simulations

June 9, 2025

[1]:
```python
from simulator import TerminationPolicy, Simulator

import os
import math

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.linear_model import LinearRegression
```

[2]:
```python
# SETTINGS
SEED = 42
WARMUP = False
warmup_events = 500
lambda_ = 1  # arrival_rate
mu = 1.5  # service_rate

os.makedirs("results", exist_ok=True) if not os.path.exists("results") else None
csv_name = (
    f"ex1_p1_mm1_{lambda_}_{mu}_warmup.csv"
    if WARMUP
    else f"mm1_{lambda_}_{mu}_nowarmup.csv"
)
path_to_csv = os.getcwd() + f"/results/{csv_name}.csv"

# delete the file if it exists
if os.path.exists(path_to_csv):
    os.remove(path_to_csv)
```

1 Setup and run the simulator

[3]:
```python
# Termination policy: stop when the simulation time reaches 1,000 time units
policy = (
    TerminationPolicy()
    .add(lambda srv, sched, narr, ndep, abs_t: abs_t >= int(1e3))
    .all()
```

```
)

# Run the simulation
print(f"Running simulation with arrival rate: {lambda_}, service rate: {mu}")
sim = Simulator(
    arrival_rate=lambda_, service_rate=mu, seed=SEED, path_to_csv=path_to_csv
)
sim.run(termination_condition=policy)
sim.report()
```

```
Running simulation with arrival rate: 1, service rate: 1.5
Arrivals: 928
Departures: 926
Final queue length: 1
Server busy: True
Total Arrivals: 928
Total Services: 926
```

2 Plots of Utilization and packets in the system

[4]:
```
# Load the simulation results from the CSV file
data = pd.read_csv(path_to_csv)

if WARMUP:
    # remove first warmup_events
    data = data.iloc[warmup_events:]

# compute total packets in the system (queue + 1 if busy, else queue)
data["total_current_packets"] = data["queue_length"] + data["server_busy"].
 ↪astype(int)

ro = lambda_ / mu
# Theoretical average number of packets in the system (stationary regime)
t_packet_mean = (
    ro / (1 - ro) if ro < 1 else np.inf
)  # if arrival rate >= service rate, the system is unstable (it saturates)

print(
    f"Average number of packets in the system: {data['total_current_packets'].
 ↪mean():.4f} (theoretical: {t_packet_mean:.4f})"
)
# compute the average utilization
data["delta_t"] = data["time"].shift(-1) - data["time"]
data["delta_t"] = data["delta_t"].fillna(0.0)  # put to zero last value


data["cumulative_busy_time"] = (data["server_busy"] * data["delta_t"]).cumsum()
```

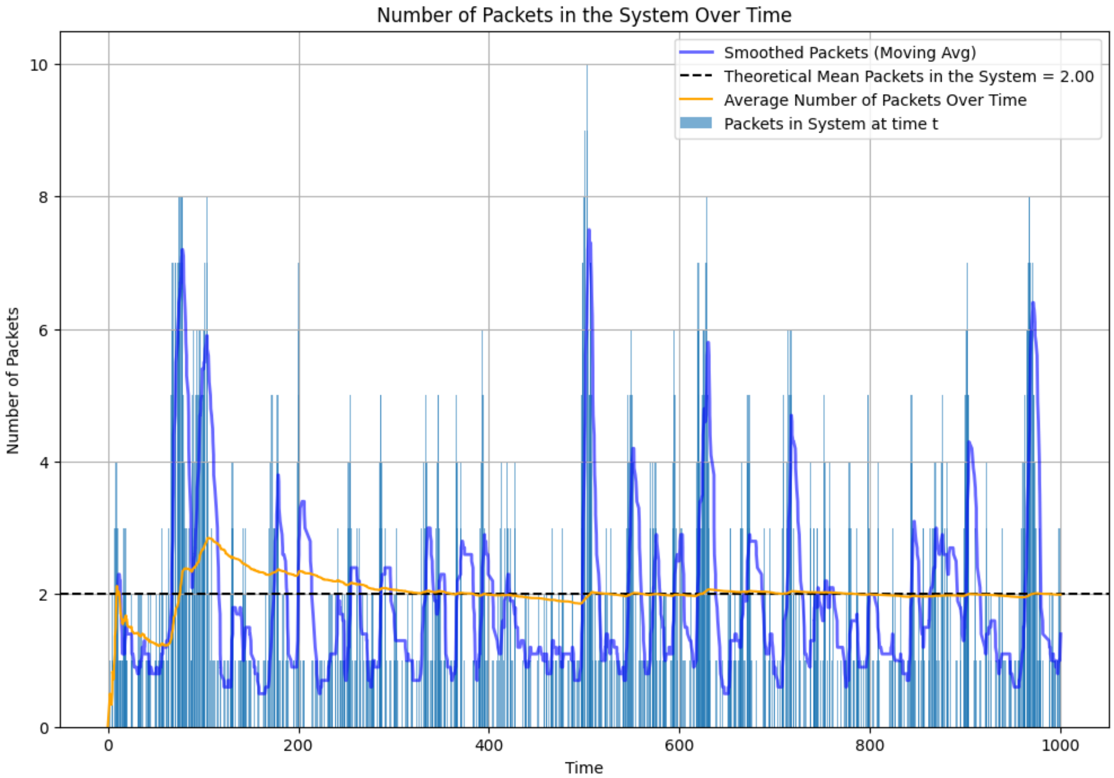
```
data["elapsed_time"] = data["time"] - data["time"].iloc[0]

data["cumulative_utilization"] = data["cumulative_busy_time"] /␣
 ↪data["elapsed_time"]
data["cumulative_utilization"] = data["cumulative_utilization"].fillna(
    0.0
)  ##fill the Nan in first row

emp_util_mean = data["cumulative_utilization"].iloc[-1]

print(f"Utilization: {emp_util_mean:.4f} (theoretical: {lambda_ / mu:.4f})")
```

