Ainur: A Framework for Repeatable End-to-End Wireless Edge Computing Testbed Research

Manuel Olguín Muñoz[*], Seyed Samie Mostafavi[†], Vishnu N. Moothedath[‡], James Gross[§]
School of Electrical Engineering & Computer Science
KTH Royal Institute of Technology, Sweden
Email: {[*]molguin, [†]ssmos, [‡]vnmo, [§]jamesgr}@kth.se

***Abstract*—Experimental research on wireless networking in combination with edge and cloud computing has been the subject of explosive interest in the last decade. This development has been driven by the increasing complexity of modern wireless technologies and the extensive softwarization of these through projects such as a Open Radio Access Network (O-RAN). In this context, a number of small- to mid-scale testbeds have emerged, employing a variety of technologies to target a wide array of use-cases and scenarios in the context of novel mobile communication technologies such as 5G and beyond-5G. Little work, however, has yet been devoted to developing a standard framework for wireless testbed automation which is hardware-agnostic and compatible with edge- and cloud-native technologies. Such a solution would simplify the development of new testbeds by completely or partially removing the requirement for custom management and orchestration software.**

In this paper, we present the first such mostly hardware-agnostic wireless testbed automation framework, *Ainur*. It is designed to configure, manage, orchestrate, and deploy workloads from an end-to-end perspective. Ainur is built on top of cloud-native technologies such as Docker, and is provided as FOSS to the community through the KTH-EXPECA/Ainur repository on GitHub. We demonstrate the utility of the platform with a series of scenarios, showcasing in particular its flexibility with respect to physical link definition, computation placement, and automation of arbitrarily complex experimental scenarios.

I. Introduction

Experimental research in the area of wireless networking has received ever increasing attention over the last years, driven, on the one hand, by the complexity of modern networked systems and corresponding applications. On the other, networked systems are more and more based on software instead of dedicated hardware, which allows experimental testbeds to be rededicated simply through an update as system versions evolve — in contrast to the redeployment of hardware necessitated 10–15 years ago. The complexity of these systems, as well as their softwarization are expected to continue growing, driving in turn an expanding interest in testbed-based experimental research in wireless systems.

Over the last years, several small- to mid-scale testbeds have emerged that leverage a large degree of freedom with respect to hardware and software, for example, the 1) COSMOS, 2) POWDER, 3) ARA, and 4) Drexel Grid Software-Defined Radio (SDR) testbeds. COSMOS (*Cloud enhanced Open Software defined MObile wireless testbed for city-Scale deployment*) is a testbed spanning an area of roughly 1 square mile ($2.6\,\mathrm{km}^2$) featuring SDRs, mm-wave equipment, optical fibers, cloud integration, and compute for core network functionality and application data processing [1, 2]. It contains over 200 rooftop, intermediate, and mobile nodes, and is controlled and managed by a central node. COSMOS relies on the ORBIT Management Framework (OMF) (originally developed for ORBIT [3]), and employs the OMF Experiment Description Language (OEDL), a domain-specific imperative language based on Ruby, for experiment development and definition.

POWDER (*Platform for Open Wireless Data-driven Experimental Research*) promises research on wireless and mobile networks with a level of programmability down to the waveform [4]. The testbed spans a $15\,\mathrm{km}^2$ area and features about 15 fixed programmable radio nodes, based on off-the-shelves SDRs and featuring edge-like compute capabilities and integration with cloud resources, which interact with 50 mobile nodes. POWDER experiments are defined and developed in *profiles*, which correspond to Virtual Machine (VM) images containing the necessary software and configurations. These profiles are defined through using Resource Specification (RSpec)[1] documents.

The Agricultural and ruRAl communities (ARA) platform is an at-scale testbed for wireless research spanning a rural area with a diameter of over $60\,\mathrm{km}$ in Iowa [5]. Its core goal is the study and deployment of advanced wireless platforms and technologies in real-world agricultural and rural settings. It includes a broad range of wireless platforms ranging from low-Ultra-High Frequency (UHF) massive Multi-Input Multi-Output (MIMO) to mmWave, deployed through both SDRs and programmable Commercial Off-The-Shelves (COTS) radios, as well as automated ground vehicles, cameras and sensors. ARA's software stack, ARASoft, is based upon the highly flexible and powerful CHameleon Infrastructure (CHI) software suite from the Chameleon testbed project [6], which in turn is based on the widely adopted OpenStack [7] cloud-computing framework. This affords the ARA testbed a high degree of flexibility, as well as lowers the learning curve for new contributors and users.

