1 a

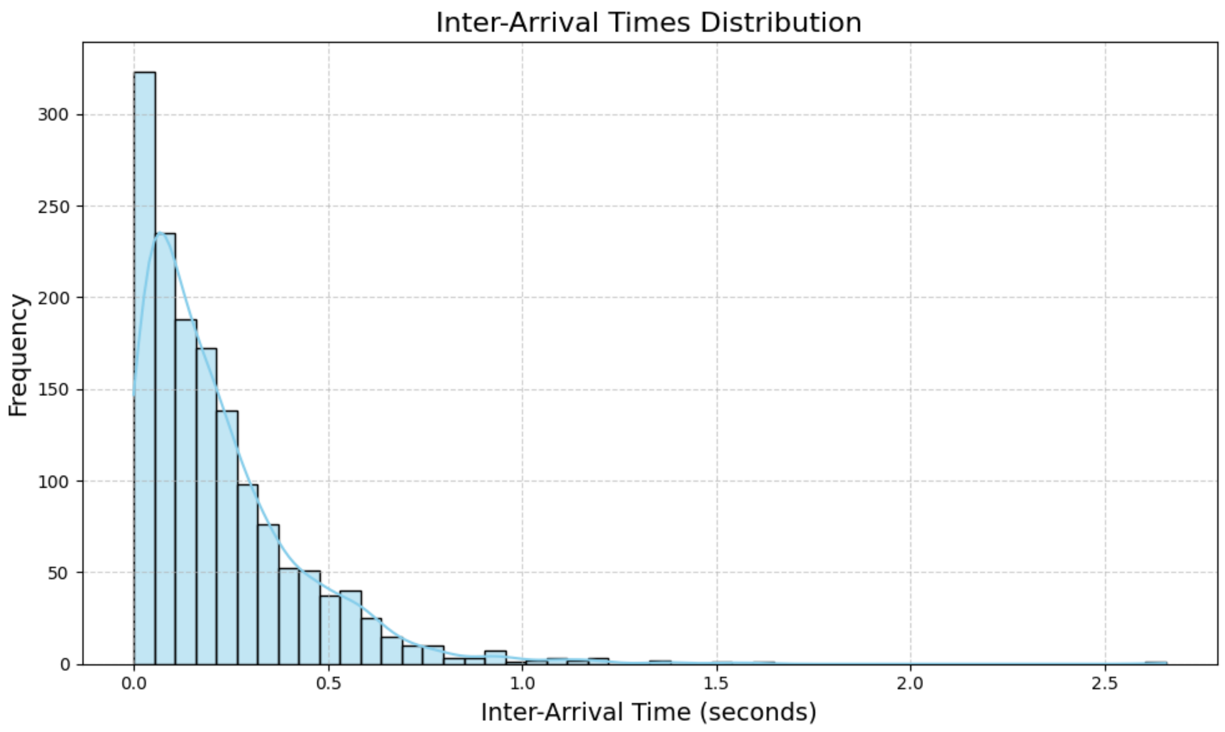

Figure 1: Enter Caption

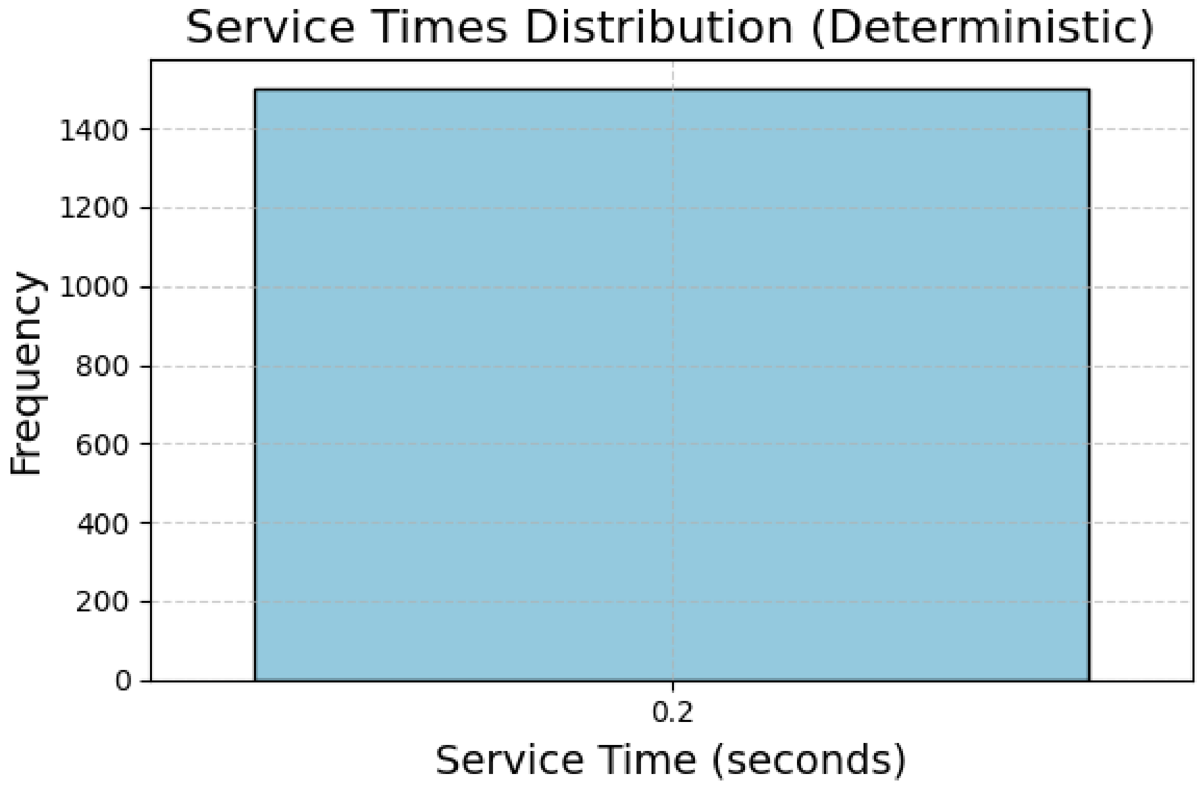

Figure 2: Enter Caption

1. **Distributions**

   - **Inter-Arrival Times:** Exhibits a right-skewed(Exponential) distribution, indicating that most inter-arrival times are clustered towards smaller values with a long tail extending towards larger values.

2. `-d 1`**: Deterministic (Fixed) Distribution**

   - **Service Times:** Deterministic service times, as evidenced by a single bar in the distribution plot, indicating that each request is processed in exactly the same amount of time (e.g., 0.2 seconds).

thus the `-d <dist.number>` parameter controls both inter-arrival times and service times.

