Assignment 3

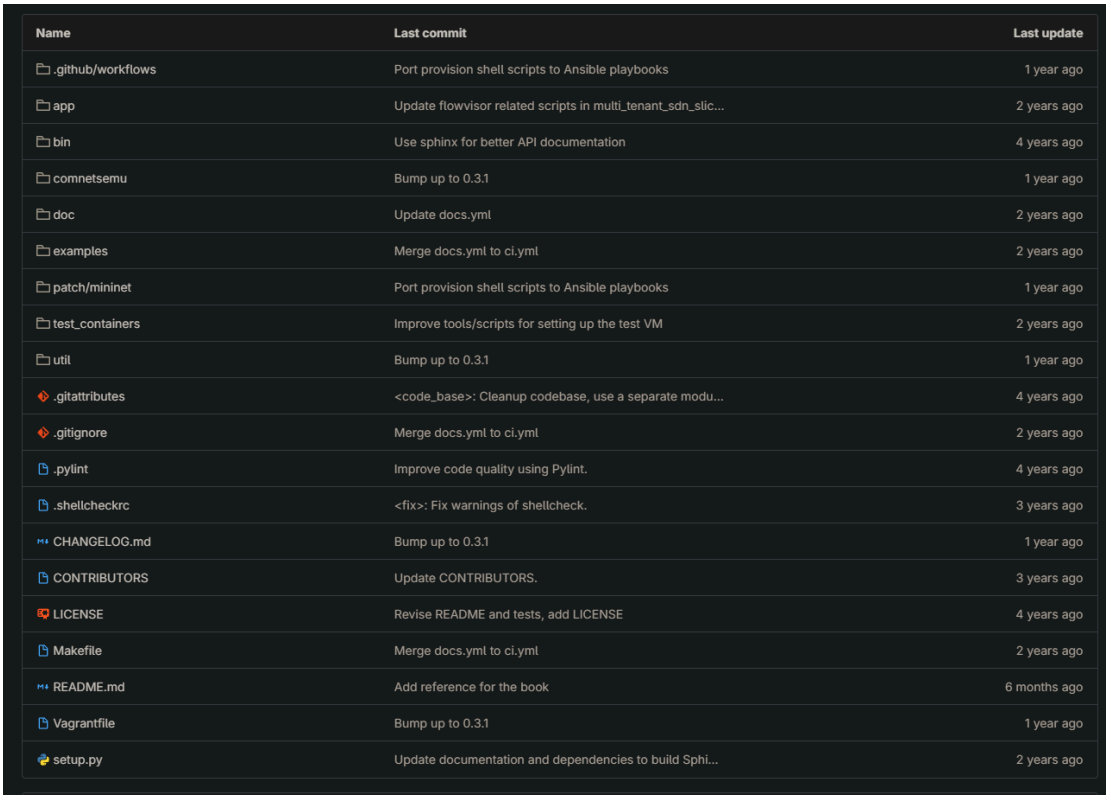
Experiments on Network slicing, SDN, ComNet simulator report

Jaishana Bindhu Priya (21bcs045)
Pawar Parth Sanjay (21bcs083)
R Eswar Naik (21bec035)
D.Rishikesh(21bec013)
R Hari prasad (21bec018)

ComNetsEmu Setup and Network Slicing:

We need to clone the repository from the GitLab public repository - https://git.comnets.net/public-repo/comnetsemu.git

Once we have cloned the repository we can see the structure as shown below



Name	Last commit	Last update
.github/workflows	Port provision shell scripts to Ansible playbooks	1 year ago
app	Update flowvisor related scripts in multi_tenant_sdn_slic...	2 years ago
bin	Use sphinx for better API documentation	4 years ago
comnetsemu	Bump up to 0.3.1	1 year ago
doc	Update docs.yml	2 years ago
examples	Merge docs.yml to ci.yml	2 years ago
patch/mininet	Port provision shell scripts to Ansible playbooks	1 year ago
test_containers	Improve tools/scripts for setting up the test VM	2 years ago
util	Bump up to 0.3.1	1 year ago
.gitattributes	<code_base>: Cleanup codebase, use a separate modu...	4 years ago
.gitignore	Merge docs.yml to ci.yml	2 years ago
.pylint	Improve code quality using Pylint.	4 years ago
.shellcheckrc	<fix>: Fix warnings of shellcheck.	3 years ago
CHANGELOG.md	Bump up to 0.3.1	1 year ago
CONTRIBUTORS	Update CONTRIBUTORS.	3 years ago
LICENSE	Revise README and tests, add LICENSE	4 years ago
Makefile	Merge docs.yml to ci.yml	2 years ago
README.md	Add reference for the book	6 months ago
Vagrantfile	Bump up to 0.3.1	1 year ago
setup.py	Update documentation and dependencies to build Sphi...	2 years ago

The examples that we will be looking into are present in the location - **comnetsemu/app/realizing_network_slicing**

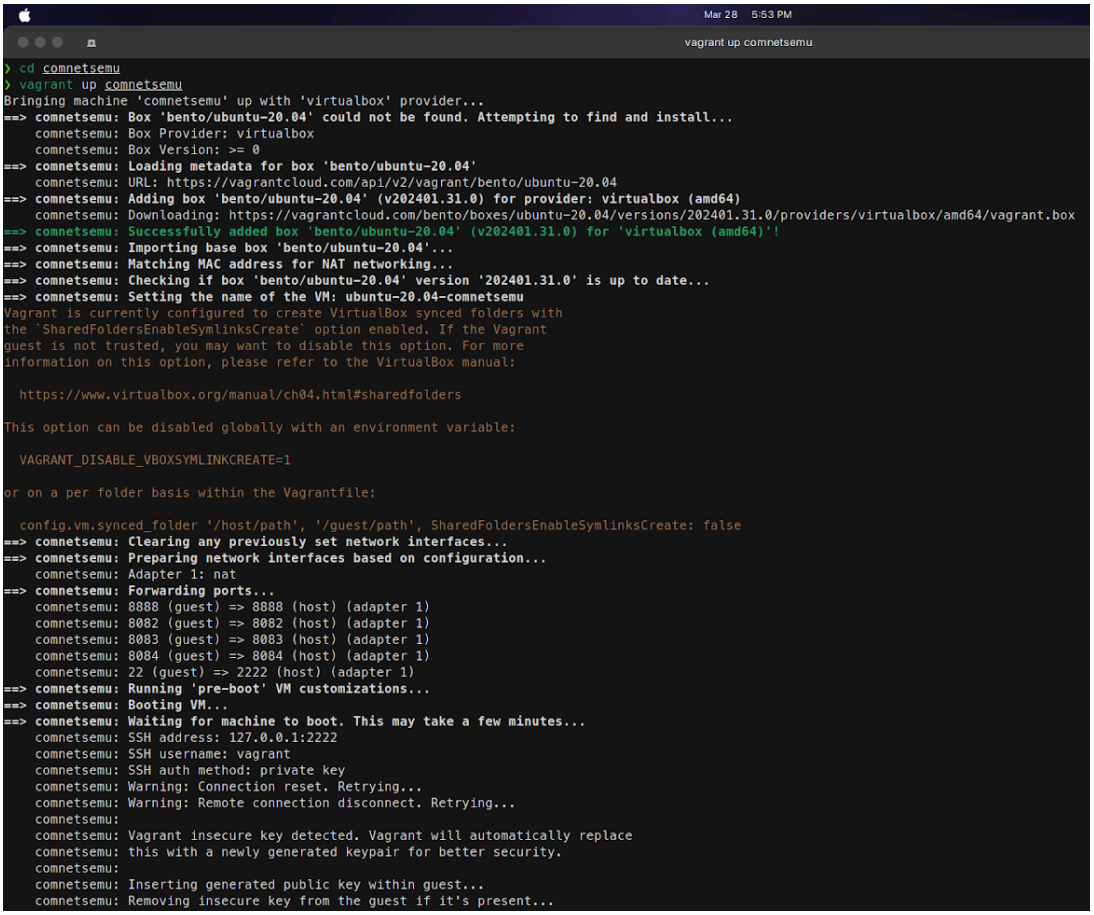
Now We will be looking into the working of different functions and APIs created to make this objective of Virtualizing the Network.

There are some basic requirements to run ComNetsEmu

- VirtualBox (or any other Virtual Machine)
- Vagrant
- Ubuntu OS

We need to run the following commands to start the virtualbox along with the ComNetsEmu :
`vagrant up comnetsemu`

- This will start up the virtualbox with username and password as comnetsemu.
- Once this runs correctly a confirmation message in green font appears stating that the virtualbox has been completed successfully. Next we run the following command.



vagrant ssh comnetsemu

- This loads up the virtual machine which opens a ssh (command line) interface so that we can run our commands in the VM.

- We can run cd comnetsemu and then try to run to check if our installation was successful by executing the “docker-in-docker” project.

```
❯ vagrant ssh comnetsemu
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-170-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

  System information as of Thu 28 Mar 2024 12:34:37 PM UTC

  System load:  0.36               Processes:               162
  Usage of /:   18.2% of 30.34GB   Users logged in:         0
  Memory usage: 14%                IPv4 address for docker0: 172.17
  Swap usage:   0%                 IPv4 address for eth0:    10.0.2


This system is built by the Bento project by Chef Software
More information can be found at https://github.com/chef/bento
/usr/bin/xauth:  file /home/vagrant/.Xauthority does not exist
vagrant@comnetsemu:~$
```

