Wydział Elektroniki
i Technik Informacyjnych

POLITECHNIKA WARSZAWSKA

Sieci wielousługowe - projekt

Zbadanie algorytmu szeregowania pakietów opartych na cyklu i porównanie z algorytmem opartym na priorytetach

Prowadzący: dr hab. inż. Wojciech Burakowski

Paweł Denst 303927, Damian Tosiński 303983

31 stycznia 2022

Spis treści

1. Założenia projektu

Projekt ma na celu zbadanie czy algorytm obsługi pakietów ma wpływ na średnie czasy oczekiwania pakietów przesyłanych w ramach danego strumienia. W tym celu zaprojektowany został symulator, zakodowany w języku Python, który ma na celu porównanie dwóch algorytmów szeregowania pakietów:

— algorytmu szeregowania pakietów opartego na priorytetach
— algorytmu szeregowania pakietów opartego na cyklu

Projekt dla uproszczenia zakłada istnienie nieskończenie długich kolejek oraz tożsame wielkości przesyłanych pakietów. Szczegółowy opis warunków symulatora został dogłębniej przedstawiony przy każdym z testów oraz w rozdziale środowisko symulacyjne.

2. Wstęp teoretyczny

2.1. Algorytm szeregowania pakietów oparty na cyklu

Algorytm szeregowania oparty na cyklu jest specjalnie zaprojektowany dla systemów z podziałem czasu. Procesy są umieszczane w gotowej kolejce, która w tym przypadku jest kolejką cykliczną. Zdefiniowana jest mała jednostka czasu znana jako kwant czasu. Algorytm wybiera pierwszy proces z kolejki i wykonuje go przez czas, określony za pomocą kwantu czasu. Następnie czas procesora przekazywany jest kolejnej kolejce. Zadanie jest wznawiane następnym razem, gdy do tej kolejki zostanie przydzielony przedział czasowy. Jeśli proces zakończy się podczas przypisanego mu kwantu czasu, algorytm wybiera do wykonania kolejny, pierwszy proces z kolejki gotowości.

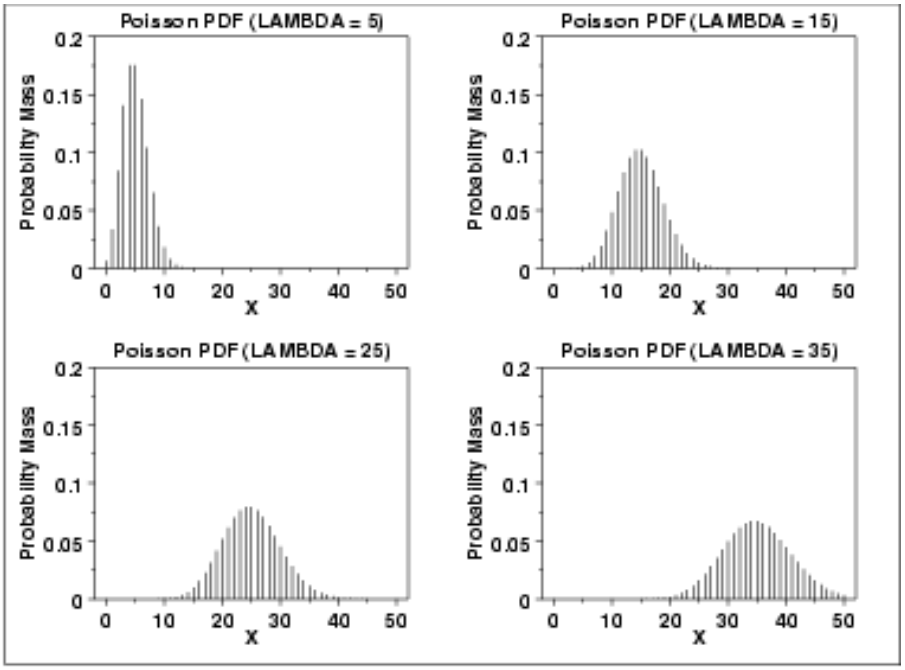
2.2. Algorytm szeregowania pakietów oparty na priorytetach

Szeregowanie pakietów składa się z kilku kolejek, z których zawsze najpierw obsługiwane są te o wyższym priorytecie. W razie przekroczenia limitu wielkości kolejek, pierwsze kasowane są pakiety z kolejki o najniższym priorytecie. Jednak w naszym przypadku wielkości kolejek są nieskończone. Ten rodzaj kolejkowania ma zastosowanie w sytuacji, kiedy określony rodzaj ruchu musi posiadać absolutne pierwszeństwo i nie nadaje się do optymalizacji wykorzystania łącza. Każda kolejka priorytetowa jest z kolei obsługiwana zgodnie z metodą „First In First Out". Pakiet nie może zostać obsłużony w danej kolejce, dopóki kolejka hierarchicznie wyższa nie zostanie opróżniona. Dopiero wtedy obsługiwane są bufory o niższych priorytetach.

2.3. Rozkład Poissona

Rozkład Poissona jest dominującym modelem używanym do analizy ruchu w tradycyjnych sieciach teleinformatycznych. Wyznacza on dyskretny rozkład prawdopodobieństwa, wyrażający prawdopodobieństwo szeregu wydarzeń mających miejsce w określonym czasie, gdy te wydarzenia występują ze znaną średnią częstotliwością i w sposób niezależny od czasu jaki upłynął od ostatniego zajścia takiego zdarzenia. Funkcja gęstości prawdopodobieństwa:

$$f(x) = \lambda e^{-\lambda x} \text{ dla x>0}$$

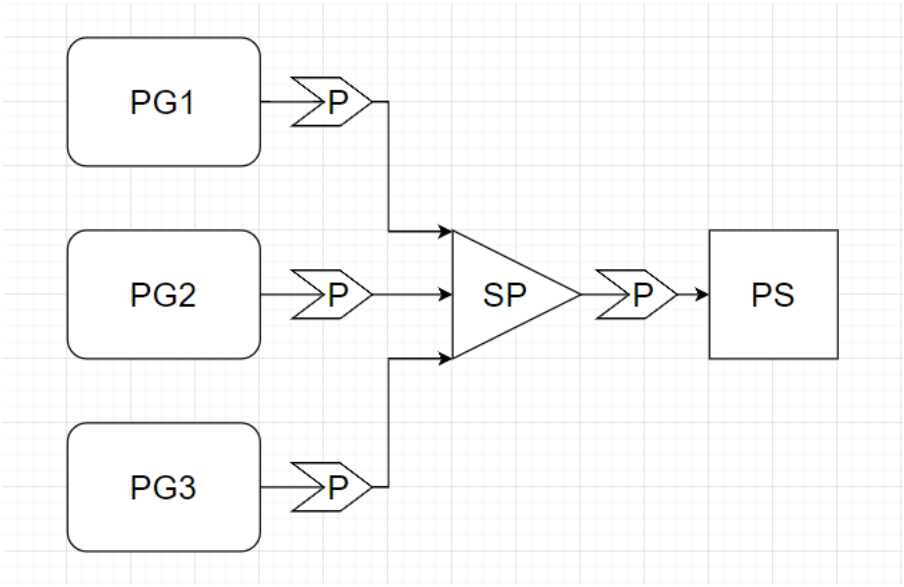


Rys. 1. Rozkład Poissona zależnie od lambdy.

