PROJECT REPORT

P4AUTH IN HULA

SANDEEP L CS20BTECH11044

SHANTANU PANDEY CS20BTECH11046

MENTOR RANJITHA K

OVERVIEW OF RESEARCH PAPERS

P4AUTH

In recent years, there has been increased use of Software-Defined Networking(SDN) & Programmable Data Planes(PDPs). This lead to a high risk of adversarial manipulation of the packets in the network.

The paper discusses the problem that an adversary can disturb the flow of packets in the network by attacking the data plane of the switch. There are many vulnerable places in the switch where the adversary can attack. At the high level, s/he can cause many problems, like changing the route of the packets by modifying the registers in the switch data plane, changing the entries in the table, etc.

P4Auth is introduced to stop these adversarial attacks. Here, we are using P4Auth in HULA Logic.

HULA-Algorithm

Nowadays, Data-centers have invested a lot to have multi-rooted topologies like Fat-Tress and Leaf-Spine to provide large bisection bandwidth. Effectively balancing traffic load across multiple paths in the data plane is critical to fully utilizing the available bisection bandwidth

HULA, a data-plane load-balancing algorithm, solves the Load balancing issue of Data Centers which uses multi-rooted topologies by using a probe to track congestion on all paths to a destination with the help of neighboring switches and to adapt quickly in case of changes in the network (asymmetric topology). It is designed for programmable switches, so it does not need custom hardware and hence saves a lot of money.

OVERVIEW OF DIRECTORIES

ONLY HULA NO P4AUTH

In this directory, we have a topology where two hosts (namely h1, h2) are connected by six switches (namely S1, S2, S3, S4, S5, S6). Hula logic is implemented in this directory for load balancing in switches, but there is NO “P4Auth (for security)”. Except for switch S6, every switch has the same code, and switch S6 works as an adversary in this topology. But switch S6 is dormant at the beginning, so we have to give some commands, ‘Run-time commands, for table forwarding so it can start working.

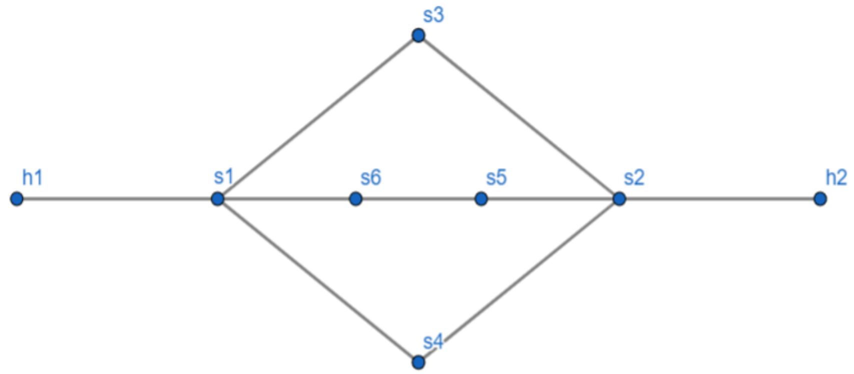


Figure 1: Topology of the network in Only HULA No P4Auth

CRC32 MODEL WITHOUT ADVERSARY

In this directory, the topology is the same as the topology in (Only HULA No P4Auth), except for two changes. One is that there is no switch S6(hence no adversary). The other one is that there is P4Auth Authentication logic using CRC32 Algorithm installed in every switch for security purposes.

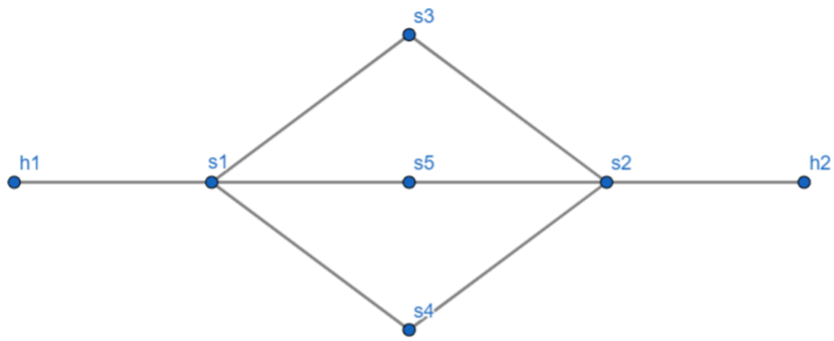


Figure 2: Topology of the network in CRC32 Model Without Adversary

CRC32 MODEL WITH ADVERSARY

In this directory, the topology is the same as the topology in (Only HULA No P4Auth), except for just one change, i.e., there P4Auth Authentication logic using CRC32 Algorithm in every switch except S6(Adversary) for securing the network. This is a combination of the above two directories.

- ➔ In the first directory, an adversary is manipulating the packets.
- ➔ In the second directory, there is P4Auth which ensures the security of each switch's data planes.

Here we are testing the security of the network while the adversary manipulates the packets in the network.

