NANOSIM: A SIMULATION FRAMEWORK FOR NANOSCALE MOLECULAR COMMUNICATION NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ERTAN GÜL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

MAY 2010

Approval of the thesis:

NANOSIM: A SIMULATION FRAMEWORK FOR NANOSCALE MOLECULAR COMMUNICATION NETWORKS

submitted by **ERTAN GÜL** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. İsmet Erkmen _____
Head of Department, **Electrical and Electronics Engineering**

Assoc. Prof. Dr. Özgür Barış Akan
Supervisor, **Electrical and Electronics Engineering Dept., METU** _____

Examining Committee Members:

Prof. Dr. Semih Bilgen _____
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Özgür Barış Akan _____
Electrical and Electronics Engineering Dept., METU

Asst. Prof. Dr. Cüneyt Bazlamaçcı _____
Electrical and Electronics Engineering Dept., METU

Asst. Prof. Dr. Şenan Ece Schmidt _____
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Tuna Tuğcu _____
Computer Engineering Dept., Boğaziçi University

Date: _____

I thereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : ERTAN GÜL

Signature :

ABSTRACT

NANOSIM: A SIMULATION FRAMEWORK FOR NANOSCALE MOLECULAR COMMUNICATION NETWORKS

Gül, Ertan

M.Sc., Department of Electrical and Electronics Engineering

Supervisor: Assoc. Prof. Dr. Özgür Barış Akan

May 2010, 118 pages

A number of nanomachines that cooperatively communicate and share information in order to achieve specific tasks is envisioned as a nanonetwork. Due to size and capabilities of nanomachines, the traditional communication paradigms cannot be used for nanonetworks in which network nodes may be composed of just several atoms or molecules and scale on the orders of few nanometers. Instead, molecular communication is a promising solution approach for nanoscale communication paradigm. However, molecular communication must be thoroughly investigated to realize the nanoscale communication and nanonetworks for many envisioned applications such as nanoscale body area networks, nanoscale molecular computers. In this thesis, a simulation framework (NanoSim) for nanoscale molecular communication networks is presented. The objective of the framework is to provide a simulation experimental tool in order to create a better understanding of nanonetworks and facilitate the development of new communication techniques and validation of theoretical results. The NanoSim framework is built on top of core components of widely

used network simulator (ns-2). It incorporates the simulation modules for various nanoscale communication paradigms based on diffusive molecular, motor-based and gap junction-based molecular communication channels. The details of NanoSim are discussed and some functional scenarios are defined to validate NanoSim. In addition to this, the numerical analyses of these functional scenarios and their experimental results are presented. The validation of NanoSim is done by comparing these experimental and numerical results.

Keywords: Molecular Communication, Molecular Network Simulator, Nanonetwork Simulator, Diffusive Molecular Communication, Motor-Based Molecular Communication, Gap Junction-Based Molecular Communication

ÖZ

NANOSIM: NANO-ÖLÇEKLİ MOLEKÜLER İLETİŞİM AĞLARI İÇİN BENZETİM ÇERÇEVESİ

Gül, Ertan

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Özgür Barış Akan

Mayıs 2010, 118 sayfa

Belirli işlevleri yerine getirmek için işbirliği yaparak haberleşen ve bilgi paylaşan bir kaç nanomakine nano ağ olarak tasavvur edilir. Boyutları ve kabiliyetlerinden dolayı, geleneksel haberleşme paradigmaları, sadece bir kaç mol atom veya molekülden oluşabilen ve birkaç nanometre boyutlarında olabilen nanomakinelerin arasındaki haberleşme için kullanılamaz. Bunun yerine, moleküler haberleşme nano-ölçekli haberleşme paradigması için gelecek vadeden bir çözüm yaklaşımıdır. Fakat, moleküler haberleşme, nano-ölçekli haberleşme ve nano ağları nano-ölçekli vücut alanı ağı, nano-ölçekli moleküler bilgisayarlar gibi birçok uygulamada gerçekleştirmek için detaylı bir şekilde araştırılmalıdır. Bu tezde, nano-ölçekli moleküler haberleşme ağları için bir benzetim çerçevesi sunulmuştur. Çerçevenin amacı nano ağları daha iyi anlayabilmek ve yeni haberleşme teknikleri ve teorik sonuçları onaylamak için deneysel benzetim aracı sağlamaktır. NanoSim çerçevesi, geniş kullanım alanı bulunan ağ simülatörünün (ns-2) temel bileşenleri üzerine geliştirilmiştir. NanoSim difüzyon, motor-temelli ve oluklu bağlantı-temelli moleküler iletişim kanalları için

benzetim modülleri içerir. NanoSim'in detayları tartışılmış ve NanoSim'in doğrulaması için bazı fonksiyonel senaryolar tanımlanmıştır. Buna ek olarak, bu fonksiyonel senaryoların nümerik analizleri ve deneysel sonuçları sunulmuştur. Bu deneysel ve nümerik sonuçlar karşılaştırılarak, NanoSim'in doğrulaması yapılmıştır.
Anahtar Kelimeler: Moleküler Haberleşme, Moleküler Ağ Simülatorü, Nano Ağ Simülatorü, Difüzyon Temelli Moleküler Haberleşme, Motor Temelli Moleküler Haberleşme, Oluklu Bağlantı Temelli Moleküler Haberleşme

To my family

ACKNOWLEDGMENTS

I would like to express my gratefulness and deepest respect to my advisor, Dr. Özgür B. Akan. I am indebted to Dr. Akan for his great advice, support, help and showing me the right direction when I lost my way at certain times during this study.

I also thank Dr. Semih Bilgen, Dr. Cüneyt Bazlamaçcı, Dr. Şenan Ece Schmidt and Dr. Tuna Tuğcu who kindly agreed to serve in my M.Sc. Examining Committee.

I also would like to thank all the members of the Next Generation Wireless Communications Laboratory (NWCL), especially Barış Atakan, for their support during my graduate study.

I acknowledge all of my friends and colleagues, especially the members of Integration Validation Verification Qualification (IVVQ) team, for their support and encouragements. I am also grateful to HAVELSAN Inc. for the facilities that made my work easier.

My special thanks go to Bilge Ceylan for her support and for proof reading my thesis.

Finally, I am so grateful to my parents, Aysel and Yaşar Gül, for their love, support and encouragement, and I thank my brother, Erhan, the most important person for me, for everything.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

2D	Two Dimensional
3D	Three Dimensional
AM	Amplitude Modulation
ATP	Adenosine Triphosphate
CICR	Calcium Induced Calcium Release
Da	Daltons
DNA	Deoxyribonucleic Acid
FM	Frequency Modulation
GMP	Gillespie Multi Particle
LGA	Lattice Gas Automata
MAC	Media Access Control
MAP	Microtubule Associated Protein
ns-2	Network Simulator
ODE	Ordinary Differential Equation
pdf	Probability Density Function
SSA	Stochastic Simulation Algorithm
ssDNA	Single Stranded DNA
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access

CHAPTER 1

INTRODUCTION

The idea of nanotechnology was introduced nearly half century ago by Richard Feynman [55]. Nanotechnology enables the practical realization of nanoscale devices, the size of which ranges from 1 to 100 nanometers. The studies on nanotechnologies are categorized into two techniques. Dry technique covers studies of fabrication of nanoscale devices, composed of carbon, silicon and other inorganic materials such as carbon nanotubes and nanowires, whereas wet technique refers to all nanotechnologies that cover studies of biological systems that exist in aqueous medium [49].

In nanotechnology, nanomachines, which have limited size and limited capability, are considered as the smallest functional unit elements. Nanomachines are capable of performing basic tasks like movement generation, releasing molecules, receiving molecules via chemical reactions. Because of their simple structure, they need to cooperate with each other to perform complex tasks. This requirement brings the necessity of communication between nanomachines.

Nanonetwork, which is the interconnection between nanomachines, can be accomplished in several methods such as acoustic, electromagnetic, nanomechanical and molecular communication [48]. In acoustic communication, information is carried via acoustic signal. In nanomechanical communication, information flows through mechanical connection of nanoscale devices. In electromagnetic communication, information is encoded into electromagnetic waves. Molecules are utilized

as information carriers in molecular communication which can be identified as wet nanotechnology [15].

Molecular communication, inspired from the real biological molecular systems, gathers several disciplines such as nanotechnology, biotechnology and communication systems. By molecular communication, short-range (nm - m) communication between nanomachines is provided and molecules are used as communication carriers instead of electromagnetic waves. For instance, in molecular communication, transmitter nanomachine inserts molecules into aqueous medium and then receiver nanomachine captures the molecules from communication channel like biological structures, i.e., cell, tissue, organ, communicating with each other via special molecules in nature [1].

Molecular communication has several potential application areas. These areas can be outlined as follows [15]:

- **Biomedical Area:** The most exciting potential applications are on biomedical area as a result of biocompatibility feature of molecular communication. The future health care examples, presented in [14], are Lab-on-a-chip, Drug/DNA delivery system, and human body monitoring. Additionally, immune system support, bio-hybrid implants, and genetic engineering implementations are promising fields.

- **Environmental Area:** Since molecular communication is compatible and inspired from nature, it has potential environmental applications such as animal and biodiversity control, air pollution control, and biodegradation.

- **Industrial Area:** Due to the chemical sensitivity of molecular communication, food and water quality controls are examples of potential industrial applications.

- **Military Area:** Nuclear, biological and chemical defenses, and nano-functionalized equipments are the examples of potential military applications.

Molecular communication systems can be categorized into three models according to their propagation mechanisms. These are gap junction-based, motor-

based and diffusive molecular communication models.

Molecular communication through gap junctions is presented in [9]. The system is based on diffusion of information molecules through gap junctions which are special proteins that provide connection between two cells. This model provides shared medium for the nanomachines in the system. Calcium signaling is used as communication carrier in the system. In [22], more detailed design of a molecular channel with gap junctions and modeling of molecular communication system are proposed. In the designed system, signal switching, filtering and aggregation functionalities are controlled by adjusting the permeability and selectivity of gap junctions. Besides, a cell-based molecular communication network based on gap junction channels is designed. The proposed system focuses on basic computational logic units [5].

Molecular motor-based communication is inspired from intra-cell communication of real biological cells. Message molecules are transported by molecular motors. Biomolecular linear systems contain molecular motors and protein filaments. Molecular motors, i.e., kinesins, dyneins, and myosins are protein complexes that transform chemical energy into mechanical work. There are two propagation types in molecular motor-based communication system. For one of them, molecular motors carry the cargoes and move along protein filaments [8], [36] . For the other one, protein filaments glide over the immobilized molecular motors [3], [14]. Thus, in this model, protein filaments called as molecular rail [11]. This model can be considered as wired communication in traditional communication.

In addition to the models mentioned above, molecules may also freely propagate in an aqueous medium. In this case, firstly, transmitter nanomachine releases molecules into free aqueous medium, then molecules propagate in the molecular communication channel, and the receiver nanomachine captures these information molecules with chemical reactions as shown in Figure 1.1. The motion of particles in one dimension is modeled via Brownian motion and the channel capacity is computed in [7]. Simplified version of ligand-receptor mechanism is analyzed and channel capacity of diffusive biochemical channel is estimated in [10]. An in-

formation theoretical approach for molecular communication is presented in [17]. Molecular communication channel capacity between two nanomachines is analyzed in that paper. Molecular multiple access, broadcast and relay channel models for molecular communication are presented, and their capacity expressions are derived in [20]. Besides, deterministic capacity of information flow is analyzed in molecular nanonetworks in [4].

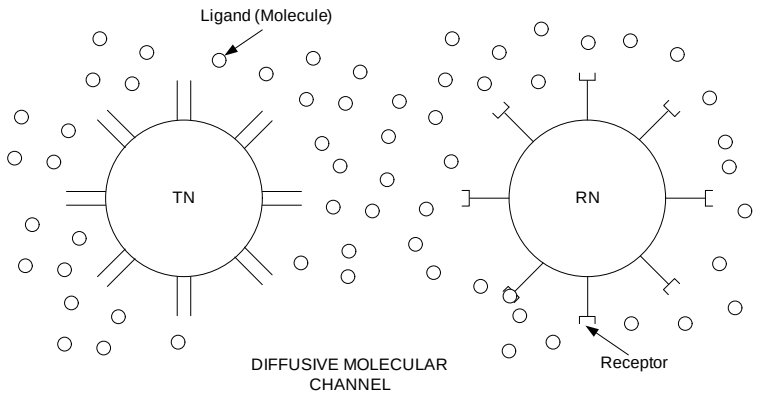

Figure 1.1: Diffusive molecular communication model [17].

The idea of developing a functional and powerful molecular communication simulator, which supplies different nanonetwork topologies and molecular communication models, has arisen, due to lack of open source molecular communication network simulator which provides a simulation environment to implement and evaluate new communication techniques for molecular communication.

The purpose of this thesis is to develop a molecular communication simulator which is based on nanoscale molecular communication networks for ns-2 that also supports the development of communication networks in addition to traditional one. The resulting simulation framework, NanoSim, provides a good infrastructure for further development and evaluation of molecular communication protocols. Besides, it will be used for validation of derived models. The design of NanoSim enables the code utilization for the implementation which will be developed for other molecular communication models. In this thesis, communication models defined above are

modeled and analyzed. The design and implementation details of NanoSim, which is developed for diffusive and motor-based communication models, are also given. Then, numerical analyses for validation scenarios of NanoSim are introduced. Finally, the performance evaluation and validation of NanoSim are presented.

In the rest of this chapter, firstly, discussion of the related work is given. Then, the background about ns-2 is introduced. Finally, the organization of thesis is presented.

1.1 Related Work

There are several reaction-diffusion models to cope with biochemical activities in different levels as examined in [38]. Highly accurate microscopic methods are too computational, whereas the macroscopic Ordinary Differential Equation (ODE) methods cannot handle biochemical activities enough. Because of that, we select mesoscopic method to simulate our system. We choose Gillespie-Multi-Particle method (GMP) [30] among several reaction-diffusion models [39], [40], [41]. In GMP method, diffusion and reaction events are discrete in contrast to [41], in which these events are merged. The distinction between reaction and diffusion phases diminishes computational cost of reaction-diffusion merged models and makes GMP method suitable for parallel computation. Moreover, the distinction between reaction and diffusion brings modularity to GMP algorithm. As an algorithm for diffusive molecular communication modules, diffusion phase of the GMP method is required in motor-based communication which does not need the reaction part of GMP method. Thus, we plug the diffusion model of GMP algorithm to motor-based communication model.

There are several cellular activity oriented simulators. One of them is E-Cell3 [33] which is an object-oriented program that aims to model, simulate and, analyze the complex bio-structures like cells. E-Cell Simulation Environment, in which numerous algorithms are applied, is a very powerful tool. There are also several similar cellular scale simulators such as V-cell [35] and cellware [34]. Even though these simulators are powerful for complex nature of biological systems, they are not ap-

propriate for molecular communication due to lack of network aspect. For instance, it is impossible to implement network protocols with them.

The simulation studies for molecular communication exist in the literature. In [18], unicast communication on microtubule topology is investigated. The probability of successful transmission for diffusive and motor-based propagation systems is compared. However, the topology of environment is composed of only one single sender, one single receiver and microtubule. The simplicity of the topology reduces the level of realism. In [13], delay characteristics of different microtubule topologies, which are fundamental structures of motor-based molecular communication system, are analyzed. In [43], three types of molecular communication propagation approaches are defined. In diffusion-only propagation system, there is not any microtubule in the environment and molecules diffuse between sender and receiver. In motor-only propagation system, sender and receiver are connected and information molecule is directly transferred from sender to receiver nanomachine. In the hybrid propagation system, information molecules are released, then information molecules may propagate to receiver by diffusing in the environment or binding microtubule which has instable behavior and moving through bound microtubule. The information rates of propagation models are compared. Additionally, the noise analysis of these models are investigated in [42].

Despite promising progress performed in molecular communication and existing cell simulators, there is currently no open source molecular communication simulator which provides practical and beneficial simulation suite to develop network protocols for nanonetworks. The necessity of a molecular communication network simulator motivates us to develop a simulator which allows the user to manage different nanonetwork topologies and molecular communication models.

1.2 Network Simulator (ns-2)

Network simulator is a discrete event-driven network simulator specially geared for scientific researches [32]. It provides the simulation of TCP, routing, and multicast protocols over wired and wireless networks, e.g., local and satellite. Moreover,

it supplies development environment to model promising networks which are different from traditional communication as [46]. Due to these features of ns-2, it is chosen as our simulation environment.

ns-2 is an object-oriented simulator which is written in C++ and OTcl interpreter. C++ is generally utilized in complicated protocol implementation as it is fast to run. On the other hand, OTcl is used for simulation configuration since it can be changed very quickly. OTcl interpreter, which is the user interface of ns-2 as shown in Figure 1.2, provides the scheduling of the simulation, setting the network topology, configuring network parameters and plumping the components of network [47]. In our simulator, ns-mol.tcl library is developed since we design a new node structure named as *NanoNode*, which is plumped in ns-mol.tcl, besides, new network components, parameters and methods for molecular communication are defined in this file.

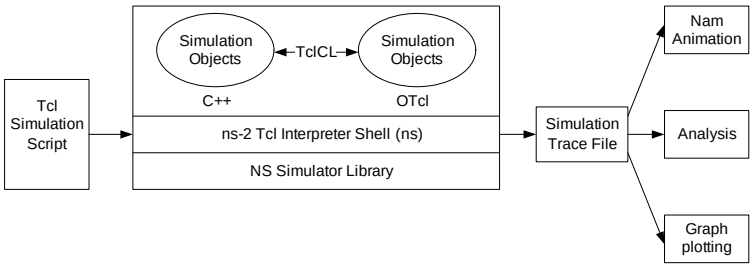

Figure 1.2: ns-2 simulation architecture [51].

ns-2 merges C++ and OTcl objects by providing support for split objects as depicted in Figure 1.2. When a simulator object is created through OTcl interpreter, first of all, an object is initialized in interpreted hierarchy, i.e., OTcl, and then a matching object in compiled hierarchy, i.e., C++, is created. The link between the objects in interpreted hierarchy and compiled hierarchy gives the control of C++ objects to OTcl. Attributes of these objects can be bound in the implementation. In this condition, when one object is changed in interpreted or compiled hierarchy, the other one is automatically changed.

One of the major components of ns-2 is the event scheduler. There are four

types of schedulers in ns-2. These are a simple linked-list, heap, calendar queue, i.e., default, and a special type called *real − time* [37]. The scheduler puts the events in a timeline. At the simulation time, the scheduler fetches the earliest event and executes it. The execution of an event means giving the event to associated handler function as input. After the execution of the event, scheduler moves to the next earliest event. This cycle continues until the simulation is over.

An event is composed of a unique id, a firing time, a handler function, and a pointer to next Event as shown in Figure 1.3. Two classes are derived from Event base class, *at − event* and *Packet* class. Instance of at-event class schedules the event at a particular time. On the other hand, Packet class is the constituent unit of exchanges between network components. In the simulator, we have implemented *MolecularData* class, which is used instead of Packet class as elaborately depicted in Section 3.3.3.

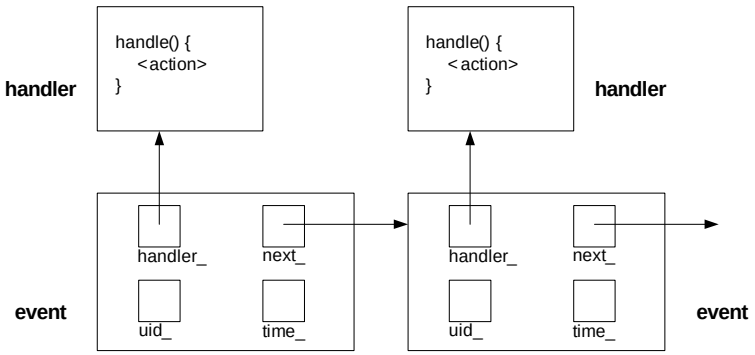

Figure 1.3: Event class architecture [51].

Although the exchange between network components is achieved by defining MolecularData class in the simulator, diffusion, and reaction events should also be included in ns-2 scheduler mechanism. Therefore, *handle()* functions are embedded into diffusion and reaction classes. Additionally, we design *MolecularEnvironment* class to trace the environment, since diffusion and reaction events depend on the environment. MolecularEnvironment class can be considered as the input of reaction-diffusion events.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, molecular communication models are explained including the details of the models in simulator design and employed algorithms. Design and implementation details of diffusive and motor-based molecular communication simulators are described in Chapter 3 and Chapter 4, respectively. Numerical analysis of molecular communication is introduced in Chapter 5. The validation and performance evaluation of NanoSim framework are presented in Chapter 6. Finally, concluding remarks are given in Chapter 7.

CHAPTER 2

MOLECULAR COMMUNICATION

In this chapter, firstly, the characteristics of molecular communication are briefly presented. Then, diffusive, motor-based and gap junction-based communication models are introduced. Meanwhile, the details of the models in simulator design are discussed.

Molecular communication has different characteristics compared to traditional communication. Some features may give the opportunity to develop more complex and efficient communication networks in the future, whereas some of them may limit the feasibility of molecular communication. Each of molecular communication capabilities is concisely given below [44]:

- **Biocompatibility:** The components of molecular communication are inspired from biological system. Thus, nanomachines use the same communication mechanism as biological systems. It brings the biocompatibility of molecular communication.

- **Functional complexity:** Cells, which have functional complexity, are composed of sensors, logic circuits, memory and actuators which generate motion [12].

- **Robustness and fragility:** Cells and biological systems have robustness against internal and external stimulations to some degree. However, they are very fragile to environmental conditions such as temperature and pH changes [12].

- **Stochasticity:** Communication models use free diffusion at least in one of their communication phase. Brownian motion, which is the cause of the free diffusion, causes particles to move in a random fashion. It is totally in a stochastic manner. Moreover, communication occurs through chemical channels between nanomachines, and chemical reactions are stochastic processes as a kind of random-walk process [27].

- **Delay:** Molecular communication is quite slower than traditional communication. Molecular motors carry molecules up to 400 mm/day inside the cell [15]. This large latency requires nanonodes to work in an asynchronous manner.

- **Energy efficiency:** As the result of evolution, molecular systems do not need any energy or they consume energy very efficiently. For example, diffusive molecular communication models do not need energy for propagation phase. Additionally, molecular motors use ATP to obtain kinetic energy. The energy efficiency of ATP consumption is about 90 percent.

Table 2.1 summarizes the features of molecular communication mentioned above and presents the comparison between traditional communication and molecular communication.

2.1 Diffusive Communication

Diffusive molecular communication is one of the fundamental mechanisms that is used by biological systems. For example, in the human body, thyrotropes in the hypophysis, i.e., senders, emit thyroid stimulating hormone, i.e., carrier molecules, into the environment, then the stimulated epithelial cells secrete thyroxine. Finally, thyroxine is received by thyroxine receptors from environment [11]. This model is based on passive transport, which means that no energy is required during transportation.

In nature, diffusive molecular communication is usually based on ligand-receptor binding mechanism. Biological sender node generates and inserts ligand molecules into the medium. The released molecules diffuse in the environment and bind to the

Table 2.1: Traditional vs. molecular communication [1].

	Traditional Communication	***Molecular Communication***
Communication carrier	Electromagnetic Waves	Molecules
Signal type	Electronic and optical (Electromagnetic)	Chemical
Propagation speed	Light ($3 \times 10^8 m/s$)	Extremely slow
Medium conditions	Almost immune	Affect communication
Noise	Electromagnetic fields and signals	Particles and molecules in medium
Encoded information	Voice, text, and video	Phenomena, chemical states, or processes
Other features	Accurate communication and high energy consumption	Stochastic communication and low energy consumption

receptor molecules of receiver node. Binding event is a chemical reaction between ligand and receptor molecules. This binding event allows the receiver node to capture the information molecules. After receiver nanomachine captures information molecules from molecular channel, it decodes the bound ligand molecules. The diffusive molecular communication model is depicted in Figure 1.1.

In the remainder of this section, first, literature background about diffusion and reaction processes is given, then reaction-diffusion algorithm which constructs the foundation of diffusive communication model is presented.

2.1.1 Diffusion Process

The source of diffusion is the Brownian motion, which is the random movement of the particles in gas or aqueous medium and a consequence of the constant thermal motion of atoms, molecules, and particles. It arises due to the collision of molecules with the atoms, molecules, and particles of the medium.

Fick's Second Law states that time rate of change in concentration is proportional to the curvature of concentration and to a constant called diffusion coefficient. In order to formulate the displacement of a single molecule, A, concentration parameter is replaced with the probability density function (pdf) of molecule A's displacement in one direction in the Fick's Second Law

$$\frac{dp_{A_x}(r,t)}{dt} = D_A \nabla^2 p_{A_x}(r,t) \tag{2.1}$$

The solution of (2.1), given in [25], [24], presents the pdf of molecule A's displacement in one direction, which has a Gaussian form as follows

$$G_S(\Delta x) \equiv \frac{1}{\sigma_A \sqrt{2\pi}} \exp(-\frac{\Delta x^2}{2\sigma_A^2}) \tag{2.2}$$

$$\sigma_A \equiv \sqrt{2D_A \Delta t} \tag{2.3}$$

in which $G_S(\Delta x)$ is normalized Gaussian with mean 0 and standard deviation is equal to σ, which is equal to step length of molecule A.

The displacements of each direction (x, y, z) are independent occurrences. Hence. the pdf of molecule A's total displacement is estimated as follows

$$p_A(r + \Delta r, t + \Delta t) = G_{S_A}(\Delta x) G_{S_A}(\Delta y) G_{S_A}(\Delta z) \tag{2.4}$$

(2.3) formulates the step length of molecule A in one dimension. The total displacement of molecule A is defined as follows

$$\Delta r = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2} \tag{2.5}$$

Therefore, the total displacement of molecule A, which is achieved by combining (2.3) and (2.5), is equal to $\sqrt{6D_A \Delta t}$.