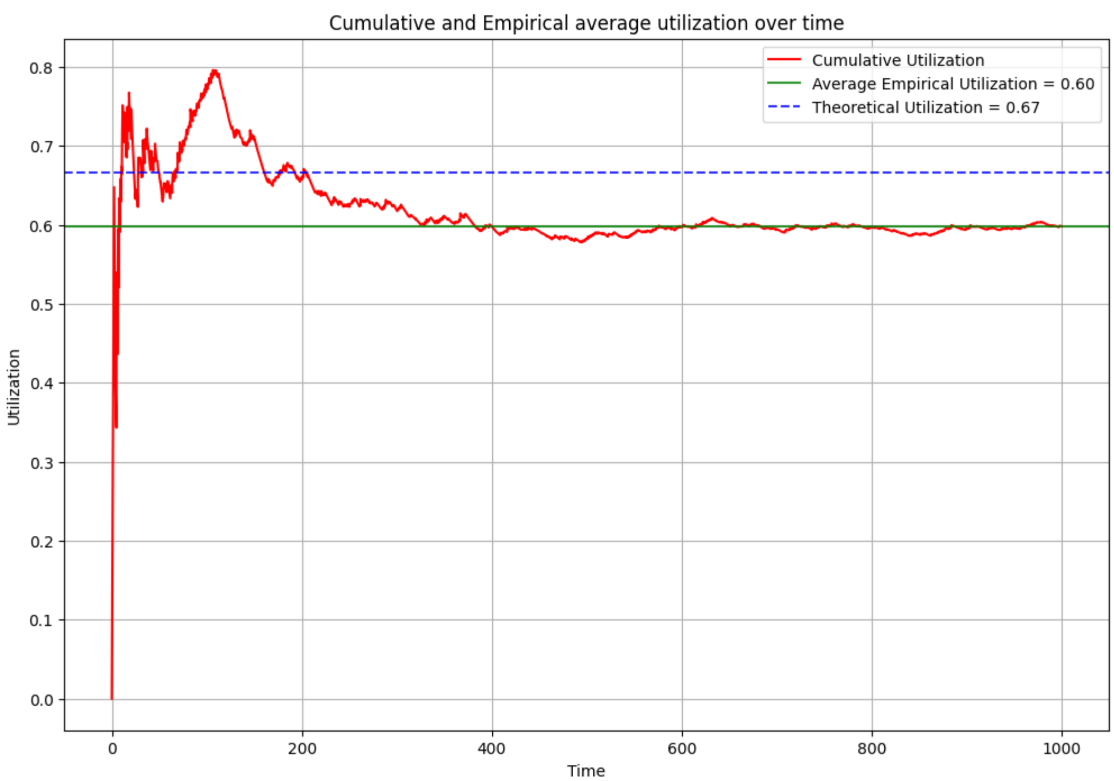
```
Average number of packets in the system: 1.9941 (theoretical: 2.0000)
Utilization: 0.5983 (theoretical: 0.6667)
```

[5]:

```
# Plot the number of packets in the system over time
plt.figure(figsize=(12, 8))
data["smoothed_packets"] = data["total_current_packets"].rolling(window=20).
 ↪mean()
plt.plot(
    data["time"],
    data["smoothed_packets"],
    color="blue",
    label="Smoothed Packets (Moving Avg)",
    linewidth=2,
    alpha=0.6,
)
plt.bar(
    data["time"],
    data["total_current_packets"],
    label="Packets in System at time t",
    alpha=0.6,
)
plt.axhline(
    y=t_packet_mean,
    color="black",
    linestyle="--",
    label=f"Theoretical Mean Packets in the System = {t_packet_mean:.2f}",
)
plt.plot(
    data["time"],
    data["total_current_packets"].expanding().mean(),
    color="orange",
    label="Average Number of Packets Over Time",
)
plt.xlabel("Time")
plt.ylabel("Number of Packets")
```

```python
plt.title("Number of Packets in the System Over Time")
plt.legend()
plt.grid()

plt.figure(figsize=(12, 8))
plt.plot(
    data["time"],
    data["cumulative_utilization"],
    label="Cumulative Utilization",
    color="red",
)
plt.axhline(
    emp_util_mean,
    color="green",
    label=f"Average Empirical Utilization = {emp_util_mean:.2f}",
    alpha=0.8,
)
plt.axhline(
    ro,
    color="blue",
    linestyle="--",
    label=f"Theoretical Utilization = {ro:.2f}",
    alpha=0.8,
)
plt.xlabel("Time")
plt.ylabel("Utilization")
plt.title("Cumulative and Empirical average utilization over time")
plt.legend()
plt.grid()
plt.show()
```


Number of Packets in the System Over Time
Smoothed Packets (Moving Avg)
Theoretical Mean Packets in the System = 2.00
Average Number of Packets Over Time
Packets in System at time t
Number of Packets
Time
0
2
4
6
8
10
200
400
600
800
1000


Cumulative and Empirical average utilization over time
Cumulative Utilization
Average Empirical Utilization = 0.60
Theoretical Utilization = 0.67
Utilization
Time
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
200
400
600
800
1000

3 Running simulation batches to assess the behavior of the system with different parameters

[41]:
```
lamda_values = [0.5, 1.0, 1.5]
mu_values = [0.5, 1.0, 1.5]

NUM_REPLICATIONS = 25
NUM_TIME_UNITS = 1000

seeds = range(SEED, SEED + NUM_REPLICATIONS)

policy = (
    TerminationPolicy()
    .add(lambda srv, sched, narr, ndep, abs_t: abs_t >= NUM_TIME_UNITS)
    .all()
)

# prepare to collect CSV paths
sim_results = []

# run simulations (unchanged)
for lambda_ in lamda_values:
    for mu in mu_values:
        for rep_pis in range(NUM_REPLICATIONS):
            seed = seeds[rep_pis]
            csv_name = f"ex1_p2_mm1_{lambda_}_{mu}_rep_{rep_pis}"
            path_to_csv = os.path.join(os.getcwd(), "results", f"{csv_name}.
↪csv")
            # Store metadata
            sim_results.append(
                {
                    "lambda": lambda_,
                    "mu": mu,
                    "replication": rep_pis,
                    "path": path_to_csv,
                }
            )

            sim = Simulator(
                arrival_rate=lambda_,
                service_rate=mu,
                seed=seed,
                path_to_csv=path_to_csv,
```

```
            )
            sim.run(termination_condition=policy)
```

