Department of Electrical Engineering,

Syed Babar Ali School of Science and Engineering,

Lahore University of Management Sciences, Lahore, Pakistan

Senior Year Design Project (Sproj 1) Report

"Call Center Simulation Using NS-3"

Submitted by

Ammarah Azmat Bajwa	**Roll No: 18100243**
Hira Tariq	**Roll No: 18100215**
Zarmeen Khan	**Roll No: 18100273**
Muhammad Shamaas	**Roll No: 18100217**

Under the supervision of

Dr. Tariq Mahmood Jadoon

Designation: Associate Professor

Email: jadoon@lums.edu.pk

Signatures of Approval

Project Advisor:

Project Co – Advisor(s) (if any):

Contents

1 Problem Statement

This project attempts to realistically simulate call centers in NS3 by incorporating call abandonment, heavy call traffic, call drops and skill based routing.

Call center modeling is a growing research field. They are an example of queuing systems. Queuing models help estimate ways to optimize staffing and maximize performance in a call center. We used NS-3, which is a discrete event network simulator, for our modeling. Calls arrive, wait in a virtual queue and then they are serviced via a routing mechanism. The most common model used at large by both practitioners and academics is the Erlang C model. The model however works under many assumptions mainly: calls arrival follows a Poisson distribution with a known average rate, callers wait infinitely without abandoning the call. Erlang X: an extension of Erlang C is more complex and realistic as it incorporates call abandonment, heavy call traffic; call drops and skill based routing. We will attempt to improve Erlang X with additive variables, which makes it close to a realistic call center. We will verify it by comparing simulation results against the performance records of a real call center.

2 Literature Review

In order to model our Call Center we will need to understand the “Standard Queuing Model” which is based on two assumptions. First, all the agents are assumed to be "homogenous" in their services i.e. having same set of skills and speed to answer the call. Secondly, calls routing is done in such a way that the call is forwarded to the agent who is idle for the longest interval of time. This oversimplification affects the performance of the call center by increasing the average waiting time of the queue. In his paper [13], the author Thomas R. Robbins has highlighted the issues in the oversimplified queuing model and has suggested another routing scheme for the call center environments. Queuing system is implemented in all the call centers. If the agent is busy then the call is placed in the queue and waits there until the agent gets free. Mostly call centers are observed as having a "standard queuing model" which is an oversimplification of the practical call center [12], [5]. However, the performance can be enhanced by introducing some changes in the routing scheme. Agent requires particular skills in handling a certain call efficiently, which depend on their experience as well. More the experience of the agent in handling calls lesser will be the time he will take to service the call, thus the average waiting time in the queue will decrease [13]. Robbins has suggested "experience-based routing" during the peak hours that is when the call arrivals are high then the calls should be delivered to the experienced agents so that they can efficiently manage the load by servicing the calls in lesser time. However, when the arrival rate of calls is small then, according to Robbins, calls should be routed to the non-experienced agents so that they can learn how to service the calls and could gain some experience. This routing scheme not only improves the "system performance" and the "customer service" but also enhance the learning of the newly hired agents [4].

Thomas R. Robbins et al. in their paper [13] have compared the queuing models that are widely used in the call center analysis. First is the Erlang C model, many assumptions in the model simplifies it. This model assumes that the agents are “statistically identical” all have the same speed and ability to answer the call. In addition, the calls, which arrive according to the Poisson distribution, are having a known average rate of arrival [8]. Moreover, it assumes that the size of the queue is infinite and calls can wait for any large amount of time inside the queue without abandoning it. It means that the average waiting time could be large enough which is quite unrealistic and can have a drastic impact on the customer service and call center operations. However, in Erlang A model, the extension of the Erlang C model, abandonments are considered. The callers have some finite patience to wait in the queue and if it turns out the caller abandons the queue [7]. “Quality-driven and the Quality and efficiency-driven (QED) regimes”, where the call centers are capable enough to handle all the calls and there is no abandonment, are considered in this paper and the performance metrics for both Erlang A and Erlang C in QED regimes are being compared. Erlang A model though give better results but it does not depict the accurate staffing requirement in the cases when the arrival rates have significant uncertainties [4].

In such cases, Erlang C is used in making the staffing decisions. However, in case of the "Efficiency-driven regime" when the arrival rate is large and all agents are occupied in servicing the calls here the Erlang C model simply is collapsed because there are significant abandonments. Erlang A model, which allows the abandonments, is used to predict the performance metrics in this kind of regime [5].

Call Center is analytically modeled using continuos time Markov Chain that we will discuss briefly now.

Continuous Time Markov Chain

A Call Center can be modeled using a Markov Chain as shown in Figure 1 where the number of the state represents the number of callers waiting in the Queue. It has the unique property of memorylessness where we can make predictions about future of the process based on the present state only. The call arrivals and service are assumed to have a Poisson distribution that expresses the probability of a given number of events occurring in a fixed interval of time if these events occur with a known constant rate and independently of the time since the last event [8].

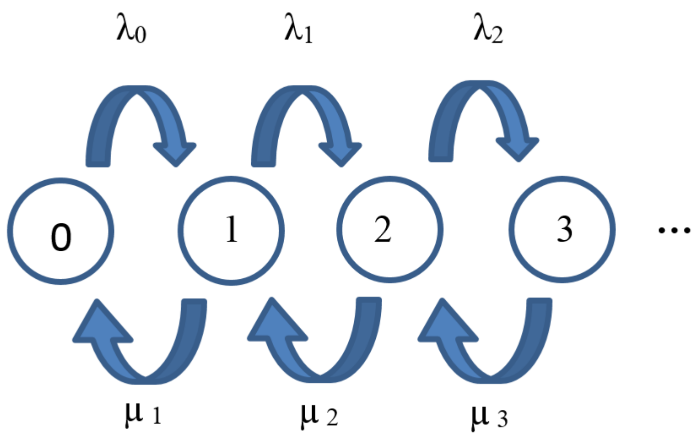


FIGURE 1: CONTINUOS TIME MARKOV CHAIN

The Probability of being in state j at time n = π_j (n).

Transition Probability P (X_{n+1}=j | X_n=i) = $p_{i,j}$ (n)

$q_{i,j}(n) = \lim_{\Delta t \to 0} \left(\frac{pij(\Delta t)}{\Delta t} \right)$

Arrival rate of Calls in state j = $\lambda_j = q_{j,j+1}$ (n)

Service Rate of Calls in state j = $\mu_j = q_{j,j-1}$ (n)

$q_{j,j} = -(\lambda_j + \mu_j)$

Time Evolution of $\pi_j(t) = \frac{d\,\pi\, j\,(t)}{dt} = q_{j,j}(t)\,\pi_j(t) + \sum_{k\neq j} q_{k,j}(t)\,\pi_k(t)$

For Stationary State $\pi_j(t) = \pi^*_{j,}\ \frac{d\,\pi\, j\,(t)}{dt} = 0$ and $\sum_j \pi\, j = 1$

The solution of these j differential equations gives the following probability distribution:

Stationary State Probability of state j = $\pi^*_j = \pi^*_0 \prod_{i=0}^{j-1} \frac{\lambda\, i}{\mu\, i+1}$

If $\lambda j = \lambda, \mu j = 0$ for all $j, \pi\, j\,(t) = \frac{(\lambda t)^j}{j!} e^{-\lambda t}$ which is the Poisson Distribution Formula

Now we will discuss three different Markov Chains; M/M/1 Queue, M/M/∞ Queue and the M/M/1/K Queue.

M/M/1 Queue

An M/M/1 Markov Chain has Markovian Arrivals and Service with single Queue Server as shown in Figure 2. The Call Arrival Rate = λ and the Call Service Rate = μ. Queue can grow infinitely hence; infinite states are possible [8].

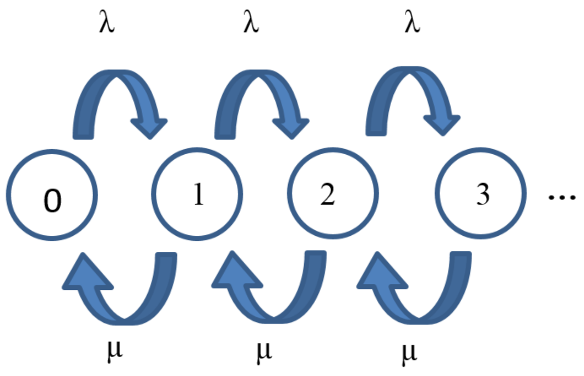


FIGURE 2: M/M/1 QUEUE MARKOV CHAIN

Stationary State Probability of state j = $\pi^*_j = \left(1 - \left(\frac{\lambda,}{\mu}\right)\right) \left(\frac{\lambda,}{\mu}\right)^j$

M/M/∞ Queue

An M/M/∞ Queue has Markovian Arrivals and Service with separate Queue Server for each Caller as shown in Figure 3. The Call Arrival Rate = λ and the Call Service Rate = nμ where n represents the number of the state. Queue can grow infinitely hence; infinite states are possible [8].

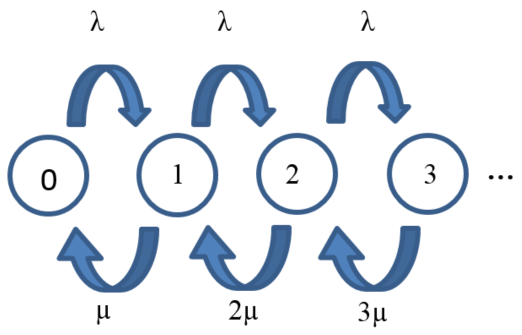


FIGURE 3: M/M/∞ MARKOV CHAIN

Stationary State Probability of state j = $\pi^*_j = \frac{e^{-\frac{\lambda}{\mu}}(\frac{\lambda}{\mu})^j}{j!}$

M/M/1/K Queue

An M/M/1/K Markov Queue has Markovian Arrivals and Service with single Queue Server as shown in Figure 4. The Maximum size of Queue = k, Call Arrival Rate = λ and Call Service Rate = μ.

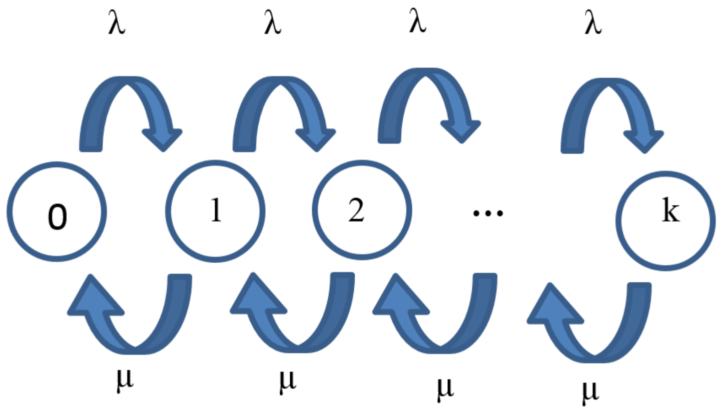


FIGURE 4: M/M/1/K MARKOV CHAIN

Stationary State Probability of state j = $\pi^*_j = \frac{1-(\frac{\lambda}{\mu})}{1-(\frac{\lambda}{\mu})^{k+1}}(\frac{\lambda}{\mu})^j$ where $j \le k$

The Real Call Center has Abandonments, Call Drops, Dynamic Agents Scheduling, Time varying call arrival rates, heterogeneous agents, Heavy Traffic and Call Retrials as shown in Figure 5, which require much more detailed modeling and calculations.

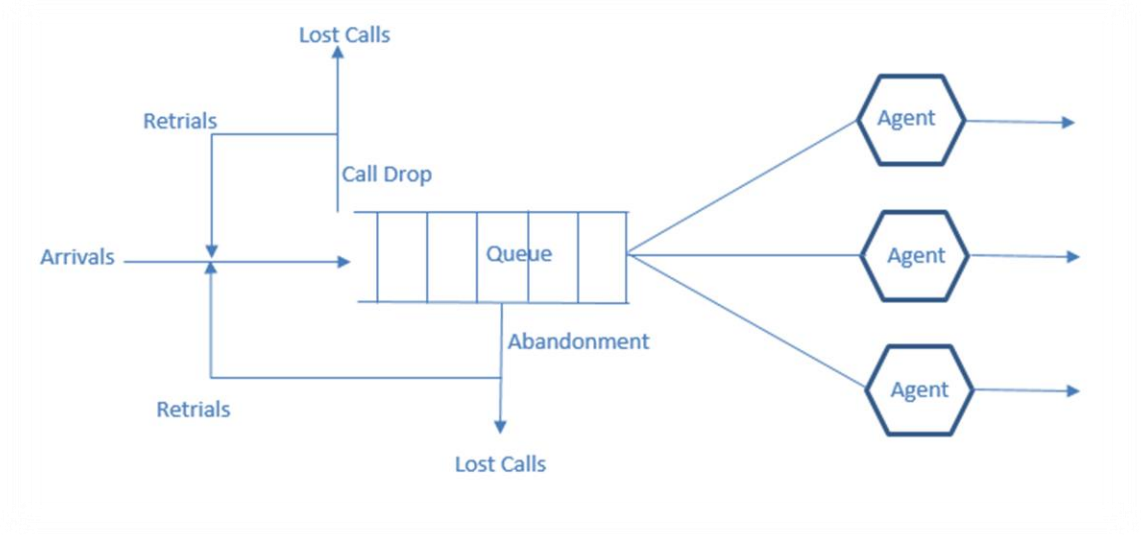


FIGURE 5: REAL CALL CENTER MODEL

In this section, we discussed the various models used for the analytical analysis of the Call Center Queuing Model. Now we will discuss the Simulation Results of our Proposed Model whose foundation is the M/M/N+M/K Queue and which was built to improve the existing Call Center Models. Our goals were:

1. Merge M/M/N and M/M/1/K models to include call drops and multiple Queue Servers.
2. Include Agent Skills and Caller Patience for call abandonments.
3. Incorporate randomness for propagation delays.
4. Perform experiments with simulator and compare results with the industrial models of Erlang C and Erlang X.
5. Iteratively perform simulation optimization to realize actual call center model.

3 Simulation Results of Proposed Design

We are implementing a Simulation of an Optimized Call Centre Model on NS3 Software, which is a discrete event simulator [6]. We have implemented packet switching in our simulation. This has been done in the software, by allocating a caller node, distributor and Agent Node. As the following is a representation of a more realistic model, so we have implemented Erlang X. It in cooperates queue abandonment of callers, unpredictable call traffic, call drops and skill based routing. Fastest Server First, Longest Idle Server First and Experience Based routings each in different circumstances have been used to service calls in our model [14].

The Call Centre structure comprises of three fundamental entities; the Caller Node, the Agent Node and the Distributor Node. Having built a structure for the Agents, we have further differentiated them based on their skills, no of calls serviced and operator number in our model. The callers have been differentiated based on call times, patience tag, waiting time in the queue and the skill type they require. While a common distributor node (buffer) has also been made which Enqueue arriving packets and Dequeue packets to the agent node.

Erlang C is a simple multi-server queuing system; calls arrive according to a Poisson process at an average rate. We verified Erlang C Formula with experimental results for mean waiting time of callers in Queue using the Ger Koole calculator [10], [11]. Our goal is to verify Erlang X with additive variables, which makes it close to a realistic call center through empirical analysis involving comparison of performance with real call center records [9].

Generally, abandonment rates cannot be estimated directly using the Erlang C model because the model assumes no abandonment occurs. Our model will assume that each caller has a finite willingness to wait, and will abandon the queue or hang up if his or her wait time exceeds his or her patience. We plan to achieve this by assigning each caller a patience counter, which is decremented after each second. Erlang X is modeled with limited available number of telephone lines thus calls will be dropped if the waiting queue is full. In addition to finite queue, we route our distributor in such a way that it can pick a blocking packet from anywhere in the queue and send it for service to prevent queue blockage [10], [11]. Furthermore, the agents in our model will also have multiple skills. There are two categories: Specialists, having only one skill and Generalists, having expertise in more than one skill [7]. We conclude by verifying experimentally that even with complex variables, we can achieve optimization in Erlang X just as that in Erlang C.

3.1 Initial System Diagram

The Initial Call Centre Diagram is shown in Figure 6 and Queue Diagram is shown in Figure 7.

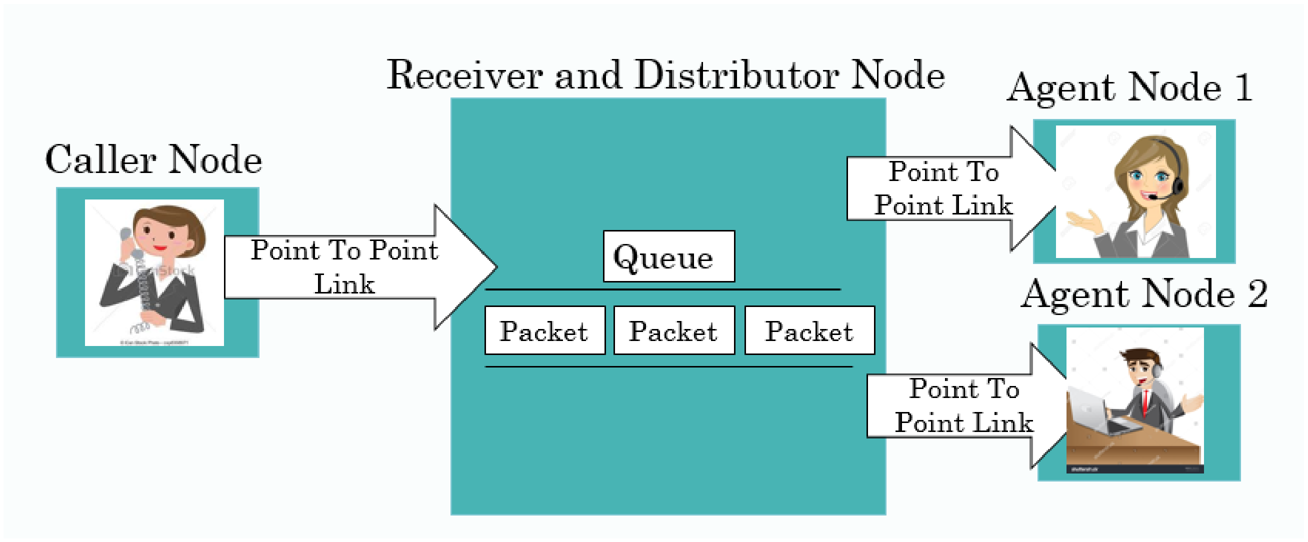


FIGURE 6: CALL CENTRE

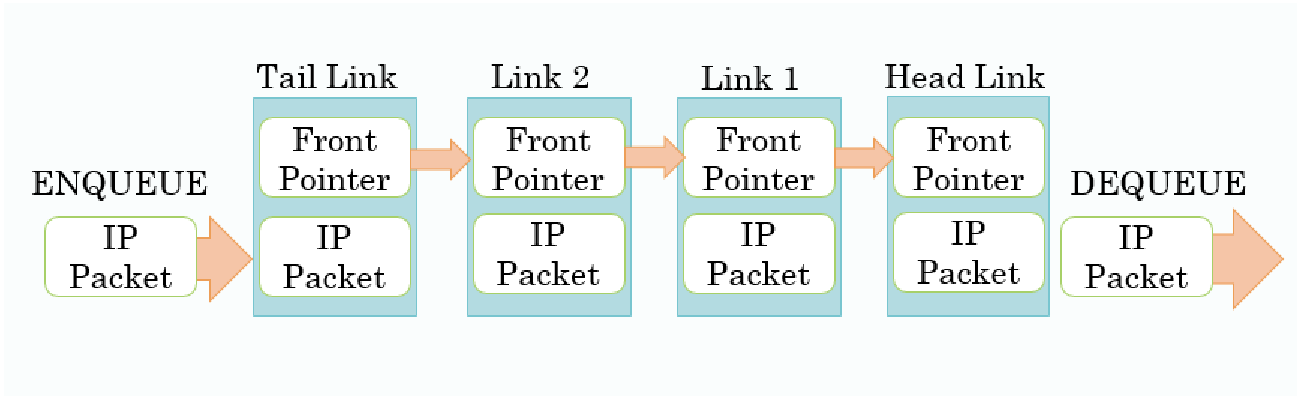


FIGURE 7: QUEUE

Now we will discuss the Initial System Model called Erlang C, which is very widely used in the industry for Call Centre Modeling.

3.2 Initial System Model: Erlang C

Erlang C is a single Queue server model that is modeled using M/M/N Queue [13] which was discussed earlier. Markovian processes model the Calls arrivals and service. There are N Queue servers or Agents, each with a service rate of μ hence the effective service rate is Nμ. All Agents are homogenous and perfectly efficient at all time. The calls arrive at a fixed rate of λ. The Calls have a mean length and Callers do not hang up. The Callers can wait infinitely for service in the queue i.e. they never abandon the queue. Hence, the queue can build up infinitely and no calls are lost before service.

3.2a Simulation Methodology

We built the Queue using a chain of Link Structures in NS3. Each Link has a front and back pointer that point to proceeding and preceding Link. The Link is a container for the Call Packet, which is stored as a Packet Pointer. The length of the packet denotes the length of the call hence the Link also stores the Packet Size as a Data element.

```
Struct Link
{
        Link *     Back;
        Link *     Front;
        Uint32_t   Data;
        Ptr<Packet> pkt;
}
```

A Caller Node keeps generating IP packets and sends them to the distributor node using a Point-to-Point Link. The Packet length follows a fixed distribution and has a fixed mean value. This is implemented in NS3 by a function that generates a random variable, makes an IP packet of that size and then sends it using a socket. The function then makes a self-call by scheduling the next call after a time interval given by a random number following a negative exponential distribution.

The Distributor Node receives the IP packets and keeps adding them to the tail of the Queue. This is done in NS3 by calling an interrupt to Enqueue function whenever a packet arrives at the socket. It also checks every second if any Agents are idle and distributes the Packets from the Queue to them. In NS3, it schedules a call to SendPacket function every second, which does the routing of calls using Point to Point Links to Agents.

The Agent Node receive the IP packet from the Distributor Node and becomes busy for the duration of the call. Hence, it changes the Boolean idle flag to zero. While it is busy, it cannot receive any more packets. It schedules a call to ServiceComplete function after a period equal to the length of the call after which it changes its status to idle again.

Erlang C Formula

The Erlang C Formula [7] is given below and the plot is shown in Figure 8:

λ = Number of calls per unit time

b = Average Call Duration

a = λ * b = Load in Erlang units

s = Number of Agents

For s > a,

Average Waiting Time in Queue = $\frac{C\,(s\,,\ a) * b}{(s - a)}$

Where Probability of Enqueue = $C(\,s,\ a) = \frac{\frac{s\,.\,a^{s}}{s!\,.\,(s-a)}}{\sum_{i=0}^{s-1}\frac{a^{i}}{i!}+\frac{s.a^{s}}{s!\,.\,(s-a)}}$

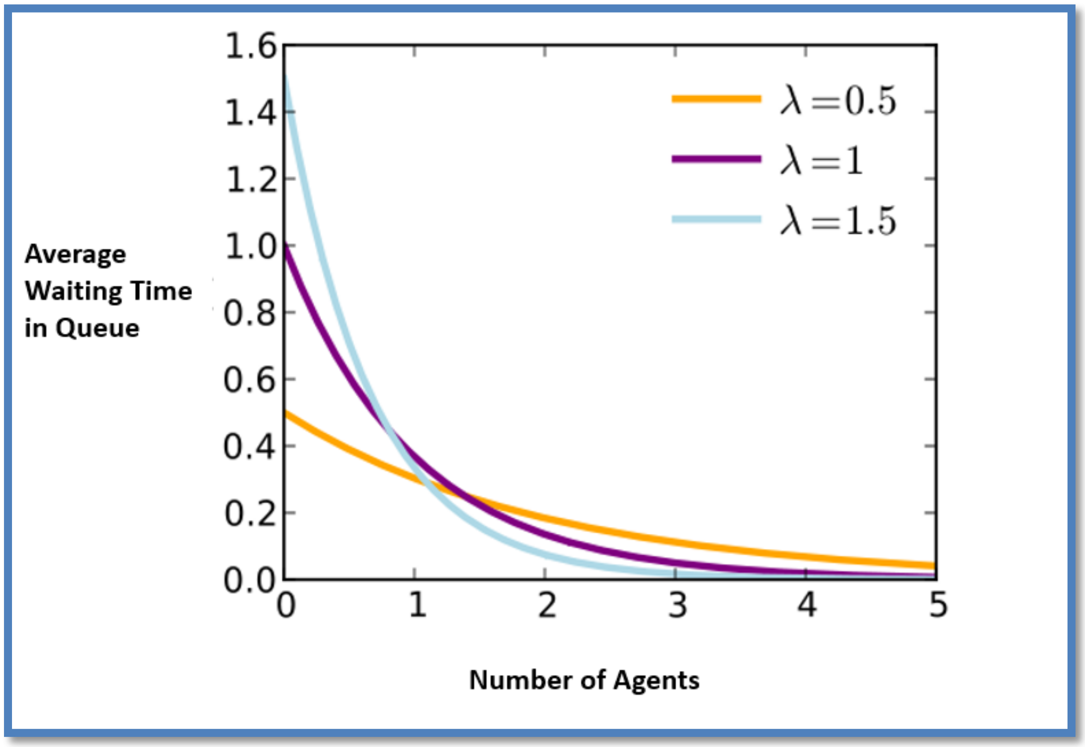


FIGURE 8: ERLANG C FORMULA FOR AVERAGE QUEUE WAITING TIME VERSUS NUMBER OF AGENTS

Next, we discuss the Caller, Distributor and Agent Node Applications designed for this Erlang C Model framework.

Caller Node Application

The Erlang C Caller Node Application is shown in Figure 9.

