Департамент образования и науки города Москвы

Государственное автономное образовательное учреждение

высшего образования города Москвы

«Московский городской педагогический университет»

Институт цифрового образования

Департамент информатики, управления и технологий

ДИСЦИПЛИНА:

Распределенные системы

Лабораторная работа 4

Выполнила: Тяпкина П.А., группа: АДЭУ-221

Преподаватель: Босенко Т.М.

Москва

2024

Лабораторная работа 4. Обнаружение отказов в распределенной системе

Цель работы: изучить принципы обнаружения отказов в распределенных системах с помощью симулятора Serf Convergence Simulator и проанализировать влияние различных параметров на время конвергенции и использование полосы пропускания.

Теоретическая часть: Serf — это инструмент для управления кластером, который использует протокол gossip для обнаружения узлов, обнаружения отказов и оркестрации событий. Протокол gossip — это метод распространения информации в распределенной системе, где узлы периодически обмениваются информацией со случайно выбранными соседями.

Таблицу результатов экспериментов:

Gossip Interval	Gossip Fanout	Nodes	Packet Loss	Node Failures	Время до "Хотя бы один узел знает"	Время до "Все живые узлы знают"	Макс. использование полосы пропускания
0.2	3	100	5%	5%			

График зависимостей:

Анализ результатов:

1) Как изменение Gossip Interval влияет на время конвергенции и использование полосы пропускания?

Изменение Gossip Interval напрямую влияет на время конвергенции и использование полосы пропускания.

Время конвергенции: Уменьшение интервала госсипа (например, с 0.5 до 0.1 секунды) может привести к более быстрому распространению информации среди узлов, что сокращает время, необходимое для достижения согласованности. Однако слишком частые обновления могут привести к увеличению нагрузки на сеть.

Использование полосы пропускания: Более короткий интервал может увеличить количество передаваемых сообщений, что увеличивает использование полосы пропускания. Это может привести к перегрузке

сети, особенно если количество узлов велико или если используется высокая степень дублирования сообщений.

2) Какое влияние оказывает увеличение Gossip Fanout на производительность системы?

Увеличение Gossip Fanout (количества узлов, к которым отправляется сообщение за один раунд) улучшает скорость распространения информации:

Производительность системы: Более высокий fanout позволяет узлам быстрее делиться информацией с большим количеством соседей, что может значительно сократить время, необходимое для достижения согласованности в системе. Однако это также может привести к увеличению использования полосы пропускания и потенциальным конфликтам при передаче сообщений.

Согласованность: Увеличение fanout также может повысить вероятность того, что информация будет быстро распространена по всей сети, уменьшая вероятность потери данных.

3) Как масштабируется система при увеличении количества узлов?

Система с использованием протокола госсипа масштабируется логарифмически по количеству узлов:

Масштабируемость: При увеличении числа узлов время, необходимое для достижения согласованности, растет медленно (логарифмически), что делает госсип-протоколы подходящими для крупных распределенных систем. Каждый узел взаимодействует только с фиксированным числом других узлов (определяемым fanout), что позволяет избежать чрезмерного роста времени конвергенции при добавлении новых узлов.

4) Каково влияние потери пакетов на время конвергенции?

Потеря пакетов негативно влияет на время конвергенции:

Замедление распространения информации: Если пакеты теряются, информация не достигает всех узлов, что приводит к необходимости повторных попыток передачи данных. Это увеличивает общее время, необходимое для достижения согласованности.

Эффективность системы: Высокий уровень потерь пакетов может вызвать значительное замедление работы протокола госсипа и увеличить нагрузку на сеть из-за повторной передачи сообщений.

5) Как процент отказавших узлов влияет на общую производительность системы?

