Openflow-Extended Queue Scheduling for Multi-tenants in Access Network

Ting Wang[1(✉)] and Gang Liu[2]

[1] Software Engineering Institute, East China Normal University, Shanghai, China
twang@sei.ecnu.edu.cn
[2] Bell Labs, Nokia Shanghai Bell Corp., Shanghai, China
gang.i.liu@nokia-sbell.com

Abstract. Network slicing is a mechanism that could be used by operators to support multiple virtual networks over underlying hardware infrastructure. Typically, it could leverage Network Function Virtualization (NFV) technology to share the computing/storage resources in data center scenarios, for example, creating multiple VM (virtual machine) instances for different network functions or network tenants. Through different VM instances, most of computing-intensive core network applications (e.g. vEPC, vIMS) could be supported. In order to achieve this goal, in this paper we propose a novel Openflow-extended queue scheduling scheme for multi-tenants in the access network.

Keywords: Network function virtualization · Network slicing · Queue scheduling

1 Introduction

The idea of splitting the data plane and the control plane is widely accepted. As an open interface between the control plane and data plane, OpenFlow [1] was originally proposed and developed at Stanford University [2] and is now marshalled by the Open Networking Foundation (ONF) [3]. The Software-Defined Access Network (SDAN) concept [4] was introduced to extend the benefits of Software Defined Networking (SDN) and Network Function Virtualization (NFV) into broadband access [2,5]. However, there are still many open issues to be addressed. For example, how to slice fixed access networks? As shown in Fig. 2, how to support multi-tenants sharing the same ONU as well as the link resource with access nodes? How to identify traffics of different virtual network operators (VNOs) while guaranteeing strong isolation between them? How to provide complete logical resource for each VNO without any conflict? For example, complete VLAN tags (0-4096) could be employed by each VNO.

In traditional access network, a variety of different access technologies and modes (from ADSL to VDSL, EPON to GPON) would coexist or even collocate in a quite long period. The diversity of access technologies increases the cost of

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2022
Published by Springer Nature Switzerland AG 2022. All Rights Reserved
K. Wu et al. (Eds.): ICECI 2021, LNICST 437, pp. 61–73, 2022.
https://doi.org/10.1007/978-3-031-04231-7_5

hardware maintenance and complicates network O&M. When operators want to migrate the line access technology (for example, from ADSL to VDSL or even G.fast), they have to manually upgrade the access device (at least LT board) accordingly. Furthermore, if operators want to deploy a specialized or customized (maybe not standardized) protocol or feature, it would be quite difficult, and they will basically have to wait for vendors to produce dedicated processing boards (LT board). Due to the highly coupled hardware and software, closed and proprietary device, and standard-oriented eco-system, the telecom-specific network equipment is prone to a lack of flexibility, and the operators usually have to change the hardware (e.g. LT board) so as to support new protocols or customized features.

Moreover, currently most of the existing access networks are dedicatedly deployed by the incumbent operators, with extremely high expenditure invested on infrastructure construction and access network deployment. Greenfield or alternative operator, who wants to extend their services to the same area, must deploy their own dedicated access infrastructure networks, though in fact the infrastructure deployed by the incumbent operator is underutilized with many unused optical fibers underground as well as other access resource. This model is accompanied with various disadvantages such as the investment waste for the operators and low resource utilization. Besides, the customers are impossible to make dynamic selections among different operators, incurring high cost on manually changing the customer lines. To deal with these issues, this paper aims to achieve a virtualized access network supporting multi-tenancy and flexible slicing through queue scheduling optimization. With this aim, this paper proposes an OpenFlow-enabled queue scheduling methodology for network slicing and traffic isolation. In order to facilitate the centralized queue management, we introduce a new "Enqueuer" module in the access controller and extend OpenFlow protocol [1] to indicate queue scheduler as well as queue property.

The remainder of this paper is organized as follows. Section 2 briefly reviews the related work. Section 4 describes the basic idea of the proposed solution. Section 5 presents the design details of the solution. Section 6 concludes this paper.

