

6G SNS



Co-funded by the
European Union



***SEamless integratiON of efficient 6G WirelesS
tEchnologies for Communication and Sensing***

D4.1 Initial SoTA and design of Wireless Edge Caching solutions

January 2025

6G-SENSES project has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement 101139282

Project Start Date: 2024-01-01

Duration: 30 months

Call: HORIZON-JU-SNS-2023

Date of delivery: 2025-01-15

Topic: HORIZON-JU-SNS-2023-STREAM-B-01-02

Version: 1.0

Co-Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission (granting authority). Neither the European Union nor the granting authority can be held responsible for them.

Dissemination Level: Public (PU)

Grant Agreement Number:	101139282
Project Name:	SEamless integratioN of efficient 6G WirelesS tEchnologies for Communication and Sensing
Project Acronym:	6G-SENSES
Document Number:	D4.1
Document Title:	Initial SoTA and design of Wireless Edge Caching solutions
Version:	1.0
Delivery Date:	2024-12-31 (2025-01-15)
Responsible:	Sapienza, University of Rome (UNIROMA1)
Editor(s):	Federico Trombetti (UNIROMA1)
Authors:	Federico Trombetti, Salvatore Pontarelli, Novella Bartolini (UNIROMA1), Markos Anastasopoulos, Anna Tzanakaki (IASA), Valerio Frascolla, Jessica Sanson, Yazhou Zhu (INT), Luis Díez, Ramón Agüero (UC), Germán Castellanos (ACC), Shahid Mumtaz (NTU), Jesús Gutiérrez (IHP).
Keywords:	6G, Sensing, Wireless Edge Caching
Status:	Draft
Dissemination Level	Public (PU)
Project URL:	https://www.6g-senses.eu

Revision History

Rev. N	Description	Author	Date
0.1	First draft layout	Federico Trombetti (UNIROMA1)	2024-09-13
0.2	Updated Table of Contents, major revision before share	Federico Trombetti (UNIROMA1)	2024-10-10
0.2.1	Fixed wrong section numbering	Federico Trombetti (UNIROMA1)	2024-10-11
0.2.2	Revised Introduction	Salvatore Pontarelli (UNIROMA1)	2024-10-18
0.3	Final Update Before Reviews New Images, Revised SoTA	Federico Trombetti (UNIROMA1)	2024-11-13
0.3.1	Chapter 1 Review	Jessica Sanson (INT)	2024-11-15
0.4	Chapter 4 and 5 update. Major revision	Federico Trombetti (UNIROMA1)	2024-12-16
0.5	Added summary and conclusion. Revised Chapters 4 and 5.	Novella Bartolini, Salvatore Pontarelli (UNIROMA1)	2024-12-18
0.6	Finalized section 4.3 Figures and tables linking	Federico Trombetti (UNIROMA1)	2024-12-20
0.6.1	Chapter 2 and 3 review	German Castellanos (ACC)	2024-12-20
0.6.2	Chapter 4 and 5 review	Luis Díez, Ramón Agüero (UC)	2024-12-26
0.6.3	Reviews on chapters 2, 4, 5 merged	Federico Trombetti (UNIROMA1)	2024-12-28
0.6.4	Section 2.1.7 addition Major review	Anna Tzanakaki (IASA), Jesús Gutiérrez (IHP)	2024-12-30
0.7	Chapter 3 review merged Top-to-bottom review	Federico Trombetti (UNIROMA1)	2024-12-30
0.8	Full revision of the document	Shahid Mumtaz (NTU), Anna Tzanakaki (IASA), Valerio Frascolla (INT)	2025-01-13
0.9	Full revision of the document	Federico Trombetti (UNIROMA1)	2025-01-15
1.0	Last revision and submission of the document	Jesús Gutiérrez (IHP)	2025-01-15

Table of Contents

LIST OF FIGURES	5
LIST OF TABLES.....	6
EXECUTIVE SUMMARY	7
1 INTRODUCTION.....	8
1.1 Organization of the document	10
2 STATE-OF-THE-ART IN WIRELESS EDGE CACHING	11
2.1 Wireless Enabling Technologies	11
2.1.1 Wireless Edge Caching Infrastructure.....	12
2.1.2 Heterogeneous Network and Coded Caching.....	12
2.1.3 Cooperative Caching in Cloud-Assisted Wireless Networks	13
2.1.4 Wireless Device-to-Device Caching Networks	14
2.1.5 Mobility-Aware Caching in Cellular Networks	15
2.1.6 Information-Centric Networking	16
2.1.7 Federated Learning	17
3 CACHING USE-CASE.....	20
4 CACHING MODEL	24
4.1 Description of the simulation framework.....	24
4.1.1 Robot Operating System (ROS)	24
4.1.2 Gazebo	26
4.1.3 Methodology	27
4.2 Structure of the Simulation	27
4.2.1 Simulation Manager.....	28
4.2.2 Base Station Controller	29
4.2.3 Cache Device Controller.....	30
4.2.4 Cache Manager	31
4.2.5 Sensor Device Controller	32
4.2.6 Movement Coordinator	33
4.3 Performance Evaluation and Testing.....	34
4.3.1 Varying number of sensors	34
4.3.2 Varying cache size	36
5 SUMMARY AND CONCLUSIONS.....	38
6 REFERENCES.....	39
7 ACRONYMS.....	41

List of Figures

Figure 1-1 6G-SENSES Architecture	9
Figure 2-1 Example of a Wireless Edge Caching infrastructure.....	12
Figure 2-2 Coded Caching architecture in a heterogeneous network.	13
Figure 2-3 Cooperative Caching architecture.....	14
Figure 2-4 Device-to-device data sharing.....	15
Figure 2-5 MEC application in execution.....	16
Figure 2-6 "Thin Waist" design of the modern network infrastructure.....	17
Figure 2-7 Indicative Architecture of a FL System with a Server.....	18
Figure 2-8 Indicative Architecture of Vertical FL with the participation of two major organizations.....	19
Figure 3-1 Application case with UAVs acting as mobile caching devices.....	20
Figure 3-2 Example on how RIS can be used to enhance the communication channels.	21
Figure 4-1 Example of a ROS topic	25
Figure 4-2 Example of a ROS service	25
Figure 4-3 Example of a ROS action.....	26
Figure 4-4 Running instance of Gazebo modelling the use-case of Chapter 3.....	27
Figure 4-5 Structure of the Simulation	28
Figure 4-6 Comparison of caching strategies, distribution of miss rate by cache type, varying number of sensors.	35
Figure 4-7 Comparison of caching strategies, cumulative E2E delay by cache type, varying number of sensors.	35
Figure 4-8 Comparison of caching strategies, average age of queried data by cache type, varying number of sensors.	35
Figure 4-9 Comparison of caching strategies, distribution of miss rate by cache type, varying cache size....	36
Figure 4-10 Comparison of caching strategies, cumulative E2E delay by cache type, varying cache size.	36
Figure 4-11 Comparison of caching strategies, average age of queried data by cache type, varying cache size.	37

List of Tables

Table 3-1 In-network, Location Aware Caching Service for Disruption Minimization	20
Table 3-2 Sensing with Wi-Fi system	22
Table 4-1 Simulation Manager	28
Table 4-2 Base Station Controller	29
Table 4-3 Cache Device Controller	30
Table 4-4 Cache Manager	31
Table 4-5 Sensor Device Controller	32
Table 4-6 Movement Coordinator	33

Executive Summary

This document corresponds to deliverable D4.1 “Report on 6G-SENSES, Technologies for Wireless Edge Caching, Context and Architecture Design” of the 6G-SENSES project. The deliverable provides an in-depth survey of the state of the art related to Wireless Edge Caching technologies, providing a wide-ranging overview of the different layers involved in the development of this technology and its integration into contemporary network infrastructures. The report focuses on challenging network scenarios in which timely and updated sensing information is required or when operating in highly dynamic, mobile-driven networks involving unmanned aerial vehicles (UAVs) or other mobile devices. In these cases, the limitations of conventional caching strategies become more pronounced, and specific solutions should be developed. As a first step in the analysis of the best Wireless Edge Caching solutions we first developed a thorough simulation framework that includes both network communication between devices and sensors. This simulation framework has been exploited to implement and compare multiple traditional caching strategies such as First-In First-Out (FIFO), Least Recently Used (LRU), and Least Frequently Used (LFU). The simulator has been used to compare the performances of all the caching strategies, under different key performance metrics, such as HIT/MISS ratios, end-to-end (E2E) delay, and data freshness. In addition, we plan to design algorithms that also exploit location related information, to develop ad-hoc location-based caching mechanism and evaluate how these approaches improve the caching performance with respect to standard caching mechanisms.

A more precise and definitive detailed architectural description of the Wireless Edge Caching mechanism will be provided in deliverable D4.2.

1 Introduction

The global demand for a robust network infrastructure is growing at an unprecedented rate. The proliferation of streaming services, the expansion of Internet of Things (IoT) applications, and the widespread adoption of smart devices are pushing the limits of traditional cloud-based systems. Consequently, conventional centralized architectures are increasingly unable to meet the escalating demands for speed, capacity, and low latency performance required by today's digital landscape. The advent of 5G and the imminent arrival of 6G technologies, are fundamentally reshaping the design and implementation of network architectures. 6G is witnessing paradigm changes, such as the elimination of the traditional "cell-like" deployment of mobile communications infrastructure, enabled by the introduction of cell-free massive multiple-input multiple-output (CF-mMIMO). This innovation is a significant technical advancement aimed at enhancing the performance of wireless networks. In a CF-mMIMO system, when all access points (APs) deployed in a system collaborate to serve a fixed number of users, as the number of APs increases, the cell boundary effect of traditional cells can be eliminated. As a result, a "cell-free" system is formed, and the capacity and performance of the communication system can be significantly improved.

Despite the power of the centralized cloud model, its use is becoming increasingly inadequate for efficiently handling the vast volumes of data traffic and the real-time processing demands of modern applications. The Content Delivery Networks (CDNs) [1] paradigm has arisen in order to accommodate the delivery of large bandwidth of data for content streaming, but CDNs are not universally applicable to all types of applications. While CDNs are particularly efficient when local storage and data rates are the bottlenecks of the infrastructure, they fall short to address all the needs of more complex applications.

To address these limitations, Wireless Edge Caching has gained significant attention and has expanded in both scope and effectiveness. Wireless Edge Caching [2], [3] decentralizes storage and processing, moving data closer to the end user by shifting from traditional centralized data centres to lower-tier infrastructures, such as base stations (BSs), cells, and user devices. This shift enables quicker access to content and reduced dependency on remote cloud servers, minimizing E2E latency and improving the overall perceived experience of the final user.

Several Wireless Edge Caching solutions have been proposed to increase the network quality in terms of throughput and latency reduction. However, while it is relatively easy to enable edge caching mechanisms in traditional scenarios, it is hard to generalize its use in a more general and complex scenario such as the one envisioned by 6G-SENSES. Specifically, when timely and updated sensing information is required, or when operating in highly dynamic, mobile-driven networks involving unmanned aerial vehicles (UAVs) or other mobile devices; in such cases, the limitations of conventional caching strategies become more pronounced. Therefore 6G-SENSES will develop ad-hoc solutions for Wireless Edge Caching that consider the above-mentioned constraints.

More specifically, 6G-SENSES considers the Open RAN (O-RAN) and 3rd Generation Partnership Project (3GPP) framework taking advantage of the disaggregation, virtualisation and network and service management capabilities inherent in O-RAN and supporting CF-mMIMO operation. A new network segment in-between the Wireless Access Technologies (WATs) and the UEs is envisioned to enable the control of advanced elements such as Reconfigurable Intelligent Surfaces (RISs). Taking a high-level view, the proposed system architecture combining all network components along with the Service Management and Orchestration (SMO) are depicted in Figure 1-1. Edge caching can be supported by the Edge Cloud functionality envisioned in the 6G-SENSES architecture located at the Multi-access Edge Compute (MEC) nodes as shown below.

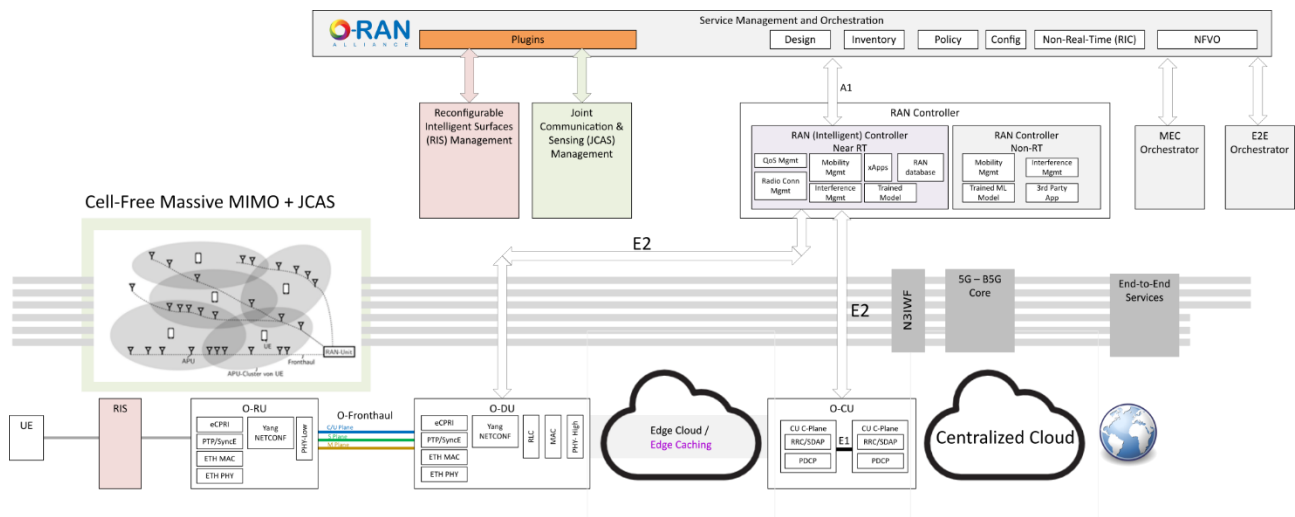


Figure 1-1 6G-SENSES Architecture

Within the 6G-SENSES project, a cross-layer approach will be developed to enhance edge caching mechanisms. This approach will integrate various features into an Artificial Intelligence (AI) / Machine Learning (ML) model for traffic prediction and characterization. These features will encompass multiple types of data, enabling a more accurate and efficient caching strategy that is responsive to the dynamic and complex nature of modern network environments:

1. **Network traffic characterization:** This will permit the classification of network user and device behaviour (e.g. bandwidth requirements, latency sensitiveness), thus providing useful insights to determine the level of caching needed by the network clients.
2. **Mobility information:** This allows proactive relocation of information to the location where and the time when the client will need it, based on predictive analytics.
3. **Content characterization:** Popular content will be selected according to statistics, client/service forecasting intelligence are excellent candidates for edge caching. This characterization can exploit big data analytics as well as AI techniques such as reinforcement learning or deep learning.
4. **Location-Aware Caching:** In IoT systems, where physical movement of agents is involved, traditional caching strategies fail to capture which data should be retained in memory. We propose a novel caching strategy which aims to capture the key values of caching in a system where mobility of both actors and sensors is fundamental.

