CHI@Edge: Supporting Experimentation in the Edge to Cloud Continuum

Kate Keahey
Argonne National Laboratory
Lemont, Illinois, USA
keahey@uchicago.edu

Michael Sherman
University of Chicago
Chicago, IL, USA
shermanm@uchicago.edu

Jason Anderson
University of Chicago
Chicago, USA
diurnalist@gmail.com

Mark Powers
University of Chicago
Chicago, IL, USA
markpowers@uchicago.edu

Abstract

Infrastructure for Computer Science research is a scientific instrument that supports a wide range of experimental scenarios through *interactive, isolated, multi-tenant usage*, that allows many investigators to conduct original experimentation while preserving the privacy of their research and without impacting each other's work. This type of infrastructure needs to evolve as the scientific frontier advances, to support new types of experiments required for investigate new phenomena. Recently, the widespread availability of inexpensive yet powerful single board computers (SBCs) and Internet of Things (IoT) devices, revolutionized the opportunities available to science, but at the same time created a need for a new type of scientific instrument that would support experimentation in the edge to cloud continuum.

This paper describes the design and implementation of such an instrument (CHI@Edge), developed as an extension of the Chameleon Cloud, which has served 11,000+ users working on 1,200+ unique projects in Computer Science research, education, and emergent applications. We describe the requirements developed with our user community, the architecture and implementation of the system, and then illustrate how it has been used by computer science applications.

CCS Concepts

• **Computer systems organization → Cloud computing**; *Embedded and cyber-physical systems.*

Keywords

Edge Computing, Cloud Computing, Edge to Cloud Continuum

ACM Reference Format:
Kate Keahey, Michael Sherman, Jason Anderson, and Mark Powers. 2025. CHI@Edge: Supporting Experimentation in the Edge to Cloud Continuum. In *Practice and Experience in Advanced Research Computing (PEARC '25), July 20–24, 2025, Columbus, OH, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3708035.3736014

1 Introduction

Computing infrastructure continues to evolve at a rapid pace through innovation in architectures, accelerators, and networking technologies. This evolution creates opportunities, but also poses a problem: how can we put such innovative infrastructure in the hands of as many students and scientists as possible so that they can understand and unlock its potential? The solution is to create a *shared scientific instrument* [25], that amortizes expensive and complex hardware procurement, by operating it in a way that that supports as large as set of experiments as possible, for as many users as possible. In computer science this type of scientific instrument is usually called a testbed. To do it's job, a testbed must support *multi-tenant, interactive, and isolated* usage, that allows many investigators to conduct a broad range of original experimentation without impacting each other's work. Such usage is typically enforced by a variety of mechanisms ranging from assigning whole resources for individual use via bare metal reconfiguration for systems research [16, 24]; use of virtual machines where performance isolation is not required [22, 24]; or the creation of isolated networks [7, 30].

Another important characteristic of scientific instruments is that to be useful, they need to evolve as the scientific frontier advances, and new research topics and opportunities emerge that necessitate the support of new types of experiments. In recent years, the widespread availability of inexpensive yet powerful single board computers (SBCs) and Internet of Things (IoT) devices revolutionized the opportunities available to science: a low-power SBC, combined with environmental sensors, can be used to obtain data supporting a better understanding of the environment. At the same time, building systems that use those devices at scale challenges the assumptions we make about infrastructure, system properties, and applications, and thus requires significant new research to make good on the promise of this new technology. To support this research, we need infrastructure that will enable experimentation in the edge to cloud continuum, allowing scientists to deploy and try new approaches to programming in the edge to edge space as well as in the edge to cloud space.

In this paper, we present the architecture and implementation of CHI@Edge, an edge-based infrastructure that extends the experimentation supported by the existing Chameleon Cloud [24], to operate in the edge to cloud continuum. We note that to effectively support interesting edge experiments, we have to extend the traditional concept of infrastructure confined to a datacenter and operated by a team of trained professionals, to allow users to add and operate their own devices as its integral part. We support this capability by introducing the *Bring Your Own Device* (BYOD) paradigm, and a new role of a *device operator* user, supplementing the traditional division between infrastructure operator and end-user roles, and resulting in a "mixed operation" resource. We introduce the architecture and architectural elements needed to support a system of this kind; describe its implementation; and discuss to what extent, this type of edge to cloud continuum infrastructure can be supported by the same abstractions that support the current Chameleon system. Lastly, we describe case studies of experiments that the extended system enables to illustrate its capabilities.

2 The Core Chameleon Infrastructure

Chameleon is an NSF-funded distributed testbed that supports computer science research, education, and emergent applications [24]. To cover the broadest possible range of experiments, Chameleon is configured as a cloud to support interactive usage, with most of the system reconfigurable at the bare metal level – as needed for experiments in performance or power management – and a smaller portion of the system reconfigurable via virtual machine (VM) deployment for more efficient use of resources where bare metal is not needed [24, 28]. Most of Chameleon's hardware is hosted at the University of Chicago (UChicago) and Texas Advanced Computing Center (TACC), soon to be joined by NCAR, as the core resource providers, as well as several volunteer sites. Since its public availability in July 2015, Chameleon has served a total of 12,000+ users (700-800 unique users log into Chameleon every month), working on 1,200+ projects; collectively this community has produced 800+ research publications that we were able to track.

