Multi-path Routing for Mission Critical Applications in Software-Defined Networks

Ramon Carreras Ramirez[1], Quoc-Tuan Vien[1], Ramona Trestian[1], Leonardo Mostarda[2], and Purav Shah[1]

[1] School of Science and Technology,
Middlesex University, London - NW4 4BT, UK
RC934@live.mdx.ac.uk
{q.vien, r.trestian, p.shah}@mdx.ac.uk
[2] Scuola di Scienze e Tecnologie,Universita' degli Studi di, Camerino, Italy
leonardo.mostarda@unicam.it

Abstract. Mission critical applications depends on the communication among other systems and/or users and thus, the traffic/flows generated by these applications could bring profound consequences in sectors such as military, hospital, automotive safety and air-traffic control systems. These critical flows require stringent QoS requirements on parameters such as throughput, packet loss, latency, jitter and redundancy. Network operators must have tools that allow them to provide special treatment to such mission-critical flows based on specific application requirements. Due to the constraints of traditional networks, we should seek for solutions supported by de-centralised approaches offered by SDN.
In this paper, we propose a solution to achieve the stringent QoS requirement of such mission critical flows in multi-path environments based on SDN. This solution allows the network operator to prioritise traffic between specific end points. Also, using the overall view of the network, the solution allows evaluation of the path loads between two endpoints and to opt for the less congested path. Moreover, this paper tries to demonstrate a satisfactory network performance by presenting trade-offs between throughput and the number of hops within a multi-path network. The proposed solution is implemented in the application and control layer of the OpenDaylight Controller. The networking devices were simulated using Mininet simulator and background traffic was generated using Iperf.

1 Introduction and Related Works

Mission-critical networks provide applications to control and monitor energy, railways, public-safety, aviation management, etc. The network thus must be reliable and resilient in order to support such applications where by in case of a network failure, peoples lives and companies can be at risk. Load balancing has been an existing problem in providing reliable network by assuring the system is not overloaded and also by high resource utilisation [1], [2]. With the emergence of the Software Defined Networking (SDN) paradigm which has been triumphed

as a reliable and resilient solution for future Internet, mission-critical networks can be improved in reliability, resilience and security by leveraging SDNs features. SDN promises to make the existing networks more easier to supervise, configure, deploy and monitor. The traditional networks are getting more complex by having a combined control and forwarding planes on the same device. Thus, the simplification of the SDN architecture by separating these planes and thus simplifying the communication using the standardised OpenFlow (OF) protocol [6] is a key to improving the network performance. A SDN controller configures the network elements by distributing the forwarding rules to the switches using low-level language.

In SDN, OpenFlow is a widely used protocol that enables the controller to communicate with OF-switches [6]. OpenFlow enables the controller to remotely install, modify, and even delete forwarding rules in OF-switchs flow table via the OpenFlow channel [7]. When a new flow enters the network, the OF-switch will encapsulate a new packet of the flow into a Packet-In message and send to the controller to ask for an appropriate behaviour. The controller then selects a forwarding path for the new flow based on the current global network status and sets up the forwarding path by using Flow Mod messages to reactively install rules on related OF-switches.

Several recent studies have focused on load balancing in the control plane. BalanceFlow [3] introduces an extension for OF-switch known as controller X. However, it comes at a cost of more complexity on the OF-switches and increased communication overhead at the control plane from using a periodical approach. In this scheme, the controllers need to publish their load information periodically via a cross-controller communication channel. In [5], a dynamic load re-balancing method based on switch migration mechanism for clustered controllers was introduced. However, the problem of cross-controller communication was not resolved. Thus, its implementation on large-scale networks is very limited. Furthermore, another dynamic and adaptive algorithm named DALB was proposed by Y. Zhou et al. [8], that forces the controllers to collect load from each other in case load on each controller exceeds a certain load threshold at the same time. This approach also causes high communication overhead in the control plane.

