Carleton University

95.495 Honors Project

Implement a simulator, in Java, for evaluating AntNet routing protocol

Chunyan Ma # 277210

Supervised by Professor Tony White
School of Computer Science

December 31, 2002

Abstract

This project is to implement a simulator, in Java, for evaluating the AntNet routing protocol.

CuRS, the simulator implemented in this project is a discrete-event simulation testbed for developing routing algorithm for wide-area computer networks. It was developed as a modified and extended Java version of an existing simulator, MaRS (Maryland Routing Simulator)-Version 1.0. CuRS is more portable for different platforms, and it extended to include AntNet algorithm. It allow user to define a network configuration consisting of physical network, routing algorithm and workload. The user can control its simulation, log the values of parameters, and save, load and modify network configurations. It also provides performance to assist comparing routing protocols. But the Graphical user Interface is not yet implemented in this project. This is a part of work needed to be extended in the next phase.

AntNet is an adaptive, distributed, mobile-agents-based algorithm for communications networks. It was inspired by recent work on the ant colony metaphor for solving optimization problems. I compare AntNet algorithm with Shortest Path First routing algorithm (SPF) based on CuRS. Actually, we should compare more routing algorithms to evaluate AntNet algorithm. This is the work needed to done further.

Acknowledgments

I gratefully acknowledge the use of the structuring concepts of MaRS, the user and programmer manuals of MaRS which helped me in writing user manual for CuRS. And I also acknowledge the use of the AntNet routing algorithm from Gianni Di Caro and Marco Dorigo.
I thank my supervisor- Professor Tory White very much for help whenever I need. I also thank Dario Accornero for answering the question about MaRS.

Table of Contents

Figure and Table List

1. Introduction

In worldwide demand about computer network such like Internet. Routing protocols plays the core role in the whole network control system. Which are responsible for choosing the best way for delivering data packets in conjunction with performance in term of delay, throughput, etc. The performance of a route changes with the difference of network traffic condition (for example, the amount of the data flow through the links), so the routing algorithm should consider those facts and adapt the information store in the route.
The routing algorithms implemented in this project, SPF and AntNet are both Next-hop routing. Each node maintains for each destination a neighbor node id (*next-hop*). Each data packet stores information about destination node id. When a node receives a packet it forwards the packet to destination node. But how to choose an optimal path to forward the packet is the key point to evaluate the routing algorithm. To achieve this, next-hop routing do the following functions: (1) each node keeps a topology table for its outgoing links, a dynamic link cost is updated regularly according to the traffic flowing through the link. (2) link cost information is spread regularly to nodes of the network; (3) each node update topology table regularly based on the link cost information. The testbed built is called CuRS, which is developed as a modified and extended version of MaRS.

AntNet, the adaptive and distributed routing algorithm we evaluated in this project is a mobile agent-based, online Monte Carlo technique which takes inspiration from real ants' behavior in finding the shortest paths using magic chemical substance deposited by other ants, and this technique been applied to optimization problems. The amazing routing algorithm, AntNet is using a set of concurrent distributed agents to collect information and update the previous information to create a more optimal solution for the adaptive routing problem while agents exploring the network. We compare AntNet with SPF algorithm using the CuRS (Carleton University Routing Simulator) as testbed to evaluate. AntNet shows the better performance.

The paper is organized as follows.
Section 2 describes the design and implementation of CuRS. It includes features and implementation considerations, structure of CuRS, parameters and packets, the user interface of CuRS.
Section 3 describes the AntNet in detail and briefly describes SPF used for comparison.
Section 4 reports the experiment result.
Section 5 is conclusion.
Section 6 discusses future work.

2. Design and Implementation of the Simulator

2.1 Features and Implementation Considerations

The dynamics of the network is very complex. A testbed is essential to understanding the complex dynamics of routing systems. Ideally, we would like the testbed to accurately

model features that are important to routing performance. However, it is not clear which features are important for routing performance. So CuRS still keeps the capabilities of the MaRS by having easily extendible modules to represent these features.
a) The testbed should be modular. That is, adding new routing algorithms or new kinds of workload nodes, links should be easy. Such additions require programming.
b) It is easy for user to build kinds of network topologies and define different workload distributions without programming.
c) The user can specify kinds of topology changes such like be able to fail and repair a link at a specified time and see if and how it affect the system and how the system converges to a new steady state.
d) It provides standard performance measures such as average delay, average throughput, average load, and so on. It also has average and instantaneous versions of these measures.

CuRS is a discrete-event simulator. The target system is characterized by discrete states and by state transitions (events) occurring at discrete time instants. For every class events, there is a corresponding eventListener. When an event fires, the corresponding method of the associated eventListener is called. Which updates the system state and cause other event occur. Discrete-event simulation is more flexible than other simulation (for example process emulation), so it can offer the greatest flexibility in modeling the target system. After checking the available discrete-event simulator MaRS for a while, I found MaRS is suitable for the purposes of my project. So I consider to develop my Java simulator based on the MaRS-Version 1.0 (C language), and extend to include AntNet.

