaining connectivity across distributed networks.

4.2.4.3 Function Execution Workflow

The function execution process starts when a request is submitted to the Qinling service, which interacts with Zun to retrieve runtime metadata, including the runtime ID and networking details. This process is initiated when the user submits a function execution request to the Qinling-API. The Qinling-API identifies the target runtime and forwards the request through a WebSocket (WS) tunnel to the corresponding IoT device. The runtime ID plays a crucial role in identifying the correct execution environment on the edge device, ensuring that the function is processed by the appropriate container, as depicted in Figure 4.42.

Upon reaching the IoT device, IoTronic's reverse proxy routes the request to the appropriate containerized runtime. The function is executed within the container, and the results are transmitted back to the cloud via the same WS tunnel. This architecture facilitates low-latency function execution by offloading computational tasks from the cloud to the edge. Additionally, the containerized environments ensure isolation and security for each function, preventing resource contention and guaranteeing execution in a controlled environment. A more detailed representation of this workflow is presented in Figure 4.44.

The following describes two critical workflows the FaaS employs for the I/Ocloud framework. The first workflow outlines the actions taken when a user requests the creation of a runtime on a specific IoT device. The second workflow details the process when a function is requested to be executed on an already created runtime. A simplified version of these workflows is presented to enhance clarity, excluding some default OpenStack procedures, such as Keystone authentication and authorization tasks, and OpenStack components like Etcd for information storage and RabbitMQ for event dispatching. This analysis focuses primarily on the interactions between Qinling, Zun, and IoTronic, omitting IoT registration phases, which are assumed to have been completed. Figure 4.42 provides a detailed comprehensive directed acyclic graph illustrating these two workflows.

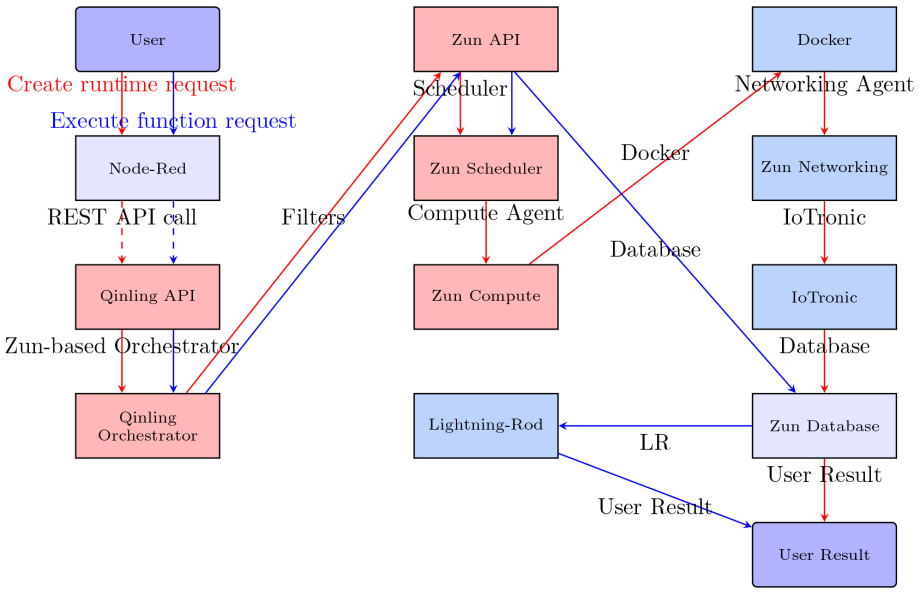


Figure 4.42: Runtime creation and function execution workflows. Source: [41]

4.2.4.4 Runtime Instantiation Workflow

In a FaaS system, function execution requires an appropriate runtime to be instantiated on the device designated to execute the function. Figure 4.42 and the activity diagram in Figure 4.43 provide a graphical overview of this process. The workflow aims to instantiate a runtime on a device deployed at the network edge.

Initially, the user requests to deploy a runtime on a specific remote node through the dashboard or Command Line Interface (CLI). The dashboard/CLI sends a REST request to the Qinling-API server. Qinling-API forwards the request to the Qinling-orchestrator driver, which communicates with Zun. The orchestrator then sends a request to the Zun-API server, specifying the *nodeName* attribute, which indicates the device's hostname where the runtime should be instantiated. The Zun-API server passes this request to the Zun scheduler, which is responsible for identifying the IoT device where the runtime will be created.

The Zun scheduler applies the *HostnameFilter* to select the appropriate host and forwards a request to the Zun-Compute agent on that host to create a container called a *capsule*. Once the Zun-Compute agent receives the request, it sends an HTTP request to the local Docker engine to create the necessary containers. After the containers are created, the Zun-Compute agent requests the Zun networking driver, which utilizes IoTronic functionalities, to expose the capsule to the users. IoTronic exposes the capsule through a public IP address associated with a specific port, creating a WS tunnel between the IoT device and the cloud. This tunnel allows any request received on the cloud's IP address and port to be forwarded to the IoT device, targeting the appropriate container for function execution.

The Zun networking driver returns the cloud IP address and port to the Zun database, which stores the runtime metadata. The Zun-Compute agent notifies the Zun scheduler of the operation's status, and the runtime ID is stored in the Qinling database, enabling future function executions. This workflow is depicted graphically in Figure 4.43.

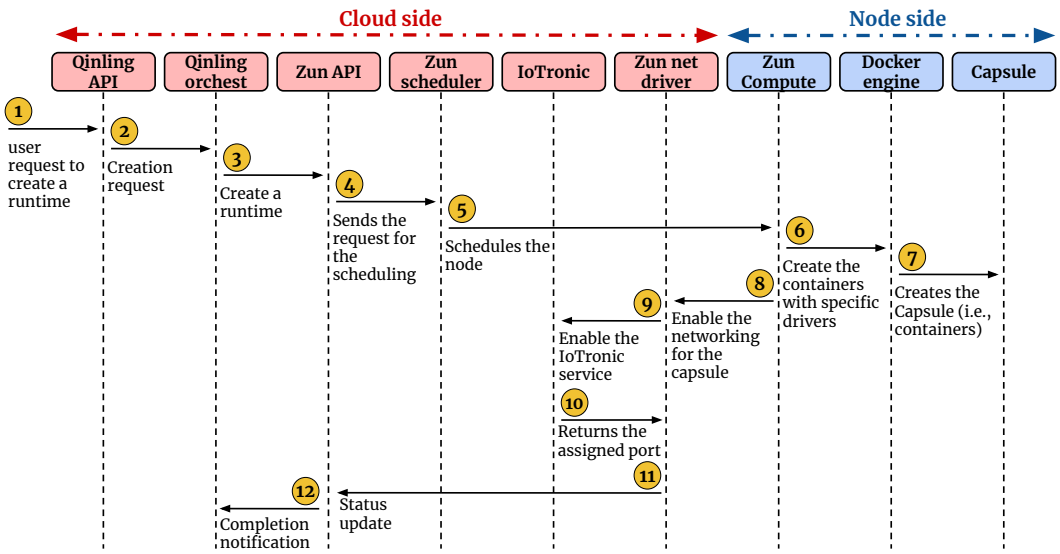


Figure 4.43: Runtime instantiation workflow. Source: [41]

4.2.4.5 Function Execution Workflow

The function execution workflow, which operates on a specific runtime deployed on an edge node, is illustrated in Figure 4.42 and the activity diagram in Figure 4.44. Before execution, it is assumed that the IoT device is registered to the cloud, the user has written the function, and the runtime has already been deployed on the edge device.

The user sends a function execution request to a specific IoT node through the dashboard or CLI. The request is processed by the Qinling-API, which retrieves the runtime ID from its database. The Qinling orchestrator then contacts Zun via the Zun-API server to obtain the runtime metadata, including the IP address and port. Once the metadata is received, the orchestrator forwards the execution request through the WS tunnel to the IoT device, where IoTronic's reverse proxy routes the request to the correct runtime based on the runtime ID.

After the function is executed on the runtime, the results are sent back through the WS tunnel to the Qinling orchestrator, which returns the result to the Qinling-API server. Finally, the function execution result is available to the user through the dashboard or CLI. This workflow, including the communication and execution process, is visually represented in Figure 4.44.

The same workflow is used when the function has already been executed at least once. In cases where the function is invoked for the first time, additional operations are required. When the request reaches the IoT device, the container retrieves the necessary packages from the cloud and stores them on the persistent volume, allowing the function to be loaded and executed by the runtime container.

4.2.4.6 Performance and Scalability Analysis

A series of tests measured response times and failure rates under varying concurrency levels to evaluate the system's performance and scalability. The system demonstrated its ability to scale the number of containers based on load dynamically, optimizing resource usage on IoT devices. As shown in Figure 4.45, the system maintains consistent response times across different levels of

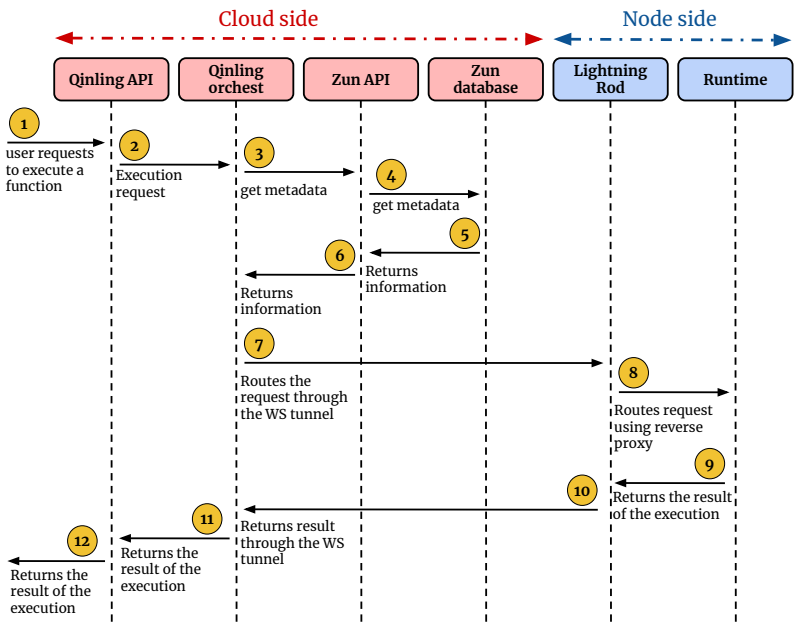


Figure 4.44: A function execution workflow.

concurrency, indicating effective load balancing and resource allocation. The system dynamically provisions additional containers to handle increased demand, preventing performance degradation during peak loads.

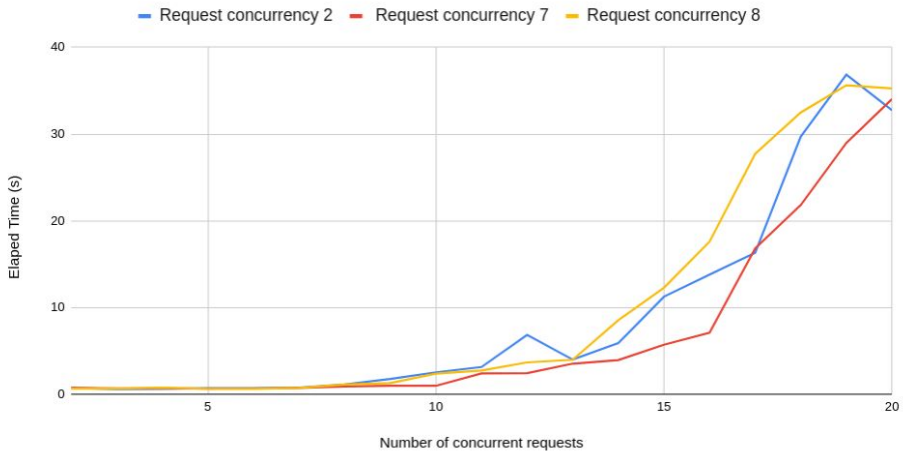


Figure 4.45: Response times for function execution under different concurrency configurations. Source: [41]

Figure 4.46 shows the failure rates for function execution under different levels of concurrency, providing insights into system robustness. Despite increasing workloads, the system maintained low failure rates, demonstrating its ability to scale and maintain reliability in resource-constrained environments.

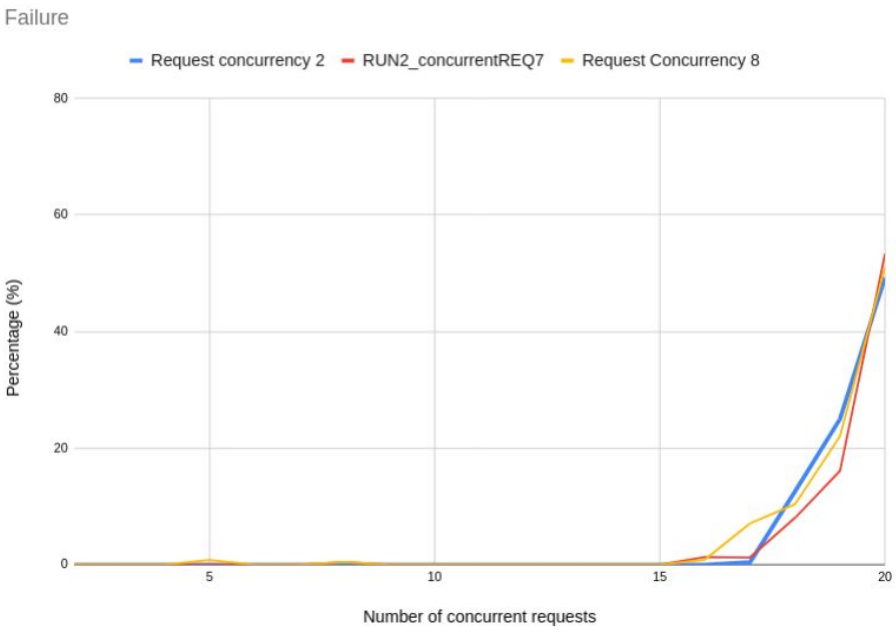


Figure 4.46: Failure rates for function execution under different concurrency levels. Source: [41]

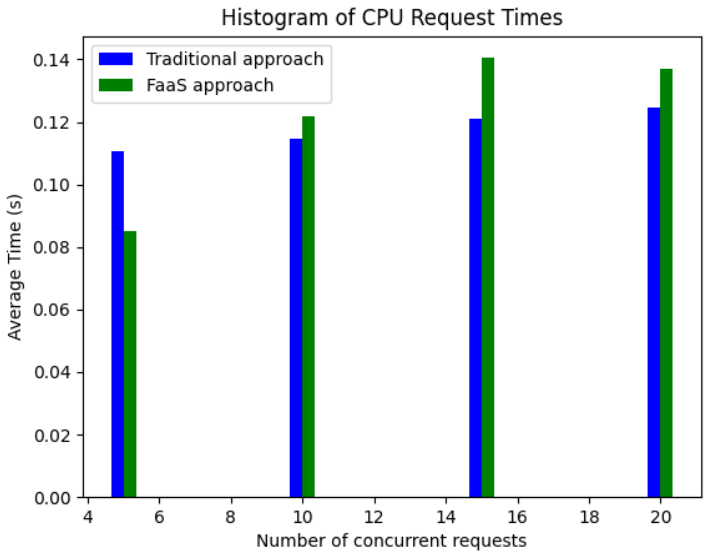


Figure 4.47: Performance comparison between traditional HTTP server and FaaS platform. Source: [41]

A comparison with a traditional HTTP-based server model was conducted to assess the overhead introduced by the FaaS platform. As shown in Figure 4.47, the FaaS platform's performance remained comparable to that of a traditional server model, with only a minimal increase in response time under high concurrency. This demonstrates that the overhead introduced by the FaaS

platform is minimal, making it well-suited for latency-sensitive IoT applications. The system's ability to dynamically manage resources while maintaining low-latency execution makes it highly effective for distributed IoT environments.

4.2.4.7 Lessons Learned

The deployment of the FaaS architecture in Edge and Fog computing environments provided several key insights:

- **Dynamic Container Orchestration**: The integration of Zun and Qinling enabled the dynamic scaling of containerized functions based on real-time system load, optimizing resource allocation and improving scalability in resource-constrained IoT environments.
- **Low-Latency Communication**: IoTronic's WebSocket-based communication facilitated seamless interaction between the cloud and IoT devices, reducing latency even when devices were deployed behind NATs or firewalls.
- **Efficient Resource Management**: The architecture distributed computational tasks across the network efficiently, utilizing IoT devices at the edge to offload the cloud, thus enhancing overall system performance.
- **Minimal Overhead**: Despite the additional layers introduced by the FaaS platform, the system maintained performance levels close to traditional server-based models, making it highly suitable for latency-sensitive IoT applications.

This architecture showcases the potential of deploying FaaS services in distributed IoT infrastructures, offering a scalable, flexible, and efficient solution for managing and executing functions at the network edge.

4.2.5 Self-Conscious Cyber-Physical Systems for Sustainable Energy Management in Communities

The growing complexity of urban infrastructures demands advanced computing paradigms for managing IoT devices and optimizing energy resources in real-time. Technologies like Edge and Fog computing are essential for handling the increasing data traffic generated by IoT systems while simultaneously improving the efficiency of energy management frameworks, particularly in Renewable Energy Communities (RECs).

Incorporating deep learning-driven Distributed Cyber-Physical Systems (DCPS) within these energy management frameworks allows for the dynamic balancing of energy production and consumption. The proposed DCPS architecture autonomously manages energy flows in RECs by utilizing edge and cloud computing resources, optimizing resource allocation, and reducing reliance on external power grids. This energy-aware approach aligns with the goals of sustainable urban development, reinforcing the resilience and scalability of smart city infrastructures.

The next section explores how the Function-as-a-Service (FaaS) model, integrated into Edge and Fog computing environments, further enhances the management of distributed IoT resources. This model provides a flexible, scalable solution that ensures low-latency response times and efficient resource utilization across smart city applications. [16].

4.2.5.1 System Architecture

The proposed architecture for the energy-aware DCPS comprises three primary components: (1) Renewable Energy Production Sites, (2) Real Estate Units acting as sub-DCPS, and (3) Cloud-based centralized computation facilities. These components ensure optimized energy production and consumption, particularly in smart grids. The structure is illustrated in Figure 4.48.

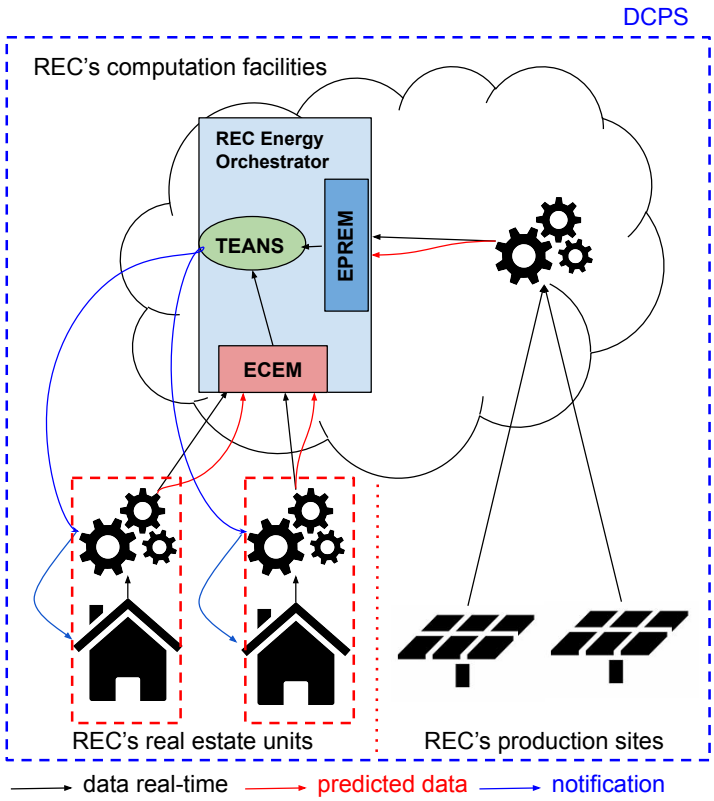


Figure 4.48: System architecture of the Renewable Energy Community (REC). ECEM: Energy Consumption Estimation Module, EPREM: Energy Production Estimation Module, TEANS: Threshold Evaluator and Notification System. Source: [16]

REC Production Sites: Energy generation occurs primarily at these sites, relying on renewable sources such as solar panels. Data regarding energy production is continuously collected via IoT sensors, with EPREM (Energy Production Estimation Module) analyzing historical data in conjunction with real-time inputs like solar irradiance and weather conditions. This allows for predictive energy generation, ensuring energy availability is sustained across different environmental conditions. EPREM dynamically adjusts energy distribution based on the predicted production, helping maintain system stability.

Real Estate Units: These units act as sub-DCPSs, each equipped with smart meters and IoT devices to monitor energy consumption. ECEM (Energy Consumption Estimation Module) operates locally within each unit, using deep learning models to predict short- and long-term consumption trends. The data collected from these units is encrypted and transmitted securely to the cloud, where it is aggregated and processed for optimization purposes. Real-time feedback

ensures the consumption does not exceed production, balancing the energy grid.

Cloud-based Computation Facilities: The cloud infrastructure serves as the computational core of the architecture and is responsible for real-time processing, storage, and management of data from both REC production sites and real estate units. The cloud hosts ECEM and EPREM models and enables large-scale data analysis, deploying machine learning algorithms to predict energy trends. The Threshold Evaluator and Notification System (TEANS), also hosted on the cloud, continuously evaluates whether predicted consumption surpasses production. When a shortfall is predicted, TEANS sends notifications to users, prompting energy-saving actions. Figure 4.49 demonstrates the energy estimation process that ensures equilibrium between energy production and consumption.

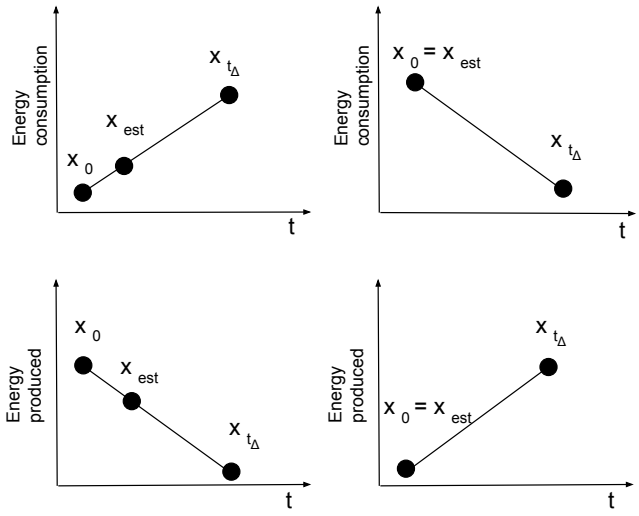


Figure 4.49: Energy estimation values for both consumption and production. The cloud ensures that consumption aligns with available production. Source: [16]

4.2.5.2 Deep Learning Models for Energy Prediction

The architecture leverages two key deep learning models: one for predicting energy consumption (ECEM) and the other for forecasting energy production (EPREM). These models, trained on historical and real-time data, enable the system to manage energy resources effectively.

Energy Consumption Estimation Module (ECEM): The ECEM model is built using a Bidirectional Long Short-Term Memory (BiLSTM) network, chosen for its capability to handle complex time-series data. BiLSTM allows the model to account for past and future energy consumption trends, providing more accurate predictions than standard LSTM networks. Input features include historical consumption data, temperature, time of day, and occupancy, which are combined to predict short-term and long-term energy demands. The model is evaluated using metrics such as Root Mean Square Error (RMSE) and Mean Absolute Error (MAE), ensuring precision in consumption prediction.

Energy Production Estimation Module (EPREM): For energy production forecasting, an LSTM network is employed to model the non-linear relationships in time-series data such as solar irradiance, cloud cover, and temperature. This model allows for precise forecasting of energy generation based on renewable sources. The EPREM model is instrumental in ensuring that

production levels are maximized during peak sunlight hours while anticipating fluctuations due to weather changes. Table 4.6 summarizes the performance of both models.

Table 4.6: Performance metrics for energy consumption (BiLSTM) and production (LSTM) models.

Model	**Feature**	**RMSE**	**MAE**
BiLSTM (Consumption)	GAP	0.457	0.299
LSTM (Production)	DC Power	909.19	509.53
LSTM (Production)	Efficiency	0.011	0.002

Both models were trained using real-world datasets, including household electricity consumption and solar power generation data. The data underwent extensive pre-processing to ensure high model performance, including normalization, interpolation, and denoising. Real-time data updates from smart meters and sensors allow the models to refine predictions dynamically.

4.2.5.3 Cloud Infrastructure and Orchestration

The cloud infrastructure is the central hub for orchestrating the DCPS's energy management functions. Key tasks performed by the cloud include data aggregation, model deployment, and energy distribution optimization:

1. Data Aggregation and Processing: The cloud aggregates data from IoT sensors, smart meters, and renewable energy sources in real-time. Distributed computing techniques are employed to ensure low-latency processing, handling the substantial data flow from multiple production sites and residential units. Advanced analytics are applied to this data to detect patterns and identify anomalies in both consumption and production.

2. Machine Learning Model Management: The cloud enables the deployment and continuous retraining of ECEM and EPREM. This allows the system to adapt to seasonal variations and evolving energy trends within the REC. Efficient model deployment mechanisms ensure scalability, with additional computing resources allocated dynamically based on the volume of data processed.

3. Orchestration and Decision-Making: The TEANS module continuously evaluates the balance between energy supply and demand within the REC. It issues notifications to users when energy consumption exceeds production, urging them to reduce their consumption. The cloud also facilitates automated control of energy distribution across the REC, ensuring optimal resource allocation and minimal dependency on external grids.

4. Scalability and Flexibility: The architecture is designed to scale seamlessly with the growth of RECs. As more production sites and real estate units are integrated, the cloud dynamically scales its processing capabilities to handle the increased data flow. Historical data stored in the cloud supports the continuous improvement of machine learning models and the development of more sophisticated predictive algorithms.

4.2.5.4 Energy Consumption and Production Estimation Modules

Two critical components of the architecture are the Energy Consumption Estimation Module (ECEM) and the Energy Production Estimation Module (EPREM). These modules utilize deep

learning models to predict energy consumption and production. The predictions are based on both real-time data and historical patterns, allowing the system to anticipate energy needs and optimize resource allocation accordingly.

Energy Consumption Estimation Module (ECEM): The ECEM is tasked with predicting the energy consumption of real estate units within the REC. The module forecasts short-term and long-term energy demands by leveraging historical consumption data and real-time inputs. It ensures that energy consumption is optimized and any excess demand is managed by notifying users to adjust their consumption patterns in real-time.