Motivated from these above challenges, this paper proposes an efficient scheme to balance the load on the network when considering mission-critical flows. The main contributions of this paper can be summarised as follows:

1. Firstly, by using multi-path routing, the proposed approach eliminates the single point of failure problem in the data plane and provides a reliable network, which is essential for SDN-based mission-critical applications.
2. Simulation results show that the proposed scheme achieves good load-balancing performance in the network as compared with previous schemes based on shortest path based routing.

The rest of this paper is organised as follows. Section II presents the proposed solution for multi-path routing and prioritisation. Section III analyses the

proposed solution and presents the a discussion on the results achieved from the simulation setup and finally, section IV finishes the paper with a brief conclusion and remarks for future work.

2 Proposed Multi-Path Routing & Prioritisation Solution

The proposed solution is implemented using the OpenDaylight SDN controller to meet three primary objectives:

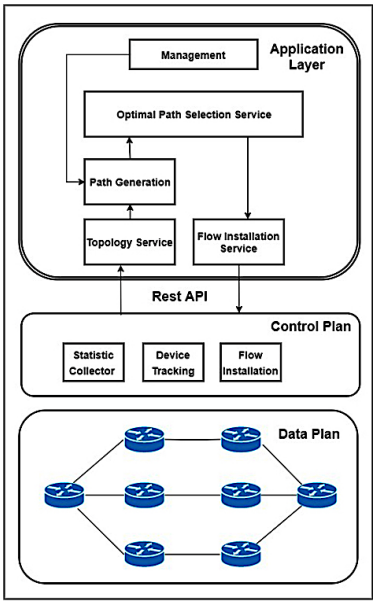
1. Provide prioritisation to selected traffic in a multi-path network environment
2. Using the SDN central management, choose the less congested path between two endpoints for the prioritised traffic
3. Demonstrate that in a multi-path environment, by considering path(s) with more hops but less congestion, it can achieve a satisfactory performance of throughput and packet loss.

2.1 Proposed Solution

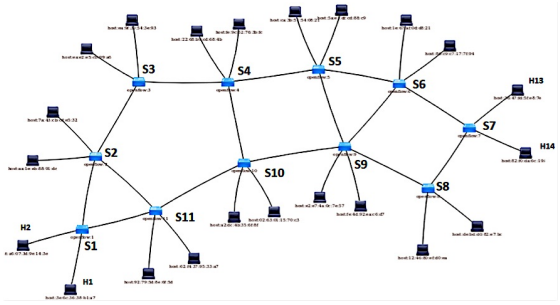
As shown in figure 1a, the proposed framework architecture consists of three layers: infrastructure layer, control layer, the application layer. The infrastructure layer or data plane consists of a set of OpenFlow Switch connected to each other to build a network topology. These switches are connected to the SDN controller; therefore, they are dynamically programmable by using southbound APIs. The control layer consisted of some internal modules of the OpenDaylight controller, those are Statistic Collector, Device Tracking, Flow programming. The top layer of the diagram is the application layer, this is the core of our proposed framework. In this layer, there are some modules developed for providing distinct roles, such as Topology Service, Path Generation, Optimal Path Selection Service, Flow Installation Service, and Management.

Control Layer In this layer, the internal modules of the OpenDayLight controller are utilised to collect all the network states (statistics collector). The traffic load information of each link is utilised to select the less congested path. The controller sends an FLOW STATS REQUEST to the switches to send information for collection. This message is replied with an FLOW STATS REPLY along with the information requested. This information is maintained in the controller and it will be consulted using REST API by the application layer. SDN allows three kinds of operational modes to setup a new flow rule (flow entry) namely proactive, reactive and hybrid modes [4], which makes it flexible to add/forward flow entries into the OF-switches.