2.2 Structure of CuRS

CuRS has the same structure as MaRS: a simulation engine, user interface (GUI, command line options and files), and a set of components that simulate physical network, workload, and transport protocols. But current version of CuRS is implemented GUI.
The simulation engine manages the event list and user interface. (GUI is not implemented in this project). When the CuRS starts, the simulation engine processes command line options (including reading any files), and then goes into the simulator loop. If GUI was added in the future, the simulation engine will process inputs from the keyboard and mouse, execute the next event from the event list, and update the graphical user display.
I use Observer/Observable design pattern (which is Object Oriented design) to design this testbed. I let Event as Observable which maintains an eventListener instance variable. There are four kinds of eventListener, all of then are Observers. All Observer objects watch the Observable object. If the specific event is fired, the associated Observer object's related method will be called. There are four different classes of event occurring in the CuRS: command type events EV_CLASS_CMD, event type events EV_CLASS_EVENT, private type events EV_CLASS_PRIVATE, and record type events EV_CLASS_RECORD. I defined an Event class as an Abstract class, which has
enqueueEvent(int type, Component src,Component dest,long time,Packet pkt,Object arg) method to create a new event and places it in the event queue to be fired at the proper time. And also has fire () method. The four classes events are four concrete classes, and each of them has its own eventListener and its fire() method which invokes the eventListener's

specific method depending on what kind of event type it is.
All CuRS objects have to react to the command class events. These events establish the basic operations of the simulator and execute in the simulator engine. Event class events are defined globally in a central location. Sending of a packet from a node to a link is kind of event type event. Private class events are only defined internally within an object, and they are called within that component. Record class events are used to record events in a record file. Each class event has different types of events. For specific meaning of each type of event, the reader can find them in the programmer or user's manual of MaRS-Version 1.0.

