Part 1: Router

Your goal will be to allow or block traffic between the different devices based on the 4 rules given below. **You will need to specify specific ports for all traffic (i.e., you cannot use flooding: `of.OFPP_FLOOD`) and the forwarding must be done by subnet,** (not by enumerating all of the IP addresses)**.** You might consider a method to determine if an IP Address is valid on a particular subnet (have you seen this function before? :)) Notice that the rules are written by identifying the subnet.

You may implement your router however you choose—although as a suggestion, you may find it easiest to determine the correct destination port by using the source and destination IP addresses or the source port on the switch from which the packet originated.

These files will get you started—you will need to modify both:
`lab3_topo_skel.py` / `lab3_controller_skel.py`

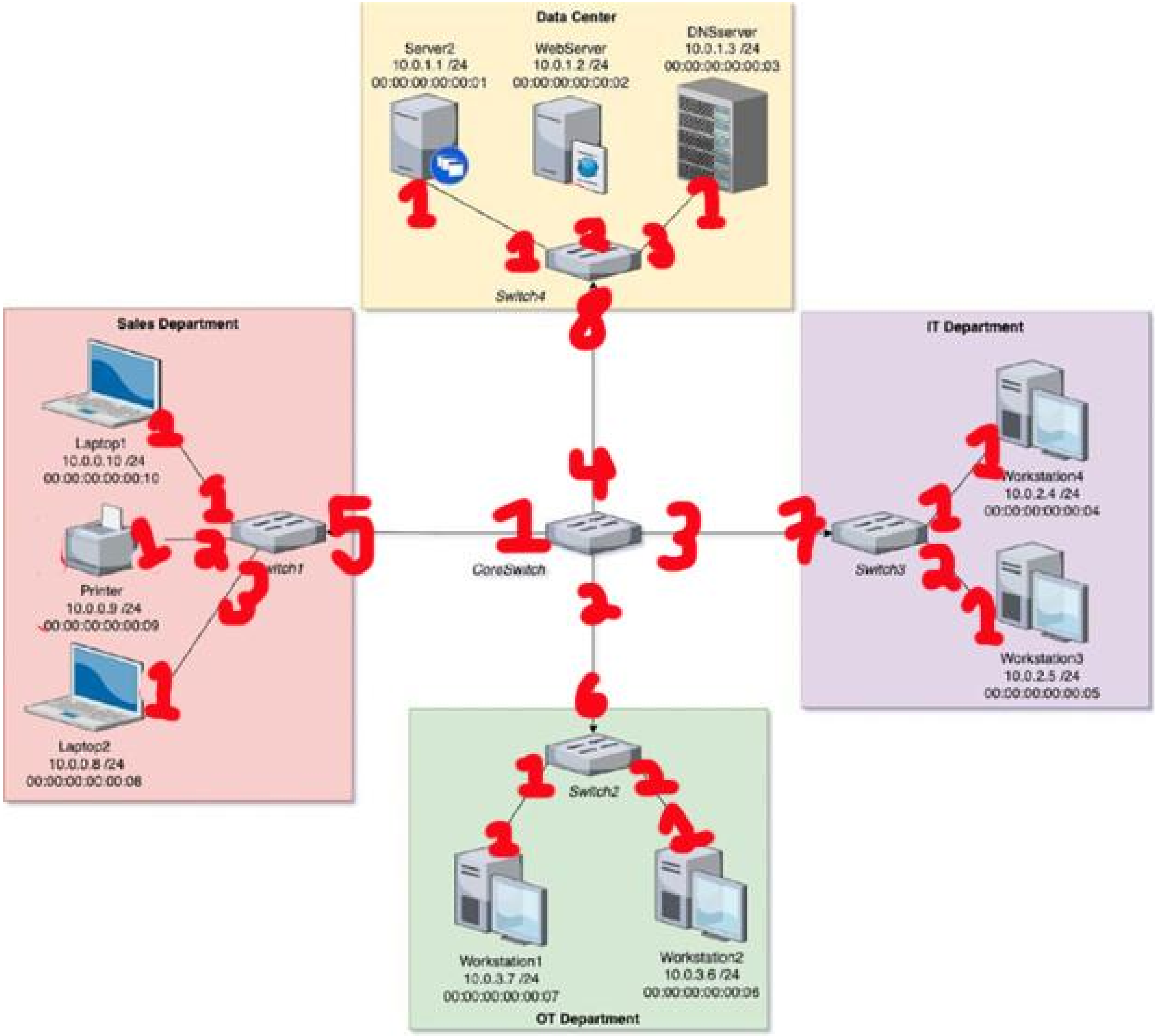
We will use the same topology as prelab3 and shown in Figure 1. The difference is that now the different departments have been organized into subnets. Finish the topology file by adding all the hosts (be sure to manually specify the MAC address, IP address and subnet for each host), switches and links, then implement the controller based on Rule 1, Rule 2 and Rule 3.

