

# 國立交通大學

資訊工程系  
碩士論文

在 NCTUns 網路模擬器上支援

Tactical Mobile Ad Hoc Networks



Supporting Tactical Mobile Ad Hoc Networks on

NCTUns Network Simulator

研究生：黃鎮遠

指導教授：王協源 教授

中華民國九十四年六月

在 NCTUns 網路模擬器上支援 Tactical Mobile Ad Hoc  
Networks  
Supporting Tactical Mobile Ad Hoc Networks on NCTUns  
Network Simulator

研究生：黃鎮遠

Student：Chen-Yuan Hwang

指導教授：王協源

Advisor：Shie-Yuan Wang



Submitted to Department of Computer Science and Information Engineering  
College of Electrical Engineering and Computer Science  
National Chiao Tung University  
in partial Fulfillment of the Requirements  
for the Degree of  
Master  
in

Computer and Information Science


June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

# 中文摘要

對於軍事戰術網路的研究者而言，網路模擬器是一個能幫助他們研究與發展下一世代的無線移動傳輸戰略通訊技術以支援未來戰場上的需要的有用的工具。藉由網路模擬器的幫助，研究者可以節省相當多的時間與金錢來建制一個發展的網路環境。此外，使用者可以藉由模擬器在其上研究複雜的或是真實世界上難以架構的網路拓樸。在網路模擬環境上，所有網路上的條件跟參數設定都可以重製，所以使用者可以輕而易舉的重製他們的實驗結果。就因為這樣，所以網路模擬器可以幫助使用者解決或設計網路的問題。NCTUns 網路模擬器已經開發數年了，由於他新穎的設計，它可以提供許多在傳統模擬器上無法做到的獨特優點。

The logo of National Central University (NCTU) is a circular emblem with a gear-like border. Inside the circle, there is a stylized blue and white design that resembles a network or a gear. The year '1896' is printed in white at the bottom of the inner circle.

軍事網路的開發者與技術人員試著尋找用於戰場上通訊的技術與方式。為了能支援這種新型態的軍事無線網路，目前模擬器的設計顯然不符合需求。因此想要在其上作相關的研究，原本的設計勢必要更改。另外，我們提供了一些原本沒有的功能，像是動態控制，動態的呈現，還有障礙物的感測等。由於支援了這些新的功能，大大的提高原有 NCTUns 網路模擬器的能力。

這些新增的功能將能幫助使用者去模擬更類似於真實的軍事戰場項的網路拓樸與型態。

# Abstract

For military network researchers, a network simulator is a very useful tool to help them study or develop new technologies to support next-generation mobile wireless high-capacity tactical communications and to meet future army communication requirements. Researchers can save much time and money required to build a real network environment with a network simulator. In addition, users can study a complicated network or a network that does not exist in the world. In a simulated network environment, all network conditions and configurations are repeatable, therefore researchers can easily repeat their experimental result. As such, they can help researchers design a good network system or solve network problems. The NCTUns network simulator has been developed for many years. Due to its several novel simulation methodologies, it provides many unique advantages that can not be achieved by traditional network simulators.



Warfare planners and tacticians are seeking ways to communicate information on the battlefield. In order to support tactical Ad Hoc network, present simulation engine has to be modified. Besides, we have to add another feature such as dynamical control, dynamical animation, and obstacle sensing. After adding these functions, it highly increases the ability of NCTUns network simulator. These functionalities can greatly help users to simulate a case which is more similar and realistic to the real battlefield.

# Table of Contents

## Supporting Tactical Mobile Ad Hoc Networks on NCTUns Network Simulator

1	Introduction .....	1
1.1	Motivation.....	1
1.2	Organization.....	2
2	Development History .....	3
3	Design Goals.....	5
3.1	Dynamic Control.....	5
3.1.1	Run-Time Get/Set Variable Registration.....	6
3.1.2	Dynamic Movement.....	6
3.1.3	Dynamic Packet Animation Player.....	7
3.2	Tactical/Strategy.....	7
3.3	Feature.....	9
3.3.1	External Control.....	10
3.3.2	Real Time Control.....	10
3.3.3	Real Time Animation.....	10
3.3.4	Obstacle Sensing.....	11
3.3.5	Using All Features and Capabilities of NCTUns Network Simulator.....	12
3.3.5.1	Support for Various Networks.....	12
3.3.5.2	Support for Various Networking Devices.....	13
3.3.5.3	Support for Various Network Protocols.....	13
3.3.5.4	Application Compatibility and Extensibility.....	13
3.3.5.5	User Friendliness.....	14
3.3.5.6	Open System Architecture.....	14
4	Background .....	15
4.1	Related Work.....	15
4.2	Required techniques.....	16
4.2.1	Ad Hoc network.....	16
4.2.1.1	AODV.....	16
4.2.2	Artificial Intelligence.....	17
4.2.2.1	Map representation.....	17
4.2.2.2	Path finding.....	18
5	Design and Implementation.....	20
5.1	Kernel Modification.....	20
5.2	Simulation Engine Modification and Design.....	25
5.2.1	Event Prediction.....	28
5.2.2	Dynamic Communication.....	31

5.2.2.1	Communication with GUI.....	31
5.2.2.2	Communication with Node Control Program.....	33
5.2.3	Obstacle Construction.....	38
5.3	Intelligence Engine.....	38
5.3.1	Map sensing .....	38
5.3.2	Obstacle Detection and Collision Avoidance.....	39
5.3.3	Path Finding.....	42
5.3.4	Event-Triggering System.....	42
5.4	Node Control program.....	43
5.4.1	Layered Design.....	44
6	A Simulation Example.....	47
6.1	Simulation Setup.....	47
6.2	Result.....	49
7	Future Work.....	52
8	Conclusion.....	53
9	Reference.....	54



## List of Figures

Figure 3.2.1: Main state machine of our tactical behavior.....	9
Figure 4.2.2.1: Map representation of Grid, Floor, and Points of Visibility.....	18
Figure 5.1.1: The kernel re-entering simulation methodology.....	20
Figure 5.1.2: Two types communication of Node Control Program.....	23
Figure 5.2.1: The architecture of the NCTUns network simulator.....	25
Figure 5.2.2: The architecture of extension for simulation engine.....	28
Figure 5.2.1.1: A snapshot of time $t_1$ .....	30
Figure 5.2.1.2: At time $t_1$ , engine will insert predict events.....	30
Figure 5.2.1.3: A snapshot of time $t_1+5$ .....	30
Figure 5.2.1.4: At time $t_1+5$ , simulation engine will insert predict events.....	30
Figure 5.2.2.2.1: Communication between Engine and Node Control Program.....	34
Figure 5.2.2.2.2: Communication between Engine and Node Control Program.....	36
Figure 5.2.2.2.3: Communication between Engine and Node Control Program.....	37
Figure 5.2.2.2.4: Communication between Engine and Node Control Program.....	37
Figure 5.3.1.1: Standard coordinate system.....	39
Figure 5.3.1.2: Our coordinate system.....	39
Figure 5.3.2.1: Obstacle detection of simulation engine.....	40
Figure 5.3.2.4: Translate an obstacle into grid of map.....	41
Figure 5.4.1.1: Layer of Node Control Program.....	45
Figure 5.4.1.2: command queue of Node Control Program.....	46
Figure 6.1.1: The scenario of simulation case.....	48
Figure 6.1.2: The scenario of simulation case with obstacle.....	48
Figure 6.2.1: The CPU utilization of simulation case without obstacle.....	49
Figure 6.2.2: The required memory space of simulation case without obstacle.....	50
Figure 6.2.3: The required time of simulation case without obstacle.....	50
Figure 6.2.4: The CPU utilization of simulation case with obstacle.....	50
Figure 6.2.5: The required memory space of simulation case with obstacle.....	51
Figure 6.2.6: The required time of simulation case with obstacle.....	51

## 致謝

感謝恩師王協源教授二年來對我的悉心指導，研究所二年，由於老師給我們的札實訓練，讓我在專業領域上獲益良多，相信以這二年札實的，不論是在工作或研究上都會是重要的基石。

感謝林華君教授以及廖婉君教授撥冗來到交通大學進行口試。

感謝父母親多年來的支持，讓我在求學過程中順暢無後顧之憂，能專心於學業之中。



最後感謝網路與系統實驗室的所有成員，因為有你們的陪伴以及彼此間的鼓勵，讓我在研究所二年不僅學到很多，生活也很充實快樂。



# 1. Introduction

## 1.1 Motivation

Warfare planners and tacticians are seeking ways to communicate information on the battlefield. With the advent of Internet technologies complex systems are becoming more networked and access to information is more critical than ever. The increasing utilization of special operations forces in Ad Hoc dynamic operations poses a need for adaptable communications to support the unit. Effective communication within the unit and how to exchange the critical information with the command center affect the overall outcome of the mission.

Wireless Ad Hoc networks are autonomous, self-organized, self-managed and adaptive. This technology has become increasingly important in communication and networking. Thus, Ad Hoc networks are excellent candidates for military tactical networks. Ad Hoc networks are the key to the army's future combat support capability. For example, several fast moving mobile nodes communicating in a military grade network using partially Ad Hoc -formed wireless access networks

Because military applications exploit the ability of Ad Hoc networks to work in situations where there is no pre-installed infrastructure available, as in combat areas, related research have increased. However, it is impossible for researchers to experiment in the real world. The cost is too high. Therefore, a network simulator is a useful tool that can help researchers to study a complicated network or a network that does not exist in the world.

Due to the limitation of the original NCTUns network simulator, it can't support above military Ad Hoc network study. In order to support this kind of research, we need to modify the NCTUns network simulator.

## 1.2 Organization

The organization of this thesis is as follows. In Chapter 3, we attempt to illustrate the goal and the feature of our new design. In Chapter 4, we will survey some techniques which are related to our topic such as AI, AODV, and graphics theory, etc. In Chapter 5, we would like to focus attention on the architecture of our design. In this chapter, we discuss the core of our Simulation Engine. In addition, we will introduce the implementation of Node Control Program. In Chapter 6, we will illustrate a simulation case to analyze its performance. In Chapter 7, we conclude our work. In Chapter 8, we point out the issues to be addressed in the future.



## 2. Development History

The NCTUns network simulator is a high-fidelity and extensible network simulator capable of simulating various devices and protocols used in both wired and wireless IP networks. It is a useful tool that can help network researchers to design a complicated network system or solve their problems. Its core technology is based on the simulation methodology invented by S.Y. Wang at Harvard University in 1999 [1, 2]. Due to this novel methodology, the NCTUns network simulator provides many unique advantages that cannot be achieved by traditional network simulator such as OPNET [3] and ns-2 [4].

The predecessor of the NCTUns network simulator is the Harvard network simulator [5], which was authored by S.Y. Wang in 1999. Due to the limited functionalities of the Harvard network simulator, we need to overcome and solve drawbacks and add some features and functions to it. For these reasons, after joining National Chiao Tung University (NCTU), Taiwan in February 2000, Wang designed a new simulation methodology for the NCTUns 1.0 network simulator. Recently, we also release NCTUns 2.0.

The NCTUns network simulator uses a distributed architecture to support remote simulations and concurrent simulations. It also uses an open-system architecture to enable protocol modules to be easily added to the simulator. In addition, it has a fully-integrated GUI environment for editing a network topology and specifying network traffic, plotting performance curves, configuring the protocol stack used inside a network node, and playing back animations of logged packet transfers.

Furthermore, Wang proposes an approach to apply discrete event simulation to the NCTUns 1.0 network simulator to speed up its simulation speed [6]. The Harvard network simulator used a time-stepped method to implement its simulation engine. As such, its simulation speed is very slow. To overcome this problem, the NCTUns 1.0 applied the event-driven [2] approach to its simulation engine. As such, its simulation speed is much faster than the Harvard network simulator.

Now, we want to modify present simulation engine to support tactical Ad Hoc network. Besides, we have to add another feature such as dynamical control, dynamical animation, and obstacle sensing. After adding these functions, it highly increases the ability of NCTUns network simulator. These functions can greatly help users to simulate a case which has extremely realistic visualizations of communication networks.

Because the Harvard network simulator can only run on the FreeBSD platform, promoting their uses has some difficulties. Now, NCTUns network simulator can run on Linux 2.6.x. Of course, all advantages and features of the FreeBSD version of NCTUns 1.0 network simulator will be reserved and even improved during the porting to the Linux platform. In addition, several new network types are implemented into the Linux version including traditional optical network, optical burst network, GSM/GPRS cellular network, etc.

Due to our continuous improvement, we had released NCTUns 2.0 on 11/01/2004 .We plan to release the Linux version of NCTUns 3.0 network simulator soon. At that time, we expect that more and more people or organizations will use our network simulator.

### 3. Design Goals

In this chapter, we attempt to illustrate functions and goals which we want to support for original NCTUns network simulator. Then we will discuss what the NCTUns network simulator lacks if we want to achieve our goals.

First, we want to use a simple way to support the dynamic control in NCTUns network simulator. We will discuss further what the dynamic control means in the later section. Dynamic control can simply be defined as dynamical input. Second, we hope our simulation results can be immediately outputted during the simulation time, or it may be nearer the truth to say that we can immediately get the location of each node and the animation of execution result. To sum up, we can easily define our requirements as two mainly parts, dynamic input and dynamic output.

In the original design of NCTUns network simulator, simulation results do not present until NCTUns network simulator finishes simulating the whole simulation case. Besides, all movements of nodes in the simulation have to be decided before executing this simulation case. This means that it is impossible to dynamically change any node's moving path during the simulation time. According to our previous description about our requirement, we have to modify the original design and architecture of NCTUns network simulator.


#### 3.1 Dynamic Control

Dynamic control is a kind of ability that can allow a network simulator to control nodes and get status of nodes dynamically. This includes dynamically changes and retrieves node's movement and status. Some of functions already exist and some need to be added or improved.

### 3.1.1 Run-Time Get/Set Variable Registration

Sometimes it is very useful to observe the status of a variable, a node, or a protocol while a simulation case is running. For example, a user may be interested in seeing how the current queue length of a FIFO queue varies during a simulation. To support this functionality, a module developer should register these variables with the simulation engine so that they can be accessed during a simulation. The simulation engine provides the macro *EXPORT()* to support this functionality. By using this method, user can dynamically set or get some node's status. This function has existed on the NCTUns network simulator and works well during simulation.