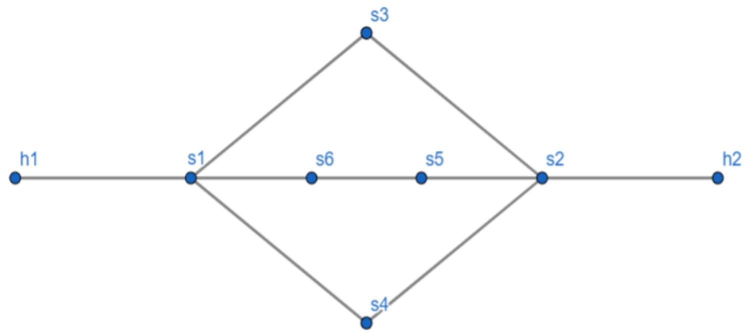


Figure 3: Topology of the network in CRC32 Model With Adversary

OVERVIEW OF FILES

IMPORTANT FILES

- **new_controller.py:** this file is used to configure the data plane. It installs hula logic/P4Auth logic into the switches and calculates RTT time for different switches.
- **probe.py:** This file sends probe packets (of Hula logic) from the hosts' terminals.
- **send.py:** This file sends any message from one host to another host. The message is sent 1000 times to study packet flow and other observations.
- **receive.py:** This file receives packets/messages from another host. Please note that the receiver and the sender should run simultaneously on different hosts' terminals to send and receive packets/messages.
- **packet_counter.py:** The file stores all the information (like checksum, dst_ip,src_ip) we receive from a packet/message into a file packets_couts.txt
- **Rtt_calculator.py:** This uses the statistics received from new_controller.py to calculate the average RTT time.
- **test.py:** woThis
- **timer.py:** This file calculates the time taken to run a process like adding table entries, changing registers, etc.
- **topology.json:** This file contains network structure, i.e., how the switches are connected to each other and to the hosts.

- ➢**keys.json:** This file gets generated at runtime and stores the port keys (secret keys) required for the secured communication (P4Auth) between the switches and to detect any manipulation of the packets.
- ➢**switch1.p4:** This file has the P4 codes of switch logic with Hula logic, which gets installed in every switch from S1 to S5.
- ➢**switch6.p4:** This file has the P4 codes without Hula logic and has code to make it work like an adversary.
- ➢**Makefile:** It starts the mininet by generating the network's topology and activating the switches.

DELETED FILES

- ➢**benchmark.py:** To evaluate the port utilization of each switch under load, which snapshots the state of various registers on each switch at runtime. This file has codes from the experiment of Rachit Nigam, and we do not need them for our experiments.
- ➢**cli_commands/command:** This file has table entry commands for switches. We already use HULA logic for this, so we don't need it.
- ➢**controller.py/controller1.py/orig_controller.py:** The *new_controller.py* file is the latest and updated version of the *controller*. So, we can use *new_controller.py,* and there is no use for the other controllers which were not working as we need them to work for our experiment. Hence, deleted.
- ➢**original_switch1/p4auth_hulaswitch/switch1_old_p4/switch_temp_halt/switch.p4.save1/switch2/switch6.p4.save/switch1_hula_alone:** These switches are the modifications of switches *switch1.p4* and *switch6.p4,* and we are not using them now. Hence, deleted.

- ➢ **out.p4:** This is generated when we run test.py so we can delete it.
- ➢ **topology-orig.json:** This file is deleted since the network's topology is changed, and we are using the updated file topology.json.

EXPERIMENTS & OBSERVATIONS

NOTE: The procedure to run the following experiments is provided in the documentation here.

ONLY HULA NO P4AUTH (WITH ADVERSARY)

EXPERIMENTS

- ➢ We used this directory as a learning ground to learn about how to run the code and learn about Hula logic and how it is implemented inside the switches and how it uses probes to calculate the keep track of best hops and of path utilizations.
- ➢ We send the packets from one host to another to see how the adversary is attacking.
- ➢ We changed the topology at run-time to see if Hula was able to adopt the change in network topology.

OBSERVATIONS

- ➔ All the information that we have collected about different files which are mentioned above are mainly due to experimenting through the files and codes of this directory.
- ➔ As we sent probes, the adversary was modifying the path utilization to divert the traffic to itself. It will set the path_util=0 so that the path will look free of any congestion, but in reality, it will be flooded with traffic.

```
action packet_forward(egressSpec_t port){
    if(hdr.hula.isValid()){
        hdr.hula.path_util = 0;
    }
    standard_metadata.egress_spec = port;
}
```

In the above code snippet of Switch S6, we can see how it is changing the path utilization to zero to divert traffic to its route.

➔ Even on changing the network's topology on run time, Hula quickly adapted to the change and diverted the packets using other routes.

CRC32 MODEL WITHOUT ADVERSARY

EXPERIMENTS

➢ In this experiment, since there is no adversary, there is no manipulation of the packets; still, there is P4Auth to block any kind of attack from an adversary.

