Stochastic Simulation of
Communication Networks and their
Protocols
University of Bremen
Prof. Dr. Anna Förster
Dr.-Ing. Asanga
Udugama

Project Report

Stochastic Modified KiBaM

of

JIBIN PJOHN

Matriculation Number: 3061574

Bremen, 1st March 2017

I assure, that this work has been done solely by me without any further help from others except for the official attendance by the Chair of Sustainable Communication Networks. The literature used is listed completely in the bibliography.

Bremen, 1st March 2017

(Jibin P John)

CONTENTS

CHAPTER 1

Introduction

A major constraint in design of mobile embedded systems today is the battery lifetime for a given size and weight of the battery. With the tremendous increase in the computing power of hardware and the relatively slow growth in the energy densities of the battery technologies, estimating the lifetime and energy delivered by the battery has become increasingly important to choose between alternative implementations and architectures for mobile computing platforms.

In early models, the electrochemical processes in the battery have been expressed using partial differential equations. Although they take the recovery effect into account, they are cumbersome and they take prohibitively long (order of days) to estimate battery life [5]. They typically rely on numerical simulation and require significant computation, which makes them impractical. Stochastic battery models [6, 8] have also been proposed which are faster than to the PDE model and also inculcate the recovery and rate capacity effects but we have found that these were not sufficiently sensitive to predict certain experimental observations dealing with recovery.

The Stochastic Modified KiBaM effectively addresses the drawbacks of its previous models. In this project Stochastic Modified KiBaM is realized using OMNET ++. The effect of battery life has been studied with respect to various parameters. This paper gives an intuitive idea about how these parameters effect on battery life based on the OMNET simulation. The rest of the paper is organized as follows. Chapter 2 discusses the theoretical background of the models. Assumptions and restrictions have been listed in Chapter 3. Chapter 4 describes about the mathematical modeling and how the values of internal parameter chosen. Also describes the implementation in software (OMNET ++). Chapter 5 discusses the simulation result and finally concluded in Chapter 6.

CHAPTER 2

Stochastic Modified KiBaM

The Stochastic Modified KiBaM model battery discharge as 3-dimensional Markov process which is a stochastic extension of the Kinetic Battery model with certain refinements and additional parameters for the accuracy. The modified Kinetic Battery model is as follows, Battery is modeled as two wells of charge, as shown in Figure 2.1. The available-charge well supplies electrons directly to the load, the bound-charge well supplies electrons only to the available-charge well. Parameter c is a capacity ratio and corresponds to the fraction of total charge in the battery that is readily available. Parameter i represents the amount of charge in the available charge well and j represents charge in bounded charge well at any point during the battery life time. The rate of charge flow between the two wells is a function of k' height h2 and the difference in the heights of the two wells, h1 and h2. The dependence on height h2 reflects the observation that a battery has more tendency to recover when it has more charge left. The internal resistance of the battery is represented by R0[2].

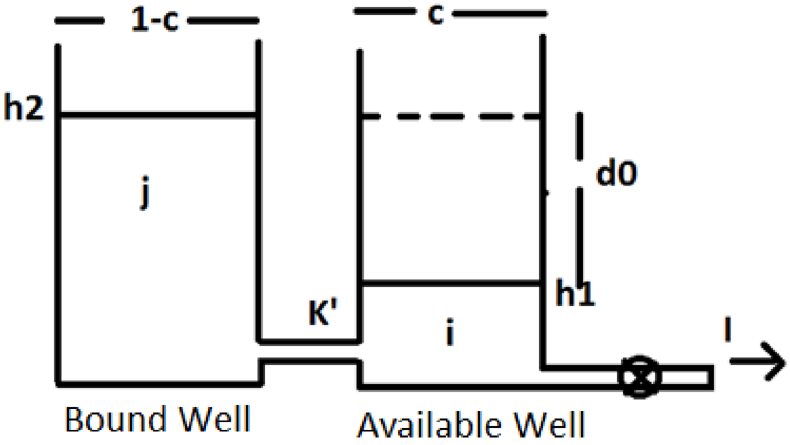


Fig 2.1 Modified KiBaM with rate of transfer between two wells is equal to k'.h2.(h2 − h1)

Without loss of generality, the wells can be considered as two dimensional which results in the following equations

$h2.(1 - c) = j$
$h1.c = i$ (2.1)

The battery is modeled using three state parameters (i, j, t), making a three dimensional Markov chain structure. Parameters i and j as explained above are the amount of charge stored in available and bound well of charge respectively, t is length of the current idle slot i.e. time since some current was drawn from the battery previous to the current moment. Parameters i and j are measured in charge units, where a charge unit is the smallest unit of charge that can be drawn from the battery or depends upon the granularity of the simulation and t is measured in time units, where a time unit is the least count of the time in simulation[1].

Battery behavior is represented as a discrete time transient Markov process, that keeps tracks of the three parameters of the battery. The battery discharge starts from the state of full charge i.e. with initial of values of parameters i and j (subject to the battery used) and t = 0, and terminates when the battery is discharged, i.e. being in any of states having i = 0. qI is the probability that in one time unit, called a time slot, I charge units are demanded. This is modeled as I charge units per unit time are drawn from theavailable charge well while some charge J[Eq.(2.2)] is being transferred or replenished by the bounded charge well, to the available charge well[1].

$$(i, j, 0) \rightarrow (i - I + J, j - J, 0)$$

$$J = k'.h2.(h2 - h1) \qquad (2.2)$$

An example of such transition (2.2) is depicted in the following figure.

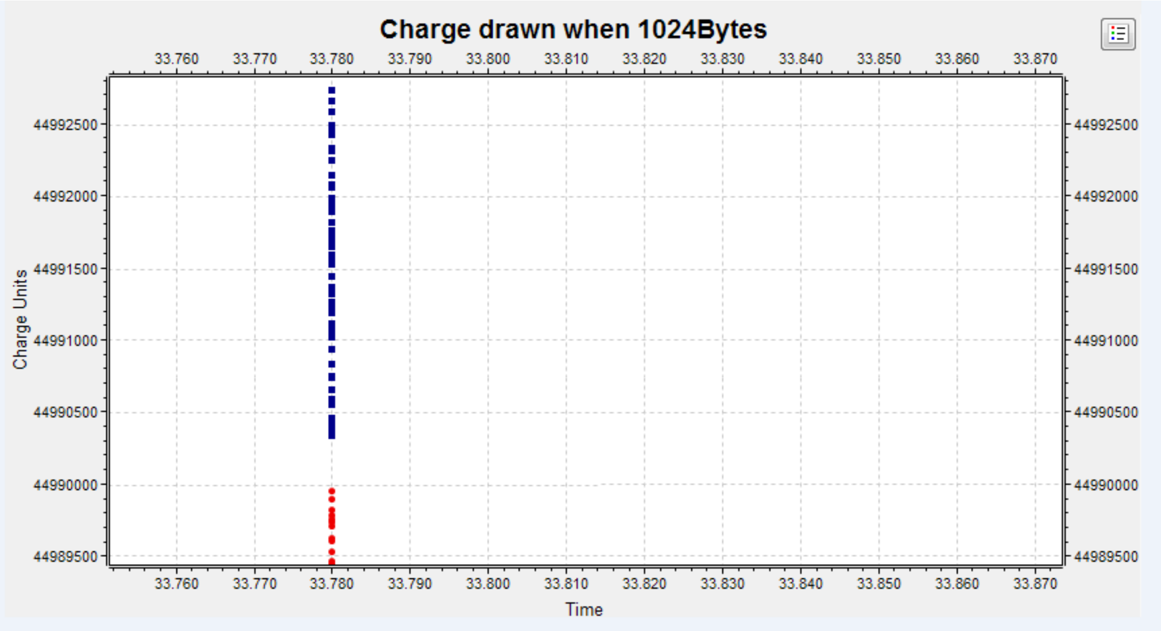


Fig 2.2 Transition based on equations 2.2

If there are idle periods in between discharges, the battery can partially recover its charge during these idle times, and thus we can drain a larger number of charge units before reaching the states having parameter i = 0 i.e. fully discharged states. Let the probability that an idle slot occurs be q0. If the given idle slot is infinitely long, there will occur adequate transfer of charge from the bound well to the available well so as to equalize the heights. However, this recovery or transfer of charge is not constant over the idle slot, but varies with the length of the idle slot[1].

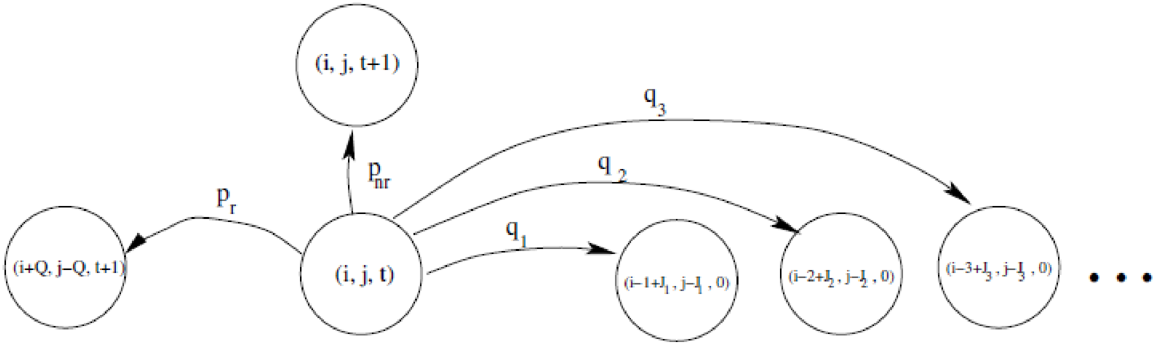


Fig 2.3 Stochastic KiBaM as 3-dimensional Markov process.

An approximated best fit curve will represent the average recovery per idle slot and serve as a characteristic for the particular battery. The differential(p(t)) of the curve gives the probability to recover with time during an idle slot. During a given idle slot the battery may or may not recover. The quanta(Q) of charge it recovers depends on the current state of the battery i.e. height h2 , (h2 − h1) and the granularity of time. The quanta(Q) of recovery is calculated so as the charge recovered for an infinitely long idle slot is equal to total charge that needs to be transferred between the two wells before there heights are equalized. If there is no recovery then there will be no change in the values of parameter i and j while t is incremented by one for the next successive idle slot. Figure 2.2 shows the possible transitions during an idle slot. Equations 2.3 and 2.4 summarizes the transitions and their probabilities respectively[1].

