2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid) | 978-1-7281-9586-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CCGrid51090.2021.00083

# VeerEdge: Towards an Edge-Centric IoT Gateway

Udhaya Kumar Dayalan, Rostand A. K. Fezeu, Nitin Varyani, Timothy J. Salo, Zhi-Li Zhang
Department of Computer Science, University of Minnesota - Twin Cities
{dayal007,fezeu001,varya001,salox049,zhang089}@umn.edu

***Abstract*—As the plethora of Internet of Things (IoT) devices gradually make their way into our lives, several Cloud Service Providers (CSPs) have developed *IoT gateway* platforms (SDKs) that solely connects IoT devices to their respective cloud. Such gateways have 1) cumbersome IoT device configuration; 2) inflexible IoT data managements; and 3) support no/little cross-vendor edge computation and cloud analytics. We term these commercial gateway SDKs as *cloud-centric*. In this paper, we study the state-of-the-art vendor-locked IoT Gateway solutions and approaches and propose an *edge-centric* paradigm through an evolutionary framework, dubbed *VeerEdge* for developing *IoT gateways*. We leverage computing and storage capabilities at the network *edge* for edge-based device & IoT service management and data processing. We exploit availability of multiple cloud services for "best" IoT data analytics. Evaluation results show that *VeerEdge* achieves this with negligible overhead in terms of latency, CPU and RAM usage when compared to state-of-the-art industrial *IoT gateways*.**

***Index Terms*—IoT devices, IoT Gateway SDK, IoT cloud.**

## I. Introduction

Many Internet of Things (IoT) systems are "stovepipe systems": they are closed, end-to-end, sensor-device-to-cloud-application systems that operate independently of each other. These stovepipe systems are unable to interact directly with each other, or share most resources. In this paper, we describe a new class of IoT gateways that is intended to break down the barriers between these stovepipes, thereby permitting these systems to share resources, particularly the applications that process and store the sensor data. As a result, these new IoT gateways proposed here will permit the processing of sensor data to be consolidated and optionally distributed or moved closer to the IoT devices themselves.

A common example of these stovepipe IoT systems, and the challenges that these systems often present, is a home that contains IoT devices from multiple vendors, such as smart speakers. For instance, a homeowner might connect both a Google Nest Mini and an Amazon Echo Dot to their home network. Each of these IoT devices is a closed system, and is unable to share resources, subsystems, or procedures with devices from other vendors. Each IoT device is managed by its own smartphone application and communicates with its own cloud-based application. The user must learn and use two different applications to configure and manage the smart speakers. More importantly, because the data from the smart speakers is forwarded to their respective applications, it is difficult for a single application to process the consolidated data from all of the devices simultaneously, perhaps correlating or fusing data from these separate devices. Likewise, processing the data from the smart speakers locally, or at the "edge" of the home network, is extremely difficult, because the data largely resides in the vendors' cloud-based applications.

Several major cloud service providers (CSPs), including Amazon, Microsoft, and Google, have made available IoT gateway frameworks, or software development kits (SDKs), that simplify the development of IoT devices. IoT device vendors can use these CSP-provided SDKs to simplify the development of their IoT systems: the SDKs can be used as a platform upon which to develop software that connects the vendor's IoT device to a cloud-based application. Unfortunately, each of these CSP-provided SDKs connect only to the respective vendor's cloud service. For our purposes, we call these CSP-provided IoT gateway platforms as *cloud-centric*, inasmuch as they connect IoT devices *only* to the cloud services of that CSP.

We propose an *edge-centric* model for developing IoT gateways. Instead of merely connecting IoT devices to cloud services, our *edge-centric* IoT gateway framework is designed to i) leverage computing and storage capabilities at the network *edge* (e.g., a Raspberry Pi device or a PC server collocated at a home Internet gateway or wireless base station) for edge-based device and IoT service management (e.g., fault detection, dynamic service subscription), data processing (e.g., data filtering & aggregation), and so forth; and ii) exploit availability of multiple cloud services (from different vendors) for "best" (e.g., fastest or cheapest) IoT data analytics. We summarize the outline and major contributions below.