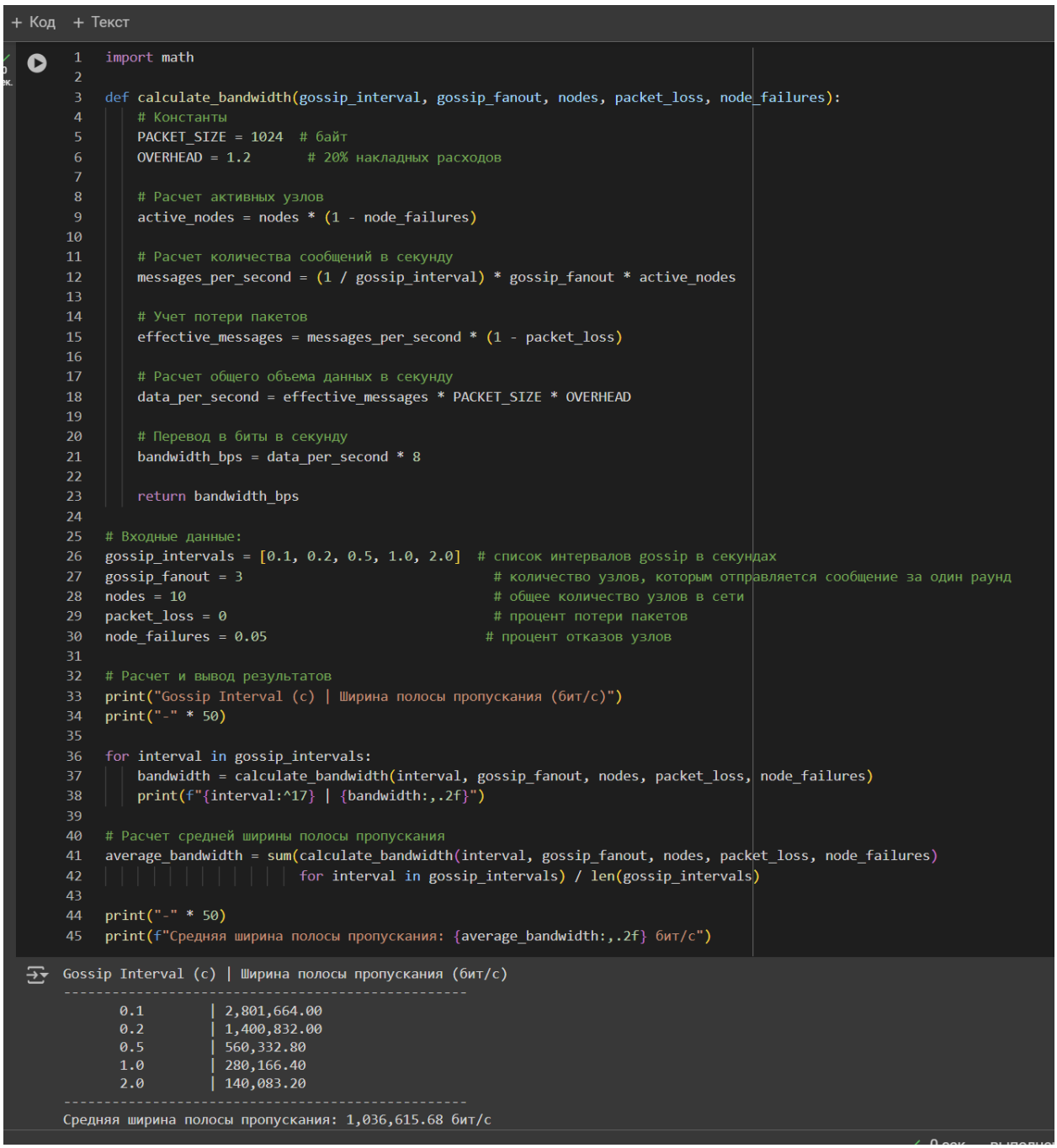
Процент отказавших узлов существенно влияет на общую производительность системы:

Снижение надежности: Увеличение процента отказавших узлов снижает общую надежность системы и может привести к увеличению времени обнаружения отказов.

Влияние на согласованность: Если слишком много узлов отказывает, это может затруднить достижение консенсуса среди оставшихся активных узлов. Это может привести к задержкам в распространении информации и ухудшению производительности системы в целом.

Увеличение нагрузки на оставшиеся узлы: Оставшиеся активные узлы могут испытывать повышенную нагрузку из-за необходимости компенсировать отсутствие отказавших узлов, что также может снизить их производительность.

