Multiflight: A Traffic Optimizer for SDN OpenFlow Networks

João Tomázio
Instituto Superior Técnico
Universidade de Lisboa
Lisboa, Portugal
joao.tomazio@tecnico.ulisboa.pt

***Abstract*—Due to the growing demand for bandwidth, computer networks are constantly reinventing themselves. The introduction of SDN represents a paradigm-shift on the history of data networks, as its mindset dramatically changed.**

In order to successfully replace traditional data networks with SDN solutions, this revolutionary concept still needs applications able to deal with the requirements of modern transmission of information. This comprises the availability of classic services, which consequently represents, among other examples, the ability to support their bandwidth consumption costs.

Layer-2 networks are an example of systems that still rely on traditional and sometimes overlooked solutions. This adoption of outdated protocols can lead to wastages on infrastructure resources, compromising the performance of the running applications.

In this thesis, we propose Multiflight, a SDN Controller module which aims to optimize layer-2 networks traffic by taking advantage of cycled topologies, exploring their benefits and mitigating their setbacks. Due to the assignment of alternative paths, Multiflight enables new forwarding techniques while ensuring basic compatibility with traditional and widely deployed protocols.

In order to materialize these concepts, Multiflight takes advantage of the Floodlight OpenFlow Controller and explores its open-source nature without compromising its modular architecture.

***Index Terms*—SDN, OpenFlow, Floodlight, Redundancy, Throughput**

I. Introduction

"Over the last years, the demand for networks' bandwidth has been growing on an unprecedentedly scale". This statement is a frequent conclusion of analysis reports from data network operators, namely Internet Service Provider (ISP) Backbones, Data Centers, Enterprise Networks, among others [1]. These events can be easily explained by multiple reasons. A few examples are given in the following list:

- Wide adoption of the concept of Digital Transformation in multiple administrative processes [2];
- Proliferation of physical devices like smartphones, tablets, laptops and Internet of Things (IoT) objects on both personal and professional environments;
- Generalization of heavy bandwidth-consuming services like video streaming and remote storage.

In the Information Technology (IT) domain, it is widely known that the quality of a data network has direct impact on the performance of its hosted applications. Quality includes reliability, as well as availability for the demanded bandwidth.

This work attempts to address these requirements, following the growing needs of data bandwidth while leveraging resources on SDN networks, in order to potentiate traffic balance and maximize data throughput of applications.

In the scope of this thesis, we developed a software package called Multiflight, being available on a GitHub repository [3].

As an important note, all the bandwidth, throughput and data size values presented on this document are expressed according to the International System of Units (SI) and International Electrotechnical Commission (IEC) standards of decimal and binary prefixes [4].

A. Motivation, Objectives and Contribution

The prime motivation behind Multiflight is the will to provide a reliable and fast transmission of information, from a source to a destination within a data network, with minimum latency. Following this premise, Multiflight performs traffic forwarding by redirecting lines of traffic to the most appropriate routes, avoiding path overloading. This procedure is applied using two main techniques:

- Inspecting the network in order to find the less congested route, taking into account bandwidth availability;
- Deployment of SDN rules along the data plane such that an optimal route is adopted;

The concept of allocation of the most available slots is frequent in the IT domain. A practical example is the case of Worst-Fit Memory Management algorithms [5], as their processes try to locate the biggest partition of free memory to allocate a certain entry of data in order to maximize availability for big chunks.

With this approach, Multiflight expects to reduce transmission delays and maximize data transfers in its associated networks.

Multiflight concepts cover only a fraction of the actual reality of computer data networks, focusing only on Layer-2 networks.

In order to materialize its concepts, Multiflight uses the SDN paradigm, taking advantage of an already established and widely deployed SDN Controller (Floodlight). However, Multiflight doesn't present itself as an upgrade package for

the Floodlight source code. This is due to the fact that Multiflight's scope does not cover all the types of data protocols already handled by Floodlight default functionality. Instead, Multiflight addresses only certain types of data traffic, being more like a proof of concept rather than a new feature package.

II. Background

A. Traditional Data Networks

Since the creation of the Advanced Research Projects Agency Network (ARPANET) in 1969, which late evolved to the Internet, data networks never really changed their paradigm. Although modern network devices include now complex and more sophisticated features, most enterprise networks still rely on conventional routers and switches, composing a fully distributed data network. In this paradigm, computer networks are composed by sets of network devices organized in a decentralized architecture of connected nodes. Although information about the network is indeed exchanged between them, which is vital for their coordination protocols, each network device computes its own decision metrics, resulting sometimes on an absence of a centralized and global view of whole network.

The internal architecture of network devices is often classified in three planes of functionality:

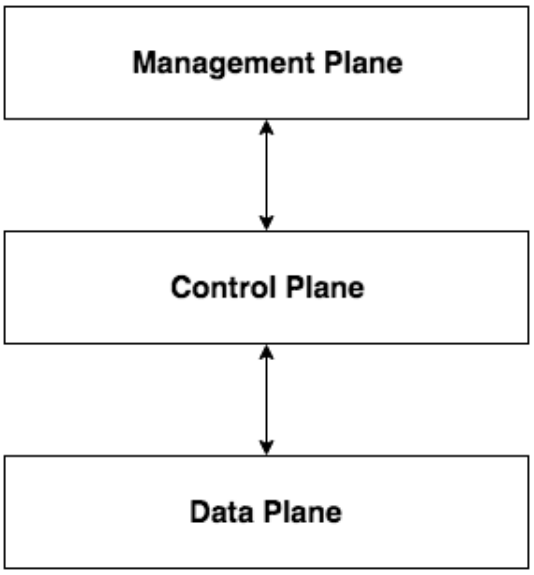

Fig. 1. Traditional Network Devices - Planes of Functionality

- **Management Plane**: The Management Plane covers functionalities associated with Command-line Interface (CLI), Graphic User Interface (GUI) interpreters, Monitoring, Access Control List (ACL), Security Policies and other Top-level Configurations
- **Control Plane**: The Control Plane is responsible for protocols that generate forwarding decisions and rules to incoming packets. Most of the times, these protocols implement well-known standards (e.g. Spanning Tree Protocol (STP), Routing Information Protocol (RIP) or Open Shortest Path First (OSPF)). However, the existence of proprietary protocols on network devices is very common. The mechanisms existing on these protocols produce rules that are needed for the Data Plane to operate.
- **Data Plane**: It is responsibility of Data Plane functions to examine incoming packets and decide actions to follow. These decisions can be made based on many indicators, e.g. destination information, type of data content, rate of congestion of incoming and/or outgoing links, etc. The rules to evaluate these metrics are computed by the Control Plane protocols.

