ITU Journal of Wireless Communications and Cybersecurity

ITU/JWCC

Protocol-Based Traffic Flow Regulation on SDN with Ryu's REST APIs for Testbeds

Sultan Çoğay[1], **Ayça Tulum** [1], **and Gökhan Seçinti**[1 2]

[1] Department of Computer Engineering, Istanbul Technical University, Istanbul, 34469, Turkey
[2] BTS - Digital Twin Application and Research Center, Istanbul Technical University, Istanbul, 34469, Turkey

Abstract: The conventional network has numerous challenges due to its strict rule-based configurations and dependencies. Software-Defined Networking (SDN) responds by separating the control plane, which controls network decisions, from the data plane, in the role of packet forwarding, therefore creating a paradigm change. This study proposes an SDN-based traffic flow regulation framework utilizing Open vSwitch (OVS), the Ryu controller, and its associated RESTful APIs. Designed primarily to enforce Quality of Service (QoS) rules dynamically, the framework is flexible enough to meet the changing needs of several application-layer protocols. The centralized control facilitated by the Ryu controller enables the dynamic and flexible implementation of QoS rules and rapid response to changes in network conditions. Validation of the proposed system is performed through extensive real-time traffic monitoring using packet analysis tools such as TCP-Dump and Wireshark. Experimental results show that SDN-based design emphasizes its potential power as a practical solution to conventional networking limitations since it greatly improves network performance and responsiveness.

Keywords: Software-defined networking, quality of service, network traffic.

Test Ortamları için Ryu'nun REST API'leri ile SDN Üzerinde Protokol Tabanlı Trafik Akışı Düzenlemesi

Özet: Geleneksel ağlar, katı kural tabanlı konfigürasyonları ve donanımsal bağımlılıklar nedeniyle çok sayıda zorluk yaşamaktadır. Bu kısıtlamalar, esnekliği sınırlamakta ve ağ cihazlarının sıkı entegre edilmiş karar verme mantığı sebebiyle değişen ağ koşullarına hızlı adaptasyonu engellemektedir. Yazılım Tanımlı Ağlar (SDN), ağ kararlarını kontrol eden kontrol düzlemini, paket iletimi görevini üstlenen veri düzleminden ayırarak önemli bir paradigma değişimi sağlar. Bu ayrım sayesinde ağ yöneticileri, programlanabilir çözümleri dinamik bir biçimde uygulayabilir ve ağ işlemlerini merkezi olarak kontrol edebilir. Böylece, ölçeklenebilirlik ve esneklik önemli ölçüde artırılır. Bu çalışma, Open vSwitch (OVS), Ryu kontrolcüsü ve ilişkili RESTful API'lerini kullanan SDN tabanlı trafik akışı düzenleme sistemi sunmaktadır. Sistem, farklı uygulama katmanı protokollerinin değişen ihtiyaçlarını karşılayacak biçimde tasarlanmış, Hizmet Kalitesi (QoS) kurallarını dinamik olarak uygulama yeteneğine sahiptir. Protokol bazlı trafik yönetimi sayesinde bant genişliği etkin şekilde tahsis edilir ve ağ gecikmesi azalır. Ryu kontrolcüsü tarafından sağlanan merkezi kontrol, QoS politikalarının dinamik ve esnek uygulanmasını kolaylaştırmakta ve ağ koşullarındaki değişikliklere hızlı cevap verilmesini mümkün kılmaktadır.

Anahtar Kelimeler: Yazılım tanımlı ağlar, hizmet kalitesi, ağ trafiği.

RESEARCH PAPER

Corresponding Author: Sultan Çoğay, cogay@itu.edu.tr

Reference: S. Çoğay, A. Tulum, and G. Seçinti, (2025), "Protocol-Based Traffic Flow Regulation on SDN with Ryu's REST APIs for Testbeds," *ITU-Journ. Wireless Comm. Cyber.*, 2(2), 51–60.

Submission Date: Mar, 13, 2025
Acceptance Date: Sep, 25, 2025
Online Publishing: Sep, 30, 2025

1 INTRODUCTION

Due to tightly integrated decision-making logic and packet forwarding systems, traditional networks are complicated, challenging to operate, and essentially rigid. Management, scalability, and the fast implementation of new network services or policies are seriously challenged by this association. Traditionally, network managers have been obliged to configure every network device separately. Network management is a difficult task; in this way, a device-by-device setup method is error-prone and ineffective. Further aggravating these management challenges is the stiffness of conventional network designs[1]. Usually based on closed, proprietary systems, such networks depend on extensive hardware changes, vendor clearances, or firmware updates to introduce new features or improvements, causing significant delays in adjusting to technology advancements and changing user expectations. The capacity of network operators to rapidly adapt to shifting traffic patterns, security concerns, or economic needs is much limited by this rigidity.

Emerging as a transforming answer to these long-standing issues, software-defined networking (SDN) drastically changed the networking context. Fundamentally, SDN separates the data plane, which handles packet forwarding, from the control plane of the network, which governs logical decision-making. From conventional IP network designs, which usually combine these two planes inside individual network devices, this separation of concerns marks a fundamental change. Separating the control and data planes allows SDN to offer a centralized control architecture under management by a software-based controller[2]. With a complete, worldwide perspective of the network structure and state provided by this centralization, network managers had previously uncommon control over network operations. Unlike conventional networks, where every device makes forwarding decisions based on its local configuration, SDN's centralized controller handles these decisions depending on the whole network perspective. Complicated network chores such as policy enforcement, fault management, traffic engineering, and security management are much simplified by this approach.

