TECHNICAL UNIVERSITY OF DENMARK

34359 Software Defined Networking

Load Balancer Service

Written by:
Mingyuan Zang s171352

Supervisors:
José Soler
Angelos Mimidis

May 20, 2019

Abstract

This report provides a design and implementation description of a server load balancing service in Software-defined Network (SDN). The design is based on ONOS controller with OpenFlow as southbound interface. The objectives for the project are presented, and all the design and implementations to reach these objectives are explained in details. The testing is based on Mininet and Python scripts. The testing strategies and results are presented and analyzed in this report.

Contents

1 Problem Statement 3
2 Project Design 4
    2.1 Project Description 4
    2.2 Involved Entities and Network Relations 5
    2.3 Service Behavior Description 5
3 Project Implementation 7
    3.1 Implementation Overview 7
    3.2 ARP Handling 8
    3.3 HTTP Handling 9
4 Project Testing Strategy 15
    4.1 Testing Tools 15
    4.2 Dynamic Topology Testing Strategy 15
    4.3 Load Balancing Performance Testing Strategy 16
5 Results Analysis 18
    5.1 Dynamic Topology Testing Results 18
    5.2 Load Balancing Performance Testing Results 19
6 Critical Conclusion 21
    6.1 Future Work 22
References 23
Appendix 24

1 Problem Statement

As the scale of online services growing fast, the service providers would run a group of servers offering the same resources to handle a large number of requests from clients. The Load Balancer plays an important role to distribute the requests to underutilized server in order to efficiently use the network resources and speed up the request handling process. Traditional Load Balancer is hardcoded in a dedicated hardware device and cannot be agile with dynamic network changes. SDN-based Load Balancer can offer the programmability and flexibility of this process. The controller can offer a centralized management over the network.

There are mainly three types of load balancing service: server load balancing, global load balancing and network balancing.[1] This project focuses on server load balancing service as shown in figure 1.1. In this service, the Load Balancer stands in between the clients and servers. The clients send requests to a virtual public IP address. The Load Balancer receives all these requests and distributes the requests to servers based on a selection criterion. The criterion can be based on network conditions, server capabilities, etc. By providing a load balancing service like this, the clients don't need to know the group of servers behind the Load Balancer, and the servers won't know the existence of the Load Balancer.

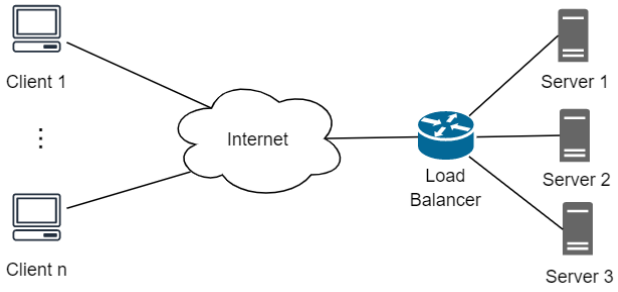


Figure 1.1 Server load balancing network

To make the implementation of the service in SDN close to the real case, the objectives of the project are:

- the service is able to work in more network topologies, ideally any
- the service is based on a more meaningful selection method (e.g. network conditions)
- the service is implemented based on ONOS controller with OpenFlow as Southbound Interface

2 Project Design

2.1 Project Description

The design of the project is mainly about introducing the ONOS controller to work at the control plane. It manages the data plane by monitoring the network and installing flow rules to the switches via southbound interface OpenFlow to instruct them how to handle the arriving packet. In the load balancing service, it is ONOS controller's job to decide which server the request should be forwarded based on the network status information collected from the switches. On the data plane, it is the Load Balancer who actually does the header rewriting and forwarding towards the selected server. In this way, the controller needs to associate the virtual IP and MAC address to the Load Balancer so that every request with this destination IP address would be forwarded to the Load Balancer. To implement this process in the controller, the following assumptions need to be made:

- MAC addresses of the group of severs are known
- the device ID of Load Balancer is known

Having the target server selected, the controller needs to install flow rules to the Load Balancer to inform it the server address which the Load Balancer need to rewrite for the coming request/response. In this project, since the reactive forwarding service in ONOS controller is deactivated, it's also the controller's job to instruct the forwarding process for each switch. To handle the case when there are more than one routing path between two switches, a routing path selection rule need to be defined. Having decided the routing paths, the flow rules are installed to the target switches. On the data plane, the switches will handle the packet based on flow entries. Figure 2.1 illustrates the main steps discussed above to forward a request from the client to the target server under the instructions from controller. The reverse way back to the client has a similar process.

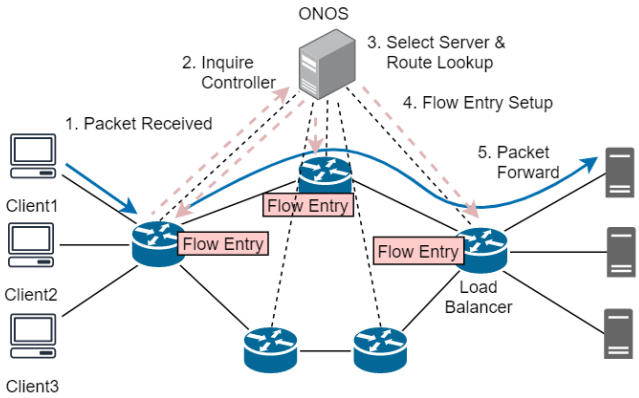


Figure 2.1 Steps to forward a request to target server

To achieve the objectives listed in previous section, possible network topologies and dynamic changes are considered in this project. By collecting network status data from

the switches, ONOS controller could know the network changes (e.g. link up/down, link load etc.) and makes decisions dynamically based on changes. As for the server selection criteria, a method based on real-time link conditions to servers is considered to support the dynamic network scenarios.

2.2 Involved Entities and Network Relations

To enable the service to work in any network topologies, the ideas start from two main kind of topologies: flat topology and hierarchical topology as shown in figure 2.2. Usually, the Load Balancer would be placed near the server side. In the flat topology, the Load Balancer would be preferred to be the edge switch connecting to the group of servers. While in the hierarchical topology, it would be better to place the Load Balancer on an upper layer in consideration of scalability problem.

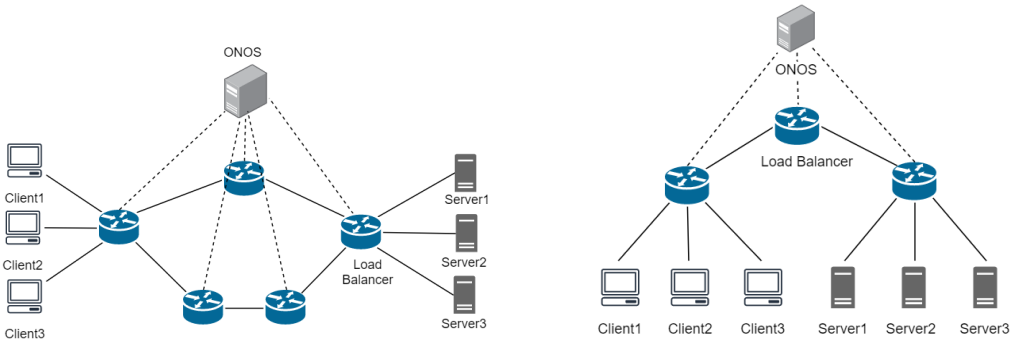


Figure 2.2 Flat topology network (left) and hierarchical topology network (right)

In both network topology scenarios, five entities are involved: client, server, switch, Load Balancer, and ONOS controller. Their functions are listed as followings:

- Client: a host which would request online resource (e.g. HTTP request asking for a package)
- Server: a host which has the resource that the client requests for
- Switch: an SDN switch which is responsible for forwarding the packets
- Load Balancer: an SDN switch which rewrites the IP and MAC addresses of the packet header and forwards the packet
- ONOS controller: a remote controller which monitors the network status and installs the flow rules to switches to instruct their packet forwarding process

2.3 Service Behavior Description

Among the five entities, ONOS controller is in charge of managing the communication process among other entities. Since the load balancing service is about balancing the HTTP traffic in the network, the ONOS controller cares mainly about the HTTP sessions and the messages exchanged on the underlying transport layer. Since the HTTP request has a virtual IP address, ONOS controller also needs to take care about the ARP triggered in the process. A chronogram of message-exchanges of a HTTP session is

shown in figure 2.3. This figure shows the case of the left topology in figure 2.2 where the Load Balancer directly connected to the group of servers.

