TECHNISCHE UNIVERSITÄT BERLIN

A Software Testbed for Drone Anti-Collision Systems Leveraging 6G Edge Computing

Master's Thesis

Author
Jan Beckschewe
401385
jan.beckschewe@campus.tu-berlin.de

Advisor
Tobias Pfandzelter

Examiners
Prof. Dr.-Ing. David Bermbach
Prof. Dr. habil. Odej Kao

Technische Universität Berlin, 2024
Faculty IV – Electrical Engineering and Computer Science
Scalable Software Systems Research Group

A Software Testbed for Drone Anti-Collision Systems Leveraging 6G Edge Computing

Master's Thesis

Submitted by:
Jan Beckschewe
401385
jan.beckschewe@campus.tu-berlin.de

Technische Universität Berlin
Faculty IV – Electrical Engineering and Computer Science
Scalable Software Systems Research Group

2024

Hiermit versichere ich, dass ich die vorliegende Arbeit eigenständig ohne Hilfe Dritter und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe. Alle Stellen die den benutzten Quellen und Hilfsmitteln unverändert oder sinngemäß entnommen sind, habe ich als solche kenntlich gemacht.

Sofern generische KI-Tools verwendet wurden, habe ich Produktnamen, Hersteller, die jeweils verwendete Softwareversion und die jeweiligen Einsatzzwecke (z. B. sprachliche Überprüfung und Verbesserung der Texte, systematische Recherche) benannt. Ich verantworte die Auswahl, die Übernahme und sämtliche Ergebnisse des von mir verwendeten KI-generierten Outputs vollumfänglich selbst.

Die Satzung zur Sicherung guter wissenschaftlicher Praxis an der TU Berlin vom 15. Februar 2023* habe ich zur Kenntnis genommen.

Ich erkläre weiterhin, dass ich die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

(Unterschrift) Jan Beckschewe, Berlin, 26. November 2024

Verwendete KI-Tools:

- OpenAI ChatGPT-4o† Release vom 8. August 2024. Einsatzzweck: Sprachliche Verbesserung der Arbeit
- DeepL Übersetzer‡. Einsatzzweck: Übersetzung der Kurzfassung

* https://www.static.tu.berlin/fileadmin/www/10002457/K3-AMBl/Amtsblatt_2023/Amtliches_Mitteilungsblatt_Nr._16_vom_30.05.2023.pdf

† https://chatgpt.com/

‡ https://www.deepl.com/translator

Abstract

UAVs, commonly known as *drones*, are increasingly utilized across various industries. Integrating cellular connectivity addresses the range limitations of traditional direct wireless links but introduces latency challenges when UAVs rely on centralized data centers. UAV edge computing offers a solution to both issues, yet testing such systems remains challenging.

This work introduces a novel approach to bridge the testing gap in UAV edge computing. We propose a software testbed architecture that simulates critical components of the UAV edge ecosystem, driven by scenario-based configurations for consistent and repeatable experiments.

A proof-of-concept implementation of this design is presented, featuring a map-based user interface and collision detection capabilities. Performance benchmarks evaluate its scalability and functionality, including simulation of UAV network connections relative to cell tower proximity, and the collision detection effectiveness. Results demonstrate that the testbed can manage several hundred concurrent UAVs, making it a robust tool for testing UAV edge computing scenarios.

Kurzfassung

UAVs, gemeinhin bekannt als *Drohnen*, werden zunehmend in verschiedenen Branchen eingesetzt. Die Integration von Mobilfunkverbindungen behebt die Reichweitenbeschränkungen herkömmlicher drahtloser Direktverbindungen, führt jedoch zu Latenzproblemen, wenn UAVs auf zentrale Rechenzentren angewiesen sind. Edge Computing für UAVs bietet eine Lösung für beide Probleme, doch das Testen solcher Systeme bleibt herausfordernd.

In dieser Arbeit wird ein neuartiger Ansatz vorgestellt, um die Lücke beim Testen von Edge Computing für UAVs zu schließen. Wir schlagen eine Software-Testbett-Architektur vor, die kritische Komponenten des UAV-Edge-Ökosystems simuliert und auf szenariobasierte Konfigurationen setzt, um konsistente und wiederholbare Experimente zu ermöglichen.

Es wird eine Proof-of-Concept-Implementierung dieses Designs vorgestellt, einschließlich einer kartenbasierten Benutzeroberfläche und Kollisionserkennung. Leistungsbenchmarks bewerten die Skalierbarkeit und Funktionalität, einschließlich der Simulation von UAV-Netzwerkverbindungen in Abhängigkeit von der Nähe zu Mobilfunkmasten, sowie die Effektivität der Kollisionserkennung. Die Ergebnisse zeigen, dass das Testbett in der Lage ist, mehrere hundert UAVs gleichzeitig zu verwalten, und sich somit als robustes Werkzeug für die Erprobung von Szenarien im UAV Edge Computing erweist.

Contents

Acronym List

UAV Unmanned Aerial Vehicle

vUAV virtual UAV

SUT System under Test

gNB gNodeB

UE User Equipment

IoT Internet of Things

eBPF extended Berkeley Packet Filters

TCAL TC Abstraction Layer

DP-MIMO Dual-Polarized Multiple Input Multiple Output

CLI Command Line Interface

AWS Amazon Web Services

GCP Google Cloud Platform

EC2 Elastic Compute Cloud

MEC Multi-access Edge Computing

VM Virtual Machine

VLOS Visual Line of Sight

BVLOS Beyond Visual Line of Sight

B5G Beyond 5th Generation

GUI Graphical User Interface

VPN Virtual Private Network

SITL Software in the Loop

GCS Ground Control Station

REST Representational State Transfer

API Application Programming Interface

HTTP Hypertext Transfer Protocol

TCP Transmission Control Protocol

UDP User Datagram Protocol

URLLC Ultra-Reliable Low-Latency Communication

1 Introduction

Unmanned Aerial Vehicles (UAVs), commonly referred to as *drones*, vary widely in shape and size, ranging from small quadcopters to large fixed-wing aircraft. They are rapidly gaining traction in a broad range of applications in various industries, including logistics, agriculture, and security [Sha+19]. Traditionally, these UAVs are typically connected to the operator via a dedicated radio frequency, limiting range to a few kilometers or less, and often requiring Visual Line of Sight (VLOS) [Ahm+22]. Using a cellular network connection mitigates this issue, because 4G, 5G, and future Beyond 5th Generation (B5G) networks are ubiquitous in many locations, especially in urban areas. Cellular UAVs allow for simplified Beyond Visual Line of Sight (BVLOS) operations, and enable offloading parts of the computations when flying semi or fully autonomously, keeping weight and power demand on the UAV itself low. When the reliance on outside computation is this great, reliability and low latency become critical. One approach to mitigate these issues is to use edge computing, since it brings the hardware for the computations into direct proximity of the cell tower [DLN23].

However, testing UAV software for edge environments is currently difficult. Several systems need to interact: the software and hardware of the UAV itself, the software and hardware of the cell towers, as well as the wireless link. McEnroe et al. [MWL22] highlight inadequate testing tools as a key barrier to widespread adoption of edge computing in UAV applications.

Addressing this gap could yield substantial benefits: Enhanced testing capabilities would not only accelerate software development, reducing costs, but also increase safety and reliability. Both of these are crucial, due to UAVs' potential proximity to other airspace users, and the risks posed to life and property on the ground. One could imagine defining scenarios, where the software is evaluated based on whether it is able to prevent an imminent collision through a last-second avoidance maneuver, and whether the collision can still be avoided in an area with poor wireless connection quality.

In this thesis, we will address the edge UAV testing gap by introducing a software testbed. However, we do not intend to emulate or simulate every part of an edge UAV system, instead we focus on a few key subsystems, as seen in Figure 1.1. We simulate the movement of virtual UAVs (vUAVs) of any shape and size. We use the positions of the vUAVs, as well as the positions of virtual cell towers, to calculate the expected network parameters. We can then use the network parameters to emulate the cellular connection and communication protocols between the vUAVs and the System under Test (SUT). The vUAVs are executed within virtualized environments, such as containers. Furthermore, we allow for the creation of scenarios to test the SUT under different conditions while being repeatable. Finally, the testbed includes a map visualization and collision detection.

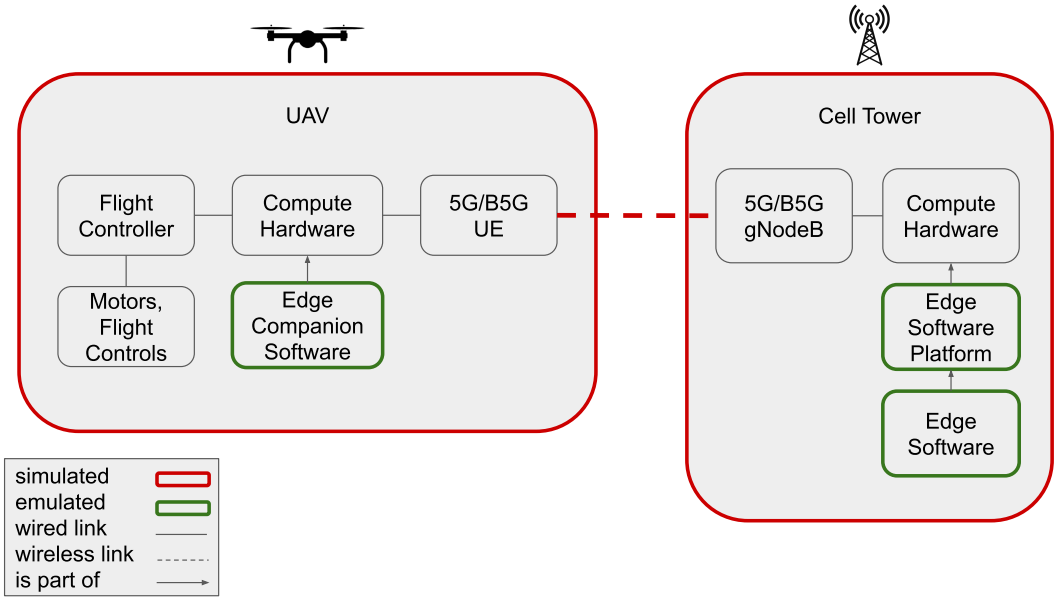

Figure 1.1: The Scope of our testbed

In this work, we make the following contributions:

1. We survey existing literature on related concepts, and on software testbeds for similar systems, to identify requirements for a UAV edge computing testbed (Section 2, Section 3)
2. We introduce *SkyBed*, a software testbed architecture that addresses the demands posed by testing edge UAV applications (Section 4)
3. We evaluate SkyBed through a proof-of-concept prototype by benchmarking it on efficacy, repeatability, and scalability (Section 5)
4. We discuss our results and limitations, and how future work might address these limitations (Section 6)

We make our implementation of SkyBed, along with instructions on how to deploy and use it, available as open source [Bec24], with the intention of aiding future researchers, who are in the process of developing their own edge UAV applications. By offering this resource, we hope to broaden access to edge UAV technology and spark further innovation and exploration in this emerging field.

2 Background

In this section we provide background information on the core technologies and concepts underlying this research, including UAVs, network simulation tools, and advancements in edge computing and 5G. These topics lay the foundation for understanding the technical and contextual landscape of the work.

2.1 Network Emulators, Simulators and Testbeds

Software testbeds are a category of system designed to test other software and hardware systems in an isolated fashion. We provide a distinction from other similar categories to clarify the goals, advantages, and drawbacks of software testbeds. Gomez et al. [Gom+23] provide a survey of network simulators, network emulators, and network testbeds and also give a comprehensive distinction between the categories:

Network simulators are built on discrete models to replicate real hardware behavior, usually without employing the same software used on the real hardware. They support virtual time, so they may run at a slower or faster pace than real-time. They provide the best reproducibility, but their results are most likely to deviate from the real world because they are always based on models of other systems. Network simulators are categorized as either time-based or discrete-event-based. Time-based simulators progress in fixed time slots, executing events incrementally, making them well suited for IoT applications, transport protocols, and congestion control algorithms. Discrete-event simulators, in contrast, progress by executing scheduled events triggered by conditions like node activation or energy changes and are used, for instance, when simulating the physical layer of optical or radio networks.

Network emulators work in steady time and directly execute the software used in the real world. They often use operating system kernel features to implement routers, hosts, and other network devices. Full system emulators build upon Virtual Machines (VMs) connected via virtual switches to represent network devices, whereas container based emulators only isolate on a process level, using fewer resources.

Network testbeds offer both real and virtualized hardware. Similar to network emulators, they operate at steady time and execute real-world software, but often use hardware for a particular subset of the workload. Testbeds tend to give the most realistic results out of the three, because of their high use of real-world components. However, they are limited in their flexibility and lead to higher costs due to the specialized hardware needed.

For the purposes of this work, we define software testbeds as incorporating some aspects of network emulators, namely the execution of the real software but also incorporating aspects of simulators, such as the simulated movement of hardware devices. This creates a system that in some ways behaves like a network testbed, but which can be used without involving any tailored hardware.

Correspondingly, software testbeds provide a similar level of realism to network emulators; a middle ground between hardware testbeds, which tend to perform very good in this category, and simulators, which sacrifice realism in favor of flexibility [WSW05].

2.2 5G and Beyond 5th Generation Networks

5G networks are designed to provide high-speed, low-latency communication with wide coverage. In these networks, User Equipment (UE), which are the client devices, connect to gNodeBs (gNBs), which are the radio components of cell towers.

