

Implementacja **routingu dynamicznego** opartego na cyklicznym **miarze zajętości sieci** (algorytm Dijkstry)

SSP 2022, grupa 4:

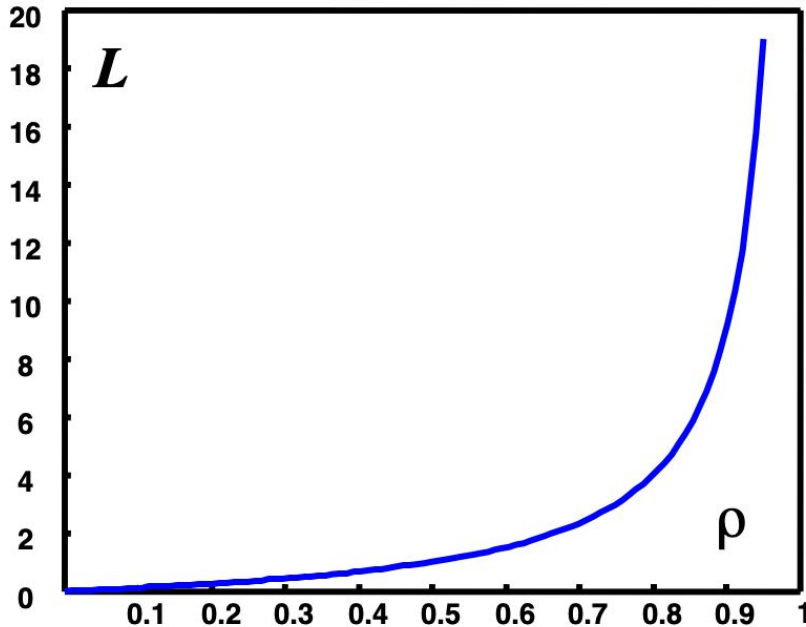
- ★ Brytańczyk Aleksandra,
- ★ Kantor Sylwia,
- ★ Kłucjasz Anna,
- ★ Kokoszka Artur

TEORIA



KOLEJKA M/M/1 – średnia zajętość

$$L = \sum_{j=1}^{\infty} j p_j = (1 - \rho) \sum_{j=1}^{\infty} j \rho^j = \frac{\rho}{1 - \rho} = \frac{\lambda / \mu}{1 - \lambda / \mu}$$



Opóźnienia w sieci, a jej zajętość

This is depicted through Figure 4.4. Note that if in the above definition, the strong inequality holds (for $0 < \alpha < 1$), then the function is called *strictly convex*.


Typically, convex cost functions appear in communications network applications to describe delay. For example, link e of a packet network can be modeled as a $M/M/1$ queue (for example, see [BG92] and the discussion in Section 1.3.1), and the average delay experienced by the packets sent along the link is expressed by a convex function of the link load \underline{y}_e given in pps (packets per second), as it goes from 0 to c_e (the given link capacity in pps). The delay function is as follows:

$$F_e(\underline{y}_e) = \frac{1}{c_e - \underline{y}_e}, \quad 0 \leq \underline{y}_e < c_e. \quad (4.3.5)$$

We now show an extension of A/PAP (4.1.7) by introducing a non-linear (convex) cost function:

TOPOLOGIA



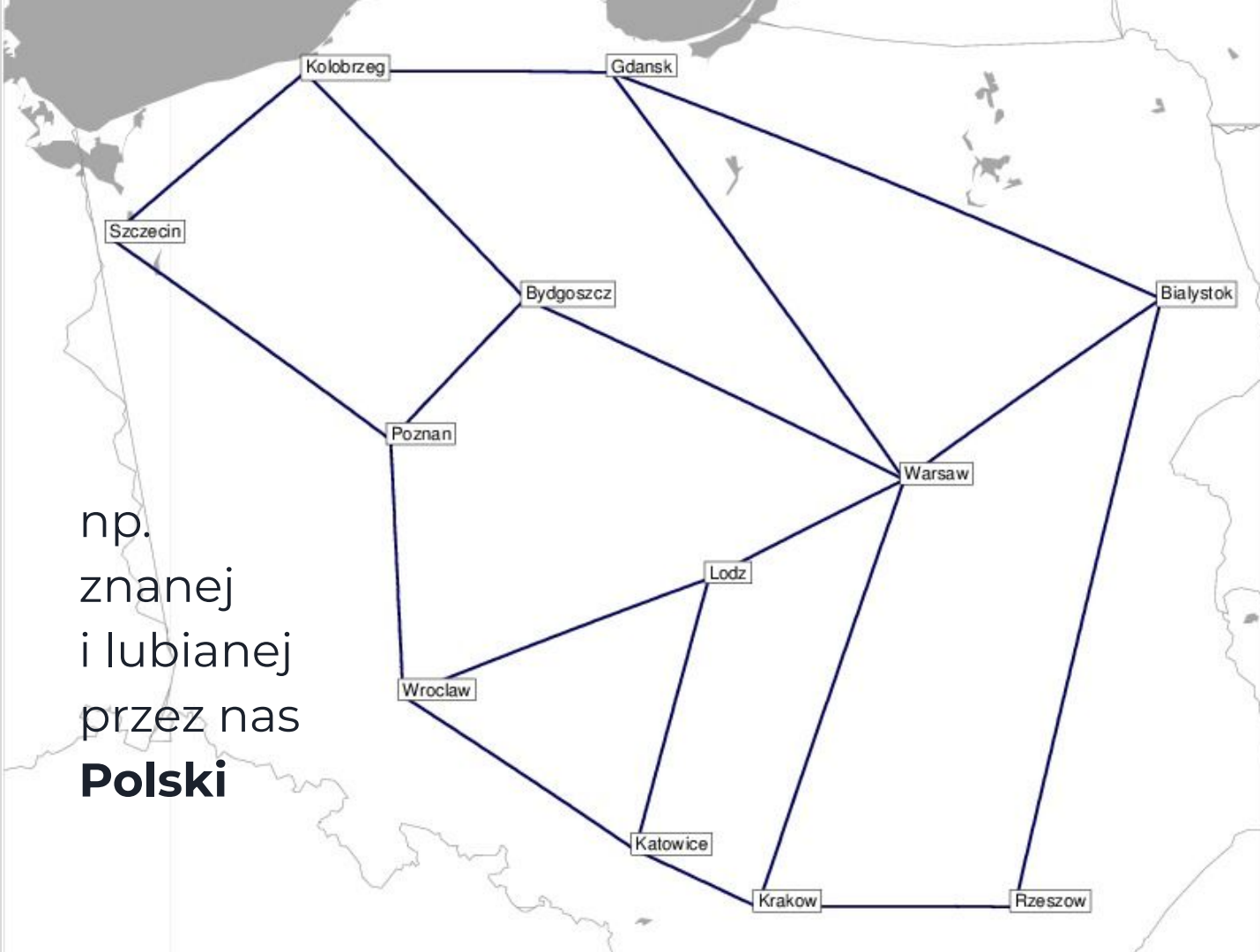


Aby pokazać funkcjonalność naszego skryptu
potrzebujemy:

- topologii typu **mesh**
- w formacie **.gml**

```
14 class MyTopoFromGML( Topo ):
15
16     def build( self ):
17         "Load topo form .gml file."
18
19         GRAPH = nx.read_gml('topos/topo-polska.gml') #works also for other .gml files, eg. janos
20
21         node_names = list(GRAPH.nodes) #helper for list()
22         numbers = [i+1 for i in range (len(node_names))] #helper for dictionary creating
23
24         NODES = dict(zip(node_names, numbers))
25
26         node_names.sort()
27
28         for i in range (len(GRAPH.nodes)):
29             self.addSwitch(f's{i+1}') #has to be number, not city name
30             self.addHost(f'{ node_names[i] }')
31             self.addLink(f'{ node_names[i] }', f's{i+1}')
32
33         for(n1, n2) in GRAPH.edges:
34             self.addLink(f's{ NODES[n1] }', f's{ NODES[n2] }')
35
```

np.
znanej
i lubianej
przez nas
Polski



REALIZACJA





Wykrywanie topologii (`event.EventSwitchEnter`)

- na podstawie wykrytych urządzeń i połączeń tworzymy obiekt typu NetworkX Graph
- każdemu połączeniu przypisujemy atrybuty:
 - port źródłowy,
 - waga,
 - przesłane bajty

Sterownik zna naszą topologię oraz jej parametry.

