LinkLab 2.0: A Multi-tenant Programmable IoT Testbed for Experimentation with Edge-Cloud Integration

Wei Dong, Borui Li, Haoyu Li, Hao Wu, Kaijie Gong, Wenzhao Zhang, and Yi Gao, *Zhejiang University*

https://www.usenix.org/conference/nsdi23/presentation/dong

This paper is included in the Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

LinkLab 2.0: A Multi-tenant Programmable IoT Testbed for Experimentation with Edge-Cloud Integration

Wei Dong[†], Borui Li[†], Haoyu Li, Hao Wu, Kaijie Gong, Wenzhao Zhang, Yi Gao[✉]
College of Computer Science, Zhejiang University,
Alibaba-Zhejiang University Joint Institute of Frontier Technologies, China
{dongw, libr, lihy, wuh, gongkj, zhangwz}@emnets.org, gaoyi@zju.edu.cn

Abstract

In this paper, we present **LinkLab 2.0**, a completely programmable and controllable IoT testbed with the support of edge devices and cloud infrastructures. To be more specific, LinkLab 2.0 leverages a tiered architecture for the programmable devices and the management system to achieve scalability. To better support the integrated experiment among IoT, edge and cloud, LinkLab 2.0 provides one-site programming support and leverages the customizable offloading with serverless functions. Moreover, LinkLab 2.0 proposes a device-involved multi-tenancy approach to ensure responsiveness for concurrent requests. Furthermore, targeting 24/7 availability for experimenters, LinkLab 2.0 leverages proactive and reactive anomaly detection to improve the reliability of the testbed. Finally, we describe the supported research experiments and the outreach usage by external users. We also report lessons learned from the four-year operation. LinkLab 2.0 has supported experiments for 2,100+ users. The accumulated usage time across all the devices exceeds 17,300 hours.

1 Introduction

Many modern IoT systems are deeply integrated with edge and cloud platforms. New edge computing platforms like NVIDIA Jetson, and new computing technologies like computational offloading [41, 47] and serverless computing [27, 51] greatly enhance the capabilities of IoT systems [9, 26, 30, 49] and will eventually usher in an era of *Internet of Everything*. For example, in an industrial machine power monitoring scenario [34], hundreds or thousands of IoT devices monitor and collect energy data at a high frequency. To reduce the bandwidth usage and improve the real-time performance, edge devices are usually required to perform data prepossessing and analytics before forwarding the data to the cloud.

However, a major challenge is the lack of a fully programmable testbed for allowing the community to deeply explore new cloud/edge technologies and their "sweet spot" with a *large* number of IoT devices and *highly heterogeneous* computing platforms.

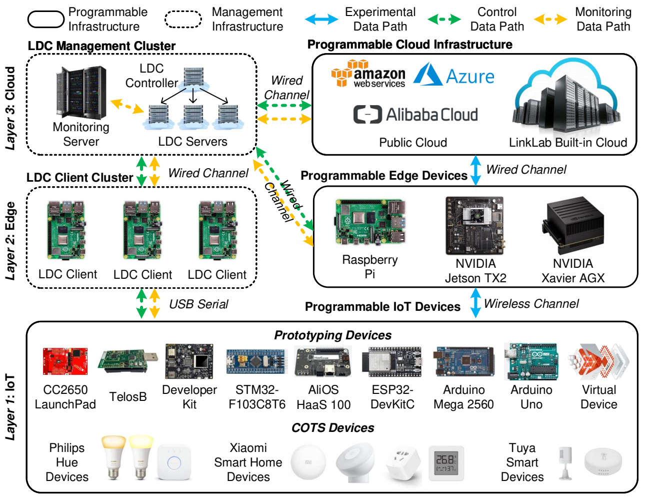


Figure 1: Overview of LinkLab 2.0.

We notice that there exist multiple sensor network testbeds such as MoteLab [67], Indriya2 [7], and FIT IoT Lab [1], allowing the research community to experiment with various sensornet/IoT hardware and IoT software. *Unfortunately*, they do not fully address the aforementioned challenge. Specifically, these testbeds do not natively support edge/cloud integration. Most testbeds do not support the device-edge-cloud communication path and do not allow programming on the edge devices. Moreover, they do not have good support for multi-tenant and high concurrent online experiments with a growing need for teaching and research purposes in the COVID-19 era. In this paper, we present **LinkLab 2.0**, a multi-tenant IoT testbed with edge-cloud integration, aiming to address the following systems and engineering challenges:

(1) How to support device-edge-cloud integrated experiments? Towards this, LinkLab 2.0 enables *one-site integrated programmability and control* for IoT, edge and cloud devices with several front-end and back-end supports. LinkLab 2.0 also supports new computation paradigms by supporting *customizable offloading with serverless functions*. Moreover, the

[†]Co-primary authors

Table 1: Functionality comparison of existing IoT testbeds.

Testbeds	Remote Develop	Edge Support	Cloud Support	Virtual Devices	Web IDE
MoteLab [67]	✓	✘	✘	✘	✘
Indriya2 [7]	✓	✘	✘	✘	✘
FIT IoT [1]	✓	✘	✓*	✘	✘
LinkLab 2.0	✓	✓	✓	✓	✓

*FIT IoT Lab must work with FIT Cloud for cloud-IoT experiments

above programming support requires a separate and reliable channel to deploy experiments onto the devices. Hence, LinkLab 2.0 employs a *vNIC-based bandwidth reservation mechanism* on edge and cloud devices to guarantee the timeliness for controlling the devices (§3.2).

(2) How to ensure dynamic and dedicated usage with a high level of concurrency and multi-tenancy? The *Kubernetes-based architecture* makes LinkLab 2.0 adaptive to highly fluctuating usages (§3.1). Furthermore, considering the concurrent programming requests in the online education scenario, LinkLab 2.0 leverages a *device-involved multi-tenancy* technique to divide a proportion of services and devices as a tenant for dedicated usage. LinkLab 2.0 also provides a nimble configuration interface for administrators to manage the tenants (§3.3).

(3) How to ensure a high level of reliability, especially for the IoT devices? In accordance with the complicated potential root causes of IoT devices, LinkLab 2.0 uses a *proactive and reactive problem detection approach* to detect whether the devices are broken and locate the error as soon as possible. For the whole testbed, LinkLab 2.0 detects anomalies by automatically analyzing the *multi-model logs* during the operation of the testbed (§3.4).

Figure 1 shows the overall architecture of LinkLab 2.0. LinkLab 2.0 consists of three layers: device layer, edge layer and cloud layer. In each layer, there are various programmable devices to facilitate different levels of programmability and control for users. Besides programmable devices, there are dedicated devices and services to manage the programmable devices, namely LinkLab 2.0 Device Center (LDC). There are three different data paths among different layers. The experimental data path is used in the experiments conducted by users. The control data path is used for programming and controlling the devices of LinkLab 2.0, while the monitoring data path is used for experimental data collection and system monitoring of LinkLab 2.0 which is important for guaranteeing 24/7 availability.

Currently, LinkLab 2.0 is equipped with 420+ real IoT/edge devices of 14 different types. Furthermore, LinkLab 2.0 supports theoretically unlimited virtual devices with device-level simulation and a web-based IDE for easier access to the devices. The controller-server-client architecture of LDC allows LinkLab 2.0 to scale easily to accommodate substantial IoT devices at different physical sites. Table 1 compares the functionality of LinkLab 2.0 with other well-known testbeds.

LinkLab 2.0 (`https://linklab.emnets.cn`) facilitates researchers to conduct a broad range of experiments to explore new system designs. It incorporates various embedded computing platforms (e.g., Arduino, ESP32), IoT protocols (e.g., LoRa, MQTT, COAP) and techniques for edge computing (e.g., container-based service composition, edge AI). Furthermore, during the four-year operation, we extend LinkLab 2.0's ability to better serve the community, especially for educational purposes. In §5, we exemplify the supported experiments and outreaches of our testbed to showcase the various capabilities of LinkLab 2.0.

2 Basics and Usage of LinkLab 2.0

Building and managing a testbed with numerous heterogeneous devices from the device, edge and the cloud layer at the same time need a prudent design. In this section, we first present the bird's-eye view and the usage of LinkLab 2.0, then we compare LinkLab 2.0 with the existing testbeds.

LinkLab 2.0 in a nutshell. Towards the aforementioned goal, as Figure 1 shows, LinkLab 2.0 exhibits a three-layer architecture for both hardware and software, namely the IoT device layer, edge layer and cloud layer. LinkLab 2.0 supports both *real* and *virtual* devices for users to program with.

Currently, LinkLab 2.0 includes over 420 real devices for IoT, edge and cloud programming. The IoT devices are deployed in various environments (e.g., multi-hop scenario) as shown in Figure 2. These devices also incorporate various sensing peripherals and networking technologies for users. Another key hardware building block is the programmable edge devices and cloud server (also listed in Table 2), which is currently not well-supported by other testbeds such as FIT [1] and CloudLab [18]. For the programmable cloud server, LinkLab 2.0 provides users with a built-in cloud infrastructure with general-purpose computing resources (i.e., CPU, GPU). Moreover, LinkLab 2.0 supports users to use the public cloud service such as Microsoft Azure or the users' own server as the cloud node in their experiment.

In addition to the real devices, LinkLab 2.0 also introduces *virtual devices* to support the experiments that are *large-scale* or *trace-driven*. We propose two kinds of virtual devices: code-level and message-level. The code-level virtual device accepts the same code as the real node and simulates all behaviors of the device. The message-level virtual device only simulates the network behavior such as MQTT publish, which enables a theoretically unlimited number of devices for large-scale experiments. Furthermore, LinkLab 2.0 also supports binding time-stamped datasets to virtual devices to reproduce an experiment with a pre-recorded trace.

Lifecycle of an experiment. LinkLab 2.0 provides comprehensive support for users to carry out experiments. Conducting experiments contains the following steps:

(1) *Project Creating and Resource Claiming*: Users should first create a project, select the hardware and claim the occupation time via our web portal. The experiment automatically terminates when the requested time quota runs out.

(2) *Programming and Provisioning*: For IoT devices, users

Table 2: List of deployed programmable devices in LinkLab 2.0.