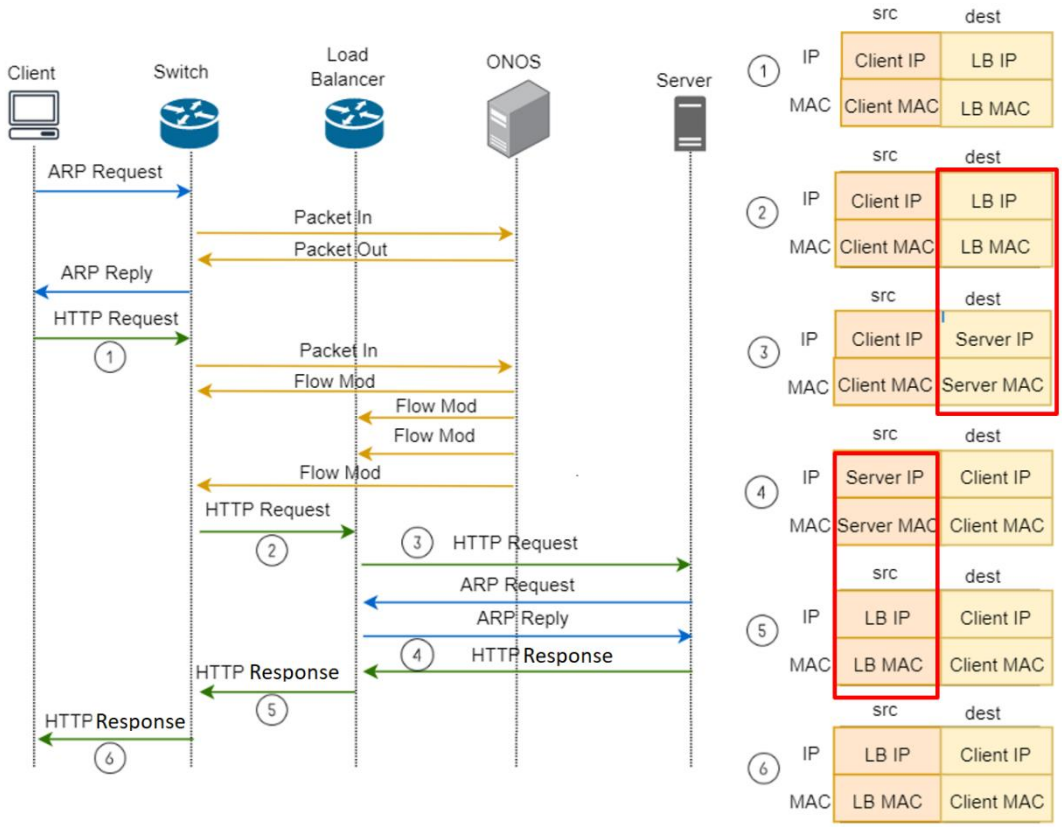


Figure 2.3 Chronogram of message-exchanges of service

The message-exchange process in figure 2.3 can be described as following steps:

- An ARP request is triggered first asking about the virtual public IP address
- ONOS controller takes the request from the edge switch and builds an ARP reply back to the client
- An HTTP session then starts by sending an HTTP request asking for a package resource
- ONOS controller gets the request from the edge switch and selects a server for this session based on network conditions and finds a routing path from the edge switch to the Load Balancer and the output port of the Load Balancer to the server, then pushes the flow rules to each switch along the path, and same for the reverse routing back
- On the data plane, the edge switch forwards the request to the Load Balancer based on the flow rules
- Load Balancer rewrites addresses of request header as table 3 shows on the right side of the figure and forwards it to the target server
- Upon receiving the request, the server sends ARP request asking about the client and then responds back. The ARP request is handled by *proxyarp* application activated in ONOS controller
- Load Balancer rewrites the response header address back as table 5 shows and then forwards the response back to the client based on the flow rules

Figure 2.3 illustrates a typical case when the Load Balancer directly connected to a group of serves. Some other cases are also possible and considered in this project:

- Neither clients nor servers are directly connected to the Load Balancer
- Clients directly connected to the Load Balancer
- Both clients and servers are directly connected to the Load Balancer

Although the latter two cases are not common in real case, it still worth consideration to enable the service to support more topologies. The message-exchange processes of these cases are similar. The only difference is whether the routing path is needed to be found between the Load Balancer and the edge switch. The flow rules would correspondingly be different.

Based on this design with the controller, the load balancing service should work in an expected way.

On the control plane:

- controller can answer the ARP request from clients
- controller can select a server in a load balancing way
- controller can know the network changes and update flow rules accordingly

On the data plane:

- requests with virtual IP address are able to arrive at the Load Balancer
- Load Balancer is able to rewrite the request/response header and forward the request/response to the selected server/client
- Switches can forward the packet along the selected path

3 Project Implementation

The implementation of the project is based on some services ONOS provides. As the chronogram shows in previous section, the project implementation is mainly about ARP handling and HTTP handling. Further details are discussed in this section.

3.1 Implementation Overview

ONOS offers various kinds of services for its controller to manage the network components. The services used in this project and their functions are listed in table 3.1.

Table 3.1 ONOS services used in the project [2]

Service	Function
Packet Service	Add/Remove the processor to the list; Punt the matching requests from the data plane to the controller; Emit ARP reply
Flow Rule Service	Remove all rules submitted by the application

Flow Objective Service	Install the forwarding flow rules to the selected device
Core Service	Register the application
Host Service	Offer the host information
Device Service	Offer the device information
Topology Service	Offer the network current topology information

The main functions of the application are achieved by overriding the process method in reactive packet processor added to the controller. Figure 3.1 is a flow chart showing how the Inbound packet is parsed in the packet processor in this application.

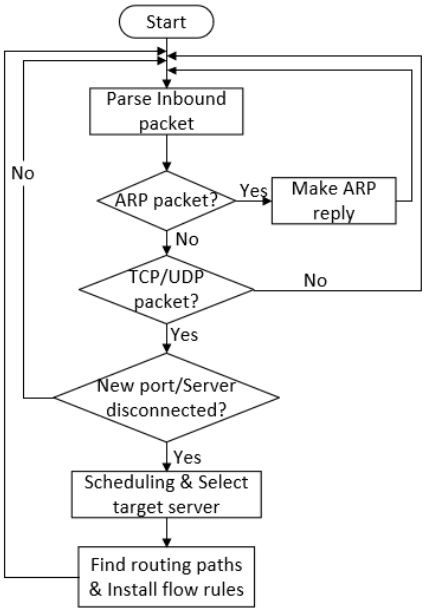


Figure 3.1 Inbound packet parsing process

Since the ARP packet and TCP/UDP packet are the ones that the controller really interested in, the controller will firstly parse the packet and check its protocol. If it's an ARP request packet and it comes from the client side, the controller will build and emit an ARP reply back to the client. Or, the controller will let the *proxyarp* application activated in controller to handle the request. If the packet is a TCP/UDP packet with a new source port number or the port number doesn't change but a server is detected disconnected from the group of servers, the controller will take care of this packet and select a new server as well as finding routing path and installing flow rules for each switch along the path.

3.2 ARP Handling

In this project, the *proxyarp* application is kept activated in ONOS controller. As shown in chronogram in figure 2.2, ARP process would be triggered from clients or servers. The controller would take the ARP packet and look at its source MAC address. If it is a server MAC address, the controller would leave it to the *proxyarp* application to reply it. If it is from the client side, the controller would build an ARP reply with the virtual

MAC address and virtual IP address based on the Ethernet payload of the ARP request. The building process is done by the *buildArpReply* method in ARP class offered in ONOS. Then an outbound packet is defined for this reply and emitted from the controller towards the client who sent the request.

3.3 HTTP Handling

The main job for this service is to balance the HTTP traffic in the network. To do this, scheduling process to select a server and routing process are needed during HTTP handling process. Their implementations are discussed in this section.

3.3.1 Server Selection

Having known a new HTTP session starts or a server is disconnected from the group of servers, a scheduling process is triggered to select a new server providing resource to the client. Since the service needs to support dynamic topology, the real-time location of the servers needs to be updated. In this way, a server lookup table is created and updated before the server selection process.

1. Server lookup table update

The server lookup table is a nesting Concurrent HashMap which records the locations of all servers in the network. Table 3.2 shows an example of the lookup table when all servers are connected with a same switch.

Table 3.2 Server lookup table format

Server MAC	Device ID	Port
Server 1 MAC	Switch 1 ID	Port 1
Server 2 MAC	Switch 1 ID	Port 2
Server 3 MAC	Switch 1 ID	Port 3

The server lookup table need to be updated in case that the server changes its location. It is possible that the server changes to another port or another switch. To handle this kind of dynamic changes, figure 3.2 shows the lookup table updating process. It is similar to the learning switch process. Whether the server exists in the network will be checked first. If it doesn't exist but the table still contains it, it will be removed from the table. Then the table will be checked if it has record of the server. If it doesn't, then an empty location table for this server will be added in the table. The switch ID will then be checked. If the location table is empty, the location information of this server will be added. If the server already has its location recorded, the port will be overwritten no matter whether the server changes its port or not.