Finally, the *Drexel Grid SDR Testbed* features SDRs that connect either over-the-air, through a channel emulator, or over a combination of the two, to facilitate realistic and reproducible experimentation [8]. Primarily intended for SDR-

[1]https://groups.geni.net/geni/wiki/GENIExperimenter/RSpecs

centric research, it does not integrate any core, cloud or edge components. However, the testbed extensively employs the LinuX Containers (LXC) runtime for the deployment of both experimental code and SDR software, which affords users great freedom when it comes to the development of experiments.

Experimentation is key to to fully understanding the implications of next-generation wireless systems, cloud, and edge computing paradigms, and thus more of these testbeds are sure to emerge in the near future. Yet, little work has so far been devoted to general-purpose, hardware-agnostic software frameworks for the management and automation of such systems. Existing platforms implement their own, ad-hoc software solutions which are not compatible with other testbeds, and in many cases are not even compatible with reigning cloud-native standards. This is, for instance, the case with COSMOS and POWDER; their reliance on domain-specific languages limits their integration with cloud-native solutions, which generally build upon general-purpose languages such as Python and Go. These testbeds further leverage virtualization technology based on VMs instead of more lightweight and edge-compatible solutions such as containers.

To the best of our knowledge, CloudRAFT [9] is the only work to tackle (to a certain extent) this challenge. CloudRAFT corresponds to a cloud-based framework for mobile network experimentation, with a focus on simplifying the management of testbed resources. The goal of this project is to integrate, coordinate, share, and improve upon existing testbeds; and employs pre-built VMs containing the necessary software for experiments. Although it provides some automation for testbed resource provisioning and experiment execution, its focus is largely rather on the sharing and partitioning of testbed systems. Testbeds currently working with CloudRAFT include a variety of domain-specific setups, including an SDR-based testbed as a well as a ground vehicular robot for mobility-related experimentation.

In this work, we present our solution to the challenge of testbed automation: Ainur, a framework for wireless testbed automation with a specific focus on end-to-end experimental research in the context of edge-computing using cloud- and edge-native technologies. Ainur is designed to deploy experimental runs from a workload perspective by configuring the physical testbed, initializing all involved software components, deploying and executing the experimental workload, collecting logs and data, and finally gracefully degrading the system. The framework allows for dynamic, software-definition of physical and logical links, network topology, cloud and edge computing resources, as well as experimental workload deployment and orchestration. It heavily leverages cloud-native technologies, such as Docker containers, in order to support a wide variety of different testbed hardware setups and experimental configurations and workloads, as well as to be as easily extendable as possible. Furthermore, we make Ainur available to the community as Free and Open Source Software (FOSS). It can be obtained from the KTH-EXPECA/Ainur repository on GitHub [10], released under an Apache version 2.0 license.

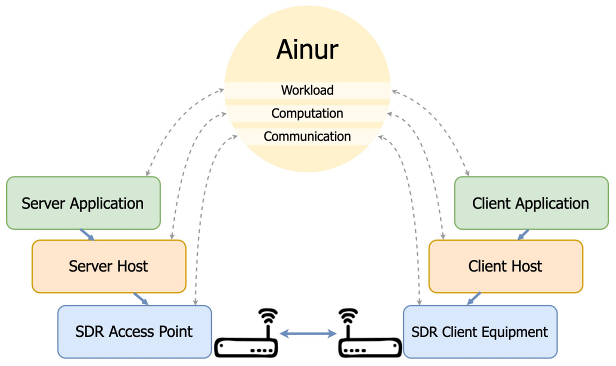


Fig. 1: Layered structure of an Ainur experiment

The rest of this paper is structured as follows. Section II presents the framework as well as the key concepts and technologies supporting its design and architecture. This section also discusses briefly the assumptions made about the underlying hardware on which the framework is set to run, and we present an overview of our experimental testbed. Next, Section III describes two demonstration procedures through which we will showcase the flexibility and potential of this tool for the automatic, repeatable, end-to-end experimentation in the context of wireless testbeds. Finally, in Section IV we summarize our contributions, discuss future directions, and conclude this paper.