The core Chameleon end-user experimental workflow has been described in detail in [28]; briefly, users first use a resource discovery service and an availability calendar to identify desired resources; allocate selected resources either on-demand or by making an advance reservation; and then reconfigure them using one of the images supported by the Chameleon team or one that they create themselves. Interfaces to the services supporting these actions are available via federated identity authentication [4] and presented through a graphical user interface (GUI), a command line interface (CLI), and programmatically via a python-chi library [12] that can be used from within a Jupyter environment (see below).

Both bare metal and virtualized Chameleon offerings are based on a version of OpenStack [19], a mainstream, open-source implementation of Infrastructure-as-a-Service (IaaS) – with significant customizations made by the Chameleon team overtime to adapt the system to our use case. Examples of the latter include bare metal snapshotting (not available from the main OpenStack installation), support for federated identity access, additional networking features including stitching, integration with Jupyter, and resource discovery services. Many of the modifications were contributed back to OpenStack by the team, in particular, Blazar (OpenStack component handling advance reservations), where the Chameleon team was one of the main contributors.

3 Requirements: the Shape of an Edge Testbed

The objective of computing at the edge is to support interaction with the environment, typically carried out via some form of environmental sensing (e.g., cameras or other IoT sensors interacting with a physical environment), combined with an enhanced ability to compute and communicate. The former implies deployment outside of the datacenter, potentially in a vulnerable physical context where power is not readily available, and in a heterogeneous networking context, giving each edge device unique properties. The latter entails not just the need for computational power but also programmability which we define as support for Linux and at least some mainstream tools. For this reason, we define the edge devices in our system to be SBCs, such as Raspberry Pis or NVIDIA Jetson Nanos, each interacting with potentially multiple *IoT peripherals*.

The central tension underlying the design of this type of infrastructure is how to support sufficiently expressive experimental topologies, given the lesser capability, security, and configurability – and significantly greater deployment heterogeneity of such instruments. In this section we boil this down to key requirements, developed over the last four years with significant input from the Chameleon community over a series of requirement discussions, pilot solution trials, as well as through a community workshop [23].

Rich and consistent testbed interfaces. To support sufficiently expressive experimental topologies the end-user capabilities and interfaces are expected to be roughly the same as in the datacenter testbed. On a basic level the entails support for the general Chameleon experiment pattern [28] consisting of resource discovery (adapted to support ephemeral resource availability); the ability to allocate time on specific devices in a multi-user environment via advance reservations or on-demand [27]; and a non-prescriptive configuration method. In addition, users expect support for a broad range of experimental topologies, including *bi-directional communication* between the cloud and the devices and the devices between themselves. Those interfaces are expected to be consistent with the original testbed and include many popular features such as federated identity access[4], consistent graphical, command-line, and programmatic interfaces accessible via JupyterHub integrated with the testbed [3], and the potential to federate with other testbeds, such as FABRIC [7].

Bring Your Own Device (BYOD). While there is some interest in using edge devices in a shared datacenter context (e.g., for performance comparisons), most users are interested in using their own devices, usually combined with peripherals associated with a specific environmental context [23]. The user expects that by adding the device – or a collection of devices – to the testbed, they will be able to provide secure sharing and programmability of those devices (as per the interfaces above). In contrast to the traditional testbed model, the primary incentive to use the testbed in this case is not so much access to resources as access to reliable methods of resource management and sharing. There are a few implications of this changed modus operandi. First, it invalidates many of the assumptions we make about resources in the datacenter – for example, resources can come up and disappear ephemerally and have to be managed without physical access – which adds a new management layer to testbed infrastructure. Second, unlike in a traditional testbed where resources typically are available to any user in the testbed, the deployment context of edge devices often means that they are only available for *limited sharing* to a whitelist of end-user collaborators (e.g., in an autonomous driving class devices may be deployed by a teacher are relevant to the participating students but not to others outside of the class). Lastly, users adding devices to the testbed now assume the role of a *device operator*, and have to be supported in this role.

Support for IoT devices/sensors. In many cases, users will associate their device with potentially multiple Internet of Things (IoT) peripherals such as sensors (e.g., cameras) or actuators (e.g., such as a throttle on autonomous vehicles [40]). Since we are building experimental systems – rapidly evolving and frequently reconfigured – it is essential that we provide the path to support many different types such IoT peripherals attached to one device in a flexible and customizable manner. This implies not only being able to connect the peripheral to a device in a useful fashion (i.e., such that they can be accessed and programmed by the users), but that this process is adaptable to support new peripherals as they become available with minimal operator assistance, and full support through testbed services like hardware discovery.

Deployment/Networking considerations. The fact that the deployment of devices is likely to be performed by an operator user rather than a network administrator with special privileges limits the assumptions we can make about device connectivity. The general principle we apply is captured by saying that connecting a device to the network should be "no harder than connecting a laptop". In particular, device deployment has to accommodate a variety of network scenarios (e.g., potentially traversing NATs or firewall layers), and should not require the device operator to make any special arrangements (e.g., that the device has been assigned a public IP address) that may not be practical or possible – at the same time, our first requirement of supporting Chameleon modus operandi still holds. Further, although no special network configurations should be required, users interested in research on alternative network mediums or protocols (e.g., software defined radios [14]) should still be able to perform their research, either by creating and using alternative communication channels, or by specializing the platform implementation.

