Politechnika Rzeszowska
im. Ignacego Łukasiewicza
Wydział Elektrotechniki i Informatyki
Zakład systemów złożonych

Badanie obciążenia połączeń węzłów sieci SDN w środowisku mininet

Opracowanie:
Jakub Przystasz
Mateusz Rejman
Miłosz Serafin

Rzeszów 2021

1 Wstęp

Celem badania było sprawdzenie wpływu algorytmu optymalizacji przepływów w sieci SDN na obciążenie połączeń węzłów sieci. Wykorzystany algorytm optymalizacji to *Genetic Algorithm based Load Balancing*. Wynikiem działania algorytmu powinno być zminimalizowanie maksymalnego obciążenia połączeń.

Badanie przeprowadzono za pomocą oprogramowania emulującego sieci wirtualne *mininet* oraz pakietu *iperf*.

2 Praca w środowisku mininet

Pakiet mininet dostępny jest w repozytoriach większości popularnych dystrybucji Linuxa. Do uruchomienia wymagany jest interpreter języka Python w wersji 2 lub 3 (wsparcie dla obu wersji interpretera jest dostępne od wydania *mininet-2.3*). Systemem operacyjnym, na którym wykonywano eksperymenty jest Debian 10. W repozytorium systemu dostępna jest wesja 2.2.2 - taką też zainstalowano.

Mininet jest oprogramowaniem, którego obsługa odbywa się za pośrednictwem linii poleceń, lub poprzez API. Do uruchomienia emulatora wymagane jest posiadanie uprawnień root.

Uruchomienie mininet w podstawowej konfiguracj poleceniem:

```
sudo mn
```

zalecane jest, aby przed uruchomieniem emulatora wyczyścić ustawienia sieci wydając polecenie:

```
sudo mn -c
```

Jeżeli proces tworzenia środowiska przebiegnie pomyślnie, system na którym uruchomiono emulator będzie posiadać nowe interfejsy sieciowe, które są emulowane przez mininet. Możliwe jest korzystanie z oprogramowania zainstalowanego na komputerze gospodarza jako hosty utworzone w mininet. Aby wykonać program jako host wystarczy podać w linii poleceń mininet:

```
[nazwa hosta] [polecenie]
```

przykładowo: *h1 ping -c 16 h2*

Znacznie wygodniejszym rozwiązaniem jest otworzenie osobnego okna terminala, z którego można wydawać polecenia jako konkretny host symulowanej sieci, umożliwia to polecenie:

```
xterm [nazwa hosta]
```

3 Definiowanie topologii sieci

Mininet pozwala na definiowanie własnej topologii sieci za pośrednictwem skryptów w języku Python, listing 1 zawiera kod definiujący topologię badanej sieci.

```
from mininet.topo import Topo

class MyTopo(Topo):
    def build(self):


        h0 = self.addHost('h0')
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')
        h5 = self.addHost('h5')
        h6 = self.addHost('h6')
        h7 = self.addHost('h7')
        h8 = self.addHost('h8')
        h9 = self.addHost('h9')


        s0 = self.addSwitch('s0')
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        s5 = self.addSwitch('s5')
        s6 = self.addSwitch('s6')
        s7 = self.addSwitch('s7')
        s8 = self.addSwitch('s8')
        s9 = self.addSwitch('s9')


        self.addLink(s0, h0)
        self.addLink(s1, h1)
        self.addLink(s2, h2)
        self.addLink(s3, h3)
        self.addLink(s4, h4)
        self.addLink(s5, h5)
        self.addLink(s6, h6)
        self.addLink(s7, h7)
        self.addLink(s8, h8)
        self.addLink(s9, h9)


        self.addLink(s9, s5)
```

```
        self.addLink(s5, s6)
        self.addLink(s7, s3)
        self.addLink(s6, s5)
        self.addLink(s5, s3)
        self.addLink(s3, s1)
        self.addLink(s1, s0)
        self.addLink(s0, s7)
        self.addLink(s0, s9)
        self.addLink(s5, s8)
        self.addLink(s8, s9)
        self.addLink(s9, s1)
        self.addLink(s4, s5)
        self.addLink(s1, s2)
        self.addLink(s2, s3)
        self.addLink(s7, s5)
        self.addLink(s5, s7)

topos = {'mytopo':(lambda:MyTopo())}
```

Listing 1: Plik *topo.py* definiujący topologię badanej sieci

Po zdefiniowaniu topologii sieci utworzono skrypt powłoki [2] który uruchamia mininet ze zdefiniowaną wcześniej siecią.

```
#!/bin/bash
sudo mn -c
sudo mn --custom ./topo.py --topo mytopo --mac --arp --controller
    none --switch ovs,protocol=OpenFlow13
sudo mn -c
```

Listing 2: Skrypt uruchamiający mininet

4 Definiowanie przepływów w sieci

Aby ułatwić wprowadzanie przedstawionych przepływów w sieci stworzono skrypt w języku Python[4] generujący polecenia środowiska mininet.

```
import sys
import copy

class Node:
    def __init__(self):
        self.interfaces = dict()

    def add_int(self, dst, port):
        self.interfaces[dst] = port

    def __repr__(self):
        return f'{self.interfaces}\n'
```

```
13
14
15  class Edge:
16      def __init__(self, node1, node2):
17          self.node1 = node1
18          self.node2 = node2
19
20
21  class Path:
22      def __init__(self, src, dst, edges):
23          self.edges = edges
24          self.src = src
25          self.dst = dst
26
27      def create_flows(self, nodes):
28          for edge in self.edges:
29              print(f"ovs-ofctl add-flow s{edge.node1} -O OpenFlow13
        "
30                    f"dl_src=00:00:00:00:00:0{(int(self.src + 1)):x
    },"
31                    f"dl_dst=00:00:00:00:00:0{(int(self.dst + 1)):x
    },"
32                    f"actions=output:{nodes[f's{edge.node1}'].
    interfaces[f's{edge.node2}']}")
33
34
35      def __str__(self):
36          return f'path{self.src}{self.dst}'
37
38      def create_reverse(self):
39          edges = self.edges.copy().reverse()
40          for edge,i in enumerate(edges):
41              self.node1,self.node2 = self.node2,self.node1
42          return Path(self.dst, self.src,edges)
43
44
45  def LinkToHosts(nodes):
46      for key in nodes.keys():
47          if key[0] == 's':
48              for key1 in nodes[key].interfaces:
49                  if key1[0] == 'h':
50                      print(f"ovs-ofctl add-flow {key} -O OpenFlow13
        "
51                            f"dl_dst=00:00:00:00:00:0{(int(key1[1:]) +
    1):x},"
52                            f"actions=output:{nodes[key].interfaces[
    key1]}")
53
54  if __name__ == "__main__":
```

```
    nodes = dict()

    with open("links.txt") as file:
        for line in file.readlines():
            line = line.rstrip()
            line = line.replace('(OK OK)', '')
            str_node = line.split('<->')
            for i, node in enumerate(str_node):
                str_node[i] = node.split('-')

            src = str_node[0]
            dst = str_node[1]

            if src[0] not in nodes.keys():
                nodes[src[0]] = Node()

            nodes[src[0]].add_int(dst[0], src[1][3:])

            if dst[0] not in nodes.keys():
                nodes[dst[0]] = Node()

            nodes[dst[0]].add_int(src[0], dst[1][3:])

            src = str_node[0]
            dst = str_node[1]

            if src[0] not in nodes.keys():
                nodes[src[0]] = Node()

            nodes[src[0]].add_int(dst[0], src[1][3:])

            if dst[0] not in nodes.keys():
                nodes[dst[0]] = Node()

            nodes[dst[0]].add_int(src[0], dst[1][3:])

    paths = [
        Path(9, 6, [Edge(9, 5), Edge(5, 6)]),

        Path(7, 3, [Edge(7, 3)]),

        Path(6, 0, [Edge(6, 5), Edge(5, 4), Edge(4, 0)]),
        Path(0, 6, [Edge(0, 4), Edge(4, 5), Edge(5, 6)]),

        Path(3, 7, [Edge(3, 1), Edge(1, 9), Edge(9, 7)]),

        Path(1, 8, [Edge(1, 9), Edge(9, 7), Edge(7, 8)]),
        Path(8, 1, [Edge(8, 7), Edge(7, 9), Edge(9, 1)]),

```

```
104         Path(6, 9, [Edge(6, 9)]),
105
106         Path(8, 5, [Edge(8, 5)]),
107         Path(5, 8, [Edge(5, 8)]),
108
109         Path(9, 1, [Edge(9, 1)]),
110         Path(1, 9, [Edge(1, 9)]),
111
112         Path(3, 0, [Edge(3, 1), Edge(1, 0)]),
113         Path(0, 3, [Edge(0, 1), Edge(1, 3)]),
114
115         Path(4, 3, [Edge(4, 5), Edge(5, 3)]),
116         Path(3, 4, [Edge(3, 5), Edge(5, 4)]),
117
118         Path(7, 2, [Edge(7, 6), Edge(6, 2)]),
119         Path(2, 7, [Edge(2, 6), Edge(6, 7)]),
120
121         Path(0, 8, [Edge(0, 5), Edge(5, 8)]),
122         Path(8, 0, [Edge(8, 5), Edge(5, 0)]),
123
124         Path(6, 1, [Edge(6, 9), Edge(9, 1)]),
125         Path(1, 6, [Edge(1, 9), Edge(9, 6)]),
126
127         Path(2, 1, [Edge(2, 3), Edge(3, 1)]),
128         Path(1, 2, [Edge(1, 3), Edge(3, 2)]),
129
130         Path(5, 0, [Edge(5, 4), Edge(4, 0)]),
131         Path(0, 5, [Edge(0, 4), Edge(4, 5)]),
132
133         Path(7, 2, [Edge(7, 6), Edge(6, 2)]),
134         Path(2, 7, [Edge(2, 6), Edge(6, 7)]),
135
136         Path(6, 5, [Edge(6, 5)]),
137         Path(5, 6, [Edge(5, 6)]),
138
139         Path(7, 5, [Edge(7, 5)]),
140
141         Path(5, 7, [Edge(5, 7)])
142     ]
143
144     for path in paths:
145         path.create_flows(nodes)
146
147     LinkToHosts(nodes)
```

Listing 3: Skrypt budujący polecenia mininet

W liniach [91, 142] listingu 4 definiowane są poszczególne przepływy, przykładowo:

```
Path(5, 7, [Edge(5, 7)])
```

jest tłumaczone na polecenie mininet:

```
ovs-ofctl add-flow s5 -O OpenFlow13 dl_src=00:00:00:00:00:06,
    dl_dst=00:00:00:00:00:07,actions=output:4
```

Do działania skryptu wymagany jest plik tekstowy zawierający wynik polecenia w mininet: *links*

```
h0-eth0<->s0-eth1 (OK OK)
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s2-eth1 (OK OK)
h3-eth0<->s3-eth1 (OK OK)
h4-eth0<->s4-eth1 (OK OK)
h5-eth0<->s5-eth1 (OK OK)
h6-eth0<->s6-eth1 (OK OK)
h7-eth0<->s7-eth1 (OK OK)
h8-eth0<->s8-eth1 (OK OK)
h9-eth0<->s9-eth1 (OK OK)
s0-eth4<->s5-eth9 (OK OK)
s1-eth5<->s0-eth3 (OK OK)
s1-eth3<->s9-eth3 (OK OK)
s2-eth3<->s3-eth5 (OK OK)
s3-eth3<->s1-eth2 (OK OK)
s4-eth3<->s0-eth2 (OK OK)
s4-eth5<->s0-eth5 (OK OK)
s4-eth4<->s5-eth7 (OK OK)
s5-eth8<->s3-eth4 (OK OK)
s5-eth5<->s4-eth2 (OK OK)
s5-eth3<->s6-eth2 (OK OK)
s5-eth12<->s7-eth7 (OK OK)
s5-eth10<->s8-eth4 (OK OK)
s6-eth6<->s2-eth2 (OK OK)
s6-eth3<->s5-eth4 (OK OK)
s6-eth4<->s9-eth5 (OK OK)
s7-eth2<->s3-eth2 (OK OK)
s7-eth6<->s5-eth11 (OK OK)
s7-eth5<->s6-eth5 (OK OK)
s7-eth4<->s8-eth2 (OK OK)
s8-eth3<->s5-eth6 (OK OK)
s9-eth6<->s1-eth4 (OK OK)
s9-eth2<->s5-eth2 (OK OK)
s9-eth4<->s7-eth3 (OK OK)
```

Listing 4: Przykładowy wynik polecenia *links*

5 Testowanie sieci - iperf

Na systemie hosta zainstalowano pakiet iperf - również dostępny w repozytorium. Następnie w środowisku mininet dla każdego hosta wywołano polecenie uruchamiające serwer iperf:

```
mininet> [nazwa hosta] iperf -s -p 5566 &
```

Po uruchomieniu serwerów iperf na każdym hoscie w badanej sieci można wykonać eksperyment mający na celu sprawdzenie obciążenia poszczególnych połączeń pomiędzy węzłami. Dla każdego ze zdefiniowanych flow uruchomiono proces iperf symulujący ruch sieciowy pomiędzy węzłami zdefiniowanymi we flow. W tym celu napisano skrypt powłoki który jako argument przyjmuje numer hosta w sieci mininet:

```
#!/bin/bash

ip=$(expr $1 + 1)
iperf -c 10.0.0."$ip" -p 5566 -t 3600
```

Listing 5: Skrypt uruchamiający proces iperf

Każdy z uruchomionych procesów pracuje przez okres 15 minut. W tym czasie należy zbadać ruch na poszczególnych interfejsach węzłów. Można to zrobić za pomocą np. Wireshark, ntop.

Najprostszą metodą na sprawdzenie obciązania danego interfejsu jest odczytanie wartości zapisanych w katalogu (ścieżka dla systemu Debian)

```
/sys/class/net/[nazwa interfejsu]/statistics/rx_bytes
oraz
/sys/class/net/[nazwa interfejsu]/statistics/tx_bytes
```

W powyższych lokalizacjach przechowywane są pliki tekstowe zawierające wartości liczbowe odebranych/przesłanych bajtów informacji na danym interfejsie od uruchomienia systemu.