Energy Production Estimation Module (EPREM): The EPREM predicts energy production from the REC's renewable energy sources. Factors such as weather conditions, solar irradiance, and the operational status of energy production facilities are considered. By forecasting the energy generated, the EPREM ensures that energy supply aligns with community demand.

The data generated by these modules are processed by the Threshold Evaluator and Notification System (TEANS), which assesses whether the predicted energy supply will meet the community's demand. If a shortfall is predicted, TEANS proactively sends notifications to relevant members of the REC, requesting them to reduce their energy consumption. This strategy helps prevent the need for external energy purchases, thus promoting self-sufficiency within the community.

$$x_{(est)} = x_{(t_0)} + \frac{\left(x_{(t_0)} - x_{t_\Delta}\right)}{(t_0 - t_\Delta)} * t_{est} - \frac{\left(x_{(t_0)} - x_{t_\Delta)}\right)}{(t_0 - t_\Delta)} \tag{4.1}$$

The equation above is used in the estimation process, where $x_{(est)}$ represents the estimated energy consumption or production, $x_{(t_0)}$ is the real-time data point, and x_{t_Δ} is the predicted data point for a future time. This equation helps ensure that the system's predictions are as accurate as possible, facilitating better decision-making within the REC.

4.2.5.5 Deep Learning Models

This study employs two types of deep learning models: Long-Short-Term Memory (LSTM) networks and Bidirectional LSTM (BiLSTM) networks. These models are specifically designed to handle time-series data, which is crucial for predicting energy consumption and production patterns in RECs.

LSTM Networks: LSTM networks are a type of recurrent neural network (RNN) particularly well-suited for time-series prediction tasks. They can learn long-term dependencies in sequential data, making them ideal for predicting energy production, which often follows predictable seasonal and daily patterns.

BiLSTM Networks: BiLSTM networks extend LSTM networks by processing input data in both forward and backward directions. This bidirectional processing allows the model to capture dependencies from past and future data points, improving prediction accuracy, particularly for complex energy consumption patterns that may exhibit variability.

The deep learning models were trained using real-world datasets, including household electricity consumption and solar power generation data. These models were evaluated using standard metrics such as Root Mean Square Error (RMSE) and Mean Absolute Error (MAE), ensuring robust and accurate predictions.

The results indicate that the deep learning models provide accurate predictions, which are crucial for optimizing energy management within the REC. The models' ability to forecast energy

consumption and production ensures that the community can make informed decisions about energy usage, minimizing reliance on external energy sources.

4.2.5.6 Use Case: Renewable Energy Community

The system was validated through a use case involving a Renewable Energy Community consisting of multiple real estate units and solar power production sites. Each unit was equipped with IoT devices and smart meters to monitor energy consumption at a granular level. The energy production sites were connected to inverters that measured DC (Direct Current) and AC (Alternating Current) power outputs, which were then processed by the deep learning models for prediction.

Dataset Description: The use case utilized two public datasets: the household electricity load diagrams dataset and the solar power generation dataset. The electricity dataset provided detailed data on household energy consumption, while the solar dataset offered insights into energy production at solar power plants. These datasets underwent extensive preprocessing, including interpolation, normalization, and integration, to ensure their suitability for the deep learning models.

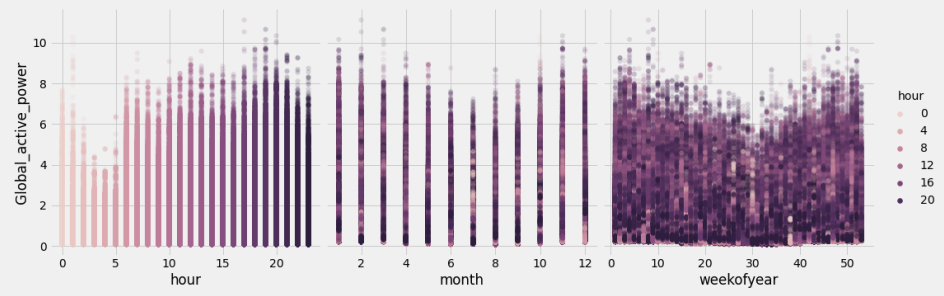


Figure 4.50: Gap seasonality of a real estate unit by hour, month, and week of the year.

Figure 4.50 highlights the seasonality of the Global Active Power (GAP) feature, which was used as the output value for the BiLSTM model's predictions. Analyzing the data reveals that higher GAP values are concentrated in specific periods, emphasizing the cyclical nature of consumption patterns. This cyclical behavior facilitates the identification of intervals characterized by increased energy usage. In the figure, the data points are displayed using a color gradient that ranges from light to dark, representing a 24-hour time frame, with color intensity corresponding to the time of day. These data are further organized into hourly, monthly, and weekly segments throughout the year.

The hourly analysis indicates that energy consumption peaks predominantly occur in the afternoon, particularly between 17:00 and midnight. Meanwhile, the monthly overview reveals a significant decrease in energy usage during summer, likely due to longer daylight hours. There is also a strong correlation between the monthly and weekly charts, which show reduced average consumption in their central regions, corresponding to the summer period.

This seasonality chart aids in predicting consumption patterns over various time frames throughout the year, which is critical for optimizing the configuration of the proposed system. Specifically, it enables more accurate forecasting of the most appropriate periods or hours to send notifications to users, encouraging them to reduce their energy consumption in line with Renewable Energy Communities' (REC) energy efficiency and sustainability policies.

The prediction results from the BiLSTM model for energy consumption and the LSTM model for energy production demonstrated the models' capability to forecast future energy needs accurately. This accuracy is critical for ensuring the REC can manage its energy resources effectively, reduce dependence on external energy providers, and support sustainability goals.

4.2.5.7 Lessons Learned

Several key lessons were derived from the design, implementation, and deployment of the energy-aware DCPS:

1. Model Scalability and Accuracy: While BiLSTM and LSTM models demonstrated high accuracy in energy forecasting, larger RECs posed scalability challenges. Edge computing techniques, such as local model inference, could be explored to offload some of the cloud's computational tasks, thereby improving responsiveness.

2. Real-Time Integration: Real-time energy management requires integrating low-latency predictive models with high-frequency data streams. The current architecture highlighted the importance of optimizing data pipelines to reduce latency in decision-making, especially for energy-critical applications.

3. Environmental Adaptation: Although the system successfully adapted to gradual environmental changes, sudden fluctuations (e.g., abrupt weather shifts) required faster model retraining cycles. Future enhancements should include real-time retraining capabilities for the EPREM model to improve responsiveness to sudden changes in renewable energy production.

4. Edge-Cloud Synergy: Integrating edge computing capabilities could improve the architecture's reliance on cloud infrastructure for model inference and data processing. Edge devices can handle time-sensitive tasks, reducing the load on the cloud and enhancing system scalability and resilience.

4.2.6 Innovative Urban Intelligence Framework with OpenStack and IoT Integration

Integrating advanced computing paradigms is crucial in optimizing various other urban services, including energy management systems. Beyond energy, technologies like Edge and Fog computing facilitate real-time data processing, enhancing the efficiency and responsiveness of smart city applications.

Urban intelligence systems take this a step further by incorporating both cloud and edge computing technologies to manage and optimize a wide range of urban services. The following subsection introduces a comprehensive architecture that leverages OpenStack and IoTronic to manage complex workflows and improve the security of smart city infrastructures. This open-source platform supports the continuous evolution and optimization of services, aligning with the broader goals of creating flexible and resilient urban environments [61].

4.2.6.1 System Architecture and Design

The architecture proposed in this research is designed to address the growing need for efficient, scalable, and secure urban intelligence systems. In light of rapid urbanization, cities require robust infrastructures capable of managing large-scale Cyber-Physical Systems (CPS) and supporting

data-driven decision-making across various smart city applications. The architecture leverages OpenStack and edge computing to integrate cloud services with real-time processing at the network edge, ensuring flexibility and minimizing latency in critical services. Figure 4.51 shows the high-level architecture.

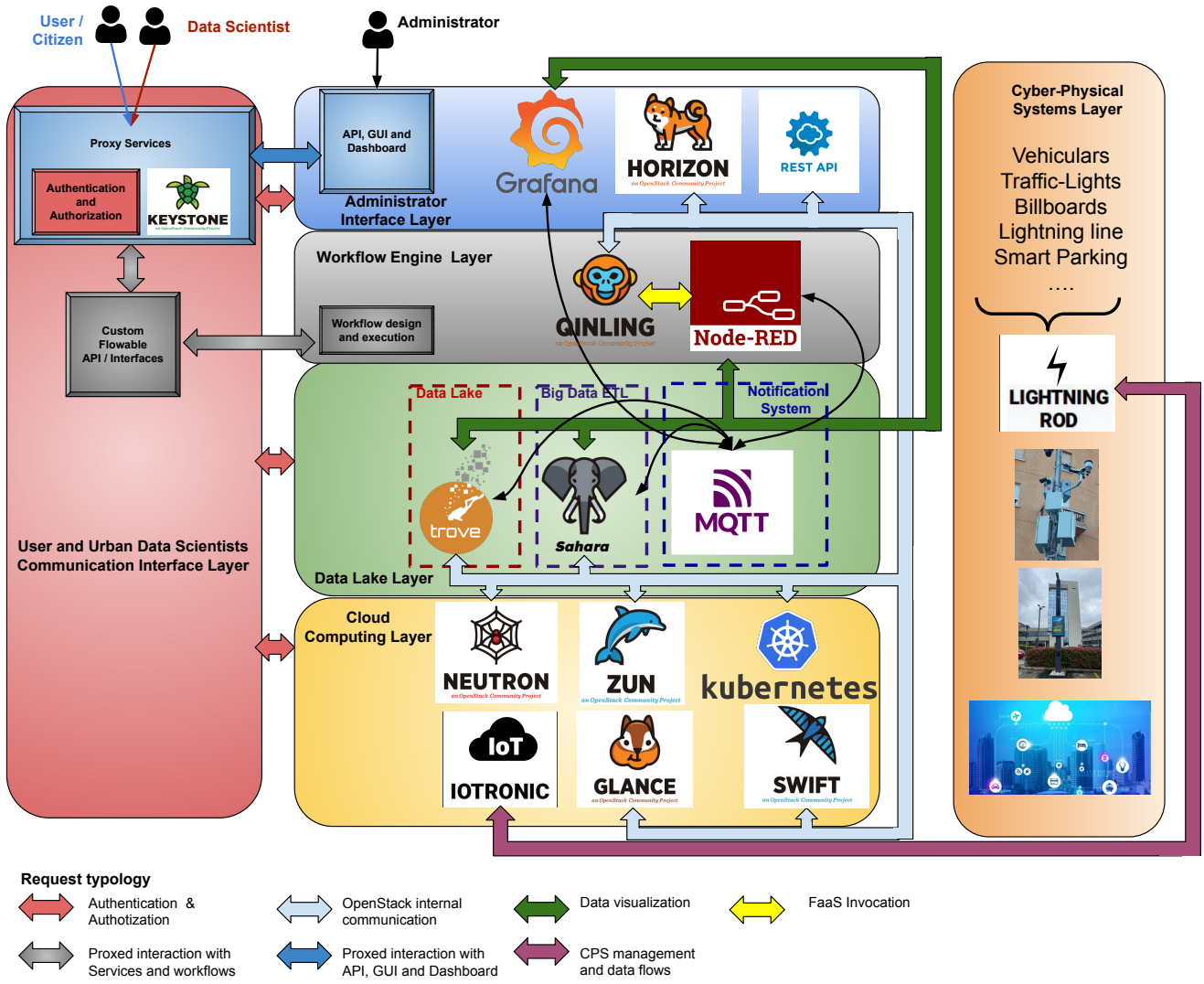


Figure 4.51: The OpenStack-based Urban Intelligence Architecture. Source: [61]

This architecture introduces several innovations, such as directly integrating Function-as-a-Service (FaaS) paradigms on IoT devices, improving security through a multi-layered approach, and reducing the external attack surface. Each architecture layer is carefully designed to handle different aspects of urban intelligence, from data acquisition at the edge to cloud-based processing and analytics.

4.2.6.2 Cyber-Physical Systems Layer

At the foundation of the architecture lies the Cyber-Physical Systems (CPS) Layer, where physical devices, such as sensors, actuators, and smart systems, interact with computational elements. These devices are managed through the **Lightning-Rod (LR)** software component, which is deployed on the IoT devices and is responsible for data acquisition and local processing at the network edge.

The LR communicates with the cloud-based **IoTronic** service through the Web Application Messaging Protocol (WAMP), ensuring efficient and reliable bi-directional communication between the devices and the cloud. This real-time data flow allows dynamic control of CPS resources, enabling responsive applications critical for smart city infrastructure.

4.2.6.3 Cloud Computing Layer

The Cloud Computing Layer provides the necessary infrastructure for managing urban services, large-scale data processing, and computational resources. This layer is powered by OpenStack services, which provide container orchestration, virtual networking, and IoT device management. Key components include:

- **Zun and Kubernetes:** These services enable container orchestration and management. Zun provides native container support within OpenStack, while Kubernetes orchestrates and scales containerized applications across the cloud infrastructure.
- **Neutron:** This component handles virtual networking, ensuring secure and efficient communication between containers and other cloud services.
- **IoTronic:** As part of the OpenStack ecosystem, IoTronic enables dynamic management of IoT devices, facilitating the integration of CPS resources into the cloud infrastructure for seamless control and orchestration.

4.2.6.4 Data Lake Layer

The Data Lake Layer collects, stores, and processes the vast amounts of data generated by CPS devices. This layer supports a wide range of data streams and enables complex analytics. Key components include:

- **Trove:** Trove offers Database-as-a-Service (DBaaS), providing support for relational and non-relational databases. It simplifies the management of large datasets generated by urban sensors and smart systems.
- **Sahara:** This Big Data processing framework uses Apache Spark to facilitate the extraction of valuable insights from the large amounts of data collected by CPS devices.
- **MQTT Broker:** The MQTT Broker enables real-time data distribution and notification services, ensuring timely delivery of critical messages and alerts across the smart city infrastructure.

4.2.6.5 Workflow Engine Layer

The Workflow Engine Layer provides the tools for Urban Data Scientists to design and manage complex data pipelines and workflows that run across cloud services and edge computing environments. This layer enables the creation of distributed computing tasks and automation of IoT workflows through the following tools:

- **Qinling:** Qinling is OpenStack's Function-as-a-Service (FaaS) platform, which allows for the execution of serverless functions, enabling developers to deploy code without managing the underlying infrastructure.
- **Node-RED:** Node-RED is a visual programming tool that simplifies the creation of IoT and automation applications using a drag-and-drop interface for designing data flows and automating processes.

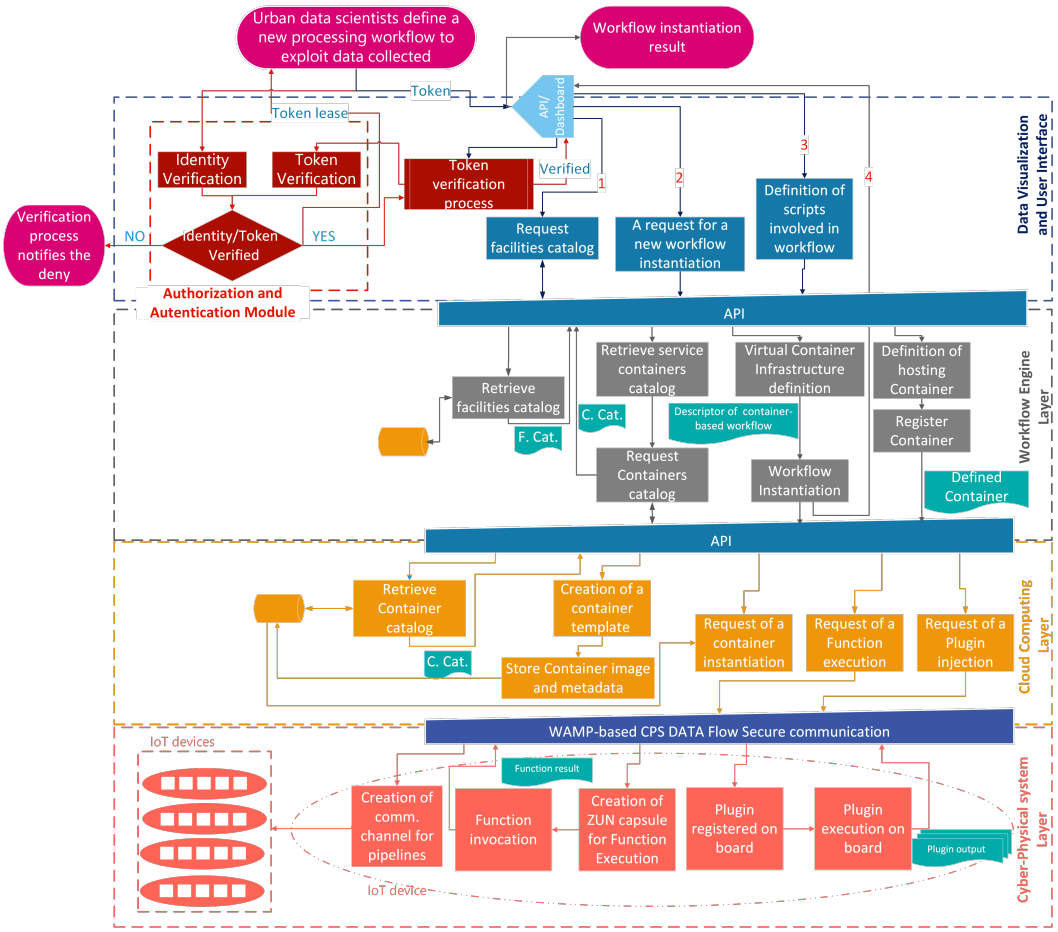


Figure 4.52: Instantiation of a new application-level workflow operated by an Urban Data Scientist. Source: [61]

Figure 4.52 illustrates the workflow of a new application-level task designed by an Urban Data Scientist. This figure highlights the orchestration of data pipelines and distributed computing tasks necessary for managing complex smart city services.

4.2.6.6 Administrator Interface Layer

The Administrator Interface Layer offers a user-friendly interface for managing and monitoring the urban intelligence infrastructure. It includes tools for managing cloud resources, visualizing real-time data, and monitoring the performance of deployed applications:

- **Horizon:** Horizon provides a graphical user interface (GUI) for managing OpenStack services, including containers, IoT devices, and virtual networks.
- **Grafana:** Grafana is integrated into the system for real-time data visualization, enabling administrators to create interactive dashboards and monitor the performance of various CPS and cloud services.

4.2.6.7 User and Urban Data Scientists Communication Interface Layer

This layer manages user authentication, authorization, and secure access control to the smart city platform. **Keystone**, OpenStack's Identity Service, provides Role-Based Access Control (RBAC) and ensures that only authorized users have access to the platform:

- **Keystone:** Keystone handles authentication and authorization, ensuring that users can securely access cloud resources based on their roles. The service is secured behind a proxy/bastion host system, providing an additional layer of security.

4.2.6.8 Service and Application Deployment Workflow

The deployment of services and applications within this architecture follows a structured workflow that leverages edge and cloud computing resources. The key steps involved include:

1. **Orchestration of CPS Resources:** IoT and CPS resources are integrated into the system, enabling real-time computing and data processing directly at the edge, reducing latency and bandwidth usage.
2. **Edge-based Elaboration:** FaaS and IoTronic plugins enable data preprocessing at the edge, reducing the load on cloud resources and improving the efficiency of real-time services.
3. **Pipeline Definition:** Data pipelines are defined using networking features, allowing for event-based or scheduled data processing from CPS resources.
4. **Access Control:** Secure access points are defined based on user roles, with administrators having privileged access and other users or data scientists accessing the platform via the Communication Interface Layer.

4.2.6.9 Lessons Learned

The development and implementation of this architecture provided several insights:

1. Scalability and Flexibility: The integration of OpenStack with edge computing enables the system to scale efficiently while maintaining flexibility, allowing for real-time data processing directly at the edge and reducing latency.

2. Security: A multi-layered security approach, particularly through Keystone and secure networking, effectively reduces the external attack surface and protects critical smart city infrastructure.

3. Workflow Automation: The Workflow Engine Layer facilitates the automation of complex data pipelines and distributed computing tasks, significantly improving the efficiency of smart city applications.

4. Data Management: The Data Lake Layer effectively manages and stores the vast amounts of data generated by CPS devices, enabling advanced analytics and data-driven decision-making for urban planners and administrators.

5. Future Directions: Further research will focus on optimizing data flow between CPS devices and cloud services, exploring decentralized architectures using blockchain for enhanced security, and expanding the architecture to accommodate autonomous systems such as self-driving vehicles.

4.3 Emergency Response and Civil Protection

As cities grow and become more complex, responding rapidly and managing resources effectively during crises, such as natural disasters or pandemics, becomes crucial. Advanced technologies such as additive manufacturing, blockchain, and UAVs (drones) are revolutionizing emergency response and civil protection strategies. By enabling more efficient and scalable solutions, these technologies play a critical role in enhancing the resilience and responsiveness of emergency management systems.

4.3.1 Integrating Cloud Computing with UAV Autopilot Systems for Coordinated Mission Efficiency

Unmanned Aerial Vehicles (UAVs) are increasingly utilized in emergency scenarios for tasks such as delivering medical supplies to remote areas or monitoring large-scale events. Integrating cloud computing with advanced autopilot systems improves the operational efficiency of UAVs by enabling real-time mission planning and obstacle avoidance. This integration supports scalable and autonomous UAV operations, which is essential for timely emergency responses. Such innovations underscore the importance of technology in optimizing emergency management systems [19].

4.3.1.1 Architectural Framework

The architectural framework integrates cloud computing capabilities with the onboard systems of Unmanned Aerial Vehicles (UAVs) to enable more complex, dynamic, and scalable mission execution. The integration is achieved through the **Stack4Things (S4T)** middleware, which extends cloud computing functionalities to UAV operations. This framework offloads computationally demanding tasks, such as real-time data processing, obstacle detection, and dynamic mission planning, from the UAV's onboard systems to the cloud infrastructure. By offloading these tasks, UAVs can operate more efficiently, focusing their onboard resources on critical control tasks while leveraging the cloud for heavy computation.

The system architecture is depicted in Figure 4.53, showing the integration between the **PX4 autopilot firmware**, which manages flight control, and cloud services facilitated by S4T. The **Lightning-Rod (LR)** daemon, deployed on the UAV, acts as an intermediary between the UAV and the cloud, allowing real-time bi-directional communication. Through this communication, sensor data from the UAV is streamed to the cloud for processing, while the cloud can send control commands and updated flight paths back to the UAV.

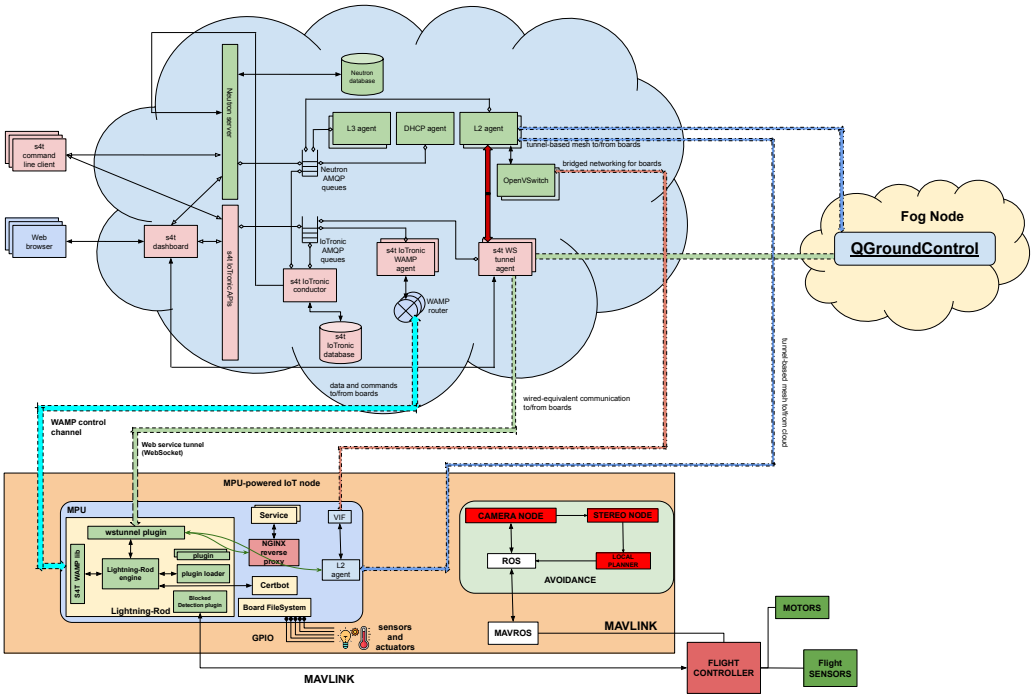


Figure 4.53: Architectural integration of PX4-Avoidance with I/O cloud, illustrating the interaction between UAV onboard systems and cloud computing services. Source: [19]

Communication between the UAV and cloud infrastructure is managed via the **MAVLink** protocol, a lightweight messaging protocol optimized for UAV systems. The **MAVROS** package bridges the PX4 autopilot firmware with the Robot Operating System (ROS), ensuring seamless exchange of telemetry and control data between the UAV and the cloud. This architecture supports real-time transmission of telemetry data for cloud processing, which includes flight path optimization, obstacle detection, and dynamic adjustment of mission parameters.

The architecture's scalability is one of its key strengths. By leveraging cloud resources, the system can accommodate multiple UAVs simultaneously. The cloud infrastructure dynamically allocates computational resources based on the real-time processing needs of each UAV, ensuring that every vehicle receives the necessary resources to execute its mission efficiently.

4.3.1.2 Path-Splitting Strategy for Workload Optimization

One of the primary innovations introduced in this framework is the **path-splitting strategy**, which optimizes workload distribution across a fleet of UAVs. The strategy addresses the challenge

of minimizing mission completion times by dynamically allocating mission waypoints to UAVs based on factors such as current position, remaining battery life, and computational load.

The path-splitting strategy is implemented using the **Greedy Path Splitting Algorithm**, presented in Algorithm 1. Initially, waypoints are assigned to each UAV, and the algorithm iteratively reassigns waypoints to minimize the total mission time. The UAVs communicate with a cloud-based orchestrator, which continuously updates the mission plan based on real-time data and computational resources available in the cloud.

