

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЗВІТ

з лабораторної роботи №4
з дисципліни ”Програмування комп’ютерних та віртуальних мереж”

Тема: Створення деревовидної SDN мережі в середовищі Mininet за допомогою Miniedit

Варіант №5

Виконав:
Студент 1 курсу, групи ІМ-51мн
Ковальов Олександр

Перевірив:
доцент, Долголенко Олександр Миколайович

Дата здачі: 16.10.2025

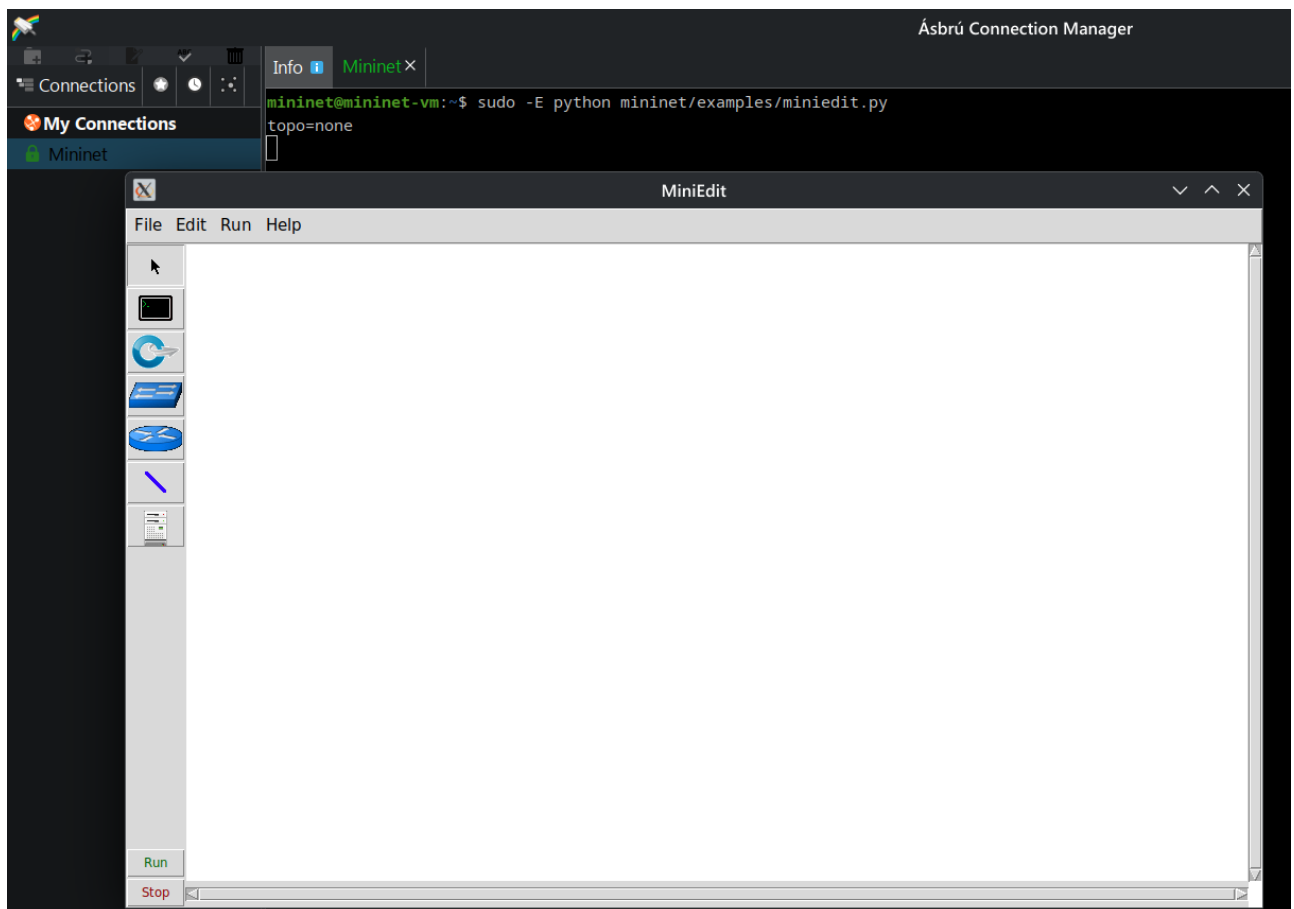
Мета роботи. Налаштувати та дослідити деревовидну топологію SDN мережі з різними рівнями комутаторів і підключених хостів, перевірити її працездатність і проаналізувати трафік за допомогою Wireshark.