In spite of a decentralized external architecture of nodes, traditional data devices still rely on an internal centralized architecture regarding the planes of functionality, assuming that each plane depends directly on the adjacent one specially designed for that device, which often leads to problems of flexibility and scalability. As information is independently gathered by each network device, as well as its computation, sometimes the view of the state of the network can be divergent between its own nodes, resulting in instability on their coordination. An example of such effect is described on section II-B1.

B. Packet Forwarding in Layer-2 Networks

1) The Loop Problem: One of the most common network topologies is the Ring or based on the Ring format [6]. This design is very popular due to its resilience to failures on links of the network. In a simple Ring Topology, if a single link fails, the network topology virtually becomes a Line one, because two nodes get disconnected and the cycle is broken, but the network continues to work because none of the nodes becomes unreachable. This is crucial in network implementations where reliability is a must-have, i.e., when fails result on heavy consequences. Figure 2 presents an example of this behavior.

The biggest challenge when implementing a loop-based topology is that coordination cannot be achieved with simple learning switch approaches. A loop on a network cause several problems:

- If a packet reaches a switch of a cycled network and gets flooded without control, it may be cycled through the loop infinitely. In addition, because multiple outgoing interfaces may be issued by a single incoming packet, the number of packets travelling in the network will exponentially increase until the capacity of the network is reached or until the network crash due to the traffic saturation;
- Due to the cycle, when a host sends packets, a bridge from this network could receive it from more than one interface. With the learning procedure, bridges are forced to overwrite their records from the Forwarding Information Base (FIB) every time a packet from the same host is received on different ports. Therefore, this can cause the FIB to become very unstable, which can crash the bridge;

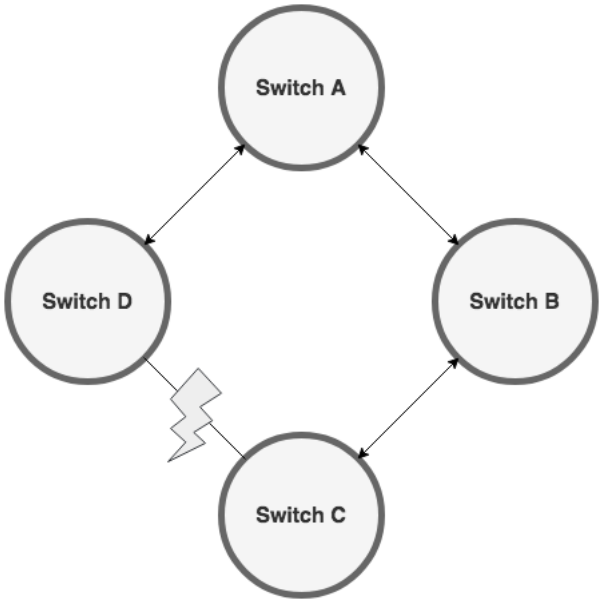

Fig. 2. Failure on a Ring Topology

- When a message is flooded by a certain bridge that doesn't know the correct segment to forward, the recipient can received it more than once, which can affect the running applications of that host.

On the subsection II-B2, we describe a special coordinating algorithm that aim to prevent these types of problems in traditional networks.

2) Spanning Tree Protocol: STP [7] is a network protocol used to prevent loop transmissions in layer-2 networks. STP assumes the concept of logical topology, creating a layer of abstraction above the physical topology. In this logical topology, some links of the physical network that may complete a loop are disabled and are not used in packet transmission. This way, the loop is eliminated. The name Spanning Tree comes from the idea of a tree, where exists a root (Root Bridge in the network) and a set of branches and leaves (the rest of the bridges of the network). There are several versions of the base protocol as Rapid Spanning Tree Protocol (RSTP), Multiple Spanning Tree Protocol (MSTP) or the Cisco proprietary Per VLAN Spanning Tree (PVST).

C. Software-Defined Networks

SDN [8] represents a paradigm-shift comparatively to the traditional way of managing, operating and using computer networks. As the name suggests, SDN networks may have their topologies, routines, forwarding and security defined by software. It is well-known that it is always the software held in the hardware that controls the device, but in the SDN paradigm, this control software doesn't run in each network device. Instead, it runs on a central external device called the SDN Controller. This entity acts on network devices, configuring, for example, which forwarding action should be followed at the reception of a certain packet or which security policy should be followed after certain event. This implies a new network architecture that is going to be explained below.

Modern SDN implementations propose to address modern network challenges with flexible and complete solutions. Based on the concept explained above, SDN resides on the idea of decoupling the three planes (section II-A) that usually compose conventional network devices existing in traditional networks. With this approach, it is possible to mitigate some of the problems referred on section II-A, but keeping (and sometimes even improving) the classic functionality of data networks. Figure 3 shows a diagram representing a structure for this new architecture.

