



Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization:
Computer Science

Autumn semester, 2011

Open access

Writer: Bo Liu

.....
(Writer's signature)

Faculty supervisor: Reggie Davidrajuh

Title of thesis: Simulation of Network Intrusion Detection System with GPenSim

Credits (ECTS): 30

Key words:

Colored Petri Net, GPenSIM, Network Intrusion Detection System, Particle Filter

Pages: ... 42.....

+ enclosure: 24.....

Stavanger, 19, December, 2011

Abstract

In recent years, network has penetrated into every aspect of our life with its rapid growth and popularization. More and more serious network security problems have occurred together with this process, especially network intrusion problem. It has seriously affected the normal use of network, so research of network intrusion detection has become one of the hottest research areas. This thesis simulated a network intrusion detection system based on particle filter to solve the network intrusion problems. It can filter the intrusion behaviors by using particle filter technique into network intrusion detection. This thesis simulates against two specific intrusion behaviors (intrusion and Trojan). The tool used for modeling is colored Petri net which is built up on basic Petri net. Colored Petri net can model complex object by adding specific meaning to tokens. It also supports structured modeling, which can divide complex object into several parts.

This thesis emphasized on the process of simulating with colored Petri nets. It simulates two network intrusion systems against intrusion and Trojan behaviors. Both of the two models for network intrusion detection system are divided into three modules. The models, functions and connecting methods of these modules are presented in the thesis. The meaning of the places and color spaces of tokens in the model are presented in detail. The meanings of the transitions and requirements for them to fire are also introduced. This thesis programs with GPenSIM (General Purpose Petri Net Simulator) toolbox. The problems in the simulation and the process of solving them are described. The result of the simulation is also presented and analyzed.

Key Words: Colored Petri Net, GPenSIM, Network Intrusion Detection System, Particle Filter

Contents

1. Introduction	1
2. Methods	3
2.1 <i>Petri net</i>	3
2.2 <i>Gpsim</i>	4
2.3 <i>Particle filter</i>	4
2.4 <i>Intrusion detection system</i>	5
3. Related Work	6
4. Model and simulation	7
4.1 <i>Overall design</i>	7
4.2 <i>Intrusion detection</i>	8
4.2.1 Model Design.....	8
4.2.2 Petri net design.....	8
4.2.3 Simulation result.....	21
4.3 <i>Trojan detection</i>	26
4.3.1 Model design	26
4.3.2 Petri net design.....	27
4.3.3 simulation result.....	36
5. conclusion.....	40
6. Future work	42
References	43
Source code	45

1. Introduction

With the rapid development of the Internet, its security problems are also coming out. Network intrusion is a big problem in network security field. Network intrusion is the behavior that illegally operates others' computer system to steal or juggle their important private messages without authorization. Network intrusion is very harmful, can cause damage to the system, lost of the secret data and illegal control (including Trojan control). The target of network intrusion detection is to detect the abnormal behaviors or abnormal packets to maintain the interests of the network users and guarantee the safety of the network. The research of network intrusion detection has both important theoretical significance and practical value.

There are various kinds of methods for modeling of network intrusion detection system. Petri net is a popular method among them. Petri net is the mathematical representatives of discrete concurrent system. It is created in 1960's, and has been expanded to many different kinds from the basic Petri nets after many years' development, including timed Petri nets, colored Petri nets, layered Petri nets, and so on. Nowadays, the applications of Petri nets have penetrated into many computer fields including network protocol, network security, concurrent systems and artificial intelligence ^[1].

Colored Petri net is used frequently in the modeling of complex models. Colored Petri net is an advanced Petri net derived from the basic Petri net. Its expressing competence is expanded by adding specific meaning to each token on the basis of normal Petri net. Color is similar to the data structure in the advanced programming language and can make a good description of the resources. During the process of modeling, colored Petri net can build hierarchical model. That is, by decomposing layer by layer, simple models are designed first, then they are combined together to form complex models. With these advantages, color Petri net is applied more and more in the modeling of complex models in recent years ^{[1][2]}. In this thesis, some practical problems in network intrusion are analyzed. Two simulation cases including intrusion detection and Trojan detection are analyzed and studied. This thesis aims at the solution of network security problem by simulating a general network intrusion detection system. This goal is achieved by filtering the abnormal packets in the network in the simulation process.

It is a great challenge to judge whether the packets and operations are abnormal in an IDS. Because the scale of network is so large that it is almost used in every corner of the world, it is also difficult to decide how to deploy the network intrusion detection system reasonably. What's more, the behaviors of network intrusion are changeable and new intrusion

behaviors appear continuously, which increases the error rate of network intrusion detection system.

In this thesis, another important problem is the process of simulation. We need to analyze the details and results of the simulation of IDS to see what happens in the network intrusion process so that we can analyze these problems thoroughly and solve them. In this thesis, the simulation of IDS is realized with colored Petri net and GPenSIM tools.

There are many problems and difficulties in the process of simulating IDS. First is the simulation of the network environment. Since the real network is changeable every minute, it is difficult to simulate every aspect. We simplified it when simulating the network, but still try to make it close to the real one in order to get ideal results. The second problem is that it is difficult to simulate the abnormal packets. We realized it by setting a color in the color space of the places for packets. We also encountered some problems concerning the bugs of GPenSIM. We solved it by some special techniques.

In the paper "Particle Filter for Depth Evaluation of Networking Intrusion Detection Using Coloured Petri Nets"^[3], Chien-Chuan Lin and Ming-Shi Wang proposed a networking intrusion detection system using particle filter concept, and simulated it with Colored Petri Nets tools (CPN Tools). This thesis used the architecture of the model of that paper. Both of two models divide the system into three parts: sender module, detection module and receiver module. And the algorithm to decide whether a packet is normal in this thesis is the same as their paper, which is based on the absolute value of the difference between the port number sent from the sender and that of the particle filter. However, the process of building Petri nets in this thesis is different from their paper, because the development platform they used is CPN tools while I used GPenSIM tools in this thesis. The Petri nets used in this thesis is designed by myself. In this way, the programming process and simulation process in this thesis is also different from that in their paper. The simulation result is also different.

2. Methods

2.1 Petri net

The word Petri spreads widely since Carl Adam Petri from Federal Germany first used reticular formation to simulate communication systems in his doctoral dissertation 'Communication with Automata' in 1962. Today, the word Petri net not only means this kind of model, but also means the theory developed on the basis of it. A great progress has been made in the field of application and theory of Petri net. A Petri net is composed of several places, transitions and arcs which point from a place to a transition or from a transition to a place. Places and transitions can be connected by many arcs, but there are not arcs between places or between transitions. A place can contain any number of tokens. When a transition fires, the tokens in its input places will be consumed and tokens will be produced in its output places. A transition will fire only when there are enough tokens in all its input places. In a basic Petri net, all the tokens are the same, while in the colored Petri net, individuals of the same kind are colored with the same color and different individuals are distinguished by different colors. The advanced Petri net system includes Colored Petri Net, Cyber Net System, Timed Petri Net, Stochastic Petri Net, etc ^[4].

The formal definition of a Petri net ^[5] :

"A Petri net is a four-tuple (P, T, A, x_0) , in which:

P is the set of places, $P = [p_1, p_2, \dots, p_n]$,

T is the set of transitions, $T = [t_1, t_2, \dots, t_n]$,

A is the set of arcs (from places to transitions and from transitions to places),

$A = (P \times T) \cup (T \times P)$,

x is the row vector of markings (tokens) on the set of places,

$x = [x(p_1), x(p_2), \dots, x(p_n)] \in \mathbb{N}^n$

x_0 is the initial marking."

2.2 Gpensim

GPenSIM tools are invented by Professor Reggeie Davidrajuh at University of Stavanger. It is a modeling and simulation tool based on Matlab. It can help realize the modeling and simulation of Petri net easily and combine the other toolboxes of Matlab with Petri net models, which makes the modeling and simulation of Petri net much more convenient ^[5].

GpenSIM is based on Matlab which has very powerful functions. Matlab has powerful toolboxes such as Stochastic toolbox which can generate random data obeying arbitrary distribution. It also has some graph toolboxes which can draw imaginal pictures and show the results of simulation tangibly. Matlab also has its own powerful programming language, which makes the simulation of Petri net using GPenSIM very easy. We only need to program some transition functions, network definition functions and initial functions when simulating Petri net with GPenSIM ^[5].

The powerful function of GPenSIM can help us simulate complex network models. In this thesis the complex Petri net model of IDS is designed with the modular function of GPenSIM. It also has its own colored Petri net functions which can help us model colored Petri net easily. What's more, GPenSIM can combine with Matlab toolboxes perfectly which can help us utilizing the powerful functions of Matlab for our simulation. Compared with other simulation tools, GPenSIM has three advantages ^[5]:

(1) Flexibility: GPenSIM can be combined conveniently with other libraries and tools, making it easy to build hybrid models. For example, Fuzzy Petri net is from the combination of Petri net and Fuzzy logic;

(2) Extensible: users can write their own extensions to extend the existing functions or create their own functions which can help them realize special needs;

(3) Easy of use: This software provides natural language user interface and hides the complex mathematical details which helps users avoid complex mathematical computations in the modeling process.

2.3 Particle filter

In statistics, particle filter is also called sequential Monte Carlo methods, which is a complex model estimation technique based on simulation. Particle filter plays an important role in econometrics and other fields. It represents probability with particle set and can be used on any forms of state space models. Its core idea is to express the distribution with random state particle extracted from posterior probability. It is one kind of Sequential Importance Sampling. In simple terms, particle filter is the process of approximating the probability

density function by looking for a set of random samples that spread in the state space and gaining the minimum variance distribution by replacing integration computation with sample means. Here the sample means particle. When the amount of samples (particles) $N \rightarrow \infty$, it can approach any form of probability density distribution ^{[6][7]}.

The superiority which the particle filter technique has shown in the nonlinear, non-Gaussian system makes its application range very wide. Additionally, the multi-mode processing ability of the particle filter is also one of the reasons which make it used widely. The particle filter has been used in many fields all over the world. In the economics field, it has been used for the economic data prediction; in the military field, it has been used for the radar to track aero planes and air to air, air to ground passive tracking; in the traffic control field, it has been used for video monitoring of the drivers and the cars; it has also been used for global localization of the robot ^{[6][7]}.

2.4 Intrusion detection system

Intrusion detection is a technique to detect the computer network and system actively and dynamically in order to recognize the events which violate the security strategy. It can find the security problems, recognize and alarm the intrusion activities, and take appropriate actions to prevent the intrusion events or make up for the damage of the computer and the network. The research of intrusion detection technology started in the late 1980s. It is a new type of network security technology that is developed on the basis of the traditional audit technology. It can monitor the security state of the system by means of statistics or intelligent analyzing algorithm according to the features of network intrusion or the traces left in the system log. Intrusion detection has become an important part of the network security system now. It will also be a hot spot in the research area of network security in the future. The intrusion detection can't stop some intrusion behaviors, but it can help the system administrator to prevent further attacks from the hacker. It makes up for the deficiency of the passive network security package ^[8].

The accuracy of the detection is a vital index to measure an intrusion detection system, which includes false reject rate and missing rate. The pattern matching method is a more mature method at present, and the biggest advantage of this method is high accuracy. But for the unknown types of attacks it can do nothing. How to improve the intelligence of the intrusion detection system has become a hot spot in research area. Many intelligent methods and theories such as artificial intelligence, immunity algorithm, machine learning, have been used in the intrusion detection field to improve the intelligence of the system ^[8].

3. Related Work

Petri nets are widely used for modeling and simulating networks.

In the paper “An Application of GSPN for Modeling and Evaluating Local Area Computer Networks”, Masahiro Tsunoyama and Hiroei Imai proposed a method for modeling local area computer networks used for processing and delivering multimedia data with Generalized Stochastic Petri Net. It can “evaluate the mean delay time and its jitter (standard deviation) for systems based on the GSPN model and tagged task approach “^[9].

Hugo Rodríguez, Rubén Carvajal, Beatriz Ontiveros and Ismael Soto proposed a method to model and formally analyze a communication system which holds an elliptic curve encryption scheme using Petri net in their paper “Using Petri Net for Modeling and Analysis of an Encryption Scheme for Wireless Sensor Networks”^[10].

Charles Lakos, John Lamp, Chris Keen and Brian Marriott examines “how object-oriented extensions to the Petri Net formalism can address a number of issues in the modeling of network protocols” in their paper “Modeling Network Protocols with Object Petri Nets”. “The object-oriented extensions lead to the formalism of Object Petri Nets, with a textual language form referred to as LOOPN++”. Their paper “considers practical examples for which clean, well-structured models can be created because of the support for modularity, inheritance, polymorphism, genericity, and mobile objects”^[11].

Congzhe Zhang and Mengchu Zhou present a Stochastic Petri net-based approach in their paper “A Stochastic Petri Net Approach to Modeling and Analysis of Ad Hoc Network”. In this paper, they illustrate how their model can exploit the characteristics of the system to construct a scalable model. The scheme they proposed is “a powerful analytical model that can be used to derive network performance much easier than a simulation-based approach”^[12].

4. Model and simulation

4.1 Overall design

Network intrusion detection system is used for the detection of misuse behaviors on computer systems so as to keep the reliability and availability of them. It plays an important role in the network defense system. The model is simulated with colored Petri nets. It is based on particle filter which makes use of a series of samples called particle to approach the posterior conditional probability density of the state variables of the object. The utilization of particle filter algorithm can filter the network packets to detect some abnormal packets. The block diagram of the network intrusion detection system proposed in this thesis is as

below:

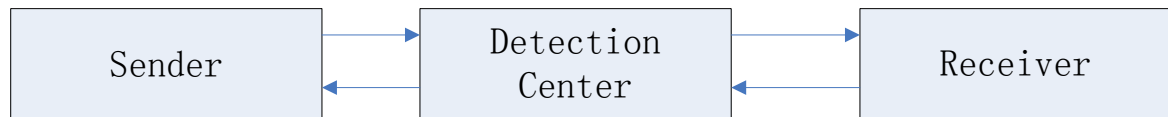


Figure 1 Block diagram for network detection system

This network intrusion detection system includes three parts:

1. Sender: It is mainly used to send the network packets.
2. Detection center: It mainly detects the network packets to find out anomalous packets and packets with intrusion behavior.
3. Receiver: It is mainly used to receive the network packets which are sent by the sender and send out confirmation packets.

The major part of this system is the detection center which mainly filters the network packets using particle filter algorithm. The major process is as below:

First, N number of characteristics are taken as particles; each of them is endowed with different weight.

Then, every packet in one time window which lasts for a few minutes is filtered to see whether it is anomalous or not.

At last, if any packet is confirmed to be anomalous, the packets with the same IP address, UDP port number and TCP port number need to be found in this time window and the weight of corresponding particles should be added.

In this thesis, two kinds of intrusion are studied: 1. Intrusion Detection 2. Trojan Detection.

4.2 Intrusion detection

4.2.1 Model Design

The intrusion packets are assumed to come from the sender, so the intrusion detection system we designed only detect the packets from the sender and ignore the confirmation packets from the receiver. The block diagram below shows the intrusion detection system:

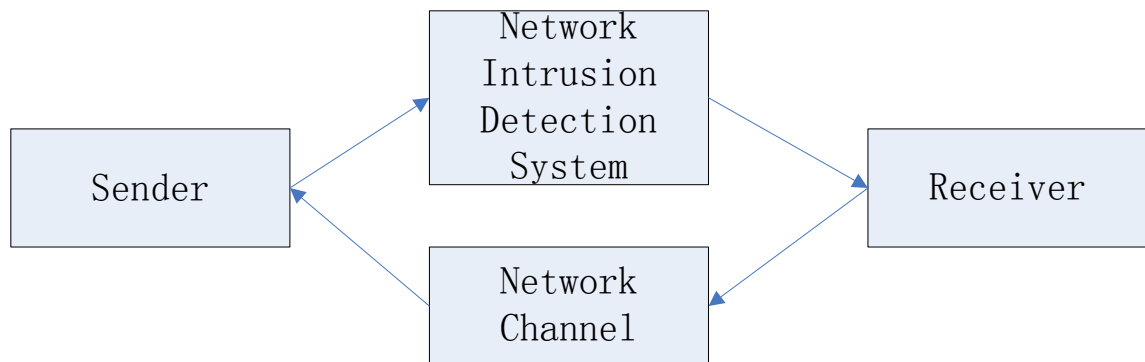


Figure 2 Block diagram for intrusion detection system

4.2.2 Petri net design

It is very difficult to completely simulate the real network environment, and it is also difficult to simulate the classification of network packets using particle filter, so the network intrusion detection system is simplified in this thesis. The colored Petri net is used for simulation.

GPenSIM toolbox is used for building the Petri nets. It works in the Matlab environment and is very powerful. It can also integrate with the available Matlab toolboxes.