Furthermore, routers, switches, and firewalls become simple, programmable forwarding elements in SDN systems that run instructions from the centralized controller. These gadgets greatly simplify and cut costs since they no longer feature sophisticated decision-making software. Instead, they concentrate just on running commands and forwarding packets depending on guidelines and instructions provided by the SDN controller[3]. This change dramatically lowers the administrative load of controlling unique device configurations and removes obstacles to implementing additional network capabilities.

The programming capabilities of SDN are one of its most important advantages. Programmable APIs offered by the SDN controller let network operators add creative network apps and services without changing hardware or depending on vendor-specific firmware upgrades. Through dynamic reconfiguration and efficient resource allocation, this programmability allows quick response to changing network needs such as unexpected traffic loads or the development of new security concerns. Furthermore, flexibility is SDN's capacity to apply abstraction layers. High-level rules and operational goals can be defined by network operators without involving close interaction with the complex hardware details. By enabling managers to better, more intuitively govern and regulate network behavior, abstraction greatly reduces operational complexity and the probability of human mistakes.

Adoption of SDN also helps with simpler integration with new technologies, such as cloud computing, Internet of Things (IoT), and 5G networks, which call for agile, responsive network infrastructures able to sustain dynamic, high-volume traffic. Because it can rapidly assign resources, handle congestion, and react fast to network faults or security events, the centralized control paradigm that SDN offers is intrinsically more suited for managing such dynamic network settings.

SDN's flexibility is especially crucial for implementing Quality of Service (QoS) requirements. To satisfy the several needs of various application protocols, QoS guarantees effective bandwidth allocation, prioritizes vital traffic, and lowers latency. Unique performance criteria for protocols such HTTP, FTP, SMTP, IMAP, and DNS can greatly burden conventional network systems. Real-time applications, for example, call for quick and effective responses, a capacity challenging with traditional network architectures.

All things considered, the complexity, rigidity, and management difficulties of conventional networks arise primarily from their natural design, which firmly connects the control and data planes. According to these concerns, in this paper, we propose a protocol-based traffic flow regulation within an SDN framework and test-bed [1] using Open vSwitch (OVS), the Ryu controller, and its RESTful APIs. The primary objective is to develop a comprehensive traffic management system capable of enforcing QoS rules tailored to application-specific requirements, thereby optimizing network performance and preventing congestion. The proposed system leverages the programmability and centralized management capabilities intrinsic to SDN to effectively address common network management challenges, including bandwidth allocation, congestion control, protocol prioritization, and Differentiated Services Code Point (DSCP) values for traffic differentiation.

Furthermore, we utilize virtual Open vSwitches and Raspberry Pi 4B. The Raspberry Pi is specially chosen for its small size, low cost, and capacity to replicate real-world

1 https://github.com/wtrlili/Protocol-Based-Traffic-Flow-Regulation

network restrictions closely. Representing a different host with each node set to replicate different application-layer protocols, including HTTP, FTP, SMTP, ICMP, and DNS, network namespaces (netns) are used to replicate isolated network environments. These hosts are connected by several virtual switches, therefore guaranteeing a dynamic and realistic simulation environment. Uniform assignment of static IP addresses helped to provide consistent testing and experimentation. To sum up, our contributions in this study are:

- SDN-based traffic flow regulation framework and testbed using Open vSwitch, the Ryu controller, and its RESTful APIs to enforce QoS policies dynamically.
- Applying and validating QoS policies tailored to various protocol demands, including HTTP, FTP, SMTP, DNS, and ICMP.
- A comprehensive network monitoring through packet-level analysis using tools such as TCPDump and Wireshark to ensure QoS policies are effectively enforced.
- Utilizing OpenFlow-compliant flow tables and queue configurations in Open vSwitch, enabling precise traffic management and efficient resource utilization.

The rest of the paper is organized as follows. In Section II, we provide a detailed literature survey. Then, Section III gives the details of the proposed system. In Section IV, we provide our experimental results. Finally, we conclude in Section V.

2 RELATED WORK

SDN has emerged as a transformative technology, providing centralized, programmable control of network functions to effectively meet diverse network requirements. Recent research explores various dimensions of SDN implementation, with an emphasis on performance evaluation, protocol optimization, and integration with emerging technologies, including Network Function Virtualization (NFV) and the Internet of Things (IoT).

Performance evaluations of SDN controllers, explicitly utilizing the Ryu controller are conducted within Mininet simulations [4]. Their research mainly addresses QoS measures of bandwidth and throughput. The outcomes show that the Ryu controller efficiently controls network resources, improving network flexibility and efficiency over conventional networks. POX and Ryu SDN controllers are compared in different network topologies using Mininet and examines their throughput, latency, and jitter performance in terms of QoS [5]. Similarly, in [6], it is compared three SDN controllers regarding packet loss and jitter within load-balancing and firewall scenarios. Their findings indicated the superior performance of Ryu and OVS controllers, effectively reducing packet loss and jitter, outperforming the basic OpenFlow reference controller. This highlights the significance of controller selection in SDN deployments, particularly regarding network reliability and service quality.

