An End-to-End QoS Routing on Software Defined Network Based on Hierarchical Token Bucket Queuing Discipline

Shuangyin Ren
School of Computer, National University of Defense Technology
Changsha China, 410073
Tel: +8615084833573
renshuangyin@gmail.com

Quanyou Feng
School of Computer, National University of Defense Technology
Changsha China, 410073
Tel: +8613548636396
fengquanyou@foxmail.com

Wenhua Dou
School of Computer, National University of Defense Technology
Changsha China, 410073
Tel: +8613786195386
douwh@vip.sina.com

ABSTRACT

Software Defined Network is a network architecture where network control is decoupled from forwarding plane and is directly programmable. Control plane monitors data plane's behavior and retrieves an integrated view of data plane. Comparing with classic RSVP based Integrated Services, Decoupled control plane provides a new access to End-to-End guarantee QoS routing. An End-to-End QoS routing algorithm is realized in Mininet with Open vSwitch as forwarding switch and Ryu as remote controller. Open vSwitch here utilizes Hierarchical Token Bucket queuing discipline to manage bandwidth. Routing algorithms are realized in Ryu controller based on graphs for Best Effort and QoS data flows respectively. Shortest path routing runs for Best Effort flow based on distance graph of switches. Shortest QoS path routing runs for QoS flow based on resource residual graph. System verification and simulation shows that routing algorithms work with different kinds of tests, and routing algorithm could also work if some failure are introduced.

CCS Concepts

• Networks → Network protocols → Network layer protocols → Routing protocols • Network algorithms → Control path algorithms → Network resources allocation

Keywords

Software Defined Network; QoS; Routing Algorithm; Hierarchical Token Bucket

1. INTRODUCTION

Open Networking Foundation (ONF) defines Software Defined Networks (SDN) as an emerging network architecture where network control is decoupled from forwarding and is directly programmable [1]. This decoupled and centralized controller deprives control and routing functions from switch/route devices, this new paradigm brings new opportunity for Quality of Service (QoS) guarantee.

In traditional and even most current networks, network functions are embedded into networking nodes like hub, switch and router. Different QoS service models have been proposed in traditional networks in last two decades: Integrated Services (IntServ) and Differential Services (DiffServ). IntServ runs Resource Reservation Protocol (RSVP) to reserve resources in links and signals this reservation periodically. RSVP deteriorates network performance severely as topology scales up. DiffServ runs in per hop manner which can't provide End-to-End QoS guarantee.

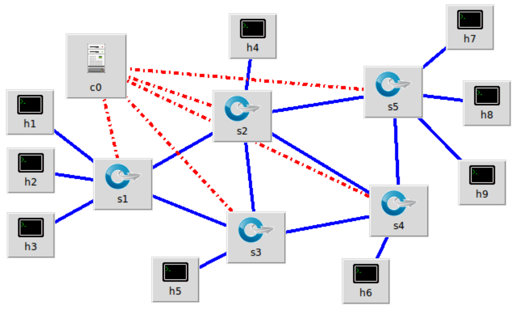
SDN paradigm is easier to provide an End-to-End QoS guarantee from is nature. SDN architecture composed by data plane, control plane and application plane. Control plane provides a relatively centralized control to switch or route devices through southbound APIs like Openflow protocol. Data plane forwards or drops traffic flow according to created flow tables, filters, classifiers and so on. In Openflow protocol, control plane retrieves data plane information and on the other hand data plane sends events like *packets_in* or *link_failure* to control plane. Based on communications with switches, controller retrieves an entire view of data plane and develop QoS algorithms to provide an End-to-End guarantee.

Xia, W. etc. [2] and Kreutz etc. [3] did comprehensive surveys on SDN. S. Tomovic proposed a SDN controller QoS routing framework for priority flows based on bandwidth resource utilization [4], however no details provided about how to realize it. In [5] M. Celenlioglu proposed routing and resource management algorithm on pre-established multi paths, SDN controller select one of paths base on their costs. H. Krishna [6] designed a routing in Ryu controller, which routes QoS flow with a Widest Shortest Path algorithm and routes Best Effort (BE) flow with a Shortest Path algorithm, however algorithms are partly realized and tested in this work. Cong L. etc. realized a software defined routing in NoC [7] and simulated power consumption and network performance comparing with traditional routing schemes. Jinzhen B. etc. [8] proposed a software defined SpaceWire network architecture to ensure end-to-end QoS by a fine grained flow control.