Listing 6 zawiera skrypt korzystający z powyższej metody odczytywania informacji o interfejsie.

```
#!/bin/bash

# how long script 'll be executed
duration=1800

start=$(date +%s)
start=$(expr $start + $duration)

```

```
if [ -z "$1" ]; then
        echo
        echo usage: $0 network-interface
        echo
        echo e.g. $0 eth0
        echo
        exit
fi

IF=$1

while true
do
        now=$(date +%s)
        if   [ "$now" -eq "$start" ]; then
                break
        fi

        R1=`cat /sys/class/net/$1/statistics/rx_bytes`
        T1=`cat /sys/class/net/$1/statistics/tx_bytes`
        sleep 1
        R2=`cat /sys/class/net/$1/statistics/rx_bytes`
        T2=`cat /sys/class/net/$1/statistics/tx_bytes`
        TBPS=`expr $T2 - $T1`
        RBPS=`expr $R2 - $R1`
        TKBPS=`expr $TBPS / 1024`
        RKBPS=`expr $RBPS / 1024`
        echo "$now $1 $TKBPS $RKBPS $2"
done
```

Listing 6: Skrypt raportujący nateżenie ruchu na interfejsie źródło:https://gist.github.com/joemiller/4069513

Podczas eksperymentu uruchomio skrypt z listingu 7, który zapisuje w jedno sekundowym interwale informacje o aktualnym przepływie na każdym z interfejsów komputera gospodarza. Dzięki temu możliwa była późniejsza analiza otrzymanych danych.

```
#!/bin/bash

interfaces=$(ls -1 /sys/class/net)

for t in ${interfaces[@]}; do
  ./measure.sh  $t "before" >> "./out/$t.dat" &
done
```

Listing 7: Skrypt zapisujący informacje na temat wszystkich interfejsów hosta

Do analizy wykorzystano skrypt w języku Python, który generuje wykresy porównujące nateżenie na wejściu/wyjściu interfejsu przed, oraz po optymalizacji

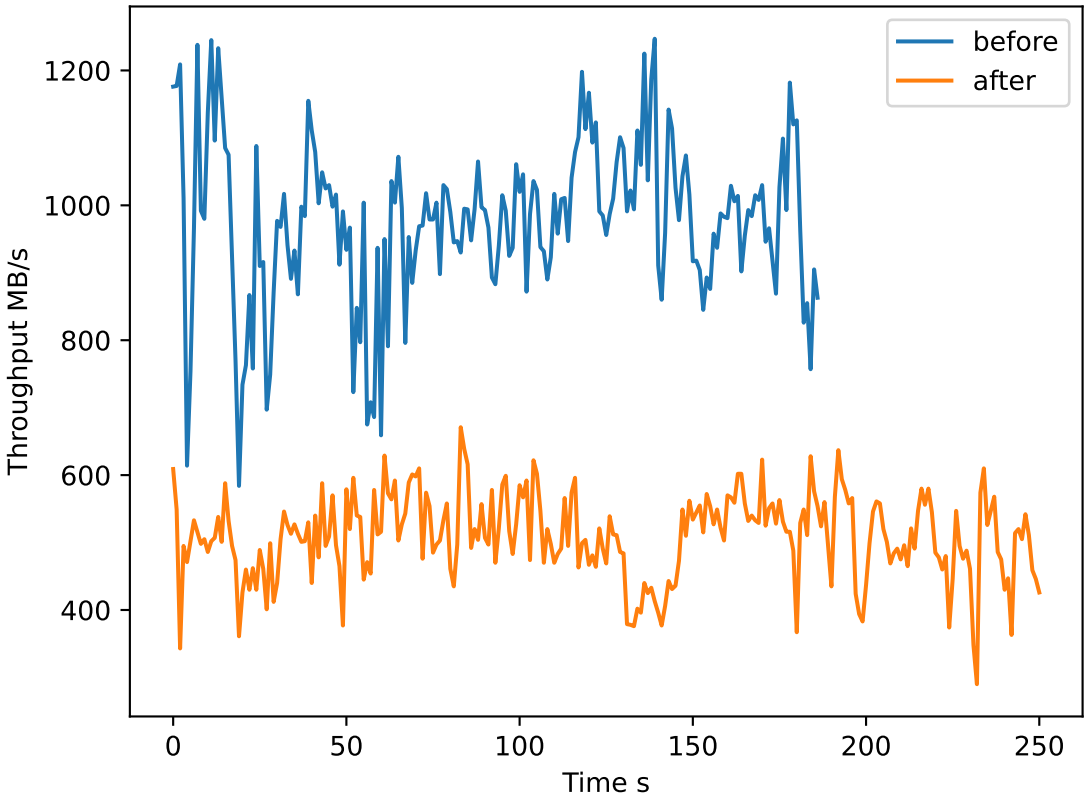
przepływów w sieci.

```
#%%
#Place that folder in directory with all .dat files
#and run: wsl ls -1 | python plot.py

import numpy as np
import math
import matplotlib.pyplot as plt
import fileinput

interfaces = dict()

for line in fileinput.input():
    line = line.rstrip()
    filename = line
    line = line.split('.')
    if len(line) > 1:
        if (line[1] == 'dat'):
            with open(filename) as file:
                i = 0
                for ln in file.readlines():
                    ln = ln.rstrip().split()

                    try:
                        ln[2] = int(int(ln[2]) / 1000)

                        ln[3] = int(int(ln[3]) / 1000)
                        ln[0] = int(ln[0])

                        if ln[1] not in interfaces:
                            interfaces[ln[1]] = dict()

                        if ln[4] not in interfaces[ln[1]]:
                            interfaces[ln[1]][ln[4]] = dict()

                        interfaces[ln[1]][ln[4]][ln[0]] = ln[2], ln[3]
                    except:
                        print(f'DUPA at: {i} {filename}')
                    i += 1


def plot_fig(x, y, if_name, type):
    plt.plot(x, y, label=dataset_name)
    plt.legend()
    plt.ylabel(f'Throughput {type} MB/s')
    plt.xlabel(f'Time')
    plt.xticks([])
```

```
47      plt.savefig(f'{if_name}_{type}.png')
48
49
50  for if_name in interfaces:
51      print(f'Plotting {if_name}')
52      interface = interfaces.get(if_name)
53      data = interface['optimal']
54
55      for dataset_name in interface:
56          data = interface.get(dataset_name)
57
58          time = list()
59          i = 0
60          rx = list()
61          tx = list()
62
63          for value in data:
64              tmp = data.get(value)
65              if tmp[0] == 1 or tmp[1] == 0:
66                  continue
67              rx.append(tmp[0])
68              tx.append(tmp[1])
69              time.append(i)
70              i += 1
71
72          time = np.array(time)
73          tx = np.array(tx)
74          rx = np.array(rx)
75
76          plot_fig(time, rx, if_name, 'rx')
77
78      plt.clf()
79
80  print('Done')
```

Listing 8: Skrpy do analizy natężenia połączeń węzłów sieci

6 Wyniki analizy

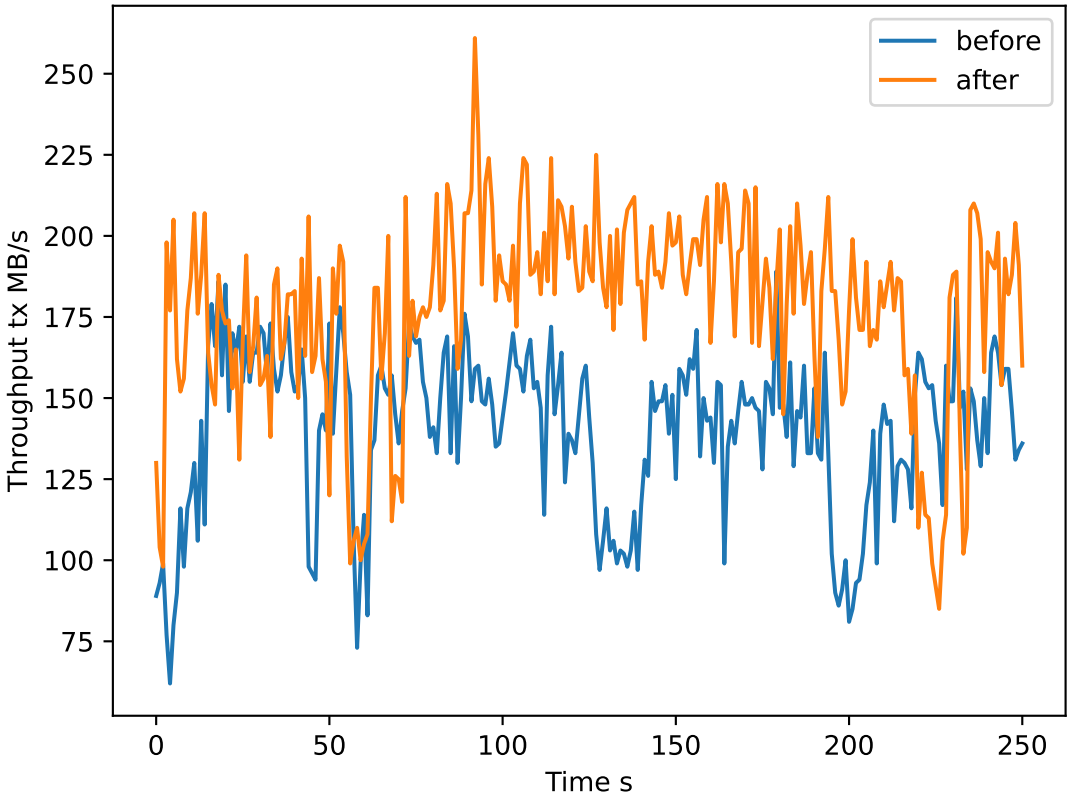


Rysunek 1: Natężenie ruchu wychodzącego z węzła numer 1 do węzła numer 3

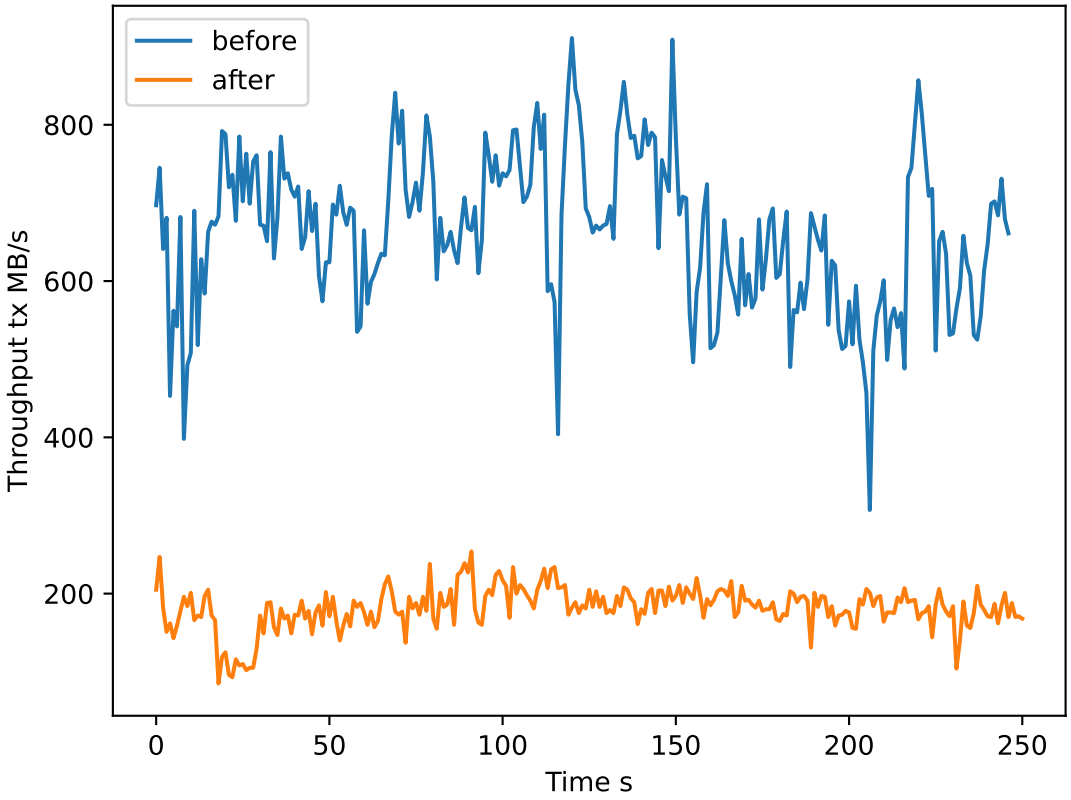
Jak pokazano na rysunku 1 zastosowanie optymalnego algorytmu pozwoliło na trzykrotne obniżenie natężenia ruchu na jednym z najbardziej obciążonych łącz w sieci.

Warto zauważyć, że większość interfejsów jest używanych jedynie po zastosowaniu optymalnej konfiguracji przepływów, co świadczy o tym, że algorytm prawidłowo przekierował ruch z najbardziej obciążonych węzłów, na te które do tej pory nie były wykorzystywane.