2 Related Work

Currently, there is little research on fixed access network sharing, especially on how to slice the access resource between ONU and access node. The most similar case is the slicing of forwarding devices.

For those OpenFlow-enabled forwarding devices (e.g. switch, router, etc.), "FlowVisor" solution could be a feasible choice to support multi-tenants [6]. As shown in Fig. 1, FlowVisor is a special OpenFlow controller that acts as a transparent proxy between OpenFlow switches and multiple OpenFlow controllers. FlowVisor creates rich "slices" of network resources and delegates control of each slice to a different controller which may belong to different VNOs, such as Alice Controller, Bob controller, and Cathy Controller. But in the "FlowVisor" solution, the InP controller could not obtain queue-relevant information from underlying devices. It means that the InP controller has no knowledge/capability to manage the queue scheduling. To address this problem, this paper extends

OpenFlow protocol to help controller get the queue information from underlying devices. In the InP controller, specific modules are designed to perform centralized queue scheduling and enable strong isolation between different VNOs. Y. Oktian, etc. [7] investigated different design approaches of SDN controller. They classified the methods into several design choices, where each design choice may influence several SDN issues such as scalability, robustness, consistency, and privacy. They further analyzed the advantages and disadvantages of each model regarding these matters.

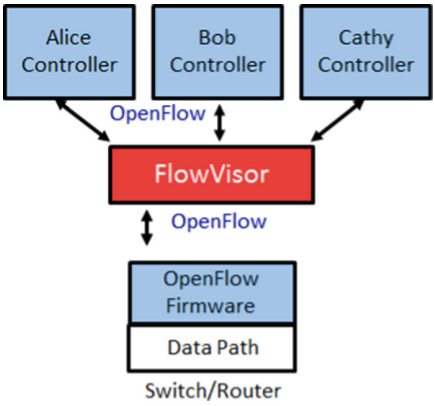

Fig. 1. Overview of FlowVisor scheme

Slices can be defined by any combination of switch ports (layer 1), source and destination Ethernet addresses or types (layer 2), source and destination IP addresses or types (layer 3), and source and destination TCP/UDP ports [8] or ICMP code/type (layer 4) [9]. It means each slice/VNO has its own flow space which is employed to distinguish traffic of different slices/VNOs.

FlowVisor enforces isolation between each slice, i.e., one slice cannot control another slice's traffic. Every slice has its own flow controller as well as the logical view. This is control plane isolation. As for the data plane isolation, FlowVisor could leverage hardware capability and OpenFlow protocol to provide bandwidth isolation. For example, OpenFlow protocol has already exposed hardware bandwidth slicing capabilities in the form of per-port queues. Thus, the traffic isolation could be guaranteed through queue scheduling, which means different queues could be designated to different slices/VNOs.

However, the FlowVisor solution has several drawbacks:

1. Since no specific mechanism/messages in OpenFlow protocol (latest 1.5 version) could help controller and switch to communicate queue-related information, e.g. queue scheduling algorithm, queue number, queue priority, queue affiliation, guaranteed/ceiling rate, and so on. Thus, FlowVisor could not obtain detailed queue information of underlying devices.

Consequently, FlowVisor has no knowledge of deciding which queue the flow/packet should be put into, let alone QoS assurance.
2. Generally, each slice/VNO could define its own Openflow table item, as shown in the left part of Fig. 3. But, basically VNOs have no knowledge of queue configuration, which is usually dominated by infrastructure operator. For the purpose of shielding or security, it is better not exposing queue details to VNOs. Thus, the VNO controller could not decide which queue the flow/-packet should be put into. It is necessary for the FlowVisor to inject queue value in each flow table item, which eventually would be delivered to the underlying switch, as shown in right part of Fig. 3. This injection action should be performed based on queue information, VNO property and even the flow characteristics. Currently, in FlowVisor solution, there is no specific module that could perform this action.