Added to these functional requirements is our general philosophy, successfully deployed in Chameleon, of building things in as much as possible using mainstream software so as to leverage sustainability it brings though working with large development communities.

4 Architecture

Based on the considerations described above, we present an architecture for CHameleon Infrastructure at Edge (CHI@Edge) that supports OpenStack interfaces consistent with the datacenter part of the testbed, in particular, support for the end-user experimental workflow defined in Section 2. Given the lack of support for out-of-band management, such as the IPMI used for Chameleon's baremetal deployment process, we chose to support reconfigurability via container deployment, since it provides a good trade-off between non-prescriptive resource configuration and implementation of isolation that is sufficiently lightweight. This was found acceptable [23], given that SBCs are low in price (if bare metal is required uses can simply buy a device). Users can combine their edge devices with IoT peripherals via a general-purpose peripheral framework allowing them to support arbitrary combinations of devices and peripherals.

The CHI@Edge architecture is depicted in Figure 1. In addition to the roles of operator and end-user present in Chameleon Infrastructure (CHI) [28], we now also have the user role of *device owner/operator*. In order to add a new device to the testbed, the device operator downloads the BYOD client and uses it to configure the device. The configuration process triggers a request to the testbed registration service (Doni), to add the device to the list of available resources together with the policies for limited sharing. The request is authenticated by the user's testbed credentials, which are then used to generate a unique token that is put on the image the device is flashed with. Subsequently, the token is used by the Device Management Services as root of trust. Once a device is added, it becomes a part of the testbed (potentially with limited sharing); the end-user workflow does not thus differ substantially from what we described in Section 2: the end-user can make advance reservations for the device and reconfigure them using container deployment via OpenStack interfaces.

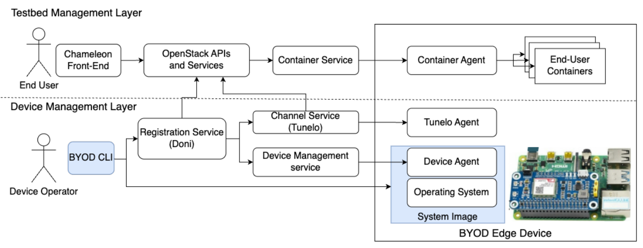


Figure 1: Interactions between the architectural roles and components of CHI@Edge.

Below, we describe components of the architecture and their requirements.

The Doni Registration Service. The Registration Service exposes an API that allows the user to register a device and fill out its profile, rooting trust in the device operator's OpenStack credentials. Subsequently, the device operator can update a device (e.g., change its sharing configuration); report information/status of the device; and refresh/sync device state. Providing an explicit service with well-defined APIs automates inventory management and allows us to handle the complexity of dynamic additions and deletions of potentially ephemeral devices. It is also key to providing support for the device operator role without granting full testbed operator permissions to operator users. This is a new service, implemented specifically to support the BYOD functionality.

Device Management Service, Agent, and system support. The role of the device management component is to provide an underlayer that allows testbed operators to manage devices without physical access. This includes reliably bootstrapping devices remotely without physical access, including the management of configuration, security, and networking. This layer also needs to provide remote access for debugging the configuration of a device, and rollback support, i.e., the ability to recover from failed configuration changes, critical to providing fully remote device management.

The device management component consists of the Device Management service; system support (rollback function); as well as an agent executing on each device.

Channel Service (Tunelo). As part of compatibility with the Chameleon modus operandi, users expect to be able to access instances via public IP address, and to send traffic between instances on non-public networks. CHI@Edge devices are often deployed behind firewalls and NAT, and the existing mechanisms in Neutron, developed for the datacenter use case, do not support connectivity in these conditions. Tunelo orchestrates a network underlayer, enabling communication between containers on different devices, and between containers and networking services on the OpenStack control-plane, that can then serve as a base for the support of such features as floating IPs. This underlayer connection is typically provisioned when a device is registered, but can be updated on-demand if necessary, e.g., to rotate encryption secrets. The channel service manages the lifecycle of these underlayer connections, and its API, while exposed externally, is only consumed by other CHI@Edge services.

BYOD client. The BYOD client consists of a Python library and CLI tooling which is installed and invoked by device operators. It allows the device operator to register a new device with the testbed, download, configure, and flash an SD card image to the device, including necessary tokens or secrets, inspect the device's status, and manage access to the device including which tenants can reserve and use the device. The BYOD client is primarily a client for the registration service (Doni).

OpenStack APIs and Chameleon Front-end. Much of the functionality of CHI@Edge mirrors the main Chameleon testbed and can be provided by the same OpenStack services with exceptions called out in Figure 1. Together, these services process user requests, allocate resources, and manage containers. Relying on the same APIs means that we can re-use Chameleon clients including the command-line interface, python-chi (both integrated with Jupyter), and the GUI for edge devices. In addition to providing user convenience, these APIs allow us to leverage investment in all of the front-end Chameleon features built over time, including support for federated identity, and orchestration features.

Container Management Service and Container Agent. The Container Management Service manages container deployment on the back-end of the OpenStack APIs. This allows us to preserve API compatibility while supporting reconfiguration via container deployment on the edge; aside from support for the ARM64 architecture, the requirements for container implementation are a minimal power footprint.

