# SegaNet: An Advanced IoT Cloud Gateway for Performant and Priority-Oriented Message Delivery

Yeonho Yoo
Korea University, Microsoft Research

Zhixiong Niu
Microsoft Research

Chuck Yoo
Korea University

Peng Cheng
Microsoft Research

Yongqiang Xiong
Microsoft Research

## ABSTRACT

With the tremendous growth of IoT, the role of IoT cloud gateways in facilitating communication between IoT devices and the cloud has become more important than ever before. Most previous studies have focused on developing interoperability between IoT and cloud to accommodate various radio protocols. However, they have often neglected the performance aspect of the IoT cloud gateway, leaving users with limited options: either purchasing multiple gateways or connecting only a small number of IoT devices. Through our comprehensive measurements and analysis, we identified five key issues in IoT cloud gateways related to high latency, CPU bottlenecks, inefficient network stacks on ARM, substantial encryption overhead, and the lack of priority support. To address these issues, we propose a new IoT cloud gateway - SegaNet. We carefully design with 1) multiple agents management, 2) efficient TLS encryption, and 3) priority-oriented message delivery. Our prototype evaluation shows up to 16.7× lower latency and 4.5× lower CPU consumption than gateways of the existing IoT-cloud ecosystem.

## CCS CONCEPTS

• **Networks → Application layer protocols**; • **Hardware → Sensor applications and deployments**.

## KEYWORDS

Internet of Things, Cloud, IoT cloud gateway, TLS, Message protocol

**ACM Reference Format:**
Yeonho Yoo, Zhixiong Niu, Chuck Yoo, Peng Cheng, and Yongqiang Xiong. 2023. SegaNet: An Advanced IoT Cloud Gateway for Performant and Priority-Oriented Message Delivery. In *7th Asia-Pacific Workshop on Networking (APNET 2023), June 29–30, 2023, Hong Kong, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3600061.3600072

## 1 INTRODUCTION

In recent years, the Internet of Things (IoT) has played a vital role in propelling the progress of smart industries [6], smart homes [40], and smart factories [7]. As reported by Statista [39] and IDC [19], the number of IoT devices reached 11.28 billion by 2021, and this figure is projected to rise to 29.43 billion by 2030. Moreover, the volume of data generated by IoT devices is anticipated to reach 73.1 zettabytes by 2025. To manage the vast number of devices and data, public cloud platforms like Azure [29] and AWS [2] have introduced IoT solutions for device management, data monitoring, aggregation, storage, and business intelligence, incorporating AI. Typically, users rely on cloud services to manage IoT devices and collect data from them via the Internet.

However, typical IoT devices are highly energy-efficient and compact in size, generally utilizing radio protocols engineered for low-power and lossy networks (LLNs) [6, 36, 43], and thus cannot directly access the Internet. A prevalent solution is the IoT cloud gateway, an intermediary device that connects IoT devices with the cloud [4]. These gateways run IoT applications (referred to as agents) and execute various functions on behalf of both IoT devices and the cloud. As each device possesses unique characteristics, it is typically more effective to run separate agents. However, given that IoT cloud gateways are often machines with resource-constrained specifications, increasing the number of agents presents a significant challenge.

Numerous studies have approached the development of connectivity between IoT and cloud from various perspectives, and they can also contribute to the advanced IoT cloud gateway. For instance, TinyNet [11] facilitates interoperability between radio protocols and TCP/IP. Gatescatter [22] utilizes backscatter technology to modulate radio signals into Wi-Fi signals for internet connectivity. Safronov et al [36] proposed gateways to provide application interoperation and break away from the existing Internet protocol. LoRaX [44] allows gateways to use both high-bandwidth and low-data rate networks to provide wide Internet access. Despite these advancements, there remains a scarcity of research specifically addressing how IoT cloud gateways can effectively interconnect a large number of IoT devices and cloud platforms within an IoT-cloud ecosystem.

As the number of IoT devices in our surroundings continues to grow significantly, their performance becomes increasingly critical. With our extensive measurements, we identify five key issues regarding the performance of IoT cloud gateway. First, IoT cloud gateway suffers high latency when handling a large number of IoT devices (§3.1). Second, IoT cloud gateway undergoes CPU bottlenecks when delivering messages from a large number of devices (§3.2). Specifically, as the number of IoT devices increases from 1 to 256, the message latency and CPU usage grow up to 1255× and 19.6×, respectively. Third, performance degradation is attributed to the lack of consideration for architectural differences (§3.3). Fourth, data encryption overhead for communication with the cloud is significantly high (§3.4). Lastly, Depending only on the existing message protocol, priority-based message delivery is not guaranteed (§3.5).

To address these challenges,we introduce SegaNet,a new IoT cloud gateway designed to enhance the scalability of cloud-connected IoT devices on a large scale.SegaNet employs three design decisions to resolve the aforementioned issues. First, to alleviate network stack overhead on ARM, SegaNet assigns multiple agents that transmit each IoT device's messages to be mapped to a designated core (§4.1). Second, to reduce excessive encryption overhead, message batching is performed (§4.2). Finally, SegaNet presents a method for prioritizing message delivery, which is not provided by existing message protocols (§4.3).