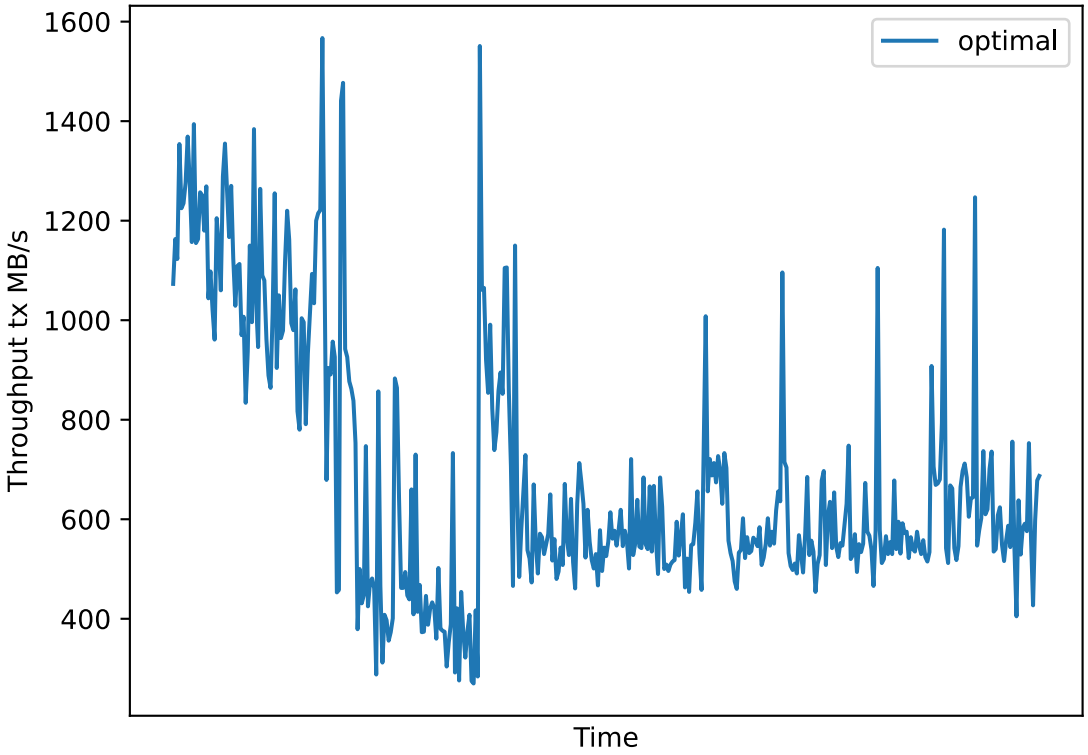
W dalszej części znajduje się zestawienie analizy dla każdego interfejsu każdego z węzłów. Do analizy wyników pomocny będzie listing 4



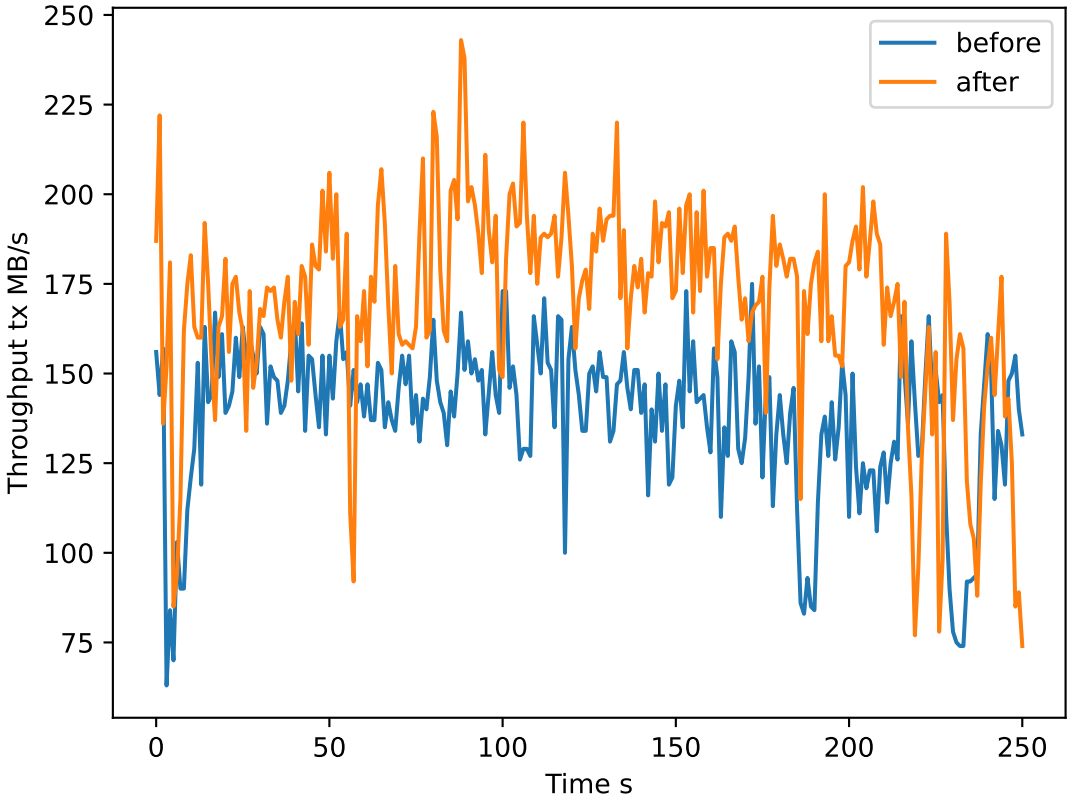
Rysunek 2: Natężenie ruchu z interfejsu eth1 w węźle s0



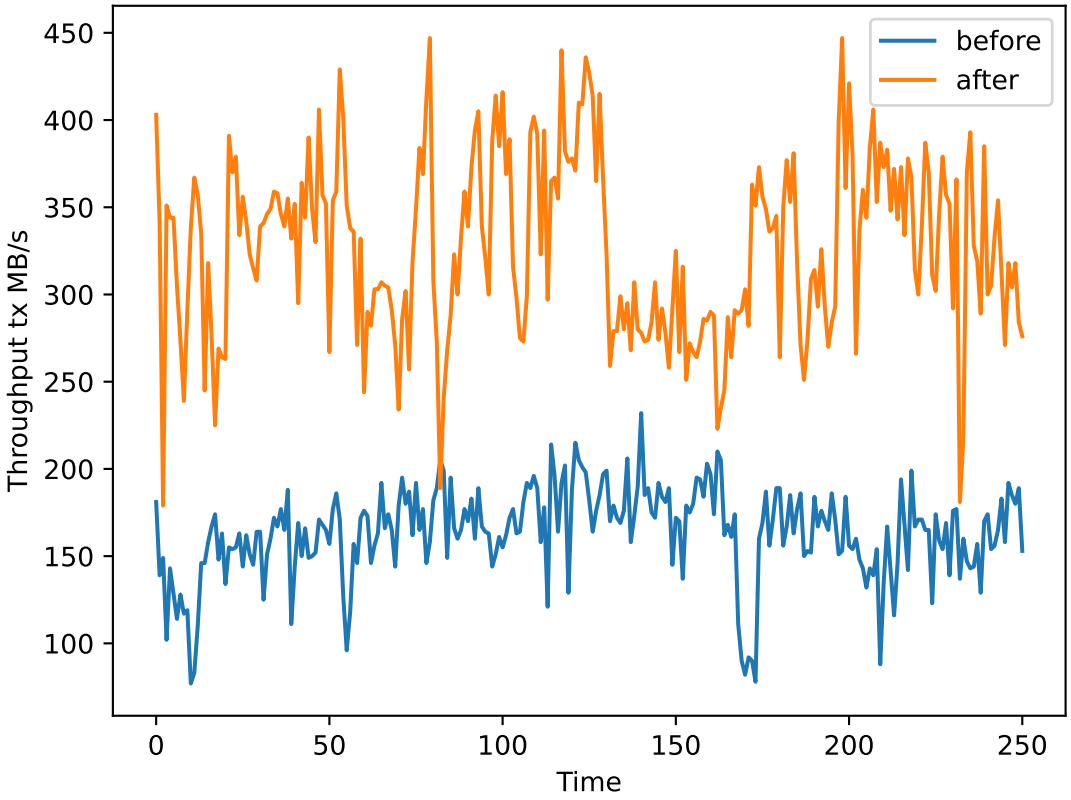
Rysunek 3: Natężenie ruchu z interfejsu eth2 w węźle s0



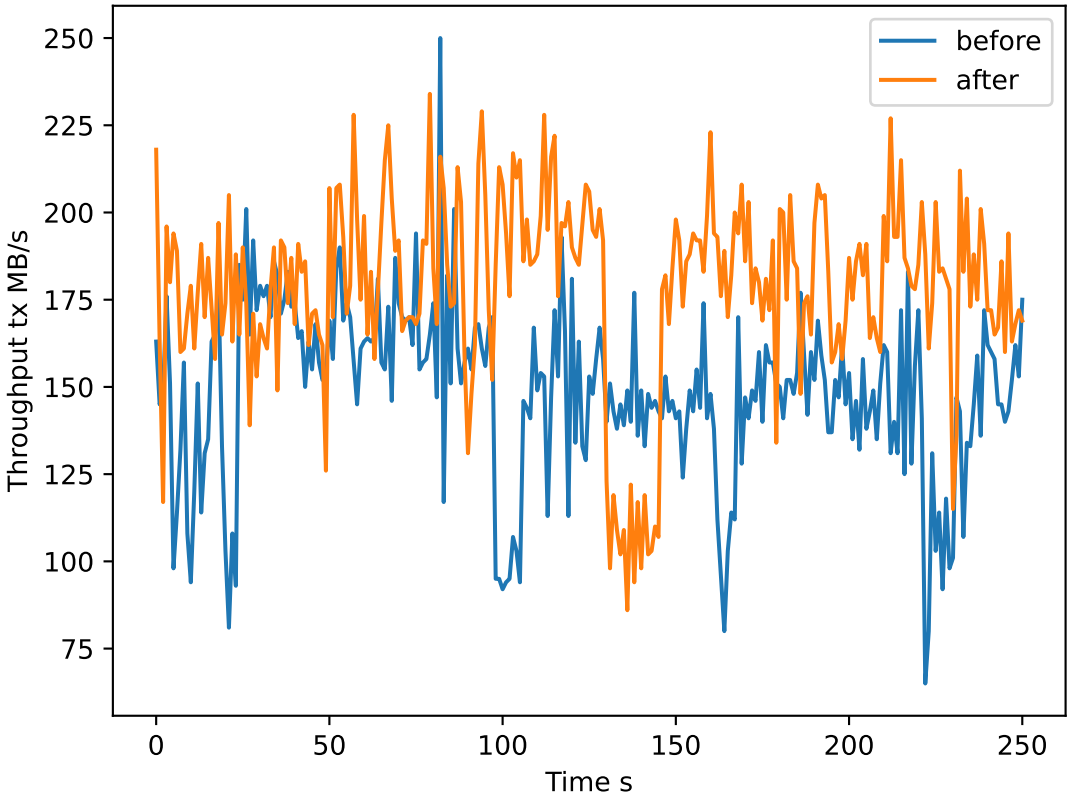
Rysunek 4: Natężenie ruchu z interfejsu eth5 w węźle s0



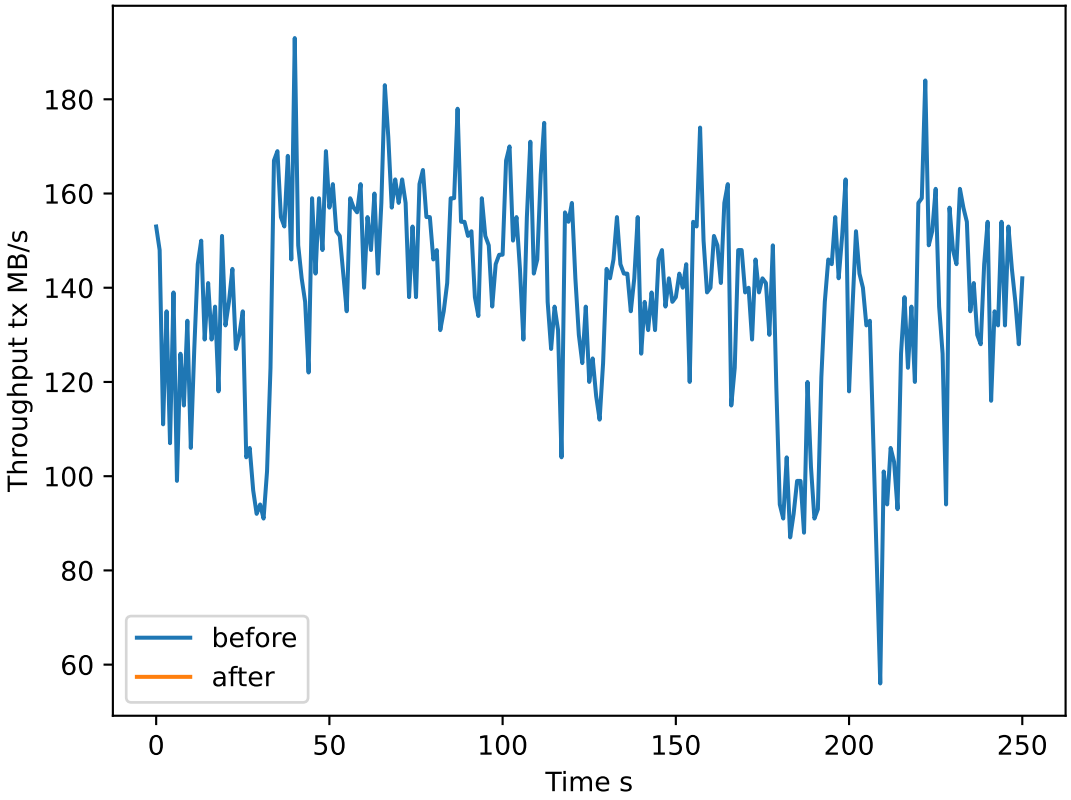
Rysunek 5: Natężenie ruchu z interfejsu eth1 w węźle s1



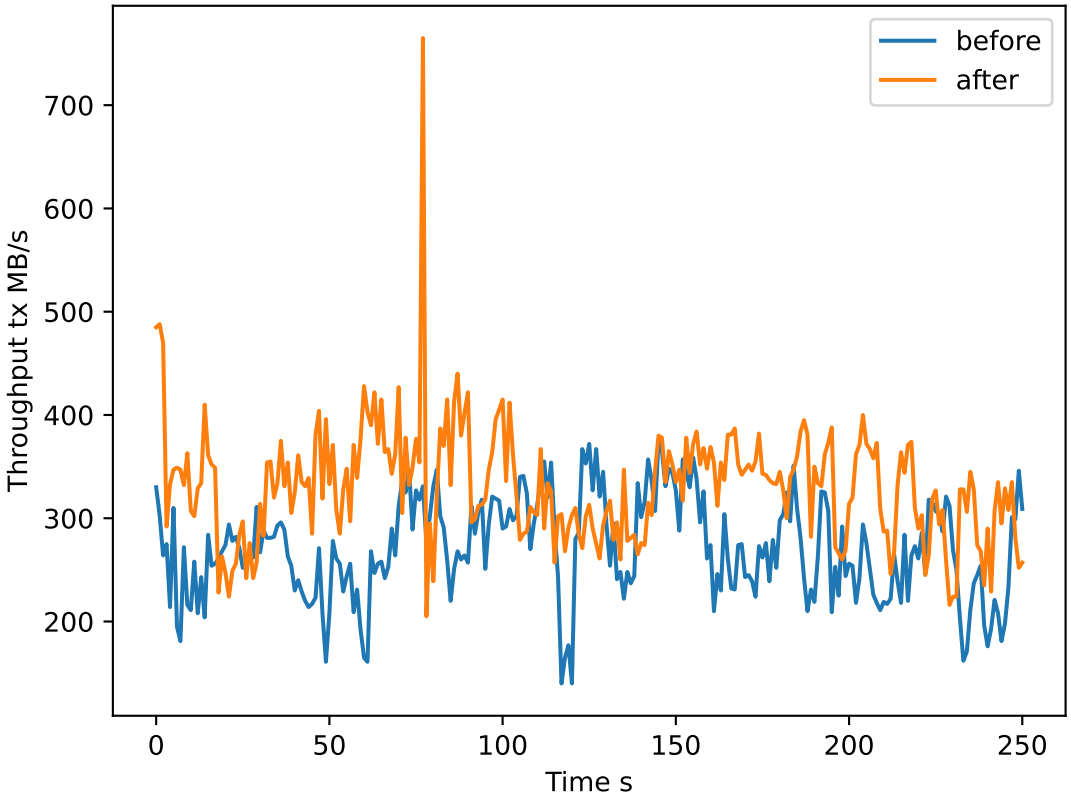
Rysunek 6: Natężenie ruchu z interfejsu eth4 w węźle s1



