EE 550: Project

Masoud Moshref Javadi

12/5/2011

1 Contents

2 Program Structure

The simulation is written in Java as a discrete event simulator in 3 levels. But all these three levels have a common structure and each level is built on the lower level. So first of all, the common structure will be presented. Then level 1, level 2 and at last level 3 will be discussed.

2.1 Common Structures

A discrete event simulator needs classes of timeline and event. Each event is generated by an object (Caller in this project) in the system and will change the state of simulation. The objects may create events randomly so there are some random number generators. Then we need to compute some statistics. These statistics are computed using some snapshots of state of system created after an event. The following list describes the classes implemented for each fundamental element. An abstract view of the common classes is presented as a class diagram in Figure 30:

- TimeLine: has a sorted list of events and keeps track of current simulation time.
- Event: is an abstract class that all other events inherit from it. For example, SnapShotEvent is an event that shows the system should create a snapshot of the current state.
- Each level implements its own State and Snapshot class that is a subclass of AbstractState and AbstractSnapshot. Note that each snapshot is associated with an event that caused its generation. So we can gather snapshots on any kind of events not only SnapShotEvent and then filter them based on the kind of statistics we want. For example, you can have a snapshot after arrival of a caller. However, I have added SnapShotEvent to keep uniform random view of system not just on arrivals or departures and make number of snapshots independent from number of callers.
- Caller: callers are entities of system and act as directories that keep track of events and the associated random number generators. Besides, each event knows to which Caller it is associated.
- RandomGenerator: There are simple classes that generate random numbers (Figure 33).
- StatisticsCalculator: It is a utility class that can compute mean, variance and the corresponding confidence interval of any property of snapshot objects. There are three points that should be explained about this class:
    - How to compute variance using weighted samples? The weight of a sample is the duration between the current snapshot and the previous one. The variance is computed using the following formula[1]. Where N is the number of samples and M is the number of non-zero weights. w_i represents the weight for sample x_i

$$\sqrt{\frac{\sum_{i=1}^{N} w_i (x_i - \bar{x})^2}{\frac{M-1}{M} \sum_{i=1}^{N} w_i}}$$

[1] http://stats.stackexchange.com/questions/6534/how-do-i-calculate-a-weighted-standard-deviation-in-excel

- Confidence interval for mean: The confidence interval of mean value for weighted samples is computed using the following formula[2], where s_w is the weighted standard deviation computed from formula in the previous paragraph:

$$\bar{x} \pm t_{1-\alpha/2} \frac{s_w}{\sqrt{\sum_{i=1}^{n} w_i}}$$

- Confidence interval for variance: I used the following formula[3] to compute the confidence interval of estimated variance, where S^2 is the estimated variance.

$$\left(\frac{n-1}{\chi_{n-1,1-\alpha/2}^2} S^2 \quad \frac{n-1}{\chi_{n-1,\alpha/2}^2} S^2 \right)$$

Each level of simulation has a procedural part (Simulator) that runs the actual simulation and all of them have a common algorithm. 1) Initialize 2) Pick the earliest event and advance time to it 3) Run the event and update state 4) take snapshot of state if necessary 5) if reach the maximum time of simulation compute statistics using the snapshots. See

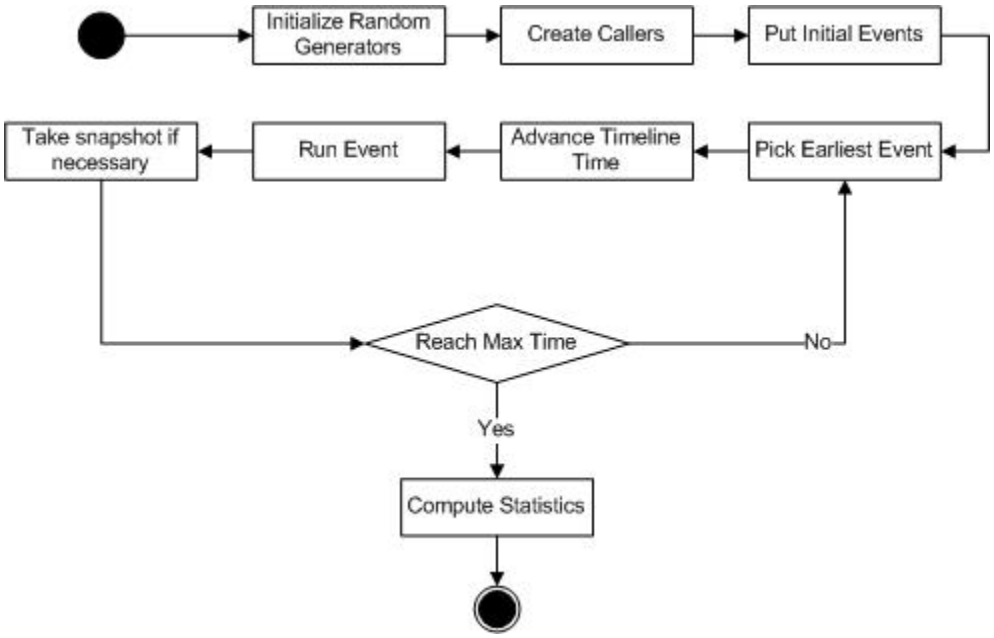

Figure 1: Flowchart of simulation procedure

Figure 31 shows an abstract view of all classes in the system. The figure contains 4 levels: common classes, level 1, level 2, and level 3 classes.

2.2 Level 1

In level 1 (Figure 34), I have implemented two events, L1CallArrival and L2CallDeparture. Besides, the state of the system contains the number of calls in progress, the total number of calls and the total number of blocked callers until that time of simulation.

[2] http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a000608466.htm
[3] http://www.fmi.uni-sofia.bg/vesta/Virtual_Labs/interval/interval6.html

The statistics we want from the simulation is the distribution of active callers and the ratio of blocked calls. Computing the ratio of blocked calls is simple and obtainable from the final state of simulation so no need to keep it in snapshots. But to compute the former statistics, each snapshot should keep the number of active callers.

2.3 Level 2

In level 2 (Figure 35), on call arrivals we need to schedule the next talk period assuming that when a caller calls, it is in silent mode. So I have implemented L2CallArrival event to handle this. L2Talk and L2Silence events control the talking and silence period of a caller. Last and the most important event is L2CallDeparture which should revoke the scheduled talk or silence event. The state of the system needs to keep the number of talking users additional to level 1 state. Besides, there is not any maximum number of callers so the L2CallArrival should ignore this number.

For statistics, we asked to compute the distribution of talking and silent callers. So the snapshots should keep track of the number of talking users in each state additional to level 1 snapshot.

2.4 Level 3

This level (Figure 36) adds the packet level to the system. There are L3PutPacket and L3SendPacket events that put and pop packet from the buffer in the L3State object. Note that for each event, we need to initialize the corresponding random number generator of events. L3Talk event overrides L2Talk to schedule the next L3PutPacket event and L3Silence and L3CallDeparture revoke the scheduled L3PutPacket besides their level 2 corresponding classes functionality. So I assumed that on silence or departure the residual bits will not be sent. I think this assumption is valid as the size of packets is very small. For each packet, we need only the arrival time of it, so a packet is basically a number. To avoid creating a Java object for each packet required by Java LinkedList class, I implemented my own buffer consisting of an array and two rotating pointers. The state of system consists of this buffer, the waiting time of the last packet sent and the total number of blocked packets.

The project specification requires the distribution of buffer occupancy so snapshots should keep track of it. Besides, to compute mean and variance waiting time, the waiting time can be gathered by generating a snapshot at L3SendPacket events. However, there are lots of packets in the system, and we cannot keep track of all of them. Besides writing these values into a file, there are two approximation approaches:

1) Sampling: sample using a uniform random variable.
2) Use an online algorithm: For mean, it is simple; we need to keep just the total sum and number of samples in the state[4].

$$\bar{x}_n = \frac{(n-1)\bar{x}_{n-1} + x_n}{n} = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$

$$M_{2,n} = M_{2,n-1} + (x_n - \bar{x}_n)(x_n - \bar{x}_{n-1})$$

[4] http://en.wikipedia.org/wiki/Algorithms_for_calculating_variance#On-line_algorithm

$$s_n^2 = \frac{M_{2,n}}{n-1}$$

I will use the online algorithm to compute the mean and variance of waiting times.

3 Results

I have run the simulations for 10^7 seconds and took an snapshot every 10 seconds.

3.1 Level 1

3.1.1 Analytic Approach

I modeled the system using a Markov chain when the distribution of call duration is exponential. If $N \geq K$ the chain is as follows:

Figure 2: State diagram of level 1 Markov chain

We can compute the probability of being in each state as follows:

$$p_i = \frac{N-i+1}{i} * \frac{\lambda}{\mu} * p_{i-1}$$

$$p_i = \frac{N!}{i!\,(N-i)!} \left(\frac{\lambda}{\mu}\right)^i p_0$$

$$p_0 \left(1 + \sum_{i=1}^{\min(K,N)} \binom{N}{i} \left(\frac{\lambda}{\mu}\right)^i\right) = 1$$

For $N \leq K$ we can make it simpler:

$$p_0 \left(1 + \frac{\lambda}{\mu}\right)^N = 1$$

So the result is:

$$p_i = \begin{cases} \dfrac{\binom{N}{i}\left(\frac{\lambda}{\mu}\right)^i}{\left(1+\frac{\lambda}{\mu}\right)^N} & N \le K, i \le N \\ \dfrac{\binom{N}{i}\left(\frac{\lambda}{\mu}\right)^i}{1+\sum_{i=1}^{K}\binom{N}{i}\left(\frac{\lambda}{\mu}\right)^i}, & N > K, i \le K \end{cases}$$

If we put the actual values in the formula we get Figure 4 which is very similar to the simulation result in Figure 5. The Table 1 presenting the mean error between two approaches also approves this similarity.

Table 1: Mean error between analytic and simulation values for the distribution of calls in progress

	N=10	N=100	N=1000
Mean error	0.000467	0.000402	3.96E-05

Knowing the probability of each state we can compute the blocking probability:

$$p_b = \frac{p_K(N-K)\lambda}{\sum_{i=0}^{K} p_i(N-i)\,\lambda}$$

$$p_b = \begin{cases} 0 & , \quad N \le K \\ \dfrac{p_K(N-K)\lambda}{\sum_{i=0}^{K} p_i(N-i)\,\lambda}, & N > K \end{cases}$$

Figure 6 compares the result of analysis and simulation for the probability of blocking a call. It shows that there is no deviation between two curves.

3.1.2 Simulation Result

Besides the discussed results in Figure 5 and Figure 6, Figure 7, Figure 8 and Figure 9 show that the distribution of number of active callers does not depend on the exact distribution of call duration while the mean is similar. So Figure 10, Figure 11 and Figure 12 have been computed only for exponential call duration. Figure 7 shows that for 10 callers, there are so few active callers in the system and according to Figure 12 there is no blocked call. However, as the number of calls increases, we expect that the distribution moves to the right. Figure 8 approves this expectation. Figure 10 also shows that the mean of this distribution increases as the number of callers increases until it approaches 24 asymptotically. The interesting point here is that the variance of the number of active callers is maximum around 200 (Figure 11). Figure 13, presenting the ratio of accepted and blocked calls, approves this trend that the number of blocked callers is small until 100 users while it increases after that.

3.2 Level 2

3.2.1 Analytic Approach

To analyze the system, I defined the state of system as (number of talking callers, number of calls in progress). As a result, I got the Markov chain in Figure 3. α is the rate of talking and β is the rate of silence. I assumed that a new user will be silent in the beginning but when a user departs it can be either

in talk or silence mode. So for state (i, j) with calls, silent users depart with rate $(j - i)\mu$ and talking users depart with rate $i * \mu$.