The 6G-SENSES project will develop suitable edge caching algorithms that will be able to increase the required level of throughput and latency, while considering the limited power budget and processing resources of edge devices.

The main contributions of this deliverable are the following:

1. **Comprehensive Survey:** We provide a comprehensive survey of state-of-the-art Wireless Edge Caching technologies, offering a broad overview of the various layers that contribute to integrating Wireless Edge Caching into modern network infrastructures.
2. **Mobile Devices as Edge Servers:** We investigate a specific approach to Wireless Edge Caching that utilizes mobile devices as edge servers.
3. **Caching Policies Study:** We implement multiple traditional caching strategies in the context of mobile devices acting as edge servers.

4. **Simulation Implementation:** we implement our use-case in a physically accurate simulated environment. In the simulated environment, we implement multiple traditional caching strategies such as FIFO, LRU, and LFU.
5. **Performance Comparison:** We compare the performances of all the caching strategies, under different key performance metrics, such as **HIT/MISS ratio**, **E2E delay**, and **data freshness**.

1.1 Organization of the document

The deliverable is organized as follows:

- Chapter 2 explores the state-of-the-art of Wireless Edge Caching techniques, providing a comprehensive survey of Wireless Edge Caching technologies, examining all the key aspects that currently enable and motivate the use of caching.
- Chapter 3 introduces a specific use case for Wireless Edge Caching. This use case is tied to Use Case #1 presented in 6G-SENSES deliverable D2.1 [4]. In this deliverable, the aspects concerning the caching mechanism are explored in more details. The concept of Wireless-Fidelity (Wi-Fi) sensing is also presented, which ties directly to the cached data of the algorithm.
- Chapter 4 presents the Wireless Edge Caching model and evaluates the performance of multiple caching strategies in the use-case presented in Chapter 3.
- Chapter 5 summarizes the deliverable and outlines plans for the upcoming period.

2 State-of-the-Art in Wireless Edge Caching

Wireless edge caching [2], [3] refers to a novel distributed architecture for storing contents at the wireless edge networks, such as BS and user equipment (UE), to efficiently accommodate the proliferation of mobile devices and new data-hungry applications.

In order to efficiently deploy a Wireless Edge Caching solution, multiple challenges need to be addressed with different and specific application requirements. Achieving efficiency in Wireless Edge Caching requires joint optimization of network-layer caching and physical-layer signal transmission.

For application-specific use-cases, dynamic content popularity [6], [3] needs to be addressed in order to be able to deploy an efficient caching policy. This approach differs significantly from traditional caching models, where standard algorithms typically store data based on historical access patterns.

By considering content popularity, caching can move beyond simply retaining previously accessed data. Since content popularity fluctuates over time, maintaining an up-to-date view of requests enables caching to anticipate future access rather than relying solely on past data.

In social networks, data location is also crucial, as nodes in proximity are more likely to request the same data [8].

Although caching strategies for traditional Internet applications have been widely explored in the past, the unique characteristics of 6G applications drive the need for developing new caching mechanisms that consider these distinct features.

In this chapter, we provide an in-depth examination of the technologies that make Wireless Edge Caching possible. We explore the advancements, tools, and methodologies currently enabling efficient caching at the network edge, focusing on how these innovations contribute to enhanced performance, reduced latency, and optimized data management.

2.1 Wireless Enabling Technologies

Advances in network technologies have deeply influenced the way the network infrastructure is shaping in these last years. The advent of 5G technologies and the support of high bandwidth rates has enabled the possibility of having ultra-low latency data on the edge. This paradigm is fundamentally shifting the way data is retrieved from centralized servers, decreasing both latency and backhaul traffic, relieving the main servers from excessive computations.

Recent advances in network technologies have drastically changed the design and functionality of network infrastructures. With the emergence of 5G and now 6G technology, characterized by high bandwidth and ultra-low latency communication, a new paradigm has been established in which data is stored and processed closer to the end user, relieving the load from the central unit servers. This shift enables the real-time access of data and faster response time, which is critical in order to accommodate the demand of new emerging applications in the IoT ecosystem. Edge computing fundamentally changes the way data is retrieved from centralized servers. The way interaction changes is deeply dependant on the application, but the main paradigm lies in the concept of increasing the authority that resources on the edges have in interacting with the end-user, either by caching actual data content, or increasing the role authority that the edge servers partake in the application ecosystem.

The intrinsic distributed method that these solutions achieve improve the overall network efficiency, relieving the usage on central processing units (CPUs). They also minimize the use of remote cloud units, while providing faster access to data, additionally offering benefits to latency-sensitive applications.

2.1.1 Wireless Edge Caching Infrastructure

Wireless Edge Caching brings data closed to the end user, by decentralizing storage and processing, thus moving from traditional centralized datacenters to lower-tier infrastructures, such as BSs, cells, and user devices. This shift enables quicker access to content and reduced dependency on remote cloud servers, minimizing E2E latency and improving the overall perceived experience of the final user.

While much of the action occurs at the lower layers of the network, it is crucial to have a comprehensive understanding of the whole service infrastructure in order to develop robust and effective edge caching strategies. Having multiple layers working together and exchanging crucial information can deeply enhance the performances of the caching on the edge. Caching on the other hand, should consider how the network is layered, such that the caching algorithm and the requests prediction not only consider edge devices, but also the requests that the central cloud may evoke based on certain events.

Figure 2-1 shows an example of edge nodes communicating through wireless channel (dotted lines) with mobile devices, with one of them (the top one) receiving data from the infrastructure network through a wireless link.

2.1.2 Heterogeneous Network and Coded Caching

Data consumption has seen a tremendous increase over the past few years, increasing not only the E2E delay requirements, but also in terms of network capacity. This increase has so far been managed with the use of CDNs, which mirror data in multiple locations, thus reducing the load on the main servers and bringing content closer to the user. While these solutions greatly accommodate the needs of some modern applications, they are particularly efficient only when local storage and data rates are the bottlenecks of the infrastructure.

These characteristics are most often true in cellular networks, where BSs tend to not have enough storage. The last-hop wireless links have, however, fundamentally low throughput which, even with the increase provided by advanced technologies such as 5G and soon 6G, is still not enough to satisfy the exploding increase in demand.

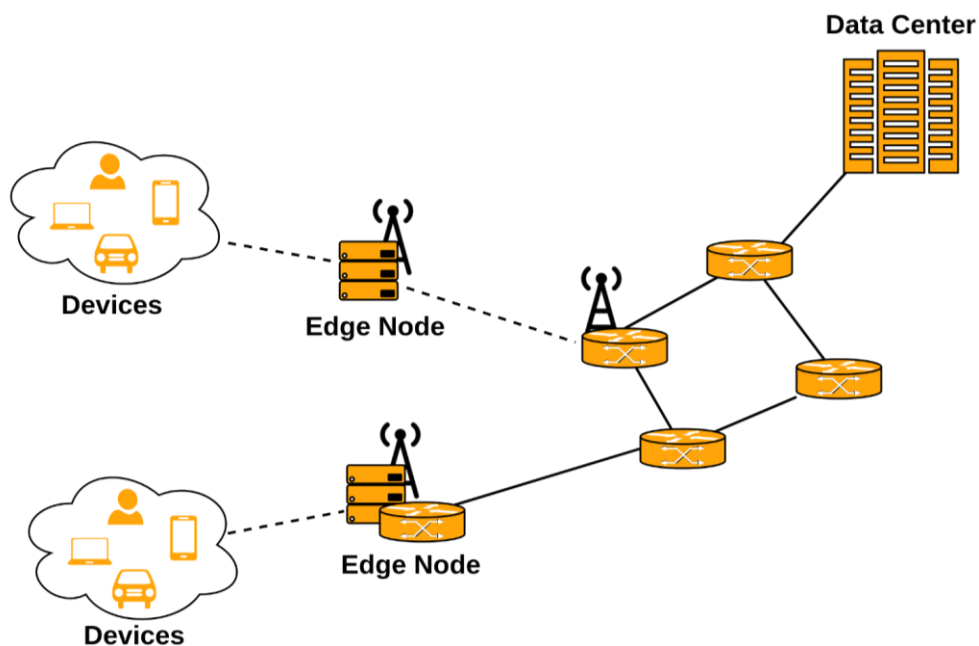


Figure 2-1 Example of a Wireless Edge Caching infrastructure

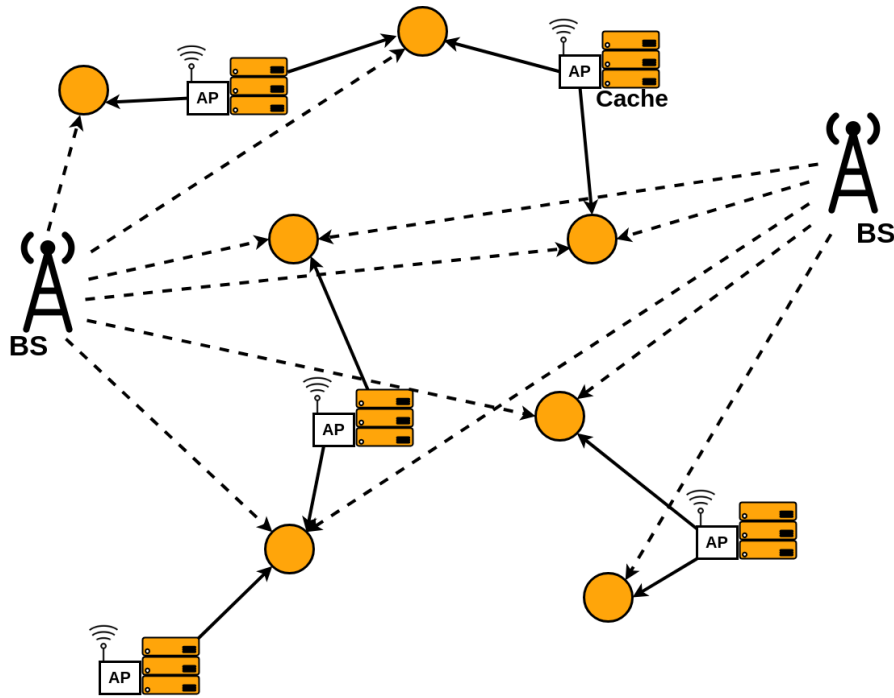


Figure 2-2 Coded Caching architecture in a heterogeneous network.

Coded caching was first introduced in 2012 [9] as a solution to the content distribution problem in a wireless setting. In order to focus on this new technique, the setup ignored variations in content popularity and limited user-to-cache access to exactly one user connecting to one cache. The authors showed that conventional caching techniques are inefficient in such a setup. Instead, one can leverage the broadcast capabilities inherent to wireless communications in order to send a small network-coded message that can serve a large number of users at once.

By providing storage capabilities at the network nodes (BSs and Wi-Fi access points) and creating a large-scale distributed cache, users will be served by connecting them to one or more nodes hosting their requested content. Delivery protocols will use algorithms that are aware of attributes of wireless networks like the broadcast medium and interference.

Figure 2-2 shows an example of a Coded Caching architecture in a heterogeneous network, with both BSs and Edge Caching Access Points providing content.

2.1.3 Cooperative Caching in Cloud-Assisted Wireless Networks

Cooperative caching aims at utilizing the computational capabilities of edge devices to handle network requests, thereby reducing the workload on central servers. The type of cooperative caching that takes place can vary based on the operation and the authority that the edge servers take in the network infrastructure.

Figure 2-3 shows an example of a Cooperative Caching architecture, where multiple layers contribute to bring data to the clients. Caching devices on the edge receive data from both a central unit and the clients.

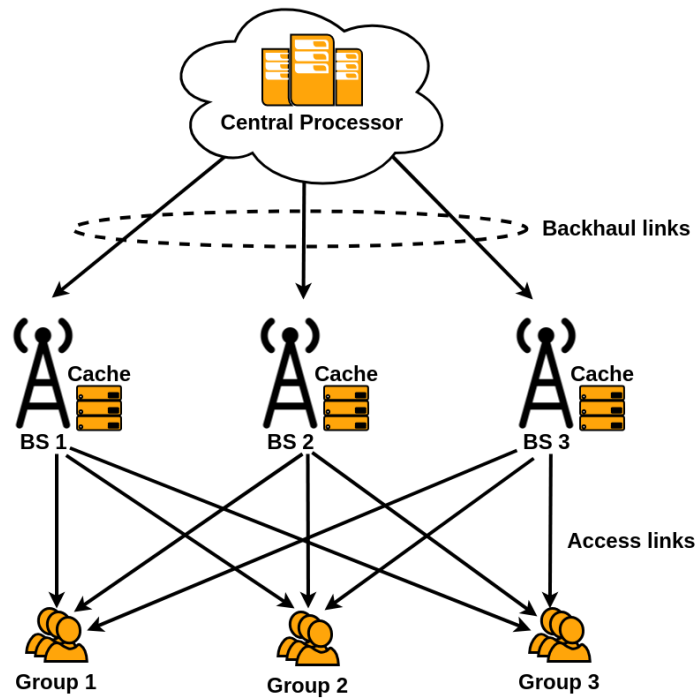


Figure 2-3 Cooperative Caching architecture.