Our prototype of SegaNet is implemented with key components, reception manager and SegaNet agent, and operating based on MQTT. Our prototype evaluation is conducted on the Raspberry Pi 4, revealing up to 16.7× faster latency and 4.5× lower CPU usage compared to existing IoT cloud gateways when delivering messages from numerous devices. SegaNet also demonstrates the ability to prioritize message delivery while ensuring seamless compatibility with existing message protocols such as MQTT.

## 2 BACKGROUND AND CHALLENGES

In this section, we will examine the significance of IoT gateways, review current IoT gateway solutions, and explore the challenges that users are facing.

### 2.1 Connectivity between IoT and cloud

At present, IoT devices mainly rely on radio protocols engineered for LLNs, such as IEEE 802.15.4, Bluetooth Low Energy (BLE), and LoRa, to achieve remarkably small sizes and exceptional energy efficiency. The public cloud acts as a platform for supplementary data storage, processing, and business intelligence. Furthermore, it can transmit alerts to users in response to particular incidents. Although the cloud employs lightweight protocols like MQTT and AMQP for IoT, TCP/IP communication required for Internet connectivity proves to be a considerable burden for small and resource-constrained IoT devices. Previous studies have improved IoT devices' networking stacks to address the interoperability issue between LLNs and the Internet networks (e.g., IPv4 and IPv6) [11, 24, 36, 38]. Since IoT devices have already been released with limited resources, it is difficult to apply their approaches in reality.

### 2.2 IoT cloud gateway

The most widely-used method to connect IoT and cloud is the use of an additional device called IoT cloud gateway. This device has the capability to handle both a specific radio protocol (e.g., LoRa, LoRaWAN, 802.15.4) and Internet connectivity [6, 36]. IoT cloud gateway runs multiple agents (e.g., message client applications) to receive and transmit messages between IoT devices and cloud. Additionally, the IoT cloud gateway runs as many agents as the number of connected IoT devices. Therefore, IoT devices can connect independently to the cloud, with each IoT device setting different message topics or different QoS. Moreover, IoT cloud gateways

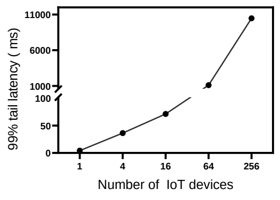

**Figure 1: Message deliver latency.**

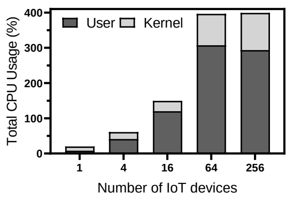

**Figure 2: Total CPU usage.**

are popular with cloud platforms because they can offload various functions such as data analysis, machine learning, and data pre-processing to the IoT cloud gateway.

### 2.3 Challenges of IoT cloud gateway

Existing IoT cloud gateways face scalability challenges, particularly regarding the number of supported LLNs protocol types and connected IoT devices. While the former can be addressed by studies such as TinyNet [11], which has developed an L2.5 stack for an IoT gateway, the latter issue remains unresolved. If this issue persists, customers may need to purchase multiple gateway devices to scale out the messages, which has led to redundant investments and additional scale-out solutions [11, 22]. In addition, there are some challenges that degrade the performance of IoT cloud gateways as follows.

First, for the IoT environment, low power consumption of devices is essential, which is why an architecture designed to efficiently consume energy, such as ARM is widely used [3]. However, this distinction can cause performance degradation [20, 32]. Furthermore, network acceleration techniques such as SmartNIC or DPDK cannot be applied to IoT cloud gateways, which are generally resource-constrained and inexpensive [27]. As a result, software-only design is necessary.

Second, data transmission over the Internet to the cloud requires security authentication, and IoT devices have to verify their certificate to establish a secure connection and encrypt data via Transport layer security (TLS) to the cloud. However, the operations such as TLS handshake and data encryption cause significant overhead [35]. This can severely degrade the performance of IoT cloud gateways when the number of IoT devices increases. Since security guarantee is indispensable in the IoT-cloud ecosystem, balancing security overhead and system performance is a key challenge.

Lastly, there is no priority consideration that the IoT cloud gateway can have on different message transmission latencies. The IoT cloud gateway is not only responsible for processing as many messages as possible from multiple devices simultaneously but also has to satisfy each requirement (e.g., latency, reliability). For example, in a smart home scenario, sensor data for a fire accident requires short latency, whereas light bulb sensor data does not. Most IoT cloud gateways process messages in a first-in-first-out manner or depend on the priority policy of the message protocols (e.g., HTTP/2, MQTT) [46]. We will discuss these challenges in more detail in the next section (§3).

## 3 MOTIVATING EXPERIMENT AND ANALYSIS

This section outlines the results of our comprehensive measurements of an IoT cloud gateway in transmitting messages between