2.1.2 Reaction Process

There are two types of homogeneous reaction approaches: deterministic approach and stochastic approach. Deterministic process is completely predictable and presented by differential reaction-rate equations. This approach assumes that reaction process has continuous and deterministic features. However, reaction system is not a continuous process as the amount of molecules is changed discretely, and it is not possible to predict the exact molecular population levels without the knowledge of the previous position and velocity of the molecules. On the other hand, in macroscopic systems, deterministic approach is applicable for a large number of interacting molecules. The randomness of this behavior averages out as the overall macroscopic state of the system becomes highly predictable [53].

Stochastic approach depicts that molecular reactions are random processes and it is impossible to predict the exact time of the next reaction and the next reaction channel. The commonly accepted algorithm for stochastic approach was proposed by Gillespie [26]. This formulation is applicable for any chemical system that is kept *well mixed*.

In order to apply stochastic approach to complex systems, a simple computational algorithm is proposed by Gillespie [27]. In Gillespie's method, N species which are able to react with other species through M reaction channels have been uniformly mixed in a fixed volume V:

$$\left.\begin{array}{r} R_1 : \ S_1 + S_2 \rightarrow C_1 \\ \vdots \\ R_M : \ S_{N-1} + S_N \rightarrow C_M \end{array}\right\} M \textit{ Reaction Channels} \tag{2.6}$$

where S is the input substance, C is the output component and R is the reaction channel.

In order to determine which reaction occurs next and when the next reaction occurs in the reaction system, i.e., "reaction probability density function", $P(\tau, \mu)$, is

defined as follows

$$P(\tau,\mu)d\tau \equiv \textit{probability that given the state } X_i(t)(i=1,...,N) \textit{ at time } t, \textit{ the next reaction in the volume will occur in the infinitesimal time interval } (t+\tau; t+\tau+d\tau) \textit{ and will be an } R_\mu. \tag{2.7}$$

where $X_i(t)(i = 1, ..., N)$ denotes the number of species.

Reaction probability of R_μ is the case that no reaction will occur during τ after reaction μ occurred. Hence, (2.7) equals the product of $P_0(\tau)$ and propensity value of R_μ, which gives the probability of occurrence of reaction μ in (t,t+τ):

$$P(\tau,\mu)d\tau = P_0(\tau)a_\mu d\tau \tag{2.8}$$

The propensity function pertains to the rate constant of reaction and the quantity of reaction input combinations as follows

$$a_\mu = h_\mu c_\mu \tag{2.9}$$

where c_μ is the rate constant which depends on the radius of the molecules, their average velocities and individual masses. h_μ is the quantity of reaction input combinations. For instance, h_μ value of reaction "$R_\mu : \ S_1 + S_2 \rightarrow anything$" equals X_1X_2. After applying derivations in [27], $P_0(\tau)$ is given as follows:

$$P_0(\tau) = \exp\left(-\sum_{\mu=1}^{M} a_\mu \tau\right) \tag{2.10}$$

From the synthesis of (2.8) and (2.10), following expression is derived

$$P(\tau_R,\mu) = \begin{cases} a_\mu \exp(-a_0\tau_R) & 0 \le \tau_R < \infty, \\ & \mu = 1, ..., M, \\ 0, & otherwise \end{cases} \tag{2.11}$$

where $a_\mu = h_\mu c_\mu$ and $a_0 = \sum_{\upsilon=1}^{M} a_\upsilon$. As shown in (2.11), $P(\tau_R,\mu)$ is equal to multiplication of two separate functions, $f(\mu)$ and $g(\tau)$. From (2.12) and (2.13), τ and μ values are obtained by assigning two random variables, r_1, r_2, in interval [0,1] [27].

$$\tau = \frac{1}{a_0}\ln\frac{1}{r_1} \tag{2.12}$$

$$\sum_{\upsilon=1}^{\mu-1} < r_2 a_0 \le \sum_{\upsilon=1}^{\mu} \tag{2.13}$$

2.1.3 Reaction-Diffusion Model

There are several stochastic approaches to model biochemical activities. We consider mesoscopic level archetype, which treats molecules discretely, but does not track positions in a compartment or within a subvolume, to model molecular channel of the simulator. Since our channel archetype needs both reaction and diffusion capabilities, we model our channel model with a reaction-diffusion paradigm.

GMP algorithm, which is particle-based spatial stochastic method, is used as reaction-diffusion model [30]. In this model, diffusion and reaction are distinct events. The basis of diffusion is the multiparticle lattice gas automata (LGA) algorithm [31]. In multiparticle LGA algorithm, medium is divided into lattice sides. Consequently, the inhomogeneity of the system is reduced to lattice volume. Each lattice site holds a discrete number of uniformly distributed particles. Molecules perform random walk on the lattices and are distributed to 6 neighbor lattices randomly. If a small number of molecules exist in the lattice, molecules move individually to neighbor lattice. If the number of molecules is larger than 60 [30], molecules are moved in a bulk to lattice according to Gaussian distribution. In the algorithm, the exact position of a molecule is not necessary, however only the lattice position of the molecule is required. Thus, lattice coordinate system is utilized in the simulator.

Normally, every species has a particular diffusion coefficient. Diffusion time step, τ_{D_S}, of each species is calculated as follows

$$\tau_{D_S} = \frac{1}{2d}\frac{\lambda^2}{D_S} \tag{2.14}$$

where D_S is diffusion coefficient of the species, λ is the length of each lattice, d is the dimension of simulation medium.

As indicated, reaction process is distinct from diffusion process. Reaction events occur between diffusion events. It is assumed that chemical reactions are local events. Therefore, Stochastic Simulation Algorithm (SSA) is applied to perform reaction events in each lattice side. The algorithm of GMP algorithm is given in Algorithm 1.

As mentioned in Section 2.1.2, SSA can only be applied to well-mixed volumes. Hence, the length of the lattice sides should not be long in order to keep

Algorithm 1: GMP Reaction-Diffusion Algorithm [30]

```
1  Initialize t_S = min{τ_Ds} for all species S
2  t_sim=0 , n_S=1 for all S
3  while t_sim < t_end do
4      while t_sim < t_S do
5          Reaction μ in τ_R on every lattice site
6          Advance simulation time t_sim = t_sim + τ_R
7      end
8      Diffuse species for which t_S = t_sim
9      Increment iteration n_S for the diffused species
10     t_S = min{τ_Ds * n_S} for all S
11 end
```

homogenization of lattice volume [1]. In the simulator, lattice length is determined by user from run script. If the length of lattice size is not assigned, the simulator uses default lattice size, which is 100 nm.

The diffusion and reaction events in the molecular channel are entirely independent from communication of nodes. Even though there is not any communication between nanonodes, the reaction-diffusion events are performed in the background. In fact, molecular channel has individual commands to start and stop channel simulation as depicted in Chapter 3.

As expected, this independency is unilateral in the simulator. The communication between nodes completely depends on reaction-diffusion events. The propagation of information molecules and reception of the carrier molecules are the crops of reaction-diffusion events. Therefore, nodes cannot communicate with each other unless reaction-diffusion events are performed by the molecular channel.

2.2 Molecular Motor-Based Communication

Motor-based communication is inspired from intra-cell communication of real biological cells. The organelles in the cell usually communicate with each other by using molecular motor-based communication. Molecular motor is a protein complex that converts chemical energy, i.e., ATP, into kinematic energy. Molecular motor can carry the information between transmitter and receiver nanomachines by using this

[1]The analysis of optimal lattice size is beyond the scope of this work. An analysis of optimal lattice size is presented in [30].

kinematic energy. It transports elements, i.e., DNA, vesicle, along the biological filaments called microtubules. This mechanism is called as active transport [28].

As mentioned, motor-based communication is composed of molecular motors, i.e., kinesins, dyneins, and myosins, and their consistent protein filaments called as molecular rails, i.e., microtubules and actin filaments. Molecular motors may flow along protein filaments or protein filaments glide over immobilized molecular motors. In [3], a molecular propagation system in which carrier molecules glide over fixed molecular motors is proposed as shown in Figure 2.1. In the proposed system, loading and unloading operations are performed with no effect or a little stimulation. Single stranded DNA (ssDNA) is utilized to load and unload information. DNA hybridization occurs during the loading and unloading of information molecules on sender and receiver sides. Information molecule is attached to long ssDNA on sender side. When carrier molecule with short ssDNA, which is partial complementary of information molecule's ssDNA, moves near the information molecule, information molecule is attached to carrier molecule by DNA hybridization. The information molecule glides over fixed molecular motors with carrier molecule. Information molecule is unloaded when carrier molecule passes over a long ssDNA which is the complementary of information molecule's ssDNA. As a yield of this propagation system, information molecule is replaced to receiver side.

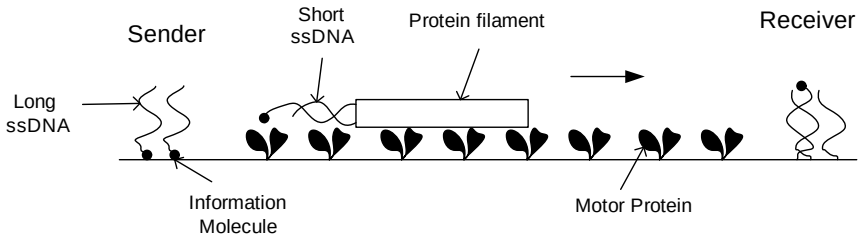

Figure 2.1: An overview of propagation system in which protein filaments glides over immobilized molecular motors [2].

As molecular motors walk along the microtubules, they are not affected by disturbance molecules which cause Brownian motion. This provides high percentage of

successful transmission between sender and receiver nanomachines. Transportation is faster and more efficient than diffusive communication. Because of the reasons given above, motor-based communication model can be imagined as wire communication in traditional communication.

Microtubules have instable behavior in the nature. Protein filaments change their lengths. The instability depends on the concentration of tubulin, i.e., the structural unit of microtubules, and on the presence of MAP activity. The instable growing and shrinking features of microtubules are taken into account in the proposed system of [18].

Transmitter node encodes the information into molecules. DNA, protein, peptides can be given as examples of carrier molecules. In [2], [29], carrier molecules are inserted into a vesicle embedded with channel proteins to transport the molecules. Vesicles, which encapsulate the information molecules, behave like communication interfaces between sender and receiver. They protect carrier molecules from environmental conditions and maintain compatibility between information molecules and environment. When vesicle contacts with sender nanomachine, hemi-channel gap junctions are constructed between sender and vesicle. Molecules are transferred from sender to the vesicle via free diffusion.

After molecules are transferred to vesicle, gap junctions between sender and vesicle are closed. Vesicle binds to molecular motor and molecular motor starts to move along rails according to direction of the filament as shown in Figure 2.2. At the end of propagation, when vesicle contacts with the receiver nanomachine, hemi-channel gap junctions are built up. Information molecules are transferred via free diffusion from vesicle to receiver nanomachine.

Additionally, in [29], an addressing mechanism is proposed. The destination address is assigned by one ssDNA sequence. Receiver nanomachines have the complementary ssDNA sequence. Thus, receiver molecule accepts the carrier molecule if received ssDNA meshes with possessed DNA sequence.

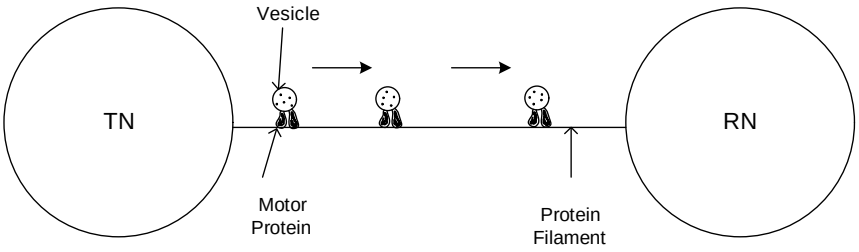

Figure 2.2: An overview of propagation system in which molecular motors flow along protein filament.

2.3 Gap Junction-Based Communication

Gap junction-based cell to cell communication model, inspired from intercellular communication in nature, is a sort of short range communication between nanomachines. In this model, signaling occurs through gap junction which is the physical contact between the neighbor cells. Transmitter nanomachine initializes signaling by stimulating the adjacent cell. Stimulated nanomachine broadcasts carrier molecules. The neighbor nanonodes receive the signal and also broadcast the carrier molecules. By this way, each node triggers the adjacent node. Thus, communication between sender and receiver nanomachines is realized as illustrated in Figure 2.3.

Ca^{2+} is a universal second messenger, which is responsible for relaying information from receptors on the surface of biological structure to molecules inside the cell. Crucial cellular activities like fertilization, contraction, secretion, are performed via calcium signaling [15]. The studies about the communication via gap junctions designate Ca^{2+} as carrier molecule in their proposed systems, like assigned in nature. Ca^{2+} spikes are utilized to encode information. For example, cells can use some modulation techniques similar to ones used in radio broadcast, e.g., AM and FM, to encode information [6]. Sending of Ca^{2+} waves is determined by encoding process. Ca^{2+} waves propagate through gap junctions via free diffusion. The amplification of Ca^{2+} spikes is possible with Calcium Induced Calcium Release (CICR)

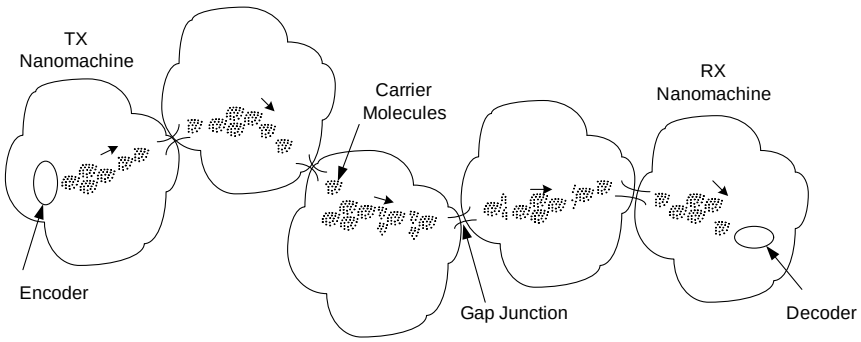

Figure 2.3: Signal propagation in gap junction-based molecular communication system.

to increase the propagation distance of calcium waves [12]. After the propagation process, finally, destination nanomachine receives the calcium waves from neighbor nanomachine and decodes the signal.

Gap junction, which connects cytoplasm of two adjacent cells, can be constructed with different types of connexin proteins. Normally, gap junctions allow the molecules which are smaller than 1000Da to pass. The permeability and selectivity of gap junctions vary based on the constructive proteins of the gap junction [50]. Besides, miscellaneous internal and external factors including cytosolic Ca^{2+} concentration, transjunctional and transmembrane potential, connexin phosphorylation, electromagnetic fields, temperature and pH level in the environment alter the permeability and selectivity of gap junction [9]. These capabilities of gap junction allow nanomachine to apply filtering, switching and signal aggregation functionalities [22].

Since gap junctions may have different permeability and selectivity, nanomachine can filter the coming carrier molecules as packet filtering in traditional network. The permeability and selectivity of gap junction can be adjusted with various external stimulations as mentioned above. Nanomachine can allow or block the carrier molecules to pass through the gap junctions. By this way, nanomachine gains switching functionality. In addition to these, signal aggregation is enabled with per-

meability and selectivity features of gap junction.

In this thesis, diffusive and motor-based molecular communication paradigms are modeled and developed among the ones explained in this chapter. The implementation details of diffusive and motor-based communication simulation frameworks adopted in NanoSim are presented in Chapter 3 and 4, respectively.

CHAPTER 3

SIMULATION OF DIFFUSIVE MOLECULAR COMMUNICATION

In this chapter, the extensions which provide the simulation of diffusive molecular communication in ns-2 are described. The extensions are inspired from diffusive molecular communication model given in Section 2.1. This diffusive molecular model is principally aimed to study networking aspects of diffusive molecular systems and to provide background for promising improvement on diffusive molecular communication. This chapter covers the internals of nanonode, network components that are used to construct the network stack for a nanonode and constituents that compose diffusive molecular communication system.

3.1 Overview of Diffusive Molecular Communication Model

Diffusive molecular communication simulator provides a mechanism for nanomachines to communicate over a short distance via propagation of molecules in aqueous medium. The main components of molecular communication systems consist of sender nanomachines, receiver nanomachines, carrier molecules and the environment that these operate in. The classes of these actors have to be designed in order to develop this simulator.

First of all, sender and receiver nanomachines are required to be designed. In order to do this, we propose *NanoNode* object to simulate nanomachines. The sender nanonode sends the molecules to receiver nanonodes by releasing carrier molecules into the environment. This case brings the necessity of another actor. We have to cre-

ate a class that has the features of carrier molecules such as proteins, ions or DNA. Information is encoded into *Molecule* objects. Carrier molecules diffuse in the environment. To simulate this, we need a mechanism which provides diffusion process. We choose diffusion model proposed in [31]. In this study, environment is divided into lattice sides. Thus, we design *Lattice* class for lattice sides. Since we divide the entire medium into lattice sides, we need to reallocate every object according to lattices. We propose a coordinate system, $MolPosition$, the unit element of which is lattice. Owing to diffusion, molecules walk randomly from one lattice side to another one. We design $Diffusion$ and *Randomizer* classes in order to accomplish this requirement.

When carrier molecules reach receiver nanonode, the receiver nanomachine should detect and receive the carrier molecules by binding them. Binding operation is a chemical process. In order to construct apprehend mechanism, reaction channel objects that define chemical reactions and reaction object which is able to apply SSA are required. Therefore, we design *Reaction* and *ReactionChannel* classes.

A class which provides shared medium for nanonode and manages propagation and binding mechanism is also required for the simulator. Hence, we design diffusive molecular channel, $MolChannelDiffusion$. We propose $MolecularEnvironment$ class to keep all data about molecular communication channel like lattice sides, reaction channels and species. $MolChannelDiffusion$ class accesses entire environment data through a pointer attribute of MolecularEnvironment class.

3.2 Network Components in Diffusive Communication System

The network stack for a nanonode consists of a molecular link layer, MAC layer, physical layer and channel which contain diffusion and reaction objects. These network components are created and plumbed together in OTcl. The relevant NanoNode method is $add-interface()$, which is coded in ./ns/tcl/lib/ns-mol.tcl. The plumbing of components creates the network stack in Figure 3.1.

Main components of molecular network interface are briefly described below:

- **MolLL:** The link layer of molecular communication. It only receives and

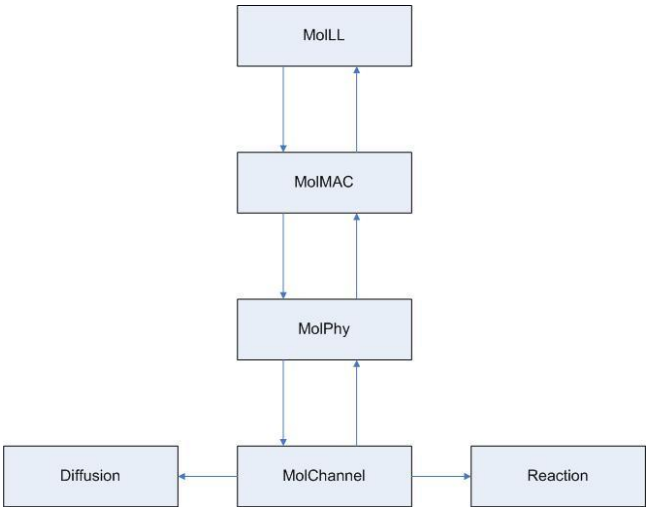

Figure 3.1: Main components of a molecular network interface.

forwards MolecularData. If needed, user can define delay for this layer. The corresponding class name for TclObject name space is LL/Mol.

- **MolMAC:** MolMAC provides connection between link layer and physical layer. MAC layer is supposed to follow a particular MAC protocol to provide multiple accesses for molecular channel. Currently, one type of MAC protocol, explicated in Section 3.4, is implemented for molecular communication. The corresponding class name for TclObject name space is MAC/Mol.

- **MolPhy:** One type of physical layer is defined. This layer passes information down (release molecules) and up (construct molecular data) according to direction molecular data. The corresponding class name for TclObject name space is Phy/Mol.

- **MolChannel:** Two molecular channel models are defined. One of them is developed for diffusive communication, while the other one is developed for motor-based communication. This layer provides releasing, propagation and binding of molecules. Diffusion and reaction objects are attached to this module. The corresponding class name for TclObject name space is $Channel/Mol$.

- **Diffusion:** One type of diffusion model is considered. This module provides diffusion functionality for simulator. The corresponding class name for TclObject name space is $Diffusion$.

- **Reaction:** Three types of reaction models are considered. These modules provide reaction functionality for simulator. The corresponding class name for TclObject name space is *Reaction*.

3.3 Implementation of Diffusive Molecular Communication

In this section, we focus on some of the key components in the implementation and take a detailed look at each component of diffusive molecular communication system in ns-2. The implementation overview of molecular communication ns-2 simulation framework is depicted in Figure 3.2. The structures of nanonode and user interface commands are built up in OTcl programming language; on the other hand, almost all of the mechanisms are implemented in C++. The interfaces of the classes implemented in C++ are given in Appendix B.

3.3.1 NanoNode: Constituent of Molecular Communication

The basic structure of molecular communication is NanoNode in ns-2. The class NanoNode is derived from Node class. Node class is a standalone class in OTcl [37]. Almost all the components of Node class are TCLObjects. NanoNode is a split object which has some additional functionalities like releasing-binding carrier molecules on molecular channel that simulate aqueous environment and *volume occupation* on environment. The volume occupation expresses occupying lattices according to the coordinates and the radius of the nanomachine. In other words, volume occupation means creating nanonode in the lattice space. Nanonode creation is performed at the beginning of the simulation when simulation time is zero. Nanomachines are immobilized in the simulation in other saying, nanonodes do not move. All functionalities of NanoNode are implemented in C++ except the construction of NanoNode itself.

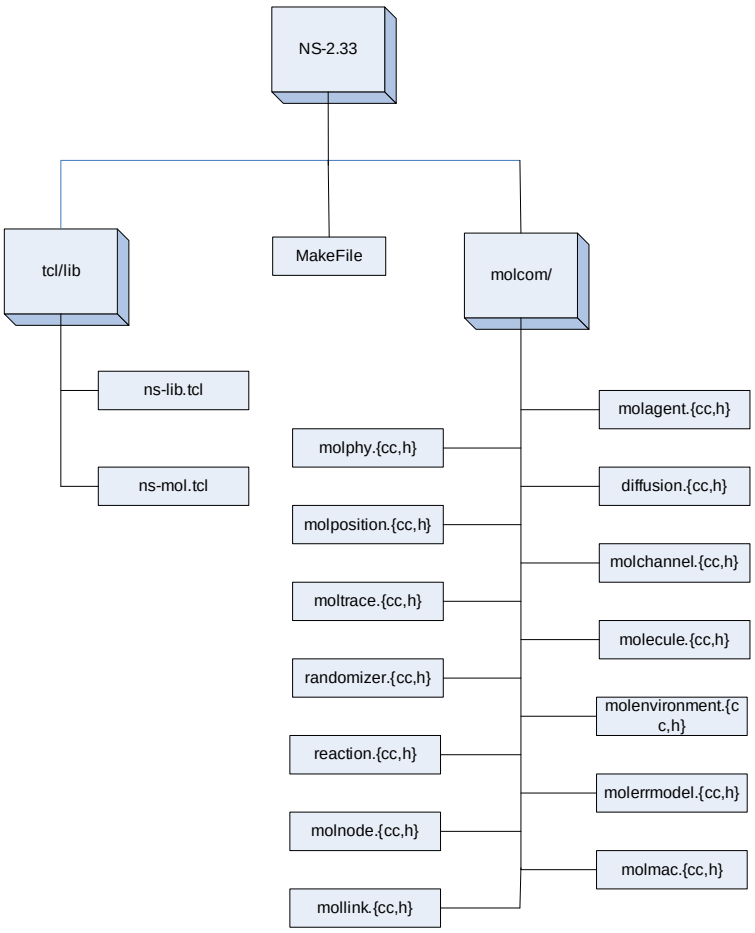

Figure 3.2: The implementation process of implemented files for NanoSim.

NanoNode has two kinds of molecule genus. One of them is *ligand* type of genus, the other one is *receptor* type of genus. The *ligand* molecules are released by the transmitter nanomachines. These molecules are carrier molecules. *Receptor* molecules are used by receiver nanomachine in order to capture *ligand* molecules only if a reaction channel between *ligand* and *receptor* molecule exists in the system. If *receptor* molecule reacts with *ligand* molecule, following reaction equation occurs:

$$RECEPTOR + LIGAND \rightarrow RECEPTOR + DATA \tag{3.1}$$

which states that if the reaction occurs, the number of *ligand* molecules decreases, whereas the number of *receptor* molecules does not change.

NanoNode is constructed with two kinds of lattices, i.e., *cytosol* and *membrane*. *Cytosol* lattices simulate the core of nanonode which can be expressed as $VIVO$, whereas *membrane* lattices simulate the membrane of the nanomachine. *Membrane* lattices contain *receptor* molecules, thus, the *ligand* molecules are captured in this type of lattices. In other words, the carrier molecules' trading occurs in this type of lattices.

Nanomachines are also built up in the lattice space. The shape of nanomachines is considered as sphere. However, spatial discretization of lattice space makes it impossible to create a perfect sphere in the simulator. As a result, we have to discretize the nanomachine.