Завдання: За допомогою скрипту `miniedit.py` Mininet (GUI) створити деревовидну топологію SDN мережі, що має глибину ієрархії комутаторів $\text{depth} = i \% 3 + 2$, а число підключених до кожного з них комутаторів, або хостів $\text{fanout} = i \% 2 + 2$, де i – номер в списку групи, а хости підключені тільки до комутаторів нижнього рівня. Продемонструвати працездатність топології з використанням Wireshark.

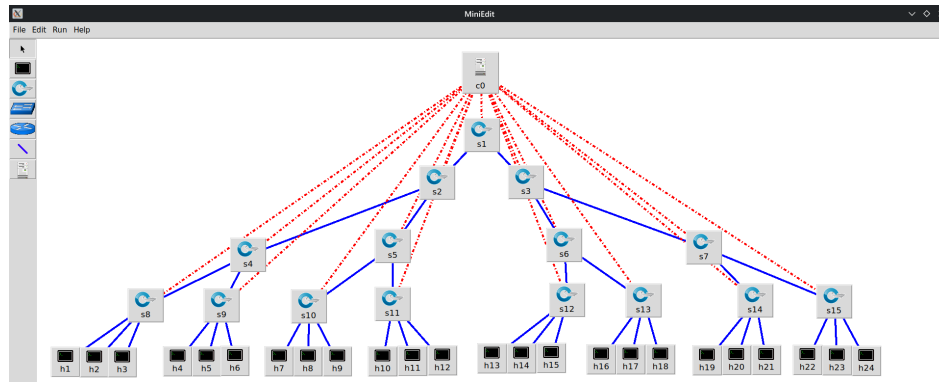
Хід роботи.

Номер в списку групи – 5, тому, відповідно, глибина ієрархії комутаторів = 4, а число пристроїв, підключених до кожного з них = 3.

Для того, щоб запустити Miniedit, треба налаштувати X11-форвардинг та мати встановлену змінну DISPLAY. Якщо все налаштовано правильно – буде відображений інтерфейс:



Була створена топологія з 15 комутаторами та 24 хостами:

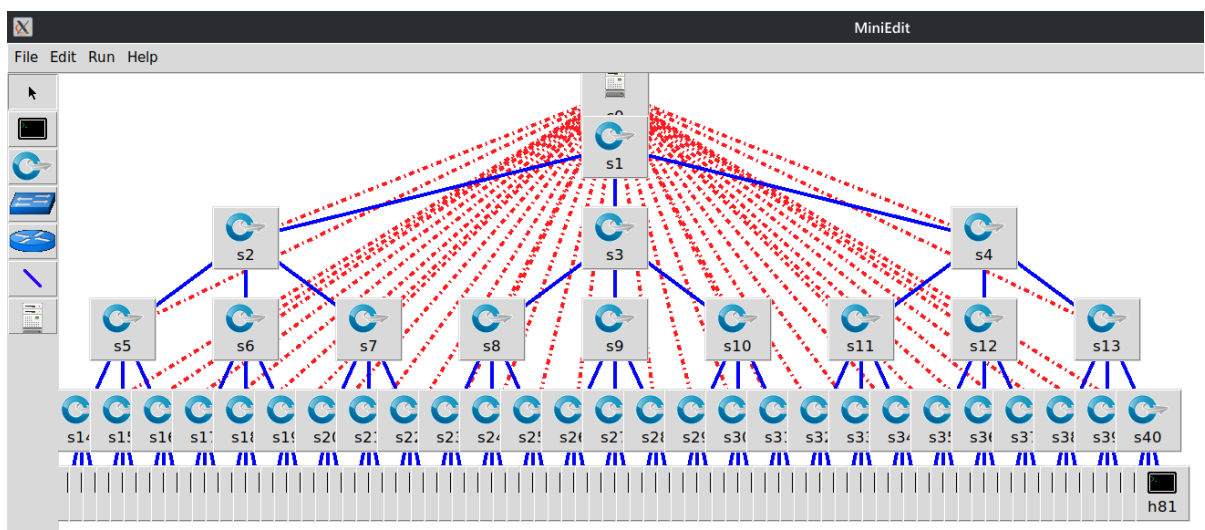


Цей приклад був створений вручну. Якщо слідувати завданню, то у кожного комутатора повинно бути 3 "спадкоємці" – загалом, виходить що топологія повинна мати 40 комутаторів та 81 хост. Це доволі складно реалізувати, тому був написаний скрипт, який реалізує JSON файл зі заданою топологією згідно завдання. Далі, його можна завантажити і працювати безпосередньо в середовищі MiniEdit.