The sender module is used to generate network packets randomly. The diagram for its Petri net is as below:

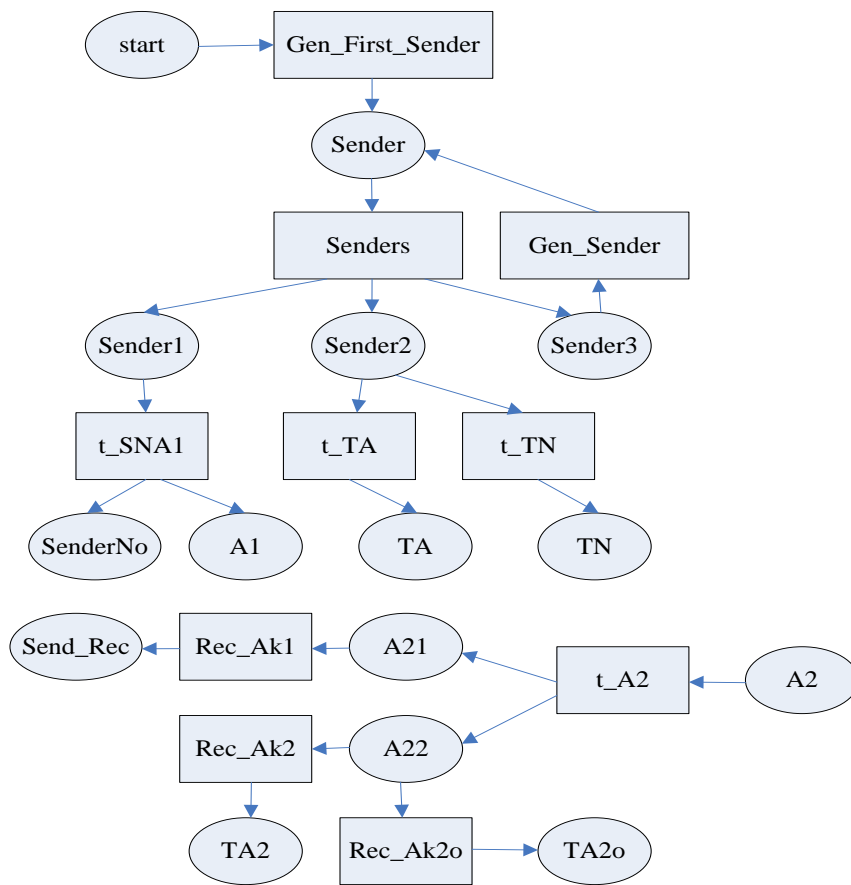


Figure 3 Diagram for the Petri net of sender module of intrusion detection system

The meanings of the places in the Petri net:

'start', this place is without color. The initial number of token is 1. It is used to generate the first packet.

'Sender', represents the network packet. Its color space is ('an', 'bshst', 'crhst', 'dprt', 'ptT'), of which the beginning characters (a, b, c, d, pt) have no specific meanings. They are used for ordering the colors when simulating. The meanings of the colors are as follows: the letter 'n' in 'an' is to mark the network packets and n is added by 1 when a new packet is generated. The 'shst' in 'bshst' stands for the IP address of the sender. Ten sets of sender computer systems are set up in this simulation and they are indicated by integers from 1 to 10 which are generated randomly. The 'rhst' in 'crhst' represents the IP address of the receiver which is the destination. For only three sets of receiver computer systems are set up in this simulation, they are indicated by integers from 1 to 3 which are generated randomly. The 'prt' in 'dprt' represents the TCP port number of the receiver which is

indicated by integers from 1 to 100 which are generated randomly. And the letter 'T' in 'ptT' represents the content of the network packets which are indicated by 'N' and 'A'. 'N' stands for the normal packets, while 'A' stands for anomalous packets. The probability to generate anomalous packets in this simulation is 10%.

'Sender 1': its color space is the same as 'Sender'. It acts as a transit station to produce places 'Sender No' and 'A1'.

'Sender 2': its color space is the same as 'Sender', and it also acts as a transit to generate place 'TA' and place 'TN'.

'Sender 3': its color space is the same as 'Sender'. It acts as a production device to generate network packets continuously, which will be put into the place 'Sender'.

'Sender No': it is used for the storage and statistics of all the packets generated in the simulation.

'A1': this place stands for the network packets that will be sent to the receiver.

'TA': it is used for the storage and statistics of all the anomalous packets generated in this simulation.

'TN': it is used for the storage and statistics of all the normal packets generated in this simulation.

'A2': it stands for the confirmation packets sent by the receiver. It is the output of the former module.

'A21': it acts as a transit to generate place 'Send_Rec'.

'A22': it acts as a transit to generate places 'TA2' and 'TA2o'.

'Send_Rec': it is used for the storage and statistics of the received confirmation packets sent by the receiver.

'TA2': it is used for the storage and statistics of the received abnormal confirmation packets sent by the receiver.

'TA2o': it is used for the storage and statistics of the received normal confirmation packets sent by the receiver.

Meanings of the transitions in the Petri net:

'Gen_First_Sender': it is used to generate the first network packet. When the simulation starts, its input place contains one token, and it will fire immediately.

'Senders': it is used to generate three network packets. When its input place 'Sender' contains token and reach the time to fire, it fires. In this simulation, GPenSIM tools and Matlab's own toolbox Stochastic Firing Times are combined for programming. Function 'unifrnd' is used to generate firing time randomly.

'Gen-Sender': it acts as a generator to generate network packets continuously. When the input place contains token, it fires. This transition is designed not to inherit the color space from input place. This function can be realized perfectly by GPenSIM. We only need to set transition.override=1. New network packets will be generated continuously.

't_TA': it is used for the statistics of anomalous packets. When 'ptT' in the color space of input place is 'ptA', it fires.

't_SNA1', it is used to generate the packets that will be sent to the receiver and the packets for statistics of the number of sent packets. It fires when its input place contains token.

't_TN', it is used for statistics of normal packets. When 'ptT' in the color space of its input place is 'ptN', it fires.

't_A2': it acts as a transit to generate packets for two channels that will be used for the follow-up operation.

'Rec_Ak1': it is used for statistics of all the received confirmation packets sent from the receiver.

'Rec_Ak2': it is used for statistics of all the received abnormal packets sent from the receiver. When 'ptT' in the color space of its input place is 'ptA', it fires.

'Rec_Ak2o': it is used for statistics of all the received normal packets sent from the receiver. When 'ptT' in the color space of its input place is 'ptN', it fires.

The analysis of the sender module:

The sender module mainly consists of two parts: 1. the part used for generation, statistics, and sending of network packets; 2. the part used for receiving and statistics of the confirmation packets sent from the receiver.

The first part is used for generation, statistics, and sending of network packets. The changing process of system status is as follows: First, the first network packet is generated from the firing of transition 'Gen_First_Sender'. Then, it waits for the arrival of the trigger time of transition 'Senders', and the trigger time is evenly distributed random integers. Three network packets are generated from the firing of 'Senders' and they are separately sent to three channels: the channel used for statistics and sending, the channel for packets

generator and the channel used for statistics of anomalous packets. The transition 't_SNA1' for the first channel is immediately triggered to generate the token for place 'SenderNo' and the token for place 'A1'. The channel for packets generator generates a network packet randomly and then waits for the trigger of transition 'Senders'. The channel for statistics of anomalous packets is used for the statistics of anomalous packets and normal packets and they are stored separately in place 'TA' and 'TN'. 'A1' is used as the input for the next module.

The second part is used for receiving and statistics of the confirmation packets sent by the receiver. The changing process of the system status is as follows: Place 'A2' stores the confirmation packets sent from the receiver. Two same packets are generated after the transition 't-A2' has a packet as input and they are sent separately to two channels: the channel for statistics of all confirmation packets and the channel for statistics of anomalous confirmation packets. The transition for the channel for statistics of all the confirmation packets is triggered immediately and place 'Send-Rec' stores all the received confirmation packets. If the confirmation packets are anomalous, transition 'Rec-Ak2' fires and cause the place 'TA2' to change. 'TA2' is used for statistics and storage of the anomalous confirmation packets sent by the receiver, If the confirmation packets are normal, transition 'Rec-Ak2' fires and cause place 'TA2o' to change. 'TA2o' is used for statistics and storage of the normal confirmation network packets sent by the receiver.

The second module is the network intrusion detection module. It is mainly used for the filtering of the sent packets to see whether they are anomalous or normal. Then it makes statistics and sends the normal ones to the next module. As it is difficult to fully simulate particle filter algorithm, this part is simplified in the model.

The diagram for the Petri net is as below:

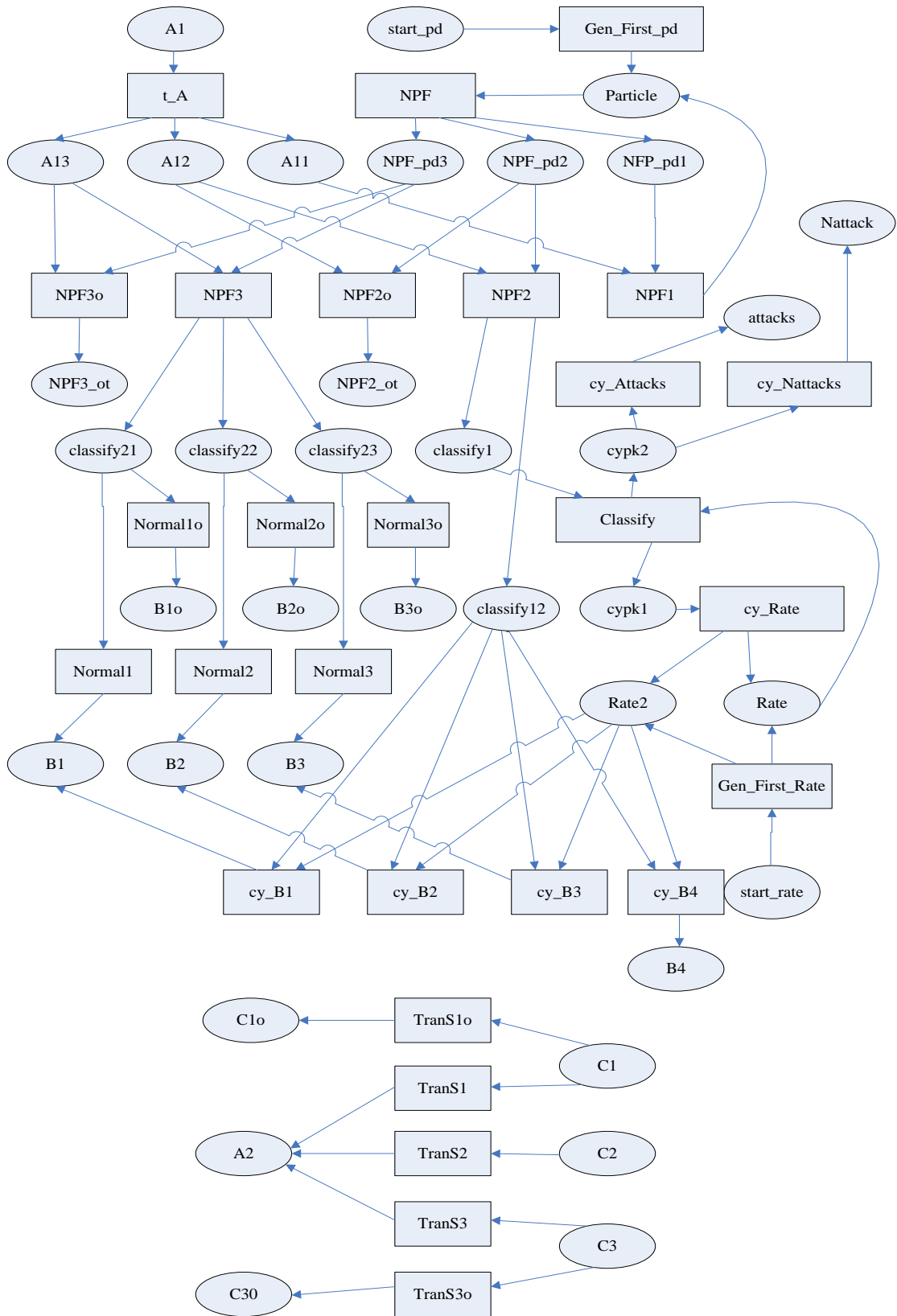


Figure 4 Diagram for the Petri net of intrusion detection module

The meanings of the places of the Petri net:

'start_pd': it is without color information. The initial number of token is 1. It is used as generator of the particles in the particle filter.

'Particle': it stands for the particles in the particle filter. Its color space is the same as the color space of the packets. The difference between them is that the 'n' in the first color 'an' in this place is represented by a random integer from 1 to 1000 while the one in the network packets stands for the generating order of the packets.

'NPF_Pd1': it stands for the particles used for the channel of particle generator.

'NPF_Pd2': it stands for the particles used for the anomalous packets channel.

'NPF_Pd3': it stands for the particles used for the normal packets channel.

'A11': it stands for the packets used in the particle generator channel.

'A12': it stands for the packets used in the anomalous packets channel.

'A13': it stands for the packets used in the normal packets channel.

'NPF2_ot': it has no specific meaning and acts as a garbage station.

'NPF3_ot': it has no specific meaning and acts as a garbage station.

'classify21': this place stands for the normal packets sent to NO.1 computer in the receiver part.

'classify22': this place stands for the normal packets sent to NO.2 computer in the receiver part.

'classify23': this place stands for the normal packets sent to NO.3 computer in the receiver part.

'classify1': this place stands for the suspicious packets judged by the particle filter. And they are sent to the first purification channel for further judgment to see whether they are intrusion packets or not.

'classify12': this place stands for the suspicious packets judged by the particle filter. And they are sent to the second purification channel for further judgment to see whether they are intrusion packets or not.

'start_Rate': it has no color space. Its initial number of tokens is one, and is used to generate the first rate when simulation begins.

'Rate': this place stands for a rate. Its color space is 'r' and it is represented by a random number from 1 to 1000. It uses the first purification channel.

'Rate2': this place also stands for a rate. Its color space is 'r' and it is represented by a random number from 1 to 1000. It uses the second purification channel.

'cypk1': it is used to generate the next rate continuously.

'cypk2': this place stores the suspicious packets and the packets will be further judged to see whether they are intrusion packets.

'attacks': it is used for the storage and statistics of the intrusion packets detected by the network intrusion detection system.

'Nattacks': it is used for the storage and statistics of the packets that are judged not to be intrusion packets in the suspicious packets.

'B1': it stores the packets judged to be normal by the network intrusion detection system and the sending target is NO.1 computer. It will be used as the input of the receiver module.

'B2': it stores the packets judged to be normal by the network intrusion detection system and the sending target is NO.2 computer. It will be used as the input of the receiver module.

'B3': it stores the packets judged to be normal by the network intrusion detection system and the sending target is NO.3 computer. It will be used as the input of the receiver module.

'B4': it stores the packets judged to be suspicious by the network intrusion detection system and abandoned by the purification system. That is to say, the packets are neither intrusion nor normal packets. They are the packets lost in the process of sending.

'B1o': this place stands for the packets judged to be normal by the network intrusion detection system, but the target of these packets is not NO.1 computer.

'B2o' this place stands for the packets judged to be normal by the network intrusion detection system, but the target of these packets is not NO.2 computer.

'B3o' this place stands for the packets judged to be normal by the network intrusion detection system, but the target of these packets is not NO.3 computer.

'C1': this place stands for the confirmation packets sent by No .1 receiver computer.

'C2': this place stands for the confirmation packets sent by No .2 receiver computer.

'C3': this place stands for the confirmation packets sent by No .3 receiver computer.

'C1o': this place stands for the lost confirmation packets sent by No .1 receiver computer.

'C3o': this place stands for the lost confirmation packets sent by No.3 receiver computer.

'A2': this place stands for the collected confirmation packets sent by No.1, No.2 and No.3 receiver computers and it will be used as the input of the receiver module.

The meanings of the transitions in the Petri net:

't_A', it is used as a replicator to generate packets 'A11', 'A12' and 'A13' used in three channels.

'Gen_First_pd': it is used to generate particles of the particle filter. As there's a token in the initial place, it immediately fires when the simulation starts.

'NPF': it is used to generate the particles of the particle filter for three channels:

'NPF_pd1', 'NPF_pd2', and 'NPF_pd1'.

'NPF1': it is used to generate particles of the particle filter continuously. This transition is designed not to inherit the color space from the input place. This function can be realized very well with GPenSIM tools. We only need to set transition.override=1. New particles will be generated continuously. The serial number is generated to be a random integer from 1 to 1000 here while it is generated orderly in the sender module.

'NPF2': it is used to judge whether the packets sent from the sender are suspicious. The judging standard is based on the absolute value of the difference between the port number sent from the sender and that of the particle filter. If the absolute value is below a threshold value, this packet is suspicious, and then this transition fires and sends the suspicious packets to the purification channel for further judging.

'NPF2o': it acts as a recycle bin and has no practical meaning. It only fires when transition 'NPF2' doesn't fire and its own fire conditions are met.

'NPF3': it is used to judge whether the packets sent from the sender are normal. The judging standard is based on the absolute value of the difference between the port number sent from the sender and that of the particle filter. If the absolute value is above a threshold value and the content of the packet is 'ptN', the packet is considered to be normal. Then this transition fires and sends the normal packets to the follow-up receiver system.

'NPF3o': it acts as a recycle bin and has no practical meaning. It only fires when transition 'NPF3' doesn't fire and its own fire conditions are met.

'Normal1': it is used to filter the packets from the sender by judging whether they are sent to NO.1 receiver computer. If yes, it will fire.

'Normal1o': it is used to filter the packets that are not sent to No.1 receiver computer. It will fire when the receiver of packets is not NO.1 computer.