II. The Ainur Framework

Ainur is designed as an end-to-end wireless network testbed management framework, fully flexible in terms of the communication network stack, computation hosts, and the distributed application deployed on top. The core goal of the framework is to facilitate the creation of the desired communication and computation elements to discover their implications to the performance of end-to-end distributed applications. Ainur achieves this by offering a Python Application Programming Interface (API) which is used to describe an end-to-end experiment in a procedural manner, and we plan to eventually provide toolkits for declarative configuration of experiments.

Conceptually, in Ainur, an experiment is decomposed into a layered structure, consisting of 1) the distributed application (workload); 2) computation hosts; and 3) communication networks connecting the hosts. In Figure 1, the workload is represented by a client-server process pair. These could correspond, for instance, to the emulation of an inverted pendulum and a matching controller. However, Ainur makes no assumptions about the nature of the workloads deployed on the framework, and virtually any process or combination of processes can be used.

Hosts correspond to either bare-metal machines or cloud instances on Amazon Web Services (AWS) Elastic Compute 2 (EC2). Users are free to combine and interconnect these in any configuration. Ainur can provision wired networks over ethernet, and supports a number of software-defined wireless communication stacks, currently including WiFi, 4G

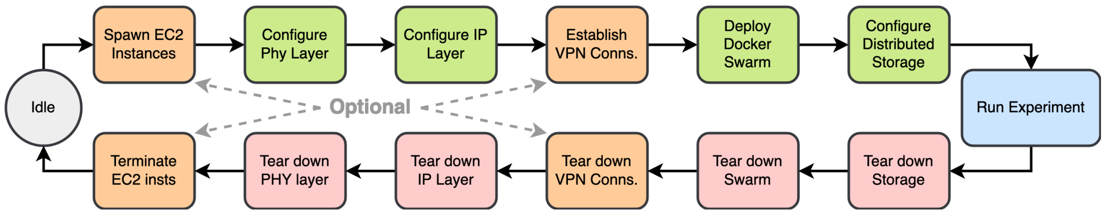


Fig. 2: Lifecycle of an experimental run in Ainur. Blocks in orange are optional.

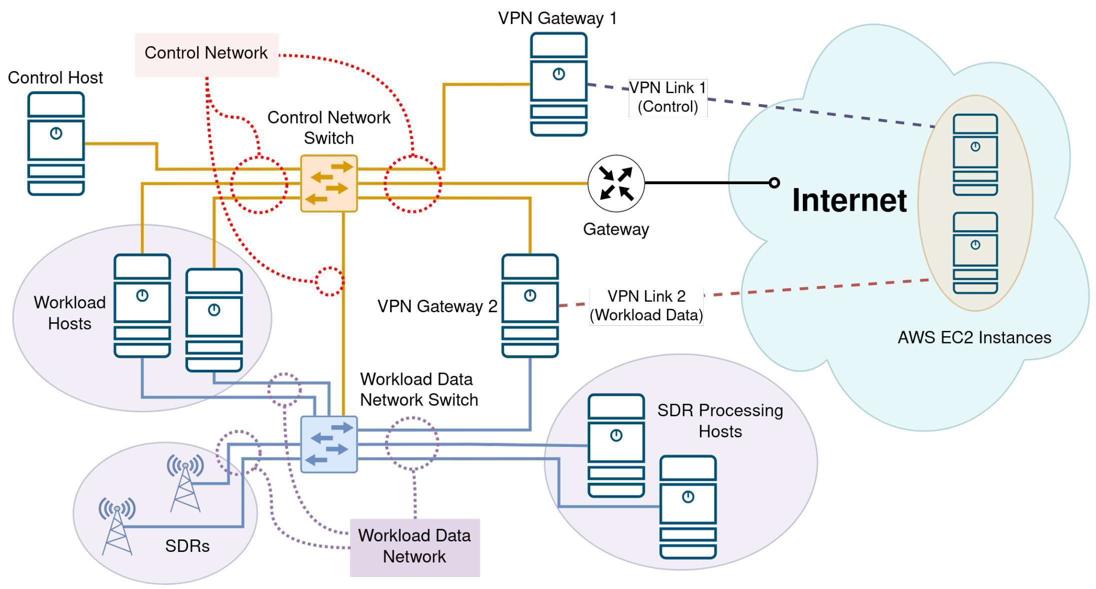


Fig. 3: Network structure assumed by Ainur.