### 3.1.2 Dynamic Movement



In a real-world network environment, a mobile node or a GPRS phone may move. To specify a mobile node or a GPRS phone's moving path, a user can construct the whole moving path. A moving path is composed of a sequence of turning points and segments. After a moving path is constructed, any of its turning points can be easily moved to any place to adjust the shape of the path. Each turning point is represented by a grey dashed square box and contains the (X-loc, Y-loc, arrival time, pause time, moving speed to the next point) information. [8]

By using this mechanism, we can make mobile node move in a simulation. However, moving path need to be decided in advance, and it can not be changed during the simulation time. Because of this limitation, we have to modify original mechanism to meet our requirement. We hope that the node's movement can be dynamically changed during the run-time.

### 3.1.3 Dynamic Packet Animation Player

After a user's simulation job is finished, the generated simulation results will be automatically transferred back to the GUI program and then saved in the user's local hard disk. Suppose that the simulation case's topology file is named "test.tpl." Then the name of the resulting packet animation trace file will be "test.ptr." Later on, when the user wants to do post analyses about the simulation results, he or she can use the "Packet Animation player" to play back the animation.[8] Though this is a very useful feature for both education and research purposes, its drawbacks is that it can only playback after simulation job is finished. According to our new requirement for tactical Ad Hoc network, we often need to adopt various strategies or tactics according to all nodes' current status. If node's location can be run-time generated as well as be played back after simulation finish, it can play as feedback for dynamic control mechanism to dynamically revise its new tactic.

In addition, the dynamic control can not only be a Node Control Program but also be a human's extern control. (Node Control Program is a child process forked by engine and embedded in the node.)

## 3.2 Tactic/Strategy

For the behavior of computer-controlled characters to become more sophisticated, efficient algorithms are required for generating intelligent tactical decisions. [7]

As we mentioned before, dynamic control can be a Node Control Program which is embedded in a node to control node's behavior according to node's status. Therefore, we have to provide a simple tactic to make Control Node Program follow. In the later chapter we shall try to give a more precise account of how to design a

tactic of Control Node Program. In addition, we will illustrate the difficulty which we encounter and corresponding solutions. For the present, we shall confine our attention to the behavior of tactic.

This tactic focuses on a problem common to many strategic war situations: determining how one might engage the enemy. While this might seem straightforward enough to the human controller, the computer AI have a little bit harder time accomplishing this.

When the AI operates in squads, it can do a lot for a tactical combat. The squad's behavior and communications construct a complicated tactic. It is easy to answer why squad AI should be part of a combat system. However, it is not easy to answer how to add squad AI to the system.

Squad tactical AI can be so elaborate that can be studied and written separately as a topic. Here, we shall focus on an easy and simple tactic, because we only want to prove that our expansion of simulation engine can work well. The complexion of the tactic is not the criterion and factor to judge whether simulation engine can work correctly, because the mechanism of communication between engine and node control program is independent of Node Control Program's tactic. The mechanism of communication between them is only a middle layer which simply provides an abstract interface for upper layer, therefore, whether upper layer is a Node Control Program or an extern AI control system or even a human's control, there is no difference. To put it in the other way round, the tactic of Node Control Program is merely a way to examine whether our modification of simulation engine can work.

Let us return to our main subject of what our tactic is. A squad can consist of several members each has its goals. These members accomplish its goals through coordinated actions. Squads typically have a leader and some of its member, they select and execute certain maneuvers to accomplish its goals to chase an enemy. A



success maneuver involves a lot of factors; one of them is correct and available communication between members.

Figure 3.2.1 sketches the behavior of our tactical strategy. It mainly consists of three state, scout, report and chase. In our demo case, there are three different roles of nodes separately; however, their behaviors are similar. Despite of enemy, member of squad and leader of squad, they all have scout state in common. Then, each of nodes will transit to different state based on their role and situation. All three kinds of node are initial in the Scout State, the enemy always runs away, if it sees any one who want to chase him, he will run away from the nearest chased one. The leader and members of squad are initially in the SCOUT STATE, then if they encounter enemy, members will report to leader and wait leader’s command, but leader will directly notify other members directly while he encounters enemy.

Below is the main stage of a maneuver while they execute a tactic, however, there are still other details which we shall have more to say about in later chapter.

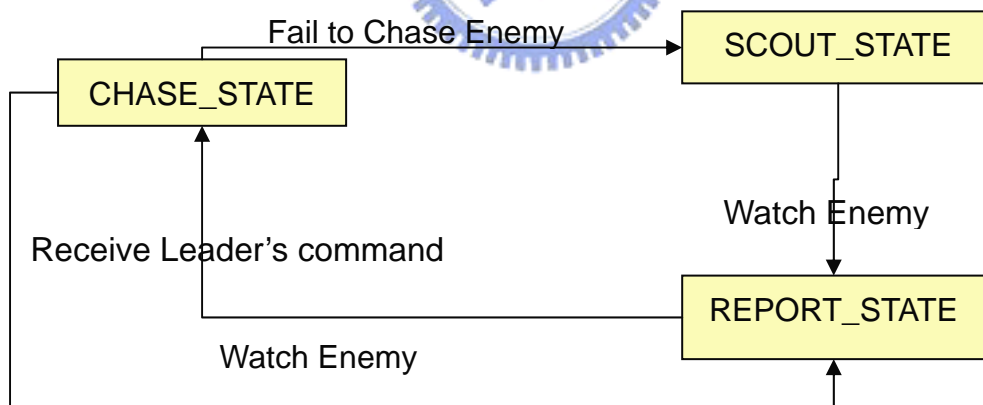


Figure 3.2.1 Main state machine of our tactical behavior


### 3.3 Feature

In this section, we will introduce and describe our new features of simulation engine.

### **3.3.1 External Control**

Besides internal control such as Node Control Program which forks by engine and executes with simulation engine, our new design make simulation engine accept external command to control the node's movement of simulation. Because of middle abstract layer of our design, it makes external control easier to carry out. Besides, it can accept any types of command to control node. For example, human can take advantage of GUI or design their Control Node Program to control node. Even, they can design their tactical engine to replace original role of GUI to communicate with simulation engine.

### **3.3.2 Real Time Control**



In previous design of simulation engine, you can change node's movement; however, you have to decide in advance. After our modification, now, it can real time accept extern or internal command to control node's movement dynamically. This means that you can decide node's situation at any time to decide what you want node to do. In the military scenario, real time control is very important, because it is so complicated that we can't decide every thing previously. If we can real time accept commands to control node's behavior, this feature highly increases the ability of simulation engine.

### **3.3.3 Real Time Animation**

As mentioned above, after a user's simulation job is finished, the generated simulation results will be automatically transferred back to the GUI program and then

saved in the user's local hard disk. Now, our new design make simulation engine dynamically output its result during it is executing.

### 3.3.4 Obstacle Sensing

In the real world, not all fields are open space for wireless signal. Some may have high mountains or tall buildings blocking wireless signal's propagation. In some researches, we may want to purposely add wireless signal obstacles to the open field to block wireless signal at some places in the field. In original design of NCTUns network simulator, a wall is a sequence of segments that completely block wireless signal and can be specified using the same way for specifying a mobile node's moving path. With walls, a user can simulate a more complicated and interesting field setting. This can facilitate testing wireless network and protocol performances (e.g., handoff) under a more realistic field setting.

Original design of wall provides the functions of wireless signal obstacles, but it still calls for further improvement. Let us now look at original design of wall in detail; because the moving path of node is described before simulation execution, no one will let the moving path go through wall. However, if someone really does this, what will happen? The answer is that it will directly go through wall without sensing the existence of wall. The reason is that we only take wireless signal obstacles into consideration without detecting collision of obstacles. If we want to support tactical Ad Hoc network on simulation, obstacle sensing is an important and indispensable ability.

In addition, we also need other kinds of wall to reach our requirement. First, we need the wall that can block node's movement without block node's signal, because if node's signal is blocked by wall, it will highly decrease the function of

communication of Ad Hoc network. Second, we also need the wall which can obstruct node's field of vision. Furthermore, we can optionally choose composition of above features. For example, we may need the wall which can block node's movement and obstruct node's vision.

We may note, in passing, that the single-hop connectivity and multi-hop connectivity calculations provided in a mobile node's dialog box take the existence of walls into account. In addition, the God routing daemon for WLAN Ad Hoc mode mobile nodes takes into account the existence of walls as well. Therefore, if we use the wall which doesn't block node's signal, we have to modify original consideration of limitation of single-hop connectivity and multi-hop connectivity calculations provided in a mobile node's dialog box.

### **3.3.5 Using All Features and Capabilities of NCTUns Network Simulator**



The NCTUns network simulator is a high fidelity and extensible network simulator. It has many unique advantages that cannot be achieved by traditional network simulators. If we support simulation with tactical Ad Hoc network, all of the capabilities that NCTUns has can still be used. By these advantages, the NCTUns network simulator will become more powerful and can provide many capabilities that simulation can not support. For the present, we shall look closely at some capabilities and features of the NCTUns network simulator.

#### **3.3.5.1 Support for Various Networks**

It can simulate wired networks with fixed nodes and point-to-point links. It can also simulate wireless networks with mobile nodes and IEEE 802.11 (b) wireless

network interfaces. For IEEE 802.11 (b), both the Ad Hoc and infrastructure modes are supported.

### **3.3.5.2 Support for Various Networking Devices**

It can simulate various networking devices such as Ethernet hubs, switches, routers, hosts, IEEE 802.11 wireless access points and interfaces, etc. A more realistic 802.11 (b) wireless physical module that considers the used modulation scheme, the received power level, the noise power level, and the derived BER is provided

### **3.3.5.3 Support for Various Network Protocols**

Because of the module-based platform, users can easily develop and add new protocols on the NCTUns simulator. Now, it can simulate numerous protocols such as IEEE 802.3 CSMA/CD MAC, IEEE 802.11 (b) CSMA/CA MAC, the learning bridge protocol used by switches, the spanning tree protocol used by switches, IP, Mobile-IP, RTP/RTCP, RIP, OSPF, UDP, TCP, HTTP, FTP, Telnet, etc. More protocols and devices are for other types of networks such as GSM/GPRS cellular networks and optical networks have been developed.

### **3.3.5.4 Application Compatibility and Extensibility**

All real-life existing or to-be-developed UNIX application programs can be run on a simulation network to generate realistic network traffic. Users do not need to modify these programs. These programs can be easily run on a simulated network as long as these UNIX programs can be correctly run on the real-world network. In addition, all real-life existing UNIX network configuration tools (e.g. route, ifconfig,

netstat, tcpdump) can be run on a simulated network to configure or monitor a simulated network. Users can easily use these tools provided by a UNIX system.

### **3.3.5.5 User Friendliness**

NCTUns provides an integrated and professional GUI environment in which users can easily conduct network simulations. All settings and configurations can be easily set up through GUI. This includes drawing network topologies, configuring the protocol modules used inside a node, specifying the initial locations and moving paths of mobile nodes, plotting network performance graphs, playing back the animation of a logged packet transfer trace, etc.

### **3.3.5.6 Open System Architecture**

By using a set of module APIs that are provided by the simulation engine, a protocol module developer can easily implement his or her own protocol and integrate it into the simulation engine. For example, user can easily develop and test his or her routing protocol used by Ad Hoc mode mobile node in our simulator.

## 4. Background

In order to support simulation with tactical Ad Hoc network, it involves a lot of techniques. Not only network but some artificial intelligence is required; furthermore, it consists of a great deal of graphic technique as well as mathematics. In this chapter, we will illustrate the related work and briefly introduce the techniques which we had adopted.

### 4.1 Related Work

In this section, we will discuss some related work.

The QualNet [9] network simulation software is a commercial network simulator. It uses architecture of layer model. This is similar to one used in physical networks with well-defined APIs between neighboring layers. Besides, it provides capability for network emulation. It also supports military network. Furthermore, it has rich 3D visualization and simulates networks of 250+ nodes in real-time.

High Capacity Tactical Network (HCTCN) [10] is a project whose main objectives are to develop and demonstrate new technologies to support next-generation mobile wireless high-capacity tactical communications and to meet future army communications requirements.

In addition, Routing is a primary technical challenge in designing a tactical Ad Hoc network. “Next-Generation Tactical Ad Hoc Mobile Wireless Networks” [11] illustrate a combat network system based on associativity-based routing (ABR), a best-effort routing protocol licensed to Northrop Grumman.

## 4.2 Required Techniques

In this section, we will devote more space to briefly discuss all of techniques which we will use.

### 4.2.1 Ad Hoc network

Ad Hoc wireless network is a collection of two or more devices equipped with wireless communications and networking capability. Such devices can communicate with another node that is immediately within their radio range or one that is outside their radio range. [12] For the latter case, an intermediate node is used to relay or forward the packet from the source toward the destination.

#### 4.2.1.1 AODV



The requirement for mobile tactical network is the capability to seamlessly connect to a local network anywhere in the deployed tactical network. Mobility requirement for tactical forces is the need for rapid setup and teardown of communications networks. Tactical network applications differ from fixed plant applications in that tactical networks are mobile. Tactical units rarely stay in the same location for the duration of an operation. Therefore, networks that require vast amounts of cabling are often impractical for use in a tactical operation. [13] [14]

MANET is an autonomous system of mobile routers and associated hosts connected by wireless links. The routers are free to move randomly and organize themselves arbitrarily, thus allowing the network's wireless topology to change rapidly and unpredictably. There are numerous routing protocols. Routing is a primary technical challenge in designing a tactical Ad Hoc network. Here, we only



pay attention to AODV. The reason is that AODV has existed in NCTUns network simulator. Besides, AODV maintains routes for as long as the route is active, it can be more suitable in mobile environment than others existed routing in NCTUns network simulator.

The Ad Hoc On Demand Distance Vector (AODV) routing algorithm is a routing protocol designed for Ad Hoc mobile networks. AODV is capable of both unicast and multicast routing. It is an on demand algorithm, meaning that it builds routes between nodes only as desired by source nodes. It maintains these routes as long as they are needed by the sources. Additionally, AODV forms trees which connect multicast group members. The trees are composed of the group members and the nodes needed to connect the members. AODV uses sequence numbers to ensure the freshness of routes. It is loop-free, self-starting, and scales to large numbers of mobile nodes

## 4.2.2 Artificial Intelligence



In this section, we attempt to illustrate some techniques related to artificial intelligence

### 4.2.2.1 Map representation

Practically, there are several patterns to transfer a world in the terrain into a map. According to [15], [16] there are several kinds of representation, Rectangular or Hexagonal Grid, Floor Representation and Points of Visibility. Each has its pros and cons.

Grid:

A grid map uses a uniform subdivision of the world into small regular shapes called "tiles". Common grids in use are square, triangular, and hexagonal.