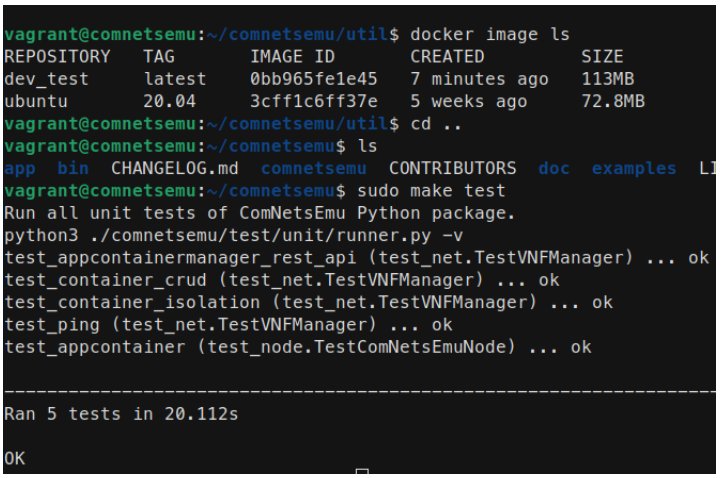
./install.sh -u

- This will install the ComNetsemu Emulator.
- Once the task is completed, it confirms to delete and close all the images that are running currently.

```
TASK [Build all test containers for ComNetsEmu (required for unit tests and some built-in examples)] ***********************************************
[WARNING]: Skipping plugin (/usr/lib/python3/dist-packages/ansible/plugins/filter/core.py) as it seems to be invalid: cannot import name 'environmentfilte
(/usr/local/lib/python3.8/dist-packages/jinja2/filters.py)
[WARNING]: Skipping plugin (/usr/lib/python3/dist-packages/ansible/plugins/filter/mathstuff.py) as it seems to be invalid: cannot import name 'environment
(/usr/local/lib/python3.8/dist-packages/jinja2/filters.py)
changed: [127.0.0.1]

TASK [Install packages required for the development of ComNetsEmu] ***************************************************************************
ok: [127.0.0.1]

TASK [Install packages required to build HTML documentation of ComNetsEmu] ******************************************************************
[WARNING]: Skipping plugin (/usr/lib/python3/dist-packages/ansible/plugins/filter/core.py) as it seems to be invalid: cannot import name 'environmentfilte
(/usr/local/lib/python3.8/dist-packages/jinja2/filters.py)
[WARNING]: Skipping plugin (/usr/lib/python3/dist-packages/ansible/plugins/filter/mathstuff.py) as it seems to be invalid: cannot import name 'environment
(/usr/local/lib/python3.8/dist-packages/jinja2/filters.py)
changed: [127.0.0.1]

TASK [Start and enable the Docker service] ***********************************************************************************************************
ok: [127.0.0.1]

PLAY RECAP *******************************************************************************************************************************************
127.0.0.1                  : ok=14   changed=7    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0

vagrant@comnetsemu:~/comnetsemu/util$
```

sudo make test

- This will try to run unit tests to check the installation of all the needed components.
- If the final print statement says “Ran 5 tests in 86.6s” and “OK”.
  This means that all the preliminary tests have been passed and we can carry on with the next step to try and execute the docker-in-docker project to test out and continue with our other objectives.



make doc

- This compiles all the important functions and provides an HTML page to show us the required functions and APIs.
- This contains all the necessary information regarding the components of ComNetsEmu.

```
OK
vagrant@comnetsemu:~/comnetsemu$ make doc
Build documentation in HTML format
rm ./doc/api/*.rst
rm: cannot remove './doc/api/*.rst': No such file or directory
make: [Makefile:84: doc] Error 1 (ignored)
rm -r ./doc/build/
rm: cannot remove './doc/build/': No such file or directory
make: [Makefile:85: doc] Error 1 (ignored)
# Use sphinx-apidoc to generate API sources
cd ./doc/ && sphinx-apidoc -o api ../comnetsemu -H 'Python API' --separate
Creating file api/comnetsemu.rst.
Creating file api/comnetsemu.clean.rst.
Creating file api/comnetsemu.cli.rst.
Creating file api/comnetsemu.exceptions.rst.
Creating file api/comnetsemu.net.rst.
Creating file api/comnetsemu.node.rst.
Creating file api/comnetsemu.overrides.rst.
Creating file api/comnetsemu.tool.rst.
Creating file api/comnetsemu.util.rst.
Creating file api/modules.rst.
# Build HTML files
cd ./doc/ && make html
make[1]: Entering directory '/home/vagrant/comnetsemu/doc'
Running Sphinx v7.1.2
WARNING: Invalid configuration value found: 'language = None'. Update your config
making output directory... done
myst v2.0.0: MdParserConfig(commonmark_only=False, gfm_only=False, enable_extensi
omains=None, fence_as_directive=set(), number_code_blocks=[], title_to_header=Fal
substitutions={}, linkify_fuzzy_links=True, dmath_allow_labels=True, dmath_allow_
_process|mathjax_process|math|output_area', enable_checkboxes=False, suppress_war
building [mo]: targets for 0 po files that are out of date
writing output...
building [html]: targets for 16 source files that are out of date
updating environment: [new config] 16 added, 0 changed, 0 removed
reading sources... [100%] vm_setup_issues
looking for now-outdated files... none found
pickling environment... done
checking consistency... done
preparing documents... done
copying assets... copying static files... done
copying extra files... done
done
writing output... [100%] vm_setup_issues
generating indices... genindex py-modindex done
writing additional pages... search done
dumping search index in English (code: en)... done
dumping object inventory... done
build succeeded, 1 warning.

The HTML pages are in build/html.
make[1]: Leaving directory '/home/vagrant/comnetsemu/doc'
```

ComNetsEmu

A virtual emulator/testbed designed for the book: Computing in Communication Networks: From Theory to Practice

Navigation

Quick search

Go

ComNetsEmu's Documentation!

A virtual emulator/testbed designed for the book: Computing in Communication Networks: From Theory to Practice

Check the README for project description.

This online page is mainly for ComNetsEmu API documentation.

class comnetsemu.net.**APPContainerManager**(*net: Mininet*)

Bases: **object**

Manager for application containers (sibling containers) deployed on Mininet hosts.

addContainer(*name: str, dhost: str, dimage: str, dcmd: str, docker_args: dict | None = None, wait: bool = True*) → APPContainer

Create and run a new container inside a Mininet DockerHost.

Parameters:
- **name** (*str*) – Name of the container.
- **dhost** – Name of the host used for deployment.
- **dimage** (*str*) – Name of the docker image.
- **dcmd** (*str*) – Command to run after the creation.
- **docker_args** (*dict*) – All other keyword arguments supported by Docker-py. e.g. CPU and memory related limitations. Some parameters are overriden for APPContainerManager's functionalities.
- **wait** (*bool*) – Wait until the container has the running state if True.

Check cls.docker_args_default.

Returns: Added APPContainer instance or None if the creation process failed.

Return type: APPContainer

docker_args_default = *{'detach': True, 'init': True, 'labels': {'comnetsemu': 'dockercontainer'}, 'security_opt': ['seccomp:unconfined'], 'tty': True}*

- The above image shows the component - **addContainer**, along with the parameters it accepts and helps us understand the working of the components.
- These components are built on the basis of MiniNet (another Virtualization software)

Now we will discuss the code and structure of dockerindocker.py in the examples folder in comnetsemu folder.

```
vagrant@comnetsemu:~/comnetsemu/examples$ ls -a
.   dockerhostcli.py                dockerhost.py      dpdk         flowvisor          network_measurement.py  README.md  service_migration  vpn
..  dockerhost_manage_appcontainer.py  dockerindocker.py  echo_server  mininet_demystify  ovx_test.py             run.sh     tun_ebpf.py
vagrant@comnetsemu:~/comnetsemu/examples$
```

- We will now look into the code and try to understand what the important functions in dockerindocker.py perform.

- Below is the structure of the network that we will be trying to replicate. (Chain Topology)

```
  GNU nano 4.8
#!/usr/bin/python3

"""
About: Basic example of running Docker container inside Docker container Host

Topo: Chain topology  h1     h2     h3         hn
                       |      |      |          |
                      s1 --- s2 --- s3 --- ... sn
Tests:
- Ping test between internal container between head (h1) and tail (hn).
- Resource limitation test for head and tail hosts: The CPU and memory
  limitation of Dockerhost are configured to (50/n)% and 10MB. The internal
  containers try to use 100% and 300MB RAM by execute stree-ng. The actual
  resource usages are monitored and reported.
"""
```

- Now in the main function we will will initialize some important variables like Number of hosts (host_num)
- Then we call the function testDockerInDocker function and pass the required parameters.

```
if __name__ == "__main__":
    setLogLevel("info")
    host_num = 3
    parser = argparse.ArgumentParser(
        description="Basic example for Docker-in-Docker setup."
    )
    parser.add_argument(
        "--host_num", default=3, type=int, help="Number of hosts in the chain topology"
    )
    args = parser.parse_args()
    print("*** The number of hosts in the chain: " + str(args.host_num))
    testDockerInDocker(args.host_num)
```

- Now we add the hosts to the network.
- We add the switches as well to the network and also provide the bandwidth and delay for each switch.
- Once this is completed we Start the Network.

```python
def testDockerInDocker(n):

    net = Containernet(controller=Controller, link=TCLink)
    mgr = VNFManager(net)

    info("*** Adding controller\n")
    net.addController("c0")

    info("*** Adding Docker hosts and switches in a chain topology\n")
    last_sw = None
    hosts = list()
    # Connect hosts
    for i in range(n):
        # Unlimited access to CPU(s) cycles in CPU_SETS
        host = net.addDockerHost(
            "h%s" % (i + 1),
            dimage="dev_test",
            ip="10.0.0.%s" % (i + 1),
            docker_args={"cpuset_cpus": "0"},
        )
        hosts.append(host)
        switch = net.addSwitch("s%s" % (i + 1))
        net.addLink(switch, host, bw=10, delay="10ms")
        if last_sw:
            # Connect switches
            net.addLink(switch, last_sw, bw=10, delay="10ms")
        last_sw = switch

    info("*** Starting network\n")
    net.start()

    info(
        "*** Run a simple ping test between two internal containers deployed on h1 and h%s\n"
        % n
    )
    head = mgr.addContainer("head", "h1", "dev_test", "/bin/bash", docker_args={})
    tail = mgr.addContainer(
        "tail", "h%s" % n, "dev_test", "ping -c 3 10.0.0.1", docker_args={}
    )
```

- We also provide various parameters to the addDockerHost function,
- **dimage** : this parameter takes in the docker image needed to assign the host.
- **Docker_args** : this parameter some arguments like cpuset_cpus to 0, which means in which cpu the code should be executed.

  - **cpuset_cpus** (*str*) – CPUs in which to allow execution (`0-3`, `0,1`).

- Now we try to ping the internal containers h1 and hn where n is the number of hosts, here it is 3.

```
info(
    "*** Run a simple ping test between two internal containers deployed on h1 and h%s\n"
    % n
)
head = mgr.addContainer("head", "h1", "dev_test", "/bin/bash", docker_args={})
tail = mgr.addContainer(
    "tail", "h%s" % n, "dev_test", "ping -c 3 10.0.0.1", docker_args={}
)

info("*** Tail start ping head, wait for 5s...")
time.sleep(5)
info("\nThe ping result of tail to head: \n")
print(tail.dins.logs().decode("utf-8"))
mgr.removeContainer(head.name)
mgr.removeContainer(tail.name)

time.sleep(3)
```

- We use the addConatiner to assign head and tail for the first and last host.
- As we can see in the image given below the function tries to a new docker within the already running docker container.
- This tries to make a copy named host1 (h1) and host3 (h3) which run on the docker image “dev_test”.
- They run on “/bin/bash” a shell and run “ping -c 3 10.0.0.1” respectively.
- This basically makes the head host ping the tail host.

comnetsemu.net module

About: ComNetsEmu Network Module.

class comnetsemu.net.**APPContainerManager**(*net: Mininet*)

Bases: **object**

Manager for application containers (sibling containers) deployed on Mininet hosts.

addContainer(*name: str*, *dhost: str*, *dimage: str*, *dcmd: str*, *docker_args: dict | None = None*, *wait: bool = True*) → APPContainer

Create and run a new container inside a Mininet DockerHost.