Features like network slicing and Ultra-Reliable Low-Latency Communication (URLLC) address diverse application needs. Network slicing creates virtual networks with dedicated resources, isolated from other traffic. URLLC ensures fast and reliable communication for time-critical operations. However, these features have so far only seen limited adoption in specific industrial applications [Kha+22].

Despite its progress, 5G has shortcomings in achieving sub-millisecond latency, precise localization, and seamless handovers in dynamic environments [Hag+23]. These gaps highlight the potential of 6G, which aims to further reduce latency, and enable more precise and reliable connectivity for future advancements [Ger+22].

2.3 Edge Computing

Edge computing is an emerging computing paradigm that aims to bring processing and storage resources closer to the data source. The "edge" in this case refers to either any node on the path between the data source and the central cloud [Shi+16], or the last network hop before the user device [You+19], depending on the definition. This is in contrast to the more established *cloud computing*, where computations are typically performed in large centralized data centers. The primary goals of edge computing are often reduced latency, improved resiliency, reduced power consumption and reduced bandwidth demands [Shi+16].

There are several related and partially overlapping computing paradigms with similar goals, such as *fog computing*, *Multi-access Edge Computing (MEC)*, *mobile cloud computing* and *cloudlets*. Their overlap in scope is shown in Figure 2.2 based on [You+19]. They all fall on a continuum in terms of how physically close to the end user they operate. They are backed by various groups, such as telecom providers, the *hyperscalers*, such as Amazon Web Services (AWS) and Google Cloud Platform (GCP), and other industrial and academic entities. Some of these are standardized by organizations, such as 3GPP, which has standardized all recent major cellular networks, including 5G [You+19][Mao+17].

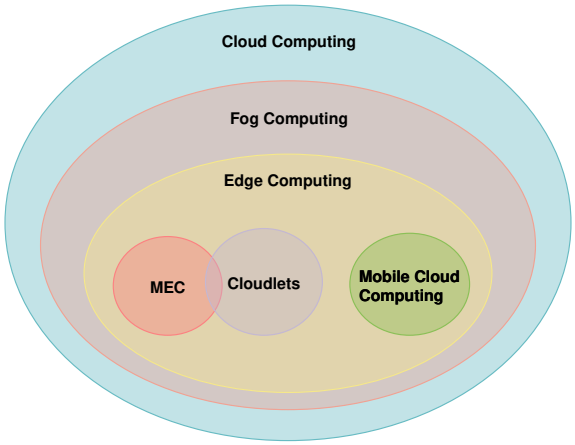

Figure 2.2: A Classification of the Scope of Paradigms related to Edge Computing

A common use case for edge computing are Internet of Things (IoT) devices, such as those used in smart homes and industrial automation, and autonomous vehicles. In these areas, real-time applications are common, necessitating very low latencies, often not achievable using centralized cloud computing [Por+18].

2.4 UAVs on the Edge

Edge computing is increasingly being integrated with UAVs to address the computational and communication challenges faced during UAV missions. UAVs commonly need to collect and analyze data in real time, yet they are limited by on-board processing power, battery life, and network range. Edge computing offers a practical solution by enabling UAVs to offload processing tasks to nearby compute devices at the network edge, which are located closer than traditional cloud servers [Zho+18].

This offloading of tasks to the edge minimizes latency, which is crucial for real-time applications where rapid decision-making is essential. By processing data locally at the edge, UAVs can respond more quickly to dynamic conditions, such as sudden obstacles or changes in mission parameters, while minimizing the delay associated with cloud-based processing. Additionally, by shifting computational demands away from the UAV, edge computing conserves battery life, allowing UAVs to stay operational for longer periods of time or carry heavier payloads [Yan+19].

Edge computing also reduces the need for constant data transmission to the cloud, which helps optimize bandwidth usage. This localized processing is especially useful for data-intensive tasks, such as video analysis or AI-based image recognition, where only essential or filtered information is sent back to the cloud or command center, reducing network congestion [Yaz+21].

One use case of UAV edge computing is currently being developed by *6G NeXt* [Mel+23a] [Mel+23b][Pfa+24]. They intend to create an anti-collision system, which uses digital twins to evaluate all flight paths in an area on the edge. If a potential collision is identified, evading trajectories are calculated and sent to the UAV.

2.5 Compute Isolation Models

The isolation of software tasks is a fundamental requirement that has driven the development of various solutions, each of which balances specific trade-offs. These solutions can be conceptualized along a spectrum, as depicted in Figure 2.3. At one end of the spectrum, threads represent minimal resource duplication, while progressively advanced models provide increasing levels of resource segregation. At the other end, VMs emulate an entire hardware system.

Virtual machines offer notable advantages. They enhance compatibility by allowing software to run independently within a self-contained environment. Scalability is another benefit, as VMs can be replicated across multiple physical machines to distribute computational workloads. Additionally, VMs provide strong isolation, mitigating the impact of threats like malware by containing their effects within the virtualized environment [SR05].

Despite these benefits, the comprehensive emulation performed by VMs incurs performance trade-offs. Virtual machines require additional operational overhead, which can slow down primary tasks. Furthermore, their startup time is significant, often taking several seconds to boot an entire operating system, whereas threads can be spawned at a much faster rate, numbering in the thousands per second.

Containers strike a balance between performance and isolation. They leverage the host operating system's kernel while isolating applications running on top of it. Among containerization technologies, *Docker [Doc13]* is the most widely adopted.

MicroVMs aim to bridge the gap between containers and traditional VMs. They offer the strong isolation of VMs, including their own kernel, while maintaining performance levels comparable to containers. A prominent implementation of this approach is *Firecracker [Aga+20]*.

Figure 2.3: The Spectrum of Compute Isolation

3 Related Work

In this section, we review prior research that informs and relates to our work. By examining these studies, we aim to contextualize current approaches and identify key gaps in the field, which this research seeks to address. This review provides a framework for understanding how our contributions build upon and differ from existing solutions. Table 3.1 provides insight into which works support which features.

3.1 Edge Testbeds

There are several testbeds for different use cases leveraging edge computing.

Celestial [PB22] is a software testbed for the software running on satellites in low earth orbit. It can scale up to planet-scale constellations, such as *Starlink*'s constellation [Han19], which contains thousands of satellites in different orbits. Scalability is achieved by deploying one *Firecracker* microVM [Aga+20] per satellite, leveraging their benefits, such as the ability to run their own Linux kernels, which provides greater flexibility for the SUT. MicroVMs offer performance comparable to containers, though with the drawback of being more challenging to implement [PB24], when compared to the mature and user-friendly ecosystem of containers. The authors of Celestial initially used tc-netem [Hem11] for network throttling, but later issued a note of caution [Bec+22], that tc-netem's performance can become a limiting factor at the scale of tens of thousands of virtual links. In their work, they deploy only thousands of satellites, yet they still ran into performance issues because a single satellite is connected to several neighboring satellites. They were able to overcome their scaling issues using extended Berkeley Packet Filters (eBPF). Celestial is the only work we found that has developed a way to test edge software, while also encompassing mobility calculations for the virtual hardware, which is why this work's architecture is primarily inspired by Celestial, despite it being incompatible with UAVs.

MockFog in version 1 [Has+19] and version 2 [HGB23] aims to emulate fog computing infrastructure in the cloud. It achieves this by deploying one VM in the cloud for each fog node it aims to emulate. Both versions support AWS Elastic Compute Cloud (EC2) [Ama06] and version 1 also supports *OpenStack* [Ope10] as its cloud provider. However, they clarify that their architecture is vendor agnostic, and only their proof-of-concept relies on these particular services. Version 1 is designed around a web-based Graphical User Interface (GUI) with a discrete graph with VMs as nodes and links as edges, whereas version 2 focuses on automation and automated repeatable experiments, which they generate using templating files. As an example, the new version supports intentionally dropping links after a specified time to simulate outages and once service level violations are detected, change workload patterns. Relying on the cloud comes with the advantage of near-unlimited scalability due to the vast resources of the hyperscalers. However, it comes at a substantial financial cost, when experimenting with many nodes. Furthermore, efficient compute resource utilization becomes challenging when each emulated device (such as a single vUAV) only uses a small fraction of a CPU thread, and the smallest unit of payment is a single vCPU.

Kollaps [Gou+20][AMS24] is fully decentralized, leaving the hardware deployment to the user, unlike MockFog. It implements *network collapsing* as a technique to avoid having to emulate the entire network topology. Instead, it emulates only the delay, data rate and similar network parameters, which would result if the topology was fully emulated, to reduce complexity. It performs the emulation on a per-container basis, and because of its decentralized design, also across VMs. Its dynamic topology emulation feature allows

changing the network characteristics at runtime based on values set in a configuration file. Similar to Celestial, Kollaps initially used tc-netem [Hem11] for its network emulation, but switched to eBPFs in a later release. Kollaps includes a TC Abstraction Layer (TCAL) around it, to not only set new link properties, but also to monitor current link utilization. Furthermore, the authors provide a survey of previous network emulation tools and their architecture, as well as which network parameters can be emulated statically or dynamically. One limitation of Kollaps is that, although it supports dynamic topology emulation, it relies on preconfigured values and lacks an API for fully dynamic behavior. This makes it unsuitable if we expect the to-be-emulated devices to make their own decisions regarding movement and thus indirectly regarding network parameters.

EmuEdge [ZCS19] also supports container-based throttling, and adds support for experiments with real cellular hardware. However, although the authors promised mobility support and integration of simulation tools, it appears that this feature was never added.*EmuFog* [May+17] also supports distributed testing of applications in Docker containers. It is based on *MaxiNet* [Wet+14], a distributed extension of *Mininet* [LHM10], a network emulator designed for the constrained resources of a single contemporary laptop. However, it lacks dynamic topologies entirely. OpenLEON [Fia+19], which is based on *Containernet* [PKV16], focuses purely on MEC and does not support similar paradigms. *iFogSim* [Gup+17], *Sphere* [Fer+20] and *Faas-sim* [Rai+23] are pure simulators and lack the emulation capability we are looking for.

The authors of Celestial and MockFog also give advice on how to build an edge testbed in general, based on their experience of building a total of five such systems [PB24]. In particular, they argue that building a new testbed for a specialized use case often takes a similar amount of time as familiarizing oneself with existing testbeds and adapting them to one's needs. They recommend building testbeds for a specific use case and seeing testbeds "as a means to an end rather than a standalone product". Additionally, they argue that the use of GUIs as a means of configuring experiments should be avoided in favor of machine usable interfaces and configuration scripts because they allow for automated and easily repeatable experiments. The authors believe that GUIs in testbeds should only be as a means of visualizing experiments to show the emulated infrastructure and to verify the correctness of the setup.

3.2 UAV Network Testbeds

FlyNetSim [BSL18] acts as an intermediary between UAVs with *ArduPilot* [Mun+09] software, and a Ground Control Station (GCS). The authors use *ns-3* [RH10] to simulate either a 4G or a Wi-Fi connection. Specifically, FlyNetSim interferes with the publish-subscribe based communication between the UAV and GCS by capturing and retransmitting messages. The UAVs are simulated as individual threads. Additionally, FlyNetSim supports hardware testbeds with real UAVs. However, since it emulates the message layer, rather than the network layer, compatibility is limited. Only UAVs equipped with ArduPilot software and the corresponding GCS are supported. Another limitation is that FlyNetSim only emulates control messages, so video streams and similar traffic cannot be emulated.

OpenUAV [Sch+18] emulates the software of UAVs inside Docker containers, with an emphasis on its use as a testing tool for machine learning algorithms, and for swarms of UAVs. One container can contain multiple UAVs, OpenUAV uses containers only to enable multi-tenancy. The testbed operates on cloud-based hardware, to reduce barriers

for new users. However, because OpenUAV assumes that the UAVs have powerful onboard compute capabilities, it does not perform any throttling of virtual network connections.

3.3 5G and Beyond 5th Generation Network Simulation and Emulation

Simu5G [Nar+20a] [Nar+20b] is a simulator for 5G networks within the *OMNeT++* [Var10] simulation environment. It is used for real-time emulation [NSV21], as well as to create a testbed for MEC [Nar+20c] applications. However, creating new scenarios requires familiarity with the complex configuration and unintuitive interface, and the existing MEC scenario requires the application developer to adapt their code base to follow the ETSI standards on MEC [ETS24].

5G-LENA [Pat+19] is a module to add 5G radio support to the network simulator *ns-3* [RH10]. It includes several example scenarios that can be adapted to different needs and executed via an API. Furthermore, Gomez [Gom23] builds upon 5G-LENA and uses ns-3 tap devices to act directly as an emulator, eliminating the need for an additional emulator, such as tc-netem. This is advantageous, because it reduces development complexity. However, the disadvantage of this approach is, that it cannot adapt to changing network conditions over time, resulting in time gaps between emulation cycles.