In this study, an End-to-End QoS guaranteed routing algorithm is realized in Mininet with Open vSwitch (OVS) as forwarding switch and Ryu [9] as remote controller. Queue discipline is realized under Hierarchical Token Bucket (HTB) classes in data plane. A network system is realized and verificated from different aspects, routing algorithms are tested in topology with 9 hosts and 5 switches. This paper is organized as this, section I is research background and related work. Routing and resource management algorithms are introduced in section II and section III is system

DMCIT '17, May 25-27, 2017, Phuket , Thailand
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5218-5/17/05…$15.00
http://dx.doi.org/10.1145/3089871.3089883

verification, simulation results and simulation analysis. Conclusions come at section IV.



2. QOS ROUTING ALGORITHMS AND RESOURCE MANAGEMENT

2.1 Routing Algorithm

When a new flow starts from a host src, src sends this flow to its connected switch *s_src*, as shown in Fig.1. If a flow table existed and hit in *s_src*, *s_src* will execute actions at the end of the hit flow table, otherwise *s_src* sends flow packets to controller c_0. To start with, c_0 need to get which switch *dst* connected to. In initializing step, network aware module generates an access table to map all hosts to its connected switch and port. According to this access table, c_0 could locate *s_dst* by *dst* host. Routes selection from *s_src* to *s_dst* differs based on its Type of Service (ToS) bits. BE flow is labeled with ToS as 0, QoS flow is labeled with different ToS values, standing for different QoS guarantees.

For a BE flow, shortest route is selected based on Dijkstra algorithm, while for QoS flow, route is selected according to QoS-shortest algorithm as shown in Pseudo Code 1. In QoS-shortest algorithm, controller initiates a resource residual graph from topology graph, named QoS graph. After a QoS route created, controller run functions to install flow tables in OVS switches, then QoS graph is updated if all flow tables are installed successfully. For the second QoS data flow, QoS-shortest algorithm takes most updated QoS graph as input to create a route. If there are some QoS routes created in a link and there are not enough bandwidth resource left for a new data flow, then QoS-shortest algorithm marks this link cost as *inf* and bypass this link. QoS-shortest algorithm selects the shortest route from all routes that have enough residual bandwidth resource.

2.2 Resource Management and Link Aggregation

OVS is an open source software implementation of switch that could be run on virtualized environments like VMs or hardware environments on switches. OVS integrated Openflow APIs to communicate with control plane. These APIs enables remote controller to update OVS flow table, set firewalls etc. While OVS and Openflow are not agree with each other over traffic control queuing disciplines [10], queue configurations are still can't realized through Openflow APIs. In this proposal, flow tables are controlled by remote SDN controller, queue discipline is realized in OVS commands in data plane.

In controller, bandwidth is assigned according to ToS bits, however as Ryu controller can't modify queues in OVS database (OVSDB), bandwidth distribution is composed by controller part and HTB parts. In Openflow version 1.3, enqueue action is disabled while a *set_queue* action added. Action *set_queue* will

Pseudo Code 1 QoS Routing Algorithm

```
Input: $Graph, src, BW\_resv$
Return: $Path$
function QOS_SHORTEST($Graph, src, BW\_resv$)
    $Switch\_list \leftarrow Graph.keys$
    if $src == None$ or $Graph == None$ then
        return $None$
    end if
    if $QoS\_graph == None$ then
        $QoS\_graph \leftarrow$ GRAPH2QOSGRAPH($Graph$)
    end if
    for all $i \in Switch\_list$ do
        for all $j \in Switch\_list$ do
            if $BW\_resv >= BW\_residual$ then
                $QoS\_graph[i][j] \leftarrow inf$
            end if
        end for
    end for
    $Path \leftarrow$ SHORTEST_DIJKSTRA($QoS\_graph, src$)
    return $Path$
end function
function SHORTEST_DIJKSTRA($Graph, src$)
    $visited \leftarrow [src]$
    $path \leftarrow \{src : \{src : []\}\}$
    $Dist\_graph \leftarrow \{src : 0\}$
    $pre \leftarrow next \leftarrow src$
    $Switch\_list.remove(src)$
    while $Switch\_list$ do
        for all $v \in visited$ do
            for all $d \in Switch\_list$ do
                $new\_dist \leftarrow dist\_src2v + dist\_v2d$
                if $new\_dist <= pre\_dist$ then
                    $pre\_dist \leftarrow new\_dist$
                    $next \leftarrow d$
                    $pre \leftarrow v$
                    $dist\_src2d \leftarrow new\_dist$
                end if
            end for
        end for
        $path[src2next].append(next)$
        $visited.append(next)$
        $Switch\_list.remove(next)$
    end while
    return $Path$
end function
```

set an existed queue ID for a data flow.

