Network slicing

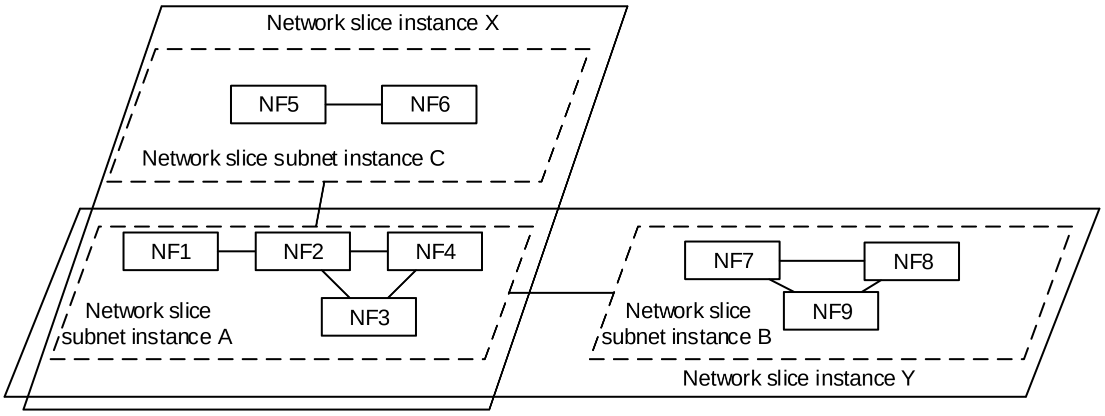
Oberseminar
29. Januar 2019

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Elektrotechnik und Informationstechnik
Institut für Nachrichtentechnik
Deutsche Telekom Chair of Communication Networks
Prof. Dr.-Ing. Dr. h.c. Frank H.P. Fitzek

Task for PBL & Oberseminar WS 2018/19

Topic: Network Slicing


Network slice instance X
NF5
NF6
Network slice subnet instance C
NF1
NF2
NF4
NF3
Network slice subnet instance A
NF7
NF8
NF9
Network slice subnet instance B
Network slice instance Y

Task description

Network Slicing is an approach to enable multiple requirements on the same physical network infrastructure. In order to satisfy e.g. low-latency requirements, isolation and guarantees, the physical network is abstracted and a virtualized environment is created. In this realm, attaching resources to different logical networks becomes feasible.
The OpenFlow protocol has emerged as an easy-to-use, accessible and widespread tool for programming network hardware. It will be used to implement network slicing and monitoring efforts.

Sources:
https://www.itu.int/en/membership/Documents/missions/GVA-mission-briefing-5G-28Sept2016.pdf
https://osrg.github.io/ryu-book/en/html/index.html

Target:
- Setup of testbed with guaranteed properties
- Implement triggered Network Slicing Functionality in Controller
- Implement on-demand Network Slicing Functionality in Controller
- Show and measure performance of various network slices under load

Required Skills: Basic knowledge of Network Softwarisation; Python (highly recommended) programming experience

Language: English

Supervisor	Dipl.-Ing. Vincent Latzko
Responsible teacher	Prof. Dr.-Ing. Dr. h.c. Frank H.P. Fitzek

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe, die aus fremden Quellen direkt oder indirekt übernommenen gedenk sind als solche kenntlich gemacht. Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützungsleistungen von folgenden Personen erhalten:

Weitere Personen waren an der geistigen Herstellung der vorliegenden Arbeit nicht beteiligt. Mir ist bekannt, dass die Nichteinhaltung dieser Erklärung zum nachträglichen Entzug des Diplomabschlusses (Masterabschlusses) führen kann.

Dresden 22, 07, 2018 Unterschrift:

Inhaltsverzeichnis

1 Introduction

1.1 Network slicing

In the 5G era, mobile networks need to serve devices of all types and needs. More application scenarios include mobile broadband, large-scale Internet of Things, and mission-critical IoT. They all need different types of networks, with different requirements in terms of mobility, billing, security, policy control, latency, and reliability. For example, a large-scale IoT service connects fixed sensors to measure temperature, humidity, rainfall, etc.

The traditional mobile communication network operates in a single mode, which means all services run on one network, and relies on IP protocols such as DiffServ to determine the priority of different services, but these protocols are fragmented and cannot implement end-to-end service orchestration. This apparently dose not enable the range of services required in the 5G era.

Fortunately the network slice has been entered into the 5G vision of next generation mobile network, which optimizes network resource allocation, achieves maximum cost efficiency, and meets the needs of multiple 5G new services. Through network slicing technology multiple logical networks could be segmented on a separate physical network. For example, specialized video network slicing, IoT network slicing, or critical communication network slicing, and the like. Certainly, multiple similar services can also be placed on one network slice. Each network slice is logically a self-sufficient network with a separate network slice for each service. For each network slice, dedicated resources such as virtual servers, network bandwidth, and quality of service are fully guaranteed. Since the slices are isolated from each other, a slice error or failure does not affect the communication of other slices.

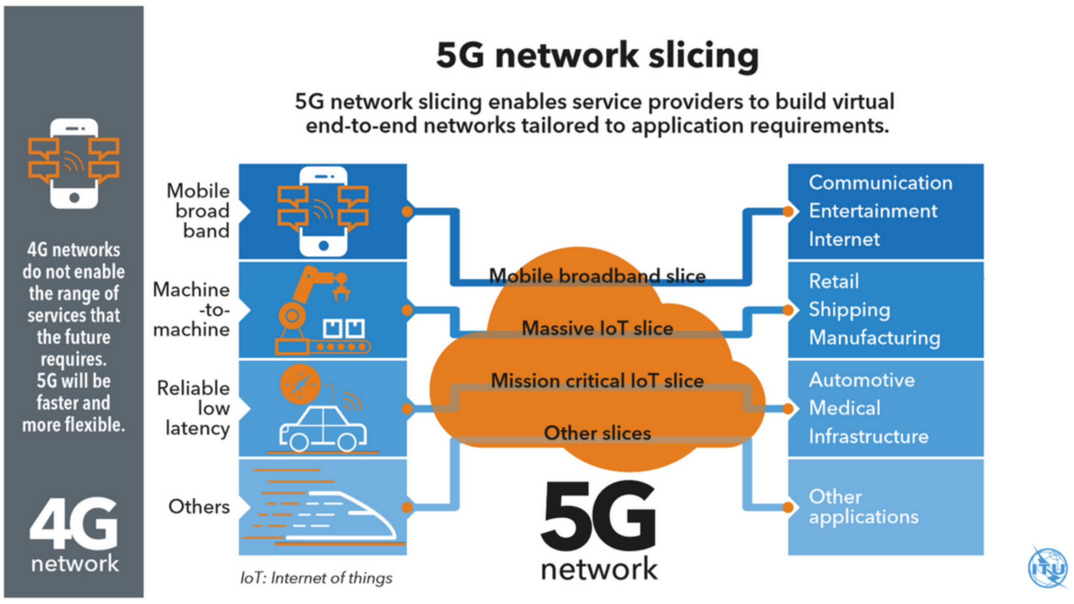

5G network slicing
5G network slicing enables service providers to build virtual end-to-end networks tailored to application requirements.
4G networks do not enable the range of services that the future requires. 5G will be faster and more flexible.
4G network
Mobile broad band
Machine -to- machine
Reliable low latency
Others
Mobile broadband slice
Massive IoT slice
Mission critical IoT slice
Other slices
Communication Entertainment Internet
Retail Shipping Manufacturing
Automotive Medical Infrastructure
Other applications
5G network
IoT: Internet of things
ITU

Abbildung 1 – 5G

1.2 Network slicing using NFV and SDN

NFV and SDN are two similar but distinct technologies that lead the digital transformation of the telecommunications industry's network infrastructure, and they will be basically used in 5G network slicing.

NFV proactively provides network services, which typically run on proprietary hardware that owns virtual machines, which could be regarded as operating systems that emulate dedicated hardware. With NFV, virtual machines provide networking capabilities such as routing, load balancing, and firewalls. Resources are no longer confined to the data center, but are spread across the network to accelerate the productivity of internal operations.

