Received 25 October 2023, accepted 28 November 2023, date of publication 4 December 2023,
date of current version 13 December 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3339154

RESEARCH ARTICLE

Phoenix: Transformative Reconfigurability for Edge IoT Devices in Small-Scale IoT Systems

MORTEZA MOGHADDASSIAN, (Member, IEEE), SHAYAN SHAFAGHI, POOYAN HABIBI, (Member, IEEE), AND ALBERTO LEON-GARCIA, (Life Fellow, IEEE)
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada

Corresponding author: Morteza Moghaddassian (m.moghaddassian@utoronto.ca)

ABSTRACT Transformative reconfigurability refers to the ability of changing the current software stack of a configurable device by fully replacing its existing one. In the context of IoT systems, such major device reconfigurations can be used to change the role, adapt new functionality, and keep reconfigurable IoT devices compatible with the IoT systems requirements as the ambient technology around them evolve, thus fostering a thriving and continuously-connected IoT environment. In this paper, we introduce Phoenix, an IoT device configuration management system that is designed to automate transformative reconfigurability for edge IoT devices at small scales. Edge IoT devices are typically computationally capable and configurable devices that have enough processing power to run user programs and control sensors and embedded devices in an IoT environment. Enabling transformative reconfigurability for such devices at small scales can increase IoT systems flexibility, efficiency, and adaptability in small IoT environments, for example, agri-farms, smart homes, micro grids, and the like. Phoenix manages the life cycle of edge IoT devices configuration and uses bare-metal provisioning to provide unattended installation of new software stacks that are defined by user intents that instruct the reconfiguration process. We implemented a Phoenix proof-of-concept system and deployed it on the SAVI testbed where we evaluated its performance in reconfiguring a variety of edge IoT devices under different network conditions. Our results indicate that Phoenix can meet the requirements of small-scale heterogeneous IoT systems in various application environments.

INDEX TERMS Bare metal provisioning, infrastructure automation, Internet of Things, reconfigurability.

I. INTRODUCTION

The Internet of Things (IoT) represents an emerging class of distributed systems that provide enhanced awareness and control of the physical environments [1]. These systems (see Figure 1) typically use a wide range of capable and configurable edge IoT devices (e.g., Raspberry Pi 4, Jetson Nano, and Panda Latte 3) to collect data from sensors, embedded devices, and data sources, to feed such data to local edge applications (i.e., users programs) for immediate and delay-sensitive processing, and to transmit the processed data to the cloud for persistent storage and for processing by cloud applications to enable remote monitoring, automation, predictive maintenance, and customizing users experience [1]. Today, edge IoT devices are used in different infrastructures and environments such as homes, farms, hospitals, factories, roads, and cities and empower IoT applications and services that are becoming essential for infrastructure management in various areas such as healthcare, agriculture, transportation, and energy systems [2], [3], [4], [5].

To meet the dynamically changing requirements of diverse IoT applications, edge IoT devices must be agile and have the ability to adjust with changes that may involve both hardware and software components. Hardware improvements may involve introducing new hardware capabilities (e.g., adding new communication interfaces and sensors) or performing maintenance upgrades on existing hardware that would typically require human intervention to implement. Software improvements, on the other hand, may involve changes and maintenance upgrades on the edge IoT devices applications, services, and operating systems and are preferably performed with little to no human intervention (that is, fully automated) to avoid timely and expensive field visits.

Software changes are sometimes minimal as in adjusting application configurations or making incremental updates for security reasons, or they may be more significant such

The associate editor coordinating the review of this manuscript and approving it for publication was Tawfik Al-Hadhrami.

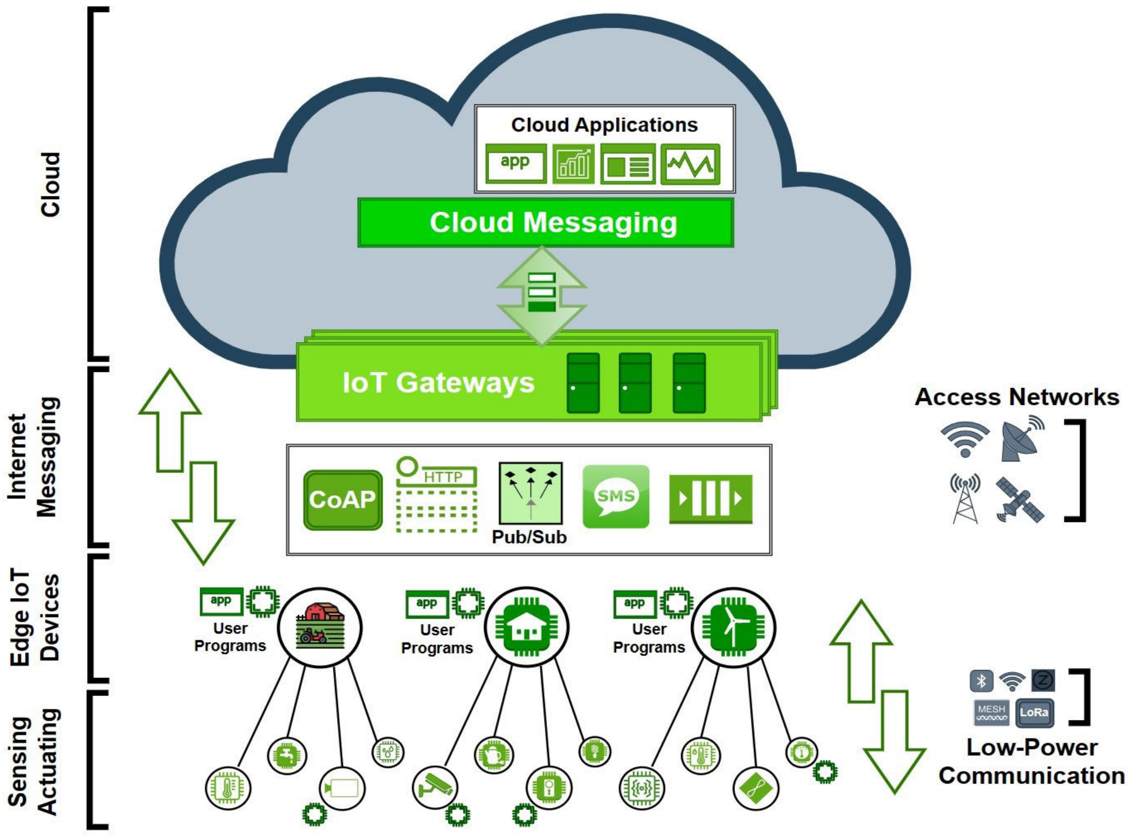

FIGURE 1. A view of IoT edge-cloud continuum, showing several IoT verticals with programmable edge IoT devices.

as changing a device's behavior, running new workloads, or adding new software capabilities to edge IoT devices. Although both groups of software changes apply modifications on a device software stack, the latter may include more fundamental changes that can involve modifying the entire device software stack (operating system and applications) and may require a comprehensive solution that can reliably reconfigure edge IoT devices across a range of scales and heterogeneity in various IoT environments.

In this paper, we focus on automated reconfigurability of edge IoT devices for scenarios where substantial and transformative reconfigurations of edge IoT devices are needed. Transformative reconfiguration entails that an edge IoT device can undergo a major or clean-slate change of its software stack, including its operating system, applications, and services. Our focus is to support transformative reconfigurability of edge IoT devices for IoT systems at small scales (e.g., smart farms, homes, buildings, medium to small factories, micro grids, hospitals, and small transportation systems) using Phoenix, an IoT device configuration management system that we designed for this purpose. Phoenix can support heterogeneous edge IoT devices and applications and can be used to support the requirements of IoT testbeds [6].

With Phoenix, the reconfiguration process can be triggered internally by the edge IoT devices or by external stimuli such as a user or an application intent. Reconfiguration can be done on both wired and wireless edge IoT devices using Over the Air (OTA) programming [7]. To support various IoT applications, edge IoT devices typically offer a variety of connectivity options (e.g., Wi-Fi, LoRa, cellular, etc.), computational power, input/output capabilities, and storage models. To be Phoenix-compatible, edge IoT devices must support iPXE/PXE [8], [9] which can be provided via the devices UEFI [10], through an intermediate boot loader, or sometimes by the device network interface card firmware. Phoenix can apply transformative changes on a single or a group of compatible edge IoT devices.

Transformative reconfigurability of a group of edge devices allows part or all of the devices in an IoT environment to be changed and adapted in an automated fashion for reasons such as providing new functionalities, meeting new application requirements, supporting dynamic resource allocation, switching workloads, and for remaining compatible and optimized. Generally, transformative reconfigurability extracts greater value from IoT investments thus promoting a thriving, continuously connected, and multipurpose IoT environment.

The rest of this paper is organized as follows. In Section II, we describe the Phoenix architecture and design. In Section III, we describe the Phoenix's proof-of-concept (POC) system implementation and deployment. Section IV, presents our evaluation results from testing transformative reconfigurability on various edge IoT devices

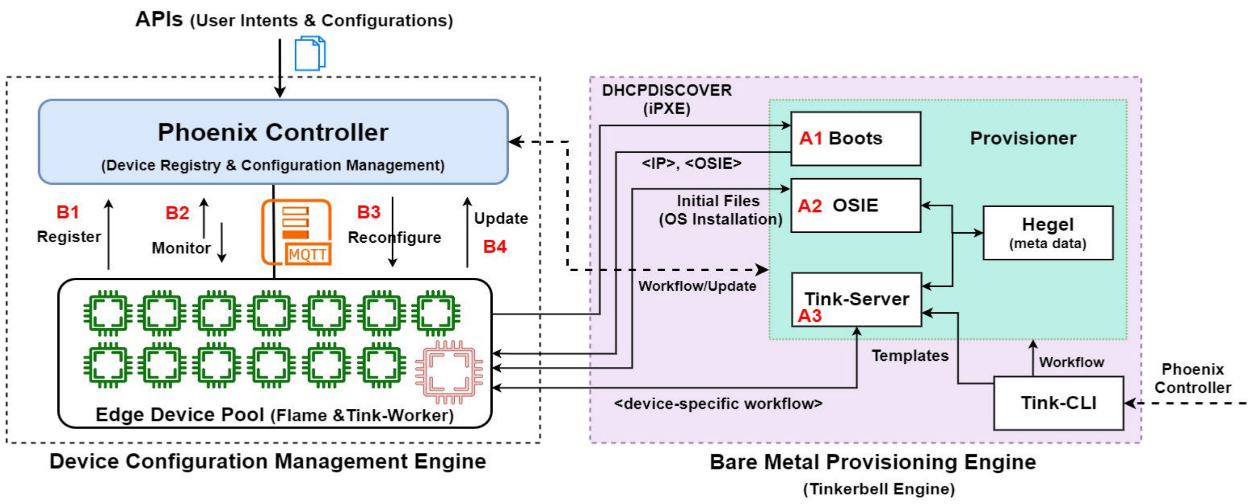
APIs (User Intents & Configurations)
Phoenix Controller
(Device Registry & Configuration Management)
B1 Register
B2 Monitor
MQTT
B3 Reconfigure
Update B4
Edge Device Pool (Flame &Tink-Worker)
Device Configuration Management Engine
DHCPDISCOVER (iPXE)
<IP>, <OSIE>
Initial Files (OS Installation)
Workflow/Update
A1 Boots
A2 OSIE
A3 Tink-Server
Provisioner
Hegel (meta data)
Templates
Workflow
Tink-CLI
Phoenix Controller
<device-specific workflow>
Bare Metal Provisioning Engine
(Tinkerbell Engine)

FIGURE 2. **A high-level view of Phoenix architecture with tinkerbell used as the provisioning engine.**

using Phoenix. Section V and VI, provide various discussions about reconfigurability requirements of different IoT applications and discuss use cases that can benefit from transformative reconfigurability. In section VII, we provide a comprehensive discussion of related work, and Section VIII gives a summary and discusses future work. We end this paper with a discussion of disclaimers.