'Normal2': it is used to filter the packets from the sender by judging whether they are sent to NO.2 receiver computer. If yes, it will fire.

'Normal2o': it is used to filter the packets that are not sent to No.2 receiver computer. It will fire when the receiver of packets is not NO.2 computer.

'Normal3': it is used to filter the packets from the sender by judging whether they are sent to NO.3 receiver computer. If yes, it will fire.

'Normal3o': it is used to filter the packets that are not sent to No.3 receiver computer. It will fire when the receiver of packets is not NO.3 computer.

'Gen_First_Rate': this transition is used to generate the first rate. As initially its input place contains one token, it will fire immediately when the simulation begins.

'Classify': this transition works as a replicator. It is used to generate packets which will be sent to rate generating channel and intrusion packets judging channel.

'cy_Rate': this transition is used to generate rate continuously. It cannot inherit the color space from the input place either. It will produce a new color space as the new rate through a random number generating function.

'cy_Attacks': this transition is used for judging whether the suspicious network packet is an intrusion packet or not. The standard is whether the 'ptT' color space of the packet is 'ptA' or not. If yes, it fires.

'cy_Nattack': this transition is used for determining whether the suspicious network packet is a normal packet or not. The standard is whether the 'ptT' color space of the packet is 'ptN' or not. If yes, it fires.

'cy_B1': this transition is used for further filtering by judging whether the suspicious packets are normal network packets, and whether the destination address is NO.1 receiver computer. When the value of the input transition 'Rate2' is below a threshold and the receiver's IP is No.1, it fires.

'cy_B2': this transition is used for further filtering by judging whether the suspicious packets are normal network packets, and whether the destination address is NO.2 receiver computer. When the value of the input transition 'Rate2' is below a threshold and the receiver's IP is No.2, it fires.

'cy_B3': this transition is used for further filtering by judging whether the suspicious packets are normal network packets, and whether the destination address is NO.3 receiver computer. When the value of the input transition 'Rate2' is below a threshold and the receiver's IP is No.3, it fires.

'cy_B4': this transition is used for statistics of the lost suspicious packets-those who are neither judged to be intrusion packets nor sent to receiver. When the value of the input transition 'Rate2' is above a threshold, it fires.

'TranS1': this transition filters the confirmation packets from NO.1 computer. The condition is whether the serial number of the packet is greater than a random integer from 1 to 1000. That is to say, it receives packets with a certain probability.

'TranS1o': this transition is used for statistics of the filtered confirmation packets sent by NO.1 receiver computer, which are the lost confirmation packets.

'TranS2': this transition is used for statistics of the confirmation packets sent by NO.2 receiver computer.

'TranS3': this transition is used for filtering the confirmation packets sent by NO.3 receiver computer. The condition is that the serial number of the packet is less than a random integer from 1 to 1000. That is to say, it receives confirmation packets with a certain probability.

'TranS3o': this transition is used for statistics of the filtered confirmation packets sent by No.3 receiver computer which are the lost confirmation packets.

Analysis of network intrusion detection module:

This module includes two parts. One is filtering of the packets from the sender, and the other is filtering of the confirmation packets from the receiver. For the network intrusion detection system in this simulation just detect the packets from the sender, the part of filtering of the packets from the sender is more important, and only in this part we need the particle filter. The first part is analyzed specifically.

Part 1: filtering of packets from the sender:

When this module receives a network packet from the sender, an initial judgment will be made through the particle filter. It will be able to decide which are normal and which are suspicious. If it is a normal packet, transition 'NPF3' will fire, and the packet will be sent to the follow-up receiver. And then through IP address of the normal packets, the packets will be sent to different host. There are three hosts in this simulation. When the packet is judged to be suspicious, it will be sent to the follow-up purifying channel. The transition 'NPF2' will fire. A packet will be considered to be intrusion packet if its content is 'ptA'. Then it will be

sent to the place 'attacks', and will be used in the final analysis for the experiment. If it is not an intrusion packet, it will be sent to the sender with a certain probability.

Part 2: filtering of the packets from the receiver. The confirmation packets will be received with a certain probability. This part is comparatively easy. Finally, the received confirmation packets will be collected and sent to the sender module for statistics.

Receiver module: because this part has little effect on the network intrusion detection, it is simplified in the simulation. After receiving the packets, each receiver computer will send confirmation packets directly, without any filtering or handling.

The diagram of the Petri net is as below:

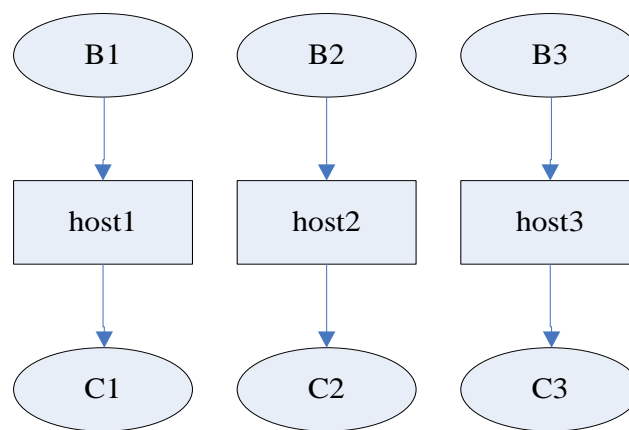


Figure 5 Diagram of the Petri net for the receiver

The meanings of the places in the Petri net:

'C1': it represents the confirmation packets from the receiver host1.

'C2': it represents the confirmation packets from the receiver host2.

'C3': it represents the confirmation packets from the receiver host3.

The meanings of the transitions in the Petri net:

'host1': this transition is from received packets to confirmation packets on receiver host1.

'host2': this transition is from received packets to confirmation packets on receiver host2.

'host3': this transition is from received packets to confirmation packets on receiver host3.

The analysis of the receiver module:

The receiver module is very simple. Each computer receives its own packets, and then sends confirmation packets.

Connection between the modules:

First is the connection between the sender module and the network intrusion detection module. The output of the sender module is the place 'A1' which will connect with the transition 't_A1' in the network intrusion detection module. And then, the place 'A2' for confirmation packets in intrusion detection module will be connected to the transition 't_A2' in the receiver module.

Next is the connection between the network detection module and the receiver module. The output places of the network intrusion detection module are 'B1', 'B2' and 'B3', which are the network packets sent to each host by the sender. They will be connected with the transitions 'host1', 'host2', 'host3' in the receiver module. And the output places of the receiver module are 'C1', 'C2' and 'C3', which are the confirmation packets that are sent from each receiver host. They will be connected with the transitions 'TranS1', 'TranS2', 'TranS3' in the network intrusion detection module. These transitions will further collect and filter the confirmation packets.

The connection of each module is achieved by using the modular tool of GPenSIM in the simulation. Modularity is a major feature of GPenSIM tools. It can help us simulate a complex network. With the modular tool, we can divide the complex Petri net into individual modules to define their network structures and the transition functions separately. Finally using a link function, we can connect the modules together. With this tool, the only thing we need to do is to define a link function.

The integral process of network intrusion detection system:

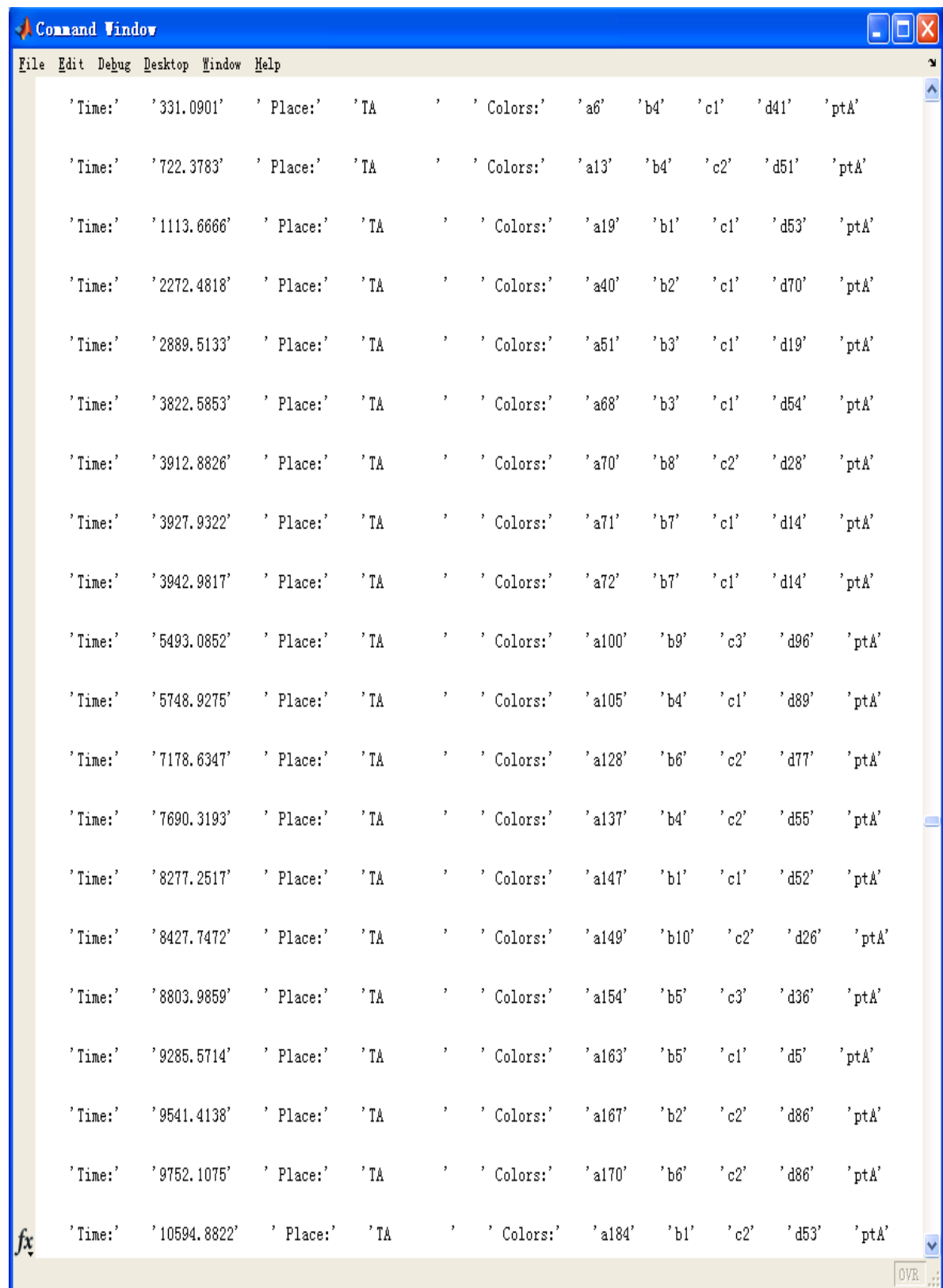
Firstly the sender module generates network packets continuously, and it will count the number of sent packets and abnormal packets, then save them in a global variable which is defined by the GPenSIM tools. The global variable can save the simulation data and use them for the final analysis and conclusion.

Then, whenever the sender sends a packet, the intrusion detection module will detect and classify it and count the number of the intrusion packets. It will also filter the packets from the sender and the confirmation packets from the receiver.

Finally, the receiver will receive the packets from the sender, and send confirmation packets to the intrusion detection module. The intrusion detection module will forward the packets to sender, and do some necessary statistics work.

4.2.3 Simulation result

The result of the simulation is shown as below:



```
Command Window
File Edit Debug Desktop Window Help
' Time: ' 331.0901 ' Place: ' TA ' Colors: ' a6 ' b4 ' c1 ' d41 ' ptA
' Time: ' 722.3783 ' Place: ' TA ' Colors: ' a13 ' b4 ' c2 ' d51 ' ptA
' Time: ' 1113.6666 ' Place: ' TA ' Colors: ' a19 ' b1 ' c1 ' d53 ' ptA
' Time: ' 2272.4818 ' Place: ' TA ' Colors: ' a40 ' b2 ' c1 ' d70 ' ptA
' Time: ' 2889.5133 ' Place: ' TA ' Colors: ' a51 ' b3 ' c1 ' d19 ' ptA
' Time: ' 3822.5853 ' Place: ' TA ' Colors: ' a68 ' b3 ' c1 ' d54 ' ptA
' Time: ' 3912.8826 ' Place: ' TA ' Colors: ' a70 ' b8 ' c2 ' d28 ' ptA
' Time: ' 3927.9322 ' Place: ' TA ' Colors: ' a71 ' b7 ' c1 ' d14 ' ptA
' Time: ' 3942.9817 ' Place: ' TA ' Colors: ' a72 ' b7 ' c1 ' d14 ' ptA
' Time: ' 5493.0852 ' Place: ' TA ' Colors: ' a100 ' b9 ' c3 ' d96 ' ptA
' Time: ' 5748.9275 ' Place: ' TA ' Colors: ' a105 ' b4 ' c1 ' d89 ' ptA
' Time: ' 7178.6347 ' Place: ' TA ' Colors: ' a128 ' b6 ' c2 ' d77 ' ptA
' Time: ' 7690.3193 ' Place: ' TA ' Colors: ' a137 ' b4 ' c2 ' d55 ' ptA
' Time: ' 8277.2517 ' Place: ' TA ' Colors: ' a147 ' b1 ' c1 ' d52 ' ptA
' Time: ' 8427.7472 ' Place: ' TA ' Colors: ' a149 ' b10 ' c2 ' d26 ' ptA
' Time: ' 8803.9859 ' Place: ' TA ' Colors: ' a154 ' b5 ' c3 ' d36 ' ptA
' Time: ' 9285.5714 ' Place: ' TA ' Colors: ' a163 ' b5 ' c1 ' d5 ' ptA
' Time: ' 9541.4138 ' Place: ' TA ' Colors: ' a167 ' b2 ' c2 ' d86 ' ptA
' Time: ' 9752.1075 ' Place: ' TA ' Colors: ' a170 ' b6 ' c2 ' d86 ' ptA
' Time: ' 10594.8822 ' Place: ' TA ' Colors: ' a184 ' b1 ' c2 ' d53 ' ptA
```

Figure 6 Simulation result of intrusion detection (1)


```

Command Window
File Edit Debug Desktop Window Help
' Time: ' 10910.9227 ' Place: ' TA ' Colors: ' a189' ' b3' ' c3' ' d55' ' ptA'
' Time: ' 11001.22 ' Place: ' TA ' Colors: ' a191' ' b2' ' c3' ' d85' ' ptA'
' Time: ' 11226.9632 ' Place: ' TA ' Colors: ' a195' ' b6' ' c2' ' d73' ' ptA'
' Time: ' 11257.0623 ' Place: ' TA ' Colors: ' a197' ' b8' ' c3' ' d45' ' ptA'
' Time: ' 12445.9766 ' Place: ' TA ' Colors: ' a218' ' b9' ' c1' ' d47' ' ptA'
' Time: ' 13394.0982 ' Place: ' TA ' Colors: ' a232' ' b3' ' c2' ' d86' ' ptA'
' Time: ' 14266.972 ' Place: ' TA ' Colors: ' a244' ' b9' ' c3' ' d22' ' ptA'
' Time: ' 14718.4585 ' Place: ' TA ' Colors: ' a253' ' b7' ' c3' ' d31' ' ptA'
' Time: ' 15019.4494 ' Place: ' TA ' Colors: ' a259' ' b2' ' c3' ' d98' ' ptA'
' Time: ' 15771.9269 ' Place: ' TA ' Colors: ' a271' ' b2' ' c3' ' d50' ' ptA'
' Time: ' 15967.571 ' Place: ' TA ' Colors: ' a274' ' b4' ' c1' ' d46' ' ptA'
' Time: ' 18465.7961 ' Place: ' TA ' Colors: ' a321' ' b8' ' c3' ' d24' ' ptA'
' Time: ' 18556.0933 ' Place: ' TA ' Colors: ' a323' ' b10' ' c2' ' d29' ' ptA'
' Time: ' 18721.6384 ' Place: ' TA ' Colors: ' a325' ' b8' ' c2' ' d26' ' ptA'
' Time: ' 19007.5798 ' Place: ' TA ' Colors: ' a329' ' b4' ' c2' ' d77' ' ptA'
' Time: ' 20933.922' ' Place: ' TA ' Colors: ' a364' ' b9' ' c1' ' d76' ' ptA'
' Time: ' 21114.5166 ' Place: ' TA ' Colors: ' a368' ' b10' ' c1' ' d30' ' ptA'
' Time: ' 21249.9625 ' Place: ' TA ' Colors: ' a370' ' b6' ' c1' ' d69' ' ptA'
' Time: ' 21385.4085 ' Place: ' TA ' Colors: ' a372' ' b4' ' c1' ' d24' ' ptA'
' Time: ' 21505.8049 ' Place: ' TA ' Colors: ' a374' ' b6' ' c1' ' d62' ' ptA'

