**Implementation of Flow Migration Technique (Flow Migration)**

## Table of Contents

1 Introduction.....3
3 Guidelines for the Implementation of the Code and the Measurement Environment.....9
3 Commenting on the Results.....9

## Tables

Table 2.1: Table of the Change in the Order of Arrival of Packages at the Consignee..... 13

## Figures

Figure 2.1: Jitter between initial and new forwarding rules (h5-h11) .....9
Figure 2.2: Loss of Original and New Promotion Rules Packages (h5-h11) .....9
Figure 2.3: Jitter between initial and new forwarding rules (h2-h8) ..... 10
Figure 2.4: Loss of Original and New Promotion Rules Packages (h2-h8) ..... 10
Figure 2.5: Jitter between initial and new forwarding rules (h1-h5) ..... 11
Figure 2.6: Loss of Original and New Promotion Rules Packages (h1-h5) ..... 11
Figure 2.7: Jitter between initial and new forwarding rules (h4-h10) ..... 12
Figure 2.8: Loss of Original and New Promotion Rules Packages (h4-h10) ..... 12
Figure 2.9: Change in the Order of Arrival of Packages at the Consignee ..... 14

# 1 Introduction

This paper aims to implement a technique for transferring a network flow to another path of a network topology. During the measurements, any complication in the communication between sender and receiver, such as increase in delay, packet losses and change in packet arrival order, should be evaluated. The flow transfer will be done using OpenFlow.

The experiments were performed in a virtual machine environment with Ubuntu 14.04 64-bit operating system and the D-ITG tool was used as the network traffic generator. The network topology where the experiments were performed was the same as that of Abilene.

## 2 Guidelines for the Execution of the Code and the Measurement Environment

First, to perform the experiments, the D-ITG tool must be installed. Following the instructions, we install it in the */home/ubuntu/mininet/custom/* directory as follows:

```
sudo apt-get install unzip
sudo apt-get install g++
cd mininet/custom/
wget https://traffic.comics.unina.it/software/ITG/codice/D-ITG-2.8.1-r1023-src.zip
--no-check-certificate
unzip D-ITG-2.8.1-r1023-src.zip
cd D-ITG-2.8.1-r1023/src
make
```

To start Mininet, open a terminal and create the network topology of the python code (placed in */home/ubuntu/mininet/custom*) with the following command:

```
sudo mn --custom mininet/custom/AbilineTopo.py --topo abilinetopo --mac --arp --
link tc,delay=2ms --switch ovsk --controller remote
```

Once this is done, on a second terminal we run the ***rule1_basic.sh*** file, which contains the forwarding rules from h5 to h11, to create a two-way communication channel between the two. This file contains the following forwarding rules:

```
sudo ovs-ofctl add-flow s5 dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:0b,actions=output:4
sudo ovs-ofctl add-flow s5 dl_src=00:00:00:00:00:0b,dl_dst=00:00:00:00:00:05,actions=output:2
sudo ovs-ofctl add-flow s7 dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:0b,actions=output:4
sudo ovs-ofctl add-flow s7 dl_src=00:00:00:00:00:0b,dl_dst=00:00:00:00:00:05,actions=output:1
sudo ovs-ofctl add-flow s10 dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:0b,actions=output:3
sudo ovs-ofctl add-flow s10 dl_src=00:00:00:00:00:0b,dl_dst=00:00:00:00:00:05,actions=output:1
sudo ovs-ofctl add-flow s11 dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:0b,actions=output:3
sudo ovs-ofctl add-flow s11 dl_src=00:00:00:00:00:0b,dl_dst=00:00:00:00:00:05,actions=output:2
```

With a simple ping between sender and receiver we can check if the forwarding rules are correctly set.

```
mininet> h5 ping h11
PING 10.0.0.11 (10.0.0.0.11) 56(84) bytes of data.
64 bytes from 10.0.0.11: icmp_seq=1 ttl=64 time=36.1 ms
64 bytes from 10.0.0.11: icmp_seq=2 ttl=64 time=23.4 ms
64 bytes from 10.0.0.11: icmp_seq=3 ttl=64 time=23.3 ms
64 bytes from 10.0.0.11: icmp_seq=4 ttl=64 time=23.2 ms
64 bytes from 10.0.0.11: icmp_seq=5 ttl=64 time=24.0 ms
64 bytes from 10.0.0.11: icmp_seq=6 ttl=64 time=23.1 ms
64 bytes from 10.0.0.11: icmp_seq=7 ttl=64 time=23.3 ms
^C
--- 10.0.0.11 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6010ms
rtt min/avg/max/mdev = 23.193/25.249/36.180/4.473 ms
```

