Choosing Distribution Parameters for faster convergence in highly mobile environment

Shivam Choudhary(sc3973)[1]*

Abstract
In Ad-Hoc networks on-demand routing [1] [2] is often used. For evaluating performance often mobility models such as described in [3][4] are used. This paper introduces a new mobility model evaluation system based on well known distributions(Normal, Log Normal and Rayleigh).The model introduced is decentralized - in sense that it assumes no node owns the information being transmitted.Furthermore impact (power and number of hops required to reach the destination) of selecting secondary sources based on such a model is discussed.

Keywords
Ad Hoc Networks — Topology — Routing Protocols — Mobility Models

[1] *Electrical Engineering, Columbia University*

Contents

Introduction

Since the emergence of Ad-Hoc networks, guarantee of information retrieval and power requirements have been central issues. For routing various efforts[5] have been tried. But as the scale of these devices go smaller protocols need to made more simpler as the available power becomes scarce. This effectively translates to simple and efficient routing protocols.

In this paper a decentralized model for selecting neighbors is discussed. The model is based on Hot Potato Routing [6] which aims to get rid of the packet as soon as possible. The model intuitively makes sense for such a device because it can allow the device to go back to sleep mode as soon as possible [7] rather than waiting for best possible neighbor. So for optimizing power the idea of best route can be dropped to best available route in a particular time interval. Hence the designed model as soon as finds a node in its range,transfers the packet to that neighbor.The model leaves the decision to accept the information on the nodes (see [8]) - if a node chooses not to accept the packet then another node in its range is queried,if no node is available a new round of node-discovery is run to get new nodes who might have wandered in the then sender's realm. Here realm is defined to the maximum range up-to which a node can transfer.

1. Model

The model developed is 2 dimensional. Figure 1 shows the initial setup of the model. The model assumes that at any given time a packet will belong to **only one node** ,the **destination is fixed** and the packet(irrespective of number of iteration) will **reach its destination**. Furthermore only case of extreme mobility is considered. This means that in every sampling interval all the nodes would change their location based on certain distribution. These assumptions do place some constraints on the destinations to which the information can be transferred.

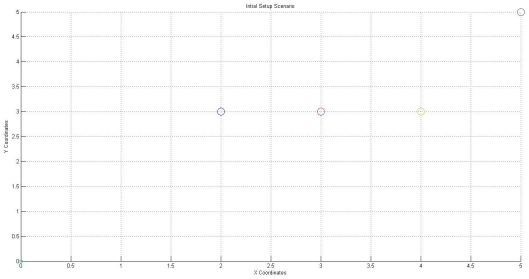

Figure 1. Initial Simulation Model

Setup Phase After establishing the realm of the experiment, the model starts from a source node and then whenever it finds a nodes in its realm(as per 1) it gives the data to that node and then the receiving node becomes the source node for the rest of the experiment. It should be noted that a node changes its role every iteration so it might happen that it has received the same data but they can't be in consecutive intervals.

Constraints Since the model is two dimensional every location can be defined by a tuple (x,y) -where x denotes the X-Axis coordinates and y denotes the Y-Axis respectively. If the range to which a node can transmit is defined as 'R' units. Then for any tuple (x_t,y_t) to be in the realm, the condition

$$\sqrt{(x-x_i)^2+(y-y_i)^2} \leq R \tag{1}$$

has to be satisfied for at-least one (i,j) which can be selected from the distribution. This specifies the bound on the location of the destination node.

Establishing Realm The destination location is first chosen arbitrarily, denoted by $\mathbf{Destination_{i,j}}$ or simply destination,then using equation 1 the boundaries of the realm are established. Then,the location of the source is chosen-its denoted by $\mathbf{Source_{initial}}$. It can be again chosen randomly from the realm or it can be fixed for experiment.

Simulation Parameters For every experiment the range 'R' is fixed. This is done to ensure that the boundary condition in equation 1 is always satisfied. There is no limits on the number of counts a message can take to reach the destination. Now the distribution for selecting the neighbors can be selected.