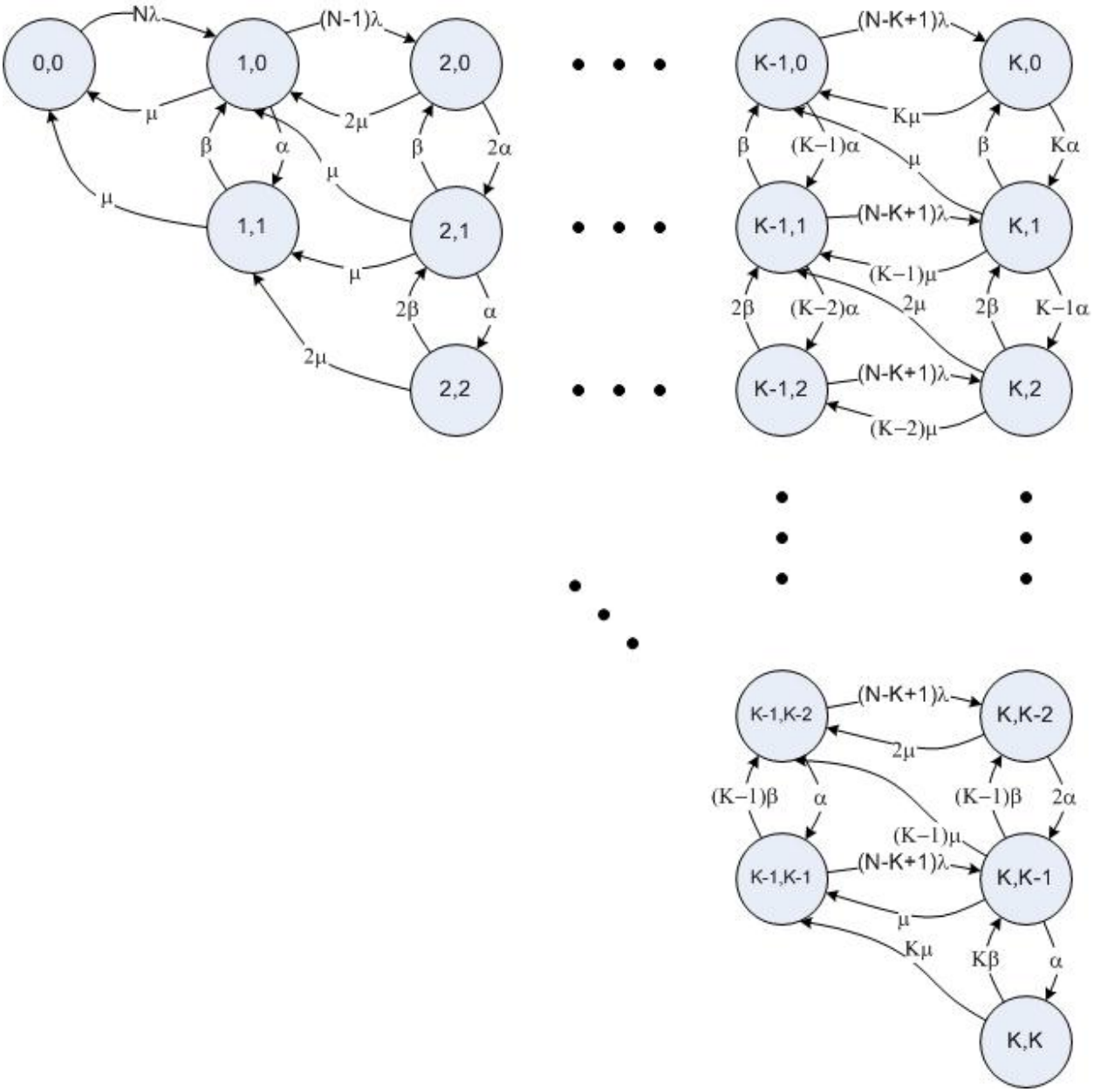

Figure 3: Markov chain model of system in level 2

Solving this system analytically is very hard so I created a linear system to solve this Markov chain: assume q_{ij} is the rate from state i to state j while q_{ii} is $-1 * output\ rate\ from\ state\ i$. If Q is the matrix of qs, we need to solve the following linear system:

$$\begin{cases} QP = 0 \\ \sum p_i = 1 \end{cases}$$

After finding the probability of each state, we can compute the distribution of number of calls in progress by sum of probabilities of states in each column. The result is exactly the same as level 1 results if we do not set $K = N$ and have blocking.

By summing the probability of states in each row, we can find the distribution of the number of talking users. Table 2 shows that the average error in simulation is very small. For $N = 100$, there are 5152 states and for N=1000 the memory is not enough to keep these states and if it was possible, the time to solve the linear system will be too long. Figure 14 depicts the distribution of talking users in analysis and simulation.

Table 2: Mean error between analytic and simulation values for the distribution of talking callers

	N=10	N=100	N=1000
Mean error	0.000237	1.61829E-05	?

7.1.2 shows the listing of MATLAB codes used in this part.

3.2.2 Simulation Result

I ran the simulation for 10^7 seconds and got snapshots using a uniform distribution with mean 10 seconds. Figure 15, Figure 16, and Figure 17 show that there is not much difference in the distribution of talking users among different call duration distributions. The resulting diagrams seems rational as the utilization of system in level 1 without blocking is $\frac{1}{10}$ so for $N = 1000$ we may assume 100 calls are in progress. From these 100 calls the talking period is $\frac{1}{3}$ of total call period. So we can expect that $\frac{100}{3}$ calls should be in progress in average. The results shows that the average number of talking callers for N=1000 is 30 which is close to our expectation.

Moreover, Figure 18, Figure 19 and Figure 20 show the distribution of silent users in the system which from the results in level 1 and previous part it was expected that all three distributions generate the same diagram. Using these diagrams, we can find out that the average number of calls in progress (talking + silent). Table 3 shows that the approximation in the previous paragraph is not so off.

Table 3: Average number of talking, silent and total calls in progress

	N=10	N=100	N=1000
Mean talking	0.300763	3.0251	30.18976
Mean Silent	0.605857	6.077216	60.70806
Sum	0.90662	9.102316	90.89782

The last point to check is the variation of mean and variance of the number of talking callers based on the number of callers in the system. Figure 21 and Figure 22 show the mean and variance of talking

users and presents their linear dependency on the number of users. This trend does not depend on the call duration distribution.

3.3 Level 3

3.3.1 Analytic Approach

Each user generates 80 packets per second of talking and the connection line can send only $\frac{1.54*10^6}{1000} = 1540$ packets per second. So we expect to be able to handle about 19 talking calls. Using simulation results of level 2 in Figure 21, we can see that 600 callers can generate 19 talking calls in average. So we expect to be able to handle at most 600 users. So for 1000 users should be a high drop rate.

3.3.2 Simulation Result

For N=10 and 100, I ran the simulation for 10^6 seconds but for N=1000 I ran the simulation only for 10^5 seconds as it contains $4 * 10^8$ events and has $1.5 * 10^8$ packet sending which I think is enough for the statistics.

Figure 23 shows the distribution of buffer occupancy in with N=10, 100, and 1000 for exponential call duration. The result is the same for other call duration distributions. This diagram approves the above expectation. In the next step, I computed filled buffer, waiting time and drop rate for different number of callers to get a more detailed view on these statistics. Figure 24 and Figure 26 show that the mean buffer occupancy and packet wait time is small for less than 400 users. In this case the drop rate is also small in Figure 28. Figure 25 shows that we have the largest variance for 600 callers. This means that we there is a high variation in buffer usage and may be it can be a sign that it uses the buffer for absorbing packet generation variation efficiently. However, we should note that the waiting time and drop rate is a little higher than 400 users.

I have tested the simulation with a bimodal call duration distribution with P(X =1)=0.95; P(X =44)=0.05, which has higher variation. The result shows a little higher variance in buffer occupancy and higher drop rate comparing to exponential case (Figure 29).

4 Conclusion

The simulation showed that the packet multiplexing system could handle 400 users instead of 24 users in circuit switching. Besides, it gives us a trade-off between the quality of calls (packet drop rate, delay) and the number of users. For example, if the system could sustain 5% error rate and 2.5 ms, we can support 600 users.

5 Appendix 1: Simulation Figures

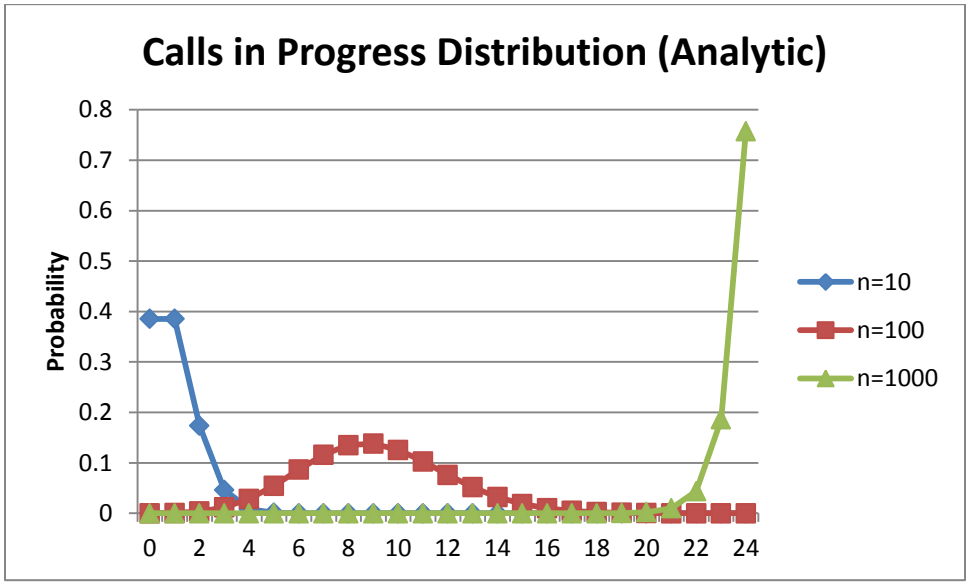

Figure 4: Distribution of the number of active callers by analytic approach for exponential call duration

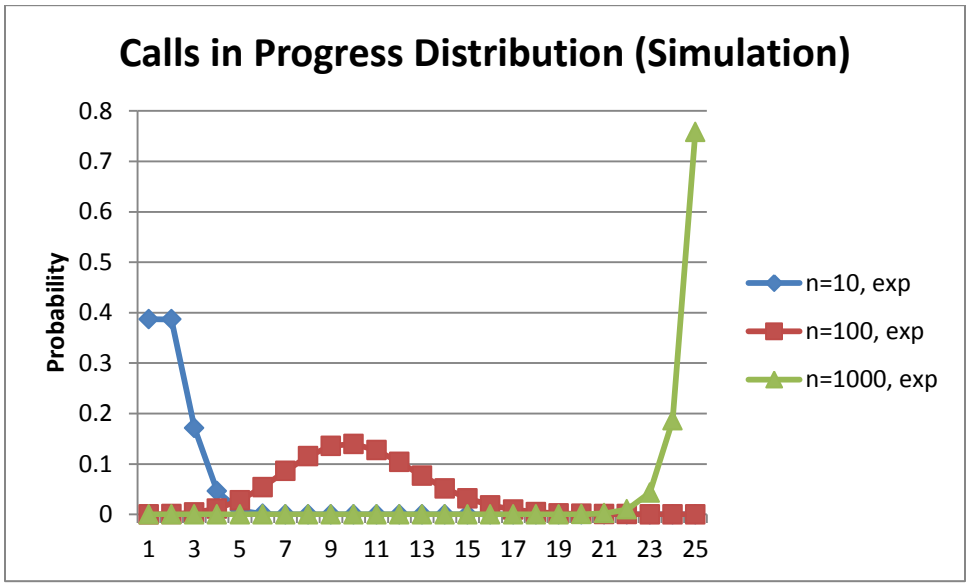

Figure 5: Distribution of the number of active callers by simulation approach for exponential call duration

