CompPerf (/github/AaronTHolt/CompPerf/tree/master) / HW8 (/github/AaronTHolt/CompPerf/tree/master/HW8)

In [2]:
```python
#problem 1
#4. Suppose an interactive system is supporting 100 users with 15 second think times and a
#system throughput of 5 interactions/second.
#a. What is the response time of the system?
think = 15  #seconds (Z)
users = 100
rate = 5   #interaction/second

response = users/rate - think
print("Response time for a = ", response, "(s)")

#b. Suppose that the service demands of the workload evolve over time so that system throughput drops to 50%
#of its former value (i.e., to 2.5 interactions/second). Assuming that there still are 100 users with 15 second
#think times, what would their response time be?

rate = 2.5
response = users/rate - think
print("Response time for b = ", response, "(s)")

#c. How do you account for the fact that response time in (b) is more than twice as large as that in (a)?

print('''There are a lot of users and they form a queue which slows things down.
        As the utilization gets\n near 100 the response time will increase significantly''')
```

```
Response time for a =  5.0 (s)
Response time for b =  25.0 (s)
There are a lot of users and they form a queue which slows things down.
        As the utilization gets
 near 100 the response time will increase significantly
```

In [3]:
```python
#Problem 2
#Part 1 - Visit Count
print("Proxy visits per packet = ", 0.7+2*0.3)
v_proxy = 0.7+2*0.3
print("Router visits per packet = ", 2*0.3)
v_router = 2*0.3
print("A visits per packet = ", 0.3*0.5)
v_A = 0.3*0.5
print("B visits per packet = ", 0.3*0.5)
v_B = 0.3*0.5
```

```
Proxy visits per packet =  1.2999999999999998
Router visits per packet =  0.6
A visits per packet =  0.15
B visits per packet =  0.15
```

In [4]:
```python
#Part 2 - Demand
#times in (s)
ser_proxy = 0.010
ser_router = 0.005
ser_A = 0.150
ser_B = 0.100
dem_proxy = ser_proxy*v_proxy
dem_router = ser_router*v_router
dem_A = ser_A*v_A
dem_B = ser_B*v_B

#Add in transmission time to total demand
dem_total = dem_proxy+dem_router+dem_A+dem_B+0.036*2*0.3

#Unit for demand = (visits*seconds)/packet
print("Proxy demand = ", dem_proxy, "(visits*seconds)/packet")
print("Router demand = ", dem_router)
print("A demand = ", dem_A)
print("B demand = ", dem_B)
print("Total demand = ", dem_total)
dem_max = max(dem_proxy,dem_router,dem_A,dem_B)
print("Max demand = ", dem_max)
```

```
Proxy demand =  0.012999999999999998 (visits*seconds)/packet
Router demand =  0.003
A demand =  0.0225
B demand =  0.015
Total demand =  0.07509999999999999
Max demand =  0.0225
```

In [5]:

```python
#Part 3
#Examine case study 5.3.1 for the process of calculating the job bounds,
#and calculate the upper bounds on throughput (X) and the lower bounds on response time (R).
#Write the bounds down and also graph those bounds (two lines for each).
import numpy as np
import scipy as sp
import scipy.stats
import matplotlib.pyplot as plt
import math
%matplotlib inline

#Calculate N*
think_time = 0.250 #(s)
N_star = (dem_total + think_time)/dem_max
print("N* = ", N_star)

#Optimistic bounds throughput
opt_1 = [0,0]
opt_2 = [1,1/(dem_max+think_time)]
line0_x = [0,1]
line0_y = [0,1/(dem_max+think_time)]

opt_3 = [0,1/dem_max]
opt_4 = [1,1/dem_max]
line1_x = [0,1]
line1_y = [1/dem_max, 1/dem_max]

#Response bounds
res_1 = [0, dem_total]
res_2 = [1, dem_total]
line2_x = [0, 1]
line2_y = [dem_total,dem_total]

res_3 = [0, -think_time]
res_4 = [1, dem_max-think_time]
line3_x = [0,1]
line3_y = [-think_time,dem_max-think_time]

line0, = plt.plot(line0_x, line0_y, 'ro-')
line1, = plt.plot(line1_x, line1_y, 'bo-')

#title and axis labels
plt.xlabel('Bounds')
plt.ylabel('Seconds')
plt.title("Throughput Upper Bounds")
#legend
plt.legend([line0, line1],
  ['Throughput Upper 1', 'Throughput Upper 2'], loc=1)
plt.show()

line2, = plt.plot(line2_x, line2_y, 'go-')
line3, = plt.plot(line3_x, line3_y, 'ko-')

#title and axis labels
plt.xlabel('Bounds')
plt.ylabel('Seconds')
plt.title("Response Time Lower Bounds")
#legend\n",
plt.legend([line2, line3],
  ['Respone Lower 1', 'Response Lower 2'], loc=1)
plt.show()
```
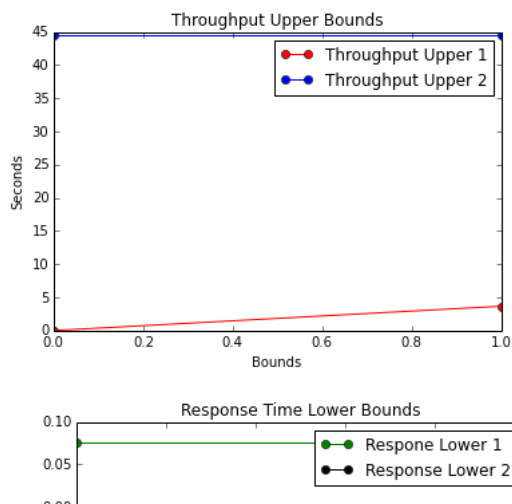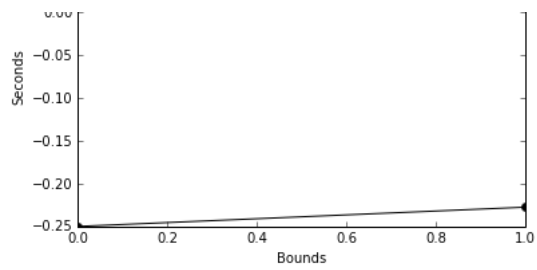
N* = 14.44888888888889

In [6]:
```python
#Part 4
#Modify the model in a method similar to 5.3.3 of the QSP text.
