2022 IEEE 23rd International Conference on High Performance Switching and Routing (HPSR) | 978-1-6654-0607-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/HPSR54439.2022.9831289

RPQ: Resilient-Priority Queue Scheduling for Delay-Sensitive Applications

Xinqiao Li*, Mingyuan Liu*, Nan Cheng‡, Kang Liu*, Wei Quan*†, Liang Guo§, and Yajuan Qin*

* School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China.
† Department of New Networks, Peng Cheng Laboratory, Shenzhen 518040, China.
‡ State Key Lab of ISN and School of Telecommunications Engineering, Xidian University, Xi'an 710071, China.
§ China Academy of Information and Communications Technology, Beijing 100191, China.
Email:{lixinqiao; mingyuanliu}@bjtu.edu.cn; dr.nan.cheng@ieee.org; {lkseagle; weiquan}@bjtu.edu.cn; guoliang1@caict.ac.cn; yjqin@bjtu.edu.cn

***Abstract*—With the continuous development of autonomous vehicles, telemedicine, digital media and other time-sensitive applications, a soaring number of network services have high demand for the quality of service (QoS) with extra low delay and jitter. Traditional network architecture only offers best-effort services which cannot meet the stringent delay and jitter requirements. In this paper, we propose a resilient-priority queue scheduling algorithm (RPQ) for delay-sensitive services. RPQ can guarantee stable delay in a fine-grained manner. Particularly, on the premise of meeting the delay requirements of high priority streams, RPQ can give consideration to the delay requirements of lower priority streams depending on its resilient scheduling mechanism. We implement RPQ on programmable switch. The experimental results show that RPQ not only guarantees QoS with low delay and low jitter for delay-sensitive streams but also improves network throughput by comparing with the existing solutions, i.e., SP-PIFO and WRR.**

***Index Terms*—queue scheduling, delay-sensitive, QoS, programmable data planes**

I. Introduction

Nowadays, various kinds of network applications, such as streaming media and online games, require low delay guarantees [1]. The traditional Internet adopts a well-known best-effort design pattern, which can not provide strong guarantee for delay-sensitive applications. Past QoS architectures such as *IntServ* and *Diffserv* have not been widely developed partly because they lack a global view of the network [2]. Software Defined Network (SDN) is a new architecture which separates the control planes and the forwarding planes of Network [3]–[5]. It has the advantages of centralized control and programmable control planes [6]. With the development of SDN, protocol independent switch architecture and P4 switch based on this architecture are proposed successively, and the programmability of data planes of SDN is further enhanced [7]–[9]. It provides a variety of metadata which allows users to obtain the status of the switch and relevant information of internal data packets [10]–[12].

Recently, many researchers are committed to guarantee quality of service (QoS) for applications based on SDN. Joakim *et al*. propose a QoS architecture that extends ONOS with dynamic management of traffic control queues in OpenFlow switches [13]. Shi *et al*. propose a cross-layer QoS publishing and subscription communication infrastructure supporting SDN, aiming to build the Internet of Things platform and improve the QoS of event delivery [14]. For streaming media, Zhu *et al*. investigate a framework of the QoS of streaming media, which can provide accurate QoS guarantee and fast routing [15]. Liu *et al*. propose a novel min-cost QoS routing algorithm (MCQRA) for SDN-WMN to find out the path with minimum cost more efficiently, meeting the demands of QoS including bandwidth, delay and packet loss [16]. However, these works focus on using the global control characteristics of SDN to study routing algorithms and can not provide fine-grained delay guarantee.