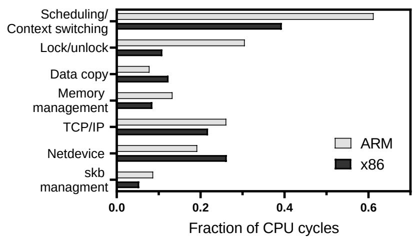

**Figure 3: CPU breakdown of two different CPU architectures.**

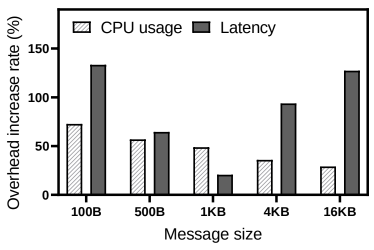

**Figure 4: Overhead increase when adding TLS encryption.**

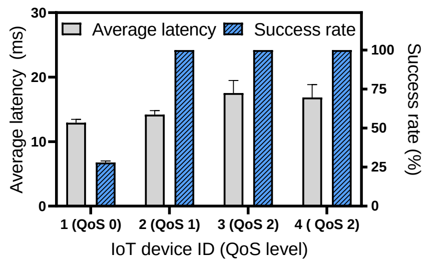

**Figure 5: Message latency and success rate of different QoS levels.**

multiple IoT devices and the cloud. Our experimental environment closely mimicked real-world IoT scenarios, incorporating cloud services like Azure [29] and AWS [2]. Referencing common IoT use cases, such as smart farm sensors [6], heartbeat sensors [37], and drone cameras [47], we simulate that each IoT device transmitted 100B–1KB of sensing data to the IoT cloud gateway every 100 ms. Subsequently, the IoT cloud gateway employed multiple agents to publish MQTT messages to the cloud side (e.g., a cloud storage) via an MQTT broker. To ensure the reliability of message delivery, we configure QoS 1 of MQTT. We also employ TLS v1.3, the latest secure layer protocol. We use a Raspberry Pi 4 as our IoT cloud gateway, which is a widely-used edge device [15, 33, 42]. It is equipped with an ARM Cortex-A72 64-bit quad core@1.5GHz CPU, a 8GB of RAM, and a 1 Gbps Ethernet chip. We also simulate a cloud-side broker using Mosquitto [13] on the server machine, which is equipped with an Intel i7-3770K octa-core@3.5GHz CPU with 64GB memory and a 256GB SSD.

## 3.1 Latency variation

In delivering and receiving feedback based on collected sensor data, message delivery latency is a crucial factor [18, 36, 45]. For instance, for SLAM, drones should update terrain information to the cloud as quickly as possible, such as in a few seconds [1, 47]. In addition, for healthcare, an electrocardiogram (ECG) sensor should report patients' heart condition without a long delay [10]. Typically, ECG data should be delivered within 1 second without any packet loss [37]. The longer latency can make the data into meaningless and useless information, as the long delivery latency prevents it from being the most current data. To report the latency variation of IoT-cloud ecosystem, we measure the IoT cloud gateway's latency, which is the time taken for a message to travel from an IoT device to the cloud.

Figure 1 displays the 99th percentile tail latency of message delivery for an IoT cloud gateway, demonstrating the scalability of its performance. We instruct each device to transmit a total of 3K messages. Each circle symbol on the line represents the latency of a specific number of IoT devices (on the x-axis). As the number of IoT devices increases from 1 to 256, the latency increases by up to 1255× and reaches 10.5 s. Nevertheless, given the expanding IoT ecosystem, an IoT cloud gateway must support a large scale of IoT devices [22, 36, 47]. In the following subsection, we investigate several root causes from various perspectives.

## 3.2 CPU bottleneck

We focus on the most common scenario, where the IoT cloud gateway is a resource-constrained device such as a Raspberry Pi [33] or Jetson Nano [31], typically equipped with a single quad-core ARM-based processor. As each IoT device sends 3K record data (e.g., sensor data, telemetry data) within 30 s, we measure the total CPU usage of the IoT cloud gateway across all cores using sysstat [16], which displays both kernel and user application usages. Figure 2 depicts the total CPU usage of the IoT cloud gateway when all agents concurrently transmit (publish) messages to the message broker in the cloud; the stacked bars in the graph indicate the application and kernel usage separately. Specifically, the CPU usage reaches saturation (400%) when the number of IoT devices is 64, leading to significant latency, as illustrated in Figure 1.

For user application CPU usage, the agents perform various functions, such as storing received data, filtering data, and creating, encrypting, and transmitting messages to the cloud. Since we execute as many agents as the number of IoT devices, the CPU usage of user applications increases linearly. Moreover, the kernel schedules CPU time for running processes, manages critical sections, and processes packets at low-level networking stacks, such as TCP, IP, and L2 layers. Although the kernel's CPU usage is smaller than that of applications, it increases rapidly, especially compared with the small number of IoT devices, which occupy CPU usage that applications must utilize. Our goal is to minimize CPU overhead, enabling us to process as many agents as possible to mitigate high latency on the IoT cloud gateway.

## 3.3 CPU architecture mismatch

To investigate the kernel-level CPU overhead, we utilize perf [17] to measure the CPU cycles per function in the kernel source code (Linux 5.15.0) and classify them into the top seven crucial roles of networking (transmitting and receiving messages) [5]. Additionally, we perform the same measurement on an x86-based machine with similar specifications and kernel version as the IoT cloud gateway device but with a distinct CPU architecture. In comparing the x86 results, we analyze any architectural mismatch points for IoT cloud gateway regarding CPU usage.

Figure 3 illustrates the breakdown of kernel CPU usage, in which 64 IoT devices are connected to an IoT cloud gateway, transmitting messages from IoT devices to the cloud. The x-axis presents the fraction of CPU cycles, while the y-axis displays the seven key roles. Among these roles, scheduling consumes a significant amount of CPU usage for both ARM and x86. As multiple agents operate simultaneously, CPU time scheduling and context switching occur frequently, resulting in high CPU usage. However, ARM uses 55.65% more CPU cycles for scheduling compared to the two architectures than x86. Moreover, with ARM, lock and unlock constitute the second-largest CPU overhead. Conversely, x86 consumes 64.35% fewer CPU cycles than ARM. The remaining roles exhibit similar differences between the two architectures with an average difference of 1.2%, which is negligible.

