فصلنامه سیستم‌های پردازشی و ارتباطی چندرسانه‌ای هوشمند
Intelligent Multimedia Processing and Communication Systems (IMPCS)

Spring, 2022, 3 (1), 9-23
DOR: 20.1001.1.27832570.1399.1.2.2.8

Received:14 March 2022
Accepted: 27 Apr 2022

QoS Management Solution in Software Defined Networking Using Ryu Controller

Amir Joz Ashoori[1], Shiva Karimi[2*]

1. Msc. Student, Faculty of Electrical and Computer Engineering, Zanjan Branch, Islamic Azad University, Zanjan, Iran. a.ashoori7@gmail.com

2. Assistant Professor, Faculty of Electrical and Computer Engineering, Zanjan Branch, Islamic Azad University, Zanjan, Iran.*(Corresponding Author)* Shkarimi2013@yahoo.com

Abstract

Introduction: Enterprise networks are increasingly becoming larger and more dynamic due to vast deployments of virtualization technologies. Consequently, the explosion of new network applications and services has strained the capabilities of traditional networking architecture in terms of scalability, agility, and efficient traffic management. SDN (Software Defined Networking) is a novel approach to build networks in which control logic is decoupled from data forwarding in order to enable programmability and ease of configuration across the entire network. The centralized control in SDN provides a global view of the entire network resources and their performance which enables the innovation of new service models. This paper demonstrates the implementation of SDN in a sample data center network topology using Mininet and the RYU controller, followed by employing policy-based network management and a differentiated service mechanism for guaranteeing the QoS for different classes of traffic. The proposed framework is a foundation to develop an enterprise-level network control and management product.

Method: The approach of this paper is an implementation of a software-based architecture in the topology of a data center. It manages and guarantees the quality of service, using network policy-oriented management and service quality methods. The presented framework is an expandable infrastructure to solve the challenge of dynamic and agile management in the network of data centers and virtualization and cloud processing service providers.

Findings: With the implementation done, h1r1 server node listens on ports 5001, 5002, 5003 with UDP protocol. The h1r4 client node sends 1Mbps UDP traffic to port 5001, 300Kbps UDP traffic to port 5002, and 600Kbps UDP traffic to port 5003 of the h1r1 server. The results obtained using the IPerf3/JPerf tool show that for traffic marked with AF41 code sent to port 5003, minimum bandwidth of 500Kbps and for traffic marked with AF31 code sent to port 5002, minimum bandwidth of 200Kbps is guaranteed. Is. When sending traffic of AF classes, the bandwidth of the best-effort traffic sent to port 5001 is limited.

Conclusion: Guaranteeing full quality of service for all types of applications is not possible with the current network architecture based on the best-effort model. Different applications have different service needs that require dynamic management of network resources. In this article, a solution based on SDN architecture was presented for service quality management, which uses a differentiated service model. Differentiated service mechanism allocates resources based on different traffic classes. In this method, all streams belonging to a class are routed equally. The results obtained from the simulations show the optimal performance of the introduced framework in meeting the needs of traffic flows and optimal and maximum use of network resources.

Keywords: Software Defined Networking, SDN Controller, Ryu, OpenFlow, Quality of Service, QoS Management, Policy- based Network Management.

فصلنامه سیستم‌های پردازشی و ارتباطی چندرسانه‌ای هوشمند
Intelligent Multimedia Processing and Communication Systems (IMPCS)
انجمن علمی تجارت الکترونیکی ایران

مدیریت کیفیت سرویس در شبکه‌های مبتنی بر نرم‌افزار با استفاده از کنترلر Ryu

تاریخ دریافت: ۱۴۰۰/۱۲/۲۳
تاریخ پذیرش:۱۴۰۱/۰۲/۰۷

سال سوم، بهار ۱۴۰۱
شماره اول، صص: ۹ – ۲۳

امیر جزء آشوری[۱]، شیوا کریمی[۲*]

۱. دانشجوی کارشناسی ارشد، دانشکده مهندسی برق و کامپیوتر، واحد زنجان، دانشگاه آزاد اسلامی، زنجان، ایران. a.ashoori7@gmail.com
۲. استادیار، دانشکده مهندسی برق و کامپیوتر، واحد زنجان، دانشگاه آزاد اسلامی، زنجان، ایران. Shkarimi2013@yahoo.com

چکیده: با ظهور و گسترش تکنولوژی‌های مجازی‌سازی در سال‌های اخیر، تعداد سرورها و حجم ترافیک مراکزداده به طرز چشمگیری افزایش یافته‌است. ازطرف دیگر، رشد و توسعه سرویس‌ها و برنامه‌های شبکه، موجب افزایش پویایی و غیرقابل پیش‌بینی شدن الگوهای ترافیک شده و نیازمندی‌های کیفیت سرویس را افزایش‌داده‌است. رویکرد نوین شبکه‌های مبتنی بر نرم‌افزار با هدف تسهیل کنترل و مدیریت شبکه و برقراری امکان نوآوری از طریق برنامه‌نویسی اجزاء شبکه به‌وجودآمد تا راه‌حلی امکان‌پذیر برای مشکلات معماری سنتی ارائه‌دهد. با جدایی سطوح داده و کنترل، روش‌های مدیریتی جدیدی ممکن‌می شود که انعطاف‌پذیری، کنترل بیشتر و نوآوری مدل‌های سرویس را به همراه خواهنداشت. رویکرد ارائه‌شده در این مقاله، برای اجرای معماری مبتنی بر نرم‌افزار در توپولوژی یک مرکز داده، استفاده از مدیریت سیاست‌محور شبکه و روش کیفیت سرویس خدمات متمایز است. چهارچوب ارائه شده، زیر ساختی قابل‌توسعه در جهت حل چالش مدیریت پویا و چابک در شبکه مراکز داده و ارائه‌دهندگان خدمات مجازی‌سازی و پردازش ابری است.

واژه‌های کلیدی: شبکه مبتنی بر نرم‌افزار، مجازی‌سازی، اوپن‌فلو، کنترلر ریو، کیفیت سرویس، مدیریت سیاست‌محور، خدمات متمایز.

۱. مقدمه

امروزه تکنولوژی شـــبکه‌های کامپیوتری، ســـومین دوره تحول خود را تجربه‌می‌کند. در اولین تحول، سوئیچینگ مداری جایگزین سوئیچینگ بســته‌ای شــد. در دومین مرحله، شــبکه ســیمی با مدل‌های بی‌ســیم جایگزین شد. در سومین انقلاب، مدل‌های سخت‌افزاری شبکه به مدل های نرم‌افزاری تبدیل‌شدند. این تحول، بر پایه مجازی‌سازی به‌وجودآمد که باعث‌شــد تا منابع و تجهیزات ســخت‌افزاری با مدل‌های نرم‌افزاری جایگزین‌شــود و بدین ترتیب بســیاری از محدودیت‌ها و موانع مرتبط با سخت‌افزار از میان برداشته‌شود[1].

با ظهور و گسترش تکنولوژی‌های مجازی‌سازی و همچنین افزایش روزافزون توان پردازش و ذخیره سازی سخت‌افزارها، هر سرور فیزیکی اکنون می‌تواند به‌طور متوســط تعداد ۱۰ تا ۲۰ ماشــین مجازی را میزبانی‌کند. همچنین، توان محاسباتی و ذخیره سازی مازاد خود را نیز در قالب ســرویس‌های پردازش ابری در اختیار طیف گســترده‌تری از کاربران قراردهد. از این جهت در سال‌های اخیر، تعداد سرورها و حجم ترافیک مبادلاتی در مراکز داده افزایش چشــمگیری یافته‌اســت[2]. ازطرف دیگر، رشد و توسعه سرویس‌ها و برنامه‌های شبکه و هم‌گرایی ترافیک حاوی صــوت، داده و تصــویر، موجب افزایش پویایی و غیرقابل پیش‌بینی شدن الگوهای ترافیک شده و نیازمندی‌های کیفیت سرویس را افزایش‌داده‌اســت. بنابراین ســه روند کلی افزایش ظرفیت شــبکه، افزایش تقاضا در نتیجه افزایش بار شبکه و در نهایت پیچیده‌تر شدن الگوهای ترافیک، باعث‌شــده‌اند تا ارائه‌دهندگان و بازیگران شــبکه، رویکردها و روش‌های ســنتی معماری شــبکه را مجدداً مورد ارزیابی قراردهند[3]. شــبکه‌های ســنتی به دلیل معماری ایســتا و پیچیده، ناسازگاری در اعمال سیاست‌های شبکه، عدم مقیاس‌پذیری و وابستگی به تجهیزات فروشــندگان، در پاســخ به روندهای نوین شــبکه، محدودیت‌های اساسی در حوزه‌های مختلف دارد.

رویکرد نوین شبکه‌های مبتنی بر نرم‌افزار با هدف تسهیل کنترل و مدیریت شــبکه و برقراری امکان نوآوری از طریق برنامه‌نویســی اجزاء شبکه به‌وجودآمد تا راه حلی امکان‌پذیر برای مشکلات معماری سنتی ارائه دهد. SDN، عملیات ســوئیچینگ را مابین دو ســطح داده[1] و کنترل[2]، تقســیم و از یکدیگر جدامی‌کند. بدین ترتیب کنترلر مرکزی با داشــتن دانش کلی از اجزاء شــبکه، می‌تواند با اســتفاده از پیام‌های نرم‌افزاری، نحوهٔ مــدیریــت جریــان‌هــای ترافیــک را بــه ســوئیچ‌هــا آموزش‌دهد[4]. با جدایی ســطوح مدیریت، ارســال و کنترل، روش‌های مدیریتی جدیدی ممکن‌می‌شــود که انعطاف‌پذیری، کنترل بیشــتر و نوآوری مدل‌های ســرویس را به همراه خواهندداشــت. جدایی ســطوح، امکان برنامه‌ریزی شبکه را از طریق رابط‌های برنامه‌نویسی و پلتفرم‌های نرم‌افزاری بــا مزیــت پردازش توزیعی، فراهم‌می‌کنــد. بــه‌علاوه، بــا متمرکزشــدن توابع مدیریت و کنترل، طراحی و عملیات شــبکه نیز تسهیل‌می‌شود.