II. PHOENIX SYSTEM OVERVIEW

As stated in section I, Phoenix is an IoT device configuration management system designed for automating transformative reconfigurability of edge IoT devices in small-scale IoT systems. Such IoT systems are typically limited, confined, and consist of a relatively small number of configurable edge IoT devices, for example, Raspberry Pi devices. We note that an IoT system may include a wide range of configurable IoT devices, including some with limited computing and storage capacity such as home appliances and smart locks that can also be remotely configured. These devices are not usually considered edge IoT devices and are not the focus of Phoenix and this study. The focus of Phoenix is on edge IoT devices with sufficient computing and storage capacity that can at least run a real-time operating system and meet the requirements of bare metal provisioning engines.

A. MAIN SYSTEM COMPONENTS

Phoenix has two main components: 1) a device configuration management engine and 2) a bare metal provisioning engine, as shown in Figure 2. The device configuration management engine is composed of the Phoenix controller and a distributed Edge Device Pool. The controller oversees the life cycle of edge IoT devices configuration after they join the Phoenix edge device pool for the first time. It also registers users, enables them to define workflows for their edge devices, and allows the users to programmatically invoke reconfiguration of specific edge IoT devices by means of intents. The bare metal provisioning engine in Phoenix is responsible for configuring edge IoT devices by deploying user-defined workflows on associated edge IoT devices. Currently, we use an open source Tinkerbell bare metal provisioning engine [11] for this purpose, but the Phoenix architecture and the services it provides are designed to be agnostic to the choice of the bare metal provisioning engine and can work with any preferred engine. Hence without loss of generality, the discussion in section II-B is specific to the Tinkerbell provisioning engine used in this study. Similar steps can be followed for other provisioning engines [12].

The Tinkerbell provisioning engine [11] has three main components: 1) Provisioner, 2) Tink-Worker, and 3) Tink-CLI, as shown in Figure 2. The Provisioner consists of several microservices that are responsible for handling the creation and deployment of workflows that configure the edge IoT devices. The Tink-Worker is a service that runs on the client/worker devices (that is, the edge IoT devices) that have been selected for provisioning. The Tink-worker service communicates with the Tink-Server to download and execute workflows that configure the client devices. The Tink-CLI is used to define workflows, insert templates, and add client-specific data. The Tink-CLI has recently been replaced by Rufio [11].

B. INITIALIZING EDGE IoT DEVICES

The first step to prepare an edge IoT device for Phoenix is to add the device in the Phoenix edge device pool, also shown in Figure 2. We assume that initially an edge IoT device is not configured and has no function to perform. At this stage, the edge IoT device must first initiate communication with the bare metal provisioning engine (i.e., Tinkerbell in this study) to undergo an initial configuration. To initiate communication with the Tinkerbell provisioning engine an edge IoT device must be pre-configured to run in iPXE mode [9]. IPXE is an open source implementation of the Preboot Execution Environment (PXE) [8] that specifies a standard client-server execution environment where clients (i.e., edge IoT devices) can retrieve and boot a pre-defined software stack (i.e., workflow) from the network. When an edge IoT device boots in iPXE mode, it broadcasts a DHCPDISCOVER packet to fetch a desired IP address [9], [11]. The request is captured and handled by the Tinkerbell Boots microservice, as shown in Figure 2 (step A1). The Boots microservice offers the desired IP address that was in the DHCPDISCOVER request to the given edge IoT device and passes the Tinkerbell OSIE microservice endpoint to the device to contact next [11].

The edge IoT device then asks the OSIE microservice for the files and configurations needed to prepare an in-memory execution environment for workflow deployment, as shown in Figure 2 (step A2). The workflow deployment involves installation of an entire software stack (i.e., operating system and its applications, services, and configurations) on the device. After communicating with OSIE, the edge IoT device will then interact with the Tink-Server (Figure 2, step A3) to obtain the specific workflow (i.e., software stack) image that has been prepared for the given edge IoT device [11]. Users can leverage the Phoenix APIs to define initial workflow configurations for their given devices, as shown in Figure 2. After the workflow image is obtained by the edge IoT device, it will be executed on the device, and upon successful completion, will boot the device into its initial configuration. We note that we can define a default initial (onboarding) workflow for all edge IoT devices that join Phoenix, or we can use different workflows for different device types (e.g., Raspberry Pi, etc.) or for different application environments where the edge IoT devices are deployed.

Once an edge IoT device boots into its initial configuration, it contacts the Phoenix controller that performs device registry and configuration management. Although, it is not necessary, the Phoenix controller can be placed on the same server where the IoT gateway is deployed to be easily reachable by all edge IoT devices. The same also applies to the bare metal provisioning engine. To establish communication with the Phoenix controller, the edge IoT device uses a client service, called Flame (see Figure 2) that comes as part of the initial software stack installed on the edge devices. We note that all devices that join Phoenix will obtain Flame during initial and forthcoming configurations.

When an edge IoT device uses Flame to contact the Phoenix controller for the first time, it introduces itself (Figure 2, step B1) by informing the controller of its device type (e.g., Raspberry Pi), MAC and IP addresses, admin user(s), energy source and battery status (if applicable), and the time since the device was booted to its initial configuration. The Phoenix controller stores this information in its registry for the device and adds the device into the Phoenix edge device pool. The controller uses the information provided by the device to continually pull other relevant information about the device (e.g., associated user-defined workloads) from the Tinkerbell Provisioner. This information will be used by the Phoenix controller when processing intents (see section II-C) and to roll back the device to its initial configuration in situations where a future reconfiguration attempt fails to complete.

C. RECONFIGURING EDGE IoT DEVICES

After an edge IoT device is added to the Phoenix edge device pool, the Phoenix controller monitors the edge device configuration life cycle, as shown in Figure 2 (step B2). The controller and the edge device communicate using Message Queuing Telemetry Transport (MQTT) protocol [13] where the edge device informs the controller of its configuration changes such as the energy levels for battery powered devices, operating system upgrades, newly installed applications and services by its users, and other user-applied changes. Phoenix is very extensible to the type of configuration information that edge IoT devices and the Phoenix controller can exchange during the monitoring phase.

The information the edge IoT devices provide to the Phoenix controller during the monitoring phase will be stored on the device records in the controller and can be used to plan reconfiguration strategies. During the monitoring phase, edge IoT devices can be automatically selected, directed through users intents, or self-triggered for reconfiguration, as shown in Figure 2 (step B3). The latter can happen safely by using the Flame client services or unwantedly via an internal stimulus that unsafely triggers reconfiguration process by rebooting the device in iPXE mode. In unsafe scenarios, the device contacts the provisioner to obtain a new workflow/configuration (i.e., software stack). However, rebooting into a new configuration without the knowledge of Phoenix controller will lock the device for safety reasons and will prevent the device from reconfiguration. The device admin user will be notified to either remove the lock or investigate the event.

If Flame client services are used for self-reconfiguration, the device asks the Phoenix controller for reconfiguration before attempting to reboot itself. In this scenario, the Phoenix controller instructs the device to safely reboot in iPXE mode to obtain a new workflow/configuration (i.e., software stack) by following the exact same steps presented in section II-B. Once the new configuration is successfully installed on an edge IoT device, the device updates its status with the Phoenix controller, as shown in Figure 2 (step B4).

```
- target: "device"
  id: <device mac/ip address>
  configuration: <intended_workflow>
- target: "device"
  id: <device mac/ip address>
  configuration: <intented_workflow>
```

```
- target: "class"
  type: "device"
  name: "raspberry pi"
  configuration: <intended_workflow>
```

```
- target: "class"
  type: "os"
  name: "raspbian"
  configuration: <intended_workflow>
```

FIGURE 3. An example YAML-based presentation of intents in Phoenix.

Similar steps are followed for obtaining a new workflow (i.e., software stack) when the reconfiguration process is initiated by a user intent. A major difference is that the edge

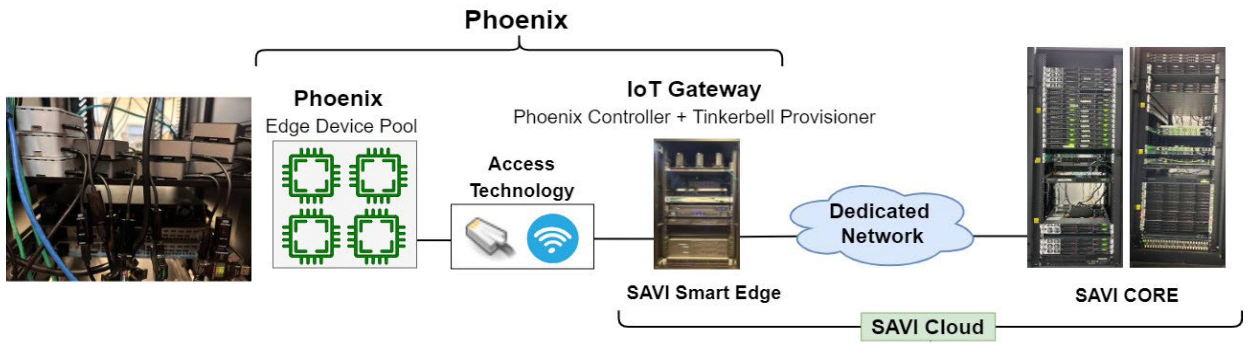

FIGURE 4. A view of the SAVI testbed demonstrating the deployment of Phoenix architecture.

IoT devices are now instructed through external stimuli (i.e., user intents). A Phoenix intent is a YAML-based definition of a desired state/configuration for an individual or group of edge IoT devices that are in the edge device pool. In the example set of intents shown in Figure 3, edge IoT devices (i.e., target devices) are selected for reconfiguration based on their MAC or current IP addresses or in bulk by creating target classes that specify a group of devices based on their types, operating system, and other grouping criteria. Phoenix also has the ability to assign edge IoT devices to specific IoT verticals, for example, the edge IoT devices that are related to smart farming in a particular region, or to smart homes of interest. Users can use vertical type or spatio-temporal attributes to reconfigure the edge IoT devices for which they have administrative rights.