3. Środowisko symulacyjne

3.1. Opis symulatora

W celu przeprowadzenia eksperymentów zaprojektowany został symulator, zakodowany w języku Python (z użyciem biblioteki SimPy - Discrete event simulation for Python), który odzwierciedla działanie rozwiązania o poniższej topologii:



Rys. 2. Topologia symulatora.

Poniżej znajduje się wydruk pliku zawierającego właściwe wywołanie symulatora. W tym miejscu ustawiana jest większość najważniejszych parametrów symulacji, takich jak:

— lambda - parametr rozkładu Poissona określający częstość wysyłania pakietów
— mean_pkt_size - rozmiar pakietu, ustawiony na 100 dla każdego z pakietów
— port_rate - określa jak szybko pakiet jest obsługiwany (u nas zawsze z przepływnością jeden pakiet na jednostkę czasu)
— env.run(until=2000) - czas symulacji mierzony w tikach

wywołanie symulatora

```
import matplotlib.pyplot as plt
import random
import functools
import simpy
from PriorityComponents import PacketGenerator, PacketSink, SwitchPort

lambda_= 0.5
mu_ = 1
mean_pkt_size = 100
def same_size():
    return 100

adist = functools.partial(random.expovariate, lambda_)
sdist = functools.partial(same_size)

port_rate = mu_*8*mean_pkt_size

env = simpy.Environment()

ps = PacketSink(env, debug=False, rec_arrivals=True, rec_waits=True)
pg = PacketGenerator(env, "ID", adist, sdist, flow_id=1)
pg2 = PacketGenerator(env, "ID", adist, sdist, flow_id=2)
pg3 = PacketGenerator(env, "ID", adist, sdist, flow_id=3)

switch_port = SwitchPort(env, port_rate, qlimit=None)

pg.out = switch_port
pg2.out = switch_port
pg3.out = switch_port

switch_port.out = ps

env.run(until=2000)

\newpage
```

3.1.1. PG - Packet Generator

Generator pakietów, którego główną funkcją jest tworzenie i wysyłanie pakietów (zgodnie z rozkładem poissona) do Switch Port'a. Dzięki parametrowi flow_id możliwe jest rozróżnienie kolejek.

klasa generatora pakietów

```
class PacketGenerator(object):
    def __init__(self, env, id,  adist, sdist, flow_id=0):
        self.id = id
        self.env = env
        self.adist = adist
        self.sdist = sdist
        self.out = None
        self.action = env.process(self.run())
        self.flow_id = flow_id

    def run(self):
        yield self.env.timeout(self.initial_delay)
        while(True):
            yield self.env.timeout(self.adist())
            p = Packet(self.env.now, self.sdist(), src=self.id,
            ↪  flow_id=self.flow_id)
            self.out.put(p)
```

3.1.2. P - Packet

Klasa pakietu pozwalająca na rozpoznanie przesyłanej informacji za pomocą flow_id. Parametr time pozwala na obliczenia związane z czasem przyjścia/oczekiwania pakietu.

klasa pakietu

```
class Packet(object):
    def __init__(self, time, size, id, src="", dst="", flow_id=0):
        self.time = time
        self.size = size
        self.id = id
        self.src = src
        self.dst = dst
        self.flow_id = flow_id
```

3.1.3. SP - Switch Port

Pośredniczy w wymianie pakietów generatorem pakietów, a klasą zbierającą informacje. Implementuje logikę badanego algorytmu.

klasa switch port algorytmu obsługi pakietów opartego na priorytetach

```
class SwitchPort(object):
    def __init__(self, env, rate, qlimit=None):
        self.store = simpy.PriorityStore(env)
        self.rate = rate
        self.env = env
        self.out = None
        self.qlimit = qlimit
        self.action = env.process(self.run())
        self.queue = []

    def run(self):
        while True:
            msg = (yield self.store.get())
            yield self.env.timeout(msg.size*8.0/self.rate)
            self.out.put(msg)
```

3.1.4. PS - Packet Sink

Klasa służąca zebraniu i przetworzeniu pomiarów związanych z poszczególnymi pakietami-/strumieniami.

klasa packet sink

```
class PacketSink(object):
    def __init__(self, env, rec_waits=True):
        self.store = simpy.Store(env)
        self.env = env
        self.rec_waits = rec_waits
        self.waits = []
        self.last_arrival = 0.0
        self.waits_flow1 = []
        self.waits_flow2 = []
        self.waits_flow3 = []
    def put(self, pkt):
        now = self.env.now
        if self.rec_waits:
            self.waits.append(self.env.now - pkt.time)
        if pkt.flow_id == 1:
            self.waits_flow1.append(self.env.now - pkt.time)
        if pkt.flow_id == 2:
            self.waits_flow2.append(self.env.now - pkt.time)
        if pkt.flow_id == 3:
            self.waits_flow3.append(self.env.now - pkt.time)
```