#In your case, assume that the two origin servers are of equal performance.
#Write down the bounds equations and graph them again, overlaying them with the original bounds from part 3.

#Change response time to 100ms for both A and B
#Part 2 - Demand
#times in (s)
ser_proxy = 0.010
ser_router = 0.005
ser_A = 0.100
ser_B = 0.100
dem_proxy = ser_proxy*v_proxy
dem_router = ser_router*v_router
dem_A = ser_A*v_A
dem_B = ser_B*v_B

#Add in transmission time to total demand
dem_total = dem_proxy+dem_router+dem_A+dem_B+0.036*2*0.3

#Unit for demand = (visits*seconds)/packet
print("Proxy demand = ", dem_proxy, "(visits*seconds)/packet")
print("Router demand = ", dem_router)
print("A demand = ", dem_A)
print("B demand = ", dem_B)
print("Total demand = ", dem_total)
dem_max = max(dem_proxy,dem_router,dem_A,dem_B)
print("Max demand = ", dem_max)

#Calculate N*
think_time = 0.250 #(s)
N_star = (dem_total + think_time)/dem_max
print("N* = ", N_star)

#Optimistic bounds throughput
opt_1 = [0,0]
opt_2 = [1,1/(dem_max+think_time)]
line0b_x = [0,1]
line0b_y = [0,1/(dem_max+think_time)]

opt_3 = [0,1/dem_max]
opt_4 = [1,1/dem_max]
line1b_x = [0,1]
line1b_y = [1/dem_max, 1/dem_max]

#Response bounds
res_1 = [0, dem_total]
res_2 = [1, dem_total]
line2b_x = [0, 1]
line2b_y = [dem_total,dem_total]

res_3 = [0, -think_time]
res_4 = [1, dem_max-think_time]
line3b_x = [0,1]
line3b_y = [-think_time,dem_max-think_time]

line0, = plt.plot(line0_x, line0_y, 'ro-')
line1, = plt.plot(line1_x, line1_y, 'bo-')
line2, = plt.plot(line0b_x, line0b_y, 'go-')
line3, = plt.plot(line1b_x, line1b_y, 'ko-')

#title and axis labels
plt.xlabel('Bounds')
plt.ylabel('Seconds')
plt.title("Throughput Upper Bounds")
#legend
plt.legend([line0, line1, line2, line3],
  ['Throughput Upper 1', 'Throughput Upper 2', 'Throughput Upper 1b', 'Throughput Upper 2b'], loc=1)
plt.show()

line0, = plt.plot(line2_x, line2_y, 'ro-')
line1, = plt.plot(line3_x, line3_y, 'bo-')
line2, = plt.plot(line2b_x, line2b_y, 'go-')
line3, = plt.plot(line3b_x, line3b_y, 'ko-')

#title and axis labels
plt.xlabel('Bounds')
plt.ylabel('Seconds')
plt.title("Response Time Lower Bounds")
#legend
plt.legend([line0, line1, line2, line3],
  ['Respone Lower 1', 'Response Lower 2', 'Respone Lower 1b', 'Response Lower 2b'], loc=1)
plt.show()
```
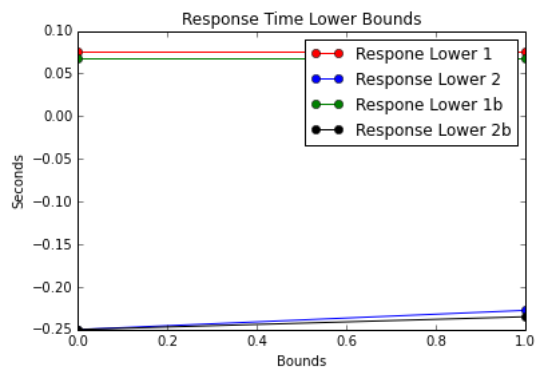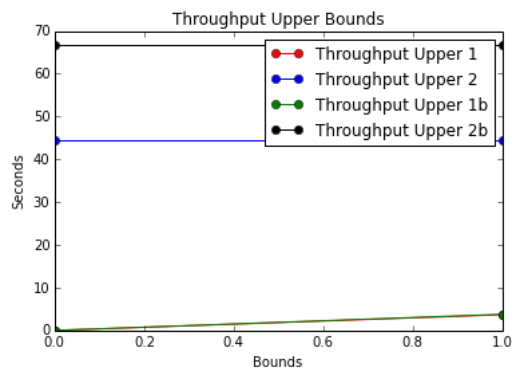
```
Proxy demand =   0.012999999999999998 (visits*seconds)/packet
Router demand =   0.003
A demand =   0.015
B demand =   0.015
Total demand =   0.0676
Max demand =   0.015
N* =   21.173333333333332
```

In [7]:

```python
#Part 5
#Now, determine the lower bounds on throughput (X) and the upper bounds on response time (R)
#for the original system using the balanced job bounds equations summarized in Table 5.2.
