Sapienza
Università di Roma

Designing and Implementing a Distributed Earthquake Early Warning System for Resilient Communities: A PhD Thesis

Computer Science department
Ph.D. in Computer Science (XXXV cycle)

Enrico Bassetti
ID number 1401568

Advisor
Prof. Emanuele Panizzi

Academic Year 2021/2022

Designing and Implementing a Distributed Earthquake Early Warning System for Resilient Communities: A PhD Thesis
PhD. Sapienza University of Rome

© 2023 Enrico Bassetti. All rights reserved

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: bassetti@di.uniroma1.it

Abstract

The present work aims to comprehensively contribute to the process, design, and technologies of Earthquake Early Warning (EEW). EEW systems aim to detect the earthquake immediately at the epicenter and relay the information in real-time to nearby areas, anticipating the arrival of the shake. These systems exploit the difference between the earthquake wave speed and the time needed to detect and send alerts.

This Ph.D. thesis aims to improve the adoption, robustness, security, and scalability of Earthquake Early Warning systems using a decentralized approach to data processing and information exchange. The proposed architecture aims to have a more resilient detection, remove Single point of failure, higher efficiency, mitigate security vulnerabilities, and improve privacy regarding centralized EEW architectures.

A prototype of the proposed architecture has been implemented using low-cost sensors and processing devices to quickly assess the ability to provide the expected information and guarantees.

The capabilities of the proposed architecture are evaluated not only on the main EEW problem but also on the quick estimation of the epicentral area of an earthquake, and the results demonstrated that our proposal is capable of matching the performance of current centralized counterparts.

Contents

1 Introduction **1**

2 Introduction to SeismoCloud **5**

3 Simplify Node-RED For End User Development in SeismoCloud **7**

3.1 Other widespread EUD interfaces 8
3.2 Node-RED 8
3.3 SeismoCloud Earthquake Early Warning system 9
3.4 Adding abstractions in Node-RED 9
3.5 User testing and results 11
3.6 Discussion 11

4 Security assessment of common open source MQTT brokers and clients **13**

4.1 MQ Telemetry Transport (MQTT) 13
4.2 Exploring security issues in MQTT protocol implementations 14
4.3 Related works on MQTT security analysis 16
4.4 Methodology of the proposed analysis 17
4.5 Experimental results 18
4.5.1 Brokers 18
4.5.2 Clients 22
4.5.3 Physical Internet-of-Things device 23
4.5.4 Results 24
4.6 Discussion 25

5 Earthquake Detection at the Edge: IoT Crowdsensing Network **27**

5.1 Creating a Peer-to-Peer distributed EEW architecture 27
5.1.1 Motivation 28
5.2 A comprehensive overview of decentralized EEW systems 29
5.2.1 Quake-Catcher Network 30
5.2.2 MyShake 30
5.2.3 CrowdQuake 30
5.2.4 SOSEWIN 30
5.2.5 SeismoCloud 31
5.3 Proposed Architecture 31
5.3.1 Probes, Detectors, and Local Authorities 31
5.3.2 Network Architecture 32

5.3.3 Bootstrap Sequence 33
5.3.4 Detection Pipeline 33
5.3.5 Scalability 34
5.3.6 Fault Tolerance 34
5.3.7 Privacy 35
5.3.8 Practical Implementation Aspects 36
5.4 Prototype Implementation 36
5.4.1 Sensors Hardware 36
5.4.2 Software 37
5.4.3 Detection Pipeline 38
5.5 Results' evaluation 38
5.5.1 Limitations 40
5.6 Discussion 41

6 Evaluation of SeismoCloud 2.0 architecture via earthquake epicenter estimation 45
6.1 Estimating the epicenter in EEW networks 45
6.2 Existing approaches and their limits 45
6.3 Proposed process 46
6.3.1 Base architecture 46
6.3.2 Process principles 47
6.3.3 Epicenter estimation 49
6.3.4 Experiments 57
6.4 Discussion 64

7 Conclusions 67
7.1 Recommendations for Future Research 68

List of publications 83

Chapter 1

Introduction

Earthquakes are not currently predictable with a high level of accuracy. Although the research made significant progress in understanding the causes of earthquakes and the types of geological features that may be more likely to experience seismic activity, the precise timing, location, and magnitude of earthquakes are still unpredictable.

Despite this, there are ongoing efforts to develop Earthquake Early Warning (EEW) systems that can alert people of potential earthquakes in advance, giving them time to take protective measures. These systems rely on detecting initial waves of an earthquake before the more damaging waves arrive and can provide a few seconds to a minute of warning time. Additionally, these systems can exploit the difference between the earthquake wave speed and the time needed to detect the quake and reach the population in the area of impact (usually via electronic means) to provide an alert even before the first wave arrives in that area. While these systems are not yet perfect, they have the potential to save lives and reduce damage in the event of an earthquake.

The primary barrier to EEW deployment is the cost of building and maintaining the sensor network. While existing infrastructures can be retrofitted to add this task to the standard detection, the coverage is far from good due to the cost of high-quality sensors. Earthquake analysis done by currently deployed seismic stations aims to provide high-precision data for post-mortem studies: tens or hundreds of sensors are enough. A high density of sensors in a given area adds no value to that task. In contrast, EEW needs many stations to increase the detection speed, while the detection accuracy can be low. In recent years, Earthquake Early Warning research has looked at crowdsensing to work around this problem.

Crowdsensing is an increasingly popular method of collecting data and building real-world applications. It relies on the voluntary participation of individuals to collect and share data. Crowdsensing EEW unlocks the possibility of high-density coverage due to the low cost of sensors and the large number of devices that embeds sensors, like smartphones or wearable "smart" gadgets. While these devices use sensors that have lower accuracy when compared to the ones used in seismic networks, it has been demonstrated that the quality is sufficient for detecting quake waves [19] [42].

Current designs for crowdsensing EEW require a centralized architecture. This

approach relies on one or more fusion centers that gather data from sensors across a vast geographic area, potentially spanning the globe. Once the fusion center receives the data, it uses complex calculations to detect an earthquake wave and determine some basic parameters, such as the epicenter and the impact area. After these parameters are extracted, the fusion center sends EEW notifications to smartphones and other intelligent devices registered in the system.

Despite the benefits of this approach, such as quick and accurate detection of earthquakes, it has a fundamental weakness: its centralized structure. The dependence on a centralized architecture means that any interruption in communication with the fusion center, or even a fault in the fusion center itself, can hinder the detection of an earthquake or delay notifications.

The present work aims to contribute to the process, design, and technologies of Earthquake Early Warning (EEW). We investigate how to enhance the adoption, security, robustness, and scalability of crowdsensing-based EEW systems.

We started by studying the interaction between the EEW system SeismoCloud [53] and the final users. The goal was to improve the accessibility of data and events via End User Development (EUD), especially for non-computer literate users, and understand which kind of data these systems need to work, hoping that resulting improvements can help boost the adoption of these networks by users. This study also served as a basis for us to understand the logic behind crowdsourcing EEW.

Next, the focus shifted toward the MQ Telemetry Transport (MQTT) protocol [7]. It is often used for Internet-of-Things applications, like SeismoCloud, as a lightweight data and control bus between sensors and central systems. This fact led us to analyze the security and integrity aspects of the protocol due to its critical role played in EEW systems.

To address the main issue of avoiding Single point of failure (SPoF) in EEW architectures, we explored a novel approach that relies on a distributed architecture. Distributed systems spread the data processing and decision-making across a network of nodes, reducing the risk of a SPoF. By leveraging the collective power of numerous nodes, the distributed architecture can improve the resiliency and reliability of the network to faults. Distributed systems also provide other beneficial properties, like data locality, which improves the sensor's privacy. The proposed design comprises no fusion centers, as the detection is made on the edge (on sensors). A prototype of this architecture was also implemented using low-cost sensors in a test environment named "SeismoCloud 2.0". Simulations have shown that the performance of this architecture, in terms of time needed to alert the population in the area of impact, is comparable to centralized ones.

Finally, we designed a process for epicenter estimation on the proposed architecture, showing that these networks can provide a good estimation without central processing, and confirming the validity of the proposed approach on a secondary, yet crucial, problem.

These results demonstrate that the proposed architecture provides the same information on earthquakes of centralized architectures while avoiding some of their pitfalls or excess information while still using low-cost sensors.

The thesis is organized as follows: first, in Chapter 2, a general introduction to the current SeismoCloud network (as an example of a standard crowdsourced EEW system) is offered. Then, the work on End User Development is presented in

Chapter 3 and on MQ Telemetry Transport is presented in Chapter 4. After that, the proposed architecture and the comparison results with centralized approaches are presented in Chapter 5. A distributed algorithm for epicenter estimation is discussed in Chapter 6. Finally, conclusions and future research directions are reported in Chapter 7.

Chapter 2

Introduction to SeismoCloud

This section introduces SeismoCloud as a crowdsourcing Earthquake Early Warning system. SeismoCloud is cited as an example multiple times in this thesis. This part provides the necessary background information for readers new to SeismoCloud or crowdsourcing Earthquake Early Warning in general.

SeismoCloud is a community network designed to track earthquakes and provide Earthquake Early Warning to citizens. We designed SeismoCloud in 2013 in our research laboratory and the Italian National Institute of Geophysics and Volcanology (INGV). The network operates using low-cost seismometers, such as IoT devices or smartphone sensors. These devices are connected to a cloud-based system ("fusion center") via MQTT for both telemetry data and control plane (an introduction to MQTT is in Section 4.1). The "fusion center" receives and analyzes seismometer data to detect earthquakes and generate EEW for potentially affected areas.

Internet-of-Things sensors are constantly online, analyzing their accelerometer feed, while smartphones are active only when placed horizontally over a table, and no traveling is detected. Both sensors analyze the accelerometer feed; they report significant vibrations to the fusion center for both shake detection, and long-term archival in a Time-Series Database (TSDB). The dataset stored in the TSDB is available to users via APIs.

SeismoCloud generates events from different sources, such as official earthquake feeds, vibrations detected by sensors deployed by users in a crowd-sensing fashion, and metadata (sensor temperature, status, etc). It uses these sources internally to store and render data, and to generate Earthquake Early Warnings. These data are long-term archived, and they are available via End User Development (EUD) and API interfaces to all users.

The precision of the detection depends on the number of seismometers connected to the network. The more seismometers are in a particular area, the higher the detection's precision and speed. While it is impossible to provide exact information about an earthquake using those sensors, the system can generate reasonably precise warnings when many devices are in the epicentral area.

Earthquake Early Warnings are delivered via a smartphone application, which provides a customizable sound and vibration notification. The application is designed to bypass sound settings, ensuring the notification is audible even when the smartphone is set to "no sound". Early warnings can be received from neighboring

areas up to 20 seconds before detecting the earthquake at the epicenter.

Internet-of-Things sensor code is open source, allowing power users to port the algorithm to new platforms. Such experiments include integrating the sensor into a home automation system named “Home Assistant”. Amateur radio operators in Italy integrated SeismoCloud in weather stations and remote unmanned radio stations (such as radio repeaters).

As part of our efforts to raise awareness about earthquakes and disaster prevention, in past years we conducted seminars in high schools, engaging with students and providing them with information on earthquake science and safety measures. To make these seminars more interactive and engaging, we have used the SeismoCloud mobile app as a vector to capture students’ attention.

Figure 2.1. SeismoCloud network in Italy (Feb 26th, 2023). Green marks are either Internet-of-Things sensors or smartphone participating in the system. Background tiles from Google Maps - ľ Google.

Chapter 3

Simplify Node-RED For End User Development in SeismoCloud

Smart objects and Internet-of-Things (IoT) devices are becoming widespread; thus, they increasingly require that end users, who usually have limited computer science and engineering knowledge, define their behavior and configure them to connect with other objects as well as with online services.

Some programming tools allow wiring together hardware devices, API, and online services, offering a powerful mechanism that provides complete control to the end user. However, that often requires users to be knowledgeable about technical issues. To overcome this limitation, different solutions have been proposed recently. One of these solutions is named *End User Development* [29].

End User Development (EUD) is the technique of creating and modifying software and processes by non-professional users. This is in contrast to traditional software development, which is typically done by professional programmers or development teams. EUD allows non-experts to customize digital systems to meet their specific needs and requirements.

Node-RED [63] is a flow-based programming [48] platform commonly used for End User Development. It allows users to create and configure networks of Internet-of-Things devices and online services, without needing to write code. However, even if Node-RED uses EUD for abstracting concepts, it still uses low-level terms and components, like MQTT broker, WebSockets, HTTP requests. By doing so, it can be difficult for different-skilled users (i.e., users with no skills in computer sciences) to use it effectively, as it requires knowledge of networking and protocols in order to configure and control the devices. In the application taken into consideration, which is SeismoCloud, users struggled to understand the role of such low-level components. Works in the literature that uses Node-RED require some prior knowledge of underlying protocols (for example, in Tabaa et al. [61], skills on Modbus standards and protocols are required).

To address this issue, we propose adding a level of abstraction to Node-RED nodes, creating a user-friendly interface that allows so-called different-skilled users to configure and easily control IoT devices and online services networks. These

abstractions hide the underlying technical details and allow users to control the devices and services without knowing how they work, which is viable using abstraction with domain-specific items [32].

The abstractions have been added to the SeismoCloud application for earthquake monitoring, which uses IoT sensors to detect and report earthquakes. With these abstractions in place, we expect that different-skilled users would be able to easily configure and control the sensors and integrate them with other online services and applications. These results will demonstrate that EUD should be raised above the components level to allow users to interact and change the system's parameter (in this case, an Earthquake Early Warning system), even if they do not have any technical knowledge or expertise.

3.1 Other widespread EUD interfaces

Node-RED is widely used as EUD interface for different projects, ranging from simple IoT sensors integration (like temperature sensors [43] or air quality monitors [14]) to home automation [58], to complex industrial automation [61]. It is also used to teach in Data Engineering courses [13] due to its simplicity.

A similar product is *IFTTT* "*if this then that*" [52], which is a cloud-based *trigger-action* End User Development (EUD) platform. However, *IFTTT* supports only simple flows [65] (from triggers to action) and not complex flows as Node-RED does. For example, it is impossible to synthesize in IFTTT a complex industrial flow, while Node-RED is capable of representing it, as demonstrated by Tabaa et al. [61].

Ghiani et al. [32] provided an extensive and detailed analysis on the mechanisms behind EUD when targeting an end-user. Node-RED itself has already implemented some of these techniques. However, while *Ghiani et al.* "TARE" system uses a composition technique with buttons to build rules, our proposal uses a visual approach based on the idea of "information flow".

3.2 Node-RED

Node-RED is built around the concept of *events* and *messages*. It allows users to create networks of nodes, where each node represents a specific function or capability. When an event occurs, such as a sensor reading or a user action, Node-RED routes a message to the appropriate nodes, which then execute one or more actions based on the message.

A group of nodes that are connected together and that execute one or more actions in response to an event is called an *action flow*. These action flows can be created by connecting the nodes in the visual editor.

Node-RED comes with a set of built-in nodes that provide a wide range of capabilities, such as reading from sensors, sending messages over the internet, and storing data in databases. Additionally, users can add third-party nodes using plugins, which can provide even more functionality and extend the platform's capabilities. This makes Node-RED highly extendable while still being very simple and user-friendly.

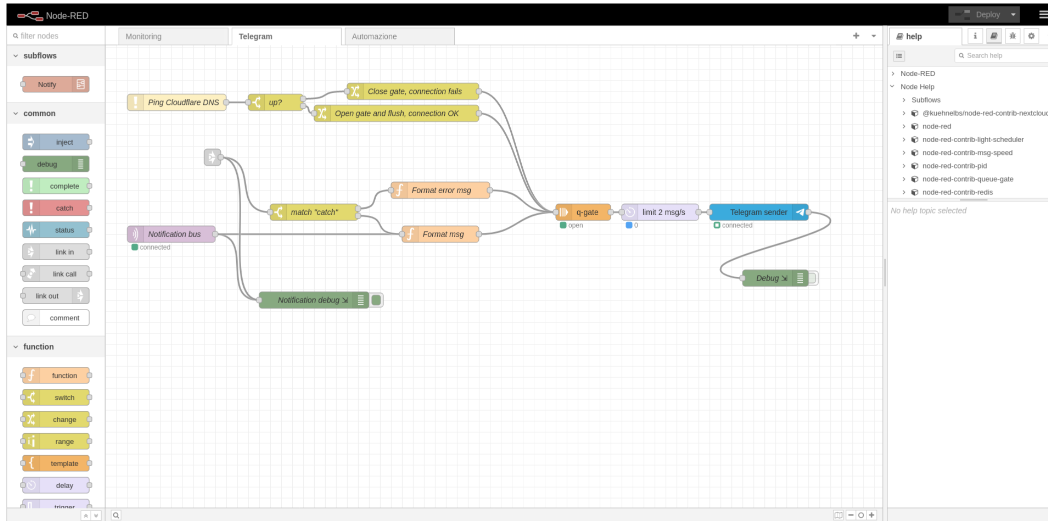


Figure 3.1. Node-RED visual editor with an action flow running.

3.3 SeismoCloud Earthquake Early Warning system

The integration of Node-RED into SeismoCloud allowed us to create a more user-friendly End User Development platform. By enabling users to create custom flows, Node-RED allowed for the development of personal online services such as alarms and statistics using the data provided by SeismoCloud. The flexibility of Node-RED's drag-and-drop interface made it accessible to users without technical skills, and our modifications to its tool set further simplified the platform by removing the need for understanding of IoT and network-related concepts.

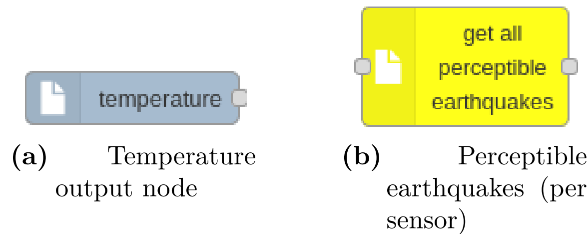
The user-friendliness of the SeismoCloud platform is crucial in ensuring that citizens can access and use earthquake data effectively. With SeismoCloud, users can easily create custom alerts and notifications based on their personal preferences and the earthquake data available. Additionally, the platform allows users to access real-time and historical seismic data.

3.4 Adding abstractions in Node-RED

SeismoCloud sensors use MQTT for signaling and data stream. While MQTT is a relatively simple protocol, it still requires some knowledge of networking and protocols to be used effectively. This can be challenging for non-technical users, who may not be familiar with concepts like broker endpoints, topic subscriptions, and message publishing. Additionally, some information about earthquakes is only available via a REST API, requiring even more technical knowledge and programming skills.

To tackle this problem, we designed and implemented domain-specific *nodes* that users can drag and drop and configure specifying domain-specific information in human-readable form. These nodes are abstractions for the Node-RED supported technologies, like MQTT or REST, built over the SeismoCloud platform. For example, the "temperature" node (fig. 3.2a) can be easily configured just by specifying the IoT sensor name (fig. 3.3), without any knowledge of the MQTT configuration

which is automatically performed.



(a) Temperature output node

(b) Perceptible earthquakes (per sensor)

Figure 3.2. Example of two new nodes created for this experiment

In fact, when the sensor sends its temperature using MQTT to all its subscribers, this value is received by all the "temperature" *nodes* (fig. 3.2a) configured using its name. All details such as *MQTT broker*, credentials, topic name, Quality of Service, Transport Layer Security configuration, are hidden from the point of view of the user.

By mixing Node-RED standard nodes and our high-level abstractions, different-skilled people can build and understand Node-RED flows, such as "when two devices emit a vibration message, send an SMS" (see figure 3.4).

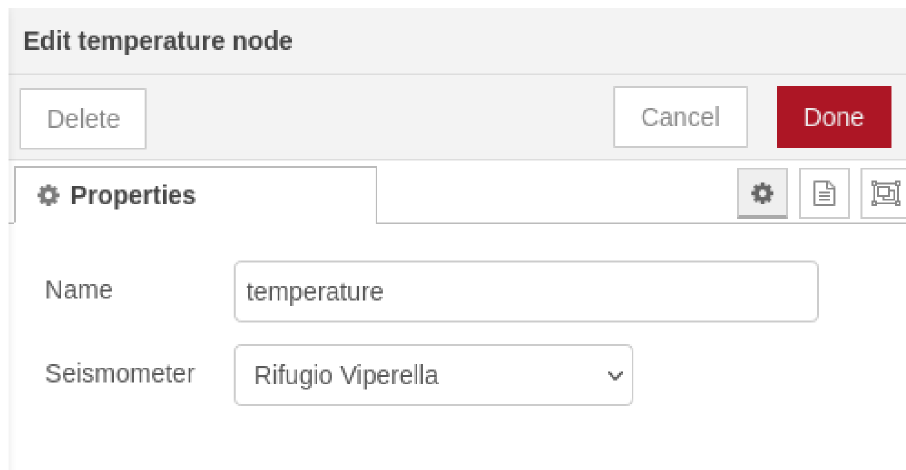


Figure 3.3. Configuration panel for the newly designed "Temperature" node.

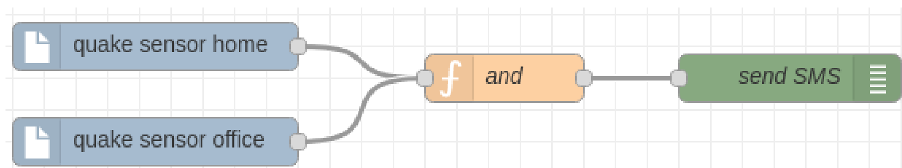


Figure 3.4. A very simple flow in Node-RED that sends a SMS whether two sensors ("home" and "office") detect a quake. Messages flow from left to right.

3.5 User testing and results

To evaluate whether our approach represents a good simplification for ordinary users and an enabler for the EUD in SeismoCloud (while maintaining enough expressive power to create actions as desired by users), we planned and executed a test with seven different-skilled users plus one expert user using different techniques:

- think-aloud, which involves asking users to verbalize their thoughts and actions as they use the interface;
- cooperative evaluation, which involves working with users to identify and evaluate the strengths and weaknesses of the interface;
- post-task walkthroughs, which require asking users to describe their experience using the interface after completing a specific task;
- expert-based testing, in which individuals with expertise in a specific area (such as usability or user experience design) are involved in evaluating the interface.