Network Topology and Rules to be implemented:

- **Rule 1:** `ICMP` traffic is forwarded only between the *Sales Department*, *IT Department* and *OT Department* subnets or between devices that are on the same subnet.
- **Rule 2:** `TCP` traffic is forwarded only between the *Datacenter*, *IT Department* and *OT Department* subnets or between devices that are on the same subnet.
- **Rule 3:** `UDP` traffic is forwarded only between the *Sales Department* and *Datacenter* subnets or between devices that are on the same subnet.
- **Rule 4:** All other traffic should be dropped.

Questions and Performance Verification:

1) Implement the topology in Figure 1 from the skeleton file. Annotate the topology with ports numbers associated with the end of each link and include a screenshot.

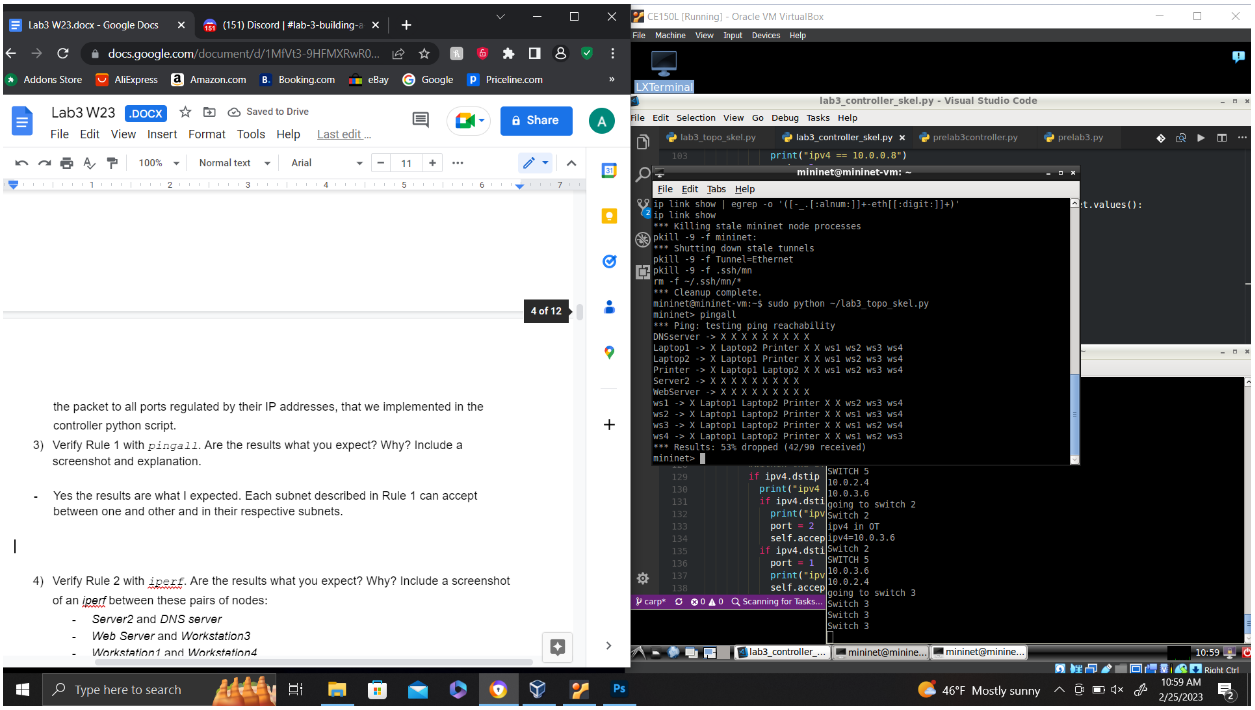


2) Explain how packets are forwarded differently in this assignment compared to Prelab3. Think about your `accept()` function in Prelab 3.