5 Implementation

Our greatest implementation challenge was to provide a secure device management layer. We chose to base its implementation on Balena [8], because it provides all the basic service/agent functionality needed for secure device management, as well as the critical capability for reliably supporting device bootstrap without physical access. The latter is provided by a rollback feature, implemented via modification of the Operating System's boot process, (configuration changes are written to a secondary partition, and if a reboot onto that partition fails, the rollback feature triggers an automatic reboot to the working configuration). Balena provides a library of base system images supporting this feature on a variety of systems, and build recipes that can be extended to support new systems if needed [9].

We use this reliable base to deploy and manage the necessary runtime configuration of each device, including the Tunelo and container agents, and host-level parameters needed for peripheral management. This runtime configuration, or "fleet" in Balena's terminology, packages a set of Docker containers, configuration for said containers, host-level parameters, and maps it to a set of devices which are members of the fleet. The registration service uses Balena's API to create an entry for each device in the fleet, and to apply per-device overrides for parameters such as authentication tokens. The registration service also consumes the API to generate a unique device bootstrap token; the device operator toolkit injects this token into the image flashed onto the device, and thereby maps a given physical device to the logical one created by the registration service.

The next implementation challenge was to provide lightweight container management while preserving Chameleon interfaces. We explored the OpenStack Zun project [36], but found it too heavyweight for our use case. Instead, we chose to use K3s, which is a small (50MB) single-binary distribution of K8s; it has good support for ARM architectures and is designed with devices like the Raspberry Pi and Jetson Nano in mind. We combined this implementation with a Zun API by writing a new plugin to translate Zun's container driver interface into the K8s API server interface; replacing Zun's default plugin which communicates with Docker daemon directly. This gave us the desired combination of rich and consistent interfaces with lightweight implementation.

To provide both connectivity and protection for user traffic, we decided to implement an overlay network, consisting of a hub-and-spoke topology implemented via WireGuard tunnels. The hub-and-spoke topology was chosen because the CHI@Edge management nodes (the hub) have publicly routable IPs, needed so that edge devices behind NAT can initiate a tunnel to that public address thereby creating a connection (the spoke). Furthermore, the Floating IP service is the primary consumer of the tunnels and is co-located with the hub port. Finally, the hub port acts as a security control, enforcing a whitelist of allowed source and destination subnets, preventing a rogue device from bridging to an arbitrary local network. WireGuard was chosen due to its lightweight encryption protocol built on asymmetric keys, and robust support for traversing NAT. The Tunelo service provides an API for creating and managing channels, each channel corresponding to a pair of tunnel endpoints. When a channel is created or updated in Tunelo, the "hub" end of the tunnels, being co-located with the Neutron networking service, is configured by a custom Neutron "Wireguard Agent" and ML2 plugin, while the "spoke" end on the devices is configured by the "Tunelo Agent" running on each device.

While this enables communication between two arbitrary edge devices, each potentially behind a NAT, it does not address latency, bandwidth, or network scaling issues such as excessive broadcast traffic or flooding. Additionally, using only the tunnel interface would mean that this communication is always routed through CHI@Edge management nodes even if the devices are on the same local network. For this reason, networking between containers is implemented by the Calico CNI (Kubernetes Container Network Interface), with a separate subnet on each device, and BGP routing and IPIP encapsulation to enable routing packets between containers on different devices. As each node running Calico (device or management server) participates in BGP, all devices can route via the hub port, but can also use a shorter path if provided by a common local network. Using this framework, we implement standard OpenStack networking semantics: floating IPs are implemented by routing between the Neutron L3 agent and container IPs on Calico networks, and security groups are supported by mapping them to Calico Network Policies. Dedicated, tenant-isolated, Layer 2 networks (as implemented by OpenStack via VxLAN or VLAN) are not supported, per-tenant isolation is instead implemented via ACLs, a default policy restricting communication to containers owned by the same tenant, the same semantics as instances launched on the same Openstack Network. While we do lose the ability to directly attach containers to L2 segments, useful for certain experiments such as ultra-low latency protocol development [35], as future work, we are investigating approaches to pass physical interfaces directly to containers, bypassing this limitation.

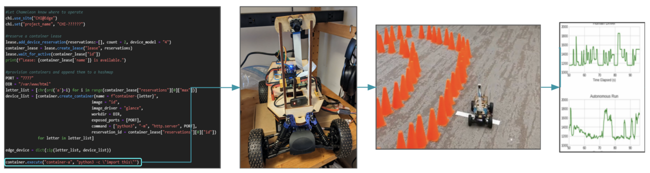
The approach to securing the system in CHI@Edge is based on the tenet that while we cannot guarantee that individual devices won't be compromised (in particular since we don't always control them), we can't allow a compromise to spread to the system or resources that it uses such as local networks. For this reason, control plane software running on edge devices has limited credentials, i.e., it can only enumerate resources on the cluster and issue writes against resources under the device's purview (e.g., its config or containers running on it.). Further, traffic between the control plane and the device uses an encrypted websocket connection (part of the K3s implementation) to avoid malicious man-in-the-middle attacks taking over the device. While containers on edge devices may be able to scan or access local network resources, we mitigate this with traffic rules that we provision on the device at enrollment; since however such access may actually be desirable this is configurable by the device owner (but not container owner). Similarly, we protect ingress to containers via security groups configured on container deployment, but we allow the container owner to override those protections at their own peril. Since containers can only be launched on devices that are reserved this is where limited sharing is implemented: the user specifies policies on who can access the device on registration or via the SDK and the information is passed to Blazar which enforces them.

