Symulacja Cyfrowa – Projekt
Raport Końcowy
Jakub Siejak, MiEPU, nr indeksu:------
Metoda Planowania zdarzeń, Algorytm Maximum Throughput(a)

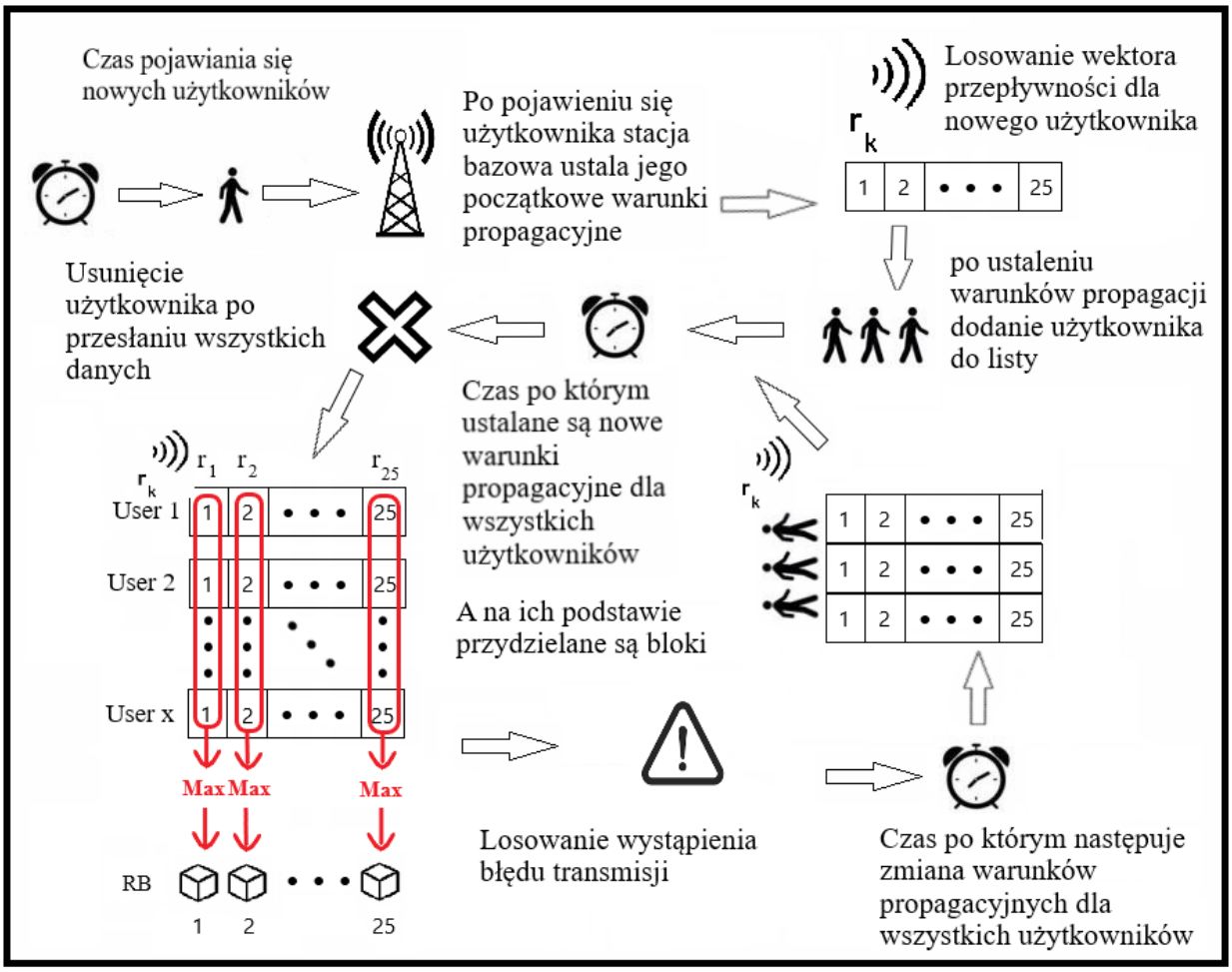
Pełny tekst rozwiązywanego zadania

Treść zadania

Rozważmy łącze w dół pojedynczej komórki sieci mobilnej, w której znajduje się stacja bazowa dysponująca pasmem podzielonym na k bloków zasobów RB (ang. *Resource Block*). W losowych odstępach czasu t w obszarze komórki pojawiają się użytkownicy, z których każdy odbiera losową ilość danych d. W chwili pojawienia się nowego użytkownika są ustalane jego warunki propagacyjne, wyrażone jako przepływność r_k losowana niezależnie dla k kolejnych bloków zasobów. Warunki propagacyjne dla każdego użytkownika zmieniają się w losowych odstępach czasu τ. W stałych odstępach czasu s stacja bazowa przydziela użytkownikom bloki zasobów zgodnie z algorytmem **A** podanym w Tabeli 2**.** Użytkownikowi można przydzielić jednocześnie nie więcej niż l bloków zasobów. Dodatkowo każda transmisja może zakończyć się błędem z prawdopodobieństwem ε**.** W takim przypadku dane należy przesyłać, aż do momentu, kiedy zostaną poprawnie odebrane. Po odebraniu wszystkich danych, użytkownik jest usuwany z systemu.
Opracuj symulator sieci bezprzewodowej zgodnie z metodą **M** (Tabela 1).

- Za pomocą symulacji wyznacz intensywność zgłoszeń **λ**, która zapewnia średni czas oczekiwania nie większy niż 50 ms, oraz:
  - średnią przepływność systemu mierzoną liczbą danych przesłanych poprawnie przez stację bazową w jednostce czasu,
  - średnią przepływność użytkownika mierzoną liczbą danych odebranych poprawnie od stacji bazowej w jednostce czasu, uśrednioną po wszystkich użytkownikach sieci; ta wartość uwzględnia okresy kiedy użytkownik nie miał przydzielonego żadnego bloku zasobów,
  - średni czas oczekiwania, tzn. czas między pojawieniem się użytkownika w sieci, a zakończeniem odbierania danych,
  - histogram średnich przepływności dla wszystkich użytkowników sieci.
- Sporządź wykres zależności średniej przepływności systemu oraz średniego czasu oczekiwa-nia w funkcji wartości **λ**.
- Porównaj wyniki z osobą implementującą inny algorytm przydziału zasobów, z inną maksymalną liczbą bloków zasobów przypisaną pojedynczemu użytkownikowi l, lub z inną liczbą dostępnych bloków zasobów k.

Krótki opis modelu symulacyjnego



Schemat 1: Schemat modelu symulacyjnego

Opis klas wchodzących w skład systemu i ich atrybutów

Nazwa obiektu	Nazwa klasy implementującej	Opis	Atrybuty
Sieć	Network	Klasa reprezentująca właściwości całej sieci. Są tutaj uwzględnione stałe elementu definiujące sieć. Wywoływane są w niej metody poszczególnych klas zdarzeń. Zaimplementowane są w niej także funkcje zliczające statystyki	Network(int seed_t, int seed_d, int seed_rk, int seed_tau, int seed_e) void Init(int) void Generateuser(int, string, int, int, int) void ResourceAllocation(int, string, int, int, int) void Function_block_allocation(User*, size_t, size_t, size_t, int, bool, int, string, int, int) void NewPropagation(int, string, int, int, int) void Remove(int, string, int, int, int) void ser_generate_timer(size_t time) size_t get_buffer_size() vector<double> get_asbt() void set_asbt(vector<double> new_asbt) vector<double> get_aubr() void set_aubr(vector<double> new_aubr) vector<int> get_uat() void set_uat(vector<int> new_uat) int get_nous() int get_nousu() int get_noe() int get_noeu() int get_nora() int get_norau()

			vector<double> get_hw() void set_hw(vector<double> new_hw) vector<int> get_hr() void set_hr(vector<int> new_hr) vector<bool> get_hzj() void set_hw(vector<bool> new_hzj) Generators* generator_t Generators* generator_d Generators* generator_rk Generators* generator_tau Generators* generator_e list<User*> buffer_ list<User*>::iterator it list<User*> buffer_temp_ int generate_timer double probability int RB int l_blocks vector<double> avg_system_bit_rate_ vector<double> avg_user_bit_rate_ vector<int> user_awaiting_time_ int number_of_users_served int number_of_users_served_unstabilized int number_of_errors int number_of_errors_unstabilized int number_of_resources_allocations int number_of_resources_allocations_unstabilized vector<int> hist_r vector<double> hist_w vector<bool> hist_zj
Stacja bazowa	BaseStation	Klasa reprezentująca stację bazową, zawiera informacje na temat użytkowników. Posiada metodę umożliwiającą dodanie ich do wektora oraz usunięcie po odebraniu wszystkich pakietów. Klasa ta nie została rozwinięta ani wykorzystana aby zbytnio nie komplikować programu. Klasa jednak została pozostawiona gdyż byłaby przydatna podczas utworzenia więcej niż jednej stacji bazowej.	unsigned int number_of_users_ – liczba użytkowników w stacji bazowej, vector<User*> user_list_ – spis użytkowników stacji bazowej