$(i, j, t) \quad \rightarrow \qquad (i + Q, j - Q, t + 1)$ (i)
$(i, j, t + 1)$ (ii)
$(i - I + J, j - J, 0)$ (iii) (2.3)

The rate capacity effect is automatically incorporated in the model, because as the discharge current increases, charge in the available well decreases more rapidly giving the bound charge well less time to replenish the available charge well and battery is declared to be discharged even though there is a lot of charge left inside the bound charge well, resulting in the inefficiency known as rate capacity effect[1].

The Model can be easily extended to calculate battery life for deterministic discharge profiles. At each step of the simulation, we look at the input discharge waveform and calculate the average current over each time slot, converting it into appropriate number of charge units depending on the time unit of the simulation. The number of charge units(I) calculated are drawn from the battery model with qI = 1,while recovery during an idle slot still remains probabilistic[1].

CHAPTER 3

Assumptions and Restrictions.

- Each network node consists of two individual modules: EnergyConsumer and BatteryModel. Energy model deals with the interaction with other nodes. It calculates the power requirement (current required) and send a trigger message to Battery model. The trigger message (BatteryMsg) encapsulates the current state of the EnergyConsumer(Idle or active) and other relevant parameters. Upon receiving this message the BatteryModel extracts these information, does the necessary calculations and it deliver power to the Energy model. Also it updates its internal parameters and state.

- The states are restricted to either idle state (deep sleep) or active state (send mode and receive mode)

- The current consumption during the idle state has been neglected due to equation 2.3,(i)restrict it.

- It has been assumed that the receiver is ideal. Whenever a message hits on receiver it wakes up from the sleep instantaneously.

- The BatteryModel keep track on the values charge in available well (i), charge in bound well (j) and (t).

- Theoretically, It possible to draw a charge unit from infinite set of values (qI)at any simulation instances. However, we have restricted to it to 100 (qI) units.

- The simulation stops when the charge in available well is zero (i=0).

CHAPTER 4

The Mathematical Model and Implementation.

4.1 Battery Model

The simulation least count in time was chosen to be 50 ms (.05s).

The parameter k’ and the initial values of i and j are the characteristics of the battery which is being discharged. The maximum capacity of the battery is defined as the charge delivered by it under infinitesimal load. Similarly the charge in the available well is defined as the charge that would be delivered if we were to draw infinite current.

The battery used in our experiments has a maximum capacity of 2000 mAh. The charge in the available well, when the battery is fully charged is taken as 1250 mAh. The remaining charge present in the bound well is 750 mAh. These initial values of charge in

the two wells gives us the value of parameter c = 0.625 by ensuring that the height of the two wells is same.

The value of k(rate constant) is set as $4.5*10^{-5}$ after referring various papers[6,4]. It is logical to assume that the value of k’=k/h2int , where h2init is the height of bound well when battery is at its full capacity. The dependence on height h2 reflects the observation that a battery has more tendency to recover when it has more charge left.

The Unit	**Charge in available well (i)**	**Charge in Bound well (j)**
mAh	**1250 mAh**	**750 mAh**
SI unit As	**4500 As**	**2700**
mAms	**$45*10^5$**	**$27*10^5$**

Table 4.1 The charge units.

4.1.1 Calculation of charge units during discharge.

The Model can be easily extended to calculate battery life for deterministic discharge profiles. At each step of the simulation, we look at the input discharge waveform and calculate the average current over each time slot, converting it into appropriate number of charge units depending on the time unit of the simulation. The number of charge units(I) calculated are drawn from the battery model with qI = 1.

The equations governing for this transition.

$$\begin{aligned}&(i,j,0) \longrightarrow (i-I+J, j-J, 0)\\ &J = k'.h_2.(h_2-h_1)\end{aligned} \tag{4.1}$$

For example, suppose the EnergyModel requests 24.3 mA. That means 24.3*.05 charge units at per .05 s. Which gives fraction value 1.215. But the model allow us to only withdraw integer values of charge units. In order to overcome the problem, we have multiplied the both the charge unit value and the capacity with an integer. The integer value has been chosen as 10^3.Which enables us to draw current greater than100µA. So the requested charge units become 1215. The table 1.1 has been updated with this multiplication factor.

The Unit	**Charge in available well (i)**	**Charge in Bound well (j)**
mAh	**1250 mAh**	**750 mAh**
SI unit As	**4500 As**	**2700**
mAms	**$45*10^5$**	**$27*10^5$**
In charge units($*10^3$)	**$45*10^8$**	**$27*10^8$**

Table 4.2 The charge units calculation.

4.1.2 Calculation of Quanta(Q) during idle period.

If there are idle periods in between discharges, the battery can partially recover its charge during these idle times,and thus we can drain a larger number of charge units before reaching the states having parameter i = 0. During idle period recovery occurs depending on the recovery probability(pr). The equations governing for this transition are

$$(i,j,t) \quad \longrightarrow \quad \begin{cases}(i+Q, j-Q, t+1)\\ (i,j,t+1)\end{cases} \tag{4.2}$$

$$\begin{aligned}&p_r(i,j,t) = q_0.p(t)\\ &p_{nr}(i,j,t) = q_0.(1-p(t))\end{aligned} \tag{4.3}$$

The value of p(t) set as .93 based on the experimental results .

The quanta(Q) of recovery is calculated so as the charge recovered for an infinitely long idle slot is equal to total charge that needs to be transferred between the two wells before there heights are equalized.

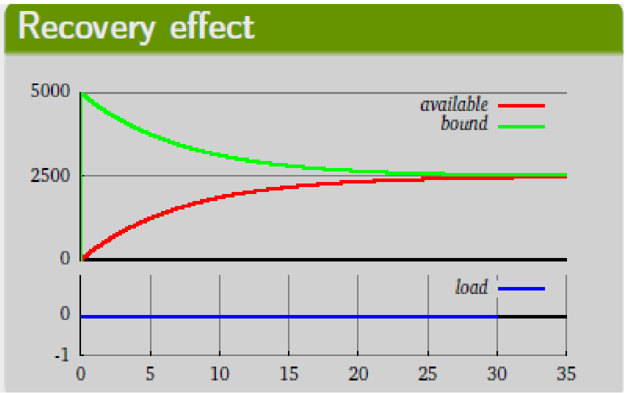


Fig 4.1 Charge transfer between wells when there is infinitely long idle slot.

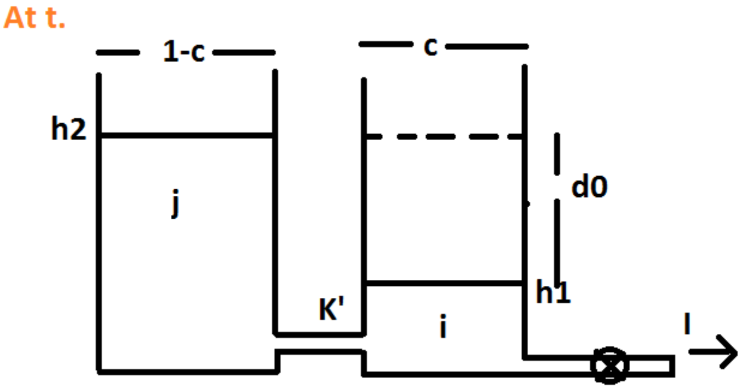


Fig 4.2 The two well model at t.

The Fig 4.1 shows the charge distribution in two well when the idle period starts. After the granularity in time (.05s), some charges has been transferred to available well from bound well. The difference (h1'-h1)*c gives the total charge transferred (Quanta [Q]) during the least count of simulation time.

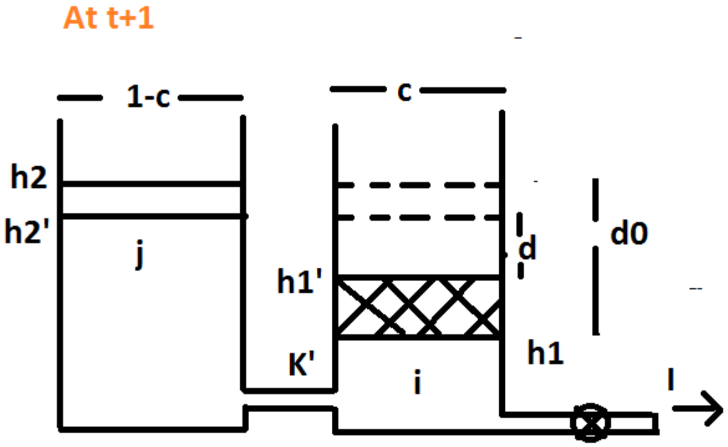


Fig 4.3 The two well model at t+1.

The d1 is calculated as follows.

The equations can be rewritten as [4]

$\frac{d\gamma}{dt} = -i(t)$ (4.4)

$\frac{d\delta}{dt} \approx \frac{i(t)}{c} - k'h2\delta$ (4.5)

When there is no discharge current [4].

$\frac{d\delta}{dt} \approx -k'h2\delta$ (4.6)

Where γ is total charge (i+j);
δ is the height difference h2-h1(d0 in figure).

The height difference at any time can be found by following equation.

$\delta(t) \approx \delta(t0)e^{-h2*t*k'}$(4.7)

This can be rewritten according to fig 4.2

$d=d_0*e^{k'*h2*.05}$ (4.8)

The new height h_1'

$h_1' = \gamma$-d(1-c) (4.9)

The quanta(**Q**) is

Q=(h1'-h1)c (4.10)

After the computation the values of d0, h1 and h2 is updated and Q is added to the available charge well and Q is reduced from bound charge well.

1.1 Energy Model

The EnergyConsumer model works as follows: it sends a message with the given message length every interval regularly, unless the interval is set to 0.After sending, the node goes to sleep and wakes up again in its next interval. The first time it wakes up is a random number between 0 and interval. The other relevant parameters are

- Message length in bytes.
- Sending interval, where 0 means that the node is not sending any messages,
- only receiving.