Simulation Algorithm 1 is a recursive implementation. The algorithm starts with setting the current node to be the source node. If the node is in the range of the destination it sends the message and bails out ,else it makes a procedural call to generate neighbors which returns set number of generated neighbors for that iteration. Then it iterates over the generated neighbor and checks if the node is in the range. If it's in the range it sends the information and sets the in_range flag. Then the selected neighbor becomes the source node and the algorithm continues.

Selecting the first neighbor is the Hot Potato Routing and as has been mentioned before the neighbor can choose to either accept or reject the message using it's own policies which is out of scope of this experiment. If it rejects the message then another round of neighbor selection is run and new neighbors are added to the mini-realm of the current source.

Once a neighbor has been selected,it becomes responsible for the message and has its own mini-realm which is defined as 'M_r' $\subseteq$ of the original realm bounded by range. Again any current node can only transmit to nodes which are in its range 'R'. So the algorithm recursively runs until a selected node finds another node to send the data to. Each iteration can provide a policy look-up for accepting or rejecting the data.

Algorithm 1 Simulation Algorithm

```
Initialize:
    message_not_sent = True
while ( message_not_sent) do
    set current_node=src_node
    if (in_range(currentnode,destination)) then
        message_not_sent = False
        break
    else
        while (in_range) do
            procedure GENERATE_NEIGHBOR
                for i = 0 to number_nodes do
                    if (in_range(current_node,destination)
then
                        current_src = neighbor
                        in_range = True
                    end if
                end for
            end procedure
        end while
    end if
end while
```

Termination Condition The algorithm terminates when a node comes in the range of the destination.The destination might chose to not accept the data,but it's assumed that destination will always accept the data.At this point the algorithm records the last hop,the number of hops and sets the message_not_sent flag to False.

2. Simulation Architecture

The model is only simulation based and has been never tested on real scenarios. The architecture of the system is as shown in Figure 2. It consists of the following components:-

1. Grid Generator
2. Neighbor Selector
3. Plotter

Grid Generator The Grid Generator concerns with the selection of limits of the grid based on realm constraint given by equation 1. It basically sets the boundaries in which the generated distributions have to reside so that the process remains bound.

Usually for simulations [3] Random Walk Models are used,but as 3-D random walk models are not ergodic following from [10],it can be argued that some other distribution albeit non ergodic in 2-D might have equal or better performance than Random Walk Mobility model. Furthermore since the study does not target the human behavior so a good bet can be to simulate on different distributions and evaluate the results.

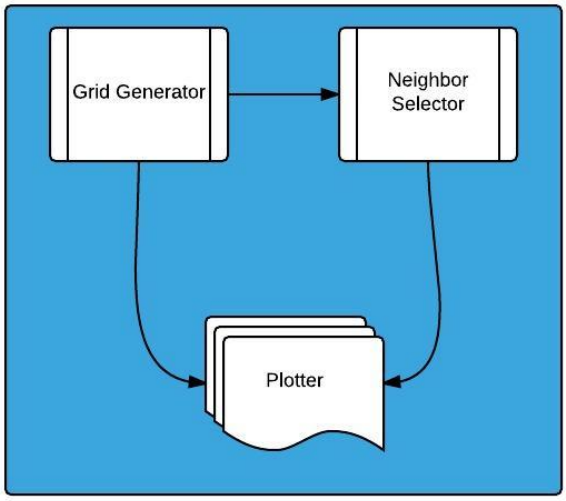

Figure 2. System Architecture

Neighbor Selector Neighbor selector is a procedure call(albeit an important one) inside the system. The algorithm for generating the neighbor is algorithm 2.

Algorithm 2 Neighbor Selection Algorithm

```
Initialize:
    neighbors=[]
    effective_counter = 0
while (effective_counter <num_nodes) do
    Generate (X,Y) from distribution
    if ((X,Y) ≠ cur_src or destination) then
        append to neighbors
    end if
end while
return neighbors
```

This algorithm also takes care of the the distribution from which the values can be generated. The algorithm also does a realm check and makes sure that the node which was selected is not the destination node or the current source node because that would get the algorithm waste one step.