1



Dodawanie przepływów niezbędnych do prawidłowego działania sieci (`event.EventSwitchEnter`)

- obsługa ARP, umożliwienie hostom dodawania nowych wpisów
- dodanie logicznych połączeń host -> switch

2



Wysyłanie requesta o statystyki sieci (every INTERVAL)

- dla każdego switcha konstruujemy i wysyłamy wiadomość openflow typu OFPFlowStatsRequest

```
req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
datapath.send_msg(req)
```

Wypisywanie statystyk (ofp_event.EventOFPFlowStatsReply)

- dekompozycja wiadomości typu ...reply

```
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def _port_stats_reply_handler(self, ev):

    body = ev.msg.body

    for stat in sorted(body, key=attrgetter('port_no')):

        if (ev.msg.datapath.id, stat.port_no) in self.interfaces.keys():
            current = stat.tx_bytes
            source = ev.msg.datapath.id
```

3



Magia nie magia

1. Z uzyskanych w kroku 3. danych otrzymujemy **ile bajtów zostało przesłane przez dany port.**
2. Liczymy średnią zajętość podczas ostatniego okresu obserwacji.
3. Dzielimy przez maksymalną przepustowość danego łącza. Uzyskujemy **średnią procentową wartość zajętości.**
4. Każdemu łączu przypisujemy wagę, zgodnie z funkcją wykładniczą:

$$\text{nowa_waga} = \text{procent}^2$$

100% odpowiada wadze 10 000

50% odpowiada więc 2 500

0% odpowiada wadze 0

4

Obliczanie najkrótszej ścieżki (every INTERVAL)

- przy użyciu alg. Dijkstry obliczamy drzewo najkrótszych ścieżek między wszystkimi urządzeniami
- w efekcie sterownik otrzymuje listę nowych tras

5

```
New routing: {(6, 1): 4, (6, 4): 4, (6, 9): 4, (6, 3): 4, (6, 5): 4, (6, 8): 4, (6, 11): 4, (6, 7): 4, (6, 12): 4, (6, 10): 4, (6, 2): 4, (1, 6): 3, (1, 4): 3, (1, 9): 3, (1, 3): 3, (1, 5): 3, (1, 8): 3, (1, 11): 3, (1, 7): 3, (1, 12): 3, (1, 10): 3, (1, 2): 3, (4, 1): 2, (4, 9): 2, (4, 3): 2, (4, 8): 2, (4, 11): 2, (4, 7): 2, (4, 12): 2, (4, 10): 2, (4, 2): 2, (4, 6): 3, (4, 5): 3, (9, 6): 2, (9, 4): 2, (9, 3): 2, (9, 5): 2, (9, 8): 2, (9, 11): 2, (9, 7): 2, (9, 12): 2, (9, 10): 2, (9, 2): 2, (9, 1): 4, (3, 1): 2, (3, 9): 2, (3, 10): 2, (3, 2): 2, (3, 6): 4, (3, 4): 4, (3, 5): 4, (3, 8): 4, (3, 11): 4, (3, 7): 4, (3, 12): 4, (5, 6): 3, (5, 1): 3, (5, 4): 3, (5, 9): 3, (5, 3): 3, (5, 8): 3, (5, 11): 3, (5, 7): 3, (5, 12): 3, (5, 10): 3, (5, 2): 3, (8, 6): 2, (8, 1): 2, (8, 4): 2, (8, 9): 2, (8, 3): 2, (8, 5): 2, (8, 11): 2, (8, 7): 2, (8, 12): 2, (8, 10): 2, (8, 2): 2, (11, 6): 5, (11, 4): 5, (11, 5): 5, (11, 8): 7, (11, 7): 7, (11, 12): 7, (11, 1): 4, (11, 9): 4, (11, 3): 4, (11, 10): 4, (11, 2): 4, (7, 6): 4, (7, 1): 4, (7, 4): 4, (7, 9): 4, (7, 3): 4, (7, 5): 4, (7, 8): 4, (7, 11): 4, (7, 12): 4, (7, 10): 4, (7, 2): 4, (12, 6): 2, (12, 1): 2, (12, 4): 2, (12, 9): 2, (12, 3): 2, (12, 5): 2, (12, 11): 2, (12, 10): 2, (12, 2): 2, (12, 8): 4, (12, 7): 4, (10, 1): 2, (10, 9): 2, (10, 6): 3, (10, 4): 3, (10, 3): 3, (10, 5): 3, (10, 8): 3, (10, 11): 3, (10, 7): 3, (10, 12): 3, (10, 2): 3, (2, 6): 4, (2, 1): 4, (2, 4): 4, (2, 9): 4, (2, 3): 4, (2, 5): 4, (2, 8): 4, (2, 11): 4, (2, 7): 4, (2, 12): 4, (2, 10): 4}
```