Użytkownik	User	Klasa reprezentująca użytkownika o odpowiednim numerze, posiada informacje o ilości danych do przesłania, aktualne przepływności, oraz jakie bloki są do niego przypisane. Posiada zaimplementowane metody umożliwiające odczytywanie i zmianę parametrów konkretnych użytkowników	User(int d, vector<int> rk, double ar, size_t id, int ubn, bool uaf, User *next) User* next_ // wskaźnik do sortowania size_t get_id() const size_t get_rk(size_t RB_id) const int get_d() const void set_d(size_t new_d) int get_ubn() void set_ubn(size_t new_ubn) void set_rk(vector<int> new_rk) double get_ar() void set_ar(double new_ar) bool get_uaf() void set_uaf(size_t new_uaf) size_t get_at() void set_at(size_t new_at) size_t get_rt() void set_rt(size_t new_rt) const size_t user_id_ // unikalny numer użytkownika int data_size_d_ // ilość danych do odebrania przez użytkownika vector<int> bit_rate_ // wektor przepływności dla wszystkich bloków int user_blocks_number_ // ilość bloków przydzielona obecnie użytkownikowi vector<ResourceBlock*> user_blocks_ // wektor bloków posiadanych aktualnie przez użytkownika double avg_rk_ // średnia przepływność obecną przepływność dodajemy gdy użytkownik uzyskał blok zasobów bool user_avg_flag_ // flaga pomagająca ustalić czy użytkownik miał już policzoną średnią w danej pętli int arrive_time_ int remove_time_
Blok zasobów	ResourceBlock	Klasa reprezentująca blok o określonym numerze, posiada informacje do którego użytkownika dany blok jest przypisany, przepływność wynikającą z tego przypisania oraz czy w obecnej chwili wystąpił błąd. Podobnie jak klasa BaseStation nie była modyfikowana ze względu na brak przydatności w postaci ułatwienia w tworzeniu symulacji. Została jednak pozostawiona jako ważny element który byłby niezbędny podczas rozwoju symulacji o kolejne stacje bazowe.	ResourceBlock(size_t RB_id, size_t u_id, bool e, size_t rk) size_t RB_id_ // numer id bloku size_t user_id_ // numer id aktualnego właściciela bloku bool error_ // aktualna informacja o wystąpieniu błędu w danym bloku size_t bit_rate_ // aktualna przepływność danego bloku
Symulator	Simulator	Klasa odpowiadająca za działanie symulacji, posiada listę zdarzeń do której zapisuje występujące zdarzenia. W klasie tej zaimplementowana jest pętla główna symulacji.	explicit Simulator(Network* network) void Simulation(int time, int tryb, int view, int initial_phase, int end, int max_user_served) size_t allocation_time size_t clock_ Network* network_ string c_clock double lambda_tau double lambda

Zdarzenie	Event	Klasa implementująca zdarzenia oraz listę zdarzeń umożliwiając ich przeglądanie i odpowiednie ułożenie w czasie.	typedef priority_queue<Event*, vector<Event*>, function<bool(Event*, Event*)>> EventList Event(size_t time, Network* network) virtual void Execute(int view, string c_clock, int ms, int time, int initial_phase) size_t get_time() size_t time_ Network* network_
Zdarzenie zmiany propagacji	ChangeOfPropagationEvent	Klasa implementująca zdarzenie zmiany propagacji	ChangeOfPropagationEvent(size_t time, Network* network, EventList* event_list); void Execute(int view, string c_clock, int ms, int time, int initial_phase) override EventList* event_list_ const size_t propagation_time
Zdarzenie przydziału bloków zasobów	ChangeOfUserRBEvent	Klasa implementująca zdarzenie alokacji zasobów	ChangeOfUsersRBEvent(size_t time, Network* network, EventList* event_list) void Execute(int view, string c_clock, int ms, int time, int initial_phase) override EventList* event_list_ const size_t allocation_time
Zdarzenie pojawienia się użytkownika	UserArriveEvent	Klasa implementująca zdarzenie pojawienia się nowego użytkownika	UserArriveEvent(size_t time, Network* network, EventList* event_list) void Execute(int view, string c_clock, int ms, int time, int initial_phase) EventList* event_list_ const size_t max_generate_user
Generatory	Generators	Klasa implementująca try rodzaje generatorów pseudolosowych	double Rand() int Rand(int min, int max) int RandExp(double lambda) int RandZeroOne(double probability) int get_kernel() int kernel_ const double kM static const int kA static const int kQ static const int kR

Tabela 1: Opis klas systemu i ich atrybutów

Opis przydzielonej metody symulacyjnej

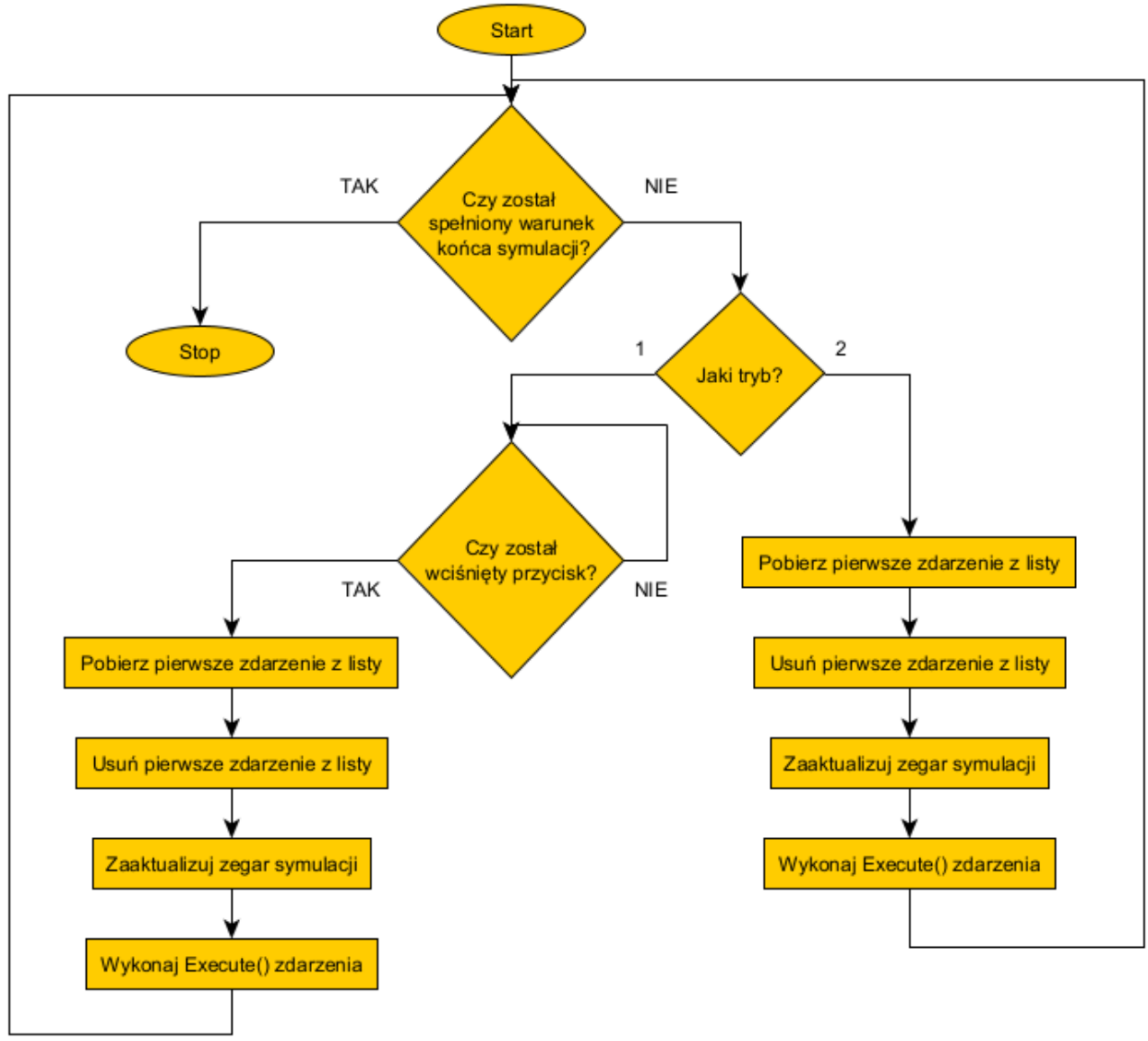
Metoda charakteryzująca się uporządkowaniem zdarzeń priorytetowo pod kątem ich wystąpienia w czasie – dziennik zdarzeń (który jest niczym innym jak Agendą w interakcji procesów). Każdy wpis w liście zdarzeń charakteryzuje się co najmniej absolutnym wskaźnikiem czasu oraz odwołaniem do procedury, która ma zostać wykonana po jego wystąpienia. Lista zdarzeń zawiera tylko implementację obsługi zdarzeń czasowych, przez co zdarzenia te koniecznie muszą implementować wewnątrz siebie obsługę możliwych do wystąpienia zdarzeń warunkowych, które wpłyną na zmian stanu systemu.