Grids are simple and easy to understand

Floor:

An artist or level designer creates a polygonal floor representation that is used to exclusively for pathfinding.

Points of Visibility:

Points are placed at convex corners on the world. Each point is then connected to all other points that it can “see”.

A simple diagram of each kind of above map is provided in Figure 4.2.2.1.1.

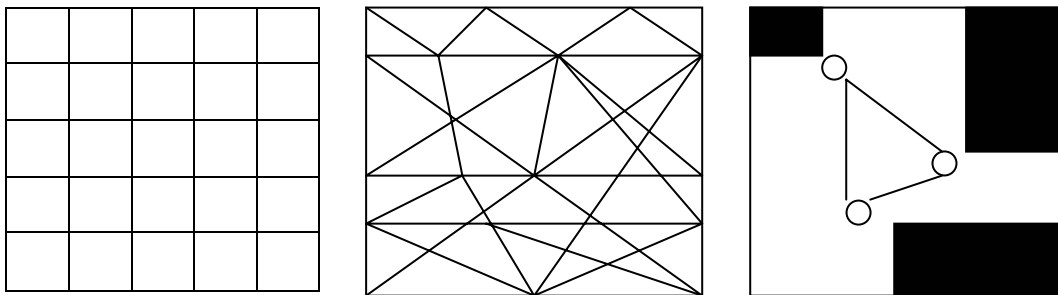


Figure 4.2.2.1 Map representation of Grid, Floor, and Points of Visibility

## 4.2.2.2 Path finding

### A\* path finding

The A\* algorithm is used to find a path between two positions on a map. A\* is a famous algorithm which widely applies in most of game, therefore there are many introductions and discussion even modification about how and what A\* is. [16][17][18][19]

A\* is a power algorithm for path finding, however, it is not a trivial algorithm, and it is extremely difficult to debug if many of the optimization have been incorporated. Therefore, there is a tradeoff between optimization and coding. [15]

## **A\* path finding optimization**

There is no doubt about that A\* path finding algorithm is a time-consuming methods. Since there is a lot list of optimization to improve this algorithm, we also adopt some advices and steps to avoid the drawback of A\* algorithm and increase performance. Our modification of design will be taken up in the latter chapter. For the present, it may be useful to look more closely at why A\* is slow and some way to speed up it.

Steve Rabin, in “A\* speed Optimization” [15], lists a lot of methods related to decrease time cost of A\* algorithm. They mainly fall into two categories, search space optimization and algorithmic optimization. The former can adopt map representation which we mention before, and the latter will be reached by following strategies.

There is no disagreeing on the point of that heuristic cost plays an import and magical role in A\* search. Many researches also reveal that by overestimating the heuristic can result in better and faster effect [15]. Since A\* consumes a large of memory in the progress of search, each node of map contains data structure which requires in the execution of search. But not all the time the search will involves all of the nodes in the map, most of nodes there unused most of time. There is no reason to couple the path finding node data with the search space. This solution reduces unnecessary memory overhead and speeds up search speed. Besides, we can pre-allocate a sufficiently large block of memory that can be recycled for each search. In the progress of A\* search, Open list tends to get large, and it retrieves nodes from Open list frequently, the node to extract is the node with the lowest total cost. Therefore, the best way to store the Open list is to keep it stored as a priority queue. Because binary heap has a property that root always has lowest value, it meets the requirement of Open list's operation. Besides, with this property, insertions and extractions takes only  $O(\log n)$ .

## 5. Design and Implementation

In this chapter, we will explain and discuss how to let the NCTUns network simulator is equipped with previously described abilities and features. In addition, we also introduce what kinds of design are proposed and how the original simulation is modified. After reading this chapter, readers will clearly understand our designed architecture.

### 5.1 Kernel Modification

The NCTUns network simulator is based on a new simulation methodology -- the kernel re-entering simulation methodology. It uses an existing real-world FreeBSD/Linux protocol stack to provide high-fidelity TCP/IP network simulation results. The Figure 5.1.1 helps to illustrate this concept.[21]

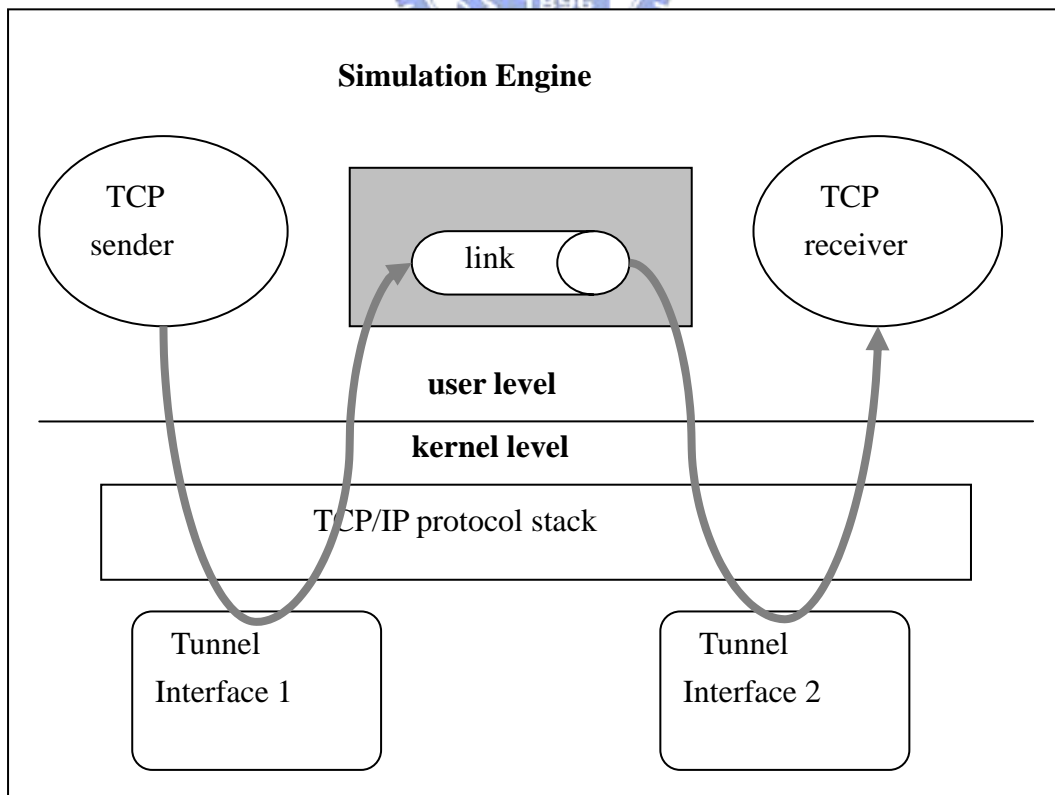


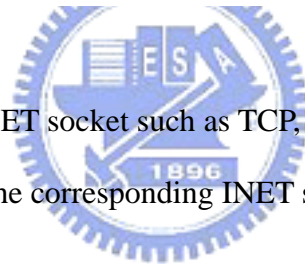
Figure 5.1.1 The kernel re-entering simulation methodology

According to the description of [21], when a process was forked by the simulation engine, the simulation engine uses system call to store the simulated node id into the process handler. This information allows the kernel to know whether a process is used by a simulation or not. The structure *task\_struct* is modified as below:

```

struct task_struct {
    ...
    pid_t          pid;
    ...
    struct signal_struct *signal;
    ...
//NCTUNS
    /* record the process belongs to which node */
    unsigned int   p_node;
//NCTUNS
}

```



If a process creates an INET socket such as TCP, UDP, RAW socket, we should also register the node id into the corresponding INET socket structure, which is structure *sock*.

```

struct sock {
    __u32          daddr;
    __u32          rcv_addr;
    ...
//NCTUNS
    u_int32_t      nodeID; /* record the process belongs to which node */
    unsigned short sk_vport; /* virtual port number */
    struct pmap    *pmap; /* point to port mapping information */
//NCTUNS
}

```

For the datagram INET socket such as UDP and RAW socket, we store the node id into the INET socket structure (*sock*) when a process calls the *socket()* system call:

```

Asmlinkage long sys_socket (int family, int type, int protocol)
{
    int          retval;
    struct socket *sock;

    retval = sock_create(family, type, protocol, &sock);
    ...
//NCTUNS
    /* If current process belongs to a simulation,
       we should store node id into sk. */
    if (current->p_node > 0) {
        sock->sk->nodeID = current->p_node;
    } else {
        sock->sk->nodeID = 0;
        sock->sk->sk_vport = 0;
    }
//NCTUNS
    ...
}

```

With the node id information, we can correctly translate the IP address in the kernel. For more details of the S.S.D.D address format, readers can refer [21] and [20].

Besides, according to this information, kernel can determine whether the port number translation has to be done or not. The port number translation is a mechanism to translate a real port number to a virtual port number. [20][21] Additionally, *nodeID* and *p\_node* are also used to identify whether the timer of this execution is based on a virtual-time timer or not. After realizing above short descriptions about how the kernel is modified, this will lead us further into the consideration of new design for our requirements.

Like other traffic generators, Node Control Program is also a process forked by the simulation, and one of its jobs is to deliver god information between the Node and the simulator engine. Another objective is to control the node's behavior. During simulation, all information, such as node's location, speed, angle and so on, is keep in the simulation engine; hence, Node Control Program has to use IPC to obtain these information. Here, we adopt the UNIX socket as our choice, because it is easy to expand and debug.

Besides, Node Control Program is also responsible for communicating with

other node's Node Control Program. As original traffic generator in simulation, this kind of communication is based on virtual clock, because it is used to simulate tactical Ad Hoc network communication instead of acquiring god information between them.

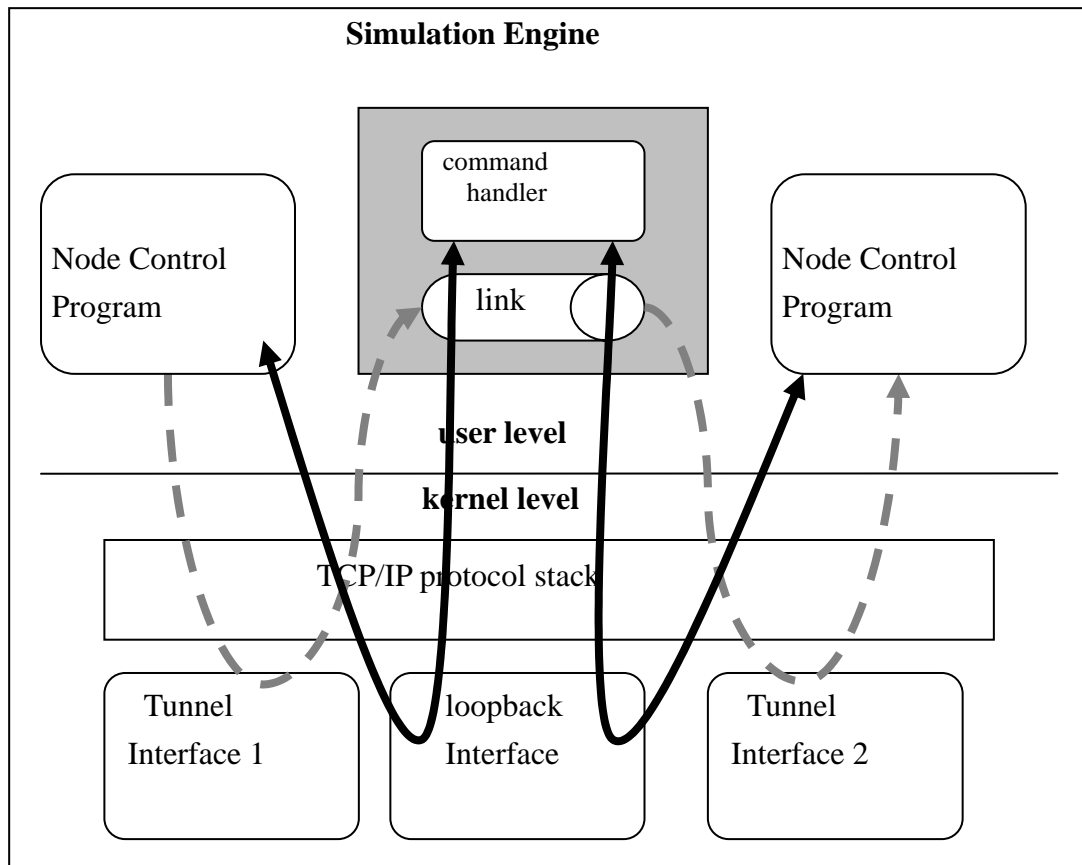


Figure 5.1.2 Two types communication of Node Control Program

Figure 5.1.2 clearly shows the difference of communication between these two types. The shading broken arrow in figure 5.1.2 is used to communicate between Node Control Programs, and black arrow is used to communicate god information between Node Control Program and simulation engine.

The question now arise: Node Control Program is a child process which was forked by the simulation engine, as we mentioned ahead, results in that all socket executes on Node Control Program will based on virtual time and IP address and port number will be translated as previously described. Therefore, the communication

between Node Control Program and simulation engine can't work correctly. The reason is that the communication between them is based on general IPC socket and its IP address is local loopback address instead of our IP address format, but all packets' address will be translated into S.S.D.D format. Due to the modified address and port number, packets cannot correctly reach its destination.

The solution of this problem is to clean nodeID we mark in kernel by calling system call `sys_NCTUNS_misc`. In order to achieve this, we add another system call `sys_NCTUNS_cancel_socknodeID` to erase it if we want to create the socket whose IP address is not S.S.D.D format and based on real time clock.

```

asmlinkage int sys_NCTUNS_cancel_socknodeID(int fd)
{
    struct socket *sock;
    struct sock *sk;
    struct tcp_opt *tp;
    int err,i;

    sock = sockfd_lookup(fd,&err);
    if(!sock)
        return 0;
    sk = sock->sk;
    sk->nodeID = 0
    if (sk->sk_protocol == IPPROTO_TCP){
        struct tcp_opt *tp = tcp_sk(sk);
        tp->nodeID = 0;
        printk("In sys_NCTUNS_cancel_socknodeID\n");
    }
    return(1);
}

```

The `sys_NCTUNS_misc` system call with parameter `NSC_REGPID` is used to enable the process that is forked by the simulation engine to know that it belongs to which simulated node. In opposition to `sys_NCTUNS_misc` system call, `sys_NCTUNS_cancel_socknodeID` is used to cancel the nodeID which was set by



sys\_NCTUNS\_misc.