- Energy consumption (current) in Sleep mode
- Energy consumption (current) in Receive mode
- Energy consumption (current) in Send mode
- Sending frequency.
- Minimum required voltage

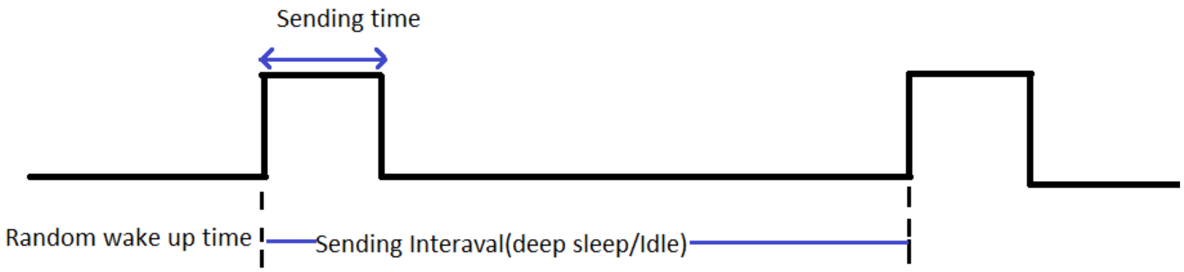


Fig 4.3 Sleep and Active cycle of EnergyModel.

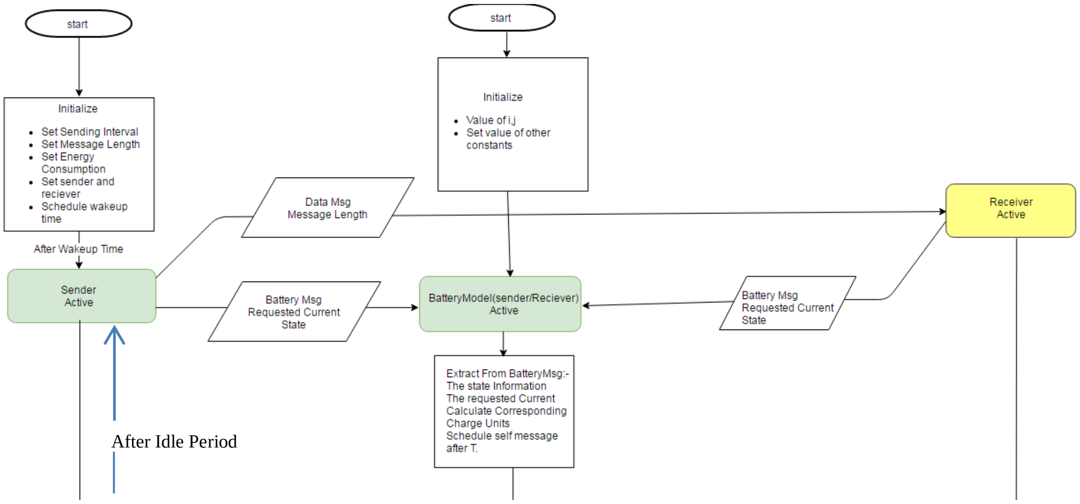
1.1.1 The relation between message length and transmission time(sending /receiving)

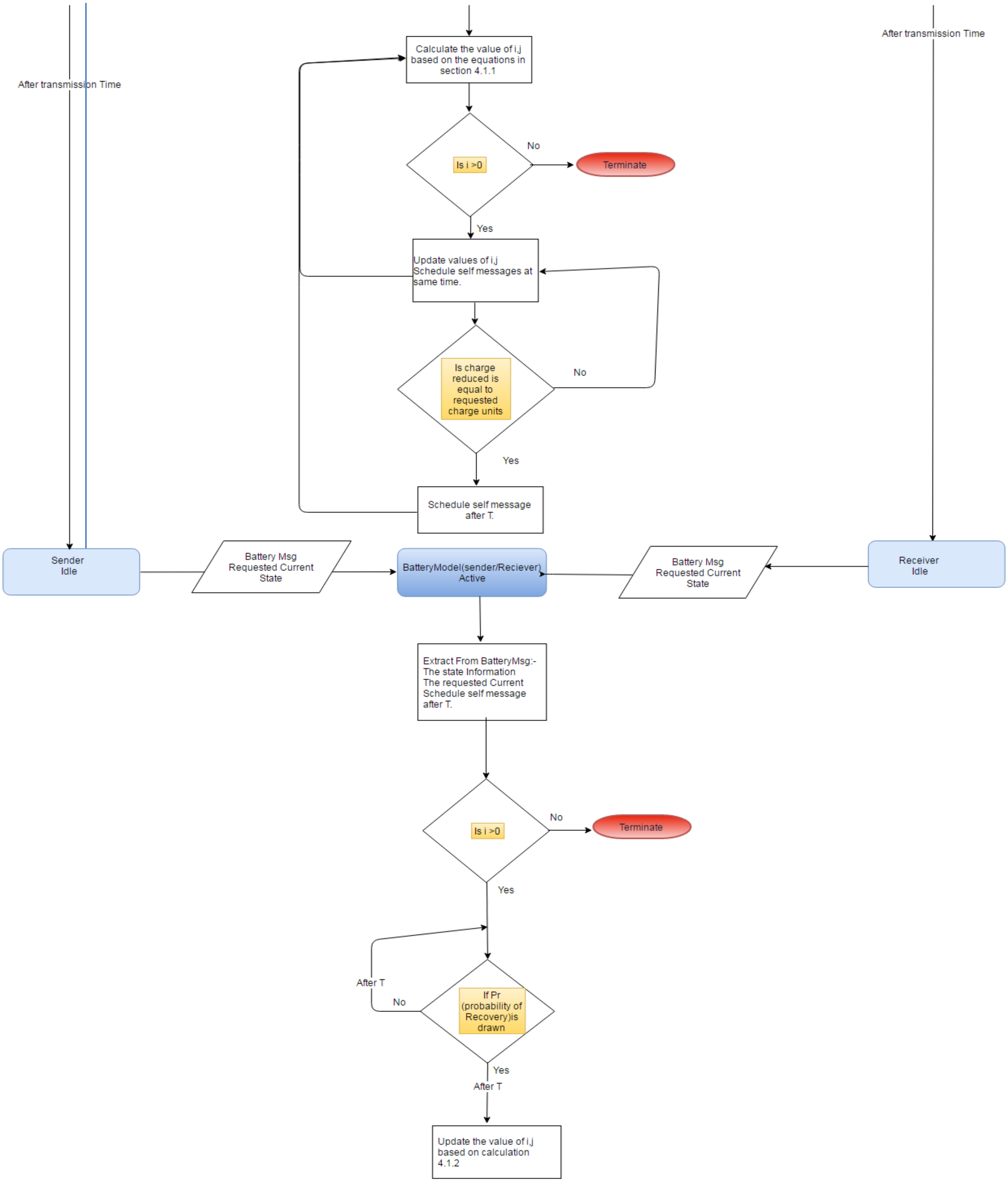
Transmission time= $\frac{Message\ Length(in\ bytes)*8}{frequency(in\frac{bits}{s})}$ (4.11)

If frequency is chosen very high the transmission time will be less than .05s, which will not be able to captured by simulation. If it is chosen very low, the transmission time will take more than 1s to transmit data. So frequency has been wisely selected to 10240 bits/s.

1.2 The Implementation in OMNET++

Flow chart here.




Calculate the value of i,j based on the equations in section 4.1.1
Is i >0
No
Terminate
Yes
Update values of i,j Schedule self messages at same time.
Is charge reduced is equal to requested charge units
No
Yes
Schedule self message after T.
After transmission Time
After transmission Time
Sender Idle
Battery Msg Requested Current State
BatteryModel(sender/Reciever) Active
Battery Msg Requested Current State
Receiver Idle
Extract From BatteryMsg:- The state Information The requested Current Schedule self message after T.
Is i >0
No
Terminate
Yes
After T
No
If Pr (probability of Recovery)is drawn
Yes
After T
Update the value of i,j based on calculation 4.1.2

CHAPTER 5

Simulation Results and Analysis

The impact of parameters sending interval and message length are studied against the battery life. Also we have studied the battery life against three different energy consumption scenarios. For easiness of taking results we have studied a 20mAh battery for the simulations. The corresponding charge units in available well and bound well are $4.5*10^7$ and $2.7*10^7$ respectively based on the calculation as described in Table 4.2.

5.1 Impact of sending interval

Here we have studied the impact of sending interval on node life time. We have set the message length to 512 bytes .We set the least count of simulation T=.1s. The following energy profile is used for the study

	CC2530 (Microcontroller & Transceiver)
Min Voltage	2.0 V
Deep Sleep	200 µA
RX	24.3 mA
TX(min)	28.7 mA
TX(max)	33.5 mA

Table 5.1 The Energy Consumption model used to study effect of sending Interval.

The result has been summarized here:-

	Sending Interval in s	Battery Life in s	Battery Life in Hours
1	1	7373.8	2.048278
2	5	28345.8	7.873833
3	10	54571	15.15861
4	15	80791.8	22.44217
5	20	107011.2	29.72533
6	25	133231.3	37.0086
7	30	159481.4	44.30039
8	35	185706.6	51.58517
9	40	211931.6	58.86989
10	45	238156.6	66.154
11	50	264431.8	73.45328
12	55	290561.8	80.71161
13	60	316892.1	88.02558

Table 5.2 The Life time vs Sending interval

A graph is drawn based on the values in MATLAB.

