ГУАП

КАФЕДРА № 41

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Н.В. Апанасенко |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2 |
| АНАЛИЗ РАБОТЫ ЭЛЕМЕНТАРНОЙ СИСТЕМЫ МНОЖЕСТВЕННОГО ДОСТУПА НА ОСНОВЕ АППАРАТА МАРКОВСКИХ ПРОЦЕССОВ |
| по курсу: ИНФОРМАЦИОННЫЕ СИСТЕМЫ, СЕТИ И ТЕЛЕКОММУНИКАЦИИ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4217 |  |  |  | Д.М. Никитин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

1. **Цель работы:** анализ элементарной системы множественного доступа. Оценка среднего количества абонентов, находящихся в системе.
2. **Краткая теоретическая информация:**

Для анализа элементарной системы множественного доступа воспользуемся аппаратом Марковских процессов с дискретным временем и состоянием. Опишем количество абонентов, имеющих сообщения для передачи Марковской цепью переходных состояний. Каждое состояние обозначает количество активных (имеющих сообщения для передачи) абонентов в системе (см. рисунок 1).



Рисунок 1 – Марковская цепь для модели случайного множественного доступа

Опишем вероятности переходов из одного состояния в другое:

1. **Вероятность перехода из состояния «ноль» в произвольное состояние *j*.**

Переход из состояния 0 в состояние *j* может произойти, только когда количество АБ, у которых появилось сообщение будет равно *j*, то есть . Так как  распределено по закону Пуассона получаем:



1. **Вероятность перехода из *i*-го состояния в *i*-**1**.**

Событие перехода из состояния *i* в состояние *i*-1 означает, что новых АБ не появилось, и один АБ передал сообщение. То есть  и :



Вероятность «успеха» (один АБ передавал, остальные не передавали), определяется как:



Вероятность отсутствия новых АБ определяется как:



1. **Вероятность остаться в состоянии *i*.**

Переход из состояния *i* в *i*. может произойти в случае, если один АБ передал сообщение, и один новый АБ появился, или если новых АБ не было, и ни один не передал:



Вероятность того, что ни один АБ не передал сообщение определяется как:



1. **Вероятность перехода из состояния *i* в состояние *j*.**

Переход из состояния *i* в *j* может произойти в случае, если один АБ передал сообщение и появились (*j*-*i*+1) АБ-ов, или если ни один не передал сообщение и появились (*j*-*i*) АБ-ов. Вероятность данного события определяется как:



Если вероятность передачи активными АБ - *p* является константой, то есть не зависит ни от количества активных АБ в текущем слоте, ни от интенсивности появления активных АБ, то такая система является неустойчивой (количество абонентов, у которых имеется сообщения для передачи неограниченно возрастает при  ).

Используя описанные выше вероятности переходов из одного состояния в другое, можно сформировать систему усеченных линейных уравнений, для фиксированного значения :





Решая данную систему усеченных линейных алгебраических уравнений (количество уравнений *K*), для фиксированного , получим вектор стационарного распределения, каждое из значений  означает вероятность того, что в системе находится *i* абонентов.

Тогда оценить среднее количество абонентов, находящихся в системе множественного доступа можно как:

,

где .

1. **Графики зависимости количества абонентов, находящихся в системе от интенсивности входного потока при K = [10, 50, 100, 1000] и код использованной программы:**

