Sequential and Parallel Discrete Event Simulation on Computer Communication Networks

Tao Zhou
Western Michigan University

Follow this and additional works at: http://scholarworks.wmich.edu/masters_theses
Part of the Computer Sciences Commons

Recommended Citation
http://scholarworks.wmich.edu/masters_theses/952

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
SEQUENTIAL AND PARALLEL DISCRETE EVENT SIMULATION
ON COMPUTER COMMUNICATION NETWORKS

by

Tao Zhou

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
June 1992
This thesis presents the discrete event simulation technique on the network design, focusing on the methodology, modeling and implementation. The developed technique is suitable for the analysis of any store-and-forward computer communication networks with any level of complexity. The proposed generic simulation model can be used to test the architectures and protocols of existing networks and networks being designed. In implementation of the sequential discrete event simulation, an object-oriented programming strategy is applied to the network simulation using the C++ language. The parallel discrete event simulation is introduced to solve the problem of long time execution of the sequential simulation program. Specifically, the hypercube architecture is used to study the problem of distribution of logical processes. A variant Time Warp scheme with partial cancellation is also proposed to solve the problem of process synchronization.
ACKNOWLEDGMENTS

I would like to express appreciation to my thesis advisor, Dr. Dionysio Kountanis for his time, encouragement, understanding and assistance with the writing of the thesis. I would like to express appreciation to my thesis committee member, Dr. Robert Trenary who taught me C++. I would like to express appreciation to my thesis committee member, Dr. Kenneth Williams for his reviewing the thesis. Without their help, this thesis would not have been complete.

A note of appreciation goes to The Graduate College for a fellowship for my first year study and the Computer Science Department for a teaching assistantship for my second year study.

Special thanks go to my wife, Amy Chang for her love, assistance and support during my study at Western Michigan University.

I am especially grateful to my mother, Xiuyun Guo and my father Xiqing Zhou for their love, encouragement, support and perseverance throughout the years of my study.

Tao Zhou
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Sequential and parallel discrete event simulation on computer communication networks

Zhou, Tao, M.Sc.

Western Michigan University, 1992
# TABLE OF CONTENTS

| ACKNOWLEDGMENTS                       | ii |
| LIST OF FIGURES                       | v  |
| **CHAPTER**                           |    |
| **I.** INTRODUCTION                   | 1  |
| Measurement and Analytical Calculation| 3  |
| Techniques                            |    |
| Simulation Technique                  | 4  |
| Discrete Event Simulation             | 5  |
| Discrete Event Simulation Suitability for Network Modeling | 8 |
| Importance of Parallel Discrete Event Simulation | 9 |
| Organization of the Following Chapters | 11 |
| **II.** OVERVIEW OF COMPUTER COMMUNICATION NETWORKS | 12 |
| ISO/OSI Reference Model               | 16 |
| Design of the Data Link Layer         | 19 |
| Design of the Network Layer           | 24 |
| Remark                                | 31 |
| **III.** SIMULATION MODEL OF COMPUTER COMMUNICATION NETWORKS | 33 |
| Modeling a Switch Node                | 34 |
| Modeling Network Topology             | 38 |
| Modeling Network Protocols            | 41 |
| Measures of Performance               | 46 |

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table of Contents—Continued

CHAPTER

IV. SEQUENTIAL DISCRETE EVENT SIMULATION PROGRAM ........................................... 49
   Discrete Events of Network Simulation ......................................... 50
   Network Simulation Program ................................................... 52
   Network Simulation Implementation via OOP in C++ ............................. 57
   Remark .............................................................................. 64

V. PARALLEL DISCRETE EVENT SIMULATION ON NETWORKS ......................................... 65
   PDES Problems ....................................................................... 65
   Hypercube Architecture for the solution to PDES Problem1 ................. 67
      Hypercube ......................................................................... 68
      Distribution of Network Simulation LPs in a Hypercube .................. 69
   A Variant Time Warp Scheme for the Solution to PDES Problem2 .......... 73
      Time Warp Scheme .................................................................. 74
      A Variant Time Warp With Partial Cancellation for Network PDES .......... 77

VI. CONCLUSION ............................................................................. 81

BIBLIOGRAPHY ............................................................................. 85
LIST OF FIGURES

1. A Typical Flow Control of Discrete Event Simulation ....................................... 7
3. Six Network Topologies ........................... 15
4. The OSI Network Architecture With 7 Layers..... 17
5. The Relationships Between Four Service Primitives.............................................. 21
6. An Example of a Frame Format in HDLC Protocol.. 22
7. A Slide Window Protocol........................................ 24
8. An Example of Application of the Shortest Path Routing Algorithm......................... 29
9. The Model of a Node................................. 35
10. One-step Transition Probability Matrix P ...... 38
11. An Example of Network Topology With Full-duplex Lines................................ 40
12. Connectivity Matrix C Representing Topology in Figure 11............................... 40
13. Routing Matrix R for the Network in Figure 8(a)............................................ 41
14. The Definition of Event Type..................... 52
15. The Modules of the Network Simulation ........ 54
16. The Inheritance Hierarchy of the Classes in the Network Simulation Program........... 61
17. A 3-Hypercube With 8 Nodes..................... 68
18. An LP Distribution With an Exact Embedding of the Given LP Structure Onto a Hypercube... 70

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
List of Figures--Continued

19. An LP distribution With an Approximate Embedding of the Given LP Structure Onto a Hypercube..... 71

20. A Flow Control of an LP for Synchronization.... 80
CHAPTER I

INTRODUCTION

Computer communication networks have been playing an important role in this information age. They dissolve the geographic barrier which separates users and make it easy for them to exchange information whether they are at the same site, at opposite ends of the country or at different countries around the world. People use computer networks to exchange information such as programs, files, mail, etc. Networks provide the facilities which allow one to communicate quickly and conveniently with others.

As the demand for the exchange of information has increased rapidly during the past decade, computer communication networks are evolving in size, speed and complexity to meet this need. In order to model, analyze, design and implement computer networks efficiently, it is necessary to have good performance evaluation techniques. Many cases of severe performance degradation of networks have been documented due to problems not anticipated by network designers (Frost, LaRue & Shanmugan, 1988; Kurose & Mouftah, 1988). With adequate performance modeling, it is possible, for those designers who work on the network layer
and the below layers of the ISO/OSI Reference Model, to have greater insight as to the operation of the network and to anticipate and correct inadequate network performance while avoiding the costs of altering an existing network.

The actual performance measures of interest to the network designer depend upon the type of network being designed, its application (e.g., data or voice transmission) and whether the network is circuit-switched or packet-switched. In general, examples of useful performance measures for the packet-switched networks are throughput, packet delay, buffer requirements, link utilization, etc.

There are several approaches which can be used to obtain performance estimates of a network (Kurose & Mouftah, 1988). The traditional approaches are measurement and analytical calculation. The first approach is to actually measure the performance of an existing network, but that will interrupt the ongoing operation of the network. The second approach is to use mathematical analysis to estimate the performance of a network. The later approach is thoroughly described in Schwartz’s book (Schwartz, 1977), but it cannot model today’s complex and diverse networks.

In order to avoid these deficiencies, simulation has been explored in the use of design and analysis of communication networks. This thesis applies discrete event
simulation to network analysis and focuses especially on
the methodology, modeling and implementation of this
 technique. The developed technique on network modeling
will be suitable for the analysis of packet-switched
networks with any amount of complexities. In implementa-
tion of simulation, an object-oriented programming tech-
nique is applied to network simulation with C++ language.
Parallel discrete event simulation on networks is investi-
gated to reduce the time consumed in the execution of
simulation.

In the following sections, some basic concepts about
evaluation techniques and simulation are introduced.

Measurement and Analytical Calculation Techniques

Measurement is a direct approach to evaluation of
network performance. It is also the most expensive one
because it needs the network to exist before any measure-
ment can be taken. Since it obtains performance estimates
from an existing network, the operation of the ongoing
network will be suspended while experiments are being
performed. These experiments may take far too much time to
produce results. It is therefore economically infeasible
to do these experiments on a real network, though the data
gathered by such measurements can be used for evaluation of
the network.

Analytical calculation is a mathematical analysis
method of network performance evaluation. It requires a high degree of abstraction as the network designer develops a performance model which reflects the network under study. However, the calculations involved in an analytic model is, in general, quite difficult to carry out. For instance, consider networks with complex job scheduling policies, finite buffers (causing blocking), simultaneous resource possession by a job, complex timing constraints, etc. Therefore, the network modeler runs a risk of distorting a model and ending up with a wrong conclusion for the real network.

Simulation Technique

As described above, the two techniques are not suitable for performance evaluation of the network if either the operation of a real network is interrupted or the model of a complex communication network is distorted. Simulation technique could become a feasible alternative to the two models discussed on the previous section.

Simulation technique can be used to model a network to an arbitrary level of detail, thus allowing the network designer to evaluate the performance of a network at any level of complexity. Less abstraction is required to simulate a network and the process of modeling is a straightforward task. Simulation is driven with random inputs generated by pseudo random number generators. The
results produced by the simulation are thus stochastic. Simulation technique can also provide better control over the experimental conditions and models because no real networks are necessarily built before experiments are carried out. Measurement and analysis methods can also be used as a complement in some places of simulation for performance evaluation. These advantages of simulation are the main reasons that simulation has been the dominant force in the analysis and evaluation of network design.

The major disadvantages of simulation are the time needed to write, debug and execute a sophisticated simulation program and the effort needed to validate the simulation and analyze its statistical output.

Many studies have been performed on the development of simulation technique in the areas of operations research and computer performance modeling. Most text books on simulation include discussions of these methods. However, the main task of this thesis is to develop simulation technique in the design and analysis of computer communication networks.