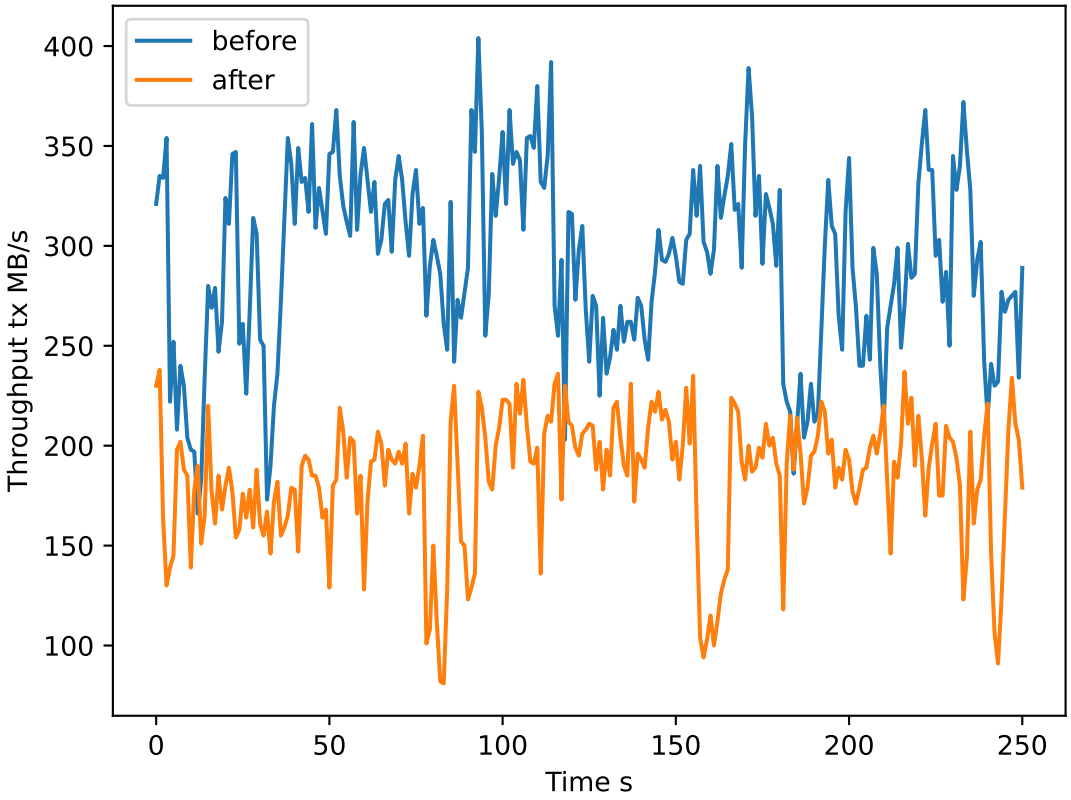
Rysunek 7: Natężenie ruchu z interfejsu eth1 w węźle s2



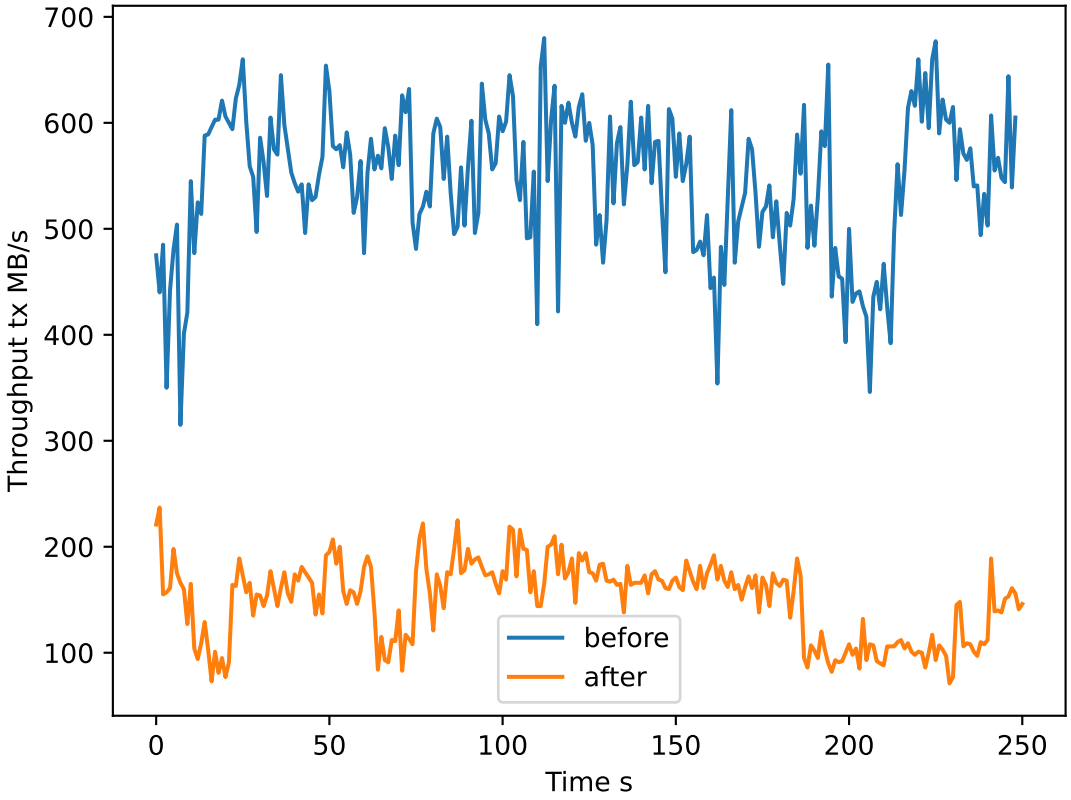
Rysunek 8: Natężenie ruchu z interfejsu eth2 w węźle s2



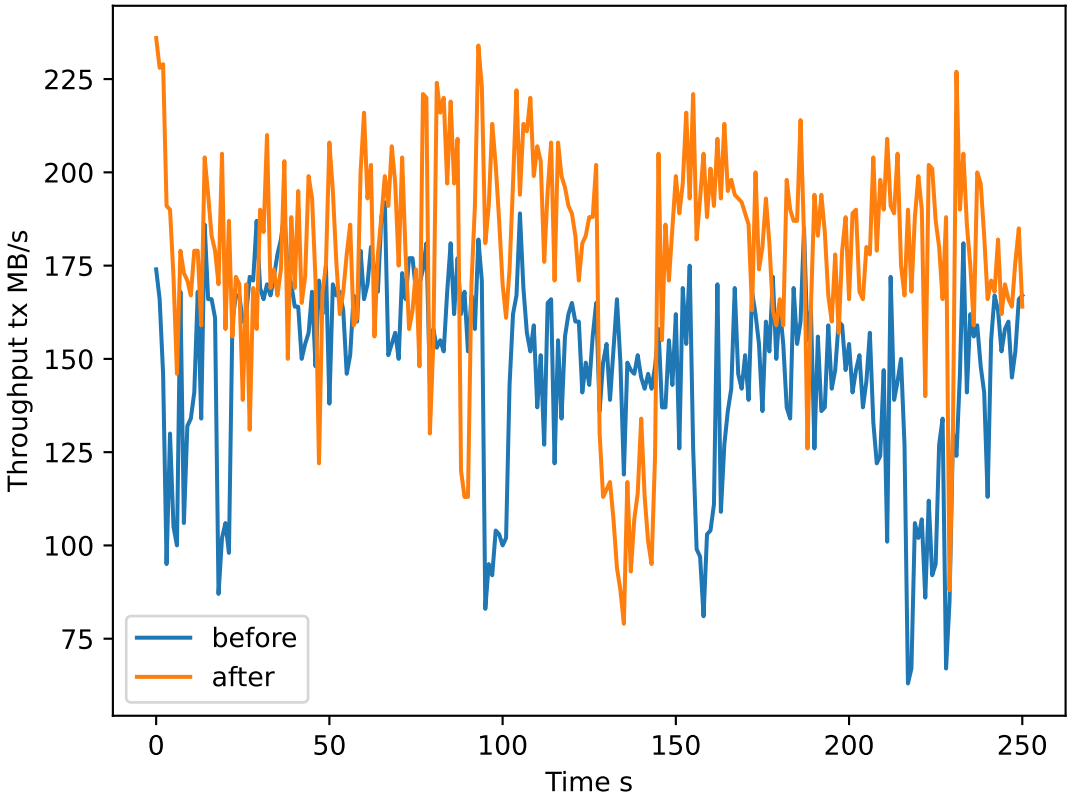
Rysunek 9: Natężenie ruchu z interfejsu eth1 w węźle s3



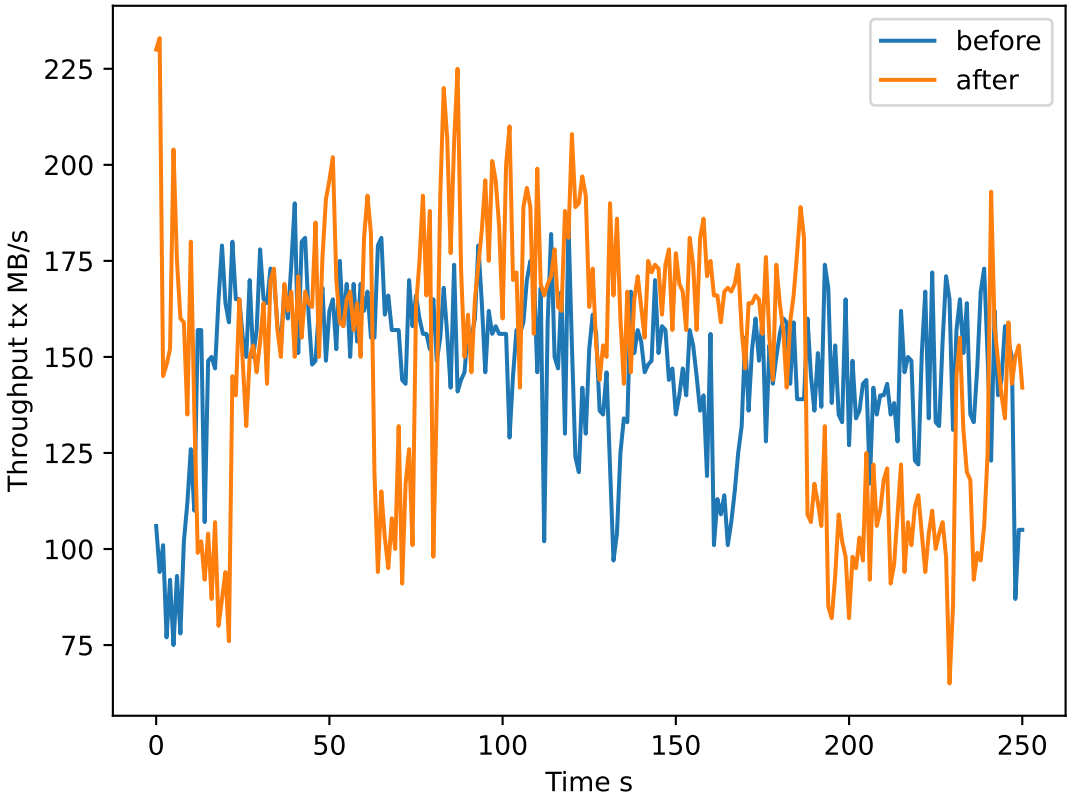
Rysunek 10: Natężenie ruchu z interfejsu eth2 w węźle s3



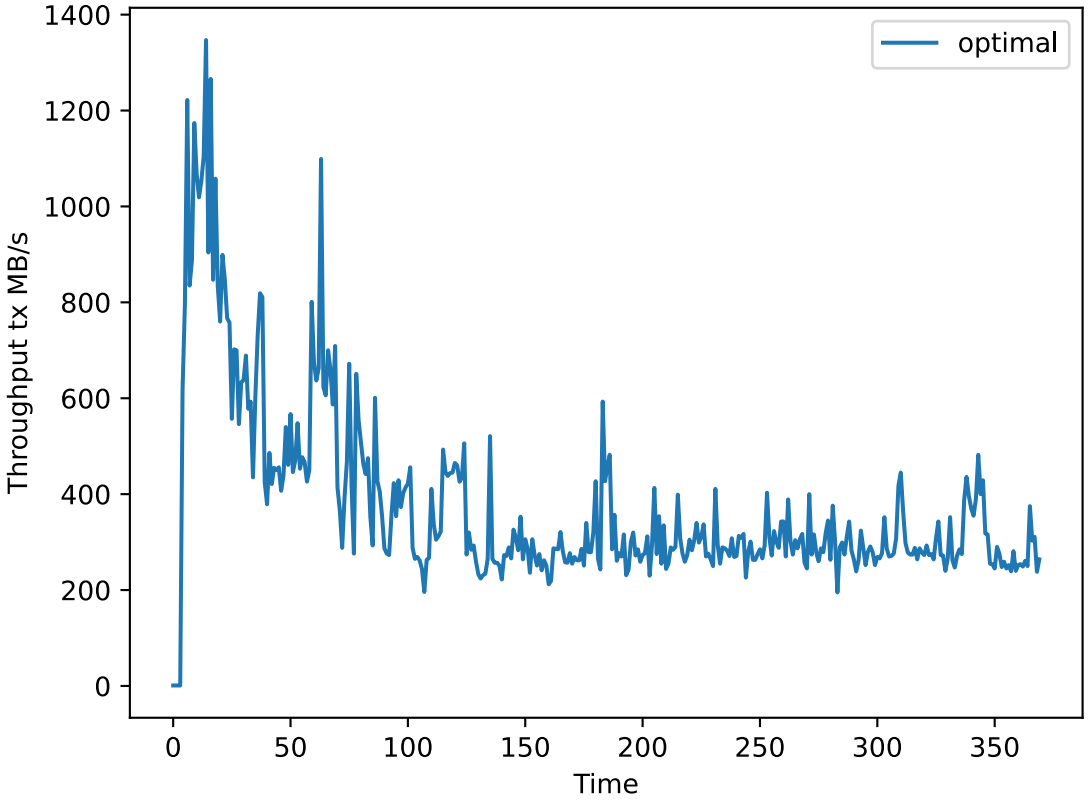
Rysunek 11: Natężenie ruchu z interfejsu eth3 w węźle s3