- (Sec. II) We study three leading CSPs IoT solutions to ensure our proposed framework augments current IoT gateway solutions . We especially evaluate similarities and differences between them to identify north-bound (cloud facing) and south-bound (on-premise IoT gateway facing) interfaces that can be leveraged within our proposed gateway framework.
- (Sec. IV) We propose *VeerEdge* - an edge-centric IoT gateway framework, that exploits the availability of multiple cloud services, storage and computing capabilities on the network for edge-based device management and configurations and "best" IoT data analytics.
- (Sec. V) We investigate a critical IoT gateway functionality dubbed - *Regulator* that realizes the edge-centric gateway vision by controlling the communication between vendor-specific IoT gateways and their respective cloud services. Proof-of-concept prototype using Amazon AWS and Microsoft Azure as case studies show that (*VeerEdge)* incurs additional negligible overhead and minimal latency.

978-1-7281-9586-5/21/$31.00 ©2021 IEEE
DOI 10.1109/CCGrid51090.2021.00083

## II. Terminology and Background

In this paper, we use the term "IoT edge device", or simply "IoT device", to describe the end nodes in IoT systems, specifically the components that include sensors or actuators. These are the devices that generate IoT sensor data, or make changes in the physical world in response to commands. For the purposes of this paper, we classify IoT devices into two categories:

1) "cloud-native" devices: IoT devices that are able to connect directly to applications running on a cloud service using media such as Wi-Fi or cellular service [1]. Typically, cloud-native devices are "locked" to a specific application running on a particular cloud service.

2) "gateway-assisted" devices: IoT devices that are incapable of connecting directly to an application running on a cloud service, and therefore require the services of an IoT gateway to forward sensor data to a remote application for processing. Gateway-assisted IoT devices usually connect to an IoT gateway via a low-power wireless medium such as Bluetooth, Zigbee, Z-Wave, Thread, or similar protocol because they lack the functionality to necessary communicate directly with cloud-based applications. Examples of these gateway-assisted devices include: door or window sensors, temperature sensors, or water sensors.

The data generated by the IoT devices is communicated to its end users using a pub-sub system. A pub-sub system is an asynchronous way of communicating between entities where a subscriber of a topic receives all the messages published to that topic. Amazon refers their IoT pub-sub system as *message subscriptions* [7] while Azure names it as *routes* [8]. In this paper, we will consistently use the term *message subscriptions* or *paths* irrespective of the vendors. There are three fields required for a message subscription [7]. First, the *source*, from where the message originated. Next, the *destination*, to which the message needs to be sent. And finally, the *topic* to which one can subscribe to or publish message to.

"IoT portal/cloud" is the entry-point on the cloud. It is tied to and authenticates only specific vendors' devices and gateways and is largely responsible for heavy analysis(computation), deployment, notifications and updates. IoT gateways manage and provide connectivity to cloud-based applications for gateway-assisted IoT devices. Common consumer-grade examples of IoT gateways include smart home gateways or home automation gateways, such as the Samsung SmartThings hub. These devices implement the protocols necessary to communicate with a cloud service and applications running on that cloud service. Typically, IoT gateways implement a light-weight messaging protocol, such as MQTT or CoAP, which manages the transfer of sensor data and commands between the IoT device and a cloud-based application.

An "Edge function" run as containers [9] in the IoT Gateway which is managed by the IoT Gateway's run-time. Container is a unit of software that contains code and all its dependencies (run-time, system tools, system libraries and settings) as a single package. In Azure IoT, the containers packaged with custom code are called modules. And in AWS IoT, these containers are called as lambda functions [6]. Additionally, in AWS, the lambda functions can run as an individual process in the IoT Gateway instead of a container. As shown in Figure 2, local database, web-server, machine learning services are some of the examples for a edge function.