Plotter The plotter is based on [9] which interfaces with the Grid Generator and Neighbor Selector. It only shows the final path and other useful information on a plot.This is just a display tool and just serves as a front end for the selector algorithm. It is included in the discussion for the purpose of exhaustiveness.

3. Results

All random numbers were generated using Python's Random Number generator library[11].

3.1 Random Integer

First the most basic algorithm "random.randint" was first chosen. According to the documentation from the website it generates a random number between two given ranges. This served as a good point to evaluate the model as the realm can be easily specified with the ranges.

Figure 3 shows a result for the model simulation. For the results shown in Table 1, the destination was fixed at (5,5) and range and number of nodes were varied.

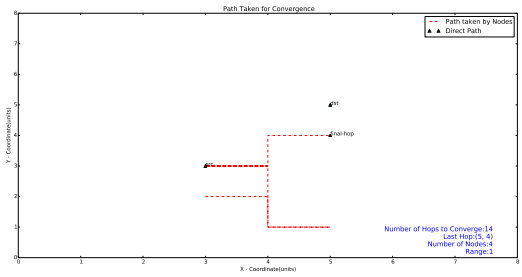

Figure 3. Simulation Result for Random Integer Selection

Table 1. Iterations with $5*5$ Realm

Num_Iter	Last Hop	Range	Num_Nodes
14	(5,4)	1	4
33	(5,4)	2	4
1	(3,3)	3	4
82	(4,5)	1	40
288	(4,4)	2	40

Discussion The results show that if the location is small increasing the number of nodes might not give the performance improvement one would expect otherwise. Also it might increase the time to converge for the algorithm. As is evident

Table 2. Iterations with $10*10$ realm

Num_Iter	Last Hop	Range	Num_Nodes
10	(5,4)	1	4
9	(6,4)	2	4
6	(6,4)	2	40
1	(3,3)	3	40
5	(7,5)	2	400

from Table,increasing the size of the realm does improve the hop_count(Number of iterations) but after some time it flattens out. As can be observed even after adding 400 nodes in the small realm the improvement is only minimal.

Furthermore this model is not actually good for evaluation of real scenarios because it generates only random integers. But for completeness purpose the model is re-run with a larger realm.

3.2 Normal Distribution

The simulation described in above section was based on discrete steps. Now a simulation based on continuous variation of neighbor coordinate selection is described.

Again the simulation begins with fixing the position of the destination node at (5,5) and the initial source node is chosen at (3,3). Simulation is rerun for different ranges and number of nodes and the results are described.

Figure 4 shows the one of the simulation iterations. Since the range of the transmission was very small it took a really long time to converge. But simulating with the same setup with increased range of 2 the algorithm converges in 32 iterations. Figure 5 shows the simulation result for this scenario.

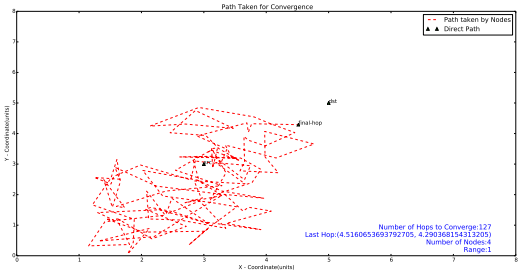

Figure 4. Normal Selection of Nodes

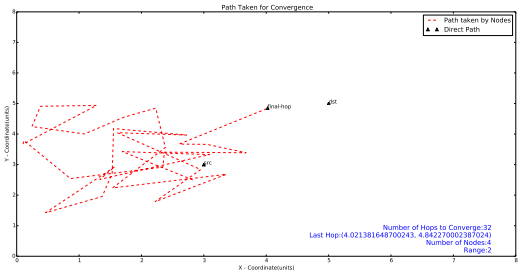

Figure 5. Normal Selection of Nodes

Table 3. Normal Iterations with $5*5$ realm

Num_Iter	Last Hop	Range	Num_Nodes
127	(4.51,4.29)	1	4
32	(4.02,4.8)	2	4
45	(4.32,4.86)	1	40
6	(4.32,4.86)	2	40

Table 3 and Figure 6 show the results for the simulation scenario. The results show a dramatic increase in decreasing the number of iterations(hop_count) as can be observed from the Table 3. In smaller realms if the nodes have a better node then they are better off with using a longer range for sending the information,if the neighbors are selected from Normal Distribution.

