MESSAGE RESPONSE TIME ANALYSIS OF CAN-BASED CONTROL SYSTEMS

By

LIANG-WEI HO

Bachelor of Science

Tamkang University

Taipei, Taiwan

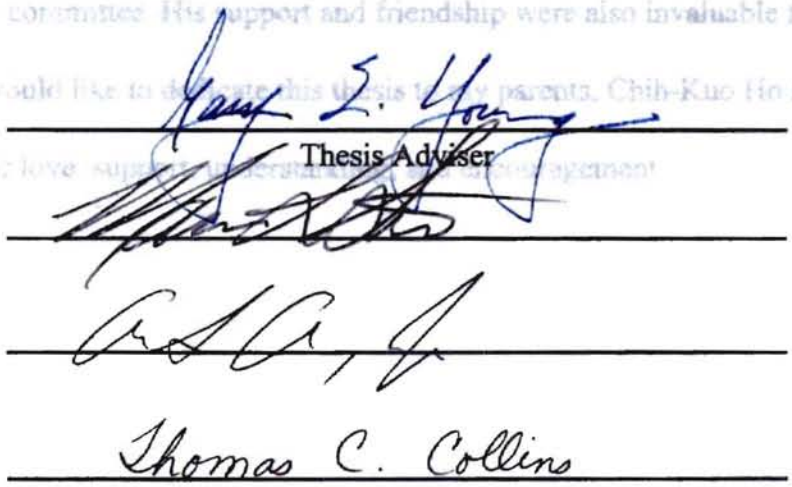
1993

Submitted to the Faculty of the
Graduate Collage of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1997

MESSAGE RESPONSE TIME ANALYSIS OF

CAN-BASED CONTROL SYSTEMS

Thesis Approved:



Thesis Adviser

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to express my appreciation to my major advisor, Dr. Gary E. Young, for his advice and encouragement throughout my graduate study. Special thanks go to Dr. Marvin L. Stone for meticulously evaluating my thesis. His professional comments and suggestions were very helpful for my research work.

More over, I wish to express my sincere gratitude to Dr. Andrew S. Arena, Jr. for serving on my committee. His support and friendship were also invaluable for me.

Finally, I would like to dedicate this thesis to my parents, Chih-Kuo Ho and Hsiu-Lien Weng, for their love, support, understanding, and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Problem background

The continuing decline in the cost of computers and microprocessors has resulted in more and more microprocessor-based controllers being used for control systems. They usually do not work alone, but are connected by a transmission medium to share the system resource, transmit the important messages and save the cost of cabling. This is called Integrated Communication and Control System (ICCS). Complex control systems like spacecraft and autonomous manufacturing plants require a high-speed and reliable communication bus to carry information between system components. These system components including microprocessors, intelligent terminals, sensors, controllers, and actuators are connected by a communication network and work in a real-time environment. Unfortunately, the network always introduces delays which may degrade the system performance and may even make the system unstable. A diagram of an ICCS network in Figure 1.1 shows how these delays are introduced into a control system. Figure 1.2 illustrates how the network-induced delays enter the control system[1].

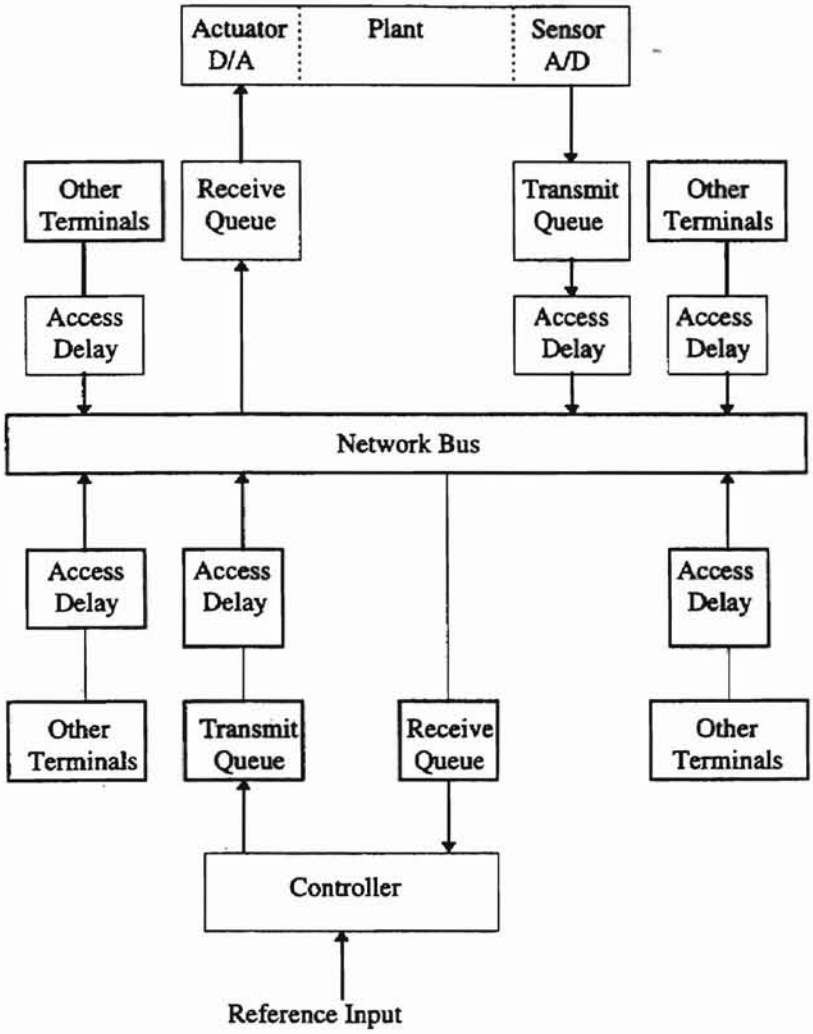


Figure 1.1 Schematic diagram for the ICCS

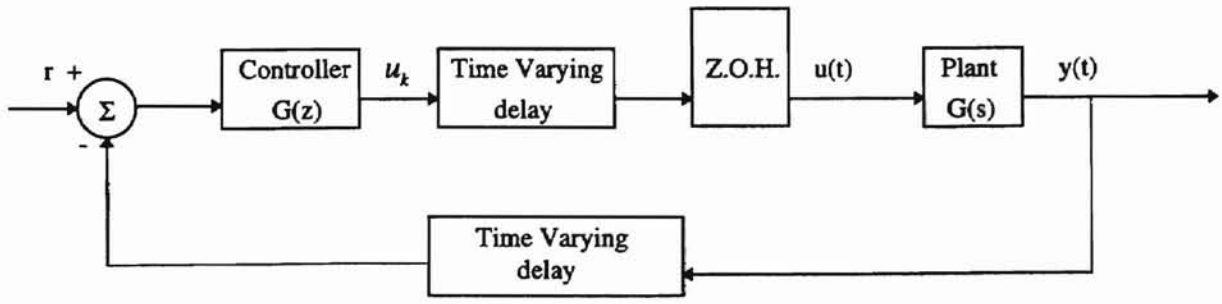


Figure 1.2 Delayed control system

Controller Area Network(CAN)

Controller Area Network, a type of Local Area Network(LAN), was originally developed by a German company, Robert Bosch, during the late 1980's for use in the car industry to provide a cost-effective communication bus for in-car electronics. CAN bus is a serial data communication bus designed for sending and receiving short real-time control signals. Using CAN, controllers, sensors, and actuators communicate with each other in a real-time environment at speeds up to 1 Megabits per second over a two wire serial data bus. A very efficient priority-based Media Access Control (MAC) protocol is being used to solve the message collision problems and provides not only the best channel utilization, but also very short message delays for higher priority messages, which makes it the best choice for use in real-time distributed control systems. The top 6 reasons for CAN to receive high attention are as follows.

1. It has excellent error detection and confinement capabilities.
2. It is cost effective to design and implement.
3. It is easy to configure and modify.
4. It will continue to operate in harsh environments.
5. It automatically detects data transmission errors.
6. It provides an environment that enables the centralized diagnosis of faults during design or in-service.

Because of its proven reliability and robustness in harsh real-time environments, CAN is now an international standard and is documented in ISO 11898 (for high-speed applications) and ISO 11519 (for lower-speed applications) [2].

Literature Review

Many research work in modeling and simulation of communication protocols have been published [4,6] and some research work regarding the significance of network-induced delays relative to the stability of feedback control systems can be found[3,5]. However, most of the literature on delayed systems considers only the case of constant delays. For the network-induced time-varying delays, a more complete treatment is given by Asok Ray[7,8].

