Vrije Universiteit Amsterdam
Future Network Services

Bachelor Thesis

Network Simulation for AI in the Digital Continuum

Author: Gleb Mishchenko (2766204)

1st supervisor: Jesse Donkervliet
2nd reader: Prof. Dr. Ir. Cav. Alexandru Iosup
daily supervisor: Matthijs Jansen

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

July 27, 2025

Abstract

Ultra-low-latency and high-throughput mobile networks are essential for a large number of emerging mission-critical applications, including autonomous vehicles or virtual reality. As these applications at least partially rely on computationally heavy machine learning algorithms, task offloading within cloud or edge computing paradigms often becomes essential to fulfill application compute requirements, making network performance critical for achieving low response latency. To support this low-latency task offloading, starting from 5G and continuing into 6G, cellular networks are becoming increasingly more integrated with the cloud computing forming a **digital continuum**. Digital continuum offers near-infinite deployment possibilities, each with a unique set of **compute** and **network** trade-offs. Therefore, performance evaluation within this heterogeneous infrastructure requires a careful consideration of both **compute** and **network** components. There is; however, no tool that can be used to accurately explore the trade-offs and make informed decisions regarding the digital continuum deployments and the infrastructure itself. In this paper, we cover this gap. We design and implement a **6G testbed** with state-of-art compute and network components empowering informed exploration of digital continuum deployments and infrastructure itself. We publish the testbed as an extendable **open-source artifact**. Allowing the users to use the testbed with a wide range of network configurations, we design a method to collect the performance data from existing network setups and apply it to construct an **open-source cellular network trace archive** containing the performance data of state-of-practice Dutch network infrastructure. Finally, we evaluate the performance of state-of-practice network and compute infrastructure with a prominent 6G use-case - ML applications - and show that 6G goals can be fulfilled only though a careful consideration and improvements to both **compute** and **network** infrastructures.

Contents

1

Introduction

1.1 Context

Over the past decade, digital infrastructure has become essential for a large number of mission-critical services. Those services range from energy to transportation, education, finance, and many others. In a recent report, the Dutch Ministry of Economic Affairs and Climate outlined a development of digital infrastructure as one of the core policies (1). Mobile networks is an essential component of modern digital infrastructure, allowing for digital services to be run without restricting a user's mobility to a singular wireless access point. Mobile networks are used by over 4.6 billion people globally (2), and in some developing areas with no home-wired internet access, mobile networks constitute a primary means of accessing digital services. In the Netherlands, mobile networks are so ubiquitous that every Dutch citizen has on average 1.4 SIM cards (3). Emerging low-latency applications, such as augmented reality or autonomous vehicles, are therefore likely to be used through mobile networks. Furthermore, for some applications, such as autonomous vehicles or augmented reality, mobile networks are essential in fulfilling their mobility requirements - application semantics assumes that the endpoint devices stay portable and thus must not be restricted to a single wireless access point. These novel applications pose increasingly strict latency and bandwidth requirements, with end-to-end latency, for some, reaching as low as 10 ms (4) and requiring minimal jitter. This is a very challenging goal considering the inherent high jitter and high latency associated with wireless transmission. Avoiding the network, by performing computation on the endpoint device, becomes increasingly difficult as the computational power required by some deep learning models is already orders of magnitude higher than what endpoint devices can offer (5). With computational demands for deep learning model training doubling every 5.9 months (6), the gap between

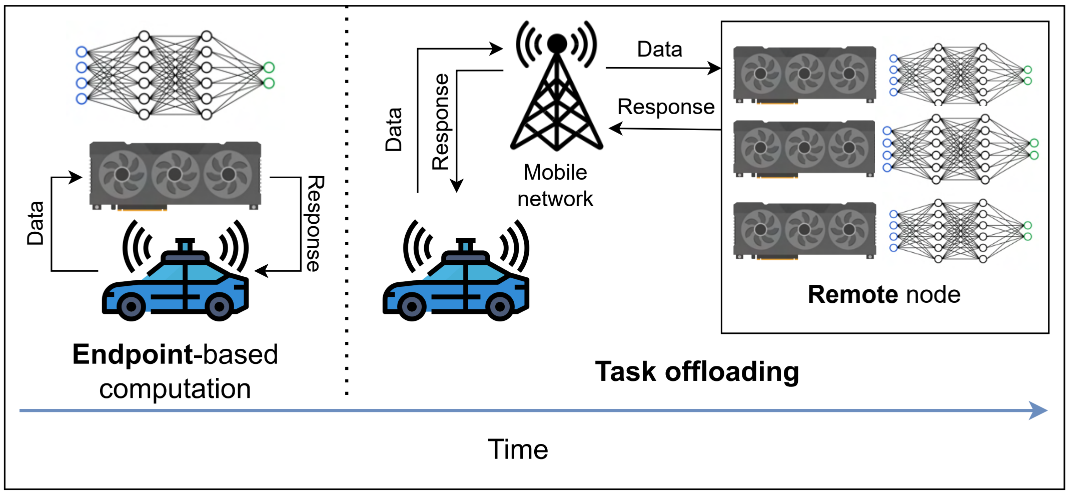

Figure 1.1: Transition to task offloading architecture as a result of an increase in co33mputational demands.

the endpoint hardware capabilities and state-of-the-art machine learning algorithms continues to increase. Due to this, the novel AI-based applications are likely to employ the task offloading architecture within the cloud computing paradigm, as shown in Figure 1.1. There, a remote server connected to the endpoint via a network hosts a computationally-heavy back-end, benefiting from higher compute resources available. To accommodate this low-latency task offloading, starting with 5G, and continuing in 6G, cellular networks are becoming more integrated into cloud and edge computing paradigms, creating a **digital continuum**. Digital continuum is an ecosystem of interconnected heterogeneous devices, each offering a unique set of **compute** and **network** trade-offs (7). With more distant computing services, such as cloud, the available compute capacity is likely to increase; however, as the geographical distance to those nodes increases, the network access latency increases as well.

1.2 Problem Statement

With the variety of digital continuum infrastructure configurations and a number of emerging low-latency applications, each with its own set of compute and networking requirements, digital continuum performance evaluation becomes increasingly difficult. As with the increase in computational complexity of ML algorithms, the compute becomes an increasingly significant component, digital continuum performance evaluation must consider both **network** and **compute**. Mathematical modeling of an end-to-end system is not feasible due to non-linear interaction of system components, which includes hardly predictable phenomena such as temporal network outage or network hand-offs impacting the end-to-end system performance. An additional challenge comes from high mobile network jitter inherent to wireless communication. To enable 6G stakeholders to make informed decisions on digital continuum infrastructure, a 6G testbed is needed. This 6G testbed should enable a **rapid** iteration over digital continuum network and compute components to find the optimal configuration while taking into account the application's unique network and compute requirements. There are two types of testbeds, hardware testbeds and software testbeds. Hardware testbeds, while offering more accurate results, are currently unfeasible due to the current lack of 6G hardware. In anticipation of 6G hardware, the 6G software testbed that offers a **rapid**, and **flexible** exploration ground, allowing for performance evaluation without binding the network to a specific hardware, is essential for an early-stage 6G development. There is, however, currently no 6G testbed that stakeholders can use to make informed decisions regarding the trade-offs offered by different components of the digital continuum infrastructure. In this paper, we address this problem by developing a 6G testbed with **jitter-aware** network simulation capabilities, which capture the inherent long- and short-term performance variability characteristic of wireless networks, and enable **rapid** and **flexible** exploration of digital continuum infrastructure configurations. We publish this testbed as an **easy-to-use** and further **extendable** by arbitrary use-cases open-source artifact on GitHub. Enabling the users to conduct performance evaluation with a variety of network setups, we construct a method to measure the performance of existing networking setups, and apply it to create a cellular network trace archive - an open-source collection of network performance data - containing the performance traces on the state-of-practice Dutch network infrastructure. Conducting an evaluation, we use the 6G testbed and collected network performance traces to determine the impact of the cellular network on ML applications. We find that fulfilling the 6G goals will require careful consideration and a number of improvements to both **compute** and **network** components of digital continuum infrastructure.

1.3 Research Questions

Main RQ: How do we facilitate the evaluation and exploration of 6G-enabled applications in the digital continuum?

RQ1 *How to design a 6G testbed?* To enable the digital continuum, trade-off exploration, we design a tool - 6G testbed. The tool contains both **compute** and **network** components, accurately capturing the digital continuum performance.

RQ2 *How to implement a 6G testbed?* Enabling the digital continuum trade-off exploration for 6G stakeholders, we implement a tool - 6G testbed - and publish it as an open-source artifact (GitHub) that can be used for arbitrary user application and digital continuum configuration out-of-the-box.

RQ3 *How to characterize network performance?* Enabling users to test the performance of various network configurations, we *(i)* construct a method to collect performance data of real-world network setups, *(ii)* use it to construct an open-source cellular network trace archive, and *(iii)* characterize the collected network traces outlining several system-level performance trends and limitations.

RQ4 *What is the impact of cellular networks on ML applications running in the digital continuum?* Using a constructed testbed and collected network performance traces, we, evaluate the impact of cellular networks on ML applications under different conditions, including: cellular network technology, offloading location, executed workload, and network load.

1.4 Research Methodology

(**High-level overview**) To answer the main research question - "*How do we facilitate the evaluation and exploration of 6G-enabled applications in the digital continuum?*" - *(i)*, we begin by answering **RQ1** - "How to design a 6G testbed?" - and **RQ2** - "How to implement a 6G testbed?" - by designing *(i)* and implementing the 6G testbed. Then, answering the **RQ3** - "How to characterize the performance of the digital continuum networking?" - we construct a method for collecting network performance data for existing network setups and use this method to construct an open-source cellular network trace archive and outline the system-level limitations of modern networking infrastructure. Finally, answering the **RQ4** - "What is the impact of cellular networks on ML applications running in the digital continuum?" - we validate the 6G testbed design by evaluating, using the collected performance traces, the state-of-practice network infrastructure with a prominent 6G use-case - ML applications.

(**RQ1**) To design a 6G testbed, we follow the AtLarge design process (8) continuously iterating through *(i)* problem analysis, *(ii)* design requirements, *(iii)* prototype, and *(iv)* validation. Designing 6G compute component, we base ourselves of the peer-reviewed SPEC-RG compute continuum reference architecture (7). Constructing the design of **jitter-aware** and **flexible** network simulation, we taxonomize end-to-end network path into core and last-mile network paths, and represent each in terms of their **network-layer** performance.

(**RQ2**) To implement a 6G testbed, we follow a "A Philosophy of Software Design" by John Ousterhout (9), focusing on implementing deep modules with simple interfaces. Implementing the 6G testbed, we leverage and integrate peer-reviewed compute continuum benchmark (10) and a state-of-the-art **jitter-aware** network simulation (11). Following the principles of open science, we publish the implementation as an open-source GitHub artifact.

(**RQ3**) Constructing the method for performance collection, we follow the end-to-end network path taxonomy constructed in answering the **RQ1**. Collecting a **jitter-sensitive** mobile network performance, we use state-of-the-art tools that produce traces at a granularity of individual packet transmission time, accurately capturing the network performance variability. We validate the method by collecting performance traces of the existing LTE and 5G Dutch networking infrastructure. For the core network, we collect the latencies to deployment locations following the taxonomy specified by Ali-Eldin et al. (12). Following the principles of open science, we publish the performance traces as an open-source cellular network trace archive. We characterize the performance of collected traces and identify system-level performance trends and limitations.

(**RQ4**) To evaluate the testbed design and implementation, we use the collected traces to evaluate the effect of the state-of-practice networking on a prominent 6G use case: AI-based workloads. Conducting the evaluation, we design and execute a number of experiments based on state-of-the-art field practices and design methodologies (13) and find the effect of changing *(i)* cellular network technology, *(ii)* deployment location, *(iii)* workload, and *(iv)* network load.

1.5 Thesis Contributions

This thesis presents a number of contributions (C):

(C1) Answering the **RQ1** and **RQ2**, we are the **first** to design, implement, and publish as an open-source artifact a general-purpose 6G software testbed. In comparison to existing compute continuum testbeds (10), this testbed includes the **jitter-aware** network simulation, allowing an accurate exploration of both network and compute components. We publish the testbed as an easy-to-use and extendable open-source artifact on (GitHub)

(C2) Answering the **RQ3**, we the first to compile and open-source cellular network trace archive containing the state-of-practice (as of Jul. 2025) LTE and 5G network performance traces. We are also the first to characterize those traces that outline system-level performance trends and limitations.

(C3) Answering the **RQ4**, we are the **first** to perform an experiment-based evaluation of state-of-practice digital continuum infrastructure on a prominent 6G use-case - AI-based applications. Based on the results we are the first to derive a number of several experimental-based insights and challedges for 5G to 6G transition.

1.6 Societal Relevance

This work goes in line with the Dutch Computer Systems Manifesto (14) and addresses the goals of **manageability** and **responsibility** of ICT infrastructure. The ICT infrastructure became essential for a large share of Dutch GDP and is a substantial requirement for many jobs (14). The dependence of our society on the ICT infrastructure makes the performance and reliability of that infrastructure an important concern for Dutch society. The result of this thesis - 6G testbed - is a tool allowing for testing different configurations of ICT infrastructure and testing deployments on top of ICT infrastructure. The 6G testbed is a tool for the **management** of ICT infrastructure with the goal of optimal **performance** and **reliability**.

1.7 Plagiarism Declaration

This thesis work is my own work, is not copied from any other source (person, Internet, or machine), and has not been submitted elsewhere for assessment.

1.8 Thesis Structure

The thesis follows a linear structure and is conceptually divided into two parts as shown in Figure 1.2. Preceding the thesis contributions is the ❷ background section containing

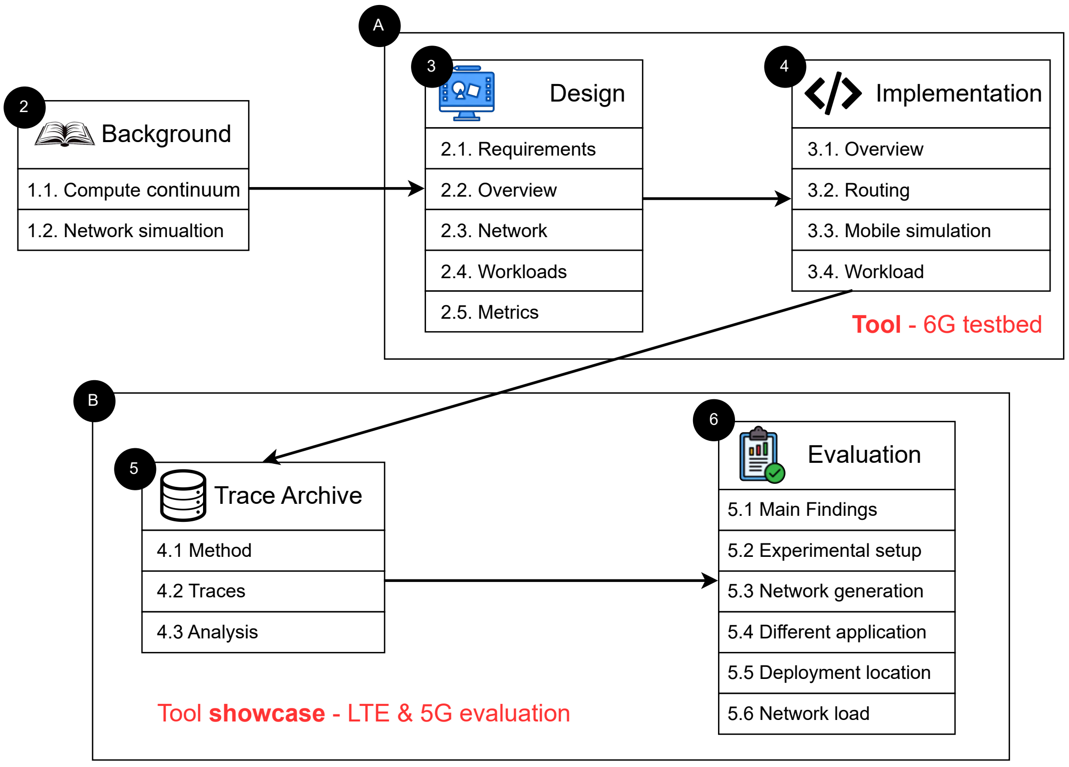

Figure 1.2: Thesis structure

the preliminary background for the reading. The sections describe Continuum - a peer-reviewed compute continuum benchmark (10) - and MahiMahi - a record-replay tool for accurate emulation of wireless link performance. The Ⓐ first part of the thesis contains ❸ Chapter 3 - **Design** (3) - and ❹ Chapter 4 - **Implementation** (4). This part focuses on the design and implementation choices of the 6G testbed itself. The ❷ Design chapter focuses on the requirements and describes the high-level design choices satisfying the posed requirement. The ❸ Implementation chapter focuses on the instruments used and the implementation choices employed to fulfill the design. The Ⓑ second part of the thesis focuses on the application of the 6G testbed, showcasing how the testbed can be used for the modern networking infrastructure. The second part of thesis contains ❺ Chapter 5 - **Data Collection** (5) - and ❻ Chapter 6 - **Evaluation** (6). The ❺ Chapter 5 focuses on the method and the network performance traces collected. The ❻ Chapter 6 uses the 6G testbed and the collected performance traces to evaluate the performance limitations of both compute and network components from the perspective of real-world applications.