2.1.3.1 Cloud Radio Access Network (C-RAN):

C-RAN leverages virtualization technology to consolidate BS functionalities within a centralized cloud infrastructure. In this architecture, the typically heavy and complex tasks performed by BSs are offloaded to a CPU, enabling more efficient resource management. A C-RAN is composed of lightweight, distributed radio remote heads (RRHs), which handle the radio frequency transmission and reception, and are strategically placed across the network to ensure wide coverage. These RRHs are connected to the centralized CPU via low-latency, high-bandwidth optical fibers, which facilitate fast and reliable communication between the distributed units and the central processor. This setup enhances the scalability and flexibility of the network while reducing operational costs.

2.1.3.2 Multi-Access Edge Computing (MEC)

Multi-Access Edge Computing (MEC) [15] focuses on enhancing mobile networks by bringing storage, computing, and networking resources closer to the edge of the radio access network (RAN). This architecture places commodity servers at the edge of the network, allowing them to handle tasks such as application execution and data processing. By positioning these resources closer to end-users, MEC significantly reduces latency and enables faster data transfers, as information no longer needs to travel to a centralized cloud for processing. This proximity improves user experiences for applications that require real-time data, such as video streaming, gaming, and IoT services, while also alleviating the load on core network infrastructure.

2.1.4 Wireless Device-to-Device Caching Networks

Caching on the edge involves more than simply relying on the final nodes or cells in the network infrastructure. End-users' devices, such as smartphones and tablets, can serve as a valuable resource for caching purposes [14], [15]. By leveraging the storage and processing capabilities of these devices, network efficiency can be significantly improved. This technique is particularly useful when there is a lot of redundancy of the data transmitted, such as streaming services. In these cases, caching content in the devices can be shared with nearby devices, thus reducing the repeated request to the centralized architecture.

However, while edge caching between devices presents notable advantages, it poses significant challenges, which mainly pertains security risks and increased power consumption.

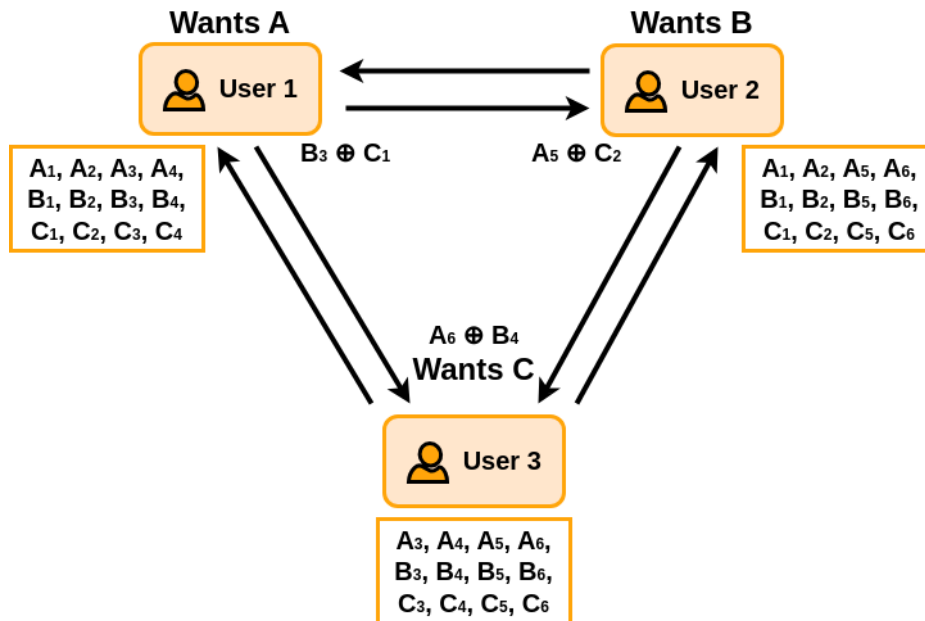


Figure 2-4 Device-to-device data sharing

Allowing direct communication between devices may expose users to malicious content or unauthorized access. On the other hand, keeping devices active to service other users may lead to some devices in the network being particularly busy and thus consuming more power than what the user may expect from the application.

Figure 2-4 shows an example of a device-to-device data sharing architecture, where devices leverage properties from coded caching to efficiently use the broadband of a wireless devices.

2.1.5 Mobility-Aware Caching in Cellular Networks

Mobility plays a crucial role in addressing the challenges of caching within wireless networks, especially in dynamic environments where users are constantly on the move. An effective caching strategy should go beyond simply storing frequently accessed data closer to the user, it must also anticipate user mobility and proactively relocate cached data to different edge nodes based on predictive models on the movement of the user. By doing so, the system can ensure that relevant data remains accessible as users transition between different cells or network areas.

This proactive approach can significantly enhance Mobile Edge Caching strategies by ensuring seamless session continuity, even during user handovers between different network cells. As a result, users experience lower E2E latency and improved service quality; this is particularly beneficial for latency-sensitive applications such as video streaming, gaming, or real-time communication. Predictive caching mechanisms not only optimize resource allocation but also reduce the likelihood of interruptions, making mobile services more reliable and responsive in dynamic network environments.

Figure 2-5 shows an example of a MEC application in execution, where session-continuity is guaranteed by moving cached data between BSs that in range to the user.

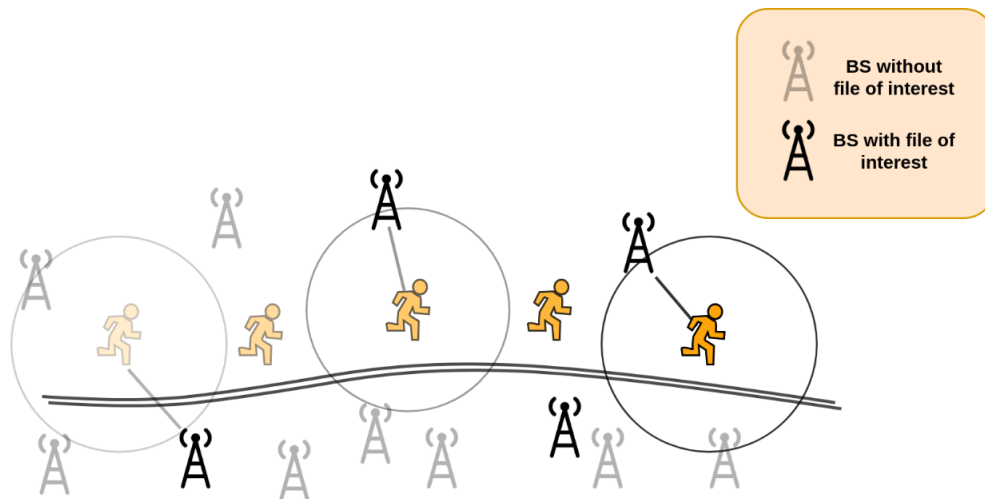


Figure 2-5 MEC application in execution.

2.1.6 Information-Centric Networking

Information-Centric Networking (ICN) [17][10] is proposed as the next-generation network paradigm, redefining how data is managed and accessed. By shifting away from the traditional IP-based communication, ICN focus on handling information at the content level, introducing a system where data is broken into content chunks. In such model, users are no longer required to establish a direct, E2E connection with the server in order to access content. Instead, they send an *interest packet*, a request which specifies the desired content by its unique name, into the network. The response can then come from any network node that may have cached the requested data, independently from its original source.

This paradigm shift allows data retrieval to become more efficient, as content can be delivered from nearby nodes that store cached copies rather than from a distant server, improving both latency and bandwidth usage in the network. This approach, additionally, inherently supports user mobility, as content can be accessed in a more flexible way, without the dependence to a specific source location.

ICN is particularly well-suited for applications in the IoT ecosystems [5], as they are intrinsically content-centric, prioritizing data over its source location. This aligns perfectly with the ICN paradigm, enabling efficient, location-independent data distribution, ensuring robust performance even in a dynamic, data-intensive environment.

Thanks to its innovative design, ICN is perfectly suited to address key challenges in network efficiency, scalability and adaptability. It seamlessly integrates with the current landscape of Wireless Edge Caching, enhancing how data is stored and distributed across networks.

Today's Internet's architecture is said to be based on a *"Thin Waist"* model, centered on a universal network layer, IP, which implements the minimal functionality necessary for global inter-connectivity. In Figure 2.6, it can be seen how ICN plans to replace the IP protocol, or wrapping it, with content chunks-based network.

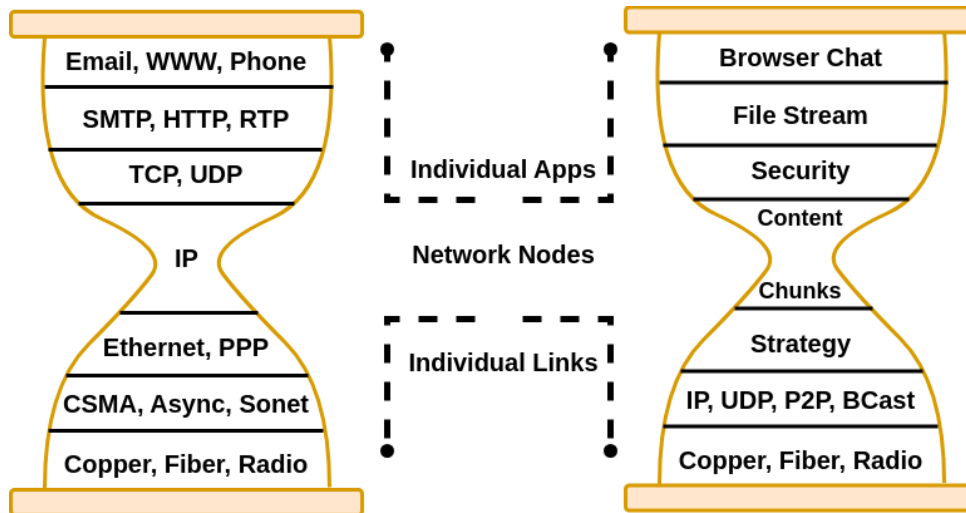


Figure 2-6 "Thin Waist" design of the modern network infrastructure.

2.1.7 Federated Learning

As mentioned in the previous subsections, **6G-SENSES** adopts the concept of edge caching to store data closer to the end-devices, typically in regional servers or edge devices. The goal is to jointly reduce latency and minimize bandwidth usage by caching frequently requested content or computational results near the "edge" of the network. Edge Caching and Federated Learning (FL) can be integrated in **6G-SENSES** improving performance and security.

Specifically, FL is a decentralized ML approach where multiple devices or nodes (such as smartphones, IoT devices, or edge servers) collaboratively train a model without sharing raw data. Instead of sending data to a central server, each device trains a local model on its own data, then only shares model updates (such as weights or gradients) with a central server, which aggregates these updates to create a global model. This assists in maintaining privacy and minimizing data transmission.

Edge caching and FL can be combined providing the following benefits to the system":

1. Efficient Data Distribution: In a FL setup, the data required to train a model might be distributed across multiple devices. edge caching can be used to pre-store model updates, training data (or metadata). This reduces the time it takes for devices to access necessary information for model updates, improving the efficiency of the FL process.
2. Reduced Latency in Model Training: Since FL often involves multiple rounds of communication between edge devices and a central server, edge caching can help cache the model parameters or updates at edge nodes, reducing latency during model synchronization. This can speed up training processes as devices do not have to send or receive updates directly from distant central servers.
3. Optimization of Bandwidth Usage: FL involves sending updates between devices and a central server. Edge caching can help reduce bandwidth usage by caching common model updates or shared data at the network's edge. Devices can then retrieve necessary updates locally rather than transmitting large amounts of data to the central server, improving scalability and minimizing congestion in the network.
4. Improved Privacy and Data Security: Both edge caching and FL emphasize data privacy. FL ensures that personal or sensitive data remains on the device, and edge caching ensures that less data needs to be transmitted across the network. By reducing the need to move raw data and model updates across long distances, these techniques can enhance security by minimizing exposure to potential breaches.

FL is a specific category of Distributed ML with distinct differences from other categories of distributed ML. This methodology addresses various problems that have emerged in ML, such as the existence of hundreds of thousands of devices with different computing capabilities scattered worldwide, that collect and generate a volume of data and information larger than ever. The increasing presence of such devices in our lives has sparked a significant debate about the personal nature of each user's data and how it can be ensured in the modern environment we describe. FL allows large organizations with thousands of clients to collaborate to produce better predictions than learning models, while simultaneously preserving the personal information of their clients. FL systems are divided into cross-device systems, consisting of thousands of clients, each with its own data, and cross-silo systems, primarily involving large organizations with datasets from many users.

Depending on how the data is distributed and its structure in a FL environment, FL is divided into three different categories:

1. **Horizontal Federated Learning:** Horizontal FL is used for datasets that share the same feature space but have different sample spaces. It can have two different architectures, with one involving a server and the other where one of the participating users plays the role of the server.
 - i. Architecture with a server: This architecture is also known as centralized FL. In this, the server coordinates the process of aggregating local updates and updates the local users with the global model that emerges (Figure 2-7). During the model training process, the parameters exchanged can be either weights or gradients of the loss function. This practically means that each user either trains the initially sent model and updates the parameters, which are then sent as updates, or calculates the gradient of the loss function, as described earlier, and sends this gradient as an update to the server. The above process continues until a specific desired model accuracy is achieved or a predetermined number of iterations of the process is completed.