معرفی اســتاندارد اوپن‌فلو را می‌توان نقطه آغاز معماری مبتنی بر نرم‌افزار دانست. استاندارد اوپن‌فلو در دانشگاه استنفورد و با هدف ارائه محیطی باز به منظور تحقیق و توسعه ارائه شد. مفهوم SDN یک سال بعد از اوپن‌فلو و در ســال ۲۰۰۹ به دنیای شــبکه معرفی‌شــد. درســال ۲۰۱۱ گروهی از اپراتورها، فرآهم‌کنندگان خدمات و فروشــندگان گرد هم آمدند و بنیاد شبکه‌های آزاد ONF[3] را بنیان‌گذاری‌کردند که امروزه بــه عنوان مهمترین مرجع اســتانــدارد‌ســازی در حوزه SDN شناخته‌می‌شود[2].

در سال‌های اخیر، شبکه SDN رشد فوق‌العاده‌ای داشته و تقریباً در تمامی انواع مختلف شبکه، مانند مراکز داده، ارتباط بی سیم و اینترنت اشیاء، شبکه ناحیه‌گستر و موبایل، استقرار یافته‌است[5]. طبق گزارش سیسکو از روندهای شبکه در سال ۲۰۱۹، حدود نیمی از شبکه‌های مراکز داده، ناحیه گستر و دسترسی، از SDN بهره‌می‌برند.

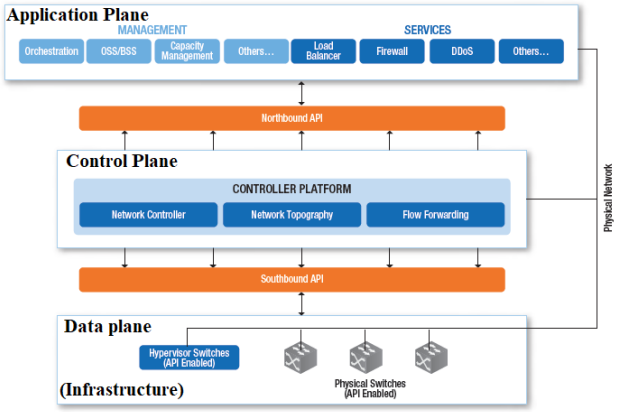
اگرچه معماری مبتنی بر نرم‌افزار راه‌حلی کلی برای مشکلات عمده شبکه‌های سنتی ارائه‌می‌دهد، نحوه پیاده‌سازی و ادغام آن با تجهیزات تولیدکنندگان، اســتانداردســازی پروتکل‌ها و چهارچوب‌های مرتبط با کنترلر چالش‌هایی جدید در حوزه شبکه می‌باشند. بهره‌برداری از SDN در سطح تجاری، مستلزم ارائه راه‌کارهایی است که اجزاء مختلف مثل سوئیچ و کنترلر را به‌طور مناسب، طراحی و در کنار هم به‌کارگیرد.

رویکرد ارائه‌شــده در این مقاله، برای اجرای معماری مبتنی بر نرم‌افزار در توپولوژی یک مرکز داده و مدیریت و تضــمین کیفیت سرویس، استفاده از مدیریت سیاست محور شبکه[4] و روش کیفیت ســرویس خدمات متمایز[5] اســت. چهارچوب ارائه‌شــده، زیرســاختی قابل‌توسعه در جهت حل چالش مدیریت پویا و چابک در شبکه مراکز داده و ارائه‌دهندگان خدمات مجازی‌سازی و پردازش ابری است.

۲. معماری SDN

براســاس تعریف بنیاد شــبکه‌های باز، SDN به معنی جدایی فیزیکی سطح کنترل و داده شبکه از یکدیگر است که در آن یک کنترولر می تواند چندین دستگاه را مدیریت‌کند. سطح داده شامل سوئیچ ساده ارسال بسته‌ها است که می‌توان آن را به صورت سخت‌افزاری (سوئیچ فیزیکی) و یا نرم‌افزاری (ســوئیچ مجازی) پیاده‌ســازی‌کرد. جزئیات پیاده‌ســازی ســوئیچ مانند اندازه بافرها، پارامترهای اولویت و ســایر ساختمان داده‌های مرتبط با ارسال بسته‌ها، می‌تواند متفاوت باشد. با این حال لازم است تا سوئیچ یک مدل انتزاعی قابل‌ارتباط با کنترلر را پیاده‌ســازی‌کند. این مدل انتزاعی در قالب یک رابط برنامه‌نویســی متن باز به‌علاوه یک پروتکل ارتباطی، بین ســطح کنترل و ســطح داده تعریف‌می‌شود. به این رابط، اصطلاحا رابط مرز جنوبی[6] نیز گفته‌می‌شود که برجســته‌ترین نمونه آن پروتکل اوپن‌فلو اســت. مشــخصــات اوپن‌فلو شــامل پروتکل ارتباط بین ســطح کنترل و داده به‌علاوه یک رابط برنامه‌نویسی فراخوانی اوپن‌فلو برای کنترلر می‌باشد.

کنترلر را می‌توان مستقیماً بر روی یک سرور فیزیکی یا مجازی اجراکرد. برای بهبود مقیاس پذیری و قابلیت اطمینان، رابط های مرز شرقی-غربی[۷] مانند Raft، AMQP و Zookeeper نیز معرفی‌شدند که پیاده سازی کنترلر به صورت توزیع شده و بر روی چند ما شین مختلف را ممکن می‌سازند[5, 6]. علاوه بر این، کنترلر یک رابط دیگر به نام رابط مرز شمالی[۸] برای ارتباط برنامه‌ها ارائه‌می‌کند که از طریق آن امکان تولید و پیاده‌سازی انواع سرویس‌هایی که قبل از SDN ممکن نبود، فراهم‌می‌شود. اگرچه استاندارد خاصی برای رابط مرز شمالی معرفی‌نشده، ا ما اغ لب تولیدکنندگان از رابط REST برای رابط برنامه‌نویسی کنترلر خود استفاده‌می‌کنند. رابط برنامه‌نویسی REST یک رابط کاربردی است که از درخواست‌ها و دستورهای HTTP به منظور ایجاد، تغییر و به‌روزرسانی، دریافت، حذف و پردازش داده‌ها و منابع بهره‌می‌برد. برنامه‌های کاربردی در SDN می‌توانند نیازمندی‌های خود را با استفاده از رابط مرزشمالی به کنترلر ارسال‌کنند. از جمله این برنامه‌ها می‌توان به مدیریت امنیت، کنترل دسترسی، بهینه‌سازی مصرف در شبکه و مدیریت شبکه اشاره‌کرد.



شکل۱: معماری SDN

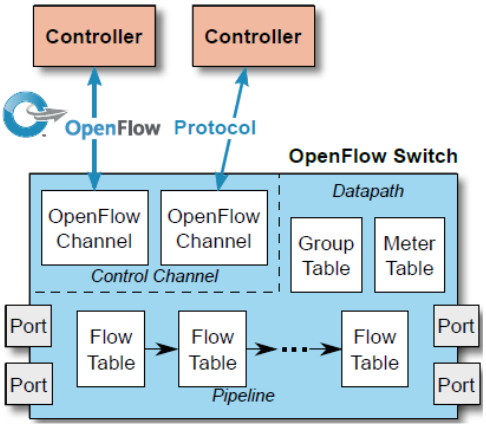
۱,۲. پروتکل OpenFlow

اوپن‌فلو، اولین استاندارد و معروفترین پیاده‌سازی SDN است که پروتکل ارتباط بین کنترلر و سطح داده تجهیزات شبکه را معرفی‌کرده‌است. از زمان ارائه اولین نسخه این پروتکل در سال ۲۰۰۹، ا ستفاده از آن در بسیاری از پروژه‌های تحقیقاتی و صنعتی، گسترش چ‌شمگیری دا شته ا ست. یکی از موفق‌ترین پیاده سازی‌های اوپن‌فلو در مقیاس صنعتی و بزرگ، پروژه B4 گوگل است که در سال ۲۰۱۳ برای ارتباط ناحیه‌گستر بین مراکز داده خود در سراسر جهان بهره‌برداری‌کرد[7].

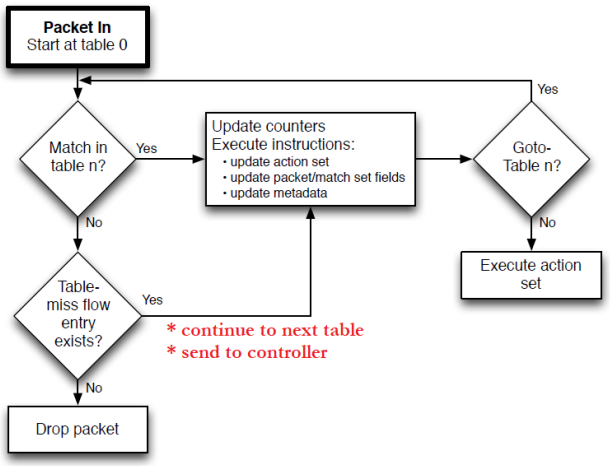
پروتکل اوپن‌فلو پیام‌های ارتباطی بین کنترلر و سوئیچ را استاندارد می‌کند. به طورکلی، این پیام‌ها شامل دستورهای نحوه برخورد با بسته‌ها برای سوئیچ و جمع‌آوری اطلاعات آماری جریان‌ها از سوئیچ می‌باشد. همچنین، اوپن‌فلو مشخصات اجزاء اصلی و نیازمندی‌های سوئیچ منطقی را تعریف‌کرده‌است.