Although it has been only 10 years since CAN was invented, many CAN research papers are published[9-15,18-19]. In the following sections, we briefly review some significant CAN research work, including a message priority assignment algorithm, a simplified idea for message delay analysis, and a general model for calculating the message response time of CAN.

Message priority assignment algorithms

Some research of message priority assignment algorithms can be found[15,16]. One significant work is the paper published by Wang, which presents an algorithm to produce an optimum message priority assignment for a given control system[15]. The basic principle behind this algorithm is that higher priorities should be assigned to the messages which are transmitted less frequently and have shorter message length. The priorities are, first, assigned to the messages according to the "short job first" principle. Then, the adjustments are made to make priorities of all messages meet their deadlines.

The algorithm is illustrated by the following simplified example. Table 1.1 gives 6 sets of messages in a control system. Each one of them has its own transmission time and

allowed maximum delay. Please note that the time unit used in this example can be bit time or any time unit.

Message name	Transmission time	Allowed Maximum delay
A	400	500
B	500	2400
C	50	850
D	600	3100
E	700	1900
F	50	300

Table 1.1 Real-time constraints of a network-based control system

Let t_i be the transmission time of the i-th priority message, D_i is the maximum delay of the i-th priority message in the worst case, TD is the sum of the maximum delays of all messages, and AD_i is the allowed maximum delay of the i-th priority message. Then,

$$D_i = \sum_{j=1}^{i} t_j < AD_i \tag{1.1}$$

and

$$TD = \sum_{i=1}^{n} D_i = \sum_{i=1}^{n} \sum_{j=1}^{i} t_j \quad \text{be minimized} \tag{1.2}$$

where n is the number of messages in the system. Please note that a smaller index represents a higher priority. The basic principle is to find out the optimal priority assignment to meet Equation (1.1) and at the same time it will minimize Equation (1.2).

By the "short job first" principle, the list of the message priorities is

F	C	A	B	D	E
50	50	400	500	600	700
300	850	500	2400	3100	1900

The numbers in the second row represent the transmission time of each message and the numbers in the third row represent the allowed maximum delay of each message. The second step is to apply Equation (1.1) and check the list from left to right.

For message F , $D_1 = 50$ and $AD_1 = 300$ $D_1 < AD_1$ Equation (1.1) is satisfied.

For message C , $D_2 = 100$ and $AD_2 = 850$ $D_2 < AD_2$ Equation (1.1) is satisfied.

For message A , $D_3 = 500$ and $AD_3 = 500$ $D_3 = AD_3$ Equation (1.1) is not satisfied.

So, we shift message C to the right of message A, which results in the following list.

F	A	C	B	D	E
50	400	50	500	600	700
300	500	850	2400	3100	1900

Apply the algorithm again, we find the following.

From message A to D, Equation (1.1) is satisfied, but, for message E, Equation (1.1) is not satisfied. So we shift message D to the right of message E.

F	A	C	B	E	D
50	400	50	500	700	600
300	500	850	2400	1900	3100

Now, we find all messages meet their minimum requirements. The priorities of all messages should be assigned as the above list. From the left to the right, message F should have the highest priority and message D should have the lowest priority. The algorithm selects messages to be shifted from right to left, beginning with the left neighbor of the current message. If the message on the leftmost has been selected and failed, there is no priority assignment that will meet the deadline requirements.

Message delay analysis of CAN

Wang presented a method to analyze message delays of CAN data frames[9]. The method is based on the following assumptions.

1. Message generation is approximated by the Poisson distribution.
2. Message length is exponentially distributed.
3. There are only data frames on the CAN bus.
4. The network bus is error free and no message retransmission is considered.

By these assumptions, the expected waiting time for a message with the i-th priority to be transmitted is the sum of the expected time for the current message occupying the bus to finish, the expected time for transmitting all higher priority messages waiting in the queue, and the expected time for transmitting higher priority messages generated during

the waiting time. The maximum delay of the i-th priority message will occur under the following conditions.

1. The ith priority message is generated with all higher priority messages at the same time.
2. There is a longest message on the bus, which just sent the first bit of its data frame and seized the bus.
3. During the waiting time, the higher priority messages are continuing to be generated at their maximum rates.

So, the maximum delay of the i-th priority message can be presented by the following equations.

$$Max.D_i = W_i + F_i \tag{1.3}$$

$$W_i = (F_{\max} + INTFS - 1) + \sum_{K=0}^{i-1}(F_k + INTFS) + \sum_{k=0}^{i-1}(F_k + INTFS) \times M_k \times \frac{W_i}{B} \tag{1.4}$$

where

$Max.D_i$: The maximum delay of the i-th priority message.

$F_{\max}$: The frame length of the longest message which seized the bus. (Bit time taken to transmit the longest message which occupied the bus.)

$INTFS$: The maximum interframe space between two consecutive frames.

M_k : The maximum message generation rate of the k-th priority message.

W_i : The maximum waiting time for the i-th priority message.

B : The bandwidth of the bus.

F_i : Bit time taken to transmit the message itself.

Please note that the time unit, bit time, used to express the message delay is the time needed to transmit one bit on the network. It is the inverse of the bus bandwidth.

Calculating message response time of CAN

Tindell presented a model to analyze the message response time of CAN in 1994[13]. Based on his research, the worst-case response time of a given message with priority m is the longest time between the queuing of the message and the time the message arrives at destination nodes(stations). It is composed of two delays : the queuing delay and the transmission delay. The queuing delay is the longest time that the message can be queued in a station and be delayed because the lower priority message seized the bus and other higher priority messages are being sent on the bus. It is composed of two parts : the longest time that any lower priority message can occupy the bus, and the longest time that all higher priority messages can be queued and occupy the bus before the message m is transmitted. The transmission delay is the time taken to transmit the message itself.

Assume the period of a given message m is denoted as T_m, the width of the queuing window for message m is denoted as J_m, and τ_{bit} is the time taken to transmit a bit on the CAN bus. To cover the error handling and message retransmission, the message response time (R_m) can be represented by the following equations.

$$R_m = t_m + C_m \tag{1.5}$$

$$t_m = B + \sum_{j \in hp(m)} \left\lceil \frac{t_m + J_j + \tau_{bit}}{T_j} \right\rceil \times C_j + E(t_m + C_m) \tag{1.6}$$

where

t_m : The probable queuing delay.

C_m : The transmission delay for the message with priority m.

B : The longest time that any lower priority message can occupy the bus.

C_j : The transmission delay for the message with priority j.

$E(t_m + C_m)$: A probable bound on the error recovery overheads before a message m arrives at the destination.

Objective of This Study

From our literature review, we find most of the CAN research covers only part of the CAN protocol or they are too general to be useful. For example, Wang presented a general idea to describe the message delays. However, his model is based on the assumption that only data frames exist on CAN bus. The oversimplified model is not good enough to represent the real network behavior because we know that transmission errors exist in real networks. Tindell presented a network model of describing the CAN message response time. It is the sum of the propagation delay because of the transmission of the message itself and the queuing delay. The queuing delay is the sum of the longest time that any lower priority message can occupy the bus, the time to transmit all higher priority messages which are queued and ready to be sent during the message response time, and the corresponding error recovery overheads including message retransmission, the transmissions of error frames, and the interframe spaces. However, his model doesn't mention any details of each term, especially, the last term; error recovery overheads. This work can't help us analyze a real CAN-based control system because we have no idea

how the system parameters are related to the message response time. Being engineers of control systems, we are more interested in determining the appropriate system parameters for a given CAN-based control system to achieve acceptable performance requirements.

Priority assignment is another important topic for setting up CAN-based control systems because CAN guarantees the data latency of the highest priority message only. Wang's "short job first" is a good and reasonable algorithm. However, it may not fit all control systems because priority assignment is a system dependent problem.

From the real-time control application point of view, the following questions need to be addressed.

1. How should the transmission rate (or sampling period) of each message (sensor signal, control signal, etc.) on the CAN bus be chosen ?
2. How should the priority of each message transmitted on the CAN bus be assigned ?
3. How are the system parameters(sampling rates, priority, network speed, number of stations, etc.) related to the data latencies ?
4. Under what kind of conditions will the network fail to meet real-time constraints ?