Проверка кода на работоспособность кода, показывающего Ширину полосы пропускания:



```
+ Код  + Текст

import math

def calculate_bandwidth(gossip_interval, gossip_fanout, nodes, packet_loss, node_failures):
    # Константы
    PACKET_SIZE = 1024  # байт
    OVERHEAD = 1.2       # 20% накладных расходов

    # Расчет активных узлов
    active_nodes = nodes * (1 - node_failures)

    # Расчет количества сообщений в секунду
    messages_per_second = (1 / gossip_interval) * gossip_fanout * active_nodes

    # Учет потери пакетов
    effective_messages = messages_per_second * (1 - packet_loss)

    # Расчет общего объема данных в секунду
    data_per_second = effective_messages * PACKET_SIZE * OVERHEAD

    # Перевод в биты в секунду
    bandwidth_bps = data_per_second * 8

    return bandwidth_bps

# Входные данные:
gossip_intervals = [0.1, 0.2, 0.5, 1.0, 2.0]  # список интервалов gossip в секундах
gossip_fanout = 3                              # количество узлов, которым отправляется сообщение за один раунд
nodes = 10                                     # общее количество узлов в сети
packet_loss = 0                                # процент потери пакетов
node_failures = 0.05                           # процент отказов узлов

# Расчет и вывод результатов
print("Gossip Interval (c) | Ширина полосы пропускания (бит/с)")
print("-" * 50)

for interval in gossip_intervals:
    bandwidth = calculate_bandwidth(interval, gossip_fanout, nodes, packet_loss, node_failures)
    print(f"{interval:^17} | {bandwidth:,.2f}")

# Расчет средней ширины полосы пропускания
average_bandwidth = sum(calculate_bandwidth(interval, gossip_fanout, nodes, packet_loss, node_failures)
                        for interval in gossip_intervals) / len(gossip_intervals)

print("-" * 50)
print(f"Средняя ширина полосы пропускания: {average_bandwidth:,.2f} бит/с")
```

```
Gossip Interval (c) | Ширина полосы пропускания (бит/с)
--------------------------------------------------
       0.1        | 2,801,664.00
       0.2        | 1,400,832.00
       0.5        | 560,332.80
       1.0        | 280,166.40
       2.0        | 140,083.20
--------------------------------------------------
Средняя ширина полосы пропускания: 1,036,615.68 бит/с
```

