Propagation Delay Models in Bio-Inspired Nanonetworks

Chukwudi James Ojukwu

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
June 2013
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yilmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Muhammed Salamah
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Doğu Arifler
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Doğu Arifler
2. Assoc. Prof. Dr. Muhammed Salamah
3. Asst. Prof. Dr. Gürcü Öz

ABSTRACT

Nanomachines are devices that are made up of nanoscale components. By themselves, nanomachines can perform only simple tasks. To achieve more complex tasks, networks of manomachines or nanonetworks are formed. Molecular communication is a biocompatible, bio-inspired alternative to traditional electromagnetic communication in nanonetworks. In molecular communication, molecules can be considered as information packets. Free diffusion based molecular communication requires no external energy and is the most basic information transport mechanism being considered for nanonetworks. This form of communication however is slow due to the random walk of the particles and the information packets can also be delivered out of order to the destination. These issues present challenges to design and implementation of molecular communication based nanonetworking protocols. While there are significant studies that address physical layer aspects of molecular communication, there is relatively less work in the link layer. In particular, modeling of channel delays or sojourn times of molecule-packets that arrive at a nanomachine is required for queueing theoretic analyses. To this end, simulations are performed to measure the propagation times of molecules between a given source and a destination in both bounded one- and two-dimensional spaces and unbounded one-dimensional spaces. Here, one-dimensional settings correspond to molecular communication that take place in very thin capillaries and two-dimensional settings correspond to communication in junctions with small widths, negligible heights or on membranes. There are no closed-form formulas for the delay distribution of freely diffusing particles in arbitrary, bounded environments. The delay measurements in bounded settings are fitted to well-known distributions that are commonly used in modeling time to complete a task. The fits can be used to generate arrival times of molecule-packets at a node. This study is expected to contribute to the analysis of link layer protocols and workload models being considered for nano communication networks.

Keywords: Distribution Fitting, Free Diffusion, Molecular Communication, Nanonetworks.

ÖZ

Nanomakineler nano ölçekte bileşenlerden oluşan cihazlardır. Nanomakineler kendi başlarına sadece basit işlemler yapabilirler. Daha karmaşık işlemler için nanomakinelerden ağlar, yani nano ağlar, oluşturulabilir. Moleküler iletişim biyolojiden ilham alınmış, biyo-uygun, geleneksel elektromagnetik iletişime alternatif bir iletişim şeklidir. Moleküler iletişimde paketler moleküllerdir. Serbest difüzyona dayalı moleküler iletişimde, harici enerji gereksinimi yoktur ve nano ağlar için düşünülen en temel veri taşıma mekanizmasıdır. Ancak bu mekanizma, parçacıkların rasgele yürüyüşünden dolayı yavaştır. Ayrıca, parçacıklar gönderilme sırasından farklı olarak hedefe ulaşabilirler. Bunlar, moleküler iletişim protokollerini tasarlamayı zorlaştırmaktadır. Moleküler iletişimin fiziksel katmanıyla ilgili birçok çalışma olmasına rağmen, bağlantı katmanıyla ilgili çalışmalar çok azdır. Özellikle, iletişim kanalında paketin yayılma zamanı kuyruklama teorisi açısından önemlidir. Bu bağlamda, yayılım zamanlarını ölçmek için bir ve iki boyutlu, sınırsız ve sınırlı ortamlarda difüzyon simulasyonları yapılmıştır. Bir boyutlu simulasyonlar kılcal damarlardaki iletişime karşılık gelebilir. İki boyutlu simulasyonlar ise kavşak ve zar üzerindeki iletişime karşılıktır. Sınırlı ortamlarda, serbest difüzyonla hareket eden parçacıkların gecikme zaman dağılımlarının formülü bulunmamaktadır. Sınırlı ortamlardaki yayılım zamanları, bilinen dağılımlara eşleştirilmiştir. Eşleştirmeler, molekül-paketlerin bir nanomakineye varma zamanlarını modellemek için kullanılabilecektir. Dolayısıyla, bu çalışma nano ağların bağlantı katmanı analizlerine katkı koyacak niteliktedir.

Anahtar Kelimeler: Dağılımlara Eşleştirme, Serbest Difüzyon, Moleküler İletişim, Nano Ağlar.

To the Holy Trinity One God; to my Blessed Mother Mary; to all the angels and saints; to my Family; to my Fiancee, to Safuriyau Ahmed, to Mehran Hosseinzadeh, to Assoc. Prof. Dr. Doğu Arifler.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The smaller the electronics are the less intrusive they are and generally the better they are for all involved. The drive for smaller and better electronics brings about the drive towards the nanomachine. A nanomachine refers to a well-arranged single unit mechanical device at nanoscale that is designed from materials and components at nanoscale to serve limited purposes [1] (in most cases just a single purpose). To appreciate the complexity of designing and achieving a machine of such a specification, it helps to take into cognizance that an object with a specification to the tune of the micro-scale cannot be seen by the naked human eye. An object with specifications measuring at nanoscale is smaller than that at the micro scale by a factor of 1000. Naturally, the quest to achieving this feat has been met with a lot of obstacles, but the benefit that is envisioned from it been implemented is the incentive that has made it possible for those challenges. One such challenge is that of energy consumption in communication among nanomachines. The most tempting means of communication in the nano arena is the use of free diffusion in which particles are let loose and they migrate randomly walking to the destination by the natural phenomenon called Brownian motion. Diffusion has major downsides: it has a very low range and high delay associated with transporting particles from one point to another. The purpose of this thesis is to analyze and characterize delay in a free diffusion based molecular channel so that engineers can plan and design nano communication networks at a level above the physical layer. It should be noted that there are only a few studies that consider the link layer [2] and above of a nano communication network.

1.1 Nanomachines in General

The general definition of a nanomachine is what is given above. Due to the size of a nanomachine, what it is able to achieve is not so much as to be felt useful in the real sense, for mostly they carry out just a single task, and this task carried out is done at a scale that would make little or no impact in the environment. The only way to make this impact felt is if a group of these machines worked together towards a given goal either by each taking on the same task or by sharing different parts of the process to reach that goal. In order that this should happen in a way so as not to negate themselves, they must communicate with each other whilst they work. This is how the concept of nano communication comes to be. Nano communication is any and every infrastructure that enables nano machines to communicate with each other.

Initially, when machines at the level of the nano were contemplated, the initial direction was the application of the traditional communication techniques at the nanoscale. The approaches to these methods were in the categories bottom-up and top-down [1]. The top-down dealt with scaling down the existing standalone units and their communication capabilities, such as transceivers, down to the nanoscale. This was found impossible with the current technological advances on ground as talked about in [1]. Also, the bottom up approach is very similar to the top down in that the traditional communication apparatus were to be applied at the nanoscale, only that in this case the parts making up this nanomachine would be manufactured separately and then assembled automatically using self assembly [3]. However, again this is theoretical only for the technology needed to make these components are not yet in existence. Yet, there are other objections uniformly common to both these presented methods given other than their infeasibility. The objections to them are due to the principle limitations, power consumption, and bio-incompatibility. Principle limitations make communication between devices at the nano level different due to quantum effects [4].

Power consumption is an important factor in this network setting because repowering it would be hard after deployment. Due to the power consumption rate during transmissions, no matter what power saving scheme is employed, the battery would eventually run out. Also, due to the diverse environments that such small devices could be deployed in and the limited options in the traditional computing world of recharging of spent power, the nanomachine would not last for so long. Also, at such a dimension, only simple tasks should be assigned to each nanomachine and the addition of power saving schemes would greatly add to its complexity. If, in addition, there are acknowledgments attached to each packet sent in the traditional sense, this also depletes the energy of the nanomachine drastically as transmitting and receiving are known to be the most power intensive part of any activities of a communicating device.

These traditional-styled nanomachines are made from components not easily or readily assimilated into the natural environment. The chief need of such a small scale technology is deployment in places unreachable before. If deployed in the human body for example, it may cause some damage when its active life is over. Also, even when not deployed in living organisms, they could still constitute health problems as they can easily be carried by the wind and deposited in places not intended for. Since they cannot be detected visually, they will not be known to be there and may prove hazardous to the environment if they cannot be assimilated into the ecosystem. The hazardous nature of these devices will result in greater ecological problems over time. For these reasons, greater strides have been made towards making biological nanomachines a reality those of the traditional sort. In fact according to [5], when a reference is made to nanomachines, more often than not, what is meant is biological nanomachines.

1.2 Biological Nanomachines

A third and by far the most promising approach to making nanotechnology a reality is the bio-hybrid approach. Biological nanomachines already exist in abundance in nature. These are cells with facilities synonymous to a miniature computing body. Of course in nature, these cells are designed to see to its survival, and the survival of similar cells around. As a result, they are not in a form to be readily manipulated for other purposes. However, with a little modification, they could do what you would want them to do with regards logic, sensing and/or actuating. Reference [6] speaks of molecular motors existing in nature, and components that could serve as building blocks for the formation of nanomachines such as biochemical molecules, complexes and circuits that can pass for processing units [7]. The construction of nanomachines from its base components is not the only way to create these nanomachines. Genetically engineered cells cited in [8] are more apt for manipulation for diverse purposes.

As in nature, these cells taken individually cannot do much, but taken as a whole, they get a lot done: the way they accomplish this is by working together. The way they are to work together is by molecular communication.

1.2.1 Molecular Communication

Molecular communication is used here over the term nano communication, because this term really does accentuate the departure of the communication technique encountered in the biological sphere of nano communication from the traditional way that network devices communicate. Earlier on, in this thesis, it was highlighted that due to quantum effects principles guiding well established ways of communicating, such as electromagnetic waves, fail. Mimicking the way cells communicate in nature, carrier molecules (information molecules) are employed as a way of transmitting information [9]. In general, the sender encodes information into these molecules which can either be produced by the sender, or freely available in the environment or attracted to the sender (as in the case of carrier bacterium [10]). Then, these are either sent by passive means (e.g. diffusion) or active means (e.g. directed molecular motor movement by chemical consumption).

1.2.1.1 Traditional Communication Methods vs. Molecular Communication

The method of communication by the molecular means is radically different from the way it is known in the traditional sense. As stated above, molecules are used in the latter as packets. What is truly unique is the way the packets are transmitted. In the traditional means, the power cost of the transmission is borne by the sender. However, in the case of these biological nanomachines, the propagation environment bears the burden of the power cost [9]. The propagation environment is an aqueous solution, but due to the noise inherent in this environment encountered by the sojourning information molecule, whether in the passive form (e.g. diffusion where the erratic movement the particle makes redundancy a necessity in this communication type) or in the active form (e.g. molecular motors consuming chemical energy in order to overcome other molecules and counter energies in the environment), the range of this transmission falls within the nano-micro scale [11]. Same obstacles render the speed of the packets in the nm/s category. On the other hand, the conventional mode of communication boasts of ranges of communication in meters to kilometers and speeds of signals matching the speed of light 3 x 10^8 km/s. Also, due to the stochastic nature of the information molecules, the probability of loss is very high, hence is unreliable relative to the conventional ones. Redundancy is encouraged in molecular communication to make sure that message is delivered because in this model, acknowledgments are not used given the number of information and energy considerations.

1.3 Field of Deployment

Nanonetworks are very attractive for deployment in so many areas. Reference [12] has a list of areas where this technology, even in the light of its limitations, would make a huge impact. These include the biomedical, industrial and consumer goods, and the military amongst others. The most promising of these is the medical profession, specifically the internal body medicine. Why this is so is that the disadvantages of the nanonetworks such as the its range and speed of information particle is downplayed by the fact that the shortest distance between the nanomachines can be achieved in the human body so easily as the dimensions of the human body is not that vast. Two case scenarios will serve to drive home this point. Also the size, self-sustaining attribute, self-replicating, and the biocompatibility of these nanomachines make them much coveted as compared to the closest competing in-body silicon-based machines which need to replaced when spoilt or not in need anymore, or they have to be brought out for replacement of batteries. With the nanomachines proposed, all of these problems will be in the past as no more needed machines could simply be assimilated by the body, and older machines could replicate themselves before self-destructing, and since the machines draw little power they need from the environment (e.g. glucose), they never need a battery change.

1.3.1 In-Body Drug Delivery

This medical application cited in [13] is the use of these nanomachines to administer drugs at certain times when needed. This involves a trigger cell (the drug repository, sender) and the target cell (the receiver) [9]. The trigger cell normally has a timer telling it when to send the needed drug. The doctor has a time frame when this drug needs to be delivered, e.g. at noon period, so since the said cells are close together and the time frame is long enough, the needed drug will always be delivered in good time. Therefore, this technology could enable a person who needs to constantly take life saving drugs at constant intervals, such as a diabetic, live a normal life by having a repository of this drug in his body administered in the right interval of time.

1.3.2 In-Body Health Monitoring

This is the application of nanomachines in the long time, day to day monitoring of the health of patients. Just like in the case of the in-body drug delivery, the nanomachines are planted in strategic parts of the body. From these parts, they monitor certain cells and organs taking the note of the pH level, cell intake, etc. They are able to do this because they are able to translate information in DNA which has been found in [14] to hold up to 9.2 Mbits of information in just 2 micrometer square of chromosomes. This information will then at regular intervals be sent to a central nanomachine which will store them and on request, will internetwork using the optical naturally occurring options of either fluorescent proteins or Molecular Organic Light Emitting Diodes (MOLED'S) [12] to get the stored up information to the outside world for analysis, possibly by a doctor or a personal health monitor/analysis device.

1.4 Outline of the Thesis

In Chapter 1, as already observed, the general overview of nanomachines in the bio-hybrid category is described. In Chapter 2, notable strides in this class of nanomachines accomplished in research and implementation are looked into. In Chapter 3, the methodology and approach of this thesis with regards to the study of the propagation delay behavior of particles transmitted by the passive means of diffusion in both one- and two- dimensional bounded and unbounded spaces are presented. In Chapter 4, the results obtained from the simulation that are the products of the methodology presented in Chapter 3 are shown and analyzed. Finally, Chapter 5 concludes the thesis.

Chapter 2

NOTABLE DEVELOPMENTS IN NANO COMMUNICATIONS

The setup of a nanonetwork is characterized by nanomachines, information molecules, and the environment which engulfs them all. The nanomachines are further divided into two classes, namely the sender and the receiver. The sender is same as the receiver except it lacks a discriminatory receptacle, but it has the added ability to encode information onto biological material (e.g. DNA translation). Also in some cases, it has the ability to synthesize information molecules from the environment. The receptacle in the receiver is meant to help it attract/capture an information molecule when the latter reaches the former. This setup is not exactly a new thing, in fact as [15][16] puts it, this is found abundantly in nature. What is new is this setup being harnessed as a network for serving purposes not designated to cells (naturally occurring nanomachines) by nature. To achieve certain aspects, an engineered molecular communication has to be developed, modified or even assembled from existing parts in its naturally occurring version. The generic architecture as illustrated in [9] shows that molecular communication contrived consists of the following states: encoding, sending, propagating, receiving and decoding. The following section will treat the developmental efforts under the headings below:

- Nanomachines

- Propagation and engulfing environments.

2. 1 Nanomachines

Nanomachines are derived basically from cells in nature in a variety of ways, either by tinkering with already existing cells by synthesis (i.e. by creating a new variation of the existing cell with added functionality through genetic engineering) [9] or by putting together a cell-like entity with components existing in nature.

The aspect of adding desirable communication attributes to existing cells by genetic engineering is illustrated in [17]. Not only that, a step further was achieved when certain sender nanomachines were designed to synthesize information molecules [18]. In the same vein, the receiver nanomachines were designed to not only receive information molecules but to receive specific ones. As such, by the differentiating amongst the different kinds of information molecules, the sender could now make sure that only the intended machines react to the sent messages. To make addressing more generic, however, work is being done on using DNA sequences to accomplish the addressing issue [16]. In this way, the work of coming up with as many variations of the information molecule types as there are receivers and also the prospect of getting a single nanomachine to synthesize all the various types information molecules can be avoided. Also, intermediary nanomachines could be employed to act as repeaters. The basic functioning of each nanomachine’s circuitry such as logic functions (biochemical inverter [19], and AND or OR gates [20], etc.), toggle switches [21], and oscillators [22] can be added through genetic engineering. Also, producing a nanomachine from base elements, making the finished product look like cells existing in nature is another product of research in this field [9]. The aim of this method is simplification so that only what is needed is included and nothing else. A lipid bilayer is used to mimic the permeable membrane of a cell [5] into which functional natural components are added such as receptors (proteins). Even though this is an artificial cell, it can achieve replication using chemical components as proposed in [23]. As noted in the previous section, in a 2 micrometer square of chromosome of a bacteria, 9.2 Mbits of information can be housed as compared to the projected achievable storage capacity for 2014 for conventional storage devices for the same area, 490 bits [9]. The possibilities are limitless with regards to transmission except limited by the receivers' capacity. Reference [24] found that the amount of information a receiver nanomachine can decode (or react to) must be proportional to the number of its configurations. Also, work has been done extensively as to how nanomachines operate in networks where the information molecules are bacteria. With regards to the attractants the following questions were investigated: how the sender attracts these empty bacteria using attractants [25], how to encode the plasmids to be inserted into a bacterium with information [26], how these loaded bacteria are attracted to the receiver nanomachines by yet another set of attractants, how they are then attached to receiver by a pilus [10], and how by DNA synthesis the information containing plasmid is recovered by the receiver.

2.2 Propagation and Environments

Propagation is that period in the communication process involving nanomachines in which the information molecule moves through the environment engulfing both nanomachines from the sender to the receiver. Research has unveiled two propagation types:

- Passive

- Active

In the former, the basic form of communication is diffusion, and in the latter, the information molecules are attached to other molecules which make marked effort against the forces in the environment (energy and non-communication molecules) to get to destination. The distinction itself is as a result of the independent work of various researchers.

2.2.1 Passive Propagation

There are various forms of this class of propagation elucidated by research efforts. The first kind is free diffusion based molecular communication in which the molecules are released by the sender by opening of a gate [9] and the molecules are scattered in all direction due to interaction with other molecules when released (broadcast style) and due to its inherent physical tendency to get away from molecules of its kind, it exhibits a hyper willingness to mingle with other kinds of molecules; that is to say, molecules move from a region of higher concentration to a region of lower concentration. In this all, surrounding nanomachines are engulfed in the ensuing stream of information molecules. However, only recipients with receptacles sensitive to the information molecules react to them (decode them) [27][28]. This mode of communication embodies perfectly all the well known attributes of nano communication (i.e. low range, lethargically slow, and unreliable but also energy efficient.)

Another class of this diffusion based communication is the gap junction mediated reaction. Here, cells are placed close to each other and the area from which the diffusion is to take place is selected so as to be directed to the next cell. This selected area of diffusion is called the gap junction channel [6]. Since the cells are adjacent to each other and the channels connect them, the propagation is simply instantaneous. Imagine now a series of these cells arranged in a row connected by gap junction channels where the intermediary cells react to information molecules diffused into it by immediately diffusing some of its own to a cell next to it. The information molecule loss will be low, and due to the number of cells in question the distance achievable increases dramatically and the speed observed will be on the order of 100 m/s [9]. This feat shown in [29] is remarkable when compared to the free diffusion, and for cases were each cell has two or more alternative paths, permeability and selectivity properties of the gap junctions have been used to put in place filtering and switching mechanisms [30]. This mode adds a lot of functionality to the diffusion based communication with one downside: this is much more structured than the free version.

2.2.2 Active Propagation

In this case, a random walk is not employed but rather molecules perform directed movement. To accomplish this directed motion, some sort of external energy must be applied in order to overcome the forces in the surrounding environment. Two major approaches have been brought to light through the efforts of researchers.

2.2.2.1 Molecular Motor-Based

The first of these involves using helper molecules to accomplish this directed motion. These helper molecules according to [9] fall into the category of molecular motors, interface molecules, and guide molecules [31][32]. The guide molecules that are engineered are self-organizing molecules which act as the path on which molecules harboring the information molecules thread to the destination; i.e. they act as a path for the molecular motors to thread. The molecular motors, by using up chemical energy, thread the guide molecules as a train's wheels would ride on rails, overcoming opposing energies and molecules with an energy efficiency of up to 90% [9]. Interface molecules are containers into which the sender nanomachines put in information molecules so as to be mounted on the molecular motor and also to prevent the information molecule from reacting with the encountered molecules in the propagation environment before it gets to the destination [33][16]. This is remarked to achieve distances to on the order of meters. The terrain here must be structured.

2.2.2.2 Bacterial Motor-Based

In this molecular communication mode, there are no set up paths but there are bacteria which act as information carriers. Bacteria propel themselves by using their flagella (motor). They are attracted to both the sender and the receiver attractants [10].

2.3 Intra Networking

Attempts have been made also to link nanonetworks to the other network types. One such attempt is the light transduction where short range molecular information is converted to optical signals and vice versa [12]. The method proposed is to utilize fluorescent proteins [34] and Molecular Organic Light Emitting Diode (MOLED'S) to make this conversion possible [35].

Chapter 3

METHODOLOGY

This section of the thesis is focused on the methods and algorithms used to generate data which will be analyzed to construct propagation delay models. Particles will be assumed to freely diffuse in both bounded and unbounded one- and two-dimensional environments. Examples of cases for which one-dimensional (1D) analysis are valid are scenarios where particles are transported in capillaries with negligible width. Transport on a membrane, a dish, or a junction are examples for which a two-dimensional (2D) analysis is valid. Three-dimensional analysis is proposed as future work. Note that the distances to be considered will be 1, 2, 4, and 8 micrometers. This means this investigation here will be based short-range communications.

3.1 General Analytic Considerations

The free diffusion talked about in Chapter 2 is very slow in packet propagation. The particle can wander in the environment for a very long time. Hence, a time to live (TTL) must be assigned to each particle so that the algorithms do not run forever by eliminating long-wandering particles from consideration after their TTL expire. Note that such assumption is realistic because generally particles decompose in the environment after a given time. Based on observations, a 10-second TTL is assigned to particles. Diffusion coefficient will be taken as D = 10^{-9} m^2/s which is the value used for small molecules in water.

In the one-dimensional bounded case, the source is placed at the beginning of the capillary ensuring that no particles can diffuse behind the enclosed barrier against which the source is located. Also for the two-dimensional bounded case, where the planar junction can have a small width (but zero height), extra boundaries are set up in that no particles can go much further than behind were source is located or forward past where the receiver is located or breach the walled width of the junction. When diffusing particles encounter these barriers, they experience a perfect reflection; that is, there is no loss or gain in kinetic energy and its direction is reversed by a reflection angle and hence its final position is a reflection of where it would have been had there been no barrier in its path. This is not always true in the real world, as there are some losses in kinetic energy, but this approach will be adopted for its ease of analysis.

3.2 One-Dimensional Setup Analysis

There are two cases to simulate:

1. Unbounded
2. Bounded

A pictorial view of what the aforementioned molecular channel is like is given in Figure 1 and Figure 2, respectively.