#Produce a single graph with the upper and lower bounds of X and R.

dem_ave = np.mean([dem_proxy,dem_router,dem_A,dem_B])

N_batch = (dem_total-dem_ave)/(dem_max-dem_ave)
N_terminal = ((dem_total+think_time)**2-dem_total*dem_ave)/((dem_total+think_time)*dem_max-dem_total*dem_ave)

print("Dem_ave, N+batch, N+terminal")
print(dem_ave, N_batch, N_terminal)

#Throughput
def terminal_t_min(N):
    output = N/(dem_total+think_time+((N-1)*dem_max)/(1+think_time/(N*dem_total)))
    return output

def terminal_t_max1(N):
    return 1/dem_max

def terminal_t_max2(N):
    output = N/(dem_total+think_time+((N-1)*dem_ave)/(1+think_time/(dem_total)))
    return output

N = []
min1 = []
max1 = []
max2 = []
for ii in range(1,int(N_terminal+1)):
    N.append(ii)
    min1.append(terminal_t_min(ii))
    max1.append(terminal_t_max1(ii))
    max2.append(terminal_t_max2(ii))

line0, = plt.plot(N, min1, 'ro-')
line1, = plt.plot(N, max1, 'bo-')
line2, = plt.plot(N, max2, 'go-')

#title and axis labels
plt.xlabel('N')
plt.ylabel('Throughput')
plt.title("Throughput Bounds")
#legend
plt.legend([line0, line1, line2, line3],
  ['Throughput Lower 1', 'Throughput Max 1', 'Throughput Max 2'], loc=4)
plt.show()


#Response Time
def terminal_r_min1(N):
    return N*dem_max-think_time

def terminal_r_min2(N):
    return dem_total+((N-1)*dem_ave)/(1+think_time/dem_total)

def terminal_r_max(N):
    return dem_total+((N-1)*dem_max)/(1+think_time/(N*dem_total))

rmin1 = []
rmin2 = []
rmax = []
for ii in range(1,int(N_terminal+1)):
    rmin1.append(terminal_r_min1(ii))
    rmin2.append(terminal_r_min2(ii))
    rmax.append(terminal_r_max(ii))

line0, = plt.plot(N, rmin1, 'ro-')
line1, = plt.plot(N, rmin2, 'bo-')
line2, = plt.plot(N, rmax, 'go-')

#title and axis labels
plt.xlabel('N')
plt.ylabel('Response')
plt.title("Response Bounds")
#legend
plt.legend([line0, line1, line2, line3],
  ['Response Lower 1', 'Response Lower 2', 'Response Max'], loc=4)
plt.show()
```

```
Dem_ave, N+batch, N+terminal
0.0115 16.0285714286 25.107199117
```


Throughput Bounds
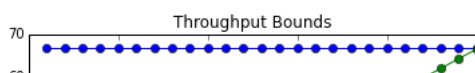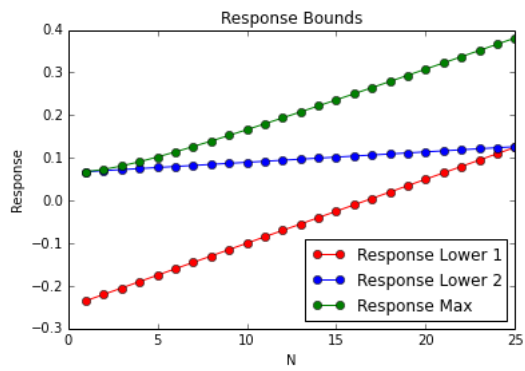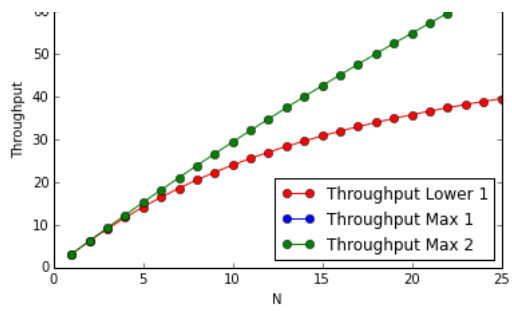
In [8]:

```
#Problem 3
#You will use the Vi and Di from the prior problem.