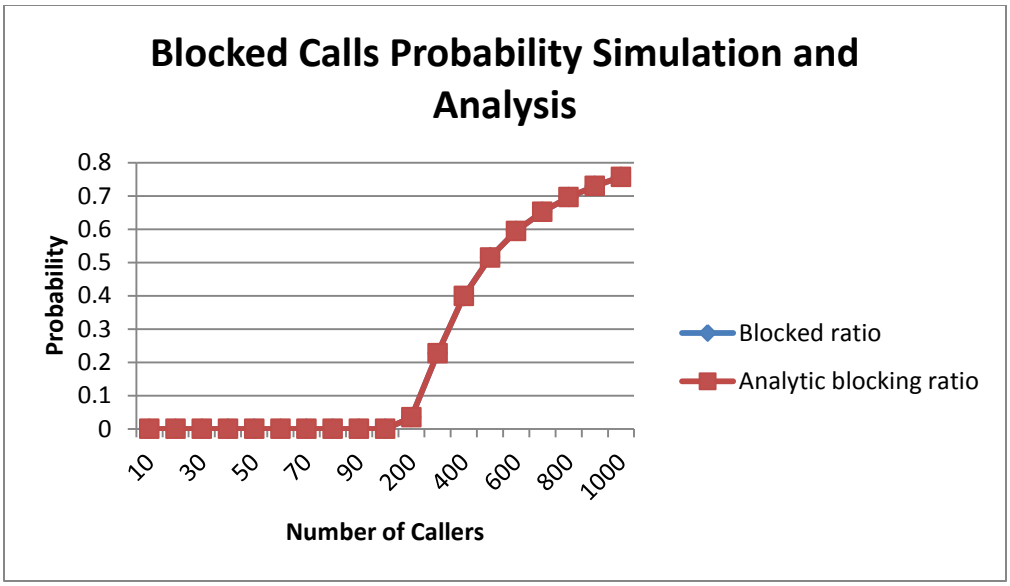

Figure 6: Comparing call blocking probability in analysis and simulation

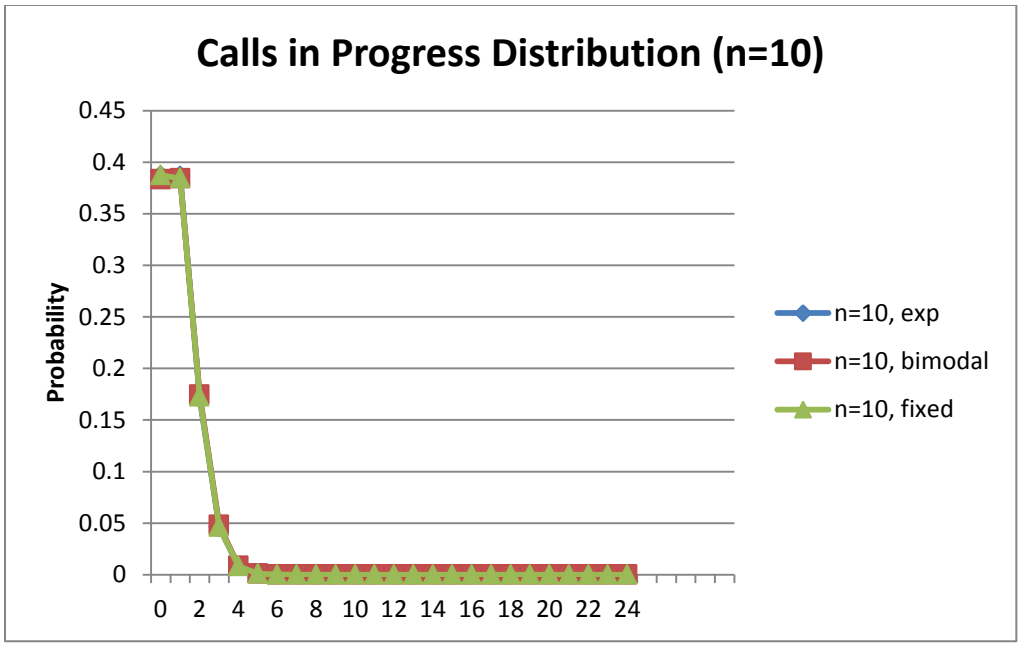

Figure 7: Active callers distribution for 10 callers

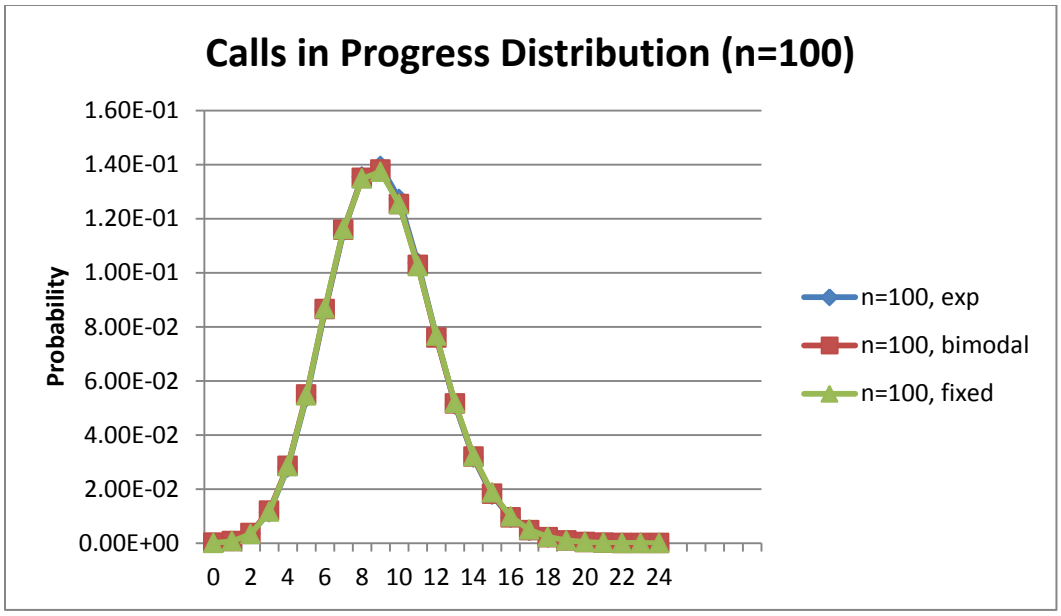

Figure 8: Active callers distribution for 100 callers

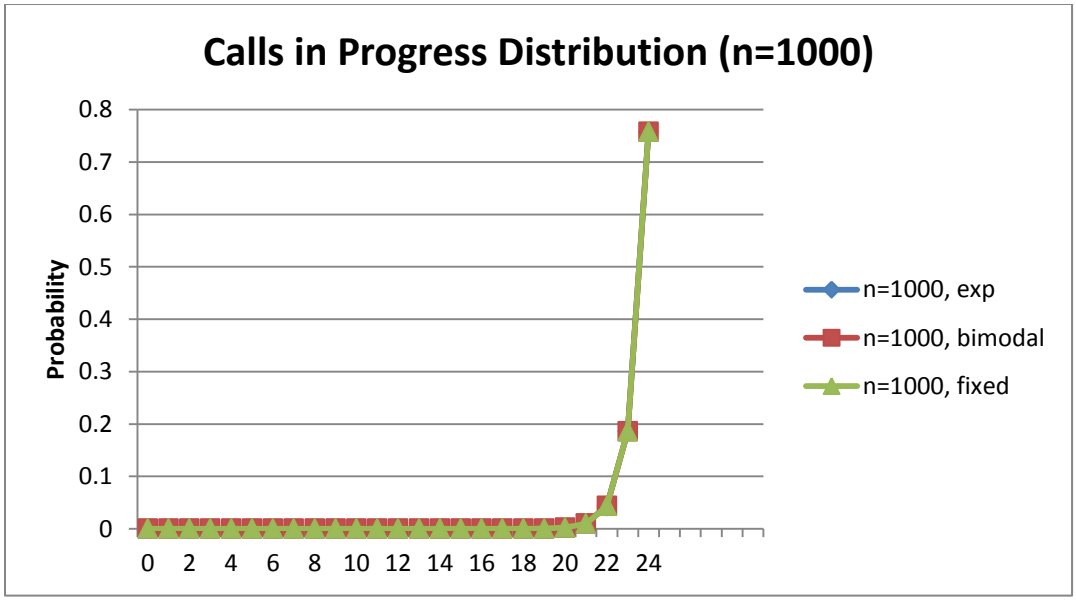

Figure 9: Active callers distribution for 1000 callers

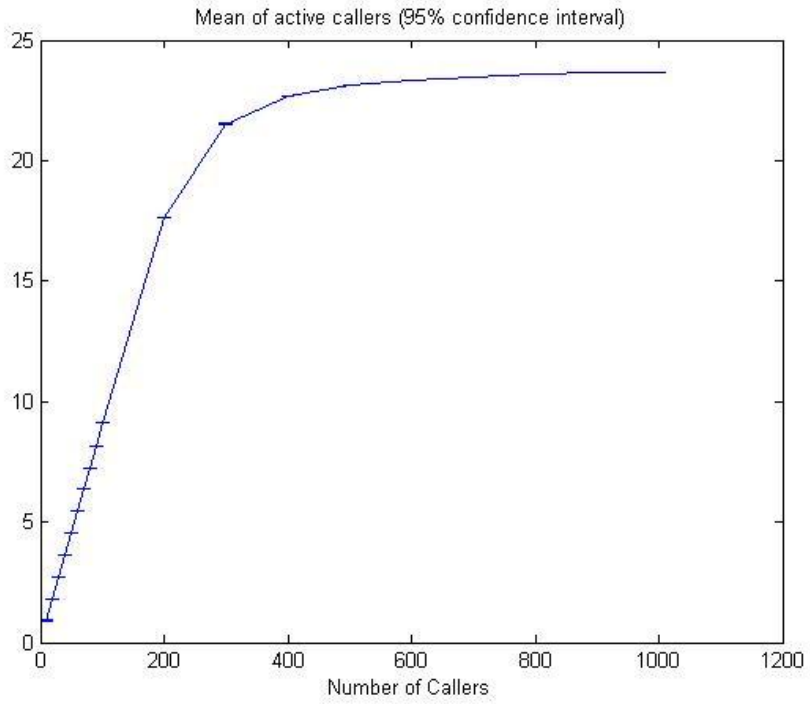

Figure 10: mean of number of calls in progress with 95% confidence interval for different number of callers

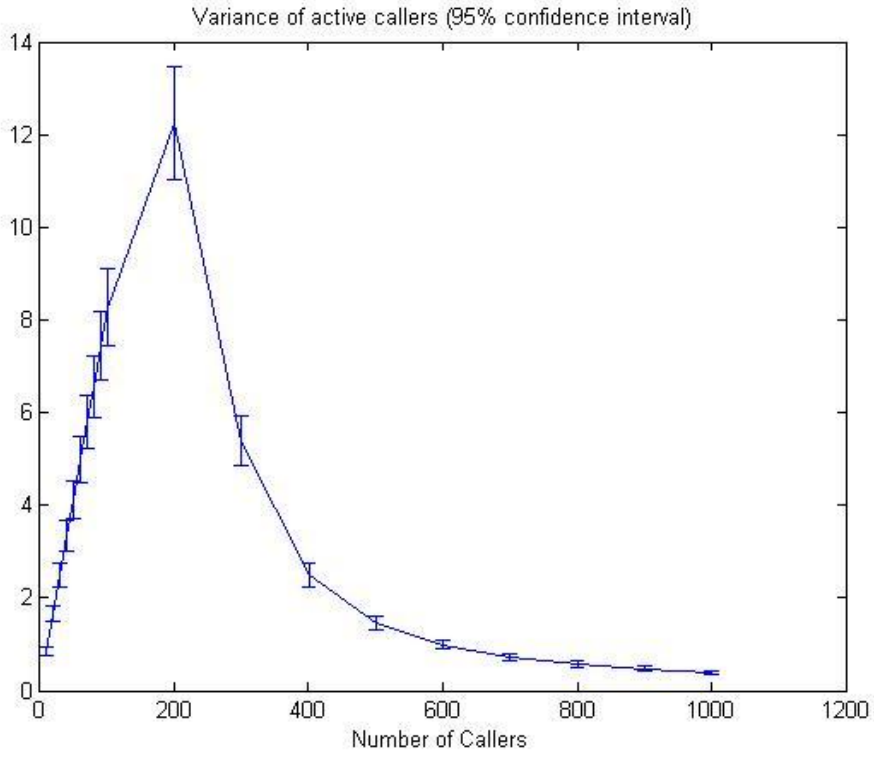

Figure 11: variance of number of calls in progress with 95% confidence interval for different number of callers

