Investigation of GINI router

Nianzhen GU[261044523], Xinyue HU[261043327], Yixin Dai[260835106]

December 22, 2021

1 Introduction

Routers are the machines that route Internet packets from hop to hop. Routers are the core part of the Internet. However, traditional ways of implementing routers, such as using special hardware, and using the software on the computer, are expensive and hard to modify existing routing operations. The GINI router is an all software-designed router that intends to implement as much functionality as the hard routers. It allows thorough experimentation with router functionalities easily and cost-effectively.

In this paper, we aim to investigate the basic operations of the GINI router and propose some suggestions for the existing problems. Firstly, we find the performance bottlenecks of the GINI router. Then we propose some methods to increase its packet processing rate. Secondly, we investigate how the GINI router is capable of handling diverse quality of service (QoS) requirements. We modify the GINI router to make weighted fair queuing (WFQ) work. We also find the software architecture features in the GINI router that inherently handicaps the application of QoS in a software router. Then we improve the design to improve the QoS delivered by GINI. Thirdly, we discuss how to change the design of the GINI router to a vertically scale. Then we list some limiting factors for vertical scaling. Fourthly, we illustrate the necessary changes for horizontal scaling in the current GINI router. Finally, we explain why the current GINI router is more suitable for horizontal scaling.

2 The performance bottlenecks and their improvements

2.1 The performance bottlenecks in the existing GINI router

According to Xu, the GINI router is designed with a module structure [4]. Different functions are divided into different modules. The UML-ADAPTER module is one of the modules in the GINI router. The UML-ADAPTER module is responsible for creating connections, and for receiving and sending packets. The QOS module implements Quality of Service (QoS) in the GINI router, which rearranges packets according to their priorities. If QoS is enabled, an IP packets will be sent to the QOS module for rearrangement according to their priority before forwarded to UML-ADAPTER. We believe these two modules are the performance bottlenecks in the GINI router. Table 1 shows the comparison of the time when sending a packet spends in each module. We can observe that time is extremely high for the UML-ADAPTER and QOS module, which means they are the bottlenecks that limit the packet processing rate of the GINI router.

More specifically, Figure 1 shows the design of a GINI router. The packet processor can be regarded as the UML-ADAPTER module. Packet classifiers, packet queues, and packet scheduler work together as QOS module. The reason is that there is only one thread in the UML-ADAPTER module (or only a single thread for the reader) to send the packets in the FIFO queues no matter to what interface. Also, the QOS module uses the queuing algorithm that costs time.

Also, we notice that although there are multi-threads for the writer, only one output queue is used. When the incoming speed of packets is fast, Head-of-the-Line (HOL) blocking problem may occur, which will cause a low processing rate.

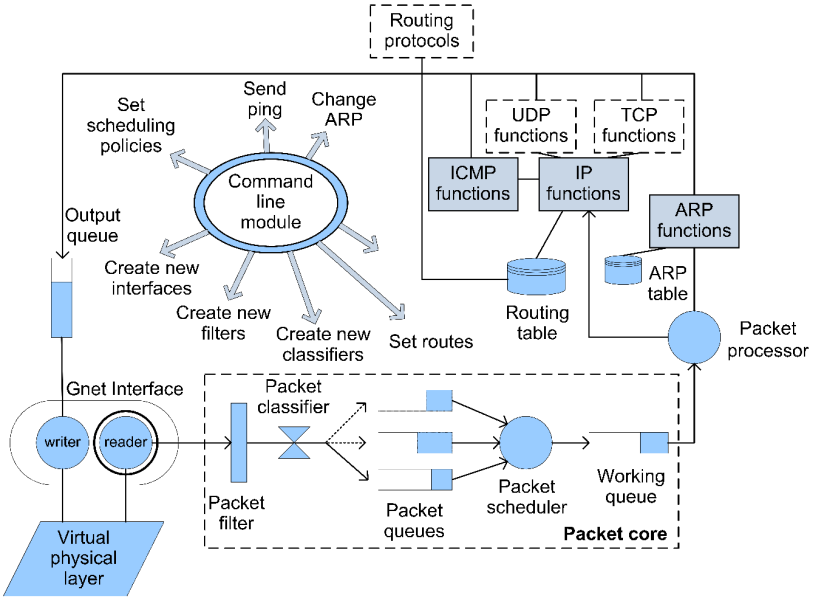


Figure 1: The Block diagram of the GINI router.