```

Figure 7 Simulation result of intrusion detection (2)

```

Command Window
File Edit Debug Desktop Window Help
' Time: ' 21731.5481' ' Place: ' TA ' Colors: ' a378' ' b10' ' c1' ' d89' ' ptA'
' Time: ' 21866.994' ' Place: ' TA ' Colors: ' a381' ' b9' ' c3' ' d42' ' ptA'
' Time: ' 22107.7868' ' Place: ' TA ' Colors: ' a385' ' b9' ' c1' ' d13' ' ptA'
' Time: ' 22468.976' ' Place: ' TA ' Colors: ' a391' ' b1' ' c1' ' d24' ' ptA'
' Time: ' 22860.2642' ' Place: ' TA ' Colors: ' a397' ' b10' ' c1' ' d12' ' ptA'
' Time: ' 23010.7597' ' Place: ' TA ' Colors: ' a400' ' b6' ' c1' ' d68' ' ptA'
' Time: ' 23567.593' ' Place: ' TA ' Colors: ' a409' ' b9' ' c1' ' d85' ' ptA'
' Time: ' 24365.2191' ' Place: ' TA ' Colors: ' a420' ' b2' ' c3' ' d44' ' ptA'
' Time: ' 24575.9128' ' Place: ' TA ' Colors: ' a424' ' b6' ' c2' ' d69' ' ptA'
' Time: ' 24861.8542' ' Place: ' TA ' Colors: ' a428' ' b6' ' c3' ' d22' ' ptA'
' Time: ' 25524.0343' ' Place: ' TA ' Colors: ' a438' ' b2' ' c1' ' d3' ' ptA'
' Time: ' 25704.6289' ' Place: ' TA ' Colors: ' a442' ' b4' ' c2' ' d40' ' ptA'
' Time: ' 26216.3135' ' Place: ' TA ' Colors: ' a450' ' b6' ' c3' ' d82' ' ptA'
' Time: ' 27074.1378' ' Place: ' TA ' Colors: ' a464' ' b5' ' c3' ' d14' ' ptA'
' Time: ' 27676.1198' ' Place: ' TA ' Colors: ' a472' ' b3' ' c2' ' d92' ' ptA'
' Time: ' 28804.8359' ' Place: ' TA ' Colors: ' a492' ' b6' ' c2' ' d73' ' ptA'
' Time: ' 29376.7187' ' Place: ' TA ' Colors: ' a503' ' b4' ' c3' ' d42' ' ptA'
' Time: ' 29527.2142' ' Place: ' TA ' Colors: ' a505' ' b8' ' c2' ' d2' ' ptA'
' Time: ' 30324.8403' ' Place: ' TA ' Colors: ' a515' ' b10' ' c2' ' d92' ' ptA'
' Time: ' 30385.0385' ' Place: ' TA ' Colors: ' a517' ' b1' ' c2' ' d67' ' ptA'
fx
OVR

```

Figure 8 Simulation result of intrusion detection (3)

```

Command Window
File Edit Debug Desktop Window Help
' Time: ' 30430.1871' ' Place: ' TA ' Colors: ' a518' ' b5' ' c2' ' d99' ' ptA'
' Time: ' 30565.6331' ' Place: ' TA ' Colors: ' a521' ' b9' ' c2' ' d76' ' ptA'
' Time: ' 31318.1105' ' Place: ' TA ' Colors: ' a535' ' b1' ' c2' ' d24' ' ptA'
' Time: ' 31709.3988' ' Place: ' TA ' Colors: ' a543' ' b3' ' c1' ' d31' ' ptA'
' Time: ' 31995.3402' ' Place: ' TA ' Colors: ' a550' ' b1' ' c1' ' d93' ' ptA'
' Time: ' 32040.4888' ' Place: ' TA ' Colors: ' a553' ' b1' ' c3' ' d46' ' ptA'
' Time: ' 32446.8266' ' Place: ' TA ' Colors: ' a561' ' b9' ' c2' ' d61' ' ptA'
' Time: ' 32913.3626' ' Place: ' TA ' Colors: ' a571' ' b8' ' c1' ' d99' ' ptA'
' Time: ' 32928.4122' ' Place: ' TA ' Colors: ' a572' ' b6' ' c1' ' d51' ' ptA'
' Time: ' 33891.5833' ' Place: ' TA ' Colors: ' a591' ' b1' ' c2' ' d15' ' ptA'
' Time: ' 34899.9031' ' Place: ' TA ' Colors: ' a608' ' b4' ' c2' ' d62' ' ptA'
' Time: ' 35923.2724' ' Place: ' TA ' Colors: ' a623' ' b2' ' c3' ' d63' ' ptA'
' Time: ' 36570.4029' ' Place: ' TA ' Colors: ' a632' ' b9' ' c2' ' d64' ' ptA'
' Time: ' 37127.2362' ' Place: ' TA ' Colors: ' a641' ' b3' ' c3' ' d70' ' ptA'
' Time: ' 37247.6326' ' Place: ' TA ' Colors: ' a643' ' b10' ' c3' ' d60' ' ptA'
' Time: ' 37428.2272' ' Place: ' TA ' Colors: ' a647' ' b10' ' c1' ' d90' ' ptA'
' Time: ' 37638.9209' ' Place: ' TA ' Colors: ' a651' ' b10' ' c3' ' d76' ' ptA'
' Time: ' 39023.4794' ' Place: ' TA ' Colors: ' a676' ' b8' ' c3' ' d36' ' ptA'
' Time: ' 39595.3622' ' Place: ' TA ' Colors: ' a688' ' b6' ' c3' ' d14' ' ptA'
' Time: ' 39670.6099' ' Place: ' TA ' Colors: ' a689' ' b4' ' c1' ' d45' ' ptA'

```

Figure 9 Simulation result of intrusion detection (4)

```

Command Window
File Edit Debug Desktop Window Help
' Time: ' 11257.0623' ' Place: ' attacks ' Colors: ' a197' ' b8' ' c3' ' d45' ' ptA'
' Time: ' 12445.9766' ' Place: ' attacks ' Colors: ' a218' ' b9' ' c1' ' d47' ' ptA'
' Time: ' 15771.9269' ' Place: ' attacks ' Colors: ' a271' ' b2' ' c3' ' d50' ' ptA'
' Time: ' 18556.0933' ' Place: ' attacks ' Colors: ' a323' ' b10' ' c2' ' d29' ' ptA'
' Time: ' 18721.6384' ' Place: ' attacks ' Colors: ' a325' ' b8' ' c2' ' d26' ' ptA'
' Time: ' 20933.922' ' Place: ' attacks ' Colors: ' a364' ' b9' ' c1' ' d76' ' ptA'
' Time: ' 21114.5166' ' Place: ' attacks ' Colors: ' a368' ' b10' ' c1' ' d30' ' ptA'
' Time: ' 21866.994' ' Place: ' attacks ' Colors: ' a381' ' b9' ' c3' ' d42' ' ptA'
' Time: ' 22468.976' ' Place: ' attacks ' Colors: ' a391' ' b1' ' c1' ' d24' ' ptA'
' Time: ' 24861.8542' ' Place: ' attacks ' Colors: ' a428' ' b6' ' c3' ' d22' ' ptA'
' Time: ' 30430.1871' ' Place: ' attacks ' Colors: ' a518' ' b5' ' c2' ' d99' ' ptA'
' Time: ' 31318.1105' ' Place: ' attacks ' Colors: ' a535' ' b1' ' c2' ' d24' ' ptA'
' Time: ' 34899.9031' ' Place: ' attacks ' Colors: ' a608' ' b4' ' c2' ' d62' ' ptA'
' Time: ' 37127.2362' ' Place: ' attacks ' Colors: ' a641' ' b3' ' c3' ' d70' ' ptA'
' Time: ' 37428.2272' ' Place: ' attacks ' Colors: ' a647' ' b10' ' c1' ' d90' ' ptA'

The whole number of the send packages is : 696
The whole number of the attack packages is :80
The number of attack packages captured by system is:15
The detect rate is :0.187500
fx >>
DVR

```

Figure 10 Simulation result of intrusion detection (5)

The curve chart for intrusion detection simulation is shown as below:

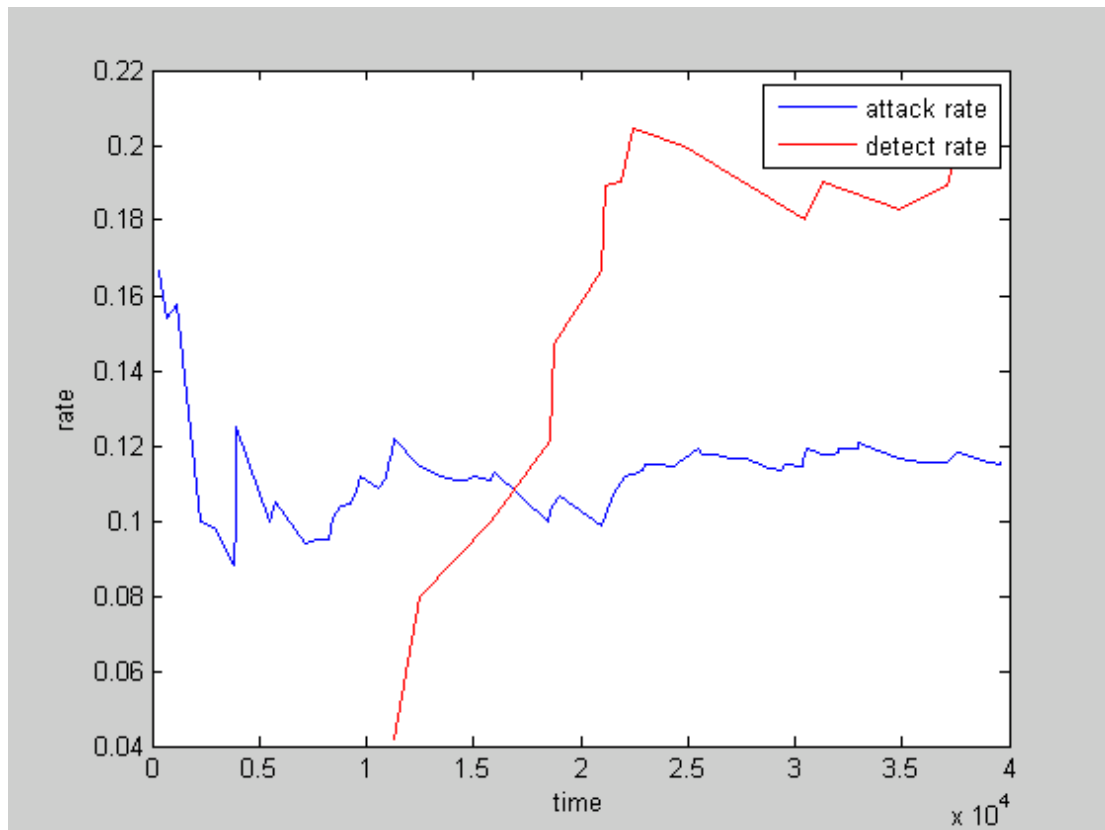


Figure 11 The curve chart for intrusion detection simulation

Figure 6 to Figure 10 show the output data of Matlab in the process of simulation. Each line of the data represents the status of the Petri net after one more intrusion packet is detected. The data follows 'time' is the running time of the program. The data follows 'place' is the place containing the intrusion packet. The data follows 'place' is the color space of this place.

Figure 11 shows the change of the detected rate and the percentage of abnormal packets in all the packets, in which the horizontal axis stands for time and the vertical axis stands for percentage. The red curve shows the changing percentage of the detected intrusion packets in all the intrusion packets. The blue curve shows the changing percentage of the intrusion packets in all the packets. According to the result of the simulation we can know that after 40000 unit time 696 packets are sent, including 80 intrusion packets, in which 15 of them are detected. The detection accuracy rate is 0.1875.

4.3 Trojan detection

4.3.1 Model design

We assume that all the Trojan packets are from the receiver, so we just detect the confirmation packets from receiver for Trojan detection in the network intrusion detection

system. The packets from sender will be ignored, which is opposite to the intrusion detection. The block diagram for the model of Trojan detection system is shown as below:

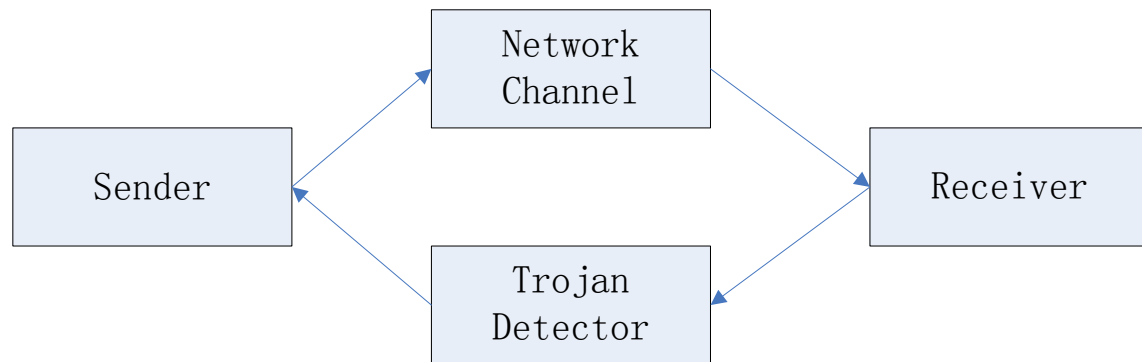


Figure 12 Block diagram for the model of Trojan detection system

4.3.2 Petri net design

In this thesis, we use GPenSIM tools for modeling and simulation. Trojan detection system also contains three modules.

The sender module is used to generate network packets randomly. The diagram for its Petri net is as below:

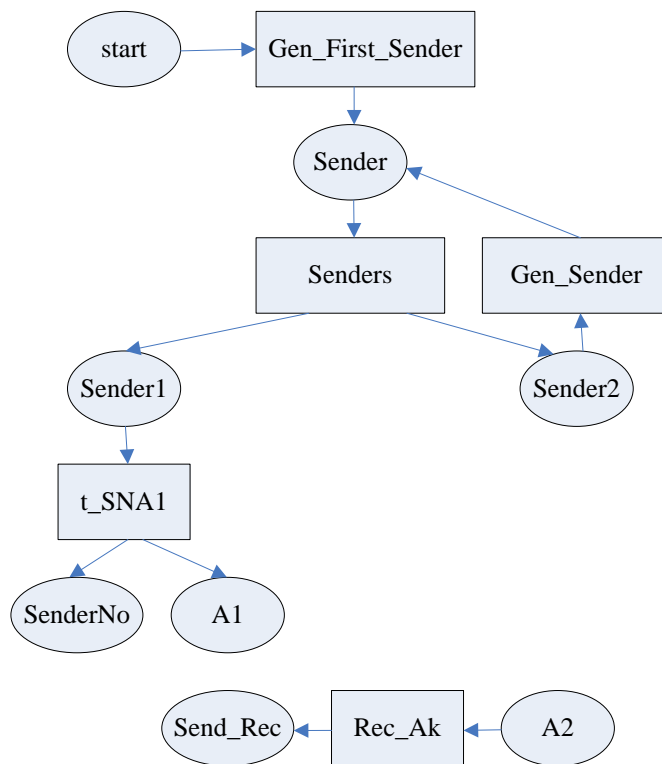


Figure 13 Diagram for the Petri net of sender of Trojan detection

The meanings of the places in the Petri net:

'start', this place is without color. The initial number of token is 1. It is used to generate the first packet.

'Sender ': it represents the network packets. The packets contain three parts: the IP address of the sender, the IP address of the receiver, and the TCP port number. The color space is {'an', 'bshst', 'crhst', 'dprt'}, of which the beginning characters (a, b, c, d) have no specific meanings. They are used for ordering of the colors when simulating. The meanings of the colors are as follows: the letter 'n' in 'an' is to mark the network packets and n is added by 1 when a new packet is generated. The 'shst' in 'bshst' stands for the IP address of the sender. Ten sets of sender computer systems are set up in this simulation and they are indicated by integers from 1 to 10 which are generated randomly. The 'rhst' in 'crhst' represents the IP address of the receiver which is the destination. For only three sets of receiver computer systems are set up in this simulation, they are indicated by integers from 1 to 3 which are generated randomly. The 'prt' in 'dprt' represents the TCP port number of the receiver which is indicated by integers from 1 to 100 which is generated randomly.

'Sender 1': its color space is the same as 'Sender'. It acts as a transit station to produce places 'Sender No' and 'A1'.

'Sender 2': its color space is the same as 'Sender'. It acts as a production device to generate network packets continuously, which will be put into place 'Sender'.

'Sender No': it is used for the storage and statistics of all the packets generated in the simulation.

'A1': this place stands for the network packets that will be sent to the receiver.

'A2': it stands for the confirmation packets sent by the receiver. It is the output of the former module.

'Send_Rec': it is used for the storage and statistics of the received confirmation packets sent by the receiver.

Meanings of the transitions in the Petri net:

'Gen_First_Sender': it is used to generate the first network packet. When the simulation starts, its input place contains one token, and it will fire immediately.

'Senders': it is used as a replicator and to generate two network packets. When its input place 'Sender' contains token and reach the time to fire, it fires. In this simulation, GPenSIM tools and Matlab's own toolbox Stochastic Firing Times are combined for programming. Function 'unifrnd' is used to generate firing time randomly.

Gen-Sender: it acts as a generator to generate network packets continuously. When the input place contains token, it fires.

't_SNA1', it is used to generate the packets that will be sent to the receiver and the packets for statistics of the number of sent packets. It fires when its input place contains token.

'Rec_Ak': it is used for statistics of all the received confirmation packets sent from the receiver.

The analysis of the sender module::

The sender module mainly consists of two parts: 1. the part used for generation, statistics, and sending of network packets; 2. the part used for receiving and statistics of the confirmation packets sent from the receiver.