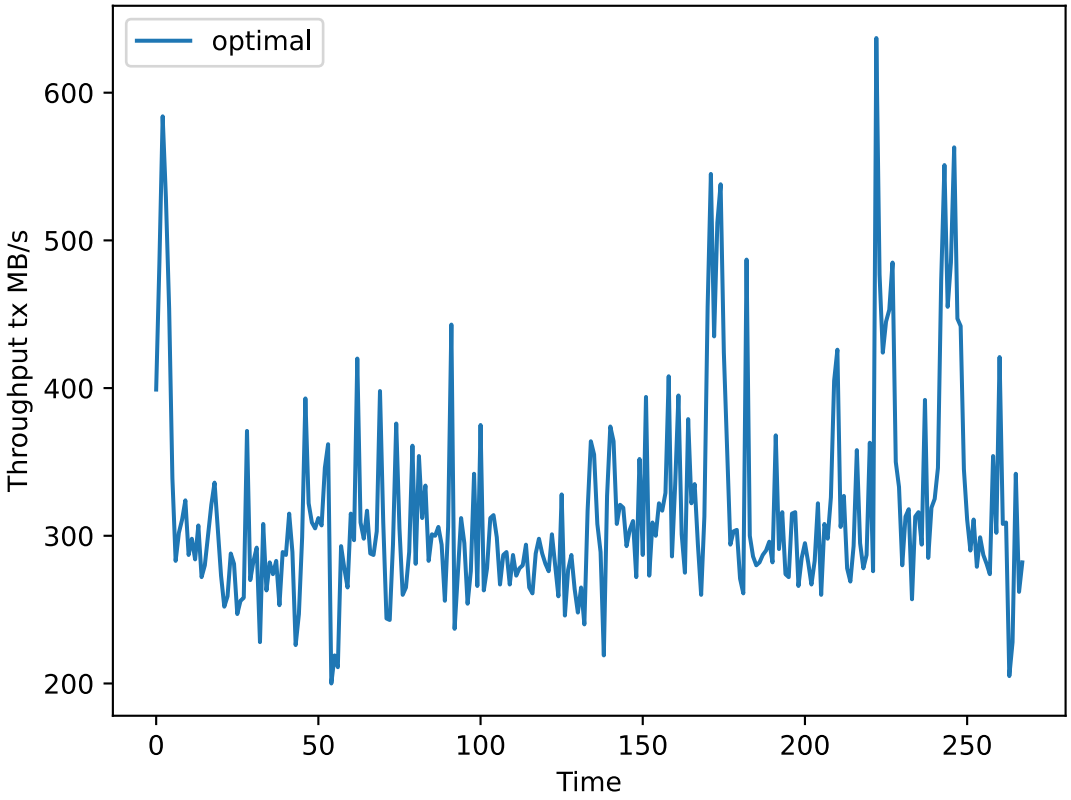
Rysunek 12: Natężenie ruchu z interfejsu eth5 w węźle s3



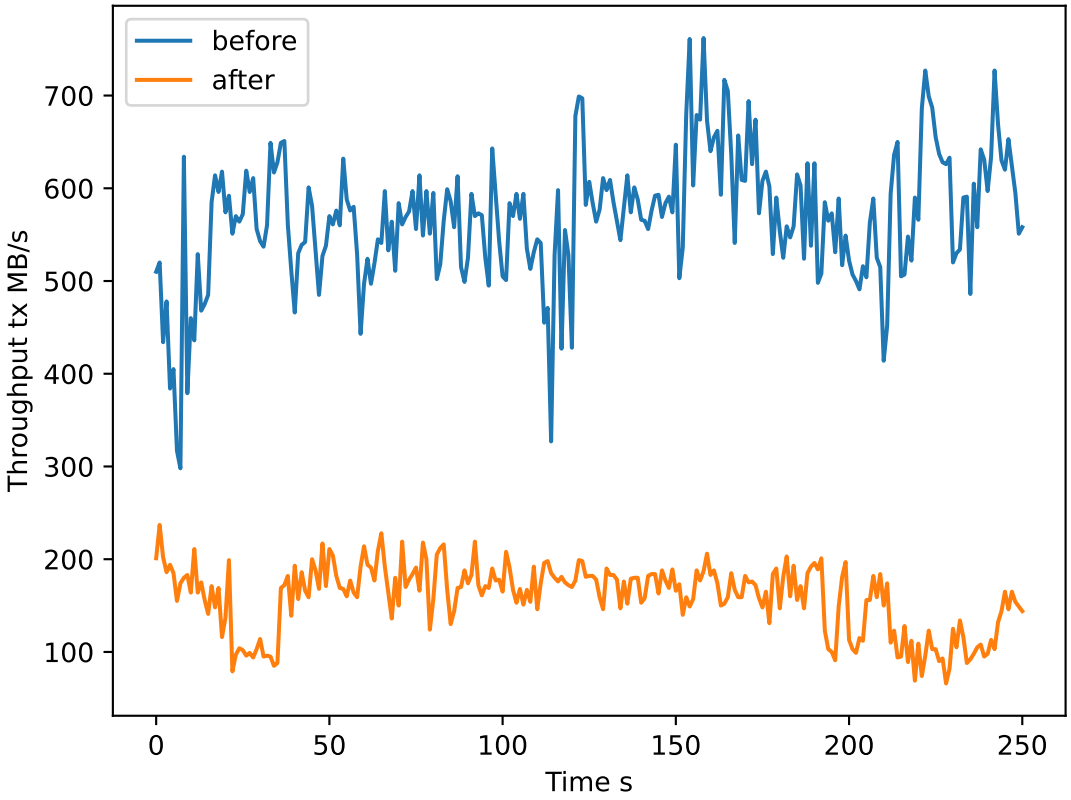
Rysunek 13: Natężenie ruchu z interfejsu eth1 w węźle s4



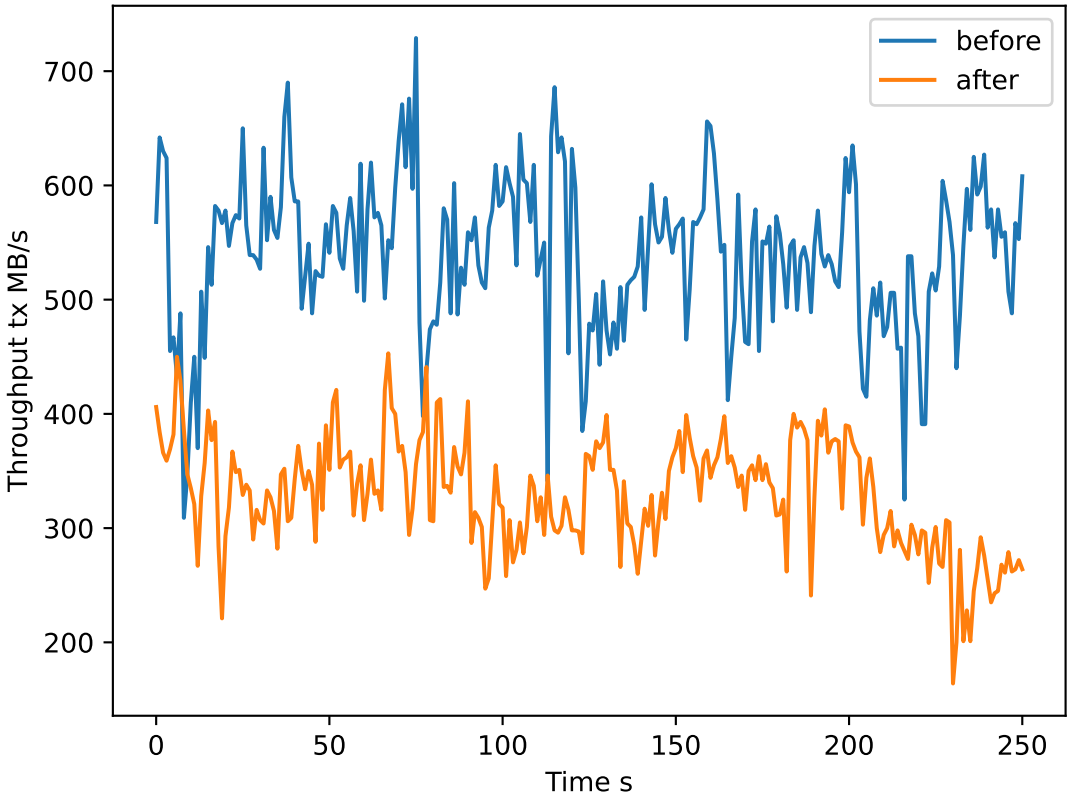
Rysunek 14: Natężenie ruchu z interfejsu eth5 w węźle s4



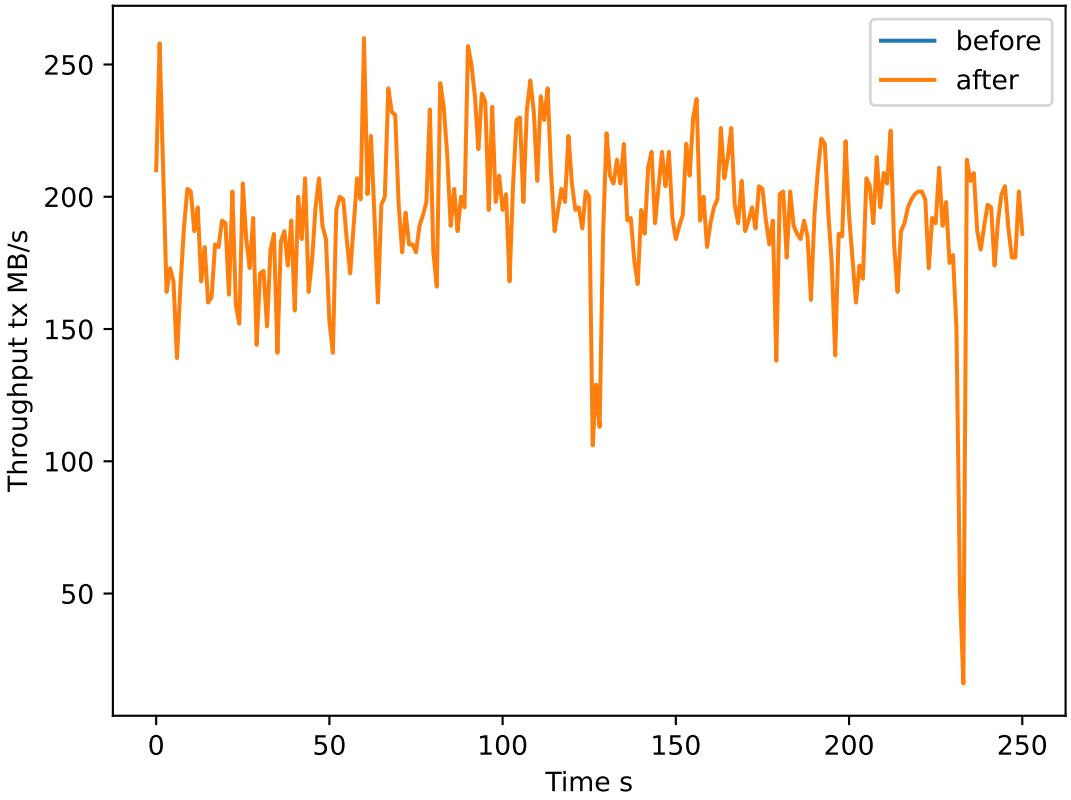
Rysunek 15: Natężenie ruchu z interfejsu eth11 w węźle s5



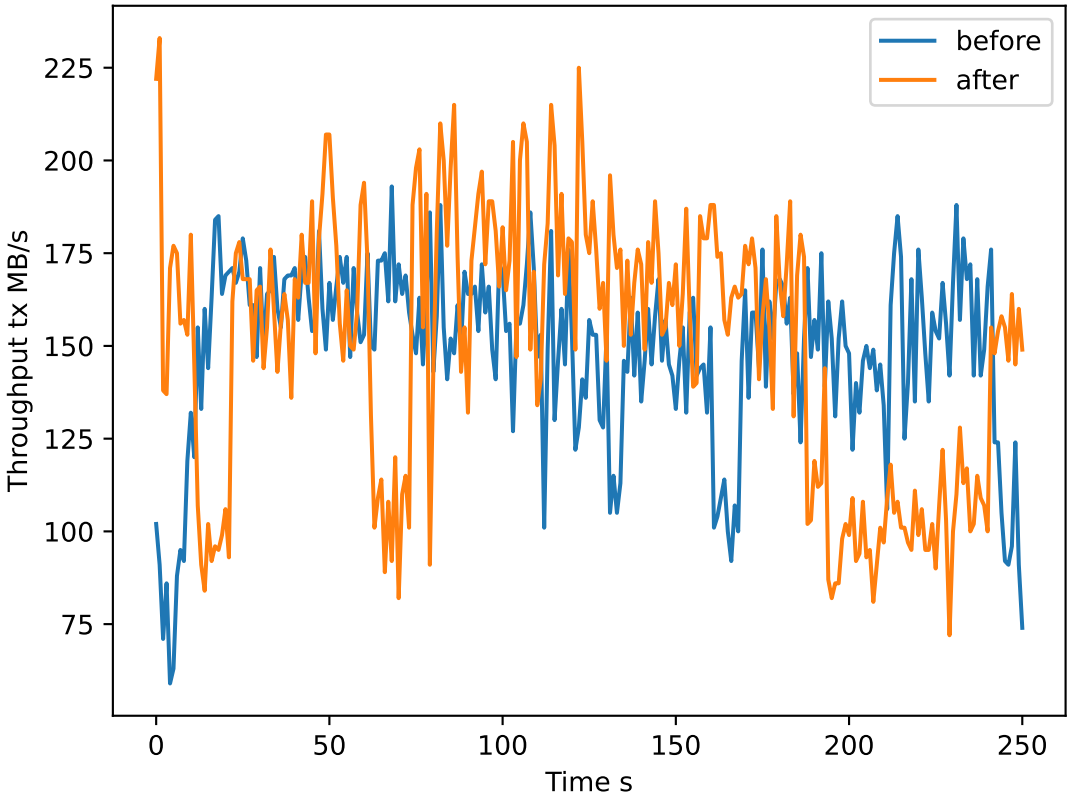
Rysunek 16: Natężenie ruchu z interfejsu eth2 w węźle s5



