Latency Optimization in IoT Networks Using SD-WAN Edge Computing

Rashmi S. Vaidya
Department of Electrical Engineering
San Jose State University in California
San Jose, CA, 95195, U.S.A
email: rashmi.s.vaidya@sjsu.edu

Nader F. Mir
Department of Electrical Engineering
San Jose State University in California
San Jose, CA, 95195, U.S.A
email: nader.mir@sjsu.edu

Abstract — This paper aims to reduce latency in Internet of Things (IoT) sensor networks using Software-Defined Wide Area Network (SD-WAN) edge computing. The motivation comes from the strong necessity of rapid decision-making in the emerging IoT applications such as smart cities and autonomous vehicles. In the real communications world, the processing speed of devices and sensor nodes in an IoT network is slow as nodes and devices have limited power and processing resources, while cloud-based methods are time-consuming to access distant servers. This paper uses a custom SD-WAN controller to coordinate the task allocation to nearby edge servers to lower the response time as well as data processing delays. Algorithms such as the ones for traffic prioritization based on QoS requirements, edge computing resource allocation, and edge caching techniques are also deployed to achieve the goal. Software implementation and simulations are used to quantify the latency reduction achieved. Through SD-WAN and edge computing, the paper focuses on techniques for reducing the IoT network latency to improve IoT operations' safety and efficiency.

Keywords—IoT Networks; Sensors; Latency Optimization; Software-Defined Wide Area Networking (SD-WAN); Traffic Prioritization; Quality of Service (QoS); Cloud Computing; Edge Computing; Edge Caching; Smart Cities; Autonomous Vehicles.

I. INTRODUCTION

Internet of Things (IoT) networks consist of interconnected sensor nodes that collect and transmit data over the Internet [1]. These networks are key to the ecosystem, enabling various industry applications such as smart homes, healthcare, industrial automation, and agriculture by integrating data collection, transmission, processing, storage, and user interaction.

Latency, defined as the round-trip delay in data transmission, is a critical factor in IoT networks where real-time responses are essential. IoT applications engage large amounts of data transfers requiring response rates and efficiency. For instance, autonomous cars need instant decisions for safety, and any data delay could cause accidents. Similarly, quick responses are crucial for smart city applications like traffic management and emergency response at the time of a natural disaster. Latency may occur due to various factors like propagation times, transmission delays, processing delays, and queuing delays [2]. Factors like distance, network congestion, transmission medium, and hardware efficiency may also affect latency.

Reducing latency in IoT networks is crucial, however, IoT devices often use small, low-cost sensors with limited memory and power, and poor network connectivity. Computing tasks can be slow or impossible on IoT devices. Cloud computing on the other hand can help, but the fact that servers might be in distant locations may add wait time, which can be hectic for IoT applications needing low delays [3].

Software-Defined Wide-Area Network (SD-WAN) is a technology that uses defined software to optimize wide area network connections resulting in more flexibility and control over traditional WAN architectures. With SD-WAN, certain network management features allow organizations to efficiently connect users across multiple locations. Similar to the software-defined networking in data centers, as long as configuration messages are supported by all the network hardware device makers, SD-WAN decouples the networking hardware from its control system and creates a central control plane in the WAN. This concept is like how software-defined networking [2] implements virtualization technology resulting in significant simplification in managing a network. The most promising feature of SD-WAN is its ability to construct higher-performance software-defined based networks. SD-WAN creates an environment through which a wide area network uses software-defined networking control for communicating over the Internet using overlay tunnels which are encrypted when destined for internal organization locations.

The current trend of designing IoT networks is traditionally concentrated on connecting IoT devices to wide area networks [4]. We try in this paper to explore certain strategies including the deployment of SD-WAN to allow centralized control of the network, enabling dynamic adjustments to traffic flow. This deployment provides easy programmability for efficient network management and automates network behavior through software applications. We also demonstrate the deployment of QoS-based traffic prioritization by prioritizing critical IoT data over less time-sensitive information, where the Quality of Service (QoS) [5][6] ensures that high-priority tasks, like emergency alerts are transmitted with minimal delay. Additionally, we apply edge computing [7] in our study where processing data is placed closer to the sensors, at the edge of the network, to minimize the need to send data to distant cloud servers. This reduces the round-trip time and enables quicker decision-making for time-sensitive applications. Finally, we run our study in this paper with the use of edge caching [8][9] through which frequently accessed data at the edge helps reduce latency by minimizing the need to retrieve data from the cloud, speeding up access times for repeated requests.