Users were asked to execute four tasks with different difficulty levels, which required getting a data source event (e.g., a seismometer vibration detection or a temperature sensor above threshold) and triggering a message using a simple label-based template. They were expected to drag and drop the proper nodes from the palette to the workspace and create a Node-RED flow. For each task, we recorded the duration, success or failure in achieving task goals, and whether the user understood newly introduced nodes.

After performing tests with the first four users, we made a second version of the nodes, simplifying some labels that describe the information required for their configuration. As an example, while asking the users if they understood what they were doing, we discovered that the *SSID*[1] node was not well understood (all the different-skilled users replied they understood that node returned a value, but the meaning was not clear). So, we changed the node label to *Wi-Fi network name* to better reflect the meaning of the sensor.

In subsequent tests, all users could complete every task without significant problems, although sometimes it took more time than we expected.

3.6 Discussion

The result of our tests are encouraging. We have found that EUD can be effectively applied to earthquake early warning applications (such as SeismoCloud) that use Node-RED. We are confident that this process simplifies the end-user development process by adding more context to the Node-RED nodes and creating domain-specific nodes. These modifications made it easier for users with limited programming experience to build and modify the SeismoCloud application, and seem to improve the overall usability and accessibility of the platform.

[1] Service Set Identifier: in Wi-Fi standard, this ID groups together multiple APs providing the same network. SSID is commonly referred as the Wi-Fi network name.

Overall, our findings suggest that end-user development is a valuable approach for enhancing the usability and flexibility of Node-RED in the context of SeismoCloud and other earthquake early warning applications. We believe these results have important implications for designing and developing other end-user development platforms.

This research has been presented at *1st International Workshop on Empowering People in Dealing with Internet of Things Ecosystems - EMPATHY 2020* [9].

Chapter 4

Security assessment of common open source MQTT brokers and clients

Security and dependability are crucial for Internet-of-Things systems, especially in sensitive applications like Earthquake Early Warning in crowdsensing. So, after working on the EUD project, my focus shifted towards the core component in the SeismoCloud network: MQ Telemetry Transport (MQTT) [7]. MQTT is the standard communication protocol for resource-constrained IoT devices, but it was not designed with security in mind. As MQTT is widely used in real-world applications, it has come under scrutiny from the security community due to the prevalence of attacks targeting IoT devices.

We conducted an empirical security evaluation of several widespread MQTT system components, including five broker libraries and three client libraries. While our research did not uncover any significant flaws, there were scenarios where some libraries did not fully adhere to the standard, leaving room for exploitation and potential system inconsistencies or denial-of-service, which might become an essential issue in safety-critical systems like SeismoCloud EEW.

4.1 MQ Telemetry Transport (MQTT)

MQ Telemetry Transport (MQTT) is a lightweight and flexible messaging protocol widely used in the Internet-of-Things and other distributed systems to exchange information between devices. It was developed in 1999 by IBM and later became an open-source protocol, and it was transferred to the Eclipse Foundation.

MQTT protocol uses a client-server model, where the clients can be either publishers or subscribers, and the server is a broker. The broker acts as an intermediary, receiving messages published by clients and delivering them to interested subscribers. Multiple servers can form a "broker network" to absorb high loads, acting as one broker. Subscribers can subscribe to specific topics they are interested in, and the broker routes the messages to the correct subscribers based on their subscriptions.

The publisher role refers to the client that generates and sends messages to the

broker for distribution. The subscriber role refers to the client that receives messages from the broker. Both publishers and subscribers can connect to the broker using a network connection and send and receive messages in real time.

Topics in MQTT are used to classify and organize messages and define the hierarchy of the topics. A *topic* is a string, and they comprise one or more levels, separated by forward slashes ("/"). MQTT subscribers can use wildcards to subscribe to multiple topics that match a specific pattern. The two types of wildcards supported in MQTT are the "+" symbol, used to match any value in a single level, and the "#" symbol, used to match multiple levels.

MQTT supports three Quality of Service levels, which define the level of guarantee for message delivery. QoS 0 provides "at most once" delivery, where a message is delivered once, but not guaranteed to be received. QoS 1 provides "at least once" delivery, where a message is guaranteed to be received at least once but may be received multiple times. QoS 2 provides "exactly once" delivery, where a message is guaranteed to be received exactly once, but at the cost of increased network overhead and the complexity of implementations.

Not all MQTT clients and libraries support all QoS levels. Due to its complexity, QoS 2 is often disregarded. Also, MQTT subscribers can subscribe using a lower or higher QoS level than the published message. MQTT brokers are expected to overcome these limitations, for example, by downgrading the QoS for the message or the subscription when deliverying a message that doesn't match on the QoS value.

Overall, MQTT's lightweight protocol specifications, flexible topic design, and support for different QoS levels make it a popular choice for many IoT and distributed systems applications. The protocol has been standardized under the ISO umbrella as ISO/IEC 20922:2016 [38].

4.2 Exploring security issues in MQTT protocol implementations

Recently, there has been a rapid surge in the number of internet-connected devices, facilitating the emergence of innovative technologies and applications across different fields. Among these trends is the Internet-of-Things (IoT), characterized by the proliferation of low-cost, small devices with a limited range of functions, featuring an Internet protocol (IP) stack, an Ethernet/Wi-Fi connection, and the capability to be re-programmed with commodity hardware, like USB ports. This development has paved the way for exploring a broad spectrum of novel applications, including nonprofessionals and crowdsourcing applications.

Internet-of-Things devices have a tiny amount of resources as their main goal is to be as small as possible and low-cost. Sometimes, widely used protocols like HTTP cannot be efficiently implemented without sacrificing critical features of the protocol itself (leading to non-standard implementation) or essential parts of the "business logic" (i.e., the primary purpose of the device). To overcome this limitation, several lightweight protocols were invented, like MQ Telemetry Transport (MQTT) protocol or Advanced Message Queuing Protocol (AMQP) [50]. When resources like memory and processing power are severely limited (as in simple sensors/actuators), and the system is in under-constrained environments (low-speed wireless access), the former

is the preferred choice [46].

MQTT is a publish-subscribe protocol based on a simple message structure, essential features, and a minimal packet size (considering the message headers). An introduction to MQTT is present in Section 4.1. Thanks to its design, a large number of IoT devices use MQTT or similar lightweight protocols to talk to each other and communicate with the rest of the world. Also, MQTT has undergone several standard processes, and MQTT v. 3.1.1 and 5.0 are both ISO standards [38].

The protocol was conceived with no security concern since initially designed for private networks of the oil and gas industry [49]. The protocol's adoption has ramped up, and several statistics show that many devices use it without any protection [39]. In the MQTT v. 3.1.1 and 5.0 specifications, Transport Layer Security is cited as supported but not required to be implemented by brokers or clients. Some implementations offer extension plugins to improve security[1] (e.g., role-based authentication, Access Control Lists), but the standard does not support any of these features, limiting the compatibility of these extensions.

Considering the privacy aspects, given its quite limited features, the MQTT protocol has no built-in encryption features; also, the use of TLS to provide a secure communication channel is very limited: at the time of writing, comparing with the Shodan search engine the prevalence of the exposed IoT and Industrial Internet-of-Things (IIoT) devices using MQTT, brokers exposing port 8883 (MQTT over TLS) are 42, while those exposing port 1883 (unencrypted MQTT) are 154632 [39].

Since MQTT is widely used for real applications, it is under the lens of the security community, also considering the widespread attacks targeting IoT devices. The research is focusing on shifting towards ensuring secure IoT systems, for example, implementing access control mechanisms [17], lightweight cryptography [26] or remote attestation of devices [31]. An essential aspect of this context is discovering unforeseen security risks resulting from the necessary interoperability with different implementations of MQTT libraries.

Following this research direction, in this chapter we present an empirical security evaluation of several widespread implementations of MQTT system components, namely five broker libraries and three client libraries. Moreover, we also applied our security analysis to an MQTT client embedded in a physical IoT device, namely a Shelly DUO Bulb. This IoT device is a remote-controlled LED light bulb. It supports Wi-Fi connectivity and acts as an MQTT subscriber to receive commands, like powering on/off or light dimming.

Our evaluation aims to verify the responses of the components of the different libraries to different MQTT messages to see their behavior in situations where the standard does not indicate clearly how the message (or the connection itself) is supposed to be handled. This mishandling might create interoperability issues or even open doors to malicious attackers. While the results of our research do not capture critical flaws, there are several scenarios where some brokers and libraries do not fully adhere to the standard and leave some margins that could be maliciously exploited and potentially cause system inconsistencies or unavailability.

[1] `https://mosquitto.org/documentation/dynamic-security`

4.3 Related works on MQTT security analysis

As the abundance of surveys suggests [2, 44, 59], security and dependability of IoT devices is paramount for the whole ecosystem. In this context, the MQTT protocol plays a determinant role. In 1999 Andy Stanford-Clark (IBM) and Arlen Nipper (then working for Eurotech, Inc.) proposed the MQTT protocol [8] to monitor oil pipelines within the SCADA framework [22]. Since then, it has been revised into two main versions, namely 3.1.1 (last update December 2015) and 5 (last update March 2019). To date, the former is by far the most used in real applications, the latter being much newer and still not widely adopted [22].

Like all the network protocols becoming a standard, it has undergone many formal and empirical reviews. Several papers focus on MQTT formal modeling and performance analysis [16, 34, 36], others on its possible vulnerabilities, and many others on its security analysis. This research analyzes the security and compares several of widely used software libraries implementing the MQTT protocol. Instead of using static analysis of their code, as in [1], we perform a dynamic analysis using the *fuzzing* methodology. In [33], the authors proposed a template-based fuzzing framework and tested its effectiveness against two implementations of MQTT. Using this method, they found some security issues: Moquette and Mosquitto brokers were affected by a vulnerability that would have led to a Denial-of-Service (DoS) attack in specific settings if exploited. In our research, we focus on possible DoS attacks and the effects of standard violations by brokers and clients. Moreover, our analysis applies to five brokers, three clients, and a physical device.

In [66], the authors evaluated the robustness of several MQTT implementations against a subtle family of attacks known as low-rate denial of service. Similarly to this work, a real testbed was set up, and several experiments were performed, validating the open vulnerability of all the MQTT implementations.

In [5], authors described a new strategy to test MQTT through fuzzing and how much it is efficient against the protocol. However, they did not present any results about the application of their strategy. A similar approach is adopted in [12], where the authors propose to apply fuzzing techniques in a container-based environment (Docker). This would allow a large-scale test of the MQTT protocol. However, again, the authors did not compare different implementations (they only consider Mosquitto), nor describe the type of attacks they performed.

A different methodology based on *attack patterns* [60] was proposed by *Sochor et al.* and was used to spot hidden vulnerabilities in different broker implementations. They adopt a method to randomly generate test sequences (Randoop) to challenge the different brokers, and they were able to find several failures and unhandled exceptions. Our research adopted a different methodology, tested different broker MQTT implementations, and included clients.

Another methodology to perform a dynamic analysis is model-based testing, as proposed for MQTT applications in [62]. The methodology considers using a finite state machine that verifies the properties of the software and proposes extensions to model-based tools for MQTT applications.

Additionally, in [39] Kant et al. showed that many consumer-grade devices do not use a secure transport for MQTT (like TLS), further expanding the attack surface.

4.4 Methodology of the proposed analysis

Our research aims to compare the behavior of various implementations of the MQTT protocol. To do this, we identified and set up the most popular open-source MQTT broker and client libraries that are commonly used to manage devices or develop software solutions. We used the number of stars and forks on the corresponding GitHub repositories and the number of blog posts citing the brokers as indicators of their popularity. Once we had identified and set up the most popular libraries, we conducted our empirical study to compare their behavior and identify any deviations from the standard that could potentially lead to inconsistent or critical states in applications.

For this assessment, we considered open source libraries and brokers as they are widely used thanks to availability and licenses: *Mosquitto, EMQ X, HiveMQ, Moquette* and *Aedes* as brokers, *paho, mqttools, and mqtt.js* as client libraries. We discuss the brokers and the clients respectively in Section 4.5.1 and in Section 4.5.2. Some of these have thousands of instances running in "production" environments in typical consumer and business-to-business solutions. We also added a popular low-cost Internet-of-Things device to the comparison, namely the Shelly DUO Bulb (Section 4.5.3), to assess whether the complete stack (i.e., when both software and hardware are combined) exposes some unexpected issues.

We thoroughly reviewed the MQTT standard, version 3.1.1, to identify any undefined behaviors, unspecified states, or other missing information related to message handling to use them in our fuzzer. In addition, we examined the standard for any sections that might result in incorrect implementation due to unclear or missing specifications for expected actions by the broker or client. By focusing our testing on this restricted subset of cases, as outlined in Section 4.5, we were able to ensure that our testing was targeted and effective.

Different sets of experiments were created to find possible anomalies in MQTT implementations. A custom fuzzer, written in Python with the help of the `twisted` library, was developed to manage different streams and send custom packets. This allowed for the manipulation of every packet bit, allowing for testing the broker's behavior even in the presence of malformed packets. On the other hand, standard MQTT libraries implement state machines that are expected in certain parts of the protocol, such as QoS 2. These libraries do not allow for arbitrary changes in the flow of messages, like out-of-order messages. Each experiment was codified in a JSON file that specified the sequence of actions or packets that should be run on or against the software under test. The behavior of the parties involved was logged and analyzed.

In our work, the model of the attacker includes the capability to modify the MQTT packet flow, delay the transmission or make it out of order, or modify the MQTT packets payload, injecting invalid values. This capability can be exploited with limited access to the broker or intermediate network devices, or even remotely, by using other attacks like *Distributed Denial-of-Service* or *flooding* against a network device in the path of the packet flow (for delaying packets, for example). These vulnerabilities can be exploited in plain MQTT. Some of these can also be exploited

when MQTT is tunneled in an older version of TLS protocol itself[2]: for example, SSL used a vulnerable *Message Authentication Code* until TLS [25]; vulnerabilities in TLS HMAC implementations are still found years after the standard [47].

4.5 Experimental results

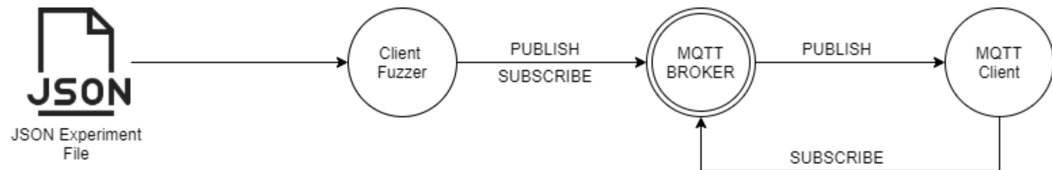


Figure 4.1. Schema of the testbed for the experiments: the fuzzer, which acts as a typical client, takes in input a "JSON experiment file" containing the client's packets to the MQTT broker. The fuzzer will also receive all the PUBLISH packets sent to the broker. The MQTT Client, instead, uses one of the libraries that are examined in the subsection 4.5.2.

4.5.1 Brokers

The task of a MQTT broker is to accept and forward messages from "publishers" (clients who send messages to a specific topic) to "subscribers" (clients who want to receive messages about specific topics) based on the message topic. This loosely coupled architecture allows clients to communicate without being aware of each other and without being designed for a direct (often specific) connection.

Modern brokers support many concurrent connections for publishers and subscribers, complex topic routing matching, and different retention mechanisms. They usually forward a large number of messages per second. A flaw in message state machines, packet parsing, or topic logic might expose a vulnerability that, if exploited by a malicious actor, might significantly impact some property of the MQTT system, like a Denial-of-Service (DoS).

The MQTT brokers analyzed are:

- **Mosquitto**[3]: it is one of the most used MQTT brokers in the world. It is a single-threaded, lightweight broker written in C. This broker has been widely used thanks to its flexibility;
- **EMQ X**[4]: it is written in Erlang and it claims to be so efficient to be "the Leader in Open Source MQTT Broker for IoT";
- **HiveMQ**[5]: a broker written in Java. It supports MQTT version 3.x and 5.0 and it is widely used in automation and industrial systems[6]. We tested the Community Edition;

[2]Note that low-cost IoT devices often implement old protocols, sometimes even partially
[3]https://mosquitto.org/
[4]https://www.emqx.io/
[5]https://www.hivemq.com/developers/community/
[6]https://www.hivemq.com/solutions/manufacturing/modernizing-the-manufacturing-industry/

- **Moquette**[7]: another Java-powered open-source broker. It is very lightweight but it is less-known and less-used, when compared to other brokers;
- **Aedes**[8]: a broker written in JavaScript/NodeJS. It is the successor of MoscaJS. It does not support version 5 of MQTT, but it is fully compatible with version 3.x and supports several extension libraries.

Each broker underwent the same set of tests. We performed more than 60 different experiments on a consumer-grade PC with local connections. A summary of the results is in Table 4.1. In Section 4.5.1 we describe the most relevant ones.

Table 4.1. Summary of test results for brokers. The tested versions were the latest stable available at the time of our experiments.

Broker	**Anomalies in**	**Security problems**	**Version**
Mosquitto	QoS	unexpected / undocumented behavior	1.16.12
EMQ X	QoS and *long topics*	unexpected / undocumented behavior	4.2.1
HiveMQ	QoS and *long topics*	unexpected / undocumented behavior	2020.5
Moquette	QoS and *long topics*	Denial-of-Service and unexpected / undocumented behavior	0.13
Aedes	QoS and *packet references*	Denial-of-Service	0.43.0

Experiments and results

All experiments were conducted multiple times in a controlled environment. Each test was performed for 1 minute continuously (where possible, or until the crash of the broker). No other load was present on both the server and the client machines.

Publish the same message using QoS 2 then 1. In MQTT, messages have an ID for Quality of Service transactions. As stated in Section 4.1, PUBREL packets should appear as a response to a QoS 2 PUBREC packet. In this experiment, we exploit the state machine confusion by sending a packet with the same ID twice with different QoS levels: first, we send a packet with QoS 2; then, we send another packet at level 1; and finally, we ask the server to continue with the transaction of QoS level 2.

An optimal response by the server to this client behavior is to reject the QoS 1 transaction when the second packet with the same ID is being published, or at least ignore the duplicated packet, as the standard explicitly says that the packet should be "currently unused" [8].

To test the server, our fuzzer performs the following steps:

[7] `https://github.com/moquette-io/moquette`
[8] `https://github.com/moscajs/aedes`

1. it sends a SUBSCRIBE packet with a specific topic;

2. it sends the first PUBLISH packet with a Quality of Service 2 and with id 1 over the topic specified in the subscription;

3. it sends the second PUBLISH packet with a Quality of Service 1, still with id 1 over the topic specified in the subscription;

4. it sends a PUBREL packet for the first packet sent.

We noticed different broker behaviors: Mosquitto publishes the first received packet with QoS 2, but then it loses the second packet that is not published to the clients due to the PUBCOMP packet that is not received, and so the packet id is not available for use. The EMQ X broker publishes both packets; it handles the flow for the first packet and then the flow for the second one. In HiveMQ and in Moquette, the client that sends packets receives the publication first and after the PUBCOMP concerning the first packet. Additionally, in HiveMQ, the client receives the PUBCOMP back first and then the PUBREC. Aedes publishes both packets, but the PUBCOMP arrives at the client after the two publications. This behavior repeated several times, also in the other experiments regarding the Quality of Service that are described below; it is a violation of the MQTT standards as it specifies that the PUBCOMP messages must arrive before the publication of the data to complete the message transmission for the packet with the QoS 2.

Publish QoS 2 and 0. This experiment is very similar to the one described above; however, in this case, the fuzzer is downgrading the QoS to zero.

The client performs the following steps:

1. it sends a SUBSCRIBE packet with a specific topic;

2. it sends the first PUBLISH packet with a Quality of Service 2 and with the id 1 over the topic specified in the subscription;

3. it sends the second PUBLISH packet with a Quality of Service 0 and with the id 1 over the topic specified in the subscription;

4. it sends a PUBREL packet for the first packet sent.

Mosquitto, in this case, publishes both packets but in reverse order: it handles the one with Quality of Service 0 first, and then it handles, correctly, all the flow regarding the first packet sent with Quality of Service 2. EMQ X and HiveMQ maintain the order of the packets published by the client; also, in the case of HiveMQ, the client received back the PUBCOMP first and then the PUBREC regarding the packet with Quality of Service 2. Moquette behaves similarly to EMQ X, but, in this case, the PUBCOMP arrives after the publication of the second packet. Aedes has the same behavior as Mosquitto, but the PUBCOMP arrives after the publication as in the previous experiment.

Double publish QoS 2. The ID in MQTT transactions should be unique during the transaction – meaning that peers should not use the same ID for different messages at the same time. In this experiment, we verify this requirement by sending two messages with the same ID and the same QoS.

To implement this test, our fuzzer performs the following steps:

1. it sends a SUBSCRIBE packet with a specific topic;
2. it sends the first PUBLISH packet with a Quality of Service 2 and with the id 1 over the topic specified in the subscription;
3. it sends the second PUBLISH packet with a Quality of Service 2 and with the id 1 over the topic specified in the subscription;
4. it sends a PUBREL packet for the first packet sent;
5. it sends a PUBREL packet for the second packet sent.

The standard does not specify what to do when encountering this situation. We discovered that brokers implement different logic: in Mosquitto and EMQ X, only one publication referred to the first packet sent, but the flow regarding the Quality of Service is properly handled. HiveMQ and Moquette both publish the two packets in the correct order (violating the standard regarding the reuse of the packet ID). In Aedes, there is a potential **dangerous behavior**: the broker publishes the first packet (sent by the client) twice instead of sending two different packets.

Long topic. In MQTT, messages have topics: a string representing the "subject". According to the MQTT standard, the maximum length of a topic is 65536 bytes. However, when analyzing the source code of EMQ X, we found that the constant that represents the maximum length for the topic handled in the program was set to 4096 bytes. This means that EMQ X has a lower limit for the maximum length of a topic compared to the requirements in the MQTT standard.