2 b

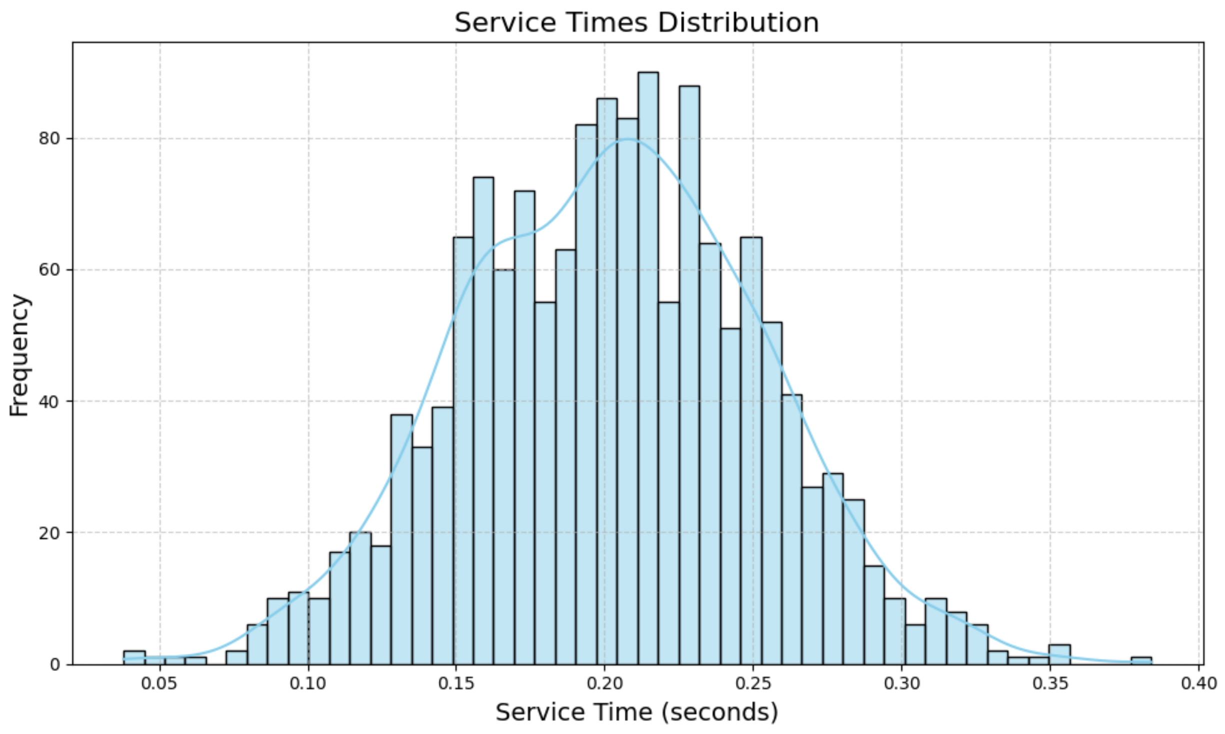

Figure 3: Enter Caption

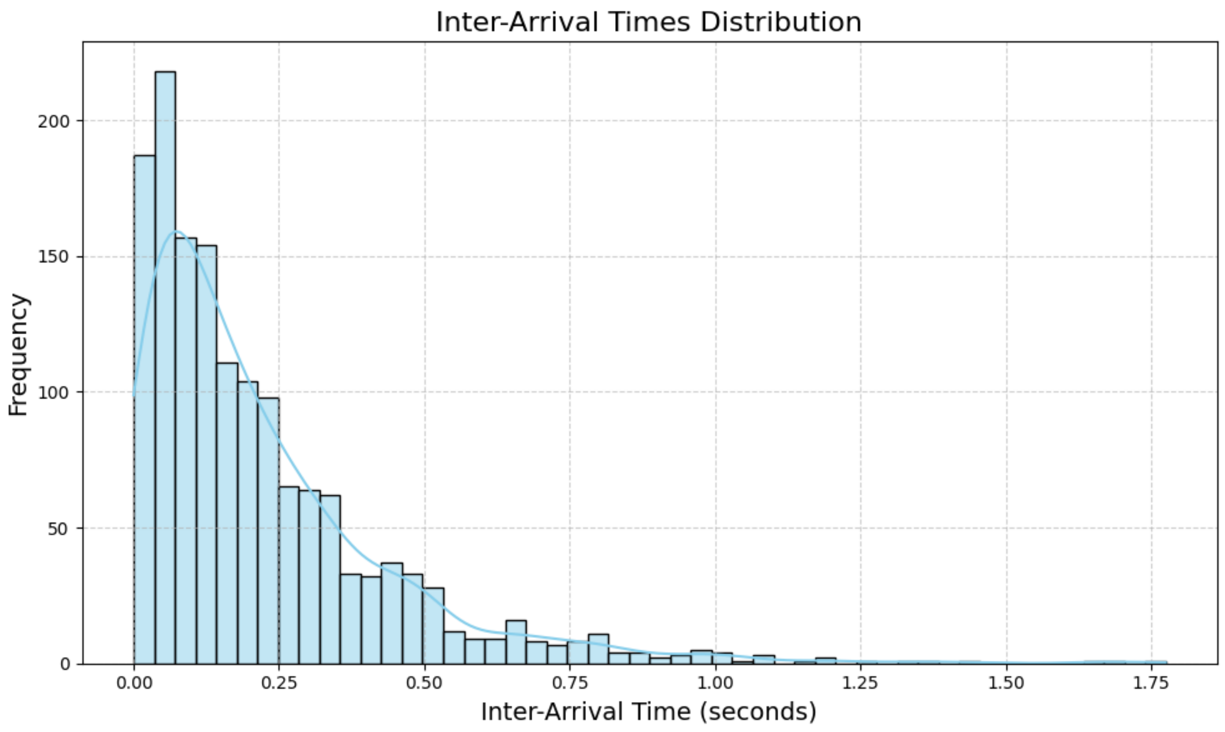

Figure 4: Enter Caption

2.1

- **Mean (μ_S):** Approximately 0.202 seconds
- **Standard Deviation (σ_S):** Approximately 0.050 seconds

1. `-d 2`: Employs an exponential distribution for inter-arrival times and a normal (Gaussian) distribution for service times.
2. **Scope of `-d <dist.number>`:** The `-d <dist.number>` parameter controls service times but not **both** inter-arrival since it keeps Exponential

2.2 script for a and b

]

```
    \begin{verbatim}
    import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import sys
import numpy as np
from scipy import stats

def parse_log_line(line):
    """
    Parses a single log line starting with 'R' and extracts the fields.
    Returns a dictionary with the extracted data.
    """
    # Regular expression to match the R lines
    match = re.match(r'R(\d+):([\d\.]+),([\d\.]+),([\d\.]+),([\d\.]+),([\d\.]+)', line)
    if match:
        request_id = int(match.group(1))
        sent_timestamp = float(match.group(2))
        request_length = float(match.group(3))
        receipt_timestamp = float(match.group(4))
        start_time = float(match.group(5))
        completion_time = float(match.group(6))
        return {
            'RequestID': request_id,
            'SentTimestamp': sent_timestamp,
            'RequestLength': request_length,
            'ReceiptTimestamp': receipt_timestamp,
            'StartTime': start_time,
            'CompletionTime': completion_time
        }
    else:
        return None

def read_log_file(file_path):
    """
    Reads the log file and extracts data from lines starting with 'R'.
    Returns a pandas DataFrame with the extracted data.
    """
    data = []
    with open(file_path, 'r') as file:
        for line in file:
            line = line.strip()
            if line.startswith('R'):