Once we have confirmed that the promotion rules are correct we can move on to more detailed measurement of communication. First, we start logging the UDP packets to a .pcap file, and stop it with Ctrl+C when the traffic is complete:

```
sudo tcpdump -i any -w UDP_routing1.pcap 'udp'
```

Then, opening a new terminal on the recipient with ***xterm h11***, type the following and wait for the flow to complete, then stop execution with Ctrl+C:

```
cd mininet/custom/D-ITG-2.8.1-r1023/bin
./ITGRecv
```

```
cd mininet/custom/D-ITG-2.8.1-r1023/bin
./ITGSend -T UDP -a 10.0.0.11 -c 100 -C 10 -t 30000 -l sender.log -x receiver.log
```

Then, using ***xterm h5***, in the sender's new terminal we write the following to send a 30-second UDP stream with the D-ITG generator from the sender to the receiver:

Because we want to have detailed information regarding jitter and packet loss, we export a .dat file, via the receiver's ***xterm, which we*** will use for visualization:

```
./ITGDec receiver.log -c 1000
```

This log file contains more information than the simple ping tested earlier. Indicatively it contains 5 columns representing Time (sec), Bitrate, Delay (milliseconds), Jitter (milliseconds) and Packet Loss, while each row represents the different values that all of the above take in relation to time.

```
0.000000 8.000000 0.030749 0.012890 0.000000
1.000000 8.000000 0.040878 0.038958 0.000000
2.000000 7.200000 0.033124 0.019451 0.000000
3.000000 6.400000 0.042278 0.044926 0.000000
4.000000 5.600000 0.106460 0.105517 0.000000
5.000000 8.800000 0.069596 0.047478 0.000000
6.000000 7.200000 0.044916 0.054908 0.000000
7.000000 8.000000 0.012257 0.000956 0.000000
8.000000 8.000000 0.011985 0.001146 0.000000
9.000000 8.000000 0.011839 0.000945 0.000000
10.000000 8.000000 0.012100 0.000993 0.000000
11.000000 7.200000 0.012350 0.001145 0.000000
12.000000 8.000000 0.012868 0.002703 0.000000
13.000000 8.000000 0.012318 0.000688 0.000000
14.000000 8.000000 0.012373 0.001508 0.000000
15.000000 8.000000 0.012359 0.001149 0.000000
16.000000 8.000000 0.012337 0.000485 0.000000
17.000000 8.000000 0.013062 0.000683 0.000000
18.000000 8.000000 0.012595 0.000881 0.000000
19.000000 7.200000 0.012356 0.001048 0.000000
20.000000 8.000000 0.012709 0.001014 0.000000
21.000000 8.000000 0.012681 0.000838 0.000000
22.000000 8.000000 0.012409 0.000858 0.000000
23.000000 8.000000 0.012825 0.000485 0.000000
24.000000 8.000000 0.012405 0.000939 0.000000
25.000000 8.000000 0.012296 0.000913 0.000000
26.000000 8.000000 0.012771 0.001409 0.000000
27.000000 8.000000 0.012311 0.000388 0.000000
28.000000 8.000000 0.012489 0.000661 0.000000
29.000000 8.000000 0.012612 0.000465 0.000000
```

In this way we were able to extract results for the flow created from h5 to h11. In a similar way we can measure valuable information if during the flow we have the original rules deleted and the packets are re-forwarded via new forwarding rules by a new route that ends up in the receiver from end to end. The deletion of forwarding rules and insertion of new ones will be done while the flow generated by the D-ITG traffic generator is executed.

So, in the terminal we opened earlier with ***xterm h11*** (and while in */home/ubuntu/mininet/custom/D-ITG-2.8.1-r1023/bin*), we write and wait for the stream to complete, then stop it with Ctrl+C:

```
./ITGRecv
```

Start a new logging of UDP packets to a .pcap file, and stop it when the traffic is complete:

```
sudo tcpdump -i any -w UDP_routing1_alt.pcap 'udp'
```

Then, on the ***xterm h5*** terminal, we create a 30-second UDP stream with the D-ITG generator from the sender to the receiver:

```
./ITGSend -T UDP -a 10.0.0.11 -c 100 -C 10 -t 30000 -l sender.log -x receiver.log
```