۲,۲. مشخصات سوئیچ OpenFlow

سوئیچ اوپن‌فلو برای هدایت بسته‌های دریافتی، یک یا چند جدول جریان[۹] و یک جدول گروهی[۱۰] را در pipeline خود نگهداری‌می‌کند. همچنین، یک یا چند کانال OpenFlow برای ارتباط با کنترلر دارد. جداول به ترتیب و از ۰ شماره‌گذاری‌می شوند و هر جدول حاوی یک سری رکورد یا ورودی جریان[۱۱] با اولویت‌های مختلف ا ست. به محض ورود بسته، سوئیچ سرآمدهای بسته را ا ستخراج و جداول جریان خود را برای یافتن رکورد تطبیق جستجومی‌کند. جستجو از اولین جدول، آغاز و ممکن ا ست با جداول بعدی ادامه‌یابد. عملیات تطبیق[۱۲] بر اساس اولویت انجام‌می‌شود که ورودی با اولویت بالا انتخاب‌خواهدشد. در صورت وجود رکورد تطبیق، دستورهای[۱۳] مربوط به هدایت بسته اجراخواهد شد و در غیر این صورت، د ستورهای رکوردی با نام Table-Miss با اولویت ۰ اجرامی‌شود که دستورهای آن برحسب تنظیمات می‌تواند متفاوت باشد. ممکن است بسته در قالب پیام packet_in برای کنترلر ارسال شود، بسته حذف‌شود و یا عملیات جستجو در باقی جداول جریان ادامه‌یابد[8, 9].

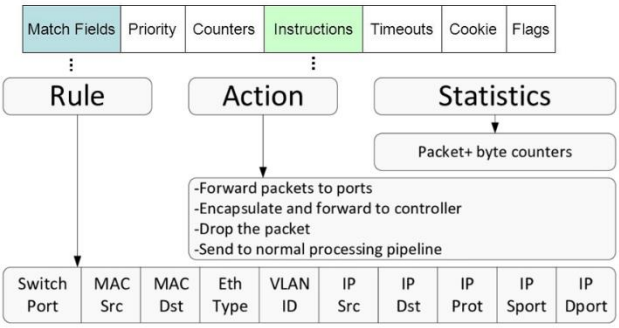


شکل ۲: اجزاء اصلی سوئیچ اوپن‌فلو



شکل ۳: فلوچارت عملیات مطابقت

هر رکورد یا ورودی جریان، شامل فیلدهای تطبیق، اولویت‌ها، شمارنده‌ها و مهلت زمانی و تعدادی دستور است. از فیلد تطبیق به همراه اولویت، برای شناسایی و انتخاب رکورد جریان منطبق با بسته استفاده‌می شود. نسخه ۱,۳,۰ اوپن‌فلو ۴۰ فیلد و نسخه ۱,۵,۰ آن ۴۴ فیلد تطبیق دارد. فیلدهای تطبیق دو نوع هستند. فیلدهای سرآمد که برحسب نوع بسته شامل فیلدهای سرآمد پروتکل‌های مختلف است، مانند آدرس مبدأ/ مقصد IPv4/v6 و VLAN ID. فیلدهای pipeline که شامل اطلاعات اضافی مورد نیاز پردازش مانند metadata و tunnel-id می‌باشد.



شکل ۴: اجزاء ورودی جریان

دستورها یا اقدامات مربوط به هر ورودی جریان، عملیاتی که بر روی بسته‌ها انجام‌خواهدشد را مشخص‌می‌کند. این اقدامات شامل نحوه هدایت بسته‌ها مانند ارسال به پورت خروجی سوئیچ، تغییر سرآمدهای بسته مانند کاهش مقدار فیلد TTL، تغییر وضعیت بسته مانند پیوند به صف و پردازش جداول گروهی مانند دستور Goto-Table می‌باشند.

۳,۲. سوئیچ‌های نرم‌افزاری[۱۴]

سوئیچ‌های لایه داده که وظیفه ارسال بسته‌ها را بر عهده دارند، می‌توانند سخت‌افزاری یا نرم‌افزاری باشند. سوئیچ‌های نرم‌افزاری، جداول جریان را در حافظه SDRAM نگهداری‌می‌کنند و عملیات تطبیق بر روی بسته‌های ورودی با استفاده از پردازشگر مرکزی انجام‌می‌شود. بنابراین، منطق پردازش بسته‌ها توسط نرم‌افزار و کتابخانه‌های نرم‌افزاری بهینه، پیاده سازی می شود. سوئیچ‌های Open vSwitch (OVS) و ofsoftswitch13 ازجمله پرکاربردترین سوئیچ‌های نرم هستند.

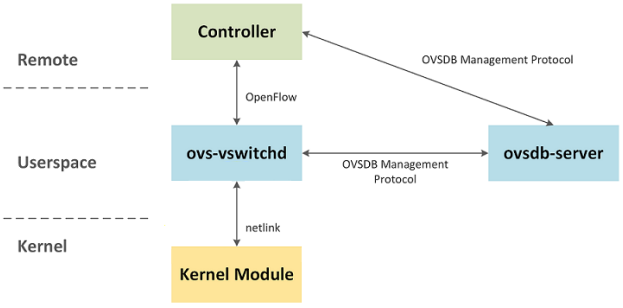
درمقابل، عملیات ارسال بسته در سوئیچ سخت‌افزاری، درون سخت‌افزار مخصوصی شامل حافظه TCAM و مدار مجتمع ASIC تعبیه‌می‌شود. همچنین برخی از این سوئیچ‌ها مجهز به SDRAM و CPU نیز هستند تا بتوانند جداول جریان را درون هر دو حافظه TCAM و SDRAM نگهداری‌کنند[10]. سرعت عملیات تطبیق با استفاده از حافظه TCAM می‌تواند برابر سرعت خط باشد. اما، اغلب سوئیچ‌های سخت به‌دلیل هزینه و مصرف انرژی بالای حافظه TCAM، ظرفیت نگهداری از تعداد محدودی ورودی جریان را دارند. درحالی‌که SDRAM هزینه و مصرف انرژی پائین‌تر و ظرفیت بالاتری دارد و امکان اجرای عملیات پیچیده‌تری را توسط نرم‌افزار ممکن می‌سازد[11].

هر دو نوع سوئیچ نرم‌و سخت، مزیت‌ها و معایب خاص خود را دارند و بسته به نوع شبکه، انتخاب هر یک ازآن‌ها تأثیر مستقیمی بر عملکرد شبکه خواهدداشت. سوئیچ‌های نرم انعطاف‌پذیری و قابلیت‌های برنامه‌نویسی بالایی دارند. اما، سرعت پردازش بسته‌ها، به‌ویژه کلاس‌بندی آن‌ها با استفاده از یک پردازشگر همه‌منظوره، نسبت به مدار ASIC و پردازشگر شبکه، کمتر است. به‌علاوه در سوئیچ‌های سخت به‌خاطر ویژگی‌های سخت‌افزاری، تراکم پورت‌ها، بسیار بالاتر از یک کارت شبکه معمولی است. بنابراین، برای شبکه‌های تجاری با ظرفیت بالا، استفاده از ترکیب هردو سوئیچ سخت و نرم توصیه‌می‌شود. اما در شبکه‌های کوچک یا پروژه‌های شبیه‌سازی و به‌ویژه در شبکه‌های overlay مجازی و پردازش ابری مانند OpenStack صرفاً سوئیچ نرم قابل‌استفاده خواهدبود.

۱,۳,۲. سوئیچ Open vSwitch (OVS)

OVS یکی از معروفترین سوئیچ‌های نرم است که توسط بنیاد لینوکس پشتیبانی می شود و به نماد سوئیچ SDN تبدیل شده‌است. این سوئیچ که از ناظر ماشین مجازی[۱۵] لینوکس بهره‌می‌برد، از تمام پروتکل‌ها و تکنولوژی‌های مهم، مانند VLAN tagging، NetFlow، Port Mirroring پشتیبانی‌می‌کند. OVS نرم‌افزاری چندلایه دارد و هسته آن پردازش پس‌زمینه ovs-vswitchd است که وظیفه مدیریت ارتباطات و نگهداری آمار جریان‌ها را برعهده‌دارد[11, 12].

OVS برای ارتباط با کنترلر از پروتکل اوپن‌فلو و برای مدیریت تنظیمات از یک سیستم پایگاه داده تحت شبکه به نام OVSDB استفاده‌می‌کند. اعمال تنظیمات در ovsdb هم ازطریق کنترلر و هم با استفاده از دستور ovs-vsctl امکان‌پذیر است. همچنین، از دستور ovs-ofctl نیز می‌توان برای مدیریت و نظارت بر سوئیچ‌ها استفاده‌کرد.



شکل ۵: معماری سوئیچ OVS

۴,۲. کنترلرهای SDN

کنترلر، هسته مرکزی و مغز شبکه SDN است که عملکردی مشابه سیستم عامل دارد و با ارائه رابط‌های باند شمالی و جنوبی، ارتباط

برنامه‌های سطح بالا با سوئیچ‌های لایه داده را ممکن‌ساخته و بدین ترتیب مدیریت و کنترل شبکه را متمرکز می‌کند.

کنترلرهای SDN اغلب به صورت یک پلتفرم شامل مجموعه ای از ماژول‌های مختلف که هرکدام وظیفه مشخصی دارند، تولید می شوند. به عناوین مثال، ماژول تشخیص لینک یا توپولوژی با استعلام وضعیت پورت‌هاو پیام‌های packet_in دریافتی، توپولوژی شبکه را شناسایی می‌کند و امکان یافتن مسیرهای بهینه را برای ماژول تصمیم‌گیرنده فراهم می‌آورد. ماژول مدیریت جریان، یکی از ماژول‌های اصلی هر کنترلر است که مستقیماً با جداول و ورودی‌های جریان سطح داده ارتباط دارد.

