

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
Ковалев Олександр

Перевірив:
доцент, Долголенко Олександр Миколайович

Дата здачі: 16.10.2025

КИЇВ – 2025

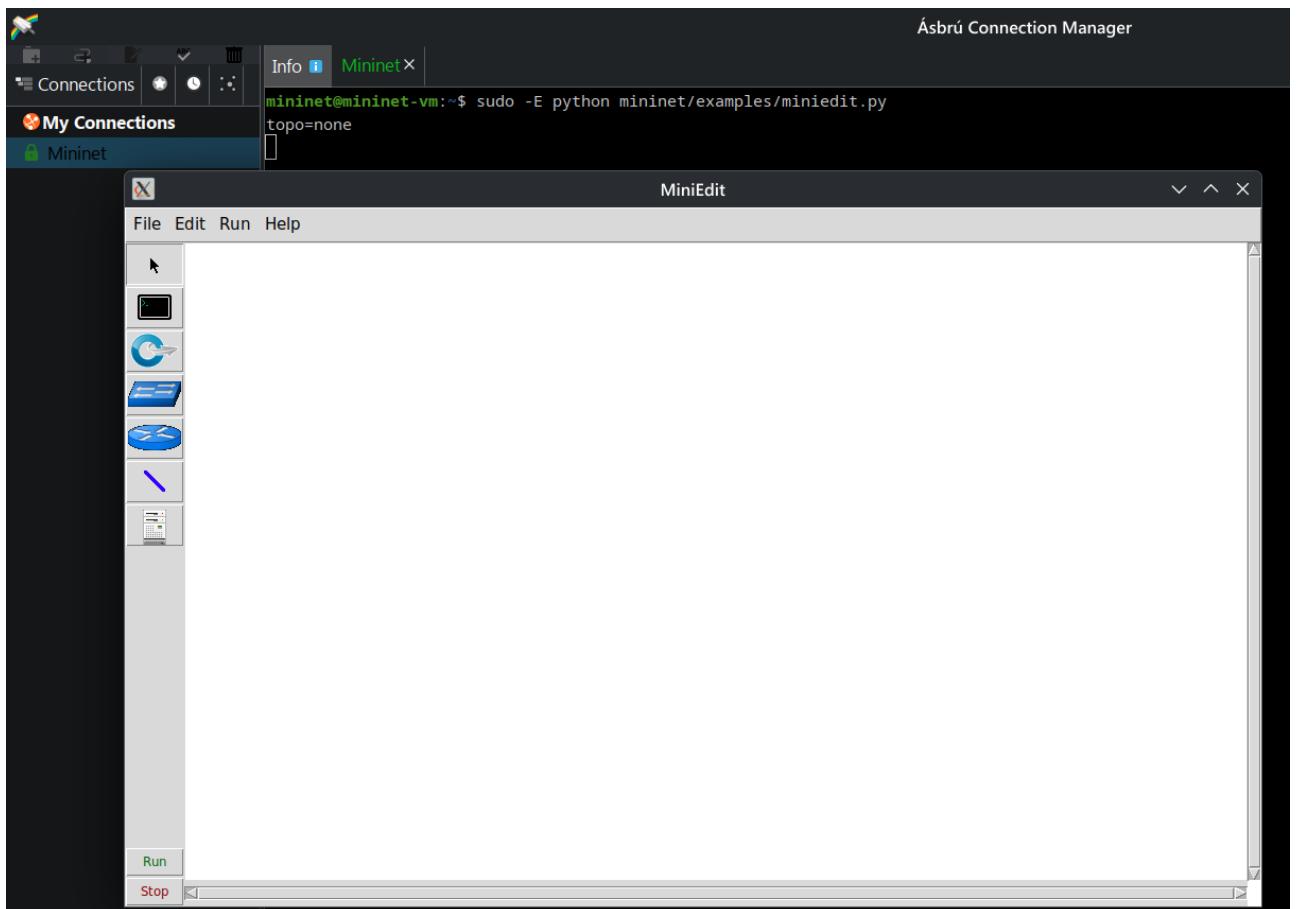
Мета роботи. Налаштувати та дослідити деревовидну топологію SDN мережі з різними рівнями комутаторів і підключених хостів, перевірити її працездатність і проаналізувати трафік за допомогою Wireshark.