This research investigates CAN protocol and the questions mentioned above. Based on Tindell's work [13], we extended the model of CAN message response time to be more specific and well-suited for the analysis of real-time control systems. We also developed some equations to calculate the message response time by using the system parameters including the message sampling rates, their priorities, the probability of error occurring during each message transmission, and the probability of sporadic signals occurring on the CAN bus. In Chapter II, we will review the CAN specification 2.0. In Chapter III, the

extended model and equations are derived and presented. Chapter IV shows how the simulation results verify the model, together with analysis and discussions. Chapter V follows with conclusions and the recommendations for further research.

CHAPTER II

OVERVIEW OF CAN

CAN vs. OSI model

ISO 7498 defines a communication standard known as Open System Interconnection (OSI). It describes how communications occur between computers on any network. Anyone that conforms to the standard can communicate with anyone else that conforms to the same standard. The OSI model defines 7 different layers as shown in Table 2.1 [21]:

Application	
Presentation	
Session	
Transport	
Network	
Data link	Logical Link Control(LLC)
	Media Access Control(MAC)
Physical	

Table 2.1 ISO/OSI model

The application layer is the highest layer which provides the interface between the user and the lower layer. The presentation layer (layer 6) handles the data formatting and code conversion (encryption & compression). The session layer (layer 5) sets up and manages the coordination of data during communications. The transport layer (layer 4) controls the sequencing of message components. The network layer (layer 3) sets up addresses and delivers message packets. The data link layer (layer 2) deals with the media access control and provides the data transfer between a node and the network. The physical layer passes the bit stream to and from the network bus (hardware components).

CAN protocol uses a layered architecture which is based on the OSI model ,but does not fully comply with the OSI model. Table 2.2 shows the layered structure of a CAN node[20]. It consists of only three layers which are the physical layer, the data link layer, and the application layer. CAN protocol does not specify any specification concerning the application layer. However, SAE J1939/71 does define some standard for the application layer. The scope of the physical layer is to define how signals are actually transmitted. It deals with the description of Bit Timing, Bit Encoding, and Synchronization[20]. There is no other detail about the physical layer specified in CAN specification 2.0, which means there is much freedom in selecting a physical layer for CAN nodes. Some standards of the physical layer can be found in ISO 11898, ISO 11519, and SAE J1939/11 & 12. The scope of CAN specification 2.0 is to define the MAC (Medium Access Control) sublayer and part of the LLC (Logical Link Control) sublayer of the Data Link Layer and to describe the consequences of the CAN protocol on the surrounding layers[20].

Application Layer	
Data Link Layer	LLC Layer -Message Filtering -Message and Status Handling
	MAC Layer -Fault Confinement -Error Detection and Signaling -Message Validation -Acknowledgment -Arbitration -Message Framing -Transfer Rate and Timing
Physical Layer -Signal Level and Bit Representation -Transmission Medium	

Table 2.2 Layered Structure of a CAN Node

The data link layer in CAN is divided into two parts, the logical link control (LLC) layer and the media access control (MAC) layer, the same as those in the OSI model. The logical link control layer, the upper part of the data link layer, is responsible for the following functions [14].

1. Frame acceptance filtering
2. Recovery management

The MAC sublayer of the data link layer doesn't perform any check on the identifier field of the received frame. It is the frame acceptance filtering function that is responsible for deciding whether or not the received frame belongs to the node. The recovery

management function handles the re-transmission of the frame when there is a transmission error or the frame loses a contention for transmission when a higher priority message is transmitted at the same time.

The media access control sublayer in the data link layer provides the following functions [14] :

1. Encapsulating the frame being transmitted
2. Decapsulating the received frames
3. Managing transmission and reception on the shared medium
4. Detecting and signaling any error which occurs during transmission and receptions

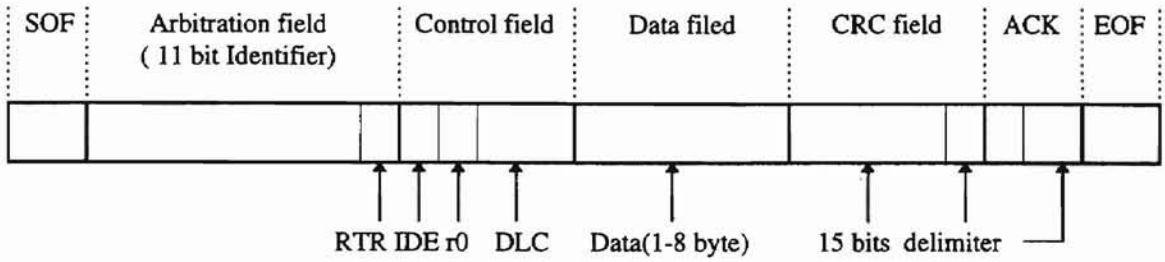
The most distinctive feature of the CAN physical layer with respect to the other networks is the set of the logical levels the bus can assume. There are two complementary logical values in CAN bus. They are called dominant and recessive, equivalent to logical value 0 and 1, respectively. The dominant bit (0) will always dominate the bus when there is more than one node transmitting dominant and recessive bits at the same time.

According to CAN specification 2.0[20], message transfer is controlled by four different frame types : Data Frame, Remote Frame, Error Frame, and Overload Frame. In the following sections, we are going to review each one of these separately.

Data Frame

CAN specification 2.0 Part B defines two message frames called the standard message frame (version 2.0 A) and the extended message frame (version 2.0 B). The 2.0 A standard message frame consists of 7 different fields. They are SOF (Start of Frame)

field, arbitration field, control field, data field, CRC (Cyclic Redundancy Code) field, ACK (acknowledge) field, and EOF (end of frame) field.

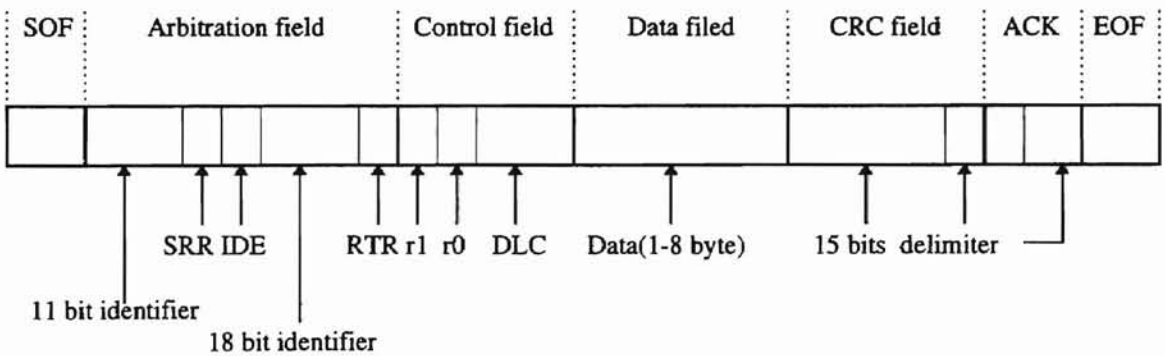


CAN 2.0 A Data Frame

1. SOF field : This field contains only one dominant bit (logic 0) that indicates the beginning of a data frame.
2. Arbitration field : This field contains an 11-bit identifier and 1 RTR (Remote Transmission Request) bit. The 11-bit identifier represents not only the name of the message, but also the priority of the message. That means 2,032 different messages can be defined in CAN 2.0A (CAN specification prohibits the most significant 7 bits from being all logic 1). A dominant RTR bit indicates that the message is a remote transmission request by one node for data from some other nodes on the bus.
3. Control field : This field contains 3 types of data, one IDE (identifier extension) bit which is dominant in the standard message frame and recessive in the extended message frame, a dominant bit (r0) which is reserved for future use, and a 3-bit DLC (Date Length Code) which indicates the number of bytes in the data field.

4. Data field : The field contains data from 0 byte for remote transmission request to a maximum of 8 bytes.
5. CRC field : This field contains 15-bit Cyclic Redundancy Code for CRC error checks.
6. ACK field : This field contains 1 ACK bit and 1 delimiter bit. The ACK bit is set to be recessive when the message frame is transmitted and is overwritten by a dominant bit transmitted from all nodes which receive the message successfully.
7. EOF field : This field consists of 7 recessive bits to indicate the end of a message frame.