Let's assume that we have a function that calculates the distance between the membrane lattice (x_m, y_m, z_m) and the central lattice (x_c, y_c, z_c) :

$$D(x_m, y_m, z_m) = \sqrt{(x_c - x_m)^2 + (y_c - y_m)^2 + (z_c - z_m)^2} \tag{3.2}$$

We can determine the membrane lattice with the following equation:

$$D(x_m, y_m, z_m) \leq R < D(x_m, y_m, z_m \pm 1) \tag{3.3}$$

where (x_m, y_m, z_m) is the coordinate of membrane lattice, R is the radius of the nanomachine defined by user. The lattices between *membrane* lattice and central lattice are *cytosol* lattices. Figure 3.3 depicts the 2D representation of a nananomachine.

There are some differences between the structure of NanoNode and Node in ns-2. Like all ns-2 nodes, the NanoNode has an *entry* point, nevertheless its entry point does not designate to regular classifiers different from the other nodes in ns-2. Molecular routing agent, which is developed for future implementations, only checks the direction of MolecularData and orients it in the current implementation of NanoSim. If incoming MolecularData is destined to this node, it is directed to $Src/Sink\ Agent$. Besides, we cancel port classifier, $demux\_$, which exists in base *Node* class, since there is no equivalent mechanism in current molecular communication models. The structure of NanoNode object is shown in Figure 3.4.

Molecules leaving the network stack are sent to the node's entry. As mentioned before, entry point is connected to routing agent. When routing agent receives a

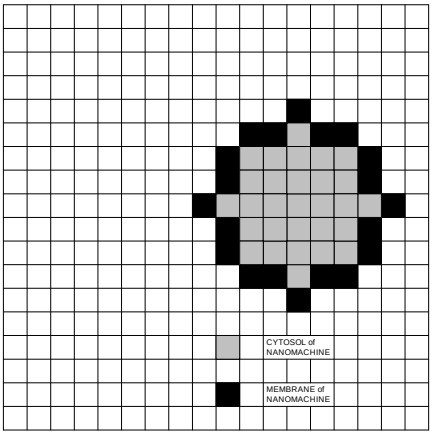

Figure 3.3: 2D slices representation of a nanomachine the radius of which is 4.

MolecularData, it checks the direction of it. If the direction field indicates down, it sends molecules to link the layer. Otherwise, routing agent transfers MolecularData to the sink.

In NanoNode class, there are pointers which show additional objects. Each NanoNode object contains a position object that points the centre of itself, a ligand and receptor molecules pointers. It also keeps the radius of nanonode as a static variable and the number of receptor molecules of nanonode.

Likewise, there are some pointers which point the network components in NanoNode. Each NanoNode contains a pointer to channel (channel_) and its physical layer (phy_).

Besides, NanoNode contains a static integer variable, *type_*, to keep the type of the molecular communication system. The type of communication channel system is distributed from this variable to the other classes in the simulator. The type of the system is assigned by the user via Tcl script.

3.3.2 MolPosition: Molecular Communication Coordinate System

This class inherits from TclObject class in order to take parameters via Tcl scripts. As a consequence of GMP algorithm, molecular communication environment is divided into cubic lattice sides. Cartesian coordinate system is used in our

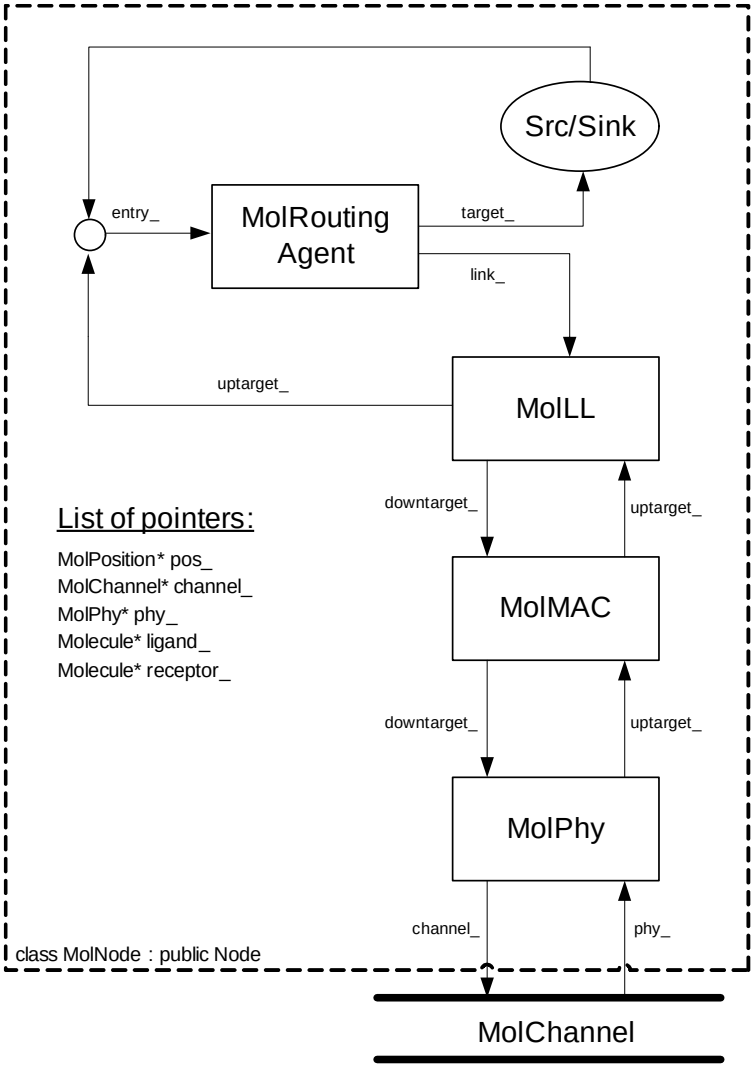

Figure 3.4: Schematic of a NanoNode.

simulator. The unit of coordination system is the lattice, the size of which is *lambda*. The coordinate entries and output of simulator are in units of lattice. For example, a nanonode is created and its position is designated as:

```
set n0 [$ns node];
$n0 set-position 10.0 20.0 30.0
```

the actual position of nanonode is equal to $(10 \times lambda, 20 \times lambda, 30 \times lambda)$ in real coordinate system. The unit of lambda is nm in the simulator.

In most cases, access to member variable is restricted to the compile time;

nevertheless, in ns-2 it is possible to change it by binding the member variable with the corresponding Tcl variable in the constructor of member owner class. Variable binding feature of ns-2 is applied to *lambda* variable. *lambda*, which is an attribute of MolecularEnvironment class, is bound with Tcl variable, i.e., *lambda_*. The value of lambda can be adjusted via Tcl run script without compiling the simulator as mentioned in Section 1.2.

3.3.3 MolecularData: Packet Unit of Molecular Communication

Packet class instances are the fundamental exchange units between objects in the ns-2 simulation [37]. Since, all *send*() and *recv*() methods of classes, derived from TclObject, accept Packet pointer, MolecularData is derived from Packet class. MolecularData, which requires to be inherited from Event class, passes through the network structures and provides the communication between network components. Its building structure is MolecularDataUnit objects. Information is encoded into the number of molecules. MolecularData contains MolecularDataUnit objects in order to keep the number of molecules. MolecularData class retains MolecularDataUnit in linked list form as figured in Figure 3.5. Only the head and the tail of linked list exist in MolecularData class.

MolecularDataUnit object keeps the pointer of molecule and the amount of the molecule. In brief, MolecularData object keeps MolecularDataUnit instances and similarly MolecularDataUnit object keeps Molecule instances. The class diagram of molecular data is given in Figure 3.6.

Molecule is the constructive structure of molecular communication. It is the most physically realistic fundamental unit of molecular communication in the simulator. In NanoSim, information is encoded in terms of molecule concentration as given in [17]. Therefore, individual molecules do not carry any information.

We assume that ligand molecules in the simulator are sphere. Thus, according to $Stokes - Einstein$ formula, diffusion coefficient of a ligand molecule is given as follows

$$D = \frac{kT}{6\pi r\eta} \tag{3.4}$$

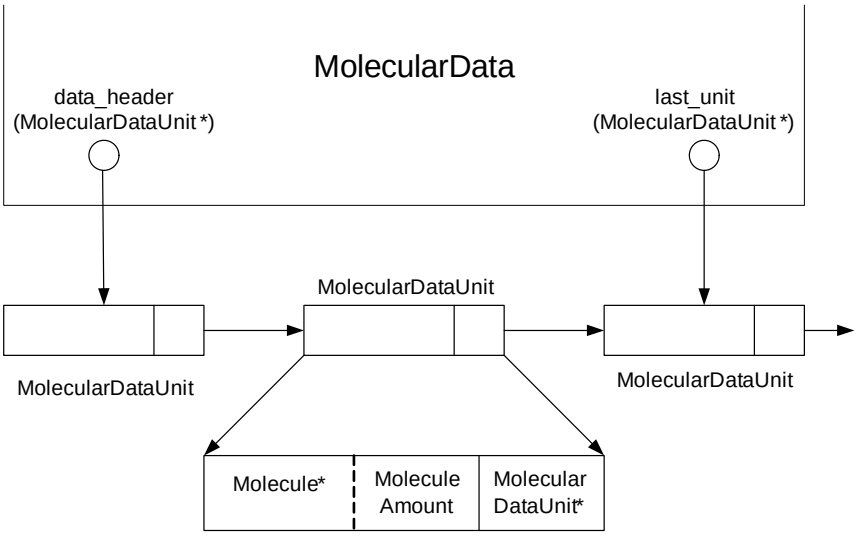

Figure 3.5: MolecularDataUnit linked link structure.

where k is Boltzmann constant, T is temperature of medium, r is radius of sphere and η is the viscosity of the aqueous medium. The derivation of this formula is available in [23].

Operator is able to define ligand and receptor molecules for a nanonode. Also, the radius of a molecule can be defined for each molecule. The diffusion coefficient of a molecule is estimated as given in (3.4) inside *getDiffusionCoefficient()* method of Molecule object. Since each molecule type has a unique diffusion coefficient, every molecule has a specific time step, which is the time interval between two consequent diffusion events of each molecule, according to (2.14).

3.3.4 Diffusion: Propagation System

Diffusion class inherits from TclObject class. Diffusion inherits from TclObject since the type of diffusion object is determined by user via *diffusion* parameter in node configuration interface. Currently, only one type of Diffusion model is implemented; nevertheless, another diffusion model can be implemented and joined into the simulator.

Diffusion class is also derived from Handler class. As explained before, Han-

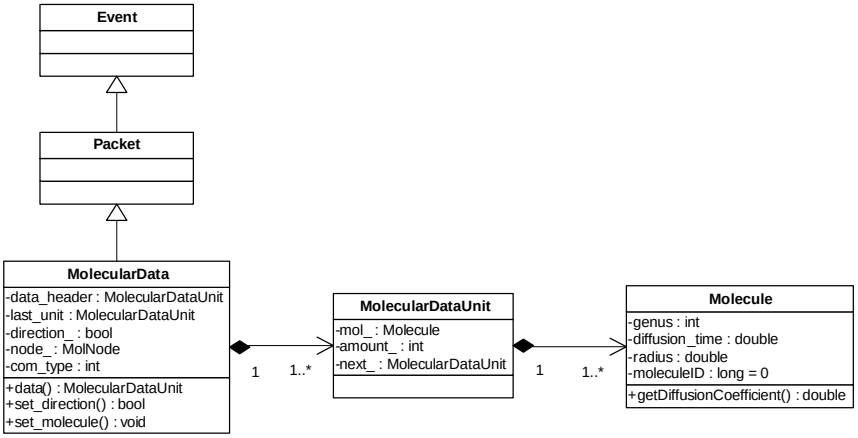

Figure 3.6: Class diagram of molecular data.

dler is an abstract class. When an event is ready, the handle method of Handler derived class, $Diffusion$, is called. In handle function of diffusion class, diffusion event is triggered. Diffusion class has an interface only with MolChannel class. The minimum interface with other components brings the modularity to diffusion object.

The basis of this diffusion processes is the multi particle LGA algorithm [30]. In this algorithm, medium is divided into lattices. Each lattice site holds a discrete number of uniformly distributed particles [31]. As a result of diffusion, molecules perform random walk on the lattices. Molecules are distributed to neighbor lattices randomly. The exact position of a molecule is not necessary, only the lattice position of the molecule is needed. Every species has particular diffusion coefficient. Diffusion time of each species is calculated with lattice length and diffusion coefficient. If the number of molecules existing in the lattice is less than 60 [30], the molecules move individually to neighbor lattice. If the number of molecules is larger than 60, molecules are moved in a bulk to lattice according to Gaussian distribution.

As mentioned before, *membrane* lattices act like a wall. Molecules coming from outside the nanomachine cannot enter *cytosol* lattices. To simulate *membrane* lattices like a wall, reflective boundary conditions are used. If molecules try to diffuse *cytosol* lattices, these molecules reflect from *cytosol* lattices and stay in their local lattice.

Every lattice has two different attributes in order to hold type and number of the molecules. One of them keeps the current state, which keeps the actual molecule number in the lattice; while the other one keeps the temporary state, which keeps the molecule numbers in diffusion state. During diffusion process, the molecules in current state are distributed to temporary states of neighbor lattice sides. At the end of diffusion process, temporary state is transferred to current state and reset. An example that depicts the state of lattices is given in Figure 3.7.

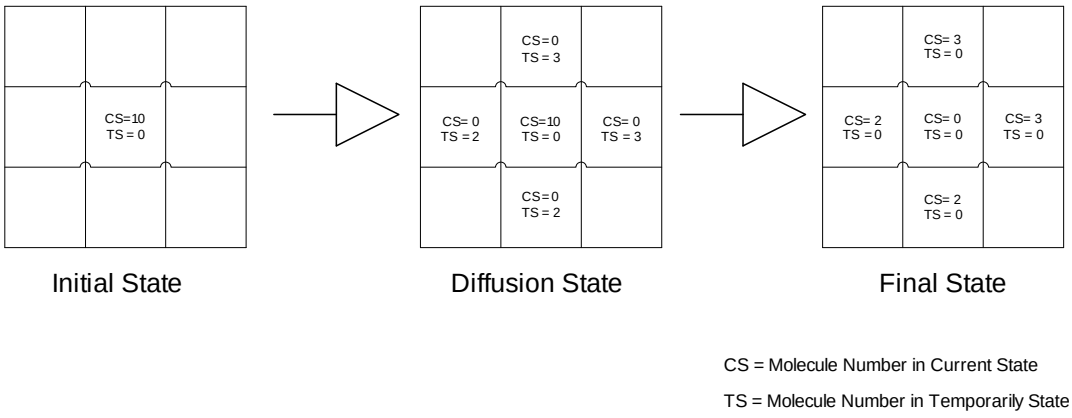

Figure 3.7: Descripton of states in diffusion process in 2D.

3.3.5 Reaction: Capture Mechanism

Reaction class is derived from TclObject and Handler classes like Diffusion object. Three types of reaction models, i.e. *NoReaction*, *Berg*, *Gillespie*, can be selected via Tcl script. In NoReaction option, ligand molecule is captured when it touches receiver nanomachine. Berg choice implements the reaction model given in [54]. The foundation of Gillespie selection is obtained from SSA as described in Section 2.1.3. The class diagram of reaction model is depicted in Figure 3.8.

The identity element of reaction mechanism is reaction channel. Reaction-Channel class presents the chemical reaction channel. Chemical reaction is defined by user via $set - reaction$ method of molecular channel. In the simulator, only second order chemical reactions can be defined. The types of molecules and diffusion

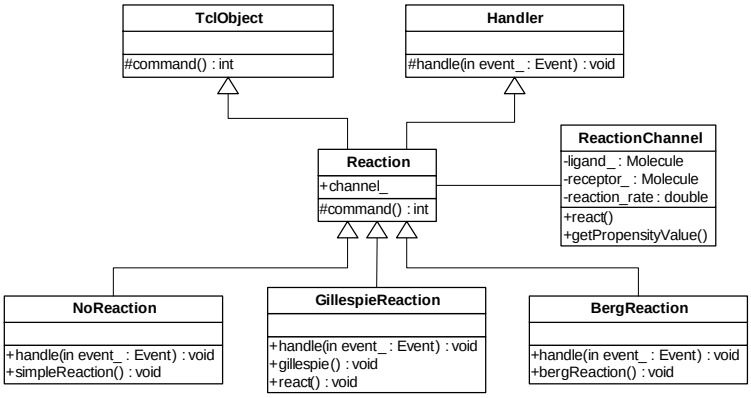

Figure 3.8: Class diagram of reaction model.

coefficient rates define a reaction channel. An example of reaction channel definition is given below:

```
$channel set-reaction A B 10.1
```

SSA is elaborately described in Section 2.1.2. Some adaptations and assumptions are added to SSA. These adaptation and assumptions are;

- We have a limited time interval to apply the algorithm,
- If a reaction occurs in a *vitro* type lattice, both of input molecules will decrease,
- If a reaction occurs in a *membrane* type lattice, the number of receptor molecule will not change; only ligand molecule will decrease,
- There cannot be a chemical reaction inside *cytosol* lattice.

After the adaptations and assumptions are applied to SSA, Algorithm 2 is achieved.

3.3.6 MolChannelDiffusion: Challenge of Molecular Communication

The challenging issue of molecular communication simulator is the modeling of molecular channel. Molecular channel is totally different from the traditional one as defined in Table 2.1. The difference between molecular communication and

Algorithm 2: SSA for NanoSim

```
1  t_delta is the input time interval
2  t_sim=0
3  forall the existence lattices except (lattice type == CYTOSOL) do
4      while t_sim < t_delta do
5          Calculate propensity function for every species a_μ = h_μ c_μ
6          if propensity function == 0 then
7              break
8          end
9          Generate two random variable r_1, r_2
10         Calculate τ , τ = (1/a_0) ln (1/r_1)
11         if t_sim + τ > t_delta then
12             break
13         end
14         Estimate μ , Σ_{υ=1}^{μ-1} < r_2 a_0 ≤ Σ_{υ=1}^{μ}
15         t_sim = t_sim + τ
16         if lattice type == VITRO then
17             Decrease input molecules
18         end
19         else if lattice type == MEMBRANE then
20             Decrease only ligand molecule
21         end
22     end
23 end
```

traditional communication in propagation style and propagation medium brings the necessity to develop a new communication channel for molecular communication.

MolChannel class is derived from Channel class. In this thesis, we model two different types of molecular communication schemes. Hence, we need two types of different molecular channels. Actually, the characteristics of both molecular channels are almost the same. Hence, we develop two types of molecular channel classes which inherit from MolChannel. These classes are MolChannelDiffusion and MolChannelMotor. Figure 3.9 depicts class diagram of molecular channels. According to *molChannelType* value given in the node configuration interface, molecular communication type of simulator is designated.

MolChannel accesses to environmental data through a pointer to MolecularEnvironment class. MolecularEnvironment class keeps all the data about the molecular channel like a repository of simulator. Simulator environment contains nanomachines, species, reaction channels and lattices.

Lattices are the members of MolecularEnvironment class which keeps the en-

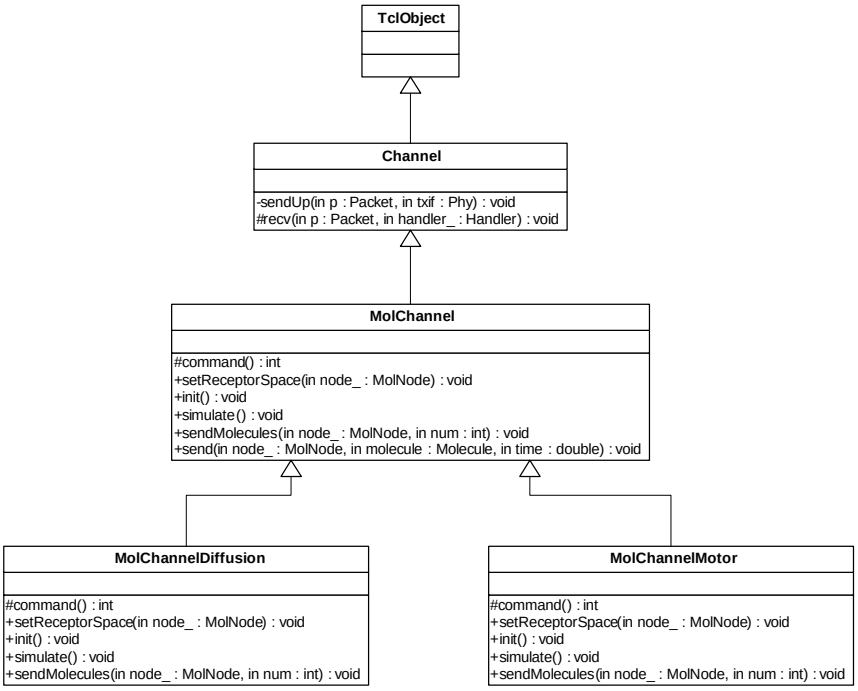

Figure 3.9: Class diagram of molecular channel.

tire lattice medium. There are three kinds of lattices for diffusive molecular communication. These lattice types are *cytosol, membrane, vitro*. As mentioned before, nanonodes are constructed with *membrane* and *cytosol* lattices. The lattices in which propagation of molecules occurs are expressed as *vitro* type lattices. *Vitro* type lattices behave like outside of nanonodes. Molecules cannot diffuse to *cytosol* lattices from *membrane* and *vitro* lattices. Lattice side is supposed to keep the molecules. Lattice class fulfills this requirement with two maps, which keep the molecule type and the number of molecules. One of them holds the current state, the other keeps the temporary state of the lattice.

Actually, data flow in the simulator occurs in the molecular environment. Because of that, molecular environment is created only once in ns-lib.tcl and linked to molecular channel which is also alone in the simulator. The lattice space is retained with following type definition:

`typedef std::map<` *int*, *Lattice*$*$ `> latticeMap;`

`typedef std::map<` *int*, *latticeMap* `> latticeMapMap;`

`typedef std::map<` *int*, *latticeMapMap* `> latticeMapMapMap;`

Three maps are put one inside the other in the type definition of *latticeMapMapMap*. It behaves like a 3D matrix, each element of which points a *Lattice* object. This structure of NanoSim has two major advantages. First the boundaries of matrix can be determined during program execution. Second, if there is no molecule inside a lattice, the lattice is deleted from matrix. By this way, the performance of the simulator is increased and huge amount of memory is saved.

In addition to keeping lattices and lattices space, MolecularEnvironment class retains species and reaction channels in vectors. Other classes can reach and use these data in any time of the simulation. It also keeps environmental attributes, which affect the result of the simulation, i.e., viscosity, temperature and lattice side length.

In diffusive molecular communication, molecular channel keeps all nanonodes in the simulator. Diffusive molecular channel is associated with diffusion and reaction classes and "has a" relationship with molecular environment class as depicted in Figure 3.10. In order to obtain modularity in the simulator, interface between components is kept at minimum as much as possible.

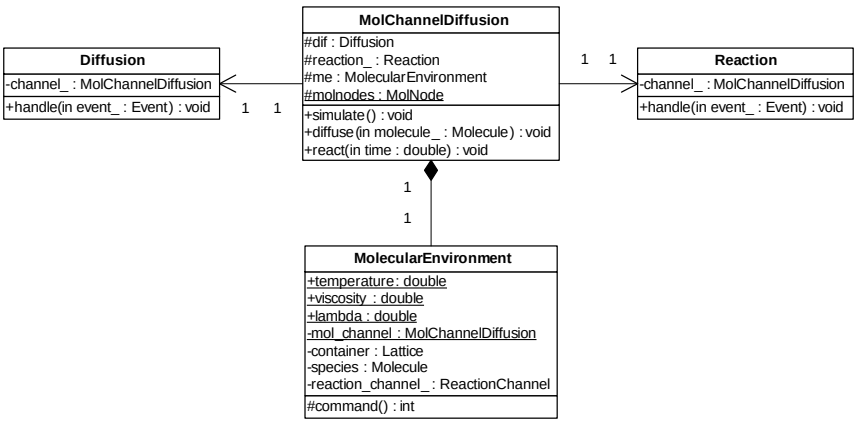

Figure 3.10: Class diagram of diffusive molecular channel.

Molecular channel also provides a shared medium for all nanonodes to com-

municate with each other. The diffusion and reaction events in the molecular channel are entirely independent from the communication of nodes. Although they are independent from communication of nodes, communication between nodes are shaped by these events. Molecules, which can be expressed as information, are transported and captured by these events.

3.4 Molecular MAC Protocol

We devise and incorporate a basic molecular MAC protocol (MolMAC) in order to serve as a model for the future MAC protocol implementation. Each transmitter uses the shared channel in its own time slot. Our MAC implementation is inspired from this behavior of simple Time Division Multiple Access (TDMA) which is a channel access protocol for shared medium networks. Additionally, the system given in [17] is taken as reference. In this system, if the captured molecules in that time slot are more than the prescribed value, transmitted information is assumed as logic 1, otherwise logic 0. Hence, MolecularData, filled with the total number of captured molecules in that time slot, has to be sent to upper layers at the end of each time slot.

Three types of time handler classes are used in this MAC implementation. One of them is used to organize time slots. The other time handler class which handles sending of MolecularData checks whether it is its nanonode turn or not. When the time slot is its nanonode turn, it calls *recv()* method of physical layer. The last time handler class deals with capturing molecules in each time slot. Actually, MolMAC receives the MolecularData immediately, when a ligand molecule is captured. MolMAC adds the received moleculardata to *repository_* which is a MolecularData type variable of MolMAC class. At the end of the time slot, a copy of *repository_* is sent to link layer and then *repository_* is reset for the next time slot.

In our MAC implementation, there is a preamble time slot. We assume that nanonodes inform the other nodes whether they send data in the forthcoming frame or not. In the protocol, each nanonode gives release order in a sequence. The releasing sequence between nanonodes is a random event. A nanonode cannot start

releasing until the current NanoNode finishes its time slot. The time slot of a nanonode the interval value of which is defined by the user is very important in order to avoid interference of molecules. Because, if time slot is too short, the receiver nanomachine cannot capture desired molecules from the channel. If it is too long, the receiver can receive noise molecules which are released by another nanonode or released in one of the previous interval.