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

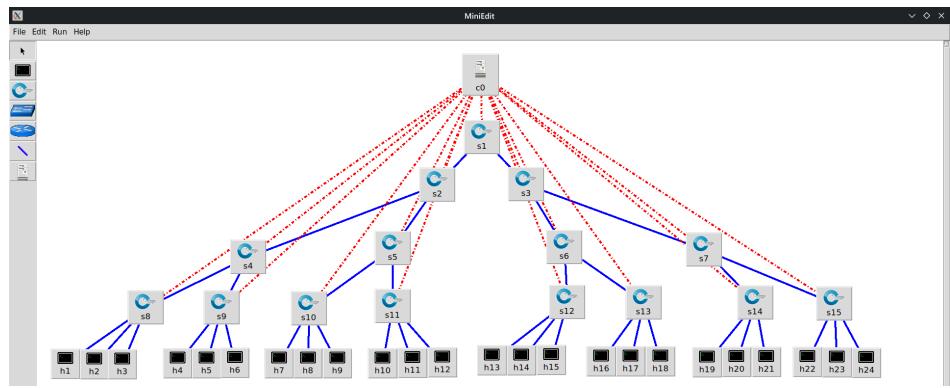
Хід роботи.

Номер в списку групи – 5, тому, відповідно, глибина ієрархії комутаторів = 4, а число пристройів, 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 відображенний інтерфейс:



Була створена топологія з 15 комутаторами та 24 хостами:

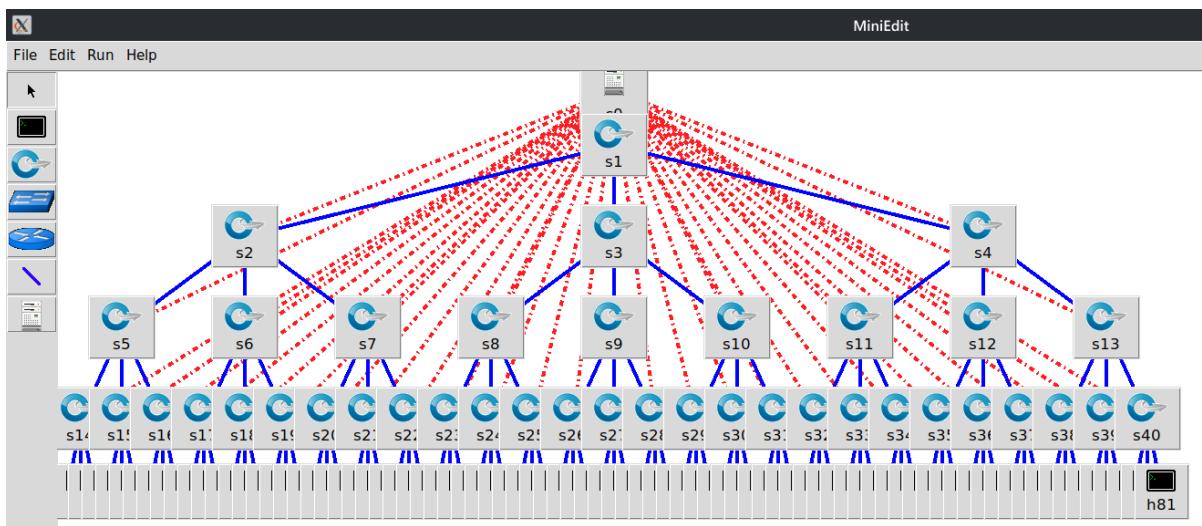


Цей приклад був створений вручну. Якщо слідувати завданню, то у кожного комутатора повинно бути 3 ”спадкоємці” – загалом, виходить що топологія повинна мати 40 комутаторів та 81 хост. Це доволі складно реалізувати, тому був написаний скрипт, який реалізує JSON файл зі заданою топологією згідно завдання. 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 ієархії, а ”–hosts” – по скільки хостів треба мати кожному комутатору на нижньому рівні.

