Problem 1

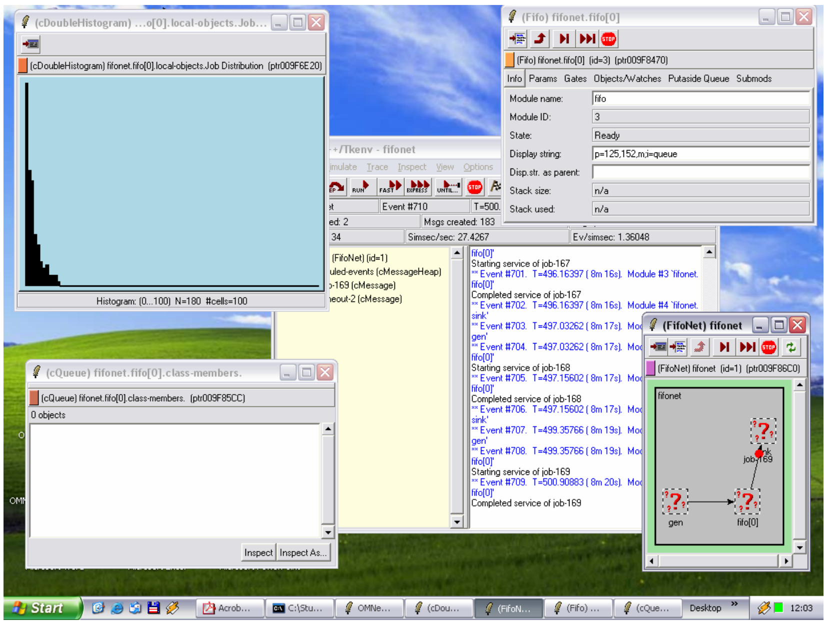
Install the OMNet++ discrete event simulation package in the latest version on your computer and run the test simulations. You can find the software at http://whale.hit.bme.hu/omnetpp/. Read the Manual.

Set up a M/M/2 simulation. Vary the load ρ as $\rho \in \{0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; 0.8\}$ (assume a service rate μ of one customer per second and vary λ accordingly). For each ρ create 10000 customers (the initial number of customers in the system is zero).

- *determine the mean system response time for all the customers (sample mean)*
- *look every 0.1 seconds at the system, observe the number of customers in the system at each sampling point and count how often exactly k customers are found ($k \geq 0$). Plot the relative frequencies in a histogram.*

Compare your simulation results with analytical results. Submit your code (C++), your simulation results and the correct analytical results.

Even though OMNet++ was originally designed for Unix/Linux systems, the new installer (latest version is 2.3b2, dated March 17th, 2003) set up the system flawlessly on our Windows 2000 and XP machines.


(Fifo) fifonet.fifo[0] (id=3) (ptr009F8470)

Module name:	fifo
Module ID:	3
State:	Ready
Display string:	p=125,152,m;i=queue
Disp.str. as parent:	
Stack size:	n/a
Stack used:	n/a

```
fifo[0]'
Starting service of job-167
** Event #701.  T=496.16397 ( 8m 16s).  Module #3 `fifonet.
fifo[0]'
Completed service of job-167
** Event #702.  T=496.16397 ( 8m 16s).  Module #4 `fifonet.
sink'
** Event #703.  T=497.03262 ( 8m 17s).  Mod
gen'
** Event #704.  T=497.03262 ( 8m 17s).  Mod
fifo[0]'
Starting service of job-168
** Event #705.  T=497.15602 ( 8m 17s).  Mod
fifo[0]'
Completed service of job-168
** Event #706.  T=497.15602 ( 8m 17s).  Mod
sink'
** Event #707.  T=499.35766 ( 8m 19s).  Mod
gen'
** Event #708.  T=499.35766 ( 8m 19s).  Mod
fifo[0]'
Starting service of job-169
** Event #709.  T=500.90883 ( 8m 20s).  Mod
fifo[0]'
Completed service of job-169
```

(cDoubleHistogram) fifonet.fifo[0].local-objects.Job Distribution (ptr009F6E20)
Histogram: (0...100) N=180 #cells=100

(cQueue) fifonet.fifo[0].class-members. (ptr009F85CC)
0 objects

Event #710 T=500. Msgs created: 183 Simsec/sec: 27.4267 Ev/simsec: 1.36048

(FifoNet) fifonet (id=1) (ptr009F86C0)

The relationship $\rho = \dfrac{\lambda}{m \cdot \mu}$ while $\mu = 1$ and $m = 2$ leads to $\lambda \in \{0.2; 0.4; 0.6; 0.8; 1.0; 1.2; 1.4; 1.6\}$, e.g., we have to evaluate eight different systems.

First, we analyze the problem in a mathematical manner to get an idealized result. The so-called “probability of queueing” – better known as Erlang’s C formula – turns out to be:

$$\sigma = P(\geq m\ jobs) = \frac{(m\rho)^m}{m! \cdot (1-\rho)} \cdot \pi_0$$

$$P(\geq 2\ jobs) = \frac{4\rho^2}{2 \cdot (1-\rho)} \cdot \pi_0$$

We need to know π_0 before performing any further calculations. According to literature:

$$\pi_0 = \left[1 + \frac{(m\rho)^m}{m! \cdot (1-\rho)} + \sum_{n=1}^{m-1} \frac{(m\rho)^n}{n!}\right]^{-1}$$
$$= \left[1 + \frac{4\rho^2}{2 \cdot (1-\rho)} + 2\rho^2\right]^{-1}$$

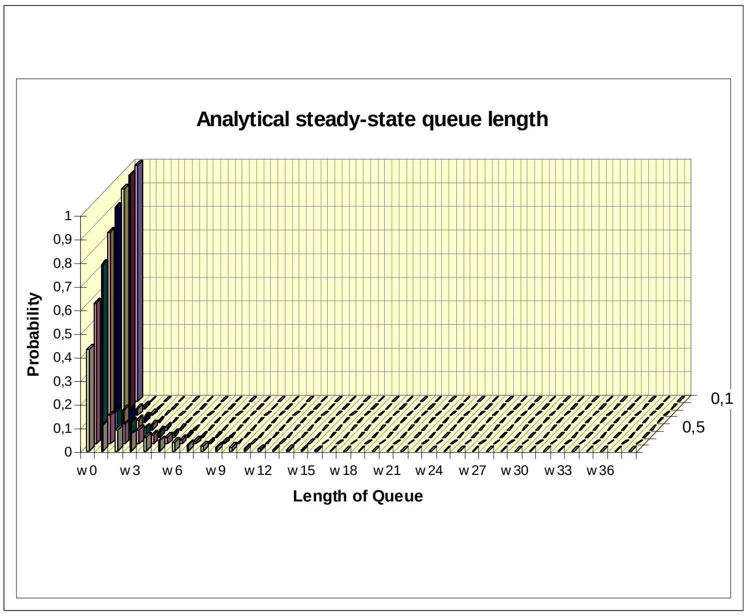
Using the given sequence of ρ we obtain:

$$\pi_0 = \left\{\frac{9}{11}, \frac{2}{3}, \frac{7}{13}, \frac{3}{7}, \frac{1}{3}, \frac{1}{4}, \frac{3}{17}, \frac{1}{9}\right\}$$
$$\approx \{0.819, 0.667, 0.538, 0.429, 0.333, 0.250, 0.176, 0.111\}$$
$$\sigma = \left\{\frac{1}{55}, \frac{1}{15}, \frac{9}{65}, \frac{8}{35}, \frac{1}{3}, \frac{9}{20}, \frac{49}{85}, \frac{32}{45}\right\}$$
$$\approx \{0.018, 0.067, 0.138, 0.229, 0.333, 0.450, 0.576, 0.711\}$$

Little’s Law gives:

$$E[r] = \frac{1}{\mu} \cdot \left(1 + \frac{\sigma}{m \cdot (1-\rho)}\right)$$
$$= 1 + \frac{\sigma}{2 \cdot (1-\rho)}$$
$$= \left\{\frac{100}{99}, \frac{25}{24}, \frac{100}{91}, \frac{25}{21}, \frac{4}{3}, \frac{25}{16}, \frac{100}{51}, \frac{25}{9}\right\}$$
$$\approx \{1.010, 1.042, 1.099, 1.190, 1.333, 1.563, 1.961, 2.778\}$$

The picture below shows the steady-state queue length as computed by Excel:

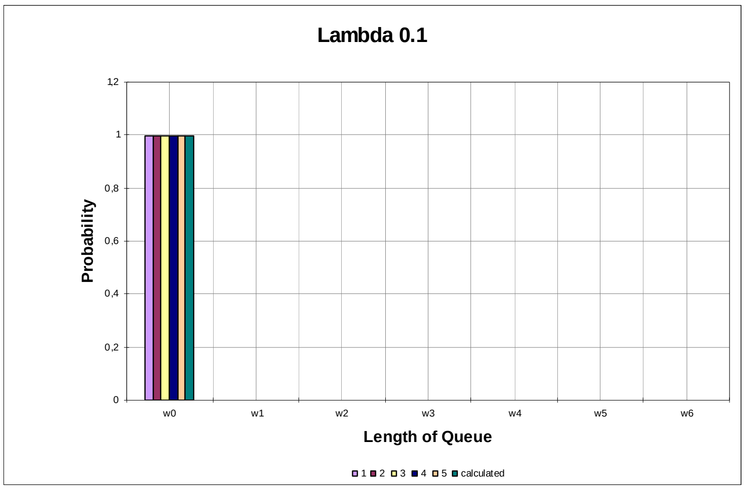


The OMNet++ suite comes along with a few comprehensive demos explaining a M/M/1 queue in detail (in the subdirectory samples/fifo1). We used the provided code and modified it to a M/M/2 queue. Most of the source code files have been generated so we avoid to print all of them. Nevertheless, the core functionality can be found in fifo1.cpp:

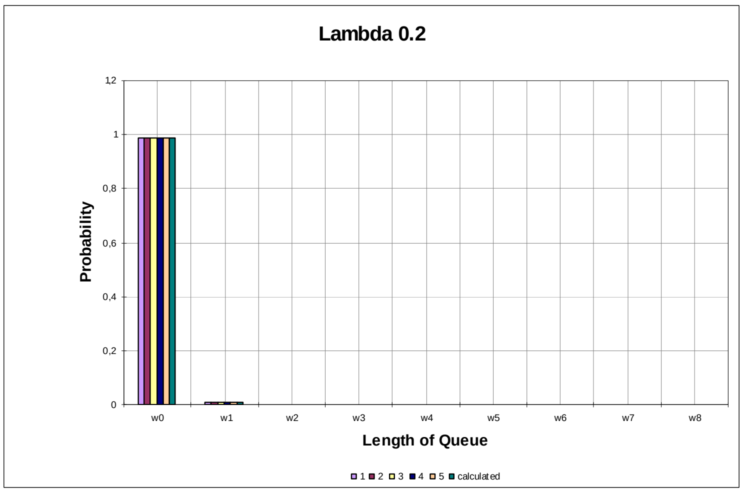
```
//-------------------------------------------------------------
// file: fifo1.cc
//        (part of Fifo1 - an OMNeT++ demo simulation)
//-------------------------------------------------------------

#include "fifo1.h"
#include <fstream>

using namespace std;


void FF2AbstractFifo::activity()
{
    msgServiced1 = NULL;
    msgServiced2 = NULL;
    endServiceMsg1 = new cMessage("end-service-server-1");
    endServiceMsg2 = new cMessage("end-service-server-2");
    recordHistogram = new cMessage("record-histogram");
    queue.setName("queue");
    hist.setRange(0.0, 50.0);
    hist.setNumCells(51);

    // send record request
    scheduleAt( simTime(), recordHistogram );

    for(;;)
    {
        cMessage *msg = receive();
        if (msg==endServiceMsg1)
        {
            // server 1 has finished a job
            endService( msgServiced1 );
            if (queue.empty())
            {
                msgServiced1 = NULL;
            }
            else
            {
                // new job available => catch it
                msgServiced1 = (cMessage *) queue.pop();
                simtime_t serviceTime = startService( msgServiced1 );
                scheduleAt( simTime()+serviceTime, endServiceMsg1 );
            }
        }
        else if (msg==endServiceMsg2)
        {
            // server 2 has finished a job
            endService( msgServiced2 );
            if (queue.empty())
            {
                msgServiced2 = NULL;
            }
            else
            {
                // new job available => catch it
                msgServiced2 = (cMessage *) queue.pop();
                simtime_t serviceTime = startService( msgServiced2 );
                scheduleAt( simTime()+serviceTime, endServiceMsg2 );
            }
        }
        else if (msg==recordHistogram)
        {
            // record current queue length
            hist.collect(queue.length());
            // resubmit message for next record 0.1s later
            scheduleAt( simTime()+0.1, recordHistogram );
        }
        else if (!msgServiced1)
        {
            // new job arrival and server 1 is idle => start there
            arrival( msg );
            msgServiced1 = msg;
            simtime_t serviceTime = startService( msgServiced1 );
            scheduleAt( simTime()+serviceTime, endServiceMsg1 );

        }
        else if (!msgServiced2)
        {
            // new job arrival and server 1 is active, server 2 is idle => start on
server 2
            arrival( msg );
            msgServiced2 = msg;
            simtime_t serviceTime = startService( msgServiced2 );
            scheduleAt( simTime()+serviceTime, endServiceMsg2 );

        }
        else
        {
            // new arrival but no idlle server => queue it
            arrival( msg );
            queue.insert( msg );
        }
    }
}

void FF2AbstractFifo::finish()
{
    ev << "*** Module: " << fullPath() << "***" << endl;
    ev << "Stack allocated:      " << stackSize() << " bytes";
    ev << " (includes " << ev.extraStackForEnvir() << " bytes for environment)" <<
endl;
    ev << "Stack actually used: " << stackUsage() << " bytes" << endl;

    ofstream of(par("hist_file"), ios::app);
    int samples = hist.samples();
    int i = 0;
    int s = 0;
    of << endl << endl << "*********** new run ***********" << endl << endl << "count:
" << samples << endl << "bucket, count, rel. perc" << endl;
    while ((s<samples) && (i<hist.cells())) {
        of << i << ", " << (int) hist.cell(i) << ", " << hist.cell(i)/samples << endl;
        s+=hist.cell(i);
        i++;
    }
/*
    FILE *f;
    f=fopen(par("hist_file"),"a+");
    hist.saveToFile(f);
    fclose(f);*/
}

//-------------------------------------------------

Define_Module( FF2PacketFifo );

simtime_t FF2PacketFifo::startService(cMessage *msg)
{
    ev << "Starting service of " << msg->name() << endl;
    return par("service_time");
}

void FF2PacketFifo::endService(cMessage *msg)
{
    ev << "Completed service of " << msg->name() << endl;
    send( msg, "out" );
}
```