                parsed = parse_log_line(line)
                if parsed:
                    data.append(parsed)
    df = pd.DataFrame(data)
    return df

def calculate_inter_arrival_times(df):
    """
    Calculates inter-arrival times based on ReceiptTimestamp.
    Adds a new column 'InterArrivalTime' to the DataFrame.
    """
    df = df.sort_values('ReceiptTimestamp').reset_index(drop=True)
    df['InterArrivalTime'] = df['ReceiptTimestamp'].diff()
    return df

def plot_inter_arrival_times(data):
    """
    Plots a histogram and density plot for inter-arrival times.
    Saves the plot to 'inter_arrival_times.png'.
    """
    plt.figure(figsize=(10, 6))
    sns.histplot(data, bins=50, kde=True, color='skyblue', edgecolor='black')
    plt.title('Inter-Arrival Times Distribution', fontsize=16)
    plt.xlabel('Inter-Arrival Time (seconds)', fontsize=14)
    plt.ylabel('Frequency', fontsize=14)
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.tight_layout()
    plt.savefig('inter_arrival_times.png')
    plt.close()
    print("Inter-arrival times plot saved as 'inter_arrival_times.png'")

def plot_service_times(df):
    """
    Plots the service times distribution.
    Decides whether to use a histogram, bar plot, or box plot based on data variance.
    Saves the plot accordingly.
    """
    service_length = df['RequestLength']
    variance = service_length.var()
    print(f"Service Times Variance: {variance}")

    if variance < 1e-6:
        # Deterministic - use bar plot or box plot
        # Bar Plot
        plt.figure(figsize=(6, 4))
        sns.countplot(x='RequestLength', data=df, color='skyblue', edgecolor='black')
        plt.title('Service Times Distribution (Deterministic)', fontsize=16)
        plt.xlabel('Service Time (seconds)', fontsize=14)
        plt.ylabel('Frequency', fontsize=14)
        plt.grid(True, linestyle='--', alpha=0.6)
        plt.tight_layout()
        plt.savefig('service_times_bar.png')