In terms of network customization, some researchers have made some trials on introducing SDN concept into traditional access networks. The three main problems in network management being addressed include: enabling frequent changes to network conditions and states, providing support for network configurations in a high-level language, and providing better visibility and control over tasks for performing network diagnosis and troubleshooting. To deal with the network configuration and management problems existed in the current state-of-the-art networks, H. Kim. etc. [10] employed SDN to improve the network management from various aspects. In addition to the systems themselves, various prototype deployments [11–14] have demonstrated how SDN can improve common network management tasks and enable business applications in campus and carrier networks.

Network slicing also has been discussed a lot in access network especially in the PON systems [14–19]. The major research activities focus on the physical or cross-layer protocol design for multiple concurrent network applications, e.g. 5G X-hauling, IoT services, and low-latency applications, etc.

In order to address the drawbacks of the existing work, we propose a novel OpenFlow-extended queue scheduling for multi-tenants in the access network.

3 Motivation and Problem Statement

Network operators are facing a very challenging problem, that is, the growth rate of revenue is far behind the growth rate of data traffic and cost. These differences in these growth rates lead to a significant income gap. In order to narrow the revenue gap and achieve a sustainable business model, one effective approach is to increase the revenue per bit through specialized network framework and customized service provision. In the highly competitive telecommunications industry, in order to create differentiation, network operators are willing to implement their own optimized protocols and provide subscribers with more and more customization possibilities. This appeal urges operators to abstract underlying network infrastructure and provides standard application programming interfaces at different levels. The concept of SDN, which separates the control plane

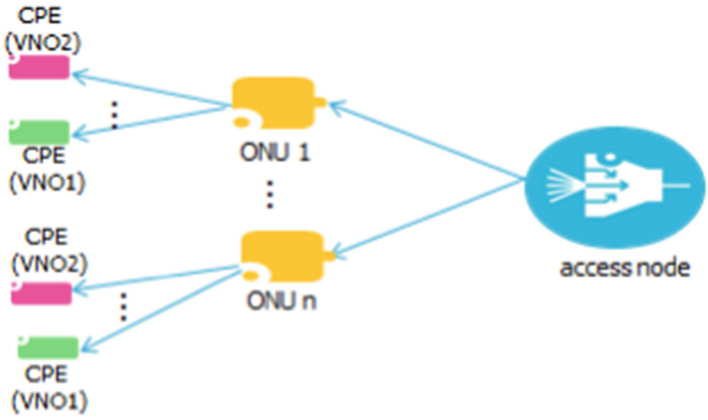

Fig. 2. Access network slicing

Table ID	Table item
0	Match{....}, action {Goto 100}
100	Match{ds=10.67.121.0, tcp_port=554}, action {output: 2}
...	...
300	Match{eth_type=arp}, action {drop}

Flow table in VNO controller

Table ID	Table item
0	Match{....}, action {Goto 100}
100	Match{ds=10.67.121.0, tcp_port=554}, action {output: 2, set_queue: 3}
...	...
300	Match{eth_type=arp}, action {drop}

Flow table in switch

Fig. 3. Queue value injection

from the data plane, just meets this requirement exactly. Moving most of the complexity of physical devices to a centralized computing pool allows easier management of those functions, and efficiently supports customized requirements in a software-defined way. Consequently, the telecommunication network can easily be programmed or reprogrammed in an IT manner in real-time. Operators can conveniently customize their environments to enable rapid service innovation.

Another approach to narrow the revenue gap is to resort to network virtualization and network slicing, which could lower the cost per bit. Network virtualization provides a powerful way to run multiple networks, each of which is customized to a specific purpose, at the same time over a shared substrate. With the benefit of network virtualization, network provider owns a physical access network that could support multi-tenancy, where network provider's access network may be used by other service providers or network tenants. In this way, the access network, as a highly multiplexed shared environment, could simultaneously offers multiple different tenants with on-demand use of network resources that may have different network characteristics. Each tenant would have its own dedicated network instance which provides complete functions just like a physical network element. Many virtual access node (VAN) instances with fully functionalities implemented on physical access node will be provided to different tenants.