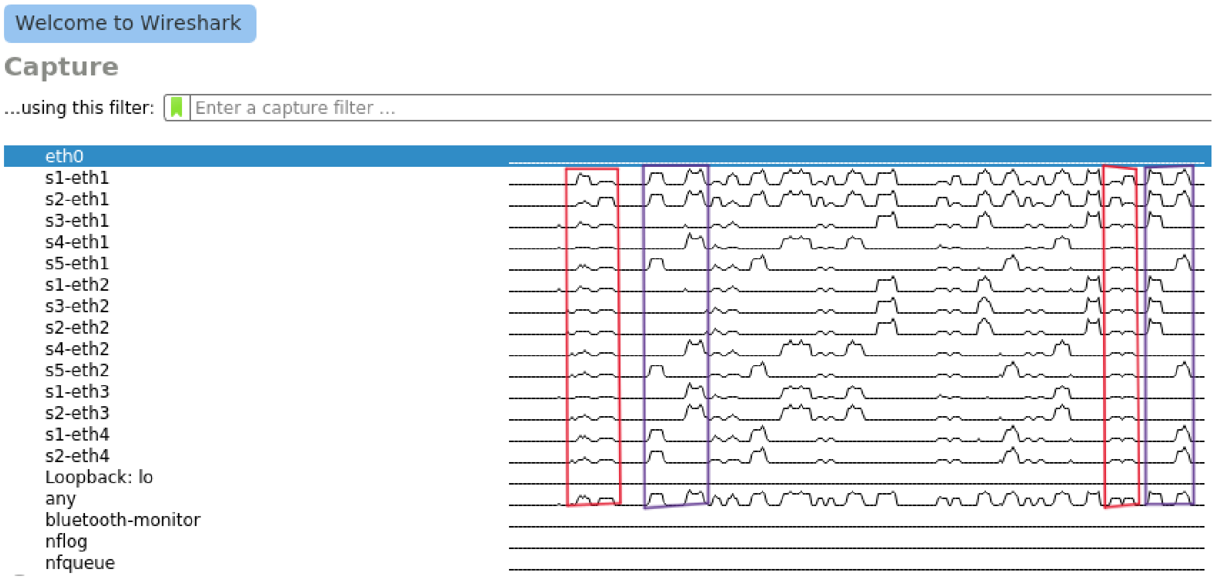
➢ We used this experiment to observe the RTT of the packets and compare these with the ones calculated in Only HULA without P4Auth experiment(Above experiment).

OBSERVATIONS

➔ There are three paths between the hosts *h1* and *h2,* as shown in Figure 2. As all three paths are identical, hula logic chooses any of the three paths uniformly at random each time we run the *probe* packets. And when we send messages from one host to the other, the packet will travel with the chosen path. In the screenshot below, the graph inside the **RED** rectangle shows the probes from hosts *h1* & *h2,* and the graph in the **PURPLE** rectangle shows the flow of the packet first part *h1* -> *h2* and second part *h2* -> *h1* within the same rectangle. If we compare it with the other **RED** and **PURPLE** rectangle pair, the paths chosen by the packets are different.

	From	Through	To
First **PURPLE** rectangle	h1	s5	h2
	h2	s4	h1
Second **PURPLE** rectangle	h1	s3	h2
	h2	s5	h1

Note: In each rectangle, the first peak(left part) is probe or message from h1, and the second peak(right peak) is probe or message from h2.



➔ There is a significant difference in the average RTT calculated. We can see the increase in RTT due to P4Auth calculating the checksum/digest of CRC32 model

```
10 iterations done
Average running time : 933231.18
p4@p4:~/Downloads/P4-Auth-siphash-main/only_hula_no_p4_auth$
p4@p4:~/Downloads/P4-Auth-siphash-main/only_hula_no_p4_auth$
```

CRC32 MODEL WITH ADVERSARY

EXPERIMENTS

- ➢ Here, we did the same experiments as the above one, but here there is an adversary in the switch S6.
- ➢ We sent probe packets from the hosts *h1 & h2,* which calculates the best path for sending the packets. After that, we sent packets from both *h1 -> h2* and *h2 -> h1* and observed what paths were chosen by the packets.

OBSERVATIONS

- ➢ In the below screenshot, the graph inside the **RED** rectangle is the graph of *probes.* We can see that the probe packet is not captured in S5 from S6 since P4Auth detects the manipulation of the packet from the adversary in S6. Likewise, it will not reach from S6 to S1 also.
- ➢ Since the probe packets are not reaching S1 or S5 from S6, HULA will not consider the path, including the switch S6, and calculates the best possible path from the remaining two paths. For the same reason, we can see that the messages from both *h1 -> h2* and *h2 -> h1* will not take the path including S6 and chooses a path uniformly at random from the other two paths since both the paths are identical.

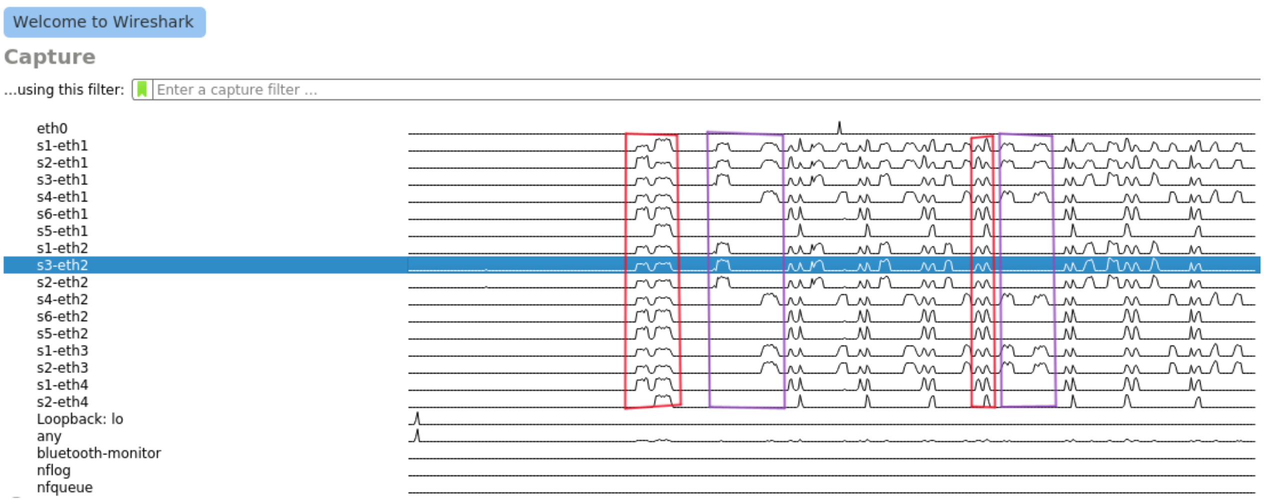
  In the below table, we can see the two examples.