При подсчете нам удается узнать среднюю ширину полосы пропуская

Далее производим расчет изменения Gossip Interval, влияющий на время конвергенции:

```python
import math

def calculate_bandwidth(gossip_interval, gossip_fanout, nodes, packet_loss, node_failures):
    PACKET_SIZE = 1024  # байт
    OVERHEAD = 1.2      # 20% накладных расходов
    active_nodes = nodes * (1 - node_failures)
    messages_per_second = (1 / gossip_interval) * gossip_fanout * active_nodes
    effective_messages = messages_per_second * (1 - packet_loss)
    data_per_second = effective_messages * PACKET_SIZE * OVERHEAD
    bandwidth_bps = data_per_second * 8
    return bandwidth_bps

def estimate_convergence_time(gossip_interval, gossip_fanout, nodes, node_failures):
    active_nodes = nodes * (1 - node_failures)
    infection_rate = gossip_fanout / active_nodes
    rounds_to_infect_99_percent = math.ceil(math.log(0.01) / math.log(1 - infection_rate))
    convergence_time = rounds_to_infect_99_percent * gossip_interval
    return convergence_time

def run_simulation(gossip_intervals, gossip_fanout, nodes, packet_loss, node_failures):
    results = []
    for interval in gossip_intervals:
        bandwidth = calculate_bandwidth(interval, gossip_fanout, nodes, packet_loss, node_failures)
        convergence_time = estimate_convergence_time(interval, gossip_fanout, nodes, node_failures)
        results.append((interval, bandwidth, convergence_time))
    return results

# Входные данные
gossip_intervals = [0.1, 0.2, 0.5, 1.0, 2.0]
gossip_fanout = 3
nodes = 10
packet_loss = 0
node_failures = 0.05

# Запуск симуляции
results = run_simulation(gossip_intervals, gossip_fanout, nodes, packet_loss, node_failures)

# Вывод результатов
print("Gossip Interval (c) | Ширина полосы пропускания (бит/с) | Время конвергенции (с)")
print("-" * 80)
for interval, bandwidth, convergence_time in results:
    print(f"{interval:^17} | {bandwidth:^33,.2f} | {convergence_time:^21,.2f}")

# Анализ результатов
print("\nАнализ результатов:")
print("1. Влияние Gossip Interval на использование полосы пропускания:")
for i in range(len(results) - 1):
    bandwidth_change = (results[i+1][1] - results[i][1]) / results[i][1] * 100
    print(f" При увеличении интервала с {results[i][0]} до {results[i+1][0]} с, использование полосы пропускания изменяется на {bandwidth_change:.2f}%")

print("\n2. Влияние Gossip Interval на время конвергенции:")
for i in range(len(results) - 1):
    convergence_change = (results[i+1][2] - results[i][2]) / results[i][2] * 100
```

```
Gossip Interval (c) | Ширина полосы пропускания (бит/с) | Время конвергенции (с)
--------------------------------------------------------------------------------
       0.1          |             2,801,664.00          |          1.30
       0.2          |             1,400,832.00          |          2.60
       0.5          |              560,332.80           |          6.50
       1.0          |              280,166.40           |         13.00
       2.0          |              140,083.20           |         26.00

Анализ результатов:
1. Влияние Gossip Interval на использование полосы пропускания:
 При увеличении интервала с 0.1 до 0.2 с, использование полосы пропускания изменяется на -50.00%
 При увеличении интервала с 0.2 до 0.5 с, использование полосы пропускания изменяется на -60.00%
 При увеличении интервала с 0.5 до 1.0 с, использование полосы пропускания изменяется на -50.00%
 При увеличении интервала с 1.0 до 2.0 с, использование полосы пропускания изменяется на -50.00%

2. Влияние Gossip Interval на время конвергенции:
 При увеличении интервала с 0.1 до 0.2 с, время конвергенции изменяется на 100.00%
 При увеличении интервала с 0.2 до 0.5 с, время конвергенции изменяется на 150.00%
 При увеличении интервала с 0.5 до 1.0 с, время конвергенции изменяется на 100.00%
 При увеличении интервала с 1.0 до 2.0 с, время конвергенции изменяется на 100.00%
```

Вариант 16. Влияние временных задержек
- Gossip Interval: 0.2 с
- Gossip Fanout: 3
- Nodes: 100
- Packet Loss: 5%
- Node Failures: 5%
- Сценарии задержек:
  1. Без задержек
  2. Случайные задержки (0-50 мс)
  3. Фиксированные задержки (25 мс)
  4. Экспоненциально распределенные задержки (среднее 25 мс)

Задача: исследовать влияние различных типов временных задержек на производительность системы.

Производим на своем варианте задания – те же действия, что и на основном задании. Ищем Ширину полосы пропускания:

```python
import math

def calculate_bandwidth(gossip_interval, gossip_fanout, nodes, packet_loss, node_failures):
    # Константы
    PACKET_SIZE = 1024  # байт
    OVERHEAD = 1.2       # 20% накладных расходов

    # Расчет активных узлов
    active_nodes = nodes * (1 - node_failures)

    # Расчет количества сообщений в секунду
    messages_per_second = (1 / gossip_interval) * gossip_fanout * active_nodes

    # Учет потери пакетов
    effective_messages = messages_per_second * (1 - packet_loss)

    # Расчет общего объема данных в секунду
    data_per_second = effective_messages * PACKET_SIZE * OVERHEAD

    # Перевод в биты в секунду
    bandwidth_bps = data_per_second * 8

    return bandwidth_bps

# Входные данные:
gossip_intervals = [0.2]  # список интервалов gossip в секундах
gossip_fanout = 3                              # количество узлов, которым отправляется сообщение за один раунд
nodes = 100                                    # общее количество узлов в сети
packet_loss = 0.05                                # процент потери пакетов
node_failures = 0.05                           # процент отказов узлов

# Расчет и вывод результатов
print("Gossip Interval (c) | Ширина полосы пропускания (бит/с)")
print("-" * 50)

for interval in gossip_intervals:
    bandwidth = calculate_bandwidth(interval, gossip_fanout, nodes, packet_loss, node_failures)
    print(f"{interval:^17} | {bandwidth:,.2f}")

# Расчет средней ширины полосы пропускания
average_bandwidth = sum(calculate_bandwidth(interval, gossip_fanout, nodes, packet_loss, node_failures)
                        for interval in gossip_intervals) / len(gossip_intervals)

print("-" * 50)
print(f"Средняя ширина полосы пропускания: {average_bandwidth:,.2f} бит/с")
```

```
Gossip Interval (c) | Ширина полосы пропускания (бит/с)
--------------------------------------------------
       0.2        | 13,307,904.00
--------------------------------------------------
Средняя ширина полосы пропускания: 13,307,904.00 бит/с
```