## 3.4 TLS encryption overhead

TLS encryption is vital for secure data transfer to and from the cloud, but it involves additional load for message processing [30]. For instance, as the TLS layer encrypts data using an authenticated key and encryption algorithm (e.g., cipher suite) per record data, approximately 100 bytes of overhead are associated with each message. In the IoT environment, the overhead of increasing data size due to encryption is not trivial due to resource usage and network performance. We measure the overhead increase with TLS encryption compared with non-TLS encryption as the message size increases (from 100B to 16KB). We assume the same total record size for every measurement. For example, the total record size is 1.6GB, so if the message size is 100B, 16K messages are transmitted. Likewise, if the size is 16KB, 100 messages are transmitted.

Regarding CPU usage (white bar in Figure 4), the smaller the message size, the greater the encryption overhead. When the message size is 100B, CPU usage increases by 72.8% compared to non-TLS encryption. However, when the message size is 16KB, it increases only by 29.2%. Therefore, considering only the encryption overhead, a maximum TLS message size of 16KB is appropriate for the IoT cloud gateway. However, from a latency perspective (gray bar in Figure 4), latency increases significantly when the message size is 16KB. This is because, with large message sizes, fragmentation frequently occurs based on MTU size in TCP. Furthermore, fragmentation optimization methods are generally challenging to apply to resource-constrained machines [26], emphasizing the criticality of determining the message size for the IoT cloud gateway. However, it is not true that the smaller the message size, the smaller the increase in latency. In fact, a message size as small as 100B could lead to a significant latency increase of 133.4% due to a larger volume of messages being processed. Therefore, selecting an appropriate message size is crucial. Based on our measurement, a message size of 1KB yields the most reasonable overhead increase.

## 3.5 Lack of latency guarantees for high-priority messages

Existing message protocols are not suitable for IoT cloud gateway to achieve priority-oriented message delivery. For example, MQTT can perform message prioritization through QoS levels, which prioritize reliability rather than latency. Higher QoS levels (1, 2) provide greater reliability but also result in higher network overhead and longer delivery times. Meanwhile, a lower QoS level (0) allows for faster processing but is less reliable and does not guarantee successful transmission. In other words, QoS levels do not provide any guarantee of real-time delivery, and higher QoS levels can result in greater network overhead and longer delivery times.

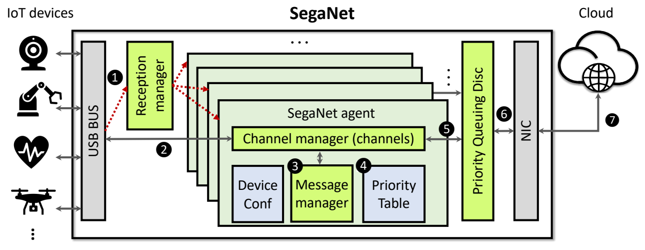

**Figure 6: Overview of SegaNet.**

Figure 5 presents the average message delivery latency and success delivered message rate of each IoT device according to different QoS levels of MQTT. We instruct that each agent publishes 10K messages within 100 seconds. We then measure the average latency and the rate of messages successfully delivered to the subscriber. The average latency of messages from IoT devices set to QoS 0 is the lowest, but only 27.9% of messages are successfully transmitted. This is because QoS 0 involves a simple, one-way send operation with no acknowledgment or retransmission. As such, the sender does not wait for an acknowledgment and thus can send the next message immediately. This speeds up communication but at the risk of potentially losing messages. While QoS 0 does not guarantee reliability, the latency of QoS 0 is still only 8.8% faster than that of QoS 1. This result demonstrates that not only does the QoS policy of the existing message protocol fail to drastically improve the message delivery latency, but also the reliability is seriously poor.

## 4 DESIGN

We propose SegaNet, a performant and priority-oriented IoT cloud gateway carefully designed to overcome the performance challenges faced by current IoT cloud gateways (which we discussed in §2.3 and §3). Figure 6 depicts the overall IoT-cloud environment, which includes a prototype of SegaNet. We first provide a workflow of SegaNet, with a detailed explanation later.

To start, IoT devices seeking to connect to the cloud initiate device registration by communicating with the reception manager of SegaNet (❶ in Figure 6). The reception module determines which core will run an agent, SegaNet agent mapping to the IoT device. The channel manager of the agent establishes two channels (connections) with the IoT device and the cloud, respectively (❷). Users can configure the settings for connection to the cloud through the device configuration file. Then, the message manager starts to receive record data from IoT device through a channel and publishes messages to be sent to the cloud according to the device configuration file (❸). Additionally, the message manager inspects the priority table to identify the message's priority level, then uses this information in the processes of publishing and transmitting the message (❹). Once the message has been published, it is transmitted to the cloud over the channel (❺). The message packet is then classified into packet queues using the priority queuing discipline before being delivered to the NIC (❻). Finally, the NIC employs priority-based scheduling to send messages from the packet queues to the cloud (❼). The workflow for transmitting messages from the cloud to the IoT device follows the same process in reverse order.