Remark: The complete program is included in the accompanied Zip archive incl. a Windows binary.

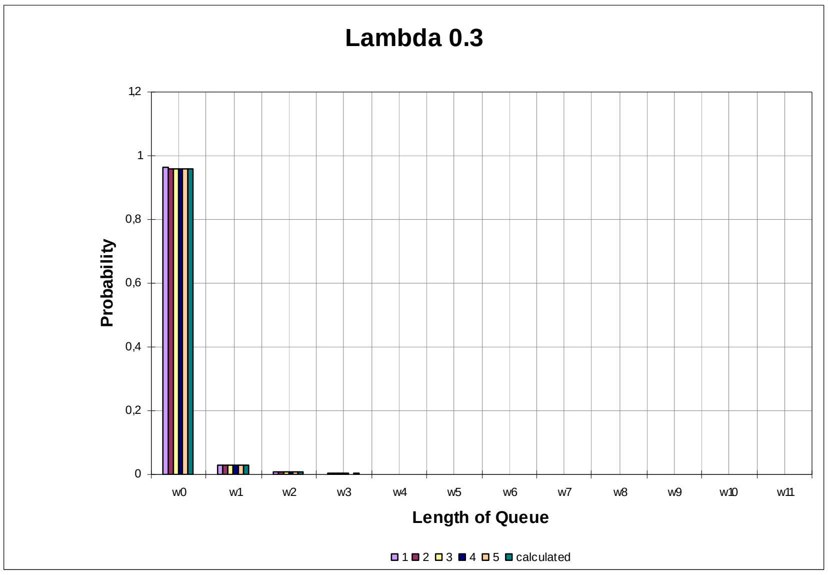
The analysis of the data generated by our OMNet++ program was done using Microsoft Excel for a second time:



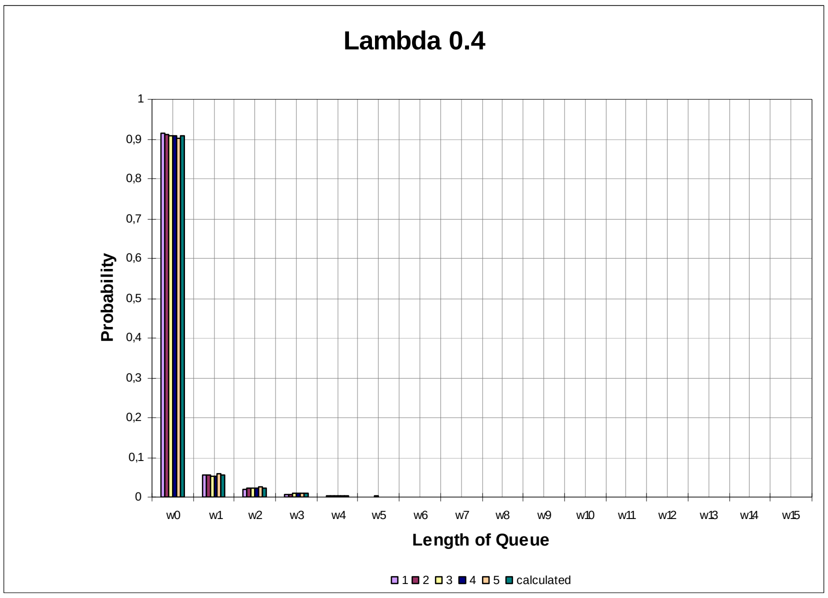
run	1	2	3	4	5	calculated
av. SRT	1,02402	1,00154	1,00481	1,00815	0,992099	1,010101
w0	0,998412	0,998116	0,99815	0,99825	0,998166	0,998182
w1	0,001417	0,001711	0,001741	0,001534	0,001714	0,001636
w2	0,000171	0,000129	0,000109	0,000211	0,00012	0,000164
w3	0	4,37E-05		6,01E-06		1,64E-05
w4						1,64E-06
w5						1,64E-07
w6						1,64E-08