3.5 Detailed Look at Diffusive Molecular Communication Links

In this section, we describe how MolecularData moves up and down the stack, and the key points to note at each layer. Here, the composite structure of diffusive molecular communication links as a *network stack*. The file ./ns/tcl/lib/ns-mol.tcl contains the various OTcl instprocs that assemble links. Figure 3.11 provides a detailed look at how molecular communication links are composed.

MolecularData leaving a node passes to the class MolLL. If an outgoing error model is defined for NanoNode, MolecularData passes to molecular error model. After error model is applied to MolecularData, the *recv()* method of MolMAC class is called. Unless an outgoing error model is declared, MolLL object sends MolecularData to MolMAC object. Next, the packet is sent to MolPhy.

MolPhy class inherits from Phy class. As usual, MolPhy is responsible for providing connection between MolMAC and molecular channel. In addition to this, MolPhy is in charge of releasing molecules to molecular channel and capturing the molecules from molecular channel. This object just releases the packet to the attached channel. When MolecularData is received from MolMAC layer, it is extracted. Molecules and the number of molecules are obtained. When MolPhy receives a MolecularData instance, it checks the direction of MolecularData. If its direction is down, it releases the molecules by calling *sendMolecules()* of molecular channel and calls *recv()* method of molecular channel. In *sendMolecules()* of diffusive molecular channel, molecules are put into *membrane* lattices of the nanonode. The release of molecules is the beginning of its own time slot.

The outgoing molecules are finally sent to MolChannelDiffusion. As described

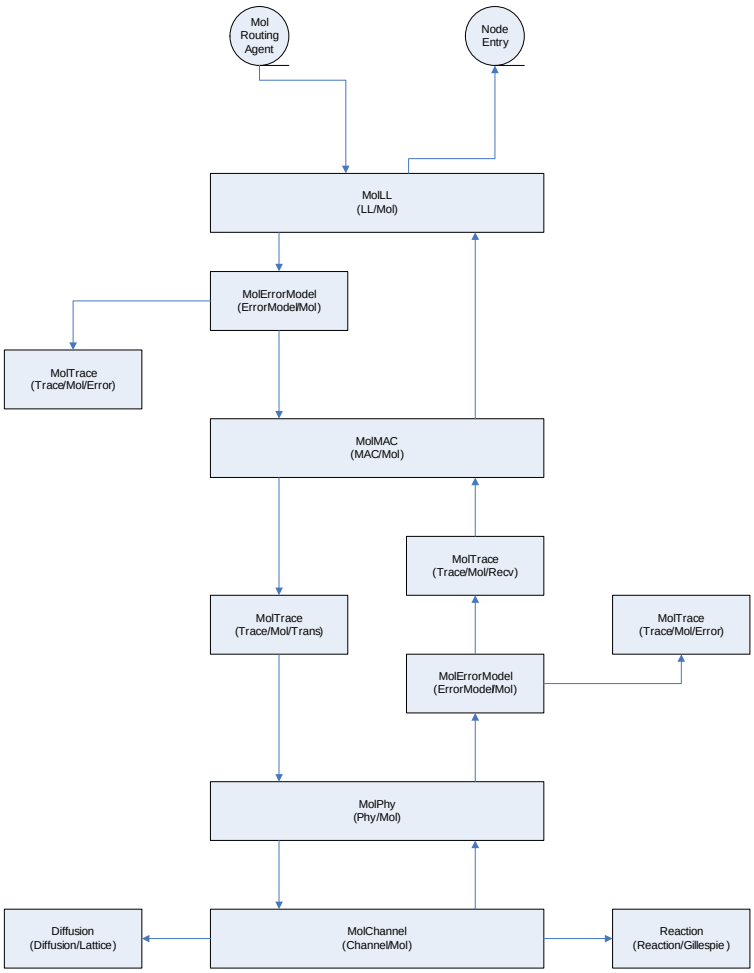

Figure 3.11: Detailed look at network interface stack.

in previous sections, diffusion and reaction processes are independent from sending and receiving cycle. It can be said that the received molecules are inserted into reaction-diffusion cycle. When a molecule is captured, MolChannel creates MolecularData using the information of captured molecule. The direction of MolecularData is set to up and this MolecularData is sent to physical layer of receiver nanonode.

When *recv()* method of MolPhy is called, MolPhy checks the direction of MolecularData, and then it decides what to do. If the direction is up, it fills the *data* field of MolecularData with the type and number of captured molecules in that interval. Then MolPhy sends MolecularData to MolMAC if no incoming error model

is declared. Otherwise, MolecularData is sent to molecular error model object. If the MolecularData arrives safely at the MAC, it next passes to MolLL object. Finally, the packet moves to the entry of the NanoNode.

3.6 Molecular Trace Support

There are several ways to collect trace data on a simulation. Generally, trace data is either displayed directly during execution of the simulation, or more commonly, stored in a file to be post-processed and analyzed. In the current ns-2 simulator, there is a *Trace* class to monitor capabilities. A trace records each individual event as it arrives, departs, or is dropped at a link or queue. Trace objects are configured into a simulation as nodes in the network topology. They are inherited from *Connector* class. Thus, they can be easily plugged into network stack.

Trace files used for molecular communication have the same structure as conventional ns-2 tracing. Class *MolTrace* derives from class Trace. Molecular trace objects are plumbed into network stack if they are enabled in Tcl script. Although the structure trace mechanism is similar with conventional ns-2 tracing, their formats are totally different, due to the communication carrier and channel structure incompatibility. Hence, the information to be traced is changed. The format of molecular trace is given in Table 3.1.

There are three types of molecular traces. These are transmitter (Trace/Mol/Trans), receiver (Trace/Mol/Recv) and error traces (Trace/Mol/Error). Transmitter ones are utilized to trace sent molecules. The type of this trace is *t*. Since the receiver of molecular trace is not known, the receiver node id, receptor molecule and position of receiver nanonode fields are set to "-1", "?" and "-999,-999,-999", respectively. Receiver traces are utilized to trace accepted molecules. The type of this trace is *r*. When molecules are received, MolecularDataUnit instances are constructed and then MolecularData is created by them. If the number of possible transmitter node of a received molecule is more than one, the sender node id and position of sender nanonode fields are set to "-1" and "-999,-999,-999", respectively. Error traces are utilized to trace molecules which are corrupted by error model. The type of this trace

Table 3.1: The format of molecular trace.

1	time	2	s	3	d	4	lig	5	rec	6	sp	7	dp	8	a	9	cp	10

Description Of Fields	
[1]	Type of the trace
[2]	Time of the trace
[3]	No of the sender NanoNode
[4]	No of destination NanoNode
[5]	Type of the ligand molecule
[6]	Type of the receptor molecule
[7]	Molecular position of sender NanoNode
[8]	Molecular position of destination NanoNode
[9]	Number of captured molecules
[10]	Molecular position of the lattice where the ligand molecule is captured

is *e*. The error traced molecules are dropped due to errors defined by the error model.

To enable tracing of all molecular links in the simulator, use the following commands before instantiating nodes and links:

```
set f [open out.tr w]
$ns trace-all $f
```

3.7 Molecular Error Model

MolErrorModel class inherits from ErrorModel class. These error models, causing molecules to be corrupted according to various probability distributions, are simple and do not necessarily correspond to any experience on an actual molecular channel. Each nanonode can insert a given statistical error model either over outgoing or incoming molecular channels. Precisely, the instantiated error model is stuck between mac and phy modules for incoming link and between link layer and mac modules for outgoing link as depicted in Figure 3.11. For the outgoing link,

the error module would be pointed by *downtarget_* of the above link layer module while for the incoming link, it would be linked by *up-target* pointer of the below phy module. Thus, the *target_* of the error module points to mac layer in both cases.

In molecular error model, every molecule is checked according to rate and distribution of random variable. If no error is obtained from error model for a molecule, molecule can continue its way. Otherwise, it is directed to NULL agent. If tracing facility is enabled, information about error molecules is written into trace file.

Molecular error model can be applied to molecular data in molecule or MolecularData level. This implies that error models can check erroneous cases for every molecule or MolecularData. This capability is determined by user with *unit* command. If *molecule* option is chosen as the unit of error model, each molecule is inspected whether it is corrupted or not. Otherwise, each MolecularData is examined.

The reason of placing error models, i.e., incoming and outgoing, over two different locations is that the outgoing error model causes all the receivers to receive the packet suffering the same degree of errors, since the erroneous situation is determined before *phy* releases the molecules to molecular channel. On the other hand, the incoming error module lets each receiver get the molecules corrupted with different degree of error, since the error is computed independently in each error module.

The following code provides an example of how to add an error model to an incoming link:

```
set em_ [new ErrorModel/Mol]
$em_ set rate_ 0.02
$em_ ranvar [new RandomVariable/Exponential]
$n1 interface-incoming-errormodel $em_
```

The following code provides an example of how to add an error model to an outgoing link:

```
set em1_ [new ErrorModel/Mol]
$em1_ set rate_ 0.2
$em1_ ranvar [new RandomVariable/Uniform]
```

```
$n1 interface-outgoing-errormodel $em1_
```

3.8 Commands at a Glance

The commands for diffusive molecular communication part of NanoSim are given in Appendix A.

CHAPTER 4

SIMULATION OF MOTOR-BASED MOLECULAR COMMUNICATION

In this chapter, the extensions that provide the simulation of motor-based molecular communication in ns-2 are described. The extensions are inspired from molecular motor-based molecular communication model given in Section 2.2. This chapter contains the components that construct motor-based molecular communication system. Motor-based molecular communication part of NanoSim is designed and implemented in order to serve as a model for the future extensions of NanoSim.

4.1 Overview of Motor-Based Molecular Communication Model

This part of the simulator is based on motor-based molecular communication. A motor-based communication model is designed according to communication system which is introduced in Section 2.2. The class of molecular motor-based molecular communication system consists of sender nanomachines, receiver nanomachines, carrier molecules and the environment in which these operate as diffusive one. The network components of both diffusive and motor-based communication systems are the same except molecular channel.

In our envisaged model, nanonodes communicate with each other through vesicle communication interface via molecular motors. Sender nanonode transfers information molecules into vesicle. Carrier molecules diffuse into vesicles through gap junctions. Vesicle with molecular motor is released next to microtubule. Vesicle makes free diffusion near the microtubule. There are two possibilities in this case.

It may be attached to protein filament or diffuse without touching molecular rail and perish. If it is attached to molecular rail, it walks through molecular rail with constant velocity [21]. Afterwards, vesicle connects to receiver nanonode through gap junction. Information molecules diffuse from vesicle to receiver nanonode through gap junction. Finally, vesicle is disconnected when receiving interval is over.

The components of both diffusive and motor-based communication systems are almost the same. There are few differences in molecular channel structure. As mentioned before, diffusive molecular communication refers to wireless communication, whereas motor-based molecular communication refers to wired communication [45]. In motor-based communication, protein filaments are considered as wired connection. In addition to components utilized in diffusive communication, molecular link is designed. Besides, there are components which are not used in motor-based communication, even though they are used in diffusive communication. For example, there is no need for reaction mechanism in motor-based communication as Reaction and ReactionChannel classes are not used in the motor-based one.

It seems that there is no necessity for molecular channel since the connection between nanonodes is established by molecular link. However, a molecular channel which can manage diffusion process and keep environmental data is required. Owing to this necessity, we design motor-based molecular channel, MolChannelMotor, which inherits from MolChannel class.

4.2 Network Components in Motor-Based Communication System

From network perspective, there is a major difference between diffusive and motor-based molecular communications. We replace molecular channel which provides a shared medium for nanonodes with communication link which supports node to node communication. Molecular link is created and inserted between nanonodes in OTcl. The network stack for motor-based molecular communication is given in Figure 4.1.

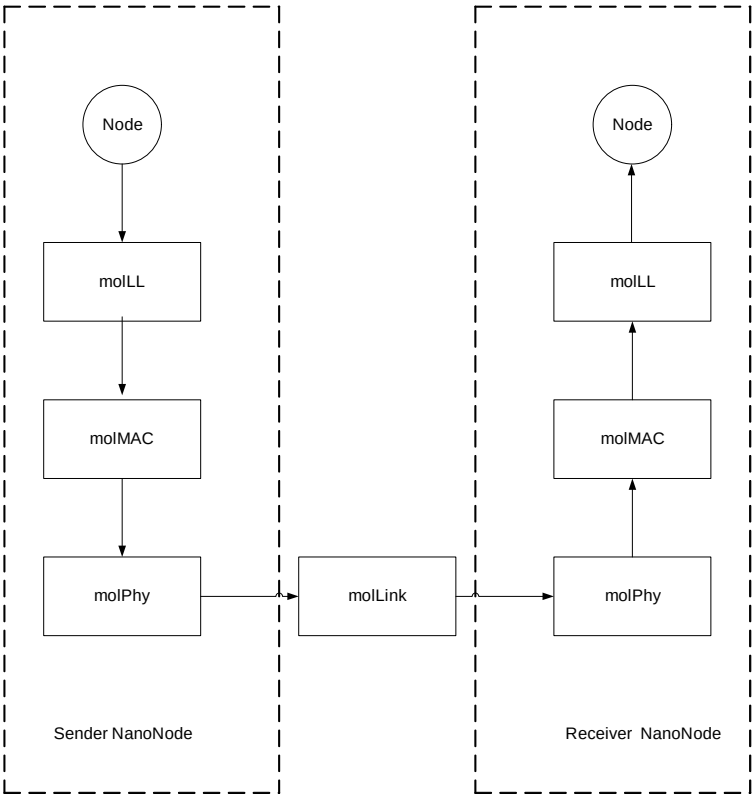

Figure 4.1: Network components of molecular motor-based molecular communication.

4.3 Implementation of Motor-Based Molecular Communication

In this section, we focus on some of the key components of the implementation and take a detailed look at each component of motor-based molecular communication system in ns-2. Since almost the same network components are used for both systems, the implementation details of same components are not described in this section. Please refer to Section 3.3 for implementation details of these components. In the rest of this section, MolLink and MolChannelMotor, new components, are introduced.

4.3.1 MolLink: Link Between NanoNodes

The MolLink class is implemented in order to establish molecular link between nanonodes. Molecular link is inspired from microtubules which have normally unstable behavior in the nature. In our model, we assume that connection between nanonodes is established by user command and shows a stable attitude.

As mentioned in Section 2.2, the propagation in microtubule is in one direction. Connection is established only from sender nanonode to receiver nanonode and the propagation in the opposite direction is not possible. If the propagation in opposite direction is required, another molecular link should be established between nanonodes.

The functionality of molecular link can be divided into 3 phases. These phases are diffusion of information from sender nanonode to vesicle, propagation of vesicle and diffusion of information molecules from vesicle to receiver nanonode. The phases are clearly depicted in Figure 4.2.

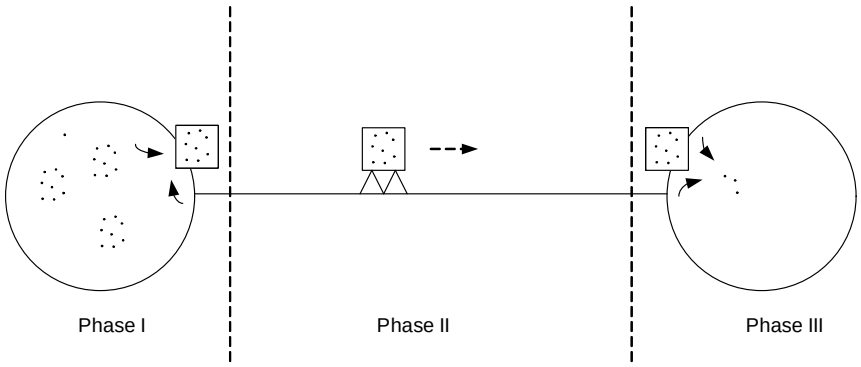

Figure 4.2: Phases of motor-based molecular communication.

In phase-1, information molecules propagate from sender nanonode to vesicle. The propagation occurs as a result of free diffusion. Vesicle is modeled as a lattice side in the simulator. We define a new type of lattice called *vesicle*. In this phase, when *recv*() method of MolecularLink is called by *MolPhy*, the type of a lattice next to *cytosol* lattice is changed to *vesicle*. The molecules which have already diffused between *cytosol* lattices begin to diffuse *vesicle* lattice. The diffusion of molecules

between *cytosol* lattices and *vesicle* continues until MolLink passes to phase-2. Period of phase-1 is designed as a constant value and designated by user.

When phase-2 starts, the type of a lattice which is next to *cytosol* on receiver nanonode side is changed to *reserve* and the molecules in *vesicle* lattice are transferred to *reserve* lattice. Then, *vesicle* lattice is reset and its type is changed to *membrane*. By this way, the connection between sender and vesicle is closed.

In second phase, vesicle is transported from sender nanonode to receiver nanonode molecular motor. We assume that molecular motor propagates with constant velocity. The velocity of molecular motor is defined by user. Time period of phase-2 is calculated as follows

$$t_{prog} = \frac{D_{memb}}{V_{motor}} \tag{4.1}$$

where the distance between two nanonode membrane, D_{memb}, is equal to

$$D_{memb} = D_{centre} - 2 \times R \tag{4.2}$$

where D_{centre} equals the distance between two nanonodes and R is the radius of nanonode. From the distance of two points in 3D space we obtain final formulation of phase-2 period:

$$t_{prog} = \frac{\sqrt{(x_s - x_r)^2 + (y_s - y_r)^2 + (z_s - z_r)^2} - 2 \times R}{V_{motor}} \tag{4.3}$$

where (x_s, y_s, z_s) and (x_r, y_r, z_r) are the central lattices of sender nanonode and receiver nanonode respectively. After phase-2, phase-3 begins. In phase-3, information molecules propagate from vesicle to receiver nanonode. At the beginning of phase-3, the type of *reserve* lattice is changed to *vesicle*. Then, information molecules begin to diffuse from vesicle to receiver nanonode. Period of phase-3 is designed as a constant value and assigned by user.

When phase-3 is over, *vesicle* lattice is reset, and its type is changed to *membrane*. By this way, the connection between vesicle and receiver nanonode is closed. Then, MolecularData is constructed from the transferred information molecules and recv() method of MolPhy is called.

Phases mentioned above are managed by MolLink class. MolLink class contains three timer classes which are derived from *Handler* class in order to cope with

these phases. It also contains two pointers that show sender and receiver nanonodes, and needs the molecular environment in order to change the type of lattices. The class diagram of MolLink is given in Figure 4.3.

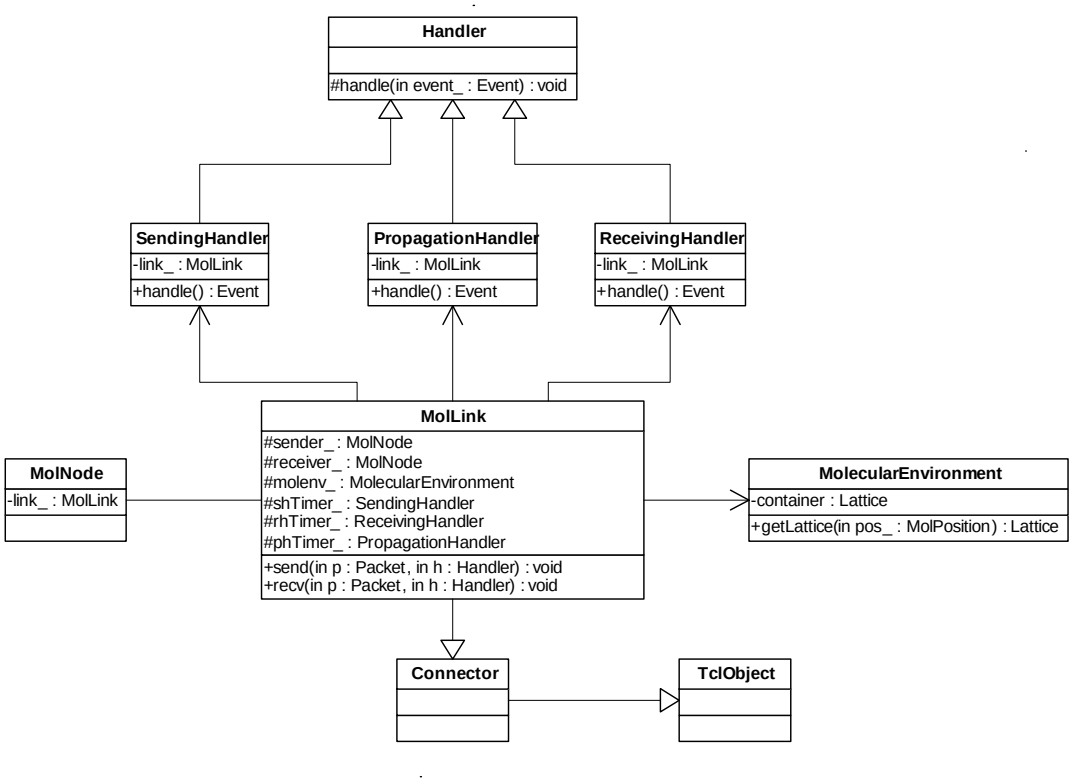

Figure 4.3: Class diagram of molecular link.

4.3.2 MolChannelMotor: Challenge Issue of Molecular Communication

As indicated in previous section, communication between nanonodes is realized through molecular links. In spite of this, our design needs molecular communication channel. In our design, communication channel manages diffusion events, and also keeps molecular environment data via *MolecularEnvironment* class.

MolChannelMotor class derives from MolChannel like MolChannelDiffusion. Motor-based communication channel has almost the same functional features as diffusive communication channel. The main difference is that there is no reaction facility in motor-based communication system. In molecular channel models, simulation

is managed in *simulation()* method. In diffusive molecular channel, diffusion and reaction events are directed in *simulation()* method, whereas only diffusion events are managed in motor-based communication channel.

4.4 Detailed Look at Motor-Based Molecular Communication Links

The same types of network components are used for diffusive and motor-based communication. MolLL, MolMAC and MolTrace objects are the common components. However, the behavior of MolPhy is different for diffusive and motor-based communications.

In motor-based communication model, when MolPhy receives MolecularData, it releases the molecules inside the NanoNode by calling *sendMolecules()* method of MolChannelMotor object. MolChannelMotor object puts molecules inside the *cytosol* lattices, whereas in diffusive communication channel they are put into *membrane* lattices. Additionally, MolPhy calls *recv()* method of molLink. Then, molLink performs the phases of molecular link. At the end of these phases, molLink constructs MolecularData with diffused molecules and sends MolecularData to MolPhy. After this point, the same way is pursued as described in Section 3.5.

4.5 Molecular Error Model

The molecular error model for motor-based communication covers diffusive communication's error model. In addition to the features of it, a molecular error model is inserted between the molecular link and the physical layer of receiver nanonode as inspired from the $hybrid - asters$ propagation approach given in [42]. In $hybrid - asters$ propagation approach, information molecules diffuse from sender nanomachine onto protein filaments in order to propagate to receiver nanomachine. While information molecule diffuses, it may touch the protein filaments and begin to move along the protein filament with constant velocity. In our implementation, we demonstrate this event with probabilistic manner. We assume that vesicle, which carries the information molecules, may not bind to protein filaments according to the probability distribution and rate, which are defined by user.

If vesicle binds to protein filament, molecular error model passes MolecularData to receiver nanonode. On the contrary, the vesicle may be considered as not binding to protein filament and canceled. Therefore, MolecularData will not pass to physical layer of receiver nanonode. There is one difference between the error models used in diffusive communication and motor-based communication. In the diffusive communication, each captured molecule is checked whether it is erroneous or not. However, we cannot realize this mechanism in motor-based one due to the fact that molecules are kept in vesicle. We have to check each vesicle to apply error model. Thus, the unit of error model should be assigned as *moldata*.

4.6 Commands at a Glance

The commands for motor-based molecular communication part of NanoSim are given in Appendix A.

CHAPTER 5

NUMERICAL ANALYSIS OF MOLECULAR COMMUNICATION

In this chapter, we present the numerical analysis of molecular communication. We designed specific scenarios that provide a feasible context to make numerical analyses. Firstly, the scenarios are introduced. After the introduction of the scenarios, they are analyzed and formulated. The numerical results of derived formulas and the results of the simulation are compared in the next section in order to validate the simulator.

5.1 The Explication of Scenarios

We designed three types of scenarios that are called as $Diffusion-Scenario$, $Berg-Scenario$ and $Gillespie-Scenario$ in the rest of the document. All of these scenarios are designed to analyze specific functionalities.

The aim of Diffusion-Scenario is to provide feasible context to analyze diffusion functionality. In Diffusion-Scenario, we cover the burst mode, in which the source emits the information molecules instantaneously into a stationary medium. A point source exists in the center of a sphere plate. On the other hand, there is a receiver nanomachine the center of which is on the surface of the outer sphere. When an information molecule contacts with receiver nanomachine, it is captured. Otherwise, if the molecule touches the outer shelter sphere, it is destroyed.

Owing to the fact that an information molecule is captured when it collides to receiver nanomachine surface, the surface of the system should be taken into account

in the analysis. However, as the concentration function is only valid for volume calculations, it is assumed that there is a shelter surrounding the surfaces. The thickness of the shelter and nanomachine, notated as d, is infinitesimal.

We include reaction functionality to the previous scenario in Berg-Scenario. Normally, reaction between ligand and receptor molecules binds the information molecules to receiver nanomachine. Because of that, not only collision is required to capture the molecule, but also reaction event is needed. Ligand molecule is bound by receptor molecules only if it collides into the binding area of receptor molecule as given in [54].