Criteria	Celestial [PB22]	MockFog [HGB23]	Kollaps [Gou+20]	EmuEdge [ZCS19]	EmuFog [May+17]	OpenLEON [Fia+19]	iFogSim [Gup+17]	Faas-sim [Rai+23]	Sphere [Fer+20]	Simu5G [Nar+20b]	5G-LENA [Pat+19]	FlyNetSim [BSL18]	OpenUAV [Sch+18]	SkyBed
Emulation	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗	✓	✓	✓
Simulation	✓	✗	✗	✗[1]	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
Hardware Support	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗
Visualization	✓	✓	✓	✗	✓	✗	✓	✗	✓	✓	✓	✓	✓	✓
For any Edge Computing Use Case	✗	✓	✓	✓	✓	✗[2]	✓	✓	✓	✗	✗	✗	✗	✗
Pre-planned Dynamic Topology	✓	✗	✓	✗	✗	✗	✓	✓	✓	✓	✓	✓	✗	✓
Real-time Dynamic Topology	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✓
Open Source	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Platform	Linux	EC2	Linux	Linux	any	Linux	any	any	Linux	any	any Unix	Linux	Linux	Linux
Container Support	✗	✗	✓	✓	✓	✓	✗	✗	✗	✓	✗	✗	✓	✓
VM Support	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Automated Cloud Deploy	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Distributed	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
uses tc-netem	✗[3]	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Table 3.1: Comparison of Testbeds

[1] Appears to be supported according to graphics, but actually marked as future work
[2] Only supports MEC
[3] Initially used tc-netem, then switched to eBPFs

4 SkyBed: a Software Testbed Architecture

In this section, we will present the architecture of SkyBed, a software testbed designed specifically for testing UAV software running on edge hardware. The architecture is illustrated in Figure 4.4. Its main components are the orchestrator, which manages the vUAVs, the SUT, which can be executed locally, or connected over a network, and the vUAVs themselves, each of which has a virtual position, and communicates with the orchestrator.

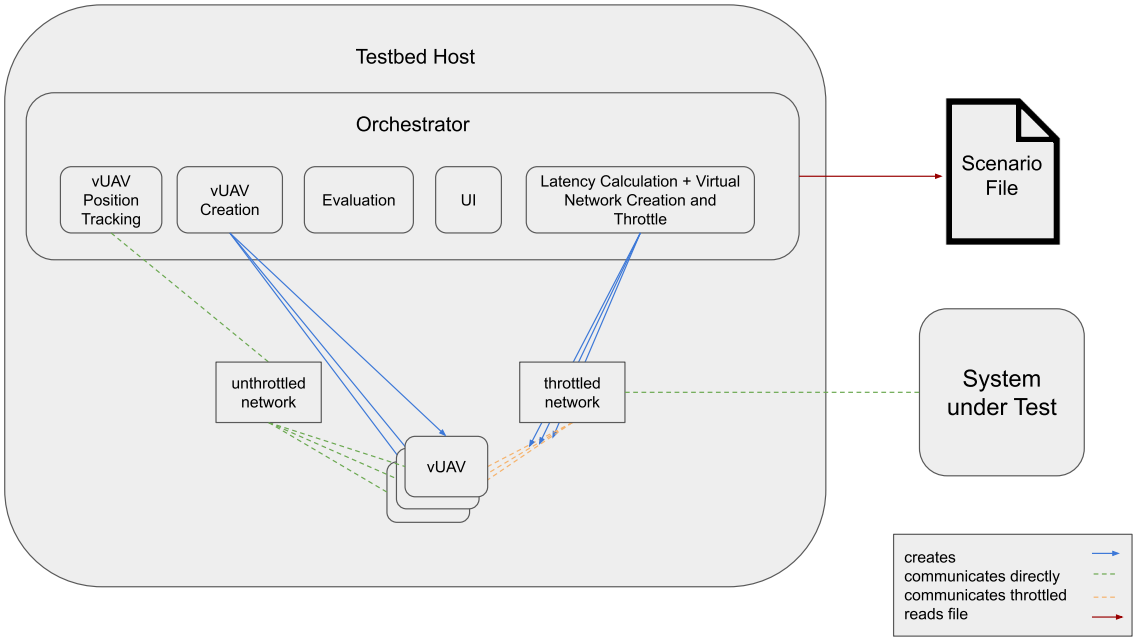

Figure 4.4: The Software Testbed Architecture

Scenarios Scenarios are typically text files specifying the virtual setup to be tested. They include the initial UAV positions, headings and speed, as well as the cell tower positions, and configuration parameters for the simulation itself, such as duration and simulation accuracy.

The Orchestrator The orchestrator generates the vUAV instances at the count specified by the scenario and with the corresponding position and route data, keeping track of which vUAV is throttled by which identifier, such as IP address. The vUAVs can use most forms of isolation from the orchestrator, such as processes, containers or (micro)VMs, that support individual OS-level traffic shaping, which is a mechanism used by the kernel to drop or hold back filtered packets for a limited time.

Our testbed establishes two virtual networks, one network where throttling can be enabled and one where it remains disabled. The unthrottled network is used to transmit the vUAVs positions to the orchestrator. This is important because not only is the throttled connection potentially much slower, but it can also be set to drop all packets when it is out of range of any virtual cell tower. In that case, the orchestrator could not be notified upon reentering cell coverage, because the way to notify the orchestrator would involve the very network that was set to drop all packets. Instead, the throttled network is used only for the communication between the vUAVs and the SUT.

To determine the correct network parameters, such as delay and data rate of the virtual connections, a simulation is performed. This could be a simulation of a full cellular network including interference on the radio link and delays along the path from the antenna to the edge computing node, or a more focused simulation considering only the radio transmission.

After determining the network parameters, the throttling can be performed by any network emulator capable of intercepting the kernel's packet handling, such as tc-netem [Hem11] or *Dummynet* [Riz97]

The vUAVs Each vUAV tracks its own virtual position, in order to send it to the orchestrator via the direct link, and to the SUT via the throttled link. In addition, the vUAVs listen to trajectory update messages from the SUT using the throttled link. The main benefit of this approach is that it ensures universal compatibility with any vUAV (or real UAV) implementation, which can use any custom protocol to communicate with the edge function implementation, separate from the orchestrator implementation.

Isolation of vUAVs from each other can be performed in any of the ways outlined in Section 2.5: using threads, processes, containers, or (micro)VMs. Baidya et al. [BSL18] highlight some advantages of each approach, when applied to UAV testbeds. However, their system can only be implemented using thread-level isolation, because it requires synchronization between UAVs. This limitation does not exist in SkyBed, so containers and VMs would be conceivable, allowing for the simulation of more features closer to the hardware of the UAVs.

Optional Features Optional features may include some form of GUI to display and, if necessary, modify the position and properties of objects within the simulation. Additionally, certain ways of evaluating and capturing the performance of UAVs, edge hardware, and the wireless link may be added.

5 Evaluation

We evaluate our approach through a proof-of-concept implementation and a number of experiments in which we benchmark *SkyBed*.

5.1 Proof-of-Concept Implementation

Analogous to the architecture, in this section we present our implementation, with its components illustrated in Figure 5.5 and its processes shown in Figure 5.7.

Our testbed implementation uses Docker containers to isolate the vUAVs. This decision factors in several considerations: Thread-level isolation is incompatible with tc-netem because it cannot distinguish between different traffic sources and targets. Process-level isolation would require cgroups [Heo15], adding complexity and potential incompatibilities with vUAV software, such as differing dependency versions. Traditional VMs are also impractical due to their high demands on compute and memory resources. This leaves containers and microVMs as viable options, where we chose containers because Pfandzelter and Bermbach [PB24] found microVMs to be more difficult to implement. MicroVMs could be explored in future work for potential benefits in isolation, in particular it supports a network stack that is fully separated from the host, which could be beneficial for UAV applications.

Each container is assigned two IP addresses: one for the throttled virtual network and another for the unthrottled network. The unthrottled network is restricted from accessing the internet beyond the Docker host, ensuring that vUAV traffic cannot bypass the network emulation when the SUT is a different machine. Additionally, if the SUT is located on the same machine, the emulation remains intact since vUAVs use the throttled network's gateway IP instead of localhost, preventing any bypass.

The orchestrator queries the position of the vUAVs using a protocol based on a small REST API over HTTP. Each response message contains the vUAV's position, direction, and speed. REST was chosen over alternatives such as message brokers or direct UDP/TCP connections, because it is straightforward to implement with tools like FastAPI [Ram18]. Additionally, REST provides flexibility to easily accommodate new data types from vUAVs in the future. Unlike some message brokers, such as *Kafka* [Apa12], which lack a built-in way to discard outdated undelivered messages, HTTP enables the simple use of timeouts. For SkyBed, this timeout approach is preferable because it signals a clear malfunction by preventing delayed and potentially inaccurate data from arriving; no data is considered better than misleading data.

To determine the network parameters of the virtual connections, we use a simulation created using 5G-LENA [Pat+19], in particular a demo simulating a Dual-Polarized Multiple Input Multiple Output (DP-MIMO) [BAL22]. The simulation takes as primary input the distance between the cell tower and the UE, i.e., the vUAV, but also cell tower frequency, MIMO antenna count, etc.

We found that 5G-LENA simulations are computationally expensive, taking several seconds each and fully consuming one CPU thread, so that the update frequency further decreases as vUAV count increases above the total CPU thread count. To counteract this behavior, we also offer a mode where the network parameters are computed in advance based on distance, and then the closest result is chosen when running the testbed. While this

approach decreases the accuracy, it reduces the computational load per vUAV by orders of magnitude.

To implement throttling based on calculated network parameters, we use tc-netem [Hem11]. This tool is widely adopted due to its direct integration with the Linux operating system, as well as the availability of wrappers designed to simplify its usage. However, the final version of our testbed does not incorporate any wrappers. We are aware of a note of caution on using tc-netem by Becker et al. [Bec+22]. They found that it can slow down packets by several milliseconds when enabling several thousand filters. However, we made the decision to still use it, because for now, we intend to operate only several hundred vUAVs or less, with only one link each. Becker et al. also acknowledge, that tc-netem remains a "popular choice for network emulation in edge and IoT testbeds" [Bec+22]. To throttle each vUAV individually, we create traffic filters based on IP address.

The architecture and communication of our implementation of vUAVs is shown in Figure 5.6. We verify our testbed using the edge-based anti-collision system by 6G NeXt, though other edge-based systems should be compatible, such as AI image processing systems. Therefore, we implement the vUAVs with the same Kafka-based protocol their system assumes, to ensure compatibility. This protocol is used, e.g., to allow the UAV to inform the edge function of its position, which can consequently command the UAV to turn to a new heading, in case a collision risk is detected.

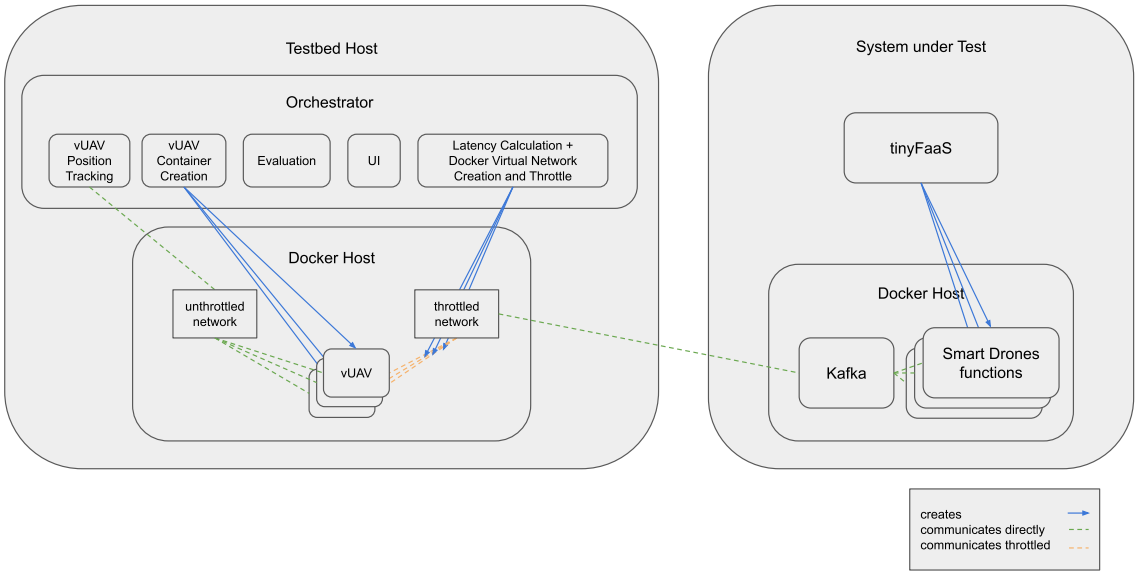

Figure 5.5: The Software Testbed Implementation Architecture

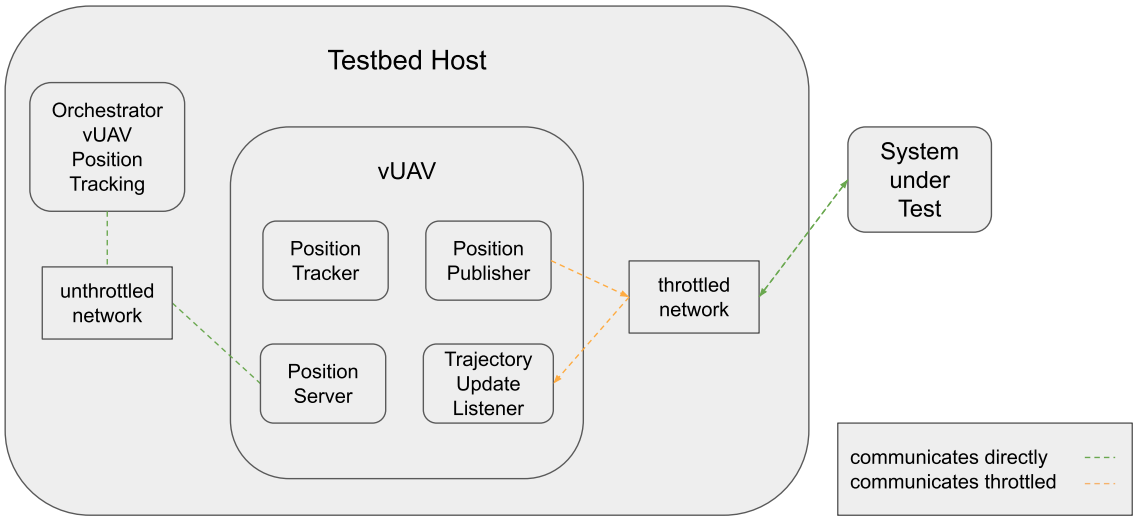

Figure 5.6: Architecture of a single vUAV

5.1.1 User Interface

Following the advice of Pfandzelter and Bermbach [PB24], the user interface is controlled by a Command Line Interface (CLI) and configuration files for specifying scenarios. We also followed the advice to create a browser-based map for visualizing the current positions of the vUAVs and the cellular base stations.