This chapter as well, outlines the main points of actions and lines of research for the 6G.

2

Background

In this section, we outline the preliminary context of the paper. We begin by introducing a compute continuum reference architecture and benchmarking tools in Section 2.1, and then move on to network simulation tools - tc and Mahimahi - in Section 2.2.

2.1 Compute Continuum

This section provides an introduction to the existing work on the compute continuum, which we later use in our design and implementation of the 6G testbed. Our 6G testbed design is based on SPEC-RG, a peer-reviewed compute continuum reference architecture (7) that is shown in Figure 2.1. The architecture outlines the responsibilities of three different device types - **endpoint**, **edge**, and **cloud** - and includes the application, data pre-processing, infrastructure, resource management, etc. On top of that, build a peer-reviewed compute continuum benchmark - Continuum (10) - which we use in the implementation of the 6G testbed. Continuum allows rapid, extendable benchmarking of user applications on user-defined compute continuum infrastructure configurations. Continuum allows changing the number of endpoints, cloud, and edge nodes, and, for each node, allows changing the number of CPU cores, amount of memory, and their quota. For resource management, Continuum uses state-of-the-practice Kubernetes, and, ensuring benchmark extensibility for future use cases, it runs the applications as Docker containers, ensuring compatibility with arbitrary user software and the set of dependencies.

2.2 Network Simulation

This section provides the introduction into existing tools for network simulation employed in the implementation of 6G testbed - **traffic control** (tc) and **MahiMahi**. **TC** (15) is a

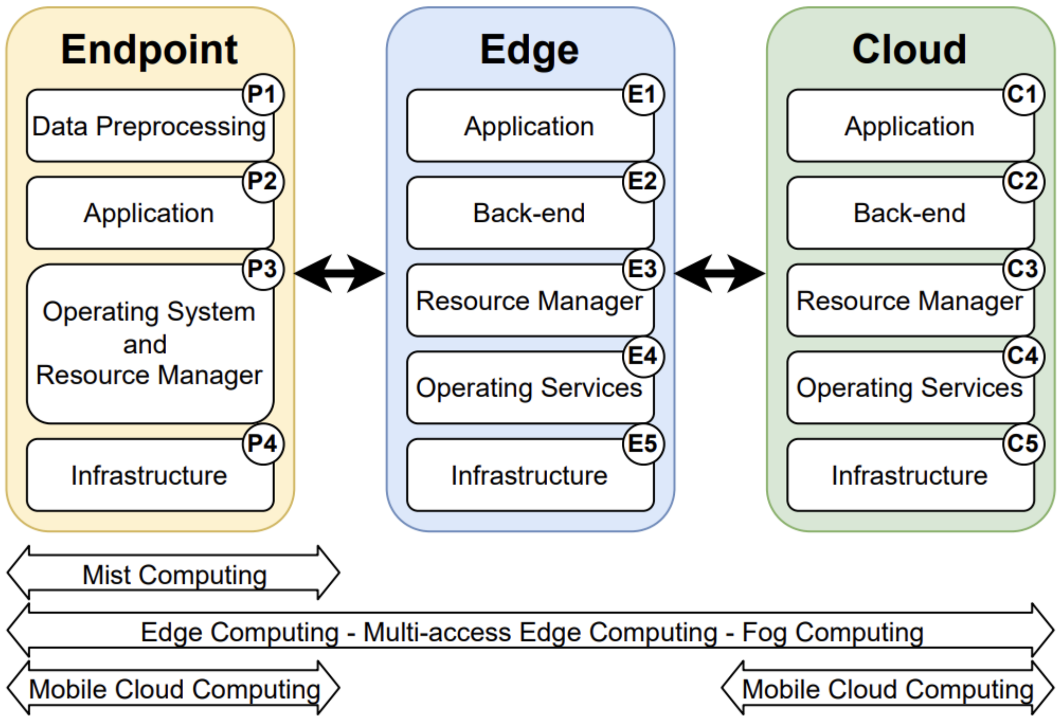

Figure 2.1: SPEC-RG reference architecture (7).

built-in Linux tool allowing the simulation of network latency and bandwidth defined via some statistical distribution such as normal, uniform, or Pareto. **TC** is already used in Continuum (10) to simulate the network latency, and, in our implementation, we use it to simulate static core network latency. The biggest limitation of **tc** is its inability to simulate dynamic network latency and bandwidth, which does not follow under any aforementioned distributions. Therefore, high-jitter links such as wireless links cannot be accurately simulated using **tc**. **MahiMahi** (11) is a record-replay tool for network simulation which we use to simulate the mobile network. Of our particular interest is MahiMahi's **mm-link**, which simulates the network at per-packet level, accurately reflecting the link performance at a millisecond granularity. MahiMahi is configured via uplink and downlink traces, with each containing data on packet delivery opportunities, millisecond-granularity timestamps, at which packets can cross successfully the mobile network hop.

3

Design

In this section, we design a 6G testbed and answer an **RQ1** - "*How to design a 6G testbed?*". The primary 6G testbed goal is to allow **rapid** performance evaluation under **flexible** digital continuum configurations. With an ever-increasing coupling of telecommunications into cloud and edge computing paradigms, combined with increasing compute demands by machine learning algorithms, the **flexible** and **rapid** digital continuum performance evaluation encompasses both **network** and **compute** components. 6G testbed offers value to 6G stakeholders by enabling the exploration and informed decisions of digital continuum infrastructure trade-offs with the goal of finding optimal configurations. We begin the section by identifying, based on the problem description in Section 1.2, 6G testbed requirements in Section 3.1. Based on derived requirements, we propose a 6G testbed design, which is summarized in Section 3.2. Later sections - Section 3.3 and Section 3.4 - provide a further description of novel 6G testbed components - network simulation and ML-based workloads.

3.1 Requirements Analysis

With a near-infinite number of digital continuum infrastructure possibilities, a tool that allows for a **rapid** and **flexible** performance evaluation to find optimal configurations becomes critical. As novel applications, such as autonomous vehicles or VR applications, pose increasingly stricter latency requirements and rely, at least partially, on increasingly computationally heavy ML algorithms, making compute a considerable part of end-to-end latency, a tool for digital continuum performance evaluation must incorporate both **compute** and **network** components. With digital continuum offering near-infinite deployment possibilities, each with a unique set of compute and network trade-offs, the tool must remain **flexible**, allowing the final user to replicate the real-world conditions of a particular deployment configuration. To replicate compute conditions, the user must be able to **configure** the hardware conditions of both the endpoint and cloud, and edge nodes by modifying parameters such as the number of CPU cores or the amount of memory. Replicating the geographical distribution of the remote node, the user should, as well, be able to represent the distance to the deployment location - **edge**, **cloud** - through the network latency experienced when traversing the core network path between the endpoint and the remote node. With a variety of availiable last-mile network designs and, yet, no clear vision on the upcoming 6G standard, the testbed must keep the mobile network **flexible** and stay abstract from its design or implementation details. To reflect the real-world mobile network jitter and accurately represent the edge cases, such as temporary network outages due to increased congestion, frequency band change, or network handovers, the user should be able to configure the mobile network performance at a millisecond level of granularity. For the testbed to have a high discriminative power for real-world use cases, the results must be **reproducible** and based on standardized workloads relevant for 6G use-cases. Workloads should include those yielding high and low network bandwidth requirements, testing for both bandwidth and latency. For further extensibility, we require workloads to stay flexible by allowing users to submit arbitrary containerized workloads.

Therefore, based on our problem definition, we pose the following set of 6G testbed requirements (TR):

(**TR1**) **Configurable CPU core count and memory**. The testbed must allow the user to replicate real-world node hardware capabilities by configuring the node CPU count and amount of memory.

(**TR2**) **Configurable distance to a remote node**. The testbed must allow the user to configure the distance to a remote compute node by configuring a latency experienced when traversing a core network path from the endpoint to a remote compute node.

(**TR3**) **Implementation-agnostic network simulation**. The last-mile network simulation must rely solely on the network layer performance.

(**TR4**) **Millisecond-granularity network configuration**. To reflect the real-world mobile network jitter and the edge cases like temporary network outage, increased congestion, or frequency band switch, the user must be able to configure mobile network performance at millisecond granularity.

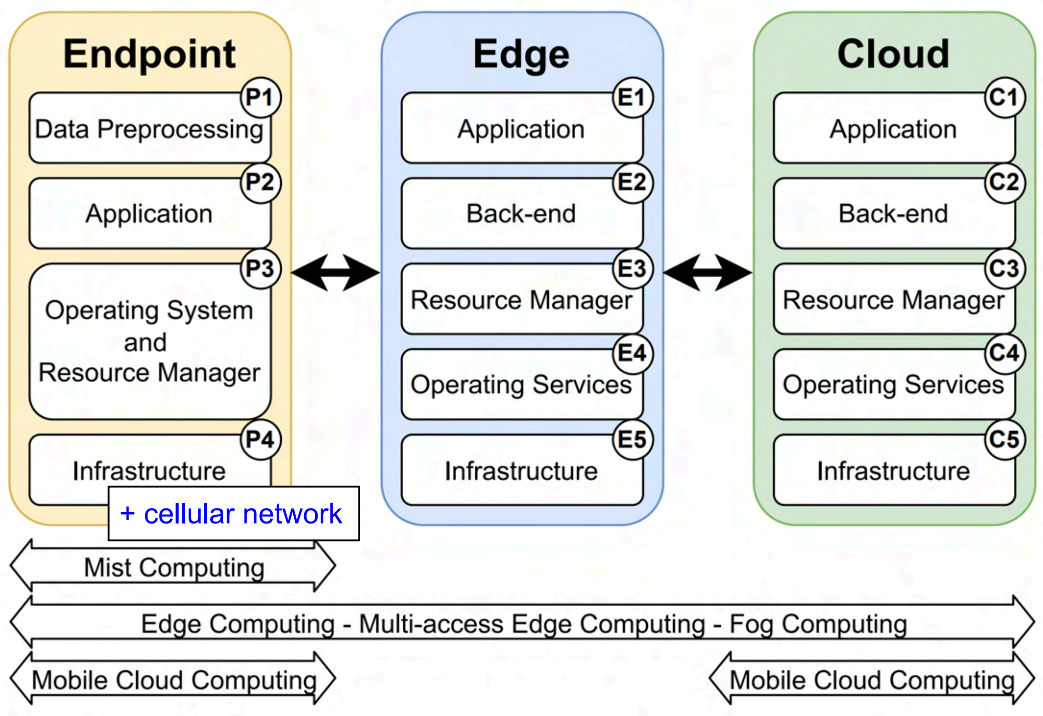

Figure 3.1: High-level design overviews

(**TR5**) **Reproducibility**. To reliably evaluate the end-to-end performance of 6G design and allow the comparison between various designs, the evaluation results must be reproducible.

(**TR6**) **Standardized and relevant use-cases**. Ensuring the relevance of testbed results for real-world use-cases, the workloads must be based in prominent 6G use-cases.

(**TR7**) **Containerized workloads**. To allow users to further extend workloads with their use-cases with unique set of dependencies, the workloads must be executed as containerized applications.

(**TR8**) **Collected network and compute metrics.** To allow the identification of performance limitations of separate system components, the testbed should collect both compute and network metrics.

3.2 Design Overview

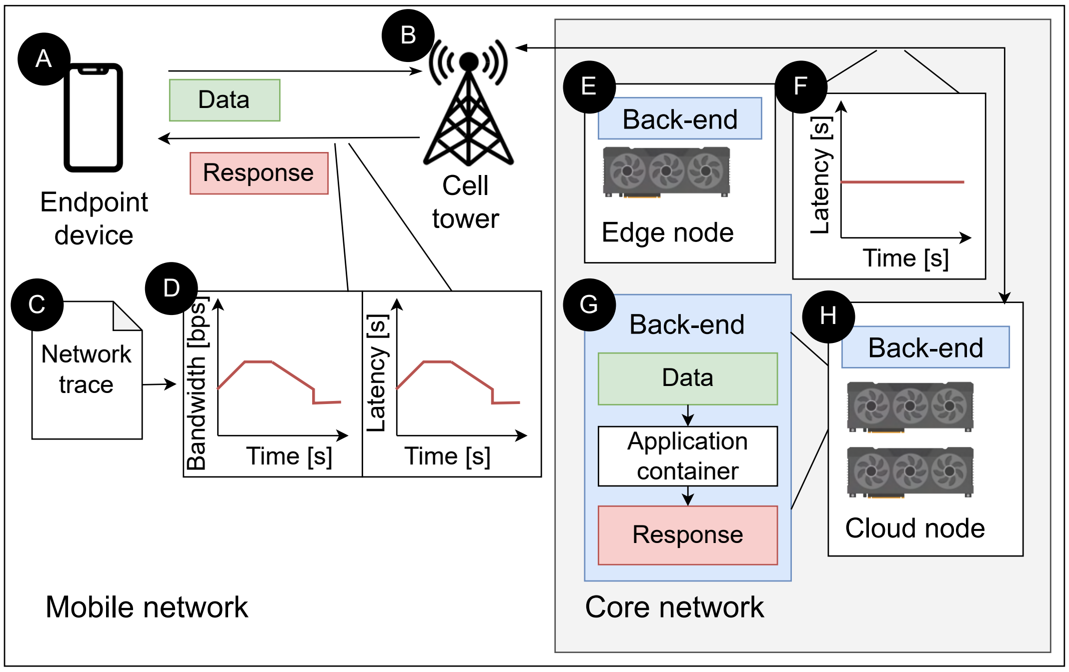

Figure 3.2: Design overview.