In Berg-Scenario, N receptor molecules, the binding area of which is r_B are uniformly distributed over the nanomachine surface. We assume that there is no intersection between the binding areas of receptor molecules. Moreover, if ligand molecule collides with nanomachine out of the binding area, it sticks into the membrane of the nanomachine and becomes unusable.

In Gillespie-Scenario, we model our medium to analyze SSA individually. The ligand and receptor molecules are released in a well-stirred volume, lattice, at the beginning of the simulation. When reaction occurs between ligand and receptor molecule, the number of receptor molecules stays the same, whereas the number of ligand molecules decreases and information molecule is obtained.

5.2 Numerical Analysis of Diffusion Mechanism

In this section, we make numerical analysis of Diffusion-Scenario. Suppose that Q molecules are released instantaneously from the origin of Cartesian coordinate system at $t = 0$. The spatial density of concentration in space through time is given as:

$$U(x, y, z, t) = \frac{Q}{(4\pi Dt)^{3/2}} e^{-r^2/4Dt} \tag{5.1}$$

where $r^2 = x^2 + y^2 + z^2$ and D is the diffusion constant that is explicated in (3.4).

In (5.1), the instant concentration of a volume unit is calculated. However, we need to integrate the concentration of molecules over time in order to evaluate the

captured molecules. The integration of concentration over time is given as

$$U(r,t) = \int_0^t \frac{Q}{(4\pi Dt^*)^{3/2}} e^{-\frac{r^2}{(4Dt^*)}} dt^* = -\frac{Q}{4D\pi r} erf(\frac{r}{\sqrt{4Dt^*}}) \mid_0^t = \frac{Q}{4D\pi r} erfc(\frac{r}{\sqrt{4Dt}}) \tag{5.2}$$

in which $erfc(x)$ is complementary error function. $erfc(x)$ is represented as

$$erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-x^2} dx \tag{5.3}$$

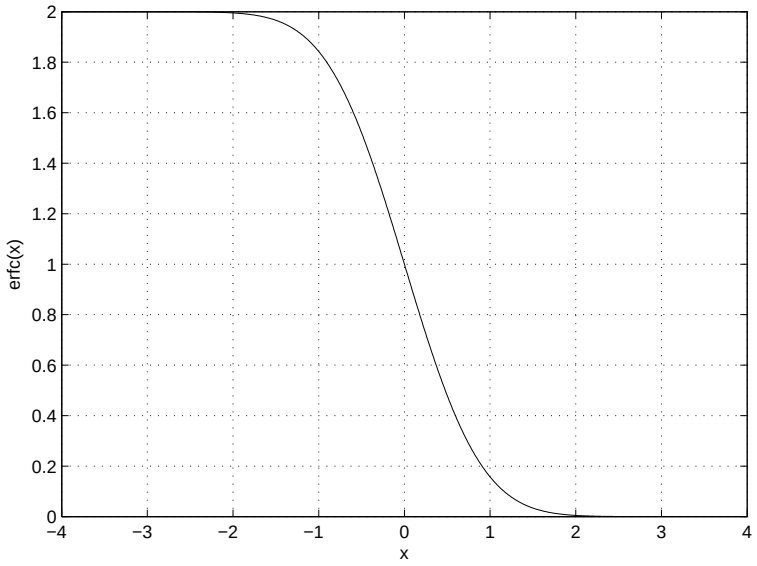

Figure 5.1: erfc(x) function.

If propagation of the molecules is observed for a long time, the value inside the $erfc()$ function approaches 0. As shown in Figure 5.1, $erfc(0)$ is equal to 1. Therefore, the density function of captured molecules approaches the limit

$$U(r) = \frac{Q}{4D\pi r} \tag{5.4}$$

In order to calculate the volume of this thin shelter, consider two spheres. The radius of first sphere is $a + d$, and the radius of other sphere which is inside of the first one is a. The subtraction of spheres' volumes gives the volume of the shelter. Since our source is a point source and the variable of (5.1) is r, the volume of sphere has to be expressed in r.

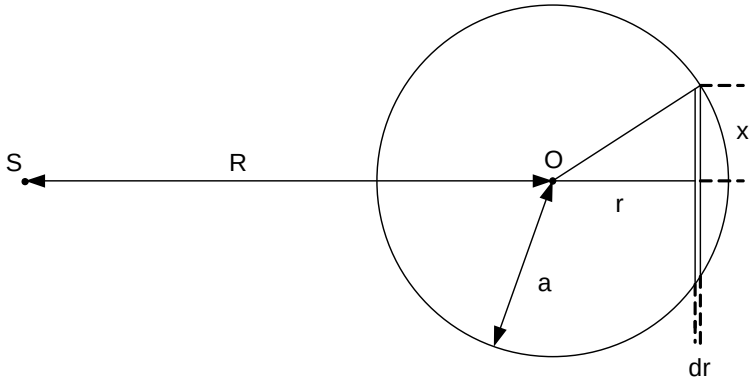

Figure 5.2: Representation of parameters used in volume calculation.

In order to calculate the volume of sphere in terms of r, the sphere is divided into small flat slices, the radius of which is $\sqrt{a^2 - r^2}$ as indicated in Figure 5.2. Then, the slices are summed up as follows

$$V = \int_{-a}^{a} \pi(a^2 - r^2)dr \tag{5.5}$$

which estimates the volume of sphere by taking the center of it as a reference point. However, our reference point is R away from the sphere center. Hence, (5.5) should be re-expressed as

$$V = \int_{R-a}^{R+a} \pi(a^2 - (r - R)^2)dr \tag{5.6}$$

The total amount of molecules existing in sphere during time t is estimated as follows

$$C(r,t) = \int_{y}^{x} \frac{Q}{4D\pi r} erfc(\frac{r}{\sqrt{4Dt}})\pi(a^2 - (r - R)^2)dr \tag{5.7}$$

where the limits of (5.7), [x,y], are determined according to the visible sides of nanomachine by point source.

As projected in Figure 5.3, point source can only view (DE) surface of nanomachine. In order to determine the limits of (5.7), we need to find l. As understood from Figure 5.3, $(\widehat{DON}) = 2 \times (\widehat{DEN}) = \alpha$. Hence,

$$\tan(\widehat{DON}) = \frac{2\tan(\widehat{DEN})}{1 - \tan^2(\widehat{DEN})} \Rightarrow \frac{2m}{n^2 - m^2} = \frac{1}{R - m} \tag{5.8}$$

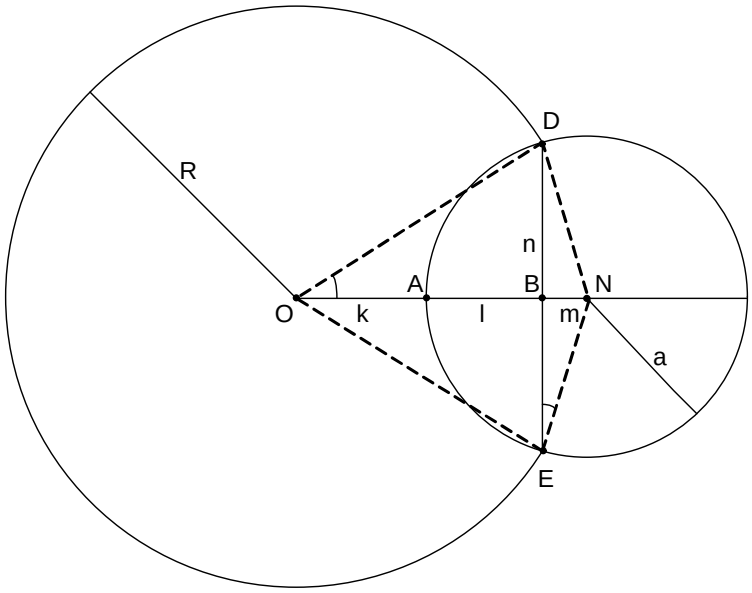

Figure 5.3: The representation used parameters of shelter sphere and nanomachine

The solution of (5.8) is

$$m = \frac{a^2}{R} \Rightarrow l = a(1 - \frac{a}{R}) \tag{5.9}$$

To find the molecules captured by thin shelter, the ones captured by bigger sphere should be subtracted from the ones captured by smaller sphere. Moreover, only visible side of sphere should be considered. Hence, the total amount molecules captured by the shelter is equal to

$$\begin{aligned} C(r,t) = \tfrac{Q}{4D}(&\int_{R-a-d}^{R-a^2/R} erfc(\tfrac{r}{\sqrt{4Dt}})\tfrac{((a+d)^2-(r-R)^2)}{r}dr \\ &- \int_{R-a}^{R-a^2/R} erfc(\tfrac{r}{\sqrt{4Dt}})\tfrac{((a)^2-(r-R)^2)}{r}dr) \end{aligned} \tag{5.10}$$

The value calculated from (5.10) is not the final result. It should be normalized with whole space which is summation of (5.10) and surrounding sphere. The molecules captured by surrounding sphere is given by

$$S(r,t) = \beta \int_{R-d}^{R} \frac{Q}{4D\pi r} erfc(\frac{r}{\sqrt{4Dt}})4\pi r^2 dr \tag{5.11}$$

in which β is the ratio between visible area from point source and entire area of surrounding sphere. Therefore, in order to find β, we need a formula that calculates

the surface of sphere in terms of r in a definite interval [52]:

$$Surface = \int_a^b 2\pi f(x)\sqrt{1+|f'(x)|^2} \tag{5.12}$$

in which $f(x) = \sqrt{a^2 - r^2}$ is applied for a sphere. Hence, β is given as

$$\beta = \frac{\int_{-a}^{a-\frac{a^2}{R}} 2\pi\sqrt{a^2-r^2}\sqrt{1+(\frac{r}{\sqrt{a^2-r^2}})^2}dr}{\int_{-a}^{a} 2\pi\sqrt{a^2-r^2}\sqrt{1+(\frac{r}{\sqrt{a^2-r^2}})^2}dr} = 1 - \frac{a}{2R} \tag{5.13}$$

After a long time period, the concentration of molecules over time approaches a limit value. The limit values should be used in the calculation of normalization factor. As a result, the normalization factor, NF, is given as

$$NF = \lim_{t\to t^*} \frac{1}{C(r,t) + S(r,t)} \tag{5.14}$$

in which t^* is the saturation time in which concentration of molecules approaches the limit value.

The multiplication of (5.10) and (5.14) provides the probability of crash of a molecule into nanomachine with respect to time:

$$P_{Col}(t) = \frac{C(r,t)}{\lim_{t\to t^*}(C(r,t) + S(r,t))} \tag{5.15}$$

The multiplication of $P_{Col}(t)$ with the the total amount of carrier molecules, Q, gives the number of collided molecules, $M(t)$, over time:

$$M(t) = P_{Col}(t)Q \tag{5.16}$$

To calculate total number of collided molecules, we need to consider that a long period of time, $t \to \infty$, passes. Hence, the probability of collision of a molecule after a long period of time is

$$P^*_{Col} = \frac{C(r,\infty)}{C(r,\infty) + S(r,\infty)} \tag{5.17}$$

As given in (5.4), when $t \to \infty$, the $erfc$ function in the equation approaches 1. After the $erfc$ functions are replaced with 1, the solution of P^*_{Col} is as follows

$$P^*_{Col} = 1 - \frac{4R(1-\frac{a}{2R})}{(R+a) + 4R(1-\frac{a}{2R}) + 2a\ln(\frac{R-\frac{a^2}{2R}}{R-a-d}) + \frac{(a^2-R^2)}{d}\ln(\frac{R-a}{R-a-d})} \tag{5.18}$$

The total number of collided molecules is given as

$$M^*_{Col} = Q(1 - \frac{4R(1-\frac{a}{2R})}{(R+a) + 4R(1-\frac{a}{2R}) + 2a\ln(\frac{R-\frac{a^2}{2R}}{R-a-d}) + \frac{(a^2-R^2)}{d}\ln(\frac{R-a}{R-a-d})}) \quad (5.19)$$

(5.19) gives the total number collided molecules in terms of R, a and d. Although diffusion coefficient, D, is one of a term that effects the diffusion mechanism as given in (5.1), it has no impact on the total number of collided molecules. The numerical results obtained in this section are compared with experimental data in Section 6.1.

5.3 Numerical Analysis of Berg Reaction Mechanism

In this section, we make numerical analysis of Berg-Scenario which includes reaction capability in addition to Diffusion-Scenario. Thus, we can utilize the derived formulas in diffusion mechanism analysis. As given in Section 5.1, receiver nanomachine contains N receptors on its surface and the ligand molecules are captured if they are collided into the binding area of a receptor molecule.

The ratio between the summation of binding areas of receptors and the surface of nanomachine gives the probability of reaction. Due to the assumption that there is not any intersection between the binding areas of receptor molecules, the summation of binding areas equals the receptor number, N, times one receptor binding area. As a result, the probability of reaction is given as

$$P_{React} = \frac{\sum binding\ areas}{nanomachine\ surface} = \frac{N\pi r_B^2}{4\pi a^2} = \frac{N r_B^2}{4a^2} \quad (5.20)$$

As mentioned above, we should consider reaction probability while analyzing capturing event. Since collision and reaction events are independent, the probability of capturing event equals the multiplication of these independent events. Therefore, the capturing probability is as follows

$$P_{Cap} = P_{Col} \times P_{React} = (\frac{N r_B^2}{4a^2})\frac{C(r,t)}{\lim_{t\to t^*}(C(r,t) + S(r,t))} \quad (5.21)$$

in which t^* is the saturation time. Finally, the total number of captured molecule is given as

$$M^*_{Cap} = Q(\frac{N r_B^2}{4a^2})(1 - \frac{4R(1-\frac{a}{2R})}{(R+a) + 4R(1-\frac{a}{2R}) + 2a\ln(\frac{R-\frac{a^2}{2R}}{R-a-d}) + \frac{(a^2-R^2)}{d}\ln(\frac{R-a}{R-a-d})}) \quad (5.22)$$

in which, it is understood that the number of captured molecules depends on the surface concentration of the receptor molecules on receiver nanomachine. The formulas derived in this section are used for validation and performance evaluation of this mechanism of NanoSim in Section 6.2.1.

5.4 Numerical Analysis of Gillespie Reaction Mechanism

The goal this section is to introduce theoretical decomposition of SSA implementation of our reaction system in NanoSim. In this section, we use Gillespie-Scenario in the analysis. As implemented in the simulator, ligand and receptor molecules react with each other considering

$$L + R \xrightarrow{k} M + R \tag{5.23}$$

in which L is ligand molecules, R is receptor molecule, M is captured molecule and k is the reaction rate.

We analyze SSA by using classical deterministic approach. The chemical deterministic description of the system is given by following ODE

$$\frac{\partial C_L(t)}{\partial t} = kC_L(t)C_R \tag{5.24}$$

where $C_L(t)$ is the concentration of ligand molecules and C_R is the concentration of receptor molecules. As it can be understood from the notation of ligand and receptor concentrations, the concentration of ligand molecules changes towards time, whereas the number of receptor molecules stays the same.

In (5.23), the variables are in terms of concentration. However, the input of analysis is in terms of the number of ligand and receptor molecules. Therefore, we need to transform the concentration values into the number of molecules. The relation between concentration and number of molecule can be expressed as

$$C = \frac{n}{NV} \tag{5.25}$$

in which n is the number of molecules, N is Avogadro constant and V is the volume in unit of *liter*. Hence, the solution of (5.23) is given as

$$n_L(t) = n_L(0)e^{-\frac{kn_R t}{NV}} \tag{5.26}$$

where $n_L(0)$ is the initial number of ligand molecules.

We can figure out the characteristics of captured molecules number, $n_M(t)$, from $n_L(t)$. Since the ligand molecules are transformed into captured molecules, we can state that captured molecule number is the complementary of ligand molecule number. The summation of ligand and captured molecules always equals the initial number of ligand molecules. Hence, the number of captured molecules can be expressed as follows

$$n_M(t) = n_L(0)(1 - e^{-\frac{kn_R t}{NV}}) \tag{5.27}$$

According to (5.27), the number of captured molecules increases as reaction rate, ligand and receptor number go up, whereas it decreases as volume increases. (5.27) is utilized in the validation of SSA in Section 6.2.2.

CHAPTER 6

VALIDATION OF NANOSIM FRAMEWORK

In this chapter, we present the validation and performance evaluation of NanoSim by comparing the outputs of NanoSim and the results of our numerical analysis which are derived in Chapter 5. The scenarios, which are described in Section 5.1, are used as simulation scenarios in this chapter.

A test version of NanoSim is implemented for the performance evaluation and validation of the simulator. Some features, e.g., point source, outer shelter, are added to the simulator and test version of simulator is achieved. At start-up of simulations, Q message molecules are released from the point source. During the propagation of molecules, some of them are captured by receiver nanomachine. The captured molecules are logged by trace mechanism of NanoSim, i.e., $MolTrace$. Then, the output file of NanoSim, *out.tr*, is processed by unix scripts. After out.tr is refined, data files which contain the captured molecules number and the summation of captured molecules number with respect to time are obtained.

The derived formulas, given in Chapter 5, are implemented in MATLAB, and numerical analysis results are plotted using MATLAB. Moreover, data files that are obtained from NanoSim are processed and plotted by means of MATLAB.

NanoSim is run with different parameters to observe different features. Since diffusion and reaction events are stochastic processes, the results of simulations are not exactly the same. Therefore, we make at least 10 trials for each simulation case. We take the average of the output data of trials by using MATLAB. However, there

is a tricky part in taking the average of NanoSim outputs. The capturing events do not occur exactly at the same time. Because of that, we divide the simulation time into periods, and take the average of number of molecules in each period.

We can categorize the main features of the simulator into two, i.e., diffusion and reaction. Firstly, validation and performance evaluation of the simulator for these functionalities are presented. Then, utilization rate and error analysis of NanoSim are given in the rest of this chapter.

6.1 Validation of Diffusion Functionality

In this section, we validate and evaluate the performance of diffusion functionality through captured molecules, diffusion coefficient and NanoSim scale.

6.1.1 Validation by Captured Molecules

In this section, we evaluate the performance of the simulator according to the number of captured molecules by NanoSim with time. We analyze the behavior of captured molecules with respect to the ratio between nanomachine radius and the shelter radius, a/R, and the amount of released molecules.

6.1.1.1 Captured molecules according to a/R

In this section, we analyze the behavior of captured molecules with respect to the ratio, a/R. We fix the simulation parameters except a and change a every 0.1 steps in the analysis. The parameters used in the simulation are given in Table 6.1. The simulator results and numerical analyses for each ratio, which are between 0.2 and 0.9, are shown in Figures 6.1, 6.2, 6.3 and 6.4.

As mentioned before, our simulator has lattice structure. Because of that, it behaves like in a discrete space. Therefore, the integral step is equal to the lattice length, $0.1\mu m$, as depicted in Table 6.1.

As depicted in Figures 6.1 – 6.4, although numerical analyses and NanoSim outputs exhibit similar behavior for each a/R ratio, they are not exactly matched. In fact, this mismatch is an expected result due to the assumptions and constraints in

Table 6.1: Simulation parameters to validate diffusion capability by captured molecules with respect to a/R.

Symbol (Unit)	***Definition***	***Quantity***
Q	amount of molecules	1000
R (μm)	radius of shelter	1
a (μm)	radius of nanomachine	$0.x$
D ($\mu m^2/s$)	diffusion coefficient	2.197
s (μm)	integral step	0.1

numerical analysis. The significant point of the behavior of NanoSim and numerical analysis is the number of captured molecules in steady state. In Figures 6.1 – 6.4, it is observed that the results of NanoSim reach the saturation time faster than numerical analysis results. As explained in forthcoming sections, the curves and the saturation

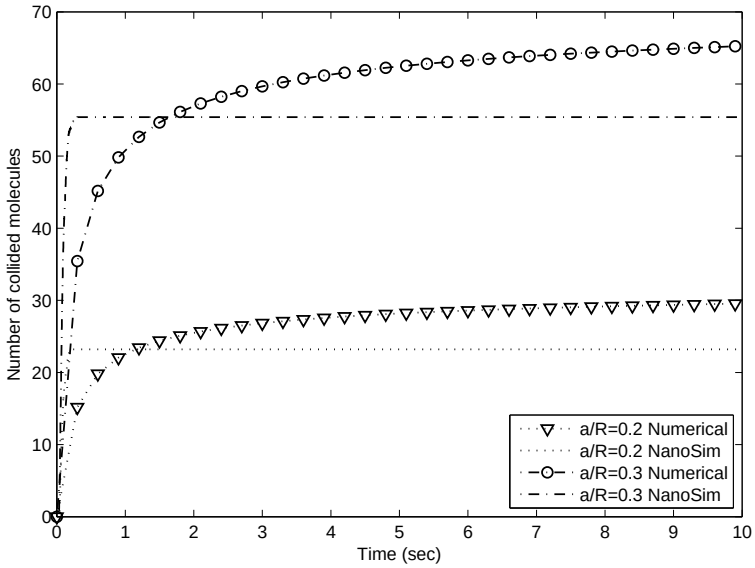

Figure 6.1: Number of collided molecules for a/R=0.2, a/R=0.3.

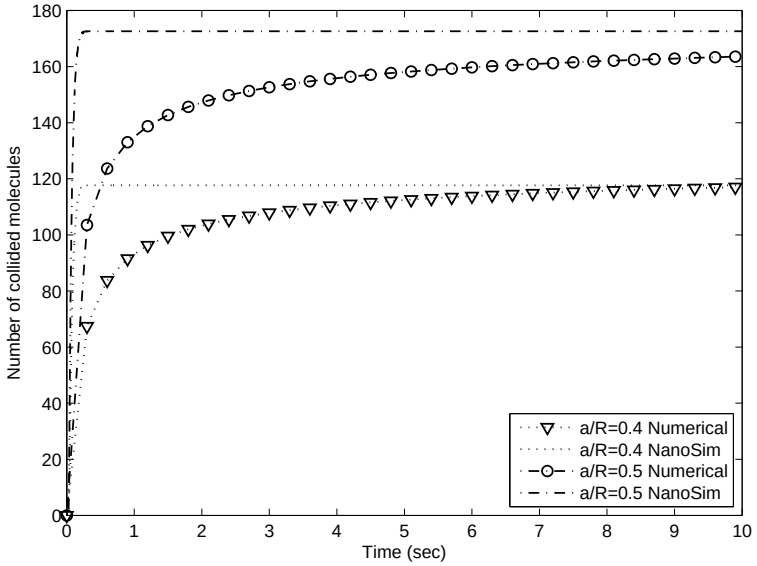

Figure 6.2: Number of collided molecules for a/R=0.4, a/R=0.5.

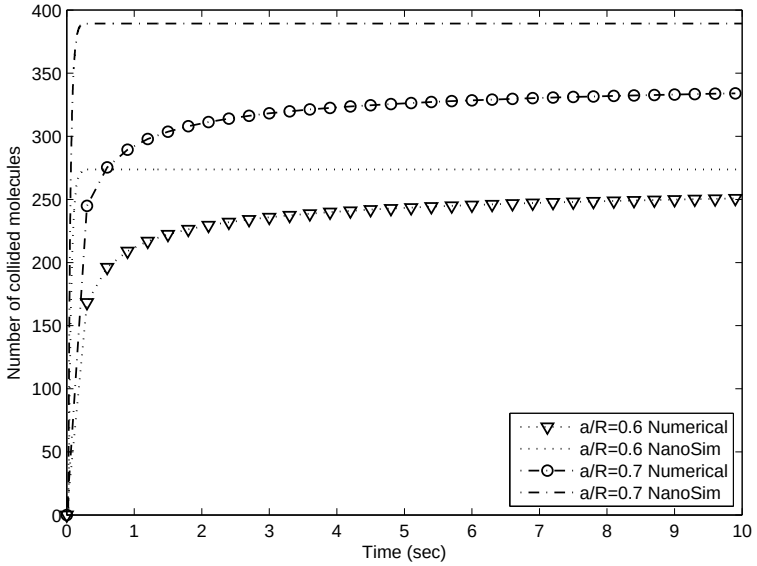

Figure 6.3: Number of collided molecules for a/R=0.6, a/R=0.7.

times of the simulations do not affect the total number of collided molecules and depend on the speed of the carrier molecules, D. The reason of this mismatch is the lattice structure of the environment. Moreover, NanoSim is implemented in Carte-

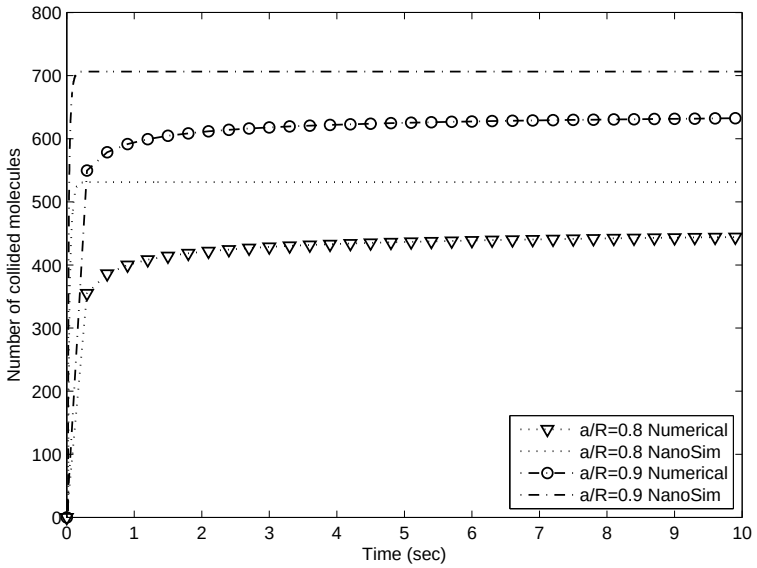

Figure 6.4: Number of collided molecules for a/R=0.8, a/R=0.9.

sian coordinate system as described in Section 3.3.1. In Cartesian coordinate system, there always exists discretization error while constructing a sphere, i.e., NanoNode. However, we do not use Cartesian coordinate system in our numerical analysis. The validation of NanoSim framework based on the comparison of numerical and experimental results is given in Section 6.3.