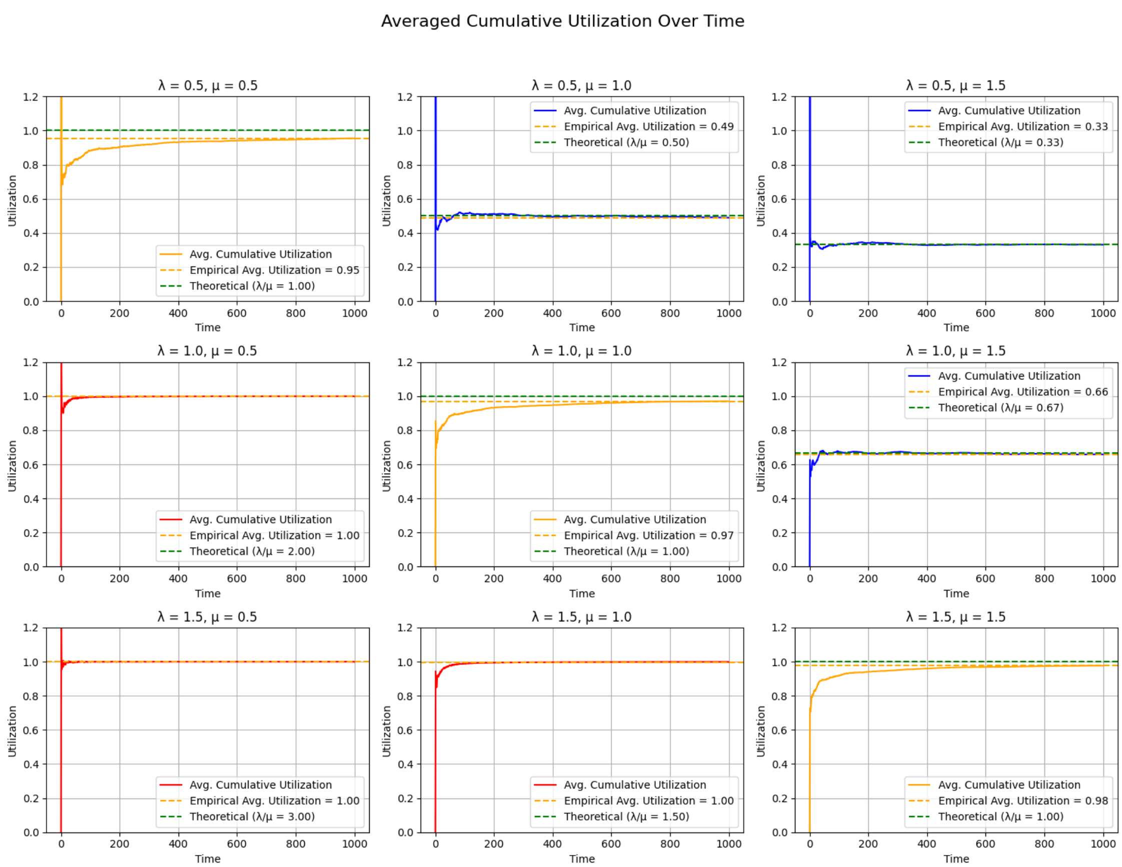
[49]:
```
def get_utils_results(
    grouped_results: dict, grid: np.ndarray, n_drops: int = 0
) -> dict:
    # Store averaged utilization curves for each (lambda, mu) pair
    utils_results = {}

    for (lambda_, mu), paths in grouped_results.items():
        utils_results[(lambda_, mu)] = {}

        utils_over_time_for_rep = np.zeros((len(paths), grid.size))
        utils_for_rep = np.zeros(len(paths))

        for i, csv_file in enumerate(paths):
            data = pd.read_csv(csv_file)
            # Drop the first n_drops rows
            # data = data.iloc[n_drops:]
            # drop the rows where time is less than n_drops
            data = data[data["time"] >= n_drops]

            # Compute delta_t, cumulative_busy_time, elapsed_time
            data["delta_t"] = data["time"].shift(-1) - data["time"]
            data.loc[data.index[-1], "delta_t"] = 0.0

            data["cumulative_busy_time"] = (
                data["server_busy"].astype(int) * data["delta_t"]
            ).cumsum()
            data["elapsed_time"] = data["time"] - data["time"].iloc[0]

            # Compute utilization, avoid divide-by-zero
            utilization = np.where(
                data["elapsed_time"] > 0,
                data["cumulative_busy_time"] / data["elapsed_time"],
                0.0,
            )

            utils_for_rep[i] = utilization[-1]

            # Interpolate onto the fixed grid
            utils_over_time_for_rep[i, :] = np.interp(
                grid, data["time"].values, utilization
            )

        utils_results[(lambda_, mu)]["utils_ot_rep"] = utils_over_time_for_rep
```

```
        # Average out the replications' avg utilization curves
        avg_util_curve = utils_over_time_for_rep.mean(axis=0)

        utils_results[(lambda_, mu)]["utils_ot_avg"] = avg_util_curve
        utils_results[(lambda_, mu)]["utils_for_rep"] = utils_for_rep

    return utils_results
```

3.1 Visualization of the simulation results to identify the initialization phase

[50]:
```
# Define a common time grid
grid = np.arange(0, NUM_TIME_UNITS + 1)  # integer times

# Group CSV results by (lambda, mu)
grouped_results = {}
for entry in sim_results:
    key = (entry["lambda"], entry["mu"])
    if key not in grouped_results:
        grouped_results[key] = []
    grouped_results[key].append(entry["path"])

utils_results = get_utils_results(grouped_results, grid)

# Determine the grid size for subplots
num_plots = len(utils_results)
cols = 3  # Choose how many columns you want
rows = math.ceil(num_plots / cols)

fig, axes = plt.subplots(rows, cols, figsize=(5 * cols, 4 * rows),␣
 ↪squeeze=False)

# Plot each utilization curve in a subplot
for idx, ((lambda_, mu), results) in enumerate(utils_results.items()):
    row, col = divmod(idx, cols)
    ax = axes[row][col]

    color = "blue" if mu > lambda_ else "red" if mu < lambda_ else "orange"

    theoretical_util = lambda_ / mu

    ax.plot(
        grid, results["utils_ot_avg"], color=color, label="Avg. Cumulative␣
 ↪Utilization"
    )
    ax.axhline(
        results["utils_for_rep"].mean(),
        color="orange",
```

```
        linestyle="--",
        label=f"Empirical Avg. Utilization = {results['utils_for_rep'].mean():.
↪2f}",
    )
    ax.axhline(
        theoretical_util,
        color="green",
        linestyle="--",
        label=f"Theoretical ( /  = {theoretical_util:.2f})",
    )
    ax.set_ylim(0, 1.2)
    ax.set_xlabel("Time")
    ax.set_ylabel("Utilization")
    ax.set_title(f"  = {lambda_},   = {mu}")
    ax.grid()
    ax.legend()

fig.suptitle("Averaged Cumulative Utilization Over Time", fontsize=16)
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

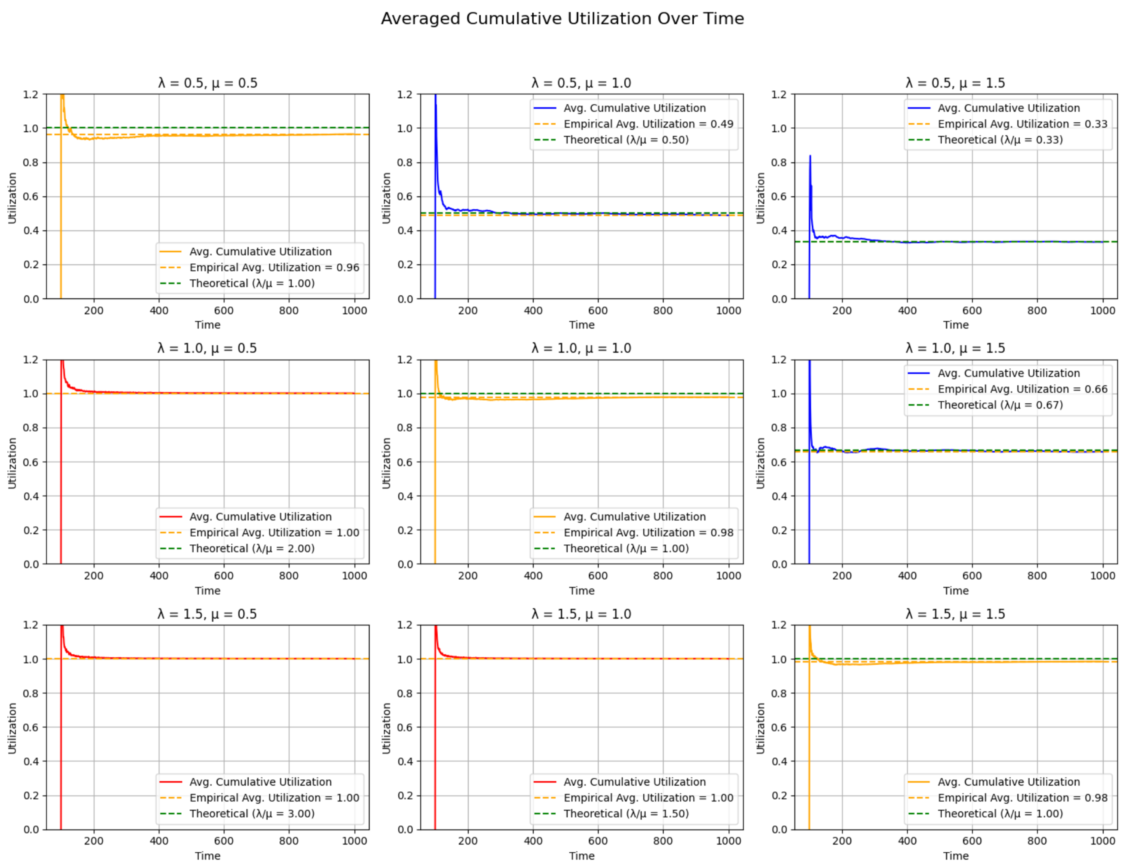


3.2 Removing the initialization phase from the results

[61]:
```
# Drop the first events
NUM_DROPPED_EVENTS = 100