In this experiment, we tried subscribing to a topic with more than 4096 bytes (up to 65536) using different messaging brokers. We observed different behavior: EMQ X disconnects the client when the topic is too long according to their internal constant. However, in HiveMQ, **the topic was cut, and the client subscribed to a shortened version**. In Moquette, there was an *IOException*, and the client connection was severed. However, **the most dangerous behavior** is in Aedes: there was a crash of the broker, and an exception was thrown with an error message stating "too many words", creating a Denial-of-Service. Mosquitto is the only broker that handles the topic subscription correctly.

Other experiments. We conducted additional experiments after the ones previously described. They have been summarized because the brokers' behavior was mostly correct, and there was no issue to report.

- we experimented with the *client id* value in the packet. We tried to fill it with non-UTF-8 encoded characters: no anomalies. In detail, we have built a connection packet with the *client id* containing particular characters, and the experiment was handled correctly by all brokers;

- fuzzing the *Keep-alive* field in connection packet: in all brokers, there is the client disconnection due to a malformed packet;
- we tried subscriptions and publications with invalid *wildcards* in topics: all brokers correctly disconnect the client due to "invalid topic";
- encoding topics and *wildcard* non-UTF-8 encoded characters: the client is correctly disconnected;
- large number of levels (i.e., a large number of `/` in the topic): in *Mosquitto, EMQ X, Moquette* and *Aedes* the client is disconnected after reaching a certain sub-level; HiveMQ cuts/limits the topic to a certain level;
- flooding the broker with QoS 0 packets at maximum packets per second: all brokers handled the flood as required by the standard;
- sending invalid protocol name (or version) in the connection packet: in all cases, the client is disconnected;
- sending a PUBREL packet that references a publication packet that was never sent: all brokers, except for Aedes, sent back a PUBCOMP message. In Aedes, the client is disconnected.

4.5.2 Clients

After testing common brokers, our focus shifted toward MQTT client libraries. As we did with brokers, we considered metrics such as the number of stars and forks on the relevant GitHub repositories. We have chosen three widely used libraries: `paho-mqtt`[9] from the Eclipse Paho project, `mqttools`[10] and `mqtt.js`[11]. The first two libraries are in Python, while the third one is in JavaScript. Especially the MQTT client from Paho is widely used: more than 500 libraries depend directly on it (source: Libraries.io[12]), including `azure-iot-device` (the official Microsoft MQTT client for Azure IoT broker) and other tools like `ESPHome`.

Our experiments did not uncover any notable anomalies, only minor issues. It is worth noting that these tests were just as necessary as those for brokers, as client libraries might be manipulated to send erroneous packets in some situations (for example, when the input is coming from the user). We conducted these tests to ensure these client libraries' reliability and functionality.

A summary of the results is in Table 4.2. Here the tests we performed:

- invalid QoS level: all libraries report an error about the QoS, blocking the sending of the packet;
- invalid *wildcard* subscription: in this case *mqtt.js* generates an "Invalid topic" error, while the other two libraries timeout;

[9] `https://pypi.org/project/paho-mqtt/`
[10] `https://pypi.org/project/mqttools/`
[11] `https://github.com/mqttjs/mqtt.js`
[12] `https://libraries.io/pypi/paho-mqtt`

- *client id* not encoded in utf-8: in *mqttools* the client refuse to connect to the broker, in *paho-mqtt* there is a successful connection to the broker and *mqtt.js* generates an error with the consequent client disconnection;

- publication (or subscription) to a topic with a length more than 65536 characters: all libraries disconnect without sending the packet.

Table 4.2. Summary of test results for client libraries. The tested versions were the latest stable available at the time of our experiments.

Library	Anomalies in	Security problems	Version
paho-mqtt	handling subscription and publication to an *invalid wildcard.*	Denial-of-Service (the library *hangs*).	1.5.1
MQTT.js	handling an invalid Quality of Service.	Denial-of-Service (the library *crashes*) due to a *TypeError*.	4.2.1
mqttools	handling subscription and publication to an *invalid wildcard* and when the *client id* value contains non-UTF-8 chars.	Denial-of-Service (the library *hangs*), resource exhaustion and other issues due to an infinite connection loop.	0.47.0

4.5.3 Physical Internet-of-Things device

The MQTT protocol is commonly used in the home automation field, as most intelligent devices support it. Numerous software applications allow users to utilize the MQTT protocol to manage their smart devices, such as *Home Assistant*. In addition, Amazon's AWS IoT platform uses MQTT to connect users' devices to other devices and services.

In order to conduct further experiments in a realistic scenario, we chose to use a physical, commercially available Internet-of-Things device as a test subject: the Shelly DUO smart light bulb. This device can connect to Wi-Fi networks and can be remotely controlled using the MQTT protocol.

We discovered that this device does not have an "anti-flood" regarding the packets it receives; for example, it is possible to turn off and on the light bulb repeatedly and quickly by sending a PUBLISH packet on the specific topic with specific content. The software that runs in the light bulb is the same as other Shelly devices, so this problem also affects them. Therefore, it is possible to send many packets that overload the device's electronic components to damage or render them unusable (DoS attack).

These problems are exacerbated due to missing or incomplete TLS support in Internet-of-Things devices, which is common in low-cost IoT devices due to resource constraints [39]. Also, very often, these devices are missing QoS 2 (sometimes even

QoS 1). The missing protection from TLS and the partial support of QoS allows attackers to perform “replay attacks”[13].

We performed the same set of experiments on the Shelly DUO, and the previous results were confirmed, indicating that some of these issues are also present when Shelly devices are paired with vulnerable brokers and clients.

In addition to these experiments already performed for the various brokers, we have tried to generate some attacks like *buffer overflows* through the payload sent to the device. However, the light bulb passed all tests without errors; in particular, the device ignores any form of payload other than what it expects to receive.

4.5.4 Results

The experiments conducted on brokers, clients, and the physical device yielded exciting results. As described in Section 4.5.1, some of the unexpected behaviors observed in the brokers could be classified as vulnerabilities and could potentially lead to attacks under certain conditions. For example, we observed that in some tests, brokers published messages that violated the protocol state machine. One example of this is the out-of-order use of PUBCOMP by Aedes (and other brokers), which could potentially be used to trigger a replay attack if the PUBCOMP itself is delayed or dropped by a malicious actor. This type of attack could potentially disrupt or damage devices, such as IoT mechanical devices, that could be continuously triggered until the mechanical component is over-stressed.

Another violation of the standard which leads to a vulnerability (in all brokers but Mosquitto) is the bad handling of long topics: in the MQTT standard, the maximum length topic is 65536 bytes. However, trying to publish to a very long topic (>4096 bytes) leads to a disconnection of the client. A malicious actor that can inject (even indirectly, think user-provided information) some characters in the topic may cause a Denial of Service for that client. Even worst, in Aedes there is a crash of the broker itself, leading to a *Denial of Service* for the entire MQTT network.

Certain parts of the standard are frequently misunderstood or interpreted differently by different brokers. For example, in the “Double Publish QoS 2” test, nearly all brokers (with the exception of Mosquitto) violate the “unique identifier” feature of MQTT in various ways. While this violation does not have a direct impact, it can potentially be exploited if a client library exhibits poor handling of this situation.

Among all brokers, our tests show that Mosquitto seems to be the strongest one in terms of MQTT standard adoption, and so the safest from a security point of view. Client libraries, instead, have shown only minor issues, many of them relating to encoding errors or long topic subscription issues. Our tests show that they are quite robust, sometimes even better than some brokers.

[13] A “replay attack” is an attack where an evil agent in the network can copy and send a packet again, causing the action to be re-executed.

4.6 Discussion

In this research, we conducted an empirical study of the most popular implementations of MQTT brokers and clients. MQTT is an essential technology for the Internet-of-Things (IoT) ecosystem, as it is widely used by IoT applications that run on devices with limited computational power, such as the SeismoCloud Earthquake Early Warning (EEW) system. Its ubiquity is due to the low resource requirements and the large number of open-source libraries that implement MQTT, making it easily accessible for developers.

We examined any deviations from the standard that could lead to inconsistent or critical states in applications. We also tested a physical smart-home device as part of our experiments. Our findings show that, while most of the libraries we tested handle most interactions correctly, some vulnerabilities are present. These could be exploited to target both brokers and client libraries, mainly by exposing them to Denial-of-Service attacks, which might jeopardize EEW networks that use them without further protection.

The presented results highlight the importance of carefully reviewing and testing the behavior of different brokers and client libraries to ensure that they comply with the MQTT standard and do not present any vulnerabilities that could be exploited. In the meantime, some of these vulnerabilities can be mitigated (at least partially) by implementing TLS (which unfortunately is not widely used for this protocol) or other layered protection (like Virtual Private Network) wrapping MQTT to protect connections in untrusted networks.

This research has been presented at *ITASEC 2021* and published as [23].

Chapter 5

Earthquake Detection at the Edge: IoT Crowdsensing Network

As the main research of this work, we focus on removing the most critical Single point of failure of EEW systems, which is the “fusion center”. State-of-the-art Earthquake Early Warning systems rely on a network of sensors connected to a fusion center in a client-server paradigm. The fusion center runs different algorithms to detect earthquakes on the whole data set. We propose moving computation to the edge, with detector nodes that probe the environment and process information from nearby probes to detect earthquakes locally. Our approach tolerates multiple node faults and partial network disruption and keeps all data locally, enhancing privacy. This chapter describes our proposal’s rationale and explains its architecture. We present an implementation using Raspberry, NodeMCU, and the CrowdQuake machine learning model.

5.1 Creating a Peer-to-Peer distributed EEW architecture

Many countries perform earthquake detection through a national network composed of hundreds of high-precision seismic stations. Each seismometer in a station has high sensitivity and can perceive low-magnitude or very distant earthquakes (sometimes from other countries). By interpolating signals from three or more seismic stations, it is possible to localize the epicenter and compute the magnitude. These seismic networks are costly, and building them might be a decades-long process. Some countries use such networks to provide an Earthquake Early Warning (EEW) system, such as the Japanese one by the Japan Meteorological Agency (JMA) [35].

An alternative that has been gaining traction in the last decade is the crowdsensing EEW network, based on the availability of low-cost Micro-Electro-Mechanical Systems (MEMS) sensors together with the widespread Internet connection. Volunteers can participate in crowdsensing using their smartphone or an Internet-of-Things (IoT) sensor as a seismometer. Crowdsensing EEW tackles the problem of

MEMS's low precision by trading quality with quantity. By leveraging the lower cost of intelligent devices and distributing such costs among participants, these systems have a large user base and thus many seismometers, i.e., thousands or more. This approach has proven to be successful, for example, in [3], at least to estimate the epicentral area and an approximated intensity.

Existing crowdsensing EEW networks adopt a centralized processing approach: seismometers send the collected data to a fusion center that processes it to understand whether the report is a quake signal or not. In some cases, the sensors send the MEMS raw signal to the fusion center (dumb approach). Other times, the edge sensors perform partial calculations (limited due to resource constraints) and send preprocessed data. The fusion center performs the detection work, adopting a post-processing filtering that involves signals from many local seismometers to exclude false positive or false negative earthquake detections.

As explained in the next section, this architecture has some drawbacks that motivate the following research.

This work proposes a peer-to-peer distributed EEW architecture that is radically different from existing architectures. It is based on edge computing, where each node in a mesh network can sense the environment and detect a local earthquake without relying on a fusion center or a leader node. It can share this information with its neighbors, propagating the detection. This system keeps all data locally and can tolerate multiple node faults and partial network disruption.

5.1.1 Motivation

Ideally, an EEW system should be fault-tolerant, which means that if one or a few of its components fail, the overall system can continue to work seamlessly. In the absence of fault tolerance, the High Availability property (HA) might be helpful: HA systems tolerate a stop or downtime between the fault and its recovery. However, if a fault occurs during an earthquake, the EEW system might be unavailable at a critical moment.

EEW systems currently built or proposed in the literature do not have a fault-tolerant architecture, as the fusion center constitutes a Single point of failure (SPoF) that, if unavailable, prevents the entire system from working. Also, it is essential to consider the connections to the fusion center, such as international internet links with sensors, as a system component that can fail, causing the isolation of the fusion center and thus the unavailability of the EEW system.

The first motivation behind our proposal is to solve the availability problem. As we describe in Section 5.3.6, our system can tolerate multiple node faults and some partial network disruption.

The mainstream EEW architecture also has a privacy-related issue. It is possible to process raw accelerometric data to extract information other than seismic data. For example, it is possible to detect some spoken words using the accelerometer in place of the microphone [69]. As another example, during the analysis of the data we present in this work, we noted that we could correlate the noise level of seismometers in our homes with the presence of people in the house. So, sending raw seismic signals to a fusion center might expose them to unwanted processing that can violate the users' privacy: An attacker could discover information about a

family's life habits or even extract words from private conversations.

Our proposed architecture enhances the privacy of the detection, keeping the collected sensitive data locally in private places by a crowdsensing EEW system.

5.2 A comprehensive overview of decentralized EEW systems

Decentralized approaches to earthquake detection have been studied for years. Tsitsiklis [64] proposed a decentralized detection architecture where a central system (named "fusion center") collects "messages" from sensors. In EEW, a "message" can be a signal sample that the sensor claims to be a quake signal. Faulkner et al. [27] proposed a new version of this architecture for massive noisy sensors networks, and Cochran et al. [15] presented an implementation using accelerometers connected to laptops and workstations, named QCN (Quake-catcher network). Similarly, MyShake, proposed by Kong et al. [40], is a machine-learning-based EEW system that uses smartphones. The Earthquake Network (Finazzi et al. [28]) is a different research project that uses smartphones and spatial correlation to detect quakes. SeismoCloud [53] is another earthquake early warning system built using smartphones and Internet-of-Things devices. All these systems differ from our proposal as they rely on a central system to collect all reports and make the final decision.

Another approach is the one described by CrowdQuake, from Huang et al. [37]. CrowdQuake runs a Convolutional-Recurrent Neural Network (CRNN) on the fusion center, while smartphones at the edge collect different-length samples and stream them to the fusion center. While relying on the fusion center, this system differs from the previous ones because it can perform both the decision and the detection in one step since the accuracy of the Convolutional-Recurrent Neural Network is very high. It also shows some architectural limits that we describe in Subsection 5.2.3.

CrowdQuake+ [68] is an extension of the CrowdQuake network: While the original network leverages only on smartphones, CrowdQuake+ is able to process data from Internet-of-Things sensors. However, the overall architecture is the same, as well as the limitations discussed in Subsection 5.2.3.

Fischer et al., in [30], described SOSEWIN, a self-organizing Earthquake Early Warning system using a wireless mesh network. They use a hybrid approach, where nodes act as local fusion centers. Instead, in our proposal, each node is independent of others.

Lee et al. [42] presented a custom-made board for EEW. The board contains common chipsets (like ESP8266) and custom software with an Artificial Neural Network for detection. They propose to send the alert to nearby intelligent devices (TV, smartphones) via low-range transmissions (Bluetooth Low Energy) or Home Automation solutions for early warning alerts. Unlike other solutions, including ours, they do not use a network to send the alert to nearby houses; their alert is "personal".

QuakeSense, presented by Boccadoro et al. in [11], is an EEW system based on LoRa, a Low Power Wide Area Network (LPWAN) technology. In their proposal, sensors send information (like vibrations) using LoRa to local base stations "LoRa gateways", which relay these data to the fusion center of QuakeSense. LoRa, un-

like Wi-Fi, allows QuakeSense to be deployed in remote locations with no access to the power grid: LPWAN transceiver's power consumption is low, allowing deployment with batteries and solar power. However, it leverages the same centralized architecture as others do.

5.2.1 Quake-Catcher Network

The QCN, Quake-Catcher Network, is an earthquake early warning system built by volunteers to "fill the gap between the earthquake and traditional networks" [15]. It has been built over BOINC [4]. QCN uses MEMS sensors in some laptop brands (usually in the anti-shock subsystem) and some USB accelerometer brands. According to [15], the sensitivity of these accelerometers is low, and the network is well suited for an earthquake of magnitude greater than 5.0.

QCN uses Z-Scores (also known as standard scores) to detect potential quakes: when z is above 3, the sensor sends all relevant data to the fusion center, such as the max amplitude or timestamp. Then, the center will again use a Z-Score against the number of reports in a given area and time slice; a value of $z > 6$ will trigger an EEW.

5.2.2 MyShake

MyShake [40] is an earthquake early warning system developed by UC Berkeley Seismology Lab, designed to collect and process data on a smartphone and send possible quakes to a fusion center for confirmation. Volunteers can download a mobile application on their smartphone to join the network.

The MyShake mobile app reads the signal from the smartphone's internal MEMS accelerometer. Then, it uses an artificial neural network to detect potential quakes and sends quake candidates to the fusion center, where a clustering algorithm reduces false positives.

5.2.3 CrowdQuake

CrowdQuake, by Huang et al. [37], has a layered approach to earthquake detection. The lower layer, composed of dedicated smartphones or custom Internet-of-Things devices, senses the seismic data and streams it to an intermediate layer of "gateways". Each gateway is a GPU-equipped server that processes seismic samples from each sensor in a CRNN. Then, it sends data to a third and fourth layer for notification, monitoring, and visualization.

Unlike others, CrowdQuake requires a stable and low-latency network connection between sensors and gateways because samples are sent for detection from the accelerometers to the gateways, which act as fusion centers. This requirement is a substantial limitation for the deployment, especially in remote sites.

5.2.4 SOSEWIN

SOSEWIN [30] is a decentralized wireless mesh network of sensors built using standard PC boards and external sensors. There is a hierarchy in the network between nodes, built using a leader election algorithm; the two most important kinds of

nodes are the *leading* node and the *sensing* node. A leading node receives information from five sensing nodes and manages alerts from them and neighbors leading nodes. A sensing node filters the accelerometer sensor information with an Infinite impulse response passband filter and some thresholds, using an internal state machine to refine the detection.

The leading node acts as a fusion center of a cluster of sensing nodes. Using leader election for leading nodes, SOSEWIN obtains High Availability.

5.2.5 SeismoCloud

In SeismoCloud [53], smartphone apps and Internet-of-Things devices make the sensor network and connect to a fusion center. Both types of sensor nodes run an algorithm based on dynamic Z-Score for candidate quakes detection and send candidate quakes to the central server, where a clustering algorithm filters out false positives.

5.3 Proposed Architecture

We propose a new architecture for EEW systems based on crowdsensing and the complete detection of earthquakes at the edge. This architecture, called SeismoCloud 2.0, is an evolution of the SeismoCloud architecture [53]. The goal is to achieve fault tolerance using low-cost commodity hardware while enhancing the privacy and scalability of the system. The idea is to use a fully decentralized approach to detect earthquakes, creating a partial-mesh network (with no single point of failure) that can survive multiple network and hardware faults.

First, we describe each component of the system. Then, we draw a comprehensive picture of the architecture.

5.3.1 Probes, Detectors, and Local Authorities

Our network has two roles for edge devices: the *probe* and the *detector*. A probe is a sensor capable of picking up the acceleration signal and streaming it to the detector. On the other hand, the detector's primary role is to run the detection algorithm over the data stream from probes and match if there are any signs of an earthquake wave. One detector can receive data from multiple probes. As the hardware for a probe is very cheap, we expect that some detectors will have multiple probes attached. Multiple probes can maximize the chance of detection because they may fail or miss some vibrations (if they are improperly installed).

In addition, the two roles can be assigned to the same detector device if it includes an accelerometer. They can both read the accelerometer signal and run the detection algorithm simultaneously. For example, smartphones and System-on-Chip boards have enough computing power to support such operations.

The detector is also equipped with a local alert device (speakers, blinking lights) to alert its owner locally. Alerts are triggered both by local earthquakes and relevant remote ones.

The Local Authority is a central system that supports the network with non-critical services: it helps nodes discover other nodes and receives EEWs from the

network to help local safety authorities prepare rescue operations. The Local Authority is stateless due to the nature of its services. It is possible to have more than one Local Authority instance to obtain high availability or load balancing. They should share available information, although they do not require synchronization but can implement eventual consistency. Such a connection can be implemented, for example, using a gossiping protocol between Local Authorities. Nodes will discover the local authority via DNS queries.

5.3.2 Network Architecture

The network architecture that we propose is a partial mesh (Figure 5.1). While a full mesh would be desirable for information exchange between detectors, it is entirely unfeasible due to the resource constraints of Internet-of-Things devices and commodity network connections.

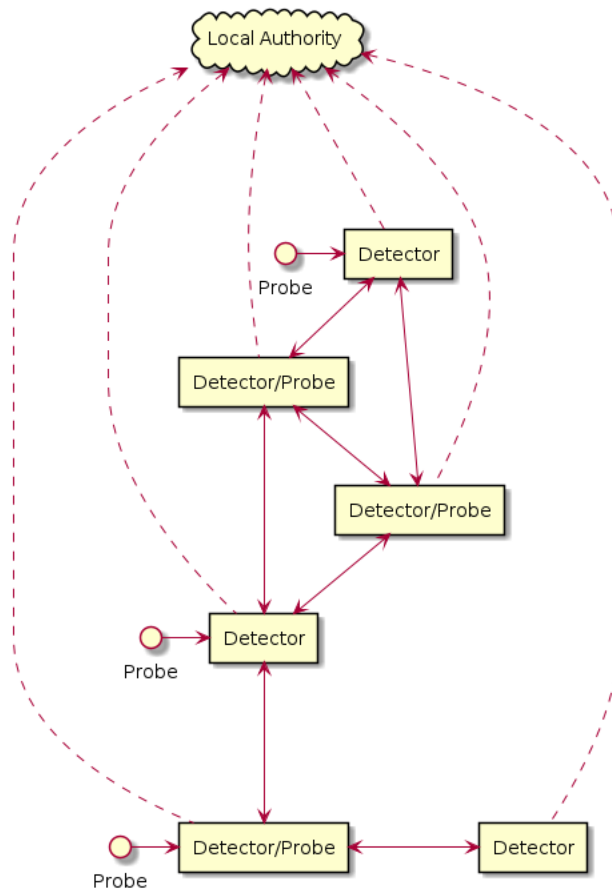


Figure 5.1. Network architecture example: six detectors (three of which have an embedded probe) and three probes. Detectors are linked to neighbors based on their location.