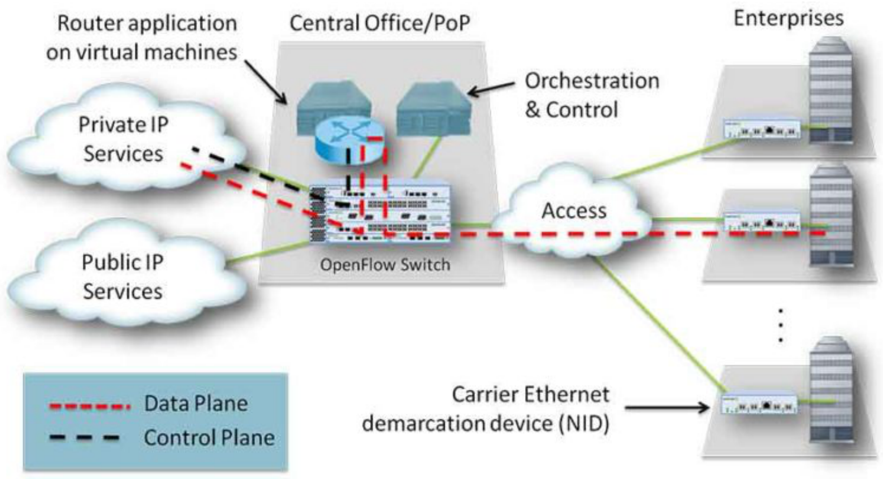


Abbildung 2 – NFV

Traditional transport networks include specialized routers and switches for data forwarding and network control. Instead of that SDN centralizes network control functions through a separate software-based SDN controller, while routers and switches are only responsible for forwarding, thus reducing the cost of forwarding network elements.The SDN controller monitors most networks and easily identifies the optimal message route, which is especially useful in situations where the network is congested or partially paralyzed. The routing decision capability of the SDN controller is also much higher than that of the routers and switches in the traditional network, because the routing decision of the latter is based on a very limited part of the network.

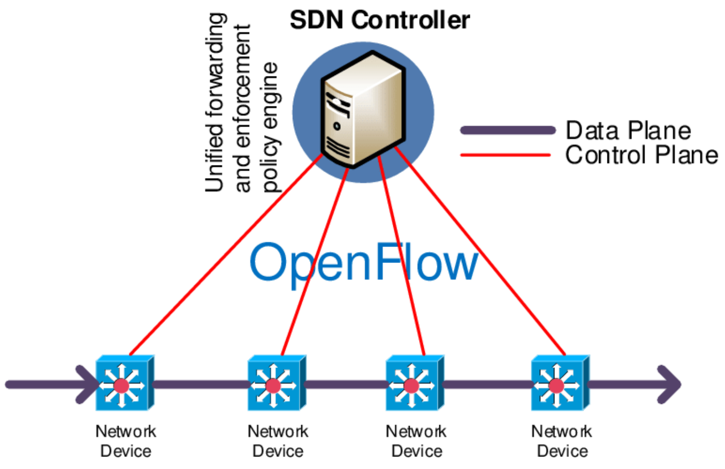


Abbildung 3 – openflow

Both SDN and NFV can provide the functions required for network slicing. The main idea of 5G network slicing is to create and divide different services on the network, so that operators can provide the best support for these services. SDN and NFV serve as the basis for network slicing, allowing the use of force and virtual resources to provide certain services

1.3 Network slicing in switch based on QoS

QoS(Quality of Service) is a set of technologies that work on the network to ensure its ability to reliably run high-priority applications and traffic with limited network capacity. QoS technology accomplishes this by providing differentiated processing and capacity allocation for specific flows in network traffic. This allows the network administrator to allocate the order in which the packets are processed and the amount of bandwidth provided for that application or traffic. The QoS-related measurements are bandwidth (throughput), delay, jitter (delay change), and error rate. This makes QoS especially important for high-bandwidth, real-time traffic such as voice over IP (VoIP), video conferencing, and video on demand with high sensitivity to delay and jitter.

In this project we build a single topo, with one controller one switch and the network slicing based on Qos is implemented in the switch.

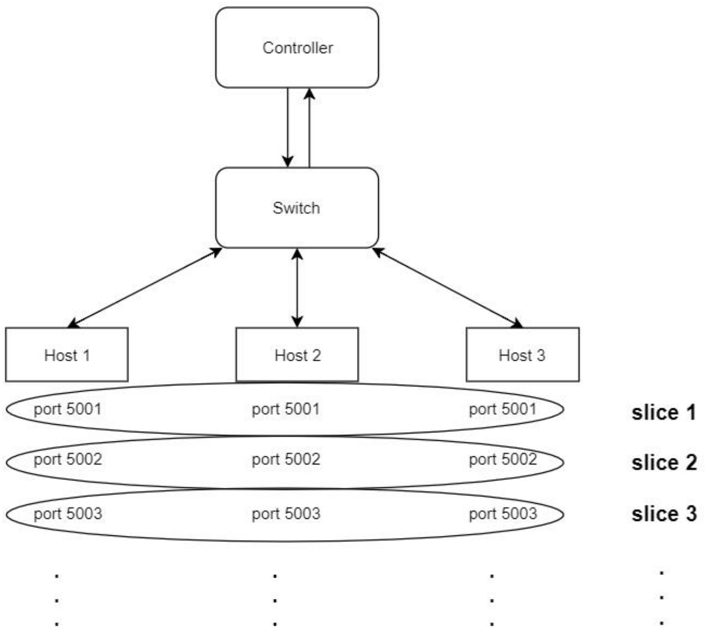


Abbildung 4 – slice

1.4 Mininet, Ryu controller and OpenvSwitch

In order to simulate a real network environment, we mainly use the following three software combinations, namely Mininet, Ryu controller and OpenvSwitch.

The first tool is Mininet. Mininet is a network emulator or perhaps more precisely a network emulation orchestration system. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. It can also be easily understood as a process-based virtualization simulation platform in the SDN network system, and more importantly, it supports OpenFlow and other protocols. Mininet also has its own switch, host, controller, at the same time, OpenvSwitch and a variety of controllers (NOX POX RYU Floodlight OpenDaylight, etc.) could be also installed on the Mininet. Its many advantages provide us with great convenience in simulating real networks. For example, we build a single topology using Mininet with one controller,one switch and two hosts and our next step is based on such a topo.

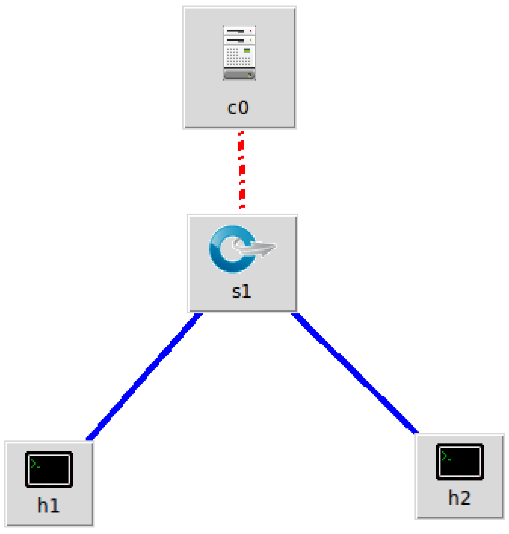


Abbildung 5 – mininet

The second one is Openvswitch(OVS). The Open vSwitch version in use at the nodes is 2.5.2, which is one of the most popular virtual switches running on a virtualization platform. Open vSwitch is developed as an open source software switch that would be able to provide adequate performance while remaining generalized and flexible. On a virtualization platform, OVS can provide Layer 2 switching for dynamically changing endpoints, providing excellent control over access policies, network isolation, traffic monitoring, and more in virtual networks.

The OpenvSwitch userspace daemon is responsible for communicating with an SDN controller using the OpenFlow protocol. It also maintains the OpenFlow tables and is responsible for matching packets received by the kernel datapath against these tables. It then translates the matched flows entries into the kernel module then caches these instructions for later use[2].OpenvSwitch is configured through the ovsdb-server, a database server containing the configuration state of the Open vSwitch instance. The ovsdb-server can be accessed Through the Open vSwitch Database (OVSDB) protocol [3]. The OVSDB protocol enables programmatic access to the Open vSwitch database, which can be done remotely from an SDN controller.

Thirdly, the controller is the most important part of the SDN network. When developing an SDN application, it needs to be developed based on a certain controller, and most open source controllers are a framework or platform. More personalized settings and applications need to be completed by the developers themselves. For the developer, a custom controller can make the controller more suitable for the development scenario, play the biggest role of the controller, and improve the development efficiency. For our project, we choose version 4.18 of the Ryu SDN framework because of its two main advantages. Ryu supports OpenFlow 1.0 to 1.5. Ryu provides a rich set of components for developers to build SDN applications.