Parameters:
- **name** (*str*) – Name of the container.
- **dhost** – Name of the host used for deployment.
- **dimage** (*str*) – Name of the docker image.
- **dcmd** (*str*) – Command to run after the creation.
- **docker_args** (*dict*) – All other keyword arguments supported by Docker-py. e.g. CPU and memory related limitations. Some parameters are overriden for APPContainerManager's functionalities.
- **wait** (*bool*) – Wait until the container has the running state if True.

Check cls.docker_args_default.

Returns: Added APPContainer instance or None if the creation process failed.

Return type: APPContainer

docker_args_default = *{'detach': True, 'init': True, 'labels': {'comnetsemu': 'dockercontainer'}, 'security_opt': ['seccomp:unconfined'], 'tty': True}*

- Next we need to set the CPU limitations for each of the hosts
- (docker containers).

```
info(
    "Update CPU limitation of all Docker hosts (h1-h%s) to %.2f %% \n"
    % (n, 50.0 / n)
)
info("Update Memory limitation of all Docker hosts (h1-h%s) to 10MB\n" % n)
info(
    "Deploy the stress app (100 % CPU and 300M memory) inside all Docker hosts to test the CPU
)
containers = list()
# Mark: CPU percent = cpu_quota / cpu_period. The default CPU period is
# 100ms = 100000 us
for i, h in enumerate(hosts):
    h.dins.update(cpu_quota=int(50000 / n))
    h.dins.update(mem_limit=10 * (1024 ** 2))  # in bytes
    c = mgr.addContainer(
        "stress_app_%s" % (i + 1),
        h.name,
        "dev_test",
        "stress-ng -c 1 -m 1 --vm-bytes 300M",
        docker_args={},
    )
    containers.append(c)

info(
    "Start monitoring resource usage of internal containers"
    "with default sample count: 3 and sample period: 1 sec\n"
)
```

- As seen in the above image, we create a list named containers and then append c into it which basically uses addContainer() and updates the

maximum CPU usage based on CPU percent = cpu_quota / cpu_period (which are defined in the below image).

- **cpu_percent** (*int*) – Usable percentage of the available CPUs (Windows only).
- **cpu_period** (*int*) – The length of a CPU period in microseconds.
- **cpu_quota** (*int*) – Microseconds of CPU time that the container can get in a CPU period.

- Now we try to set the cpu_quota and memory limitation. We also assign these to the addContainer() and add it to the container list.
- We use a tool named stress-ng "stress-ng -c 1 -m 1 –vm-bytes 300M". This means that we assign 1 core, 100% of the CPU and 300MB of memory or RAM.
- Next we make use of a method called monResourceStats that basically monitors the Resources used by a container.

```
for c in containers:
    usages = mgr.monResourceStats(c.name)
    if usages:
        print(
            " The average CPU and Memory usage of container:{} is {:.2f}%, {:.2f}MB".format(
                c.name,
                (sum(u[0] for u in usages) / len(usages)),
                (sum(u[1] for u in usages) / len(usages)),
            )
        )
    else:
        print("[ERROR] Failed to get resource usages from manager")

for c in containers:
    mgr.removeContainer(c.name)

info("*** Stopping network\n")
net.stop()
mgr.stop()
```

- Here we iterate over all the containers and print out the usages of each container using the monResourceStats() function and print the CPU and Memory Usages.

- Once these are done we remove the containers and stop the network emulation.
- Now we run the example in the virtual machine in the example directory.

```
sudo python3 dockerindocker.py
```

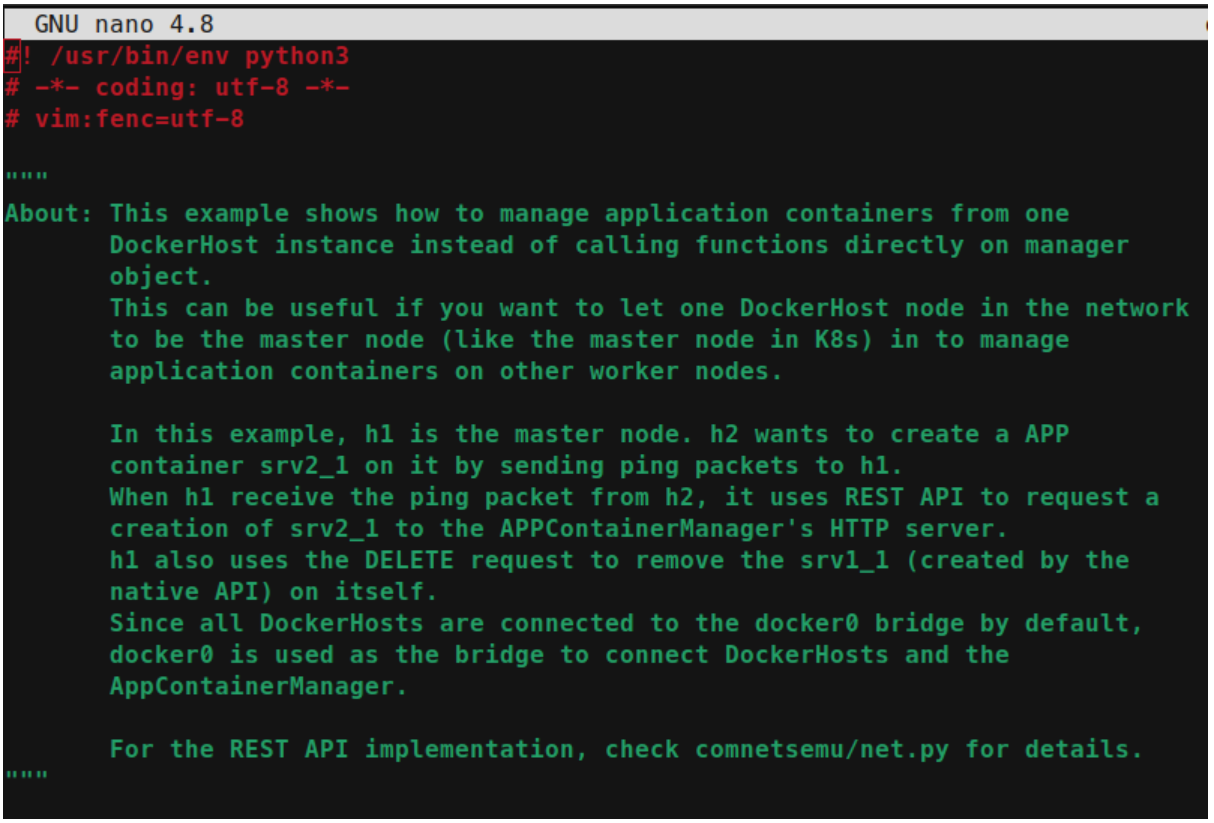
```
vagrant@comnetsemu:~/comnetsemu/examples$ sudo python3 dockerindocker.py
*** The number of hosts in the chain: 3
*** Adding controller
*** Adding Docker hosts and switches in a chain topology
(10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms d
Mbit 10ms delay) (10.00Mbit 10ms delay) *** Starting network
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) s2 (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit
elay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit
*** Run a simple ping test between two internal containers deployed on h1 and h3
*** Tail start ping head, wait for 5s...
The ping result of tail to head:
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=196 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=101 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=355 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 100.618/217.278/355.232/105.030 ms

*** Run CPU resource limitation test.Update CPU limitation of all Docker hosts (h1-h3) to 16.67 %
Update Memory limitation of all Docker hosts (h1-h3) to 10MB
Deploy the stress app (100 % CPU and 300M memory) inside all Docker hosts to test the CPU/Memory limitation
Start monitoring resource usage of internal containerswith default sample count: 3 and sample period: 1 sec
 The average CPU and Memory usage of container:stress_app_1 is 17.75%, 9.03MB
 The average CPU and Memory usage of container:stress_app_2 is 17.44%, 8.94MB
 The average CPU and Memory usage of container:stress_app_3 is 17.53%, 8.91MB
*** Stopping network
*** Stopping 1 controllers
c0
*** Stopping 5 links
.....
*** Stopping 3 switches
s1 s2 s3
*** Stopping 3 hosts
h1 h2 h3
*** Done
```

- The above is the results obtained after executing the file.
- Here we can make the following observations :
- First packet takes more time (nearly 1815ms) due to initialization of the app.

- We also limited the maximum CPU usages of each container to sum upto a maximum of 50%.
- In the monitoring function we can observe that we use a maximum of 16% of CPU and 10MB of RAM.
-
- We can see that all the containers are working based on the constraints that we provideR

In another example, dockerhost_manage_appcontainer.py

- Here we try to manage the application container from one Docker Host Instance rather than directly from the manager.
- Here docker0 bridges DockerHosts and AppContainerManager. We also make use of REST APIs to send requests to create and delete an APP container on the master node h1.



- Now we will go through the code to explain the working of function AppConatinerManager()

```python
if __name__ == "__main__":

    # Only used for auto-testing.
    AUTOTEST_MODE = os.environ.get("COMNETSEMU_AUTOTEST_MODE", 0)

    setLogLevel("info")

    net = Containernet(controller=Controller, link=TCLink, xterms=False)
    mgr = VNFManager(net)

    info("*** Add controller\n")
    net.addController("c0")

    info("*** Creating hosts\n")
    h1 = net.addDockerHost(
        "h1", dimage="dev_test", ip="10.0.0.1", docker_args={"hostname": "h1"}
    )
    h2 = net.addDockerHost(
        "h2", dimage="dev_test", ip="10.0.0.2", docker_args={"hostname": "h2"}
    )

    info("*** Adding switch and links\n")
    switch1 = net.addSwitch("s1")
    switch2 = net.addSwitch("s2")
    net.addLink(switch1, h1, bw=10, delay="10ms")
    net.addLink(switch1, switch2, bw=10, delay="10ms")
    net.addLink(switch2, h2, bw=10, delay="10ms")

    info("\n*** Starting network\n")
    net.start()

    ip_route = pyroute2.IPRoute()
    mgr_api_ip = ip_route.get_addr(label="docker0")[0].get_attr("IFA_ADDRESS")
    print("*** Deploy srv1_1 on h1 with native API.")
    mgr.addContainer("srv1_1", "h1", "dev_test", "bash", docker_args={})
    print_deployed_containers(mgr, ["h1", "h2"])
```

- As we can see above, It is similar to the previous example.
- The main change here is that :
- We deploy the master node directly.
- But we also need to manually run the HTTP server by providing the port and IP address.

```python
print("*** Run REST API server thread. It listens on docker0 interface")
mgr.runRESTServerThread(ip=mgr_api_ip, port=8000)
time.sleep(1)
```

- We can use the REST API to add and remove the APP from h1.