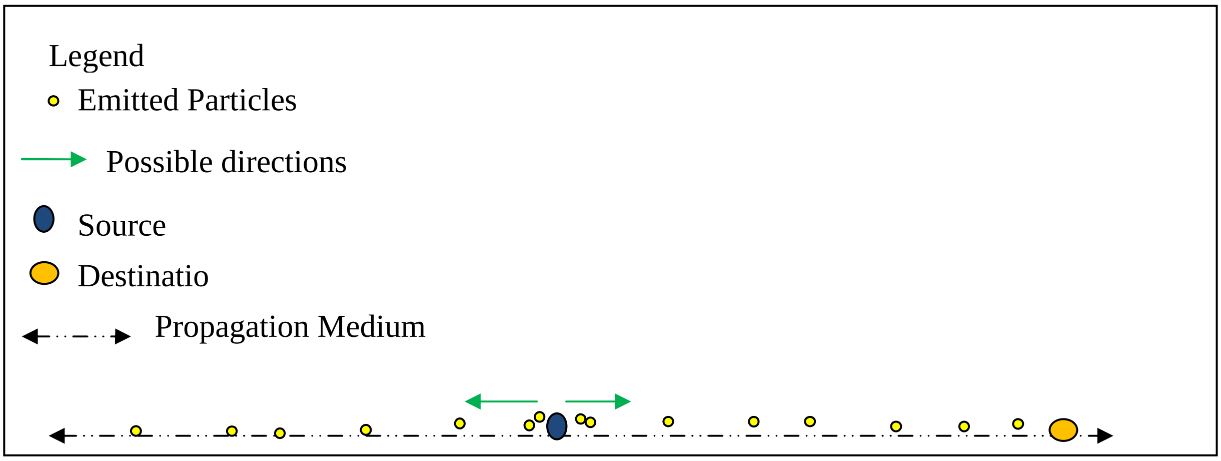


Figure 1: One-Dimensional Molecular Channel (Unbounded Case)

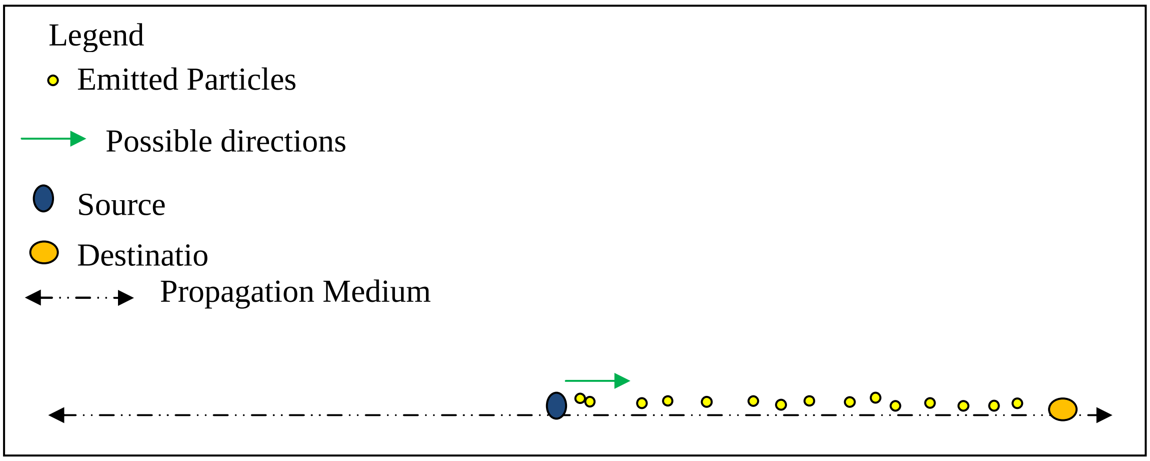


Figure 2: One-Dimensional Molecular Channel (Bounded Case)

It can be observed from the figure that in the unbounded case, the molecules in 1D are unrestricted in both directions. This increases the possibility that some particles will never tend towards the intended destination. The particle's step Δd in the *x*-direction is going to be dictated by the following equation:

$$\Delta \mathrm{d} = \sqrt[2]{2D\Delta \mathrm{t}} * rand1Dim \qquad (1)$$

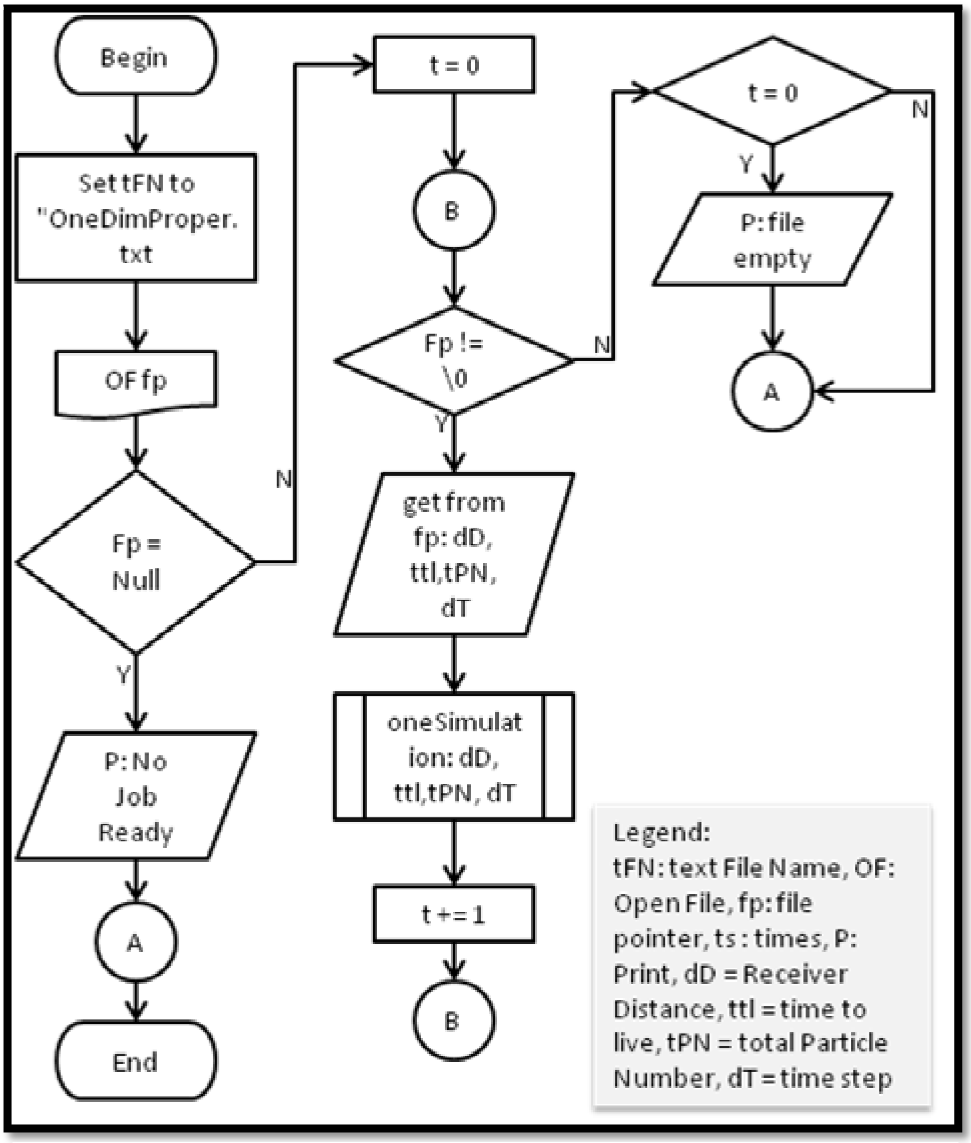
where *rand1Dim* = ± 1 equally likely. Here, Δt is taken as 1 μs.

The bounded case is exactly the same as the unbounded only that in this case, there is an impenetrable boundary at the source. The boundary condition is implemented by a perfect reflection that negates the position of the particle in question by the exact amount by which it would have breached the boundary.

3.2.1 Flow Charts of Subroutines Implementing the Required Scenarios (1D)

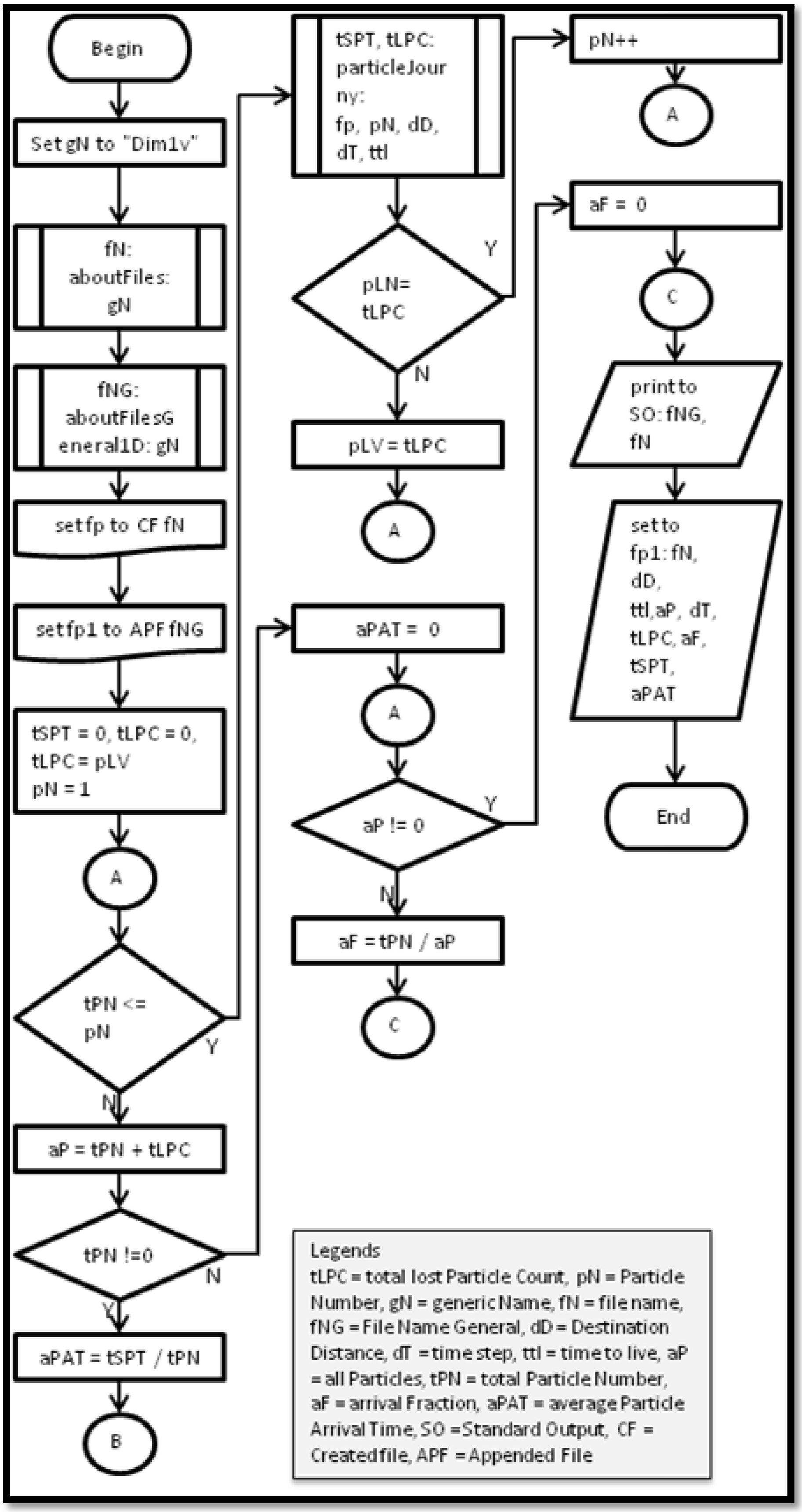
There are several flow charts that describe the algorithm that is implemented in this thesis and they are linked together chiefly by subroutines. Here is the 1D implementation in terms of flow charts.

3.2.1.1 One-Dimensional Driver


Begin
Set tFN to "OneDimProper.txt
OF fp
Fp = Null
N
Y
P: No Job Ready
A
End
t = 0
B
Fp != \0
N
Y
get from fp: dD, ttl,tPN, dT
oneSimulation: dD, ttl,tPN, dT
t += 1
B
t = 0
N
Y
P: file empty
A
Legend:
tFN: text File Name, OF: Open File, fp: file pointer, ts : times, P: Print, dD = Receiver Distance, ttl = time to live, tPN = total Particle Number, dT = time step

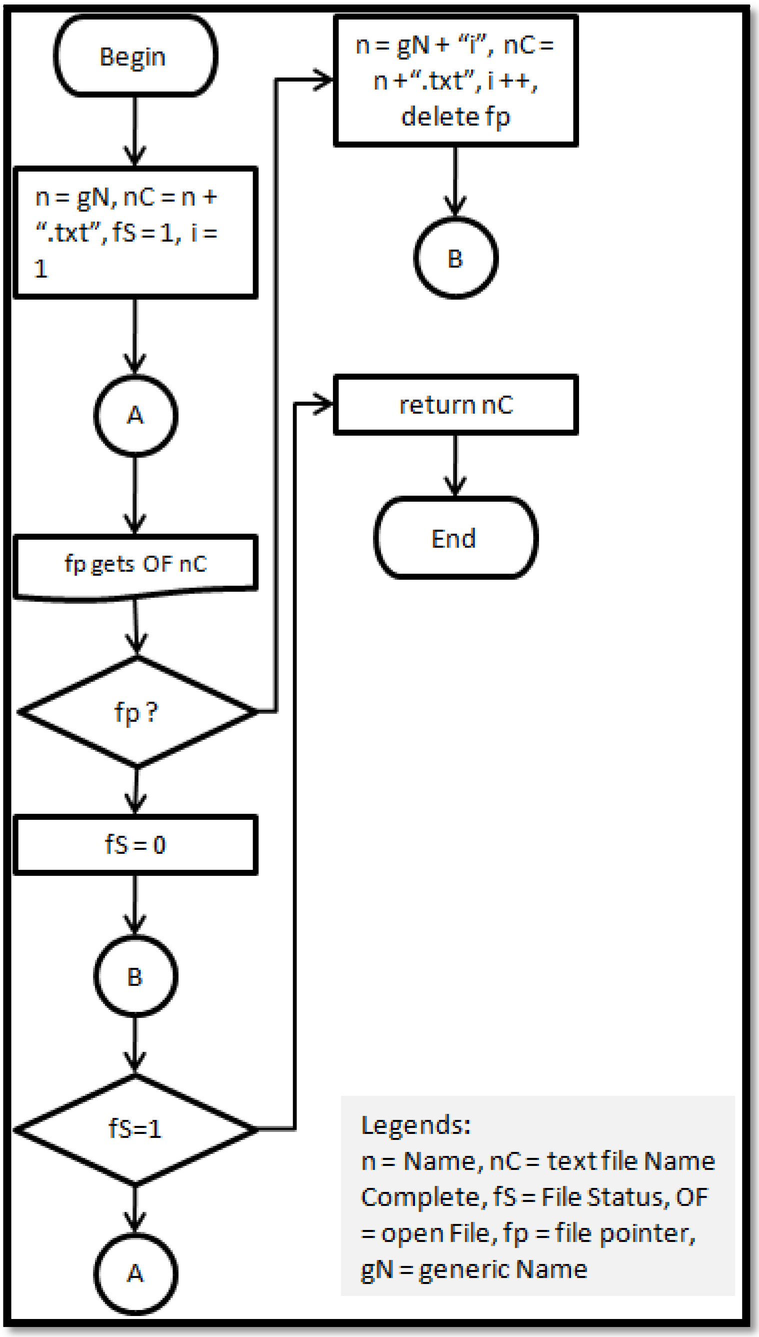
This phase of the subroutine is the same for both the bounded and unbounded case. The subroutine shown in Figure 3 chiefly deals with preparing particles for transmission. There is a file which has the parameters of each required simulation. A complete set of data for a simulation is contained on a line. The number of different simulations is equal to the lines. The values required for a simulation is given by the five parameters dD, ttl, tPN and dT which correspond to destination (or receiver) distance, time to live, total particle number, and time step. This algorithm takes (mimics) the encoding process of the transmitter in that it gets the emission requirements from the file and passes it on to the subroutine “OneSimulation” which in turn carries out much more complex work on the particles meant for transmission. The remaining parts in the flow chart are there to allow for interactivity with the user and minimize error in the system. The program allows for multiple simulations to be carried out.

3.2.1.2 The Subroutine “OneSimulation”

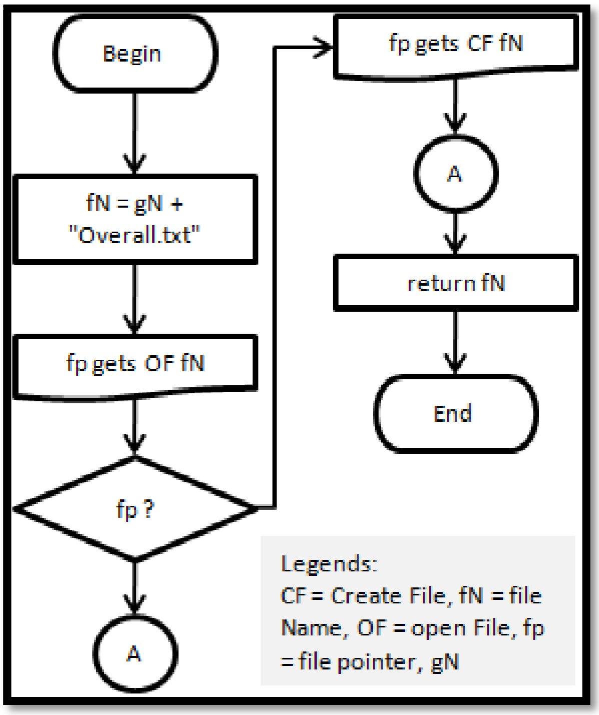


Legends
tLPC = total lost Particle Count, pN = Particle Number, gN = generic Name, fN = file name, fNG = File Name General, dD = Destination Distance, dT = time step, ttl = time to live, aP = all Particles, tPN = total Particle Number, aF = arrival Fraction, aPAT = average Particle Arrival Time, SO = Standard Output, CF = Created file, APF = Appended File

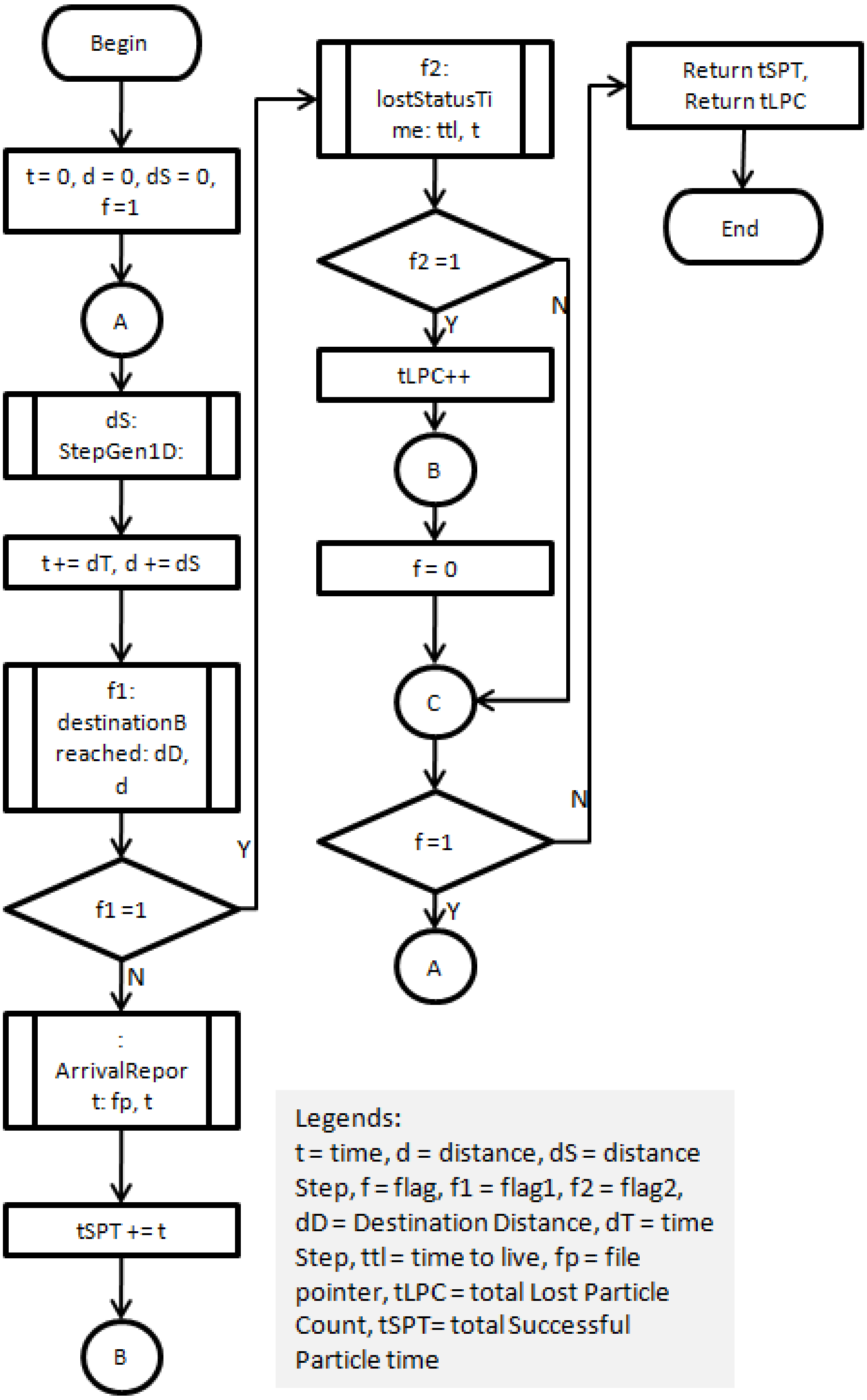
Figure 4 shows the actions of the subroutine “OneSimulation”. This part mostly is concerned with setting up unique files to record the collected hit times to the receiver. The subroutine in Figure 5 is called “AboutFiles”. “AboutFiles” looks for a unused file name with the smallest integer value attachment, which is then sent back to the calling function.


Begin
n = gN, nC = n + “.txt”, fS = 1, i = 1
A
fp gets OF nC
fp ?
fS = 0
B
fS=1
A
n = gN + “i”, nC = n +“.txt”, i ++, delete fp
B
return nC
End
Legends:
n = Name, nC = text file Name Complete, fS = File Status, OF = open File, fp = file pointer, gN = generic Name

Another function of the "OneSimulation" subroutine is that it maintains a log file of all the unique files that have ever been created in the transmission process. This takes note of the certain attributes of that transmission such as the unique file name, average time of arrival, the number of particles that reached to the destination, and also the total number of particles that were transmitted in that transmission. This helps the process of comparison of data across simulations during the analysis. Another routine "AboutFilesGeneral" shown in Figure 6 does the creation or appending and then returns a complete file name to the calling function. The file in this case as in the case of creating the unique files is dependent on a general name given. This simulation, at the end of each unique transmission, prints to the screen the file name of both the log file and the unique file whose run was just concluded. This helps inform the user where and what to look for in monitoring the progress of the simulation.

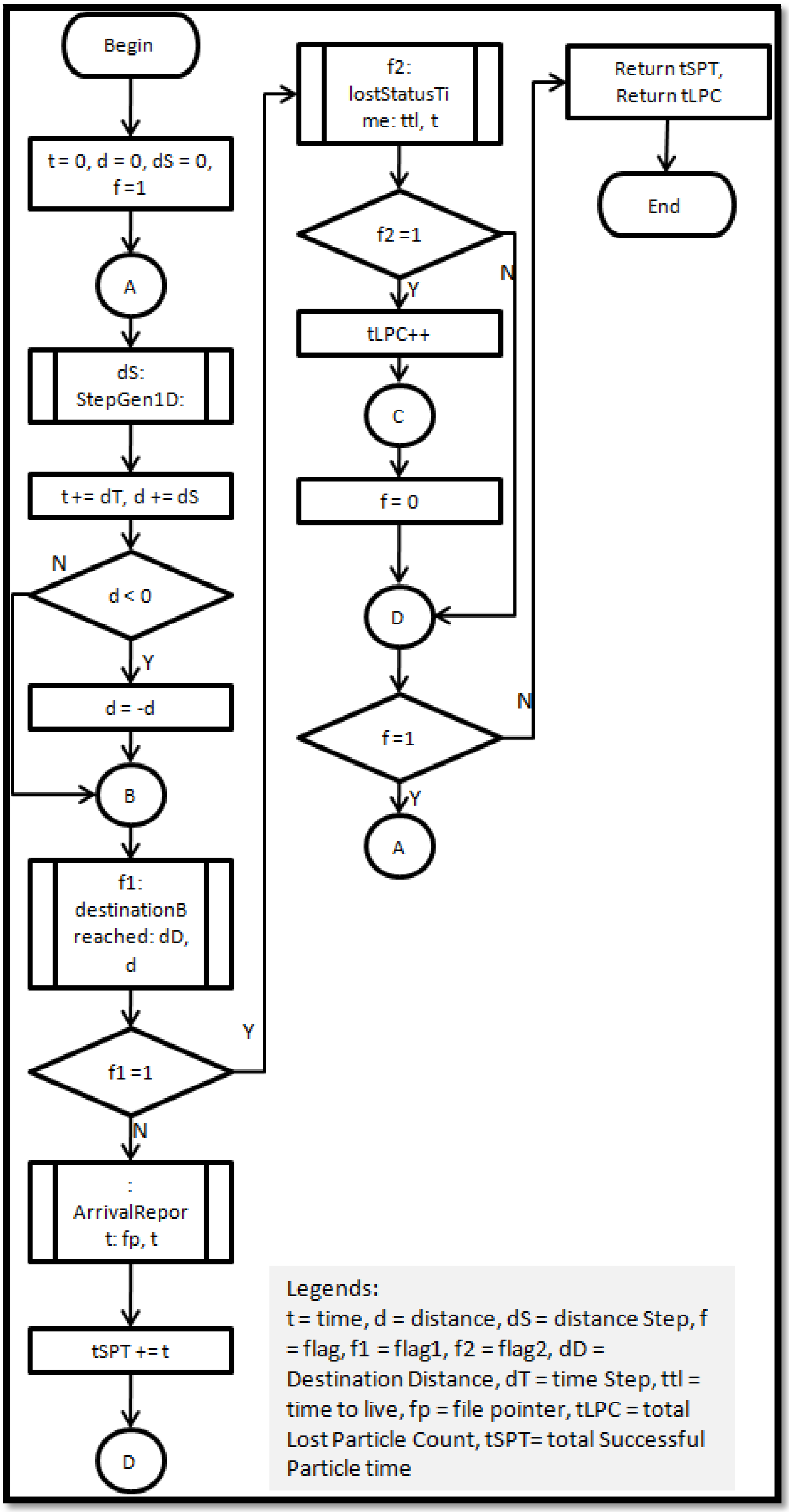


3.2.1.3 The Subroutine “ParticleJourney”



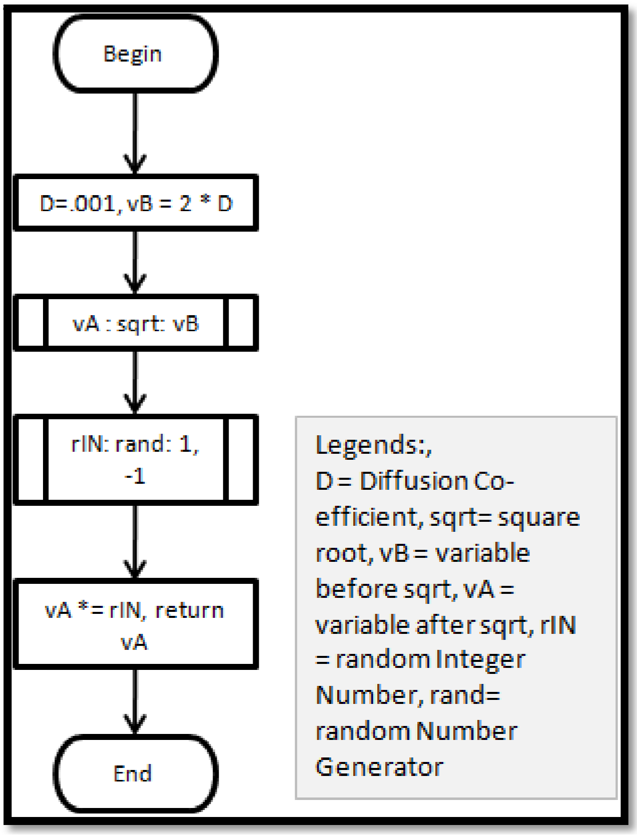
Legends:
t = time, d = distance, dS = distance Step, f = flag, f1 = flag1, f2 = flag2, dD = Destination Distance, dT = time Step, ttl = time to live, fp = file pointer, tLPC = total Lost Particle Count, tSPT= total Successful Particle time