### 4.1 Multiple agents management

The reception manager is responsible for managing and executing agents, which handle messages from IoT devices connected to the cloud. Given the necessity to support connectivity for a variety of IoT devices, the IoT cloud gateway must manage numerous agents, which invoke a lot of system calls for networking. As discussed in §3.3, scheduling and lock overhead can consume high CPU cycles for networking where many applications are running simultaneously. While CFS scheduling guarantees dynamically fair CPU time scheduling, it is not efficient that consumes excessive CPU cycles for resource-constrained and ARM-based machines.

Thus, reception manager ensures that each SegaNet agent runs only on designated cores (core affinity) to minimize CPU scheduling overhead. This can be beneficial because it reduces the number of context switches between cores, which in turn mitigates the overhead of acquiring and releasing locks. Moreover, pinning the core on modern Linux kernel systems, where the application and kernel tend to run on the same core [5], increases the cache hit rate. The reception manager determines the core number by hash key, which is calculated by hash function with an input set consisting of the device ID and MAC address of the IoT device. We use the Jenkins hash function [21] rather than a round-robin table, which aims to achieve both efficiency and even distribution of hash values for a given input set with less overhead.

### 4.2 Efficient TLS encryption

The message manager is responsible for publishing messages (e.g., MQTT)—formatting messages with record data from an IoT device (e.g., set topic name, QoS level, and TLS options). Here, we consider TLS overhead when publishing messages. As discussed in §3.4, the TLS encryption overhead is heavily reliant on the message size. Therefore, we design the message manager to publish a message by message batching.

Since overhead is not a significant issue when processing a small number of devices, message batching is unnecessary, as shown in Figures 1 and 2. As a result, the message manager checks the number of actively connected IoT devices to SegaNet, which can be checked by the reception manager. Consequently, message batching only occurs if the number of connected IoT devices exceeds 50, a value that can be modified by the administrator and included in the device configuration file. If the answer is yes, the message manager aggregates record data while checking two conditions. First, the message size must be limited to 1KB, as determined by our measurements in Figure 4. Second, the maximum waiting time for aggregating record data cannot exceed 1 second, which is the maximum waiting time limit for batching. If either of these conditions is met, the message manager publishes the aggregated record data as a message.

### 4.3 Priority-oriented message delivery

In the previous sections (§4.1 and §4.2), we discussed the behavior of SegaNet in normal situations. In this section, we will describe how SegaNet handles messages based on the priority of each IoT device. Users can specify a priority value in the priority table of the SegaNet agent, with options of either 1 (high-priority) or 2 (normal-priority). The message manager will publish messages based on their priority value. For message manager, if the priority value is 1, message batching (§4.2) is not working, and the message manager will publish messages directly without aggregating record data.

Furthermore, the channel managers of high-priority IoT devices install queuing discipline filter rules to ensure that packets are processed with high priority when they are sent out through the channel. These filter rules are identified by the unique source port number of each agent. The queuing disciplines in the networking stack then classify and enqueue packets into multiple packet queues according to the source port number. Most modern NICs support multiple independent transmission packet queues [9], even on resource-constrained machines [31, 33]. Therefore, we have designed a priority scheduling algorithm for a NIC that makes packets enqueued into different packet queues and dequeued according to the priority. Thus, NIC will send out packets in the high-priority queue first, followed by packets from other queues.

## 5 PRELIMINARY RESULT

### 5.1 Prototype implementation

We have implemented a prototype of the SegaNet with reception manager and SegaNet agent. Our prototype is based on MQTT message protocol. We implement reception manager with Python codes that determines the core affinity that will execute SegaNet agents and pin them to each core using *taskset* in Linux. To implement channel manager and message manager, which are parts of the SegaNet agent, we use Paho client Java library [14]. We leverage *tc* to allow the channel manager to flexibly assign priority to packets and a NIC to perform priority-based queuing scheduling.

### 5.2 Prototype evaluation

In this section, we present an evaluation of our prototype's ability to improve latency and CPU usage compared to the existing IoT cloud gateway (native). We also evaluate whether our priority-oriented message delivery design can ensure that high-priority messages are delivered faster. Our experimental environment is the same as described in §3, and we conduct each measurement three times.

**Message latency improvement.** Figure 7 shows the 99th percentile tail latency of messages for SegaNet and native. As the number of connected IoT devices increases, latency increases up to 12.7 s for native and up to 0.76 s for SegaNet on average. Compared to the IoT device number is 1, the latency of native and SegaNet increases by 2067× and by 78×, respectively. In particular, when the number of IoT devices is 64 compared to 16 in the SegaNet, it increases rapidly (26.5× increase). This is because of the waiting time to aggregate record data for message batching (400 ms on average). However, it is reasonable that SegaNet improves latency by up to 16.7× compared to native, and latency grows up to 762.5 ms, which is within 1s even the number of IoT devices is 256.

**CPU usage improvement.** Figure 8 shows the total CPU usage of both gateways. In all cases, SegaNet shows lower CPU usage than native. When the number of IoT devices ranges from 1 to 16, SegaNet consumes 26.7% less CPU usage than native on average, and for 64 and 256 IoT devices, it consumes 77.6% and 64.6% less, respectively. In particular, for native, CPU becomes saturated from the number of IoT devices is 64, whereas, for SegaNet, CPU is not saturated even when the number of IoT devices reaches 256.