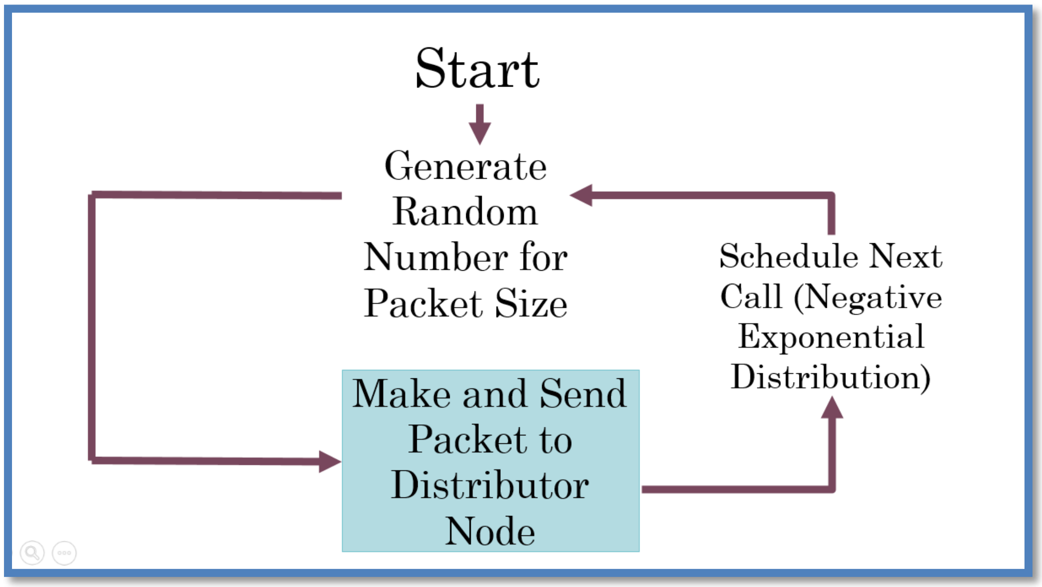

Start
Generate Random Number for Packet Size
Schedule Next Call (Negative Exponential Distribution)
Make and Send Packet to Distributor Node

FIGURE 9: ERLANG C CALLER NODE APPLICATION

```
void SendPacket ()
{…

        Size=PacketSizeGenerator->GetInteger ();                    //Generate Random Size
        Ptr<Packet> packet=Create<Packet> (size);                        //Make Packet

        Socket->Send (packet);                                           //Send Packet

        time= InterArrivalTimeGenerator->GetInteger ();
        Time tnext (Seconds (time));                                //Schedule self-call
        SendEvent=Simulator: Schedule (tnext, &CallerApp: SendPacket, this);

}
```

Distributor Node Application

The Erlang C Distributor Node Application is shown in Figure 10.

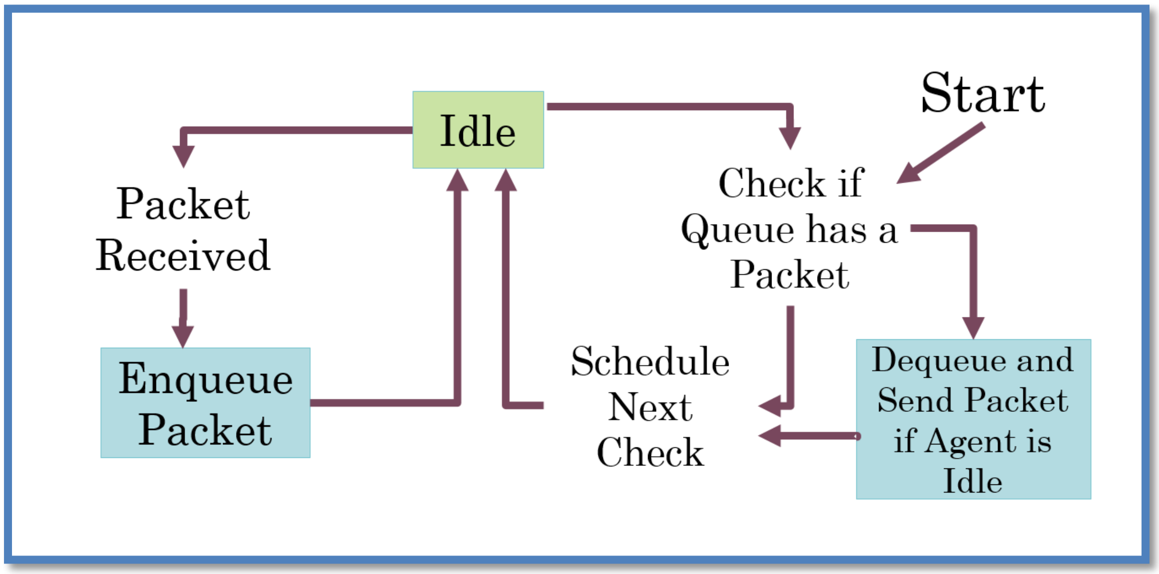

Idle
Start
Packet Received
Check if Queue has a Packet
Enqueue Packet
Schedule Next Check
Dequeue and Send Packet if Agent is Idle

FIGURE 10: ERLANG C DISTRIBUTOR NODE APPLICATION

```
void SocketRecv (Ptr<Socket> socket)                              //Receive Call Back Function
{…
         Ptr<Packet> packet=MySocket->Recv ();
        Queue->Enqueue (packet);                                          //Enqueue Packet
        MySocket->SetRecvCallback (MakeCallback (&DistributorApp: SocketRecv, this));
}
```

```
void SendPacket ()
{…
         if (idleArray [PeerNumber] && (queue->NumberOfPacketsInQueue>0))          //Check
                                                          if Agent is idle and Queue has a Packet
         {
                MyPacket=queue->Dequeue ();                                //Dequeue Packet
                MySocket->Send (MyPacket);                            //Send Packet to Agent
                Time tnext (Seconds (1);                                 //Schedule self-call
                SendEvent=Simulator: Schedule (tnext, &DistributorApp: SendPacket, this);
         }
}
```

Agent Node Application

The Erlang C Agent Node Application is shown in Figure 11.

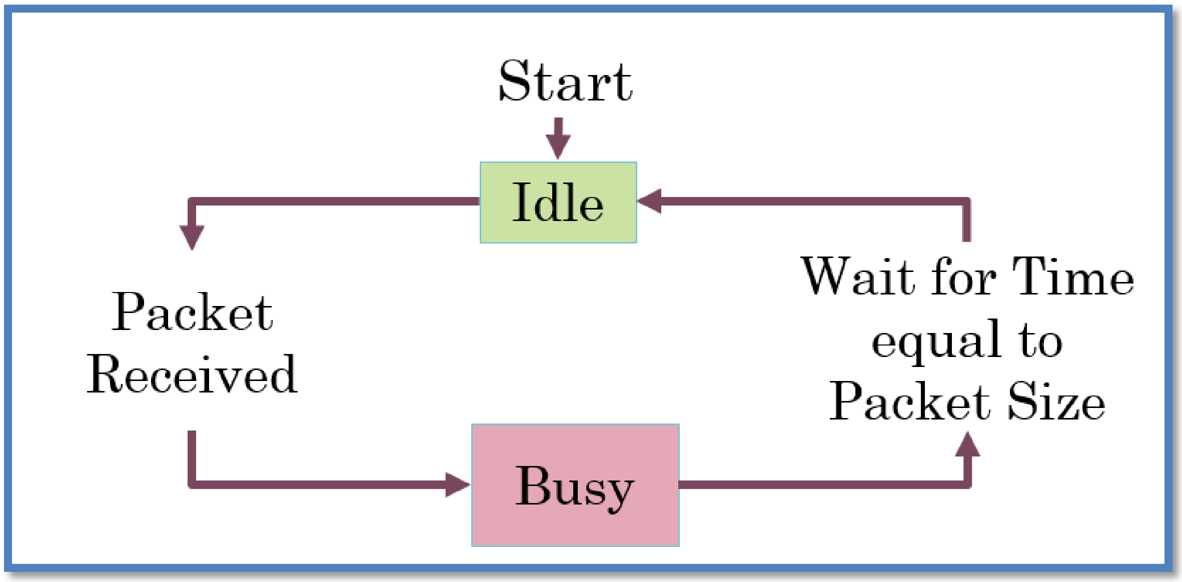


FIGURE 11: ERLANG C AGENT NODE APPLICATION

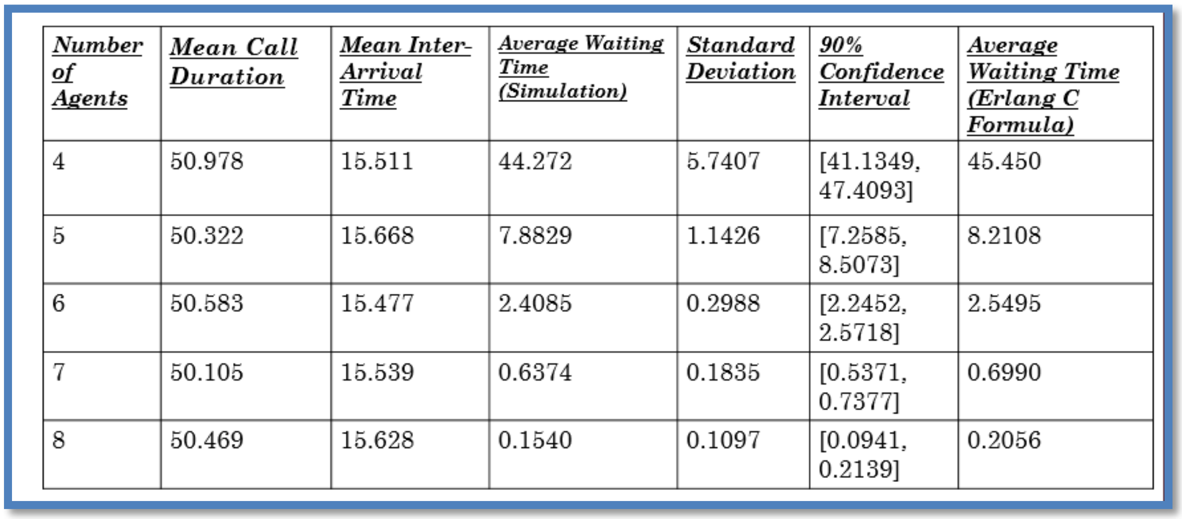
```
void SocketRecv (Ptr<Socket> socket)
{…
        MyPacket=MySocket->Recv ();                                          //Receive Packet
        PacketSize=MyPacket->GetSize ();
        IdleArray [OperatorNumber] = 0;                                                //Busy
        Time tnext (Seconds (PacketSize));        //Call ServiceComplete after time = PacketSize
        SendEvent=Simulator: Schedule (tnext, OperatorApp::ServiceComplete, this);
}
```

```
void ServiceComplete ()
 {…
        IdleArray [OperatorNumber] =1;                                                 //Idle
        MySocket->SetRecvCallback (MakeCallback (&OperatorApp: SocketRecv, this));
                                                           //Call SocketRecv when Packet Comes
}
```

After having discussed the Caller, Distributor and Agent Node Applications, now we will discuss the simulation results and compare them with Erlang C Calculator [2].

3.2b The Erlang C Simulation and Erlang C Calculator Results

The Simulation technique was that we generated 10000 calls and calculated the mean queue waiting time for that run. Then we repeated these 10 times to obtain a 90% confidence interval and mean of distribution for a particular number of agents. The results are given in Table 1. We plotted the simulation results against Erlang C Calculator results [2] as shown in Figure 12.



Number of Agents	*Mean Call Duration*	*Mean Inter-Arrival Time*	*Average Waiting Time (Simulation)*	*Standard Deviation*	*90% Confidence Interval*	*Average Waiting Time (Erlang C Formula)*
4	50.978	15.511	44.272	5.7407	[41.1349, 47.4093]	45.450
5	50.322	15.668	7.8829	1.1426	[7.2585, 8.5073]	8.2108
6	50.583	15.477	2.4085	0.2988	[2.2452, 2.5718]	2.5495
7	50.105	15.539	0.6374	0.1835	[0.5371, 0.7377]	0.6990
8	50.469	15.628	0.1540	0.1097	[0.0941, 0.2139]	0.2056

TABLE 1: ERLANG C SIMULATION RESULTS

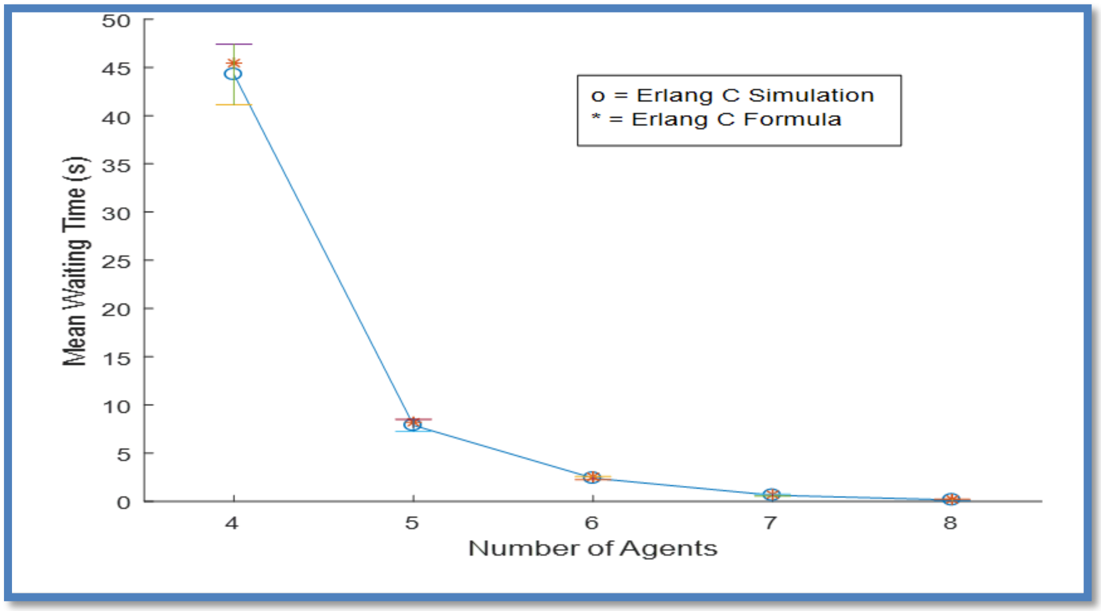


FIGURE 12: ERLANG C SIMULATION AND CALCULATOR GRAPHS

After Verifying the Erlang C Model, we built the popular Erlang X model, which incorporates call abandonments, call drops, and skill based routing.

3.3 Improved System Model: Erlang X

The Erlang X or Erlang A model builds on the Erlang C Model by including abandonments of callers i.e. Callers can abandon Queue after waiting for some time as shown in Figure 14. It is modeled using M/M/M+N Markov Chain [13]. Packets can be dropped if queue is full which models the limited number of telephone lines available. Calls can require multiple skills and Agents can have multiple skills as shown in Figure 13. Specialists have only one skill whereas Generalists can have two or more skills. All Calls have same Call Duration distribution.

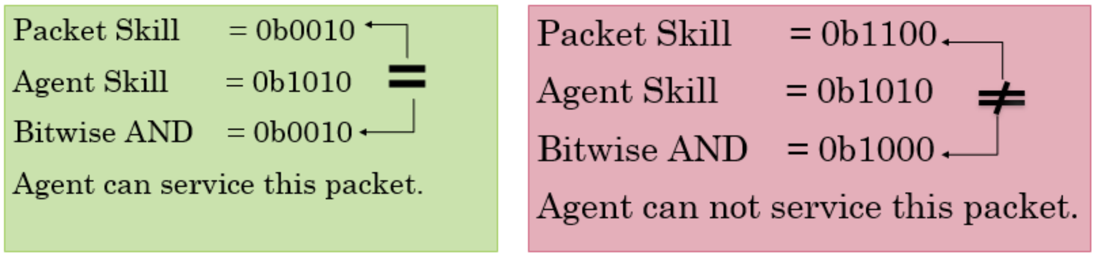


FIGURE 13: SKILLS MATCHING FOR SKILL BASED ROUTING

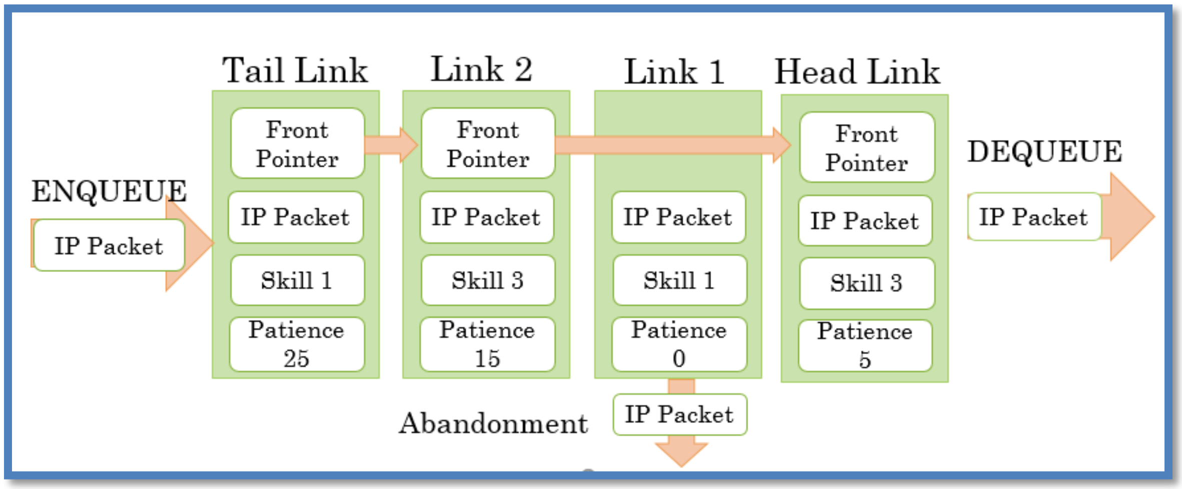


FIGURE 14: QUEUE ABANDONMENTS

Next, we discuss the implementation details and Simulation Methodology of Erlang X model.

3.3a Simulation Methodology

Building on the Erlang C model, we modified the Queue Link by keeping a Skill section in each Link to aid distribution of packets to Agents of desired skill. The QueueWaitingTime Element denotes the Maximum time the caller will wait in the Queue before abandoning the Queue.

```
Struct Link
{
            Link *      Back;
            Link *      Front;
            Uint32_t    Data;
            Uint32_t    Skill;
            Ptr<Packet> pkt;
            Time        EnqueueTime;
            Uint32_t    QueueWaitingTime;
}
```

The Patience Counter of each Link decrements every second. If Timer of a Link hits zero, we remove that Link from Queue. Hence, the caller has abandoned the Queue before being serviced.

```
void CountDown ()
{…
            If (NumberOfPacketsInQueue>0)
            {
                      Link* a            =Head;
                      While (a! =Tail)                                        //Traverse Entire Queue
                      {
                       a                 =a->Back;
                       a                 ->QueueWaitingTime--;                //Decrement counter
                      If (a->QueueWaitingTime==0) {a-Back->Front = a->Front ;}    // Abandonment
                       }
            }
}
```

Before Enqueue, we check if queue is full. If queue is full, packet is dropped.

```
void Enqueue (Ptr<Packet> Packet)
{…
            If (NumberOfPacketsInQueue<MaximumQueueLimit)              //Check Queue Limit
           {
                      Packet->Front = Tail-> Front;
                      Tail->Front = packet;                                      //Enqueue
           }
            Else
             {
                       Myfile<<"Packet Dropped, MaximumQueueLimit Exceeded"<<endl;        //Drop
             }
}
```

To prevent one packet from blocking entire queue, we allow distributors to pick packet from any link of queue.

```
If (idle [PeerNumber] && (queue->NumberOfPacketsInQueue>0))
{…
          DistributorQueue::Link* a=queue->Head->Back;
          DistributorQueue::Link* b=0;
          While (a! =queue->Tail)                                  //Traverse Entire Queue
          {
                    If ((SkillArray [PeerNumber] & a->Skill)>0)              //Skill Matching
                    {
                              Socket->Send (a->Packet)                         //Send Packet
                    }
          }
}
```

A Caller Node keeps generating IP packets and sends them to the distributor node using a Point-to-Point Link. The Packet length follows a fixed distribution and has a fixed mean value. The Call Skill is denoted by a random variable and is stored in the TOS Tag of the IP Packet. The Caller Patience random variable is stored in the TTL Tag of the IP Packet. This is implemented in NS3 by a function that generates a random variable for Call length, Call Skill and Caller Patience, makes an IP packet of that size, adds the IP Tags and then sends it using a socket. The function then makes a self-call by scheduling the next call after a time interval given by a random number following a negative exponential distribution.

The Distributor Node receives the IP packets and keeps adding them to the tail of the Queue if the Queue is not completely full in which case call is dropped and lost. This is done in NS3 by calling an interrupt to Enqueue function whenever a packet arrives at the socket. It also checks every second if any Agents are idle and distributes the Packets from the Queue to them. It traverses the entire Queue and checks if the idle Agents have the skill required by the Packet. In NS3, it schedules a call to SendPacket function every second, which does the routing of calls using Point to Point Links to Agents with required skill.The Agent Node receive the IP packet from the Distributor Node and becomes busy for the duration of the call. Hence, it changes the Boolean idle flag to zero. While it is busy, it cannot receive any more packets. It schedules a call to ServiceComplete function after a period equal to the length of the call after which it changes its status to idle again.

The Analytical Erlang X formula [14] is presented below.

Mean Patience of Callers = θ^{-1}, Arrival Rate = λ, Service rate = $n\mu$

Gamma Function $\gamma(x, y) = \int_0^y t^{x-1} e^{-t} dt$, $x \geq 0$, $y \geq 0$

$$J = \frac{e^{\lambda/\theta}}{\theta} \left(\frac{\theta}{\lambda}\right)^{\frac{n\mu}{\theta}} \gamma\left(\frac{n\mu}{\theta}, \frac{\lambda}{\theta}\right),\ \varepsilon = \frac{\sum_{j=0}^{n-1} \left(\frac{1}{j!}\right)\left(\frac{\lambda}{\mu}\right)^j}{\left(\frac{1}{(n-1)!}\right)\left(\frac{\lambda}{\mu}\right)^{n-1}},\ P\,(\text{Wait} > 0) = \frac{\lambda J}{\varepsilon + \lambda J}\,(1- \theta)$$

Now we discuss the Caller, Distributor and Agent Node Applications designed for this Erlang X Model framework.

Caller Node Application

The Erlang X Caller Node Application is shown in Figure 15.

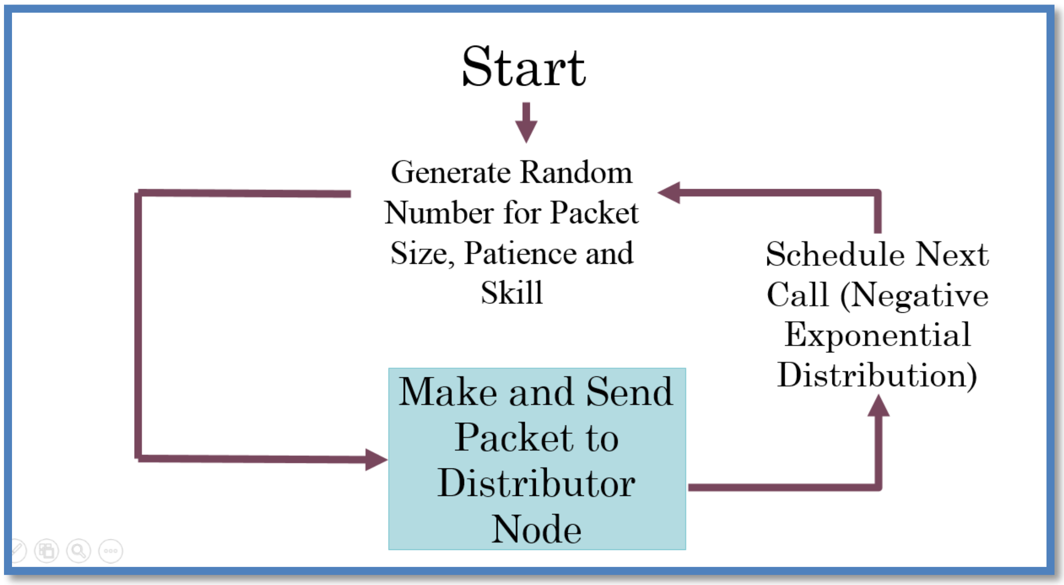

Start
Generate Random Number for Packet Size, Patience and Skill
Schedule Next Call (Negative Exponential Distribution)
Make and Send Packet to Distributor Node

FIGURE 15: ERLANG X CALLER NODE APPLICATION

```
void SendPacket ()
{…
         Size          =PacketSizeGenerator        ->GetInteger ();
        Waitingtime =WaitingTimeGenerator       ->GetInteger ();
        Skill          =SkillTypeGenerator         ->GetInteger ();
        Time          =InterArrivalTimeGenerator->GetInteger ();   //Generate Random Numbers

         Socket->SetIpTos (skill);                                        //Set Packet Skill
         Socket->SetIpTtl (waitingtime);                          //Set Packet Waiting Time

         Ptr<Packet> packet=Create<Packet> (size);                          //Make Packet

         Socket->Send (packet);                                              //Send Packet

        Time tnext (Seconds (static_cast<double> (time)));                 //Schedule self-call
         SendEvent=Simulator: Schedule (tnext, &CallerApp: SendPacket, this);
}
```

Distributor Node Application

The Erlang X Distributor Node Application is shown in Figure 16.