truncated_grid = grid[NUM_DROPPED_EVENTS:]

truncated_results = get_utils_results(
    grouped_results, truncated_grid, n_drops=NUM_DROPPED_EVENTS
)
```

[62]:
```
fig, axes = plt.subplots(rows, cols, figsize=(5 * cols, 4 * rows),␣
↪squeeze=False)

# Plot each utilization curve in a subplot
for idx, ((lambda_, mu), results) in enumerate(truncated_results.items()):
    row, col = divmod(idx, cols)
    ax = axes[row][col]

    color = "blue" if mu > lambda_ else "red" if mu < lambda_ else "orange"

    theoretical_util = lambda_ / mu

    ax.plot(
        truncated_grid,
        results["utils_ot_avg"],
        color=color,
        label="Avg. Cumulative Utilization",
    )
    ax.axhline(
        results["utils_for_rep"].mean(),
        color="orange",
        linestyle="--",
        label=f"Empirical Avg. Utilization = {results['utils_for_rep'].mean():.
↪2f}",
    )
    ax.axhline(
        theoretical_util,
        color="green",
        linestyle="--",
        label=f"Theoretical ( /  = {theoretical_util:.2f})",
    )
    ax.set_ylim(0, 1.2)
    ax.set_xlabel("Time")
    ax.set_ylabel("Utilization")
    ax.set_title(f" = {lambda_},  = {mu}")
    ax.grid()
```

```
    ax.legend()

fig.suptitle("Averaged Cumulative Utilization Over Time", fontsize=16)
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



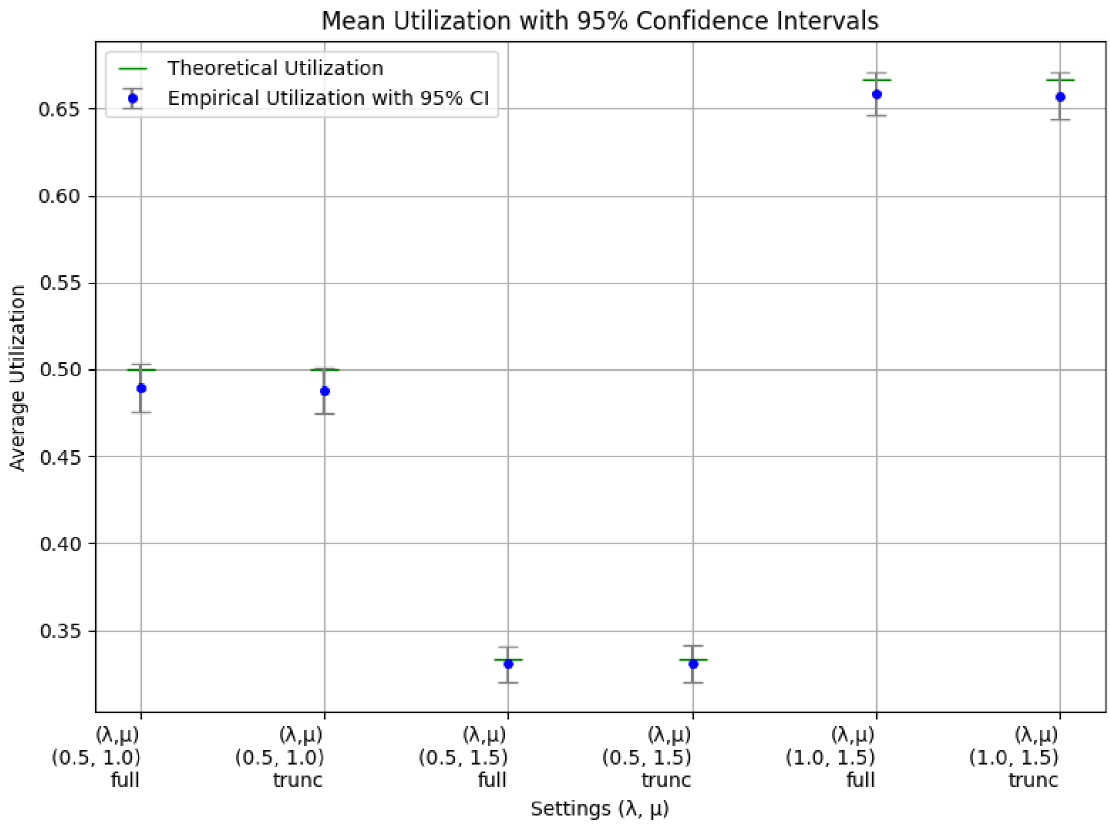
[53]:
```
def compute_mean_ci(data: np.ndarray, confidence: float = 0.95):
    """
    Compute the confidence interval for the mean of the data.
    """
    n = len(data)
    if n < 2:
        return np.nan, np.nan  # Not enough data to compute CI

    mean = np.mean(data)
    var = sum((data - mean) ** 2) / (n - 1)  # Sample variance
    se = np.sqrt(var / n)
    t_quantile = stats.t.ppf((1 + confidence) / 2, df=n - 1)  # t-distribution␣
 ↪quantile
    margin = t_quantile * se
```

```python
    return mean - margin, mean + margin
```

[63]:
```python
summary_rows = []

for (lambda_, mu), results in utils_results.items():
    # convert to numpy array for stats
    utils = results["utils_for_rep"]

    ci_low, ci_high = compute_mean_ci(utils)

    truncated_utils = truncated_results[(lambda_, mu)]["utils_for_rep"]

    truncated_ci_low, truncated_ci_high = compute_mean_ci(truncated_utils)

    # collect
    summary_rows.append(
        {
            "lambda": lambda_,
            "mu": mu,
            "n_reps": len(utils),
            "theoretical": lambda_ / mu,
            "full": {
                "mean_util": np.mean(utils),
                "ci95_lo": ci_low,
                "ci95_hi": ci_high,
            },
            "truncated": {
                "mean_util": np.mean(truncated_utils),
                "ci95_lo": truncated_ci_low,
                "ci95_hi": truncated_ci_high,
            },
        }
    )
```