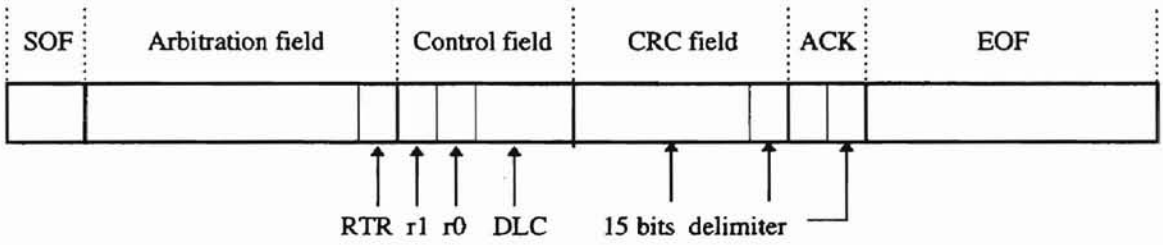
2.0 B extended message frame contains 32 bits in arbitration field including a 29-bit identifier which provides the ability to maintain over 500 million message types[10], one SRR (Substitute Remote Request) bit, and one IDE bit. The SRR is always set to be recessive in order to ensure that the standard message frame will dominate the extended message frame when both message frames have the same priority in the first 11 bits of arbitration field. Actually, this design is to make sure the 2.0 B extended message frame has 100% compatibility with the standard message frame because there are still a lot of CAN chip providers who support only 2.0 A in their products. All other fields in 2.0 B are identical to that of 2.0 A.



CAN 2.0 B Data Frame

Remote Frame

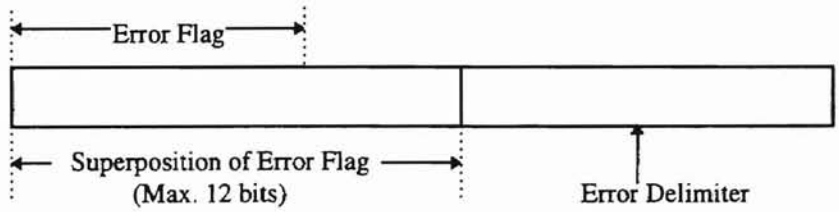
Any station acting as a receiver for certain data can initiate the transmission of respective data by sending the Remote Frame. It exists in both standard and extended format and contains six different fields : SOF, Arbitration filed, Control filed, CRC filed, ACK field, and EOF. Please note that there is no Data field needed in a Remote Frame and contrary to Data Frame, the RTR bit in Remote Frame is always set to be "recessive"[20].



CAN 2.0 Remote Frame

Error Frame

The Error Frame is composed of two parts. The first part is the superposition of Error Flags contributed from different stations and the second part is the Error Delimiter.

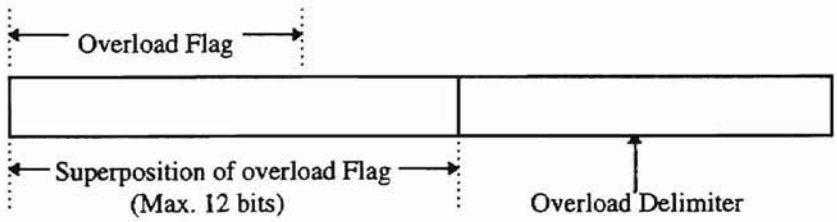


CAN 2.0 Error Frame

There are two forms of Error Flags : a Passive Error Flag and an Active Error Flag. The Passive Error Flag contains 6 consecutive 'dominant' bits and the Active Error Flag contains 6 'recessive' bits. An 'error active' station transmits an Active Error Flag whenever it detects an error condition. The Error Flag's form violates the rule of bit stuffing applied to all fields from SOF to CRC Delimiter or destroys the fields from ACK field to the EOF field. As a consequence, all other stations detect the error condition and transmit an Error Flag on their parts. So the sequence of 'dominant' bits results in the superposition of different Error Flags transmitted by individual stations. The total length of this sequence varies from a minimum 6 bits to a maximum 12 bits. An 'error passive' station transmits the Passive Error Flag when it detects an error condition. The 'error passive' station waits 6 consecutive bits of equal polarity, beginning at the start of the Passive Error Flag. The Passive Error Flag is complete when these 6 equal bits have been detected. The Error Delimiter is composed of 8 'recessive' bits. After transmission of an Error Flag, every station sends 'recessive' bits and monitors the bus until it detects a 'recessive' bit. Afterwards, it starts to transmit seven 'recessive' bits[20].

Overload Frame

The overload frame contains two fields : Overload Flag and Overload Delimiter.



CAN 2.0 Overload Frame

The Overload Flag consists of 6 'dominant' bits. The overall form corresponds to that of the Active Error Flag. Its form destroys the fixed form of the Intermission field. As a consequence, all other stations detect the Overload Flag and on their parts start transmission of an Overload Flag. Please note that if there is a 'dominant' bit detected during the 3rd bit of Intermission, it will interpret this bit as Start Of Frame, not the Overload Flag. The Overload Delimiter consists of 8 'recessive' bits. It is the same as that of an Error Frame. After transmission of an Overload Flag, the station monitors the bus until it detects a transition from a 'dominant' bit to a 'recessive' bit. At this time, every station has finished sending its Overload Flag and all stations start transmission of seven more 'recessive' bits.

The following conditions lead to the transmission of an Overload Flag[20].

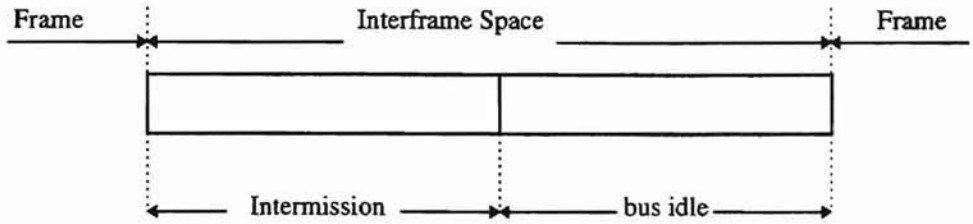
1. The internal conditions of a receiver, which require a delay of the next Data Frame or Remote Frame.
2. Detection of a 'dominant' bit at the first and second bit of Intermission.
3. When a CAN node detects a 'dominant' bit at the eight bit of an Error Delimiter or Overload Delimiter, it will start transmitting an Overload Frame.

The condition 1 is only allowed to be started at the first bit time of an expected Intermission. Conditions 2 and 3 are allowed to be started one bit after detecting the 'dominant' bit.

Interframe Space

All Data Frames and Remote Frames are separated from preceding frames (Data Frame, Remote Frame, Error Frame, or Overload Frame) by a bit field called the

Interframe Space. In contrast, the Overload Frame and Error Frame are not preceded by an Interframe Space[20].



CAN 2.0 Interframe Space

The Intermission field consists of 3 'recessive' bits and the bus idle fields may be of arbitrary length. If the bus is recognized to be free, any station having something to send can access the bus. Any message which is waiting for transmission during the transmission of another message, is started in the first bit following the Intermission. The detection of a 'dominant' bit on the bus is interpreted as a Start Of Frame.

Message Validation

A message is valid for the message transmitter if no error has been detected by the end of the End Of Frame field. If there is any error during the transmission of a message, retransmission will follow automatically and according to the prioritization. In order to compete for bus access with other messages, retransmission has to start as soon as the bus is idle. A message is valid for the message receivers, if there is no error until the last bit of the End Of Frame.

Coding

The fields, SOF, Arbitration field, Control field, Data field, and CRC field, of a Data frame or a Remote Frame are encoded by a method called bit stuffing. Whenever a transmitter detects five consecutive bits of identical value in the bit stream to be transmitted, it automatically inserts a complementary bit in the actual transmitted bit stream. The remaining bit fields in the Data Frame or the Remote Frame are of fixed form and not stuffed. The Error Frame and the Overload Frame do not perform the bit stuffing method to their bit streams.

Error Handling

CAN provides 5 types of error detection, 3 of which are at the frame level and 2 of which are at the bit level.

1. Bit Monitoring Error

   A station that is sending a message on the bus also monitors the bus at the same time. When the bit value that is monitored is different from the bit value that is sent, a Bit Monitoring Error is flagged. One exception of this rule is the sending of a 'recessive' bit during the Arbitration field or during the ACK slot. Detecting a 'dominant' bit in these fields is not considered to be a Bit Monitoring Error.

2. Bit Stuffing Error

   Right after five identical consecutive bit levels have been sent, the transmitter will automatically add one opposite bit level into the bit stream and transmit it (Bit Stuffing). The message receiver will automatically detect the stuff bit and delete it

(De-Stuff). If the receiver detects six consecutive bits with the same level, a Bit Stuffing Error is flagged.