Each node of the partial mesh is a detector. It is connected to neighbor detectors using direct peer-to-peer links. Detectors exchange EEW messages using a gossiping protocol: Each message is forwarded to neighbor detectors until it reaches a certain distance from the reported quake location. The threshold distance can be set by

deriving it from the acceleration value detected on the nodes that are sending the EEW messages.

Nodes are connected to at least one Local Authority to advertise their presence to others and get a list of neighbors to connect to. Due to the design of the Local Authority, a node can connect to any Local Authority in the network. Moreover, detectors report all quakes to the cloud service of the Local Authority to relay this information to other services (e.g., rescue teams and TV broadcasts).

Probes connect to their nearest detector directly. They are not connected among themselves and do not participate in any message exchange between detectors.

5.3.3 Bootstrap Sequence

When a detector powers up for the first time (Figure 5.2), it starts a discovery phase of its neighbors using a registration and discovery service of the local authority. The detector advertises its presence by using that service, providing its location, and in turn, it receives the list of neighbors and details on how to connect to them.

After completing this exchange, the detector will connect to the indicated neighbors and keep those connections alive, ready to relay information about early warnings. Periodically, the detector repeats the registration and receives a new list of neighbors.

Differently, probes query the local authority for the detector they should connect to. They do not advertise any information to the local authority (see Figure 5.3).

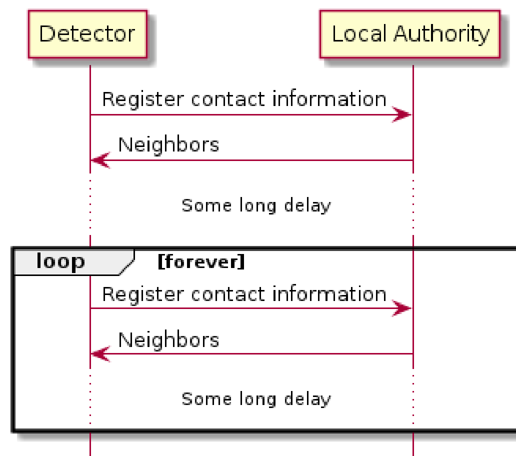


Figure 5.2. Detector bootstrap sequence diagram.

5.3.4 Detection Pipeline

Figure 5.4 shows the detection pipeline. Probes stream the signal using their network connection to the detector. The detector has one buffer per probe, where it collects and stores the accelerometer signal for some time. A sliding signal window is extracted and sent to the detection algorithm at given intervals.

Suppose the detection algorithm detects a quake from any of its probes. In that case, the detector relays the earthquake alert, together with its location and the signal data (Table 5.1) to the local authority and neighbors.

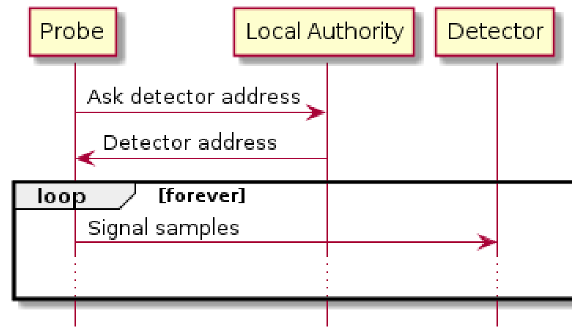


Figure 5.3. Probe bootstrap sequence diagram.

Table 5.1. Earthquake Early Warning message content. This message is relayed between *detectors*.

EEW Message Content	Description
Timestamp	Timestamp of the detection
Origin Location	Coordinates of the detector which originated the message
Signal Data	Accelerometric samples

5.3.5 Scalability

The detectors mesh network can scale to an infinite number of nodes. Each detector of the partial mesh is connected only to a few nearest neighbors. In addition, each message will reach a subset of the whole network as it is geographically limited, as described in Section 5.3.2. There is no need to scale up a central server to handle sensor traffic for detection purposes.

The Local Authority system should be scaled according to the number of sensors. Unlike the fusion center of the server-client model, a local authority instance is stateless and not involved in the detection pipeline or EEW message dissemination. Scaling it is more straightforward than scaling a fusion center.

5.3.6 Fault Tolerance

The network is fully fault-tolerant. A fault of one or few sensors will not stop the gossiping. An EEW message can be prevented from reaching the entire network only if multiple faults occur so that the network temporarily splits into two or more partitions. However, the more sensors in the network, the less the chance of having such a split. Even if this split occurs, it will not affect messages from other sensors inside other partitions. If sensors in the different partitions detect the quake, the EEW can still be sent to the whole network (albeit with different origins).

A fault on a specific sensor itself will not stop the detection: neighbors can still detect the earthquake, and they will still be able to pass information to others.

The Local Authority is using gossiping and eventual consistency: a fault in one or multiple Local Authority instances is not affecting the rest of the network. Nodes connected to faulted instances can switch to other instances with no data loss. In case of faults in all instances of the Local Authority, which causes the unreachability of its service, new nodes will not be able to connect to the network.

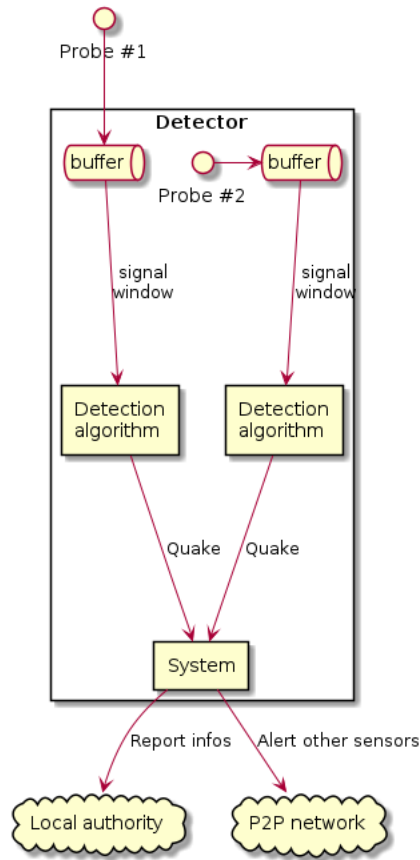


Figure 5.4. Pipeline diagram. Each probe is attached to a buffer that feeds the detection algorithm's dedicated instance. Probe #3 is internal, as this detector also has the probe role.

However, already connected nodes will keep their current connections. The Local Authority will not receive an EEW during the downtime, but the EEW message gossiping will not stop, and earthquake detection will remain active.

5.3.7 Privacy

Centralized systems that collect and analyze accelerometer signals in fusion centers present significant privacy concerns. In these systems, a massive amount of data is collected from many sensors and aggregated in a centralized location. This data collection creates a potential risk for data breaches and unauthorized access to sensitive information. Furthermore, the analysis of accelerometer signals can reveal a lot about an individual's movements and activities [6] and, in some cases, can even lead to the recognition of spoken words [70].

Decentralization can offer a solution to some of the privacy issues associated with centralized systems that collect and analyze accelerometer signals. In decentralized systems like the one proposed here, the signal is analyzed locally on each device, and no central authority can infer data from the accelerometer signal. Our approach reduces the risk of data breaches and increases user privacy by limiting the amount

of data shared with third parties. An attacker who wants to monitor sensors (for example, to recognize spoken words or detect the presence of people in a building) will need to attack specific sensors actively.

5.3.8 Practical Implementation Aspects

An essential aspect of crowdsensing EEW is an excellent practical implementation. Complex user interfaces and systems that are difficult to understand can create obstacles to widespread adoption, which is fundamental for these systems. Users of the proposed EEW system should be able to use it without computer science or domain skills. We suggest distributing our system by leveraging mobile apps (for smartphones) and IoT devices to overcome these difficulties.

Mobile apps can introduce users to the system (and, under the hood, they act as a sensor themselves, when and where possible). The app's User Interface will guide users to configure a new sensor and easily access the sensor's data. The app's design must exploit a User-Centered Design (UCD) approach to avoid mistakes that jeopardize the entire project.

Another vital aspect is the simplicity of installing and managing IoT devices. Users with no background in electronics or computer science should be able to install and run one or more detectors or probes. We suggest addressing this problem by leveraging companies that build custom boards on-demand: Users will receive a device that is no different from other intelligent boxes at home (such as smart TVs). Once this "box" is connected to the power supply and an ethernet cable (or Wi-Fi), the sensor will receive its location from the companion app on the user's smartphone, requiring no further configuration on the user's part.

We are in the process of designing and testing these implementation aspects.

5.4 Prototype Implementation

We present the following implementation as an example of the architecture described above. This implementation is currently running in a test environment.

5.4.1 Sensors Hardware

The detector device is a Raspberry Pi, made by the *Raspberry Pi Foundation*. It is a *System-on-Chip* board with various ports (Ethernet, USB, HDMI, I^2C, GPIO), Wi-Fi, and Bluetooth wireless chipsets. For the current prototype, we use the Ethernet port to provide an Internet connection to the detector, the I^2C bus to connect the accelerometer (to implement a Detector/Probe device), and a Wi-Fi card to create a dedicated Wi-Fi network for external probes.

We tested different device versions: 2B, 3B, 3B+, and 4 (Table 5.2).

The accelerometer is the MPU6050, widely used in low-cost IoT applications involving acceleration measurements. It has been demonstrated by Crisnapati et al. [19] and Lee et al. [42] that this accelerometer can be used in EEW applications. The sensitivity of such accelerometers allows the detection of significant earthquakes only.

Table 5.2. Hardware specifications for prototype detectors

	Raspberry Pi			
Model	**2B**	**3B**	**3B+**	**4**
CPU	BCM2836 4 x Cortex-A7 900MHz	BCM2837 4 x Cortex-A53 1.2GHz	BCM2837 4 x Cortex-A53 1.4GHz	BCM2711 4 x Cortex-A72 1.5GHz
RAM	1GB	1GB	1GB	4GB
Disk	64GB SD	64GB SD	64GB SD	64GB SD
Wi-Fi	-	2.4 GHz	2.4 / 5 GHz	2.4 / 5 GHz
Ethernet	Fast Ethernet	Fast Ethernet	Gigabit	Gigabit
GPIO	40 pin	40 pin	40 pin	40 pin

The MPU6050 provides a 100Hz feed via I^2C to the NodeMCU board (Table 5.3) or the Raspberry Pi board. The probe and detector roles merge by connecting the MPU6050 directly to the Raspberry Pi.

Table 5.3. Hardware specifications for prototype probes

Model	NodeMCU
CPU	106Micro L106 160 MHz
RAM	128kBytes
Disk	4MBytes
Wi-Fi	2.4 GHz
Ethernet	-
GPIO	13 pin

Probe sensors use an MCU board and an accelerometer, packed to run on 5v from a power supply or battery. They transmit values using the Wi-Fi connection to the detector. The MCU board is the "NodeMCU DEVKIT" that contains an ESP8266 SoC [41], based on ESP-12 hardware. It has multiple GPIO ports and an integrated Wi-Fi network connection.

5.4.2 Software

The Detector runs Raspberry PI OS (previously known as Raspbian), a Debian-based GNU/Linux distribution. We use "Podman" to manage the lifecycle of our software, an Open Container Initiative (OCI) image runner alternative to "Docker" that we chose as it is *daemon-less*. The absence of a central process makes Podman more robust and less resource-hungry than Docker. Podman runs a container with an implementation of our proposal that is using the CrowdQuake CRNN [37] detection algorithm. We built the primary container executable using Go (and TensorFlow C bindings). The use of containers simplifies the deployment of new algorithms for testing.

The Probe runs a customized firmware that we built using the Expressif SDK for Arduino. The firmware reads data from the accelerometer sensor and sends

the stream via WebSocket to the detector. The connection uses WebSocket to be compatible with HTTP middlewares, such as network proxies and firewalls. The firmware checks for updates and configuration at probe boot, querying the local authority. If it fails, it still connects to the detector. We chose this Probe software architecture after comparison with an MQ Telemetry Transport (MQTT) implementation. In our tests, we found MQTT too complex for this scenario: while MQTT requires a broker, different publisher and subscriber roles, and topics, in our case, we only had a publisher (the Probe) and a subscriber (the Detector) with no need for topics. Thus, the implementation of the MQTT was overly complex for our purpose, with no advantages.

We developed the Local Authority software using the Go language. In our experiments, the Local Authority runs on our servers. It exposes Application Program Interfaces for sensor discovery and debugging web pages. This implementation is not fully scalable yet, but we successfully tested it with more than 1000 detectors.

5.4.3 Detection Pipeline

The primary container exposes a WebSocket endpoint for probes, and it reads the accelerometer connected to the GPIO of the Raspberry Pi. The code spawns multiple processes (based on the number of cores/CPUs of the Raspberry Pi) so that reading a local sensor while receiving a network stream does not interfere with each other.

The probe data stream is buffered in memory by the primary container to have a 2-second signal window (200 values), with a 1-second sliding window, as shown in Figure 5.5. The signal window is then sent to the detection algorithm, the CrowdQuake CRNN (running on the CPU). The structure of the CrowdQuake CRNN is briefly reported in Figure 5.6 and presented by Huang et al. in [37].

The detection CRNN is run every second (as it receives a new set of samples each second), and it looks for quakes in the last 2 s in the probe signal buffer. Each probe has its independent buffer, and CRNN is run in parallel on each buffer.

5.5 Results' evaluation

We ran our prototype in three different scenarios: First, we tested the pipeline using a single software-only detector, feeding it with dummy accelerations to verify the system's soundness. Then, we built an actual probe to test the detection speed in real hardware. Finally, we tested a network of sensors to measure the elapsed time between the first detection and the EEWs.

The system's soundness was tested using the CrowdQuake dataset [37]. The dataset comprises 174 tracks from natural earthquakes and 79 tracks from "noise". Each earthquake track is made of 30'000 accelerations triples (X, Y, and Z), sampled at 10Hz. Noise tracks are recorded using smartphones in day-to-day activities [37].

As expected, feeding the CrowdQuake dataset into the pipeline triggers the CrowdQuake-based detector in the same way as using their neural network directly (e.g., for evaluation purposes). This result was anticipated because CrowdQuake

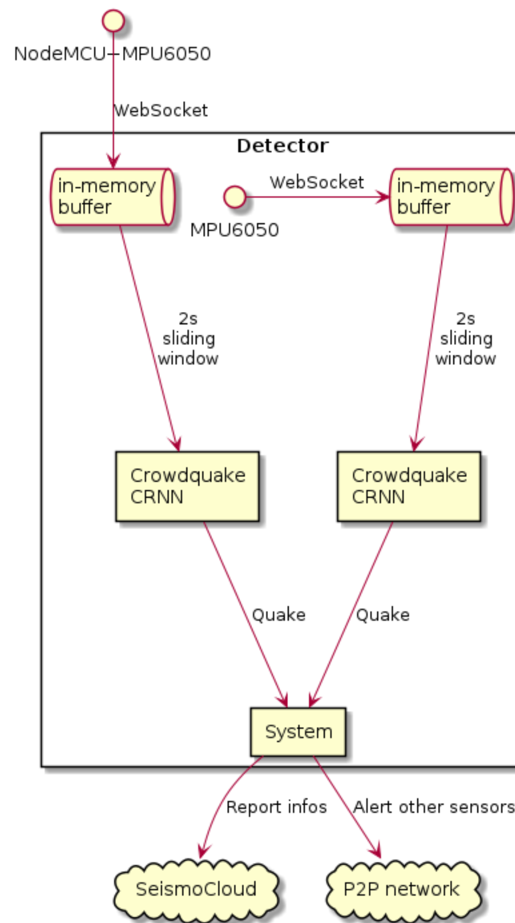


Figure 5.5. Prototype pipeline. The detection algorithm is the CrowdQuake CRNN, and the time window is set to 2 s.

CRNN runs unmodified in our proposal (so there was no reason to expect different performances).

The detection pipeline can analyze and output the result in a few milliseconds, as shown in Table 5.4. The detection latency for Raspberry Pi 4 is the lowest (Figures 5.7 and 5.8) thanks to the faster processor and different onboard bus wiring. This lower speed for detection opens up the possibility for incrementing the detection frequency, which is currently 1 Hz, to higher values, depending on the platform and the number of probes for each detector. Further analysis is needed to assess the benefits and limits of having sub-second detections.

We also tested the impact of having multiple probes feeding data concurrently in a detector. We loaded the detection algorithm in memory and streamed the same dataset we used in previous tests. As shown in Figure 5.7, the impact of having multiple parallel executions is minimal in the latest Raspberry Pi version, while it can be significant in previous versions.

The Go garbage collector is causing a spike that nearly doubles the detection latency when it executes concurrently with the detection algorithm (primarily visible in Raspberry Pi 3B/3B+, Figure 5.8). It can be further optimized by running the

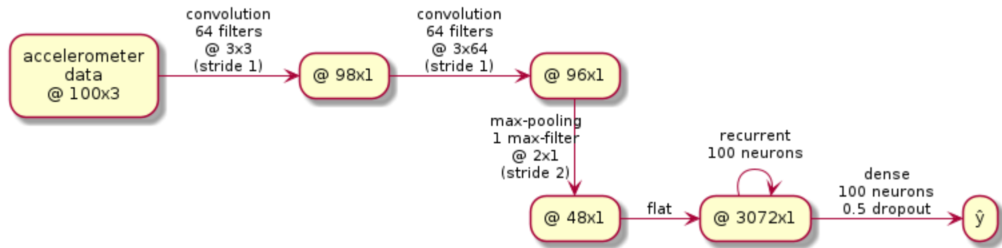


Figure 5.6. CrowdQuake's Convolutional-Recurrent Neural Network.

Table 5.4. CRNN response speed for 2-second signal (200 values), averages on 300 samples

Raspberry Pi	Average time	Standard deviation	90-percentile
2B	27.19 ms	1.61 ms	28.73 ms
3B	27.78 ms	5.36 ms	30.74 ms
3B+	22.44 ms	4.29 ms	24.59 ms
4	7.84 ms	0.41 ms	8.37 ms

garbage collector manually, rewriting the buffer code (where most of the allocation takes place), or switching to a language with no automatic memory management (e.g., Rust, C).

The detector code loads the network in memory, and it launches an "empty" run to pre-fill the system cache so that the first latency test is not affected by the cache miss, and it is comparable to all subsequent tests (Figure 5.8).

Finally, we tested a network of detectors to measure the time between the first detection and the time when the message was received in the network. We built the test network using 20 instances of the node code running in a single machine, each instance connected to 10 random neighbors (partial mesh). Figure 5.9 represents the network. The test machine is a Dell XPS 15, 6-Core i7 @ 2.20GHz. Delays of 5 to 205 milliseconds were randomly injected into the packet transmissions to emulate the network latency. We performed six tests using the same topology but different points of origin for EEWs.

As shown in Figure 5.10, in all but two test, the EEW reached every node of the network in less than 450 milliseconds (including simulated network delay).

5.5.1 Limitations

We have not address yet the security of the network. This area has multiple aspects, mainly related to the trust in early warnings from neighbors' sensors. Today, any byzantine probe in the network can cause a false alarm by sending an alert with a signal that resembles an earthquake wave (downloadable from public datasets). A byzantine detector can even send an earthquake early warning. We originally designed the protocol message so that a signal could be sent together with the EEW as a primary security measure: We planned to check that the signal attached to the EEW was triggering an EEW by replicating the detection on each detector. However, we did not clearly define or implement this part at this time, so we omitted

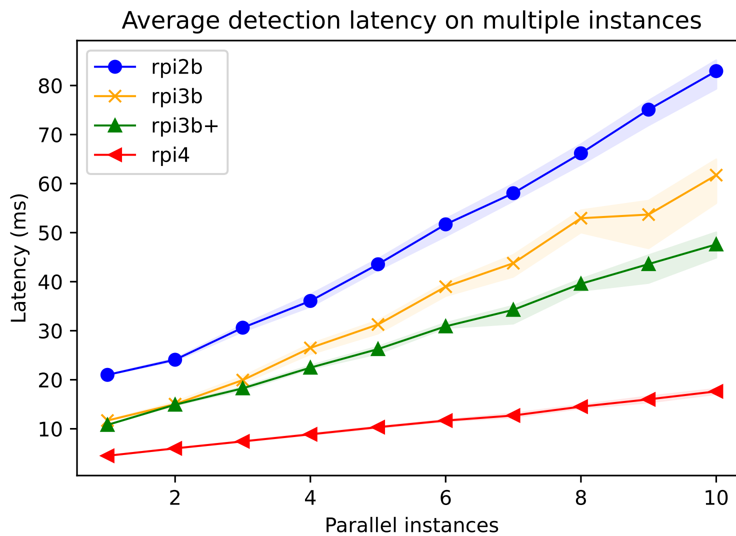


Figure 5.7. Detection latency for the ML model when multiple probes are sending data to a single detector.

this from the proposal, and it constitutes future work.

Another limitation is that we did not address the problem of setting up a secure transmission between peers. In the prototype, we implemented a plain-text protocol in which attackers can eavesdrop on the message exchange (loss of confidentiality) and inject or modify messages. However, this problem can be solved trivially by using widely studied and deployed protocols such as TLS [24].

It is worth underlining that the accuracy of the detection algorithm plays a central role in the trust in this system. Users will trust an EEW system with this architecture only if they receive very low false positives and false negatives EEWs. We are working on this by allowing different detection algorithms to plug in to compare their accuracy.

We did not consider epicenter location estimation while designing this proposal. In a dense network, the sensor's location that detects the earthquake before other sensors can be considered an approximation of the epicenter. However, the approximation error depends on how close the sensor is to the real epicenter, its sensitivity, its physical installation, and several geophysical characteristics, such as the terrain composition (which influences quake waves). Further analysis should be conducted to minimize the estimation error.