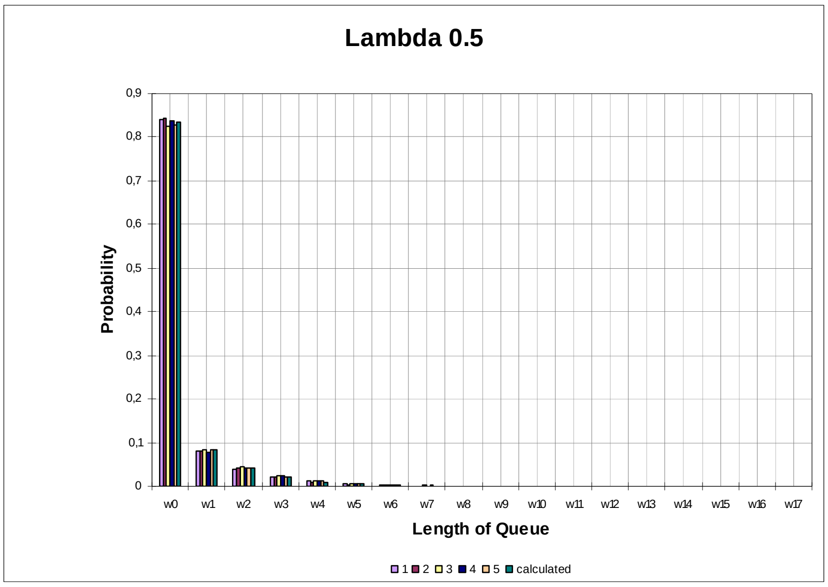
run	1	2	3	4	5	calculated
av. SRT	1,03434	1,03008	1,02436	1,04157	1,0408	1,041667
w0	0,987497	0,987126	0,987842	0,987223	0,988154	0,986667
w1	0,010006	0,010218	0,009724	0,010401	0,009443	0,010667
w2	0,001966	0,001992	0,00198	0,00187	0,001971	0,002133
w3	0,000424	5,66E-04	0,000337	0,000413	0,000385	0,000427
w4	9,51E-05	8,55E-05	0,000104	5,67E-05	3,93E-05	8,53E-05
w5	1,19E-05	1,22E-05	1,2E-05	3,64E-05	7,85E-06	1,71E-05
w6						3,41E-06
w7						6,83E-07
w8						1,37E-07



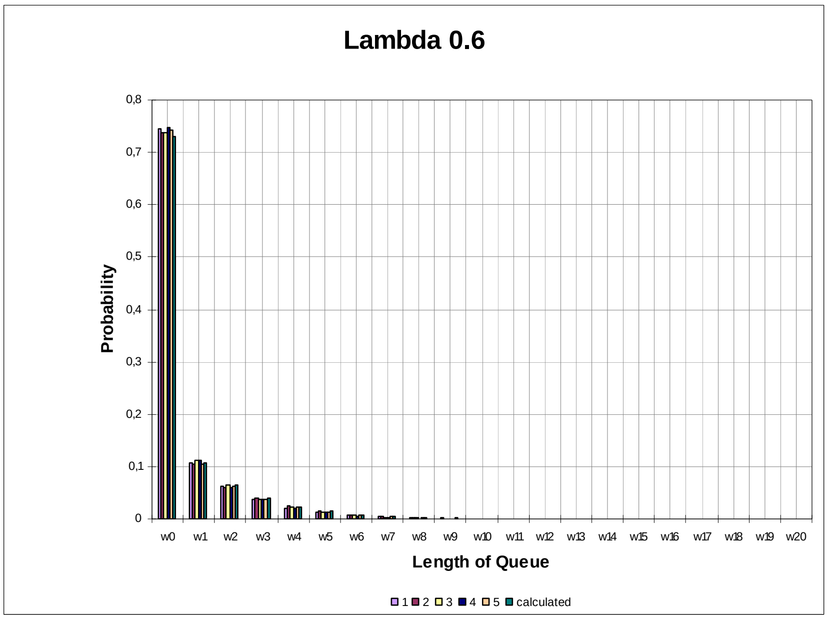
run	1	2	3	4	5	calculated
av. SRT	1,08738	1,09529	1,1064	1,11825	1,09652	1,098901
w0	0,961931	0,959802	0,958691	0,957746	0,95728	0,958462
w1	0,027306	0,028887	0,028284	0,029093	0,030037	0,029077
w2	0,007599	0,007985	0,008574	0,008587	0,009227	0,008723
w3	0,00241	2,23E-03	0,003124	0,002943	0,001875	0,002617
w4	0,000497	0,000705	0,001053	0,001016	0,000775	0,000785
w5	0,00017	0,000344	0,000274	0,000467	0,000313	0,000236
w6	8,19E-05	4,74E-05		7,09E-05	0,000385	7,07E-05
w7	5,85E-06			7,68E-05	7,21E-05	2,12E-05
w8					3,61E-05	6,36E-06
w9						1,91E-06
w10						5,72E-07
w11						1,72E-07



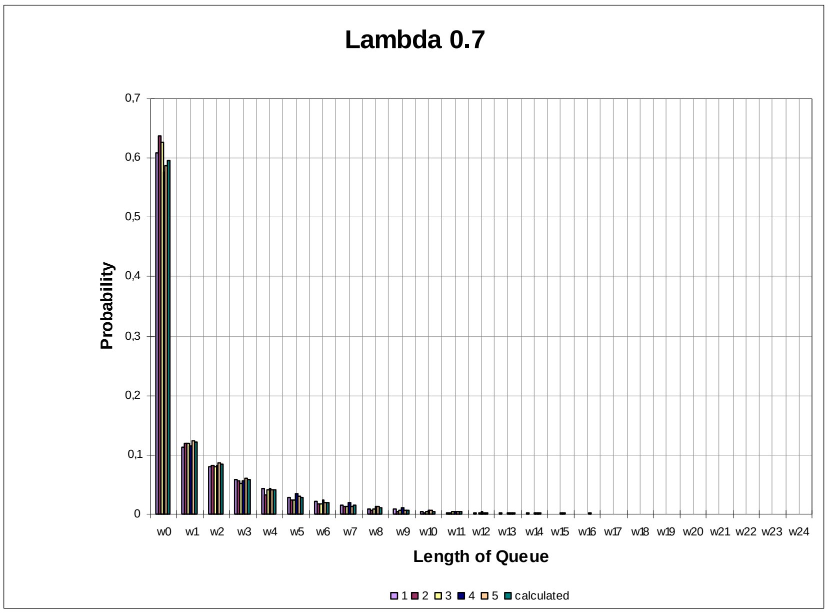
run	1	2	3	4	5	calculated
av. SRT	1,15899	1,1926	1,20358	1,21632	1,21632	1,190476
w0	0,913339	0,909849	0,90775	0,909676	0,900912	0,908571
w1	0,055638	0,056616	0,052665	0,053519	0,058087	0,054857
w2	0,019999	0,021873	0,021556	0,022484	0,02491	0,021943
w3	0,007541	7,29E-03	0,0094	0,008393	0,009373	0,008777
w4	0,002351	0,002361	0,00474	0,003441	0,004133	0,003511
w5	0,000813	0,001532	0,001687	0,001137	0,001509	0,001404
w6	0,000183	0,000426	0,000908	0,000596	0,000835	0,000562
w7	6,38E-05	5,53E-05	0,000603	0,000469	0,000241	0,000225
w8	7,17E-05		0,000386	0,000238		8,99E-05
w9			0,000177	4,77E-05		3,6E-05
w10			5,62E-05			1,44E-05
w11			8,03E-06			5,75E-06
w12			6,43E-05			2,3E-06
w13						9,2E-07
w14						3,68E-07
w15						1,47E-07