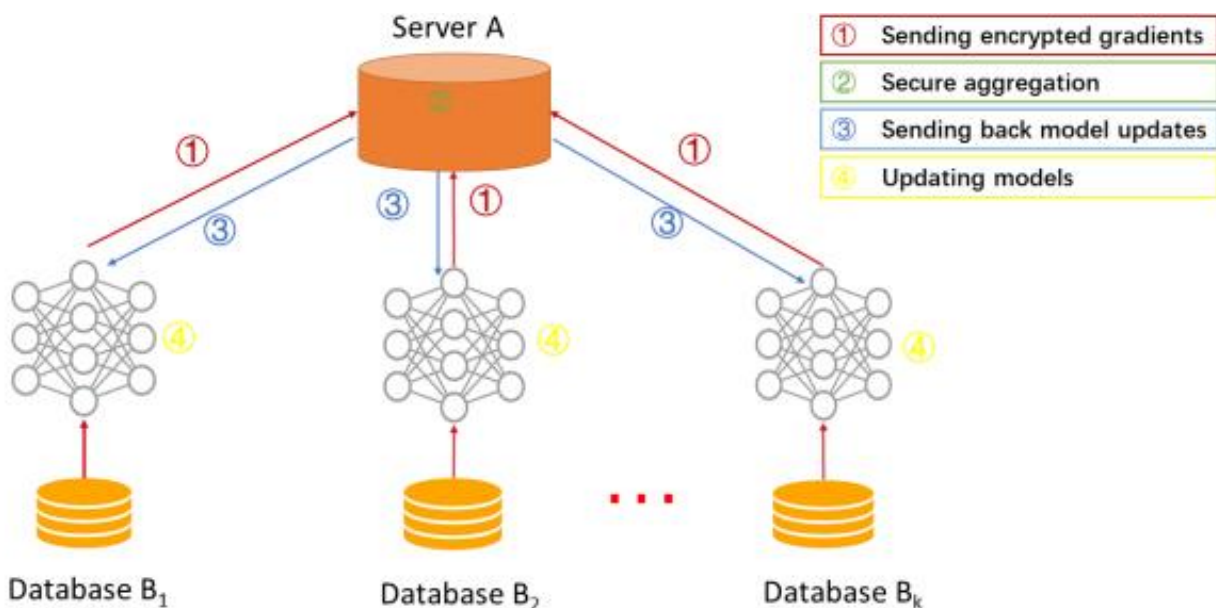


Figure 2-7 Indicative Architecture of a FL System with a Server.

- ii. Architecture without the Presence of a Server: In this specific architecture, there is no central server coordinating the training process. Instead, updates are sent directly between users, and they adjust their models based on the parameters of other users, generating new updates by training the model on their own data. Due to the absence of a server coordinating the entire process, the existence of protocols is necessary. These protocols can be either cyclic transmission or random transmission.

2. **Vertical Federated Learning:** In vertical FL, two datasets share the same sample space but differ in the feature space. Vertical FL aims to leverage the union of the two available feature spaces for the same customers to make any predictions. Thus, with vertical FL, various organizations can collaborate to achieve better results since they have much more available features for the same customer samples. The primary risk of user data leakage in this context occurs when there are curious but honest users who may seek to extract information about the data of other users. In this case, there are two proposed architectures, with or without a third-party organization coordinating the process.

3. **Architecture with a third coordinator:** Assuming that 2 clients, C1 and C2, collectively aim to train a FL model using their local available data. The two clients, C1 and C2, are honest but curious, meaning we assume their data is valid, but they may seek information about the other client's data. A 3rd client, C3, plays the role of a coordinator. We assume the coordinator is a trusted organization.

The described architecture ensures that C3 cannot extract information about the data of C1 and C2 since it only sees the local updates with the addition of a noise factor (mask). C1 and C2 have access to each other's gradients during the training process, but apart from the fact that these may concern features not present in the other feature space, the gradients at each corresponding stage are not sufficient to extract information about the other client's data. We assume that the number of samples, N_a , is much larger than the number of input features, n_a . We considered that clients C1 and C2 are honest but curious. In the case that one of them is malicious, they can provide a suitable unique non-zero input with a unique non-zero feature. In this case, they may determine the gradient of the other client for that specific feature of the sample. However, this is not sufficient to determine either the parameters or to extract information about the data. Additionally, the other client will easily detect malicious actions from the influence they have on the model. During the prediction/estimation operation, as shown in the table, there is also no information leakage.

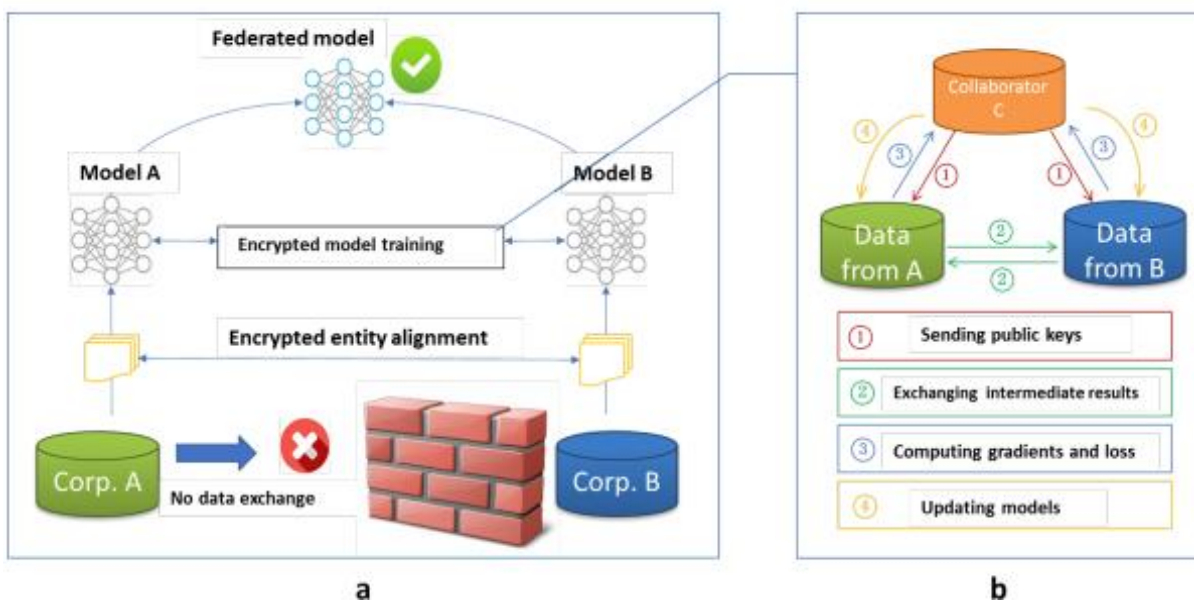


Figure 2-8 Indicative Architecture of Vertical FL with the participation of two major organizations

3 Caching Use-Case

Our proposed use case is tailored to the network scenario depicted in the Use Case #1, extracted from 6G-SENSES deliverable D2.1 [4].

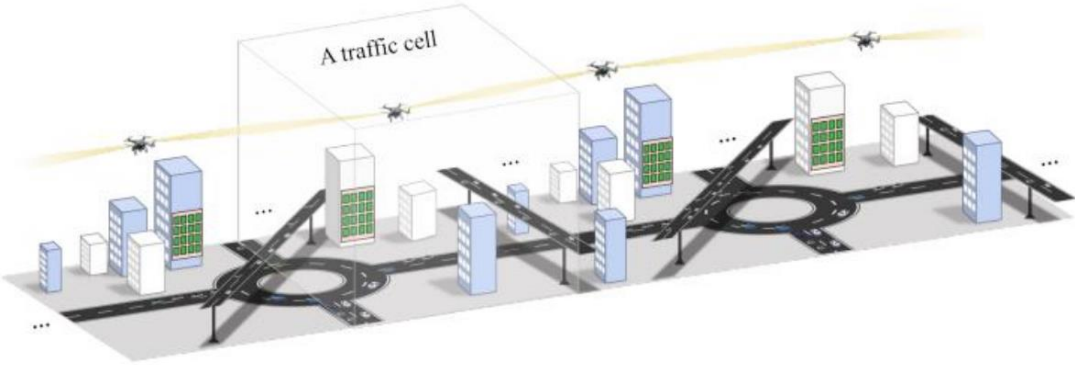
In the context of IoT, sensing plays a crucial role. By collecting data taken directly from the field, sensors are able to monitor remote and hazardous environment without the need of human intervention.

Sensing devices are able to produce great amount of data, such as constant temperature readings, motion detection logs, or also video feeds from cameras, which have to be delivered and processed by a central unit, in order to ensure seamless operations. Sensors are, most of the time, part of a wireless network of devices, used to relay information to central units [24].

For operations to be carried out smoothly, this network must remain consistently connected and online. However, the dynamic nature of wireless networks can compromise the availability and reliability of sensing services, as well as to reduce the available bandwidth for the client connected to the network. Continuous data caching can help in solving issues related to the temporary unavailability of the communication channel, preventing a permanent loss of critical sensing data. Furthermore, it can decrease the requirements in terms of channel bandwidth and reduce the network latency. While sensing data can provide insightful information in order to optimize the communication channel, it cannot cover those instances during which disruptions cannot be avoided.

In Table 3-1 we provide a more in-depth review of the use case, and all its components.

Table 3-1 In-network, Location Aware Caching Service for Disruption Minimization

<i>Use Case: In-network, Location Aware Caching Service for Disruption Minimization</i>
<p>Roles - Stakeholders: RAN/O-RAN/RIS operator (capture/process sensing information), Network Operator (NOP) takes decisions based on the information, Service Provider, Mobile Caching Device (caches data when necessary/to reduce delays)</p>
<p>Objective: The main objective of the proposed use-case is to reduce the service disruption of the sensing activity by leveraging novel in-network caching techniques.</p>

<p>Figure 3-1 Application case with UAVs acting as mobile caching devices.</p>

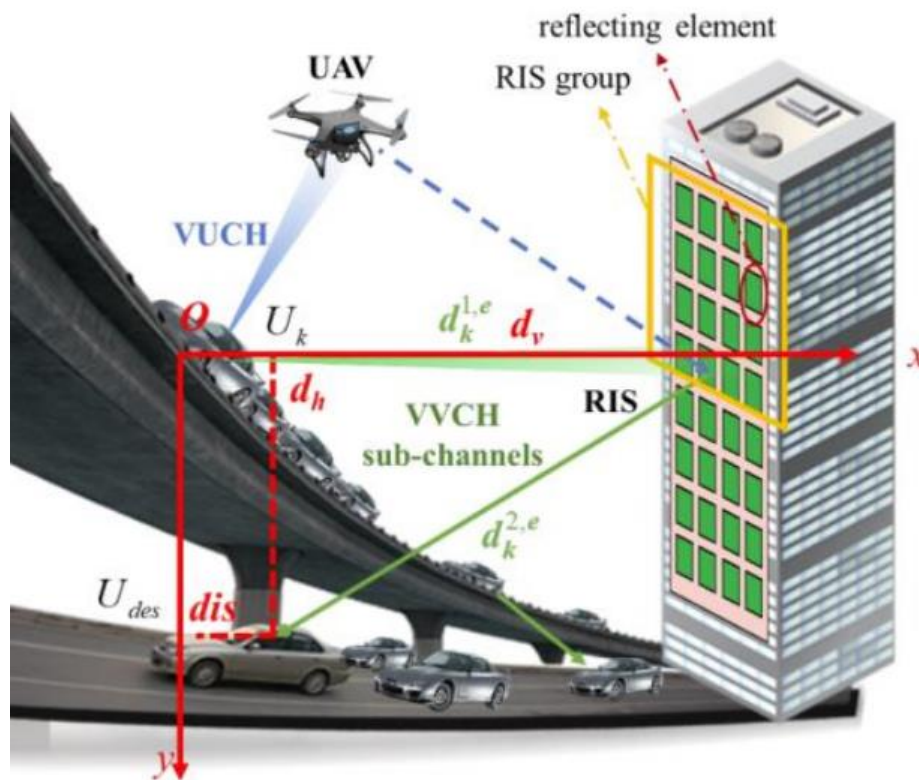


Figure 3-2 Example on how RIS can be used to enhance the communication channels.

Description:

RIS-assisted V2X Architecture

RIS-assisted sensing panels can be integrated with devices able to perform sensing data to enhance the overall performance of communication and sensing systems [18][19][20]. The integration involves leveraging the capabilities of RIS and can lead to both optimization of wireless communication links and improvement of sensing capabilities. One aspect of integration is utilizing RIS to enhance the quality of communication in Integrated Sensing and Communication (ISAC) systems that 6G-SENSES is focusing on. The intelligent tuning of the electromagnetic properties of the RIS panels can improve signal quality, increase coverage, and mitigate interference. This leads to improved reliability and higher data rates in the communication link between sensors, devices, or networks involved in ISAC applications.

Current status, Problem statement - limitations of today's situation.

The concept of caching at the wireless edge has undergone a profound transformation in recent years. The rapid evolution of emerging technologies has redefined many long-standing principles that were once rooted in existing network infrastructures.

While numerous studies have explored caching solutions in traditional wireless networks, these approaches tend to perform well only in static or conventional wireless edge environments. Such approaches often struggle to adapt to scenarios where mobility is an integral part of the architecture. In highly dynamic, mobile-driven networks, with multiple mobile devices, the limitations of conventional caching strategies become more pronounced.

Service Definition:

RAN/RIS providers can deliver sensing data as a service to NOPS, enabling network optimization either on-demand or automatically for self-optimization by the RAN operator. Additionally, RAN/NOP

operators can provision sensing data to service providers (SPs) to support the deployment of specialized vertical services for end-users.

SPs can offer applications and services with physical awareness capabilities, incorporating Collaborative Robots and Cooperating Mobile Robots to operate effectively in both static and dynamically changing radio environments.