v_proxy = 0.7+2*0.3
v_router = 2*0.3
v_A = 0.3*0.5
v_B = 0.3*0.5

#times in (s)
ser_proxy = 0.010
ser_router = 0.005
ser_A = 0.150
ser_B = 0.100
dem_proxy = ser_proxy*v_proxy
dem_router = ser_router*v_router
dem_A = ser_A*v_A
dem_B = ser_B*v_B

#Add in transmission time to total demand
dem_total = dem_proxy+dem_router+dem_A+dem_B+0.036*2*0.3

#Unit for demand = (visits*seconds)/packet
dem_max = max(dem_proxy,dem_router,dem_A,dem_B)

#Write down the equations for the Response for the individual components for N customers in the system,
#using the appropriate service times (again, in the PDF document)
#and the Queue length for N-1 customers in the system

#residence times
#delay center = D_k
#queue center = D_k*(1+A_k(N))

#6.1 system throughput

Q_proxy = []
Q_router = []
Q_A = []
Q_B = []

r_proxy_l = []
r_router_l = []
r_A_l = []
r_B_l = []
r_PR_l = []

XX = []

K=5

#Utilization
def U(N):
    return (N+1)/(N+K)

#Service Center times
def r_proxy(N):
    if N==0:
        return dem_proxy
    return dem_proxy*(1+Q_proxy[N])

def r_router(N):
    if N==0:
        return dem_router
    return dem_router*(1+Q_router[N])

def r_A(N):
    if N==0:
        return dem_A
    return dem_A*(1+Q_A[N])

def r_B(N):
    if N==0:
        return dem_B
    return dem_B*(1+Q_B[N])

def r_PR_delay(N):
    return 0.036*2*0.3

def X(N):
    output = N/(think_time + (r_proxy_l[N]+r_router_l[N]+r_A_l[N]+r_B_l[N]+r_PR_l[N]))
    return output

#Queue times for service centers
def Q_proxy_find(N):
    if N==0:
        return U(N)
    else:
```

```python
        return X(N)*r_proxy(N)

def Q_router_find(N):
    if N==0:
        return U(N)
    else:
        return X(N)*r_router(N)

def Q_A_find(N):
    if N==0:
        return U(N)
    else:
        return X(N)*r_A(N)

def Q_B_find(N):
    if N==0:
        return U(N)
    else:
        return X(N)*r_B(N)

def Q_PR_find(N):
    if N==0:
        return U(N)
    else:
        return X(N)*r_PR_delay(N)


class prettyfloat(float):
    def __repr__(self):
        return "%0.6f" % self
```

Problem 1 Unique Service Times

N, Qproxy, Qrouter, QA,   QB,   proxy,   router,   A,   B,   PR_delay, Throughput

N	Qproxy	Qrouter	QA	QB	proxy	router	A	B	PR_delay	Throughput
0	0.000000	0.000000	0.000000	0.000000	0.013000	0.003000	0.022500	0.015000	0.021600	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.015600	0.003600	0.027000	0.018000	0.021600	0.071865
2	0.046456	0.010721	0.080405	0.053603	0.013604	0.003032	0.024309	0.015804	0.021600	6.091075
3	0.082863	0.018469	0.148069	0.096264	0.014077	0.003055	0.025832	0.016444	0.021600	9.063222
4	0.127585	0.027602	0.234117	0.149035	0.014659	0.003083	0.027768	0.017236	0.021600	11.963696
5	0.175371	0.036885	0.332204	0.206201	0.015280	0.003111	0.029975	0.018093	0.021600	14.790358
6	0.229594	0.046008	0.443335	0.267602	0.015938	0.003138	0.032475	0.019014	0.021600	17.535399
7	0.289478	0.055076	0.569463	0.333419	0.016633	0.003165	0.035313	0.020001	0.021600	20.189640
8	0.335819	0.063902	0.712955	0.403819	0.017366	0.003192	0.038541	0.021057	0.021600	22.743031
9	0.394947	0.072189	0.876550	0.478906	0.018134	0.003218	0.042222	0.022184	0.021600	25.184825
10	0.456710	0.081039	1.063363	0.558690	0.018937	0.003243	0.046426	0.023380	0.021600	27.503782
11	0.520845	0.089198	1.276882	0.643048	0.019771	0.003268	0.051230	0.024646	0.021600	29.688476
12	0.586711	0.097010	1.520036	0.731694	0.020631	0.003291	0.056721	0.025975	0.021600	31.727725
13	0.654567	0.104417	1.799063	0.824141	0.021509	0.003313	0.062992	0.027362	0.021600	33.611156
14	0.722953	0.111362	2.117223	0.919672	0.022398	0.003334	0.070138	0.028795	0.021600	35.329886
15	0.791332	0.117891	2.477951	1.017327	0.023287	0.003353	0.078254	0.030260	0.021600	36.877282
16	0.858773	0.123608	2.885791	1.115903	0.024164	0.003371	0.087430	0.031739	0.021600	38.249705