Discrete Event Simulation

Discrete event simulation models a dynamic system as it evolves over the passage of time by a representation in which the state of the system changes only at a countable number of points in time. At these points in time an event
occurs. An event is an instantaneous occurrence which changes the system state. In order to keep track of the current value of simulated time in a dynamic system as the simulation proceeds, a simulation clock is needed to advance simulated time. The approach to advancing the simulation clock is called next-event time advance. With this approach, the simulation clock is initialized to zero and the times of occurrence of future events are determined. The simulation clock is then advanced to the time when the most imminent of these future events will occur. At this point of time the system state is updated and new future events are determined to account for the fact that an event has already occurred. These future events are inserted into an event list which contains the next time when each event will occur. The simulation clock is advanced to the time when the new most imminent event will occur, and the state is changed and future events are scheduled. This process is continued until some prespecified stopping condition is finally satisfied.

A typical flow control of discrete event simulation is given in Figure 1, which includes four routines called by the main program. A similar chart can be found in Law and Kelton’s book (Law & Kelton, 1982).

In the initialization routine, the simulation clock is set to zero, and the system state, the statistical counters and the event list are initialized.
In the time routine, the most imminent event is determined. If an event $i$ is the next to occur, the simulation clock is advanced to the time that event $i$ will occur.

---

**Initialization routine**

- Set the simulation clock = 0;
- Initialize the system state and the statistical counters;
- Initialize the event list.

---

**Timing routine**

- Determine the next event;
- Advance the simulation clock.

---

**Event routine**

- Update the system state;
- Update the statistical counters;
- Generate the future events and add to the event list.

---

**Report routine**

- Compute estimates of interest;
- Print the report.

---

Figure 1. A Typical Flow Control of Discrete Event Simulation.

In the event routine $i$, three types of activities occur. The system state is updated due to the occurrence
of the event \( i \), information about the system's performance is gathered by updating the statistical counters, and the future events and their times of occurrence are generated and added to the event list. When the above activities have been completed, the satisfiability of the ending condition is checked. The end of the simulation means that enough statistical information has been gathered and thus, the system performance can be evaluated. The ending conditions can depend on a specified time, on the occurrence of a specified event, or on the steadiness of the system state. When the simulation is terminated, estimates of the simulated network's performance are computed and printed in the report routine.

Discrete Event Simulation Suitability for Network Modeling

The function of a computer communication network is to transmit information from one place to another on the network through shared transmission facilities. The network is composed of a set of stations (referred to as nodes), switches, multiplexers and concentrators which are interconnected by means of transmission systems. These facilities route messages from a source node on which one user generates the messages to a destination node on which another user receives the messages. Messages are randomly generated at different nodes and are transmitted to other nodes through different paths on the network. Messages can
have different priorities and lengths and can require different amount of processing. Therefore, a communication network system is a dynamic, stochastic and discrete system evolving over run-time.

Studies of network systems have showed that the operation of networks is suited for discrete event simulation (Chlamtac & Pranta, 1982; Chlamtac & Jain, 1984; Frost, LaRue & Shanmugan, 1988; Kurose & Mouftah, 1988; Lubachevsky, 1989). In a packet-switched network (store-and-forward network), the system state may include the length of message at some network node, forwarding a message to another node, acknowledgment, time-over, component failures, etc. These states and events which are updated and occur during the actual operation of the network can be simulated during the execution of the discrete event simulation program. The simulation program can also collect statistical information about the changing system state and later analyze the system’s performance based on the recorded information. Therefore, discrete event simulation is a powerful tool in the design and analysis of computer communication networks.

Importance of Parallel Discrete Event Simulation

The major disadvantage of simulation technique, as mentioned in the previous section, is the amount of time required to simulate a model. Today’s communication
networks may contain a great number of nodes, carry a large amount of traffic and operate at a very high speed. The simulation of such complicated systems needs a huge amount of computational time to carry out the simulation on a uniprocessor system. There are two options to reduce the time required to run the simulation and produce the results. One is to abstract or lessen the details of the simulation model, but in this way the simulation may destroy useful results and also diminish its attractiveness as an evaluation model. The other is to use techniques other than simulation, but it has been shown that the available techniques may not give the desired solution to the given problem. A new technique is being proposed to solve the problem. This technique is called parallel discrete event simulation (Fujimoto, 1990).

Parallel discrete event simulation, sometimes also called distributed discrete event simulation, concerns the execution of a single discrete event simulation program on a parallel computer. The general idea is to divide the simulation into numerous processes and distribute these processes among the processors of a parallel computer in order to obtain the results with a faster simulation time. Due to inherent parallelism in the operations of a communication network, network simulation is suited for parallel discrete event simulation and therefore helps to solve the problem of too long execution time. However, there are two
major problems encountered in the proposed approach: the distribution of processes on the processors of a parallel computer and synchronization of the processes. Solutions can be found that improve the application of parallel computations in simulation. This thesis will provide two solutions to the above two problems based on a special parallel architecture.

Organization of the Following Chapters

Chapter II reviews computer communication network architectures and protocols, including the ISO/OSI Reference Model, the functions of each of the seven standard layers and the design of data link and network layers which are the fundamental concepts in establishing a simulation model. Chapter III defines the generic simulation model of communication networks including the models of nodes, topologies and protocols. Chapter IV describes discrete events and simulation program along with the application of object-oriented programming via C++ in the network simulation. Chapter V studies parallel discrete event simulation on communication networks and provides two solutions to the above mentioned problems in the parallel network simulation. Chapter VI concludes with remarks on the results obtained and proposes future studies and problems.
CHAPTER II
OVERVIEW OF COMPUTER COMMUNICATION NETWORKS

A computer communication network (also called computer network or network) is defined to be a collection of interconnected computers, communication controllers and other devices. Many of today’s computer networks use public telecommunications facilities to provide users with access to the processing capabilities and data storage facilities associated with the mainframes and also to permit fast exchange of information among the users of the network. As the cost of micro electronic devices has decreased and the demand of information exchange has increased over the past decade, complex and diverse networks have been built. These networks connect together mainframes, minicomputers, microcomputers, intelligent terminals, communication controllers, cluster controllers, other programming devices, etc. A network connecting users who are widely separated geographically is called a wide area network (WAN).

In parallel with the growth of wide area networks, microcomputers have been spread rapidly and widely throughout organizations. These computers are generally used for word processing, spreadsheets, data bases, financial
analysis, processing control and other business applications. As the use of microcomputers has grown, a need for communications and resource sharing among microcomputers within relatively short distances has also increased. A network connecting users in an organization or building is called a local area network (LAN).

In order to fill the gap between local area networks and wide area networks, a metropolitan area network (MAN) has been designed. A MAN connects together computers in an organization's buildings or a cluster of factories and offices within a city.

In general, a network consists of a number of hosts and a communication subnet (referred to as subnet). The hosts are connected by the subnet. The subnet is composed of transmission links and switch nodes (referred to as links and nodes respectively). The switch nodes are specialized computers used to connect two or more transmission links. A generalized network structure is shown in Figure 2. Cloud boundary in the figure is used to represent the topology of the network. Six different network topologies are shown in Figure 3. These topologies are star, ring, tree, complete, mesh and mesh of tree. The similar figures can be found in Tanenbaum's book and Green's book respectively (Tanenbaum, 1989; Green, 1982).

There are two types of communication subnets: point-to-point subnet and broadcast subnet.
Figure 2. A Generalized Network Structure.

In the point-to-point subnet, a pair of switch nodes are connected by a transmission link which can be a cable telephone line, fiber optic, etc. If two switch nodes which do not share a link need to communicate, they must exchange messages indirectly through other switch nodes on the subnet. When a message (called a packet in the subnet) is sent from a source node to a destination node, it is received by each intermediate node, stored there until the chosen or required outgoing link is idle, and then forwarded to the next node. Such a subnet is called a point-to-point subnet, or a store-and-forward subnet. Almost all wide area networks are store-and-forward networks such as ARPANET, SNA, DNA, TYMNET, NSFNET, etc.

In the broadcast subnet, all switch nodes are connected by a single communication link. A packet sent by any
Figure 3. Six Network Topologies.

node is received by all other nodes. The packet contains
an address field which indicates the destination node. When a node receives a packet it checks the address to see if the address matches its own address. If it does, the packet is accepted, otherwise it is ignored. All packet radio communication networks are broadcast networks such as ALOHA system, AMPR, etc., and so are some local area networks such as Ethernet.

Various types of networks have been developed with various protocols. These protocols perform the functions of the networks and they can be implemented in software, hardware, or firmware. In order to standardize the protocols, provide a guide for developing new protocols and also compare and classify existing network architectures, a basic model of a computer network was proposed.

**ISO/OSI Reference Model**

As mentioned above, in order to provide a common basis for coordinating network development, the International Standards Organization (ISO) proposed in 1983 a basic model of a computer network to internationally standardize the various protocols. The model is called the ISO/OSI (Open Systems Interconnection) Reference Model which deals with the interconnection of systems open for communication with other systems. The OSI model has seven layers which define the network architecture. Each layer perform distinct functions according to the layer’s protocol and provides
different services to its immediate upper layer. The OSI network architecture with seven layers is illustrated in Figure 4. The functions of each of the seven layers are defined as follows. The materials below are from (Green, 1982; Martin, 1989; Tanenbaum, 1989; Yakubaitis, 1986).

**Figure 4. The OSI Network Architecture With 7 Layers.**

1. **Physical Layer:** The physical layer is concerned with the signals, functions and connectors for accessing and using the physical data path; e.g., wire coaxial cable, light link, fiber optic, radio, etc.

2. **Data Link Layer:** The data link layer provides for the transmission of data on single links, including connection establishment, error control, flow control and maintenance of the link through the use of the physical...
layer. It also defines and forms the data link frame.

3. Network Layer: The network layer provides a functional data path between two nodes in the network, and defines procedures for routing, packet sequencing, error detection and recovery, and congestion control. Packet switching (storing and forwarding) and data transmission between communication networks (internetworking) are performed in this layer.

4. Transport Layer: The transport layer provides a transparent end-to-end (host-to-host) path between two users and ensures data integrity. This layer does not depend on the particular network used. The transport layer handles message sequencing, host-to-host error and flow control and addressing between user processes. It also optimizes the usage of network resources by minimizing costs.

5. Session Layer: The session layer is concerned with the setting up and the termination of user sessions between the user application programs on the previously established paths and controlling the dialog flow. This layer may also provide error recovery without terminating the session or disconnecting the data path. Finally it provides synchronization between user processes.