OpenFlow controllers are able to collect statistics of a switch at different aggregation levels such as port, table, flow, and queue. In the proposed framework, we are using the port level statistics. Port-level statistics provides bytes counter (transmitted and received), packets counter (transmitted and received), packets dropped or errors occurred. The Device tracking module consists on discovery



(a) Proposed Solution Framework



(b) OpenDaylight Controller's Topology View

Fig. 1: Proposed Solution and Topology View

each device, ports, and links of the whole network. The information collected by this module helps the application plane to keep a global view of the network. The Flow Programming module is responsible for pushing the flow rules to the OpenFlow device by means of Southbound API.

Application Layer The main functionality of our proposed solution is located in the Application Layer.

User/Management - The algorithm starts at User/ Management service. Basically, the network operator/application establishes the source and the target of the traffic that must be prioritised. This can be done at port-level.

Topology Service - Topology service is responsible to obtain the entire network view and maintain this information in the application layer. This module sends the data collected to the Path Generation service for path computations.

Generation Path - Generation Path service is responsible to compute all possible paths between two endpoints. This module uses a modified depth-first search to generate the paths.

Flow Installation Service - This service is the last step of the proposed algorithm solution. When the optimal path selection service selects the best path and send the information to this service, it prepares the flow entries to be inserted in the OpenFlow switch and using the Southbound API installs new flows on the network devices.

Optimal Path Selection Service - This service is implemented in the application layer and is responsible to select the best path between two endpoints based on the availability and throughput of the paths. It receives the distinct paths to compute from the Generation Path module. Using the information from the statistics collector, it evaluate the load of each path, chooses the path with higher throughput availability and lastly sends the path elected to the flow installation service.

3 Simulation and Results

3.1 Experimental Setup

The simulation was set up on a Laptop with standard physical specifications of Intel i7 processor and 16GB RAM. The OpenDaylight SDN controller and Mininet simulator were setup using a Linux virtual machine named ODL-MIN. The OpenDaylight controller version used was Carbon (6^{th} release). The additional features that were required by the controller in order to run the simulation were customised according to the proposed algorithm and statistics monitor. For simplicity, we have not considered any background traffic, however, considering background traffic would significantly impact the network performance.

Topology We took the Abilene topology from TopologyZoo. This network was a high-performance backbone network created by the Internet2 community in the late 1990s. It comprises eleven nodes within USA. Abilene was selected since it is an ideal multi-path topology to test our proposed solution.

The network topology was simulated using Mininet simulator. This network topology as shown in figure 1b consists of one SDN Controller node, eleven (11) OpenFlow Switches and two (2) hosts connected to each switch. We used the host H1 and H2 as source nodes and H13 and H14 as destination nodes. The bandwidth of each link is limited to 100 Mbps from switch to switch and switch to host. From the sources and destination, we have multiples paths. The shortest path is Path 1, which has six hops. But as our solution considers all possible paths, we have multiple paths to get the destination with more hops. For our scenarios, we are going to consider the following paths as shown in table 1. These paths have between six and seven hops. Some links are part of more

Table 1: List of paths

Path No.	Path (Source to Dest)	Hop Count	Load/Traffic (Availability)
1	S7, S8, S9, S10, S11, S1	6	25Mbps
2	S7, S6, S9, S10, S11, S1	6	25Mbps
3	S7, S6, S5, S4, S3, S2, S1	7	100 Mbps
4	S7, S6, S5, S9, S10, S11, S1	7	25Mbps
5	S7, S6, S9, S10, S11, S2, S1	7	25Mbps
6	S7, S8, S9, S10, S11, S2, S1	7	25Mbps
7	S7, S6, S5, S9, S10, S11, S1	7	25Mbps

than one path; for example, paths 1 and 2 share the links between S10, S11, and S1. Iperf is used to generate traffic/load from sources to destinations. Iperf provides statistical information about the packet loss, throughput, and jitter from the traffic generated. This information was used to create the statistics report of the simulation. This tool allows generating traffic UDP or TCP. For our scenarios, we are using UDP traffic only for ease of purposes and since TCP traffic cannot provide packet loss information.The parameters used in the simulation are: Throughput = 75 Mbps, Simulation Time = 60 sec and Monitoring iteration time = 2 sec.