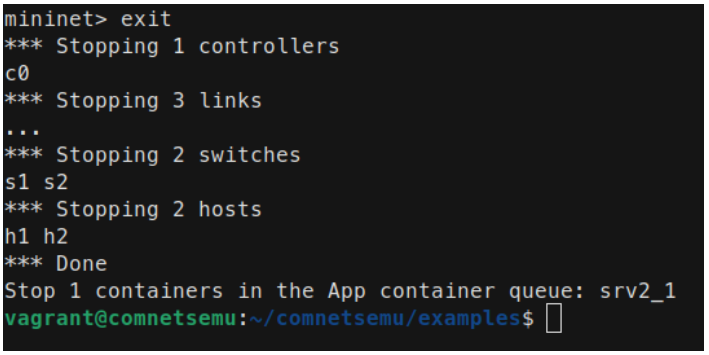
- We are using TCP dump to print out the result after the ping to the master node h1.

```
request_url = f"{mgr_api_ip}:8000/containers"
print("**** h1 waits for one Ping message from h1.")
h2.cmd("ping 10.0.0.1 &")
ret = h1.cmd("tcpdump -i h1-eth0 -c 1 icmp")
print(f"h1: output of tcpdump:\n {ret}")
print("**** h1 request to create a container named srv2_1 on h2 via REST API.")
h1.cmd(
    'curl -X POST --data \'{"name": "srv2_1", "dhost": "h2","dimage":"dev_test", "dcmd": "bash", "docker_args": {}}\' '
    + request_url
)
time.sleep(1)
print_deployed_containers(mgr, ["h1", "h2"])

print("**** h1 request to delete a container named srv1_1 on itself via REST API.")
request_url += "/srv1_1"
h1.cmd("curl -X DELETE " + request_url)
time.sleep(1)
print_deployed_containers(mgr, ["h1", "h2"])
```

We run the file by using the following command
`sudo python3 dockerhost_manage_appcontainer.py`

```
vagrant@comnetsemu:~/comnetsemu/examples$ sudo python3 dockerhost_manage_appcontainer.py
*** Add controller
*** Creating hosts
*** Adding switch and links
(10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) (10.00Mbit 10ms dela
*** Starting network
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s1 (10.00Mbit 10ms delay) (10.00Mbit 10ms delay) s2 (10.00Mbit 10ms delay) (10.00Mbit 10m
y)
*** Deploy srv1_1 on h1 with native API.
*** Current deployed APP containers:
DockerHost: h1, APP containers on it: srv1_1
DockerHost: h2, APP containers on it:
*** Run REST API server thread. It listens on docker0 interface
Start REST API server on address: 172.17.0.1:8000.
**** h1 waits for one Ping message from h1.
h1: output of tcpdump:
 tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:02:32.108128 IP 10.0.0.2 > 10.0.0.1: ICMP echo request, id 8, seq 1, length 64
1 packet captured
26 packets received by filter
0 packets dropped by kernel

**** h1 request to create a container named srv2_1 on h2 via REST API.
172.17.0.2 - - [28/Mar/2024 13:02:44] "POST /containers HTTP/1.1" 200 -
*** Current deployed APP containers:
DockerHost: h1, APP containers on it: srv1_1
DockerHost: h2, APP containers on it: srv2_1
**** h1 request to delete a container named srv1_1 on itself via REST API.
172.17.0.2 - - [28/Mar/2024 13:02:45] "DELETE /containers/srv1_1 HTTP/1.1" 200 -
*** Current deployed APP containers:
DockerHost: h1, APP containers on it:
DockerHost: h2, APP containers on it: srv2_1
*** Starting CLI:
```

- As we can see in the above image,
- Once the ping is received on the master node h1, it creates an APP named “srv2_1” and we also remove the APP named “srv1_1”.
- We can exit the docker once we type “exit” in the miniNet CLI.



Open vSwitch Exercise 1: Create a bridge in OpenvSwitch

The focus of the exercise is the creation and configuration of a network bridge using Open vSwitch (OVS).

Environment:

- Virtual Machine: The exercise is conducted within a "comnetsemu" virtual machine, likely running on VirtualBox or a similar platform. This VM is specifically tailored for network emulation and experimentation.

- ComNetSEmu Framework: ComNetSEmu provides the underlying tools and environment for conducting network simulations.

- Mininet: A popular network emulator built upon Linux namespaces and utilities, Mininet allows users to create realistic network topologies within a single system.

- OpenFlow Controller: This exercise involves an OpenFlow controller, likely running on the same machine (localhost - 127.0.0.1), which is responsible for managing and controlling the flow of network traffic through the OVS bridge.

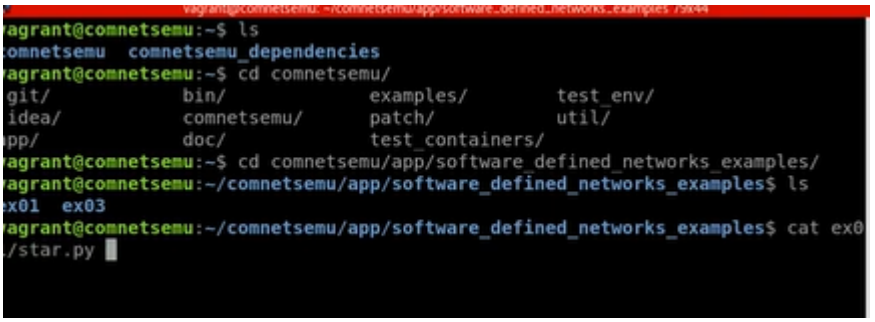
Exercise Steps:

Terminal Access & VM Login:

- Access a terminal on the host machine.
- Log into the ComNetSEmu virtual machine using the command: vagrant ssh comnetsemu.

Navigation to Exercise Directory:

- Within the VM, navigate to the exercise directory:cd~/comnetsemu/app/software_defined_networks_examples/



Python Script Execution:

- Execute the Python script to initiate the exercise scenario: sudo python3 ex01/star.py

```
"""

from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from nullswitch import NullSwitch

def emptyNet():

    "Create an empty network and add nodes to it."

    net = Mininet( controller=RemoteController, switch=NullSwitch )

    info( '*** Adding controller\n' )
    net.addController( 'c0' )

    info( '*** Adding hosts\n' )
    h1 = net.addHost( 'h1')
    h2 = net.addHost( 'h2')
    h3 = net.addHost( 'h3')

    info( '*** Adding switch\n' )
    s1 = net.addSwitch( 's1' )

    info( '*** Creating links\n' )
    net.addLink( h1, s1 )
    net.addLink( h2, s1 )
    net.addLink( h3, s1 )

    info( '*** Starting network\n')
    net.start()

    info( '*** Running CLI\n' )
    CLI( net )

    info( '*** Stopping network' )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    emptyNet()
```

- This script likely sets up the initial Mininet network topology.

```
vagrant@comnetsemu:~/comnetsemu/app/software_defined_networks_examples$ sudo py
thon3 ex01/star.py
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts
*** Adding switch
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1
*** Running CLI
*** Starting CLI:
mininet>
```

Mininet Shell Interaction:

- The user is now within the Mininet CLI.
- pingall: This command verifies network connectivity by sending ICMP (ping) requests between nodes in the emulated network.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet>
```

- s1 xterm &: Opens an xterm (terminal window) connected to node "s1," enabling direct interaction with the node.

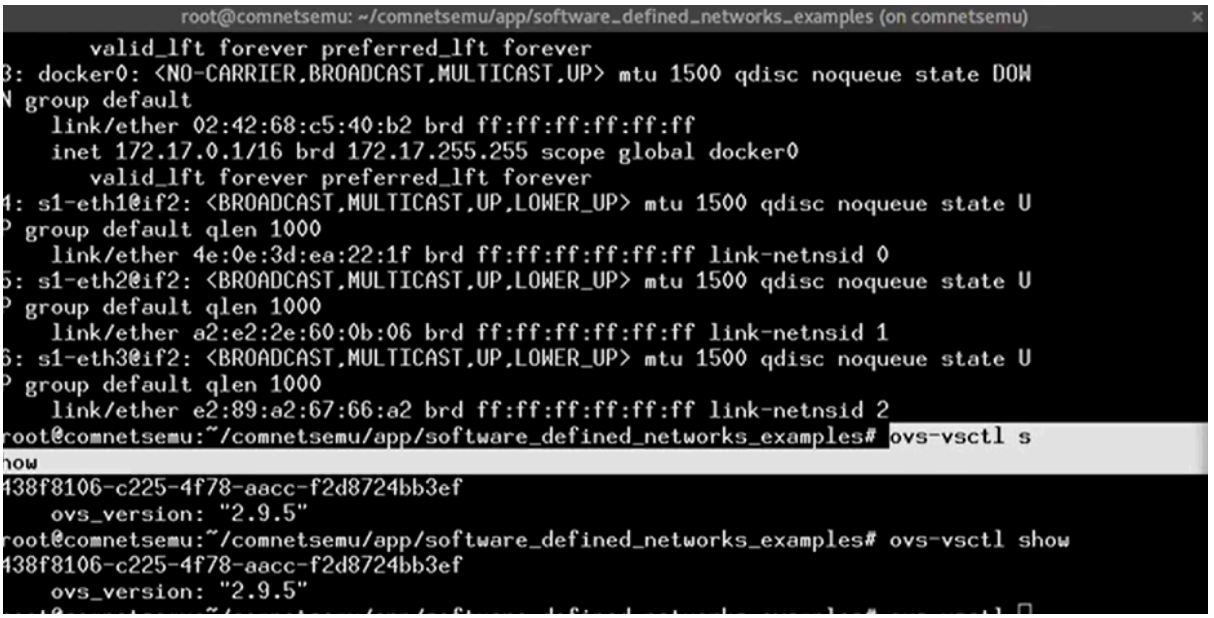
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet> s1 xterm &
```