[65]:
```python
# Filter out rows where lambda is greater than mu
filtered_summaries = [row for row in summary_rows if row["lambda"] < row["mu"]]

# Create labels
labels = []
y, yerr_lo, yerr_hi = [], [], []
theoretical_utils = []
for row in filtered_summaries:
    base_label = f"( , )\n({row['lambda']}, {row['mu']})\n"
    labels.append(base_label + "full")
    labels.append(base_label + "trunc")
    y.append(row["full"]["mean_util"])
```

```python
    y.append(row["truncated"]["mean_util"])
    yerr_lo.append(row["full"]["mean_util"] - row["full"]["ci95_lo"])
    yerr_lo.append(row["truncated"]["mean_util"] - row["truncated"]["ci95_lo"])
    yerr_hi.append(row["full"]["ci95_hi"] - row["full"]["mean_util"])
    yerr_hi.append(row["truncated"]["ci95_hi"] - row["truncated"]["mean_util"])

    theoretical_utils.append(row["theoretical"])
    theoretical_utils.append(row["theoretical"])

yerr = np.array([yerr_lo, yerr_hi])
x = np.arange(len(labels))

# Compute error bars (asymmetrical)


# Plot
plt.figure(figsize=(8, 6))
plt.plot(x, theoretical_utils, "g_", markersize=15, label="Theoretical␣
↪Utilization")
plt.errorbar(
    x,
    y,
    yerr=yerr,
    fmt="o",
    capsize=5,
    markersize=4,
    color="blue",
    ecolor="gray",
    label="Empirical Utilization with 95% CI",
)

plt.xticks(x, labels, ha="right")
# plt.ylim(0, 1.2)
plt.ylabel("Average Utilization")
plt.xlabel("Settings ( ,  )")
plt.title("Mean Utilization with 95% Confidence Intervals")
plt.grid(True)
plt.tight_layout()
plt.legend()
plt.show()
```



4 Exercise 2: estimate the average transit time of packets in the system

[67]:
```
NUM_REPLICATIONS = 100
NUM_TIME_UNITS = 10000

# lambda_ = 1.0
lambda_ = 0.5
mu = 1.5


seeds = range(SEED, SEED + NUM_REPLICATIONS)

policy = (
    TerminationPolicy()
    .add(lambda srv, sched, narr, ndep, abs_t: abs_t >= NUM_TIME_UNITS)
    .all()
)
```

```python
# prepare to collect CSV paths
sim_results = []

for rep_pis in range(NUM_REPLICATIONS):
    seed = seeds[rep_pis]
    csv_name = f"ex2_mm1_{lambda_}_{mu}_rep_{rep_pis}"
    path_to_csv = os.path.join(os.getcwd(), "results", f"{csv_name}.csv")
    # Store metadata
    sim_results.append(path_to_csv)

    sim = Simulator(
        arrival_rate=lambda_,
        service_rate=mu,
        seed=seed,
        path_to_csv=path_to_csv,
    )
    sim.run(termination_condition=policy)
```

[68]:

```python
DROP_EVENTS = 0
```

4.1 Naive approach: average the transit times of all packets

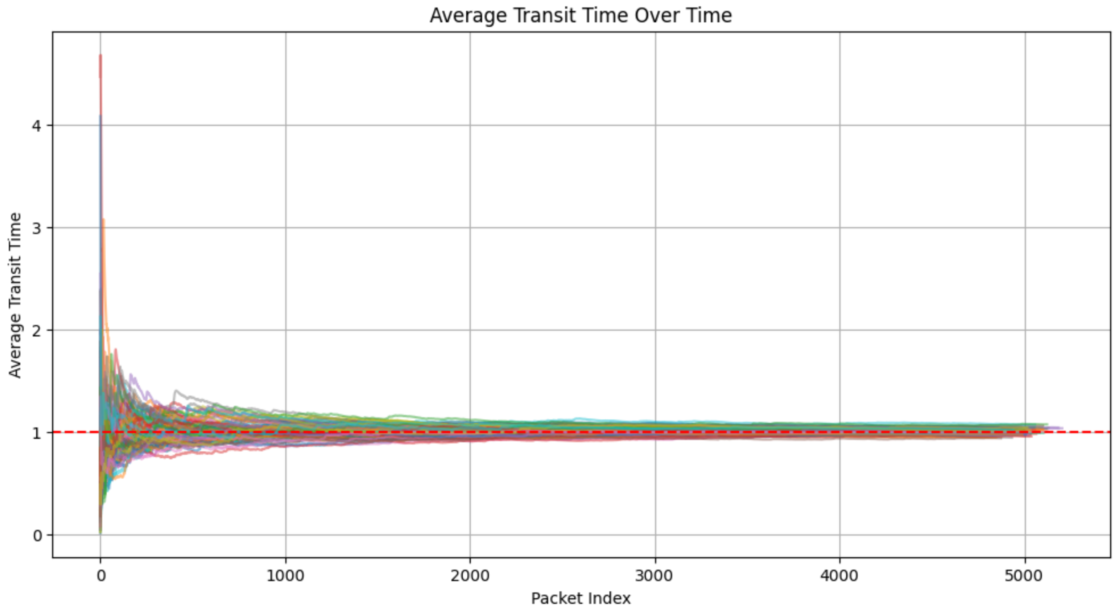
[69]:

```python
def get_transit_times(sim_results, drop_num=0):
    """
    Compute transit times from simulation results.

    :param sim_results: List of CSV file paths containing simulation results.
    :param drop_num: Number of initial events to drop from each simulation.
    :return: List of lists containing transit times for each replication.
    """
    transit_times_for_all_reps = np.empty(len(sim_results), dtype=object)

    for rep_idx, csv_file in enumerate(sim_results):
        data = pd.read_csv(csv_file)

        transit_times = []

        # Store arrival times to then compute transit times when the packet is␣
↪served
        arrival_times = []

        # Store previous arr/dep counts to understand the current type of event
        prev_arrivals = data["arrivals"].iloc[0]
        prev_departures = data["departures"].iloc[0]

        for i, row in data.iterrows():
            if prev_arrivals < row["arrivals"]:
```

```
                # This is an arrival event
                prev_arrivals += 1

                arrival_times.append(row["time"])
            elif prev_departures < row["departures"]:
                # This is a departure event
                prev_departures += 1

                cur_packet_at = arrival_times.pop(0)  # Get the first arrival␣
 ↪time
                if i >= drop_num:  # drop the first `drop_num` events
                    transit_time = row["time"] - cur_packet_at
                    transit_times.append(transit_time)

        transit_times_for_all_reps[rep_idx] = transit_times

    return transit_times_for_all_reps
```