Moreover, MEC devices can supply data to applications and services when RAN/RIS sensing capabilities are insufficient to meet the required E2E performance, by leveraging caching mechanisms. This can result in faster responses than simply retrieving data directly from end-users.

6G-SENSES Wireless Edge Caching Technical Solution/Innovation

Our proposed solution addresses a key aspect of this evolving architecture, which it is still in its early stage. Although the concept of caching in mobile-assisted environments is not entirely new, it has largely been confined to theoretical explorations. The technologies necessary to fully implement these ideas, particularly those that support the dynamic and flexible nature of mobility, are still in the early stages of development. As a result, practical applications have been limited.

However, with the advent of 6G and the novelties introduced by the 6G-SENSES project proposal, we are presented with a unique opportunity to establish a new paradigm. Our approach enables us to test and validate the efficiency of new proposed caching protocols, by leveraging specific architectures and cutting-edge technologies designed to meet the demands of this highly mobile environment. By doing so, we aim to bridge the gap between theory and practice, laying the groundwork for a new era of Wireless Edge Caching.

Table 3-2 Sensing with Wi-Fi system

Use Case: Sensing with Wi-Fi system (presence detection)

Roles - Stakeholders: Service Providers (sensing data provider)

Objective:

The primary objective is to integrate sensing technology within Wi-Fi systems, enhancing both communication and sensing capabilities. Considering the 6G capabilities landscape, this use case aims to contribute to the development of technologies with applicability on (IMT 2030) new capability “ISAC,” especially enabling Immersive Communication and partially Hyper Reliable Low Latency Communication (HRLLC) usage scenarios. This use case details on the Next Generation Mobile Networks (NGMN) Enabling Service and Network Evolution categories. Considering the SNS ICE use case categories, the concepts to be implemented in this use case will be directly applicable to various service categories, e.g., the “Immersive Experience / Seamless Immersive Reality” along with “Physical Awareness” service components in various environments – where environment sensing will be needed.

Description:

Current status, Problem statement - limitations of today’s situation.

The current SoTA primarily utilizes passive Wi-Fi signals for sensing, but the active approach proposed here is a novel concept that has yet to be fully explored. By integrating active sensing, we can expect to see a new wave of innovation in ISAC, leading to more immersive Augmented Reality (AR) / Virtual Reality (VR)/ Extended Reality (XR) experiences and a transformative impact on user interaction with laptops and other smart devices. This proposal not only aligns with the current trajectory of technological advancement but also charts a course for future innovations that will enrich our interaction with the digital world. An additional significant impact of active sensing lies in motion presence detection. This capability plays a crucial role in managing network resources and optimizing of the network based on

sensing information. By intelligently detecting the human presence, the system can optimize the energy and resources of the network based on this information.

6G-SENSES Concept

Considering the 6G vision, a multitude of basic and advanced services will be delivered to verticals or individuals as end users in versatile environments. The incorporation of sensing capabilities is a key enabler for the support of services such as “Immersive Experience/ Seamless Immersive Reality” along with “Collaborative Robots/ Cooperating Mobile Robots”. Multiple RAN Resource providers/RAN Operators undertake the role of deploying RAN elements (of various technologies), including Wi-Fi, that enable ISAC. The sensing data/information captured by RAN (Wi-Fi) resource providing elements/roles is either processed for network optimisation or sent to a provider as a service-to-service providers/verticals; to enable Immersive Communication and Collaborative Robots service elements for relevant applications. In more technical terms, considering the technical limitations of current solutions, the algorithms developed for this purpose are designed to refine the process of delay and Doppler estimation, which are critical parameters in determining the relative motion and distance of objects in the vicinity of the sensor.

Service Definition:

RAN/ NOP Operator provisioning sensing data to SPs – for SPs to enable specific vertical services to end-users. SPs to provide applications/services with Immersive Experience/Seamless Immersive Reality along with Collaborative Robots/Cooperating Mobile Robots service components in various environments. MEC device provides data to the applications/services when RAN/RIS sensing is not enough, or to improve E2E delay when caching is faster than data retrieval from the end-user.

6G-SENSES Wireless Edge Caching Technical Solution/Innovation

Our 6G-SENSES solution leverages advanced Wireless Edge Caching to enhance Wi-Fi sensing presence detection. In this context, presence detection involves using Wi-Fi signals to identify the presence of individuals or objects within a specific area. By implementing advanced caching protocols, our solution ensures that presence detection data is reliably stored and quickly accessible, even in the event of network disruptions. This reduces the risk of data loss and ensures continuous monitoring. Continuous data caching helps to optimize bandwidth usage by storing frequently accessed data locally. This reduces the need for constant data transmission over the network, freeing up bandwidth for other critical operations and reducing overall network congestion.

4 Caching Model

We devote this section to an in-depth exposition of the caching model utilized in the use-case presented in the previous chapter. We explore the implementation specifics of the simulation framework used to evaluate multiple caching strategies and present the results based on metrics such as **HIT/MISS ratio**, **E2E delay**, and **data freshness**.

Traditional caching strategies are usually applied in environments where data is structured hierarchically. Memory is organized into tiers, with lower tiers featuring slower read performance. It is generally assumed that accessing and transferring data within the same memory tier requires a similar amount of time.

In a sensing context, where sensors output data which is sent/offloaded to the nearest device acting as an edge server, this assumption falls short. There is an intrinsic difficulty when data is being retrieved by sensors or other mobile devices in the field, which depends on the device's position, its offloading capabilities, the energy consumption of the sensing tasks takes, and more.

In the following, different caching strategies in the context of Wireless Edge Caching are analysed. To that end, the simulation environment and structure are first presented. Afterwards, the performance of the multiple strategies is discussed.

4.1 Description of the simulation framework

We study the effectiveness of various caching strategies over a series of multiple simulated scenarios. We utilize two widely recognized tools in the IoT and robotics domains: the Robot Operating System (ROS) and Gazebo. ROS serves as a versatile and powerful framework, offering a rich collection of libraries and tools designed to facilitate the development and deployment of standardized robotic applications. Complementing ROS, Gazebo provides a highly accurate and physics-based simulation environment, enabling realistic testing of robotic systems in virtual settings. Together, these tools ensure a rigorous and reliable evaluation of our implementation under controlled and realistic conditions.

4.1.1 Robot Operating System (ROS)

ROS [21] is a set of software libraries and tools that help developers build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what is needed for developing robotics project.

With its open-source approach and the broad compatibility across multiple systems, the flexibility of the ROS framework allows to easily move solutions from simulation-based implementation to real hardware. ROS simplifies the development of robotic systems by enabling modular and reusable code. This is achieved through a component-based architecture that allows various software components to interact seamlessly, regardless of the programming languages or hardware platforms they are running on.

The primary entity in a ROS system is the node, an abstract concept that represents a key component within the robot's ecosystem. Nodes interact among them through various interfaces to enable coordination and ensure the seamless operation of the entire robotic system. ROS nodes interact following different communication paradigms, as commented below:

- **Topics**

Topics enable asynchronous, many-to-many communication between nodes. Nodes can publish messages to a named topic, and other nodes can subscribe to that topic to receive messages. Topics are ideal for continuous data streams, such as sensor readings.

Figure 4-1 shows a graphical example of a ROS topic, and how it connects different nodes with a publisher/subscriber approach.

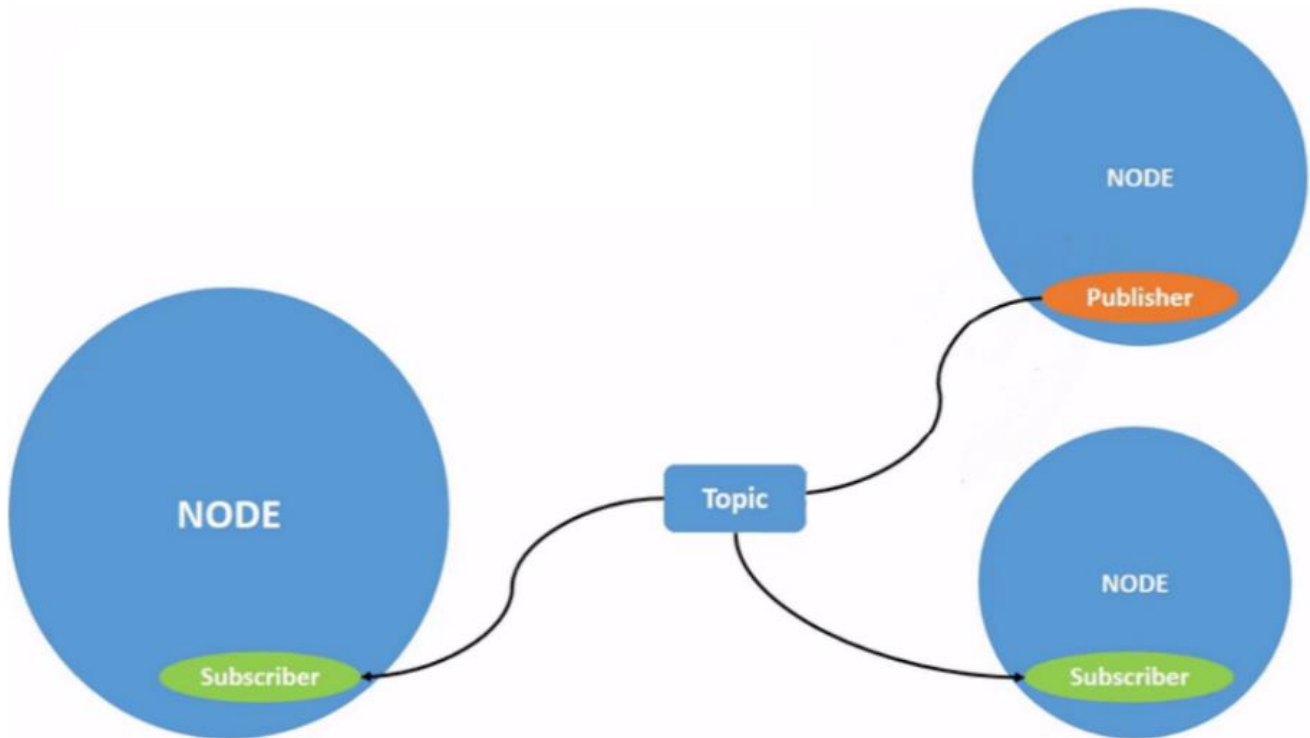


Figure 4-1 Example of a ROS topic

- **Services**

Services support synchronous, one-to-one communication. They are used for short-lived tasks that require a direct request-response mechanism. Each service has a name, a defined request structure, and a corresponding response structure.

Figure 4-2 shows an example of a ROS service, where a service server is advertised by a node. Multiple nodes can then act as service clients, making requests to the server.

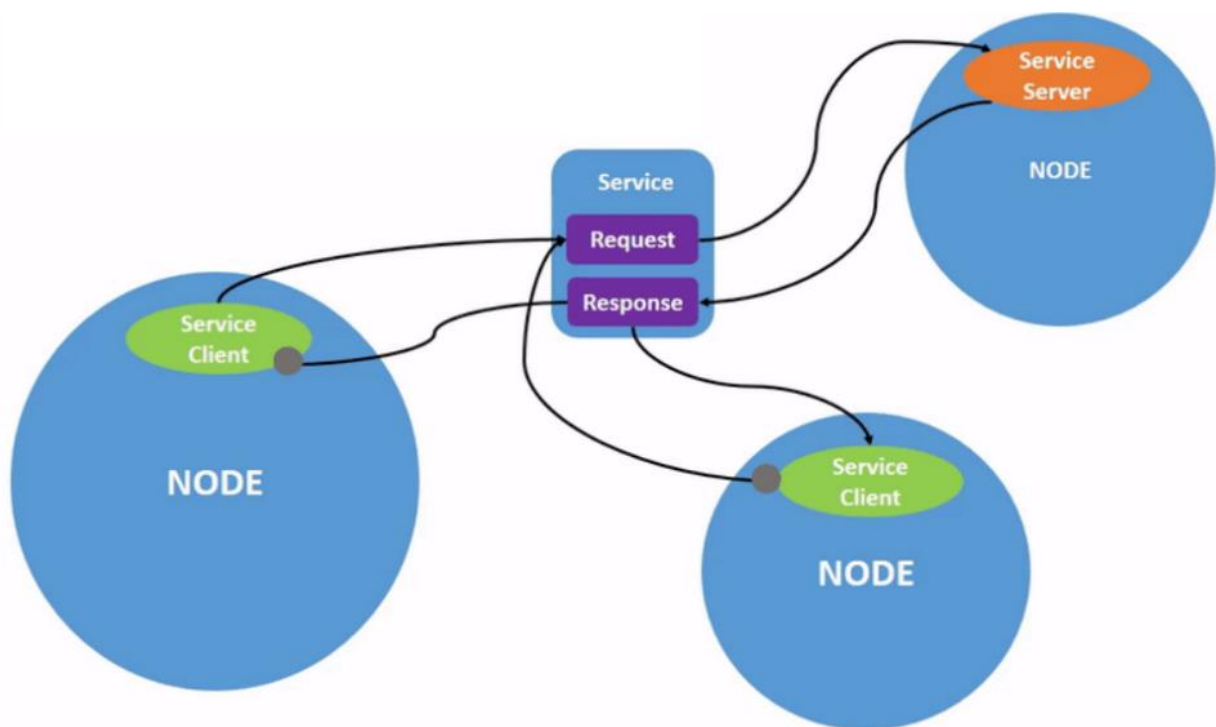


Figure 4-2 Example of a ROS service

- **Actions**

Actions are a hybrid mechanism for long-running, asynchronous tasks, that provide feedback during execution. An action client sends a goal to an action server, which processes the task and sends periodic feedback and a final result. Actions are well-suited for tasks like motion planning or object manipulation, where the operation takes time and intermediate updates are beneficial.