Besides, we extend the functionality of system call sys\_NCTUNS\_misc with parameter NSC\_PIDTONID to let Node Control Program acquire its node id according to the process id.

## 5.2 Simulation Engine Modification and Design

The NCTUns network simulator uses a distributed architecture to support remote simulations and concurrent simulations. Users can easily add protocol modules to the simulator. The simulation engine is the core of the NCTUns network simulator. In figure 5.2.1, the architecture of the Simulation Engine is described. We need not elaborate on the detail of simulation engine; it is treated much more adequately in [20] and [21]. Here, we only need to mention the extension and improvement of the Simulation Engine.

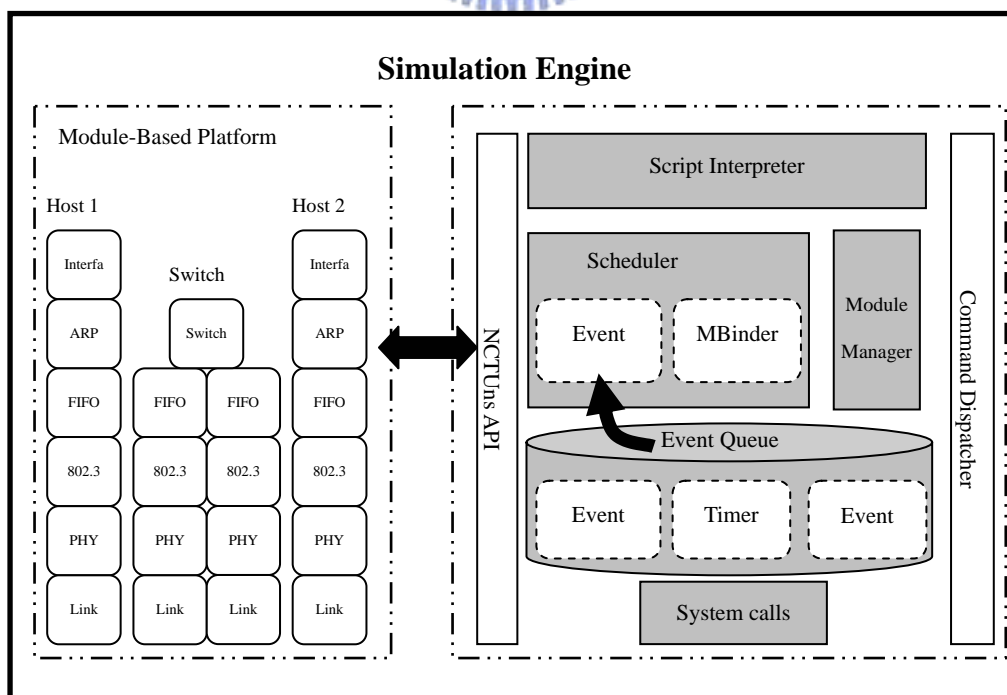


Figure 5.2.1 The architecture of the NCTUns network simulator

Since the new addition of the Simulation Engine is fully discussed in the following section of this chapter, we only briefly explain what we had added and modified for the Simulation Engine here.

We can simply divide our modification of the simulation engine into several components:

(I) Map Construction System:

The Map Construction System reads the \*.obs file which is generated by GUI while users draw walls on the simulation case. Map construction will represent the whole terrain as rectangular grid and identify if grid belongs to obstacle grid or not.

(II) Obstacle Detect APIs:

The Obstacle Detect APIs is composed of various mathematic functions. All of engines can ask for the simulation engine's services via these APIs such as calculating the distance between obstacle and node, predicating collision time of node into obstacle, etc.

(III) Ptr & Moving path packer

It is necessary for simulation engine to dynamically generate the real time result to GUI. Users can choose whether to dynamically generate real time ptr and movement results or not.

(IV) Command handler

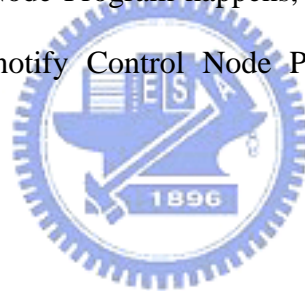
The command handler can mainly be divided into two parts: the command input parser and notification for Node Control Program. The former can receive commands from inner command requesters like Node Control Program or external ones such as

GUI or an independent tactical strategy system. The latter is responsible for notifying Node Control Program if simulation wants to send messages to it.

#### (V) Trigger System

The main job of the trigger system is to check register event which was registered by control node program in the event queue. The register event is different form original event in the Simulation Engine. The original event is used to encapsulate messages that are exchanged between protocol modules.

The trigger system always iterates through unexpired register events to check whether the current situation matches register event's condition or not. If interesting event registered by Control Node Program happens, the trigger system will execute corresponding reactiona or notify Control Node Program their interesting event happening.



#### (VI) Node Control Program

Node Control Program acts as an agent for node to communicate with the Simulation Engine. When a Node wants to send a command to the simulation engine such as change speed, turn its direction, stop movement or register interesting event etc, the Node Control program should send these requirement to the command handler. Then the command handler parses the command to the simulation engine or trigger system according to type of requirement. After the interesting event happens, command handler will send notification back to the Node Control Program and the Node control Program will respond via their tactic or strategy.

Figure 5.2.2 we can see the whole architecture of components we mentioned

above and relations between them. The part of core engine & scheduler represents the original simulation engine in Figure 5.2.1

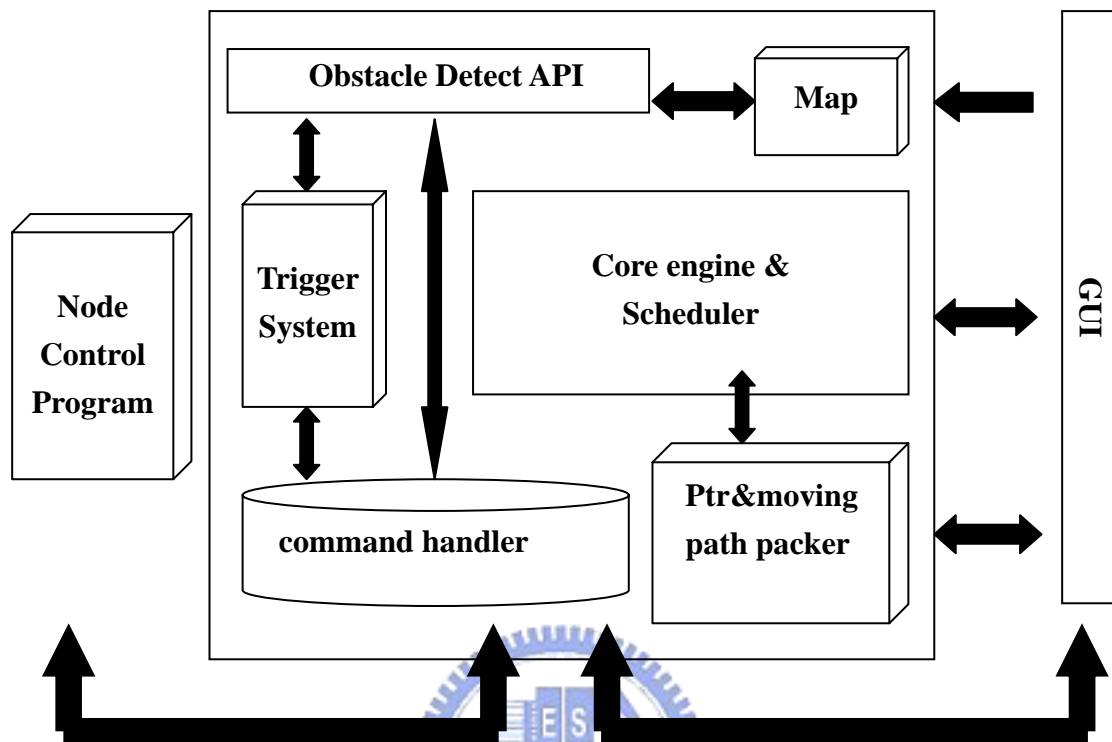


Figure 5.2.2 The architecture of extension for simulation engine

## 5.2.1 Event Prediction

First of all, let us consider one scenario, for example, in a simulation case; there are some obstacles around nodes. If a node collides into obstacle, we have to let it stop instead of going through it.

Before we come to a closer discussion of this scenario, a few remarks should be made concerning how scheduler of Simulation Engine works. The scheduler in the Simulation Engine maintains a system virtual time; all events will be triggered based on this virtual time. The scheduler always selects the event or timer which has the smallest timestamp to execute. In the meantime, the scheduler will advance the simulation time to the timestamp of the event. Viewed in this sight, the Simulation Engine only stays on the timestamp that event will execute. To put it the other way, if

there is no event exists during this period of time, nothing will be done.

Having observed the simulation engine's character of event trigger, we may now turn to the real subject of why we need event prediction. Due to simulation only stays on the timestamp when event will be executed. In view of this, considering the scenario quoted above, if we want to let engine check whether something happens or not in the future, we shall have to predict when this will happen and insert a check event with the timestamp equaling checking time.

We will take a complex example to illustrate this concept. Consider a scenario in Figure 5.2.1.1, there are four nodes (A~D) and nine obstacles (a~f). It is a snapshot at timestamp  $t_1$ . Assuming that Node A, B, C, D will collide into obstacle f,h,d,b after 16,7,25,5 sec separately if nothing influences them and they don't change speed or direction. Then, we have to insert four prediction events with timestamp  $t_1+5$ ,  $t_1+7$ ,  $t_1+16$ ,  $t_1+25$  respectively. (See Figure 5.2.1.2) After 5 sec, current timestamp is  $t_2$  and its scenario becomes Figure 5.2.1.3. The similar thing is done repeatedly like at timestamp  $t_1$ . It will insert three events with timestamp  $t_2+2$ ,  $t_2+11$ ,  $t_2+20$ , if noting cause them to modify their process of movement.

In fact, this method has some shortcuts, and how to improve will be presented in the next chapter.

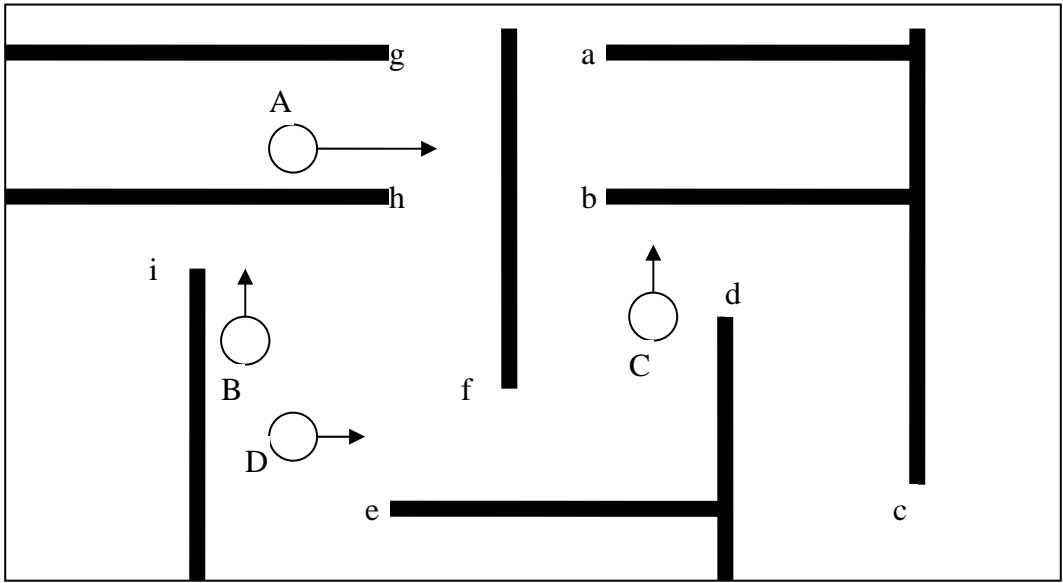


Figure 5.2.1.1 A snapshot of time  $t_1$

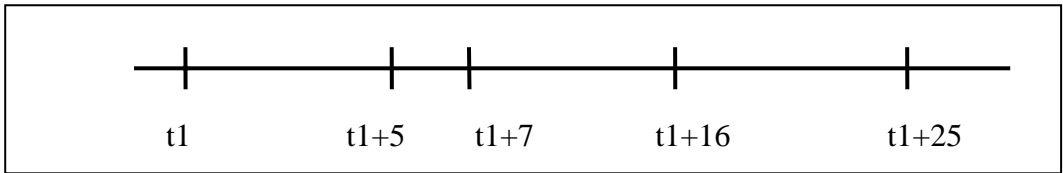


Figure 5.2.1.2 At time  $t_1$ , simulation engine will insert predict events.

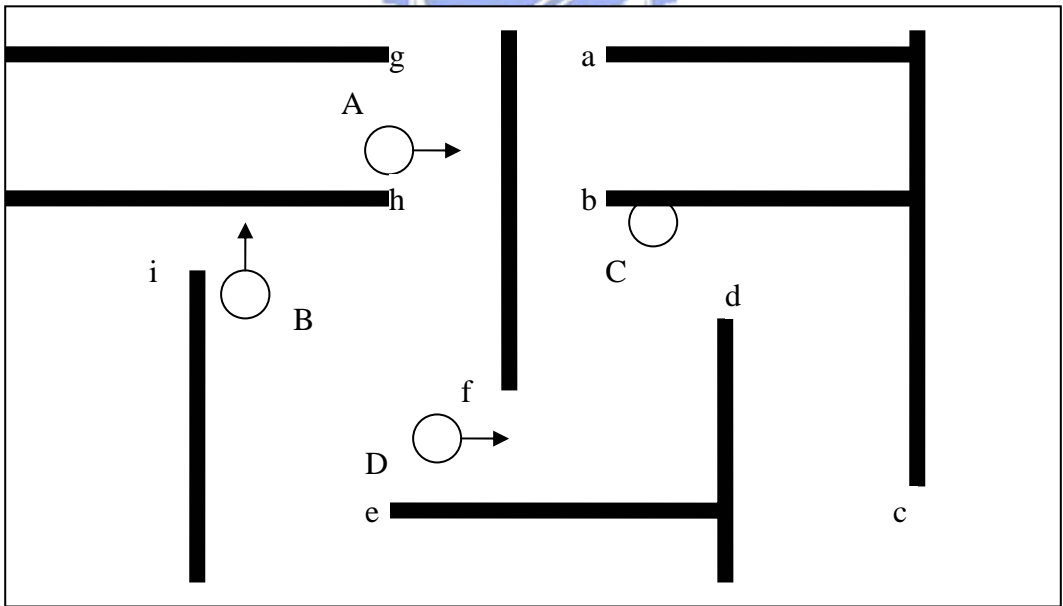


Figure 5.2.1.3 A snapshot of time  $t_1+5$

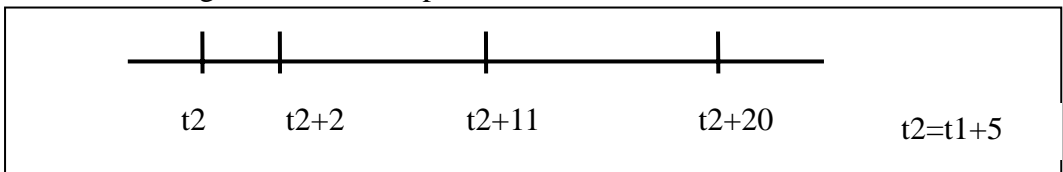


Figure 5.2.1.4 At time  $t_1+5$ , simulation engine will insert predict events.