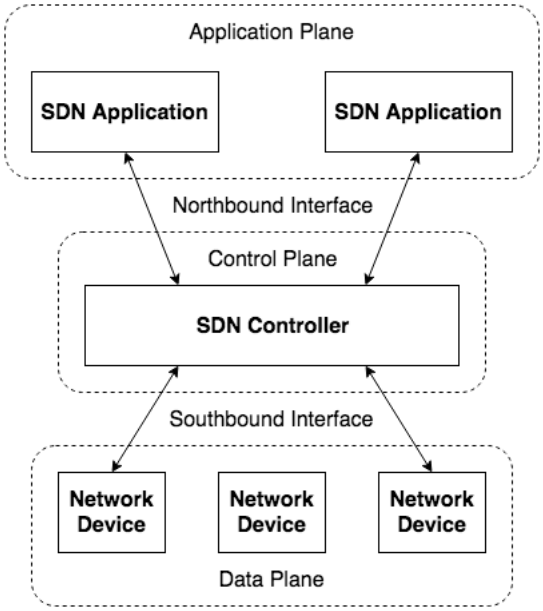

Fig. 3. SDN - Planes of Functionality

As we can see on the figure 3, the three planes are maintained, but they are now logically separated and almost always running in different systems. The control plane is now ruled by the so-called SDN Controller. This controller programs and injects rules on the data plane. With this architecture, network nodes are only associated with the data plane and serve the simple task of forwarding. To manage all the rules and reports, the application plane can, with this architecture, be driven by multiple SDN Applications that communicate with the controller.

1) OpenFlow: OpenFlow [9] is a communications protocol intended to support the Southbound Interface (SBI) interface of an SDN-type network. It should be implement by both Controllers and Network Nodes, elements that compose the Data Plane of the SDN architecture. As referred on the section II-C, this plane has the SBI as associated interface which OpenFlow has become the *de facto* standard. It provides the ability to a SDN controller to deploy rules on a network, in order to trigger certain actions upon certain incoming packet matches.

With this architecture, OpenFlow is able to match profiles, statically or dynamically. The dynamic definition of flows is going to be a strong component of the work presented in this document, due to the volatility of the decisions sent from the SDN Controller to the Network Nodes. Another very important component of Multiflight is going to be the Statistics Module of the node, both internal or traffic-related. These metrics are going to be offered to the SDN Controller that will take them as inputs of algorithms that, depending of their results, will indicate which instructions should be given to the network. One of these metrics needed for the controller should incorporate data from the Counter fields of the Flow Entries, as it provides a good estimate of the amount of traffic load that reached the node, matching some profiles.

III. Architecture

A. Concept

As referred on section I, the main goal of Multiflight is to increase the efficient use of resources on a subnet, i.e., on a Layer-2 computer network. To achieve that, Multiflight attempts to develop a solution that mitigates the known problems of the STP (section II-B2), considering the key-properties of a SDN environment. The idea is to develop an innovative procedure to efficiently exchange data packets between hosts on a network, i.e., to improve traffic accommodation over the available resources that a certain network can offer at the moment they are required.

Previously in this document (section II-B2), we presented and explained the attributes of the STP. Besides problems of convergence and coordination, we point the characteristic of blocked links. Used in STP to prevent loops (section II-B1), we consider this approach over-conservative, because it disables valuable resources, the links between network devices. These blocked links, if not blocked, could be used to compose alternative paths. If redundant data paths exist, data traffic may potentially be distributed and consequently reduce usage stress on links, comparatively with the usage of single path solutions like STP. Figure 4 and 5 show two logical topologies of a same network, using STP and Multiflight, regarding accommodation of traffic between two hosts.

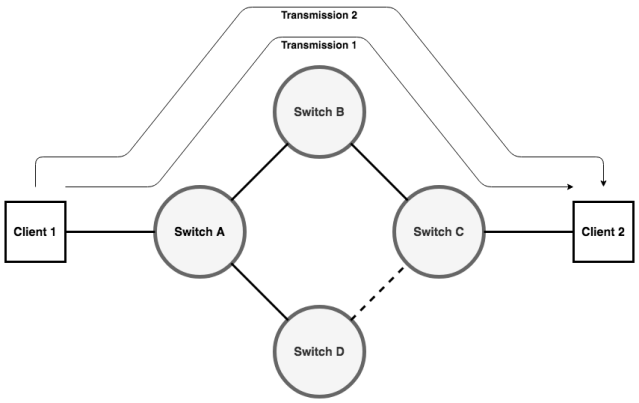

Fig. 4. Transmission on a STP logical topology

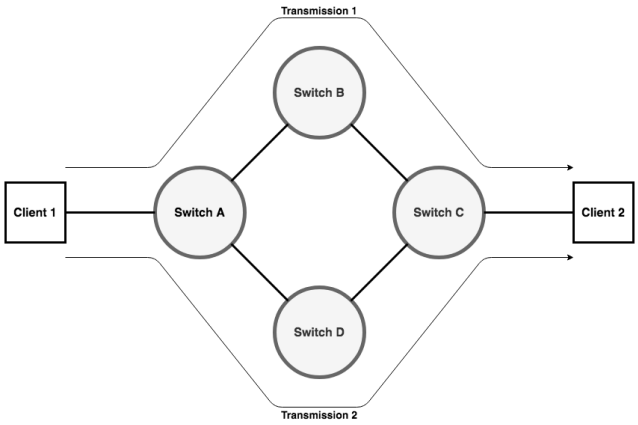

Fig. 5. Transmission on a Multiflight logical topology

Moreover, in the solution presented by the STP, paths between network devices are computed and settled during a pre-operational configurations phase, i.e., before main data transmissions, until a reset or a reconfiguration occurs. In addition to this limitation, the metrics used to generate link costs are often administrative costs, i.e., costs given by the network administrator, during the configurations phase, to achieve a certain default logical topology. This approach is slightly outdated and does not suit the modern reality of computer networks, as it lacks on flexibility and adaptability to the dynamic network needs given operational conditions at each time. In other way, we think the path that data packets may take should be decided at the moment, or at least, very close to the moment that the necessity to this path arises. In our expectation, we think that this approach could offer a better path to communications that are supported by a network than traditional protocols like STP. By better path, we consider a path that offers better data throughput, less latency and increased availability. And this best path could be obtained by choosing, among multiple paths, the one that is less obstructed, i.e., the one with more available bandwidth.