This section defines a high-level design that fulfills the 6G testbed requirements defined in Section 3.1. To fulfill the requirements (**TR1**), (**TR2**), and (**TR3**), and allow the maximum flexibility supporting the goal of **rapid** and **flexible** evaluation of digital continuum performance, we take the conceptual approach of designing a software testbed with configurable compute and networking components. An example configuration of 6G testbed compute and network components is shown in Listing 1. A high-level design overview, building on top of the peer-reviewed compute continuum benchmark in shown Figure 3.1. Our 6G testbed supports endpoint, edge, and cloud deployments, extending the compute continuum benchmark functionality with a **jitter-aware** network simulation, configuring the network performance at a **millisecond**-level granularity. We dive into the design of **jitter-aware** network simulation in Figure 3.2. Meeting the requirements (**TR3**) and (**TR4**), we allow the user to submit Ⓒ network trace where every line represents a **millisecond** timestamp when an maximum transmission unit (MTU) sized packet (1500 bytes (11)) can cross the mobile network link. This abstraction reflects only a final performance perceived by the endpoint network layer and follows modern medium access control time division multiplexing-based mechanisms used by modern 5G infrastructure (16), which al-

```
[compute]
cloud
    count=1
    cpu_count=2
    ram=2G
edge
    count=1
    cpu_count=2
    ram=1G
endpoint
    count=1
    cpu_count=1
    ram=512M
[network]
core
    edge_deployment: base
    cloud_deployment: regional
mobile
    up_trace = /Verizon/LTE.up
    down_trace = /Verizon/LTE.down
```

Listing 1: Configuration file structure

low the endpoint to hog the channel for at most an MTU-sized frame transmission. As a result of observed, on modern cellular networking infrastructure, performance disparity between the uplink and downlink (17), the user can submit two traces - uplink and downlink traces. The traces are **re-playable**, ensuring the **reproducibility** of results and meeting the requirements (**TR5**). Meeting the requirement (**TR2**), we allow the users to configure the core network latency experienced when transmitting from Ⓑ cell tower to either Ⓔ edge or Ⓗ cloud nodes. Finally, to meet the requirement (**TR7**), we allow the end use to extend the workloads by submitting a containerized applications to be executed on all endpoint, edge, and cloud nodes. To meet the requirement (**TR6**), we, along with testbed, provide several relevant ML-based workloads packaged as containers. We further describe these in Section 3.4. During the workload execution the testbed collects both the compute and network metrics meeting the requirement (**TR8**). We further describe these metrics in Section 3.5.

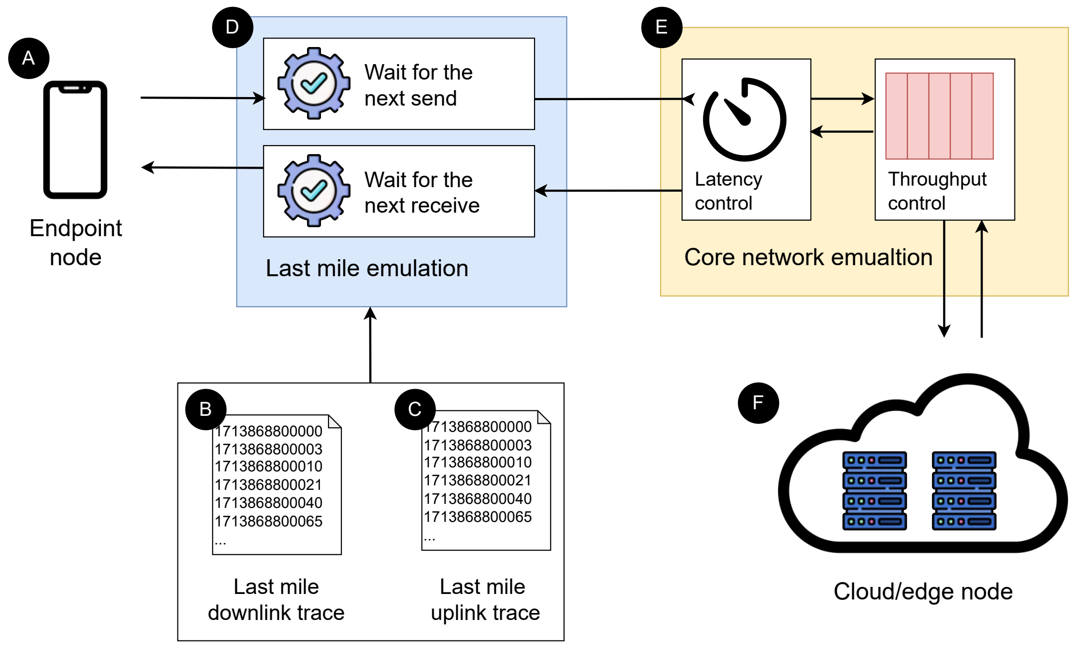

Figure 3.3: Network simulation overview.

3.3 Network simulation

In this section, we dive into the design of network simulation and show how it meets the requirements (**TR2**), (**TR3**), (**TR4**). To meet both requirements (**TR2**) and (**TR4**), we separately simulate the core and cellular networks. We motivate this design decision by performance disparity between the core and last-mile networks with cellular network showing lower bandwidth and higher jitter (4). Meeting the requirement (**TR4**) we configure a **jitter-aware** cellular network simulation through **uplink** and **downlink** traces. Within the traces, the user configures the timestamps at the **millisecond** level of granularity, at which an MTU-sized packet can cross either the **up-** or **down-** links. There can be either no packets that can cross the link at a particular millisecond or there can be one or more packets that can traverse the link at a particular timestamp. At each delivery opportunity, the endpoint can transmit at most an MTU-sized packet since the MAC-layer protocols, commonly adopted by modern 5G infrastructure, assume the endpoint(s) hogging the channel for a time slot allowing to transmit at most an MTU-sized packet (18). The simulation, therefore, meets the requirement (**TR2**) by depending only on the performance perceived by the network layer, represented by timestamps at which at most

Table 3.1: Workloads.

ID	Name	Uplink	Downlink	Compute requirements
T1	Image classification	Image	Text	Low
T2	Text translation	Text	Text	High

an MTU-sized packet can cross the link. To archive this functionality, up- and down- link traces are read by the Ⓓ mobile network simulation component, which buffers the packets and waits for the next delivery opportunity to release the packet to be sent or received. If the packet is larger than an MTU, it will consume multiple delivery opportunities before it can be either sent or received. Apart from mobile network simulation component, to meet the requirement (**TR2**), we introduce a core network simulation component, which can be used can be used to set up a bidirectional static latency representing the distance in the core network to the Ⓕ cloud or edge node. This core network simulation archives the required functionality by buffering the packets and withholding those from being sent or received based on the internal timer.

3.4 Workloads

To meet the requirement (**TR6**), in this section, we design two 6G testbed workloads and outline their relevance for the future 6G use-cases. We chose ML-based workloads, as AI applications are one of the core 6G use-cases, with a wide range of opportunities for complex data processing, new content generation, and others (19). Popular use cases include video and image enhancement, text generation, and object detection. Our workloads are intended to remain representative of these use-cases. For comprehensiveness, we design two different workloads with distinct tranmitted data types - image and text - requiring different network bandwidth and targeting transmission patterns that are bandwidth and latency bound. To keep the network load adjustable, as a proxy for network load exerted by workloads, each workload has a request frequency parameter. Designed workloads are summarized in Table 3.1. Workloads follow the same design that is summarized in Figure 3.4. Workload design employs a task offloading architecture where a computationally heavy back-end is offloaded to a remote Ⓕ cloud or endpoint node that hosts a computationally demanding Ⓓ deep learning model. The Ⓐ endpoint device produces an Ⓔ input data - either Ⓒ image or textual data - and Ⓑ pre-processes the data. Pre-processing is a computationally

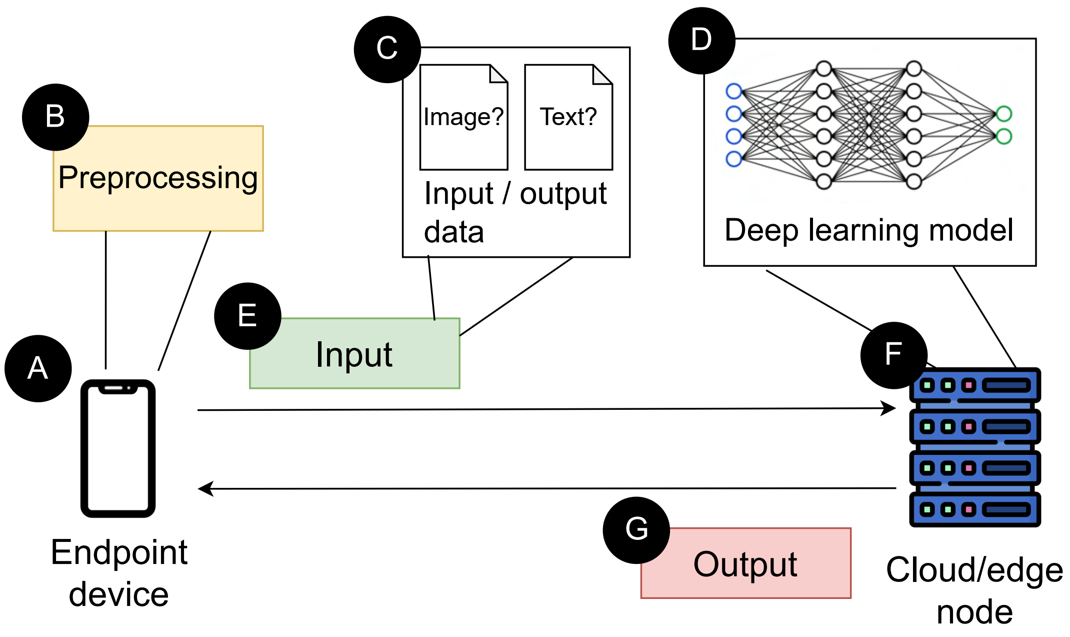

Figure 3.4: Workload design.

un-intensive task which, for example, can be image normalization or text tokenization. Preprocessed input data travels from the endpoint device to the remote node via a network, and then, after the data is processed on the remote node, the G output is sent back to the endpoint device.

3.5 Metrics

In this section, we outline a number of compute and network metrics that we collect and report with a goal of meeting the requirement (**TR8**). When recording the metrics, we, in particular, are interested in their median value reflecting the average performance and in the P99 value reflecting the performance variability. The metrics are summarized in Table 3.2

(**M1**) **Uplink network latency** Due to the difference in uplink and downlink network performance and the difference for some applications in the network load exhibited on network up and down links, we separately record the latency experienced both by **up-** and **down-** link transmissions

Table 3.2: Metrics.

ID	Name	Compute / Network
(M1)	Uplink latency	Network
(M2)	Downlink latency	
(M3)	Pre-processing latency	Compute
(M4)	Server-side processing latency	

(**M2**) **Downlink network latency**. See (**M1**).

(**M3**) **Pre-processing latency** Due to the difference in computational power between the endpoint and the remote node, the pre-processing latency, even for computationally un-intensive tasks, is essential to record.

(**M4**) **Server-side processing latency** With an increase in the compute requirements of ML applications, the server-side compute latency becomes essential to consider.

4

Implementation

In this section, we implement an **easy-to-use** and **adaptable** to user's unique use-case 6G testbed design for which we defined in Chapter 3. The final implementation can be used **out-of-the-box** and extended for any use-case without requiring any change to the internal structure of testbed. This is the implementation we later use to conduct the experiments in Chapter 6. As the result of this section, we answer a research question **RQ2** - "*How to implement a 6G testbed?*". We begin this section by providing a general implementation overview in section 4.1. Then, we dive deeper into the implementation of networking simulation in Section 4.2 and Section 4.3. Finally, in Section 4.4 we go over the implementation of workloads. Following the principles of open science, we publish the implementation as GitHub open-source artifact that allows for the full reproduction of results presented in the paper.

4.1 Implementation Overview

Fulfilling the design presented in Chapter 3, our implementation of the 6G testbed is based on the Continuum implementation (10). Continuum is a peer-reviewed benchmark for computing continuum deployments. This benchmark takes care of provisioning a configurable Ⓐ endpoint, Ⓑ edge, and cloud compute infrastructure through QEMU virtual machines, allowing the end user to configure the 6G testbed compute for their particular infrastructure use case. Within the all the nodes, the user can benchmark an arbitrary workload by submitting it as a Ⓒ Docker container, allowing the user to extend the tested workloads without needing to change the internal structure of 6G testbed. We chose Docker as Docker containers are an industry standard in terms of application containerization among various

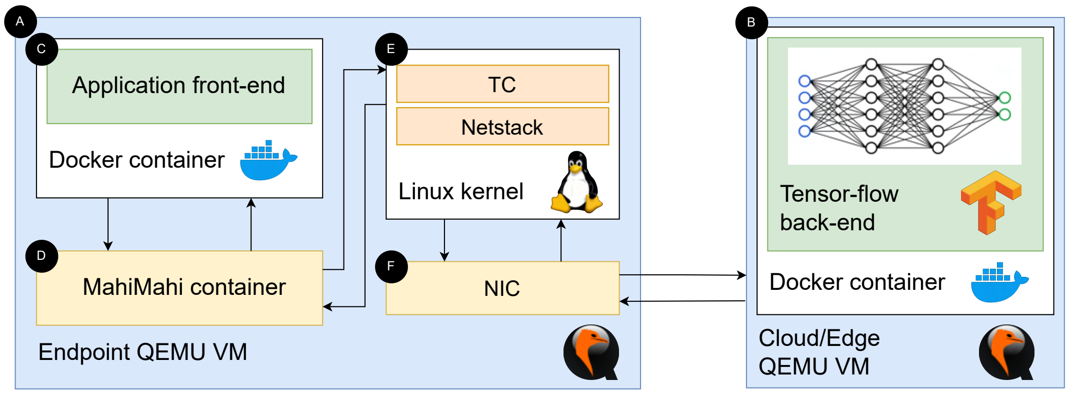

Figure 4.1: Implementation overview.

cloud infrastructure providers (20). For the use-case discussed in this paper - ML applications - the endpoint is responsible for running the application front-end and the remote node is responsible for running application back-end containing a computationally-heavy deep learning model. The deep learning model, is implemented on top of the either Tensorflow or PyTorch libraries - both state of practice deep learning libraries (21) (22). Following the network simulation design, presented in Section 3.3, we separately simulate core and mobile networks. To simulate the mobile network, we redirect the default Linux traffic path from going directly to the Linux kernel to going through a Ⓓ MahiMahi - a state-of-art network simulation tool allowing for network record-replay at millisecond-granularity level (11). We discuss the applied routing rules used to archive this in Section 4.2. The MahiMahi container simulation, is based on the user-provided **up-** and **down-** link **saturattr** traces, which define the delivery opportunities as timestamps at which at most an MTU-sized packet can cross the link. Allowing the container to act as a pass-through for both ingress and egress traffic, we make a number of changes to the original MahiMahi implementation which we describe in Section 4.3. After leaving the MahiMahi container, the packets are redirected through Ⓔ Linux kernel network stack where, using built-in traffic control (**tc**) tool, we apply a static core network latency. We simulate the core network using the **tc** instad of **mm-delay** for the reasons of compatibility with previous Continuum versions which use **tc** for static jitter-unaware network simulation. After, the traffic is redirected to VM network interface card (**NIC**) from where it is forwarded to the Ⓑ cloud or edge node. To ensure that the correctness of latency simulation, the implementation assumes that both the endpoint and the cloud and edge VMs are hosted on the

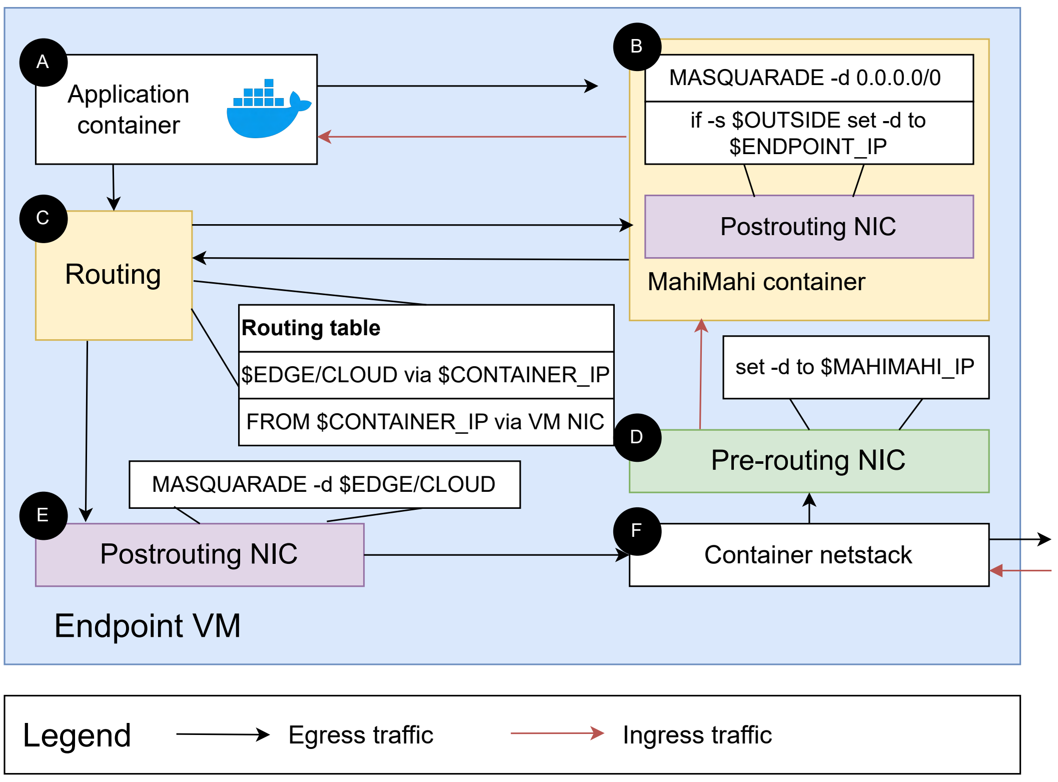

Figure 4.2: Implementation routing overview.

same physical machine and there is disregard-able latency for endpoint to cloud or edge transmissions.