## 5.2.2 Dynamic Communication

In the preceding chapter we pointed out that dynamic control is one of our design's features. This is composed of the dynamic communication with GUI and with Node Control Program. For the present, let us look closely at how to implement it and problems which we encounter and their corresponding solutions.

### 5.2.2.1 Communication with GUI

Owing to receiving real-time commands from GUI and sending real-time information to GUI; we have to establish a communication between the Simulation Engine and GUI. In the previous design of NCTUns 2.0, Coordinator plays a role of translator which used to exchange message between the Simulation Engine and GUI. When the simulation engine process is running, the Coordinator will communicate with the job dispatcher and the GUI program. For example, the simulation engine process will periodically send the current virtual time of the simulation network to the coordinator. Then the coordinator will relay the information to the GUI program. Besides, the user can also set or get a protocol module's value on line during simulation. (e.g. to query or set a switch's switch table). Message exchanges happening between the simulation engine process and the GUI program are all done via the coordinator. [8][20][21]

In addition, we need to add more functions. It falls into two parts: periodically sending simulation result to GUI (e.g. ptr and node's location) and polling GUI's commands, for example, change node's location and speed etc.

The Log Pack component is responsible for packing all information of simulation results into packets. When the simulation starts, these packets will be

periodically sent to GUI.

```
#define PackSizePtr          20
#define PackNum_SendPtr     20
#define PackSizeLoc         200
#define PackNum_SendLoc     10
```

Due to performance consideration, we can't let one packet encapsulate only one data. Therefore, we make one packet that was sent to GUI contains several data. There are two principles to encapsulate data into packet and send. One is depend on time; engine will periodically send packet. This period can be decided by user. Another is depending on number of data; PackNum\_SendPtr and PackNum\_SendLoc respectively act as watermark to decide whether to send packet or not.



```
struct PtrPacket{
    char type;
    int num;
    LogObject LogObjectArray[PackSizePtr];
};
struct LocPacket{
    char type;
    int num;
    struct NODE_POSITION LocArray[PackSizeLoc];
};
```

In addition, to reduce the overhead of memory copy, we directly put Ptr and Location separately into PtrPacket and LocPacket. Struct of PtrPacket and LocPacket. We previously allocate a large array which collects all data that want to send. MaxPackSizePtr and MaxPackSizeLoc are separately the max limit of capacity that one packet can contain ptr or location information.

In NCTUns 2.0 and the previous version, user can depict node's movement



before simulation, and all node's moving path will be described in the file of \*.sce. Yet new design of Simulation Engine allow node dynamically change its movement. Therefore, it shall be useless for the Simulation Engine if we want to modify node's moving path in real time. At the same time, we need another file of \*.nll to record all node's change. The file of \*.nll, the abbreviation of node location list, periodically keeps the track of all node's location. It is useful for user to trace node's movement and debug after simulation.

Additionally, in order to receive commands from GUI, we have to establish a connection between GUI and Simulation Engine. We shall have to perform periodical polling if GUI sends any communication, because we can not predict when GUI does.

### **5.2.2.2 Communication with Node Control Program**

There are some issues to be discussed about how to communicate with Node Control Program. As we mentioned previously, the Simulation Engine is based on event-trigger. If no event exists, the Simulation Engine will advance its clock quickly.

Because node's information is store in the Simulation Engine, if Node Control Program wants to know current node's location, speed or direction of movement, it has to query Simulation Engine. When all Node Control Program has finished querying or sending information to Simulation Engine, Simulation Engine can advance its virtual clock. To take a simple example, Node Control Program will query node's location at virtual time  $t_1$  in Figure5.2.2.2.1. If Simulation Engine can respond it immediately at virtual time  $t_1$  instead of advancing virtual clock, Node Control Program will get the information of location at time  $t_1$ , which is what it really wants.

At virtual time  $t_1$ , if Simulation Engine thinks all nodes have no longer communicated with it and advance its virtual clock to  $t_2$ , but in fact one of Node

Control Program want to query its node's location at  $t_1$ . While Simulation Engine receives this query at virtual time  $t_2$ , Simulation Engine will respond node's location at  $t_2$  instead of location at  $t_1$ . This results in inaccuracy and incorrect result. The question now arises is how Simulation Engine knows all of Node Control Programs has finished communicating with it.

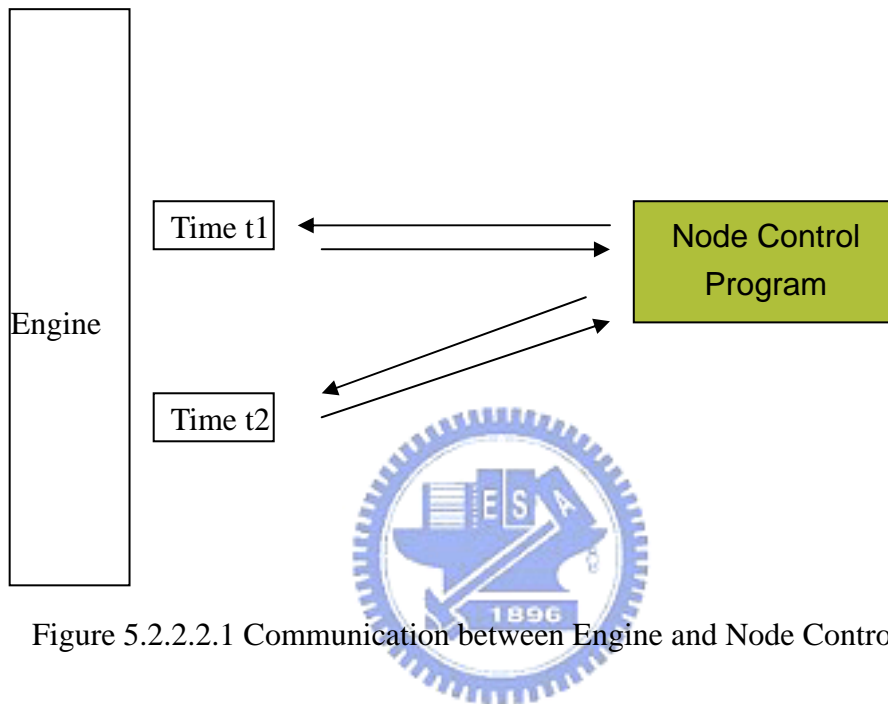


Figure 5.2.2.2.1 Communication between Engine and Node Control Program

To answer this question, let us discuss the process scheduling first. Since we can understand this concept more fully in [21] we shall outline here only briefly.

In Linux, a user-level process also has two priorities: static priority and dynamic priority [22]. The static priority is assigned by the users for real-time processes and never changed by the process scheduler in the kernel. The dynamic priority is used for normal processes and is essentially the sum of the base time quantum (which is therefore also called the base priority of the process) and of the number of ticks of CPU time left to the process before its quantum expires in the current epoch. The static priority of a real-time process is always higher than the dynamic priority of a normal one. The scheduler will run normal processes only when there is no real-time process in the executable state.

The simulation engine sets a traffic generator as a real-time process when forking it. At the same time, the simulation engine is still a normal process. In other words, the traffic generator's priority is always higher than the simulation engine. Therefore, only when all of traffic generators both get blocked in the kernel mode, the simulation engine will be able to get CPU control.

In view of this, while simulation engine get the CPU control, all Node Control Program get blocked in select for socket between simulation engine and itself. As soon as Simulation Engine sends any information to Node Control Program, Node Control Program will get control immediately.

This was equivalent to saying that Simulation Engine's notification packet which sends from simulation engine to Node Control Program let Node Control Program get control of CPU. Therefore, we can let notification packet to query how many commands which Node Control Program will send to Simulation Engine. The query result will respond to Simulation Engine in the form of ACK packet. As soon as Simulation Engine receives ACK packet which contains nonzero ACK number, it will wait Node Control Program instead of advancing simulation clock. Figure 5.2.2.2.2 shows this scenario.

However, there is a problem as Figure 5.2.2.2.3 shows. In the Figure 5.2.2.2.4, it describes a solution for above situation. When simulation engine receives an ACK packet with zero number, it will send a query packet to query Node Control Program how many commands it will send. At this time, if simulation engine receives corresponding ACK with zero number, we can make sure that Node Control Program will not send any more commands to simulation engine at this time. There is a further point which needs to be clarified. Readers may doubt that if Node Control Program generates another command which will send to simulation engine after it responds ACK packet which corresponding to query packet just like Figure 5.2.2.3.3.

In fact, query packet is different from notification packet. The latter is used for simulation engine to notify Node Control Program, it may influence Node Control Program. For example, the Simulation Engine sends a notification packet which tells node that has collided into obstacle, while Node Control Program receives this packet; it may change its direction, speed etc. Therefore, it causes Node Control Program to generate another packet. However, query packet is just like a god packet which doesn't carry any information except query if Node Control Program will send any command in the future. In view of this, query packet will not influence Node Control Program's behavior and decision, so Node Control Program will not generate other commands after receiving this kind of packet.