6. Presentation Layer: The presentation layer provides the user application processes with a variety of data transformation services which avoid the necessity of
changing the data representations for each new session. Such services include formatting, code conversion, graphics, text compression, encryption and virtual terminate coding. Such services are generally done by user library routines.

7. Application Layer: The application layer contains the particular user application programs for accomplishing the user's objectives. It includes such services as privacy and authentication, network virtual definition, remote file access, message transfers, logging and accounting, data base access and allocation of certain centralized or distributed resources. In many network architectures this layer is left entirely or partially to the user to define and use.

Now that the framework of the OSI Reference Model has been defined, the design of data link and of the network layer will be discussed in detail in the following two sections. The data link and network layers are the fundamental tasks for the network simulation.

Design of the Data Link Layer

The design of the data link layer is mainly concerned with services provided to the network layer, formation of frames from a stream of data, transmission error control, synchronization between the sender and the receiver, i.e., source node and destination node. All these functions are
accomplished by data link layer protocols.

The data link layer provides the network layer with various services which vary from protocol to protocol. Two main kinds of services are connectionless service and connection-oriented service. With connectionless service, no connection is established before a frame is sent by the source node. If the destination node does not have to acknowledge the source whether a frame has been safely received or not or it has been lost, this type of connectionless service is called unacknowledged connectionless. Otherwise, it is called acknowledged connectionless. The sender knows whether the frame has arrived at the receiver correctly and also whether it has arrived on time. If an error occurs or the expected time expires, the sender sends the frame again. Many LAN data link protocols are of connectionless type, for example the CSMA/CD protocol.

With connection-oriented service, connection is established on both sides before a frame is sent. The standard OSI service primitives between the network layer and the data link layer are REQUIRE, INDICATION, RESPONSE and CONFIRM. The network layer on the source node uses REQUEST to ask the data link layer to establish or release a connection or to transmit a frame. The network layer on the destination node uses INDICATION to indicate that the corresponding required function has been done on the data link layer, then it uses RESPONSE to reply to the sender's
request. The sender uses CONFIRM to check whether the request was successfully or unsuccessfully carried out. Many WAN data link protocols are of connection-oriented type, for example the X.25 protocol. The relationships between the four primitives are shown in Figure 5.

![Figure 5. The Relationships Between Four Service Primitives.](image)

As a stream of data is transmitted on a link through the physical layer, it can not be guaranteed to be error free. In order to diminish a transmission error, the data link layer breaks the stream of data into a number of discrete frames. Each frame may contain a header indicating the start of the frame, a trailer indicating the end of the frame, an address field containing the addresses of the source node and the destination node in the network, a
control field containing the sequence number of that frame in the stream and acknowledgment, a checksum field having a sequence of codes used to detect and correct bit errors in transmission, and a Data field containing one part of the broken data or nothing in acknowledgment. Techniques used to generate Checksum vary from protocol to protocol. Some use Hamming code, some use CRC code and some use a variations of well known error-detection and correction codes. The frame format in the HDLC protocol is shown in Figure 6.

<table>
<thead>
<tr>
<th>Bits 8</th>
<th>8</th>
<th>8</th>
<th>20</th>
<th>16</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Address</td>
<td>Control</td>
<td>Data</td>
<td>CRC Checksum</td>
<td>Trailer</td>
</tr>
<tr>
<td>01111110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01111110</td>
</tr>
</tbody>
</table>

Figure 6. An Example of a Frame Format in HDLC Protocol.

In order to guarantee that frames are reliably transmitted from the sender to the receiver, the receiver needs to inform the sender with a feedback about the transmission, i.e., with acknowledgment mentioned earlier. If a frame successfully arrives on the destination node, the receiver sends back a positive acknowledgment, otherwise a negative acknowledgment. Such an acknowledgment is carried out by a special control frame. If the sender receives a negative acknowledgment, then it should send the previous frame again.

For error control, the sender also sets a timer to
determine whether a frame will arrive on time at the destination node and an expected acknowledgment will be sent back on time after the frame is sent. If the sender does not receive any acknowledgment when the timer expires, it should send the previous frame again, assuming that the frame has been lost.

Another important consideration of the data link layer design is the synchronization between the sender and the receiver, which guarantees that the sender sends frames no faster than the receiver can receive incoming frames, i.e., the sender runs faster than the receiver. Two main generic protocols are available for the synchronization. One is the stop-and-wait protocol, another is the sliding window protocol. With the stop-and-wait protocol, the sender sends a single frame and then stops and waits for the acknowledgment from the receiver. The advantage of this protocol is that it is very simple, but it cannot take full advantage of the network's capabilities. With the sliding window protocol, a window is used to identify the frames that the sender has transmitted but the receiver has not yet acknowledged. When the sender receives an acknowledgment for the frame that has been sent, the window advances and a new frame is exposed to send. Figure 7 shows a sliding window protocol with the window size of four. The sliding window in (a) means that the first four frames have been sent and the sender is waiting for their
acknowledgments. The sliding window in (b) means that after the sender receives the acknowledgments for frame 1 and 2 two other frames have been sent. A sliding window protocol can be found in TCP/IP and some other protocols.

![Diagram](image)

Figure 7. A Sliding Window Protocol.

The main issues in the design of the data link layer have been discussed so far. The next section will describe the design of the network layer.

Design of the Network Layer

The network layer is concerned with transmitting packets from the source node to the destination node on the subnet. Compared with the data link layer which transmits frames from a sender to a receiver only on a single link, the network layer aims on the end-to-end transmission from one node to the other on the subnet through some intermediate nodes. The transmission may require many hops along the way from the source to the destination. Data transmit-
ted on the network layer are organized as packets which are like frames on the data link layer but encapsulated with the network addresses of the source and destination and also other control information.

In this end-to-end transmission, the design of the network layer mainly deals with defining the services provided to the transport layer, determining appropriate routes/paths on which packets are transmitted from the source to the destination, whether they are in the same network or different ones, controlling congestion on paths to avoid performance degradation, and interconnecting networks.

Like the services provided by the data link layer, the network layer has connectionless and connection-oriented services. The principles of these two different services are the same. In the connectionless service, a connection between the source and the destination nodes is not established before the packet is sent and the destination may or may not acknowledge the source that whether the packet has been successfully received. In the connection-oriented, a connection is set up and the destination should reply to the source with an acknowledgment. The connectionless service can be found in the IP part of TCP/IP and in other network protocols. The connection-oriented can be found in X.25 and others. The OSI provides primitives of the two kinds of network layer services (Tanenbaum, 1989).
For examples, N-CONNECT.REQUEST, N-CONNECT.INDICATION, N-CONNECT.REQUEST and N-CONNECT.CONFIRM primitives are applied to set up a connection; N-DATA-ACKNOWLEDGE.REQUEST and N-DATA-ACKNOWLEDGE.INDICATION primitives are used for the destination to reply when a packet has been received.

In the connection-oriented subnet, a connection is called a virtual circuit, and packets follow the connection/route which is established before transmission. In the connectionless subnet, packets can be, independently of predecessors, routed on different routes which are not set up in advance, and these packets are called datagrams.

According to whether the subnet is connection-oriented or connectionless, routing decisions should be made either before transmission or on the way a packet is being sent from the source to the destination. The routing algorithm plays an important role in the network layer protocol to determine which outgoing link the incoming packet will be sent onto. The design of the routing algorithm should guarantee that all packets will be transmitted correctly, economically, optimally and efficiently. There are two major classes of routing algorithms which are nonadaptive and adaptive respectively. In nonadaptive algorithms, routing decisions are made without the consideration of the current traffic and topology of the network; while in adaptive algorithm, routing decisions are made with the consideration of the current traffic and changed topology.
The typical approaches used in nonadaptive routing algorithms can be shortest path routing and multipath routing, and those in adaptive routing algorithms can be centralized routing, decentralized routing, distributed routing, etc. Sometimes nonadaptive algorithms are called static routing algorithms and adaptive algorithms are called dynamic routing algorithms. All these routing algorithms have been discussed in Tanenbaum's book (Tanenbaum, 1989). One shortest path algorithm will be described in detail below to demonstrate routing protocols in the network layer. Other algorithms will be briefly introduced.

The idea of the shortest path routing algorithm is to find the shortest path between two nodes of a subnet when a packet is going from node \( i \) to node \( j \). The measurement of the shortest path can be the number of hops (links) between two nodes, the total geographic distance (length) between two nodes, or the total communication cost between two nodes. The algorithm can be implemented by several existing shortest path algorithms. One well-known algorithm due to Dijkstra is adapted to centralized computation (Green, 1982).

The algorithm is a step-by-step procedure to find the shortest paths from a single node to all other node. By the \( k \)th step, the shortest paths to the \( k \) nodes closest to the source node have be found. At the \( (k+1) \)th step, a new node among the remaining nodes other than the previously
chosen $k$ nodes is found to be the closest to the source. Formally, suppose $N$ is a set of nodes of a subnet, $C$ is a set containing the chosen nodes, $l(i,j)$ is the length of the link between node $i$ and node $j$ ($l(i,j) = \infty$ if no link exist between $i$ and $j$), $D(n)$ is the distance from the source to node $n$ along the shortest path, and $n_0$ is the source node. The algorithm is presented below:

1. Set $C = \{n_0\}$, and set $D(v) = l(n_0,v)$ for each $v$ in $N-C$.

2. While $C \neq N$ do the following:

   Find a node $w$ in $N-C$ for which $D(w)$ is a minimum and $C = C \cup \{w\}$.

   Update $D(v)$ for the remaining nodes in $N-C$ by computing $D(v) = \min\{D(v), D(w) + l(w,v)\}$.

Using the above algorithm to the network shown in Figure 8(a), the computation procedures are given in Figure 8(b), the shortest path tree is resulted in Figure 8(c), and a routing table for initial node 1 is presented in Figure 8(d). (A similar but different example can be found in Green’s book (Green, 1982).) The routing table is used for each node to indicate which outgoing link a packet arriving at that node should forward. The algorithm can also be applied to calculate the shortest path if the initial node is defined to be any other nodes. Here, the links in the network are supposed to be full-duplex and have the same length in the both directions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 8. An Example of Application of the Shortest Path Routing Algorithm.