Приклад застосування показаний на скріншоті. Аргумент "--branching" відповідає за кількість дочірніх комутаторів, "--levels" визначає кількість рівнів ієрархії, а "--hosts" – по скільки хостів треба мати кожному комутатору на нижньому рівні.

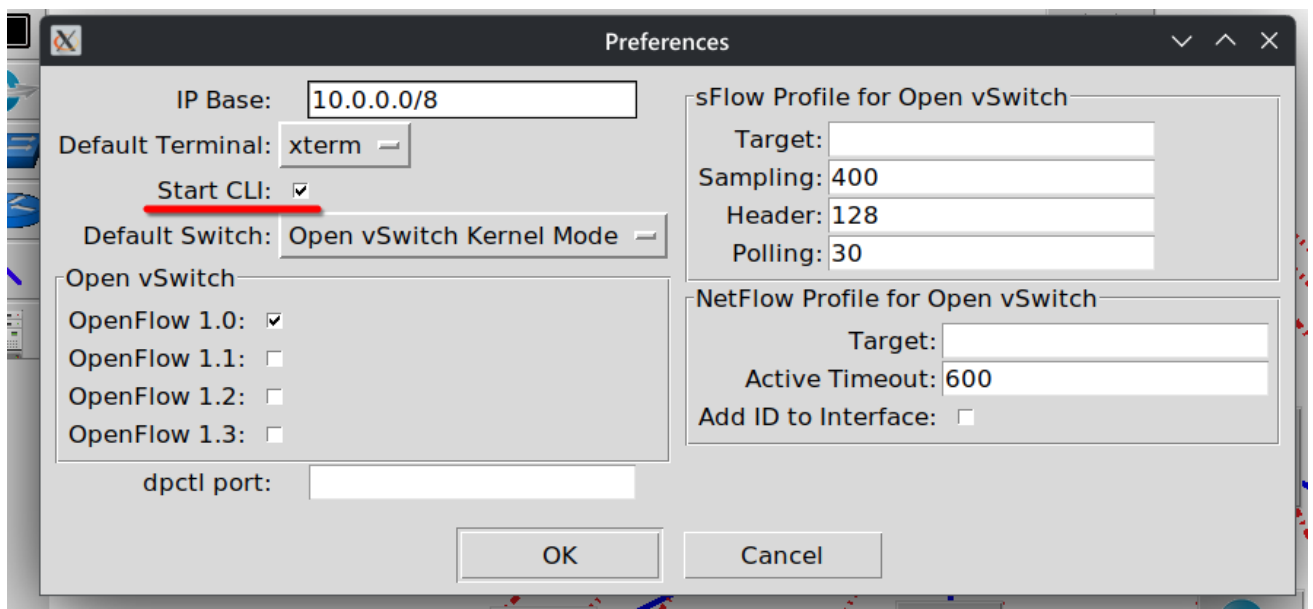
```
mininet@mininet-vm:~/Labs/Lab4$ python gui_topology_creator.py --branching 3 --levels 4 --hosts 3 --out topology.mn
Saved topology to topology.mn (branching=3, levels=4, hosts_per_bottom=3)
mininet@mininet-vm:~/Labs/Lab4$ echo "Kovalov Oleksandr, Lab 4"
Kovalov Oleksandr, Lab 4
mininet@mininet-vm:~/Labs/Lab4$
```

Після цього, власне, можна спробувати запустити топологію в графічному інтерфейсі. Відразу стає зрозуміло, що виставляти вручну всі пристрої було б недоцільною задачею.



Ще одною проблемою є застарілість програмного забезпечення системи Mininet. Проект вже давно закинутий, а деякі функції, такі як, наприклад, збереження топології в файл – не працюють. Тому, автором лабораторної роботи було створене рішення – декілька рядків програми Miniedit були змінені, а самі зміни були надіслані авторам проекту. Їх можна знайти за [цим посиланням](#). Також, доцільно весь файл Miniedit на віртуальній машині замінити на новий, безпосередньо з репозиторію Github. Якщо ж при цьому виникне помилка пов'язана з файлом `utils.py` – то його теж бажано замінити новим кодом, за [даним посиланням](#).