17	0.924268	0.128939	3.344183	1.213990	0.025015	0.003387	0.097744	0.033210	0.021600	39.447158
18	0.986700	0.133600	3.855728	1.310034	0.025828	0.003401	0.109254	0.034651	0.021600	40.473682
19	1.045365	0.137643	4.421906	1.402434	0.026590	0.003413	0.121993	0.036037	0.021600	41.337411
20	1.099151	0.141082	5.042870	1.489656	0.027289	0.003423	0.135965	0.037345	0.021600	42.050232
21	1.147507	0.143948	5.717342	1.570359	0.027918	0.003432	0.151140	0.038555	0.021600	42.627041
22	1.190045	0.146289	6.442660	1.643502	0.028471	0.003439	0.167460	0.039653	0.021600	43.084724
23	1.226642	0.148163	7.214961	1.708418	0.028946	0.003444	0.184837	0.040626	0.021600	43.440996
24	1.257461	0.149632	8.029487	1.764846	0.029347	0.003449	0.203163	0.041473	0.021600	43.713297
25	1.282814	0.150761	8.880964	1.812900	0.029677	0.003452	0.222321	0.042194	0.021600	43.917879
26	1.303355	0.151617	9.763828	1.853054	0.029944	0.003455	0.242189	0.042796	0.021600	44.069176
27	1.319591	0.152252	10.672992	1.885976	0.030155	0.003457	0.262642	0.043290	0.021600	44.179483
28	1.332218	0.152718	11.603402	1.912514	0.030319	0.003458	0.283577	0.043688	0.021600	44.258891
29	1.341878	0.153054	12.550784	1.933570	0.030444	0.003459	0.304893	0.044004	0.021600	44.315420
30	1.349157	0.153294	13.511445	1.950036	0.030539	0.003460	0.326508	0.044251	0.021600	44.355276

In [9]:

```python
#BALANCED SERVICE TIMES

#You will use the Vi and Di from the prior problem.

v_proxy = 0.7+2*0.3
v_router = 2*0.3
v_A = 0.3*0.5
v_B = 0.3*0.5

#times in (s)
ser_proxy = 0.010
ser_router = 0.005
ser_A = 0.100
ser_B = 0.100
dem_proxy = ser_proxy*v_proxy
dem_router = ser_router*v_router
dem_A = ser_A*v_A
dem_B = ser_B*v_B

#Add in transmission time to total demand
dem_total = dem_proxy+dem_router+dem_A+dem_B+0.036*2*0.3

#Unit for demand = (visits*seconds)/packet
dem_max = max(dem_proxy,dem_router,dem_A,dem_B)

#Write down the equations for the Response for the individual components for N customers in the system,
#using the appropriate service times (again, in the PDF document)
#and the Queue length for N-1 customers in the system

#residence times
#delay center = D_k
#queue center = D_k*(1+A_k(N))

#6.1 system throughput

Q_proxy = []
Q_router = []
Q_A = []
Q_B = []

r_proxy_l = []
r_router_l = []
r_A_l = []
r_B_l = []
r_PR_l = []

XX = []

K=5

for N in range(1):
    Q_proxy.append(0)
    Q_router.append(0)
    Q_A.append(0)
    Q_B.append(0)

#Utilization
def U(N):
    return (N+1)/(N+K)

#Service Center times
def r_proxy(N):
    return dem_proxy*(1+Q_proxy[N-1])

def r_router(N):
    return dem_router*(1+Q_router[N-1])

def r_A(N):
    return dem_A*(1+Q_A[N-1])

def r_B(N):
    return dem_B*(1+Q_B[N-1])

def r_PR_delay(N):
    return 0.036*2*0.3

def X(N):
    output = N/(think_time + (r_proxy_l[N]+r_router_l[N]+r_A_l[N]+r_B_l[N]+r_PR_l[N]))
    return output

#Queue times for service centers
def Q_proxy_find(N):
    if N==0:
        return U(N)
    else:
        return X(N)*r_proxy(N)
```

```python
def Q_router_find(N):
    if N==0:
        return U(N)
    else:
        return X(N)*r_router(N)

def Q_A_find(N):
    if N==0:
        return U(N)
    else:
        return X(N)*r_A(N)

def Q_B_find(N):
    if N==0:
        return U(N)
    else:
        return X(N)*r_B(N)

def Q_PR_find(N):
    if N==0:
        return U(N)
    else:
        return X(N)*r_PR(N)
```

```
Problem 3 - Balanced Service Times (100ms each)
N, Qproxy, Qrouter, PR_delay(N)
```

```
class Closeform():

    def rep(N):
        return X(N)

    print('''Problem 3 Balanced service times, (100ms each)''')
    print('''N, Qproxy, Qrouter, QA, QB, proxy, router, A, B, PR_delay Throughput''')
    for N in range(0,31):
        Qproxy.append(Q_proxy(N))
        Qrouter.append(Q_router(N))
        QA.append(Q_A(N))