6.1.1.2 Captured molecules according to Q

We have analyzed the behavior of captured molecules according to a/R value, up to now. In this section, we analyze the attitude of captured molecules with respect to the amount of released molecules, Q. In the analysis, only Q is changed while the other simulation parameters are fixed as given in Table 6.2.

In Figures 6.5 and 6.6, although NanoSim results quickly reach the saturation time, it is observed that the simulator results and numerical analyses of various Q's show similar behavior. Besides, the normalized manners of simulation results and numerical analysis, which are normalized with Q, are presented in Figure 6.7. Here, the normalized simulation results for various Q show similar behavior.

Table 6.2: Simulation parameters to validate diffusion capability by captured molecules with respect to Q.

	Q	a (μm)	R (μm)	D ($\mu m^2/s$)
Trial 1	100	0.5	1	2.197
Trial 2	200	0.5	1	2.197
Trial 3	500	0.5	1	2.197
Trial 4	1000	0.5	1	2.197
Trial 5	2000	0.5	1	2.197
Trial 6	5000	0.5	1	2.197
Trial 7	10000	0.5	1	2.197
Trial 8	20000	0.5	1	2.197

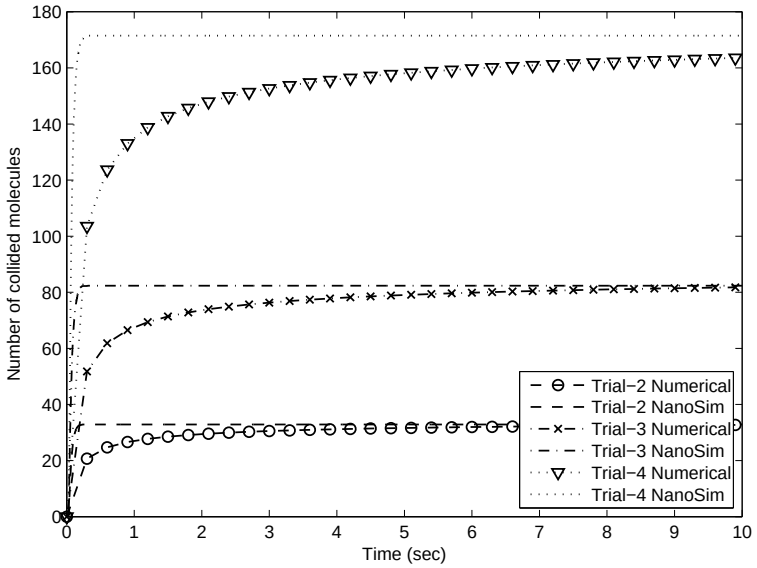

Figure 6.5: Behavior of collided molecules for Q = 200, 500, 1000.

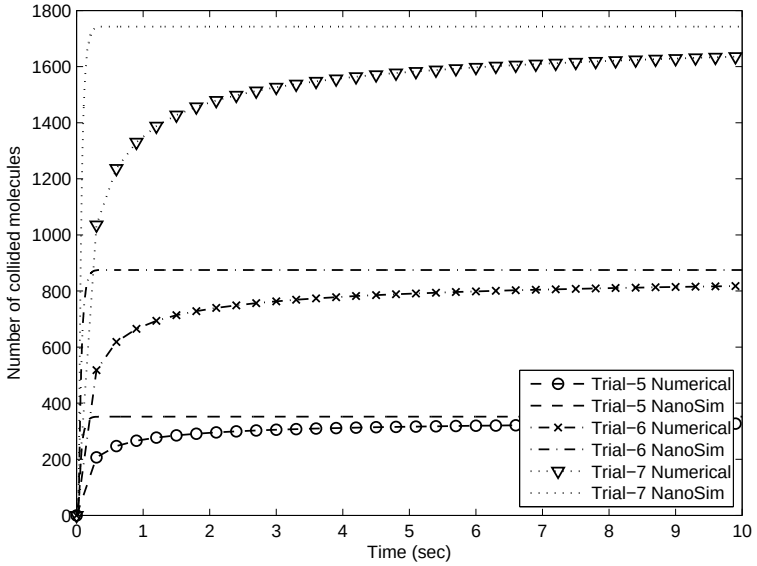

Figure 6.6: Behavior of collided molecules for Q = 2000, 5000, 10000.

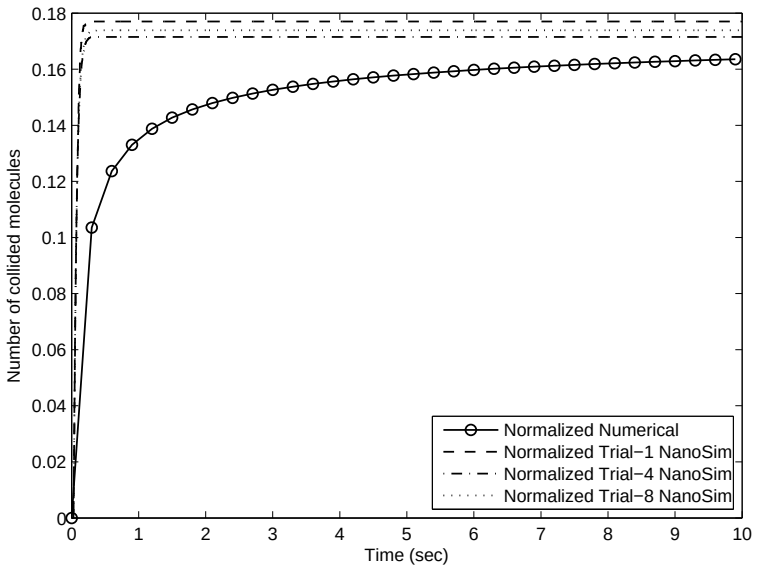

Figure 6.7: Normalized number of collided molecules for Q = 100, 1000, 20000.

In addition to these, we present exhaustive performance evaluation of captured

molecules according to Q in Table 6.3. Here, error ratio is estimated as follows

$$ER = \frac{|ECM - OCM|}{RM} \tag{6.1}$$

where ECM is the expected captured molecules according to numerical analyses, OCM is the captured molecules obtained from NanoSim, RM is the total amount of released molecules. As observed from Table 6.3, although the number of variance molecules increases with the amount of released molecule, error ratios of trials do not depend on the amount of released molecules. The error ratios of trials range from 0.07% to 1.35%. The narrowness of the range, which can be an acceptable value, shows that the amount of released molecules does not affect the error ratio of NanoSim.

Table 6.3: Error analysis of diffusion capability by captured molecules with respect to Q.

	Obtained Captured Molecules	***Expected Captured Molecules***	***Variance Molecules***	***Error Ratio***
Trial 1	17.7	16.4	1.34	1.35%
Trial 2	32.9	32.7	0.14	0.07%
Trial 3	82.3	81.8	0.57	0.12%
Trial 4	171.5	163.5	7.95	0.8%
Trial 5	351.9	327.1	24.76	1.24%
Trial 6	874.9	817.7	57.13	1.14%
Trial 7	1742.8	1635.4	107.32	1.07%
Trial 8	3477.8	3270.9	206.94	1.03%

6.1.2 Validation of Diffusion Coefficient

In this section, we validate diffusion functionality of NanoSim framework with respect to diffusion coefficient, D, which is the parameter that affects the speed of the diffusion. D is expressed in terms of temperature, viscosity and the radius of molecule as explained in (3.4). First of all, we analyze the effects of temperature, viscosity and the radius of molecule on simulation results one by one. Then, we analyze the behavior of D on NanoSim. Moreover, we measure the performance of NanoSim with respect to diffusion coefficient.

If we consider our nanomachine as a unit volume, we can use (5.2) in our analysis. In (5.2), the saturation time, t^*, can be expressed in terms of D as follows

$$t^* = \frac{C_1}{D(erfc'(C_2 D))^2} \tag{6.2}$$

in which $erfc'$ is the inverse function of $erfc$, and C_1, C_2 are constant values. From (6.2), it is observed that while D increases, t^* decreases and vice versa.

We categorize the trials in order to inspect the effects of temperature, viscosity and the radius of molecule on D individually. We analyze temperature, viscosity and the radius of molecule in category a, b and c, respectively. The simulation parameters and categorization of them are given in Table 6.4.

In category a, we examine the impact of temperature on simulation results. We fix all simulation parameters except temperature. In Figure 6.8, while temperature increases, which means increase of D, the saturation time of trials decreases as derived in (6.2).

We examine the influence of viscosity of medium on simulation results in category b. We fix all simulation parameters except viscosity. In Figure 6.9, while viscosity increases, which means decrease of D, the saturation time of trials increases as derived in (6.2).

In category c, we inspect the effect of the radius of carrier molecule on simulation results. We fix all simulation parameters except the radius of information molecule. In Figure 6.10, while the radius of carrier molecule increases, which means decline of D, the saturation time of trials increases as derived in (6.2).

Table 6.4: Simulation parameters to analysis impact of temperature, viscosity of medium and radius of the carrier molecules on simulation results.

		Q	$a\ (\mu m)$	$R\ (\mu m)$	$T\ (K)$	$\eta\ (J/K)$	$r_M\ (nm)$	$D\ (\mu m^2/s)$
a	**Trial 1**	1000	0.3	1	150	0.001	100	1.098
	Trial 2	1000	0.3	1	300	0.001	100	2.197
	Trial 3	1000	0.3	1	600	0.001	100	4.394
	Trial 4	1000	0.3	1	900	0.001	100	6.591
b	**Trial 5**	1000	0.4	1	300	0.0005	100	4.394
	Trial 6	1000	0.4	1	300	0.001	100	2.197
	Trial 7	1000	0.4	1	300	0.002	100	1.098
	Trial 8	1000	0.4	1	300	0.003	100	0.732
c	**Trial 9**	1000	0.6	1	300	0.001	50	4.394
	Trial 10	1000	0.6	1	300	0.001	100	2.197
	Trial 11	1000	0.6	1	300	0.001	200	1.098
	Trial 12	1000	0.6	1	300	0.001	300	0.732

In the rest of this section, we examine the impact of D on NanoSim. To achieve our purpose, the simulator is run with the parameters given in Table 6.5.

Figure 6.11 plots the number of collided molecules with time behavior of all trials given in Table 6.5. Diffusion coefficient has no impact on the total number of collided molecules according to (5.18). If we check the total number of collided molecules of trials on Figure 6.11, we see that they are in the range 170 – 180. The width of range, 10, is an acceptable value in order to infer that diffusion coefficient does not effect the total number of collided molecules.

In 6.2, we can neglect the impact of D on $erfc'$, hence we can assume $erfc'$ expression as a constant value. After the assumption, it is seen that t^* is propor-

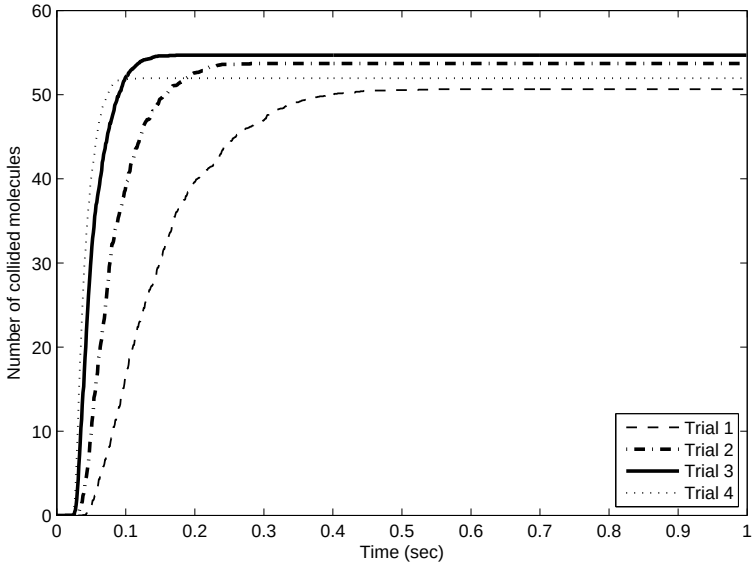

Figure 6.8: Impact of the temperature on simulation results.

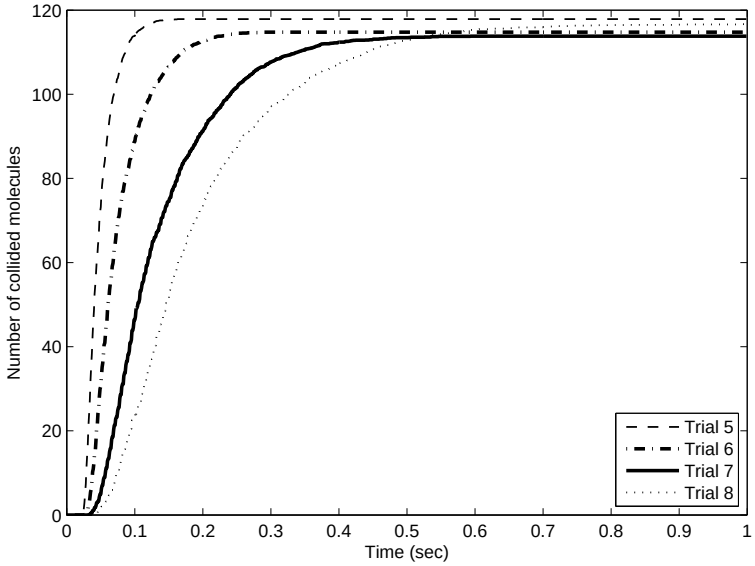

Figure 6.9: Impact of the viscosity of medium on simulation results.

tional to $\frac{1}{D}$. The inverse proportion relation between saturation time and diffusion coefficient is depicted in Figure 6.12.

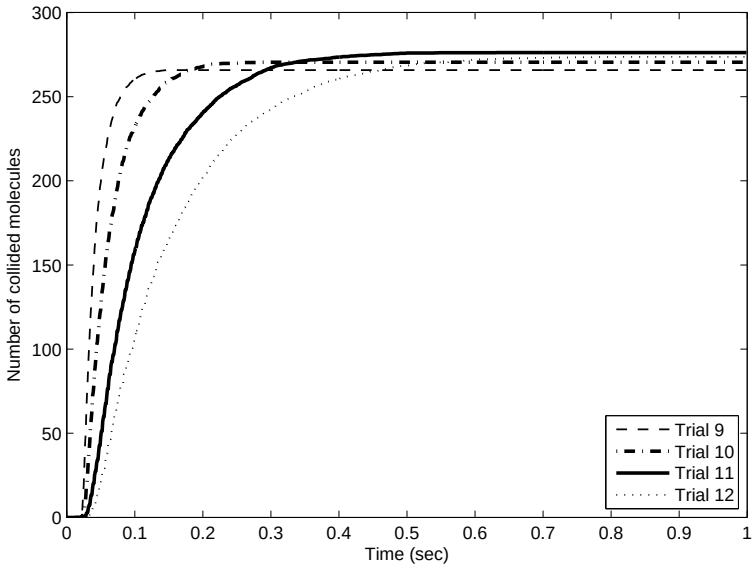

Figure 6.10: Impact of the radius of carrier molecule on simulation results.

Table 6.5: Simulation parameters to validate diffusion capability by diffusion coefficient.

Symbol (Unit)	***Trial 1***	***Trial 2***	***Trial 3***	***Trial 4***
Q	1000	1000	1000	1000
$R\ (\mu m)$	1	1	1	1
$a\ (\mu m)$	0.5	0.5	0.5	0.5
$s\ (\mu m)$	0.1	0.1	0.1	0.1
$T\ (K)$	300	100	100	100
$\eta\ (J/K)$	0.001	0.001	0.002	0.002
$r_M\ (nm)$	10	10	10	20
$D\ (\mu m^2/s)$	21.973	7.324	3.662	1.831

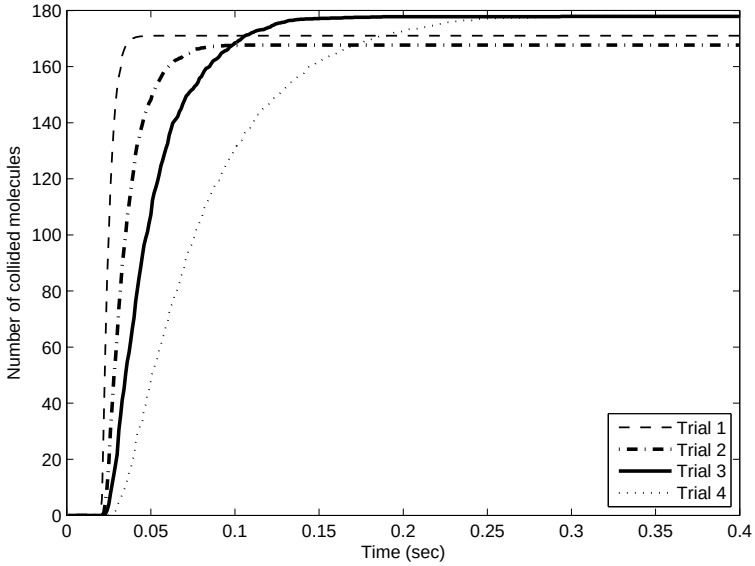

Figure 6.11: Number of collided molecules for different diffusion coefficients.

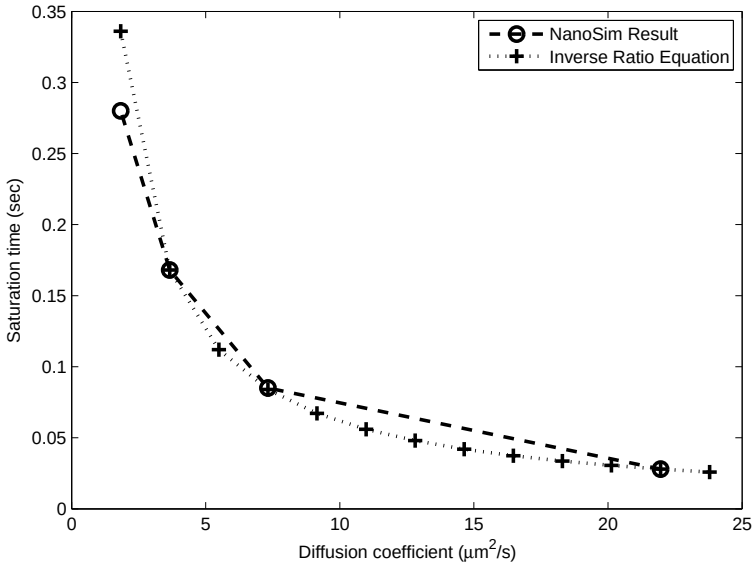

Figure 6.12: The inverse proportion between t^* and diffusion coefficient. The inverse ratio equation is equal to $y = 0.615/x$.

6.1.3 Validation of NanoSim Scalability

We evaluate the performance of the simulator with respect to the ratio between the radius of nanomachine and radius of the shelter up to now. However, it is not standalone enough for validation and performance evaluation of the simulator. Scalability factor with respect to a/R should also be examined in the validation and performance evaluation. We make our scaling analysis for a specific ratio, i.e., 0.4. We ran the simulator with the parameters given in Table 6.6.

Table 6.6: Simulation parameters to validate diffusion capability by simulator scale.

Symbol (Unit)	***Trial 1***	***Trial 2***	***Trial 3***	***Trial 4***	***Trial 5***
Q	2000	2000	2000	2000	2000
R (μm)	0.5	1	1.5	2.0	2.5
a (μm)	0.2	0.4	0.6	0.8	1
D $(\mu m^2/s)$	2.197	2.197	2.197	2.197	2.197
λ (nm)	100	100	100	100	100

Figure 6.13 plots the number of captured molecules with time behavior for all trials given in Table 6.6. Since lattice size of the simulation is fixed, the distance between source and nanomachine is proportional to the number of lattice.

If we consider our nanomachine as a unit volume, we can use (5.2) in our analysis. In (5.2), the saturation time, t^*, can be given in terms of r as follows

$$t^* = \frac{r^2}{C_1(erfc'(C_2 r))^2} \tag{6.3}$$

in which $erfc'$ is the inverse $erfc$ function and C_1, C_2 are constant values. It is seen that t^* is proportional to r^2. Hence, the following relationship exists between the t^* trials

$$R_1 < R_2 < R_3 < R_4 < R_5 \Rightarrow t_1^* < t_2^* < t_3^* < t_4^* < t_5^* \tag{6.4}$$

It is observed in Figure 6.13 that the relationship among t^* of simulator results is consistent with (6.4).

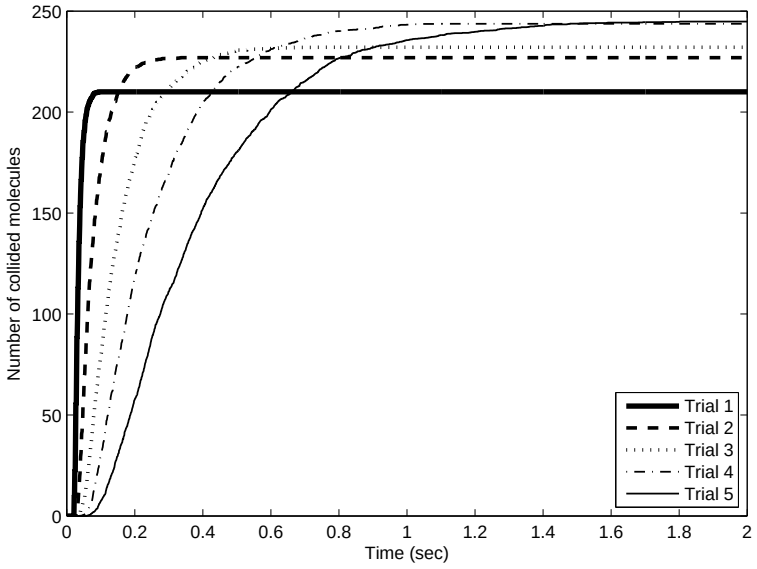

Figure 6.13: Number of collided molecules for different scales.

If the effect of scaling is analyzed considering (5.18), all parameters except $\frac{(a^2-R^2)}{d}\ln(\frac{R-a}{R-a-d})$ is proportional to the scaling factor, whereas $\frac{(a^2-R^2)}{d}\ln(\frac{R-a}{R-a-d})$ is proportional to k^2. Since $(a^2 - R^2)$ is always negative, the number of collided molecules decreases when k increases as depicted in Figure 6.14.

We plot the relation between utilization of molecules and the radius of the shelter in Figure 6.14 for all trials given in Table 6.6. It is observed that when the radius of covering sphere is $0.5\mu m$, the utilization ratio is almost the same for NanoSim and numerical analysis, whereas the ratio of difference between them over utilization ratio of numerical analysis, error ratio, increases up to 20% as radius goes up to $2.5\mu m$. Although error ratio is approximately 20% for big scaling factor, the ratio of difference over total utilization ratio is approximately 2% which is an acceptable figure.

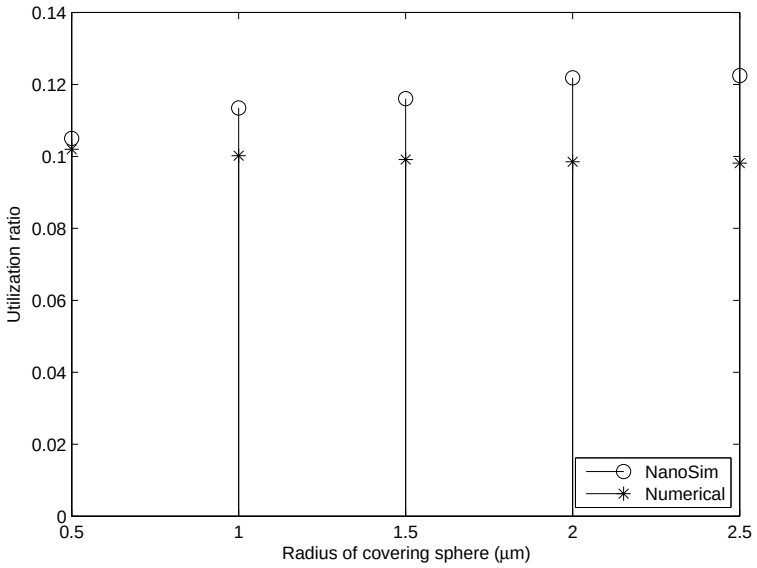

Figure 6.14: Utilization ratio for different scales.

6.2 Validation of Reaction Functionality

We can categorize reaction capability of NanoSim into two components. One of them realizes the reaction mechanism given in [54] and the other one applies SSA in NanoSim. In the rest of this section, we validate and make performance analysis of reaction characteristics of NanoSim.

6.2.1 Validation of Berg Reaction Capability

The reaction probability, P_{React}, is independent from collision probability of the molecule as mentioned in Section 5.3. Hence, there is a linear relationship between collided and captured number of molecules. The reaction parameters which affect P_{React} are the number of receptor molecules, binding radius of receptor molecules and the the radius of the nanomachine. We prepared various data sets for each ratio to analyze reaction functionality. The data sets are represented in Table 6.7.

The simulator is run with the parameters given in Table 6.7. The behavior of collided molecules and captured molecules, achieved from simulator output, are plotted in Figure 6.15.

Table 6.7: Simulation parameters to validate Berg reaction functionality.