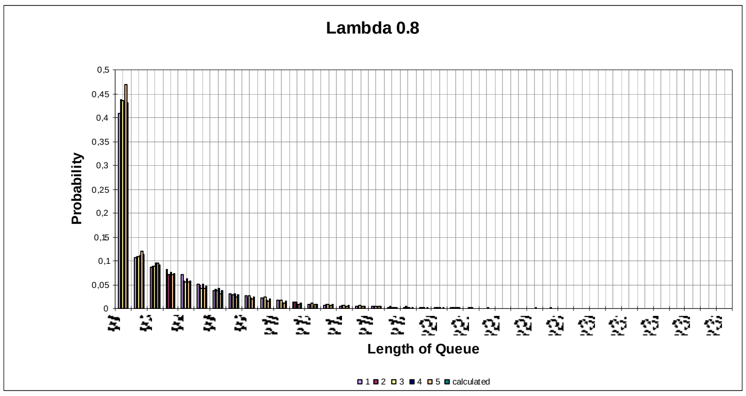
run	1	2	3	4	5	calculated
av. SRT	1,32481	1,27708	1,37664	1,32088	1,36301	1,333333
w0	0,83897	0,844304	0,824401	0,836303	0,82916	0,833333
w1	0,080885	0,080029	0,083138	0,076712	0,083413	0,083333
w2	0,039856	0,040891	0,046105	0,043199	0,042336	0,041667
w3	0,020864	2,01E-02	0,023788	0,024046	0,020544	0,020833
w4	0,011486	0,007806	0,010785	0,011128	0,011833	0,010417
w5	0,005142	0,004162	0,005272	0,00542	0,006105	0,005208
w6	0,001714	0,001583	0,003084	0,002218	0,002517	0,002604
w7	0,000719	0,000946	0,001572	0,000746	0,002309	0,001302
w8	0,000227	0,000139	0,000665	0,000229	0,000892	0,000651
w9	0,000108	9,96E-06	0,000312		0,000783	0,000326
w10	1,97E-05		0,000171		0,000109	0,000163
w11	9,85E-06		0,000151			8,14E-05
w12			0,000181			4,07E-05
w13			0,000242			2,03E-05
w14			0,000131			1,02E-05
w15						5,09E-06
w16						2,54E-06
w17						1,27E-06



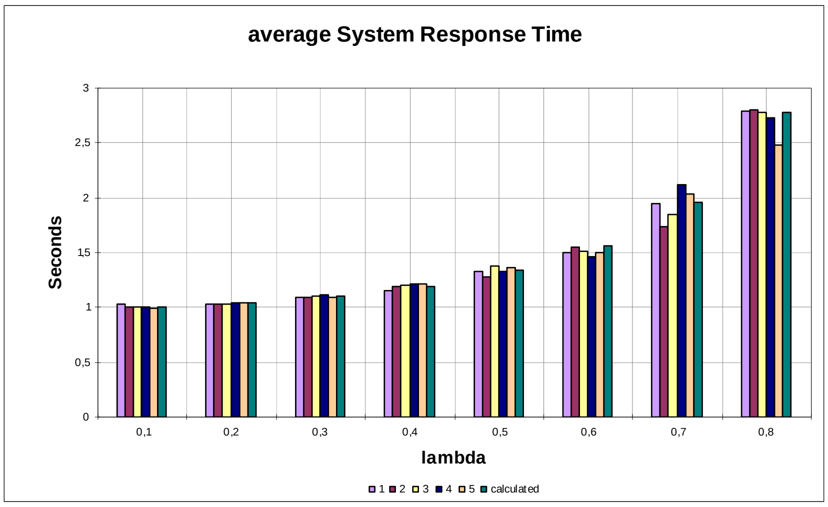
run	1	2	3	4	5	calculated
av. SRT	1,49692	1,54439	1,51567	1,46048	1,50551	1,5625
w0	0,746341	0,736754	0,736558	0,747135	0,743439	0,73
w1	0,10665	0,104946	0,112284	0,111574	0,103917	0,108
w2	0,06185	0,059902	0,06448	0,060567	0,063141	0,0648
w3	0,037117	3,95E-02	0,038074	0,036945	0,037634	0,03888
w4	0,021045	0,02539	0,022091	0,021149	0,021567	0,023328
w5	0,012255	0,015456	0,012492	0,012821	0,012592	0,013997
w6	0,006587	0,00861	0,006407	0,005747	0,007986	0,008398
w7	0,004006	0,003935	0,00306	0,002927	0,005296	0,005039
w8	0,001862	0,001765	0,001829	0,001004	0,002345	0,003023
w9	0,001473	0,000894	0,000873	0,000131	0,000738	0,001814
w10	0,00033	0,000584	0,000586		0,000488	0,001088
w11	0,000436	0,000692	0,000657		0,000214	0,000653
w12	4,71E-05	0,000286	0,000323		0,000333	0,000392
w13		0,000656	0,000263		0,000131	0,000235
w14		0,00031	2,39E-05		0,000179	0,000141
w15		2,39E-05				8,46E-05
w16		0,000167				5,08E-05
w17		0,000143				3,05E-05
w18						1,83E-05
w19						1,1E-05
w20						6,58E-06



run	1	2	3	4	5	calculated
av. SRT	1,94744	1,73237	1,85033	2,11762	2,02985	1,960784
w0	0,610013	0,63815	0,626592	0,576754	0,58784	0,596471
w1	0,113243	0,1201	0,118877	0,114911	0,122949	0,121059
w2	0,081168	0,08249	0,079957	0,082945	0,085848	0,084741
w3	0,057609	5,54E-02	0,05292	0,056376	0,060255	0,059319
w4	0,042346	0,033206	0,040474	0,044263	0,040221	0,041523
w5	0,02899	0,023222	0,023627	0,033797	0,030176	0,029066
w6	0,021269	0,0176	0,017301	0,023516	0,019087	0,020346
w7	0,01455	0,012947	0,012446	0,018929	0,013542	0,014242
w8	0,009243	0,007159	0,008843	0,012965	0,013375	0,00997
w9	0,007624	0,004653	0,006065	0,010523	0,007147	0,006979
w10	0,004388	0,002326	0,005075	0,006404	0,005949	0,004885
w11	0,00266	0,001288	0,003424	0,005098	0,00333	0,00342
w12	0,002578	0,000983	0,002104	0,003323	0,002619	0,002394
w13	0,0024	0,000388	0,000839	0,003053	0,001909	0,001676
w14	0,001358	9,69E-05	0,000935	0,002528	0,001658	0,001173
w15	0,000357		0,000303	0,001463	0,001769	0,000821
w16	5,49E-05		9,63E-05	0,000866	0,001435	0,000575
w17	0,000151		9,63E-05	0,000866	0,000641	0,000402
w18			2,75E-05	0,000426	0,000251	0,000282
w19				0,000611		0,000197
w20				0,000227		0,000138
w21				0,000156		9,66E-05
w22						6,76E-05
w23						4,73E-05
w24						3,31E-05