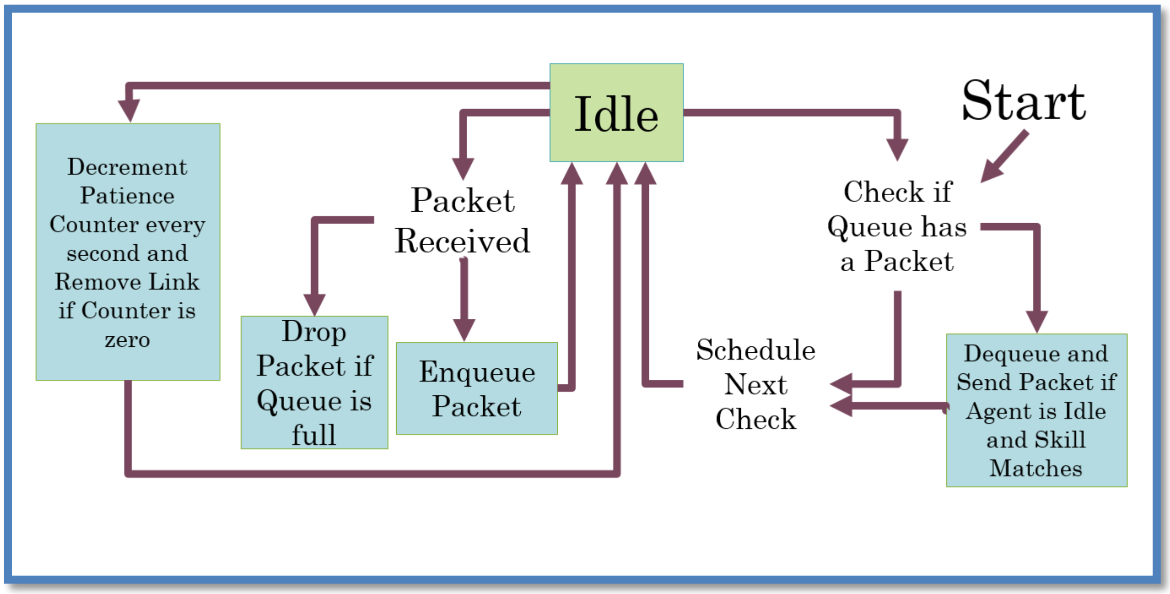

Idle
Start
Decrement Patience Counter every second and Remove Link if Counter is zero
Packet Received
Drop Packet if Queue is full
Enqueue Packet
Schedule Next Check
Check if Queue has a Packet
Dequeue and Send Packet if Agent is Idle and Skill Matches

FIGURE 16: ERLANG X DISTRIBUTOR NODE APPLICATION

```
void SkillBasedRouting ()
{…
            If (idle [PeerNumber] && (queue->NumberOfPacketsInQueue>0)) //Checks
            {
                        DistributorQueue::Link* a=queue->Head->Back;
                         If ((SkillArray [PeerNumber] & a->Skill)>0)                              //Match Skill
                         MySocket                  ->Send (a->Packet);                            //Send Packet
                          Time                   tnext (Seconds (1));                          //Schedule self-call
             SendEvent =Simulator::Schedule (tnext, &DistributorApp: SkillBasedRouting, this);
            }
}
```

```
void SocketRecv (Ptr<Socket> socket)
{…
           Address from;
            Ptr<Packet> packet=MySocket->RecvFrom (from);                          //Receive Packet
            Queue->Enqueue (packet);                                               //Enqueue Packet
           MySocket->SetRecvCallback (MakeCallback (&ReceiverApp: SocketRecv, this));
}
```

Agent Node Application

The Erlang X Agent Node Application is shown in Figure 17.

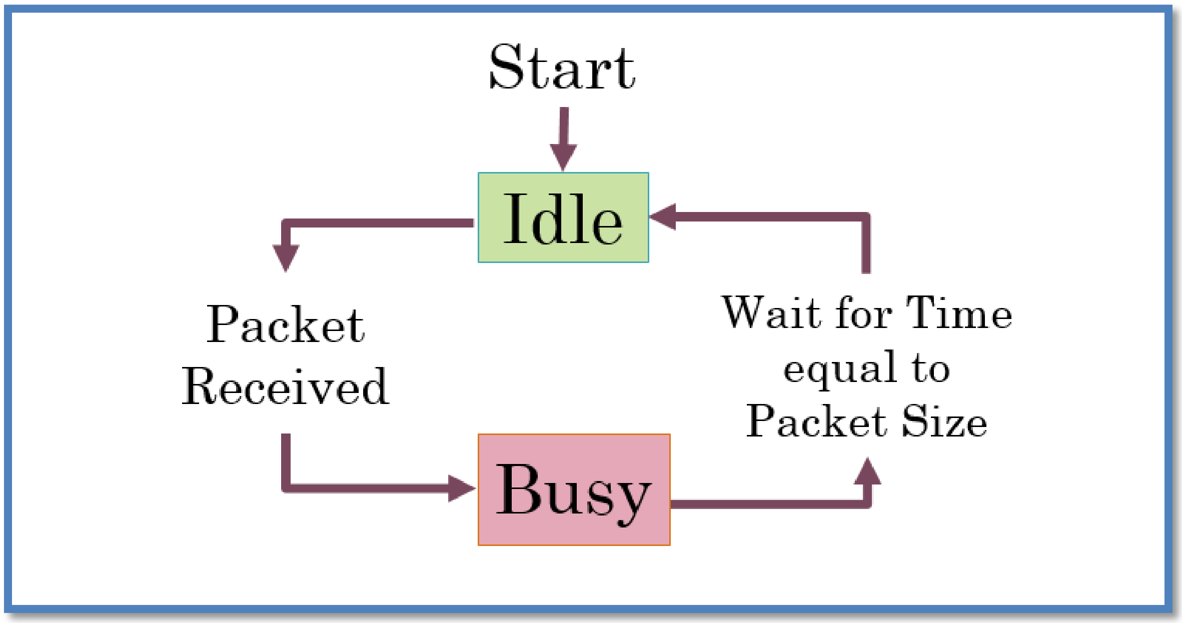


FIGURE 17: ERLANG X AGENT NODE APPLICATION

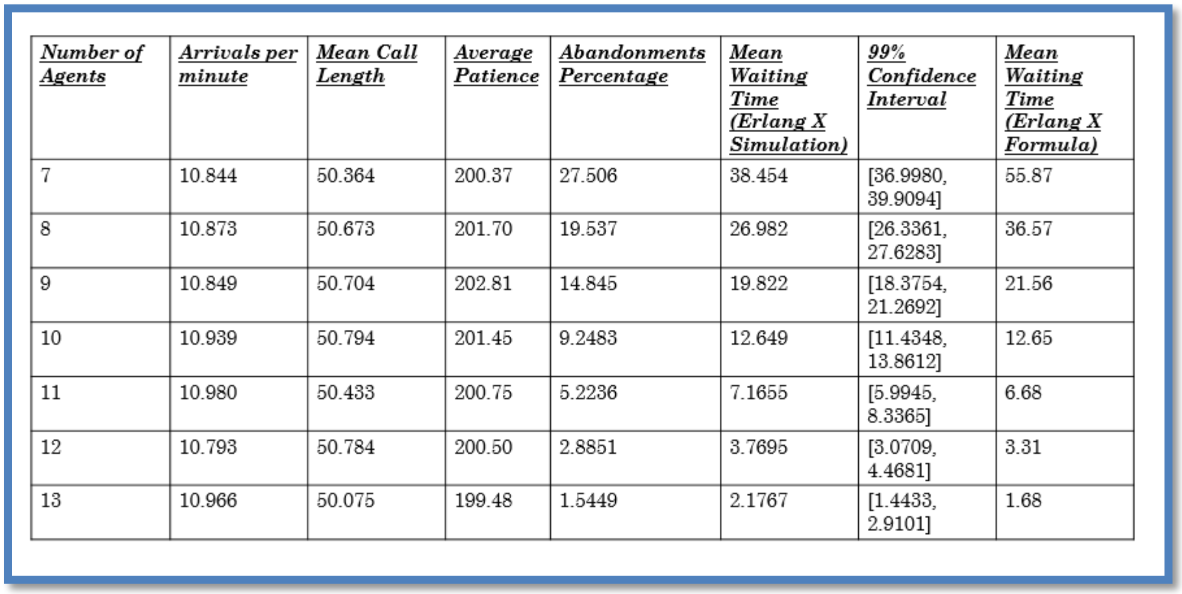
```
void SocketRecv (Ptr<Socket> socket)
{…
 MyPacket      =MySocket->Recv ();                                    //Receive Packet
 PacketSize     =MyPacket->GetSize ();
 Time tnext (Seconds (PacketSize);              //call ServiceComplete after time = Packet Size
 SendEvent      =Simulator::Schedule (tnext, &AgentApp: ServiceComplete, this);
IdleArray [AgentNumber] =0;                                              //Become Busy
 }
```

```
void ServiceComplete ()
{…
 IdleArray [AgentNumber] =1;                                              //Become Idle
MySocket ->SetRecvCallback (MakeCallback (&AgentApp: SocketRecv, this));
                                          //Call SocketRecv Function when call arrives
}
```

After having discussed the Caller, Distributor and Agent Node Applications, we will now present the simulation results and compare them with Erlang X Calculator [3].

3.3b The Erlang X Simulation and Erlang X Calculator Results.

Our Simulation technique was that we generated 10000 calls and calculated the mean queue waiting time for that run. Then we repeated 10 times to obtain a 90% confidence interval and mean of distribution for a particular number of agents. We plotted the simulation results against Erlang X Calculator results [3] as shown in Table 2 and Figure 18.



Number of Agents	*Arrivals per minute*	*Mean Call Length*	*Average Patience*	*Abandonments Percentage*	*Mean Waiting Time (Erlang X Simulation)*	*99% Confidence Interval*	*Mean Waiting Time (Erlang X Formula)*
7	10.844	50.364	200.37	27.506	38.454	[36.9980, 39.9094]	55.87
8	10.873	50.673	201.70	19.537	26.982	[26.3361, 27.6283]	36.57
9	10.849	50.704	202.81	14.845	19.822	[18.3754, 21.2692]	21.56
10	10.939	50.794	201.45	9.2483	12.649	[11.4348, 13.8612]	12.65
11	10.980	50.433	200.75	5.2236	7.1655	[5.9945, 8.3365]	6.68
12	10.793	50.784	200.50	2.8851	3.7695	[3.0709, 4.4681]	3.31
13	10.966	50.075	199.48	1.5449	2.1767	[1.4433, 2.9101]	1.68

TABLE 2: ERLANG X SIMULATION AND CALCULATOR RESULTS

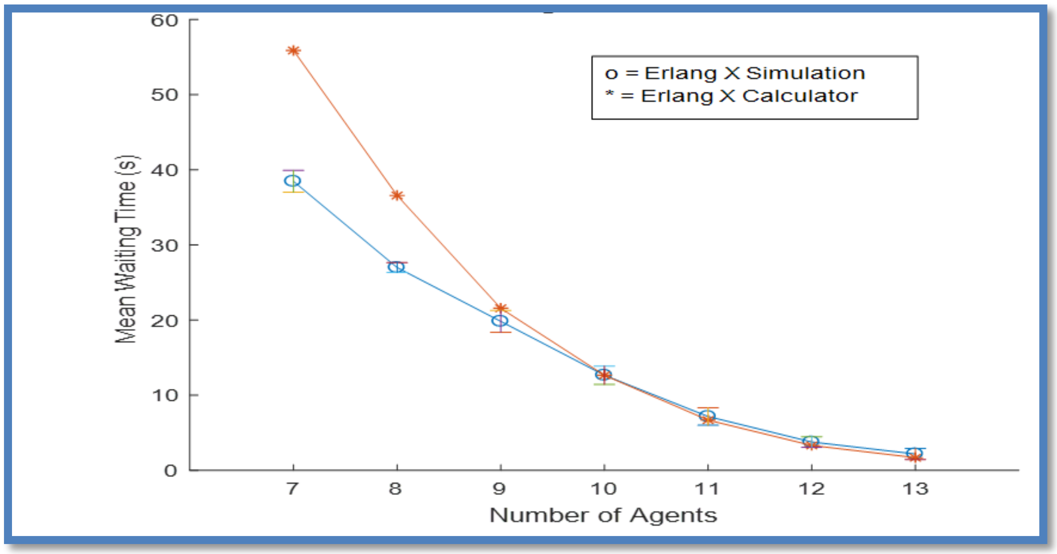
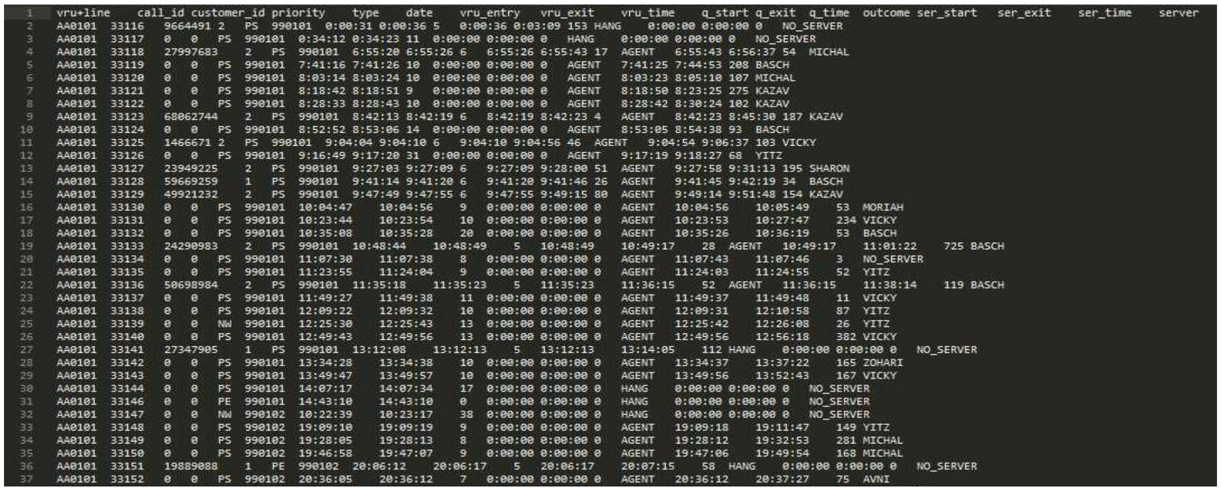


FIGURE 18: ERLANG X SIMULATION AND CALCULATOR GRAPHS

After verifying the Erlang X model, we built our final Call Centre model by adding retrials, Virtual Response Unit, heterogeneous agents, time varying arrival rates and Agent Schedules.

4 Finalized System Model: Real Call Center

In order to better understand the real call center, we received the data from materials of a course entitled Service Engineering, taught by Professor Avishai Mandelbaum at the Technion, Israel [9]. The records are shown in Figure 19. It emphasized, theoretically, on Queuing Theory, and, practically, on Telephone Call Centers and Service Networks. We extracted a one-day data, sorted its respected fields, and based our simulations upon them. This worked as both a limitation, in terms of more close approximation, and an extension, to match theoretical results for our proposed model to that of theoretical models.



```
vru+line    call_id customer_id priority    type    date    vru_entry   vru_exit    vru_time    q_start q_exit  q_time  outcome ser_start   ser_exit    ser_time    server
AA0101  33116   9664491 2   PS  990101  0:00:31 0:00:36 5   0:00:36 0:03:09 153 HANG    0:00:00 0:00:00 0   NO_SERVER
AA0101  33117   0   0   PS  990101  0:34:12 0:34:23 11  0:00:00 0:00:00 0   HANG    0:00:00 0:00:00 0   NO_SERVER
AA0101  33118   27997683    2   PS  990101  6:55:20 6:55:26 6   6:55:26 6:55:43 17  AGENT   6:55:43 6:56:37 54  MICHAL
AA0101  33119   0   0   PS  990101  7:41:16 7:41:26 10  0:00:00 0:00:00 0   AGENT   7:41:25 7:44:53 208 BASCH
AA0101  33120   0   0   PS  990101  8:03:14 8:03:24 10  0:00:00 0:00:00 0   AGENT   8:03:23 8:05:10 107 MICHAL
AA0101  33121   0   0   PS  990101  8:18:42 8:18:51 9   0:00:00 0:00:00 0   AGENT   8:18:50 8:23:25 275 KAZAV
AA0101  33122   0   0   PS  990101  8:28:33 8:28:43 10  0:00:00 0:00:00 0   AGENT   8:28:42 8:30:24 102 KAZAV
AA0101  33123   68062744    2   PS  990101  8:42:13 8:42:19 6   8:42:19 8:42:23 4   AGENT   8:42:23 8:45:30 187 KAZAV
AA0101  33124   0   0   PS  990101  8:52:52 8:53:06 14  0:00:00 0:00:00 0   AGENT   8:53:05 8:54:38 93  BASCH
AA0101  33125   1466671 2   PS  990101  9:04:04 9:04:10 6   9:04:10 9:04:56 46  AGENT   9:04:54 9:06:37 103 VICKY
AA0101  33126   0   0   PS  990101  9:16:49 9:17:20 31  0:00:00 0:00:00 0   AGENT   9:17:19 9:18:27 68  YITZ
AA0101  33127   23949225    2   PS  990101  9:27:03 9:27:09 6   9:27:09 9:28:00 51  AGENT   9:27:58 9:31:13 195 SHARON
AA0101  33128   59669259    1   PS  990101  9:41:14 9:41:20 6   9:41:20 9:41:46 26  AGENT   9:41:45 9:42:19 34  BASCH
AA0101  33129   49921232    2   PS  990101  9:47:49 9:47:55 6   9:47:55 9:49:15 80  AGENT   9:49:14 9:51:48 154 KAZAV
AA0101  33130   0   0   PS  990101  10:04:47    10:04:56    9   0:00:00 0:00:00 0   AGENT   10:04:56    10:05:49    53  MORIAH
AA0101  33131   0   0   PS  990101  10:23:44    10:23:54    10  0:00:00 0:00:00 0   AGENT   10:23:53    10:27:47    234 VICKY
AA0101  33132   0   0   PS  990101  10:35:08    10:35:28    20  0:00:00 0:00:00 0   AGENT   10:35:26    10:36:19    53  BASCH
AA0101  33133   24290983    2   PS  990101  10:48:44    10:48:49    5   10:48:49    10:49:17    28  AGENT   10:49:17    11:01:22    725 BASCH
AA0101  33134   0   0   PS  990101  11:07:30    11:07:38    8   0:00:00 0:00:00 0   AGENT   11:07:43    11:07:46    3   NO_SERVER
AA0101  33135   0   0   PS  990101  11:23:55    11:24:04    9   0:00:00 0:00:00 0   AGENT   11:24:03    11:24:55    52  YITZ
AA0101  33136   50698984    2   PS  990101  11:35:18    11:35:23    5   11:35:23    11:36:15    52  AGENT   11:36:15    11:38:14    119 BASCH
AA0101  33137   0   0   PS  990101  11:49:27    11:49:38    11  0:00:00 0:00:00 0   AGENT   11:49:37    11:49:48    11  VICKY
AA0101  33138   0   0   PS  990101  12:09:22    12:09:32    10  0:00:00 0:00:00 0   AGENT   12:09:31    12:10:58    87  YITZ
AA0101  33139   0   0   NW  990101  12:25:30    12:25:43    13  0:00:00 0:00:00 0   AGENT   12:25:42    12:26:08    26  YITZ
AA0101  33140   0   0   PS  990101  12:49:43    12:49:56    13  0:00:00 0:00:00 0   AGENT   12:49:56    12:56:18    382 VICKY
AA0101  33141   27347905    1   PS  990101  13:12:08    13:12:13    5   13:12:13    13:14:05    112 HANG    0:00:00 0:00:00 0   NO_SERVER
AA0101  33142   0   0   PS  990101  13:34:28    13:34:38    10  0:00:00 0:00:00 0   AGENT   13:34:37    13:37:22    165 ZOHARI
AA0101  33143   0   0   PS  990101  13:49:47    13:49:57    10  0:00:00 0:00:00 0   AGENT   13:49:56    13:52:43    167 VICKY
AA0101  33144   0   0   PS  990101  14:07:17    14:07:34    17  0:00:00 0:00:00 0   HANG    0:00:00 0:00:00 0   NO_SERVER
AA0101  33146   0   0   PE  990101  14:43:10    14:43:10    0   0:00:00 0:00:00 0   HANG    0:00:00 0:00:00 0   NO_SERVER
AA0101  33147   0   0   NW  990102  10:22:39    10:23:17    38  0:00:00 0:00:00 0   HANG    0:00:00 0:00:00 0   NO_SERVER
AA0101  33148   0   0   PS  990102  19:09:10    19:09:19    9   0:00:00 0:00:00 0   AGENT   19:09:18    19:11:47    149 YITZ
AA0101  33149   0   0   PS  990102  19:28:05    19:28:13    8   0:00:00 0:00:00 0   AGENT   19:28:12    19:32:53    281 MICHAL
AA0101  33150   0   0   PS  990102  19:46:58    19:47:07    9   0:00:00 0:00:00 0   AGENT   19:47:06    19:49:54    168 MICHAL
AA0101  33151   19889088    1   PE  990102  20:06:12    20:06:17    5   20:06:17    20:07:15    58  HANG    0:00:00 0:00:00 0   NO_SERVER
AA0101  33152   0   0   PS  990102  20:36:05    20:36:12    7   0:00:00 0:00:00 0   AGENT   20:36:12    20:37:27    75  AVNI
```

FIGURE 19: CALL CENTER RECORDS

Next, we present the strategy used to extract meaningful information from the obtained call center records in factors influencing the choice of final design section.

4.1 Factors Influencing the Choice of Final Design

Our initial model was closer to the ideal situation in which the calls arrival follows the Poisson distribution with the fixed arrival rate. However, in case of real call center, the arrival rate is not fixed but it varies with time and also follows a distribution. To make our model more realistic we also added the Virtual Response Unit (VRU) functionality. Also, the calls were routed according to the single routing scheme, which was skill based routing, irrespective of the calls arrival rate, server's experience, call serving speed etc. which resulted in the average performance. In order to increase the performance and to reduce the queue waiting time we devised four different routing schemes, which are: Fastest Server First Routing (FSF), Longest Idle First routing (LIF), Lowest cost first Routing (LCR) and the Skill based routing (SBR). We utilized these routing schemes according to the changing call arrival rate with respect to time.

Moreover, in the results of our initial model, the mean waiting time found from the simulation was deviating from that of the real call center for lesser number of agents, though the results were quite similar for the larger number of agents. Thus, in order to match the results with the real call center, we did the data analysis on the real call center data and found the distributions for each variable like average call time, probability of waiting in the queue, traffic intensity, agent's service level, agent's salary etc. and used these variables for implementing various routing schemes.
In the real call center Arrival, rates are different at different times of day. Hence, we must change the average arrival rate of calls each hour. Callers must specify their required skill using Virtual Response Unit VRU; Callers can abandon Queue after waiting for some time and hang up at any time. The calls are routed based on their required skill. The Agents have inhomogeneous service time distributions and up to six skills. There are 23 agents available in the call center. Their scheduled are dynamic i.e. more agents are called to handle calls when traffic is high. Each Agents has an average call service time of three hours and answers around 75 calls each day. As in Erlang X, Specialists have only one skill and Generalists can have two or more skills. Agents are scheduled depending on Traffic Intensity.

We used the call center data to extract information about agents and callers. First, we used all the calls to extract the Call Durations Distribution. The same distribution can be generated in NS3 using a Random Variable Generator. There were a lot of call drops and call abandonments with a call duration of zero seconds. They were used to estimate the average patience of caller in the queue. The data revealed that the Agents were scheduled dynamically based on the arriving Call traffic. The Queue Waiting Times Distribution revealed that Queue waiting time was higher when call traffic was higher. We also used the Virtual Response Unit Times Distribution to estimate how much time the average caller spends in the VRU. The Distributions were input into the simulator to simulate what happened in the real call center on a particular day.

The Extracted Distributions of Call Durations are given in Figure 19.

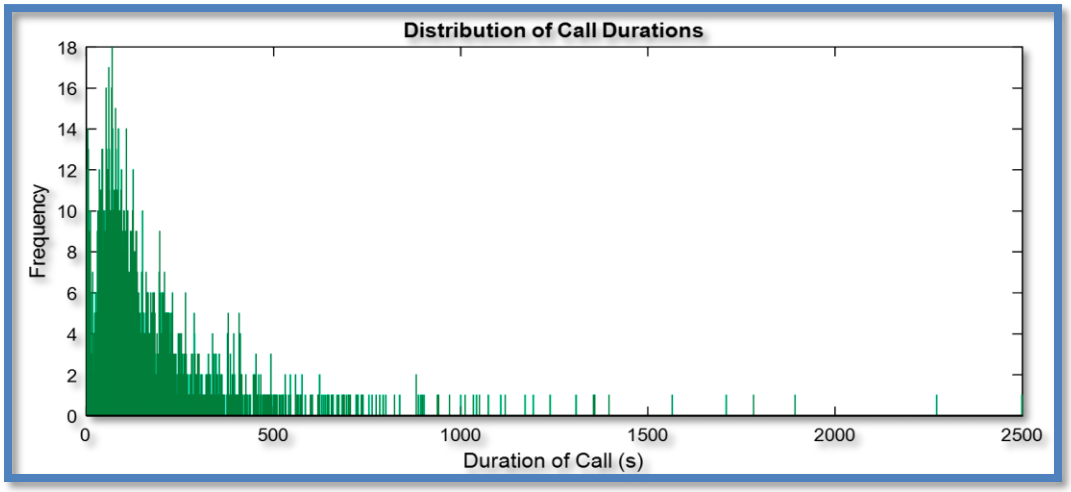


FIGURE 20: DISTRIBUTION OF CALL DURATIONS

The Extracted Distributions of Queue Waiting Times are given in Figure 20.

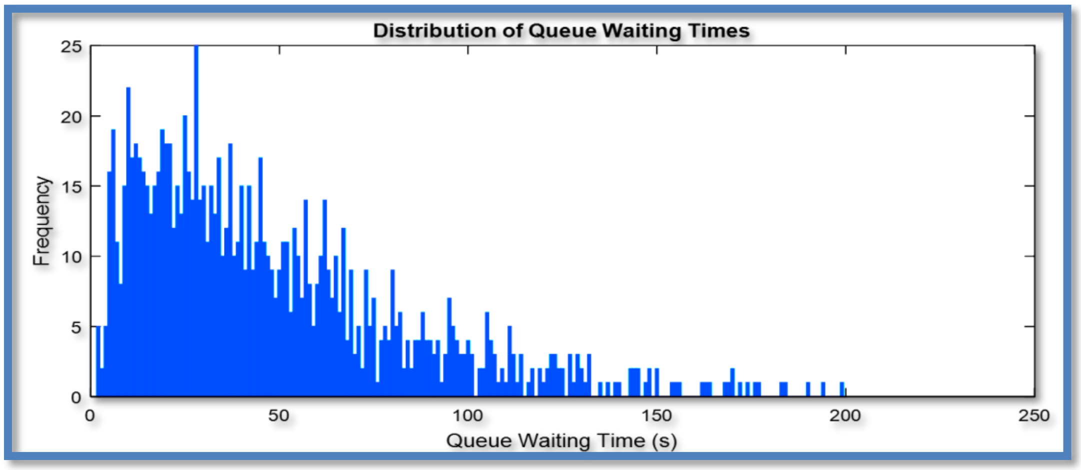


FIGURE 21: DISTRIBUTION OF QUEUE WAITING TIMES

The Extracted Distributions of VRU Durations are given in Figure 21.