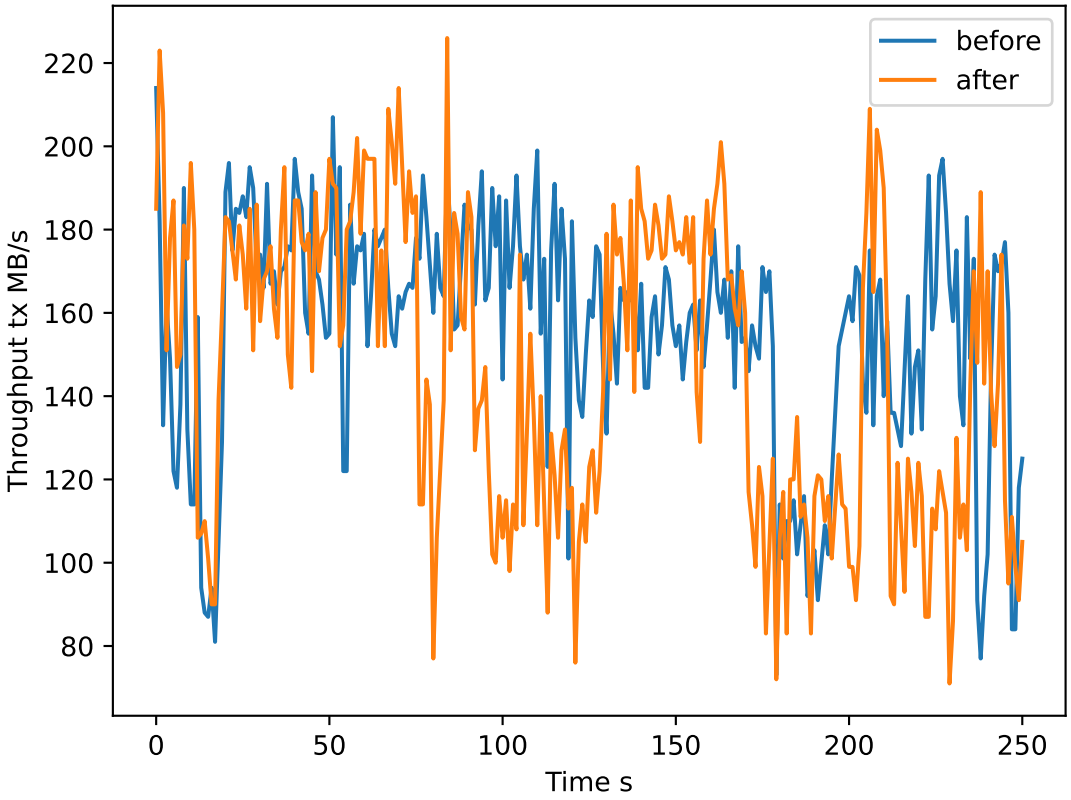
Rysunek 17: Natężenie ruchu z interfejsu eth4 w węźle s5



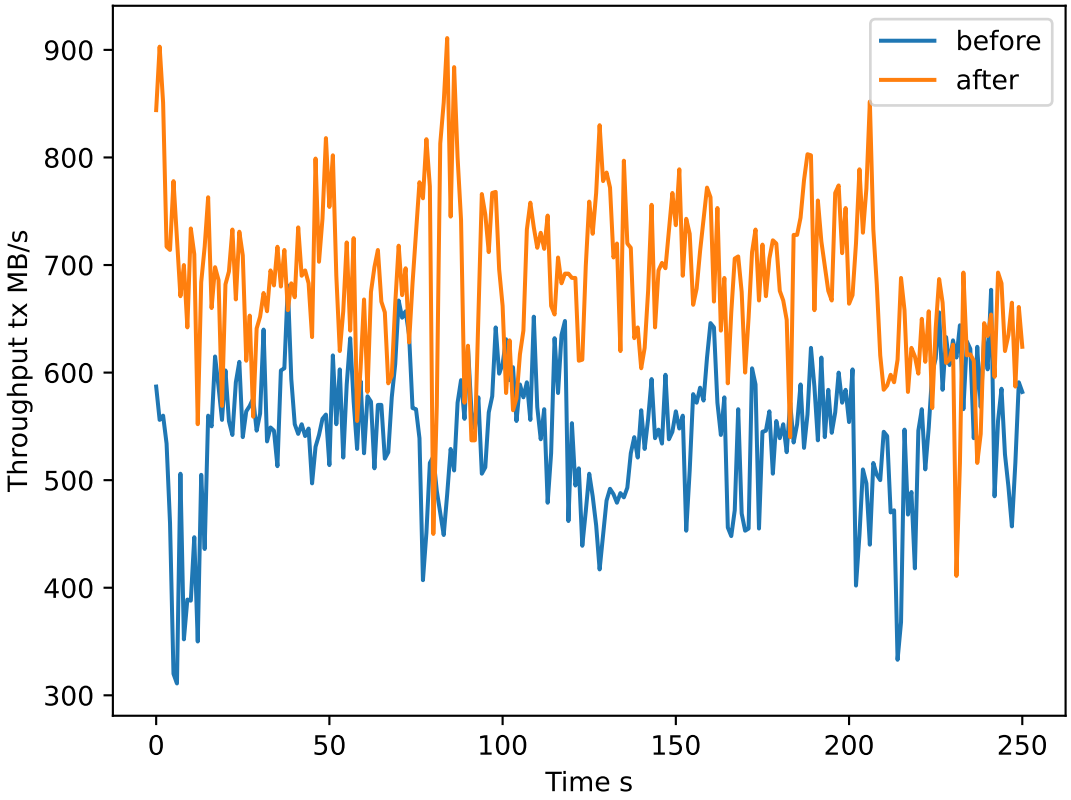
Rysunek 18: Natężenie ruchu z interfejsu eth6 w węźle s5



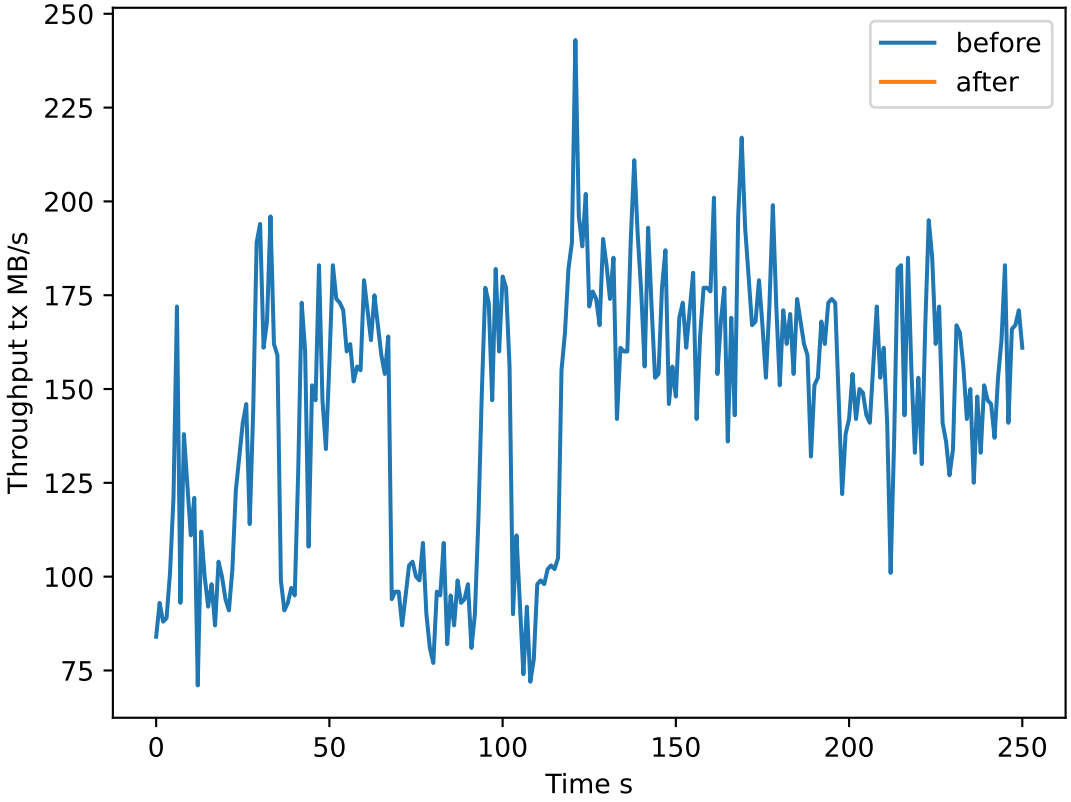
Rysunek 19: Natężenie ruchu z interfejsu eth7 w węźle s5



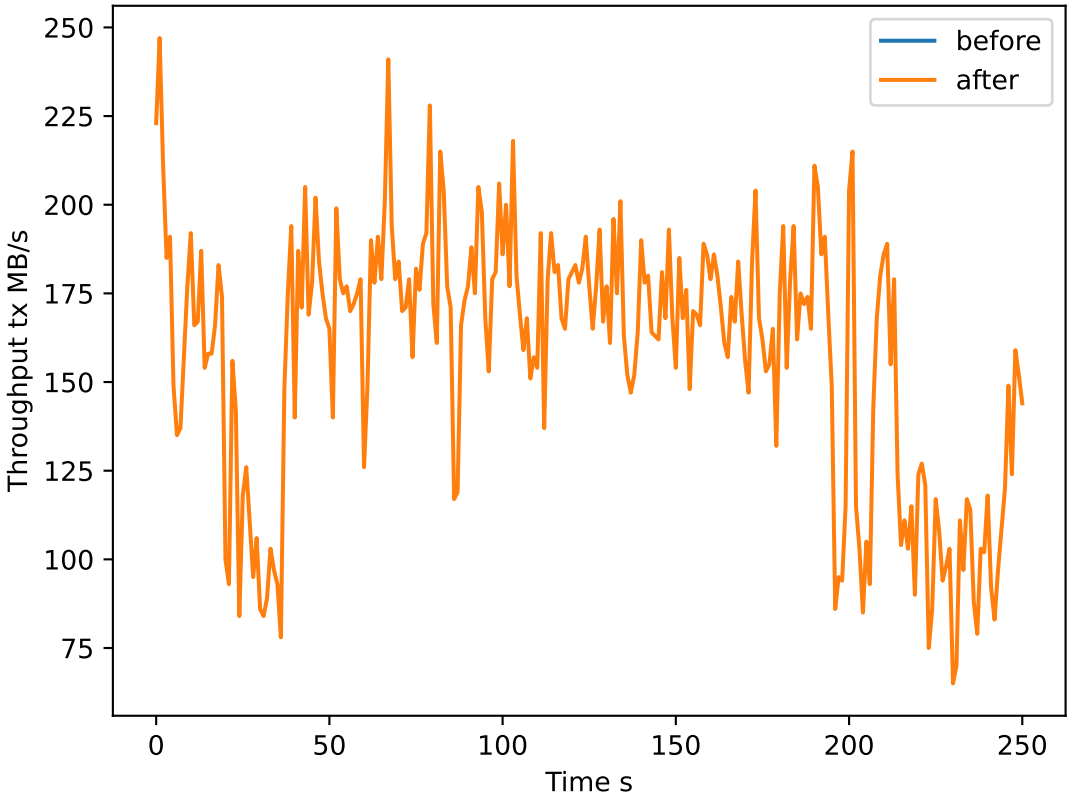
Rysunek 20: Natężenie ruchu z interfejsu eth8 w węźle s5



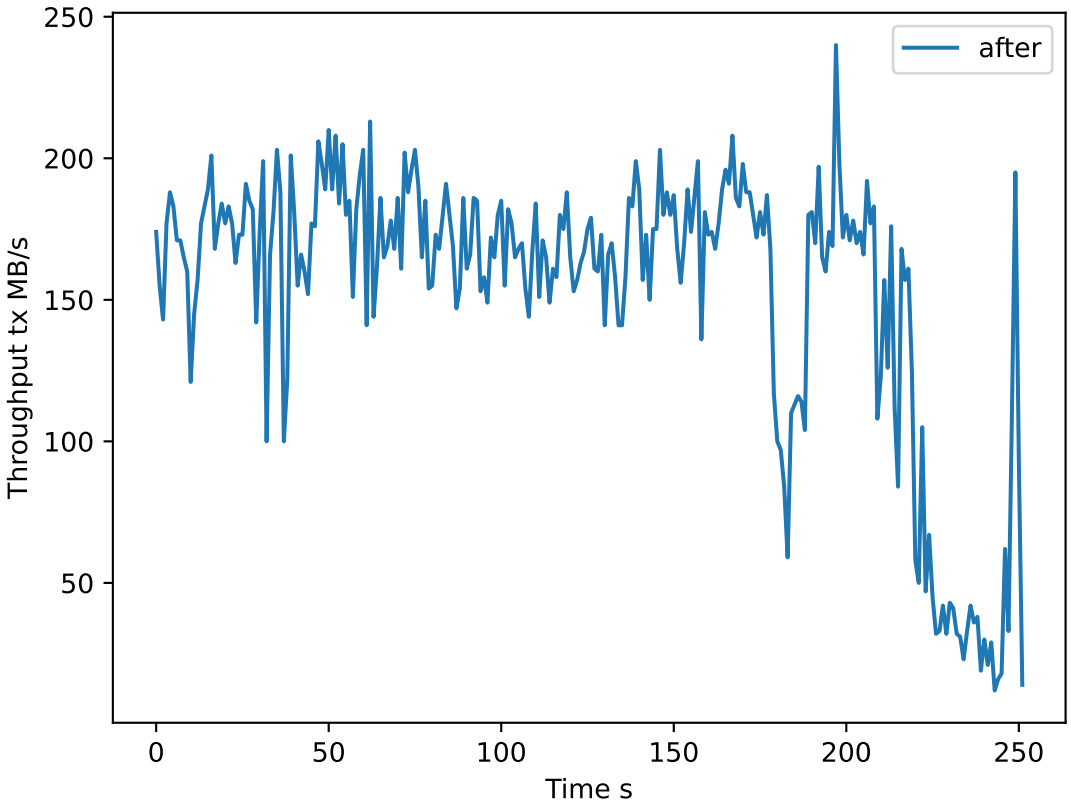
Rysunek 21: Natężenie ruchu z interfejsu eth1 w węźle s6