In the shortest path routing algorithm, it is assumed that there exists only a single shortest path between the source and the destination. Many networks may have several
shortest paths between them. By splitting the traffic over these multiple paths, the load on each of communication links can be reduced and therefore performance of the network can be improved. If the routing protocol uses this technique of transmitting packets over multiple routes from the source to the destination, the algorithm is called multiple routing algorithm.

The shortest path routing and multipath routing algorithms are typical static routing algorithms, because they require the network with the static topology and rarely changed traffic to set up an unchanged routing table for each node once. If links or switch nodes go down and come back up, or the traffic varies frequently, the two routing protocols will meet problems. The centralized routing algorithm, however, solves the problem of a changing network. It has a routing control center which collects global information about the entire network and finds the optimal routes for each pair of nodes based on the collected information. The centralized routing algorithm can establish a new routing table as the network topology and traffic change. So it is an adaptive and dynamic algorithm.

In contrast with the centralized routing algorithm, the decentralized routing algorithm makes routing decisions based on the local information of the corresponding node but not the entire knowledge of the network. An example of
the decentralized routing is the isolated routing which forwards the incoming packet onto the outgoing link with the smallest link queue length.

The centralized routing and decentralized routing are two extreme cases in the adaptive algorithms. The distributed routing algorithm is the mixture of the centralized and decentralized, in which each node exchanges routing information with each of its neighbors and then builds the routing table.

So far, various protocols have been carefully discussed because they are the most important part in the network layer. The other significant protocols in the subnet cover congestion control protocols and internetworking protocols. Several strategies for controlling congestion and deadlock have been developed, for example, preallocation of buffers in the intermediate switch nodes on the way from the source to the destination, packet discarding if no buffer to store it on the node, isarithmic congestion control limiting the number of packets in the subnet, etc. (Tanenbaum, 1989). The typical internetworking protocols are X.75 for WANs and IEEE802 family for LANs (Tanenbaum, 1989; Yakubaitis, 1986).

Remark

This chapter introduced the concepts of network architectures and the design of data link and network
layers. Since many protocols have been proposed in the design of data link and network layer for different kinds of networks, only those important parts are discussed. Those preliminaries will be used for the design of the generic simulation model of the network in the following chapter.
CHAPTER III

SIMULATION MODEL OF COMPUTER COMMUNICATION NETWORKS

The computer communication network has been defined in terms of OSI seven layer architecture. Modeling a network system is to combine network components representing specific nodes, topology, protocols etc., in a generic framework so that the required architecture is obtained. Such a framework is called a simulation model of networks. This generic model should be flexible, because if network components are dynamically characterized the simulation can make evaluation of performance for distinct network systems with collection of those interleaved components.

Reviewing the communication network structure in Figure 2 in Chapter II, a network consists of hosts and a subnet composed of switch nodes and communication links. If two users of two hosts communicate with each other, message will be transmitted through the way, host-to-node, node-to-node and node-to-host. By the definition of seven conceptual layers, node-to-node transmission is carried out on the subnet and it is obviously the task of the network layer and the below layers, whereas host-to-node and node-to-host transmissions can be fulfilled by the transport layer as host-to-host (end-to-end) control. Host-to-host
control may include functions breaking messages into packets, sequencing messages, etc. But node-to-node transmission on the subnet deals with formatting and sequencing packets, defining an optimal route from the source node to the destination node, error detection and recovery, congestion, etc. It is the major task for the network designer because the subnet is the backbone of the network. Therefore node-to-node communication and subnet architecture will mainly be modeled for the network simulation. However, host-to-node transmission is simply assumed to be message breaking and packet arriving at the subnet and node-to-host transmission is simply supposed to be packet leaving from the subnet and message reforming.

In modeling the network, three important components are considered. These are switch node, topology and protocols. A node serves to store and forward packets. Topology defines interconnection of nodes. Protocols determine strategies controlling packet transmission from the source node to destination node. They are interrelated and interactive. The methodology of modeling these components is discussed in the following sections.

Modeling a Switch Node

In a store-and-forward network, a switch node performs the functions of storing an incoming packet and forwarding it onto an outgoing communication link.
Figure 9. The Model of a Node.

To store the packet, the node puts it in a transmission buffer. The packets in the buffer wait for the processor to serve them in order of arriving time or priorities. To forward the packet, the node determines whether it has arrived at the destination node or it needs to go to the next intermediate node on the way from the source to the destination. If the node is not the packet's destination, the processor forwards the packet to a link on a chosen optimal route from the source to the destination. If it is, the packet is transmitted out of the subnet to the host or it is simply discarded if node-to-host transmission is not considered. That processor is usually called a server in the simulation. Hence, a node is composed of two components in the simulation model: a server and a queue. Figure 9 illustrates the model of a
node.

This node can be called a single-server queuing system in terms of queuing theory. The queuing discipline may utilize first-in-first-out (FIFO) protocol or priority protocol, whether the size of the queue is defined to be infinite or finite. In the FIFO protocol, the order of storing and forwarding an incoming packet depends on when it arrives at the node. In the priority protocol, the order depends on the degree of significance, magnitude or urgency of the packet. For example, the packet with an acknowledgment may have higher priority than one with an acknowledgment. Therefore the first waiting packet in the queue should be the one first arriving at the node or with the highest priority among all incoming packets according to the application of the protocols. The server processes the first packet in the queue when it finishes serving the last packet. The server is always in one of two states: BUSY and IDLE, if the possibility of the server's going down is not considered. BUSY means that the server is working for the packet; while IDLE means that the server has done job and is waiting for a coming packet.

In the analysis of the computer network by queuing theory, an incoming packet is not cared where it comes from, i.e., whether it is from the host or from some other node. In this simulation model, however, packets will be distinguished into two classes. One is the packet from a
host, and the other is the packet from a node. The packet from a host connected with the node is one of pieces of a splitted message which is randomly generated in the modeling of host-to-end transmission, because a user sends messages to another user at random intervals through our observations. But the packet from another on the subnet is not a random one, because the performance of that packet will depend on the topology and protocols of the subnet and the performance of other packets on the subnet after it has gone into the subnet. In order to classify these two kinds of arriving packets on the node, a packet from a host is called PACKET_H and a packet from a node is called PACKET_N.

By statistical studies, the process of messages' arriving a node from a host is a stochastic process which has the following properties (Law & Kelton, 1982):

1. Messages arrive one at a time.
2. The number of arrivals in the interval \((t, t+s]\) is independent of the number in the period \([0,t]\), where \(s\) is the interval length.
3. The distributing of the number of arrivals in \((t, t+s]\) is independent of \(t\) for all \(t, s \geq 0\).

Therefore, the process of message generation is a Poisson process because of the above three properties (Law & Kelton, 1982). Actually, the Poisson process is the most commonly used model for the arrival process in a queuing

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
system. With the Poisson process, exponentially distributed interarrival times can be computed if the interarrival rate has been defined, that is when a message will be generated can be known. For the detailed theory about Poisson process, a general reference may be (Kleinrock, 1975). Thus, for each node in the simulation model, there is a random generator which generates messages in Poisson process.

Modeling Network Topology

\[
P = \begin{bmatrix}
p_{00} & p_{01} & \cdots & p_{0N} \\
p_{10} & p_{11} & \cdots & p_{1N} \\
\vdots & \vdots & \ddots & \vdots \\
p_{N0} & p_{N1} & \cdots & p_{NN}
\end{bmatrix}
\]

Figure 10. One-step Transition Probability Matrix P.

As discussed in Chapter II, switch nodes on a communication network are connected in patterns of star, ring, tree, mess, etc. The connection of nodes combined with corresponding link length, capacity and cost is topology of the network. The model used to represent the network topology is that of a Markov process with \( N+1 \) discrete states. Each node of the network represents one state.

The Markov model is represented by a one-step transi-
tion probability matrix $P$ shown in Figure 10. The entry $P_{ij}$ is the probability that a packet at node $i$ is sent to node $j$, where $P_{ij}$ $\geq$ 0. Since $M$ is a probability transition matrix, each row will sum to unity, i.e., $P_{i} = 1$, for all $0 \leq i \leq N$. Therefore, the connectivity of network topology can be implicitly described by $P$. The links will only exist from nodes $i$ to nodes $j$ where $P_{ij} > 0$. If a matrix $C$ represents the connectivity of the network, each entry $c_{ij}$ is defined to be 0 or 1 to indicate whether there is a connection/link from node $i$ to node $j$, where a link between node $i$ and node $j$ can be full-duplex or not.

$$c_{ij} = \begin{cases} 
0 & \text{No connection between node } i \text{ and } j. \\
1 & \text{A connection in the direction of from node } i \text{ to node } j.
\end{cases}$$

An example of representing a network shown in Figure 11 in the form of a connectivity matrix $C$ is given in Figure 12.

In modeling network topology, lengths and capacities on links should also be represented as the part of the topology model. This can be done in the same way as a connectivity matrix $C$ representing connections nodes in the network. A link length matrix $LL$ is defined to represent lengths on corresponding links, i.e., the entry $ll_{ij}$ will equal 0 if no connection exists between node $i$ and node $j$ or length of the link if there exists a link from $i$ to $j$. A link capacity matrix $LC$ can also be defined with $c_{ij}$ equals 0 or capacity on the link from $i$ to $j$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 11. An Example of Network Topology With Full-duplex Links.

\[
\begin{array}{ccccccc}
\text{node}_i & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 \\
2 & 1 & 0 & 1 & 1 & 1 & 0 \\
3 & 1 & 1 & 0 & 0 & 1 & 0 \\
4 & 1 & 1 & 0 & 0 & 1 & 1 \\
5 & 0 & 1 & 1 & 1 & 0 & 1 \\
6 & 0 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

Figure 12. Connectivity Matrix \( C \) Representing Topology in Figure 11.