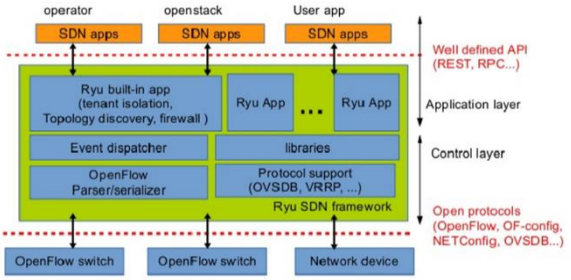
ازجمله پلتفرم های کنترلر شناخته‌شده و پرکاربرد می‌توان، OpenDaylight [14]، Floodlight [15]، POX [16]، ONOS [17]، RYU [18, 19] و VMWare NSX [20] را برشمرد که هرکدام مزیت‌ها و محدودیت‌های خود را دارند. از نظر پیاده‌سازی عملی در یک نوع خاص شبکه، تشخیص اینکه کدام یک عملکرد بهتری خواهندداشت، مستلزم آزمایش‌های گسترده ای است. ابزارها و متریک‌های مختلفی برای محک‌زدن کنترلرها مانند CBench، HCprobe، OFCBenchmark، OFNet و PktBlaster تولید شده‌اند. نتایج مطالعات صورت گرفته در این زمینه، ترکیبی از پارامترهای مختلف را در برتری عملکرد کنترلر معرفی می‌کنند. به عنوان نمونه، کنترلرهای مبتنی بر زبان C در موقعیت افزایش تعداد سوئیچ‌ها، گذردهی بالاتری داشتند. درحالی‌که کنترلرهای مبتنی بر پایتون امتیاز بهتری در پارامتر تأخیر کسب‌کردند. یکی دیگر از ویژگی‌های کنترلر، قابلیت آن در پردازش چند نخی است که ارتباط مستقیمی با توان گذردهی آن دارد[5].

با درنظر گرفتن محدودیت توان سخت‌افزارهای مورد استفاده در این پروژه و بسیاری از مطالعات دانشگاهی مشابه و مزیت‌های متعدد کنترلر RYU و زبان پایتون، در این مقاله از کنترلر RYU استفاده می‌شود.

۱،۴،۲. مشخصات کنترلر RYU

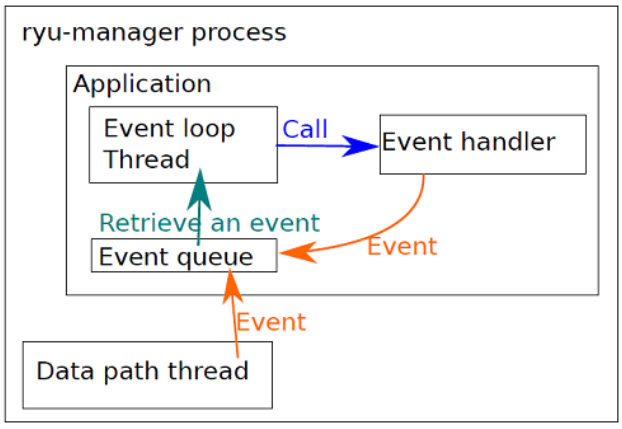
Ryu چهارچوب متن باز و مؤلفه‌محور SDN است که توسط شرکت ژاپنی نیپون NTT تولیدشد. این کنترلر به زبان پایتون برنامه‌نویسی شده و مؤلفه‌های آن، رابط‌های برنامه‌نویسی خوش‌تعریف و قدرتمندی را ارائه‌می‌دهند. Ryu امکان تولید برنامه‌های مدیریتی و کنترلی را برای تولیدکنندگان تسهیل کرده‌است. ریو پروتکل‌های مختلف مدیریت شبکه مانند NetConf، OF-config و همچنین نسخه‌های مختلف اوپن‌فلو را پشتیبانی می‌کند.

معماری کنترلر Ryu دارای سه لایه است. لایه کاربردی شامل برنامه‌های منطقی و تجاری شبکه، لایه میانی شامل سرویس‌های لایه کنترل SDN و لایه زیر ساخت شبکه شامل تجهیزات سخت‌افزاری یا مجازی است. لایه میانی میزبان رابط‌های باند شمالی و جنوبی است. رابط‌های باند شمالی شامل رابط مدیریتی REST، رابط Quantum، و رابط‌های قابل‌تعریف با استفاده از REST یا RPC است. رابط‌های باند جنوبی قادر به پشتیبانی از پروتکل‌های مختلف مدیریت تجهیزات شبکه است. همچنین Ryu گروهی از مؤلفه‌ها مانند OpenStack Quantum و Firewall را فراهم‌کرده‌است. Ryu با سوئیچ‌های مختلف OpenFlow از جمله OpenvSwitch، Centec، HP، IBM و NEC تست و تأییدشده‌است.



شکل ۶: معماری چارچوب Ryu

درمدل برنامه‌نویسی Ryu، برنامه‌ها، موجودیت‌های تک‌رشته‌ای[۱۶] هستند که عملکردهای مختلف را پیاده‌سازی می‌کنند و با ارسال پیام‌های نامتقارن[۱۷] event با یکدیگر ارتباط دارند. هر برنامه، یک ماژول پایتون است که از کلاس ryu.base.app_manager.RyuApp ارث بری شده و الگوی طراحی آن singleton است. به این معنی که تنها یک نمونه از هر برنامه قابل‌اجراست. پردازش eventها نیز، با فراخوانی متد event handler مناسب انجام‌می‌شود.



شکل ۷: مدل برنامه‌نویسی Ryu

۵،۲. مفاهیم کیفیت سرویس

کیفیت سرویس، یکی از نیازهای بنیادی زیرساخت شبکه است که برای سال‌ها پیشرفت چشم‌گیری نداشت. مشکل اولیه در ارسال ارتباطات بلادرنگ به‌وسیله سوئیچینگ بسته، تأخیر و حذف بسته‌ها بود. برای حل این مشکل و اصلاح خطا در ارتباطات، پروتکل‌های هوشمند لایه هفت OSI مانند مهلت زمانی و بازارسالی معرفی شدند. به منظور ایجاد محیط منطقی مدار جهت ارسال بدون تأخیر ارتباط

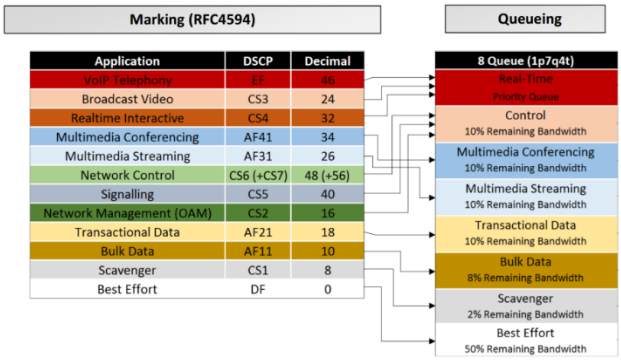
بلادرنگ در یک شبکه سوئیچینگ بسته مانند شبکه IP، مفهوم کلاس‌بندی ترافیک معرفی‌شد. کلاس ترافیک براساس پارامترهای بنیادی مانند تأخیر[۱۸]، لرزش[۱۹] و اتلاف[۲۰] تعریف‌می‌شود. هر کلاس ترافیک ممکن است به هرکدام از این پارامترها حساس باشد. اگر بتوان منابع پردازش، بافر، پهنای باند و حافظه شبکه را برای کاهش پارامترهایی که هر کلاس ترافیک نسبت به آن‌ها حساس است، به‌کارگرفت، می‌توان انواع مختلف ترافیک را بروی یک شبکه ارسال‌کرده و تجربه نهایی کاربران را نیز افزایش‌داد [21].

کیفیت سرویس در توانایی فراهم‌آوردن نیازهای یک بخش از ترافیک تعریف‌می‌شود و هدف اصلی آن، مدیریت مبارزه برای منابع شبکه به منظور افزایش کیفیت تجربه و رضایت کاربر است. مشخصات کیفیت سرویس، یک سیستم ناعادلانه مدیریت‌شده به‌وجودمی‌آورد که در آن، تعدادی از جلسه‌ها اهمیت بیشتری از سایرین دارند. جلسه‌های حساس به تأخیر، از صف بسته‌های حاوی جلسه‌های با حساسیت کمتر، گذرانده‌می‌شوند ودر زمان سرریز صف بافرها نیز از بسته‌های جلسه کم‌اهمیت‌تر حذف‌خواهد شد. به منظور ایجاد پهنای باند مناسب برای جلسات مهم، جلسات کم‌اهمیت‌تر مدیریت می‌شوند. به این معنی که بسته‌های آن‌ها براساس نیازهای کیفیت سرویس شبکه، به‌طور انتخابی با تأخیر ارسال و یا حذف‌می‌شوند.

۱،۵،۲. مکانیزم‌های کیفیت سرویس

مکانیزم های کیفیت سرویس به دو مدل اصلی جریان‌محور و کلاس‌بندی تقسیم‌می‌شوند. در مدل جریان‌محور مانند خدمات مجتمع[۲۱]، میزان مشخصی از منابع در طول مسیر یک جریان رزرو می‌شود. این روش که به Hard QoS نیز معروف است، علاوه بر عدم مقیاس‌پذیری، منابع شبکه را نیز تلف‌می‌کند. درنتیجه صرفاً برای شبکه‌های کوچک قابل‌استفاده است.

درمقابل، مدل کلاس‌بندی ترافیک که به مدل خدمات متمایز معروف است، شبکه را برای سرویس‌دهی به چند کلاس ترافیک با اولویت های مختلف پیکربندی‌می‌کند. کلاس‌بندی با درنظرگرفتن مشخصات انواع ترافیک و نیازهای برنامه ها و خدمات مختلف انجام‌می‌شود. به منظور علامت گذاری کلاس ترافیک بسته ها دراستاندارد خدمات متمایز، فیلد سرآمد جدیدی به نام DS جایگزین فیلد ToS[۲۲] می‌شود که ۶ بیت اولیه آن کد کلاس DSCP[۲۳] را مشخص‌می‌کند. نحوه ارسال بسته‌ها بر اساس اولویت‌های رفتار گام به گام[۲۴] یا PHB انجام‌می‌شود. PHBها به چهار گروه Default، CS[۲۵]، AF[۲۶] و EF[۲۷] تقسیم‌می شوند که هریک تعدادی کلاس ترافیک در زیر مجموعه خود تعریف‌می‌کنند. به عنوان مثال، گروه AF چهار کلاس ترافیک با مقادیر مختلف، حداقل پهنای باند تضمینی و اجازه دسترسی به پهنای باند اضافه تعریف‌کرده‌است. هریک از کلاس‌های نوع AF شامل سه زیرکلاس با تقدم حذف[۲۸] متفاوت هستند.