Bridge Creation:

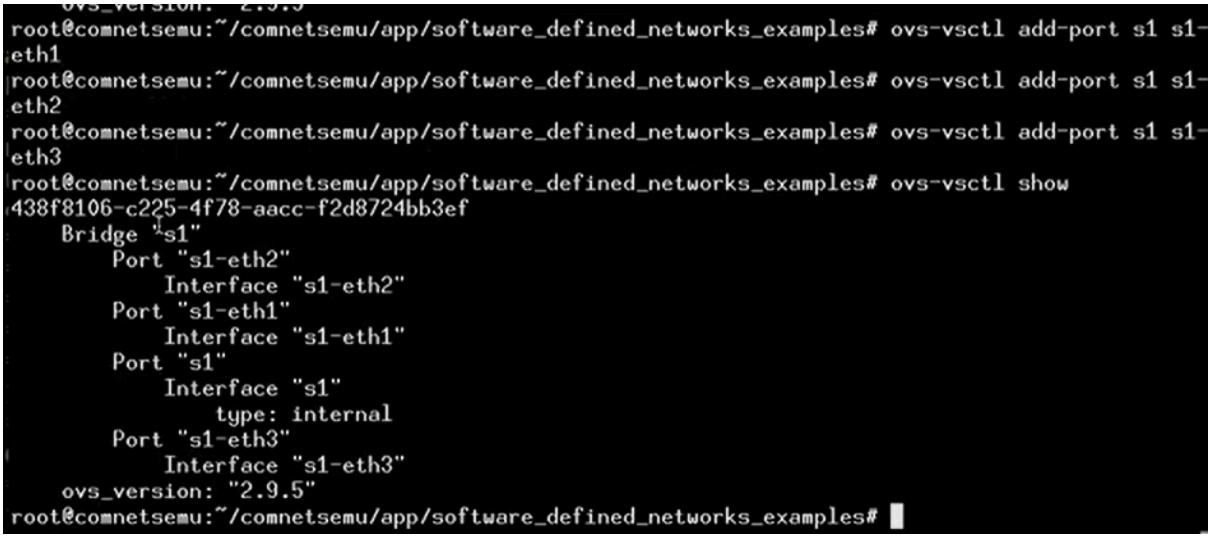
- The central task is creating an OVS bridge named "s1", acting as a virtual switch connecting multiple network interfaces.

- OVS commands used for bridge creation and interface addition:ovs-vsctl add-br s1 (Adds bridge "s1")

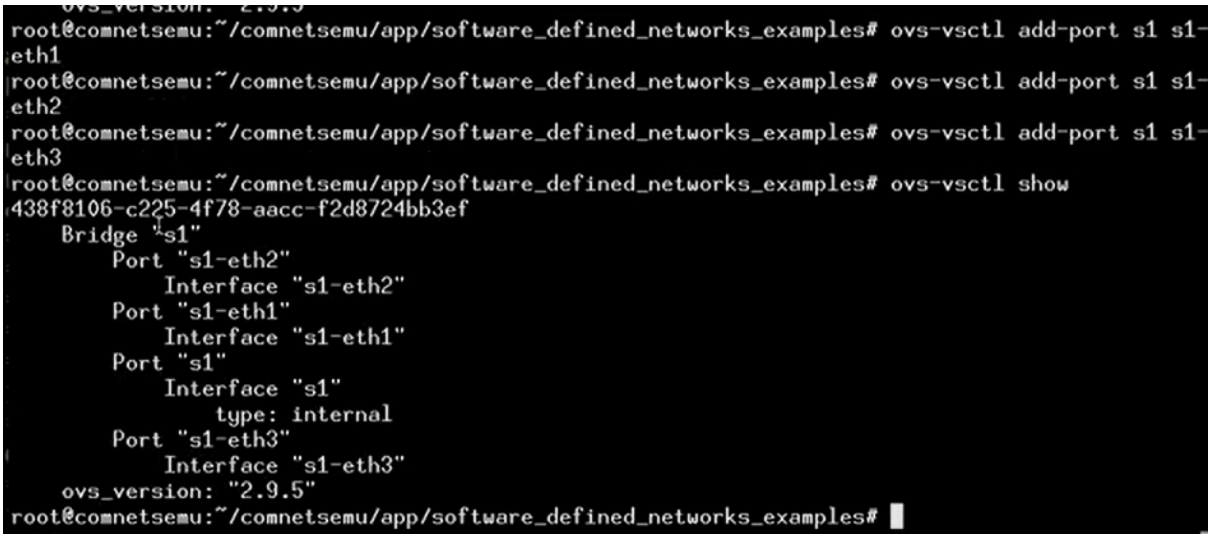
- ovs-vsctl add-port s1 s1-eth1 (Adds interface "s1-eth1" to the bridge)



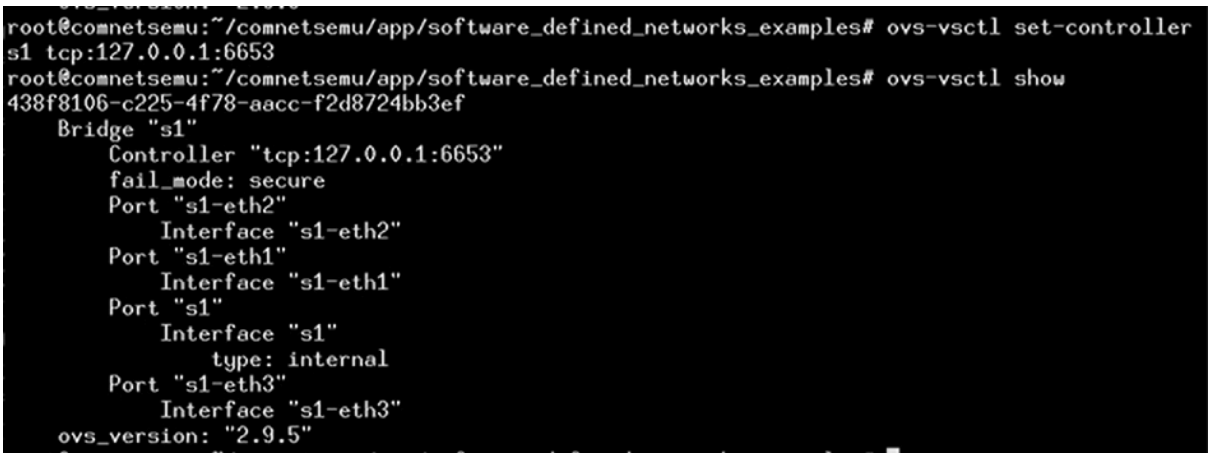
- ovs-vsctl add-port s1 s1-eth2 (Adds interface "s1-eth2" to the bridge)



- ovs-vsctl add-port s1 s1-eth3 (Adds interface "s1-eth3" to the bridge)



Verification:
After bridge creation and interface connections, pingall is used again within the Mininet shell to confirm successful communication between nodes via the new bridge.



Conclusion:
Users follow step-by-step instructions to create a network bridge, connect interfaces, and verify connectivity using ping. This practical approach to learning is fundamental for understanding the creation and management of virtual network infrastructures with OVS, a key technology in Software Defined Networking (SDN).

Open vSwitch Exercise 2: Connect to Controller

The second exercise focuses on connecting an Open vSwitch (OVS) bridge to a reference OpenFlow controller.

Environment:

- **Virtual Machine:** The exercise takes place within a virtual machine likely named "comnetsemu", designed for network emulation.
- **ComNetSEmu Framework:** Provides the foundational tools and environment for network simulation.
- **Mininet:** Used to create the emulated network topology within the VM.
- **Open vSwitch (OVS):** Functions as the virtual switch in this scenario.
- **ovs-testcontroller:** A simple reference OpenFlow controller provided by OVS for testing and demonstration purposes.

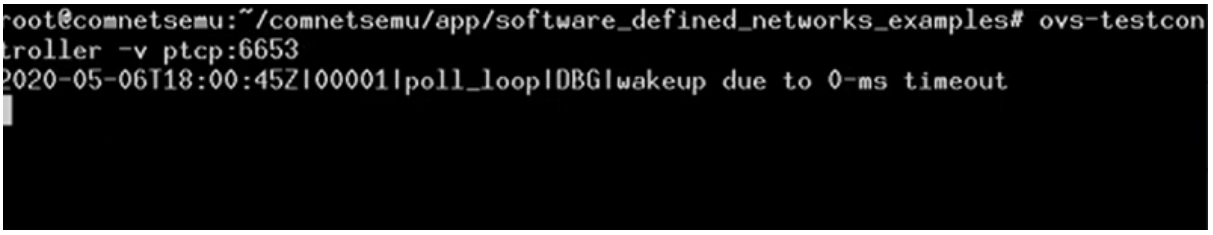
Exercise Steps:

1. **Secure Fail Mode Configuration:**

- Ensure the OVS bridge is set to the "secure" fail mode, which prevents the switch from independently adding flow entries, relying entirely on the controller for flow management.

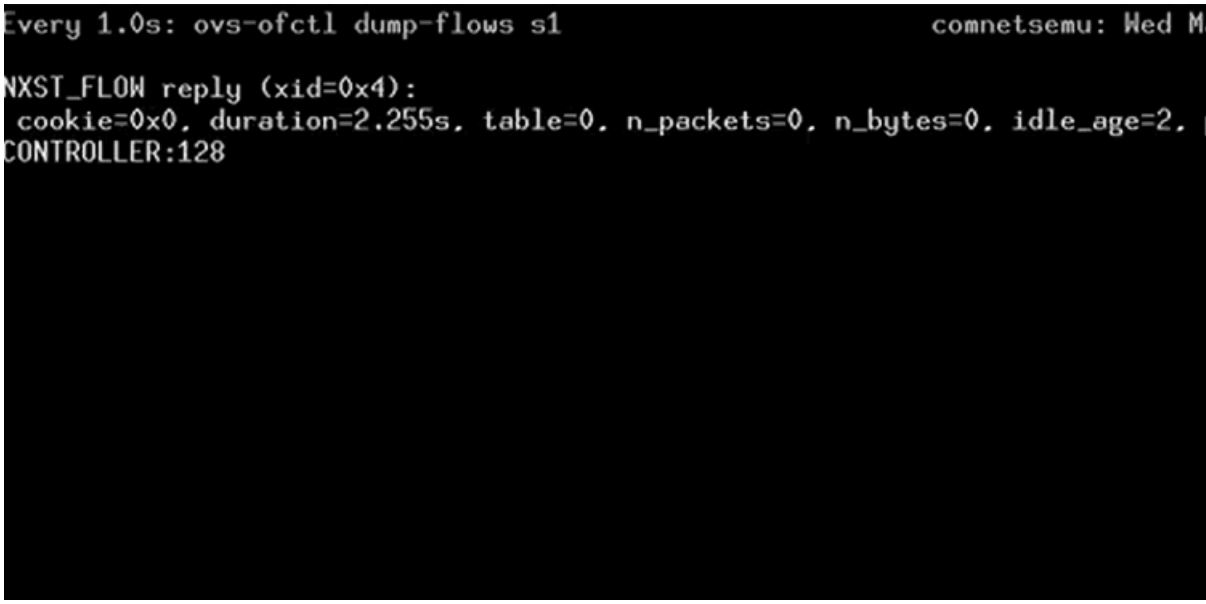
2. **Connect Switch to Controller:**

- Establish a connection between the OVS bridge and the reference OpenFlow controller.
- This is likely achieved using a command similar to the one used in the previous exercise, specifying the controller's IP address and port: `ovs-vsctl set-controller [bridge_name] tcp:127.0.0.1:6653`
- In this case, the controller is running locally on port 6653.