- In this lab packets are forwarded using specific ports for each host, switch, and the port to the core switch. In Prelab 3 everytime we would accept the packets we would flood the packet to all ports regulated by their IP addresses, that we implemented in the controller python script.

3) Verify Rule 1 with *`pingall`*. Are the results what you expect? Why? Include a screenshot and explanation.

- Yes the results are what I expected. Each subnet described in Rule 1 can accept between one and other and in their respective subnets.


Lab3 W23.docx - Google Docs
Lab3 W23 .DOCX
Saved to Drive
Share
4 of 12
CE150L [Running] - Oracle VM VirtualBox
lab3_controller_skel.py - Visual Studio Code
mininet@mininet-vm: ~

the packet to all ports regulated by their IP addresses, that we implemented in the controller python script.

3) Verify Rule 1 with `pingall`. Are the results what you expect? Why? Include a screenshot and explanation.

- Yes the results are what I expected. Each subnet described in Rule 1 can accept between one and other and in their respective subnets.

4) Verify Rule 2 with `iperf`. Are the results what you expect? Why? Include a screenshot of an *iperf* between these pairs of nodes:
   - *Server2* and *DNS server*
   - *Web Server* and *Workstation3*
   - *Workstation1* and *Workstation4*

```
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
mininet@mininet-vm:~$ sudo python ~/lab3_topo_skel.py
mininet> pingall
*** Ping: testing ping reachability
DNSserver -> X X X X X X X X X
Laptop1 -> X Laptop2 Printer X X ws1 ws2 ws3 ws4
Laptop2 -> X Laptop1 Printer X X ws1 ws2 ws3 ws4
Printer -> X Laptop1 Laptop2 X X ws1 ws2 ws3 ws4
Server2 -> X X X X X X X X X
WebServer -> X X X X X X X X X
ws1 -> X Laptop1 Laptop2 Printer X X ws2 ws3 ws4
ws2 -> X Laptop1 Laptop2 Printer X X ws1 ws3 ws4
ws3 -> X Laptop1 Laptop2 Printer X X ws1 ws2 ws4
ws4 -> X Laptop1 Laptop2 Printer X X ws1 ws2 ws3
*** Results: 53% dropped (42/90 received)
mininet>
```