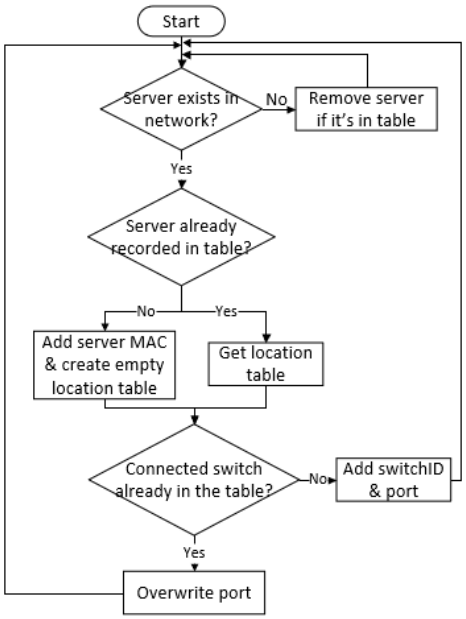


Figure 3.2 Server lookup table updating process

2. Server selection

Once having an updated server lookup table, a selection criterion needs to be followed to select a server. It is expected to be the one that would decide based on the network status, for example, the bandwidth of the link connecting to the server. However, the link bandwidth information cannot be found in the ONOS controller version 1.13.0. Thus, an alternative solution is considered which is based on the bytes sent statistics of the port connecting to each server. These statistics are assumed as the traffic load sending to server, which can also reflect the real-time link information to server. The sent bytes information is got from *bytesSent* method offered in *getStatisticsforPort* function in Device Service in ONOS. Regarding the selection method, two other classical schemes are also considered and compared here: Round-robin and Weighted Round-robin.[3] The pros and cons of these three methods are listed in table 3.3.

Table 3.3 Pros and cons of three kinds of selection method

Scheme	Principle	Pros	Cons
Round-robin	Equally consider each server and select one in order	Simple implementation	Not consider real-time network status
Weighted Round-robin	Weight is defined by the link bandwidth to server, the one with the highest weight is selected in order	Server connecting to larger link bandwidth has higher chance	Hardcoded link bandwidth
Port bytes-sent-based Selection	Collect the port bytes sent information of servers and select the one with the least bytes sent	Consider real-time link status	Not consider the limitation of server link bandwidth

To enable the load balancing service to work in a dynamic network, port bytes-sent-based selection method is implemented in the project. The port bytes sent information need to be collected for each server. Take the flat topology network in figure 2.1 as an example, the ports circled in yellow as in figure 3.3 are the source port of the link to the servers. These are the ports where the bytes sent are collected so that it's possible to estimate the load sending to server. The bytes-sent values are saved in a Concurrent HashMap "port statistics table" with the server MAC address as the key.

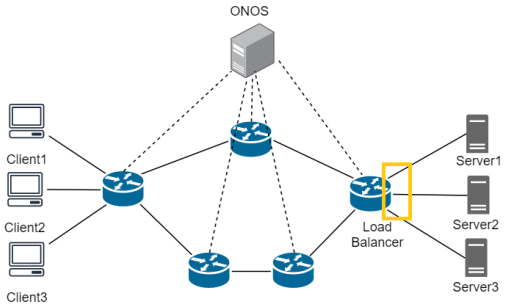


Figure 3.3 Ports where the byte sent collected from (circled in yellow)

Having got the bytes sent information for all servers, it's now possible to select a port with the minimum bytes sent number. To utilize all server resources, the repeated selection of a server is not wanted. In this way, a repeated selection avoiding process is run in this project. There could be a case that there is another server has the same minimum bytes-sent value as the repeated selected one. If this occurs, the repeated selection will be changed to another server with the same minimum bytes-sent value. After this process, the selection result will be returned in the form of Concurrent HashMap with the device ID and the port number that the selected server connected to. The process to handle the repetition is shown as flow chart in figure 3.4.

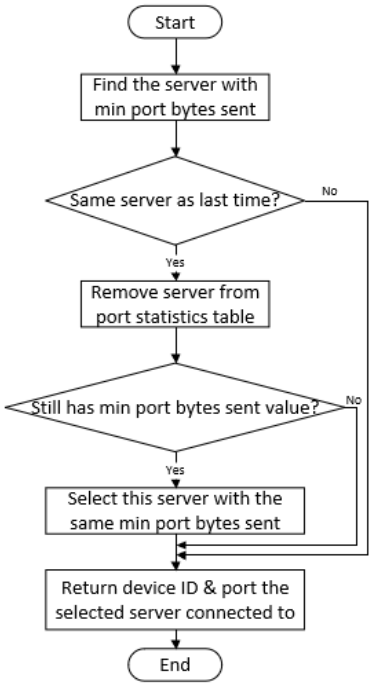


Figure 3.4 Repeated selection avoiding process

3.3.2 Routing Path Selection

When the target server is decided, the routing paths for both directions need to be found so that the controller could instruct the switch to forward the packet. ONOS controller version 1.13.0 offers the service to find a shortest path between two switches or hosts, but it cannot take the link status into consideration. Thus, a routing path selection criterion based on the link load is defined in this project.

1. Routing path selection

It won't be a problem if there is only one path between two devices. Take the hierarchical topology network in figure 2.1 as an example, there is only one path between the edge switch and the Load Balancer. However, if the network is like the flat topology network in figure 2.1, it would be a problem to choose a path from edge switch to Load Balancer since there are two paths available. To solve the problem, two steps are needed: find all available paths and select a path. To balance the traffic in the network, the selected path should be the one with the lowest link load among all paths.

Step 1: Find all available paths

Finding all available paths is based on *getK-ShortestPaths* method offered in ONOS controller. The method uses k-shortest path routing algorithm to find a stream of k-shortest paths between two devices in the current network topology. To make the result callable in other parts of the codes, the result of the stream is converted to a set of paths.

Step 2: Select a path

In this project, the link load is estimated by the bytes received statistics at the destination port of the link. The received bytes information is got from the *bytesReceived* method offered in *getStatisticsforPort* function in Device Service in ONOS controller. For all links in each path, the link with the largest load (i.e. most bytes received) is regarded as the "bottleneck" of this path. Knowing the "bottleneck" for each path, it's possible to select a path with the minimum "bottleneck". Figure 3.5 illustrated the implementation of this path selection process. A Concurrent HashMap is used to record the statistics data for each path.

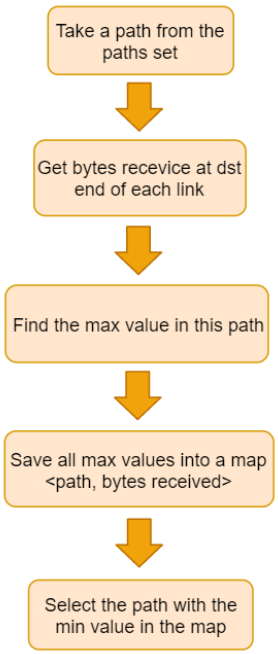


Figure 3.5 Implementation flow of path selection process

2. Flow rule installation

Having selected a server and routing paths, the only thing left for the controller is to install the flow rules to the switches to instruct them forwarding the packets along the selected paths.

Take the two topologies in figure 2.2 as an example, the green lines marked in figure 3.6 are the forwarding process on the data plane that need to be taken care of. The flow rules need to be installed into the source switch of each link to forward the packet.

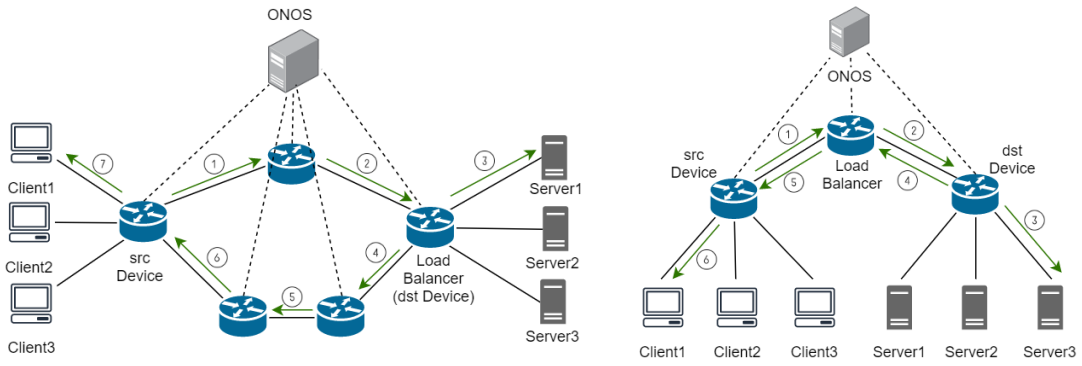


Figure 3.6 Forwarding process on the data plane

The location of the Load Balancer would define how the flow rules would be. The network on the right side of the figure shows a common case when the Load Balancer is neither the source device nor the destination device. Six methods corresponded to these 6 link steps marked in the right network are defined in codes to build and install flow rules. If the network gets complex, it's possible that the response has a different routing path from request, so the forward and backward process are considered

separately. Table 3.4 illustrates the functions of methods. The flow rule definition for edge switch need to be taken care specifically. There could be case like the network on the left side in figure 3.6 where the destination edge switch is the Load Balancer. It could also be possible that the source switch connected to the clients is the Load Balancer. To enable the edge switch to act as a Load Balancer, selectors and treatments for rewriting header are also defined in both Start edge and End edge method.