5.6 Discussion

We described a crowdsensing EEW architecture that moves the computation to the edge, with detector nodes that probe the environment and process information from nearby probes to detect earthquakes locally. Our approach tolerates multiple node faults and partial network disruption and keeps all data locally, enhancing privacy. We described our proposal's rationale, explained its architecture, and presented an implementation using Raspberry, NodeMCU, and the CrowdQuake machine learning model.

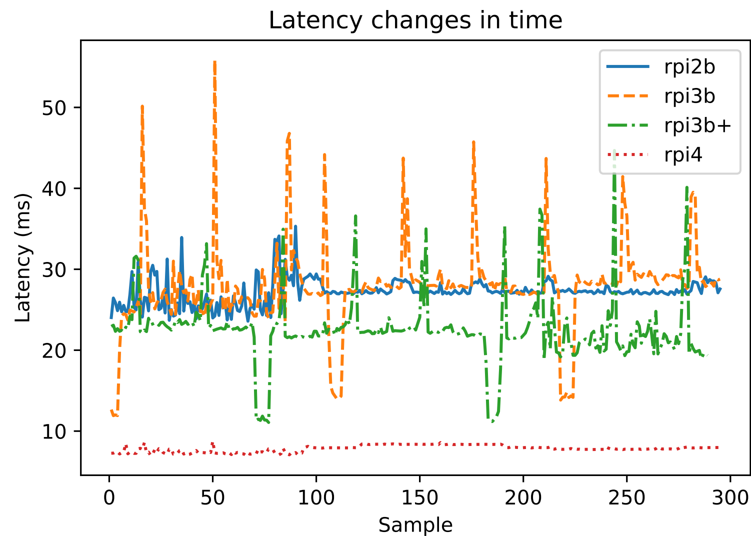


Figure 5.8. Detection latency changes in time. The model was tested over 300 samples. Each point represents the latency for a single sample.

Ongoing research on this topic focuses on the security of this architecture and its implementations. It is essential to find a viable and secure solution to the problem of trust in *peer-to-peer* EEW message exchange to use this architecture in crowdsensing networks. At least the system should resist some byzantine nodes.

We are developing an app that will be integrated into this architecture both as a sensor and as a "companion app" for IoT sensors. The app is being built using a user-centered design as we aim to produce an interface that is easy to use. The app will be able to provide the user with all data from its sensors and monitor them. Thanks to the app, we will be able to test the system from the point of view of a typical user, starting from initial configuration to maintenance to data access and retrieval.

Another focus of our current research is the implementation of the architecture that we presented over low-power, long-range radio protocols (LPWAN), such as LoRa. These wireless protocols are very effective in long-range transmissions and power efficiency compared to Wi-Fi networks. However, they usually lack coordination, so collision-avoidance algorithms such as water-filling cannot be used (unlike in LoRaWAN, where water-filling can be implemented in Base Transceiver Station (BTS) [21] or in the control plane [20]), and we will need to overcome this limitation. The detection network can be deployed seamlessly from big cities to remote sites using LPWAN for IoT [11], LTE for smartphones, and Fixed-Wireless Access or FTTx for others. In big cities the Internet is ubiquitous, while in remote sites LPWAN can be a low-cost, low-latency alternative to satellite links.

The proposal and the results in this chapter have been published in *MDPI Information* journal as [23].

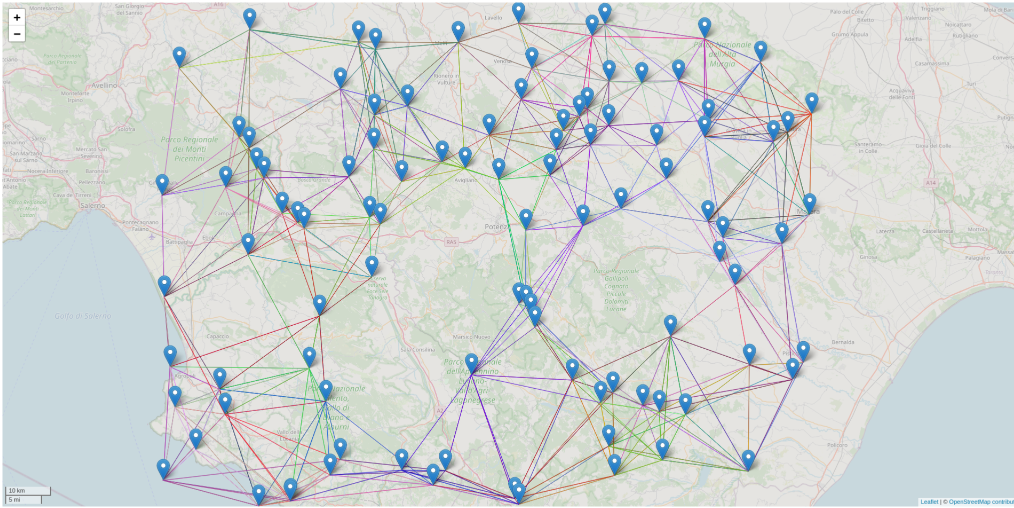


Figure 5.9. Detector network used in simulations. Lines between detectors represents connections. Detectors are placed in a random pattern. Tile images by OpenStreetMap [51].

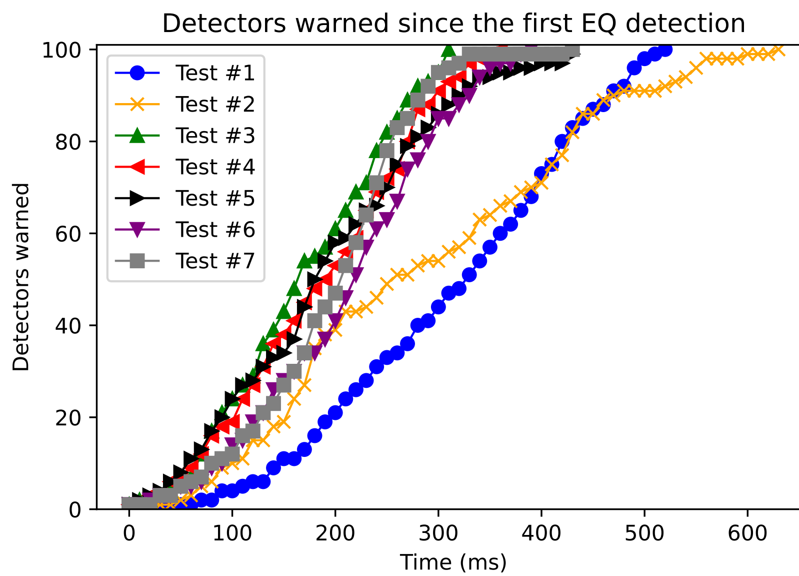


Figure 5.10. Number of detectors warned since the first detection on the source node. Note that the number of detectors is sampled each 10 ms. The y axis represent the cumulative number of detectors alerted.

Chapter 6

Evaluation of SeismoCloud 2.0 architecture via earthquake epicenter estimation

6.1 Estimating the epicenter in EEW networks

Estimating an epicenter is crucial for an EEW application. Not only for coordinating alert systems but also for first responders: directing first aid to the area hardest hit by the earthquake can have a strong positive impact on the outcome of rescue operations.

In order to detect the earthquake and its epicenter from sensor data, many existing crowdsensing EEW networks use a centralized strategy: information is sent to a fusion center for processing. In some systems, the sensors provide the MEMS raw signal; in others, the edge sensor performs an initial analysis and sends the result. The task of earthquake detection is assigned to the fusion center, which uses post-processing filtering to filter out false positive or false negative earthquake detections using signals from several local seismometers in the same area. A fusion center can also estimate the epicenter using standard methods already employed in non-EEW networks by having a full view of an area [3].

Another approach to low-cost EEW is fully decentralized: networks of peers can analyze their own sensors' data and alert neighbors about a possible earthquake. Each network node can sense the environment, detect a local earthquake (without relying on a fusion center or a leader node) and share the information with neighbors. Estimating an epicenter in such networks requires the cooperation of other nodes and less computationally expensive algorithms.

In this chapter, we present an algorithm that uses a novel approach for estimating the epicenter that can run on nodes in a fully decentralized sensor network.

6.2 Existing approaches and their limits

Multiple EEW systems based on crowdsensing, which are operational or in research, utilize a centralized architecture and traditional methods to estimate earthquake

parameters such as the location of the epicenter; MyShake [3] is a state-of-the-art example of such systems.

The need and utility of decentralized EEW systems have been demonstrated [54], especially with regards to public safety; ACROSS [55] is a small network of seismic stations deployed in strategical regions of Kyrgyzstan, that performs EEW in a decentralized way.

Pujol [56] compared three methods to locate an earthquake's epicenter, including the P-S waves approach, which consists of analyzing the travel times of the P and S waves at each seismometer. This method—widely adopted and implemented in MyShake and ACROSS—first computes the distance between the epicenter and each seismometer, using the travel time difference. Then, it uses these distances as the radius of circles around seismometers, whose intersection is the estimated epicenter location.

The P-S waves approach can take several seconds according to the geophysical characteristics of the area, thus increasing the time needed to obtain an estimation, hence the risk of losing that data in the event of a disruptive earthquake. This approach is not optimal in contexts where quick response times are essential, such as the proposed process. Geometrical methods that do not consider the travel-time difference between the two waves [45] can also be used to estimate the epicenter of an earthquake, as demonstrated by Pujol et al. [57]. In this chapter, we present our geometrical strategy.

6.3 Proposed process

The proposed process estimates the position of the earthquake's epicenter by analyzing the state information that seismometers collect over time. The process is fully decentralized and executed locally by each seismometer; the relevant state information is composed of the position of the nodes that detected an earthquake, along with the detected intensity and the timestamp.

Subsections 6.3.1 and 6.3.2 contain a description of the architecture and principles of the proposal; subsection 6.3.3 explains how it works and solves the earthquake epicenter estimation problem; subsection 6.3.4 presents the conducted experiments and the obtained results.

6.3.1 Base architecture

The work presented in this chapter is based on the SeismoCloud 2.0 architecture described in Chapter 5. The following paragraphs summarize the subset of the SeismoCloud 2.0 architecture relevant to the proposed process.

Seismometers A *seismometer* is a device composed of two internal components: the sensor (probe) and the detector (controller). Multiple sensors can be attached to the same detector. The sensor can capture ground motion signals (vibrations), whereas the detector feeds the signal into the detection algorithm and performs local earthquake alerting. Each seismometer has a peering connection with nearby seismometers used to exchange messages. The network can thus be seen as a graph (N, V) where N is the set of seismometers and V is the set of peering connections.

Messages When a seismometer's detector classifies a signal as an earthquake, an EEW message is created and sent to all the connected peers. EEW messages contain the detection timestamp, the geographical coordinates of the originating detector, and acceleration samples. Messages are then forwarded to other seismometers that are not peers of the originating node, thus realizing a *gossiping* behavior by which eventually the whole network receives the alert. Messages are encapsulated in an envelope that also mentions the sending and receiving nodes; in the case of a gossip message, the sender is not equal to the originating node.

6.3.2 Process principles

As introduced before, the process solves the epicenter estimation problem by analyzing the recorded earthquake intensity of each node. Hence, the detector needs to map the input signal to a numerical value comparable to the Modified Mercalli intensity scale (MCS). Said mapping is achievable by using the correlation between Peak Ground Velocity (PGV), Peak Ground Acceleration (PGA), and the instrumental Mercalli scale [67].

The following section describes how the intensity information is integrated into the distributed system and used to solve the estimation task.

Messages

The EEW messages structure is modified as shown in Tab. 6.1 to include the intensity.

Table 6.1. Structure of an EEW message

Item	Description
Timestamp	Detection instant
Origin node	Identifier of the detecting seismometer
Origin location	A (latitude, longitude) pair describing the location of the detecting node
Intensity	Detected intensity in MCS scale

State table

Each node maintains a state table (described in Tab. 6.2) that associates other nodes with notable data extracted from direct and gossip messages. The table represents the node's view of the system's global state regarding the information needed to perform the epicenter estimation.

A node N updates its state table according to triggers:

a) N detects a vibration: a message msg is generated;

b) N receives a message msg.

In either case N updates its state table adding the tuple: `(msg.originNode, msg.originLocation, msg.intensity, msg.timestamp)`.

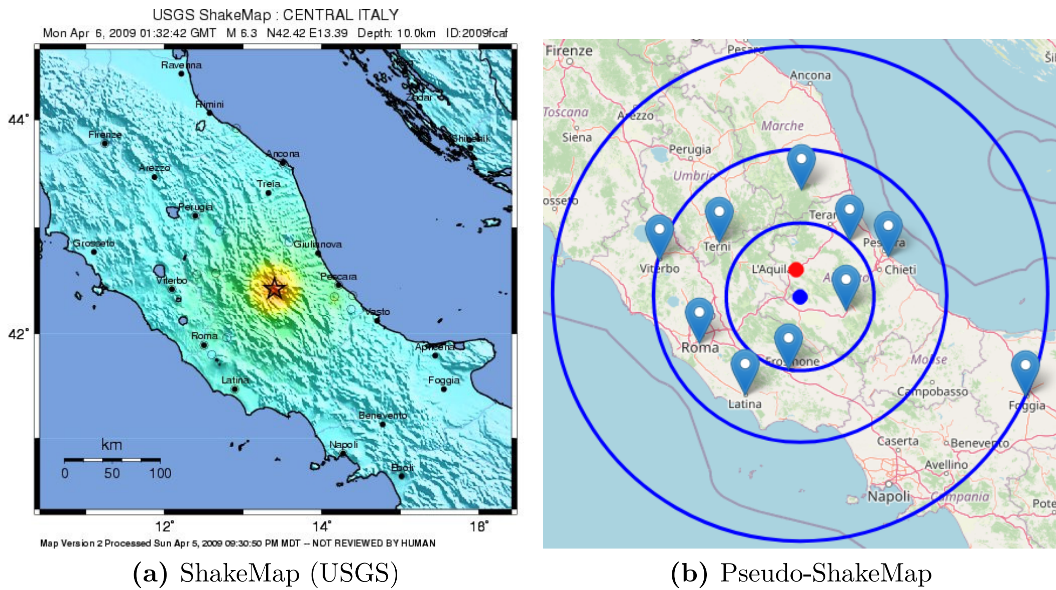
Table 6.2. Structure of a node's state table

Node	Location	Intensity	Timestamp
N_0	(lat_0, lon_0)	i_0	ts_0
N_1	(lat_1, lon_1)	i_1	ts_1
...	...	...	...
N_n	(lat_n, lon_n)	i_n	ts_n

Given **Trigger b)** and the **gossiping** behavior, all nodes have the same state table eventually. Since the estimation procedure uses the information in the state table, eventually, all nodes perform the same estimation.

Pseudo-ShakeMap

A **ShakeMap** is a standard graphical tool used to represent the *degree of shaking* of a geographical area in the event of an earthquake. In ShakeMaps, MCS intensities are associated with colors, which are interpolated to compose a gradient.



(a) ShakeMap (USGS) (b) Pseudo-ShakeMap

Figure 6.1. Comparison between ShakeMap and pseudo-ShakeMap for the L'Aquila (Italy) earthquake of Apr 6, 2009.

In the context of the proposed process, we can define the **pseudo-ShakeMap** as the graphical visualization of the result of the proposed estimation procedure; the pseudo-ShakeMap is a simplification of the ShakeMap because of the following properties:

- absence of interpolated coloring;
- the geological features of the terrain are not considered;
- use of concentric circles to represent intensity rings;

- circles are built according to simple rules, considering nodes' position and recorded intensity.

Circles in the pseudo-ShakeMap represent *intensity rings*, namely circular partitions of the plane that inscribe seismometers that detect vibrations of a certain intensity. The radius of the circles in the pseudo-ShakeMap is defined in Alg. 1:

Algorithm 1 Definition of the radius of intensity rings

```
Let I be the set of all detected intensities.
Let N_i be the set of nodes that detected vibration of intensity i.
Let center be the center of the pseudo-ShakeMap (estimation result).

for all i ∈ I do
    limit ← FINDFARTHESTNODEBYINTENSITY(i, center)
    radius ← DISTANCEKM(center, limit)
end for
function FINDFARTHESTNODEBYINTENSITY(i, center)
                        ▷ Sort N_i descending, by the distance between each node and the center
    X ← N_i.SORT(compareWhat: distanceKm(n_i ∈ N_i, center))
    return X[0]                                   ▷ The farthest node is at index 0
end function
```

To summarize, a circle of intensity i includes the farthest node from the center that detected intensity vibration i.

The pseudo-ShakeMap is helpful for graphically visualizing an earthquake's state in real-time. In multiple parts of this chapter, we use the pseudo-ShakeMap to assist explanations. Objects in a pseudo-ShakeMap have the following meaning:

- the red dot is the real earthquake epicenter;
- blue markers are seismometers;
- circles are colored based on the phase in which they are produced;
- the same color is chosen for the other dot in the map, which denotes the estimated epicenter.

6.3.3 Epicenter estimation

The proposed earthquake epicenter estimation strategy is composed of three steps executed locally by each node: candidate election, estimation refinement, and offside removal.

The **candidate election** step examines the state table to find the node that detected the highest-intensity vibration before all other nodes.

The **estimation refinement** can be made to reduce the estimation's dependency on the nodes' positioning. Refining means finding the center of gravity of a plane where nodes are objects, and their recorded intensity is their mass.

Offside removal is a further optimization that consists of moving the previously estimated point after an analysis comprising the recorded intensities and the relative positioning of nodes and intensity rings.

Candidate election

The **candidate seismometer** is defined as the node that has recorded a vibration before all other nodes, and such vibration is of the highest intensity. **Candidate election** means finding such node. As an example, if three seismometers have recorded a vibration, the maximum recorded intensity is 8, and the first detection happened at ts_0 (Tab. 6.3), N_0 is the **candidate**, since $8 > 7 \wedge ts_0 < ts_1$.

Table 6.3. Example of state table with three nodes

Node	Intensity	Timestamp
N_0	8	ts_0
N_1	8	ts_1
N_2	7	ts_2

Identifying the candidate is essential since, by definition, it is most probably the nearest node to the epicenter. As described before, since the state table's update rules and the gossip behavior make the network converge to the same state table, all nodes eventually elect the same candidate.

Each node performs a candidate election whenever a new message is sent or received. Every time the state table is updated, it is also examined to check whether a new candidate should be elected. Alg. 2 describes how candidate election is performed; at the end of the function, *candidateMsg.src* is the candidate node.

Algorithm 2 Candidate election

Let *msg* be the last sent or received message.
Let *candidateMsg* be the message that made a candidate be elected.

```
function ELECTCANDIDATE(msg)
    if ∄candidateMsg then
        candidateMsg ← msg
    else if msg.timestamp ≤ candidateMsg.timestamp∧msg.intensity ≥ candidateMsg.intensity then
        candidateMsg ← msg
    end if
end function
```

At the end of this phase, the candidate's position can already be considered an epicenter estimation; hence nodes can build a pseudo-ShakeMap (as shown in Fig. 6.2) using the candidate as the center.

Refinement

However, the candidate node is not always the best estimation, as shown in Fig. 6.2. The result can be further improved by executing the **refinement** phase, which is an optimization that produces a new point called **virtual epicenter**. Refinement consists of finding the center of gravity of the area constituted by the set of all seismometers, considering their detected intensity as mass. Alg. 3 describes how the gravity center is computed.