Queue management and scheduling is an effective way to ensure QoS. Traditional queue scheduling algorithms include weighted round robin (WRR), weighted fair queuing (WFQ) and default round robin (DRR) [17]. WRR assigns a weight to each sub queue and uses the weight on these sub queues to determine the number of listed packets. When a sub queue has excess bandwidth, WFQ will redistribute the bandwidth to other sub queues in proportion to the allocated weight that improves bandwidth efficiency. DRR scheduler improves fairness in throughput between streams. However, these traditional scheduling methods do not take into account the allowable delay of each process. Belma *et al*. propose a queue scheduling algorithm called p4QoS based on P4 programmable switch which embed QoS requirements into the data packet and used P4 programmable switch to process traffic based on it [18]. However, it ignores the contention of priority queues for transmission resources when the network load is large, and it reduces the delay by reducing the switch utilization through packet loss which results in a serious decline of throughput. Albert *et al*. implement SP-PIFO based on strict-priority queues of P4 programmable switch which guarantees QoS of traffic by dynamically changing the priority of the packet [19], but SP-PIFO has the problem that the low priority queue cannot serve for a long time when there are a number of priorities. At 23 s, the transmission of high priority flow is completed, and the bandwidth of medium and low priority flow is restored to 300 Mbps and 100 Mbps. At 43 s, the transmission of medium priority traffic is completed, and the

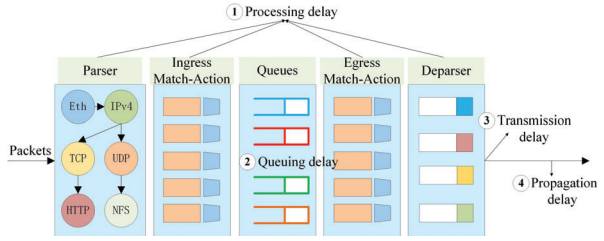

Fig. 1: The composition of delay.

bandwidth of low priority traffic becomes about 450 Mbps.

In this paper, we analyze the delay composition of packets in the transmission process and propose a **R**esilient-**P**riority **Q**ueue scheduling algorithm (RPQ) for delay-sensitive applications based on SDN and programmable data planes. It is worth noting, different from strict-priority queue, RPQ gives consideration to streams of all priority and effectively solves the hunger problem of low priority queues. Firstly, RPQ calculates the delay of data packets in each queue through the global view of SDN controller and packet metadata of programmable switch. Then, RPQ compares the real-time delay with the required delay in the order from high priority to low priority. Finally, if there are packets whose delay does not meet the requirements, RPQ regulates the delay of packets by adjusting the weight of queues.

The main contributions of this paper are as follows.

1) To accurately calculate the delay of data packets in the transmission process, we analyse the delay composition of network based on programmable switch.
2) We propose a resilient-priority queue scheduling algorithm (RPQ) to guarantee low delay and low jitter of delay-sensitive applications leveraging the programmable data planes.
3) We compare RPQ with other two queue scheduling algorithms. The experimental results show that RPQ not only ensures the deterministic delay of data flow, but also has high bandwidth utilization.

The remainder of this paper is organized as follows. Section II analyzes packet delay in detail. The resilient-priority queue scheduling algorithm is introduced in Section III. Section IV presents the performance evaluations for the proposed algorithm. Finally, we conclude this paper in Section V.

II. System Model

In this section, we provide the delay model of network composed of programmable switches. Assuming that the routing path of flow l is $path_l$, T^l denotes the end-to-end delay of flow l along path $path_l$, as shown in the Fig. 1, T^l is composed of processing delay, queuing delay, transmission delay and propagation delay.

Processing delay refers to the time required by the software switch to parse packet information, query matching flow table, read and write registers, check and reseal, etc. For a programmable software switch, the table, action and other processing behaviors have been determined before the switch starts, the value of this part can be regarded as a constant and can be measured. Queuing delay refers to the time from the data packet entering the queue to leaving the queue and it generates when the arrival speed of the packets exceeds transmission speed of the router. The main influencing factors of queuing delay are router utilization and the buffer size of the switch. Transmission delay refers to the time required for a node to make data enter the transmission media when sending data, that is, the time required for a switch to send a data frame or for a switch to receive a data frame. Propagation delay refers to the transmission time of electrical or optical signals on the link, which is related to the media used for transmission and the transmission distance.