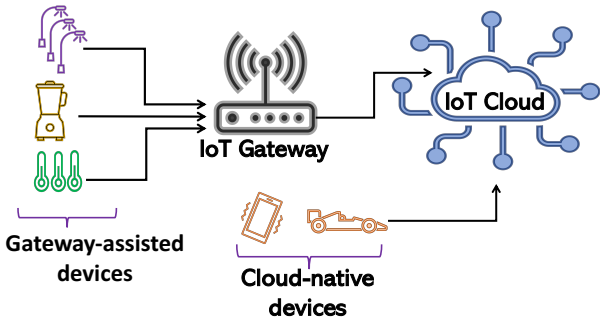


Fig. 1. Cloud-native and gateway-assisted IoT devices

Major cloud service providers offer what we refer to as "IoT cloud services", specialized services that support IoT devices and IoT gateways. For example, these IoT cloud services generally support one or more common IoT messaging protocols, such as MQTT or CoAP. While terminology differs among cloud vendors, we refer to these IoT-specific services as "IoT portals". Cloud-hosted applications process and store IoT sensor data, initiate notifications in response to IoT sensor data, and manage IoT devices and users.

### A. Cloud-Centric IoT Gateways

Recently, several CSPs, including Google [2], Amazon Web Services (AWS) [10] and Azure [11] have made available IoT gateway platforms, or SDKs, that IoT vendors may integrate into their IoT devices or gateways. These CSP-provided SDKs simplify the development of IoT devices and gateways, but at the expense of locking the vendors into the SDKs' respective cloud services. We summarize their similarities and differences in Table I and briefly discuss them here.

*1) **AWS IoT Core**:* Amazon's IoT Gateway SDK is called "Greengrass" (GG) [12]. One major feature of Greengrass is the runtime. GG's runtime serves both as a client to the AWS cloud and a server to "gateway-assisted" AWS devices to marshal data and enable bi-directional communication between these entities. GG's runtime is equipped with a modified Paho-based MQTT 3.1.1 implementation over TLS 1.2 encryption (MQTT over Websocket) with X.509 certificate-based mutual authentication [1] [2]. The runtime also offers support for Lambda functions – a server-less compute service to run code in response to an event [10]. This functionality may simplify

[1]While we often describe applications as running on a cloud service, they could, in fact, be running on any sort of server.

[2]At the time of writing this paper, GG V1 included MQTT QoS 0, "fire and forgot" which does not require any acknowledgement and QoS 1, "fire and wait for acknowledgement" ensures an acknowledgement is received.

TABLE I
FEATURES SUPPORTED IN CURRENT VENDOR-LOCKED IOT GATEWAYS

| Features | AWS | Azure | Google |
|---|---|---|---|
| Protocols | Modified Paho MQTT 3.1.1 (QoS 0 & QoS) 1 HTTP[S][1] | MQTT 3.1.1 HTTP[S] 1.1 over TLS 1.2 AMQP | Paho MQTT 3.1.1 QoS 0 & QoS 1 HTTP[S] [2], [3], [4] |
| Security | X.509 CA Signed X.509 Self-Signed certificates [1] | X.509 CA Signed X.509 Self-Signed certificates Symmetric keys | JSON Web Tokens [5] |
| Containerization Support | ✓ | ✓ | × |
| Message Subscriptions | ✓ | ✓ | × |
| Stream Manager | ✓ | × | × |
| Device Twins/Shadow | ✓ | ✓ | × |
| On-demand Containerization [6] | ✓ | × | × |
| Device Monitoring | × | ✓ | × |

events-response and control within IoT applications. The GG's runtime maintains IoT device state information using "Device shadows" via a JSON serialization format [1] which simplifies management for mobility support. Additionally, the runtime performs data aggregation, queuing and scheduling before forwarding IoT data to the cloud.

*2) **Microsoft Azure IoT**:* Microsoft Azure IoT gateway SDK is also equipped with a customized runtime. At this time, the runtime relies on a broker to communicate with the cloud. This broker can be configured with either MQTT 3.1.1, AMQP or HTTP 1.1 protocols secured with TLS 1.2. and token-based authentication [13]. Azure provides a simplified version of Lambda functions called "modules" – to run code based on a trigger [8]. Mobility support is provided via "device twins" – a customized but different [3] JSON serialization format [14] to maintain IoT device state information.