Instalowanie nowych przepływów w sieci (every INTERVAL)

- jeśli nastąpi dopasowanie ip_destination, wyślij wiadomość na odpowiedni out_port
- zmieniamy również routing dla eth_type ARP

```
match = parser.OFPMatch(ipv4_dst=ip_dst, eth_type=0x800)
actions = [parser.OFPACTIONOutput(port=out_port)]
self.add_flow(dp_value, 10, match, actions)
```

```
mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                        match=match, instructions=inst, hard_timeout=hard_timeout)
datapath.send_msg(mod)
```


6



NEXT POLLING INTERVAL


GENEROWANIE RUCHU





Aby zobrazować i przetestować działanie korzystamy z generowania przepływów testowych (**iperf3**):

- generujemy długi przepływ pomiędzy Krakowem, a Gdańskiem (10.0.0.6 a 10.0.0.3)
- generujemy ruchy “przeszkadzające” pomiędzy pozostałymi hostami
- w międzyczasie trasa zostaje zmieniona na mniej zajętą
- generujemy drugi przepływ pomiędzy Krakowem, a Gdańskiem (10.0.0.6 a 10.0.0.3) i.....



```
#!/bin/bash
```

```
iperf -s -u &
```

```
ips=($1)
```

```
len=$( expr ${#ips[@]} - 1 )
```

```
id=$2
```

```
while :
```

```
do
```

```
    # Krakow do Gdanska reszta background ruch
```

```
    if [ "${2}" == "10.0.0.6" ]
```

```
    then
```

```
        echo "${2} generate traffic to 10.0.0.3 with 900 Kbits/s" >> results/results${2}.txt
```

```
        iperf -c 10.0.0.3 -u -t 20 -b 800Kbits >> results/results${2}.txt
```

```
        waiting=$(shuf -i 1-2 -n 1)
```

```
        echo "No traffic for ${waiting} seconds" >> results/results${2}.txt
```

```
        sleep ${waiting}
```

```
    elif [ "${2}" == "10.0.0.3" ]
```

```
    then
```

```
        echo "No traffic for ${waiting} seconds" >> results/results${2}.txt
```

```
        sleep ${waiting}
```

```
    else
```

```
        ruch=$(shuf -i 100-300 -n 1)
```

```
        echo "${2} generate traffic to ${ips[$i]} with ${ruch} Kbits/s" >> results/results${2}.txt
```

```
        i=$(shuf -i 0-${len} -n 1)
```

```
        iperf -c $(echo "${ips[$i]}") -u -t 20 -b $(echo "${ruch}")Kbits >> results/results${2}.txt
```

```
        waiting=$(shuf -i 1-5 -n 1)
```

```
        echo "No traffic for ${waiting} seconds" >> results/results${2}.txt
```

```
        sleep ${waiting}
```

```
    fi
```

```
done
```

New routing for 6_Krakow 3_Gdansk: 6_Krakow - 5_Rzeszow - 4_Bialystok - 3_Gdansk

Wypisywanie statystyk dla 6:Krakow

datapath	eth_type	ipv4_dst	out-port	packets	bytes	speed
0000000000000006	800	10.0.0.3	2	14374	21733488	55263
0000000000000006	800	10.0.0.6	1	0	0	0

Wypisywanie statystyk dla 5:Rzeszow

datapath	eth_type	ipv4_dst	out-port	packets	bytes	speed
0000000000000005	800	10.0.0.3	2	731	1105272	55263
0000000000000005	800	10.0.0.6	2	0	0	0

Wypisywanie statystyk dla 9:Poznan

datapath	eth_type	ipv4_dst	out-port	packets	bytes	speed
0000000000000009	800	10.0.0.3	3	1659	2508408	0
0000000000000009	800	10.0.0.6	3	0	0	0

Wypisywanie statystyk dla 3:Gdansk

datapath	eth_type	ipv4_dst	out-port	packets	bytes	speed
0000000000000003	800	10.0.0.3	1	14476	21887712	54810
0000000000000003	800	10.0.0.6	4	0	0	0