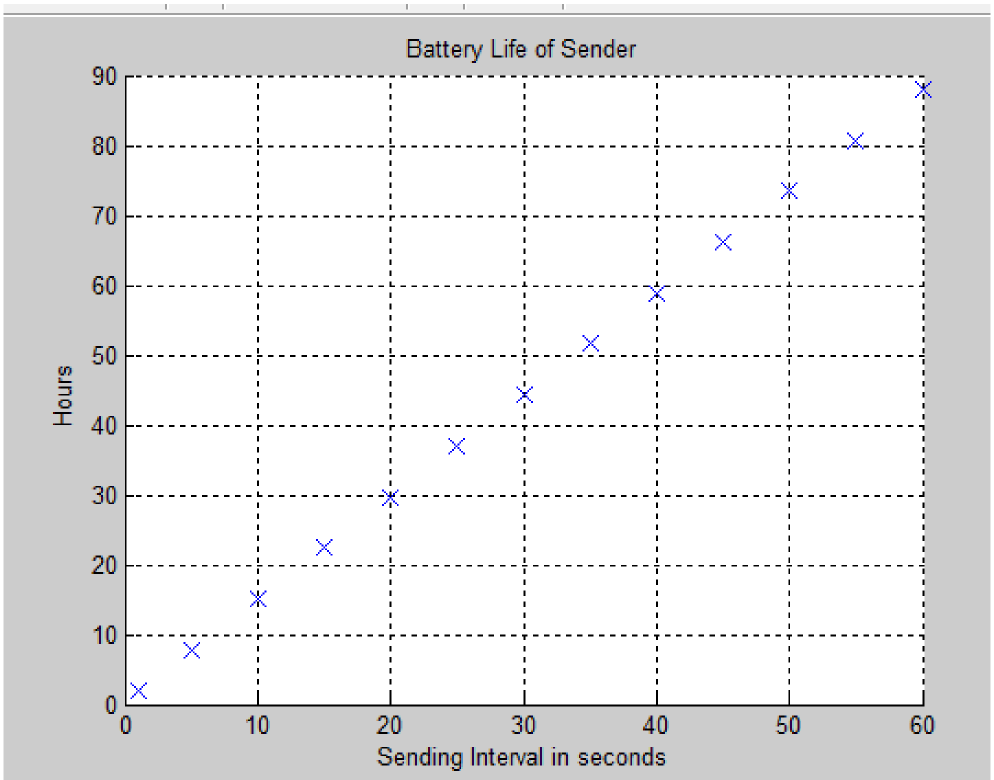


Fig 5.1 Battery Life against sending interval.

Figure 5.2 shows the capacity left in the receiver battery when the sender drained off at 1s. Sending interval

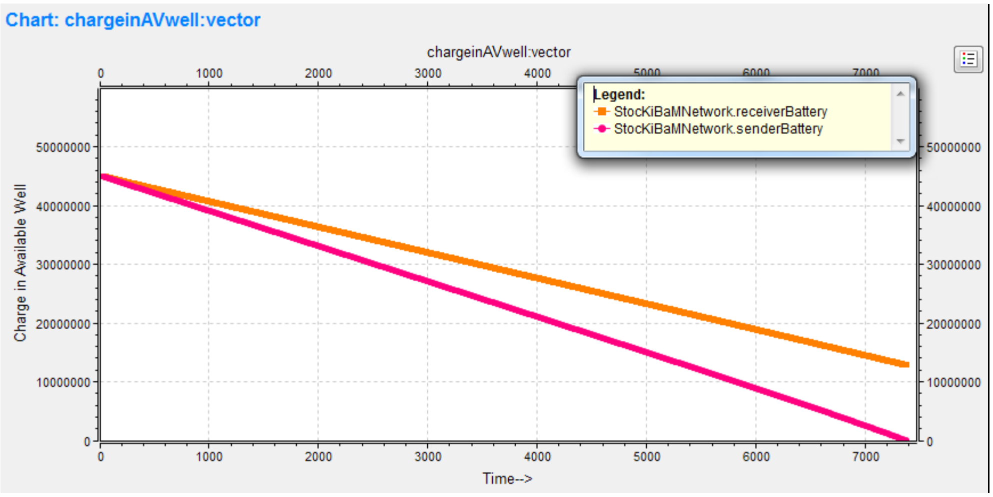


Fig 5.2 The comparison of charge drained between sender and receiver, when sending interval is 1s

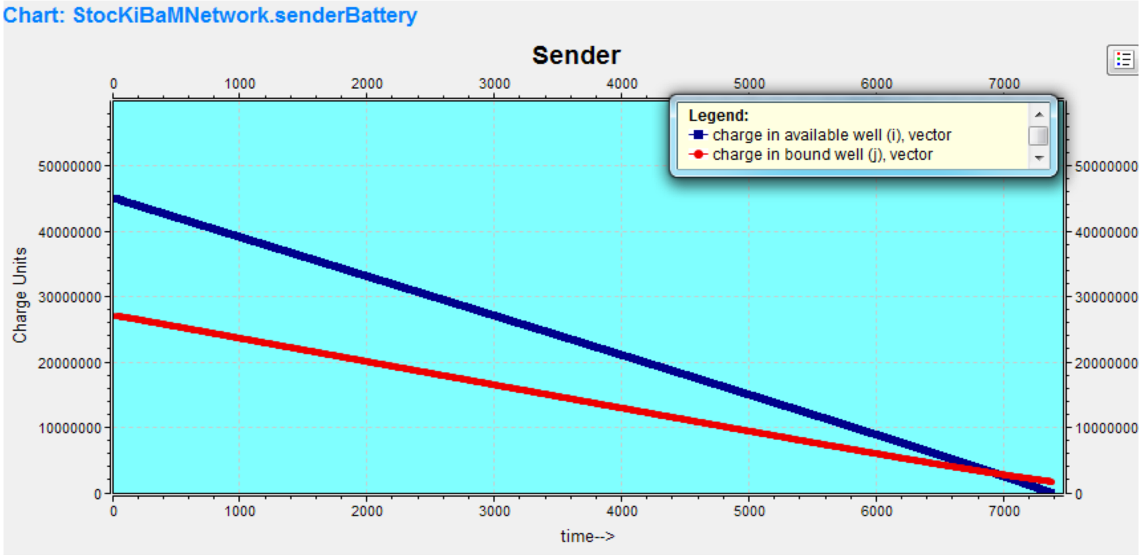


Fig 5.3 The comparison of charge in available well and bound well, when sending interval is 1s

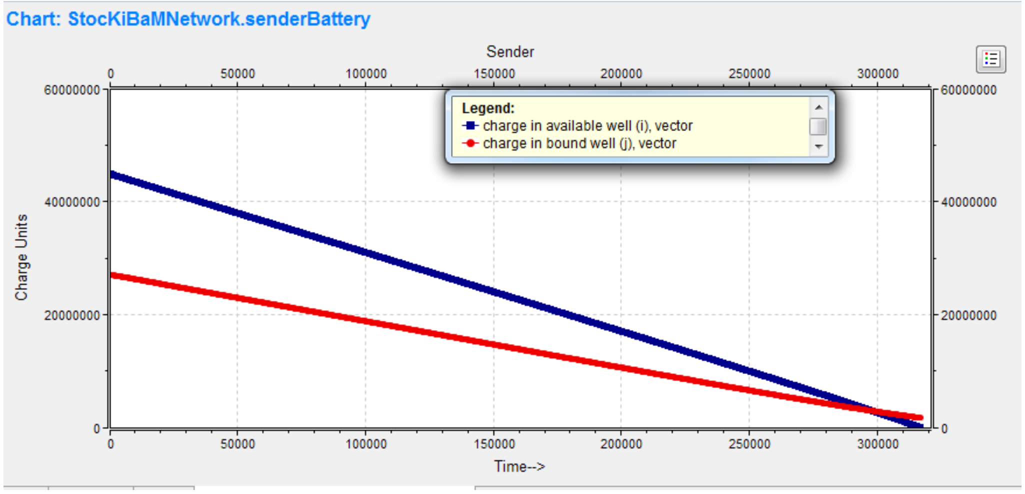


Fig 5.4 The comparison of charge in available well and bound well, when sending interval is 60s

If you compare the graphs 5.2 and 5.3, it shows that the battery life has been considerably increased from 2 hours to 88 hours. It is due to the fact of recovery effect. When sending interval is 60s, the available well get enough time to replenish the charge from the bound well.

5.2 Impact of Message Length.

Here we have studied the impact of message length on node life time. We have set the sending interval to 20 s. The energy profile used was same as previous one(Table 5.1).

The results is summarized in following table

	Sending Interval in s	Battery Life in s	Battery Life in Hours
1	64	1263335.4	350.92
2	128	632133.4	175.59
3	192	421982.6	117.21
4	256	316922.17	88.03
5	320	253952	70.54
6	384	211951.87	58.87
7	418	181981.729	50.55
8	512	159481.47	44.30
9	576	141991.57	39.44
10	640	128265.7	35.62
11	704	116551.4	32.37
12	768	107011.52	29.72
13	832	98941	27.48
14	896	92041.4	25.56
15	960	86041.4	23.90
16	1024	80791.17	22.44

Table 5.3 The Life time vs Sending interval

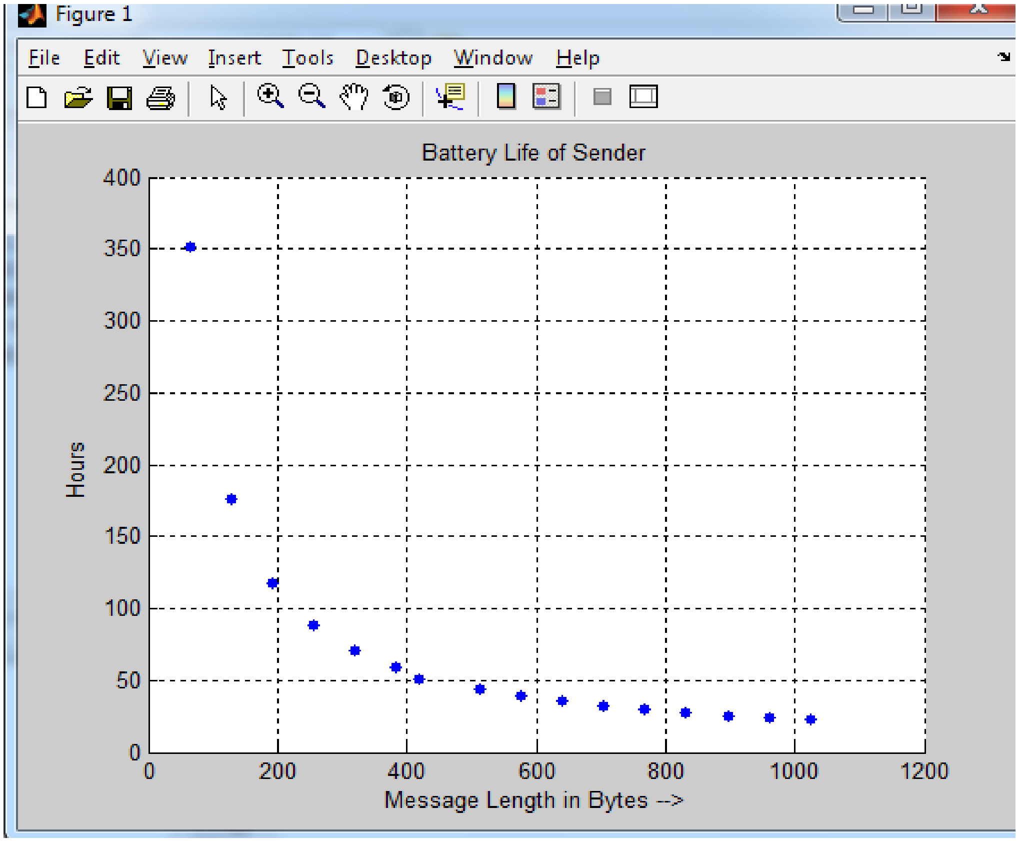


Fig 5.5 Battery Life against message length.