At the same time and while we have created the stream, we go to the terminal we opened earlier, which is located in the */home/ubuntu/* directory where we have placed all the **.sh** files. There we run the bash script file ***del.sh*** which contains commands like ***sudo ovs-ofctl del-flows s#Num to*** delete the original forwarding rules. Without wasting time we also execute the ***rule2_alternate.sh*** file, which contains the new forwarding rules from h5 to h11 from end to beginning via a different path. This file contains the following forwarding rules:

```
sudo ovs-ofctl add-flow s11 dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:0b,actions=output:3
sudo ovs-ofctl add-flow s11 dl_src=00:00:00:00:00:0b,dl_dst=00:00:00:00:00:05,actions=output:1
sudo ovs-ofctl add-flow s9 dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:0b,actions=output:3
sudo ovs-ofctl add-flow s9 dl_src=00:00:00:00:00:0b,dl_dst=00:00:00:00:00:05,actions=output:1
sudo ovs-ofctl add-flow s8 dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:0b,actions=output:4
sudo ovs-ofctl add-flow s8 dl_src=00:00:00:00:00:0b,dl_dst=00:00:00:00:00:05,actions=output:1
sudo ovs-ofctl add-flow s6 dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:0b,actions=output:4
sudo ovs-ofctl add-flow s6 dl_src=00:00:00:00:00:0b,dl_dst=00:00:00:00:00:05,actions=output:2
sudo ovs-ofctl add-flow s5 dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:0b,actions=output:3
sudo ovs-ofctl add-flow s5 dl_src=00:00:00:00:00:0b,dl_dst=00:00:00:00:00:05,actions=output:2
```

After completing the stream in the recipient's ***xterm***, we create a new .dat file:

```
./ITGDec receiver.log -c 1000
```

In which we notice in the last column that **packet loss** is no longer zero everywhere since packets were lost during the deletion of the original rules and the creation of the new ones:

```
0.000000 8.000000 0.012446 0.001303 0.000000
1.000000 8.000000 0.012556 0.001122 0.000000
2.000000 8.000000 0.012642 0.001166 0.000000
3.000000 7.200000 0.047987 0.039665 0.000000
4.000000 5.600000 0.135808 0.133902 0.000000
5.000000 5.600000 0.080400 0.086072 0.000000
6.000000 8.000000 0.048462 0.034606 0.000000
7.000000 8.000000 0.012130 0.000801 0.000000
8.000000 8.000000 0.012057 0.000638 0.000000
9.000000 8.000000 0.012432 0.000531 0.000000
10.000000 7.200000 0.012630 0.001099 0.000000
11.000000 8.000000 0.012970 0.000710 0.000000
12.000000 2.400000 0.011598 0.000944 0.000000
13.000000 0.000000 0.000000 0.000000 0.000000
14.000000 0.000000 0.000000 0.000000 0.000000
15.000000 0.000000 0.000000 0.000000 0.000000
16.000000 0.000000 0.000000 0.000000 0.000000
17.000000 0.000000 0.000000 0.000000 0.000000
18.000000 8.000000 0.017314 0.001735 56.000000
19.000000 8.000000 0.016998 0.001020 0.000000
20.000000 8.000000 0.017186 0.001334 0.000000
21.000000 4.000000 0.160677 0.141202 0.000000
22.000000 7.200000 0.156619 0.096664 0.000000
23.000000 8.000000 0.034245 0.025583 0.000000
24.000000 7.200000 0.045044 0.040834 0.000000
25.000000 8.000000 0.017158 0.000586 0.000000
26.000000 7.200000 0.037069 0.033615 0.000000
27.000000 5.600000 0.106198 0.101270 0.000000
28.000000 8.800000 0.025133 0.014636 0.000000
29.000000 5.600000 0.049807 0.036065 0.000000
30.000000 1.600000 0.239554 0.116725 0.000000
```

Finally, to measure **packet reordering we** will use the tshark tool, which is the terminal version of Wireshark, in combination with the .pcap files we recorded during the creation of the various streams. The "| *wc -l*" only shows the number of packets, not a breakdown of all those that satisfy this filter:

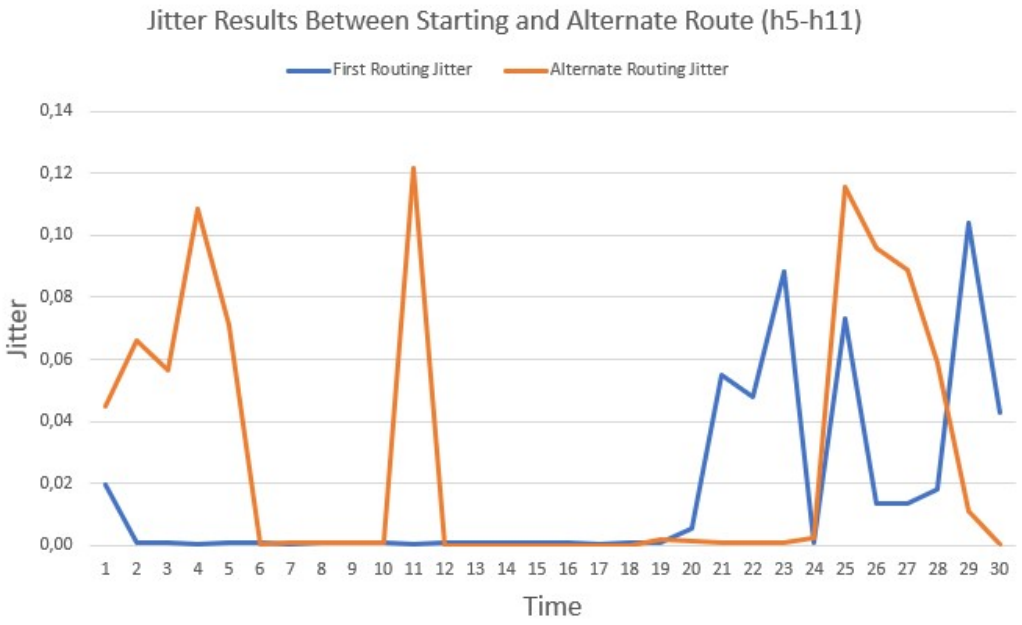
```
sudo tshark -r UDP_routing1.pcap -Y "frame.time_delta < 0" | wc -l
```

In the following section, the rationale for measuring packet reordering in this way will be discussed.

# 3 Commenting on the Results

Below are the visualizations obtained for **jitter** and **packet loss** from the .dat files. More specifically, the graphs will all be annotated together at the end to draw conclusions, since they all seem to follow the same pattern.