Algorithm 1 Greedy Path Splitting Algorithm for UAVs

Data: $N \leftarrow$ set of UAVs
Data: $P \leftarrow$ set of mission waypoints
Data: $D \leftarrow$ distance matrix between waypoints
foreach *UAV* $u \in N$ **do**
  Assign initial waypoint p_0 to u
while *not all waypoints are assigned* **do**
  foreach *UAV* $u \in N$ **do**
    Find nearest unassigned waypoint p_{next} to u Assign p_{next} to u Update u's mission path

Table 4.7 compares mission times for single UAV operations and dual UAV configurations. The dual UAV configuration, enabled by the path-splitting strategy, significantly reduces mission completion time, with an average improvement of 48.7%. This reduction highlights the effectiveness of distributing the workload across multiple UAVs.

Table 4.7: Comparative Analysis of Mission Completion Times

Path ID	Single UAV Time (s)	Dual UAV Time (s)
Path1	355.25	199.45
Path2	720.49	421.15
Path3	375.69	181.05
Path4	327.06	166.19
Path5	803.95	344.90

The path-splitting strategy is especially beneficial in missions with waypoints distributed across large geographic areas. By dividing the mission into smaller, manageable segments and assigning these to individual UAVs, the system reduces overall mission time and improves resource utilization. The cloud-based orchestrator also reassigns waypoints dynamically in response to real-time conditions, such as obstacle detection or UAV failure, ensuring mission success under variable conditions.

4.3.1.3 Obstacle Avoidance and Block-Detection System

This research introduces an enhanced obstacle avoidance system integrated into the UAV's operational framework through the **PX4-Avoidance** module to complement the path-splitting strategy. This system employs a **block-detection algorithm** that enhances the UAV's ability to autonomously navigate complex environments filled with dynamic obstacles.

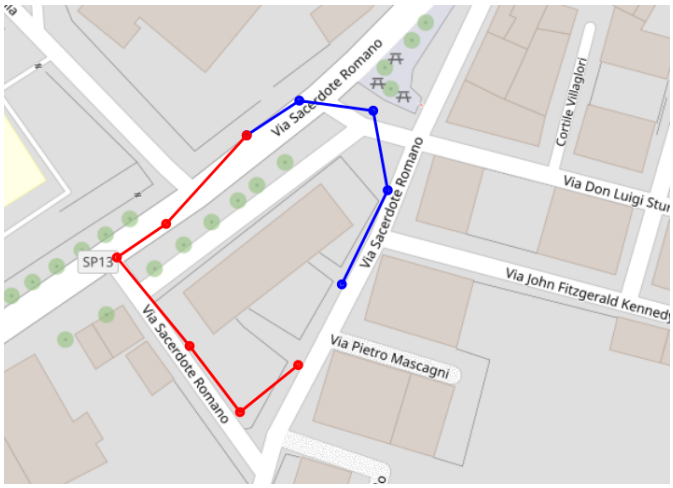


Figure 4.54: Subdivision of a generic mission path into two segments (V1 and V2), demonstrating the path-splitting strategy. Source: [19]

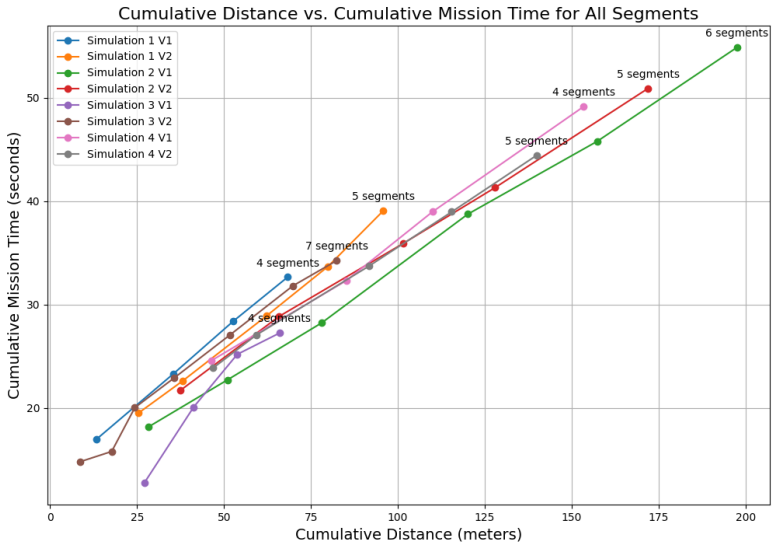


Figure 4.55: Cumulative distance vs. cumulative mission time, illustrating the efficiency gains through path-splitting. Source: [19]

The block-detection algorithm uses the Haversine formula to calculate the distance between the UAV's current position and target destination. If the UAV's progress falls below a predefined threshold, indicating that an obstacle blocks the UAV, the algorithm triggers a corrective action. This action may involve rerouting the UAV or adjusting its flight parameters to avoid obstacles while maintaining progress toward the mission objectives.

The performance of the block-detection system was evaluated in a high-fidelity simulation environment designed to replicate real-world conditions with various obstacles. The results demonstrated that the block-detection algorithm significantly enhanced the UAV's ability to navigate

through complex environments autonomously, reducing the likelihood of mission failure due to unanticipated obstacles. Integration with the cloud infrastructure enables real-time monitoring and dynamic adjustment of flight paths, further improving UAV operational reliability.

Figure 4.54 illustrates the path-splitting strategy, highlighting the subdivision of a mission path into two distinct segments assigned to separate UAVs. Figure 4.55 shows the cumulative distance traveled versus cumulative mission time, illustrating the efficiency gains achieved through the path-splitting strategy.

4.3.1.4 Experimental Evaluations and Results

A series of experiments were conducted in simulated environments that closely resembled real-world UAV operations to validate the block-detection system's performance and assess the effectiveness of the path-splitting strategy.

The obstacle avoidance system was evaluated in environments featuring dense obstacles, where the UAVs had to navigate through narrow corridors and avoid collisions. The block detection algorithm successfully reduced mission failures by enabling real-time obstacle avoidance and rerouting. This system effectively maintained UAV autonomy and ensured successful mission completion, even in challenging environments.

The path-splitting strategy was evaluated across multiple scenarios involving varying levels of complexity and geographic distribution of mission objectives. The results of these experiments are summarized in Figures 4.54 and 4.55, illustrating the efficiency improvements achieved by distributing mission tasks across multiple UAVs. These data show that the path-splitting strategy reduces mission completion time and optimizes resource utilization, making it a valuable tool for large-scale UAV missions.

4.3.1.5 Lessons Learned

The key insights gained from this research are as follows:

1. Scalable UAV Operations: The integration of cloud computing with UAV operations enables scalable and efficient task distribution, allowing for more complex mission execution across multiple UAVs.

2. Path-Splitting Efficiency: The path-splitting strategy demonstrated significant reductions in mission completion times, proving its effectiveness in optimizing UAV resource utilization in large-scale missions.

3. Advanced Obstacle Avoidance: The block-detection system significantly enhanced UAV autonomy by enabling real-time obstacle detection and rerouting, ensuring successful mission completion in dynamic environments.

4. Future Enhancements: Future research will focus on improving the system's scalability to accommodate even larger UAV fleets. Integrating AI and machine learning techniques into the cloud-based control system could enhance decision-making capabilities and adaptive mission planning. Security considerations, particularly concerning cloud-UAV communication, will also be a focal point to ensure the reliability and integrity of these systems in real-world deployments.

4.3.2 Decentralized Emergency Manufacturing and Supply Chains Using 3D Printing and Blockchain

UAVs have become crucial in emergencies, such as delivering medical supplies or monitoring affected areas. The integration of cloud computing enhances the efficiency of UAV operations by enabling real-time mission planning and coordination, which is essential for timely and effective responses during crises [19].

Similarly, the COVID-19 pandemic highlighted significant vulnerabilities in global supply chains, particularly in ensuring the timely availability of medical supplies. The Air Factories 2.0 initiative was developed to address these challenges, leveraging 3D printing and blockchain technology to create a decentralized production system capable of rapidly producing and distributing essential medical devices during emergencies.

Air Factories 2.0 ensures transparency, security, and efficiency in emergency supply chains by integrating smart contracts, tokenization, decentralized governance, and blockchain-based quality control. These innovations facilitate a more agile and resilient response to critical medical needs, demonstrating how advanced technologies can transform emergency logistics and production systems [22].

4.3.2.1 System Architecture and Blockchain Integration for Air Factories 2.0

The Air Factories 2.0 platform is designed with a decentralized architecture integrating blockchain technology with distributed 3D printing capabilities. The primary goal is to create a secure, transparent, and efficient system for emergency response supply chains. The architecture, depicted in Figure 4.56, provides the framework for managing the production and distribution of 3D-printed products across a distributed network of Air Makers (3D printer operators) and Air Callers (users requesting printed products). This architecture combines two distinct blockchain technologies, Ethereum and Hyperledger Fabric, enabling both public and permission operations to coexist.

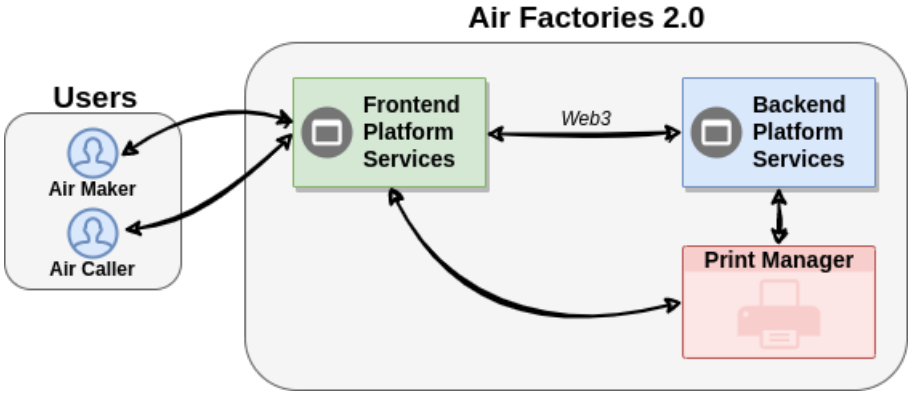


Figure 4.56: System architecture of the Air Factories 2.0 platform, integrating Ethereum and Hyperledger Fabric blockchains. Source: [22]

The dual blockchain framework ensures efficient decentralized production while securing the data management of sensitive processes. As a public blockchain, Ethereum manages tokenized

transactions and smart contracts, while Hyperledger Fabric, a permissioned blockchain, controls internal operations, such as production scheduling, identity management, and access control.

4.3.2.2 Ethereum Smart Contracts and Hyperledger Fabric

The Ethereum blockchain facilitates decentralized transactions through the use of smart contracts, which are self-executing agreements coded directly on the blockchain. These contracts automate key aspects of the manufacturing process, such as order placement, payment, and quality control, ensuring transparency and security without intermediaries. Smart contracts deployed on Ethereum manage Air Factories Tokens (AFTs), an ERC-20 token, which acts as the medium of exchange within the ecosystem. Token transactions recorded on Ethereum ensure an immutable, transparent history of exchanges.

Hyperledger Fabric, on the other hand, is employed to handle sensitive data and high-throughput transactions. Its permissioned network structure ensures that only authorized participants can access internal operations, such as production data and business logic related to scheduling and quality control. Hyperledger's modularity separates business logic components through chaincode, thus improving system flexibility and security. For example, Hyperledger Fabric stores production data manages identities and enforces business rules regarding production scheduling and quality verification.

4.3.2.3 3D Printer Integration, Oracles, and Middleware

The platform interfaces with 3D printers through a middleware layer that connects the blockchain network to physical production units. This middleware consists of APIs and communication protocols that enable real-time monitoring and control of each print job, collecting data such as material availability, print status, and environmental conditions. The middleware facilitates real-time status updates on production, making it possible to track the progress of a 3D print, gather production data, and enforce quality control measures.

Oracles are crucial in interfacing off-chain data (i.e., external data from 3D printers) with the blockchain. Oracles collect real-time metrics from printers and communicate them to the blockchain, ensuring that smart contracts are triggered based on accurate and timely data. For instance, oracles transmit data such as print progress, material consumption, and environmental metrics to inform blockchain-based decisions, such as payment release, production validation, and scheduling adjustments.

4.3.2.4 Proof-of-Print (PoP) Mechanism and Quality Control

A critical feature of Air Factories 2.0 is its Proof-of-Print (PoP) mechanism, a blockchain-based verification system designed to guarantee the quality and authenticity of the 3D-printed products. As illustrated in Figure 4.57, the Printer Controller interfaces with the 3D printer to monitor key parameters throughout the printing process, including layer-by-layer progress, environmental conditions (e.g., temperature, humidity), and material usage. The collected data is stored on the InterPlanetary File System (IPFS), an off-chain storage solution, and cross-referenced with the original design specifications stored on the blockchain.

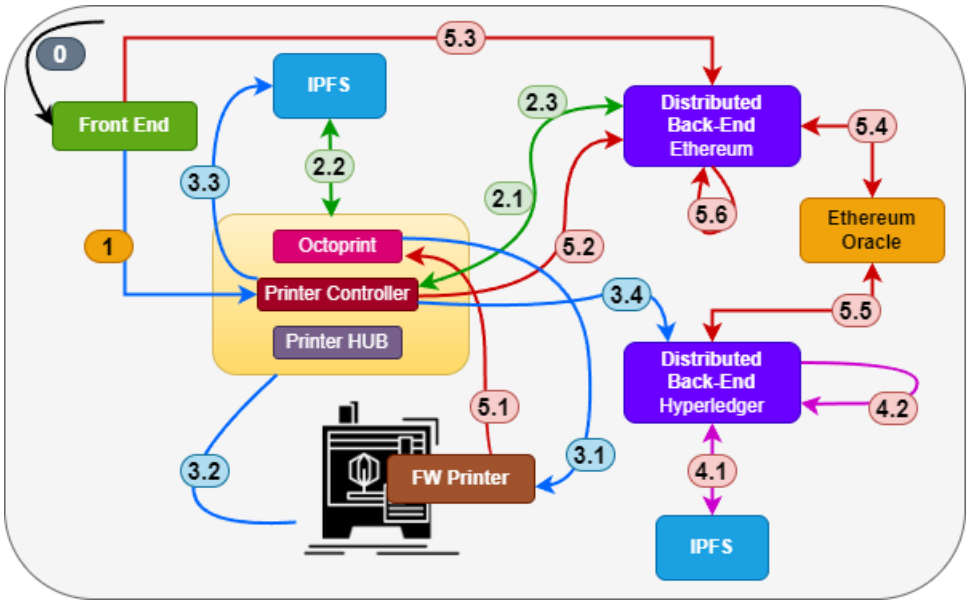


Figure 4.57: Proof of Print (PoP) process in Air Factories 2.0, ensuring the quality and authenticity of the printed products. Source: [22]

Once the printing is completed, if the final product meets the required standards, the PoP mechanism triggers the release of payment to the Air Maker. In the event of discrepancies between the printed product and the digital design, corrective actions are initiated, potentially halting the print job. The PoP mechanism ensures that only verified and quality-controlled products are distributed, thereby safeguarding the integrity of the production process.

Quality control is further enhanced through blockchain chaincode and image analysis. Product images are captured using low-cost cameras installed on the 3D printers during the printing process. These images are processed using the OpenCV library, comparing the real-time product with the original design model. Any discrepancies are flagged for inspection, ensuring the final product meets the predefined specifications. The results of these quality checks are securely recorded on Hyperledger Fabric, making the entire process tamper-proof and auditable.

4.3.2.5 Geolocated Production Scheduling and Resource Management

Air Factories 2.0 employs a geolocated scheduling algorithm to optimize production across a distributed network of Air Makers. The algorithm matches Air Callers (users requesting printed products) with Air Makers based on geographic proximity, material availability, and printer specifications. This matching process reduces transportation costs and minimizes the environmental impact by prioritizing nearby printers. The allocation of print jobs is determined by a scoring system, which considers various factors, as shown in Table 4.8.

This scoring system ensures that the most suitable printer is selected for each job, balancing speed, quality, and cost-effectiveness. Furthermore, the onboarding process for new air makers and air callers involves collecting critical data, such as geographic location, printer specifications, material stocks, and production capacity, and securely registering them on the Ethereum blockchain. Smart contracts automate the onboarding, ensuring data is validated and recorded accurately.

Table 4.8: Printer Scoring Criteria for Job Allocation

Criteria	Weightage
Printer Availability	30%
Material Compatibility	25%
Print Quality	20%
Geographic Proximity	15%
Printer Reliability	10%

4.3.2.6 Tokenization and Decentralized Governance

The Air Factories Tokens (AFTs) are used within the platform as the primary medium of exchange. These ERC-20 tokens allow users to pay for printing services, access resources, and reward participants. The decentralized nature of the platform's economy ensures that users can propose new products, vote on key decisions, and participate in the platform's governance. The tokenization of the economy also introduces transparency and security into the payment processes, as all transactions are recorded immutably on the Ethereum blockchain.

In addition to facilitating the marketplace for printing services, AFTs empower decentralized governance by enabling token holders to vote on platform upgrades, changes in reward structures, and new feature implementations. Figure 4.58 outlines the decentralized voting process used to propose and validate new products for printing, ensuring that only technically feasible and ethically sound products are approved.

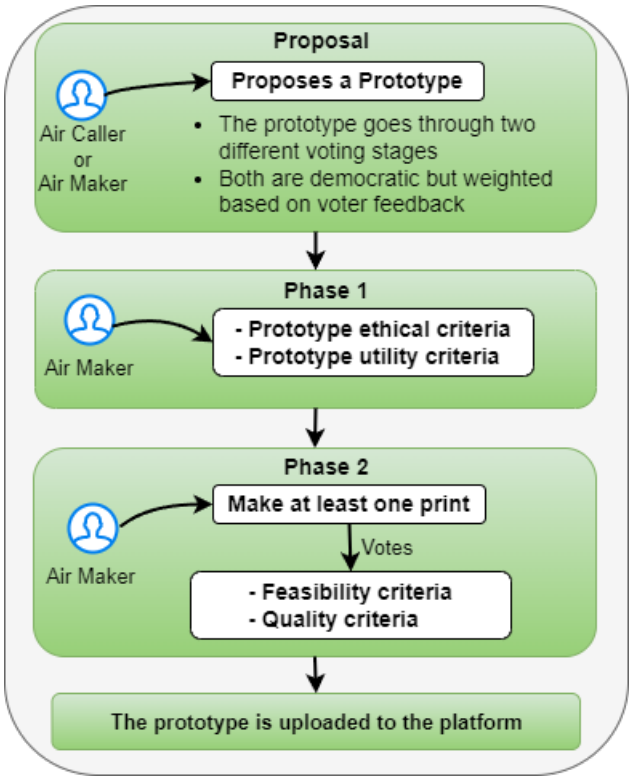


Figure 4.58: Voting process for new product proposals in Air Factories 2.0. Source: [22]

4.3.2.7 Lessons Learned

The development and implementation of Air Factories 2.0 has provided key insights into the potential of decentralized production platforms:

1. Dual Blockchain Efficiency: The integration of Ethereum and Hyperledger Fabric allows the system to leverage the strengths of both public and permissioned blockchain networks, providing security, scalability, and transparency.

2. Secure Production and Quality Assurance: The PoP mechanism ensures that every product meets quality standards, with real-time data captured and verified via blockchain-based chaincode and IPFS. The automated quality control processes minimize errors and prevent fraud.

3. Decentralized Economy: AFTs and decentralized governance foster an ecosystem where users have a direct stake in the platform's development, creating a resilient and flexible marketplace for 3D-printed products.

4. Scalability and Flexibility: The decentralized nature of the system allows it to scale across geographies, and the geolocated scheduling algorithm optimizes production by reducing costs and environmental impact.

4.4 Healthcare: Additive Manufacturing and Telemedicine

The same transformative potential seen in emergency response logistics also drives significant advancements in the healthcare sector. Building on of technologies like 3D printing and IoT devices, healthcare is similarly being reshaped by the advent of telemedicine and personalized care solutions. These innovations are improving patient monitoring, diagnosis, and treatment, especially in remote or underserved areas.

By integrating additive manufacturing techniques, such as 3D printing, with telemedicine, healthcare delivery becomes more accessible, customizable, and efficient. This combination allows for the rapid production of personalized medical devices and the provision of care tailored to individual patient needs, enhancing the overall quality and accessibility of healthcare services.

4.4.1 Torque Force Applied Evaluation in Dental Implant Employing IoT Device

In addition to advancements in personalized healthcare delivery, another critical area of investigation focuses on precision in medical procedures. In dental implantology, precision plays a crucial role in ensuring the long-term stability of implants and minimizing complications. A preliminary pilot study explores this by employing an IoT device, such as an Arduino board, to measure the manual tightening torque applied by clinicians during dental implant procedures.

This study seeks to quantify the impact of clinician-controlled preload on complication rates, aiming to reduce implant failure. Although still in its early stages, the results of this research could be further enhanced through cloud-based architectures, enabling real-time data analysis and refinement of clinical techniques. The following research lays the foundation for developing more precise medical procedures [45].

4.4.1.1 Materials and Methods

To accurately quantify the preload applied to dental implants during manual tightening, a precise measurement system was developed. The system comprises both mechanical and electronic components to ensure comprehensive data capture and minimize human error. The mechanical component includes a custom-designed torque wrench, fabricated using Fused Deposition Modeling (FDM) additive manufacturing, while the electronic component integrates a strain-gauge-based load cell connected to an Arduino microcontroller. This configuration enables the system to provide real-time torque data with high precision, independent of clinician variability.

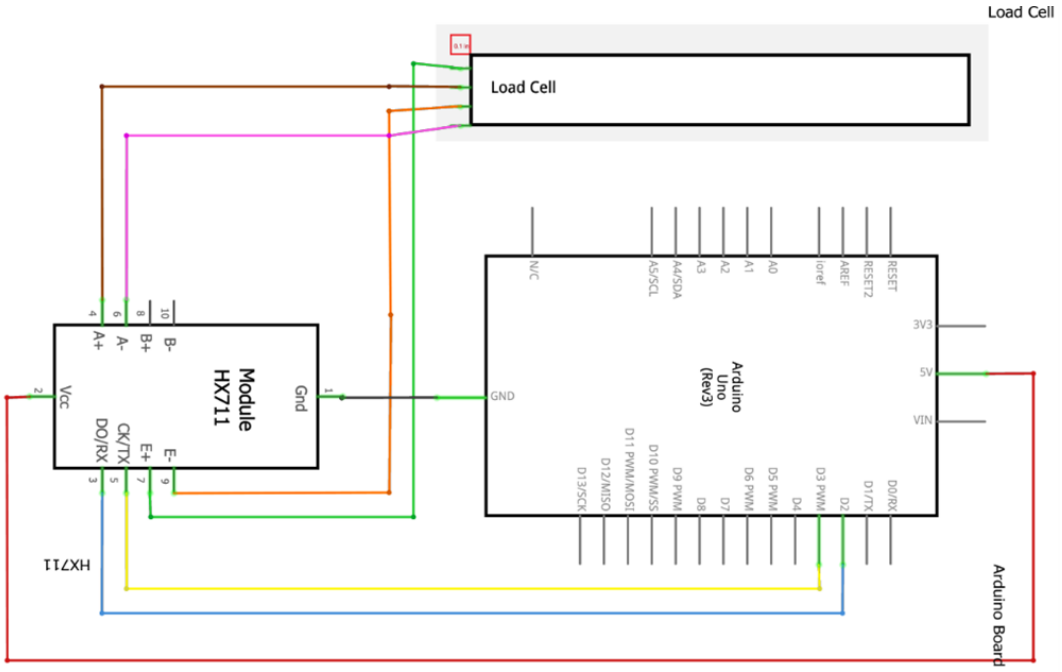


Figure 4.59: Wiring configuration of the load cell, HX711 ADC, and Arduino microcontroller, optimized for signal integrity and minimal noise. Source: [45]

4.4.1.2 Hardware Configuration and Sensor Integration

The core of the measurement system is a strain gauge load cell embedded within a Wheatstone bridge circuit. The Wheatstone bridge, which consists of four resistors arranged in a diamond configuration, is a highly sensitive and precise method for detecting small changes in resistance. The resistors are arranged such that any deformation in the load cell induces a proportional change in resistance, which is captured as a differential voltage across the bridge.

The strain gauge operates on the principle of resistance change under mechanical strain, as described by the following equation:

$$\Delta R = R \times (1 + 2 \times \epsilon + k \times \epsilon^2)$$

where ΔR is the change in resistance, R is the initial resistance, ϵ represents the mechanical strain, and k is the non-linearity constant, typically negligible for small deformations. The Wheat-

stone bridge configuration ensures the system compensates for environmental variations such as temperature fluctuations, thereby maintaining measurement accuracy.

The differential voltage generated by the Wheatstone bridge is amplified using an HX711 Analog-to-Digital Converter (ADC), which is specifically designed for load cell applications. The HX711 includes an internal Programmable Gain Amplifier (PGA) with selectable gains of 32, 64, and 128, enabling it to amplify the small signals from the load cell. The ADC converts the amplified analog signal into a 24-bit digital signal, ensuring high-resolution force measurements even under small strain variations.

The HX711 is connected to an Arduino Rev4 microcontroller via I2C protocol, allowing for efficient, low-latency communication between the load cell and the processing unit. The Arduino, featuring an ATmega328P microprocessor clocked at 16 MHz, provides sufficient computational power for real-time data acquisition and signal processing, while also offering multiple GPIO pins for potential expansion. The overall wiring configuration is designed to minimize electrical noise and interference, ensuring robust signal integrity throughout the system. Figure 4.59 illustrates the load cell configuration, HX711, and Arduino.

4.4.1.3 Software Infrastructure and Signal Processing

The software infrastructure was developed using the Arduino IDE, incorporating custom libraries to interface with the HX711 ADC and manage the torque data acquisition process. The system leverages the HX711 library's core functions, including 'begin()', 'read_average()', and 'get_units()', to handle the initialization, data reading, and calibration of the load cell.

The data acquisition process begins with the 'read_average()' function, which collects multiple samples of the raw data from the load cell and averages them to minimize the impact of noise. The system performs a real-time calibration using known weights, enabling the conversion of raw data into meaningful torque units Newton meter (Nm). The calibration process is further refined using a least squares optimization algorithm, described in the following section, to ensure that the measurements align with physical standards.

To reduce electromagnetic interference and signal degradation, the wiring between the load cell, HX711, and Arduino is kept as short as possible, and the power lines are shielded from the signal lines. The following wiring configuration is implemented:

- **E+ (Green):** Connects the load cell to the HX711 E+ terminal and Arduino GND.
- **E- (Orange):** Connects the load cell to the HX711 E- terminal.
- **A+ (Purple)** and **A- (Brown):** Connect the load cell to the A+ and A- terminals of the HX711, respectively.
- **VCC (HX711):** Connected to the Arduino's 5V power supply to ensure stable voltage.
- **I2C Communication:** The HX711 communicates with the Arduino over the I2C protocol via the Serial Clock Line (SCL) and Serial Data Line (SDA) pins, connected to Arduino D2 and D3, respectively.