The configuration files the specify starting location and waypoints of the vUAVs, as well as the locations and certain characteristics of the cell towers. Additionally, they may specify whether to use real-time calculated, precalculated, or no network throttling parameters. The files are written as Python code, rather than structured data formats, such as YAML or JSON. This improves flexibility is and adds support for features, such as different probability distributions of vUAV positions, without explicitly implementing them.

Given the nature of text files, they allow for repeatable experiments with an easy way to make only certain changes and experience the corresponding effects. A potential scenario might be one where the user has identified an area, where cell coverage is not strong enough for reliable UAV operation, and experiment with adding another virtual cell tower in a closer location, or alternatively whether it might be enough to adjust the transmission frequency of the existing cell network, which could reduce costs when rolling out the system in the real world.

The map visualization supports interactive zooming and panning to focus on specific sub-regions of the experiment area, e.g., to visually distinguish between collisions and near-collisions. vUAVs are visualized as fixed-wing aircraft pointing in their direction of travel. Base stations are visualized as circles, as seen in Figure 5.8.

The map visualization is generated using *Plotly* [Plo14], a Python graphing library that supports rendering scatter plots overlaid on tile maps, which it can source from OpenStreetMap [Ope04]. To make these visualizations accessible in a web browser, we use *Plotly Dash* [Plo15]. Dash extends Plotly's functionality by allowing the deployment of interactive applications with graphs, in a web browser. It incorporates live updates, so that movement of vUAVs can be monitored in real time.

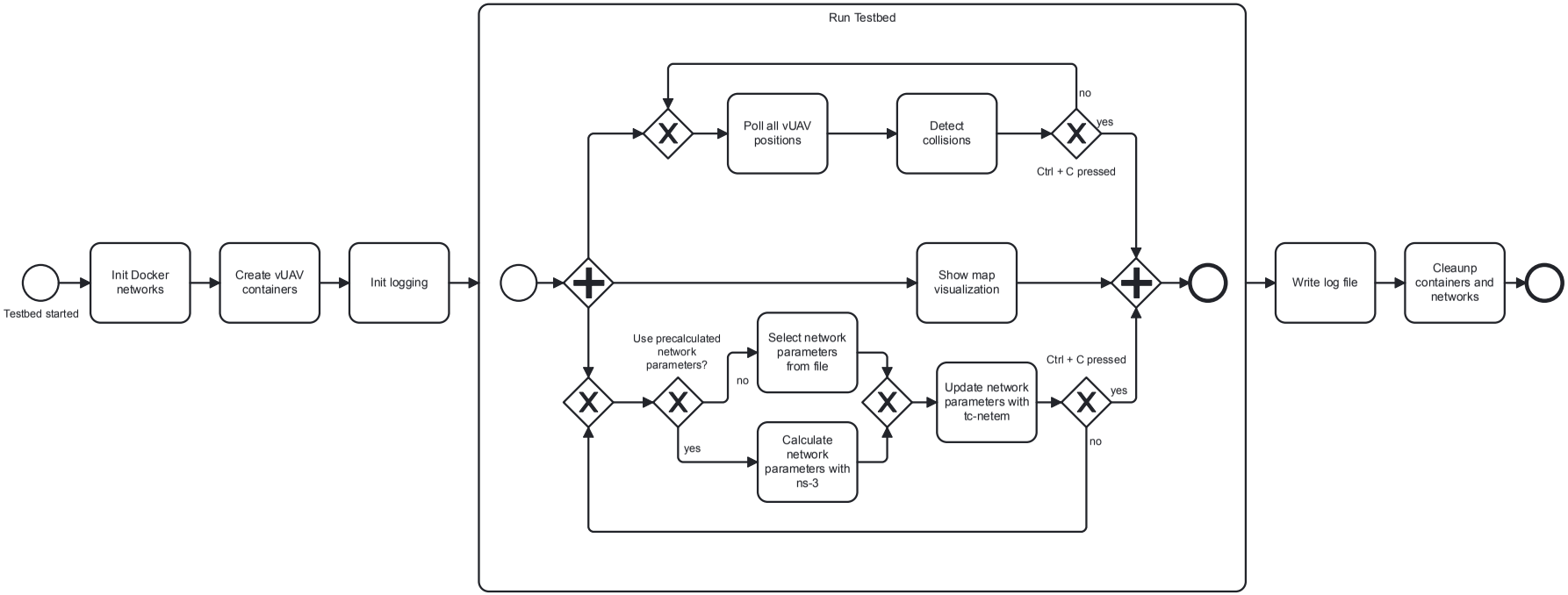

Figure 5.7: Testbed Processes modelled in BPMN [CT12]

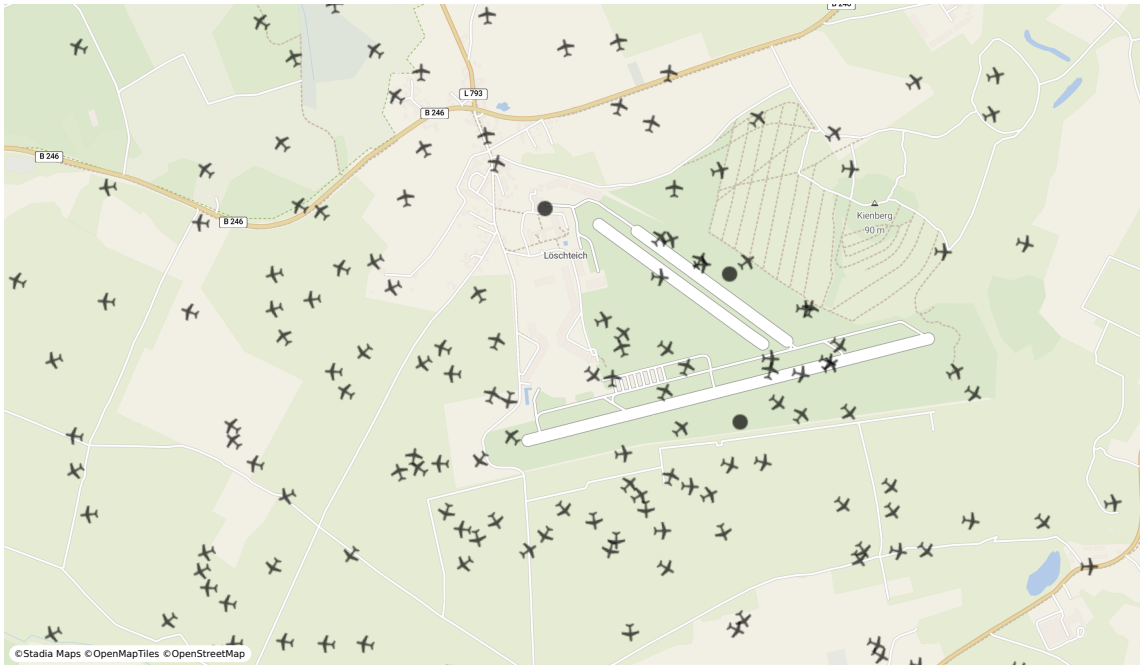

Figure 5.8: The Testbed Map Visualization while executing a Scenario with 200 UAVs

5.1.2 Collision Detection

To achieve efficient collision detection, we use the ball tree data structure, implemented in the scikit-learn library [sci07]. A ball tree is similar to a k-d tree, but with design optimizations tailored for data in spherical or Euclidean spaces. This allows us to find nearest neighbors, and their distances, on large spheres, such as the Earth. If the nearest neighbor is closer than the set distance threshold, we assume that a collision has occurred.

Compared to the trivial implementation of measuring the distance of each UAV from every other UAV, the computational complexity is reduced from $O(n^2)$, to only $O(n(log(n))^2)$ for the construction of the ball tree plus $O(n \cdot log(n))$ for the lookups [Omo89].

5.2 Experiment Setup

We evaluate our testbed implementation on efficacy, scalability, and repeatability.

5.2.1 Efficacy

Track and Throttle To assess our testbed's ability to track vUAVs, while at the same time throttling their network connection, we set up a simple scenario with a single vUAV at a simulated environment resembling Schönhagen Airport [Flu13]. The vUAV flies from northwest to southeast, above the runway, along which three cell towers are located, as seen in Figure 5.9. We monitor the throughput of the connection using iperf3 [ESn07], and the latency using ping.

Collision Detection To evaluate SkyBed's collision detection capabilities, we set up another scenario in which two vUAVs are placed on a direct head-on collision course to determine if the testbed can reliably detect collisions as their speeds increase. Because collision detection in SkyBed relies on periodic polling, we anticipate that the system might

fail to detect a collision at very high speeds, unless we increase the detection radius. For this experiment, we set the radius to 10 meters, and the default polling rate is 0.1 seconds. To account for the variability introduced by the polling intervals, we define a successful detection as registering the collision in two consecutive polling cycles, ensuring that the detection is not merely the result of favorable timing. Based on the parameters given, we can calculate the maximum speed for accurate collision detection, if our testbed works as intended:

$$v_{max} = \frac{\frac{10m}{0.1s}}{2} = 50\frac{m}{s} = 180\frac{km}{h}$$

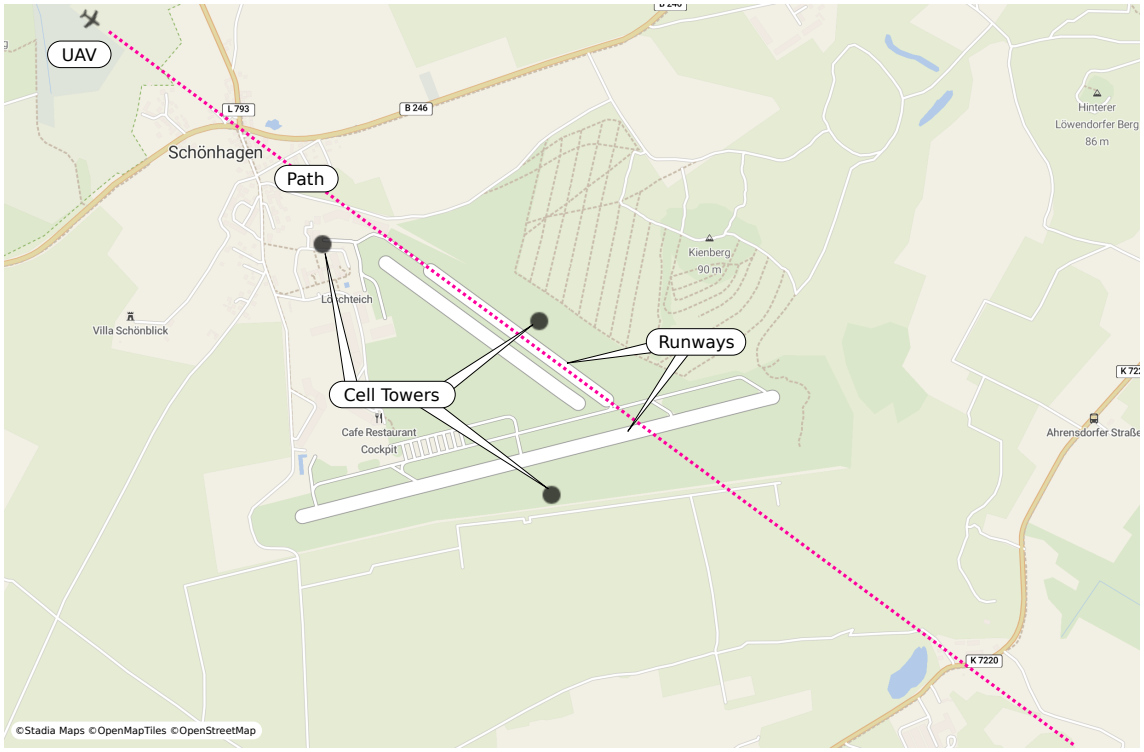

Figure 5.9: The path of the vUAV

5.2.2 Scalability

To determine up to how many concurrent vUAVs the testbed can operate, we first stress-test it with an increasing number of vUAVs. As the system relies on knowing the positions of the vUAVs, we observe whether the position requests succeed. If the request times out, the orchestrator cannot update the vUAV's position, and correspondingly, cannot determine collisions nor update the visualization. This experiment is performed on a GCP e2-standard-32 cloud VM with 128 GB of memory, because otherwise memory would be a limiting factor, as we found that each vUAV occupies close to 100 MB of RAM.

Next, we measure the impact of precalculated vs. real-time simulated network parameters on the average time per update. Since this operation is asynchronous, a slow update time does not directly impact the orchestrator's ability to monitor for collisions and update the visualization. However, it can serve as a factor in deciding which option to use for a specific case, since a very long update time can mean that the vUAV has already flown

a significant distance, possibly far enough for the network parameters to differ enough, as to cause inaccurate results. Experiment 3 and 4 of Table 5.2 explain the respective experiment setups.