*3) **Google IoT Core**:* Googles' gateway SDK is an embedded-device SDK. It supports HTTP 1.1 or a custom Paho-based MQTT 3.1.1 protocol with TLS 1.2 with JSON Web Tokens (JWTs) [15] for authentication [2], [3], [4]. Google's gateway does not have a runtime. However, device state information is maintained using "device metadata" (maximum size of 256 KB), "device configuration" (maximum size of 64 KB) and "device state" (maximum size of 64 KB) [16]. Unlike Azure and AWS that only support JSON serialization format, Google also supports binary, text or serialized protocol buffers data formats [16].

We acknowledge that these vendor gateway SDKs render significant contributions within the IoT systems. Nonetheless, they all embrace a cloud-centric approach and still posses major drawbacks. Next, we describe their drawbacks and then make a case for an edge-centric approach for IoT gateways.

## III. CHALLENGES WITH CLOUD-CENTRIC IOT GATEWAYS

Due to the recent surge of IoT devices, several IoT applications have been deployed in the cloud to perform computation on the IoT data. Limitations like latency perceived by end-systems, and increased bandwidth usage between the end-systems and the cloud ought to move IoT data computation towards the edge. However, key-players like Microsoft, Amazon, and Google only allow the management of IoT data on the cloud. This in turn makes the management of IoT data that utilizes cloud-based applications from different vendors, cumbersome. In this section, we explain in detail these challenges posed by cloud-centric IoT gateways. As a results, presents a unique opportunity to advocate an edge-centric IoT gateway in order to unlock the benefits of various IoT applications provided by different vendors and to also enable convenient management of IoT data.

[3]When compared with AWS device shadows.

### A. Cloud-Centric IoT Gateways: Issues

To marshal IoT data in current industrial IoT solutions, *message subscriptions* needs to be configured in the cloud and then deployed in the IoT gateways. The data from IoT devices is then routed through the IoT gateway to the relevant users based on these subscriptions. This, however, poses several challenges in developing innovative and rich IoT solutions, as discussed next.

**Cumbersome cloud-based IoT device configuration.** In-order to disable or enable communication of IoT devices with the cloud, we need to completely remove or re-add the paths that exist in the cloud and redeploy them on the IoT gateway. In AWS and Azure, the paths need to be configured to enable communication between IoT devices, edge functions and the IoT cloud.

**Inflexible cloud-only IoT data management.** Cloud-only management of IoT data makes its management very difficult. To utilize applications from different CSPs' clouds, we need multiple IoT gateways and also need to configure several paths on each cloud portal. Such configurations in the cloud portal incurs higher latency since the cloud is generally far away from the end-users. Moreover, these paths have to be deployed in the IoT gateway after re-configuration in the cloud, thus, adding up to this latency. For example, if a building already have a Azure IoT gateway, in order to send videos data from an IP camera to AWS Kinetic Streams, an AWS GG IoT gateway need to be installed and configured in the building. Management of multiple IoT gateways is time consuming and challenging due to maintainability.