Once an intent is created by a user, it is passed to the Phoenix controller for rendering using the Phoenix controller APIs, as shown in Figure 2. The controller processes the intent and signals the target edge devices in the pool to reboot in iPXE mode if the device information in the intent matches the existing pool devices, their device records, and intended workflows. The target edge devices then follow the same steps described in section II-B to undergo reconfiguration without being locked for security reasons. We note that users who create intents must have administrative rights to manage the target devices. These users can directly use Phoenix APIs to define the intended workflows for their target edge IoT devices, as shown in Figure 2.

III. PHOENIX IMPLEMENTATION

We created a Phoenix proof-of-concept (POC) system to demonstrate the practicality of Phoenix in real-life scenarios involving edge IoT devices in small-scale IoT systems. We deployed our POC system on the SAVI IoT testbed. In the following subsections we briefly introduce the SAVI testbed and discuss our proof-of-concept implementation and deployment in more details.

A. THE SAVI IoT TESTBED

The Smart Applications on Virtual Infrastructure (SAVI) [14] is a private research cloud and network testbed that provides a multi-tier cloud infrastructure. SAVI has two infrastructure tiers: the SAVI Core and the SAVI Smart Edges, as shown in Figure 4. The SAVI Core provides high-performance compute and storage nodes for cloud applications, while the SAVI Smart Edges connect end-user devices to the testbed and provide computing closer to the end users to meet low-latency applications requirements. The two tiers provide a fully connected environment for IoT applications. Currently, end-user devices can either directly connect to SAVI Smart Edges via SAVI-supported access technologies or through the Internet. SAVI currently supports various wired and wireless access technologies, including Ethernet, Wi-Fi, Bluetooth, and LoRa for direct device connectivity, and it is expanding its access to support 5G radio. The SAVI Core and Smart Edges are connected directly via a dedicated network, as shown in Figure 4. The SAVI Core node is placed in Bahen Center for Information Technology at University of Toronto and the SAVI Smart Edges are distributed across different locations in the Greater Toronto Area (GTA). In this study, we used the SAVI Toronto Smart Edge node to place our IoT Gateway, Phoenix Controller, and Tinkerbell Provisioning engine. This SAVI Edge node is directly connected to the SAVI Core node.

B. POC IMPLEMENTATION AND DEPLOYMENT

As explained in Section II-A, Phoenix has two major components: the device configuration management engine and the bare metal provisioning engine. As shown in Figure 4, we deployed the IoT Gateway and the Tinkerbell bare metal provisioning engine on the SAVI Toronto Smart Edge node. The IoT Gateway role is to handle messaging with cloud applications, as shown in Figure 1. (We hope that in the future we can extend the Phoenix services to cloud-based automation and monitoring applications that are running on the SAVI IoT testbed.) The Phoenix device configuration management engine has two major components, the Phoenix controller and the edge device pool. The controller is deployed on the SAVI Toronto Smart Edge node and uses the PostgreSQL [15], [16] to maintain its device registry and the Nginx [17] to power its API server.

The edge device pool is distributed in different geographical locations and currently contains a set of heterogeneous edge IoT devices. These edge devices include Raspberry Pi 3, Raspberry Pi 4 (8GB configuration models), Panda Latte 3, BeeLink Mini PCs, and Intel NUC II. In the future, we plan to add Jetson Nano, Jetson AGX, Asus Thinker, and Beagle Bone devices. All edge IoT devices in the edge device pool can use Ethernet or Wi-Fi to connect directly to the SAVI Toronto Smart Edge Node where the Phoenix controller and Tinkerbell provisioning engine are also deployed, as shown in Figure 4. We use MQTT messaging between the Phoenix controller and the edge IoT devices in the device pool for which we leverage the open source Mosquitto message broker [18], [19]. The message broker is also deployed on the SAVI Toronto Smart Edge Node. We note that our Phoenix deployment strategy is fully aligned with the real-life IoT system design, also presented in Figure 1: that is all our system components are deployed where they would usually be deployed in practice, giving us confidence that our POC evaluations (see section IV) can provide tangible and realistic evaluation results.

IV. EVALUATION OF EDGE IoT DEVICE CONFIGURATION USING PHOENIX POC

To assess the ability of Phoenix to meet our small-scale design goal, we measured the performance of edge IoT devices in terms of boot time and energy consumption during the reconfiguration process. Boot time is the duration it takes for an edge IoT device to be instructed for reconfiguration by the Phoenix controller, transformed to its new configuration, and to update the Phoenix controller of its new status. Energy consumption is the energy consumed by an edge IoT device during the boot time.

In our experiments we used a variety of edge IoT devices to demonstrate the scale and heterogeneity needed by small-scale IoT systems. Raspberry Pi 4 is the main edge IoT device used in our evaluations in sections IV-A and IV-B because of its highly affordable and popular nature that allows us to generalize our experiment results to a wide range of IoT applications. We report performance evaluations on other edge IoT devices in section IV-C.

A. RASPBERRY PI 4 (FIVE-DEVICE SCALE)

We first measured the boot time and energy consumption during the reconfiguration process for a single Raspberry Pi 4 Model B device [20] (see Figure 5) and then increased by one the number of Raspberry Pi 4 devices that undergo concurrent reconfiguration, up to the five-device scale.

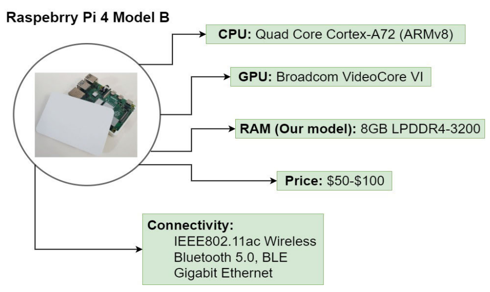

FIGURE 5. Raspberry Pi 4 Model B specifications.

In each trial, we connect the given number of unconfigured Raspberry Pi 4 devices (e.g., one, two, etc.) to Phoenix and allow them to be configured to their initial configuration, and shortly there after, to have them concurrently undergo two rounds of reconfiguration invoked by the Phoenix controller. The results in the figures below are for the average of the performance metrics in the two reconfiguration rounds of all the devices included in each trial. We explored a variety of network conditions by repeating the above cycle of evaluations in settings with different levels of packet loss, ranging from zero to five percent. The workflow image size is 2.5GB in all rounds of reconfiguration.

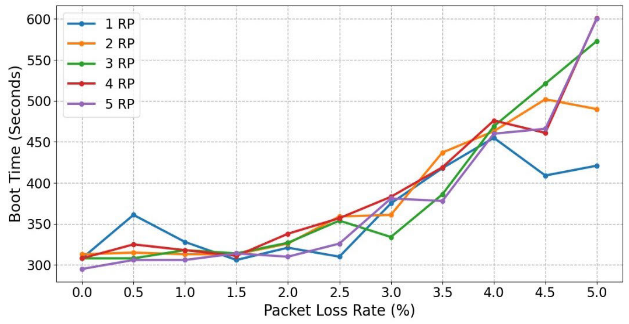

FIGURE 6. Average boot time for Raspberry Pi 4 Model B devices in the presence of network packet loss.

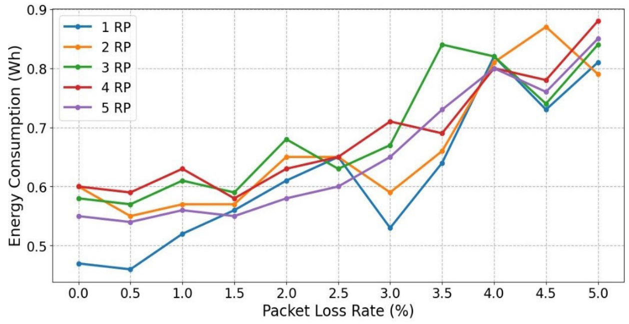

FIGURE 7. Average energy consumption for Raspberry Pi 4 Model B devices in the presence of network packet loss.

As shown in Figures 6 and 7, the average boot time for the lower packet loss rates is about 5-6 minutes and the average energy consumption is between 0.55 to 0.65 Watt-hour per device in 1 to 5 device-scale. We can see that when the packet loss rate is zero, the average boot time per device remains nearly constant as the number of Raspberry Pi 4 devices increases from 1 to 5 (see Figure 6). However the average boot time increases gradually up to a packet loss rate of 2.5 percent and then increases at a faster rate as the packet loss rate increases.

Figure 7 shows that the average energy consumption per device for a single Raspberry Pi 4 is lower than when more Raspberry Pi 4 devices are concurrently reconfigured. Nonetheless, as with boot time, the average energy consumption of Raspberry Pi 4 devices increases slowly up to around 2.5 percent, and then increases at a faster rate as the packet loss rate increases. We can also see that average energy consumption of Raspberry Pi 4 devices increases with increasing boot time. We measured energy consumption using special hardware that is designed for this purpose to improve measurement accuracy (see Figure 8).

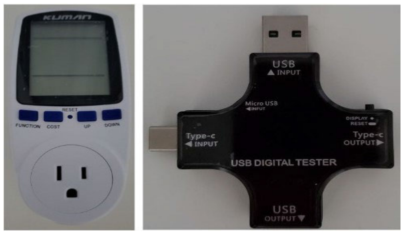

FIGURE 8. Energy tester equipment (Left is a tester for non-USB powered devices and Right is a tester for USB powered devices).

We believe that the reason why increasing packet loss rates impact boot time and energy consumption levels of Raspberry Pi 4 devices are twofold. First, this can be partly due to retransmissions in the Trivial File Transfer Protocol (TFTP) [21] that is used by iPXE/PXE to load parameters and configurations [8], [9]. The TFTP protocol uses simple ARQ mechanisms for reliability [21]. Second, Tinkerbell uses HTTP(S) for workload image retrieval which relies on TCP byte streaming for image delivery [11]. As the packet loss rate increases in the network, the TCP session can observe more timeouts and consequently devices may experience more delays and consume more energy due to packet retransmissions. This reasoning similarly applies to all other edge IoT devices in our experiments. We believe that by using other protocols supported by iPXE clients like Fiber Channel over Ethernet (FCoE) [22] or leveraging new Internet protocols for image retrieval like NDN [23] with its ability to use network caches we can help to reduce the boot times and energy consumption at higher network packet loss rates.

It is also important to note that as indicated in prior work [24], Raspberry Pi 4 devices may exhibit performance fluctuations during workload execution. This may also result in some fluctuations in the observed boot time and energy consumption levels. We repeated each round of experiment to reduce such effects.

B. RASPBERRY PI 4 (TEN-DEVICE SCALE)

To confirm our results at higher scales, we extended our setup to 10 Raspberry Pi 4 devices. Table 1 shows that the average boot time and energy consumption levels for 10 Raspberry Pi devices exhibit the same pattern as for the smaller number of Raspberry Pi 4 devices do (see section IV-A). Below %2.5 loss rate, the average boot time per device hovers between 5-6 minutes for the same workflow image size (i.e., 2.5GB) and the average energy consumption per device is also similarly between 0.55 to 0.65 Watt hour. Average boot time and energy consumption rise quickly as

TABLE 1. Average boot time and energy consumption levels for 10 Raspberry Pi 4 Model B devices in the presence of network packet loss.