	Maximum Travel Time (μ sec)	Maximum Queue Length	Packet Drop
UML-ADAPTER	224195	49	0
IP	710	0	0
ARP	837	0	0
QOS	204812	50	486

Table 1: Internal Overload Behavior of the GINI Router [4].

2.2 Modify the GINI router to increase its packet processing rate

As mentioned above, there is only one thread for the reader to send the packets. Therefore, we can create more threads to increase its packet processing rate. For QoS, it is hard to improve since a faster queuing algorithm is needed. For the HOL blocking problem, we can create more output queues that have different categories. Therefore, different packets can go to their specific queue. Thus increasing the packet processing rate.

3 GINI router and QoS

3.1 How GINI router handles diverse QoS requirements

Figure 2 shows the internal structure of the QOS module. The Classifier determines the queue type of packets according to some predefined rules, e.g. Type of Service (TOS) bits, source and destination IP address, etc. Multiple queues are available to store packets classified as belonging

to different traffic. The Scheduler is responsible for picking the next packet from the set of queues and sending the selected packet to the work queue. Using the Command line module, we can create new classifiers to meet different QoS conditions.

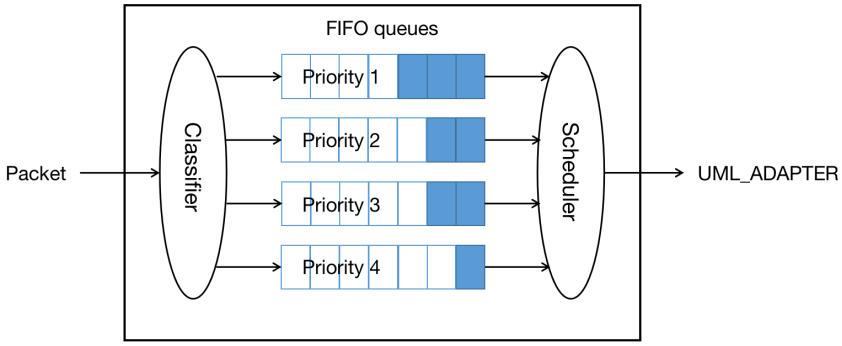


Figure 2: The internal structure of the QOS module.

3.2 Modify the GINI router to make WFQ work

Figure 3 shows the source code of the GINI router. We notice that the Round Robin Scheduler is used to dequeue.

```
pthread_t PktCoreSchedulerInit(pktcore_t *pcore)
{
        int threadstat;
        pthread_t threadid;

        threadstat = pthread_create((pthread_t *)&threadid, NULL, (void *)roundRobinScheduler, (void *)pcore);
        if (threadstat != 0)
        {
                verbose(1, "[PKTCoreSchedulerInit]:: unable to create thread.. ");
                return -1;
        }

        return threadid;
}
```

Figure 3: The dequeue method used in the Scheduler.

If we want to apply WFQ, we need to classify packets by flow (same source IP address, destination IP address, source port number, etc.), and each flow is assigned to a queue. This process is called hashing. WFQ allocates the bandwidth that each flow should occupy according to the priority of the flow. The smaller the priority value, the less bandwidth is obtained and vice versa. In this way, fairness between services of the same priority is ensured, and the weights between services of different priorities are reflected.

3.3 Software architecture features in the GINI router that handicaps the application of QoS and its improvement

In a software router, the different components are implemented by threads, which can be interfering with each other in terms of the delivered performance. Therefore, the GINI router needs to provide traffic isolation that split the available bandwidth in an outgoing link according to a specified fraction among the different traffic flows.

According to COMP535 lecture nodes, there are four principles of QoS: packet classification, scheduling and policing, resource utilization, and call admission. In the GINI router, we can make every flow declare its needs, then the Scheduler can allocate dynamic (non-sharable) bandwidth to it, which is an efficient use of bandwidth. Also, we can use a leaky bucket regulator to implement policing, which is beneficial when there is busy traffic.