The rest of this paper is organized as follows. In Section II, we present the details of a proposed architecture for IoT networks that optimizes the latency. In Section III, we present

detailed results of our analysis, and in Section IV, a conclusive statement is presented.

II. PROPOSED ARCHITECTURE

The proposed architecture uses SD-WAN to enhance the latency performance of IoT sensor networks. This can be done by implementing a network topology with a central SD-WAN controller, OpenFlow SD-WAN switches, QoS based traffic prioritization, edge servers and edge cache, as shown in Figure 1. The key components of the architecture are SD-WAN controller to manage the traffic flow and implement adaptive strategies, and SD-WAN switches that perform switching and forwarding in layers 2 and 3. The SD-WAN controller optionally uses Ryu software [10]. Ryu is an open, software-defined networking controller, written in Python and is supported and used in cloud data centers. In this networking set-up, IoT sensor devices send data of varying rates and sizes while edge servers and caches preprocess data and store frequently accessed information.

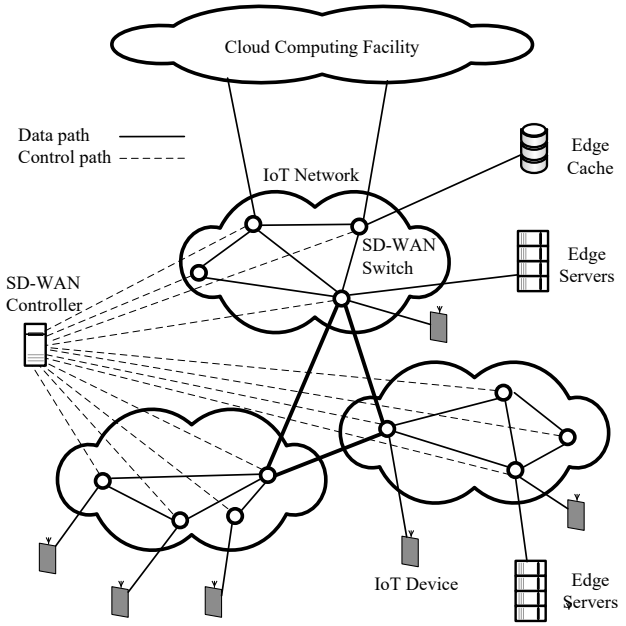

Figure 1. The setup for the IoT network with SD-WAN features.

For implementing effective traffic management, we utilize the Type of Service (ToS) byte existing in the IP packet header, as shown in Figure 2. The ToS field is used for enforcing Quality of Service (QoS) and prioritizing packets. The field uses its first 6-bits as Differentiated Services Code Point (DSCP) to classify traffic based on better QoS. The remaining two bits of ToS is called Explicit Congestion Notification (ECN) which is used for signaling a network for congestion. Based on this available feature, we classify packets from IoT devices into three types; each assigned a specific DSCP value to facilitate QoS prioritization. The SD-WAN controller uses DSCP values to prioritize certain traffic based on its importance, ensuring that real-time data receives preferential treatment over less critical traffic. Also, to implement QoS-based prioritization, each SD-WAN-enabled switch or router maintains three separate queues.

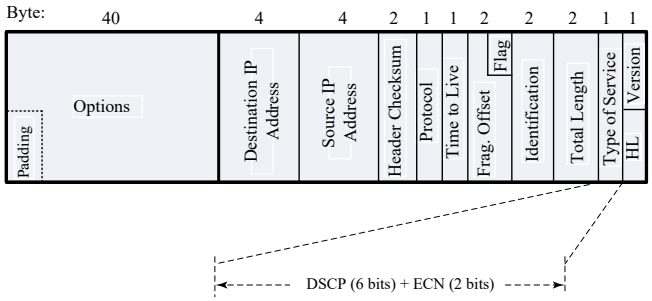

Figure 2. DSCP fields in IP header for QoS enforcement.