В связи с единственным интервалом – получаем единственное значение, которое и является средним.

Исправляем код, чтобы подходил под условия варианта «Сценарии задержек»:

```python
import math

def calculate_bandwidth(gossip_interval, gossip_fanout, nodes, packet_loss, node_failures):
    # Константы
    PACKET_SIZE = 1024  # байт
    OVERHEAD = 1.2       # 20% накладных расходов

    # Расчет активных узлов
    active_nodes = nodes * (1 - node_failures)

    # Расчет количества сообщений в секунду
    messages_per_second = (1 / gossip_interval) * gossip_fanout * active_nodes

    # Учет потери пакетов
    effective_messages = messages_per_second * (1 - packet_loss)

    # Расчет общего объема данных в секунду
    data_per_second = effective_messages * PACKET_SIZE * OVERHEAD

    # Перевод в биты в секунду
    bandwidth_bps = data_per_second * 8

    return bandwidth_bps

def simulate_delay(scenario, gossip_interval):
    if scenario == "без задержек":
        return gossip_interval
    elif scenario == "случайные задержки":
        delay = random.uniform(0, 0.05)  # случайная задержка от 0 до 50 мс
        return gossip_interval + delay
    elif scenario == "фиксированные задержки":
        return gossip_interval + 0.025  # фиксированная задержка 25 мс
    elif scenario == "экспоненциально распределенные задержки":
        delay = random.expovariate(1 / 0.025)  # среднее значение 25 мс
        return gossip_interval + delay

# Входные данные:
gossip_intervals = [0.2]  # список интервалов gossip в секундах
gossip_fanout = 3         # количество узлов, которым отправляется сообщение за один раунд
nodes = 100               # общее количество узлов в сети
packet_loss = 0.05        # процент потери пакетов
node_failures = 0.05      # процент отказов узлов

# Сценарии задержек
delay_scenarios = [
    "без задержек",
    "случайные задержки",
    "фиксированные задержки",
    "экспоненциально распределенные задержки"
]

# Расчет и вывод результатов для каждого сценария задержек
print("Сценарий задержек               | Ширина полосы пропускания (бит/с)")
print("-" * 70)

for scenario in delay_scenarios:
    adjusted_interval = simulate_delay(scenario, gossip_intervals[0])
    bandwidth = calculate_bandwidth(adjusted_interval, gossip_fanout, nodes, packet_loss, node_failures)
    print(f"{scenario:<35} | {bandwidth:,.2f}")

# Расчет средней ширины полосы пропускания для всех сценариев
average_bandwidth = sum(
    calculate_bandwidth(simulate_delay(scenario, gossip_intervals[0]), gossip_fanout, nodes, packet_loss, node_failures)
    for scenario in delay_scenarios) / len(delay_scenarios)

print("-" * 70)
print(f"Средняя ширина полосы пропускания: {average_bandwidth:,.2f} бит/с")
```

```
Сценарий задержек                | Ширина полосы пропускания (бит/с)
----------------------------------------------------------------------
без задержек                       | 13,307,904.00
случайные задержки                 | 12,894,787.77
фиксированные задержки             | 11,829,248.00
экспоненциально распределенные задержки | 12,374,462.02
----------------------------------------------------------------------
Средняя ширина полосы пропускания: 12,015,113.88 бит/с
```