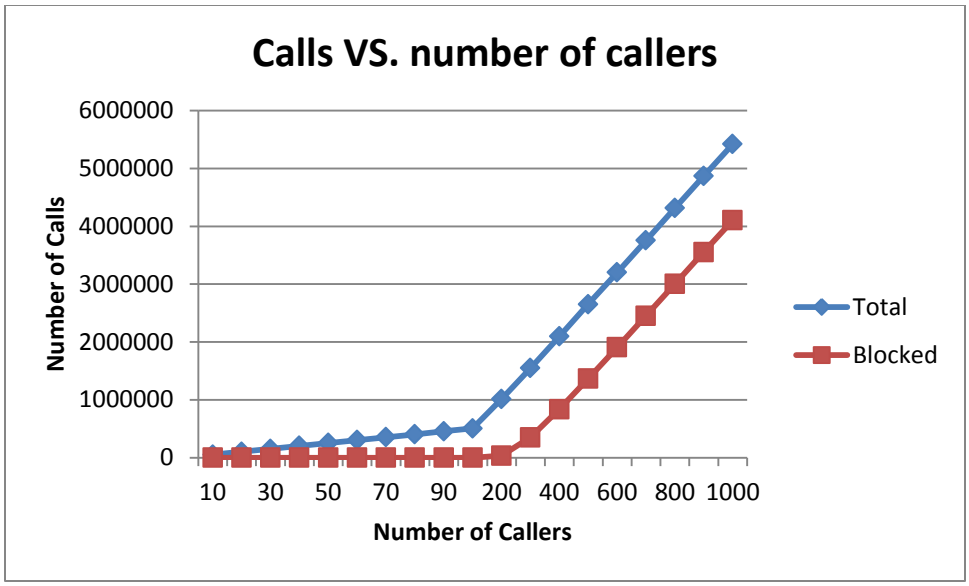

Figure 12: Blocked and Total number of calls vs. number of callers

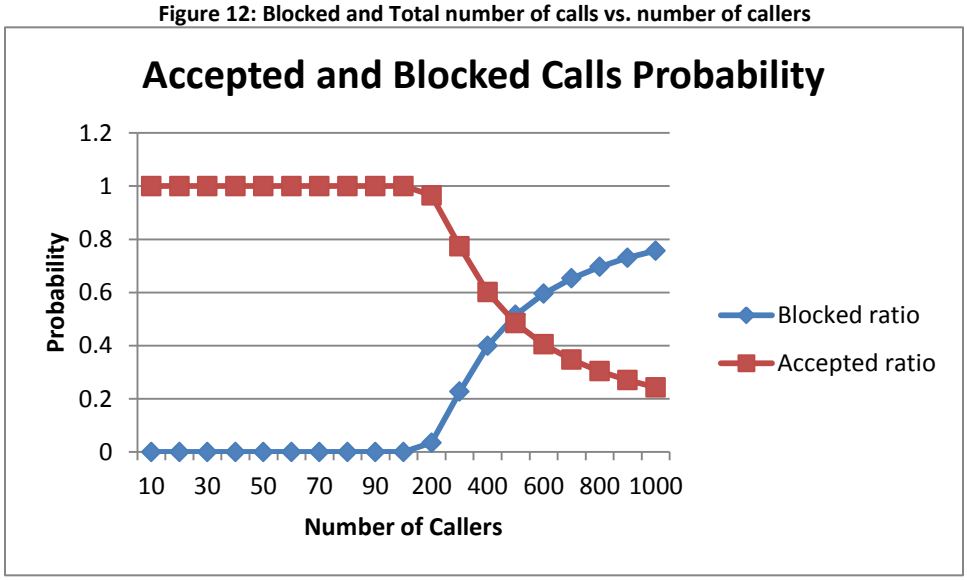

Figure 13: Accepted and Blocked calls vs number of callers

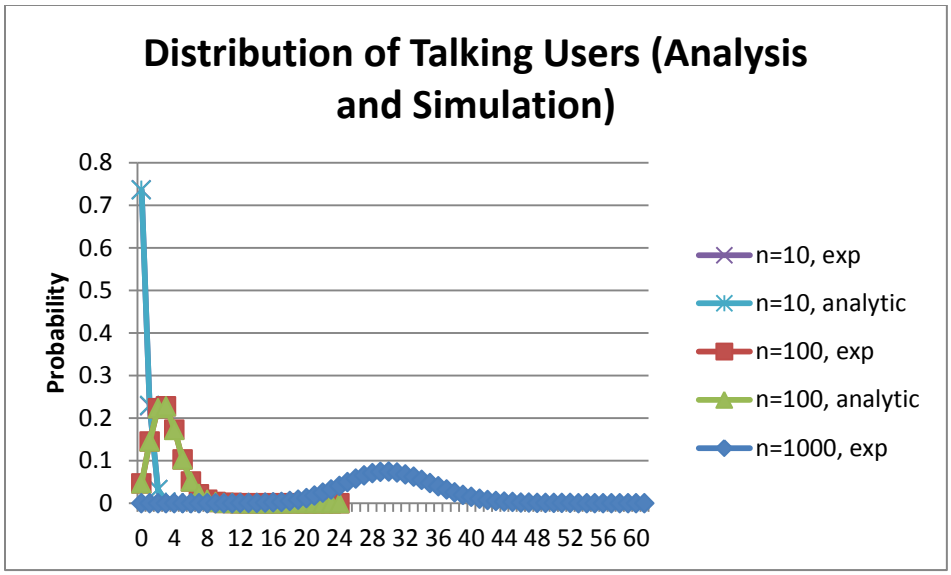

Figure 14: Comparing distribution of talking callers in analysis and simulation for exponential call duration