Table 3.4 Methods defined for flow rule installation

Forward Method	Function
Src device to LB	Forward request to LB along the selected path
LB to dst device	Rewrite request header; forward request to dst device along the selected path
End edge	(Rewrite request header;) Forward request to target server
Dst device to LB	Forward response to LB along the selected path
LB to src device	Rewrite response header; forward response to src device along the selected path
Start edge	(Rewrite response header;) Forward response back to client

Table 3.5 lists the details of the selectors and treatments defined in flow rules in End edge method. The first rule is defined in case that the switch acts as a Load Balancer while the second rule is for an edge switch only doing the forwarding. Having these defined, a forwarding objective with each selector and treatment pair is then built to generate and push the flow rule as a whole. The app ID is this load balancing application ID, the priority is set to 100 and the timeout is set to 10 so that the flow rule can be cleaned by the controller after timeouts to avoid a really large flow table.

Table 3.5 Selector and treatment defined in “End edge” method

Device	Selector	Treatment
Dst device (LB device)	**Ethernet Type:** IPv4; **IP Protocol:** TCP; **TCP src port:** src port; **TCP dst port:** dst port; **IP src:** Client IP; **IP dst:** LB IP	**Eth dst**: server MAC address **IP dst**: server IP address; **Output**: port connected to server
	Ethernet Type: IPv4; **IP Protocol:** TCP; **TCP src port:** src port; **TCP dst port:** dst port; **IP src:** Client IP; **IP dst:** server IP	**Output**: port connected to server

This table is an example of flow rule definitions. Other definitions are similar to this one. The device could be a link source device, an edge device or the Load Balancer. The IP addresses in the selector could be different depending on the role of the device.

Correspondingly, the treatment could be different in whether the header need to be rewritten and which port to output the packet.

4 Project Testing Strategy

4.1 Testing Tools

To test the load balancing service in this project, Mininet is used to build the network and Python is used to run servers and clients.

1. Mininet
   Mininet is used to create a network with clients and servers. Different network topologies can be created and the link bandwidth can be defined. When a network is built, it's possible to dynamically put a link down/up and change host port from the Mininet CLI, which enables the realizations of testing strategy discussed in the following part.
2. Python
   Python scripts are created to run a client-server scenario. The server is built by running a SimpleHTTPServer.py script offered in Python. A package with specific size is created by running Linux command: `dd if=/dev/zero of=./100MB.zip bs=1M count=100`. This package is the resource that a server could offer to the client. To enable a host in the network to act as a client, a python script is run to send HTTP requests asking for package resource at the virtual public address. A single request is coded like: `requests.get(url='http://10.0.0.100:8000/100MB.zip')`. The python script with requests is run with Linux *time* command to measure the script execution time. This execution time is regarded as the total time it takes for the client to send requests and get response from server, which is used to test the load balancing performance.

4.2 Dynamic Topology Testing Strategy

Since the design is based on the knowledge of Load Balancer and the group of servers information, assumptions are made for the testing as following.

Table 4.1 Assumptions made for testing

Server 1 MAC address	00:00:00:00:00:05
Server 2 MAC address	00:00:00:00:00:06
Server 3 MAC address	00:00:00:00:00:07
Load Balancer Device ID	of:0000000000000001

Since the location of the Load Balancer matters, to test if the service can support various network topologies, three kinds of Load Balancer locations need to be tested:

- Servers directly connected the Load Balancer
- Neither clients nor servers are connected directly to the Load Balancer
- Clients and servers are both connected directly to the Load Balancer

Thus, three basic network topologies are created in Mininet as shown in figure 4.2. If the Load Balancer can support these basic topologies, then it won't be a problem to support topology with more switches added into these networks.

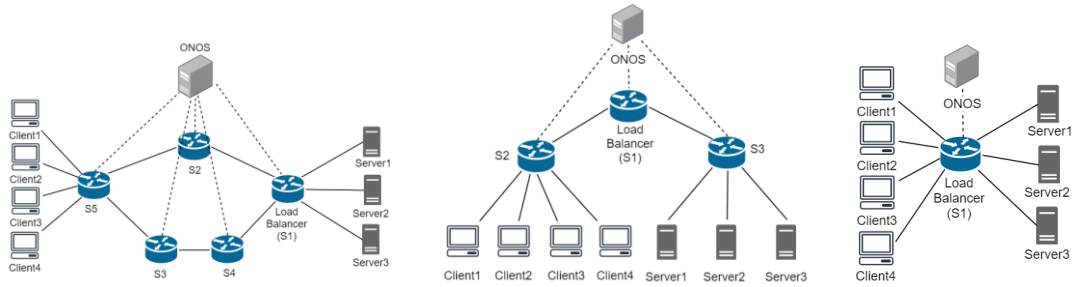


Figure 4.2 Network topologies for topology testing

To test if the service can be resilient to network dynamic changes, the testing is based on the topology in figure 4.3. Two switches are added in between the servers and Load Balancer so that it's easier to run commands in Mininet CLI to change the connecting port for a server or put down a server by putting down the link between the switch and server.[4] With the client keep sending requests, dynamic changes are done one by one. If the service can still work without any interruption of packet transmission, it means the service can be resilient to these dynamic changes.

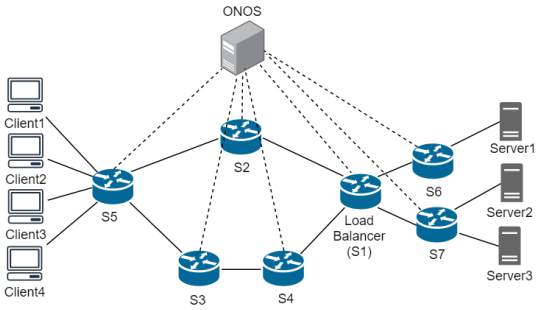


Figure 4.3 Topology used for dynamic change testing

4.3 Load Balancing Performance Testing Strategy

The idea of assessing the load balancing performance is to add various traffic load to the network so that it's possible to inspect how the service would behave under different load conditions. To test concurrent requests handling ability of the Load Balancer and enlarge execution time result, a round of requests from the clients are set up like this:

- 4 clients send requests at the same time
- each client sends 5 serial requests in one round
- requests ask for packages with size of 200MB/150 MB/100 MB/50 MB

With this kind of testing design, the load balancing performance can be assessed by the execution time of Python script run in client terminal under different network load conditions. To make sure the routing path selection also works in a load balancing way as designed, the traffic flow also needs to be inspected during the request handling process from the ONOS CLI.

To clearly see whether the load balancing service can perform well as expected, it needs to be compared with the request handling performance when no Load Balancer is deployed in the network. Also, to assess the performance of the server selection method used in the project, it's necessary to compare it with the performance of other methods Round-robin and Weighted Round-robin listed in table 3.3. These three cases are implemented in a simple way as following:

- No Load Balancer: all requests are forwarded to server 2
- Round-robin: Define a counter to count the requests. If the counter can be divided by 3, forward the request to server 1. If it has a remainder of 1, forward the request to server 2. In other cases, the request will be forwarded to server 3.[1]
- Weighted Round-robin: A flow chart illustrating the Weighted Round-robin implementation is shown in figure 4.4. The weight for each server is initialized based on the hardcoded link bandwidth between the server and the switch. If the link bandwidth is the same, then the weights are assigned with same values.

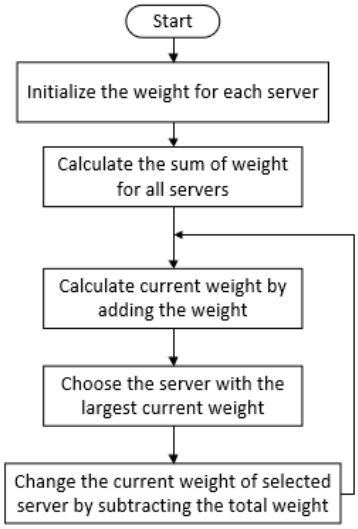


Figure 4.4 Weighted Round-robin scheduling process

In real case, sometimes the Load Balancer is deployed because of the limited link resource to the servers. To test the performance of the design under different link bandwidth resources, the bandwidth of link connected to servers in network topology in figure 4.3 are set to the following three cases:

- Case 1: Server 1 - 1000Mbps; Server 2 - 1000Mbps; Server 3 - 1000Mbps
- Case 2: Server 1 - 500Mbps; Server 2 - 500Mbps; Server 3 - 500Mbps
- Case 3: Server 1 - 800Mbps; Server 2- 500Mbps; Server 3 - 300Mbps

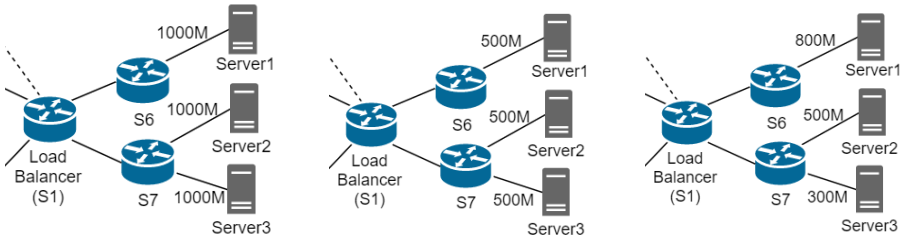


Figure 4.5 Bandwidth settings (Left: case 1; Middle: case 2; Right: case 3)

5 Results Analysis

5.1 Dynamic Topology Testing Results

By running the load balancing service in networks shown in figure 4.2 separately, the HTTP traffic can be balanced properly in these network topologies. Based on the network in figure 4.3, some dynamic network changes are done one by one in Mininet CLI. Figure 5.1 shows an example scenario of port change. By running several configuration commands in Mininet CLI, the server 2 (h6) has been changed its connection from S7 to S6 (circled in red). Having the controller detected the existence of new attached sever 2 on S7, the controller starts to select this server to respond to requests. By conducting the dynamic changes with requests from a client kept sending, the service can be resilient to the changes listed below without interruption of requests handling process.