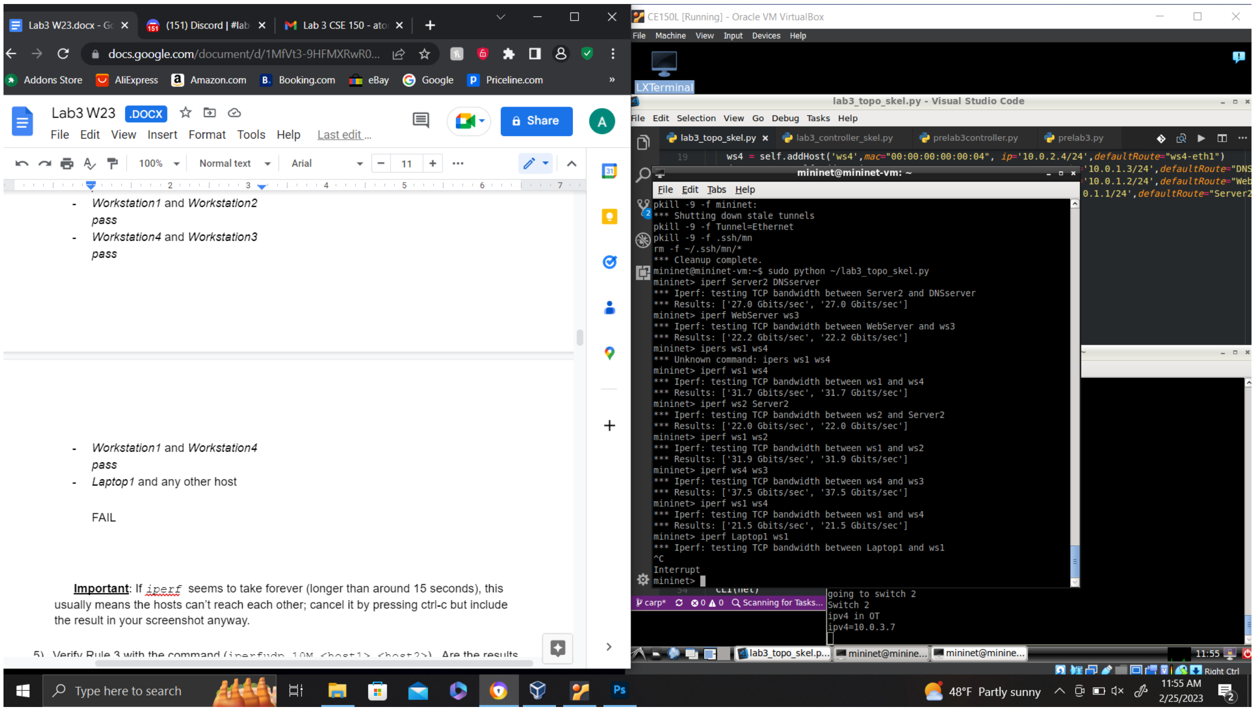
```
SWITCH 5
10.0.2.4
10.0.3.6
going to switch 2
Switch 2
ipv4 in OT
ipv4=10.0.3.6
Switch 2
SWITCH 5
10.0.3.6
10.0.2.4
going to switch 3
Switch 3
Switch 3
Switch 3
```

4) Verify Rule 2 with `iperf`. Are the results what you expect? Why? Include a screenshot of an *iperf* between these pairs of nodes:

- *Server2* and *DNS server :*
- *pass*
- *Web Server* and *Workstation3 :* *pass*
- *Workstation1* and *Workstation4* *pass*
- *Workstation2* and *Server2* *pass*
- *Workstation1* and *Workstation2* *pass*
- *Workstation4* and *Workstation3* *pass*
- *Workstation1* and *Workstation4* *pass*
- *Laptop1* and any other host

  FAIL

These are the results I expected because by implementing rule two, I have allowed TCP traffic to only flow between The Data Center, the IT Department, The OT Department and their respective subnets.



- *Workstation1* and *Workstation2*
  pass
- *Workstation4* and *Workstation3*
  pass

- *Workstation1* and *Workstation4*
  pass
- *Laptop1* and any other host

  FAIL

Important: If `iperf` seems to take forever (longer than around 15 seconds), this usually means the hosts can't reach each other; cancel it by pressing ctrl-c but include the result in your screenshot anyway.

```
ws4 = self.addHost('ws4',mac="00:00:00:00:00:04", ip='10.0.2.4/24',defaultRoute="ws4-eth1")
```