➔

	From	Through	To
First **PURPLE**	h1	s3	h2

rectangle	h2	s4	h1
Second **PURPLE** rectangle	h1	s4	h2
	h2	s4	h1

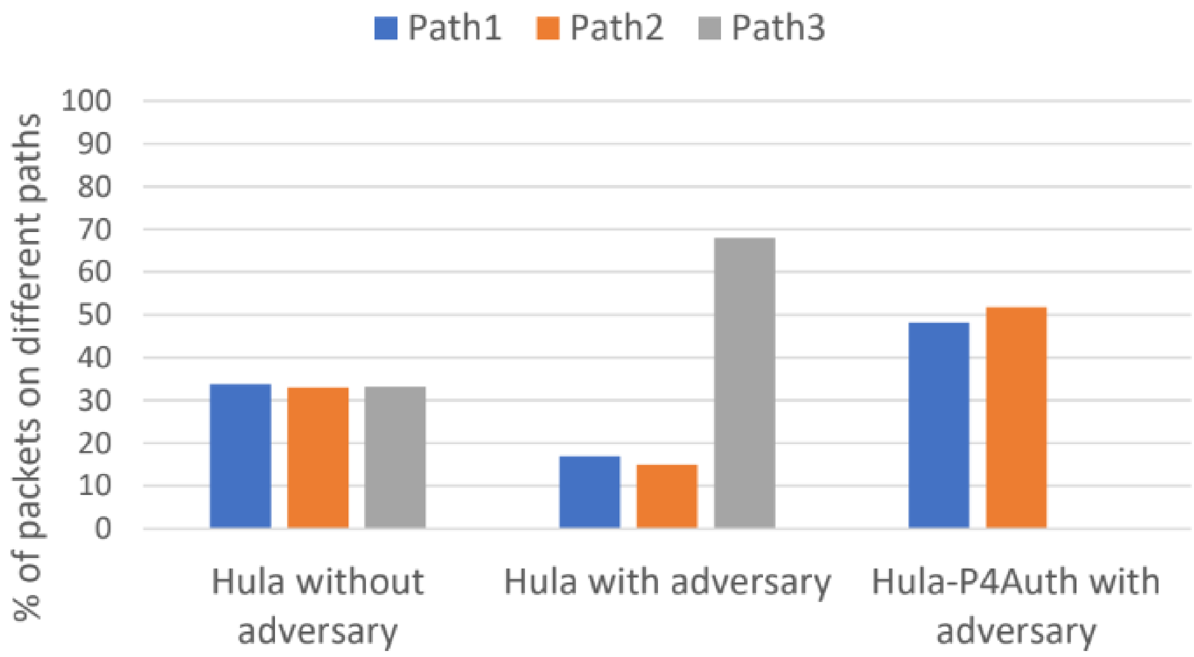
Note: In each rectangle, the first peak(left part) is probe or message from h1, and the second peak(right peak) is probe or message from h2.



EXPERIMENTS & OBSERVATIONS

Sir asked us whether we did the below experiment (while giving the presentation).

So we did the below experiment and found similar observations.



Graph 1

EXPERIMENTS

We conducted the above-mentioned three experiments as it is.
But this time, we ran the probes continuously by opening the h1 and h2 terminals again(only for probes) while sending the packets(messages) between hosts. Since the *probe* is calculating the best possible path dynamically at run time, the path of the packets(messages) was changing continuously, and the observations made among the three directories are shown below.

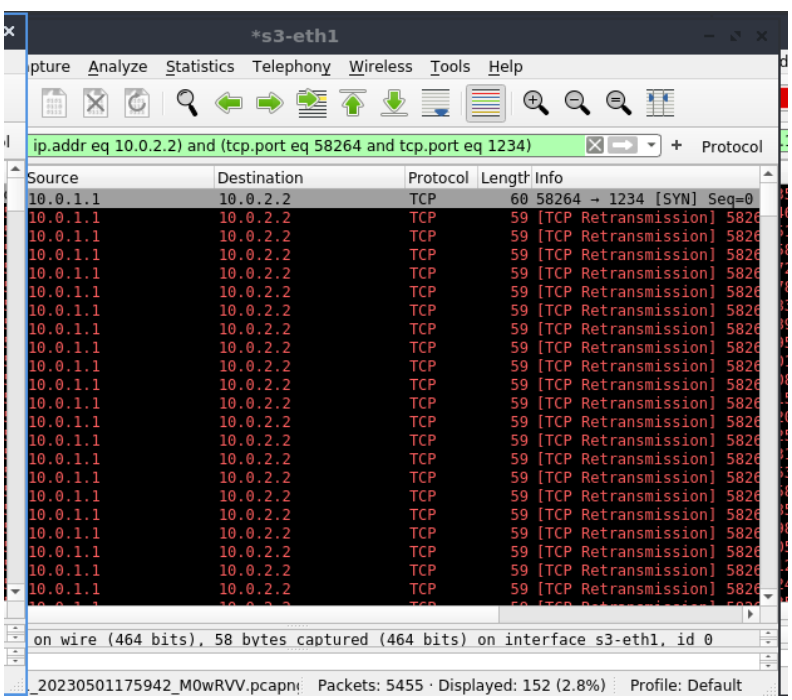
NOTE: The number of packets(messages excluding probes) is shown in the lower right corner of the further images in the field “Displayed”.