The first part is used for generation, statistics, and sending of network packets. The changing process of system status is as follows: First, the first network packet is generated from the firing of transition 'Gen_First_Sender'. Then, it waits for the arrival of the trigger time of transition 'Senders', and the trigger time is evenly distributed random integers. Two network packets are generated from the firing of 'Senders' and they are separately sent to two channels: the channel used for statistics and sending, the channel for packets generator. The transition 't_SNA1' for the first channel is immediately triggered to generate the token for place 'SenderNo' and the token for place 'A1'. The channel for packets generator generates a network packet randomly and then waits for the trigger of transition 'Senders'. 'A1' is used as the input for the next module.

The second part is used for receiving and statistics of the confirmation packets sent by the receiver. The changing process of the system status is as follows: Place 'A2' stores the confirmation packets sent from the receiver. The transition 'Rec_AK' for statistics of all the confirmation packets is triggered immediately. Place 'Send-Rec' stores all the received confirmation packets.

The second module is the Trojan intrusion detection module. It is mainly used for the filtering of the confirmation packets sent by the receiver to see whether they are Trojan or normal. Then it makes statistics and sends the normal confirmation packets to the sender.

The diagram for the Petri net is as below:

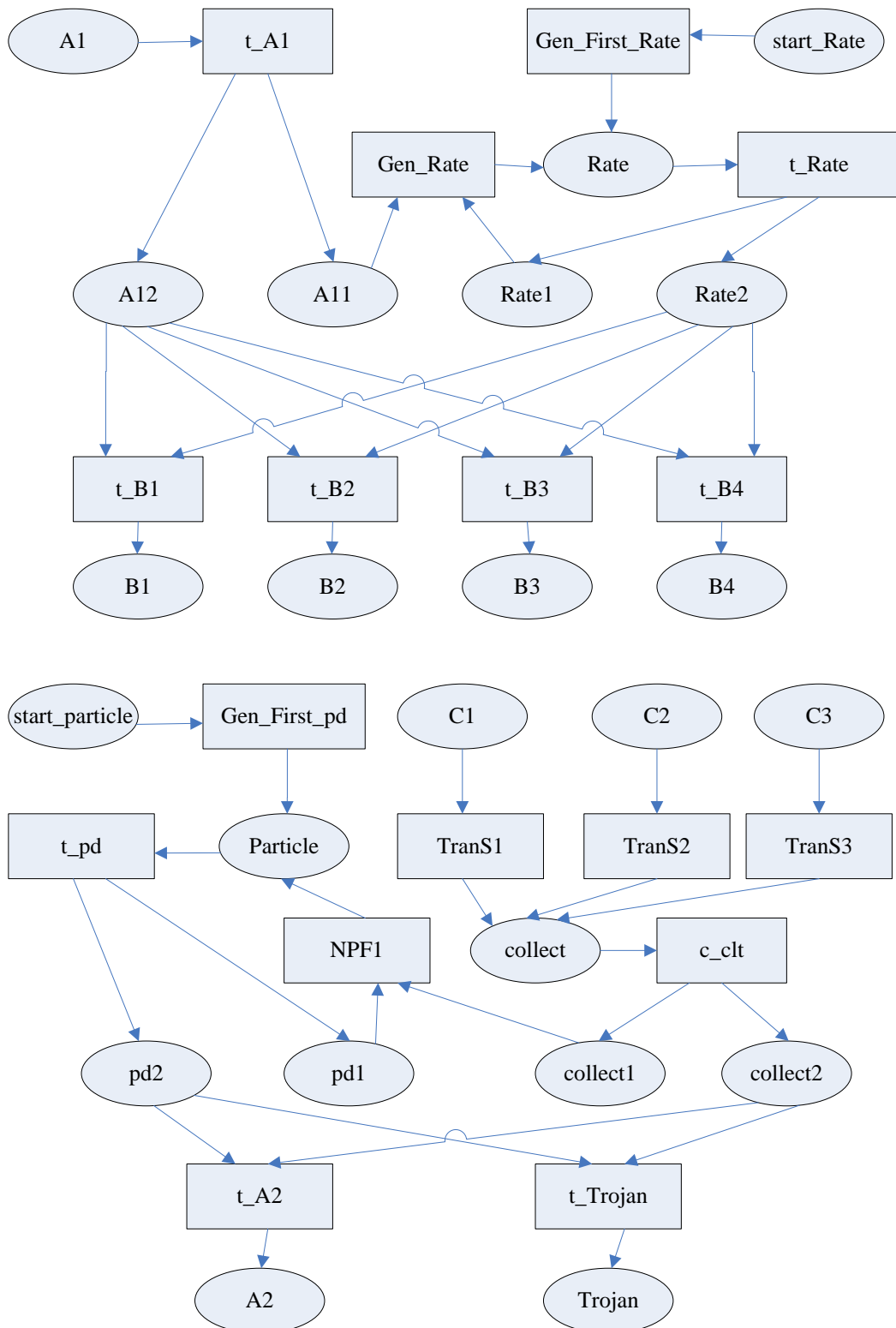


Figure 14 Diagram for the Petri net of detection module of Trojan detection simulation

The meanings of the places of the Petri net:

'Start-Rate': it has no color space. Its initial number of tokens is one, and is used to generate the first rate when simulation begins.

'Rate1': this place stands for a rate. Its color space is 'r' and it is represented by a random number from 1 to 1000. It is used to generate the next rate.

'Rate2': this place also stands for a rate. Its color space is 'r' and it is represented by a random number from 1 to 1000. It is used for filtering the packets from sender.

'B1': it stores the packets that have passed the filtration and sending target is NO.1 receiver computer. It will be used as the input of the receiver module.

'B2': it stores the packets that have passed the filtration and sending target is NO.2 receiver computer. It will be used as the input of the receiver module.

'B3': it stores the packets that have passed the filtration and sending target is NO.3 receiver computer. It will be used as the input of the receiver module.

'B4' stores the packets that have failed to pass the filtration. In other words, they are the lost packets during the transmission process.

'C1': this place stands for the confirmation packets sent by No .1 receiver computer.

'C2': this place stands for the confirmation packets sent by No .2 receiver computer.

'C3': this place stands for the confirmation packets sent by No .3 receiver computer.

'collect': this place represents the confirmation packets sent by NO.1, NO.2 and NO.3 receiver computers.

'collect1': this place represents the confirmation packets used for particle generator channel.

'collect 2': this place represents the confirmation packets used for Trojan Detection channel.

'Start_pd': it is without color information. The initial number of token is 1. It is used as generator of the first particle in the particle filter.

'Particle': it stands for the particles in the particle filter. Its color space is the same as the color space of the confirmation packets. The difference between them is that the 'n' in the first color 'an' in this place is represented by a random integer from 1 to 1000.

'Pd1': it stands for the particle used for the particle generator channel.

'Pd2': it stands for the particle used for the Trojan detection channel.

'Trojan': it is used for storage and statistics of the Trojan packets detected by the Trojan detection system.

'A2': this place represents the normal network confirmation packets which have passed the Trojan detection system. And it will be sent to the sender.

The meanings of the transitions in the Petri net:

'Gen_First_Rate': this transition is used to generate the first rate. As initially its input place contains one token, it will fire immediately when the simulation begins.

't_Rate': this transition acts as a replicator to generate two same rates which will be used in the rate generator channel and packets filtering channel.

'Gen_Rate': it is used to generate the next rate.

't_B1': this transition is used for filtering the packets from sender and judge whether it will be sent to NO.1 receiver computer. If the destination address is NO.1 receiver computer and the rate is less than a threshold, it fires.

't_B2': this transition is used for filtering the packets from sender and judge whether it will be sent to NO.2 receiver computer. If the destination address is NO.2 receiver computer and the rate is less than a threshold, it fires.

't_B3': this transition is used for filtering the packets from sender and judge whether it will be sent to NO.3 receiver computer. If the destination address is NO.3 receiver computer and the rate is less than a threshold, it fires.

't_B4': this transition is used for statistics of the lost network packets, which fail to pass the filtration. When the value of the input token from 'Rate2' is higher than a threshold, it fires.

'TranS1': this transition is used for statistics of confirmation packets sent by No.1 receiver computer.

'TranS2': this transition is used for statistics of confirmation packets sent by No.2 receiver computer.

'TranS3': this transition is used for statistics of confirmation packets sent by No.3 receiver computer.

't_clt': this transition acts as a replicator. It is used for generating two confirmation packets which are used in particle generating channel and Trojan detection channel.

'Gen_First_pd': it is used to generate particles of the particle filter. As there's a token in the initial place, it immediately fires when the simulation starts.

't_pd': this transition acts as a replicator. It is used for generating two particles which are used in Trojan detection channel and particle generating channel.

'NPF1': it is used to generate particles of the particle filter. This transition is designed not to inherit the color space from its input place. This function can be realized well by GPenSIM tools. We only need to set transition.override=1. New particles can be generated continuously.

't_Trojan': it is used to judge whether the confirmation packets sent by the receiver are Trojan packets. The judging standard is based on the absolute value of the difference between the port number of the packet and that of the particle. If the absolute value is smaller than a threshold value, this packet is a Trojan packet. Then the transition fires, and sends the packet to the place 'Trojan' for statistics.

't_A2': it is used to judge whether the confirmation packets sent by the receiver are normal. The judging standard is based on the absolute value of the difference between the port number of the packet and that of the particle. If the absolute value is higher than a threshold value, this packet is normal, then this transition will fire and send the packet to the follow-up receiver system.

This module includes two parts. One is used for filtering of the confirmation packets from the receiver, and the other is used for filtering of the packets from the sender. For the Trojan detection system in this simulation just detects the confirmation packets from the receiver, the part for filtering of the confirmation packets from the receiver is more important. And only in this part we need the particle filter, so this part is analyzed specifically.

Part 1: filtering of packets from the sender:

The packets will be decided whether to be sent or not with a certain probability. This part is simple. At last, it will be sent to different receiver host according to the port number of the packet.

Part two: filtering of the packets from the receiver.

When this module has received the confirmation packet from the receiver, a judgment will be made by the particle filter to decide which normal packets are and which Trojan packets are. The judging standard is based on the absolute value of the difference between the port number of the confirmation packets and that of the particle.

Receiver module: because this part has little effect on the network intrusion detection, it is simplified in the simulation. After receiving the packets, each receiver computer will send confirmation packets directly, without any filtering or handling.

The diagram of the Petri net is as below:

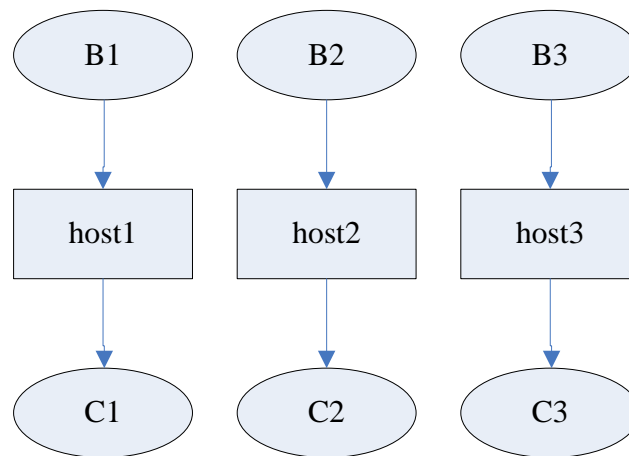


Figure 15 Diagram for the Petri net of receiver of Trojan detection

The meanings of the places of the Petri net:

'C1': it represents the confirmation packets from the host1 receiver.

'C2': it represents the confirmation packets from the host2 receiver.

'C3': it represents the confirmation packets from the host3 receiver.

The meanings of transitions in Petri net:

'host1': this transition is from received packets to confirmation packets on receiver host1.

'host2': this transition is from received packets to confirmation packets on receiver host2.

'host3': this transition is from received packets to confirmation packets on receiver host3.

The analysis of the receiver module:

The receiver module is very simple. Each computer receives its own packets, and then sends confirmation packets.

Connection of the modules:

First is the connection between the sender module and the Trojan detection module. The output of the sender module is the place 'A1' which will be connected with the transition 't_A1' in the Trojan detection module. And then, the place 'A2' for confirmation

packets sent by the receiver will be connected to the transition 't_A2' in the detection module.

Next is the connection between the Trojan detection module and the receiver module. The output places of the Trojan detection module are 'B1', 'B2' and 'B3', which are the network packets sent by each host of the sender. They will be connected to the transitions 'host1', 'host2', 'host3' in the receiver module. And the output places of the receiver module are 'C1', 'C2' and 'C3', which are the confirmation packets that are sent from each receiver host. They will be connected to the transitions 'TranS1', 'TranS2', 'TranS3' in the Trojan detection module. These transitions will further collect and filter the confirmation packets.

The connection of each module is also achieved by using the modular tool of GPenSIM tools in the simulation. The overall process of the Trojan detection system:

Firstly the sender module generates network packets continuously, and it will count the number of sent packets and received confirmation packets, then save them in a global variable which is defined by the GPenSIM tools.

Then, each packet sent by the sender will be filtered by the Trojan detection module with a certain probability, and send the passed packets to the receiver. The number of Trojan packets will be counted when the confirmation packet from the receiver is received.

At last, the receiver will receive the packets from the sender, and send confirmation packets to the Trojan detection module. The normal confirmation packets will be sent to the sender.

4.3.3 simulation result

The result of the simulation is shown as below:

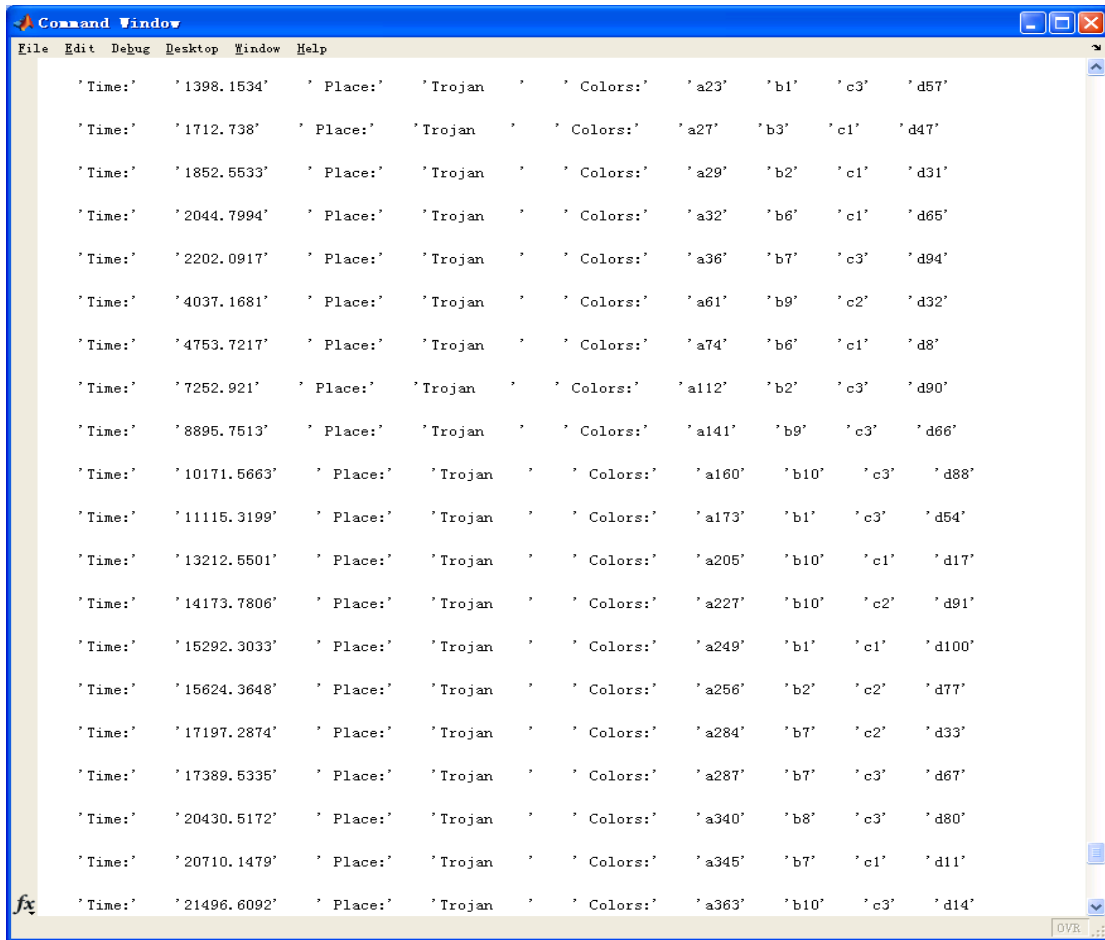


Figure 16 Result of Trojan detection(1)


```

Command Window
File Edit Debug Desktop Window Help
' Time: ' 22143.2552' ' Place: ' Trojan ' Colors: ' a373' ' b4' ' c3' ' d27'
' Time: ' 24327.8699' ' Place: ' Trojan ' Colors: ' a408' ' b1' ' c2' ' d94'
' Time: ' 24852.1775' ' Place: ' Trojan ' Colors: ' a420' ' b8' ' c2' ' d37'
' Time: ' 27089.223' ' Place: ' Trojan ' Colors: ' a452' ' b1' ' c3' ' d91'
' Time: ' 28382.5149' ' Place: ' Trojan ' Colors: ' a479' ' b6' ' c1' ' d46'
' Time: ' 28539.8072' ' Place: ' Trojan ' Colors: ' a482' ' b1' ' c1' ' d56'
' Time: ' 28662.1456' ' Place: ' Trojan ' Colors: ' a485' ' b1' ' c1' ' d33'
' Time: ' 28679.6225' ' Place: ' Trojan ' Colors: ' a486' ' b1' ' c1' ' d2'
' Time: ' 28976.7302' ' Place: ' Trojan ' Colors: ' a492' ' b3' ' c2' ' d42'
' Time: ' 29675.8069' ' Place: ' Trojan ' Colors: ' a506' ' b5' ' c3' ' d42'
' Time: ' 30409.8374' ' Place: ' Trojan ' Colors: ' a516' ' b10' ' c2' ' d76'
' Time: ' 31143.868' ' Place: ' Trojan ' Colors: ' a529' ' b4' ' c2' ' d2'
' Time: ' 31248.7295' ' Place: ' Trojan ' Colors: ' a530' ' b7' ' c1' ' d54'
' Time: ' 31353.591' ' Place: ' Trojan ' Colors: ' a533' ' b10' ' c3' ' d74'
' Time: ' 31738.0832' ' Place: ' Trojan ' Colors: ' a539' ' b10' ' c3' ' d25'
fx
OVR