2 Network slicing Implementation

2.1 Problems in rest_qos

The Ryu SDN framework provides various software components for implementing OpenFlow SDN applications in Python. And the modul rest_qos is the core to implement network slicing and QoS. With running this module the Ryu SDN works itself like a Python web application. The controller communicates with the switches using the standard OpenFlow protocol, and the Open vSwitch Database (OVSDB) management protocol for configuring virtual switches. It also provides lots of API which are implemented in the form of REST API. They provides not only the means to limit bandwidth available to each slice on each virtual link through QoS rules, but also allows the tenant networks to have the possibility of being partitioned and managed, that means slice in the network could be added or removed from tenant network.

Through using these rest commands, the user can add suitable rules on each port of the switch to control the bandwidth and ensure the purpose of QoS. However, after many experiments, we found that the API which are provided by this rest qos file still has three main problems.

2.1.1 add a new queue flexibly

The first problem is that the user does not have a suitable command that allows the user to add a new slice based on the original slice flexibly. Let's look at an example that user add just two slice in the in his network.

```
curl -X POST -d {”port_name” : ”s1 − eth1”, ”type” : ”linux − htb”, ”max_rate” :
”1000000”, ”queues” : [{”max_rate” : ”500000”}, {”min_rate” : ”800000”}]}
http://localhost:8080/qos/queue/0000000000000001
```

```
curl -X POST -d {”match” : {”nw_dst” : ”10.0.0.1”, ”nw_proto” : ”UDP”, ”tp_dst” :
”5002”}, ”actions” : {”queue” : ”1”}}
http://localhost:8080/qos/rules/0000000000000001
```

The first rest command means that user set a list of queues which containing two queues on port eth1 of switch s1. The second one means that, a rule is set with a qos id. For the udp packet whose destination IP address is 10.0.0.1 and the port is 5002, the bandwidth requirement must be the same as the first item in the queue. It is no less than 800k. Through these two rest command user can add two slices to the port s1_eth1. The first slice applies to all other ports. The maximum bandwith is 500k. The second slice is only applicable to the port 5002 and its bandwidth can not be less than 800k.

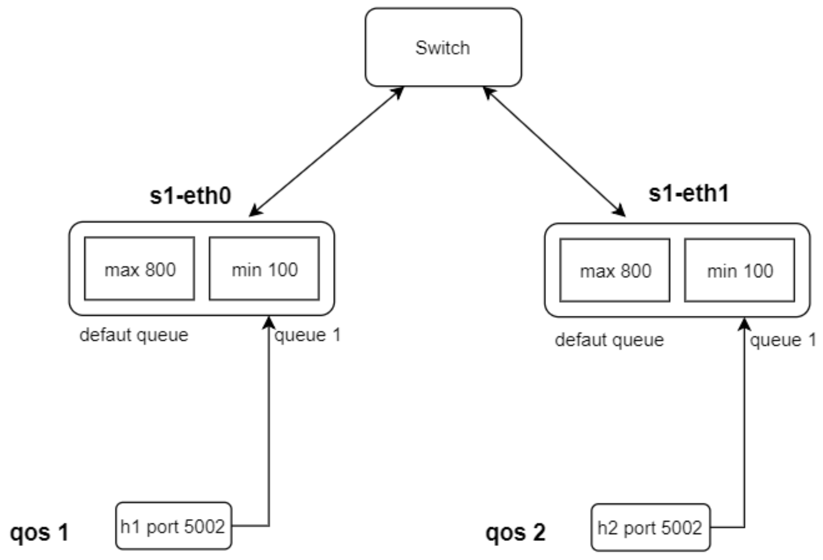


Abbildung 6 – two queue

But when the user needs to add a new slice, the following two operations must be performed. First we have to delete the original queue and qos_id on my eth1 port.

curl -X DELETE http://localhost:8080/qos/queue/0000000000000001

curl -X DELETE {"*qos_id*", "*all*"} http://localhost:8080/qos/rules/0000000000000001

Then add a queue list with more queues and add two qos id at the same time.

curl -X POST -d {"*port_name*" : "*s*1 – *eth*1", "*type*" : "*linux* – *htb*", "*max_rate*" : "1000000", "*queues*" : [{"*max_rate*" : "500000"}, {"*min_rate*" : "800000"}, {"*min_rate*" : "600000"}]} http://localhost:8080/qos/queue/0000000000000001

curl -X POST -d {"*match*" : {"*nw_dst*" : "10.0.0.1", "*nw_proto*" : "*UDP*", "*tp_dst*" : "5002"}, "*actions*" : {"*queue*" : "1"}} http://localhost:8080/qos/rules/0000000000000001

curl -X POST -d {"*match*" : {"*nw_dst*" : "10.0.0.1", "*nw_proto*" : "*UDP*", "*tp_dst*" : "5003"}, *äctions*" : {"*queue*" : "1"}} http://localhost:8080/qos/rules/0000000000000001

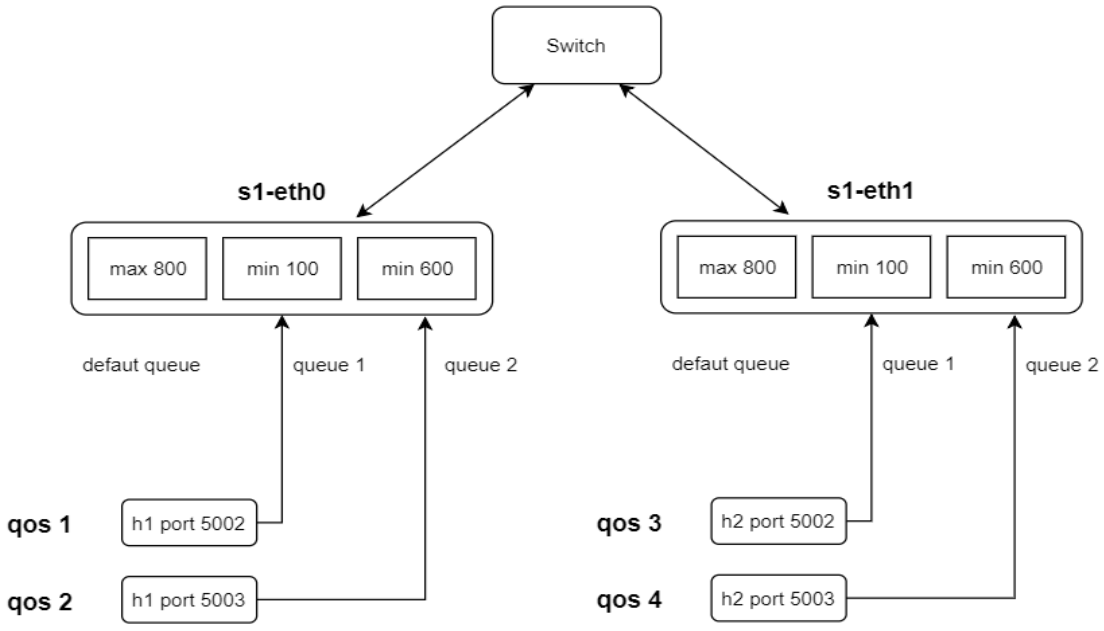


Abbildung 7 – three queue

If the user just add one slice, the operations could barely be accepted, but if the user wants to add 10 slices or 100 slices, then we must write 100 different queues in the queues list, and also add 99 qos ids. Obviously, this kind of operation is very troublesome. The most fundamental idea of SDN is to make our network be more flexible. When the amount of slices required by users is very large, the flexibility of this rest api command is not far enough.

Solution: We can design a new rest command and have some built-in methods that allow the user to add just one queue every time and add the qos id at the same time when they want to add a new slice.

2.1.2 optimize all slices

Secondly, sometimes when we urgently need to use one or more slices, the controller can not design a special sclice to make the new slice work properly, and provide a optimize method for other sclices.
For example, if there are such two slices in our network now.

$"queues" : [\{"max\_rate" : "500000"\}, \{"min\_rate" : "800000"\}]$

And now we need to use a new device urgently, and make sure this device works properly at same time, which means that this device has the highest priority. If the requirement of this equipment is at least 900k,

$\{"min\_rate" : "900000"\}$

In other words, all the slice requirements in our network are now like this.

$"queues" : [\{"max\_rate" : "500000"\}, \{"min\_rate" : "800000"\}, \{"min\_rate" : "900000"\}]$

By observation, we found that the second slice requires at least 800k of bandwidth, while the third slice requires at least 900k of bandwidth, but our total bandwidth is only 1M. The sum of these two slices is already greater than 1M. Obviously in this way, the second slice and the third slice will be treated fairly. So we can not guarantee that our newly added devices will work properly.