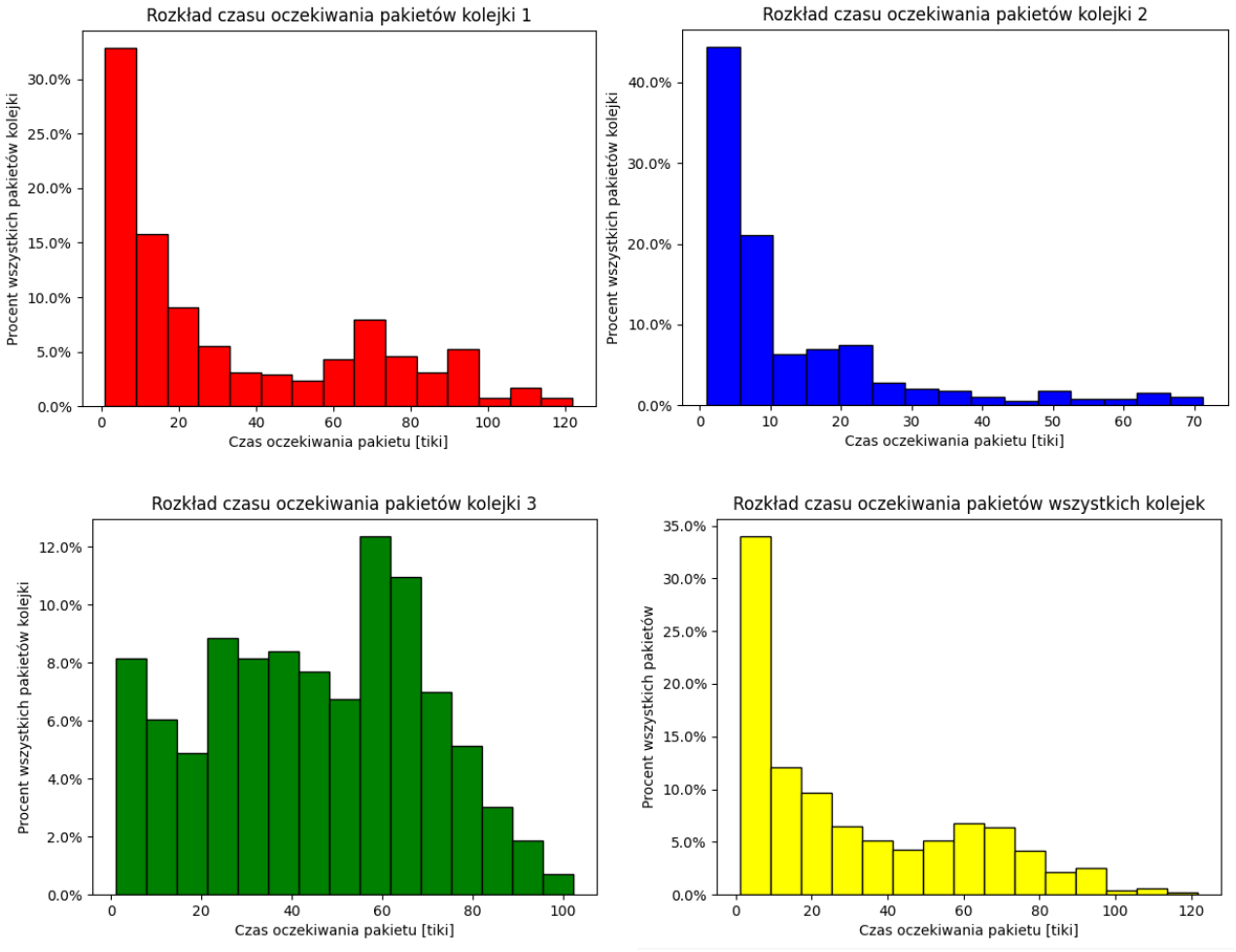
4. Scenariusze testów i rezultaty

W celu zbadania obu algorytmów przeprowadziliśmy szereg symulacji i testów, które pozwoliły nam zrozumieć wpływ poszczególnych parametrów na zasadę działania danego algorytmu. W przypadku szeregowania opartym na cyklu zmianie ulegała wartość lambdy i czas trwania cyklów. Dla algorytmu z priorytetami zmienialiśmy jedynie lambdę. Czas podany jest w ilości tyknięć zegara. Do analizy głównie poddawaliśmy średnie czasy obsługi danej kolejki.

4.1. Algorytm szeregowania oparty na cyklu

4.1.1. Lambda - 0.2, czas obsługi kolejki - 3 jednostki czasu

W tym przypadku pakiety przychodzą średnio rzadziej niż czas trwania cyklu.

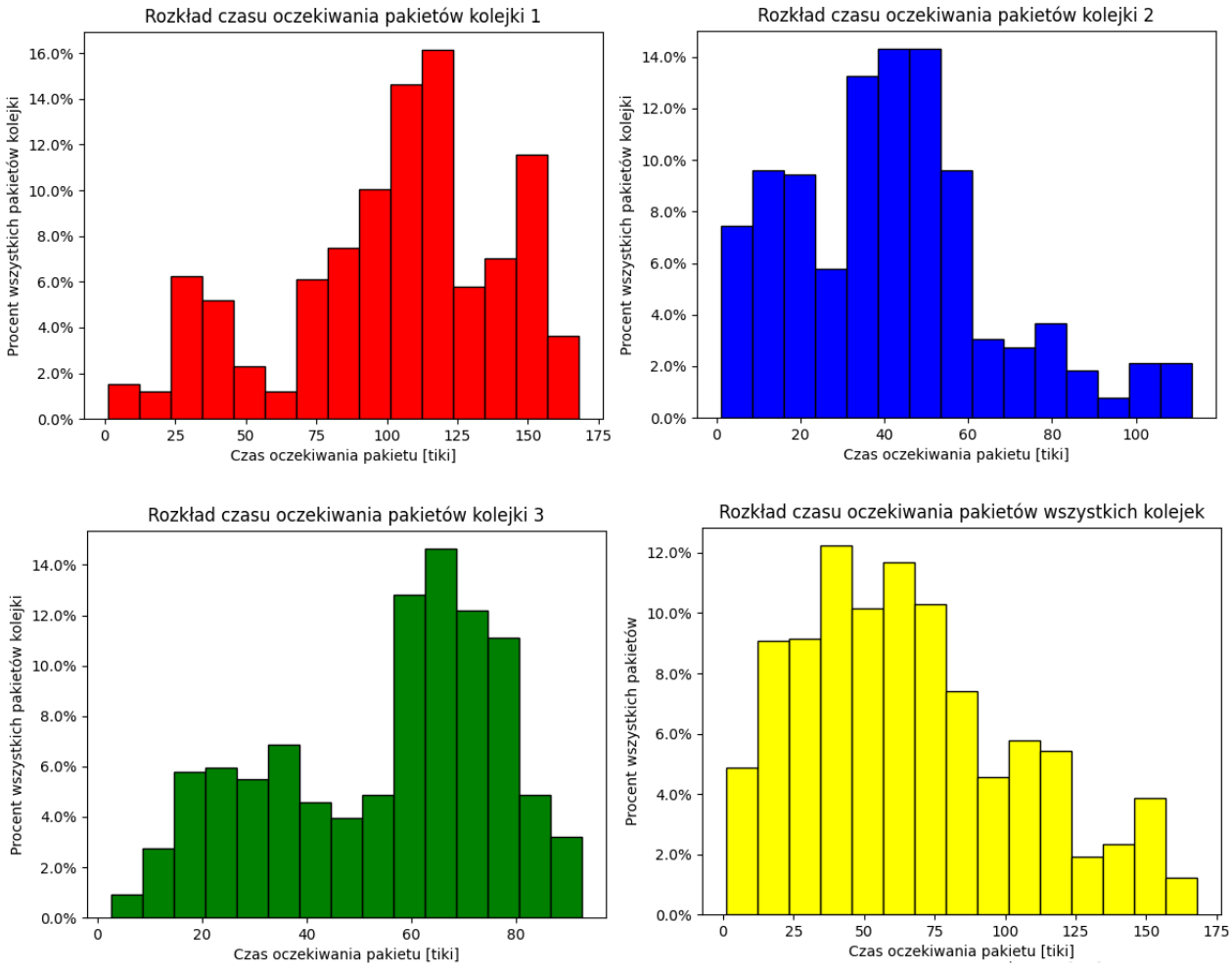


Rys. 3. Histogramy obsługi pakietów poszczególnych kolejek

Średni czas oczekiwania pakietu = 30.649
Średni czas oczekiwania pakietu kolejki 1 = 33.166
Średni czas oczekiwania pakietu kolejki 2 = 12.386
Średni czas oczekiwania pakietu kolejki 3 = 45.189

4.1.2. Lambda - 0.333, czas obsługi kolejki - 2 jednostki czasu

W przypadku, kiedy przychodzi więcej pakietów niż jesteśmy w stanie obsłużyć w jednostce czasu, to następuje przeciążenie.