Network topology is taken to be an integral part of the routing protocol when a routing table is built on each node as it has been illustrated in the last chapter. All routing tables on the network can be combined together into a big table containing all routing information. This table is defined to be a routing matrix \( R \). The entry \( r_{ij} \) of \( R \) represents the next node that a packet goes to on its way...
from source node \( i \) to destination node \( j \). Reconsider the network shown in Figure 8(a); the routing matrix \( R \) of that network can be obtained by computing each routing table for each node and then putting them together. If the shortest path routing algorithm is applied, the routing matrix \( R \) is resulted in as shown in Figure 13. \( r_{ij} = 0 \) means that a packet has arrived at its destination node.

Now that the models of node and network topology have been defined, the model of protocols will be proposed in the next section where the previously discussed models will be effectively used.

\[
\begin{array}{ccccccc}
\text{source} & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 0 & 2 & 3 & 2 & 2 & 2 \\
2 & 1 & 0 & 3 & 5 & 5 & 5 \\
3 & 1 & 2 & 0 & 2 & 2 & 2 \\
4 & 5 & 5 & 5 & 0 & 5 & 5 \\
5 & 2 & 2 & 2 & 4 & 0 & 6 \\
6 & 5 & 5 & 5 & 5 & 5 & 0 \\
\end{array}
\]

Figure 13. Routing Matrix \( R \) for the Network in Figure 8(a).

Modeling Network Protocols

As declared previously, the proposed generic simulation model should be flexible in order to fit any (or at worst most) kinds of usages of network and data link layer protocols including connection-oriented or connectionless, sliding window, routing and flow and congestion controls.
To maximize flexibility, no specific protocols will be described in the generic model, but the generalized methodology suitable for various protocols will be suggested during modeling. It has been known that node-to-node transmission will be focused on in simulating a communication network on the network layer and below. The reason and importance are that the network and data link layer protocols are concerned about the way and control of packet transmission from a source node to a destination node. So the node-to-node transmission along with its protocols is especially observed and investigated in order to establish the model.

The node-to-node transmission can be modeled by three phases: preparation, transmission and reception. The preparation phase is to manage the queue for packets to be sent and retransmit, schedule an optimal route, compute expected acknowledgment arriving time, perform sending window protocols, etc. The transmission phase is to seize and free link, change link and node status, detect and correct error and collision, estimate link transmission time, etc. The reception phase is to generate acknowledgment, perform receiving window protocols, etc. These three phases are discussed in detail in the following.

In the preparation phase, arriving packets whether they are PACKET_Hs or PACKET_Ns are stored in the queue of the node $i$ in order of arriving times or priorities. When
the server of node $i$ becomes IDLE, the first packet in the queue is going to be transferred. Before transmission, the outgoing link, i.e., the next intermediate node should be determined. This can be done by choosing the next node $j$ for node $i$ in the routing matrix. For example, in Figure 13, the next intermediate node of node 2 is 5 if the destination node is 4. If a dynamic routing algorithm is applied, the routing matrix or routing table for node $i$ should be recomputed before selection. If a connection-oriented protocol is used, a connection from the source to the destination must have been established, so that the routing link has already been determined.

Before transmission, the sending packet will be temporarily stored in a retransmission buffer in case that the routing packet is lost or an error occurs during transmission and a retransmission is needed. To determine whether the packet should be retransmitted or not, an expected acknowledgment arriving time will be estimated. That time is generally called round trip time (RTT) which covers transmission time from node $i$ to node $j$, transmission time from node $j$ back to node $i$ and queueing delay in node $j$. If acknowledgment is not received within the RTT, the packet will be retransmitted. The RTT is one of important dynamic network characteristics, because underestimating the RTT will lead to waste of retransmission of each packet twice and therefore increase network load, and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
overestimating the RTT will result in the degradation of efficient utilization of network resources. Some typical measurements for RTT can be found in (Malamud, 1992).

Recall that most network and data link layer protocols belong to sliding window protocols. Network throughput is increased if the size of sending and/or receiving window are increased. When the sender overwhelms the receiver and the receiver runs out of buffers, the sender should stop sending packets. When the network is congested, the sender must decrease its transmission rate by shrinking the sending window size. Therefore the window size is also one dynamic characteristic which should be considered in the network evaluation. In the preparation phase, this size may be estimated according to protocol applied.

When a packet is ready to send from node $i$ to node $j$ as determined in the preparation phase, the link from node $i$ to node $j$ is checked to see if it is BUSY or IDLE in the transmission phase. Note the link from node $i$ to node $j$ is different from one from node $j$ to node $i$ if the communication link between $i$ and $j$ is full-duplex. If the link is IDLE the packet can be transmitted on it. This time the server in node $i$ is released, i.e., set to IDLE so that the next packet in the queue can be processed for preparation. The status of the applied link is changed to BUSY in order to prohibit the next packet from being on the link simultaneously. If the link is BUSY the processed packet in the
preparation phase waits until it becomes IDLE.

In modeling the transmission phase, some error activities may be created in order to mimic the performance of a real network, because a network cannot be perfect when it is being designed. These errors can be packet lost, code error, collision of packets on the link, etc. They may be generated randomly or in other ways. Then any error should be detected in the transmission and corresponding correction should be scheduled. For example, if two packets collide, they can be sensed and will be retransmitted on the link but the order of transmission will be determined so that no collision will happen again for these two packets. Different protocols have different solutions to correcting errors.

In the transmission phase, the time needed for transmission of the packet from node \( i \) to node \( j \) is computed. With it, when the next node \( j \) will receive the packet can be determined. The computation can be implemented with the length of packet, the capacity of the link and length of the link.

In the reception phase, when node \( j \) receives a packet, it allocates a buffer to it, and it modifies the receiving window if a window protocol is applied. The first received packet is analyzed by the server if it is IDLE. It checks whether there is any error in the packet. Then it generates an acknowledgment containing information whether the packet
has been successfully received or it is needed to retransmit if an error occurs. The server also checks whether the received packet is an acknowledgment from node \( i \) to which it sent a packet before. If it is, the packet stored in the retransmission buffer in the preparation phase should be removed.

It will be proven in the next chapter that the methodology discussed above for the node-to-node transmission can be efficiently applied to modeling specific protocols with the established generic simulation model. With the above analysis, discrete event for the simulation program can be easily defined. This will be discussed in the next chapter as the simulation program is designed.

Before concluding the discussion on the modeling, however, some significant measures of performance of the network are defined in the last section of the chapter. The simulation program will collect statistical information and provide estimates of the network performance.

**Measures of Performance**

The final goal of the network simulation is to provide estimates of the network performance as the different dynamic characteristics and starting parameters are assigned in the generic simulation model. The optimal network topology and protocols can therefore be obtained after analyzing performance estimated on proposed network
architectures. These important measures of network performance can be:

1. The queue length: the length of queues needed to store packets at various nodes.

2. The queuing delay: the time length between the time a packet arrives at a node and the time it is served.

3. The service delay: the time length between the time a packet enters a server and the time it is sent onto a link, which is directly proportional to packet size and node speed.

4. The node transmission delay: the time length between the time a packet arrives at a node and the time it leaves from the node, i.e. the queuing delay service delay.

5. The link transmission delay: the time length between the time a packet is sent on a link and the time it arrives at the next node, which is directly proportional to packet size and link length and inversely proportional to link transmission capacity.

6. The total delay: the time length between the time a packet arrives at the source node from a host and the time it leaves from the destination node.

7. The link throughput: the number of packets transmitted on a link per unit time.

8. The total link utilization: the fraction of the total time during which the link is being used to successfully transmit a packet.
9. The number of transmitted packets: the number of packets sent more than once because acknowledgment is not received in the RTT.

10. The number of packet collisions: the number of packets transmitted on a single link simultaneously.

The above measures of performance are typical characteristics for evaluation of a network. Some other measures, such as link failure, overhead, etc., can also be used as parameters in the simulation.
CHAPTER IV

SEQUENTIAL DISCRETE EVENT NETWORK SIMULATION PROGRAM

As introduced in Chapter I, the simulation program applies three utilities: events, event list and simulation clock. An event time describes activities of the simulated system. It contains a time stamp called event time indicating when the event occurs. The event list stores all future events in order of event times. The simulation clock denotes how far the simulation has advanced. The main iteration of the simulation program repeatedly removes the first event with the smallest event time from the event list and processes it. These events happen at discrete points in time in a real network system, so they are referred to as discrete events, while the simulation driven by discrete events is called discrete simulation. A typical flow control of the discrete event simulation was given in Figure 1. The design of the network simulation program can follow the illustration in the figure.

When the simulation program is designed, the discrete events of the network system need to be defined based on the generic simulation model proposed in the last chapter. The goal of this chapter is to develop the modules of these events and the structure of the network simulation program.
The discrete event simulation program described in this chapter is designed on a conventional computer, which is said to be the sequential discrete event simulation program, in contrast with the parallel discrete event simulation on a parallel computer.

In implementing this simulation program, an object-oriented programming (OOP) language, C++, is applied for developing discrete event simulation on networks. The appropriateness of using OOP for network simulation program is also discussed in the chapter.

Discrete Events of Network Simulation

Discrete events play an important role in simulation, because events describe changes of states of the network and the simulation clock is advanced to the current event time. For instance, a message is going to a node from a host. This activity of the network can be called an event, for which the current operation of the simulation is on a node-to-host transmission based on the proposed simulation model. If a message departs from a node to a host, this event can represent the activity of node-to-host transmission. The discrete event simulation changes from one activity to another according to the occurrence time of future events. Suppose the last event occurred at 5:05 p.m. and the future events will occur at 5:07 p.m., 5:10 p.m., 5:15 p.m., etc. The event of 5:07 p.m. occurs next
and the simulation clock is changed to 5:07 p.m. These future events ready to happen were scheduled by previous events and are saved in an event list in order of occurrence time. Consider the event that a packet is sent from node \( i \) to node \( j \), it must schedule an event that the packet arrives at node \( j \), including the event time and insertion of the event in the event list.