The following DSCP values and queue types are used in this paper to differentiate between three types of traffic: **EF 46**: high-priority time-sensitive traffic, such as real-time sensor data. The traffic is marked as Expedited Forwarding (EF) with a DSCP value of 46 (binary 101110), guaranteeing it is processed ahead of other traffic. The High-Priority Queue is used for this real-time traffic ensuring minimal latency. **AF31 26**: medium-priority routine sensor data, such as periodic status updates. The traffic is marked as Assured Forwarding (AF) with a DSCP value of 26 (binary 011010), providing medium-level priority. This queue is processed only when the high-priority queue is empty. **CS1-8**: low-priority non-urgent data, such as software updates or system logs, which do not require immediate processing and can tolerate longer delays. The traffic is assigned a low priority DSCP value, typically CS1 with a value of 8 (binary 001000), processed only after higher-priority traffic. For this traffic (CS1-8 or unmarked), a low priority queue is used and is processed only when the high and medium-priority queues are empty, ensuring real-time traffic is not delayed.

For the analysis of our network set-up, we utilized several software tools and platforms such as Virtual Machine, Mininet, Ryu controller, Python test scripts, Wireshark, iperf, and ping tools. The virtual machine setup includes VMware Fusion running Ubuntu OS Desktop 22.04 LTS “Jammy Jellyfish” Daily Build for Arm64 architecture on macOS (Apple ATM M2 silicon), with 4 GB of RAM and 2 CPU cores. The VM uses the same network adapter as the host OS. Mininet serves as the primary tool for network emulation, capable of creating both traditional non-SD-WAN- and SD-WAN-enabled IoT networks with edge computing capabilities. Although Mininet provides its own controller by default, this project uses the Ryu SD-WAN controller. The SD-WAN controller using the Ryu controller has been installed within the Ubuntu VM and serves as the central control unit for the SD-WAN-enabled network. It enables dynamic network management and policy implementation. Different network topologies are defined using Python scripts that run Mininet and instantiate hosts, switches, and routers. Various tests, such as the QoS test, Edge Server offloading test, and Edge Cache test, were also implemented using Python scripts. The QoS test requires C code to generate traffic of different sizes and QoS values. These test scripts save latency values into CSV files, which are later imported

by another set of Python scripts to generate graphs from the results.

The network simulation also uses Wireshark to capture and analyze network traffic within the Mininet environment. It is used for verifying QoS policies, monitoring data flow, and measuring latency improvements. Ping and iperf command-line tools are used for network performance analysis. Ping measures Round-Trip Time (RTT) between hosts, providing information on minimum, average, and maximum latency, while iperf measures maximum achievable bandwidth on IP networks.

The SD-WAN topology is set up using Mininet emulator with the inclusion of an external Ryu controller. Initial packet captures show the exchange of OpenFlow messages between the SD-WAN controller and SD-WAN switches, establishing the SD-WAN control plane and subsequent network operations.

Upon initializing the topology, OpenFlow Hello messages are exchanged between the SD-WAN controller and each switch to negotiate the OpenFlow version. Following this, Feature Request and Feature Reply messages are exchanged, with the controller sending a Feature Request to each switch and the switches responding with Feature Reply messages containing their capabilities and available ports. Packet-In messages are sent from switches to the controller when packets without matching flow entries are encountered, and the controller responds with Flow-Mod messages instructing the switches on packet handling. The flow tables can be examined to verify the appropriate flow entries installed by the SD-WAN controller. The following SD-WAN request response pairs are used to make QoS settings on the SD-WAN switches. The request response pairs show how the queues are set along with DSCP flow rules [10]. For example, note the queue type in response to the request **46** which is a high priority traffic, as explained earlier.

```
Request:
{"port_name":"s1-eth1","type":"linux-
htb","max_rate":"1000000","queues":[{"min
_rate":"800000"},{"min_rate":"500000"},{"
max_rate":"500000"}]}

Response:
[{"switch_id":"0000000000000001","command
_result":{"result":"success","details":{"
0":{"config":{"min-
rate":"800000"}},"1":{"config":{"min-
rate":"500000"}},"2":{"config":{"max-
rate":"500000"}}}}}]

Request:
{"match":{"ip_dscp":"46"},"actions":{"que
ue":"1"}}

Response:
[{"switch_id":"0000000000000001","command
_result":[{"result":"success","details":"
QoS added. : qos_id=1"}]}]
```

III. Analysis and Results

A. Effect of QoS Marking on Latency

To capture the effect of QoS marking on the network latency, a test was set up that sends three types of messages continuously on the network link and shows how the network performs with and without QoS settings. The test creates three concurrent threads, each simulating a different traffic type: real-time sensor readings, periodic status messages, and system logs. Each thread runs for 30 seconds, both with and without QoS settings to obtain latency values. The results included in Table I show a comparison of latencies from three readings: high priority real-time readings, medium status readings, and low priority periodic readings. QoS effectively manages network congestion and prioritizes traffic. Without QoS, latency rapidly increases for all traffic types, reaching several seconds by the end of the test. In contrast, the QoS-enabled scenario shows consistent, low latencies for all traffic types, with a preference for higher-priority traffic.