6 CHI@Edge Use for Education and Research

CHI@Edge has been used in a variety of edge to cloud research projects in areas such as network fingerprinting, federated learning, and in a range of projects exploring using edge to cloud continuum in domain science. Some of those use cases are described in the report from the 3rd Chameleon User Meeting [23], and some are described via the Chameleon user blog or in reported user publications [13, 18, 20, 21, 41, 44]. In this section, we walk through two case studies, each describing how the requirements laid out in Section 3 are used in the implemented system to create experimental environments; for results relating to these projects see the referenced publications.

6.1 AutoLearn

Autonomous driving is a fun and experiential way of introducing students to concepts ranging from engineering to ML. In this case study, mainstream SBCs are either tied to a custom vehicle or integrated in a commercially available model, equipped with a camera and a steering system: the SBC receives input from the camera and provides commands to the throttle to steer the car. The full instruction or experiment cycle consists of data collection from a manually driven vehicle; training phase where the obtained data is used to train models in the cloud; and inference phase where the models are used to drive the vehicle autonomously. CHI@Edge has been used in this capacity for education in Rick Anderson's "Foo Cars" project [5] at Rutgers University using the DonkeyCar autonomous vehicle software framework [15]; has been translated into an instructional module packaged for use on Chameleon [17, 18]; and ultimately leveraged to support independent student projects [20] [44].



```
#let Chameleon know where to operate
chi.use_site("CHI@Edge")
chi.set("project_name", "CHI-??????")

#reserve a container lease
lease.add_device_reservation(reservations:=[], count = 2, device_model = "4")
container_lease = lease.create_lease("lease", reservations)
lease.wait_for_active(container_lease["id"])
print(f"Lease: {container_lease['name']} is available.")

#provision containers and append them to a hashmap
PORT = "7777"
DIR = "/var/www/html"
letter_list = [chr(ord('a')+i) for i in range(container_lease["reservations"][0]["max"])]
device_list = [container.create_container(name = f"container-{letter}",
                                          image = "id",
                                          image_driver = "glance",
                                          workdir = DIR,
                                          exposed_ports = [PORT],
                                          command = ["python3", "-m", "http.server", PORT],
                                          reservation_id = container_lease["reservations"][0]["id"])
               for letter in letter_list]

edge_device = dict(zip(letter_list, device_list))

container.execute("container-a", "python3 -c \"import this\"")
```

Figure 2: Experimentation with autonomous vehicles. Left to right: programming the vehicle from a Jupyter notebook; custom-made autonomous vehicle with camera, steering system, and Raspberry Pi; data collection along a self made track; comparison of steering data between human-controlled and autonomous runs.

This case study illustrates how the BYOD supports the experimental cycle by first allowing the user to add the SBCs associated with the car to Chameleon; this is easily done by ensuring that the SBC is connected to a local wifi and following the device operator workflow. The detection of peripherals, in this case, camera and serial interface, takes place automatically. As a result of this process, the SBCs, each representing a vehicle, become available via the testbed resource discovery mechanisms for limited sharing with members of a research group or a class, with the group lead or the class teaching assistant (TA) assuming the role of device operator. Members of the group/class can then allocate the devices, including via advance reservations, and deploy instances on them for data collection or inference; allocate additional GPU resources on Chameleon as needed for training purposes, or execute edge to cloud inference workloads [44]. Since both CHI@Edge and the core Chameleon testbed support consistent interfaces and front-end infrastructure, users can program the cycle from one collection of Jupyter notebooks integrated with the testbed, effectively programming all the resource allocation across edge and cloud from one session.

6.2 5G in Practice

Fifth-generation (5G) wireless technologies offer the promise of better Internet access in rural communities; in order to prove it we need to measure it. Ideally, we'd like to understand how 5G wireless technologies compare to wired connections and existing rural wireless in speed (throughput and latency), and in supporting distributed computation workloads. One way to do this is to deploy SBCs, equipped with a mix of mmWave 5G radios, fiberoptic network connections, and pulse-per-second (PPS) GPS timing signals in key locations across an area of interest and then compare network speed over different connections. This approach was deployed in a project measuring network speeds across rural Iowa [34].

Using CHI@Edge BYOD, a researcher was able to deploy several Raspberry Pi devices, configured with the IoT peripherals enumerated above, across a 6 mile radius, including university buildings, city buses, and farms, to cover a variety of wireless environments. Once deployed, the Pis became programmable; the researcher could now deploy several distinct applications representing different measurement methods, ranging from simple bandwidth and latency measurements to both local and distant servers, to an additional application packaging Apache Hadoop to run the TeraSort benchmark [6] illustrating how the scaling of this workload changes if the workers are connected via a 5G wireless connection versus a wired Ethernet one. The data from those measurements was sent to services deployed in the datacenter to collect, store, and analyze results. Once, the BYOD deployment portion of the experiment was accomplished, the edge to cloud workflow consisted of allocating and deploying the different measurement applications; cloud storage for data; long-running compute cycles for the necessary services on the KVM part of Chameleon; and analysis resources on the bare metal partition. As before, common interfaces meant that the whole workflow together could be supported from a consistent set of Jupyter notebooks.