B. Controller Integration

Multiflight presents itself as a technique to optimize data traffic in layer-2 subnets on a SDN environment. Starting from this statement, and taking in account the nature of an SDN network, it becomes clear that Multiflight will have to operate at the controller level.

Among multiple SDN controllers, the choice has fallen upon Floodlight because of its recent popularity on SDN researches and papers.

Floodlight [10] is an Apache-licensed implementation of a SDN controller that supports OpenFlow. It is currently on version 1.2 supporting OpenFlow until version 1.5. Based on Java, it is one of the most prolific SDN Controllers indicated to research purposes. The majority of its functionalities are exposed to top-level SDN Applications via a Representational State Transfer (REST) Application Programming Interface

(API). For user interaction, Floodlight has a web-based User Interface (UI) and a Java-based UI, called Avior.

In order to perform its tasks, Multiflight hooks in the Floodlight structure as an internal Module Application. Consequently, Multiflight takes advantage of multiple low-level services in order to get access to basic network information status like connected switches and hosts.

Multiflight uses the same mechanisms and tools as the ones used by the already existing module application Forwarding. This module already deploys OpenFlow flows to network devices in order to forward data traffic. However, this module doesn't take full advantage of the capabilities offered by the Core Services of Floodlight neither the possibilities of the SDN architecture. This issue is described on section III-C.

Due to the similarities between Multiflight and Forwarding, the default module application, Multiflight requires the deactivation of this module in order to prevent concurrent actions that may be counter-productive and lead the network to an inconsistent state.

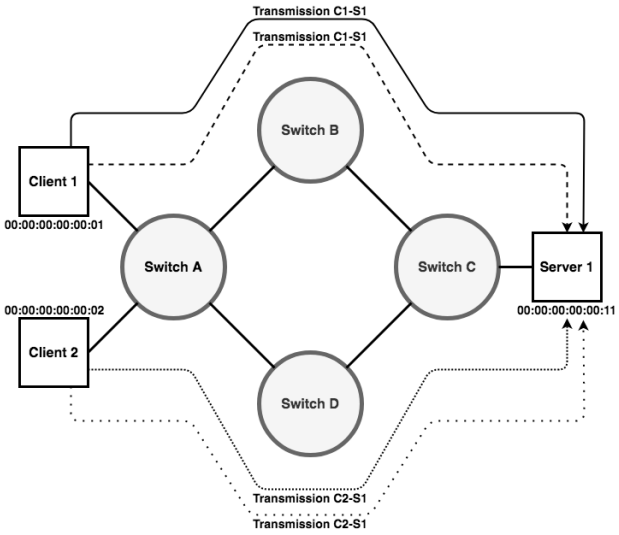

Fig. 6. Functionality of the Multiflight's Host-based combination of traffic

C. Functional Architecture

As referred on the section III-B, Multiflight replaces the default Forwarding module on the Floodlight environment, with a different approach to packet forwarding. The default approach only contemplates the acknowledgement of the two associated hosts with a strict type of flow, concretely with a fixed value for its OpenFlow Idle timeout of five seconds.

On the Multiflight approach, forwarding methodologies are more flexible and more adaptive than the default ones. The lookup of the best path is computed on demand, i.e., at the moment that a packet arrives on the first switch. In order to focus the scope of the work, Multiflight operates only on certain type of traffic, more precisely on Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP) transmissions. With the objective of maximizing the performance of the data transmission, traffic is classified and aggregated in one of two modes of operation:

- **Host-Combined:** In this aggregation mode, traffic with the same source and destination is aggregated to follow the same rules, taking in account the existing OpenFlow flows. This ensures that, for example, a best path is reserved for a connection between a client and a server, shared for all of its applications' subconnections. Figure 6 shows an example of this behavior.
  In this example, Client 1 attempts to send data packets to Server 1. When the packet reaches switch A, they are forwarded to the SDN Controller, entity that computes link costs and deploys flows in order to set this best path on the involved switches, in this case, switches A, B and C. The match of these flows are the following:
  - **Source Media Access Control (MAC) Address:** 00:00:00:00:00:01;
  - **Destination MAC Address:** 00:00:00:00:00:11;

  After this moment, the next received packets on switch A from Client 1 to Server 1, are going to be forward to the Switch B, then to Switch C, and finally to Server 1. Moments later (while the first transmission is still running), another client (Client 2) starts to communicate with Server 1. Its packets reach the first hop (Switch A) that forward them to the controller. At this moment, the controller realizes that the path with less usage is the one composed by switches A, D and C, so flows are deployed on them. The match for this new connection is the following:
  - **Source MAC Address:** 00:00:00:00:00:02;
  - **Destination MAC Address:** 00:00:00:00:00:11;

  After this deployment, data packets from Client 2 to Server 1 will follow this rule. After some pre-defined delay, the flows existing on the switches expire and the network returns to the initial state.
  As it can be seen, traffic from 2 different clients towards a single server can be balanced on a network, using the pair Source Address-Destination Address to differentiate the multiple kind of aggregations.
- **Application-Combined:** In this second aggregation mode, traffic is combined using a more granular approach. In this case, grouping follows the match of 4 parameters: Source MAC Address, Source Port, Destination MAC Address and Destination Port. This way of aggregation increases the dilution of traffic on a network, automatically balancing the load from the same client and to the same server, on multiple paths. Figure 7 illustrates an example of this approach.
  In this example, Application 1 from the Client 1 tries to send data packets to the listening Application 1 of Server 1. When they reach Switch A, packets are forward to the SDN controller that deploys OpenFlow flows on the involved switches of the best path (A, B and C) that contain the following kind of matches:
  - **Source MAC Address:** 00:00:00:00:00:01;