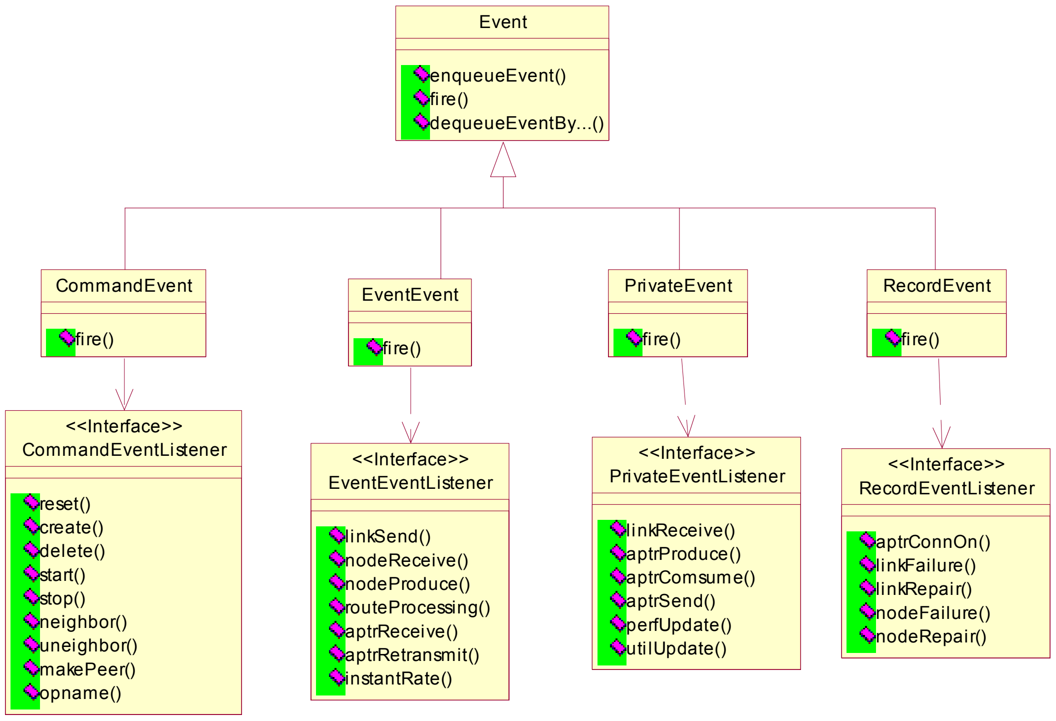

Figure 1: the relationship between Events and EventLsiteners

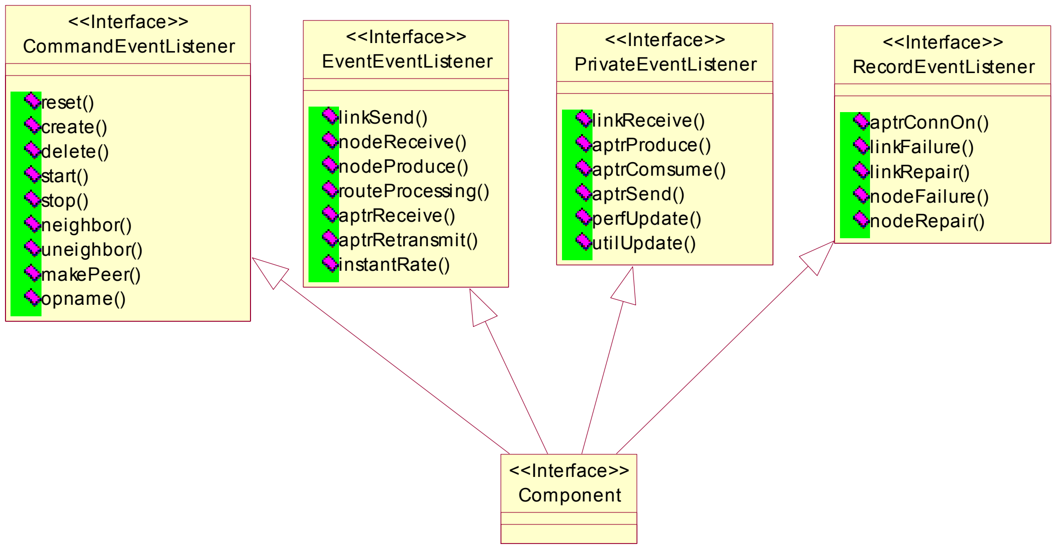

Figure 2: the relationship between EventLsiteners and Component

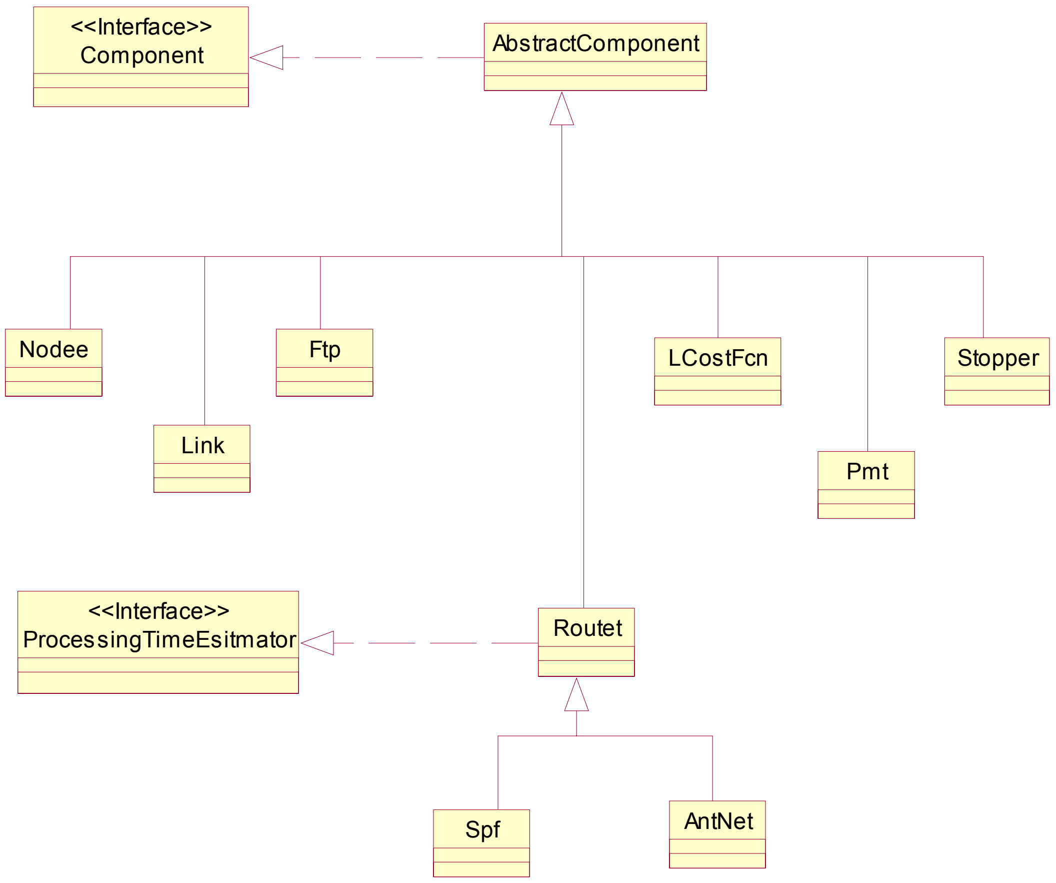

Figure 3: The relationship between Components

The components are used for modeling the target system and certain simulation functions. Each component is object which has its states and behaviors. In MaRS every component has its action routine associated with different events specified in the MaRS version 1.0. When an event occurs, each component's action routine is invoked for that event. Actually it uses function point in C to achieve this. While there is not function point available In Java. So in CuRS I use Observer/Observable Pattern to achieve this. Figure 1, 2, 3 shows the relationship of components Diagrams. Component was designed as Interface to represent all components in the target system. Such as Node, Link, LinkCostFunction component, Routing component, Workload component (FTP) and performance monitor. Component Interface extends four kinds of EventListener Interface which has different methods for different types events and also has association with one of four classes' event. Every concrete component implements its associated methods to override the methods defined in the AbstractComponent. Each component is an Object. The target system consists of a set of objects. By interconnecting multiple objects of various types of components, a target system of arbitrary topology, routing protocol, and workload can be obtained.