Далее снова проводим Расчет изменение Gossip Interval, который влияет на время конвергенции.

```python
import math

def calculate_bandwidth(gossip_interval, gossip_fanout, nodes, packet_loss, node_failures):
    PACKET_SIZE = 1024  # байт
    OVERHEAD = 1.2      # 20% накладных расходов
    active_nodes = nodes * (1 - node_failures)
    messages_per_second = (1 / gossip_interval) * gossip_fanout * active_nodes
    effective_messages = messages_per_second * (1 - packet_loss)
    data_per_second = effective_messages * PACKET_SIZE * OVERHEAD
    bandwidth_bps = data_per_second * 8
    return bandwidth_bps

def estimate_convergence_time(gossip_interval, gossip_fanout, nodes, node_failures):
    active_nodes = nodes * (1 - node_failures)
    infection_rate = gossip_fanout / active_nodes
    rounds_to_infect_99_percent = math.ceil(math.log(0.01) / math.log(1 - infection_rate))
    convergence_time = rounds_to_infect_99_percent * gossip_interval
    return convergence_time

def run_simulation(gossip_intervals, gossip_fanout, nodes, packet_loss, node_failures):
    results = []
    for interval in gossip_intervals:
        bandwidth = calculate_bandwidth(interval, gossip_fanout, nodes, packet_loss, node_failures)
        convergence_time = estimate_convergence_time(interval, gossip_fanout, nodes, node_failures)
        results.append((interval, bandwidth, convergence_time))
    return results

# Входные данные
gossip_intervals = [0.2]
gossip_fanout = 3
nodes = 100
packet_loss = 0.05
node_failures = 0.05

# Запуск симуляции
results = run_simulation(gossip_intervals, gossip_fanout, nodes, packet_loss, node_failures)

# Вывод результатов
print("Gossip Interval (c) | Ширина полосы пропускания (бит/с) | Время конвергенции (с)")
print("-" * 80)
for interval, bandwidth, convergence_time in results:
    print(f"{interval:^17} | {bandwidth:^33,.2f} | {convergence_time:^21,.2f}")

# Анализ результатов
print("\nАнализ результатов:")
print("1. Влияние Gossip Interval на использование полосы пропускания:")
for i in range(len(results) - 1):
    bandwidth_change = (results[i+1][1] - results[i][1]) / results[i][1] * 100
    print(f" При увеличении интервала с {results[i][0]} до {results[i+1][0]} с, использование полосы пропускания изменяется на {bandwidth_change:.2f}%")

print("\n2. Влияние Gossip Interval на время конвергенции:")
for i in range(len(results) - 1):
    convergence_change = (results[i+1][2] - results[i][2]) / results[i][2] * 100
    print(f" При увеличении интервала с {results[i][0]} до {results[i+1][0]} с, время конвергенции изменяется на {convergence_change:.2f}%")

```

```
Gossip Interval (c) | Ширина полосы пропускания (бит/с) | Время конвергенции (с)
--------------------------------------------------------------------------------
       0.2        |          13,307,904.00          |         28.80

Анализ результатов:
1. Влияние Gossip Interval на использование полосы пропускания:

2. Влияние Gossip Interval на время конвергенции:
```