Let T_i denote the time the packet has spent on each switch on path $path_l$, T_i^{proc} denote processing delay, T_i^{queue} denote queuing delay, T_i^{trans} denote transmission delay and T_i^{prop} denote propagation delay. T_i can be denoted by

$$T_i = T_i^{proc} + T_i^{queue} + T_i^{trans} + T_i^{prop}. \tag{1}$$

For a transmission path with n relay nodes, the end-to-end delay is defined as

$$T^l = \sum_{i=1}^{n}(T_i^{queue} + T_i^{trans} + T_i^{proc}) + \sum_{i=1}^{n+1} T_i^{prop}. \tag{2}$$

Assuming that all programmable switches have the same configuration, thus, each programmable switch has the same processing delay. Assuming that the transmission medium between each two switches is the same, thus the propagation delay between each two switches is the same. Let L denote the average size of packets, and B_i denote the real-time bandwidth of the switch, P_i denote the real-time queue rate, (2) is converted to

$$\begin{aligned} T^l &= \sum_{i=1}^{n} T_i^{queue} + \sum_{i=1}^{n}(\frac{L}{B_i}) + n * T^{proc} + (n+1) * T^{prop} \\ &= \sum_{i=1}^{n}(T_i^{queue} + P_i^{-1}) + n * T^{proc} + (n+1) * T^{prop}. \end{aligned} \tag{3}$$

In (3), T^{proc} and T^{prop} can be measured by sending probe packet with the simple topology in Fig. 2. There is no queuing delay due to low load, and the transmission delay is negligible. In Fig. 2, the total delay $T^{'}$ of the topology with one switch denote to

$$T^{'} = T^{proc} + 2T^{prop}. \tag{4}$$

For the topology of two switches, the total delay $T^{''}$ is

$$T^{''} = 2T^{proc} + 3T^{prop}. \tag{5}$$

Combined with (4) and (5), we can get the T^{proc} and T^{prop}. Therefore, for a certain network topology based on programmable switches, queuing delay and transmission speed are the decisive factors to end-to-end delay according to (3). (3) is the basis for RPQ to calculate packet delay and schedule queues resiliently.

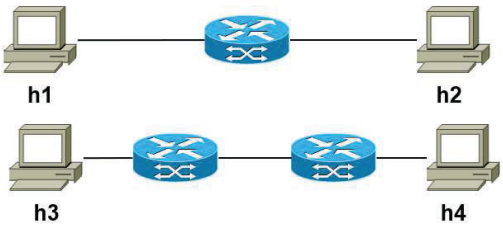

Fig. 2: The topology to measure processing delay and propagation delay.

III. Proposed RPQ Solution

This section introduces the workflow of RPQ and the system we built to realize RPQ based on the programmable data plane.

A. Programmable Architecture

In order to calculate the real-time delay of packets more accurately to provide effective delay guarantee for time delay-sensitive services, we construct a programmable architecture based on P4 programmable switch, and implement our RPQ on it. The proposed programmable architecture is shown in Fig. 3. P4 is a protocol independent programmable networking device language, which supports the analysis, modification and encapsulation of packets. P4 programmable switch maintains metadata for incoming packets such as its in and out queue timestamp, queue depth when entering the queue and so on for users to query, and users can configure the queue depth, queue rate and read or write registers of P4 programmable switch through the controller.

The process of forwarding packets by the system is as follows. Firstly, after a packet is transmitted into the switch, the parser block parses the packet header to obtain the packet quintuple. Then, in the ingress pipeline processing, the programmable switch obtains the priority information of the packet according to match-action tables and writes the information into the TOS field of IP header. The switch then selects a queue for the packet based on the value of the TOS field. The deparser block reassembles the header of the packet. Finally, the packet are forwarded out through the corresponding port.

B. RPQ Workflow

Based on programmable architecture, we propose a resilient-priority queue scheduling algorithm to meet the delay requirements of delay-sensitive services.