- Link between server and switch down/up: e.g. link between S7 and server 2
- Link between 2 switches down/up: e.g. link between S5 and S2
- Port changing of a server: e.g. server 2 changes connection from S7 to S6

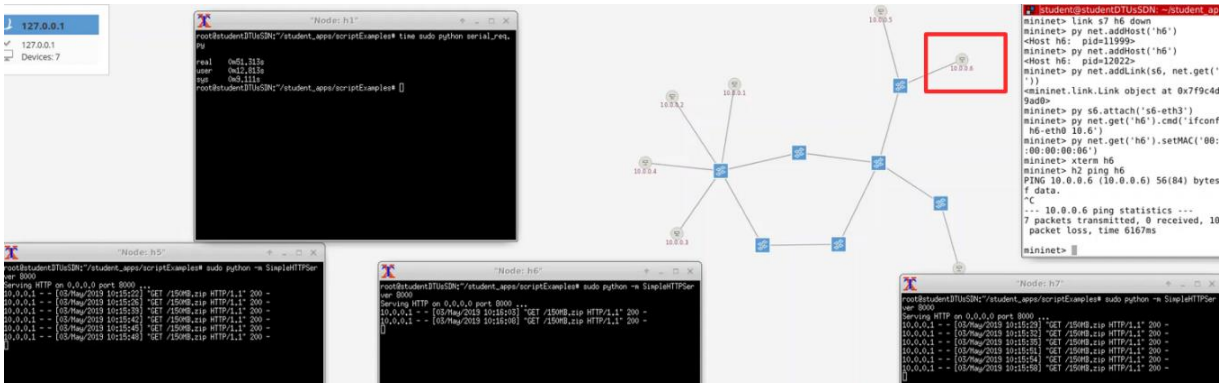


Figure 5.1 Server port changing test result

Based on these results, the load balancing service implemented in this project can support basically any topology with the knowledge of Load Balancer Device ID and the MAC address of servers and can be resilient to dynamic changes in network.

5.2 Load Balancing Performance Testing Results

Figure 5.2 shows the traffic flows in the network inspected from the ONOS CLI during the request handling process. Two requests from client 1 are handled separately by server 1 and server 2. The traffic flows on both paths between S5 and S1 show that the routing path selection method works. It proves that the controller can select a relative underutilized path for the packet transferring process.

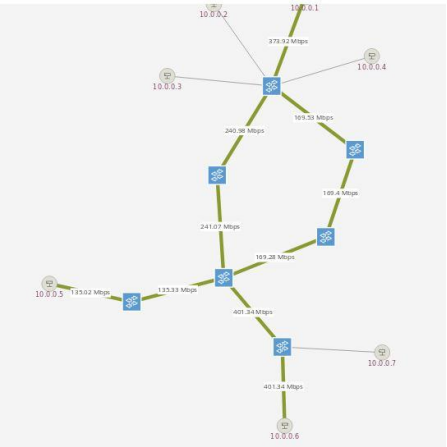


Figure 5.2 Traffic flows when 2 requests sent from client 1

Figure 5.3 is the screenshot after 4 clients sending 5 requests asking for 150MB packages at the same time with bandwidth setting in case 1 (1000Mbps). It's clear to see that the traffic requests are handled evenly by three servers (h5, h6, h7). The load balancing performance is assessed based on the execution time results in clients' terminals (circled in yellow). The shorter the execution time is, the better the Load Balancer performs.

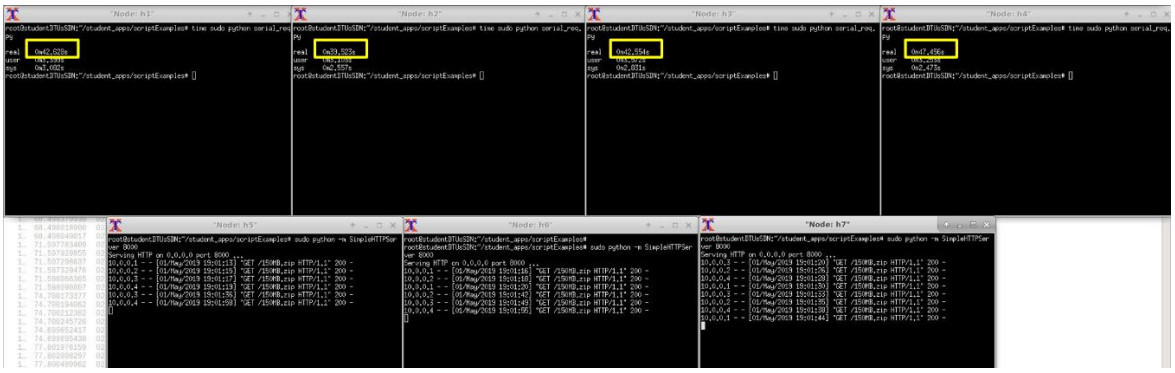


Figure 5.3 Testing result with bandwidth setting in case 1 (1000Mbps)

Figure 5.4 shows the comparison of load balancing performances when the links to the servers have sufficient bandwidth 1000M (as in left setting in figure 4.5). Under this case, three kinds of server selection method perform similarly. The weighted Round-robin acts the same as Round-robin here since the link bandwidths are the same. The port bytes-sent based balancer shows a slightly better performance compared to these two. When there's no load balancing process involved, the request handling process would fail when the request is asking for large package resources.

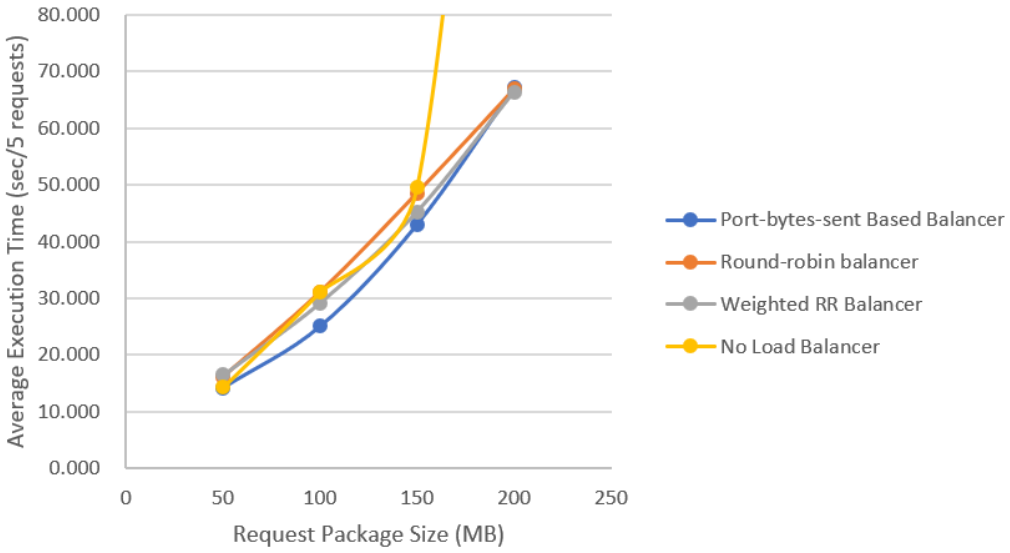


Figure 5.4 Load balancing performances under sufficient link bandwidth resource

Figure 5.5 shows the result when the link bandwidths are cut half down from the last case (as in middle setting in figure 4.5). With these kind of relatively limited link resources, the network without the load balancing service shows a poor performance on requests handling, especially the requests for large package resources. While the other three Load Balancers with different selection methods show similar performances.