In IoT contexts, vertical handover times for IoT devices are analyzed within SDN-enabled heterogeneous network environments [7]. Using Raspberry Pi testbeds, their study finds significant differences in handover latency depending on network interfaces and transition orientations, highlighting the requirement for tailored SDN solutions in IoT environments to get flawless connectivity. It is assessed that whether Raspberry Pi devices are suitable for IoT architecture as SDN-activated switches [8]. The study confirms Raspberry Pi as a affordable, low-cost substitute for SDN in IoT, greatly lowering hardware costs while preserving necessary QoS standards, including bandwidth, delay, jitter, and packet loss. This work underscores the potential for economical yet efficient network management solutions using open-source platforms. It is offered that a lightweight framework that dynamically balances traffic load in SDN-based 5G/6G networks using OpenFlow Group Tables [9]. Experiments based on Mininet and Ryu have demonstrated that the method improves link utilization and throughput.

OpenPATH further expands the application-aware capabilities of SDN through a modular data-plane approach [10]. By supporting network function chaining and parallel execution of network functions (NFs), the framework increases throughput, reduces latency, and most effectively uses resources. By including programmable logic right inside the dataplane, the design of Open PATH solves constraints in traditional SDN systems and lowers controller overhead, hence improving general network performance. It is investigated live migration capabilities of virtualized Evolved Packet Core (EPC) network components using both container-based and Virtual Machine (VM)-based virtualization platforms [11]. They demonstrated that while container platforms achieved shorter migration completion times, VM-based systems provided reduced service downtime. Insights are crucial for optimizing NFV deployments within mobile networks, particularly in 5G contexts requiring stringent QoS guarantees. It is explored that network slicing in 5G core networks using the Ryu SDN controller [12]. Their lightweight implementation leveraged existing Linux kernel tools for resource allocation and effectively demonstrated performance isolation among slices. This approach highlights the flexibility of SDN in meeting diverse application demands within a shared network infrastructure, reinforcing SDN's role in facilitating next-generation network services.

From IoT to sophisticated virtualized mobile networks, these studies collectively show the revolutionary potential of SDN in obtaining improved performance, flexibility, and cost-efficiencies across many networking environments.

3 PROTOCOL-BASED TRAFFIC FLOW REGULATION

The proposed system consists of a Raspberry Pi 4B running Open vSwitch, with traffic regulation managed by the Ryu controller. The architecture separates the data and control planes, where OVS handles packet forwarding and Ryu enforces flow rules based on protocol-specific QoS requirements. We explain our proposed flow regulation framework in three main parts. The details are given in the sections below.

3.1 Data Model

This section highlights how traffic is managed using flow tables, REST API-based QoS configurations, and IPv4 traffic classification to improve network performance. The Flow Table in the system ensures efficient traffic management by defining rules based on match fields (such as IP addresses and protocols), actions (like forwarding or dropping packets), and counters for monitoring traffic. The REST API Data Models enable dynamic QoS configuration, including queue assignments and bandwidth management, by using parameters such as port names, queue IDs, and packet-matching rules. The IPv4 protocol is essential for sorting traffic and enforcing QoS. It uses fields like DSCP to set priorities, source, and target addresses to route traffic, and the protocol field to decide how to handle packets.

3.1.1 Flow Table

The data plane components act as simple forwarding devices within the system. However, the system would be inefficient if all the actions relied solely on the communication between the forwarding device and controller. Thus, a Flow Table structure is used to maintain a more efficient organization. These tables determine how network traffic is processed and forwarded within the topology based on predefined criteria. If a packet matches the rule defined in the table, the forwarding device performs the associated action on the packet. However, if there is no matching entry, the system advises the controller to take appropriate action. While flow tables do not provide switches' decision-making capabilities, they allow switches to execute actions based on rules pre-configured by the controller.

The flow tables are crucial in enforcing QoS rules and directing traffic according to protocol-specific requirements. For example, two hosts can send different HTTP requests to the same server. The switch must be able to direct the request to the server and then send responses to the respective hosts without any mismatches. The structure of the flow table can be listed as follows [13].

1. **Match Fields:** These fields define the criteria for matching incoming packets. The following values can be used for matching the packets with the actions:
   - Source IP address
   - Destination IP address
   - Transport-layer Protocol
   - Port Number
2. **Actions:** Once a packet matches the defined criteria, specific actions are performed on the packet. The commonly used actions are:
   - Forwarding to a port or switch
   - Mapping packets to QoS queue for prioritized processing
   - Drop packets for congestion management
3. **Counters:** Each flow entry is associated with a counter that monitors traffic for the matched criteria. These metrics include:
   - Packet Count
   - Byte Count

The flow table in Table 1 shows Rule R1, which forwards packets to a specific port. The use of wildcard characters in match fields (e.g *) enables flexible rule matching and reduces the complexity of rule definitions. In the suggested system, the flow tables are managed dynamically by the Ryu controller via the OpenFlow13 protocol. Moreover, the prioritization protocols are applied via the RESTful API of the Ryu controller.