شکل ۸: کلاس‌های خدمات متمایز

۶،۲. پشتیبانی از QoS در نسخه‌های مختلف OpenFlow

پشتیبانی از کیفیت سرویس در نسخه‌های ابتدایی اوپن‌فلو بسیار محدود است. به مرور زمان و با انتشار هر نسخه جدید، ویژگی جدیدی به آن اضافه می‌شود . در نسخه ۱،۰ فقط امکان صف با نرخ حداقل وجود دارد و صف با نرخ حداقل/حداکثر در نسخه ۱،۲ اضافه شده است. صف‌های اوپن‌فلو به‌طور وسیع و توسط تمامی سوئیچ‌های نرم و سخت موجود پشتیبانی‌می‌شوند.

برای دستیابی به کیفیت سرویس دقیق‌تر، در نسخه ۱،۳ اوپن‌فلو مفهوم میزمتر یا Meter Table معرفی‌می شود. درحالی‌که ویژگی صف، نرخ خروجی[۲۹] ترافیک را کنترل‌می‌کند، از میزمتر می‌توان برای کنترل نرخ ترافیک قبل از ارسال، یا همان نرخ ورود[۳۰] استفاده‌کرد. بدین ترتیب، صف و میزمتر مکمل یکدیگر هستند. همچنین سوئیچ‌های اوپن‌فلو قابلیت خواندن و نوشتن بیت‌های کد کلاس سرویس را در سرآمد بسته دارند.

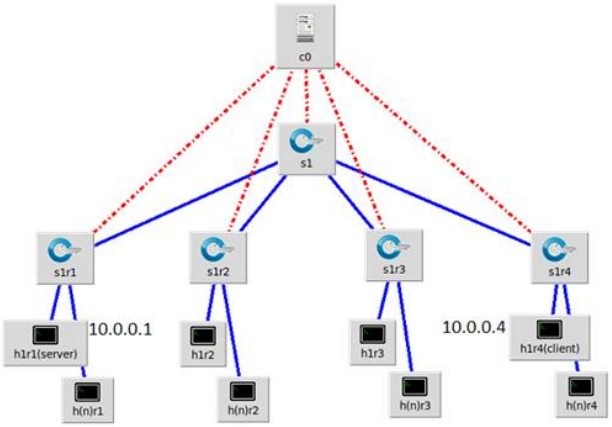
۳. چهارچوب راه حل پیشنهادی

در مقاله موجود، دو هدف در نظر گرفته شده است. بهره برداری از کنترلر RYU در شبکه SDN به منظور پیاده‌سازی مکانیزم کیفیت سرویس و تضمین کیفیت تجربه برای کاربران شبکه، به عنوان هدف اول است. سپس، زیر ساخت و چهارچوب منظم و قابل توسعه برای مطالعات بیشتر ارائه می شود. توپولوژی مورد نظر ما، شبکه یک مرکز داده با تعداد ۴ رک، یک سوئیچ ToR[۳۱] به ازای هر رک، یک سوئیچ مرکزی و تعداد ۴ host در هر رک می باشد. همچنین کد های پیاده‌سازی در گیت‌هاب پروژه [22] قابل‌دسترس هستند.

۱،۳. عملیات کیفیت سرویس Per-Flow

روش per-flow از مکانیزم‌های رزرو منابع برای جریان‌های مختلف در مدیریت صف ترافیک استفاده‌می‌کند. پیاده‌سازی این قسمت با تنظیمات مربوط به جریان در سوئیچ‌های مابین سرور و کلاینت با استفاده از رابط REST کنترلر Ryu انجام‌می‌شود. با استفاده از ابزارهای تولید و محاسبه ترافیک، میزان پهنای باند تضمین‌شده و تقدم و تأخیر محاسبه‌می شود. نتایج به‌دست‌آمده نشان‌می‌دهد که ترافیک ارسالی به

پورت ۵۰۰۱ تا سقف Kbps۵۰۰ محدود شده و برای ترافیک ارسالی به پورت ۵۰۰۲، پهنای باند Kbps۸۰۰ تضمین شده‌است.



شکل ۹: توپولوژی تست شماره ۱

پارامترهای درنظرگرفته‌شده برای صف و جدول جریان مطابق جداول ۱و ۲ می‌باشند.

جدول ۱: مشخصات صف~~ها~~ها

Queue ID	Max rate	Min rate
0	500 Kbps	-
1	1 Mbps	800 Kbps

جدول ۲: مشخصات جدول جریان

Priority	Dest Addr	Dest Port	Protocol	Queue ID	QoS ID
1	10.0.0.1	5002	UDP	1	1

۱،۱،۳. افزودن جدول جریان

به منظور ایجاد یک جدول جریان جهت درج ورودی‌های مربوط به کیفیت سرویس، تغییراتی در کدهای برنامه simple_switch_13.py ایجادشد و برنامه جدیدی با نام qos_simple_switch_13.py ساخته‌شد.

```
51      def add_flow(self, datapath, priority, match, actions, buffer_id=None):
52          ofproto = datapath.ofproto
53          parser = datapath.ofproto_parser
54
55          inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
56                                               actions)]
57          if buffer_id:
58              mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
59                                      priority=priority, match=match,
60                                      instructions=inst, table_id=1)
61          else:
62              mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
63                                      match=match, instructions=inst, table_id=1)
64          datapath.send_msg(mod)
```

شکل ۱۰: برنامه qos_simple_switch_13.py

۲،۱،۳. ایجاد محیط شبکه در Mininet

توپولوژی شبکه نمونه در mininet اجرا می شود و با ا ستفاده از پنجره xTerm به کنترلر، سوئیچ‌ها، کلاینت و سرور اتصال برقرارمی‌شود.

```
(thesisENV) root@voyager3:~/thesisENV/sdn_qos# mn --custom topology/datacenterBasic.py \
> --topo dcbasic --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
*** Adding switches:
s1 s1r1 s1r2 s1r3 s1r4
*** Adding links:
(s1, s1r1) (s1, s1r2) (s1, s1r3) (s1, s1r4) (s1r1, h1r1) (s1r1, h2r1) (s1r1, h3r1) (s1r1, h4r1)
) (s1r4, h2r4) (s1r4, h3r4) (s1r4, h4r4)
*** Configuring hosts
h1r1 h1r2 h1r3 h1r4 h2r1 h2r2 h2r3 h2r4 h3r1 h3r2 h3r3 h3r4 h4r1 h4r2 h4r3 h4r4
*** Starting controller
c0
*** Starting 5 switches
s1 s1r1 s1r2 s1r3 s1r4 ...
*** Starting CLI:
mininet>
```

شکل۱۱: اجرای توپولوژی datacenterBasic.py

۳،۱،۳. تنظیم پورت OVSDB و نسخه OpenFlow1.3

نسخه اوپن‌فلو مرجع در سوئیچ و کنترلر باید یکسان باشد. همچنین لازم ا ست تا پورت OVSDB نیز در تنظیمات سوئیچ‌ها اعمال شود. به این منظور، اسکریپت~~ها~~های ovsbr_set_s1*.sh در سوئیچ‌های بین کلاینت و سرور اجرامی‌شوند.

```
echo "setting openflow protocol version on Bridge s1..." $'\n
ovs-vsctl set Bridge s1 protocols=OpenFlow13
ovs-vsctl set-manager ptcp:6632
```

شکل ۱۲: اجرای اسکریپت‌های ovsbr_set_s1*.py

۴،۱،۳. اجرای برنامه‌های کنترلر

به منظور راه‌اندازی کنترلر RYU و رابط‌های آن، برنامه‌های rest_qos، qos_simple_switch_13 و rest_conf_switch در پنجره کنترلر اجرامی‌شوند. کنترلر به محض اجرا، تمام سوئیچ‌ها را شناسایی و به آن‌ها متصل‌می‌شود.

برنامه qos_simple_switch_13 عملکردهای سوئیچ یادگیری لایه ۲، مانند یادگیری آدرس MAC نودها را اجرامی‌کند. با تغییراتی که در مرحله ۱-۱-۳ در این برنامه انجام‌دادیم، اکنون ورودی‌های مربوط به QoS در جدول جریان table_id:1 ثبت‌می‌شوند.

شکل ۱۳: اجرای برنامه‌های کنترلر RYU

برنامه rest_qos رابط لازم برای انجام تنظیمات صف و جدول جریان را مطابق شکل ۱۴ فراهم‌می‌کند.

```
# ============================
#          REST API
# ============================
#
#  Note: specify switch and vlan group, as follows.
#    {switch-id} : 'all' or switchID
#    {vlan-id}   : 'all' or vlanID
#
# about queue status
#
# get status of queue
# GET /qos/queue/status/{switch-id}
#
# about queues
# get a queue configurations
# GET /qos/queue/{switch-id}
#
# set a queue to the switches
# POST /qos/queue/{switch-id}
```

شکل ۱۴: رابط تعریف شده در برنامه rest_qos

۵،۱،۳. انجام تنظیمات صف و جدول جریان

دستورهای لازم برای انجام تنظیمات مربوط به صف و ورودی‌های جریان در اسکریپت perflow_qos_script.sh نوشته‌شد که در این مرحله اجرامی‌شود.