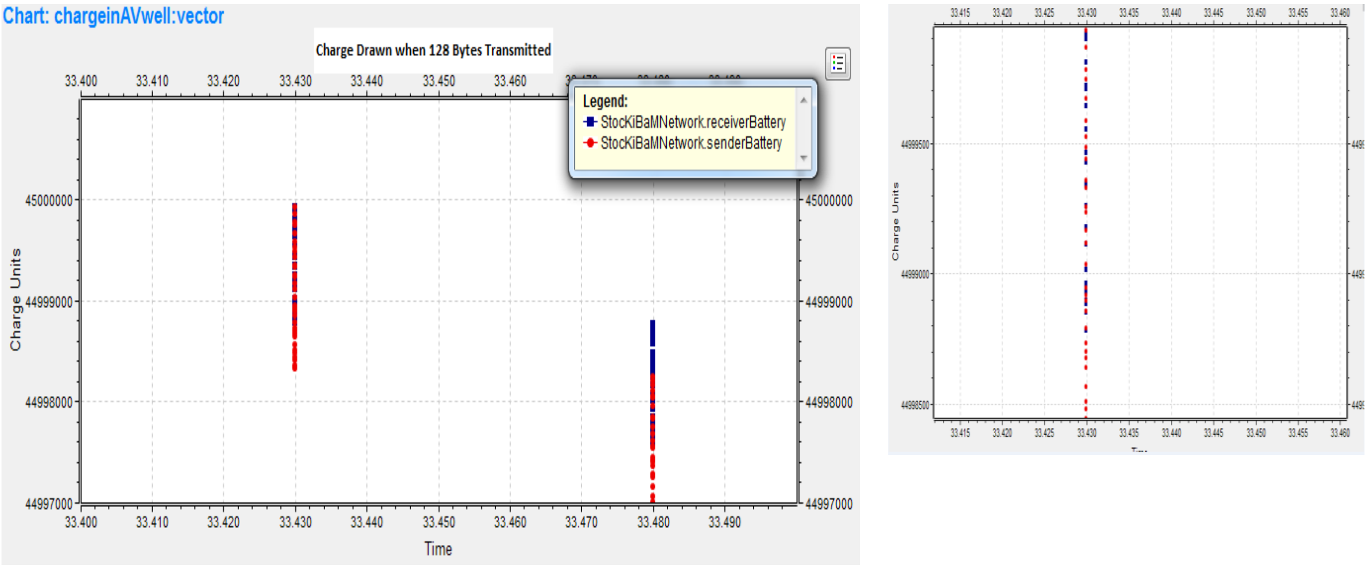


Fig 5.6 The discharge profile at first time nodes are active when 128 Bytes transmitted; expanded view on right.

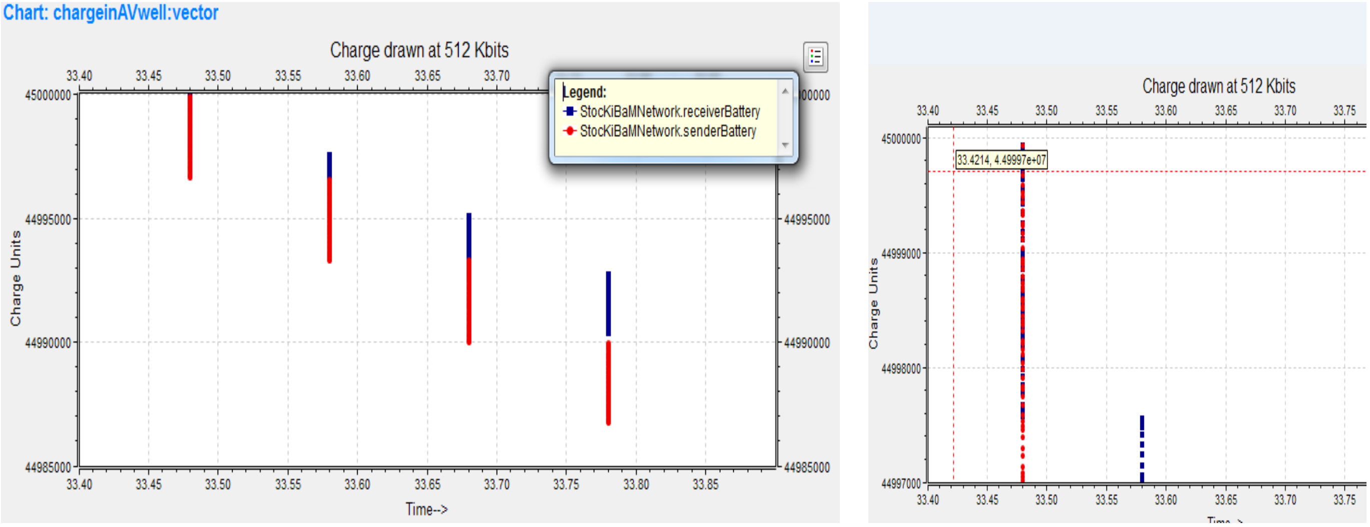


Fig 5.7 The discharge profile at first time nodes are active when 512 Bytes transmitted; expanded view on right.

The Number of charge Units drawn are based on the calculation explained in the section 4.1.1.As message length increases the time to transmit message will increases. So the node will draw more current. This would contribute to the life time of battery. Fig 5.6, 5.7 and 5.8 indicate this fact. In figure 5.6 100ms is taken to transmit the data where as in 5.8 8ms is taken to transmit the data. As a result the 128 byte node has almost eight times battery life.

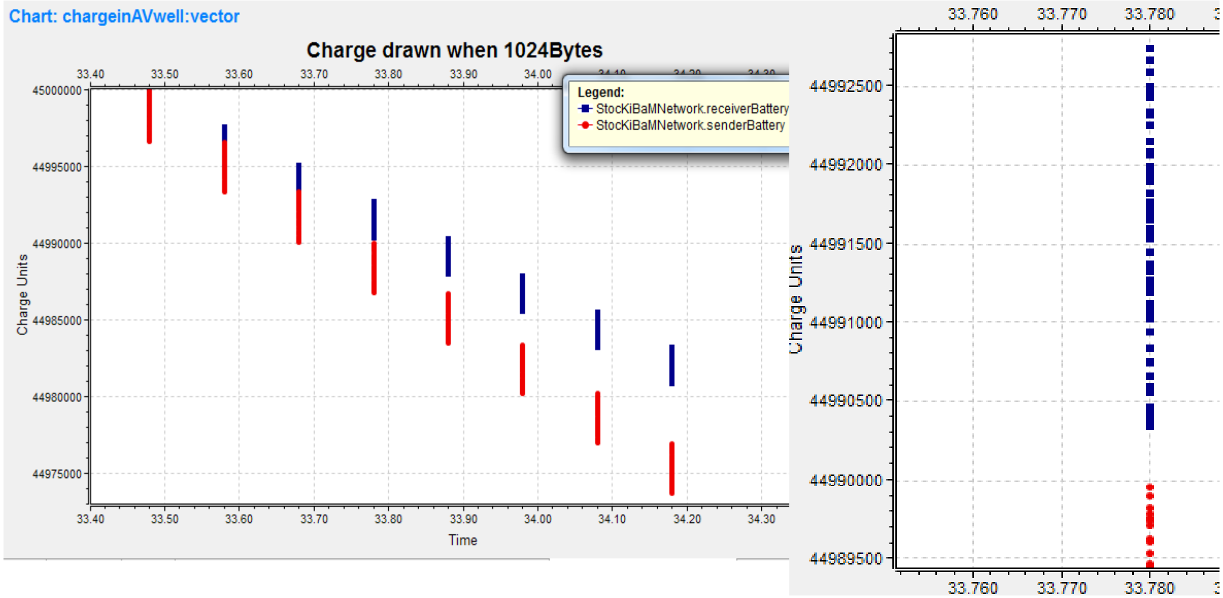


Fig 5.8 The discharge profile at first time nodes are active when 1024 Bytes transmitted; expanded view on right.

5.3 Impact of Message Length.

Three different energy profiles were used for the simulation. Here, we have set the sending interval to 20 s and 1024 Bytes message length. The details of energy profile are given below.

	CC2530 (Microcontroller& Transceiver)	ESP8266c WIFI	CC2650 BLE & 802.15.4
Profile Number	1	2	3
Min Voltage	2.0 V	1.7 V	1.8 V
Deep Sleep	200 µA	10 µA	1 µA
RX	24.3 mA	60 mA	6.1 mA
TX(min)	28.7 mA	135 mA	6.1 mA
TX(max)	33.5 mA	215 mA	9.1 mA

Table 5.4 The Energy Consumption Profile used for study.