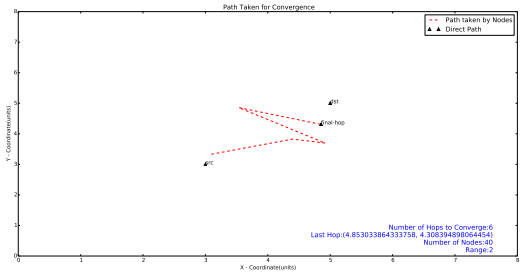

Figure 6. Normal Simulation $5*5$ realm range 2, Node 40

3.3 Log Normal Distribution

Further extending the discussion ,Log Normal Distribution was considered. This distribution has a property that log of this distribution gives back Normal distribution with same μ and σ as of the Log Normal Distribution. For the simulation $\mathcal{N}(0,1)$ was chosen and it was scaled down by factors to bring in the realm of the simulation. Since the generated numbers were more than 1 so a upper threshold of 0.499 was put to ceil the numbers. Each number was multiplied by 10 to bring it to the scaled format.[1]. Again the same set of simulations were repeated and the results were recorded.

Figure 7,8,9,10 show the results of the simulation on a $5*5$ realm. The results shows an effective decrease in number of hops as the number of nodes increase. Table 4 shows the results of the simulation in the tabulated form.

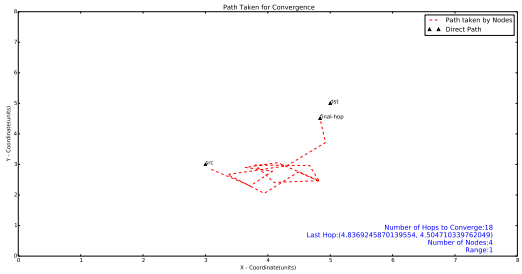

Figure 7. Log Normal Simulation $5*5$ realm range 1, Node 4

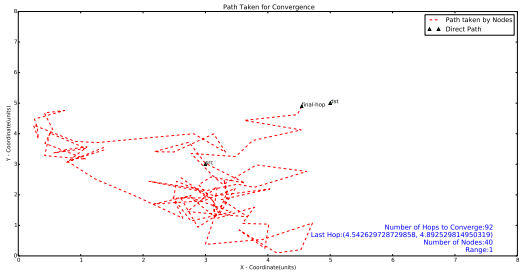

Figure 8. Log Normal Simulation $5*5$ realm range 1, Node 40

[1] I think I made a mistake by multiplying it directly by 10 as it would never give me values like 0.1,0.2...

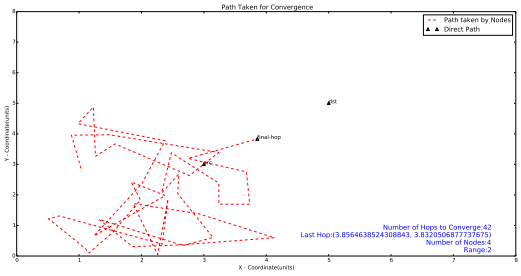

Figure 9. Log Normal Simulation $5*5$ realm range 2, Node 4

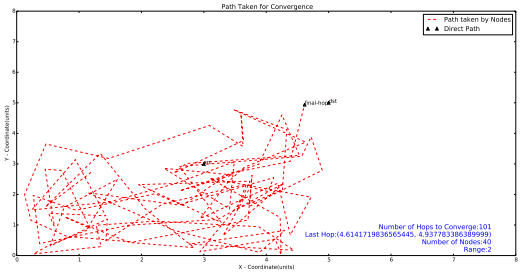

Figure 10. Log Normal Simulation $5*5$ realm range 2, Node 40

Table 4. Log Normal Iterations with $5*5$ realm

Num_Iter	Last Hop	Range	Num_Nodes
18	(4.83,4.50)	1	4
92	(4.54,4.89)	1	40
42	(3.85,3.83)	2	4
101	(4.61,4.93)	2	40