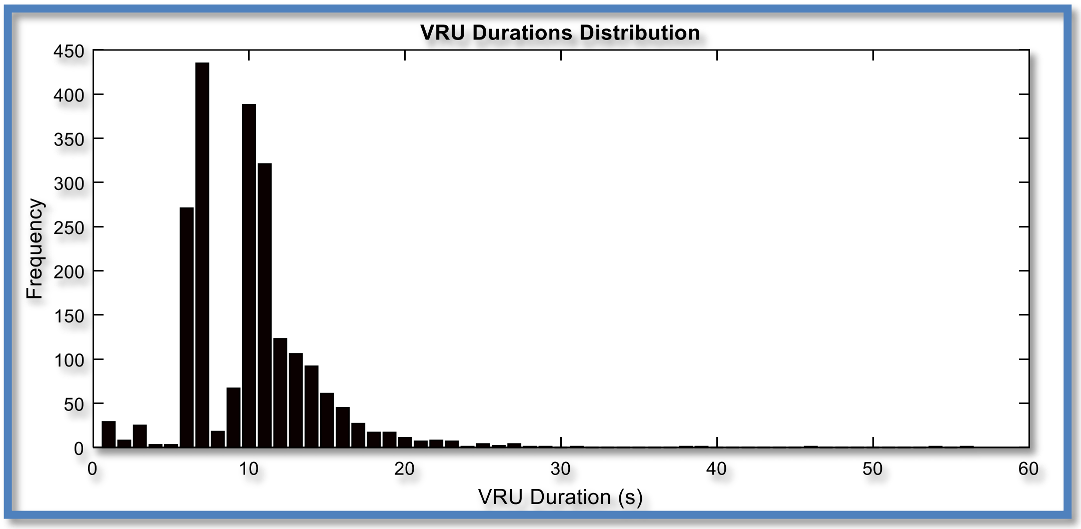
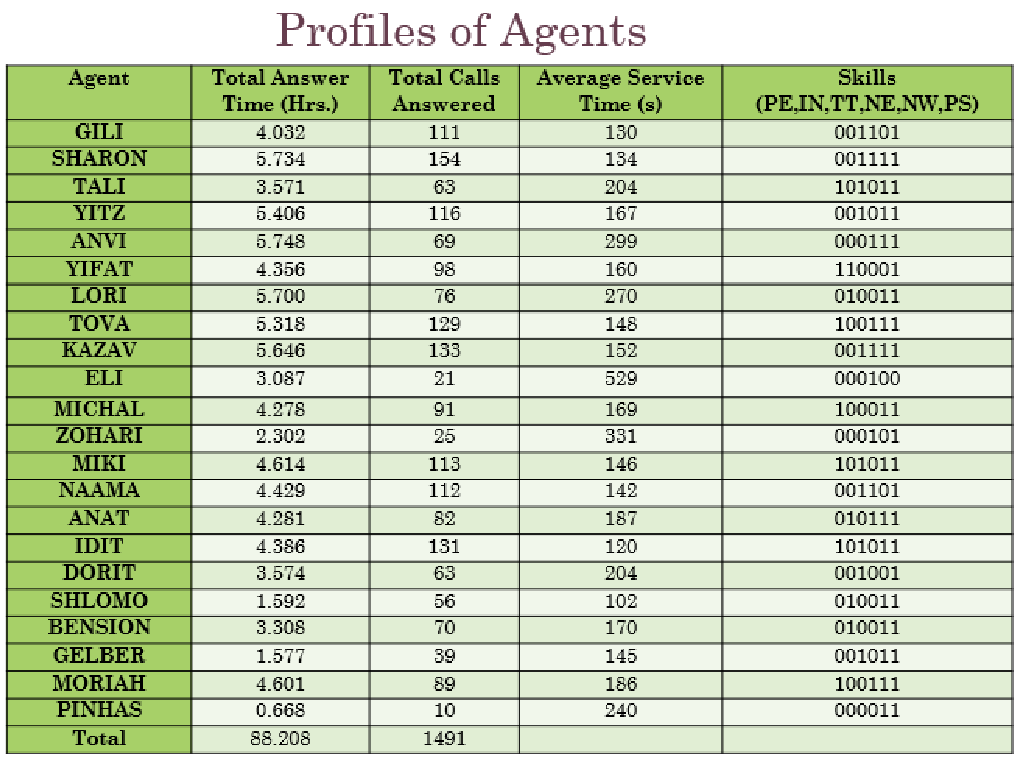


FIGURE 22: DISTRIBUTION OF VRU DURATIONS

The Extracted Profiles of Agents are given in Table 3.



Profiles of Agents

Agent	Total Answer Time (Hrs.)	Total Calls Answered	Average Service Time (s)	Skills (PE,IN,TT,NE,NW,PS)
GILI	4.032	111	130	001101
SHARON	5.734	154	134	001111
TALI	3.571	63	204	101011
YITZ	5.406	116	167	001011
ANVI	5.748	69	299	000111
YIFAT	4.356	98	160	110001
LORI	5.700	76	270	010011
TOVA	5.318	129	148	100111
KAZAV	5.646	133	152	001111
ELI	3.087	21	529	000100
MICHAL	4.278	91	169	100011
ZOHARI	2.302	25	331	000101
MIKI	4.614	113	146	101011
NAAMA	4.429	112	142	001101
ANAT	4.281	82	187	010111
IDIT	4.386	131	120	101011
DORIT	3.574	63	204	001001
SHLOMO	1.592	56	102	010011
BENSION	3.308	70	170	010011
GELBER	1.577	39	145	001011
MORIAH	4.601	89	186	100111
PINHAS	0.668	10	240	000011
Total	88.208	1491		

TABLE 3: PROFILES OF AGENTS

Now that we understand the factors influencing the choice of final design, we can visualize the Final System Diagram.

4.2 Final System Diagram

In our final design, the calls arrive according to the Poisson distribution with the variable arrival rate, which keeps, on changing with time. There is also the virtual response queue in which all the calls wait for some time specifying their requirements. The agent nodes are characterized based on their exhaustion level, efficiency level, training level, age, experience, leaves, breaks, schedules, speed, and learning, Calls Serviced, Mean Service Time, and Quitting Probability. Where the distribution of all such variables are drawn from the real call center data. In the final design, the basic routing was done based on the agent’s skills and the caller’s requirements. However, according to the variable arrival rate, dynamic routing was utilized. FSF was used when the calls arrival rate was higher so that more calls could be served and the calls drop rate would be small. In case of moderate arrival, rate lowest cost first routing was used to optimize the cost of running the call center. LIF was used when the arrival rate was small so that the non-experienced agents could gain experience by handling more calls. Upon Arrival, the Caller must enter the Virtual Response Unit Linked List where it is assigned a VRU Counter that decrements every second. When the VRU Counter hits zero, the Caller enters the Callers Queue if Agent with the required skill is busy. When the Agent becomes idle, the packet is routed to him. His job is to service the packet for a random period dictated by a random distribution with a mean value depending on the Agent’s Experience. The Agents are paid on hourly basis and their scheduling is done based on the hourly traffic intensity.The Call Centre is shown in Figure 23, the Virtual Response Unit is shown in Figure 24 and the Queue is shown in Figure 25.

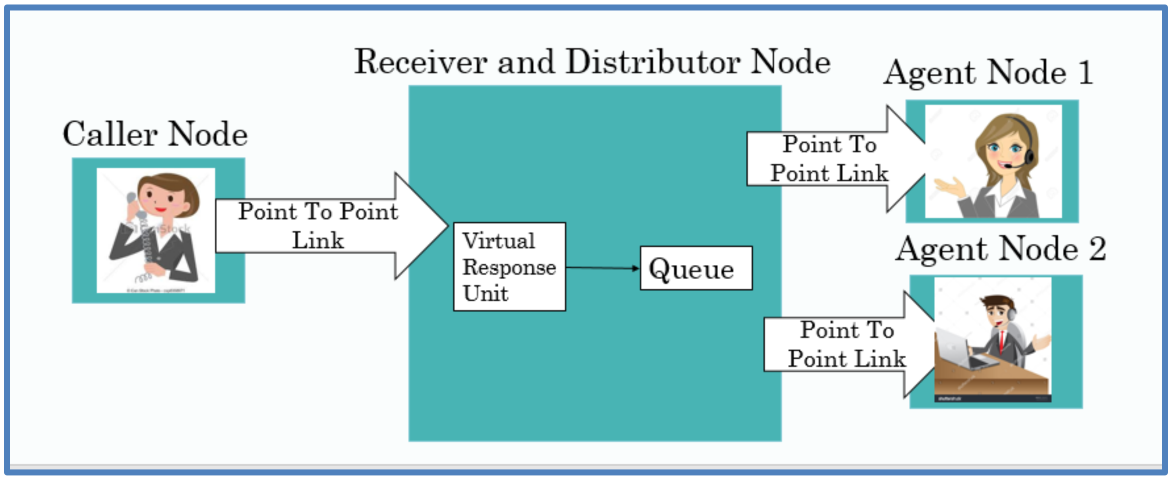


FIGURE 23: CALL CENTRE

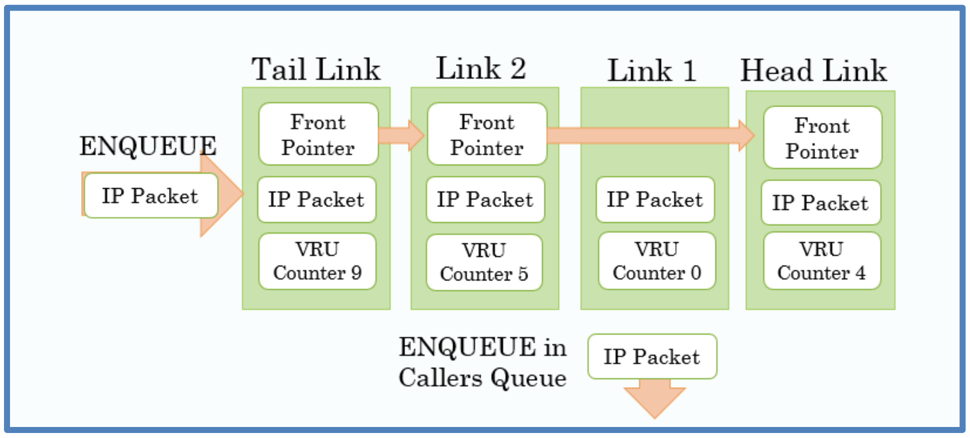


FIGURE 24: VIRTUAL RESPONSE UNIT

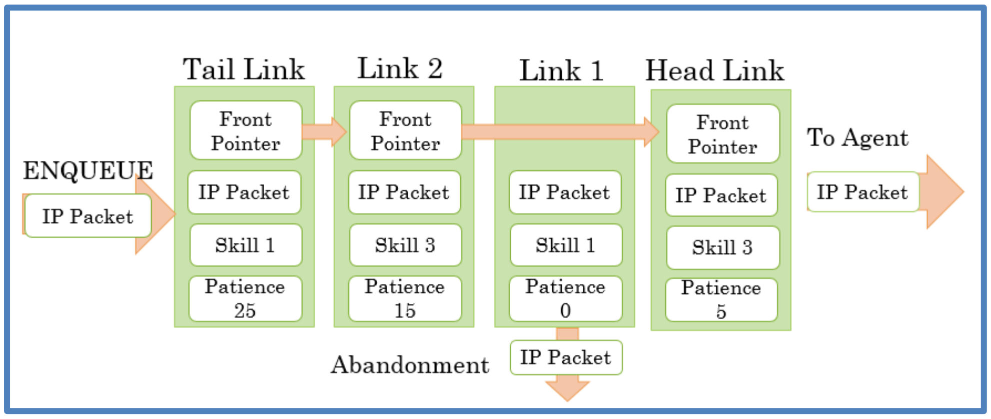


FIGURE 25: QUEUE

In the following section, we discuss the Simulation Methodology used to make the Model work as planned.

4.3 Simulation Methodology

The technique that we used was that using the Agent profiles extracted earlier, we programmed the agents to have similar properties as the real agents. Then we generated traffic similar to the actual traffic and kept a log of all the calls in the simulation including: Packet Size, skill, VRU Time, Patience, Id, EnqueueTime, Dequeue Time, Drop Time, Abandonment Time, Agent Id, Agent Wage, Service Time and used this data to calculate important performance metrics of the call center and to compare results with real data.

We made a primary Virtual Response Unit Linked List that comes before Callers Queue. For that, we added a Virtual Response Unit counter in each Link. We decrement Virtual Response Unit Counter of each Link after every second and if Virtual Response Unit Timer hits zero, we remove the Link from Virtual Response Unit Queue and Enqueue in Callers Queue. Before Enqueue, we check if queue is full. If queue is full, packet is dropped.

```
//Whenever a packet is received at receiver node, we Enqueue it in VRU Queue. Each Link has a
Packet Pointer, VRU Timer, EnqueueTime, Front and Back Link Pointers.
void VRUEnqueue (Ptr<Packet> packet, uint32_t VRUTime)
{…
 Link* p              =new Link;
 p->VRUtime        =VRUTime;
 p->pkt               =packet;
 p->EnqueueTime =Time (Seconds (Simulator: Now (). Get Seconds ()));
 Link * t              =Tail->Front;
 Tail->Front         =p;
 p->Front             =t;
 t->Back              =p;
 p->Back              =Tail;
 NumberOfPacketsInQueue ++;
 Total Packets       ++;
}
```

//This Function is called every second. It decrements the VRU counter of each Link in VRU Queue. If the timer of any Link is zero, the packet in that link is removed and we Enqueue it in Caller's Queue.

```
void VRUCountDown (Distributor Queue * queue)
{…
        if (NumberOfPacketsInQueue>0)
         {
                Link* a =Head;
                while (a! =Tail)
                {
                        a =a->Back;
                        a ->VRUtime= ((a->VRUtime)-1);
                         if (a->VRUtime==0)
                         {
                                Link* b         =a->Back;
                                Link* c         =a->Front;
                                b->Front        =c;
                                c->Back         =b;
                                NumberOfPacketsInQueue--;
                                queue->Enqueue (a->pkt);
                        }
                   }
        }
}
```

Each Agent has his own distribution of service times and skills.

//SocketRecv is called whenever Agent Receives a Packet for service. This Function generates a random number for Operator Efficiency. Service time duration is average of Packet Size and Operator Efficiency. Agent remains busy for that duration and calls ServiceComplete function after that time to become idle again.

```
void SocketRecv (Ptr<Socket> socket)
{…
MyPacket     =MySocket->Recv ();
PacketSize   =MyPacket->GetSize ();
OP_Efficiency = EfficiencyGenerator->GetInteger ();
Servicing [AgentNumber] =1;
Time         tnext (Seconds (static_cast<double> ((PacketSize+OP_Efficiency)/2)));
SendEvent    =Simulator: Schedule (tnext, &AgentApp: ServiceComplete, this);
}
```

```
void ServiceComplete ()
{…
 idleArray [AgentNumber] =1;
 Servicing [AgentNumber] =0;
 Packets Served         ++;
 MySocket               ->SetRecvCallback (MakeCallback (&AgentApp: SocketRecv, this));
 Total Wages+=WagePerCall;
}
```

A Caller Node keeps generating IP packets and sends them to the distributor node using a Point-to-Point Link. The Packet length follows a fixed distribution and has a fixed mean value. The Call Skill is denoted by a random variable and is stored in the TOS Tag of the IP Packet. The Caller Patience random variable is stored in the TTL Tag of the IP Packet. This is implemented in NS3 by a function that generates a random variable for Call length, Call Skill and Caller Patience, makes an IP packet of that size, adds the IP Tags and then sends it using a socket. The function then makes a self-call by scheduling the next call after a time interval given by a random number following a negative exponential distribution.

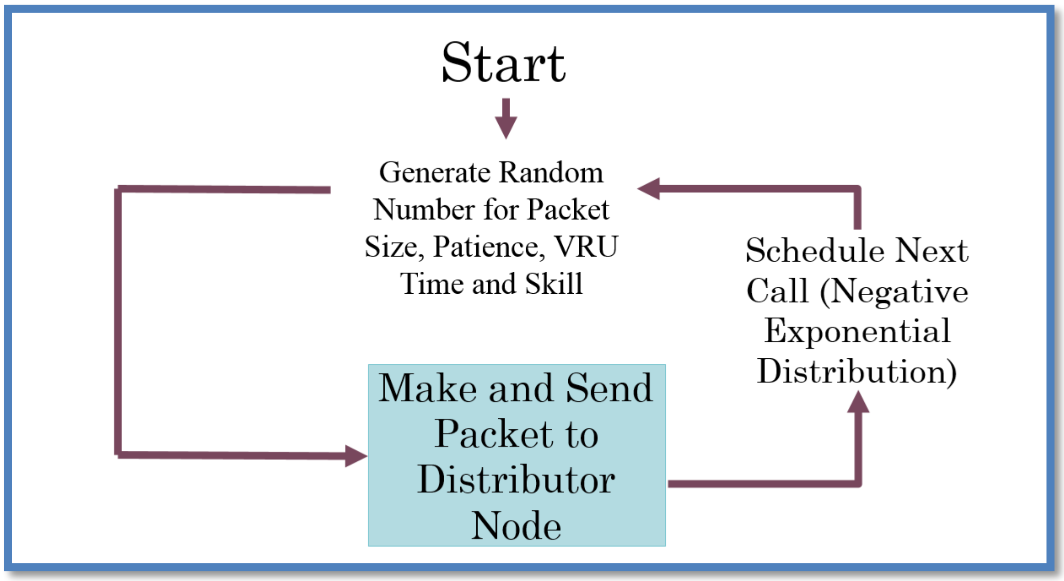
The Distributor Node receives the IP packets and keeps adding them to the tail of the VRU Linked List along with a VRU Timer Number, which is decremented every second. Once VRU counter becomes zero, packet is Enqueued in Callers Queue if the Queue is not completely full. If Queue is full, call is dropped. Caller can either redial or else call is lost. It also checks every second if any Agents are idle and distributes the Packets from the Queue to them. It traverses the entire Queue and checks if the idle Agents have the skill required by the Packet. In NS3, it schedules a call to SendPacket function every second, which does the routing of calls using Point to Point Links to Agents with required skill.

The Agent Node receive the IP packet from the Distributor Node and becomes busy for the duration of the call. Hence, it changes the Boolean idle flag to zero. It schedules a call to ServiceComplete function after a period equal to the length of the call after which it changes its status to idle again. Agents are scheduled and leave service based on schedules.

Now we present the Caller, Distributor and Agent Node Applications designed for this Final Call Center Model framework.

Caller Node Application

The Caller Agent Node Application is shown in Figure 26.



Start
Generate Random Number for Packet Size, Patience, VRU Time and Skill
Schedule Next Call (Negative Exponential Distribution)
Make and Send Packet to Distributor Node

FIGURE 26: CALLER NODE APPLICATION

```
void SendPacket ()
{…
        Size            =PacketSizeGenerator        ->GetInteger ();
        Waitingtime =WaitingTimeGenerator      ->GetInteger ();
        Skill           =SkillTypeGenerator          ->GetInteger ();
        Time           =InterArrivalTimeGenerator->GetInteger ();   //Generate Random Numbers

        Socket->SetIpTos (skill);                                    //Set Packet Skill Tag
        Socket->SetIpTtl (waitingtime);                        /Set Packet Waiting Time Tag

        Ptr<Packet> packet=Create<Packet> (size);                         //Make Packet

        Socket->Send (packet);                                                    //Send Packet

        Time tnext (Seconds (static_cast<double> (time)));                 //Schedule self-call
        SendEvent=Simulator: Schedule (tnext, &CallerApp: SendPacket, this);
}
```

Distributor Node Application

The Distributor Node Application is shown in Figure 27.

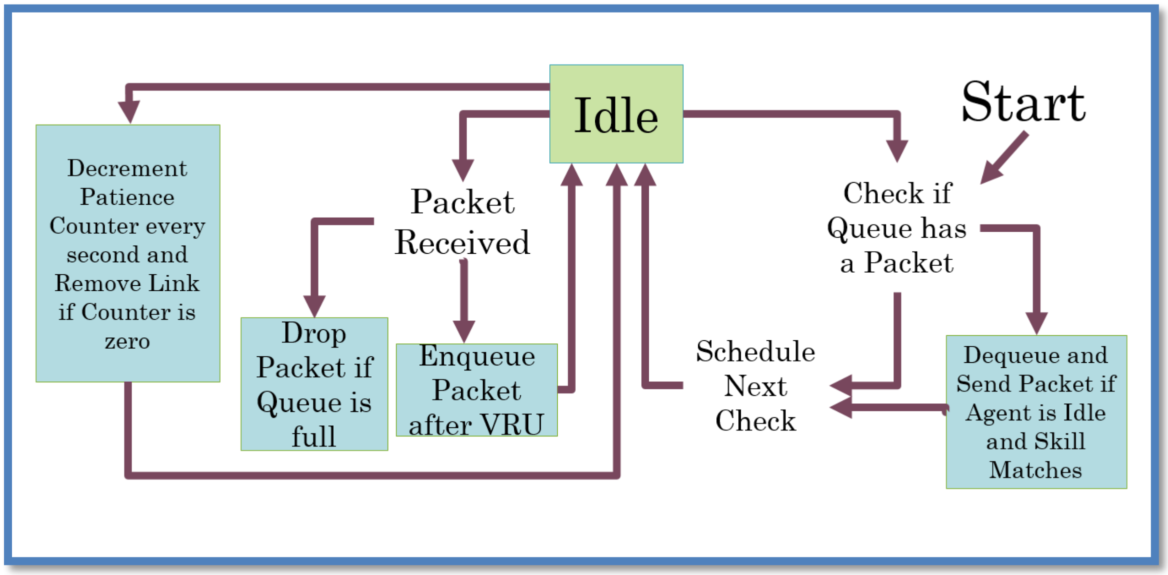

Idle
Start
Decrement Patience Counter every second and Remove Link if Counter is zero
Packet Received
Drop Packet if Queue is full
Enqueue Packet after VRU
Check if Queue has a Packet
Schedule Next Check
Dequeue and Send Packet if Agent is Idle and Skill Matches

FIGURE 27: DISTRIBUTOR NODE APPLICATION

```
void SkillBasedRouting ()
{…
        If (idle [PeerNumber] && (queue->NumberOfPacketsInQueue>0))               //Checks
        {
                DistributorQueue::Link* a=queue->Head->Back;
                 If ((SkillArray [PeerNumber] & a->Skill)>0)                      //Match Skill
                 MySocket              ->Send (a->Packet);                        //Send Packet
                  Time               tnext (Seconds (1));                   //Schedule self-call
         SendEvent =Simulator::Schedule (tnext, &DistributorApp: SkillBasedRouting, this);
        }
}
```

```
void SocketRecv (Ptr<Socket> socket)
{…
        Address from;
         Ptr<Packet> packet=MySocket->RecvFrom (from);                   //Receive Packet
         Queue->Enqueue (packet);                                         //Enqueue Packet
        MySocket->SetRecvCallback (MakeCallback (&ReceiverApp: SocketRecv, this));
}
```

Agent Node Application

The Agent Node Application is shown in Figure 28.

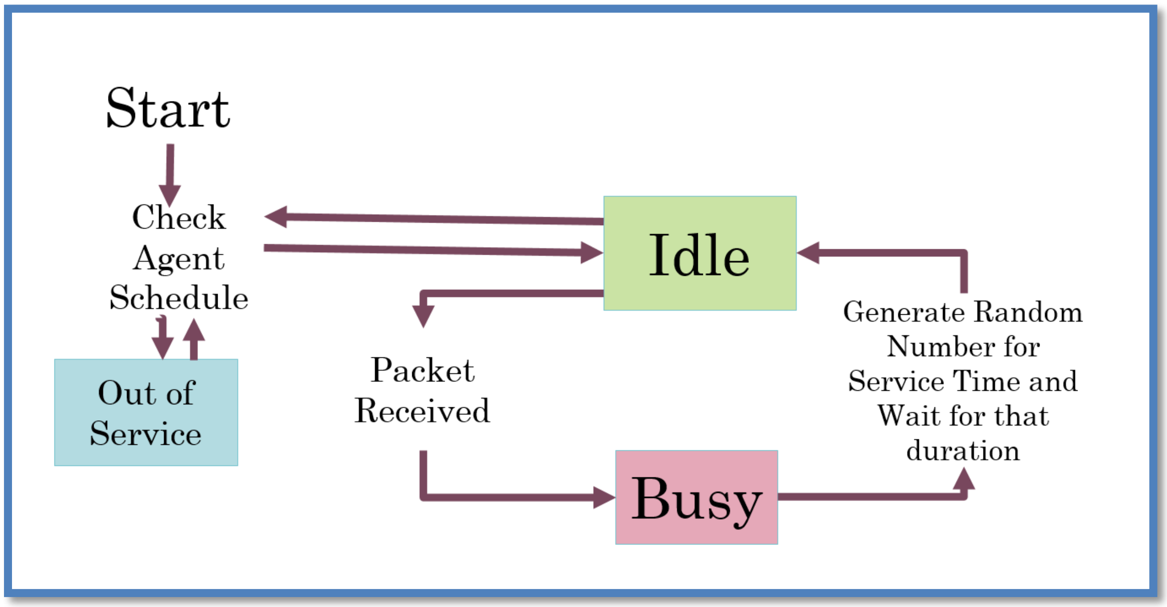

Start
Check Agent Schedule
Idle
Out of Service
Packet Received
Generate Random Number for Service Time and Wait for that duration
Busy

FIGURE 28: AGENT NODE APPLICATION

```
void SocketRecv (Ptr<Socket> socket)
{…
 MyPacket      =MySocket->Recv ();                                   //Receive Packet
 PacketSize     =MyPacket->GetSize ();
  OP_Efficiency = EfficiencyGenerator->GetInteger ();       //Generate random number for service
                                                                                 time
 Time          tnext (Seconds (static_cast<double> ((PacketSize+OP_Efficiency)/2)));
   SendEvent      =Simulator: Schedule (tnext, &AgentApp: ServiceComplete, this);//Schedule call
                              to ServiceComplete Function after Service Time is over
 IdleArray [AgentNumber] =0;                                                       //Busy
 }

void ServiceComplete ()
{…
 IdleArray [AgentNumber] =1;                                                        //Idle
MySocket ->SetRecvCallback (MakeCallback (&AgentApp: SocketRecv, this));
                                   //Call SocketRecv Function when call arrives
}
```

Finally, we discuss the simulation results and compare them with real Call Center results.