The components schedule events for each other. Also, a component can schedule an event for itself. The time of this event to happen depends on the link propagation delay. Each component has methods to update system state and may schedule other events depending on the type of the event received by the component.

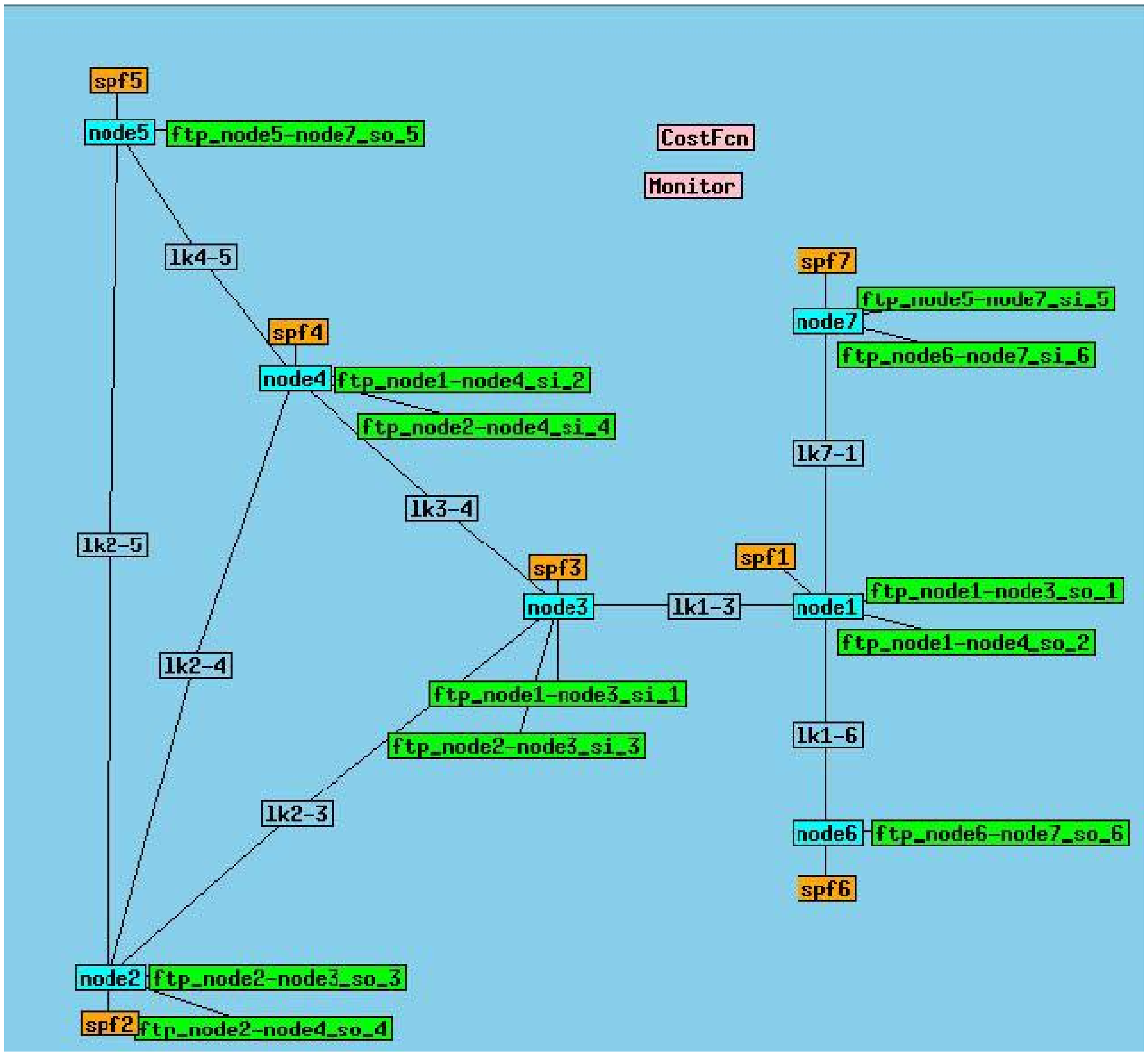

Figure 4: The example network

Figure 4 shows a target system of seven nodes and eight links objects. Each link is connected to exactly two nodes. A node models the physical aspects of a store and forward entity such as a host or a switch. Each node is connected to an SPF routing algorithm and zero or more workload components. Each workload component is either a source or a sink of FTP. For each source object, there is a corresponding "peer" sink object. There is one link-cost function object which is not connected to any one.