There are no actions available to configure a queue in OVS switch from Ryu controller. So assigned bandwidth by Ryu controller parts can't actually be distributed to a data flow directly. This does not mean resource management algorithm can't work. From controller's view, assigned resource and residual resource is taken into account during route computing. Flow will be forwarded to a route with enough residual resource, in other words, if a route is created then there must be enough bandwidth resource in data plane to guarantee a QoS flow, otherwise this route would not be created. From data plane's view, HTB queuing discipline introduced min rate and max rate, min rate is a guaranteed data rate for a QoS flow and max rate is QoS flow could get at most when network is light loaded. HTB introduces bandwidth 'borrowing' mechanism, which means that even though bandwidth is assigned, a flow could get more bandwidth if there are idle bandwidth from its parent class.

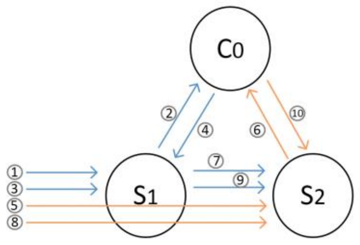


Figure 2. Packets arriving sequence before flow table installed.

In this proposal two queues are configured, QoS data flows are aggregated into one queue while BE data flow aggregated into the other. QoS queue and BE queue could borrow bandwidth resource from each other. This design could satisfy the bandwidth guarantee under constraint that Ryu controller can't configure queues. So HTB QoS is configured for each port and two queues added under HTB QoS. Traffic flows are distributed to different queues by flow tables depending on output port and ToS bits. In this manner, guaranteed bandwidth is satisfied from control plane and data plane.

In Ryu controller c_0, bandwidth resource is distributed according to ToS bits. c_0 assigns 8Mbps bandwidth to QoS flow with ToS as 128, 4 Mbps is to flow with ToS as 64, 2Mbps for ToS as 32. For a BE data flow with ToS as 0, 1Mbps bandwidth is distributed as min-rate in data plane, QoS graph will not be updated.

2.3 Progress lock during creating flow tables

When host *src* starts a new flow and a corresponding flow table in s_1 does not existed, s_1 sends packets to controller c_0. During controller c_0 running routing algorithms and switch s_1 s_2 installing flow tables, s_1 and s_2 continues sending packets to c_0, as shown in Fig.2. When packet ① arrives s_1, s_1 sends request to c_0, s_1 receives flow ③ before flow table installed, the same to switch s_2.

In QoS routing algorithm, a progress lock is added to lock packets of the same flow, so as to prevent consuming bandwidth resource by the same flow. Resource residual map is only updated after flow tables are added in switches successfully. In progress lock a tuple of <*ip_src, ip_dst, ip_tos*> is utilized to identify a data flow. All packets with the same *ip_src, ip_dst, ip_tos* are stored before flow tables are installed in all switches in a routing path.

3. VERIFICATION AND SIMULATION RESULTS

A SDN network is realized on Mininet as shown in Fig.1. Controller c_0 is realized in Ryu. Ryu controller c_0 is composed by three components: Network Monitor, Network Aware and Routing & Resource Management. All switches in this simulation are OVS. Openflow protocol version is v1.3. Time stamp in Mininet is real time, simulation speed varies according to computing power. Hardware server is Thinkpad X220 with dual-core 2.3GHz i5 as CPU, and 64bit Ubuntu14.04 as system.

Before simulating routing and resource management algorithms, system is verificated from several aspects.

3.1 System Verification

3.1.1 Hierarchical Token Bucket min/max Rate

In HTB queue discipline, nodes utilize buckets and tokens for bandwidth sharing. HTB includes root class, interior class and leaf class. Classes are descripted by assured rate *AR*, ceil rate *CR*, priority *P* etc. Packets could be dequeued to output port only if there are available tokens in that corresponding leaf class. For each leaf class, if data rate exceeds AR and less than CR, and if there are excess bandwidth from its parent class, this leaf class could borrow bandwidth from its parent. A simulation of bandwidth sharing in HTB QoS link is shown in Fig.3-Fig.5, and simulation parameters are shown in Table1. During these three tests, total bandwidth is 200Mbps, minimum and maximum data rate for QoS flow is 140Mbps and 200Mbps respectively, while minimum and maximum data rate for BE flow is 60Mbps and 200Mbps respectively.