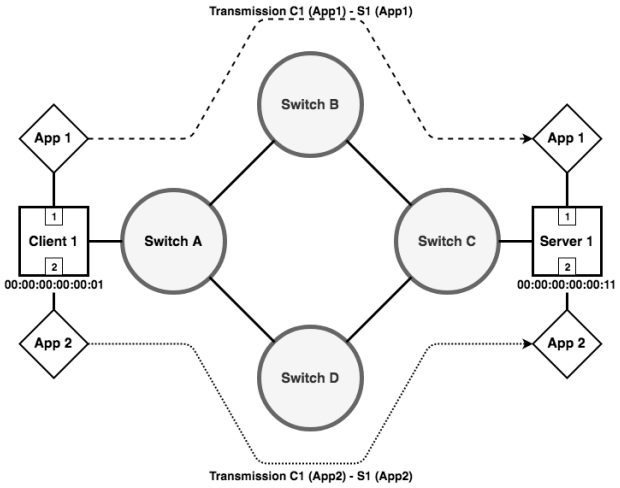

Fig. 7. Functionality of the Multiflight's App-based combination of traffic

- **Source Application Port:** 1;
- **Destination MAC Address:** 00:00:00:00:00:11;
- **Destination Application Port:** 1.

The next referred data packets will match the flow rule and follow the path until they reach Server 1.
While the flow described above is still active, Application 2 from Client 1 starts to send packets to other listening application, identified by the number 2, from Server 1. Again, when packets reach the first hop (Switch A), are forwarded to the controller that calculates a new best path, composed by the links with best dynamic availability (switches A, D and C). Then, flows are deployed on these switches with the following match header:

- **Source MAC Address:** 00:00:00:00:00:01;
- **Source Application Port:** 2;
- **Destination MAC Address:** 00:00:00:00:00:11;
- **Destination Application Port:** 2.

Subsequent incoming packets on the first hop will follow this rule. Thus, in this aggregation mode, Multiflight establishes two different routes for data packets from the same client and to the same server, if they come from different application ports, which induces balancing of data transmission in the referred network.

IV. Evaluation

A. Tests Objectives

As in any software system, implementation phases must be complemented with testing phases. In the case of experimental or academic researches, test beds represent a way of quantifying the success of the product of the performed work [11].

The tests developed in the scope of this thesis are elaborated in order to fulfill the following objectives:

- Verify Multiflight's general forwarding functionality according to the architecure proposed;
- Ensure operability of the Multiflight's forwarding functionality in order to validate its usability on a real computer network;
- Compare performance results between both default Floodlight and Multiflight libraries regarding its forwarding capabilities;
- Highlight Multiflight's performance gains against default Floodlight libraries when compared on equal test conditions;
- Promote cycled and redundant topologies against lined and tree topologies.

B. Tests Scenarios

Multiflight's Evaluation phase was supported by an extensive set of cross testing benchmarks, among the forwarding libraries (default Floodlight, Multiflight Host-profiling and Multiflight App-profiling), multiple topologies, multiple test modes and multiple network conditions. After this, all data of each test was analysed and the final result is presented by the form of its Mean Value and the associated Standard Deviation over 10 repetitions. The result of each test consists on the measured throughput between a client and a server, both in TCP and UDP, using traffic generated by the Iperf tool.

In order to trigger all the tests sequentially, a shell script was created to cross all the combinations of network variants described above. On each test, a reset is made to the controller and a new emulated network is issued. This ensures isolation of every test, preventing a network state of a test to influence the next test. Then, and depending on the nature of the test, a certain network condition is set and the test operation is performed.

1) Test Topologies: Multiflight's Evaluation Phase used a set of testing network topologies in order to cover the most common scenarios regarding different styles of interconnections between devices. Each topology is composed by a minimum of two clients and two servers. This configuration is due to the setup of pre-congested networks on certain tests, as C1 traffic is the one analysed, while the role of the rest of the clients is to generate dummy traffic.

In order to simplify the descriptions on the following topologies, to indicate a Client is used the letter C, to indicate a Server, letter S and to indicate a Switch, it is used the expression SW.

Test network topologies were the following:

- **Pair:** This topology is composed by two switches (SW1 and SW2) arranged in a line formation and four hosts (C1, C2, S1 and S2). The base bandwidth is set to 100 Mb/s when connecting clients, 300 Mb/s when connecting servers and 1000 Mb/s when interconnecting switches.
- **Cycle:** This topology is composed by four switches (SW1, SW2, SW3 and SW4) arranged in a ring formation and four hosts (C1, C2, S1 and S2). This topology is the first to introduce a cycle of switches, as its evaluation intends to demonstrate Multiflight's loop management techniques and multiple path arrangements. In order to notice different optimizations on traffic allocation, the

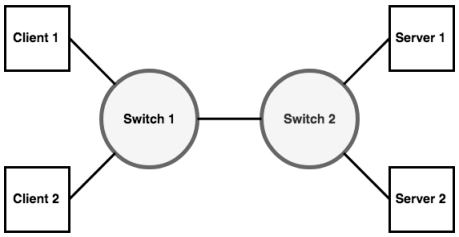

Fig. 8. Multiflight's Pair Testing Topology

base bandwidth is set to 1000 Mb/s when connecting clients, 300 Mb/s when connecting servers and 100 Mb/s when interconnecting switches.