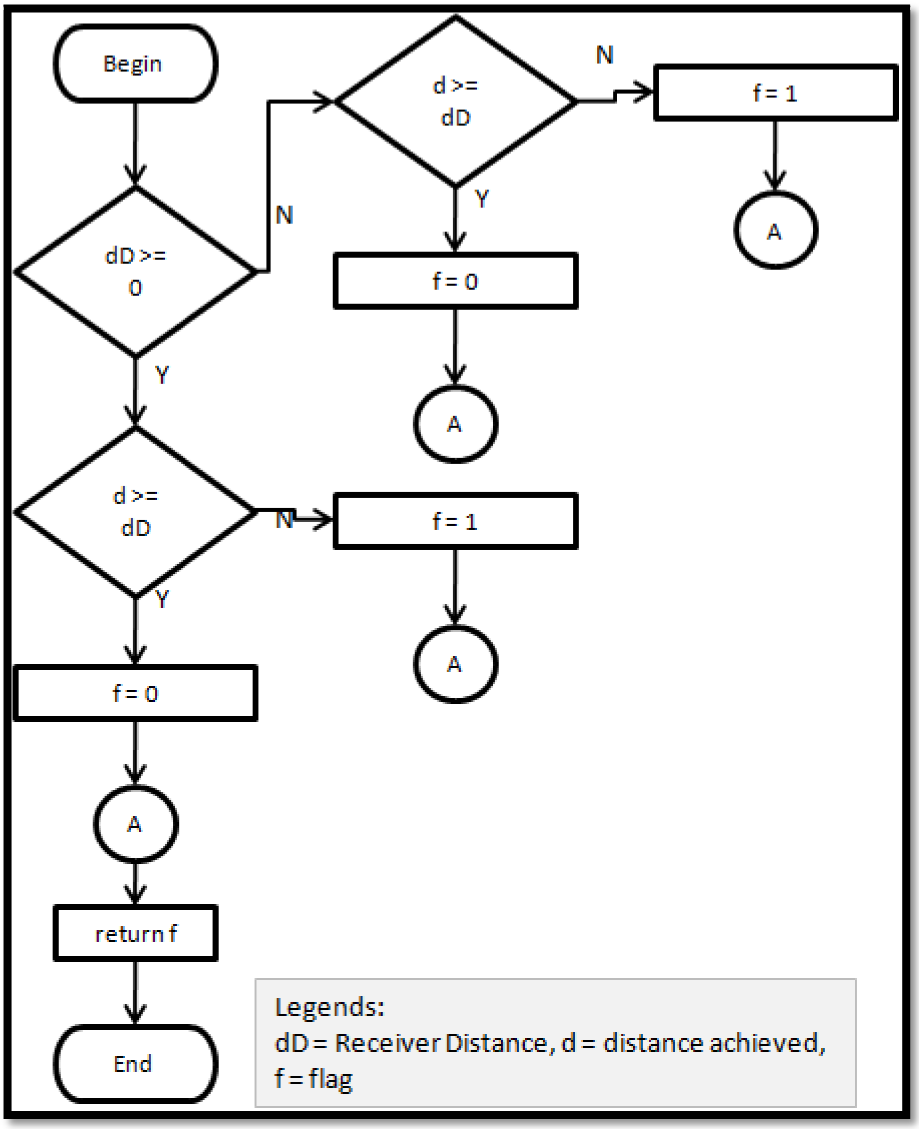
ed)



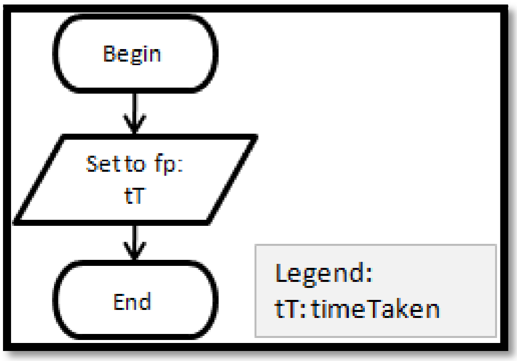
Legends:
t = time, d = distance, dS = distance Step, f = flag, f1 = flag1, f2 = flag2, dD = Destination Distance, dT = time Step, ttl = time to live, fp = file pointer, tLPC = total Lost Particle Count, tSPT= total Successful Particle time

The subroutine responsible for the journey of the each particle, “ParticleJourney”, is the only point where the bounded and the unbound form of the 1D molecular channels part way as illustrated in Figure 7 and Figure 8. The subrountine makes use of several of its own subroutines, for example the “step1DGen” given in Figure 9 generates each step simultaneously as the time increases. The “destinationBreached” subroutine function is to indicate whether the destination has been reached. The “LostStatusTime” indicates whether at any point in time a particle is dead or not. The “ArrivalReport” puts in the unique file created in subroutine “OneSimulation”, the time the particle took to gets to its destination.


Begin
D=.001, vB = 2 * D
vA : sqrt: vB
rIN: rand: 1, -1
vA *= rIN, return vA
End
Legends:, D = Diffusion Co-efficient, sqrt= square root, vB = variable before sqrt, vA = variable after sqrt, rIN = random Integer Number, rand= random Number Generator


Begin
dD >= 0
N
Y
d >= dD
N
Y
f = 1
A
f = 0
A
d >= dD
N
Y
f = 1
A
f = 0
A
return f
End
Legends:
dD = Receiver Distance, d = distance achieved,
f = flag

d”


Begin
Set to fp:
tT
End
Legend:
tT: timeTaken

3.3 Two-Dimensional Setup Analysis

Here only the bounded case will be simulated since the time complexity of the 2D unbounded is high. A pictorial view of what the aforementioned molecule channel is like is given in Figure 12 and Figure 13, respectively.

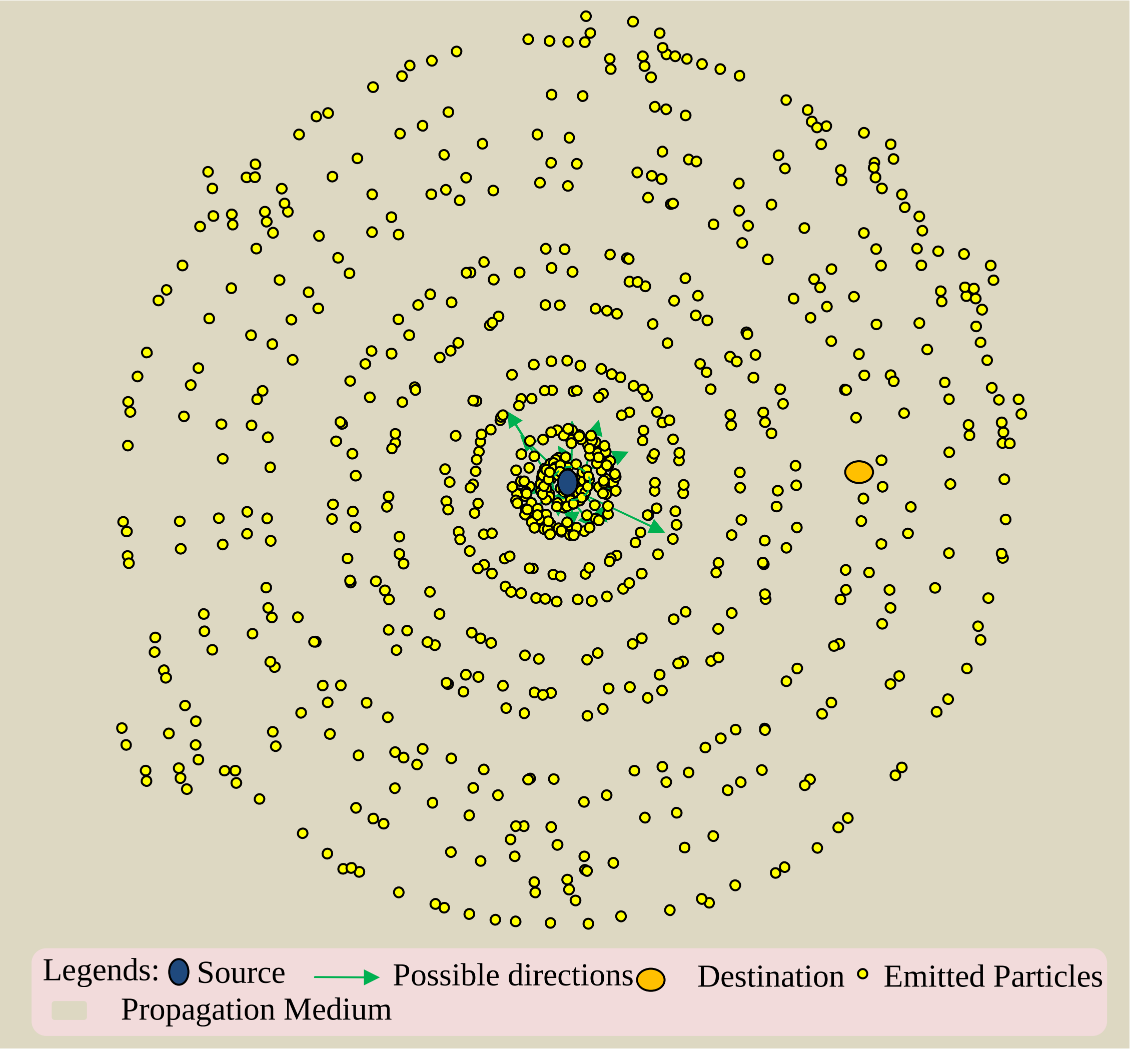


Figure 12: Two-Dimensional Molecular Channel (Unbounded)

The assumption in Figure 12 is that both the source and the receiver have fixed positions in the medium, not free flowing like the emitted particles. In the bounded case, the source and the destination resides at the opposite ends of a junction and there is also the assumption that none of these particles can go beyond the opposite ends. If in their traversal they encounter boundaries, there is a perfect bounce back. In addition, there is another assumption that the channel has a width in which restricts the journeying particles. Again, if there is an attempt at breaching these walls, the particle in question will spring back by a factor equal to the amount it would have crossed that boundary.

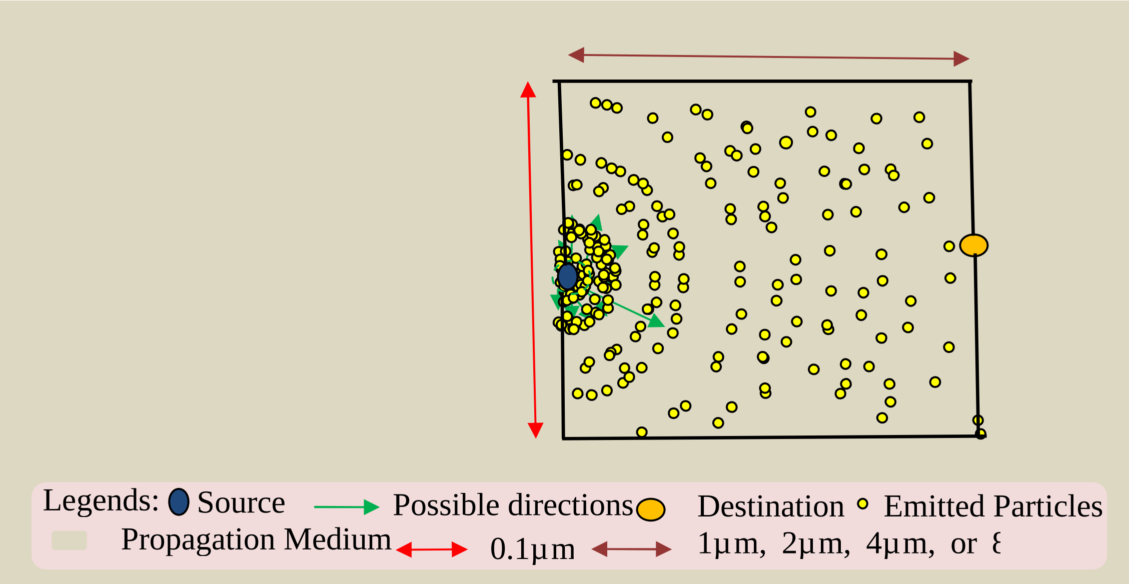


Figure 13: Two-Dimensional Molecular Channel (Bounded)

It can be seen that released particles have two elements to its step and an increased degree of freedom. In Figure 12 the particles are not restricted in any way, hence they can go as they like. In the second case, Figure 13, their movement is much more restricted. The step formulas to account for steps in the x- and y- directions are:

$$\Delta \mathrm{x} = \sqrt[2]{4D\Delta \mathrm{t}}\cos\,(PI * rand2Dim) \qquad (2)$$

$$\Delta \mathrm{y} = \sqrt[2]{4D\Delta \mathrm{t}}\sin\,(PI * rand2Dim) \qquad (3)$$

rand2Dim is a randomly picked number in range [0-2]. The trigonometric functions are in radians. *PI* is 3.141592654. As before, $\Delta t=1\ \mu s$.

3.3.1 Flow Charts of Subroutines Implementing the Required Scenarios (2D)

The algorithm of the 2D case is closely related to the 1D one but differs in minor details such as the data to be read from the configuration file, the generation of the steps, and of course, the complexity of the boundary conditions in the bounded case. Most of the subroutines employed for the 1D case are employed in this case too. The new configuration reading not present in the previous is *w*, which stands for the width which gives us the upper and lower boundary of our molecular channel. The 2D case also differs in the number of subroutines. The reason for the differences lies in the physical difference as showed in their pictorial world view as depicted in Figure 12 and Figure 13. In the figures, it can be seen the receiver is an aperture that has a width equal the 1/20 of the size of the width of the channel. The check as to whether the destination has been reached is as follows:

$$(X\_receiver - X\_current\_particle)^2 + (Y\_receiver - Y\_current\_particle)^2 \leq Aperture\_Width^2 \quad (4)$$

where

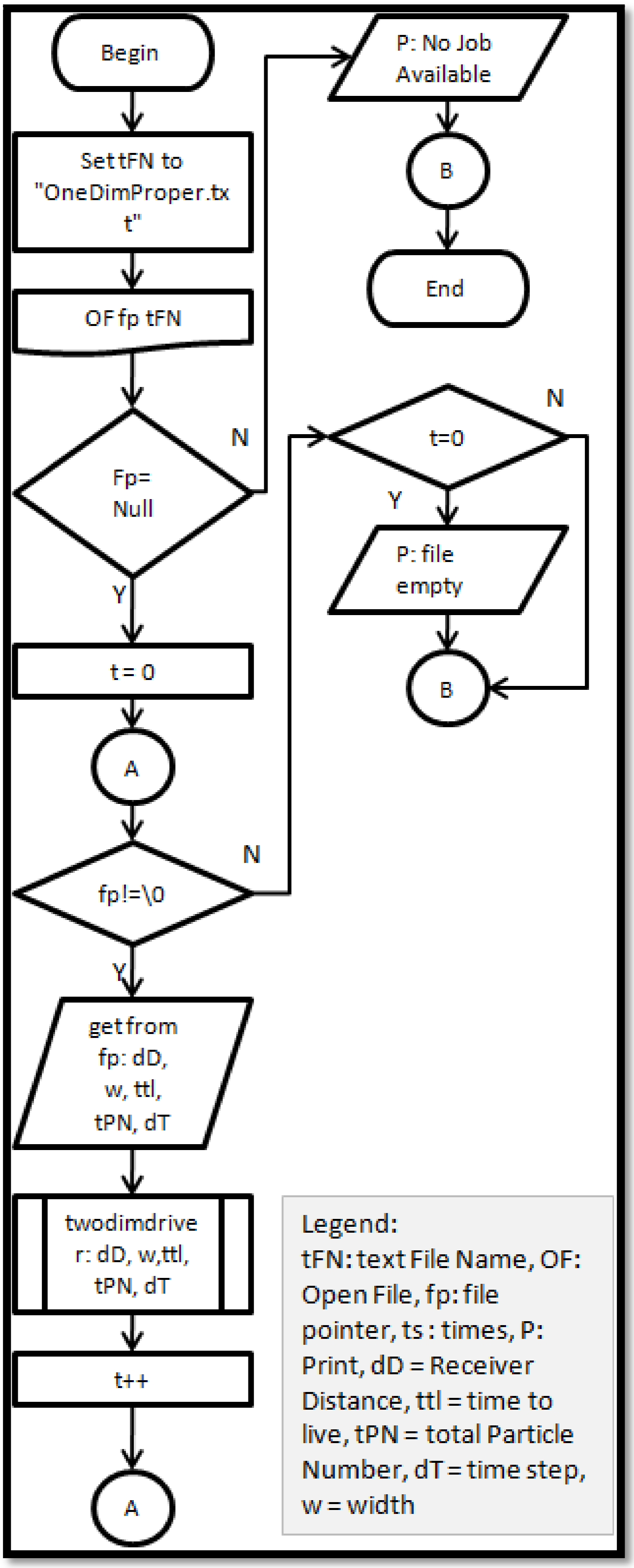
X_receiver is the x-component to the position of the receiver,

Y_receiver is the y-component to the position of the receiver,

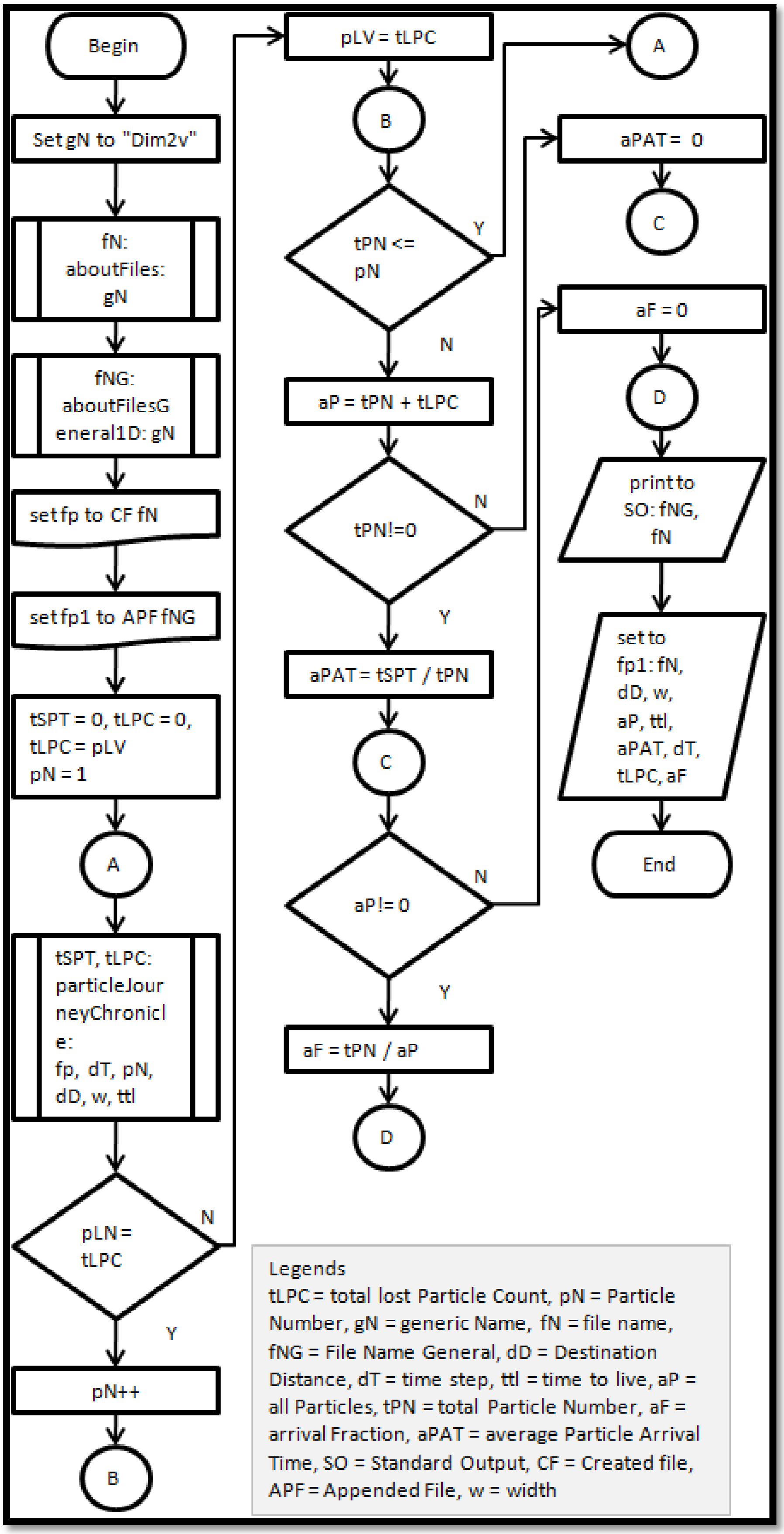
X_ current_particle is the x-component of the current position of the emitted particle, Y_ current_particle is the y-component of the current position of the emitted particle,

Aperture_Width is the radius of the aperture that makes up the receiver.Figure 14 to Figure 18 show the remaining subroutines.

3.3.1.1 The Subroutine “TwoDimensional”

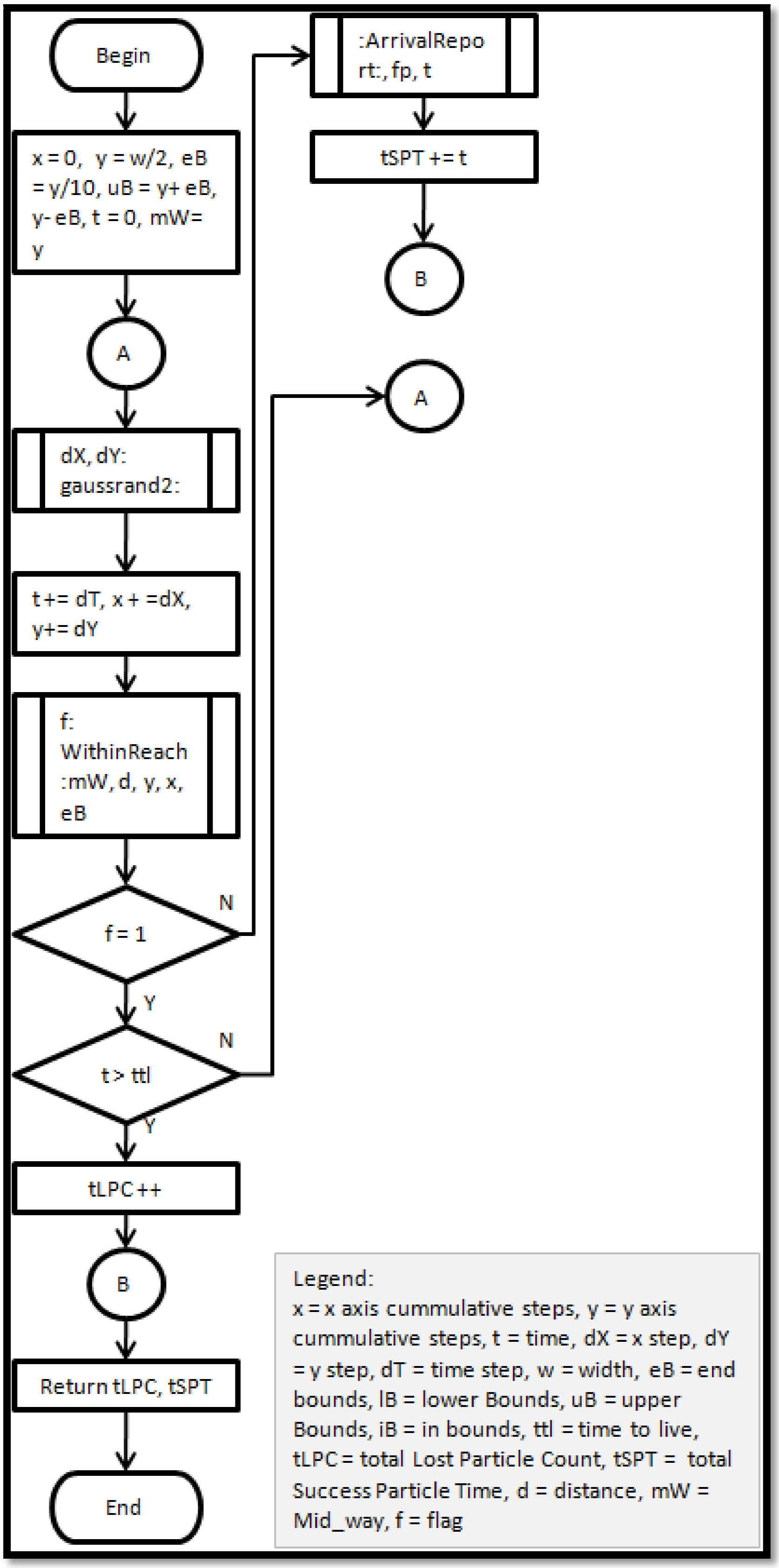


Begin
Set tFN to "OneDimProper.txt"
OF fp tFN
Fp= Null
N
Y
t = 0
A
fp!=\0
N
Y
get from fp: dD, w, ttl, tPN, dT
twodimdriver: dD, w,ttl, tPN, dT
t++
A
P: No Job Available
B
End
t=0
N
Y
P: file empty
B
Legend:
tFN: text File Name, OF: Open File, fp: file pointer, ts : times, P: Print, dD = Receiver Distance, ttl = time to live, tPN = total Particle Number, dT = time step, w = width