3. **Run the Reference Controller:**

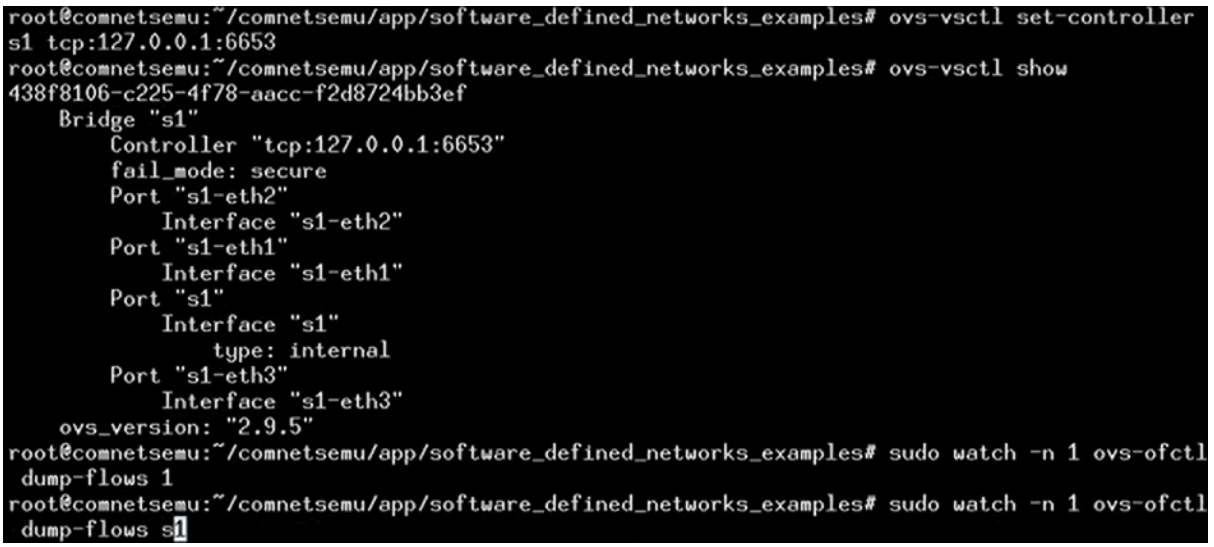
- In a second terminal window, start the "ovs-testcontroller" to act as the OpenFlow controller: `sudo ovs-testcontroller -v ptcp:6653`
- The -v flag increases verbosity, providing more detailed output.
    - ptcp:6653 specifies the protocol (passive TCP) and the port on which the controller listens.



4. **Dump Flow Entries:**

- In another terminal window, use the ovs-ofctl utility to display the flow entries currently installed in the OVS bridge: `sudo ovs-ofctl`

`dump-flows s1`



- This command retrieves and displays the flow table for the bridge named "s1".

Conclusion:

- This exercise demonstrates the practical steps involved in connecting an OVS bridge to an external OpenFlow controller. By setting the switch to "secure" fail mode, we enforce the controller's authority over traffic flow rules.
- Using the `ovs-testcontroller` allows for a simple yet effective way to observe and understand how an OpenFlow controller interacts with an OVS bridge. The `ovs-ofctl dump-flows` command provides visibility into the flow table, which reflects the rules installed by the controller to govern traffic within the emulated network.
- This exercise is fundamental for understanding the concepts and mechanisms of SDN, highlighting the role of the controller as the central point of control in managing network traffic.

Mininet:

Mininet provides a lightweight and scalable platform for creating virtual networks. It leverages Linux container (LXC) technology to create lightweight virtual network elements such as hosts, switches, and routers within a single Linux kernel. These virtual elements can be interconnected to form arbitrary network topologies, facilitating the emulation of various network scenarios.

Creating network Topology

Steps:

1. Install python3-pip

```
rishikesh@Ubuntu:~$ sudo apt install python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-pip is already the newest version (22.0.2+dfsg-1ubuntu0.4).
0 upgraded, 0 newly installed, 0 to remove and 216 not upgraded.
rishikesh@Ubuntu:~$
```

2. Install Ryu

```
rishikesh@Ubuntu:~$ sudo pip3 install ryu
[sudo] password for rishikesh:
Requirement already satisfied: ryu in /usr/local/lib/python3.10/dist-packages (4.34)
Requirement already satisfied: six>=1.4.0 in /usr/lib/python3/dist-packages (from ryu) (1.16.0)
Requirement already satisfied: routes in /usr/local/lib/python3.10/dist-packages (from ryu) (2.5.1)
Requirement already satisfied: oslo.config>=2.5.0 in /usr/local/lib/python3.10/dist-packages (from ryu) (9.4.0)
Requirement already satisfied: webob>=1.2 in /usr/local/lib/python3.10/dist-packages (from ryu) (1.8.7)
Requirement already satisfied: netaddr in /usr/local/lib/python3.10/dist-packages (from ryu) (1.2.1)
Requirement already satisfied: tinyrpc in /usr/local/lib/python3.10/dist-packages (from ryu) (1.1.7)
Requirement already satisfied: ovs>=2.6.0 in /usr/lib/python3/dist-packages (from ryu) (2.17.9)
Requirement already satisfied: msgpack>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from ryu) (1.0.8)
Requirement already satisfied: eventlet!=0.18.3,!=0.20.1,!=0.21.0,!=0.23.0,>=0.18.2 in /usr/local/lib/python3.10/dist-packages (from
 ryu) (0.36.1)
Requirement already satisfied: greenlet>=1.0 in /usr/local/lib/python3.10/dist-packages (from eventlet!=0.18.3,!=0.20.1,!=0.21.0,!=0
.23.0,>=0.18.2->ryu) (3.0.3)
Requirement already satisfied: dnspython>=1.15.0 in /usr/local/lib/python3.10/dist-packages (from eventlet!=0.18.3,!=0.20.1,!=0.21.0
,!=0.23.0,>=0.18.2->ryu) (2.6.1)
Requirement already satisfied: requests>=2.18.0 in /usr/lib/python3/dist-packages (from oslo.config>=2.5.0->ryu) (2.25.1)
Requirement already satisfied: stevedore>=1.20.0 in /usr/local/lib/python3.10/dist-packages (from oslo.config>=2.5.0->ryu) (5.2.0)
Requirement already satisfied: PyYAML>=5.1 in /usr/lib/python3/dist-packages (from oslo.config>=2.5.0->ryu) (5.4.1)
Requirement already satisfied: oslo.i18n>=3.15.3 in /usr/local/lib/python3.10/dist-packages (from oslo.config>=2.5.0->ryu) (6.3.0)
Requirement already satisfied: debtcollector>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from oslo.config>=2.5.0->ryu) (3.0.0
)
Requirement already satisfied: rfc3986>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from oslo.config>=2.5.0->ryu) (2.0.0)
Requirement already satisfied: repoze.lru>=0.3 in /usr/local/lib/python3.10/dist-packages (from routes->ryu) (0.7)
Requirement already satisfied: wrapt>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from debtcollector>=1.2.0->oslo.config>=2.5.
0->ryu) (1.16.0)
Requirement already satisfied: pbr!=2.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from oslo.i18n>=3.15.3->oslo.config>=2
.5.0->ryu) (6.0.0)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

3. Install git

```
rishikesh@Ubuntu:~$ sudo apt install git
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.34.1-1ubuntu1.10).
0 upgraded, 0 newly installed, 0 to remove and 216 not upgraded.
```

4. clone git

```
rishikesh@Ubuntu:~$ git clone git://github.com/mininet/mininet
fatal: destination path 'mininet' already exists and is not an empty directory.
rishikesh@Ubuntu:~$ git clone https://github.com/mininet/mininet
fatal: destination path 'mininet' already exists and is not an empty directory.
rishikesh@Ubuntu:~$ ls
Desktop    Downloads  inv.tbh  mealy.v  Music  oflops  openflow  Pictures  Public  Templates  vlsi   vlsi1.zip  xor.cir
Documents  home       inv.v    mininet  myenv  oftest  path      pox       snap    Videos     vlsi1  vlsi.zip
rishikesh@Ubuntu:~$
```

5. Command promote environment

```
rishikesh@Ubuntu:~$ cd mininet/
rishikesh@Ubuntu:~/mininet$ cd util/
rishikesh@Ubuntu:~/mininet/util$ ls
build-ovs-packages.sh  colorfilters  install.sh  m            openflow-patches  sysctl_addon  versioncheck.py
clustersetup.sh        doxify.py     kbuild      nox-patches  sch_htb-ofbuf     unpep8        vm
rishikesh@Ubuntu:~/mininet/util$ sudo./install.sh -a
bash: sudo./install.sh: No such file or directory
```

6. execute script named "install.sh" with administrative privileges