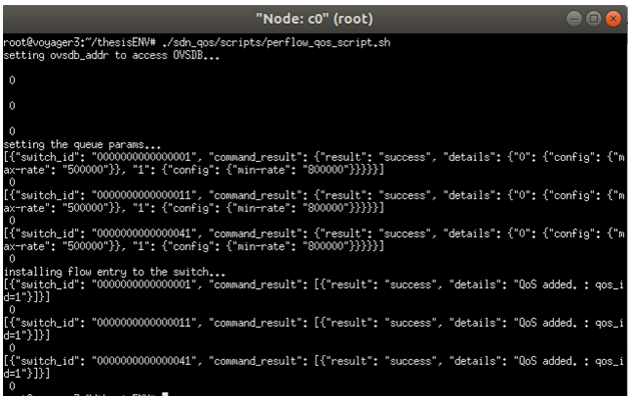
```
#!/bin/bash

s1_URL="http://localhost:8080/v1.0/conf/switches/0000000000000001/ovsdb_addr"
s1r1_URL="http://localhost:8080/v1.0/conf/switches/0000000000000011/ovsdb_addr"
s1r4_URL="http://localhost:8080/v1.0/conf/switches/0000000000000041/ovsdb_addr"
echo "setting ovsdb_addr to access OVSDB..."
curl -X PUT -d ' "tcp:127.0.0.1:6632" ' "$s1_URL"
curl -X PUT -d ' "tcp:127.0.0.1:6632" ' "$s1r1_URL"
curl -X PUT -d ' "tcp:127.0.0.1:6632" ' "$s1r4_URL"

s1_queue_URL="http://localhost:8080/qos/queue/0000000000000001"
s1r1_queue_URL="http://localhost:8080/qos/queue/0000000000000011"
s1r4_queue_URL="http://localhost:8080/qos/queue/0000000000000041"
echo "setting the queue params..."
curl -X POST -d '{"port_name": "s1-eth1", "type": "linux-htb", "max_rate": "1000000", "queues": [{"max_rate": "500000"}, {"min_rate": "800000"}]}'
curl -X POST -d '{"port_name": "s1r1-eth1", "type": "linux-htb", "max_rate": "1000000", "queues": [{"max_rate": "500000"}, {"min_rate": "800000"}]}'
curl -X POST -d '{"port_name": "s1r4-eth1", "type": "linux-htb", "max_rate": "1000000", "queues": [{"max_rate": "500000"}, {"min_rate": "800000"}]}'

s1_flow_URL="http://localhost:8080/qos/rules/0000000000000001"
s1r1_flow_URL="http://localhost:8080/qos/rules/0000000000000011"
s1r4_flow_URL="http://localhost:8080/qos/rules/0000000000000041"
echo "installing flow entry to the switch..."
curl -X POST -d '{"match": {"nw_dst": "10.0.0.1", "nw_proto": "UDP", "tp_dst": "5002"}, "actions":{"queue": "1"}}' "$s1_flow_URL"
curl -X POST -d '{"match": {"nw_dst": "10.0.0.1", "nw_proto": "UDP", "tp_dst": "5002"}, "actions":{"queue": "1"}}' "$s1r1_flow_URL"
curl -X POST -d '{"match": {"nw_dst": "10.0.0.1", "nw_proto": "UDP", "tp_dst": "5002"}, "actions":{"queue": "1"}}' "$s1r4_flow_URL"
```

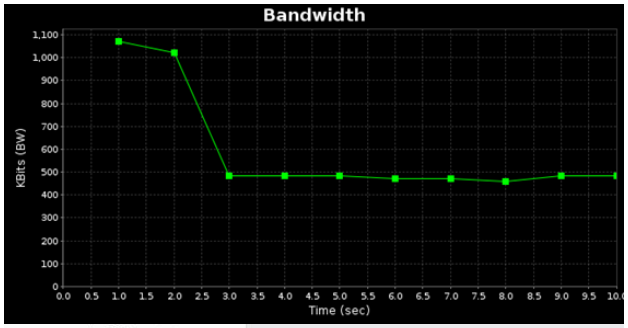
شکل ۱۵: اسکریپت perflow_qos_script.sh



شکل ۱۶: خروجی اسکریپت perflow_qos_script.sh

۶،۱،۳. نتایج

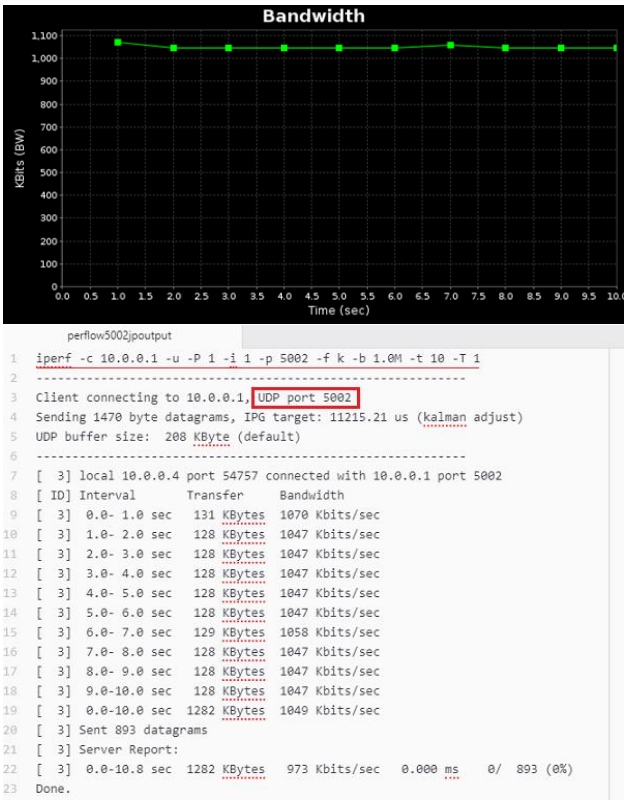
نود سرور h1r1، پورت 5002-5001 را با پروتکل UDP شنودمی‌کند. نود کلاینت h1r4، 1Mbps ترافیک UDP به پورت 5001 و 1Mbps ترافیک UDP به پورت 5002 ارسال‌می‌کند. نتایج بدست آمده با استفاده از ابزار Jperf نشان‌می‌دهد که ترافیک ارسالی به پورت 5001 تا سقف 500 Kbps محدودمی‌شود تا ترافیک ارسالی به پورت 5002 تا سقف 1Mbps تضمین‌شود. نتایج پورت‌های 5001 و 5002 در شکل‌های ۱۷و ۱۸ مشاهده‌می‌شوند.



perflow5001joutput

```
iperf -c 10.0.0.1 -u -P 1 -i 1 -p 5001 -f k -b 1.0M -t 10 -T 1
------------------------------------------------------------
Client connecting to 10.0.0.1, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.4 port 38797 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec   131 KBytes  1070 Kbits/sec
[  3]  1.0- 2.0 sec   125 KBytes  1023 Kbits/sec
[  3]  2.0- 3.0 sec  58.9 KBytes   482 Kbits/sec
[  3]  3.0- 4.0 sec  58.9 KBytes   482 Kbits/sec
[  3]  4.0- 5.0 sec  58.9 KBytes   482 Kbits/sec
[  3]  5.0- 6.0 sec  57.4 KBytes   470 Kbits/sec
[  3]  6.0- 7.0 sec  57.4 KBytes   470 Kbits/sec
[  3]  7.0- 8.0 sec  56.0 KBytes   459 Kbits/sec
[  3]  8.0- 9.0 sec  58.9 KBytes   482 Kbits/sec
[  3]  9.0-10.0 sec  58.9 KBytes   482 Kbits/sec
[  3]  0.0-10.4 sec   721 KBytes   569 Kbits/sec
[  3] Sent 502 datagrams
[  3] Server Report:
[  3]  0.0-12.1 sec   721 KBytes   487 Kbits/sec   0.000 ms    0/  502 (0%)
Done.
```

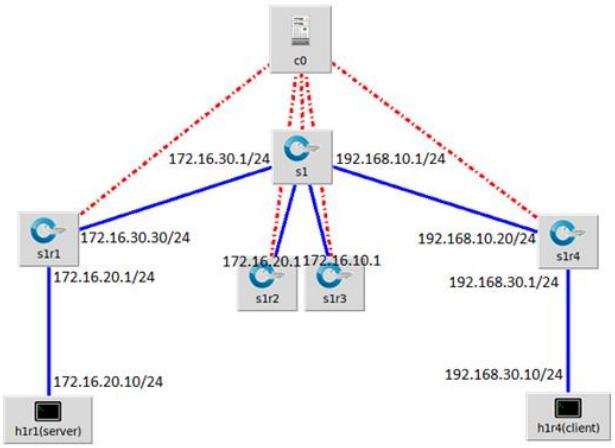
شکل ۱۷: خروجی ترافیک ارسالی به پورت 5001



شکل ۱۸: خروجی ترافیک ارسالی به پورت 5002

۲،۳. عملیات کیفیت سرویس خدمات متمایز

برای اجرای مکانیزم خدمات متمایز، لازم است تا حداقل یک دامنه خدمات متمایز DS-domain تعریف‌شود به نحوی که جریان‌های ورودی به دامنه با کد DSCP کلاس‌بندی و بر اساس تنظیمات PHB در سراسر مسیر مورد نظر ارسال شوند. در تست مورد نظر، دامنه DS در طول مسیر بین کلاینت و سرور تعریف‌می شود. تنظیمات صف‌ها مطابق جدول ۳ و قوانین علامت گذاری DSCP مطابق جدول ۵ اعمال‌می‌شود.



شکل ۱۹: توپولوژی تست شماره ۲

با استفاده از ابزارهای تولید و محاسبه ترافیک، میزان پهنای باند تضمین‌شده و تقدم و تأخیر محاسبه‌می‌شود. نتایج به‌دست‌آمده نشان‌می‌دهد که با محدودسازی پهنای باند ترافیک best-effort می‌توان برای ترافیک علامت‌گذاری‌شده با AF41 ارسالی به پورت ۵۰۰۳، پهنای باند 500 Kbps و برای ترافیک علامت‌گذاری‌شده با AF31 ارسالی به پورت ۵۰۰۲، پهنای باند 200 Kbps را تضمین کرد.

پارامترهای درنظرگرفته‌شده برای صف‌ها، ورودی‌های جریان و قوانین علامت‌گذاری مطابق جداول ۳، ۴و ۵ می‌باشند.