run	1	2	3	4	5	calculated
av. SRT	2,79439	2,7958	2,77929	2,72989	2,48392	2,777778
w0	0,409483	0,438591	0,434795	0,434183	0,469562	0,431111
w1	0,106806	0,109193	0,109057	0,111276	0,120165	0,113778
w2	0,087305	0,087868	0,089183	0,095588	0,095466	0,091022
w3	0,081671	7,11E-02	0,070162	0,075629	0,071972	0,072818
w4	0,070209	0,057887	0,05555	0,061492	0,054788	0,058254
w5	0,051789	0,048431	0,043111	0,05172	0,041598	0,046603
w6	0,038115	0,038927	0,037318	0,04286	0,030977	0,037283
w7	0,032142	0,0288	0,02961	0,030114	0,023589	0,029826
w8	0,026815	0,022027	0,026005	0,02191	0,01988	0,023861
w9	0,022795	0,01888	0,023414	0,017032	0,015599	0,019089
w10	0,018533	0,013386	0,017766	0,011691	0,010558	0,015271
w11	0,014061	0,013066	0,014322	0,008156	0,008783	0,012217
w12	0,009412	0,009712	0,012053	0,004334	0,008862	0,009773
w13	0,006361	0,007971	0,008416	0,003007	0,007181	0,007819
w14	0,004762	0,006182	0,005616	0,003262	0,004899	0,006255
w15	0,004908	0,004504	0,006002	0,001983	0,00363	0,005004
w16	0,004117	0,004776	0,004586	0,001583	0,004153	0,004003
w17	0,002648	0,004185	0,002349	0,001519	0,001982	0,003203
w18	0,002502	0,00337	0,00243	0,001775	0,000935	0,002562
w19	0,001453	0,002156	0,001802	0,002415	0,000713	0,00205
w20	0,001227	0,001949	0,001545	0,002639	0,000999	0,00164
w21	0,001227	0,001262	0,001625	0,002943	0,001189	0,001312
w22	0,00071	0,001454	0,001143	0,003135	0,001046	0,001049
w23	8,07E-05	0,000926	0,000949	0,002207	0,000856	0,00084
w24	4,84E-05	0,000671	0,000628	0,000992	0,000285	0,000672
w25	0,000371	0,00024	0,000434	0,000848	0,000254	0,000537
w26	0,000178	0,000128	9,66E-05	0,001407	7,93E-05	0,00043
w27	0,000161	0,000463	3,22E-05	0,001743		0,000344
w28	4,84E-05	0,000639		0,0008		0,000275
w29	6,46E-05	4,79E-05		0,000432		0,00022
w30		0,000256		0,000416		0,000176
w31		0,000272		0,000352		0,000141
w32		0,000144		0,00032		0,000113
w33		0,000272		0,000112		9,01E-05
w34		0,000192		0,000128		7,21E-05
w35		6,39E-05				5,77E-05
w36						4,62E-05
w37						3,69E-05
w38						2,95E-05



run	1	2	3	4	5	calculated
0,1	1,02402	1,00154	1,00481	1,00815	0,992099	1,010101
0,2	1,03434	1,03008	1,02436	1,04157	1,0408	1,041667
0,3	1,08738	1,09529	1,1064	1,11825	1,09652	1,098901
0,4	1,15899	1,1926	1,20358	1,21632	1,21632	1,190476
0,5	1,32481	1,27708	1,37664	1,32088	1,36301	1,333333
0,6	1,49692	1,54439	1,51567	1,46048	1,50551	1,5625
0,7	1,94744	1,73237	1,85033	2,11762	2,02985	1,960784
0,8	2,79439	2,7958	2,77929	2,72989	2,48392	2,777778

Problem 2

Use the Pollaczek-Khintchine mean value formula to show that a M/M/1 system has twice the expected number of customers in the system as the M/D/1 system as $\rho \to 1$ *(* $\rho < 1$*).*

The Pollaczek-Khintchine mean value formula:

$$E[n] = \rho + \frac{\rho^2 \cdot \left(1 + C_b^2\right)}{2 \cdot (1-\rho)}$$

That formula can be applied to both the M/M/1 and the M/D/1 case. The term C_b^2 differs, in the M/M/1 case:

$$C_b^2 = \frac{1}{\lambda^2} \cdot \lambda^2 = 1$$

And for M/D/1:

$$C_b^2 = \frac{0}{\mu} = 0$$

We find that the expected number of customers in a M/M/1 system is determined by:

$$\begin{aligned} E_{M/M/1}[n] &= \rho + \frac{\rho^2 \cdot (1+1)}{2 \cdot (1-\rho)} \\ &= \rho + \frac{2\rho^2}{2 \cdot (1-\rho)} \\ &= \rho + \frac{\rho^2}{1-\rho} \\ &= \frac{\rho}{1-\rho} \end{aligned}$$

Simplifying the Pollaczek-Khintchine formula of a M/D/1 system:

$$\begin{aligned} E_{M/D/1}[n] &= \rho + \frac{\rho^2 \cdot (1+0)}{2 \cdot (1-\rho)} \\ &= \frac{\rho \cdot (2-\rho)}{2 \cdot (1-\rho)} \end{aligned}$$

Let’s compute the limit:

$$
\begin{aligned}
\lim_{\rho \to 1} \frac{E_{M/M/1}[n]}{E_{M/D/1}[n]} &= \frac{\frac{\rho}{1-\rho}}{\frac{\rho \cdot (2-\rho)}{2 \cdot (1-\rho)}} \\
&= \frac{\rho}{1-\rho} \cdot \frac{2 \cdot (1-\rho)}{\rho \cdot (2-\rho)} \\
&= \frac{2\rho}{\rho \cdot (2-\rho)} \\
&= \frac{2}{2-\rho} \\
&= \frac{2}{2-1} \\
&= 2
\end{aligned}
$$

Indeed, a M/M/1 system has twice the expected number of customers in a system as the M/D/1 system for large ρ ($\rho \to 1$, $\rho < 1$).