Solution: When the user want to add a new slice in the netowrk, the user must define the priority of this slice at the same time. After adding a new slice, the controller will re-prioritize all the slices and re-design them if necessary. All slices are designed to provide better service for higher priority slices and abandon the guarantee of the original slices from lower priority.

2.1.3 delete slice

When we want to delete a new slice, although we can use rest command which has been designed in qos_rest to realize it. But manual operations are always complicated and easy to be forgetten by the user. When the user no longer uses this slice, and does not delete the slice immediately, this will cause a waste of resources, and this slice may also have more or less impact on other slices.

Solution: So we need the controller to monitor the traffic of the all slices at all times. Once the traffic of a slice not changed for a certain period of time, the controller can assume that the user no longer uses the slice and then delete the slice automatically

2.2 Solutions

In order to solve the above three main problems, we have designed a brand new rest command.

Put rules
    Method PUT
    url qos/rules/switch
    Data
        Mod: 0 or 1
        queue_priority: any natural number
    match:
        tp_dst:[0-65535]
    actions:
        max_rate:[$Bandwidth(bps)$]
        min_rate:[$Bandwidth(bps)$]
    Remark: mod decide different way to delete a sclice. 0: automatically 1: manually

In order to more clearly illustrate the execution process and advantages of this rest command, the following is its program flow chart.

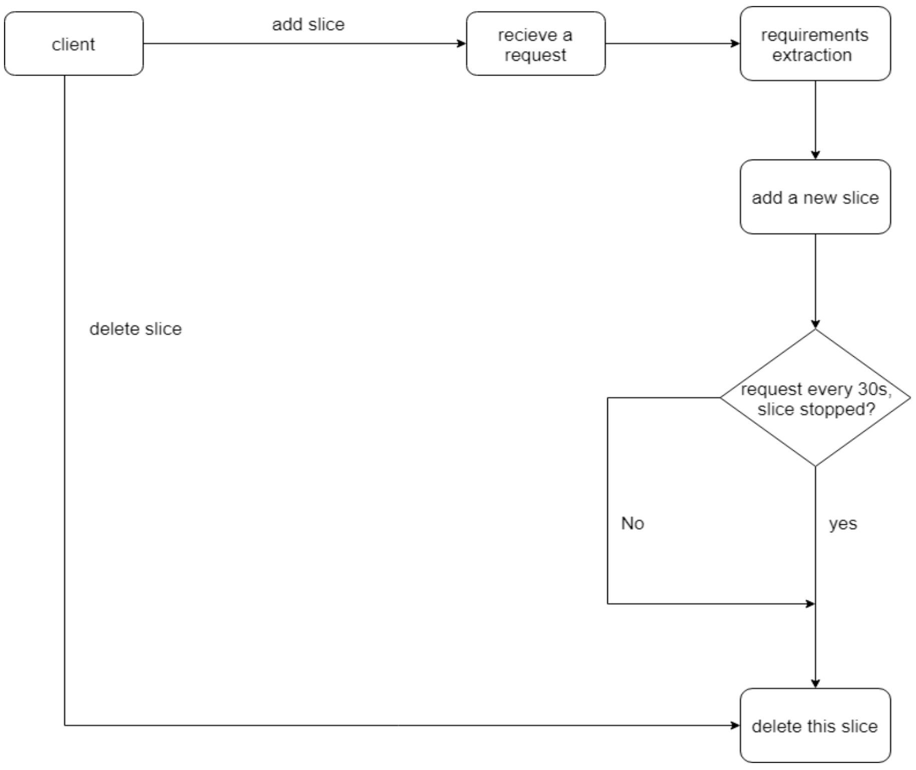


Abbildung 8 – generally flow chart

2.2.1 add a new sclice

To the first part adding a new slice, We greatly simplify the user's operation. From a user's point of view, when he wants to use a new device or software in his network, the user mainly needs to consider two issues.
1. Which network port that I want to use for this device.
2. In order to ensure that the device can be used, does the device have a clear minimum bandwidth requirement?
This is the two most important parameters in our new rest command, which are following the match and action. Let's expand the simple flow chart just below.

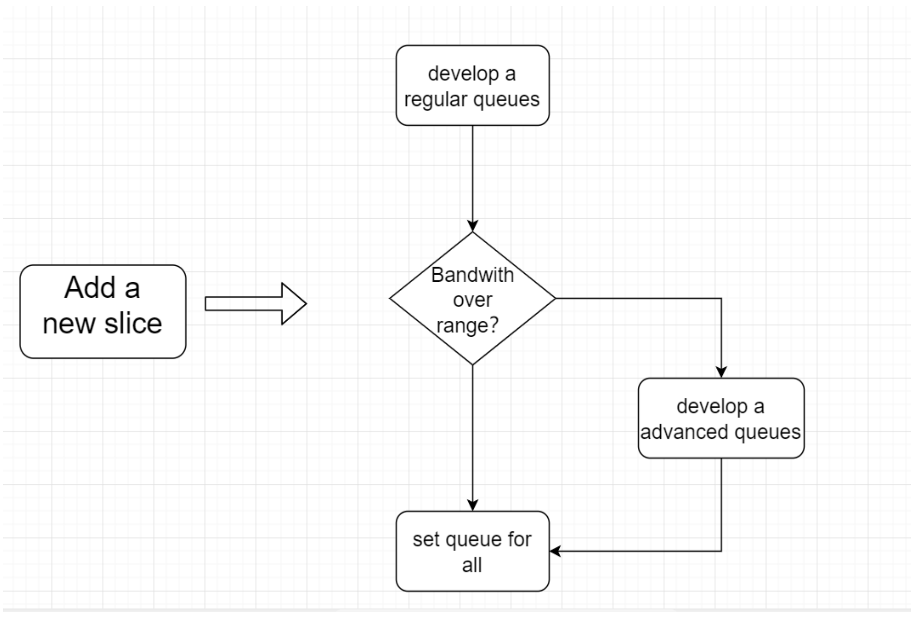
a new slice.png 

Abbildung 9 – add a new slice

The main task of Develop a regular queues is to solve the first problem. It can extract the original queues of the port and add the new queue. This will achieve the goal to add the queue automatically, and the user only needs to input the value of the requirements of the port which the user needs.

Old: "queues":[$\{"max\_rate" : "500000"\}, \{"min\_rate" : "800000"\}$]

What we add: $\{"min\_rate" : "800000"\}$

New : "queues": [$\{"max\_rate" : "500000"\}, \{"min\_rate" : "800000"\}, \{"min\_rate" : "800000"\}$]

Another task of this method is, summarize all the qosids that need to be added and add them to a list.
For example

Data1 = $\{"mod" : "1", "queue\_priority" : "1", "match" : \{"nw\_dst" : "10.0.0.1", "nw\_proto" : "UDP", "tp\_dst" : "5002"\}, "actions" : \{"queue" : "1"\}\}$

Data2 =$\{"mod" : "1", "queue\_priority" : "1", "match" : \{"nw\_dst" : "10.0.0.1", "nw\_proto" : "UDP", "tp\_dst" : "5003"\}, "actions" : \{"queue" : "2"\}\}$

qos_list =$\{Data\_1, Data\_2\}$
queues=$\{queue, queues1, queues2\}$

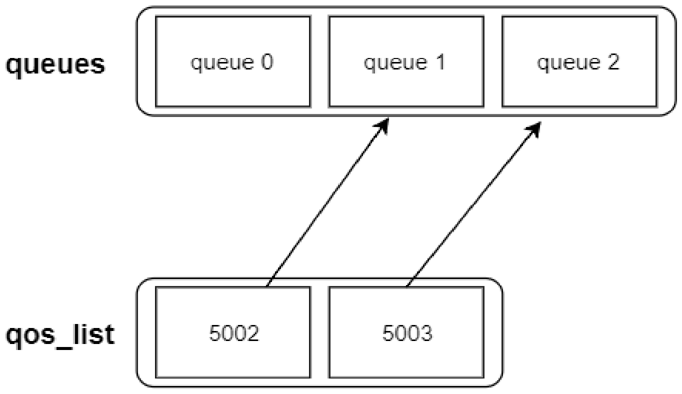


Abbildung 10 – one-to-one correspondence

And make sure the order of the qos id inside this qos list and the queue inside the queues be one-to-one correspondence. It is very convenient for us to delete a queue in our next step.

In the case that the controller don't need to redesign the slice, we can directly call the method (set queue for all) to add each qos rule on the port of switch by using the list of qos_id. The integration of set queue and set qos has been achieved, which greatly simplifies the user's operation steps and achieves the purpose of solving the first problem.