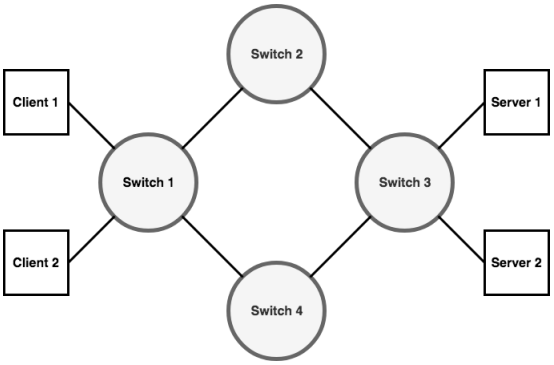

Fig. 9. Multiflight's Cycle Testing Topology

- **FatTree:** This topology is composed by seven switches (SW1, SW2, SW3, SW4, SW5, SW6 and SW7) arranged in a tree formation and eight hosts (C1, C2, C3, C4, S1, S2, S3 and S4). The main purpose of this topology is to recreate a common FatTree, a topology that is typically introduced on Data Centers. The base bandwidth is set to 100 Mb/s when connecting clients, 300 Mb/s when interconnecting first and second tier switches, 500 Mb/s when interconnecting second and third tier switches and 1000 Mb/s when interconnecting third and fourth tier switches.

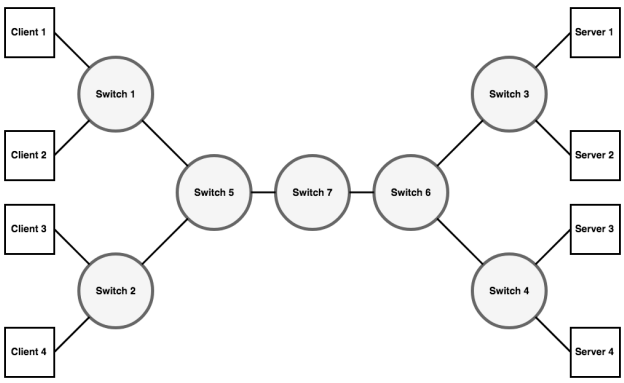

Fig. 10. Multiflight's FatTree Testing Topology

- **Mesh:** This topology is composed by seven switches (SW1, SW2, SW3, SW4, SW5, SW6 and SW7) arranged in a mesh formation and eight hosts (C1, C2, C3, C4, S1, S2, S3 and S4). The main purpose of this topology is to introduce multiple cycles and multiple paths options, while using the same resources (seven switches) of the FatTree topology, due to the intention of proving the benefits of mesh formations when compared with tree ones. The base bandwidth is set to 100 Mb/s when connecting clients, 300 Mb/s when connecting servers and 100 Mb/s when interconnecting switches.

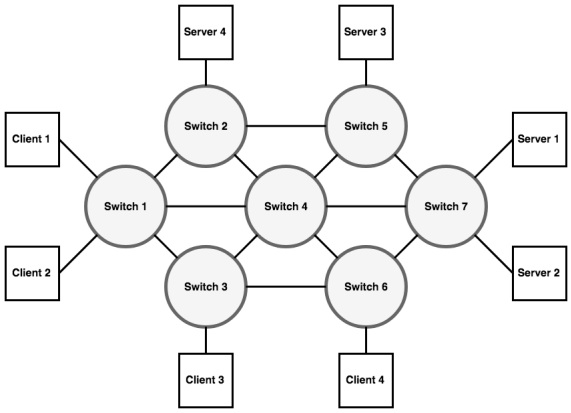

Fig. 11. Multiflight's Mesh Testing Topology

2) Test Modes: Regarding network clients and servers, multiple styles of transmissions can be performed in order to support a certain type of applications. Multiflight's Evaluation Phase collected some of these transmissions patterns and included them on its test bed. The test modes are the following:

- **Single:** This test mode consists on a single transmission between C1 and S1 for a duration of 10 seconds;
- **Sequence:** This test mode consists on a sequence of two transmissions between C1 and S1, each with a duration of 10 seconds;
- **Burst:** This test mode consists on a burst of three transmissions between C1 and S1, each with a duration of 15 seconds and starting 5 seconds after the start of the previous transmission;
- **Parallel:** This test mode consists on two parallel transmissions between two clients and one server (C1-S1 and C2-S1), each with a duration of 10 seconds and starting 5 seconds after the start of the previous transmission.

3) Test Conditions: In order to create the most realistic network conditions, Multiflight's Evaluation phase established two environments to be set before the execution of each individual test. The environments are the following:

- **Cold-start:** This scenario materializes the absence of environments setup, i.e., a test performed under this cold-start condition will find the network right after the emulator instantiation, with its Address Resolution Protocol (ARP) caches tables clean and minimal traffic on the network links;

- **Crowded:** This scenario sets a partially crowded network, with certain clients generating dummy traffic, depending on each topology:
  - **Pair and Cycle:** On these topologies, Crowded condition will set Client 2 to generate dummy traffic towards Server 2 for the duration of the associated Test Mode;
  - **FatTree and Mesh:** On these topologies, Crowded condition will set Client 3 to generate dummy traffic towards Server 3 and Client 4 to generate dummy towards Server 4, for the duration of the associated Test Mode;

C. Test Results

Due to the extensive amount of test data, only a few scenarios will be discussed in detail. For each one of the following test results, there is a graph illustrating throughput rates of multiple transmissions over their length, with step values of one second. Each one of the presented throughput values is computed by the Arithmetic Mean of 10 repetitions.

cycle-burst-cold-tcp: This test comprises the Cycle topology, Burst mode, Cold condition using TCP traffic.

As it can be seen of figure 12, some transmissions of certain controllers resist better than others to the introduction of their next bursts.

As easily noticed on the graph, during the first 5 seconds, first bursts of each controller take the whole available bandwidth and reach the maximum theoretical throughput rate.

With the introduction of the second burst, the rates of both Floodlight and Multiflight's Host mode transmissions start to drop, while the Multiflight's App mode transmissions remain at their maximum rate.