Figure 4-3 shows an example of a ROS action. Internally composed of two services and one topic, it provides a solid interaction for long-lasting requests.

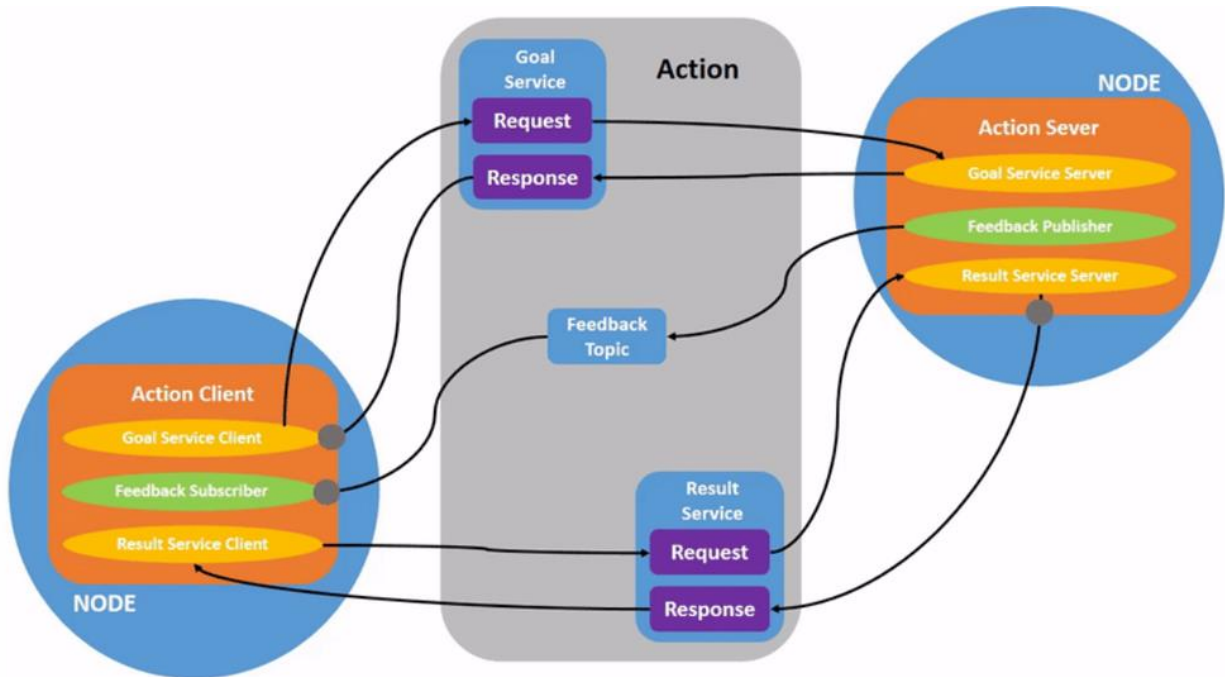


Figure 4-3 Example of a ROS action

4.1.2 Gazebo

Gazebo [22] is a 3D dynamic simulator with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. Gazebo offers physics simulation at a high degree of fidelity, a suite of sensors, and interfaces for both users and programs.

It is directly tied to ROS, and solutions developed in Gazebo can be seamlessly integrated with ROS topics and services, allowing for a standard ROS interaction between the two environments.

The flexibility of the environment allows for the direct deployment of solutions developed in Gazebo in real hardware, reducing the weight, time, and cost of real-world simulations.

We employ Gazebo in our solution due to its highly flexible plugin architecture, which enables the realistic modelling of hardware components, including complex elements such as wireless interfaces. This capability is essential for accurately simulating interactions and performances in our use-cases, providing a reliable foundation for future advancements and scalability in our work

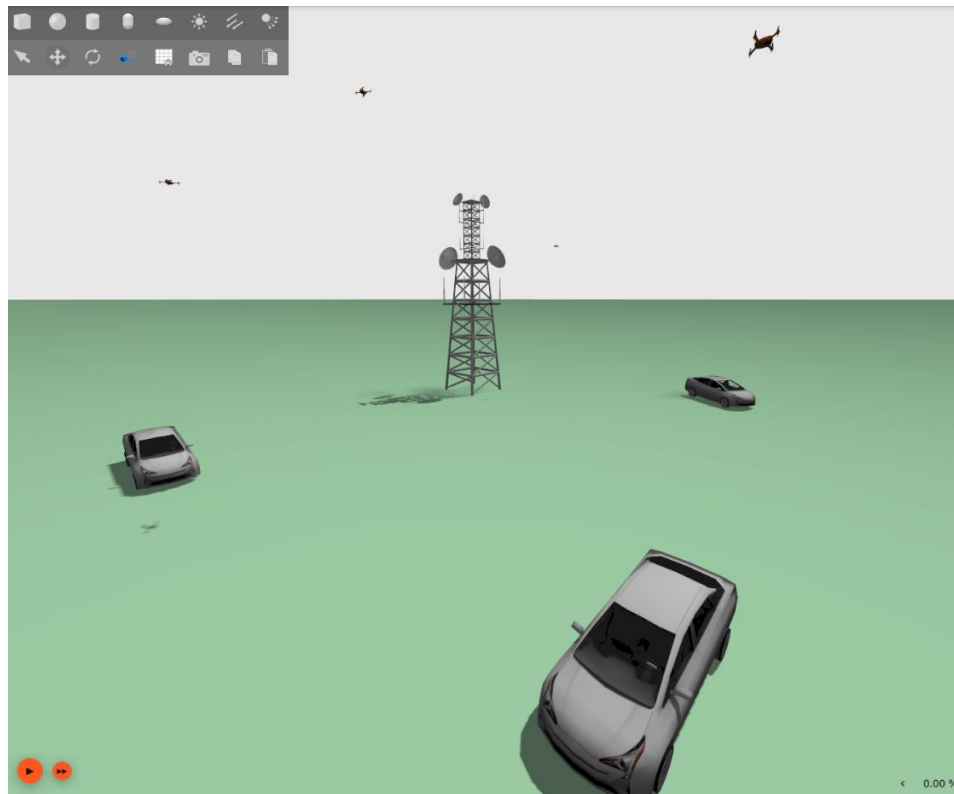


Figure 4-4 Running instance of Gazebo modelling the use-case of Chapter 3

4.1.3 Methodology

The simulated sensing scenarios embrace three types of entities: BS, caching devices in the form of UAVs, and mobile sensors.

- **Base Station (BS):** Placed at the centre of the simulation scenario, it periodically requests sensors' data by querying the data on the caching devices.
- **Caching devices:** The caching devices, in this particular case UAVs, are placed in regular intervals along the circumference of a circle with a pre-defined radius. This radius grows as the number of placed devices increases, maintaining an optimal distance between UAVs with respect to the communication range of the sensors on the field.
- **Sensors' placement:** The sensors are spawned randomly inside the coverage area of the caching devices. The deployment is configured in such a way that at the start of the execution all sensors are able to send their sensing data to at least one caching device. Sensors may move outside the range of the UAVs during the execution.

Figure 4-4 shows an example of the simulation instance running in Gazebo. In this context, caching devices are mounted on top of UAVs, while sensors are the moving cars. Evaluation tests are run on simplified graphic instances, in order to save computational power.

4.2 Structure of the Simulation

The behaviour of the simulation is implemented through multiple classes that communicate with each other through ROS topics and services. An experiment is executed inside a Gazebo simulation, running in real time. We devote this section to present all the classes that bring the whole simulation together. The definition of each class includes an overall description, the class configuration/setup and the functional components it

implements. Figure 4-5 illustrates the hierarchy of all the classes operating within the simulation. Each class is individually described in detail in the next sections.

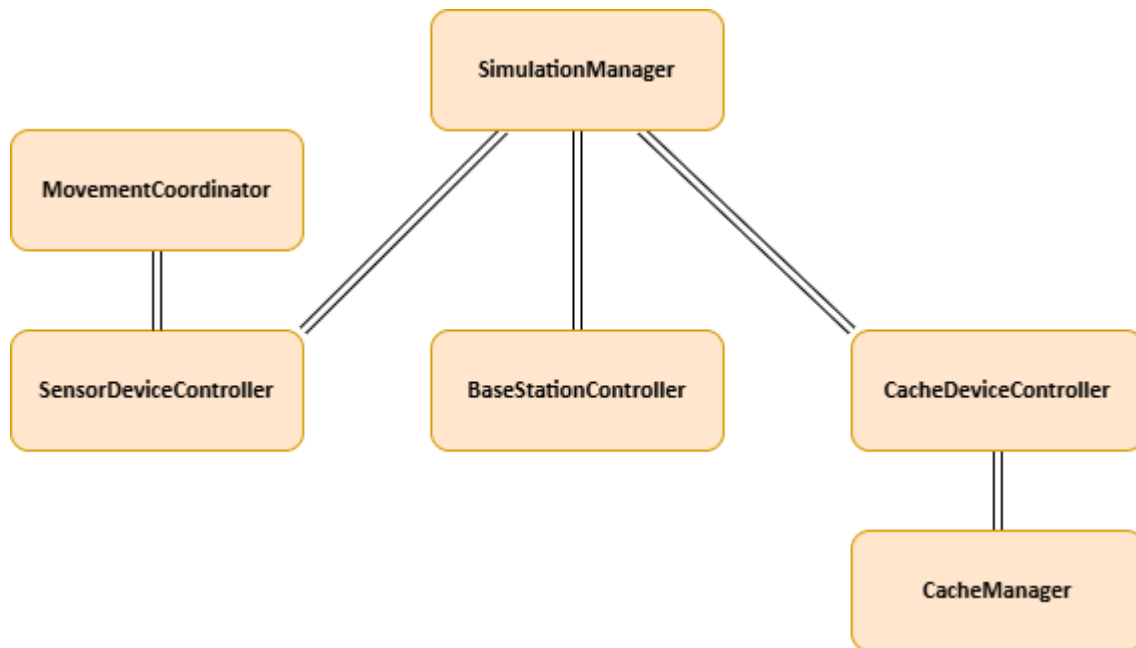


Figure 4-5 Structure of the Simulation

4.2.1 Simulation Manager

Table 4-1 Simulation Manager

Class: <i>SimulationManager</i>
<p>Class Description: The <i>SimulationManager</i> class orchestrates the interaction between mobile sensors and caching devices within the system. Its most important task is the forwarding of sensor data to nearby caching devices.</p>
<p>Initialization and Setup: At startup, the <i>SimulationManager</i> automatically configures as both publisher and subscriber to all the topics required to read the state of the devices in the simulation. It simulates their communication interfaces by using ad-hoc topics, namely, <i>tx_data</i> and <i>rx_data</i>, used by all the sensors and devices in the simulation instance to push new, and listen for, incoming messages.</p>
<p>Components:</p> <p>Position Management The class implements the <i>store_sensor_position()</i> and <i>store_device_position()</i> methods to keep track of the current positions of the sensors and caching devices. Notably, since the caching devices are deployed and then stay fixed, the subscription on devices' odometry is destroyed as soon as the first position measurement is registered.</p> <p>Data Forwarding The <i>forward_data()</i> method is responsible for relaying sensor data to the appropriate caching devices. It evaluates the distance between each sensor and caching device, ensuring that only nearby devices receive the data:</p>

- The method calculates the Euclidean distance between the sensor and each device using the *point_distance()* function.
- If a device is within the defined sensors range, it publishes the received sensor data to its corresponding topic.

4.2.2 Base Station Controller

Table 4-2 Base Station Controller

<i>Class: Base Station Controller</i>
<p>Class Description:</p> <p>The <i>BaseStationController</i> class serves as the central communication hub for managing interactions between the BS, caching devices and sensors. Its primary responsibilities include sending queries for sensor data, receiving responses, and storing the collected data for further analysis.</p>
<p>Initialization and Setup:</p> <p>During initialization, the <i>init()</i> method performs several actions, being the most relevant one the creation of a JSON file to log the configuration parameters and the results of sensor queries.</p>
<p>Components:</p> <p>Class Structure</p> <p>The <i>BaseStationController</i> class has the following key components:</p> <ul style="list-style-type: none"> • Parameters: The controller initializes several parameters such as the number of caching devices, sensors, cache settings, and query rates. • Action Clients: It establishes action clients to interact with each caching device, enabling asynchronous communication for querying sensor data. • Sensor Position Tracking: It subscribes to odometry data from the sensors to track their positions (just to appropriately know when each sensor is active, and sensing and so to start the querying phase). • Query Management: The class manages the querying process, including the sending of queries and handling of responses.
<p>Query Management</p> <p>The primary functionality of the <i>BaseStationController</i> revolves around managing queries to the caching devices:</p> <ul style="list-style-type: none"> • Sending Queries: The <i>send_queries()</i> method runs in a separate thread to continuously monitor the status of previous queries and initiate new queries at defined time intervals. • Starting Queries: The <i>start_querying()</i> method ensures that all action servers are online and that sensors are positioned before proceeding to send queries. • Sending Goals: The <i>send_goal()</i> method constructs and sends a goal message to each caching device, which includes the sensor ID and parameters to query.
<p>Response Handling</p> <p>The controller also manages responses from the caching devices:</p> <ul style="list-style-type: none"> • Response Callbacks: The <i>goal_response_callback()</i> method processes the responses to the sent goals. If the goal is accepted, it waits for the result, and if rejected, it logs the information.

- **Result Callbacks:** The `get_result_callback()` method handles the results received from the caching devices. It processes the sensor data and logs any errors encountered during the query.
- **Concluding Queries:** The `conclude_query()` method stores the collected results and resets the state for the next query cycle.