**Figure 2.1: Jitter between initial and new forwarding rules (h5-h11)**




**Graph 2.2: Loss of Original and New Promotion Rules Packages (h5-h11)**

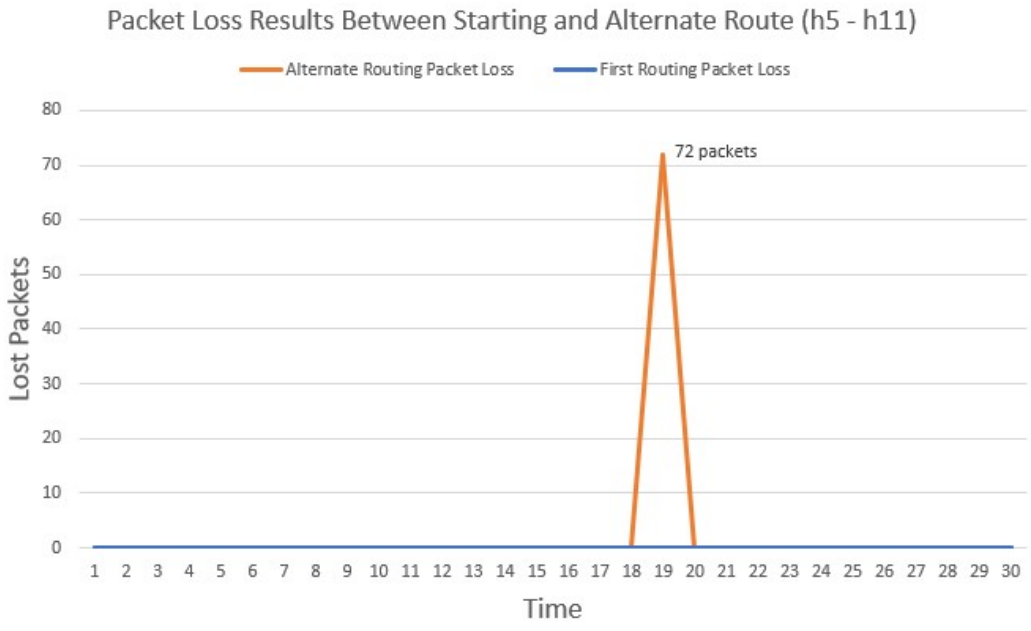


**Figure 2.3: Jitter between initial and new forwarding rules (h2-h8)**

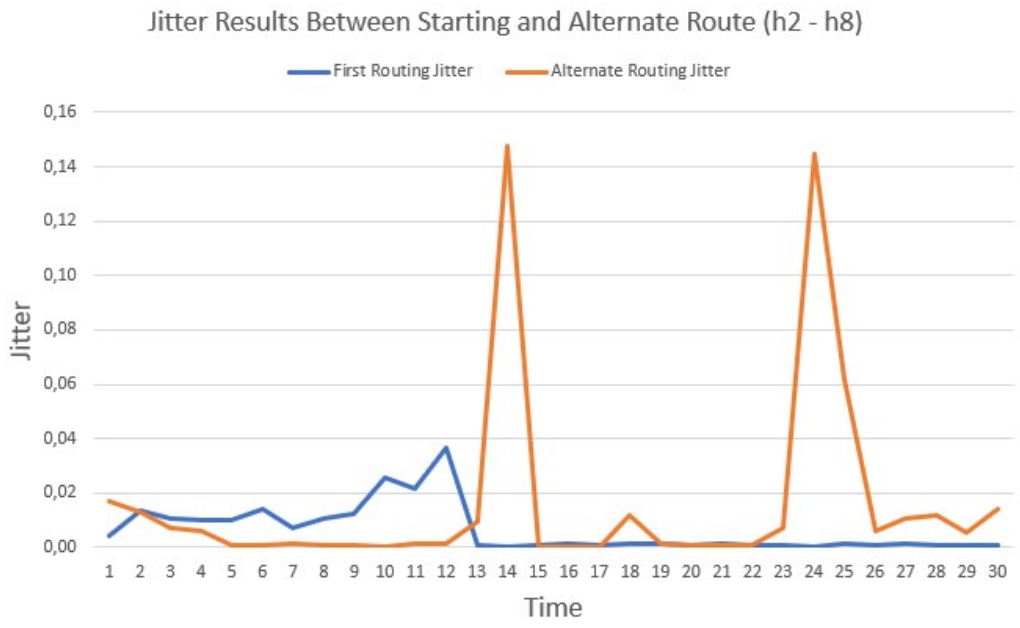



**Figure 2.4: Loss of Original and New Promotion Rules Packages (h2-h8)**

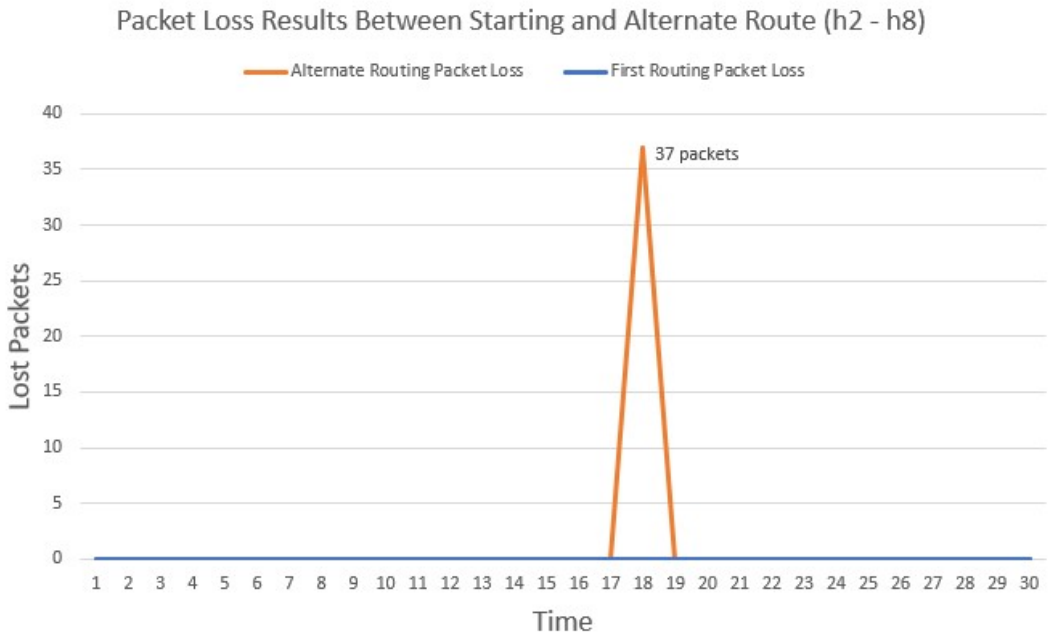