Щоб запустити симуляцію, потрібно натиснути "Run". Щоб переглянути всі події в консолі, треба увімкнути її перед цим. Ця опція доступна в меню "Edit" → "Preferences" → "Start CLI":



Щоб мати змогу протестувати мережу, треба перейти в термінал:



Далі, повертаючись до терміналу, можна почати тестувати мережу. Наприклад, пропінгувати хости з різних кінців дерева:

```
mininet@mininet-vm:~/Labs/Lab4$ sudo -E python ../../mininet/examples/miniedit.py
topo=None
New Prefs = {'ipBase': '10.0.0.0/8', 'terminalType': 'xterm', 'dpctl': '', 'sflow': {'sflowTarget': '', 'sflowSampling': '400', 'sflowHead': 'nflowTarget': '', 'nflowTimeout': '600', 'nflowAddId': '0'}, 'startCLI': '1', 'switchType': 'ovs', 'openFlowVersions': {'ovsOf10': '1', 'ovsOf12': '1'}}
Getting Hosts and Switches.
Getting controller selection:ref
Getting Links.
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64 h65 h66 h67 h68 h69 h70 h71 h72 h73 h74 h75 h76 h77 h78 h79 h80 h81
**** Starting 1 controllers
c0
**** Starting 40 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s22 s23 s24 s25 s26 s27 s28 s29 s30 s31 s32 s33 s34 s35 s36 s37 s38 s39 s40
No NetFlow targets specified.
No sFlow targets specified.

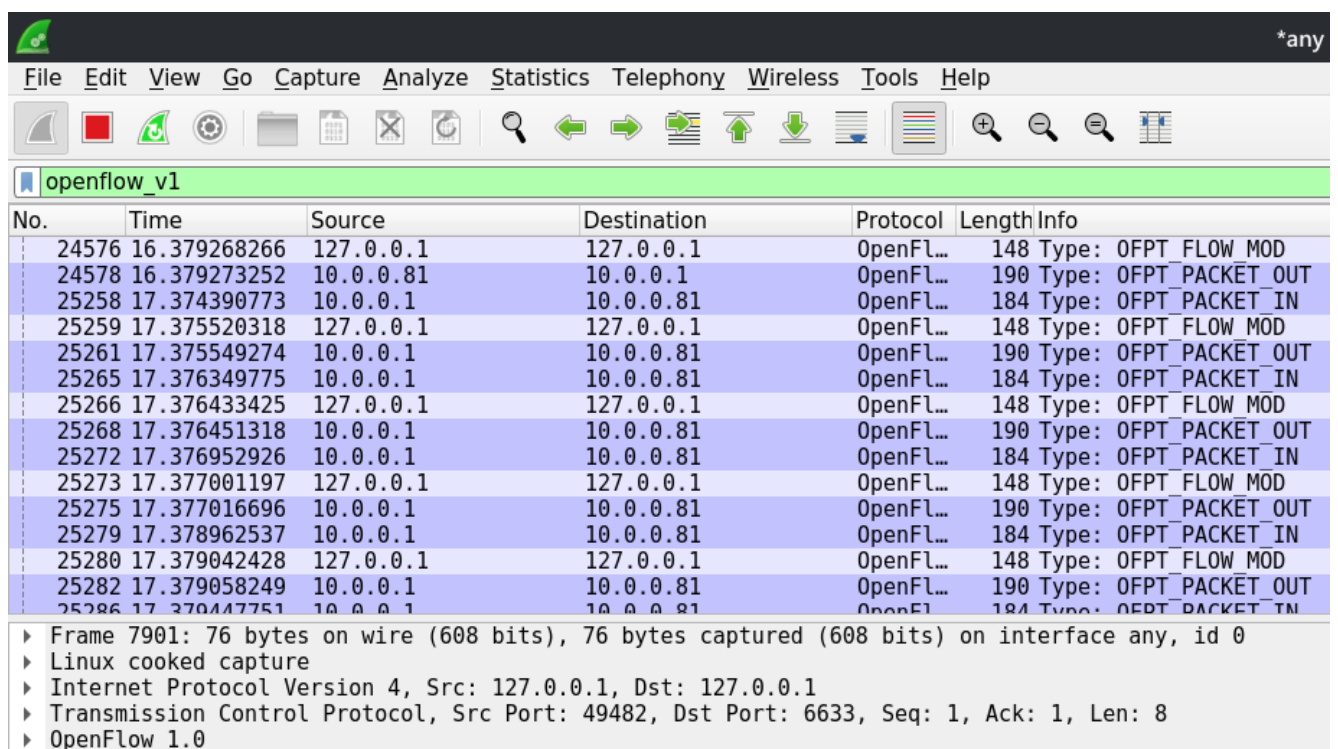
NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exiting will prevent MiniEdit from quitting and will prevent your session.

*** Starting CLI:
mininet> h1 ping h81
PING 10.0.0.81 (10.0.0.81) 56(84) bytes of data.
64 bytes from 10.0.0.81: icmp_seq=1 ttl=64 time=16.0 ms
64 bytes from 10.0.0.81: icmp_seq=2 ttl=64 time=1.11 ms
64 bytes from 10.0.0.81: icmp_seq=3 ttl=64 time=0.158 ms
^C
--- 10.0.0.81 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.158/5.749/15.980/7.244 ms
mininet>
```