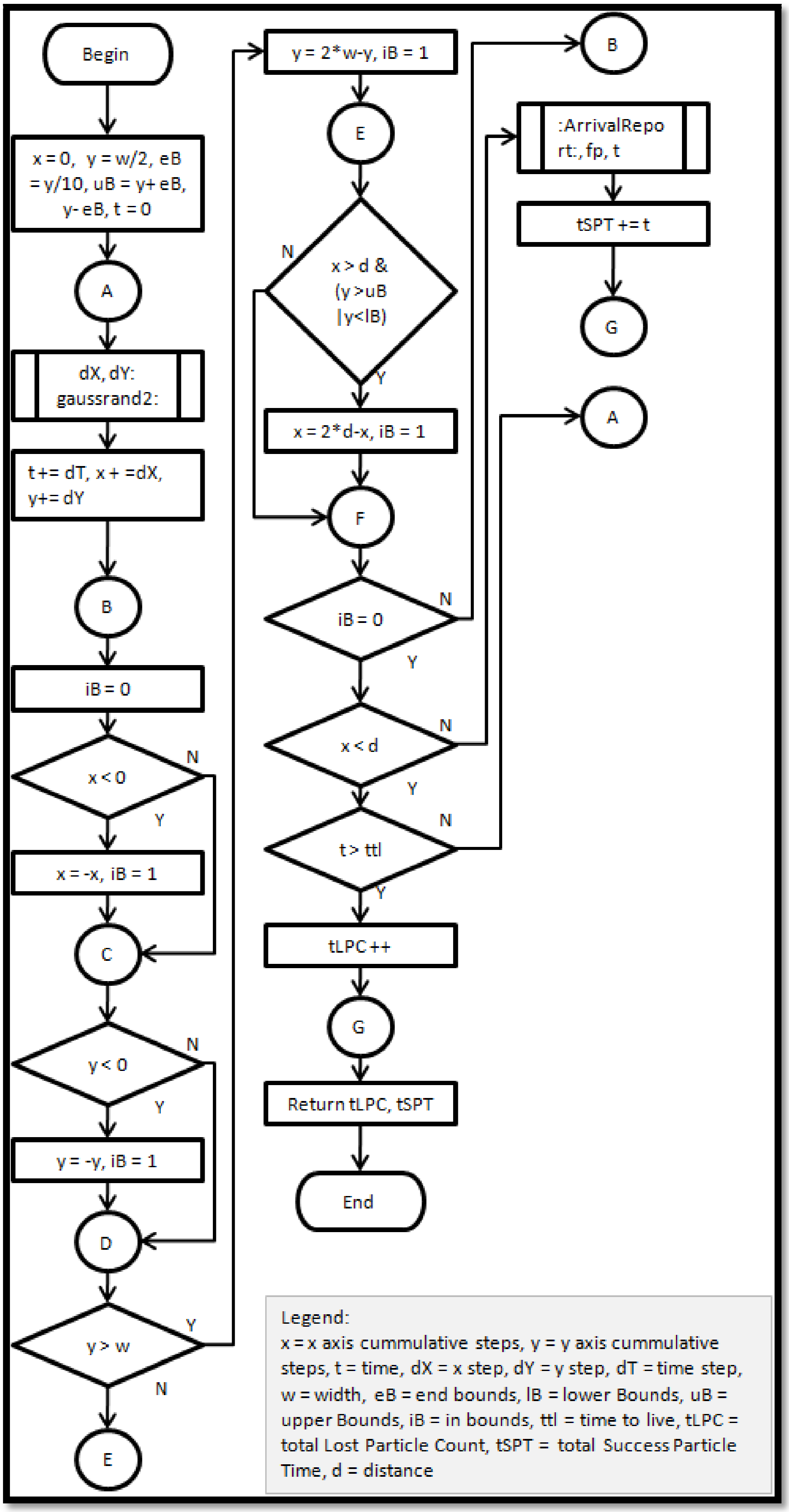


Legends
tLPC = total lost Particle Count, pN = Particle Number, gN = generic Name, fN = file name, fNG = File Name General, dD = Destination Distance, dT = time step, ttl = time to live, aP = all Particles, tPN = total Particle Number, aF = arrival Fraction, aPAT = average Particle Arrival Time, SO = Standard Output, CF = Created file, APF = Appended File, w = width

alDriver”

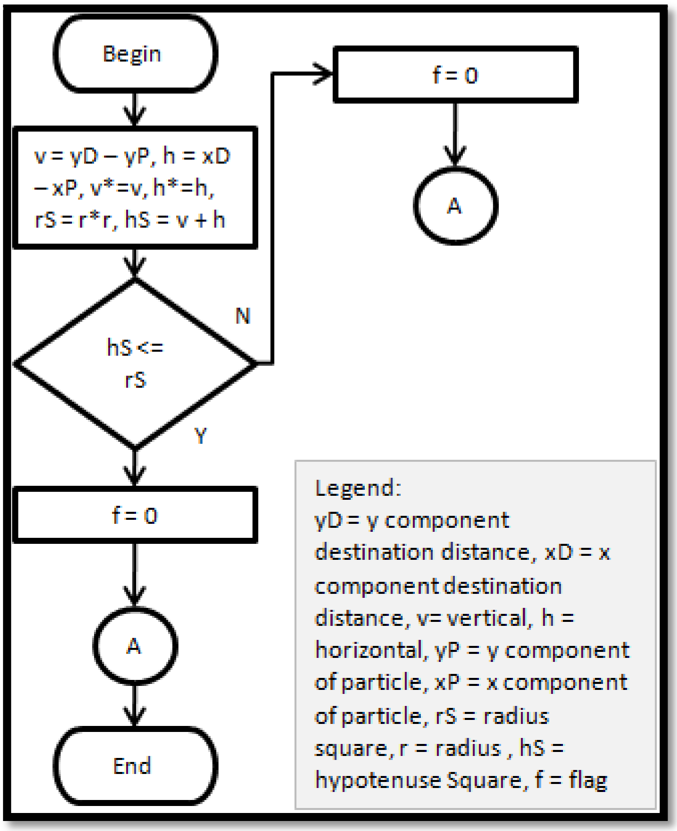

Begin
x = 0, y = w/2, eB = y/10, uB = y+ eB, y- eB, t = 0, mW= y
A
dX, dY: gaussrand2:
t+= dT, x + =dX, y+= dY
f: WithinReach :mW, d, y, x, eB
f = 1
N
Y
t > ttl
N
Y
tLPC++
B
Return tLPC, tSPT
End
:ArrivalReport:, fp, t
tSPT += t
B
A
Legend:
x = x axis cummulative steps, y = y axis cummulative steps, t = time, dX = x step, dY = y step, dT = time step, w = width, eB = end bounds, lB = lower Bounds, uB = upper Bounds, iB = in bounds, ttl = time to live, tLPC = total Lost Particle Count, tSPT = total Success Particle Time, d = distance, mW = Mid_way, f = flag

Unbounded)



Legend:
x = x axis cummulative steps, y = y axis cummulative steps, t = time, dX = x step, dY = y step, dT = time step, w = width, eB = end bounds, lB = lower Bounds, uB = upper Bounds, iB = in bounds, ttl = time to live, tLPC = total Lost Particle Count, tSPT = total Success Particle Time, d = distance

(Bounded)


Begin
v = yD – yP, h = xD – xP, v*=v, h*=h, rS = r*r, hS = v + h
hS <= rS
N
Y
f = 0
A
f = 0
A
End
Legend:
yD = y component destination distance, xD = x component destination distance, v= vertical, h = horizontal, yP = y component of particle, xP = x component of particle, rS = radius square, r = radius , hS = hypotenuse Square, f = flag

3.4 Other Tools Employed

The algorithms presented above generate the data. The algorithms are implemented in C/C++. To analyze and present the data, Microsoft Excel and EasyFit software from Mathwave (http://www.mathwave.com) are used. In the next chapter, the results and analysis are presented.

Chapter 4

RESULTS AND ANALYSIS

4.1 Histograms of Propagation Delay in One-Dimensional (1D) and Two-Dimensional (2D) Molecular Communication Channel Scenarios

The histograms of propagation delay data collected from representative 1D and 2D unbounded and bounded simulation runs are shown in Figure 19 through Figure 21. In each simulation, 1000 particles are transmitted. In each case, there are 50 bins. The size of each bin, average arrival times (propagation delay), and number of lost particles are also reported. The scenarios are for source-destination separation distances 1, 2, 4, and 8 μm.

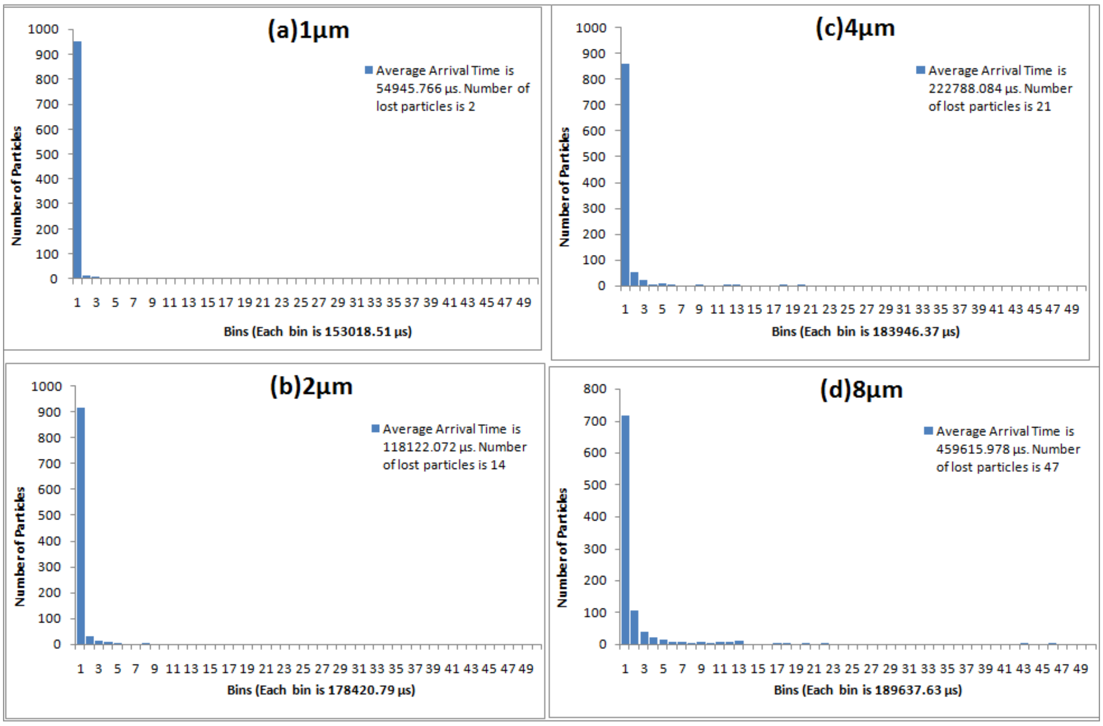


Figure 19: Histograms of Propagation Delay for the 1D Unbounded Case

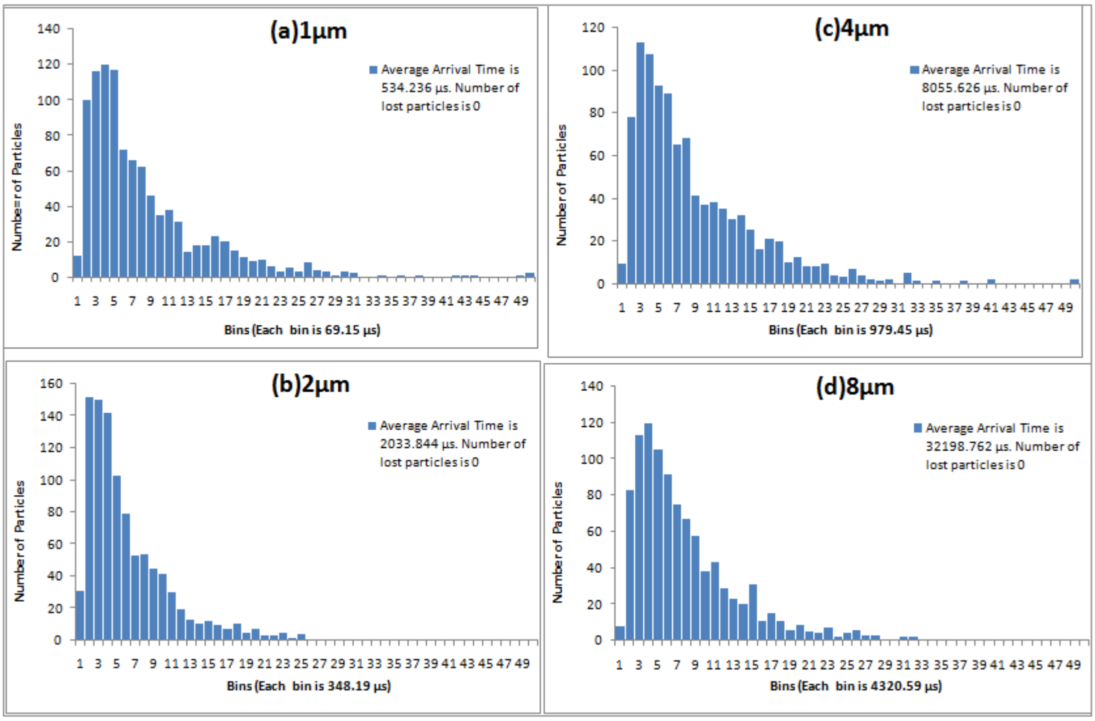


Figure 20: Histograms of Propagation Delay for the 1D Bounded Case

1D histograms reveal the fact that unbounded scenarios have propagation times that are very widely dispersed. The bounded ones on the other hand have less variance.

Table 1: 1D Average Propagation Times in µs (U: Unbounded, B: Bounded)

C_x/R_y	C[1-8]	UB: 1µm	UB: 2µm	UB: 4µm	UB: 8µm	B: 1µm	B: 2µm	B: 4µm	B: 8µm
R[1-8]		54945.77	118122	222788.1	459616	534.24	2033.844	8055.63	32199
UB: 1µm	54945.77	1	0.46516	0.246628	0.119547	102.85	27.01572	6.82079	1.706
UB: 2µm	118122.1	2.149794	1	0.530199	0.257002	221.1	58.07825	14.6633	3.669
UB: 4µm	222788.1	4.054691	1.88608	1	0.484727	417.02	109.5404	27.6562	6.919
UB: 8µm	459616	8.364903	3.89102	2.063019	1	860.32	225.9839	57.0553	14.27
B: 1µm	534.236	0.009723	0.00452	0.002398	0.001162	1	0.262673	0.06632	0.017
B: 2µm	2033.844	0.037015	0.01722	0.009129	0.004425	3.807	1	0.25247	0.063
B: 4µm	8055.626	0.14661	0.0682	0.036158	0.017527	15.079	3.960789	1	0.25
B: 8µm	32198.76	0.58601	0.27259	0.144526	0.070056	60.271	15.83148	3.99705	1

In Table 1, the ratios of the propagation delays (value in a column to a value in a row) across 1D scenarios are provided. The difference between same distance considerations across the bounded-unbounded is highlighted in yellow. Across intermediate distance considerations but the same category (e.g. bounded) differences are given by blue color. Those which compare the disparity of all the other distance considerations to the nearest distance consideration are highlighted in red.

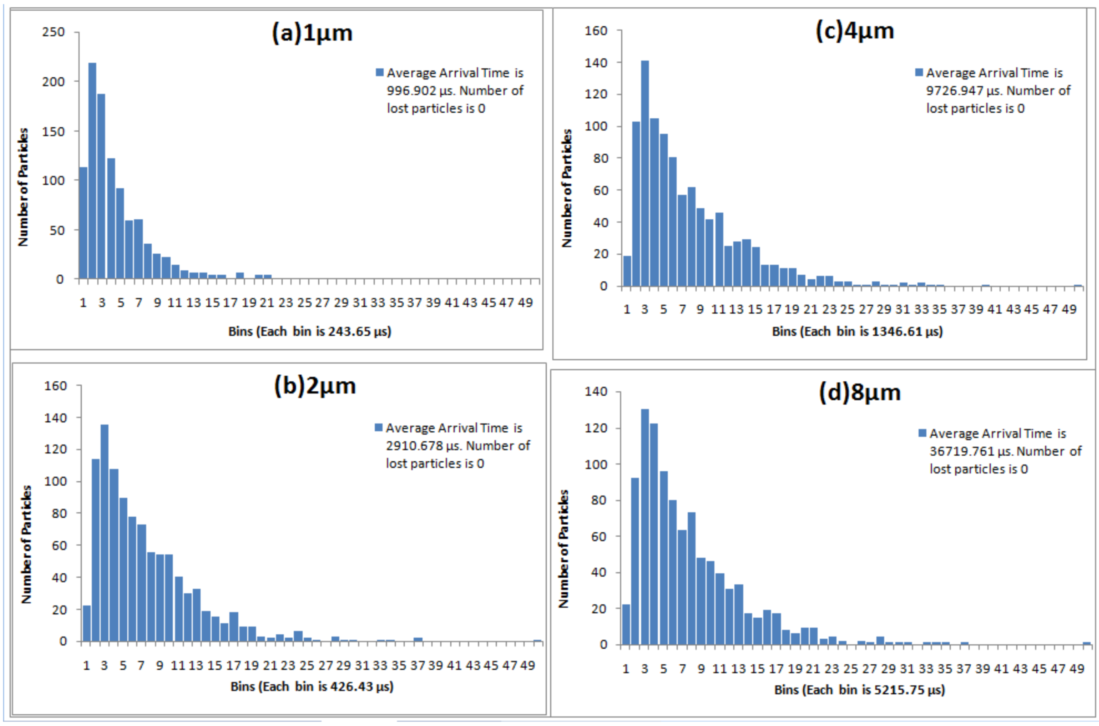


Figure 21: Histograms of Propagation Delay for the 2D Bounded Case

The histograms of propagation delay data collected from 2D bounded simulation runs are shown in Figure 21. 2D unbounded simulations are left as future work due to their time complexity.

4.3 Fitting Delay Data to Distributions

Although there are analytical formulas for the distribution of hitting times of particles in unbounded environments, there are no closed-form formulas for the delay distribution of freely diffusing particles in arbitrary, bounded environments. In this section, data from simulations of bounded settings will be considered and matched to well-known distributions. The distributions investigated are (1) Gamma, (2) Gamma (3P), (3) Inverse Gaussian, (4) Inverse Gaussian (3P), (5) Log-Gamma, (6) Lognormal, (7) Lognormal (3P), (8) Weibull, (9) Weibull (3P) (see Appendix A). These distributions are those commonly used for modeling “time to complete a task.” In order to have reliable results, each scenario is repeated 10 times and the averages are reported. There are 10 rows in the each distribution fitting table. These 10 rows report the parameters fitted and the 95% Kolmogorov-Smirnov (KS) test results (see Appendix B). The last row gives the average values and the number of “accepts” obtained in 10 runs.

4.3.1 One-Dimensional Scenarios

4.3.1.1 One Micrometer, One-Dimensional

Table 2: Fitting 1 µm Data to Distributions 1-3

Gamma.				Gamma(3P)					Inv.Gaussian.		
α	β	KS test		α	β	γ	KS test		λ	μ	KS test
1,3048	409,43	Reject		1,3826	359,94	36,567	Reject		697,09	534,24	Accept
1,3236	414,98	Reject		1,3958	370,2	32,554	Reject		727,04	549,28	Reject
1,5211	330,66	Reject		1,4315	320,46	44,255	Accept		765,08	502,98	Reject
1,4347	351,98	Reject		1,4151	326,78	42,547	Reject		724,5	504,98	Reject
1,4197	380,08	Reject		1,4798	341,43	34,387	Accept		766,11	539,62	Accept
1,3168	423,2	Reject		1,4826	351,45	36,206	Accept		733,76	557,25	Accept
1,4679	358,79	Reject		1,4129	342,68	42,526	Reject		773,12	526,68	Reject
1,3732	403,65	Reject		1,2954	388,77	50,682	Accept		761,12	554,28	Reject
1,4476	350,93	Reject		1,4998	315,81	34,384	Reject		735,45	508,03	Accept
1,6673	308,55	Reject		1,5894	299,72	38,063	Accept		857,72	514,44	Reject
1,42767	373,225	0		1,43849	341,724	39,2171	5		754,099	529,178	4

Table 3: Fitting 1 µm Data to Distributions 4-6

Inv.Gaussian(3P)					Log-Gamma.				Lognormal.		
λ	μ	γ	KS test		α	β	KS test		σ	μ	KS test
672,96	545,83	-11,598	Accept		54,14	0,11008	Accept		0,80957	5,9598	Accept
695,57	565,65	-16,373	Accept		52,724	0,11344	Accept		0,82333	5,9813	Accept
731,78	516,06	-13,082	Accept		59,103	0,10037	Accept		0,77123	5,9321	Accept
659,29	510,24	-5,2563	Accept		57,608	0,10287	Reject		0,78037	5,926	Accept
793,98	562,48	-22,859	Accept		56,879	0,10526	Accept		0,79348	5,9873	Accept
839	581,21	-23,966	Accept		58,911	0,10221	Accept		0,78414	6,0216	Accept
695,98	535,57	-8,8945	Accept		57,282	0,10413	Accept		0,78773	5,9649	Accept
692,82	563,67	-9,3917	Accept		55,533	0,1081	Accept		0,80514	6,0029	Accept
752,44	526,56	-18,533	Accept		57,646	0,10295	Accept		0,78122	5,9344	Accept
878,54	539,29	-24,851	Accept		62,5	0,09553	Accept		0,75484	5,9705	Accept
741,236	544,656	-15,4805	10		57,2326	0,104494	9		0,789105	5,96808	10

Table 4: Fitting 1 µm Data to Distributions 7-9

Lognormal(3P)				Weibull.				Weibull(3P)			
σ	μ	γ	KS test	α	β	KS test		α	β	γ	KS test
0,85207	5,9072	14,116	Accept	1,5207	564,17	Reject		1,1621	526,36	36,872	Reject
0,85614	5,9414	10,902	Accept	1,4987	579,66	Reject		1,1718	547,79	32,858	Accept
0,80588	5,8874	12,108	Accept	1,5991	538,81	Reject		1,1988	488,65	44,812	Accept
0,83837	5,8519	19,379	Accept	1,5742	538,52	Reject		1,184	491,54	42,858	Reject
0,80693	5,9704	4,8518	Accept	1,5608	574,32	Reject		1,2108	540,34	34,814	Accept
0,79179	6,0119	2,9313	Accept	1,5816	591,32	Reject		1,1987	555,91	36,813	Reject
0,83894	5,9001	17,617	Accept	1,5609	561,74	Reject		1,1877	514,94	42,845	Reject
0,86267	5,932	19,65	Accept	1,5289	587,89	Reject		1,1348	528,26	50,894	Accept
0,80145	5,9085	7,0541	Accept	1,5833	541,89	Reject		1,2185	507,41	34,816	Reject
0,75865	5,9655	1,4788	Accept	1,6414	554,8	Reject		1,2723	514,96	38,728	Reject
0,82129	5,9276	11,0088	10	1,56496	563,312	0		1,194	521,616	39,631	4

These distributions ordered from the worst to the best are as follows: Gamma.(0), Weibull.(0), Weibull(3P)(4), Inv.Gaussian.(4), Gamma(3P)(5), Log-Gamma.(9), Inv.Gaussian(3P)(10), Lognormal.(10) and Lognormal(3P)(10).

4.3.1.2 Two Micrometer One-Dimensional

Table 5: Fitting 2 µm Data to Distributions 1-3

Gamma.				Gamma(3P)					Inv.Gaussian.		
α	β	KS test		α	β	γ	KS test		λ	μ	KS test
1,3316	1527,4	Reject		1,2992	1416,6	193,4	Reject		2708,3	2033,8	Accept
1,5933	1247,8	Reject		1,5198	1221,1	132,36	Accept		3167,8	1988,2	Reject
1,4046	1405,7	Reject		1,4417	1258,1	160,75	Reject		2773,3	1974,5	Accept
1,4183	1465,5	Reject		1,4046	1372,2	151,1	Accept		2947,8	2078,5	Reject
1,3869	1484	Reject		1,4373	1340,8	130,96	Accept		2854,3	2058,1	Reject
1,3504	1499,4	Reject		1,3783	1362,2	147,21	Accept		2734,2	2024,7	Accept
1,5277	1359	Reject		1,4123	1350,5	168,82	Accept		3171,7	2076,1	Reject
1,5193	1428,3	Reject		1,4806	1352,1	168,18	Accept		3297	2170,1	Reject
1,431	1409,1	Reject		1,3577	1361,8	167,64	Reject		2885,6	2016,5	Reject
1,623	1273,1	Reject		1,5274	1255,1	149,12	Accept		3353,4	2066,2	Reject
1,45861	1409,93	0		1,4259	1329,1	156,95	7		2989,3	2048,7	3

Table 6: Fitting 2 µm Data to Distributions 4-6

Inv.Gaussian(3P)					Log-Gamma				Lognormal.		
λ	μ	γ	KS test		α	β	KS test		σ	μ	KS test
2580	2062	-28,167	Accept		84,721	0,08625	Accept		0,79352	7,3075	Accept
3021,1	2070,8	-82,656	Accept		86,638	0,08429	Accept		0,78416	7,3026	Accept
2781,5	2017	-42,497	Accept		89,203	0,08176	Accept		0,77178	7,2929	Accept
2816,3	2141,3	-62,835	Accept		83,486	0,08779	Accept		0,80173	7,3292	Accept
2969,8	2153,6	-95,528	Accept		82	0,08924	Accept		0,80769	7,3176	Accept
2951,5	2123,6	-98,819	Accept		81,751	0,08928	Accept		0,80687	7,2991	Accept
3010,7	2150,7	-74,556	Accept		86,267	0,08508	Accept		0,78987	7,34	Accept
3132,1	2228	-57,893	Accept		90,74	0,08147	Accept		0,77567	7,3925	Accept
2370,8	2022	-5,4974	Accept		82,642	0,08829	Accept		0,80225	7,2967	Accept
3467,9	2180,3	-114,12	Accept		90,584	0,08114	Accept		0,77183	7,3497	Accept
2910,2	2114,93	-66,2568	10		85,8032	0,085459	10		0,79054	7,32278	10

Table 7: Fitting 2 µm Data to Distributions 7-9

Lognormal(3P)					Weibull.				Weibull(3P)			
σ	μ	γ	KS test		α	β	KS test		α	β	γ	KS test
0,85136	7,235	74,786	Accept		1,5521	2154,2	Reject		1,1304	1928,3	194,58	Reject
0,79991	7,2826	21,511	Accept		1,5778	2131,8	Reject		1,2445	1996,6	134,12	Accept
0,81046	7,2429	52,709	Accept		1,5957	2102,4	Reject		1,1885	1930,3	162,4	Reject
0,83373	7,2893	42,691	Accept		1,5401	2208,3	Reject		1,183	2047,6	152,4	Reject
0,8177	7,3052	13,301	Accept		1,5346	2185,7	Reject		1,1948	2053,3	132,36	Accept
0,81234	7,2923	7,186	Accept		1,537	2144,4	Reject		1,1675	1987,2	148,47	Reject
0,81496	7,3083	34,815	Accept		1,5648	2219,5	Reject		1,197	2031,2	170,3	Accept
0,80883	7,3499	49,554	Accept		1,5908	2325,1	Reject		1,2196	2143,4	170,17	Accept
0,88084	7,1994	96,642	Accept		1,5288	2143,7	Reject		1,1635	1954,9	168,52	Reject
0,7677	7,355	-6,2038	Accept		1,6065	2220,3	Reject		1,248	2062,2	151,96	Reject
0,81978	7,28599	38,69912	10		1,56282	2183,54	0		1,19368	2013,5	158,528	4

These distributions ordered from the worst to the best are as follows: Gamma.(0), Weibull.(0), Inv.Gaussian.(3), Weibull(3P)(4), Gamma(3P)(7), Log-Gamma(10), Inv.Gaussian(3P)(10), Lognormal.(10) and Lognormal(3P)(10).