Zalety:
- Wysoka efektywność,
- Nie jest konieczne sprawdzanie warunków wykonania wielu zdarzeń, ponieważ zdarzenia są planowane priorytetowo,

Wady:
- Brak jawnej obsługi zdarzeń warunkowych, bo planowane są tylko zdarzenia czasowe,
- Wymaga użycia zaawansowanych struktur danych i algorytmów w celu utworzenia i odpowiedniego zarządzania listą zdarzeń,

Wniosek:
Jest to metoda odpowiednia dla systemów z duża liczbą zdarzeń czasowych oraz o niskim stopniu skomplikowania.

Schemat blokowy pętli głownej



Schemat 2: Schemat blokowy pętli głównej

Lista zdarzeń czasowych i warunkowych

Zdarzenia czasowe:

Zdarzenie	Warunek	Algorytm
Pojawienie się nowego użytkownika	W odstępach czasu t	Nowi użytkownicy pojawiają się w odstępach czasu t, następuje ustalenie jego warunków propagacyjnych r_k oraz przydzielenie bloków zasobów RB jeśli są one dostępne
Ustalanie nowych warunków propagacyjnych dla użytkowników	W odstępach czasu τ	Dla każdego użytkownika ustalane są nowe warunki propagacyjne r_k co odstęp czasu τ
Przydzielanie bloków zasobów w odstępach	W odstępach czasu s	Co odstęp czasu s następuje przydział bloków zasobów poprzedzony ustaleniem nowych warunków propagacyjnych dla każdego użytkownika na tej podstawie użytkownicy z największą przepływnością i ilością już przypisanych bloków zasobów nie większą niż l otrzymują do wykorzystania odpowiedni blok zasobów. Jeśli dwóch użytkowników ma uzyskało taką samą przepływność algorytm oblicza dotychczasową średnią przepływność i blok zasobów przydziela użytkownikowi z niższą średnią

Tabela 2: Tabela zdarzeń czasowych

Zdarzenia warunkowe:

Zdarzenie	Warunek	Algorytm
Retransmisja	Pod warunkiem pojawienia się informacji o błędzie transmisji	Jeśli podczas wysyłania danych nastąpi błąd I zostanie to zasygnalizowane przez odpowiednią flagę, wtedy należy uznać, że cała transmisja trwająca 5ms nie została poprawnie przesłana a zatem należy uznać wartość przesłanych danych jako 0

		bitów
Usunięcie użytkownika	Pod warunkiem odebrania przez niego wszystkich danych	Gdy wartość zmiennej odpowiadającej za ilość danych pozostałych do odebrania będzie równa 0 bitów, należy usunąć użytkownika z wektora oraz ze stacji bazowej

Tabela 3: Tabela zdarzeń warunkowych

Parametry wywołania programu

Po uruchomieniu programu od użytkownika zostaną pobrane dane określające konkretną symulację, pobieranie będzie następować sekwencyjnie a pobierane dane będą zależeć od poprzednich odpowiedzi.

Zapytanie	Zmienna	Opis
Choose end condition of simulation:	end	Wybór warunku końca symulacji
Enter simulation time in ms:	simulation_time_	Wybór czasu końca symulacji
Enter number of users to be served:	max_user_served_	Wybór ilości obsłużonych użytkowników przed zakończeniem symulacji
Enter time of initial phase in simulation in ms:	initial_phase	Wybór czasu trwania fazy początkowej
Choose mode of simulation:	mode	Wybór metody pracy symulacji
Choose displayed information:	view	Wybór ilości wybieranych informacji
Choose set of seeds from 1 – 10:	set_of_seeds	Wybór zestawu ziaren
Enter lambda:	lambda_	Wybór lambdy (intensywność zgłoszeń)

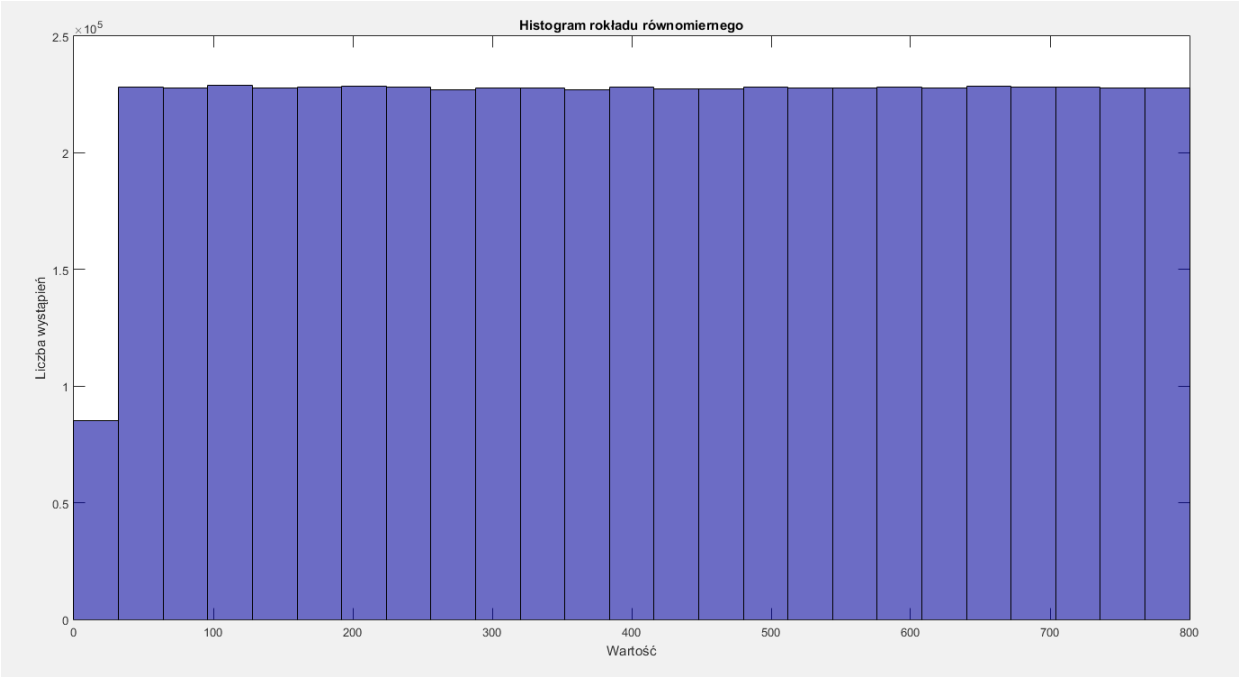
Tabela 4: Tabela parametrów wywołań

Generatory

Opis zastosowanych generatorów liczb losowych z histogramami

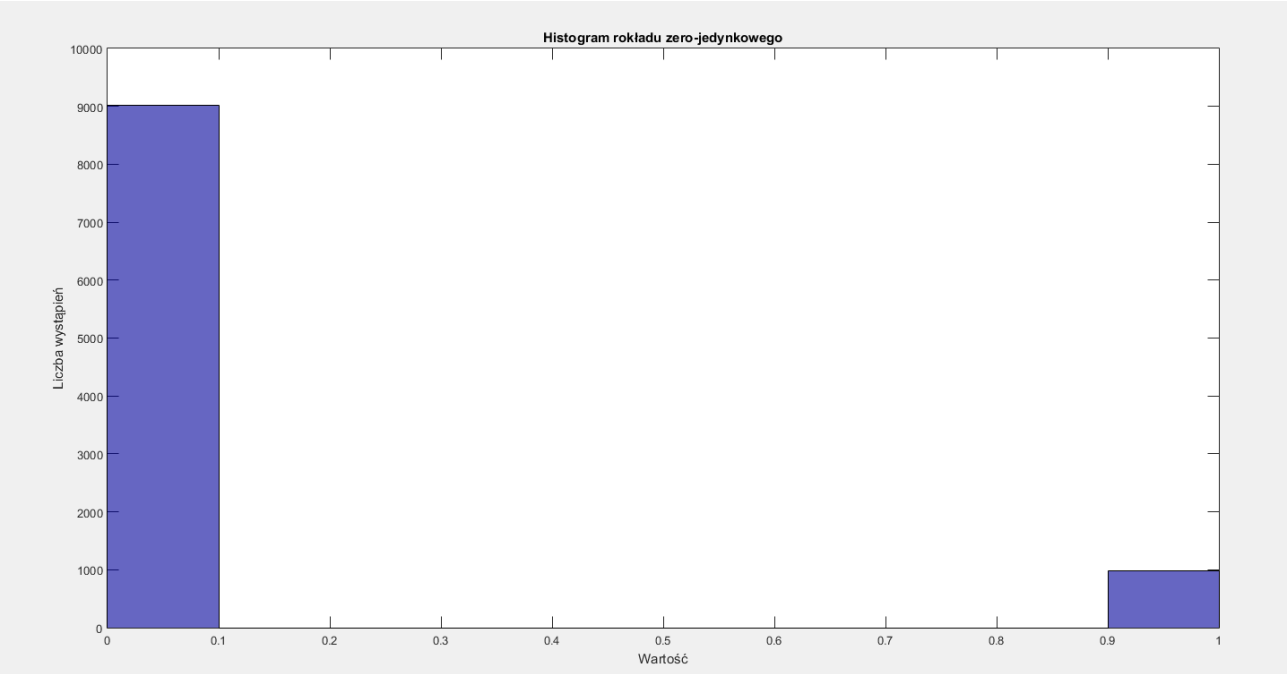
W symulacji zostały wykorzystane trzy typy generatorów:

Generator dla rozkładu równomiernego – generator ten losuje wartości, które są do siebie zbliżone pod względem liczby wystąpień, możemy określić zakres tego rozkładu, umożliwia to nam generowanie zadanych wartości pseudolosowych. Jest on wykorzystywany do wyznaczenia **d** oraz $\mathbf{r_k}$.