TABLE I. COMPARISON OF LATENCIES FROM THREE READINGS: HIGH PRIORITY REAL-TIME SENSORS, MEDIUM STATUS MESSAGES, AND LOW PRIORORITY PERIODIC MESSAGES.

	High Priority real-time sensor readings (100-byte messages)		**Medium Priority real-time sensor readings (250-byte messages)**		**Low Priority real-time sensor readings (2000-byte messages)**	
Sim. Run Time (sec)	***Latency (ms) Without QoS***	***Latency (ms) With QoS (DSCP value EF 46)***	***Latency (ms) Without QoS***	***Latency (ms) With QoS (DSCP value AF31 26)***	***Latency (ms) Without QoS***	***Latency (ms) With QoS (DSCP value CS1-8)***
0	18.4	15.8	17.8	16.7	26.7	33.9
5	1261	80.8	1485	81.2	1538	95.4
10	2782	82.9	3081	72.4	3083	93.9
15	4345	80.2	4508	90.8	4659	92.8
20	5054	87.3	5096	84.3	5100	97.7
25	5349	74.6	5448	95.1	5446	90.2
30	8244	80.5	7906	87.6	8172	85.4

B. Effect of Edge Server Offloading on Latency

The next test measures the latency of the SD-WAN topology when an edge server offloads part of the workload of the main cloud server. This task implements offloading percentages ranging from 10% to 90% and measures the latency for each scenario. The results presented in Figure 3 show a trend across different distances between the edge and cloud servers (1,600 km to 8,000 km). As the offloading rate increases due to more tasks processed in the edge server, latency decreases, particularly for greater distances. At lower rates offloading near 10% (mostly cloud computing), the

latency ranges from about 7.5ms (1,600 km) to about 30.5ms (8,000 km). At higher rates offloading near 90% (mostly edge processing), latency ranges from about 0.5ms (1,600 km) to 5.0ms (8,000 km). Figure 3 also expresses that edge computing significantly reduces latency, especially with higher offloading percentages. The impact is more pronounced as the distance to the cloud server increases. Even a small percentage of edge offloading can provide noticeable latency improvements, particularly for longer distances. It is noticeable that implementing 90% offloading reduces latency significantly.

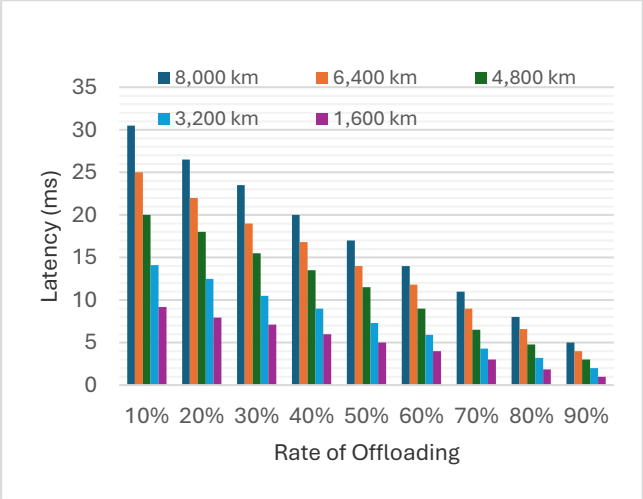

Figure 3. Latency reduction with edge server offloading.

C. *Effect of Edge Cache Size on Latency*

This study evaluates the impact of different edge cache sizes and request frequencies on latency in the SD-WAN topology. It checks how the latency is impacted by increasing request frequencies (10, 100, 500 and 1000 requests/sec) on increasing Edge Cache sizes (50, 100, 150 and 200 KB). Random requests are generated to derive different hit ratios based on the cache size and the Round-Trip Time (RTT) is measured for requests to both a cloud server and an edge server. Cache eviction policy is Least Recently Used (LRU). The code reads the Minimum; Average and Maximum Latency obtained from the test and plots the maximum latency values on the graph.