3. CRC Error (Cyclic Redundancy Code Error)

   Each transmitted message frame contains a 15-bit CRC code. This code will be computed by using the message content before the massage frame is transmitted. Every node which receives the message will compute the CRC code by using the content of the received message based on the same algorithm, and then compare the CRC code with the one in the received message frame. If they are different, the receiver flags a CRC error.

4. Form Error

   There are some pre-defined bit values at certain points such as the CRC Delimiter, ACK Delimiter, End Of Frame bit, and Interframe Space in the transmitted message frame. If a receiver detects any invalid bit value in these positions, a frame error will be flagged.

5. Acknowledgment Error (ACK Error)

   If the message transmitter detects that the ACK slot bit in the transmitted message frame is not overwritten by a dominant bit, the ACK error is flagged.

The 3 types of error detection, CRC Error, ACK Error, and Form Error, are at the frame level and the other 2 types of error detection, Bit Monitoring Error and Bit Stuffing Error, are at the bit level.

Fault Confinement

Fault Confinement is a mechanism which provides a method for discriminating between temporary errors and permanent failures. Each CAN node has two Error Count registers. One is in the receiving device and the other is in the transmitting device. If a receive error occurs, the value of the Receive Error Count register will increase by 1. If a transmission error occurs, the value of the Transmit Error Count register will increase by 8. However, good receptions and transmissions of messages will decrease the register values by the same quantities. It is the Error Count register that determines the error status of a node. All nodes in CAN usually operate at a state called Error Active mode. When the value in either register of a node exceeds 127, the state of the node will change from the Error Active mode to a state called Error Passive mode. The node in Error Passive mode can still receive and send messages. Once both values of the Transmit Count and the Receive Count registers reduce to less than 128, the node can change its state from the Error Passive mode back to the Error Active mode. If the error condition persists, such that the value of the Transmit Error Count register exceeds 255, the node will take itself off the bus and go to a state called BusOff mode. That means a permanently faulty node will cease to be active on the bus. Nodes in the BusOff state are still permitted to return to the Error Active mode, with the error counts set to zero, having monitored 128 occurrences of 11 consecutive recessive bits on the bus and received an explicit software reset. The 128 occurrences of 11 consecutive bits equates to 128 messages without any errors, or equivalent bus idle periods[22].

Priority-based algorithm & Media Access Control (MAC)

Basically, CAN is a broadcast Local Area Network(LAN). All messages transmitted via CAN bus will be read by all nodes on the CAN bus. Every message in CAN bus will be assigned an unique identifier which represents the node name and its priority. Each node will decide where or not the message belongs to it by using a mask register to check the identifier of the received message. As is mentioned above, CAN protocol uses the identifier of each message to represent its priority. The lower the identifier value, the higher the priority. When a node has a message to transmit, it will listen to the bus. If the bus is idle, it can start to transmit its most significant bit of the Arbitration field and at the same time it will listen to the bus. If what it reads from the bus is different from what it just sent, that means some node is trying to send a higher priority message and it loses the transmission contention (recall that the dominant bit will always overwrite the recessive bit). Then, it has to stop transmitting, become a listener, and wait until the bus is idle again. If what it reads from the bus is identical to what it just sent, it can proceed to send the second significant bit of the arbitration field. The same procedure will repeat until it transmits all bits in its arbitration field. That means that the node wins the contention. The message it is trying to send has the highest priority on the bus at that time and it can proceed to send all the remaining bits in the message frame.

Whenever a transmission error occurs and is detected by the message transmitter, it will stop the transmission immediately and try to resend the message automatically as soon as it detects the bus is idle again.

By this priority-based Media Access Control protocol, the bus utilization is highly promoted and the most significant advantage is that it guarantees the message with the highest priority will always be transmitted first without any queuing delay, which is also one of the most important things in many real-time control systems.

CHAPTER III

MODEL OF MESSAGE RESPONSE TIME

First, we would like to define the terminology that we are going to use in this section and the following chapters.

1. Message response time :

   The time interval which starts from the time that a message is ready to be sent in a station to the time that the message is transmitted successfully.

2. Periodic signal(message):

   This is the most common message transmitted in CAN-based control systems, e.g., a sensor signal or a control signal. It is generated and ready to be sent every given time period.

3. Sporadic signal(message):

   This is a special high priority signal transmitted in CAN-based control systems. Just like its name, it can happen at any time when the system is working. But it doesn't happen often and always has higher priority than any periodic signal. We use it to represent an emergency signal in a system.

4. Bit time :

This is the time taken to transmit one single bit on CAN bus. In the following section, we will use this time unit to represent the message response time.

Analysis of data latencies

Based on CAN specification 2.0,

Message response time (T) = t1 + t2 + t3 + t4 + t5 + t6 , where

t1 = Time taken to transmit the remaining bits of a current message occupying the bus.

t2 = Time taken to transmit the message itself.

t3 = Time taken to transmit all higher priority (periodic) messages which were queued and ready to be sent during the message response time.

t4 = Time taken to transmit all error frames and the interframe spaces occurring during the message response time.

t5 = Time taken to retransmit the messages which failed to be transmitted during the message response time because of transmission errors.

t6 = Time taken to transmit the sporadic signals.

The third term, t3, can be represented by the message response time, the sampling rate of each message whose priority is higher than m, and the corresponding message length. The fourth term, t4, can be represented by the number of errors occurring during the message response time and the corresponding error overhead. So, we assume message response time of a periodic signal is a function of the sampling rate of each periodic signal (S_i), the priority of each message (P_i), the probability of an error frame occurring per message

transmission (P), and the probability of a sporadic signal occurring in each bit time (Psp).

Assume the message response time of a message with priority m is T_m,

$$T_m = f(S_i, P_i, P, Psp) \tag{3.1}$$

The following model can be generated.

$$T_m = T_{cmsg} + T_{msg_m} + \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i} \right) \times T_{msg_i} \right] + N_{error} \times T_{errorframe+\text{int}ermission} + \sum_{j \in E(m)} T_{msg_j} + \sum_{k=1}^{N_{sp}} T_{smsg_k} \tag{3.2}$$

where,

the first term, T_{cmsg} = t1, second term T_{msg_m} = t2, third term = t3, fourth term = t4, fifth term = t5, and the last term = t6, as defined above.

$p(m)$: A set of periodic messages in which each message has higher priority than m.

t_i : The sampling rate of the periodic message with priority i.

T_{msg_i} : The time taken to transmit the message i.

N_{error} : The number of error frames occurring during the time interval T_m.

$T_{error+\text{int}ermission}$: The time taken to transmit each error frame and the corresponding interframe space.

$E(m)$: A set of messages which were retransmitted during the message response time because transmission errors occurred.

(The number of elements in $E(m)$ should be equal to N_{error}.)

T_{msg_j} : Time taken to retransmit the message j.

N_{sp} : The number of sporadic signals occurring during T_m.

T_{smsg_k} : Time taken to transmit the sporadic signal k.

The "int" in the third term represents the integer function which means we should take the integer value of $\frac{T_m}{t_i}$ and then add 1. Let N be the number of messages to be transmitted before message m is transmitted successfully. Theoretically, N, N_{sp}, and the number of message re-transmissions can be infinity because there is no upper bound for the number of transmission errors occurring during the time interval T_m, which means there is no guarantee for the maximum message response time of a message which doesn't have the highest priority. However, the maximum expected message response time does exist under certain error rate and sporadic signal rate assumptions.

Calculation formulae

Assumption : P and Psp are small enough to make $P*P = 0$ and $P*Psp = 0$, which imply each message will be retransmitted once at most.

Then, the expected value of N_{error} is $N * P$. The expected value of N_{sp} is $T_m * Psp$. Using the real number of standard data frame specified on CAN specification 2.0, we can derive the following equations to calculate the maximum expected message response time.

1. For the periodic signal with the highest priority:

   $N = N_{sp} + 2$ (including T_{cmsg} and one retransmission of the message itself)

The maximum number of bits in a standard data frame to be transmitted on CAN is 127 including stuff bits. The maximum number of bits of the error overhead is 23 which includes the maximum length of superposition of error flags, 12, 8-bit error frame delimiter, and 3-bit Intermission of the Interframe Space.