Experiment	Exp. 1	Exp. 2	Exp. 3	Exp. 4	Exp. 5
Time per run	10 Minutes				
Machine	AMD Ryzen 5700X3D PC			GCP e2-standard-32	5700X3D PC; e2-standard-32; i5-8250U PC
vCPUs	16			32	4-32
RAM [GB]	32			128	8-128
UAVs	1	2	4, 8, ..., 120	20, 40, ..., 600	20, repeated 5x
Throttling	prec.	prec.	prec., then sim.	disabled	sim.

Table 5.2: Experiment Setups

5.2.3 Repeatability

To evaluate the testbed's repeatability, we perform another experiment on heterogeneous hardware to ensure that it does not significantly affect the results. The setup is explained in Experiment 5 of Table 5.2. In this experiment, the final positions of the vUAVs are stored. With identical starting positions and paths, the final positions should ideally be the same.

5.3 Experiment Results

5.3.1 Efficacy

Track and Throttle The results for throughput and latency throughout the experiment are shown in Figure 5.10 and Figure 5.11. Throughput peaks around 150 and 200 seconds into the experiment, corresponding to the vUAV passing the first two cell towers. A third, smaller spike occurs around 250 seconds, likely due to the increased distance between the flight path and the third cell tower. As expected, latency exhibits an inverse relationship to throughput, being lowest when throughput is at its highest. These results align with our prediction that proximity to cell towers would result in faster connections and lower latency.

However, the throughput demonstrates an unexpected "angular" pattern, rising and falling in what appear to be discrete steps. This effect is particularly noticeable in the later stages of the experiment, after 400 seconds, where throughput oscillates between 0 Mbit/s and approximately 10 Mbit/s. Interestingly, this phenomenon is absent in the latency measurements. The precise cause of this behavior remains unclear, but may be attributed to the TCP windowing mechanism used by iperf for its measurements or to the filter bucket implementation in tc-netem, but further investigation is needed.

We performed the experiments using both the real-time 5G-LENA simulation and the precalculated 5G-LENA simulation. The results for throughput and latency exhibit minimal deviation between the two approaches, suggesting that using precalculated simulation values is a viable option with negligible impact on testbed accuracy.

It is worth noting that the observed latency values are higher than anticipated. Under optimal conditions, with a direct line of sight and a distance of approximately 100 meters between the vUAV and the cell tower, latency values were expected to remain well below 60 ms, closer to the 10 ms range reported by Kiesel et al. [Kie+22].

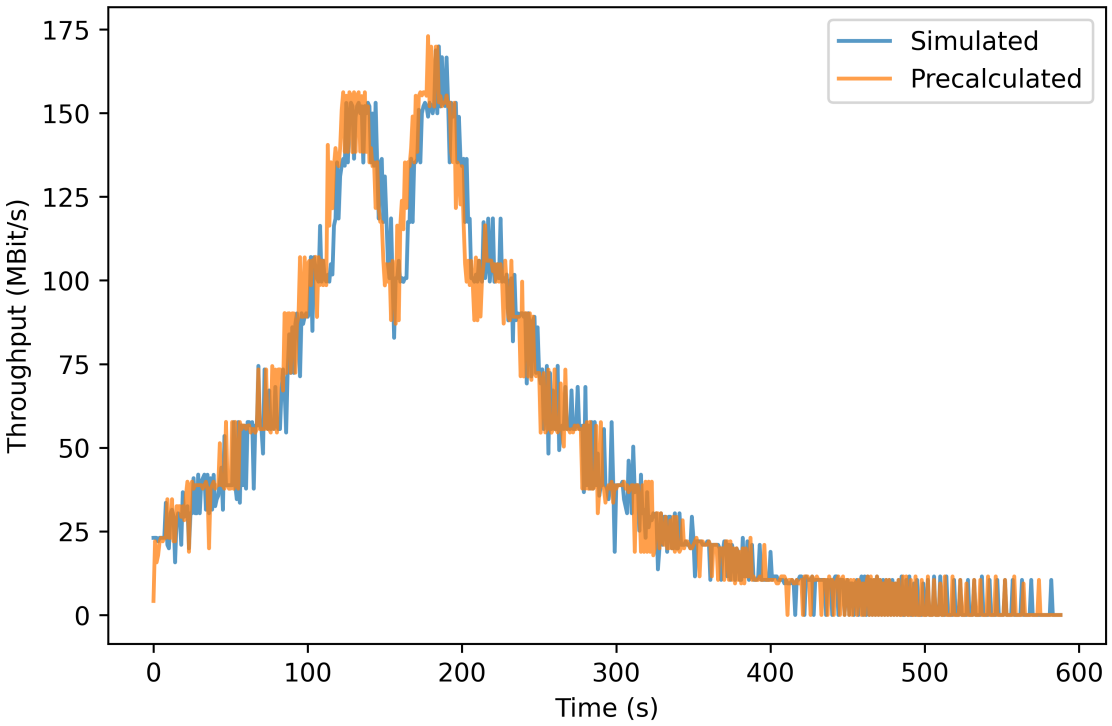

Figure 5.10: vUAV Throughput over Time

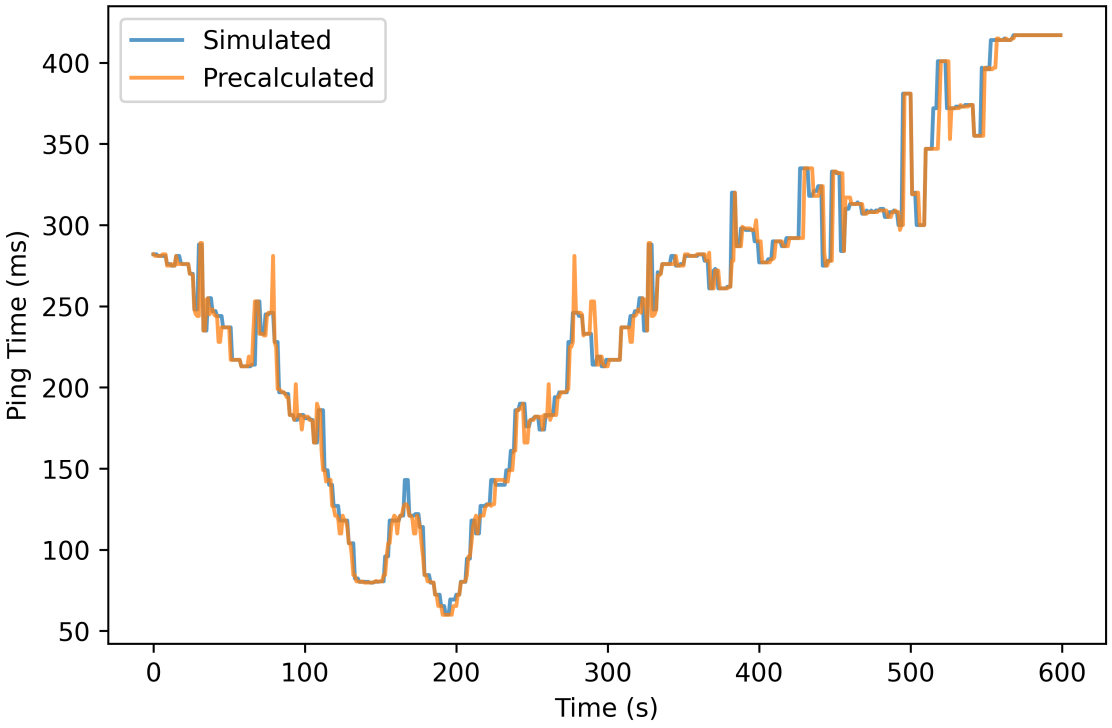

Figure 5.11: vUAV Latency over Time

Collision Detection Table 5.3 shows the results of the collision detection experiment. At collision radius of 10 m, the detection stops deterministically detecting all collisions above 150 km/h vUAV velocity. However, if even faster speeds are required, the collision radius can simply be increased.

Speed [km/h]	Collision detected
100	Yes
150	Yes
200	No

Table 5.3: Collision Detection Experiment Results

5.3.2 Scalability

Figure 5.12 and Figure 5.13 show that the number of successful requests equals the number of total requests until approximately 150 UAVs, when the first requests start to time out. At approximately 250 UAVs, the total number of successful requests peaks, but drops near zero as the count reaches 600 UAVs. The number of requests per UAV decreases slowly from 900 (because the experiment was running for 15 minutes at 1-second update frequency) to roughly 600, indicating that the testbed itself was also slowed down to some extent.

However, as illustrated in Figure 5.14, the distribution of request success rates is highly uneven. For instance, with 320 vUAVs, over 20% of vUAVs experience a success rate that is either close to or exactly zero. This occurs despite Figure 5.13 indicating that successful

requests still outnumber failed requests at this vUAV count. This becomes particularly evident when observing the map visualization, where some vUAVs move as expected, while others remain stationary.

This issue cannot be attributed to memory constraints or high CPU utilization, as Figure 5.15 demonstrates that neither resource approaches 100% utilization, even at 600 UAVs. However, the figure does reveal a direct linear correlation between the number of UAVs and resource consumption. This indicates the presence of another bottleneck, unrelated to memory or CPU usage, that is causing the low request success rate. Further investigation is needed to identify the root cause of this issue.

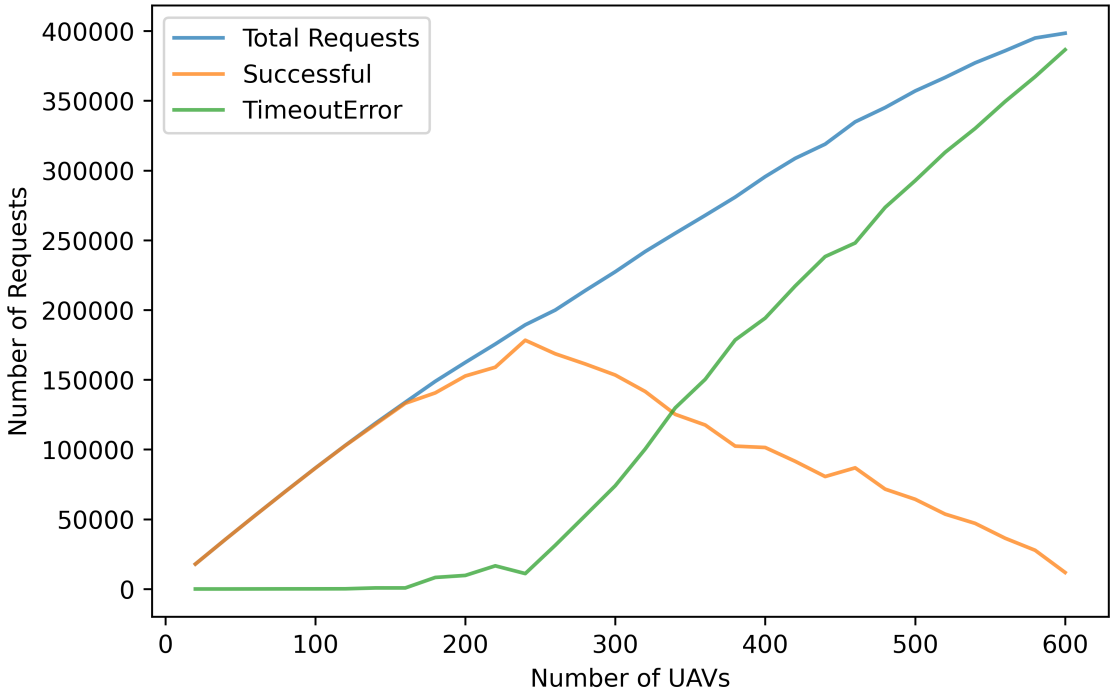

Figure 5.12: Number of UAVs plotted against the Success Rate of the HTTP Position Requests

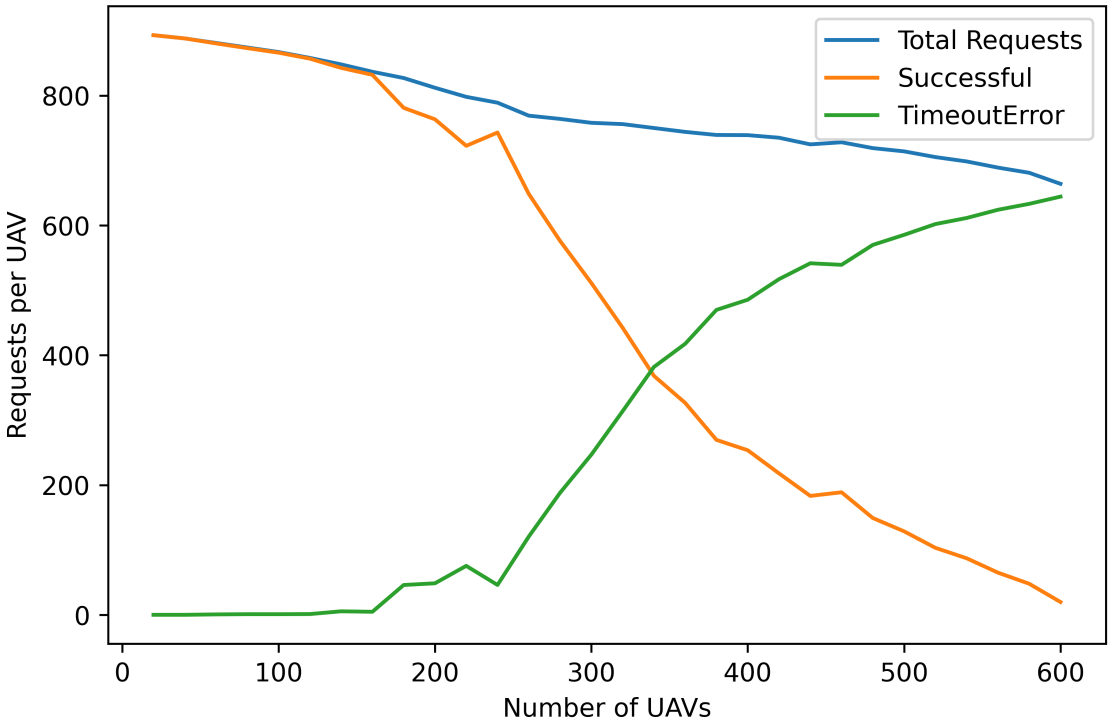

Figure 5.13: Number of UAVs plotted against the Success Rate of the Position Requests per UAV

