Report of Mininet Project

Edoardo Coli

July 2023

1 Introduction

The objective of this report is to investigate the multi-connection capability of a cluster network through the utilization of Mininet, a popular open-source network emulator. Specifically, I will focus on measuring the performance and throughput of the network under varying connection scenarios using iperf, a widely adopted tool for network performance testing. Within the report I will also discuss how to use Python to recreate a network topology using the Mininet API. Those API provides a comprehensive set of classes and methods that enable fine-grained control over network topologies and network elements such as hosts, switches, and links.

In the following sections of this report, I will outline the methodology employed using Mininet emulated setup and the experimental setup for real-world comparison. In the end I will discuss the results obtained and provide a comparative analysis.

2 Settings

The experiment concerns: The real testbench that is a physical hardware cluster; The emulated testbench to replicate the behavior in a software-based environment. By presenting the real and emulated testbenches side by side, I aim to show Mininet force points and limitations.

2.1 Features physical cluster (Steffe)

Before start this is a detailed overview of the hardware specifications of the cluster, Rock4+ nodes and the TP-Link switch, highlighting their key features and capabilities:

1. Rock4+ Single-Board Computers:
    - CPU: Rockchip RK3399 Hexa-core processor (Dual-core Cortex-A72 and Quad-core Cortex-A53)
    - GPU: Mali-T860MP4
    - RAM: 4GB LPDDR4 dual channel
    - Connectivity: Gigabit Ethernet (POE)
    - Operating System: Linux-based distribution
2. TP-Link Switch (Model: TL-SG1428PE):
    - Ports: 26x 10/100/1000 Mbps RJ45

- PoE Ports: 24x PoE (802.3at/af)
- Power Budget: Up to 350W

The network topology of the cluster has 21 nodes, where one node is designated as the master (referred to as "steffe0") and the remaining nodes are connected using a switch, can be described as a star topology. In this topology, the master node acts as the central hub or controller, while the switch serves as a central point for connecting all the other nodes. In the setup, the master node “steffe0” is directly connected to the switch, using an Ethernet cable. The switch acts as a central point of connectivity and provides multiple ports to accommodate the remaining 20 nodes. Each of the 20 nodes is connected to the switch using individual Ethernet cables, with each cable connected to an available port on the switch is provided a 2 Gbps bandwidth, 1 Gbps in UL and the 1 Gbps in DL.

2.2 Features emulated cluster

The overview of the emulated network concerns which components have been created within Mininet. 21 nodes hosts were created to recreate the physical network, as well as physical links and switches. The topology configuration has been simplified and placed in a special configuration file *SteffeCluster.conf*. The link creation parameters are specified within the file, in this regard we have links between the switch and the various hosts with a bandwidth of GitHub repository*.

3 Data collection

For the data collection, the same test procedure was carried out for both testbenches. What we want to analyze is the ability of the switch to divide the bandwidth between multiple requesting nodes, as if to simulate a massive sending of more data by the network to a centralized controller. The iperf software was used to perform the measurements. An iperf server is started, as a process, inside the master node using the command *'iperf -s -P 20'*. The nodes of the network connect as clients to the master through the command *'iperf -c steffe0' -t 5*. The -P flag in iperf parameters means that the server can handle up to N connections in parallel. This parameter

* Can’t be configurea in Mininet standard Switch a switch with a bandwidth greater than 1000Mbps.

becomes necessary when we want to try to test the flow of data simultaneously from each host of the network, in this case 20.

The type of measurement that performs iperf consists, in the absence of explicitly defined parameters, to communicate as much data as possible with the server node. The data we are going to capture will be given by measurements implemented with a number of nodes increasing starting from one. Everything has been saved within a file *X_to1.test* (present in the *dataset.zip* file) and is built by appending the output of the individual clients in it. With the use of Bash 'awk' command, it was possible to analyze the data collected within the output file.

In the following subsections we can find the data tables.

3.1 Physical outcome

The hosts subset, incremental of cardinality, was chosen starting from the first node and adding the subsequent ones. For each order of cardinality, 100 measurements were made in a time of 5 seconds using the parameter t in the iperf command. The data that are shown in the table report for each node the arithmetic mean of the hundred values observed.