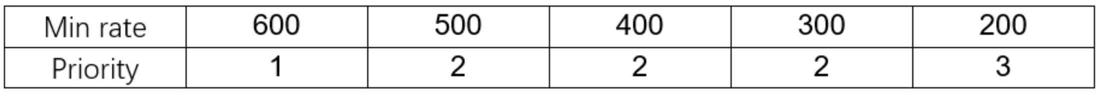
2.2.2 optimize all slices

In order to solve the second problem, we first need to judge the bandwidth requirements of all slices. Whether their sum has exceeded the total bandwidth. Once the total bandwidth is exceeded, the controller need to adjust the bandwidth of all slices based on the priority. The core method of adjusting each slice is to set the requirements and speed of each queue in each list of queues. So we designed a simple algorithm for this.

First, we developed three design principles for our algorithm:
1.slices from higher priority will be first considered.
2.ensures more slices work properly
3.treats the slices from same priority sequally

Our algorithm can be clearly explained by the following example.

$"queues" : [\{"max\_rate" : "800000"\}, \{"min\_rate" : "500000"\}, \{"min\_rate" : "400000"\}, \{"min\_rate" : "300000"\}, \{"min\_rate" : "200000"\}, \{"min\_rate" : "100000"\}]$



Min rate	600	500	400	300	200
Priority	1	2	2	2	3

Abbildung 11 – pri list

First, we found that there are three kinds of priority 1, 2, and 3.

When it traverse at the first time, it will collect the priority of 3, which is 200k here, then it will be compared with the total bandwidth of 1M. When the total bandwidth is enough, it will be added in a suitable position.

queues =[queue1, queue2, queue3, queue4, queue5 = min_rate: 200k]

It will collect the requirements from priority 2 when it traverse the second time. We find that there are 4 requirements from priority 2. According to the second rule of the algorithm principle we formulated at the beginning, We choose the reqirements from small to large, because it ensures that more slices can be used at the same time. So the fourth queue and fifth queue could be added.

queues =[queue1, queue2, queue3, queue4= min_rate: 400k, queue5= min_rate: 300k, queue6 = min_rate: 200k]

But for the other two slices, min_rate600k or 700k, because our total bandwidth left is only 100k, we can't guarantee their speed for them. According to the third rule of our design principle, treat them as equal as possible. We will divide the remaining 100k into these two slices, that is

queues =[queue1 , queue2= min_rate: 50k, queue3= min_rate: 50k, queue4= min_rate: 400k, queue5 = min_rate: 300k, queue6= min_rate: 200k]

In the end, what's left is the bandwith reqiuement from priority 1. Because at this time our total bandwidth has not been left, so the speed of queue1 is 0.

queues =[queue1=min_rate: 0k, queue2= min_rate: 50k, queue3= min_rate: 50k, queue4= min_rate : 400k, queue5= min_rate: 300k, queue6= min_rate: 200k]

Below is our program flow chart.

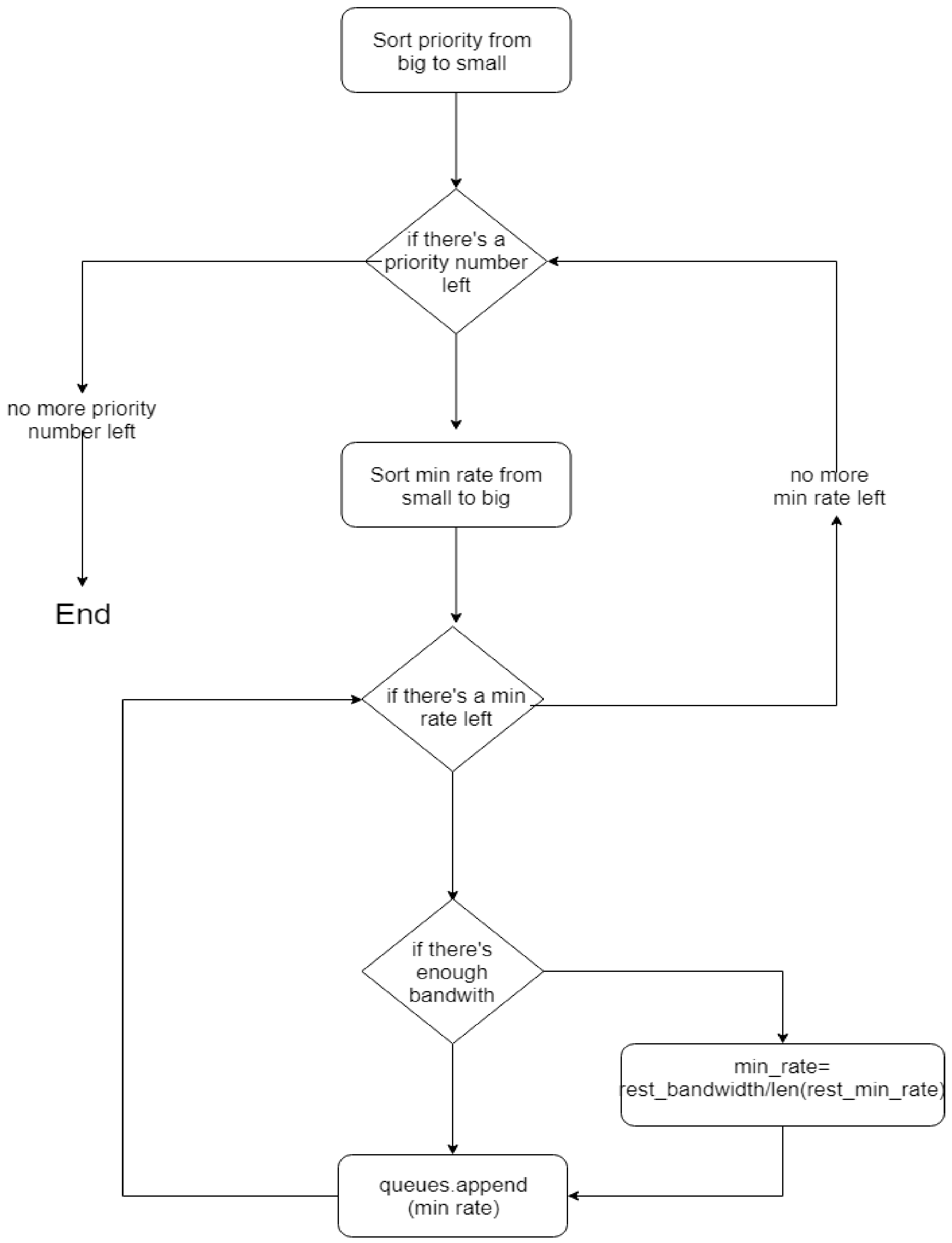

Sort priority from big to small
if there's a priority number left
no more priority number left
End
Sort min rate from small to big
no more min rate left
if there's a min rate left
if there's enough bandwith
min_rate= rest_bandwidth/len(rest_min_rate)
queues.append (min rate)

Abbildung 12 – algorithm flow chart

2.2.3 Deleting slices

In order to increase the bandwidth utilization, and prevent the slice from occupying of the bandwidth resources, which does not transmit and receive data, the newly added slice should be deleted after it is stopped, and then added again at the next time, when the client sends the request.

Since a single queue in the Queue_list cannot be directly deleted in the original file rest_qos.py, it can only be deleted indirectly: Save all current slice-information first, and then delete all slices. Next, extract the slice-information, and remove the part of slice-information, which should be deleted now. Finally, set the Queue and Qos again for each host. That is to say, The process of the deletion is just delete all the slices first, and re-add the slices again, which are still expected to stay behind.

However, when a new slice is being added, the controller may redistribute the bandwidth for each slice depending on the priority of the slices. Therefore, before re-add the slices, the bandwidth requirements of the primitive slices must be restored, and then redistribute the bandwidth for each slice according to the priorities and the total bandwidth. In this way it will ensure that the process of deleting slices can be done orderly.

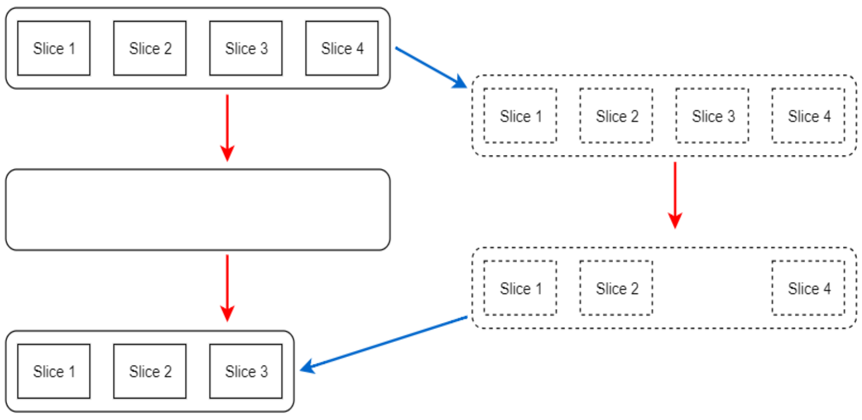


Abbildung 13 – delete principle

In actual situations, some clients need large bandwidth, they use it just for a short time, some clients do not need a large bandwidth, but they want using it for a long time, and the data stream may not be received and transmitted while waiting to receive data. In this case, the slice cannot be deleted after the data was just stopped.