This configuration ensures optimal signal quality with minimal interference, allowing for precise torque measurements.

4.4.1.4 Mathematical Framework and Calibration

The accuracy of torque measurements is ensured through a rigorous calibration process. Known weights are applied to the load cell to establish a linear relationship between the sensor readings and the actual torque values. The calibration factor F is derived using the following equation:

$$F = \frac{\sum L}{\sum P}$$

where $\sum L$ represents the sum of the sensor readings and $\sum P$ represents the sum of the known calibration weights. This factor is stored in the system's non-volatile memory, ensuring consistent accuracy across multiple uses without the need for repeated calibration.

The system also incorporates real-time error compensation for temperature variations and mechanical hysteresis, factors that can influence strain gauge measurements. The least squares optimization algorithm employed during calibration minimizes the residual error between the measured and expected values, ensuring a high degree of linearity in the sensor's response across the operating range.

4.4.1.5 Design and Prototyping of the Torque Wrench

The mechanical component of the system is a custom torque wrench, designed using Siemens NX and fabricated using Polylactic Acid (PLA) through additive manufacturing. The wrench was designed to be ambidextrous, making it suitable for both left- and right-handed clinicians. The design underwent Finite Element Analysis (FEA) to ensure that it could withstand the forces applied during manual tightening without structural failure.

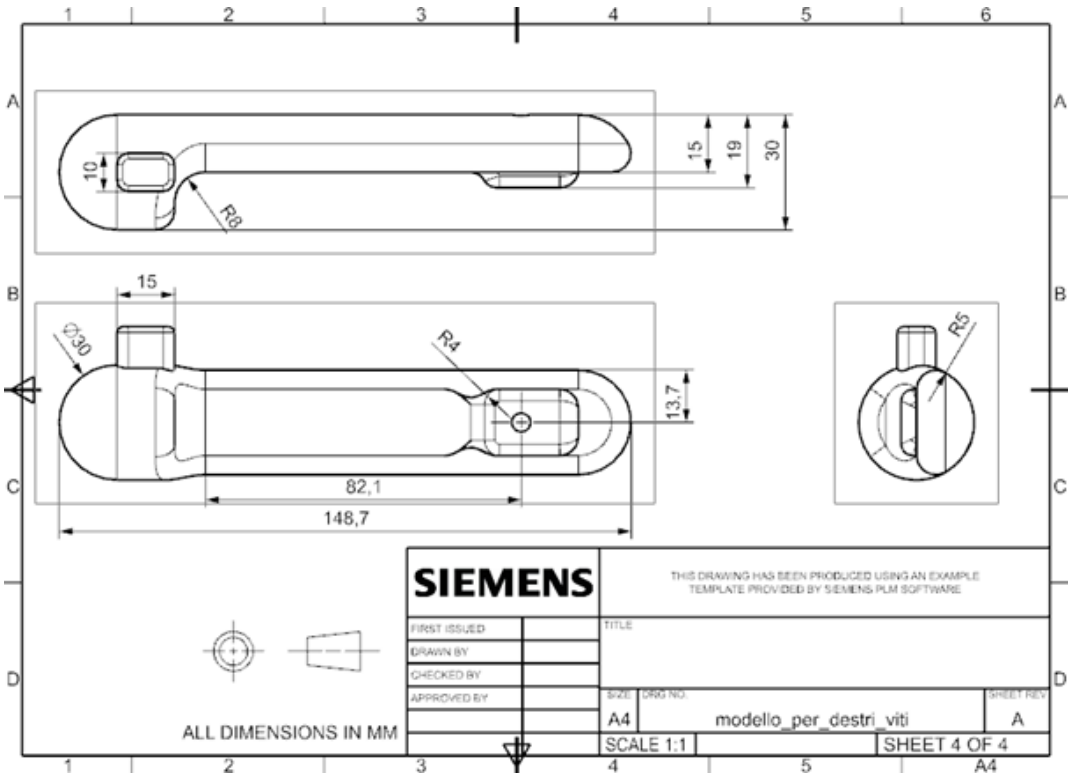


Figure 4.60: Orthogonal projection view of the 3D-printed torque wrench, designed using Siemens NX software. Source: [45]

The FEA simulation, conducted with Tetra 10 elements, involved a total of 23,905 elements and 38,841 nodes, as shown in Figure 4.61. The model was subjected to a worst-case scenario force of 20 kg applied parallel to the load cell axis. The results, depicted in Figure 4.62, indicate that the maximum deflection along the z-axis was 1 mm, well within the acceptable range for clinical applications. This deflection ensures that the wrench maintains its structural integrity without affecting the accuracy of the torque measurements.

Figure 4.61: Loads and boundary conditions applied during Finite Element Analysis (FEA) of the torque wrench. Source: [45]

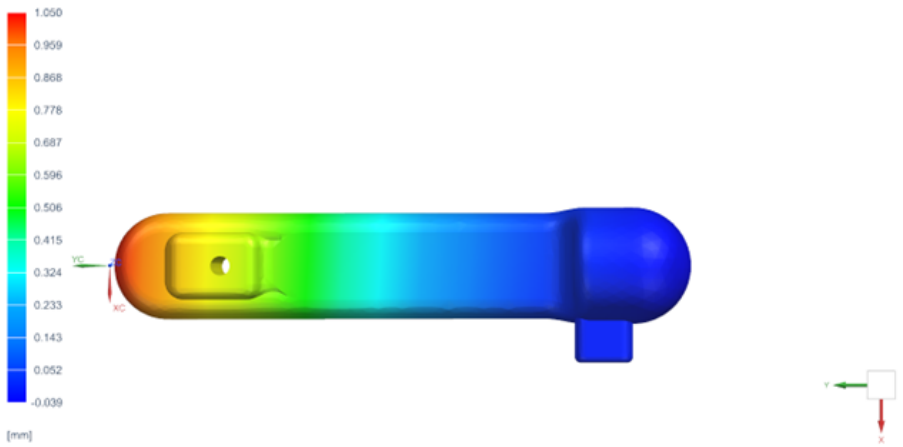


Figure 4.62: Deformation along the z-axis under maximum applied load, showing a maximum deflection of 1 mm. Source: [45]

4.4.1.6 Results and Discussion

The FEA results confirmed that the torque wrench design can withstand the forces applied during clinical use without significant deformation. The maximum deflection in the z-axis was measured at 1 mm, which is within the acceptable tolerance for the system's intended application. Furthermore, the load cell detected load variations as small as 20 grams, ensuring the precision necessary for accurate preload application during implant procedures.

The calibration process demonstrated a high degree of linearity between applied force and sensor readings, with a minimal residual error after applying the least squares optimization algorithm. This ensures that the device can provide clinicians with reliable and consistent torque measurements, reducing the risk of over- or under-tightening dental implants.

4.4.1.7 Lessons Learned

This study yielded several key insights into the design and implementation of a system for accurately measuring preload in dental implant procedures:

1. Sensor Calibration and Error Compensation: The integration of the strain gauge load cell with a Wheatstone bridge circuit, coupled with the HX711 ADC, provided precise torque measurements. The calibration process, enhanced by a least squares optimization algorithm, minimized measurement error and ensured high accuracy.

2. Mechanical Design Validation: Finite Element Analysis (FEA) verified that the PLA-printed torque wrench could withstand clinical forces with acceptable deflection under worst-case loading conditions.

3. Signal Integrity and Noise Reduction: Careful attention to the wiring configuration and using a low-noise I2C protocol minimized signal degradation. They ensured that the system delivered accurate real-time data under clinical conditions.

4. Future Work: Future studies should focus on real-world clinical trials to evaluate the system's performance across different clinicians and procedures. Additional sensor integrations, such as temperature or humidity sensors, could further enhance measurement reliability.

4.4.2 Web of Things Architecture for Remote Healthcare Monitoring

As IoT continues to be integrated into healthcare, the focus is on precision in medical procedures, remote patient monitoring, and device management. Telemedicine has increasingly become a vital application of IoT and cloud computing, enabling real-time, remote patient health monitoring. The Web of Things (WoT) paradigm builds on this by addressing the challenges of scalability, interoperability, and security often encountered in traditional IoT systems.

WoT employs web standards, allowing IoT devices to communicate via web protocols and expose functionalities through RESTful APIs. This approach is particularly critical in telemedicine, ensuring secure, real-time interaction between medical devices and healthcare providers across geographically distributed environments. Furthermore, integrating Hypermedia as the Engine of Application State (HATEOAS) enhances system flexibility, decoupling client interactions from server functionalities and allowing independent server-side updates without disrupting client operations—significant as new devices and capabilities are continuously introduced in healthcare settings [18].

4.4.2.1 System Architecture and Components

The system's architecture is built on the Stack4Things (S4T) framework, an extension of OpenStack designed for managing IoT devices. The architecture comprises several core components, as depicted in Figures 4.63 and 4.64, which ensure scalability, security, and flexibility in managing medical IoT devices within telemedicine environments.

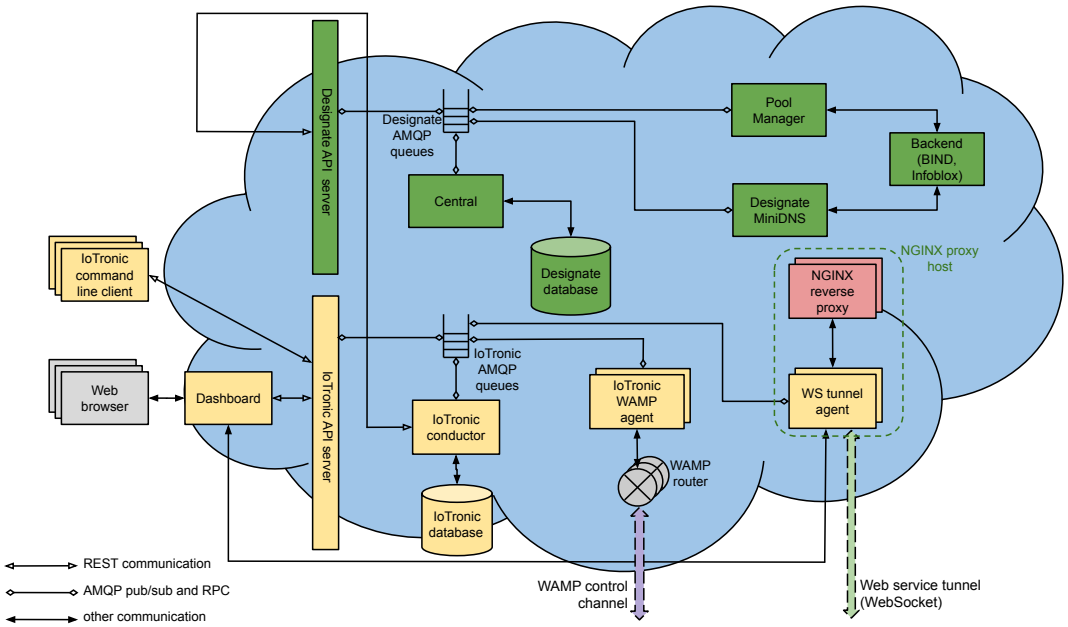


Figure 4.63: IoTronic and Designate integration design for cloud-side IoT management. Source: [18]

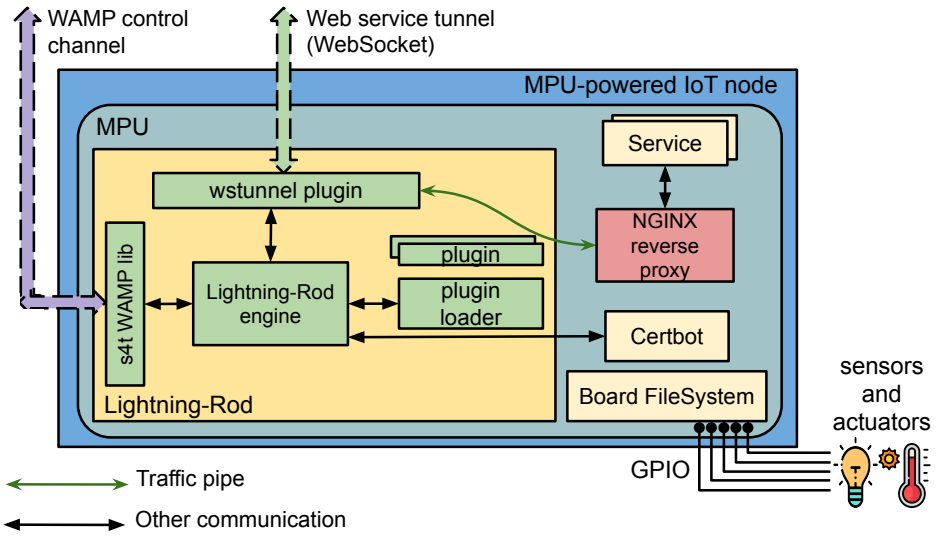


Figure 4.64: S4T Lightning-Rod architecture for node-side operations and interaction with cloud-based IoTronic subsystem. Source: [18]

- **IoTronic Subsystem (Cloud-Side)**: IoTronic is the cloud-based subsystem within OpenStack responsible for managing IoT nodes distributed across diverse geographic locations. This subsystem offers a centralized platform for deploying, configuring, and monitoring IoT devices. IoTronic communicates with IoT nodes using persistent WebSocket connections, ensuring real-time bidirectional data flow between the cloud and devices.

- **Lightning-Rod Agent (Node-Side)**: The Lightning-Rod (LR) agent is deployed on each IoT node. It serves as the intermediary between local IoT devices (e.g., sensors and actuators) and the cloud IoTronic subsystem. LR manages local resources, maintaining secure communication with the IoTronic subsystem via WebSocket tunnels, which ensure uninterrupted data exchange.

- **Designate Service (DNS-as-a-Service)**: Designate, an OpenStack service providing DNS-as-a-Service (DNSaaS), manages DNS records dynamically, allowing IoT devices to be accessed via public domain names. This eliminates the need for static IP addresses or dedicated domains for each IoT node, streamlining the process of exposing IoT resources on the web.

- **NGINX Reverse Proxies**: NGINX reverse proxies are utilized to route incoming client requests to the appropriate IoT nodes. When a client sends a request, it is routed to the NGINX proxy in the cloud, which forwards the request through the WebSocket tunnel to the correct LR agent on the target IoT node, ensuring secure communication.

- **HATEOAS-Enabled RESTful APIs**: The system interacts with IoT resources through RESTful APIs, utilizing HATEOAS to allow clients to navigate and discover resources dynamically. For instance, a client may request a list of available sensors on a medical device, and the system will provide hypermedia links to those sensors, enabling further interactions without requiring prior knowledge of the resource structure.

The combined use of IoTronic, Lightning-Rod, and Designate services ensures the efficient management of medical IoT devices. This provides scalable solutions for real-time data retrieval and control while maintaining security across distributed environments.

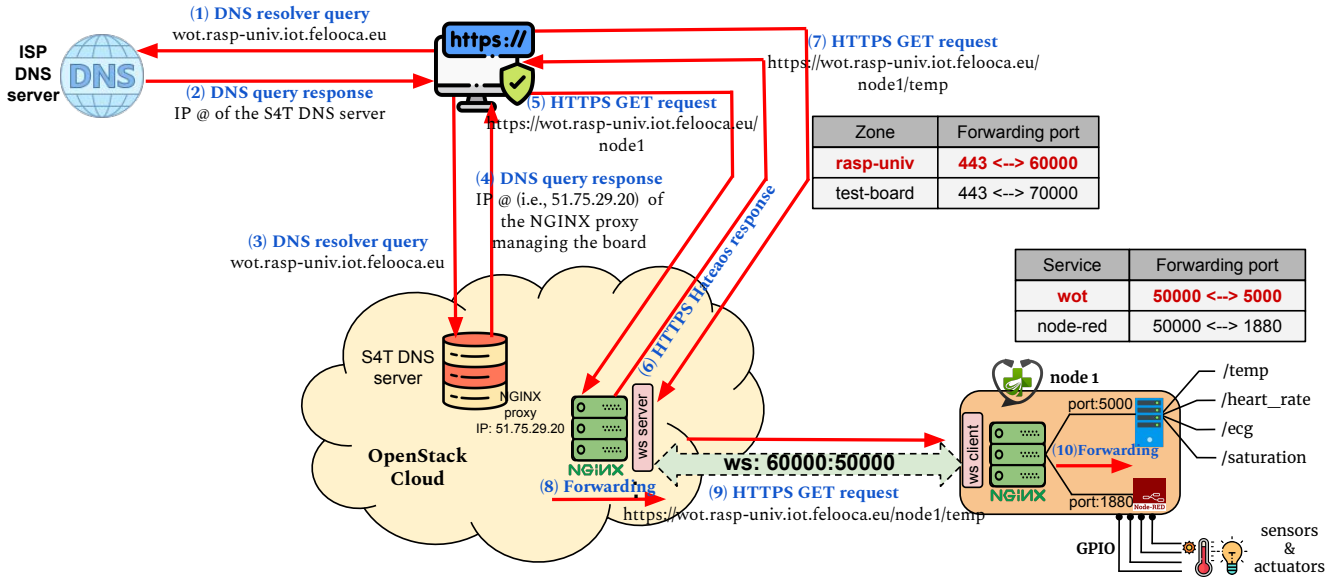


Figure 4.65: Telemedicine workflow utilizing the Stack4Things WoT routing mechanism. Source: [18]

4.4.2.2 Workflow: Telemedicine Use Case

In the telemedicine scenario, IoT-enabled medical devices continuously monitor patient data. The workflow, depicted in Figure 4.65, shows the interaction between a client (e.g., a web application) and a medical sensor connected to an IoT node. The sequence of operations is as follows:

1. **DNS Resolution**: The client begins by querying the DNS to resolve the domain name of the IoT device (e.g., 'wot.rasp-univ.iot.felooca.eu'). The DNS server responds with the IP address of the Stack4Things (S4T) DNS server.

2. **Proxy Routing**: The client sends an HTTP request to the S4T DNS server, which resolves the subdomain and directs the request to the appropriate NGINX reverse proxy.

3. **WebSocket Tunnel**: The NGINX proxy forwards the request through a secure WebSocket tunnel to the Lightning-Rod agent on the targeted IoT node. The LR agent processes the request and retrieves the requested data from the connected medical device or sensor.

4. **HATEOAS Interaction**: The system responds with hypermedia links, enabling the client to further interact with resources dynamically. For example, after retrieving patient vitals, the client may follow links to initiate an ECG reading or adjust medical device settings.

5. **Secure Data Transmission**: All communications between the client and IoT device are encrypted via HTTPS, with X.509 certificates issued and validated by the Let's Encrypt

Certificate Authority (CA) client. This ensures secure transmission of sensitive medical data.

4.4.2.3 HATEOAS and RESTful API Design

The combination of HATEOAS and RESTful APIs enables flexible and dynamic client-server interactions. Each IoT node exposes a unified entry point (e.g., '/wot') through which the client can explore and interact with various resources based on the state of the system and client permissions. This simplifies client-side interactions and minimizes the need for updates when new devices or services are introduced.

An example of a hypermedia API response is shown in Listing 4.1. The client is dynamically guided through the available resources, such as triggering an ECG reading or retrieving heart rate data from a medical sensor.

Listing 4.1: Example hypermedia API response enabling dynamic resource discovery

```
{
 "data": {
   "patient_id": '001',
   "age": '48',
   "sex": 'M'
 },
 "_links": {
    "ecg": {
      "href": "https://wot.rasp-univ.iot.felooca.eu/001/ecg",
      "type":"POST"
    },
    "heart_rate": {
      "href": "https://wot.rasp-univ.iot.felooca.eu/001/ecg/
         ecg_result",
      "type":"GET"
    }
  }
}
```

This dynamic interaction model ensures the system can adapt to changes in the IoT landscape, allowing clients to discover new functionalities and data without requiring modifications to the API or client code.

4.4.2.4 Pragmatic Proof Algorithm for Goal Achievement

To efficiently achieve complex goals, such as retrieving patient data or controlling medical devices, the system employs a Pragmatic Proof Algorithm. This algorithm iteratively generates and evaluates pre-proofs and post-proofs to guide clients through API operations to meet their desired outcomes.

The algorithm operates as follows:

1. **Pre-Proof Generation**: Based on the current state and the client's goal, the system generates a pre-proof outlining the required API calls.
2. **HTTP Request Execution**: The client executes the specified HTTP requests, interacting with IoT devices and retrieving or sending data as needed.
3. **Post-Proof Evaluation**: The system evaluates post-proofs after each interaction to verify whether the goal has been achieved. If further steps are required, additional pre-proofs are generated.
4. **Goal Completion**: Upon achieving the goal, the system delivers the final data or confirms the completed action.

This process ensures efficient, dynamic client interaction with IoT devices, adapting to system changes and achieving complex tasks without requiring prior knowledge of the API structure.

4.4.2.5 Lessons Learned

The integration of WoT, HATEOAS, and Stack4Things in the telemedicine use case yielded valuable insights:

- **Scalability**: The architecture demonstrated strong scalability, supporting the seamless addition of new IoT devices without requiring modifications on the client side.
- **Interoperability**: By utilizing web standards and RESTful APIs, the system ensured interoperability between various medical devices, facilitating easy integration and management of heterogeneous IoT resources.
- **Security**: The use of HTTPS and X.509 certificates provided robust security, protecting patient data and ensuring compliance with healthcare data regulations.
- **Dynamic Interaction**: HATEOAS allowed for flexible, dynamic interactions between clients and IoT resources, enabling discovery and use of new devices and features without requiring client updates.
- **Efficiency**: The Pragmatic Proof Algorithm enhanced system efficiency by guiding clients through API steps necessary to achieve their goals, minimizing redundant interactions and optimizing resource usage.

The integration of these technologies within a cloud-based telemedicine architecture offers a robust, scalable, and secure solution for real-time patient monitoring and control of medical devices.

4.4.3 Motion Capture-Based Design of Hip Prostheses

As telemedicine and IoT improve patient monitoring and care, additive manufacturing (AM) advancements are revolutionizing how medical devices, such as implants, are customized for individual patients. Customizing medical implants is crucial for improving patient outcomes, enhancing comfort, and reducing the need for revision surgeries.

This study presents an advanced framework for designing hip prostheses using AM technologies, combining numerical simulations and experimental methodologies. Integrating machine learning (ML) algorithms further optimizes the design process, refining load distribution and enhancing the mechanical durability of the implants. This approach demonstrates how advanced manufacturing techniques and computational tools are critical in developing patient-specific medical devices, further transforming modern healthcare [46].

4.4.3.1 System Architecture for Customized Hip Prostheses

Optimizing customized hip prostheses leverages advanced additive manufacturing techniques, numerical simulations, and biomechanical analysis to create prostheses tailored to patients' needs. This study integrates biomechanical data from motion capture, parametric multibody dynamic models, and Finite Element Method (FEM) simulations. These methods are combined with machine learning algorithms to enhance prosthesis design, ensuring durability and biomechanical compatibility under physiological loading conditions.

4.4.3.2 Biomechanical Data Acquisition and Joint Kinematics

To accurately simulate the physiological conditions experienced by a hip prosthesis, the biomechanical data of patients during common movements such as walking is critical. Motion capture technology employs the OpenPose framework, which detects 135 key points on the human body (Figure 4.66). Key joints such as the hip, knee, and ankle are tracked, providing precise kinematic data that forms the foundation for biomechanical analysis.

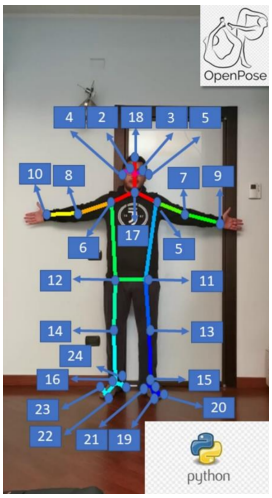


Figure 4.66: Keypoints detected by OpenPose during motion capture, enabling the analysis of joint kinematics. Source: [46]

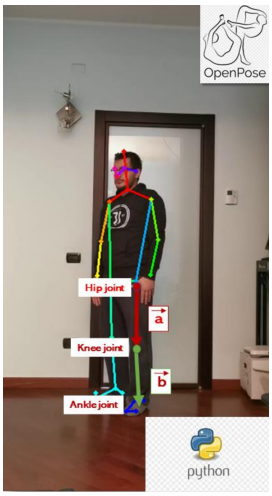


Figure 4.67: Knee angle calculation using vector geometry based on motion capture data. Source: [46]

The joint angles, particularly the knee angle, are calculated based on the positions of key points using vector mathematics. Given two vectors, $\vec{a}$ and $\vec{b}$, constructed from the coordinates of the hip, knee, and ankle joints, the knee angle θ is determined using the following equation:

$$\theta = \cos^{-1}\left(\frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|}\right) \tag{4.2}$$

This equation represents the cosine of the angle between two vectors, where $\vec{a}$ and $\vec{b}$ are derived from the joint coordinate differences. Figure 4.67 illustrates the geometric representation of the vector calculations for knee angle determination.

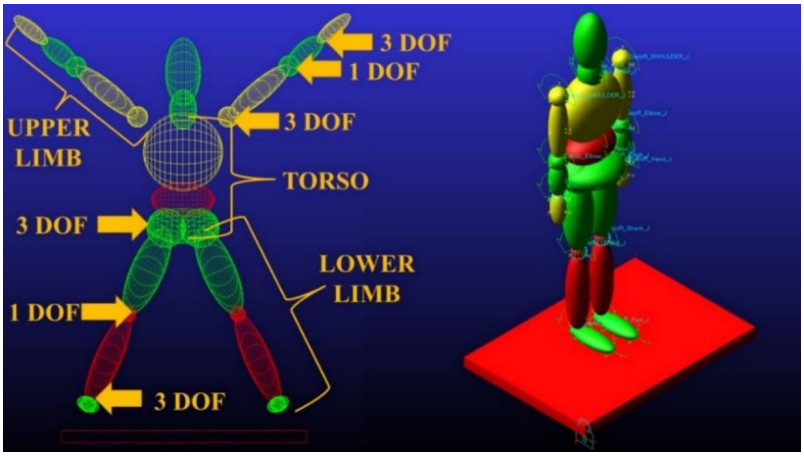


Figure 4.68: Parametric multibody model used for simulating gait dynamics and force transmission through the hip joint. Source: [46]

4.4.3.3 Parametric Multibody Model for Gait Simulation

Once the joint kinematic data is obtained, a parametric multibody dynamic model is developed in the MSC ADAMS® environment. This model comprises rigid body segments representing the upper limbs, torso, and lower limbs (Figure 4.68). Each segment is connected by joints, allowing for the simulation of the human gait cycle, with particular attention to the hip joint.