So, the maximum expected message response time (T_m) :

$$T_m = 127 + 127 + N \times P \times 23 + N \times P \times 127 + N_{sp} \times 127$$

$$= 254 + (N_{sp} + 2) \times P \times 23 + (N_{sp} + 2) \times P \times 127 + N_{sp} \times 127$$

$$= 254 + (N_{sp} + 2) \times P \times 150 + N_{sp} \times 127$$

$$= 254 + (T_m \times Psp + 2) \times P \times 150 + (T_m \times Psp) \times 127$$

$$= 254 + 300 \times P + 127 \times Psp \times T_m$$

$$\rightarrow T_m = \frac{254 + 300P}{1 - 127Psp} \tag{3.3}$$

2. For a periodic message with priority m :

$$N = N_{sp} + 2 + \sum(\text{int}(\frac{T_m}{t_i})) = T_m \times Psp + 2 + \sum(\text{int}(\frac{T_m}{t_i}))$$

$$T_m = 127 + 127 + \sum_{i=p(m)} \left[\text{int}\left(\frac{T_m}{t_i}\right) \times 127 \right] + N_{error} \times T_{errorframe+intermission} + \sum_{j=E(m)} T_{msg_j} + \sum_{k=1}^{N_{sp}} T_{smsg_k}$$

$$= 254 + 127 \times \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i}\right) \right] + (N_{sp} + 2 + \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i}\right) \right]) \times P \times 23$$

$$+ (N_{sp} + 2 + \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i}\right) \right]) \times P \times 127 + (T_m \times Psp) \times 127$$

$$= 254 + 127 \times \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i}\right) \right] + (T_m \times Psp + 2 + \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i}\right) \right]) \times P \times 23$$

$$+ (T_m \times Psp + 2 + \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i} \right) \right]) \times P \times 127 + (T_m \times Psp) \times 127$$

$$= 254 + 127 \times \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i} \right) \right] + (2 + \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i} \right) \right]) \times P \times 150 + (T_m \times Psp) \times 127$$

$$\rightarrow T_m = 254 + 127 \times \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i} \right) \right] + (2 + \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i} \right) \right]) \times P \times 150 + (T_m \times Psp) \times 127 \quad (3.4)$$

An iteration method can be used to obtain the minimum T_m which satisfies Equation (3.4). Please note that a smaller index number represents a higher priority.

3. For sporadic signals :

   $N = 2 + N_{sp}$

   where,

   $$N_{sp} = \left(\sum_{k \in h(p)} Psp_k \right) \times Tsp_p$$

   $h(p)$: A set of sporadic signals whose priorities are higher than p.

   Psp_k : The probability of the sporadic signal k occurring in each bit time.

The maximum expected message response time for a sporadic signal with priority p :

$$Tsp_p = 254 + N \times P \times 23 + N \times P \times 127 + N_{sp} \times 127$$

$$= 254 + (2 + N_{sp}) \times P \times 23 + (2 + N_{sp}) \times 127 + N_{sp} \times 127$$

$$= 254 + 150 \times P \times (2 + \sum_{k \in h(p)} Psp_k \times Tsp_p) + 127 \times \sum_{k \in h(p)} Psp_k \times Tsp_p \quad (3.5)$$

For the sporadic signal with highest priority :

$$Tsp_{hp} = 254 + 23 \times P + 127 \times P \tag{3.6}$$

By appropriately choosing P and Psp to meet the assumptions, we can find the expected message response time of each message transmitted on the CAN bus. With the model (3.2), we can easily derive formulae to calculate the maximum expected message response time for any specific CAN based control system.

Figure 3.1 shows the timing diagram of a typical CAN-based control system.

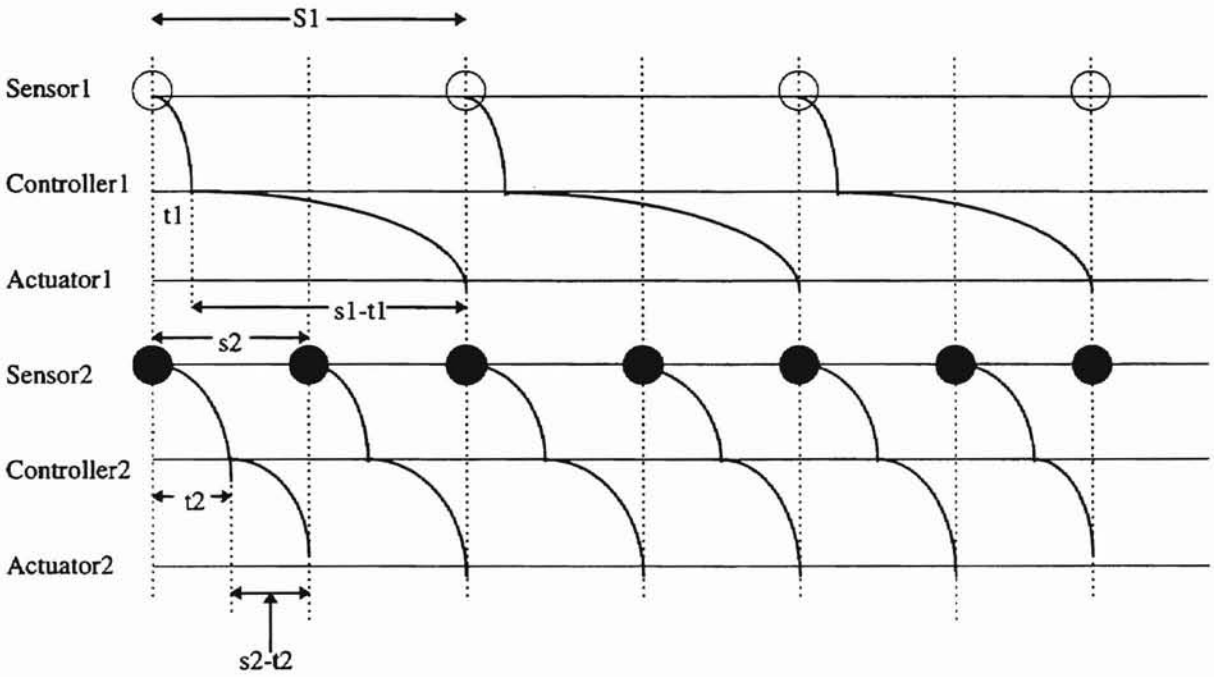


Figure 3.1 Timing diagram of a CAN-based control system

The sampling period of the sensor1 is s1 and the sampling period of the sensor2 is s2. If the time taken to transmit the sensor1 signal to the controller1 is t1 and the time taken to transmit the sensor2 signal to the controller2 is t2, in order to avoid delay, the controller1 signal must be transmitted successfully before the next sampling instant of sensor1 and the controller2 signal must be transmitted successfully before the next sampling instant of sensor2. In other words, if we ignore the processing time for the control signal, the control signal must be transmitted successfully from controller1 to actuator1 in the time interval, s1-t1, and the control signal from controller2 to actuator2 must be transmitted successfully in the time interval, s2-t2, which means that the data latency of the message from controller1 must be less than s1-t1 and the data latency of the message from controller2 must be less than s2-t2 . So, if we can determine the message response time of each message with its priority, we can predict whether or not the control system meets the real-time constraints. In the next chapter, computer simulation will be used to verify the model and show how to use the model to analyze the message response time of each specific message and how to predict message delays.

CHAPTER IV

SIMULATION OF CAN

Pennathur designed a CAN simulation program which covered most of the CAN real-time features in 1993 [17]. However, his program contains the following characteristics which do not meet our requirements for the analysis of CAN-based control systems.

1. Only periodic signals exist on the CAN bus (Treat sporadic signals as periodic signals).
2. A fixed priority assignment algorithm is used to decide the priority of each message.
3. Not enough useful information can be found from the outputs of the simulation program. In order to analyze CAN-based control systems, we want to know the number of message delays, messages dropped, messages rejected, and the error rate.

Based on CAN specification 2.0, a bitwise CAN simulator has been designed to simulate the network behaviors of CAN-based control systems. In addition to simulating CAN bit by bit, it breaks the total simulation time into each bit time and simulates how each station (sensors, controllers, etc.) will behave during each bit time (including error handling, Interframe space, Overload Frame, etc.). This simulator doesn't cover Remote Frame and Fault Confinement because we are concerned with real-time constraints more than with the other issues.

To use the simulator, all we need to do is to put the system parameters of a CAN-based control system into an input data file. The simulator will save the total simulation results to an output file which contains all the information we need to know about the network behaviors of a CAN-based control system including the throughput, error rate, number of message delays, messages dropped, messages rejected, vacant sampling, and message re-transmissions. The input data format is as follows.