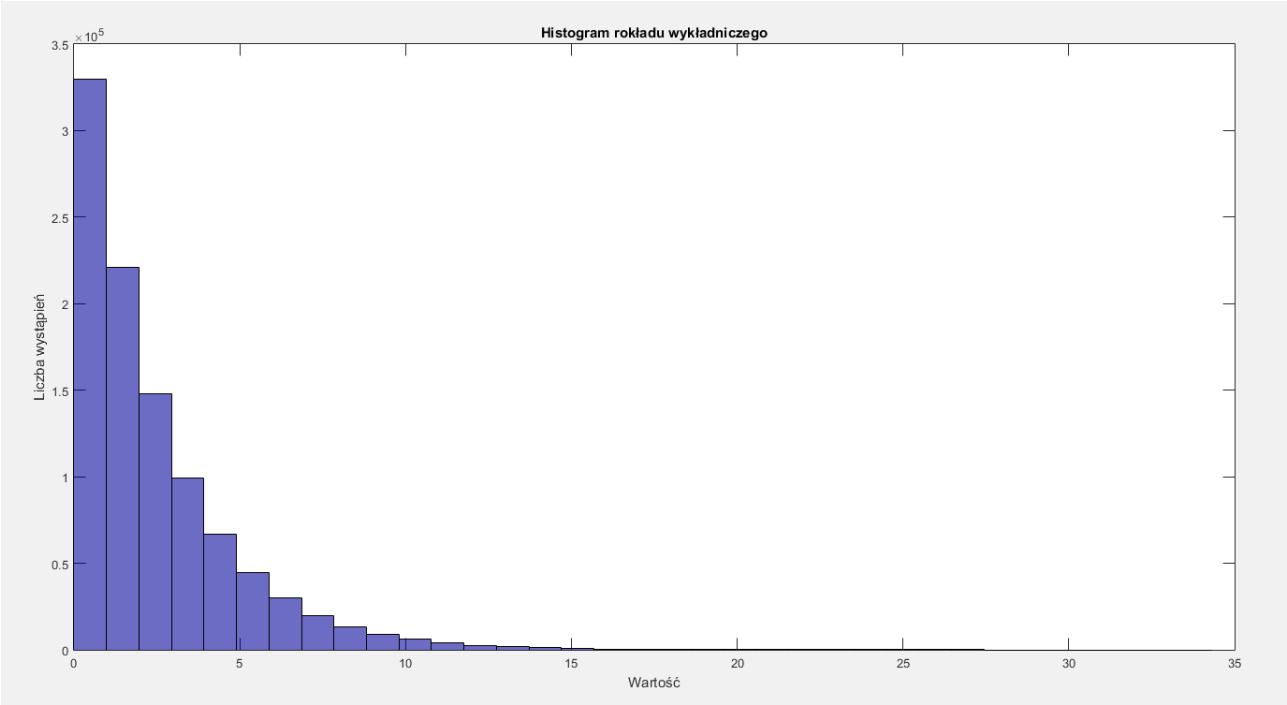
Histogram 1: Histogram rozkładu równomiernego

Generator dla rozkładu zero jedynkowego – generator ten losuję z zadanym prawdopodobieństwem wystąpienie „zera” lub „jedynki” w danym zdarzeniu, jest on wykorzystywany do generowania błędu **ε** – błędu odebrania danych,



Histogram 2: Histogram rozkładu zero-jedynkowego

Generator dla rozkładu wykładniczego – generator ten losuje wartości w sposób wykładniczy, im większa wartość tym maleje liczba jej wystąpień. Generator ten generuje dane wartości z intensywnością lambda λ. Użyty został do generowania losowego czasu **t** oraz **τ**.



Histogram 3: Histogram rozkładu wykładniczego

Sposób zapewnienia niezależność sekwencji losowych w różnych symulacjach

Niezależność sekwencji losowych została zapewniona poprzez wylosowanie różnych ziaren z odstępem co 100 000 losowań dla każdego zestawu symulacji. Generowanie z takim odstępem ziarna zapewniają nam możliwość przeprowadzenia niezależnych między zestawami oraz wiarygodnych badań danego zestawu parametrów. Ziaren wygenerowano 50, liczba ta wynika z generowania ziaren dla trzech rozkładów: Równomiernego (Rand()), Zero jedynkowego (RandZeroOne), Wykładniczego (RandExp), w symulacji istnieje 5 zmiennych, których wartości muszą być generowane pseudolosowo, liczba 50 wynika też z ilości symulacji (10). Ziarna przechowywane są w pliku tekstowym ziarna.txt.

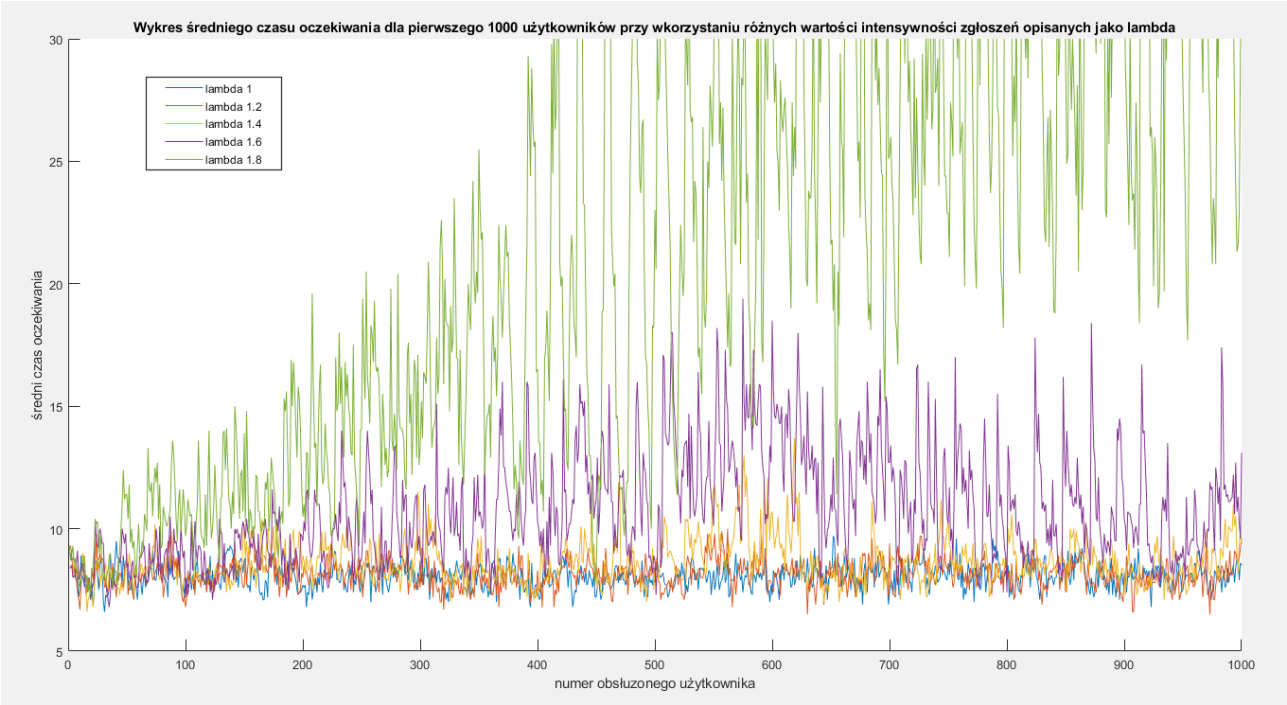
Metoda weryfikacji poprawności implementacji symulatora:

Zgodnie z postanowieniem metoda weryfikacji poprawności została zmieniona z metody uwzględniającej wyniki uzyskane przez innych studentów prowadzących równlegle badania na własnych symulatorach, na metodę samo-sprawdzenia za pomocą bardzo dokładnego wyświetlania danych przy użyciu loggera. W tym przypadku logger został stworzony przeze mnie i jest on częścią Network.cpp.