Cat.	Device	ISA	#	Operating System	Wireless	Peripherals/Characteristics
IoT	TelosB	`MSP`	30	Contiki/RIOT	Zigbee	Temperature, Humidity etc.
	Arduino Mega 2560	`AVR`	26	Bare-metal/RIOT	WiFi/BLE	Temperature, Humidity, SD Card, etc.
	Arduino Uno	`AVR`	16	Bare-metal/RIOT	LoRa	LoRa Shield
	ESP32-DevKitC	`Xtensa`	180	Zephyr/RIOT/etc.	WiFi/BLE	LED
	nRF52840	`ARM32`	10	Zephyr/RIOT/etc.	Zigbee/BLE/Thread	LED
	STM32 F103C8	`ARM32`	54	FreeRTOS/RIOT/etc.	LTE	Temperature, Humidity, Light, etc.
	AliOS Things DevKit	`ARM32`	8	AliOS Things	WiFi/BLE	9-axis IMU, pressure, Mic., etc.
	HaaS100	`ARM32`	29	AliOS Things	WiFi/BLE/Ethernet	SD Card, LED
	COTS IoT devices	`/`	11	Philips/Xiaomi/Tuya	WiFi/BLE/Zigbee	Water/Temp./PIR sensor, Plug, Bulb, etc
Edge	Raspberry Pi 4B	`ARM64`	47	Raspbian Buster (Linux)	WiFi/BLE/LoRa	with 8GB RAM
	NVIDIA Xavier AGX	`ARM64`	3	Ubuntu 18.04 (Linux)	Ethernet	with AI accelerator
	NVIDIA Xavier NX	`ARM64`	1	Ubuntu 18.04 (Linux)	Ethernet	with AI accelerator
	NVIDIA Jetson TX2	`ARM64`	1	Ubuntu 18.04 (Linux)	WiFi/Bluetooth	with AI accelerator
	NVIDIA Jetson Nano	`ARM64`	8	Ubuntu 18.04 (Linux)	Ethernet	with AI accelerator
Cloud	LinkLab 2.0 Built-in Server	`x86_64`	1	Ubuntu 20.04 (Linux)	WiFi/Ethernet	with 36-core CPU and GPU

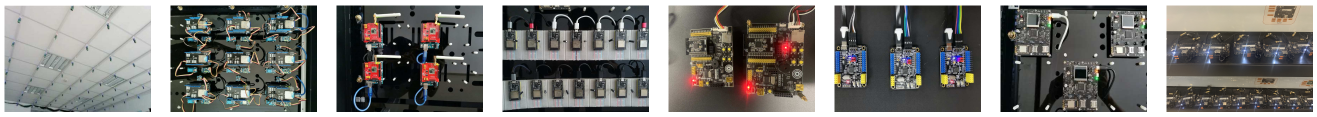

(a) TelosB (b) Arduino Mega (c) Arduino Uno (d) ESP32 (e) nRF52840 (f) STM32 F103 (g) AOS DevKit (h) HaaS100

Figure 2: IoT devices deployment in LinkLab 2.0.

could simply upload the experiment binary via our web portal. Furthermore, LinkLab 2.0 also provides a web-based IDE and online compiling services to allow users to conduct experiments anytime and anywhere. For programming the edge and cloud, LinkLab 2.0 supports both programming with serverless functions and Docker-based development.

(3) *Data Collection Configuring*: The experimental data collection of LinkLab 2.0 is based on logging "channels". Each channel represents a specific category of experimental data. Currently, LinkLab 2.0 provides three channels: console logs, network traffic and energy measurement (for some devices that are connected to a Monsoon Power Monitor).

(4) *Experiment Execution*: Once finished the provisioning, users could start the experiment by clicking the "start" button. During the experiment, users could select one or more devices to view the serial/console outputs instantly and adjust the configurations from the web portal.

(5) *Report Acquisition and Data Processing*: After the execution, users could download the experimental report from the web portal and use data processing tools such as `Python` or `R` to perform further analysis.

Comparison to other testbeds/infrastructure. We compare the development process of LinkLab 2.0 with the one using FIT IoT Lab (for IoT device deployment) and FIT Cloud Lab/Microsoft Azure (for edge/cloud development) in Figure 3 and summarize the differences as follows. (1) LinkLab 2.0 is the only testbed that includes the IoT, edge and cloud, which facilitates users to do a one-stop development. Users are not asked to login with multiple credentials and the inter-device network is automatically configured. Users of FIT IoT Lab will spend a long period configuring the border router and setting up the connectivity between IoT and the cloud. (2) Furthermore, thanks to our integrated programming support (§3.2), users could have a bird's-eye view when selecting devices and the networking parameters are shown when programming, which could shorten the development time. (3) The Web-based IDE and built-in online compilation environment enable the one-key provision of developed experiments. However, using FIT and Azure, developers could only separately write code for the IoT and cloud and manually deploy the program, which both need much coordination efforts between platforms.

3 Designs of Management Services

Managing such a multi-tiered testbed with various heterogeneous devices faces several non-trivial challenges. In this part, we will present the challenges and solutions from the basic architecture design to programming and reliability issues.

3.1 Overview of Management Architecture

As shown in Figure 4, all the IoT, edge and cloud devices are managed by a three-tier architecture named LinkLab 2.0 Device Control (LDC). LDC contains three building blocks: the controller, the server and the client. The LDC controller is the top-level management service, which is responsible for gathering the programming and controlling tasks from the users. The LDC server takes the experiment tasks as input, assigns the tasks to the devices and forward the binaries or configurations to the LDC client (for IoT devices) or directly to the programmable devices (for the edge/cloud). The LDC client is at the lowest level, which directly interacts with the IoT devices and provides device control interfaces (e.g., program, reset, and keep-alive) to the LDC server via the network.

Kubernetes-based management services. The architectural design of LinkLab 2.0 does not happen overnight. We next elaborate on the alternatives and our considerations dur-

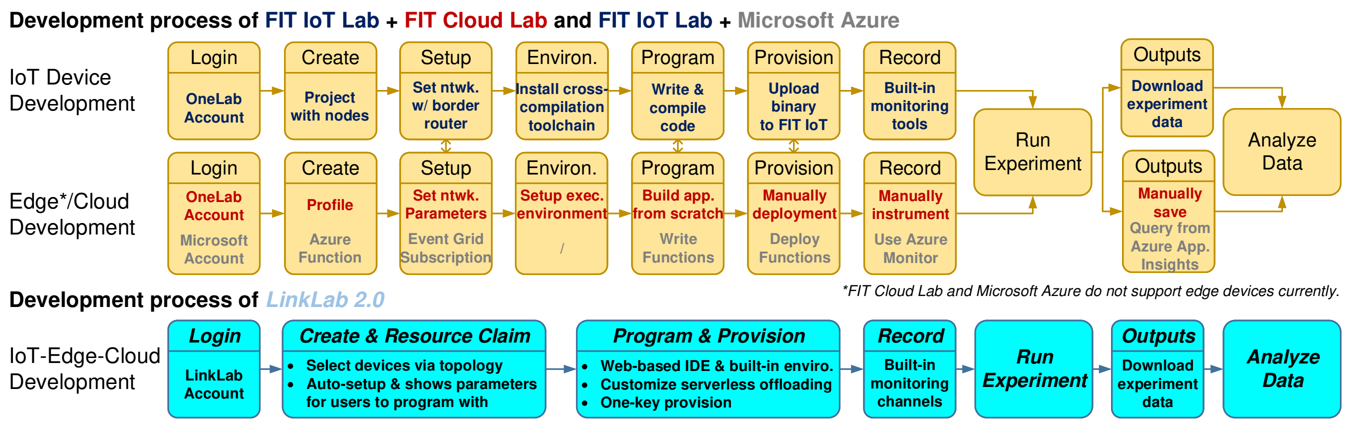


Figure 3: Development process of FIT IoT Lab (for IoT), FIT Cloud Lab or Microsoft Azure (for cloud) against LinkLab 2.0.

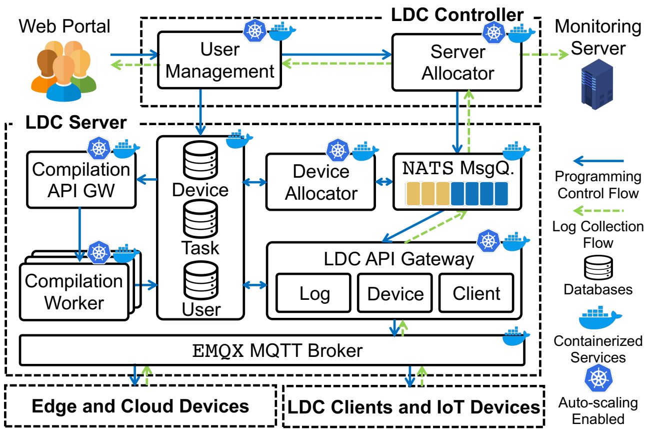


Figure 4: Kubernetes-based management services.

ing the development of such a heterogeneous and large-scale testbed.

Monolithic or cloud-native? In retrospect, LinkLab 2.0 is originally built in a native, monolithic way, which means the management functionalities are centralized in a handful of monolithic services and bare-metally deployed on the server with binaries. Nevertheless, after a few irritating experiences of migrating the services between servers or establishing sub-sites, we decided to adopt a cloud-native architecture (see Figure 4), which means decoupling functionalities to microservices and deploying them with containers. The rationale is we frequently build sub-sites to extend LinkLab 2.0's coverage and community, which makes us value the ease of service management and migration brought by cloud-native more than the extra overhead brought by containerization.

Be adaptive to highly fluctuating workloads. Since one of LinkLab 2.0's usage scenarios is online education, we observe a highly fluctuating workload during the operation of LinkLab 2.0, i.e., many concurrent requests during classes while few users are active at night. Simply over-provisioning the management services is too conservative and uneconomical, hence LinkLab 2.0 leverages Kubernetes for adaptivity.

Kubernetes [53] is an open-source system that enables the automated scaling of containerized services by instantiating service replicas. As Figure 4 shows, all the key services are containerized and managed by Kubernetes (except `NATS`, `EMQX` and databases because they have their own scaling policy). The addition of a *server allocator* makes LinkLab 2.0 scalable for building LDC servers in multiple remote sub-sites.

Benefits. This Kubernetes-based management architecture is scalable to user requests. Once the user request bursts, services shown in Figure 4 will automatically scale up to handle the requests and scale down to save the resources when there are few requests.

Tiered management of devices. Due to the different programming approaches between IoT and edge/cloud devices, LinkLab 2.0 employs tiered management of the devices.

IoT devices. The real IoT devices are connected to a Raspberry Pi (RPI), which the LDC client deployed on, via USB serial. LDC client includes a programming service based on the burning tool provided by the manufacturers of the devices (e.g., `avrdude` for Arduino-series boards). An alternative is using the Over-The-Air (OTA) technology to update the binaries, while we gave up this idea due to the scarce storage space on IoT devices and the wireless interference.