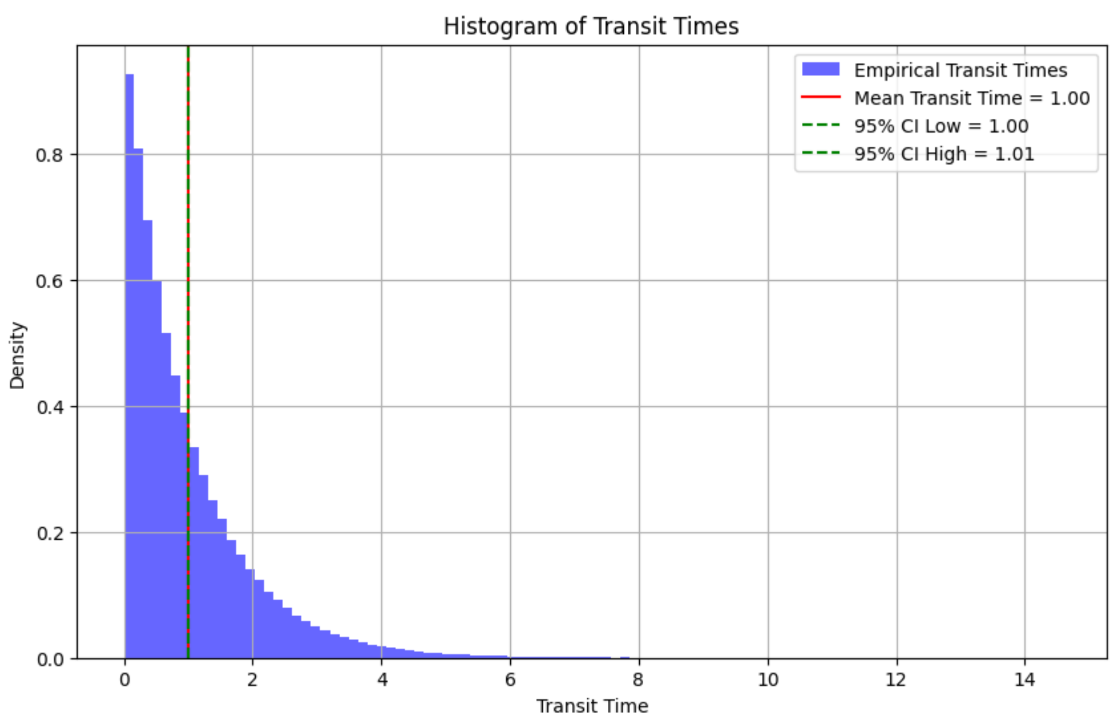
[70]:

```
plt.figure(figsize=(12, 6))

transit_times_for_rep = get_transit_times(sim_results, drop_num=DROP_EVENTS)

for rep_pis in transit_times_for_rep:
    avg_transit_time_over_time = np.cumsum(rep_pis) / np.arange(1, len(rep_pis)␣
 ↪+ 1)
    plt.plot(
        avg_transit_time_over_time, alpha=0.5, label="Cumulative Mean Transit␣
 ↪Time"
    )

    # plt.plot(rep, alpha=0.1, label="Replication Transit Times")

plt.axhline(
    y=1 / (mu - lambda_),
    label="Theoretical Mean Transit Time",
    color="red",
    linestyle="--",
)

plt.xlabel("Packet Index")
plt.ylabel("Average Transit Time")
plt.title("Average Transit Time Over Time")
plt.grid()
plt.show()
```



[71]:
```
# Plot the histogram of transit times

all_transit_times = np.concatenate(transit_times_for_rep)

mean_tt_for_rep = np.array([np.mean(rep) for rep in transit_times_for_rep])

mean_ci_low, mean_ci_high = compute_mean_ci(mean_tt_for_rep)

naive_results = {
    "mean": np.mean(mean_tt_for_rep),
    "values": mean_tt_for_rep,
    "cis": (mean_ci_low, mean_ci_high),
    "label": "Naive Mean Estimator",
}

plt.figure(figsize=(10, 6))
plt.hist(
    all_transit_times,  # Use the flattened array of transit times
    bins=100,
    density=True,
    alpha=0.6,
    color="blue",
    label="Empirical Transit Times",
)
plt.axvline(
    mean_tt_for_rep.mean(),
```

```
    color="red",
    label=f"Mean Transit Time = {mean_tt_for_rep.mean():.2f}",
)
plt.axvline(
    mean_ci_low,
    color="green",
    linestyle="--",
    label=f"95% CI Low = {mean_ci_low:.2f}",
)
plt.axvline(
    mean_ci_high,
    color="green",
    linestyle="--",
    label=f"95% CI High = {mean_ci_high:.2f}",
)
plt.xlabel("Transit Time")
plt.ylabel("Density")
plt.title("Histogram of Transit Times")
plt.grid()
plt.legend()
plt.show()
```



4.2 Control variates: use the average number of packets in the queue as a control variate to reduce variance

[72]:
```python
def get_packets_in_queue(sim_results, drop_num=0):
    packets_inq_for_rep = np.empty(len(sim_results), dtype=object)

    for rep_idx, csv_file in enumerate(sim_results):
        data = pd.read_csv(csv_file)
        data = data.iloc[drop_num:].copy()

        data["time"] = (
            data["time"] - data["time"].iloc[0]
        )  # normalize time to start from 0

        data["prev_queue"] = data["queue_length"].shift(1).fillna(0)
        data["delta_t"] = data["time"].shift(-1) - data["time"]
        data.loc[data.index[-1], "delta_t"] = 0.0  # last value is zero

        data["cum_queue_length"] = (data["prev_queue"] * data["delta_t"]).
 ↪cumsum()

        data["avg_queue_length"] = data["cum_queue_length"] / data["time"]
        data.loc[data.index[0], "avg_queue_length"] = 0.0  # first value is zero

        data = data.copy()
        data = data[["time", "queue_length", "avg_queue_length"]]

        packets_inq_for_rep[rep_idx] = data

    return packets_inq_for_rep
```