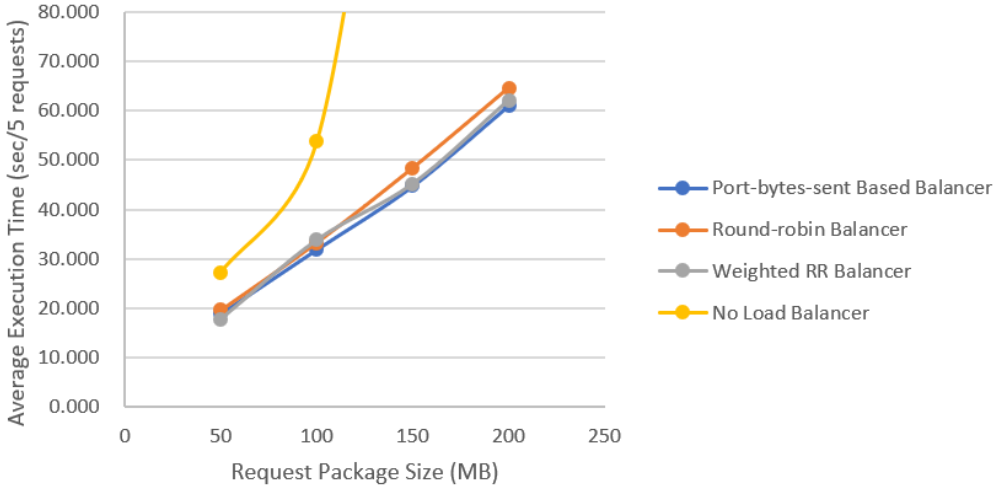


Figure 5.5 Load balancing performances under limited link bandwidth resource

Figure 5.6 shows the case when each server has different link resource. In this case, the Load Balancer with Weighted Round-robin shows its advantage since it takes the link bandwidth into consideration. The one with the larger bandwidth would have a larger weight and would stand a higher chance to be selected. The port bytes-sent based Load Balancer proposed in this project still has a good performance when package size is not close to link limit.

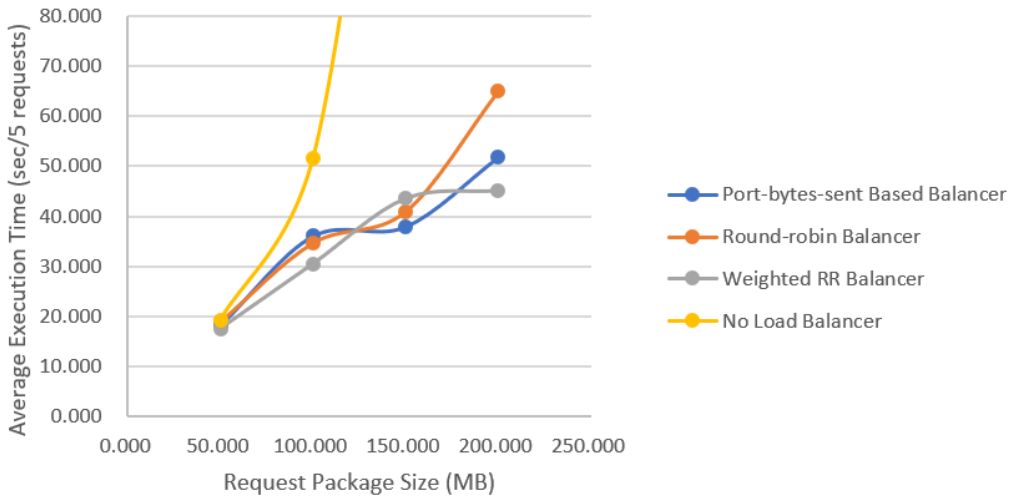


Figure 5.6 Load balancing performances under diverse link band width resource

By comparing the results of three different kinds of link bandwidth scenario, the performance of Load Balancer implemented in this project can be assessed. In all cases, the network with a Load Balancer would have a faster request handling process than the one without any Load Balancer. Although the port bytes-sent based balancer is an alternative solution for the link bandwidth monitoring solution, it shows a good performance under these link bandwidth scenarios. When the link resources are sufficient, the Load Balancer proposed in this project shows slightly better performance for shortening the request handling time. While for the network with equally limited resources, which would be the case where a Load Balancer is needed, it works as well as the other two kinds of selection method. Plus, it can work in a dynamic way by considering the real-time status as discussed in table 3.3. For the network with uneven link resources, which would also be a case that a Load Balancer is needed, Load Balancer implemented in this project didn't perform as well as Weighted Round-robin Balancer when the request is asking for a large package. This is because the port bytes-sent based selection method doesn't know the limitation of link bandwidth. However, it still shows a pretty good performance when requested package size does not reach the link limit. Also, it still has its advantage in dynamic handling than the hardcoded Weight Round-robin method.

6 Critical Conclusion

This project implements a load balancing service in SDN by introducing ONOS controller with OpenFlow as southbound interface to take control over the data plane. Based on the programmability and flexibility the SDN solution offers, the service can support basically any network topologies with the knowledge of servers' MAC address and Load Balancer device ID. It can be resilient to possible dynamic network changes comparing to traditional hardcoded Load Balancer. The server selection criterion is based on the port bytes-sent statistics of links connecting to servers. This method is

proposed as an alternative solution for monitoring the link bandwidth to do the selection. Based on the testing results, this alternative solution can work well under dynamic network scenarios and perform well with limited link bandwidth resources. The load balancing performance testing results demonstrate that the server load balancing service implemented in this project can balance the HTTP traffic in network and shorten the HTTP request handling time by efficiently using the network resources.

6.1 Future Work

The server selection criterion implemented in this project is aiming at selecting the target server based on real-time network conditions. This part of design is kept in a simple way. Regarding this, two potential future work could be considered:

- Regarding the port bytes-sent statistics, now the statistics are collected at the time when the selection method is called. The statistics result would be an accumulative byte sent value. It could also be considered about adding a time window to the statistics collecting process. By doing this, it may show a more straightforward result on traffic changes arriving at servers.
- Regarding the selection criterion, the current solution doesn't consider the capability of servers. A possible selection method could be based on Weighted Round-robin and assign different weight to the servers according to their CPU abilities. The tests could be done by defining the server as CPU Limited Host in Mininet.

References

[1] 34359 Software Defined Networking, Angelos Mimidis & José Soler, Programming network services with ONOS slides, March 2019
[2] All classes in ONOS Java API (1.13.0), http://api.onosproject.org/1.13.0/allclasses-noframe.html
[3] LVS Scheduling Overview, https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Virtual_Server_Administration/s1-lvs-scheduling-VSA.html
[4] Dynamic Topology Changes in Mininet—Advanced Users, December 2016, https://kiranvemuri.info/dynamic-topology-changes-in-mininet-advanced-users-5c452f7f302a

Appendix

Load balancing performance testing result: raw data

Note: the execution time for failed executions are estimated and recorded as the first timeout occurred in the client terminals

Table 1 Execution time (sec/5requests) in testing case 1

		Package Size (MB)			
		200	**150**	**100**	**50**
Port bytes-sent based	Client 1	65.073	42.628	24.143	14.782
	Client 2	74.175	39.523	25.872	12.314
	Client 3	66.775	42.554	27.552	15.529
	Client 4	62.535	47.456	22.696	14.307
	Average	67.140	43.040	25.066	14.233
RR	Client 1	62.496	51.689	31.957	14.494
	Client 2	65.240	49.958	29.917	16.762
	Client 3	71.360	41.769	31.750	14.714
	Client 4	68.492	51.010	30.956	18.829
	Average	66.897	48.607	31.145	16.200
WRR	Client 1	62.416	43.710	30.209	15.334
	Client 2	66.563	46.006	28.297	18.090
	Client 3	69.124	48.197	30.228	17.119
	Client 4	67.298	43.115	28.023	15.422
	Average	66.350	45.257	29.189	16.491
no LB	Client 1	200	52.224	32.001	14.721
	Client 2	200	50.404	32.947	14.505
	Client 3	200	48.549	30.420	14.230
	Client 4	200	47.147	29.352	13.980
	Average	200	49.581	31.180	14.359

Table 2 Execution time (sec/5requests) in testing case 2

		Package Size (MB)			
		200	**150**	**100**	**50**
Port bytes-sent based	Client 1	68.752	42.510	36.976	21.227
	Client 2	61.670	45.833	35.213	20.546
	Client 3	57.490	45.917	33.464	18.449
	Client 4	56.274	44.881	27.115	15.575
	Average	61.047	44.785	31.931	18.949
RR	Client 1	68.874	48.542	31.986	19.215
	Client 2	58.766	50.004	31.218	20.206
	Client 3	65.290	47.157	35.906	20.254

	Client 4	65.401	47.793	33.920	18.732
	Average	64.583	48.374	33.258	19.602
WRR	Client 1	68.483	43.027	34.925	18.472
	Client 2	53.127	48.591	37.430	17.442
	Client 3	63.880	42.899	32.679	18.374
	Client 4	63.264	46.434	30.358	17.095
	Average	62.189	45.238	33.848	17.846
no LB	Client 1	200	160	57.049	28.495
	Client 2	200	160	54.971	27.779
	Client 3	200	160	52.694	26.907
	Client 4	200	160	50.407	25.874
	Average	200	160	53.780	27.264