Problem 3

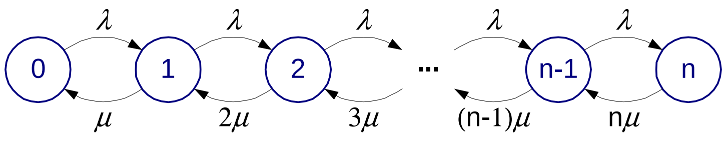
Consider the M/M/N/N loss system ($N \geq 1$) with arrival rate λ and μ being the rate of a single server.

- *Draw the state diagram*
- *Give the generator matrix Q*
- *Find the steady-state vector $\pi = (\pi_0, \pi_1, ..., \pi_N)$*
- *Using this, show that with $\rho = \frac{\lambda}{\mu}$:*

$$\Pr[\text{"Customer loss"}] = \pi_N = \frac{\rho^N}{N!} \cdot \frac{1}{\sum_{k=0}^{N} \frac{\rho^k}{k!}} \qquad \textit{(this is the Erlang loss formula)}$$

- *Assume $\rho = \frac{\lambda}{\mu} = 5$ and evaluate $\Pr[\text{"Customer loss"}]$ for $N = 1, 2, 3, ..., 20$.*
- *How might a telephone company use this formula ?*

If any of the servers in the M/M/N/N queue is idle, the arriving job is serviced immediately. If all server are busy, the arriving job waits in a queue. The state of the system is represented by the number of jobs in the system. Thus the state transition diagram looks as follows:



The corresponding $(n+1)\times(n+1)$ generator matrix Q can be derived instantly from the diagram shown above:

$$Q = \begin{pmatrix}
-\lambda & \lambda & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
\mu & -(\lambda+\mu) & \lambda & 0 & 0 & & 0 & 0 & 0 \\
0 & 2\mu & -(\lambda+2\mu) & \lambda & 0 & & 0 & 0 & 0 \\
0 & 0 & 3\mu & -(\lambda+3\mu) & \lambda & & 0 & 0 & 0 \\
\vdots & & & & & \ddots & & \vdots & \\
0 & 0 & 0 & 0 & 0 & & -(\lambda+(n-2)\cdot\mu) & \lambda & 0 \\
0 & 0 & 0 & 0 & 0 & & (n-1)\cdot\mu & -(\lambda+(n-1)\cdot\mu) & \lambda \\
0 & 0 & 0 & 0 & 0 & & 0 & n\mu & -n\mu
\end{pmatrix}$$

The steady-state vector can be inferred from Q since each row equals zero. Therefore:

$$0 = \pi \cdot Q$$

And more in detail:

$$\begin{aligned}
0 &= \mu\pi_1 - \lambda\pi_0 \\
0 &= \mu\pi_2 - (\lambda+\mu)\cdot\pi_1 + \lambda\pi_0 \\
0 &= 2\mu\pi_3 - (\lambda+2\mu)\cdot\pi_2 + \lambda\pi_1 \\
0 &= 3\mu\pi_4 - (\lambda+3\mu)\cdot\pi_3 + \lambda\pi_2 \\
&\ldots \\
0 &= n\mu\pi_{n+1} - (\lambda+n\mu)\cdot\pi_n + \lambda\pi_{n-1}
\end{aligned}$$

Solving these equations depending on π_0 yields:

$$\begin{aligned}
\pi_1 &= \frac{\lambda}{\mu}\cdot\pi_0 \\
\pi_2 &= \frac{\lambda^2}{2\mu^2}\cdot\pi_0 \\
\pi_3 &= \frac{\lambda^3}{6\mu^3}\cdot\pi_0 \\
\pi_4 &= \frac{\lambda^4}{24\mu^4}\cdot\pi_0 \\
&\ldots \\
\pi_n &= \frac{\lambda^n}{n!\cdot\mu^n}\cdot\pi_0
\end{aligned}$$

A closer look to the diagram reveals that a M/M/N/N queue is a birth-death process. Therefore:

$$\begin{aligned}
\lambda_i &= \lambda \\
\mu_i &= i\cdot\mu \\
\pi_i &= \frac{\lambda_i}{\mu_i}\cdot\pi_{i-1} \\
&= \frac{\lambda}{n\cdot\mu}\cdot\pi_{i-1} \\
&= \frac{\lambda^i}{i!\cdot\mu^i}\cdot\pi_0
\end{aligned}$$

According to the normalization condition:

$$\sum_{k=0}^{N}\pi_k = 1$$

We solve for π_0:

$$\begin{aligned} 1 &= \sum_{k=0}^{N} \pi_k \\ &= \pi_0 + \sum_{k=1}^{N} \pi_k \\ &= \pi_0 + \sum_{k=1}^{N} \frac{\lambda^k}{\mu^k \cdot k!} \cdot \pi_0 \\ &= \pi_0 \cdot \left(1 + \sum_{k=1}^{N} \frac{\lambda^k}{\mu^k \cdot k!} \right) \\ \pi_0 &= \left(1 + \sum_{k=1}^{N} \frac{\lambda^k}{\mu^k \cdot k!} \right)^{-1} \end{aligned}$$

Since $\frac{\lambda^k}{\mu^k \cdot k!} = 1$ for $k = 0$ the sum can be extended to:

$$\pi_0 = \left(\sum_{k=0}^{N} \frac{\lambda^k}{\mu^k \cdot k!} \right)^{-1}$$

Using π_0:

$$\pi_i = \frac{\lambda^i}{i! \cdot \mu^i \cdot \sum_{k=0}^{N} \frac{\lambda^k}{\mu^k \cdot k!}}$$

Substituting $\rho = \frac{\lambda}{\mu}$ and $i = N$:

$$\pi_N = \frac{\rho^N}{N! \cdot \sum_{k=0}^{N} \frac{\rho^k}{k!}}$$

That way we proved Erlang’s loss formula. Similarly, we get the steady-state vector:

$$\pi = \left(\frac{1}{\sum_{k=0}^{N} \frac{\rho^k}{k!}}, \quad \frac{\rho}{\sum_{k=0}^{N} \frac{\rho^k}{k!}}, \quad \frac{\rho^2}{2 \cdot \sum_{k=0}^{N} \frac{\rho^k}{k!}}, \quad \frac{\rho^3}{6 \cdot \sum_{k=0}^{N} \frac{\rho^k}{k!}}, \quad \cdots, \quad \frac{\rho^N}{N! \cdot \sum_{k=0}^{N} \frac{\rho^k}{k!}} \right)$$