4.4 Simulation Results and Real Call Center Results

The technique used was that we generated similar call traffic and calculated the mean queue waiting time for each hour. Then we repeated these 10 times to obtain a 90% confidence interval and mean of distribution for a particular hour. We plotted the results against Real Call Center results. The Caller Traffic is shown in Figure 29. The Mean Queue Waiting Times for Simulation and Real Call Center are shown in Figure 30.

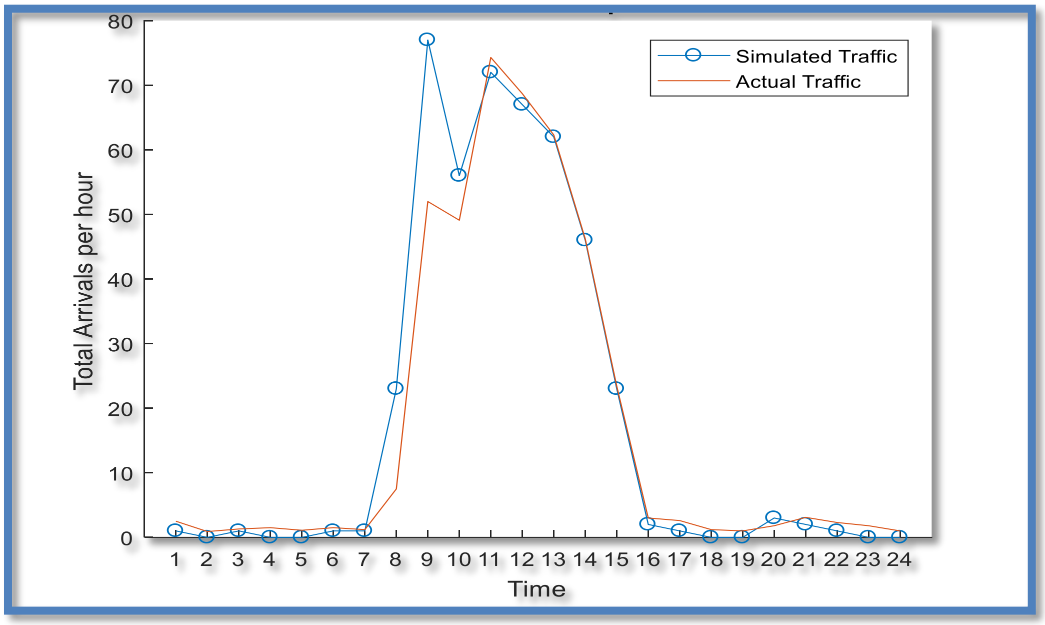


FIGURE 29: NUMBER OF ARRIVALS PER HOUR

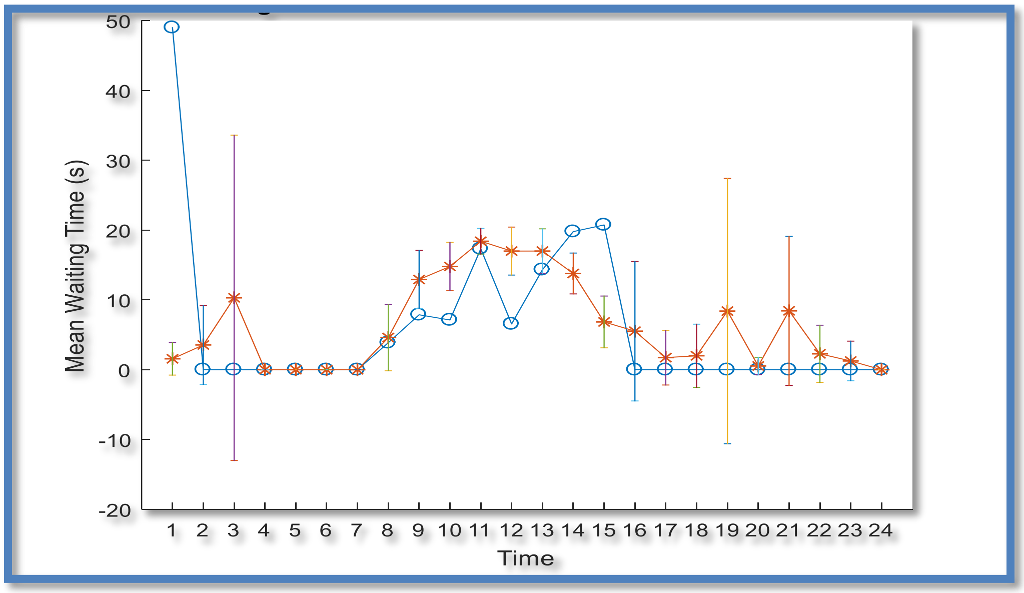


FIGURE 30: AVERAGE WAITING TIME: ACTUAL DATA VS SIMULATED RESULTS

Two Sample Simulator Runs are shown in Figures 31 and 32.

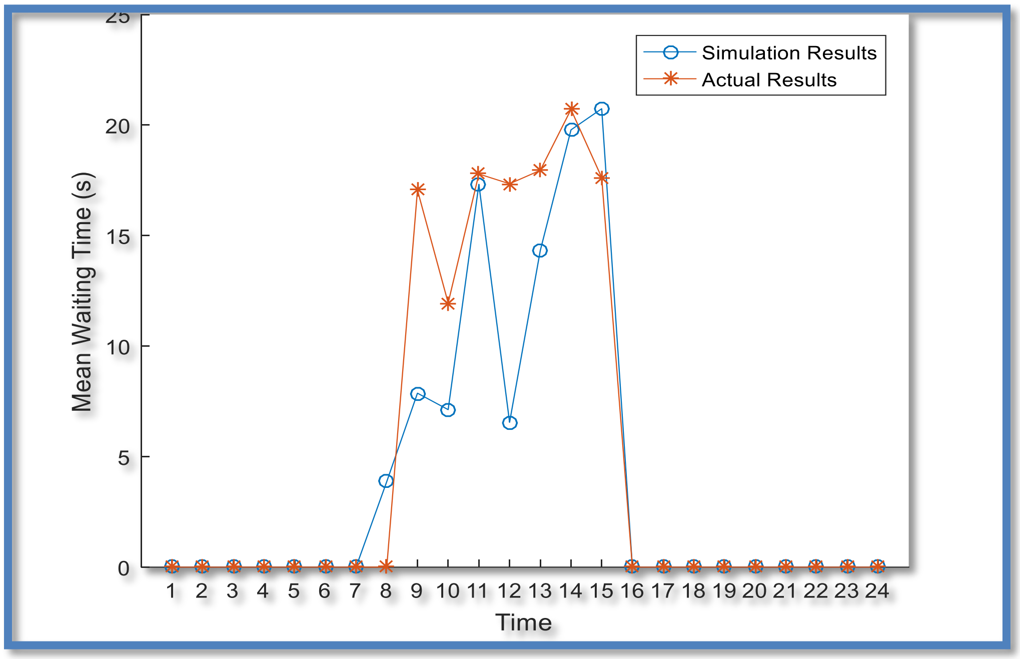


FIGURE 31: AVERAGE WAIT TIME: ACTUAL DATA VS SIMULATED RESULTS

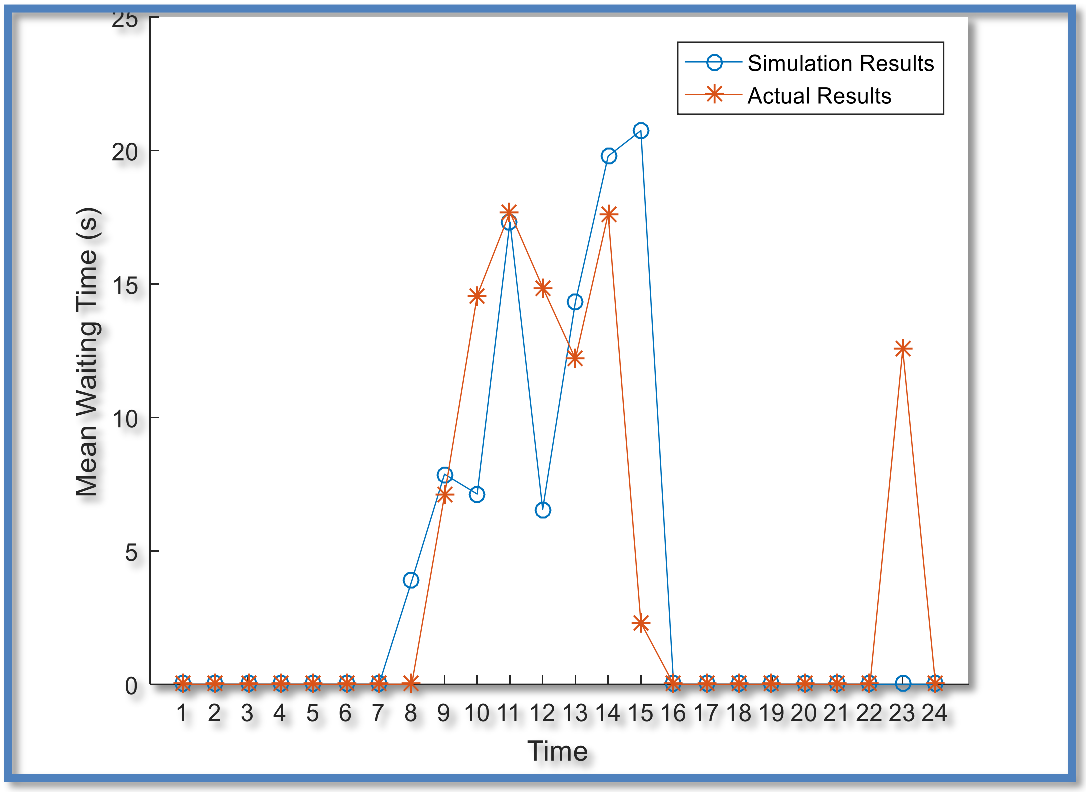


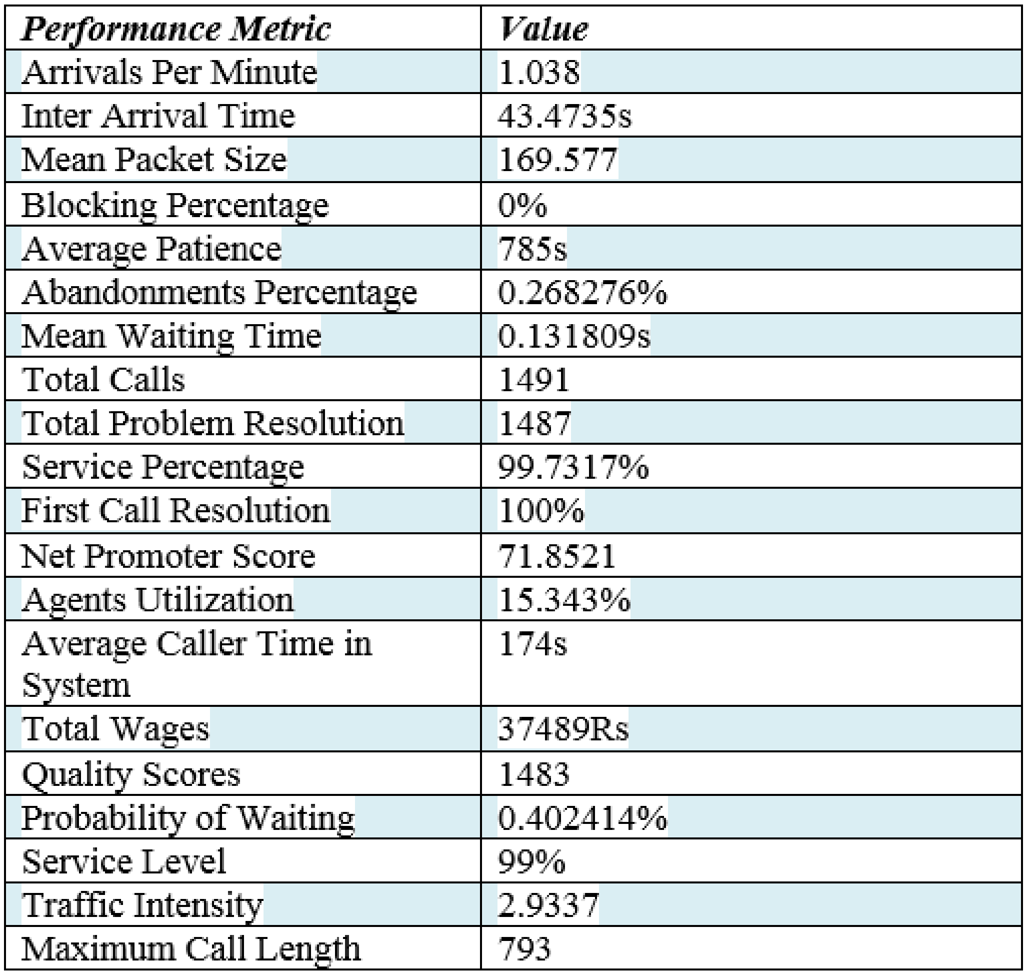
FIGURE 32: AVERAGE WAITING TIME: ACTUAL DATA VS SIMULATED RESULTS

Data Analysis

In addition to the Observations taken above, we kept a log of all the calls in the simulation including: Packet Size, skill, VRU Time, Patience, Id, EnqueueTime, Dequeue Time, Drop Time, Abandonment Time, Agent Id, Agent Wage, Service Time and used this data to calculate important performance metrics of the call center:

1. **Service level of the call**: it is the percentage of calls answered in less than specific time limit.
2. **Total Problem Resolution**: This was found by calculating the number calls resolved initially to those of the incoming calls that is the percentage of calls resolved in the first attempt.
3. **Agent Salaries** (hourly basis) depending on their performance
4. **Net Promoter Score**: this was calculated in response to every call, so was used to determine the performance of our call center system overall. Under this system every call received resulted in an increment in the NPS by 10, if dropped contributed nothing to it, if abandoned resulted in an increment by 10 and if was answered the NPS was further incremented by 10.
5. **Quality Scores**: these were agent rating done by analyzing and calculating the speed of answer of each agent and also the frequency of calls a specific agent answers in a given period of time.
6. **Profits**: this was calculated by finding the ratio of the calls received in a given interval of time to the agent's hourly salary.
7. **Agents utilization** (percentage of time the agents are busy)
8. **Average Caller Time in System** : sum of the queuing delay and the call time for each call.
9. **Probability of wait** : this was calculated by considering the arrival of calls when queue is not empty.
10. **Traffic Intensity** (Arrival rate / Service Rate)

The Calculated Performance metrics are shown in Table 4.



Performance Metric	***Value***
Arrivals Per Minute	1.038
Inter Arrival Time	43.4735s
Mean Packet Size	169.577
Blocking Percentage	0%
Average Patience	785s
Abandonments Percentage	0.268276%
Mean Waiting Time	0.131809s
Total Calls	1491
Total Problem Resolution	1487
Service Percentage	99.7317%
First Call Resolution	100%
Net Promoter Score	71.8521
Agents Utilization	15.343%
Average Caller Time in System	174s
Total Wages	37489Rs
Quality Scores	1483
Probability of Waiting	0.402414%
Service Level	99%
Traffic Intensity	2.9337
Maximum Call Length	793

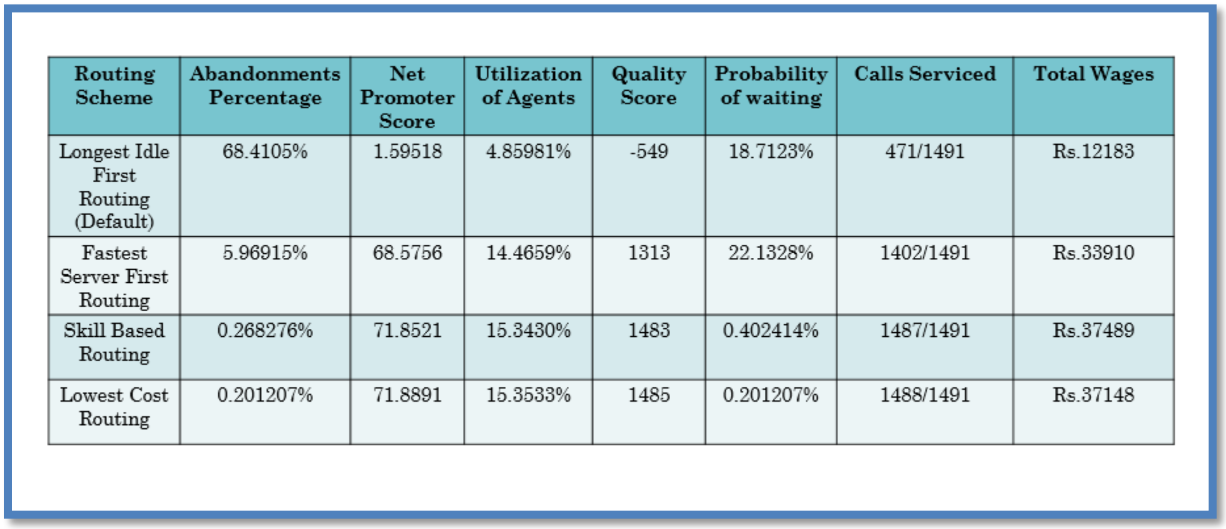
TABLE 4: CALL CENTER SIMULATION PERFORMANCE METRICS

Now that we had verified the call center model and we could guage the different performance metrics, we attempted to improve the performance by using four different routing schemes.

Various Routing Schemes

1. Fastest Server First (FSF): calls are routed to the idle agents having high speed.
2. Longest Idle First (LIF): calls are routed to the agents who are idle for longest time.
3. Lowest Cost Routing (LCR): calls are routed to the idle agents having lowest wage.
4. Skill Based Routing (SBR): calls are routed to the idle agents having required skills.

The schemes are compared in Table 5 and the results are plotted in Figure 33.



Routing Scheme	Abandonments Percentage	Net Promoter Score	Utilization of Agents	Quality Score	Probability of waiting	Calls Serviced	Total Wages
Longest Idle First Routing (Default)	68.4105%	1.59518	4.85981%	-549	18.7123%	471/1491	Rs.12183
Fastest Server First Routing	5.96915%	68.5756	14.4659%	1313	22.1328%	1402/1491	Rs.33910
Skill Based Routing	0.268276%	71.8521	15.3430%	1483	0.402414%	1487/1491	Rs.37489
Lowest Cost Routing	0.201207%	71.8891	15.3533%	1485	0.201207%	1488/1491	Rs.37148

TABLE 5: COMPARISON OF ROUTING SCHEMES

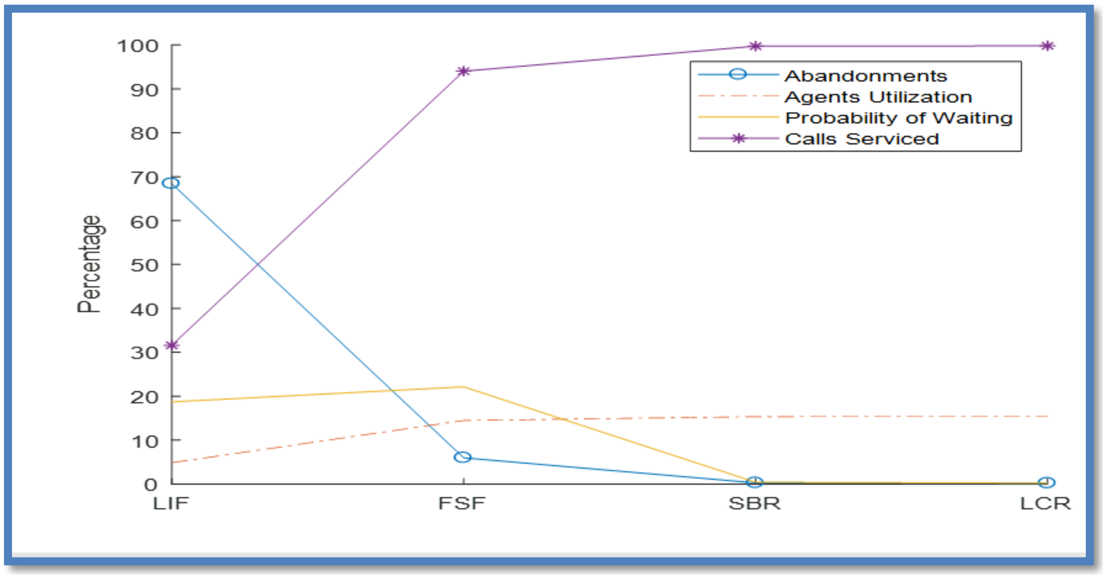


FIGURE 33: COMPARISON OF ROUTING SCHEMES

4.5 Relative Ranking of Design Options

After matching the results to that of the real call center, our main goal was to reduce the mean waiting time in the queue in order to increase the performance of the call center. For this, we used different routing schemes according to the changing calls arrival rates. The relative ranking of these schemes are as follows.

- Amongst four different routing schemes, the lowest Cost Routing (LCR) gave the best results. In this routing scheme, calls are routed to the idle agents having lowest wage. It is used when the calls arrival rate is highest. It reduces the cost of paying the agents thus the Cost of running the call center can be optimized by using this routing, since expenses are minimized. LCR not only reduces the abandonment percentage and probability of waiting but it increased the utilization of agents and the number of calls being served.
- The second-best Routing scheme is the skill based routing, which we were already using in our initial model. According to this routing method, calls are routed to the agent whose skills are matched to the caller requirements. It gave nicer results as abandonment percentage and probability of waiting was lower.
- Fastest Server First (FSF) is on the third number. According to this routing scheme, the calls are routed according to the agent's serving speed and it is used when the calls arrival rate is high so that more calls could be answered in lesser time. However, the abandonment percentage and probability of waiting was slightly higher than the other schemes.
- Longest Idle First (LIF) is the worst among all the routing schemes. It is utilized when the arrival rate is very low and calls are routes to the agent who has been idle, waiting for the call, for the longest time. It gave the worst results.

The following sections summarize our work and enlist upcoming challanges.

5 Progress

1. Verified Erlang C and Erlang X models.

2. Calculated Service Level (percentage of calls answered in less than specific time limit), Total Problem Resolution, Net Promoter score, Quality Scores, Agent Salaries (hourly basis), Profit, Utilization of Agents (percentage of time they are busy), Average Caller Time in System, Probability of wait (Arrivals when queue is not empty), Service Rate (Total Calls Serviced / Total Time), Traffic Intensity (Arrival rate * Service Rate) [1].

3. Plotted Queue size versus time.

4. Introduced Virtual Response Unit using negative exponential random generator, which waits for random time before Enqueue of arrived call.

5. Using call center history [9], made arrays of call durations, inter-arrival times, waiting times (VRU), waiting times (Service). Converted time format from hour: minute: second to seconds. Calculated mean service time of each agent. Plotted Service time versus time for one agent. Calculated mean arrival rate for each 10-minute interval and plotted queue size versus time.

6. Made a structure for Agent Nodes to model exhaustion level, experience, speed (dependent on exhaustion and experience), Mean Service Time, wage, time since last service, breaks.

7. Implemented Longest Idle First, Fastest Server First, skill based [14], lowest Cost and Longest Queue Wait First Routing schemes.

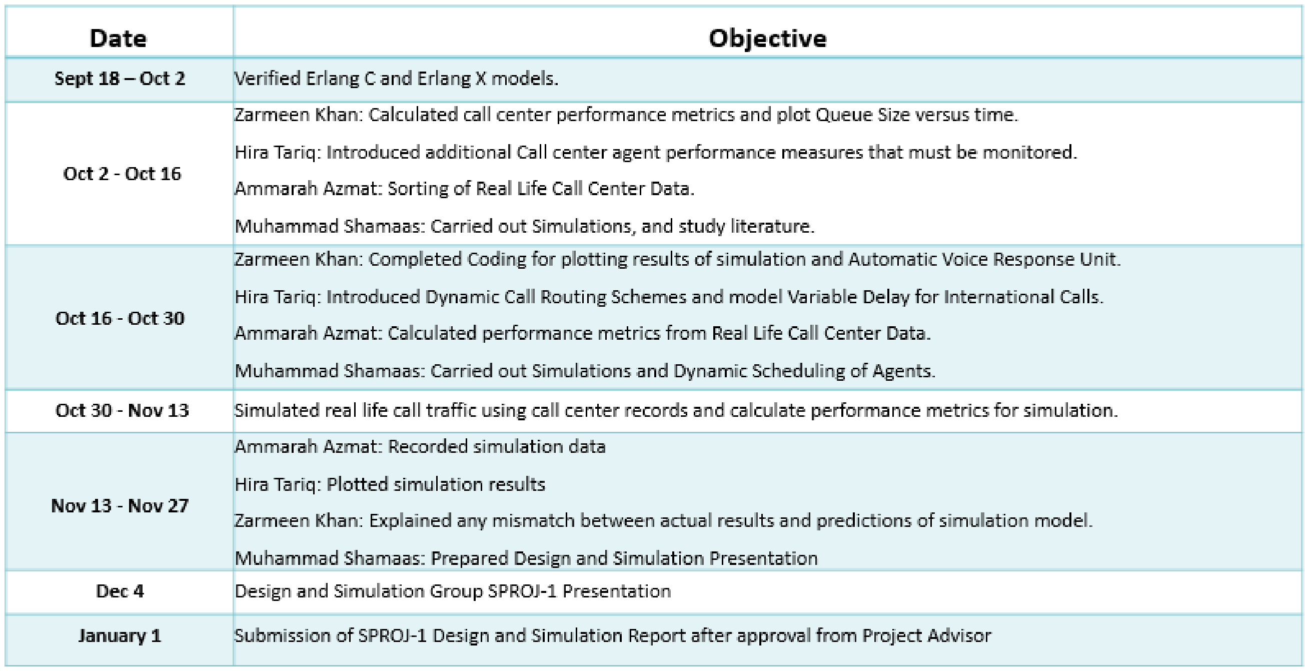
8. Introduced separate point-to-point helper for caller-distributor link. Modeled Delays for international calls for that point-to-point link using negative exponential random generator [10], [11].

9. Simulated real life call traffic using call center records and compared performance metrics for simulation and Real-Life Data. Recorded data and plotted results to explain any mismatch between actual results and predictions of simulation model.

10. Prepared Design and Simulation Presentation

6 Updated Timeline

The Updated Project Timeline is given in Table 6.