Although the network customization and virtualization can effectively drive revenue growth and forge differentiation, there are still many technical issues to be solved for the operators during the actual deployment and implementation of network customization and virtualization. The key intractable issues are listed as below.

- What kind of architecture could enable programmability, automation and customization, and facilitate carrier to build highly scalable, flexible networks that readily adapt to changing business needs?
- What methodology could provide the ability to deliver new network capability and services without the need to configure individual devices or waiting for vendor releases?
- How to dynamically create the virtual instances on the physical device while providing support for flexibility? Through what means to manage/control/-maintain these virtual instances while reducing the OPEX?
- How to support customized requirements of multi-tenants? For example, how to make one MDU (Multiple Dwelling Unit) serve for multi-tenants simultaneously; How to provide flexible baseband protocol and configurable packet processing to different tenants?
- How to support different access technologies (e.g. ADSL, G.fast, EPON) simultaneously without dedicated processing board (line termination board)? For example, one virtual access node may support ADSL protocol at one moment, and support G.fast protocol (even EPON) at another moment.
- How to support dynamic port/link configuration for various deployment scenarios? How to dynamically change the serving object of one physical port on MDU or OLT (Optical Line Terminal)? It implies that one MDU/OLT physical port may serve for one VNO (Virtual Network Operator) at one moment, and serve for other VNOs at another moment.

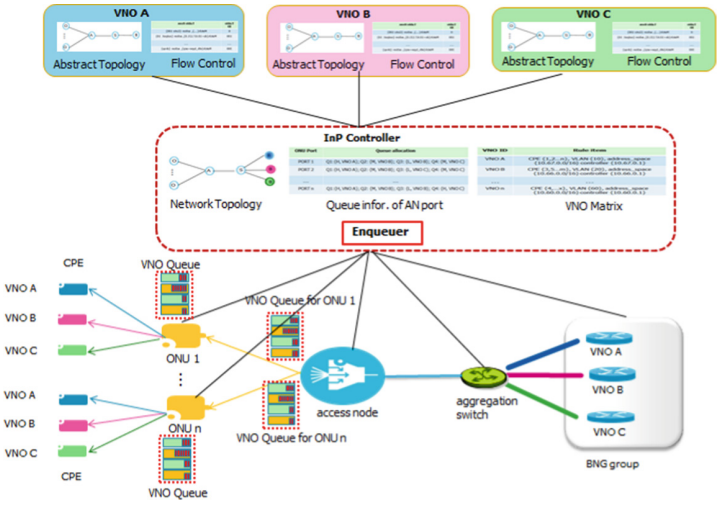

Fig. 4. Architecture of access network sharing

These emerging problems push the industry to explore new methodologies to allow enterprises and carriers to gain unprecedented programmability, automation, and network control, and enable them to build highly scalable and flexible networks that readily adapt to changing business needs. Software defined networking, decoupling the control plane and data plane of the network, has been considered as a potential solution, which can meet all these challenges. From the perspective of access network, SDN concept could offer a more intelligent and automatic network framework, which facilitates fast service provisioning and enables a wide variety of access technologies. Based on these observations, this paper aims to design a new SDN-based architecture for fixed access network, which can provide cost-efficient network control and management with high scalability and customization support.

4 Basic Idea of the Solution

In order to support multi-tenants sharing the same fixed access network, especially the access node and ONU, we extend OpenFlow protocol and introduce a queue scheduling methodology for traffic isolation and QoS guarantee. The whole architecture is shown in Fig. 4.

The main contributions of this paper can be summarized as below.