We set up three types of priority queues in the system and each queue carries a weight value. The high priority queue carries delay-sensitive applications such as driverless services, and it has the highest weight. The medium priority queue carries generally delay-sensitive traffic such as ordinary video services, and it has a medium weight. The low priority queues carry delay-insensitive applications such as web services, and it have the lowest weight. The weight of a queue is proportional to the number of dequeue packets per unit time. The key of RPQ algorithm is to dynamically adjust the weight

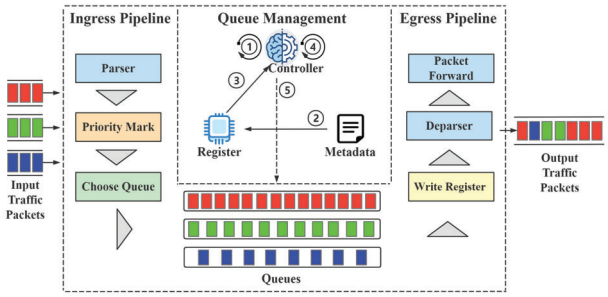

Fig. 3: The architecture of RPQ.

Algorithm 1 Queue scheduling mechanism

Input:
Network topology
Delay demand of each queue: T^h_{max},T^m_{max},T^l_{max}
Output:
Queue weight adjustment policy
1: Get the network topology
2: Initialize the weight of each queue
3: Get propagation delay and processing delay by (4) and (5)
4: **while** TRUE **do**
5: Get the queuing delay of each queue of each switch by reading the register
6: Calculate the total real-time delay of each queue by (3)
7: Generate queue weight adjustment strategy
8: Sleep 1 second
9: **end while**

of queues according to the real-time delay of packets so as to continuously meet the delay requirements of services.

As shown in Fig. 3 and algorithm 1, the queue management process of the RPQ is divided into the following five steps. In the first step, the controller sends a probe packet to obtain the propagation delay under the current network topology and the processing delay of the switch. According to the user's demand for service flow delay, the controller generates the corresponding priority configuration flow table and sends it to the programmable switch. In the second step, the switch writes the time stamp of data packets in and out of the queue into the register in egress pipeline processing. In the third step, the controller reads the time stamp stored in the register and calculates the queuing delay of each queue. In the fourth step, the controller calculates the real-time delay of each queue according to the delay information obtained in step 1 and step 3, compares it with the allowable delay range and generates the weight adjustment strategy of each queue. In the fifth step, the controller sends the queue weight adjustment strategy to the programmable switch through p4runtime protocol.

The queue weight adjustment strategy is as follows. Let T_{max} denote the maximum allowable delay of one flow. We define three states for flow according to the delay of the packet in it. When the flow delay is less than $0.75T_{max}$, it is in ideal state. When the flow delay is between $0.75T_{max}$ and $0.9T_{max}$,

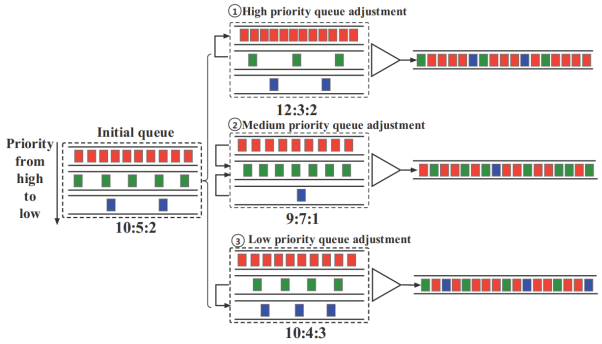

Fig. 4: The queue weight adjustment of RPQ.

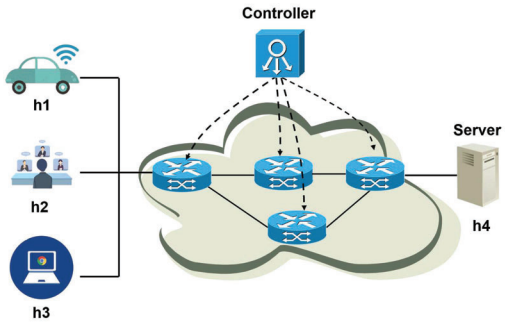

Fig. 5: Experimental scenario.