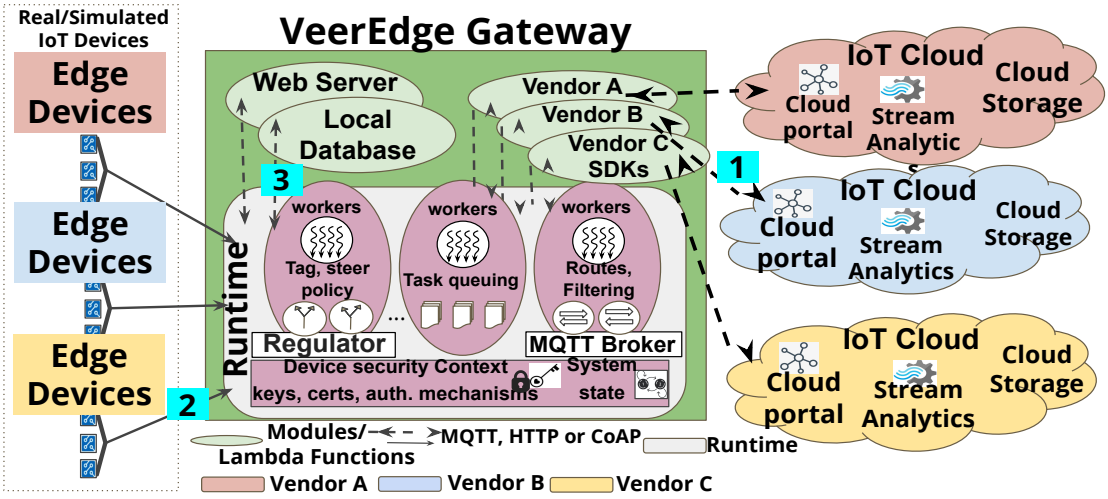


Fig. 2. Edge-Centric IoT Gateway Framework

**No/little support for cross-vendor edge computation and data analytics.** Vendor IoT gateways use the paths deployed in it to simply forward the IoT data to vendor-specific applications hosted in the cloud. The inability to configure these paths at the edge prevents IoT data to be dynamically routed to other CSPs' cloud or edge. Leveraging "better" stream analytics and machine learning application of another vendor is extremely difficult and impractical.

### B. Case for an edge-centric IoT gateway

The challenges identified in the previous subsection leads us to advocate an *edge-centric* architecture for designing IoT gateways. Instead of merely connecting IoT devices to cloud services, we envision an *edge-centric* IoT gateway that i) leverages computing and storage capabilities at the network for edge-based device management, configuration and control, and ii) exploits the availability of multiple cloud services (from different vendors) for "best" (e.g., fastest or cheapest) IoT data analytic. An *edge-centric* IoT gateway ought to enable sending IoT data to clouds of different vendors instead of locking it to a specific vendor. Moreover, Edge-centric IoT gateways also make IoT applications less prone to security attacks since there is more privacy compared to in the cloud as the data resides locally. We achieve this by introducing *regulator*. *Regulator* is a subsystem built atop existing vendor IoT gateway SDKs and enables flexible device configuration and data management, dynamic cloud service subscriptions and message routing. We provide a detailed description of *regulator* later.

## IV. How to address these challenges?

Given the challenges mentioned earlier, in this section, we discuss various solution approaches and highlight their limitations. We, then present our *VeerEdge* IoT gateway architecture design.

### A. Re-designing the IoT Gateway

One approach to address these challenges might be to re-design an IoT gateway from scratch. This gateway should be open and not tied to any specific CSP IoT cloud portal. Rather, it should provide multi-cloud support to connect, communicate and exchange data with several vendor IoT cloud portals while still enabling local configurations. This approach replaces the current CSPs' IoT gateway solutions and imposes a unified ontology or a consensus of the communication protocols and RESTFul APIs adopted. According to the European project Unify-IoT, more than 300 IoT cloud platforms exist today [17]. Therefore, an obvious problem with this approach is its inability to work with existing IoT cloud portals, *i.e.,* CSPs may need to re-design their IoT cloud portals to add support for the unified APIs and protocols. This is impractical, time consuming and might be less beneficial for some vendors. Thus, several vendors might be reluctant to proceed with an agreement.

### B. Building atop current IoT solutions

In this study, we take a different approach. Instead of re-designing an IoT gateway from scratch, we build atop existing IoT gateways and only leverage their runtimes. We propose a wrapper dubbed, *Regulator* which leverages vendor gateway SDKs to connect to their respective IoT cloud portals. Specifically, this approach augments current systems' runtimes and take control of all communication happening between the different vendor IoT gateways and their clouds portals.

The limitations with our approach are two folds. 1) There is no unified way for the IoT devices to communication with the IoT gateway. Each vendor will still advocate its unique security mechanisms for IoT device to authenticate with the IoT gateway. 2) This solution is limited to "gateway-assisted" IoT devices. Nonetheless, in this study, we investigate this approach and augment current IoT gateway SDKs enabling multi-cloud support, edge-computation, dynamic manageability using payload information within *Regulator*, while still using the cloud.