1. We introduce a queue scheduling mechanism for multi-tenants. According to the hardware capability, the queue scheduling could be implemented on Access Node or ONU. Option 1: the Access Node maintains a queue set (composed of several queues) for each ONU; In the queue set, different queues would be designated to different VNOs. Note that multiple queues could serve for one VNO. Option 2: ONU maintains only one queue set for different VNOs. In the queue set, different queues would be designated to different VNOs. The difference from option 1 is that there no specific queue set for each CPE.
2. We introduce a new module "Enqueuer" into InP controller (Infrastructure Provider controller), which is responsible for adding the queue action into flow table item. When VNOs deliver specific flow table item (mainly those configured with detailed output port) to InP controller, the "Enqueuer" module designates the queue value in the action field according to the queue information, VNO matrix and flow property. Then, the "Enqueuer" module injects the queue value in the original flow table item and delivers it to underlying devices (in our case, ONU or Access Node).
3. New data structs and property fields are defined and introduced into OpenFlow protocol to indicate queue-related information. Through these extensions, the controller would have the capability to get queue information of each port, e.g. queue number, queue scheduler, queue priority, queue affiliation, guaranteed/ceiling rate, etc. Then controller could make queue decision for each flow.

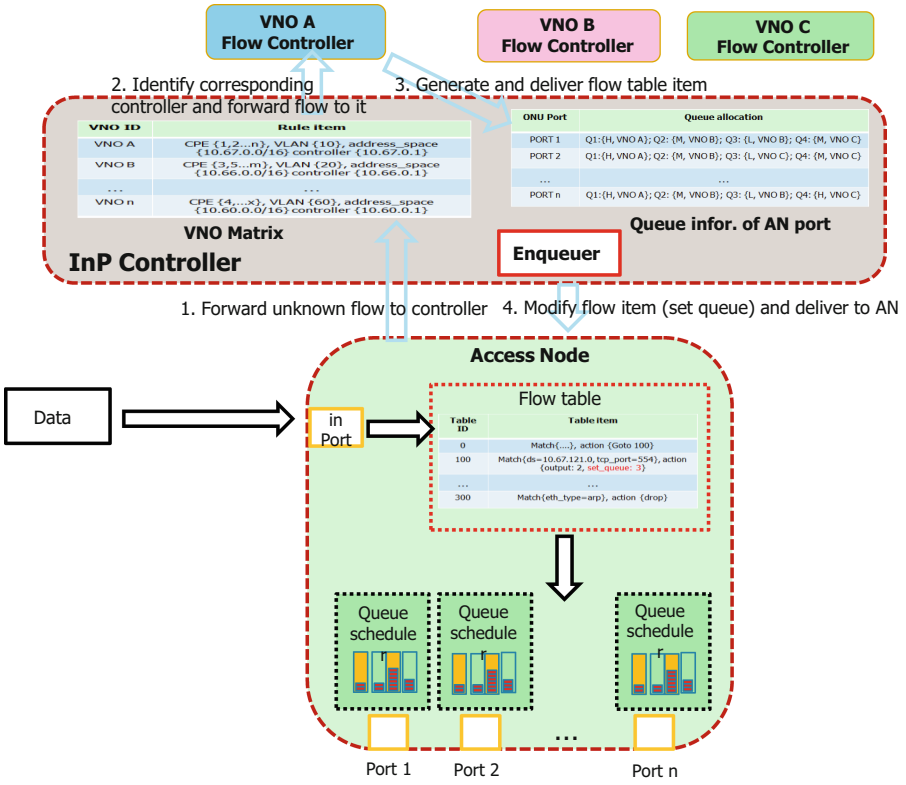

Fig. 5. Queue scheduling for multi-tenants

5 Solution Design and Implementation

This section presents the design and implementation of our approach, introducing how to slice fixed access network based on OpenFlow-enabled queue scheduling.