Wypisywanie statystyk dla 8:Wroclaw

datapath	eth_type	ipv4_dst	out-port	packets	bytes	speed
0000000000000008	800	10.0.0.3	4	890	1345680	0
0000000000000008	800	10.0.0.6	4	0	0	0

Wypisywanie statystyk dla 4:Bialystok

datapath	eth_type	ipv4_dst	out-port	packets	bytes	speed
0000000000000004	800	10.0.0.3	2	7764	11739168	53978
0000000000000004	800	10.0.0.6	2	0	0	0

Wypisywanie statystyk dla 12:Lodz

datapath	eth_type	ipv4_dst	out-port	packets	bytes	speed
000000000000000c	800	10.0.0.3	3	2	3024	0
000000000000000c	800	10.0.0.6	3	0	0	0

Wypisywanie statystyk dla 1:Szczecin

datapath	eth_type	ipv4_dst	out-port	packets	bytes	speed
0000000000000001	800	10.0.0.3	2	388	586656	0
0000000000000001	800	10.0.0.6	2	0	0	0

Wypisywanie statystyk dla 10:Bydgoszcz

datapath	eth_type	ipv4_dst	out-port	packets	bytes	speed
000000000000000a	800	10.0.0.3	4	844	1276128	0
000000000000000a	800	10.0.0.6	4	0	0	0

Wypisywanie statystyk dla 11:Warszawa

datapath	eth_type	ipv4_dst	out-port	packets	bytes	speed
000000000000000b	800	10.0.0.3	5	1551	2345112	0
000000000000000b	800	10.0.0.6	6	0	0	0

Wypisywanie statystyk dla 7:Katowice

datapath	eth_type	ipv4_dst	out-port	packets	bytes	speed
0000000000000007	800	10.0.0.3	2	892	1348704	0
0000000000000007	800	10.0.0.6	2	0	0	0

Wypisywanie statystyk dla 2:Kolobrzeg

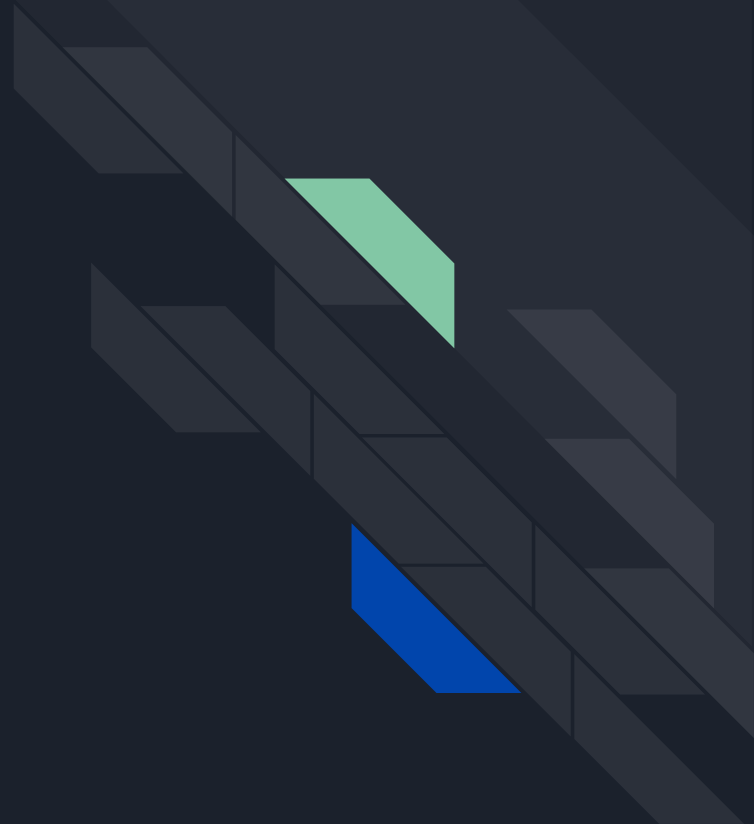
datapath	eth_type	ipv4_dst	out-port	packets	bytes	speed
0000000000000002	800	10.0.0.3	3	840	1270080	0
0000000000000002	800	10.0.0.6	3	0	0	0

DEMO



GITHUB PROJECT

klik



Dzięki i cześć!