Long-Term Evolution (LTE), and 5G. To realize these various wireless communication protocols, it is assumed that the testbed is equipped with Universal Software Radio Peripheral (USRP) SDRs and some computation hosts dedicated to signal processing and/or radio management.

To collect data from workloads, Ainur configures both a shared, distributed, storage location which all processes can reach, as well as a logging service which automatically captures structured and unstructured data from the standard output of workload processes. The logging service additionally collects data from the other two layers, and the resulting dataset could, for instance, be analyzed to relate the performance of the control loop to the quality of the wireless link.

A. Ainur Software Stack

To realize our vision for an automated, flexible, cloud-native, and workload-agnostic framework for cloud and edge computing experimentation, we built Ainur on top of a combination of well-established tools and frameworks. Below we briefly touch on the most important of these:

Python: Ainur is built on-top of Python 3.8. We chose this language for its flexibility, ease of prototyping, and for the extensive ecosystem of third-party cloud-native frameworks and libraries.

Ansible: One of the “cloud-native frameworks” mentioned above, Red Hat Ansible is a powerful Python framework for bare-metal configuration, provisioning, and automation. We use it extensively in Ainur for configuration of network interfaces and services on managed hosts.

Containers: We leverage Docker containers to great extent for the virtualization and orchestration of workloads, as well as the encapsulation and management of complex network configurations. See Section II-C for more details.

AWS and `boto3`: Ainur employs the Amazon `boto3` library for Python to directly interface with AWS EC2 and deploy, configure, and manage remote cloud instances.

OpenAirInterface (OAI): OAI is an open-source project that implements 3GPP technology on general purpose `x86` computing hardware and off-the-shelf SDRs like the USRP [11]. Ainur can deploy, configure, and manage OAI software components that implement 4G LTE and 5G New Radio (NR).

Mango Communications 802.11: Finally, the Mango Communications project implements real-time 802.11 (WiFi) MAC and PHY in Xilinx Field-Programmable Gate Arrays (FPGAs). It can be used on a variety of hardware platforms including USRP SDRs, and is employed in Ainur to provision end-to-end WiFi links.

B. Main Software Components

Ainur follows a layered architecture which closely mimics the conceptual layers of the TCP/IP stack. Components are deployed in bottom-up order, starting with the establishment of physical links, through the establishment of Internet Protocol (IP) connectivity and deployment of links to the cloud, and ending with the distribution and initialization of workloads on top of a container orchestration layer. The architectural modules of the framework can broadly be classified according to the below categories:

Configuration Layer: The lowest layer of Ainur. It handles the parsing of configuration files describing experimental scenarios. Optional, as it is only needed when Ainur is running experiments described in a declarative manner.

Logging layer: This layer handles logging across all layers of the system to a central repository. Concretely, it manages the configuration and lifecycle of a Fluentd server to which any component in the network can send logs. Currently, the two main components relying on this scaffolding are the physical layer and the workloads.

Physical Layer: Creates and deploys the underlying physical connections of the workload data network. This layer interacts with hardware such as managed switches and SDRs (and their associated computation hosts) to create links between the desired workload hosts. These links correspond to ethernet, WiFi, and even an entire software-defined 4G LTE and 5G networks.

Cloud Layer: Handles integration with cloud services (currently, AWS EC2). Only deployed in the case of an experiment requiring cloud instances, it manages the instantiation and configuration of remote cloud resources.

IP Layer: Configures and establishes IP layer connectivity of packets between hosts in the experimental setup, both local and cloud. This includes assigning valid IP addresses to hosts, establishing Virtual Private Network (VPN) routes to cloud instances through a pair of pre-configured VPN gateways, and configuring routing tables to ensure any two hosts in the workload network can communicate with each other.