5.1 Solution Design and Implementation

As mentioned before, Access Node or ONU maintains several queues for different tenants/VNOs. Dedicated queue (or multi-queues) is reserved for specific tenant/VNO. According to the flow table item (with configured queue field) delivered by InP controller, Access Node or ONU could identify different traffics and set them to proper queue. It means that a slice of Access Node or ONU is created for each VNO, which could work as a fully functional Access Node or ONU with complete physical resources (port, link) and logical resources (VLAN space and other label resources). It provides fully functional access network slices for different VNOs while guaranteeing strong isolation between them.

```
enum ofp_queue_desc_prop_type {
        OFPQDPT_MIN_RATE        =0;                 /* Minimum data rate guaranteed. */
        OFPQDPT_MAX_RATE        =1;                 /* Maximum data rate. */
        OFPQDPT_PRIORITY        =2;                 /* Queue priority in SP algorithms. */
        OFPQDPT_WEIGHT          =3;                 /* Weight value in WRR algorithms. */
        OFPQDPT_AFFILIATION     =4;                 /* Queue affiliation. */
        OFPQDPT_EXPERIMENTER    =0xFFFF;            /* Experimenter defined property. */
};
/* Common header for all queue properties. */
struct ofp_queue_desc_prop_header {
        uint16_t        type;               /* One of OFPQDPT_*. */
        uint16_t        length;             /* Length in bytes of this property. */
}
/* queue description.*/
struct ofp_queue_desc {
        uint32_t        port_no;                        /* Port this queue is attached to. */
        uint32_t        queue_id;                       /* id for the specific queue. */
        uint16_t        len;                            /* Length in bytes of this queue desc. */
        uint8_t         pad[6];                         /* 64-bit alignment. */
        struct ofp_queue_desc_prop_header   properties [0];     /*List of properties. */
}
```

Fig. 6. Queue property type and queue description struct-1

InP controller maintains the VNO matrix, which is composed of a series of rules. Each rule item describes the properties of one VNO, such as VNO type, VNO priority, CPE identities, flow space, VNO controller IP, etc. When an unknown flow arrives at the Access Node, it would be forwarded to InP controller. According to the VNO matrix, InP controller identifies the flow and forwards it to the corresponding VNO controller. Then, the VNO controller generates and delivers flow table item to InP controller. Since the VNO controller could obtain the queue information of underlying devices (AN or ONU) through the extended OpenFlow protocol, it could decide which queue is used for forwarding the flow. The "Enqueuer" module injects proper queue value in the action field of flow table item and delivers it to the Access Node or ONU. The whole procedure is shown in Fig. 5.

In order to communicate the queue information between the controller and underlying devices, we introduce a new enum value (as shown in Fig. 6) to describe different types of queue properties, which could be included in queue description struct (ofp_queue_desc) defined in OpenFlow protocol. As shown in Fig. 7, three new property objects (ofp_queue_desc_prop_priority, ofp_queue_desc_prop_weight, ofp_queue_desc_prop_affiliation) are introduced to describe the priority, weight and affiliation property, respectively. Note that all these properties are not mandatory and they are defined and used only when applicable. For example, only when weighted round-robin algorithms are employed for queue scheduling, the weight property would be used to describe the queue weights. Only when the queue is

```
/* queue priority description.*/
struct ofp_queue_desc_prop_priority {
        uint16_t        type;                   /*  OFPQDPT_PRIORITY. */
        uint16_t        length;                 /*  Length is 8. */
        uint16_t        priority;               /*  priority value. */
        uint8_t         pad[2];                 /* 64-bit alignment */
}

/* queue weight description.*/
struct ofp_queue_desc_prop_weight {
        uint16_t        type;                   /*  OFPQDPT_WEIGHT. */
        uint16_t        length;                 /*  Length is 8. */
        uint16_t        weight;                 /*  weight value. */
        uint8_t         pad[2];                 /* 64-bit alignment */
}

/* queue affiliation description.*/
struct ofp_queue_desc_prop_affiliation{
        uint16_t        type;                   /*  OFPQDPT_AFFILIATION. */
        uint16_t        length;                 /*  Length is 8. */
        uint16_t        affiliation;            /* Tenant ID to indicate queue affiliation.  Set 0xFF if not exclusive */
        uint8_t         pad[2];                 /* 64-bit alignment */
}
```

Fig. 7. Queue property type and queue description struct-2

dedicated to one specific VNO, the affiliation property would be set to the corresponding VNO ID.

A series of queue scheduler structs are defined to describe different scheduling algorithms. As shown in Fig. 8 and Fig. 9, specific parameters are defined for different schedulers. Different queue schedulers are defined to describe different queue scheduler algorithms. Each queue scheduler struct could include multiple ofp_queue_desc structs, which describe queues under the scheduling of this queue scheduler.