```

Figure 17 Result of Trojan detection(2)

```

Command Window
File Edit Debug Desktop Window Help
' Time: ' 31807.9909' ' Place: ' Trojan ' Colors: ' a541' ' b6' ' c1' ' d56'
' Time: ' 33311.0058' ' Place: ' Trojan ' Colors: ' a565' ' b4' ' c1' ' d62'
' Time: ' 33922.698' ' Place: ' Trojan ' Colors: ' a577' ' b4' ' c3' ' d51'
' Time: ' 35268.4207' ' Place: ' Trojan ' Colors: ' a598' ' b7' ' c1' ' d53'
' Time: ' 35880.1128' ' Place: ' Trojan ' Colors: ' a610' ' b4' ' c2' ' d20'
' Time: ' 36579.1895' ' Place: ' Trojan ' Colors: ' a624' ' b9' ' c2' ' d43'
' Time: ' 36981.1586' ' Place: ' Trojan ' Colors: ' a633' ' b1' ' c3' ' d37'
' Time: ' 37243.3124' ' Place: ' Trojan ' Colors: ' a637' ' b2' ' c1' ' d96'
' Time: ' 37767.6199' ' Place: ' Trojan ' Colors: ' a645' ' b1' ' c3' ' d60'
' Time: ' 38938.5735' ' Place: ' Trojan ' Colors: ' a668' ' b7' ' c3' ' d87'

The whole number of the send packages is : 686
The whole number of the received packages is:494
The whole number of the trojan packages is :46
The trojan rate is :0.093117
fx >>
OVR

```

Figure 18 Result of Trojan detection(3)

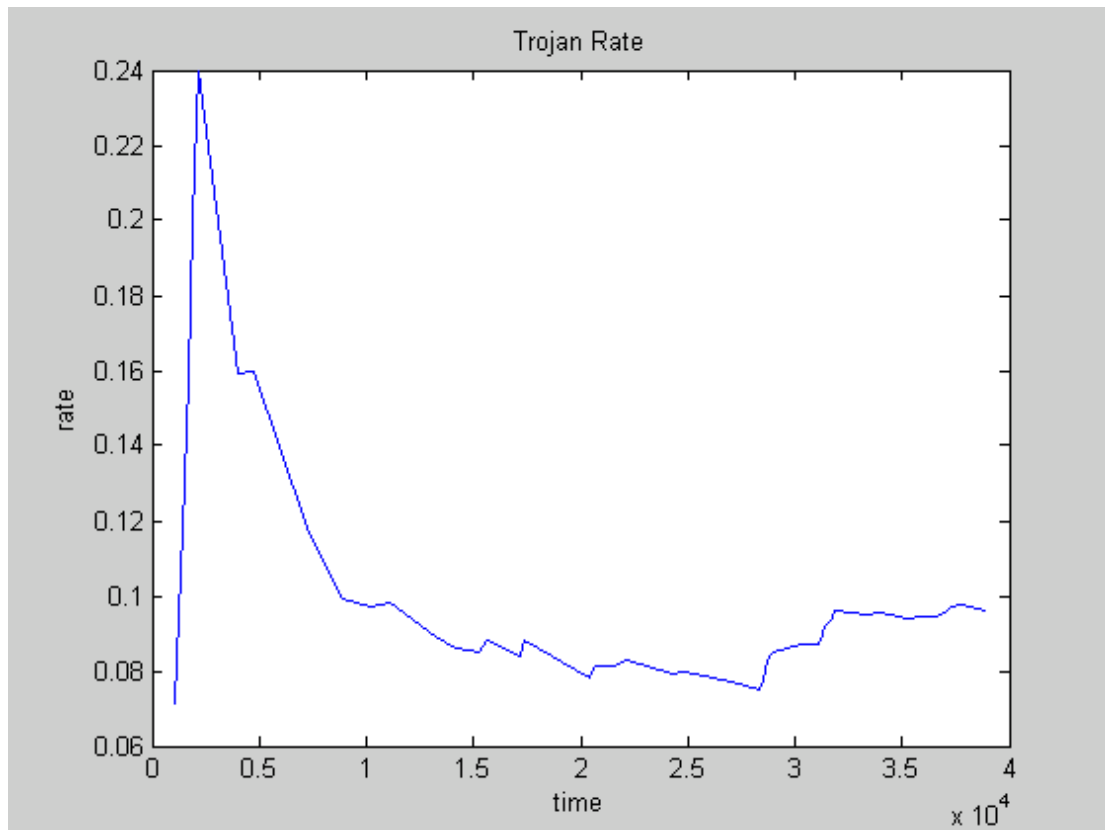


Figure 19 Curve diagram for the result of Trojan detection

Figure 16 to Figure 18 show part of the output data of Matlab in the process of simulation. Each line of the data represents the status of the Petri net after one more Trojan packet is detected. The data follows 'time' is the running time of the program. The data follows 'place' is the place containing the Trojan packet. The data follows 'place' is the color space of this place.

Figure 19 shows the change of the detected rate with time, in which the horizontal axis stands for time and the vertical axis stands for percentage. According to the result of the simulation we can know that after 40000 unit time 686 packets are sent, and 494 confirmation packets are received, including 46 Trojan packets. The overall detection rate is 0.093117.

5. conclusion

In this thesis, I used the architecture of the paper "Particle Filter for Depth Evaluation of Networking Intrusion Detection Using Coloured Petri Nets", the overall design of the system is the same as in that paper. I also used their algorithm to judge whether a packet is normal. However, since the tool to build Petri nets I used in this thesis is totally different from theirs, the Petri nets built in this thesis is also different from the precious paper. The programming process and simulation process are also different between this thesis and that paper.

This thesis analyzed some problems in the process of network intrusion detection, and simulated a network detection system based on particle filter. Two kinds of cases were considered: intrusion detection and Trojan detection, which have different schemes.

Against the intrusion detection, this thesis just considered the packets from the sender. That is, only the packets from the sender were detected. For the Trojan detection, this thesis just considered the confirmation packets from the receiver. In other words, it just detected the packets from the receiver.

The colored Petri net was used in this thesis for simulation. The intrusion detection system was simplified in the simulation, the reasons were as below:

- (1) It is hard to simulate the real network environment. Although I tried my best to make the simulated environment close to the real one, it is still simplified significantly.
- (2) It is also very difficult to apply particle filter to network intrusion detection, mainly due to the limitation of the particle filter itself and the characteristic of the network environment. The data flow is discontinuous, and it won't remain a fixed model and probability distribution. Moreover, the number of the particles is difficult to decide. The result will be better if the number of filters is more, but it will also be less efficient because of this.
- (3) It is also very difficult to get the truly intrusive network packets. In this simulation experiment, it was generated with a random probability. The process is also simplified.

For the reasons above, the simulation results in this thesis are not very ideal. From the data of the results, we can see that the detection rates are not so high and the overall result is not so satisfying. However, the analyses and experiments realized in this thesis can be a good guidance for designing a network detection system. It can analyze what happened during the process of the network intrusion, and can get the details of the problems during the network intrusion. It is useful for the further improvement of the IDS.

The GPenSIM toolbox itself also has some problems. One bug I found in the process of programming is the problem for it to deal with the color space. If the values of two colors in

the same color space are the same, the toolbox will combine them as one color. For example, an example in the manual can show this bug very clearly.

This example is to add two numbers together. When I input two same numbers, such as one plus one, then it will show 'input error' in the Matlab command window, because it only finds one color when calculating. I solved this problem from another point of view in this thesis. For example, when I define the color space of the network packets, it is defined like { 'a1' , ' b4' , ' c2' , ' d43' , ' ptN' }. I played a little trick here by adding one character before each color in order to solve this problem, because there will not be two same colors in one color space anymore. There is also a bug concerning the sequence of the colors in one color space. They are not sorted by their generated order but by the first letter of each color. Problems would be raised because of this. For example, the position of each color is very important in the network packets, because it is related to some specific meaning. Such as the third color, it means the IP address of the receiver. If the colors are reordered by GPenSIM, the meaning of each color would be confusing. I also solved this problem by adding a first letter, for example, { 'a1' , ' b2' , ' c3' , ' d4' , ' e5' }.

Another major problem is the memory leak error. That is, when the program runs for a long time, the leak error will occur sometimes. To solve this problem, I set the color space of some unimportant places to be null, so that these places do not save tokens. This can save memory space and avoid such error. Now we can run at least 40000 units time without lead error.

6. Future work

The future work will mainly be focused on the optimization of the model. How to simulate the network environment more reasonably? How to get more reasonable intrusion packets? How to make a better use of the particle filter in the network intrusion detection? The experiment result in this thesis is not so ideal because I have not solved these problems so well. I will focus on these problems in the future work.

Another problem which I need to move on to study is how to analyze more kinds of network intrusion. And when analyzing and detecting the network packets, more factors and connections should be considered. In this thesis we only discussed two kinds of attacks, actually the network intrusion conditions are much more complex than this. Another problem is how to judge whether a packet is abnormal, that is, how to distinguish the normal and abnormal packets. We just considered a packet as an individual in this thesis. However, the effect between the previous packet and the next one and the effect between multiple time windows should also be considered. In the future work I will analyze the impact and connections between each other to get a more reasonable intrusion detection scheme.

References

- [1] Jensen, K. (1997). Coloured Petri Nets Basic Concepts, Analysis Methods and Practical Use Springer-Verlag.
- [2] Dahl, O. M. and S. D. Wolthusen (2006). Modeling and execution of complex attack scenarios using interval timed colored Petri nets. the Fourth IEEE International Workshop on Information Assurance.
- [3] Chien-Chuan Lin, Ming-Shi Wang. Particle Filter for Depth Evaluation of Networking Intrusion Detection Using Coloured Petri Nets. InTech.2010.
- [4] Kristensen, L. M., J. B. Jorgensen, et al., Eds. Application of Coloured Petri Nets in detection rate. The attack packet rate is the ratio of attack packets against all sent packets. System Development. Lecture Notes in Computer Science, Springer-Verlag.2004.
- [5] Reggie Davidrajuh. GPenSIM: A New Petri Net Simulator. InTech. 2010.
- [6] Kotecha, J. H. and P. M. Djuric (2003). "Gaussian sum partide filtering." IEEE Trans. Signal Process 51(10): 2602-2612.
- [7] Lehn-Schioler, T., D. Erdogmus, et al. (2004). "Parzen partide filters." IEEE Int. Conf. Acoust., Speech, Signal Process 5: 781-784.
- [8] Servilla, R. H. G. L. A. M. M. The architecture of a network level intrusion detection System. Technical report, Department of Computer Science, University of New Mexico.1990.
- [9] Masahiro Tsunoyama, Hiroei Imai. An Application of GSPN for Modeling and Evaluating Local Area Computer Networks. InTech.2010.
- [10] Hugo Rodríguez, Rubén Carvajal, Beatriz Ontiveros, Ismael Soto. Using Petri Net for Modeling and Analysis of a Encryption Scheme for Wireless Sensor Networks. InTech.2010.
- [11] Charles Lakos, John Lamp, Chris Keenk, Brian Marriott. Modeling Network Protocols with Object Petri Nets. Workshop on Petri Nets Applied to Protocols.1995.
- [12] Congzhe Zhang, Mengchu Zhou. A Stochastic Petri Net Approach to Modeling and Analysis of Ad Hoc Network. Information Technology: Research and Education.2003.
- [13] Hurzeler, M., H. R. Kunsch. Monte Carlo approximations for general state space models. Computat. Graph. Statist.1998.
- [14] Savage, S., D. Wetherall, et al. Network Support for IP Traceback. IEEE/ACM

TRANSACTIONS ON NETWORKING 9(3).2001.

[15] David, R. & Alla, H. Discrete, Continuous, and Hybrid Petri Nets, Springer-Verlag, Berlin.
2005.

Source code

Instructions about how to run the source code :

1. GPenSIM must be installed first.

(1)Unzip the file "GPenSIM_v60_System_Files.zip" (attached in the CD) under a directory, say "d:\GPenSIM\".

(2)Set MATLAB Path Command. Start MATLAB, and go to the file menu in MATLAB, and select "set path" command, then select "Add folder". A new dialog box will appear, then browse through the directories and select the directory where you have unzipped the GPenSIM toolbox functions.

(3)Test Installation. Go to MATLAB command window and type 'gpensim'; if the following output is printed, then the installation is complete:

```
" GPenSIM version 6.0;   Lastupdate: May 2011
```

```
(C) Reggie.Davidrajuh@uis.no
```

```
http://www.davidrajuh.net/gpensim "
```

2. Unzip the "source code.rar" file (attached in the CD) under a directory, say "d:\sourcecode\".

3. Run Matlab, open the file "d:\sourcecode\pn_sim_part1\NPF_Intrusion.m" and run it. Wait a moment, and then you will get the detection simulation result.

4. Run Matlab, open the file "d:\sourcecode\pn_sim_part2\NPF_Intrusion.m" and run it. Wait a moment, and then you will get the Trojan simulation result.

%INTRUSION DETECTION PART:

```
%main function
addpath('./toolbox/'); %libraries for simulation
clear all;
clc;
global global_info; %global variable
global_info.STOP_AT=40000; %run time of simulation
global_info.package_count=0; %statistics of the number of packets
global_info.TN_count=0; %statistics of the number of normal packets
global_info.TA_count=0; %statistics of the number of abnormal packets
global_info.attack_count=0; %statistics of the number of intrusion packets
png=petrinetgraph({'Sender_def','Detect_def','Receive_def','conn_def'}); %combine all
the modules
%png=petrinetgraph('NPF_Intrusion_def');
dynamicpart.initial_markings={'start_pd',1,'start_Rate',1,'start',1}; %generate initial token
dynamicpart.firing_times={'Senders','unifrnd(0,100)'}; %define the time of transition
```



```

dynamicpart.initial_priority={'TranS1', 1,'TranS3',1}; %define priorityof the transition
results = gpensim(png, dynamicpart); %simulation
print_finalcolors(results); %output the simulation result
fprintf('The whole number of the send packages is: %d\n',global_info.package_count);
fprintf('The whole number of the attack packages is : %d\n',global_info.TA_count);
fprintf('The number of attack packages captured by system is: %d\n',
global_info.attack_count);
fprintf('The detect rate is : %f\n',global_info.attack_count/global_info.TA_count);

```

```

%define the transition Classify
function [fire,transition]=Classify_pre(transition) %define the transition function
tokID1 = select_token('classify1', 1); %get a token as the input from classify1
colors1 = get_color(tokID1); %get the color of the token
transition.override = 1; %the output token doesn't inherit the colors of the input token
transition.new_color=colors1; %get new color space from colors1
fire=1; %fire

```

```

%define the connection module
function [png]=conn_def() %module definition function
png.PN_name = 'Connections Profile'; %define the name of this module
png.set_of_Ps = []; %define the places in this module
png.set_of_Ts = []; %define the transitions in this module
png.set_of_As = {'A1','t_A',1,... %ddefine the arcs in this module
'B1','host1',1,...
'B2','host2',1,...
'B3','host3',1,...
'C1','TranS1',1,...
'C1','TranS1o',1,...
'C2','TranS2',1,...
'C3','TranS3',1,...
'C3','TranS3o',1,...
'A2','t_A2',1};

```

```

%define the transition cy_Attacks
function [fire,transition]=cy_Attacks_pre(transition) %define the transition function
global global_info; %declare global variable
tokID1 = select_token('cypk2', 1); %return the ID of the selected token
colors1 = get_color(tokID1); %get the color
pt = colors1{5}; %the fifth color
if strcmp(pt,'ptA') %compare two strings
    fire=1; %fire
    global_info.attack_count=global_info.attack_count+1; %statistics
else

```

```

        fire=0; %not to fire
    end

%define the transition cy_B1
function [fire,transition]=cy_B1_pre(transition) %define the transition function
tokID1 = select_token('classify12', 1); %get a token from classify12 as the input
tokID2 = select_token('Rate2', 1); %get a token from Rate2 as another input
colors1 = get_color(tokID1); %get the color of the token
colors2 = get_color(tokID2); %get the color of the token
num1 = str2num(colors1{3}(2:end)); %get the port number of input token
num2 = str2num(colors2{1}); %get the rate
transition.override = 1; %the output token doesn't inherit the colors of the input token
transition.new_color=colors1; %get new color space from colors1
if num1==1&&num2<=800 %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