Algorithm 2 Queue weight updating

Input:
Delay demand of each queue: T^h_{max},T^m_{max},T^l_{max}
Real-time delay of each queue: T_h,T_m,T_l
Output:
Queue weight adjustment policy

/**high priority queue weight adjustment**/
if $T_h > 0.9{*}T^h_{max}$ **then**
  if $T_m > 0.9{*}T^m_{max}$ **then**
    Update $W_h = W_h + 0.9 * W_m$
  else
    Update $W_h = W_h + 0.5 * W_m$
  end if
end if
/**medium priority queue weight adjustment**/
if $T_m > 0.9{*}T^m_{max}$ **then**
  if $T_h < 0.75{*}T^h_{max}$ **then**
    Update $W_m = W_m + 0.1 * W_h$
  end if
  if $T_l < 0.9{*}T^l_{max}$ **then**
    Update $W_m = W_m + 0.2 * W_l$
  else
    Update $W_m = W_m + 0.9 * W_l$
  end if
end if
/**low priority queue weight adjustment**/
if $T_l > 0.9{*}T^l_{max}$ **then**
  if $T_m < 0.75{*}T^m_{max}$ **then**
    Update $W_l = W_l + 0.1 * W_m$
  end if
end if

it is in normal state. When the flow delay is greater than $0.9T_{max}$, it is in congestion state. Let T_h, T_m, T_l denote the real-time delay of the high, medium, and low priority queues, T^h_{max}, T^m_{max}, T^l_{max} denote the maximum acceptable delay of the high, medium, and low priority queues, and W_h, W_m, W_l denote the weight of the high, medium, and low priority queues.

The resilient queue weight adjustment strategy is shown in Fig. 4 and algorithm 2. For the three types of priority queues, when they are in ideal or normal state, there is no need to actively preempt the weight of other queues. When high priority queue is in congestion state, if medium priority queue is also in congestion state, the network state is poor this moment. In order to meet the delay requirements of high priority traffic, high priority queue will preempt ninety percent of weight from medium priority queue. If medium priority queue is in a ideal or normal state, high priority queue will preempt half of weight from it. When medium priority is in congestion state, if high priority queue is in ideal state, that is, the high priority queue holds redundant transmission resources, medium priority queue will preempt one tenth of weight from it. If low priority queue is in ideal or normal state, medium priority queue will preempt one fifth of the weight from it. If low priority queue is in congestion state, medium priority queue will preempt ninety percent of the weight from it. When low priority queue is in congestion state, if medium priority queue is in ideal state, the low priority queue will preempt one tenth of weight from it. The resilient queue weight adjustment strategy enables the high priority queue to obtain appropriate weight rather than excessive weight. Therefore, RPQ can balance the requirements of streams of all priority.

IV. Experiments and Performance Results

A. Setting of the experiments

In order to evaluate our solution, we install P4-based software switch on the general physical server and install ONOS as the controller. Behavioral model version 2 (bmv2) [20] takes a JSON file generated from P4 program as input and interprets it to implement the packet-processing behavior. ONOS is the first open source SDN network operating system, mainly for service providers and enterprise backbone networks. The experimental scenario is as Fig. 5. *h1* represents high priority service like autonomous vehicles, *h2* represents medium priority service like multi-person conferences, and *h3* represents low priority service like general web service.

B. Performance and Results

1) System Function: The experiment is designed to verify basic function of RPQ, which is to provide efficient services for high-priority streams while balancing the other priority

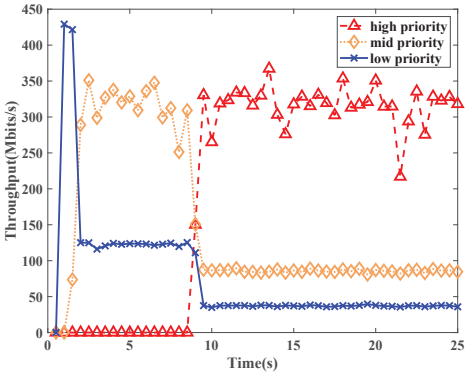

Fig. 6: Comparison of three priority flows with throughput.