        plt.close()
        print("Service times bar plot saved as 'service_times_bar.png'")
    else:
        # Variable - use histogram
        plt.figure(figsize=(10, 6))
        sns.histplot(service_length, bins=50, kde=True, color='skyblue', edgecolor='black')
        plt.title('Service Times Distribution', fontsize=16)
        plt.xlabel('Service Time (seconds)', fontsize=14)
        plt.ylabel('Frequency', fontsize=14)
        plt.grid(True, linestyle='--', alpha=0.6)
        plt.tight_layout()
        plt.savefig('service_times.png')
        plt.close()
        print("Service times plot saved as 'service_times.png'")

def print_summary_statistics(df):
    print("\n--- Summary Statistics ---")
    print(df[['InterArrivalTime', 'RequestLength']].describe())

def main():
    if len(sys.argv) != 2:
        print("Usage: python log_analysis_updated.py <path_to_log_file>")
        sys.exit(1)

    log_file = sys.argv[1]
    print(f"Reading log file: {log_file}")

    # Step 1: Read and parse the log file
    df = read_log_file(log_file)
    if df.empty:
        print("No valid 'R' lines found in the log file.")
        sys.exit(1)
    print(f"Total requests parsed: {len(df)}")

    # Step 2: Calculate inter-arrival times
    df = calculate_inter_arrival_times(df)
    # Drop the first row which has NaN for InterArrivalTime
    df = df.dropna(subset=['InterArrivalTime']).reset_index(drop=True)
    print("Inter-arrival times calculated.")

    # Step 3: Plot Inter-Arrival Times
    plot_inter_arrival_times(df['InterArrivalTime'])

    # Step 4: Plot Service Times
    plot_service_times(df)

    # Step 5: Print Summary Statistics
    print_summary_statistics(df)

    # Optional: Save the processed data to a CSV for further analysis
    df.to_csv('processed_log_data_updated.csv', index=False)
    print("Processed data saved to 'processed_log_data_updated.csv'.")

if __name__ == "__main__":
    main()
```