4.3.1.3 Four Micrometer One-Dimensional

Table 8: Fitting 4 µm Data to Distributions 1-3

	Gamma.				Gamma(3P)					Inv.Gaussian.	
α	β	KS test		α	β	γ	KS test		λ	μ	KS test
1,541	5227,5	Reject		1,4163	5223,9	656,87	Accept		12414	8055,6	Reject
1,3732	6017,6	Reject		1,3678	5548	674,88	Accept		11348	8263,7	Accept
1,5493	5267,7	Reject		1,4143	5275,4	699,85	Reject		12644	8161,1	Reject
1,4384	5477,4	Reject		1,5	4900,8	527,73	Accept		11333	7878,9	Reject
1,2361	6826	Reject		1,4284	5579	468,74	Reject		10430	8437,6	Accept
1,4932	5288,4	Reject		1,5131	4879,5	513,67	Accept		11791	7896,6	Accept
1,4718	5496	Reject		1,5378	4931,8	504,78	Reject		11906	8089,1	Accept
1,5154	5374,4	Reject		1,4813	5084	613,46	Accept		12341	8144,2	Reject
1,3761	6177,5	Reject		1,2497	6183	774,18	Accept		11698	8501	Reject
1,5417	5270,5	Reject		1,4802	5069	622,38	Reject		12527	8125,6	Reject
1,45362	5642,3	0		1,43889	5267,44	605,654	6		11843,2	8155,34	4

Table 9: Fitting 4 µm Data to Distributions 4-6

	Inv.Gaussian(3P)					Log-Gamma				Lognormal.	
λ	μ	γ	KS test		α	β	KS test		σ	μ	KS test
11781	8350,4	-294,8	Accept		121,54	0,07156	Accept		0,78855	8,6978	Accept
11620	8550,7	-286,95	Accept		119,87	0,07268	Accept		0,79538	8,7124	Accept
11372	8353,3	-192,28	Accept		123,78	0,07038	Accept		0,78268	8,7123	Accept
11966	8207,9	-329,05	Accept		123,17	0,07044	Accept		0,78133	8,6758	Accept
10189	8579	-141,41	Accept		114,2	0,07631	Accept		0,81511	8,7151	Accept
12887	8348,7	-452,06	Accept		122,67	0,07076	Accept		0,78337	8,6805	Accept
12335	8416,1	-327,02	Accept		125	0,06964	Accept		0,77817	8,7047	Accept
12786	8517,5	-373,34	Accept		125,3	0,06955	Accept		0,77812	8,7144	Accept
10494	8710,2	-209,2	Accept		111,88	0,07798	Reject		0,82441	8,7243	Accept
12859	8501,3	-375,67	Accept		125,86	0,06923	Accept		0,77631	8,7135	Accept
11828,9	8453,51	-298,178	10		121,327	0,071853	9		0,790343	8,70508	10

Table 10: Fitting 4 µm Data to Distributions 7-9

Lognormal(3P)					Weibull.				Weibull(3P)			
σ	μ	γ	KS test		α	β	KS test		α	β	γ	KS test
0,81201	8,6681	127,09	Accept		1,568	8622	Reject		1,1994	7877,3	666,57	Accept
0,8202	8,6813	134,65	Accept		1,5551	8774,3	Reject		1,1631	8017,8	683,15	Accept
0,8248	8,6588	229,9	Accept		1,5738	8738,5	Reject		1,1973	7947,5	705,31	Reject
0,79305	8,6608	63,932	Accept		1,5849	8401,2	Reject		1,2181	7873,5	535,09	Reject
0,8602	8,6594	233,45	Accept		1,5118	8886,9	Reject		1,1748	8460,1	473,84	Reject
0,77479	8,6915	-47,719	Accept		1,5866	8436,9	Reject		1,2309	7922,8	522,65	Reject
0,7913	8,6878	74,121	Accept		1,5902	8636,8	Reject		1,2345	8151,8	512,74	Reject
0,78655	8,7036	48,219	Accept		1,5926	8717	Reject		1,2204	8063,9	620,86	Accept
0,87719	8,6609	263,57	Accept		1,4958	9010,6	Reject		1,1205	8068,3	776,58	Accept
0,78445	8,703	46,679	Accept		1,5951	8705,3	Reject		1,2221	8034,9	630,7	Reject
0,812454	8,67752	117,3892	10		1,56539	8692,95	0		1,19811	8041,79	612,749	4

These distributions ordered from the worst to the best are as follows: Gamma.(0), Weibull.(0), Inv.Gaussian.(4), Weibull(3P)(4), Gamma(3P)(6), Log-Gamma(9), Inv.Gaussian(3P)(10), Lognormal.(10) and Lognormal(3P)(10).

4.3.1.3 Eight Micrometer One-Dimensional

Table 11: Fitting 8 µm Data to Distributions 1-3

Gamma.				Gamma(3P)					Inv.Gaussian.		
α	β	KS test		α	β	γ	KS test		λ	μ	KS test
1,6505	19508	Reject		1,6763	18027	1979,9	Accept		53145	32199	Accept
1,5113	20514	Reject		1,5021	19287	2033,2	Accept		46853	31003	Reject
1,4239	23023	Reject		1,3231	22408	3134	Accept		46681	32783	Reject
1,4798	20937	Reject		1,4458	19781	2383,7	Accept		45848	30983	Reject
1,4499	21720	Reject		1,507	19526	2066,4	Accept		45659	31492	Accept
1,4641	22321	Reject		1,429	20917	2789	Accept		47847	32680	Reject
1,5243	20649	Reject		1,4779	19467	2706,4	Accept		47980	31476	Accept
1,4971	21700	Reject		1,4439	20864	2361,2	Reject		48638	32488	Accept
1,6487	19573	Accept		1,4357	20736	2499,7	Accept		53201	32269	Reject
1,4145	23045	Reject		1,4087	21523	2278,3	Accept		46107	32597	Reject
1,50641	21299	1		1,46495	20253,6	2423,18	9		48196	31997	4

Table 12: Fitting 8 µm Data to Distributions 4-6

Inv.Gaussian(3P)					Log-Gamma				Lognormal.		
λ	μ	γ	KS test		α	β	KS test		σ	μ	KS test
56689	33861	-1662,1	Accept		181,98	0,05555	Accept		0,74906	10,11	Accept
53244	33215	-2212	Accept		161,62	0,06217	Accept		0,78996	10,048	Accept
41251	33128	-345,5	Accept		161,86	0,06235	Accept		0,7929	10,093	Accept
44757	31985	-1001,8	Accept		164,05	0,06123	Accept		0,78384	10,045	Accept
50819	33207	-1715,8	Accept		165,54	0,06079	Accept		0,78173	10,063	Accept
45631	33379	-699,34	Accept		169,05	0,05975	Accept		0,77652	10,101	Accept
52933	33106	-1630,1	Accept		173,75	0,058	Accept		0,76409	10,077	Accept
50246	34141	-1652,9	Accept		162,09	0,06224	Accept		0,79206	10,089	Accept
57725	34900	-2630,1	Accept		161,04	0,06265	Accept		0,79468	10,09	Accept
45436	33797	-1200,1	Accept		157,31	0,06409	Accept		0,80337	10,081	Accept
49873	33471,9	-1474,97	10		165,83	0,060882	10		0,782821	10,0797	10

Table 13: Fitting 8 µm Data to Distributions 7-9

Lognormal(3P)					Weibull.				Weibull(3P)			
σ	μ	γ	KS test		α	β	KS test		α	β	γ	KS test
0,74731	10,112	-43,559	Accept		1,6571	34701	Reject		1,2992	32836	2026,3	Reject
0,76344	10,082	-583,1	Accept		1,578	33177	Reject		1,2328	31082	2067,5	Accept
0,85933	10,01	1372,6	Accept		1,5526	34907	Reject		1,15	31227	3147,9	Accept
0,81002	10,011	550,02	Accept		1,5749	33102	Reject		1,2028	30509	2412	Accept
0,77599	10,07	-127,3	Accept		1,5885	33607	Reject		1,2224	31522	2109,5	Accept
0,81978	10,046	957,13	Accept		1,5883	34915	Reject		1,193	31830	2811	Accept
0,76394	10,077	-3,5643	Accept		1,6277	33768	Reject		1,2203	30791	2738,6	Accept
0,7946	10,086	56,03	Accept		1,565	34688	Reject		1,2068	32156	2391	Accept
0,75808	10,136	-841,88	Accept		1,5684	34685	Reject		1,2239	31828	2537,4	Accept
0,82376	10,056	430,09	Accept		1,5379	34640	Reject		1,1847	32208	2308,3	Accept
0,791625	10,0686	176,6467	10		1,58384	34219	0		1,21359	31598,9	2454,95	9

These distributions ordered from the worst to the best are as follows: Weibull.(0), Gamma.(1), Inv.Gaussian.(4), Weibull(3P)(9), Gamma(3P)(9), Log-Gamma(10), Inv.Gaussian(3P)(10), Lognormal.(10) and Lognormal(3P)(10).

The popular 2 parameter distributions Gamma and Weibull seem not be a fit at all. Even the Inverse Gaussian which just 4 matches almost every time cannot be considered as a good fit. The Weibull (3p) distribution and Gamma(3p), while giving an admirable accuracy in some cases of distance further along, their lack of consistence make them unadvisable for modeling delay. The log-gamma does admirably well by fluctuating only between 9 accepts and 10 accepts during the whole evaluation process. The fit should be considered only second to the Inverse Gaussian(3p), Lognormal and Lognormal (3p) which all through give a steady output of 10 accepts.

4.3.2 Two-Dimensional Scenarios

4.3.2.1 One Micrometer Two-Dimensional

Table 14: Fitting 1 µm Data to Distributions1-3

	Gamma.				Gamma(3P)					Inv.Gaussian.	
α	β	KS test		α	β	γ	KS test		λ	μ	KS test
1,3041	796,38	Reject		1,2345	792,33	60,419	Accept		1354,3	1038,5	Reject
1,3842	703,04	Reject		1,3541	679,39	53,167	Accept		1347	973,14	Reject
1,361	711,88	Reject		1,3482	672,5	62,206	Accept		1318,7	968,9	Reject
1,3102	752,24	Reject		1,2896	725,98	49,425	Accept		1291,4	985,62	Reject
1,4135	688,65	Reject		1,3796	661,28	61,119	Accept		1376	973,43	Reject
1,338	721,1	Reject		1,3368	678,88	57,305	Accept		1291	964,84	Reject
1,3912	643,15	Reject		1,3408	623,08	59,326	Accept		1244,8	894,78	Reject
1,2927	744,86	Reject		1,1844	751,62	72,7	Accept		1244,8	962,91	Reject
1,0761	927,1	Reject		1,3772	694,5	41,22	Reject		1073,6	997,68	Accept
1,4304	691,33	Accept		1,3138	706,87	60,189	Accept		1414,5	988,89	Reject
1,33014	737,973	1		1,3159	698,643	57,7076	9		1295,61	974,869	1

Table 15: Fitting 1 µm Data to Distributions 4-6

Inv.Gaussian(3P)					Log-Gamma				Lognormal.		
λ	μ	γ	KS test		α	β	KS test		σ	μ	KS test
1254,8	1093,2	-54,625	Accept		54,772	0,12016	Reject		0,88883	6,5814	Accept
1493,1	1054	-80,878	Accept		58,351	0,11216	Reject		0,85632	6,5445	Accept
1352,6	1021,7	-52,777	Accept		61,374	0,10667	Accept		0,83528	6,547	Accept
1220,2	1040,3	-54,635	Accept		54,394	0,12009	Reject		0,88528	6,5324	Accept
1380,1	1026,3	-52,834	Accept		62,296	0,10526	Accept		0,83039	6,5574	Accept
1275,2	1013,6	-48,742	Accept		59,239	0,1103	Reject		0,8485	6,5339	Accept
1219,4	939,14	-44,36	Accept		59,846	0,10807	Accept		0,83563	6,4677	Accept
1160	1006,6	-43,705	Accept		55,54	0,11729	Reject		0,87367	6,5143	Accept
1310,8	1051,5	-53,79	Accept		59,229	0,11066	Accept		0,85125	6,5546	Accept
1443,8	1061,8	-72,935	Accept		58,298	0,11251	Reject		0,85859	6,5589	Accept
1311	1030,814	-55,9281	10		58,3339	0,112317	4		0,856374	6,53921	10

Table 16: Fitting 1 µm Data to Distributions 7-9

Lognormal(3P)					Weibull.				Weibull(3P)			
σ	μ	γ	KS test		α	β	KS test		α	β	γ	KS test
0,89627	6,573	4,0229	Accept		1,3953	1086,7	Reject		1,1149	1019,4	60,806	Accept
0,81288	6,5956	-25,486	Accept		1,4576	1029,1	Reject		1,173	973,95	53,717	Accept
0,83517	6,5472	-0,06774	Accept		1,4862	1023,9	Reject		1,162	957,81	62,748	Accept
0,88692	6,5306	0,85599	Accept		1,4016	1032,9	Reject		1,139	982,82	49,793	Accept
0,83058	6,5571	0,11172	Accept		1,4946	1032,3	Reject		1,1802	967,91	61,71	Accept
0,85531	6,5259	3,816	Accept		1,4608	1017,5	Reject		1,1564	957,46	57,767	Accept
0,84437	6,4573	4,6938	Accept		1,483	946,72	Reject		1,1617	882,4	59,772	Accept
0,89705	6,4876	11,973	Accept		1,4171	1009,8	Reject		1,0908	920,92	72,884	Accept
0,84254	6,5648	-5,0259	Accept		1,4625	1037,4	Reject		1,1435	1007,4	41,797	Reject
0,8311	6,591	-15,996	Accept		1,4508	1045,9	Reject		1,1605	979,64	60,721	Accept
0,853219	6,54301	-2,11022	10		1,45095	1026,222	0		1,1482	964,971	58,1715	9

These distributions ordered from the worst to the best are as follows: Weibull.(0), Gamma.(1), Inv.Gaussian.(1), Log-Gamma(4), Weibull(3P)(9), Gamma(3P)(9), Inv.Gaussian(3P)(10), Lognormal.(10) and Lognormal(3P)(10).

4.3.2.2 Two Micrometer Two-Dimensional

Table 17: Fitting 2 µm Data to Distributions 1-3

Gamma.				Gamma(3P)				Inv.Gaussian.		
α	β	KS test		α	β	γ	KS test	λ	μ	KS test
1,3515	2064,9	Reject		1,3674	1906	184,47	Accept	3771,8	2790,7	Reject
1,4177	2016,4	Reject		1,3707	1938,9	201,12	Accept	4053	2858,8	Reject
1,6315	1759,5	Accept		1,681	1644,8	105,76	Accept	4683,3	2870,6	Reject
1,533	1912,9	Reject		1,4045	1942,4	204,36	Accept	4495,4	2932,5	Reject
1,4523	1997,4	Reject		1,3008	2047,1	238,02	Accept	4213,2	2900,9	Reject
1,3381	2184,1	Reject		1,2959	2056	258,23	Reject	3910,8	2922,6	Accept
1,5549	1873,1	Reject		1,3943	1914,9	242,68	Accept	4528,8	2912,5	Reject
1,539	1906,7	Reject		1,5569	1804,2	125,48	Accept	4516,2	2934,4	Reject
1,4152	2087,1	Reject		1,4807	1911,6	123,19	Reject	4180,2	2953,8	Reject
1,3151	2152,6	Reject		1,3451	1967,2	184,63	Accept	3722,8	2830,8	Reject
1,45483	1995,47	1		1,41973	1913,31	186,794	8	4207,55	2890,76	1

Table 18: Fitting 2 µm Data to Distributions 4-6

Inv.Gaussian(3P)				Log-Gamma				Lognormal.		
λ	μ	γ	KS test	α	β	KS test		σ	μ	KS test
3659,1	2890	-99,303	Accept	85,522	0,08898	Accept		0,82246	7,6097	Accept
4103,2	3002,2	-143,44	Accept	87,325	0,0875	Accept		0,81728	7,6411	Accept
5122,2	3090,4	-219,88	Accept	94,952	0,08081	Accept		0,78706	7,6732	Accept
4694,7	3136,4	-203,95	Accept	88,699	0,08654	Reject		0,8146	7,6757	Accept
4125,6	3050,3	-149,38	Accept	86,071	0,08893	Reject		0,82467	7,6547	Reject
3783,8	2997,5	-74,962	Accept	89,917	0,08523	Accept		0,80775	7,6633	Accept
4656,5	3084,9	-172,42	Accept	92,844	0,08272	Reject		0,79668	7,6803	Accept
4856,4	3157,2	-222,76	Accept	88,932	0,08633	Accept		0,81374	7,6777	Accept
4438,7	3146,8	-193,07	Accept	85,43	0,08975	Accept		0,82913	7,6674	Accept
3923,3	2975,2	-144,36	Accept	84,202	0,09049	Accept		0,82994	7,6195	Accept
4336,35	3053,09	-162,353	10	88,3894	0,086728	7		0,814331	7,65626	9

Table 19: Fitting 2 µm Data to Distributions 7-9

Lognormal(3P)				Weibull.			Weibull(3P)			
σ	μ	γ	KS test	α	β	KS test	α	β	γ	KS test
0,84643	7,5806	40,951	Accept	1,5021	2951,6	Reject	1,1652	2756,4	186,26	Accept
0,82259	7,6346	9,6129	Accept	1,5169	3034,5	Reject	1,1738	2815,1	203,15	Reject
0,75226	7,7175	-71,946	Accept	1,5827	3084,5	Reject	1,3103	3007,9	110,14	Accept
0,79295	7,7024	-41,94	Accept	1,5273	3133,2	Reject	1,2023	2906	206,87	Accept
0,83222	7,6455	13,594	Reject	1,5034	3086,2	Reject	1,1524	2803,6	239,29	Accept
0,85096	7,6101	78,452	Accept	1,5268	3094,6	Reject	1,1329	2793,5	259,44	Accept
0,79018	7,6885	-12,919	Accept	1,5599	3122,2	Reject	1,1985	2841,3	244,93	Accept
0,7812	7,7179	-63,95	Accept	1,5308	3136,9	Reject	1,2608	3030,8	128,54	Accept
0,80655	7,6945	-41,931	Accept	1,4999	3128,4	Reject	1,217	3028,8	126,72	Reject
0,8319	7,6172	3,3951	Accept	1,4956	2985	Reject	1,1546	2790,9	186,31	Accept
0,810724	7,66088	-8,6681	9	1,52454	3075,71	0	1,19678	2877,43	189,165	8

These distributions ordered from the worst to the best are as follows: Weibull.(0), Gamma.(1), Inv.Gaussian.(1), Log-Gamma(7), Weibull(3P)(8), Gamma(3P)(8), Lognormal.(9), Lognormal(3P)(9) and Inv.Gaussian(3P)(10).

4.3.2.3 Four Micrometer Two Dimensional

Table 20: Fitting 4 µm Data to Distribution 1-3

Gamma.				Gamma(3P)					Inv.Gaussian.		
α	β	KS test		α	β	γ	KS test		λ	μ	KS test
1,4856	6427,8	Reject		1,451	6073,6	736,34	Accept		14186	9549	Accept
1,4891	6858	Reject		1,3249	7063,9	853,37	Accept		15208	10212	Reject
1,3063	7506,4	Reject		1,2721	7011,6	886,27	Accept		12809	9805,5	Reject
1,4929	6519,9	Reject		1,2954	6809,7	912,51	Accept		14532	9733,7	Reject
1,567	6595,5	Reject		1,4797	6524,6	680,4	Accept		16195	10335	Reject
1,4056	6811,6	Reject		1,3285	6611,2	791,08	Reject		13458	9574,3	Reject
1,4457	7093,1	Reject		1,4481	6684,7	574,53	Accept		14825	10255	Reject
1,5227	6522	Reject		1,3953	6592,1	733,26	Accept		15122	9931	Reject
1,5566	6432,3	Reject		1,6259	5880,7	450,94	Reject		15585	10012	Reject
1,6146	6328,9	Accept		1,4598	6518,2	703,64	Accept		16500	10219	Reject
1,48861	6709,55	1		1,40807	6577,03	732,234	8		14842	9962,65	1

Table 21: Fitting 4 µm Data to Distribution 4-6

	Inv.Gaussian(3P)					Log-Gamma				Lognormal.	
λ	μ	γ	KS test		α	β	KS test		σ	μ	KS test
14691	9976,5	-427,52	Accept		128,21	0,06918	Accept		0,78294	8,8697	Accept
14668	10703	-490,16	Accept		119,67	0,07453	Reject		0,81495	8,9197	Accept
12187	10005	-199,13	Accept		119,22	0,0744	Accept		0,81198	8,8705	Accept
13775	10115	-381,55	Accept		121,67	0,07296	Reject		0,80439	8,8773	Accept
17255	11060	-724,43	Reject		124,96	0,07159	Reject		0,79985	8,9456	Reject
12387	9809,4	-235,13	Accept		119,99	0,07377	Accept		0,80762	8,851	Accept
14427	10738	-483,34	Accept		118,14	0,07549	Reject		0,8201	8,9182	Accept
14313	10379	-447,63	Accept		120,87	0,07359	Accept		0,80866	8,895	Accept
16914	10658	-645,61	Accept		129,01	0,06914	Accept		0,78492	8,9198	Accept
17031	10931	-712,04	Accept		124,48	0,07177	Reject		0,80036	8,9342	Accept
14764,8	10437,49	-474,654	9		122,622	0,072642	5		0,803577	8,9001	9

Table 22: Fitting 4 µm Data to Distribution 7-9

	Lognormal(3P)					Weibull.				Weibull(3P)		
σ	μ	γ	KS test		α	β	KS test		α	β	γ	KS test
0,79347	8,8563	69,493	Accept		1,5829	10203	Reject		1,2061	9405,1	746,33	Accept
0,82753	8,9043	81,38	Accept		1,52	10889	Reject		1,164	9881,9	859,1	Accept
0,86418	8,8066	310,98	Accept		1,5174	10373	Reject		1,1209	9319	890,29	Accept
0,82839	8,8476	150,22	Accept		1,5385	10389	Reject		1,1496	9273,2	921,2	Accept
0,77663	8,9747	-165,56	Reject		1,5553	11081	Reject		1,233	10354	690,41	Accept
0,85	8,7986	253,7	Accept		1,5247	10155	Reject		1,1517	9255,1	795,79	Accept
0,83015	8,906	64,488	Accept		1,5106	10899	Reject		1,2076	10339	581,46	Accept
0,82238	8,8781	87,765	Accept		1,5308	10597	Reject		1,1936	9785,2	739,69	Accept
0,76484	8,9454	-142,9	Accept		1,5836	10727	Reject		1,2798	10356	464,53	Accept
0,77891	8,961	-150,63	Accept		1,5541	10959	Reject		1,2292	10190	715,96	Accept
0,813648	8,88786	55,8936	9		1,54179	10627,2	0		1,19355	9815,85	740,476	10

These distributions ordered from the worst to the best are as follows: Weibull.(0), Gamma.(1), Inv.Gaussian.(1), Log-Gamma(5), Gamma(3P)(8), Lognormal.(9), Lognormal(3P)(9), Inv.Gaussian(3P)(9) and Weibull(3P)(10).