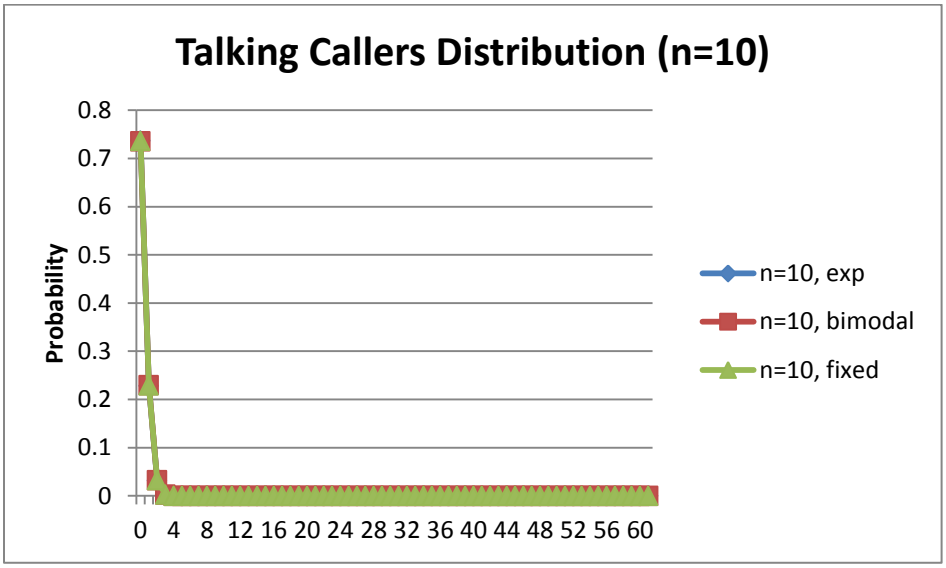

Figure 15: Distribution of talking users in simulation with n=10

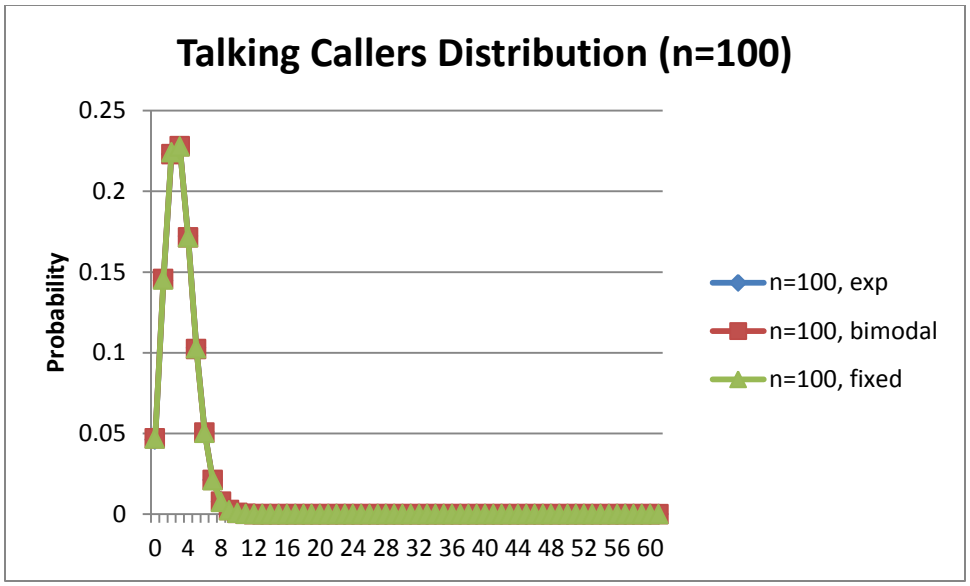

Figure 16: Distribution of talking users in simulation with n=100

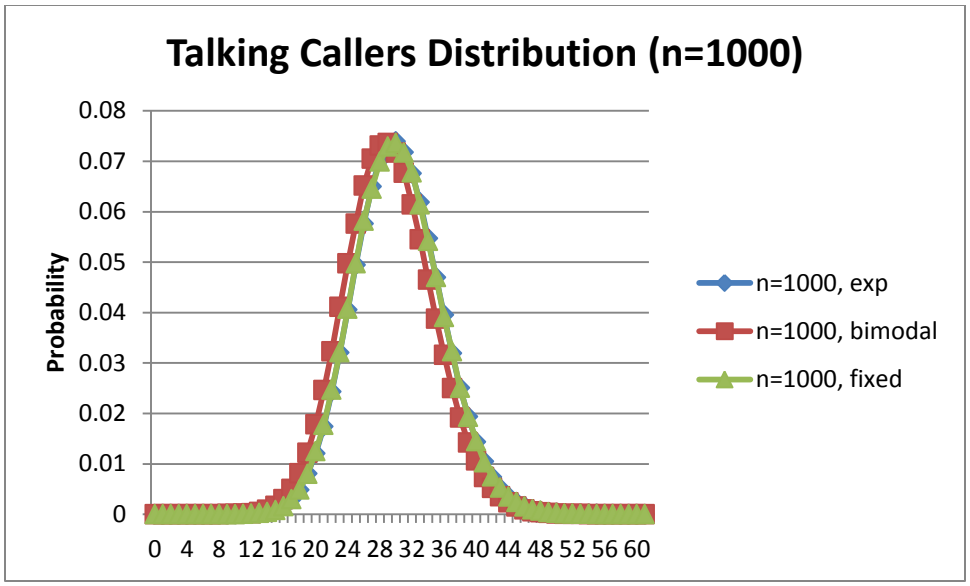

Figure 17: Distribution of talking users in simulation with n=1000

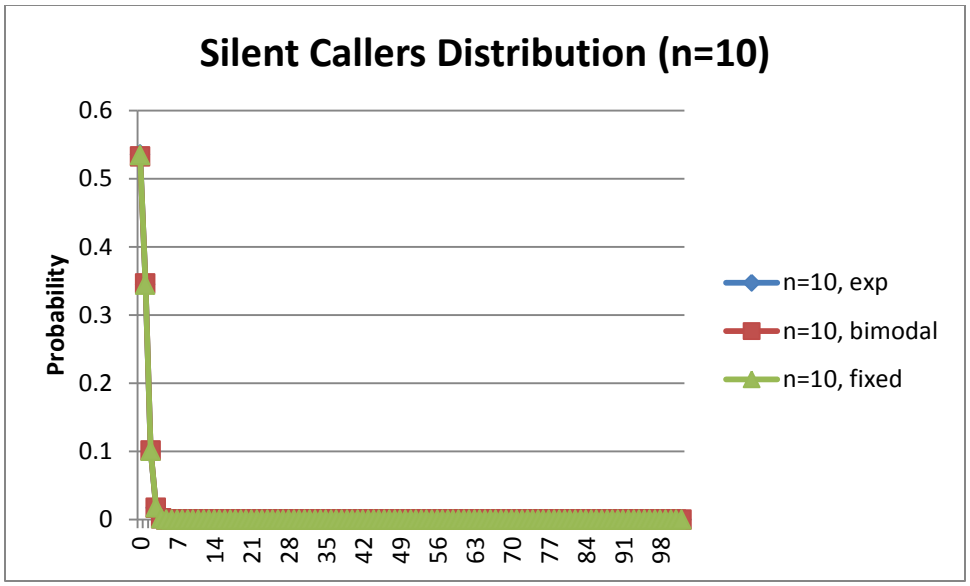

Figure 18 : Distribution of silent users in simulation with n=10

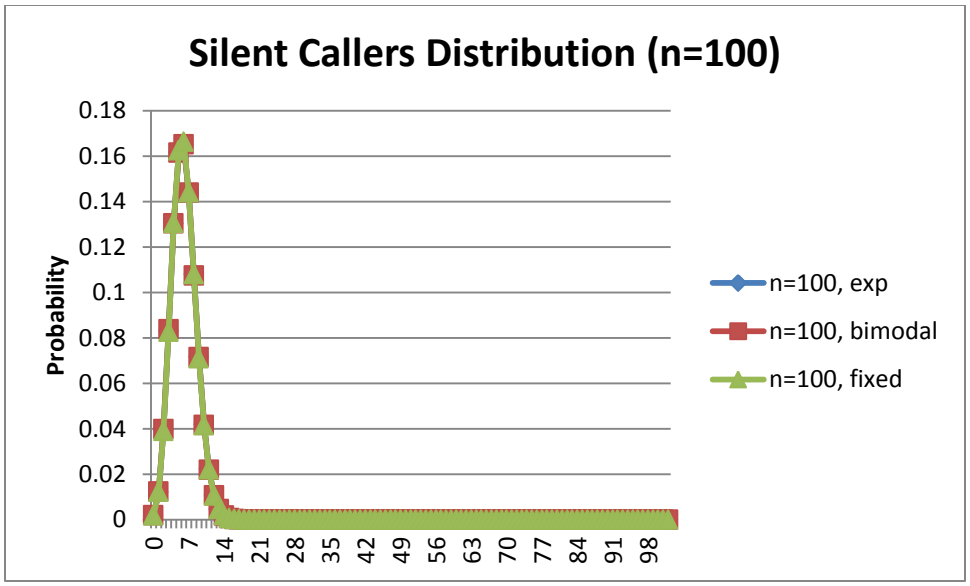

Figure 19: : Distribution of silent users in simulation with n=100

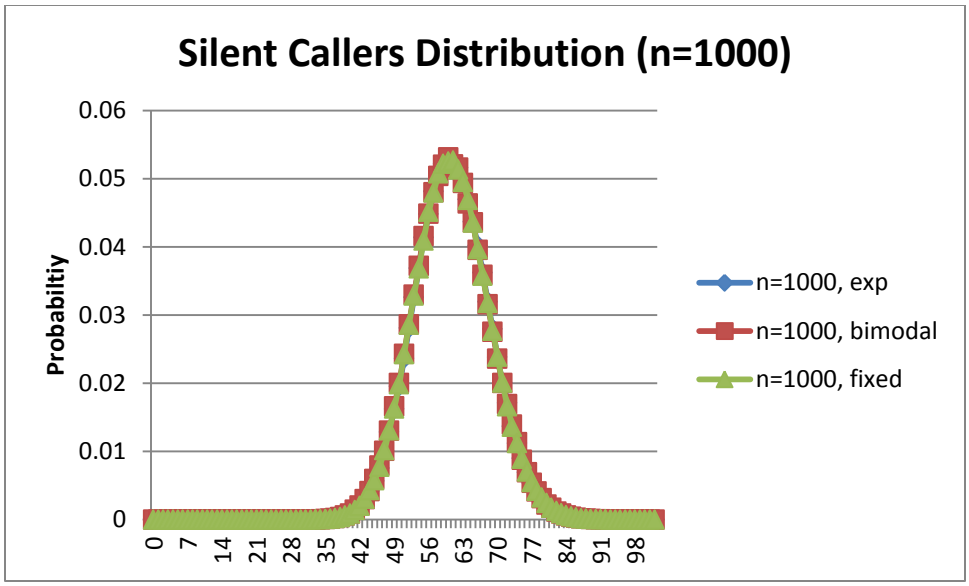

Figure 20: : Distribution of silent users in simulation with n=1000

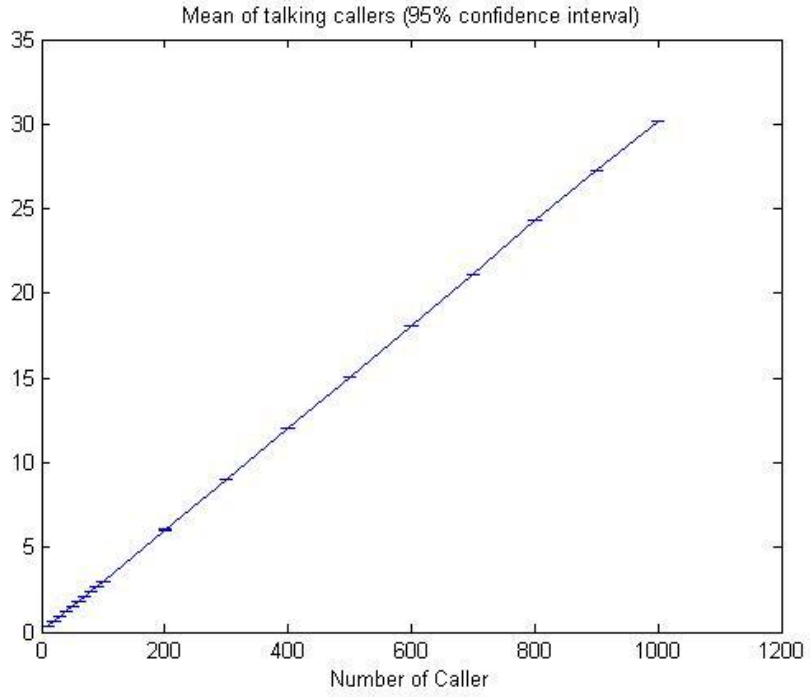

Figure 21: mean of number of talking calls with 95% confidence interval for different number of callers for fixed call duration

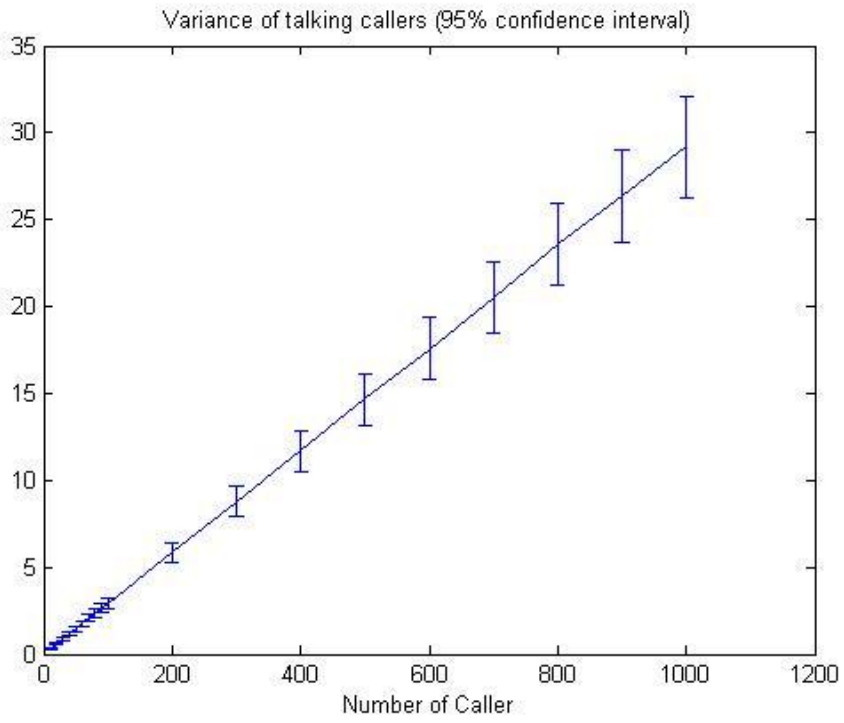

Figure 22: variance of number of talking calls with 95% confidence interval for different number of callers for fixed call duration