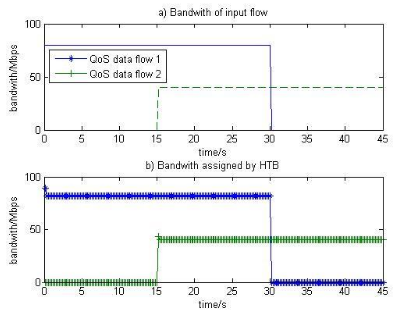


Figure 3. Bandwidth assignment in Hierarchical Token Bucket for both QoS flows.

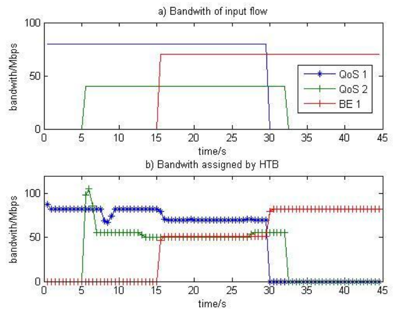


Figure 4. Bandwidth assignment for Best Effort flow in Hierarchical Token Bucket.

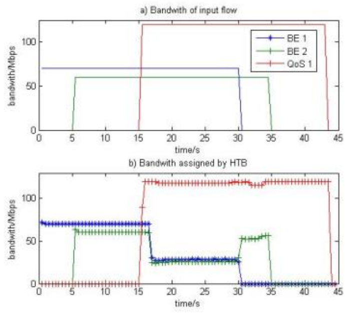


Figure 5. Bandwidth assignment for QoS flow in Hierarchical Token Bucket.

In Fig.3 two QoS data flow share QoS bandwidth without any confliction, and since routing algorithm will not accept any QoS data flow if this link can't provide a guaranteed service, so there would be no conflict happens. In Fig.4, a 70Mbps BE data flow is tested under preinstalled 80Mbps and 50Mbps QoS data flow, Fig.4 shows that BE data flow could only get enough bandwidth until QoS data flow release its resource, before QoS data flow releases its resource, BE data flow get its minimum rate bandwidth. In Fig.5, a 120Mbps QoS data flow is tested under two pre-installed BE data flows, which are 70Mbps and 60Mbps

Table 1. Parameters of HTB resource management simulation

	Testing flow		Conflict flow	
	Bandwidth /type	Time (start/end)	Bandwidth /type	Time (start/end)
1	80Mbps/QoS 40Mbps/QoS	0/30s 15/45s	null	
2	70Mbps/BE	15/45s	80Mbps/QoS 50Mbps/QoS	0/30s 5/35s
3	120Mbps/QoS	15/45s	70Mbps/BE 60Mbps/BE	0/30s 5/35s

respectively. Fig.4 shows that QoS could get its guaranteed bandwidth after installed, pre-installed BE data flows would free part of their under using bandwidth and just keep minimum rate bandwidth. Starting and ending time of data flows are shown in Fig.3-Fig.5 time labels.

In HTB queue discipline BE flows could receive enough bandwidth in light loaded networks by borrowing bandwidth from parent class. However, for a heavy loaded environment competitions happen when there are more BE flow than a link could hold, and packet loss happens. Fig. 6 shows data loss rate of BE flows in a heavy loaded environment. This simulation reduced BE minimum bandwidth to 20Mbps. When flow f_3 starts, competition in BE queue brings data loss increase in all three BE flows. As routing algorithm would not accept new QoS data flow if its bandwidth can't be guaranteed, obvious packet loss rate can't happen.

3.1.2 *Failing a route*

QoS routing algorithm creates routes according to available resource, however a route may not function because of some other settings, such as a firewall is added in OVS to reject a flow from specific port like *iptables -A FORWARD -i s1-eth1 -o s1-eth4 -j DROP*, or a flow table already existed with actions to drop any received packet *ovs-ofctl add-flow s1 "dl type=0x0800,nw proto=17, in port=1 idle timeout=0 actions=drop"*, in which manner a routing created however it can't function in OVS. In this scenario, information can't be updated to controller plane so controller QoS routing algorithm can't create a successful route, this malfunction flow table would self-destruct after idle timeout seconds. Bandwidth reserve/release and idle timeout is shown in Fig.7.