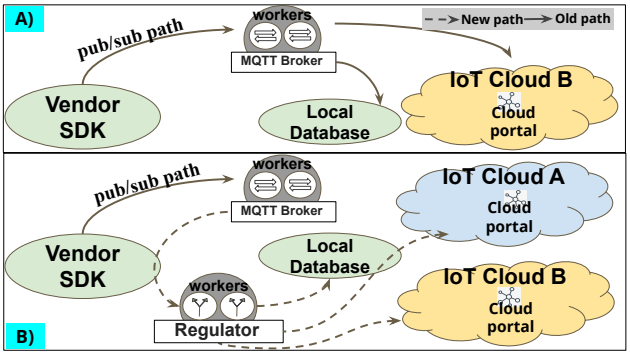


Fig. 3. *Regulator* operation scenario.

### C. Proposed VeerEdge Gateway Design

Fig. 2 summarizes the detailed architecture of *VeerEdge*. We discuss the major components of *VeerEdge* below.

*1)* ***Runtime****:* In our design, the runtime does the heavy work. We build atop existing IoT gateways by leveraging existing vendor gateway runtimes. AWS GG and Azure IoT gateway runtimes come pre-build with a task scheduler, and an MQTT Broker. We therefore use both runtimes during our implementation and show evaluation results in Sec. V. We augment their runtimes with *Regulator* - edge function.

*2)* ***IoT gateway SDKs****:* Within our edge-centric IoT gateway, we run Google, Azure and AWS gateway SDKs [18], [11], [12] as edge functions. They provide the RESTFul APIs to communicate with their IoT cloud portal. However, as describe later in sec. IV-C3, *Regulator* controls all paths in our design bringing IoT data closer to the edge. We host a local database for data storage and a web server for local configurations and management, alleviating the cloud-centric management.

*3)* ***Regulator****: Regulator* is primarily controlled by user configuration via the local webserver. Specific configuration options like ”disable publish to cloud”, ”delete path x” and ”create path y” can be configured. We consider ”disable publish to cloud” in Sec. V within *regulator* during our implementation. This is because, our approach here involves creating and deleting (temporarily disabling) paths. Traditionally, performing this function (”disable publish to cloud”) requires, manually deleting and re-deploying the configuration locally on the gateway on premise. In *VeerEdge*, the webserver performs a runtime interrupt via *regulator*. (i) *Regulator* leverages the runtimes’ exposed APIs to discover the current static paths (source, destination, topics) pairs. (ii) It creates (if it does not exist) a new ”shared topic” and subscriber (usually the local database) on the system and (iii) temporally disables the cloud facing path and redirects every packet via the new ”topic” (path). This simple operation is summarized in Fig. 3.

By means of this functionality, *Regulator* can, 1) dynamically operate on all vendor IoT Platform SDKs edge functions deployed on the system, 2) avoid downtime that exist when redeploying new cloud configurations and 3) seamlessly reduce unconnected ”stovepipes” between vendors, thus interoperability. However, for the first time, the path need to be configured in the IoT cloud and deployed to the IoT gateway. And the *regulator* controls the paths thereafter.

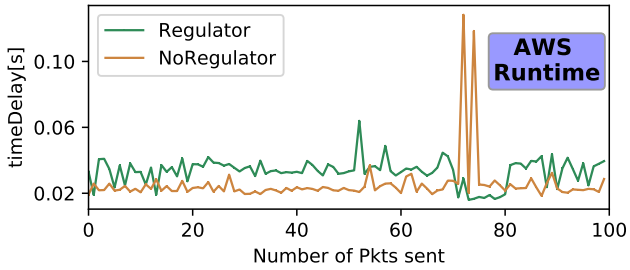


Fig. 4. AWS runtime time delay - Jetson Nano

## V. Implementation and Evaluation

In this section, we discuss the implementation and evaluation results of our proposed design.