The management of virtual devices is akin to the real nodes. The only difference is the addition of creating and deleting interfaces for users to adjust the number and type of simulated devices, and the managing commands are transmitted via the network rather than USB serial.

Programmable edge and cloud. The management of the programmable edge/cloud devices differs from IoT devices in two ways: (1) LinkLab 2.0 leverages network-based controlling instead of USB serial because the transmission speed of USB serial (115.2Kbps) is generally much slower than the network (>100Mbps), while the data size for provisioning edge device is mainly in gigabyte magnitude. (2) The LDC client is directly deployed on the programmable edge/cloud devices because they have enough computing ability to handle the management commands.

Hence, LinkLab 2.0 develops a runtime for the edge/cloud (Figure 5) to support the programming, controlling and moni-

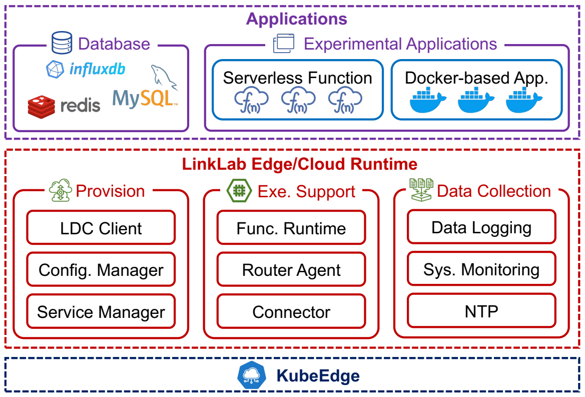


Figure 5: LinkLab 2.0 edge/cloud software stack.

toring tasks. The LDC client is part of the edge/cloud runtime and responsible for handling the commands sent by the LDC server and reports the status of the edge device. Once the LDC client receives an experiment deployment configuration, the *configuration manager* parses it and deploys the experiment. After provisioning, the lifecycle of the services of the experimental application is managed by a *service manager*, including starting, restarting and destroying the instances.

Benefits. This tiered management system as well as the distributed design of LDC client and server dramatically reduces the efforts for adding the devices to LinkLab 2.0. Users who intend to add new IoT devices only need to deploy an LDC client and plug the device into the client. Without these designs, adding devices requires plugging them into the same hardware device that the LDC server lies on, which reduces the scalability for deploying the IoT devices in the wild or physically remote from the LDC management cluster.

3.2 Achieving IoT-Edge-Cloud Integration

We now introduce how LinkLab 2.0 achieves the integrated programming for IoT, edge and cloud and how to guarantee the reliable programming of the multi-layered devices.

Integrated programming for IoT, edge and cloud. The most important feature of LinkLab 2.0 is to facilitate one-site programming for IoT devices, edge and cloud. Moreover, LinkLab 2.0 also supports serverless functions and computation offloading to lower the threshold for experiencing cutting-edge programming paradigms.

One-site programming for the three layers. In order to facilitate the one-site programming for IoT, edge and cloud, LinkLab 2.0 optimizes each step of conducting an experiment. (1) During the selection of devices, LinkLab 2.0 provides users with the hardware network topology. With this view of topology, users could choose devices from different layers with fully aware of the connectivity between devices. (2) For the programming step, users could use a Web-based IDE of LinkLab 2.0 to write code directly in the web browser and leave the compilation and deployment to LinkLab 2.0 backend services. LinkLab 2.0 also automatically handles the network between devices and shows the networking parameters (e.g., IP address) for each device during users' programming. (3) During the experiment execution, LinkLab 2.0 supports the customization of configurations, especially the parameters for inter-layer communication, which could not be easily configured in other testbeds such as FIT IoT Lab. The programmable configurations include connectivity settings (e.g., round-trip time, bandwidth, packet loss rate) and performance settings (e.g., resource quota of services, docker priority).

Customizable offloading with serverless functions. Recent advances in serverless computing [60, 61, 74] allow users to focus on the application logic other than wasting time on the configuring environment and parallelism from scratch. Hence, besides the basic Docker-based development, users of LinkLab 2.0 could decompose their experiment logic into serverless functions and deploy them on edge or cloud devices. Moreover, to further simplify the serverless programming in the IoT-edge-cloud scenario, LinkLab 2.0 provides *device-interaction APIs* for serverless functions to read data from an IoT device.

Based on the serverless functions, LinkLab 2.0 provides systematic support for function offloading. Primarily, users could use `@remotable` annotation to mark the function that could be offloaded. Moreover, a large amount of related research [36, 50] concentrates on the offloading policy. Therefore, LinkLab 2.0 presents a *customizable offloading framework* to enable users to define and test their own offloading policies. Towards this, LinkLab 2.0 first decouples the offloading policy module from the offloading handler, which is responsible to intercept the function execution and transmit it to the offloading destination. Second, LinkLab 2.0 provides customization interfaces for users to define their own policies.

Timely edge control based on vNICs. As we stated before, LinkLab 2.0 uses the network to manage the programmable edge and cloud, which is the same data channel that most experiments use. Suppose an experiment intends to occupy as much bandwidth as it can (which most experiments do), the management delay of this device will dramatically increase, or even the device will stop responding to management commands. Existing cloud testbeds could alleviate this by employing a dedicated network interface card (NIC) for management and control. Nevertheless, most of the COTS edge devices only have one NIC and are not customizable after manufacture.

Hence, we develop a software-based tool, `resGuard`, to ensure the responsiveness of the management service under any circumstances. (1) For outbound traffic, the `resGuard` first uses `cgroup` to categorize the key management tasks (processes) and other user tasks. Then, `resGuard` leverages `tc rate` to guarantee the minimum bandwidth of management tasks. (2) For inbound traffic, however, the aforementioned approach is not applicable because `tc` only implements an egress packet queue and `cgroup` could not classify the inbound traffic to its destination process. Hence, `resGuard` takes advantage of the `ifb` virtual NIC mechanism of Linux.

```
tenant:
  name: NSDI2023
  user: "alice", "bob"
  hardware_exclusive: "AMega-1", "AMega-2", "ESP32-1", "RPI-1"
  hardware_shared: "ESP32-2", "ESP32-3", "RPI-2"
  services: "$all$" # enable all services
  service_quota:
    – compiling: 10 # unit: req/s
    – burning: 5 # unit: req/s
```

Figure 6: Example configuration to create a tenant.

The `ifb` vNIC is a message queue by implementation and any packet that is redirected to `ifb` will return to its original NIC automatically after traffic shaping. Therefore, `resGuard` redirects all the inbound traffic to `ifb` and uses `tc` to prioritize the packet from the IP addresses which host the management services. Note that `resGuard` only prioritizes the management service, the experiments could only utilize all the resources if there is no management task. In addition, `resGuard` also reserves CPU quota for management tasks via `cgroup`.

3.3 Achieving Multi-tenancy

During the COVID-19 pandemic, we make LinkLab 2.0 available to teachers and students for educational purposes. However, this scenario poses new concurrency and multi-tenancy challenges for LinkLab 2.0, which is guaranteeing a dedicated and responsive usage of services and devices during a certain period of time. This is because the programming requests are expected to be handled immediately in a limited class time.

The containerized architecture of our management services (Figure 4) is a good start for achieving multi-tenancy for its scalability and isolation property. However, the programmable devices, which are one of the building blocks of LinkLab 2.0, are not included in this isolation framework. Hence, LinkLab 2.0 proposes *device-involved multi-tenancy*. With this technique, operators could easily create a new tenant for a dedicated usage, and LinkLab 2.0 assures the programming responsiveness of services and devices within a preset quota.

Device-involved multi-tenancy. Services in Figure 4 are reconstructed to support multi-tenancy. First of all, each database owns a `TenantID` field for other services to look up. To avoid the interference of the tasks from different tenants, the device allocator creates waiting queues for each tenant and launches an appropriate number of instances to handle the requests. Other services will subscript the task from `NATS` once they are assigned to a tenant.

Once a new request of a user is received, LinkLab 2.0 will first query which tenant the user belongs to and put the request to the corresponding queue. Then, the device allocator searches for if there are idle devices that meet the user request and belong to the requesting tenant. If so, the request will be assigned to the device. Furthermore, this device-involved multi-tenancy is also a lightweight approach to IoT device virtualization. When creating a new tenant, administrators could assign devices to the tenant in the “exclusive” or “shared” manner. “Exclusive” means the device could only be accessed by the users of the tenant, while “shared” means the device is shared with other tenants. There are primarily sensing and actuation IoT devices in LinkLab 2.0. If a sensing device is set to “shared”, we can multiplex its sensing data to multiple tenants if necessary. On the other hand, the actuation devices cannot be shared after being allocated to avoid conflicting operations.

We also leverage an accounting mechanism for the containerized services in Figure 4 to record the resource usage of each tenant. If a tenant exceeds its preset quota, the user management will reject the new request from this tenant.

Tenant management. We build a command-line interface (CLI) for administrators to manage tenants, such as creating a tenant or to moving a device/user to an existing tenant. As Figure 6 shows, administrators could allocate users, hardware resources, and software services with quota in a `YAML` configuration file when creating a new tenant. Then use our management CLI with `LinkLab 2.0-manage tenant -c <config>.yaml` to create (`-c`) a new tenant to LinkLab 2.0. Besides adding new tenants, LinkLab 2.0 also supports modifying (`-m`) and deleting (`-d`) tenants.

3.4 Achieving Reliability

Reliability is the principal requirement of a testbed that is used for both academic and educational purposes. To achieve 24/7 availability, LinkLab 2.0 employs an anomaly detection system. The goal of the monitoring system is to keep two groups of reliability problems away from LinkLab 2.0: (1) Device-related problems, such as unexpected offline and abnormal sensor readings; (2) Service-related problems, such as improper resource occupation and service breakdown.

Proactive and reactive device anomaly detection. Towards the device-related problems, we use both proactive and reactive detection to catch the exceptions as soon as possible.

Reactive detecting. Basically, the devices in LinkLab 2.0 are monitored reactively, which means we only use the data that is non-intrusively collected from the IoT device. Such as we monitor the working state of the peripherals using the readings piggybacked from users’ experiment applications.