7 Related Work

The most relevant comparison of our work is to other *exploratory testbeds* supporting computing at the edge, including projects like COSMOS [38], ORBIT [37], POWDER [11], AERPAW [29], ARA [43], KTH's ExPECA [33] and the FIT IoT Lab [1]. These infrastructures support isolated multi-tenant use, but focus on fewer, more powerful and expensive edge devices, that are deployed and operated by the testbed rather than users, and therefore assume full control over the network between edge devices and the testbed core. This means that they don't support BYOD and everything it entails, in particular overlay networks or tunnel services to ensure secure and isolated network use. We note that while ARA falls into this category, it has derived its operations framework from CHI@Edge and thus could be extended to provide such support as needed.

Observatory infrastructure, such as FLOTO [26] and Sage [39], focus on experiments that support observation and measurement [26]. These platforms support isolated multi-tenant use via container-based reconfiguration, allowing for a wide range of interactions but limit their focus to observational applications (e.g., processing camera inputs or running tests of Internet infrastructure) and send collected data to a central aggregation point. These types of infastructure disallow inbound traffic to the devices (only the device sends data) which simplifies the security structure so that overlay networks or tunnel services are again not needed.

Commercial IoT systems such as AWS IoT Greengrass [2], Azure IoT Edge [32], Balena [8], and Mender [31] focus on targeted edge deployments that install an agent on a device and then use the agent to manage the device to support a specific application and thus do not support the isolated multi-tenant pattern. CHI@Edge is built on top of OpenBalena, and adds support for the isolated multi-tenant pattern as well as other features such as advance reservations allowing for interactive and timed usage and support for federated identity.

Some approaches choose to use more "prescriptive" methods of interacting with a device, limiting what types of actions a user can deploy on a device (e.g., by using function as a service approach). CSPOT [42] and Stack4Things [10] fall into this category; the narrower interfaces they provide are capable of supporting only a subset of our use cases but have the advantage of being more easily sustained on lower power devices.

8 Conclusions

In this paper, we describe CHI@Edge, an extension of Chameleon – a popular testbed for computer science research operated as a cloud – to the edge. We note that this extension changes the understanding of what research infrastructure may constitute. Traditionally, it has been understood primarily as datacenter-based hardware resources, secure and operated by professionals. In contrast, CHI@Edge represents a type of resource in which relatively inexpensive resources, usually deployed outside of a datacenter, are part operated by a specialized type of user, the device operator. This suggests that the key service provided by the infrastructure is an effective implementation of sharing and a connection to residual/cloud testbed resources, shareable in ways consistent with the edge devices.

Our case studies show examples of research applications/experiments that can effectively leverage resources available in the "edge to cloud continuum". We note that in practice, the work of such applications will touch on many types of continuum: the computing continuum as users move from powerful – and high powered – datacenter resources to less powerful resources with more modest power requirements, configuration continuum as we move from the ability to reconfigure resources at bare metal to methods allowing greater encapsulation, or networking continuum as we move from powerful and fully controlled networks to a mix of fabrics with lesser availability and reliability. Achieving effective programmability over this type of infrastructure will require building abstractions supporting computation is likely to move dynamically based on the interplay of multiple optimization factors.

Acknowledgments

Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation grant number 2027170. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

References

[1] Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and Thomas Watteyne. 2015. FIT IoT-LAB: A large scale open experimental IoT testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, Milan, Italy, 459–464. doi:10.1109/WF-IoT.2015.7389098

[2] Amazon. 2023. AWS IoT Greengrass. https://aws.amazon.com/greengrass/ [Online; accessed 21-December-2023].

[3] Jason Anderson and Kate Keahey. 2019. A case for integrating experimental containers with notebooks. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom* 2019-December (12 2019), 151–158. doi:10.1109/CLOUDCOM.2019.00032

[4] Jason Anderson and Kate Keahey. 2022. Migrating towards Single Sign-On and Federated Identity. In *Practice and Experience in Advanced Research Computing 2022: Revolutionary: Computing, Connections, You* (Boston, MA, USA) *(PEARC '22)*. Association for Computing Machinery, New York, NY, USA, Article 8, 8 pages. doi:10.1145/3491418.3530770

[5] Rick Anderson. 2023. FooCars. https://github.com/fubarlabs/foocars [Online; accessed 21-December-2023].

[6] Apache Software Foundation. 2024. TeraSort - Apache Hadoop. https://hadoop.apache.org/docs/stable/api/org/apache/hadoop/examples/terasort/package-summary.html Apache Hadoop 3.3.6 API Documentation.

[7] Ilya Baldin, Anita Nikolich, James Griffioen, Indermohan Inder S. Monga, Kuang Ching Wang, Tom Lehman, Paul Ruth, and Ewa Deelman. 2019. FABRIC: A National-Scale Programmable Experimental Network Infrastructure. *IEEE Internet Computing* 23 (11 2019), 38–47. Issue 6. doi:10.1109/MIC.2019.2958545

[8] "Balena". 2023. Balena Cloud. https://www.balena.io/ [Online; accessed 20-December-2023].

[9] Balena. 2025. Customer Board Support. https://docs.balena.io/reference/OS/customer-board-support/ Last accessed 30 June 2025.