3 c

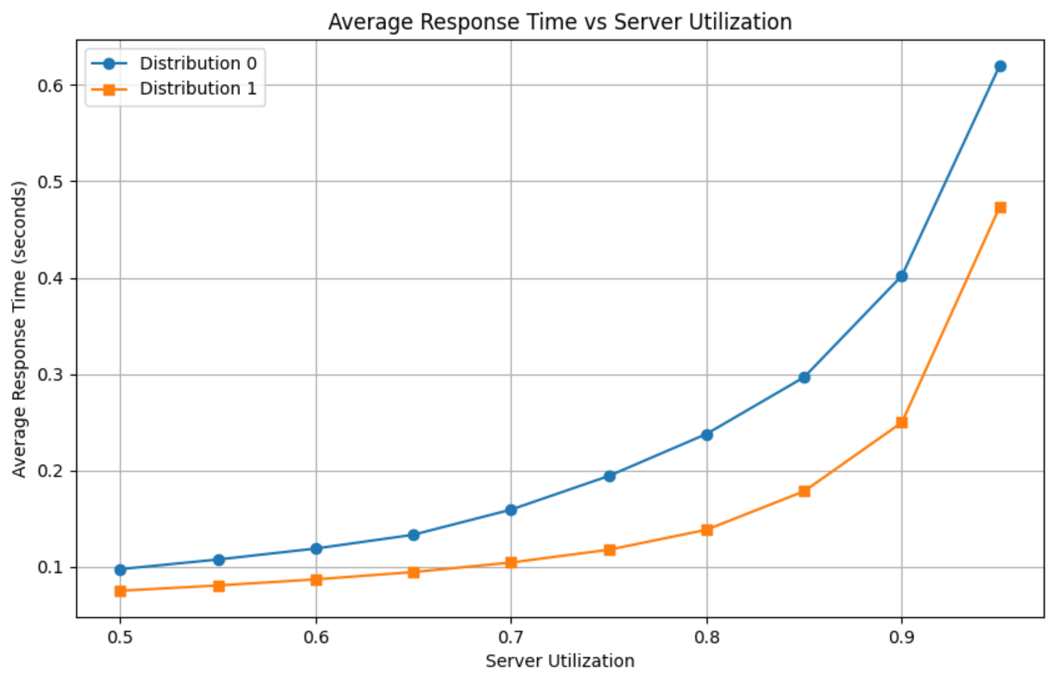

Figure 5: Enter Caption

From the graph of average response time vs. server utilization:

- When server utilization is comparable, **Distribution 1** provides better quality of service, as seen in the lower response times. This implies that Distribution 1 may have less variance in inter-arrival times, making it more efficient for handling higher loads.
- On the other hand, systems using an **exponential distribution** (Distribution 0) tend to degrade in performance as load increases, due to the higher variability and randomness in the arrival process.
- In practical terms, for applications requiring predictable and low-latency responses (e.g., real-time systems), it would be beneficial to avoid exponential distributions and opt for arrival patterns resembling Distribution 1, which maintains better response times under high server utilization.

```
    import os
import re
import matplotlib.pyplot as plt

# Data structures to store results
data = {
    'd0': {},
    'd1': {}
}

# Service time is 0.05 seconds (from request_length in the logs)
service_time = 0.05
```

```
# Regular expression to parse filenames
filename_regex = re.compile(r'server_log_d(?P<distribution>\d+)_arr(?P<arrival_rate>\d+)_.*')

# Get list of all log files in the current directory
log_files = [f for f in os.listdir('.') if f.startswith('server_log')]

for log_file in log_files:
    match = filename_regex.match(log_file)
    if match:
        distribution = 'd' + match.group('distribution')
        arrival_rate = int(match.group('arrival_rate'))

        # Initialize data structures if not already done
        if arrival_rate not in data[distribution]:
            data[distribution][arrival_rate] = {
                'response_times': [],
                'utilization': arrival_rate * service_time
            }

        # Read the log file
        with open(log_file, 'r') as f:
            for line in f:
                line = line.strip()
                if line.startswith('R'):
                    # Parse the line
                    # Format: R<request_id>:<sent_timestamp>,<request_length>,<receipt_timestamp>,<star
                    try:
                        request_id_part, times_part = line.split(':')
                        request_id = int(request_id_part[1:])  # Skip the 'R' character
                        times = times_part.split(',')
                        sent_timestamp = float(times[0])
                        request_length = float(times[1])  # Should be 0.05
                        receipt_timestamp = float(times[2])
                        start_time = float(times[3])
                        completion_time = float(times[4])

                        # Compute response time
                        response_time = completion_time - sent_timestamp

                        # Append to response_times list
                        data[distribution][arrival_rate]['response_times'].append(response_time)
                    except Exception as e:
                        print(f"Error parsing line in {log_file}: {line}")
                        print(e)
                        continue

# Now, compute average response times and prepare data for plotting
arrival_rates = sorted(set(arr for dist in data.values() for arr in dist.keys()))
utilizations = [rate * service_time for rate in arrival_rates]

avg_response_times_d0 = []
avg_response_times_d1 = []
```

```
for arrival_rate in arrival_rates:
    # Distribution d0
    if arrival_rate in data['d0']:
        response_times = data['d0'][arrival_rate]['response_times']
        avg_response_time = sum(response_times) / len(response_times)
        avg_response_times_d0.append(avg_response_time)
    else:
        avg_response_times_d0.append(None)

    # Distribution d1
    if arrival_rate in data['d1']:
        response_times = data['d1'][arrival_rate]['response_times']
        avg_response_time = sum(response_times) / len(response_times)
        avg_response_times_d1.append(avg_response_time)
    else:
        avg_response_times_d1.append(None)

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(utilizations, avg_response_times_d0, 'o-', label='Distribution 0')
plt.plot(utilizations, avg_response_times_d1, 's-', label='Distribution 1')
plt.xlabel('Server Utilization')
plt.ylabel('Average Response Time (seconds)')
plt.title('Average Response Time vs Server Utilization')
plt.legend()
plt.grid(True)
plt.show()
```