Due to the different requirements, we have adopted a client-defined deletion method.

According to the requirements of the clients, they can decide whether the slice is automatically deleted after it was stopped or the clients manually delete the slice at any time.

In the request text of the client, there is a Mod parameter that marks the client's selection of the deletion mode. The controller will process the slices differently according to different mode requirements. The specific operations of the manual deletion mode and the automatic deletion mode are explained in detail below.

2.2.4 Automatically delete slices

When the client sends a request of adding a new slice, if he wants the automatic deletion mode, the delete mode Mod=0 can be added to the request text. The Mod parameter can also not be set, in this case, the default Mod=0 will be given.

After the controller receives the request of new slice from the client for the first time, a timer will start counting, so that the program can extract the flow table information at regular intervals, to determine whether the slices have not received data from the last time period until now.

If no new data is received, then this slice will be disabled by default. In the flow table information, the Byte_count parameter records the number of bytes received by the host port. If the extracted Byte_count value has not changed in the previous time period, the program will automatically consider that the slice has been stopped and will not be used any more, and its port number will be recorded. All the slices, whose port numbers will be passed to the delete method, and these slices will be deleted, except the two initialized slices, which are always present by default, and the slices, which are defined by the client to be manually deleted.

2.2.5 Manually deleting slices

If the client chooses to manually delete the slice, it needs to mark the parameter Mod=1 in the request message when sending the request. The parameter of the slice will be judged before the automatic deletion mode is deleted. The slices of the manual deletion mode will be removed from the list, that tell the delete method, which of the slices should be deleted. This prevents the slice from being deleted by mistake.

After the client stops using the slice, a delete request must to be send and inform the port number of the slice to be deleted, and delete the slice manually. This slice will persist until the controller receives a request to manually delete this slice. After the receiving controller receives the delete request, the program directly transfers the port number of the slice in the request information to the delete method, and deletes the slice.

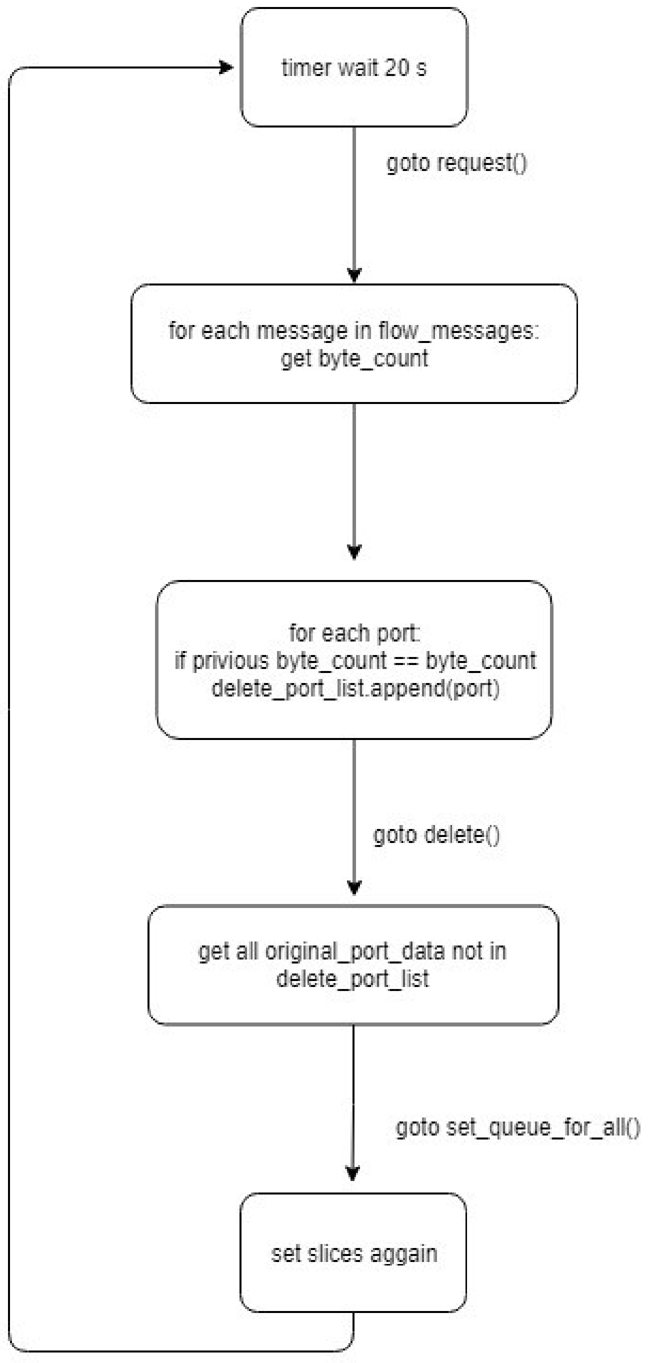

timer wait 20 s
goto request()
for each message in flow_messages:
get byte_count
for each port:
if privious byte_count == byte_count
delete_port_list.append(port)
goto delete()
get all original_port_data not in
delete_port_list
goto set_queue_for_all()
set slices aggain

Abbildung 14 – delete flow chart

3 Test performance

In order to explore the performance of our network slicing scheme, we built a small switch-based network in the mininet, and used the iperf command tool to monitor and send data streams to the target host.

Iperf is a network performance testing tool that measures maximum TCP and UDP bandwidth with multiple parameters and features to record bandwidth, delay jitter and packet loss, maximum group and MTU statistics. Through this information, you can discover network problems, check network quality, and locate network bottlenecks. The UDP data is sent in the tests in this chapter.

The controller can control the switch network of multiple hosts. The added slices are applicable to each host, but for the convenience of testing, only two hosts are connected, one for receiving data and one for transmitting data. The port corresponding to each slice is monitored by the iperf tool, and the monitoring results are automatically recorded so that the data can easily graphed.

We simulated the case of adding a slice, adding multiple slices, and compared the performance of manual deletion and automatic deletion.

3.1 Adding a sclice

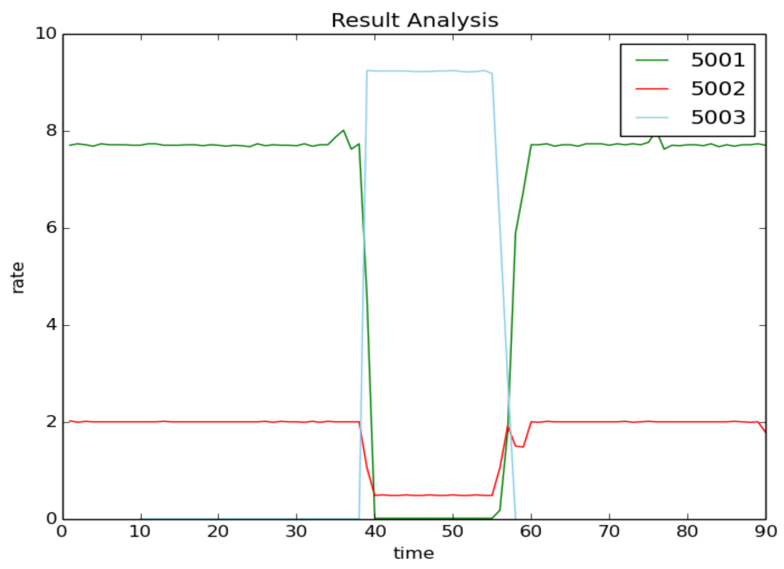


Abbildung 15 – one sclice

As shown in the figure above, a new slice port 5003 is added after ports 5001 and 5002, which have been added by default and received data for a period of time. Since the priority of 5003 is 2, higher than the other two slices, the bandwidth required by 5003 must be guaranteed. It can be seen from the figure that after the 5003 is added and starts receiving data, the bandwidth reaches 9M/s, it means the bandwidth is guaranteed. At the same time, the speed of the other two slices decreased significantly. After a period of time, port 5003 stops receiving data, and then the other two ports quickly resume receiving bandwidth. At this time, due to the automatic deletion mode set in the test, port 5003 has not been deleted, but no data has been received, that is to say, there is no The bandwidth is occupied, so the other two ports can recover at the same bandwidth as before. It is worth to mention that in addition to the slight fluctuations of the system itself when receiving data, when a new slice is added, the process of deleting the slice and adding the slice requires a certain processing time due to the indirect deletion. Although it does not take a long time, during this period, the receiving bandwidth of other slices is obviously fluctuating, but this short-term fluctuation does not affect the overall trend.