Proactive probing. To monitor the devices that are used infrequently and improve the coverage of device problem detection, LinkLab 2.0 also employs proactive probing. We create a benchmark set for each kind of IoT device that covers most of the functionalities of the device. Once a device is idled for a period of time (empirically set to 2 hours), our monitoring system programs the benchmarks to the devices and analyzes the output. Note that we make the regular user requests could preempt the execution of probing benchmarks for a better device utilization. Once abnormal behavior occurs, our monitoring system will send an alarm to the operators.

Benefits. Besides alarming the maintainers of device failures, our device anomaly detection approach is also beneficial to the reproducibility of experiments conducted on LinkLab 2.0. To ensure the correctness and accuracy of the experiments, we pay attention to the result consistency and abrasion of our devices by correlating the piggybacked sensor data in the reactive detection with the environmental data collected by the maintainers and the outputs of proactive probing with our pre-defined ground truth.

Service anomaly detection via multi-model log fusion. In this part, we introduce how LinkLab 2.0 detects service abnormalities through log analysis during system operation.

Multi-model system runtime data collection. The runtime log is an implicit indicator of the system's health. LinkLab 2.0 collects both structured and semi-structured runtime data for further anomaly detection.

(1) Structured key performance indicators (KPIs). KPIs are the quantitative metrics that reflex the operating status of the system. For each layer of LinkLab 2.0 (IoT, edge and cloud), we collect different KPIs according to the intrinsic difference of each layer. (a) For the edge/cloud layer, we collect the network throughput, CPU, memory, etc., as the KPIs. (b) For devices of the IoT layer, the connectivity of each device (indicated by its heartbeat message to the LDC client) is recorded. The KPIs are periodically collected by our monitoring system. The interval is empirically set to 2min in the current deployment.

(2) Semi-structured event logs. Each building block of LinkLab 2.0, especially the management services in Figure 4, reports the underlying behavior of the component via the semi-structured event logs. These logs are mainly service-specific outputs with timestamps and labels of severity levels such as `FATAL` and `ERROR`. Our monitoring system collects these logs from each layer of LinkLab 2.0 for further analysis.

Multi-model log fusion based anomaly detection. The collected KPIs are time-series data, which are eligible for further processing. Nevertheless, the collected semi-structured event logs also need to be structured for anomaly detection. We propose the simFDT approach, which extends the widely-used fixed depth tree (FDT) model with the ability to parse logs in variable lengths using the similarity between logs and templates, to cope with diverse logs generated by heterogeneous devices. After parsing by simFDT, we employ a sliding window to count the different events in a period of time as vectors and feed the vectors to our anomaly detection algorithm.

We employ Autoencoder (AE) to detect abnormal events of our entire system using the KPIs and the vectors parsed by simFDT. AE is widely used for anomaly detection on time-series data [13, 42, 55]. In order to reduce the overhead of transmitting monitoring data, we leverage multi-level detection by separately training anomaly detection AEs for each layer and deploying the models on the nearest upper layer of the devices/services being monitored. For example, the AE for IoT devices is deployed in the LDC client cluster and the AE for the LDC client cluster is on LDC servers.

Operational findings of anomaly detection. We have implemented and deployed the aforementioned anomaly detection approaches to our production environment to achieve 24/7 reliability. Up to now, LinkLab 2.0 achieves 98.2% hardware available time across the IoT, edge and cloud layer since the introduction of this system, and Table 3 shows the detected anomalies. We can see from the table that our anomaly detection approaches could recognize the abnormal states for services, devices and the entire system.

Table 3: Detected anomalies during the operation of LinkLab 2.0 (sorted by the occurrence in descending order). Dev. and Svc. mean the device- and service-related problems, respectively.

#	Type	Root causes of detected anomalies
1	Svc.	Docker images of services are recycled by Kubernetes.
2	Svc.	Failed deployment of a newly developed feature.
3	Dev.	The power supply of an edge/IoT device is broken.
4	Dev.	Loose connection between devices and peripherals.
5	Svc.	Disconnection of the reverse proxy to the cluster.
6	Dev.	The peripheral/pinout of an IoT device is broken.
7	Svc.	Network fluctuations for hours or days.
8	Svc.	Deletion of key files caused by maintainer's misoperation.
9	Svc.	Power outage in the equipment room of LinkLab 2.0

Nevertheless, we observe false positives (FPs) and true negatives (TNs) during the long-term deployment. Here, we will elaborate on the occurrence of FPs and TNs, and discuss how LinkLab 2.0 evolves to cope with the problems.

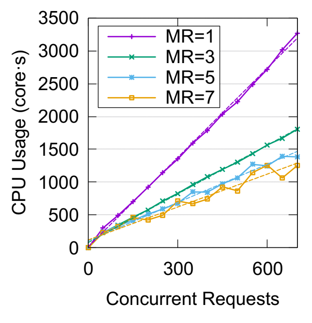
The observed FPs include: (1) The number of testbed usages increases sharply, which leads to the false warning due to unusual high resource usage. (2) The regular operational actions by the maintenance team run out of the resources (i.e., deploying a new version, too many concurrent proactive tests for anomaly detection). (3) The false warning on unprecedented low resource usage after a server upgrade. The rationale behind these FPs is the data sources of the anomaly detection are inadequate. Hence, we subsequently improve the detection system by correlating the detection results with the operational data (e.g., active user, deployment status) and the domain knowledge of operators (e.g., server upgrade).

The TNs are mainly short-time service unavailability (usually for a few to tens of seconds) incurred by the network fluctuation. These TNs could be easily addressed by shortening the interval of KPI collection and proactive probing. We argue that the selection of the interval in the current deployment exhibits a good tradeoff between detecting accuracy and overhead, and we will adjust the interval for the time-sensitive situations such as supporting live classes and exams.

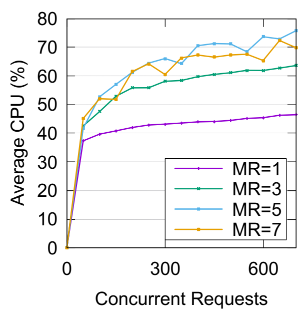
4 System Performance

We test our system to answer the following questions: (1) Is LinkLab 2.0 scalable? (2) Whether the `resGuard` of LinkLab 2.0 is reliable for programming edge/cloud devices and what is the performance? (3) How does the multi-tenant design of LinkLab 2.0 perform?

Effectiveness of the scalable architecture. In this part, we evaluate the scalability of LinkLab 2.0's architectural design by simultaneously issuing hundreds of concurrent requests and see how many resources that used by LinkLab 2.0.



(a) CPU usage.



(b) Average CPU utilization.

Figure 7: Resource usage of management services for different concurrencies. MR is the maximum number of service replicas (i.e., the number of containers instantiated to serve the requests of a service) when auto-scaling.

Figure 7(a) illustrates the usage of the CPU. Note that we use the product of CPU utilization and time as the metric of CPU usage in order to illustrate the overall usage during a period of time rather than the instant usage, and we omitted the illustration of memory usage because it shares almost the same distribution to the CPU and page limit. We can observe in Figure 7(a) that the relationship between CPU usage and the concurrent request is *linear* across all the settings, which is scalable because the per-user resource is stable even under extreme concurrencies. Another observation is that the CPU usage seems to be decreasing when the maximum number of replicas (MR) increases. To investigate more on the reason for the decrease, we go into the average CPU utilization of each setting, as Figure 7(b) shows. We can see that the CPU is under-utilized with lower MR, which may be because the tasks have to wait longer when there is no worker replica to process them. Nevertheless, the benefit of auto-scaling also has its upper bound. In our setting, the upper bound is 5 replicas, which we can see in both Figure 7(a) and 7(b), because the management services of LinkLab 2.0 are deployed on a cloud cluster with *five* nodes (each has a 2.5GHz CPU core).

Effectiveness of `resGuard`. In this part, we evaluate the programming reliability of the IoT devices and edge/cloud devices with our three-tiered management system.

For the IoT devices, there are only 504 (1.9%) failed trials in 27,216 programming tasks from May 2020 to January 2021. For edge/cloud devices, we conduct an experiment to evaluate the effectiveness of the `resGuard` we introduced in §3.2. The evaluation methodology is that we attempt to deploy a new experiment on the edge device while another experiment is still running on that device, and we adjust the bandwidth occupation of the existing experiment to see how the deployment time varies under different situations. We use two jobs as new deployment tasks: (1) the user deploys an `InfluxDB` and views the logs, and (2) the user deploys a `Kafka` logging service to the edge device. Figure 8 shows the results. We can see that the deployment time increases rapidly without `resGuard` while the deployment times stay still with `resGuard` when the existing experiment takes up more than

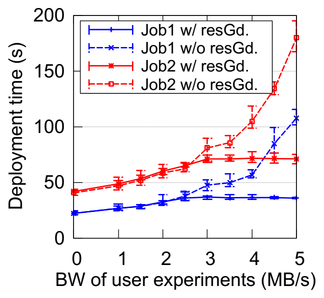


Figure 8: Provisioning time of new experiments when another experiment is running on the edge device with and without `resGuard`.

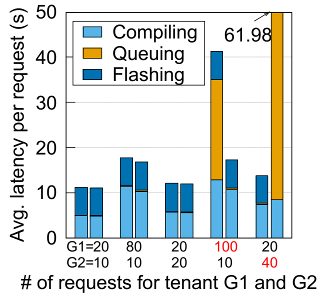


Figure 9: Average programming latency different concurrent request for tenant G1 (80 devices maximum) and G2 (20 devices maximum).

3MB/s bandwidth.

Effectiveness of multi-tenancy. To evaluate the effectiveness of multi-tenancy, we use 100 devices of LinkLab 2.0 and separate them into two tenants: G1 and G2. Then we emit concurrent requests on behalf of the two tenants in different distributions and record the latency per request of each stage in LinkLab 2.0. We set the device quota as: G1 has 80 devices and G2 has 20 devices. Figure 9 shows the results. We can observe that once the request number exceeds the device quota of the tenant (e.g., G1=100>80, G2=10<20), its per request latency increases greatly while the latency of the other tenant is not affected. Moreover, the increased time is mainly spent on waiting for allocation, which is a piece of direct evidence that the tenants would not preempt the resources of others.

5 Representative Use Cases

LinkLab 2.0 facilitates various new usages in addition to the basic wireless and embedded experiments supported by existing IoT testbeds [3, 7, 28, 57]. Furthermore, during the evolution of LinkLab 2.0, we extend it from an academic testbed to an innovative learning and examining platform for schools and individuals. We summarize the representative use cases of LinkLab 2.0 in Table 4 and Table 6 and will elaborate on both the research usages and outreaches in the rest of this section, respectively.