The principle behind the refinement is to consider for the estimation not only the candidate but all seismometers, weighting them according to their detected

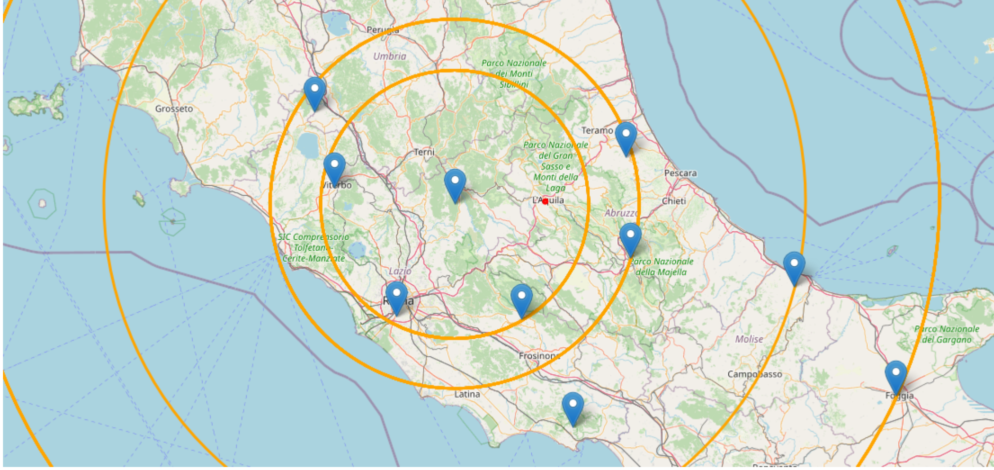


Figure 6.2. Pseudo-ShakeMap resulting after the candidate election step

Algorithm 3 Gravity center computing

Let $S = \{(node_0, intensity_0), (node_1, intensity_1), ...\}$ be the state table.

function GETCENTEROFGRAVITY
  $sum_x \leftarrow 0$
  $sum_y \leftarrow 0$
  $sum_z \leftarrow 0$
  for all $(n, i) \in S$ **do**
    $lat \leftarrow n.lat * \pi/180$
    $lon \leftarrow n.lon * \pi/180$
    $sum_x \leftarrow sum_x + cos(lat) * cos(lon) * i$
    $sum_y \leftarrow sum_y + cos(lat) * sin(lon) * i$
    $sum_z \leftarrow sum_z + sin(lat) * i$
  end for
  $avg_x \leftarrow sum_x/|S|$
  $avg_y \leftarrow sum_y/|S|$
  $avg_z \leftarrow sum_z/|S|$

  $lon \leftarrow atan2(avg_y, avg_x)$
  $hyp \leftarrow \sqrt{(avg_x * avg_x) + (avg_y * avg_y)}$
  $lat \leftarrow atan2(avg_z, hyp)$

  return $(lat * 180/\pi,\ lon * 180/\pi)$
end function

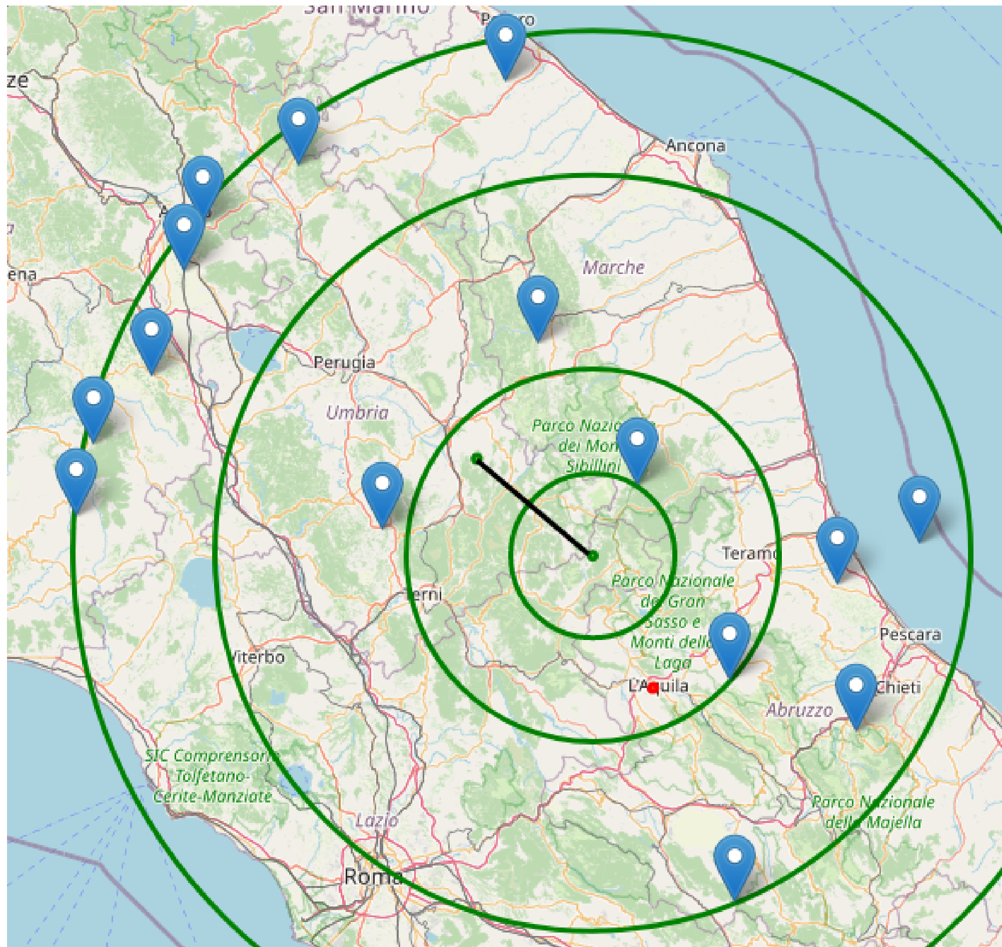


Figure 6.3. Comparison between the gravity center (also center of the pseudo-ShakeMap) and the center of shape; the black line represents the distance between the two points.

intensity, thus making the ones that detected the highest intensities (i.e., nearest to the epicenter) more relevant in the estimation.

Refinement is executed when one of the following conditions is true:

- all nodes reported the same intensity;
- given $maxIntensity$ as the highest detected intensity, at least one node reported $maxIntensity - 1$.

From the rules follows that refinement does not happen if there is a single node with the maximum intensity and the immediately lower intensity is not reported. The rationale behind this heuristic is to avoid moving away the virtual epicenter from the single max-intensity node (the candidate) when other nodes have much smaller intensity than it; Fig. 6.3 shows an example of such behavior and provides a graphical comparison.

The newly computed point is elected as the new virtual epicenter, and an improved pseudo-ShakeMap can be built (Figure 6.4).

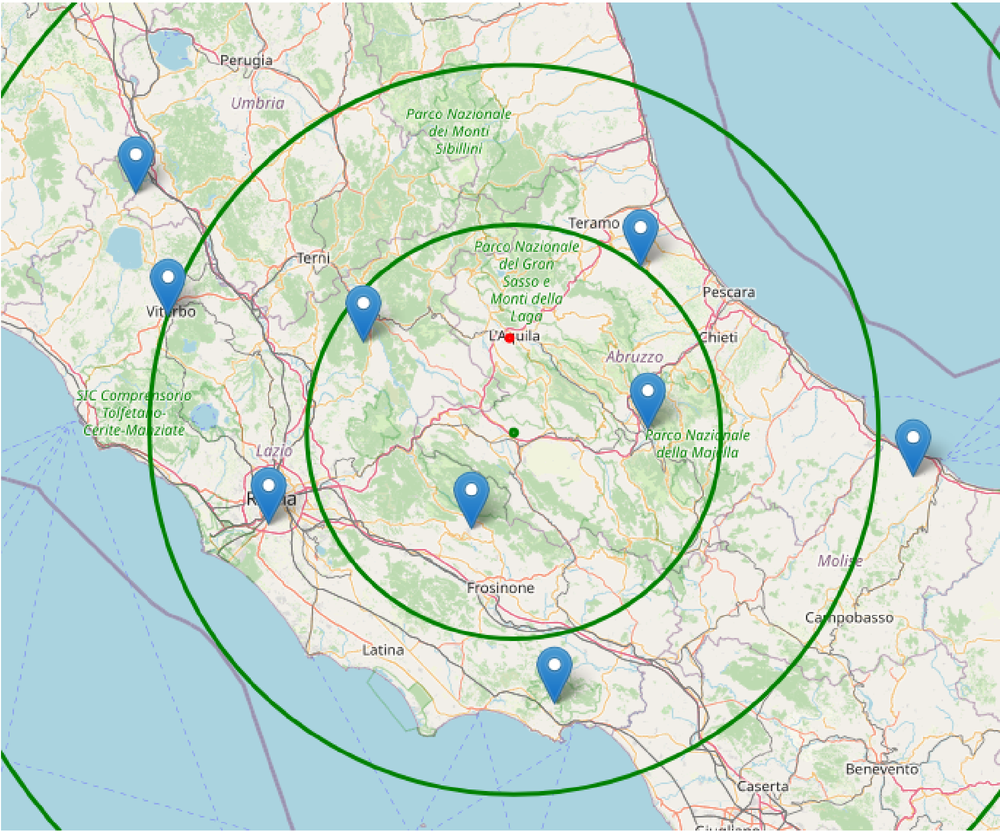


Figure 6.4. Pseudo-ShakeMap resulting after the refinement step

Offside removal

The estimation produced after the refinement can further be improved; in what follows, we describe a procedure to perform such improvement by reducing the error between the real epicenter and the estimated one after analyzing the position of some nodes and the produced intensity rings.

A node is defined as **offside** if it results in being placed in a wrong intensity ring on the pseudo-ShakeMap. More formally:

- let N_n^i be the seismometer identified with number n that detected a vibration of intensity i;
- let R_i be an intensity ring defined as the set $\{N_0^i, N_1^i, N_2^i, ...\}$ of all seismometers that detected a vibration of intensity i;
- let R_i' be the set of all seismometers that in the pseudo-ShakeMap are inscribed by the circle associated with intensity i;
- then an offside is present if $\exists i : R_i \neq R_i'$. In such case, the offside nodes are in the symmetric difference set $R_i \triangle R_i'$.

As an example, Fig. 6.5 shows two intensity circles of intensity 6 and 5, and nodes S_0 and S_9 are offside: the former for having intensity 5 but being in ring 6; the latter for having intensity 4 but being in ring 5.

The principle behind this phase is that if one or more nodes are offside, it may indicate that the virtual epicenter is not the best estimation with the available information. Exploring the problem from another point of view, an offside node suggests that the virtual epicenter is in a position where the resulting intensity rings cannot correctly partition the plane.

It must be noted that offside situations cannot be solved by rebuilding the intensity rings according to other rules. As an example of why it is true, consider the scenario in Fig. 6.6: attempting to solve the offside of node C by rebuilding new intensity rings with different criteria will ultimately result in placing B and D offside. Therefore offsides are treated with the strategy described below.

Detecting offsides An offside of node N can be detected by comparing its distance relative to the virtual epicenter to one of the nodes at the border of the next outer intensity ring; if the distance is smaller, then the node is offside. For example, in Fig. 6.6, the offside of node C can be detected because $\overline{bc} < \overline{ac}$, being $\overline{ac}$ the distance between the virtual epicenter and the border node B. The algorithm to find the first offset node is in Alg. 4.

Fixing offsides An offside can be fixed by moving the virtual epicenter away from the offside node, following a straight line, with an offset equal to the previously computed distance difference. In the example of Fig. 6.6, the offside of node C can be fixed by moving the virtual epicenter from point c to d, by an offset of $\overline{ab} = \overline{cd}$, on the segment $\overline{ad}$. The algorithm to fix an offside is in Alg. 5.

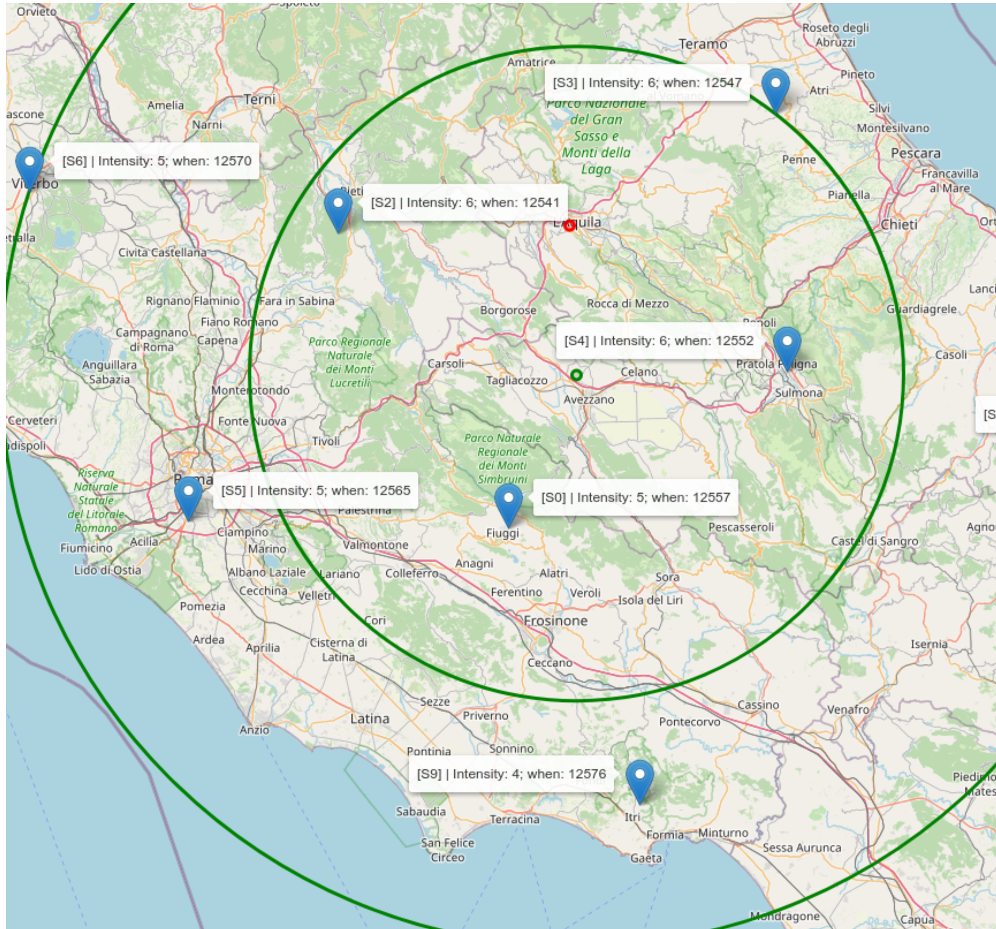


Figure 6.5. Portion of a pseudo-ShakeMap after the refinement phase, in which 2 nodes are offside

Algorithm 4 Offside detection

Let $S = \{(node_0, intensity_0), (node_1, intensity_1), ...\}$ be the state table.
Let v be the current virtual epicenter.

function DETECTOFFSIDE
  for all $(n, i) \in S$ **do**
    $nodeDist \leftarrow$ DISTANCEKM(n, v)
    $lowerBorder \leftarrow$ FINDNEARESTNODEBYINTENSITY$(i - 1, from\colon v)$
    if $\exists lowerBorder$ **then**
      $lowerBorderDist \leftarrow$ DISTANCEKM$(lowerBorder, v)$
      $offset \leftarrow (nodeDist - lowerBorderDist)/111$
      ▷ $1° \approx 111km$
      if ABS$(offset) < 0.01$ **then**
        return $\perp$
      end if
      if $nodeDist > lowerBorderDist$ **then**
        return $(n, offset)$ ▷ node n is offside of $offset°$
      end if
    end if
  end for
end function

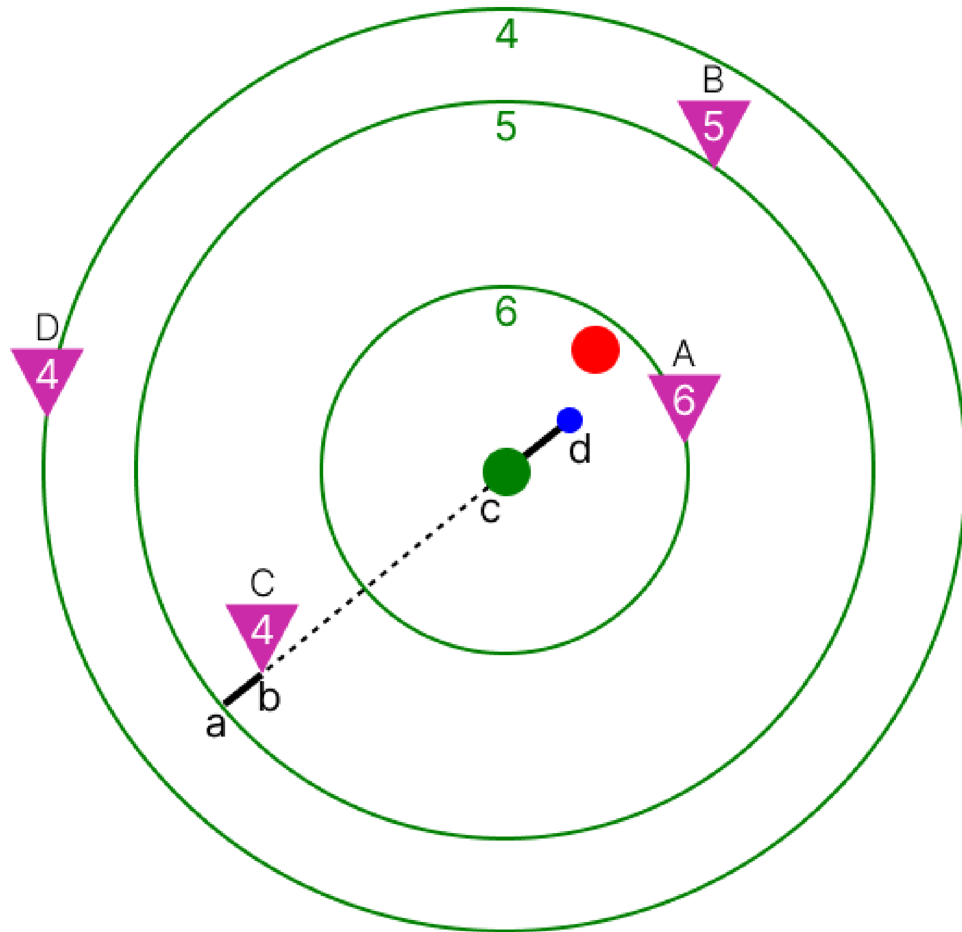


Figure 6.6. Schematic visualization of a pseudo-ShakeMap built after the refinement phase, where: (1) the green dot is the virtual epicenter; (2) the red dot is the real epicenter; (3) green circles are intensity rings, under which their intensity is reported; (4) reversed triangles are seismometers: the top label is their identifier, and the inner number is their reported intensity. In this scenario, node C is offside of the offset $\overline{ab}$.

Algorithm 5 Offside fix

Let $S = \{(node_0, intensity_0), (node_1, intensity_1), ...\}$ be the state table.
Let $v = (lat, lon)$ be the current virtual epicenter.

function FIXOFFSIDE($node, offset$)
  $angle \leftarrow atan2(node.lon - v.lon, node.lat - v.lat) * (180/\pi)$
  $v_{lat} \leftarrow v.lat + offset * cos(angle * \pi/180)$
  $v_{lon} \leftarrow v.lon + offset * sin(angle * \pi/180)$
  $v \leftarrow (v_{lat}, v_{lon})$ ▷ new virtual epicenter
end function

The described procedure can be repeated for each possible offside node until no node is offside. The approach is based on the hypothesis that after solving all offsides, the new final virtual epicenter is an improved version of the one computed in the refinement phase; the hypothesis has been confirmed, as shown in the Evaluation section.

At the end of the offside removal phase, a final and improved pseudo-ShakeMap can be built, as shown in Fig. 6.7. From Fig. 6.8, it can be seen that each phase produced a better estimation.

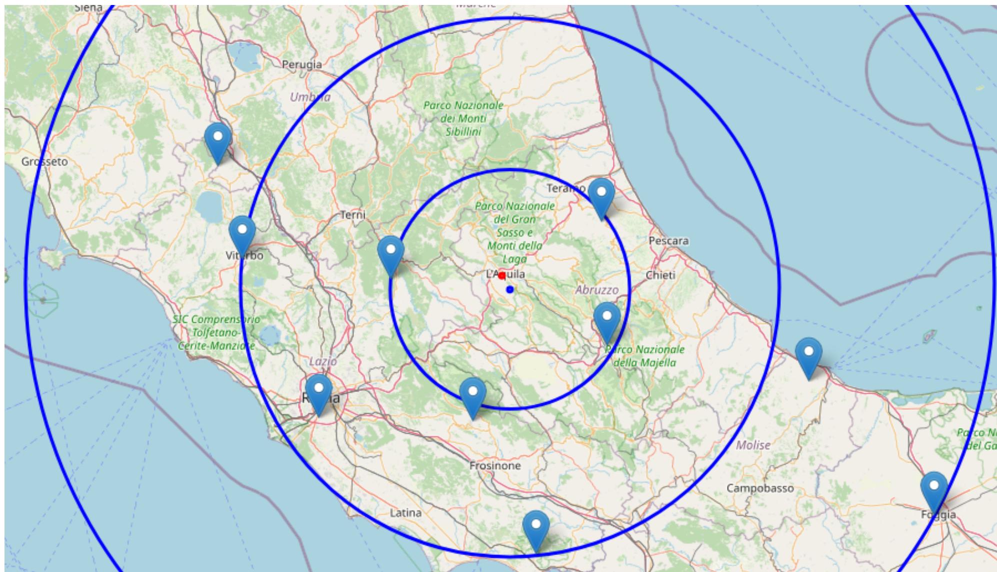


Figure 6.7. Pseudo-ShakeMap resulting after the offside removal step

6.3.4 Experiments

We built an epicenter estimation prototype that allows to:

- configure the earthquake to simulate;
- set the number of nodes, randomly positioned;
- perform evaluations by setting the initial and final number of nodes, how many nodes to add at each evaluation step, and the number of evaluation iterations (samples);

Figure 6.8. Unified view of all pseudo-ShakeMaps after executing all steps. Orange: candidate election; green: refinement; blue: offside removal.

- visualize results in real time;
- read and download evaluation statistics.

The prototype has been written in JavaScript, re-implementing a subset of the previously introduced foundation architecture and adding the new elements needed to perform the epicenter estimation tasks.

Multiple evaluation runs have been executed to assess each estimation step's performance. The leading quality indicator is the estimation error, meaning the kilometers between the estimated and the real epicenter. Another quality factor is the improvement scale, meaning each estimation phase should ideally yield a better estimation.

Setup

At the end of the evaluation, the resulting JSON file contains structured data comprising raw and aggregated information that will be used in subsequent steps.

Such data has then been analyzed with Pandas: a Python data analysis and manipulation library. Specifically, we have analyzed the following:

- error distribution for each estimation step (candidate election, refinement, offside removal);
- impact of the offside removal attempts on the error;

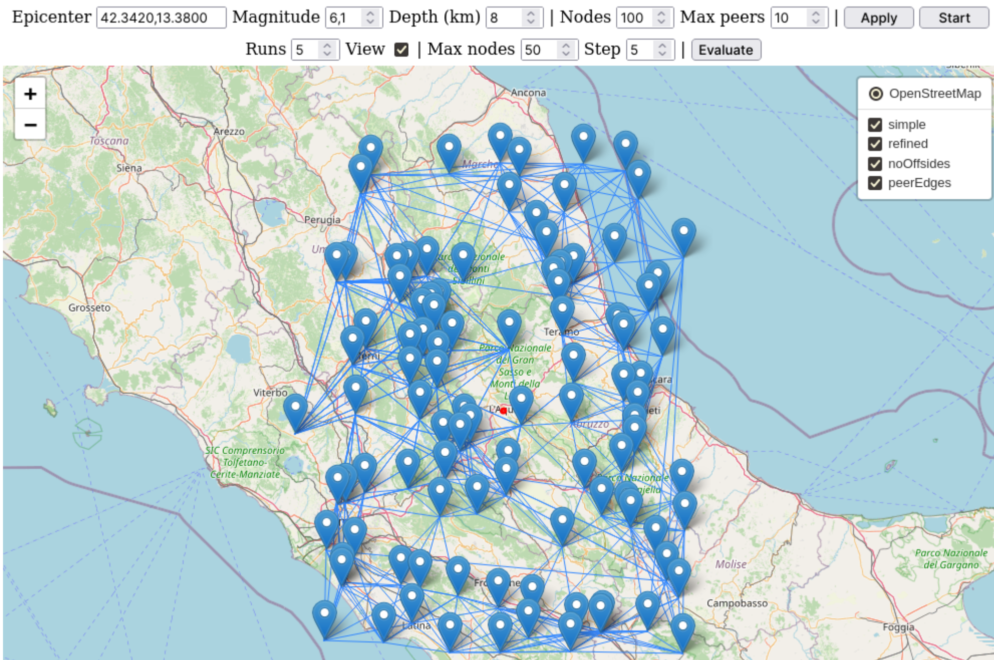


Figure 6.9. Screenshot of the prototype's web page, composed of: (1) a panel to configure various simulation parameters; (2) a Leaflet map to optionally see results in real time

- impact of the number of nodes on the quality of the estimation.