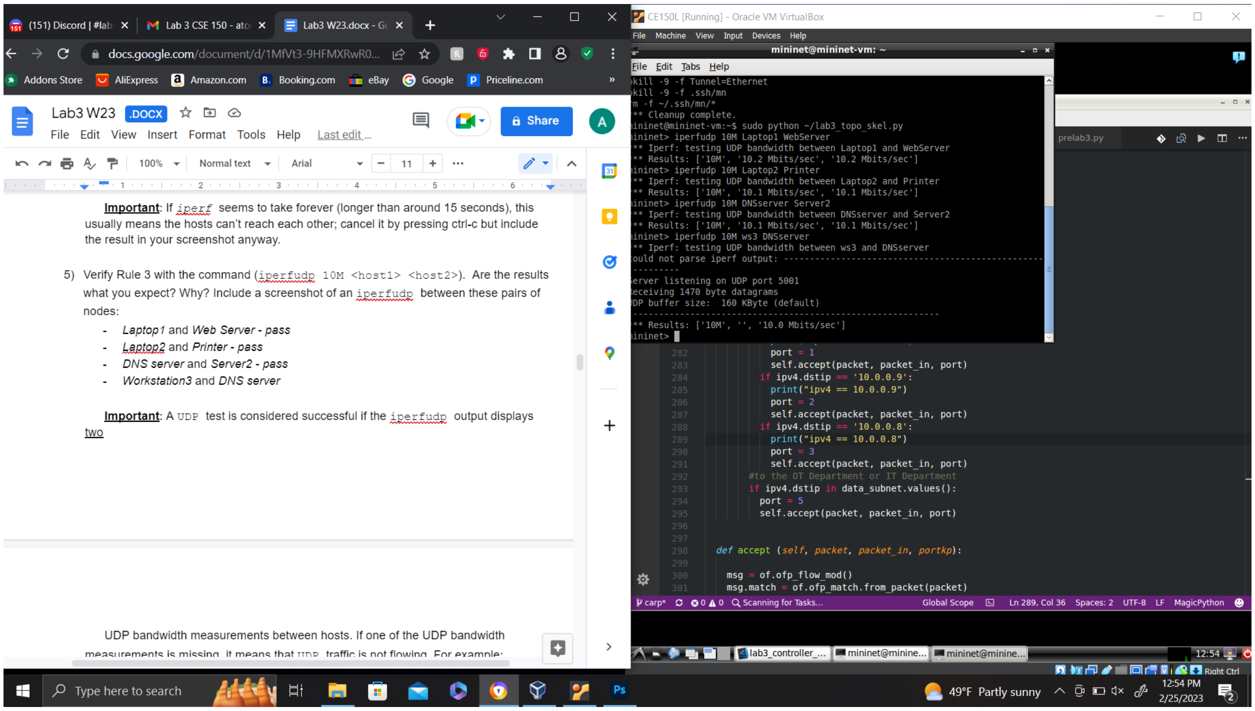
```
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
mininet@mininet-vm:~$ sudo python ~/lab3_topo_skel.py
mininet> iperf Server2 DNSserver
*** Iperf: testing TCP bandwidth between Server2 and DNSserver
*** Results: ['27.0 Gbits/sec', '27.0 Gbits/sec']
mininet> iperf WebServer ws3
*** Iperf: testing TCP bandwidth between WebServer and ws3
*** Results: ['22.2 Gbits/sec', '22.2 Gbits/sec']
mininet> ipers ws1 ws4
*** Unknown command: ipers ws1 ws4
mininet> iperf ws1 ws4
*** Iperf: testing TCP bandwidth between ws1 and ws4
*** Results: ['31.7 Gbits/sec', '31.7 Gbits/sec']
mininet> iperf ws2 Server2
*** Iperf: testing TCP bandwidth between ws2 and Server2
*** Results: ['22.0 Gbits/sec', '22.0 Gbits/sec']
mininet> iperf ws1 ws2
*** Iperf: testing TCP bandwidth between ws1 and ws2
*** Results: ['31.9 Gbits/sec', '31.9 Gbits/sec']
mininet> iperf ws4 ws3
*** Iperf: testing TCP bandwidth between ws4 and ws3
*** Results: ['37.5 Gbits/sec', '37.5 Gbits/sec']
mininet> iperf ws1 ws4
*** Iperf: testing TCP bandwidth between ws1 and ws4
*** Results: ['21.5 Gbits/sec', '21.5 Gbits/sec']
mininet> iperf Laptop1 ws1
*** Iperf: testing TCP bandwidth between Laptop1 and ws1
^C
Interrupt
mininet>
```

```
going to switch 2
Switch 2
ipv4 in OT
ipv4=10.0.3.7
```

Important: If `iperf` seems to take forever (longer than around 15 seconds), this usually means the hosts can't reach each other; cancel it by pressing ctrl-c but include the result in your screenshot anyway.