جدول ۳: مشخصات صف‌ها

Queue ID	Max Rate	Min Rate	Class
0	1Mbps	-	Default
1	1Mbps	200Kbps	AF31
2	1Mbps	500Kbps	AF41

جدول ۴: ورودی‌های جریان برای روترهای s1 و s1r1

Priority	DSCP	Queue ID	QoS ID
1	26(AF31)	1	1
1	34(AF41)	2	2

جدول ۵: قوانین علامت گذاری DSCP برای روتر s1r4

Priority	Dest Addr	Dest Port	Protocol	DSCP	QoS ID
1	172.16.20.10	5002	UDP	26(AF31)	1
1	172.16.20.10	5003	UDP	34(AF41)	2

۱،۲،۳. افزودن جدول جریان

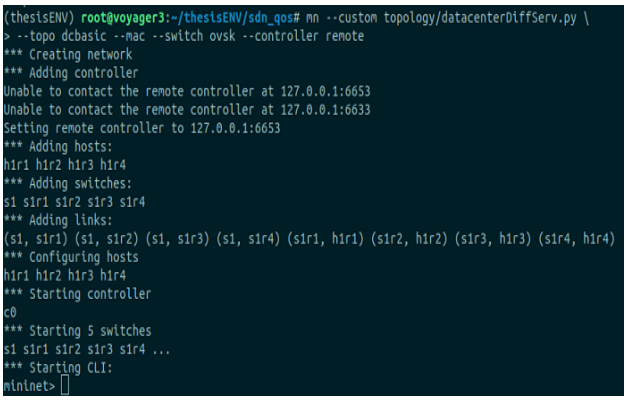
کنترلر RYU برای انجام تنظیمات و اجرای عملیات مسیریابی در شبکه multi-tenant از برنامه rest_router.py استفاده‌می‌کند. به منظور ایجاد جدول جریان مربوط به کیفیت سرویس، تغییراتی در کدهای برنامه ایجادشده و برنامه جدیدی با نام qos_rest_router.py ساخته‌می‌شود.

```
1764        # Instructions
1765        actions = actions or []
1766        inst = [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS,
1767                                                 actions)]
1768
1769        m = ofp_parser.OFPFlowMod(self.dp, cookie, 0, 1, cmd, idle_timeout,
1770                                  0, priority, UINT32_MAX, ofp.OFPP_ANY,
1771                                  ofp.OFPG_ANY, 0, match, inst)
1806
1807        flow_mod = ofp_parser.OFPFlowMod(self.dp, cookie, cookie_mask, 1, cmd,
1808                                         0, 0, 0, UINT32_MAX, ofp.OFPP_ANY,
1809                                         ofp.OFPG_ANY, 0, match, inst)
1810        self.dp.send_msg(flow_mod)
1811        self.logger.info('Delete flow [cookie=0x%x]', cookie, extra=self.sw_id)
```

شکل ۲۰: برنامه qos_rest_router.py

۲،۲،۳. ایجاد محیط شبکه در Mininet

توپولوژی شبکه نمونه در mininet اجرامی‌شود و با استفاده از پنجره xTerm اتصال به کنترلر، سوئیچ‌ها، کلاینت و سرور برقرارمی‌شود.



شکل ۲۱: اجرای توپولوژی datacenterDiffServ.py

۳،۲،۳. تنظیم آدرس IP جدید کلاینت و سرور

ازآنجا که mininet برای هر نود، آدرس پیش فرض اختصـــاص‌می‌دهد، برای تنظیم آدرس‌هایی که در توپولوژی شـکل ۱۸ تعریف‌شـدند، لازم ا ست تا آدرس‌ها بازنشانی شوند. د ستورهای مربوط به بازنشانی آدرس کلاینت و سـرور در اسـکریپت‌های h1r1_set_ip.sh و h1r4_set_ip.sh قرارداده‌شد که هریک در نود مربوطه اجرامی‌شود.

شکل ۲۲: اجرای اسکریپت‌های h1r1/4_set_ip.sh

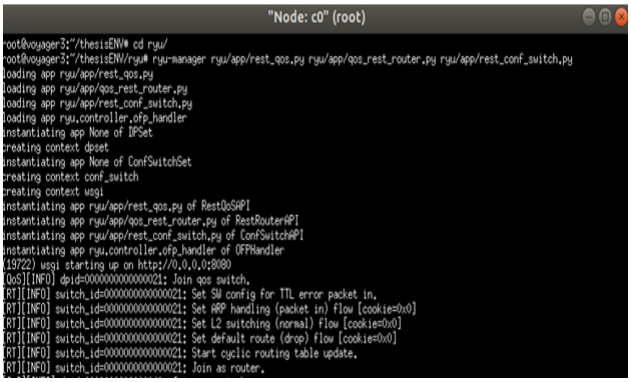
۴،۲،۳. تنظیم پورت OVSDB و نسخه OpenFlow1.3

مرحله ۳-۱-۳ تکرارمی‌شود.

۵،۲،۳. اجرای برنامه‌های کنترلر

بـه منظور راه‌انـدازی کنترلر RYU و رابط‌هـای آن، برنـامـه‌هـای rest_qos.py، qos_rest_router.py و rest_conf_switch.py اجرامی‌شوند.

برنامه qos_rest_router عملکردهای معمول روتر مانند شـــناســـایی و ارتباط با ســـوئیچ‌ها، تعریف و مدیریت VLANها و مســـیریابی را انجام‌می‌هد. با تغییراتی که در مرحله ۱-۲-۳ در این برنامه انجام‌شـد، اکنون ورودی‌هـای مربوط بـه QoS در جـدول جریـان table_id:1 ثبت‌می‌شوند.



شکل ۲۳: اجرای برنامه‌های کنترلر RYU

برنامه rest_conf_switch رابط لازم برای انجام تنظیمات ســـوئیچ‌ها و مسیرها را مطابق شکل ۲۴ فراهم‌می‌کند.

```
# REST API for switch configuration
#
# get all the switches
# GET /v1.0/conf/switches
#
# get all the configuration keys of a switch
# GET /v1.0/conf/switches/<dpid>
#
# delete all the configuration of a switch
# DELETE /v1.0/conf/switches/<dpid>
#
# set the <key> configuration of a switch
# PUT /v1.0/conf/switches/<dpid>/<key>
#
# get the <key> configuration of a switch
# GET /v1.0/conf/switches/<dpid>/<key>
#
# delete the <key> configuration of a switch
# DELETE /v1.0/conf/switches/<dpid>/<key>
#
# where
# <dpid>: datapath id in 16 hex
```

شکل ۲۴: رابط تعریف شده در برنامه rest_conf_switch

۶،۲،۳. تنظیم آدرس‌های جدید روترها و تنظیمات مسیریابی

برای تنظیم آدرس‌های جدید روترها مطابق توپولوژی شکل ۱۸و ایجاد مســـیرهای لازم برای ارتباط کلاینت و ســـرور، دســـتورهای لازم در ا سکریپت‌های router_set_ip.sh و route_setting.sh نو شته شد که در این مرحله اجرامی‌شوند.

شکل ۲۵: اجرای اسکریپت router_set_ip.sh

```
root@voyager3:~/thesisENV# ./sdn_qos/scripts/route_setting.sh
setting the default route for router s1r1...

{"switch_id": "0000000000000011", "command_result": [{"result": "success", "details": "Add route [route_id=1]"}]}]
0
setting the default route for router s1r4...

{"switch_id": "0000000000000041", "command_result": [{"result": "success", "details": "Add route [route_id=1]"}]}]
0
setting static route for s1 ...

{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "Add route [route_id=1]"}]}]
0
setting static route for s1 ...

{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "Add route [route_id=2]"}]}]
0
```

شکل ۲۶: اجرای اسکریپت route_setting.sh

۷،۲،۳. انجام تنظیمات QoS و ورودی جریان‌ها

به منظور انجام تنظیمات صف‌ها، مطابق جدول ۳ و همچنین نصب ورودی‌های جریان مطابق جدول ۴، دستورهای لازم در اسکریپت diffserv_qos_script.sh نوشته‌شد که در این مرحله اجرامی‌شود.

شکل ۲۷: اجرای اسکریپت diffserv_qos_script.sh

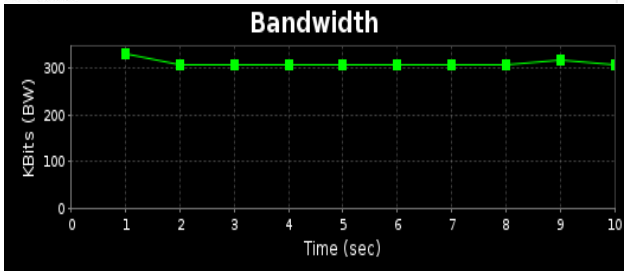
۸،۲،۳. نتایج

با پیاده‌سازی انجام‌شده، نود سرور h1r1 پورت‌های 5001, 5002, 5003 را با پروتکل UDP شنودمی‌کند. نود کلاینت h1r4، 1Mbps ترافیک UDP به پورت 5001، 300Kbps ترافیک UDP به پورت 5002 و 600Kbps ترافیک UDP به پورت 5003 سرور h1r1 ارسال‌می‌کند.

نتایج به‌دست‌آمده با استفاده از ابزار IPerf3/JPerf نشان‌می‌دهد که برای ترافیک علامت‌گذاری‌شده با کد AF41 ارسالی به پورت 5003، حداقل پهنای باند 500Kbps و برای ترافیک علامت‌گذاری‌شده با کد AF31 ارسالی به پورت 5002، حداقل پهنای باند 200Kbps تضمین شده‌است. در زمان ارسال ترافیک کلاس‌های AF، پهنای باند ترافیک بهترین تلاش و ارسالی به پورت 5001، محدود شده‌است. نتایج پورت‌های 5001، 5002، 5003 در شکل‌های ۲۸، ۲۹و ۳۰ مشاهده‌می‌شوند.

jp5002output

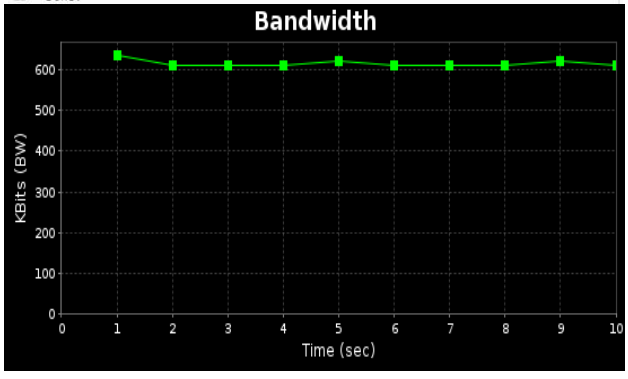
```
iperf -c 172.16.20.10 -u -P 1 -i 1 -p 5002 -f k -b 300.0K -t 10 -T 1
------------------------------------------------------------
Client connecting to 172.16.20.10, UDP port 5002
Sending 1470 byte datagrams, IPG target: 38281.25 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.30.10 port 43754 connected with 172.16.20.10 port 5002
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  40.2 KBytes   329 Kbits/sec
[  3]  1.0- 2.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  2.0- 3.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  3.0- 4.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  4.0- 5.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  5.0- 6.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  6.0- 7.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  7.0- 8.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  8.0- 9.0 sec  38.8 KBytes   318 Kbits/sec
[  3]  9.0-10.0 sec  37.3 KBytes   306 Kbits/sec
[  3]  0.0-10.1 sec   378 KBytes   307 Kbits/sec
[  3] Sent 263 datagrams
[  3] Server Report:
[  3]  0.0-10.5 sec   378 KBytes   294 Kbits/sec   0.000 ms    0/  263 (0%)
Done.
```