Workload Layer: Finally, this layer deploys, scales, and orchestrates containerized workloads, as well as configures a shared, distributed storage for workloads to store data. This components leverages Docker Swarm to spin up and manage workload containers on desired hosts. It also establishes overlay networks abstracting away the physical topology and allowing containers to interconnect through dynamically assigned hostnames.

C. Containerization

Containers are a key cloud-native technology for the virtualization and sandboxing of arbitrary processes [12]. They allow for easy packaging of software in predefined, consistent, and conflict-free execution environments, including all necessary dependencies. Containers are lightweight and portable compared to VM-based virtualization, making them an ideal solution for the distribution, deployment, and orchestration of software in distributed computing environments. They deploy quickly and are very configurable, while abstracting away the complexities and delays that come with in creating, managing, and moving (potentially huge) VM disk images.

Ainur leverages containerization extensively across the framework, in order to 1) deploy and orchestrate workloads, 2) support a wide spectrum of different physical layer configurations out-of-the-box, as well as allow for easy extension to new ones, and 3) support automated collection of logs. Workloads in the framework are deployed packaged in Docker containers, orchestrated through `docker-compose` and Docker Swarm. We have selected Docker Swarm as our container orchestration layer instead of more advanced toolkits such as Kubernetes due to 1) it being more lightweight; 2) its comparative simplicity in terms of configuration and setup; and 3) the fact that it is included in the default Docker runtime installation. This allows the framework to remain mostly agnostic to the nature of the workloads, and thus support a wide ranged of different applications and system architectures. It also allows for easy deployment to different compute nodes in the network, without having to take into consideration details such as required libraries and versions.

As expressed above, Ainur also leverages containers for the communication stack. With the advent of Open Radio Access Network (O-RAN) and SDRs, all components of the wireless network stack can run on general-purpose processors. They can thus be containerized and distributed across multiple hosts.

As a final point, the automation of logging is another advantage of using containers and an orchestration framework. Ainur employs the Fluentd unified layer for log collection. It natively integrates with Docker containers and allows for the automatic collection of text data from the standard output and error streams of processes executing inside containers. It automatically decouples data sources from containers running on different systems and allows Ainur to collect and classify the logs from any components of the network, whether on the wireless stack or workload.

D. Obtaining Ainur and deploying it on a new testbed

The framework is released as Free and Open Source Software under a permissive Apache license. It can be downloaded from the Ainur repository [10] of the KTH-EXPECA organization on GitHub. Given the software's complexity, the repository also includes detailed documentation on its requirements, configuration, deployment, and execution, as well as links to external resources containing information and guides about related concepts and services.

Ainur makes a number of assumptions about its execution environment. Apart from generic ones regarding the presence of necessary authentication information for remote access to local and remote hosts and services, the framework assumes a split network architecture. Ainur runs in a dedicated control host and the management plane resides in a physically distinct network from the workload data. This is a key requirement to

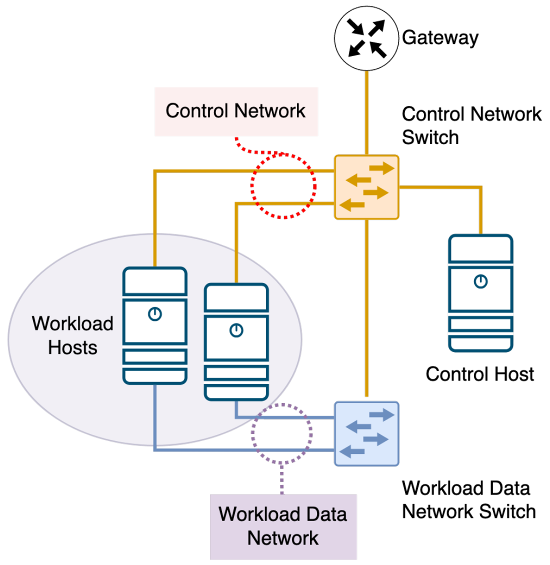


Fig. 4: Minimal testbed setup on which Ainur can be deployed.

be able to reconfigure the physical links in the workload data network without disrupting management traffic.