5) Verify Rule 3 with the command (`iperfudp 10M <host1> <host2>`). Are the results what you expect? Why? Include a screenshot of an `iperfudp` between these pairs of nodes:
   - *Laptop1* and *Web Server - pass*
   - *Laptop2* and *Printer - pass*
   - *DNS server* and *Server2 - pass*
   - *Workstation3* and *DNS server - fail*

- The results were as expected. The rule implemented was to allow UDP traffic between the Sales Department and the Data center, and in their respective subnets . The test above shows this and shows the UDP traffic not being allowed to pass between the Sales department and the IT Department.



Important: If *iperf* seems to take forever (longer than around 15 seconds), this usually means the hosts can't reach each other; cancel it by pressing ctrl-c but include the result in your screenshot anyway.

5) Verify Rule 3 with the command (`iperfudp 10M <host1> <host2>`). Are the results what you expect? Why? Include a screenshot of an `iperfudp` between these pairs of nodes:
    - *Laptop1* and *Web Server - pass*
    - *Laptop2* and *Printer - pass*
    - *DNS server* and *Server2 - pass*
    - *Workstation3* and *DNS server*

Important: A `UDP` test is considered successful if the `iperfudp` output displays two

UDP bandwidth measurements between hosts. If one of the UDP bandwidth measurements is missing, it means that `UDP` traffic is not flowing. For example:

```
kill -9 -f Tunnel=Ethernet
kill -9 -f .ssh/mn
m -f ~/.ssh/mn/*
** Cleanup complete.
ininet@mininet-vm:~$ sudo python ~/lab3_topo_skel.py
ininet> iperfudp 10M Laptop1 WebServer
** Iperf: testing UDP bandwidth between Laptop1 and WebServer
** Results: ['10M', '10.2 Mbits/sec', '10.2 Mbits/sec']
ininet> iperfudp 10M Laptop2 Printer
** Iperf: testing UDP bandwidth between Laptop2 and Printer
** Results: ['10M', '10.1 Mbits/sec', '10.1 Mbits/sec']
ininet> iperfudp 10M DNSserver Server2
** Iperf: testing UDP bandwidth between DNSserver and Server2
** Results: ['10M', '10.1 Mbits/sec', '10.1 Mbits/sec']
ininet> iperfudp 10M ws3 DNSserver
** Iperf: testing UDP bandwidth between ws3 and DNSserver
ould not parse iperf output: ------------------------------------------------------------
---------
erver listening on UDP port 5001
eceiving 1470 byte datagrams
DP buffer size:  160 KByte (default)
------------------------------------------------------------
** Results: ['10M', '', '10.0 Mbits/sec']
ininet> 
```

```
          port = 1
          self.accept(packet, packet_in, port)
        if ipv4.dstip == '10.0.0.9':
          print("ipv4 == 10.0.0.9")
          port = 2
          self.accept(packet, packet_in, port)
        if ipv4.dstip == '10.0.0.8':
          print("ipv4 == 10.0.0.8")
          port = 3
          self.accept(packet, packet_in, port)
      #to the OT Department or IT Department
      if ipv4.dstip in data_subnet.values():
        port = 5
        self.accept(packet, packet_in, port)


  def accept (self, packet, packet_in, portkp):

    msg = of.ofp_flow_mod()
    msg.match = of.ofp_match.from_packet(packet)
```

Important: A `UDP` test is considered successful if the `iperfudp` output displays two

UDP bandwidth measurements between hosts. If one of the UDP bandwidth measurements is missing, it means that `UDP` traffic is not flowing. For example:

Pass: `*** Results: ['10M', '10.1 Mbits/sec', '10.1 Mbits/sec']`

Failed: `*** Results: ['10M', '', '10.0 Mbits/sec']`

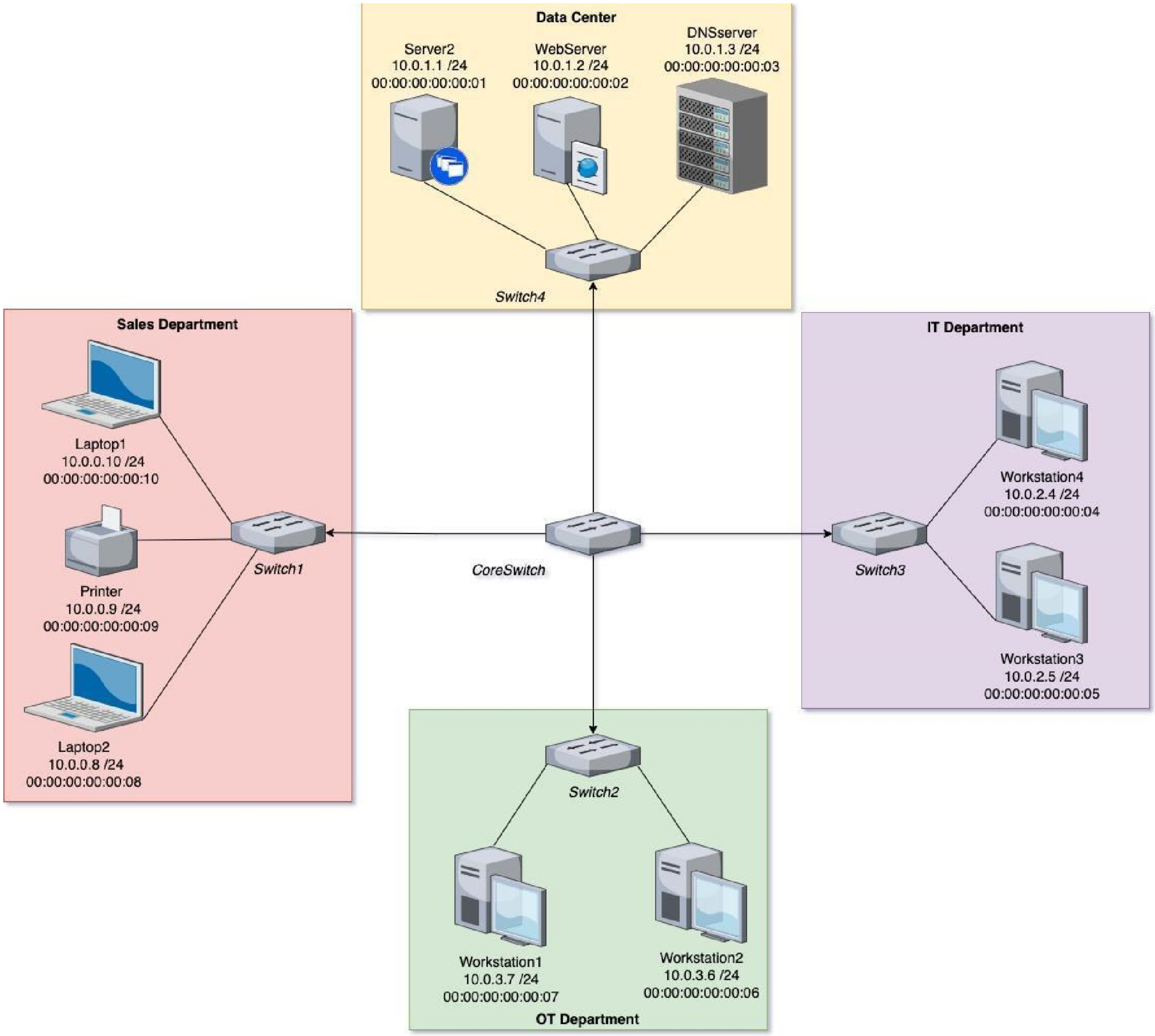

Data Center
Server2
10.0.1.1 /24
00:00:00:00:00:01
WebServer
10.0.1.2 /24
00:00:00:00:00:02
DNSserver
10.0.1.3 /24
00:00:00:00:00:03
Switch4
Sales Department
Laptop1
10.0.0.10 /24
00:00:00:00:00:10
Printer
10.0.0.9 /24
00:00:00:00:00:09
Laptop2
10.0.0.8 /24
00:00:00:00:00:08
Switch1
CoreSwitch
IT Department
Switch3
Workstation4
10.0.2.4 /24
00:00:00:00:00:04
Workstation3
10.0.2.5 /24
00:00:00:00:00:05
Switch2
Workstation1
10.0.3.7 /24
00:00:00:00:00:07
Workstation2
10.0.3.6 /24
00:00:00:00:00:06
OT Department

Figure 1