As show in Fig. 10, specific queue scheduler struct would be added to ofp_port_desc_prop_* struct if the corresponding scheduling algorithm is employed on this port.

Through these extensions to OpenFlow protocol, the queue status and scheduler information could be communicated between the controller and underlying devices. Based on this information, the "Enqueuer" module can accurately make decisions and designate proper queue value in the flow action field.

Overall, this paper proposed an OpenFlow-enabled queue scheduling methodology to support multi-tenants sharing the same fixed access network. It provides fully functional access network slices for different VNOs while guaranteeing strong isolation between them. Besides, the OpenFlow protocol is extended to enable communicating the queue-relevant information between the controller and underlying devices.

```
/* HTB struct. */
struct ofp_queue_scheduler_HTB {
        uint16_t                type;                   /* OFPQST_HTB */
        uint16_t                length;                 /* Length in bytes of this scheduler. */
        uint16_t                leaf_number;            /* total leaves number. */
        uint32_t                guaranteed_rate;        /* total guaranteed rate. */
        uint32_t                ceil_rate;              /* total ceiling rate. */
        struct ofp_queue_desc   queue[leaf_number];     /* property description of each queue. */
};

/* WRR struct. */
struct ofp_queue_scheduler_WRR {
        uint16_t                type;                   /* OFPQST_WRR */
        uint16_t                length;                 /* Length in bytes of this scheduler. */
        uint16_t                queue_number;           /* total queue number. */
        uint32_t                max_weight ;            /* maximum weight value. */
        uint32_t                mini_weight;            /* minimal granularity of weight. */
    struct ofp_queue_desc       queue[queue_number];    /* property description of each queue. */
};
```

Fig. 8. New structs of queue scheduler-1

```
/* SP struct. */
struct ofp_queue_scheduler_SP {
        uint16_t                type;                   /* OFPQST_SP */
        uint16_t                length;                 /* Length in bytes of this scheduler. */
        uint16_t                queue_number;           /* total queue number. */
        uint32_t                highest_priority;       /* value of the highest priority . */
        uint32_t                lowest_priority;        /* value of the lowest priority. */
    struct ofp_queue_desc       queue[queue_number];    /* property description of each queue. */
};

/* no queue description defined in FIFO struct . */
struct ofp_queue_scheduler_FIFO {
        uint16_t                type;                   /* OFPQST_FIFO*/
        uint16_t                length;                 /* Length in bytes of this scheduler. */
};

/* queue scheduler types */
enum ofp_queue_scheduler_type {
    OFPQST_HTB          =0;                 /* HTB scheduler. */
    OFPQST_WRR          =1;                 /* WRR scheduler. */
    OFPQST_SP           =2;                 /* SP scheduler. */
    OFPQST_FIFO         =3;                 /* FIFO scheduler. */
    OFPQST_EXPERIMENTER =0xFFFF;            /* Experimenter scheduler. */
};
```

Fig. 9. New structs of queue scheduler-2

```
/* Ethernet port description property. */
  struct ofp_port_desc_prop_ethernet {
      uint16_t     type;                          /* OFPPDPT_ETHERNET. */
      uint16_t     length;                        /* Length in bytes of this property. */
      uint8_t      pad[4];                         /* Align to 64 bits. */
  /* Bitmaps of OFPPF_* that describe features. All bits zeroed if
    unsupported or unavailable. */
      uint32_t     curr;                                  /* Current features. */
      uint32_t     advertised;                            /* Features being advertised by the port. */
      uint32_t     supported;                             /* Features supported by the port. */
      uint32_t     peer;                                  /* Features advertised by peer. */
      uint32_t     curr_speed;                           /* Current port bitrate in kbps. */
      uint32_t     max_speed;                             /* Max port bitrate in kbps */
      uint16_t     max_queue ;                            /* Max amount of queue; zero if unsupported */
      struct  ofp_queue_scheduler_header   scheduler;    /* dedicated scheduler property*/
};
```

Fig. 10. Extension to port property description struct

6 Conclusion

The rise of 5g technology greatly promotes the development of communication network, and a large number of new business scenarios emerge, such as network slicing, and so on. In order to support multi-tenants sharing the same fixed access network, especially the access node and ONU, in this paper we propose a novel queue scheduling mechanism for multi-tenants by extending OpenFlow protocol and introduce a queue scheduling methodology for traffic isolation and QoS guarantee. Besides, a new "Enqueuer" module is introduced for the SDN controller to perform centralized queue management and guarantee strong isolation between different VNOs.