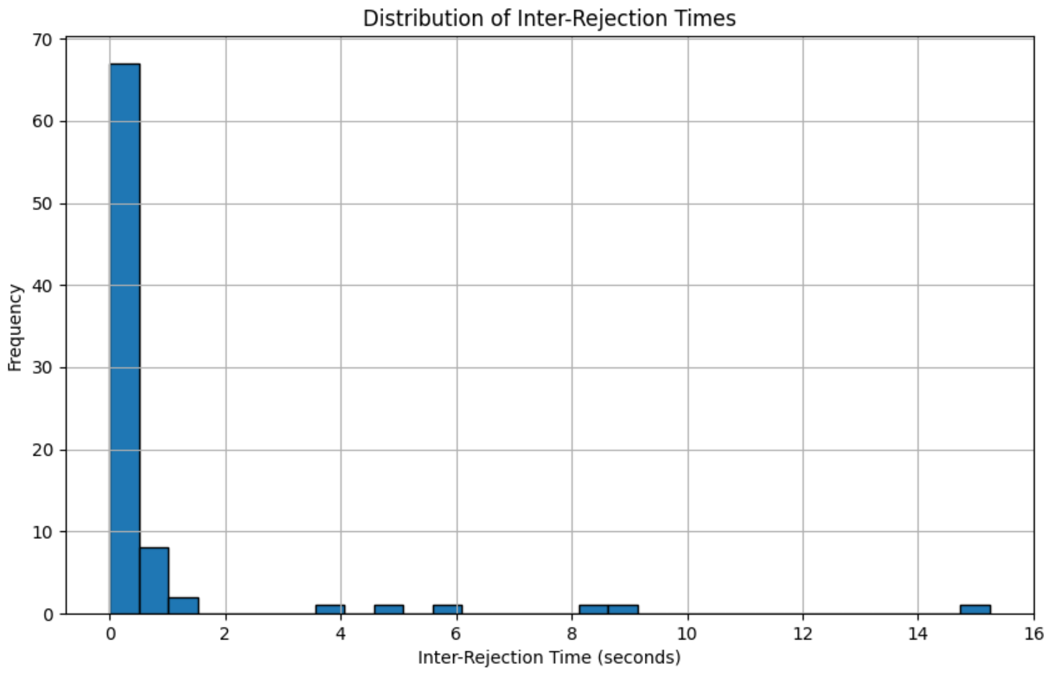

Figure 6: Enter Caption

4 d

4.1

- The distribution is heavily **right-skewed**, with the majority of inter-rejection times concentrated near zero. This suggests that rejected requests tend to occur in rapid succession, with many rejections happening within very short intervals.