Table 3 Execution time (sec/5requests) in testing case 3

		Package Size (MB)			
		200	**150**	**100**	**50**
Port bytes-sent based	Client 1	54.735	32.416	37.892	19.808
	Client 2	50.014	38.883	37.626	15.403
	Client 3	45.780	34.271	33.350	18.099
	Client 4	56.779	46.232	35.372	19.381
	Average	51.827	37.951	36.060	18.173
RR	Client 1	66.303	44.205	34.972	18.789
	Client 2	56.828	39.333	35.973	16.404
	Client 3	76.463	40.029	31.646	20.581
	Client 4	60.291	40.217	36.043	19.797
	Average	64.971	40.946	34.659	18.893
WRR	Client 1	37.682	45.011	31.551	17.866
	Client 2	52.303	42.177	29.667	16.961
	Client 3	40.67	38.883	35.291	18.138
	Client 4	50.276	48.468	25.614	17.042
	Average	45.234	43.635	30.531	17.502
no LB	Client 1	200	160	55.758	20.131
	Client 2	200	160	52.933	19.573
	Client 3	200	160	49.952	18.970
	Client 4	200	160	47.609	18.279
	Average	200	160	51.563	19.238

Load Balancer Service

34359 Software Defined Networking

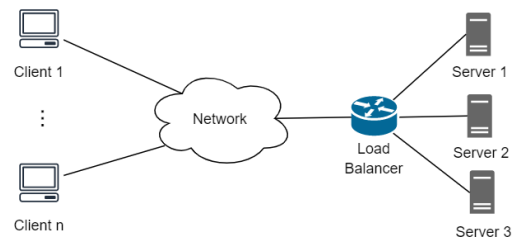
s171352 Mingyuan Zang

Supervisors: José Soler, Angelos Mimidis

DTU Fotonik
Department of Photonics Engineering

Server Load Balancer

- A group of servers have replica resource
- All clients send request to virtual public IP address
- Load Balancer distributes requests to relatively underutilized server



- Traditional LB:
  Hardcoded in a dedicated hardware device 😐
- SDN LB:
  Programmability & Flexibility 😃

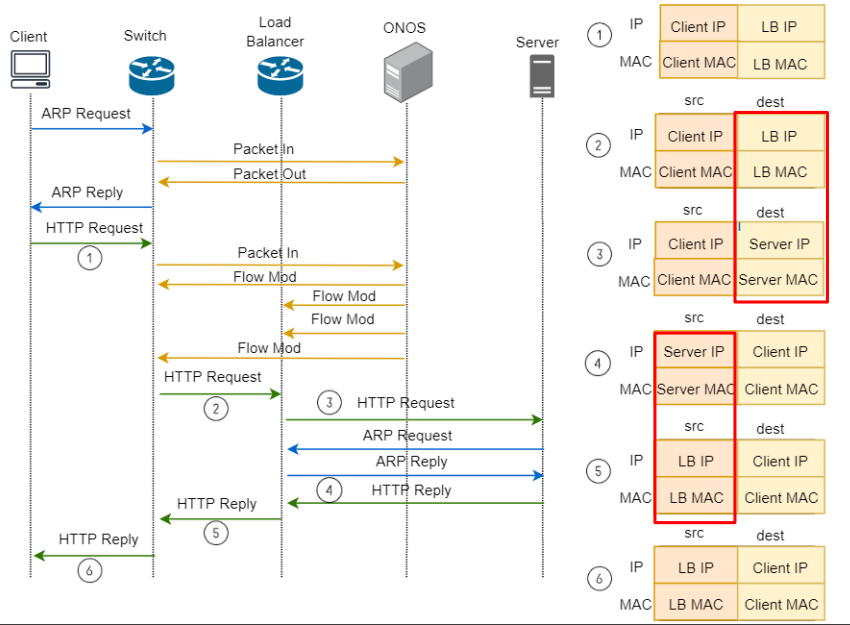
Objectives

- The service is able to work in more network topologies (Ideally any)
- The service is based on a more meaningful selection method (e.g. network conditions)
- Based on ONOS controller with OpenFlow as Southbound Interface

Load Balancer Service

- ✓ Able to support basically any topology with the knowledge of servers' MAC addresses and Load Balancer Device ID
- ✓ Resilient to network dynamic change (e.g. link down, server down, server change port)
- ✓ Server selection based on port bytes sent statistics
- ✓ Routing path selection based on link load

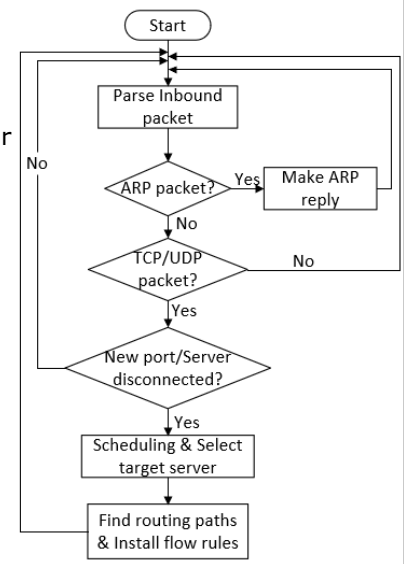
Service Behavior Description



Implementation Overview

Use ONOS services:

- Packet Service
  - Add/Remove processor to the list
  - Punt the matching packets to controller
  - Emit ARP reply
- Flow Rule Service
  - Remove all rules of app
- Flow Objective Service
  - Install flow rules to device
- Core Service
  - Register the application
- Host Service
- Device Service
- Topology Service

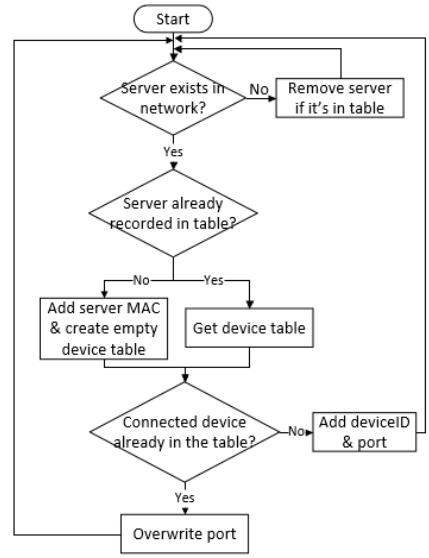


source: http://api.onosproject.org/1.13.0/allclasses-noframe.html

HTTP Handling: Server Selection

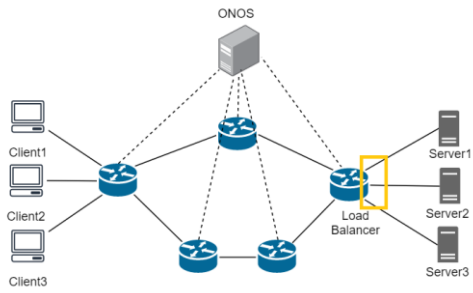
- Server Lookup Table:
Concurrent HashMap recording the server location

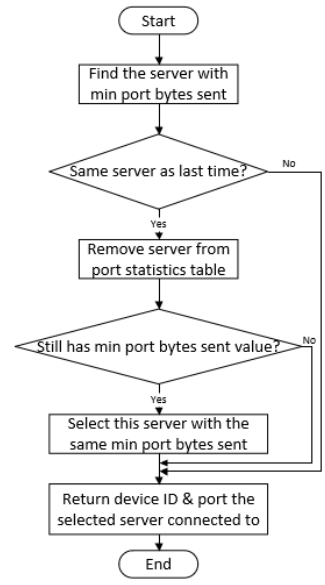
Server MAC	Device ID	Port
Server 1 MAC	Switch 1 ID	Port 1
Server 2 MAC	Switch 1 ID	Port 2



HTTP Handling: Server Selection

```
DeviceService.getStatisticsForPort(DeviceID, port)
             .bytesSent()
```





HTTP Handling: Routing Path Selection

- Step 1: Find all available paths

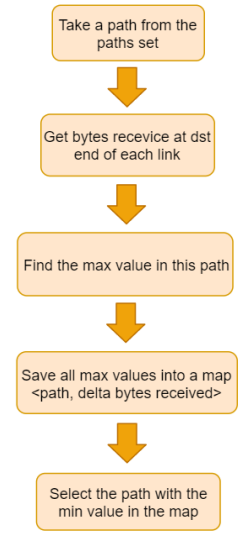
  `TopologyService.getKShortestPaths`

- Step 2: Select a path

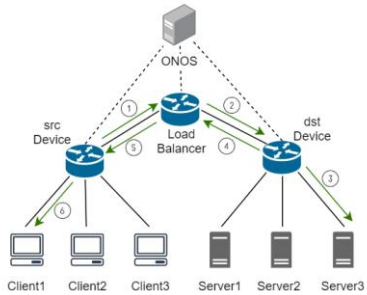
  Link with the largest load (i.e. most bytes received)

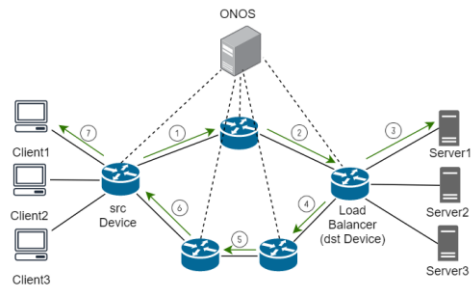
  => path "bottleneck"