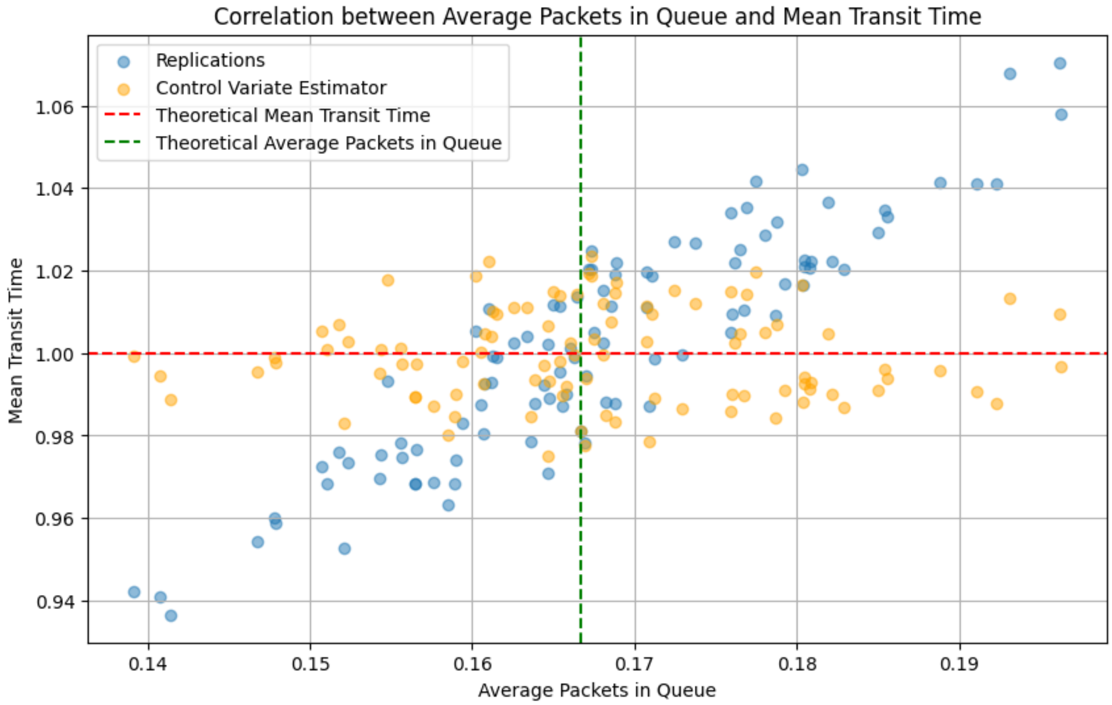
[81]:
```python
packets_in_queue_for_rep = get_packets_in_queue(sim_results,␣
 ↪drop_num=DROP_EVENTS)

# Compute the average number of packets in the queue for each replication
avg_packets_inq = np.array(
    [
        rep["avg_queue_length"].iloc[
            -1
        ]  # Get the last value of avg_queue_length for each replication
        for rep in packets_in_queue_for_rep
    ],
    dtype=object,
)
print(f"Average packets in queue: {avg_packets_inq.mean():.4f}")

# for rep in packets_in_queue_for_rep:
```

```
# Compute the theoretical average number of packets in the queue
rho = lambda_ / mu
theoretical_piq = (rho**2) / (1 - rho)

print(f"Theoretical average packets in queue: {theoretical_piq:.4f}")

# Fit a linear regression model
model = LinearRegression()
model.fit(avg_packets_inq.reshape(-1, 1), mean_tt_for_rep)
b = model.coef_[0]

# Compute the estimator with the cv
tt_cv_lr = mean_tt_for_rep - b * (avg_packets_inq - theoretical_piq)

# # Classic approach
# cov = np.cov(mean_tt_for_rep, avg_packets_inq)[0][1]
# piq_var = np.var(avg_packets_inq, ddof=1)  # sample variance
# c = cov / piq_var
# tt_cv = mean_tt_for_rep - c * (avg_packets_inq - theoretical_piq)

cv_results = {
    "mean": np.mean(tt_cv_lr),
    "values": tt_cv_lr,
    "cis": compute_mean_ci(tt_cv_lr),
    "label": "Control Variate Estimator",
}
```

```
Average packets in queue: 0.1678
Theoretical average packets in queue: 0.1667
```

[88]:
```
# Plot the correlation between average packets in queue and mean transit time
plt.figure(figsize=(10, 6))
plt.scatter(avg_packets_inq, mean_tt_for_rep, alpha=0.5, label="Replications")
plt.scatter(
    avg_packets_inq,
    tt_cv_lr,
    alpha=0.5,
    label="Control Variate Estimator",
    color="orange",
)
plt.axhline(
    y=1 / (mu - lambda_),
    label="Theoretical Mean Transit Time",
    color="red",
    linestyle="--",
)
```

```python
plt.axvline(
    x=theoretical_piq,
    label="Theoretical Average Packets in Queue",
    color="green",
    linestyle="--",
)
plt.xlabel("Average Packets in Queue")
plt.ylabel("Mean Transit Time")
plt.title("Correlation between Average Packets in Queue and Mean Transit Time")
plt.grid()
plt.legend()
plt.show()
```



4.3 Post-stratification: use the number of packets in the queue to stratify the results and reduce variance

[89]:
```python
def get_pis_at_arrival(sim_results, drop_num=0):
    """
    Compute number of packets in the system ahead of a packet arrival.

    :param sim_results: List of CSV file paths containing simulation results.
    :param drop_num: Number of initial events to drop from each simulation.
    :return: List of lists containing number of packets in the system at the␣
    ↪time of each packet arrival for each replication.
```

```
    """
    packets_in_sys_for_rep = np.empty(len(sim_results), dtype=object)

    for rep_idx, csv_file in enumerate(sim_results):
        data = pd.read_csv(csv_file)

        pis_at_arrival = []  # number of packets in queue when a packet arrive

        # Store previous arr/dep counts to understand the current type of event
        prev_arrivals = data["arrivals"].iloc[0]
        prev_departures = data["departures"].iloc[0]

        n_pis_to_drop = 0
        for i, row in data.iterrows():
            if i == drop_num - 1:
                n_to_keep = row["queue_length"] + (1 if row["server_busy"] else␣
↪0)
                n_pis_to_drop = row["arrivals"] - n_to_keep

            if prev_arrivals < row["arrivals"]:
                # This is an arrival event
                prev_arrivals += 1

                pis_at_arrival.append(
                    row["queue_length"] + int(row["server_busy"]) - 1
                )  # packets in queue + the one that is being served - the one␣
↪that just arrived
            elif prev_departures < row["departures"]:
                # This is a departure event
                prev_departures += 1

        unserved_packets = data["queue_length"].iloc[-1] + int(
            data["server_busy"].iloc[-1]
        )
        # Drop the last unserved packets if any
        if unserved_packets > 0:
            pis_at_arrival = pis_at_arrival[:-unserved_packets]
        # Drop the first `drop_num` events
        pis_at_arrival = pis_at_arrival[n_pis_to_drop:]

        packets_in_sys_for_rep[rep_idx] = np.array(pis_at_arrival)

    return packets_in_sys_for_rep
```

[90]:
```
pis_at_arrival_for_rep = get_pis_at_arrival(sim_results, drop_num=DROP_EVENTS)
```

[91]:
```python
# Compute strata for each replication

strata_for_rep = []

for i, rep_pis in enumerate(pis_at_arrival_for_rep):
    # Create strata from 0 to max pis_at_arrival and append to the list
    num_strat = rep_pis.max() + 1
    stratum = np.arange(0, num_strat)

    occurrences = np.zeros(num_strat, dtype=int)
    for pis in rep_pis:
        occurrences[pis] += 1

    sum_tt_stratum = np.zeros(num_strat, dtype=float)
    for j, tt in enumerate(transit_times_for_rep[i]):
        pis = rep_pis[j]
        sum_tt_stratum[pis] += tt

    mean_tt_stratum = sum_tt_stratum / occurrences

    strata_for_rep.append((stratum, occurrences, mean_tt_stratum))
```

[92]:
```python
def pis_probability(rho: float, n_pis: int) -> float:
    """
    Compute the probability of having `n_pis` packets in the system in a M/M/1␣
↪queueing system.
    """
    return (rho**n_pis) * (1 - rho)
```

[93]:
```python
# Compute post-stratification estimator


ps_tt_for_rep = np.zeros(len(pis_at_arrival_for_rep))

for i, (stratum, occurrences, mean_tt_stratum) in enumerate(strata_for_rep):
    probabilities = np.array([pis_probability(rho, pis) for pis in stratum])
    probabilities[-1] = (
        rho ** stratum[-1]
    )  # last stratum is the maximum, so we use rho^n_pis

    ps_tt_for_rep[i] = np.sum(probabilities * mean_tt_stratum)


print(f"Post-stratification estimator: {ps_tt_for_rep.mean():.4f}")

ps_results = {
    "mean": np.mean(ps_tt_for_rep),
```

```
    "values": ps_tt_for_rep,
    "cis": compute_mean_ci(ps_tt_for_rep),
    "label": "Post-Stratification Estimator",
}
```