Measurements / Loss Rate (%)	Boot Time (Seconds)	Energy Consumption (Wh)
0	303	0.58
0.5	326	0.61
1	341	0.60
1.5	332	0.59
2	329	0.61
2.5	361	0.67
3	405	0.71
3.5	413	0.76
4	465	0.86
4.5	521	0.85
5	593	0.89

the loss rate increases beyond %2.5. From these experimental results, we can conclude that Phoenix is able to deliver a consistent performance under network packet loss rates less than %2.5 as the number of edge IoT devices (i.e., Raspberry Pi 4 devices in this case) increases.

At higher packet loss rates, Raspberry Pi 4 devices exhibit higher boot times and energy consumption levels but the higher values appear to be independent of the number of devices. The higher measured values are related to retransmissions which can be partially optimized via protocol changes/improvements as discussed earlier. We note that the packet loss rate in Ethernet and Wi-Fi networks as well as in the Internet is typically deemed acceptable between 0.1 to 1 percent where Phoenix can exhibit a consistent performance regardless of the number of parallel reconfigurations. Beyond %1 percent, the quality of communication may begin to deteriorate depending on the nature of data and applications.

We also note that the network bandwidth and the image server capacity in terms of available Memory and CPU can become a bottleneck if the available bandwidth is low or if the number of edge IoT devices undergoing concurrent reconfiguration scales beyond the processing power of the image server. In these situations, scaling the network bandwidth or increasing the image server capacity can be beneficial. In our experiments, the image server is equipped with 8 physical CPU cores (64-bits) and 32GB of RAM.

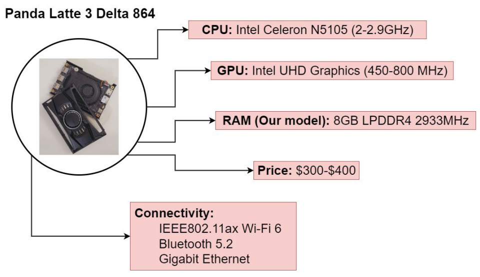
Panda Latte 3 Delta 864
CPU: Intel Celeron N5105 (2-2.9GHz)
GPU: Intel UHD Graphics (450-800 MHz)
RAM (Our model): 8GB LPDDR4 2933MHz
Price: $300-$400
Connectivity:
IEEE802.11ax Wi-Fi 6
Bluetooth 5.2
Gigabit Ethernet

FIGURE 9. Panda Latte 3 Delta 864 specifications.

C. OTHER EDGE IoT DEVICE TYPES

In this section, we repeat the experiment we performed on Raspberry Pi 4 with several other edge IoT devices to expand the heterogeneity of our measurements. We follow the same steps and use the same workflow image size of 2.5GB.

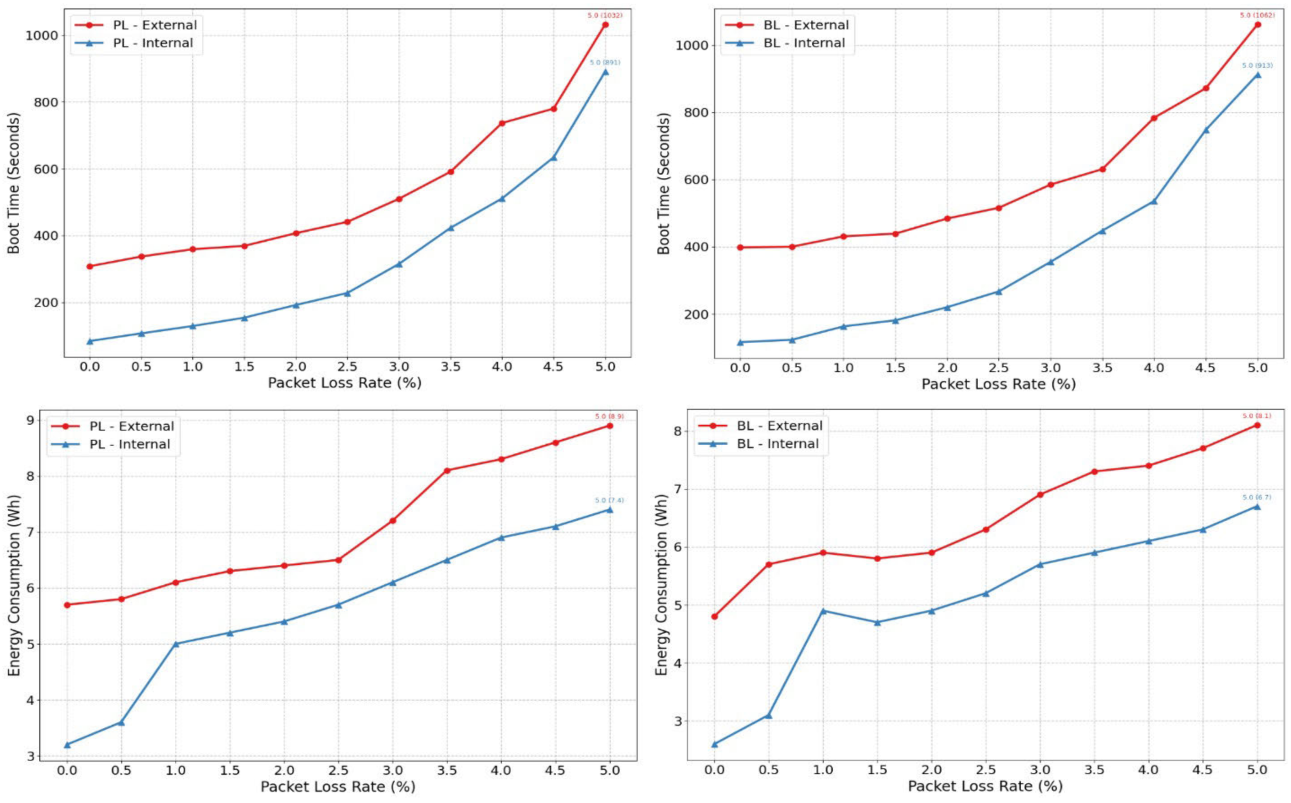

FIGURE 10. **The average boot time and energy consumption levels of Panda Latte 3 Delta 864 (Left) and BeeLink Mini PC M1 (right) in the presence of network packet loss.**

1) PANDA LATTE 3 DELTA 864

Panda Latte 3 [25] is a powerful Single Board Computer (SBC) that supports an array of input/output options, as shown in Figure 9. Panda Latte 3 comes with a semi-powerful Intel Celeron CPU that can support a diverse set of IoT applications ranging from smart homes to industrial IoT, and automation applications. Panda Latte 3 can be a reasonable choice for a capable edge IoT device. We measured the performance of Panda Latte 3 in terms of the average boot time and energy consumption during reconfiguration process with Phoenix. We carried our measurements in two different setups where: 1) the device internal eMMC storage is used for workflow deployment, and 2) an external USB Memory is used for workflow deployment. We do this to allow a fair comparison with Raspberry Pi 4 performance in terms of boot time and energy consumption during reconfiguration process, and second, to support scenarios that may involve using different storage systems by different IoT applications. We note that Raspberry Pi 4 devices do not come with an embedded internal memory.

Our measurement results of boot time (see Figure 10) reveals that the average boot time is significantly less at lower packet loss rates (i.e., less than %2.5) when the Panda Latte internal eMMC memory is used. The difference in average boot time between using internal versus external memory systems become less distinct at higher packet loss rates which indicates that the impact of network condition on the boot time is higher than the performance of the underlying storage system used for workflow deployment as the packet loss rate increases.

In terms of the average energy consumption during reconfiguration, Panda Latte 3 noticeably consumes less energy when its internal eMMC memory is used for workflow deployment, as shown in Figure 10. Our results indicate that energy consumption can rise significantly for Panda Latte 3 devices regardless of the memory system used as the network packet loss rate increases above %2.5. Our results also indicate that the energy consumption of Panda Latte 3 devices at packet loss rates below %0.5 is significantly much lower when the Panda Latte 3 internal memory is used for workflow deployment. At higher packet loss rates (i.e., above %0.5), energy consumption of Panda Latte 3 devices can rise significantly on internal memory, as shown in Figure 10. We believe that this may be related to how increasing packet losses impacts the device I/O energy consumption.

2) BEELINK MINI PC M1

Like the Panda Latte 3, the BeeLink Mini PC M1 [26], [27] is a reasonable choice for a capable edge IoT device. BeeLink Mini PC devices are powerful small computers with several input/output options (see Figure 12) that can support

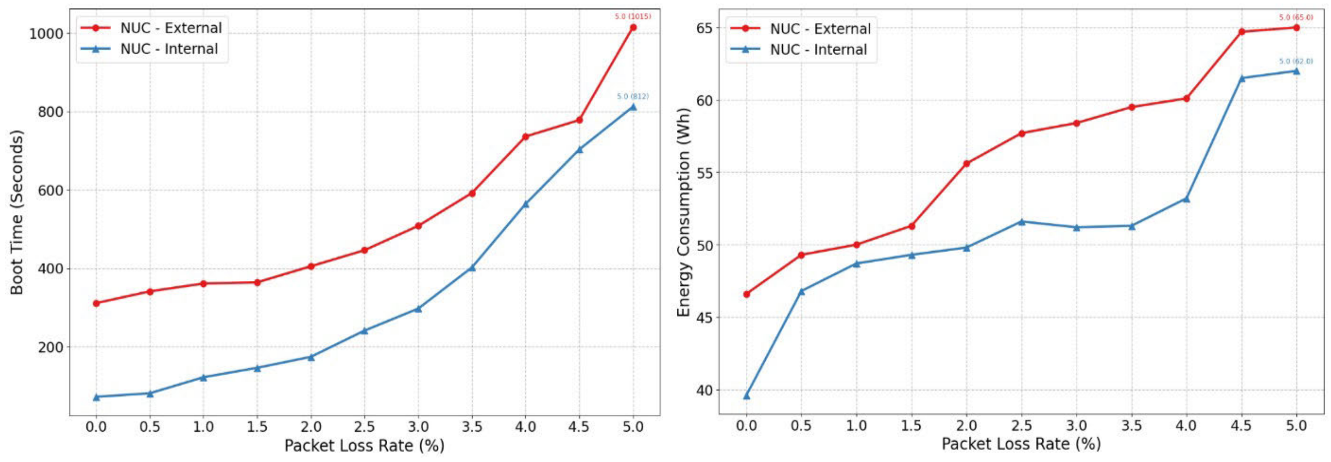

FIGURE 11. **The average boot time and energy consumption levels of Intel NUC 11 in the presence of network packet loss.**

a variety of IoT applications, especially those that require more powerful edge resources like industrial IoT, automation, and transportation systems. We measured the performance of BeeLink Mini PC M1 in terms of boot time and energy consumption during reconfiguration process with Phoenix using either its internal eMMC storage or an external USB memory in the presence of various packet loss rates.