        QB.append(Q_B(N))
        PR_delay.append(PR_delay(N))
        proxy.append(r_proxy(N))
        router.append(r_router(N))
        A.append(r_A(N))
        B.append(r_B(N))
        PR_Q.append(PR_delay(N))
        XX.append(X(N))
        print(N, [Qproxy, Qrouter, QA, QB, proxy, router, A, B, PR_Q, X[N]])
```

```
N, Qproxy   Qrouter   QA        QB        proxy     router    A         B         PR_delay, Throughput
0 [0.000000, 0.000000, 0.000000, 0.000000, 0.013000, 0.003000, 0.015000, 0.015000, 0.021600, 0.000000]
1 [0.200000, 0.200000, 0.200000, 0.200000, 0.013000, 0.003000, 0.015000, 0.015000, 0.021600, 3.148615]
2 [0.088329, 0.009446, 0.047229, 0.047229, 0.015600, 0.003600, 0.018000, 0.018000, 0.021600, 6.119951]
3 [0.095471, 0.022032, 0.110159, 0.110159, 0.013532, 0.003028, 0.015708, 0.015708, 0.021600, 9.387399]
4 [0.127046, 0.030488, 0.147461, 0.147461, 0.014241, 0.003066, 0.016652, 0.016652, 0.021600, 12.414187]
5 [0.176792, 0.038063, 0.206726, 0.206726, 0.014651, 0.003085, 0.017212, 0.017212, 0.021600, 15.443513]
6 [0.220269, 0.047683, 0.265813, 0.265813, 0.015288, 0.003114, 0.018101, 0.018101, 0.021600, 18.392819]
7 [0.281379, 0.057279, 0.332926, 0.332926, 0.015941, 0.003143, 0.018987, 0.018987, 0.021600, 21.298683]
8 [0.339533, 0.066941, 0.404402, 0.404402, 0.016658, 0.003172, 0.019994, 0.019994, 0.021600, 24.138734]
9 [0.402101, 0.076564, 0.482627, 0.482627, 0.017414, 0.003201, 0.021066, 0.021066, 0.021600, 26.918157]
10 [0.468751, 0.086160, 0.567059, 0.567059, 0.018227, 0.003230, 0.022239, 0.022239, 0.021600, 29.626485]
11 [0.540911, 0.095684, 0.658876, 0.658876, 0.019094, 0.003258, 0.023506, 0.023506, 0.021600, 32.261470]
12 [0.616995, 0.105123, 0.758334, 0.758334, 0.020020, 0.003287, 0.024883, 0.024883, 0.021600, 34.815561]
13 [0.697610, 0.114440, 0.866320, 0.866320, 0.021008, 0.003315, 0.026375, 0.026375, 0.021600, 37.284185]
14 [0.783263, 0.123611, 0.983371, 0.983371, 0.022061, 0.003343, 0.027995, 0.027995, 0.021600, 39.660720]
15 [0.874962, 0.132594, 1.110294, 1.110294, 0.023182, 0.003371, 0.029751, 0.029751, 0.021600, 41.939932]
16 [0.972269, 0.141372, 1.247737, 1.247737, 0.024375, 0.003398, 0.031654, 0.031654, 0.021600, 44.115889]
17 [1.073300, 0.149897, 1.396462, 1.396462, 0.025639, 0.003424, 0.033716, 0.033716, 0.021600, 46.183641]
18 [1.184125, 0.158138, 1.557130, 1.557130, 0.026979, 0.003450, 0.035947, 0.035947, 0.021600, 48.138317]
19 [1.298721, 0.066062, 1.730425, 1.730425, 0.028394, 0.003474, 0.038357, 0.038357, 0.021600, 49.976072]
20 [1.419002, 0.173608, 1.916930, 1.916930, 0.029883, 0.003498, 0.040956, 0.040956, 0.021600, 51.693705]
21 [1.544788, 0.180834, 2.117187, 2.117187, 0.031447, 0.003521, 0.043754, 0.043754, 0.021600, 53.289237]
22 [1.675788, 0.187627, 2.331614, 2.331614, 0.033082, 0.003543, 0.046758, 0.046758, 0.021600, 54.761749]
23 [1.811637, 0.193994, 2.560539, 2.560539, 0.034785, 0.003563, 0.049974, 0.049974, 0.021600, 56.111719]
24 [1.951809, 0.199910, 2.804439, 2.804439, 0.036550, 0.003582, 0.053408, 0.053408, 0.021600, 57.340897]
25 [2.095883, 0.205394, 3.062468, 3.062468, 0.038378, 0.003600, 0.057062, 0.057062, 0.021600, 58.452445]
26 [2.243065, 0.210415, 3.335418, 3.335418, 0.040246, 0.003616, 0.060937, 0.060937, 0.021600, 59.450763]
27 [2.392684, 0.214985, 3.622752, 3.622752, 0.042160, 0.003631, 0.065031, 0.065031, 0.021600, 60.341447]
28 [2.543986, 0.219115, 3.924081, 3.924081, 0.044105, 0.003645, 0.069341, 0.069341, 0.021600, 61.131046]
29 [2.696178, 0.222820, 4.238905, 4.238905, 0.046072, 0.003657, 0.073861, 0.073861, 0.021600, 61.826887]
30 [2.848477, 0.226122, 4.566609, 4.566609, 0.048050, 0.003668, 0.078584, 0.078584, 0.021600, 62.436793]
```

In [10]:
```python
#Compare the results from the simulation model, MVA model and the balanced job bounds.