I briefly describe each type of object in the following.

Link

A link object models a bidirectional transmission channel between two node objects. I use the same data structure to implement Link class as Link Component in the MaRS. Each link object contains two LinkedLists of packets, each representing the packets in the transit in one direction. A link can be subject to failure and fails. When a link fails, all packets that are in transit are lost.

The input parameters of link are Propagation delay[microseconds], Bandwidth [bytes/second], Distribution of interfailure times, Distribution of repair times; The output parameters are Link status(up/down), Instantaneous workload utilization for each direction, Instantaneous routing utilization for each direction.

Link cost function

The link cost function is used to maintain and periodically update a cost for each link in the network. When the cost is updated, the link cost value is calculated by applying a transformation to a statistic cost "raw cost" which is obtained on the link during the previous update period. Hop-count, utilization, delay, and hop-normalized delay are the row cost supported by CuRS.

There should be only one LCostFcn object. So I use Singleton pattern to ensure only one instance of Link cost function exist in the target system. There is a static variable costFcn and a static method named "getCostFcn()" in the LcostFcn.java. Link cost function is not a part of the network so do not connect it to a component in the network. If an interaction is needed between link cost function and components, link cost function object obtained from LcostFcn.getCostFcn ()".

Routing

I define Routet as abstract class to represent the common things of each routing algorithm; Routet is also a kind of component. Every specific routing object is the concrete class of Routet and executes a routing algorithm. At each node, the attached routing object builds and maintains a routing table, which specifies a next-hop for each destination.

The routing object exchanges information by routing packets, more precisely, a routing object passes a routing packet to its local node, which then sends it to a neighboring node, which then passes it to its local routing object. In the current version, I just implemented two types of routing components corresponding to two routing algorithms: SPF and AntNet.

The input parameters of SPF are Mean and standard deviation of topology broadcast period [milliseconds]; The output parameters are Global topology table (adjacency matrix), local topology table, and Routing table.

Workload

The workload component model the user traffic in the target system. In each workload source-sink pair, the source component produces workload packets and passes them to its node component. These packets are then forwarded by the node components until they reach their destination node, where they are consumed by the sink component.

Input parameters of FTP_Source and FTP_Sink are Initial source-sink status (on/off), Number of connections between the specified FTP_Source and FTP_Sink, Average number of packets per connection, Packet length [bytes], Average delay between connections [milliseconds], Delay between packets [milliseconds], Produce window size [packets], Send window size [packets]; The output parameters are Source-sink status(on/off), Number of the current connection, Number of packets in the current connection, Number of the current packet in the current connection, Number of packets produced, Number of packets sent, Number of packets acknowledged, Number of packets received, Number of packets retransmitted, Roundtrip time estimate [microseconds], Instantaneous sent rate, Instantaneous acked rate, Instantaneous retransmission rate, Instantaneous delay per packet [milliseconds].

Node

A node has a separate LinkedList to store packets for each outgoing link, and a single

common LinkedList for all routing packets received on its incoming links. The node passes the routing packets to the local routing object. For the incoming workload packet that is not destined for a local sink, the node consults the local routing table and appends the packet to appropriate outgoing LinkedList. A node can also be subject to failure and repair. When a node fails, all packets stored are lost.
The input parameters of node are Delay to process a packet [microseconds], Total available buffer space [bytes], Distribution of interfailure times, Distribution of repair times; The output parameters are Node status 9up/down), Amount of buffer space currently used [bytes], Maximum amount of occupied node buffer space so far in the simulation, Number of workload and routing packets dropped by the node in the simulation, Instantaneous packet drop rate, Current buffer utilization, Queue length [packets].

Performance monitor

The performance monitor is used to collect and evaluate statistics about the network. There are two ways to update the performance measures: periodically-updated and event-updated. Updating periodically at time instants called "update period" is periodically-updated performance measure. An event-updated performance measure is updated whenever a related event occurs. A performance measure can be either an average measure or an instantaneous measure. An average measure is computed based on statistics collected after a specified "startup interval" from the beginning of the simulation. An instantaneous measure is computed based on current state or statistics collected during the last update period.
When a target system is built to evaluate one routing algorithm, there should be only one performance monitor object. So I use Singleton pattern to ensure only one instance of Performance Monitor exist in the target system. There is a static variable performanceMonitor and a static method named "getPmt()" in the Pmt.java. Performance monitor not a part of the network so do not connect it to a component in the network. If an interaction is needed between performance monitor and components, Performance Monitor object obtained from Pmt.getPmt() is called in an event driven way like pm(link, LINK, LINK_DOWN,0,0,0) or poll at start time the component should issue pm(component, compTtype, NEW_COMPONENT), so the pm knows the component.
The output parameters of the performance monitor are Average network throughput, Instantaneous network throughput, Average delay per packet, Instantaneous delay per packet, Maximum packet delay, Current number of connections in the network, Average number of packets per FTP connection, Average life-time of an FTP connection, Current number of failed links, Total number of dropped workload packets, Total number of routing packets, Average data load, Instantaneous data load, Average routing load, Instantaneous routing load, Buffer usage statistics.