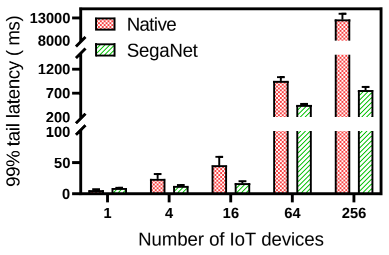

Figure 7: Message delivery Latency comparison.

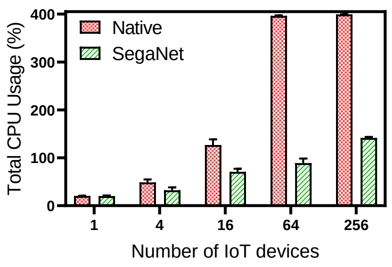

Figure 8: Total CPU usage comparison.

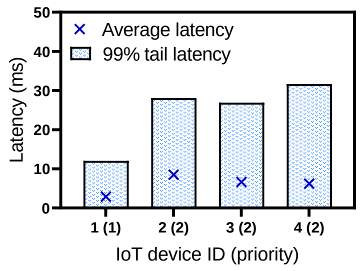

Figure 9: Latency comparison according to priority.

It consumes up to 141.6% in total, which means there is no CPU bottleneck to handle a number of IoT devices simultaneously.

**Priority-oriented message delivery.** Figure 9 shows message delivery latency of four IoT devices that have different priorities. We set a priority of device 1 as 1 and others as 2. Also, all of IoT devices set QoS 1 for delivery reliability. We instruct each IoT device to send 10K record data within 100 s and measure the average and 99th percentile tail latency of each message. Device 1, which has a higher priority, has up to 35.8% and 43.17% better (lower) for average latency and tail latency than other devices, respectively. Moreover, not only does device 1 achieve fast message transmission, but all messages were delivered successfully because it keeps the reliability of the message protocol (QoS 1 of MQTT).

## 6 DISCUSSION AND FUTURE WORK

**Containerized environment.** Modern public cloud platforms gather IoT telemetry and harness machine learning to provide advanced services for IoT [28]. These applications are deployed and executed via container images and container runtimes (e.g., Docker), ensuring simplified deployment [8]. Moreover, IoT agents can operate as container runtimes on an IoT cloud gateway. Nevertheless, such containerized environments necessitate supplementary networking techniques, such as overlay or NAT, for external communication, thereby generating significant overhead [41]. In our future work, we propose investigating the integration of existing efficient solutions, such as Slim [48] and k3s [23], with the goal of enhancing SegaNet's container networking performance.

**Priority maintenance challenge in cloud.** It is vital to devise methods that consistently maintain the order of message importance in the cloud, even though messages are dispatched according to their priority from the IoT cloud gateway. The principal challenge is that messages from various devices converge in the cloud, potentially disrupting the initial priority order. As the cloud connects to a multitude of IoT devices, maintaining priority order becomes increasingly complex. Crucial factors, such as the geographical locations (e.g., round-trip time) and the individual message priority policy of IoT devices, necessitate further consideration. Hence, we intend to collaborate with the cloud side in developing algorithms or policies that effectively account for these aspects, thereby ensuring accurate priority maintenance in the cloud.

**Communication bottlenecks from IoT devices.** As IoT devices and the IoT cloud gateway communicate, several interference issues can occur among the devices. A key problem emerges when different radios use the same frequency, as in the case of both Zigbee and Bluetooth operating on the 2.4 GHz ISM band. Despite using frequency hopping and coexistence mechanisms, interference still occurs when radios share the same frequency band [34]. Additionally, a bottleneck might occur in the USB bus if the IoT cloud gateway processes multiple data streams simultaneously from additional USB module-type sensors [25]. These interferences can then result in delayed message delivery and disrupt the priority order of messages. We therefore intend to appoint the order of record data transmission of IoT devices while considering interference.

**Evaluation in large-scale IoT environment.** SegaNet assures robust internet connectivity for a multitude of IoT devices, while ensuring low message latency and optimal CPU utilization. Given its capabilities, it is essential to contemplate the potential impact of SegaNet in real-world and large-scale settings for future studies. To this end, we aim to persist with the evaluation of SegaNet, leveraging the diverse nodes and networks provided by OpenNetLab[12].

## 7 CONCLUSION

We have meticulously designed and implemented a prototype of SegaNet, aimed at facilitating the scalability of numerous IoT devices connected to the cloud. The promising results from our prototype evaluation demonstrate the potential for IoT cloud gateways to seamlessly accommodate a vast array of devices. Moreover, we have showcased the gateway's ability to prioritize message delivery while maintaining compatibility with existing message protocols.

SegaNet has the potential to serve not only as an IoT cloud gateway but also as an efficient framework to bridge the gap between IoT and cloud systems. By providing a vision for efficiently connecting IoT and cloud technologies, we believe that SegaNet can contribute significantly to the growth of a thriving IoT-cloud ecosystem.

## ACKNOWLEDGMENTS

This work was supported by OpenNetLab project and Institute of Information & communications Technology Planning & Evaluation funded by the Korea government (Ministry of Science and ICT) (2022-00155958, Collaborative Research Projects with Microsoft Research Asia). This work was also partly supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2021R1A6A1A13044830), and by the NRF grant funded by the Korea government (MSIT) (NRF-2023R1A2C3004145).