#Do the bounds correctly bound the MVA model? How different are the MVA results than the simulation?

#Simulation results: MeanResp = 0.211 Qrouter = 0.132 Qprox = .57 QA = 0.985 QB = 0.66
balanced_resp_time = 0.048050+0.003668+0.078584+0.021600
print("balanced response time = ", balanced_resp_time, "s")
mva_resp_time = ((0.030539 +0.003460 +0.326508+ 0.021600)+(0.030539+0.003460+ 0.044251+0.021600))/2
print("mva response time = ", mva_resp_time, "s")
sim_resp_time = 0.42
print("simulation response time", sim_resp_time, "s")
print("The bounds correctly bounded the MVA model. The response time and throughput was within the bounded range.")
print('''The MVA analysis was not very close to the actual simluation. It ended up having a response time
        which was about 40% faster. ''')
```

```
balanced response time =  0.151902 s
mva response time =  0.2409785 s
simulation response time 0.42 s
The bounds correctly bounded the MVA model. The response time and throughput was within the bounded range.
The MVA analysis was not very close to the actual simluation. It ended up having a response time
        which was about 40% faster.
```

In [14]:

```
#Problem 4

#Original SERVICE TIMES


v_proxy = 0.7+2*0.3
v_router = 2*0.3
v_A = 0.3*0.5
v_B = 0.3*0.5

#times in (s)
ser_proxy = 0.010
ser_router = 0.005
ser_A = 0.150
ser_B = 0.100
dem_proxy = ser_proxy*v_proxy
dem_router = ser_router*v_router
dem_A = ser_A*v_A
dem_B = ser_B*v_B

#Add in transmission time to total demand
dem_total = dem_proxy+dem_router+dem_A+dem_B+0.036*2*0.3

#Unit for demand = (visits*seconds)/packet
dem_max = max(dem_proxy,dem_router,dem_A,dem_B)

#Write down the equations for the Response for the individual components for N customers in the system,
#using the appropriate service times (again, in the PDF document)
#and the Queue length for N-1 customers in the system

#residence times
#delay center = D_k
#queue center = D_k*(1+A_k(N))

#6.1 system throughput

Q_proxy = []
Q_router = []
Q_A = []
Q_B = []

r_proxy_l = []
r_router_l = []
r_A_l = []
r_B_l = []
r_PR_l = []

XX = []

K=4

# h function selected
# h = ((N-1)/N)*Q[N]

#Service Center times
def r_proxy(N):
    return dem_proxy*(1+((N-1)/N)*Q_proxy[N-1])

def r_router(N):
    return dem_router*(1+((N-1)/N)*Q_router[N-1])

def r_A(N):
    return dem_A*(1+((N-1)/N)*Q_A[N-1])

def r_B(N):
    return dem_B*(1+((N-1)/N)*Q_B[N-1])

def r_PR_delay(N):
    return 0.036*2*0.3

def X(jobs, N):
    n = N-1 #account for starting at N=1
    output = jobs/(think_time + (r_proxy_l[n]+r_router_l[n]+r_A_l[n]+r_B_l[n]+r_PR_l[n]))
    return output

#Queue times for service centers
def Q_proxy_find(jobs, N):
    return X(jobs, N)*r_proxy(N)

def Q_router_find(jobs, N):
    return X(jobs, N)*r_router(N)
def Q_A_find(jobs, N):
    return X(jobs, N)*r_A(N)

def Q_B_find(jobs, N):
    return X(jobs, N)*r_B(N)
```

```python
def Q_PR_find(jobs, N):
    return X(jobs, N)*r_PR_delay(N)


class prettyfloat(float):
    def __repr__(self):
        return "%0.6f" % self

print("Problem4 - Original Service Times")
print("h function - h=((N-1)/N)*Q")
```

```
Problem4 - Original Service Times'''QA,        QB,      proxy,    router,    A,        B,        PR_delay, Throughput'''
h function - h=((N-1)/N)*Q