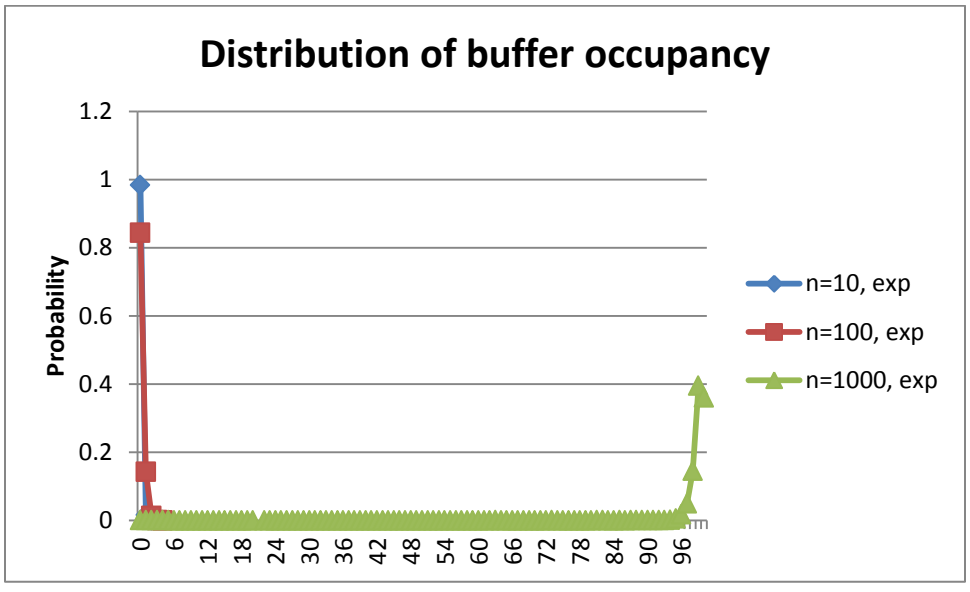

Figure 23: Distribution of buffer occupancy in level 3

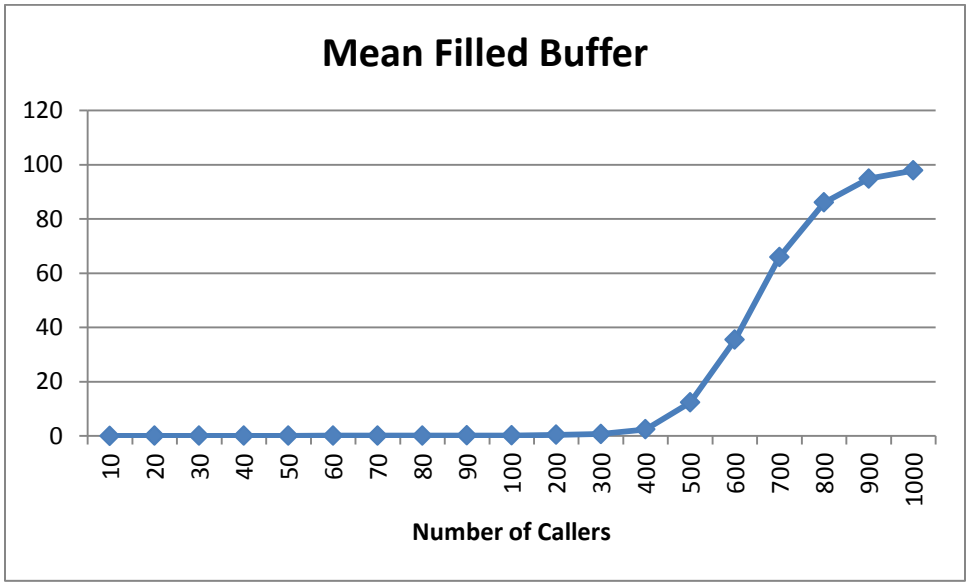

Figure 24: Mean of buffer occupancy for different number of callers

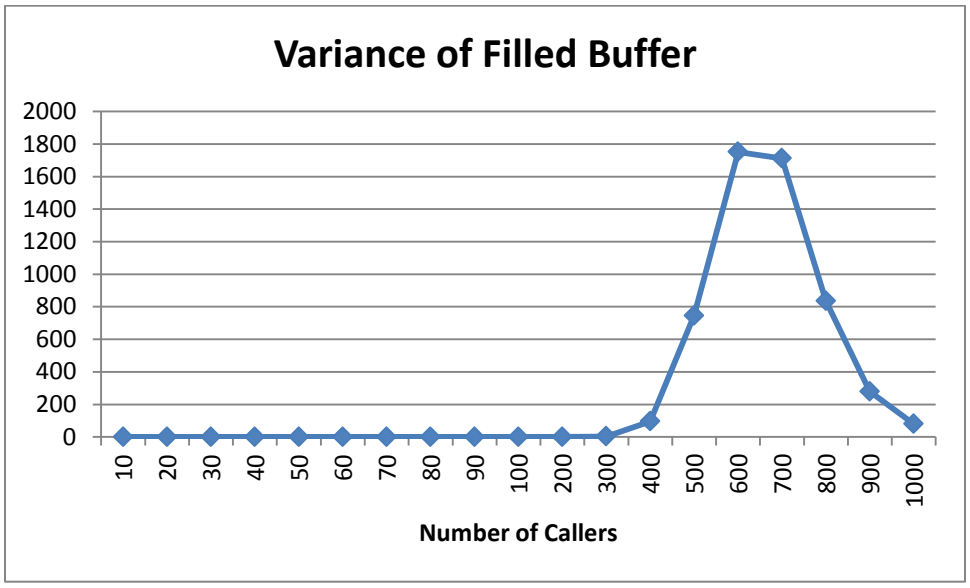

Figure 25: Variance of buffer occupancy for different number of callers

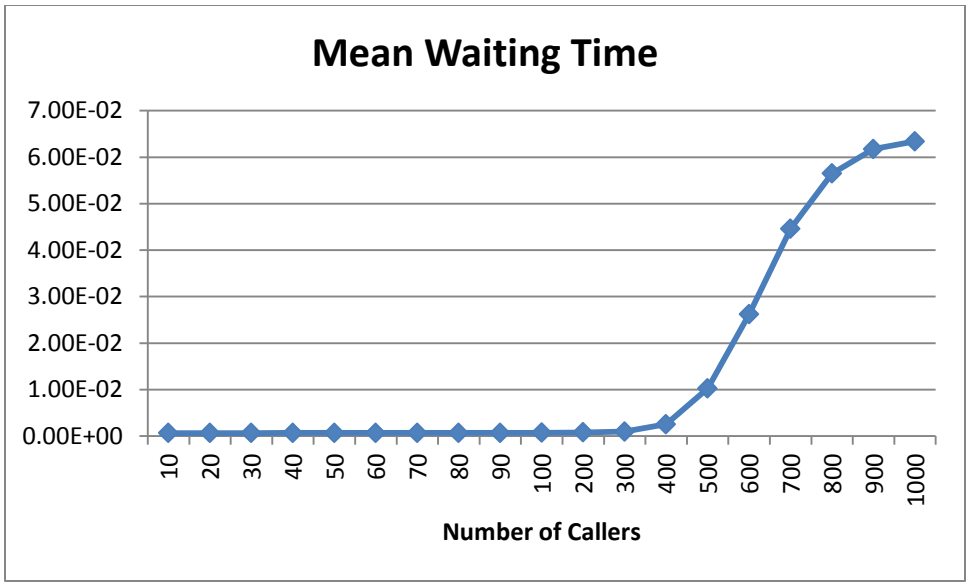

Figure 26: Mean of waiting time of packets in buffer in level 3

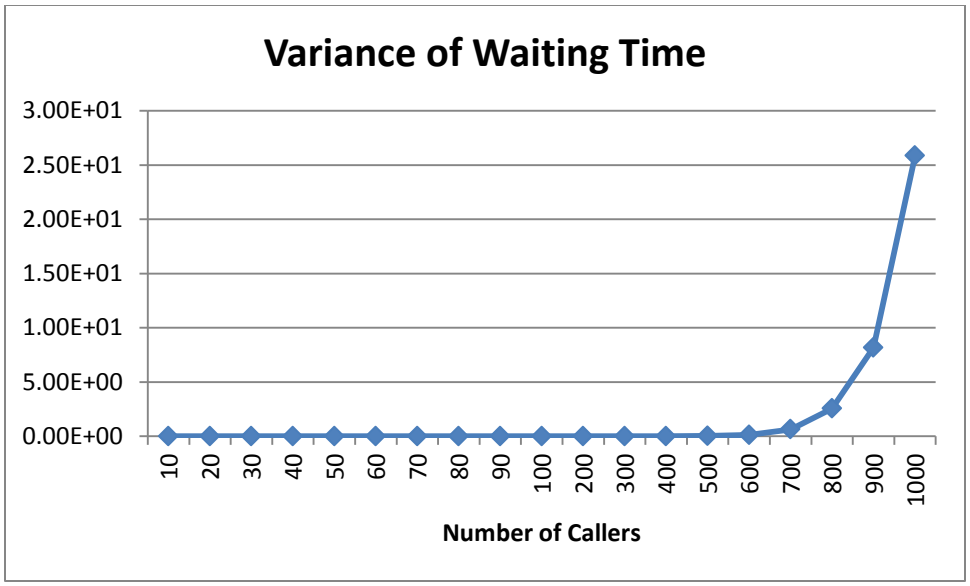

Figure 27: Variance of waiting time of packets in buffer in level 3

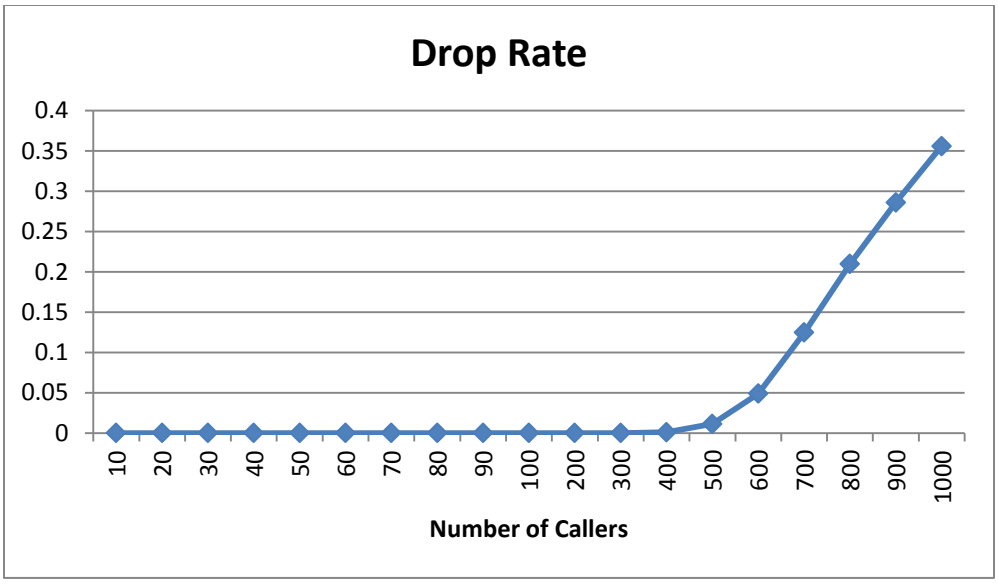

Figure 28: Drop rate of packets in level 3 with different number of callers

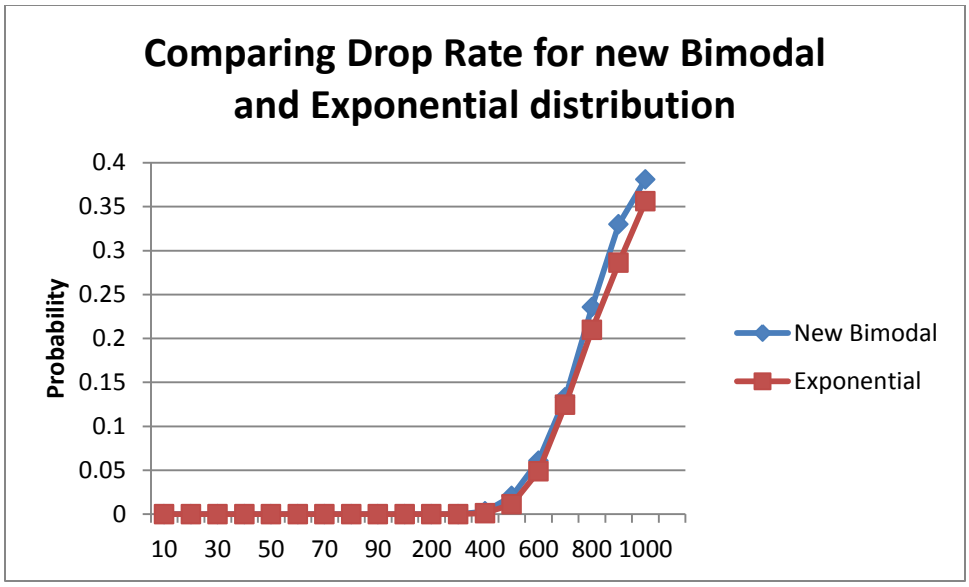

Figure 29: Comparing Drop Rate for new BiModal and Exponential distribution

6 Appendix 2: Program Structure Figures

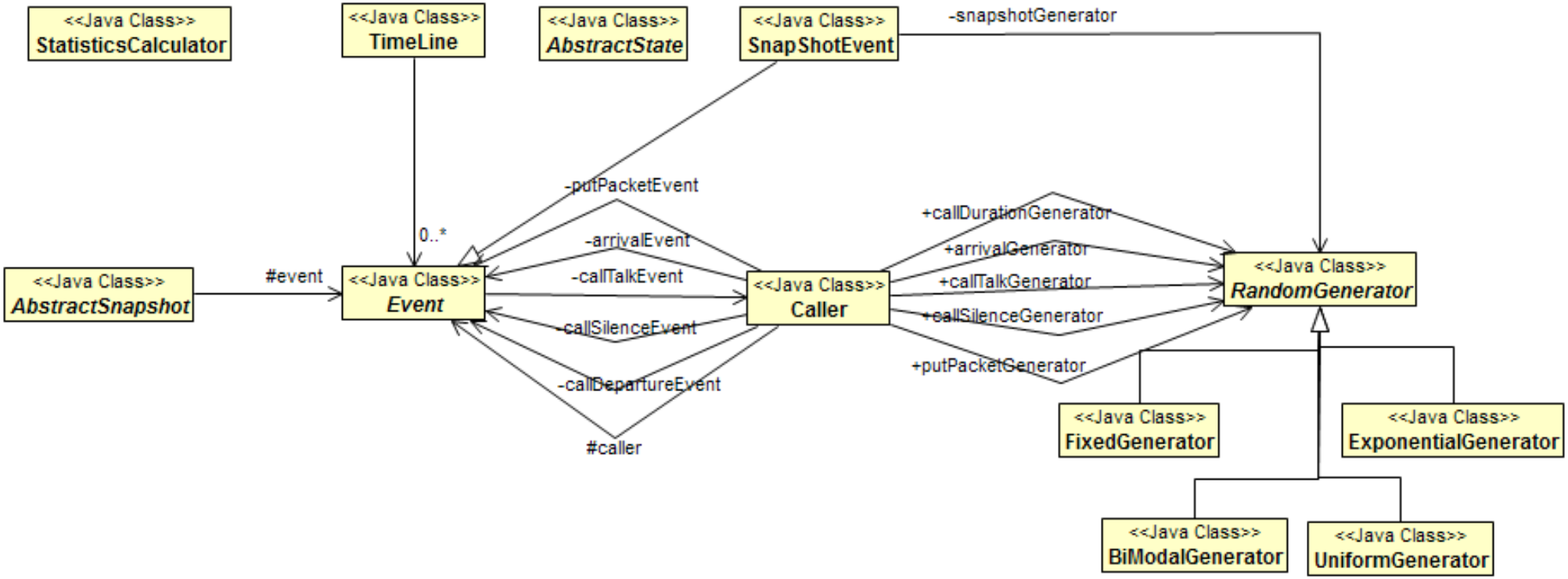

Figure 30: An abstract view of common structure in all levels

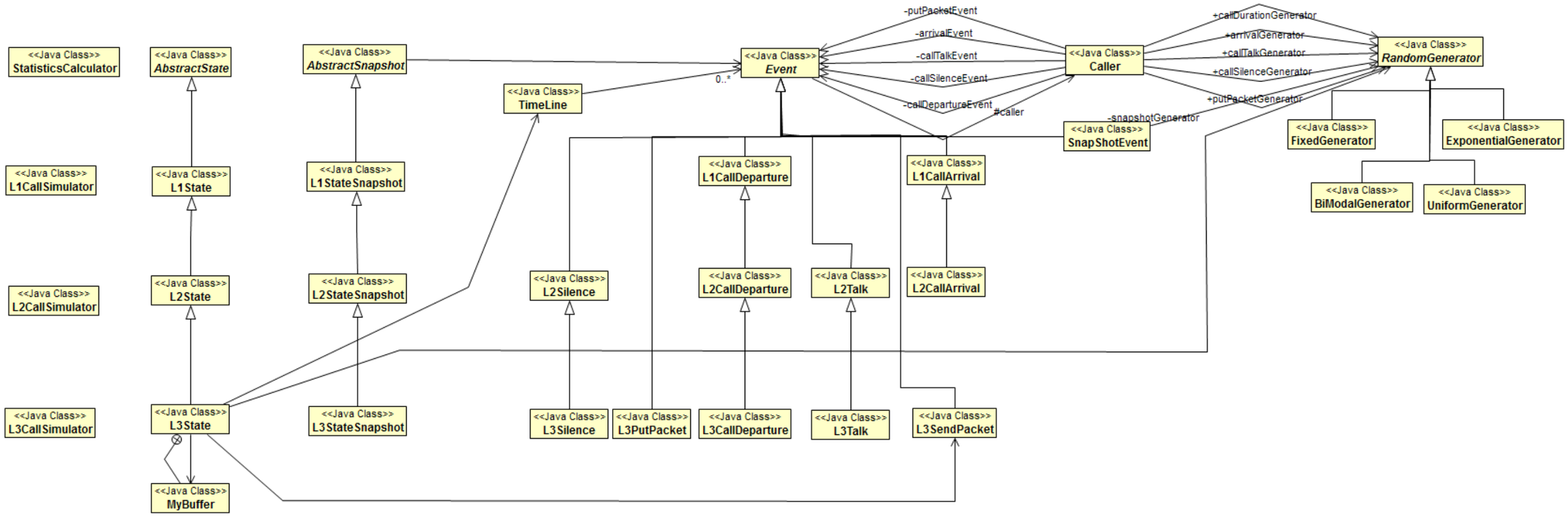

Figure 31: The abstract view of all classes in the system

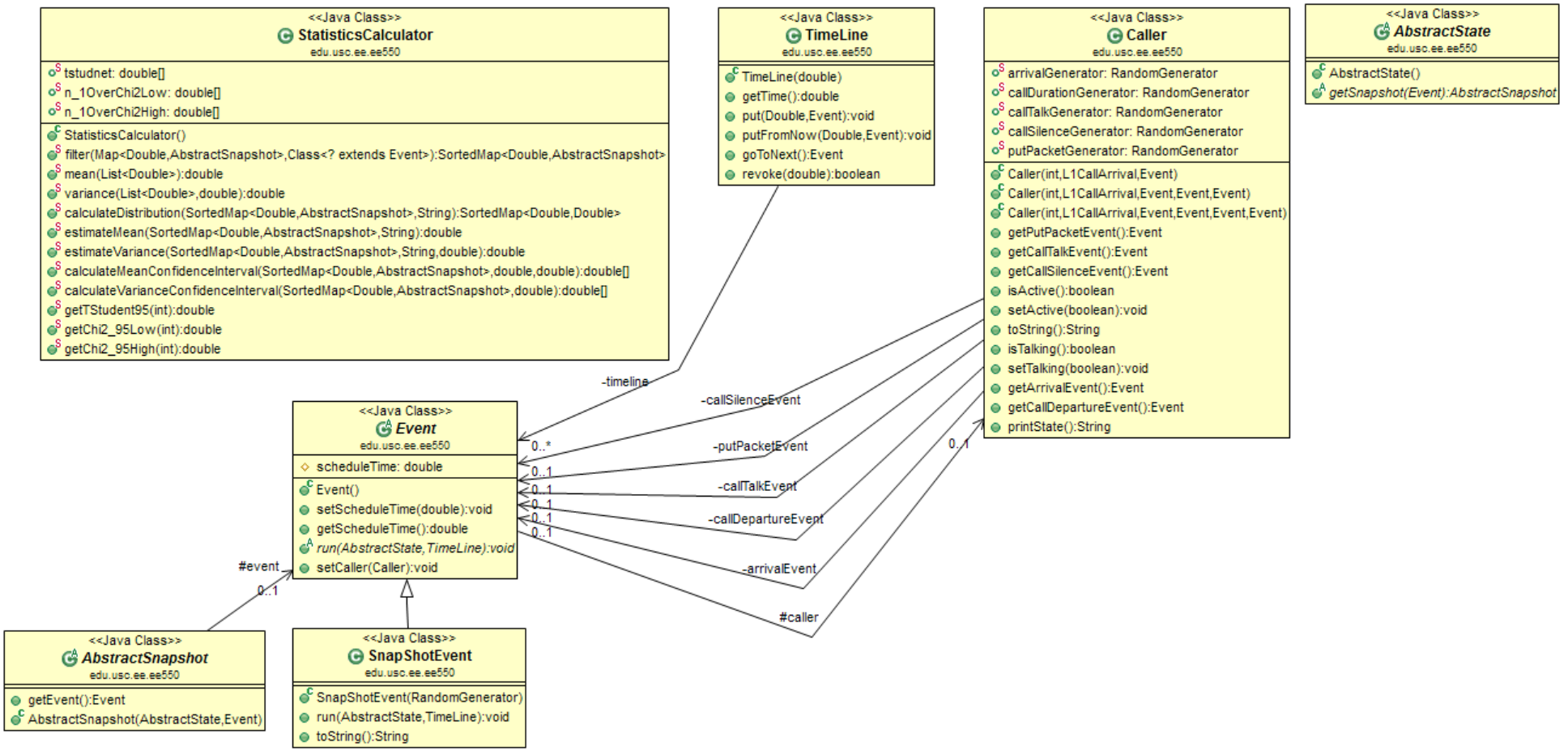

Figure 32: Detailed class diagram of common structures

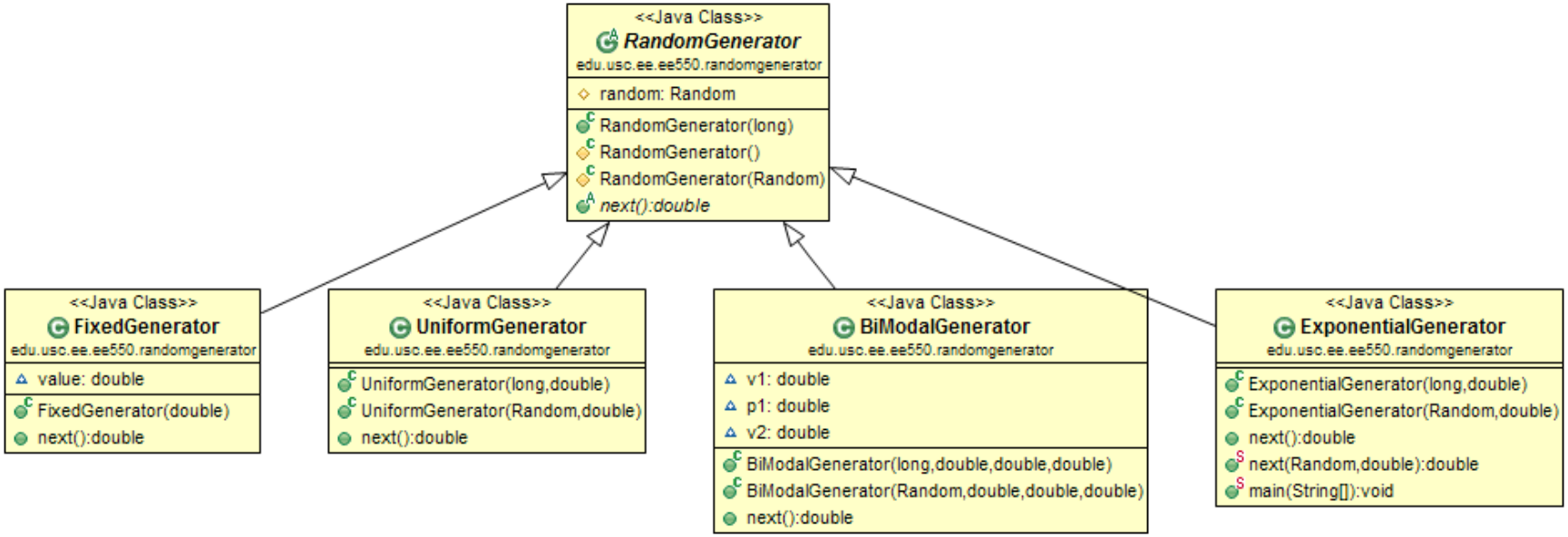

Figure 33: Detailed class diagram of random generator package

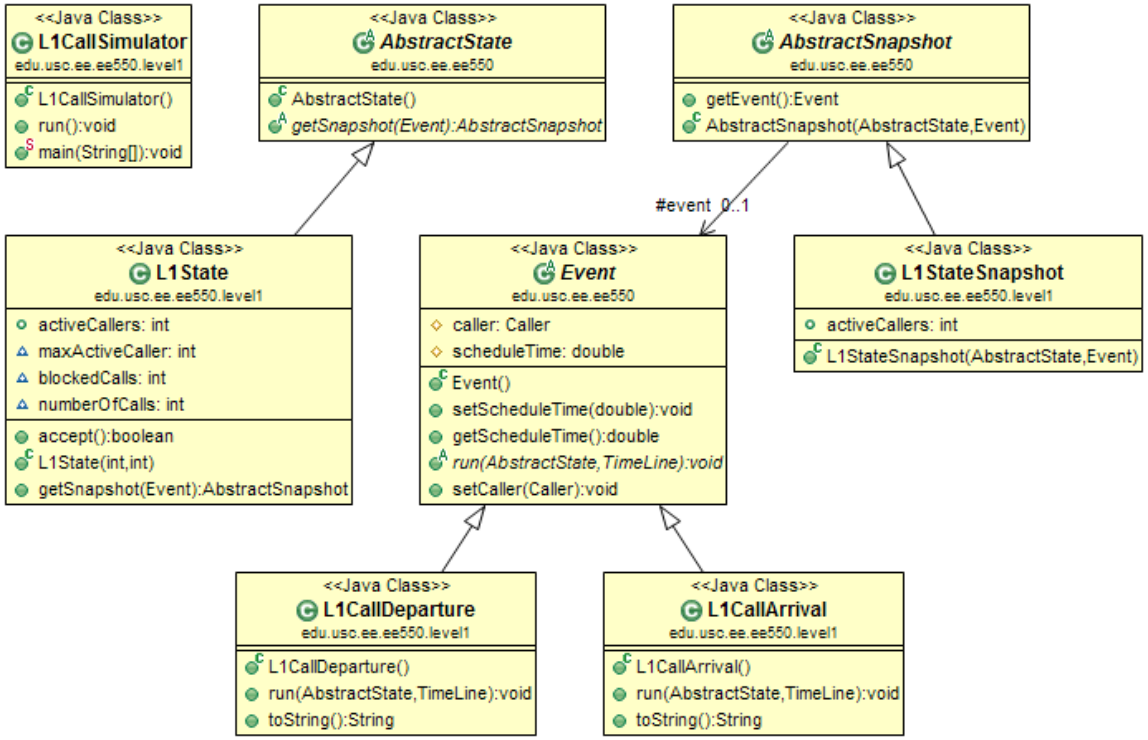

Figure 34: Detailed class diagram of level 1 structures

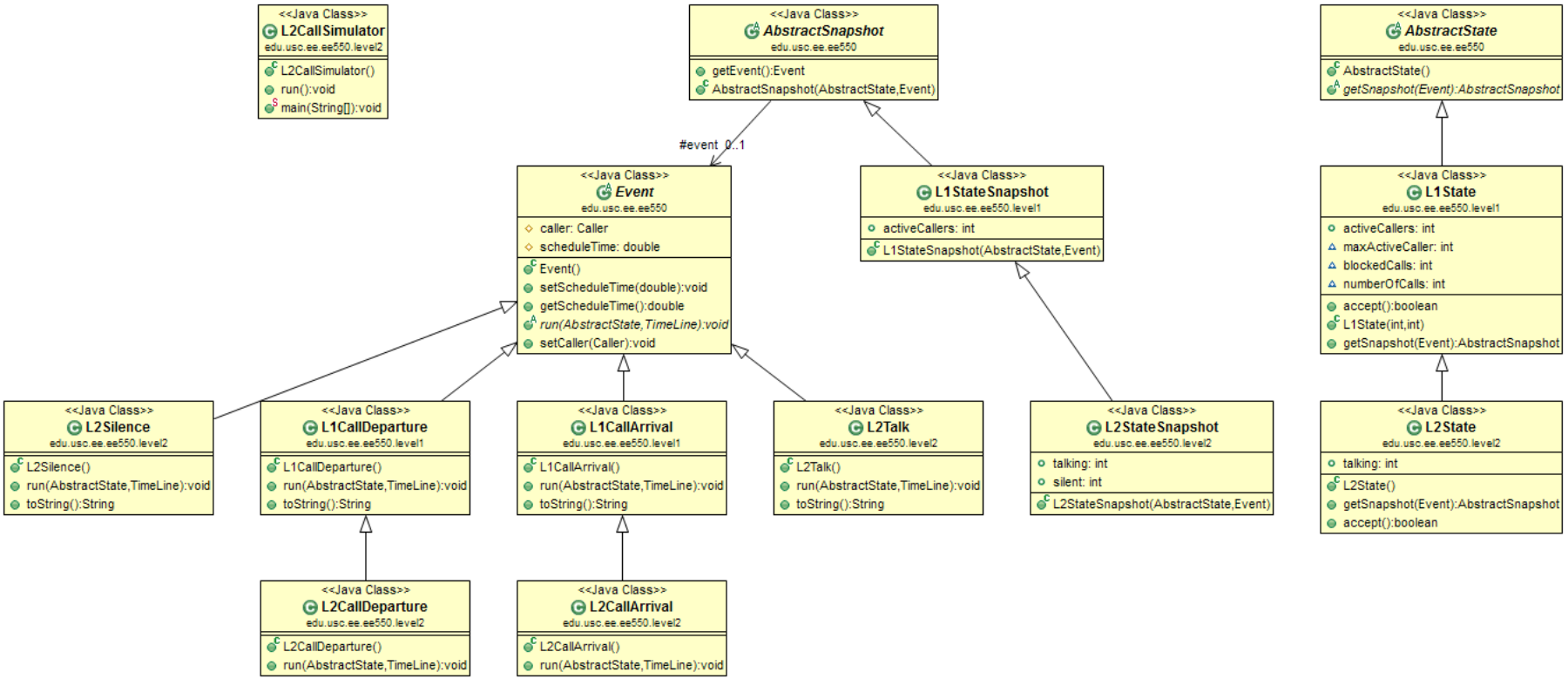

Figure 35: Detailed class diagram of level 2 structures

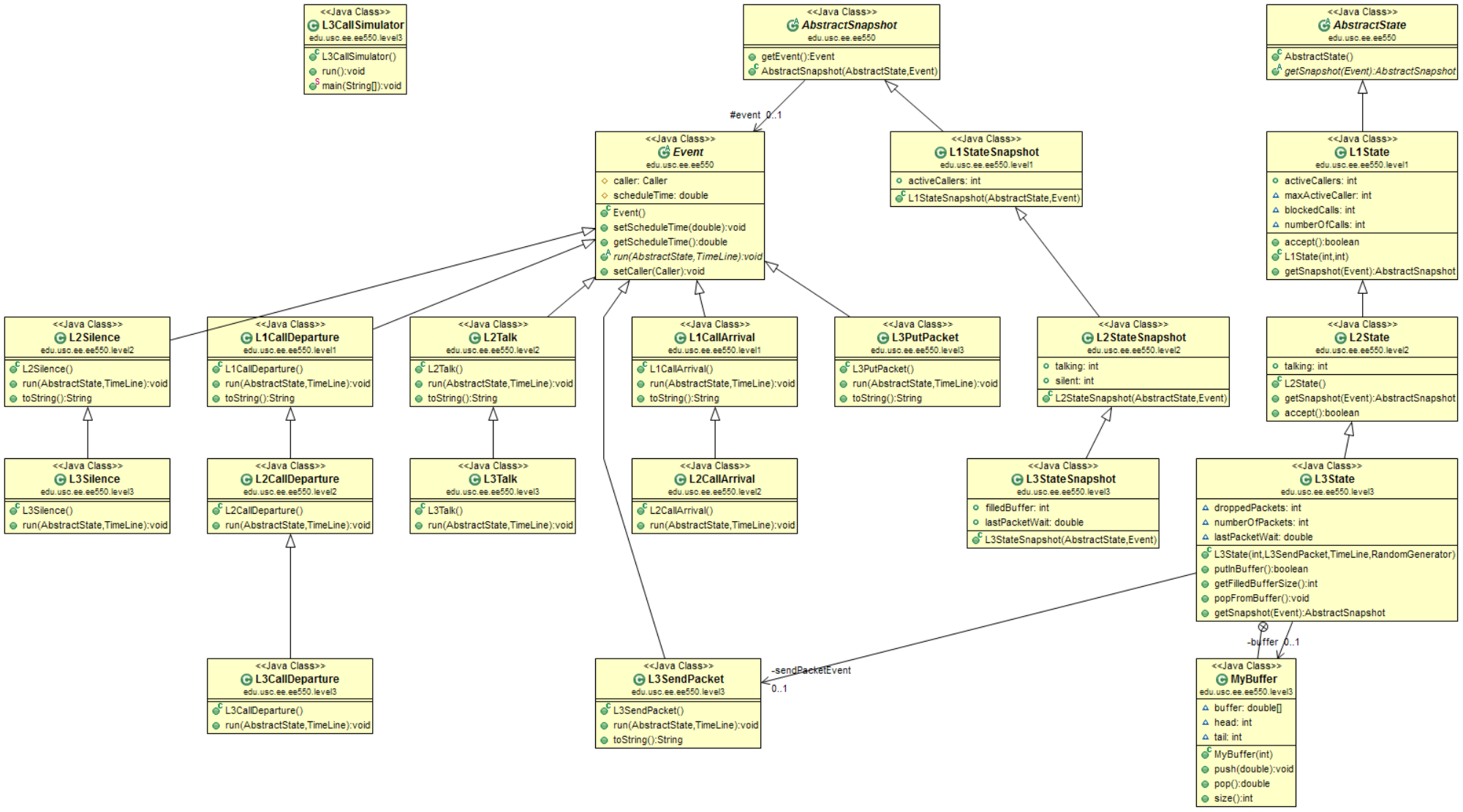

Figure 36: Detailed class diagram of level 3 structures

7 Appendix 3: Codes

7.1 MATLAB Codes

7.1.1 Level 1

I implemented the choice function as for large n, the usual usage of factorial hits the machine constraint

```
function [prod]=choice(n,k)
    prod=1;
    for i=1:k
        prod=prod*(n-i+1)/(k-i+1);