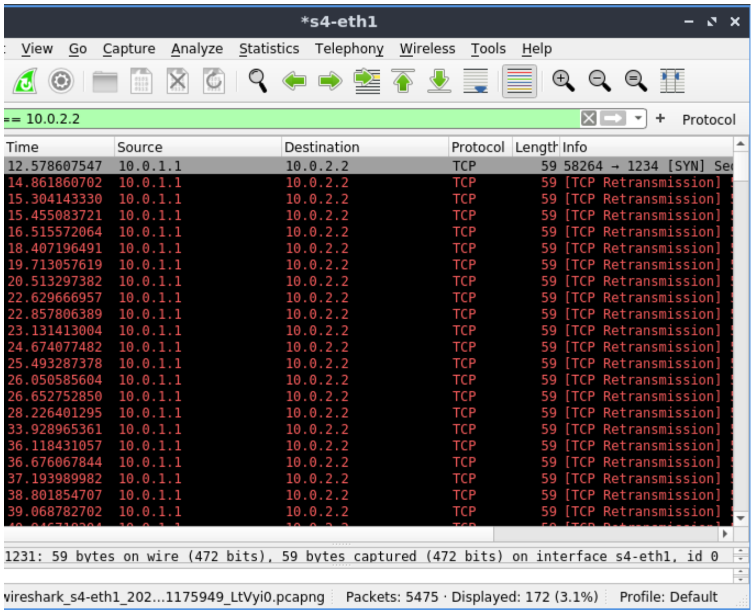
OBSERVATIONS

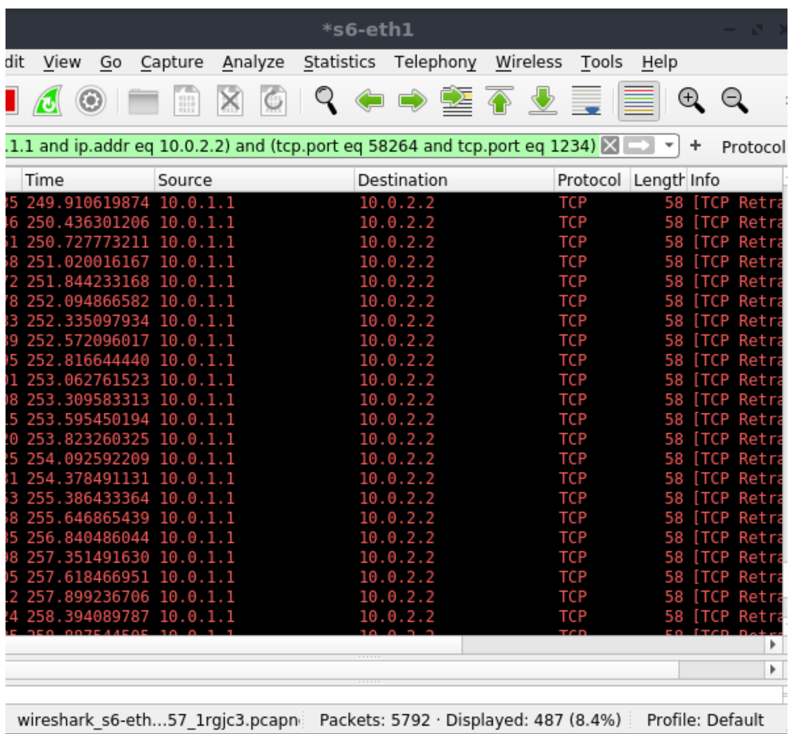
ONLY HULA NO P4AUTH (WITH ADVERSARY)

- ➔ This was the second experiment in Graph 1.
- ➔ In the below table, we can see that since there is an adversary and the adversary will try to manipulate the *probes* that there is very less traffic going through the adversary, so choose that path (here path containing S6). So, in the table below, we can see that most of the packets(messages) were sent from *h1 -> h2* through the path containing S6, and the remaining packets(messages) were sent through S3 and S4 almost equally since they both are identical.

	s3	s4	s6
Percentage of packets transmitted	152/811*100 = 18.74 %	172/811*100 = 21.20 %	487/811*100 = 60.04 %