3.2 Test Case 1

This test case has two main objectives: first, prioritise selected traffic in a multi-path network environment and second, selecting the less congested path between two endpoints using the SDN central management for the prioritised traffic. As shown in table 2, we prepared two new flows from source to destination. Every new flow generates the same predetermined amount of load/traffic from source to destination using Iperf generation tool for the time shown in table 2. The Host H1 started to send traffic in the first second, when the switch S1 received the first packet sent by H1, the switch did not have any flow entry defined, therefore encapsulated the packet and sent it to the controller. The OpenDaylight controller took the packet and evaluated the header of the packet and determined the best path based on the shortest path algorithm. The path selected is Path 1 since it has six hops. The controller prepared the flow entries and installed them to each switch of the path 1. So, the traffic from **H1 to H13** took the **Path 1**. During the simulation, at 20 sec, the Host H2 started to send traffic to the Host H14. Basically, the switch and the controller executed the same procedure of H1 and H13 because the S1 did not have any flow entry about how to deal with this traffic. Based on the shortest path, it selected the path 1. At the same instance, we executed our proposed solution for the traffic from H2 to H14. Our algorithm evaluated the load of every path taking into account the bottleneck link. As shown in Table 1, we are considering seven paths, if we compare the

table with the topology, we realise that some links from the path 1 are used on the other paths, except in the path 3. Each link from the path 1 has a load of 75 Mbps. Therefore, the throughput availability of each path that share at least one link with the path 1 is 25 Mbps. The proposed solution selects the path 3 as it has the higher availability and redirects the traffic from **H2 to H14 via path 3**.

Table 2: Test Case 1: Simulation Parameters

Flow No.	Source/Dest	Traffic/Load	Duration(sec)	Start at
1	H1 to H13	75Mbps	60	0 sec
2	H2 to H14	75Mbps	40	20 sec

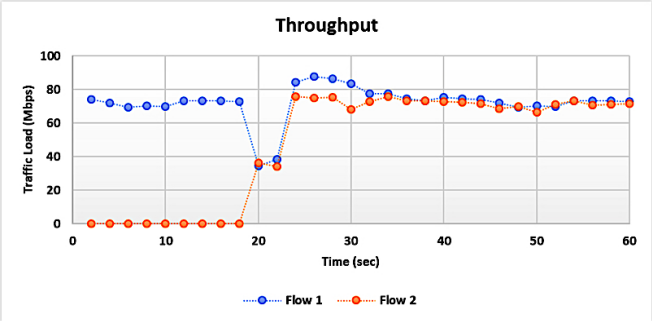
3.3 Test Case 2

The second test case is realized with the objective to compare our proposed solution (PS) with algorithm based on the shortest path (SP). First, give a special treatment to a selected traffic in a multi-path network environment; and to prove that the shortest path is not always the best option. In this scenario, we sent traffic from H1 to H13 and from H2 from H14. The throughput used in this scenario is 75 Mbps. The controller was taking routing decision based on the shortest path. Therefore, both flows are using the same path (1). We captured the statistic information for 60 seconds. After that, we executed the proposed solution for the traffic from H2 to H14 and started to send traffic again from both sources (H1 and H2). The Flow 1 kept the same path (1) and the flow 2 was redirected to the path 3. We captured the statistic information for 60 seconds. In this test case, both the flows 1 and 2 of table 2 have same characteristics, except both are for 60 seconds duration and both start at 0 seconds.