As explained on section IV-B2, Burst mode is composed by 3 transmissions of the same client, but from different source applications. Since Multiflight's App mode is the only that differentiate traffic between two hosts depending on its applications ports, a free alternative path was chosen by this controller to the second burst, resulting on a different path for each transmission.

Consequently, both first and second bursts indicated maximum throughput during the 5 seconds.

Thereafter, with the introduction of the third burst, both Floodlight and Multiflight's Host mode added the third transmission to the already congested path, due to the aggregation by source-destination hosts of each Controller.

In the case of Multiflight's App mode, the controller selected the least congested of the two paths between Client 1 and Server 1. Because they were both saturated, the throughput rate of this third burst was considerable low.

As soon as both first and second bursts ended, paths started to become less congested which led to an increase of the throughput rate of the last standing transmissions.

This test provides clear conclusions towards the advantages of Multiflight's App mode in cycled topologies, in situations where a single host produces and/or receives data traffic regarding multiple applications.

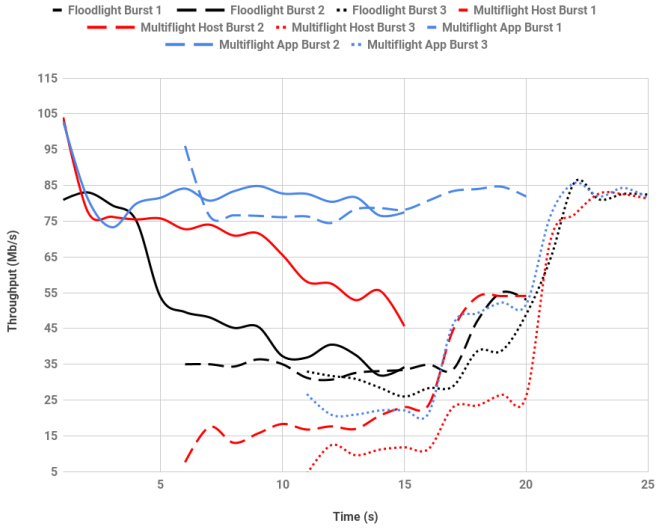

Fig. 12. Result of the cycle-burst-cold-tcp test

fatTree-single-crowded-tcp: This test comprises the fatTree topology, Single mode, Crowded condition using TCP traffic.

As it can be seen of figure 13, the transmissions ruled by each one of the controllers indicated a maximum throughput rate on this test. Yet in crowded condition, fatTree topology is composed by smaller links near the hosts, and big links on its backbone, so the congestion problem never arises on this test.

This FatTree topology comprises a single path between its Clients and Servers, so Multiflight's concepts cannot provide any advantage when comparing to the default library of Floodlight. Nevertheless, this test confirms the ability to ensure standard functionality on this type of topologies.

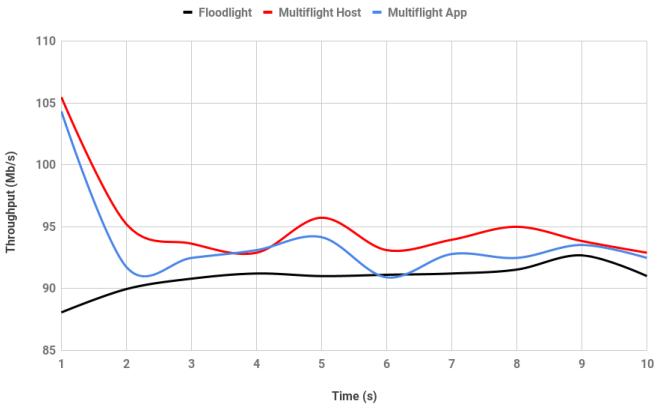

Fig. 13. Result of the fatTree-single-crowded-tcp test

mesh-single-crowded-tcp: This test comprises the Mesh topology, Single mode, Crowded condition using TCP traffic.

As it can be seen of figure 14, transmissions ruled by Multiflight's Host and App libraries produce rates considerable higher than the one associated with the transmission ruled by the Floodlight library.

Like the fatTree topology, crowded condition in Mesh topology is also supported by C3-S3 and C4-S4. This creates a great challenge to Multiflight's libraries due to the multiple cycle possibilities that this topology induce.

By analysis of the graph, it is noticeable that Floodlight has chosen some congested links in order to establish its path. Consequently, Floodlight rates suffered a big drop, resulting on a low performance of the controller during this transmission.

As expected, both Multiflight's Host and App modes selected the best path for their transmissions, resulting on an overwhelming result, demonstrating the advantages of their concepts.

Comparing these results with the ones obtained in the fatTree-single-crowded-tcp test, it is quite noticeable the advantages of cycled topologies against lined ones. Mesh topology is composed by the same number of switches than the fatTree topology, but with much lighter links. Comparing the single-crowded-tcp tests of both topologies, Multiflight effectively evened the throughput rate using the benefits of cycles, in spite of the Mesh topology handicap due to its reduced links bandwidth.

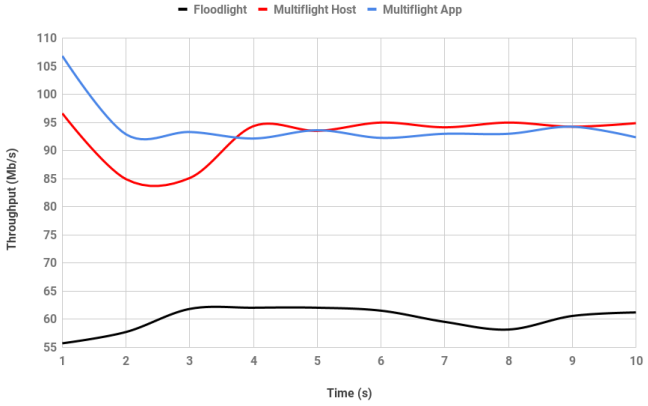

Fig. 14. Result of the mesh-single-crowded-tcp test

mesh-parallel-crowded-tcp: This test comprises the Mesh topology, Parallel mode, Crowded condition using TCP traffic.