## REFERENCES

[1] Fawad Ahmad, Hang Qiu, Ray Eells, Fan Bai, and Ramesh Govindan. 2020. CarMap: Fast 3D Feature Map Updates for Automobiles.. In *NSDI*. 1063–1081.

[2] AWS Amazon. 2022. AWS IoT Core Features. (Feb 2022). https://aws.amazon.com/iot-core/features/ [Accessed: Mar. 15. 2023.].

[3] Rafael Vidal Aroca and Luiz Marcos Garcia Gonçalves. 2012. Towards green data centers: A comparison of x86 and ARM architectures power efficiency. *J. Parallel and Distrib. Comput.* 72, 12 (2012), 1770–1780.

[4] Gunjan Beniwal and Anita Singhrova. 2022. A systematic literature review on IoT gateways. *Journal of King Saud University-Computer and Information Sciences* 34, 10 (2022), 9541–9563.

[5] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. 2021. Understanding host network stack overheads. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 65–77.

[6] Tusher Chakraborty, Heping Shi, Zerina Kapetanovic, Bodhi Priyantha, Deepak Vasisht, Binh Vu, Parag Pandit, Prasad Pillai, Yaswant Chabria, Andrew Nelson, Michael Daum, and Ranveer Chandra. 2022. Whisper: IoT in the TV White Space Spectrum. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, 401–418.

[7] Djabir Abdeldjalil Chekired, Lyes Khoukhi, and Hussein T Mouftah. 2018. Industrial IoT data scheduling based on hierarchical fog computing: A key for enabling smart factory. *IEEE Transactions on Industrial Informatics* 14, 10 (2018), 4590–4602.

[8] Jun Lin Chen, Daniyal Liaqat, Moshe Gabel, and Eyal de Lara. 2022. Starlight: Fast container provisioning on the edge and over the {WAN}. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 35–50.

[9] Pavel Chuprikov, Sergey Nikolenko, and Kirill Kogan. 2015. Priority queueing with multiple packet characteristics. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 1418–1426.

[10] Berken Utku Demirel, Islam Abdelsalam Bayoumy, and Mohammad Abdullah Al Faruque. 2021. Energy-efficient real-time heart monitoring on edge–fog–cloud internet of medical things. *IEEE Internet of Things Journal* 9, 14 (2021), 12472–12481.

[11] Wei Dong, Jiamei Lv, Gonglong Chen, Yihui Wang, Huikang Li, Yi Gao, and Dinesh Bharadia. 2022. TinyNet: A lightweight, modular, and unified network architecture for the internet of things. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 248–260.

[12] Jeongyoon Eo, Zhixiong Niu, Wenxue Cheng, Francis Y Yan, Rui Gao, Jorina Kardhashi, Scott Inglis, Michael Revow, Byung-Gon Chun, Peng Cheng, et al. 2022. OpenNetLab: Open platform for RL-based congestion control for real-time communications. (2022).

[13] Eclipse foundation. 2023. Eclipse Mosquitto, An open source MQTT broker. (2023). https://mosquitto.org/ [Accessed: Mar. 1. 2023.].

[14] Eclipse foundation. 2023. Eclipse Paho Java Client. (2023). https://www.eclipse.org/paho/index.php?page=clients/java/index.php [Accessed: Mar. 1. 2023.].

[15] Tyler Gizinski and Xiang Cao. 2022. Design, Implementation and Performance of an Edge Computing Prototype Using Raspberry Pis. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. 0592–0601.

[16] Sebastien Godard. 2023. Performance monitoring tools for Linux. (2023). https://github.com/sysstat/sysstat [Accessed: Mar. 1. 2023.].

[17] Brendan Gregg. 2020. Linux perf Examples. (July 2020). https://www.brendangregg.com/perf.html [Accessed: Mar. 15. 2023.].

[18] Jin Huang, Colin Samplawski, Deepak Ganesan, Benjamin Marlin, and Heesung Kwon. 2020. Clio: Enabling automatic compilation of deep learning pipelines across iot and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–12.

[19] IDC. 2023. IDC's Worldwide Global DataSphere IoT Device and Data Forecast, 2019–2023. (2023). https://www.idc.com/ [Accessed: Mar. 15. 2023.].

[20] Ayush Jalan. 2022. x86 vs. ARM: Which Architecture Should Your Next PC Use? (2022). https://www.makeuseof.com/x86-vs-arm-which-architecture-should-pc-use/ [Accessed: Mar. 1. 2023.].

[21] Bob Jenkins. 2013. A hash function for hash Table lookup. (Nov 2013). http://www.burtleburtle.net/bob/hash/doobs.html [Accessed: Mar. 15. 2023.].

[22] Jinhwan Jung, Jihoon Ryoo, Yung Yi, and Song Min Kim. 2020. Gateway over the air: Towards pervasive internet connectivity for commodity iot. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. 54–66.

[23] k3s. 2023. K3s - Lightweight Kubernetes. (2023). https://docs.k3s.io/ [Accessed: May. 1. 2023.].

[24] Hyung-Sin Kim, Sam Kumar, and David E Culler. 2019. Thread/OpenThread: A compromise in low-power wireless multihop network architecture for the Internet of Things. *IEEE Communications Magazine* 57, 7 (2019), 55–61.

[25] Carel P Kruger and Gerhard P Hancke. 2014. Benchmarking Internet of things devices. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 611–616.