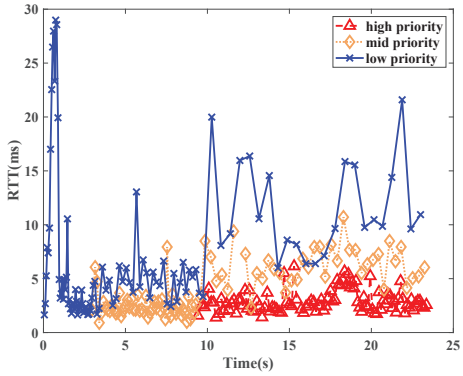

Fig. 7: Comparison of three priority flows with RTT.

streams. The specific process is as follows. *h3* establishes a TCP link with *h4* as the background traffic. *h2* sends 8 Gb of data to *h4*. *h1* sends 4 Gb of data to *h4*.

Fig. 6 shows that the bandwidth variation curve of high, medium and low priority flows in the experiment. At 3 s, the medium priority queue preempts the bandwidth of low priority queue after entering the system, and its bandwidth is about 300 Mbps, the bandwidth of low priority flow is about 100 Mbps. At 9 s, the high priority flow enters the system, high priority queue preempts the bandwidth of the medium priority queue and the low priority queue. The bandwidth of high priority flow is about 300 Mbps, the bandwidth of medium priority flow is about 90 Mbps, and the bandwidth of low priority bandwidth is about 40 Mbps.

Fig. 7 shows that the delay variation curve of high, medium and low priority flows in the experiment. We intercepts the part before the end of high priority flow transmission of the curve. It can be seen from the Fig. 7 that after the high priority flow enters the system, in order to ensure low delay, the RTT of the medium and low priority flow rises, and the RTT of the high priority flow remains between 2 ms and 5 ms. To sum up, our system can provide efficient services for high-priority traffic. Besides, on the premise of meeting the delay requirements of high priority streams, RPQ prevents high priority streams from preempting too much bandwidth with low priority and medium priority streams.

2) Delay and jitter: This experiment is to test the performance of delay guarantee of RPQ for high priority flows when the network fluctuates.

We compare the performance of RPQ with two existing priority scheduling algorithms WRR [17] and SP-PIFO [19]. WRR is a simple queue scheduling algorithm that can avoid the hunger problem. The main idea of the algorithm is to give each queue a weight. The weight of the queue with high priority is higher, and the dequeue speed of packets is faster, so as to realize that the high priority flows get better service. SP-PIFO is a programmable packet scheduler which closely approximates the behavior of PIFO queues using strict-priority queues. Its key point is to dynamically adapt the mapping between packet ranks and available strict-priority queues to minimize the scheduling errors with respect to an ideal PIFO.

Table I: Average delay and jitter comparision of high-priority streams of RPQ, SP-PIFO and WRR.

	RPQ	SP-PIFO	WRR
Delay(ms)	2.951	4.143	3.800
Jitter(ms)	0.93427	1.62911	1.17841

The topology diagram is also as Fig. 5. The specific process is as follows. *h1*, *h2* and *h3* continuously keep a request to *h4* for 30 s from the same time. We set the delay demand to 3.5 ms for the high-priority stream, 10 ms for the medium-priority stream, and no delay demand for the low-priority stream. At 10 s, we manually limit the network bandwidth to simulate the fluctuation of the network. And we recover the link bandwidth at 20 s.

Fig. 8 shows the curve of RTT with time of high priority flows under three scheduling algorithms in the experimental process. Table I shows the mean value and jitter of RTT of high priority flows under three scheduling algorithms. As shown in Fig. 8, when the network fluctuates at 10 s, RPQ allows to sacrifice the transmission performance of medium and low priority flows to provide delay guarantee for high priority flows due to its adaptive ability. However, under the WRR and SP-PIFO algorithms, the network fluctuation will affect the delay performance of three types of priority flows at the same time. According to the results calculated in Table I, RPQ is better than WRR and SP-PIFO in terms of delay and jitter guarantee of high priority flow.