4.2 Routing modifications

To ensure that the traffic is routed through the MahiMahi container and mobile network simulation is applied both for **ingress**, and **egress** traffic, we apply a number of IP route and NAT rules. In comparison to modifying the Linux kernel traffic rules through tools like *eBPF*, our solution applies traffic rules on top of the Linux kernel. This makes the solution easier and, as relying only on user-level OS API, ensures easier compatibility with future Linux versions. We visualize our solution in Figure 4.2. The simulated ingress traffic is redirected by using a D pre-routing NAT box, where the destination is set to the IP of MahiMahi container. We take this approach as we cannot rewrite the routing table for destined for the local IP addresses. Within the container, based on the origin IP address, we separate the traffic and redirect it to the A application container. The egress

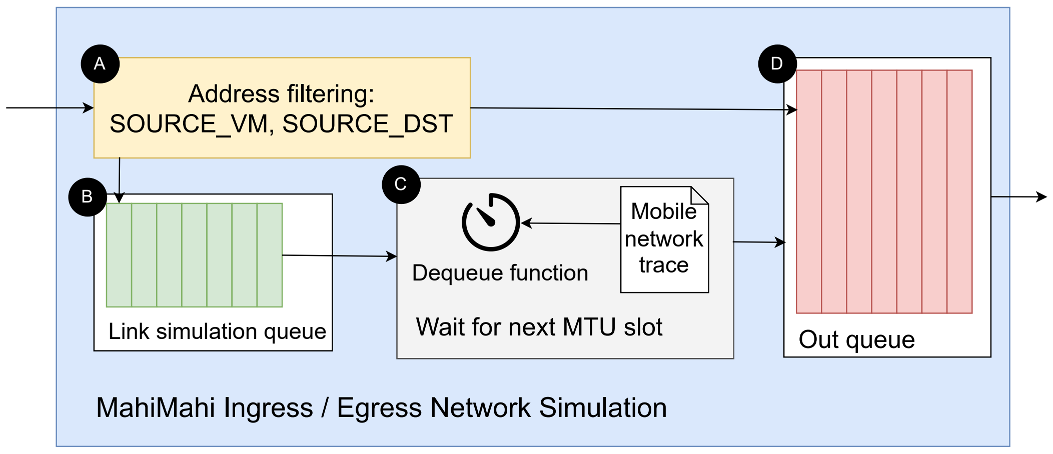

Figure 4.3: MahiMahi NIC traffic shaping.

traffic, on the other hand, is redirected through MahiMahi container, using Ⓒ IP route table rules. Within the MahiMahi container, we *MASQUERADE* the traffic substituting the source IP address with the address of MahiMahi container to ensure that upon traffic redirection from MahiMahi container back into the endpoint VM, we are able to distinguish that traffic from the traffic that still has to go through a MahiMahi network simulation. After that, the traffic is redirected into Ⓕ Linux netstack where the core network latency is applied and the traffic is sent out from endpoint VM.

4.3 Mahimahi modifications

By default MahiMahi is not implemented to act as a pass-through container, and it applies the network simulation only for applications that run inside the container. However, for compatibility reasons with the rest of Continuum project, we modify the MahiMahi implementation to act a pass-through container. In this paper, simulating the mobile network link, we are particularly interested in the container spawned by the *mm-link* command. For that, we start-up a MahiMahi container and make it run an empty application continuously looping on *sleep 10* shell command. MahiMahi has two queues for emulating the traffic - the queue for uplink and the queue for downlink traffic. They are identical in their internal structure; however, they operate on different traces. Using default MahiMahi implementation for pass through traffic would result in mobile network traffic shaping being applied twice instead of once. For example, for egress traffic, the mobile network shaping

Table 4.1: Workloads.

ID	Name	Uplink per request (bytes)	Downlink per request (bytes)	Deep learning library	Model used
W1	Image classification	10K-100K	20	Tensorflow	MobileNetV2
W2	Text translation	100-200	100-200	Pytorch	MarianMT

rules will be applied both when traffic enters the container and when traffic leaves the container. It is impossible to account for this by applying statistical modifications to uplink and downlink network traces - the traces are different and we cannot assume the packet ordering of ingress and egress traffic. To achieve a reliable mobile network emulation for the Continuum use-case, we add an 🅐 address filtering component that allows to exclude a list of sources and destinations from mobile network traffic shaping ensuring that the traffic shaping rules are applies only once for both ingress and egress traffic. If traffic is not filtered out, it first goes into the 🅑 link simulation queue. From the link simulation queue, the traffic is dequeued by 🅒 a function that dequeues a packet of at most an MTU size. It does it in accordance with uplink or downlink traces, which contain timestamps at which an MTU-sized packet can successfully cross the link. After the packet is dequeued, it is sent to the 🅑 out queue from which, via container kernel stack, it is forwarded towards its destination. In case, however, the packet is filtered from mobile network traffic shaping by the address filtering component, we put it directly into the out queue without first passing it through the link simulation queue.

4.4 Workload implementation

We implement two different ML-based workloads with distinct send and receive data types as designed in Section 3.4. We summarize the list of implemented workloads in Table 4.1. The first workload - **W1** image classification - sends uplink an image data and downlink a data containing an image class. The uplink request size varies from around 10 kilobytes to 100 kilobytes, depending on the image size. The downlink response size stays constant to 20 bytes. The second workload - **W2** text translation - sends uplink a textual phrase that needs to be translated and receives back a translated phrase. The request and response sizes vary depending on the length of the phrase but stay in the range of 100-200 bytes. All the metrics are recorded using the Python *time_ns()* function. The underlying physical machine or machines should be set up to guarantee VM clock synchronization. Architecturally, all workloads are implemented the same, and their implementation is summarized

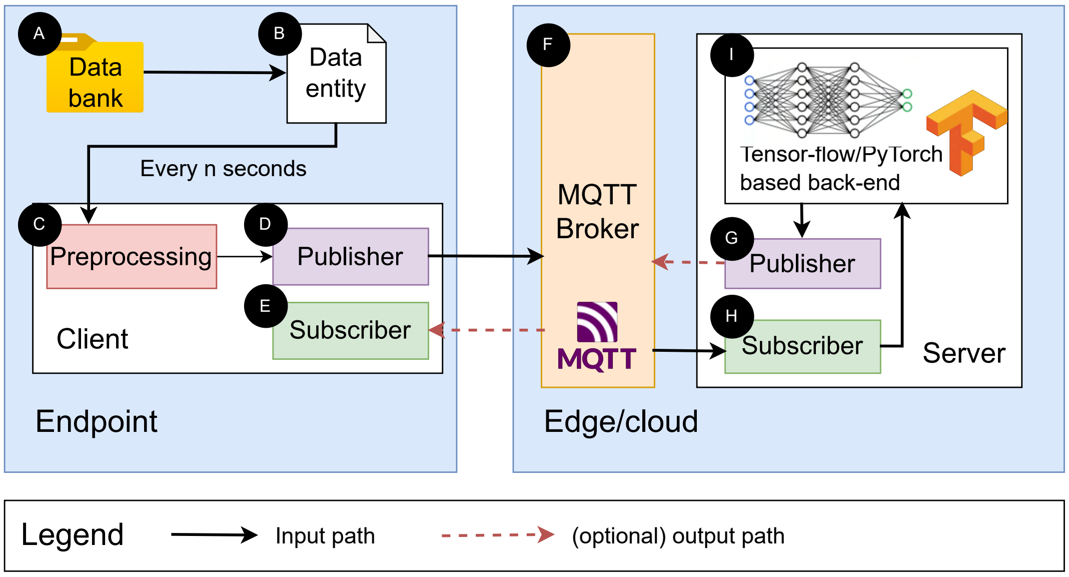

Figure 4.4: Workload implementation.

in Figure 4.4. For network communication, every workload uses uses MQTT protocol - a state-of-the-practice low-latency application-level protocol for IoT devices (23). There, a Ⓑ request data is pulled from a Ⓐ data bank one-by-one at configurable intervals. There, it is passed through a Ⓒ pre-processing component. Pre-processed data is then sent out through the MQTT Ⓕ publisher to the Ⓕ MQTT broker hosted on the cloud or edge node. The MQTT broker then redirects the data to the edge/cloud Ⓗ subscriber, which gives it to an application running a TensorFlow or PyTorch-based back-end. We record the uplink network latency as the time it takes for a request to go from Ⓓ endpoint publisher to Ⓗ cloud or edge subscriber. Whenever the result is computed, it is sent to the edge or cloud Ⓖ publisher, which sends it to the MQTT broker. The MQTT broker, in turn, passes the data to the endpoint Ⓔ subscriber. We record downlink request time as the time it takes to go from Ⓖ edge or cloud publisher to Ⓔ endpoint subscriber.

5

Cellular Network Trace Archive

This section defines and applies a method to collect network performance of existing network setups. The result of this section is an open-source cellular network trace archive containing the performance traces of the core and mobile Dutch network infrastructure. We also conduct performance characterization of the collected traces, outlining several system-level trends and limitations. We begin this section by defining a method used for performance data collection in Section 5.1. We then outline the collected traces and motivation behind collecting those traces in Section 5.2. Finally, we characterize the performance of those traces in Section 5.3.

5.1 Method

In this section, we define a method we use to collect performance traces of existing networking infrastructure. Due to the difference in bandwidth and jitter, we, following the taxonomy introduced in Section 3.3, collect the core network and mobile network performance data separately. Our method assumes that *(i)* core network jitter is insignificant compared to mobile network jitter and *(ii)* that the mobile network has significantly lower bandwidth compared to the core network.

We collect the mobile network performance data using the setup shown in Figure 5.1. To collect data, at a Ⓐ fixed location, the Ⓑ user runs a Ⓒ saturatr client. *saturatr* is a tool that is used to collect MahiMahi compatible traces (11). It does so by saturating the network between the client and the server with as many packets as the network can handle and recording the packet arrival timestamp on the server side. It then treats the difference between the timestamps of two consecutive packets' arrival times as a ground truth for the difference between two consecutive MTU-sized delivery opportunities. To

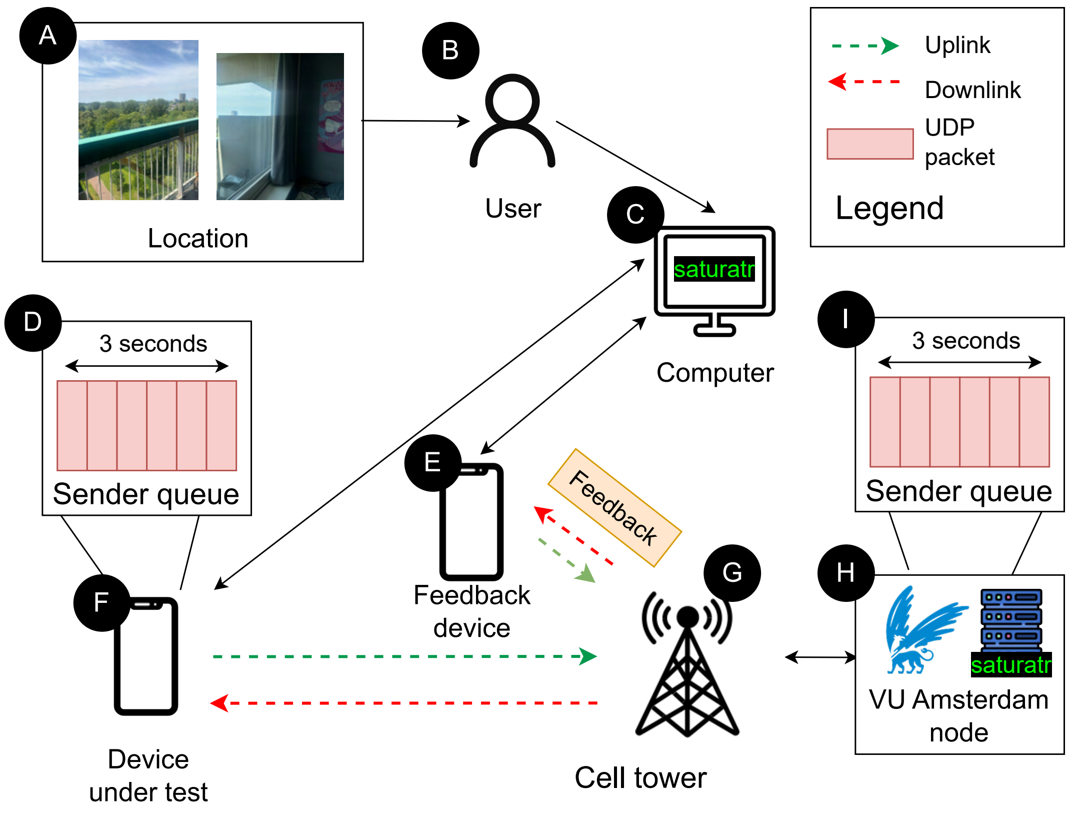

Figure 5.1: Mobile network performance collection setup.

reliably saturate the link under constantly changing bandwidth conditions without causing the cellular network provider to drop the packets, the computer running *saturatr* client is connected to two phones: F device under test (DUT) and E the feedback device. To saturate the link, the F DUT aims to maintain the sender queue between 0.75 and 3 seconds (11). To adjust the queue size under constantly changing network conditions, the DUT uses the E feedback device, which receives the feedback on the packet 1-way trip time. Packets from both devices travel to G cell tower, and from the cell tower travel to H a remote node. Whenever the packet reaches the remote node, the received timestamp is recorded. The difference between two consecutive timestamps is treated as the ground truth for the intervals between two mobile network delivery opportunities. To record downlink performance, the *saturatr* server, like *saturatr* client, maintains a I packet queue that 0.75-3 seconds long. The *satuatr* client, in turns, records the timestamps at which the packets are received and treats the difference between those as a ground truth

Table 5.1: Mobile network data collection traces

Standard	Obstacle	Date	Location	Provider
LTE	None	Jun. 17	52.320949, 4.875078	KPN
5G	Outside			
	Window			

Table 5.2: Core network data collection.