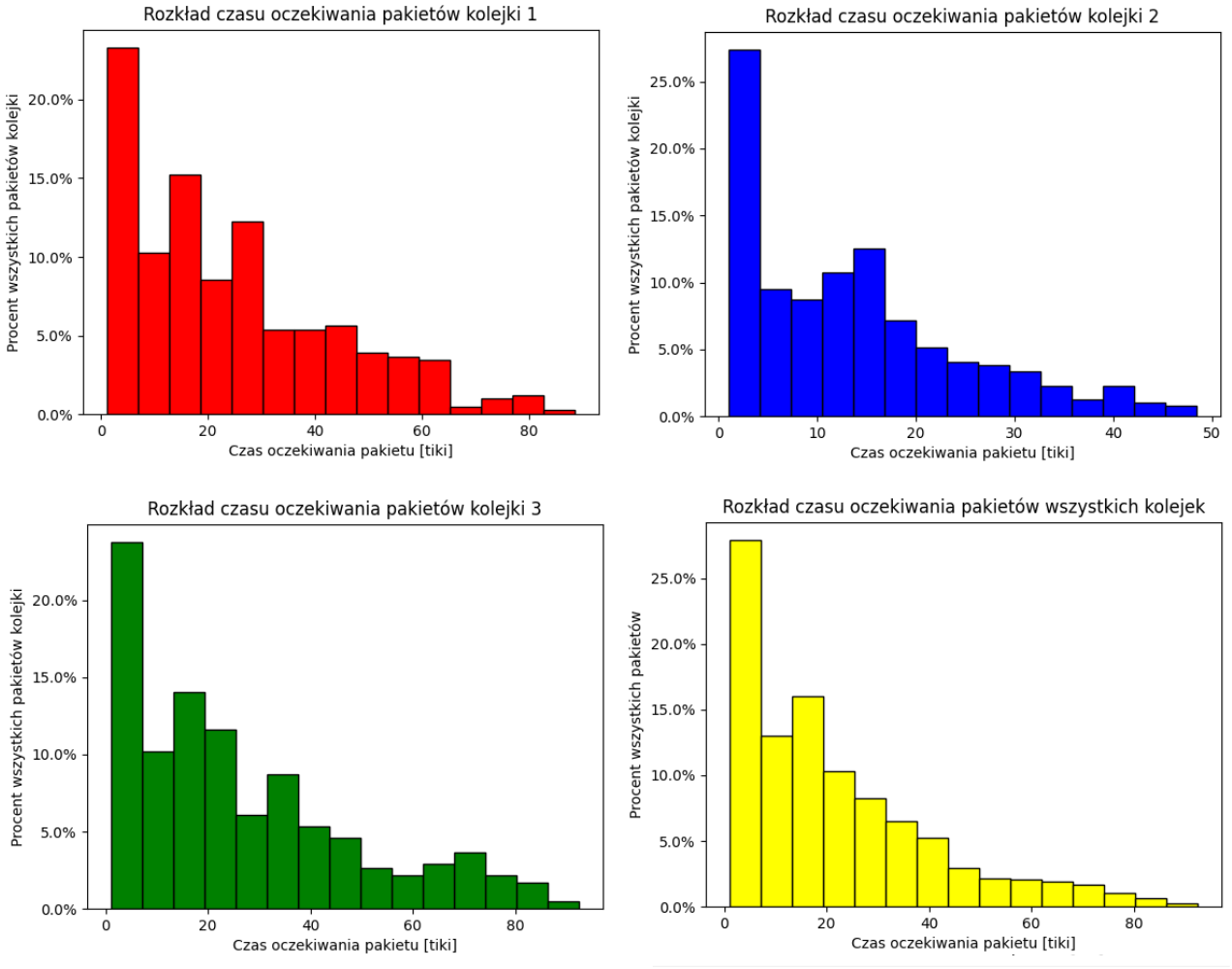


Rys. 4. Histogramy obsługi pakietów poszczególnych kolejek

Średni czas oczekiwania pakietu = 66.250
Średni czas oczekiwania pakietu kolejki 1 = 101.936
Średni czas oczekiwania pakietu kolejki 2 = 41.791
Średni czas oczekiwania pakietu kolejki 3 = 55.007

4.1.3. Lambda - 0.2, czas obsługi kolejki - 6 jednostek czasu

Czas trwania cyklu jest znacząco większy niż częstość przychodzenia pakietów.

