In [12]:

```
#ORIE 5129 HW3
#Abe and Calvin
```

In [13]:

```
import math
import random
import sys
from scipy import stats
```

In [21]:

```
#state of sys: #cars idling in each region (DT, MT, UT)

#initialization

def initData():
    locations = [2, 8, 2] # DT, MT, UT
    numCars = sum(locations)
    travelTimes = [[17, 10, 24], # travelTimes(i,j) is travel time from i to j
                   [10, 12, 17],
                   [24, 17, 10]]
    tripProbabilities = [[1/9, 1/9, 1/9],  #tripProbabilities(i, j) is
                         [1/9, 1/9, 1/9],  #the prob of a customer arriving at region i
                         [1/9, 1/9, 1/9]]  #and requesting a ride to region j
    meanInterarrival = 2
    time = 0
    endTime = 8*60 #minutes in a workday
    return (locations, time, meanInterarrival, travelTimes, tripProbabilities, endTime, numCars)

def initMetrics():
    idleTime = 0
    customersLost = 0
    pickups = 0
    dropoffs = 0
    return (idleTime, pickups, dropoffs, customersLost)

#future event list functionality

def addEvent(eventType, eventTime):
    global fel
    event = (eventType, eventTime)
    fel.append(event)

def deleteEvent(index):
    global fel
    fel.pop(index)

def findNextEvent():
    global fel
    earliestTime = 1e30
    for i in range(len(fel)):
        event = fel[i]
        eventTime = event[1]
        if eventTime < earliestTime:
            earliestTime = eventTime
            earliestIndex = i
    return earliestIndex

#random variable generation:

def generateInterarrival():
    global meanInterarrival
    sample = random.expovariate( 1 / meanInterarrival )
    return sample

def generateTrip(): #randomly generates a trip: (pickup location, dropoff location)
    global tripProbabilities
    n = len(tripProbabilities)
    cumulative = 0
    s = random.random()
    for i in range(n):
        for j in range(n):
            cumulative += tripProbabilities[i][j]
            if s < cumulative:
                return (i, j)

#event handling

def handleArrival(eventTime):
    global fel, locations, travelTimes, time, idleTime, customersLost, pickups
    #metrics
    idleTime += sum(locations)*(eventTime - time)

    #next arrival:
    ia = generateInterarrival()
    addEvent("a", eventTime + ia)


    trip = generateTrip()
    pickup = trip[0]
    dropoff = trip[1]
    if locations[pickup] == 0:
        customersLost += 1

    else:
        pickups += 1
        locations[pickup] -= 1
        travelTime = travelTimes[pickup][dropoff]
        addEvent("d" + str(dropoff), eventTime + travelTime)

    time = eventTime

def handleDropoff(eventTime, dropLocation):
    global locations, time, idleTime, dropoffs
    #metrics
    dropoffs += 1
    idleTime += sum(locations)*(eventTime - time)
    locations[dropLocation] += 1
    time = eventTime

def handleEnd(eventTime):
    global time, idleTime
    idleTime += sum(locations)*(eventTime - time)
    time = eventTime

#metrics stuff

def initMetricsDict(keys): #creates an empty list for each key
    metrics = dict()
    for key in keys:
        metrics[key] = list()
    return metrics

def handleMetrics(): #calculate the metrics and append the results to the list for that metric
    global metrics, idleTime, pickups, dropoffs, customersLost, numCars
    metrics["Pickups"].append(pickups)
    metrics["Dropoffs"].append(dropoffs)
    metrics["Customers Lost"].append(customersLost)
    metrics["Idle Time (hrs)"].append(idleTime/60) #total time spent idle
    metrics["Avg Idle Time"].append(idleTime/numCars/60) #hours spent idle per car
    metrics["Fraction Lost"].append( customersLost/(customersLost + pickups) )

# Estimate the expected value of each metric with a 95% CI
def printConfidenceIntervals(keys):
    global metrics, reps
    Z = stats.norm.ppf(.975) #0.975 for 95% confidence, 0.995 for 99% confidence
    metrics_avg = dict()
    metrics_stdev = dict()
    metrics_CI = dict()
    for key in keys:

        #sample mean of each metric
        metrics_avg[key] = sum(metrics[key])/reps

        #sample stdev of each metric
        squares = 0
        for i in range(reps):
            squares += (metrics_avg[key] - metrics[key][i])**2
        metrics_stdev[key] =  math.sqrt( squares / (reps-1) ) #ensure replications > 1

        #95% CI on the expected value of each metric: CI = X +- Z*stdev/sqrt(reps)
        val = Z * metrics_stdev[key] / math.sqrt(reps)
        metrics_CI[key] = ( metrics_avg[key] - val , metrics_avg[key] + val)

        #print the CI for each metric
        print(key + ":", metrics_CI[key])
```

In [27]:

```
reps = 1000 #10,000 takes ~12s

#Create a dictionary to keep track of the values of each metric for each replication
keys = ["Pickups",  "Dropoffs", "Customers Lost", "Fraction Lost", "Idle Time (hrs)", "Avg Idle Time"]
metrics = initMetricsDict(keys)

for r in range(reps):

    (locations, time, meanInterarrival, travelTimes, tripProbabilities, endTime, numCars) = initData()
    (idleTime, pickups, dropoffs, customersLost) = initMetrics()

    fel = list()
    ia = generateInterarrival()
    addEvent("a", ia)
    addEvent("e", endTime)

    eventType = "s"

    while eventType != "e":
        earliestIndex = findNextEvent()
        earliestEvent = fel[earliestIndex]
        earliestType = earliestEvent[0]
        earliestTime = earliestEvent[1]

        if earliestType == "a":
            handleArrival(earliestTime)

        elif earliestType[0] == "d":
            dropLocation = int(earliestType[1:])
            handleDropoff(earliestTime, dropLocation)

        elif earliestType == "e":
            handleEnd(earliestTime)

        else:
            print("Invalid event type:", earliestType)
            sys.exit(1)

        deleteEvent(earliestIndex)
        eventType = earliestType

    handleMetrics()
```

```
printKeys = ["Idle Time (hrs)", "Fraction Lost"]
printConfidenceIntervals(printKeys)
```

```
Idle Time (hrs): (51.08420827741541, 51.51931691767039)
Fraction Lost: (0.26651826835790643, 0.27173305928994024)
```