As it can be seen on figure 15, transmission rates from both modes of Multiflight were considerable higher than the ones presented by parallel transmissions of default Floodlight library.

Due to the crowded condition, Floodlight performance was affected, resulting on reduced throughput rate. With the introduction of the the second transmission, the results got even worse, with every transmission (evaluated and dummy) sharing sharing links of their paths.

In the case of the two operating modes of Multiflight, first transmissions of both Host and App modes showed immunity to the crowded condition. However, with the introduction of the second transmission, Multiflight couldn't be able to select a new path for this new set of traffic. This was due to the Routing Service of the Floodlight's Core. Multiflight's instructs this service to consider Link Utilization as the metric of its operation, in order to perform its computations. Nonetheless, this service presents a noticeable setback. With Link Utilization as metric, Routing Service performs its computation in an operation with two phases:

- Firstly, Routing Service select a group of possible paths with the ones containing the minimum number of hops from source to destination, being less hops the better. In other words, each link of the network is considered to be associated with a fixed cost of 1;
- Lastly, Routing Service sorts the group of computed paths by the actual rate of Link Utilization.

This approach effectively achieves its purposes when alternating paths composed by a same number of links. However, in the case of this test, the selected path for the first transmission was SW1-SW4-SW7 (3 hops) and for the second transmission, even though the best path would be either SW1-SW2-SW5-SW7 or SW1-SW3-SW6-SW7 (4 hops) because of their full availability, Multiflight couldn't consider these options because it was limited to the maximum number of 3 hosts, so SW1-SW4-SW7 was selected again, resulting on a shared path by the two transmissions.

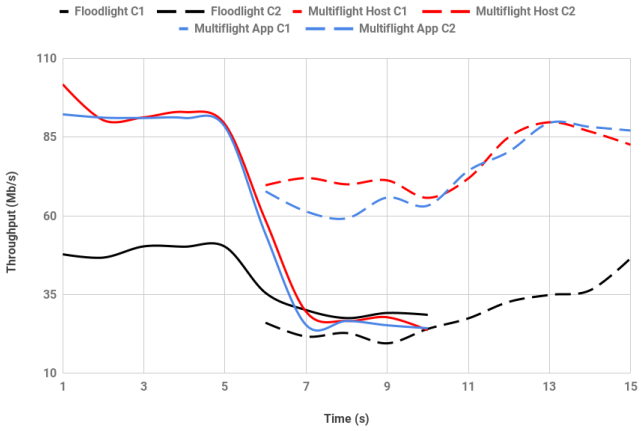

Fig. 15. Result of the mesh-parallel-crowded-tcp test

V. Conclusions

A. Summary

SDN represents a paradigm shift, enabling a revolution on the way data networks are known. With its introduction, networks architecture evolved and in addition to new services that emerged [12], new ways of optimizing data also appeared.

Multiflight addressed this opportunity, expanding previously developed work regarding the SDN Controller Floodlight, with the goal of maximizing the performance extracted from layer-2 networks. As stated in the section I-A, Multiflight's

purpose doesn't meet the direct extension of the Floodlight source code. Instead, this work can be seen as a motivation materialized into a proof of concept implemented in order to support Multiflight's main ideas.

B. Achievements

As any software package or application, its usefulness has a direct dependency on its performance. In the case of Multiflight, its performance was extensively evaluated by the performed set of tests described on section IV.

After the analysis of the test results, it is quite noticeable the success of Multiflight. Almost in every performed test, the option of Multiflight over Floodlight default library resulted on a positive benefit.

Moreover, the test results have successively accomplished each one of their proposed goals, presented on section IV-A.

C. Future Work

This work suggests several open lines of research and possible developments, both in mitigating the identified limitations as well as improving the positive results that were obtained.

Regarding the successively implemented concepts of Multiflight, future work may be developed in order to enable its full integration with the Floodlight source code, liberalizing its benefits to the community of developers than constantly rely on this controller either to support their researches or to integrate their production solutions on real computer networks.

REFERENCES

[1] S. Talpur and T. Kechadi, "A Forecasting Model for Data Center Bandwidth Utilization", SAI Intelligent Systems Conference 2016, September 2016.

[2] D. Delgado, "Governance Model for Digital Transformation", November 2017.

[3] J. Tomázio, Multiflight's Source Code, https://github.com/joaotomazio/multiflight.

[4] IEEE, "1541-2002 - IEEE Standard for Prefixes for Binary Multiples", 2002.

[5] M. Awais, "Memory Management: Challenges and Techniques for Traditional Memory Allocation Algorithms in Relation with Today's Real Time Needs", International Journal of Multidisciplinary Sciences and Engineering (IJMSE), vol. 7, no. 3, March 2016.

[6] H. Singh, S. Singh and R. Malhotra, "Modeling, Evaluation and Analysis of Ring Topology for Computer Applications Using Simulation", International Journal of Computer Science and Mobile Computing, vol. 2, no. 1, January 2013.

[7] G. Thomas, "Introduction to Spanning Tree Protocol", The Extension — A Technical Supplement to Control Network, vol. 6, no. 5, September-October 2005.

[8] J. Cox, J. Chung, S. Donovan, J. Ivey, R. Clark, G. Riley and H. Owen, "Advancing Software-Defined Networks: A Survey", IEEE Access, vol. 5, October 2017.

[9] ONF, "OpenFlow Switch Specification - Version 1.4.0", October 2013.

[10] V. Harkal and A. Deshmukh, "Software Defined Networking with Floodlight Controller", International Conference on Internet of Things, Next Generation Networks and Cloud Computing, 2016.

[11] N. Gonçalves, "A Testbed for research and development of SDN applications using OpenFlow", June 2014.

[12] J. Leite, "Complex Network Services using OpenFlow Barebones", October 2018.