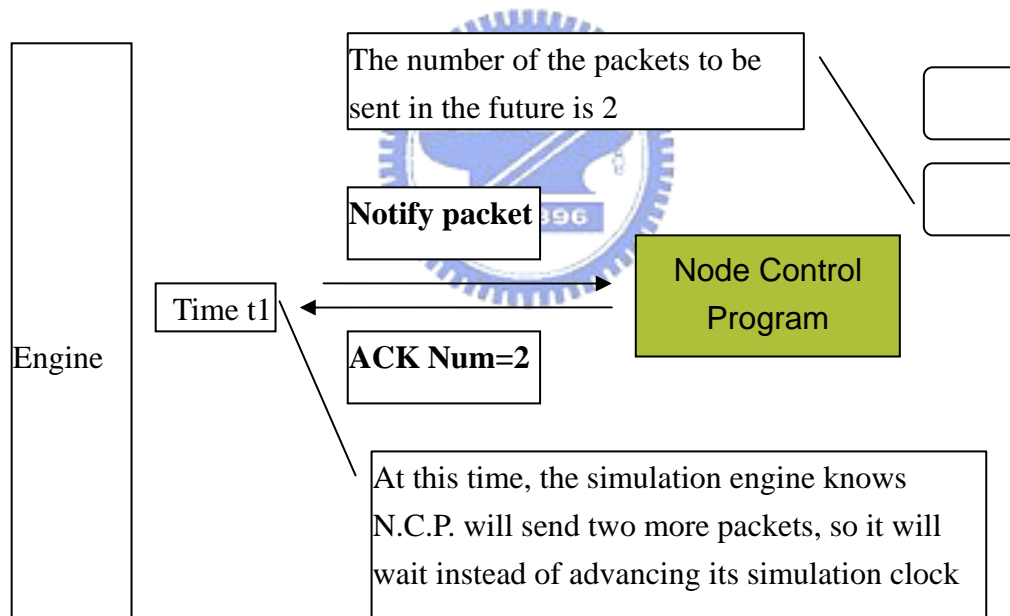


Figure 5.2.2.2.2 Communication between Engine and Node Control Program

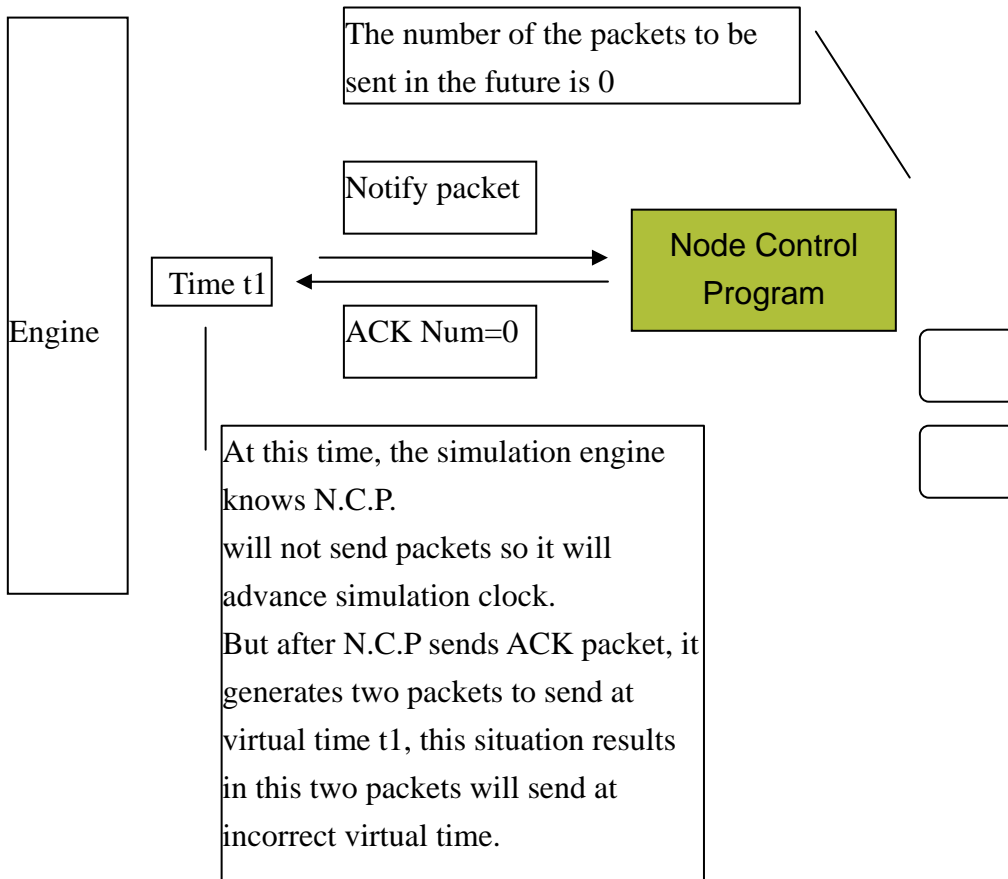


Figure 5.2.2.2.3 Communication between Engine and Node Control Program

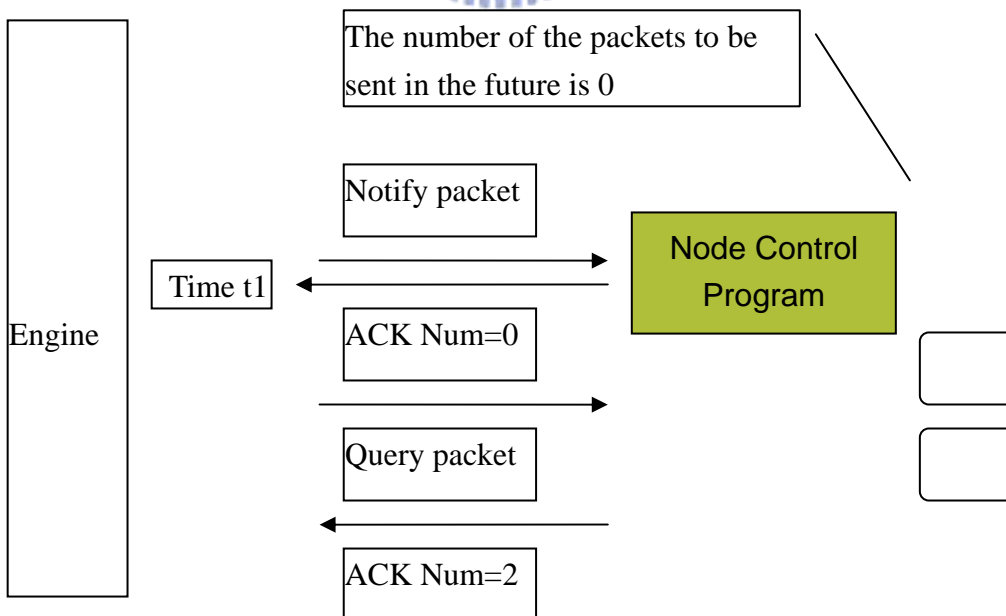


Figure 5.2.2.2.4 Communication between Engine and Node Control Program

### 5.2.3 Obstacle Construction

Obstacle plays an import role in our new features. We modify original design of wall. Now we can support three kinds of obstacles. First, it provides the function to block wireless signal. Secondly, it can block node's movement. Thirdly, it can obstruct node's field of vision.

```
struct obstacle {  
    double          x1,y1;  
    double          x2,y2;  
    double          x3,y3;  
    double          x4,y4;  
    char           type;  
    struct obstacle *next;  
};
```

The above data structure is used to store information of obstacles. In this structure, 'type' is used to discriminate the type of obstacle. All obstacles are linked in a linked-list which headed by pointer *Obs\_head*. We can iterate all of obstacles by this pointer. Besides, *Num\_obstacle* is a number used to count how many obstacles exists.

## 5.3 Intelligence Engine

In this section, we will shift the emphasis away from basic Simulation Engine to Intelligence Simulation Engine. We will use the term "Intelligence Engine" to refer to the part of simulation engine which involved with some techniques of artificial intelligence.

### 5.3.1 Map Sensing

The first point to notice is our coordinate system is different from standard. Our

origin is on the left-top. The abscissa will increase toward right; however, its ordinate will decrease toward north. Figure 5.3.1.1 and Figure 5.3.1.2 show the difference between them. The reason why we adopt this coordinate system is that GUI's location also makes use of this kind of coordinate system. However, some problems increase when we turn from standard coordinate system to this one. We shall have more to say about these problems later on.

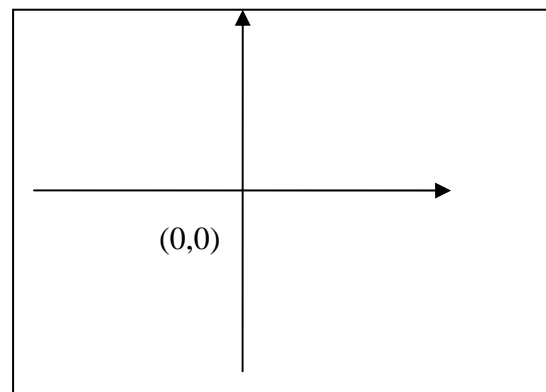


Figure 5.3.1.1 Standard coordinate system

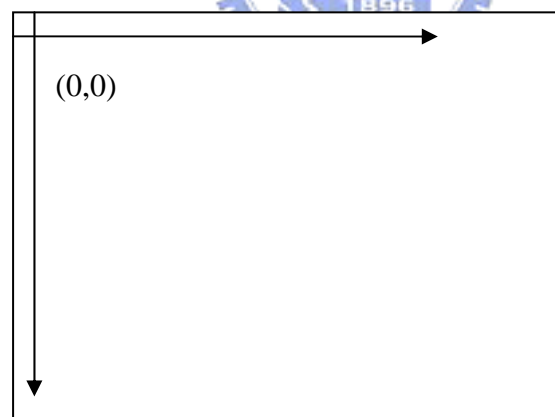


Figure 5.3.1.2 Our coordinate system

## 5.3.2 Obstacle Detection and Collision Avoidance

The Simulation Engine provides many APIs for obstacle detection and collision avoidance. Any prediction of obstacle collision can query through these APIs. Here, we only introduce briefly how Simulation Engine works by making use of these API

to reach our purpose. Let us consider following case.

At time  $t_1$ , the node “A” moves toward east. We first call the API FindShorestCollisionObstacle. This API finds out all candidates which node A will collide in the future. In this case, node “A” will collide into obstacle “a” at “A’ ” and collide into obstacle “b” at “B’ ”. However, it selects the shortest one. Then we will calculate the distance between “A” and “A’ ”. We assume the distance is  $d$ . Therefore, we can predict when node “A” will collide into “a” via dividing “ $d$ ” by node’s speed. If the calculate result is “ $T$ ”, we insert an event with timestamp “ $t_1+T$ ”. It means Node “A” may collide into “a” at time “ $t_1+T$ ”. Figure 5.3.2.1 shows above scenario.

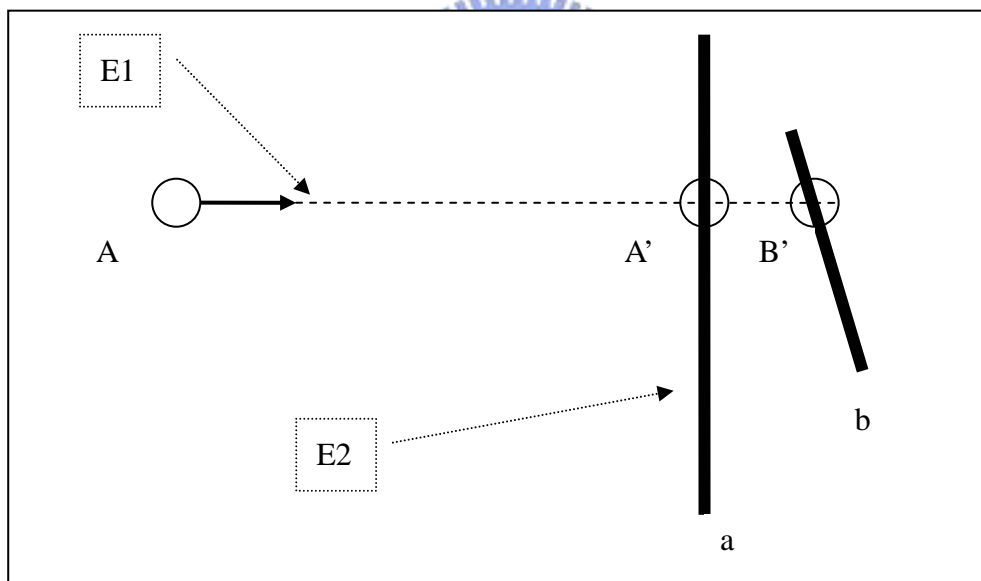


Figure 5.3.2.1 Obstacle detection of simulation engine

For the present, we shall confine our attention to another problem. We also use above example in the Figure 5.3.2.1 to illustrate this problem in the following.

In API FindShorestCollisionObstacle, it first calculates the linear equation “E1” of node A’s moving path and linear equation “E2” of obstacle “a”. Then, point “A’ ” is gotten by calculating the solution of linear equation “E1” and “E2”. However, there



are some inaccuracies in mathematical calculating. These inaccuracies cause some problems. For example, we assume that the location of point “A” is (100, 10) and point “A’ ” is (100,50) . In addition, node’s speed is 3m/sec. Therefore, the distance between “A” and “ A’ ” is 40m. The Simulation Engine will insert predict event with timestamp “13.3333+ t1”. However, after 13.333 sec, the location of node “A” is (100,49.9999) instead of (100,50). This inaccuracy will cause node ”A” not collide into obstacle after 13.333 sec. Similar examples are numerous.

With these points in mind we can look at how to overcome this problem. In order to take inaccuracy into consideration, we translate the whole map into rectangular grid. Figure5.3.2.2 helps account for this concept.

If two points in the same grid, we treat them as the same one. For example, we assume that each grid’s width is 1, then point (100, 50) and point (100, 49.9) will belong to the same grid. This can ignore some inaccuracy while calculating.

Figure 5.3.2.4 shows how to translate obstacle into grid. The grids with dots and with oblique lines belong to the grid of obstacle.

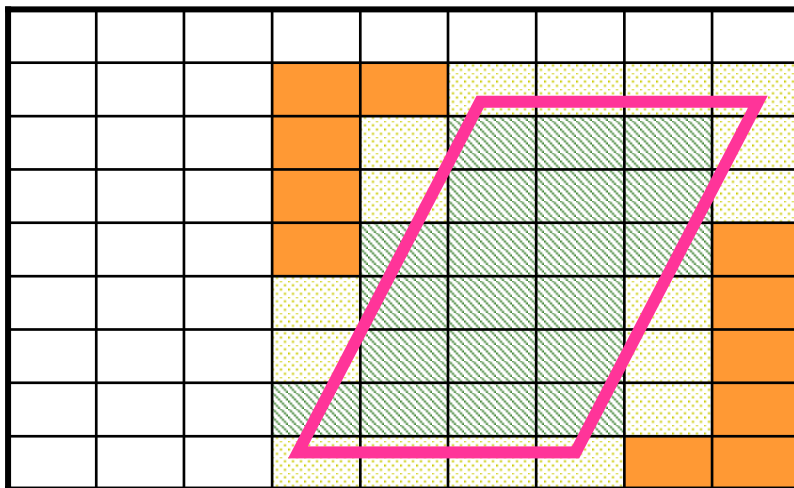


Figure 5.3.2.4 Translate an obstacle into grid of map

### 5.3.3 Path Finding

The Simulation Engine adopts A\* algorithm for our path finding algorithm. Readers can refer to references to get more detail illustration. The important point to note is that the implementation of A\* can be a nightmare to realize. In [19], it makes an error assumption of path's cost.

```
void CPathFinder::UpdateParents(_asNode *node) {
    int g = node->g, c = node->numchildren;

    _asNode *kid = NULL;
    for (int i=0;i<c;i++) {
        kid = node->children[i];
        if (g+1 < kid->g) {
            kid->g = g+1;
            kid->f = kid->g + kid->h;
            kid->parent = node;

            Push(kid);
        }
    }
}
```

It will cause error while going through priority grid, because it only adds 1 instead of priority cost

### 5.3.4 Event-Triggering System

A trigger system serves two main purposes in a game: it keeps track of events in the game world that agents can respond to, and it minimizes the amount of processing agents need to response events. [23]

The TriggerRecordStruct defines an instance of a trigger. The EnumTriggerType are enumerated as bit-flags. It was defines what a node interests. Each node has its responding agent to respond central trigger event. The agent can specify what trigger types it pays attention to.

The trigger system itself is used to stores records for existing triggers, and is

responsible for registering, removing, and updating triggers. Node can add what it interest by calling RegisterTrigger() API.

```
struct TriggerRecordStruct
{
    EnumTriggerType    eTriggerType;
    unsigned long      nTriggerID;
    unsigned long      idsource;
    unsigned long      idgoal;
    Vector              vPos;
    double              fRadius;
    u_int64_t          nExecuteInterval;
    u_int64_t          nTimeStamp;
    u_int64_t          nExpirationTime;
    double              nExecuteCount;
    bool                bDynamicSourcePos;
    EnumGroupType      nGroupType;
    static unsigned long s_nNextTriggerID;
};
```

```
enum EnumTriggerType
{
    kTrig_None          = 0,
    kTrig_CollisonObstacle = (1 << 0),
    kTrig_CollisonPred    = (1 << 1),
    kTrig_ReachLoc        = (1 << 2),
    kTrig_VisualEnemy     = (1 << 3),
    kTrig_EncounterEnemy  = (1 << 4),
    kTrig_EnemyVisualOther = (1 << 5),
};
```

## 5.4 Node Control Program

Let's consider following scenario. When Simulation Engine notifies a node that it encounters enemy, the node will want to know where enemy is and where its

location (because all node's information is stored in Simulation Engine) first before it sends alarm message to its members. This drives us to the question about how we implement it. After receiving the notification packet from Simulation Engine, it has to send a command packet to query the location of the enemy. Then Node Control Program has to wait Simulation Engine's reply packet before doing something others. The core of the question is that it is hard to implement for Node Control Program. It is desire to discuss some techniques in the following topic before solving this problem.

### 5.4.1 Layered Design

Our goal is to make the AI components move, react, and think like human. But computers are not yet smart, we must write code to answer everything from high-level issues such as "How do I flank a group of soldiers behind cover?" to very low-level issues such as "Where do I put my foot in order to walk?". Implementing the solutions to these issues as a monolithic state machine that is hard to extend and maintain. [24]

A subsumption architecture [25] cleanly decomposes the implementation into concurrently executing layers of finite state machines (FSMs). [26][27] Lower layers take care of immediate goals, and upper levels take care of long-term goals. The architecture solves some major problems with AI: interrupting causing a character to forget what it was doing, characters getting stuck on obstacle goals.

We adopt this concept and apply this into our Node Control Program. We divide the whole AI of Control Node Program into three layers. The Layers are shows in Figure5.4.1.1.