To launch the processes in parallel from the cluster master node, is used the following Bash+Parallel command:

```
for i in {1..100}; do
        parallel -p 20 --nonall --slf hosts.list 'iperf -c
steffe0 -t 5' >> X_to1.test;
done
```

	HOST INVOLVED																			
HOST NAME	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
STEFFE1	938	479	325	249	198	167	141	122	113	207	285	366	419	486	524	588	621	672	701	679
STEFFE2	0	467	322	246	195	164	138	120	109	101	92	85	80	75	68	65	61	58	55	107
STEFFE3	0	0	323	244	198	163	138	119	107	98	90	85	77	72	67	64	61	59	57	54
STEFFE4	0	0	0	241	197	169	140	119	107	97	89	83	76	71	67	64	62	58	55	54
STEFFE5	0	0	0	0	201	169	141	120	107	97	88	82	76	72	67	64	61	58	56	54
STEFFE6	0	0	0	0	0	166	142	119	107	97	89	83	76	71	67	64	62	58	56	55
STEFFE7	0	0	0	0	0	0	141	120	107	97	88	83	77	72	67	65	62	61	57	55
STEFFE8	0	0	0	0	0	0	0	119	106	97	89	83	76	72	68	65	65	60	57	55
STEFFE9	0	0	0	0	0	0	0	0	108	98	89	83	78	72	70	66	63	61	59	57
STEFFE10	0	0	0	0	0	0	0	0	0	100	94	92	86	83	75	78	74	75	75	71
STEFFE11	0	0	0	0	0	0	0	0	0	0	93	88	82	82	76	77	73	74	68	67
STEFFE12	0	0	0	0	0	0	0	0	0	0	0	86	80	77	73	73	70	70	68	64
STEFFE13	0	0	0	0	0	0	0	0	0	0	0	0	79	76	71	69	69	68	62	61
STEFFE14	0	0	0	0	0	0	0	0	0	0	0	0	0	74	70	68	65	63	60	62
STEFFE15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	68	68	62	61	59	57
STEFFE16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65	61	59	57	54
STEFFE17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	62	58	56	53
STEFFE18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	58	55	52
STEFFE19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	56	54
STEFFE20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	53

3.2 Mininet outcome

The hosts subset, incremental of cardinality, was chosen starting from the first node and adding the subsequent ones. For each order of cardinality, 100 measurements were made in a time of 5 seconds using the parameter t in the iperf command. The data that are shown in the table report for each node the arithmetic mean of the hundred values observed.

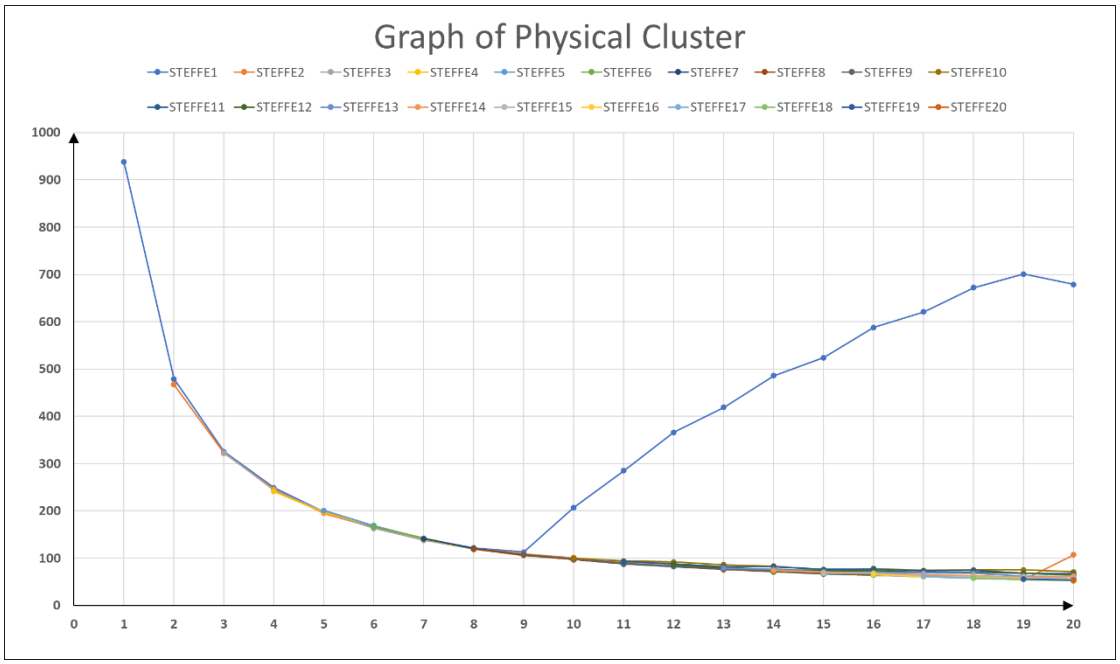
To launch the processes in parallel in Mininet via the python interface is used the following code:

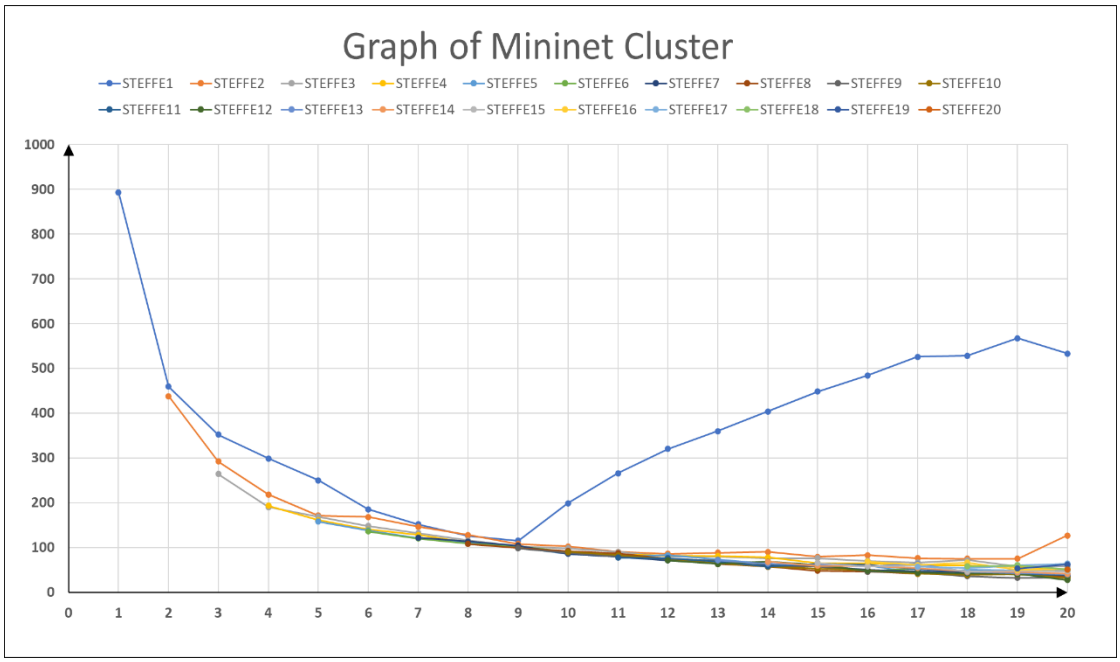
```
def run_command(host, command):
        output = host.cmd(command)
        with open('X_to1.test', 'a') as file:
                file.write(output)


for i in range(0,100):
        print(i)
        processes = []
        for host in hosts:
                command = 'iperf -c 10.0.0.1 -t 5'
                p = multiprocessing.Process(target=run_com-
mand, args=(net.getNodeByName(host), command))
                p.start()
                processes.append(p)
        for p in processes:
                p.join()
```

		HOST INVOLVED																			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
HOSTNAME	STEFFE1	893	459	352	299	250	185	152	126	115	199	266	320	360	404	448	484	526	528	567	533
	STEFFE2	0	438	292	218	171	168	146	128	108	102	90	86	88	90	79	83	76	75	75	127
	STEFFE3	0	0	264	190	169	148	132	116	103	98	90	79	80	76	76	70	66	72	58	57
	STEFFE4	0	0	0	193	161	140	128	111	99	91	84	82	81	78	65	65	60	60	56	51
	STEFFE5	0	0	0	0	158	138	121	111	100	86	80	83	73	64	58	60	43	50	49	48
	STEFFE6	0	0	0	0	0	136	120	109	102	90	82	73	64	61	58	48	52	47	39	36
	STEFFE7	0	0	0	0	0	0	122	113	104	86	79	71	65	60	59	50	46	45	40	35
	STEFFE8	0	0	0	0	0	0	0	108	99	92	87	74	63	57	48	46	42	41	41	36
	STEFFE9	0	0	0	0	0	0	0	0	98	88	80	74	71	57	58	46	44	36	32	33
	STEFFE10	0	0	0	0	0	0	0	0	0	90	83	75	65	57	53	49	41	39	40	32
	STEFFE11	0	0	0	0	0	0	0	0	0	0	77	76	68	58	63	61	51	46	41	38
	STEFFE12	0	0	0	0	0	0	0	0	0	0	0	71	63	68	61	48	44	42	41	27
	STEFFE13	0	0	0	0	0	0	0	0	0	0	0	0	71	64	61	58	63	45	43	40
	STEFFE14	0	0	0	0	0	0	0	0	0	0	0	0	0	68	59	59	54	46	45	41
	STEFFE15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65	58	60	47	48	47
	STEFFE16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	69	61	66	50	51
	STEFFE17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	57	54	60	64
	STEFFE18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	58	60	51
	STEFFE19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	53	62
	STEFFE20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	51