4.3.2.4 Eight Micrometer Two-Dimensional

Table 23: Fitting 8 µm Data to Distributions 1-3

Gamma.				Gamma(3P)					Inv.Gaussian.		
α	β	KS test		α	β	γ	KS test		λ	μ	KS test
1,5215	22576	Reject		1,4194	22237	2786,4	Accept		52261	34349	Reject
1,5728	23351	Reject		1,5929	21840	1937,2	Accept		57765	36727	Reject
1,5067	24268	Reject		1,4159	23940	2668	Accept		55088	36563	Reject
1,493	22916	Reject		1,5685	20371	2262,6	Reject		51083	34215	Accept
1,5872	22093	Reject		1,6209	20349	2081,3	Accept		55654	35065	Reject
1,6103	22439	Reject		1,4794	22586	2719,3	Accept		58183	36132	Reject
1,4288	24378	Reject		1,4586	22135	2546,4	Reject		49769	34832	Accept
1,5356	22785	Reject		1,4584	22309	2451,5	Accept		53726	34987	Reject
1,6157	22745	Reject		1,4168	23728	3133	Accept		59378	36750	Reject
1,4945	23648	Reject		1,3963	23045	3163,9	Accept		52821	35343	Accept
1,53661	23119,9	0		1,48271	22254	2574,96	8		54572,8	35496,3	3

Table 24: Fitting 8 µm Data to Distributions 4-6

Inv.Gaussian(3P)					Log-Gamma				Lognormal.		
λ	μ	γ	KS test		α	β	KS test		σ	μ	KS test
51777	35791	-1441,7	Accept		166,44	0,06098	Accept		0,78625	10,148	Accept
63471	39304	-2577,4	Accept		168,64	0,06061	Reject		0,78667	10,221	Accept
56237	38511	-1948	Accept		163,08	0,06256	Accept		0,79857	10,203	Accept
53281	35441	-1225,9	Accept		176,82	0,05743	Accept		0,76336	10,156	Accept
57722	36744	-1678,7	Accept		176,61	0,05766	Accept		0,76594	10,184	Accept
59872	38243	-2111,2	Accept		170,2	0,05996	Accept		0,78188	10,206	Accept
46268	35374	-541,47	Accept		168,32	0,06035	Accept		0,78252	10,157	Accept
49607	36151	-1164,1	Accept		163,87	0,06201	Accept		0,79342	10,162	Accept
55297	38261	-1510,8	Accept		168,31	0,06072	Accept		0,7873	10,219	Accept
52013	36502	-1159,8	Accept		171	0,05953	Accept		0,77806	10,18	Accept
54554,5	37032,2	-1535,91	10		169,329	0,060181	9		0,782397	10,1836	10

Table 25: Fitting 8 µm Data to Distributions 7-9

Lognormal(3P)					Weibull.				Weibull(3P)			
σ	μ	γ	KS test		α	β	KS test		α	β	γ	KS test
0,80077	10,13	340,69	Accept		1,5735	36738	Reject		1,1985	33614	2819,4	Accept
0,76245	10,252	-635,59	Accept		1,5833	39398	Reject		1,272	37615	1980,8	Accept
0,79789	10,204	-16,754	Accept		1,5517	39004	Reject		1,197	36073	2713	Reject
0,78009	10,134	414,73	Accept		1,6196	36621	Reject		1,2442	34401	2307,6	Reject
0,76925	10,18	84,794	Accept		1,6182	37687	Reject		1,2767	35712	2123,9	Accept
0,7753	10,214	-168,59	Accept		1,5873	38776	Reject		1,2315	35826	2756,9	Accept
0,83242	10,094	1153,8	Accept		1,5706	37087	Reject		1,1988	34451	2570,2	Reject
0,8195	10,129	604,76	Accept		1,5552	37394	Reject		1,2155	34798	2480,6	Reject
0,80798	10,193	514,32	Accept		1,5708	39455	Reject		1,2083	35870	3157,6	Accept
0,80556	10,145	667,24	Accept		1,5876	37782	Reject		1,1831	34144	3206,1	Accept
0,795121	10,1675	295,94	10		1,58178	37994,2	0		1,22256	35250,4	2611,61	6

These distributions ordered from the worst to the best are as follows: Weibull.(0), Gamma.(0), Inv.Gaussian.(3), Weibull(3P)(6), Gamma(3P)(8), Log-Gamma(9), Lognormal.(10), Lognormal(3P)(10) and Inv.Gaussian(3P)(10).

The popular Weibull, Gamma and Inverse Gaussian distributions still give very poor fits. The Log-Gamma, which in the 1D case gave stable values, in this scenario has given widely varying fittings proving unsuitable in the considerations for the 2D realm. The 3P version of the Gamma and Weibull, which did very poorly in the 1D case, were shown to be outstanding in this case as lowest match record in both cases was 8 and the highest for the Gamma(3P) was 9, while for the Weibull(3p), it was 10. General distributions that would both accommodate the 1D as the 2D cases are the Lognormal, Lognormal(3p) and the Inverse Gaussian.

Chapter 5

CONCLUSION

5.1 Summary

Thus far what has been accomplished is the recreation of the one- and two-dimensional molecular channel with and without boundaries. The propagation delays of diffusing particles in both scenarios were analyzed. The considered communication ranges were short range.

In a bid to set the foundations for the development of workload models for the bounded case, an effort was made to fit exhaustively several popular distributions to the delay data generated from simulations. The effort resulted in at least 3 very viable distributions which cut across both the 1D and the 2D cases. These distributions are the Inverse Gaussian (3p), the lognormal, and the Lognormal (3p).

5.2 Future Work

Due to the close relations of the particular behavior in both 1D and 2D, I speculate that the 3D case will follow same pattern although checking for hits to boundaries will be slightly more complex. I also expect that the time complexity to be higher due to three degrees of freedom in which particles can move. The stage that the nanotechnology has been developed thus far makes it deployable in point to point communication networks. Due to this fact, I hope in the nearest future, this will be made a reality especially in the field of drug delivery where medicine can be delivered within a range of time comfortable enough for these nanomachines to communicate effectively within. In addition, studies should be encouraged in making multi-transmitter and multi-receiver type a reality (i.e. nanonetworks) so as to make the application much more widespread.

REFERENCES

[1] I. Akyildiz, F. Brunetti and C. Blázquez, "Nanonetworks: A new communication paradigm," *Computer Networks*, vol. 52, pp. 2260-2279, 2008.

[2] D. Arifler, "Link Layer Modeling of Bio-inspired Communication in Nanonetworks," *Nano Communication Networks*, vol. 2, no. 4, pp. 223-229, 2011.

[3] V. Balzani, A. Credi, S. Silvi and M. Venturi, "Artificial nanomachines based on interlocked molecular species: recent advances," *Chemical Society Reviews*, vol. 35, p. 1135–1149, 2006.

[4] G. Hanson, "Fundamental transmitting properties of carbon nanotube antennas," *IEEE Transactions, Antennas and Propagation*, vol. 53, no. 11, pp. 3426 - 3435, 2005.

[5] J. Kikuchi, A. Ikeda and M. Hashizume, "Biomimetic Materials," in *Encyclopedia of Biomaterials and Biomedical Engineering*, New York, Marcel Dekker, 2004.

[6] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts and P. Walter, Molecular biology of the cell, New York: Garland Science, 2002.

[7] C. Montemagno, "Nanomachines: a roadmap for realizing the vision," *Biomed J Nanopart Res,* vol. 3(1), p. 1–3, 2001.

[8] C. Mavroidis, A. Dubey and M. Yarmush, "Molecular machines," *Annu. Rev. Biomed. Eng,* vol. 6, p. 363–395, 2004.

[9] T. Nakano, M. Moore, A. Enomoto and T. Suda, "Molecular Communication Technology as a Biological ICT," in *Biological Functions for Information and Communication Technologies, Studies in Computational Intelligence,* vol. 320, H. Sawai, Ed., Springer-Verlag Berlin Heidelberg, 2011, pp. 49-86.

[10] M. Gregori, I. Llatser, A. Cabellos-Aparicio and E. Alarcón, "Physical channel characterization for medium-range nanonetworks using flagellated bacteria," *Computer Networks,* vol. 55, p. 779–791, 2011.

[11] S. Hiyama, Y. Moritani, T. Suda, R. Egashira, A. Enomoto, M. Moore and T. Nakano, "Molecular Communication," in *Proceedings of the 2005 NSTI Nanotechnology Conference, poster presentation*, USA, May 2005.

[12] L. P. Giné and I. Akyildiz, "Molecular communication options for long range nanonetworks," *Computer Networks: The International Journal of Computer and Telecommunications Networking,* vol. 53, no. 16, p. 2753–2766, 2009.

[13] Y. Moritani, S. Hiyama and T. Suda, "Molecular communication for health care applications.," in *Proceedings of the Fourth Annual IEEE International*

Conference on Pervasive Computing and Communications Workshops, Washington, DC,, 2006.

[14] A. Adamatzky, B. Costello and T. Asai, Reaction-diffusion computers, Elsevier, 2005.

[15] A. Enomoto, M. Moore, T. Nakano, R. Egashira, T. Suda, A. Kayasuga, H. Kojima, H. Sakakibara and K. Oiwa, "A molecular communication system using a network of cytoskeletal filaments," in *Technical Proceedings of the 2006 NSTI Nanotechnology Conference and Trade,* 2006.

[16] S. Hiyama, Y. Isogawa, T. Suda, Y. Moritani and K. Suto, "A design of an autonomous molecule loading/transporting/unloading system using DNA hybridization and biomolecular linear motors in molecular communication," *European Nano Systems, Grenoble, France,* 2005.

[17] T. Dennis, J. Lee, T. Ozdere, T. Lee and L. You, "Engineering gene circuits: foundations and applications," in *Nanotechnology in Biology and Medicine Methods, Devices and Applications,* e. b. T. Vo-Dinh, Ed., USA, CRC Press, 2007.

[18] S. Basu, Y. Gerchman, C. Collins, F. Arnold and R. Weiss, "A synthetic multicellular system for programmed pattern formation," *Nature,* vol. 434, p. 1130–1134, April 21 2005.

[19] R. Weiss, S. Basu, S. Hooshangi, A. Kalmbach, D. Karig, R. Mehreja and I. Netravali, "Genetic circuit building blocks for cellular computation, communications, and signal processing," *Natural Computing*, vol. 2, no. 1, p. 47–84, 2003.

[20] J. Dueber, E. Mirsky and W. Lim, "Engineering synthetic signaling proteins with ultrasensitive input/output control," *Nat. Biotechnol.,* vol. 25, p. 660–662, 2007.

[21] T. Gardner, C. Cantor and J. Collins, "Construction of a genetic toggle switch in Escherichia coli," *Nature,* vol. 403(6767), p. 339–342, 20 Jan 2000.

[22] M. Elowitz and S. Leibler, "A synthetic oscillatory network of transcriptional regulators," *Nature,* vol. 403(6767), p. 335–338, 20 Jan 2000.

[23] A. Forster and G. Church, "Towards synthesis of a minimal cell," *Mol. Syst. Biol.,* 2006.

[24] T. Schneider, "Theory of molecular machines I. Channel capacity of molecular machines," *J. Theor. Biol.*, vol. 148, p. 83–123, 1991.

[25] G. Hazelbauer, R. Miesibov and J. Adler, "Escherichia coli mutants defective in chemotaxis toward specific chemicals," in *Proceedings of the National Academy of Sciences of the United States of America(PNAS)*, December 1969.

[26] D. Nelson and M. Cox, Lehninger Principles of Biochemistry, fourth ed., W.H. Freeman and Company, 2005.

[27] Y. Gerchman and R. Weiss, "Teaching bacteria a new language," in *Proceedings of the National Academy of Sciences101(8)*, 2004.

[28] R. Weiss and T. Knight, "Engineered communications for microbial robotics. DNA computing.," in *6th International Meeting on DNA Based Computers, DNA*, New York, 2000.

[29] T. Nakano, Y. Hsu, W. Tang, T. Suda, D. Lin, T. Koujin, T. Haraguchi and Y. Hiraoka, "Microplatform for intercellular communication," in *Proceedings of the Third Annual IEEE International Conference on Nano/Micro Engineered and Molecular Systems*, 2008.

[30] T. Nakano, T. Suda, T. Koujin, T. Haraguchi and Y. Hiraoka, "Molecular communication through gap junction channels: system design, experiments and modeling," in *Proceedings of the 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*, Dec. 2007.

[31] M. Moore, A. Enomoto, T. Nakano, R. Egashira, T. Suda, A. Kayasuga, H. Kojima, H. Sakakibara and K. Oiwa, "A design of a molecular communication system for nanomachines using molecular motors," in *Proceedings of the Fourth Annual IEEE Conference on Pervasive Computing and Communications*

Workshops., Washington, DC, 2006.

[32] T. Suda, M. Moore, T. Nakano, R. Egashira and A. Enomoto, "Exploratory research on molecular communication between nanomachines.," in *2005 Genetic and Evolutionary Computation Conference, Late-breaking Papers*, New York, 2005.

[33] H. Hess, C. Matzke, R. Doot, J. Clemmens, G. Bachand, B. Bunker and V. Vogel, "Molecular shuttles operating undercover: a new photolithographic approach for the fabrication of structured surfaces supporting directed motility," *Nano Lett*, vol. 3(12), p. 1651–1655, 2003.

[34] G. Patterson, R. Day and D. Piston, "Fluorescent protein spectra," *Journal of Cell Science,* 2001.

[35] J. Yu, J. Wang, S. Lou, T. Wang and Y. Jiang, "Small molecular and polymer organic light-emitting diodes based on novel iridium complex phosphor," *Displays,* vol. 29, no. 5, p. 493–496, 2008.

[36] L. Parcerisa and I. F. Akyildiz, "Molecular communication options for long range nanonetworks," *Computer Networks,* vol. 53, no. 16, p. 2753–2766, 2009.

[37] I. Llatser, E. Alarcon and M. Pierobony, "Diffusion-based channel characterization in molecular nanonetworks," in *IEEE Conference on Computer*

Communications Workshops, 2011.

[38] W. H. Bossert and E. O. Wilson, "The analysis of olfactory communication among animals.," *Journal of theoretical biology*, vol. 5, no. 3, p. 443–69, 1963.

APPENDICES

Appendix A: Distributions

The text below is reproduced directly from the Help File of EasyFit Software for easy reference.

Gamma Distribution

Parameters

α - continuous shape parameter ($\alpha > 0$)
β - continuous scale parameter ($\beta > 0$)
γ - continuous location parameter ($\gamma \equiv 0$ yields the two-parameter Gamma distribution)

Domain

$\gamma \leq x < +\infty$

Three-Parameter Gamma Distribution

Probability Density Function

$$f(x) = \frac{(x-\gamma)^{\alpha-1}}{\beta^{\alpha}\Gamma(\alpha)} \exp(-(x-\gamma)/\beta)$$

Cumulative Distribution Function

$$F(x) = \frac{\Gamma_{(x-\gamma)/\beta}(\alpha)}{\Gamma(\alpha)}$$

Two-Parameter Gamma Distribution

Probability Density Function

$$f(x) = \frac{x^{\alpha-1}}{\beta^{\alpha}\Gamma(\alpha)} \exp(-x/\beta)$$

Cumulative Distribution Function

$$F(x) = \frac{\Gamma_{x/\beta}(\alpha)}{\Gamma(\alpha)}$$

where Γ is the Gamma Function, and Γ_z is the Incomplete Gamma Function.

Weibull Distribution

Parameters

α - continuous shape parameter ($\alpha > 0$)
β - continuous scale parameter ($\beta > 0$)
γ - continuous location parameter ($\gamma \equiv 0$ yields the two-parameter Weibull distribution)

Domain

$\gamma \leq x < +\infty$

Three-Parameter Weibull Distribution

Probability Density Function

$$f(x) = \frac{\alpha}{\beta}\left(\frac{x-\gamma}{\beta}\right)^{\alpha-1} \exp\left(-\left(\frac{x-\gamma}{\beta}\right)^{\alpha}\right)$$

Cumulative Distribution Function

$$F(x) = 1 - \exp\left(-\left(\frac{x-\gamma}{\beta}\right)^{\alpha}\right)$$

Two-Parameter Weibull Distribution

Probability Density Function

$$f(x) = \frac{\alpha}{\beta}\left(\frac{x}{\beta}\right)^{\alpha-1} \exp\left(-\left(\frac{x}{\beta}\right)^{\alpha}\right)$$

Cumulative Distribution Function

$$F(x) = 1 - \exp\left(-\left(\frac{x}{\beta}\right)^{\alpha}\right)$$

Lognormal Distribution

Parameters

σ - continuous parameter ($\sigma > 0$)
μ - continuous parameter
γ - continuous location parameter ($\gamma \equiv 0$ yields the two-parameter Lognormal distribution)

Domain

$\gamma < x < +\infty$

Three-Parameter Lognormal Distribution

Probability Density Function

$$f(x) = \frac{\exp\left(-\frac{1}{2}\left(\frac{\ln(x-\gamma)-\mu}{\sigma}\right)^2\right)}{(x-\gamma)\,\sigma\sqrt{2\,\pi}}$$

Cumulative Distribution Function

$$F(x) = \Phi\left(\frac{\ln(x-\gamma)-\mu}{\sigma}\right)$$

Two-Parameter Lognormal Distribution

Probability Density Function

$$f(x) = \frac{\exp\left(-\frac{1}{2}\left(\frac{\ln x-\mu}{\sigma}\right)^2\right)}{x\,\sigma\sqrt{2\,\pi}}$$

Cumulative Distribution Function

$$F(x) = \Phi\left(\frac{\ln x-\mu}{\sigma}\right)$$

where Φ is the Laplace Integral.

Inverse Gaussian Distribution

Parameters

λ - continuous parameter ($\lambda > 0$)
μ - continuous parameter ($\mu > 0$)
γ - continuous location parameter ($\gamma \equiv 0$ yields the two-parameter Inverse Gaussian distribution)

Domain

$\gamma < x < +\infty$

Three-Parameter Inverse Gaussian Distribution

Probability Density Function

$$f(x) = \sqrt{\frac{\lambda}{2\,\pi\,(x-\gamma)^3}}\,\exp\left(-\frac{\lambda\,(x-\gamma-\mu)^2}{2\,\mu^2\,(x-\gamma)}\right)$$

Cumulative Distribution Function

$$F(x) = \Phi\left(\sqrt{\frac{\lambda}{x-\gamma}}\left(\frac{x-\gamma}{\mu}-1\right)\right) +$$
$$+\Phi\left(-\sqrt{\frac{\lambda}{x-\gamma}}\left(\frac{x-\gamma}{\mu}+1\right)\right)\exp(2\,\lambda/\mu)$$

Two-Parameter Inverse Gaussian Distribution

Probability Density Function

$$f(x) = \sqrt{\frac{\lambda}{2\,\pi\,x^3}}\,\exp\left(-\frac{\lambda\,(x-\mu)^2}{2\,\mu^2\,x}\right)$$

Cumulative Distribution Function

$$F(x) = \Phi\left(\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu}-1\right)\right) + \Phi\left(-\sqrt{\frac{\lambda}{x}}\left(\frac{x}{\mu}+1\right)\right)\exp(2\,\lambda/\mu)$$

where Φ is the Laplace Integral.

Log-Gamma Distribution

Parameters

α - continuous parameter ($\alpha > 0$)
β - continuous parameter ($\beta > 0$)

Domain

$0 < x < +\infty$

Probability Density Function

$$f(x) = \frac{(\ln(x))^{\alpha-1}}{x\beta^{\alpha}\Gamma(\alpha)}\exp(-\ln(x)/\beta)$$

Cumulative Distribution Function

$$F(x) = \frac{\Gamma_{\ln(x)/\beta}(\alpha)}{\Gamma(\alpha)}$$

Appendix B: KS (Kolmogorov-Smirnov) Test

The text below is mainly based on the Help File of EasyFit Software.

Kolmogorov-Smirnov Test

The KS test is used to determine if a sample comes from a hypothesized continuous distribution. Assume that a random sample X_1, ... , X_n from some distribution with CDF *F(X)* is given. The empirical CDF is denoted by

$$F_n(x) = \frac{1}{n} \cdot \left[\text{Number of observations} \le x\right]$$

Definition

The Kolmogorov-Smirnov statistic (D) is based on “the largest vertical difference between the theoretical and the empirical cumulative distribution function”:

$$D = \max_{1 \le i \le n}\left(F(x_i) - \frac{i-1}{n}, \frac{i}{n} - F(x_i)\right)$$

Hypothesis Testing

The null and the alternative hypotheses are:

H_0: the data follow the specified distribution;

H_A: the data do not follow the specified distribution.

The null hypothesis is rejected at the given significance level (α) if the test statistic, *D*, is greater than the critical value obtained from a table. The fixed values of α that are generally used to evaluate the null hypothesis (H_0) at various significance levels are 0.01, 0.05 etc. For most applications, a typical value used is 0.05.

Appendix C: Programs

Abouts Files

```
// nice
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
// The purpose of this function is to o
char * aboutFiles(char *genericName )
{
      FILE* fp;// file pointer used to point to the filename
generated
      char end[]=".txt";// this as the name implies is to be at the
tail of whatever name is generated, this gives the .txt extension to
the file
      int endSize = sizeof(end);// this gives the character count of
the file's extension, this is used inturn to give the totalsize of
the characters of the filename when combined with the part that came
before it
      int gnS = strlen(genericName);// this gets the file size of
the original file name that was passed into the program. This
constitues the first part of the filename, and this integer combined
with endsize would give the size of the generic file, but where
another other than the generic file is to be created the middle part
which consist of numbers in form of ordinary letters will be added
to the mix.
      int n = gnS+endSize;// this gives total size of the generic
file name, which is to be created
      char *fileName = (char*) malloc(n* sizeof(char));// this
creates the generic file name holder character
      strcpy(fileName,genericName);// this copies the generic
filename in to the foremost position
      strcat(fileName, end); // this copies in the extension to the
tail most point of the newly created container
      int i = 1, // numeral for te file to be generate
            j= 1, // this is the number to indicate the number of
spaces the generate integer will need in the newly named file
            divide = i , // divide is the helps j determine the
number of spaces it will need by disintegrated by the dividing 10
till it reaches 0 and at each loop j is increased by 1
            fStatus=1 ;// flag that determines whether the following
loop would go on repeating itself, in this repeation newer tests and
names are made possible, this will only change to zero when an
unused name is found, and hence signifying the end of the search.
      do
      {
            fp = NULL;// the file pointer is initial made to point
to no object, as a point of precaution, so if it had being used
before it was now free
            fp = fopen(fileName, "r");// this function attempts to
open a file given by the string pointed to by the fileName pointer,
if exists it will return an object, not a null, hence indicating
that the file already exists, hence the program needs to generate a
new name. If it does not exist however the pointer returned points
to null, hence the pointer is safe to return to the calling function
the pointer to that string of characters discovered by it
            if(fp != NULL)// tests whether the file name exists or
not, if it exists, its body is executed
              {
                  fclose(fp);// this releases the pointer from the
previous file it was pointing to
                  free(fileName);// this frees the character pointer
object from the string it was previously pointing to
                  do{j++;}while(divide /= 10);
                  j++;// numbers converted to characters will take
up as much characters as the digit positions the occupy, so the 1-9
will occupy just one space, 10-99 will occupy 2 and 100-999 3 etc.
as 'divide' is an integer and 10 is an integer, their division
leaves no decimal part. So if 10 is divide by a number in the 1-9
range it gives 0, in the 10-99 zone: 1, 100-999: 3 etc. Since we
employ a do while loop here it gives us one extra in each instance.
So for d above stated category we have 1, 2, 3 etc. the initial
value of j in each instance is 1 so the each digit has and extra
character added to its string. The reason for this is that in the
copying functions, they require an extra space to put in the '\0' at
the end, if they don't have that space to put it they either
truncate the character by that one space and put the null character
in that place or the null character might be neglected completely.
So the hence the starting value of j on each run as 1 rather than
zero
                  char *a;// name holder/ potential pointer to the
string of integer character soon to b created for the corresponding
generated number i
                  a = (char*) malloc(j* sizeof(char));// this
creates to location of space the string is going to point to
                  sprintf(a,"_%d", i);// this coverts the integer i
into its corresponding character letter, this will be pointed to by
a
                  n = gnS +endSize+ j;// getting the size of the
completely new string
                  fileName = (char*) malloc(n* sizeof(char));//
making a pointer to this new string/name
                  strcpy(fileName,genericName);// copying the
generic part of this newfile name
                  strcat(fileName, a);// adjoining the number that
makes the file unique to the generic part
                  strcat(fileName, end);// attaching the extention
part to this file name
                  i++;// this increase the file name number just in
case this last created file might not be found to be unique
                  j= 1;//resetting the spaces needed for the number
portion of the file name to be one
                  divide = i;// this gives the number of the
generated file number to the variable that will help the program
determine the number of spaces the number part next file name
generated will need
                  free(a);// this frees the pointer pointing to the
character holding the number part of the file name
              }
            else// this is executed when the string generated is
found not to exist in the specified directory, making it alright to
return the string in question to the calling function. This makes
the condition "fStatus to fail"
            {
                  fStatus = 0;// loop control is now set to exit
            }
      }while(fStatus);// controls the string generation process
      return fileName;// the valid file name is returned at this
point
}
```

About files General File

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

char * aboutFilesGeneralVD(char *genericName, int* T1, int* T2, int*
T3, int* T4)
{
      FILE* fp;
      //char genericName[]= "freedom";
      //int genericNameSize = sizeof(genericName);
      //char *fileName;
      char end[]="Overall.txt";
      int endSize = sizeof(end);
      int gnS = strlen(genericName);
      int n = gnS+endSize;
      char Tee1[]="FileName",
            Tee2[]="Time",
            Tee3[]="TotalParticles",
            Tee4[]="Timestep";

      *T1= strlen(Tee1)+6;
      *T2= strlen(Tee2)+3;
      *T3= strlen(Tee3);
      *T4= strlen(Tee4);
      //free(fileName);
      char *fileName = (char*) malloc(n* sizeof(char));
      strcpy(fileName,genericName);
      strcat(fileName, end);
      // generating file names
      fp = NULL;
      fp = fopen(fileName, "r");
      if(fp != NULL)
        {
            fclose(fp);
        }
      else
      {
            fp = fopen(fileName, "w+");
            fprintf(fp, "%*s, %*s, %s, %s, %s, %s\n\n",  *T1, Tee1,
*T2, Tee2, Tee3,  Tee4);
            fclose(fp);
      }
      return fileName;
}

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

char * aboutFilesGeneral1D(char *genericName, int* T1, int* T2, int*
T3, int* T4, int* T5, int* T6, int* T7, int* T8, int* T9)
{
      FILE* fp;
      //char genericName[]= "freedom";
      //int genericNameSize = sizeof(genericName);
      //char *fileName;
      char end[]="Overall.txt";// this is the text of the end part
to the string
      int endSize = sizeof(end);// this is the integer counts to the
words fo the string pointed to by end
```