Table 1 Example Flow Table

Rule	Field 1	Field 2	Priority	Action	Counter
R1	1**	0**	1	Forward 1	25
R2	0**	1**	2	Forward 2	35
R3	10*	11*	2	Drop	25
R4	01*	11*	3	Forward 4	17
R5	00*	10*	3	Forward 5	32

3.1.2 REST API Data Models

Queue Configuration: The Queue Configuration Model represents how Quality of Service (QoS) queues are defined and implemented in Open vSwitch to regulate network traffic. The fields used for the definition can be enlisted as:

- **port_name:** The port on the switch where the queues are applied.
- **type:** Defines the queuing discipline (e.g., linux-htb for hierarchical bandwidth management)
- **max_rate:** The maximum bandwidth allocated to the port (in bits per second).
- **queues:** It consists of a list of queue configurations.

- **queue_id:** A unique identifier
- **max_rate or min_rate:** Specifies the minimum or maximum bandwidth allocated to the queue to ensure traffic constraints.

QoS Rule Model: The QoS Rules Model defines the policies used to regulate network traffic in accordance with application-specific requirements. The Ryu controller dynamically configures these rules and implemented in Open vSwitches to enforce prioritization, bandwidth allocation, and trafficking.

- **match:** Defines the criteria for identifying packets to which QoS policies should be applied
  1. **nw_src:** Matches packets by source IP address.
  2. **tp_dst:** Matches packets by the port number.
  3. **nw_proto:** Matches packets by the protocol.
- **actions:** Specifies the actions to be applied to packets that match the conditions
  1. **queue:** Assigns the packet to a queue for prioritized processing.
  2. **mark:** Sets the DSCP value in the packet's IP header **ryubook_qos**

3.1.3 IPv4

The IPv4 protocol is fundamental as it provides the essential fields for identifying and classifying traffic. The fields on the protocol, such as DSCP (Differentiated Services Code Point), Protocol, and Source/Destination IP addresses, are integral for the Ryu controller to enforce QoS policies.

The ToS (Type of Service) field in the IPv4 header includes DSCP, which is specifically used for marking the packets to indicate their priority level. For example, DSCP value of 46 (Expedited Forwarding) is often used for latency-sensitive traffic, whereas a value of 0 (Best Effort) is assigned to non-prioritized traffic [14].

The source and destination address fields identify traffic endpoints to determine the path of the traffic, whereas the protocol field specifies the transport protocol (e.g., TCP, UDP) used for transmitting the packet. The fields enable the system to classify packets and then apply prioritization techniques for ensuring QoS.

3.2 Network Architecture

The network architecture, as illustrated in the Figure 1 represents the fundamental characteristic of SDN by decoupling the data plane and the control plane. This design facilitates centralized traffic management and application of QoS policies. At the core of the system, the Ryu controller acts as the centralized management unit. The controller communicates with the data plane components via the Southbound API OpenFlow 1.3 protocol. It is responsible for managing the network behavior by enforcing the flow rules.

The data plane consists of seven Open vSwitches, Bridge 1 to Bridge 7. These switches are interconnected in a ring topology form to forward the traffic between hosts and servers based on the rules that lie in the Flow Table. The hosts simulate end-user devices to generate traffic. The servers represent various applications to allow the system to enforce protocol-specific QoS rules. These include:

- **HTTP Server** For web traffic IP: 10.0.0.1
- **FTP Server** For file transfer services IP: 10.0.0.3
- **DNS Server** For domain name resolution IP: 10.0.0.5
- **SMTP Server** For email trafficing IP: 10.0.0.6
- **Two Hosts** For pinging and ICMP-based communication IP: 10.0.0.2 and 10.0.0.4

Furthermore, the architecture supports the addition of new hosts, servers, or switches without requiring major reconfiguration. These demonstrate its adaptability to changing network requirements. The hardware design of the system is centered around a single Raspberry Pi 4B (8GB RAM), which serves as the platform for deploying multiple OVS. 4 Python Scripts are used for initializing and configuring Ryu manager to interact with RESTful API:

- **qos_simple_switch_13.py:** Ryu application for implementing a basic OpenFlow 1.3 switch. Handles forwarding and QoS rules by working with flow tables and queues.
- **rest_qos.py** Exposes a RESTful API to interact with the Ryu controller for managing QoS policies. It allows administrators to define, update and remove QoS rules and queue configurations.
- **rest_conf_switch.py** Provides RESTful APIs to manage and configure OF switches.
- **rest_topology.py** Enables RESTful APIs to retrieve and manage network topology information. It can dynamically discover the interconnections between the switches and hosts [15].

In this step, we implement three main algorithms: Initialization, AddFlow, and EventHandling. Algorithm Initialization initializes the MAC-to-port mapping dictionary and installs a default "table-miss" flow on the OpenFlow switch. This default flow rule forwards all unmatched packets to the controller for further handling. Initially, it sets up an empty dictionary for MAC-to-port mappings, retrieves datapath and related protocol objects, and defines a low-priority