While if a links fails by OVS deleting link commands, data plane issues an event to control plane then controller updates network graph and QoS graph for future data flows.

3.2 Simulation of Routing and Resource Management

Routing algorithm creates route for data flow based on its TOS and available bandwidth. For a QoS flow, routing algorithm creates a shortest path route with prerequisite that all heavy loaded links are not selected, while for a best effort flow a shortest path route is selected based on Dijkstra Algorithm.

A simulation of QoS routes and BE routes is shown in Fig. 8. In Fig. 8 f_1, f_2, f_3 and f_4 starts in a time sequence. All links between switches have a capacity of 200Mbps, QoS min-rate/max-rate is 140Mbps and 200Mbps respectively. f_1 clarifies a bandwidth guarantee of 80Mbps, Both f_2 and f_3 clarify a bandwidth guarantee

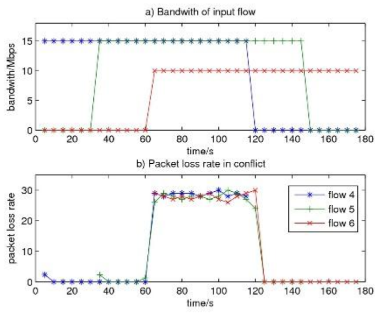


Figure 6. Best Effort flow data loss rate in heavy loaded networks.

of 40Mbps. When f_3 starts, resource residual in link s_1–s_2 and s_2–s_5, is less than guaranteed bandwidth, c_0 QoS algorithm bypass

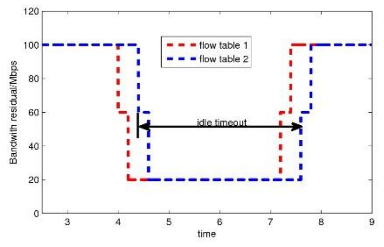


Figure 7. Bandwidth Reserve and Release for self-deconstruction.

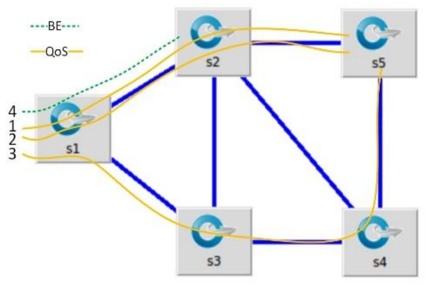


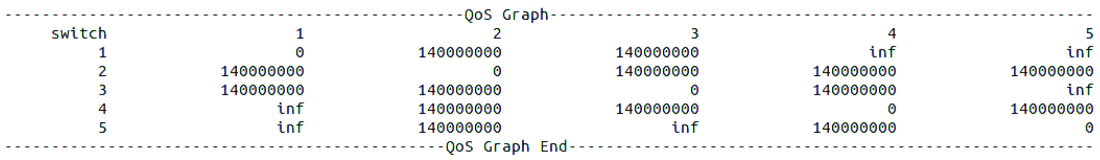
Figure 8. Route selection for QoS and BE flow based on traffic loads.

this shortest link and chooses a second shortest link s_1–s_3–s_4–s_5. Figure 9.1 is initial bandwidth before f_1, f_2, f_3 installed, and residual bandwidth after f_1, f_2, f_3 is shown in Figure 9.2. If residual bandwidth of all available route links can't guarantee a flow f_i routing algorithm declines it.

For a BE data flow, QoS routing algorithm chooses a shortest path even though it is heavy loaded. So route of flow f_4 is s_1-s_2 otherwise $s_1-s_3-s_2$. From Pseudo Code 1, the running time of shortest path routing based on Dijkstra algorithm is $O(n^3)$, n is number of OVS switches. Memory overhead of routing algorithm is $O(3V)$ to save maps, paths and visited switches, where V is a set of all network nodes. The speed for QoS routing algorithm is, $O(n^3)$. Memory overhead in QoS routing algorithm is $O(4V)$, as a *qos_graph* is added to save residual bandwidth for each link.