[10] Zakaria Benomar, Francesco Longo, Giovanni Merlino, and Antonio Puliafito. 2020. A Stack4Things-based Web of Things Architecture. In *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*. IEEE, Rhodes, Greece, 113–120. doi:10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics50389.2020.00036

[11] Joe Breen, Andrew Buffmire, Jonathon Duerig, Kevin Dutt, Eric Eide, Mike Hibler, David Johnson, Sneha Kumar Kasera, Earl Lewis, Dustin Maas, Alex Orange, Neal Patwari, Daniel Reading, Robert Ricci, David Schurig, Leigh B. Stoller, Jacobus Van der Merwe, Kirk Webb, and Gary Wong. 2020. POWDER: Platform for Open Wireless Data-driven Experimental Research. In *Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization* (London, United Kingdom) *(WiNTECH '20)*. Association for Computing Machinery, New York, NY, USA, 17–24. doi:10.1145/3411276.3412204

[12] ChameleonCloud. 2023. python-chi. https://python-chi.readthedocs.io/en/latest/ [Online; accessed 20-December-2023].

[13] Khushi Choudhary, Nona Nersisyan, Edward Lin, Shobana Chandrasekaran, Rajiv Mayani, Loïc Pottier, Angela P. Murillo, Nicole K. Virdone, Kerk Kee, and Ewa Deelman. 2022. Application of Edge-to-Cloud Methods Toward Deep Learning. In *2022 IEEE 18th International Conference on e-Science (e-Science)*. IEEE, Salt Lake City, UT, USA, 415–416. doi:10.1109/eScience55777.2022.00065

[14] Markus Dillinger, Kambiz Madani, and Nancy Alonistioti. 2005. *Software defined radio: Architectures, systems and functions.* John Wiley & Sons, Hoboken, NJ.

[15] donkeycar. 2023. donkeycar. https://docs.donkeycar.com/ [Online; accessed 21-December-2023].

[16] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The design and operation of cloudlab. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference* (Renton, WA, USA) *(USENIX ATC '19)*. USENIX Association, USA, 1–14.

[17] Alicia Esquivel Morel. 2023. Chameleon Trovi Artifact: AutoLearn - Learning in the Edge to Cloud Continuum. https://www.chameleoncloud.org/experiment/share/8800ebd1-411e-4e94-9b62-6883f09188e7/version/2023-07-12

[18] Alicia Esquivel Morel, William Fowler, Kate Keahey, Kyle Zheng, Michael Sherman, and Richard Anderson. 2023. AutoLearn: Learning in the Edge to Cloud Continuum. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis* (Denver, CO, USA) *(SC-W '23)*. Association for Computing Machinery, New York, NY, USA, 350–356. doi:10.1145/3624062.3624101

[19] OpenInfra Foundation. 2023. OpenStack. https://www.openstack.org/

[20] William Fowler, Kate Keahey, and A Esquivel Morel. 2023. Road to reliability: Optimizing self-driving consistency with real-time speed data. In *ACM Student Research Competition Posters Display (SC'23 Poster)*. IEEE Press, Denver, CO, USA.

[21] Akram Hakiri, Sadok Ben Yahia, and S Gokhale Aniruddha. 2023. Hyper-5G: A Cross-Atlantic Digital Twin Testbed for Next Generation 5G IoT Networks and Beyond. In *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*. 230–235. doi:10.1109/ISORC58943.2023.00041

[22] David Y. Hancock, Jeremy Fischer, John Michael Lowe, Winona Snapp-Childs, Marlon Pierce, Suresh Marru, J. Eric Coulter, Matthew Vaughn, Brian Beck, Nirav Merchant, Edwin Skidmore, and Gwen Jacobs. 2021. Jetstream2: Accelerating Cloud Computing via Jetstream. In *Practice and Experience in Advanced Research Computing* (Boston, MA, USA) *(PEARC '21)*. Association for Computing Machinery, New York, NY, USA, Article 11, 8 pages. doi:10.1145/3437359.3465565

[23] Kate Keahey, Jason Anderson, Michael Sherman, Zhuo Zhen, Mark Powers, Isabel Brunkan, and Adam Cooper. 2021. Chameleon@Edge Community Workshop Report. doi:10.5281/zenodo.5777344

[24] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons learned from the Chameleon testbed. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'20)*. USENIX Association, USA, Article 15, 15 pages.

[25] Kate Keahey and Ewa Deelman. 2020. The Silver Lining. *IEEE Internet Computing* 24 (7 2020), 55–59. Issue 4. doi:10.1109/MIC.2020.3013361

[26] Kate Keahey, Nick Feamster, Guilherme Martins, Mark Powers, Marc Richardson, Alexis Schrubbe, and Michael Sherman. 2023. Discovery Testbed: An Observational Instrument for Broadband Research. In *2023 IEEE 19th International Conference on e-Science (e-Science)*. IEEE, 1–4. doi:10.1109/e-Science58273.2023.10254876

[27] Kate Keahey, Pierre Riteau, Jason Anderson, and Zhuo Zhen. 2019. Managing Allocatable Resources. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 41–49. doi:10.1109/CLOUD.2019.00019

[28] Kate Keahey, Pierre Riteau, Dan Stanzione, Tim Cockerill, Joe Mambretti, Paul Rad, and Paul Ruth. 2019. Chameleon: A Scalable Production Testbed for Computer Science Research. *Contemporary High Performance Computing* (7 2019), 123–148. doi:10.1201/9781351036863-5

[29] Vuk Marojevic, Ismail Guvenc, Rudra Dutta, Mihail L Sichitiu, and Brian A Floyd. 2020. Advanced Wireless for Unmanned Aerial Systems: 5G Standardization, Research Challenges, and AERPAW Architecture. *IEEE Vehicular Technology Magazine* 15 (2020), 22–30. Issue 2. doi:10.1109/MVT.2020.2979494

[30] Rick McGeer, Mark Berman, Chip Elliott, and Robert Ricci. 2018. *The GENI Book* (1st ed.). Springer Publishing Company, Incorporated.