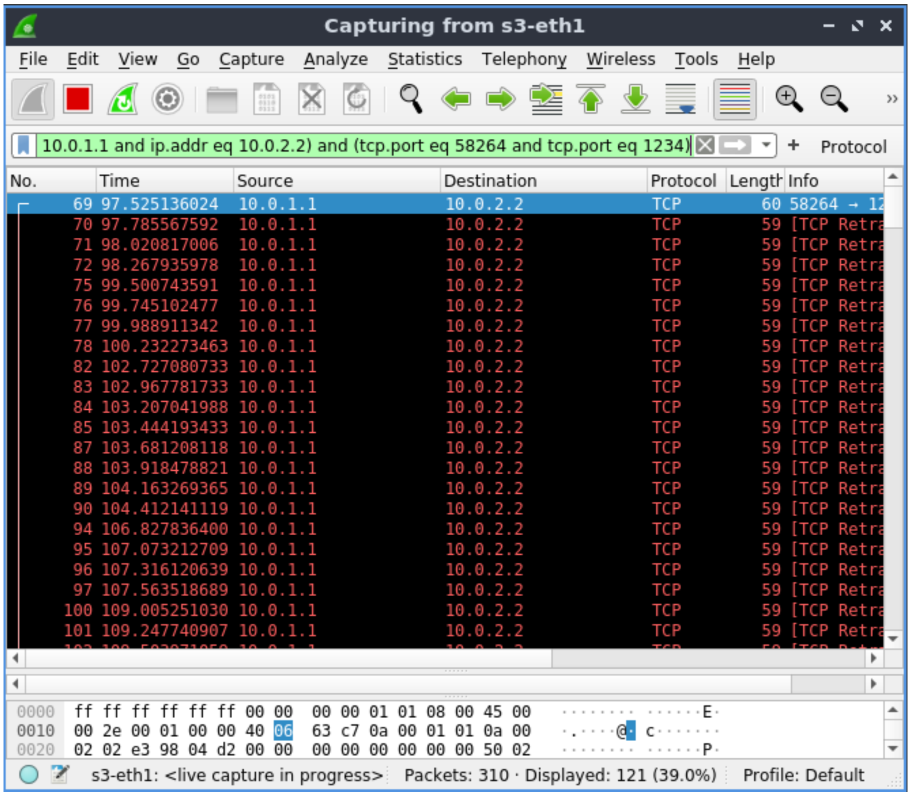


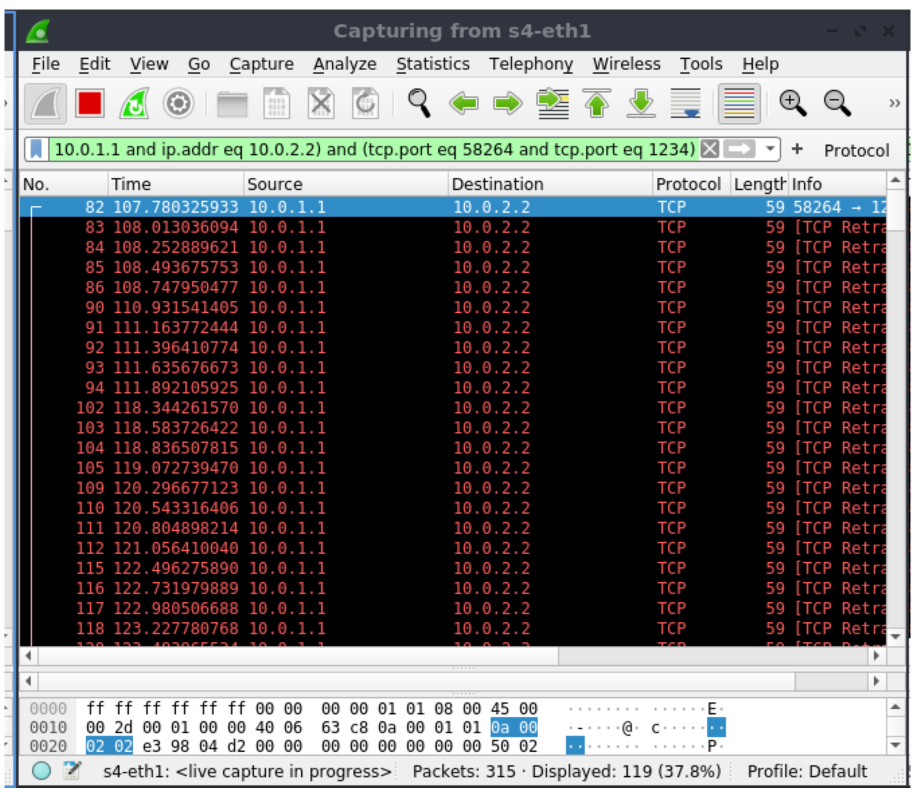
CRC32 MODEL WITHOUT ADVERSARY

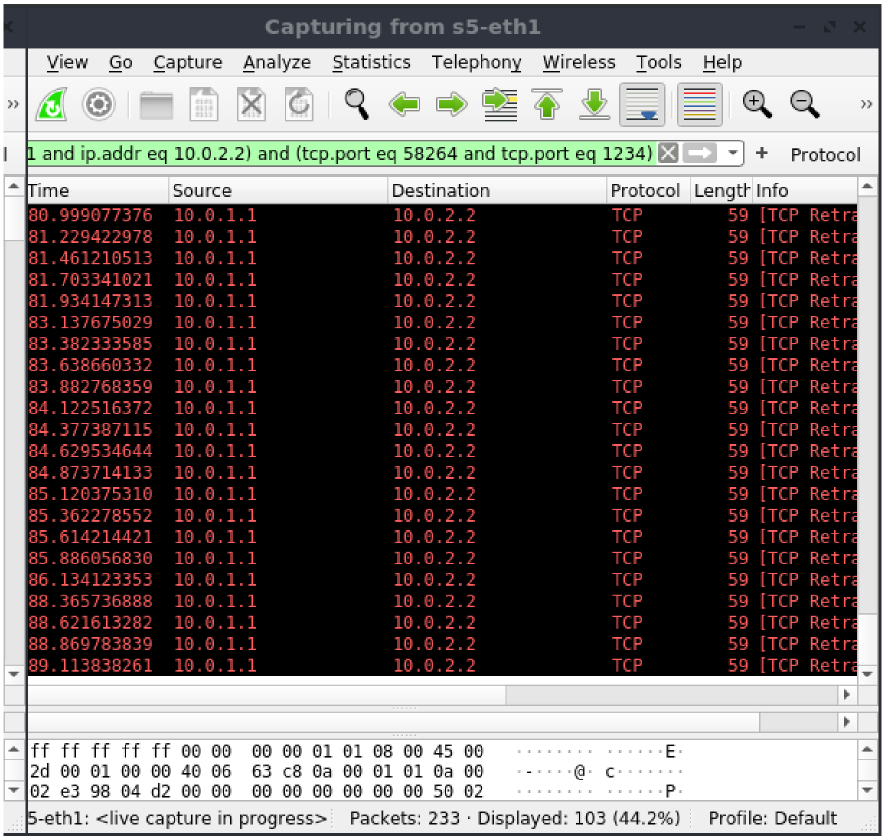
➔ This was the first experiment in Graph 1.