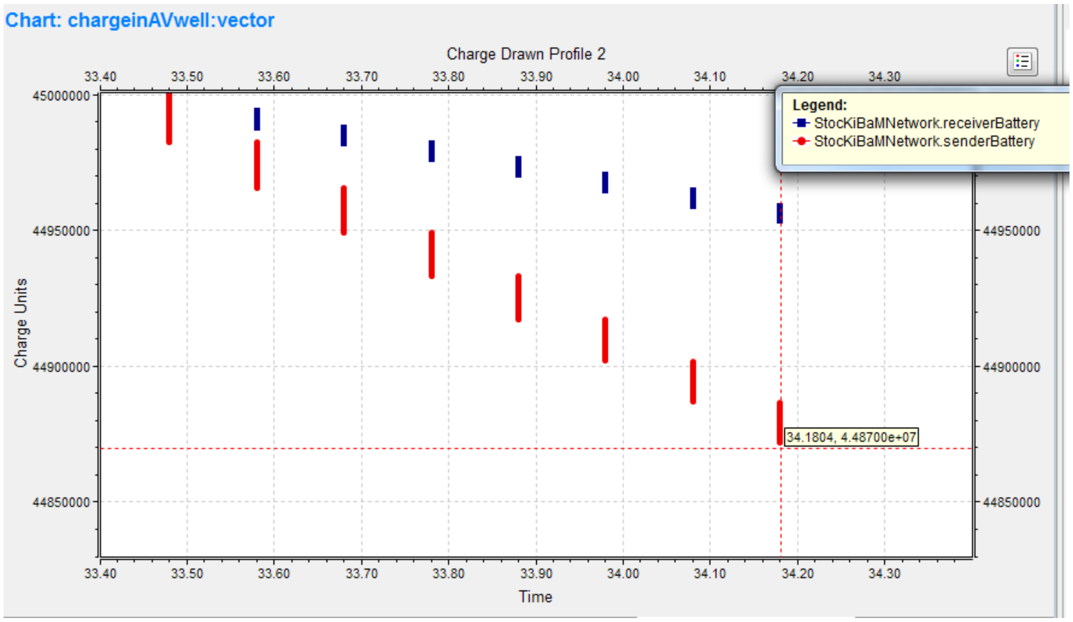


Fig 5.9 The discharge pattern at first time (after wake up time) using profile2

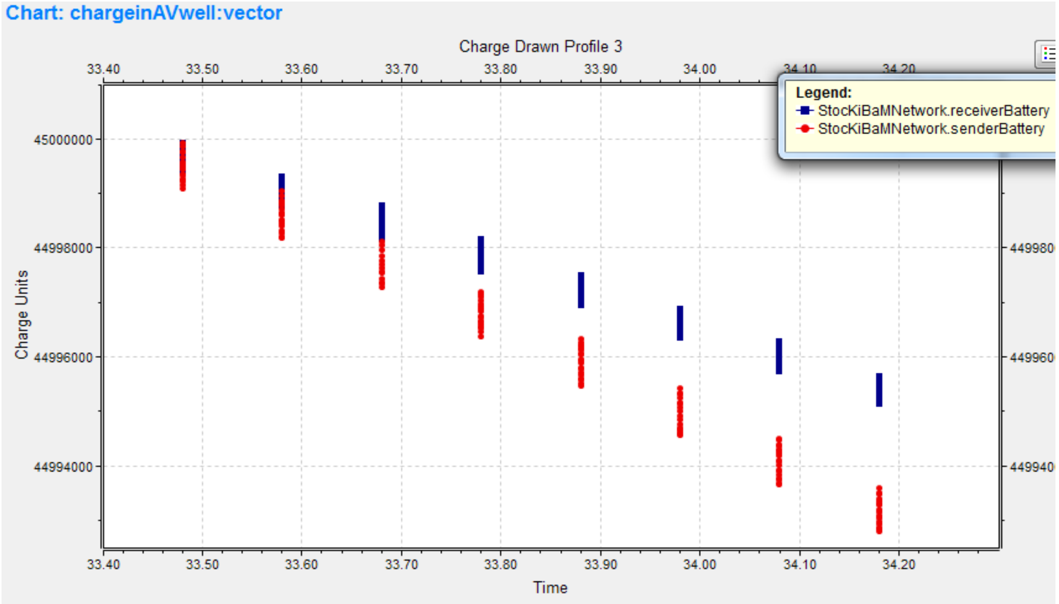


Fig 5.10 The discharge pattern at first time (after wake up time) using profile3

The figures 5.9 and 5.10 show the disparity of charge drawn when different energy consumption profiles were employed. More current has been drawn in the first case (profile 2). As a result the life time of profile 2 is considerably decreased.

Profile Number	Profile 1	Profile 2	Profile 3
Battery Life in s	80760.9	15464.87	297531.4
Battery Life in hours	22.44	4.29	82.64

Table 5.5 The Life time vs Sending interval.

The Table 5.5 clearly indicates the variation Battery life with regards to energy profile. The dominant factor contribute to this effect is the rate capacity effect. The lifetime of a cell depends on the availability and reachability of active reaction sites in the cathode. When discharge current is low, the inactive sites (made inactive by previous cathode reactions) are distributed uniformly throughout the cathode. But, at higher discharge current, reductions occur at the outer surface of the cathode making the inner active sites inaccessible. Hence, the energy delivered (or the battery lifetime) decreases since many active sites in the cathode remain un-utilized when the battery is declared discharged. As the intensity of the current is increased, the deviation of the concentration from the average becomes more significant and the state of charge as well as the cell voltage decrease. This phenomenon is called Rate Capacity effect.

Here, charge in the available well decreases more rapidly giving the bound charge well less time to replenish the available charge well and battery is declared to be discharged even though there is a lot of charge left inside the bound charge. In figure 5.7 the red line indicates the unused capacity.

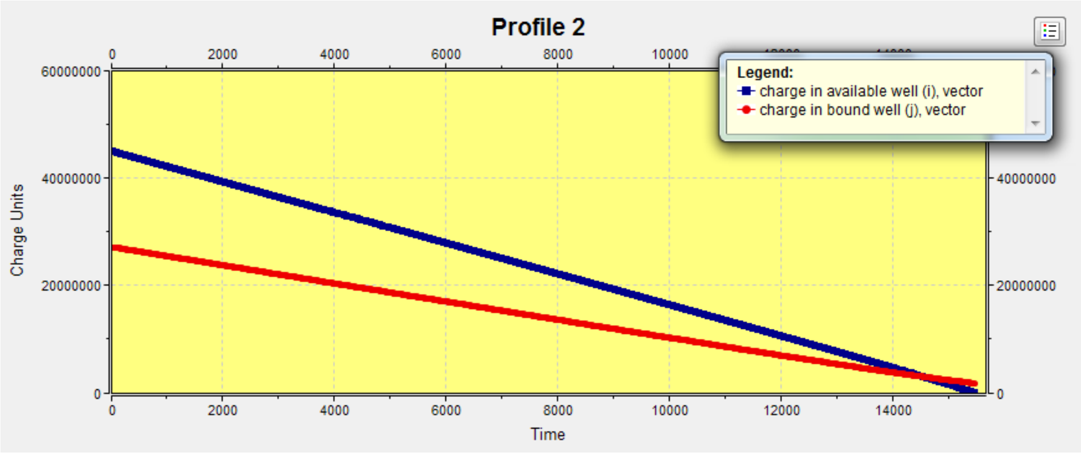


Fig 5.11 Discharge profile 2 v/s Battery Life.

Conclusion

In this project, the various aspects of Modified Stochastic KiBaM model have been studied. We have created a mathematical model and implemented it in the OMNET++. However there are few restrictions involved, the program gives almost accurate implementation of the Modified Stochastic KiBaM model. Simulation runs were carried out and we have studied the effect on battery life by various parameters such as message sending interval, message length and energy consumption profile. We have analyzed these three parameters separately keeping others constant.

The results of studies conducted on sending interval lead to the conclusion that as sending interval increases the battery life increases. It is not only by the increase of sending interval but also due to the Recovery effect. During the idle period battery is trying to regain the capacity from the charge available in bound well. So longer the length of idle period greater the battery life. The results of message length v/s battery life suggest that battery life decreases as message length increases. Which is due to the increase in transmission time. Also simulation runs conducted against various discharge profile. When the value of discharge profile increases, the life time of battery decreases considerably The dominant effect contribute to this is the Rate capacity effect. Here the charge in the available well decreases more rapidly giving the bound charge well less time to replenish the available charge well and battery is declared to be discharged even though there is a lot of charge left inside the bound charge.

In conclusion the Modified Stochastic KiBaM model serve as good reference model for those who want to study the effect of Rate capacity effect and Recovery effect. Also, since it is based on Markov process it is easier to implement in DES such as OMNET++.

REFERENCE

[1] *V. Rao, G. Singhal, A. Kumar, and N. Navet, "Battery model for embedded systems," in Proceedings of the 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design (VLSID'05). Washington,DC, USA: IEEE Computer Society, 2005, pp. 105–110.*

[2] *J. Manwell and J. McGowan, "Lead acid battery storage model for hybrid energy systems," Solar Energy, vol. 50, pp. 399–405, 1993.*

[3] *——, "Extension of the kinetic battery model for wind/hybrid power systems," in Proceedings of the 5th European Wind Energy Association Conference (EWEC '94),1994, pp. 284–289.*

[4] *Model-based energy analysis of battery powered systems, Marijn Remco Jongerden CTITPh.D.-thesis Series No. 10-183.Centre for Telematics and Information Technology, University of Twente.*

[5] *D. Panigrahi, C. Chiasserini, S. Dey, R. Rao, A. Raghunathan, and K. Lahiri. Battery Life Estimation of Mobile Embedded Systems. In Proceedings of International Conference on VLSI Design, pages 55–63, January 2001.*

[6] Which battery model to use? Marijn R. Jongerden*!* Boudewijn R. Haverkort. *A. Argent-Katwala, N.J. Dingle and U. Harder (Eds.): UKPEW 2008, Imperial College London, DTR08-9*
pp. 76–88, http://ukpew.org