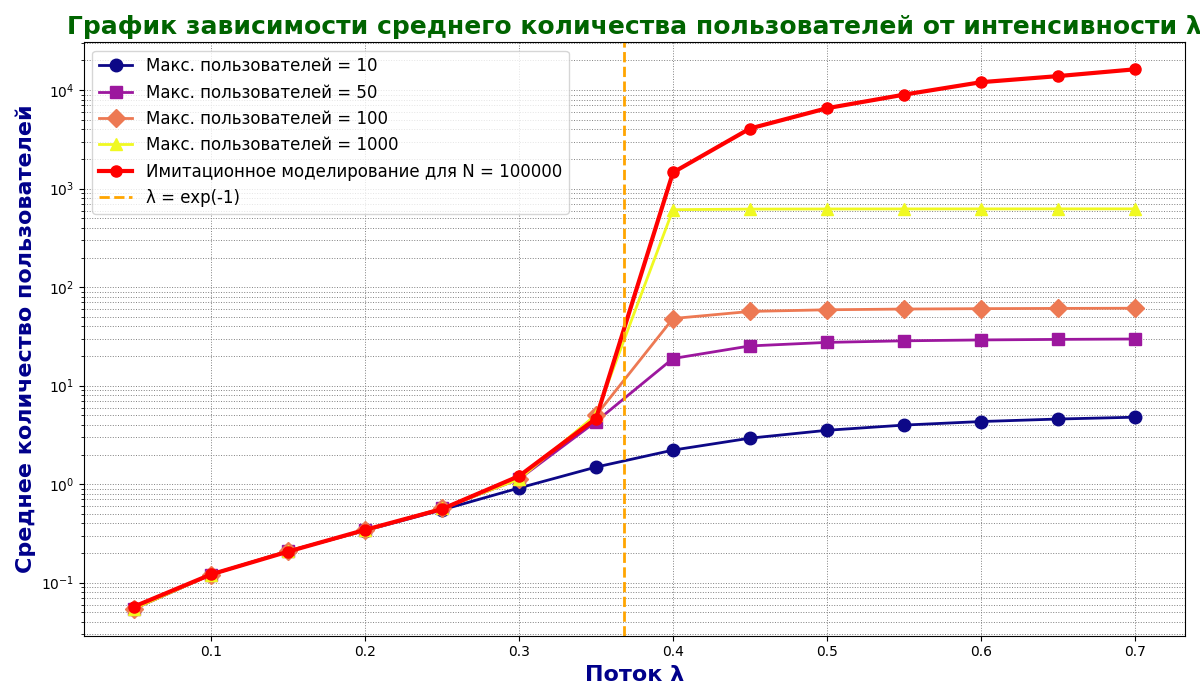


Рисунок 2 – Графики зависимости количества абонентов, находящихся в системе от интенсивности входного потока при K = [10, 50, 100, 1000]

import math  
import numpy as np  
import matplotlib.pyplot as plt  
from functools import lru\_cache  
from scipy.special import factorial  
  
  
class ProbabilityModel:  
 def \_\_init\_\_(self, lambda\_val):  
 self.lambda\_val = lambda\_val  
  
 @lru\_cache(maxsize=None)  
 def calculate\_probability(self, i):  
 if i == 1:  
 return 1  
 if i <= 0:  
 return 0  
 return (1 - 1 / i) \*\* (i - 1)  
  
 @lru\_cache(maxsize=None)  
 def poisson\_distribution(self, j):  
 if j < 0:  
 return 0  
 if j == 0:  
 return math.exp(-self.lambda\_val)  
 return (self.lambda\_val \*\* j) \* math.exp(-self.lambda\_val) / factorial(j)  
  
 @lru\_cache(maxsize=None)  
 def probability\_to\_zero(self, j):  
 return self.poisson\_distribution(j)  
  
 @lru\_cache(maxsize=None)  
 def probability\_from\_i\_to\_0(self, i):  
 if i < 1:  
 return 0  
 return self.calculate\_probability(i) \* self.poisson\_distribution(0)  
  
 @lru\_cache(maxsize=None)  
 def probability\_to\_stay(self, i):  
 return (1 - self.calculate\_probability(i)) \* self.poisson\_distribution(0) + self.calculate\_probability(i) \* self.poisson\_distribution(1)  
  
 @lru\_cache(maxsize=None)  
 def probability\_from\_i\_to\_j(self, i, j):  
 if j < i:  
 return 0  
 return self.calculate\_probability(i) \* self.poisson\_distribution(j - i + 1) + (1 - self.calculate\_probability(i)) \* self.poisson\_distribution(j - i)  
  
  
def process\_data(lambda\_val, slot\_count):  
 user\_count = []  
 state = []  
 current\_state = 0  
 for \_ in range(slot\_count):  
 poisson\_val = np.random.poisson(lambda\_val)  
 if current\_state > 0:  
 transmission\_prob = 1 / current\_state  
 binomial\_val = np.random.binomial(current\_state, transmission\_prob)  
 else:  
 binomial\_val = 0  
 if binomial\_val == 1:  
 current\_state -= 1  
 elif binomial\_val > 1:  
 pass  
 current\_state += poisson\_val  
 user\_count.append(current\_state)  
 state.append(1 if binomial\_val == 1 else 0)  
 avg\_users = np.mean(user\_count)  
 avg\_time = avg\_users / lambda\_val  
 avg\_state = np.mean(state)  
 return avg\_users, avg\_time, avg\_state  
  
  
def transition\_matrix(lambda\_val, max\_users):  
 prob\_model = ProbabilityModel(lambda\_val)  
 matrix = np.zeros((max\_users + 1, max\_users + 1))  
 for i in range(max\_users + 1):  
 for j in range(max\_users + 1):  
 if i == 0:  
 matrix[i, j] = prob\_model.probability\_to\_zero(j)  
 elif i == j:  
 matrix[i, j] = prob\_model.probability\_to\_stay(i)  
 elif i < j:  
 matrix[i, j] = prob\_model.probability\_from\_i\_to\_j(i, j)  
 elif i == j + 1:  
 matrix[i, j] = prob\_model.probability\_from\_i\_to\_0(i)  
 else:  
 matrix[i, j] = 0  
  
 matrix\_with\_ones = np.vstack([matrix.T - np.eye(max\_users + 1), np.ones(max\_users + 1)])  
  
 coeffs = np.zeros(max\_users + 2)  
 coeffs[-1] = 1  
  
 steady\_state = np.linalg.lstsq(matrix\_with\_ones, coeffs, rcond=None)[0]  
 return steady\_state  
  
  
def average\_users\_across\_lambdas(lambda\_values, max\_users):  
 averages = []  
 for lambda\_val in lambda\_values:  
 steady\_state = transition\_matrix(lambda\_val, max\_users)  
 avg\_users = np.dot(steady\_state, np.arange(max\_users + 1))  
 averages.append(avg\_users)  
 return averages  
  