### A. Implementation

We first implemented *VeerEdge* on a Raspberry Pi2 and show preliminary implementation results in [19]. Next, we extended our evaluations and implemented *VeerEdge* on an NVIDIA Jetson Nano. The web-server is used for local configuration of paths, the local database stored IoT data locally and Vendor A, B and C SDKs are Google’s, Azure’s and AWS’ Gateway SDKs equipped with all security context for authenticating and communicating with their clouds portals. *Regulator* leverages these SDKs to control and steer traffic based on the paths configured. At start-up, *regulator* takes over the paths and controls the traffic based on the local user configuration. Through this implementation, we were able to enhance the existing IoT Gateway frameworks to support local enable or disable of the message subscriptions without using the cloud portal. Since the paths already exists, the communication between two entities pass through without any issues and our implementation controls the pass-through only to disable or alter the communication. Through this approach, we have eliminated cloud-based configuration and an additional deployment. Additionally, *regulator* enables the local control of paths between vendor-specific IoT Gateway and multi-vendor IoT clouds, which is not supported with current vendor Gateways.

### B. Evaluation

First, we seek to quantify the additional delay incurred by processing every packets via *regulator*. We simulated IoT devices to publish data to our universal gateway and logged the time when every packet was sent. We configured a path to route every packet to the local database, where we logged the time every packet was received. We repeated this experience with and without *regulator*. In Fig. 4, we show the additional delay incurred with *regulator* (the blue curve) and without *regulator* (the red curve) leveraging the AWS GG runtime. Noticed that, *regulator* incurs negligible overhead

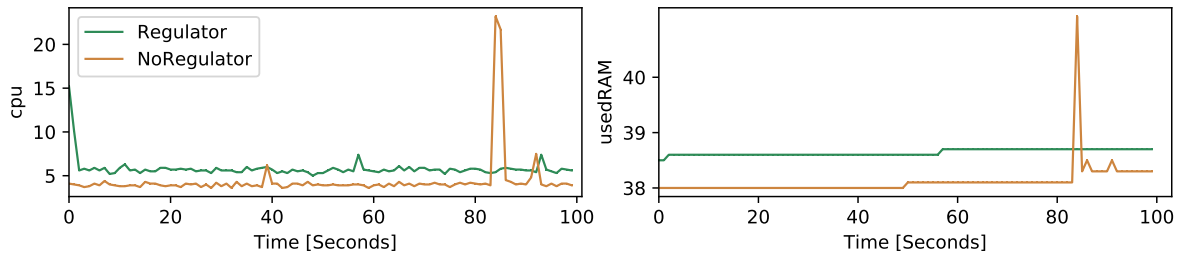


Fig. 5. AWS runtime memory and cpu utilization - Jetson Nano

while addressing interoperability challenges in current CSPs' IoT platforms.

Next, we collect the CPU and memory utilization with and without *regulator* to understanding the additional overhead incurred with our approach. As shown in Fig 5, augmenting both AWS GG runtime with regulator result in more CPU and memory usage as expected. This is because, all packets through the gateway are processed by regulator, i.e., the payload needs to be matched to paths deployed on the system before routing. These results show this approach incurs additional negligible overhead and minimal latency. It is important to mention that, we acknowledge that, the additional overhead incurred by this approach can be problematic in low latency IoT application scenarios. Nonetheless, the results are promising.

## VI. Conclusions and Future Work

In this study, we make the case to advocate a shift from a cloud-centric to an edge-centric approach in IoT gateways. We proposed *regulator*, which augments current vendor-locked IoT platform solutions by controlling paths from various vendor gateway SDKs to the cloud. We evaluated our methods by using both AWS and Azure runtimes. Our experiments show that our approach incurs negligible overhead and minimal latency. Although we developed an approach to support interoperability from the IoT Gateway to the cloud, unlocking multi-vendor downstream IoT devices to vendor IoT Gateway still remains a challenge, especially since vendors adopt custom security mechanisms in their IoT devices, gate SDKs and IoT cloud. Moreover, there is no standard message subscriptions framework followed by the CSPs: AWS uses MQTT topics and Azure uses endpoints. This challenge is left for future works. Our experiments clearly show that the resource consumed by the Azure IoT Gateway framework is relatively higher than the AWS Greengrass. Thus, further exploration of both frameworks to understand why can be a possible research direction. In summary, suggested future research direction can be; 1) addressing the interoperability issues with vendor IoT edge devices and 2) building an IoT gateway runtime that supports heavy edge computational tasks and connects to multiple vendor cloud platforms.