The multibody model simulates human motion dynamics, providing critical information about the forces and torques acting on the hip joint. These forces are directly influenced by the joint angles and the external loads applied during activities such as walking. The multibody dynamic simulation provides input forces for subsequent FEM analysis, ensuring that the prosthesis design accounts for realistic biomechanical conditions.

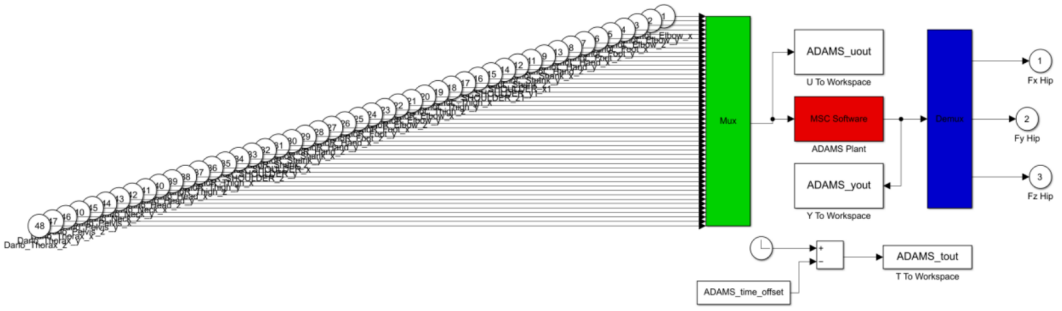


Figure 4.69: Mesh refinement process during FEM analysis to achieve stress distribution accuracy. Source: [46]

4.4.3.4 Finite Element Analysis (FEM) for Stress Distribution

The forces and torques obtained from the multibody dynamics simulation are then applied to the prosthesis model for Finite Element Method (FEM) analysis. The goal is to determine the stress distribution within the prosthesis under physiological loading conditions, ensuring the design's durability and mechanical integrity. The FEM analysis begins with a coarse mesh, which is progressively refined until convergence is achieved. The mesh refinement process is illustrated in Figure 4.69, where the mesh is adjusted iteratively until the displacement between successive iterations differs by less than 2%.

The material selected for the prosthesis is AISI 316L stainless steel, which is chosen for its superior mechanical properties, such as high yield strength, fatigue resistance, and biocompatibility. Table 4.9 provides the mechanical properties used in the FEM simulations, which are critical for accurately simulating the prosthesis's response to physiological loads.

Table 4.9: Mechanical properties of AISI 316L used in FEM analysis.

Property	**Value**
Density [kg/mm^3]	7954
Young's Modulus [GPa]	195
Poisson's Ratio	0.25
Yield Strength [MPa]	250

4.4.3.5 Shape and Lattice Structure Optimization

The optimization process of the prosthesis is twofold: geometric optimization and lattice structure optimization. Geometric optimization is performed using the FMINCON algorithm, which minimizes the stress concentrations in the prosthesis while maintaining its structural integrity. Figure 4.70 illustrates the optimization workflow, where the prosthesis dimensions are iteratively adjusted to achieve the desired mechanical performance.

To further reduce the mass of the prosthesis without compromising strength, lattice structures are introduced into areas where stresses are low. Lattice structures reduce material usage and weight while maintaining adequate support where required. The lattice optimization technique, shown in Figure 4.71, helps reduce the prosthesis's overall weight by up to 35%, making it more comfortable for the patient while maintaining mechanical performance.

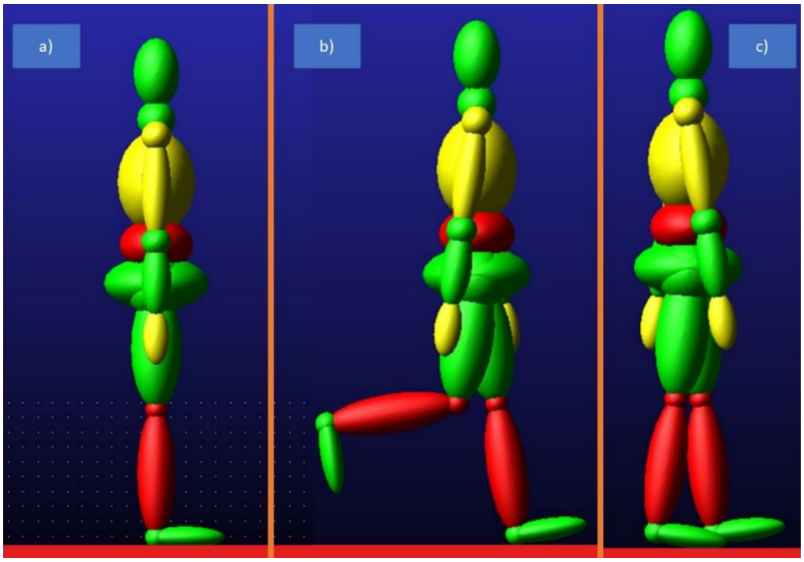


Figure 4.70: Optimization process for shape refinement of the prosthesis. Source: [46]

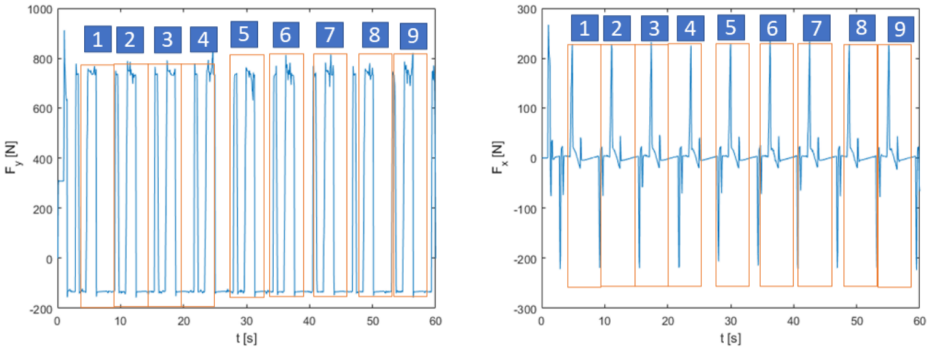


Figure 4.71: Lattice optimization applied to reduce prosthesis weight. Source: [46]

4.4.3.6 Lessons Learned

The optimization of customized hip prostheses, achieved through the integration of multibody dynamics, FEM, and additive manufacturing, provided several key insights:

- The use of OpenPose for biomechanical data acquisition enabled precise tracking of human motion, critical for accurate joint force calculations in multibody dynamics.
- The parametric multibody model effectively simulated the biomechanical conditions experienced by the hip joint, providing realistic loading scenarios for FEM analysis.
- Mesh refinement in FEM analysis was essential to capture the accurate stress distributions within the prosthesis, ensuring the design's mechanical durability under physiological loads.
- The combination of shape and lattice optimization techniques significantly reduced the prosthesis weight while maintaining its mechanical strength, enhancing both patient comfort and implant longevity.

This research highlights the importance of combining biomechanical analysis, numerical simulation, and optimization techniques to develop more efficient, durable, and patient-specific hip prostheses.

Chapter 5

MocapMe Framework

5.1 Motion Capture: Techniques and Applications

Motion capture (MoCap) is the process of recording and digitally reconstructing the movement of objects or individuals within a virtual environment. This technology is widely employed across diverse fields, including film production, video game development, sports science, and biomechanical analysis. By capturing precise motion data, MoCap provides valuable insights into movement dynamics, enabling various applications, from entertainment to advanced medical research [34].

In clinical rehabilitation, MoCap serves as a critical tool for monitoring patients' progress in recovery. By capturing movements during therapy sessions, clinicians can analyze and assess biomechanical improvements with high accuracy [54]. In the realm of sports science, MoCap facilitates the study of athletes' biomechanics, offering valuable data that can enhance performance and prevent injuries. Furthermore, MoCap technology is integral to real-time animation rendering, enabling the creation of highly realistic character animations by recording the precise movements of live actors [50].

A key component of MoCap systems is *human pose estimation*, which involves identifying key anatomical landmarks, such as joints, using advanced computational methods. In two-dimensional (2D) pose estimation, the primary challenge lies in accurately detecting these key points, or "parts." Pose estimation algorithms typically follow one of two approaches: *top-down* or *bottom-up* [13]. Top-down methods estimate the pose of each individual in the scene, leading to increased computational complexity as the number of subjects grows. Conversely, bottom-up approaches, such as OpenPose [13], detect all key points in an image before associating them with specific individuals, thereby reducing computational complexity relative to the number of people present.

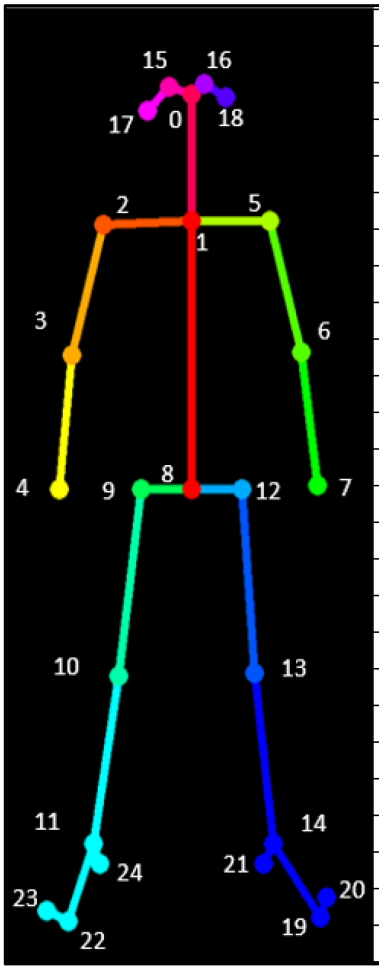
OpenPose is a widely adopted bottom-up, real-time, multi-person human pose detection library. It is capable of detecting 135 key points across the body, hands, face, and feet from a single image without requiring physical markers. OpenPose employs a convolutional neural network (CNN) to perform key-point detection and association, providing markerless motion capture capabilities [64].

The OpenPose library offers three main pose models, each varying in the number of key points detected:

- **MPI**: Estimates 15 key points.
- **COCO**: Estimates 18 key points [37].

- **BODY_25**: Estimates 25 key points, including additional descriptors for the feet and pelvic center [3].

The **BODY_25** model (depicted in Figure 5.1) is particularly comprehensive, making it suitable for detailed biomechanical analysis. Furthermore, an experimental model, *body_25b*, has enhanced accuracy and reduced false positives, thereby improving the reliability of key-point estimation.



Keypoint number	Body part
0	Nose
1	Neck
2	RShoulder
3	RElbow
4	RWrist
5	LShoulder
6	LElbow
7	LWrist
8	MidHip
9	RHip
10	RKnee
11	RAnkle
12	LHip
13	LKnee
14	LAnkle
15	REye
16	LEye
17	REar
18	LEar
19	LBigToe
20	LSmallToe
21	LHeel
22	RBigToe
23	RSmallToe
24	RHeel

Figure 5.1: COCO Pose Model estimating 25 key points. Source: [3]

As the demand for real-time data processing and low-latency feedback increases, horizontal scalability becomes a critical factor in the design of MoCap systems. Scalable architectures enable MoCap systems to handle larger datasets, enhance processing speeds, and deliver real-time results, which are crucial for time-sensitive applications such as high-performance sports analysis or real-time animation rendering [9]. The concept of scalability in MoCap systems is further elaborated in Chapter 3.

5.2 Edge-Centric Optimization of Motion Capture: A MocapMe Integration Perspective

Integrating the MocapMe framework within edge computing architectures represents a significant advancement in MoCap technology, particularly in its ability to meet the increasing demand for high-fidelity, real-time motion analysis. Applications such as clinical rehabilitation, sports biomechanics, and real-time animation rendering require precision in capturing complex movements and the ability to process and analyze large datasets in near-instantaneous timeframes. The decentralized nature of edge computing addresses these critical requirements by allowing data to be processed directly at the point of capture, minimizing latency and enabling immediate feedback loops.

Real-time feedback is paramount in scenarios where MoCap is employed for clinical assessments or athletic performance tracking. Processing data locally at the edge ensures that vital information is available without the delays typically associated with transferring large datasets to centralized cloud systems. Additionally, edge processing alleviates the load on centralized data centers, reducing bandwidth consumption and ensuring that motion capture systems remain responsive and efficient, even in resource-constrained environments.

This chapter delves into the enabling technologies that make this integration possible, focusing on the technical underpinnings of deploying MocapMe within an edge computing ecosystem. Specifically, the chapter examines how distributed computing resources are leveraged to optimize motion capture data processing, enhancing system scalability, responsiveness, and robustness. By exploiting edge computing's capacity to distribute workloads, MocapMe reduces the reliance on traditional cloud-based architectures, thus allowing MoCap systems to operate seamlessly in highly dynamic and demanding environments.

5.3 Enabling Architectures for ML-based Bioengineering: LoRa, P2P Networking, and I/Ocloud

Integrating MocapMe into edge computing could also rely on several enabling technologies, particularly LoRa communication, Peer-to-Peer (P2P) networking, and the I/Ocloud paradigm infrastructure. These technologies can ensure robust data transmission and communication between devices, especially in environments with limited or unavailable traditional network infrastructures.

5.3.1 LoRa as an Enabling Technology

LoRa technology is a wireless communication protocol for long-range, low-power data transmission explained in chapter 2. It is a candidate for integration into motion capture systems deployed in distributed and remote environments. For instance, in a bioengineering context where motion capture is used to monitor patients in rural healthcare facilities, LoRa enables transmitting critical motion data across vast distances without needing a robust, high-bandwidth infrastructure.

Consider a scenario in which a motion capture system is deployed in a remote rehabilitation center where internet connectivity is sporadic and unreliable. Employing LoRa, the system can transmit patient movement data to a central processing unit located far away, ensuring continuous

monitoring and data collection. The low-power aspect of LoRa is particularly beneficial in such scenarios, where devices may need to operate on battery power for extended periods without frequent recharging.

Moreover, LoRa's ability to operate in environments with high interference makes it suitable for motion capture systems deployed in urban or industrial environments where electromagnetic interference is expected. This robustness ensures that the motion capture data is transmitted reliably, even in challenging conditions.

5.3.2 Peer-to-Peer (P2P) Communication

P2P communication, explained in chapter 2, offers a decentralized approach to data exchange that complements the capabilities of LoRa, particularly in scenarios where direct device-to-device communication is needed. In the context of motion capture applications, P2P communication can be instrumental in environments that require real-time data sharing between multiple devices without relying on a centralized server, for instance, during the synchronization process between devices in the multi-camera acquisition approach. Moreover, in a multi-patient monitoring system, each patient's motion capture device can communicate directly with others using P2P networking. This allows the system to aggregate data locally and perform initial processing on-site, significantly reducing the amount of data that needs to be transmitted to a central server for further analysis. This decentralized approach also enhances the system's fault tolerance, if one device or communication link fails, the other devices can continue to operate and share data with minimal disruption. Additionally, P2P communication is highly scalable, making it suitable for large-scale deployments involving many devices.

5.3.3 I/Ocloud as a Supporting Infrastructure

I/Ocloud, a multi-tenant IoT solution explained in chapter 2, is essential in managing the complex workflows associated with distributed motion capture systems. It provides a flexible and scalable platform that integrates with edge devices, such as those using LoRa and P2P, and central cloud servers, enabling seamless data processing, storage, and retrieval across a distributed infrastructure.

Appling the I/Ocloud paradigm in the context of bioengineering applications offers several key advantages:

- **Virtualization of IoT Resources:** I/Ocloud extends the concept of Cloud virtualization to include IoT resources, allowing sensors and actuators attached to IoT nodes to be managed as virtual entities within the Cloud. This means physical IoT devices deployed at the edge can be treated as extensions of the Cloud infrastructure, enabling more flexible and dynamic management of motion capture systems. For instance, motion data collected from a patient's wearable sensors can be virtually connected to Cloud-based analytics services without requiring direct physical interaction with the devices.

- **Edge Computing Integration:** I/Ocloud supports edge computing by allowing IoT nodes to process data locally before transmitting it to the Cloud. In a clinical setting, where real-time motion data analysis is crucial, this capability decreases latency and makes critical information available to clinicians practically instantaneously. For example, in a scenario

where a motion capture system is monitoring the rehabilitation of a patient with neuromuscular disorders, the initial processing of data (such as filtering and noise reduction) can be performed at the edge using I/Ocloud, thereby accelerating the feedback loop between data collection and clinical intervention.

- **Multi-Tenancy and Resource Sharing:** One of the most powerful features of I/Ocloud is its ability to support multi-tenancy, where multiple users or applications can share the same IoT infrastructure without interference. In a research context, this allows different bioengineering projects to run the same motion capture setup, optimizing resource use and reducing overall costs. For instance, a MoCap laboratory could support simultaneous experiments on gait analysis, sports performance, and elderly mobility using the same underlying infrastructure managed by I/Ocloud.

- **Comprehensive Data Management:** I/Ocloud is equipped to handle both structured and unstructured data, making it suitable for the various data types generated in bioengineering applications. Additional sensor inputs, such as heart rate, muscle activation, environmental factors, and motion capture data, can seamlessly integrate into the analysis pipeline. This holistic approach is important for applications that require the synthesis of multiple data streams to generate accurate and comprehensive insights into patient health.

- **Security and Privacy:** I/Ocloud enhances security and privacy by allowing sensitive data to be processed locally at the edge before being transmitted to the Cloud. This is important in healthcare applications, where patient data must be confidential. By minimizing the exposure of raw data to the broader network, I/Ocloud helps protect patient privacy while still leveraging the computational power of the Cloud.

5.4 MocapMe: Machine Learning for Markerless Motion Tracking

This section explores the implementation and outcomes of MocapMe, a framework implemented as a research project, highlighting its efficiency in tracking the sit-to-stand (STS) movement. The system integrates DeepLabCut (DLC) [33, 36, 40, 49, 39] for enhanced accuracy, refining initial estimates from OpenPose [63, 13, 57, 12] to improve stability and precision. This markerless approach eliminates the need for physical markers, making it less intrusive and adaptable to various clinical settings.

The following research shows that MocapMe offers highly efficient and accurate motion data capture compared to other frameworks. It is considered a beta-version tool for physical therapy, rehabilitation, and sports science applications [44]. Markerless motion capture using MocapMe provides significant advantages that are potentially applicable to the medical field. It can be used to monitor and analyze patient movements. Machine learning models within the MocapMe platform enable precise motion tracking for accurate diagnosis and treatment planning in clinical settings. This capability can also extend to sports science, where detailed motion analysis can help develop performance optimization strategies.

5.4.1 Methodology

This section presents a detailed methodology for optimizing and evaluating the DeepLabCut (DLC) model for motion capture, focusing on the Sit-to-Stand (STS) movement. The approach combines OpenPose (OP) for initial keypoint detection and further refines these outputs through DLC training, thus improving both motion-tracking precision and computational efficiency.

5.4.2 Data Collection and Preparation

The dataset used in this study is composed of two distinct sources. The primary dataset comprises 493 videos sourced from a publicly available repository, focusing on individuals performing the STS movement. These videos were standardized to ensure a consistent left-side view of each subject, facilitating uniform analysis and interpretation of the STS biomechanics.

To complement the primary dataset and enhance the robustness of the DLC model, an additional set of 48 videos was collected specifically for this study. This secondary dataset includes three subjects, all of Italian nationality, aged between 28 and 37 years, with a controlled variation in distance (2m, 3m, 4m, and 5m) and angle (0°, 15°, 30°, and 45°) relative to the camera. The setup is depicted schematically in Figure 5.2, which details the experimental design used for video acquisition.

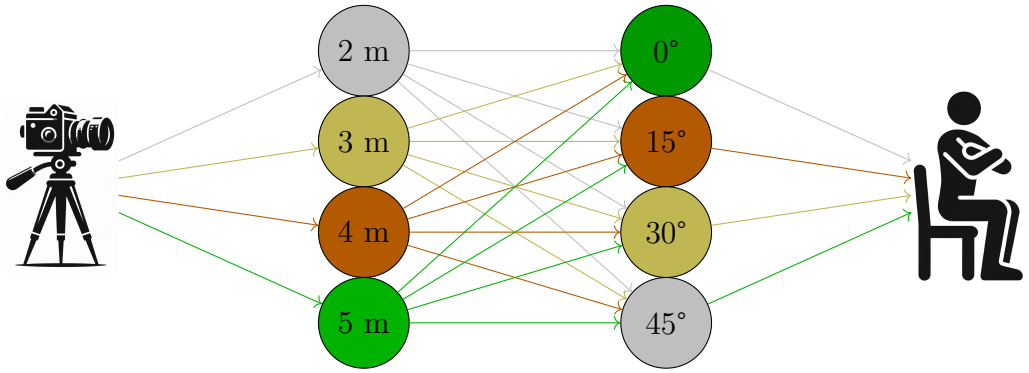


Figure 5.2: Schematic representation of distances and angles used during video acquisition.

The videos were processed through the OpenPose framework to extract initial keypoints for major joints, such as the shoulder, hip, knee, ankle, and foot. To ensure high-quality data, frames with keypoints showing confidence values below 98% were filtered out. The output data were subsequently used to train the DLC model, enhancing motion capture precision by refining the initial OP detections.

5.4.3 Model Architecture

The model architecture utilized in this study is based on ResNet34, specifically tailored for use within the DeepLabCut framework. ResNet34 was selected due to its ability to balance computational efficiency and accuracy, crucial for motion capture tasks requiring both speed and precision. The architecture leverages residual blocks, which mitigate the vanishing gradient issue commonly encountered in deep neural networks. Each residual block includes convolutional layers, batch

normalization, and ReLU activations. The input to each block is fed back into the output via a shortcut connection, ensuring the original information is preserved and facilitating deeper network training without performance degradation.

Input Processing: Input frames were resized to $224 \times 224 \times 3$ Red, Green, Blue (RGB) images. These were passed through an initial convolutional layer with a kernel size of 7×7, a stride of 2, and were followed by a max-pooling operation. The first convolution reduces the spatial resolution, producing an output of $112 \times 112 \times 64$, and further downscales the feature map to $55 \times 55 \times 64$ after max-pooling. Table 5.1 presents the architecture of the ResNet34 model and compares it with deeper ResNet models.

Table 5.1: Architecture of ResNet34 employed in the study, comparing it with deeper ResNet models.

Layer Name	**Output Size**	**18-layer**	**34-layer**	**50-layer**	**101-layer**	**152-layer**
conv1	112x112	7x7, 64, stride 2				
conv2_x	56x56	3x3 max pool, stride 2				
		$\begin{bmatrix} 3x3, 64 \\ 3x3, 64 \end{bmatrix}$ x2	$\begin{bmatrix} 3x3, 64 \\ 3x3, 64 \end{bmatrix}$ x3	$\begin{bmatrix} 1x1, 64 \\ 3x3, 64 \\ 1x1, 256 \end{bmatrix}$ x3	$\begin{bmatrix} 1x1, 64 \\ 3x3, 64 \\ 1x1, 256 \end{bmatrix}$ x3	$\begin{bmatrix} 1x1, 64 \\ 3x3, 64 \\ 1x1, 256 \end{bmatrix}$ x3
conv3_x	28x28	$\begin{bmatrix} 3x3, 128 \\ 3x3, 128 \end{bmatrix}$ x2	$\begin{bmatrix} 3x3, 128 \\ 3x3, 128 \end{bmatrix}$ x4	$\begin{bmatrix} 1x1, 128 \\ 3x3, 128 \\ 1x1, 512 \end{bmatrix}$ x4	$\begin{bmatrix} 1x1, 128 \\ 3x3, 128 \\ 1x1, 512 \end{bmatrix}$ x4	$\begin{bmatrix} 1x1, 128 \\ 3x3, 128 \\ 1x1, 512 \end{bmatrix}$ x8
conv4_x	14x14	$\begin{bmatrix} 3x3, 256 \\ 3x3, 256 \end{bmatrix}$ x2	$\begin{bmatrix} 3x3, 256 \\ 3x3, 256 \end{bmatrix}$ x6	$\begin{bmatrix} 1x1, 256 \\ 3x3, 256 \\ 1x1, 1024 \end{bmatrix}$ x6	$\begin{bmatrix} 1x1, 256 \\ 3x3, 256 \\ 1x1, 1024 \end{bmatrix}$ x23	$\begin{bmatrix} 1x1, 256 \\ 3x3, 256 \\ 1x1, 1024 \end{bmatrix}$ x36
conv5_x	7x7	$\begin{bmatrix} 3x3, 512 \\ 3x3, 512 \end{bmatrix}$ x2	$\begin{bmatrix} 3x3, 512 \\ 3x3, 512 \end{bmatrix}$ x3	$\begin{bmatrix} 1x1, 512 \\ 3x3, 512 \\ 1x1, 2048 \end{bmatrix}$ x3	$\begin{bmatrix} 1x1, 512 \\ 3x3, 512 \\ 1x1, 2048 \end{bmatrix}$ x3	$\begin{bmatrix} 1x1, 512 \\ 3x3, 512 \\ 1x1, 2048 \end{bmatrix}$ x3
	1x1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

5.4.4 Training Strategy

The training process involved several key steps. The DLC framework was initialized, and the preprocessed keypoints from OpenPose served as initial labels for training. The model was trained iteratively to refine these initial detections.

Data Augmentation: Data augmentation techniques such as random rotations, scaling, and flipping were applied to increase dataset diversity and improve the model's ability to generalize to new, unseen data. This is critical for handling real-world camera angles, lighting, and subject movement variations.

Dataset Splitting: The dataset was split into training (80%) and validation (20%) sets using an automated procedure within the DLC framework, ensuring a balanced representation of data in both sets.

Model Training: The model was trained using stochastic gradient descent (SGD) with a learning rate of 0.001, for 300,000 iterations with a batch size 16. Early stopping was employed if validation loss did not improve over 50 consecutive iterations. The mean squared error (MSE) between predicted keypoints and ground truth was minimized throughout training. Table 5.2 presents the training and validation errors (in pixels) and the p-cutoff values, representing the probability threshold for keypoint detection.

Table 5.2: Training results of the DeepLabCut model.

Iteration	Training Iterations	Dataset (%)	Shuffle	Train Error (px)	Validation Error (px)	p-cutoff
1	200,000	80	1	10.7	10.6	0.6
2	250,000	80	1	11.45	11.41	0.6
3	300,000	80	1	10.26	10.25	0.6

5.4.5 Results and Evaluation

The loss curve in Figure 5.3 demonstrates the model's convergence during training. The loss decreased sharply in the early stages, followed by a gradual decline as the model refined its predictions.