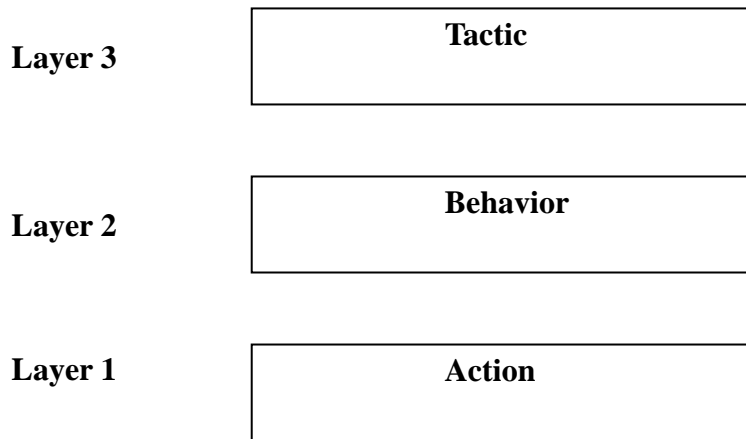


Figure5.4.1.1 Layer of Node Control Program

**Action Layer:** The lowest layer handles the communication with Simulation Engine. It is responsible for all atomic action. Each atomic action is a communication command with simulation such as querying node's location, changing node's speed and registering an interest event etc.

**Behavior Layer:** This layer is constructed out of actions. Examples of behaviors are "Scout", "Chase Enemy", and "Report message to Leader" etc. The behavior is just a sequence of actions.

In case of the "Chase Enemy" behavior, the behavior constructs a series of actions in order to fight the intended target. This might include getting node's location, getting target's location, notifying its members and moving toward targets.

**Tactic Layer:** Node may have some high-level tactics and some simple planning. A tactic may contain a lot of states. In this layer, Node Control Program decides how to do next and transform its state.

Now, let us discuss how to make it easy to implement a behavior which contains a sequence of actions. The answer to this question is that we have to use a command queue of function pointer to reach our goal. As Figure5.4.1.2 shows the command queue, this queue mainly has two parts separately, first is the function point of action which wants to be executed, and second is the flag "have\_to\_wait". The latter is used

to identify this command has to wait all of queue previous itself to finish. If corresponding respond packet from Simulation Engine has received by Node Control Program, it will be removed from the queue.

For example, a behavior contains five actions A, B, C, D, and E. Each action will cause Simulation Engine to reply. If Action C has to wait until Action A and B finishing and replying from Simulation Engine .In addition, Action E also has to wait until Action C finishing. Action A, B, D can execute independently. Then it will separately set the flag “have\_to\_wait” of command queue to be false, false, true, false, and true.

In the beginning, Action A, and B will execute. But Action C has to wait until the respond packet of Action A and B received by Node Control Program. After Node Control Program receives respond packet of A and B from Simulation Engine. This means Node Control Program has acquired related information which required by Action C. Therefore, Action C and D can execute. In the same reason, Action E will execute until Action C’s respond packet is received. Above concepts offer the key to solve our problem which we had mentioned in the section 5.4.

**command queue**

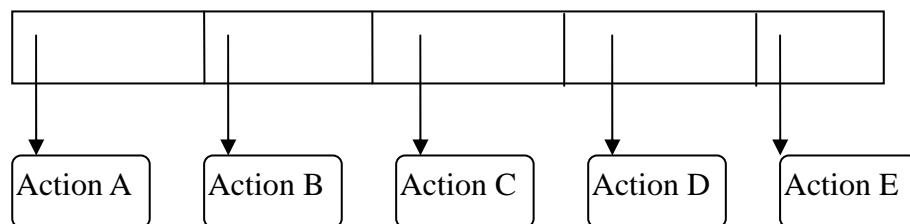


Figure5.4.1.2 command queue of Node Control Program

## 6. A Simulation Example

In this section, we will mainly measure how much time the modified simulation engine takes to run a tactical ad-hoc network case.

### 6.1 Simulation Setup

Figure 6.1.1 depicts the scenario of a simulation case. As the figure shows, there are 100 nodes; there is only one case of our simulation scenario. In fact, we will run 10 kinds of case, from 10 nodes to 100 nodes. The only difference between them is the number of nodes. Each node will execute Node Control Program to communicate with the Simulation Engine. Based on AODV routing protocol, Each node adopts IEEE 802.11 module.

In above simulation cases, there is always an enemy node and all other nodes will chase it. All other nodes will organize a squad. This group will periodically broadcast to each other to make sure which one it can communicate with. This squad will select a node as a leader. If other member encounters an enemy node (the distance is smaller than 10m), it will notify the leader to organize a chase action.

To make sure all of simulation cases will be run the same simulation time (100 sec simulation time). We will let an enemy never be arrested by other nodes. (Make this condition fail to achieve)

Figure 6.1.2 is another kind of simulation case. All settings are the same as above except they are surrounded by some obstacles. Because obstacle make simulation engine to do extra calculation, it may result in more time consumptions.

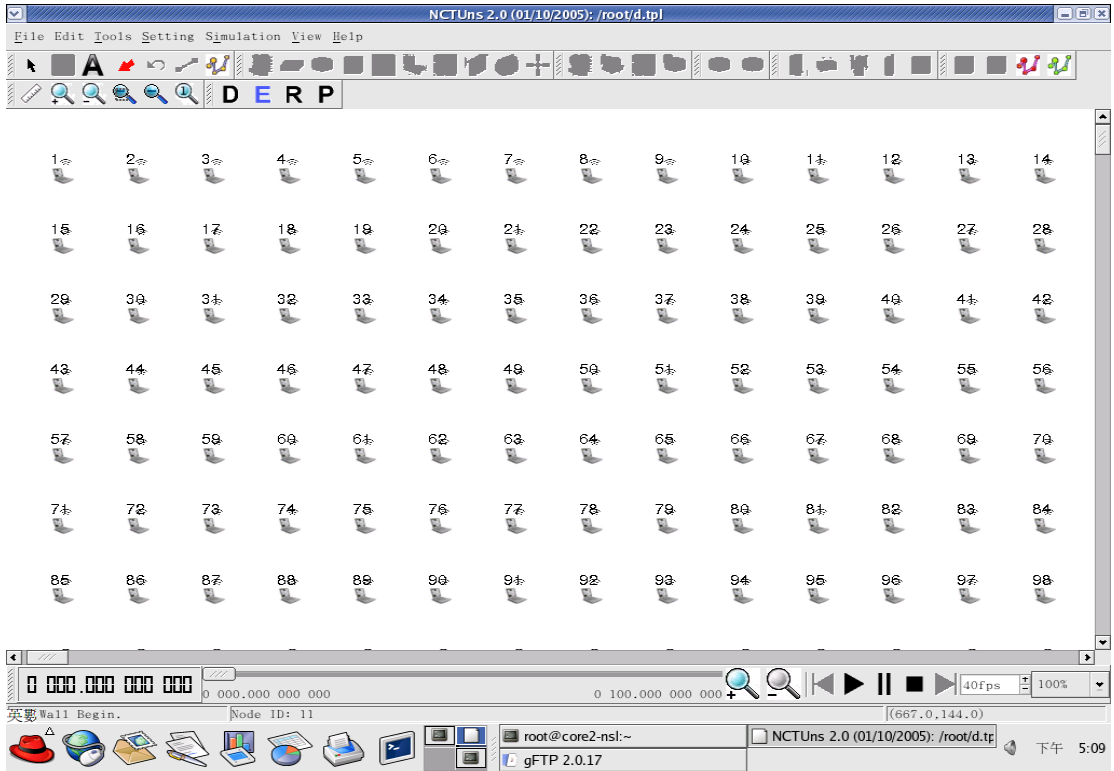


Figure6.1.1 The scenario of simulation case

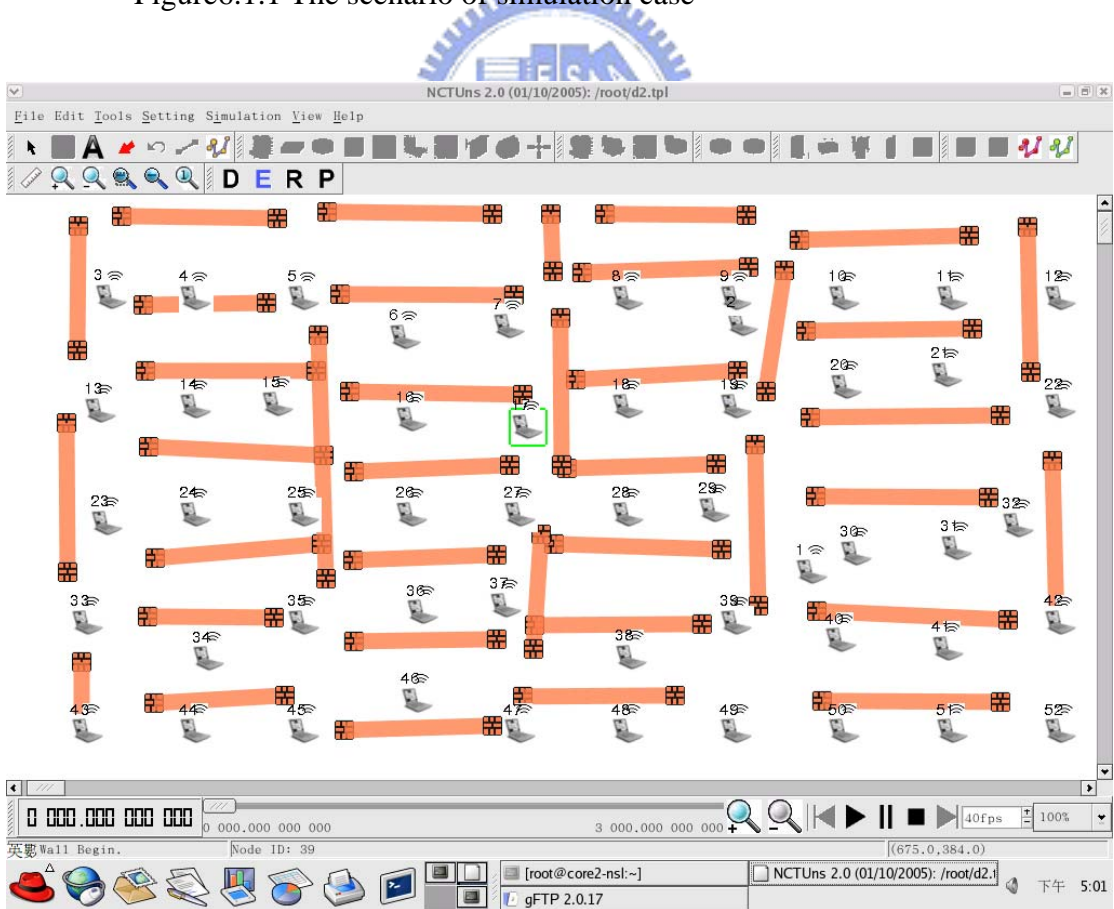


Figure6.1.2 The scenario of simulation case with obstacles



## 6.2 Result

Figure 6.1.1 ~ Figure 6.1.3 are the simulation results of cases which have no obstacle. Figure 6.1.4 ~ Figure 6.1.6 are the simulation results of cases with obstacle. We evaluate the simulation performance from three aspects: CPU utilization, required memory space, and required time of simulation.

The reason of low CPU utilization is that most of time of simulation is spent on transmitting packets and recording dynamic log. As we expect, simulation cases with obstacle require more memory, CPU utilization, and time. Time is the obvious difference between them, because obstacles will make simulation engine to avoid obstacle collision. Besides, it will generate more predict event packets as we described in previous chapter. In addition, we can observe that the cure of required time rises rapidly when the numbers of nodes increases. The reason is that more nodes will result in more interaction between each other. Therefore, it causes the Simulation Engine to generate more events.