Iter,Qproxy, Qrouter,  QA,        QB,        proxy,     router,    A,         B,         PR_delay, Throughput
1 [7.500000, 7.500000, 7.500000, 7.500000, 0.013000, 0.003000, 0.022500, 0.015000, 0.021600, 92.279299]
2 [1.996284, 0.276824, 2.076284, 1.384189, 0.020798, 0.003415, 0.045858, 0.025381, 0.021600, 81.732183]
3 [1.698833, 0.270630, 3.748091, 2.074479, 0.027732, 0.003558, 0.078721, 0.035745, 0.021600, 71.881027]
4 [1.993207, 0.255772, 5.658572, 2.569372, 0.032436, 0.003575, 0.117988, 0.043905, 0.021600, 63.897091]
5 [2.022542, 0.228463, 7.539116, 2.805430, 0.034554, 0.003548, 0.158204, 0.048665, 0.021600, 58.075159]
6 [2.006754, 0.206069, 9.187727, 2.826237, 0.034740, 0.003515, 0.194770, 0.050328, 0.021600, 54.058646]
7 [1.877989, 0.190025, 10.528996, 2.720661, 0.033926, 0.003489, 0.225559, 0.049980, 0.021600, 51.321183]
8 [1.741713, 0.179041, 11.575966, 2.565029, 0.032805, 0.003470, 0.250402, 0.048666, 0.021600, 49.428021]
9 [1.621504, 0.171514, 12.376867, 2.405464, 0.031737, 0.003457, 0.270037, 0.047073, 0.021600, 48.084248]
10 [1.526968, 0.166245, 12.984542, 2.263463, 0.030855, 0.003449, 0.285437, 0.045557, 0.021600, 47.103334]
11 [1.453373, 0.162453, 13.445013, 2.145875, 0.030176, 0.003443, 0.297512, 0.044262, 0.021600, 46.368335]
12 [1.396922, 0.159649, 13.795138, 2.052352, 0.029674, 0.003439, 0.307025, 0.043220, 0.021600, 45.804488]
13 [1.353520, 0.157512, 14.063110, 1.979663, 0.029310, 0.003436, 0.314580, 0.042411, 0.021600, 45.362626]
14 [1.329599, 0.155876, 14.270174, 1.923861, 0.029050, 0.003434, 0.320645, 0.041797, 0.021600, 45.009515]
15 [1.307534, 0.154573, 14.432063, 1.881246, 0.028865, 0.003433, 0.325573, 0.041337, 0.021600, 44.722165]
16 [1.290804, 0.153522, 14.560344, 1.848700, 0.028733, 0.003432, 0.329632, 0.040997, 0.021600, 44.484372]
17 [1.278159, 0.152661, 14.663483, 1.823741, 0.028639, 0.003431, 0.333021, 0.040747, 0.021600, 44.284531]
18 [1.268249, 0.151941, 14.747077, 1.804459, 0.028571, 0.003431, 0.335888, 0.040563, 0.021600, 44.114211]
19 [1.260400, 0.151304, 14.817400, 1.789411, 0.028523, 0.003430, 0.338345, 0.040428, 0.021600, 43.967206]
20 [1.254060, 0.151081, 14.876098, 1.777528, 0.028488, 0.003430, 0.340477, 0.040330, 0.021600, 43.838888]
21 [1.248870, 0.150350, 14.926115, 1.768012, 0.028462, 0.003430, 0.342345, 0.040257, 0.021600, 43.725757]
22 [1.244532, 0.149962, 14.969309, 1.760282, 0.028444, 0.003429, 0.344000, 0.040204, 0.021600, 43.625138]
23 [1.240852, 0.149620, 15.007044, 1.753907, 0.028430, 0.003429, 0.345478, 0.040165, 0.021600, 43.534952]
24 [1.237687, 0.149705, 15.040355, 1.748570, 0.028420, 0.003429, 0.346808, 0.040136, 0.021600, 43.453569]
25 [1.234929, 0.149172, 15.070070, 1.744039, 0.028412, 0.003429, 0.348010, 0.040114, 0.021600, 43.379689]
26 [1.232500, 0.148756, 15.096811, 1.740140, 0.028406, 0.003429, 0.349101, 0.040098, 0.021600, 43.312268]
27 [1.230339, 0.148522, 15.120784, 1.736743, 0.028402, 0.003429, 0.350117, 0.040086, 0.021600, 43.250453]
28 [1.228400, 0.148309, 15.142718, 1.733750, 0.028399, 0.003429, 0.351043, 0.040077, 0.021600, 43.193542]
29 [1.226648, 0.148112, 15.162786, 1.731087, 0.028397, 0.003429, 0.351898, 0.040071, 0.021600, 43.140952]
30 [1.225054, 0.147931, 15.181235, 1.728698, 0.028395, 0.003429, 0.352692, 0.040066, 0.021600, 43.092191]
```

```
In [ ]:
```