    end
end
```

This code computes the probability distribution of states in level 1. Then it computes the blocking probability in y.

```
ns=[30,40,50,60,70,80,90,100,200,300,400,500,600,700,800,900,1000];
K=24
l=1/(60*30);
mu=1/(3*60);
clear y;
for j=1:length(ns),
    x=[];
    N=ns(j);
    sum=0;
    for i = 0:K,
        sum=sum+choice(N,i)*(l/mu).^i;
    end
    for i =0:K,
        x(i+1)=choice(N,i)*(l/mu).^i/sum;
    end
    sum2=0;
    for f=0:K,
        sum2=sum2+x(f+1)*(N-f)*l;
    end,
    y(j)=x(K+1)*(N-K)*l/sum2;
end
```

7.1.2 Level 2

To solve the linear system in level 2 first we need to fill the matrices. The following code prepares a and b matrices for linear solver.

```
N=100;
K=100;
mu=1/(3*60);
l=1/(60*30);
alpha=1;
beta=2;
s=getindex(K,K,K);
a=zeros(s,s);
for i=0:K,
    for j=i:K,
        r=zeros(1,s);
        index=getindex(i,j,K);
        toleft=0;
        toright=0;
        totop=0;
        todown=0;
        totopleft=0;

        fromleft=0;
        fromright=0;
        fromtop=0;
        fromdown=0;
        fromdownright=0;

        if i>0
            fromtop=(j-(i-1))*alpha;
            totop=i*beta;
            r(getindex(i-1,j,K))=fromtop;

            totopleft=i*mu;
        end
        if j>i
            fromleft=(N-j+1)*l;
            toleft=(j-i)*mu;
            r(getindex(i,j-1,K))=fromleft;
```

```
            fromdown=(i+1)*beta;
            todown=(j-i)*alpha;
            r(getindex(i+1,j,K))=fromdown;
        end
        if  j<K
            toright=(N-j)*l;
            fromright=(j+1-i)*mu;
            r(getindex(i,j+1,K))=fromright;

            fromdownright=(i+1)*mu;
            r(getindex(i+1,j+1,K))=fromdownright;
        end

        output=toleft+toright+totop+todown+totopleft;