Далее выполняем Python-скрипт:

```

Python
```

```
import random
import time
import matplotlib.pyplot as plt

class Node:
    def __init__(self, node_id):
        self.id = node_id
        self.knows_failure = False

class BaseSimulator:
    def __init__(self, num_nodes, interval, node_failures):
        self.nodes = [Node(i) for i in range(num_nodes)]
        self.interval = interval
        self.node_failures = node_failures
        self.failed_nodes = set()
        self.bandwidth_usage = 0

    def simulate_failure(self):
        num_failures = int(len(self.nodes) * self.node_failures /
100)
        self.failed_nodes =
set(random.sample(range(len(self.nodes)), num_failures))
        if self.failed_nodes:
            self.nodes[random.choice(list(self.failed_nodes))].knows
_failure = True
        elif self.nodes:
            self.nodes[0].knows_failure = True

    def run_simulation(self):
        self.simulate_failure()
        start_time = time.time()
        first_knowledge_time = None
        all_knowledge_time = None

        while True:
            self.detect_failures()
            current_time = time.time() - start_time

            if first_knowledge_time is None and
any(node.knows_failure for node in self.nodes if node.id not in
self.failed_nodes):
                first_knowledge_time = current_time
```

```
            if all(node.knows_failure for node in self.nodes if
node.id not in self.failed_nodes):
                all_knowledge_time = current_time
                break

            if current_time > 10:  # Ограничение времени симуляции
                break

            time.sleep(self.interval)

        return first_knowledge_time or 0, all_knowledge_time or
current_time, self.bandwidth_usage

class SerfSimulator(BaseSimulator):
    def __init__(self, num_nodes, gossip_interval, gossip_fanout,
packet_loss, node_failures):
        super().__init__(num_nodes, gossip_interval, node_failures)
        self.gossip_fanout = gossip_fanout
        self.packet_loss = packet_loss

    def detect_failures(self):
        for node in self.nodes:
            if node.id not in self.failed_nodes and
node.knows_failure:
                active_nodes = [n for n in range(len(self.nodes)) if
n != node.id and n not in self.failed_nodes]
                targets = random.sample(active_nodes,
min(self.gossip_fanout, len(active_nodes))) if active_nodes else []
                for target in targets:
                    if random.random() > self.packet_loss / 100:
                        self.nodes[target].knows_failure = True
                        self.bandwidth_usage += 1

class HeartbeatSimulator(BaseSimulator):
    def detect_failures(self):
        for node in self.nodes:
            if node.id not in self.failed_nodes:
                for other_node in self.nodes:
                    if other_node.id != node.id:
                        if other_node.id in self.failed_nodes:
                            node.knows_failure = True
                        self.bandwidth_usage += 1

class PingSimulator(BaseSimulator):
```

```
    def detect_failures(self):
        for node in self.nodes:
            if node.id not in self.failed_nodes:
                target = random.choice([n for n in
range(len(self.nodes)) if n != node.id])
                if target in self.failed_nodes:
                    node.knows_failure = True
                self.bandwidth_usage += 1

def run_comparison(num_nodes, node_failures):
    serf_sim = SerfSimulator(num_nodes, 0.2, 3, 0, node_failures)
    heartbeat_sim = HeartbeatSimulator(num_nodes, 0.2,
node_failures)
    ping_sim = PingSimulator(num_nodes, 0.2, node_failures)

    serf_result = serf_sim.run_simulation()
    heartbeat_result = heartbeat_sim.run_simulation()
    ping_result = ping_sim.run_simulation()

    return serf_result, heartbeat_result, ping_result

def plot_comparison(results):
    protocols = ['Serf', 'Heartbeat', 'Ping']
    first_times = [r[0] for r in results]
    all_times = [r[1] for r in results]
    bandwidths = [r[2] for r in results]

    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))

    ax1.bar(protocols, first_times)
    ax1.set_ylabel('Время до первого обнаружения (с)')
    ax1.set_title('Время первого обнаружения отказа')

    ax2.bar(protocols, all_times)
    ax2.set_ylabel('Время до полного обнаружения (с)')
    ax2.set_title('Время полного обнаружения отказов')

    ax3.bar(protocols, bandwidths)
    ax3.set_ylabel('Использование полосы пропускания')
    ax3.set_title('Использование полосы пропускания')

    plt.tight_layout()
    plt.show()

# Запуск сравнения
```

```
num_nodes = 100
node_failures = 5  # 5% узлов отказывают

results = run_comparison(num_nodes, node_failures)
plot_comparison(results)

print("Результаты симуляции:")
for protocol, result in zip(['Serf', 'Heartbeat', 'Ping'], results):
    print(f"\n{protocol}:")
    print(f"Время до 'Хотя бы один узел знает': {result[0]:.2f} с")
    print(f"Время до 'Все живые узлы знают': {result[1]:.2f} с")
    print(f"Использование полосы пропускания: {result[2]} (условных
единиц)")
```

Данный скрипт выводит сравнение производительности Serf с другими протоколами обнаружения отказов (heartbeat или ping-based):