Stopper

The stopper component enables the user to define a termination condition for the simulation, based on the input parameters and the statistics collected by performance monitor.
The input parameters of the stopper are Number of intervals, Interval length [microseconds], Error bound [percentage].

2.3 Parameters and packets

Parameter is an object used to store any information about a component that needs to be displayed on the screen, logged to disk, or saved in a network file. The instance variables of a Parameter object are as follows:

```
  Private String name;
  private int displayType;
  private int flags;
  private double scale;
  private int logType;
  private Object value;
```

The actual value of a parameter object is stored in the value variable. Because it is Object type, so it can be an Integer, Double, a LinkedList, a String or something else depending on what kind of type value an Component object want to store in that parameter. And flags describe how to save and/or display the parameter. It will be used when the GUI was added to the CuRS. A component object can have as many parameters as needed. They are stored in the LinkedList named params which is defined in the AbstractComponent class.
The target system is a message passing system. Messages are passed in the packets. There are two main types packets in the simulator: TR_PACKET is the application/transport part of the packet. ROUTE_PACKET is the routing part of the packet. Routing packet has higher priority than the data packet.

2.4 Interface design and implementation

User Interface
Currently CuRS just provides command line options and files as user interface. The graphical user interface will be added in the next stage of this project. The command line options include the specification of the length of the simulation run, the seed for random number generator, the name of the network file, the name of the record file, the length of the startup interval, and the performance monitor update period. The user manual is provided in section 7 of this report.
Files
CuRS works with four ASC|| files, they are the network file, the snap file, the record file and the log file.
A network file includes a description of the network to be simulated. It consists of the three distinct parts. The first part is a list of all component objects with their input parameter values; the second part indicates the interconnection of the objects. The third part lists the peer components. The networkfile is created manually. If the GUI is added in the future, then the user can create network using GUI. The network files include all information needed to carry out a simulation.
A snap file actually is a networkfile with additional information such like the values of the current output parameters, the seed for the random number generator, the time of the snapshot, and the flags that indicates the parameters logged.
CuRS provides record and play options which is specified from the command line. With those options, the record-class events are read and scheduled from the record file. So we can simulate different routing algorithms subject to the same sequence of the event and occurrence times for a special set of events.

CuRS can also log the value of the selected parameters to a log file. For the programmer to implement GUI or extend CuRS for specific purposes, he/she can debug or check error or warning in the log file. Actually I use Log 4j to log things. If the programmer would like to log specific level log information, he/she can specify the level in the log.property file that is in the classes subdirectory.
The file interface used in the CuRS is same as MaRS.

3. AntNet: An adaptive Agent-based Routing Algorithm

The AntNet algorithm implemented in this project is come from the AntNet described in Gianni Di Caro and Marco Dorigo's paper "AntNet: Distributed Stigmergetic Control for Communications Networks". AntNet is used to solve combinatorial optimization problems and telephone network routing. The main ideas of these techniques are using the repeated and concurrent simulations carried out by a set of ant agents and the information collected by the ant agents to generate new solutions to the problem. In an iterative process, each ant builds a solution by using problem specific information and information added by other ants, and it can access that information locally. So each ant collects information on the problem characteristics and on its own performance, and also use this information and other information got by other ants to modify the representation of the problem, further builds a better solution.

The scheme of the AntNet algorithm can be summarized as follows:

- Mobile agents (forward ants) are launched asynchronously from each network node to the random selected destination nodes, and within regular periods and with concurrently the data traffic.
- Agents communicate with each other through the information they read and write locally to the nodes. They act concurrently and independently.
- Searching a minimum cost path between its source and destination is the mission of each agent.
- Each agent moves step-by-step towards its destination node. At each intermediate node, a stochastic policy makes use of local agent-generated and maintained information and local problem-dependent heuristic and agent-private information to help choose the next node for the ant agent to move on.
- The time length, the congestion status and the node ids of the followed path are the information collected by moving agents.
- Once agents reached the destination, they will die and launch backward ants to their source nodes by moving backwards along the same path in the opposite direction which forward ant traveled.
- When agents go backward, local models of the network status and the local routing table of each visited node are modified by the agents as a function of the path they followed and of its goodness.
- When agents (backward ants) have returned to the source node subject to forward ant, they die.