3.4 Rayleigh Distribution

Rayleigh distribution is defined as

$$P(x;scale) = \frac{x}{scale^2} * e^{(\frac{-x^2}{2.scale^2})} \quad (2)$$

When two Gaussian distributions are orthogonal,the result of combining them is Rayleigh. Here scale is ≥ 0. So for the simulation $\mathscr{R}$ (3,10000) for the distribution was chosen. Again as has been done in previous sections the simulation was run keeping the destination constant at (5,5) and choosing the neighbor coordinates from a Rayleigh distribution.

Table 5. Rayleigh Iterations with $5*5$ realm

Num_Iter	Last Hop	Range	Num_Nodes
136	(4.77,4.86)	1	4
8	(4.54,4.89)	1	40
4	(3.85,3.83)	2	4
2	(4.61,4.93)	2	40

It can be seen that from Table 5,that Rayleigh Distribution does offer very high hop count reduction as the range is doubled.

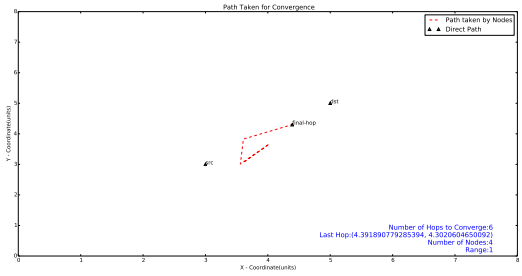

Figure 11. Rayleigh Distribution Simulation $5*5$ single result

4. Conclusion

Many more distributions can be tested using the developed model.The algorithm is mathematically correct so the model is guaranteed to terminate each time the simulation is run(though there is no bound on the maximum number of iterations). The model serves as a bench mark for selecting neighbors based on certain patterns.

Random Integer Selection : Grid following bots follow a single rail or a fixed trail which can be expressed in form of discrete integers. Also in cases of emergencies,the bots might move across the rails in a haphazard manner. Therefor the random integer models can capture these kinds of movements when the device is moving in haphazard way but can only chose fixed number of destinations as it is bounded.Random Integer simulation shows that increasing the number of nodes has very less/minimal hop count improvement. If the devices are follow a Random Integer pattern, the best bet is to transmit with high range and the same number of nodes. This effectively translates to the conclusion that increasing the number of nodes in a region (following Random Integer) provides no gain,it might increase the number of counts for the destination. Also a rule of thumb can be to keep the quantity

$$\frac{m*n}{N^2} \leq 1 \quad (3)$$

where m*n is the dimension of the realm and N is the number of nodes. This is an empirical formula based on the simulations.

Normal Distribution : Since Normal distributions are continuous,there are more chances of reaching the target as there are more possibility of selecting the values. As expected it does follow expected trend that increasing the number of nodes would give faster convergence. This also follows to distributing the nodes in normal distribution fashion would give linear reduction in hop count as the number/range of nodes are increased. For this no empirical formula can be derived.

Log Normal Distribution : Log Normal distribution behaves erratically in the sense that increasing the number of nodes may/may not always lead to less hop_count. Comparing

the distribution with the Random Integer,it oscillates really fast with respect to the number of nodes or range of nodes. Also log normal distribution cannot be generally attributed to any normal mobility. So if there is a choice between Log Normal and other distributions in a highly mobile environment it should never be chosen.

Rayleigh Distribution : Since Normal Distribution scales linearly with increasing range and number of nodes,Rayleigh Distribution as it's an orthogonal combination of two Gaussian Distribution the pattern should have been linear. This is evident from the results,as the number of nodes increase the algorithm converges to the destination. For the same set of scenario (w.r.t Normal Distribution), Rayleigh Distribution converges to the destination really fast.