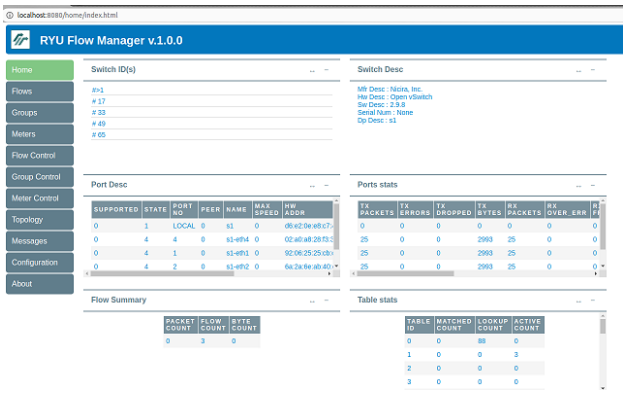
شکل ۲۸: خروجی ترافیک ارسالی به پورت 5002

jp5003output

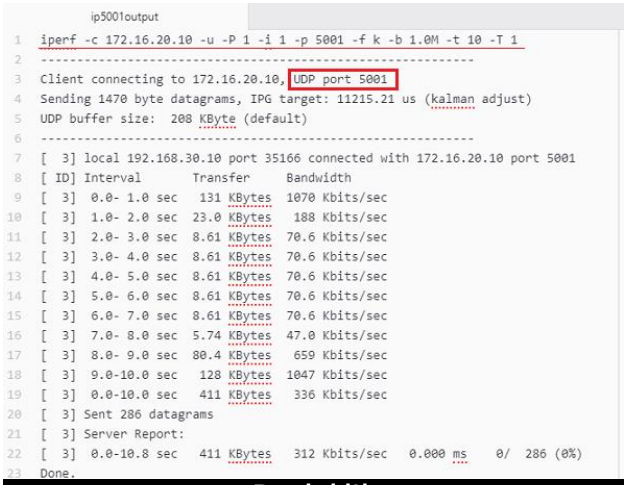
```
iperf -c 172.16.20.10 -u -P 1 -i 1 -p 5003 -f k -b 600.0K -t 10 -T 1
------------------------------------------------------------
Client connecting to 172.16.20.10, UDP port 5003
Sending 1470 byte datagrams, IPG target: 19140.62 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.30.10 port 54533 connected with 172.16.20.10 port 5003
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 1.0 sec  77.5 KBytes   635 Kbits/sec
[  3]  1.0- 2.0 sec  74.6 KBytes   612 Kbits/sec
[  3]  2.0- 3.0 sec  74.6 KBytes   612 Kbits/sec
[  3]  3.0- 4.0 sec  74.6 KBytes   612 Kbits/sec
[  3]  4.0- 5.0 sec  76.1 KBytes   623 Kbits/sec
[  3]  5.0- 6.0 sec  74.6 KBytes   612 Kbits/sec
[  3]  6.0- 7.0 sec  74.6 KBytes   612 Kbits/sec
[  3]  7.0- 8.0 sec  74.6 KBytes   612 Kbits/sec
[  3]  8.0- 9.0 sec  76.1 KBytes   623 Kbits/sec
[  3]  9.0-10.0 sec  74.6 KBytes   612 Kbits/sec
[  3]  0.0-10.0 sec   752 KBytes   614 Kbits/sec
[  3] Sent 524 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec   752 KBytes   614 Kbits/sec   0.000 ms    0/  524 (0%)
Done.
```



شکل ۲۹: خروجی ترافیک ارسالی به پورت 5003



شکل ۳۱: رابط گرافیک ابزار مدیریت جریان



شکل ۳۰: خروجی ترافیک ارسالی به پورت ۵۰۰۱

۴. نتیجه‌گیری

تضمین کیفیت سرویس کامل برای انواع برنامه‌ها با معماری کنونی شبکه که مبتنی بر مدل بهترین تلاش[۳۲] است، امکان‌پذیر نیست. برنامه‌های مختلف، نیازهای خدماتی متفاوتی دارند که تأمین آن‌ها مستلزم مدیریت پویای منابع شبکه است. در این مقاله، راه‌حلی براساس معماری SDN جهت مدیریت کیفیت سرویس ارائه‌شد که از مدل خدمات متمایز استفاده‌می‌کند. مکانیزم خدمات متمایز، منابع را بر اساس کلاس‌های مختلف ترافیک تخصیص‌می‌دهد. دراین روش، همه جریان‌هایی که به یک کلاس تعلق‌دارند، به‌طور یکسان هدایت‌می‌شوند.

نتایج به‌دست‌آمده از شبیه‌سازی‌ها، نشان‌دهنده عملکرد مطلوب چهارچوب معرفی‌شده در تأمین نیازهای جریان‌های ترافیکی و استفاده بهینه و حداکثری از منابع شبکه می‌باشد.

یکی از نیازمندی‌های هر سیستم مدیریت سیاست شبکه، داشتن رابط گرافیکی مناسب برای انجام عملیات مدیریتی مانند گزارش‌گیری و اجرای تنظیمات است. بدین منظور، با استفاده از کدهای متن باز و انجام تغییرات لازم، یک ماژول مدیریت جریان معرفی‌شد. گزارش‌گیری از شبکه امکان‌پذیرشده، ولی اعمال تنظیمات از طریق آن، ممکن نیست. با ارائه راه‌حلی، این مشکل نیز حل خواهدشد.

مراجع

[1] Guy Pujolle; *Software Networks (Virtualization, SDN, 5G and Security)*, Volume 1, Wiley, 2015.

[2] Paul Goransson and Chuck Black; *Software Defined Networks (A Comprehensive Approach)*, Second Edition, Elsevier, 2017.

[3] William Stallings; *Foundations of Modern Networking (SDN, NFV, QoE, IoT and Cloud)*, First Edition, Pearson, 2016.

[4] Open Data Center Alliance; *Software-Defined Networking Rev. 2.0 (Master Usage Model)*, ODCA White Paper, 2014.

[5] Liehuang Zhu, Md Monjurul Karim, Kashif Sharif, Fan Li, Xiaojiang Du, and Mohsen Guizani, *SDN Controllers: Benchmarking & Performance Evaluation*, IEEE JSAC, 2019.

[6] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Verı´ssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig, *Software-Defined Networking: A Comprehensive Survey*, Proceedings of the IEEE, Vol. 103, No. 1, January 2015

[7] S. Jain et al.; B4: *Experience with a globally-deployed software defined WAN*, ACM SIGCOMM Computer Communication Review. Vol. 43. 4. ACM. 2013, pp. 3–14.

[8] Open Networking Foundation; *OpenFlow Switch Specification version 1.5.1*, 2015.

[9] *Open Networking Foundation; OpenFlow Switch Specification version 1.3.5*, 2015.

[10] P. Rygielski, M. Seliuchenko, S. Kounev, M. Klymash, *Performance Analysis of SDN Switches with Hardware and Software Flow Tables*.

[11] Deepak Singh, Bryan Ng, Yuan-Cheng Lai, Ying-Dar Lin, Winston K.G. Seah, *Modelling Software-Defined Networking: Software and hardware switches*, Journal of Network and Computer Applications, 2018

[12] https://docs.openvswitch.org/en/latest/intro/what-is-ovs/

[13] https://github.com/openvswitch/ovs/blob/master/Documentation/tutorials/ovs-advanced.rst

[14] OpenDaylight: A Linux Foundation Collaborative Project, Available: https://www.opendaylight.org/

[15] Big Switch Networks, Project Floodlight, Available: http://www.projectfloodlight.org/floodlight/

[16] POX Controller Manual, Current Documentation, Available: https://noxrepo.github.io/pox-doc/html/

[17] P. Berde, M. Gerola, J. Hart et al., *Onos: Towards an open, distributed sdn os*, in Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, ser. HotSDN '14. ACM, 2014, pp. 1–6.

[18] RYU the Network Operating System, Documentation, Available: https://ryu.readthedocs.io/en/latest/
[19] Faucet SDN Foundation Repository Maintains RYU Controller Codes, Available: https://github.com/faucetsdn/ryu
[20] VMWare NSX Controller, Documentation, Available: https://docs.vmware.com/en/VMware-NSX-T-Data-Center/2.3/com.vmware.nsxt.install.doc/GUID-91955EED-05C4-4BED-88C3-FB5ED7420BAE.html
[21] Tim Szigeti, Robert Barton, Christina Hattingh and Kenneth Briley, Jr., *End-to-End QoS Network Design; Quality of Service for Rich-Media & Cloud Networks*, Second Edition, Cisco Press, 2014
[22] https://github.com/amirashoori7/sdn_qos

پی‌نوشت

[1] Data/Forwarding Plane
[2] Control Plane
[3] Open Network Foundation
[4] PBNM, Policy Based Network Management
[5] Differentiated Service (DiffServ)
[6] Southbound API
[7] East/West-bound API
[8] Northbound API
[9] Flow Table
[10] Group Table
[11] Flow Entry
[12] Matching
[13] Instructions or Actions
[14] Software Switch (Soft Switch)
[15] Hypervisor
[16] Single Threaded Entity
[17] Asynchronous Messages
[18] Delay
[19] Jitter
[20] Packet Loss
[21] Integrated Service (Int-Serv)
[22] Type of Service (ToS)
[23] Differentiated Service Code Point (DSCP)
[24] Per Hop Behavior (PHB)
[25] Class Selector
[26] Assured Forwarding
[27] Expedited Forwarding
[28] Drop Precedence
[29] Egress Rate
[30] Ingress Rate
[31] Top of Rack Switch (ToR)
[32] Best-Effort