3) Throughput and flow completion time: This experiment is to test the performance of RPQ in terms of throughput and flow completion time. The specific process is that *h1*, *h2* and *h3* transmit 4000 Mb data to *h4* from the same time. We also set the delay demand to 3.5 ms for the high-priority stream, 10 ms for the medium-priority stream, and no delay demand for the low-priority stream.

Fig. 9 shows the curve of the total throughput of the system with time under the three algorithms. Table II shows the average throughput and flow completion time of three types of algorithms. As shown in Fig. 9, even if there are network fluctuations, the throughput of RPQ is between 400 Mbps and 500 Mbps at most times, which is higher than the throughput of SP-PIFO and WRR. As shown in Table II, on the premise

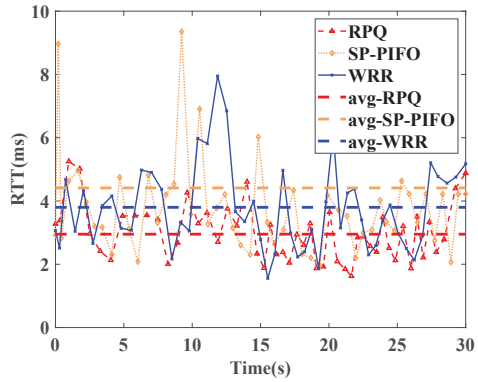

Fig. 8: RTT comparison of high-priority streams of RPQ, SP-PIFO and WRR.

Table II: Average throughput and flow completion time of RPQ, SP-PIFO and WRR.

	RPQ	SP-PIFO	WRR
Throughput(Mbps)	441.685	358.429	385.321
Flow Completion Time(s)	26.7	32.3	29.5

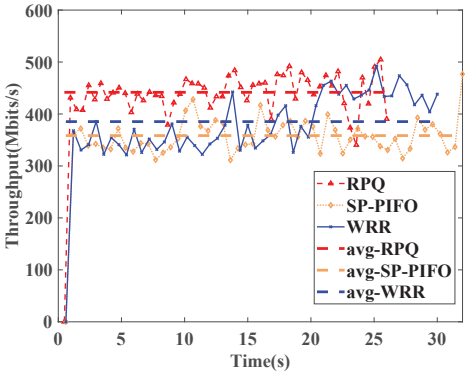

Fig. 9: Throughput comparison of RPQ, SP-PIFO and WRR.

of meeting the delay requirements, the average throughput of RPQ is 441.685 Mbps which is higher than the one of other two algorithms. Meanwhile, the flow completion time of RPQ is also shorter when transmitting data streams of equal size.

V. Conclusion

In this paper, we have proposed a resilient-priority queue scheduling algorithm to meet the low delay requirements of delay-sensitive applications. We establish a delay model for packet transmission. Besides, we describe in detail how to adjust the queue weight to ensure the low delay of data flows based on RPQ. The experimental results show that RPQ can provide differentiated services for multi-priority streams, which prevents high-priority streams from preempting too much bandwidth with low-priority and medium-priority streams. Meanwhile, RPQ also has good performance in throughput and flow completion time. In the future work, we will further research priority queue scheduling mechanism to improve the quality of service for multiple types of traffic.

Acknowledgement

This work is supported by the Natural Science Foundation of Beijing under Grant No. 4212010 and the Open Research Projects of Zhejiang Lab under Grant No. 2022QA0AB06.