В окремому підключенні до віртуальної машини запусимо Wireshark:

```
1 mininet@mininet-vm:~$ sudo -E wireshark &
2 [1] 2958
```

Всі пакети відображаються – мережа працює.



The screenshot shows the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons for file operations, capture control, and analysis. The packet list pane on the left shows a single capture named 'openflow_v1'. The main packet details pane displays a list of captured packets with columns for No., Time, Source, Destination, Protocol, Length, and Info. The selected packet (No. 25286) is an OpenFlow 1.0 packet from 10.0.0.1 to 10.0.0.81. The packet details pane shows the structure of the OpenFlow 1.0 packet, including the Frame 7901, Linux cooked capture, Internet Protocol Version 4, Transmission Control Protocol, and OpenFlow 1.0 details.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-------|--------------|-----------|-------------|-----------|--------|-----------------------|
| 24576 | 16.379268266 | 127.0.0.1 | 127.0.0.1 | OpenFl... | 148 | Type: OFPT_FLOW_MOD |
| 24578 | 16.379273252 | 10.0.0.81 | 10.0.0.1 | OpenFl... | 190 | Type: OFPT_PACKET_OUT |
| 25258 | 17.374390773 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 184 | Type: OFPT_PACKET_IN |
| 25259 | 17.375520318 | 127.0.0.1 | 127.0.0.1 | OpenFl... | 148 | Type: OFPT_FLOW_MOD |
| 25261 | 17.375549274 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 190 | Type: OFPT_PACKET_OUT |
| 25265 | 17.376349775 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 184 | Type: OFPT_PACKET_IN |
| 25266 | 17.376433425 | 127.0.0.1 | 127.0.0.1 | OpenFl... | 148 | Type: OFPT_FLOW_MOD |
| 25268 | 17.376451318 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 190 | Type: OFPT_PACKET_OUT |
| 25272 | 17.376952926 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 184 | Type: OFPT_PACKET_IN |
| 25273 | 17.377001197 | 127.0.0.1 | 127.0.0.1 | OpenFl... | 148 | Type: OFPT_FLOW_MOD |
| 25275 | 17.377016696 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 190 | Type: OFPT_PACKET_OUT |
| 25279 | 17.378962537 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 184 | Type: OFPT_PACKET_IN |
| 25280 | 17.379042428 | 127.0.0.1 | 127.0.0.1 | OpenFl... | 148 | Type: OFPT_FLOW_MOD |
| 25282 | 17.379058249 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 190 | Type: OFPT_PACKET_OUT |
| 25286 | 17.379447751 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 184 | Type: OFPT_PACKET_IN |

Frame 7901: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface any, id 0
Linux cooked capture
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 49482, Dst Port: 6633, Seq: 1, Ack: 1, Len: 8
OpenFlow 1.0