[31] "mender.io". 2023. Mender.io: Over-the-Air Software Updates for IoT Devices. https://mender.io/ [Online; accessed 20-December-2023].

[32] Microsoft. 2023. Azure IoT Edge. https://azure.microsoft.com/en-us/products/iot-edge [Online; accessed 21-December-2023].

[33] Samie Mostafavi, Vishnu Narayanan Moothedath, Stefan Ronngren, Neelabhro Roy, Gourav Prateek Sharma, Sangwon Seo, Manuel Olguin Munoz, and James Gross. 2024. ExPECA: An Experimental Platform for Trustworthy Edge Computing Applications. In *Proceedings of the Eighth ACM/IEEE Symposium on Edge Computing* (Wilmington, DE, USA) *(SEC '23)*. Association for Computing Machinery, New York, NY, USA, 294–299. doi:10.1145/3583740.3626819

[34] Zack Murry, Alicia Esquivel Morel, and Kate Keahey. 2024. 5G in Practice: Measuring Emerging Wireless Technology in Rural Iowa for Edge Devices in Distributed Computation Workloads. In *ACM Student Research Competition Posters Display (SC'24 Poster)*.

[35] Ahmed Nasrallah, Akhilesh S. Thyagaturu, Ziyad Alharbi, Cuixiang Wang, Xing Shao, Martin Reisslein, and Hesham ElBakoury. 2019. Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G Ull research. *IEEE Communications Surveys and Tutorials* 21 (2019). Issue 1. doi:10.1109/COMST.2018.2869350

[36] "Openstack". 2023. Openstack Zun. https://docs.openstack.org/zun/latest/ [Online; accessed 20-December-2023].

[37] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. 2005. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. In *IEEE Wireless Communications and Networking Conference, 2005*, Vol. 3. 1664–1669 Vol. 3. doi:10.1109/WCNC.2005.1424763

[38] Dipankar Raychaudhuri, Ivan Seskar, Gil Zussman, Thanasis Korakis, Dan Kilper, Tingjun Chen, Jakub Kolodziejski, Michael Sherman, Zoran Kostic, Xiaoxiong Gu, Harish Krishnaswamy, Sumit Maheshwari, Panagiotis Skrimponis, and Craig Gutterman. 2020. 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* (London, United Kingdom) *(MobiCom '20)*. Association for Computing Machinery, New York, NY, USA, Article 14, 13 pages. doi:10.1145/3372224.3380891

[39] Sage. 2023. SAGE: A Software-defined Sensor Network. https://sagecontinuum.org/ [Online; accessed 21-December-2023].

[40] Gonzalo De La Torre, Paul Rad, and Kim-Kwang Raymond Choo. 2020. Driverless vehicle security: Challenges and future research opportunities. *Future Generation Computer Systems* 108 (7 2020), 1092–1111. doi:10.1016/j.future.2017.12.041

[41] Jonathan Tsen, Jason Anderson, Leonardo Bobadilla, and Kate Keahey. 2021. One fish two fish: Choosing optimal edge topologies for real-time autonomous fish surveys. In *ACM Student Research Competition Posters Display (SC'21 Poster)*. IEEE Press, St. Louis, MO, USA.

[42] Rich Wolski, Chandra Krintz, Fatih Bakir, Gareth George, and Wei-Tsung Lin. 2019. CSPOT: portable, multi-scale functions-as-a-service for IoT. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing* (Arlington, Virginia) *(SEC '19)*. Association for Computing Machinery, New York, NY, USA, 236–249. doi:10.1145/3318216.3363314

[43] Hongwei Zhang, Yong Guan, Ahmed Kamal, Daji Qiao, Mai Zheng, Anish Arora, Ozdal Boyraz, Brian Cox, Thomas Daniels, Matthew Darr, Doug Jacobson, Ashfaq

Khokhar, Sang Kim, James Koltes, Jia Liu, Mike Luby, Larysa Nadolny, Joshua Peschel, Patrick Schnable, Anuj Sharma, Arun Somani, and Lie Tang. 2021. 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* (New Orleans, LA, USA) *(WiNTECH '21)*. Association for Computing Machinery, New York, NY, USA, 9–16. doi:10.1145/3477086.3480837

[44] Kyle Zheng, Kate Keahey, and Alicia Esquivel Morel. 2023. Chasing Clouds with Donkeycar: Holistic Exploration of Edge and Cloud Inferencing Trade-Offs in E2E Self-Driving Cars. In *ACM Student Research Competition Posters Display (SC'23 Poster)*. IEEE Press, Denver, CO, USA.