5.1 Supported Research Experiments

Potential research domains. We summarize the potential research domains that LinkLab 2.0 could support in Table 4. Besides traditional wireless and embedded experiments, researchers could conduct experiments with respect to serverless computing, edge AI and other research topics easily and holistically with the integrated architecture, heterogeneous devices and various deployment methods of LinkLab 2.0. Furthermore, with the involvement of edge and cloud, researchers could extend the networking protocol experiments to the IoT domain and obtain data from both the server and client.

While implementing all the research above is beyond the scope of this paper, we internally developed three represen-

Table 4: Research usages of LinkLab 2.0. F1-F4 are LinkLab 2.0's features: F1: distributed and scalable architecture (§3.1), F2: serverless/docker-based development (§3.2, S for serverless, D for docker), F3: offloading (§3.2), F4: multi-tenancy (§3.3).

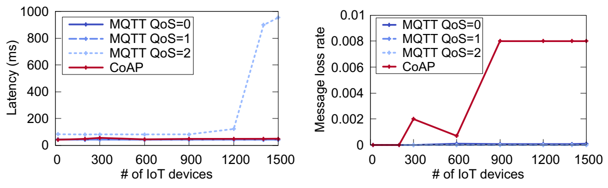
Category	LinkLab 2.0's representative use cases	Scale	LinkLab 2.0's components/features used in researches						
			Cloud	Edge	IoT	F1	F2	F3	F4
Potential Research Domains	Cloud-Edge-IoT integrated application [21,31,73,76]	2+ devices	●	●	●		S, D	◗	
	Wireless and embedded experiments [39,64,77]	2+ devices	◗	●	●		D	◗	
	Offloading algorithms [23–25,40,43]	2+ devices	◗	◗	●		S	◗	
	FaaS and serverless computing [2,5,6,14,17,60,61]	2+ devices	◗	◗	●		S	●	
	Container-based service composition [35,62,63,75]	1+ devices	◗	◗	●		D	◗	
	Edge AI [29,38,45,54,71,72]	1+ devices	◗	◗	●	*N/A*	N	◗	*N/A*
	IoT networking protocols [15,37,59]	3+ devices	◗	◗	●		D	❍	
	Industrial Internet of Things [11,32,44,65,69]	3+ devices	◗	◗	●		D	◗	
Example Researches	dSpace: Composable abstractions for smart spaces [21]	5~10 devices	●	●	●		D	❍	
	HRank: AI model pruning for edge computing [45]	3+ devices	❍	●	●		S, D	❍	
	Measurement of different IoT messaging protocol [59]	3+ devices	●	❍	●		D	❍	

[1] ●=all of the cases use the component/feature, ◗=many but not all of the cases use the component/feature, ❍=the component/feature is not used.
[2] N/A means feature F1 and F4 is not applicable for individual usages.

Table 5: Collected data in Edge AI experiment with LinkLab.

Device	Action	Time (s)	Size	Acc.[¶]
NVIDIA Xavier AGX (w/ accelerator)	Rank generation	590.7	Before: 115MB ⇓ After: 15MB	92.9% (93.9%)
	Model pruning	4,213.5		
	Inference	15.2		
Raspberry Pi (no accelerator)	Rank generation	180.1		93.9%[‡] (93.9%)
	Model pruning	–[†]		
	Inference	385.4[‡]		

[¶] Values in the parentheses are the results presented by the authors of [45].
[†] Model pruning on Raspberry Pi (RPI) is too long to finish.
[‡] Measured using HRank authors' model since pruning on RPI is too long.



(a) End-to-end latency. (b) End-to-end message loss rate.

Figure 10: Collected performance of IoT protocols.

tative experiments that could evaluate the functions and lead the further usage of LinkLab 2.0. Code and tutorials of these experiments are publicly available[1].

Experiment 1: Cloud-Edge-IoT integrated application. We use a programming framework for smart spaces named dSpace [21] as an example to show the potential of LinkLab 2.0 for deploying cloud-edge-IoT integrated research.

Implementation. We directly deploy the open-source dSpace runtime [20] on the cloud. A home automation framework (Home Assistant in our implementation) is deployed on the edge to manage the IoT devices locally and provide device-controlling APIs for the dSpace runtime.

Findings. (1) The original version of dSpace only takes the COTS IoT devices into consideration. With the help of LinkLab 2.0, could explore a larger design space by obtaining more detailed data on network traffics and energy consumption by deploying instrumented codes on the prototyping devices of LinkLab 2.0. (2) The integrated architecture of cloud-edge-IoT and the docker-/serverless-based programming approach of LinkLab 2.0 facilitate users to explore the "sweet spot", and this dSpace case is an ideal example. To be more specific, users could easily move the Home Assistant module between edge and cloud to evaluate the tradeoff between deploying the Home Assistant module on the edge (unreliable edge-cloud connection but shorter local control latency) and cloud (responsive device-controlling APIs but higher device control latency).

Experiment 2: Edge AI. Recently, Edge AI is recognized as one of the emerging technologies by Gartner [22] since it could provide lower latency and better privacy for users. In order to illustrate how LinkLab 2.0 facilitates Edge AI experiments, we conducted an example experiment based on HRank [45] which prunes the AI model to make it feasible to be executed on the resource-constrained edge devices.

Implementation. For model pruning, HRank first generates the rank of each layer of the model by performing the training process with a very small portion of the dataset. Then, HRank prunes the less important filter by retraining the original model with the generated rank. With LinkLab 2.0, users could simply apply for an edge device, deploy the rank generation and pruning algorithm, and evaluate the accuracy degradation of inference with the serverless functions or native dockers. Table 5 shows the comparison of the execution time, model size and inference accuracy of HRank on heterogeneous edge devices, namely Xavier AGX (AGX for short) and Raspberry Pi (RPI for short).

Findings. (1) During our reproduction of HRank, we encountered an abnormal result. It is known that AGX's CPU is more powerful than RPI's, and AGX is also featured with a GPU-like accelerator. Nevertheless, the rank generation time on AGX is much longer than on RPI (see Table 5), which is unusual. After our investigation, we finally found the reason as follows. The HRank code leverages a GPU-based matrix algebra library named `MAGMA` to accelerate the rank generation, but it does not have an ARM distribution for AGX. Hence, on the AGX, HRank must move the intermediate variables during training from GPU to CPU to calculate rank and move back to the GPU, which leads to massive performance degradation. We attribute this interesting finding to LinkLab 2.0's various and heterogeneous edge devices that could be used to obtain a

[1] https://linklab.emnets.cn/tutorials

Table 6: Outreaches of LinkLab 2.0. Similar to Table 4, F1~F4 represent LinkLab's features and S/D represent serverless/Docker.

Category	Outreaches of LinkLab 2.0	Scale	LinkLab 2.0's components/features used in cases						
			Cloud	Edge	IoT	F1	F2	F3	F4
Educational Institutions	E1: IoT curriculum of undergraduates in College A	~40 users/yr.	●	●	●	❍	D	❍	●
	E2: IoT fieldwork courses for students in College B	~90 users/yr.	●	❍	●	❍	S, D	◗	❍
	E3: Joint construction of an IoT Laboratory with University C	~400 users	●	●	●	●	S, D	◗	●
Commercial Cooperations	C1: Online IoT device playground of Merchant D	~150 users	❍	❍	●	●	D	❍	●
	C2: IoT Engineer Certification Exams of Merchant D	~500 users	●	●	●	❍	D	❍	●
Third-party Individuals	T1: Geek developers	1000+ users	●	●	●	❍	S, D	◗	❍
	T2: Self-learners of rudimentary IoT developments		●	❍	●	❍	S, D	❍	❍

●=all of the cases use the component/feature, ◗=many but not all of the cases use the component/feature, ❍=the component/feature is not used in the case.

more thorough view of the performance of the algorithms. (2) In addition to the above experiment, LinkLab 2.0 also allows users to build edge AI applications with the real-world sensing data which could be acquired from our IoT devices and test the performance brought by unstable wireless networks, which are currently not supported by other testbeds.

Experiment 3: Measurement of IoT protocols. Different from the dominance of HTTP for web applications, there is no unique protocol that could serve all the diverse application scenarios of IoT. Hence, conducting measurement studies on IoT protocols under different scenarios is worthwhile, especially before the actual deployment of IoT applications.

Implementation. As [59] illustrates, we compare the two most-used IoT protocols, MQTT and CoAP, using LinkLab 2.0. To make the measurement closer to the real-world application, we leverage the widely-used `EMQX` message broker on the cloud for both CoAP and MQTT. With respect to the clients, we develop the client based on `libcoap` and `paho`. Figure 10 shows the measured end-to-end latency (device-edge-cloud-device) and message loss rate against the increasing number of devices.

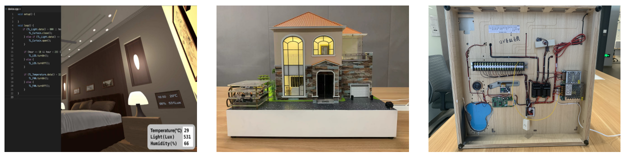
Findings. (1) By virtue of the LinkLab 2.0's theoretically unlimited number of virtual IoT devices, users could easily simulate large-scale, real-world IoT applications to evaluate the performance of edge cases. (2) With the help of LinkLab 2.0's bandwidth management, users could measure the performance of protocols under different network conditions.

5.2 Outreaches

We now report the three types of external users for non-academical usages as shown in Table 6.

Educational institutions. The most valued and long-acting outreach of LinkLab 2.0 is supporting the IoT programming practices of the relevant courses in schools, especially after the outbreak of the COVID-19 pandemic.

As shown in Table 4, the usage of LinkLab 2.0 covers regular IoT curriculum (E1) to fieldwork courses (E2). A and B are both technical colleges that teach students both theoretical expertise and practical skills, and students of these colleges mainly choose to serve the local economy for non-academic jobs after their graduation. To cope with this teaching objective, LinkLab 2.0 decides to establish a series of experiments to better train students from various knowledge and socioeconomic backgrounds ready for job markets.



(a) Online virtualization (b) Tabletop model (c) Tabletop schematics

Figure 11: Online-offline integrated smart elderly care education toolkit for College B.