## VII. Acknowledgement

The research was supported in part by NSF under Grants CNS-1618339, CNS-1617729, CNS-1814322, CNS-1836722 and CNS-1901103.

## References

[1] A. IoT, "Aws iot developer guide," pp. 428 – 472, 2020. [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/iot-dg.pdf#iot-device-shadows

[2] G. C. , "Overview of internet of things — solutions — google cloud," Google Cloud, 02 2019. [Online]. Available: https://cloud.google.com/solutions/iot-overview

[3] G. Cloud, "Publishing over the http bridge — cloud iot core documentation," Google Cloud, 2019. [Online]. Available: https://cloud.google.com/iot/docs/how-tos/http-bridge

[4] ——, "Publishing over the mqtt bridge — cloud iot core documentation," Google Cloud, 2019. [Online]. Available: https://cloud.google.com/iot/docs/how-tos/mqtt-bridge

[5] G. IoT Core, "Using json web tokens (jwts) — cloud iot core documentation," Google Cloud, 06 2020. [Online]. Available: https://cloud.google.com/iot/docs/how-tos/credentials/jwts

[6] A. AWS, "Aws lambda – product features," Amazon Web Services, Inc., 2019. [Online]. Available: https://aws.amazon.com/lambda/features/

[7] AWS, "Configure devices and subscriptions," Amazon. [Online]. Available: https://docs.aws.amazon.com/greengrass/latest/developerguide/config-dev-subs.html

[8] Microsoft, "Deploy azure iot edge modules from the azure portal," Microsoft. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/how-to-deploy-modules-portal

[9] Docker, "What is a container?" Docker. [Online]. Available: https://www.docker.com/resources/what-container

[10] AWS, "Run lambda functions on the aws iot greengrass core - aws iot greengrass," docs.aws.amazon.com, 11 2018. [Online]. Available: https://docs.aws.amazon.com/greengrass/latest/developerguide/lambda-functions.html

[11] Microsoft, "Configure an iot edge device to act as a transparent gateway," Microsoft. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/how-to-create-transparent-gateway

[12] AWS, "Machine learning inference with aws iot greengrass solution accelerator," Amazon Web Services, Inc., 10 2019. [Online]. Available: https://aws.amazon.com/iot/solutions/mli-accelerator/

[13] M. kgremban, "Learn how the runtime manages devices - azure iot edge," docs.microsoft.com, 11 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-edge/iot-edge-runtime

[14] Microsoft, "Understand azure iot hub device twins," docs.microsoft.com, February 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-device-twins

[15] M. Jones, J. Bradley, and N. Sakimura, "Json web token (jwt)." [Online]. Available: https://tools.ietf.org/html/rfc7519

[16] G. Cloud, "Devices, configuration, and state — cloud iot core documentation," Google Cloud, 06 2020. [Online]. Available: https://cloud.google.com/iot/docs/concepts/devices

[17] U.-I. Project, "Deliverable d03.01, report on iot platform activities," 2016. [Online]. Available: https://docbox.etsi.org/SmartM2M/Open/AIOTI/IoTPlatformsAnalysisToImprove/D03_01_WP03_H2020_UNIFY-IoT_Final.pdf

[18] Google, "Using gateways," Google. [Online]. Available: https://cloud.google.com/iot/docs/how-tos/gateway

[19] U. K. Dayalan, R. A. Fezeu, N. Varyani, T. J. Salo, and Z.-L. Zhang, "Eciot: Case for an edge-centric iot gateway," in *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, 2021, pp. 154–156.