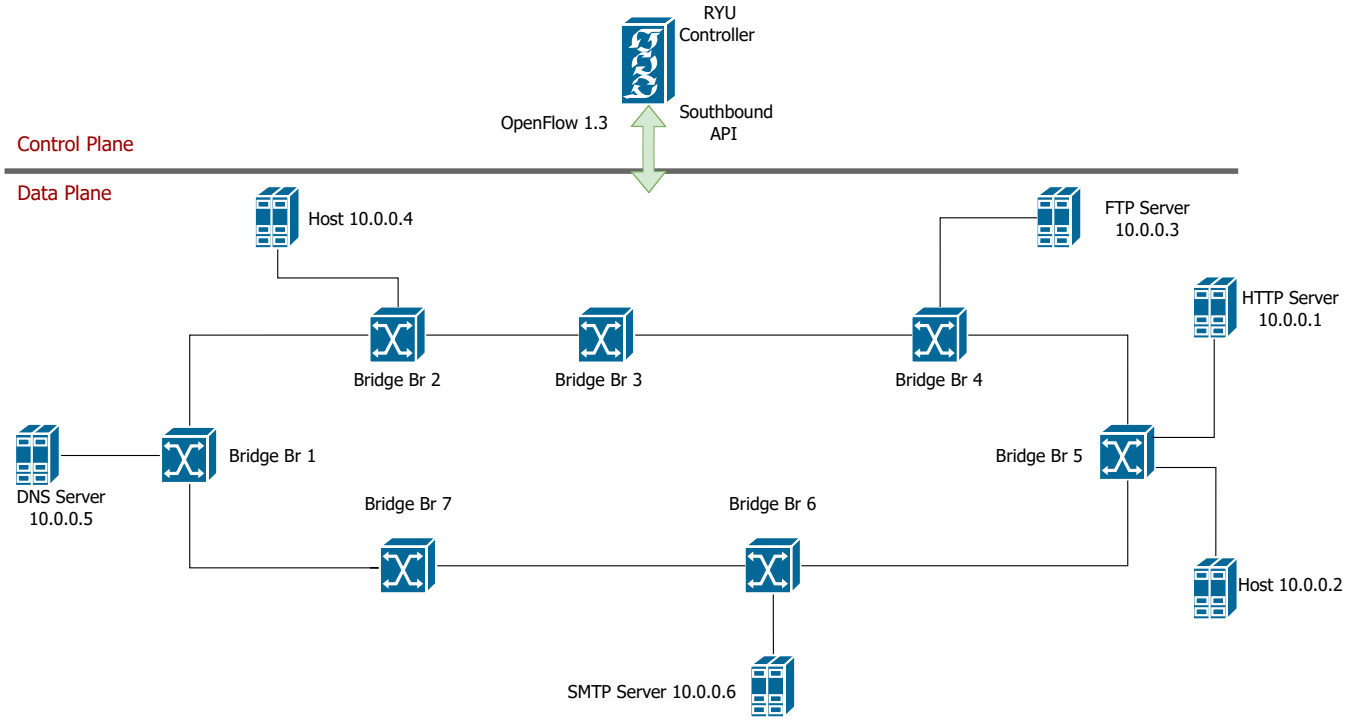

Fig. 1 Network architecture.

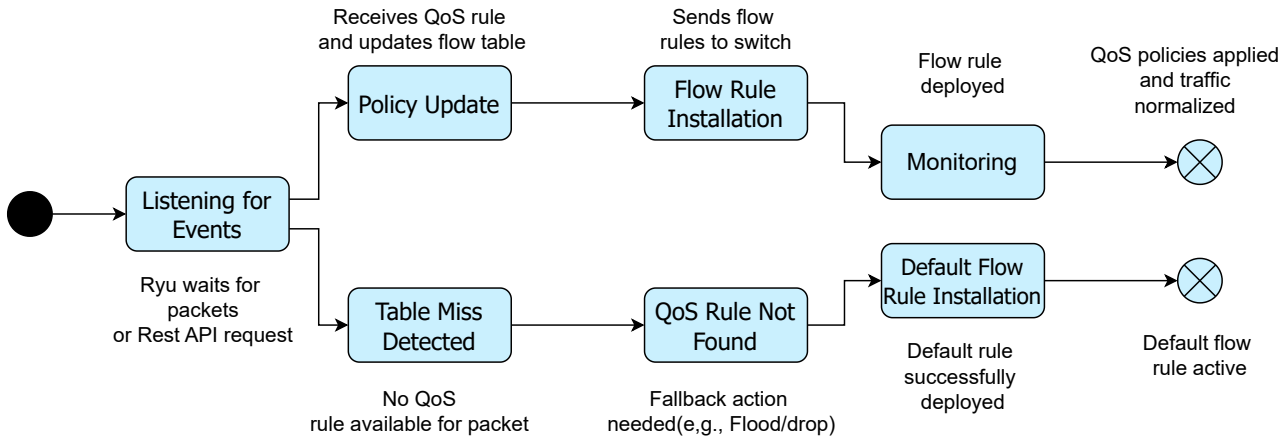

Fig. 2 State machine diagram.

flow entry matching all incoming packets. This ensures effective management of unknown packets by directing them to the controller.

Algorithm AddFlow describes the process of adding flow rules to an Open vSwitch. It constructs and sends a flow modification message containing specified match criteria and actions. The algorithm checks for an optional buffer ID; if provided, the flow modification message includes this ID. The flow rule defines precisely which packets it applies to and the actions to perform, such as forwarding or applying QoS policies.