The results in Figure 4 show that increasing the edge cache size improves performance, with lower latencies and higher hit rates. However, the improvement is not linear and gives diminishing returns as cache size grows. Higher request frequencies lead to slightly increased latencies due to increased system load and cache contention, however, this is not consistently seen across all cache sizes. In general, it can be concluded that a four times bigger cache gives about 20% lower latency values.

D. *Future Trends Based on the Use of AI*

The future research and development efforts of this project can focus on various aspects such as security enhancement, and the deployment of Artificial Intelligence (AI). Security measures such as blockchain and encryption techniques could be incorporated to protect against cyberattacks. AI integration could particularly enable predictive traffic management, and resource allocation resulted in enhancing network performance of the IoT networks.

Figure 4. Latency reduction with edge cache sizes.

IV. Conclusions and Future Work

This paper addressed the improvement of latency in IoT networks by using the combination of Software-Defined Wide-Area Network (SD-WAN), traffic prioritization, and edge computing. IoT applications, such as smart cities and autonomous vehicles, require rapid data processing. We discussed in this paper how IoT device processing and power limitations and the use of cloud computing introduce noticeable delays. The framework we suggested required SD-WAN's centralized control and edge computing's localized processing to create a low-latency IoT infrastructure. We created SD-WAN, QoS prioritization, and edge computing, and demonstrated substantial latency reductions. In the QoS network setup with the help of DSCP field of IP packet headers, high-priority traffic experienced a dramatic decrease in latency compared to the network setup without QoS. This highlighted the effectiveness of SD-WAN's centralized control in managing network congestion and prioritizing critical data. The edge server offloading test revealed noticeable reduction in latency when high-rate tasks were offloaded to nearby edge servers, particularly in scenarios involving greater distances between edge and cloud servers. Edge caching study confirmed its role in minimizing data retrieval delays. By increasing the size of cache, the traffic latency was reduced. The future work must mainly concentrate on the inclusion of Artificial Intelligence (AI) in the structure of SD-WAN to enable predictive traffic management, resource allocation, and anomaly detection.

References

[1] H. Xu, W. Liu, L. Li, and Q. Zhou, “An IoT-based low-cost architecture for smart libraries using SDN,” Scientific Reports, 2024.

[2] J. Kurose and K. W. Ross, “Computer Networks, A Top-Down Approach,” Pearson, 2017.

[3] K. Liu, Y. Meng, and G. Sun, “An Overview on Edge Computing Research,” IEEE Access. pp. 1-1, 2020, [Online]. Available from:

https://www.researchgate.net/publication/341096184_An_Overview_on_Edge_Computing_Research [Accessed Apr. 9, 2024].

[4] W. Stallings, "Foundations of Modern Networking SDN, NFV, QoE, IoT, and Cloud," Pearson Education, Inc, 2016.

[5] M. Beshley, N. Kryvinska, H. Beshley, O. Panchenko, and M. Medvetskyi, "Traffic Engineering and QoS/QoE Supporting Techniques for Emerging Service-Oriented Software-Defined Network", Journal of Communications and Networks, vol. 26, no. 1, Feb. 2024.

[6] Hillstone Networks, "Introduction to QoS." [Online]. Available from: https://www.hillstonenet.com/support/4.5/en/preface.html#config_qos_intro.html, [Accessed Sept. 11, 2024].

[7] S. Douch, M. R. Abid, K. Zine-Dine, D. Bouzidi and D. Benhaddou, "Edge Computing Technology Enablers: A Systematic Lecture Study," IEEE Access, vol. 10, pp. 69264-69302, 2022. [Online]. Available from: https://ieeexplore.ieee.org/document/9797685 [Accessed Apr. 9, 2024].

[8] A. Jebamani and G. Winster, "A Survey of Edge Computing in IOT devices," Proceedings of the International Conference on Innovative Computing & Communication (ICICC) 2022. [Online]. Available from: https://ssrn.com/abstract=4023176 [Accessed Apr. 9, 2024].

[9] H. Li, M. Sun, F. Xia, X. Xu, and M. Bilal. "A Survey of Edge Caching: Key Issues and Challenges", Tsinghua Science and Technology, ISSN 1007-0214 14/20 pp. 818−842 DOI: 10.26599/TST.2023.9010051, vol. 29, no. 3, June 2024.

[10] RYU Project Team, "RYU SDN Framework: Using OpenFlow 1.3", 2014. [Online]. Available from: https://osrg.github.io/Ryu-book/en/Ryubook.pdf, [Accessed Nov. 30, 2024].