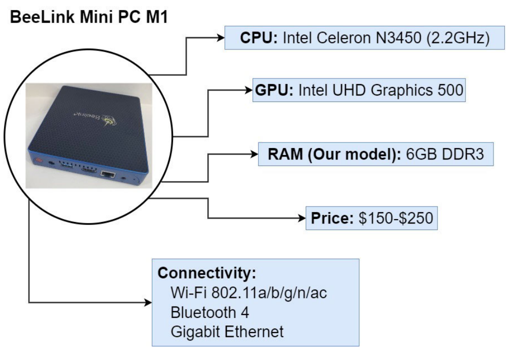

FIGURE 12. **BeeLink Mini PC M1 specifications.**

Figure 10, shows a similar performance between average boot time of BeeLink Mini PC M1 devices as Panda Latte 3 for scenarios that involve using internal as well as external storage systems. This is primarily due to the fact that both devices benefit from relatively similar hardware configurations with Panda Latte 3 have a slightly more powerful CPU and RAM (see Figures 9 and 12). For higher network packet loss rates (i.e., larger than %2.5), BeeLink Mini PC M1 devices can show on average 2 to 5 percent higher boot time. A comparable pattern of energy consumption can be also seen between the BeeLink Mini PC M1 devices and the Panda Latte 3 for both scenarios of using internal and external storage systems, as shown in Figure 10. However, our result suggest that BeeLink M1 devices can consume on average about 5 to 10 percent less energy than Panda Latte 3 devices during the boot time. Interestingly, our results suggest that Raspberry Pi 4 devices can deliver a much better boot time at higher network packet loss rates when compared with Panda Latte 3 and BeeLink Mini PC M1 on external memory at a much lower energy consumption. BeeLink Mini PC M1 and Panda Latte 3, however, can outperform Raspberry Pi 4 devices in boot time when an internal memory is used for workflow deployment for network packet loss rates below %3. We note that less than %1 percent network packet loss rate is deemed acceptable by various network applications on the Internet.

3) INTEL NUC 11 ENTHUSIAST

The Intel NUC 11 is a more performant device than the Panda Latte 3 or Beelink Mini PC M1. The use of NUC 11 as an Edge IoT device can be very beneficial to high-performance and AI-oriented IoT applications such as smart transportation, smart building, and federated learning applications [28]. The version used in our experiments is equipped with an 11th generation Intel Core-i7 CPU and a very high performance Nvidia Geforce RTX 2060 GPU, as described in Figure 13 [29]. The Intel NUC 11 is also equipped with Intel Ethernet i225-LM network interface card that is capable of delivering bandwidth up to 2.5Gbps. However, in our experiments, we used a Gigabit Ethernet for all devices for a fair performance comparison.

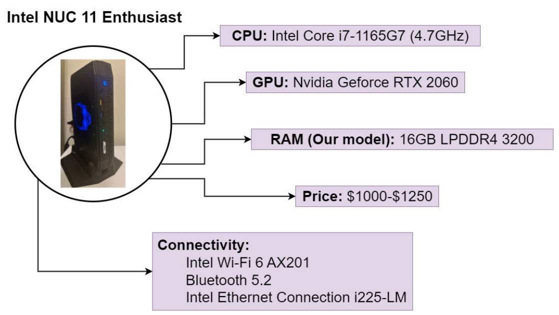
Intel NUC 11 Enthusiast
CPU: Intel Core i7-1165G7 (4.7GHz)
GPU: Nvidia Geforce RTX 2060
RAM (Our model): 16GB LPDDR4 3200
Price: $1000-$1250
Connectivity:
Intel Wi-Fi 6 AX201
Bluetooth 5.2
Intel Ethernet Connection i225-LM

FIGURE 13. **Intel NUC 11 enthusiast specifications.**

We measured the performance of Intel NUC 11 devices using both internal and external USB memory options. Our results suggest that (see Figure 11) the Intel NUC 11 can deliver a reasonably lower boot time than all other edge devices in our experiments for network packet loss rates below %3 when internal memory is used. However, when external memory is used or at higher network packet loss rates, the results are comparable to Panda Latte 3 and BeeLink Mini PC M1 devices. In terms of energy consumption, NUC 11 devices consume much higher energy during the boot time and workflow deployment than any other edge IoT device we used in our experiments. This is, however, expected as NUC 11 uses very powerful CPU, GPU, and Connectivity interfaces that can lead to higher power consumption.

4) RASPBERRY PI 3 MODEL B+

In our experiments, we tried to also work with Raspberry Pi 3 Model B+ devices [30]. Our findings show that while Raspberry Pi 3 Model B+ supports iPXE and is Phoenix-compatible, the lack of enough RAM to support large image retrieval can prevent them from successful workflow deployments. In our case, we used a 2.5GB workflow image in our experiments across all other edge devices. The 2.5GB image size does not work with Raspberry Pi 3 Model B+.

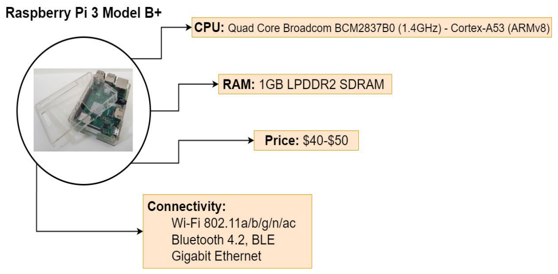

FIGURE 14. Raspberry Pi 3 Model B+ specifications.

We believe to successfully use Raspberry Pi 3 model B+ devices as a reconfigurable edge IoT device with Phoenix, a much smaller workflow image must be created and used, preferably less than 1GB in size. A difficulty is to create compatible images that also match the applications software requirement. We note that the 2.5GB workflow image we used in our experiments is a very basic and minimal workflow image that only contains a basic operating system and necessary services that allows the device to work with Phoenix successfully.

V. RECONFIGURABILITY REQUIREMENTS OF IoT APPLICATIONS

Transformative reconfigurability of edge IoT devices is a powerful tool for enabling IoT systems and applications to change role, adapt new functionalities, and remain compatible. In section IV, we evaluated the reconfigurability performance of various small PCs and Single Board Computers (SBCs) that can be used in different IoT environments. In this section, we review some common IoT applications in terms of their reconfigurability requirements and provide guidelines for choosing appropriate edge IoT devices to meet IoT application reconfigurability requirements. We hope the insights from this section will be applicable to a wider range of IoT application scenarios.

A. BOOT TIME SENSITIVITY

While boot time varies from one edge IoT device to another, IoT applications have different degrees of boot time sensitivity in order to remain functional and this affects the type of edge IoT devices they can use.

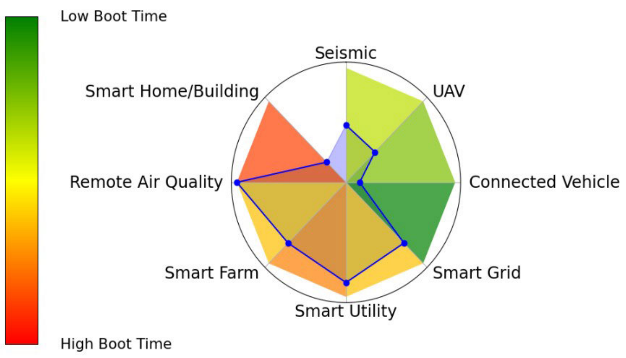

FIGURE 15. IoT applications boot time requirements.

Figure 15 considers some IoT applications where Seismic, UAV, and Connected Vehicles demand a low boot time. These applications are boot time sensitive and reconfiguration of edge IoT devices for these applications must be as fast as possible to ensure reliable and continued functionality. Under lower network packet loss rates (i.e., less than %3.5), these IoT applications can benefit from affordable edge IoT devices such as or similar to Panda Latte 3 and BeeLink Mini PC M1 with internal memory which have shown the fastest boot time in our tests, considering the devices cost. Generally, edge IoT devices with internal memory are the most ideal for low boot time IoT applications.

Intel NUC 11 with internal memory is the fastest to reconfigure under lower packet loss rates (i.e., less than %3.5). However, if we also consider cost and portability constraints (power and weight) of Intel NUC 11 devices, some applications like Seismic and UAV may prefer smaller devices like Panda Latte 3. Raspberry Pi 4 devices are also ideal for boot time sensitive applications. They show a higher boot time when compared with Panda Latte 3 and BeeLink Mini PC M1 with internal memory, but they are less expensive, more portable, consume far less energy, and can also perform relatively better under higher network packet loss rates (i.e., more than %3.5). We also note that boot time sensitive applications should use reasonable image sizes for workflow deployment since image size can affect boot time, especially in IoT applications where the network environment is not reliable, such as Seismic.

Many IoT applications can sustain a much longer boot time (reconfiguration time), as depicted by Figure 15. For example, a smart home application can tolerate a much longer boot time as sensors and smart devices in a smart home environment are less time sensitive to manage. The same applies to some extent to a smart farm or a smart building management system. For these applications, Raspberry Pi 4 devices are ideal as they offer a reasonable boot time and energy consumption while being very affordable. If cost is not an issue, other edge IoT devices are also ideal for such applications. For example edge IoT devices like the Intel NUC 11 can provide GPU processing that can provide AI-based safety features using image and video processing. Panda Latte 3 can also provide a wide range of input/output options that are suitable for smart homes and farming at an affordable price.

B. ENERGY CONSUMPTION REQUIREMENTS

IoT applications may also have different levels of energy constraints (see Figure 16) that may affect their choice of edge IoT devices. In general, the energy consumption of edge IoT devices during the reconfiguration process increases as the edge devices boot time increases, so it would appear that IoT applications with energy constraints can also benefit from devices that can deliver faster boot time to meet their reconfigurability requirements. However, this requires careful consideration. For example, Intel NUC 11 devices can deliver a very fast boot time under normal network conditions but they consume large amounts of energy to deliver that performance and are not ideal for energy sensitive IoT applications like UAV and Connected Vehicles, as shown in Figure 16.

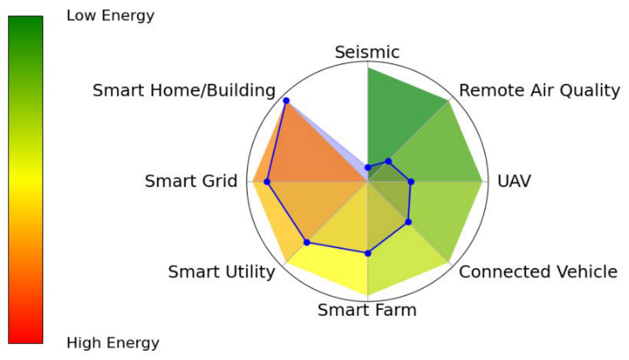

FIGURE 16. **IoT applications energy requirements.**

On the other hand, edge IoT devices such as or similar to Panda Latte 3 and BeeLink Mini PC M1 can relatively deliver the same boot time as Intel NUC 11 but with 5 to 6 times less energy consumption. These devices can be ideal for connected vehicles, UAV, and smart farming applications, as shown in Figure 16. For IoT applications with tighter energy constraints like Seismic, remote air quality, and even some UAV applications, Raspberry Pi 4 devices can deliver a reasonable performance with much less energy consumption. In IoT applications like Seismic and remote air quality, edge IoT device may function on battery for a period of time. Consuming low amount of energy (i.e., on average 0.55 to 0.65 Watt hour) during reconfiguration process makes Raspberry Pi 4 an ideal candidate for these type of applications to meet their reconfigurability requirements.