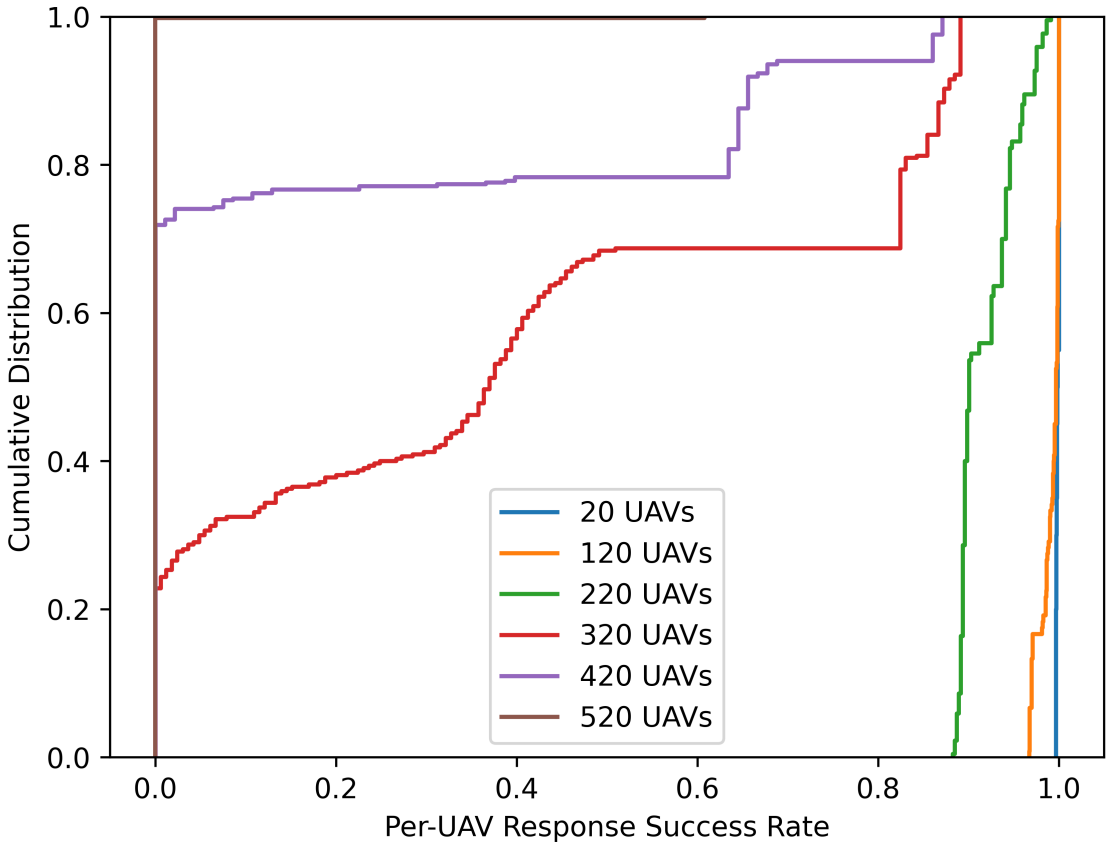

Figure 5.14: Cumulative Distribution Function of the Per-UAV Rate of Successful Responses

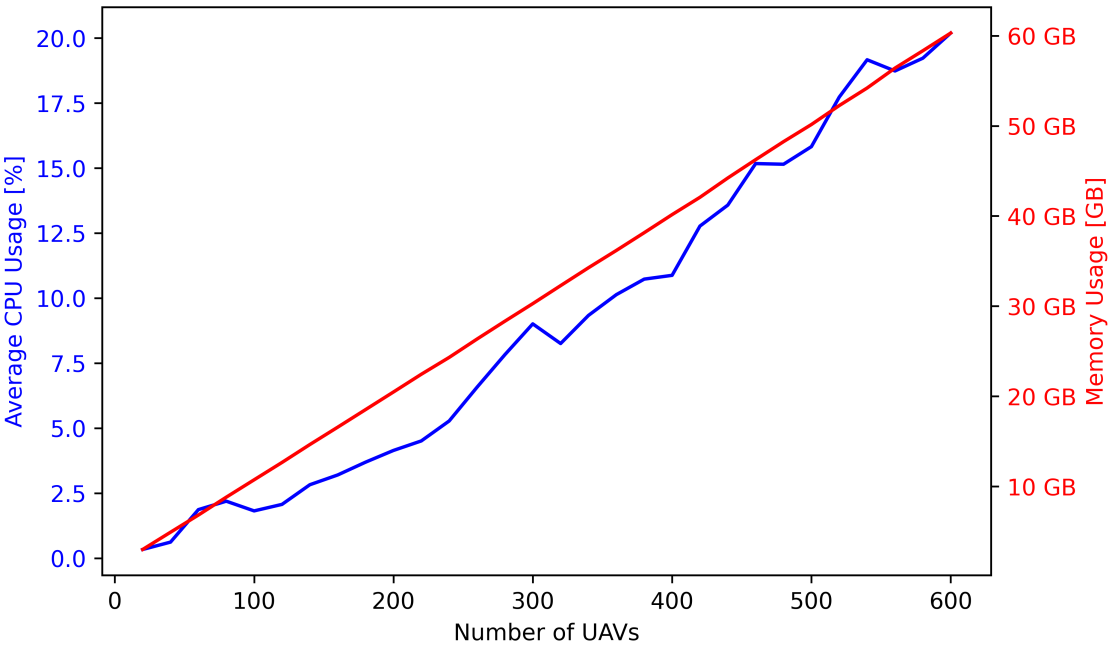

Figure 5.15: Number of UAVs plotted against Resource Consumption

Figure 5.16 shows that the average time per network update grows approximately linearly with the number of vUAVs for the real-time simulation, up to almost one minute at 120 vUAVs. The time also grows when using precalculated values, but so much slower, that it cannot be seen in the figure. The lowest time when using precalculated values at 4 vUAVs is approximately 0.5 seconds, and the highest time at 120 vUAVs is approximately one second.

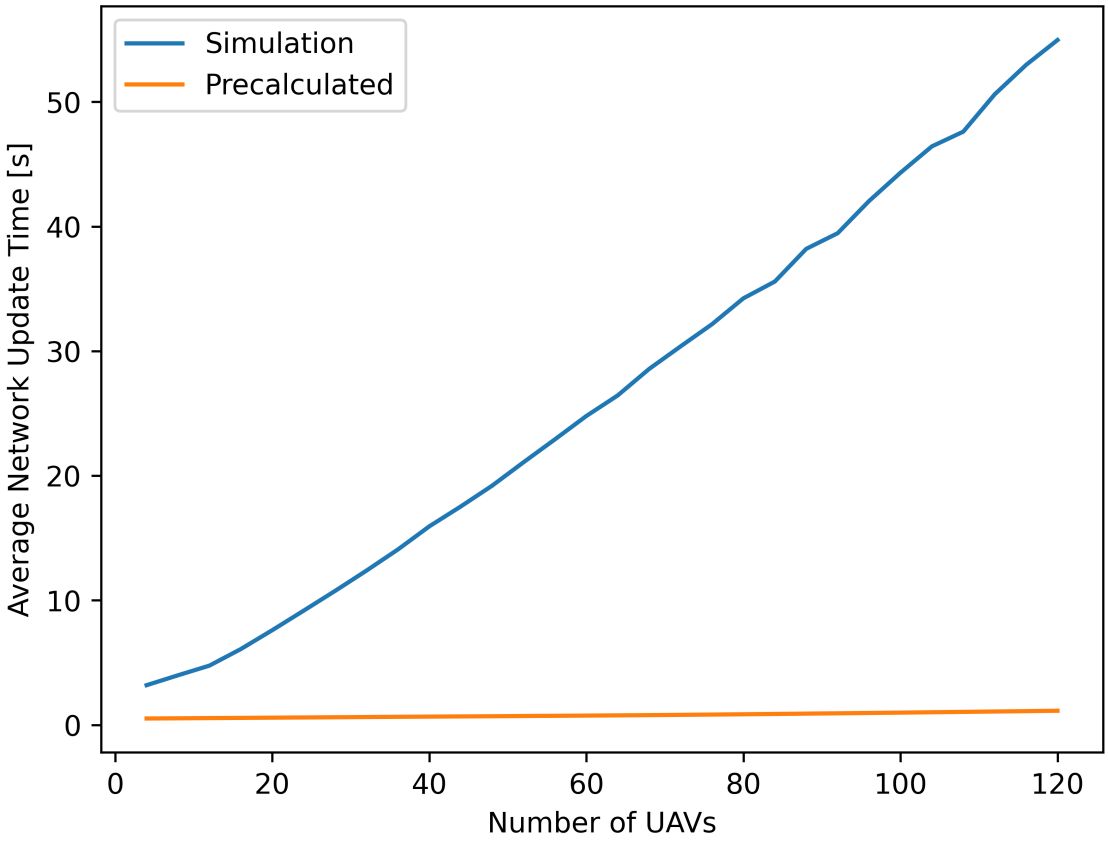

Figure 5.16: Number of UAVs plotted against the Average Network Update Time

5.3.3 Repeatability

Table 5.4 shows our findings regarding repeatability. All vUAVs travel about 12 kilometers. Repeated runs on a single machine result in minimal deviations from the average, only 0.255 %. The difference when including the results from all three machines is much larger at 2.2 %.

Metric	Only the GCP VM	All Systems
Average distance traveled	12276.1 m	12090.9 m
Average standard deviation at destination	31.3 m	262.5 m
Relative to the total distance traveled	0.00255	0.022

Table 5.4: Results of the Repeatability Evaluation

6 Discussion

In this section, we will discuss the limitations of our work and how future work might overcome them, as well as attempt a prediction of what the future of UAVs might look like.

6.1 Efficacy

Our experiments indicate that precalculated values minimally affect accuracy, making them the preferred choice in most scenarios due to their vastly superior scalability. The primary exception arises when conducting experiments while varying antenna configurations, such as testing different frequencies. In these cases, the time required for precalculation may exceed the duration of the experiments, rendering real-time simulation more practical. Future work might consider adding a hybrid approach, using precalculated values when available, and real-time simulated values otherwise.

To enhance collision detection, a dynamic collision threshold could be introduced, adjusted based on the relative speed of the UAVs. This approach would help ensure that high-velocity UAVs are less likely to evade detection. It is also consistent with the principle that higher relative speeds require greater separation to mitigate risk, further improving the safety and reliability of the system.

6.2 Scalability

When only considering the context of 6G NeXt, our testbed implementation can handle more UAVs than will ever fly concurrently during the project. However, when considering a borader context, even 1000 UAVs, which is enough to overload our implementation, is a small number. Doole et al. [DEH20] expect 24500 package delivery UAVs per hour in the Paris metropolitan area alone by the year 2035. Future work could close this gap in a number of ways, which are ideally combined to maximize scalability:

Our vUAV implementation is currently written in Python and uses FastAPI for its HTTP server. This has allowed us to prototype quickly and reuse parts of the code base already built for the testbed. However, it comes at the cost of a large memory footprint and considerable CPU usage. Rewriting the vUAVs in a lower level programming language would likely be 1 to 2 orders of magnitude less resource intensive. Reduced resource consumption would also reduce the variability we found during the repeatability analysis.

Another approach towards solving the scalability problem would be to implement horizontal scaling, i.e., distributing the vUAVs across different hardware. Because they already communicate using IP-based protocols, this would likely require relatively few modifications. For instance, Celestial creates an overlay network using WireGuard [Don17], a type of Virtual Private Network (VPN), which allows them to connect hosts and provide routing across satellites. This approach could be replicated in our system.

If these approaches are implemented, the number of links between vUAVs and virtual cell towers could grow to the point where tc-netem becomes a performance bottleneck. A potential solution, as demonstrated in previous work [PB22][AMS24], is to leverage eBPFs, a technology that is rapidly gaining popularity. SkyBed could adopt this approach to effectively overcome scalability challenges.

However, before taking any of these steps, future work might also investigate the findings from Figure 5.14 and Figure 5.15, because they indicate, that there might exist another bottleneck yet to be identified.

6.3 Realism

We spend considerable effort trying to evaluate, whether our testbed behaves the way we expect it to. However, to truly assess realism, we need to know, how UAVs behave in the real world. Future work involves visiting Schönhagen Airport to compare the testbed's behavior with that of real UAVs. An intermediate step could also be building a hardware testbed with real UAV compute hardware.

We set our initial scope in Figure 1.1 to include only a subset of systems, in part because simulators already exist for some other subsystems for UAVs. Several simulators exist for UAV flight dynamics, flight controls, and motors. In particular, ArduPilot [Mun+09] and its Software in the Loop (SITL) module [Ard09] could be integrated into our testbed implementation to provide these capabilities.