Simulation time (in seconds)
Network speed (bandwidth in kilobits per second)
The number of stations(nodes) on the CAN bus
Probability of Overload Frames occurring in total simulation time (I)
Probability of Overload Frames occurring in total simulation time (II)
Probability of Form(Frame) errors occurring in total simulation time (I)
Probability of Form(Frame) errors occurring in total simulation time (II)
Probability of Form(Frame) errors occurring in total simulation time (III)
Probability of Form(Frame) errors occurring in total simulation time (IV)
Probability of Bit Stuffing errors occurring in total simulation time (I)
Probability of Bit Stuffing errors occurring in total simulation time (II)
Probability of Bit Stuffing errors occurring in total simulation time (III)
Probability of Bit Stuffing errors occurring in total simulation time (IV)
Probability of CRC errors occurring in total simulation time (I)
Probability of CRC errors occurring in total simulation time (II)
Probability of CRC errors occurring in total simulation time (III)
Probability of CRC errors occurring in total simulation time (IV)
Probability of ACK errors occurring in total simulation time (I)
Probability of ACK errors occurring in total simulation time (II)
Probability of ACK errors occurring in total simulation time (III)
Probability of ACK errors occurring in total simulation time (IV)

(The real probability of Overload Frames occurring is equal to $\frac{(I)}{(II)}$ and the real probability of each Error Frame occurring is $\frac{(I)}{(II)} \times \frac{(III)}{(IV)}$.)

(For periodic signals)
Node name Sampling period(in seconds) Sampling start instant(in seconds) Time skew(in seconds) Message identifier Data frame type The number of messages received Message Masks(id.)

(For sporadic signals)
Node name 0 (Sampling period) Probability of sporadic signal(I) Probability of sporadic signal(II) Time skew(in seconds) Message identifier Data frame type The number of messages received Message Masks(id.)

(The real probability of sporadic signals occurring is equal to $\frac{(I)}{(II)}$.)

The definition of the time skew is the time difference of the sampling instants between the sensor and the controller. The data frame type is used to represent which type of the Data Frame will be sent by the node. 0 represents the Standard Data Frame and 1 represents the Extended Data Frame. In the Message Masks section, the identifiers of the messages are used to perform the function of mask registers. The data length in the Data Field of each data frame is generated by a computer-generated pseudo-random number varying from 1 to 8 bytes. The Overload Frame, 5 different error flags, and the sporadic signals are generated by using computer-generated pseudo-random numbers based on their probabilities in the input data files.

Table 4.1 shows the error generation and error detection performed in this simulator.

	Generated by	Detected by
Form(Frame) Error	Message Transmitter	All stations
Bit Stuffing Error	Message Transmitter	All stations
CRC Error	Message receivers	Message Transmitter
ACK Error	Message receivers	Message Transmitter
Bit Monitoring Error	All stations	Message Transmitter

Table 4.1 Error generation and detection in the simulator

Form Error rate = $P_{form} \times 3$ (4.1)

CRC Error rate = $P_{CRC} \times 15 \times (N-1)$ (4.2)

ACK Error rate = $P_{ACK} \times (N-1)$ (4.3)

(For both Standard Data Frame and Extended Data Frame)

Bit Stuffing Error rate = $\left(\frac{21+8\times datalen}{5}\right) \times P_{bstuff}$ (4.4)

where,

N : The number of nodes in CAN

P_{form} : Probability of Forma Error occurring

P_{CRC} : Probability of CRC Error occurring

P_{ACK} : Probability of ACK Error occurring

P_{bstuff} : Probability of Bit Stuffing Error occurring

datalen : The data length of the Data field in the message (1-8 bytes for a Data Frame)

The overall error rate P during each message transmission is the sum of the above error rates.

$$P = P_{form} \times 3 + P_{CRC} \times 15 \times (N-1) + P_{ACK} \times (N-1) + \left(\frac{21+8\times datalen}{5}\right) \times P_{bstuff} \quad (4.5)$$

Thirteen different sets of input data were chosen to run the simulation. The simulation time of each data file is 10 seconds. Each data file was run 10 times and averaged to determinate the final results. The number of nodes varies from 5 to 15 and each station transmits one message only. This simulator does not use any specific priority assignment

algorithm to decide the priority of each message. It leaves this assignment to the user as a system parameter. However, in the simulations of this study, we do use the "short job, first" principle to decide the priorities, which means the higher priorities are assigned to the messages which are transmitted less frequently.

After assigning the priority of each message and appropriately choosing the probability of each error rate to meet the assumptions, another program was used to calculate the expected message response time of each message based on the formulae derived in Chapter III. In order to show how to analyze and predict the message delays, consider the following example. First, we use the program to calculate the expected message response time of each message. The results are shown as follows.

Expected message response time calculation for the first input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):4
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.05
Input the sampling rate of the priority 2 message (in Sec.):0.04
Input the sampling rate of the priority 3 message (in Sec.):0.04
Input the sampling rate of the priority 4 message (in Sec.):0.05
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected message response time of priority (1):0.0011640
Expected message response time of priority (2):0.0017440
Expected message response time of priority (3):0.0023280
Expected message response time of priority (4):0.0029120

As shown in the first input data file, the signal from sensor1 to controller1 is the highest priority periodic signal, the signal from sensor2 to controller2 is the second, the control signal from controller2 is the third, and the control signal from controller1 is the lowest

priority signal. From the results of the message response time calculations, we find that the sampling interval of the message from sensor1 is 0.05 sec. and its expected message response time is 0.001164 sec., which means the control signal from controller1 must be transmitted successfully in the time interval, 0.05-0.001164(= 0.048836) in order to avoid a message delay. As we can see, the expected message response time of the lowest priority signal, the control signal from controller1, is 0.002912 which is less than the time interval, 0.05-0.001164. The same calculation is used to check every other message in the input data. So, we predict that all messages in this input data file should meet their deadlines.

By the same procedure, all messages from the second to the fourth input data file shouldn't have any problem meeting their time constraints. However, we do find some problems from the fifth to the eleventh input data file. The calculation results of the fifth data file show that the message response time of the control signal from controller6, the seventh highest priority signal, is 0.00466 which is larger than the deadline, 0.008-0.004076(= 0.003924). So, we can not expect that all messages will meet their deadlines. The same conditions happen from the sixth to the eleventh data file. So, we predict that there may be some message delays in the simulation results. In the twelfth and the thirteenth data file, the expected message response times of all signals are larger than their deadlines. We can expect to see many message delays in the results of the simulations.

The overall results of the simulations are summarized and shown in the following figures.

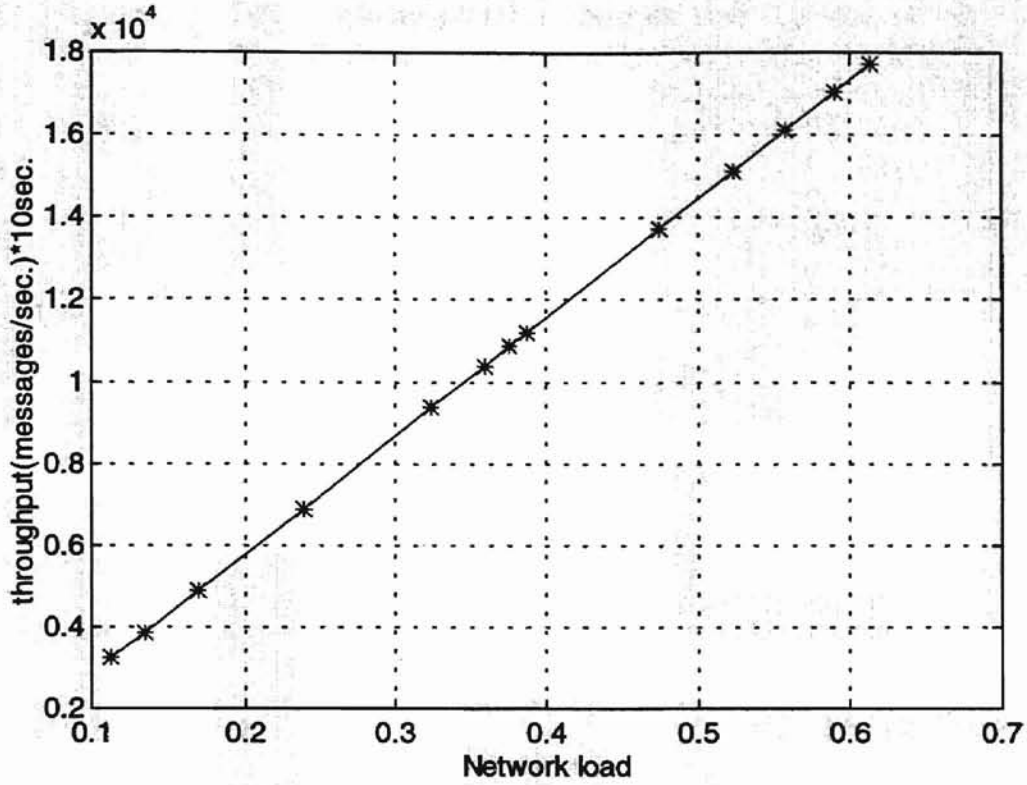


Figure 4.1 Throughput plot (in 10 Sec.)