[26] Sam Kumar, Michael P Andersen, Hyung-Sin Kim, and David E Culler. 2020. Performant TCP for Low-Power Wireless Networks.. In *NSDI*. 911–932.

[27] Xiaofeng Lin, Yu Chen, Xiaodong Li, Junjie Mao, Jiaquan He, Wei Xu, and Yuanchun Shi. 2016. Scalable kernel TCP design and implementation for short-lived connections. *ACM SIGARCH Computer Architecture News* 44, 2 (2016), 339–352.

[28] Microsoft. 2023. Understand Azure IoT Edge modules. (2023). https://learn.microsoft.com/en-us/azure/iot-edge/iot-edge-modules?source=recommendations&view=iotedge-1.4 [Accessed: May. 1. 2023.].

[29] Azure Microsoft. 2023. How an IoT Edge device can be used as a gateway. (Feb 2023). https://learn.microsoft.com/en-us/azure/iot-edge/iot-edge-as-gateway [Accessed: Mar. 15. 2023.].

[30] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. 2015. Multi-context TLS (mcTLS) enabling secure in-network functionality in TLS. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 199–212.

[31] NVIDIA. 2023. Jetson Nano. (2023). https://developer.nvidia.com/embedded/jetson-nano [Accessed: Mar. 15. 2023.].

[32] Jongseok Park, Kyungmin Bin, and Kyunghan Lee. 2022. mGEMM: low-latency convolution with minimal memory overhead optimized for mobile devices. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 222–234.

[33] Raspberry pi. 2023. Raspberry Pi for industry. (2023). https://www.raspberrypi.com/for-industry [Accessed: Mar. 15. 2023.].

[34] Fengzhong Qu, Fei-Yue Wang, and Liuqing Yang. 2010. Intelligent transportation spaces: vehicles, traffic, communications, and beyond. *IEEE Communications Magazine* 48, 11 (2010), 136–142.

[35] Florentin Rochet, Emery Assogba, and Olivier Bonaventure. 2020. TCPLS: Closely Integrating TCP and TLS. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*. 45–52.

[36] Vadim Safronov, Justas Brazauskas, Matthew Danish, Rohit Verma, Ian Lewis, and Richard Mortier. 2021. Do we want the New Old Internet? Towards Seamless and Protocol-Independent IoT Application Interoperability. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*. 185–191.

[37] Saurabh Shukla, Mohd Fadzil Hassan, Muhammad Khalid Khan, Low Tang Jung, and Azlan Awang. 2019. An analytical model to minimize the latency in healthcare internet-of-things in fog computing environment. *PloS one* 14, 11 (2019), e0224934.

[38] Michael Spörk, Carlo Alberto Boano, Marco Zimmerling, and Kay Römer. 2017. Bleach: Exploiting the full potential of ipv6 over ble in constrained embedded iot devices. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–14.

[39] Statista. 2023. Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030. (2023). https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ [Accessed: Mar. 15. 2023.].

[40] Biljana L Risteska Stojkoska and Kire V Trivodaliev. 2017. A review of Internet of Things for smart home: Challenges and solutions. *Journal of cleaner production* 140 (2017), 1454–1464.

[41] Kun Suo, Yong Zhao, Wei Chen, and Jia Rao. 2018. An analysis and empirical study of container networks. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 189–197.

[42] Blesson Varghese, Nan Wang, David Bermbach, Cheol-Ho Hong, Eyal De Lara, Weisong Shi, and Christopher Stewart. 2021. A survey on edge performance benchmarking. *ACM Computing Surveys (CSUR)* 54, 3 (2021), 1–33.

[43] JP. Vasseur. 2014. Terms Used in Routing for Low-Power and Lossy Networks. (2014). https://datatracker.ietf.org/doc/html/rfc7102 [Accessed: Mar. 1. 2023.].

[44] Morgan Vigil-Hayes, Md Nazmul Hossain, Alexander K Elliott, Elizabeth M Belding, and Ellen Zegura. 2022. LoRaX: Repurposing LoRa as a Low Data Rate Messaging System to Extend Internet Boundaries. In *ACM SIGCAS/SIGCHI Conference on Computing and Sustainable Societies (COMPASS)*. 195–213.

[45] Chen Wang, Ruonan Zhang, Haotong Cao, Junhao Song, and Wei Zhang. 2022. Joint optimization for latency minimization in UAV-assisted MEC networks. In *Proceedings of the 5th International ACM Mobicom Workshop on Drone Assisted Wireless Communications for 5G and Beyond*. 19–24.

[46] Maarten Wijnants, Robin Marx, Peter Quax, and Wim Lamotte. 2018. Http/2 prioritization and its impact on web performance. In *Proceedings of the 2018 World Wide Web Conference*. 1755–1764.

[47] Jingao Xu, Hao Cao, Zheng Yang, Longfei Shangguan, Jialin Zhang, Xiaowu He, and Yunhao Liu. 2022. SwarmMap: Scaling up real-time collaborative visual SLAM at the edge. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 977–993.

[48] Danyang Zhuo, Kaiyuan Zhang, Yibo Zhu, Hongqiang Harry Liu, Matthew Rockett, Arvind Krishnamurthy, and Thomas Anderson. 2019. Slim:{OS} kernel support for a low-overhead container overlay network. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 331–344.