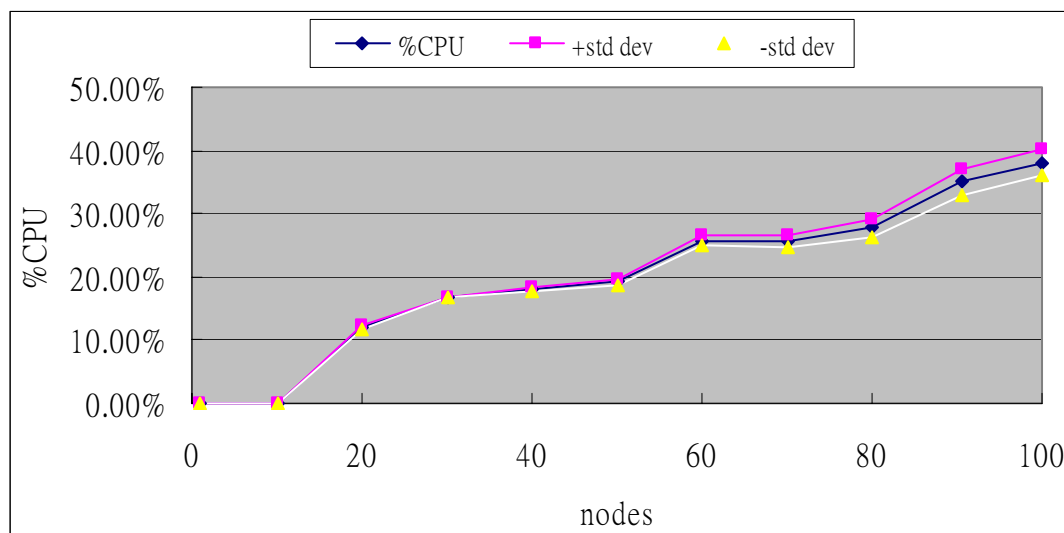


Figure 6.2.1 The CPU utilization of simulation case without obstacle

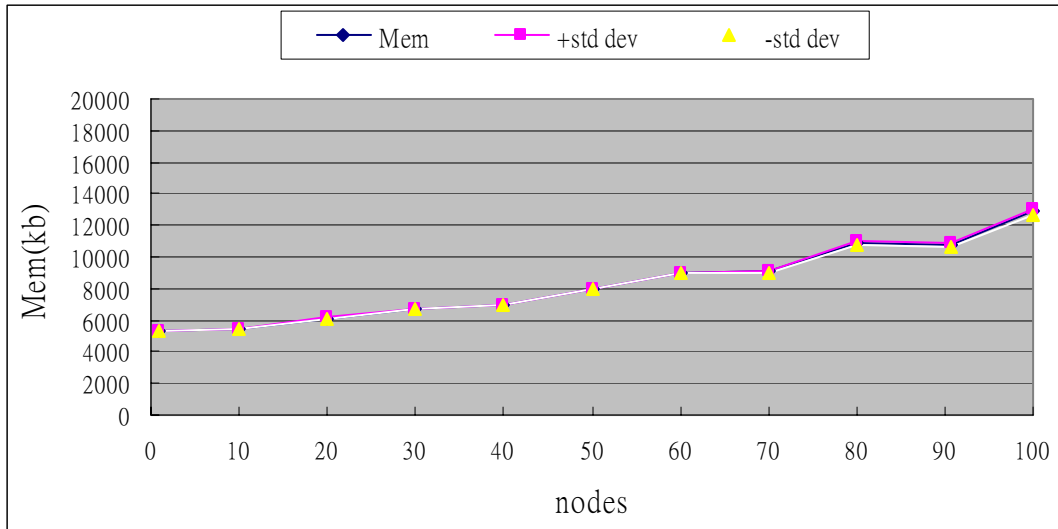


Figure 6.2.2 The required memory space of simulation case without obstacle

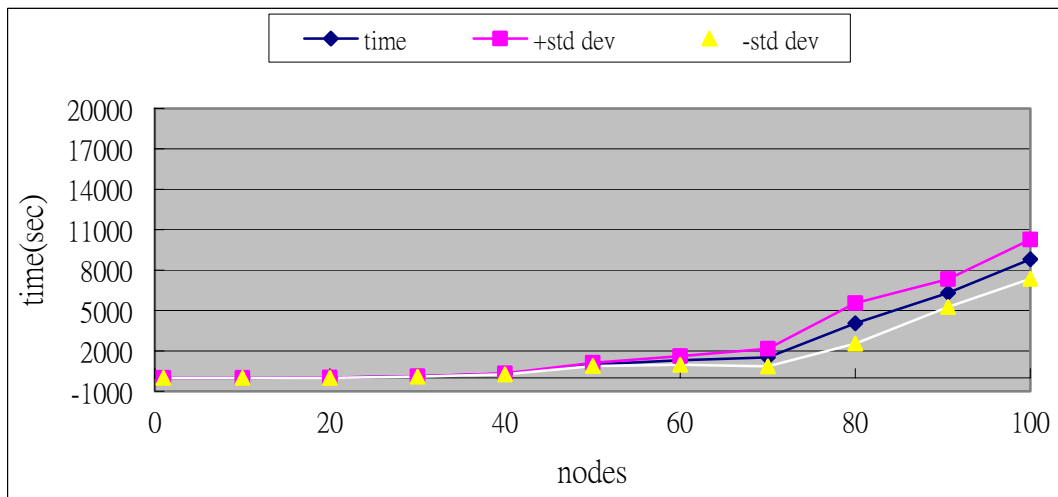


Figure 6.2.3 The required time of simulation case without obstacle

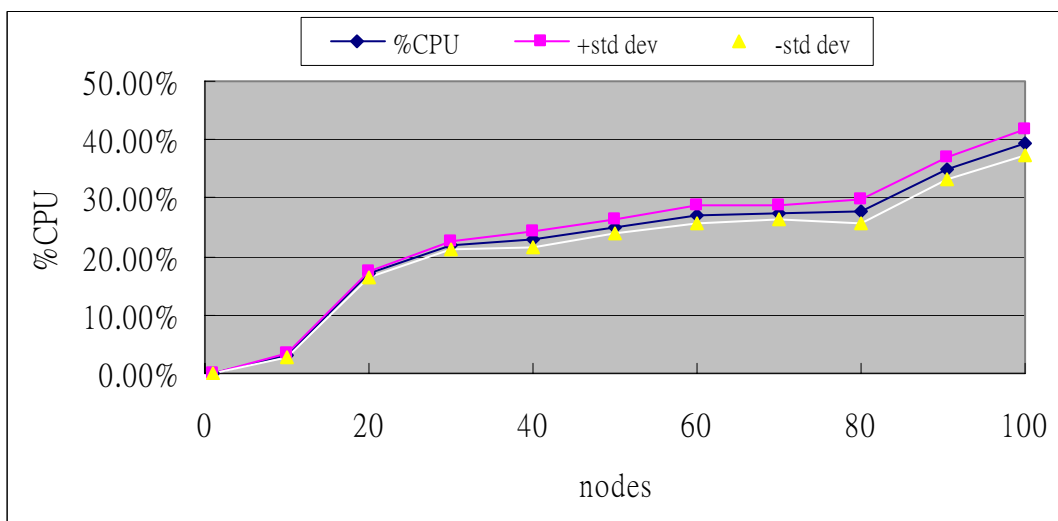


Figure 6.2.4 The CPU utilization of simulation case with obstacle

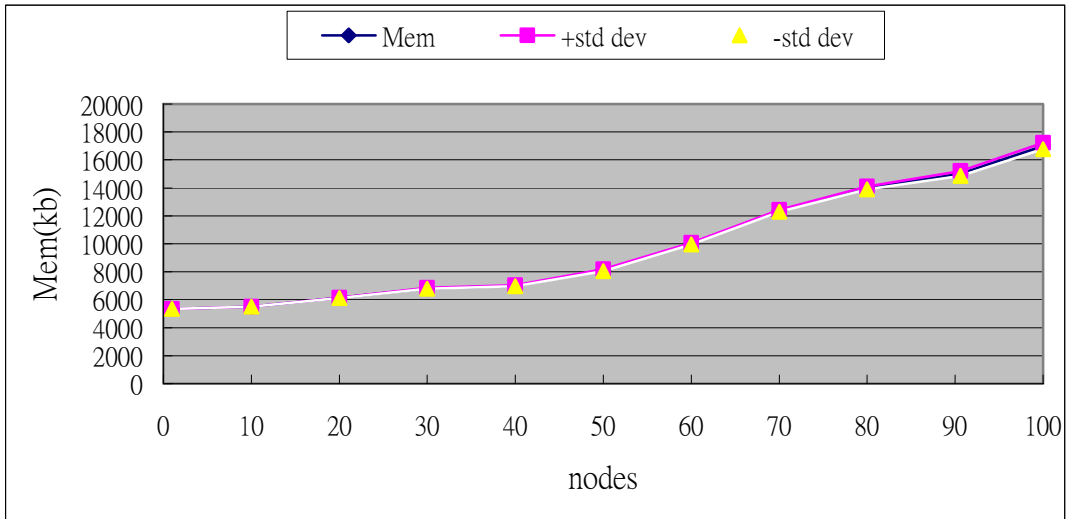


Figure 6.2.5 The required memory space of simulation case with obstacle

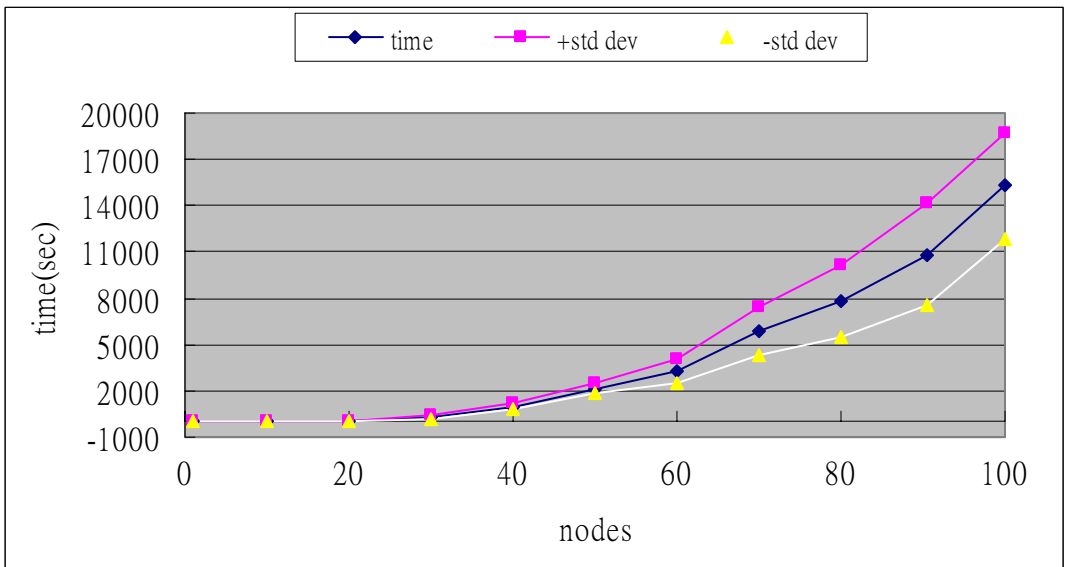


Figure 6.2.6 The required time of simulation case with obstacle

## 7. Future Work

In the current version, Simulation Engine only supports 2D terrain, though node's location is 3-dimension. However, if we want to support 3-dimension graphical user interface for visualizations of communication networks, it needs to be modified further. Besides, our tactic for Node Control Program only supports a group of soldiers to chase one enemy. In the future, its tactic should be improved to support two hostile groups to fight. Due to the consideration of performance, the simulation time for a large simulation case is long. This aspect may be solved by parallel simulation. Future work can address the challenges of scale (from a small network to a large one) in a heterogeneous battle space involving joint and coalition forces.



## 8. Conclusion

During the past several years, more and more organization and researcher focus on military network. Therefore, if NCTUns network simulator can be equipped with the function to support tactical Ad Hoc network, it is useful for researchers to develop and design new technologies to support next-generation mobile wireless high-capacity tactical communications and to meet future army communications requirements. In addition, it can highly extend the ability of NCTUns network simulator.

In this paper, we present the internal design, modification and implementation of the simulation engine to support tactical Ad Hoc network on NCTUns network simulator. In order to reach this goal, a number of features, for instance, obstacle prediction and collision avoidance, path finding, event-triggering, and dynamical control and real-time animation, have to be added. Therefore, it involves a lot of techniques, such as graphics theory, mathematics, and artificial intelligence, etc. There is no need to go into details about every subject of these techniques. We only briefly illustrate some of their important concepts. Readers can refer to references for more detail discussion.

## Reference

- [1] S.Y. Wang and H.T. Kung, “A Simple Methodology for Constructing Extensible and High-Fidelity TCP/IP Network Simulators”, IEEE INFOCOM’99, March 21-25, 1999, New York, USA.
- [2] S.Y. Wang and H.T. Kung, “A New Methodology for Easily Constructing Extensible and High-Fidelity TCP/IP Network Simulators”, accepted and to appear in “Computer Networks” Journal
- [3] OPNET Inc., <http://www.opnet.com>
- [4] S. McCanne, S. Floyd, ns-LBNL Network Simulator, <http://www.isi.edu/nsnam/ns/>
- [5] Harvard TCP/IP network simulator 1.0, available at <http://www.eecs.harvard.edu/networking/simulator.htm>
- [6] S.Y. Wang, C.L. Chou, C.C. Hwang, A.J. Su, C.C. Lin, K.C. Liao, H.Y. Chen, and M.C. Yu, “Applying Discrete Event Simulation to the NCTUns 1.0 Network Simulator”.
- [7] William van der Sterren, “Squad Tactics: Team AI and Emergent Maneuvers” , AI Game Programming Wisdom, Charles River Media, 2002.
- [8] S.Y. Wang, “The GUI User Manual for the NCTUns 2.0 Network Simulator and Emulator”, available at <http://nsl10.csie.nctu.edu.tw/>, January 10, 2005.
- [9] QualNet Inc., <http://www.qualnet.com/>
- [10] High Capacity Tactical Network (HCTCN) , available at <http://www.crc.ca/en/html/manetsensor/home/projects/hctcn>
- [11] C.K.Toh, E.C.Lee, N.Ramos, “Next-Generation Tactical Ad Hoc Mobile Wireless Networks”, in Northop Grumman Technology Review Journal, Volume 12-1, Spring/Summer 2004.

- [12] Foo Yee Loo, "Ad Hoc Network: Prospects and Challenges," Graduate School Research Paper (Rinkou), Department of Information and Communication Engineering, University of Tokyo, January 2004
- [13] Elizabeth M. Royer and Charles E. Perkins. "An Implementation Study of the AODV Routing Protocol." *Proceedings of the IEEE Wireless Communications and Networking Conference*, Chicago, IL, September 2000.
- [14] Charles E. Perkins, Elizabeth M. Belding-Royer, and Ian Chakeres. "Ad Hoc On Demand Distance Vector (AODV) Routing." *IETF Internet draft*, draft-perkins-manet-aodvbis-00.txt, Oct 2003 (Work in Progress).
- [15] Rabin, Steve, "A\* Speed Optimizations", Game Programming Gems, Charles River Media, 2000
- [16] Patel, Amit J., "Amit's Game Programming Information", available online at <http://www-cs-students.stanford.edu/~amitp/gameprog.html>, 2001.
- [17] Matthews, James, "Basic A\* Pathfinding Made Simple", AI Game Programming Wisdom, Charles Rivers Media, 2002
- [18] Patel, Amit J., "Amit's Thoughts on Pathfinding", available online at <http://theory.stanford.edu/~amitp/GameProgramming/>, November 27, 1999.
- [19] Matthews, James, "A\* Pathing Finding: CPathFinder", available online at <http://www.generation5.org/content/2000/cpathfinder.asp>
- [20] S.Y. Wang, C.L. Chou, C.H. Huang, C.C. Hwang, Z.M. Yang, C.C. Chiou, and C.C. Lin , "The Design and Implementation of the NCTUns 1.0 Network Simulator", *Computer Networks*, Vol. 42, Issue 2, June 2003, pp. 175-197.
- [21] K.C. Liao, "Porting the NCTUns Network Simulator to Linux and Supporting Emulation," Master thesis, National Chiao Tung University, Hsinchu, Taiwan, 2004.
- [22] Daniel P. Bovet and Marco Cesati, "Understanding the Linux Kernel, 2nd Edition", O'Reilly, 2002.
- [23] Orkin Jeff, "A General-Purpose Trigger System", AI Game Programming Wisdom, Charles Rivers Media, 2002

[24] Yiskis Eric, “A Subsumption Architecture for Character-Based Games”, AI Game Programming Wisdom 2, Charles Rivers Media, 2004

[25] Arkin, Ronald C., Behavior Based Robotics, MIT Press, 1998, pp. 130-140.

[26] Fu, Dan, and Houlette, Ryan, “The Ultimate Guide to FSMs in Games”, AI Game Programming Wisdom 2, Charles Rivers Media, 2004

[27] LaMothe, Andre, “Finite State Machines”, Tricks of the Windows Game Programming Gurus Second Edition, Sams Publishing (Macmillian), 2002, pp.737-742