A modern math suite computes the customer loss very quickly. We wrote a short program for Maple 8 Trial version. Its two lines are:

```
rho:=5:
erlang:=N->rho^N/(N!*sum((rho^k)/k!, k=0..N));
```

The program is targeted at high precision computations and therefore outputs chunky ratios. However, we also wanted to obtain rounded values:

```
result:=seq(erlang(N), N=1..20);
```

$$result := \frac{5}{6}, \frac{25}{37}, \frac{125}{236}, \frac{625}{1569}, \frac{625}{2194}, \frac{3125}{16289}, \frac{15625}{129648}, \frac{78125}{1115309}, \frac{390625}{10428406}, \frac{390625}{21247437}, \frac{1953125}{235674932}, \frac{9765625}{2837864809}, \frac{48828125}{36941070642}, \frac{244140625}{517419129613}, \frac{244140625}{1552501529464}, \frac{1220703125}{24841245174549}, \frac{6103515625}{422307271482958}, \frac{30517578125}{7601561404271369}, \frac{152587890625}{144429819269046636}, \frac{152587890625}{577719429664077169}$$

```
resultf:=evalf(result);
```

$$resultf := 0.8333333333, 0.6756756757, 0.5296610169, 0.3983428936, 0.2848678213, 0.1918472589, 0.1205186351, 0.07004785221, 0.03745778597, 0.01838457034, 0.008287368467, 0.003441187533, 0.001321784240, 0.0004718430592, 0.0001572562863, 0.00004914017459, 0.00001445278364, 0.4014646005\ 10^{-5}, 0.1056484675\ 10^{-5}, 0.2641210989\ 10^{-6}$$

In a M/M/N/N system, queueing is not necessary since there are as many servers as customers. So if a customer is able to actually enter the system then he/she will be served, too. Otherwise, he/she will be rejected and does not enter the system at all.

The capacity of a telephone network (a backbone) ought to be as high as needed in order to master the case that each customer calls/arrives at the same time (up to N calls). In reality that case was never observed, hence, the companies reduce their capacities to the typical case in order to cut lots of costs.

The telecommunication companies aim to serve as many customers as possible and to refuse as few as feasible. Due to long-term observations, it is approximately known in advance how many new calls per time have to be served and how long they last in average. Now the companies define some percentages they like to achieve, i.e., there are 1,000 new calls per minute, they last about 10 minutes each and only 0.01% should be rejected. Then:

$$\lambda = \frac{1}{1000}$$

$$\mu = \frac{1}{10}$$

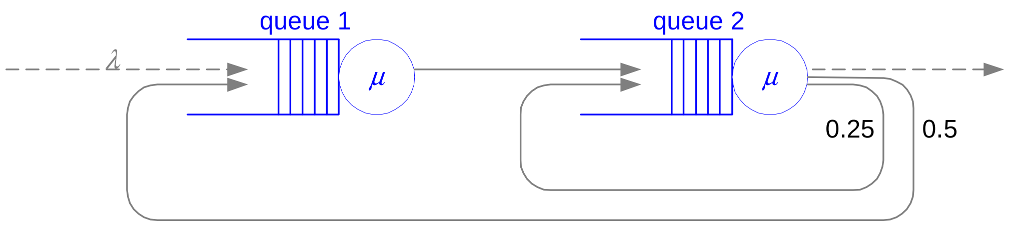
$$\frac{\lambda}{\mu} = 100$$

The final step is to find the smallest N where $\Pr\left[\text{"}Customer\ loss\text{"}\right] < 0.01\%$ according to Erlang's loss formula. That N then denotes the lowest capacity needed to achieve the proposed service availability.

If one has to estimate the loss using a given capacity, throughput and arrival rate, he can utilize Erlang's loss formula, too. As an example, a telephone company gets a chance to approximate how many calls are lost (in average) with their current telephone exchange (or switch). If that number is too high, the company should calculate how many of the lost calls may have been served by installing an additional exchange. Hence, the outcome of all these calculations are future investments (the new telephone exchange) and upcoming earnings (served calls). The responsible manager then can decide upon these facts whether he admits to the new investments or not.

Problem 4

Consider the fully Markovian queueing network shown in this figure:



- *Find a stability condition for this system.*
- *Find the mean time for a customer to proceed through the system.*

From the diagram we infer that the given network is an open network and apply Jackson's theorem. The system's routing probabilities can be described as:

$$\begin{array}{lll} & p_{0,1} = 1 & p_{0,2} = 0 \\ p_{1,0} = 0 & p_{1,1} = 0 & p_{1,2} = 1 \\ p_{2,0} = 0.25 & p_{2,1} = 0.5 & p_{2,2} = 0.25 \end{array}$$

The traffic equations:

$$e_1 = p_{0,1} + p_{1,1} \cdot e_1 + p_{2,1} \cdot e_2$$
$$e_2 = p_{0,2} + p_{1,2} \cdot e_1 + p_{2,2} \cdot e_2$$

Reduced:

$$e_1 = 1 + 0.5 \cdot e_2$$
$$e_2 = e_1 + 0.25 \cdot e_2$$

Solved:

$$e_1 = 3$$
$$e_2 = 4$$
$$\lambda_1 = e_1 \cdot \lambda = 3\lambda$$
$$\lambda_2 = e_2 \cdot \lambda = 4\lambda$$

Queue 1 has to process the tasks three times faster than they enter the system while queue 2 has to be even quicker: it must be able to handle four times the number of arrivals. Now we examine both M/M/1 queues separately. Each node has to fulfill the stability condition $\lambda_i < \mu_i$. Due to $\mu_1 = \mu_2 = \mu$ we find $\mu > 4\lambda$.

The mean time for a customer to proceed through the system is the outcome of the equation:

$$E[t] = e_1 \cdot E[T_1] + e_2 \cdot E[T_2]$$

Until now, the visit ratios are known but we need to further investigate the estimated time in each node:

$$E[T_1] = \frac{1}{\mu_1 - \lambda_1} = \frac{1}{\mu - 3\lambda}$$
$$E[T_2] = \frac{1}{\mu_2 - \lambda_2} = \frac{1}{\mu - 4\lambda}$$

Hence:

$$\begin{aligned} E[t] &= \frac{3}{\mu - 3\lambda} + \frac{4}{\mu - 4\lambda} \\ &= \frac{7\mu - 24\lambda}{(\mu - 3\lambda)\cdot(\mu - 4\lambda)} \end{aligned}$$