**Figure 2.5: Jitter between initial and new forwarding rules (h1-h5)**

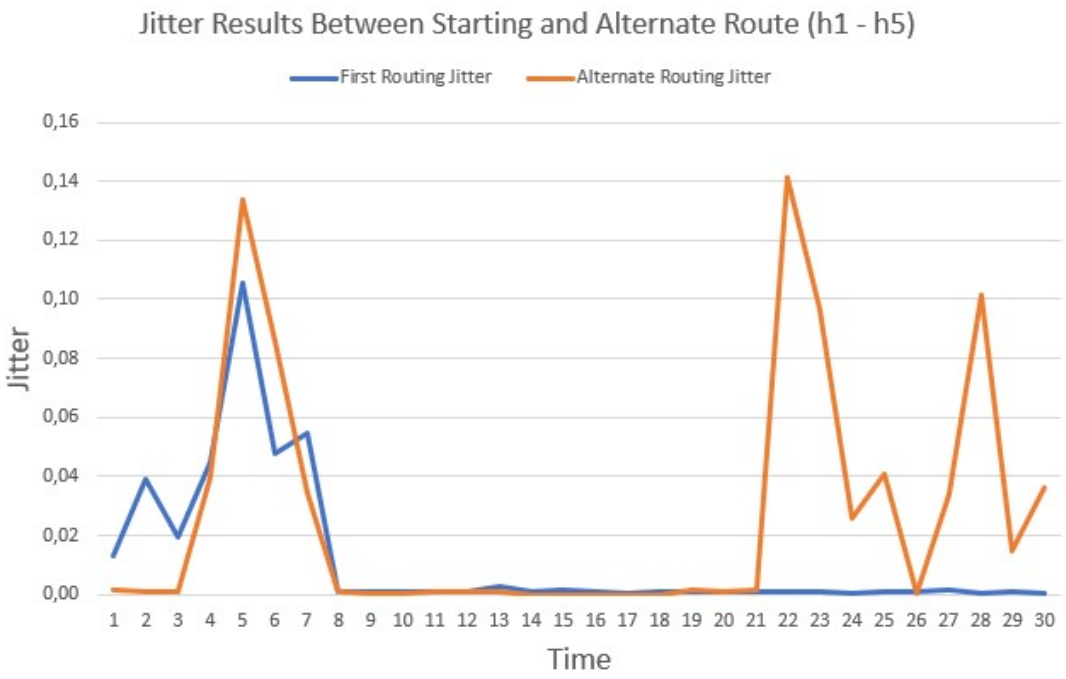



**Figure 2.6: Loss of Original and New Promotion Rules Packages (h1-h5)**

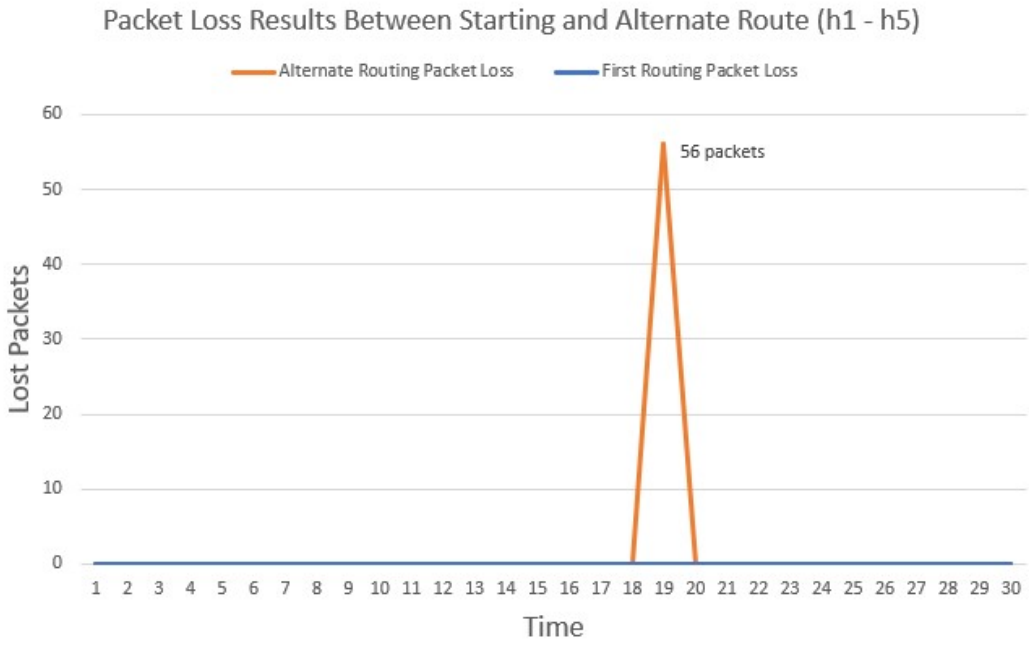


**Figure 2.7: Jitter between initial and new forwarding rules (h4-h10)**

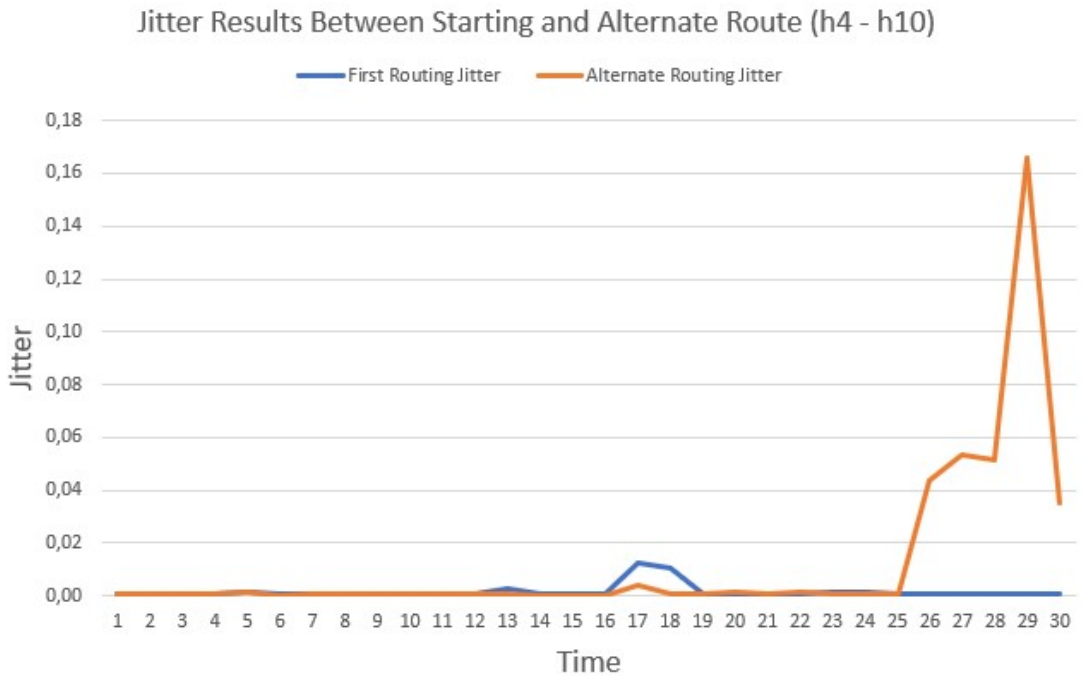