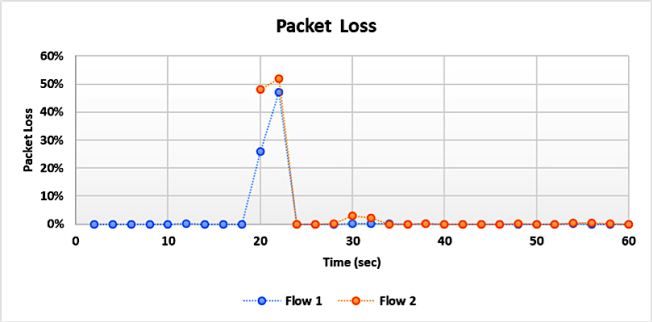
3.4 Results

In figure 2, throughput and packet loss are compared for both the flows for test case 1. As seen in figure 2a, when the flow 2 started to send traffic, it had a direct impact on the throughput of Flow1 since both flows were taking the same path and those links have a bandwidth of 100 Mbps. Both flows are trying to send traffic with a throughput of 75 Mbps, but the most that they can achieve before the execution of the proposed algorithm is around 40 Mbps. After the PS was executed, we can see that both flows achieve a throughput of about 75 Mbps. We can clearly observe that our solution increases the overall throughput of the network by using multiple paths and exceeding 100Mbps link in total. We are prioritising flow 2 and by using our algorithm, this flow is achieving the throughput that the host is sending by taking the less congested path.

Figure 2b shows the results of Packet Loss that was monitored in the test case 1. Packet loss is typically caused by network congestion. When the Flow 2 started to send traffic, the links of path 1 get congested. The percentage of packet loss reaches a peak of around 50 % in both flows. After our solution inserted the



(a) Throughput of Flows



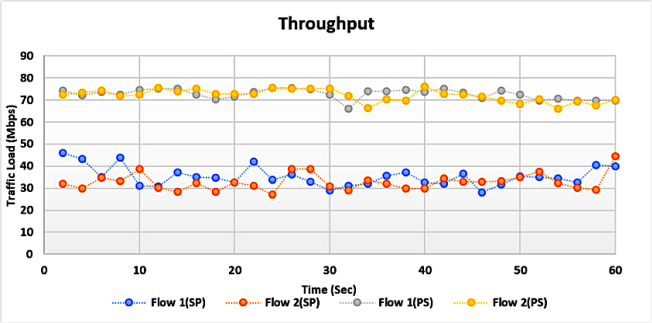
(b) Packet Loss of Flows

Fig. 2: Comparison of Flows in Test Case 1

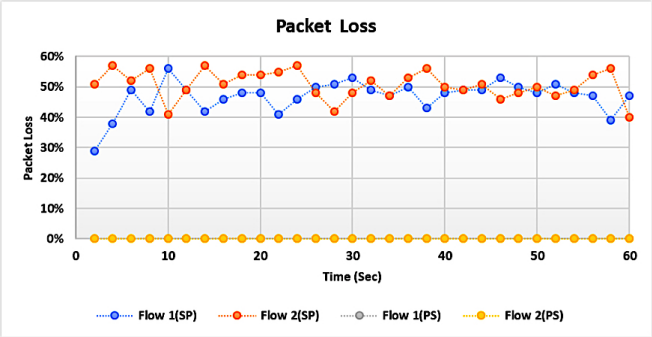
new flows entries in the switches, the packet loss percentage returned back to 0%.

The throughput results of test case 2 are shown in figure 3a. Here, we compare the resulting flows of our proposed solution (PS) with the existing shortest path (SP) solution. The bottom flows peaking at maximum around 47 Mbps are the results of trying to transmit 75 Mbps each flow into links limited to a 100 Mbps bandwidth. The problem with this routing is that once the algorithm selected the shortest path (SP), it keeps sending traffic by this path even it is overloaded. Using the PS, we can see clearly the difference in the throughput. The top flows peaking at maximum of 75 Mbps are taking the less congested path, the grey is taking the path 1 and the yellow is taking the path 3. We demonstrated that even though it is taking a path with more hops (7 hops in this case), it achieved a satisfactory performance on throughput and on packet loss. Therefore, we demonstrated that in multi-path environment, despite selecting a path with more hops but less congested, we can achieve a satisfactory performance of throughput and packet loss.