According to the generic simulation model, transmission of a message from a host to another host is modelled by host-to-node, node-to-node and node-to-host transmission respectively. The node-to-node transmission is divided into three phases, preparation, transmission and reception. At the conception of a discrete event, changes of network states happen in these transmission procedures respectively. Therefore, the corresponding events representing the network operation can be defined to be MESSAGE ARRIVAL, PACKET ARRIVAL, PACKET RECEPTION, PACKET SENDING and MESSAGE DEPARTURE.

The MESSAGE ARRIVAL event randomly generates a message and implements a host-to-node transmission; the PACKET ARRIVAL, PACKET RECEPTION AND PACKET SENDING events describe node-to-node transmission covering the three phases; and the MESSAGE DEPARTMENT event fulfills node-to-host transmission.

These events can occur on each node of the network. In order to distinguish different events occurring on
different nodes, event types can be defined to represent them, which are given in Figure 14, where N is the number of nodes of the simulated network.

The modules of these events and structure of the simulation program based on them are discussed in the next section.

<table>
<thead>
<tr>
<th>Event</th>
<th>Event description</th>
<th>Event type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message arrival i</td>
<td>Arrival of a message at node i</td>
<td>i</td>
</tr>
<tr>
<td>Packet arrival i</td>
<td>Arrival of a packet at node i</td>
<td>i+N</td>
</tr>
<tr>
<td>Packet reception i</td>
<td>Reception of a packet at node i</td>
<td>i+2N</td>
</tr>
<tr>
<td>Packet sending i</td>
<td>Sending of a packet from node i</td>
<td>i+3N</td>
</tr>
<tr>
<td>Message departure i</td>
<td>Departure of a message from node i</td>
<td>i+4N</td>
</tr>
</tbody>
</table>

Figure 14. The Definition of Event Type.

Network Simulation Program

By the flow control of discrete event simulation illustrated in Figure 1, the network simulation program can follow the routines, initialization, timing, event and report, where the timing and event routines iterates until the simulation is over. In the initialization routine, the simulation clock is set to zero, the network system states are initialized, i.e., SERVER=IDLE, LINK=IDLE, etc., and the event list is initialized to have a MESSAGE ARRIVAL.
event as the first necessary event to start the simulation. In the timing routine, the first event in the event list is taken as the current event to occur, then the simulation clock is set to that event time. In the event routine, corresponding operations about this event are performed, such as system state updating, statistical information collection and future event scheduling and insertion into the event list. The timing and event routines keep the network operation evolving until enough statistical information is gathered. They are the core of the simulation program. After the simulation is over, estimates of the network performance can be made in the report routine, so that evaluation is obtained for the analysis and design of the network.

In order to program the simulation in modules, the structure representing the control of the timing and event routines is illustrated in Figure 15, followed by the description of five event modules. The description here is used for the experiment but the specification for each module can be modified according to what protocols and topologies are applied or being designed. This agrees with the idea that the proposed network simulation is capable of testing and analyzing various networks.

(1) MESSAGE ARRIVAL $i$
   - Randomly generate the message length in exponential distribution;
Operation depends on the chosen next event

- Randomly generate the destination node \( j \) the message goes to from the source node \( i \) in uniform distribution;
- Schedule the next MESSAGE ARRIVAL \( i \) event by Poisson process;
- Break the message into a sequence of packets, where each packet contains a sequential number, packet size, source node, destination node and generation time;
- Put the packets in queue \( i \) in order of sequential number;
- If queue \( i \) is EMPTY, then schedule a PACKET
RECEPTION $i$ event;
- Gather statistical information.

(2) PACKET ARRIVAL $i$
- Set link$_i$ to IDLE;
- Put the arriving packet at node $i$ at the end of the queue $i$;
- Decrease the receiving window size;
- If queue $i$ is EMPTY, then schedule PACKET RECEPTION $i$ event.
- Gather statistical information.

(3) PACKET RECEPTION $i$
- If server $i$ is IDLE, then:
  set server $i$ to BUSY;
  Remove the first packet in queue $i$;
  Increase the receiving window size;
  If the packet is a data packet reaching the destination node, then:
    Save it in the message buffer $i$;
    If all packets of a message have arrived, then:
    Schedule a MESSAGE DEPARTURE event;
  If the packet is a data packet not reaching the destination node, then:
    Schedule a PACKET SENDING $i$ event.
  If the packet is an acknowledgment, then:
    Remove the corresponding packet from the
retransmission buffer \( i \);
Increase the sending window size;
Set server \( i \) to IDLE.
-Gather statistical information.

(4) PACKET SENDING \( i \)
-If server \( i \) is IDLE, then:
  Set server \( i \) to BUSY;
  Choose an optimal outgoing link \( j \) by the
  routing algorithm applied;
  If link \( j \) is IDLE, then:
    Estimate the RRC;
    Keep the packet in the retransmission
    buffer with the retransmission time (RRC +
    simulation clock time);
    Compute link transmission delay;
    Schedule a PACKET ARRIVAL \( j \) event at node \( j \);
    Set link \( j \) to BUSY;
    Decrease the sending window size.
  If link \( j \) is BUSY, then:
    Schedule a PACKET SENDING \( i \) event for that
    packet;
  If the simulation clock \( \geq \) retransmission time of
  a packet in the retransmission buffer, then:
    Schedule a PACKET SENDING \( i \) event;
  Set server \( i \) to IDLE.
-Gather statistical information.
(5) MESSAGE DEPARTURE i
- Sequence the received packets reaching the destination node i to form a message.
- Remove them from message buffer.
- Gather statistical information.

Now that the five event modules have been defined for the experiment, another critical decision is to choose an appropriate language to implement the simulation. The following section will discuss the topic.

Network Simulation Implementation via OOP in C++

In the process of developing a validated and useful simulation, one important decision is the selection of an efficient computer language to implement the simulation program. Traditionally, simulations were implemented in general purpose programming languages such as FORTRAN, Pascal, C, ALGOL, PL/1 and BASIC, or in special purpose simulation languages such as SIMUSCRIPT, SIMAN, SLAM, SIMULA, GASP, and GPSS. Although special simulation languages provide a relatively fixed set of procedures to carry out the simulation automatically, a general purpose language FORTRAN is still the most popular language used for the simulation (Eldredge, McGregor & Summers, 1990). It is attributed to the fact that the choice of a language primarily depends on the availability of the language to programmers and programmers' knowledge of the language,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
rather than the appropriateness or performance capability of the language (Eldredge, McGregor & Summers, 1990).

Languages such as FORTRAN, Pascal and C, all share a common approach to solving a problem, that is using the procedural paradigm. Consider the design of the discrete event network simulation program in this thesis. Using the procedural paradigm, the design would focus on the decomposition of the overall loop into subtasks. These subtasks are implemented in functions or subroutines. But the individual functions are not systematically related to one another. A function may invoke or be invoked by other functions. Beyond that, individual functions have no defined relation to one another. When the specification for each subtask is defined, the relation of functions is fixed. But as the specification is changed, the established relation may totally be changed such that a programmer must re-make each function and relation. This procedural paradigm does definitely not fit the flexibility of the proposed network simulation, because different and various protocols are tested in the process of designing and analyzing communication networks using the generic simulation model. Therefore, a new paradigm called object-oriented programming (OOP) is applied to the network simulation.

Consider the design of a network simulation with OOP. The majority of the design will focus on the description of
entities in the simulation domain such as queue and server, rather than algorithms for solving the problem such as data structure for the queue. Each entity is defined abstractly in terms of class. A class is roughly a type definition of a complex record which includes member functions as well as member data. The actual entities are represented as instances of those abstract classes. The instances are implemented as objects. These object classes are relatively orderly and well defined. They can have well-defined relationships with one another by inheritance. Inheritance is the relationship that allows the programmer to use an existing class definition as the basis for a new definition. The inheritance hierarchy of object classes provides a much higher level of capability than individual functions. Therefore, application of OOP in network simulation will help define the structure of the design and develop network analysis on the generic simulation model.

C++ is an object-oriented superset of the standard C programming language. It was conceived over a decade ago, but reliable and efficient commercial C++ compilers have only recently been available. It is both a modern object-oriented language and well-known ordinary procedural language. This provides the advantages: easy transition to OOP for programmers familiar with C; easy incorporation of existing functions into the environment of C++ objects and classes; and efficiency in making, compiling and executing

However, OOP has not been widely used in simulations. It is one of the purposes of this thesis to explore the use of an OOP language, C++, for implementing the discrete event network simulation. This development is expected to demonstrate the capabilities of OOP and stimulate the widespread use of OOP in the simulation community.

The classes defined in the simulation describe the general properties of the network system as the generic simulation model requires, while the instances of the classes implement the specific properties of the example network as the specific protocols and topologies are defined for the generic model. The defined classes and the description of the role of each class in the network simulation is listed below.

1. Msg_Generator: determines the probability distribution for arriving message and randomly generates them.
2. EventNotice: defines the attributes of an event.
3. EventList: deletes the next event from and inserts future events into the event list.
4. Queue: defines the attributes of a packet stored in the queue of a node and the data structure of the queue.
5. Server: provides service to packets, including the reception and sending of packets.
6. Net_Environment: determines the network's condi-
tions and operations for the simulation, including all events.

An important difference of OOP is the inheritance hierarchy of classes. The arguments that an object needs from other objects in lower levels of the hierarchy are transmitted through the paths from the lower levels to the object. Increasingly specialized knowledge about the object’s action is in lower levels, while the basic idea reflecting properties common to a number of classes is at the top of the hierarchy. Figure 16 presents the inheritance hierarchy of the classes in the implemented simulation program, followed by an outline of the class definitions in C++. A network simulation program was implemented in C++ on SUN Station in author’s Simulation class, CS518.