```
rishikesh@Ubuntu:~/mininet/util$ sudo ./install.sh -a
Detected Linux distribution: Ubuntu 22.04 jammy amd64
sys.version_info(major=3, minor=10, micro=12, releaselevel='final', serial=0)
Detected Python (python3) version 3
Installing all packages except for -eix (doxypy, ivs, nox-classic)...
Install Mininet-compatible kernel if necessary
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Hit:2 http://in.archive.ubuntu.com/ubuntu jammy InRelease
Get:3 http://in.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Hit:4 http://in.archive.ubuntu.com/ubuntu jammy-backports InRelease
Fetched 229 kB in 3s (65.7 kB/s)
Reading package lists... Done
Reading package lists...
Building dependency tree...
Reading state information...
linux-image-6.2.0-39-generic is already the newest version (6.2.0-39.40~22.04.1).
0 upgraded, 0 newly installed, 0 to remove and 216 not upgraded.
Installing Mininet dependencies
Reading package lists...
Building dependency tree...
Reading state information...
gcc is already the newest version (4:11.2.0-1ubuntu1).
make is already the newest version (4.3-4.1build1).
net-tools is already the newest version (1.60+git20181103.0eebece-1ubuntu5).
psmisc is already the newest version (23.4-2build3).
socat is already the newest version (1.7.4.1-3ubuntu4).
telnet is already the newest version (0.17-44build1).
help2man is already the newest version (1.49.1).
iperf is already the newest version (2.1.5+dfsg1-1).
pep8 is already the newest version (1.7.1-9ubuntu1).
pyflakes3 is already the newest version (2.4.0-2).
```

```
/home/rishikesh/mininet/util
Installing OpenFlow reference implementation...
Reading package lists...
Building dependency tree...
Reading state information...
autoconf is already the newest version (2.71-2).
automake is already the newest version (1:1.16.5-1.3).
gcc is already the newest version (4:11.2.0-1ubuntu1).
libtool is already the newest version (2.4.6-15build2).
make is already the newest version (4.3-4.1build1).
0 upgraded, 0 newly installed, 0 to remove and 216 not upgraded.
Reading package lists...
Building dependency tree...
Reading state information...
autotools-dev is already the newest version (20220109.1).
pkg-config is already the newest version (0.29.2-1ubuntu3).
git is already the newest version (1:2.34.1-1ubuntu1.10).
libc6-dev is already the newest version (2.35-0ubuntu3.7).
0 upgraded, 0 newly installed, 0 to remove and 216 not upgraded.
fatal: destination path 'openflow' already exists and is not an empty directory.
rishikesh@Ubuntu:~/mininet/util$
```

7. Running Mininet with elevated privileges for creating and testing network configurations requiring administrative access.

```
rishikesh@Ubuntu:~/mininet/util$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 29.512 seconds
```

8. Creating a network with one switch and three hosts, configuring MAC addresses, using the Open vSwitch, and setting a remote controller using sudo privileges.

```
rishikesh@Ubuntu:~/mininet/util$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 1 switches
s1
*** Stopping 3 hosts
h1 h2 h3
*** Done
completed in 43.889 seconds
rishikesh@Ubuntu:~/mininet/util$
```

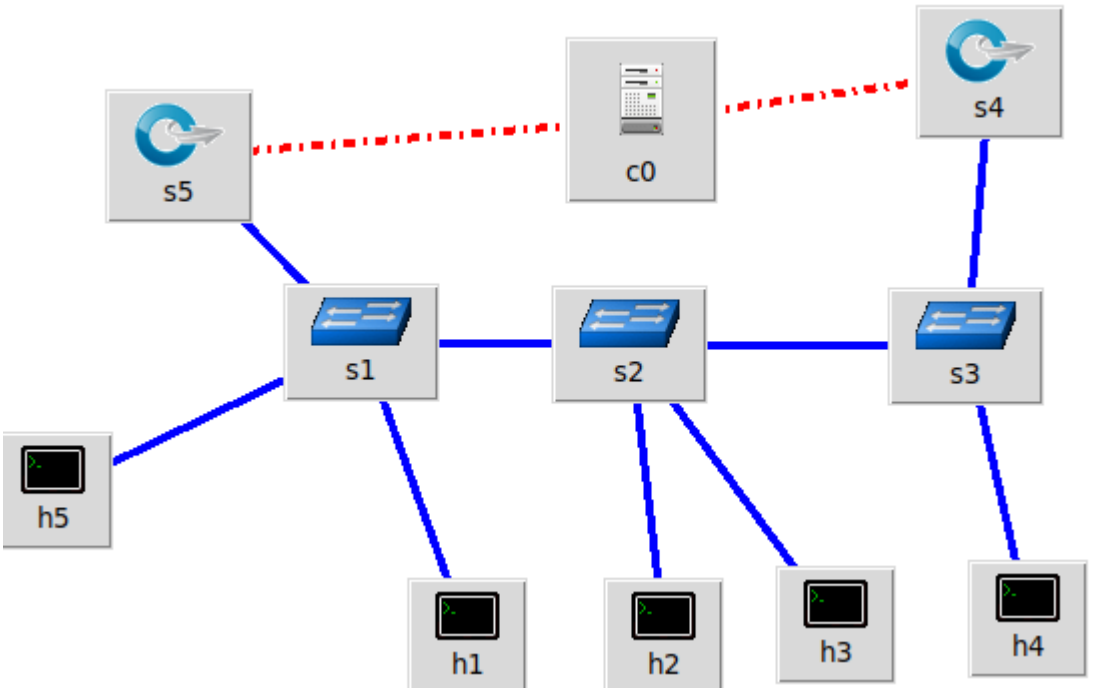
9.Launching the Mininet GUI editor from the specified directory with sudo privileges.

```
rishikesh@Ubuntu:~$ sudo ~/mininet/examples/miniedit.py
[sudo] password for rishikesh:
/usr/bin/env: ‘python’: No such file or directory
rishikesh@Ubuntu:~$ sudo apt install python3
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3 is already the newest version (3.10.6-1~22.04).
0 upgraded, 0 newly installed, 0 to remove and 216 not upgraded.
rishikesh@Ubuntu:~$
```

```
rishikesh@Ubuntu:~$ cd mininet/
rishikesh@Ubuntu:~/mininet$ ls
bin           custom  examples  Makefile          mn.1      mnexec.c   util
build         debian  INSTALL   mininet           mnexec    README.md
CONTRIBUTORS  doc     LICENSE   mininet.egg-info  mnexec.1  setup.py
rishikesh@Ubuntu:~/mininet$ cd examples/
```

10. Executing the Mininet GUI editor script using Python 3 with administrative privileges.

```
rishikesh@Ubuntu:~/mininet/examples$  sudo python3 ./miniedit.py
topo=none
```



3. RyU SN Controller: Example: Sending a message from switch

Ryu Applications :

- my_app.py: from ryu.base import app_manager from ryu.ofproto import ofproto_v1_3
 class MyApp(app_manager.RyuApp): OFP_VERSIONS = [ofproto_v1_3.0FP_VERSION]

- Run application with: ryu-manager my_app.py ss

Asychronous Messages :
◦ Messages from switches triggered by events
 • Call function inside the application class.
 • Example: @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER) def packet_in_handler(self, ev):
 msg= ev.msg dp = msg.datapath ofp = dp.ofproto ofp_parser = dp.ofproto_parser

Controller to Switch Messages:

• Messages sent from the controller to instruct the switch
 ◦ Eg. forward packets, add flows, query statistics, ..
 ◦ Example: flood a packet
 actions = [ofp_parser.0FPActionOutput(ofp.OFPP_FLOOD)] out = opf_parser.OFPPacketOut(datapath=dp, buffer_id=msg.buffer_id, in_port=msg.in_port, actions=actions) dp.send_msg(out)

Exercise : 3 — Ryu Examples

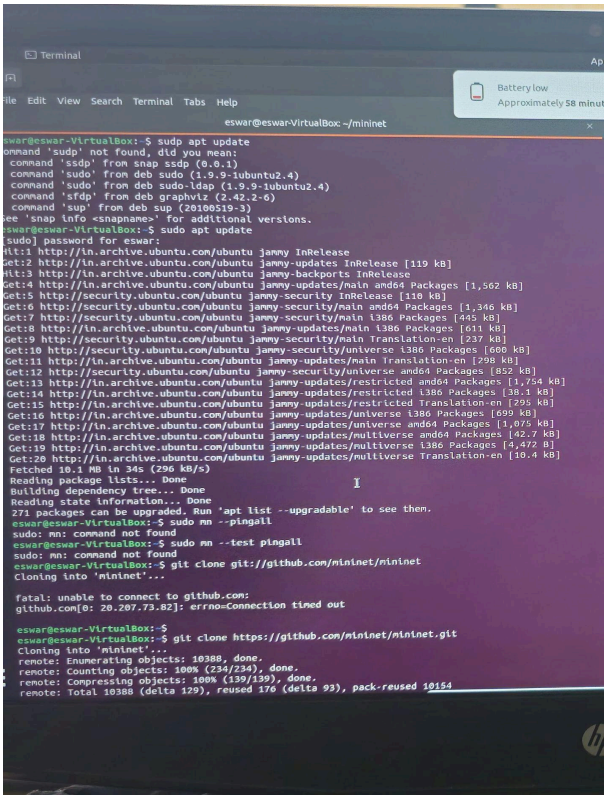
Run the Ryu application hub.py (stored in ~/sdn/ex03/hub.py)
ryu-manager --verbose hub.py
 • Open second terminal
 • Run mininet and connect to the controller

• sudo mn --topo single,3 \ --controller remote
• Send some packets with mininet, e.g. with „ingall or ‚iperf"
• Observe the output in Ryu
• Repeat with switch_13.py and sdn_switch_13.py
ryu-manager --verbose hub.py

Install Ryu and mininet on Ubuntu :