We use our cooperation with college B, which is proficient in serving the local smart elderly care industry, to exemplify how LinkLab 2.0 works with the institutions and builds curricula for their students. To initialize the cooperation, we set up seminars with teachers in college B to introduce the functionalities of LinkLab 2.0 and co-create a basic version of the experiment series that fulfills the knowledge points of the courses. Then we support students for one semester's usage, collect operational data for each experiment (e.g., average completion time, retry counts) and discuss again with the teachers to adjust the difficulty of each experiment or extend the experiments to cope with the changing demands of the job market. The above process is performed recursively in the last three academic years. Up to now, the outcome of our collaboration with College B includes (1) a set of lab experiments for the technical school students who major in IoT, and (2) an online/offline integrated education toolkit for smart elderly care (see Figure 11). The toolkit includes an offline tabletop model containing necessary sensors and actuators for students to realize their ideas on smart elderly care, as well as an online simulated environment to remotely test their programs in a visualized way with the identical program to the offline realization.

Furthermore, thanks to the cloud-native architecture, LinkLab 2.0 also jointly built an IoT laboratory (E3) with University C in a short time, which contains a cabinet of devices with LinkLab 2.0's containerized management software preinstalled. This laboratory is used for the experiments, exams and research of all the IoT-related curricula in University C.

Commercial cooperation. LinkLab 2.0 also draws the attention from commercial institutions. LinkLab 2.0 facilitates an online IoT device playground named HaaS Lab (C1) for Merchant D, which releases a new IoT development board named HaaS (Hardware-as-a-Service). HaaS Lab allows users to try the features of the HaaS board online before buying it

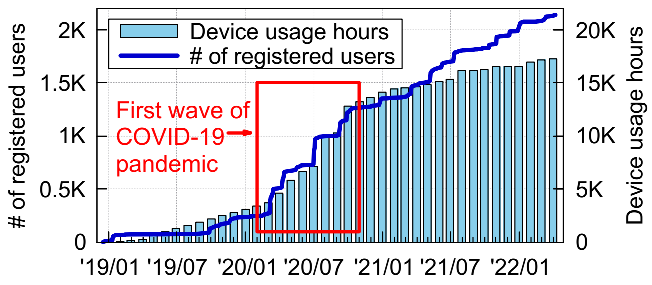


Figure 12: Illustration of the number of registered users and accumulated working hours of devices of LinkLab 2.0.

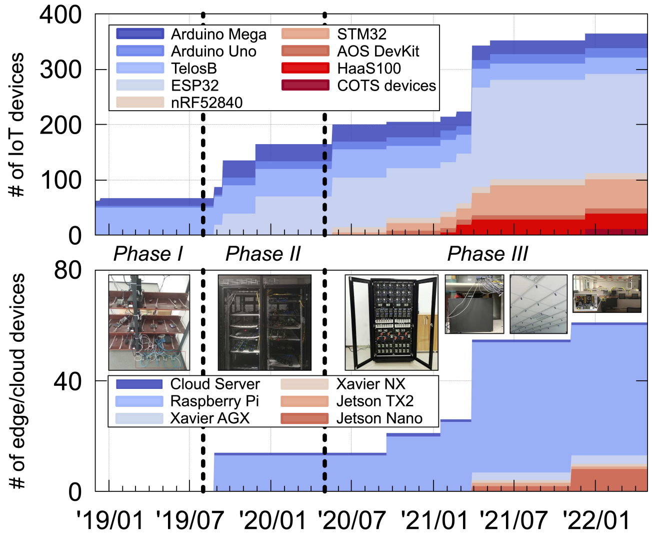


Figure 13: Evolution of LinkLab 2.0.

and Merchant D initiates HaaS Lab to promote its sale. Moreover, LinkLab 2.0 also supports an IoT engineer certification program (C2), which contains a qualifying exam and a lab exam, by creating a dedicated examination tenant.

Third-party individuals. There are 1,000+ *third-party individuals* that use LinkLab 2.0. According to the questionnaire we collected when users are registered, geek developers and self-learners take a large proportion of individual users. The geek developers use our testbed to prototype their new ideas without buying the actual hardware (T1), and the self-learners study rudimentary IoT developments (T2) with LinkLab 2.0.

6 Evolution and Lessons Learned

We are excited to see a broad use of LinkLab 2.0 in diverse scenarios since December 2018. Figure 12 illustrates the number of registered users and the device usage hours of LinkLab 2.0. We can see from the figure that LinkLab 2.0 has over 2,100 registered users and they have conducted 17,300 hours of experiments in total. We also noticed that during the first wave of the COVID-19 pandemic in the first half of 2020, both the new user registration and experiment hours of LinkLab 2.0 increased rapidly. Before January 2020, the concept of online education of IoT is not widely recognized and most of the experiments are occasionally conducted for research purposes.

We have progressively extended the functionality of LinkLab 2.0 since its first public release. According to the different focuses of the LinkLab 2.0's development, we divide the four-year operation into three phases, as shown in Figure 13.

Phase I (2018/12~2019/8): *IoT device testbed.* In this phase, we focus on the basic building blocks of LinkLab 2.0 as an IoT device testbed, such as device management, online compiling, and the web-based programming interface. Because of the diversity of the IoT hardware used in IoT applications, we investigated popular online IoT forums and academic projects, then selected five far-reaching development boards that cover the mainstream IoT ISAs (i.e., `ARM`, `AVR`, `MSP` and `Xtensa`).

Phase II (2019/8~2020/5): *Integration with cloud and edge.* Based on our experience in IoT application development and the requests from LinkLab 2.0's user community, we realized that the cloud and edge were indispensable for a complete IoT application. Therefore, we started to bring cloud and edge devices to LinkLab 2.0 and refactored the management software to cope with this architectural change. In this phase, we continue to increase the number of IoT devices. To better cope with the community's demand and avoid the waste of investments, we leverage an expenditure utilization metric with the device usage per day and the cost of a device to assess how to allocate the purchase budget of hardware.

Phase III (2020/5~now): *Cloud-native and multi-tenancy.* Advancing to the third phase, we extended LinkLab 2.0 with Kubernetes-based scalability and multi-tenancy. Moreover, we continued to enrich the list of supported IoT devices in this phase, such as the nRF52840 and COTS IoT devices.

During this four-year evolution, we have learned a lot while developing and operating LinkLab 2.0 on a broad scale. Hence, we report five lessons learned as follows.

Lessons Learned ①: **It is not easy to support heterogeneous devices for online experimentation.** Before LinkLab 2.0, we experienced a smooth development process using a sensor network testbed with about 100 TelosB motes. However, it is not a plain sailing when we try to bring more heterogeneous IoT devices to LinkLab 2.0.

For example, ESP32 boards can not automatically enter the programming mode because the flashing signal issued by the uploading software is too short. We finally came up with the idea of adding a 10μF electrolytic capacitor to the `EN` port of ESP32 to lengthen the flashing signal, which works properly. Unfortunately, we failed to attach some boards that need to manually switch to the programming mode or restart the device by pushing an on-device button (e.g., TI CC2650 DK and HiSilicon Hi3861). This manual intervention leads to the human-in-the-loop problem and does not conform to the design principle of LinkLab 2.0.

According to our operational experience, we summarize *a checklist on what device could be attached*, which is: the devices (1) do not need a manual restart, (2) owns the interface to get operational logs, and (3) support compilation and upload via a command-line interface. We believe this checklist is favorable for the operators of existing or blueprinting testbeds that use IoT devices, and the researchers that intend to conduct large-scale experiments automatically.

Lessons Learned ②: **It is difficult but important to support CI/CD (continuous integration/deployment) as well as online monitoring for providing 24/7 continuous services**. We planned to adopt the CI/CD process at the beginning of developing LinkLab 2.0. However, building a proper testing environment for CI/CD is challenging for LinkLab 2.0. Specifically, a complete testing environment of LinkLab 2.0 includes various devices and cloud servers with the proper software. It is not easy to build such an environment for each developer, especially those who worked remotely in a different city during the pandemic. Therefore, CI/CD was not supported in LinkLab 2.0 during Phase I. However, after experiencing problems like software incompatibilities and devices offline, we added CI/CD support to LinkLab 2.0 in Phase II.

Specifically, we set up a testing environment with dedicated cloud servers and a representative subset of devices for CI/CD. Note that compared with the cloud servers, the IoT/edge devices are much more problematic. Therefore, using a subset of operating devices instead of using dedicated testing devices can significantly increase the similarity between the testing environment and the operating environment. To further avoid the interference of automatic testing to the operating devices, a device will only be selected to act as a testing device without any adjacent tasks.

Even with CI/CD, online monitoring is also important for quality assurance. Device malfunctioning, wireless connection instabilities, and bursty usages are inevitable for an IoT testbed. Hence, LinkLab 2.0 performs online monitoring for all its software components and devices constantly. In case of any problems, alerts will be sent to maintainers automatically.

Lessons Learned ③: **Cloud-native is not optional, but necessary.** As shown in Figure 12, the number of LinkLab 2.0's users surges during the COVID-19 pandemic. Without the containerized architecture, resource management became painful due to the busty usages (e.g., concurrent experiments of a class of students) and various QoE requirements (e.g., time constraints for online exams). Table 4 gives such examples including classes (E1) and exams (C2). Therefore, during Phase III, LinkLab 2.0 became fully cloud-native by containerizing all services and using Kubernetes for on-demand auto-scaling, service provisioning, etc.

An unexpected benefit of becoming cloud-native is that it is one of the necessities to support multi-tenancy. After using LinkLab 2.0 for experiments or teaching, some users started to ask for dedicated devices instead of shared ones, for performance and privacy considerations. Therefore, we extended LinkLab 2.0 to support multi-tenancy (e.g., E3 and C1 in Table 4 are two typical tenants), which was straightforward with cloud-native. With multi-tenancy, we could divide a certain proportion of services and devices into a dedicated tenant to meet various requirements of the usage scenarios.

Lessons Learned ④: **Incorporating open-source projects can not always accelerate the development process.** LinkLab 2.0 builds on top of a large body of open-source projects, such as the Kubernetes for service management and EMQX for message dispatch. However, open-source software is not always ready for out-of-the-box usage, and an active community is profoundly important. For example, we build our web-based IDE based on Eclipse Theia [19], but we have to extend it for interacting with remote IoT devices and many other features, which means massive modifications. With an active community like Theia's, some of the modifications may be inspired by the existing discussions or solved by submitting issues. Unluckily, some modifications are unprecedented to the community and should be conducted by a developer who knows this open-source project well. Nevertheless, gigantic projects like Theia are too complicated to know everything about, especially for student developers.