4 GINI router with vertically scale

GINI routers can do the vertical scale without modification to some extents possible because allocating more resource to a specific GINI router is feasible, either by allocating more hardware resource to the whole GINI/whole virtual machine to use more processors or adjusting the parameter represents memory space of a router, the parameter might be found inside GINI architecture. However, it's not a good way to organize vertical scaling as each time adjusting some number means changing GINI at compile time, so this solution is inelegant and cumbersome.

There is a better strategy (run-time solution): as shown in Figure 4, it is better to change the design and create a separate VNF manager to perform the instantiation, vertical scaling, and termination. In addition, there is a need for an orchestrator since the functions and resources could break down if NFV orchestrator is not present to do the overall management. Fortunately, according to GINI5 documentation [1], orchestrator is already available in the latest version. By using these components, GINI routers can provide dynamic services whenever required to grow the service dimensions.

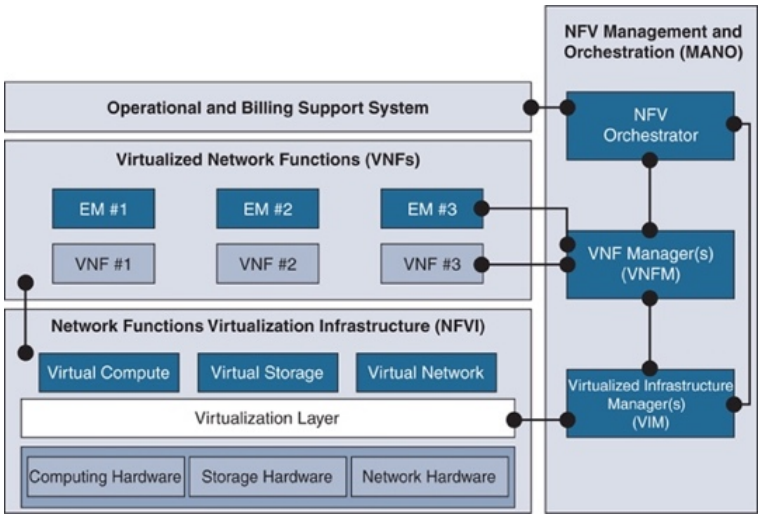


Figure 4: NFV architectural framework [3]

To make GINI even more flexible for vertically scaling and to take full advantage of the NFV architectural framework, a monitor can be added in orchestrator or managers that constantly checks the router's current load while the topology is running and can request and allocate more resources to the router if the threshold is exceeded (administrator must ensure that managers/orchestrator are given permission to change the number of resources a router occupies). At the same time, if a router is shown with a very low load for a long time, then it is likely that the router could function well even without so many resources, so we can safely release some resources and put them into the routers that need more space. This is indeed a tradeoff: it adds more complexity to the system, but it allows resources to be used more efficiently.

There are two main limiting factors of vertical scaling:

- One of the biggest limiting factors of vertical scaling is the single point failure problem, because all the resources and functionality are added into a signal instance (a box running as a router), all services the router provides are terminated if that box has a problem.
- Another major disadvantage is that it is not practical to always allocate resources to a single box because the hardware has capacity, it will run out of memory if the user tries to aggregate too many resources into a single box. The maximum performance can get out of a single box really depends on the architecture and capability of VNF, and it is not infinite!

5 GINI router with horizontal scale

Horizontal scaling in the GINI router can be considered as creating instances of routers somewhere, the simplest strategy is to add more packet processors within a router and a fast load balancer between the working queue and packet processors, load balancer will select the optimal packet processor (less load, for example) to do the next job and in case all the packet processors are about to be full, it requests that a new packet processor be constructed. The processor will know which function should be deployed to the current packet (UDP or TCP, for example) and send it to the output queue. The advantage of this solution is that the idea of creating multiple packet processors inside a single router make the instantiation less expensive (less overhead) compared with creating multiple router instances. But the disadvantages are obvious, since this approach treats each processor as a mini router (with incomplete functionality) and there is still only one packet core and output queue (bottleneck) in a router, the overall computing speed is limited.