APPENDIX

The Code in OMNET

```
//Energy consumer simple module
simple EnergyConsumer
{
    parameters:
        @display("i=device/antennatower");
        double dMessageSendingInterval;
        int SendingFrequency; // bits/s
        double MessageLength; //bytes


        volatile double nextMsgGenInterval @unit(s) =  default(uniform(1.0s, 60.0s));

        int sending_interval;
        double energy_consumption_sleep;
        double energy_consumption_send;
        double energy_consumption_rcv;
        double dMinBatteryLevel;
    gates:
        input in;
        input in1;
        output out;
        output out1;
}


// Battery Model simple module

package final;

simple BatteryModel
{


parameters:
       @display("i=block/routing");
        double chargeinAvailablewell;
        double chargeinBoundwell;
        double dRateconstant;
        double dProbabilityrecovery;
        double dCapacityratio;
        double dLeastCountTime;

        @signal[av_well](type="double");
        @statistic[chargeinAVwell](title="charge in available well (i)";
source="av_well"; record=vector,stats; interpolationmode=none);
       @signal[bound_well](type="double");
        @statistic[chargeinBDwell](title="charge in bound well (j)"; source="bound_well";
record=vector,stats; interpolationmode=none);


      gates:
        input in;
        output out;
}
```

//network file

```
package final;

network StocKiBaMNetwork
{
    types:
        channel Channel extends ned.DelayChannel
        {
            delay = 100ms;
        }
    submodules:
        sender: EnergyConsumer {
            @display("p=234,50");
        }
        receiver: EnergyConsumer {
            @display("p=333,249");
        }
        senderBattery: BatteryModel {
            @display("p=103,100");
        }
        receiverBattery: BatteryModel {
            @display("p=490,249");
        }
    connections:
        sender.out --> {  delay = 0ms; } --> receiver.in;
        sender.in <-- {  delay = 0ms; } <-- receiver.out;
        sender.out1 --> {  delay = 0ms; } --> senderBattery.in;
        sender.in1 <-- {  delay = 0ms; } <-- senderBattery.out;
        receiver.out1 --> {  delay = 0ms; } --> receiverBattery.in;
        receiver.in1 <-- {  delay = 0ms; } <-- receiverBattery.out;

}

//Battery Message
packet BatteryMsg {
    int state;
    double current;
}

//Data message

packet DataMsg {
    double dMsgLength;
     bool senderalive;
}

//Energy .h
#ifndef __ENERGYCONSUMPTION_DEVICENODE_H_
#define __ENERGYCONSUMPTION_DEVICENODE_H_
#include <omnetpp.h>
//using namespace omnetpp;

#include <stdio.h>
#include <string.h>
#include <math.h>

#include "dataMsg_m.h"
#include "Battery.h"
#include "batteryMsg_m.h"


/**
 * TODO - Generated class
 */
class EnergyConsumer : public cSimpleModule
```

```
{
private:
    //consumer parameters
    int iMessageInterval;        // message sending interval
    double dMessageLength;          // message length in bytes
    double dEnergyUsedInSleep;      // energy used in sleep
    double dEnergyUsedInSend;       // energy consumed in send mode
    double dEnergyUsedInReceive;    // energy consumed while sending
    double dMinimumBatteryLevel;    // The cutoff freequency
    int iSendingFrequency;       // sending Frequency
    int state;
    double frequency;
    double transmissionTime;

    cMessage *eventActive;  // pointer to the event object which we'll use for timing
    cMessage *eventSleep;  // pointer to the event object which we'll use for timing
    DataMsg *dataMsg;  // variable to remember the message until we send it back
    BatteryMsg *batteryMsg;



    simsignal_t batterySignal;

protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
    virtual void changeDisplay();



public:
    void doSomething(); //Termination of Energy consumer module act like destructor
    EnergyConsumer();
    virtual ~EnergyConsumer();
};

#endif




//Energy.cc

#include "Energy.h"

Define_Module(EnergyConsumer);

EnergyConsumer::EnergyConsumer()
{
    eventActive= eventSleep = dataMsg = nullptr;  // set variables to null
}


EnergyConsumer::~EnergyConsumer()
{
    cancelAndDelete(eventActive);         // cancel and delete event object
    cancelAndDelete(eventSleep);
    delete dataMsg;             // delete message when destructor is called
}

void EnergyConsumer::initialize()
{
    // TODO - Generated method body

    dataMsg = nullptr;
    batteryMsg=nullptr;
    eventActive=new cMessage();    // event is used to schedule wakeup
    eventSleep = new cMessage();    // used to schedule sleep



    // consumer parameters
    iMessageInterval = 0;       // set message sending interval
    dMessageLength = 0;         //  message length in bytes

    dEnergyUsedInSend=par("energy_consumption_send");
    dEnergyUsedInReceive=par("energy_consumption_rcv");
    iSendingFrequency = par("SendingFrequency");

    state=0;



    changeDisplay();      // show the current state

    double dWakeupTime = par("nextMsgGenInterval");   // get a random number between 0
and interval, wake up device if not 0
    // double dWakeupTime=120;
    iMessageInterval=par("sending_interval");
    dMessageLength=par("MessageLength");
    frequency=par("SendingFrequency");
    transmissionTime=(dMessageLength*8)/frequency;
    scheduleAt((simTime() + dWakeupTime), eventActive);
    EV << "sending time :"  << transmissionTime<<"\n";

}

void EnergyConsumer::handleMessage(cMessage *msg)
{

    if(msg == eventActive)  // if it is a wake up self message
    {
        // if message interval is greater than 0, node is a sender
        if(iMessageInterval>0)
        {
            state=1;
            changeDisplay();
            // send message
            dataMsg = new DataMsg();   //create data message
            dataMsg->setByteLength(dMessageLength); //set the message length
            send(dataMsg,"out");  // send message
            batteryMsg= new BatteryMsg();   //notify the battery module
            batteryMsg->setCurrent(dEnergyUsedInSend);  //set the parameters, current and
state
            batteryMsg->setState(state);
            send(batteryMsg,"out1");
            simtime_t sending;
            sending=batteryMsg->getSendingTime();   //to print the sending time
            EV << "sending time: " << sending << "\n";



            // go to sleep after sending the message, depending on transmission time
            if(!eventSleep->isScheduled())
            {
                scheduleAt((simTime() +transmissionTime), eventSleep);  // schedule self
message
            }
        }
        else
        {
            // node receives only
            // schedule self message
            if(!eventSleep->isScheduled())
            {
                scheduleAt((simTime() + transmissionTime), eventSleep);  // schedule
sleep for rcv node
            }
        }

    }
    //check whether self message for sleep/idle
    else if(msg == eventSleep)
    {
        if(iMessageInterval>0)
        {
            state=0;
            changeDisplay();
            batteryMsg= new BatteryMsg();         //Notify battery module about state
change
            batteryMsg->setState(state);
            batteryMsg->setCurrent(0);
            send(batteryMsg,"out1");


            if(!eventActive->isScheduled())                   //if active is not
schedule, scheduling it
            {
                scheduleAt((simTime() + iMessageInterval), eventActive);
            }
        }
        //for receiver, no need to schedule next active
        else
        {
            state=0;
            changeDisplay();
            batteryMsg= new BatteryMsg();
            batteryMsg->setState(state);
            batteryMsg->setCurrent(0);
            send(batteryMsg,"out1");

        }
    }

    //To handle the messages from other module
    else
    {
        //To handle the message from battery module notifying the termination.
        if(strcmp(msg->getName(),"battery") == 0)
        {
            EV << "Message from battery module arrived" << msg << " arrived.\n";
            batteryMsg=check_and_cast<BatteryMsg *>(msg);
            cancelAndDelete(eventSleep);
            cancelAndDelete(eventActive);
            eventSleep=eventActive=nullptr;
            //delete msg;
        }
        //To handle the message from Energy module
        else
        {
            state=2;
            changeDisplay();
            EV << "Message from sender has been arrived" << msg << " arrived.\n";
            batteryMsg= new BatteryMsg();
            batteryMsg->setState(state);
            batteryMsg->setCurrent(dEnergyUsedInReceive);
            send(batteryMsg,"out1");

            if(!eventSleep->isScheduled())
            {
                scheduleAt((simTime() + transmissionTime), eventSleep);  // schedule
sleep for node
            }
            //delete msg;
        }
        delete msg;
    }



}
//To display the value of state
//state 0 means idle, 1 means sending, 2 receiving
void EnergyConsumer::changeDisplay()
{

    char buffer[30];

    if(state==1)
    {
        sprintf(buffer,"The node is in state:Sending");
        getDisplayString().setTagArg("t",0,buffer);


        getDisplayString().setTagArg("i",1,"green");
        getDisplayString().setTagArg("t",2,"green");

        getDisplayString().setTagArg("i2",0,"status/up");
    }

    else if(state==0)
    {
        sprintf(buffer,"The node is in state:Idle");
        getDisplayString().setTagArg("t",0,buffer);


        getDisplayString().setTagArg("i",1,"blue");
        getDisplayString().setTagArg("t",2,"blue");

        getDisplayString().setTagArg("i2",0,"status/down");

    }

    else
    {
        sprintf(buffer,"The node is in state:receiving");
        getDisplayString().setTagArg("t",0,buffer);


        getDisplayString().setTagArg("i",1,"yellow");
        getDisplayString().setTagArg("t",2,"yellow");

        getDisplayString().setTagArg("i2",0,"status/up");
    }
}
```

```
//Termination of Energy consumer module act like destructor
void EnergyConsumer::doSomething()
{
    if(eventActive->isScheduled())
    {
        cancelAndDelete(eventActive);
    }

    if(eventSleep->isScheduled())
    {
        cancelAndDelete(eventSleep);
    }
}
```

//Battery.h

```
#ifndef __KIBAM_BATTERYMODEL_H_
#define __KIBAM_BATTERYMODEL_H_

#include <omnetpp.h>
#include <math.h>

#include "batteryMsg_m.h"
#include "Energy.h"

/**
 * TODO - Generated class
 */
class BatteryModel : public cSimpleModule
{
protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);


public:
    virtual void changeDisplay(double h4); //to display
    virtual void terminate(); // to finish the battery module
    void doSomething();    //act like destructor
    BatteryModel();
    virtual ~BatteryModel();
private:
    double h1;
    double h2;
    double i;
    double j;
    double c;
    double requestedcurrent;
    double requestedchargeunits;
    double pi;
    double r;
    double k;
    double kp;
    double ks;
    double T;
    double d0;
    double d;
    double qtotal;
    double hd;
    double tempcharge;
    double charge;
    double J;
    double sJ;
    double sI;

    int  Integer_multiple;
    int current_state;
    int qi;
    int I;
    //int count;
    BatteryMsg *patteryMsg;
    cMessage *eventSleep;
    cMessage *eventActive;
    // cMessage *eventdelay;
    simsignal_t availablewellsignal;
    simsignal_t boundwellsignal;
    simtime_t t;
    cModule *targetModule1;
    cModule *targetModule2;
};

#endif
```