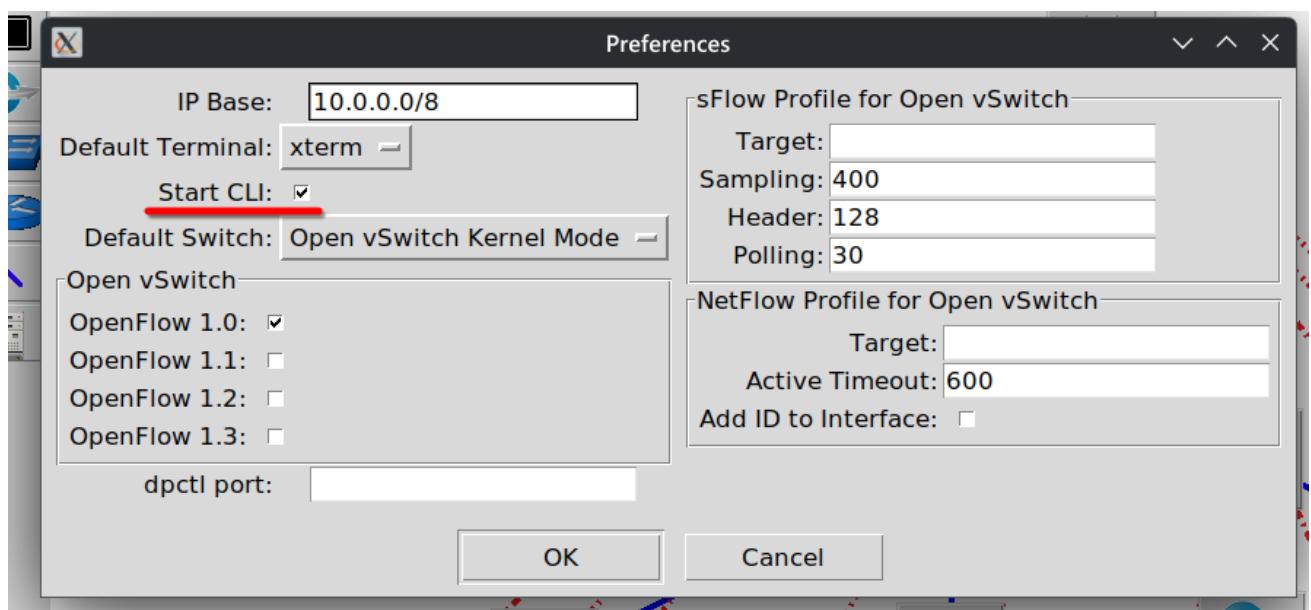
```
mininet@mininet-vm:~/Labs/Lab4$ python gui_topology_creator.py --branching 3 --levels 4 --hosts 3 --out topology.mn
Saved topology to topology.mn (branching=3, levels=4, hosts_per_bottom=3)
mininet@mininet-vm:~/Labs/Lab4$ echo "Kovalov Oleksandr, Lab 4"
Kovalov Oleksandr, Lab 4
mininet@mininet-vm:~/Labs/Lab4$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

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 тестиувати 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', 'sflowHeader': '1', 'nflowTarget': '', 'nflowTimeout': '600', 'nflowAddId': '0'}, 'startCLI': '1', 'switchType': 'ovs', 'openFlowVersions': {'ovsOf10': '1', 'ovsOf12': '1', 'ovsOf13': '1', 'ovsOf14': '1', 'ovsOf15': '1', 'ovsOf16': '1', 'ovsOf17': '1', 'ovsOf18': '1', 'ovsOf19': '1', 