Name	Type	Region	RTT (12)
Base tower edge	Edge	Amsterdam, The Netherlands	∼ 1ms
EU-Central-1	Local cloud	Frankfurt, Germany	∼ 10ms
EU-West-1	Regional cloud	Paris, France	∼ 20 ms
US-Virginia-1	Transcontinental cloud	Virginia, US	∼ 80 ms

for the intervals at which downlink packets can traverse the mobile network link.

To collect the core network performance data, we follow the taxonomy introduced in (12), and collect the latency data to the edge, local, regional, and transcontinental clouds with the expected set of latencies of less than 1, 10, 20, and 80 ms, respectively. To collect the data, we run, from a client with a wired internet connection, a *ping* utility which records a round-trip time (RTT) to a server by exchanging *ICMPv4/6* packets. Our method assumes that the core network has bidirectional equal latency and 1-way latency by dividing the RTT by 2.

5.2 Tracing

The mobile network traces collected are summarized in Table 5.1. We collect mobile network performance traces at a fixed distance of **230 meters** from the nearest LTE/5G cell tower. All traces are collected on the same day - Jun. 17th and at the same time and within 2-hour window between 12:00-14:00. We collect the traces for both LTE and 5G, as these are, at the moment, the most used mobile network standards. For 5G, as well, we collect the traces both with and without an obstacle in between the cell tower and the DUT to explore how the cell network performance changes inside and outside the buildings. All the traces are collected with the KPN infrastructure, as KPN (as of June 19th, 2025) does not limit the bandwidth of tethered devices, which is essential for our network data collection method.

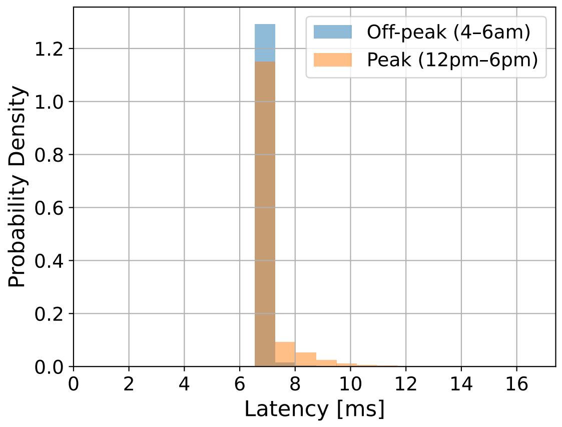

Figure 5.2: Frankfurt, Germany (EU-Central-1)

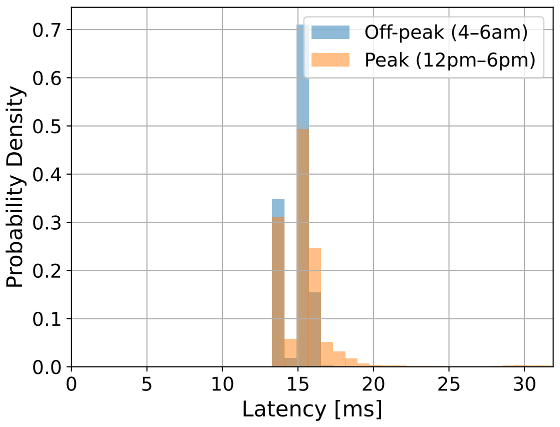

Figure 5.3: Paris, France (EU-West-3)

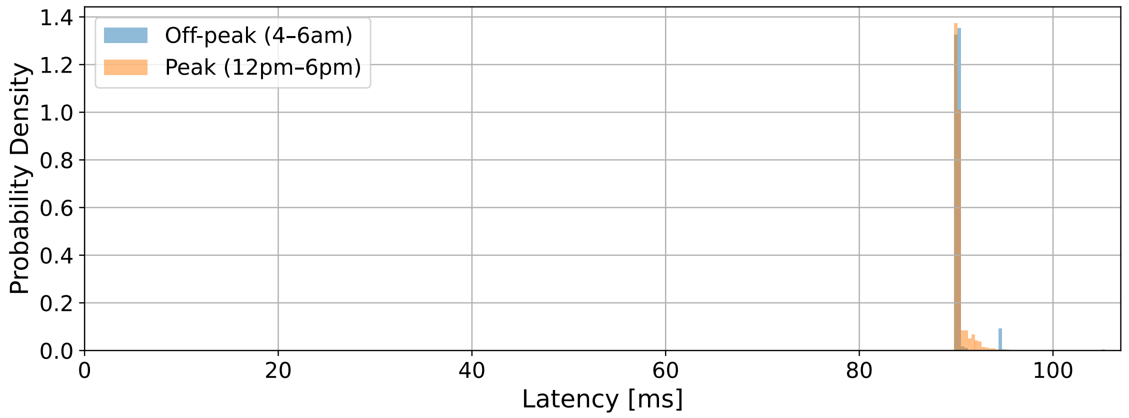

Figure 5.4: Virginia, USA (US-East-1)

Figure 5.5: Core network RTT distribution for different geographical destinations

We collect the core network traces using the public AWS *ICMPv4* API. We summarize collected core network latencies in Table 5.2. Following the expected RTT range, we identify the EU-Central-1, EU-West-3, and US-East-1 as local, regional, and transcontinental cloud nodes. For the edge device, we assume that it will be located near a cell tower and have a negligible core network latency. We collect the traces over multiple days to capture the network performance, short-term, and long-term performance variability, including the variability caused by daily differences in utilization between the peak and off-peak hours and the differences caused by temporary changes in the BGP routing.

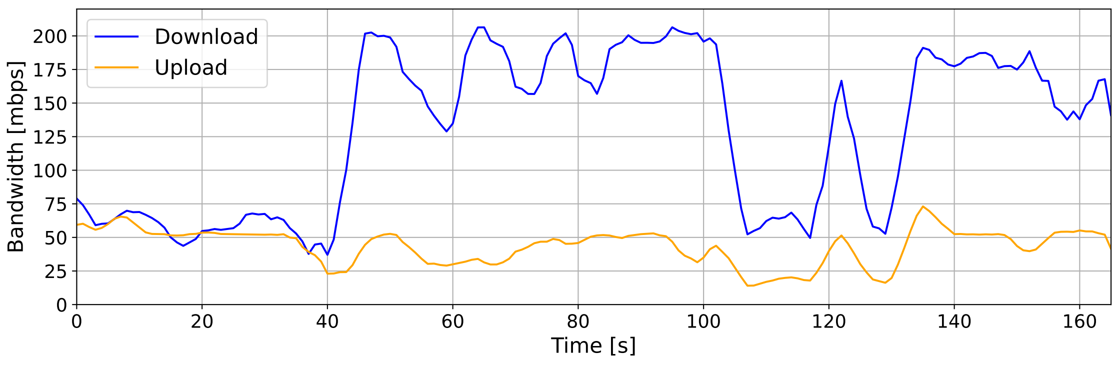

Figure 5.6: 5G no obstacles uplink and downlink traces

5.3 Analysis

The core network latency data obtained for different deployment locations described in Section 5.2 is shown in Figure 5.5. The data obtained demonstrates that, compared to the current mobile network generation, the core network shows minimal jitter, with the majority of requests having similar RTT. However, there is still observable performance variability, with RTTs falling under a long right-tail distribution, which is a well-established, by current academic literature, network latency distribution pattern (24). We can also observe that there is a slight difference between the RTT recorded during the peak and off-peak hours, which we hypothesize is a result of increased network congestion levels and core network queue build-ups during the peak hours. It is also important to note the presence of a binomial distribution of EU-West-3 RTT, which can be important for some applications. We hypothesize that this variation in latency is due to the temporary changes in BGP routing resulting in two different routes - a phenomenon well established in the academic literature (25).

Mobile network performance data collected for LTE and 5G standard recorded with and without window obstacle (as described in Section 5.2) is shown in Figure 5.8, Figure 5.7, and Figure 5.6 respectively. The traces are not directly comparable, as during the recording, we do not control for many compounding factors such as current network noise or experienced network load, which can influence mobile network system-level performance. From the traces, we can observe that the biggest difference between LTE and 5G system-level performance comes from the increased download capacity - up to **200 Mbps** for 5G compared to only **80 Mbps** for LTE. 5G upload capacity; however, is significantly higher than LTE upload capacity only for the trace recorded without any obstacles between the

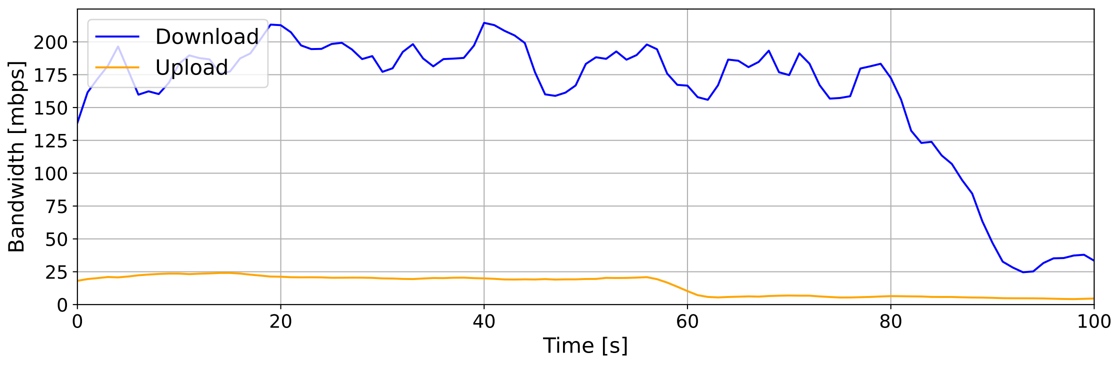

Figure 5.7: 5G window obstacle uplink and downlink traces

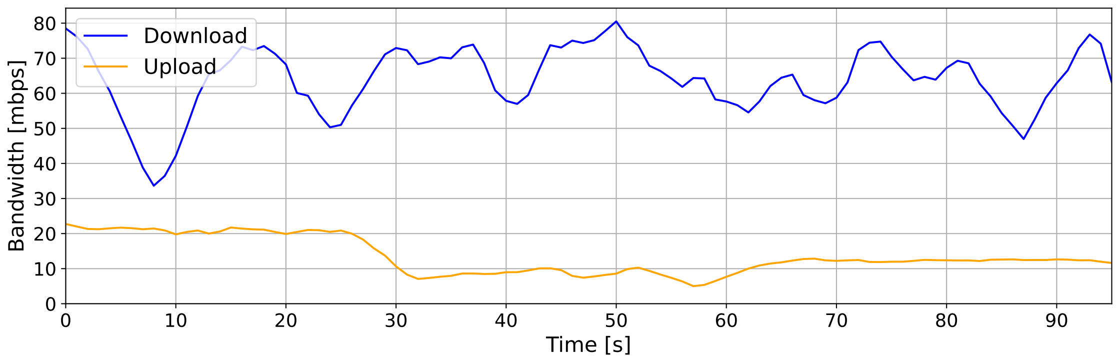

Figure 5.8: LTE uplink and downlink traces

endpoint and the cell tower - up to **75 Mbps** for 5G compared to only **20 Mbps** for LTE. The trace recorded with a window obstacle between the endpoint and the cell tower shows a similar to LTE uplink capacity, falling largely between **10 Mbps** and **25 Mbps**. We hypothesize that this decrease in 5G upload capacity comes from the endpoint's lower energy available to for transmission in comparison to the cell tower. This results in a significant decrease in the bandwidth whenever an obstacle is present. We can, as well, observe that all the traces have a high jitter, with network performance constantly changing in unpredictable patterns. All the traces show upload speed to be significantly lower than the download speed, highlighting the disparity between the downlink and uplink mobile network performance. This difference is the most pronounced for 5G recorded with a window obstacle, where the upload speed does not exceed **25 Mbps**, while the download reaches **200 Mbps**.

6

Evaluation

In this section, using the 6G testbed designed and implemented in Chapter 3 and Chapter 4, and using the cellular network trace archive constructed in Chapter 5, we evaluate the impact of current and upcoming 6G cellular network generations on ML applications under different real-world conditions. To do that, we attempt to answer the following experimental questions:

(**Q1**) **What is the impact of mobile network technology on end-to-end latency?** We evaluate the difference in end-to-end latency caused by a change in cellular networking technology and demonstrate the need to explicitly address the variability of cellular network performance. (Section 6.3)

(**Q2**) **What is the impact of text and image-based AI-workloads?** We evaluate the difference in end-to-end latency caused by transmitting different data types - image and text - and demonstrate how modern network standards scale under increased application bandwidth requirements. (Section 6.4)

(**Q3**) **What is the impact of offloading to edge, local, regional, or transcontinental cloud?** We evaluate the difference in end-to-end latency caused by a change in offloading to edge or cloud offloading and demonstrate the need for close geographical offloading for ultra-low-latency applications. (Section 6.5)

(**Q4**) **What is the impact of an increase in network load?** We evaluate the difference in end-to-end latency caused by increasing application bandwidth demands and demonstrate a need for application-level congestion control mechanisms to tackle cellular network performance variability. (Section 6.6)

(**Q5**) **What is the impact of projected 6G on end-to-end applications?** We evaluate the impact of projected 6G on end-to-end latency and show the need for multi-component optimization for ultra-low latency applications. (Section 6.7)

We begin the section by defining the set of main findings in Section 6.1. Then, we discuss the experiment setup used for obtaining results in Section 6.2 and discuss separately each experiment in Section 6.3, Section 6.4, Section 6.5, Section 6.6, and Section 6.7.

6.1 Main findings

We summarize the following set of main findings (MF):

(**MF1**) In our setup, the uplink capacity became a limiting factor when considering the end-to-end network latency for image classification workload. The uplink latency accounted for over **90%** of the total network latency for both LTE and 5G networks (Section 6.3).

(**MF2**) In our setup, textual data transmission results in a **1.5 time** decrease in P99 network latency when compared to the transmission of uncompressed image data. This highlights the importance of input data pre-processing with the goal of compression for real-time applications (Section 6.4).

(**MF3**) In our setup, the 30-second drop of mobile network performance below a critical value resulted in **200 times** increase in experienced uplink latency for image classification workload (Section 6.6). This highlights the importance of application-level congestion control for real-time applications (Section 6.6).

(**MF4**) Our results suggest that achieving end-to-end latency requirements of novel applications such as autonomous vehicles or virtual reality requires not only improvements to the network but also improvements to baseline CPU performance achieved in our setup. This suggests that unlocking the novel low-latency applications requires an approach towards a decrease in both network and compute latencies (Section 6.7).

6.2 Experiment Setup

As described in the introduction, we conduct a number of experiments to answer the **RQ4** determining the impact of cellular networks on the ML applications under different conditions. The conditions include *(i)* mobile network technology, *(ii)* deployment location, *(iii)* workload difference, and *(iv)* network load. We summarize the experiments and their

Table 6.1: Experiments.

ID	Focus	Independent variable	Section
E1	Impact of last-mile networking	Last-mile network	6.3
E2	Impact of image and text-based workloads	Workload type	6.4
E3	Impact of offloading location	Core network	6.5
E4	Application scalability under increased load	Network load	6.6

independent variables in Table 6.1. Unless otherwise noted, all the experiments employ the image classification workload described in Section 3.4. As our primary goal is to explore the impact of cellular network, unless otherwise noted, the workload is deployed on the edge with no core network latency simulated (Section 5.2). Keeping the results up to date with state-of-practice mobile network infrastructure, unless noted otherwise, we use the cellular network trace with the highest recorded uplink and downlink capacities - KPN 5G trace recorded with no obstacles between the endpoint and the cell tower (Section 5.2). As the network is the primary focus of this work, we ensure that there is always sufficient computing resources available for processing endpoint requests. In our setup, we empirically find that this property holds if the request frequency is kept to 1 per second. In the first experiment, to identify the effect of mobile network technology, we take the last-mile network as an independent variable. In the second experiment, identifying the difference between image and text-based workloads, we take a workload type as an independent variable. In the third experiment, identifying the difference caused by changing offloading locations, we take the core network latency, used as a proxy for the geographical distance to the offloading location, as an independent variable. In the fourth experiment, identifying the effect of increased network load on application scalability, we take the network load as an independent variable. We use the request frequency as a proxy for network load. To ensure that there is enough compute capacity to process all the requests, even if we saturate the cellular network with the requests, we use the MahiMahi Verizon 4G trace (11) from 2016, which has lower, than those recorded by ourselves downlink and uplink capacities. We empirically verify that for all the request frequencies used in that experiment, the compute does not become a bottleneck. We run all the experiments in a computer cluster at VU Amsterdam. Its hardware configuration are summarized in Table 6.2. The cluseter experiments were run on a 20-core Intel(R) Xeon(R) Silver 4210R CPU running at 2.40GHz and a total of 256 GB of memory. We did not overclock the CPU throughout the experimentation. The installed Linux kernel

Table 6.2: Experiment hardware.