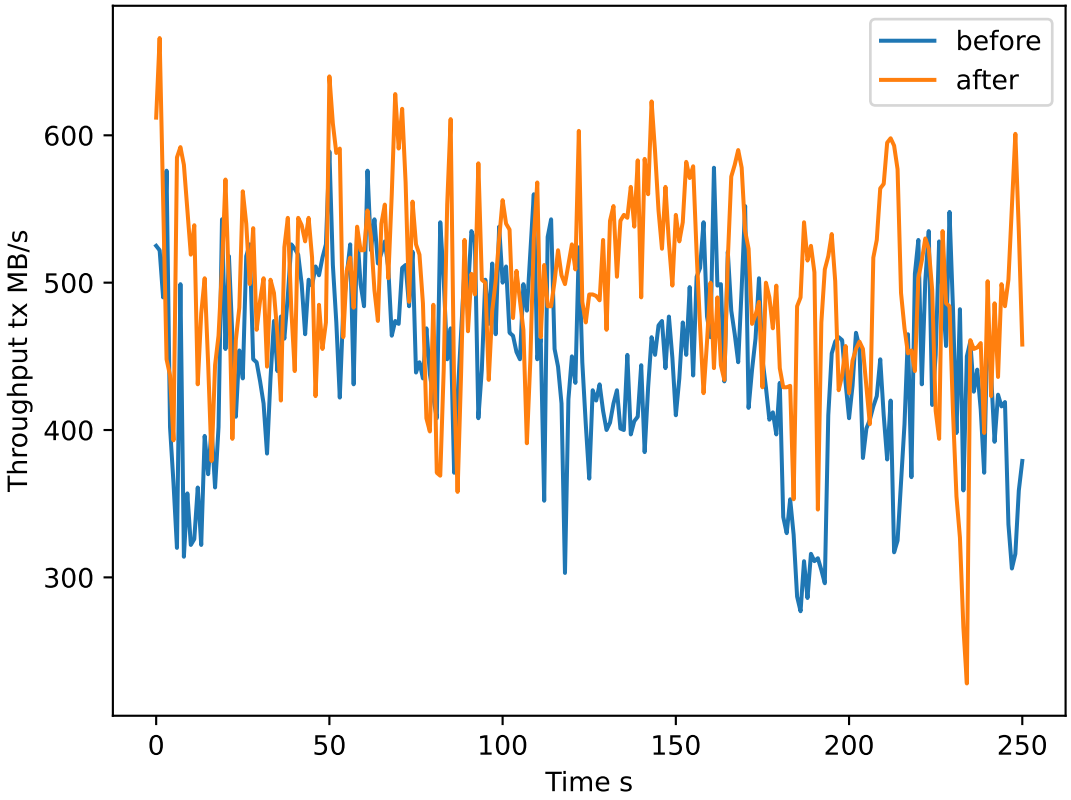
Rysunek 22: Natężenie ruchu z interfejsu eth2 w węźle s6



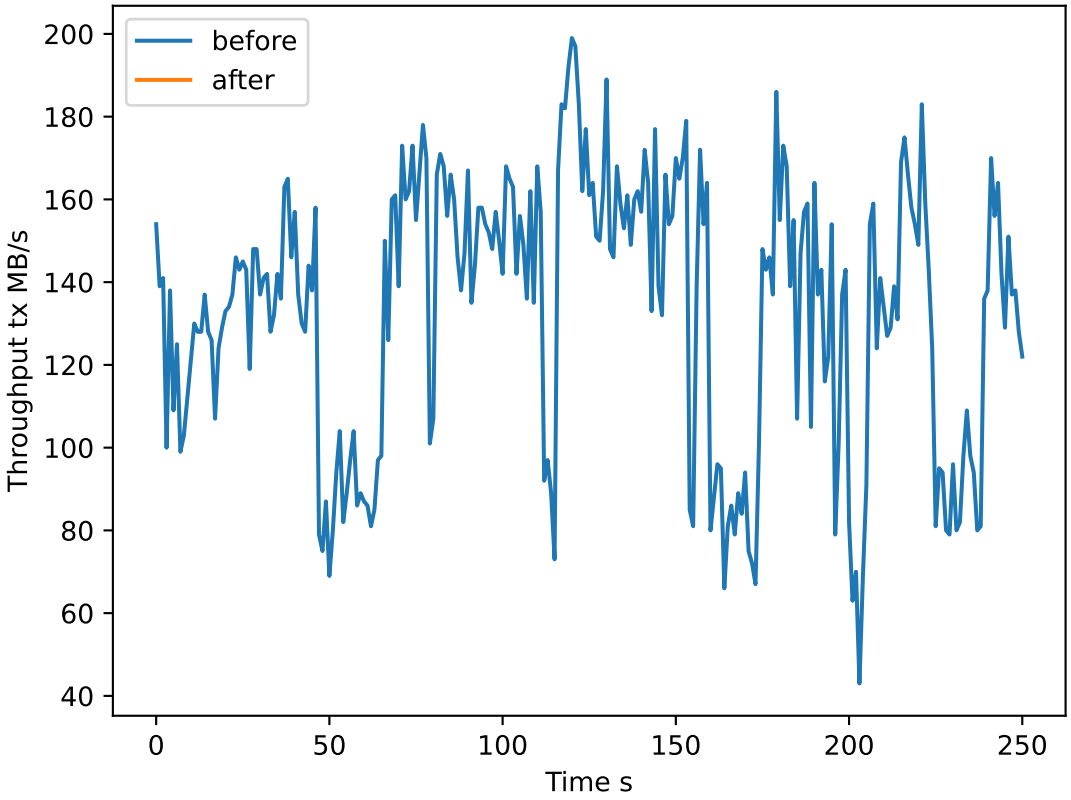
Rysunek 23: Natężenie ruchu z interfejsu eth3 w węźle s6



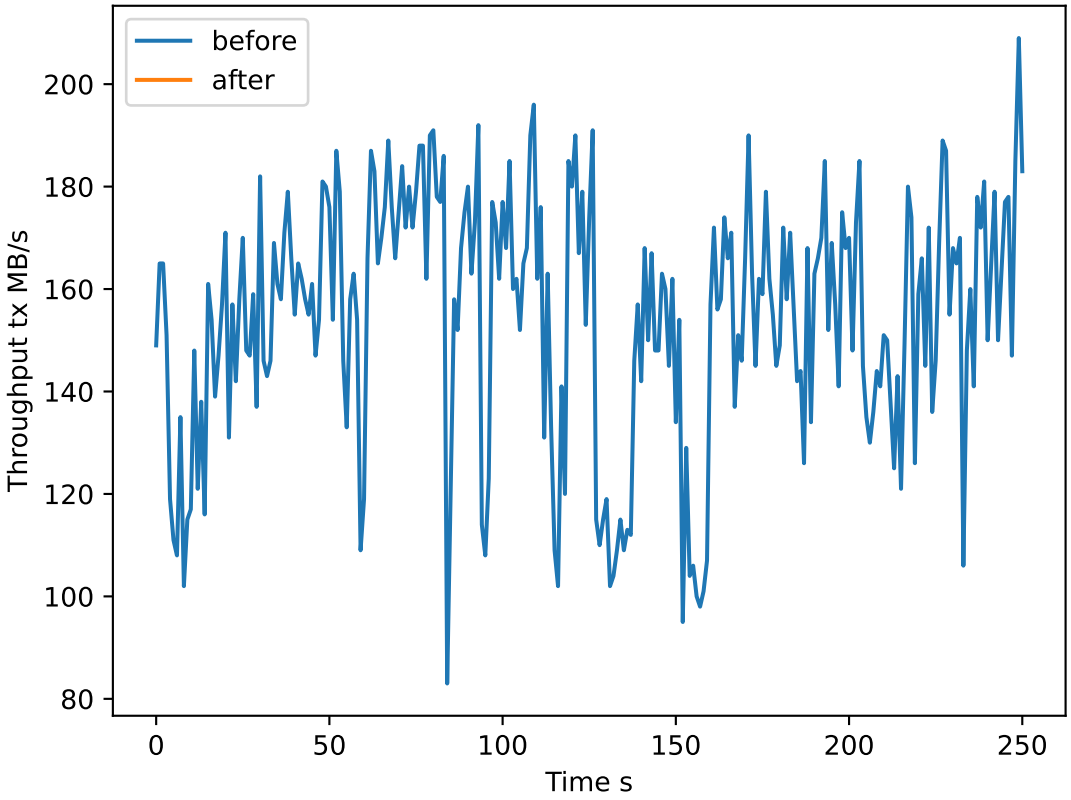
Rysunek 24: Natężenie ruchu z interfejsu eth5 w węźle s6



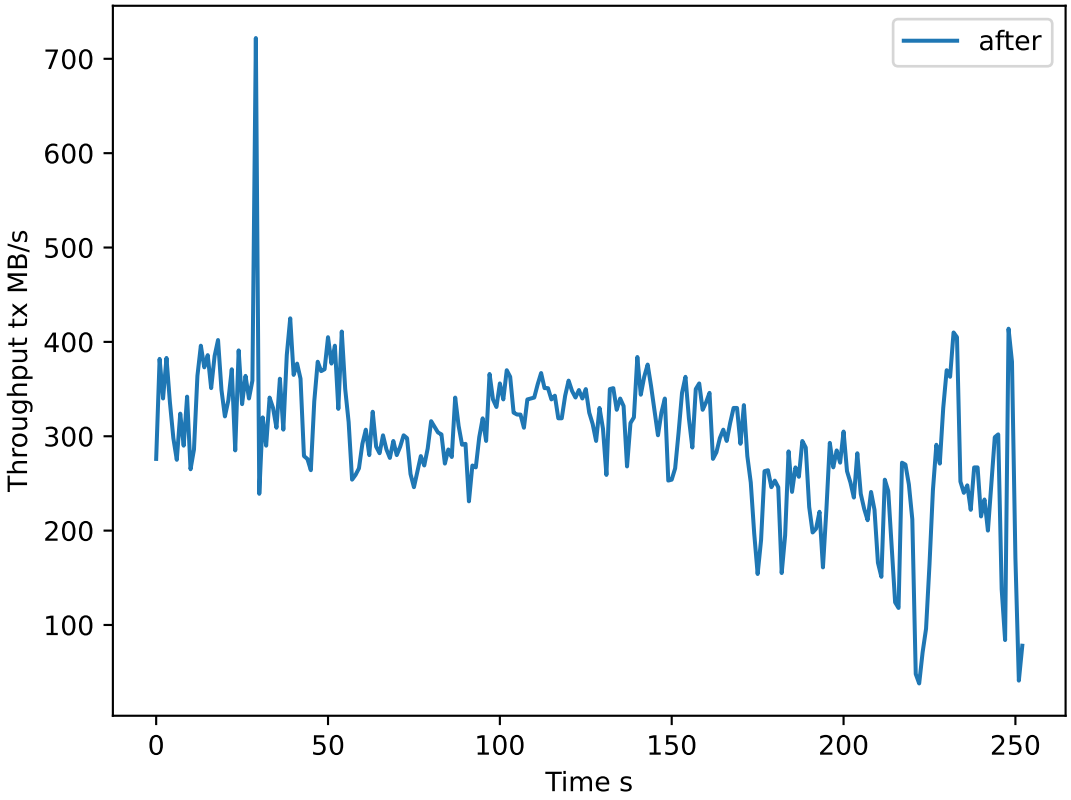
Rysunek 25: Natężenie ruchu z interfejsu eth1 w węźle s7



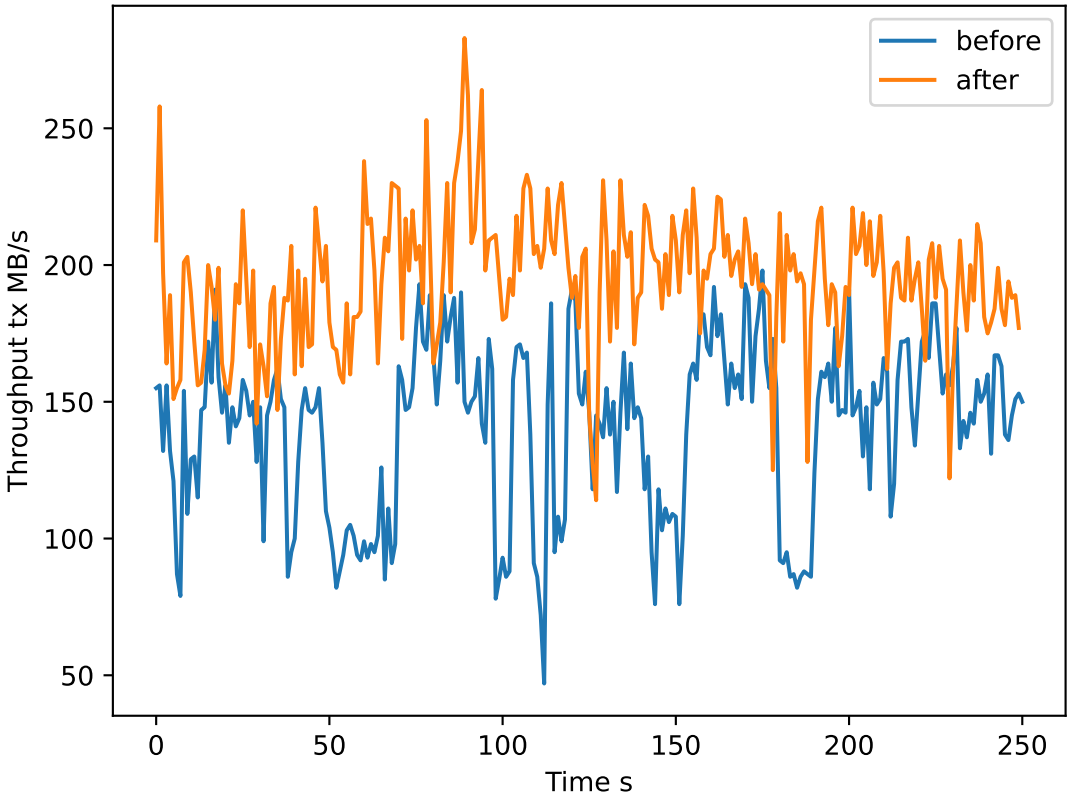
Rysunek 26: Natężenie ruchu z interfejsu eth3 w węźle s7