Utility Methods

Several utility methods support the main functionalities of the class:

- **`compute_next_goal()`:** It determines the next sensor to query and the parameters to be included in the request.
- **`convert_dict_to_sensordata()`:** It is responsible for converting a dictionary of sensor data into an instance of the `SensorData` class for easier manipulation and storage.
- **`store_result()`:** This method serializes the results of the queries and appends them to the JSON log file created during initialization

4.2.3 Cache Device Controller

Table 4-3 Cache Device Controller

Class: CacheDeviceController

Class Description:

The cache device is controlled by its own class, and is responsible for managing individual devices in the distributed caching system. Each caching device acts as an intermediary between the mobile sensors and the BS, facilitating the storage and retrieval of sensor data while maintaining coverage.

This class integrates message-passing, caching strategies, and action handling to ensure efficient data management and response.

Initialization and Setup:

The `CacheDeviceController` class is initialized with multiple parameters. The device keeps an updated table of its cached data, and implements on top of it a run-time specified replacement policy. The following parameters are fully customizable, and can be changed at run-time:

- **Cache type:** Defines the type of cache replacement policy (e.g., FIFO, LRU, LFU).
- **Cache size:** Specifies the maximum number of entries the cache can hold.
- **Cache expiration:** Sets the maximum age for cache entries, ensuring that outdated sensor data is automatically removed.

Components:

Cache Manager:

Queries from sensors are handled by the Cache Manager, and the results of the requests are logged accordingly to their outcome. When new sensor data is received, it is parsed from JSON format, converted into an ad-hoc `SensorData` object (presented in the next section), and stored in the device's cache (or not, based on the cache policy).

This structure ensures that each caching device in the system can effectively handle its responsibilities, including maintaining accurate sensor data, managing limited storage efficiently, and responding to queries from the BS in a timely manner.

Communication Handler:

Responsible of handling incoming and outgoing messages. Its interface is tied directly to ROS topics and services, ensuring that the managed data can be seamlessly switched to different implementation levels, i.e., from different types of simulated environments, up to real hardware. Actions on the communication handler regarding sensing messages automatically trigger events on the caching table stored.

4.2.4 Cache Manager**Table 4-4 Cache Manager****Class: *CacheManager*****Class Description:**

The *CacheManager* class is designed to handle data storage efficiently, allowing the devices to manage limited memory effectively by implementing various cache replacement policies. It incorporates expiration functionality to ensure that stale data does not occupy valuable storage space.

Initialization and Setup:

The *CacheManager* class is initiated by specifying its cache type, size and expiration time for the entries. Each caching device utilizes a single instance of a *CacheManager* in order to perform its caching task.

Components:**Class Structure**

The *CacheManager* class comprises several key components:

- **Cache Types:** The *CacheType* enumerator defines different cache strategies, including FIFO, RR, LRU, LFU and LOC.
- **Expiration Handling:** The *ExpiringCache* class implements functionality for expiring cache entries based on a time threshold, ensuring that old data is removed.
- **Cache Factory:** The *CacheFactory* class creates instances of the *ExpiringCache* with specific cache types and configurations.

Cache Types

The *CacheType* enumeration provides a structured approach to defining different cache policies. Each policy dictates how entries are managed within the cache:

- **FIFO:** The oldest entries are removed first.
- **RR:** Entries are replaced at random.
- **LRU:** The least recently used entries are evicted first.
- **LFU:** The least frequently accessed entries are discarded.
- **LOC:** The closest entry to the caching device is discarded.

Expiration Handling

The *ExpiringCache* sub-class manages cache entries with a defined expiration time:

- **Initialization:** Upon instantiation, it accepts a cache object and an expiration time in seconds. The cache is initialized using the specified replacement policy and is wrapped with expiration capabilities.

- **Item Management:** The *setitem()* method allows for adding items to the cache. The *getitem()* method retrieves items while checking for expiration:
 - If the item is found and has not expired, it is returned.
 - If the item has expired, it is removed from the cache, and a *CacheExpiredError* is raised.
 - If the item does not exist, a *KeyNotFoundError* is raised.
 - If the item does not exist, but the queried sensor is in range of some caching device, *DelayedDataResult* is raised instead.
- **Cache Maintenance:** The class includes methods to clear the cache and check its size, ensuring effective management of stored data.

Error Handling

The caching system includes robust error handling to manage exceptional situations:

- **CacheExpiredError** is raised when a requested cache entry has expired, indicating that the data is no longer valid. The error will be caught by the caching device, stored and if requested forwarded to the BS.
- **KeyNotFoundError** is raised when attempting to access an entry that does not exist in the cache. Again, the error will be caught and stored by the caching device, and forwarded to the BS if requested.

4.2.5 Sensor Device Controller

Table 4-5 Sensor Device Controller

Class: *SensorDeviceController*

Class Description:

We simulate data produced by the sensors using a specific class, which keeps track of the sensor which produced the data, the time it was produced, and its given validity time. Sensors constantly send produced data at a predefined interval.

Initialization and Setup:

The *SensorDeviceController* class is initialized with several core components, facilitating both data generation and sensor mobility:

- **Sensor Data Generation and Publishing:** Sensors randomly generate data, which is then published over a designated ROS topic.
- **Movement Control:** The sensor can patrol specified target points, moving autonomously while tracking its position and orientation.

The constructor method *init()* sets up the sensor controller by declaring essential parameters and establishing publishers, subscribers, and action servers for sensor data transmission and movement control:

- **Data Publisher:** The *tx_topic* is used by a ROS topic publisher to broadcast sensor data over the network.
- **Patrol Action Server:** This action server allows external nodes (such as the *MovementCoordinator*, presented in the next section) to issue patrol commands, instructing the sensor to move to specific locations.

Components:**Sensor Data Generation and Publishing**

Each sensor generates data at random intervals. The *publish_sensor_data()* method is responsible for generating and publishing this data as a ROS message.

The generated data is then serialized into JSON format and published on the *tx_data* topic. This ensures that each sensor can periodically report its data to caching devices which cache the sensor data for later retrieval.

Position Tracking

The sensor's position are continuously updated through the *store_position()* method, which subscribes to the odometry messages published by the Gazebo simulation, and keeps track of the position and also the orientation of the device within the simulation.

Movement Control and Patrol Action

The sensor's movement is managed by the Patrol action server. The *execute_patrol_action()* method is triggered when a patrol command is received from the *MovementCoordinator* (presented below). The method controls the sensor's rotation and movement to specified target positions.

These methods ensure smooth and controlled movement, allowing the sensor to patrol its environment autonomously.

4.2.6 Movement Coordinator**Table 4-6 Movement Coordinator****Class: *MovementCoordinator*****Class Description:**

The *MovementCoordinator* class is responsible for managing a fleet of mobile sensors within the simulated environment. It controls the sensor movements by submitting patrol tasks to individual sensors and monitors their state. The class also computes the area coverage of the caching devices in order to guide correct sensor movement.

Initialization and Setup:

In the *init()* method, key simulation parameters are declared, and subscriptions are established to track the positions of sensors:

- **Parameters:** The number of sensors, caching devices, sensor range, and field dimensions are declared as parameters, allowing flexibility in different simulation setups.
- **Action Clients:** Each sensor has a corresponding *ActionClient* for submitting patrol tasks, enabling asynchronous control over the fleet's movements.
- **Position Tracking:** The class subscribes to odometry topics to monitor the real-time positions of sensors, storing this information for task submission and area coverage computation.

Components:**Task Submission**

The primary responsibility of the *MovementCoordinator* is to keep the sensors patrolling the covered field. The *patrol_targets()* method continuously resubmits tasks to sensors that are idle.

This method ensures that each sensor is tasked with moving to a new target position once it completes a previous patrol task. The `submit_task()` method assigns the specific patrol target to each sensor. The target is chosen based on the coverage grid calculated from caching devices' positions.

Sensor States

The `SensorState` enumerator defines the possible states of a sensor:

- **IDLE**: The sensor is not currently patrolling or sensing.
- **SENSING**: The sensor is stationary and collecting data.
- **MOVING**: The sensor is in the process of moving to a patrol target.

4.3 Performance Evaluation and Testing

To comprehensively evaluate the performance of the caching strategies implemented, we conducted a set of tests, running different eviction strategies on the same simulation scenarios. These tests focused on analysing various metrics related to cache performance, including HIT/MISS rate, data freshness, and cumulative delay. The results of these tests are reported in this section.

The BS requests data from sensors following a Zipf's distribution [23]. In all the tests, we deploy a set of 4 caching devices, following the specifications mentioned in the chapter before.

We confront a total of 5 eviction strategies:

- **RR**: Random Replacement. One element is replaced at random
- **FIFO**: First In, First Out. Elements are replaced in a queued order.
- **LRU**: Least Recently Used entry is evicted.
- **LFU**: Least Frequently Used entry is evicted.
- **LOC**: Location-based caching, where the closest entry to the caching device is evicted.

We then conduct two sets of experiment:

- Fixed size of cache to a value of 5, varying number of sensors, from 10 to 20.
- Fixed numbers of sensors to 10, varying cache size, from 4 to 8.

Data produced by sensors is constrained by a maximum age, and is set to expire after 10 seconds. All the evaluations are based on a total of 200 queries. In all cases, the distribution of the corresponding metric is shown using a box plot representation. Box limits correspond to the 25 and 75 percentiles, while whiskers represent the range where samples are not considered outliers (1.5 times the interquartile range). The line within the boxes is the median (percentile-50) of the error.

4.3.1 Varying number of sensors

Figure 4-6 shows the results in terms of MISS rate. LFU manages to deliver the best performance in this context. Random Replacement is able to outperform all the other strategies. Performances of the location-based caching are particularly interesting, given the simplicity of the replacement policy, where closest entries are discarded regardless of their use.

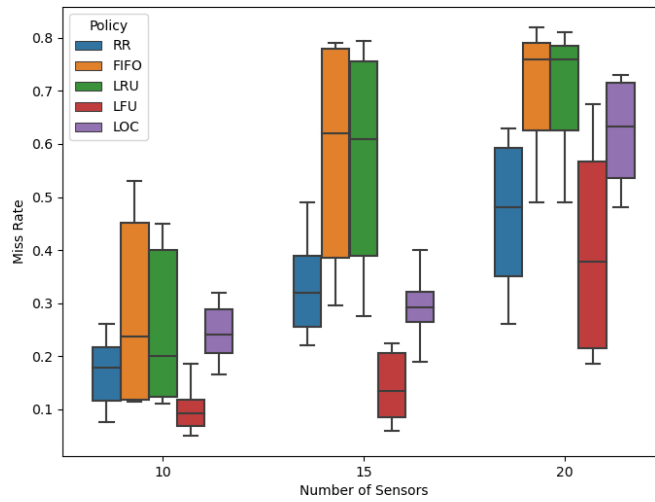


Figure 4-6 Comparison of caching strategies, distribution of miss rate by cache type, varying number of sensors.

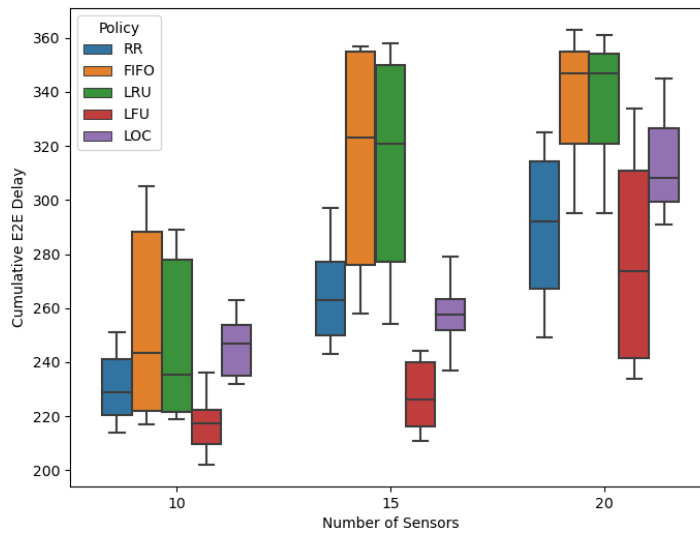


Figure 4-7 Comparison of caching strategies, cumulative E2E delay by cache type, varying number of sensors.

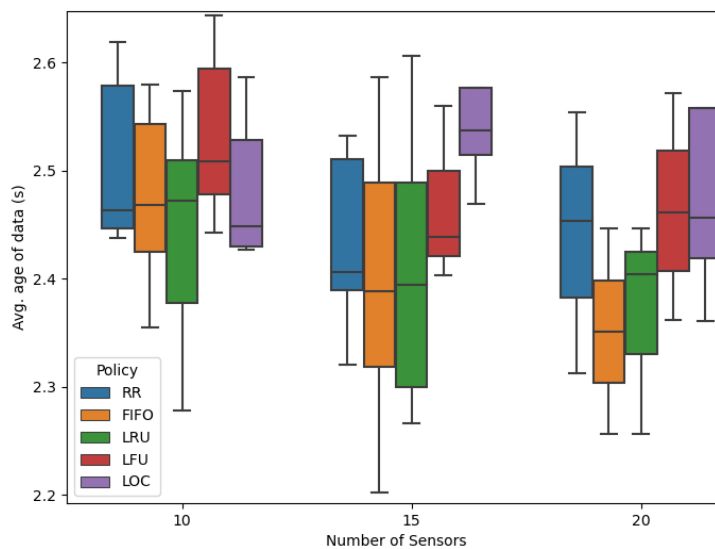


Figure 4-8 Comparison of caching strategies, average age of queried data by cache type, varying number of sensors.

Figure 4-7 shows the results in terms of E2E delay. We value cache hits with a value of 1, while cache misses with a value of 2. Results are in line with the performances displayed in Figure 4-6.

Figure 4-8 shows the results in terms of data freshness. It is interesting to see that even if FIFO and LRU perform worse in terms of MISS rate, they manage to achieve slightly better results in this context. Their use may be interesting in application where the freshness of the data produced by sensors is a priority. Increasing the number of sensors in this context makes data more volatile on the cache, and therefore lowers the average age of the data which is successfully retrieved from the cache.