'ovsOf20': '1', 'ovsOf21': '1', 'ovsOf22': '1', 'ovsOf23': '1', 'ovsOf24': '1', 'ovsOf25': '1', 'ovsOf26': '1', 'ovsOf27': '1', 'ovsOf28': '1', 'ovsOf29': '1', 'ovsOf30': '1', 'ovsOf31': '1', 'ovsOf32': '1', 'ovsOf33': '1', 'ovsOf34': '1', 'ovsOf35': '1', 'ovsOf36': '1', 'ovsOf37': '1', 'ovsOf38': '1', 'ovsOf39': '1', 'ovsOf40': '1'}, 'controller': 'c0', 'links': 'l1', 'net': 'net'}
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 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
No NetFlow targets specified.
No sFlow targets specified.

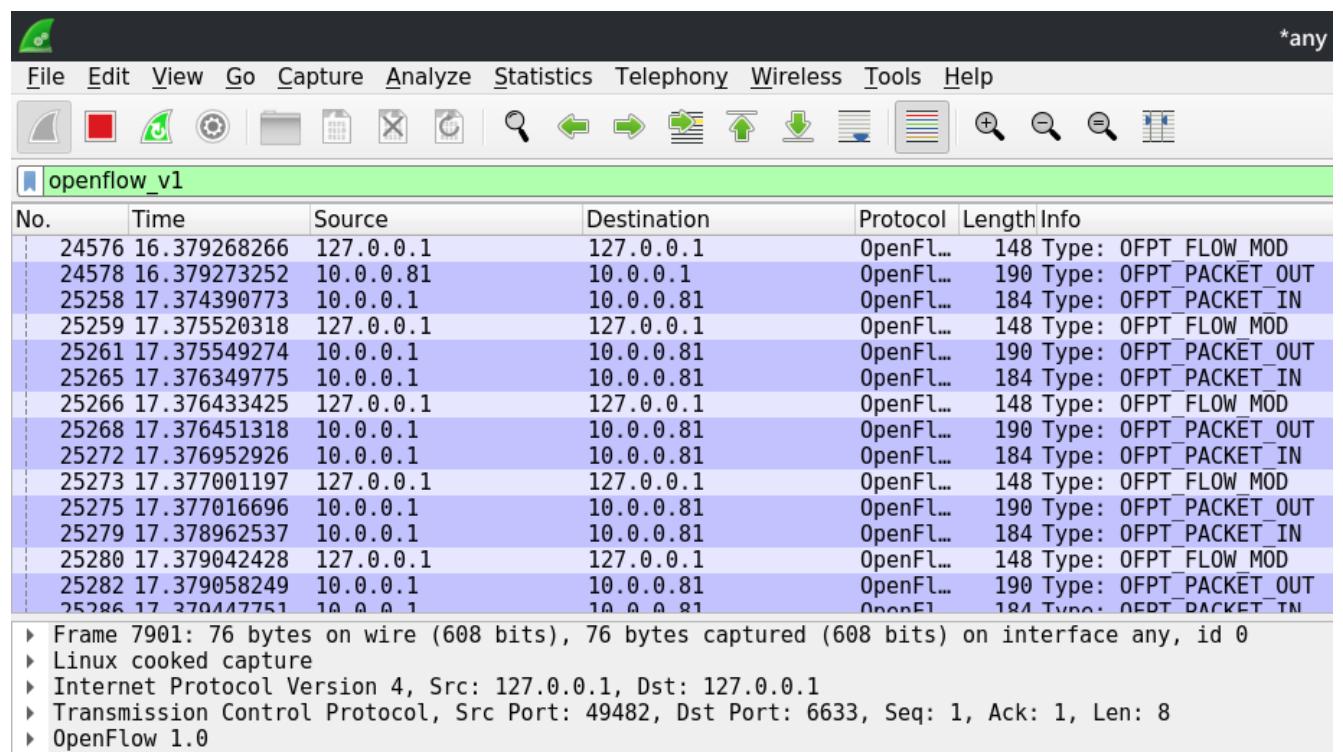
NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exiting will prevent MiniEdit from quitting and will prevent you from saving your session.

*** 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>
```