Furthermore, the way we currently perform our network parameter simulations comes with the drawback that some inaccuracy may occur, as mentioned in Section 5.3.1. SkyBed does not currently account for antenna angles, or interference between the individual UAVs. It also does not consider that cell tower antennas are typically angled downward to enhance coverage for ground users [Ger+22], reducing coverage in the sky. Future work could expand on the way 5G-LENA performs its simulations, or try an entirely different simulation environment, such as Simu5G [Nar+20b].

Moreover, simulating 5G features such as network slicing and URLLC, which are designed to reduce latency, could significantly benefit real-world UAVs. Therefore, incorporating these features into a testbed simulation is a logical step.

Finally, we made the simplifying assumption, that the latency between the cell tower and the edge computing hardware is zero. We made this assumption because the radio transmission is likely to have a much larger impact on latency. Nonetheless, when aiming for maximized realism, simulating this part of the route is needed. Therefore, future work could include containerizing the edge application platform, to spawn multiple instances. This would be further beneficial by allowing the emulation of handovers between edge functions.

In contrast, while causing somewhat reduced repeatability during long-running experiments, the deviations observed during the repeatability evaluation are unlikely to significantly impact realism. Real-world conditions often exhibit similar or larger deviations, due to variations in wind and atmospheric factors [Wil16].

6.4 Strengths

While the previously discussed limitations are specific to our current implementation, they are not inherent flaws in the architecture itself. In fact, we believe that our architecture holds significant potential due to its simplicity and adaptability. By implementing the future work outlined, we anticipate that the limitations mentioned can be eliminated.

Moreover, as Janßen et al. [Jan+23] observe, "there appears to be an oversight when it comes to physical safety. Specifically, strategies to prevent UAV collisions with obstacles, other UAVs, or crucially, ensuring safe behavior during connection losses, are not comprehensively addressed." Our work can play an important role in filling this particular gap. While it does not directly prevent collisions or handle connection losses, it gives other researchers, such as those working on 6G NeXt, the ability to build such systems.

6.5 Outlook

The future of UAVs is exceedingly promising. Many applications are already profitable and deployed at scale, such as cost-effective and fast last-mile package delivery [Bet24], more efficient pesticide and fertilizer spraying [Rad+20], and reliable power transmission line inspections [Li+23]. As UAV technology advances, their integration into various industries will likely expand further, driven by increasing autonomy and reduced operating costs.

Whether UAVs will fully adopt cellular connectivity and leverage edge computing to address scalability and latency challenges remains an open question. The answer to this will shape how UAVs evolve in the coming years and their role in leveraging technologies like 6G to unlock even greater potential.

7 Conclusion

In this work, we have demonstrated the benefits of having a dedicated testbed for edge UAV systems, enabling researchers, developers, and engineers to test and evaluate their UAV edge computing software in a realistic yet controlled environment. We examined concepts related to UAV edge computing, as well as existing testbeds, what we could learn from them, and what gaps they left unaddressed. With SkyBed, we have introduced a software testbed architecture to fulfill this role. Through a proof-of-concept implementation, we have showcased how SkyBed can emulate UAV interactions in edge environments, enabling rigorous testing of collision detection systems and other edge-based applications.

In support of these claims, we evaluated our prototype on the critical metrics of efficacy, scalability, and repeatability, highlighting its ability to simulate UAV behavior under different scenarios, such as varying network conditions, or with hundreds of UAVs simultaneously.

SkyBed also opens exciting avenues for future research. Researchers can now build and evaluate edge platforms addressing challenges, such as fault tolerance in UAV systems. To further enhance the realism of SkyBed, integrating real-world UAV telemetry, along with improved scalability, and network and flight dynamics modeling, would be key. By making SkyBed available as an open-source tool, we aim to empower researchers to keep pushing the boundaries of UAV edge computing, fostering innovation and driving the adoption of autonomous aerial technologies.

References

[Aga+20] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. "Firecracker: Lightweight Virtualization for Serverless Applications". In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). 2020, pp. 419–434. ISBN: 9781939133137.

[Ahm+22] Faiyaz Ahmed, J. C. Mohanta, Anupam Keshari, and Pankaj Singh Yadav. "Recent Advances in Unmanned Aerial Vehicles: A Review". In: *Arabian Journal for Science and Engineering* 47.7 (July 1, 2022), pp. 7963–7984. ISSN: 2191-4281. DOI: 10.1007/s13369-022-06738-0.

[Ama06] Amazon. *Amazon EC2*. 2006. URL: https://aws.amazon.com/de/ec2/.

[AMS24] Sebastião Amaro, Miguel Matos, and Valerio Schiavoni. "Kollaps: Decentralized and Efficient Network Emulation for Large-Scale Systems". In: *IEEE/ACM Transactions on Networking* (2024), pp. 1–16. ISSN: 1063-6692, 1558-2566. DOI: 10.1109/TNET.2024.3478050.

[Apa12] Apache Software Foundation. *Apache Kafka*. 2012. URL: https://kafka.apache.org/.

[Ard09] ArduPilot Dev Team. *SITL Simulator (Software in the Loop)*. 2009. URL: https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html.

[BAL22] Biljana Bojovic, Zoraze Ali, and Sandra Lagen. "ns-3 and 5G-LENA Extensions to Support Dual-Polarized MIMO". In: *Proceedings of the 2022 Workshop on ns-3*. WNS3 '22. New York, NY, USA: Association for Computing Machinery, June 22, 2022, pp. 1–9. ISBN: 9781450396516. DOI: 10.1145/3532577.3532595.

[Bec+22] Soeren Becker, Tobias Pfandzelter, Nils Japke, David Bermbach, and Odej Kao. "Network Emulation in Large-Scale Virtual Edge Testbeds: A Note of Caution and the Way Forward". In: *2022 IEEE International Conference on Cloud Engineering (IC2E)*. 2022 IEEE International Conference on Cloud Engineering (IC2E). Sept. 2022, pp. 1–7. DOI: 10.1109/IC2E55432.2022.00007.

[Bec24] Jan Beckschewe. *SkyBed*. 2024. URL: https://github.com/jan-be/skybed.

[Bet24] Francesco Betti Sorbelli. "UAV-Based Delivery Systems: A Systematic Review, Current Trends, and Research Challenges". In: *ACM Journal on Autonomous Transportation Systems* 1.3 (Sept. 30, 2024), pp. 1–40. ISSN: 2833-0528. DOI: 10.1145/3649224.

[BSL18] Sabur Baidya, Zoheb Shaikh, and Marco Levorato. "FlyNetSim: An Open Source Synchronized UAV Network Simulator based on ns-3 and Ardupilot". In: *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. MSWIM '18: 21st ACM Int'l Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems. Montreal QC Canada: ACM, Oct. 25, 2018, pp. 37–45. ISBN: 9781450359603. DOI: 10.1145/3242102.3242118.

[CT12] Michele Chinosi and Alberto Trombetta. "BPMN: An introduction to the standard". In: *Computer Standards & Interfaces* 34.1 (Jan. 2012), pp. 124–134. ISSN: 09205489. DOI: 10.1016/j.csi.2011.06.002.

[DEH20] Malik Doole, Joost Ellerbroek, and Jacco Hoekstra. "Estimation of traffic density from drone-based delivery in very low level urban airspace". In: *Journal of Air Transport Management* 88 (Sept. 2020), p. 101862. ISSN: 09696997. DOI: 10.1016/j.jairtraman.2020.101862.

[DLN23] Gerasimos Damigos, Tore Lindgren, and George Nikolakopoulos. "Toward 5G Edge Computing for Enabling Autonomous Aerial Vehicles". In: *IEEE Access* 11 (2023), pp. 3926–3941. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3235067.

[Doc13] Docker, Inc. *Docker*. 2013. URL: https://www.docker.com/.

[Don17] Jason A. Donenfeld. "WireGuard: Next Generation Kernel Network Tunnel". In: *Proceedings 2017 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA: Internet Society, 2017. ISBN: 9781891562464. DOI: 10.14722/ndss.2017.23160.

[ESn07] ESnet / Lawrence Berkeley National Laboratory. *iperf3*. 2007. URL: https://github.com/esnet/iperf.

[ETS24] ETSI. *Multi-access Edge Computing (MEC); Framework and Reference Architecture*. Apr. 2024.

[Fer+20] Damián Fernández-Cerero, Alejandro Fernández-Montes, F. Javier Ortega, Agnieszka Jakóbik, and Adrian Widlak. "Sphere: Simulator of edge infrastructures for the optimization of performance and resources energy consumption". In: *Simulation Modelling Practice and Theory* 101 (May 2020), p. 101966. ISSN: 1569190X. DOI: 10.1016/j.simpat.2019.101966.

[Fia+19] Claudio Fiandrino, Alejandro Blanco Pizarro, Pablo Jiménez Mateo, Carlos Andrés Ramiro, Norbert Ludant, and Joerg Widmer. "openLEON: An end-to-end emulation platform from the edge data center to the mobile user". In: *Computer Communications* 148 (Dec. 15, 2019), pp. 17–26. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2019.08.024.

[Flu13] Flugplatzgesellschaft Schönhagen. *Flugplatz Schönhagen*. 2013. URL: https://www.flugplatz-schoenhagen.aero/.

[Ger+22] Giovanni Geraci, Adrian Garcia-Rodriguez, M. Mahdi Azari, Angel Lozano, Marco Mezzavilla, Symeon Chatzinotas, Yun Chen, Sundeep Rangan, and Marco Di Renzo. "What Will the Future of UAV Cellular Communications Be? A Flight From 5G to 6G". In: *IEEE Communications Surveys & Tutorials* 24.3 (2022), pp. 1304–1335. ISSN: 1553-877X. DOI: 10.1109/COMST.2022.3171135.

[Gom+23] Jose Gomez, Elie F. Kfoury, Jorge Crichigno, and Gautam Srivastava. "A survey on network simulators, emulators, and testbeds used for research and education". In: *Computer Networks* 237 (Dec. 2023), p. 110054. ISSN: 13891286. DOI: 10.1016/j.comnet.2023.110054.

[Gom23] Alejandro Gomez. *How to Use Docker and NS-3 to Create Realistic Network Simulations*. 2023. DOI: 10.1184/R1/22348669.V1.

[Gou+20] Paulo Gouveia, João Neves, Carlos Segarra, Luca Liechti, Shady Issa, Valerio Schiavoni, and Miguel Matos. "Kollaps: decentralized and dynamic topology emulation". In: *Proceedings of the Fifteenth European Conference on Computer Systems*. EuroSys '20: Fifteenth EuroSys Conference 2020. Heraklion Greece: ACM, Apr. 15, 2020, pp. 1–16. ISBN: 9781450368827. DOI: 10.1145/3342195.3387540.

[Gup+17] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments". In: *Software: Practice and Experience* 47.9 (Sept. 2017), pp. 1275–1296. ISSN: 0038-0644, 1097-024X. DOI: 10.1002/spe.2509.

[Hag+23] Amiraslan Haghrah, Mehran Pourmohammad Abdollahi, Hosein Azarhava, and Javad Musevi Niya. "A survey on the handover management in 5G-NR cellular networks: aspects, approaches and challenges". In: *EURASIP Journal*

on Wireless Communications and Networking 2023.1 (June 22, 2023), p. 52. ISSN: 1687-1499. DOI: 10.1186/s13638-023-02261-4.

[Han19] Mark Handley. "Using ground relays for low-latency wide-area routing in megaconstellations". In: *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*. HotNets '19: The 18th ACM Workshop on Hot Topics in Networks. Princeton NJ USA: ACM, Nov. 14, 2019, pp. 125–132. ISBN: 9781450370202. DOI: 10.1145/3365609.3365859.

[Has+19] Jonathan Hasenburg, Martin Grambow, Elias Grünewald, Sascha Huk, and David Bermbach. "MockFog: Emulating Fog Computing Infrastructure in the Cloud". In: *2019 IEEE International Conference on Fog Computing (ICFC)*. 2019 IEEE International Conference on Fog Computing (ICFC). June 2019, pp. 144–152. DOI: 10.1109/ICFC.2019.00026.

[Hem11] Stephen Hemminger. *tc-netem(8) — Linux manual page*. Nov. 25, 2011. URL: https://man7.org/linux/man-pages/man8/tc-netem.8.html Last accessed: 10/13/2024.

[Heo15] Tejun Heo. *cgroups(7) — Linux manual page*. Oct. 2015. URL: https://man7.org/linux/man-pages/man7/cgroups.7.html.

[HGB23] Jonathan Hasenburg, Martin Grambow, and David Bermbach. "MockFog 2.0: Automated Execution of Fog Application Experiments in the Cloud". In: *IEEE Transactions on Cloud Computing* 11.1 (Jan. 1, 2023), pp. 58–70. ISSN: 2168-7161, 2372-0018. DOI: 10.1109/TCC.2021.3074988. arXiv: 2009.10579[cs].

[Jan+23] Malte Janßen, Tobias Pfandzelter, Minghe Wang, and David Bermbach. *Supporting UAVs with Edge Computing: A Review of Opportunities and Challenges*. Oct. 18, 2023. arXiv: 2310.11957[cs,eess].

[Kha+22] Benish Sharfeen Khan, Sobia Jangsher, Ashfaq Ahmed, and Arafat Al-Dweik. "URLLC and eMBB in 5G Industrial IoT: A Survey". In: *IEEE Open Journal of the Communications Society* 3 (2022), pp. 1134–1163. ISSN: 2644-125X. DOI: 10.1109/OJCOMS.2022.3189013.