Date	Objective
Sept 18 – Oct 2	Verified Erlang C and Erlang X models.
Oct 2 - Oct 16	Zarmeen Khan: Calculated call center performance metrics and plot Queue Size versus time. Hira Tariq: Introduced additional Call center agent performance measures that must be monitored. Ammarah Azmat: Sorting of Real Life Call Center Data. Muhammad Shamaas: Carried out Simulations, and study literature.
Oct 16 - Oct 30	Zarmeen Khan: Completed Coding for plotting results of simulation and Automatic Voice Response Unit. Hira Tariq: Introduced Dynamic Call Routing Schemes and model Variable Delay for International Calls. Ammarah Azmat: Calculated performance metrics from Real Life Call Center Data. Muhammad Shamaas: Carried out Simulations and Dynamic Scheduling of Agents.
Oct 30 - Nov 13	Simulated real life call traffic using call center records and calculate performance metrics for simulation.
Nov 13 - Nov 27	Ammarah Azmat: Recorded simulation data Hira Tariq: Plotted simulation results Zarmeen Khan: Explained any mismatch between actual results and predictions of simulation model. Muhammad Shamaas: Prepared Design and Simulation Presentation
Dec 4	Design and Simulation Group SPROJ-1 Presentation
January 1	Submission of SPROJ-1 Design and Simulation Report after approval from Project Advisor

TABLE 6: PROJECT TIMELINE

7 Upcoming Challenges

1. We will research and set up a HTTPS Server in Node.js, which will listen for connection requests on the particular IP Address corresponding to Wireless Router of Wireless Area Network. It will receive voice calls from a mobile phone or computer that can connect to an internet browser and has voice input facility.

2. We will connect computers using Wired Local Area Network to create a network of connected Call Center Agents. They will communicate with each other through this network. It will also serve as a hardware database since the agents can write in the shared directory to keep a time log of special events e.g. Call received, Call Enqueue, Call Dequeue, Service Started, Service Complete etc.

3. Agent and Caller Applications will be built in Socket.io to communicate caller voice to Agent Computer and Agent voice to Caller Device via a wireless duplex communication channel.

4. We will automate Caller Nodes to keep making Voice Calls of variable durations to the Call Center IP Address.

5. We will automate the HTTPS Server to receive and forward them to idle Agents continuously. It will also keep a log of Call Enqueue and Dequeue times in the hardware database built using the shared directory of Wired Local Area Network. It will also keep a record of which agents are idle and which are busy.

6. The Agent Nodes will remain busy for duration of the voice call and then become idle again. They will write Service Start and Service End Events in the shared directory log. A lock will be used to allow only one Agent to write at a time. They will communicate their idle or busy states to the HTTPS Server using socket connection.

7. We will vary call arrival rates; call duration times, Agent Numbers, Caller Numbers etc. This will allow us to measure important performance metrics about our call center. We will run the simulation until we get substantial data.

8. We can also add the facility of relaying text messages, pictures and files between callers and Agents.

8 Societal Relevance

The solution for optimizing call center performance metrics like mean Queue Waiting Time, Service Level, Total Problem Resolution and Agents Utilization by tuning Total Agents Employed, scheduling of agents and Dynamic Caller Routing scheme based on traffic intensity can be used to improve the real-life call center efficiency. Even though the behavior of a real call center is much more complex and the model simplifies the system by many assumptions about caller and agent behavior, we have tried to make the model more realistic by including variable call arrival rates, variable propagation delays, queue abandonments, call drops and inhomogeneous agents (with variable speed of service, hourly wages, experience and skills) [4], [13].

As a result, if we input the predicted traffic profile to the simulator, we can achieve the optimal number, performance ratings and schedules of the agents that a call center must hire to meet the desired performance metric goals. The accuracy, validity and applicability of the simulation predictions depends on how many iterations we carry out and how much detail we incorporate about the real call center into the simulation model.

The inherent simplicity and generic nature of the Queue System model lends itself for adaptation in many different environments besides call centers as well [5]. Many other queue networks like shopping malls cash counters, pick and drop services, hospitals, printer stations, billing stations, ticket booths, banks, industrial workforce management etc. can benefit from the model. The system can be used to model any queue system with one or more servers that involves time and/ or resource management after a few modifications.

Many Queuing systems like those that call centers and billing stations are concerned with minimizing waiting time in queue i.e. finding the optimal number of agents needed to achieve the desired service level. The simulation can be run multiple times with different number of agents employed and mean waiting time recorded until the desired result is obtained. For time management, we will implement the fastest server first routing schemes. Another application includes minimizing queue abandonments by Longest Queue Wait First Priority routing scheme. For efficient and maximum utilization of agents we will implement the Skill based routing and Longest Idle First Priority routing [14]. This is desired in banks, hospitals and international call centers where differentiated agents and services are provided hence agents specialized in a particular skill are needed.

9 References

[1] "Call Centre." *Wikipedia*, Wikimedia Foundation, 17 Sept. 2017, en.wikipedia.org/wiki/Call_centre. Accessed Sept. 2017.

[2] "Erlang C Calculator." *Call Center Optimization*, www.gerkoole.com/CCO/erlang-c.php. Accessed Sept. 2017.

[3] "Erlang X Calculator." *Call Center Optimization*, www.gerkoole.com/CCO/erlang-x.php. Accessed Sept. 2017.

[4] Gans, Noah, et al. *Telephone Call Centers: Tutorial, Review, and Research Prospects*. 2nd ed., vol. 5, Amsterdam, Netherlands, 2003.

[5] Iversen, Villy B. *Teletraffic Engineering*. Lyngby, Denmark, 2001.

[6] Jerry, et al. *Discrete-Event System Simulation*. 4th ed., Prentice Hall.

[7] Koole, Ger. *Call Center Mathematics*. Amsterdam, Netherlands, 2007.

[8] Keshav, Srinivasan. *Mathematical Foundations of Computer Networking*. 2012.

[9] Mandelbaum, A. Call Center Data. http://iew3.technion.ac.il/serveng/callcenterdata/index.html. 2002

[10] *Ns-3 Tutorial*. 2017, www.nsnam.org/docs/release/3.26/tutorial/ns-3-tutorial.pdf. Accessed Sept. 2017.

[11] *Ns-3 Documentation*, www.nsnam.org/docs/release/3.26/doxygen/index.html. Accessed Sept. 2017.

[12] Perros, Harry. *Computer Simulation Techniques: The definitive introduction!* Raleigh, North Carolina, 2009.

[13] Robbins, Thomas R. *Experience-Based Routing in Call Center Environments*. 2nd ed., vol. 7, Greenville, North Carolina, 2015.

[14] Robbins, Thomas R., et al. *Evaluating the Erlang C and Erlang A Models for Call Center Modeling Working Paper*. Greenville, North Carolina.

10 Appendix

Simulator

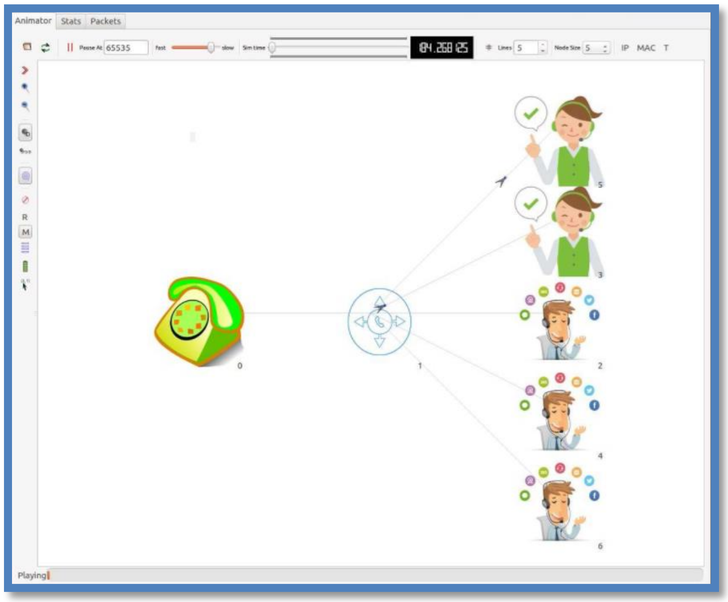
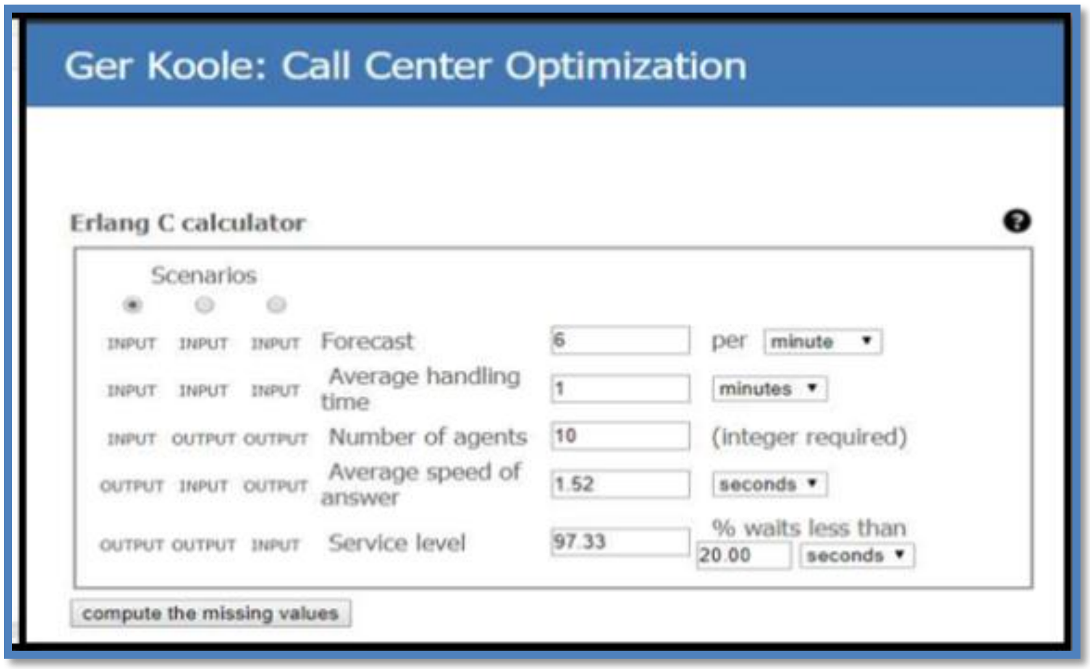


FIGURE 34: NETANIM SIMULATOR

Online Erlang C Calculator [2]



Ger Koole: Call Center Optimization

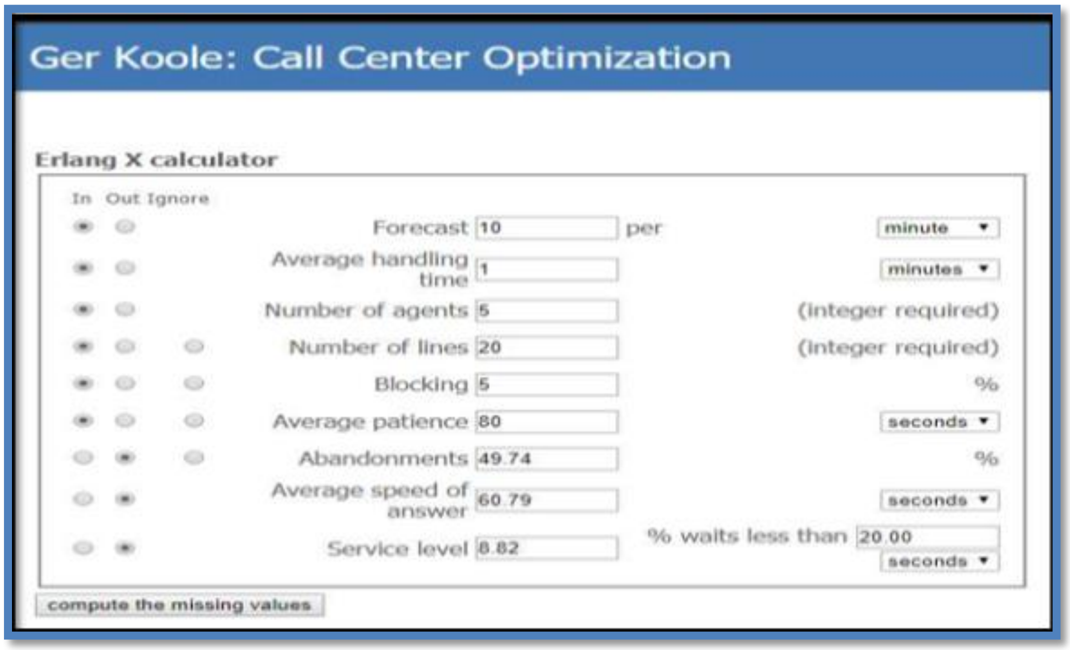
Erlang C calculator

Scenarios					
●	○	○			
INPUT	INPUT	INPUT	Forecast	6	per minute
INPUT	INPUT	INPUT	Average handling time	1	minutes
INPUT	OUTPUT	OUTPUT	Number of agents	10	(integer required)
OUTPUT	INPUT	OUTPUT	Average speed of answer	1.52	seconds
OUTPUT	OUTPUT	INPUT	Service level	97.33	% waits less than 20.00 seconds

compute the missing values

FIGURE 35: ONLINE ERLANG C CALCULATOR

Erlang X Calculator [3]



Ger Koole: Call Center Optimization

Erlang X calculator

In	Out	Ignore			
●	○		Forecast	10	per minute
●	○		Average handling time	1	minutes
●	○		Number of agents	5	(integer required)
●	○	○	Number of lines	20	(integer required)
●	○	○	Blocking	5	%
●	○	○	Average patience	80	seconds
○	●	○	Abandonments	49.74	%
○	●		Average speed of answer	60.79	seconds
○	●		Service level	8.82	% waits less than 20.00 seconds

compute the missing values

FIGURE 36: ONLINE ERLANG X CALCULATOR

Code

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/onoff-application.h"
#include "ns3/netanim-module.h"
#include "ns3/csma-module.h"
#include "ns3/animation-interface.h"
#include "ns3/ipv4-address-helper.h"
#include "ns3/internet-stack-helper.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/ipv4-global-routing.h"
#include "ns3/callback.h"
#include "ns3/point-to-point-helper.h"
#include "ns3/simulator.h"
#include <string>
#include <cstring>
#include <sstream>
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
using namespace ns3;
using namespace std;
ofstream myfile;


int Mean_InterArrival_Time        =5;
int Mean_Packet_Size              =50;
int Mean_Patience                 =60;
int number_of_Agent_nodes         =5;
int simtime                       =86500;
int randm                         =1;
int Maximum_Queue_Length          =20;
uint32_t WagePerCall              =100;
uint32_t AverageServiceTime       =45;
double precision                  =1;
uint32_t meanskill                =31;
uint32_t meanVRUTime              =10;
int RoutingType                   =4;//1-SBR, 2-FSF, 3-LIF, 4-LCR
int MaxCallsPerHour               =0;
uint32_t TotalWages               =0;
double MaxCallLength              =0;
int    **TABLE                    =new int   *  [25];
bool * idleArray                  =new bool    [number_of_Agent_nodes];
bool * Servicing                  =new bool    [number_of_Agent_nodes];
uint32_t * SkillArray             =new uint32_t[number_of_Agent_nodes];
uint32_t * WagesArray             =new uint32_t[number_of_Agent_nodes];
uint32_t * AvgServiceTimeArray  =new uint32_t[number_of_Agent_nodes];
uint32_t *TimeOfLastServiceArray=new uint32_t[number_of_Agent_nodes];
string **TEMP3                    =new string*[90000];
string **AVAIL_AGENT              =new string*[90000];
int    **AG_SER                   =new int   *[90000];
string *TEMP                      =new string [90000];
string *ENTRYTIME                 =new string [90000];
int    *VRU_ENTRY                 =new int    [90000];
string *ordered                   =new string [90000];
Ptr<Node> DistributorN;
```

```
class DistributorQueue{
public:
DistributorQueue(int m,AnimationInterface* anim)
{
 MyLink              =0;
 Head                =new Link;
 Tail                =new Link;
 TotalPackets        =0;
 Head->Back          =Tail;
 Tail->Back          =NULL;
 Head->Front         =NULL;
 Tail->Front         =Head;
 MaximumQueueLimit    =m;
 TotalPacketsDequeued  =0;
 NumberOfPacketsInQueue=0;
 MyAnimator=anim;
}


struct Link
{
 Link *      Back;
 Link *      Front;
 uint32_t    Data;
 uint32_t    Skill;
 Ptr<Packet> pkt;
 Time        EnqueueTime;
 uint32_t    QueueWaitingTime;
 uint32_t    VRUtime;
};


void Enqueue(Ptr<Packet> packet)
{
 if(NumberOfPacketsInQueue<MaximumQueueLimit)
  {
  uint32_t data          =packet->GetSize();
  SocketIpTosTag          skillTag;
  SocketIpTtlTag          waitingtimeTag;
  packet                 ->RemovePacketTag (skillTag);
  packet                 ->RemovePacketTag (waitingtimeTag);
  uint32_t skill         =(uint32_t)skillTag      .GetTos ();
  uint32_t waitingtime    =(uint32_t)waitingtimeTag.GetTtl ();
  packet                 ->AddPacketTag(skillTag);
  packet                 ->AddPacketTag(waitingtimeTag);
  Link* p                =new Link;
  p->Data                =data;
  p->Skill               =skill;
  p->QueueWaitingTime     =waitingtime;
  p->pkt                 =packet;
  p->EnqueueTime          =Time(Seconds(Simulator::Now().GetSeconds()));
  Link * t               =Tail->Front;
  Tail->Front            =p;
  p->Front               =t;
  t->Back                =p;
  p->Back                =Tail;
  NumberOfPacketsInQueue  ++;
  TotalPackets           ++;
  stringstream ss;
  ss<<(NumberOfPacketsInQueue);
  MyAnimator->UpdateNodeDescription(DistributorN,ss.str());

  /*myfile <<endl;
  myfile<<"Enqueue"<<endl;
  myfile<<"["<<endl;
  myfile<<"PacketNumber: "<<TotalPackets<<endl;
  myfile<<"NumberOfPacketsInQueue: "<<NumberOfPacketsInQueue<<endl;
  myfile<<"PacketSize: "<<data<<endl;
  myfile<<"Packet Skill: "<<skill<<endl;
  myfile<<"Packet Waiting Time: "<<waitingtime<<endl;
  myfile<<"EnqueueTime: "<<p->EnqueueTime<<endl;
  myfile<<"Time: "<<Time(Seconds(Simulator::Now().GetSeconds()))<<endl;
  myfile<<"]"<<endl;
  */

  myfile <<endl;
  myfile<<"Enqueue"<<endl;
  myfile<<"NumberOfPacketsInQueue:"<<NumberOfPacketsInQueue<<endl;
  myfile<<"Time:"<<Time(Seconds(Simulator::Now().GetSeconds()))<<endl;
  myfile<<"Packet Waiting Time: "<<waitingtime<<endl;
  myfile<<"EnqueueTime: "<<p->EnqueueTime<<endl;
  }
 else
  {
  myfile<<endl;
  myfile<<"Packet Dropped , MaximumQueueLimit Exceeded"<<endl;
  }
}


void CountDown()
{
 if(NumberOfPacketsInQueue>0)
 {
  Link* a           =Head;
  while(a!=Tail)
  {
   a                =a->Back;
   a                ->QueueWaitingTime=((a->QueueWaitingTime)-1);
   if(a->QueueWaitingTime==0)
   {
   Link* b          =a->Back;
   Link* c          =a->Front;
   b->Front         =c;
   c->Back          =b;
   NumberOfPacketsInQueue--;
   myfile <<endl;
   myfile<<"Queue Abandonement "<<endl;
   myfile<<"Packet Waiting Time: "<<Time(Seconds(Simulator::Now().GetSeconds()))-a->EnqueueTime<<endl;
   myfile<<"Time: "<<Time(Seconds(Simulator::Now().GetSeconds()))<<endl;
   myfile<<"]"<<endl;
   }
  }
 }
}


/*
 The code is very similar to Enqueue funcion.
 Make a Link, put the packet, EnqueueTime and VRUtime in it.
 Do not add Skill, Data or QueueWaitingTime of Link.
 Enqueue it at tail of VRUQueue.
 Even when there is no packet in Queue, the Head and Tail Links must be present.
*/
void VRUEnqueue(Ptr<Packet> packet,uint32_t VRUTime)
{
  Link* p              =new Link;
  p->VRUtime           =VRUTime;
  p->pkt                =packet;
  p->EnqueueTime        =Time(Seconds(Simulator::Now().GetSeconds()));
  Link * t              =Tail->Front;
  Tail->Front           =p;
  p->Front              =t;
  t->Back               =p;
  p->Back               =Tail;
  NumberOfPacketsInQueue  ++;
  TotalPackets          ++;
}



/*
 The function is very similar to Countdown function.
 Start from the Head Link of the VRU Queue and go Back towards the Tail Link.
 Decement VRUtime in every Link except the Head and Tail Links.
 If any Link has VRUtime=0, remove it's link from VRUQueue and enqueue it in Callers queue.
 The pointer for Caller's queue is the input of the VRUCountDown Function, DistributorQueue * queue.
*/
void VRUCountDown(DistributorQueue * queue)
{
                if(NumberOfPacketsInQueue>0)
 {
  Link* a           =Head;
  while(a!=Tail)
   {
     a               =a->Back;
     a               ->VRUtime=((a->VRUtime)-1);
     if(a->VRUtime==0)
     {
        Link* b          =a->Back;
        Link* c          =a->Front;
        b->Front         =c;
        c->Back          =b;
        NumberOfPacketsInQueue--;
        queue->Enqueue(a->pkt);
      }

    }
 }
}


Ptr<Packet> Dequeue()
{
 if(NumberOfPacketsInQueue>0)
  {
  Link* p    =Head      ->Back;
  Link* q    =p         ->Back;
  Head       ->Back     =q;
  q          ->Front    =Head;
  NumberOfPacketsInQueue--;
  TotalPacketsDequeued  ++;
  myfile <<endl;
  myfile<<"Packet Removed from Queue"<<endl;
  myfile<<"Packet Address:"<<(((p->Data)<<16)+((p->Skill)<<8)+((p->QueueWaitingTime)))<<endl;
  myfile<<"PacketEnqueueTime: "<<p->EnqueueTime<<endl;
  myfile<<"Time: "<<Time(Seconds(Simulator::Now().GetSeconds()))<<endl;
  myfile<<"]"<<endl;
  stringstream ss;
  ss<<(NumberOfPacketsInQueue);
  MyAnimator->UpdateNodeDescription(DistributorN,ss.str());
  return p->pkt;
  }
 else
  {return 0;}
}


Link * Head;
Link * Tail;
Link * MyLink;
AnimationInterface* MyAnimator;
int TotalPackets,NumberOfPacketsInQueue,TotalPacketsDequeued,MaximumQueueLimit;

};









class CallerApp:public Application{
public:
CallerApp()
{
 socket                 =0;
 peer                   =Address();
 TotalPacketsSent       =0;
 PacketSizeGenerator     =CreateObject<ExponentialRandomVariable>();
 SkillTypeGenerator      =CreateObject<ExponentialRandomVariable>();
 WaitingTimeGenerator    =CreateObject<ExponentialRandomVariable>();
 InterArrivalTimeGenerator=CreateObject<ExponentialRandomVariable>();
}


void Setup(Ptr<Socket> mysocket, Address address,uint32_t meantime, uint32_t meansize, uint32_t meanwaitingtime, uint32_t
minskill, uint32_t boundskill)
{
 InterArrivalTimeGenerator->SetAttribute("Mean",DoubleValue(meantime));
 PacketSizeGenerator      ->SetAttribute("Mean",DoubleValue(meansize));
 WaitingTimeGenerator     ->SetAttribute("Mean",DoubleValue(meanwaitingtime));
 SkillTypeGenerator       ->SetAttribute("Mean" ,DoubleValue(boundskill));
 //SkillTypeGenerator       ->SetAttribute("Max" ,DoubleValue(boundskill));
 peer       =address;
 socket     =mysocket;
 myboundskill=boundskill;
}


void StartApplication()
{
 socket->Bind();
 bool ipRecvTtl=true;
 bool ipRecvTos=true;
 socket       ->SetIpRecvTos (ipRecvTos);
 socket       ->SetIpRecvTtl (ipRecvTtl);
 socket       ->Connect(peer);
 SetHourlyPacketInterArrivalTime();
 SendPacket();
}
```

```
void StopApplication ()
{
    if (SendEvent.IsRunning ())
      {
        Simulator::Cancel (SendEvent);
      }
      if (SendEvent1.IsRunning ())
      {
        Simulator::Cancel (SendEvent1);
      }
      if (socket)
      {
        socket->Close ();
      }
}


void SetHourlyPacketInterArrivalTime()
{

 {InterArrivalTimeGenerator->SetAttribute("Mean",DoubleValue(3600/(1+TABLE[int(Simulator::Now().GetSeconds())/3600][0])));}


 if((int((Simulator::Now().GetSeconds()))+3600)<(simtime-10))
 {
                cout<<(int((Simulator::Now().GetSeconds()))*100)/simtime<<" %"<<endl;
  Time      tnext(Seconds(3600));
  SendEvent1=Simulator::Schedule(tnext,&CallerApp::SetHourlyPacketInterArrivalTime,this);

 //cout<<(TABLE[int(Simulator::Now().GetSeconds())/3600][0])/11<<endl;
  for (int i = (TABLE[int(Simulator::Now().GetSeconds())/3600][0])/(MaxCallsPerHour/number_of_Agent_nodes); i >=0; i--)
  {
                if(!Servicing[i])
                {idleArray[i]=1;}
  }

  for (int i = (TABLE[int(Simulator::Now().GetSeconds())/3600][0])/(MaxCallsPerHour/number_of_Agent_nodes); i <
number_of_Agent_nodes; i++)
  {
                idleArray[i]=0;
  }
 }
}


void SendPacket()
{
 time=1;
 if(TABLE[int(Simulator::Now().GetSeconds())/3600][0]>0)
 {for (int i = 0; i < randm; i++)
 {size     =PacketSizeGenerator     ->GetInteger();
 waitingtime=WaitingTimeGenerator    ->GetInteger();
 skill     =SkillTypeGenerator      ->GetInteger();
 time      =InterArrivalTimeGenerator->GetInteger();}

 size=0;
 time=0;
 skill=0;
 waitingtime=0;

 while((size<=0))                    {size       =PacketSizeGenerator     ->GetInteger();}
 while((waitingtime<=0) )            {waitingtime =WaitingTimeGenerator    ->GetInteger();}
 while((skill<=0) | (skill>myboundskill)){skill      =SkillTypeGenerator      ->GetInteger();}
 while(time<=0)                      {time       =InterArrivalTimeGenerator->GetInteger();}
 socket->SetIpTos(skill);
 socket->SetIpTtl(waitingtime);