Evaluation

We have produced a Jupyter Notebook to load, transform and analyze the collected data. In the following paragraphs, we comment on each of the performed analyses.

Error distribution per strategy step The first data analysis task we performed aimed to understand how errors are distributed for each estimation step.

The plot in Figure 6.10 can be commented as follows:

- the candidate election step (orange line) has a vast error range, with the highest maximum error;
- the refinement step (green line) reduces the max error and shrinks the error in the mid-lower range;
- the offside removal step (blue line) further reduces the max error and further shrinks the error in the lower range.

To summarize, since the quality of the first step is inversely proportional to the distance between the nearest node to the epicenter and the epicenter itself, its minimum error can be as low as said distance. As hypothesized, the refinement step reduces the reliance on the positioning of the nearest node, generally reducing the maximum error. A similar principle holds for the offside removal step.

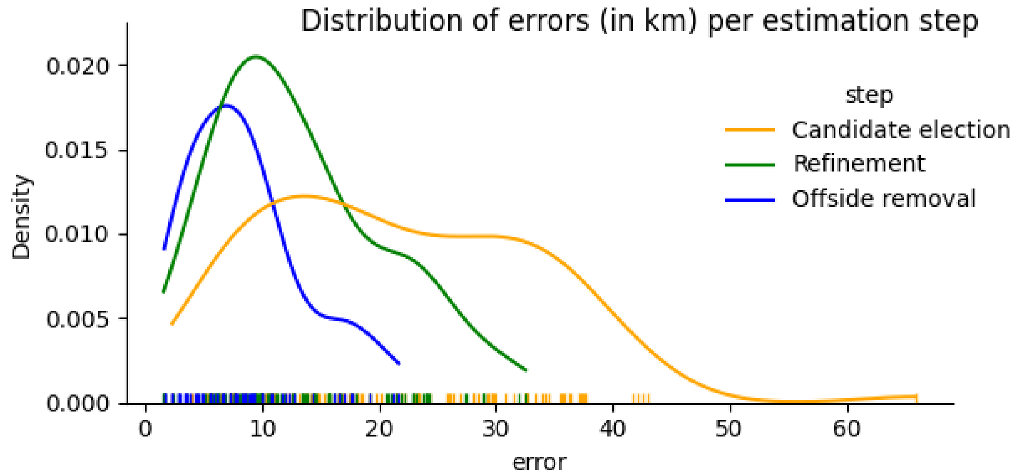


Figure 6.10. Kernel density estimation plot, showing the error on the x axis and its distribution on the y axis

Offside-attempt analysis The following evaluation is about the offside removal step. Specifically, we have analyzed how the estimation error changes at each attempt, which is executed temporally sequentially, representing a time figure.

As a preliminary evaluation, we have placed a hard limit on the number of offside removal attempts to 100 to let the algorithm finish with a purposely simple rule; a rule is needed to handle situations in which the positioning of nodes is such that offsides are numerous and minimal.

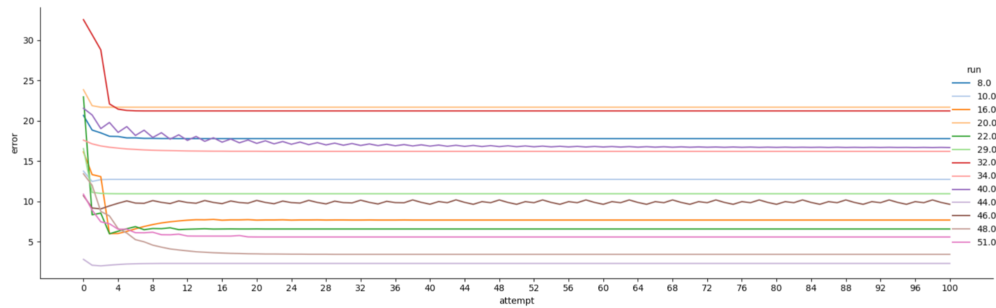


Figure 6.11. Relation plot showing—for multiple evaluation samples—the offside removal attempts on the x axis and the relative estimation error on the y axis. A lower error is better.

The plot in Fig. 6.11 can be commented as follows:

- at each new attempt, errors are typically not increasing;
- in a minority of the instances, the error increases after certain attempts;
- in other minority cases, errors fluctuate;
- generally, errors are reduced very early (i.e. within the first 4 attempts).

To summarize, offside removal produces a general improvement regarding the error.

Stop rule

In the following paragraph, we illustrate an analysis to find a better algorithm stop rule. Specifically, we have analyzed the offset value's behavior over time. As shown in Fig. 6.12:

- offsides with the highest offset value are solved very early (within the first 4 attempts);
- after solving said offsides, the offset tends to 0.

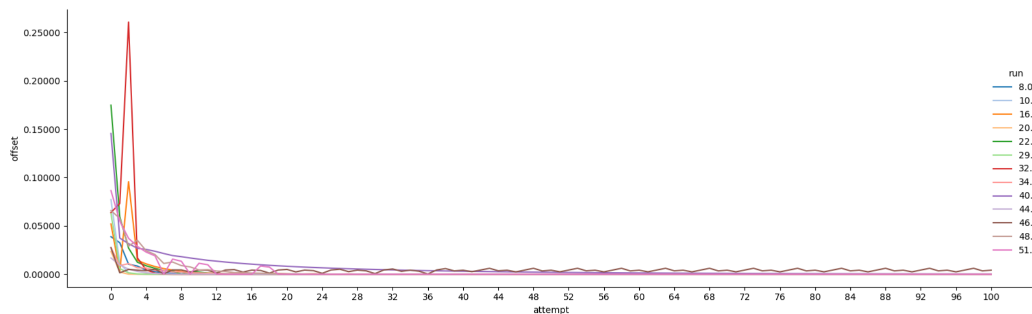


Figure 6.12. Relation plot showing—for multiple evaluation samples—the offside removal attempts on the x axis and the offset on the y axis

A notable observation is that the *elbow* of most lines is located within the early attempts, precisely between the offsets $[0.01, 0.00]$, which indicates that after solving an offside which offset is in such interval, the offsets of the subsequent offsides tends to 0, meaning that they are increasingly less important. This discovery is helpful to replace the simple hard limit and instead use an offset threshold as a stopping rule. More specifically, the offset threshold avoids fixing below-threshold offsides, thus eventually ending the procedure with a good compromise between efficiency and correctness.

A new analysis has been executed after setting the offset threshold to 0.01 (Fig. 6.13). We can observe the following:

- similarly to the plot of the previous analysis, offsides are solved early, some of which are at the first attempt;
- there is a general behavior of error reduction at each attempt;
- in the minority of instances in which there is a degradation of the result, the final result is still an improvement of the starting estimation;
- a single instance presented the need for many attempts, all producing an increasingly better estimation.

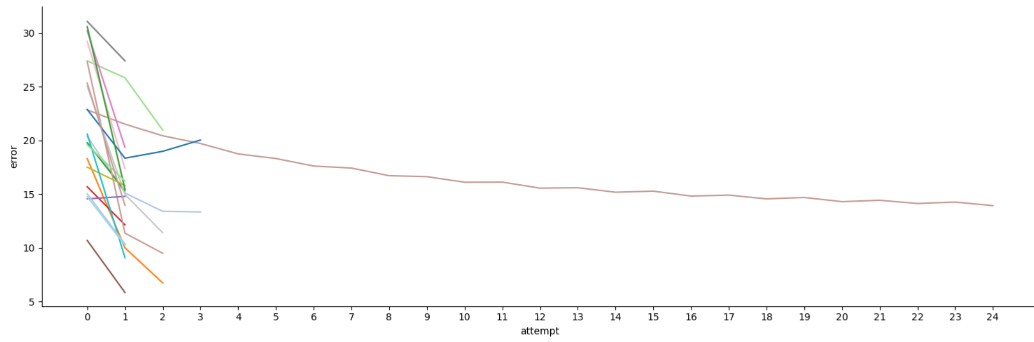


Figure 6.13. Relation plot portraying the offside removal attempts on the x axis and the estimation error on the y axis. A lower y is better.

To recap, introducing the offset threshold as a data-driven stop rule improves the efficiency of the offside removal step.

Furthermore, the distribution plot (Fig. 6.14) does not show notable differences concerning the one shown before.

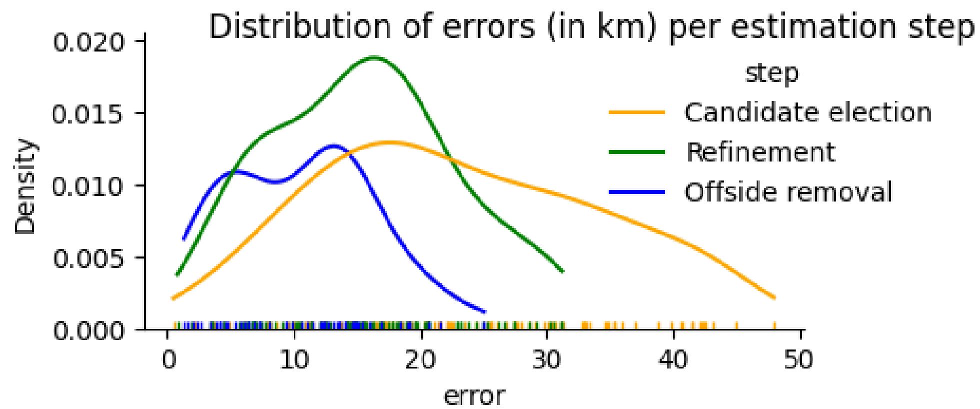


Figure 6.14. Kernel density estimation plot, showing the error on the x axis and its distribution on the y axis

Number of nodes analysis We have evaluated how the procedure's precision changes regarding the number of nodes. Specifically, the following analysis describes how the minimum, average, and maximum errors for each estimation step change according to the number of nodes.

From the plot in Fig. 6.15, it can be observed that among 100 evaluation runs, with a varying amount of nodes in the interval $[5, 200]$, with an increasing step of 5 nodes per run:

- the minimum error does not considerably change among the different steps, i.e., each step produces a comparable best estimation;
- on average, the offside removal step improves the results from the refinement step and is considerably better than the candidate election step;

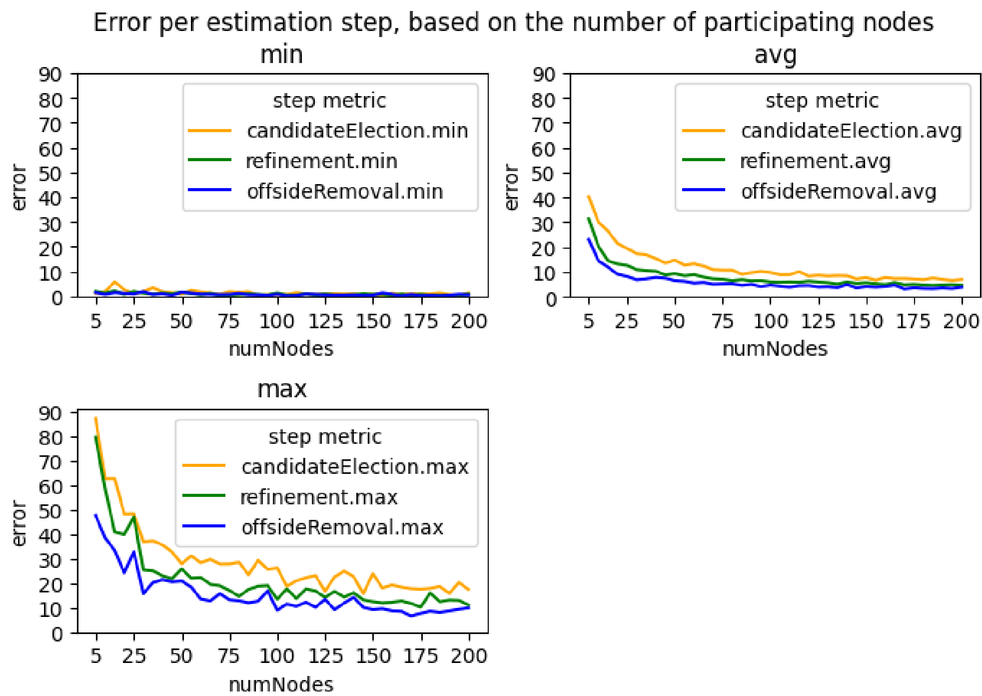


Figure 6.15. Line plots portraying the relationship between the number of nodes (x axis) and the estimation error (y axis, lower is better). Each plot shows the behavior of such variables concerning each estimation step (candidate election, refinement, offside removal).

- the maximum error is the greatest quality divider:
  - the candidate election step always produces the lowest-quality estimations;
  - the refinement step always improves the estimations from the previous step;
  - similarly, the offside removal step always improves the estimations from the refinement.

We have simulated five earthquakes in Italy using the earthquake parameters from the INGV's historical earthquakes repository:

- L'Aquila (AQ) of Apr 6, 2009;
- Finale Emilia (MO) of May 20, 2012;
- Accumoli (RI) of Aug 24, 2016;
- Norcia (PG) of Oct 30, 2016;
- Capitignano (AQ) of Jan 18, 2017.

As shown in Tab. 6.4, all simulations produced results with less than 4 km of error on average, and 10 km of error in the worst case. These parameters are compatible with an estimation of the epicenter for earthquake first response.

Table 6.4. Evaluation results of multiple earthquakes, executed with 200 nodes across 100 random samples. Values in km.

Earthquake	Min error	Avg error	Max error
L'Aquila	0.61	2.75	6.04
Finale Emilia	0.83	3.27	8.23
Accumoli	0.58	3.06	6.35
Norcia	0.66	3.59	9.69
Capitignano	0.88	3.94	9.21

6.4 Discussion

Estimating the epicenter of an earthquake is a process that requires interpolating data originating from multiple seismic stations; this process is generally executed on servers called *fusion centers*, which represent a single point of failure.

The process we propose solves the earthquake epicenter estimation problem in a decentralized way and in real-time, using a fully-decentralized architecture such as the SeismoCloud 2.0 one, with minimal changes. Specifically, the architecture is extended with elements to (1) extract relevant information from Earthquake Early Warning (EEW) messages that nodes exchange; (2) let nodes independently build a converging global state of the system; (3) analyzing said global state to estimate the epicenter.

The estimation procedure is composed of three phases that continuously improve the result. These steps are:

1. candidate election: consisting of finding the node that detected a vibration with the highest intensity before every other node;
2. refinement: improvement of the previous result by computing the center of mass of the seismometers on the plan, i.e., a new point that considers the detected intensities of all seismometers as a weight factor;
3. offside removal: optimization that improves the estimation by analyzing the relative positioning of nodes and their detected intensity.

Each estimation step improves the previous result, meaning that the entire estimation pipeline produces good estimations with an average error that, with the expected number of nodes composing the network, can be as low as 3km.

This estimation procedure and results have been published as [10] and presented at *9th International Conference on Internet of Things: Systems, Management and Security (IOTSMS 2022)* conference.

Chapter 7

Conclusions

The present work aims to comprehensively contribute to the process, design, and technologies of Earthquake Early Warning (EEW).

We dealt with topics relevant to enhancing the adoption, security, robustness, and scalability of crowdsensing-based EEW systems.

We proposed a decentralized approach to crowdsensed data processing and information exchange that leads to a more resilient networking architecture, removing Single Points of Failure, demonstrating higher efficiency, assessing and mitigating security vulnerabilities, and showing improved privacy.

We demonstrated the capabilities of the proposed architecture not only on the main EEW problem but also on the crucial aspect of estimating the epicentral area of an earthquake quickly.

In particular, this Ph.D. thesis tackled several challenges related to Earthquake Early Warning (EEW) systems and their underlying technologies. Chapter 3 presented the successful application of End User Development (EUD) to EEW systems, making it more accessible and usable and incentivizing the contribution of the crowd. Chapter 4 highlighted the importance of testing and securing MQ Telemetry Transport (MQTT) brokers and client libraries used by EEW systems to prevent potential vulnerabilities and attacks. Chapter 5 proposed a novel crowdsensing EEW architecture that leverages edge computing and decentralized processing, enhancing privacy and fault tolerance. Chapter 6 presented a fully-decentralized architecture for estimating earthquake epicenters in real time that eliminates the need for centralized fusion centers and can produce highly accurate results with low error rates.

The proposed architecture, described in Chapters 5 and 6, tackles primary issues for traditional EEW systems: resilience and privacy. While the traditional approach involves deploying high-cost, high-precision sensor networks, which can be prohibitively expensive for many communities, crowdsensing offers a potential solution by leveraging the collective power of low-cost sensors deployed by individuals or organizations. However, existing crowdsensing designs for EEW systems are based on a centralized architecture, which can be vulnerable to partial faults in communication or the fusion center itself. The proposed fully decentralized architecture addresses these vulnerabilities by removing the need for centralized fusion centers and allowing each node to detect and relay earthquake information independently. The approach is based on a topology with no strong hierarchy, enabling

fault tolerance and enhancing the scalability of Earthquake Early Warning systems. By using low-cost sensors, our proposed architecture can be deployed more widely, making EEW systems more accessible and affordable for communities worldwide.

The overall findings of this thesis contribute to advancing the field of Earthquake Early Warning and its associated technologies, highlighting the importance of usability, security, privacy, and decentralization. These findings have significant implications for researchers, developers, and stakeholders involved in designing, implementing, and improving EEW systems. Future work can enhance the proposed architectures' security and scalability and evaluate their performance in different settings and scenarios.

The research outcomes presented in this thesis have been disseminated in various scientific forums and publications, demonstrating their relevance and impact. They contribute to the ongoing efforts to develop more reliable and efficient EEW systems that can better protect communities from the devastating consequences of earthquakes.

7.1 Recommendations for Future Research

While the proposed architecture provides the information needed for earthquake Earthquake Early Warning, many other aspects must still be addressed. For example, providing End User Development in a distributed architecture poses challenges on how to retrieve data and where to run the EUD system itself.

A critical yet-to-be-addressed issue is the resilience against byzantine nodes (where the definition of byzantine might expand from a simple fault to an active attacker). This security issue is common in crowdsourcing projects, as members can build devices to enroll in the network. Even when they cannot, proving that a device exists (i.e., not simulated via software), behaves correctly, or was not altered is an open problem. Remote attestation is the most prominent solution [18]. In centralized architectures, this problem may be partially mitigated by some data pre-processing, but it is hardly a definitive protection.

Another open challenge is in the network topology algorithms. A topology based on geographical distances (like the one presented in this proposal) might be sub-optimal: gossip protocols are sensitive to latency and Round-trip time (RTT) in communication networks. Further studies on the effect of network and geographical characteristics are needed to improve the speed of the system response further.