Component	Value
Core count	20
Memory	256 GB
CPU	Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz
Endpoint	
Core count	2
Memory	4 GB
Remote node (edge/cloud)	
Core count	4
Memory	16 GB

is 5.4.0-193-generic, and the lsb version is Ubuntu 20.04.6 LTS. Determining the available compute for the endpoint and the remote edge/cloud node, we used the default values for the endpoint and cloud nodes provided by the Continuum (10) paper. For all experiments, when considering a median or P99 latency decomposition, we consider the median and P99 values for each component separately, assuming that their performance is independent of other components.

6.3 Network generation

In this section, we discuss the impact of mobile network technology on the ML application performance. Obtaining the results, we compare the end-to-end latency and its decomposition when executing the image classification workload with LTE traces and 5G traces, which were recorded both with and without an obstacle in between the endpoint and the cell tower. The traces are further discussed in Section 5.2. As a result we find that, in our setup, the median end-to-end latency is largely determined by compute, which comprises up to **80%** of end-to-end image classification request latency. However, consistent among network standards was the issue of uplink performance variability. For P99 latency, the last-mile network comprised up to **80%** of the end-to-end image classification request time with the uplink latency comprising up to **95%** of the experienced end-to-end network latency for an image classification request.

In Figure 6.1, we show the end-to-end latency for every image request. From the graph, we can observe that, in our setup, the difference between the LTE and 5G is minimal, and both deployment having a high performance variability with often latency spikes. We can also observe that there is a plateau that limits the end-to-end latency. To explain the

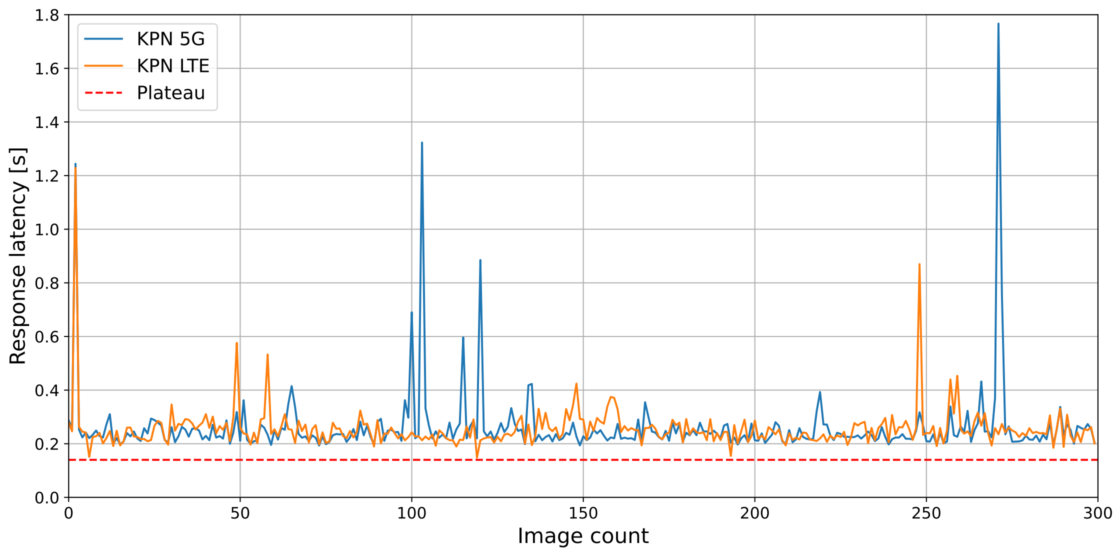

Figure 6.1: End-to-end request latencies

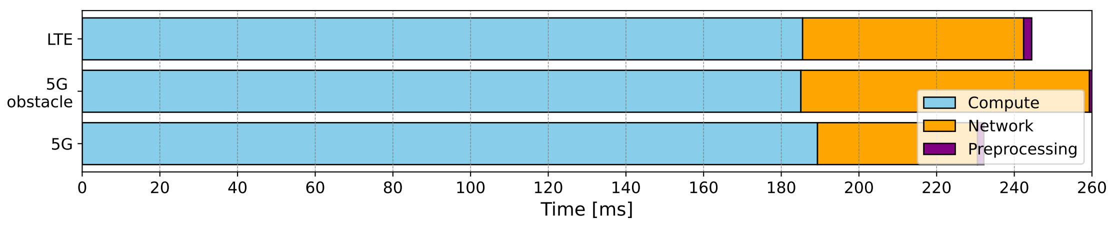

Figure 6.2: Median network latency decomposition

source of performance variability and plateau, we consider a decomposition of the median and the P99 end-to-end latencies. We show those in Figure 6.2 and Figure 6.3. There, we observe that the network comprises only a minority of end-to-end latency with compute responsible for up to **80%** of end-to-end request time. We can also see that there is no improvement going from the obstructed 5G (recorded with a window obstacle as described in Section 5.2) to LTE. When we consider the P99 latency, however, we can see that the performance is limited by the network and there is no little correlation between the network standard and P99 network latency.

We explain this difference by considering the median and the P99 network latency decomposition. We show those in Figure 6.4 and Figure 6.5. For all, we can see that for both median and P99 latency, the uplink performance is a limiting factor. The trace resulting in the best median value latency value is the unobstructed 5G trace - the trace with the high-

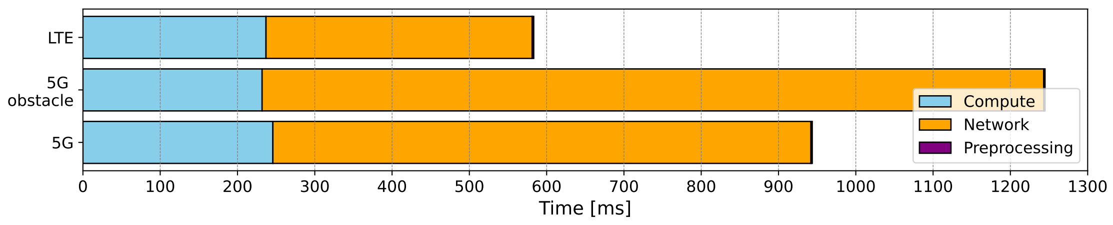

Figure 6.3: P99 network latency decomposition

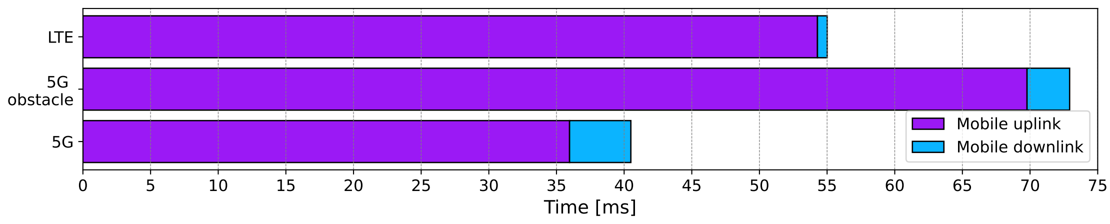

Figure 6.4: Median network latency decomposition.

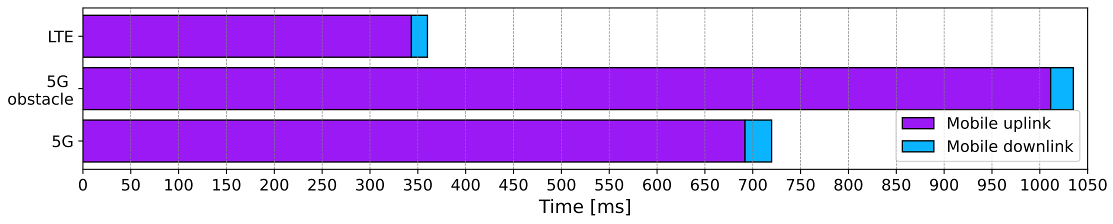

Figure 6.5: P99 network latency decompositon.

est recorded uplink speed. When considering the P99 latency, however, we observe that no improvement with selected mobile network standards, indicating that uplink performance variability remains a challenge for future mobile network standards.

6.4 Different applications

In this section, we discuss the difference in application response latency under image classification and text translation workloads. The workload, along with their bandwidth uplink and downlink requirements, are summarized in Section 3.4. We find that the image classification workload, transmitting uplink a bandwidth-heavy image, results in higher network latency and network performance variability when compared to the text translation workload, transmitting uplink, lower in bandwidth requirements, textual data.

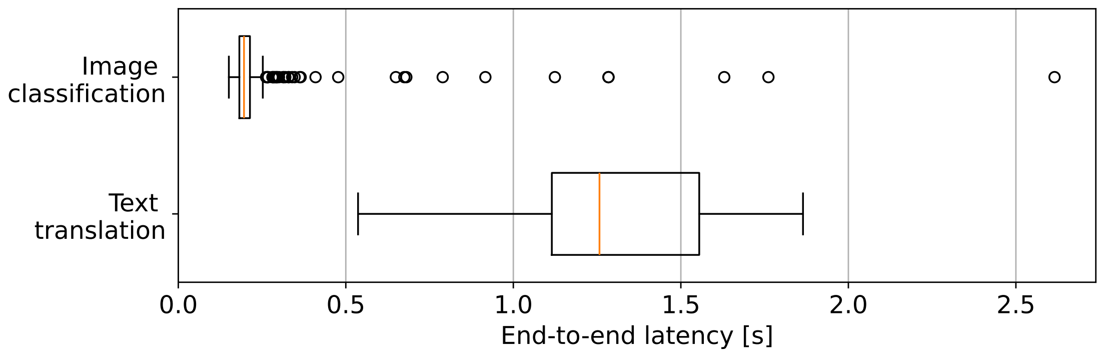

Figure 6.6: End-to-end latency distribution by workload

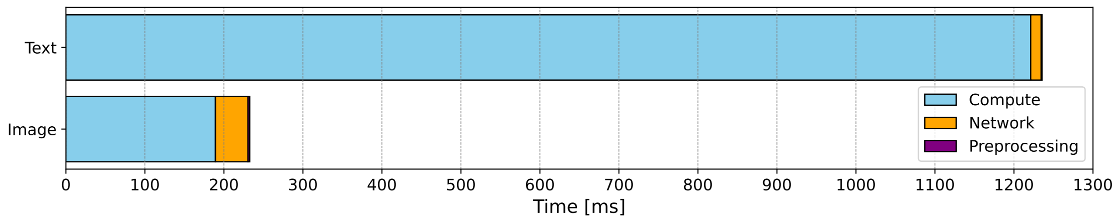

Figure 6.7: Median end-to-end latency decomposition

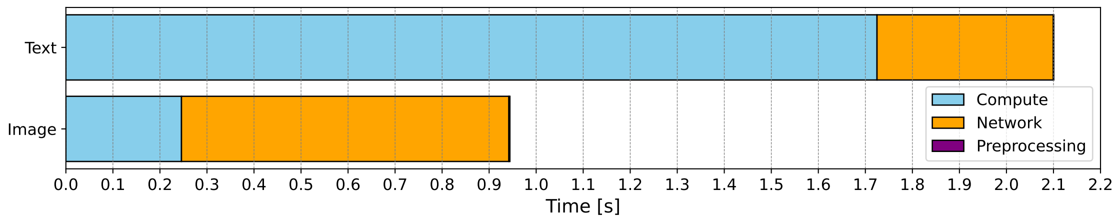

Figure 6.8: P99 end-to-end latency decomposition

We show the difference in the end-to-end request latency in Figure 6.6. As observed, text translation has a higher end-to-end latency, but the end-to-end latency itself shows lower performance variability than observed for the image classification workload. To explain this, we refer to the end-to-end latency decomposition of the median and P99 request, which we show in Figure 6.7 and Figure 6.8 respectively. From those, we find that the difference in median response latency is largely explained by the difference in compute latency. The difference in the median latency is explained by the difference in compute. For text translation workload, the server-side compute comprises over **99%** of the end-to-

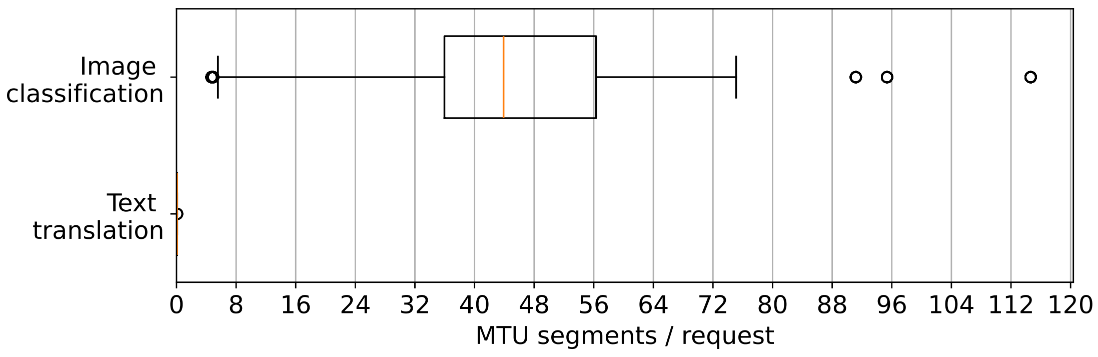

Figure 6.9: Uplink requirements per application

end latency. For P99; however, the network is responsible for a more significant portion of the end-to-end latency. It is also evident that the P99 network latency for the text translation workload is significantly lower than for the image classification workload. This is explained by the difference in the uplink utilization patterns for the two workloads. We show the average request size, measured in the number of MTU slots in Figure 6.9. There, we can see that while all the text translation requests can be transmitted in one MTU slot, the image classification requests take around 50 MTU slots per uplink request.

6.5 Deployment location

In this section, we discuss the differences resulting from running the workload at different deployment locations. The information on deployments - edge, EU-Central-1, EU-West-1, US-Virginia-1 - and the information on their respective core network latencies can be found in Figure 5.5. We find that while there is a significant effect of core network latency on the P1 and median request times, the P99, largely dominated by high mobile network jitter, experiences minimal improvements for different offloading locations.