✓ Select minimum "bottleneck"



HTTP Handling: Flow Rule Installation





Device	Selector	Treatment
Dst device (LB device)	Ethernet Type: IPv4; IP Protocol: TCP; TCP src port: src port; TCP dst port: 8000; IP src: Client IP; IP dst: LB IP	Eth dst: Server MAC address IP dst: server IP address; Output: port connected to server
	Ethernet Type: IPv4; IP Protocol: TCP; TCP src port: src port; TCP dst port: 8000; IP src: Client IP; IP dst: server IP	Output: port connected to server

Testing Strategy

- Testing tools:
  - ➢Mininet
  - ➢Python

  Server: SimpleHTTPServer

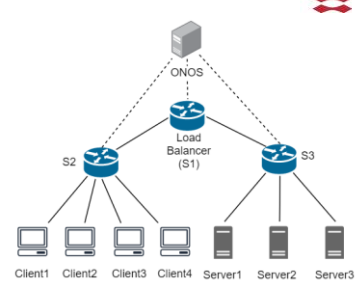
  Client: request = http://10.0.0.100:8000/file

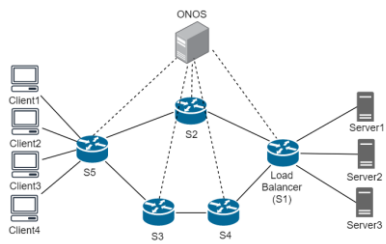
  time command: measure the script execution time
- Testing Assumptions:
  - Knowledge of MAC addresses of 3 servers
  - Load Balancer Device ID: of:0000000000000001
- Dynamic Topology Tests
- LB Performance Tests:
  - 4 Clients send requests at the same time
  - 5 serial requests from each client
  - Request for package with size of 50MB/100MB/150MB/200MB

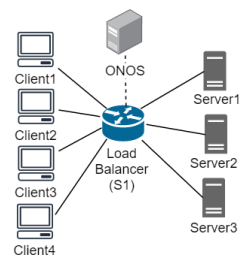
source: Lecture 6-Programming network services with ONOS

Dynamic Topology Testing

- ✓ Link between server and switch down/up
- ✓ Link between 2 switches down/up
- ✓ Port changing of a server

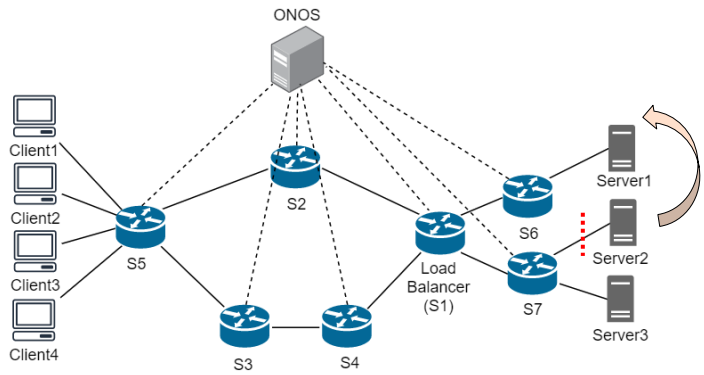




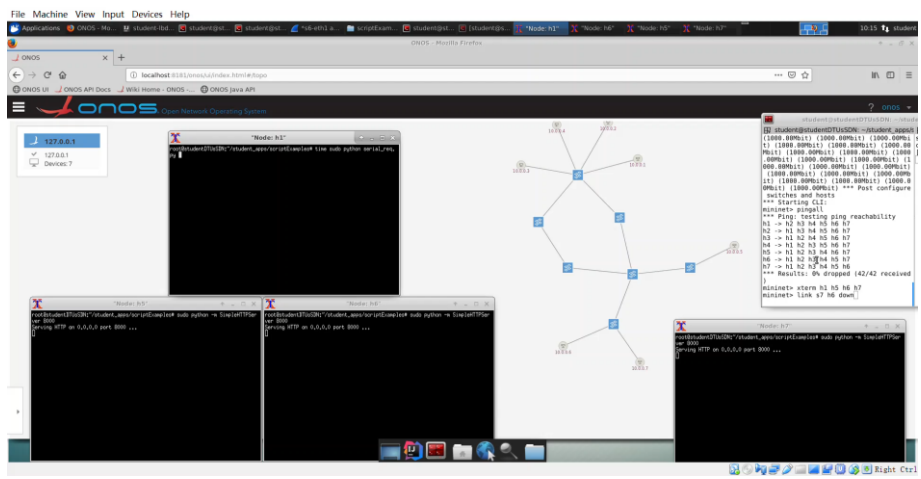


Dynamic Topology Testing

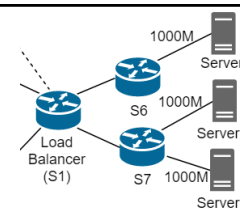
- ✓ Port changing of a server

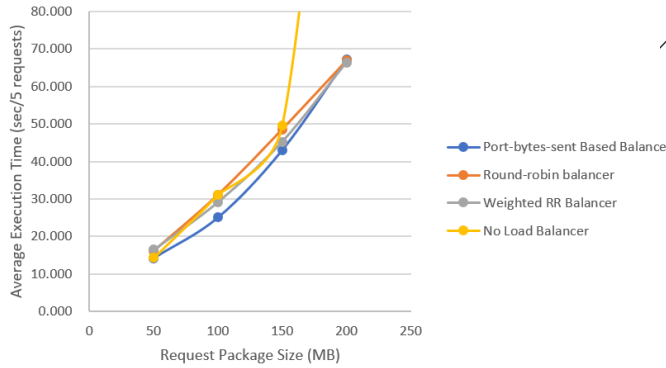


Dynamic Topology Testing



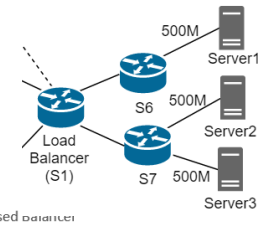
Load Balancing Performance Testing

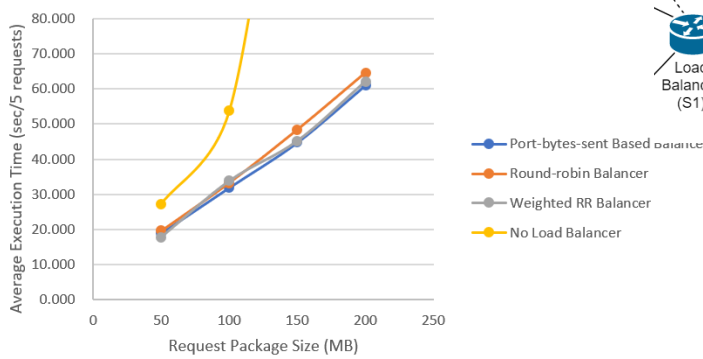




- When sufficient bandwidth resource:
  - Port-bytes-sent Based Balancer shows slightly better performance
  - Weighted RR acts the same as RR
  - No LB failed to finish when request for large package

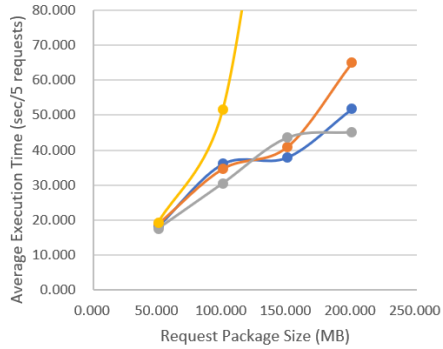
Load Balancing Performance Testing

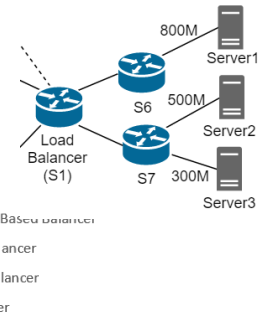




- When less bandwidth resource:
  - No LB shows poor performance
  - 3 scheduling schemes show similar performance

Load Balancing Performance Testing





- When less bandwidth resource:
  - WRR shows its advantage
  - Port-bytes-sent Balancer performs well when package size not close to link limit
  - No LB shows poor performance

Conclusion

- ✓ SDN provides the programmability of LB and flexibility with network conditions
- ✓ LB can speed up request handling process
- ✓ Service supports basically any topology with the knowledge of servers' MAC addresses and Load Balancer Device ID
- ✓ Service is resilient to network dynamic change
- ✓ Port-bytes-sent selection criteria can work in dynamic network and shows a good load balancing performance under limited link resource

Future Work

- Problem:

  Port bytes-sent based selection criteria doesn't consider the capability of servers

- Possible Solution: develop WRR
  - Define the weight based on CPU ability of server
  - Test in Mininet by defining server as CPU Limited Host

Q&A

Thank you