```

```

%define the transition cy_B2
function [fire,transition]=cy_B2_pre(transition) %define the transition function
tokID1 = select_token('classify12', 1); %get a token from classify12 as the input
tokID2 = select_token('Rate2', 1); %get a token from Rate2 as another input
colors1 = get_color(tokID1); %get the color of the token
colors2 = get_color(tokID2); %get the color of the token
num1 = str2num(colors1{3}(2:end)); %get the port number of input token
num2 = str2num(colors2{1}); %get the rate
transition.override = 1; %the output token doesn't inherit the colors of the input token
transition.new_color=colors1; %get new color space from colors1
if num1==2&&num2<=800 %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

```

```

%define the transition cy_B3
function [fire,transition]=cy_B3_pre(transition) %define the transition function
tokID1 = select_token('classify12', 1); %get a token from classify12 as the input
tokID2 = select_token('Rate2', 1); %get a token from Rate2 as another input
colors1 = get_color(tokID1); %get the color of the token
colors2 = get_color(tokID2); %get the color of the token
num1 = str2num(colors1{3}(2:end)); %get the port number of input token
num2 = str2num(colors2{1}); %get the rate
transition.override = 1; %the output token doesn't inherit the colors of the input token

```

```

transition.new_color=colors1; %get new color space from colors1
if num1==3&&num2<=800 %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

%define the transition cy_B4
function [fire,transition]=cy_B4_pre(transition) %define the transition function
tokID1 = select_token('classify12', 1); %get a token from classify12 as the input
tokID2 = select_token('Rate2', 1); %get a token from Rate2 as another input
colors1 = get_color(tokID1); %get the color of the token
colors2 = get_color(tokID2); %get the color of the token
num2 = str2num(colors2{1}); %get the rate
transition.override = 1; %the output token doesn't inherit the colors of the input token
transition.new_color=colors1; %get new color space from colors1
if num2>800 %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

%define the transition cy_Nattack
function [fire,transition]=cy_Nattack_pre(transition) %define the transition function
tokID1 = select_token('cypk2', 1); %get a token from cypk2 as the input
colors1 = get_color(tokID1); %get the color of the token
pt = colors1{5}; %judge the packet to be normal or abnormal
% transition.override = 1;
if strcmp(pt,'ptN') %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

%define the transition cy_Rate
function [fire,transition]=cy_Rate_pre(transition) %define the transition function
transition.override = 1; %the output token doesn't inherit the colors of the input token
rd=ceil(unifrnd(0,1000)); %get a random integer between 0 and 1000
transition.new_color=num2str(rd); %use the integer as the color of the token
fire=1; %fire

%define detection module
function [png]=Detect_def() %define module name
png.PN_name = 'Dectect Module'; %define the name of this module

```

```

png.set_of_Ps={ 'A11','A12','A13',... %define the places in this module
'A2','start_pd','Partide',...
'NPF_Pd1','NPF_Pd2','NPF_Pd3',...
'NPF2_ot','NPF3_ot','classify21',...
'classify22','classify23','start_Rate',...
'Rate','Rate2','classify1',...
'classify12','attacks','Nattacks',...
'cypk1','cypk2','cypk3',...
'B1','B2','B3',...
'B4','B1o','B2o',...
'B3o','C1o','C3o'};
png.set_of_Ts={ 't_A','Gen_First_pd','NPF',... %define the transitions in this module
'NPF3','NPF1','NPF2',...
'NPF2o','NPF3o','Normal1',...
'Normal2','Normal3','Normal1o',...
'Normal2o','Normal3o','Gen_First_Rate',...
'cy_Rate','cy_Attacks','cy_Nattack',...
'cy_B1','cy_B2','cy_B3',...
'cy_B4','Classify','ClassifyT',...
'TranS1','TranS1o','TranS2',...
'TranS3','TranS3o'};
png.set_of_As={ 't_A','A11',1,... %ddefine the arcs in this module
't_A','A12',1,...
't_A','A13',1,...
'start_pd','Gen_First_pd',1,...
'Gen_First_pd','Partide',1,...
'Partide','NPF',1,...
'NPF','NPF_Pd1',1,...
'NPF','NPF_Pd2',1,...
'NPF','NPF_Pd3',1,...
'NPF_Pd1','NPF1',1,...
'A11','NPF1',1,...
'NPF1','Partide',1,...
'NPF_Pd2','NPF2',1,...
'A12','NPF2',1,...
'NPF2','classify1',1,...
'NPF2','classify12',1,...
'NPF_Pd2','NPF2o',1,...
'A12','NPF2o',1,...
'NPF2o','NPF2_ot',1,...
'NPF_Pd3','NPF3',1,...
'A13','NPF3',1,...
'NPF3','classify21',1,...
'NPF3','classify22',1,...

```

'NPF3','classify23',1,...
 'NPF_Pd3','NPF3o',1,...
 'A13','NPF3o',1,...
 'NPF3o','NPF3_ot',1,...
 'classify21','Normal1',1,...
 'Normal1','B1',1,...
 'classify21','Normal1o',1,...
 'Normal1o','B1o',1,...
 'classify22','Normal2',1,...
 'Normal2','B2',1,...
 'classify22','Normal2o',1,...
 'Normal2o','B2o',1,...
 'classify23','Normal3',1,...
 'Normal3','B3',1,...
 'classify23','Normal3o',1,...
 'Normal3o','B3o',1,...
 'start_Rate','Gen_First_Rate',1,...
 'Gen_First_Rate','Rate',1,...
 'Gen_First_Rate','Rate2',1,...
 'Rate','Classify',1,...
 'classify1','Classify',1,...
 'Classify','cypk1',1,...
 'Classify','cypk2',1,...
 'cypk1','cy_Rate',1,...
 'cy_Rate','Rate',1,...
 'cy_Rate','Rate2',1,...
 'cypk2','cy_Attacks',1,...
 'cy_Attacks','attacks',1,...
 'cypk2','cy_Nattack',1,...
 'cy_Nattack','Nattacks',1,...
 'classify12','cy_B1',1,...
 'Rate2','cy_B1',1,...
 'cy_B1','B1',1,...
 'classify12','cy_B2',1,...
 'Rate2','cy_B2',1,...
 'cy_B2','B2',1,...
 'classify12','cy_B3',1,...
 'Rate2','cy_B3',1,...
 'cy_B3','B3',1,...
 'classify12','cy_B4',1,...
 'Rate2','cy_B4',1,...
 'cy_B4','B4',1,...
 'TranS1','A2',1,...
 'TranS1o','C1o',1,...

```
'TranS2','A2',1,...
'TranS3','A2',1,...
'TranS3o','C3o',1];
```

```
%define the transition Gen_First_pd
function [fire,transition]=Gen_First_pd_pre(transition) %define the transition function
rd=ceil(unifrnd(0,1000)); %generate a random integer between 0 and 1000
transition.new_color{1}=['a' num2str(rd)]; %get the first color of the input token
rd=ceil(unifrnd(0,10)); %get a random integer between 0 and 10
transition.new_color{2}=['b' num2str(rd)]; %get the second color of the input token
rd=ceil(unifrnd(0,3)); %generate a random integer between 0 and 3
transition.new_color{3}=['c' num2str(rd)]; %get the third color of the input token
rd=ceil(unifrnd(0,100)); %generate a random integer between 0 and 100
transition.new_color{4}=['d' num2str(rd)]; %get the fourth color of the input token
rd=unifrnd(1,1000); %generate a random integer between 1 and 1000
if rd>100
    transition.new_color{5}=['pt' 'N']; %get the fifth color of the input token
else
    transition.new_color{5}=['pt' 'A']; %get the fifth color of the input token
end
fire=1; %fire
```

```
%define the transition Gen_First_Rate
function [fire,transition]=Gen_First_Rate_pre(transition) %define the transition function
rd=ceil(unifrnd(0,1000)); %generate a random integer between 0 and 1000
transition.new_color=num2str(rd); %get the color of the output token
fire=1; %fire
```

```
%define the transition Gen_First_Sender
function [fire,transition]=Gen_First_Sender_pre(transition) %define the transition function
global global_info; %define global variable
global_info.package_count=global_info.package_count+1; %statistics of the number of
packets
transition.new_color{1}=['a' num2str(global_info.package_count)]; %the color of
generated token
rd2=ceil(unifrnd(0,10)); %generate a random integer
transition.new_color{2}=['b' num2str(rd2)]; %the color of generated token
rd3=ceil(unifrnd(0,3)); %generate a random integer
transition.new_color{3}=['c' num2str(rd3)]; %the color of generated token
rd4=ceil(unifrnd(0,100)); %generate a random integer
transition.new_color{4}=['d' num2str(rd4)]; %the color of generated token
rd=unifrnd(1,1000); %generate a random integer
if rd>100 %the color of generated token
    transition.new_color{5}=['pt' 'N'];
```

```

else
    transition.new_color{5}=['pt'A'];
end
fire=1; %fire

%define the transition Gen_Sender
function [fire,transition]=Gen_Sender_pre(transition) %define the transition function
global global_info; %define global variable
transition.override = 1;
global_info.package_count=global_info.package_count+1; %statistics of the number of
packets
rd1=global_info.package_count;
transition.new_color{1}=['a' num2str(global_info.package_count)]; %the color of
generated token
rd2=ceil(unifrnd(0,10)); %generate a random integer
transition.new_color{2}=['b' num2str(rd2)]; %the color of generated token
rd3=ceil(unifrnd(0,3)); %generate a random integer
transition.new_color{3}=['c' num2str(rd3)]; %the color of generated token
rd4=ceil(unifrnd(0,100)); %generate a random integer
transition.new_color{4}=['d' num2str(rd4)]; %the color of generated token
rd=unifrnd(1,1000); %generate a random integer
if rd>100
    transition.new_color{5}=['pt'N'];
else
    transition.new_color{5}=['pt'A'];
end
fire=1;

%define the transition Normal1
function [fire,transition]=Normal1_pre(transition) %define the transition function
tokID1 = select_token('classify21', 1); %get a token from classify21 as the input
colors1 = get_color(tokID1); %get the color of the token
num1 = str2num(colors1{3}(2:end)); %get the port number of input token
if num1==1 %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

%define the transition Normal1o
function [fire,transition]=Normal1o_pre(transition) %define the transition function
tokID1 = select_token('classify21', 1); %get a token from classify21 as the input
colors1 = get_color(tokID1); %get the color of the token
num1 = str2num(colors1{3}(2:end)); %get the port number of input token

```

```

    % transition.override = 1;
if num1~=1 %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

```

```

%define the transition Normal2
function [fire,transition]=Normal2_pre(transition) %define the transition function
tokID1 = select_token('classify22', 1); %get a token from classify22 as the input
colors1 = get_color(tokID1); %get the color of the token
num1 = str2num(colors1{3}(2:end)); %get the port number of input token
if num1==2 %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

```

```

%define the transition Normal2o
function [fire,transition]=Normal2o_pre(transition) %define the transition function
tokID1 = select_token('classify22', 1); %get a token from classify22 as the input
colors1 = get_color(tokID1); %get the color of the token
num1 = str2num(colors1{3}(2:end)); %get the port number of input token
    % transition.override = 1;
if num1~=2 %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

```

```

%define the transition Normal3
function [fire,transition]=Normal3_pre(transition) %define the transition function
tokID1 = select_token('classify23', 1); %get a token from classify23 as the input
colors1 = get_color(tokID1); %get the color of the token
num1 = str2num(colors1{3}(2:end)); %get the port number of input token
if num1==3 %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

```

```

%define the transition Normal3o
function [fire,transition]=Normal3o_pre(transition) %define the transition function
tokID1 = select_token('classify23', 1); %get a token from classify23 as the input

```



```

colors1 = get_color(tokID1); %get the color of the token
num1 = str2num(colors1{3}(2:end)); %get the port number of input token
% transition.override = 1;
if num1~=3 %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

%define the transition NPF1
function [fire,transition]=NPF1_pre(transition) %define the transition function
rd=ceil(unifrnd(0,1000)); %generate a random integer
transition.new_color{1}=['a' num2str(rd)]; %the color of generated token
rd=ceil(unifrnd(0,10)); %generate a random integer
transition.new_color{2}=['b' num2str(rd)]; %the color of generated token
rd=ceil(unifrnd(0,3)); %generate a random integer
transition.new_color{3}=['c' num2str(rd)]; %the color of generated token
rd=ceil(unifrnd(0,100)); %generate a random integer
transition.new_color{4}=['d' num2str(rd)]; %the color of generated token
rd=unifrnd(1,1000); %generate a random integer
if rd>100
    transition.new_color{5}=['pt' 'N']; %the color of generated token
else
    transition.new_color{5}=['pt' 'A']; %the color of generated token
end
transition.override = 1;
fire=1; %fire

%define the transition NPF2
function [fire,transition]=NPF2_pre(transition) %define the transition function
tokID1 = select_token('A12', 1); %get a token from A12 as the input
tokID2 = select_token('NPF_Pd2', 1); %get a token from NPF_Pd2 as another input
colors1 = get_color(tokID1); %get the color of the token
colors2 = get_color(tokID2); %get the color of the token
num1 = str2num(colors1{4}(2:end)); %get the port number of the token
num2 = str2num(colors2{4}(2:end)); %get the port number of the particle
transition.override = 1; %the output token doesn't inherit the colors of the input token
transition.new_color=colors1; %get new color space from colors1
if abs(num1-num2)/100<=0.1 %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end
end

```

```

%define the transition NPF2o
function [fire,transition]=NPF2o_pre(transition) %define the transition function
tokID1 = select_token('A12', 1); %get a token from A12 as the input
tokID2 = select_token('NPF_Pd2', 1); %get a token from NPF_Pd2 as the input
colors1 = get_color(tokID1); %get the color of the token
colors2 = get_color(tokID2); %get the color of the token
num1 = str2num(colors1{4}(2:end)); %get the port number of input token
num2 = str2num(colors2{4}(2:end)); %get the port number of the particle
transition.override = 1; %the output token doesn't inherit the colors of the input token
transition.new_color=colors1; %get new color space from colors1
if abs(num1-num2)/100>0.1 %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

```

```

%define the transition NPF3
function [fire,transition]=NPF3_pre(transition) %define the transition function
tokID1 = select_token('A13', 1); %get a token from A13 as the input
tokID2 = select_token('NPF_Pd3', 1); %get a token from NPF_Pd3 as another input
colors1 = get_color(tokID1); %get the color of the token
colors2 = get_color(tokID2); %get the color of the token
num1 = str2num(colors1{4}(2:end)); %get the port number of input token
num2 = str2num(colors2{4}(2:end)); %get the port number of the particle
transition.override = 1; %the output token doesn't inherit the colors of the input token
transition.new_color=colors1; %get new color space from colors1
pt=colors1{5};
if abs(num1-num2)/100>0.1&&strcmp(pt,'ptN') %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

```

```

%define the transition NPF3o
function [fire,transition]=NPF3o_pre(transition) %define the transition function
tokID1 = select_token('A13', 1); %get a token from A13 as the input
tokID2 = select_token('NPF_Pd3', 1); %get a token from NPF_Pd3 as another input
colors1 = get_color(tokID1); %get the color of the token
colors2 = get_color(tokID2); %get the color of the token
num1 = str2num(colors1{4}(2:end)); %get the port number of input token
num2 = str2num(colors2{4}(2:end)); %get the port number of the particle
pt=colors1{5};
transition.override = 1; %the output token doesn't inherit the colors of the input token
transition.new_color=colors1; %get new color space from colors1

```

```

if ~(abs(num1-num2)/100>0.1&&strcmp(pt,'ptN')) %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

%define the transition Rec_Ak2
function [fire,transition]=Rec_Ak2_pre(transition) %define the transition function
tokID1 = select_token('A22', 1); %get a token from A22 as the input
colors1 = get_color(tokID1); %get the color of the token
pt = colors1{5};

if strcmp(pt,'ptA') %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

%define the transition Rec_Ak2o
function [fire,transition]=Rec_Ak2o_pre(transition) %define the transition function
tokID1 = select_token('A22', 1); %get a token from A22 as the input
colors1 = get_color(tokID1); %get the color of the token
pt = colors1{5};
% transition.override = 1;
if strcmp(pt,'ptN') %the condition to fire
    fire=1;
else
    fire=0; %not to fire
end

%define the receiver module
function [png]=Receive_def() %module definition function
png.PN_name = 'Receive Profile'; %define the name of this module
png.set_of_Ps = {'C1','C2','C3'}; %define the places in this module
png.set_of_Ts = {'host1','host2','host3'}; %define the transitions in this module
png.set_of_As = {'host1','C1',1,... %ddefine the arcs in this module
    'host2','C2',1,...
    'host3','C3',1};

%define the sender module
function [png]=Sender_def() %module definition function
png.PN_name = 'Sender Module'; %define the name of this module
png.set_of_Ps = {'start','Sender1','Sender2',... %define the places in this module
    'Sender3','Sender','SenderNo',...