```
sudo apt update
sudo apt install ryu
ryu-manager --version
sudo apt update
git clone git://github.com/mininet/mininet
cd mininet
sudo ./util/install.sh -a
sudo mn --test pingall
```

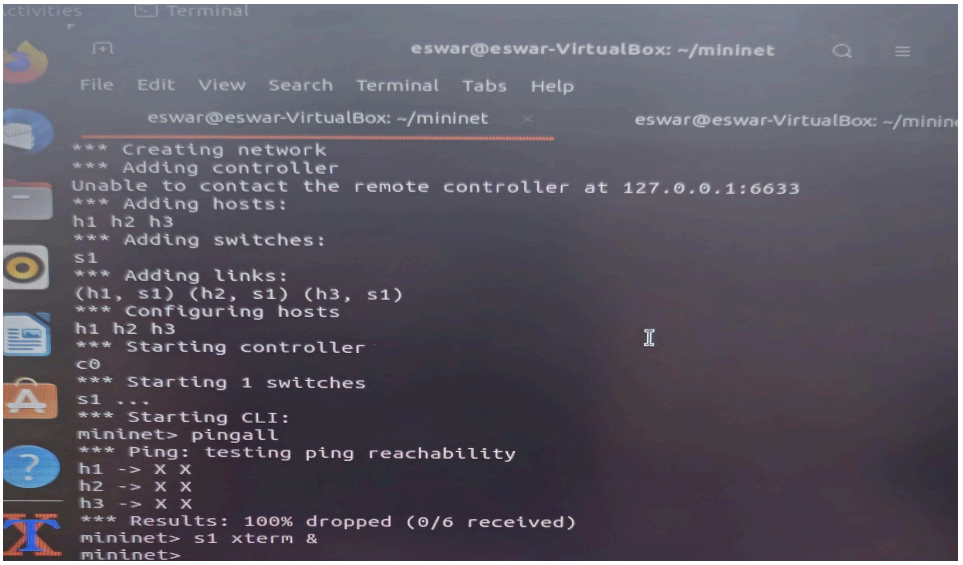
After using all commands given up you can connect to the github and install the require commands like ryu and mininet



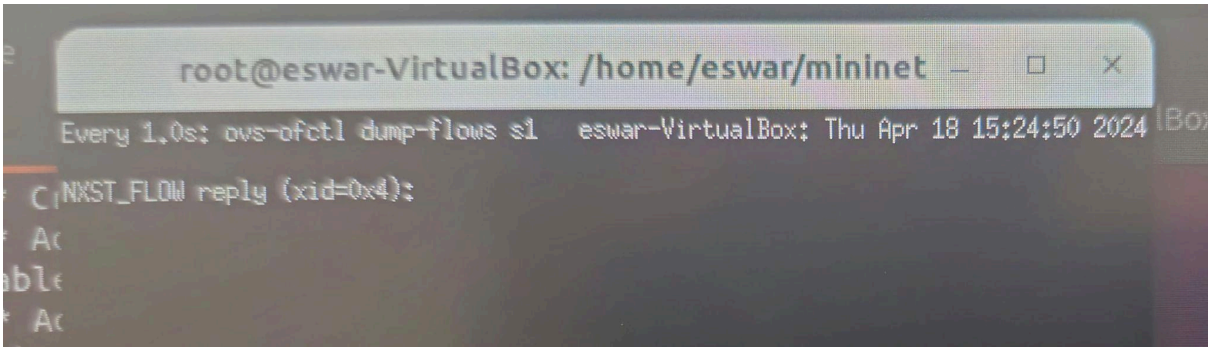
Next use the command

- Sudo mn-- topo single, 3 --controller remote,ip

for creating a network and adding controllers and then to test pingall



On testing the pingall with some values like (h1 h2) it will open another tab of root where we can see the NXST_FLOW reply (xid=0*4)



You can check your input messages values here
Like you want mininet>h1 ping h2

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=6.52 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=6.17 ms
```

```
root@comnetsemu:~/comnetsemu/app/software_defined_networks_examples# sudo watch
-n 1 ovs-ofctl dump-flows s1
root@comnetsemu:~/comnetsemu/app/software_defined_networks_examples# tcpdump -i
s1-eth3
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s1-eth3, link-type EN10MB (Ethernet), capture size 262144 bytes
18:42:09.649753 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4790, seq 1, lengt
h 64
18:42:09.654396 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 4790, seq 1, length
64
18:42:10.650515 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4790, seq 2, lengt
h 64
18:42:10.655416 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 4790, seq 2, length
64
```

You can see the packages loading
Now in the second terminal we are going to run a command nano hub.py

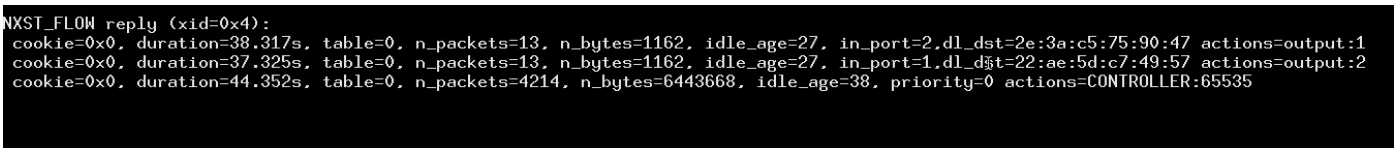
```
# here we gather the actions the switch should execute with the packet
# OFPActionOutput instructs the switch to send the message to an output port
# OFPP_FLOOD is a special output port for sending on all output ports
actions = [ofp_parser.OFPActionOutput(ofproto.OFPP_FLOOD)]

# if the packet is not in the switch buffer, include it in the PacketOut message
data = None
if msg.buffer_id == ofproto.OFP_NO_BUFFER:
    data = msg.data

# we put the actions into a packet_out message
# the message also contains
out = ofp_parser.OFPPacketOut(
    datapath=dp,  # assign the message to the switch
    buffer_id=msg.buffer_id,  # reflect the id assigned by the switch
    in_port=in_port,  # input port
    actions=actions,  # our list of actions
    data=data)  # send the packet back
```

Here we can see all the packages used and upcoming packages

Now by using ryu-manager sdn_switch_13.py --verbose command So, when you run `ryu-manager sdn_switch_13.py --verbose`, you're essentially starting the Ryu controller with the specified application (`sdn_switch_13.py`) and opting to receive verbose output, which can aid in troubleshooting and monitoring the SDN environment.

After do the ping test for mininet>h1 ping h2

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=14.1 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=5.59 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.414 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.132 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.144 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.069 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.120 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.143 ms
```



As you can see the difference here it is more simple and fast

NXST FLOW reply: This might indicate the software or tool used to generate the information, possibly related to network flow analysis.
xid: This stands for transaction ID, a unique identifier for this specific network flow.
cookie: This value is likely used for internal bookkeeping by the network analysis tool.
duration: This represents the amount of time (in seconds) that this network flow was active.
table: This could refer to a routing table or table used for managing network flows.
n_packets: This indicates the total number of data packets included in the network flow.
n_bytes: This represents the total size (in bytes) of all the data packets in the network flow.
idle_age: This might indicate how long it's been since the last packet was received in this flow.
in_port: This refers to the network port that the data packets in this flow originated from.
dl_dst: This stands for "destination link layer address," which is the MAC address of the device that received the data packets.
actions: This section likely specifies what actions were taken when processing this network flow (e.g., routing it to a specific output port).

Next use the command
mininet>pingall
Then use mininet>iperf for testing bandwidth between h1 and h3
As You can see the result for that its 16.1 Gbits/sec

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['16.1 Gbits/sec', '16.1 Gbits/sec']
mininet>
```

```
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=150.686s, table=0, n_packets=17, n_bytes=1442, idle_age=41, in_port=2,dl_dst=2e:3a:c5:75:90:47 actions=output:1
 cookie=0x0, duration=149.694s, table=0, n_packets=17, n_bytes=1442, idle_age=41, in_port=1,dl_dst=22:ae:5d:c7:49:57 actions=output:2
 cookie=0x0, duration=46.897s, table=0, n_packets=82851, n_bytes=5468886, idle_age=2, in_port=3,dl_dst=2e:3a:c5:75:90:47 actions=output:1
 cookie=0x0, duration=46.872s, table=0, n_packets=3, n_bytes=182, idle_age=41, in_port=2,dl_dst=da:f2:f7:06:71:b0 actions=output:3
 cookie=0x0, duration=46.869s, table=0, n_packets=3, n_bytes=182, idle_age=41, in_port=3,dl_dst=22:ae:5d:c7:49:57 actions=output:2
 cookie=0x0, duration=46.864s, table=0, n_packets=195105, n_bytes=10088577802, idle_age=2, in_port=1,dl_dst=da:f2:f7:06:71:b0 actions=output:3
 cookie=0x0, duration=156.721s, table=0, n_packets=4218, n_bytes=6444060, idle_age=46, priority=0 actions=CONTROLLER:65535
```

As we can see that the duration is increasing and package

```
    if dst in self.mac_to_port[dpid]:
        # if we already know the mac we can select the output port
        out_port = self.mac_to_port[dpid][dst]
        self.logger.info('Message for known {} at port {}'.format(src, out_port))

        # if we know the destination we can establish a flow
        self.add_flow(datapath, dst, in_port, out_port)
    else:
        # but if it can't be found we just flood the packet
        self.logger.info('Message for unknown {}'.format(src))
        out_port = ofproto.OFPP_FLOOD
```

function used for processing data packets

```
def add_flow(self, datapath, dst, in_port, out_port):
    ofproto = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    # to add a flow we have to define some match criteria
    match = ofp_parser.OFPMatch(
        in_port=in_port, # we match the input port
        eth_dst=dst) # and the destination MAC address

    # we also need to define the desired action
    actions = [ofp_parser.OFPActionOutput(out_port)]
    inst = [ofp_parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
    # now we build a flow table entry
    flow = ofp_parser.OFPFlowMod(
        datapath=datapath, # assign datapath
```

Function Definition: This code defines a function named `add_flow` that takes four arguments:

- `datapath`: This represents the switch or network device where the flow entry will be added.
- `dst`: This is the destination MAC address that the flow entry should match.
- `in_port`: This is the input port number that the flow entry should match.
- `out_port`: This is the output port number to which packets matching the flow entry should be sent.

Match Criteria:

- The code defines a match object using `ofp_parser.OFPMatch`. This object specifies the criteria that packets must meet in order for the flow entry to be applied.
- In this case, the match criteria include:
    - `in_port`: Packets must arrive on the specified input port (`in_port`).
    - `eth_dst`: Packets must have a destination MAC address that matches the specified value (`dst`).

Actions:

- The code defines a list of actions using `ofp_parser.OFPActionOutput`. This list specifies what actions should be taken when a packet matches the flow entry.
- In this case, the only action is to send the packet to the specified output port (`out_port`).

Instruction:

- The code combines the match criteria and actions into an instruction using `ofp_parser.OFPInstructionActions`. This instruction tells the switch what to do with packets that match the flow entry.

Flow Entry:

- The code creates a flow entry object using `ofp_parser.OFPFlowMod`. This object encapsulates all the information about the flow, including:
    - `datapath`: The switch where the flow entry will be added (same as the function argument).
    - `match`: The matching criteria defined earlier.
    - `instructions`: The instructions to be applied to matching packets.
    - `priority` (optional): An arbitrary priority level for the flow entry (higher priority entries are more likely to be applied). In this case, a priority of 1000 is set.

Adding the Flow:

- Finally, the code sends the newly created flow entry (`flow`) to the switch using `datapath.send_msg`. This adds the flow entry to the flow table on the switch.

More SDN Applications

◦ Smarter firewalls (distributed, content sensitive, …
• Full network load balancing (Multipath)
• Quality of Service (metering supported since OpenFlow 1.3)
• Network slicing
• Intrusion detection systems
• Overlay networks (e.g. VPN)
• Resilience and Failover mechanisms