Окрім цього, можна помітити цікавий момент – дійсно, всі пакети дублюються і слугують корисним навантаженням на рівні OpenFlow.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-------|--------------|-----------|-------------|-----------|--------|-----------------------|
| 24576 | 16.379268266 | 127.0.0.1 | 127.0.0.1 | OpenFl... | 148 | Type: OFPT FLOW MOD |
| 24578 | 16.379273252 | 10.0.0.81 | 10.0.0.1 | OpenFl... | 190 | Type: OFPT PACKET OUT |
| 25258 | 17.374390773 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 184 | Type: OFPT PACKET IN |
| 25259 | 17.375520318 | 127.0.0.1 | 127.0.0.1 | OpenFl... | 148 | Type: OFPT FLOW MOD |
| 25261 | 17.375549274 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 190 | Type: OFPT PACKET OUT |
| 25265 | 17.376349775 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 184 | Type: OFPT PACKET IN |
| 25266 | 17.376433425 | 127.0.0.1 | 127.0.0.1 | OpenFl... | 148 | Type: OFPT FLOW MOD |
| 25268 | 17.376451318 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 190 | Type: OFPT PACKET OUT |
| 25272 | 17.376952926 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 184 | Type: OFPT PACKET IN |
| 25273 | 17.377001197 | 127.0.0.1 | 127.0.0.1 | OpenFl... | 148 | Type: OFPT FLOW MOD |
| 25275 | 17.377016696 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 190 | Type: OFPT PACKET OUT |
| 25279 | 17.378962537 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 184 | Type: OFPT PACKET IN |
| 25280 | 17.379042428 | 127.0.0.1 | 127.0.0.1 | OpenFl... | 148 | Type: OFPT FLOW MOD |
| 25282 | 17.379058249 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 190 | Type: OFPT PACKET OUT |
| 25286 | 17.379447751 | 10.0.0.1 | 10.0.0.81 | OpenFl... | 184 | Type: OFPT PACKET IN |

▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 ▶ Transmission Control Protocol, Src Port: 49264, Dst Port: 6633, Seq: 257, Ack: 349, Len: 116
 ▼ **OpenFlow 1.0**
 .000 0001 = Version: 1.0 (0x01)
 Type: OFPT_PACKET_IN (10)
 Length: 116
 Transaction ID: 0
 Buffer Id: 0xffffffff
 Total length: 98
 In port: 2
 Reason: No matching flow (table-miss flow entry) (0)
 Pad: 00
 ▶ Ethernet II, Src: 26:3c:04:76:6f:ce (26:3c:04:76:6f:ce), Dst: b6:63:76:5f:c6:8c (b6:63:76:5f:c6:8c)
 ▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.81
 ▶ Internet Control Message Protocol

```

0000  00 00 03 04 00 06 00 00 00 00 00 00 ff ff 08 00  .....
0010  45 c0 00 a8 51 30 40 00 40 06 ea 5d 7f 00 00 01  E...Q0@. @..]...
0020  7f 00 00 01 c0 70 19 e9 80 44 f9 b0 63 de bb ff  ....p...D..c...
0030  80 18 00 56 fe 9c 00 00 01 01 08 0a c7 c7 f1 e9  ...V.....
0040  c7 c7 ee 02 01 0a 00 74 00 00 00 00 ff ff ff ff  ....t.....
0050  00 62 00 02 00 00 b6 63 76 5f c6 8c 26 3c 04 76  .b....c v...&<.v
0060  6f ce 08 00 45 00 00 54 f7 20 40 00 40 01 2f 37  o...E..T _@./7
0070  0a 00 00 01 0a 00 00 51 08 00 31 6d 42 eb 00 02  ....Q...ImB...
0080  63 bd f2 68 00 00 00 00 62 ac 0c 00 00 00 00 00  c..h...b.....
0090  10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f  .....
00a0  20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f  !"#%&'()*+,-./
00b0  30 31 32 33 34 35 36 37 01234567
  
```

Висновок. У роботі створено та протестовано деревоподібну SDN-топологію – 40 комутаторів і 81 хост. Тестування зв'язності (pingall, пінги між листовими вузлами) підтвердило коректну роботу мережі; у Wireshark зафіксовано OpenFlow-повідомлення (PACKET_IN, PACKET_OUT, FLOW_MOD), що свідчить про роботу контролера та динамічне встановлення правил. Завдання виконано.