  
def plot\_results(lambda\_values, avg\_results, max\_users\_values, simulation\_results):  
 plt.figure(figsize=(12, 7))  
  
 colors = plt.cm.plasma(np.linspace(0, 1, len(max\_users\_values)))  
 markers = ['o', 's', 'D', '^', 'v']  
  
 for idx, max\_users in enumerate(max\_users\_values):  
 plt.plot(lambda\_values, avg\_results[idx],  
 label=f"Макс. пользователей = {max\_users}",  
 marker=markers[idx % len(markers)],  
 linestyle='-',  
 color=colors[idx],  
 markersize=9,  
 linewidth=2)  
  
 plt.plot(lambda\_values, simulation\_results,  
 label="Имитационное моделирование для N = 100000",  
 marker='o',  
 linestyle='-',  
 color='red',  
 markersize=8,  
 linewidth=3)  
  
 plt.xlabel("Поток λ", fontsize=16, fontweight='bold', color='darkblue')  
 plt.ylabel("Среднее количество пользователей", fontsize=16, fontweight='bold', color='darkblue')  
 plt.yscale("log")  
 plt.title("График зависимости среднего количества пользователей от интенсивности λ", fontsize=18, fontweight='bold', color='darkgreen')  
 plt.axvline(x=np.exp(-1), color='orange', linestyle='--', label='λ = exp(-1)', linewidth=2)  
 plt.legend(fontsize=12)  
 plt.grid(True, which='both', linestyle=':', linewidth=0.7, color='gray')  
 plt.tight\_layout()  
 plt.savefig(f"lambda\_plot.png")  
 plt.show()  
  
  
def main():  
 lambda\_values = np.arange(0.05, 0.71, 0.05)  
 num\_slots = 100000  
 simulation\_results = []  
 for lambda\_val in lambda\_values:  
 avg\_users, avg\_time, avg\_state = process\_data(lambda\_val, num\_slots)  
 simulation\_results.append(avg\_users)  
  
 max\_users\_values = [10, 50, 100, 1000]  
 avg\_results = []  
 for max\_users in max\_users\_values:  
 averages = average\_users\_across\_lambdas(lambda\_values, max\_users)  
 avg\_results.append(averages)  
  
 print(f"{'Макс. пользователей':^25}{'λ (Интенсивность)':^20}{'Среднее количество пользователей':^30}")  
 print("-" \* 77)  
 for i, max\_users in enumerate(max\_users\_values):  
 for lambda\_val, avg in zip(lambda\_values, avg\_results[i]):  
 print(f"{max\_users:^25}{lambda\_val:^20.2f}{avg:^30.4f}")  
 print()  
  
 plot\_results(lambda\_values, avg\_results, max\_users\_values, simulation\_results)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

1. **Качественные и количественные выводы:**

**Качественные:**

С ростом интенсивности λ среднее количество пользователей сначала растёт медленно, но после некоторого критического значения λ наблюдается резкий рост. Для разных максимальных ограничений пользователей (10, 50, 100, 1000) среднее количество пользователей "насыщается" на разных уровнях. При ограничении максимального числа пользователей система достигает насыщения (плато) быстрее, чем в моделировании без жёстких ограничений. Чем меньше предельное количество пользователей (например, 10 или 50), тем более "устойчивым" остаётся рост среднего числа пользователей с увеличением λ. Ограничение пользователей позволяет системе "контролировать" нагрузку, и среднее количество пользователей перестаёт расти после насыщения.

Красная линия показывает имитационное моделирование для N=100000. Она растёт стремительно и в отсутствие ограничения пользователей продолжается выше, чем при фиксированных значениях максимального числа пользователей. Линия демонстрирует, что система становится нестабильной, так как среднее количество пользователей быстро растёт, если λ превышает критическое значение.

Для всех случаев при низкой интенсивности λ, среднее количество пользователей растёт почти линейно, так как система ещё не перегружена, и интенсивность поступления заявок пропорциональна количеству пользователей.

Для систем с ограниченным максимальным числом пользователей (10,50,100,1000) среднее число пользователей **достигает предела** и стабилизируется. Это происходит из-за **ограниченности ресурсов** — система не может обслуживать больше пользователей, чем её предел.

На графике вертикальной пунктирной линией оранжевого цвета выделена **точка перехода** (λ=e-1)**,** после которой начинается резкоеувеличение числа пользователей для имитационного моделирования.

**Количественные:**

При λ=e-1 происходит резкий рост среднего числа пользователей для моделирования без ограничения (красная линия).

При Макс. пользователей=10 среднее число пользователей стабилизируется около **10**. При Макс. пользователей=50 стабилизация происходит на уровне **50**. При Макс. пользователей=100 плато достигается около **100**. При Макс. пользователей=1000 среднее число пользователей стабилизируется на уровне **1000**.

Для N=100000 среднее количество пользователей при больших λ (близких к 1) достигает значений порядка **10000 и выше**, что на несколько порядков превышает ограничения в других сценариях.