References

[1] H. Zhang and W. Quan, "Networking automation and intelligence: A new era of network innovation," *Engineering*, 2021.
[2] J. Yan, H. Zhang, Q. Shuai, B. Liu, and X. Guo, "HiQoS: An SDN-based multipath QoS solution," *China Communications*, vol. 12, no. 5, pp. 123–133, 2015.
[3] J. Wang, J. Liu, H. Guo, and B. Mao, "Deep reinforcement learning for securing software defined industrial networks with distributed control plane," *IEEE Transactions on Industrial Informatics*, 2021.
[4] B. Mao, F. Tang, Z. M. Fadlullah, and N. Kato, "An intelligent route computation approach based on real-time deep learning strategy for software defined communication systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1554–1565, 2019.
[5] W. Quan, M. Liu, N. Cheng, X. Zhang, D. Gao, and H. Zhang, "Cybertwin-driven adaptive transmission scheduling for software defined vehicular networks," *IEEE Transactions on Vehicular Technology*, 2022.
[6] H. Ghalwash and C.-H. Huang, "A QoS framework for SDN-based networks," in *Proceedings of IEEE 4th International Conference on Collaboration and Internet Computing*. IEEE, 2018, pp. 98–105.
[7] N. Thazin, K. M. Nwe, and Y. Ishibashi, "End-to-end dynamic bandwidth resource allocation based on QoS demand in SDN," in *Proceedings of Asia-Pacific Conference on Communications*. IEEE, 2019, pp. 244–249.
[8] P. Podili and K. Kataoka, "Effective resource provisioning for QoS-aware virtual networks in SDN," in *Proceedings of IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–9.
[9] C. Yu, W. Quan, D. Gao, Y. Zhang, K. Liu, W. Wu, H. Zhang, and X. Shen, "Reliable cybertwin-driven concurrent multipath transfer with deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 8, no. 22, pp. 16 207–16 218, 2021.
[10] G. Liu, W. Quan, N. Cheng, D. Gao, N. Lu, H. Zhang, and X. Shen, "Softwarized IoT network immunity against eavesdropping with programmable data planes," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6578–6590, 2021.
[11] J. Shi, W. Quan, D. Gao, M. Liu, G. Liu, C. Yu, and W. Su, "Flowlet-based stateful multipath forwarding in heterogeneous Internet of Things," *IEEE Access*, vol. 8, pp. 74 875–74 886, 2020.
[12] M. Liu, D. Gao, G. Liu, J. He, L. Jin, C. Zhou, and F. Yang, "Learning based adaptive network immune mechanism to defense eavesdropping attacks," *IEEE Access*, vol. 7, pp. 182 814–182 826, 2019.
[13] J. Flathagen, T. M. Mjelde, and O. I. Bentstuen, "A combined network access control and QoS scheme for software defined networks," in *Proceedings of IEEE Conference on Network Function Virtualization and Software Defined Networks*. IEEE, 2018, pp. 1–6.
[14] Y. Shi, Y. Zhang, and J. Chen, "Cross-layer QoS enabled SDN-like publish/subscribe communication infrastructure for IoT," *China Communications*, vol. 17, no. 3, pp. 149–167, 2020.
[15] S. Zhu, Z. Sun, Y. Lu, L. Zhang, Y. Wei, and G. Min, "Centralized QoS routing using network calculus for SDN-based streaming media networks," *IEEE Access*, vol. 7, pp. 146 566–146 576, 2019.
[16] K. Liu, Y. Cao, Y. Liu, G. Xie, and C. Wu, "A novel min-cost QoS routing algorithm for SDN-based wireless mesh network," in *Proceedings of IEEE International Conference on Computer and Communications*. IEEE, 2016, pp. 1998–2003.
[17] S. Noda and K. Yamaoka, "Approach to optimal WRR weight assignment method in delay-limited environment," in *Proceedings of IEEE Annual Consumer Communications & Networking Conference*. IEEE, 2016, pp. 1113–1118.
[18] B. Turkovic, S. Biswal, A. Vijay, A. Hüfner, and F. Kuipers, "P4QoS: QoS-based Packet Processing with P4," in *Proceedings of International Conference on Network Softwarization*. IEEE, 2021, pp. 216–220.
[19] A. G. Alcoz, A. Dietmüller, and L. Vanbever, "SP-PIFO: Approximating Push-In First-Out Behaviors using Strict-Priority Queues," in *Proceedings of USENIX Symposium on Networked Systems Design and Implementation*, 2020, pp. 59–76.
[20] "P4 behavioral model," https://github.com/p4lang/behavioral-model, 2018.