AntNet has two set of mobile agents: forward and backward ants. They have the same data structure, but different situation in the environment. That is, they sense the different inputs and produce different, independent outputs. The detailed description about AntNet routing algorithm is not represented here. Reader can find it in the paper "AntNet: Distributed Stigmergetic Control for Communications Networks" written by Gianni Di Caro and Marco Dorigo in 1998. Here I just talk about how to implement the AntNet in this simulator. It also illustrates how to implement a new component in the CuRS. As described above, AntNet routing algorithm needs ant agents to collect and update the information about network condition. So each agent was defined as an Object. Here two kinds of events: LaunchForwardAnt, LaunchBackwardAnt, are added in the Event class. AntNet is an Object used to deal with routing processing.

To evaluate the performance of the AntNet, SPF (Shortest Path First) routing algorithm is implemented in this project for comparison. Which is link-state routing algorithm. In link-state routing algorithms, each node tries to maintain a database describing the network topology and the link costs. Using this database, each node independently calculates shortest paths, and uses them to decide the next hop for any destination. Each node broadcasts the costs of its entire outgoing links to all other nodes in the network.
For better evaluate the performance of the AntNet, OSPF should also be implemented for comparison routing algorithm.

4. Experiments and Results

When evaluating the routing algorithm, the most common performance measures are throughput and average packet delay. Throughput indicates the quantity of the service that the network has been able to offer in a certain amount of time; While the average packet delay is used to measure the quality of the service produced at the same time. We already defined the performance monitor in the simulator. So the user just need create the network topology with the condition set. After simulating for the specific period, then compares the difference of the output parameters of the Performance Monitor using the different routing algorithms. then obtains the results.
In this project, I just use two network instances to evaluate the performance of the routing algorithms. One is the example network in the Figure 4, another is the NSFNET in the Figure 5.

4.1 Example Network (7 nodes, 8 links)

The example network composed of 7 nodes and 8 bidirectional links. In the experiments with example network, we initialize the random number generator using the 10 different values of seed which generate 10 different sequences of events. For each sequence of events, we collected the measures of both routing algorithms we try to compare. The other option value we used in this experiment are: -perfdt 1000000 –skipt 5000000 ./etc/example-net 10000000.

Routing Algorithm	seed	Time of Snapshot(ticks)	Total network throughput	Average delay/packet
SPF	758596	1000162	76.77512485954551	18.397466666666666
ANTNET	758596			
SPF	98056	1000162	76.77512485954551	18.388720000000003
ANTNET	98056			

SPF	98	1000022	51.19774729911884	20.04286
ANTNET	98			
SPF	6	1000162	76.77512485954551	18.38612
ANTNET	6			
SPF	666	1000045	76.79308862202402	18.37616
ANTNET	666			
SPF	666898989	1000045	51.19539241468268	20.04538
ANTNET	666898989			
SPF	6899	1000051	51.19477813263047	20.04344
ANTNET	6899			
SPF	16	1000002	76.7996928012288	18.374733333333335
ANTNET	16			
SPF	160	1000141	51.18556567048092	20.04496
ANTNET	160			
SPF	1600	1000096	76.78525723061172	18.385493333333333
ANTNET	1600			

Table 1: Measures with different routing algorithms and seeds

4.2 NSFNET (14 nodes, 21 bidirectional links, average degree 3)

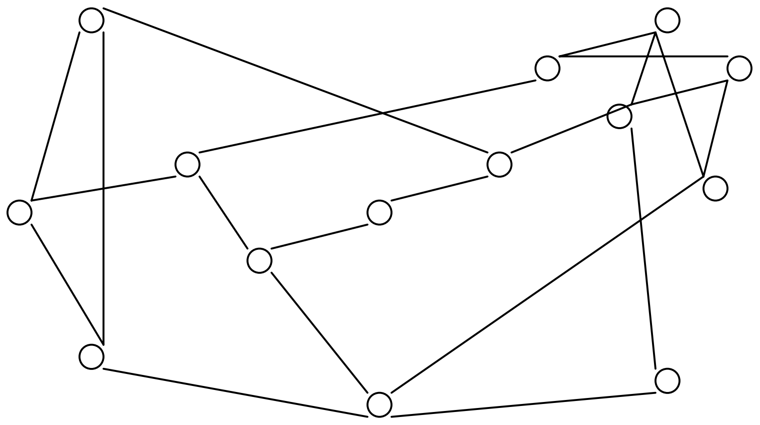

Figure 5: NSFNET.

5. Conclusion

CuRS provides a flexible platform for the evaluation and comparison of routing algorithms. The user do not need to worry what kind environment CuRS can be runnable. The user can compile and run it on most platforms such as: Windows, UNIX and Linux. However, MaRS is no that portable. For example, when I downloaded Linux version of MaRS from the website, it is not runnable in Linux 7.x (latest version Linux). I do not want to install old version of Linux again, so it take me for a while to fix the makefile of MaRS to make it work in Linux 7.x. Also for the user would like to extend CuRS for specific purpose, the Java code is easier to modify and test than C. The design of CuRS is kind of Object Oriented design, and Observer/Observable design pattern is used to design the relationship between event and component, instead of the function point call used in the MaRS. So the code is more reusable.