By considering four distributions it can be seen that if the neighbors are according to Rayleigh Distribution then any On Demand Routing Protocol would be able to converge the fastest. The model is only valid for extreme mobility environment,i.e majority of nodes change their position after every sampling/sending time. This case is extreme and the paper presents a way to tackle the models.

Future Work

The work can be extended for various other distributions(like Pareto,Poisson) and their performance can be evaluated using the same setup. Also additions can be done to limit the extreme mobility for the nodes. Furthermore a limit on the number of maximum hops can be incorporated for early algorithm termination. The work has never been tested on real scenarios where a node can choose to reject the messages. This is a a very generous assumptions as a node based on its own power and queue length can chose not to accept the message.

Limitations and Shortcomings

The biggest limitation of the model is that its extremely mobile and there is no way to control the mobility. Also the work is untested on real devices so the results should be taken with a grain of salt. The algorithm assumes that the destination would always accept the data but it might not be the case as the source/sink can have no longer use for the data and in that case the algorithm would stall. A quick-fix would be to drop the packet in that case but that has not been incorporated in the algorithm. The algorithm is available at [12].

Appendix

Recursive Algorithm for Computing Neighbors: This implementation assumes that all the nodes change their position after each iteration.

```
def run(self):
        """
        Recursively computes the distance and
    staggers them.
        """
        msg_not_sent = True
        while msg_not_sent:
            if self.distance(self.cursrc, self.dst
    ):
                self.counter += 1
                print "Message Successfully
    Delivered in {},last hop {}".\
                      format(self.counter, self.
    cursrc)
                msg_not_sent = False
            else:
                in_range = False
                while not in_range:
                    neighbours = self.generate(
    self.cursrc)
                    for neighbour in neighbours:
                        if self.distance(neighbour
    , self.cursrc):
                            self.cursrc =
    neighbour
                            self.path_x.append(
    self.cursrc[0])
                            self.path_y.append(
    self.cursrc[1])
                            self.counter += 1
                            in_range = True
```

Generating the Neighbors using different distributions: Generates sets of neighbor(bound by maximum nodes),while removing the source and destination nodes.

```
def generate(self, exclusion_tuple):
        """
        param: exclusion_tuple Tuple that has to
    be removed neighbours:.
        return list of neighbours
        """
        neighbours = []
        i = 0
        while i < self.num_nodes:
            x = random.uniform(self.x[0], self.x
    [1])
            y = random.uniform(self.y[0], self.y
    [1])

            if (x, y) != self.dst and (x, y) !=
    exclusion_tuple:
                neighbours.append((x, y))
                i += 1
        return neighbours
```

References

[1] Perkins, C.E.;Royer, E.M.;Das, S.R.;Marina, M.K., Performance comparison of two on-demand routing protocols for ad hoc networks

[2] Elizabeth M. Royer,Chai-Keong Toh A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks

[3] Tracy Camp, Jeff Boleng and Vanessa Davies A survey of mobility models for ad hoc network research.

[4] Understanding individual human mobility patterns
Marta C. Gonzalez,Cesar A. Hidalgo,Albert-Laszlo Barabasi

[5] Charles E. Perkins,Pravin Bhagwat
Highly Dynamic Destination-Sequenced Distance-Vector Routing(DSDV) for Mobile Computers

[6] Costas Busch, Maurice Herlihy, and Roger Wattenhofer
Randomized Greedy Hot-Potato Routing

[7] Yu-Chee Tsenga, Chih-Shun Hsub, Ten-Yueng Hsieha
Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks

[8] I.F. Akyildiz, W. Su,Y. Sankarasubramaniam ,E. Cayirci
Wireless sensor networks: a survey

[9] Hunter, J. D.
Matplotlib: A 2D graphics environment

[10] Lyons, Terry
A Simple Criterion for Transience of a Reversible Markov Chain

[11] Python Random Number Generator Library
https://docs.python.org/2/library/random.html

[12] GitHub Code base
https://github.com/shivamchoudhary/eureka