References

1. Openflow swtich specifications. https://www.opennetworking.org/sdn-resources/onf-specifications. OPEN NETWORKING FOUNDATION
2. McKeown, N., et al.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM comput. Commun. Rev. **38**(2), 69–74 (2008)
3. Open networking foundation. https://www.opennetworking.org
4. Kerpez, K., Cioffi, J., Ginis, G., Goldburg, M., Galli, S., Silverman, P.: Software-defined access networks. IEEE Commun. Mag. **52**(9), 152–159 (2014)
5. Hu, F., Hao, Q., Bao, K.: A survey on software-defined network and OpenFlow: from concept to implementation. IEEE Commun. Surv. Tutor. **16**(4), 2181–2206 (2014)
6. Sherwood, R., et al.: FlowVisor: a network virtualization layer. OpenFlow Switch Consortium, Technical report, vol. 1, p. 132 (2009)
7. Oktian, Y.E., Lee, S., Lee, H., Lam, J.: Distributed SDN controller system: a survey on design choice. Comput. Netw. **121**, 100–111 (2017)

8. Forouzan, B.A.: TCP/IP Protocol Suite. McGraw-Hill Higher Education (2002)
9. Li, X., Bao, C., Baker, F.: IP/ICMP translation algorithm. Internet Engineering Task Force, RFC, vol. 6145 (2011)
10. Kim, H., Feamster, N.: Improving network management with software defined networking. IEEE Commun. Mag. **51**(2), 114–119 (2013)
11. Alvarez, P., Slyne, F., Blumm, C., Marquezbarja, J., Dasilva, L., Ruffini, M.: Experimental demonstration of SDN-controlled variable-rate fronthaul for converged LTE-over-PON. In: Optical Fiber Communication Conference (2018)
12. Kosmetos, E., Matrakidis, C., Stevdas, A., Orfanoudakis, T.: An SDN architecture for PON networks enabling unified management using abstractions. In: 2018 European Conference on Optical Communication (ECOC), pp. 1–3 (2018)
13. Parol, P., Pawlowski, M.: Towards networks of the future: SDN paradigm introduction to PON networking for business applications. In: 2013 Federated Conference on Computer Science and Information Systems, pp. 829–836 (2013)
14. Hwang, I.-S., Tesi, C., Pakpahan, A.F., Ab-Rahman, M.S., Liem, A.T., Rianto, A.: Software-defined time-shifted IPTV architecture for locality-awareness TWDM-PON. Optik **207**, 164179 (2020)
15. Schneir, J.R., Xiong, Y.: Cost analysis of network sharing in FTTH/PONs. IEEE Commun. Mag. **52**(8), 126–134 (2014)
16. Uzawa, H., Honda, K., Nakamura, H., Hirano, Y., Terada, J.: Dynamic bandwidth allocation scheme for network-slicing-based TDM-PON toward the beyond-5G era. J. Opt. Commun. Netw. **12**(2), A135-143 (2019)
17. Dong, L., Gu, R., Guo, Q., Ji, Y.: Demonstration of multi-vendor multi-standard PON networks for network slicing in 5G-oriented mobile network. In: Asia Communications and Photonics Conference (2017)
18. Gu, R., Zhang, S., Ji, Y., Yan, Z.: Network slicing and efficient ONU migration for reliable communications in converged vehicular and fixed access network. Veh. Commun. S2214209617301584 (2018)
19. Zhang, L., et al.: Service-aware network slicing supporting delay-sensitive services for 5G fronthaul. In: 2018 Proceedings Paper (2018)