Algorithm EventHandling manages incoming packets through Packet-In events. It extracts packet data, learns source MAC addresses, and updates MAC-to-port mappings to determine packet forwarding decisions. If the destination MAC address is already known, the algorithm installs a flow rule to handle similar packets directly in the future, reducing overhead on the controller. The packet is flooded to all switch ports if the destination MAC is unknown. This approach optimizes network performance and efficiently handles packet forwarding decisions.

3.3 Dynamic Behavior

The dynamic behavior of the system is illustrated by two complementary diagrams: State Machine Diagram, and QoS Activity Diagram. Each of these provides a view of the interaction between system components and QoS enforcement.

The diagram in Figure 2 illustrates the key states of the Ryu Controller as it operates and interacts with Open vSwitch (OVS). The controller adapts to incoming traffic and implements QoS policies when necessary. After initialization, the controller begins listening for incoming packets or REST API requests. Subsequently, it handles Packet-In events triggered when no matching flow rule exists for the packet. A table miss is detected if a packet has no matching rule. In the absence of a specific QoS policy, a fallback action, such as fallback or drop, is applied. When a QoS policy is added via REST API, the controller updates the respective flow table. The controller sends the appropriate flow rule to Open vSwitch to enforce QoS. Finally, the controller monitors network traffic to validate the application and QoS policies.

Secondly, upon packet arrival, the switch first analyzes the packet header to identify its protocol type (e.g., HTTP), which is crucial for determining the traffic class and whether special QoS handling is necessary. Next, the controller verifies whether a specific QoS rule exists for this traffic; if no matching rule is found, default QoS rules are applied. If a matching rule is present, the controller applies the set QoS policy that is unique to that protocol. These QoS rules control bandwidth allocation, delay, and priority, therefore giving critical traffic preference over less critical traffic. Applying the necessary QoS rules directs the packet towards its intended recipient. Using monitoring tools like tcpdump or Wireshark for a thorough investigation, the system constantly checks network traffic to verify adherence to QoS rules throughout this process. In the end, the packet reaches its target completely according to the imposed QoS criteria. The activity diagram provides a detailed view of how the system enforces QoS policies for an HTTP GET request in Figure 3.

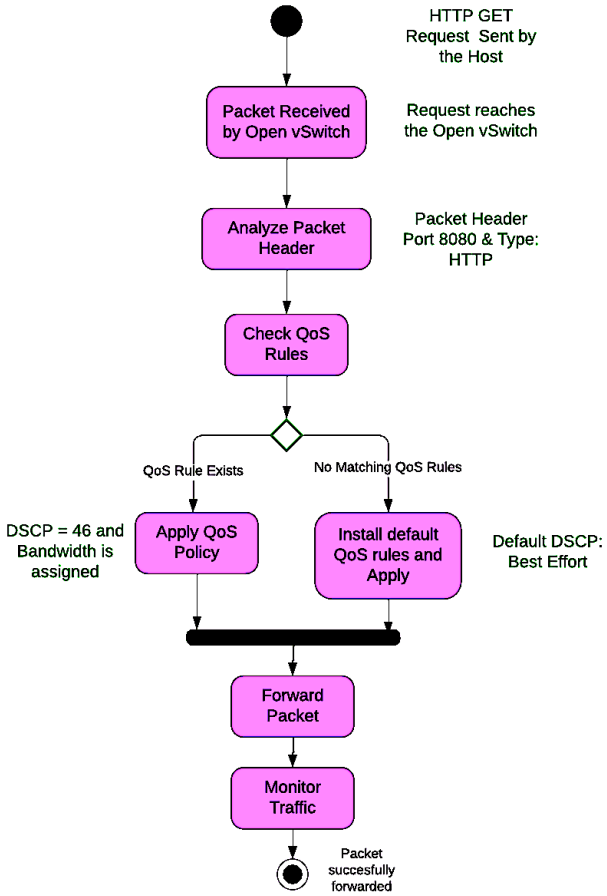

Fig. 3 Activity diagram for QoS enforcement.

4 PERFORMANCE EVALUATION

Implementing the data plane within the network topology is mostly dependent on the Raspberry Pi 4B (8 GB RAM). The Raspberry Pi 4B has various fundamental parts to enable functioning and simple deployment. The Raspberry Pi is connected to a monitor using a micro HDMI connection, therefore granting Graphical User Interface (GUI) access. Storage and operating system functionality are provided by a 64 GB microSD card, which contains the Raspberry Pi OS (64-bit) along with the Open vSwitch software. In the software environment, various network management and simulation tools are employed to conduct experiments effectively. Open vSwitch version 2.15 is selected due to its

compatibility and proven stability with the Ryu controller. The Ryu SDN controller supports Open vSwitch versions that extend 2.10 to 2.15. Hence, version 2.15 is the best option because of its improved compatibility and steady performance. These software choices are specifically selected to ensure seamless integration, stability, and compatibility between the controller and the network switch infrastructure.