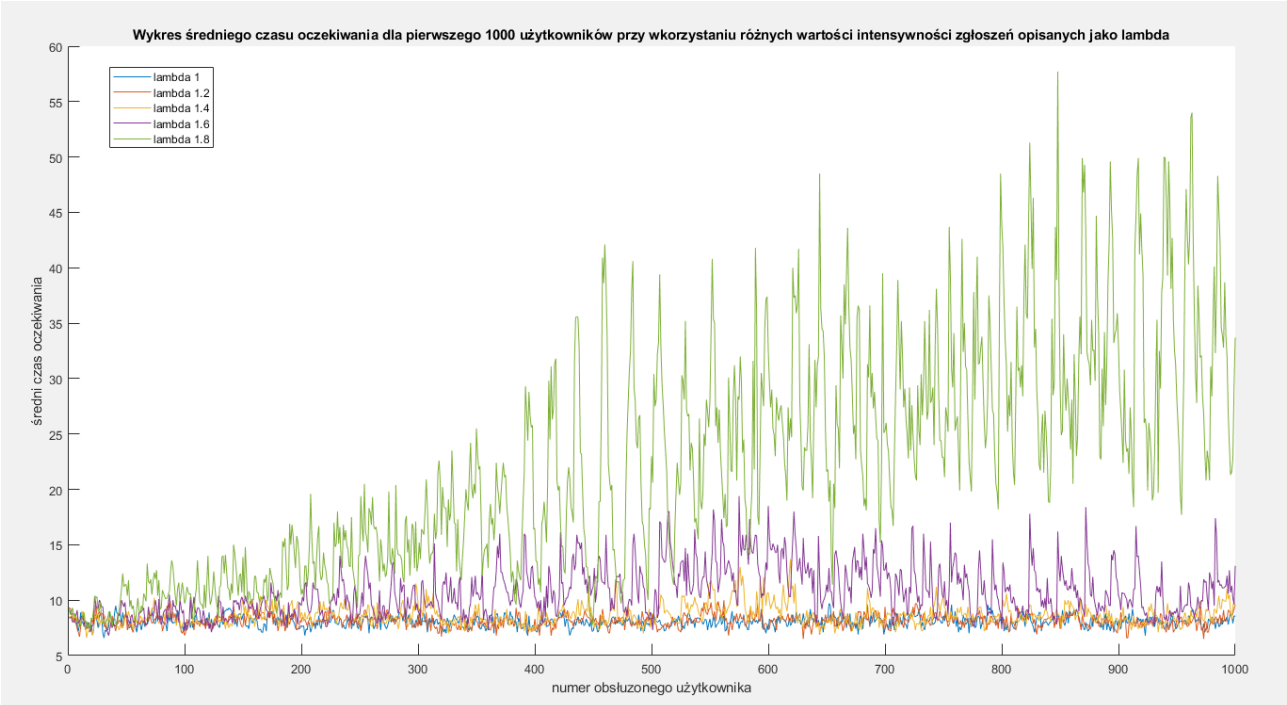
Wyniki symulacji

Wyznaczenie długości fazy początkowej

Pierwsze podejście do wyznaczenia fazy początkowej przy użyciu średnich czasów oczekiwania na kolejnych użytkowników pokazane na wykresach poniżej, wykazało, że dla różnych lambd (intensywności zgłoszeń) stabilizacja następuje po różnych odstępach czasu, a sama stabilizacja posiada inną wartość (czas oczekiwania w tym przypadku). Dodatkowo oznaczenie numeru użytkownika jako końca fazy początkowej nie jest miarodajnym wyznacznikiem i dla każdej symulacji użytkownik o tym samym numerze może pojawiać się w różnych czasach symulacji.



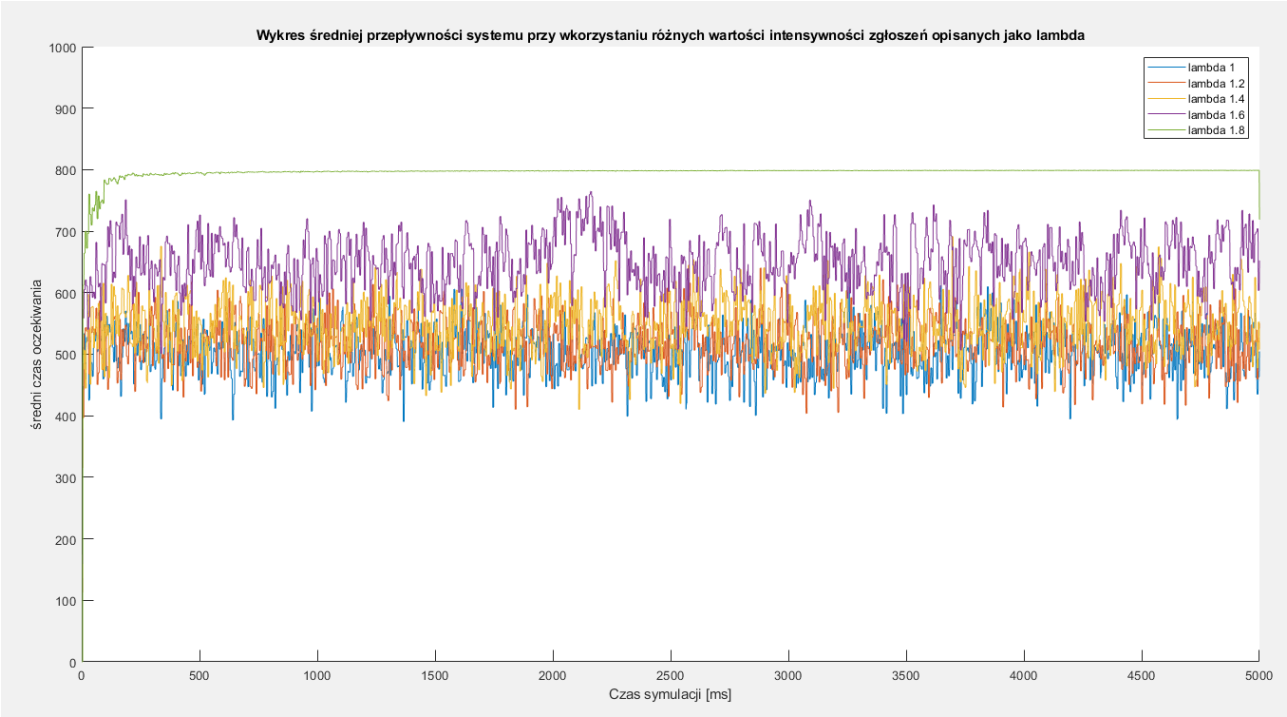
Wykres 1: Wykresy średnich czasów oczekiwań dla kolejnych użytkowników dla różnych lambd



Wykres 2: Wykresy średnich czasów oczekiwań dla kolejnych użytkowników dla różnych lambd

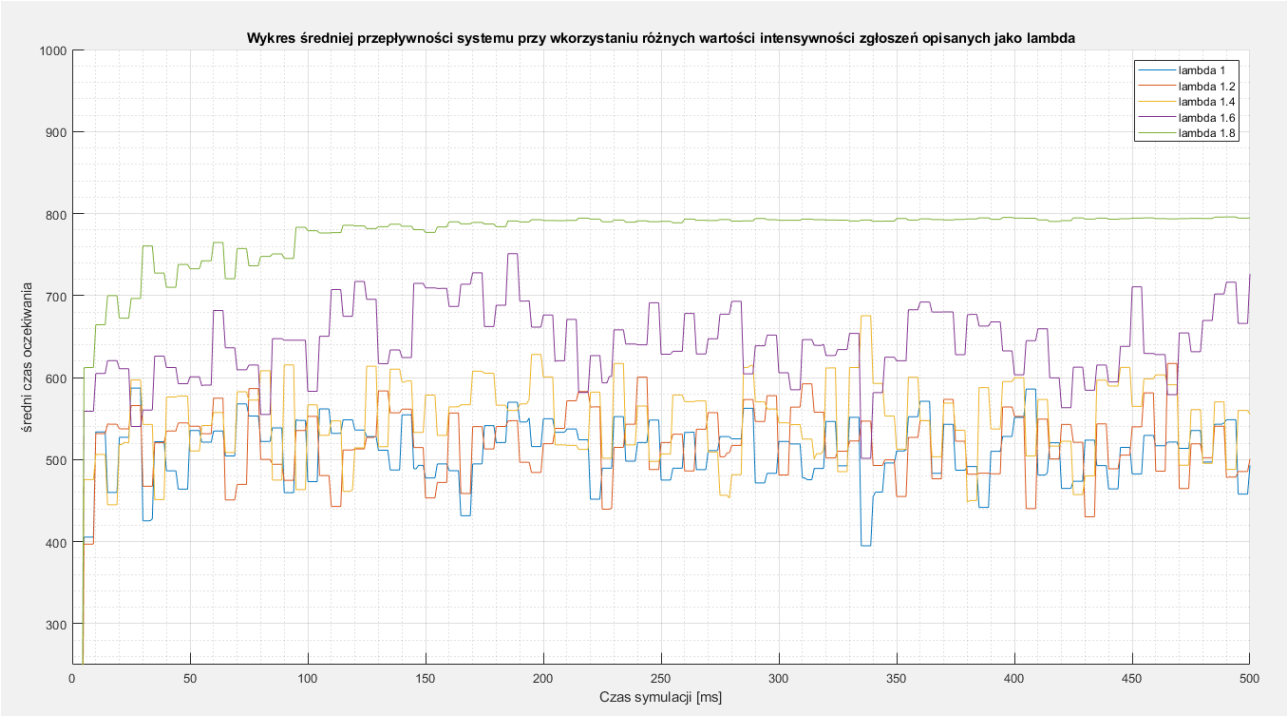
Z powyżej opisanych powodów zdecydowałem się wykonać wyznaczenie fazy początkowej za pomocą średniej przepływności systemu. Poprzednie wyniki nie były jednak bezużyteczne i uświadomiły mi, że lambdy które wybrałem dają wyniki nadające się do zbadania to znaczy dają stabilne wyniki symulacji.