[Kie+22] Raphael Kiesel, Sarah Schmitt, Niels König, Maximilian Brochhaus, Thomas Vollmer, Kirstin Stichling, Alexander Mann, and Robert H. Schmitt. "Techno-Economic Evaluation of 5G-NSA-NPN for Networked Control Systems". In: *Electronics* 11.11 (Jan. 2022), p. 1736. ISSN: 2079-9292. DOI: 10.3390/electronics11111736.

[LHM10] Bob Lantz, Brandon Heller, and Nick McKeown. "A network in a laptop: rapid prototyping for software-defined networks". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-X: 10th ACM Workshop on Hot Topics in Networks. Monterey California: ACM, Oct. 20, 2010, pp. 1–6. ISBN: 9781450304092. DOI: 10.1145/1868447.1868466.

[Li+23] Ziran Li, Yanwen Zhang, Hao Wu, Satoshi Suzuki, Akio Namiki, and Wei Wang. "Design and Application of a UAV Autonomous Inspection System for High-Voltage Power Transmission Lines". In: *Remote Sensing* 15.3 (Feb. 3, 2023), p. 865. ISSN: 2072-4292. DOI: 10.3390/rs15030865.

[Mao+17] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B. Letaief. "A Survey on Mobile Edge Computing: The Communication Perspective". In: *IEEE Communications Surveys & Tutorials* 19.4 (2017), pp. 2322–2358. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2745201.

[May+17] Ruben Mayer, Leon Graser, Harshit Gupta, Enrique Saurez, and Umakishore Ramachandran. "EmuFog: Extensible and scalable emulation of large-scale

fog computing infrastructures". In: *2017 IEEE Fog World Congress (FWC)*. 2017 IEEE Fog World Congress (FWC). Oct. 2017, pp. 1–6. DOI: 10.1109/FWC.2017.8368525.

[Mel+23a] Sergiy Melnyk, Qiuheng Zhou, Hans D. Schotten, Mandy Galkow-Schneider, Ingo Friese, Tobias Pfandzelter, David Bermbach, Louay Bassbouss, Alexander Zoubarev, Andy Neparidze, Arndt Kritzner, Enrico Zschau, Prasenjit Dhara, Steve Goering, William Menz, Alexander Raake, Wolfgang Ruether-Kindel, Fabian Quaeck, Nick Stuckert, and Robert Vilter. "6G NeXt — Towards 6G Split Computing Network Applications: Use Cases and Architecture". In: *Mobile Communication - Technologies and Applications; 27th ITG-Symposium*. Mobile Communication - Technologies and Applications; 27th ITG-Symposium. May 2023, pp. 126–131.

[Mel+23b] Sergiy Melnyk, Qiuheng Zhou, Hans D. Schotten, Wolfgang Rüther-Kindel, Fabian Quaeck, Nick Stuckert, Robert Vilter, Lisa Gebauer, Mandy Galkow-Schneider, Ingo Friese, Steffen Drüsedow, Tobias Pfandzelter, Mohammadreza Malekabbasi, David Bermbach, Louay Bassbouss, Alexander Zoubarev, Andy Neparidze, Arndt Kritzner, Jakob Hartbrich, Alexander Raake, Enrico Zschau, and Klaus-Jürgen Schwahn. "6G NeXt - joint communication and computer mobile network : use cases and architecture". In: (2023). DOI: 10.25673/111638.

[Mun+09] Jordi Munoz, Andrew Tridgell, Randy Mackay, and Chris Anderson. *ArduPilot*. 2009. URL: https://ardupilot.org/.

[MWL22] Patrick McEnroe, Shen Wang, and Madhusanka Liyanage. "A Survey on the Convergence of Edge Computing and AI for UAVs: Opportunities and Challenges". In: *IEEE Internet of Things Journal* 9.17 (Sept. 2022), pp. 15435–15459. ISSN: 2327-4662. DOI: 10.1109/JIOT.2022.3176400.

[Nar+20a] Giovanni Nardini, Dario Sabella, Giovanni Stea, Purvi Thakkar, and Antonio Virdis. "Simu5G–An OMNeT++ Library for End-to-End Performance Evaluation of 5G Networks". In: *IEEE Access* 8 (2020), pp. 181176–181191. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3028550.

[Nar+20b] Giovanni Nardini, Giovanni Stea, Antonio Virdis, and Dario Sabella. "Simu5G: A System-level Simulator for 5G Networks:" in: *Proceedings of the 10th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. 10th International Conference on Simulation and Modeling Methodologies, Technologies and Applications. Lieusaint - Paris, France: SCITEPRESS - Science and Technology Publications, 2020, pp. 68–80. ISBN: 9789897584442. DOI: 10.5220/0009826400680080.

[Nar+20c] Giovanni Nardini, Giovanni Stea, Antonio Virdis, Dario Sabella, and Purvi Thakkar. "Using Simu5G as a Realtime Network Emulator to Test MEC Apps in an End-To-End 5G Testbed". In: *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*. 2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications. ISSN: 2166-9589. Aug. 2020, pp. 1–7. DOI: 10.1109/PIMRC48278.2020.9217177.

[NSV21] Giovanni Nardini, Giovanni Stea, and Antonio Virdis. "Scalable Real-Time Emulation of 5G Networks With Simu5G". In: *IEEE Access* 9 (2021), pp. 148504–148520. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3123873.

[Omo89] Stephen M Omohundro. "Five balltree construction algorithms". In: *ICSI Technical Report TR-89-063* (1989). Publisher: Citeseer.

[Ope04] OpenStreetMap Foundation. *OpenStreetMap*. 2004. URL: https://www.openstreetmap.org/.

[Ope10] OpenInfra Foundation. *OpenStack*. 2010. URL: https://www.openstack.org/.

[Pat+19] Natale Patriciello, Sandra Lagen, Biljana Bojovic, and Lorenza Giupponi. "An E2E simulator for 5G NR networks". In: *Simulation Modelling Practice and Theory* 96 (Nov. 1, 2019), p. 101933. ISSN: 1569-190X. DOI: 10.1016/j.simpat.2019.101933.

[PB22] Tobias Pfandzelter and David Bermbach. "Celestial: Virtual Software System Testbeds for the LEO Edge". In: *Proceedings of the 23rd ACM/IFIP International Middleware Conference*. Nov. 7, 2022, pp. 69–81. DOI: 10.1145/3528535.3531517. arXiv: 2204.06282[cs].

[PB24] Tobias Pfandzelter and David Bermbach. *Lessons Learned from Building Edge Software System Testbeds*. 2024. DOI: 10.48550/ARXIV.2403.16869.

[Pfa+24] Tobias Pfandzelter, David Bermbach, Robert Vilter, Ingo Friese, Sergiy Melnyk, Qiuheng Zhou, and Hans D. Schotten. "Towards Anti-Collision Coordination for UAVs with Serverless Edge Computing". In: *Proceedings of the 12th IEEE International Conference on Cloud Engineering*. IC2E '24. event-place: Paphos, Cyprus. New York, NY, USA: IEEE, Sept. 2024, MISSING.

[PKV16] Manuel Peuster, Holger Karl, and Steven Van Rossem. "MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments". In: *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2016 IEEE Conference on Network Function Virtualization and Software-Defined Networks (NFV-SDN). Palo Alto, CA: IEEE, Nov. 2016, pp. 148–153. ISBN: 9781509009336. DOI: 10.1109/NFV-SDN.2016.7919490.

[Plo14] Plotly. *Plotly Open Source Graphing Library for Python*. 2014. URL: https://plotly.com/python/.

[Plo15] Plotly. *Dash Python User Guide*. 2015. URL: https://dash.plotly.com/.

[Por+18] Pawani Porambage, Jude Okwuibe, Madhusanka Liyanage, Mika Ylianttila, and Tarik Taleb. "Survey on Multi-Access Edge Computing for Internet of Things Realization". In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 2961–2991. ISSN: 1553-877X. DOI: 10.1109/COMST.2018.2849509.

[Rad+20] Panagiotis Radoglou-Grammatikis, Panagiotis Sarigiannidis, Thomas Lagkas, and Ioannis Moscholios. "A compilation of UAV applications for precision agriculture". In: *Computer Networks* 172 (May 2020), p. 107148. ISSN: 13891286. DOI: 10.1016/j.comnet.2020.107148.

[Rai+23] Philipp Raith, Thomas Rausch, Alireza Furutanpey, and Schahram Dustdar. "*faas-sim* : A trace-driven simulation framework for serverless edge computing platforms". In: *Software: Practice and Experience* 53.12 (Dec. 2023), pp. 2327–2361. ISSN: 0038-0644, 1097-024X. DOI: 10.1002/spe.3277.

[Ram18] Sebastián Ramírez. *FastAPI*. 2018. URL: https://fastapi.tiangolo.com/.

[RH10] George F. Riley and Thomas R. Henderson. "The ns-3 Network Simulator". In: *Modeling and Tools for Network Simulation*. Ed. by Klaus Wehrle, Mesut Güneş, and James Gross. Berlin, Heidelberg: Springer, 2010, pp. 15–34. ISBN: 9783642123313. DOI: 10.1007/978-3-642-12331-3_2.

[Riz97] Luigi Rizzo. "Dummynet: a simple approach to the evaluation of network protocols". In: *ACM SIGCOMM Computer Communication Review* 27.1 (Jan. 1997), pp. 31–41. ISSN: 0146-4833. DOI: 10.1145/251007.251012.

[Sch+18] Matt Schmittle, Anna Lukina, Lukas Vacek, Jnaneshwar Das, Christopher P. Buskirk, Stephen Rees, Janos Sztipanovits, Radu Grosu, and Vijay Kumar. "OpenUAV: A UAV Testbed for the CPS and Robotics Community". In: *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS). Apr. 2018, pp. 130–139. DOI: 10.1109/ICCPS.2018.00021.

[sci07] scikit-learn. *BallTree*. 2007. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html.

[Sha+19] Hazim Shakhatreh, Ahmad H. Sawalmeh, Ala Al-Fuqaha, Zuochao Dou, Eyad Almaita, Issa Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. "Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges". In: *IEEE Access* 7 (2019), pp. 48572–48634. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2909530.

[Shi+16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. "Edge Computing: Vision and Challenges". In: *IEEE Internet of Things Journal* 3.5 (Oct. 2016), pp. 637–646. ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2579198.

[SR05] J.E. Smith and Ravi Nair. "The architecture of virtual machines". In: *Computer* 38.5 (May 2005), pp. 32–38. ISSN: 0018-9162. DOI: 10.1109/MC.2005.173.

[Var10] Andras Varga. "OMNeT++". In: *Modeling and Tools for Network Simulation*. Ed. by Klaus Wehrle, Mesut Güneş, and James Gross. Berlin, Heidelberg: Springer, 2010, pp. 35–59. ISBN: 9783642123313. DOI: 10.1007/978-3-642-12331-3_3.

[Wet+14] Philip Wette, Martin Dräxler, Arne Schwabe, Felix Wallaschek, Mohammad Hassan Zahraee, and Holger Karl. "MaxiNet: Distributed emulation of software-defined networks". In: *2014 IFIP Networking Conference*. 2014 IFIP Networking Conference. June 2014, pp. 1–9. DOI: 10.1109/IFIPNetworking.2014.6857078.

[Wil16] Paul D Williams. "Transatlantic flight times and climate change". In: *Environmental Research Letters* 11.2 (Feb. 1, 2016), p. 024008. ISSN: 1748-9326. DOI: 10.1088/1748-9326/11/2/024008.

[WSW05] G. Werner-Allen, P. Swieskowski, and M. Welsh. "MoteLab: a wireless sensor network testbed". In: *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.* IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005. Apr. 2005, pp. 483–488. DOI: 10.1109/IPSN.2005.1440979.

[Yan+19] Zhaohui Yang, Cunhua Pan, Kezhi Wang, and Mohammad Shikh-Bahaei. "Energy Efficient Resource Allocation in UAV-Enabled Mobile Edge Computing Networks". In: *IEEE Transactions on Wireless Communications* 18.9 (Sept. 2019), pp. 4576–4589. ISSN: 1536-1276, 1558-2248. DOI: 10.1109/TWC.2019.2927313.

[Yaz+21] Yassine Yazid, Imad Ez-Zazi, Antonio Guerrero-González, Ahmed El Oualkadi, and Mounir Arioua. "UAV-Enabled Mobile Edge-Computing for IoT Based on AI: A Comprehensive Review". In: *Drones* 5.4 (Dec. 13, 2021), p. 148. ISSN: 2504-446X. DOI: 10.3390/drones5040148.

[You+19] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. "All one needs to know about fog computing and related edge computing paradigms: A complete survey". In: *Journal of Systems Architecture* 98 (Sept. 2019), pp. 289–330. ISSN: 13837621. DOI: 10.1016/j.sysarc.2019.02.009.

[ZCS19] Yukun Zeng, Mengyuan Chao, and Radu Stoleru. “EmuEdge: A Hybrid Emulator for Reproducible and Realistic Edge Computing Experiments”. In: *2019 IEEE International Conference on Fog Computing (ICFC)*. 2019 IEEE International Conference on Fog Computing (ICFC). June 2019, pp. 153–164. DOI: 10.1109/ICFC.2019.00027.

[Zho+18] Fuhui Zhou, Yongpeng Wu, Haijian Sun, and Zheng Chu. “UAV-Enabled Mobile Edge Computing: Offloading Optimization and Trajectory Design”. In: *2018 IEEE International Conference on Communications (ICC)*. 2018 IEEE International Conference on Communications (ICC 2018). Kansas City, MO: IEEE, May 2018, pp. 1–6. ISBN: 9781538631805. DOI: 10.1109/ICC.2018.8422277.