Symbol (Unit)	***Trial 1***	***Trial 2***	***Trial 3***	***Trial 4***	***Trial 5***	***Trial 6***	***Trial 7***
Q	1000	1000	1000	1000	1000	1000	1000
R (μm)	1	1	1	1	1	1	1
a (μm)	0.3	0.4	0.5	0.6	0.7	0.8	0.9
D $(\mu m^2/s)$	2.197	2.197	2.197	2.197	2.197	2.197	2.197
N	5000	3000	3000	5000	10000	5000	2000
r_B (nm)	7	7	12	13	6	14	21

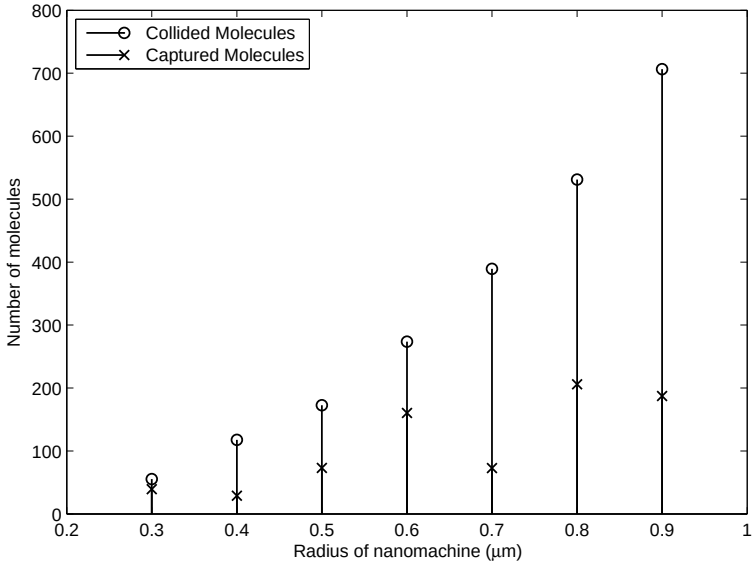

Figure 6.15: Collided and captured molecules.

It is expected that the number of captured molecules is obtained by multiplying the collided molecule number with the probability of reaction which is given in (5.22). However, the number of obtained captured molecules is not exactly the same as the excepted one. We present exhaustive performance evaluation in Table 6.8. In

the table, error ratio is estimated as follows

$$Error\ Ratio = \frac{(Expected\ Cap.\ Mol. - Obtained\ Cap.\ Mol.)}{Col.\ Mol.} \tag{6.5}$$

Table 6.8: Error analysis of Berg reaction capability.

	Collision Number	***Obtained Captured Number***	***Expected Captured Number***	***Error Ratio***
Trial 1	55.4	39.3	37.7	2.8%
Trial 2	117.7	28.8	27.3	1.3%
Trial 3	172.6	73	74.56	0.9%
Trial 4	273.7	160.4	160.61	0%
Trial 5	389.3	72.8	71.5	0.3%
Trial 6	531.2	205.8	203.35	0.5%
Trial 7	706.4	187.3	192.30	0.7%

As observed from Table 6.8, the error ratio of trials is at most 2.8%, which is an acceptable value.

6.2.2 Validation of Gillespie Reaction Capability

In this section, we validate and evaluate the performance of SSA implementation of NanoSim. We make numerical analysis of SSA according to Gillespie-Scenario, in which ligand and receptor molecules are in a lattice side, in Section 5.4. In the validation of SSA implementation, we make our trials according to the parameters given in Table 6.9.

As derived in (5.27), the behavior of captured molecules depends on initial number of ligand molecules, receptor molecules, reaction rate, volume of well-stirred medium and time. We check the impact of each of these parameters in Figures

Table 6.9: Simulation parameters to validate SSA.

	# of Ligand Molecules	***# of Receptor Molecules***	***Reaction Rate*** $(s/\mu M^2)$	***Volume Side*** *(nm)*
$Trial$ 1	40	20	0.01	100
$Trial$ 2	40	20	0.001	100
$Trial$ 3	40	20	0.005	100
$Trial$ 4	80	10	0.005	100
$Trial$ 5	80	20	0.005	200
$Trial$ 6	80	20	0.005	100
$Trial$ 7	80	20	0.005	50

6.16−6.18. In each figure, the response of NanoSim and numerical analysis are plotted in order to validate each of these parameters one by one.

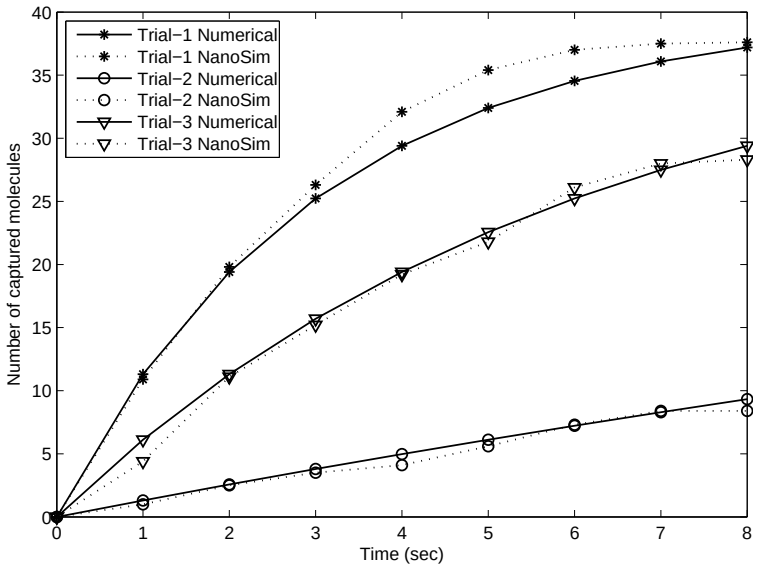

Figure 6.16: The captured molecules for varying reaction rates.

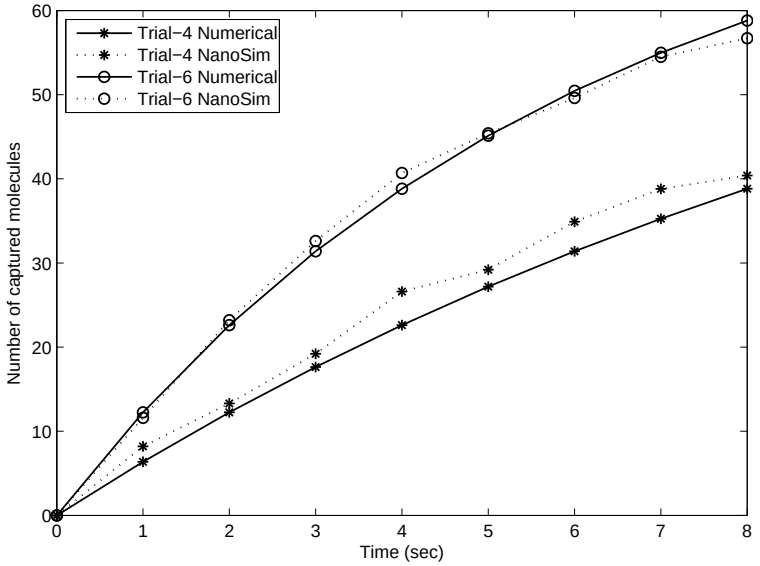

Figure 6.17: The captured molecules for varying number of receptors.

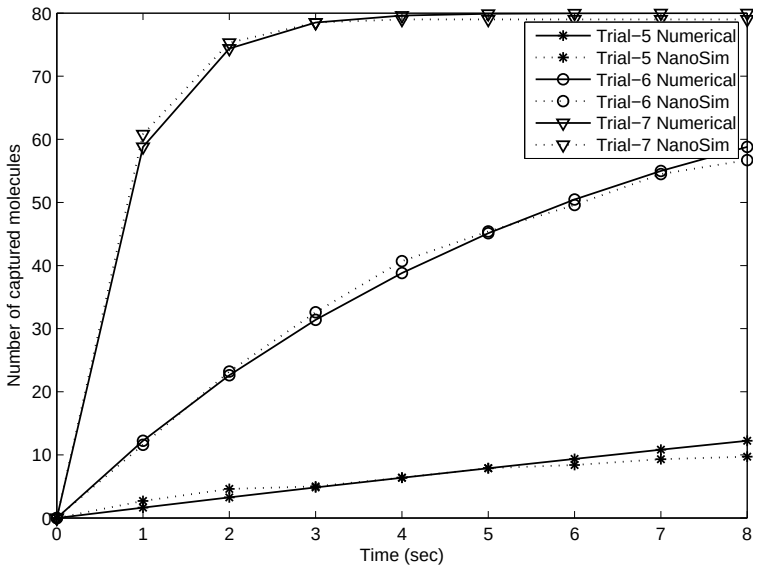

Figure 6.18: The captured molecules for varying volumes.

The impact of reaction rate can be observed in Figure 6.16. Numerical result and NanoSim output show almost the same behavior. Nevertheless, Trial 1, in which a deviation exists between NanoSim and numerical results, has the largest reaction

rate. Figure 6.17 shows the behavior of NanoSim and numerical analysis for varying number of receptor molecules. Numerical result and NanoSim output show almost the same behavior as the previous one. The trias that have less number of receptor molecules, and show more inconsistency with numerical analysis. Figure 6.18 gives the behavior of NanoSim for varying volumes. Here, NanoSim exhibits similar responses with the numerical results for alternating volumes.

6.3 Utilization Ratio and Error Analysis of Simulator

In this section, we make performance evaluation according to the utilization ratio of NanoSim. Utilization ratio of a trial, UR, can be achieved as follows

$$UR = \frac{\#\ of\ Captured\ Molecules}{Total\ \#\ of\ Molecules} \tag{6.6}$$

The utilization of the simulator for different a/R ratios is given in Figure 6.19. Here, captured molecules are plotted for two sources which are obtained from simulator and numerical analysis. It is observed that the results shows parallel attitudes. However, they are not exactly matched.

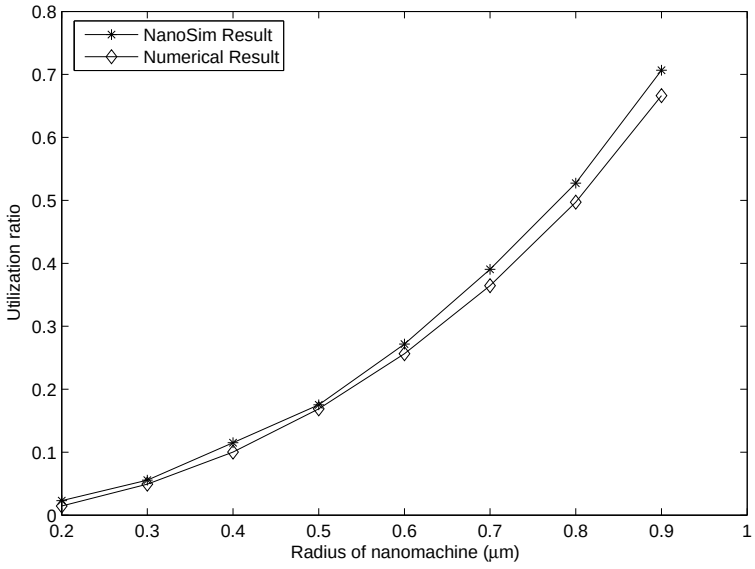

Figure 6.19: The utilization ratio of NanoSim and numerical analysis.

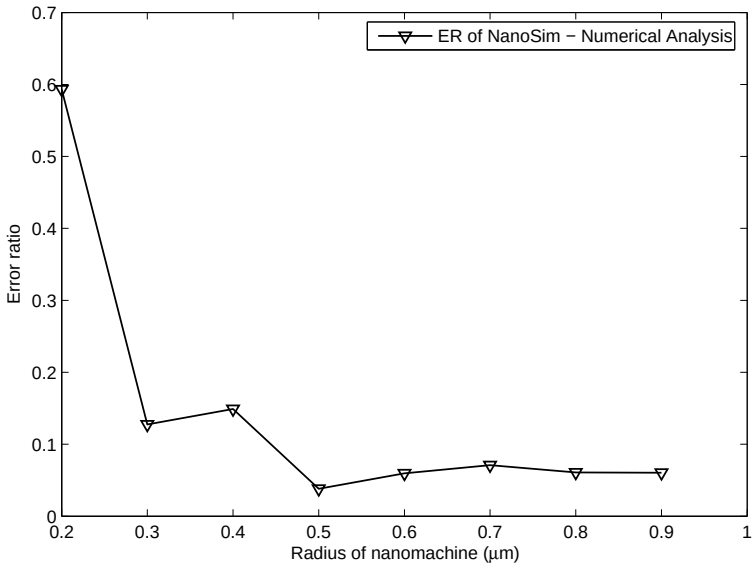

Figure 6.20: NanoSim error ratio.

We analyze this discrepancy in an error model. In this error model, we divide the difference in the number of captured molecules between the NanoSim and numerical analysis results by numerical analysis result as follows

$$ER = \frac{|SR - AR|}{AR} \tag{6.7}$$

in which ER is the error ratio, SR is the number of captured molecules by NanoSim, AR is the number of captured molecules with respect to analysis result. Figure 6.20 depicts the ER value of NanoSim in varying a/R ratios.

We can find error ratio of the NanoSim by taking the average of the error ratios of the radius given in Figure 6.20. Here, it is observed that the error ratio decreases and becomes stable for a/R values larger than 0.5. Thus, it is reasonable to select a/R values in the simulations.

The average error ratio of NanoSim is 14.9% according to error ratio values given in Figure 6.7. However, if the figure is inspected, it is noticed that there is a peak value on $a = 0.2$ for numerical analysis result. The reason of this peak value is that the captured molecule number is quite small for $a = 0.2$. Although the difference between SR and AR is quite small with respect to total number of receptor

molecules, a greater error ratio is observed for $a = 0.2$. Thus, it is rational to ignore $a = 0.2$ while taking the average. If we ignore $a = 0.2$, the error ratio of the NanoSim becomes 8.3%.

CHAPTER 7

CONCLUSION

Molecular communication, inspired from the real biological molecular systems, gathers several disciplines such as nanotechnology, biotechnology and communication systems. Most of the existing communication networks' background is not applicable for nanonetworks. For instance, by molecular communication, short-range communication between nanomachines is provided, and instead of electromagnetic waves, molecules are used as communication carriers. Owing to the distinctions between traditional and molecular communication, and lack of open source and widely used molecular communication simulator, we have designed and implemented our molecular channel model in ns-2.

In this thesis, we investigated the characterization of molecular channel properties as the first step. The molecular channel models in the literature, i.e., diffusive, molecular motor-based and gap junction-based molecular channels, are described. We analyzed and modeled diffusive molecular communication. We chose reaction-diffusion model to simulate diffusive molecular channel. GMP, reaction-diffusion modeling algorithm, is selected as our fundamental algorithm. We represented our classes, their functionalities and relationship with each other in order to build-up diffusive molecular channel. We have constructed our network stack by creating classes which are inherited from basic structures of ns-2. We have merged diffusive channel structures and this network stack. Eventually, NanoSim is formed. Nevertheless, we have not implemented complex MAC, routing or transport layer protocols. We have

only implemented a basic MAC protocol as an illustration.

Another aim of this thesis is to provide development environment for the other types of nanonetworks. NanoSim has an object-oriented structure like ns-2 which brings modularity to it. For instance, we have also designed and implemented network simulator for molecular motor-based communication. Motor-based communication part of NanoSim, which requires diffusion functionality, uses Diffusion class of diffusive communication part to handle diffusion functionality. This demonstrates the modularity of NanoSim. By adding new classes and using the current ones, NanoSim can gain abilities to simulate new molecular channel models.

We exhibit numerical analysis of molecular communication for specific scenarios. In the analysis, we formulate the behavior of captured molecules with time in terms of radius of nanomachine, distance from source and time. Additionally, the formulation of final number of bound molecules is obtained in terms of radius of nanomachine and distance from source. The accuracy of formulas were checked with MATLAB simulation.

We presented performance evaluation and validation for diffusion and reaction capabilities of NanoSim. The performance evaluation and validation are made in terms of captured molecule, diffusion coefficient, simulation scalability and utilization rate. Additionally, we analyze the error ratio of NanoSim, ER. The average ER value of NanoSim is 8.3%. Nevertheless, there is a drawback in the performance evaluation of NanoSim scalability. The error ratio is approximately 20% for big scaling factors. Fortunately, the amount of error is almost 2% of the total number of molecules for $a/R = 0.4$.

We have elaborately modeled and implemented diffusive molecular channel. On the other hand, motor-based molecular communication part of NanoSim is designed and implemented in order to serve as a model for the future extensions of NanoSim. Hence, elaborately modeling and implementation of motor-based communication can be considered as future work. Gap junction-based communication can also be considered in the same way. Moreover, in the current implementation of NanoSim, nanomachines are fixed objects. Adding mobility functionality

to nanomachines is another future work. In addition, the experimental validation of NanoSim can be considered as one of the important future directions.

Finally, a full protocol stack which contains MAC, routing and transport layer protocols is required to complete ns-2 molecular simulation suite. However, research on molecular communication area is too new. Hence, network protocols for each layer do not exist or are just at beginning stage for molecular communication. We believe that NanoSim will provide practical and beneficial simulation suite to develop network protocols for nanonetworks.

REFERENCES

[1] S. Hiyama, Y. Moritani, T. Suda, R. Egashira, A. Enomoto, M. Moore, T. Nakano, "Molecular Communication", *Proc. NSTI Nanotech'05*, Vol. 3, pp. 391-394, May 2005.

[2] Y. Moritani, S. Hiyama, T. Suda, "Molecular Communication among Nanomachines Using Vesicles", *Proc. NSTI Nanotech'06*, Vol. 2, pp. 705-708, May 2006.

[3] S. Hiyama, Y. Isogawa, T. Suda, Y. Moritani, K. Sutoh, "A Design of an Autonomous Molecule Loading/Transporting/Unloading System Using DNA Hybridization and Biomolecular Linear Motors", *Proc. ENS'05*, Dec. 2005.

[4] B. Atakan, O. B. Akan, "Deterministic Capacity of Information Flow in Molecular Nanonetworks," to appear in *Nano Communication Networks Journal (Elsevier)*, 2010.

[5] T. Nakano, M. Moore, A. Enomoto, T. Suda, T. Koujin, T. Haraguchi, Y. Hiraoka, "A Cell-based Molecular Communication Network", *Proc. of the IEEE/ACM International Conference on Bio Inspired Models of Network, Information and Computing Science Systems (BIONETICS)*, Dec. 2006.

[6] M. J. Berridge, "The AM and FM of Calcium Signaling", *Nature*, Vol. 386, pp. 759-780, 1997.

[7] A. Eckford, "Nanoscale Communication with Brownian Motion", *Proc. 41st Annual Conference on Information Sciences and Systems (CISS 2007)*, 2007.

[8] M. Moore, A. Enomoto, T. Nakano, R. Egashira, ,T. Suda, A. Kayasuga, H. Kojima, H. Sakakibara, K. Oiwa, "A Design of a Molecular Communication System for Nanomachines Using Molecular Motors", *IEEE Conference on Pervasive Computing and Communications*, 2006.

[9] T. Nakano, T. Suda, M. Moore, R. Egashira, A. Enomoto, K. Arima, "Molecular Communication for Nanomachines Using Intercellular Calcium Signaling", *Proc. 5th IEEE Conference on Nanotechnology*, 2005.

[10] P. J. Thomas, D. J. Spencer, S. K. Hampton, P. Park, J. P. Zurkus, "The Diffusion Mediated Biochemical Signal Relay Channel", *Proc. 17th Annual Conference on Neural Information Processing Systems (NIPS '03)*, 2003.

[11] T. Suda, M. Moore, T. Nakano, R. Egashira, A. Enomoto, S. Hiyama, Y. Moritani, "Exploratory Research in Molecular Communication between Nanomachines", *UCI Technical Report, 05-3*, Mar. 2005.

[12] T. Nakano, T. Suda, T. Koujin, T. Haraguchi, Y. Hiraoka, "Molecular Communication Through Gap Junction Channels", *Springer Trans. Comput. Syst. Biol.*, Vol. 5410 , pp. 81-99, 2008.

[13] M. J. Moore, A. Enomoto, T. Nakano, Y. Okaie, A. Kayasuga, H. Kojima, H. Sakakibara, K. Oiwa, T. Suda, "Molecular Communication: Simulation of Microtubule Topology", *Proc. 2nd Int. Workshop Natural Comput.*, pp. 134-144, 2007.

[14] Y. Moritani, S. Hiyama, T. Suda, "Molecular Communication for Health Care Applications", *Proc. 4th Annu. IEEE Conf. Pervasive Comput. Commun. Workshops*, pp. 553-557, 2006.

[15] I. F. Akyildiz, F. Brunetti, C. Blazquez, "NanoNetworking: A New Communication Paradigm", *Computer Networks Journal (Elsevier)*, June 2008.

[16] B. Atakan, O. B. Akan, "An Information Theoretical Approach for Molecular Communication", *Bio-Inspired Models of Network, Information and Computing Systems, 2007. Bionetics 2007. 2nd*, Dec. 2007.

[17] B. Atakan, O. B. Akan, "On Channel Capacity and Error Compensation in Molecular Communication", *Springer Trans. on Computational System Biology*, 2008.

[18] M. J. Moore, A. Enomoto, K. Oiwa, T. Suda, "Molecular Communication: Uni-cast Communication on a Microtubule Topology", *IEEE International Conference on Systems, Man and Cybernetics(SMC)*, pp.18-23, Oct. 2008.

[19] F. Walsh , S. Balasubramaniam, D. Botvich, T. Suda , T. Nakano, S. F. Bush, M. Ó. Foghlú "Hybrid DNA and Enzyme Based Computing for Address Encoding, Link Switching and Error Correction in Molecular Communication", *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*,Vol. 3, pp. 28-38, 2009.

[20] B. Atakan, O. B. Akan, "On Molecular Multiple-Access, Broadcast, and Relay Channels in Nanonetworks", *International Conference On Bio-Inspired Models Of Network, Information And Computing Systems archive Proceedings of the 3rd International Conference on Bio-Inspired Models of Network*, 2008.

[21] M. Moore, A. Enomoto, T. Nakano, T. Suda, A. Kayasuga, H. Kojima, H. Sakakibara, K. Oiwa, "Simulation of a Molecular Motor Based Communication Network", *Bio-Inspired Models of Network, Information and Computing Systems*, 2006.

[22] T. Nakano, T. Suda, T. Koujin, T. Haraguchi, Y. Hiraoka, "Molecular Communication through Gap Junction Channels: System Design, Experiments and Modeling", *Proc. 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems (BIONETICS 2007)*, Dec. 2007.

[23] G. Peskir "On the diffusion coefficient: The Einstein Relation and Beyond. Stochastic Models", *Stochastic Models*, Vol. 19 (3), pp. 383-405, 2003.

[24] H. C. Berg "Random Walks in Biology", *2nd edn (Princeton, NJ: Princeton University Press)*, 1993.

[25] J. Crank "The Mathematics of Diffusion", *2nd edn (Oxford:Oxford University Press)*, 1975.

[26] D. T. Gillespie "A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions", *J. Comput. Phys.*, Vol. 22, pp. 403-434, 1976.

[27] D. T. Gillespie "Exact Stochastic Simulation of Coupled Chemical Reactions", *The Journal of Physical Chemistry*, Vol. 81, No. 25, 1977.

[28] S. Hiyama, Y. Moritani, T. Suda "Molecular Transport System in Molecular Communication", *NTT DOCOMO Technical Journal*, Vol. 10, No. 3, Dec. 2008.

[29] Y. Moritani, S. Hiyama, S. M. Nomura, K. Akiyoshi, T. Suda, "A Communication Interface Using Vesicles Embedded with Channel Forming Proteins in Molecular Communication", *Proc. of the 2nd ICST International Conference on Bio-Inspired Models of Network, Information and Computing Systems (BIONETICS)*, Dec. 2007.

[30] J. V. RodrÃguez, J. A. Kaandorp, M. Dobrzyński, J. G. Blom, "Spatial Stochastic Modelling of the Phosphoenolpyruvatedependent Phosphotransferase (PTS) Pathway in Escherichia Coli", *Bioinformatics, Oxford University Press*, Vol. 22, No. 15, pp. 1895-1901, Aug. 2006.

[31] B. Chopard, L. Frachebourg, M. Droz, "Multiparticle Lattice Gas Automata for Reaction Diffusion Systems", *Int. J. Mod. Phys. C*, Vol. 5, pp. 47-63, 1994.

[32] "The Network Simulator," [Online]. Available: http://www.isi.edu/nsnam/ns/. [Accessed: 09 May 2010].

[33] "E-Cell Simulation," [Online]. Available: http://www.e-cell.org/ecell/. [Accessed: 09 May 2010].

[34] T. C. Meng, S. Somani, L. Ye, A. Sairam, Z. Hao, A. Krishnan, K. Sakharkar, P. Dhar, "Cellware: The Grid-Enabled Tool for Cell Modeling And Simulation", *Bioinformatics (in press)*, 2004.

[35] L. Loew, J. C. Schaff, "The Virtual Cell: A Software Environment for Computational Cell Biology", *Trends in Biotech*, Vol. 19, No.10, 2001.

[36] Y. Moritani, S. Hiyama, T. Suda, "Molecular Communication - A Biochemically-Engineered Communication System", *Proc. of the Frontiers in the Convergence of Bioscience and Information Technologies (FBIT)*, pp. 839-844, Oct. 2007.

[37] K. Fall, "The ns Manual (formerly ns Notes and Documentation)", *The VINT Project*, Jan. 2009.

[38] K.Takahashi, S.N. Arjunan, M. Tomita, "Space in Systems Biology of Signaling Pathways-Towards Intracellular Molecular Crowding in Silico", *FEBS Lett.*, Vol. 579, pp. 1783-1788, 2005.

[39] M. Ander, P. Beltrao, B. D. Ventura, B. J. Ferkinghoff, M. Foglierini, A. Kaplan, C. Lemerle, T. I. Oliveira, L. Serrano, "SmartCell, a Framework to Simulate Cellular Processes that Combines Stochastic Approximation with Diffusion and Localisation: Analysis of Simple Networks", *Syst. Biol.*, No. 1, pp. 129-138, 2004.