The protocol serves the listed packages:

- **HTTP Server:** Hosted on H1. It runs a Python-based HTTP server module. The configuration includes two key types of HTTP traffic:
  - A *maximum bandwidth of 5 Mbps* is allocated for general HTTP traffic on **port 80**. *DSCP value is set to 10* for indicating standard priority for regular operations
  - A *minimum bandwidth of 2 Mbps* is reserved for administrative HTTP operations on **port 8080**. The *DSCP value is set to 46*, which is used for expedited forwarding (EF) to ensure high-priority and low-latency delivery.
- **FTP:** Hosted on H3. Uses vsftpd FTP server to support passive connections. Traffic is divided based on:
  - Data transfer is done via **port 50000** in passive connections. The *maximum rate bandwidth of 10 Mbps* is allocated. The *DSCP value is set to 16* to ensure moderate priority for FTP data transfer to make sure to balance its requirements with other network traffic.
  - **Port 21** is connected for establishing the connection between the host and the server. The *minimum bandwidth of 128 Kbps*is reserved. The *DSCP value is set 8* for indicating low priority for control traffic since it typically requires less bandwidth.
- **SMTP** Hosted on H6. Uses Postfix for test email transmission over the network.
  - The *minimum bandwidth of 512 Kbps and DSCP value of 10* is selected moderate priority. It ensures the email is delivered without delay under normal network conditions on **port 25**.
- **DNS** Hosted on H5. Uses bind9 for local domain name resolution requests.
  - A *minimum bandwidth of 256 Kbps* is allocated for DNS query responses. The *DSCP value is set to 24* to ensure moderate priority for DNS traffic on **port 53**. A quick resolution is important but not as critical as high-priority tasks like administrative HTTP traffic.

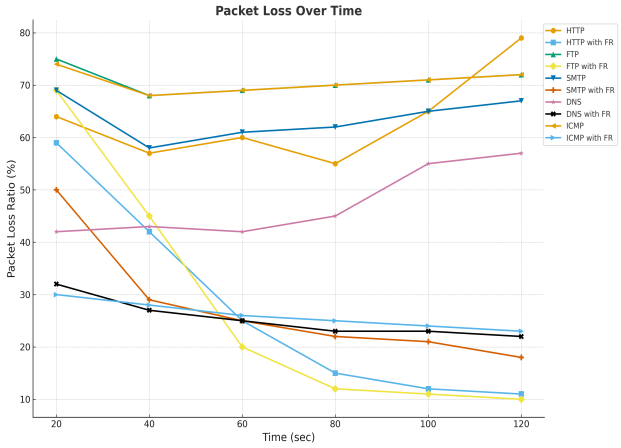

Fig. 4 Packet loss results.

  - **ICMP** Hosted on H2. Used for network diagnostics.
    - The *minimum bandwidth of 64 Kbps and DSCP value of 8* is selected to indicate low priority. Rather than enforcing the policy on a specific port, it's applied to all ICMP traffic that arises from host 2.

The packet loss analysis results, as shown in Figure 4, demonstrate substantial improvements following the application of flow regulation (FR) via QoS policies across various network protocols (HTTP, FTP, SMTP, DNS, and ICMP) over a monitoring period of 120 seconds, with measurements taken every 20 seconds. Significant improvements in packet loss ratios are observed in protocols after implementing flow regulation. After using FR, FTP traffic shows the most affected drop—from over 70% to almost 10%, suggesting significant improvement in handling data-intensive transfers. Likewise, the SMTP protocol shows a notable drop from roughly 55% down to roughly 25%, underscoring how well FR guarantees constant and dependable email transmission. From about 45% to about 20%, DNS protocol packet loss drops indicate significant increases in network dependability for moderate-priority services. ICMP also clearly benefits from packet loss reduction ranging from roughly 65% to roughly 20%, hence confirming the effectiveness of the framework even for lower-priority diagnostic traffic. While showing progress, HTTP traffic maintains much higher packet loss than other protocols, thereby lowering from about 60% to about 40%. For web-based systems, this implies more optimizations could be helpful.

Figure 5 clearly illustrates the effectiveness of the implemented QoS policies for different network protocols. The allocated bandwidth for the HTTP protocol ranged from a minimum of around 2 Mbps to a high of 5 Mbps, and the measured bandwidth aligned at about 4.8 Mbps. This alignment suggests that QoS regulations efficiently con-

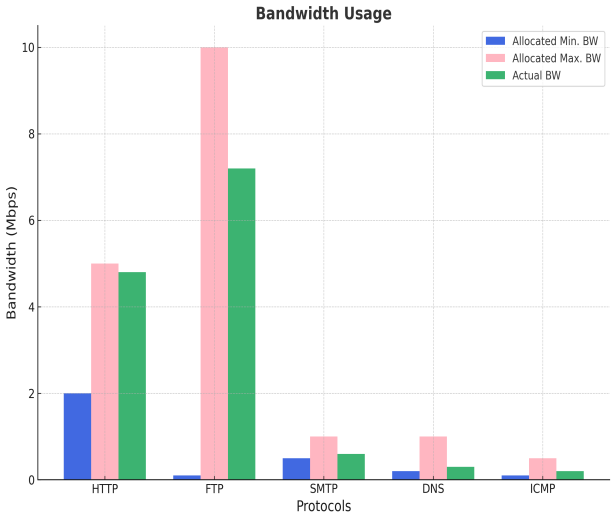

Fig. 5 Bandwidth usage results.

trol HTTP traffic without running over the allocated resources. Using roughly 7.24 Mbps, FTP protocol traffic closely matched its permitted maximum bandwidth of 10 Mbps. This shows the system's effective use of resources and adherence to the desired QoS standards. With SMTP using 0.57 Mbps, DNS 0.28 Mbps, and ICMP traffic having the lowest actual utilization, protocols including SMTP, DNS, and ICMP revealed reduced bandwidth usage. These findings confirm the relevance of the given resources by stressing protocol prioritizing and effective bandwidth limitation depending on relative priority.