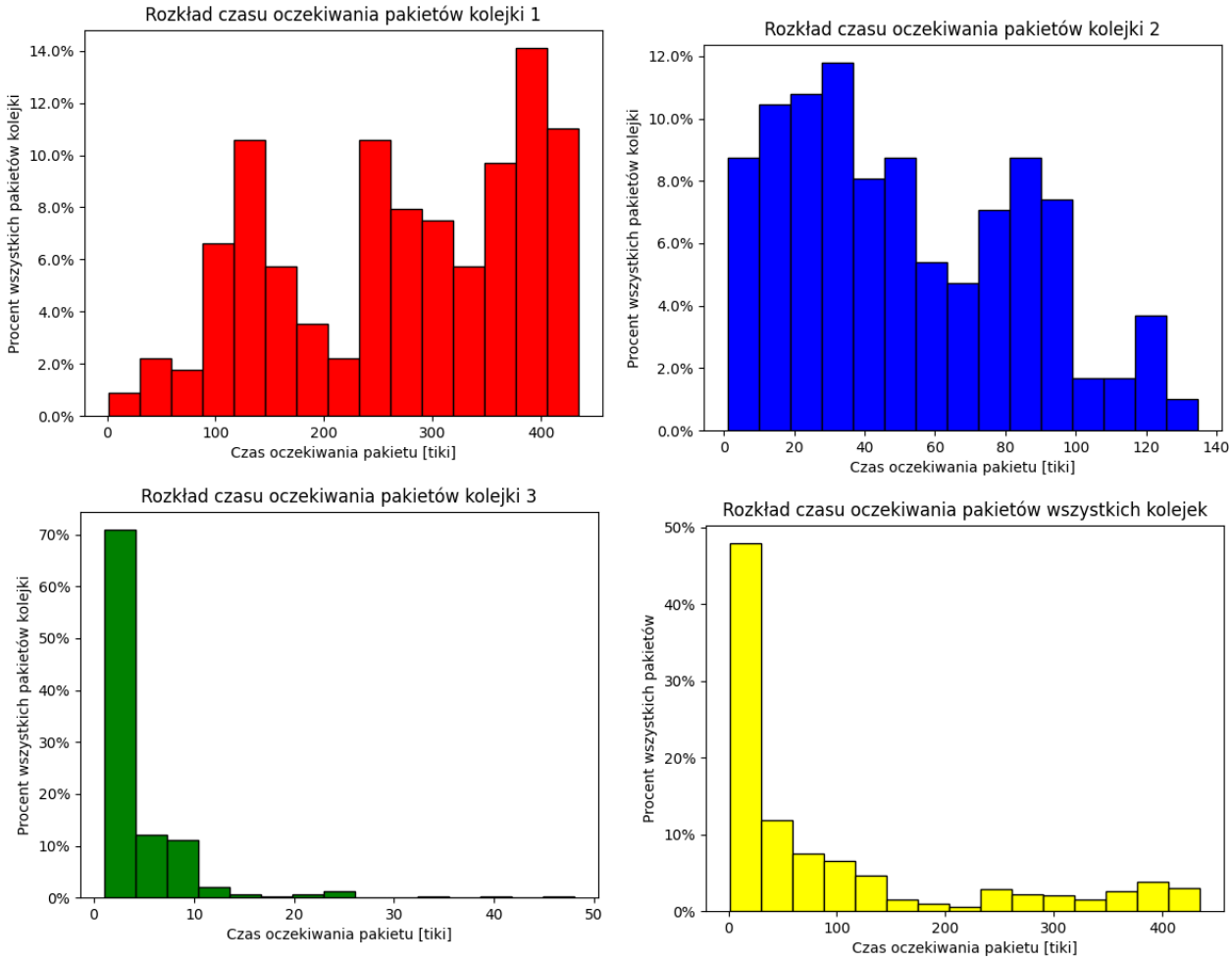


Rys. 5. Histogramy obsługi pakietów poszczególnych kolejek

Średni czas oczekiwania pakietu = 21.300
Średni czas oczekiwania pakietu kolejki 1 = 23.940
Średni czas oczekiwania pakietu kolejki 2 = 13.428
Średni czas oczekiwania pakietu kolejki 3 = 26.143

4.1.4. Lambda - 0.15, czas obsługi kolejki 1/2/3 - 2/3/5 jednostek czasu

W tym przypadku postanowiliśmy rozróżnić czas obsługi kolejek. Każda następna kolejka jest obsługiwana przez dłuższy czas.

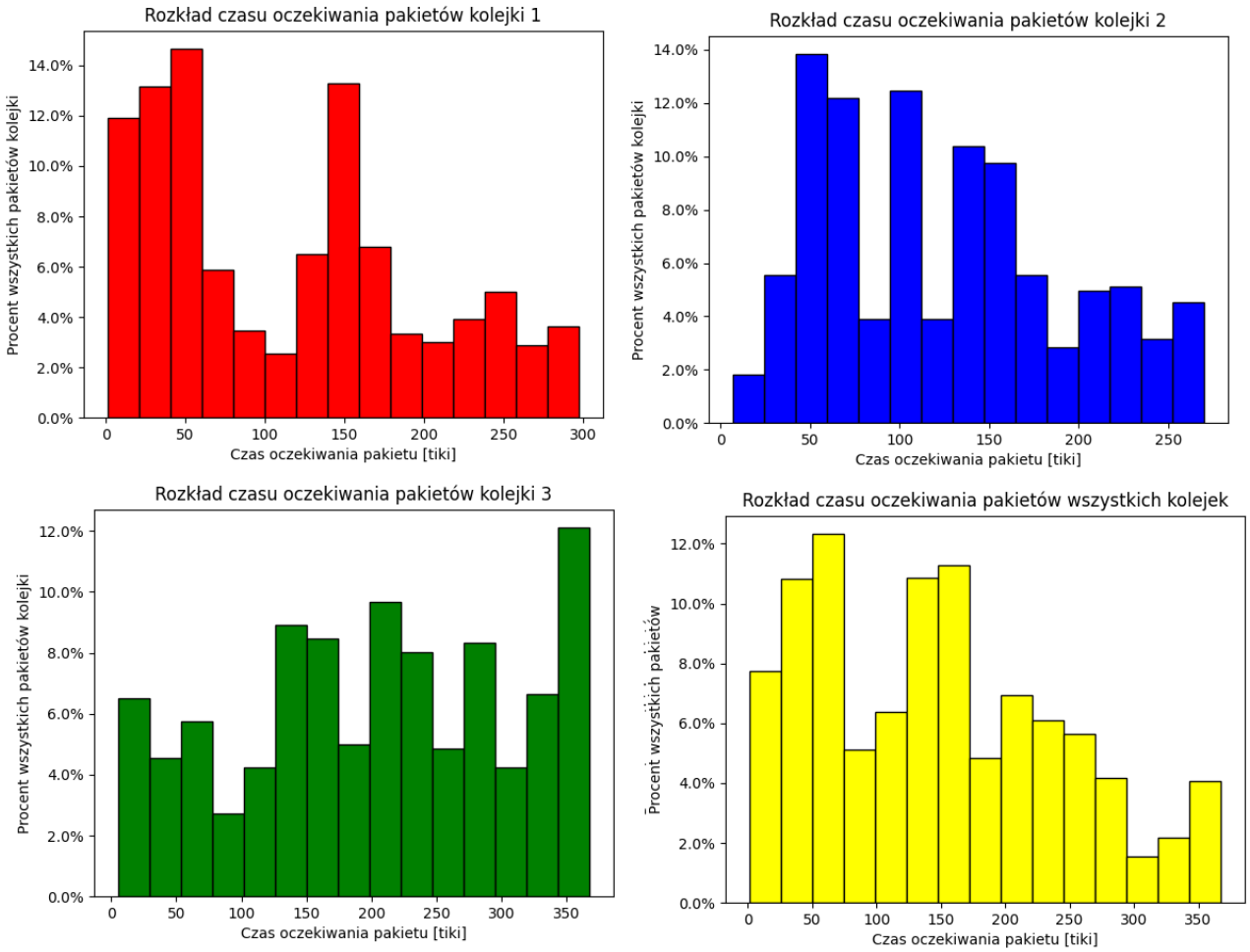


Rys. 6. Histogramy obsługi pakietów poszczególnych kolejek

Średni czas oczekiwania pakietu = 94.602
Średni czas oczekiwania pakietu kolejki 1 = 270.607
Średni czas oczekiwania pakietu kolejki 2 = 51.400
Średni czas oczekiwania pakietu kolejki 3 = 3.893

4.1.5. Lambda - 0.4, czas obsługi kolejki - 3 jednostki czasu

Tym razem spróbowaliśmy dać lambdę większą niż czas obsługi kolejki, przy czym suma lambd była większa od 1. Stało się to co przewidywaliśmy, czyli intensywność napływu pakietów była za duża i kolejka rosła w nieskończoność.