A full-featured, and rather complex deployment is depicted in Figure 3, including a number of optional testbed features such as offload to the cloud through VPN gateways. However, a minimal a setup can be achieved with a control host, at least two workload hosts, and a couple of managed (smart) switches, as illustrated in Figure 4. The only requirements for the aforementioned hosts are that a general purpose Linux distribution such as Ubuntu can be deployed on them, and — in the case of the workload hosts — the presence of two network interfaces. Thus, a minimal testbed can be achieved cheaply and quickly by employing small single-board computers such as Raspberry Pi, Jetson Nano, or BeagleBone units. Finally, the configuration and deployment of an Ainur-compatible testbed can be easily automated with tools such as Ansible, and playbooks for this effect can be made available to the community upon request.

III. Demo

In this demo, we will show the flexibility of Ainur for running end-to-end experimental workloads on both the Edge and the Cloud. This will be done interactively, and members of the audience will be invited to propose testbed configurations.

Figure 5 illustrates the possible configurations for the testbed. Workloads are deployed on client hosts, and computation is offloaded either to an edge server (cloudlet), or to AWS EC2 instances on the cloud. Communication between the client- and server-sides of the workload will occur over one of three possible physical link layer setups:

WiFi: an SDR is configured as an IEEE 802.11n access point, and client hosts connect to it using on-board WiFi.

4G LTE: an SDR is configured as an 4G LTE base station, and another is configured as an 4G LTE User Equipment (UE). Together these radios act as an LTE bridge between the client-side and the server-side of the network, and all client-server traffic is routed through them.

Ethernet: connects everything through plain ethernet.

The number of workload instances (and therefore client hosts) deployed in each execution of this demonstration will range from 1 to 10. Workloads deployed to the cloud will be able to target any of AWS's datacenters.

We will employ two different workloads for this demonstration, illustrated in Figure 6. The first of these (Figure 6a) consists of a proof-of-concept web application which implements a simple classifier to identify hand-drawn digits from the widely-used Modified National Institute of Standards and Technology (MNIST) dataset [13]. The application consists of a client with a web interface, and a Hyper-Text Transfer Protocol (HTTP) server which responds to requests from the client and performs the actual image recognition. This workload is intended to showcase in an interactive manner the effects of placing computation at different points of the network, and how Ainur simplifies these deployments.

The second workload (Figure 6b) corresponds to a Networked Control System (NCS) balancing an inverted pendulum, implemented on a software framework for the emulation of NCSs using cloud-native technologies [14]. It consists of an emulation of the physical inverted pendulum system plant and a software-implemented proportional-differential controller. These components communicate with each other over the User Datagram Protocol (UDP). This workload will be used to showcase the utility of Ainur for automating the execution of batches of experiments potentially including multiple different clients, servers, and physical layers.

A. Testbed Setup

This demonstration will be performed on a testbed consisting of 1) 10 Raspberry Pi 4 Model B boards, acting as client-side workload hosts; 2) a Raspberry Pi 4 Model B acting as VPN gateway for the workload network; 3) a Raspberry Pi 4 Model B acting as VPN gateway for the management network, as well as hosting the necessary Network Time Protocol (NTP) and Domain Name System (DNS) server software; 4) a `i386` workstation, acting as an edge-side workload host; 5) a configurable number of AWS EC2 cloud instances acting as cloud servers; and finally 6) a separate `i386` workstation hosting the Fluent server and on which Ainur is deployed as well. These nodes are interconnected using a combination of managed switches, SDRs, and VPN gateways; please refer to Figure 3 for an architectural overview of this setup.

B. Demo Procedure

1) Workload 1: Participants will be asked to choose 1) a location to offload computation to (local (i.e. no offloading), edge, or any AWS datacenter); and 2) a physical layer for the first hop of the network (WiFi, 4G LTE, or ethernet). Through the Ainur command line, we will deploy a single client-server pair according to the specifications. Next, participants will be able to access the client web interface and interact with the server by requesting classification of images from the MNIST