Лістинг.

gui_topology_creator.py

```
1  #!/usr/bin/env python3
2  """
3  gui_topology_creator.py
4
5  Creating tree topology and saving it as JSON (.mn) file.
6
7  Usage example:
8  python gui_topology_creator.py --branching 2 --levels 3 --hosts 2 --out topology.mn
9  """
10
11  import json
12  import argparse
13  import math
14
15  BASE_APPLICATION = {
16      "dpctl": "",
17      "ipBase": "10.0.0.0/8",
18      "netflow": {
19          "nflowAddId": "0",
20          "nflowTarget": "",
21          "nflowTimeout": "600"
22      },
23      "openFlowVersions": {
24          "ovsOf10": "1",
25          "ovsOf11": "0",
26          "ovsOf12": "0",
27          "ovsOf13": "0"
28      },
29      "sflow": {
30          "sflowHeader": "128",
31          "sflowPolling": "30",
32          "sflowSampling": "400",
33          "sflowTarget": ""
34      },
35      "startCLI": "0",
36      "switchType": "ovs",
37      "terminalType": "xterm"
38  }
39
40  CONTROLLER_TEMPLATE = {
41      "opts": {
42          "controllerProtocol": "tcp",
43          "controllerType": "ref",
44          "hostname": "c0",
45          "remoteIP": "127.0.0.1",
46          "remotePort": 6633
47      },
```

```

48     "x": "0.0",
49     "y": "0.0"
50 }
51
52
53 def build_tree(branching: int, levels: int, hosts_per_bottom: int):
54     if levels < 1:
55         raise ValueError("levels must be >= 1")
56     if branching < 1:
57         raise ValueError("branching must be >= 1")
58     if hosts_per_bottom < 0:
59         raise ValueError("hosts_per_bottom must be >= 0")
60
61     controllers = [CONTROLLER_TEMPLATE.copy()]
62
63     switches = []
64     hosts = []
65     links = []
66
67     # Build tree structure and remember parent->children mapping
68     level_nodes = [] # list of lists: level_nodes[0] = [root], ..., level_nodes[-1] = bottom
69     switches
70     parent_children = {} # parent_name -> list of child switch names
71     next_switch_id = 1
72
73     # create root switch
74     root_name = f"s{next_switch_id}"
75     level_nodes.append([root_name])
76     switch_by_name = {}
77     switch_obj = {
78         "number": str(next_switch_id),
79         "opts": {
80             "controllers": ["c0"],
81             "hostname": root_name,
82             "nodeNum": next_switch_id,
83             "switchType": "default"
84         },
85         "x": "0.0",
86         "y": "0.0"
87     }
88     switches.append(switch_obj)
89     switch_by_name[root_name] = switch_obj
90     next_switch_id += 1
91
92     # create child switch levels and links (child -> parent)
93     for lvl in range(1, levels):
94         parents = level_nodes[lvl - 1]
95         this_level = []
96         for p in parents:

```

```

97     name = f"s{next_switch_id}"
98     this_level.append(name)
99     obj = {
100         "number": str(next_switch_id),
101         "opts": {
102             "controllers": ["c0"],
103             "hostname": name,
104             "nodeNum": next_switch_id,
105             "switchType": "default"
106         },
107         "x": "0.0",
108         "y": "0.0"
109     }
110     switches.append(obj)
111     switch_by_name[name] = obj
112     # record parent->child
113     parent_children.setdefault(p, []).append(name)
114     # link child -> parent
115     links.append({
116         "dest": p,
117         "opts": {},
118         "src": name
119     })
120     next_switch_id += 1
121     level_nodes.append(this_level)
122
123     # attach hosts to every switch in the last level (bottom)
124     host_id = 1
125     bottom_level = level_nodes[-1]
126     for s in bottom_level:
127         for i in range(hosts_per_bottom):
128             hname = f"h{host_id}"
129             hosts.append({
130                 "number": str(host_id),
131                 "opts": {
132                     "hostname": hname,
133                     "nodeNum": host_id,
134                     "sched": "host"
135                 },
136                 "x": "0.0", # will set numeric x immediately
137                 "y": "0.0" # will set actual y after switch y computed
138             })
139     # link host -> its switch (src host, dest switch)
140     links.append({
141         "dest": s,
142         "opts": {},
143         "src": hname
144     })
145     host_id += 1
146