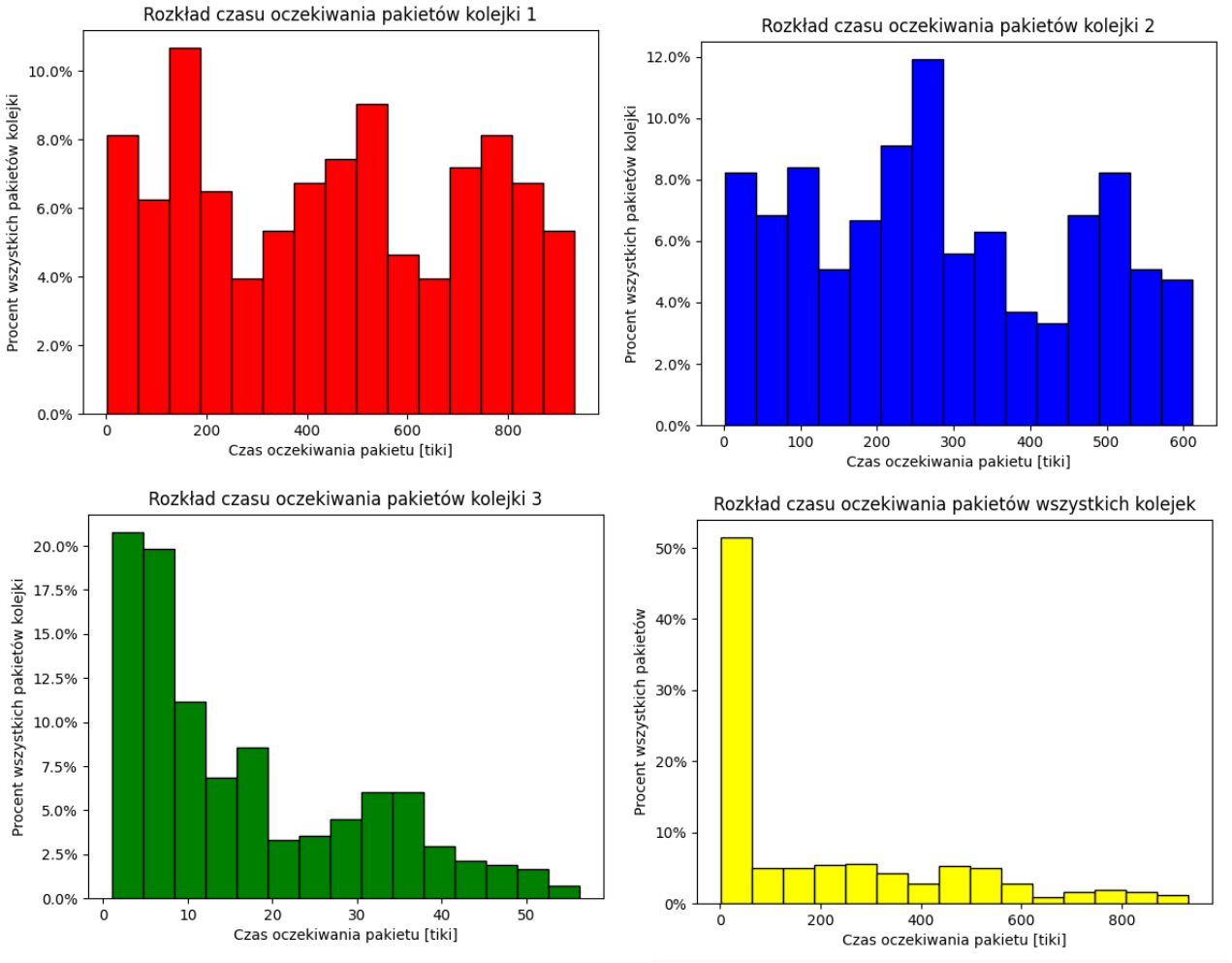


Rys. 7. Histogramy obsługi pakietów poszczególnych kolejek

Średni czas oczekiwania pakietu = 147.326
Średni czas oczekiwania pakietu kolejki 1 = 115.011
Średni czas oczekiwania pakietu kolejki 2 = 124.434
Średni czas oczekiwania pakietu kolejki 3 = 202.721

4.1.6. Lambda - 0.4, czas obsługi kolejki 1/2/3 - 2/3/5 jednostek czasu

W ostatnim teście chcieliśmy jeszcze sprawdzić co się sanie, kiedy lambda będzie mniejsza od czasu obsługi jednej kolejki, ale jednocześnie suma lambd będzie większa od 1. Z powodu ustawienia zbyt dużej lambdy, kolejki rosły w nieskończoność.