Lessons Learned ⑤: **Plan for obsolescence.** For a long-term project, the availability of underlying open-source software could change unexpectedly. The first example is an IoT OS named AliOS Things gives up the support of ESP32, STM32 and other boards after a version iteration due to their adjustment of commercial strategy. To make things worse, they also stopped the maintenance of the compilation toolchain for the old version, which influences LinkLab 2.0's online compilation and flashing service. To this end, we managed to rebuild the obsolete tools with public documentation and older versions, which costs much labor work. The second one is the API obsolescence in open-source projects, which occurs when we update Theia from v0.8 to v1.1. After the update, most of our WebIDE components work improperly. Afterward, we check our entire code base for external dependencies, persist the current version to avoid the discontinuance impact, and be cautious when adopting newer versions.

Hence, according to the above five lessons learned, our suggestions for future testbed stakeholders are four-fold. (1) The devices which satisfy the checklist in lessons learned 1 are easier to attach into the testbed. (2) Take fully advantages of the cloud-native micro-service software architecture to better cope with bursty requests and minimize the maintenance overhead. (3) Try to incorporate the open-source software which owns an active community and has been tested by time. (4) Always remember to leave a copy of external resources such as compiling tools to avoid the obsolescence.

7 Future Directions

In this section, we envision the potential future directions and our plans for Phase IV of LinkLab 2.0.

Firstly, we plan to attach more types of devices in LinkLab 2.0. Recently, more kinds of heterogeneous devices are used in edge computing scenario. For example, the FPGA-based edge devices are widely used [68, 70] because they exhibit excellent performance on accelerating specific tasks while keeping high energy efficiency. However, FPGA devices have limited support for multi-tenancy, which is a key feature of LinkLab 2.0. We plan to borrow the partial reconfiguration approach [33] to provide concurrent programming support and design a shim [16] for our LDC client to manage the FPGAs. Another important group of devices are the boards featured with the trusted execution environment (TEE) such as Arm Trustzone [8]. Such devices are used for secure AI model training [52], sensor data collection [48, 58], etc. Nevertheless, developing a TEE-based application is different from existing development process because it requires to develop the trusted and untrusted part of the application separately and deploy them individually. Hence, we plan to embrace the new programming pattern of TEE-based applications and attach more devices facilitated with TEE to serve a broader research community.

Secondly, we also consider improving the granularity and performance of our monitoring system as the next milestone of LinkLab 2.0. As the number of supported devices and core services grow continuously, configuring our monitoring system becomes more difficult and requires more human-on-the-loop efforts. This is mainly because the current metrics-based and log-based monitoring system is not fine-grained enough to track the micro-service invocation of each request. Recently, the micro-service observability based on eBPF (extended Berkeley Packet Filter) is widely studied both in academia and industry [10, 12, 46]. It can provide more fine-grained monitoring by leveraging user-space programmable kernel extensions. Hence, we plan to incorporate eBPF with our monitoring system to achieve more fine-grained and low-overhead anomaly detection.

8 Related Work

With the proliferation of IoT technology, applications and research of IoT grow rapidly. Several testbeds are proposed to ease IoT research and application developments.

Device-edge/cloud integrated testbeds. FIT IoT Lab testbed [1], proposed by the FIT consortium, is a large-scale wireless sensor network testbed deployed across France. Except for the full programmability of the sensor devices, FIT IoT Lab also provides researchers with energy monitoring and network sniffing infrastructure to obtain the experiment data from multiple perspectives. Another testbed that consists of experiments for both edge/cloud and IoT devices is COSMOS [56], deployed in New York. COSMOS focuses on supporting the experiments of advanced wireless technologies such as mmWave and dynamic spectrum sharing.

LinkLab 2.0 differs from the above testbeds which also include the programmability on both IoT and cloud/edge devices in the following: (1) LinkLab 2.0 is the only testbed that supports the one-site development for IoT, edge and cloud. (2) For edge/cloud experiments, existing testbeds only provide bare-metal development, while LinkLab 2.0 supports both Docker- and serverless-based development. The above two differences accelerate the experiment setup process for the users of LinkLab 2.0.

IoT and sensor network testbeds. MoteLab [67] is a wireless sensor network testbed maintained by Harvard University. MoteLab includes 30 MicaZ motes and a web interface. Indriya2 [7] is a sensor network testbed with 41 TelosBs and 17 CC2650 sensortags that enables users to upload the executable, monitor the outputs of the devices and obtain data from the database. SmartSantander [57] is a city-scale testbed containing thousands of sensors with IEEE 802.15.4 or RFID connections that deploy across Santander city. Users could both work with the sensing data and program the sensors with executables to for their experiments. GioTTO [3, 4, 66] is a campus-scale sensor testbed deployed at scale on the UCSD and CMU campuses across several buildings. With the data accessing APIs and the machine learning layer of GioTTO, users could build intelligent inference applications without writing much code.

Compared with these testbeds, LinkLab 2.0 proposes a novel multi-tiered architecture for managing and controlling IoT devices to achieve high deployment extensibility. Furthermore, LinkLab 2.0 leverages the containerized management services for elastic scaling to enable the concurrent experimentation of multiple users.

9 Concluding Remarks

We introduce LinkLab 2.0, a fully programmable testbed that facilitates the integrated experiment with IoT devices, edge devices and the cloud infrastructures. LinkLab 2.0 achieves multi-tenancy and scalability by leveraging the container-based architecture and the auto-scaling technique of Kubernetes. Moreover, our multi-level monitoring system ensures the 24/7 availability for both the hardware infrastructure and software services. Since its public availability in December 2018, LinkLab 2.0 has supported over 17,000 submissions of experiments from 2,100+ users, and the accumulated usage time across all the devices exceeds 17,000 hours. From the four-year operational experience of LinkLab 2.0 for academia and education, we summarize five key observations and lessons learned concerning the device support, CI/CD process and open-source projects adoption, which we believe is favorable for other projects and testbeds requiring the integration of cloud, edge and IoT.

Acknowledgement

We would like to express our gratitude to the reviewers and our shepherd, Anirudh Badam, for their invaluable advice to improve this work. We also thank all members in the EmNets group for their contributions to this work. This work is supported by National Science Foundation of China under grant No. 62072396 and 62272407, Zhejiang Provincial Natural Science Foundation for Distinguished Young Scholars under grant No. LR19F020001, and the Fundamental Research Funds for the Central Universities (No. 226-2022-00087). Yi Gao is the corresponding author.

References

[1] Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, et al. FIT IoT-LAB: A large scale open experimental iot testbed. In *Proc. of IEEE WF-IoT*, 2015.

[2] Alexandru Agache, Marc Brooker, Andreea Florescu, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight virtualization for serverless applications. In *Proc. of USENIX NSDI*, 2020.

[3] Yuvraj Agarwal and Anind K Dey. Toward building a safe, secure, and easy-to-use internet of things infrastructure. *IEEE Computer*, 49(4):88–91, 2016.

[4] Yuvraj Agarwal, Rajesh Gupta, Daisuke Komaki, and Thomas Weng. BuildingDepot: an extensible and distributed architecture for building data storage, access and sharing. In *Proc. of ACM BuildSys*, 2012.

[5] Nabeel Akhtar, Ali Raza, Vatche Ishakian, and Ibrahim Matta. Cose: Configuring serverless functions using statistical learning. In *Proc. of IEEE INFOCOM*, 2020.

[6] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. SAND: Towards high-performance serverless computing. In *Proc. of USENIX ATC)*, 2018.

[7] Paramasiven Appavoo, Ebram Kamal William, Mun Choon Chan, and Mobashir Mohammad. Indriya2: A heterogeneous wireless sensor network (wsn) testbed. In *Proc. of International Conference on Testbeds and Research Infrastructures*, 2018.

[8] Arm. Trustzone for cortex-a. https://www.arm.com/technologies/trustzone-for-cortex-a, 2022.

[9] Mohsen Azimi, Armin Dadras Eslamlou, and Gokhan Pekcan. Data-driven structural health monitoring and damage detection through deep learning: State-of-the-art review. *Sensors*, 20(10):2778, 2020.

[10] Ido Ben-Yair, Pavel Rogovoy, and Nezer Zaidenberg. AI & eBPF based performance anomaly detection system. In *Proc. of ACM ICSS*, 2019.

[11] Jiangfeng Cheng, He Zhang, Fei Tao, and Chia-Feng Juang. DT-II: Digital twin enhanced Industrial Internet reference framework towards smart manufacturing. *Robotics and Computer-Integrated Manufacturing*, 62:101881, 2020.

[12] Cilium. ebpf-based networking, observability, security. https://cilium.io, 2022.

[13] Andrew Cook, Göksel Mısırlı, and Zhong Fan. Anomaly detection for iot time-series data: A survey. *IEEE Internet of Things Journal*, 2019.

[14] Pubali Datta, Prabuddha Kumar, Tristan Morris, Michael Grace, Amir Rahmati, and Adam Bates. Valve: Securing function workflows on serverless computing platforms. In *Proc. of ACM WWW*, 2020.

[15] Jasenka Dizdarević and Admela Jukan. Experimental Benchmarking of HTTP/QUIC Protocol in IoT Cloud/Edge Continuum. In *Proc. of IEEE ICC*. IEEE, 2021.

[16] Dong Du, Qingyuan Liu, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. Serverless computing on heterogeneous computers. In *Proceedings of ACM ASPLOS*, 2022.

[17] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting. In *Proc. of ACM ASPLOS*, 2020.

[18] 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. The design and operation of CloudLab. In *Proc. of the USENIX ATC*, 2019.

[19] Eclipse Foundation. Theia: An open, flexible and extensible cloud and desktop ide platform. https://theia-ide.org/, 2022.

[20] Silvery Fu. Github repository of dspace. https://github.com/digi-project/dspace, 2022.

[21] Silvery Fu and Sylvia Ratnasamy. dSpace: Composable Abstractions for Smart Spaces. In *Proc. of ACM SOSP*, 2021.

[22] Gartner. Trends emerge in the gartner hype cycle for emerging technologies 2018, 2018.

[23] Petko Georgiev, Nicholas D Lane, Kiran K Rachuri, and Cecilia Mascolo. Leo: Scheduling sensor inference algorithms across heterogeneous mobile processors and network resources. In *Proc. of ACM MobiCom*, 2016.

[24] Gaoyang Guan, Wei Dong, Jiadong Zhang, Yi Gao, Tao Gu, and Jiajun Bu. Queec: Qoe-aware edge computing for complex iot event processing under dynamic workloads. In *Proc. of ACM TURC*, 2019.

[25] Gaoyang Guan, Borui Li, Yi Gao, Yuxuan Zhang, Jiajun Bu, and Wei Dong. Tinylink 2.0: integrating device, cloud, and client development for iot applications. In *Proc. of ACM MobiCom*, 2020.