```
      int gnS = strlen(genericName);// this is the length of the
sting pointed to by genericName
      int n = gnS+endSize;// this the total size of the of the
string of the overall file name
      // the next following lines are arrays containing the names
that will be used for formanting the overal file
      char Tee1[]="FileName",// isolated run name file
            Tee2[]="Distance_µm",// the constant distance set for
the this file
            Tee3[]="Time_µs",// time to live
            Tee4[]="TotalParticles",// Total number of particles
considered
            Tee5[]="Timestep_µs",//Time per each step of particle
            Tee6[]="Particles_Lost",// Particle number that made it
to the destination
            Tee7[]="Arrival_Fraction",// fraction of particles that
made the destination
            Tee8[]="Tot_Particle_Transit_Time",// Total transmission
time
            Tee9[]="Average_Arrival_Time";// Average arrival time
      // retriving by interger the sizes of the area for the
preceeding headers
      *T1= strlen(Tee1)+6;
      *T2= strlen(Tee2);
      *T3= strlen(Tee3)+3;
      *T4= strlen(Tee4);
      *T5= strlen(Tee5);
      *T6= strlen(Tee6);
      *T7= strlen(Tee7);
      *T8= strlen(Tee8);
      *T9= strlen(Tee9)+ 4;
      //free(fileName);
      char *fileName = (char*) malloc(n* sizeof(char));// creating
the pointer for the file name
      strcpy(fileName,genericName);// copying in the initial part of
the file name
      strcat(fileName, end);// attaching the end part
      // testing for its existence
      fp = NULL;// points no where initially
      fp = fopen(fileName, "r");// points somewhere if it exists,
points no where if it doesnt
      if(fp != NULL)
        {
            fclose(fp);// close the file if it exists
        }
      else
      {
            // if it doesnt exist create it, set up the heading
formatting and then close the file
            fp = fopen(fileName, "w+");// open for writing
            fprintf(fp, "%*s, %s, %*s, %s, %s, %s, %s, %s, %*s\n\n",
*T1, Tee1,  Tee2,  *T3, Tee3,  Tee4,  Tee5,  Tee6, Tee7,
                  Tee8, *T9, Tee9);//  Formating and putting the
names. The star gives room for the integer to create the
correspoding number of space and the strings fill in from right to
left
            fclose(fp);// file closed
      }
      return fileName;// generated string name is returned
}
#include<stdio.h>
```

```
#include<string.h>
#include<stdlib.h>

char * aboutFilesGeneral2D(char *genericName, int* T1, int* T2, int*
T3, int* T4, int* T5, int* T6, int* T7, int* T8, int* T9)
{
      FILE* fp;
      char end[]="Overall.txt";
      int endSize = sizeof(end);
      int gnS = strlen(genericName);
      int n = gnS+endSize;
      char Tee1[]="FileName",
            Tee2[]="Xcod",
            Tee3[]="Ycod",
            Tee4[]="Radius",
            Tee5[]="TTL",
            Tee6[]="Average_Arrival_Time",
            Tee7[]="DelT",
            Tee8[]="SucParts",
            Tee9[]="AverageSucParts";

      *T1= strlen(Tee1)+6;
      *T2= strlen(Tee2);
      *T3= strlen(Tee3);
      *T4= strlen(Tee4);
      *T5= strlen(Tee5)+3;
      *T6= strlen(Tee6);
      *T7= strlen(Tee7);
      *T8= strlen(Tee8);
      *T9= strlen(Tee9)+ 4;
      char *fileName = (char*) malloc(n* sizeof(char));
      strcpy(fileName,genericName);
      strcat(fileName, end);
      fp = NULL;
      fp = fopen(fileName, "r");
      if(fp != NULL)
        {
            fclose(fp);
        }
      else
      {
            fp = fopen(fileName, "w+");
            fprintf(fp, "%*s, %s, %s, %s, %*s, %s, %s, %s, %*s\n\n",
*T1, Tee1,  Tee2, Tee3,  Tee4,  *T5,  Tee5,  Tee6, Tee7,
                  Tee8, *T9, Tee9);
            fclose(fp);
      }
      return fileName;
}

#include<stdio.h>

void ArrivalReport(int PN, FILE *fp, int AT,
                              int formatL, int formatR)
{
      /*fprintf(fp, "%-*d||%*d\n", formatL,
            PN,  formatR, AT);*/
      fprintf(fp, "%d\n", AT);
}

#include<stdio.h>
```

```
void ArrivalReportDouble(int PN, FILE *fp, double AT, int formatL,
int formatR)
{
      fprintf(fp, "%-*d||%*.5f\n", formatL,
            PN,  formatR, AT);
}
#include<stdio.h>
#include<io.h>
#include<stdlib.h>
#include<time.h>
#include<direct.h>
#include<string.h>
#include"threein1.h"

void constTimeVaryingDist()
{
      int partTransitTime , // ttl
            totalParticles ,
            deltaTime;// time steps
      char textFileName[] = "constTimeVaryingDist.txt";
      // check if file exist
      if((_access(textFileName ,0)))
      {
            printf("no job for the function %s\n", textFileName);
            return;
      }//if
      time_t rawtime;
        struct tm * timeinfo;
        char buffer [40];
        time (&rawtime);
        timeinfo = localtime (&rawtime);
        strftime (buffer,40,"ODC\\%a_%Y-%m-%d_%I_%M",timeinfo);
        char prefix[]="md ";
        int i = strlen(prefix)+ strlen(buffer);
        char * combo = (char *) malloc (i * sizeof (char));
        strcpy(combo,prefix);
      strcat(combo, buffer);
      system(combo);
      char prefix2[]="move constTimeVaryingDist.txt ";
      int i2 = strlen(prefix2)+ strlen(buffer);
      char * combo2 = (char *) malloc (i2 * sizeof (char));
      strcpy(combo2,prefix2);
      strcat(combo2, buffer);
      system(combo2);
      if (chdir (buffer) == -1)
      {
        printf ("chdir failed - %s\n", strerror (errno));
            return;
    }
      FILE *fp = fopen(textFileName, "r");
      // getting rid of labels
      char getRid = 'q';
      while(getRid != '\n')
            fscanf(fp,"%c", &getRid);
      int times = 0;

      while(fscanf(fp,"%d%d%d", &partTransitTime, &totalParticles,
&deltaTime) != EOF)
      {
            times++;
            particleTrajectory(partTransitTime, totalParticles, 
deltaTime);// particle journey
      }
      fclose(fp);
      if(!times)
            printf("nothing in the file\n");
      //system("rename constTimeVaryingDist.txt 
constTimeVaryingDistOld.txt");

}
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
int fracNumCount(double dobNum)
{
      int negPow = 8;
      int dobNumInt = dobNum * 100000000;
      while(dobNumInt%10==0&& negPow!=0)
      {
            negPow--;
            dobNumInt/=10;
      }
      return negPow;
}


/*int main()
{
      double testDob = 0.000431;
      printf("%d\n",fracNumCount(testDob));
      system("pause");
}*/
#include<stdlib.h>
#include<stdio.h>

void destDistAndBoundSettings(double *dD, double *lB, double *rB, 
int *tTLL, int *tTL, int *tPN, int *dT)
{
      double DestDistance = *dD;// distance of destination from 
source
      int ttlLoss = *tTLL,
            ttl = *tTL,
            totParNum = *tPN,
            delTime = *dT;
      double leftBound = *lB,
            rightBound = *rB;
      do{
            printf("\n\n\n");
            printf("Current Values are as follows\n");
            printf("Destination Distance: %f\n", *dD);
            printf("total particle number: %d\n", *tPN);
            printf("time per step: %d\n", *dT);
            if(*tTLL)
            {
                  printf("Loss determined by time steps, with ttl 
%d\n", *tTL);
            }//if
            else
            {
                  printf("Loss determined by dimensions\n");
                  printf("Left Bound: %f\n", *lB);
                  printf("Right Bound: %f\n", *rB);
            }
            system("pause");
            int select = 0, below = 0, above = 4;

            do
            {
                  printf("\n\n");
                  printf("type \n");
                  printf("{0} to proceed with the program\n");
                  printf("{1} to change the Destination
Distance\n");
                  printf("{2} to change the Total Particle Size\n");
                  printf("{3} to change the time per step\n");
                  printf("{4} to make changes regarding Loss
determination\n");
                  printf("value: ");
                  scanf("%d", &select);
                  if(select > above || select < below)
                        printf("\nOut of range please try again\n");
                  else if(select == 0)
                        return;
                  else if(select == 1)
                  {
                        printf("\n\n");
                        printf("Put in the new Destination Distance:
");
                        scanf("%lf", dD);
                        printf("The new distance of %.0f is now
set\n", *dD);
                  }
                  else if(select == 2)
                  {
                        printf("\n\n");
                        printf("Put in the new Total Particle Size:
");
                        scanf("%d", tPN);
                        printf("The new distance of %d is now
set\n", *tPN);
                  }
                  else if(select == 3)
                  {
                        printf("\n\n");
                        printf("Put in the new time per step: ");
                        scanf("%d", dT);
                        printf("The new distance of %d is now
set\n", *dT);
                  }
                  else if(select == 4)
                  {
                        printf("\n\n");
                        printf("type \n");
                        printf("{0} to to change the loss
determination type(time or dimension).\n");
                        printf("{1} to change the loss parameter
value(s)\n");
                        printf("Value: ");
                        int select4;
                        scanf("%d", &select4);
                        if(select4 == 0)
                        {
                              if(*tTLL)
                              {
                                    *tTLL = 0;
                                    printf("\n\n");
                                    printf("Loss will now determined
by dimensions\n");
                                    printf("Left Bound: %.0f\n",
*lB);
                                    printf("Right Bound: %.0f\n",
*rB);
                                    printf("to change this type 1, to
allow it type 0\n");
                                    printf("Value: ");
                                    scanf("%d", &select4);
                              }
                              else if(!(*tTLL))
                              {
                                    printf("\n\n");
                                    *tTLL = 1;
                                    printf("Loss will now determined
by time\n");
                                    printf("TTL: %d\n", *tTL);
                                    printf("to change this type 1, to
allow it type 0\n");
                                    printf("Value: ");
                                    scanf("%d", &select4);
                              }

                        }
                        if((*tTLL)&&select4 == 1)
                        {
                              printf("\n\n");
                              printf("Put in the new TTL value: ");
                              scanf("%d", tTL);
                              printf("The new TTL value of %d is now
set:\n",*tTL);
                        }
                        if(!(*tTLL)&&select4 == 1)
                        {
                              printf("\n\n");
                              printf("Put in the new Left Bound
value:\n");
                              scanf("%lf", lB);
                              printf("The new Left Bound value of
%.0f is now set:\n",*lB);
                              printf("Put in the new Right Bound
value:\n");
                              scanf("%lf", rB);
                              printf("The new Right Bound value of
%.0f is now set:\n",*rB);
                        }
                  }
            }while(select > above || select < below);
      }while(1);
}//function end
#include<stdio.h>
#include<stdlib.h>
void getPreferedSettings(double *distanceCompare, int
*partTransitTime, int *totalParticles, int *deltaTime)
{
      do{
            printf("\n\n\n");
            printf("Current Values are as follows\n");
            printf("Comparison Distance: %f\n", *distanceCompare);
            printf("total particle number: %d\n", *totalParticles);
            printf("time per step: %d\n", *deltaTime);
            printf("Particle transit time: %d\n", *partTransitTime);
            system("pause");

            int select = 0, below = 0, above = 4;

            do
            {
                  printf("\n\n");
                  printf("type \n");
                  printf("{0} to proceed with the program\n");
                  printf("{1} to change the Comparison Distance\n");
                  printf("{2} to change the Total Particle Size\n");
                  printf("{3} to change the time per step\n");
                  printf("{4} to change the Particle Transmit
Time\n");
                  printf("value: ");
                  scanf("%d", &select);
                  if(select > above || select < below)
                        printf("\nOut of range please try again\n");
                  else if(select == 0)
                        return;
                  else if(select == 1)
                  {
                        printf("\n");
                        printf("Put in the new Comparison Distance:
");
                        scanf("%lf", distanceCompare);
                        printf("The new distance of %.0f is now
set\n",
                              *distanceCompare);
                  }
                  else if(select == 2)
                  {
                        printf("Put in the new Total Particle Size:
");
                        scanf("%d", totalParticles);
                        printf("The new Total Particle Size of %d is
now set\n",
                              *totalParticles);
                  }
                  else if(select == 3)
                  {
                        printf("\n");
                        printf("Put in the new time per step: ");
                        scanf("%d", deltaTime);
                        printf("The new time per step of %d is now
set\n",
                              *deltaTime);
                  }
                  else if(select == 4)
                  {
                        printf("\n");
                        printf("Put in the Particle Transmit Time:
");
                        scanf("%d", partTransitTime);
                        printf("The new Particle Transmit Time of %d
is now set\n",
                              *partTransitTime);
                  }
            }while(select > above || select < below);
      }while(1);
}//getPreferedSettings
#include<stdio.h>
#include<string.h>

void HeadingAndFormating(FILE *fp, int* Left, int *Right, char*
LeftString, char* RightString)
{
      int bound = 6;// this gives more space in the file passed in
for writing
      *Left = bound + strlen(LeftString);// this gives the formating
integer for the left column
      *Right = bound + strlen(RightString);// this gives the
formating integer for the right column
      /*fprintf(fp, "%-*s||%*s\n", *Left, LeftString,
            *Right, RightString);*/
      fprintf(fp, "%s\n",RightString);// this puts on the column
titles, it can be notice that the left side is left aligned and the
right is right aligned

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

void headingFormating3(FILE *fp, int* Left, int *Center,int *Right,
char* LeftString, char* CenterString, char* RightString)
{
      int bound = 6;
      *Left = bound + strlen(LeftString);
      *Right = bound + strlen(RightString);
      *Center = bound + strlen(CenterString);
      fprintf(fp, "%-*s||%*s||%-*s\n", *Left, LeftString,
            *Right, RightString, *Center, CenterString);

}
#include<stdio.h>
#include<stdlib.h>

void LostReport(int PN, FILE *fp, int formatL, int formatR)
{
      fprintf(fp, "%-*d||%*s\n", formatL, PN,
            formatR,"LOST");// reports formating, number particle
left and "LOST" to the right
}
#include<stdio.h>
#include<stdlib.h>

int lostStatusDimensions(double leftBound, double rightBound, double
distance)
{
      if(distance >= leftBound && distance <= rightBound)
            return 0;
      else
            return 1;
}
```

```
int lostStatusTime(int tTL, int time)
{
      if(time <= tTL)
            return 0;
      else
            return 1;
}

int destinationBreached(double destinationDistance, double distance)
{
      if(destinationDistance >= 0)
            if(distance >= destinationDistance)
                  return 0;
            else
                  return 1;
      else
            if(distance <= destinationDistance)
                  return 0;
            else
                  return 1;
}
#include <math.h>
#include<stdio.h>
#include<stdlib.h>
#define PI 3.141592654

/*This function generates the random walk of the particle under the
influence of diffusion
math lib need by the sqrt, and the RAND_MAX constant. the values Z
is within the range -2.9 to 2.9
phase makes it possible for the values of U and V to be reused
again, at least once*/

double gaussrand()
{

      int negOr = (rand() / (RAND_MAX + 1.0) * (2 - 0) + 0);
      if(!negOr)
            negOr=-1;
      return sqrt(2 * .001)*negOr;
}
void gaussrand2d(double *deltaX, double *deltaY)
{
      double consT = sqrt(4 * .001);
      double pisConst = (rand() / (RAND_MAX + 1.0) * (2.000001 - 0)
+ 0);
      double thetha = pisConst * PI;
      double cosThetha = cos ( thetha);
      double sinThetha = sin ( thetha);
      *deltaX = consT * cosThetha;;
      *deltaY = consT  * sinThetha;
}
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<io.h>
#include<time.h>
#include<direct.h>
#include"threein1.h"
//I am confused about the purpose of this experiment, is it suppose
to account of the time it takes to get to a particular destination,
or the time to get to a particular distance from the source either
way. That means when we say 50nm as the distance from the source in
a one dimensional plane do we mean a just at 50nm or we mean -50nm
and 50nm. If a specific distance is the case then the calculation of
negative destinations will be taken into consideration. Because this
program so far has considered only positive destinations such that
the destination is gotten to if the particle is >= destination
distance. But if the destination be negative then the distance can
only be reached if the particle is <= destination distance. So the
destinations polarity must be determined ever before the testing
beginnings to makes sure of what testing parameters should be used.
But if the destination just means a specific distance away from the
source regardless of the polarity, then just the input distance is
going to always be positive, so the distance in measuring for the
negative symetric half is to make sure that the distance is less
than -ve of the distance, for it to have arrived at that distance.
But on consideration of the specific distance investigation, it is
much more proper that  specific distances and not symettric
distances be considered, so this program at relevant points will be
modified to reflect this change in reasoning, so two if statements
is solicited for, one for when the distance is positive, and another
when negative

void OneDimProper()
{
      double DestDistance;// distance of under investigation
      int ttl, iterations;// time to live
      int TotalParticleNumber; // total number of particles to be
investigated

      int deltaTime;// time step
      char textFileName[] = "OneDimProper.txt";// this is the name
of the file from which all the data for the program is to be gotten,
it must exist if not this part of the experiment will not run. And
the data for each run must be on each row, and the number of
simulation scenario depends on how many rows there are. If the file
exists and no dat is in it, the experiment still doesnt run...
      // check if file exist
      if((_access(textFileName ,0)))//this checks whether the file
exists if it exist it give 0, otherwise 1
      {
            printf("no job for the function %s\n", textFileName);//
prints the fact d file exists not
            return;// returns to the calling function without doing
anywork
      }//if
      time_t rawtime;
        struct tm * timeinfo;
        char buffer [40];
        time (&rawtime);
        timeinfo = localtime (&rawtime);
        strftime (buffer,40,"OD\\%a_%Y-%m-%d\\%I_%M_%p",timeinfo);
        char prefix[]="md ";
        int bufferNum = strlen(buffer), prefixNum = strlen(prefix),i
=bufferNum + prefixNum;
        char * combo = (char *) malloc (i * sizeof (char));
        strcpy(combo,prefix);
      strcat(combo, buffer);
      system(combo);
      char prefix2[]="move ";
      char moveEnd[]= " ";
      int textFileNameNum =strlen(textFileName);
      int moveEndNum = strlen(moveEnd);
      int prefix2Num = strlen(prefix2);
      int combo2Num = prefix2Num + textFileNameNum + moveEndNum +
bufferNum;
      char * combo2 = (char *) malloc (combo2Num * sizeof (char));
      strcpy(combo2,prefix2);
      strcat(combo2, textFileName);
      strcat(combo2, moveEnd);
      strcat(combo2, buffer);
      system(combo2);
      if (chdir (buffer) == -1)
      {
        printf ("chdir failed - %s\n", strerror (errno));
            return;
    }
      //system("pause");
      FILE *fp = fopen(textFileName, "r");// this opens the file in
question for reading
      // getting rid of labels
      char getRid = 'q';// this part helps get rid of the labels in
the text file. the character will read each character and do nothing
with it till it reads the new character then it stops
      while(getRid != '\n')// runs until getRid has a value of '\n'
            fscanf(fp,"%c", &getRid);// reads just one character
      int times = 0;
      while(fscanf(fp,"%lf%d%d%d%d", &DestDistance, &ttl,
&TotalParticleNumber, &deltaTime,&iterations) != EOF)
      {
            times++;
            int counter = 1;
            while(counter++ <= iterations)
            oneSimulation(DestDistance, ttl, TotalParticleNumber,
deltaTime, iterations);//uses the data to facilitate the simulation
      }
      printf("containing folder is %s\n", buffer);
      fclose(fp);// file close when finished
      if(!times)// execute when no data in file
            printf("nothing in the file\n");
      //system("rename OneDimProper.txt OneDimProperOld.txt");//
rename to avoid reuse of same old files on another run.

}// one dim proper
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<time.h>
#include<direct.h>
#include"threein1.h"
#include<io.h>

void oneSimulation(double DestDistance, int ttl, int
TotalParticleNumber, int deltaTime, int iterations)
{
      int Fstcount=0,Sndcount=0,Trdcount=0;
      int g = (int)DestDistance;
    do{Fstcount++;}while(g/=10);
    Fstcount++;
    g = ttl;
    do{Sndcount++;}while(g/=10);
    Sndcount++;
    g = TotalParticleNumber;
    do{Trdcount++;}while(g/=10);
    Trdcount++;
    char *CategoryName = (char*) malloc
((Fstcount+Sndcount+Trdcount)*sizeof(char));
      sprintf(CategoryName,"%d_%d_%d",(int)DestDistance,ttl,TotalPar
ticleNumber);
      char dirStarting[]="OD";
      char dirSlash[]="/";
      int dirStartingNum = strlen(dirStarting),
dirSlashNum=strlen(dirSlash),
            dirEndNum= strlen(CategoryName),
            dirComWithOutNum = dirStartingNum + dirEndNum,
            dirComWithNum = dirStartingNum + dirEndNum +
dirSlashNum;
      if(iterations == 1)
      {
            dirComWithOutNum = dirStartingNum;
            dirComWithNum = dirStartingNum + dirSlashNum;
      }
      char *dirComWithOut = (char*) malloc
((dirComWithOutNum)*sizeof(char));
      strcpy(dirComWithOut,dirStarting);
      if(iterations > 1)
      {
            strcat(dirComWithOut,CategoryName);
      }
      char *dirComWith = (char*) malloc
((dirComWithNum)*sizeof(char));
      strcpy(dirComWith,dirStarting);
      if(iterations > 1)
      {
            strcat(dirComWith,CategoryName);
      }
      //strcat(dirComWith,CategoryName);
      strcat(dirComWith,dirSlash);

      /*destDistAndBoundSettings(&DestDistance, &leftBound,
&rightBound
            ,&ttlLoss, &ttl, &TotalParticleNumber, &deltaTime);*/
      char genericName[]= "Dim1v";// here the name of the directory,
and generic name of all the files produced
      int track = 0;    // for number of characters of directory
      /*while(genericName[++track]!= '/');// counting characters
before directory indicator
      // create directory array
      char *direct = (char*) malloc (track * sizeof(char));//
creating name holder for directory
      // store directory name
      strcpy(direct, genericName);//copying in the directory name
and the slash following
      direct[track] = '\0';// putting the null character in place of
the '/'
      // check whether the directory exists not 0 if it exists, 1 if
not
      if(_access(direct,0))
      {// creating directory if it doesn't already exist
            char prefix[] = "md ";// start point of the creation of
directory folder
            int total = track + strlen(prefix); // total size of the
characters of the create diretory command
```

```
            char *combo = (char*) malloc (total *
sizeof(char));//creation of a create command character holder
            strcpy(combo, prefix);// copying in the prefix of the
comand
            strcat(combo, direct);// attaching the end to the
command characters
            system(combo);// executing the create command finally
      }//if*/
      char *fileName, // filename holder the current simulation
            *fileNameGen;// filename holder for the log textfile for
one dimensional case, storing all the d summarys in each simulation
in one file, to help comparison
      FILE* fp, //file pointer to specific sim log file
            *fp1; // " " " general
      int T1, // this r ints will help in formating d outputs in d
general file
            T2,
            T3,
            T4,
            T5,
            T6,
            T7,
            T8,
            T9;
      /*char prefix[]="OD/";
      int prefixNum = strlen(prefix) - 1;
      char * stripedDir = (char*) malloc ( prefixNum *
sizeof(char));
      strcpy(stripedDir,prefix);
      stripedDir[prefixNum]='\0';*/
      if(_access(dirComWithOut,0))
      {
            char prefixx[] = "md ";
            int all = strlen(prefixx)+strlen(dirComWithOut);
            char * together = (char*) malloc ( all *sizeof(char));
            strcpy(together,prefixx);
            strcat(together,dirComWithOut);
            //together[all]='\0';
            system(together);
      }

      if (chdir (dirComWithOut) == -1)
      {
        printf ("chdir failed - %s\n", strerror (errno));
            return;
    }
      fileName = aboutFiles(CategoryName);//Located in
aboutFiles.cpp, purpose to greate a unique file name for the
simulation at hand taking into cognizance the fact that each
simulation is given to 1 or more particle at a with distance and ttl
kept constant with each run. In this file each particle has a log
position in the this textfile, whether it be lost of if it gets to
its destination. This about file is only concerned with creating the
name of the files, i.e is making sure that a file with that name
existed not before. The purpose of this is to make sure there a
excessive logs of ran and re-ran experiments for the investigators
purposes. This file name comprises of the generic name first and
then a number attached to it to give it is uniqueness. This files go
from the following model: FileName, FileName1,...., This function
creates a name using the filename at first checks if it exists, if
it does exist, it attaches 1 to that file name check whether the
```