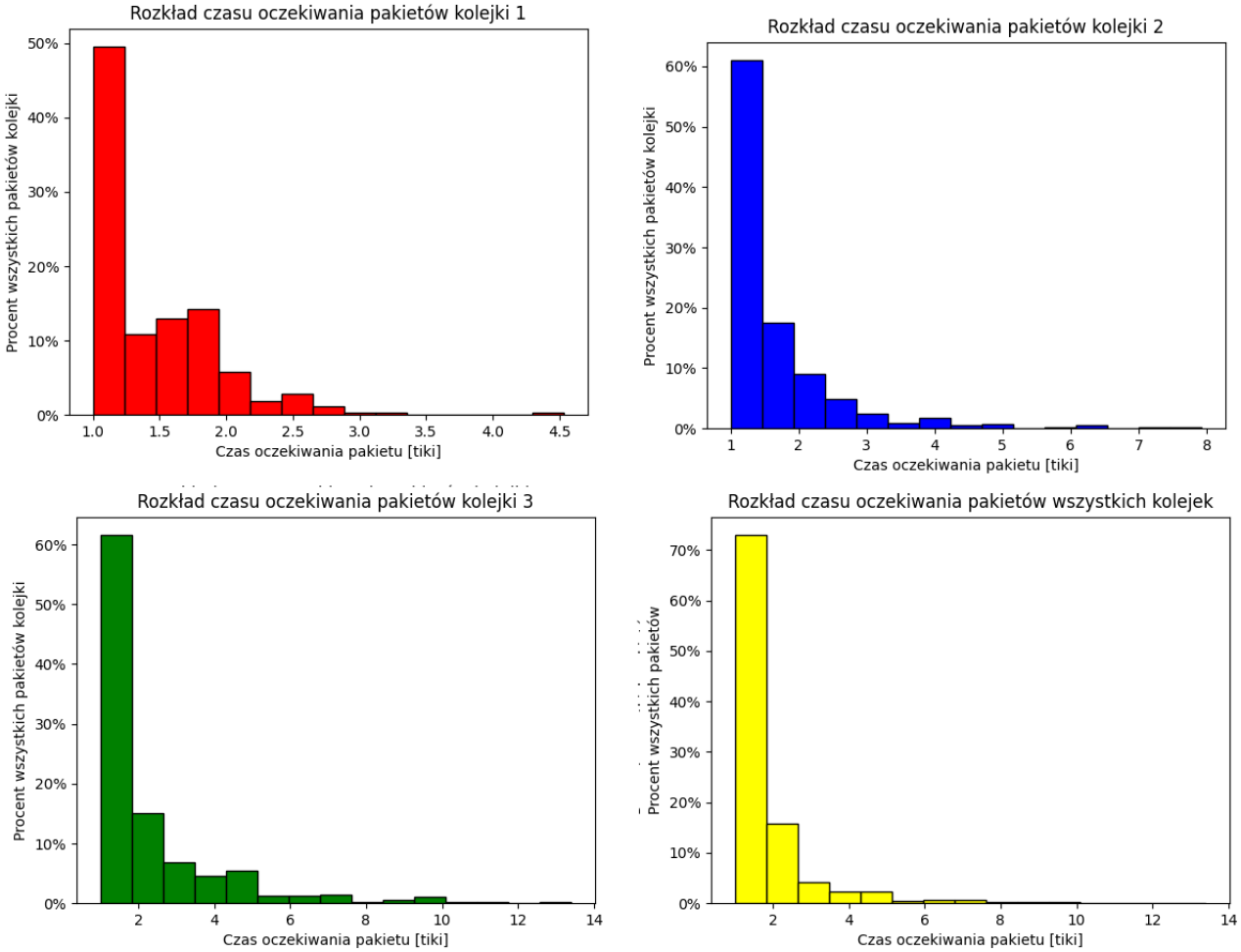
Rys. 8. Histogramy obsługi pakietów poszczególnych kolejek

Średni czas oczekiwania pakietu = 201.609
Średni czas oczekiwania pakietu kolejki 1 = 453.304
Średni czas oczekiwania pakietu kolejki 2 = 284.598
Średni czas oczekiwania pakietu kolejki 3 = 16.714

4.2. Algorytm szeregowania oparty na priorytetach

4.2.1. Lambda - 0.2

W pierwszym przypadku lambda była równa 0.2 i wszystko działało poprawnie. Czasy obsługi pakietów były bardzo krótkie, a w kolejkach z wyższymi priorytetami najkrótsze.

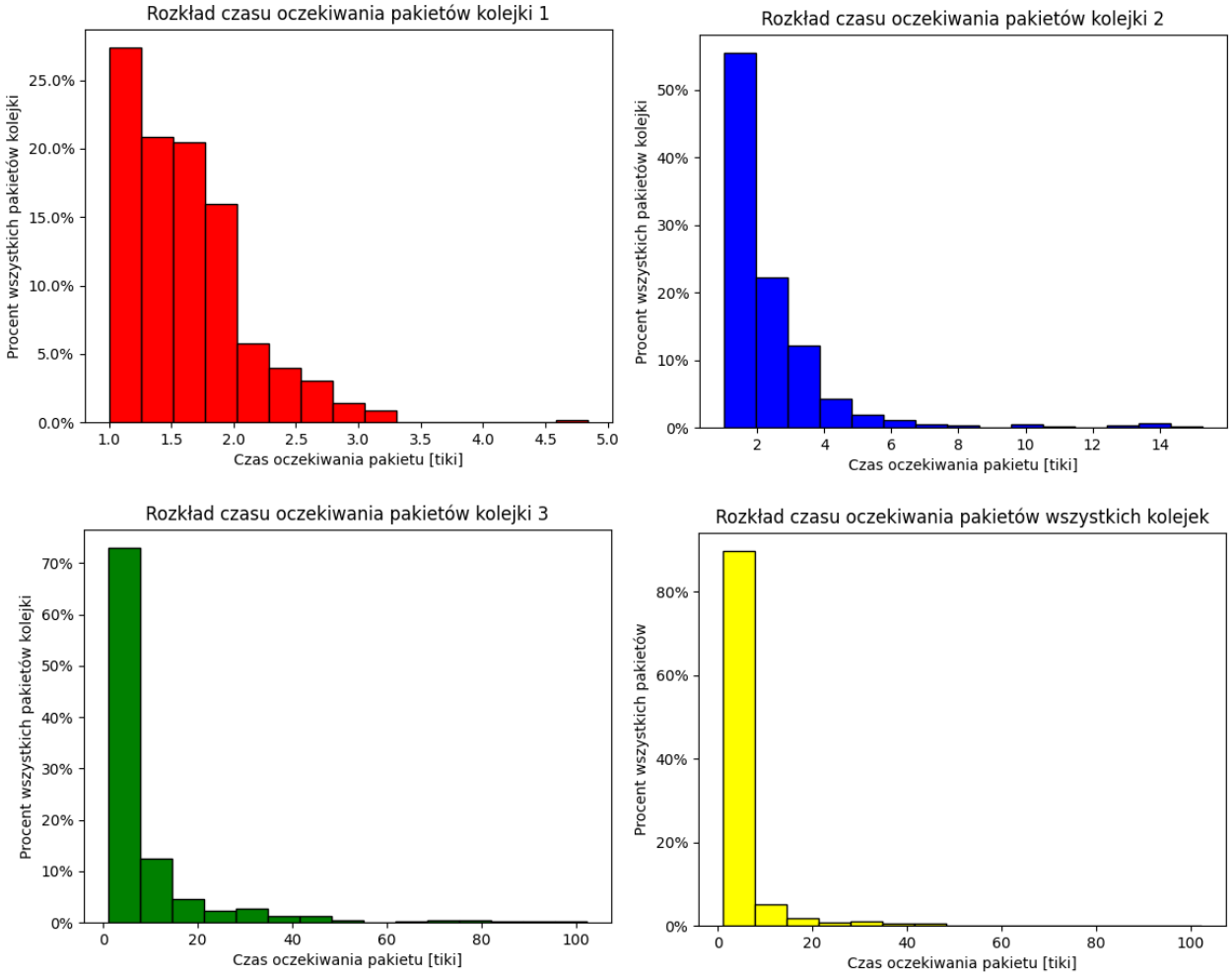


Rys. 9. Histogramy obsługi pakietów poszczególnych kolejek

Średni czas oczekiwania pakietu = 1.730
Średni czas oczekiwania pakietu kolejki 1 = 1.414
Średni czas oczekiwania pakietu kolejki 2 = 1.587
Średni czas oczekiwania pakietu kolejki 3 = 2.206

4.2.2. Lambda - 0.3

Zwiększenie ilości żądań generowanych na sekundę zwiększyło czas obsługi pakietów. Uwydatniło to różnice w czasach czekania pakietów, pomiędzy kolejkami z różnymi priorytetami.