Po podstawieniu wyników dla takich samych lambd jak w poprzedniej próbie oraz ustawieniu maksymalnego czasu na 5000 ms otrzymałem niesatysfakcjonujące wyniki w następującej postaci:



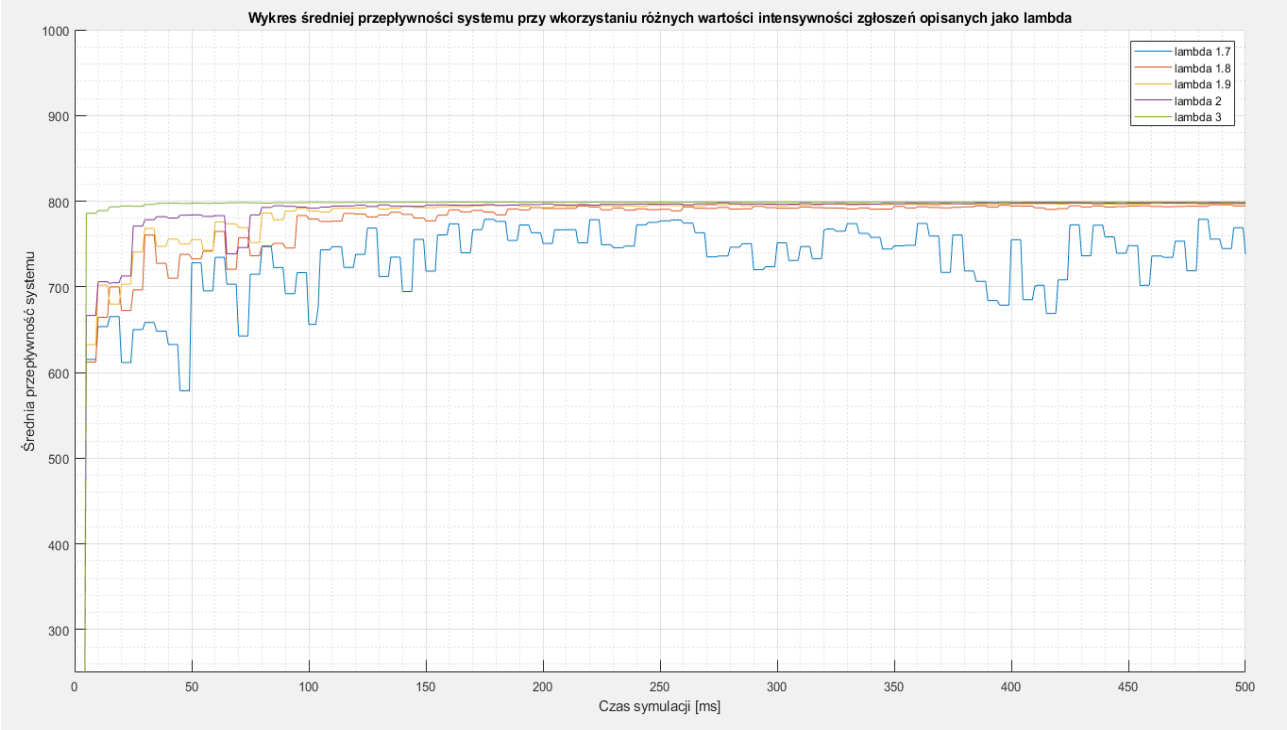
Wykres 3:Wykresy średniej przepływności systemu w czasie dla różnych lambd

Po zawężeniu wykresu do 500 ms otrzymałem następujący wykres:



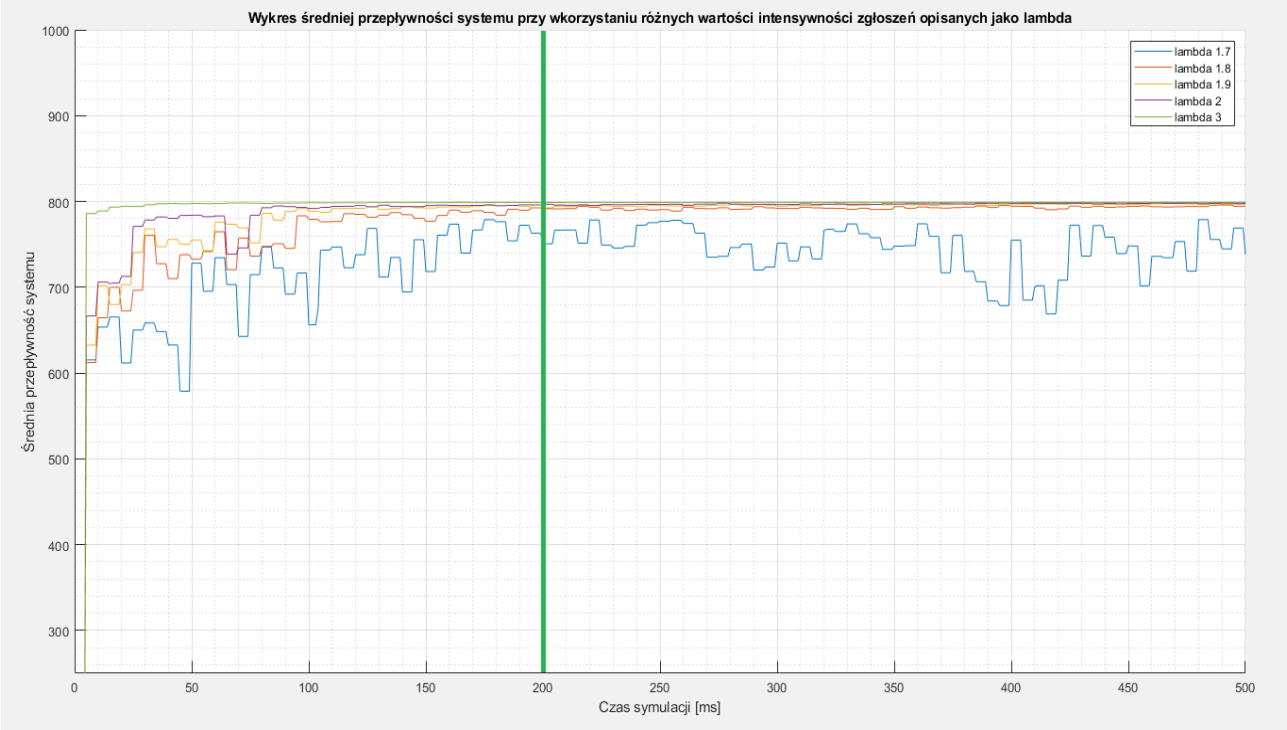
Wykres 4: Wykresy średniej przepływności systemu w czasie dla różnych lambd

Mógłbym na nim oznaczyć fazę początkową jako 200 ms jednak byłaby to niepewna wartość gdyż wynikałaby tylko z jednego wykresu. Dlatego też postanowiłem przesunąć zakres wartości lambdy do zakresu 1.7-2.0 oraz dodać do niego skrajną wyższą wartość dla porównania.



Wykres 5: Wykresy średniej przepływności systemu w czasie dla różnych lambd

Po zaznaczeniu fazy początkowej jako **200ms** wynikające w wykresów:



Wykres 6: Wykresy średniej przepływności systemu w czasie dla różnych lambd z zaznaczoną wybraną wartością końca fazy początkowej