VI. OPPORTUNITIES FOR USING TRANSFORMATIVE RECONFIGURABILITY

In this section, we consider examples where transformative reconfigurability can benefit IoT applications. We also discuss unique opportunities where transformative reconfigurability is a technology enabler in IoT settings and beyond. We aim for the insights presented to have broader applicability across IoT systems and digital infrastructures that require automated reconfigurability.

A. ON-DEMAND WORKLOAD ADJUSTMENT

Transformative reconfigurability can empower edge IoT devices to perform on-demand tasks as requested by IoT applications. An example could involve a small group of edge IoT devices transforming from one role to another, such as transforming a simple protocol gateway to also support data pre-processing as part of a machine learning pipeline [31] or into becoming a managed firewall for protecting an entire or part of a connected infrastructure or environment [32], [33]. In a broader context, a heterogeneous class of edge IoT devices (e.g., all edge IoT devices in a region, all Connected Vehicles of a particular model belonging to a company, and all edge IoT devices in a power grid) may need to be transformed to adapt to new roles or to support new protocols and functionalities. These forms of reconfigurability may require transforming the entire edge device software stack and can directly benefit from Phoenix's ability of bulk reconfiguration.

B. CONTEXT SWITCHING

Edge IoT devices with transformative reconfigurability can be configured to dynamically switch context to support various application requirements. For example, the software stack of an edge IoT device in a smart home environment [34] can be switched between running in safety mode [35] when residents are not at home and comfort mode when residents are present in the home. We note that implementing such scenarios can be also done without changing the entire software stack of the edge IoT devices, but having the ability to safely transform these devices can allow their power and resources to be optimally used in accordance to changing conditions without needing to keep all workloads accessible and running at all times. To enable context switching, the Phoenix's client service (i.e., Flame) can be used to inform the Phoenix controller when a change of stack is needed. Basically, once the edge IoT device is informed that the condition of the home has changed (i.e., all residents left the premises), it can use Flame to request the Phoenix controller for reconfiguration. If this process has been allowed previously by the device admin user, Phoenix will perform a context switch on the corresponding device. If this process has not been allowed before, the Phoenix controller can lock the device to prevent the smart home application from being compromised (see section II-C).

C. IoT TESTBED

IoT testbeds are controlled environments with sensors, actuators, and integrated computing and storage capabilities that are specifically designed for testing and validating Internet of Things technologies, applications, and devices [6]. IoT system and application developers can use IoT testbeds to verify the applicability and functionality of their systems and applications, to analyze evolving IoT technologies, and to plan implementation strategies in smaller scales before conducting real-world experiments and deployments [36]. A system enabled with transformative reconfiguration can reduce the operational costs of the testbed and allow the developers to quickly validate and refine their implementation once it is changed without having to touch the actual devices and equipment [6], [36]. Transformative reconfigurability also allows IoT testbeds to dynamically allocate their resources (i.e., physical and virtual) to different IoT applications that use the testbed without needing to manually prepare the testbed resources for new usage.

D. IoT NETWORK RECONFIGURABILITY

In addition to programmable edge IoT devices, recent hardware developments in the area of reconfigurable Single Board Routers (SBRs) have shown promising designs and implementations that offer an array of programmable connectivity options. The Banana Pi BPI-R3 single board router shown in Figure 17 is one of the most versatile SBRs of its kind [37]. The Banana Pi BPI-R3 provides a low-power and powerful ARM Cortex A-53 CPU and 2GB of DDR RAM that together can empower the device to benefit from Phoenix's transformative reconfigurability power [37].

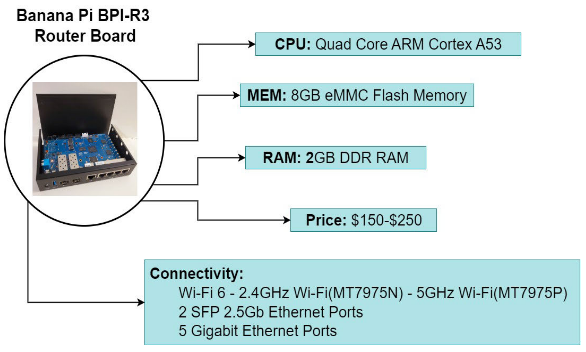
Banana Pi BPI-R3 Router Board
CPU: Quad Core ARM Cortex A53
MEM: 8GB eMMC Flash Memory
RAM: 2GB DDR RAM
Price: $150-$250
Connectivity:
Wi-Fi 6 - 2.4GHz Wi-Fi(MT7975N) - 5GHz Wi-Fi(MT7975P)
2 SFP 2.5Gb Ethernet Ports
5 Gigabit Ethernet Ports

FIGURE 17. Banana Pi BPI-R3 router board specifications.

Banana Pi BPI-R3 and similar other reconfigurable SBRs, like the Banana Pi BPI-R64 and Mikrotik router boards, are powerful devices that can be used as: 1) edge IoT devices with extended input/output capabilities and, 2) network routers to support building reconfigurable IoT networking infrastructures. In the latter case, Phoenix's capability of managing the life cycle of SBRs configuration (i.e., software stack) can enable versioning and on-demand adjustment of the IoT networking infrastructures. We note that Phoenix can also provide the same services to any networking infrastructure with reconfigurable hardware.

E. IoT SERVICE RECONFIGURABILITY

Service reconfigurability refers to the automation of service definitions, deployments, and configuration management of service resources. In the context of IoT, service reconfigurability enables IoT systems and applications to efficiently define, deploy, and update the software stack of several edge IoT devices that belong to the same service definition, as supported by Phoenix.

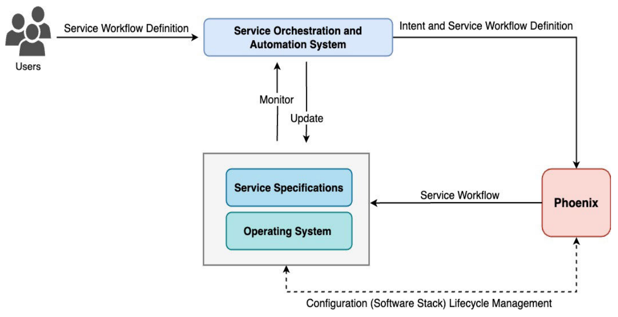
Users
Service Workflow Definition
Service Orchestration and Automation System
Intent and Service Workflow Definition
Monitor
Update
Service Specifications
Operating System
Service Workflow
Phoenix
Configuration (Software Stack) Lifecycle Management

FIGURE 18. Service orchestration and automation using Phoenix.

As shown in Figure 18, users can request an IoT service in the form of a workflow definition from a service orchestration and automation system. Such systems must typically handle the service workflow deployment on a group of separate edge IoT (i.e., service infrastructure) devices that involves provisioning of the entire device software stacks. State-of-the-art service orchestration systems typically support deployment of changes at the level of service software configurations on a pre-configured (provisioned) infrastructure [38]. Using Phoenix's transformative reconfigurability capability, IoT service orchestration systems can improve their functionality by configuring the entire software stack of infrastructure devices and applying dynamic changes and versioning of the entire service infrastructure. We note that the applicability of service reconfigurability is beyond the IoT ecosystem and can be also used in other service-oriented settings.

VII. RELATED WORK

A. BARE METAL PROVISIONING ENGINES

Bare metal provisioning engines are the closest existing systems to Phoenix [12]. Tinkerbell [11], TOSKA [39], and Foreman [40] are examples of bare metal engines that can support compatible IoT devices. Tinkerbell, for instance, can deploy workflows on PXE-compatible IoT devices [11], as discussed in section II-B. TOSKA [39] is primarily used for automating cloud services rather than provisioning of IoT devices, but by using the TOSKA Nodetype models, one can define compatible IoT device models that TOSKA can use for provisioning [39]. Foreman [40] is also capable of providing life cycle management tools for automating the provisioning of physical and virtual servers, including IoT devices [40].

Using Foreman, devices can be either specified by their MAC address or discovered via the Foreman discovery service. Once a device is known to Foreman, the provisioner can automatically orchestrate the configuration process. There are also components called Smart Proxies which provide easy ways for system developers to extend the existing subsystems and APIs supported by Foreman [40]. Such proxies can be extended with subsystems to further support PXE-compatible IoT devices. In contrast to Phoenix, none of the aforementioned systems can alone provide automated life cycle management of edge IoT devices configuration for the purpose of transformative reconfigurability. These systems mainly provide the management tools for provisioning of the IoT devices. In addition to life cycle management, Phoenix allows users to control the state and configuration of edge IoT devices using intents.

Like the aforementioned provisioning engines, Ironic [41] and MAAS [42] can be used for unattended workflow installation and provisioning of bare metal devices, including compatible IoT devices [41]. While Ironic and MAAS have different goals, together they can provide provisioning and life cycle management of any physical server [43]. A major difference, however, would be the scale of operation, as these services are better suited for large-scale solutions (e.g., cloud data centers) that can justify the cost. Phoenix, on the other hand, is well-suited for small to medium scale IoT environments. Besides, Phoenix can additionally provide configuration management of edge IoT devices.

B. AUTOMATION AND ORCHESTRATION TOOLS

In addition to bare-metal provisioning engines, there are also automation and orchestration tools and systems that are designed for unattended configuration and installation of software and application environments. While these tools and systems are specific to certain infrastructures, they can also be used for automated configuration of IoT devices (including edge devices). We note that in compare, Phoenix is agnostic to infrastructure specifications and can additionally provide: 1) life cycle management of edge IoT device configurations and, 2) an intent-driven method of supporting transformative reconfigurability. The followings are some examples of automation and orchestration tools.

Razor [44] is an open source tool for automating physical and virtual server configurations, including IoT devices. Razor is designed to work with Puppet [45] and can automatically discover and provision different Linux distributions on bare metal machines. Cobbler [46] is another powerful tool and provisioning engine for automated installation of Linux distributions. Cobbler enables a rapid installation of Linux systems and can put together many associated tasks that Linux system developers must perform in a manual installation [47]. Using Cobbler, Linux-based edge IoT devices can be automatically provisioned.