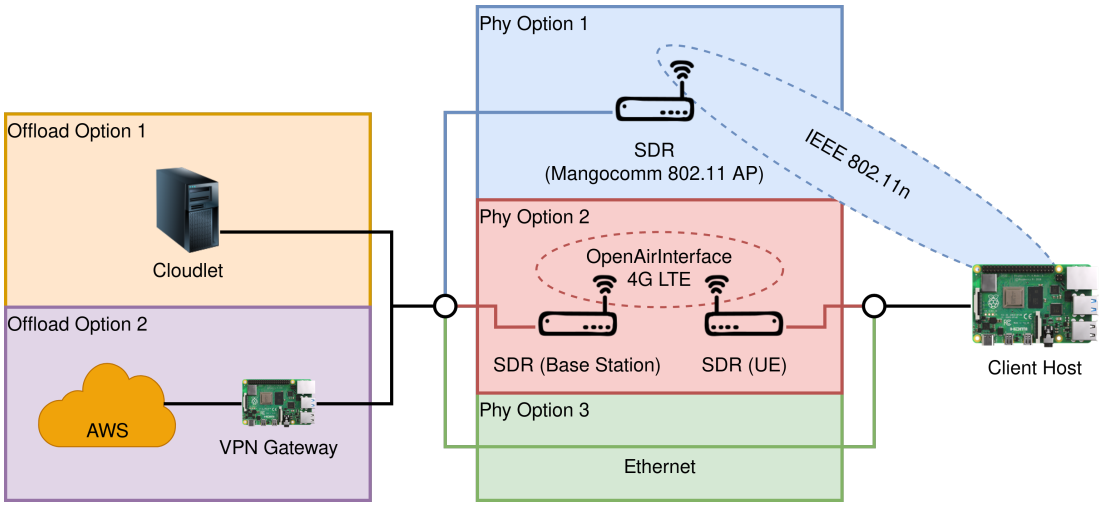
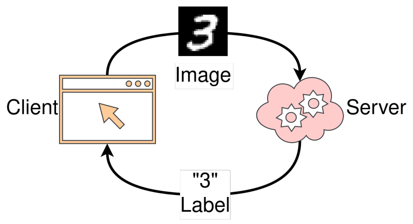
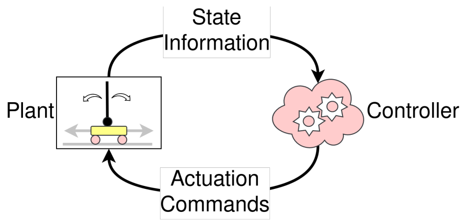


Fig. 5: The possible workload offloading setups and physical layer configurations used in the demo. Not pictured, in Phy Option 2: 1) SDR processing hosts; and 2) additional base-station host for the Core Network and Evolved Node B (eNodeB).

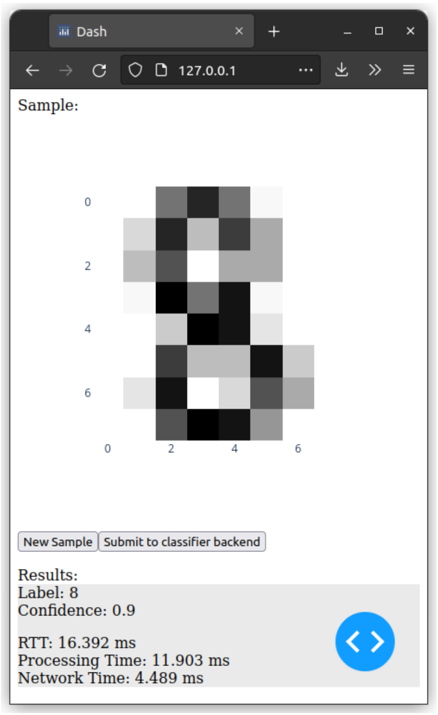


(a) Workload 1: MNIST image classifier



(b) Workload 2: inverted pendulum NCS

Fig. 6: Demonstration workloads



Results:
Label: 8
Confidence: 0.9

RTT: 16.392 ms
Processing Time: 11.903 ms
Network Time: 4.489 ms

Fig. 7: Screenshot of the user interface of demo workload 1. After deciding on the placement of the compute backend and the type of physical layer connecting client and backend, participants will use this interface to interact with the system.

database. The server will respond to these, and the assigned labels will be visible on the web interface together with timing statistics for each request, see Figure 7.

2) Workload 2: Participants will be asked to specify the full, arbitrarily complex, experimental scenario, including the number of clients, physical layers (single or multiple, and which clients are on each physical link), and offloading configuration (number of instances on the edge vs. on the cloud, which AWS datacenters to deploy to). Configuration will be specified through a YAML file, which will then be parsed by Ainur for the automatic execution of the scenario.