Acronyms

AMQP Advanced Message Queuing Protocol. 14

API Application Programming Interface. 5, 7, 9

BTS Base Transceiver Station. 42

CRNN Convolutional-Recurrent Neural Network. 29, 30, 37–40, 73

DDoS Distributed Denial-of-Service. 17

DoS Denial-of-Service. 16, 18, 19, 21, 23, 25

EEW Earthquake Early Warning. iii, v, vi, 1, 2, 5, 8, 9, 13, 25, 27–32, 34–36, 38, 40–42, 45–47, 64, 67, 68, 75

EUD End User Development. v, 2, 5, 7–9, 11, 13, 67, 68

FWA Fixed-Wireless Access. 42

GPIO General Purposes Input/Output. 36–38

HMAC Hash-based Message Authentication Code. 18

HTTP Hyper-text Transfer Protocol. 7, 14

IIoT Industrial Internet-of-Things. 15

IoT Internet-of-Things. v, 2, 5–9, 13–18, 22, 23, 25, 27–32, 34, 36, 38, 40, 42, 73

IP Internet protocol. 14

JMA Japan Meteorological Agency. 27

JSON JavaScript Object Notation. 17, 18, 73

LPWAN Low Power Wide Area Network. 29, 30, 42

MAC Message Authentication Code. 18

MCS Modified Mercalli intensity scale. 47, 48

MEMS Micro-Electro-Mechanical Systems. 27, 28, 30, 45

MQTT MQ Telemetry Transport. v, 2, 3, 5, 7, 9, 10, 13–25, 38, 67, 73

PGA Peak Ground Acceleration. 47

PGV Peak Ground Velocity. 47

QoS Quality of Service. 10, 14, 17, 19–24

REST Representation of State Transfer. 9

RTT Round-trip time. 68

SPoF Single point of failure. iii, 2, 27, 28

SSL Secure-Socket Layer. 18

TLS Transport Layer Security. 10, 15, 16, 18, 23–25, 41

TSDB Time-Series Database. 5

UCD User-Centered Design. 36

VPN Virtual Private Network. 25

List of Figures

2.1 SeismoCloud network in Italy (Feb 26th, 2023). Green marks are either Internet-of-Things sensors or smartphone participating in the system. Background tiles from Google Maps - l' Google. 6

3.1 Node-RED visual editor with an action flow running. 9
3.2 Example of two new nodes created for this experiment 10
3.3 Configuration panel for the newly designed "Temperature" node. . . 10
3.4 A very simple flow in Node-RED that sends a SMS whether two sensors (“home” and “office”) detect a quake. Messages flow from left to right. 10

4.1 Schema of the testbed for the experiments: the fuzzer, which acts as a typical client, takes in input a “JSON experiment file" containing the client’s packets to the MQTT broker. The fuzzer will also receive all the PUBLISH packets sent to the broker. The MQTT Client, instead, uses one of the libraries that are examined in the subsection 4.5.2. . 18

5.1 Network architecture example: six detectors (three of which have an embedded probe) and three probes. Detectors are linked to neighbors based on their location. 32
5.2 Detector bootstrap sequence diagram. 33
5.3 Probe bootstrap sequence diagram. 34
5.4 Pipeline diagram. Each probe is attached to a buffer that feeds the detection algorithm’s dedicated instance. Probe #3 is internal, as this detector also has the probe role. 35
5.5 Prototype pipeline. The detection algorithm is the CrowdQuake CRNN, and the time window is set to 2 s. 39
5.6 CrowdQuake’s Convolutional-Recurrent Neural Network. 40
5.7 Detection latency for the ML model when multiple probes are sending data to a single detector. 41
5.8 Detection latency changes in time. The model was tested over 300 samples. Each point represents the latency for a single sample. . . . 42
5.9 Detector network used in simulations. Lines between detectors represents connections. Detectors are placed in a random pattern. Tile images by OpenStreetMap [51]. 43

5.10 Number of detectors warned since the first detection on the source node. Note that the number of detectors is sampled each 10 ms. The y axis represent the cumulative number of detectors alerted. 43

6.1 Comparison between ShakeMap and pseudo-ShakeMap for the L'Aquila (Italy) earthquake of Apr 6, 2009. 48
6.2 Pseudo-ShakeMap resulting after the candidate election step 51
6.3 Comparison between the gravity center (also center of the pseudo-ShakeMap) and the center of shape; the black line represents the distance between the two points. 52
6.4 Pseudo-ShakeMap resulting after the refinement step 53
6.5 Portion of a pseudo-ShakeMap after the refinement phase, in which 2 nodes are offside 55
6.6 Schematic visualization of a pseudo-ShakeMap built after the refinement phase, where: (1) the green dot is the virtual epicenter; (2) the red dot is the real epicenter; (3) green circles are intensity rings, under which their intensity is reported; (4) reversed triangles are seismometers: the top label is their identifier, and the inner number is their reported intensity. In this scenario, node C is offside of the offset $\overline{ab}$. 56
6.7 Pseudo-ShakeMap resulting after the offside removal step 57
6.8 Unified view of all pseudo-ShakeMaps after executing all steps. Orange: candidate election; green: refinement; blue: offside removal. . . 58
6.9 Screenshot of the prototype's web page, composed of: (1) a panel to configure various simulation parameters; (2) a Leaflet map to optionally see results in real time 59
6.10 Kernel density estimation plot, showing the error on the x axis and its distribution on the y axis 60
6.11 Relation plot showing—for multiple evaluation samples—the offside removal attempts on the x axis and the relative estimation error on the y axis. A lower error is better. 60
6.12 Relation plot showing—for multiple evaluation samples—the offside removal attempts on the x axis and the offset on the y axis 61
6.13 Relation plot portraying the offside removal attempts on the x axis and the estimation error on the y axis. A lower y is better. 62
6.14 Kernel density estimation plot, showing the error on the x axis and its distribution on the y axis 62
6.15 Line plots portraying the relationship between the number of nodes (x axis) and the estimation error (y axis, lower is better). Each plot shows the behavior of such variables concerning each estimation step (candidate election, refinement, offside removal). 63

List of Tables

4.1 Summary of test results for brokers. The tested versions were the latest stable available at the time of our experiments. 19
4.2 Summary of test results for client libraries. The tested versions were the latest stable available at the time of our experiments. 23

5.1 Earthquake Early Warning message content. This message is relayed between *detectors*. 34
5.2 Hardware specifications for prototype detectors 37
5.3 Hardware specifications for prototype probes 37
5.4 CRNN response speed for 2-second signal (200 values), averages on 300 samples 40

6.1 Structure of an EEW message 47
6.2 Structure of a node's state table 48
6.3 Example of state table with three nodes 50
6.4 Evaluation results of multiple earthquakes, executed with 200 nodes across 100 random samples. Values in km. 64

Bibliography

[1] Ferrara, Pietro and Mandal, Amit Kr and Cortesi, Agostino and Spoto, Fausto. Static analysis for discovering IoT vulnerabilities. *International Journal on Software Tools for Technology Transfer*, **23** (2021), 71. doi:10.1007/s10009-020-00592-x.

[2] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, **17** (2015), 2347. doi:10.1109/COMST.2015.2444095.

[3] Allen, R. M., Kong, Q., and Martin-Short, R. The myshake platform: a global vision for earthquake early warning. *Pure and Applied Geophysics*, **177** (2020), 1699.

[4] Anderson, D. P. Boinc: A system for public-resource computing and storage. In *Fifth IEEE/ACM international workshop on grid computing*, pp. 4–10. IEEE (2004).

[5] Araujo Rodriguez, L. G. and Macêdo Batista, D. Program-aware fuzzing for mqtt applications. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2020, p. 582586. Association for Computing Machinery, New York, NY, USA (2020). ISBN 9781450380089. Available from: https://doi.org/10.1145/3395363.3402645, doi:10.1145/3395363.3402645.

[6] Asim, Y., Azam, M. A., Ehatisham-ul Haq, M., Naeem, U., and Khalid, A. Context-aware human activity recognition (cahar) in-the-wild using smartphone accelerometer. *IEEE Sensors Journal*, **20** (2020), 4361.

[7] Banks, A. and Gupta, R. Mqtt version 3.1. 1. *OASIS standard*, **29** (2014), 89.

[8] Banks, A. and Gupta, R. MQTT version 3.1.1 plus errata 01 (2014). http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[9] Bassetti, E., Ottavianelli, E., and Panizzi, E. Simplify node-red for end user development in seismocloud. In *Proceedings of the 1st International Workshop on Empowering People in Dealing with Internet of Things Ecosystems*, p. 4 (2020).

[10] Bassetti, E., Quaranta, D., and Panizzi, E. Quick decentralized estimation of earthquake epicenter with low-cost iot network. In *2022 9th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pp. 1–8. IEEE (2022).

[11] Boccadoro, P., Montaruli, B., and Grieco, L. A. Quakesense, a lora-compliant earthquake monitoring open system. In *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pp. 1–8. IEEE (2019).

[12] Casteur, G., Aubaret, A., Blondeau, B., Clouet, V., Quemat, A., Pical, V., and Zitouni, R. Fuzzing attacks for vulnerability discovery within MQTT protocol. In *16th International Wireless Communications and Mobile Computing Conference, IWCMC 2020, Limassol, Cyprus, June 15-19, 2020*, pp. 420–425. IEEE (2020). Available from: `https://doi.org/10.1109/IWCMC48107.2020.9148320`, `doi:10.1109/IWCMC48107.2020.9148320`.

[13] Chaczko, Z. and Braun, R. Learning data engineering: Creating iot apps using the node-red and the rpi technologies. In *2017 16th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pp. 1–8. IEEE, IEEE, Piscataway, NJ (2017).

[14] Chanthakit, S. and Rattanapoka, C. Mqtt based air quality monitoring system using node mcu and node-red. In *2018 Seventh ICT International Student Project Conference (ICT-ISPC)*, pp. 1–5. IEEE, IEEE, Piscataway, NJ (2018).

[15] Cochran, E. S., Lawrence, J. F., Christensen, C., and Jakka, R. S. The quake-catcher network: Citizen science expanding seismic horizons. *Seismological Research Letters*, **80** (2009), 26.

[16] Collina, M., Bartolucci, M., Vanelli-Coralli, A., and Corazza, G. E. Internet of things application layer protocol analysis over error and delay prone links. In *2014 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, pp. 398–404 (2014). `doi:10.1109/ASMS-SPSC.2014.6934573`.

[17] Colombo, P. and Ferrari, E. Access Control Enforcement within MQTT-Based Internet of Things Ecosystems. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, SACMAT '18, p. 223234. Association for Computing Machinery, New York, NY, USA (2018). ISBN 9781450356664. Available from: `https://doi.org/10.1145/3205977.3205986`, `doi:10.1145/3205977.3205986`.

[18] Conti, M., Dushku, E., and Mancini, L. V. Radis: Remote attestation of distributed iot services. In *2019 Sixth International Conference on Software Defined Systems (SDS)*, pp. 25–32. IEEE (2019).

[19] Crisnapati, P. N., Wulaning, P. D., Hendrawan, I. N. R., and Bandanagara, A. A. K. B. Earthquake damage intensity scaling system based on raspberry pi and arduino uno. In *2018 6th International Conference on Cyber and IT Service Management (CITSM)*, pp. 1–4 (2018). `doi:10.1109/CITSM.2018.8674321`.

[20] Cuomo, F., Campo, M., Bassetti, E., Cartella, L., Sole, F., and Bianchi, G. Adaptive mitigation of the air-time pressure in lora multi-gateway architectures. In *European Wireless 2018; 24th European Wireless Conference*, pp. 1–6. VDE (2018).

[21] Cuomo, F., Campo, M., Caponi, A., Bianchi, G., Rossini, G., and Pisani, P. Explora: Extending the performance of lora by suitable spreading factor allocations. In *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 1–8. IEEE (2017).

[22] Dave Locke, IBM UK Labs. MQTT Past, Present and Future: 20 Years of MQTT (2019). `https://info.thingstream.io/hubfs/IBMWatsonMQTT, MQTT-SNpresentation.pdf`, last accessed: 28/02/2021.

[23] Di Paolo, E., Bassetti, E., and Spognardi, A. Security assessment of common open source mqtt brokers and clients. In *Proceedings of the Italian Conference on Cybersecurity (ITASEC 2021)*, pp. 475–487 (2021).

[24] Dierks, T. and Allen, C. The TLS Protocol Version 1.0. RFC 2246, RFC Editor (1999). Available from: `https://www.rfc-editor.org/rfc/rfc2246.txt`.

[25] Dierks, T., Allen, C., et al. The tls protocol version 1.0 (1999).

[26] Dinculean, Dan and Cheng, Xiaochun. Vulnerabilities and Limitations of MQTT Protocol Used between IoT Devices. *Applied Sciences*, **9** (2019). Available from: `https://www.mdpi.com/2076-3417/9/5/848`, `doi:10.3390/app9050848`.

[27] Faulkner, M., Liu, A. H., and Krause, A. A fresh perspective: Learning to sparsify for detection in massive noisy sensor networks. In *Proceedings of the 12th international conference on Information processing in sensor networks*, pp. 7–18 (2013).

[28] Finazzi, F. The earthquake network project: Toward a crowdsourced smartphone-based earthquake early warning system. *Bulletin of the Seismological Society of America*, **106** (2016), 1088.

[29] Fischer, G. Meta-design: Expanding boundaries and redistributing control in design. In *Proceedings of the 11th IFIP TC 13 International Conference on Human-Computer Interaction*, INTERACT'07, p. 193206. Springer-Verlag, Berlin, Heidelberg (2007). ISBN 354074794X.

[30] Fischer, J., Kühnlenz, F., Ahrens, K., and Eveslage, I. Model-based development of self-organizing earthquake early warning systems. *Simul. Notes Eur.*, **19** (2009), 9.

[31] Francillon, A., Nguyen, Q., Rasmussen, K. B., and Tsudik, G. A minimalist approach to remote attestation. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, DATE '14, pp. 1–6. Leuven, BEL (2014). ISBN 9783981537024.

[32] Ghiani, G., Manca, M., Paternò, F., and Santoro, C. Personalization of context-dependent applications through trigger-action rules. *ACM Transactions on Computer-Human Interaction (TOCHI)*, **24** (2017), 1.

[33] Hernández Ramos, S., Villalba, M. T., and Lacuesta, R. MQTT Security: A novel fuzzing approach. *Wireless Communications and Mobile Computing*, **2018** (2018). doi:10.1155/2018/8261746.

[34] Hofer-Schmitz, K. and Stojanovi, B. Towards formal methods of iot application layer protocols. In *2019 12th CMI Conference on Cybersecurity and Privacy (CMI)*, pp. 1–6 (2019). doi:10.1109/CMI48017.2019.8962139.

[35] Hoshiba, M., Kamigaichi, O., Saito, M., Tsukada, S., and Hamada, N. Earthquake early warning starts nationwide in japan. *EOS, Transactions American geophysical union*, **89** (2008), 73.

[36] Houimli, M., Kahloul, L., and Benaoun, S. Formal specification, verification and evaluation of the mqtt protocol in the internet of things. In *2017 International Conference on Mathematics and Information Technology (ICMIT)*, pp. 214–221 (2017). doi:10.1109/MATHIT.2017.8259720.

[37] Huang, X., Lee, J., Kwon, Y.-W., and Lee, C.-H. Crowdquake: A networked system of low-cost sensors for earthquake detection via deep learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3261–3271 (2020).

[38] ISO. ISO/IEC 20922:2016 Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1 (2016). Available from: https://www.iso.org/standard/69466.html.

[39] Kant, D., Johannsen, A., and Creutzburg, R. Analysis of iot security risks based on the exposure of the mqtt protocol. Tech. rep., Technische Hochschule Brandenburg (2021).

[40] Kong, Q., Allen, R. M., Schreier, L., and Kwon, Y.-W. Myshake: A smartphone seismic network for earthquake early warning and beyond. *Science advances*, **2** (2016), e1501055.

[41] Kumar, A. and Sharma, A. Internet of life (iol). In *International Conference of Advance Research and Innovation (ICARI-2015)* (2015).

[42] Lee, J., Khan, I., Choi, S., and Kwon, Y.-W. A smart iot device for detecting and responding to earthquakes. *Electronics*, **8** (2019), 1546.

[43] Lekić, M. and Gardašević, G. Iot sensor integration to node-red platform. In *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pp. 1–5. IEEE, IEEE, Piscataway, NJ (2018).

[44] Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., and Zhao, W. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal*, **4** (2017), 1125. doi:10.1109/JIOT.2017.2683200.

[45] Lomnitz, C. How Not to Locate Earthquake Epicenters. *Seismological Research Letters*, **72** (2001), 477. Available from: https://doi.

org/10.1785/gssrl.72.4.477, arXiv:https://pubs.geoscienceworld.org/ssa/srl/article-pdf/72/4/477/2755109/srl072004_0477.pdf, doi:10.1785/gssrl.72.4.477.

[46] Luzuriaga, J. E., Perez, M., Boronat, P., Cano, J. C., Calafate, C., and Manzoni, P. A comparative evaluation of amqp and mqtt protocols over unstable and mobile networks. In *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pp. 931–936 (2015). doi:10.1109/CCNC.2015.7158101.

[47] CVE-2015-4458. Available from MITRE, CVE-ID CVE-2015-4458. (2015). Available from: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-4458.

[48] Morrison, J. P. *Flow-Based Programming: A new approach to application development.* CreateSpace, South Carolina, US (2010).

[49] MQTT.org. Who invented MQTT (2021). https://mqtt.org/faq/, last accessed: 28/02/2021.

[50] Naik, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE International Systems Engineering Symposium (ISSE)*, pp. 1–7 (2017). doi:10.1109/SysEng.2017.8088251.

[51] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org (2017).

[52] Ovadia, S. Automate the internet with if this then that(ifttt). *Behavioral & social sciences librarian*, **33** (2014), 208.

[53] Panizzi, E. The seismocloud app: Your smartphone as a seismometer. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pp. 336–337. ACM, 1601 Broadway, Times Square, New York City (2016).

[54] Parolai, S., Moratto, L., Bertoni, M., Scaini, C., and Rebez, A. Could a Decentralized Onsite Earthquake Early Warning System Help in Mitigating Seismic Risk in Northeastern Italy? The Case of the 1976 Ms 6.5 Friuli Earthquake. *Seismological Research Letters*, **91** (2020), 3323. Available from: https://doi.org/10.1785/0220200177, arXiv:https://pubs.geoscienceworld.org/ssa/srl/article-pdf/91/6/3323/5176896/srl-2020177.1.pdf, doi:10.1785/0220200177.

[55] Parolai, S., et al. Assessing earthquake early warning using sparse networks in developing countries: Case study of the kyrgyz republic. *Frontiers in Earth Science*, **5** (2017). Available from: https://www.frontiersin.org/articles/10.3389/feart.2017.00074, doi:10.3389/feart.2017.00074.

[56] PUJOL, J. Earthquake Location Tutorial: Graphical Approach and Approximate Epicentral Location Techniques. *Seismological Research Letters*, **75** (2004), 63. Available from: `https://doi.org/10.1785/gssrl.75.1.63`, `arXiv:https://pubs.geoscienceworld.org/ssa/srl/article-pdf/75/1/63/2757401/srl075001_0063.pdf`, `doi:10.1785/gssrl.75.1.63`.

[57] PUJOL, J. AND SMALLEY JR, R. A preliminary earthquake location method based on a hyperbolic approximation to travel times. *Bulletin of the Seismological Society of America*, **80** (1990), 1629.

[58] RAJALAKSHMI, A. AND SHAHNASSER, H. Internet of things using node-red and alexa. In *2017 17th International Symposium on Communications and Information Technologies (ISCIT)*, pp. 1–4. IEEE, IEEE, Piscataway, NJ (2017).

[59] RAZZAQUE, M. A., MILOJEVIC-JEVRIC, M., PALADE, A., AND CLARKE, S. Middleware for internet of things: A survey. *IEEE Internet of Things Journal*, **3** (2016), 70. `doi:10.1109/JIOT.2015.2498900`.

[60] SOCHOR, H., FERRAROTTI, F., AND RAMLER, R. Automated security test generation for mqtt using attack patterns. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ARES '20, pp. 1–9. Association for Computing Machinery, New York, NY, USA (2020). ISBN 9781450388337. Available from: `https://doi.org/10.1145/3407023.3407078`, `doi:10.1145/3407023.3407078`.

[61] TABAA, M., CHOURI, B., SAADAOUI, S., AND ALAMI, K. Industrial communication based on modbus and node-red. *Procedia computer science*, **130** (2018), 583.

[62] TANABE, K., TANABE, Y., AND HAGIYA, M. Model-based testing for mqtt applications. In *Knowledge-Based Software Engineering: 2020* (edited by M. Virvou, H. Nakagawa, and L. C. Jain), pp. 47–59. Springer International Publishing, Cham (2020). ISBN 978-3-030-53949-8.

[63] TECHNOLOGY, I. E. Node-red (2020). `https://nodered.org`.

[64] TSITSIKLIS, J. Decentralized detection-advances in syatistical signal processing (1990).

[65] UR, B., MCMANUS, E., PAK YONG HO, M., AND LITTMAN, M. L. Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 803–812 (2014).

[66] VACCARI, I., AIELLO, M., AND CAMBIASO, E. Slowite, a novel denial of service attack affecting mqtt. *Sensors (Basel, Switzerland)*, **20** (2020).

[67] WALD, D. J., QUITORIANO, V., HEATON, T. H., AND KANAMORI, H. Relationships between peak ground acceleration, peak ground velocity, and modified mercalli intensity in california. *Earthquake Spectra*, **15** (1999), 557. Available from: `https://doi.org/10.1193/1.1586058`, `arXiv:https://doi.org/10.1193/1.1586058`, `doi:10.1193/1.1586058`.

[68] Wu, A., Lee, J., Khan, I., and Kwon, Y.-W. Crowdquake+: Data-driven earthquake early warning via iot and deep learning. In *2021 IEEE International Conference on Big Data (Big Data)*, pp. 2068–2075. IEEE (2021).

[69] Zhang, L., Pathak, P. H., Wu, M., Zhao, Y., and Mohapatra, P. Accelword: Energy efficient hotword detection through accelerometer. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 301–315 (2015).

[70] Zhang, L., Pathak, P. H., Wu, M., Zhao, Y., and Mohapatra, P. Accelword: Energy efficient hotword detection through accelerometer. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, p. 301315. Association for Computing Machinery, New York, NY, USA (2015). ISBN 9781450334945. Available from: `https://doi.org/10.1145/2742647.2742658`, `doi:10.1145/2742647.2742658`.

List of publications

Here is the list of all publications we published in literature during my Ph.D.:

[1] Bassetti, E., Berti, A., Bisante, A., Magnante, A., and Panizzi, E. Exploiting user behavior to predict parking availability through machine learning. *Smart Cities*, **5** (2022), 1243.

[2] Bassetti, E., Conti, A., Panizzi, E., and Tolomei, G. Iside: Proactively assist university students at risk of dropout. In *2022 IEEE International Conference on Big Data (Big Data)*, pp. 1776–1783 (2022).

[3] Bassetti, E., Luciani, A., and Panizzi, E. Ml classification of car parking with implicit interaction on the driver's smartphone. In *Human-Computer Interaction–INTERACT 2021: 18th IFIP TC 13 International Conference, Bari, Italy, August 30–September 3, 2021, Proceedings, Part III*, pp. 291–299. Springer (2021).

[4] Bassetti, E., Luciani, A., and Panizzi, E. Ml-based re-orientation of smartphone-collected car motion data. *Procedia Computer Science*, **198** (2022), 237.

[5] Bassetti, E., Luciani, A., and Panizzi, E. Re-orienting smartphone-collected car motion data using least-squares estimation and machine learning. *Sensors*, **22** (2022), 1606.

[6] Bassetti, E., Ottavianelli, E., and Panizzi, E. Simplify node-red for end user development in seismocloud. In *Proceedings of the 1st International Workshop on Empowering People in Dealing with Internet of Things Ecosystems*, p. 4 (2020).

[7] Bassetti, E. and Panizzi, E. Earthquake detection at the edge: Iot crowdsensing network. *Information*, **13** (2022), 195.

[8] Bassetti, E., Quaranta, D., and Panizzi, E. Quick decentralized estimation of earthquake epicenter with low-cost iot network. In *Proceedings of the 9th International Conference on Internet of Things: Systems, Management and Security (IOTSMS 2022)*, pp. 1–8 (2022).

[9] Di Paolo, E., Bassetti, E., and Spognardi, A. Security assessment of common open source mqtt brokers and clients. In *Proceedings of the Italian Conference on Cybersecurity (ITASEC 2021)*, pp. 475–487 (2021).