Terraform [48] and Cloud-init [49] are also well-known open-source tools for provisioning, changing, and also versioning of infrastructure configurations [50]. Unlike Phoenix which is designed to work in small to medium scale IoT environments, Terraform and Cloud-init are primarily used with large scale cloud service providers, but they can be also leveraged for automated configuration and deployment of IoT networks that can also include edge IoT devices. We must note, however, that these systems are not specifically designed for supporting IoT settings. There are also other orchestration and automation tools like Ansible [51], Chef [52], [53], and Salt [54], [55] that can be similarly used for automated configuration and deployment of digital infrastructures including IoT environments. We note that automation tools like Ansible, Chef, and Puppet must typically be used by an automation controller. In this case, Phoenix can use the services of such automation tools to enable group-based transformative reconfigurability.

C. CONFIGURABILITY IN IoT TESTBEDS

Modern IoT testbeds provide configurability support of programmable IoT devices, including the edge IoT devices. Such configurability is typically provided in two ways: 1) changing the IoT device configurations (e.g., updating its applications) but without changing the device operating system, and 2) provisioning of programmable IoT devices. For instance, FIT IoT-LAB [56] is an IoT testbed that provides a programmable environment to test and verify small wireless sensor network nodes. Using FIT IoT-LAB, IoT developers can automate workflow deployment on a variety of wireless sensor nodes that are supported by the IoT testbed. Similarly, the Poor Man's IoT testbed [57] is a well-known testbed that supports remote configuration of IoT devices. The testbed is open access and there are many IoT projects that have used the testbed for their test, debugging, and implementation efforts. TinySDR is also another IoT testbed that enables configuration of programmable IoT devices [58]. TinySDR mainly provides a Low-Power SDR testbed platform.

LinkLab is a powerful IoT testbed that can support experimentation with programmable IoT devices and facilitates Edge-Cloud integration of distributed IoT applications [59]. Link Lab can provide programming support for on-site testbed devices and enables offloading of serverless functions on programmable IoT devices. The testbed also supports a heterogeneous set of IoT devices and provides support for external users. As in IoT testbeds, there are also software systems that can support Over-the-Air Software updates for IoT devices. Mender [60] is an example that can provide a secure and reliable remote update service for IoT devices in connected IoT environments [60]. Mender's support can include both device and application levels configurability and software updates and can cover wide range of configurability features that can help automating remotely connected IoT environments [61]. JFrog [62] is a very secure and scalable software supply chain platform that can support remote software delivery to programmable IoT environments. JFrong uses JFrog Connect [63] for controlling updates and managing remotely monitored IoT devices.

In conclusion, IoT testbeds can provide powerful tools and systems that can support configuration, software updates and adjustments, and provisioning of programmable IoT devices in different sizes and scales. IoT testbeds also play a crucial role in testing and verifying real-world IoT applications deployment. As stated in section VI-C, Phoenix can empower IoT Testbeds with transformative reconfigurability that enables an IoT testbed to dynamically allocate their resources to different applications in an automated way.

D. CLOUD IoT SERVICE PROVIDERS

IoT cloud service providers also provide various tools and systems for configuration and automation of programmable IoT devices [64], [65] at different scales. These tools and systems can also support integration of IoT applications with edge and cloud resources, including data processing and machine learning systems. However, aside from scale, a major difference between Phoenix and cloud service providers is their management goal. While Phoenix enables clean-slate reconfigurability of edge IoT devices, IoT cloud service providers mainly focus on offering incremental updates or service provisioning without changing the operating system or default software stack that empowers IoT devices [64], [65]. In fact, they may also have specific preferences. In this regard, Phoenix can improve cloud IoT providers services as the Phoenix controller can be easily integrated with cloud IoT gateways.

VIII. SUMMARY AND FUTURE WORK

As cost falls and complexity of programmable IoT devices increases, reconfiguration of these devices becomes highly desirable. In this paper, we proposed Phoenix as an IoT device configuration management system for enabling automated transformative reconfiguration of edge IoT devices in small-scale IoT systems. We implemented a proof-of-concept Phoenix system and deployed it on a real research cloud testbed, called SAVI [14]. Using the Phoenix's proof-of-concept system, we evaluated various edge IoT devices performance in terms of boot time and energy consumption during the reconfiguration process in the presence of various network conditions (i.e., network packet loss). Our results indicate that Phoenix can provide consistent service quality under normal network conditions, independent of the number and heterogeneity of edge IoT devices that undergo parallel reconfiguration.

In the future, we are looking to improve this study and Phoenix capabilities in multiple ways. We are planning to extend support for wider range of reconfigurable edge IoT devices including Jetson Nano, Jetson AGXs, Asus Thinker, and Beagle Bone devices. We are also planning to expand our evaluation results to include experiments with larger number of parallel reconfigurations, using Raspberry Pi 4 devices and possibly other single board computers. The aim is to understand the performance of edge IoT devices and the consistency of Phoenix's quality at larger scales. We are also planning to perform and evaluate transformative reconfiguration for programmable router boards like Banana Pi BPI-R3. Another area of improvement is designing and implementation of the Phoenix's device discovery service. The service can enable Phoenix to recognize and add edge IoT devices to the edge device pool before they even undergo their initial configuration round (see section II-B). In terms of workflow deployment, we are also looking to improve Phoenix's support of portable edge IoT devices by improving support for QUIC [66] and Named Data Networking (NDN) [23] protocols for workflow image retrieval. Finally, we are also looking into expanding the Phoenix's intent-driven model to also support IoT service deployments in bare metal infrastructures.

IX. DISCLAIMERS

In this paper, we used multiple open source software solutions and various devices in the forms of Single Board and Standard computers to evaluate the performance of workflow deployment using Phoenix's proof-of-concept system. Although various experimental results are shared in this paper, the intention is not to compare nor to promote any performance advantage of the software solutions and the devices we used in our experiments. These experiments are solely performed for academic purposes and are designed specifically to evaluate transformative reconfigurability for different devices, network conditions, and IoT use cases using the Phoenix POC and its deployment environment. Hence, the reported results can not be used nor they are valid for benchmarking of the software solutions and the devices we used in our experiments.

REFERENCES

[1] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of Things (IoT): A literature review," *J. Comput. Commun.*, vol. 3, no. 5, p. 164, 2015.

[2] M. W. Condry and C. B. Nelson, "Using smart edge IoT devices for safer, rapid response with industry IoT control operations," *Proc. IEEE*, vol. 104, no. 5, pp. 938–946, May 2016.

[3] Q. N. Minh, V.-H. Nguyen, V. K. Quy, L. A. Ngoc, A. Chehri, and G. Jeon, "Edge computing for IoT-enabled smart grid: The future of energy," *Energies*, vol. 15, no. 17, p. 6140, Aug. 2022.

[4] V. Hayyolalam, M. Aloqaily, Ö. Özkasap, and M. Guizani, "Edge intelligence for empowering IoT-based healthcare systems," *IEEE Wireless Commun.*, vol. 28, no. 3, pp. 6–14, Jun. 2021.

[5] H. A. Alharbi and M. Aldossary, "Energy-efficient edge-fog-cloud architecture for IoT-based smart agriculture environment," *IEEE Access*, vol. 9, pp. 110480–110492, 2021.

[6] M. Chernyshev, Z. Baig, O. Bello, and S. Zeadally, "Internet of Things (IoT): Research, simulators, and testbeds," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1637–1647, Jun. 2018.

[7] J. Bauwens, P. Ruckebusch, S. Giannoulis, I. Moerman, and E. D. Poorter, "Over-the-air software updates in the Internet of Things: An overview of key principles," *IEEE Commun. Mag.*, vol. 58, no. 2, pp. 35–41, Feb. 2020.

[8] M. O. Johnston and S. Venaas, *Dynamic Host Configuration Protocol (DHCP) Options for the Intel Preboot Execution Environment (PXE)*, document RFC 4578, Nov. 2006. [Online]. Available: https://www.rfc-editor.org/info/rfc4578

[9] *iPXE—Open Source Boot Firmware*. Accessed: Jul. 30, 2023. [Online]. Available: https://ipxe.org/

[10] *Links to UEFI Specification—Related Documents | Unified Extensible Firmware Interface Forum*. Accessed: Oct. 24, 2023. [Online]. Available: https://uefi.org/uefi

[11] T Community. *Flexible Automation for Bare Metal*. Accessed: Jul. 22, 2023. [Online]. Available: https://tinkerbell.org/

[12] A. Chandrasekar and G. Gibson, "A comparative study of baremetal provisioning frameworks," Parallel Data Lab., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-PDL-14-109, 2014.

[13] *MQTT Version 5.0*. Accessed: Jul. 30, 2023. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

[14] J.-M. Kang, T. Lin, H. Bannazadeh, and A. Leon-Garcia, "Software-defined infrastructure and the SAVI testbed," in *Proc. 9th Int. ICST Conf. Testbeds Res. Infrastruct., Develop. Netw. Communities*, Guangzhou, China: Springer, 2014, pp. 3–13, doi: 10.1007/978-3-319-13326-31.

[15] R. O. Obe and L. S. Hsu, *PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database*. Sebastopol, CA, USA: O'Reilly Media, 2017.

[16] PGD Group. *PostgreSQL*. Accessed: Jul. 30, 2023. [Online]. Available: https://www.postgresql.org/

[17] *NGINX: Advanced Load Balancer, Web Server, & Reverse Proxy*. Accessed: Jul. 30, 2023. [Online]. Available: https://www.nginx.com/

[18] R. A. Light, "Mosquitto: Server and client implementation of the MQTT protocol," *J. Open Source Softw.*, vol. 2, no. 13, p. 265, May 2017.

[19] *Eclipse Mosquitto*. Accessed: Jul. 30, 2023. [Online]. Available: https://mosquitto.org/

[20] RP Ltd. *Raspberry Pi 4 Model B Specifications—Raspberry Pi*. Accessed: Oct. 20, 2023. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

[21] G. S. Malkin and A. Harkin, *TFTP Timeout Interval and Transfer Size Options*, document RFC 2349, May 1998. [Online]. Available: https://www.rfc-editor.org/info/rfc2349

[22] D. Melman, T. Mizrahi, and D. Eastlake, *Fibre Channel Over Ethernet (FCoE) Over Transparent Interconnection of Lots of Links (TRILL)*, document RFC 6847, 2013.

[23] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[24] V. Vujovic and M. Maksimovic, "Raspberry pi as a wireless sensor node: Performances and constraints," in *Proc. 37th Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, May 2014, pp. 1013–1018.

[25] *LattePanda 3 Delta—A Pocket-Sized x86 Windows Single Board Computer*. Accessed: Oct. 20, 2023. [Online]. Available: https://www.lattepanda.com/lattepanda-3-delta

[26] *Beelink M1 Mini PC review*. Accessed: Oct. 25, 2023. [Online]. Available: https://www.techradar.com/reviews/beelink-m1-mini-pc

[27] *Beelink Mini Desktop Home Page*. Accessed: Oct. 25, 2023. [Online]. Available: https://www.bee-link.com/

[28] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for Internet of Things: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 1622–1658, 3rd Quart., 2021.