➔ In the below table, we can see that since there was no adversary and all the paths(here switches S3, S4, S5) are identical, there will almost be equal no of packets through each path.

	s3	s4	s5
Percentage of packets transmitted	121/343*100 = 35.27 %	119/343*100 = 34.69 %	103/343*100 = 30.02 %


Capturing from s3-eth1
10.0.1.1 and ip.addr eq 10.0.2.2) and (tcp.port eq 58264 and tcp.port eq 1234)
s3-eth1: <live capture in progress> Packets: 310 · Displayed: 121 (39.0%) Profile: Default

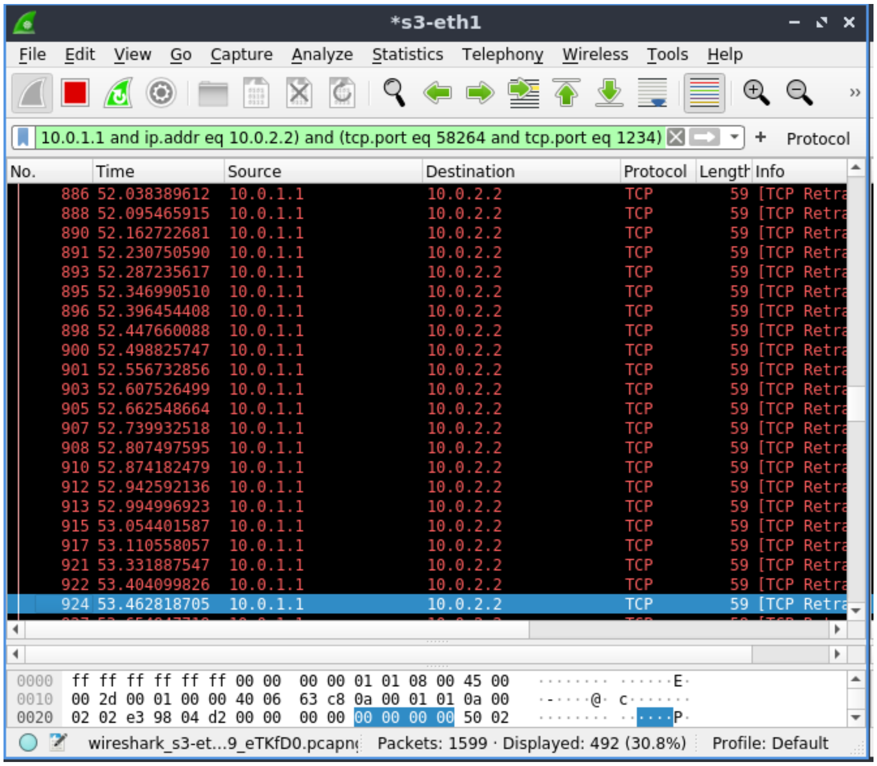

Capturing from s4-eth1
10.0.1.1 and ip.addr eq 10.0.2.2) and (tcp.port eq 58264 and tcp.port eq 1234)
s4-eth1: <live capture in progress> Packets: 315 · Displayed: 119 (37.8%) Profile: Default