5 e

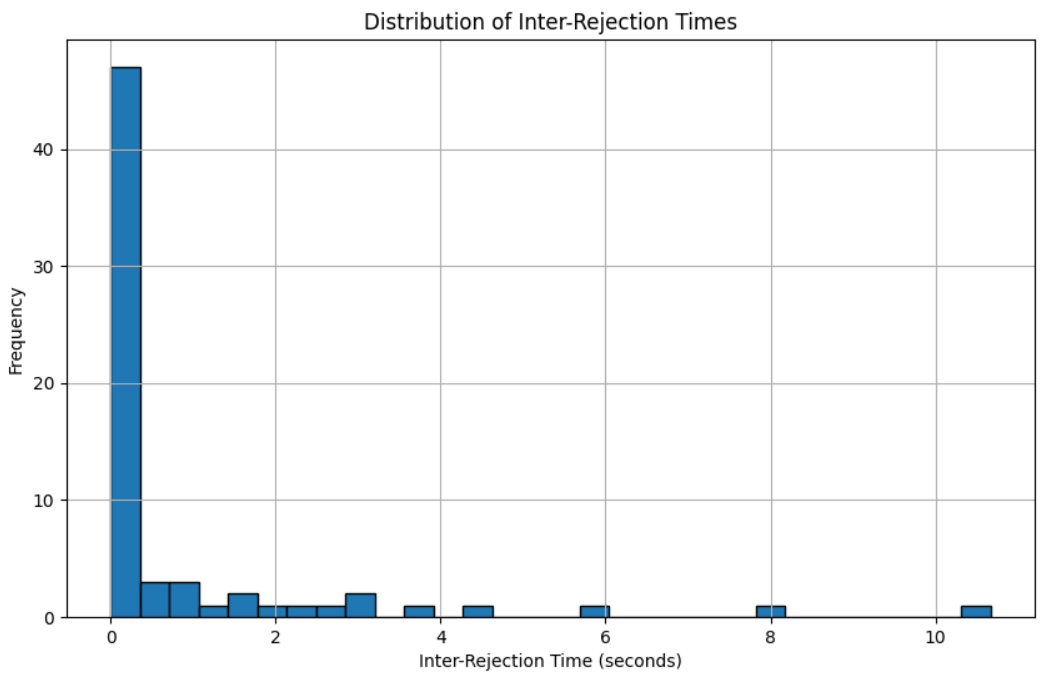

Figure 7: Enter Caption

Rejection Rate

In the current system using `-d 1`:

- Total requests: 1500
- Rejected requests: 67
- Rejection ratio: 4.47%

Compared to the previous system (rejection ratio of 5.60%), the current system using `-d 1` has a lower rejection rate. This indicates that the system is rejecting fewer requests and, therefore, handling a larger portion of the incoming traffic more effectively.

```
    import re
import matplotlib.pyplot as plt

# Initialize counters and lists
total_requests = 0
rejected_requests = 0
rejection_times = []

# Regular expression pattern for the report lines
report_line_pattern = re.compile(
    r'\[#CLIENT#\] R\[(\d+)\]: Sent: ([\d\.]+) Recv: ([\d\.]+) Exp: ([\d\.]+) Len: ([\d\.]+) Rejected:
)
```

```
# Name of your log file
log_file = 'server_lim_log2.txt'  # Replace with your actual filename

# Flag to indicate when we've reached the report section
in_report_section = False

# Read and process the log file
with open(log_file, 'r') as f:
    for line in f:
        line = line.strip()

        # Check if we've reached the report section
        if '==== REPORT ====' in line:
            in_report_section = True
            continue  # Skip the report header line

        if in_report_section:
            # Parse the report lines
            report_match = report_line_pattern.match(line)
            if report_match:
                total_requests += 1
                request_id = int(report_match.group(1))
                sent_timestamp = float(report_match.group(2))
                recv_timestamp = float(report_match.group(3))
                exp_timestamp = float(report_match.group(4))
                length = float(report_match.group(5))
                rejected = report_match.group(6)

                if rejected == 'Yes':
                    rejected_requests += 1
                    rejection_times.append(sent_timestamp)
            else:
                # Ignore lines that don't match the report pattern
                continue

# Calculate the ratio of rejected requests over the total
if total_requests > 0:
    rejection_ratio = rejected_requests / total_requests
else:
    rejection_ratio = 0

print(f"Total Requests: {total_requests}")
print(f"Rejected Requests: {rejected_requests}")
print(f"Rejection Ratio: {rejection_ratio:.2%}")

# Calculate inter-rejection times
inter_rejection_times = []
if len(rejection_times) > 1:
    # Sort the rejection times to ensure chronological order
    rejection_times.sort()
    inter_rejection_times = [
        t2 - t1 for t1, t2 in zip(rejection_times[:-1], rejection_times[1:])
    ]
```

```
# Plot the distribution of inter-rejection times
if inter_rejection_times:
    plt.figure(figsize=(10, 6))
    plt.hist(inter_rejection_times, bins=30, edgecolor='black')
    plt.xlabel('Inter-Rejection Time (seconds)')
    plt.ylabel('Frequency')
    plt.title('Distribution of Inter-Rejection Times')
    plt.grid(True)
    plt.show()
else:
    print("Not enough rejection events to plot inter-rejection times.")
```