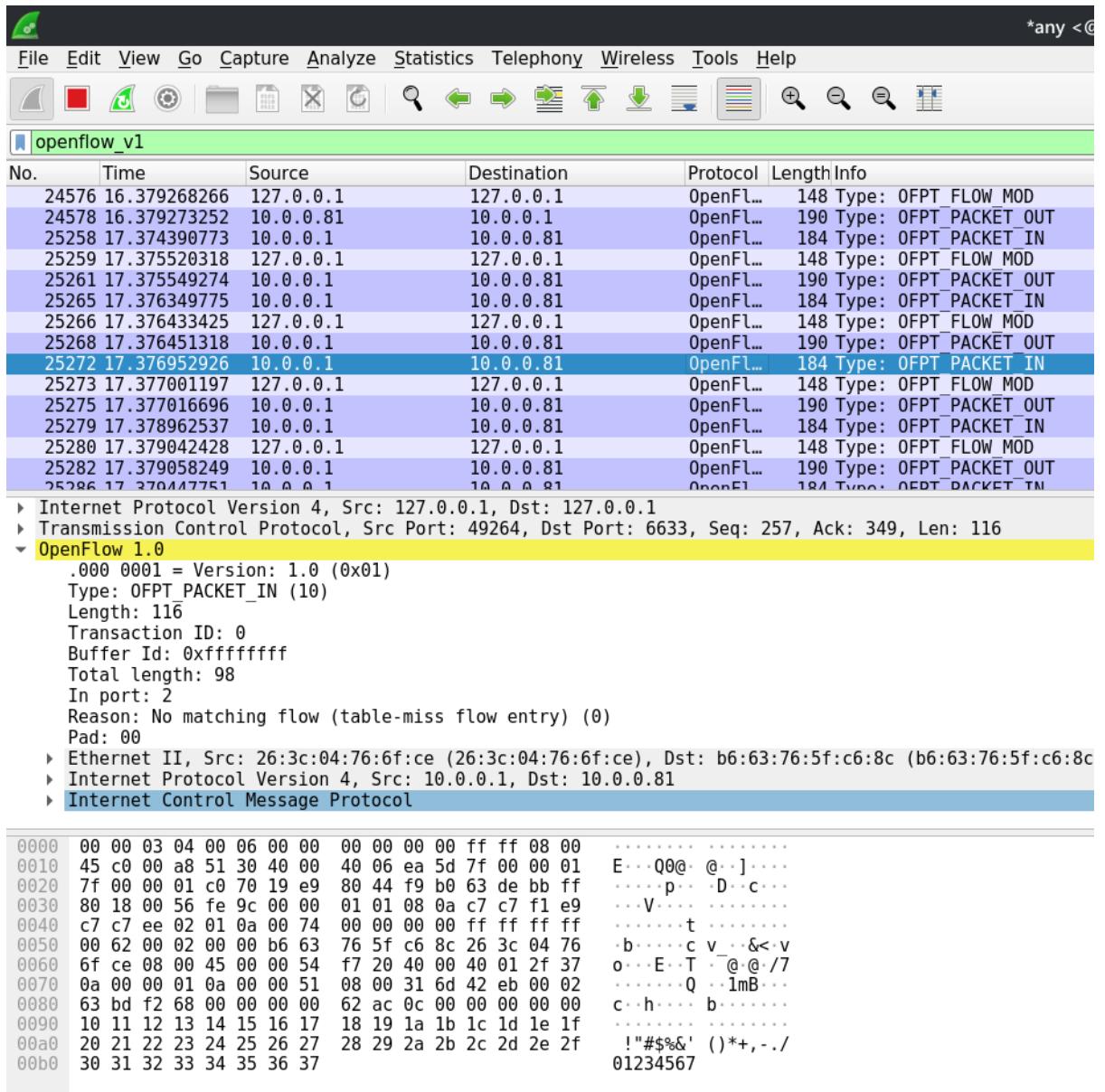
В окремому підключені до віртуальної машини запустимо Wireshark:

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

Всі пакети відображаються – мережа працює.



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



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 }
```

```

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         for i in range(branching):

```

```

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]

```



```
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
```