IV. Conclusions and Future Directions

In this paper, we have introduced a framework for repeatable end-to-end testbed automation in the context of wireless networking and edge computing research. Named Ainur, it simplifies the execution and verification of end-to-end experimentation by automating the 1) establishment of physical links between hosts, including the configuration of complex wireless systems such as 4G LTE and 5G; 2) provisioning of and connection to remote cloud instances; 3) initialization of IP layer connectivity between hosts; 4) collection of logs and data; and 5) deployment, scaling, and lifecycle management of containerized processes. We have described its general architecture, which follows a layered design mimicking the network stack layers the framework directly interacts with, as well as the underlying assumptions about its deployment environment and specific requirements for its deployment. We have also outlined a demonstration which showcases the flexibility and power of the framework by deploying two different workloads to our testbed. We believe our framework represents an important step towards repeatable, replicable, yet low-access barrier end-to-end wireless testbed experimentation. It has been released as FOSS and can be found on GitHub [10].

In terms of future work, our current efforts are focused on the expansion and integration of Ainur with CHI [6]. While we believe Ainur to be unique and valuable in its focus on fully-automated end-to-end wireless edge computing experimentation, integrating with the above software stacks will provide a number of features and functionalities which will greatly expand Ainur's potential. Features like resource reservation, multi-tenant testbed access, and automated bare-metal and VM-host instantiation will make Ainur better suited for larger scale experimentation and allow us to expand the research domain targeted by the framework. Our ultimate goal is to eventually provide Ainur as a service running inside an OpenStack/CHI environment, leveraging the flexibility of the platform to automate the reservation of resources, instantiation of nodes and networks, execution of experiments, and final collection of results.

Acknowledgements

This research has been partially funded by the Swedish Foundation for Strategic Research (SSF), through grant number ITM17–0246.

We also extend our gratitude to the reviewers for their insightful comments and suggestions, which greatly helped us improve this work.

References

[1] Dipankar Raychaudhuri et al. "Challenge: COSMOS: A City-Scale Programmable Testbed for Experimentation with Advanced Wireless". In: *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. Association for Computing Machinery, 2020.

[2] Jiakai Yu et al. "COSMOS: Optical Architecture and Prototyping". In: *2019 Optical Fiber Communications Conference and Exhibition (OFC)*. 2019, pp. 1–3.

[3] M. Ott et al. "ORBIT testbed software architecture: supporting experiments as a service". In: *First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*. 2005, pp. 136–145.

[4] Joe Breen et al. "POWDER: Platform for Open Wireless Data-driven Experimental Research". In: *Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH)*. Sept. 2020.

[5] Hongwei Zhang et al. "ARA: A Wireless Living Lab Vision for Smart and Connected Rural Communities". In: *Proceedings of the 15th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & CHaracterization*. 2022, pp. 9–16.

[6] Kate Keahey et al. "Lessons Learned from the Chameleon Testbed". In: *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, 2020.

[7] *OpenStack: The Most Widely Deployed Open Source Cloud Software in the World*. URL: https://www.openstack.org/.

[8] Kapil R. Dandekar et al. "Grid Software Defined Radio Network Testbed for Hybrid Measurement and Emulation". In: *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. 2019, pp. 1–9.

[9] Sabarish Krishna Moorthy et al. "CloudRAFT: A Cloud-based Framework for Remote Experimentation for Mobile Networks". In: *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*. 2022, pp. 1–6.

[10] *Ainur GitHub Repository*. URL: https://github.com/KTH-EXPECA/Ainur.

[11] Florian Kaltenberger et al. "OpenAirInterface: Democratizing innovation in the 5G Era". In: *Computer Networks* 176 (2020), p. 107284. ISSN: 1389-1286. URL: https://www.sciencedirect.com/science/article/pii/S1389128619314410.

[12] Dirk Merkel et al. "Docker: lightweight linux containers for consistent development and deployment". In: *Linux journal* 2014.239 (2014), p. 2.

[13] Li Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[14] Manuel Olguín Muñoz et al. "CLEAVE: Scalable and Edge-Native Benchmarking of Networked Control Systems". In: *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking*. Association for Computing Machinery, 2022, pp. 37–42.