In figure 3b, we can see the comparison related to packet loss of the SP algorithm and our solution. Basically, the top flows red and blue are overloading the links of path 1 trying to send 75 Mbps each one into links limited to 100



(a) Throughput of Flows



(b) Packet Loss of Flows

Fig. 3: Comparison of Flows in Test Case 2

Mbps. Using the PS, the prioritised traffic is sent by a different path which is not congested. Therefore, the packet loss is maintained at 0% as we can see in the flows grey and yellow (bottom). In this test, we managed to treat mission critical flows with the stringent QoS requirements of zero packet loss.

4 Conclusions and Future Work

This paper proposes a system to provide special treatment to mission-critical applications in multi-path environments based in SDN. The system was implemented in the OpenDaylight controller and using the OpenFlow protocol. It was developed in the Application layer taking advantage of the rich northbound API provided by the control plane. We can conclude that our solution achieved the three proposed objectives. First, give a special treatment to a selected traffic in a multi-path network environment, second, using the SDN central management choose the less congested path between two endpoints for the prioritised traffic, and third, demonstrate that in multi-path environment, consider paths with more hops but less congested, it can achieve a satisfactory performance of throughput and packet loss. The results of the simulation validate that our

solution increases the overall throughput of the entire network. Furthermore, observing the comparison of our solution versus shortest path algorithm, we can conclude that our solution offers better performance than the shortest path algorithm and single path routing. However, it was not possible to monitor the latency and the delay of these flow paths. Generally, those values may increase while more hops there are into a path. For future work, latency and delay should be considered in the evaluation of choosing the less congested path and establishing a threshold for this value. Also, it is recommended to keep monitoring the QoS parameters and handle policies and events to assure continuous QoS on selected traffic. To realise these possible improvements, it is necessary to develop a statistic module within the controller. Unlike to our solution which processes the information in the application layer to obtain the throughput, there will be an increase in the response time due to policy execution to prioritise the traffic. A statistic module in the control plane will allow creating monitors for distinct QoS parameter such as delay, throughput, packet loss, among others in real time. The efficiency of this module is the base for any QoS system that offers solutions to the relentless growth of traffic volume, network size, and diversity in QoS requirements.

References

1. Dilmaghani, R., Kwon, D.: Evaluation of openflow load balancing for navy. In: MILCOM 2015 - 2015 IEEE Military Communications Conference. pp. 133–138 (Oct 2015). https://doi.org/10.1109/MILCOM.2015.7357431
2. Hojiev, S.Q., Kim, D.S.: Dynamic load balancing algorithm based on users immigration in wireless lans. Journal of Advances in Computer Networks **3**(2), 114–118 (2015)
3. Hu, Y., Wang, W., Gong, X., Que, X., Cheng, S.: Balanceflow: Controller load balancing for openflow networks. In: 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems. vol. 02, pp. 780–785 (Oct 2012). https://doi.org/10.1109/CCIS.2012.6664282
4. Karakus, M., Durresi, A.: Quality of service (qos) in software defined networking (sdn): A survey. Journal of Network and Computer Applications **80**, 200 – 218 (2017)
5. Liang, C., Kawashima, R., Matsuo, H.: Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers. In: 2014 Second International Symposium on Computing and Networking. pp. 171–177 (Dec 2014)
6. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: Openflow: Enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (Mar 2008)
7. Verma, D.C.: Simplifying network administration using policy-based management. IEEE Network **16**(2), 20–26 (Mar 2002)
8. Zhou, Y., Zhu, M., Xiao, L., Ruan, L., Duan, W., Li, D., Liu, R., Zhu, M.: A load balancing strategy of sdn controller based on distributed decision. In: 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications. pp. 851–856 (Sept 2014). https://doi.org/10.1109/TrustCom.2014.112