Figure 5.3: Training and validation loss during the training process, illustrating convergence. Source: [44]

5.4.6 Deployment and Modular Design

The final implementation of the model was developed in Python, utilizing OpenCV for video processing, pandas for data manipulation, and DeepLabCut for model training and evaluation. The workflow was encapsulated within a Python class called *Model_Calculation*, which streamlined the entire process from data extraction to keypoint prediction. Key methods within this class include:

- **LoadData**: Initializes OpenPose and extracts initial keypoints from video data.
- **EvaluationDataDeepLabCut**: Transforms OpenPose output into DeepLabCut-compatible format.
- **DropUnderThreshold**: Filters keypoints with low confidence values.
- **FeatureExtraction**: Processes keypoints for further analysis with DeepLabCut.

- **DeepLabCutModule**: Manages the initialization, training, and evaluation of the DLC model.

The entire pipeline was executed on a high-performance computing cluster equipped with NVIDIA Graphics Processing Units (GPUs), allowing efficient training and inference.

5.4.7 Computational Efficiency and Precision

Computational Time: A comparison between OpenPose and DLC regarding computational efficiency is shown in Figure 5.4. The DLC-based model significantly reduced the processing time per video due to its optimized architecture.

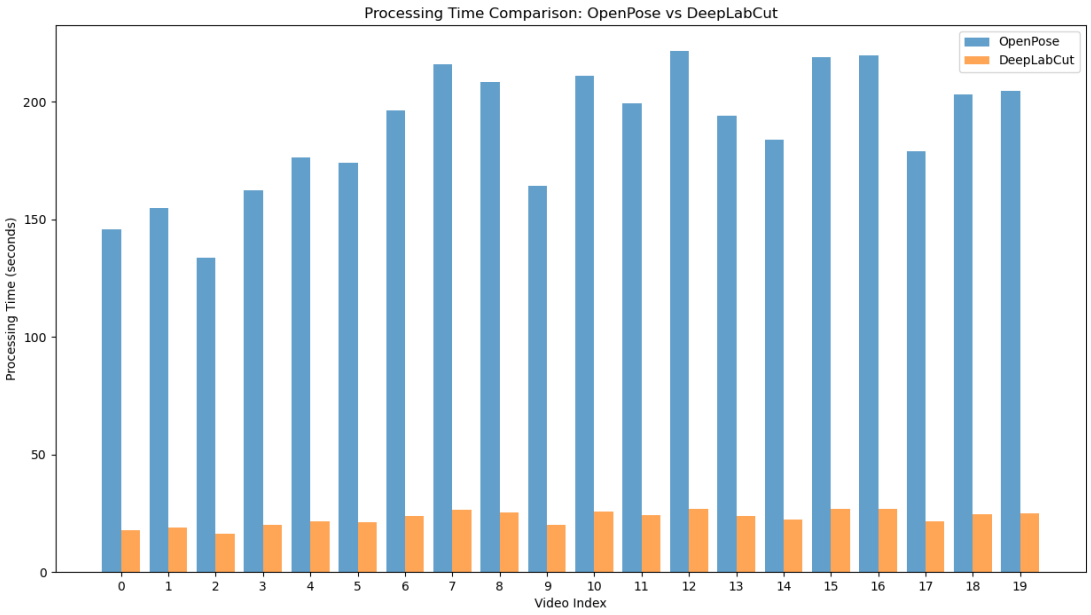


Figure 5.4: Processing time comparison between OpenPose and DLC. Source: [44]

Reliability and Precision of Keypoint Tracking: The reliability of keypoint detection is illustrated in Figure 5.5, where MocapME demonstrates consistently higher confidence across keypoints compared to OpenPose. Figure 5.6 shows the increased stability of ankle and foot keypoints.

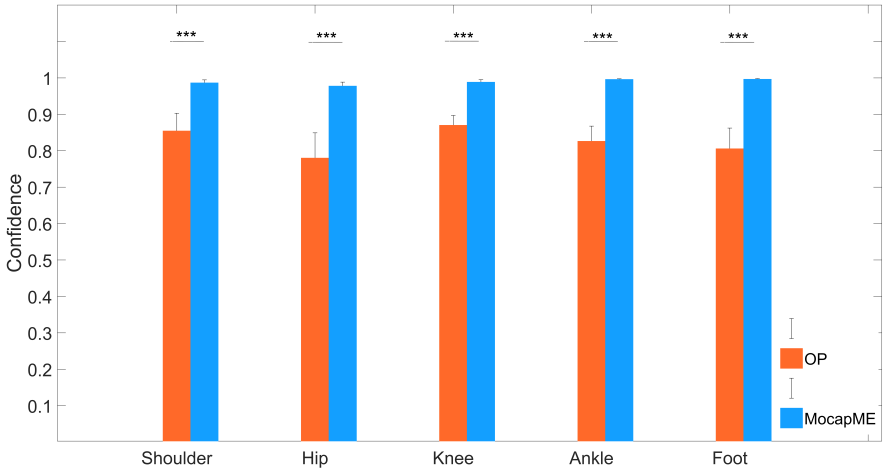


Figure 5.5: Confidence of selected keypoints for OpenPose and DLC-based MocapME. Source: [44]

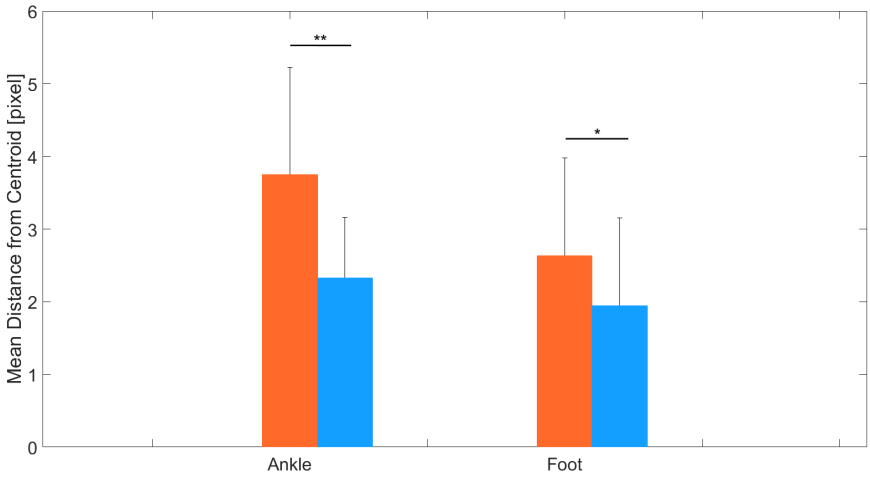


Figure 5.6: Stability of ankle and foot keypoints for OpenPose and MocapME (DLC-based). Source: [44]

5.4.8 Lessons Learned

Through this study, several key lessons were identified:

1. **Combining OP and DLC:** The integration of OP with DLC led to a marked improvement in keypoint tracking precision and computational efficiency. This hybrid approach effectively addresses limitations present in OP-based models alone.

2. **Data Augmentation Impact:** The extensive use of data augmentation improved the model's robustness, allowing it to generalize better to unseen scenarios.

3. **Hardware Utilization:** Efficient GPU utilization was critical in accelerating the training

and inference processes, demonstrating the importance of computational resources in deep learning-based motion capture models.

This comprehensive methodological framework paves the way for future advancements in markerless motion capture, particularly in clinical settings where accurate, efficient, and reliable analysis is paramount.

5.4.9 Deterministic Real-time Performance in MocapMe

In real-time systems, processing tasks rapidly and consistently is essential to ensure that the processing meets predefined timing constraints. A deterministic real-time system guarantees tasks are completed within a deadline, providing predictable performance even under varying workload conditions. In the context of MoCap, this requirement is crucial for applications like clinical rehabilitation, sports analysis, and real-time animation rendering, where processing delays can negatively impact feedback and decision-making processes.

The MocapMe framework aims to achieve deterministic real-time performance by ensuring that motion capture data is processed within a known, predictable timeframe. Even though not all videos may be processed at the original frame rate, the system's deterministic behavior allows us to establish worst-case and best-case execution times, ensuring that deadlines are consistently met. This section presents a detailed analysis of MocapMe's real-time capabilities, discussing how the system performs across different video resolutions and complexities.

5.4.9.1 FPS Ratio: Processed vs. Original

The FPS ratio of processed frames per second to the selected STS video frames per second is a primary real-time performance indicator. A system is considered real-time if it holds an FPS ratio of 1 or higher, indicating that it is processing frames at the same or higher rate than the video's original playback.

In Figure 5.7, it is evident that while some videos exceed the real-time threshold (FPS ratio > 1), the majority do not. This variation indicates that MocapMe is capable of real-time performance in some instances, especially for videos with lower complexity or resolution. However, the system's deterministic behavior ensures the processing time is predictable even when the FPS ratio falls below 1. This allows the system to establish precise processing times and adjust task scheduling to meet the application's real-time deadlines for specific video categories. We can consistently optimize the system to meet real-time application timing constraints by leveraging this predictability.

5.4.9.2 Computation Time per Second of Video vs. Resolution

One of the most significant factors affecting real-time performance is the video resolution. Higher-resolution videos require more computational resources, leading to longer processing times. Figure 5.8 shows the relationship between the video's computation time per second and the video's resolution (total pixels).

As Figure 5.8 shows, videos with lower resolutions have shorter computation times, while higher-resolution videos take longer to process. However, the system demonstrates a consistent and predictable relationship between video resolution and computation time, making it possible

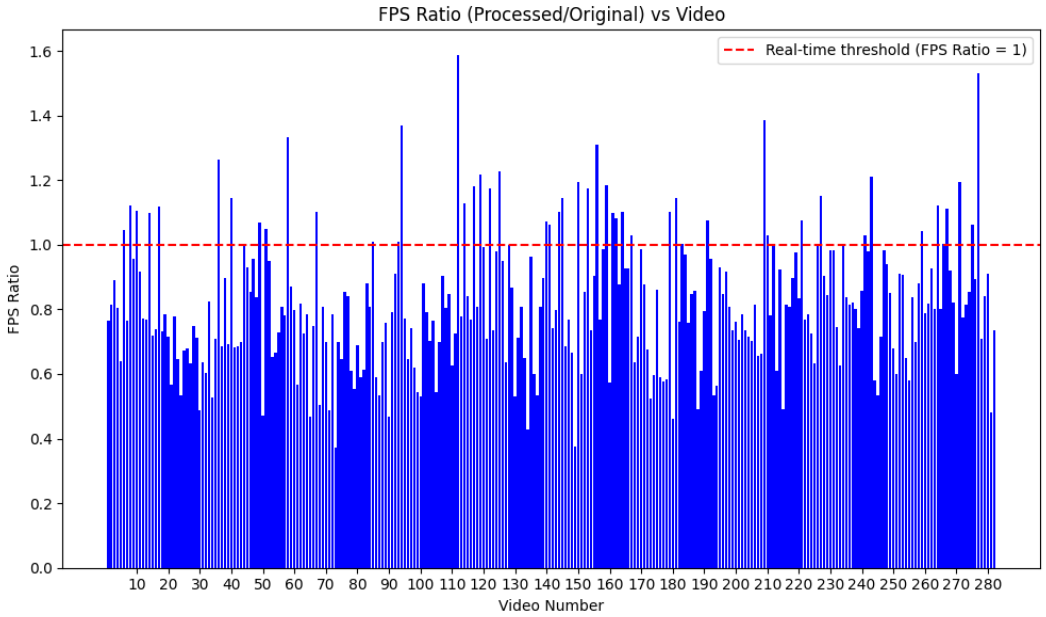


Figure 5.7: FPS Ratio (Processed/Original) vs. Video. The red dashed line represents the real-time threshold (FPS ratio = 1).

to estimate processing times before the task begins. This predictability enables the definition of precise processing schedules and deadlines, ensuring that even more complex, high-resolution videos can be processed within their predefined time limits, albeit at a lower frame rate if necessary.

5.4.9.3 Computation Time per Frame per Pixel

Another metric for evaluating real-time performance is the computation time per frame per pixel, which indicates how efficiently the system processes each frame relative to the video's resolution. Figure 5.9 provides insights into the system's performance per-pixel basis.

In Figure 5.9, it's possible to observe that while there is some variability in the computation time per pixel across videos, the system maintains a relatively stable performance for most of the videos. The variability in some videos can be attributed to differences in content complexity and movement. By analyzing these results, we can determine a worst-case scenario for computation time per pixel and adjust the system's scheduling and resource allocation accordingly. This predictability level ensures that even when some videos are more computationally demanding, the system remains deterministic and can meet deadlines reliably.

5.4.9.4 FPS Processed vs. FPS Original

Finally, the original FPS of the videos and the FPS processed by the MocapMe application are compared. Maintaining an FPS processed close to the original FPS is a crucial goal for real-time performance. Figure 5.10 compares these metrics.

As displayed in Figure 5.10, although the processed FPS often falls below the original FPS, particularly for higher resolution videos, the system is still capable of near-real-time processing for lower resolution or less complex videos. This demonstrates that the system can meet real-time

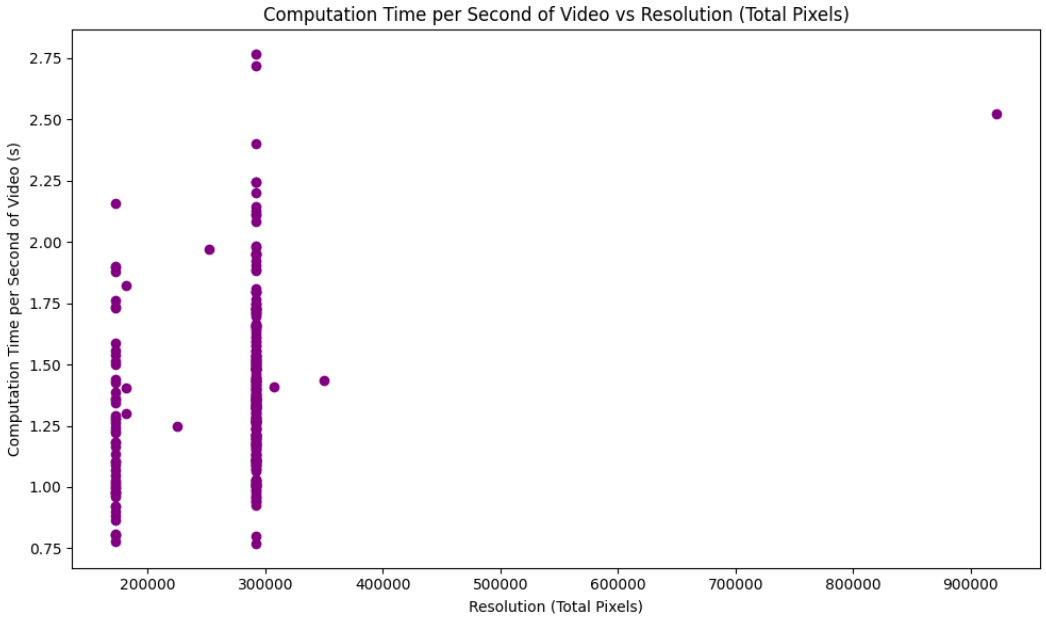


Figure 5.8: Computation Time per Second of Video vs. Resolution (Total Pixels).

processing demands under specific conditions. Extending this real-time capability to a broader range of videos is possible by optimizing the application's computational pipeline.

5.4.9.5 Challenges in Real-Time Performance and the Need for Cloud-Edge Integration

The evaluation of MocapMe's real-time performance reveals several limitations, particularly in processing high-resolution videos and ensuring scalability. While the framework demonstrates reliable and deterministic behavior for lower-resolution videos, the increased computational demand associated with higher resolutions introduces significant challenges in maintaining real-time performance. These challenges highlight the need for a more flexible architecture to handle the growing complexity of motion capture tasks.

Resolution and Scalability Challenges: High-resolution videos demand more computational resources, resulting in decreased frame rates and extended processing times. This limitation highlights the need for a distributed architecture that dynamically allocates tasks based on video complexity. Although MocapMe can process lower-resolution videos in near real-time, the system struggles with higher resolutions, which exceed its current processing capabilities.

The Cloud-Edge Solution: To address these constraints, integrating MocapMe within a cloud-edge architecture offers an effective solution. In this model, edge devices can manage real-time data acquisition and preliminary processing. At the same time, more computationally intensive tasks—such as advanced motion analysis and deep learning—are offloaded to cloud infrastructure. This task division optimizes resource use, reduces latency, and maintains real-time feedback even for high-resolution or complex video streams.

Scalability through Cloud-Edge Integration: The cloud-edge approach facilitates overcoming the computational challenges posed by high-resolution video processing and enhances the

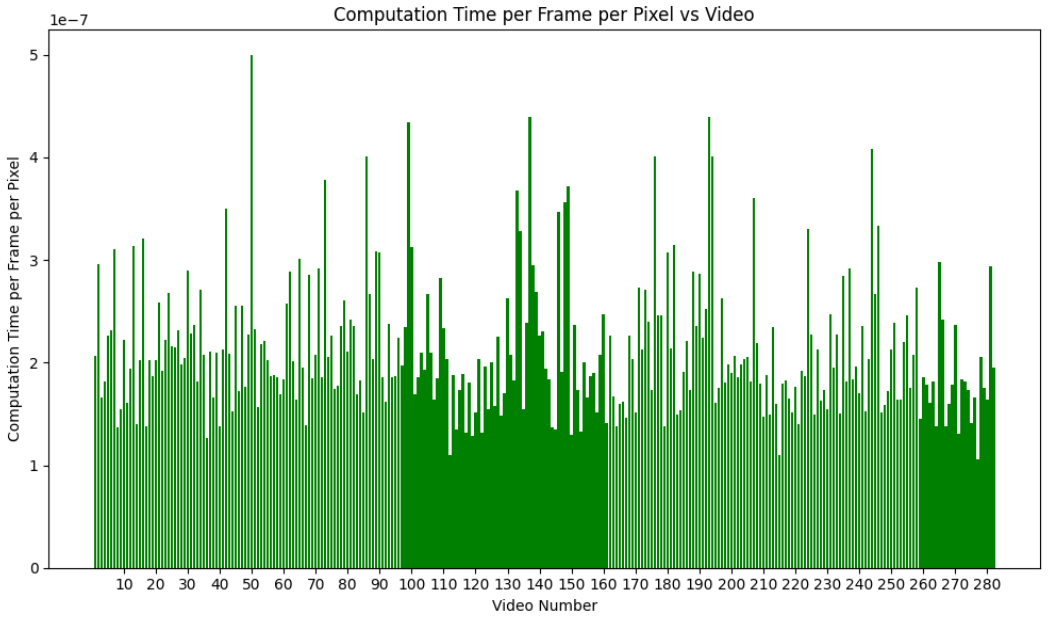


Figure 5.9: Computation Time per Frame per Pixel vs. Video.

system's scalability. MocapMe can handle multiple video streams and users concurrently by distributing workloads between edge devices and cloud servers. This scalability is particularly advantageous for applications such as clinical rehabilitation and sports performance monitoring, where real-time analysis of large data volumes is critical.

The following section explores the implementation of cloud-edge integration within MocapMe and its ability to support real-time, large-scale motion capture applications by leveraging the strengths of both edge and cloud computing.

5.5 Integration of Motion Capture in Edge Computing

Integrating MoCap technologies within edge computing architectures enables the real-time processing of the large data volumes generated by MoCap systems. The previous proposed approach leverages advanced markerless motion capture techniques, enhanced by deep learning models, to provide precise and efficient motion analysis [44]. The objective is to integrate with the Compute Continuum architecture to minimize latency by processing MoCap data at the edge, enabling real-time analytics and immediate feedback—crucial for scenarios such as fall detection in elderly care or performance monitoring in sports training. Additionally, edge deployment reduces reliance on high-bandwidth network connections, making this solution viable even in environments with limited connectivity.

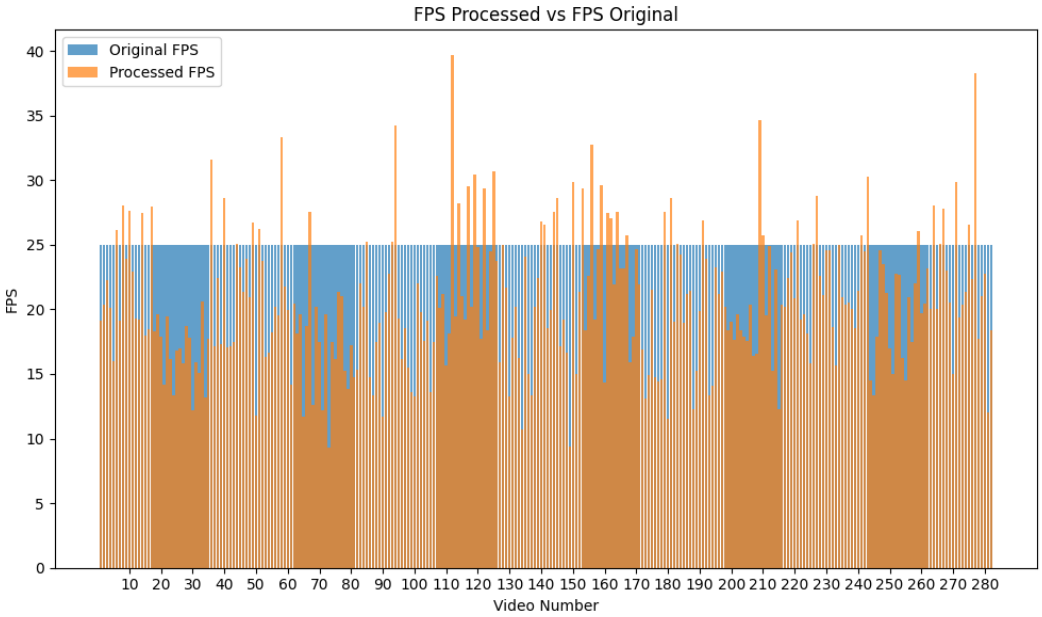


Figure 5.10: FPS Processed vs. FPS Original.

5.5.1 3D Motion Capture with MocapMe: Integration with Cloud-Edge Computing

Integrating cloud-edge computing paradigms with advanced MoCap technologies denotes a significant advancement in real-time data processing and analysis. This section presents the current state of research on enhancing 3D motion capture employing the MocapMe framework within a cloud-edge infrastructure. The deployment uses Raspberry Pi devices at the edge for video and image acquisition, while the cloud handles intensive processing tasks such as calibration and motion capture. This approach leverages the flexibility and scalability of cloud-edge architectures, particularly within the Stack4Things frameworks, to provide an efficient and robust solution for 3D motion capture.

5.5.1.1 System Architecture and Configuration

The new MocapMe system's architecture is designed to capitalize on the strengths of both edge and cloud computing. The system employs a dual-camera setup consisting of two Logitech C920 webcams (Full HD video with a full HD glass lens, 78° field of view, and HD auto light correction) strategically mounted on a 3D printed rigid frame in a binocular configuration 5.11. The cameras are connected to Raspberry Pi devices positioned at the network edge, enabling local video and image data acquisition. This edge-based configuration enables low-latency data capture, which is essential for maintaining temporal coherence in 3D motion capture.

Figure 5.11: 3D camera setup with Logitech C920 cameras and rigid frame.

5.5.1.2 Cloud-Edge Infrastructure

The core of the system's architecture consists of integrating the Stack4Things middleware, which extends the OpenStack ecosystem to manage IoT and edge devices. In this setup, Raspberry Pi devices serve as edge nodes, capturing video and image data, which is then transmitted to the cloud for processing. Appendix B explains the code running inside the Raspberry. The cloud environment employs OpenStack to handle calibration and motion capture processes. This separation of tasks ensures that computationally intensive operations are performed in the cloud, where resources are available. At the same time, real-time data acquisition occurs at the edge, minimizing latency and ensuring temporal accuracy.

5.5.1.3 Camera Calibration Process

Accurate 3D motion capture requires accurate camera system calibration, which involves estimating intrinsic and extrinsic parameters. The calibration process is critical to ensure the 3D reconstruction is accurate and reliable.

The calibration uses Aruco markers, known for their robustness in computer vision tasks. The calibration involves several key steps:

1. **Marker Placement:** A grid of Aruco markers is placed within the capture area. These markers are reference points for scaling and aligning the reconstructed 3D space.

2. **Image Capture:** The cameras, each connected to a Raspberry Pi, capture images of the Aruco marker grid. The spatial consistency between the two cameras is maintained throughout this process to ensure that the markers are visible in both camera frames.

3. **Parameter Estimation:** Using OpenCV's calibration tools, intrinsic parameters (e.g., focal lengths, lens distortions) and extrinsic parameters (e.g., relative positions and orientations

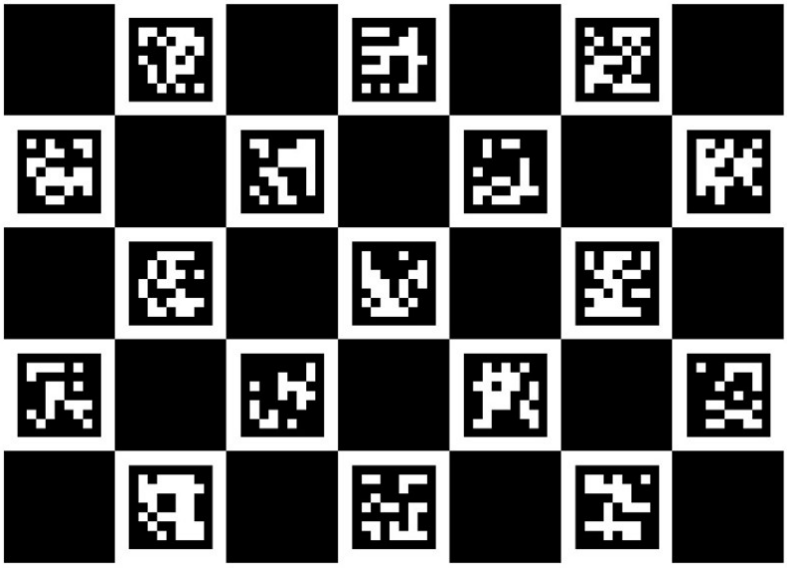


Figure 5.12: Aruco board used during the camera calibration process.

of the cameras) are computed. These parameters are essential for mapping 2D image coordinates to a standard 3D coordinate system.

4. **Validation:** The calibration is validated by calculating the reprojection error, which measures the discrepancy between the original 2D points and the projected 3D points. A low reprojection error indicates successful calibration, ensuring accurate depth perception in the 3D reconstruction.