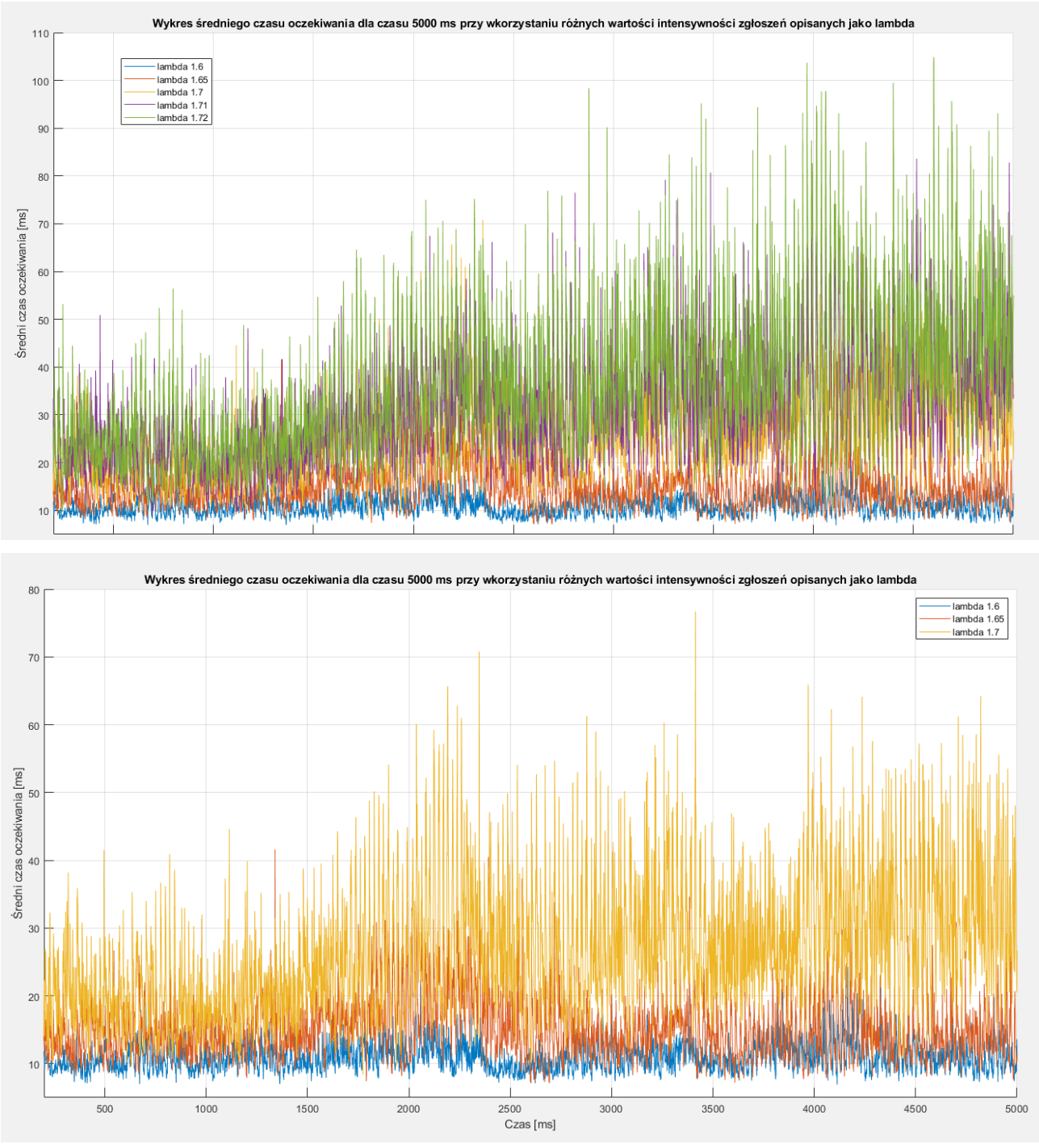
Z wykresu możemy odczytać też kilka informacji opisujących zasymulowaną sieć, są to maksymalna przepływność systemu w jednostce czasu oraz informacje o tym że współczynnik lambda ma wpływ ma bezpośredni wpływ na czas po jakim system zaczyna pracować z pełną wydajnością (jest to zależność odwrotnie proporcjonalna)

Wyznaczanie wartości parametru lambda

Posiadając wiedzę z poprzedniego badania zawęziłem obszar poszukiwań do przedziału 1.6-1.8 wartości lambda. Po pierwszym wykreśleniu oraz obliczeniu średniej otrzymałem następujące wyniki:

Wartość parametru lambda	1.6	1.65	1.7	1.71	1.72
Średni czas oczekiwania [ms]	10,9458	14,9885	26,1765	30,9681	35,6259

Tabela 5: Tabela przedstawiająca zależność średniego czasu oczekiwania od lambdy

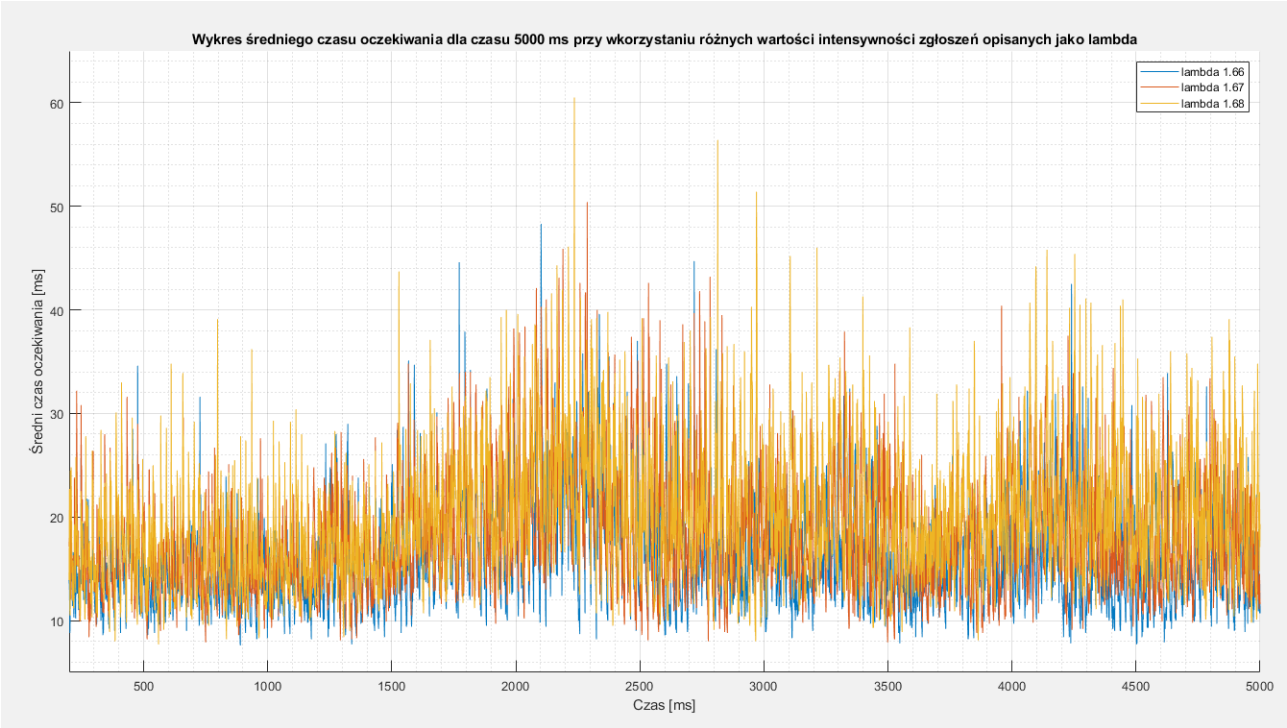


Wykres 9: Wykresy średnich czasów oczekiwań dla kolejnych użytkowników dla różnych lambd

Na podstawie danych z tabeli oraz powyższych wykresów zawęziłem zakres wartości lambda do przedziału 1.65-1.7. Uzyskałem następujące wyniki:

Wartość parametru lambda	1.66	1.67	1.68
Średni czas oczekiwania [ms]	16,1070	17,8878	19,6197

Tabela 6: Tabela przedstawiająca zależność średniego czasu oczekiwania od lambdy



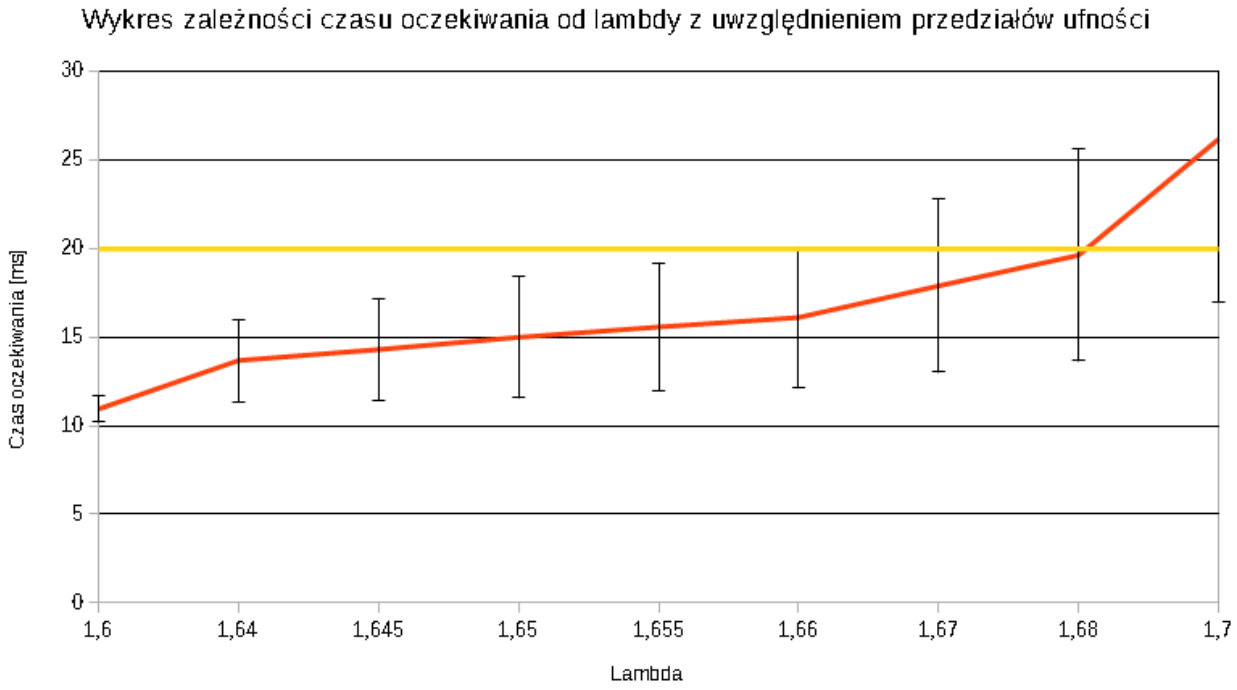
Wykres 10: Wykresy średnich czasów oczekiwań dla kolejnych użytkowników dla różnych lambd