```
name exists again, if it does increments it by one and tries again,
it continues in this fashion until it finds one that exists. This
one it then returns to the the calling function as a string.
      int fileNum = strlen(fileName);
      if (chdir ("..") == -1)
      {
        printf ("chdir failed - %s\n", strerror (errno));
            return;
    }
      int fileAndDirNum = fileNum + strlen(dirComWith);
      char *fileAndDirName = (char*) malloc
((fileAndDirNum)*sizeof(char));
      strcpy(fileAndDirName, dirComWith);
      strcat(fileAndDirName, fileName);
      fp = fopen(fileAndDirName,"w");// this open a new file for
writing with the returned file name
      fileNameGen = aboutFilesGeneral1D(genericName,
&T1,&T2,&T3,&T4,&T5,&T6,&T7,&T8,&T9);// this function is located in
a file called "aboutFilesGeneral1D.cpp. The purpose of this function
is to be a log file for all files generated by the 1D case of the
varying dimensions case. What it does is to either create the file,
set up initial formating and return integer values that would help
format results of each file for each simulation result in this
general file. Each simulation in the special case will have its
summary stored as an entry in this log file. The initial part of the
file will bear the Prepart of the names common with all the files
generated in this experiment. Its end will actual bear the name
"Overall". What will be logged in this file for each simulation will
be: The file name, the distance considered, The ttl, Total Particle
number, Unit of time per step, Number of particles that got to the
distination, fraction of particles that got to the distination,
Total time it took for the transmission and the Average Arrival
time. The integers passed in are to help in formating the table in
the file. If the file already does exist only the retrival of the
integers and the setting the end cursor to a newline will be
achieved by this function.
      fp1 = fopen(fileNameGen,"a+");// The overall file name is
opened in appending mode here
      // the following integers are to help in formating the current
simulation file, both in the set up of the headings of each column
and the arrangement of the input integer. The elements of each
colomn as with that of the general file will be right justified, the
Left will be for the unique number of particle input and the right
will be for the time it took to get there, in cases where no such
gotten that space will contain the word "LOST" instead
      char *freshCategoryName = (char*) malloc ((fileNum -
3)*sizeof(char));
      strncpy(freshCategoryName, fileName, fileNum-4);
      freshCategoryName[fileNum-4] = '\0';
      int Left,
            Right;
      HeadingAndFormating(fp, &Left, &Right,
                                     "ParticleNumber",
freshCategoryName);// this function does the initial formating of
the text file to be created for the executing simulation and also
returns the integer values of integers left and right. This function
is located in the HeadingAndFormating.cpp file
      int totSuccessfulParticlesTime = 0, // This gives the total
time it took to transmit those particles that got to the destination
      totLostParticleCount = 0, // this takes into account only
those particles that made it to the destination
```

```
      particleNumber = 1;// this keeps track of the particle being
observed and so it is set to one initially for the first will be
observed at the beginning
      double averageParticleArrivalTime = 0, // this will keep the
record of the average arrival time of the particles given by
totSuccessfulParticlesTime/ totSuccessfulParticleCount
      arrivalFraction = 0; // This is to give the fraction of
particles that made it to the destination, and this is given by
totSuccessfulParticleCount/ TotalParticleNumber
      // the Journey of the particles take place in the following do
while loop, one particle at a time in the function particleJourny
the  progress of one particle is observed in terms of it progress
from the source in direction at each step time is increased, but the
step could be additive or subtractive to the total distance covered.
This function is located in the ccp file particleJourney. It takes
as argument two reference variables of int type, a file pointer to a
file unique to the simulation at hand for reporting, the particle
number for reporting, the destination distance to know when the
limit is breached, the time step which shows the time each step
takes, time to live integer variable to decide when it is right to
drop a particle, Left and Right integers to help in the report file
formating. The particles a dealt with one after the other until all
the particles have being transmitted
      int preLostValue = totLostParticleCount;
      do{
            particleJourny(&totSuccessfulParticlesTime,
                                    &totLostParticleCount,
                                    fp, particleNumber,
                                    DestDistance, deltaTime,
                                    ttl, Left, Right);// to simulate
the journey of a particle, recording it time after it reaches,
recording it lost
            if(preLostValue == totLostParticleCount)
                  particleNumber++;// heralds the next particle of
the transmission
            else
                  preLostValue = totLostParticleCount;
      }while(particleNumber <= TotalParticleNumber);// this
construct ensures that Number of particles proposed is processed
      int allParticles = totLostParticleCount + TotalParticleNumber;

      if(TotalParticleNumber)// is makes sure that division by zero
does not occur
      averageParticleArrivalTime =
(double)totSuccessfulParticlesTime/ (double)TotalParticleNumber;//
average time if not zero
      else
            averageParticleArrivalTime = 0;// if zero
      if(allParticles)
      arrivalFraction = (double)TotalParticleNumber/
(double)allParticles;
      else
            arrivalFraction = 0;
      /*printf("\n\n");
      printf("totSuccessfulParticlesTime = %d\n",
totSuccessfulParticlesTime);
      printf("totSuccessfulParticleCount = %d\n",
totSuccessfulParticleCount);
      printf("averageParticleArrivalTime = %f\n",
averageParticleArrivalTime);
      printf("arrivalFraction = %f\n", arrivalFraction);
      printf("\n\n");*/
      static int once = 0;
      if(!(once++))
            printf("\n\nLog file name is %s\n\n", fileNameGen);//
printing the overall file name
      printf("current file name is %s\n", fileName);// printing the
recently concluded simulations filename

      // fprintf, prints to the general file particulars of the
simulation file just concluded, this is done to enable comparisons
with other runs
      fprintf(fp1, "%*s, %*.0f, %*d, %*d, %*d, %*d, %*f, %*d,
%*f\n", T1, fileName,
            T2, DestDistance,
            T3, ttl,
            T4, allParticles,
            T5, deltaTime,
            T6, totLostParticleCount,
            T7, arrivalFraction,
            T8, totSuccessfulParticlesTime,
            T9, averageParticleArrivalTime);
      fclose(fp1);// general file closed
      fclose(fp);// specific simulation file closed
}
#include<stdlib.h>
#include<stdio.h>
#include"threein1.h"

void particleJourny(int *totSuccessfulParticlesTime, int
*totLostParticleCount, FILE *fp, int particleNumber,
                              double destinationDistance, int
deltaTime, int tTL, int formatL, int formatR)
{
      int time = 0;// each particle starts from time zero
      double deltaDistance = 0,// initial step at time zero is zero
of course
      distance = 0;// initial distance is 0
      do// this is an infinitive do while which enables the movement
to the particle in a brownian fashion. This movement is helped by
the brownian fashioned time steps enable by the gaussian() step
generator powered by the gaussian distribution which has bin found
to mimick the brownian motion to a large degree
      {
            deltaDistance = gaussrand();// this function returns at
random steps ranging between (-3 to 3). In NormalRandGenerator.cpp
            distance = distance + deltaDistance;
            if (distance<0)
                  distance= - distance; // update to the distance*/
            time += deltaTime;// added to the time this does
            if(destinationBreached(destinationDistance, distance))//
return 0 if reached but 1 otherwise , located in lostStatus.cpp
            {
                  if(lostStatusTime(tTL, time))// if time to live
exceeded returns 1 if not 0. located in lostStatus.cpp, if exceed
report made and put infile if not next step is taken
                  {
                        //LostReport(particleNumber, fp, formatL,
formatR);// reports missing located in LostReport.cpp
                        *totLostParticleCount += 1; // adding to the
already amassed successful particle number
                        return;// exit current particle journey
                  }
            }
            else
            {
                  ArrivalReport( particleNumber, fp, time, formatL,
formatR);// reports successful located in ArrivalReport.cpp
                  *totSuccessfulParticlesTime += time; // adding to
the already amass time

                  return;// exit current particle journey
            }
      }
      while(1);
}
#include<stdio.h>
#include<stdlib.h>
#include"threein1.h"

void ParticuleJourneyChronicles(FILE *fp, int timeStep, int
ParticleNumber, double distance, double width,
     int ttl, int *totLostParticleCount, int
*totalSuccessParticleTime, int formatL, int formatR)
{
      double mid_Way = width/2.0, x = 0, y = mid_Way, deltaX,
deltaY, endBounds = y/10.0,
            upperBounds = y + endBounds, lowerBounds = y - endBounds
;
      int t = 0;
      do
      {
            gaussrand2d(&deltaX,&deltaY);
            //deltaX = gaussrand();
            //deltaY = gaussrand();
            x = x + deltaX;
            y = y + deltaY;
            t = t + timeStep;
            //printf("x= %f, y = %f, deltaX = %f, deltaY = %f t =
%d, ttl = %d\n", x, y, deltaX, deltaY, t, ttl);
            //system("pause");
            /*int inbounds;
            do{
                  inbounds = 0;
                  if(x<0)
                  {
                        x = -x;
                        inbounds =1;
                  }
                  if(y<0)
                  {
                        y = -y;
                        inbounds =1;
                  }
                  if(y>width)
                  {
                        y = 2*width - y;
                        inbounds =1;
                  }
                  if(x> distance&&(y>upperBounds||y<lowerBounds))
                  {
                        x = 2*distance - x;
                        inbounds =1;
                  }
            }while(inbounds);*/
            if(WithinReach(mid_Way, distance, y, x, endBounds))
            {
                  if(t== ttl)
                  {
                        /*LostReport(ParticleNumber, fp,
                              formatL, formatR);*/
                  (*totLostParticleCount)++;
                  //printf("mark if");
                  printf("%d\n", t);
                        return;
                  }
            }
            else
            {

                  if(y<upperBounds&&y>lowerBounds)
                  {
                        ArrivalReport( ParticleNumber, fp, t,
                                    formatL, formatR);
                        (*totalSuccessParticleTime)+=t;
                        //printf("mark else");
                        return;
                  }
            }
      }
      while(1);
}
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include"threein1.h"
#include<io.h>

void particleTrajectory(int partTransitTime, int totalParticles, int
deltaTime)
{
      int Fstcount=0,Sndcount=0,Trdcount=0;
      int g = partTransitTime;
    do{Fstcount++;}while(g/=10);
    Fstcount++;
    g = totalParticles;
    do{Sndcount++;}while(g/=10);
    Sndcount++;
    g = deltaTime;
    do{Trdcount++;}while(g/=10);
    Trdcount++;
    char *CategoryName = (char*) malloc
((Fstcount+Sndcount+Trdcount+3)*sizeof(char));
      sprintf(CategoryName,"OD/%d_%d_%d",partTransitTime,totalPartic
les,deltaTime);

      int particleCount = 0, track = 0// char numbers
            ;
      /*getPreferedSettings(&distanceCompare, &partTransitTime,
            &totalParticles, &deltaTime);*/
      char genericName[]= "ODC/Dim1c"
            ;
      while(genericName[++track]!= '/');
      // create directory array
      char *direct = (char*) malloc (track * sizeof(char));
      // store directory name
      strcpy(direct, genericName);
      direct[track] = '\0';
      // check whether it exists
      if(_access(direct,0))
      {
            char prefix[] = "md ";
            int total = track + strlen(prefix);
            char *combo = (char*) malloc (total * sizeof(char));
            strcpy(combo, prefix);
            strcat(combo, direct);
            system(combo);// stopped here
      }//if

      char *fileName, *fileNameGen;
      FILE* fp, *fp1;
      int T1, T2, T3, T4;
      fileName = aboutFiles(genericName);
      fileNameGen =
aboutFilesGeneralVD(genericName,&T1,&T2,&T3,&T4);
      fp = fopen(fileName,"w");
      fp1 = fopen(fileNameGen,"a+");
      int Left, Right;
      HeadingAndFormating(fp, &Left, &Right,
            "ParticleNumber", "Distance");
      transmitAndObserveParticles(partTransitTime,
            totalParticles, deltaTime, Left, Right, fp);
      static int once = 0;
      if(!(once++))
            printf("\n\nLog file name is %s\n\n", fileNameGen);
      printf("current file name is %s\n", fileName);
      fprintf(fp1, "%*s, %*d, %*d, %*d\n", T1, fileName,
            T2, partTransitTime,
            T3, totalParticles,
            T4, deltaTime);
      fclose(fp);
      fclose(fp1);
}//particleTrajectory
#include"threein1.h"
#include<stdio.h>

void transmitAndObserveParticles(int partTransitTime, int
totalParticles,
    int deltaTime, int Left, int Right, FILE *fp)
{
      int particleNumber = 1, flag = 1;
      do{
            int Time = 0, stillJourneying = 1;
            double distance = 0, deltaDist;
            do{
                  deltaDist = gaussrand();
                  distance += deltaDist;
                  Time += deltaTime;
                  if(Time >= partTransitTime)
                  {
                         ArrivalReportDouble(particleNumber, fp,
distance,
                              Left, Right);
                         stillJourneying = 0;
                         if(particleNumber<=totalParticles)
```

```
                        {
                              particleNumber++;
                        }//if
                        else
                              flag = 0;
                  }//if
            }while(stillJourneying);
      }while(flag);
}//transmitAndObserveParticles
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<direct.h>
#include "threein1.h"
#include<io.h>

void twodimdriver(double distance, double width, int ttl, int
totalParticles, int deltaTime, int iterations)
{
      int Fstcount=0,Sndcount=0,Trdcount=0,Fothcount=0,Fithcount=0;
      int g = (int)distance;
    do{Fstcount++;}while(g/=10);
    Fstcount++;
    /*g = (int)destYCood;
    do{Sndcount++;}while(g/=10);
    Sndcount++;*/
    /*g = (int)radius;
    do{Trdcount++;}while(g/=10);
    Trdcount++;*/
      double fl = width;
      int padding = 3;
      int dp=fracNumCount(fl);
      int dpAndPadding = dp + padding;
    g = ttl;
    do{Fothcount++;}while(g/=10);
    Fothcount++;
    g = totalParticles;
    do{Fithcount++;}while(g/=10);
    Fithcount++;
    char *CategoryName = (char*) malloc
((Fstcount+dpAndPadding+Fothcount+Fithcount)*sizeof(char));
      sprintf(CategoryName,"%d_%.*f_%d_%d", (int)distance, dp, fl,
ttl, totalParticles);
      char dirStarting[]="TD";
      char dirSlash[]="/";
      int dirStartingNum = strlen(dirStarting),
dirSlashNum=strlen(dirSlash),
            dirEndNum= strlen(CategoryName),
            dirComWithOutNum = dirStartingNum + dirEndNum,
            dirComWithNum = dirStartingNum + dirEndNum +
dirSlashNum;
      if(iterations == 1)
      {
            dirComWithOutNum = dirStartingNum;
            dirComWithNum = dirStartingNum + dirSlashNum;
      }
      char *dirComWithOut = (char*) malloc
((dirComWithOutNum)*sizeof(char));
      strcpy(dirComWithOut,dirStarting);
      if(iterations > 1)
      {
            strcat(dirComWithOut,CategoryName);
      }
      char *dirComWith = (char*) malloc
((dirComWithNum)*sizeof(char));
      strcpy(dirComWith,dirStarting);
      if(iterations > 1)
      {
            strcat(dirComWith,CategoryName);
      }
      //strcat(dirComWith,CategoryName);
      strcat(dirComWith,dirSlash);

      int totLostParticleCount = 0, particleNumber = 1,
totSuccessfulParticlesTime = 0;// left to right: successful particle
count and soujourning particle number
      /*getPreferedSettings1(&destXCood, &destYCood, &radius, &ttl,
            &totalParticles, &deltaTime);*/
      char genericName[]= "Dim2v";// generic name for simulations
      /*int track = 0;
      while(genericName[++track]!= '/');// isolating the directory
part of the generic name in terms of numbers
      // create directory name array
      char *direct = (char*) malloc (track * sizeof(char));
      // store directory name
      strcpy(direct, genericName);
      direct[track] = '\0';
      // check whether it exists
      if(_access(direct,0))
      {// if it doesn't it is created here :-D
            char prefix[] = "md ";
            int total = track + strlen(prefix);
            char *combo = (char*) malloc (total * sizeof(char));
            strcpy(combo, prefix);
            strcat(combo, direct);
            system(combo);// stopped here
      }//if*/

      char *fileName, *fileNameGen;// single simulation file and log
file name holders
      FILE* fp, *fp1;// file pointers
      int T1, T2, T3, T4, T5, T6, T7, T8, T9;// format aids
      if(_access(dirComWithOut,0))
      {
            char prefixx[] = "md ";
            int all = strlen(prefixx)+strlen(dirComWithOut);
            char * together = (char*) malloc ( all *sizeof(char));
            strcpy(together,prefixx);
            strcat(together,dirComWithOut);
            //together[all]='\0';
            system(together);
      }

      if (chdir (dirComWithOut) == -1)
      {
        printf ("chdir failed - %s\n", strerror (errno));
            return;
    }
      fileName = aboutFiles(CategoryName);//creates single
simulation unique file name, aboutFiles.cpp
      int fileNum = strlen(fileName);
      if (chdir ("..") == -1)
      {
        printf ("chdir failed - %s\n", strerror (errno));
            return;
    }
      int fileAndDirNum = fileNum + strlen(dirComWith);
      char *fileAndDirName = (char*) malloc
((fileAndDirNum)*sizeof(char));
      strcpy(fileAndDirName, dirComWith);
      strcat(fileAndDirName, fileName);
      fileNameGen =
aboutFilesGeneral2D(genericName,&T1,&T2,&T3,&T4,&T5,&T6,&T7,&T8,&T9)
;
      fp = fopen(fileAndDirName,"w");
      fp1 = fopen(fileNameGen,"a+");
      char *freshCategoryName = (char*) malloc ((fileNum -
3)*sizeof(char));
      strncpy(freshCategoryName, fileName, fileNum-4);
      freshCategoryName[fileNum-4] = '\0';
      int L, R;
      HeadingAndFormating( fp,&L, &R, "ParticleNumber",
freshCategoryName);
      int preLostValue = totLostParticleCount;
      int itNum = 1;
      do
      {
            //printf("%d\n",particleNumber);
            //system("pause");
            ParticuleJourneyChronicles(fp,
                                             deltaTime,
                                             particleNumber,
                                             distance,
                                             width,
                                             ttl,

&totLostParticleCount,&totSuccessfulParticlesTime,
                                             L,
                                             R);

            if(preLostValue == totLostParticleCount)
                  particleNumber++;// heralds the next particle of
the transmission
            else
                  preLostValue = totLostParticleCount;
            printf("%d. Lost: %d, reached: %d\n", itNum++,
totLostParticleCount, particleNumber-1);
            /*if(!(itNum%100))
                  system("Pause");*/
      }
      while(particleNumber <= totalParticles);
      double averageParticleArrivalTime;
      int allParticles = totLostParticleCount + totalParticles;
      double arrivalFraction = 0;

      if(totalParticles)
            arrivalFraction = (double)totalParticles/
(double)allParticles;
      else
            arrivalFraction = 0;
      if(totalParticles)// is makes sure that division by zero does
not occur
```

```
            averageParticleArrivalTime =
(double)totSuccessfulParticlesTime/ (double)totalParticles;//
average time if not zero
      else
            averageParticleArrivalTime = 0;// if zero
      /*printf("\n\n");
      printf("totSuccessfulParticleCount = %d\n", particleCount);
      printf("arrivalFraction = %f\n", arrivalFraction);
      printf("\n\n");
      printf("current file name is %s\n", fileName);
      printf("Log file name is %s\n", fileNameGen);*/
      static int once = 0;
      if(!(once++))
            printf("\n\nLog file name is %s\n\n", fileNameGen);
      printf("current file name is %s\n", fileName);

      fprintf(fp1, "%*s, %*.0f, %*.0f, %*.0d, %*d, %*f, %*d, %*d,
%*f\n", T1, fileName,
            T2, distance,
            T3, width,
            T4, allParticles,
            T5, ttl,
            T6, averageParticleArrivalTime,
            T7, deltaTime,
            T8, totLostParticleCount,
            T9, arrivalFraction);
      fclose(fp1);

      fclose(fp);
      return;
}//twodimdriver
#include<stdio.h>
#include<stdlib.h>
#include<direct.h>
#include<string.h>
#include<time.h>
#include<io.h>
#include "threein1.h"


void TwoDimensional()
{
      double dist, width;
      int ttl, totalParticles, iterations,
            deltaTime;// particlars of unique to each simulation
      char textFileName[] = "TwoDimensional.txt";// file to read the
simulation particlars from
      // check if file exist
      if((_access(textFileName ,0)))
      {
            printf("no job for the function %s\n", textFileName);//
info for no job
            return;// exiting function
      }//if, checking for the file

      time_t rawtime;
        struct tm * timeinfo;
        char buffer [40];
        time (&rawtime);
        timeinfo = localtime (&rawtime);
        strftime (buffer,40,"TD\\%a_%Y-%m-%d\\%I_%M_%p",timeinfo);
        char prefix[]="md ";
        int bufferNum = strlen(buffer), prefixNum = strlen(prefix),i
=bufferNum + prefixNum;
        char * combo = (char *) malloc (i * sizeof (char));
        strcpy(combo,prefix);
      strcat(combo, buffer);
      system(combo);
      char prefix2[]="move ";
      char moveEnd[]= " ";
      int textFileNameNum =strlen(textFileName);
      int moveEndNum = strlen(moveEnd);
      int prefix2Num = strlen(prefix2);
      int combo2Num = prefix2Num + textFileNameNum + moveEndNum +
bufferNum;
      char * combo2 = (char *) malloc (combo2Num * sizeof (char));
      strcpy(combo2,prefix2);
      strcat(combo2, textFileName);
      strcat(combo2, moveEnd);
      strcat(combo2, buffer);
      system(combo2);
      if (chdir (buffer) == -1)
      {
        printf ("chdir failed - %s\n", strerror (errno));
            return;
    }
      FILE *fp = fopen(textFileName, "r");// file opening
      // getting rid of labels
      char getRid = 'q';
      while(getRid != '\n')
            fscanf(fp,"%c", &getRid);
      int times = 0;// flag to ascertain run
      printf("containing folder is %s\n", buffer);

      while(fscanf(fp,"%lf%lf%d%d%d%d", &dist, &width, &ttl,
&totalParticles, &deltaTime,&iterations) != EOF)//order in file of
particulars
      {
            times++;
            int counter = 1;
            while(counter++ <= iterations)
                  twodimdriver(dist, width, ttl, totalParticles,
deltaTime, iterations);// particle journey, twodimdriver.cpp
      }
      fclose(fp);// closes file
      if(!times)
            printf("nothing in the file\n");// if notin in file
      //system("rename TwoDimensional.txt TwoDimensionalOld.txt");//
rename after
}
#include<stdio.h>
#include<math.h>
#include "threein1.h"

int WithinReach(double yOrigin, double xOrigin, double yCood, double
xCood, double radius)
{
      double first, second, third, sum;
      first = yOrigin - yCood;
      first*=first;
      second = xOrigin - xCood;
      second*=second;
      third = radius * radius;
      sum = first + second;
      if(sum <= third)
            return 0;
      else
            return 1;
}
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include"threein1.h"
/*This is documentation for my Thesis program. This program is meant
to simulate the motion of hypothetical particles from, a particle at
a time, from a server to a destination under the influence of
diffusion through a medium. This diffusion is governed by a law of
normal distribution. So what this simulates is the times it, the
particle takes to get from the source to the destination. To aid
this simualation, automatic filing systems have being worked into
this program.
To include the header file(threein1.h) in the project you need to
take note of where the file is located then in vs2008 you do the
following Project->Project(Name)properties->C/C++(Left Pane) ->
General -> Additional Include directory(Right Pane) -> put in
directory
To put in source files: Open solution explorer, right click on
source folder, -> add new-> existing files-> browse to where the
files are*/
int main ()
{
      // RANDOM GENERATOR INITIATION
      srand((unsigned) time (NULL));
      /*this function is to help the psuedorand number generator of
the c enviroment to generate random numbers by seeding it with the
underlying oses time. C's internal random number generator is inturn
used in the gaussian random number generator function to generate
random numbers which mimic the steps of the nano particle. The time
of this particles soujourn is tied to each step of this particle.
And in the case of the one dimension, this particle can either go
forward or backwards, but the time step is always added with each
generate step. Hence with this we can but in the time to live
feature which accordiing to the research, can be put in place by a
natural phenomenon. This initialization will be employed by the
gaussrand() located in the NormalRandGenerator.cpp file. Think for
this function stdio or stdlib is needed*/

      //Preamble
      printf("Preamble\n\n");
      printf("This program contains 3 simulations:\n"
            " 1)One Dimensional Dist\n"
            " 2) One Dimensional Time\n"
            " 3) Two Dimensional Dist\n");
      system("pause");
      printf("The data required for these simulations are contained
in the following respective files:\n"
            " 1) OneDimProper.txt, columns: DestinationDist, ttl,
TotPartNum, timeStep\n"
            " 2) constTimeVaringDist.txt, columns:PartTransTime,
TotPartNum, timeStep\n"
            " 3) TwoDimensional.txt, columns: Xcood, Ycood, Radius,
ttl, TotPartNum, timeStep\n");
      system("pause");
      /*The above just helps the user understand a little better how
this program works, it uses the input provided for by textfiles. As
observed above The Program does 3 major duties, 1. Takes note of
times achieved at specific distances, and how many packets gets to
destination at a given different distances, 2. The 2nd program takes
note of the largest distance achieved by each particle giving a
fixed time 3. This is the repeat of the first case scenario only
that this is a 2 dimensional plane and the catchment area of the
destination is circular in this case
      How this work is if there is no file, or if there is no data
given in the file that part does not run otherwise, it runs. Each
section has a prefered data format if not it won't work properly, so
the data must be arranged in each file as given in the column
description above. Now if a file does exist, the after the run, the
file is renamed so as to prevent rerunning same data by mistake in
the next run. Please note that this file must be in the same
directory as the that contain the main source file*/

      OneDimProper();// attempts executing the first, this function
is located in OneDimProper.cpp
      system("pause");// pauses the program to let the user observer
the result before moving on to the next, to move on, the enter
button should be employed
      constTimeVaryingDist();//constTimeVaryingDist.cpp
      system("pause");
      TwoDimensional();//TwoDimensional.cpp
      system("pause");
      /*int select = 0, below = 0, above = 3;

      do
      {
            printf("\n\n\n");
            printf("type \n");
            printf("{0} to Exit the Program\n");
            printf("{1} To Run the Constant Destination One Dimen
Simulation\n");
            printf("{2} to Run the Constant Ttl One Dimen
Simulation\n");
            printf("{3} to Run the Constant Destination Two Dimen
Simulation\n");
            printf("value: ");
            scanf("%d", &select);
            if(select > above || select < below)
                  printf("Out of range please try again\n");
            else if(select == 0)
                  return 0;
            else if(select == 1)
            {
                  OneDimProper();
            }
            else if(select == 2)
            {
                  constTimeVaryingDist();
            }
            else if(select == 3)
            {
                  TwoDimensional();
            }
      }while(1);
*/
      }
```