[26] Dolvara Gunatilaka and Chenyang Lu. REACT: an agile control plane for industrial wireless sensor-actuator networks. In *Proc. of IEEE/ACM IoTDI*, 2020.

[27] Adam Hall and Umakishore Ramachandran. An execution model for serverless functions at the edge. In *Proc. of ACM/IEEE IoTDI*, 2019.

[28] Mehrdad Hessar, Ali Najafi, Vikram Iyer, and Shyamnath Gollakota. TinySDR: Low-Power SDR Platform for Over-the-Air Programmable IoT Testbeds. In *Proc. of USENIX NSDI*, 2020.

[29] Pan Hu, Junha Im, Zain Asgar, and Sachin Katti. Starfish: resilient image compression for aiot cameras. In *Proc. of ACM SenSys*, 2020.

[30] Kang Huang, Wanchun Liu, Yonghui Li, Branka Vucetic, and Andrey V. Savkin. Optimal downlink-uplink scheduling of wireless networked control for industrial iot. *IEEE Internet Things Journal*, 7(3):1756–1772, 2020.

[31] Yutao Huang, Feng Wang, Fangxin Wang, and Jiangchuan Liu. Deepar: A hybrid device-edge-cloud execution framework for mobile deep learning applications. In *Proc. of IEEE INFOCOM Workshops*, 2019.

[32] Hongwen Hui, Chengcheng Zhou, Shenggang Xu, and Fuhong Lin. A novel secure data transmission scheme in industrial internet of things. *China Communications*, 17(1):73–88, 2020.

[33] Zsolt István, Gustavo Alonso, and Ankit Singla. Providing multi-tenant services with FPGAs: Case study on a key-value store. In *Proc. of FPL*. IEEE, 2018.

[34] Akshay Ramesh Jadhav, Sai Kiran MPR, and Rajalakshmi Pachamuthu. Development of a novel IoT-enabled power-monitoring architecture with real-time data visualization for use in domestic and industrial scenarios. *IEEE Transactions on Instrumentation and Measurement*, 70:1–14, 2020.

[35] Kavita Jaiswal, Srichandan Sobhanayak, Ashok Kumar Turuk, Sahoo L Bibhudatta, Bhabendu Kumar Mohanta, and Debasish Jena. An iot-cloud based smart healthcare monitoring system using container based virtual environment in edge device. In *Proc. IEEE ICETIETR*, 2018.

[36] Young Geun Kim, Young Seo Lee, and Sung Woo Chung. Signal strength-aware adaptive offloading with local image preprocessing for energy efficient mobile devices. *IEEE Transactions on Computers*, 69(1):99–111, 2019.

[37] Puneet Kumar and Behnam Dezfouli. Implementation and analysis of QUIC for MQTT. *Computer Networks*, 150:28–45, 2019.

[38] Seulki Lee and Shahriar Nirjon. Fast and scalable in-memory deep multitask learning via neural weight virtualization. In *Proc. of ACM MobiSys*, 2020.

[39] Xinyu Lei, Guan-Hua Tu, Chi-Yu Li, Tian Xie, and Mi Zhang. SecWIR: securing smart home IoT communications via wi-fi routers with embedded intelligence. In *Proc. of ACM MobiSys*, 2020.

[40] Borui Li and Wei Dong. EdgeProg: Edge-centric Programming for IoT Applications. In *Proc. of IEEE ICDCS*, 2020.

[41] Borui Li, Wei Dong, and Gao Yi. WiProg: A WebAssembly-based Approach to Integrated IoT Programming. In *Proc. of IEEE INFOCOM*, 2021.

[42] Nanjun Li, Faliang Chang, and Chunsheng Liu. Spatial-temporal cascade autoencoder for video anomaly detection in crowded scenes. *IEEE Transactions on Multimedia*, 2020.

[43] Yongbo Li, Yurong Chen, Tian Lan, and Guru Venkataramani. MobiQoR: Pushing the envelope of mobile edge computing via quality-of-result optimization. In *Proc. of IEEE ICDCS*, 2017.

[44] Fan Liang, Wei Yu, Xing Liu, David Griffith, and Nada Golmie. Toward edge-based deep learning in industrial internet of things. *IEEE Internet of Things Journal*, 7(5):4329–4341, 2020.

[45] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proc. of the IEEE/CVF CVPR*, 2020.

[46] Chang Liu, Zhengong Cai, Bingshen Wang, Zhimin Tang, and Jiaxu Liu. A protocol-independent container network observability analysis system based on ebpf. In *Proc. of IEEE ICPADS*. IEEE, 2020.

[47] Peng Liu, Dale Willis, and Suman Banerjee. ParaDrop: Enabling lightweight multi-tenancy at the network's extreme edge. In *Proc. of ACM/IEEE SEC*, 2016.

[48] Tianyuan Liu, Avesta Hojjati, Adam Bates, and Klara Nahrstedt. Alidrone: Enabling trustworthy proof-of-alibi for commercial drone compliance. In *Proc. of IEEE ICDCS*, 2018.

[49] Ye Liu, Thiemo Voigt, Niklas Wirström, and Joel Höglund. Ecovibe: On-demand sensing for railway bridge structural health monitoring. *IEEE Internet Things Journal*, 6(1):1068–1078, 2019.

[50] Yuyi Mao, Jun Zhang, and Khaled B Letaief. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590–3605, 2016.

[51] Garrett McGrath and Paul R Brenner. Serverless computing: Design, implementation, and performance. In *Proc. of IEEE ICDCS Workshops*. IEEE, 2017.

[52] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. DarkneTZ: towards model privacy at the edge using trusted execution environments. In *Proc. of ACM MobiSys*, 2020.

[53] NGINX. Using nginx as http load balancer. https://kubernetes.io/, 2020.

[54] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proc. of ACM ASPLOS*, 2020.

[55] Francisco Lucas F Pereira, Iago Castro Chaves, João Paulo P Gomes, and Javam C Machado. Using autoencoders for anomaly detection in hard disk drives. In *Proc. of IEEE IJCNN*, 2020.

[56] Dipankar Raychaudhuri, Ivan Seskar, Gil Zussman, Thanasis Korakis, Dan Kilper, Tingjun Chen, Jakub Kolodziejski, Michael Sherman, Zoran Kostic, Xiaoxiong Gu, et al. Challenge: Cosmos: A city-scale programmable testbed for experimentation with advanced wireless. In *Proc. of ACM MobiCom*, 2020.

[57] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, et al. Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks*, 61:217–238, 2014.

[58] Carlos Segarra, Ricard Delgado-Gonzalo, and Valerio Schiavoni. MQT-TZ: Hardening IoT Brokers Using ARM TrustZone:(Practical Experience Report). In *Proc. of SRDS*. IEEE, 2020.

[59] Victor Seoane, Carlos Garcia-Rubio, Florina Almenares, and Celeste Campo. Performance evaluation of coap and mqtt with security support for iot environments. *Computer Networks*, 197:108338, 2021.

[60] Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *Proc. of USENIX ATC*, 2020.

[61] Simon Shillaker and Peter Pietzuch. Faasm: Lightweight isolation for efficient stateful serverless computing. In *Proc. of USENIX ATC*, 2020.

[62] Christopher Stelly and Vassil Roussev. Scarf: A container-based approach to cloud-scale digital forensic processing. *Digital Investigation*, 22:S39–S47, 2017.

[63] Timur Tasci, Jan Melcher, and Alexander Verl. A container-based architecture for real-time control applications. In *Proc. IEEE ICE/ITMC*, 2018.

[64] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. Packet-level signatures for smart home devices. *Signature*, 10(13):54, 2020.

[65] Junliang Wang, Chuqiao Xu, Jie Zhang, Jingsong Bao, and Ray Zhong. A collaborative architecture of the industrial internet platform for manufacturing systems. *Robotics and Computer-Integrated Manufacturing*, 61:101854, 2020.

[66] Thomas Weng, Anthony Nwokafor, and Yuvraj Agarwal. Buildingdepot 2.0: An integrated management system for building analysis and control. In *Proc. of ACM BuildSys*, pages 1–8, 2013.

[67] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. Motelab: A wireless sensor network testbed. In *Proc. of IEEE IPSN*, 2005.

[68] Song Wu, Die Hu, Shadi Ibrahim, Hai Jin, Jiang Xiao, Fei Chen, and Haikun Liu. When fpga-accelerator meets stream data processing in the edge. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1818–1829. IEEE, 2019.

[69] Changqing Xia, Xi Jin, Chi Xu, Yan Wang, and Peng Zeng. Real-time scheduling under heterogeneous routing for industrial internet of things. *Computers & Electrical Engineering*, 86:106740, 2020.

[70] Chenren Xu, Shuang Jiang, Guojie Luo, Guangyu Sun, Ning An, Gang Huang, and Xuanzhe Liu. The case for fpga-based edge computing. *IEEE Transactions on Mobile Computing*, 2020.

[71] Ran Xu, Chen-lin Zhang, Pengcheng Wang, Jayoung Lee, Subrata Mitra, Somali Chaterji, Yin Li, and Saurabh Bagchi. Approxdet: content and contention-aware approximate object detection for mobiles. In *Proc. of ACM SenSys*, 2020.

[72] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. Deep compressive offloading: speeding up neural network inference by trading edge computation for network latency. In *Proc. of ACM SenSys*, 2020.

[73] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. Awstream: Adaptive wide-area streaming analytics. In *Proc. of ACM SIGCOMM*, 2018.

[74] Hong Zhang, Yupeng Tang, Anurag Khandelwal, Jingrong Chen, and Ion Stoica. Caerus: Nimble task scheduling for serverless analytics. In *Proc. of USENIX NSDI*, 2021.

[75] Wenzhao Zhang, Hongchang Fan, Yuxuan Zhang, Yi Gao, and Wei Dong. Enabling rapid edge system deployment with tinyedge. In *Proc. of ACM SIGCOMM Posters and Demos*, 2019.

[76] Wuyang Zhang, Jiachen Chen, Yanyong Zhang, and Dipankar Raychaudhuri. Towards efficient edge cloud augmentation for virtual reality mmogs. In *Proc. of ACM/IEEE SEC*, 2017.

[77] Xiao Zhu, Jiachen Sun, Xumiao Zhang, Y Ethan Guo, Feng Qian, and Z Morley Mao. Mpbond: efficient network-level collaboration among personal mobile devices. In *Proc. of ACM MobiSys*, 2020.