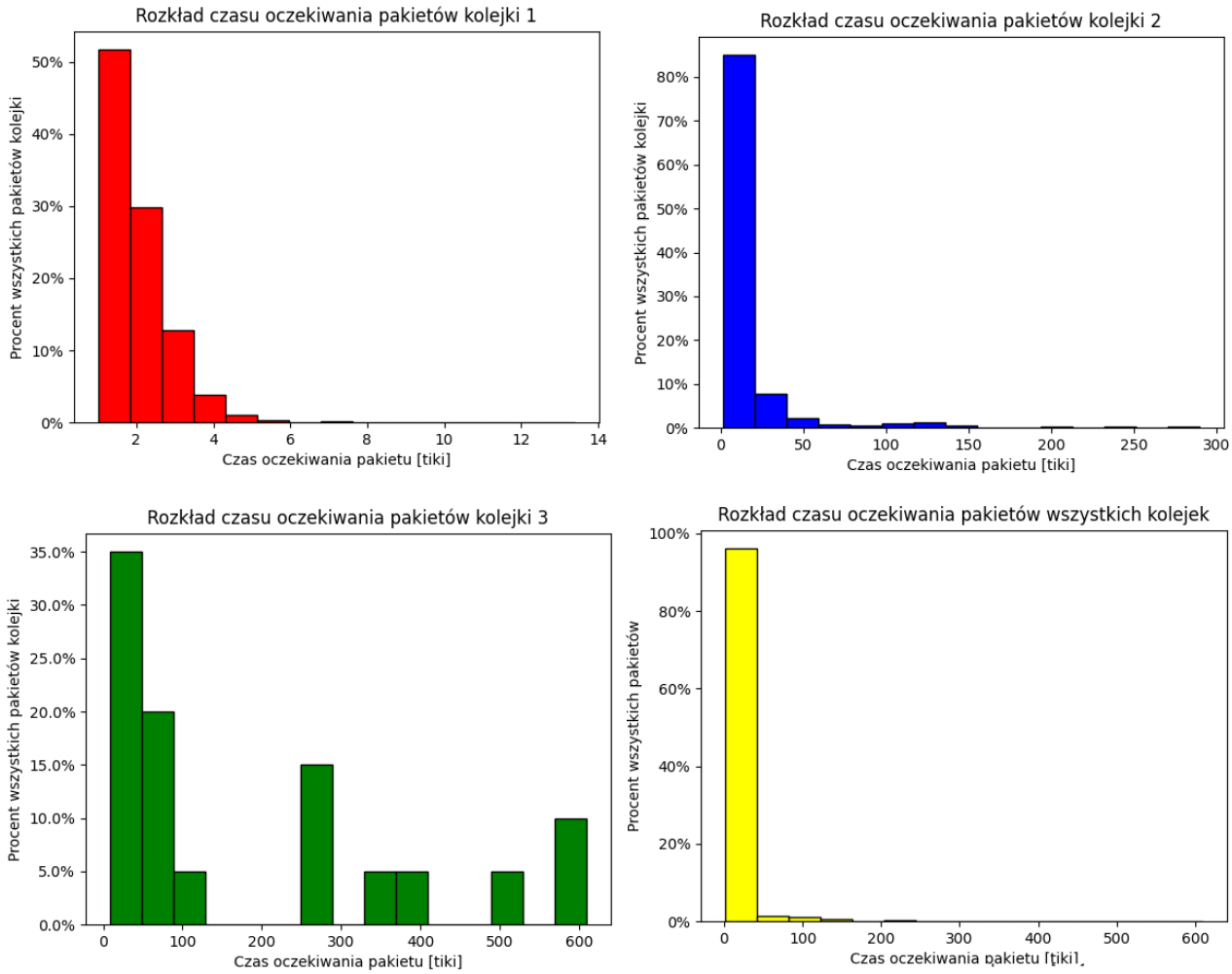


Rys. 10. Histogramy obsługi pakietów poszczególnych kolejek

Średni czas oczekiwania pakietu = 4.364
Średni czas oczekiwania pakietu kolejki 1 = 1.607
Średni czas oczekiwania pakietu kolejki 2 = 2.338
Średni czas oczekiwania pakietu kolejki 3 = 8.750

4.2.3. Lambda - 0.5

Dla tak wysokiej lambdy różnice zrobiły się jeszcze bardziej widoczne. Kolejka tworzy się nieskończenie długa i praktycznie nie ma możliwości obsługi pakietów z kolejki z najmniejszym priorytetem.



Rys. 11. Histogramy obsługi pakietów poszczególnych kolejek

Średni czas oczekiwania pakietu = 11.101
Średni czas oczekiwania pakietu kolejki 1 = 2.021
Średni czas oczekiwania pakietu kolejki 2 = 16.936
Średni czas oczekiwania pakietu kolejki 3 = 188.373

5. Podsumowanie

Wyniki jasno wskazują na to, że średni czas oczekiwania pakietu w kolejce jest mocno zależny od wybranego algorytmu, a przede wszystkim od użytych parametrów. W przypadku kolejki opartej na priorytetach przy częstym występowaniu oddzielnych zdarzeń przesył pakietów dalej jest możliwy. Jednak przy zwiększaniu parametru lambda, opóźnienia w kolejkach o niższym priorytecie drastycznie wzrastają. Algorytm szeregowania pakietów oparty na cyklu jest trudniejszy do efektywnego wykorzystania, gdyż nie jest on algorytmem typu work conserving. Nieumyślne dopasowanie parametrów algorytmu może skutkować tym, że żadna z kolejek nie będzie miała zapewnionej jakości przesyłu. W związku z powyższym algorytm szeregowania pakietów oparty na priorytetach wydaje się być efektywniejszym w większej ilości zastosowań.

6. Bibliografia

Materiały wykładowe z przedmiotu SWUS
https://www.itl.nist.gov
https://pl.wikipedia.org/wiki/Rozkład_Poissona