 Ptr<Packet> packet=Create<Packet>(size);

 socket->Send(packet);
 TotalPacketsSent++;
 myfile <<endl;
 myfile<<"Call Sent"<<endl;
 myfile<<"PacketSize:"<<size<<endl;
 myfile<<"InterArrivalTime:"<<time<<endl;
 myfile<<"Packet Waiting Time:"<<waitingtime<<endl;
 myfile<<"Time: "<<Time(Seconds(Simulator::Now().GetSeconds()))<<endl;
 myfile<<"Skill: "<<skill<<endl;
 myfile<<"]"<<endl;
 }

 if((int((Simulator::Now().GetSeconds()))+int(time))<(simtime-10))
 {
 Time tnext(Seconds(static_cast<double>(time)));
 SendEvent=Simulator::Schedule(tnext,&CallerApp::SendPacket,this);
 }


}

Address             peer;
EventId             SendEvent;
EventId             SendEvent1;
Ptr<Socket>         socket;
int                 TotalPacketsSent;
Ptr<ExponentialRandomVariable> PacketSizeGenerator;
Ptr<ExponentialRandomVariable> SkillTypeGenerator;
Ptr<ExponentialRandomVariable> WaitingTimeGenerator;
Ptr<ExponentialRandomVariable> InterArrivalTimeGenerator;
uint32_t time,size,waitingtime,skill,myboundsize,myboundwaitingtime,myboundskill;

};

/*....Random Generator for the Agent's efficiency....
-Shoud be -ve exponential random variable with some mean and interarrival rate.......// Check the distributin from Ammara

## IN Function Main::
Ptr<ExponentialRandomVariable> EfficiencyGenerator;

## in DEfault Constructor
EfficiencyGenerator ->SetAttribute("Mean",DoubleValue(meansize));

##
*/









//
class ReceiverApp: public Application{
public:
ReceiverApp():peer(),MySocket(0)
```

```
{
 VRUTimeGenerator     =CreateObject<ExponentialRandomVariable>();
}


void Setup(Ptr<Socket> socket,Address Agentaddress,DistributorQueue* myqueue,int mypeernumber,uint32_t
meanwaitingtime,DistributorQueue* VRUqueue)
{
 queue      =myqueue;
 VRUQueue   =VRUqueue;
 MySocket   =socket;
 peer       =Agentaddress;
 PeerNumber =mypeernumber;
 VRUTimeGenerator     ->SetAttribute("Mean",DoubleValue(meanwaitingtime));
}


void StartApplication()
{
 MySocket->Bind();

 bool ipRecvTtl=true;
 bool ipRecvTos=true;


 MySocket     ->SetIpRecvTos (ipRecvTos);
 MySocket     ->SetIpRecvTtl (ipRecvTtl);
 MySocket     ->Connect(peer);

 MySocket->    SetRecvCallback(MakeCallback(&ReceiverApp::SocketRecv,this));
 if((int((Simulator::Now().GetSeconds()))+1)<(simtime-10)){
 Time          tnext(Seconds(1));
 Simulator::Schedule(tnext,&ReceiverApp::CountDown,this);
 }
}

 void StopApplication ()
 {
    if (SendEvent.IsRunning ())
      {
        Simulator::Cancel (SendEvent);
      }
      if (MySocket)
      {
        MySocket->Close ();
      }
 }

void SocketRecv(Ptr<Socket> socket)
{

 for (int i = 0; i < randm; i++)
 {
 VRUtime=VRUTimeGenerator->GetInteger();
 }

 VRUtime = 0;

 while((VRUtime<=0)){VRUtime =VRUTimeGenerator->GetInteger();}
 Address from;
 Ptr<Packet> packet=MySocket->RecvFrom(from);

 VRUQueue->VRUEnqueue(packet,VRUtime);
 MySocket->SetRecvCallback(MakeCallback(&ReceiverApp::SocketRecv,this));
}
```

```
void CountDown()
{
 queue->    CountDown();
 VRUQueue-> VRUCountDown(queue);
 if((int((Simulator::Now().GetSeconds()))+1)<(simtime-10)){
 Time       tnext(Seconds(1));
 SendEvent=Simulator::Schedule(tnext,&ReceiverApp::CountDown,this);
}
}


Address          peer;
int              PeerNumber;
EventId          SendEvent;
uint32_t         PacketSize;
Ptr<Packet>      MyPacket;
Ptr<Socket>      MySocket;
Ptr<ExponentialRandomVariable> VRUTimeGenerator;
DistributorQueue* queue;
DistributorQueue* VRUQueue;
uint32_t VRUtime;
};








class DistributorApp: public Application{
public:
DistributorApp():peer(),dataRate(0),MySocket(0){}

void Setup(Ptr<Socket> socket, DataRate datarate,Address Agentaddress,uint32_t* mySkillArray,DistributorQueue* myqueue,bool*
myidle,int mypeernumber,AnimationInterface * anim)
{
 idle       =myidle;
 queue      =myqueue;
 MySocket   =socket;
 dataRate   =datarate;
 peer       =Agentaddress;
 PeerNumber =mypeernumber;
 SkillArray =mySkillArray;
 MyAnimator = anim;
}


void StartApplication()
{
 MySocket->Bind();

 bool ipRecvTtl=true;
 bool ipRecvTos=true;

 MySocket     ->SetIpRecvTos (ipRecvTos);
 MySocket     ->SetIpRecvTtl (ipRecvTtl);
 MySocket     ->Connect(peer);

 if(RoutingType==1){
 if((int((Simulator::Now().GetSeconds()))+1)<(simtime-10))
 {
  Time      tnext(Seconds(1/precision));
  SendEvent=Simulator::Schedule(tnext,&DistributorApp::SkillBasedRouting,this);// call the send packet func after each ms and sends
the packets in a queue.
 }
 }

 if(RoutingType==2){
 if((int((Simulator::Now().GetSeconds()))+1)<(simtime-10))
 {
  Time      tnext(Seconds(1/precision));
  SendEvent=Simulator::Schedule(tnext,&DistributorApp::FSF,this);// call the send packet func after each ms and sends the packets
in a queue.
 }
 }

 if(RoutingType==3){
 if((int((Simulator::Now().GetSeconds()))+1)<(simtime-10))
 {
  Time      tnext(Seconds(1/precision));
  SendEvent=Simulator::Schedule(tnext,&DistributorApp::LIF,this);// call the send packet func after each ms and sends the packets in
a queue.
 }
 }

 if(RoutingType==4){
 if((int((Simulator::Now().GetSeconds()))+1)<(simtime-10))
 {
  Time      tnext(Seconds(1/precision));
  SendEvent=Simulator::Schedule(tnext,&DistributorApp::LCR,this);// call the send packet func after each ms and sends the packets
in a queue.
 }
 }
}


void StopApplication ()
{
    if (SendEvent.IsRunning ())
    {
      Simulator::Cancel (SendEvent);
    }
    if (MySocket)
      {
        MySocket->Close ();
      }
}


void SendPacket(int AgentNumber)
{
 MyPacket = queue->Dequeue();
 idle[AgentNumber] = 0;
 MySocket ->Send(MyPacket);
 stringstream ss;
 ss<<(queue->NumberOfPacketsInQueue);
 MyAnimator->UpdateNodeDescription(DistributorN,ss.str());
}



void swap(uint32_t* Arr,int a, int b)
{
                uint32_t  temp = Arr[a];
                Arr[a] = Arr[b];
```

```
                    Arr[b] = temp;
}


uint32_t * SortMax(uint32_t Array[], int n)
{
                    int i=0,j=0,max_index=0;
                    for (i = 0; i < n-1; i++)
                    {
                                        max_index = i;
                                        for (j = i+1; j < n; j++)
                                                            if (Array[j] > Array[max_index])
                                                                                max_index = j;
                                                            swap(Array,max_index,i);
                                        }
                                        return Array;
}


//Fastest Server First Routing Scheme...
void FSF()
{
 if(idle[PeerNumber]&&(queue->NumberOfPacketsInQueue>0))
 {
 bool IsIdle=0;
 uint32_t * MaxServiceArray = new uint32_t[number_of_Agent_nodes];
 uint32_t index;
 //maxServiceArray has the agent nodes indexes in the sorted form.
 MaxServiceArray =SortMax(AvgServiceTimeArray,number_of_Agent_nodes);
 for(int i=0; i<number_of_Agent_nodes;i++)
 {
 // index... AT which index of the maxServiceArray is the peer number. All //agents before that will have higher speed.
 if(MaxServiceArray[i] == AvgServiceTimeArray[PeerNumber])
 index = i;
 }
 // check if All AgentsFasterThan MyAgentAreBusy. Use idleArray
 for(int i = 0; i<int(index); i++)
 {
                    for (int j = 0; j < number_of_Agent_nodes; j++)
                    {
      if(MaxServiceArray[i]==AvgServiceTimeArray[j])
      {
                     if(idleArray[j])
                     {
                      IsIdle=1;
          i=int(index);
        }
      }
   }
 }
 // AllAgentsFasterThanMyAgentAreBusy, Dequeue Packet from queue and send it to My Agent.
 // send packet to peerNumber...ASK!!
 if(!IsIdle){
 SendPacket(PeerNumber);
 }
 }

 if((int((Simulator::Now().GetSeconds()))+1)<(simtime-10))
 {
 Time      tnext(Seconds(1/precision));
 SendEvent=Simulator::Schedule(tnext,&DistributorApp::FSF,this);
 }
}
```

```
//Longest Idle First...
void LIF()
{
 if(idle[PeerNumber]&&(queue->NumberOfPacketsInQueue>0))
 {
 uint32_t * MaxTimeOfLastServiceArray=new uint32_t[number_of_Agent_nodes];
 int index;
 // MaxTimeOfLastServiceArray has the agent nodes indexes in the sorted form.

 MaxTimeOfLastServiceArray=SortMax(TimeOfLastServiceArray,number_of_Agent_nodes);
 for(int i=0; i<number_of_Agent_nodes;i++)
 {
 if(MaxTimeOfLastServiceArray [i] == TimeOfLastServiceArray[PeerNumber])
 index = i;
 }

 // check if All AgentsHavingLargerWaitingTimeThan MyAgentAreBusy. Use

 //MySocket ->Send(MyPacket);

 if(index==(number_of_Agent_nodes-1))
 {SendPacket(PeerNumber);
 }
 }

 if((int((Simulator::Now().GetSeconds()))+1)<(simtime-10))
 {
 Time     tnext(Seconds(1/precision));
 SendEvent=Simulator::Schedule(tnext,&DistributorApp::LIF,this);
 }
}


// Lowest Cost First routing Scheme..
void LCR()
{

 if(idle[PeerNumber]&&(queue->NumberOfPacketsInQueue>0))
 {
 bool IsIdle=0;
 uint32_t * MaxCostArray = new uint32_t[number_of_Agent_nodes];
 int index=0;
 //maxServiceArray has the agent nodes indexes in the sorted form.
 MaxCostArray =SortMax(WagesArray,number_of_Agent_nodes);
 // index... AT which index of the maxServiceArray is the peer number. All //agents before that will have higher speed.
 for(int i=0; i<number_of_Agent_nodes;i++)
 {
 if(MaxCostArray[i] == WagesArray[PeerNumber])
 index = i;i=number_of_Agent_nodes;
 }
 // check if All Agents having lower cost Than MyAgentAreBusy. Use //idleArray
 for(int i = 0; i<int(index); i++)
 {
                 for (int j = 0; j < number_of_Agent_nodes; j++)
                 {
     if(MaxCostArray[i]==WagesArray[j])
      {
                  if(idleArray[j])
                  {
                   IsIdle=1;
         i=int(index);
        }
      }
   }
 }
 // AllAgentshaving lesser costThanMyAgentAreBusy, Dequeue Packet from //queue and send it to My Agent.
 // send packet to peerNumber...ASK!!
 if(!IsIdle){
 SendPacket(PeerNumber);
 }
 }

 if((int((Simulator::Now().GetSeconds()))+1)<(simtime-10))
 {
 Time     tnext(Seconds(1/precision));
 SendEvent=Simulator::Schedule(tnext,&DistributorApp::LCR,this);
 }
}


void SkillBasedRouting()
{
 // Fastest Service Firt...........

 // Skill based routing........
   if(idle[PeerNumber]&&(queue->NumberOfPacketsInQueue>0))
                 {
                  bool found=0;
                  DistributorQueue::Link* a=queue->Head->Back;
                  DistributorQueue::Link* b=0;
                  while(a!=queue->Tail)
                  {
                   if((SkillArray[PeerNumber] & a->Skill)>0)
                   {
                    found=1;
                    b    =a;
                    a    =queue->Tail->Front;
                   }
                   a=a->Back;
                  }
                  if(found)
                   {
      MyPacket=b->pkt;
                    myfile <<endl;
                    myfile<<"Packet Removed from Queue"<<endl;
                    myfile<<"Packet Address:"<<(((b->Data)<<16)+((b->Skill)<<8)+((b->QueueWaitingTime)))<<endl;
                    myfile<<"PacketEnqueueTime: "<<b->EnqueueTime<<endl;
                    myfile<<"Time: "<<Time(Seconds(Simulator::Now().GetSeconds()))<<endl;
                    myfile<<"]"<<endl;
                    queue->NumberOfPacketsInQueue=queue->NumberOfPacketsInQueue-1;
                    queue->TotalPacketsDequeued  =queue->TotalPacketsDequeued+1;
                    b->Back->Front             =b->Front;
                    b->Front->Back             =b->Back;
                    PacketSize                 =MyPacket->GetSize();
                    idle[PeerNumber]           =0;
                    MySocket                   ->Send(MyPacket);
                    stringstream ss;
                    ss<<(queue->NumberOfPacketsInQueue);
                    MyAnimator->UpdateNodeDescription(DistributorN,ss.str());

                    if((int((Simulator::Now().GetSeconds()))+1)<(simtime-10)){
                    Time                       tnext(Seconds(1/precision));
                    SendEvent                  =Simulator::Schedule(tnext,&DistributorApp::SkillBasedRouting,this);
                    }
                   }
                  else
                   {
                            if((int((Simulator::Now().GetSeconds()))+1)<(simtime-10)){
                    Time     tnext(Seconds(1/precision));
                    SendEvent=Simulator::Schedule(tnext,&DistributorApp::SkillBasedRouting,this);
                  }
                 }
                }
                else
                {
                                if((int((Simulator::Now().GetSeconds()))+1)<(simtime-10)){
                 Time      tnext(Seconds(1/precision));
                 SendEvent=Simulator::Schedule(tnext,&DistributorApp::SkillBasedRouting,this);
                 }
                }
}

bool *           idle;
Address          peer;
int              PeerNumber;
DataRate         dataRate;
EventId          SendEvent;
uint32_t         PacketSize;
Ptr<Packet>      MyPacket;
uint32_t *       SkillArray;
Ptr<Socket>      MySocket;
DistributorQueue* queue;
AnimationInterface* MyAnimator;
};





class AgentApp:public Application{
public:
AgentApp()
{
 mya         =0;
 myp1        =0;
 myp2        =0;
 MySocket    =0;
 PacketSize   =0;
 peer        =Address();
 PacketsServed=0;
 SendEvent    =EventId();
 EfficiencyGenerator = CreateObject<ExponentialRandomVariable>();
}

void Setup(Ptr<Socket> socket,Address address, uint32_t myAgentskill,int myAgentnumber,AnimationInterface* animator,uint32_t
pic1,uint32_t pic2, uint32_t wage, uint32_t AvgServiceTime)
{
 mya          =animator;
 myp1         =pic1;
 myp2         =pic2;
 peer         =address;
 MySocket     =socket;
 AgentSkill    =myAgentskill;
 AgentNumber   =myAgentnumber;
 WagePerCall   = wage;
 AverageServiceTime  = AvgServiceTime;
 EfficiencyGenerator ->SetAttribute("Mean", DoubleValue(AverageServiceTime));
}

void StartApplication()
{
 MySocket->Bind();
 bool ipRecvTtl=true;
 bool ipRecvTos=true;

 MySocket->SetIpRecvTos (ipRecvTos);
 MySocket->SetIpRecvTtl (ipRecvTtl);
 MySocket->Connect(peer);
 MySocket->SetRecvCallback(MakeCallback(&AgentApp::SocketRecv,this));
}

 void StopApplication ()
 {
    if (SendEvent.IsRunning ())
      {
        Simulator::Cancel (SendEvent);
      }
      if (MySocket)
      {
        MySocket->Close ();
      }

 }
void SocketRecv(Ptr<Socket> socket)
{
 mya           ->UpdateNodeImage (AgentNumber+2, myp2);
 MyPacket      =MySocket->Recv();
 PacketSize    =MyPacket->GetSize();
 PacketSize    =PacketSize-40;
 SocketIpTosTag skillTag;
 MyPacket      ->RemovePacketTag(skillTag);
 MyPacket      ->AddPacketTag(skillTag);
 TimeOfLastArrival = (uint32_t(Simulator::Now().GetSeconds()));
 /*myfile <<endl;
 myfile<<"Agent Connected"<<endl;
 myfile<<"["<<endl;
 myfile<<"AgentNumber: "<<AgentNumber<<endl;
 myfile<<"AgentSkill: "<<AgentSkill<<endl;
 myfile<<"PacketsServed: "<<PacketsServed<<endl;
 myfile<<"PacketSize:"<<PacketSize<<endl;
 myfile<<"Packet Skill: "<<((uint32_t)skillTag.GetTos())<<endl;
 myfile<<"Time: "<<Time(Seconds(Simulator::Now().GetSeconds()))<<endl;
 myfile<<"]"<<endl;
 */
 for(int i = 0; i < randm; i++)
 {OP_Efficiency = EfficiencyGenerator->GetInteger();}
 OP_Efficiency = 0;
 while(OP_Efficiency<=0){OP_Efficiency = EfficiencyGenerator->GetInteger();}
 if((int((Simulator::Now().GetSeconds()))+int((PacketSize+OP_Efficiency)/2))<(simtime-10)){
                Servicing[AgentNumber]=1;
 Time          tnext (Seconds(static_cast<double>((PacketSize+OP_Efficiency)/2)));
 if(((PacketSize+OP_Efficiency)/2)>MaxCallLength){MaxCallLength=((PacketSize+OP_Efficiency)/2);}
 SendEvent     =Simulator::Schedule(tnext,&AgentApp::ServiceComplete,this);
 }
}

void ServiceComplete()
{
 idleArray[AgentNumber]=1;
 Servicing[AgentNumber]=0;
 mya                ->UpdateNodeImage (AgentNumber+2,myp1);
 PacketsServed          ++;
 CallsRecieved_SinceJoined = PacketsServed;
 MySocket               ->SetRecvCallback(MakeCallback(&AgentApp::SocketRecv,this));
 TimeOfLastService = (uint32_t(Simulator::Now().GetSeconds()));
 TimeOfLastServiceArray[AgentNumber]=TimeOfLastService;
 TotalWages+=WagePerCall;
 /*myfile <<endl;
 myfile<<"Agent Reply"<<endl;
 myfile<<"["<<endl;
 myfile<<"AgentNumber: "<<AgentNumber<<endl;
 myfile<<"AgentSkill: "<<AgentSkill<<endl;
 myfile<<"PacketsServed: "<<PacketsServed<<endl;
 myfile<<"PacketSize: "<<PacketSize<<endl;
 myfile<<"Time: "<<Time(Seconds(Simulator::Now().GetSeconds()))<<endl;
 myfile<<"]"<<endl;
 */
 TimeOfLastService = (uint32_t(Simulator::Now().GetSeconds()));
}

Address          peer;
uint32_t CallsRecieved_InADay; //......to model the exhaustion level.
uint32_t CallsRecieved_SinceJoined; //......to model the experience level.
uint32_t AverageServiceTime;
uint32_t WagePerCall; //......to model the wages.
uint32_t TimeOfLastService; //..... For Longest Idle First Routing scheme.
uint32_t TimeOfLastArrival; //..... For Longest Idle First Routing scheme.
uint32_t breaks; //......to model the .
uint32_t         myp1;
uint32_t         myp2;
int            PacketsServed;
EventId          SendEvent;
int            AgentNumber;
uint32_t         PacketSize;
Ptr<Socket>        MySocket;
Ptr<Packet>        MyPacket;
uint32_t         AgentSkill;
AnimationInterface* mya;
//EFFICIENCY OF THE Agent.
uint32_t OP_Efficiency;
Ptr<ExponentialRandomVariable> EfficiencyGenerator;

};
// Random variable generator with meani Avg service time->time of Service.
// Routing Schemens LIR,FSF, Lowest cost




class App:public Application
{
  public:
    App()
    {
      MySocket=0;
      peer=Address();
    }

    void Setup(Ptr<Socket> socket,Address address)
    {
      peer=address;
      MySocket=socket;
    }

    void StartApplication()
    {
      MySocket->Bind();
      MySocket->Connect(peer);
      Ptr<Packet> p = Create<Packet> (100);
      for(int i = 0; i<1; i++){MySocket->Send(p);}

      MySocket->SetRecvCallback(MakeCallback(&App::SocketRecv,this));
```

```
        }

        void SocketRecv(Ptr<Socket> socket)
        {
           MyPacket=MySocket->Recv();
           MySocket->Send(MyPacket);
        }

        Address peer;
        Ptr<Socket> MySocket;
        Ptr<Packet> MyPacket;

};







int stringtoint(string S)
{
 stringstream ss(S);
 int H;
 ss>>H;
 return H;
}











NS_LOG_COMPONENT_DEFINE ("CallCenterSimulation");
int
main (int argc, char *argv[])
{

 for(int i=0;i<90000;i++)
 {
                TEMP3[i]=new string[9];
                AVAIL_AGENT[i]=new string[2];
                AG_SER[i]=new int[4];AG_SER[i][0]=0;AG_SER[i][1]=0;AG_SER[i][2]=0;AG_SER[i][3]=0;
 }

 for(int i=0;i<90000;i++)
 {
                ordered[i]="a";
 }

 for(int i=0;i<25;i++)
 {TABLE[i]=new int[3];TABLE[i][0]=0;TABLE[i][1]=0;TABLE[i][2]=0;
 }


 ifstream INSTREAM;
 INSTREAM.open("january.txt");
 ofstream o;
```

```
o.open("res");
string BUFFER;
int index     =0;
int TOTALCALLS=0;
int SUM1      =0;
int SUM2      =0;
int MAX_Q     =0;
int SUM4      =0;
getline(INSTREAM,BUFFER);



while(!(INSTREAM.eof()))
{
getline(INSTREAM,BUFFER);

istringstream ss(BUFFER);

for(int j=0;j<17;j++){ ss>>TEMP[j];}

ENTRYTIME[index]=TEMP[6];


string day="990103";
string agent="AGENT";

if(TEMP[5].compare(day)==0)
  {

    int POS1          = ENTRYTIME[index].find(":");
    int HRS = stringtoint(ENTRYTIME[index].substr(0,POS1))*3600;
    if(TEMP[12].compare(agent)==0){
    TABLE[stringtoint(ENTRYTIME[index].substr(0,POS1))][0]++;
    TABLE[stringtoint(ENTRYTIME[index].substr(0,POS1))][1]+=stringtoint(TEMP[11]);
    TABLE[stringtoint(ENTRYTIME[index].substr(0,POS1))][2]+=stringtoint(TEMP[15]);
    }
    string TEMP1       = ENTRYTIME[index].erase(0,POS1+1);
    int POS2=TEMP1.find(":");
    if(stringtoint(TEMP[11])<5){SUM4++;}
    VRU_ENTRY[index]=HRS+stringtoint(TEMP1.substr(0,POS2))*60+stringtoint(TEMP1.substr(POS2+1));
    ordered[VRU_ENTRY[index]]=BUFFER;
    /*
    TYPE        [VRU_ENTRY[index]]=TEMP[4];
    DATE        [VRU_ENTRY[index]]=TEMP[5];
    VRU_DURATION[VRU_ENTRY[index]]=TEMP[8];
    Q_TIME      [VRU_ENTRY[index]]=TEMP[11];
    OUTCOME     [VRU_ENTRY[index]]=TEMP[12];
    SERV_TIME   [VRU_ENTRY[index]]=TEMP[15];
    SERVER      [VRU_ENTRY[index]]=TEMP[16];
    */

    TEMP3[TOTALCALLS][1]=TEMP[4];
    TEMP3[TOTALCALLS][2]=TEMP[5];
    TEMP3[TOTALCALLS][3]=TEMP[6];
    TEMP3[TOTALCALLS][4]=TEMP[8];
    TEMP3[TOTALCALLS][5]=TEMP[11];
    TEMP3[TOTALCALLS][6]=TEMP[12];
    TEMP3[TOTALCALLS][7]=TEMP[15];
    TEMP3[TOTALCALLS][8]=TEMP[16];
    TOTALCALLS++;

  }
index++;
}
```

```
int COUNT=0;
for(int i=0;i<TOTALCALLS;i++)
{

string noSERVER="NO_SERVER";
int POS=0;

if((TEMP3[i][8].compare(noSERVER))!=0)
 {

                bool PRESENT=0;
                 for(int h=0;h<COUNT;h++)
                   {
                     if((AVAIL_AGENT[h][0].find(TEMP3[i][8]))==0){PRESENT=1;POS=h;h=COUNT;}
     }

                 if(!PRESENT)
                   {
                     AVAIL_AGENT[COUNT][0]=TEMP3[i][8];

                     AVAIL_AGENT[COUNT][1]+=TEMP3[i][1];AVAIL_AGENT[COUNT][1]+="-";