[40] J. Hattne, D. Fange, J. Elf, "Stochastic Reaction-Diffusion Simulation with MesoRD", *Bioinformatics*, Vol. 21, pp. 2923-2924, 2005.

[41] A. B. Stundzia, C. J. Lumsden, "Stochastic Simulation of Coupled Reaction-Diffusion Processes", *J. Comput. Phys.*, Vol. 127, pp. 196-207, 1996.

[42] M. Moore, T. Suda, K. Oiwa, "Molecular Communication: Modeling Noise Effects on Information Rate", *IEEE Transactions on Nanobioscience*, Vol. 8, pp. 169-180, June 2009.

[43] M. J. Moore, A. Enomoto, S. Watanabe, K. Oiwa, T. Suda, "Simulating Molecular Motor Uni-cast Information Rate for Molecular Communication", *the 43rd IEEE Annual Conference on Information Sciences and Systems (CISS)*, Mar. 2009.

[44] M. Moore, A. Enomoto, T. Nakano, Y. Okaie, T. Suda, "Interfacing with Nanomachines through Molecular Communication", *Proc. of the IEEE International Conference on Systems, Man and Cybernetics (SMC)*, Oct. 2007.

[45] A. W. Eckford, “Molecular Communication: Physically Realistic Models and Achievable Information Rates”, *Submitted to IEEE Transactions on Information Theory*, 2008.

[46] A. F. Harris, M. Zorzi, “Modeling the Underwater Acoustic Channel in ns2”, *Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools*, 22-27 Oct. 2007.

[47] “NS by Example,” [Online]. Available: http://nile.wpi.edu/NS/. [Accessed: 09 May 2010].

[48] R. A. Freitas, “Nanomedicine, Volume I: Basic Capabilities”, *Landes Biosience*, 1999.

[49] G. Alfano, D. Miorandi, “On Information Transmission Among Nanomachines”, *in Proc. of the First Int. Conference on Nano-Networks (NanoNet’06)*, pp. 1-5, 2006.

[50] C. Elfgang, R. Eckert, H. Lichtenberg-Frate, A. Butterweck, O. Traub, R.A. Klein, D. F. Hulser, K. Willecke, “Specific Permeability and Selective Formation of Gap Junction Channels in Connexin-Transfected HeLa Cells”, *J. Cell Biol.*, Vol. 129, pp. 805-817, 1995.

[51] T. Issariyakul, E. Hossain, “Introduction to Network Simulator NS2”, *Springer*, 2008.

[52] H. Anton, I. Bivens, S. Davis, “Calculus”, *New York: Wiley*, 2005.

[53] T. E. Turner, S. Schnell, K. Burrage, “Stochastic Approaches for Modelling in Vivo Reactions”, *Comput. Biol. Chem.*, Vol. 28, pp. 165-178, 2004.

[54] H. C. Berg, E. M. Purcell, “Physics of Chemoreception”, *Biophys J.*, pp.193-219 , Nov. 1977.

[55] R. Feynman, “There’s Plenty of Room at the Bottom”, *American Physical Society at the California Institute of Technology (Caltech)*, 29 Dec. 1959.

APPENDIX A

COMMANDS AT A GLANCE

A.1 Commands for Both Diffusive and Motor-Based Molecular Communication

The list of commands related to both diffusive and motor-based molecular communication are as follows:

```
$ns_ node-config -molChannelType <type>
                 -llType <type>
                 -macType <type>
                 -phyType <type>
                 -diffusion <type>
                 -reaction <type>
```

`molChannelType` parameter declares that the subsequent new nodes created will be nanonodes. It determines the molecular communication type. Possible values of `molChannelType` parameters are *Mol / Channel / Diffusion* and *Mol / Channel / Motor*. For diffusive molecular communication, `molChannelType` parameter shall be *Mol / Channel / Diffusion*. Additionally, three types of reaction model can be selected with `reaction` parameter. The options of are `reaction` *Reaction/No*, *Reaction/Berg* and *Reaction/Gillespie*

`$nanonode set-position <x_coordinate> <y_coordinate> <z_coordinate>`
This is a wrapper method that assigns the coordinates of a nanonode. The <x_coordi-

nate>, <y_coordinate> and <z_coordinate> define the x,y and z Cartesian coordinates in units of lattice system.

`$nanonode setMolNode <ligand_molecule> <receptor_molecule>`

This is a wrapper method that assigns the ligand and receptor molecules of nanonode. If ligand or receptor molecule is not defined in the script, it automatically generates a molecule whose diffusion coefficient is "1".

`$mol_channel set-reaction <molecule> <molecule> <value>`

This is a wrapper method that defines a reaction channel. Molecular channel decides reaction occurrences according to these reaction channels defined by the user. <value> is the reaction rate constant. Its unit is $\mu M^{-1} s^{-1}$.

`$mol_channel set-radius <molecule> <value>`

This is a wrapper method that defines the molecule and radius of the molecule. <value> is radius of the molecule. Its unit is nm.

`$molagent start`

This is a wrapper method that starts corresponding agent to release carrier molecules.

`$molagent stop`

This is a wrapper method that stops corresponding agent to release carrier molecules.

`$mol_channel start_simulation`

This is a wrapper method that starts channel activities which are diffusion and reaction. Molecular Channel event simulation command is distinct from molecular agent "start" command. This command should be specially given in the running script otherwise molecules do not start to diffuse and react.

`$mol_channel stop_simulation`

This is a wrapper method that stops channel activities which are diffusion and reaction. Molecular Channel event simulation command is distinct from molecular agent "stop" command. This command should be specially given in the running script otherwise molecules do not stop to diffuse and react.

`$ns trace-all-mollinks <value>`

This is a wrapper method that enables all molecular traces and direct molecular traces to output file the name of which is <value>.

`Agent/Mol set interval_ <value>`

This parameter defines the system interval of the simulator. Its unit is ms.

`Node/MolNode set radius_ <value>`

This parameter defines the radius of nanomachines. Its unit is one lattice length.

`MolecularEnvironment set lambda_ <value>`

This parameter defines the length of a lattice. Its unit is nm.

`Mac/Mol set interval_ <value>`

This parameter defines time interval of MolMAC. Its unit is ms.

`Agent/Mol set send_amount_ <value>`

This parameter defines the amount of molecules that MolAgent sends in one shot.

`Channel/Mol/Diffusion set const_memb_concentration_ <value>`

if const_memb_concentration_ is bigger than 0, every membrane lattice has the same receptor number value.

`MolecularEnvironment set temperature_ <value>`

This parameter defines the temperature of molecular environment. Its unit is K.

`MolecularEnvironment set viscosity_ <value>`

This parameter defines the viscosity of molecular environment. Its unit is J/K , $Pa\ m^3 K^{-1}$.

`Molecule set radii_ <value>`

This parameter defines the default value of radius of molecules. Its unit is nm.

`Channel/Mol set MIN_LAT_ <value>`

This parameter defines the minimum boundary lattice value.

`Channel/Mol set MAX_LAT_ <value>`

This parameter defines the maximum boundary lattice value.

`set error_model [new ErrorModel/Mol]`

This script creates a *MolErrorModel* object and assigns to `error_model` object.

`$error_model unit <value>`

This script determines the unit of `error_model` molecular error model object. The value of this parameter can be *molecule* or *moldata*.

`$error_model ranvar [new RandomVariable/<value>]`

This script determines the probability distribution of `error_model` molecular error model object. The value of this parameter can be $Exponential$ or $Uniform$.

`$error_model set rate_ <value>`

This script determines the rate of `error_model` molecular error model object. The value of this parameter must be between 0 and 1.

`$n1 interface-outgoing-errormodel $error_model`

This method inserts the error_model molecular error model object into outgoing link of NanoNode object, n1.

`$n1 interface-incoming-errormodel $error_model`

This method inserts the error_model molecular error model object into incoming link of NanoNode object, n1.

A.2 Commands for Only Motor-Based Molecular Communication

The list of commands related to only motor-based molecular communication are the following:

molChannelType parameter of node-config interface shall be *Mol*/*Channel*/*Motor*.

`set link [new MolLink $n1]`

This is script creates a *MolLink* object, the receiver node of which is NanoNode object, n1, then assigns it to link object.

`$sender_nanonode mollink $link`

This is a wrapper method that provides unidirectional molecular link between sender nanonode object and link molecular link object.

`Channel/Mol/Motor set concentration_quantity_ <value>`

This parameter defines whether concentration or quantity distribution is applied. CONCENTRATION = 1, QUANTITY = 2

`Channel/Mol/Motor set concentration_ <value>`

This parameter defines the molecule concentration of nanonode cytosol.

`Channel/Mol/Motor set quantity_ <value>`

This parameter defines the quantity of molecule inside one lattice of nanonode cy-

tosol.

`MolLink set sendingInterval_ <value>`

This parameter defines the time interval to transfer molecules from nanomachine to vesicle for motor-based communication. Its unit is s.

`MolLink set receivingInterval_ <value>`

This parameter defines the time interval to transfer molecules from vesicle to nanomachine for motor-based communication. Its unit is s.

`MolecularEnvironment set motorVelocity_ <value>`

This parameter defines the velocity of molecular motor. Its unit is nm/s.

`$link mollink-errormodel $error_model`

This method inserts the `error_model` molecular error model object into molecular link, i.e., `link`.

APPENDIX B

CODES

The interfaces of NanoSim, the header files, are given as follows:

Diffusion.h :

```
class Diffusion :  public Handler, public TclObject {
private:
      static MolecularEnvironment* molenv;
      long seed;
      Randomizer* r;
      int moleculePerLattice[6];
      MolChannel* channel_;
public:
      Diffusion();
      Diffusion(MolChannel*);
      static void setMolecularEnvironment(MolecularEnvironment* m);
      void diffuseEnvironment(Molecule*);
      void diffuseLattice(Lattice*, Molecule*);
      void assignMoleculeNumbers(int);
      void diffuseLarge(int);
      void diffuseSmall(int);
      void setChannel(MolChannel*);
protected:
      int command(int argc, const char*const * argv);
      void handle(Event*);
```

```
};
```

Randomizer.h:

```
class Randomizer {
private:
      long random_number;
public:
      Randomizer();
      long next();
      double gasdev(long* idum, double mean, double var);
      double rando(long* idum);
};
```

Molagent.h:

```
class Mol_Agent :  public Agent {
private:
      MolNode* node_;
      static int send_amount;
      static int agent_number;
public:
      Mol_Agent();
      int command(int argc, const char * const * argv);
      void timeout();
      void start_channel_simulation();
      static double interval;
      void start(void);
      void stop(void);
      void send(Packet* p, Handler* h);
protected:
      void recv(Packet*, Handler*);
      Mol_Timer* mol_timer_;
      int running_;
```

```
        double nextReleasetime_;
};
```

MolChannel.h:

```
class MolChannel :  public Channel {
public:
        //variables
        static int state_;
        static vector< MolNode*> molnodes;
        static int MAX_LAT;
        static int MIN_LAT;
        MolPosition* t_senderLatticePosition;
        MolPosition* t_receiverLatticePosition;
        //constructor and methods
        MolChannel();
        void add_captured_mol(MolNode*, Molecule*);
        void start_simulation();
        void stop_simulation();
        void react(double);
        void add_node(MolNode*);
        void addSpecies(Molecule*);
        vector< Molecule*> getSpecies();
        void addMolecule(MolPosition*, Molecule*, int);
        void addMoleculeInitially(MolPosition*, Molecule*, int);
        void addReactionChannel(ReactionChannel*);
        void diffuse(Molecule*);
        int total_transmitter_node_number();
        int node_queue_number(MolPhy*);
        double t_distance(MolPosition*);
        double t_totalDistance();
        //inline methods which set or get any variable
        inline Reaction* reaction();
```

```
        void set_reaction_time(double t);
        void set_diffusion_molecule(Molecule* m);
        vector< captured_molecules*> node_captured_mol();
        static int state();
        static void setState(int s);
        void set_reaction_channel(ReactionChannel* r);
        ReactionChannel* get_reaction_channel();
        MolEvent* event_instance();
        MolecularEnvironment* get_mol_env();
        vector< MolNode*> get_molnodes();
        void send(MolNode*, Molecule*, double);
        void send(MolNode*, Molecule*, int, Lattice*, double);
        //virtual methods
        virtual void init() {}
        virtual void setReceptorSpace(MolNode*) {}
        virtual void setReceptorSpaceTest(MolNode*) {}
        virtual void check(){}
        virtual void sendMolecules(MolNode* m, int a){}
        virtual void sendMoleculesTest(MolNode* m, int a) {}
        virtual void simulate(){}
protected:
        //varibles
        Lattice* passive;
        Diffusion* dif;
        Reaction* reaction_;
        MolecularEnvironment* me;
        Simulation_Timer* simulation_timer_;
        ReactionChannel* reaction_channel;
        MolEvent* channel_event;
        vector< captured_molecules*> node_captured_mol_;
        int running_;
        int* diffusion_times;
```

```
        double* diffusion_durations;
        double tsim;
        //methods
        int command(int argc, const char*const * argv);
};
```

MolChannelDiffusion.h:

```
class MolChannelDiffusion :  public MolChannel {
public:
        MolChannelDiffusion();
        void init();
        void print();
        void setReceptorSpace(MolNode*);
        void setReceptorSpaceTest(MolNode*);
        static int const_memb_concentration;
        void sendMolecules(MolNode* m, int a);
        void simulate();
        void sendMoleculesTest(MolNode* m, int a);
};
```

MolChannelMotor.h:

```
class MolChannelMotor :  public MolChannel {
public:
        MolChannelMotor();
        void init();
        void print();
        void setReceptorSpace(MolNode*);
        static int concentration_quantity;
        static double concentration;
        static int quantity;
        void sendMolecules(MolNode* m, int a);
        void simulate();
```

```
        void connectVesicle(Lattice);
        void disconnectVesicle();
};
```

MolTrace.h:

```
class MolTrace :  public Trace {
protected:
        void format(int tt, int s, int d, Packet* p);
public:
        MolTrace(int type):  Trace(type){}
};
```

MolPosition.h:

```
class MolPosition :  public TclObject {
public:
        MolPosition();
        MolPosition(double x_, double y_, double z_);
        MolPosition(int x_, int y_, int z_);
        Node* node();
        double x, y, z;
        int latX, latY, latZ;
        double getY();
        double getX();
        double getZ();
        void translatePosition(MolPosition* p);
        bool operator==(const MolPosition& other) const;
        MolPosition* copy();
        void print();
        double distance(MolPosition*);
protected:
        int command(int argc, const char*const * argv);
        Node* node_;
```

```
};
```

MolPhy.h:

```
class MolPhy :  public Phy {
public:
      MolPhy():  Phy(){}
      void sendDown(Packet* p);
      int sendUp(Packet* p);
      void recv(Packet* p, Handler*);
      void handle(Event*);
protected:
      int command(int argc, const char*const * argv);
};
```

MolLL.h:

```
class MolLL : public LL {
public:
      MolLL():  LL();
      virtual void sendDown(Packet* p);
      virtual void sendUp(Packet* p);
      virtual void recv(Packet* p, Handler* h);
protected:
      int command(int argc, const char*const * argv);
};
```

MolMAC.h:

```
class MolMac :  public Mac {
      friend class SlotTimer;
      friend class TxMoleculeTimer;
      friend class RxMoleculeTimer;
public:
      MolMac();
```

```
        virtual void recv(Packet* p, Handler* h);
        MolNode* molnode();
        static int queue_number;
        MolChannel* channel;
        void handle(Event*);
        void slotHandler(Event* e);
        void sendHandler(Event* e);
        void receiveHandler(Event* e);
        static double interval;
protected:
        int command(int argc, const char*const * argv);
        MolNode* molnode_;
        int max_node_num_;
private:
        void re_schedule();
        void makePreamble();
        void sendUp(Packet* p);
        void sendDown(Packet* p);
        void send();
        inline int is_idle(void);
        SlotTimer mhSlot_;
        TxMoleculeTimer mhTxPkt_;
        RxMoleculeTimer mhRxPkt_;
        MacState tx_state_; // outgoing state
        int tx_active_; // transmitter is ACTIVE
        static int max_slot_num_;
        static double slot_time_;
        static double start_time_;
        static int active_node_;
        static int* tdma_schedule_;
        int slot_num_; // The slot number it's allocated.
        static int* tdma_preamble_;// The preamble data structure.
```

```
        int slot_count_;
        MolecularData* repository_;
};
```

MolNode.h:

```
class MolNode :  public Node {
        friend class NodePositionHandler;
        friend class NodePositionTestHandler;
public:
        MolNode();
        MolPosition* position();
        MolChannel* channel();
        int receptor_number();
        Molecule* ligand();
        Molecule* receptor();
        double radius();
        MolPhy* phy();
        static int type();
        void set_type(int t);
        MolLink* molLink();
        static double radius_;
        void write();
        static double binding_radius;
protected:
        int command(int argc, const char*const * argv);
        MolChannel* channel_;
        MolPhy* phy_;
        MolPosition* pos_;
        Molecule* ligand_;
        Molecule* receptor_;
        int receptor_number_;
        NodePositionHandler* position_handler;
```

```
        NodePositionTestHandler* position_test_handler;
        Event* position_event;
        static int type_;//it can be diffusion,motor or junction
        MolLink* molLink_;
};
```

MolError.h:

```
class MolErrorModel :  public ErrorModel {
public:
        MolErrorModel();
        virtual void recv(Packet*, Handler*);
        MolErrorUnit unit();
protected:
        virtual int command(int argc, const char*const * argv);
        bool CorruptMolecule();
        MolErrorUnit unit_;// mol error unit in molecule or moleculardata
};
```

MolEnvironment.h:

```
class MolecularEnvironment :  public TclObject {
private:
        static MolChannel* mol_channel;
        vector< Lattice*> container;
        vector< Molecule*> species;
        vector< ReactionChannel*> reaction_channel_;
        latticeMapMapMap lattice_structure;
        long seed;
        Randomizer* random;
public:
        MolecularEnvironment();
        MolecularEnvironment(MolChannel*);
        static void setChannel(MolChannel* m);
```

```
        Lattice* getLattice(MolPosition*);
        Lattice* getLattice(int, int, int);
        Lattice* getLattice(Lattice*, int);
        void setLattice(Lattice*, int, int, int);
        void setLattice(Lattice*);
        void addLattice(Lattice*);
        void addLattices(vector< Lattice*>);
        void print();
        void printOzel();
        void deleteLattice(Lattice*);
        void deleteLattice(Lattice*, int);
        void addSpecies(Molecule* m);
        void addReactionChannel(Molecule*, Molecule*, double);
        void addReactionChannel(ReactionChannel*);
        vector< Lattice*> getContainer();
        vector< Molecule*> getSpecies();
        vector< ReactionChannel*> reaction_channel();
        static double temperature;
        static double viscosity;
        static double lambda;
        static double motorVelocity;
protected:
        int command(int argc, const char*const * argv);
};
```

Lattice.h:

```
class Lattice {
private:
        static MolecularEnvironment* me;
        Lattice* neighbours[6];
        int lattice_ID;
        map< string,int> currentState;
```

```
        map< string,int> tempState;
        int assignLatticeID();
        static int currentLatticeID;
        MolPosition* centre_point;
        int type;//CYTOSOL,MEMBRANE,VITRO,VESICLE
        MolNode* node_;
public:
        Lattice();
        Lattice(MolPosition*);
        Lattice(MolPosition*, MolNode*);
        static void setMolecularEnvironment(MolecularEnvironment* m);
        int getLatticeID();
        void den();
        MolPosition* getPosition();
        void addMolecule(Molecule* m, int number);
        void addMoleculeInitially(Molecule* m, int number);
        void removeMolecule(Molecule* m);
        void addLink(Lattice*, int t);
        Lattice* getNeighbourLattice(int);
        bool operator==(const Lattice& other) const;
        map< string,int> inline getCurrentState();
        map< string,int> inline getTempState();
        bool finalizeState(Molecule*, int);
        void finalizeState(Molecule*);
        void inline setLatticeID(int l);
        void print();
        void printMoleculeInfo();
        void printMoleculeInfoOzel();
        bool moleculeTypeExist(Molecule*);
        int getMoleculeNumber(Molecule*);
        int getMoleculeNumberT(Molecule*);
        void inline setType(int a);
```

```
        int inline getType();
        bool checkReactionPossibility();
        MolNode* node();
        void zeroise();
        void zeroise(Molecule*);
};
```

Molecule.h:

```
class Molecule :  public TclObject {
private:
        long assignMoleculID();
        static long currentMoleculeID;
        int sender_id;
        int receiver_id;
        long moleculID;
        string type;
        double diffusion_time;//diffusion time to diffuse a lattice
        int genus;//RECEPTOR or LIGAND
        double diffusionCoefficient;//D
        double radius;
        static double radii;
public:
        Molecule();
        Molecule(string);
        Molecule(string, double);
        long getMoleculID();
        string inline getType();
        double getDiffusionTime();
        void inline setDiffusionTime(double t);
        bool operator==(const Molecule& other) const;
        void inline setGenus(int a);
        int inline getGenus();
```

```
        void inline setDiffusionCoefficient(double d);
        double inline getDiffusionCoefficient;
        void print();
        static double default_coef;
protected:
        int command(int argc, const char*const * argv);
        MolNode* node_;
};
```

MolecularDataUnit.h:

```
class MolecularDataUnit {
private:
        Molecule* mol_;
        int amount_;
        MolecularDataUnit* next_;
        Lattice* captured_lattice;
public:
        MolecularDataUnit() :  next_(NULL), captured_lattice(NULL){}
        MolecularDataUnit(Molecule* m, int q, Lattice* lat) :
        next_(NULL), mol_(m), amount_(q), captured_lattice(lat){}
        MolecularDataUnit* next();
        MolecularDataUnit* next(MolecularDataUnit* n);
        Molecule* mol();
        int amount();
        Molecule* mol(Molecule* m);
        int amount(int q);
        void setLattice(Lattice* l);
        Lattice* getLattice();
};
```

MolecularData.h:

```
class MolecularData :  public Packet {
```

```
private:
      bool direction_;
      MolecularDataUnit* data_header;
      MolecularDataUnit* last_unit;
      MolNode* node_;
      int com_type;
      double ts_;
public:
      MolecularData(MolNode* n);
      MolecularData(MolNode*, Molecule*, int);
      MolecularData(MolNode*, Molecule*, int, Lattice*);
      void up();
      void down();
      bool direction();
      void set_direction(bool d);
      int get_molecule(Molecule* m);
      void set_molecule(Molecule* m, int i, Lattice* lat);
      void set_molecule(Molecule* m, int i);
      MolecularDataUnit* data();
      void print();
      MolNode* node();
      int comType();
      MolecularData* copy();
      void timestamp(double t);
      double timestamp();
      void zeroise();
};
```

MolLink.h:

```
class MolLink :  public Connector {
public:
      MolLink(MolNode*);
```

```
        void recv(Packet* p, Handler*);
        void send(Packet* p, Handler*);
        double sendingInterval();
        double receivingInterval();
        SendingHandler* shTimer();
        ReceivingHandler* rhTimer();
        PropagationHandler* phTimer();
        Event intr();
        double txtime();
        Lattice* senderConnection();
        Lattice* receiverConnection();
        MolecularEnvironment* molenv();
        MolNode* sender();
        MolNode* receiver();
protected:
        int command(int argc, const char*const* argv);
        static double sendingInterval_;
        static double receivingInterval_;
        Event intr_;
        SendingHandler* shTimer_;
        ReceivingHandler* rhTimer_;
        PropagationHandler* phTimer_;
        MolNode* sender_;
        MolNode* receiver_;
        MolecularEnvironment* molenv_;
};
```

Reaction.h::

```
class Reaction :  public TclObject, public Handler {
public:
        vector< ReactionChannel*> allReactions;
        Reaction();
```

```
        Reaction(MolChannel*);
        void addReactionChannel(ReactionChannel*);
        void addReactionChannel(Molecule*, Molecule*, double);
        void print();
        double get_reaction_time(double tau);
        Event* event_instance();
        virtual void handle(Event*){}
        MolChannel* channel();
        void setChannel(MolChannel*);
protected:
        int command(int argc, const char*const * argv);
        MolecularEnvironment* molenv;
        Randomizer* r;
        long seed;
        MolChannel* channel_;
        double t; /*current simulation time*/
        Lattice* lattice_;//current lattice
        ReactionChannel* reaction_channel_;//current reaction channel
};
```

GillespieReaction.h:

```
class ReactionGillespie :  public Reaction {
public:
        ReactionGillespie():Reaction(){}
        ReactionGillespie(MolChannel* m):Reaction(m){}
        virtual void handle(Event*);
        void gillespie(Lattice*, double);
        void react(Lattice*, double);
};
```

ReactionBerg.h:

```
class ReactionBerg :  public Reaction {
```

```
public:
      ReactionBerg():Reaction(){}
      ReactionBerg(MolChannel* m):Reaction(m){}
      virtual void handle(Event*);
      void bergReaction(Lattice*);
};
```

ReactionNo.h:

```
class ReactionNo :  public Reaction {
public:
      ReactionNo():Reaction(){}
      ReactionNo(MolChannel* m):Reaction(m){}
      virtual void handle(Event*);
      void simpleReaction(Lattice*);
};
```

ReactionChannel.h:

```
class ReactionChannel :  public TclObject {
private:
      Molecule* A;
      Molecule* B;
      double c;
      int reactionNumber;
public:
      Molecule* getA();
      Molecule* getB();
      ReactionChannel();
      ReactionChannel(Molecule*, Molecule*, double);
      void react(Lattice*);
      double getPropensityValue(Lattice*);
      void incReactionNumber();
      int getReactionNumber();
```

```
        Molecule* get_ligand(MolNode*);
        void print();
protected:
        int command(int argc, const char*const * argv);
};
```