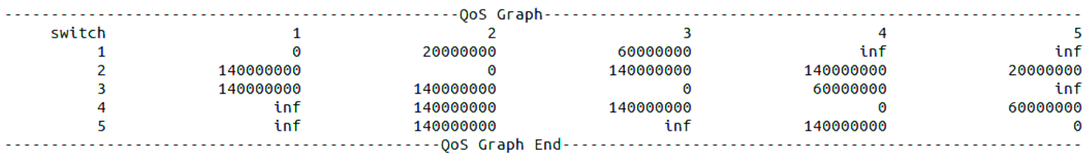
4. CONCLUSIONS

An End-to-End QoS routing algorithm is realized in Mininet with OVS as forwarding switch and Ryu as remote controller. HTB queuing discipline is utilized to share links and aggregate data



```
-----------------------------------------------QoS Graph---------------------------------------------------------
    switch                    1                    2                    3                    4                    5
         1                    0            140000000            140000000                  inf                  inf
         2            140000000                    0            140000000            140000000            140000000
         3            140000000            140000000                    0            140000000                  inf
         4                  inf            140000000            140000000                    0            140000000
         5                  inf            140000000                  inf            140000000                    0
---------------------------------------------QoS Graph End--------------------------------------------------------
```

Figure 9.1 Initial QoS bandwidth resource after network created (unit:bps).



```
-----------------------------------------------QoS Graph---------------------------------------------------------
    switch                    1                    2                    3                    4                    5
         1                    0             20000000             60000000                  inf                  inf
         2            140000000                    0            140000000            140000000             20000000
         3            140000000            140000000                    0             60000000                  inf
         4                  inf            140000000            140000000                    0             60000000
         5                  inf            140000000                  inf            140000000                    0
---------------------------------------------QoS Graph End--------------------------------------------------------
```

Figure 9.2 Residual QoS bandwidth resource after f_1 f_2 f_3 installed (unit:bps).

flows. Routing algorithms for BE flow and QoS flows are realized respectively in Ryu controller. Shortest path routing runs for BE flow based on distance map of switches. Shortest QoS path routing runs for QoS flow based on resource residual graph. System verification and simulation shows that this routing algorithms works with different kinds of tests. QoS routing algorithm could also work if some switches or links fail.

5. ACKNOWLEDGMENTS

This research if partially supported by the grants from National Natural Science Foundation of China (Grant No.61402502)

6. REFERENCES

[1] Foundation, Open Networking, 2012. *Software-defined networking: The new norm for networks*. ONF White Paper.

[2] Xia, W., Wen, Y., Foh, C. H., Niyato, D., & Xie, H. 2015. *A survey on software-defined networking*. IEEE Communications Surveys & Tutorials, 17(1), 27-51. DOI= https://doi.org/10.1109/COMST.2014.2330903

[3] Kreutz, Diego, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. 2015. *Software-defined networking: A comprehensive survey*. Proceedings of the IEEE 103, no. 1, 14-76. DOI= https://doi.org/10.1109/JPROC.2014.2371999

[4] Tomovic, Slavica, Neeli P., and Igor Radusinovic. 2014. *SDN control framework for QoS provisioning*. In Telecommunications Forum Telfor (TELFOR), 22nd, 111-114. DOI= https://doi.org/10.1109/TELFOR.2014.7034369

[5] Celenlioglu, M. Rasih, and H. Ali Mantar. 2015. *An SDN Based Intra-Domain Routing and Resource Management Model*. In Cloud Engineering (IC2E), 2015 IEEE International Conference on, 347-352. DOI=https://doi.org/10.1109/IC2E.2015.47

[6] Hedi K. 2016. *Providing End-to-end Bandwidth Guarantees with OpenFlow*. Master thesis. Delft University of Technology. July.

[7] Cong L., Wang W., and Zhiying W.. 2014. *A configurable, programmable and software-defined network on chip*. IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA), 813-816. DOI= https://doi.org/10.1109/WARTIA.2014.6976396

[8] Jinzhen B., Baokang Z., Zhenghu G., Chunqing W., Wanrong Y., and Zhenqian F.. 2014. *Towards Software Defined SpaceWire Networks*. In SpaceWire Conference (SpaceWire), International, 1-4. IEEE. DOI=https://doi.org/10.1109/SpaceWire.2014.6936256

[9] Ryu, S. D. N. 2015. *Framework Community, Ryu SDN Framework*. http://osrg.github.io/ryu.

[10] Team, Doxygen. 2016. *Mininet Python API Reference Manual*. http://mininet.org/api/classmininet_1_1node_1_1OVSSwitch.html