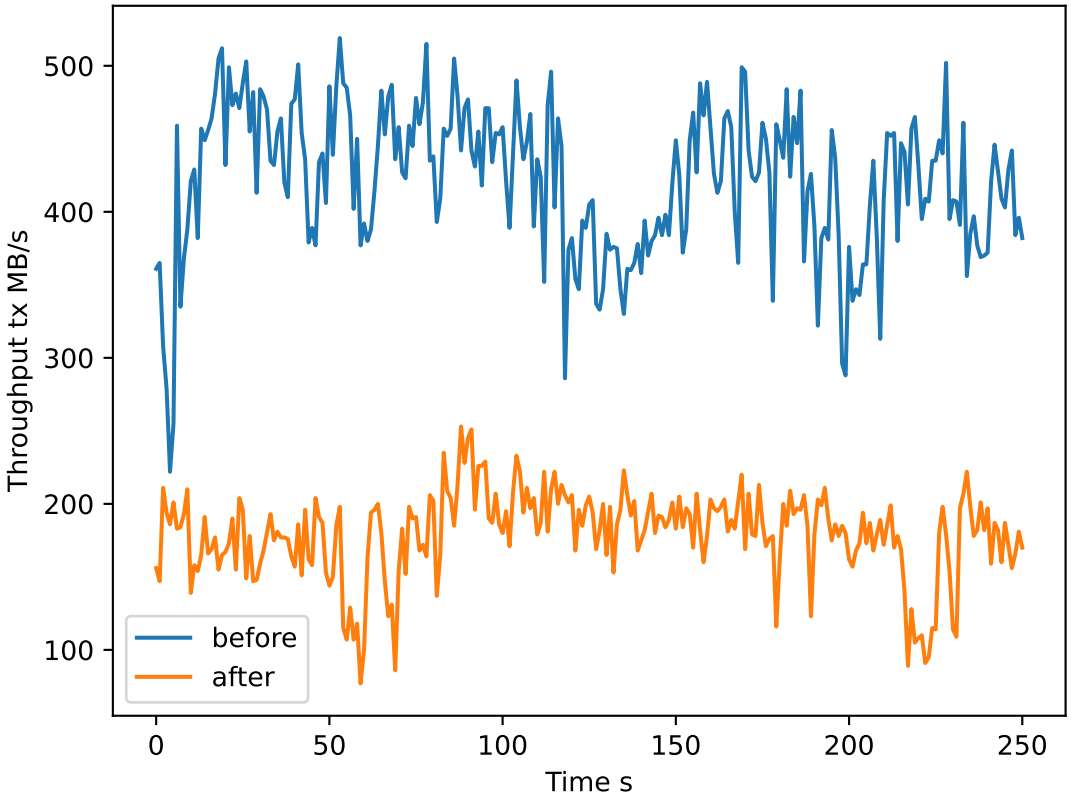
Rysunek 27: Natężenie ruchu z interfejsu eth5 w węźle s7



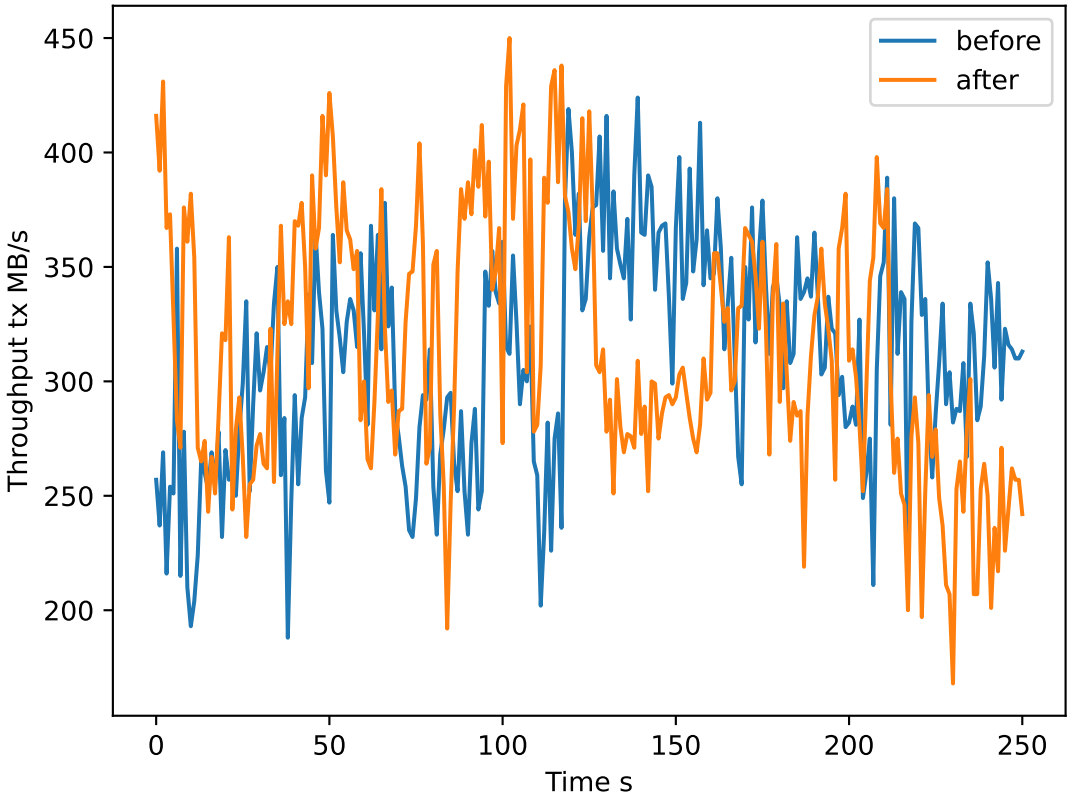
Rysunek 28: Natężenie ruchu z interfejsu eth6 w węźle s7



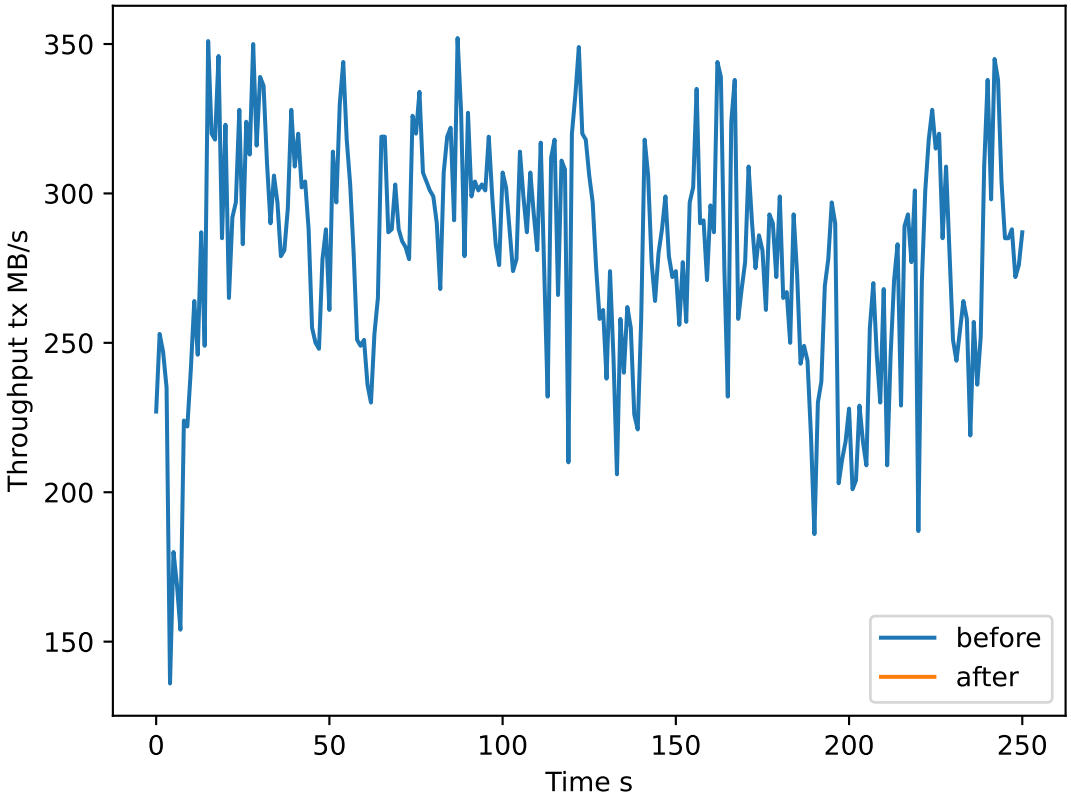
Rysunek 29: Natężenie ruchu z interfejsu eth1 w węźle s8



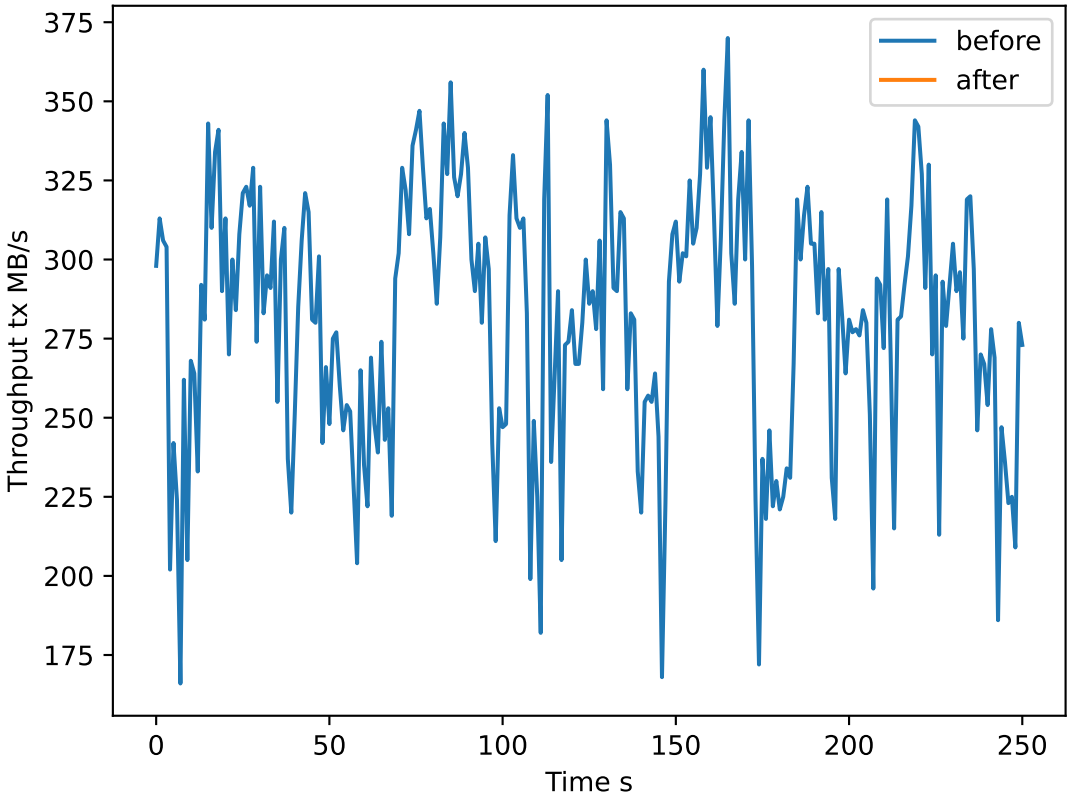
Rysunek 30: Natężenie ruchu z interfejsu eth2 w węźle s8



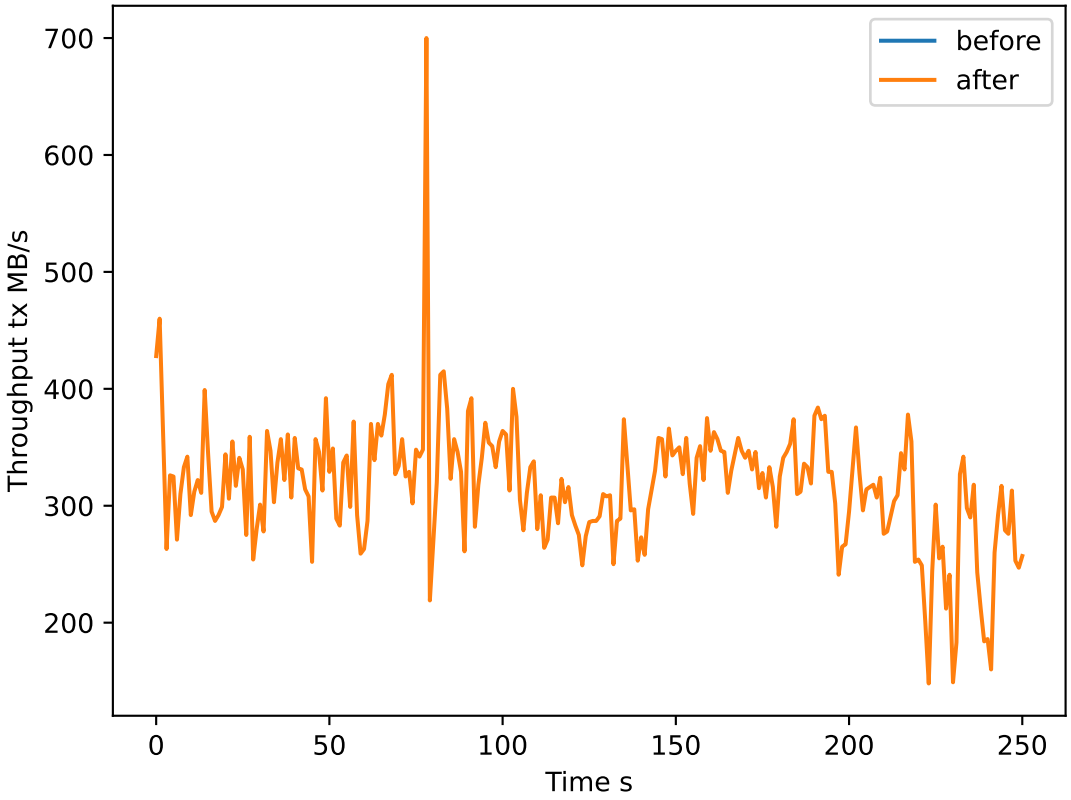
Rysunek 31: Natężenie ruchu z interfejsu eth1 w węźle s9



Rysunek 32: Natężenie ruchu z interfejsu eth3 w węźle s9



Rysunek 33: Natężenie ruchu z interfejsu eth4 w węźle s9



Rysunek 34: Natężenie ruchu z interfejsu eth5 w węźle s9