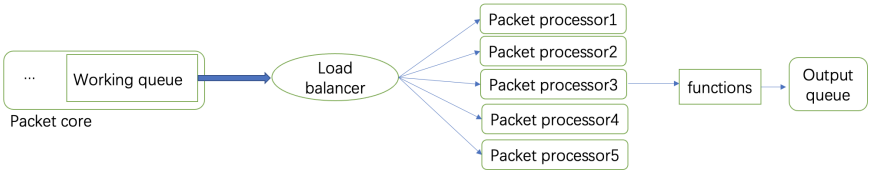


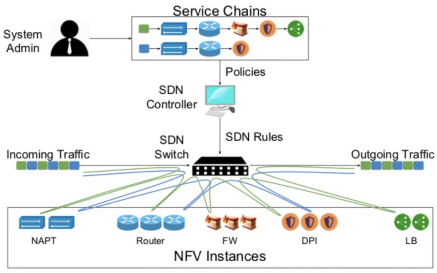
Figure 5: Multiple processors in a router.

Creating whole copies of the router and copies of load balancer eliminates the above shortcomings and greatly improves the overall performance. To make the GINI router work together with other types of instances (not just router), service function chain needs to be implemented, note that the latest version of GINI5 has support some basic functionalities of service function chain (SFC) [1], such as adding/deleting new VNF instance, creating a service function chain from active VNF and creating a service path from source to destination and traverse a service chain in order. To make all the instances work, controller needs to pass rules to virtual switch and then to boxes: assign them different tasks in correct sequence. The good thing happening now is that those commands about adding node, chain and path indeed make sure creating new instances and apply any services the system admin wants at run time, respecting the important idea of VNF's dynamic service provisioning!

There are two ways of doing load balancing, one way is to add instances of routers and load balancers and connect them as a chain, the benefit is load balancing is done explicitly so users can see the load balancers from the user interface clearly, while the drawback is that it brings up the scale of the network topology and increases the amount of network services. Another way is to modify the OF Controller and make it able to server as an implicit load balancer, in this way GINI can do load balancing in one shot and avoid any external load balancer! One important problem to

```
$ sfc_init
SFC orchestrator started
$ sfc_addnode firewall f01
Service added
$ sfc_addchain f01
Service chain created with id 7a93b5007f40-1
$ sfc_addpath internet web 7a93b5007f40-1
```

(a) SFC command in GINI5 [1]



(b) Service function chain

Figure 6: The SFC command in GINI5 and its architecture.

think about is what to do if a VNF instance fails in a chain, controller in implicit load balancer or external load balancer needs to be more powerful to mitigate the service interruption, one possible way is to create backup instances and activate them when the primary one goes down, keeping the backup in sync with primary one.

If this question is intended to ask how to modify the old versions of GINI (without SFC), which means to build up the NFV management and orchestration from the start, then the following are the requirements GINI should meet [2]:

- Add functionality that allows creating of service node, service chain and service path.
- Add functionality that handles instance failure.
- Add functionality that brings up boxes on demand basis.
- Add a controller that can do explicit load balancing and pass traffic rules.

6 GINI Why GINI router is more suitable for horizonal scaling

The current GINI router is more suitable for horizontal scaling, because horizontal scaling can promise a better service and performance of GINI:

- Scaling out is just adding more instance and would not result in boxes getting caught in a resource deficit (hardware limited problem).
- Horizontal scaling is more fault-tolerant: if one point is failure other instance can still be able to run and restore the service.
- Enables load balancing is easy in horizontal scaling, while it needs much more work in vertical scaling.

7 Conclusion

In this paper, we do some investigations on the GINI router. Especially, we focus on the GINI router performance, QoS, and horizontal or vertical scale. Different section answers the different question of the COMP535 PA2.

References

[1] Service function chaining. URL: https://citelab.github.io/gini5/features/service-function-chaining/.

[2] l4-l7 service function chaining solution, Jun 2015. URL: https://opennetworking.org/wp-content/uploads/2014/10/L4-L7_Service_Function_Chaining_Solution_Architecture.pdf.

[3] Rajendra Chayapathi, Syed Farrukh Hassan, and Paresh Shah. *Network functions virtualization (NFV) with a touch of SDN*. Addison-Wesley, 2017.

[4] Weiling Xu. The gini router: A routing element for user-level micro internet. Jun 2004.