5.5.1.4 Data Acquisition and Processing Workflow

The data acquisition process is initiated at the edge, where the dual Raspberry Pi devices capture synchronized frames from the Logitech C920 cameras. This setup maintains the temporal alignment between frames, which is crucial for accurate 3D reconstruction. The captured data is then transmitted to the cloud, where the MocapMe system processes it using a series of well-defined steps:

5.5.1.5 Keypoint Extraction

In the cloud, the MocapME framework is employed to extract keypoints from the 2D images. MocapME detects and tracks multiple anatomical points in real-time, providing 2D coordinates for each keypoint. These 2D coordinates constitute the basis for subsequent 3D reconstruction.

5.5.1.6 Data Refinement and Filtering and 3D Reconstruction

A filtering algorithm is implemented in the cloud to enhance the accuracy and reliability of the captured 3D data. This algorithm addresses common issues such as noise and measurement inaccuracies. The filtering process involves several key steps:

1. **Temporal Smoothing:** The coordinates of keypoints are smoothed across successive frames to mitigate the impact of noise and ensure that the motion appears continuous and natural.

2. **Consistency Check:** The algorithm checks for inconsistencies in the spatial relationships between keypoints. In cases of sudden jumps or anomalies, the algorithm corrects the position to maintain consistent tracking.

3. **Noise Reduction:** Outlier data points that deviate from the expected motion trajectory are identified and removed, reducing errors caused by occlusions or rapid movements.

The 2D keypoints extracted from the images are transformed into 3D coordinates using the calibration parameters obtained from the Aruco marker calibration process. A triangulation method is applied to compute the 3D positions of the keypoints based on the corresponding 2D points from both camera views. This process enables the generation of a precise representation of the subject's movements in three-dimensional space.

The refined 3D motion data is exported in the C3D format, a widely used standard in 3D analysis. This format supports various data types, including kinematic and kinetic information, and is compatible with numerous analysis tools and software platforms. This ensures that the captured data can be further analyzed, interpreted, and integrated into clinical or research applications.

The implementation of calibration and data Acquisition and Processing processes is provided in Appendix C.

5.5.1.7 Depth Accuracy

One of this system's significant advancements is its ability to capture depth information accurately, which is essential for reconstructing 3D motion. The dual-camera setup and the cloud processing power effectively reduce depth perception errors, resulting in more precise tracking of complex movements.

5.5.1.8 Occlusion Handling

The dual-camera configuration also addresses occlusion issues that commonly occur in single-camera setups. By capturing the subject from two different angles, the system maintains visibility of keypoints even when one camera's view is obstructed, thereby improving the overall reliability of motion capture.

5.5.1.9 Sequence Diagram of the Workflow

To exemplify the architecture depicted in 5.13 and workflow of the MocapMe system within the cloud-edge infrastructure, figure 5.14 illustrates the workflow of the MocapMe framework. The process begins with video data acquisition from two Logitech C920 cameras (User side) connected to Raspberry Pi devices. These edge devices capture and transmit the data to the cloud through the Stack4Things middleware. Once the cloud receives the data, the system calibers the camera using Aruco markers, as described earlier. Subsequently, the MocapMe framework extracts keypoints from the 2D images, filters them, and processes them to improve accuracy and stability. Finally, the keypoints are triangulated to produce 3D coordinates of the movement, and the final data is

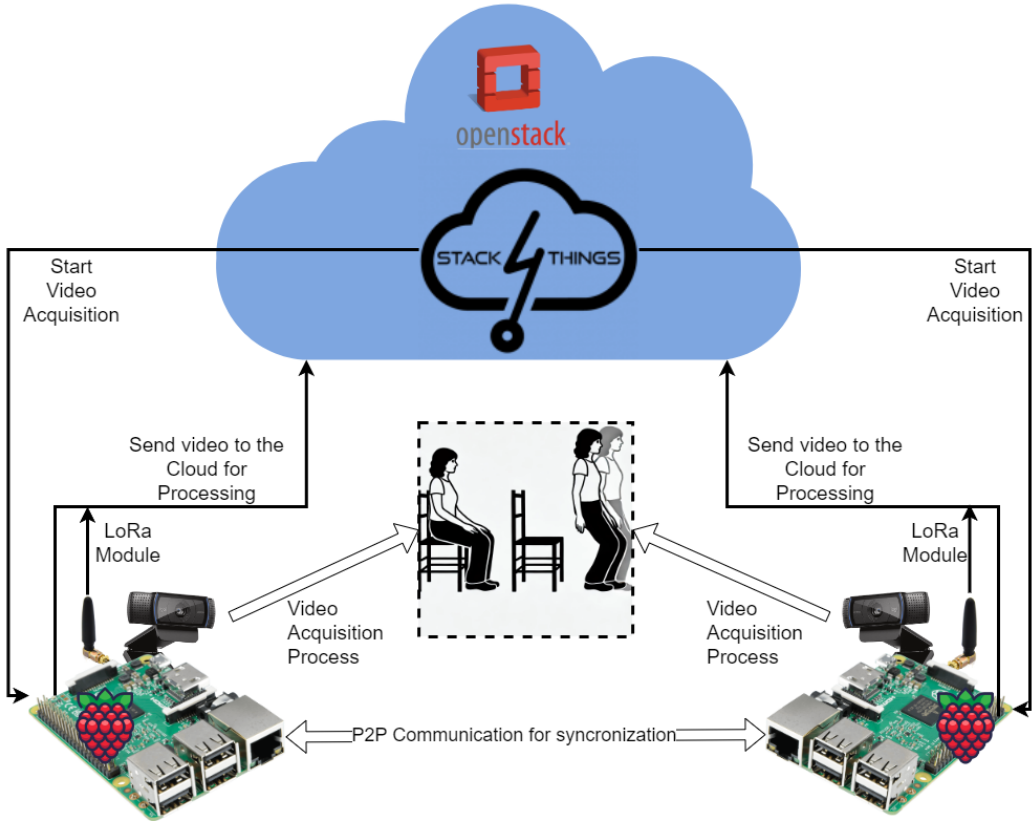


Figure 5.13: Architecture

exported in the C3D format for further analysis. The diagram demonstrates how each process phase is executed and which components are involved in each interaction.

5.5.2 Graphical User Interface (GUI)

The MocapMe framework includes a user-friendly Graphic User Interface (GUI) designed to facilitate interaction with the system. The GUI allows for tasks such as video calibration, motion analysis, and data visualization and provides an intuitive way for users to manage various aspects of the motion capture process.

The main interface (shown in Figure 5.15) consists of three primary options:

- **Calibration**: This option allows users to calibrate the cameras by capturing images, where they can specify the destination folder and the camera setup.
- **Video**: This option manages video acquisition for motion capture analysis.
- **Analysis**: This option provides motion data analysis tools access.

The calibration window (on the right in Figure 5.15) enables users to choose specific cameras and capture calibration images, ensuring the accuracy of the motion capture system's calibration process.

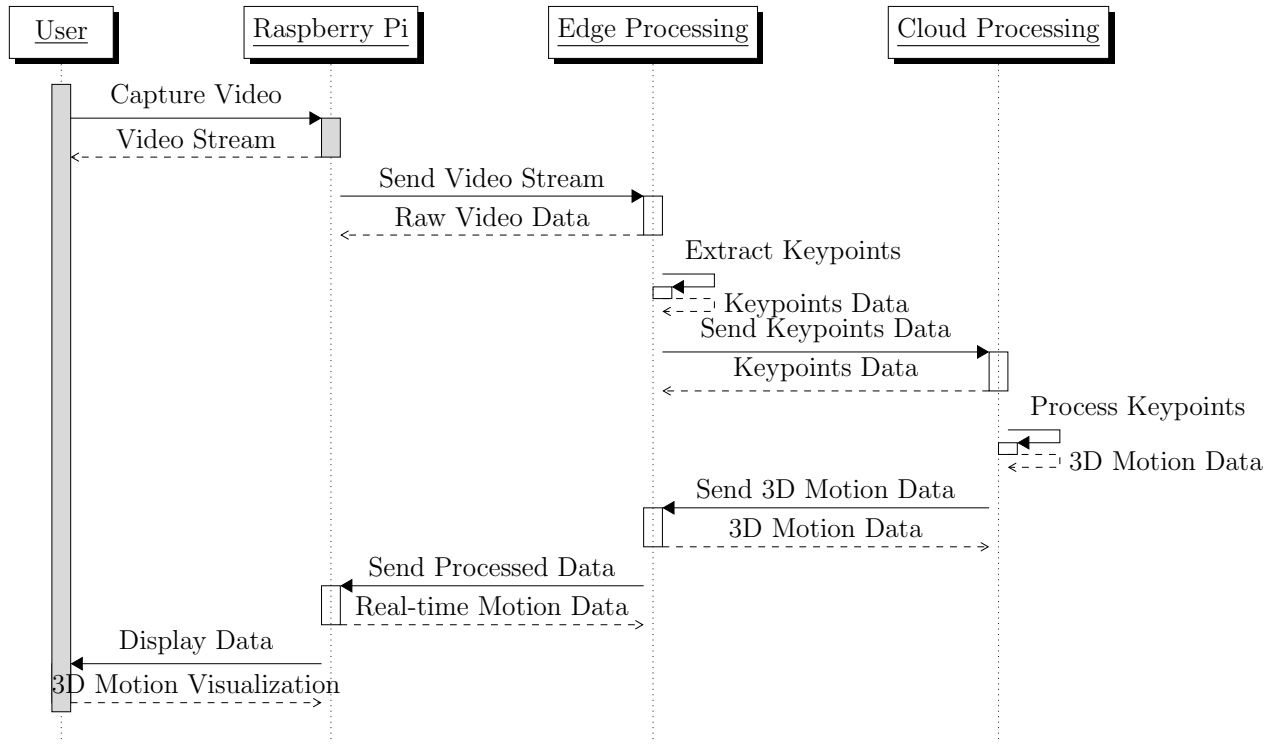


Figure 5.14: Sequence diagram of the MocapMe workflow

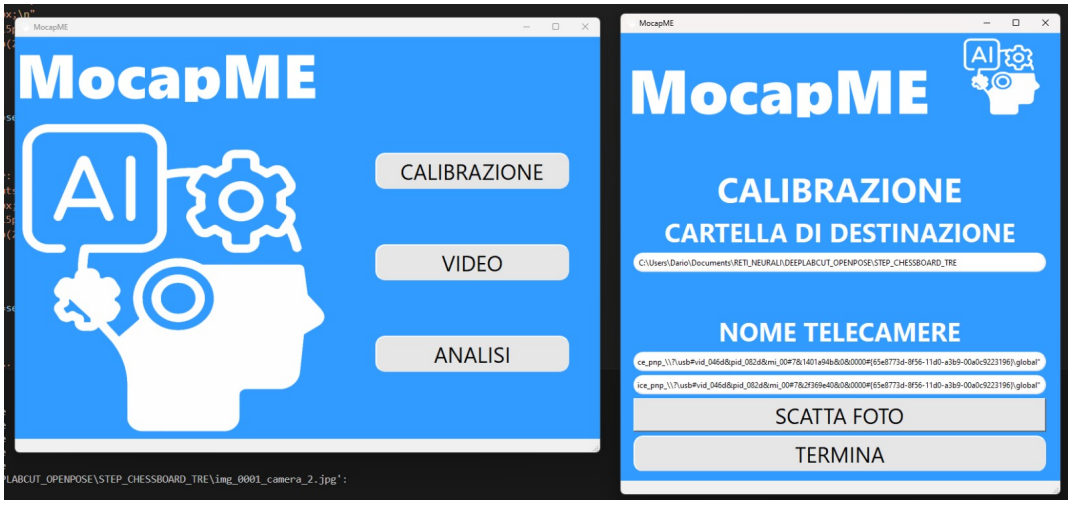


Figure 5.15: The MocapMe Graphical User Interface. The main window (left) offers options for calibration, video processing, and analysis, while the calibration window (right) allows users to set the destination folder, select cameras, and capture calibration images.

5.5.2.1 Lessons Learned

Integrating LoRa, P2P networking, and I/Ocloud within the MocapMe framework has provided valuable insights into designing and implementing distributed motion capture systems for bioengineering applications. Several key lessons have emerged from this research:

- **Scalability and Flexibility:** The deployment of LoRa and P2P technologies has demonstrated the importance of scalable and flexible communication architectures. These technologies allow motion capture systems to be deployed in many environments, from densely populated urban areas to remote rural settings, without requiring extensive infrastructure investments.
- **Resilience in Challenging Environments:** The robustness of LoRa in environments with limited connectivity and high interference has been critical in ensuring reliable data transmission. This resilience is particularly valuable in bioengineering applications where consistent data capture is essential for patient care and research accuracy.
- **Decentralized Processing:** P2P communication has underscored the benefits of decentralized data processing, enabling real-time analysis and reducing dependence on central servers. This approach enhances the system's fault tolerance and allows for more responsive and adaptive data management, which is crucial in dynamic clinical environments.
- **Comprehensive Data Integration:** The use of I/Ocloud has highlighted the need for comprehensive data management solutions that can handle diverse data types and workflows. Integrating multiple data streams within a unified platform has proven essential for delivering meaningful insights in bioengineering applications, where understanding the full context of motion data is critical.
- **Cost-Effectiveness:** The combination of these technologies has shown that distributed motion capture systems can be both cost-effective and efficient. By leveraging low-power, long-range communication, and decentralized data processing, it is possible to deploy sophisticated bioengineering applications without the need for prohibitively expensive infrastructure.

These lessons provide a foundation for future research and development, guiding the ongoing enhancement of motion capture systems and their applications in bioengineering.

Chapter 6

Conclusions and Future Developments

The research presented in this dissertation has delved into integrating cloud-edge computing across various application domains, beginning with an analysis of the main communication techniques commonly used in the IoT domain. Subsequently, the focus shifted towards applying these technologies in more complex scenarios, such as Smart Cities. Different solutions for collecting and processing heterogeneous data from distributed sensors were studied in this context, considering both edge and cloud processing approaches. These analyses have been instrumental in understanding how different architectural approaches affect network behavior and resource utilization.

Building on this foundation, the research explored specific bioengineering use cases. IoT devices were employed to collect data supporting clinical decisions, offering physicians advanced analytical tools to improve diagnosis and treatment processes. Furthermore, specific case studies were examined concerning the design of smart prostheses.

The WoT and the HATEOAS protocol analysis revealed a significant point of convergence between the cloud-edge architectures studied. This resulted in the development of a specific biomedical use case that provides remote smart health monitoring tools.

One of the key findings of this study is the ability of cloud-edge architectures to leverage highly distributed and powerful computational resources. This opened an additional research domain, focusing on data processing for advanced motion capture systems. The study of MoCap has enabled the acquisition of accurate data on human movement through markerless systems, and the research has led to improvements in widely used systems such as OpenPose and DeepLabCut.

A cloud-edge solution was proposed to ensure these results can be utilized in real-time. This solution allows real-time motion capture processing without the hardware limitations typical of traditional solutions. This approach enables the management of the substantial computational demands required to execute deep learning models without compromising performance.

These innovations can potentially transform healthcare delivery by enabling tailored treatments and improving clinical outcomes through more accurate and efficient diagnostic and therapeutic processes. However, the future presents numerous opportunities and significant challenges that must be addressed to ensure the success and scalability of these technologies in real-world applications.

6.1 Key Findings and Contributions

This thesis has demonstrated that integrating cloud-edge architectures enables more responsive, scalable, and efficient systems. One of the key contributions is developing the MocapMe framework, which leverages markerless motion capture technology, enhanced by deep learning, to provide high-precision movement analysis suitable for clinical and sports applications. By integrating this framework within a Compute Continuum architecture, latency is minimized, and real-time feedback is improved, providing significant benefits for time-sensitive tasks such as rehabilitation and athletic performance monitoring [44].

Further contributions include the exploration of enabling technologies such as LoRaWAN, P2P networking, and the I/Ocloud paradigm, which collectively provide a robust foundation for dynamic and scalable IoT ecosystems. These innovations offer solutions to data transmission and resource management challenges in distributed and resource-constrained environments [23, 42].

Moreover, this research has advanced the understanding of how cloud-edge infrastructures can be optimized through AI-driven resource management, federated learning, and blockchain for secure data handling. These contributions lay the groundwork for future systems, addressing both scalability and data security challenges [7, 18].

Several case studies have further enriched this work. In the domain of smart cities, for instance, the research explored how managed ELK deployments at the edge can efficiently process large-scale data from IoT devices distributed across urban infrastructures [7]. Another study focused on integrating IPA in hotel rooms for seamless IoT control through an SDI/O approach, automating the management of smart devices [60].

In the context of LoRaWAN mobility, a novel approach to enabling seamless roaming across heterogeneous networks was proposed, allowing for more dynamic and flexible deployments without requiring predefined agreements between Network Servers [42].

The research also explored telemedicine applications, where the WoT paradigm was used to manage IoT devices in healthcare settings, allowing for remote monitoring of patient health parameters [18]. Similarly, innovations in 3D printing and blockchain technologies were applied to address supply chain issues during emergency responses, demonstrating the utility of distributed manufacturing in crisis situations [22].

Additionally, the research focused on designing smart prostheses using additive manufacturing techniques, combining numerical and experimental methodologies to enhance the customization and functionality of the designs [46]. This was further complemented by studies on the quantification of clinician-controlled preload in dental implants, providing insights into how manual tightening torques influence complication rates [45].

These findings and the development of innovative cloud-edge solutions and motion capture systems significantly contribute to advancements in engineering and scientific fields, with potential applications in bioengineering and medical domains.

6.2 Future Challenges and Opportunities

Despite these advancements, several challenges remain. The rapid evolution of cloud-edge computing technologies will require continuous optimization to meet the growing demands of healthcare applications. The need for scalable, low-latency systems to manage increasingly complex datasets

is a pressing problem, mainly as more devices generate extensive amounts of real-time data in healthcare environments.

6.2.1 Enhancing Performance and Scalability

Future research should focus on developing hierarchical edge computing architectures that distribute computational loads across multiple tiers of edge nodes. This approach would enable more efficient resource allocation, reducing latency and improving the performance of data-intensive applications like motion capture. AI-driven resource management holds promise in this regard, enabling systems to predict and adapt to real-time demands, optimizing both computational and network resources.

Moreover, the development of lightweight AI models and energy-efficient hardware for edge devices will be essential in expanding the capabilities of edge computing, particularly for applications requiring real-time analytics and low-power consumption, such as wearable health monitors and portable diagnostic tools.

6.2.2 Security and Ethical Considerations

As cloud-edge systems become more integrated into healthcare, data security and privacy will remain paramount. Blockchain technology offers a promising solution for secure and transparent data management, though scalability challenges must be addressed to make it viable in high-throughput environments. The use of secure multi-party computation (SMPC) and homomorphic encryption should also be explored to ensure that sensitive healthcare data can be processed securely at the edge without exposing raw data to external threats.

Ethical considerations are equally important. As AI systems become more autonomous, ensuring transparency, accountability, and fairness in decision-making processes will be essential. Addressing biases in AI algorithms and ensuring that patients retain control over their data through informed consent are critical steps toward building trust in these technologies.

6.3 Implications for Bioengineering

The research presented in this thesis has notable implications for the broader field of bioengineering. Specifically, integrating cloud-edge computing with IoT-enabled technologies offers a scalable, adaptable infrastructure that supports various applications. These range from remote health monitoring to advanced diagnostic systems, where latency reduction and real-time data processing are crucial to improving healthcare workflows. By enabling real-time feedback and minimizing delays, the integration of cloud-edge infrastructures empowers healthcare providers with timely, actionable insights into patient health, enhancing clinical decision-making [18].

One of the key contributions to this field is demonstrated through the development of the MocapMe framework. This markerless motion capture system, enhanced by deep learning algorithms such as DeepLabCut, enables precise movement analysis suitable for rehabilitation and sports science. The focus on the STS motion highlights the importance of functional movement assessments in clinical applications. By leveraging the Compute Continuum architecture, MocapMe

reduces processing times and enhances the accuracy of real-time assessments, which is essential in time-sensitive medical contexts such as post-operative rehabilitation [44].

Moreover, cloud-edge technologies are particularly relevant in bioengineering applications involving remote or underdeveloped areas. As presented in this research, exploring enabling technologies like LoRaWAN and P2P networking offers scalable solutions for resource-constrained environments. These technologies can be instrumental in telemedicine, where continuous monitoring and data transmission are essential for managing patient health remotely. Collecting and processing data in real-time allows for continuous monitoring, even in areas with limited access to traditional healthcare facilities [23].

This research also addresses key concerns in data security and management. Incorporating blockchain technology for secure data handling within healthcare applications ensures that patient data remains protected across distributed systems. By providing a decentralized and transparent infrastructure, blockchain enhances the reliability and security of telemedicine platforms, addressing common concerns about data integrity in cloud-based healthcare solutions [22].

These technologies could pave the way for even more decentralized and autonomous systems in the future, particularly in bioengineering. The advancement of AI-driven resource management, federated learning, and real-time analytics will further enhance the scalability and efficiency of healthcare systems. These developments will enable personalized, efficient care models that respond dynamically to patient needs, particularly in rehabilitation and functional assessments, where real-time feedback is crucial.

6.4 Future Directions

Several areas warrant further exploration:

6.4.1 Advanced AI and Machine Learning Integration

AI will play an increasingly central role in the future of bioengineering systems. Federated learning, in particular, offers a way to improve machine learning models without compromising data privacy. Future research should explore the use of federated learning to enable continuous model updates across distributed healthcare environments, enhancing the accuracy and efficiency of diagnostic algorithms while maintaining privacy.

6.4.2 Blockchain for Secure Data Management

Blockchain can significantly enhance the security and transparency of healthcare data management. Future research should investigate lightweight blockchain solutions tailored for cloud-edge environments, focusing on reducing computational overhead and improving scalability without compromising security.

6.4.3 Scalability and Interoperability Solutions

As cloud-edge systems continue to grow, ensuring scalability and interoperability will be necessary. Open standards and protocols must be developed to allow seamless integration between different

devices and platforms, enabling widespread adoption of these technologies in healthcare. Research should also focus on developing resource-efficient architectures that can dynamically scale to meet the growing demands of bioengineering applications.

6.4.4 Ethical and Regulatory Frameworks

Finally, ethical and regulatory frameworks must evolve in tandem with technological advancements. Developing clear guidelines for the ethical use of AI and IoT in healthcare will be critical to ensuring that these technologies are deployed responsibly, safeguarding patient privacy and autonomy. Future research should contribute to the creation of standards that balance innovation with ethical considerations, ensuring that cloud-edge technologies are used to benefit all patients equitably.

6.5 Concluding Remarks

This thesis has provided a foundation for the integration of cloud-edge computing, AI, and IoT in bioengineering applications, demonstrating their transformative potential for healthcare. However, continued research and interdisciplinary collaboration will be essential to address the challenges ahead and ensure that these technologies are deployed responsibly and effectively.

The advancements presented here represent only the beginning of what is possible. By continuing to explore new technologies and approaches, researchers and practitioners can develop solutions that not only improve patient outcomes but also advance the field of bioengineering as a whole.

Appendix A

Appendix A: Cloud Processing for Motion Capture Framework

This appendix details the cloud-side processing in the motion capture framework developed in this thesis. The system leverages cloud computing to handle the bulk of the data processing, focusing on video processing, keypoint extraction, and biomechanical analysis using custom Python scripts and the DeepLabCut (DLC) model that was trained using the method described in Appendices B and C. The cloud module ensures efficiency and accuracy in analyzing motion data, reducing the computational burden on edge devices.

A.0.1 Overview of Cloud-Based Processing

The cloud-based processing component extracts keypoints from pre-processed video data, filters noisy detections, and performs biomechanical analysis. It integrates several modules, including:

- **Keypoint Extraction**: Pre-trained models (such as the one described in Appendix B) are deployed in the cloud to extract keypoints from videos.
- **Data Filtering and Smoothing**: To ensure high data quality, the system applies filtering algorithms to remove low-confidence keypoints and smoothing algorithms to reduce noise in the motion data.
- **Angle Calculation**: The system computes joint angles for biomechanical analysis, essential for assessing motion patterns like the Sit-to-Stand (STS) task.

The code below is responsible for these tasks, focusing on computational efficiency and scalability in a cloud environment.

A.0.2 Keypoint Extraction

The following Python script handles the core functionality of extracting keypoints from videos using a pre-trained DeepLabCut model. The model is applied to each input video frame, generating 2D keypoint coordinates for each detected body part.

Listing A.1: Keypoint Extraction

```
import os
import pandas as pd
import cv2
import glob

# Function to load video data and process it using a pre-trained model
def load_data(video_path, output_path, model_config):
    # Load pre-trained DeepLabCut model
    import deeplabcut
    deeplabcut.analyze_videos(model_config, [video_path], save_as_csv=True, destfolder=outp
    # Output CSV file contains keypoint data for each frame of the video
%
```

This function loads the video from the specified path, applies the pre-trained model to each frame, and exports the results as a CSV file containing the coordinates of each keypoint.

A.0.3 Data Filtering and Smoothing

Once keypoints are extracted, the next step is to filter out low-confidence detections and smooth the data to ensure robust analysis. Below is the code that applies filtering based on a confidence threshold and smooths the motion data using a Savitzky-Golay filter.

Listing A.2: Data Filtering and Smoothing

```
from scipy.signal import savgol_filter

# Function to filter out low-confidence keypoints and apply smoothing
def filter_and_smooth_data(input_csv, confidence_threshold=0.95, window_length=11, polyorde
    # Load keypoint data
    df = pd.read_csv(input_csv)

    # Filter keypoints with confidence below the threshold
    filtered_df = df[df['confidence'] >= confidence_threshold]

    # Apply Savitzky-Golay filter for smoothing
    for col in filtered_df.columns:
        if 'x' in col or 'y' in col:  # Apply smoothing only to coordinates
            filtered_df[col] = savgol_filter(filtered_df[col], window_length=window_length

    # Save the smoothed and filtered data
    filtered_df.to_csv('smoothed_' + input_csv, index=False)
```

This function processes the CSV file generated by the keypoint extraction step, removing keypoints with confidence values below the set threshold. It then applies the Savitzky-Golay filter to smooth the keypoint trajectories, ensuring the final data is noise-free.

A.0.4 Angle Calculation for Biomechanical Analysis

Joint angles are crucial metrics for evaluating biomechanical movements such as the Sit-to-Stand task. The following code computes angles between three consecutive keypoints: the shoulder, elbow, and wrist.