We show the results for edge, EU-Central-1, EU-West-1, and US-Virginia-1 deployment in Figure 6.10. For edge, we see that while the median latency stays well below 100 ms, the distribution has a very long end with a some outliers reaching above 1000 ms. EU-Central-1, EU-West-1, and US-Virginia-1 latency distribution experiences a shift with an increase in the core network latency; however, the jitter, largely caused by mobile network hop, remains significant for all. Upon analysis of the core network latency and experienced uplink latency, we find that every 1 ms drop in core network, results in a **4 ms** drop in experienced median uplink latency. We show this in Section 6.5, where we find the

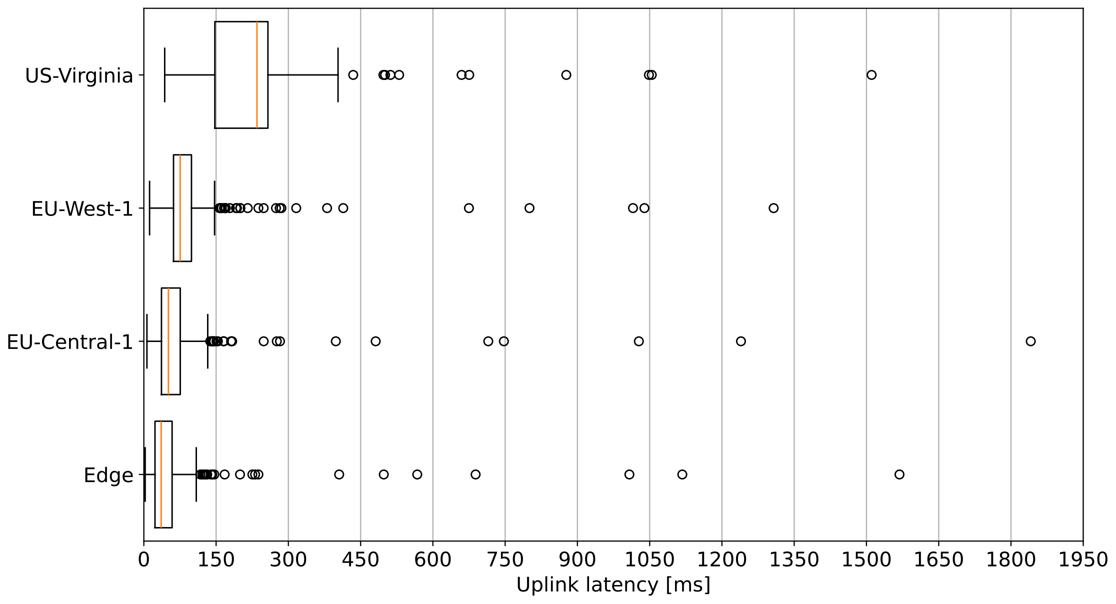

Figure 6.10: Deployment location latencies.

correlation by applying a linear regression with minimal squared error objective on the collected data. We hypothesize that this suboptimal scaling is a result of limitations posed by the TCP congestion control protocol, resulting in a suboptimal choice of congestion window size on **high-jitter** long links. Validating the results, we analyze the trend in P1 latency, shown in Section 6.5, and find that, as expected, with each 1 ms increase in the core network latency, the uplink latency increases by around 1 ms. The P99 latency shown in Section 6.5; however, defined largely by the mobile network performance variability, only shows a weak correlation between the core network latency and uplink latency.

6.6 Network load

In this section, we identify a number of scalability concerns of the applications deployed on top of mobile networks. To obtain the results, we run the image classification workload with *2*, *5*, and *10* requests per second. For this experiment, we used the default MahiMahi Verizon 4G mobile network trace. This trace, being recorded in 2016, has lower network capacity than shown by modern network infrastructure traces collected in Section 5.2. We use this trace; however, to saturate the mobile network link with the request data without a compute becoming a bottleneck for the end-to-end request processing. We empirically validate that this property holds for our experimental setup. The results, however, is a

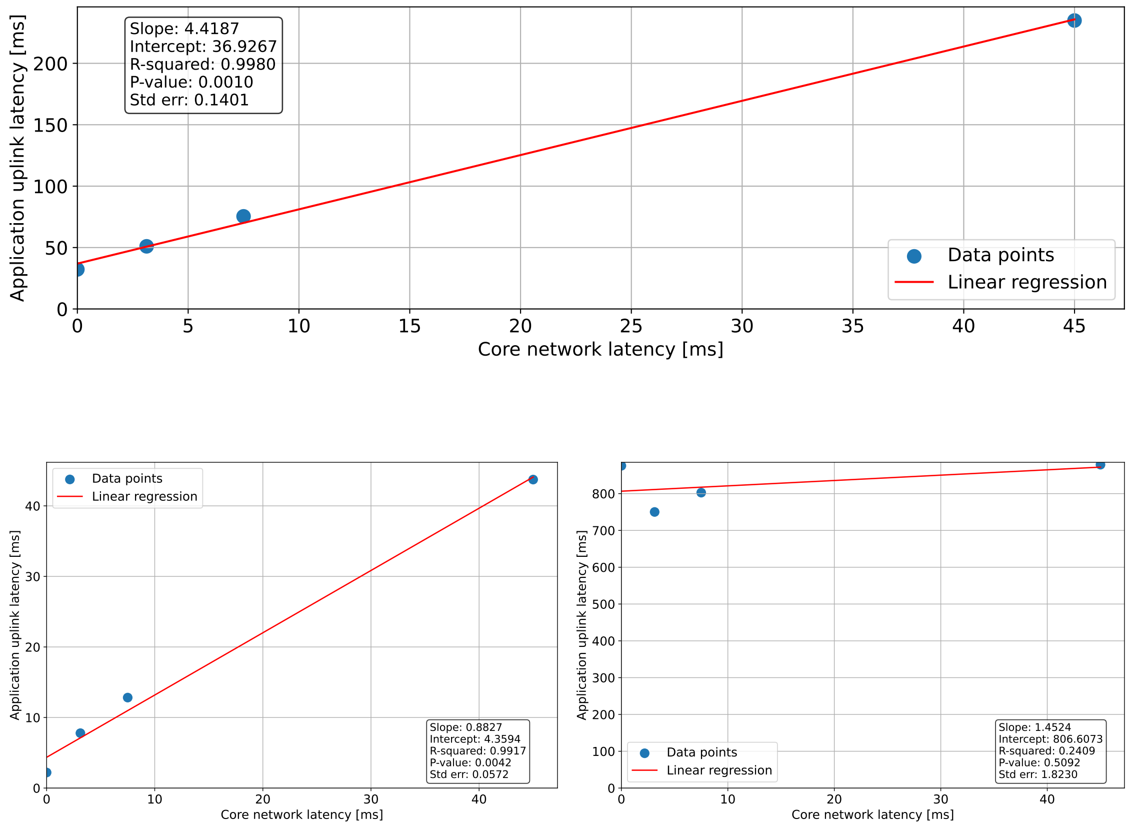

Figure 6.11: Core network latency analysis: (top) Median uplink core network latency, (bottom left) P1 latency, (bottom right) P99 latency.

result of inherent mobile traces performance variability; and, therefore, apply to any high-jitter networks. Our results indicate the necessity of application-level congestion control for real-time applications. Our results show that with trace and workload that is ran, the uplink queue builds up, resulting from low mobile network performance during limited time periods, can lead to a **200 times** increase in uplink latency as a result of congestion collapse.

We show the obtained uplink latencies for different request frequencies in fig. 6.12. There, we observe that while the uplink latency tends to stay under **100 ms** under the optimal network conditions. However, during the periods of low uplink performance, in our setup, the uplink latency increases up to **20 seconds**. In Figure 6.13, we show the correlation between the uplink capacity and the uplink latency when workload is run with 10 requests/second. We observe that whenever the uplink capacity drops below a certain value, which is around 5 Mbps, the uplink latency increases as well. The longer the period of network deterioration, the higher the increase in the uplink latency. As we show in Fig-

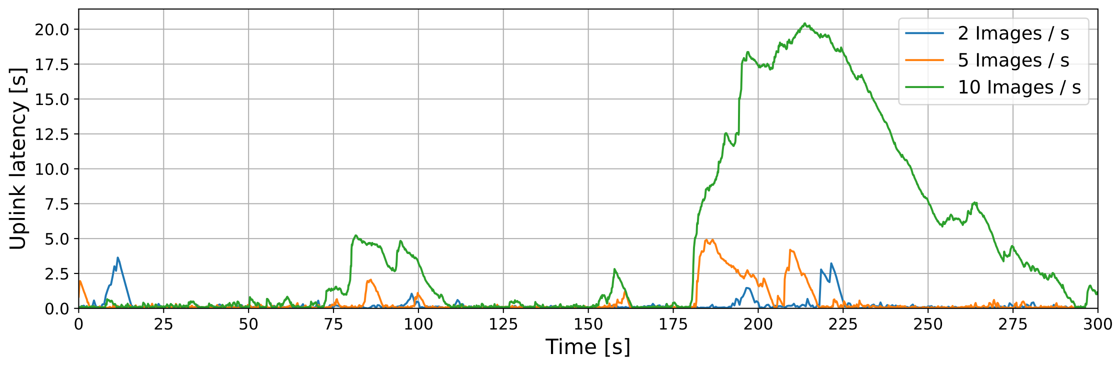

Figure 6.12: Uplink latency vs time

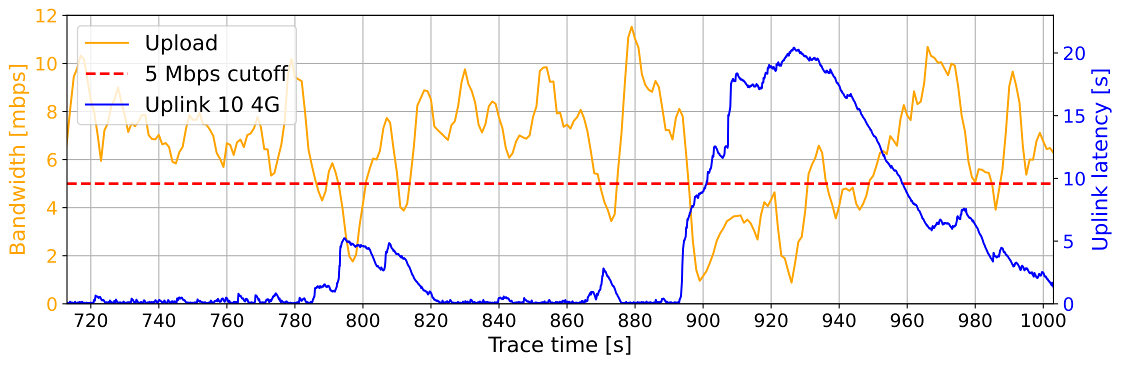

Figure 6.13: Upload latency vs upload speed

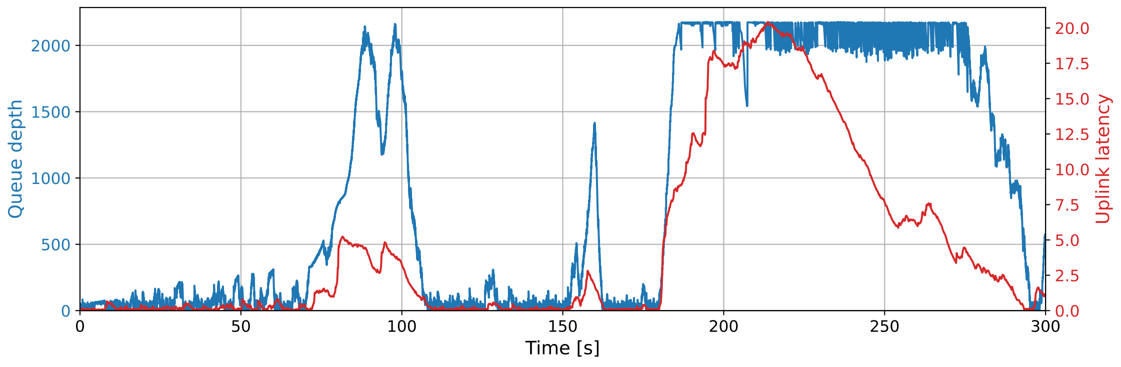

Figure 6.14: Uplink latency vs queue depth

ure 6.14, this is a result of the uplink queue buildup. Whenever the uplink queue increases, the uplink latency increases as well. Whenever the queue size flattens out, indicating that

```
if latency > 500 ms:
    sending_rate = config["frequency"] / 2
else:
    sending_rate = config["frequency"]
```

Listing 2: Congestion control

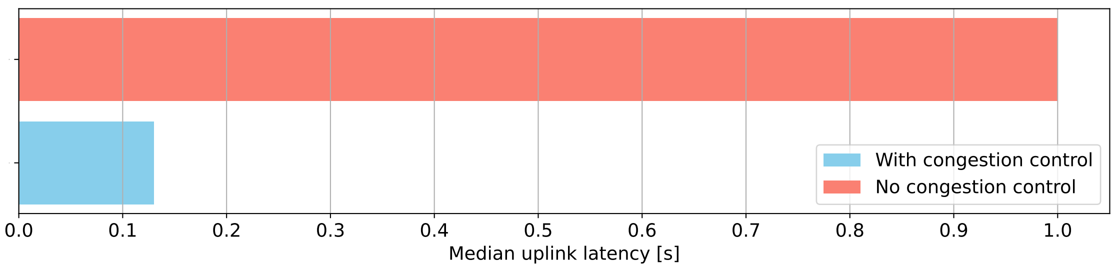

Figure 6.15: Application-level congestion control performance. Median value

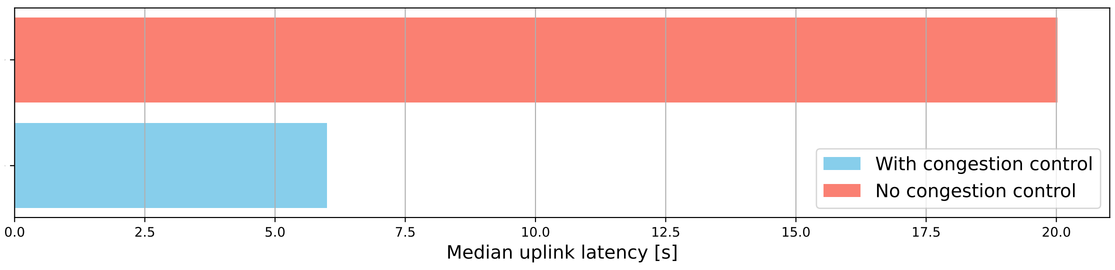

Figure 6.16: Application-level congestion control performance. P99

the queue is full and the link starts to drop packets, the uplink latency increases by up to **200 times**. We validate this hypothesis by constructing an application-level congestion control mechanism that divides the sending rate by 2 whenever the end-to-end latency is higher than 500 ms. We show the pseudocode for this congestion control mechanism in Listing 2. Use of this congestion control results in, as shown in Figure 6.15 and Figure 6.16, a 9 times latency decrease for median end-to-end request latency and a 3.5 time decrease for the P99 end-to-end latency.

To demonstrate how application developers could handle this performance variability, we modified the application with a simple application-level congestion control mechanism. We summarize this congestion control mechanism implemented in Listing 2. This mechanism checks if the end-to-end latency is above 500 ms and, in case it is, it halves the rate at which

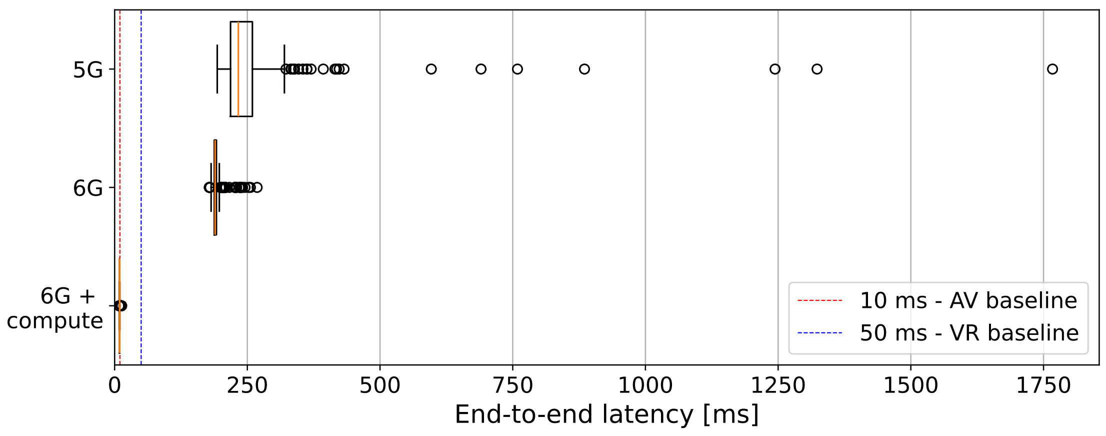

Figure 6.17: 6G Prospected.

data is being sent by halving the number of images that the endpoint sends to the server. Applications can implement a different way of reducing the application data rate, which will depend on the semantics of the applications. An example of an alternative technique is reducing the image resolution instead of reducing the frequency at which those are sent. In Figure 6.15, we show a comparison of median uplink times, and in Figure 6.16, we show a comparison of P99 uplink times. There, we can see that implementing this simple application-level congestion control mechanism can speed up the performance almost 10 times, dropping the median delay from 1 second to 0.12. It, as well, drops the performance variability almost 4 times, from 20 seconds to 5.5.