```

```

147 # Coordinate policy parameters
148 host_x_start = 10.0
149 host_x_step = 15.0 # produces 10,20,30,...
150 controller_y = 20.0
151 switch_y_start = 80.0 # root level y
152 y_spacing = 100.0 # distance between switch levels
153 host_y_offset = 80.0 # hosts below the bottom switches
154
155 # assign X to hosts in a simple left-to-right sequence: 10,20,30,...
156 total_hosts = len(hosts)
157 host_xs = [host_x_start + i * host_x_step for i in range(total_hosts)]
158 for idx, h in enumerate(hosts):
159     h["x"] = f"{host_xs[idx]:.1f}"
160     # y left as placeholder; we'll compute after computing switch Y
161
162 # compute X for bottom switches:
163 switch_xs = {} # name -> numeric x
164 # hosts are ordered in the same sequence we created them: for each bottom switch it has
165 # hosts_per_bottom hosts sequentially
166 if hosts_per_bottom > 0:
167     # mapping: for each bottom switch in order, take next hosts_per_bottom host_xs
168     hi = 0
169     for s in bottom_level:
170         slice_xs = host_xs[hi:hi + hosts_per_bottom]
171         if slice_xs:
172             sx = sum(slice_xs) / len(slice_xs)
173         else:
174             sx = host_x_start + hi * host_x_step
175         switch_xs[s] = sx
176         hi += hosts_per_bottom
177 else:
178     # no hosts attached: spread bottom switches evenly across a small span
179     n = len(bottom_level)
180     if n == 1:
181         switch_xs[bottom_level[0]] = host_x_start
182     else:
183         span = (n - 1) * host_x_step * 2
184         for i, s in enumerate(bottom_level):
185             switch_xs[s] = host_x_start - span / 2 + i * (span / max(1, n - 1))
186
187 # compute X for higher-level switches by averaging their children's X
188 # process levels from bottom-1 up to root
189 for lvl in range(levels - 2, -1, -1):
190     for parent in level_nodes[lvl]:
191         children = parent_children.get(parent, [])
192         if not children:
193             # isolated parent (no children) - keep its X if exists or set to 0
194             sx = switch_xs.get(parent, host_x_start)
195         else:
196             child_xs = [switch_xs[child] for child in children]

```

```

196         sx = sum(child_xs) / len(child_xs)
197         switch_xs[parent] = sx
198
199     # now set switch objects' x and y strings
200     for lvl, nodes in enumerate(level_nodes):
201         y = switch_y_start + lvl * y_spacing
202         for name in nodes:
203             sx = switch_xs.get(name, host_x_start)
204             sw = switch_by_name[name]
205             sw["x"] = f"{sx:.1f}"
206             sw["y"] = f"{y:.1f}"
207
208     # set hosts' y based on their parent switch Y (each host was sequentially assigned to bottom
209     switches)
210     if hosts_per_bottom > 0:
211         hi = 0
212         for s in bottom_level:
213             parent_sw = switch_by_name[s]
214             py = float(parent_sw["y"])
215             for i in range(hosts_per_bottom):
216                 if hi >= len(hosts):
217                     break
218                 hosts[hi]["y"] = f"{py + host_y_offset:.1f}"
219                 hi += 1
220     else:
221         # no hosts, nothing to update
222         pass
223
224     # controller coordinates: center above root
225     root_x = switch_xs.get(level_nodes[0][0], host_x_start)
226     controllers[0]["x"] = f"{root_x:.1f}"
227     controllers[0]["y"] = f"{controller_y:.1f}"
228
229     result = {
230         "application": BASE_APPLICATION,
231         "controllers": controllers,
232         "hosts": hosts,
233         "links": links,
234         "switches": switches,
235         "version": "2"
236     }
237     return result
238
239 def main():
240     p = argparse.ArgumentParser(description="Build tree topology JSON")
241     p.add_argument("--branching", "-b", type=int, default=2,
242                   help="number of child switches per switch (branching factor)")
243     p.add_argument("--levels", "-l", type=int, default=2,
244                   help="number of switch levels (1 = root only)")

```

```

245     p.add_argument("--hosts", "-s", dest="hosts", type=int, default=1,
246                  help="number of hosts per bottom-level switch")
247     p.add_argument("--out", "-o", type=str, default="topology.json",
248                  help="output JSON filename")
249     args = p.parse_args()
250
251     topo = build_tree(args.branching, args.levels, args.hosts)
252
253     with open(args.out, "w", encoding="utf-8") as f:
254         json.dump(topo, f, ensure_ascii=False, indent=4)
255         print(f"Saved topology to {args.out} (branching={args.branching}, levels={args.levels},
256              hosts_per_bottom={args.hosts})")
257
258 if __name__ == "__main__":
259     main()
260

```