                     AG_SER[COUNT][0]+=stringtoint(TEMP3[i][7]);
                     AG_SER[COUNT][1]+=1;
                     if(TEMP3[i][1]=="PS"){AG_SER[COUNT][3]|=(1<<0);}
                     if(TEMP3[i][1]=="NW"){AG_SER[COUNT][3]|=(1<<1);}
                     if(TEMP3[i][1]=="NE"){AG_SER[COUNT][3]|=(1<<2);}
                     if(TEMP3[i][1]=="TT"){AG_SER[COUNT][3]|=(1<<3);}
                     if(TEMP3[i][1]=="IN"){AG_SER[COUNT][3]|=(1<<4);}
                     if(TEMP3[i][1]=="PE"){AG_SER[COUNT][3]|=(1<<5);}

                     SUM1+=stringtoint(TEMP3[i][5]);
                     SUM2+=stringtoint(TEMP3[i][4]);
                     if(stringtoint(TEMP3[i][5])>MAX_Q){MAX_Q=stringtoint(TEMP3[i][5]);        }
                     COUNT++;
                   }

                 if(PRESENT)
                   {

                     AVAIL_AGENT[POS][1]+=TEMP3[i][1];AVAIL_AGENT[POS][1]+="-";

                     AG_SER[POS][0]+=stringtoint(TEMP3[i][7]);
                     AG_SER[POS][1]+=1;
                     if(TEMP3[i][1]=="PS"){AG_SER[POS][3]|=(1<<0);}
                     if(TEMP3[i][1]=="NW"){AG_SER[POS][3]|=(1<<1);}
                     if(TEMP3[i][1]=="NE"){AG_SER[POS][3]|=(1<<2);}
                     if(TEMP3[i][1]=="TT"){AG_SER[POS][3]|=(1<<3);}
                     if(TEMP3[i][1]=="IN"){AG_SER[POS][3]|=(1<<4);}
                     if(TEMP3[i][1]=="PE"){AG_SER[POS][3]|=(1<<5);}
                     SUM1+=stringtoint(TEMP3[i][5]);
                     SUM2+=stringtoint(TEMP3[i][4]);
                     if(stringtoint(TEMP3[i][5])>MAX_Q){MAX_Q=stringtoint(TEMP3[i][5]);        }
                   }

  }
}




int SUM =0;
int SUM3=0;
int MaxCallDuration = 0;

for(int i=0;i<COUNT;i++)
{

                AG_SER[i][2]=AG_SER[i][0]/AG_SER[i][1];
                cout<<AVAIL_AGENT[i][0];//<<" "<<AVAIL_AGENT[i][1];
                cout<<endl;
                cout<<"TotalAnswerTime:    "<<AG_SER[i][0]<<endl;
                cout<<"TotalcallsAnswered: "<<AG_SER[i][1]<<endl;
                cout<<"AverageServiceTime: "<<AG_SER[i][2]<<endl;
                cout<<"Agent Skill:        "<<AG_SER[i][3]<<endl<<endl;
  SUM +=AG_SER[i][0];
  if(AG_SER[i][0]>MaxCallDuration){MaxCallDuration=AG_SER[i][0];}
  SUM3+=AG_SER[i][1];
}

TABLE[24][0]=0;
o<<"Calls Received Per Hour="<<endl;
for(int i=0;i<24;i++)
{
o<<TABLE[i][0]<<" ";
if (TABLE[i][0]>MaxCallsPerHour){MaxCallsPerHour=TABLE[i][0];}
}
o<<endl<<endl;



o<<"Average Queue Times per Hour="<<endl;
for(int i=0;i<24;i++)
{
                if(double(TABLE[i][0])!=0)
                {
                   o<<double(TABLE[i][1])/double(TABLE[i][0])<<" ";
                }
  else{
                o<<0<<" ";
  }
}
o<<endl<<endl;



o<<"Average Service Times per Hour="<<endl;
for(int i=0;i<24;i++)
{
                if(double(TABLE[i][0])!=0)
                {
                                o<<double(TABLE[i][2])/double(TABLE[i][0])<<" ";
                }
  else
  {
                o<<0<<" ";
  }
}
o<<endl<<endl;



o<<"Total Calls Answered=      "<<SUM3<<" / "<<TOTALCALLS<<endl;
o<<"Average Duration of Call=  "<<SUM/SUM3<<endl;
o<<"Maximum Duration of Call=  "<<MaxCallDuration<<endl;
o<<"Average Queue Time=        "<<SUM1/SUM3<<endl;
o<<"Maximum Patience in Queue= "<<MAX_Q<<endl;
```

```
o<<"Average VRUQueue Time=     "<<SUM2/SUM3<<endl;
o<<"Probaility Of Wait=        "<<(double(TOTALCALLS-SUM4)*100)/double(TOTALCALLS)<<endl;
INSTREAM.close();

ofstream OUTSTREAM;
OUTSTREAM.open("ordered.txt");
for(int i=0;i<90000;i++)
{
        if(ordered[i]!="a")
        OUTSTREAM<<ordered[i]<<endl;
}
OUTSTREAM.close();



Mean_Packet_Size           =SUM/SUM3;
Mean_Patience              =MAX_Q/2;
number_of_Agent_nodes      =COUNT;
WagePerCall                =100;
AverageServiceTime         =SUM/SUM3;
meanVRUTime                =SUM2/SUM3;

idleArray               =new bool    [number_of_Agent_nodes];
SkillArray              =new uint32_t[number_of_Agent_nodes];
WagesArray              =new uint32_t[number_of_Agent_nodes];
AvgServiceTimeArray     =new uint32_t[number_of_Agent_nodes];
TimeOfLastServiceArray  =new uint32_t[number_of_Agent_nodes];
Servicing               =new bool    [number_of_Agent_nodes];


for (int i = 0; i < number_of_Agent_nodes; i++)
{
SkillArray[i]        =AG_SER[i][3];
idleArray[i]         =1;
WagesArray[i]        =AG_SER[i][3]*100;
AvgServiceTimeArray[i]=AG_SER[i][2];
TimeOfLastServiceArray[i]=0;
Servicing[i]         =0;
}










LogComponentEnable ("CallCenterSimulation", LOG_LEVEL_INFO);


CommandLine cmd;
// cmd.AddValue ("nue_node", "Number of \"extra\" CSMA nodes/devices", nue_node    );
//cmd.AddValue ("enb_node", "Number of \"extra\" CSMA nodes/devices", enb_node);
cmd.Parse (argc, argv);

myfile.open ("Packets.txt");
```

```
NodeContainer caller_node_container;
NodeContainer distributor_node_container;
NodeContainer caller_distributor_nodes_container;
NodeContainer Agent_nodes_container;
NodeContainer distributor_Agents_node_container[number_of_Agent_nodes];
NodeContainer c;

caller_node_container                 .Create(1);
distributor_node_container             .Create(1);
Agent_nodes_container                  .Create(number_of_Agent_nodes);

caller_distributor_nodes_container     .Add(caller_node_container);
caller_distributor_nodes_container     .Add(distributor_node_container);

for (int i=0;i<number_of_Agent_nodes;i++)
{distributor_Agents_node_container[i]   .Add(distributor_node_container.Get(0));
distributor_Agents_node_container[i]    .Add(Agent_nodes_container  .Get(i));}

c                                   .Add(caller_node_container);
c                                   .Add(distributor_node_container);
c                                   .Add(Agent_nodes_container);




PointToPointHelper caller_distributor_p2p_Helper;
caller_distributor_p2p_Helper.SetDeviceAttribute ("DataRate", StringValue ("64kbps"));
caller_distributor_p2p_Helper.SetChannelAttribute("Delay"   , StringValue ("3s"    ));

NetDeviceContainer caller_distributor_net_device_container;
NetDeviceContainer distributor_Agents_net_device_container_Array[number_of_Agent_nodes];

caller_distributor_net_device_container = caller_distributor_p2p_Helper.Install (caller_distributor_nodes_container);
for(int i=0;i<number_of_Agent_nodes;i++){distributor_Agents_net_device_container_Array[i] =
caller_distributor_p2p_Helper.Install (distributor_Agents_node_container[i]);}



InternetStackHelper stack;
Ipv4AddressHelper address;
stack  .Install (c);
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interface     =address.Assign (caller_distributor_net_device_container);
Ipv4Address Ipv4Addresses           [number_of_Agent_nodes+2];
Ipv4Addresses                      [0]=interface.GetAddress(0);
Ipv4Addresses                      [1]=interface.GetAddress(1);
Ipv4Address DistributorIpv4Addresses [number_of_Agent_nodes];
Ipv4Address AgentIpv4Addresses       [number_of_Agent_nodes];
Address NetDeviceAddresses          [number_of_Agent_nodes];
Address DistributorNetDeviceAddresses[number_of_Agent_nodes];
Address AgentNetDeviceAddresses      [number_of_Agent_nodes];

NetDeviceAddresses[0]=caller_distributor_net_device_container.Get(0)->GetAddress();
NetDeviceAddresses[1]=caller_distributor_net_device_container.Get(1)->GetAddress();

uint16_t Ports[number_of_Agent_nodes+1];
Ports[0]=0;
Ports[1]=1;

char a     [9]  ={'1','0','.','1','.','1','.','0'};
char b     [10] ={'1','0','.','1','.','1','0','.','0'};
char chars [8]  ={'2','3','4','5','6','7','8','9'};
char chars2[10] ={'0','1','2','3','4','5','6','7','8','9'};

for (int i = 0; i < (number_of_Agent_nodes); i++)
```

```
{
if(i<8){a[5]=chars[i];address.SetBase (a, "255.255.255.0");}
if(i>7  && i<18){b[5]='1';b[6]=chars2[i- 8];address.SetBase (b, "255.255.255.0");}
if(i>17 && i<28){b[5]='2';b[6]=chars2[i-18];address.SetBase (b, "255.255.255.0");}
if(i>27 && i<38){b[5]='3';b[6]=chars2[i-28];address.SetBase (b, "255.255.255.0");}
if(i>37 && i<48){b[5]='4';b[6]=chars2[i-38];address.SetBase (b, "255.255.255.0");}
if(i>47 && i<58){b[5]='5';b[6]=chars2[i-48];address.SetBase (b, "255.255.255.0");}
if(i>57 && i<68){b[5]='6';b[6]=chars2[i-58];address.SetBase (b, "255.255.255.0");}
if(i>67 && i<78){b[5]='7';b[6]=chars2[i-68];address.SetBase (b, "255.255.255.0");}
if(i>77 && i<88){b[5]='8';b[6]=chars2[i-78];address.SetBase (b, "255.255.255.0");}
if(i>87 && i<98){b[5]='9';b[6]=chars2[i-88];address.SetBase (b, "255.255.255.0");}

Ipv4InterfaceContainer interfaces =address   .Assign (distributor_Agents_net_device_container_Array[i] );
AgentNetDeviceAddresses[i]       =distributor_Agents_net_device_container_Array[i].Get(1)->GetAddress();
DistributorNetDeviceAddresses[i]  =distributor_Agents_net_device_container_Array[i].Get(0)->GetAddress();
AgentIpv4Addresses[i]            =interfaces.GetAddress(1);
DistributorIpv4Addresses[i]      =interfaces.GetAddress(0);
Ports[i]                         =i;
}
```

```
Ptr<Socket> CallerSocket       = Socket::CreateSocket(c.Get(0)  ,TypeId::LookupByName ("ns3::Ipv4RawSocketFactory"));
CallerSocket                  ->Bind();
CallerSocket                  ->BindToNetDevice(caller_distributor_net_device_container.Get(0));

Ptr<Socket> DistributorSocket   = Socket::CreateSocket(c.Get(1)  ,TypeId::LookupByName ("ns3::Ipv4RawSocketFactory"));
DistributorSocket             ->Bind();
DistributorSocket             ->BindToNetDevice(caller_distributor_net_device_container.Get(1));


Ptr<Socket> AgentSocketArray      [number_of_Agent_nodes];
Ptr<Socket> DistributorSocketArray[number_of_Agent_nodes];

for (int i = 0; i < number_of_Agent_nodes; i++)
{
AgentSocketArray   [i]         = Socket::CreateSocket(c.Get(i+2),TypeId::LookupByName ("ns3::Ipv4RawSocketFactory"));
AgentSocketArray   [i]         ->Bind();
AgentSocketArray   [i]         ->BindToNetDevice(distributor_Agents_net_device_container_Array[i].Get(1));
DistributorSocketArray[i]      = Socket::CreateSocket(c.Get(1)  ,TypeId::LookupByName ("ns3::Ipv4RawSocketFactory"));
DistributorSocketArray[i]      ->Bind();
DistributorSocketArray[i]      ->BindToNetDevice(distributor_Agents_net_device_container_Array[i].Get(0));
}
```

```
 AnimationInterface* anim = new AnimationInterface ("anim1.xml");
 DistributorQueue q         =DistributorQueue(Maximum_Queue_Length,anim);
 DistributorQueue* queue     = &q;
 DistributorQueue Q         =DistributorQueue(1000,anim);
 DistributorQueue* VRUqueue  = &Q;
 uint32_t resourceId1    = anim->AddResource      ("/home/eelab/workspace/ns-allinone-3.26/netanim-3.107/b.png");
 uint32_t resourceId2    = anim->AddResource      ("/home/eelab/workspace/ns-allinone-3.26/netanim-3.107/a.png");
 uint32_t resourceId3    = anim->AddResource      ("/home/eelab/workspace/ns-allinone-3.26/netanim-3.107/d.png");
 uint32_t resourceId4    = anim->AddResource      ("/home/eelab/workspace/ns-allinone-3.26/netanim-3.107/c.png");

 DistributorN = c.Get(1);
 Ptr<CallerApp>     capp = CreateObject<CallerApp>     ();
 Ptr<ReceiverApp>    rapp = CreateObject<ReceiverApp>    ();

 capp->Setup (CallerSocket, InetSocketAddress (Ipv4Addresses[1],
Ports[1]),uint32_t(Mean_InterArrival_Time),uint32_t(Mean_Packet_Size),uint32_t(Mean_Patience),uint32_t(1),uint32_t(meanskill))
;
 c.Get (0)  ->AddApplication (capp           );
 capp      ->SetStartTime   (Seconds (1.     ));
 capp      ->SetStopTime    (Seconds (simtime-10));

 rapp->Setup (DistributorSocket, InetSocketAddress (Ipv4Addresses[0], Ports[0]), queue,0,meanVRUTime,VRUqueue);
 c.Get (1)  ->AddApplication (rapp           );
 rapp      ->SetStartTime   (Seconds (1.     ));
 rapp      ->SetStopTime    (Seconds (simtime-10));

 for (int i = 0; i < number_of_Agent_nodes; i++)
 {
 Ptr<AgentApp>    Oapp   =CreateObject<AgentApp>   ();
 Ptr<DistributorApp> Dapp=CreateObject<DistributorApp>();

 Oapp->Setup (AgentSocketArray[i], InetSocketAddress (DistributorIpv4Addresses[i],
Ports[1]),SkillArray[i],i,anim,resourceId3,resourceId4,WagesArray[i],AvgServiceTimeArray[i]);
 c.Get (i+2)->AddApplication (Oapp           );
 Oapp      ->SetStartTime   (Seconds (1.     ));
 Oapp      ->SetStopTime    (Seconds (simtime-10));

 Dapp->Setup (DistributorSocketArray[i],DataRate("64kbps"), InetSocketAddress (AgentIpv4Addresses[i], Ports[i+2]),SkillArray,
queue,idleArray,i,anim);
 c.Get (1)  ->AddApplication (Dapp           );
 Dapp      ->SetStartTime   (Seconds (1.     ));
 Dapp      ->SetStopTime    (Seconds (simtime-10));
 }




 AsciiTraceHelper ascii;
 //caller_distributor_p2p_Helper.EnableAsciiAll (ascii.CreateFileStream ("p2p.tr"));

 //caller_distributor_p2p_Helper.EnablePcapAll ("simple-point-to-point-olsr");

 Ipv4GlobalRoutingHelper Router = Ipv4GlobalRoutingHelper();
 Router.PopulateRoutingTables();


 anim->SetConstantPosition(caller_node_container     .Get(0), 0,10);
 anim->SetConstantPosition(distributor_node_container.Get(0),10,10);
 int sign=1;
```

```
for(int i=0;i<number_of_Agent_nodes;i++)
{anim->SetConstantPosition(Agent_nodes_container.Get(i),20,10+sign*5*((i+1)/2));
sign =-sign;}

//anim->SetBackgroundImage ("/home/eelab/workspace/ns-allinone-3.26/netanim-3.107/a.png", -50, -50, .1, .1, 1.0);

anim->UpdateNodeImage (0, resourceId1);
anim->UpdateNodeImage (1, resourceId2);

for (int i = 2; i <= number_of_Agent_nodes+1; i++)
{
anim->UpdateNodeImage (i, resourceId3);
}

Simulator::Stop (Seconds (simtime));
Simulator::Run ();
NS_LOG_INFO("Ending Topology");
Simulator::Destroy ();
myfile.close();



ifstream in;
ofstream out;
in.open("Packets.txt");
string buffer;
int HourlyWage                =400;  // per hour
int var                       =0;
int calls                     =0;
int Drops                     =0;
int totalcalls                =0;
int Abandonements             =0;
unsigned long long  sum7       =0;
unsigned long long avgCalTime   =0;
unsigned long long delay        =3;
double Promoter               =0;
double Detractor              =0;
double Passive                =0;
double prob                   =0;
int counter                   =0;
int QS                        =0;
int pep                       =0;
int servicelevel              =0;
int qsize                     =0;
int sum8                      =0;
double v                                        [1000000];
unsigned long long swait        = (1000000/precision);
unsigned long long *Patience    =new unsigned long long[1000000];
unsigned long long *DequeueTimes=new unsigned long long[1000000];
unsigned long long *EnqueueTimes=new unsigned long long[1000000];
unsigned long long *WaitingTimes=new unsigned long long[1000000];
unsigned long long *CallTimes   =new unsigned long long[1000000];
unsigned long long *PacketSizes =new unsigned long long[1000000];
unsigned long long  HourlyAverageQueueWaitingTimes [24];
unsigned long long CallsPerHour [24];
std::string x;
//out.open("outfile.txt");

for (int i = 0; i < 24; i++)
{
        CallsPerHour[i]=0;
}

while (!(in.eof())){
getline(in, buffer);
```

```
if (buffer == "Call Sent")
{
getline(in, buffer);
std::size_t pos          = buffer.find(":");
std::string str3         = buffer.substr(pos + 1);
std::string::size_type sz = 0;
unsigned long long li_dc  = std::stoull(str3, &sz, 10);
PacketSizes[calls]       = li_dc;


getline(in, buffer);
pos                   = buffer.find(":");
str3                  = buffer.substr(pos + 1);
sz                   = 0;
li_dc                 = std::stoull(str3, &sz, 10);
sum7                   += li_dc;

getline(in, buffer);
pos                   = buffer.find(":");
str3                  = buffer.substr(pos + 1);
sz                   = 0;
li_dc                 = std::stoull(str3, &sz, 10);
Patience[calls]          = li_dc;

getline(in, buffer);
pos                   = buffer.find("+");
str3                  = buffer.substr(pos + 1);
str3                  .erase(str3.end() - 7, str3.end());
sz                   = 0;
li_dc                 = std::stoull(str3, &sz, 10);
CallTimes[calls]         = li_dc;
calls++;
qsize++;
}

if(buffer=="Enqueue")
{
getline(in, buffer);
std::size_t pos          = buffer.find(":");
std::string str3         = buffer.substr(pos + 1);
std::string::size_type sz = 0;
//unsigned long long li_dc  = std::stoull(str3, &sz, 10);
//out<<li_dc<<" ";

getline(in, buffer);
pos          = buffer.find(":");
str3         = buffer.substr(pos + 1);
sz = 0;
//li_dc  = std::stoull(str3, &sz, 10);
//out<<li_dc<<endl;

}

if (buffer == "Packet Removed from Queue")
{
std::size_t pos;
std::string str3;
std::string::size_type sz = 0;

getline(in, buffer);
pos                   = buffer.find(":");
x                    = buffer.substr(pos+1);
double addr             = stod(x,&sz);
```

```
getline(in, buffer);
pos                    = buffer.find("+");
str3                   = buffer.substr(pos+1);
str3                   .erase(str3.end() - 7, str3.end());
sz                     = 0;
unsigned long long li_dec = std::stoull(str3, &sz,10);
EnqueueTimes[totalcalls]  = li_dec;


getline(in, buffer);
pos                    = buffer.find("+");
str3                   = buffer.substr(pos+1);
str3                   .erase(str3.end() - 7, str3.end());
sz                     = 0;
unsigned long long li_dec2= std::stoull(str3, &sz,10);
DequeueTimes[totalcalls]  = li_dec2;

HourlyAverageQueueWaitingTimes[int((li_dec2/1e6)/3600)]+=li_dec2-li_dec;
CallsPerHour[int((li_dec2/1e6)/3600)]++;
totalcalls++;
QS = QS+1;
for(int i=0;i<counter;i++)
{
  if(v[i]==addr)
  {
     prob++;i=counter;QS = QS-1;
  }
}
v[counter]=addr;
counter++;

}



if (buffer == "Queue Abandonement ")
{
Abandonements++;

getline(in, buffer);
std::size_t pos                    = buffer.find("+");
string str3                   = buffer.substr(pos+1);
str3                   .erase(str3.end() - 7, str3.end());
std::string::size_type sz                     = 0;
unsigned long long li_dec= std::stoull(str3, &sz,10);


getline(in, buffer);
pos                    = buffer.find("+");
str3                   = buffer.substr(pos+1);
str3                   .erase(str3.end() - 7, str3.end());
sz                     = 0;
unsigned long long li_dec2= std::stoull(str3, &sz,10);

HourlyAverageQueueWaitingTimes[int((li_dec2/1e6)/3600)]+=li_dec;
CallsPerHour[int((li_dec2/1e6)/3600)]++;

QS -= 1;
}


if(buffer == "Packet Dropped , MaximumQueueLimit Exceeded")
{
  Drops++;
  QS -= 1;

}
}




in.close();

unsigned long long sum=0,sum2=0,sum4=0,sum5=0;
int rest = 0;
for (int i = 0; i < totalcalls; i++)
{
//if ((DequeueTimes[i]>0) && (EnqueueTimes[i] > 0)&&(DequeueTimes[i]>EnqueueTimes[i]))
{
WaitingTimes[i] = DequeueTimes[i] - EnqueueTimes[i];
//cout<<EnqueueTimes[i]<<"  "<<DequeueTimes[i]<<"  "<<WaitingTimes[i]<<endl;
sum += WaitingTimes[i];
rest++;
}
}
//cout<<sum<<" "<<rest<<" "<<totalcalls<<endl;
unsigned long long mean = sum / rest;
sum5+=Patience[0];
sum4+=PacketSizes[0];
for (int i = 0; i < calls; i++)
{
sum2+=(CallTimes[i]-CallTimes[i-1]);
sum5+=Patience[i];
sum4+=PacketSizes[i];
//cout<<CallTimes[i]<<" "<<CallTimes[i-1]<<" "<<Patience[i]<<" "<<PacketSizes[i]<<endl;
}

Promoter = totalcalls*9;
Detractor = Abandonements*4;
Passive = ((Drops)+(0.5*totalcalls))*7;

avgCalTime = ((sum/1e6) +(totalcalls*(sum4/calls))+(delay*calls)+(delay*totalcalls))/(calls);

var = totalcalls*(sum4/calls);

for(int k=0;k<calls;k++)
{
 if(WaitingTimes[k]>swait)
  {
     pep++;
  }
}

for (int i=0;i<totalcalls;i++){if(WaitingTimes[i]<=10e6){sum8++;}}
servicelevel = (double(sum8)/double(totalcalls))*100;

o<<"CallsPerHour="<<endl;
for (int i = 0; i < 24; i++)
{
		o<<CallsPerHour[i]<<" ";
}
o<<endl;
o<<"HourlyAverageQueueWaitingTimes="<<endl;
for (int i = 0; i < 24; i++)
{
		if(CallsPerHour[i]>0)
		{
		o<<(double(HourlyAverageQueueWaitingTimes[i])/double(CallsPerHour[i]))/1e6<<" ";}
```

```
            else {o<<0<<" ";}
 }
 o                                                                              <<endl;
 o                                                                              <<endl;
 o<<"Arrivals per Minute=        "<<60/((double(sum2)/double(calls-1))/1e6)                                  <<endl;
 o<<"InterArrivalTime=           "<<((double(sum7)/double(calls)))                                    <<"s"<<endl;
 o<<"Mean Packet Size=           "<<(double(sum4)/double(calls))                                          <<endl;
 o<<"Blocking Percentage=        "<<((double(Drops)*100)/double(calls))                                  <<"%"<<endl;
 o<<"Average Patience=           "<<(double(sum5)/double(calls))                                      <<"s"<<endl;
 o<<"Abandonements Percentage=   "<<((double(Abandonements)*100)/double(calls))
<<"%"<<endl;
 o<<"Mean WaitingTime=           "<<(double(mean)/1e6)                                               <<"s"<<endl;
 o<<"Total Calls=                "<< calls                                   <<endl;
 o<<"Total Problem Resolution=   "<< totalcalls                                          <<endl;
 o<<"Percentage of calls serviced=  "<< ((double(calls)-double(Abandonements)-double(Drops))/double(calls))*100
<<"%"<<endl;
 o<<"First Call Resolution=      "<< ((double(calls)-double(prob))/double(calls))*100                       <<"%"<<endl;
 o<<"Net Promotor Score=         "<<(((double(Promoter)-
double(Detractor)))/(double(Promoter)+double(Passive)+double(Detractor)))*100     <<endl;
 o<<"Percentage of time Agents
busy="<<((double(totalcalls)*(double(sum4)/double(calls)))/(double(simtime*number_of_Agent_nodes)))*100   <<"%"<<endl;
 o<<"Average Caller time in system= "<<(double(avgCalTime))                                      <<"s"<<endl;
 o<<"Total Agents Salary=        "<<(double((var/3600)*HourlyWage))                                 <<"Rs"<<endl;
 o<<"Quality Score=              "<<(double(QS))                                             <<endl;
 o<<"Prob of waiting=            "<<(double(pep)/double(calls))*100                               <<"%"<<endl;
 o<<"Service Level=              "<<double(servicelevel)                                   <<"%"<<endl;
 o<<"Traffic Intensity=          "<<(double(double(calls-1)*1000000)/((double(sum2))))*(double(sum4)/double(calls))
<<endl;
 o<<"Total Wages=                "<<TotalWages                                               <<endl;
 o<<"MaxCallLength=              "<<MaxCallLength                                              <<endl;
 return 0;
 o.close();
}
```