6.7 6G Impact

In this section, we analyze how the upcoming 6G standard can make novel low-latency applications possible. For that, we assume edge deployments with minimal core network latency, and we assume the 6G has negligible jitter. The 6G performance baselines used are 100 Gbps bandwidth and 0.1 ms latency (26). We calculate network transmission latency using the formula:

$$latency(ms) = (\frac{image\_size}{100\ Gbps}) + 0.1 + 0.1$$

(0.1 is repeated twice to account for both uplink and downlink latencies. We assume that the response containing only an image class has negligible size). We summarize the results of end-to-end latency for image classification workload in Figure 6.17. We can observe that, in our setup, while the transition from 5G to 6G will decrease the end-to-end latency and, importantly, will drastically decrease the request performance variability, the improvement solely to the cellular network are not sufficient to allow for novel applications. The limiting factor, in this case, becomes the server-side compute. If, however, along with the network, we improve the compute by a factor of 20 when compared to the compute configuration used in the setup, the end-to-end latency becomes sufficiently low to allow for novel low-latency applications. This shows that transitioning to 6G-enabled applications requires consideration of both **compute** and **network** components.

7

Conclusion

In this paper, we aimed to facilitate an **accurate** and **flexible** exploration of digital continuum **network** and **compute** infrastructure trade-offs. We achieved this by *(i)* designing and implementing a 6G testbed, *(ii)* designing a method to record the performance of existing networking setups and using it to construct an open-source cellular network trace archive, *(iii)* analyzing the limitations of novel applications with state-of-practice networking infrastructure.

7.1 Answering the Research Questions

(**RQ1**) **How to design a 6G testbed?** We approached the design by constructing a set of requirements essential for an accurate 6G testbed. We used those to construct a design for a 6G testbed with separate compute and network components. The network simulation, in particular, is designed by separately simulating, based on the distinct performance properties, the core and mobile networks.

(**RQ2**) **How to implement a 6G testbed?** We implemented the 6G by using the peer-reviewed Continuum benchmark (10) for compute and state-of-the-art record-replay MahiMahi (11) for network simulation. Our implementation aims to provide a tool that can easily be configured by the end user for their particular use case. Following the principles of open science, and allowing further testbed extension along with existing experimental results reproduction, we published the testbed as an open-source artifact.

(**RQ3**) **How to characterize the performance of the digital continuum networking?** We characterized the network performance by separately collecting the performance data for core and mobile networks. We first constructed a method that can be used to collect the performance of any networking setup, and then we used it to construct an open-source cellular network trace archive containing the performance traces of real-world network infrastructure. We finished the section by analyzing the traces and gave several insights on network system-level performance trends and limitations.

(RQ4) What is the impact of cellular networks on ML applications running in the digital continuum? Using the 6G testbed and a trace archive, we constructed several experiments that evaluated the effect of the cellular network on the end-to-end application performance under different network conditions. These conditions included cellular network technology, location of offloading, type of data transmitted, and network load. Based on the results of our experiments, we identified a number of challenges for the 5G to 6G transition and a number of application-level techniques that can be used for tackling the limitations of the state-of-practice network infrastructure.

7.2 Threats to Validity

We identify a number of threats to validity (TV):

(TV1) Overestimated network capacity. Current network setup assumes that the cellular network performance is workload-independent and the same transmission slots would be available as when recording its performance by continuously transmitting network packets. Modern research (27); however, shows that the cellular network performance depends on the workload, which can lead to overestimation of network capacity by tools like *saturatr* (11).

(TV2) Insufficient time recording mobile network performance. Due to associated costs, we record mobile network traces over a short time span up to 120 seconds. Our traces do not show long-term performance trends and daily load patterns observed on mobile network infrastructure (28).

(TV3) Insufficient number of providers and geographical locations surveyed. In this paper, we recorded the traces for only a single cellular network provider - KPN - and at the same geographical location. We, as well, do not analyze how network performance changes during the network handoffs, which is relevant when the user is moving rather than staying static. Meanwhile, the cellular network performance can differ vastly based on the cellular network operator and the location (29). As well it is known that network handoffs can deteriorate experienced network performance (30).

(**TV4**) **Ignored core network bandwidth** In the current simulation setup, we assume that the limiting bandwidth factor is mobile network bandwidth, and the core network transmission time is sufficiently small to be disregarded when compared to mobile network transmission time.

(**TV5**) **Ignored core network performance variability**. While the core network has significantly lower jitter, as seen from the collected data, it can still have performance spikes associated with increased network congestion.

(**TV6**) **CPU-based experiment setup**. While our experiment setup executed the AI-based workloads on CPU, running those workloads using GPU is likely to significantly decrease the inference time (31).

(**TV7**) **Performance suboptimal AI frameworks used**. While PyTorch and Tensorflow are state-of-practice frameworks widely used both in production and research, there exist other state-of-art deep learning frameworks that show lower inference time than ones shown by PyTorch and Tensorflow (32).

(**TV8**) **Impact of higher network Quality of Service**. In this study, we do not control the quality of service selected by the internet service provider to carry the network traffic during the network performance data collection. However, the higher network quality of service reserved for real-time latency-sensitive applications makes internet service providers to prioritize the traffic, increasing experienced network performance (33).

7.3 Future Work

We identify several future research (FR) trajectories:

(**FR1**) **Implementation of high-speed jitter-aware network simulators**. During the experimentation, we found that the CPU utilization of the default MahiMahi implementation grows as the bandwidth of the emulated link increases, which makes it unfeasible to emulate high-bandwidth links.

(**FR2**) **AI inference performance** As shown in the evaluation, improvements to the network make AI inference latency an increasing concern.

(**FR3**) **Large scale cellular network trace archive** Current trace archive can be extended by collecting the performance data of multiple cellular network providers, from multiple geographic locations. The trace archive would, as well, benefit from capturing traces at timestamps with different load levels to capture cellular network performance variability associated with the daily usage patterns. Finally, the trace archive would benefit from capturing the network performance with a non-static endpoint experiencing performance effects of network handoffs.

(**FR4**) **Tackling the cellular network upload capacity and performance variability**. In our setup, it was shown that the main network limitation for the studied workloads is low upload capacity and high cellular network performance variability. Therefore, tackling the limited upload capacity of the cellular network and high performance variability is essential.

(**FR5**) **Software practices for real-time mobile network-based applications.** As identified in the evaluation, some software practices like application-level congestion control or a higher degree of input data pre-processing to decrease upload data size can help with tackling the limitations of cellular networking infrastructure.

References

[1] NLConnect. **Staat-van-de-Digitale-Infrastructuur.pdf**, 2024. Accessed: 2025-03-31. 1

[2] GSMA. **Over Half World's Population Now Using Mobile Internet**. Accessed: 2025-04-02. 1

[3] DataReportal. **Digital 2023: The Netherlands**. Accessed: 2025-04-01. 1

[4] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. **Pruning edge research with latency shears**. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, pages 182–189. ACM, 2020. 1, 16

[5] Neil C Thompson, Kristjan Greenewald, Keeheon Lee, Gabriel F Manso, et al. **The computational limits of deep learning**. *arXiv preprint arXiv:2007.05558*, **10**:1–24, 2020. 1

[6] Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn, and Pablo Villalobos. **Compute trends across three eras of machine learning**. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022. 1

[7] Matthijs Jansen, Auday Al-Dulaimy, Alessandro V Papadopoulos, Animesh Trivedi, and Alexandru Iosup. **The SPEC-RG reference architecture for the compute continuum**. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 469–484. IEEE, 2023. 2, 5, 9, 10

[8] Alexandru Iosup, Laurens Versluis, Animesh Trivedi, Erwin Van Eyk, Lucian Toader, Vincent Van Beek, Giulia Frascaria, Ahmed Musaafir,

and Sacheendra Talluri. **The atlarge vision on the design of distributed systems and ecosystems**. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1765–1776. IEEE, 2019. 5

[9] John Ousterhout. *A Philosophy of Software Design*. Yaknyam Press, 2018. 5

[10] Matthijs Jansen, Linus Wagner, Animesh Trivedi, and Alexandru Iosup. **Continuum: Automate infrastructure deployment and benchmarking in the compute continuum**. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, pages 181–188. ACM, 2023. 5, 6, 7, 9, 10, 21, 36, 47

[11] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. **Mahimahi: Accurate Record-and-Replay for HTTP**. *USENIX Annual Technical Conference*, pages 177–189, 2015. 5, 10, 14, 22, 27, 28, 35, 47, 48

[12] Ahmed Ali-Eldin, Bin Wang, and Prashant Shenoy. **The hidden cost of the edge: a performance comparison of edge and cloud latencies**. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2021. 5, 29

[13] John Ousterhout. **Always measure one level deeper**. *Communications of the ACM*, **61**(7):74–83, 2018. 5

[14] Alexandru Iosup, Fernando Kuipers, Ana Lucia Varbanescu, Paola Grosso, Animesh Trivedi, Jan Rellermeyer, Lin Wang, Alexandru Uta, and Francesco Regazzoni. **Future computer systems and networking research in the netherlands: A manifesto**. *arXiv preprint arXiv:2206.03259*, 2022. 6

[15] **tc(8) - Linux manual page**, n.d. Accessed: 2025-06-04. 9

[16] Evolved Universal Terrestrial Radio Access. **Medium Access Control (MAC) Protocol Specification (3GPP TS 36.321 version 10.1.0 Release 10)**. *ETSI TS*, **136**(321):V11, 2011. 14

[17] Christos Bouras, Charalampos Chatzigeorgiou, Vasileios Kokkinos, Apostolos Gkamas, and Philippos Pouyioutas. **Optimizing Network Performance in 5G Systems with Downlink and Uplink Decoupling**. In *2023 6th*

International Conference on Advanced Communication Technologies and Networking (CommNet), pages 1–6. IEEE, 2023. 15

[18] Andrew S. Tanenbaum, Nick Feamster, and David J. Wetherall. *Computer Networks*. Pearson, Harlow, England, 6 edition, 2021. 16

[19] Ericsson. **6G Use cases: Beyond communication by 2030**, 2024. Accessed: 2025-06-16. 17

[20] Docker, Inc. **What is a Container?** `https://www.docker.com/resources/what-container/`, 2025. Accessed: 2025-07-10. 22

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. **PyTorch: An Imperative Style, High-Performance Deep Learning Library**. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 22

[22] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems**. `http://download.tensorflow.org/paper/whitepaper2015.pdf`, 2015. Accessed: 2025-07-10. 22

[23] OASIS Standard. **MQTT Version 5.0**. `https://mqtt.org/mqtt-specification/`, 2019. Accessed: 2025-07-10. 26

[24] Robert Underwood, Jason Anderson, and Amy Apon. **Measuring network latency variation impacts to high performance computing application performance**. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 68–79. ACM, 2018. 31

[25] Uwe Bauknecht and Tobias Enderle. **An investigation on core network latency**. In *2020 30th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–6. IEEE, 2020. 31

[26] M. Anafaa, I. Shayea, J. Din, M. H. Azmi, A. Alashbi, Y. I. Daradkeh, and A. Alhammadi. **6G mobile communication technology: Requirements, targets, applications, challenges, advantages, and opportunities**. *Alexandria Engineering Journal*, **64**:245–274, 2023. 45

[27] William Sentosa, Balakrishnan Chandrasekaran, P. Brighten Godfrey, and Haitham Hassanieh. **CellReplay: Towards accurate record-and-replay for cellular networks**. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 1169–1186, Philadelphia, PA, apr 2025. USENIX Association. 48

[28] Huandong Wang, Fengli Xu, Yong Li, Pengyu Zhang, and Depeng Jin. **Understanding Mobile Traffic Patterns of Large Scale Cellular Towers in Urban Environment**. In *Proceedings of the 2015 Internet Measurement Conference (IMC)*, pages 225–238. ACM, 2015. 48

[29] Ahmet Yildirim, Engin Zeydan, and Ibrahim Onuralp Yigit. **A statistical comparative performance analysis of mobile network operators**. *Wireless Networks*, **26**(2):1105–1124, 2020. 48

[30] Anne Fladenmuller and Ranil De Silva. **The effect of Mobile IP handoffs on the performance of TCP**. *Mobile Networks and Applications*, **4**:131–135, 1999. 48

[31] Hyperstack. **AI Inference Optimisation: Examples, Techniques, Benefits and more**, 2025. Accessed: 2025-07-10. 49

[32] Berk Ulker, Sander Stuijk, Henk Corporaal, and Rob Wijnhoven. **Reviewing Inference Performance of State-of-the-Art Deep Learning Frameworks**. In *Proceedings of the 23rd International Workshop on Software and Compilers for Embedded Systems (SCOPES '20)*, pages 63–72. ACM, 2020. 49

[33] **3GPP TS 23.203: Policy and Charging Control Architecture**. Technical Report TS 23.203, 3rd Generation Partnership Project (3GPP), 2024. Version 18.5.0, Release 18. 49

Appendix A

Reproducibility

A.1 Abstract

GitHub repository

A.2 Artifact check-list (meta-information)

Obligatory. Use just a few informal keywords in all fields applicable to your artifacts and remove the rest. This information is needed to find appropriate reviewers and gradually unify artifact meta information in Digital Libraries.

- **Program: Continuum, MahiMahi**
- **Compilation: none. Python 3.9 used for interpretation**
- **Binary: none**
- **Model: MobileNetV2, MarianMT**
- **Run-time environment: QEMU**
- **Hardware: Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz, 256 GB**
- **Run-time state: clean**
- **Execution: Python**
- **Metrics: network, compute latency**
- **Experiments: network generation, continuum offloading, text and image transmission, scalability under load**
- **How much disk space required (approximately)?: 5 GB**
- **How much time is needed to prepare workflow (approximately)?: 0**

- **How much time is needed to complete experiments (approximately)?: 5 minutes per experiment**
- **Publicly available?: Yes**
- **Code licenses (if publicly available)?: None**
- **Data licenses (if publicly available)?: None**
- **Workflow framework used?: Ansible**
- **Archived (provide DOI)?: No**

A.3 Description

How to access

You can access 6G testbed through GitHub. You can access the mobile network performance traces through Google Drive.

Software dependencies

LibVirt, Ansible, Kubernetes, Docker

Data sets

Experiments used for last-mile network simulation, network traces available through Google Drive.

Experimental hardware

A.4 Experiment customization

Experiments can be customized by specifying your mobile network trace, your core network latency, or by specifying your endpoint and cloud, and edge node Docker containers.

A.5 Notes

The wireless network presets used are:

- `lte_nl_kpn_mahimahi` for KPN LTE
- `5g_obstacled_nl_kpn_mahimahi` for obstacled KPN 5G
- `5g_nl_kpn_mahimahi` for unobstacled KPN 5G

- `4g_us_verizon_mahimahi` for Verizon 4G

The used offloading location presets are:

- **base_edge** for base edge
- **eu_central_1** for EU-Central-1 located in Paris, France, representing the local cloud
- **eu_west_3** for EU-West-3 located in Paris, France, representing the regional cloud
- **us_east_1** for US-East-1 located in Virginia, USA, representing the transcontinental cloud