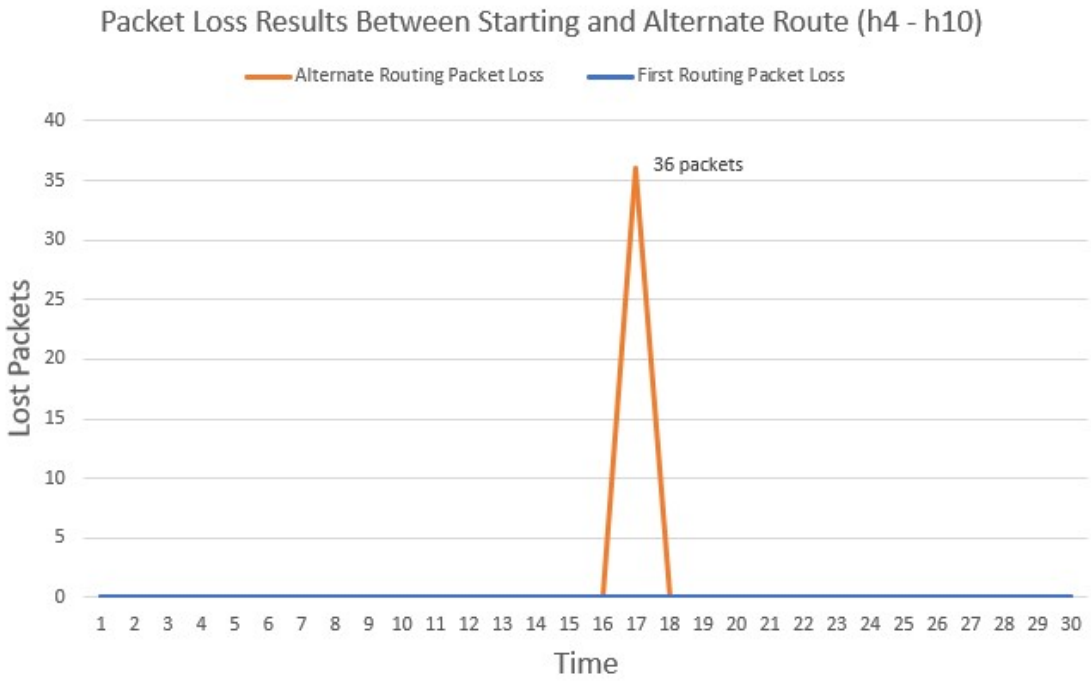



**Figure 2.8: Loss of Original and New Promotion Rules Packages (h4-h10)**



Jitter is the variation in the delay of packet arrival times on a network. It is essentially the difference between the actual arrival times of packets at the destination compared to their expected arrival times. In other words, jitter is the inconsistency in delay between successive packets.