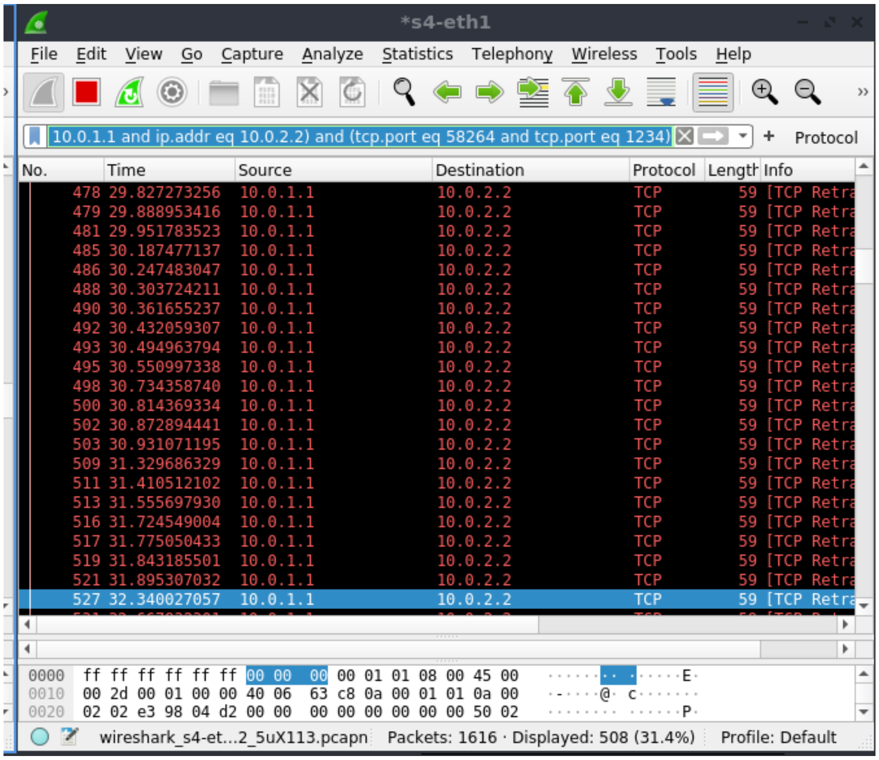


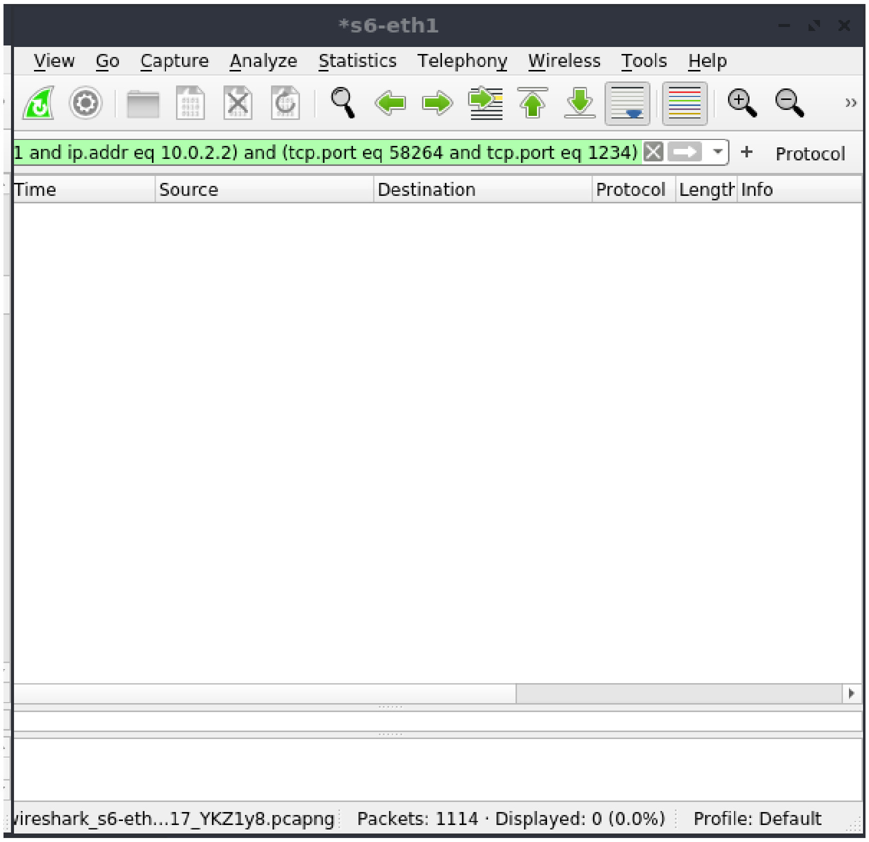
CRC32 MODEL WITH ADVERSARY

- This was the third experiment in Graph 1.
- In the below table, we can see that there is an adversary, and also we have P4Auth. The adversary tries to manipulate the *probe* packets from switch S6 and the P4Auth installed in other switches detects this manipulation and doesn't include those probe packets while calculating the best path using HULA logic. As a result, there will not be any flow of packets(messages) through the adversary(here, S6). And the packets will be sent through any of the other two paths. Since they are identical, almost equal number of packets(messages) will pass through switches S3 and S4.

	s3	s4	s6
Percentage of packets transmitted	492/1000*100 = 49.2 %	508/1000*100 = 50.8 %	0/1000*100 = 0 %







CONCLUSION

- ➢ The Hula Logic works very efficiently with the help of a continuous stream of probes. It never fails to find the best path for every end-to-end communication. It handles changes in topology, asymmetry, and failure of switches with ease. Hence it has the potential to be used extensively in Data centers.
- ➢ Although HULA is a very effective Load Balancer, it also opens up a weak point in the system that can be targeted by an adversary, and that can lead to improper load balancing, which results are not in favor of data centers in many ways.
- ➢ P4Auth has a lot of potentials to provide security in the data plane of a network. One of its potentials is to combine it with HULA logic to secure its weak point by using a CRC32 model.
- ➢ P4Auth combined with Hula is able to keep away the adversarial attacks that can disrupt the load balancing.
- ➢ P4Auth is adding an additional time overhead in RTTs, which is there due to the verification of digest (CRC32).

Pending experiment:

There is an experiment that we had to perform where we had to calculate the difference between the time is taken to read or write table entries and register entries with Hula only and with P4Auth & Hula

To do this experiment, we found out that we have to use “extern” and even if we implement the “extern” to calculate the time take we also have to modify the P4 compiler code to print the time on the console that is calculated inside the switch which is being run using P4 code. We tried to learn about the extern and P4 compiler, but we were not able to manage the time needed to go deep into it.