3.2 Adding mutiple sclices

3.2.1 Add multiple slices with different priority performance

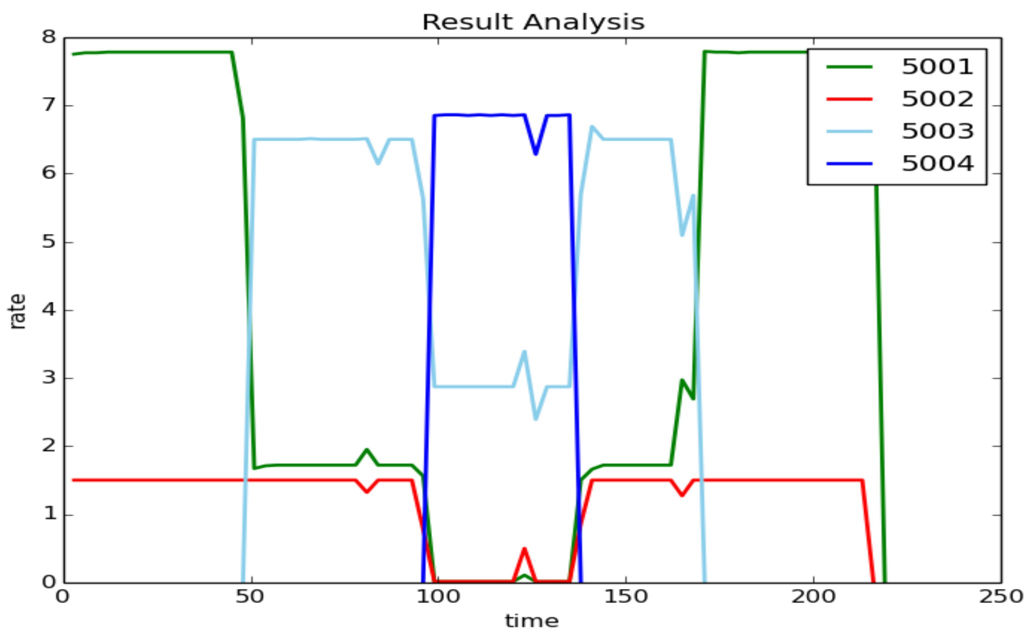


Abbildung 16 – multiple slices with different priority

As shown in the figure above, this is the result of bandwidth of each slice when adding slices with different priority in the network. There are only two original slices in the network, two ports 5001,5002 are being used. When a slice 5003 with a higher priority is added to at certain moment, the speed of 5001 will drop drastically to ensure the requirements of 5003. After a period of time, a new slice is added to the network. This new slice uses the port 5004 and its priority is higher than 5003. Therefore, in order to meet the needs of the port 5004, the other three slices will make changes for it. They will reduce their bandwidth to ensure port 5004 works properly. After port 5004 finished sending packets, the other three ports start to recover to the original Bandwidth. When 5003 port no longer sends information too, the 5001 and 5002 slices will return to the original state. This process also prove the correctness of our algorithm, ensuring that slices from higher priority will always be processed at first.

3.2.2 Same priority, different slice performance

In order to observe the performance of each slice from the same priority, we did two sets of experiments.

The same port number corresponds to the same requirement in both experiments,

Port number	5002	5003	5004	5005
Min_rate kbit/s	100	200	300	500
Priority	1	1	1	2

Abbildung 17 – pri list

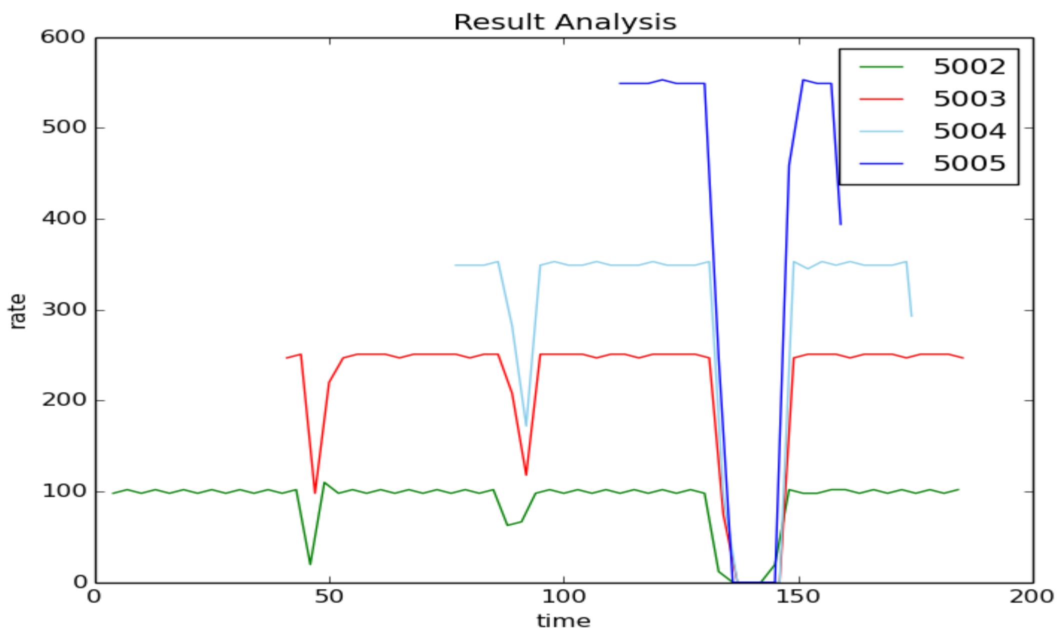


Abbildung 18 – 1.5M Bandwidth

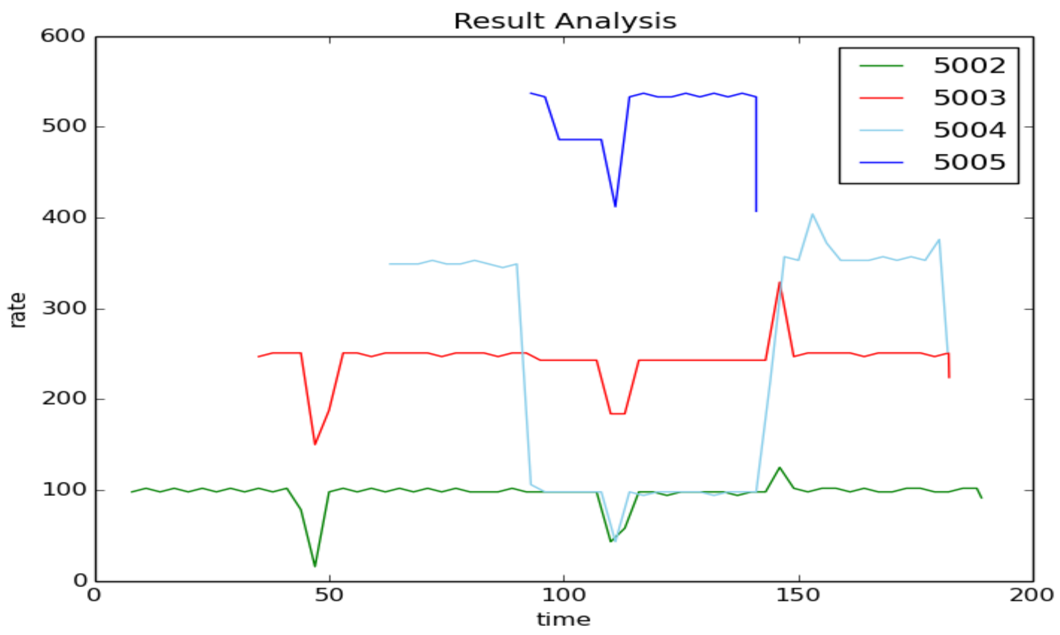


Abbildung 19 – 1M Bandwidth

However, the total bandwidth of the first set of experiments was set to 1.5M, and the total bandwidth of the second set of experiments was set to 1M. From the above table, it can be concluded that the bandwidth requirement of all port numbers is 1.1M, which means that the total bandwidth meets the requirements of all port numbers in the first set of experiments, so it can be seen from the figure that even 4 ports are used at the same time. It can also guarantee the banwidth that each port needs. But once the total bandwidth is insufficient for all ports, we will call the algorithm we mentioned earlier to reset the corresponding demand for all slices. As you can see in Figure 1, since the priority value of 5005 is higher than the other three ports, Its speed has been guaranteed first, then the other three ports, because for the same priority queues, the algorithm will let him add from small to large, the total bandwidth 1M minus the bandwidth used by 5005 500k, the remaining 500k can continue Guarantee the use of 5002, 5003 two slices, but the last port 5004 can only be allocated to the remaining bandwidth. It can also be seen from the figure that after adding port 5005, the speed of 5002, 5003 does not change, but the speed of 5004 drops a lot. After port 5005 finished sending packets, the speed of the 5004 is restored to its original state. The comparison of these two sets of experiments also proves another rule in the algorithm we designed, as much as possible to ensure more slices work properly.