Figure 16. The Inheritance Hierarchy of the Class in the Network Simulation Program.

```cpp
class Net_Environment : Msg_Generator, Sim_Run, Server
```
{  
private:
  float simulation_terminate_time;
  int topology[][];
  int capacity[][];
  int routing[][];
  servers server_state;
  links link_state;
  int queue_state;
  float next_event_time;
};

class Net_Environment
{
private:
  void simulate(); //member function to schedule first
                  //and start simulation run.
  void simulation_terminates(); //member function to check if the
                                //simulation should be terminated.
  void message_arrival(); //member function to implement five
                           //events.
  void packet_arrival();
  void packet_reception();
  void packet_sending();
  void message_departure();
};

class Msg_Generator
{
private:
  int source, destination, message_size;
  float generate_time;
};

class Sim_Run: EventList
{
private:
  float sim_clock;
public:
  Sim_Run(); //constructor.
  void run(); //member function to drive simulation
               //run.
  float now(); //current simulation time.
};

class EventNotice
{
private:
  float event_time;
  event event_type;
  int event_node;
};
int source, destination, packet_size, packet_seq;
float generation_time;
EventNotice* next;  //to link event notices.

public:
    EventNotice();    //constructor.
friend EventList    //let class EventList modify
    //EventNotice.
};

class EventList
{
private:
    EventNotice* first, last //point to the first and last
        //of the event list.

public:
    EventList();    //constructor.
    void remove();  //remove the current event from
                   //the eventlist and execute it.
    void insert();  //insert the scheduled future
                    //event into the event list.

};

class Queue
{
private:
    float arrival_time,
        generation_time;
    int source, destination,
        packet_size, packet_seq;
    Queue* next;    //point to the next packet in
                    //the queue.

public:
    Queue();        //constructor.
    friend Server_Queue()    //let class Server modify Queue.
    //
};

class Server
{
private:
    Sim_R* sim;    //let Server access Sim Run.
    Queue* first[], last[]; //define state variables and
               //statistical variables.

public:
    Server();      //constructor.
    void append();  //append and arrival packet at
                    //the end of queue.
    void reception(); //remove the first packet in the
                     //queue and implement reception
                     //process.
    void sending();  //send the packet to the next node.

}
Remark

This chapter has discussed the sequential discrete event network simulation program and its implementation in OOP environment. The introduced methodology is quite useful for the network simulation. But the following problem still remains. Because of the sequential execution of the simulation program (not OOP), it takes a long execution time on the execution to collect enough statistical information for the performance evaluation. This problem motivated the introduction of parallel simulation on networks which is a new field in simulation. The next chapter will focus on this subject.
CHAPTER V

PARALLEL DISCRETE EVENT SIMULATION ON NETWORKS

Parallel discrete event simulation deals with the execution of a single discrete event simulation program on a parallel computer in order to decrease the long execution time of the simulation program (Fujimoto, 1990; Misra, 1986; Mouftah & Sturgeon, 1990). There are two major problems in the parallel discrete event simulation: distribution of processes on processors of a parallel computer, and synchronization of these processes to guarantee the necessary sequence of events occurring. These two problems also exist in the parallel discrete event simulation on computer networks. In this chapter, two solutions to the problems are proposed for the network simulation via applications of a hypercube and a variant Time Warp scheme with partial cancellation respectively.

PDES Problems

In contrast with parallel discrete event simulation, the discrete event simulation described before is called sequential discrete event simulation (SDES). In the execution of a SDES program, the first event with smallest event timestamp is always selected from event list as the
next occurring event. Only one event is currently executed in the processor of a conventional computer. With the PDES, however, the execution of a simulation program should be decomposed into a set of processes which can be concurrently executed in local processors of a parallel computer. Each processor is referred to as a logical process (LP). Therefore, Problem1 arises, that is, how parallelism in the simulation program exists, how to partition the simulated state variables into a set of disjoint states and ensure that an event in LP_i least accesses states of other events in LP_j, and how to distribute processes in the parallel computer applied.

With the sequential execution paradigm, it is guaranteed that event E_i is executed before the event E_j where i<j and i and j are event timestamps. If it is possible that E_j is executed before E_i and that E_j can modify state variables used by E_i, this will result in an error that the future can affect the past. This kind of error in the simulation is called a causality error. But this causality error may occur in PDES. Assume LP_1 has E_i with the event timestamp 5 and LP_2 has E_j with timestamp 10, LP_1 and LP_2 are executed concurrently, and E_j finishes as E_i finishes. If E_i schedules a new event E_3 with the timestamp 7 for LP_2 and E_3 needs to modify state variables used by E_j, then the causality error in LP_2. Therefore, Problem2 arises, that is, how to synchronize the activities of LPs and ensure the
necessary sequence of events.

These two general PDES problems obviously exist in the parallel network simulation too. In the following two sections, two solutions focusing on the network PDES are especially proposed.

Hypercube Architecture for the Solution to PDES Problem1

Problem1 is concerned about the decomposition of a network simulation program into a set of LPs and distribution of LPs on parallel computer. Consider the inherent parallelism in the operation of a network. The solution to the first part of Problem1 is straightforward. It is natural to create on LP for each server of node in the simulation network. Therefore, if the simulated network has $n$ servers of nodes, the PDES will contains $n$ logical processes $LP_0, LP_1, \ldots, LP_{n-1}$. These $n$ LPs exactly correspond to the $n$ nodes on the topology of the network. The remaining part of Problem1 is how to distribute the $n$ processes on $n$ nodes in a parallel computer.

There are many different types of parallel computers. They are generally differentiated by the interconnectivity of processors, the control mechanism between processors, shared memory or distributed memory, etc. The most popular parallel architecture is a hypercube which has been playing a significant role in the development of parallel processing (McHugh, 1990; Wager, 1988; Zhou & Kountanis, 1992).
Here, this chapter concentrates on a special case study of the distribution of \( n \) processes on a hypercube for the network PDES with efficient communication between processes.

**Hypercube**

An \( m \)-dimensional hypercube is a multiprocessor consisting of \( 2^m \) nodes that are linked together in an \( m \)-dimensional binary cube network. Specifically, a 0-dimensional hypercube is just a single point, a 1-hypercube is a line segment, a 2-hypercube is a square, a 3-hypercube is regular cube, and, in general, an \( m \)-dimensional hypercube \( H \) consists of two \((m-1)\)-dimensional hypercubes \( H_1 \) and \( H_2 \) and \((m-1)^2\) links between the corresponding nodes on \( H_1 \) and \( H_2 \). A 3-hypercube with 8 nodes is shown in Figure 17 (Zhou & Kountanis, 1992).

![Figure 17. A 3-Hypercube With 8 Nodes.](image)

Each node in \( H \) is given a unique \( m \)-bit identification number, which is referred to as the node address. The node
addresses are assigned inductively as follows: (a) for $m=1$, the nodes are numbered 0 and 1, and (b) for $m>1$, append a 0 to the front of $H_1$'s node addresses and a 1 to the front of $H_2$'s node addresses.

From the above definition, it immediately follows, that two nodes are neighbors if and only if their corresponding addresses differ in exactly one bit.

**Distribution of Network Simulation LPs in a Hypercube**

It has been defined that each LP corresponds to the behavior of one server of node in the simulation network. When a server works, it may schedule a future event for some other nodes on the network. Suppose in LP$_i$, the server is processing a PACKET SENDING event. It schedules a new event PACKET ARRIVAL for LP$_j$ of the neighboring node $j$ if the packet is being transmitted to node $j$. There is a communication between LP$_i$ and LP$_j$ through link$_{ij}$ of the simulated network. LP$_i$ sends a message to LP$_j$ and LP$_j$ receives a message. This kind of information exchange is defined to be message-passing between two LPs. Assume that message-passing exists on each link between two LPs on two immediately connected nodes on the simulated network. Therefore, the LP structure, which represents the relationship of message-passing of LPs, is equivalent to the topology of the simulated network according to the assumption above. If the LP structure can be embedded in a
hypercube, the distribution of LPs must be optimal, because message-passing between two LPs can be done without interrupting other processes’ progresses.

There are two cases of embedding an LP structure onto a hypercube: an exact embedding and an approximate embedding.

Figure 18. An LP Distribution With an Exact Embedding of the Given LP Structure Onto a Hypercube.

The first case is that if the LP structure of the simulated network is an exact subset of the connectivity of a hypercube, it is straightforward to distribute \( n \) LPs in the corresponding nodes of the hypercube. For example, given an LP structure with ring topology shown in Figure 18(a), \( LP_0, \ldots, LP_7 \) can be obviously distributed on node 000, 010, 011, 001, 101, 111, 110 and 100 respectively, and each link of the ring exactly corresponds to each marked links in Figure 18(b). This kind of distribution is
referred to as an LP distribution with an exact embedding of LP structure onto a hypercube.

Figure 19. An LP Distribution With an Approximate Embedding of the Given LP Structure onto a Hypercube.

The second case is that the LP structure of the simulated network is not an exact subset of the connectivity of a hypercube, but all message-passing between two neighboring LPs in the structure can be done through 0 or more intermediate hypercube nodes. A intermediate node is not allocated an LP and is only used to transit message. This is defined to be an LP distribution with an approximate embedding of LP structure onto a hypercube. For example, given an LP structure with complete topology in Figure 19(a), LP₀, LP₁, LP₂ and LP₃, can be distributed on node 000, 001, 010 and 100 respectively. The links of message-passing for each LP pair are marked dark in Figure

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Each link is either an immediate connection of two nodes or a connection with intermediate nodes which do not have an LP. For instance, the link between LP₀ and LP₁ is an immediate connection, whereas the link between LP₃ and LP₄ has an intermediate node 101 on the hypercube. Observe the two cases. The LP distribution with exact embedding is a special case of that with an approximate embedding.

It can be proven that n LPs with any message-passing structure can be distributed on an (n-1)-hypercube with an approximate embedding.

Lemma: Given n LPs with a complete message-passing structure can be distributed on an (n-1)-hypercube with an approximate embedding.

Proof: Use induction to prove the lemma.

Basic: For n=2, 2 LPs can be distributed on a 1-hypercube with an exact embedding.

Induction step:

Induction hypothesis: Assume the lemma holds for n=k-1, i.e. k-1 LPs with a complete structure can be distributed on an (k-2)-hypercube with an approximate embedding.

Prove the lemma holds for n=k:

A (k-1)-hypercube is composed of two (k-2) hypercubes H₁ and H₂ with 2ᵏ⁻² links between the corresponding nodes of two hypercube. By the assumption, distribute k-1 LPs, LP₀, ..., LP_k₋₁.
on $H_1$ with an approximate embedding. Allocate $LP_{k+1}$ on a node of $H_2$ and connect $LP_{k+1}$ with $LP_0, ..., LP_{k-2}$ by an immediate link or a connection with intermediate nodes which are not assigned an LP. Therefore, the distribution of $k$ LPs is done with an approximate embedding.