4 Data analysis

With the data contained in the tables it is possible to create a graph that has, as a unit of measurement of x-axis, the number of nodes involved in the measurement and, as a unit of measurement of y-axis, the amount of bandwidth expressed in Mbps.





We can see that unexpected jumps occur at 9 and 19 x-values, but we see that Mininet emulates exactly this behavior.

Other values of interest that we can extract from the tables are related to the average value (μ) calculated according to the arithmetic mean and the variance (σ), calculated taking into account the nodes from 1 to N and those from 2 to N. In the values of the average we can see how the emulated network remains less performing than the physical network. From the values of the variance instead we see, as well as from the graph, a considerable discrepancy once the threshold of 9 nodes is exceeded.

All the tests that were performed went to analyze the behavior in UL towards the master node.

Average for Testbench (Physical Cluster)																				
μ	938,00	473,00	323,33	245,00	197,80	166,33	140,14	119,75	107,89	108,90	107,82	108,25	104,77	103,93	99,87	100,19	97,29	96,17	93,11	90,90
σ(1-N)	0	8,485281	1,527525	3,366502	2,167948	2,503331	1,573592	1,035098	2,088327	34,49783	58,79935	81,21926	94,46	110,0353	117,3711	130,1626	135,0244	143,8182	147,3077	138,9623
σ(2-N)		0	0,707107	2,516611	2,5	2,774887	1,67332	0,534522	0,886405	1,5	2,13177	2,960344	3,058768	4,0128	3,10618	4,700557	4,441753	5,774564	5,562479	12,54978

Average for Mininet (Emulated Cluster)																				
μ	893,00	448,50	302,67	225,00	181,80	152,50	131,57	115,25	103,11	102,20	99,82	97,00	93,23	90,14	87,40	84,63	81,53	77,61	75,68	73,00
σ(1-N)	0	14,84924	44,95924	50,90514	38,50584	19,77625	12,75222	7,667184	5,441609	34,38604	55,28439	70,39241	80,5338	90,84548	100,0598	106,9952	114,9593	112,9467	119,4097	110,2934
σ(2-N)		0	19,79899	15,37314	6,238322	13,11488	9,887703	6,824326	3,335416	5,364492	4,54117	5,061441	8,15754	10,01665	8,072814	10,64358	10,16202	11,41754	10,44391	21,5911

4.1 Future works

Future work to carry out a more precise analysis may be to make measurements in the presence of traffic between client nodes. We can apply a traffic generator between the nodes engaged in communication, what we expect to see is a load of data relative to the part of DL, client side that not affect the performances, and a division of the UL channel, one part to communicate between clients and one to communicate with the master. An accurate analysis would show the threshold after which the traffic load between the hosts negatively impacts the reception of the data from master node. This would happen because the communication bottleneck would be the master-switch link, but if the ability of the nodes to send packets is reduced due to internal traffic then the problem will no longer be related to having only one master-switch link.

However, the split part of UL and DL is not applied to the switch band within Mininet, so proper configuration is required.

5 Topology builder

Along with the other files and programs of the project is present in my GitHub repository* also a program in Python with the aim of facilitating the creation of a topology without acting directly with Python but using a properly built configuration file. With the use of this my software called *MininetNetPractice.py* I modeled the cluster network to emulate, in the repository there is also another configuration file that shows how you can use the configuration file for more complex topologies. This program takes a file as input by specifying it via the -f flag or within the current directory uses the default file called *MininetTopo.conf*.

For more details the topic is better covered on project README.md .

* https://github.com/edoardoColi/Communication_Sandbox