3.3 Comparison of manual deletion and automatic deletion

As in the 3.1 test, when the new slice stops receiving data, the remaining two slices can be restored to their original bandwidth without the new slice being deleted. However, as shown in the figure below, the new slice on the first picture is the automatic deletion mode, and the second picture is the manual deletion mode. After the new slice 5003 stops receiving data, there is no significant difference between the two modes, but after receiving the data for the second time on port 5003, The performance of the two modes is significantly different. The first picture shows that after receiving the data for the second time, since the data is not received again in the timer inquiry period, the slice has been deleted, and port 5003 is redirected to the default queue, so the performance is basically the same as port 5001. However, the second picture does not send a manual delete request in the entire test, so the slice always exists, and the 9M/s receiving bandwidth can still be guaranteed when the slice receives the data again.

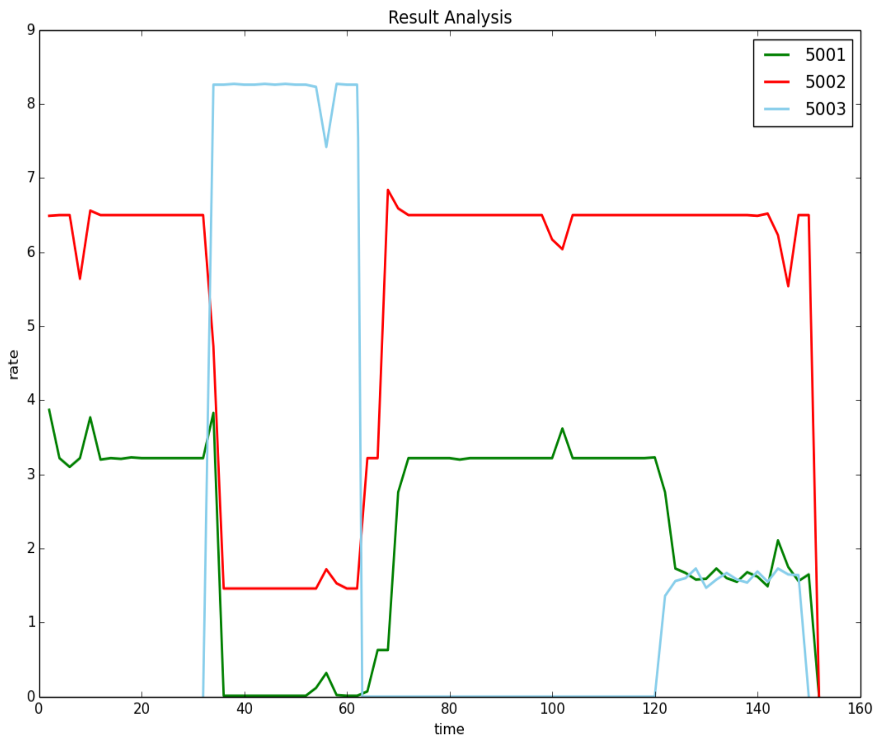


Abbildung 20 – automatic deletion

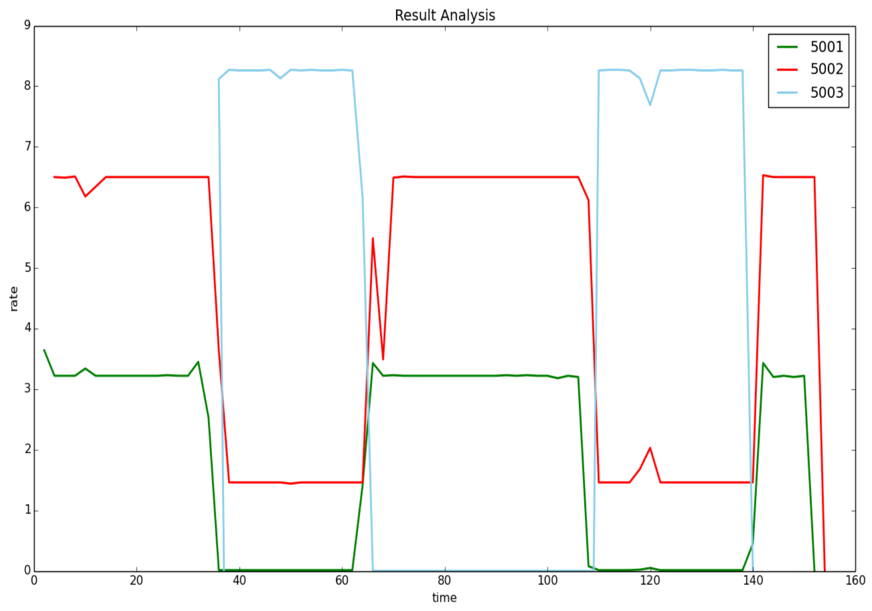


Abbildung 21 – manual deletion

When the sum of the slice bandwidths is greater than the total bandwidth, if the slice is not deleted in time, although the data is not received, the slice always exists because the priority is high, and the bandwidth requirement of other slices is squeezed, which may waste some resources. Therefore, when the client is unsure of the deletion mode, setting the automatic deletion mode can prevent waste of resources and better improve the utilization of bandwidth.

4 Conclusion

Problems encountered:

Beside the basic switch, the controller's specific structure and the process of receiving and sending information streams, we spent a lot of time trying to understand the theoretical knowledge about OpenvSwitch, Mininet, Ryu, and other frameworks and tools and after that we did a lot of related exercises to better grasp this knowledge.

In order to realize the network slicing function in the switch level, we carefully studied the QoS chapter in the Ryu tool book and learned the basic principles and application methods of the QoS service through practical exercises. After a long time of programming with the extention on the basis of rest_qos.py, the algorithm is improved and modified. Finally, the expected goal is achieved, which means users can add and delete specified slices according to their own needs. In addition, the demand for high priority slices will be prioritized.

Meanwhile, we have also encountered some difficult problems. In the management scheme of three slices, the program only needs to delete the last one of all the slices, but in the case of multiple slices the program can not delete the specified one, which may cause accidental deletion of other slices. After discussions and analysing, we developed the algorithm that can manage slices in an orderly manner in the case of multiple slices. The bandwidth resources are redistributed according to different priorities, using the indirect deletion of the slices to ensure that the specified slice is correctly deleted.

The next problem is how to determine if the slice is not used. In the beginning, we did not implement the function of periodically reading the flow table information. Since no method to directly judge the use of the slice, the scheme of automatically deleting the slice after adding a fixed time is adopted. However, this solution is lack flexiblility, as a result of that we spent a lot of time improving the algorithm, finding a way to read the information, determining whether the slice is stopped, and finally implement the user-defined deletion of the slice mode(Manual deletion and automatic deletion).

Insufficient:

Due to the limited time, we only implement the network slicing function in a switching level, and the difference between each slice is only limited to the bandwidth size, which cannot meet other requirements of network slicing such as low latency and high capacity.

Outlook:

In order to better realize the function of network slicing, we can extend to the router level based on our program supplemented by a multipath scheme, which customizes more functions for slicing and make better use of network resources.

5 Bibliography

[1]https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[2] Ben Pfaff et al. The Design and Implementation of Open vSwitch. In:Nsdi. 2015, pp. 117–130.isbn:9781931971218.
url:https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[3]Ben Pfaff and Bruce Davie. RFC 7047 - The Open vSwitch Database Management Protocol. 2013. doi: 10.17487/RFC7047. url: https://rfceditor.org/rfc/rfc7047.txt.

[4]https://community.cisco.com/t5/wireless-mobility-documents/iperf-test-for-measuring-the-throughput-speed-of-a-wlan-client/ta-p/3142047