Zgodnie z początkowym zaleceniem wartość średnia czasu oczekiwania miała wynosić 50 ms jednakże jak można zauważyć na powyższych wykresach symulator staje się niestabilny dla wartości lambda dającej taki wynik. Dlatego też poziom średniego czasu oczekiwania został obniżony do zakresu 15-20 ms. Symulator napisany przeze mnie wykazuje stabilność dla czasu oczekiwania na poziomie 20 ms zatem przyjmę ją jako wartość graniczną.

nr. symulacji\ lambda	1,6	1,64	1,65	1,65	1,66	1,66	1,67	1,68	1,7
1	13,05	17,16	17,96	19,49	20,12	20,66	23,62	27,47	34,33
2	9,87	11,06	11,11	11,68	11,77	11,61	12,22	13,29	15,23
3	9,71	11,41	11,52	12,03	11,81	12,26	12,47	13,21	13,99
4	12,21	21,12	23,98	26,64	27,61	29,32	32,35	36,33	50,18
5	10,98	12,72	13,08	12,93	13,23	13,6	14,18	14,14	20,37
6	10,84	13,63	14,06	14,48	16,82	18,12	24,04	27,82	38,96
7	10,61	11,19	11,5	11,66	11,89	11,96	12,77	13,09	14,46
8	11,04	12,87	12,96	13,01	13,4	13,88	14,54	15,29	20,72
9	9,87	11,07	11,97	11,9	12,46	12,24	13,11	13,29	16,39
10	11,27	14,56	14,81	16,07	16,61	17,43	19,57	22,25	37,14

Średnia	10,9458	13,6790	14,2962	14,9885	15,5707	16,1070	17,8878	19,6197	26,1765
Odchylenie standardowe	1,0591	3,2463	3,9716	4,7834	5,0571	5,5737	6,8157	8,3513	12,8733
Ufność	0,6564	2,0121	2,4616	2,9647	3,1344	3,4545	4,2243	5,1761	7,9788
Ufność t studenta	0,7576	2,3223	2,8411	3,4218	3,6177	3,9872	4,8756	5,9741	9,2090

Tabela 7: Tabela czasów oczekiwań dl wszystkich symulacji wraz policzoną średnią, odchyleniem standardowym oraz ufnością standardową oraz t studenta

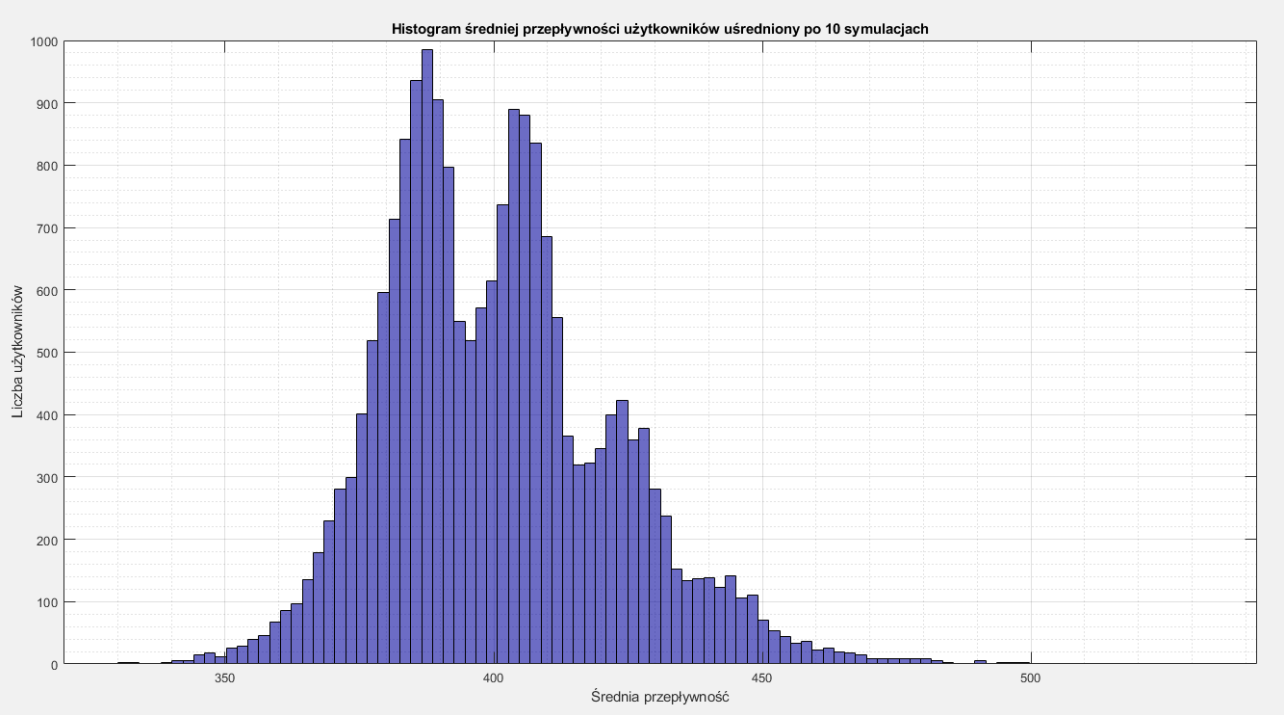


Wykres 11: Wykres zależności czasu oczekiwania od lambdy użyty do wyznaczenia poszukiwanej wartości lambda mieszczącej się w zadanym zakresie

Na podstawie obliczeń ukazanych w tabeli wykreśliłem wykres który wskazuje, że poszukiwaną lambdą jest lambda o wartości **1,655**

Histogram przepływności użytkowników

Histogram wykreślony na podstawie 10 przebiegów symulacji dla parametrów: lambda = 1.655, faza początkowa = 200ms oraz na podstawie 20 tysięcy użytkowników w każdej symulacji.



Histogram 4: Histogram przepływności użytkowników uśredniony po 10 symulacjach

Tabela z wynikami symulacji dla każdego przebiegu symulacyjnego

Tabela została stworzona na podstawie danych z 10 symulacji zainicjowanych danymi: czas symulacji 5000ms, faza początkowa 200ms, lambda = 1.655.

nr. symulacji \parametr	Średnia przepływność systemu [b/ms]	Średni czas oczekiwania [ms]	Średnia przepływność użytkownika [b/ms]	Liczba obsłużonych użytkowników	Stosunek błędu odbioru do wszystkich transmisji
1	725,73	22,64	403,68	20653	0,0982
2	697,38	13,85	397,86	20199	0,0997
3	701,94	16,66	398,15	20297	0,0967
4	717,38	17,62	402,13	20509	0,0984

5	699,5	14,49	398,57	20342	0,0994
6	714,02	18,04	401,89	20362	0,1000
7	712,7	16,72	400,98	20315	0,0993
8	703,53	14,17	399,63	20369	0,1016
9	709,03	15,07	400,61	20469	0,1012
10	713,63	14,97	400,58	20472	0,0997

Tabela 8: Tabela z wynikami symulacji dla każdego przebiegu

Tabela z uśrednionymi wynikami końcowymi uśrednionymi po wszystkich przebiegach z przedziałami ufności

	Średnia przepływność systemu [b/ms]	Średni czas oczekiwania [ms]	Średnia przepływność użytkownika [b/ms]	Liczba obsłużonych użytkowników	Stosunek błędu odbioru do wszystkich transmisji
Średnia	709,4837	16,4233	400,4071	20398,7000	0,0994
Odchylenie standardowe	8,9002	2,6305	1,8810	129,0229	0,0014
Ufność	5,5163	1,6304	1,1659	79,9677	0,0009
Ufność t studenta	6,3668	1,8818	1,3456	92,2974	0,0010

Tabela 9: Tabela z uśrednionymi wynikami końcowymi symulacji

Wyniki Końcowe

Średnia przepływność systemu = 709,4837 ± 6,3668 [b/ms]

Średni czas oczekiwania = 16,4233 ± 1,8818 [ms]

Średnia przepływność użytkownika = 400,4071 ± 1,3456 [b/ms]

Liczba obsłużonych użytkowników = 20398,7000 ± 92,2974

Stosunek błędu odbioru do wszystkich transmisji = 0,0994 ± 0,0010

Wnioski

Utworzenie symulatora oraz badania symulacyjne na nim przeprowadzone przybliżyły nam sposób wykorzystania symulacji, jej zalety oraz wady, a także pomogły zauważyć wagę doboru wykorzystywanych metod symulacyjnych jak i algorytmów kolejkowania. Projekt ten nie tylko połączył wcześniej przedstawianą wiedzę z przedmiotów o sieciach i ich strukturach, programowania oraz symulacji cyfrowej, ale także dał możliwość własnoręcznego wykorzystania tej wiedzy.

Zasymulowana przeze mnie sieć mogłaby zostać wykorzystana w rzeczywistości, niestety poprzez założenie dużej częstotliwości występowania błędów odbioru danych nie nadawałaby się dla miejsc

z duża migracją użytkowników. Najlepszym zastosowaniem takiej sieci jest domowe zacisze lub mała prywatna firma (na przykład na potrzeby klientów oczekujących w kolejce).