6. Future Work

The current version of CuRS does not provide GUI yet. However, the GUI is friendlier for user to create the network and efficiently measure the performance of the different routing algorithms or workloads. MaRS provides command line interface and also can be compiled with one of two optional graphical interfaces: a simple X interface, or an X-Motif interface. The details of how to design and implement the GUI, the programmer can refer to part 6. User Interface of the MaRS Version 1.0 User's Manual which is available on the website I listed in the reference part. MaRS can also be downloaded from that website. But that version of MaRS does not work in the Linux 7.x. While I have a modified version of MaRS which is runnable in the Linux 7.x. I would like share it with the people who enjoy implementing the GUI for CuRS. And also understanding the Motif of the MaRS will be very helpful to implement GUI for CuRS. Because CuRS is implemented in Java, so it will be better to implement the GUI using Java AWT or Swing. And I am glad to answer and discuss the questions about CuRS.

7. User Manual

7.1 CuRS Directory Structure

The CuRS directory contains subdirectories for the source code, class files, libraries files, and properties file. It is organized as follows:

README	
classes	Subdirectory for the executable files (i.e. .class files)
doc	Subdirectory for the project report
etc	Subdirectory for the miscellaneous files like example networks
lib	Subdirectory for the library files (i.e. fada-http.jar)
src	Subdirectory for the source code files (i.e. .java files)

7.2 Start CuRS

After CuRS is installed, it can be started by typing

```
java −Dlog.properties=config/Log.properties SimMain [options] networkfile [stoptime]
```

The networkfile contains the description of the target system. The stoptime value, if present, is used to stop the simulation when the simulated time exceeds this value. In the current version, the option are as follows:

-x

start the simulation of the target system immediately, without the GUI. This option should not be used if the networkfile is not present.

-s seed

initialize the random number generator using the value of seed.

-i

when the simulated time exceeds th stoptime value specified, the user is asked whether or not to continue the simulation.

-record

write the occurrence of the record type events to the file sim_event.process_id.

-play filename

read and schedule events from the file filename, which should be a file that was created using the -record option.

-perfdt update_period

for periodically-updated performance measure use the value of update_periiod as the update period length in microseconds.

-skipt startup_interval

average performance measures are based on statistics collected after the simulated time exceeds startup_interval in microseconds.

The -x option is useful for long simulation runs; with this option the simulator runs faster. The -record/-play options allows the user to repeat a sequence of record type event occurrences across different simulation runs. It is useful when we evaluate different routing algorithms subject to the same failure/repair event occurrences. The -perfdt option can be used to control the size of the log file, since most of the performance measures are periodically -updated performance measures.
The current version of the CuRS does not support GUI, so the user need to create the networkfile manually. There is a example networkfile in the CuRS. When the GUI part is added in the future, the user can create the network using GUI and save the network configuration by the clicking the save button.

7.3 Example

CuRS provides a small networkfile called "example-net" in the etc subdirectory. The user can start CuRS with this networkfile to gain first experience with the simulator. After getting familiar with CuRS, the user should try to build his/her own networkfile.

8. References

- MaRS-Linux Version : http://www.cs.bu.edu/fac/matta/software.html

- C. Alaettinoglu, A. U. Shankar, K. Dussa-Zieger, and I. Matta. *Design and Implementation of MaRS: A Routing Testbed.* http://citeseer.nj.nec.com/update/204312

- C. Alaettinouglu, K. Dussa-Zieger, I. Matta, A.U. Shankar, and O. Gudmundsson. Introducing MaRS, a Routing Testbed. *ACM SIGCOMM Computer Communication Review*, 95-96 (1992).

- C. Alaettinouglu, K. Dussa-Zieger, I. Matta, A.U. Shankar, and O. Gudmundsson. MaRS (Maryland Routing Simulator) -User Manual. UMCP CSD CS-TR-2687 (1991).

- C. Alaettinouglu, K. Dussa-Zieger, I. Matta, A.U. Shankar, and O. Gudmundsson. MaRS (Maryland Routing Simulator) -Programmer Manual. UMCP CSD CS-TR-2723 (1991).
- Dandamudi P Sivarama, Networking courses notes (95.323), 2001 Winter

- Gianni Di Caro, Marco Dorigo, *AntNet: A Mobile Agents Approach to Adaptive Routing* (1997), http://citeseer.nj.nec.com/update/409626

- Peterson L Larry & Davie S. Bruce, *Computer Networks*, Morgan Kaufmann, San Francisco, California, Second Edition, 2000.

- Roberts Simon, Heller Philip & Ernest Michael, *Complete Java 2 Certification Study Guide*, SYBEX, Alameda, CA, 1999

- Waite Mitchell & Prata Stephen, *The Waite Group's New C Primer Plus*, SAMS, Carmel, Indiana, Second Edition, 1993

- Wall Kurt, *Linux Programming by example*, Dean Miller, Indianapolis, Indiana, 2000

9.Appandix