```

```

'A1','TA','TN',...
'A21','A22','Send_Rec',...
'TA2','TA2o'};
png.set_of_Ts = {'Gen_First_Sender','Senders','Gen_Sender',... %define the transitions in
this module
't_TA','t_SNA1','t_TN','t_A2',...
'Rec_Ak1','Rec_Ak2','Rec_Ak2o'};
png.set_of_As={'start','Gen_First_Sender',1,... %ddefine the arcs in this module
'Gen_First_Sender','Sender',1,...
'Sender','Senders',1,...
'Senders','Sender1',1,...
'Senders','Sender2',1,...
'Senders','Sender3',1,...
'Sender1','t_SNA1',1,...
't_SNA1','A1',1,...
't_SNA1','SenderNo',1,...
'Sender2','t_TA',1,...
't_TA','TA',1,...
'Sender2','t_TN',1,...
't_TN','TN',1,...
'Sender3','Gen_Sender',1,...
'Gen_Sender','Sender',1,...
't_A2','A21',1,...
't_A2','A22',1,...
'A21','Rec_Ak1',1,...
'Rec_Ak1','Send_Rec',1,...
'A22','Rec_Ak2',1,...
'Rec_Ak2','TA2',1,...
'A22','Rec_Ak2o',1,...
'Rec_Ak2o','TA2o',1};

%define the transition t_SNA1
function [fire,transition]=t_SNA1_pre(transition)
fire=1;

%define the transition t_TA
function [fire,transition]=t_TA_pre(transition) %define the transition function
global global_info; %define global variable
% transition.selected_tokens = select_token_with_colors('Sender2',1,'ptA'); %get a token
from Sender2 as the input
tokID1 = select_token_with_colors('Sender2',1,'ptA'); %get the select token whose color is
ptA from Sender2
transition.selected_tokens=tokID1;
colors1 = get_color(tokID1); %get the color of the token

```

```

transition.override = 1; %the output token doesn't inherit the colors of the input token
transition.new_color=colors1; %get new color space from colors1
if ~isempty(transition.selected_tokens) %the condition to fire
fire = 1;
global_info.TA_count=global_info.TA_count+1; %statistics of number of abnormal packets
else
fire = 0;
end;

```

```

%define the transition t_TN
function [fire,transition]=t_TN_pre(transition) %define the transition function
global global_info;
tokID1 = select_token_with_colors('Sender2',1,'ptN'); %select the tokens whose color is
ptN from Sender2
transition.selected_tokens=tokID1;
colors1 = get_color(tokID1); %get the color of the token
transition.override = 1; %the output token doesn't inherit the colors of the input token
transition.new_color=colors1; %get new color space from colors1
if ~isempty(transition.selected_tokens),
fire = 1;
global_info.TN_count=global_info.TN_count+1; %statistics of number of normal packets
else
fire = 0;
end;

```

```

%define the transition TranS1
function [fire,transition]=TranS1_pre(transition) %define the transition function
tokID1 = select_token('C1', 1); %get a token from C1 as the input
colors1 = get_color(tokID1); %get the color of the token
num1 = str2num(colors1{1});
rd=ceil(unifrnd(0,1000));
if num1>rd
fire=1;
else
fire=0;
end

```

```

%define the transition TranS1o
function [fire,transition]=TranS1o_pre(transition)
fire=1;

```

```

%define the transition TranS3
function [fire,transition]=TranS3_pre(transition) %define the transition function
tokID1 = select_token('C3', 1); %get a token from C3 as the input

```

```

colors1 = get_color(tokID1); %get the color of the token
num1 = str2num(colors1{1});
rd=ceil(unifrnd(0,1000));
if num1<rd
    fire=1;
else
    fire=0;
end

%define the transition Trans3o
function [fire,transition]=Trans3o_pre(transition)
fire=1;

%TROJAN DETECTION PART:

%main function
clear all;
clc;
addpath('./toolbox/'); %libraries for simulation
global global_info; %global variable
global_info.STOP_AT=10000; %run time of simulation
global_info.package_count=0; %statistics of the number of packets
global_info.Trojan_count=0; %statistics of number of Trojan packets
global_info.Rec_count=0; %statistics of the number of received confirmation packets
% png=petrinetgraph('NPF_Intrusion_def');
png=petrinetgraph({'Sender_def','Detect_def','Receive_def','conn_def'}); %combine the
modules into a Petri net
dynamicpart.initial_markings = {'start',1,'Start_Rate',1,'Start_Partide',1}; %generate the
initial token
% dynamicpart.firing_times = {'Senders', '100'};
dynamicpart.firing_times = {'Senders', 'unifrnd(0,100)'}; %define the time for transition
results = gpensim(png, dynamicpart); %simulation
print_finalcolors(results); %output simulation result
fprintf('The whole number of the send packages is: %d\n',global_info.package_count);
fprintf('The whole number of the received packages is: %d\n', global_info.Rec_count);
fprintf('The whole number of the trojan packages is: %d\n',global_info.Trojan_count);
fprintf('The trojan rate is: %f\n',global_info.Trojan_count/global_info.Rec_count);

%define the connection module
function [png]=conn_def() %module definition function
png.PN_name = 'Connections Profile'; %define the name of this module

```

```

png.set_of_Ps=[]; %define the places in this module
png.set_of_Ts=[]; %define the transitions in this module
png.set_of_As={'A1','t_A1',1,... %ddefine the arcs in this module
    'B1','host1',1,...
    'B2','host2',1,...
    'B3','host3',1,...
    'C1','TranS1',1,...
    'C2','TranS2',1,...
    'C3','TranS3',1,...
    'A2','Rec_Ak',1};

%define detection module
function [png]=Detect_def() %define module name
png.PN_name = 'Dectect Module'; %define the name of this module
png.set_of_Ps={'A11','A12','Start_Rate',... %define the places in this module
    'Rate','Rate1','Rate2',...
    'B1','B2','B3','B4',...
    'Collect','Collect1','Collect2',...
    'Start_Partide','Partide','Pd1',...
    'Pd2','Trojan','A2',...
};
png.set_of_Ts={'t_A1','Gen_First_Rate','Gen_Rate',... %define the transitions in this
module
    't_Rate','t_B1','t_B2','t_B3','t_B4',...
    'TranS3','TranS1','TranS2','t_clt',...
    'NPF1','Gen_First_pd','t_pd',...
    't_Trojan','t_A2'}};
png.set_of_As={'t_A1','A11',1,... %ddefine the arcs in this module
    't_A1','A12',1,...
    'Start_Rate','Gen_First_Rate',1,...
    'Gen_First_Rate','Rate',1,...
    'Rate','t_Rate',1,...
    't_Rate','Rate1',1,...
    't_Rate','Rate2',1,...
    'Rate1','Gen_Rate',1,...
    'A11','Gen_Rate',1,...
    'Gen_Rate','Rate',1,...
    'Rate2','t_B1',1,...
    'A12','t_B1',1,...
    't_B1','B1',1,...
    'Rate2','t_B2',1,...
    'A12','t_B2',1,...
    't_B2','B2',1,...
    'Rate2','t_B3',1,...

```

```

'A12','t_B3',1,...
't_B3','B3',1,...
'Rate2','t_B4',1,...
'A12','t_B4',1,...
't_B4','B4',1,...
'TranS1','Collect',1,...
'TranS2','Collect',1,...
'TranS3','Collect',1,...
'Collect','t_clt',1,...
't_clt','Collect1',1,...
't_clt','Collect2',1,...
'Start_Partide','Gen_First_pd',1,...
'Gen_First_pd','Partide',1,...
'Partide','t_pd',1,...
't_pd','Pd1',1,...
't_pd','Pd2',1,...
'Pd1','NPF1',1,...
'Collect1','NPF1',1,...
'NPF1','Partide',1,...
'Pd2','t_Trojan',1,...
'Collect2','t_Trojan',1,...
't_Trojan','Trojan',1,...
'Pd2','t_A2',1,...
'Collect2','t_A2',1,...
't_A2','A2',1,...
};

```

```

%define the transition Gen_First_pd
function [fire,transition]=Gen_First_pd_pre(transition) %define the transition function
rd=ceil(unifrnd(0,1000)); %generate a random integer between 0 and 1000
transition.new_color{1}=['a' num2str(rd)]; %get the first color of the input token
rd=ceil(unifrnd(0,10));
transition.new_color{2}=['b' num2str(rd)]; %get the second color of the input token
rd=ceil(unifrnd(0,3));
transition.new_color{3}=['c' num2str(rd)]; %get the third color of the input token
rd=ceil(unifrnd(0,100));
transition.new_color{4}=['d' num2str(rd)]; %get the fourth color of the input token
fire=1; %fire

```

```

%define the transition Gen_First_Rate
function [fire,transition]=Gen_First_Rate_pre(transition) %define the transition function
rd=ceil(unifrnd(0,1000)); %generate a random integer between 0 and 1000
transition.new_color=num2str(rd); %get the color of the output token
fire=1; %fire

```



```

%define the transition Gen_First_Sender
function [fire,transition]=Gen_First_Sender_pre(transition) %define the transition function
global global_info; %define global variable
global_info.package_count=global_info.package_count+1; %statistics of the number of
packets
transition.new_color{1}=['a' num2str(global_info.package_count)]; %the color of
generated token
rd2=ceil(unifrnd(0,10)); %generate a random integer
transition.new_color{2}=['b' num2str(rd2)]; %the color of generated token
rd3=ceil(unifrnd(0,3)); %generate a random integer
transition.new_color{3}=['c' num2str(rd3)]; %the color of generated token
rd4=ceil(unifrnd(0,100)); %generate a random integer
transition.new_color{4}=['d' num2str(rd4)]; %the color of generated token
fire=1; %fire

```

```

%define the transition Gen_Rate
function [fire,transition]=Gen_Rate_pre(transition) %define the transition function
transition.override = 1;
rd=ceil(unifrnd(0,1000));
transition.new_color=num2str(rd); %the color of generated token
fire=1; %fire

```

```

function [fire,transition]=Gen_Sender_pre(transition)
global global_info;
transition.override = 1;
global_info.package_count=global_info.package_count+1;
rd1=global_info.package_count;
transition.new_color{1}=num2str(global_info.package_count);
while(1)
    rd2=ceil(unifrnd(0,10));
    if(rd2~=rd1)
        transition.new_color{2}=num2str(rd2);
        break;
    end
end
rd3=ceil(unifrnd(0,3));
while(1)
    rd2=ceil(unifrnd(0,10));
    if(rd2~=rd1&&)
        transition.new_color{3}=num2str(rd3);
        break;
    end
end
end

```

```

rd=ceil(unifrnd(0,100));
transition.new_color{4}=num2str(rd4);
rd=unifrnd(1,1000);
if rd>100
    transition.new_color{5}='N';
else
    transition.new_color{5}='A';
end
fire=1;

%define the transition Gen_Sender
function [fire,transition]=Gen_Sender_pre(transition) %define the transition function
global global_info; %define global variable
transition.override = 1;
global_info.package_count=global_info.package_count+1; %statistics of the number of
packets
rd1=global_info.package_count;
transition.new_color{1}=['a' num2str(global_info.package_count)]; %statistics of the
number of packets
rd2=ceil(unifrnd(0,10));
transition.new_color{2}=['b' num2str(rd2)]; %statistics of the number of packets
rd3=ceil(unifrnd(0,3));
transition.new_color{3}=['c' num2str(rd3)]; %statistics of the number of packets
rd4=ceil(unifrnd(0,100));
transition.new_color{4}=['d' num2str(rd4)]; %statistics of the number of packets
fire=1;

%define the transition NPF1
function [fire,transition]=NPF1_pre(transition) %define the transition function
rd=ceil(unifrnd(0,1000)); %generate a random integer
transition.new_color{1}=['a' num2str(rd)]; %the color of generated token
rd=ceil(unifrnd(0,10)); %generate a random integer
transition.new_color{2}=['b' num2str(rd)]; %the color of generated token
rd=ceil(unifrnd(0,3)); %generate a random integer
transition.new_color{3}=['c' num2str(rd)]; %the color of generated token
rd=ceil(unifrnd(0,100)); %generate a random integer
transition.new_color{4}=['d' num2str(rd)]; %the color of generated token
transition.override = 1;
fire=1; %fire

%define the transition Rec_Ak
function [fire,transition]=Rec_Ak_pre(transition) %define the transition function
global global_info;

```

```

global_info.Rec_count=global_info.Rec_count+1; %statistics of the number of packets
fire=1;

```

```

%define the receiver module

```

```

function [png]=Receive_def() %module definition function
png.PN_name = 'Receive Profile'; %define the name of this module
png.set_of_Ps = {'C1','C2','C3'}; %define the places in this module
png.set_of_Ts = {'host1','host2','host3'}; %define the transitions in this module
png.set_of_As = {'host1','C1',1,... %ddefine the arcs in this module
    'host2','C2',1,...
    'host3','C3',1};

```

```

%define the sender module

```

```

function [png]=Sender_def() %module definition function
png.PN_name = 'Sender Module'; %define the name of this module
png.set_of_Ps = {'start','Sender','Sender1',... %define the places in this module
    'Sender2','SenderNo','A1','Send_Rec'};
png.set_of_Ts = {'Gen_First_Sender','Senders',... %define the transitions in this module
    'Gen_Sender','t_SNA1','Rec_Ak'};
png.set_of_As = {'start','Gen_First_Sender',1,... %ddefine the arcs in this module
    'Gen_First_Sender','Sender',1,...
    'Sender','Senders',1,...
    'Senders','Sender1',1,...
    'Senders','Sender2',1,...
    'Sender1','t_SNA1',1,...
    't_SNA1','A1',1,...
    't_SNA1','SenderNo',1,...
    'Sender2','Gen_Sender',1,...
    'Gen_Sender','Sender',1,...
    'Rec_Ak','Send_Rec',1,...
};

```

```

%define the transition t_A2

```

```

function [fire,transition]=t_A2_pre(transition) %define the transition function
tokID1 = select_token('Collect2', 1); %get a token from Collect2 as the input
tokID2 = select_token('Pd2', 1); %get a token from Pd2 as the input
colors1 = get_color(tokID1); %get the color of the token
colors2 = get_color(tokID2); %get the color of the token
num1 = str2num(colors1{4}(2:end));
num2 = str2num(colors2{4}(2:end));
transition.override = 1;
transition.new_color=colors1; %get new color space from colors1
if abs(num1-num2)/100>=0.05
    fire=1;

```

```

else
    fire=0;
end

%define the transition t_B1
function [fire,transition]=t_B1_pre(transition) %define the transition function
tokID1 = select_token('A12', 1); %get a token from A12 as the input
tokID2 = select_token('Rate2', 1); %get a token from Rate2 as the input
colors1 = get_color(tokID1); %get the color of the token
colors2 = get_color(tokID2); %get the color of the token
num1 = str2num(colors1{3}(2:end));
num2 = str2num(colors2{1});
transition.override = 1;
transition.new_color=colors1; %get new color space from colors1
if num1==1&&num2<=800
    fire=1;
else
    fire=0;
end

%define the transition t_B2
function [fire,transition]=t_B2_pre(transition) %define the transition function
tokID1 = select_token('A12', 1); %get the select token whose color is ptA from A12
tokID2 = select_token('Rate2', 1); %get the select token whose color is ptA from Rate2
colors1 = get_color(tokID1); %get the color of the token
colors2 = get_color(tokID2); %get the color of the token
num1 = str2num(colors1{3}(2:end));
num2 = str2num(colors2{1});
transition.override = 1;
transition.new_color=colors1; %get new color space from colors1
if num1==2&&num2<=800
    fire=1;
else
    fire=0;
end

%define the transition t_B3
function [fire,transition]=t_B3_pre(transition) %define the transition function
tokID1 = select_token('A12', 1); %get a token from A12 as the input
tokID2 = select_token('Rate2', 1); %get a token from Rate2 as the input
colors1 = get_color(tokID1);
colors2 = get_color(tokID2);
num1 = str2num(colors1{3}(2:end));
num2 = str2num(colors2{1});

```

```

transition.override = 1;
transition.new_color=colors1; %get new color space from colors1
if num1==3&&num2<800
    fire=1;
else
    fire=0;
end

%define the transition t_B4
function [fire,transition]=t_B4_pre(transition) %define the transition function
tokID2 = select_token('Rate2', 1); %get a token from Rate2 as the input
colors2 = get_color(tokID2); %get the color of the token
num2 = str2num(colors2{1});
transition.override = 1;
if num2>800
    fire=1;
else
    fire=0;
end

%define the transition t_SNA1
function [fire,transition]=t_SNA1_pre(transition)
fire=1;

%define the transition t_Trojan
function [fire,transition]=t_Trojan_pre(transition) %define the transition function
global global_info; %define global variable
tokID1 = select_token('Collect2', 1); %get a token Collect2 as the input
tokID2 = select_token('Pd2', 1); %get a token from Pd2 as the input
colors1 = get_color(tokID1); %get the color of the token
colors2 = get_color(tokID2); %get the color of the token
num1 = str2num(colors1{4}(2:end)); %get the port number of the token
num2 = str2num(colors2{4}(2:end)); %get the port number of the particle
transition.override = 1;
transition.new_color=colors1; %get the color of generated token
if abs(num1-num2)/100<0.05
    fire=1;
    global_info.Trojan_count=global_info.Trojan_count+1; %statistics of number of
Trojan packets
else
    fire=0;
end

```