Lemma 2: Given $n$ LPs with any message-passing structure, they can be distributed on an $(n-1)$-hypercube with an approximate embedding.

Proof: Any message-passing structure of $n$ LPs is a subset of a complete message-passing structure of $n$ LPs. By Lemma 1, $n$ LPs with a complete message-passing structure can be distributed on an $(n-1)$-hypercube with an approximate embedding. If we remove those connections between LPs in the complete structure which are not in the given structure, the resulting structure is the given message-passing structure. Therefore, the lemma holds.

A Variant Time Warp Scheme for the Solution to PDES Problem 2

In the previous section Problem 1 of distribution of LPs was solved by applying a hypercube architecture. In this section, Problem 2 is solved, which is concerned with the synchronization of LPs to guarantee the necessary sequence of occurring events in distributed LPs.

There are mainly two mechanisms to solve the problem:
conservative and optimistic (Fujimoto, 1990). Conservative approaches strictly avoid the possibility of any causality error occurring. With these approaches, the decision should be made whether it is safe to process an event, i.e., it is guaranteed that all events which should affect the event have been processed before it can be processed. On the other hand, optimistic approaches allow causality errors occur, but they use a detection and recovery strategy. Causality errors are detected and event sequence is recovered by a rolling back method. The main idea of the method is to go back to the point of time after which the error occurs and re-simulate from that past time. In this chapter, a variant Time Warp scheme with partial cancellation is suggested for the network PDES based on Jefferson’s Time Warp (Jefferson, 1985).

Time Warp Scheme

It has been assumed that the simulation program is decomposed into \( n \) logical processes, \( LP_0, \ldots, LP_{n-1} \), each of which can process event independently. In each process, there is a local clock \( C_i \) which refers to the simulation time of \( LP_i \), i.e., when an event is processed, clock \( C_i \) is automatically advanced to the timestamp of the event. The local time in \( C_i \) may or may not correspond to real time and is independent of other processes's local clock times. This time is referred to as virtual time.
Jefferson's Time Warp scheme is based on virtual time, which is an optimistic mechanism for process synchronization. With the scheme, all LPs are allowed to continue their simulations. In the ideal situation, the processes execute normally occurring events in their virtual time order. In the real situation, however, a later arriving message may require a process to re-simulate a previous part of the simulation, if the sending time of message is behind the present time of the receiver. The re-simulation can be implemented by rolling back the LP's state and undoing its effects on other processes. Therefore, Time Warp means that LPs in the parallel simulation can move forward and backward according to virtual times.

In the Time Warp scheme, there are two major control mechanisms to synchronize processes: a local control mechanism and a global control mechanism.

As introduced above, in the local control mechanism, each process LP maintains a local virtual clock $C_j$ which is independent of other LPs' clocks. All actions of LP$_j$ are in consequence with $C_j$. However, a global control mechanism is needed in order to synchronize LPs. As we know, when a message is received by an LP, if the sending time of the message is in the past time of receiving LP, it requires a rolling back of the LP. It may modify the LP states and/or it may send message to some other LPs to unsend previously sent messages. Rolling back the states
can be done by periodically saving the LP's states and restoring the old states at the point of that sending time. Unsending previously sent message can be done by sending negative-message which cancel the original ones when they reach other related LPs. Those original message are referred to as positive-message. If an LP receives a negative-message whose positive-message has already been processed, that LP should also be rolled back to undo the effect of the positive-message. This procedure is recursively repeated until all erroneous effects are canceled.

In the global control mechanism, a global virtual time (GVT) defined by Jefferson keeps the minimum time of all LPs' virtual times. The reason is due to the memory constraints of a real computer system. In order to roll back states and unsend message, a track of past states and messages passed to other LPs must be saved. Keeping this information needs extraordinarily large amount of memory buffers, if no useless information is discarded as the simulation goes on. Using the global control mechanism with its GVT, any messages and any states stored in buffers which have a time less than the GVT can be discarded because no event can occur before this GVT. The strategy to update the GVT depends on the balance of the number of stored past states and message versus the times the system updates the GVT. Some algorithms were proposed for computing the GVT (Fujimoto, 1990), but they are beyond the
A Variant Time Warp With Partial Cancellation for Network PDES

By Jefferson's Time Warp scheme, an LP should roll back to the previous state and unsend previously sent messages immediately after it receives a positive-message whose sending time in the past time of the LP. All events that the LP scheduled and processed after that sending time will be completely canceled whether the computation is correct or not. All positive-messages that the LP sent will be completely canceled whether they have had or will have no incorrect effects on another LP or not. This kind of cancellation can be called a complete cancellation.

However, analyzing the properties of the network PDES, it was found that a complete cancellation may not be necessary, because some events in LP are invariant to any other occurring events. For example, LP_i receives an arriving message with the sending time T less than LP_i's current time T+t. LP_i scheduled MESSAGE ARRIVAL events in the interval [T,T+t], some of which occurred in [T,T+t] and some of which will occur after T+t. For those MESSAGE ARRIVAL events which were processed in [T,T+t], LP_i could have processed their PACKET RECEPTION events and scheduled the corresponding PACKET ARRIVAL events for LP_j that packets routed to. Analyze the property of MESSAGE ARRIVAL

scope of this thesis.
event which simulates the host-to-node transmission. A message coming from a host to node \( i \) is invariant to any events from other nodes. Therefore, even though MESSAGE ARRIVAL events are canceled as \( LP_i \)'s local time rolls back to \( T \) from \( T+t \), these events will exactly re-occur again, and those scheduled PACKET ARRIVAL events for \( LP_j \) will exactly re-occur again in \( LP_j \) if they are unsent. Should those events in \( LP_i \) and messages passed to \( LP_j \) be canceled? No, it is not necessary to cancel them, because no incorrect effects are caused by those events and message-passing. The only thing to do is that those events and messages should be marked so that they will not be canceled as \( LP_i \) rolls back and sends negative-message and they will not be recomputed. But for those packets arriving at \( LP_i \) from an intermediate node of the network, their scheduled events PACKET RECEPTION in \( LP_i \) and PACKET ARRIVAL in \( LP_j \) may have to be canceled because of incorrect computation in \([T+t]\).

Therefore, the above proposed method by which only some of the events and messages in \([T+t]\) are canceled and some are not canceled as \( LP \) rolls back to \( T \), is defined to be a variant Time Warp with partial cancellation. Compared with Jefferson's complete cancellation, the proposed scheme can save much computation time on each \( LP \) which does not re-compute some events and message with no incorrect effects on itself and others.
To end the discussion, a flow control of an LP for the synchronization is presented in Figure 20 on the next page. By the figure, the main properties of network PDES can be inferred and comparison can also be made with SDES in Figure 1.
Figure 20. A Flow Control of an LP for Synchronization.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHAPTER VI

CONCLUSION

In this thesis, the state-of-the-art discrete event simulation for the design and analysis of computer communication networks has been developed and described. The particular focus has been on the methodology, modeling and implementation of sequential and parallel discrete event simulation on networks. These proposed techniques will be quite beneficial and useful for those network designers and analysts who work on the network layer and the below layers of ISO/OSI Reference Model in the evaluation of the network performance. To conclude the thesis, the research done so far is reviewed and future research discrete are identified below.

In the first part of the thesis, it was demonstrated that the traditional performance measurement methods are not suitable for the analysis of today's computer communication networks with large size, high speed and great complexity, but the discrete event simulation technique is suitable for it. The flow control of the general discrete event simulation was studied as the preliminary. The importance of a parallel simulation was also mentioned in order to overcome the disadvantage of a sequential simula-

81
tion. Then the focus went to the simulation concepts to computer network study. The seven layers of the ISO/OSI Reference Model was reviewed, and the design of the network layer and data link layer was especially discussed because it is the basic and fundamental for the definition of the proposed generic simulation model.

In the second part of the thesis, a generic simulation model was proposed. The model is flexible to model any store-and-forward communication networks only by defining the specification of the network being designed or analyzed. The focus was on modeling of nodes, network topologies and protocols respectively. Then based on the generic model, the discrete events of the simulation was depicted. In implementation of the simulation, an object-oriented programming technique was introduced to the simulation community. The class definitions of network simulation in C++ were described to enhance the understanding of inheritance hierarchy of classes in the network simulation. By the defined model and the classes with hierarchy, a generalization of the network simulation was obtained.

In the third part of the thesis, the parallel discrete event simulation was studied to solve the problems in the parallel simulation were proposed, the focus was directly onto the solutions to the problems for the network simulation. A special parallel architecture, the hypercube was
applied to solve the problem of process distribution as a special case study. Then a variant Time Warp scheme with partial cancellation was proposed to solve the problem of process synchronization. The flow control of a parallel discrete event simulation was depicted in order for the comparison with that of a sequential one.

Throughout the thesis, the research has endeavored to keep the current and future research direction. It is believed that one major tendency of the analysis and evaluation of computer networks is to diminish the gap between the researchers who design networks and engineers who use networks. The issue of the network management system has been addressed (Dupuy, Schwartz, Yemini & Bacon, 1990; Wolfson, Sengupta & Yemini, 1991). The network management will combine the techniques in network design, network simulation, graphical interface and management information base, in which network researchers and administrators both participate. It is believed that the other major tendency of the network evaluation is the further application of parallel computation techniques in the simulation. Because if a general and efficient solution to the parallel simulation problem can be found, it in fact leads toward a new development of parallel processing as a whole. Here, a hypercube for the parallel network simulation is greatly recommended because of the suitability of network simulation and embedding on a hypercube and the
possibility of implementation (Steiz, Seizovic & Su, 1988).


Chirlian, P. M. Programming in C++. Columbus, OH: Merrill, 1990.


Trenary, R. A Note from CS518, Computer Simulation and Modeling. Kalamazoo, MI: Western Michigan University, Fall 1990.