        r(index)=-output;

        a(index,:)=r;
    end
end

a=[a;ones(1,size(a,2))];
b=zeros(size(a,1)-1,1);
b=[b;1];
```

Use the following code to solve the system

```
p=linsolve(a,b);
```

To compute the distribution of talking users user the following code:

```
talking=zeros(K+1,1);
for i=0:K,
    sum=0;
    for j=i:K,
        index=getindex(i,j,K);
        sum=sum+p(index);
    end
    talking(i+1)=sum;
```

```
end
```

You can also user the following code to find the distribution of calls in progress using the level 2 model:

```
active=zeros(K+1,1);
for j=0:K,
    sum=0;
    for i=0:j,
        index=getindex(i,j,K);
        sum=sum+p(index);
    end
    active(i+1)=sum;
end
```

7.2 Java Codes

There are so much java codes for this project that I could prefer to give the source files as a zip file besides this document. Here is only a guide to the directory structure:

- edu.usc.ee.ee550:
  - level1
    - L1CallArrival.java
    - L1CallDeparture.java
    - L1CallSimulator.java
    - L1NCompareCallSimulator.java
    - L1State.java
    - L1StateSnapshot.java
  - level2
    - L2CallArrival.java
    - L2CallDeparture.java
    - L2CallSimulator.java
    - L2NCompareCallSimulator.java
    - L2Silence.java
    - L2State.java
    - L2StateSnapshot.java
    - L2Talk.java

- level3
  - L3CallDeparture.java
  - L3CallSimulator.java
  - L3NCompareCallSimulator.java
  - L3PutPacket.java
  - L3SendPacket.java
  - L3Silence.java
  - L3State.java
  - L3StateSnapshot.java
  - L3Talk.java
- randomgenerator
  - BiModalGenerator.java
  - ExponentialGenerator.java
  - FixedGenerator
  - RandomGenerator.java
  - UniformGenerator.java
- AbstractSnapshot.java
- AbstractState.java
- Caller.java
- Even.java
- SnapShotEvent.java
- StatisticsCalculator.java
- TimeLine.java