//Battery.cc

```
#include "Battery.h"

Define_Module(BatteryModel);
BatteryModel::BatteryModel()
{
    eventActive= eventSleep = patteryMsg = nullptr;  // set variables to null
}


BatteryModel::~BatteryModel()
{
    cancelAndDelete(eventActive);         // cancel and delete event object
    cancelAndDelete(eventSleep);
    delete patteryMsg;              // delete message when destructor is called
}

//to initialize parameters
void BatteryModel::initialize()
{
    c=par("dCapacityratio");
    k=par("dRateconstant");

    i=par("chargeinAvailablewell");
    h1=i/c;

    j=par("chargeinBoundwell");
    h2=j/(1-c);

    I=0;

    kp=k/(c*(1-c));
    ks=k/h2;
    charge=0;
    qtotal=i+j;
    sJ=0;
    sI=0;
    current_state=0; //state 0 means idle, 1 means sending, 2 receiving

    T=par("dLeastCountTime");

    requestedcurrent=0;
    requestedchargeunits=0;
    Integer_multiple=1000;
    patteryMsg=nullptr;

    availablewellsignal = registerSignal("av_well");
    boundwellsignal=registerSignal("bound_well");
    changeDisplay(i);
    EV << "The i is"  << i<<"\n";
    EV << "The j is"  << j<<"\n";

    emit(availablewellsignal, i);
    emit(boundwellsignal, j);
    t=simTime();
    EV << "the current time"  << t<<"\n";
    eventSleep = new cMessage();
    eventActive=new cMessage();

}

void BatteryModel::handleMessage(cMessage *msg)
{

    // check if it is self message for sleep
    //The calculation details explained in section 4.1.2 in report
    if(msg==eventSleep)
    {
        //check the exit condition, if i<=0 the simulation ends

        if(i>0)
        {

            pi=uniform(0,100);
            EV << "Node is in sleep "  << "\n";
            EV << "the value of pi:" <<pi<< "\n";

            // check the probability of recovery, if condition is satisfied battery
module would recover partially.
            if(pi<93)
            {

                //The calculation details explained in section 4.1.2 in report
                d0=h2-h1;
                qtotal=i+j;

                r=exp(-(ks*T*h2));
                d=d0*r;

                hd=qtotal-(d*(1-c));

                /*The following code allow to deduct only integer values from i and j,
                 * The values have been accumulated in sI*/


                sI=sI+((hd-h1)*c);

                J=floor(sI);
                sI=sI-J;
                EV << "Value of J: " << J<< "\n";
                i=i+J;
                j=j-J;

                t=simTime();
                EV << "Value of i: " << i << "\n";
                EV << "Value of j: " << j << "\n";
                EV << "Value of time: " << t << "\n";
                h1=i/c;
                h2=j/(1-c);

                //collecting statitics
                emit(availablewellsignal, i);
                emit(boundwellsignal, j);

                if (i>0)
                {
                    changeDisplay(i);

                    //scheduling next sleep after T.

                    if(!eventSleep->isScheduled())
                    {
                        scheduleAt(simTime()+T,eventSleep);
                    }



                }
                // if i<=0 the simulation ends, code to end the simulation
                else
                {
                    i=0;
                    changeDisplay(i);
                    // terminate();

                }


            }
            //when no recovery
            else
            {
                t=simTime();

                EV << "Value of i: " << i << "\n";
                EV << "Value of j: " << j << "\n";
                EV << "Value of time: " << t << "\n";

                if(!eventSleep->isScheduled())
                {
                    scheduleAt(simTime()+T,eventSleep);
                }


            }

        }
        // if i<=0 the simulation ends, code to end the simulation
        else
        {
            i=0;
            changeDisplay(i);
            terminate();

        }

    }


    // check if it is self message for sleep
```

```
//The calculation details explained in section 4.1.1 in report
else if(msg==eventActive)
{



    //check the exit condition, if i<=0 the simulation ends
    if(i>0)
    {
        //the loop will be executed until battery delivers the requested charge unit.

        if (charge<=requestedchargeunits)
        {
            //qI is the probability that in one time
            //unit, called a time slot, I charge units are demanded

            qi=intuniform(1,100);
            I=qi;
            d0=h2-h1;
            J=ks*d0*h2;

            // EV << "Value of J: " << J << "\n";
            /*The following code allow to deduct only integer values from  j,
                             * The values have been accumulated in sJ*/
            sJ=sJ+J;
            J=floor(sJ);
            sJ=sJ-J;
            i=i+J;
            j=j-J;
            charge=charge+I;
            if(charge<requestedchargeunits)
            {
                i=i-I;
                if(i<=0)
                {
                    i=0;
                    changeDisplay(i);

                }
            }

            else
            {
                charge=charge-requestedchargeunits;
                I=I-charge;
                i=i-I;
                if(i<=0)
                {
                    i=0;

                    changeDisplay(i);
                }
                charge= requestedchargeunits+1;
                // count++;
            }
            h1=i/c;
            h2=j/(1-c);

            t=simTime();

            EV << "Value of t: " << t << "\n";
            EV << "The i is"  << i<<"\n";
            EV << "The j is"  << j<<"\n";
```

```
                emit(availablewellsignal, i);
                emit(boundwellsignal, j);


                changeDisplay(i);

                //schedule self message to same instant until battery delivers the
requested charge unit.

                if(!eventActive->isScheduled())
                {
                    scheduleAt(simTime(),eventActive);
                }
            }
            //schedule self message after simulation instance T.
            else {
                charge=0;
                scheduleAt(simTime()+T, eventActive);
            }
        }


        // if i<=0 the simulation ends, code to end the simulation
        else
        {
            changeDisplay(i);
            i=0;
            terminate();
        }

    }

    //to handle message recvd from Energy Consumption module

    else {
        // received message from the Energy module

        EV << "Message " << msg << " arrived.\n";


        simtime_t arrival_time;

        //receive and extract parameters
        patteryMsg=check_and_cast<BatteryMsg *>(msg);
        arrival_time=patteryMsg->getArrivalTime();
        EV << "rcv time: " << arrival_time<< "\n";

        current_state= patteryMsg->getState();
        requestedcurrent=patteryMsg->getCurrent();

        requestedchargeunits=requestedcurrent*Integer_multiple*T;
        EV << "requestedchargeunits are" <<requestedchargeunits<< "\n";


        if(current_state==0)
        {
            EV << "The energy consumer is Idle" << "\n";

            if(!eventSleep->isScheduled())
            {
                //sI=0;
                scheduleAt(simTime()+(T-.0001), eventSleep);  // schedule sleep for node
                                                                //.0001 to adjust timing
            }



            if(eventActive->isScheduled())
            {
                cancelAndDelete(eventActive);
                eventActive=new cMessage();// cancel the scheduled active state

            }
        }



        else
        {
            EV << "The energy consumer is Active" << "\n";
            if(!eventActive->isScheduled())      // schedule active state for the node
            {
                scheduleAt(simTime()+(T-.0001), eventActive); //.0001 to adjust timing
            }

            if(eventSleep->isScheduled())
            {
                cancelAndDelete(eventSleep);  // cancel the scheduled sleep state
                eventSleep = new cMessage();
            }


        }

        delete msg;
    }

}

//to display charge in the available well
void BatteryModel::changeDisplay(double h4)
{
    char buffer[30];
    sprintf(buffer,"available well(i): %.0f ",h4);
    getDisplayString().setTagArg("t",0,buffer);
    //    if(dCurrent >=50)
    //    {
    getDisplayString().setTagArg("i",1,"gold");
    getDisplayString().setTagArg("t",2,"blue");

}


//The code for terminating when i<0
void BatteryModel::terminate()
{

    //notify the Energy consumer module modules
    patteryMsg= new BatteryMsg();
    patteryMsg->setName("battery");

    send(patteryMsg,"out");

    //cancel the sheduled events
    if(eventActive->isScheduled())
    {
        cancelAndDelete(eventActive);
    }

    if(eventSleep->isScheduled())
    {
        cancelAndDelete(eventSleep);
    }

    //if it is sender battery module, terminate receiver module as well
    if (strcmp("senderBattery", getName()) == 0)
    {
        targetModule1=getModuleByPath("StocKiBaMNetwork.receiverBattery");
        targetModule2=getModuleByPath("StocKiBaMNetwork.receiver");

    }

    else
    {
        targetModule1=getModuleByPath("StocKiBaMNetwork.senderBattery");
        targetModule2=getModuleByPath("StocKiBaMNetwork.sender");

    }
    EnergyConsumer *target2 = check_and_cast<EnergyConsumer *>(targetModule2);
    BatteryModel *target1 = check_and_cast<BatteryModel *>(targetModule1);

    target1->doSomething();
    target2->doSomething();


}

//Act as destructor when program terminates
void BatteryModel::doSomething()
{
    if(eventActive->isScheduled())
    {
        cancelAndDelete(eventActive);
    }

    if(eventSleep->isScheduled())
    {
        cancelAndDelete(eventSleep);
    }
}



//ini file

[General]
network = final.StocKiBaMNetwork
**.dMessageSendingInterval = 1
**.SendingFrequency =10240
**.MessageLength= 1024
**.dMinBatteryLevel = 5
**.energy_consumption_sleep=0
**.energy_consumption_send=9.1
**.energy_consumption_rcv=6.1
**.sender.sending_interval = 30
**.receiver.sending_interval = 0
**.senderBattery.chargeinAvailablewell = 45000000
**.senderBattery.chargeinBoundwell = 27000000
**.senderBattery.dRateconstant = .000045
**.receiverBattery.chargeinAvailablewell = 45000000
**.receiverBattery.chargeinBoundwell = 27000000
```

```
**.receiverBattery.dRateconstant = .000045
**.senderBattery.dProbabilityrecovery = .93
**.receiverBattery.dProbabilityrecovery = .93
**.senderBattery.dCapacityratio = .625
**.receiverBattery.dCapacityratio = .625
**.senderBattery.dLeastCountTime = .1
**.receiverBattery.dLeastCountTime
```