5 CONCLUSION

This work presents a complete SDN-based traffic flow control system and test-bed using OVS, the Ryu controller, and related RESTful APIs to enforce QoS requirements dynamically. The experimental evaluation dramatically enhances network performance and dependability, showing that the proposed solution addresses conventional network constraints. In many application-layer protocols, including HTTP, FTP, SMTP, DNS, and ICMP, the applied protocol-based management efficiently allocates bandwidth resources and significantly lowers packet loss. The bandwidth analysis proves that the QoS system effectively used the assigned bandwidth resources, so it tightly matched observed bandwidth consumption with predetermined minimum and maximum thresholds. Effective management of protocol prioritizing revealed the valuable advantages of centralized control given by the Ryu controller. Moreover, QoS interventions showed clear benefits regarding packet loss metrics; packet loss ratios across all protocols showed significant decreases, highlighting the framework's capacity to control high-priority network traffic and increase general network responsiveness. The results confirm the adaptability of SDN architectures for dynamically changing network resources depending on real-time needs and guarantee the best network operation by validating their scalability and flexibility.

ACKNOWLEDGEMENTS

This work is supported by The Scientific and Technological Research Council of Turkey (TUBITAK) 1515 Frontier R&D Laboratories Support Program for BTS Advanced AI Hub: BTS Autonomous Networks and Data Innovation Lab. Project 5239903 and Istanbul Technical University.

REFERENCES

[1] M. Ariman, G. Seçinti, M. Erel, and B. Canberk, "Software defined wireless network testbed using raspberry pi of switches with routing add-on," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, 2015, pp. 20–21. DOI: 10.1109/NFV-SDN.2015.7387397.

[2] A. Liatifis, P. Sarigiannidis, V. Argyriou, and T. Lagkas, "Advancing sdn from openflow to p4: A survey," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–37, 2023.

[3] I. M. Varma and N. Kumar, "A comprehensive survey on sdn and blockchain-based secure vehicular networks," *Vehicular Communications*, vol. 44, p. 100 663, 2023.

[4] H. Babbar and S. Rani, "Performance evaluation of qos metrics in software defined networking using ryu controller," in *IOP conference series: materials science and engineering*, IOP Publishing, vol. 1022, 2021, p. 012 024.

[5] C. Jayawardena, J. Chen, A. Bhalla, and L. Bu, "Comparative analysis of pox and ryu sdn controllers in scalable networks," *arXiv preprint arXiv:2504.12770*, 2025.

[6] M. F. Monir, A. F. Hasan, M. M. Hoque, T. Ahmed, and F. Granelli, "Benchmarking network functionality: Performance evaluation of sdn controllers on different network functions," in *2024 IEEE 100th Vehicular Technology Conference (VTC2024-Fall)*, IEEE, 2024, pp. 1–5.

[7] R. Nozaki, L. Guillen, S. Izumi, T. Abe, and T. Suganuma, "A study on heterogeneous network switching time in iot environments using sdn," in *2021 IEEE 10th Global Conference on Consumer Electronics (GCCE)*, 2021, pp. 859–860. DOI: 10.1109/GCCE53005.2021.9622084.

[8] M. N. Elham, S. M. Sam, A. Azizan, Y. M. Yusof, N. Mohamed, and N. Ahmad, "Performance evaluation of sdn-enabled switching system for iot infrastructure," in *2023 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*, 2023, pp. 136–142. DOI: 10.1109/IAICT59002.2023.10205953.

[9] L. Agrawal and A. Mondal, "Tron: Traffic management and resource allocation for sdn-enabled 5g/6g networks," in *2025 IEEE 31st International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2025, pp. 1–2. DOI: 10.1109/LANMAN66415.2025.11154616.

[10] P. Krishnan, S. Duttagupta, and R. Buyya, "Openpath: Application aware high-performance software-defined switching framework," *Journal of Network and Computer Applications*, vol. 193, p. 103 196, 2021.

[11] S. Ramanathan et al., "A comprehensive study of virtual machine and container based core network components migration in openroadm sdn-enabled network," *arXiv preprint arXiv:2108.12509*, 2021.

[12] P.-T. Tivig, E. Borcoci, M.-C. Vochin, I. A. M. Balapuwaduge, and F. Y. Li, "Slicing 5g core network based on the ryu sdn controller for everything as a service," in *2023 26th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, 2023, pp. 1–6. DOI: 10.1109/WPMC59531.2023.10338844.

[13] Open vSwitch Documentation Team. "Tutorials — open vswitch 3.5.90 documentation." Retrieved January 14, 2025. [Online]. Available: https://docs.openvswitch.org/en/latest/tutorials/.

[14] Cisco. "Cisco nexus 1000v quality of service configuration guide, release 4.0(4)sv1(1)." Retrieved January 20, 2025. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus1000/sw/4_0/qos/configuration/guide/nexus1000v_qos.html.

[15] Faucetsdn, *Ryu sdn framework repository*, https://github.com/faucetsdn/ryu/tree/master/ryu/app, Accessed: January 14, 2025, n.d.