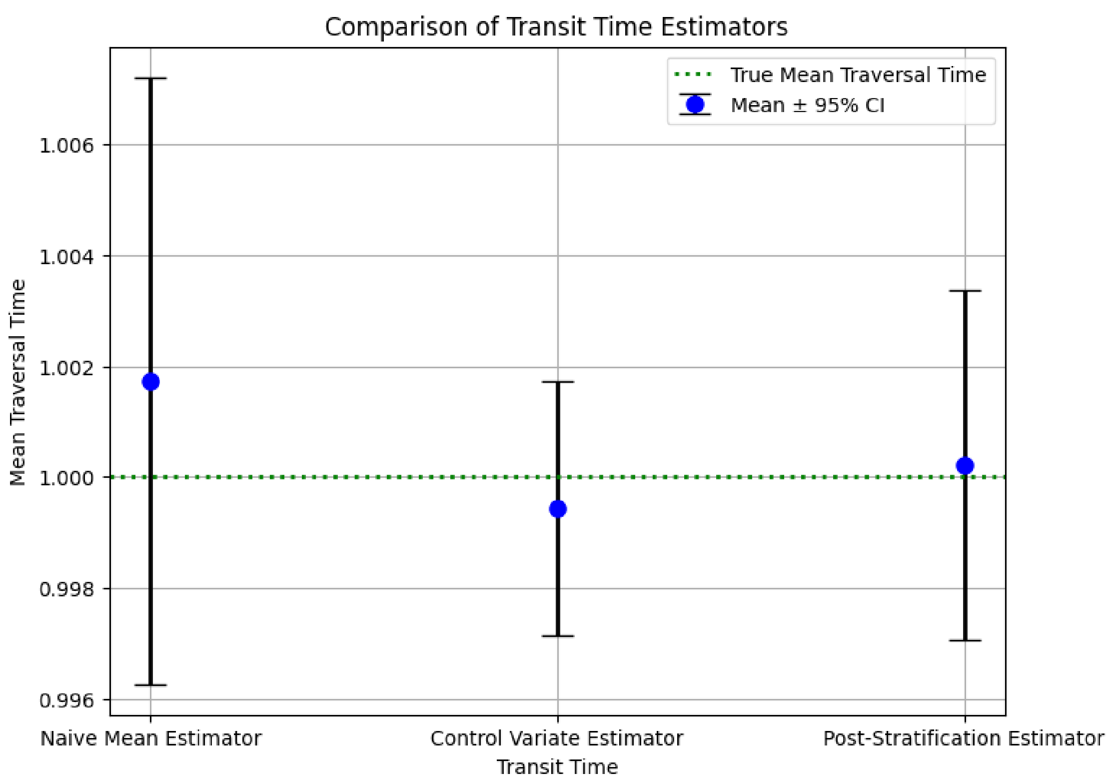
```
Post-stratification estimator: 1.0002
```

4.4 Compare the different estimatimators

[94]:
```
plt.figure(figsize=(8, 6))

estimators = [naive_results, cv_results, ps_results]


plt.errorbar(
    x=range(len(estimators)),
    y=[est["mean"] for est in estimators],
    yerr=np.array(
        [
            (est["mean"] - est["cis"][0], est["cis"][1] - est["mean"])
            for est in estimators
        ]
    ).T,
    fmt="o",
    capsize=8,
    elinewidth=2,
    markersize=8,
    color="blue",
    ecolor="black",
    label="Mean ± 95% CI",
)
plt.xticks(ticks=range(len(estimators)), labels=[est["label"] for est in␣
 ↪estimators])

for est in estimators:
    print(
        f"{est['label']}:\n\tMean:\t\t{est['mean']:.4f}\n\tCI width:
 ↪\t{est['cis'][1] - est['cis'][0]:.4f}\n"
    )

plt.axhline(
    1 / (mu - lambda_),
    color="green",
    linestyle=":",
    linewidth=2,
    label="True Mean Traversal Time",
)
```

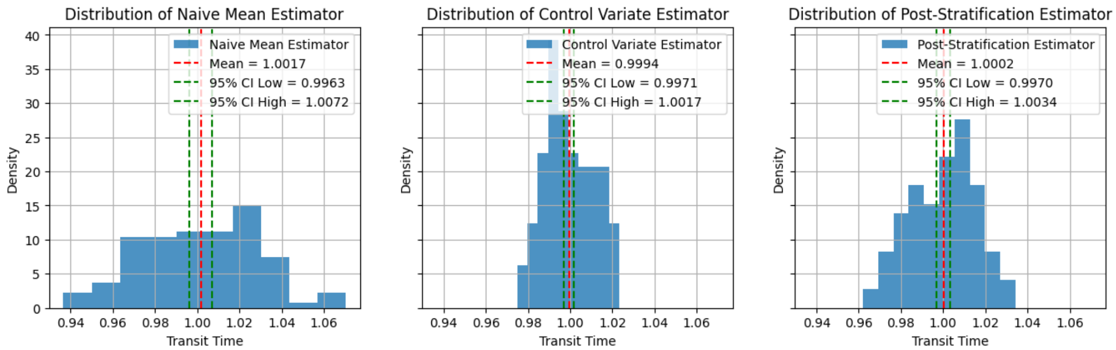
```python
plt.xlabel("Transit Time")
plt.ylabel("Mean Traversal Time")
plt.title("Comparison of Transit Time Estimators")
plt.grid()
plt.legend()
plt.show()
# Print the results
```

```
Naive Mean Estimator:
        Mean:           1.0017
        CI width:       0.0110

Control Variate Estimator:
        Mean:           0.9994
        CI width:       0.0046

Post-Stratification Estimator:
        Mean:           1.0002
        CI width:       0.0063
```



[101]:

```python
# Plot the distribution of each estimator to verify the normality

fig, axes = plt.subplots(
    1, len(estimators), figsize=(5 * len(estimators), 4), sharey=True,␣
 ↪sharex=True
)
for i, est in enumerate(estimators):
    axes[i].hist(
        est["values"],
        density=True,
        alpha=0.8,
        label=est["label"],
    )
    axes[i].axvline(
        est["mean"],
        color="red",
        linestyle="--",
        label=f"Mean = {est['mean']:.4f}",
    )
    axes[i].axvline(
        est["cis"][0],
        color="green",
        linestyle="--",
        label=f"95% CI Low = {est['cis'][0]:.4f}",
    )
    axes[i].axvline(
        est["cis"][1],
        color="green",
        linestyle="--",
        label=f"95% CI High = {est['cis'][1]:.4f}",
    )
    axes[i].set_title(f"Distribution of {est['label']}")
    axes[i].set_ylabel("Density")
    axes[i].set_xlabel("Transit Time")
    axes[i].legend()
    axes[i].grid()

plt.show()
```


Distribution of Naive Mean Estimator
Naive Mean Estimator
Mean = 1.0017
95% CI Low = 0.9963
95% CI High = 1.0072
Density
Transit Time
Distribution of Control Variate Estimator
Control Variate Estimator
Mean = 0.9994
95% CI Low = 0.9971
95% CI High = 1.0017
Density
Transit Time
Distribution of Post-Stratification Estimator
Post-Stratification Estimator
Mean = 1.0002
95% CI Low = 0.9970
95% CI High = 1.0034
Density
Transit Time