In all the above diagrams we can observe that from the moment the packet loss on the alternate path starts, the jitter becomes more active and has a larger variation. This means that the stable arrival time of packets after the initial forwarding rules are deleted starts to be disturbed since packets are lost when rules are changed.

Therefore, jitter variation can affect the quality and reliability of real-time applications, such as voice over IP (VoIP) calls or video streams, where maintaining a constant packet arrival time is important for smooth playback or communication.

Regarding packet reordering and its measurement, since the UDP packets do not contain sequence number in their header in order to calculate whether they are in-order or out-of-order we chose to filter the file based on the "frame.time_delta" rule, which could be a possible indication of packet reordering. With this logic, I thought that a negative value might highlight packets that are out-of-order, since it represents the time difference between the current packet and the previous packet recorded. However, I recognize that this approach may not be a correct metric for packet reordering, as negative values could be due to network delays, capture problems, or inaccuracies in timestamping.

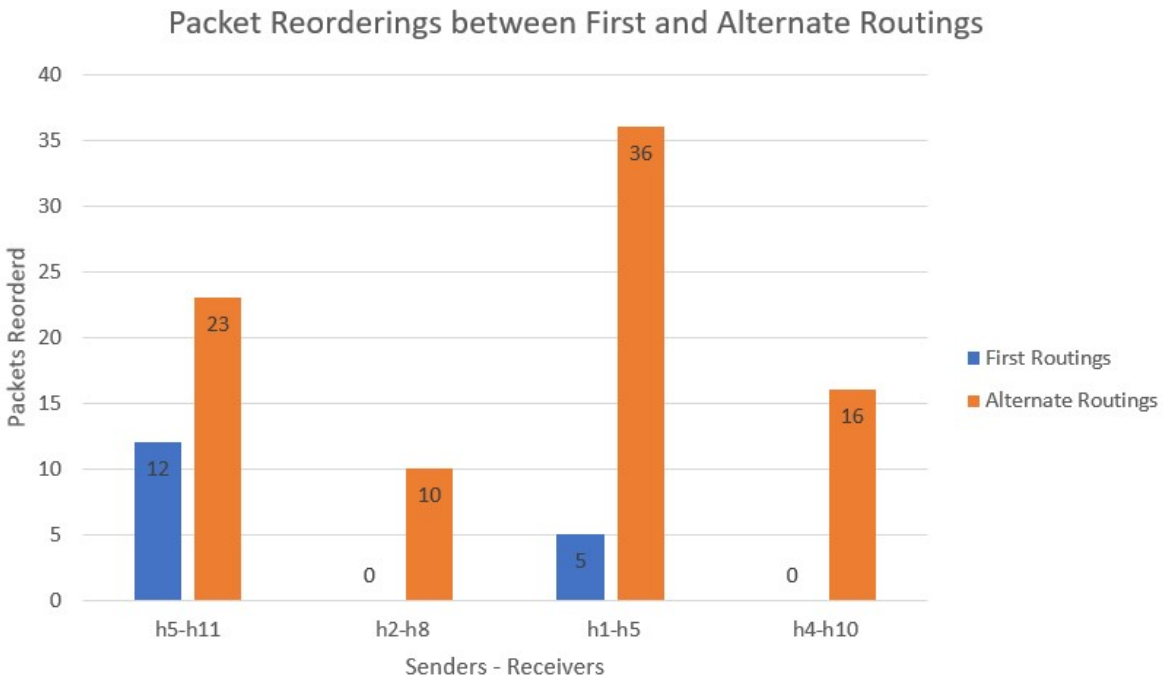
**Table 2.1: Table of the Change in the Order of Arrival of Packages at the Consignee**

packet reordering

| Nodes | | Routings | |
|---|---|---|---|
| From | To | First | Alternate |
| h5 | h11 | 12 | 23 |
| h2 | h8 | 0 | 10 |
| h1 | h5 | 5 | 36 |
| h4 | h10 | 0 | 16 |

***Comments*:** *Table 2.1,* contains the resulting packet reordering between the original and alternate paths for all different sender-receiver pairs.

**Figure 2.9: Change in the Order of Arrival of Packages at the Consignee**




***Comment*:** In *Figure 2.9*, it is clearly shown that packet reordering increase after deleting the original packets and introducing new forwarding rules.

Packet reordering is essentially a situation in which packets arrive at the destination in a different order from their original order. In a network, packets can take different paths and encounter different delays, resulting in packets arriving at their destination in a different order than the one in which they were sent. Deleting the original forwarding rules for a short period of time and introducing new alternate path rules, combined with the greater variation in jitter, contributes to the reordering of packets and delay in their arrival times.