[29] *Intel NUC 11 Enthusiast Kit—NUC11PHKi7C—Product Specifications | Intel*. Accessed: Oct. 20, 2023. [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/202783/intel-nuc-11-enthusiast-kit-nuc11phki7c/specifications.html

[30] Raspberry PI Ltd. *Buy a Raspberry Pi 3 Model B+*. Accessed: Oct. 25, 2023. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/

[31] F. Samie, L. Bauer, and J. Henkel, "From cloud down to things: An overview of machine learning in Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4921–4934, Jun. 2019.

[32] C. Haar and E. Buchmann, "FANE: A firewall appliance for the smart home," in *Proc. Federated Conf. Comput. Sci. Inf. Syst. (FedCSIS)*, Sep. 2019, pp. 449–458.

[33] N. Gupta, V. Naik, and S. Sengupta, "A firewall for Internet of Things," in *Proc. 9th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2017, pp. 411–412.

[34] D. Mocrii, Y. Chen, and P. Musilek, "IoT-based smart homes: A review of system architecture, software, communications, privacy and security," *Internet Things*, vols. 1–2, pp. 81–98, Sep. 2018.

[35] H. Touqeer, S. Zaman, R. Amin, M. Hussain, F. Al-Turjman, and M. Bilal, "Smart home security: Challenges, issues and solutions at different IoT layers," *J. Supercomput.*, vol. 77, no. 12, pp. 14053–14089, Dec. 2021.

[36] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *Proc. IEEE 2nd World Forum Internet Things (WF-IoT)*, Dec. 2015, pp. 459–464.

[37] *Banana Pi—A Highend Single-Board Computer*. Accessed: Oct. 23, 2023. [Online]. Available: http://www.banana-pi.com

[38] Z. Wang, M. Goudarzi, J. Aryal, and R. Buyya, "Container orchestration in edge and fog computing environments for real-time IoT applications," in *Proc. Comput. Intell. Data Anal. (ICCIDA)*. Singapore: Springer, 2022, pp. 1–21.

[39] T. Ebner, M. Meissler, and F. Steiler, "Bare-Metal Provisioning of Internet of Things Devices by Means of TOSCA," in *Proc. Wissenschaftliche Konferenz Student Conf.* Hermann Hollerith Zentrum, Böblingen: TEMMF Steiler, 2017. [Online]. Available: https://raw.githubusercontent.com/wiki/steilerDev/NOOBS4IoT/assets/TOSCA4IoT_BareMetal.pdf

[40] *Foreman*. Accessed: Jul. 30, 2023. [Online]. Available: https://www.theforeman.org/

[41] *OpenStack Ironic Bare Metal*. Accessed: Jul. 31, 2023. [Online]. Available: https://www.openstack.org/use-cases/bare-metal/

[42] *MAAS | Metal as a Service*. Accessed: Jul. 31, 2023. [Online]. Available: https://maas.io/

[43] P. Rad, A. T. Chronopoulos, P. Lama, P. Madduri, and C. Loader, "Benchmarking bare metal cloud servers for HPC applications," in *Proc. IEEE Int. Conf. Cloud Comput. Emerg. Markets (CCEM)*, Nov. 2015, pp. 153–159.

[44] PLabs. *Puppet Module Puppetlabs/Razor on Puppet Forge*. Accessed: Oct. 15, 2023. [Online]. Available: https://forge.puppet.com/modules/puppetlabs/razor/readme

[45] *Infrastructure Automation & Compliance at Enterprise Scale | Puppet by Perforce*. Accessed: Oct. 15, 2023. [Online]. Available: https://www.puppet.com/

[46] *The Cobbler Linux Installation Server*. Accessed: Oct. 15, 2023. [Online]. Available: https://www.cobblerd.org/

[47] R. M. M. Ribeiro, "Common infrastructure provisioning," Ph.D. dissertation, Dept. Inform., Universidade Minho, Porto, Portugal, 2017.

[48] Y. Brikman, *Terraform: Up and Running*. Sebastopol, CA, USA: O'Reilly Media, 2022.

[49] *GitHub—Official Upstream for the Cloud-Init: Cloud Instance Initialization*. Accessed: Oct. 15, 2023. [Online]. Available: https://github.com/canonical/cloud-init

[50] S. Jourdan and P. Pomès, *Infrastructure as Code (IAC) Cookbook*. Birmingham, U.K.: Packt, 2017.

[51] *Red Hat Ansible Automation Platform*. Accessed: Oct. 16, 2023. [Online]. Available: https://www.redhat.com/en/technologies/management/ansible

[52] *Chef Software DevOps Automation Solutions | Chef*. Accessed: Oct. 15, 2023. [Online]. Available: https://www.chef.io/

[53] M. Marschall, *Chef Infrastructure Automation Cookbook*. Birmingham, U.K.: Packt, 2015.

[54] *Saltproject.io*. Accessed: Oct. 15, 2023. [Online]. Available: https://saltproject.io/

[55] C. Sebenik and T. Hatch, *Salt Essentials: Getting Started With Automation at Scale*. Sebastopol, CA, USA: O'Reilly Media, 2015.

[56] *FIT IoT-LAB*. Accessed: Oct. 15, 2023. [Online]. Available: https://www.iot-lab.info/

[57] J. Muñoz, F. Rincon, T. Chang, X. Vilajosana, B. Vermeulen, T. Walcarius, W. van de Meerssche, and T. Watteyne, "OpenTestBed: Poor man's IoT testbed," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 467–471.

[58] M. Hessar, A. Najafi, V. Iyer, and S. Gollakota, "TinySDR: Low-power SDR platform for over-the-air programmable IoT testbeds," in *Proc. 17th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2020, pp. 1031–1046.

[59] W. Dong, B. Li, H. Li, H. Wu, K. Gong, W. Zhang, and Y. Gao, "LinkLab 2.0: A multi-tenant programmable IoT testbed for experimentation with edge-cloud integration," in *Proc. 20th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2023, pp. 1683–1699.

[60] Northern.Tech. *Open Source Over-the-Air Software Updates for Linux Devices*. Accessed: Oct. 15, 2023. [Online]. Available: https://mender.io/

[61] S. El Jaouhari and E. Bouvet, "Secure firmware over-the-air updates for IoT: Survey, challenges, and discussions," *Internet Things*, vol. 18, May 2022, Art. no. 100508.

[62] *JFrog: Software Supply Chain Platform for DevOps & Security*. Accessed: Oct. 15, 2023. [Online]. Available: https://jfrog.com/
[63] *JFrog Connect—IoT Management Platform*. Accessed: Oct. 15, 2023. [Online]. Available: https://jfrog.com/connect/
[64] R. Sikarwar, P. Yadav, and A. Dubey, "A survey on IoT enabled cloud platforms," in *Proc. IEEE 9th Int. Conf. Commun. Syst. Netw. Technol. (CSNT)*, Apr. 2020, pp. 120–124.
[65] P. P. Ray, "A survey of IoT cloud platforms," *Future Comput. Informat. J.*, vol. 1, nos. 1–2, pp. 35–46, Dec. 2016.
[66] J. Iyengar and M. Thomson, *QUIC: A UDP-Based Multiplexed and Secure Transport*, document RFC 9000, Omtermet Emgomeeromg Task Force, 2021.

MORTEZA MOGHADDASSIAN (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Toronto, Canada, in 2023. During the Ph.D. studies, he has primarily worked on name-based networking and its applications in network-level multicast. He has also been working in the areas of IoT and cloud computing. He has been a Researcher of the national smart applications on virtual infrastructures (SAVI) testbed working on network virtualization, infrastructure management, SAVI-IoT, and SAVI-ICN testbed projects. He has supervised several student projects. CampusX, an IoT-powered smart campus air quality monitoring system co-supervised by him recognized as a distinct student capstone project by the Electrical and Computer Engineering Department, University of Toronto, in 2022. He has also been working in smart grid area, where he has been mainly focused on the design of smart home platforms for optimizing the buying and selling of electricity in smart power grids. He is currently a Postdoctoral Fellow with the Network Architecture Laboratory (NAL), Electrical and Computer Engineering Department, University of Toronto. He has received Doctoral Completion Award, Edward S. Rogers Sr. Graduate Scholarship, and Edward S. Rogers Departmental Ph.D. Fellowship for his contributions and achievements. His current research interests include computer networking, distributed systems, cloud computing, the Internet of Things, and intelligent infrastructures.

SHAYAN SHAFAGHI received the B.Sc. degree in computer engineering from the Amirkabir University of Technology, Tehran, Iran, in 2021. He is currently pursuing the M.Sc. degree in electrical and computer engineering with the University of Toronto. He is also a Graduate Research Assistant with the Network Architecture Laboratory, University of Toronto. His research interests include distributed systems, cloud computing, computer networks, and operating systems.

POOYAN HABIBI (Member, IEEE) received the bachelor's degree in computer engineering and the master's degree in computer network engineering from the Amirkabir University of Technology, Tehran, Iran. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Toronto, Canada. He is also a Graduate Research Assistant with the Network Architecture Laboratory (NAL), University of Toronto. During the Ph.D. degree, he has been mainly working on cloud-native orchestration and automation to improve the life cycle management of cloud native applications on heterogeneous infrastructures. He has been a Researcher of the national smart applications on virtual infrastructures (SAVI) testbed project focusing on network virtualization, SAVI-IoT, and the next generation cloud native SAVI testbed. He has also been working on the design and development of cloud native orchestration platforms for multi-FPGA VNF systems. His research interests include cloud computing, distributed systems, and computer networks. He has been granted the Edward S. Rogers Departmental Ph.D. Fellowship and holds several prestigious IT certifications, such as CCDE, CKA, CKAD, Azure Solutions Architect Expert, and AWS Certified Cloud Solutions Architect Professional.

ALBERTO LEON-GARCIA (Life Fellow, IEEE) is currently a Professor in electrical and computer engineering with the University of Toronto. He was the Founder and the CTO of AcceLight Networks, Ottawa, from 1999 to 2002. AcceLight developed a multi-service switch-router with all-optical fabric. He was the Scientific Director of the NSERC Strategic Network for Smart Applications on Virtual Infrastructures (SAVI) and a Principal Investigator of the Project on Connected Vehicles and Smart Transportation. SAVI designed and deployed a national testbed in Canada that converges cloud computing and software-defined networking. CVST designed and deployed an application platform for smart transportation. He was the Founder of StreamWorx.ai which developed massive-scale, real-time streaming analytics software for network operations, and cybersecurity applications. StreamWorx was acquired and is now part of OpenText. He has authored the textbooks: *Probability and Random Processes for Electrical Engineering* and *Communication Networks: Fundamental Concepts and Key Architectures*. His current research interest includes the design of intelligent automated management systems for large-scale infrastructures. He is a fellow of the Royal Society of Canada and a Life Fellow of the Institute of Electronics and Electrical Engineering "For contributions to multiplexing and switching of integrated services traffic." He has held the several chair positions with the University of Toronto: the Nortel Chair of Network Architecture and Services, the Skoll Chair of Computer Networks and Innovation, the Canada Research Chair of Autonomic Service Architecture, and a Distinguished Professor in smart infrastructures.