Listing A.3: Angle Calculation

```
import numpy as np

# Function to calculate angles between three consecutive keypoints
def calculate_angles(keypoints_csv):
    df = pd.read_csv(keypoints_csv)
    angles = []

    # Iterate over frames and calculate angles for each
    for i, row in df.iterrows():
        # Define three points (e.g., shoulder, elbow, wrist)
        A = np.array([row['x1'], row['y1']])
        B = np.array([row['x2'], row['y2']])
        C = np.array([row['x3'], row['y3']])

        # Calculate vectors
        AB = B - A
        BC = C - B

        # Compute angle between vectors
        cos_angle = np.dot(AB, BC) / (np.linalg.norm(AB) * np.linalg.norm(BC))
        angle = np.degrees(np.arccos(np.clip(cos_angle, -1.0, 1.0)))

        angles.append(angle)

    # Save angles to a CSV file
    angle_df = pd.DataFrame(angles, columns=['angle'])
    angle_df.to_csv('angles.csv', index=False)
```

This function calculates the angles between three points for each video frame. The results are stored in a CSV file, which can be used for further analysis, such as evaluating the motion dynamics of joints during the Sit-to-Stand task.

A.0.5 Integration with Cloud Storage and APIs

The cloud-based system uses APIs to transfer processed data between edge devices and cloud storage. The processed CSV files, containing keypoints and angles, are uploaded to the cloud for real-time analysis or offline review.

Listing A.4: Upload Processed Data to Cloud Storage

```
import requests

# Function to upload processed data to cloud storage
def upload_to_cloud(file_path, cloud_url):
    with open(file_path, 'rb') as f:
        r = requests.post(cloud_url, files={'file': f})
        if r.status_code == 200:
            print("File uploaded successfully.")
        else:
            print("File upload failed.")
```

This function uploads the processed CSV data to the cloud, enabling real-time analysis and integration with higher-level services, such as clinical monitoring systems or motion analysis dashboards.

Appendix B

Appendix: Python Code for Synchronous Video Acquisition and Cloud Upload

This appendix provides the Python implementation of two Raspberry Pi devices for synchronous video acquisition using WebRTC. Both Raspberry Pi devices simultaneously capture video in raw format, compress it into MP4, and upload it to the cloud. If TCP connectivity to the cloud is unavailable, the system employs the LoRa protocol to ensure the video is transmitted.

Listing B.1: Synchronous Video Acquisition

```
import asyncio
import websockets
import subprocess
import requests
import logging
import os
import time
import serial  # For LoRa communication

# Setup logging for debugging and error handling
logging.basicConfig(level=logging.INFO)

# Video file paths
raw_video_path = "/tmp/video_raw.yuv"
mp4_video_path = "/tmp/video_encoded.mp4"

# API endpoint for cloud upload
CLOUD_API_URL = "http://cloud-server/api/upload"

# LoRa setup (LoRa module via serial)
LORA_PORT = "/dev/ttyUSB0"
```

```
LORA_BAUDRATE = 9600

# Initialize LoRa communication
def init_lora():
    try:
        lora = serial.Serial(LORA_PORT, LORA_BAUDRATE, timeout=10)
        logging.info("LoRa module initialized")
        return lora
    except Exception as e:
        logging.error(f"Error initializing LoRa: {e}")
        return None

# Synchronization function to ensure both Raspberry Pi start at the same time
async def synchronize_acquisition():
    async with websockets.connect('ws://raspberry2.local:8765') as websocket:
        await websocket.send("SYNC_START")
        logging.info("Video acquisition synchronized, starting now.")
        return True

# Function to capture and save raw video locally
def capture_video():
    logging.info("Starting video capture...")
    ffmpeg_command = [
        'ffmpeg', '-f', 'v4l2', '-framerate', '30', '-video_size', '1920x1080',
        '-i', '/dev/video0', raw_video_path
    ]
    subprocess.run(ffmpeg_command)
    logging.info(f"Raw video saved to {raw_video_path}")

# Function to compress raw video into MP4 using system's ffmpeg
def compress_video():
    try:
        logging.info("Compressing video to MP4 format...")
        ffmpeg_command = [
            'ffmpeg', '-f', 'rawvideo', '-pixel_format', 'rgb24', '-video_size', '1920x108
            '-framerate', '30', '-i', raw_video_path, '-c:v', 'libx264', mp4_video_path
        ]
        subprocess.run(ffmpeg_command, check=True)
        logging.info(f"Compression complete. MP4 saved to {mp4_video_path}")
    except subprocess.CalledProcessError as e:
        logging.error(f"Error during video compression: {e}")

# Function to upload the compressed video to the cloud via TCP (HTTP API)
def upload_to_cloud():
    try:
```

```
        logging.info(f"Uploading {mp4_video_path} to cloud via TCP...")
        with open(mp4_video_path, 'rb') as video_file:
            files = {'file': video_file}
            response = requests.post(CLOUD_API_URL, files=files, timeout=10)
        if response.status_code == 200:
            logging.info("Video uploaded successfully via TCP.")
        else:
            logging.error(f"Failed to upload video via TCP. Status code: {response.status_c
    except (requests.ConnectionError, requests.Timeout) as e:
        logging.error(f"TCP connection to cloud failed: {e}")
        logging.info("Falling back to LoRa for video upload...")
        upload_to_cloud_via_lora()

# Function to upload the compressed video to the cloud via LoRa
def upload_to_cloud_via_lora():
    lora = init_lora()
    if lora:
        try:
            logging.info("Starting LoRa transmission...")
            with open(mp4_video_path, 'rb') as video_file:
                while True:
                    chunk = video_file.read(128)  # Sending small chunks over LoRa
                    if not chunk:
                        break
                    lora.write(chunk)
                    time.sleep(0.2)  # Small delay to ensure stable transmission
            logging.info("Video uploaded successfully via LoRa.")
        except Exception as e:
            logging.error(f"Error during LoRa transmission: {e}")
        finally:
            lora.close()
    else:
        logging.error("LoRa module not available. Unable to upload video.")

# Main function for Raspberry Pi 1 and Pi 2
async def main():
    # Synchronize video acquisition across both Raspberry Pi devices
    await synchronize_acquisition()

    # Capture raw video
    capture_video()

    # Compress the video to MP4
    compress_video()
```

```
    # Upload video to the cloud
    upload_to_cloud()

if __name__ == "__main__":
    asyncio.run(main())
```

B.0.1 Code Explanation

This implementation synchronizes the video acquisition between two Raspberry Pi devices, simultaneously ensuring both start and end capturing. Each Raspberry Pi compresses the video to MP4 and uploads it to the cloud. If TCP connectivity to the cloud fails, the system employs LoRa for transmission.

Synchronization of Acquisition The ‘synchronize_acquisition‘ function ensures that both Raspberry Pi devices start the video capture at the same time. This is achieved via WebRTC communication to send a synchronization signal.

Raw Video Capture and Compression Each Raspberry Pi captures raw video using the system’s ‘ffmpeg‘, which is saved locally. After the acquisition, the video is compressed into MP4 format for efficient uploading to the cloud.

Cloud Upload and LoRa Fallback The primary upload mechanism is through a standard TCP connection (HTTP POST). If the TCP connection fails, the system uses LoRa as a fallback mechanism to ensure the video is still transmitted. LoRa transmission is optimized by sending the video in small chunks with slight delays between each transmission to ensure reliable delivery, even over low-bandwidth networks.

Error Handling and Logging The code includes robust logging and error handling to track the state of the video acquisition, compression, and upload processes, ensuring smooth operation and easy debugging in a production environment.

Appendix C

Appendix: Python Code for Camera Calibration and 3D Reconstruction

This appendix provides the Python implementation for camera calibration using Charuco or Checkerboard boards and 3D reconstruction. The script is designed to process multiple videos captured from different camera angles, triangulate the points, and perform a 3D reconstruction. Additionally, the code includes options for smoothing the 3D data, calculating joint angles, and exporting the results in CSV and C3D formats.

Listing C.1: Camera Calibration and 3D Reconstruction

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from aniposelib.boards import CharucoBoard, Checkerboard
from aniposelib.cameras import CameraGroup
from aniposelib.utils import load_pose2d_fnames
import ezc3d
from scipy.interpolate import interp1d
from scipy.signal import savgol_filter

# Function for 3D reconstruction and saving C3D output
def recostruction_3d(
        vidnames,
        fps=60,
        nomi_punti=['P_0', 'P_1', 'P_2', 'P_3', 'P_4'],
        unit_measure=['mm'],
        angles_degrees=[-90, 0, 0],
        traslation=[0, 0, 0],
        cam_names=['A', 'B'],
        board_dim=[46, 32],
        square=25,
```

```
    fname_dict=None,
    score_threshold=0.5,
    c3d_file_path="output-2.c3d",
    csv_file_path="output-2.csv",
    toml_path="calibration_result.toml",
    aruco=False,
    calibration=True,
    C3D_FILE=True,
    smooth=True,
    smoothed_output_csv="smoothed_output.csv",
    calcolo_angoli=True,
    angoli_csv="angoli.csv",
    c3d_file_path_smooth="smoothed_output.c3d",
    marker_length=30):

# Load or calibrate the camera group
if calibration:
    board = CharucoBoard(board_dim[0], board_dim[1], square_length=square, marker_lengt
    cgroup = CameraGroup.from_names(cam_names, fisheye=True)
    cgroup.calibrate_videos(vidnames, board)
    cgroup.dump(toml_path)
else:
    cgroup = CameraGroup.load(toml_path)

# Load 2D points from h5 files
d = load_pose2d_fnames(fname_dict, cam_names=cgroup.get_names())
points = d['points']
scores = d['scores']
points[scores < score_threshold] = np.nan  # Filter points based on score threshold

# Triangulate 3D points
points_flat = points.reshape(len(cam_names), -1, 2)
p3ds_flat = cgroup.triangulate(points_flat, progress=True)

# Prepare 3D points data for saving
p3ds = p3ds_flat.reshape(len(d['points']), len(d['bodyparts']), 3)

# Rotate and translate the points
rotated_points = np.array([rotate_and_translate(point, angles_degrees, traslation) for

# Save as CSV for further processing or analysis
matrix_2d = pd.DataFrame(rotated_points.reshape(-1, len(nomi_punti)*3))
matrix_2d.to_csv(csv_file_path)

# Optionally create C3D file
```

```
    if C3D_FILE:
        create_c3d_from_csv(csv_file_path, unit_measure, fps, nomi_punti, c3d_file_path)

    # Apply smoothing if needed
    if smooth:
        smoothed_df = smooth_data(matrix_2d)
        smoothed_df.to_csv(smoothed_output_csv)
        create_c3d_from_csv(smoothed_output_csv, unit_measure, fps, nomi_punti, c3d_file_pa

    # Calculate angles
    if calcolo_angoli:
        angoli = calculate_angles_3d(smoothed_df)
        np.savetxt(angoli_csv, angoli, delimiter=";")
        plot_angles(angoli)

# Helper functions
def rotate_and_translate(point, angles_degrees, translation):
    """Apply 3D rotation and translation to a point."""
    angles_radians = np.radians(angles_degrees)
    point = np.array(point)

    # Rotation matrices
    rot_x = np.array([[1, 0, 0], [0, np.cos(angles_radians[0]), -np.sin(angles_radians[0])]
    rot_y = np.array([[np.cos(angles_radians[1]), 0, np.sin(angles_radians[1])], [0, 1, 0]
    rot_z = np.array([[np.cos(angles_radians[2]), -np.sin(angles_radians[2]), 0], [np.sin(a

    # Apply rotations
    rotated_point = rot_z @ (rot_y @ (rot_x @ point))

    # Apply translation
    return rotated_point + np.array(translation)

def create_c3d_from_csv(csv_file, unit_measure, fps, nomi_punti, c3d_file_path):
    """Create a C3D file from a CSV containing 3D points."""
    df = pd.read_csv(csv_file)
    p3ds = df.to_numpy().reshape(df.shape[0], len(nomi_punti), 3).transpose(2, 1, 0)

    c3d = ezc3d.c3d()
    c3d['parameters']['POINT']['UNITS']['value'] = unit_measure
    c3d['parameters']['POINT']['RATE']['value'] = [fps]
    c3d['parameters']['POINT']['LABELS']['value'] = nomi_punti
    c3d['data']['points'] = p3ds
    c3d.write(c3d_file_path)

def smooth_data(df):
```

```
    """Apply smoothing to the 3D data using Savitzky-Golay filter."""
    smoothed_df = df.copy()
    for col in df.columns:
        smoothed_df[col] = savgol_filter(df[col], window_length=11, polyorder=3)
    return smoothed_df

def calculate_angles_3d(df):
    """Calculate 3D angles between adjacent points."""
    num_points = df.shape[1] // 3
    angles = np.zeros((df.shape[0], num_points - 2))

    for i in range(num_points - 2):
        A = df.iloc[:, 3*i:3*i+3].values
        B = df.iloc[:, 3*(i+1):3*(i+1)+3].values
        C = df.iloc[:, 3*(i+2):3*(i+2)+3].values
        AB = B - A
        BC = C - B
        dot_product = np.sum(AB * BC, axis=1)
        norm_AB = np.linalg.norm(AB, axis=1)
        norm_BC = np.linalg.norm(BC, axis=1)
        cos_theta = dot_product / (norm_AB * norm_BC)
        angles[:, i] = np.degrees(np.arccos(np.clip(cos_theta, -1.0, 1.0)))

    return angles

def plot_angles(angles):
    """Plot angles over time."""
    for i in range(angles.shape[1]):
        plt.plot(angles[:, i], label=f'Angle {i+1}')
    plt.xlabel('Frames')
    plt.ylabel('Angle (degrees)')
    plt.title('Joint Angles Over Time')
    plt.legend()
    plt.show()

# Example usage
if __name__ == "__main__":
    vidnames = [
        ["camera_1_video.mp4"],
        ["camera_2_video.mp4"]
    ]

    fname_dict = {
        'Cam_1': 'cam_1_h5_file.h5',
        'Cam_2': 'cam_2_h5_file.h5',
```

```
}

recostruction_3d(
    vidnames=vidnames,
    fname_dict=fname_dict,
    toml_path="calibration_result.toml",
    calibration=True
)
```

C.0.1 Code Explanation

Camera Calibration and 3D Reconstruction: The script uses videos from different cameras to perform camera calibration based on a checkerboard or Charuco board. It then triangulates the 3D points from the detected 2D keypoints in the video frames.

3D Point Rotation and Translation: The function `rotate_and_translate()` applies 3D rotations and translations to adjust the camera coordinate system based on the calibration setup.

Smoothing: The 3D data can be smoothed using the Savitzky-Golay filter, which helps reduce noise in the triangulated points.

Angle Calculation: The script computes the angles between three consecutive points (e.g., joints) using the 3D coordinates. This is useful for analyzing joint movements and understanding the kinematics of motion captured from the videos. The calculated angles are plotted over time to visualize how they change during the motion sequence.

C3D File Export: If the C3D file export option is enabled, the script generates a C3D file.

CSV Export: The script exports the reconstructed 3D coordinates in CSV format for further processing or analysis. Additionally, smoothed versions of the data can be saved if the smoothing option is activated.

Bibliography

[1] Amos Azaria and Jason Hong. "Recommender Systems with Personality". In: *Proceedings of the 10th ACM Conference on Recommender Systems*. RecSys '16. Boston, Massachusetts, USA: Association for Computing Machinery, 2016, pp. 207–210. ISBN: 9781450340359. DOI: 10.1145/2959100.2959138.

[2] Prith Banerjee et al. "Everything as a service: Powering the new information economy". In: *Computer* 44.3 (2011), pp. 36–43.

[3] Emmanuele Barberi et al. "Fast Three-Dimensional Posture Reconstruction of Motorcyclists Using OpenPose and a Custom MATLAB Script". In: *Sensors* 23.17 (2023). ISSN: 1424-8220. DOI: 10.3390/s23177415. URL: https://www.mdpi.com/1424-8220/23/17/7415.

[4] Daniel Beimborn, Thomas Miletzki, and Stefan Wenzel. "Platform as a service (PaaS)". In: *Wirtschaftsinformatik* 53 (2011), pp. 371–375.

[5] Zakaria Benomar et al. "Deviceless: A serverless approach for the Internet of Things". In: *2021 ITU Kaleidoscope: Connecting Physical and Virtual Worlds (ITU K)*. IEEE. 2021, pp. 1–8.

[6] Zakaria Benomar et al. "Extending openstack for cloud-based networking at the edge". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE. 2018, pp. 162–169.

[7] Zakaria Benomar et al. "Managed ELK deployments at the Edge with OpenStack and IoTronic: an italian Smart City case study". In: *2022 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE. 2022, pp. 287–292.

[8] Adhitya Bhawiyuga et al. "Architectural design of IoT-cloud computing integration platform". In: *TELKOMNIKA (Telecommunication Computing Electronics and Control)* 17.3 (2019), pp. 1399–1408.

[9] Adnane Boukhayma and Edmond Boyer. "Surface motion capture animation synthesis". In: *IEEE transactions on visualization and computer graphics* 25.6 (2018), pp. 2270–2283.

[10] Dario Bruneo et al. "I/Ocloud: Adding an IoT dimension to cloud infrastructures". In: *Computer* 51.1 (2018), pp. 57–65.

[11] Ismail Butun, Nuno Pereira, and Mikael Gidlund. "Analysis of LoRaWAN v1.1 Security: Research Paper". In: *Proceedings of the 4th ACM MobiHoc Workshop on Experiences with the Design and Implementation of Smart Objects*. SMARTOBJECTS '18. Los Angeles, California: Association for Computing Machinery, 2018. ISBN: 9781450358576. DOI: 10.1145/3213299.3213304. URL: https://doi.org/10.1145/3213299.3213304.

[12] Z. Cao et al. "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[13] Zhe Cao et al. "Realtime multi-person 2d pose estimation using part affinity fields". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7291–7299.

[14] R. D. Center. *RAK5146 WisLink LPWAN Concentrator*. 2023. URL: https://docs.rakwireless.com/Product-Categories/WisLink/RAK5146/ (visited on 09/13/2023).

[15] *ChirpStack, open-source lorawan® network server stack*. URL: https://www.chirpstack.io/.

[16] Giovanni Cicceri et al. "A Deep Learning-Driven Self-Conscious Distributed Cyber-Physical System for Renewable Energy Communities". In: *Sensors* 23.9 (2023), p. 4549.

[17] Hybrid Cloud. "The nist definition of cloud computing". In: *National institute of science and technology, special publication* 800.2011 (2011), p. 145.

[18] Luca D'Agati et al. "Cloud-based Web of Things: A Telemedicine Use Case". In: *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*. IEEE. 2023, pp. 1–6.

[19] Luca D'Agati et al. "Enhancing UAV Operational Efficiency through Cloud Computing and Autopilot System Integration". In: *2024 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE. 2024, pp. 85–92.

[20] Luca D'Agati et al. "IoT/cloud-powered crowdsourced mobility services for green smart cities". In: *2021 IEEE 20th international symposium on network computing and applications (NCA)*. IEEE. 2021, pp. 1–8.

[21] Luca D'Agati et al. "Seamless Remote Roaming Activation in Lorawan Via an Api-Driven Gateway Bridge Service". In: *Available at SSRN 4883223* ().

[22] Luca D'Agati et al. "3D Printing and Blockchains for an Emergency Response Supply Chain". In: *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE. 2023, pp. 255–260.

[23] Luca D'Agati et al. "BLE-Enabled On-Site Diagnostics For An IoT/Cloud-Controlled Energy Substation". In: *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE. 2021, pp. 329–334.

[24] Allan de Barcelos Silva et al. "Intelligent personal assistants: A systematic literature review". In: *Expert Systems with Applications* 147 (2020), p. 113193. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113193.

[25] Tharam Dillon, Chen Wu, and Elizabeth Chang. "Cloud computing: issues and challenges". In: *2010 24th IEEE international conference on advanced information networking and applications*. Ieee. 2010, pp. 27–33.

[26] Salvatore Distefano, Giovanni Merlino, and Antonio Puliafito. "Device-centric sensing: an alternative to data-centric approaches". In: *IEEE Systems Journal* 11.1 (2015), pp. 231–241.

[27] Ian Fette and Alexey Melnikov. *Rfc 6455: The websocket protocol*. 2011.

[28] Raspberry Pi Foundation. *Raspberry Pi 4 Product Brief*. https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf. Accessed: 2023-09-13. 2021.

[29] Johann Hauswald et al. "Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers". In: Istanbul, Turkey: Association for Computing Machinery, 2015, pp. 223–238. ISBN: 9781450328357. DOI: 10.1145/2694344.2694347.

[30] Shadi Ibrahim, Bingsheng He, and Hai Jin. "Towards pay-as-you-consume cloud computing". In: *2011 IEEE International Conference on Services Computing*. IEEE. 2011, pp. 370–377.

[31] IETF. *RFC 3746 - forwarding and Control Element Separation (forces) framework*. URL: https://datatracker.ietf.org/doc/html/rfc3746.

[32] Muhammad Usman Ilyas, Muneeb Ahmad, and Sajid Saleem. "Internet-of-Things-Infrastructure-as-a-Service: The democratization of access to public Internet-of-Things Infrastructure". In: *International Journal of Communication Systems* 33.16 (2020), e4562.

[33] Eldar Insafutdinov et al. "DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model". In: (). URL: http://arxiv.org/abs/1605.03170.

[34] Midori Kitagawa and Brian Windsor. *MoCap for artists: workflow and techniques for motion capture*. Routledge, 2020.

[35] Parmod Kumar. "Impact of Cloud Computing on Environmental Sustainability". In: *International Journal of Science, Engineering and Computer Technology* 7.3/4 (2017), pp. 213–214.

[36] Jessy Lauer et al. "Multi-animal pose estimation, identification and tracking with DeepLabCut". In: *Nature Methods* 19 (2022), pp. 496–504.

[37] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 740–755.

[38] Francesco Longo et al. "Stack4Things: a sensing-and-actuation-as-a-service framework for IoT and cloud integration". In: *Annals of Telecommunications* 72 (2017), pp. 53–70.

[39] Alexander Mathis et al. "DeepLabCut: markerless pose estimation of user-defined body parts with deep learning". In: *Nature Neuroscience* (2018). URL: https://www.nature.com/articles/s41593-018-0209-y.

[40] Alexander Mathis et al. "Pretraining Boosts Out-of-Domain Robustness for Pose Estimation". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Jan. 2021, pp. 1859–1868.

[41] Giovanni Merlino et al. "FaaS for IoT: Evolving Serverless towards Deviceless in I/Oclouds". In: *Future Generation Computer Systems* 154 (2024), pp. 189–205.

[42] Giovanni Merlino et al. "Infrastructure-centric, NetworkServer-agnostic LoRaWAN Roaming". In: *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)*. Vol. 21. IEEE. 2022, pp. 149–156.

[43] Giovanni Merlino et al. "Software defined cities: A novel paradigm for smart cities through iot clouds". In: *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*. IEEE. 2015, pp. 909–916.

[44] Dario Milone et al. "MocapMe: DeepLabCut-Enhanced Neural Network for Enhanced Markerless Stability in Sit-to-Stand Motion Capture". In: *Sensors* 24.10 (2024), p. 3022.

[45] Dario Milone et al. "Quantifying Clinician-Controlled Preload in Dental Implants: Analysis of Manual Tightening Torque and Complication Rates". In: *Engineering Proceedings* 56.1 (2023), p. 252.

[46] Dario Milone et al. "Smart design of customized hip prostheses in additive manufacturing by combining numerical and experimental methodologies". In: *IOP Conference Series: Materials Science and Engineering* 1275 (Feb. 2023), p. 012004. DOI: 10.1088/1757-899X/1275/1/012004.

[47] Rabeb Mizouni and May El Barachi. "Mobile phone sensing as a service: Business model and use cases". In: *2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies*. IEEE. 2013, pp. 116–121.

[48] Christine Murad et al. "Revolution or Evolution? Speech Interaction and HCI Design Guidelines". In: *IEEE Pervasive Computing* 18.2 (2019), pp. 33–45. DOI: 10.1109/MPRV.2019.2906991.

[49] Tanmay Nath* et al. "Using DeepLabCut for 3D markerless pose estimation across species and behaviors". In: *Nature Protocols* (2019). URL: https://doi.org/10.1038/s41596-019-0176-0.

[50] Rick Parent. *Computer animation: algorithms and techniques*. Newnes, 2012.

[51] Charith Perera et al. "Sensing as a service model for smart cities supported by internet of things". In: *Transactions on emerging telecommunications technologies* 25.1 (2014), pp. 81–93.

[52] Yongrui Qin et al. "When things matter: A survey on data-centric internet of things". In: *Journal of Network and Computer Applications* 64 (2016), pp. 137–153.

[53] Partha Pratim Ray. "A survey of IoT cloud platforms". In: *Future Computing and Informatics Journal* 1.1-2 (2016), pp. 35–46.

[54] D Gordon E Robertson et al. *Research methods in biomechanics*. Human kinetics, 2013.

[55] Corrado Santoro and Federico Fausto Santoro. "LWN Simulator - A LoRaWAN Network Simulator". In: *2023 IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. 2023, pp. 1–6. DOI: 10.1109/WETICE57085.2023.10477816.

[56] Xiang Sheng et al. "Sensing as a service: Challenges, solutions and future directions". In: *IEEE Sensors journal* 13.10 (2013), pp. 3733–3741.

[57] Tomas Simon et al. "Hand Keypoint Detection in Single Images using Multiview Bootstrapping". In: *CVPR*. 2017.

[58] Sukhpal Singh and Inderveer Chana. "Cloud resource provisioning: survey, status and future research directions". In: *Knowledge and Information Systems* 49 (2016), pp. 1005–1069.

[59] Kehua Su, Jie Li, and Hongbo Fu. "Smart city and the applications". In: *2011 international conference on electronics, communications and control (ICECC)*. IEEE. 2011, pp. 1028–1031.

[60] Giuseppe Tricomi et al. “Interfacing Intelligent Personal Assistant to SDI/O with one click”. In: *2022 International Conference on Computer Communications and Networks (ICCCN)*. IEEE. 2022, pp. 1–8.

[61] Giuseppe Tricomi et al. “Paving the way for an Urban Intelligence OpenStack-based Architecture”. In: *2024 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE. 2024, pp. 284–289.

[62] Luis M Vaquero et al. *A break in the clouds: towards a cloud definition*. 2008.

[63] Shih-En Wei et al. “Convolutional pose machines”. In: *CVPR*. 2016.

[64] Rao Yingdong. “Research on Different Convolutional Neural Networks in the Classification Scenes of Yoga Poses based on OpenPose Extraction”. In: *2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*. IEEE. 2022, pp. 1532–1535.

[65] Ashkan Yousefpour et al. “All one needs to know about fog computing and related edge computing paradigms: A complete survey”. In: *Journal of Systems Architecture* 98 (2019), pp. 289–330.