4.3.2 Varying cache size

Increasing the size of the cache to a value closer to the number of sensors slowly trivializes the studied problem. We can see in Figure 4-9 how the MISS rate drastically decreases as soon as the cache size gets closer to the number of sensors, 10.

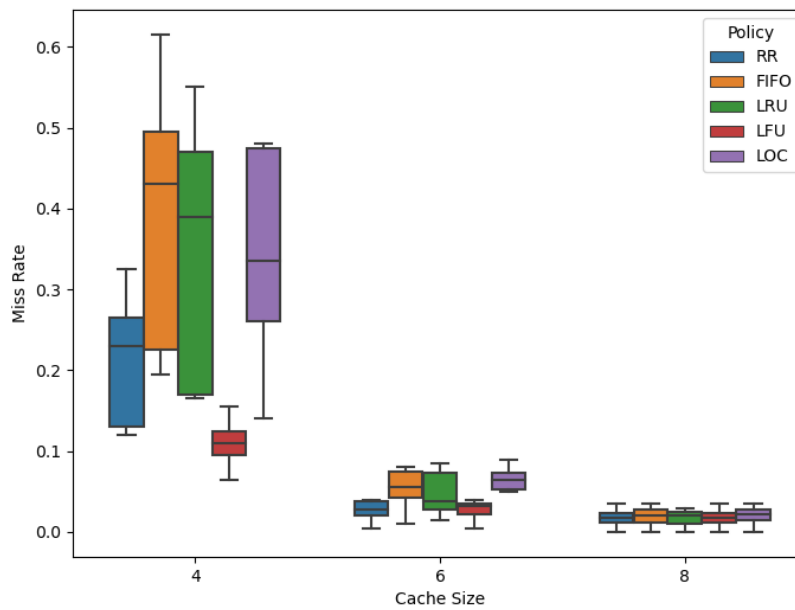


Figure 4-9 Comparison of caching strategies, distribution of miss rate by cache type, varying cache size.

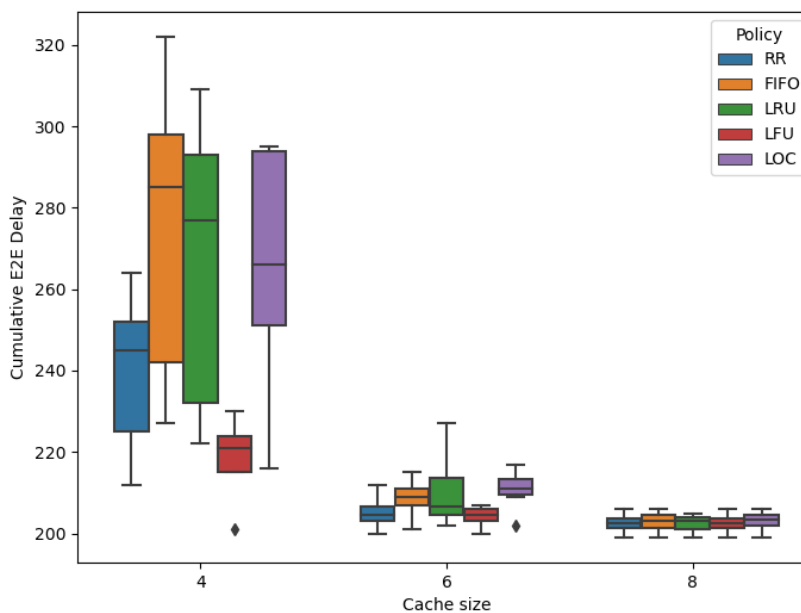


Figure 4-10 Comparison of caching strategies, cumulative E2E delay by cache type, varying cache size.

E2E delay is calculated as in Figure 4-7, so that the delay considered for cache misses is twice that of hits. As can be seen in Figure 4-10, the delay mirrors closely the results in terms of MISS rate, just like for the tests before.

Data freshness, shown in Figure 4-11, tends to stabilize to a fixed value, and the difference between the eviction policies tends to disappear.

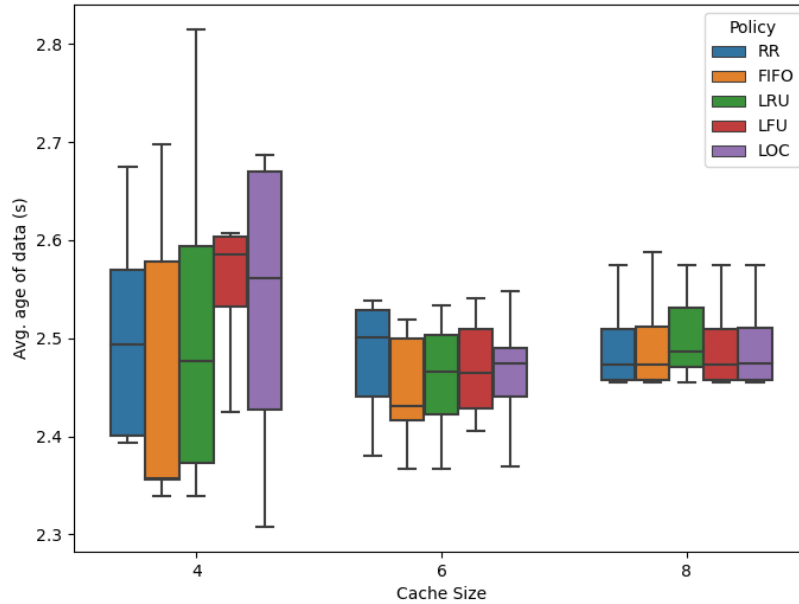


Figure 4-11 Comparison of caching strategies, average age of queried data by cache type, varying cache size.

5 Summary and Conclusions

This document provides study of the SoTA on caching solutions. It also aims to identify the architectural design principles of a wireless edge caching solution needed to support 6G systems. The deliverable identifies a challenging network scenario taken from Use Case #1 that was described in deliverable D2.1 [4] of the 6G-SENSES project, in which the use of wireless edge sensing mechanisms can improve the network performance.

To study, design and evaluate different caching algorithms, we developed a detailed simulation framework that includes network communication between devices and sensors. The report describes this framework and presents the initial performance results achieved by testing several conventional caching strategies used in the literature under various scenarios. Different key performance metrics, such as HIT/MISS ratios, E2E delay, and data freshness are evaluated.

The work done until now mainly focused on the development of the simulation framework. Leveraging this environment, we plan to test and validate different scenarios and request patterns, considering localization information.

Therefore, we intend to build upon the work performed until now by designing additional algorithms that leverage location-related information to create novel location-based caching mechanisms. These approaches will be evaluated to assess their impact on caching performance compared to standard caching mechanisms.

A more detailed and precise architectural description of the Wireless Edge Caching mechanism, together with a thorough performance analysis, will be included in deliverable D4.2.

6 References

- [1] D. Robinson, "Content Delivery Networks: Fundamentals, Design, and Evolution." *Wiley*, September 2017.
- [2] Y. Fu, Y. Zhang, Q. Zhu, H.-N. Dai, M. Li, and T. Q. S. Quek, "A new vision of wireless edge caching networks (wecns): Issues, technologies, and open research trends", in *IEEE Network*, 38(1):247–253, 2024.
- [3] T. X. Vu, E. Baştuğ, S. Chatzinotas, and T. Q. S. Quek, "Wireless Edge Caching: Modeling, Analysis, and Optimization", in *Cambridge University Press*, 2021.
- [4] 6G-SENSES deliverable D2.1, "Report on 6G-SENSES use cases, requirements and KPIs, and key technological advancements", September 2024, https://6g-senses.eu/wp-content/uploads/2024/10/2024-09-30-6G-SENSES_Deliverable_2_1_vf.pdf
- [5] Y. Ren, X. Zhang, T. Wu, and Y. Tan, "In-network caching for the green internet of things", in *IEEE Access*, 9:76413–76422, 2021.
- [6] K. S. Tharakan, B. N. Bharath, V. Bhatia, J. Nebhen, M. Dobrovolny and T. Ratnarajah, "Wireless Edge Caching and Content Popularity Prediction Using Machine Learning," in *IEEE Consumer Electronics Magazine*, vol. 13, no. 4, pp. 32-41, July 2024, doi: 10.1109/MCE.2022.3160585.
- [7] X Wei, J Liu, Y. Wang, C. Tang, and Y. Hu, "Wireless edge caching based on content similarity in dynamic environments", in *Journal of Systems Architecture*, 115:102000, 2021.
- [8] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang and X. Shen, "Content Popularity Prediction Towards Location-Aware Mobile Edge Caching," in *IEEE Transactions on Multimedia*, vol. 21, no. 4, pp. 915-929, April 2019, doi: 10.1109/TMM.2018.2870521.
- [9] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," in *IEEE Transactions on Information Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.
- [10] Z. Zhang, C. -H. Lung, X. Wei, M. Chen, S. Chatterjee and Z. Zhang, "In-Network Caching for ICN-Based IoT (ICN-IoT): A Comprehensive Survey," in *IEEE Internet of Things Journal*, vol. 10, no. 16, pp. 14595-14620, 15 Aug.15, 2023, doi: 10.1109/JIOT.2023.3274653.
- [11] C. -H. Nguyen et al., "Encrypted Data Caching and Learning Framework for Robust Federated Learning-Based Mobile Edge Computing," in *IEEE/ACM Transactions on Networking*, vol. 32, no. 3, pp. 2705-2720, June 2024, doi: 10.1109/TNET.2024.3365815.
- [12] S. Vural, N. Wang, P. Navaratnam and R. Tafazolli, "Caching Transient Data in Internet Content Routers," in *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1048-1061, April 2017, doi: 10.1109/TNET.2016.2616359.
- [13] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta and D. Sabella, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," in *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657-1681, thirdquarter 2017, doi: 10.1109/COMST.2017.2705720.
- [14] G. Orsini, D. Bade and W. Lamersdorf, "Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading," in *8th IFIP Wireless and Mobile Networking Conference (WMNC)*, Munich, Germany, 2015, pp. 112-119, doi: 10.1109/WMNC.2015.10.

- [15] M. Mehrabi, D. You, V. Latzko, H. Salah, M. Reisslein and F. H. P. Fitzek, "Device-Enhanced MEC: Multi-Access Edge Computing (MEC) Aided by End Device Computation and Caching: A Survey," in *IEEE Access*, vol. 7, pp. 166079-166108, 2019, doi: 10.1109/ACCESS.2019.2953172.
- [16] T. X. Tran, A. Hajisami, P. Pandey and D. Pompili, "Collaborative Mobile Edge Computing in 5G Networks: New Paradigms, Scenarios, and Challenges," in *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54-61, April 2017, doi: 10.1109/MCOM.2017.1600863.
- [17] K. Yu et al., "Information-Centric Networking: Research and Standardization Status," in *IEEE Access*, vol. 7, pp. 126164-126176, 2019, doi: 10.1109/ACCESS.2019.2938586.
- [18] H. Zhang, H. Zhang, B. Di, and L. Song, "Holographic integrated sensing and communications: Principles, technology, and implementation," in *IEEE Communications Magazine*, vol. 61, no. 5, pp. 83–89, 2023.
- [19] A. M. Elbir, K. V. Mishra, M. R. B. Shankar, and S. Chatzinotas, "The rise of intelligent reflecting surfaces in integrated sensing and communications paradigms," in *IEEE Network*, pp. 1–8, 2022.
- [20] S. P. Chepuri et al., "Integrated sensing and communications with reconfigurable intelligent surfaces: From signal modeling to processing," in *IEEE Signal Processing Magazine*, vol. 40, no. 6, pp. 41–62, 2023.
- [21] Open Robotics, "ROS: The Robot Operating System". Retrieved December 18, 2024, from <https://ros.org/>
- [22] Open Robotics, "Gazebo Simulator". Retrieved December 18, 2024, from <https://gazebosim.org/>
- [23] D. N. Serpanos, G. Karakostas and W. H. Wolf, "Effective caching of Web objects using Zipf's law", in *IEEE International Conference on Multimedia and Expo (ICME2000)*. Proceedings. Latest Advances in the Fast Changing World of Multimedia (Cat. No.00TH8532), New York, NY, USA, 2000, pp. 727-730 vol.2, doi: 10.1109/ICME.2000.871464.
- [24] J. Yick, B. Mukherjee, D. Ghosal, "Wireless sensor network survey," *Computer Networks*, Volume 52, Issue 12, 2008, Pages 2292-2330, ISSN 1389-1286, <https://doi.org/10.1016/j.comnet.2008.04.002>

7 Acronyms

Acronym	Description
AI	Artificial Intelligence
AP	Access Point
AR	Augmented Reality
BS	Base Station
CDN	Content Delivery Network
CF	Cell-Free
CF-mMIMO	Cell-Free massive MIMO
CPU	Central Processing Unit
C-RAN	Cloud Radio Access Network
E2E	End-to-End
FIFO	First In, First Out
FL	Federated Learning
HRLLC	Hyper Reliable Low Latency Communication
ICN	Information-Centric Networking
IMT	International Mobile Telecommunications
IoT	Internet of Things
ISAC	Integrated Sensing and Communication
JSON	JavaScript Object Notation
LFU	Least Frequently Used
LOC	Location
LRU	Least Recently Used
MEC	Multi-access Edge Computing
MIMO	Multiple-Input Multiple-Output
ML	Machine Learning
mMIMO	Massive MIMO
MRU	Most Recently Used
NGMN	Next Generation Mobile Networks
NOP	Network Operator
O-RAN	Open RAN
RAN	Radio Access Network
RIS	Reconfigurable Intelligent Surface
ROS	Robot Operating System
RR	Random Replacement
RRH	Radio Remote Head
SMO	Service Management and Orchestration
SNS JU	Smart Networks and Services Joint Undertaking
SP	Service Provider
UAV	Unmanned Aerial Vehicle

UE	User Equipment
V2X	Vehicle-to-Everything
VR	Virtual Reality
WAT	Wireless Access Technology
Wi-Fi	Wireless-Fidelity
XR	eXtended Reality