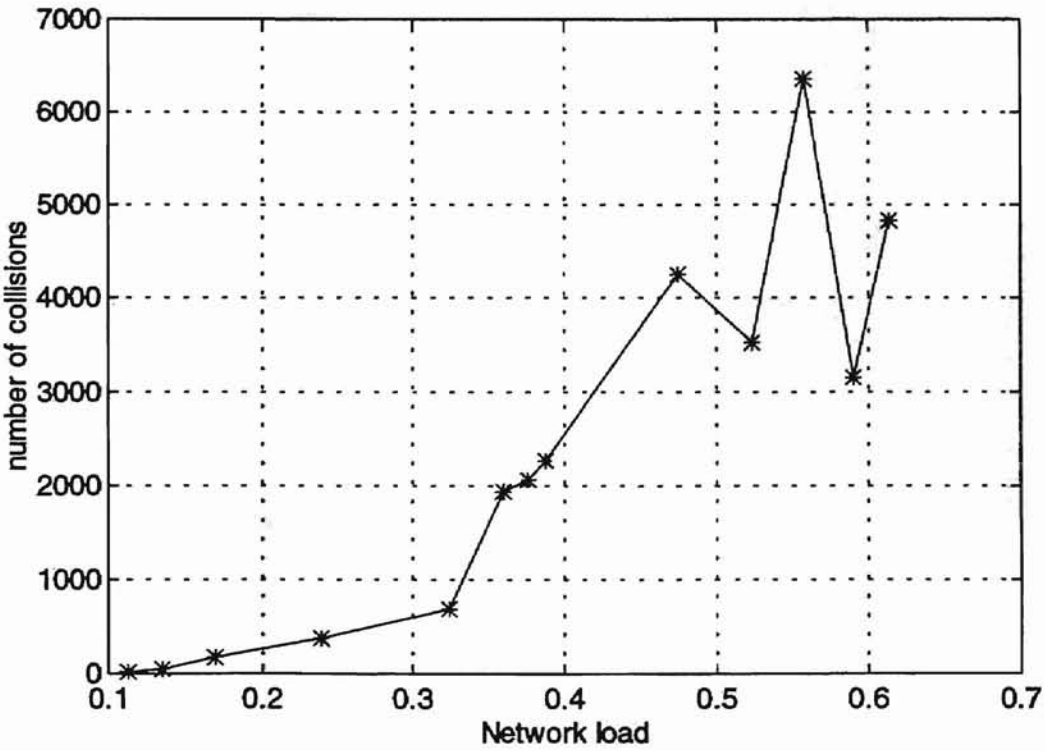


Figure 4.2 Number of collision plot

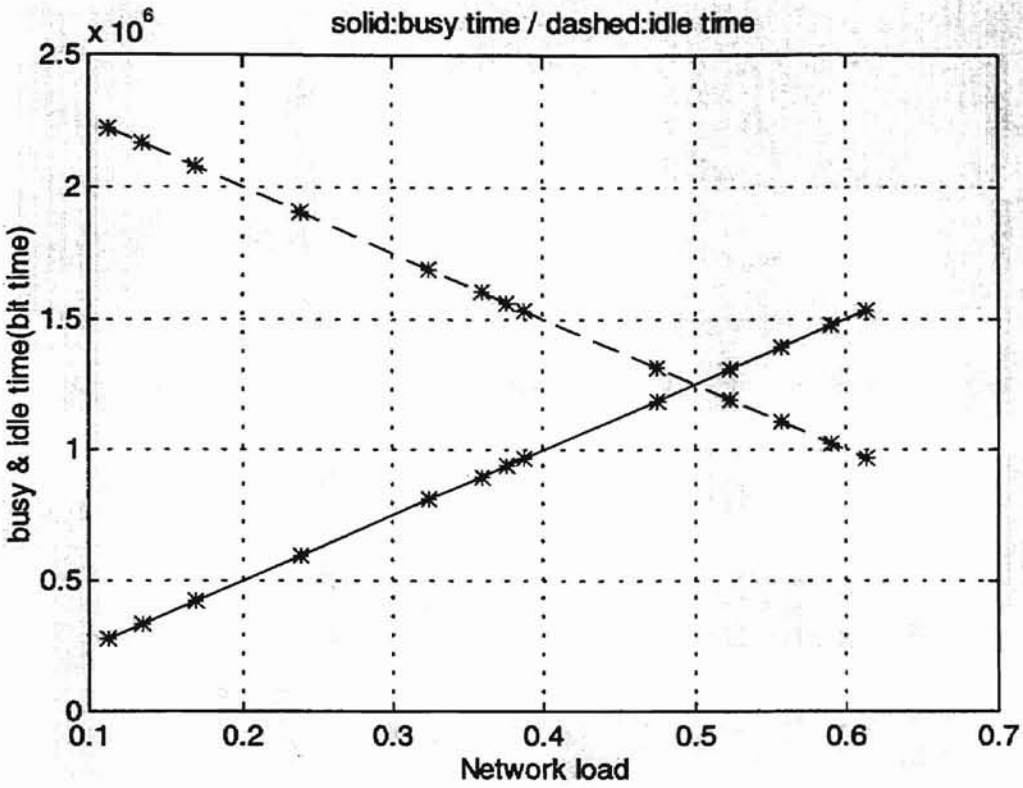


Figure 4.3 Busy/idle time plot

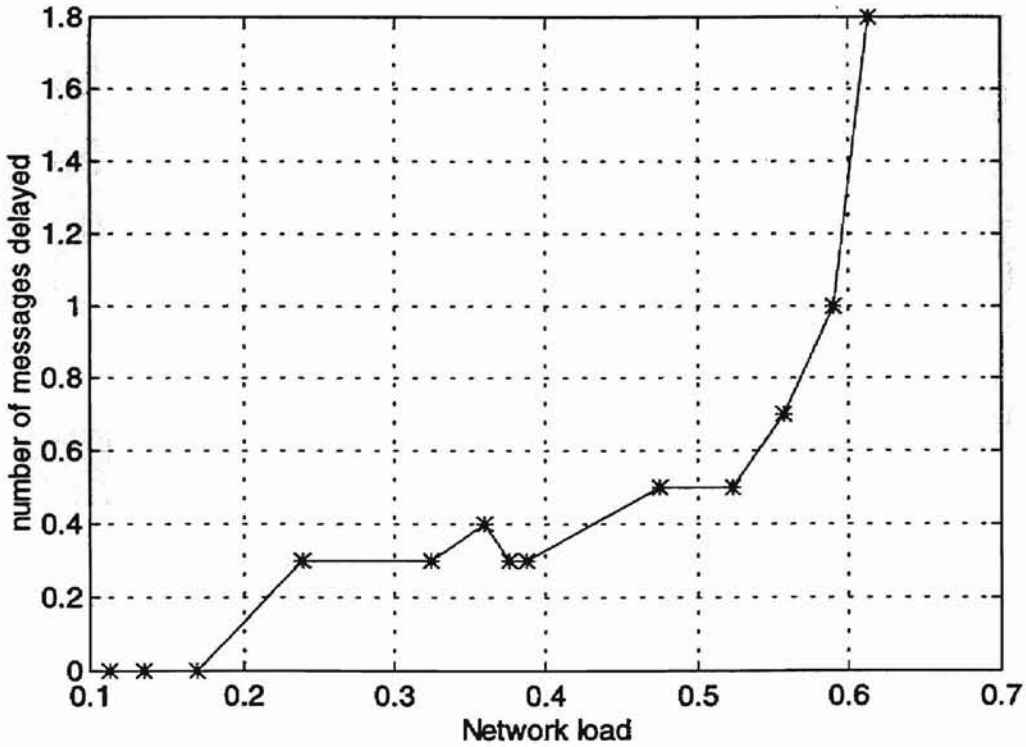


Figure 4.4 Number of message delay plot

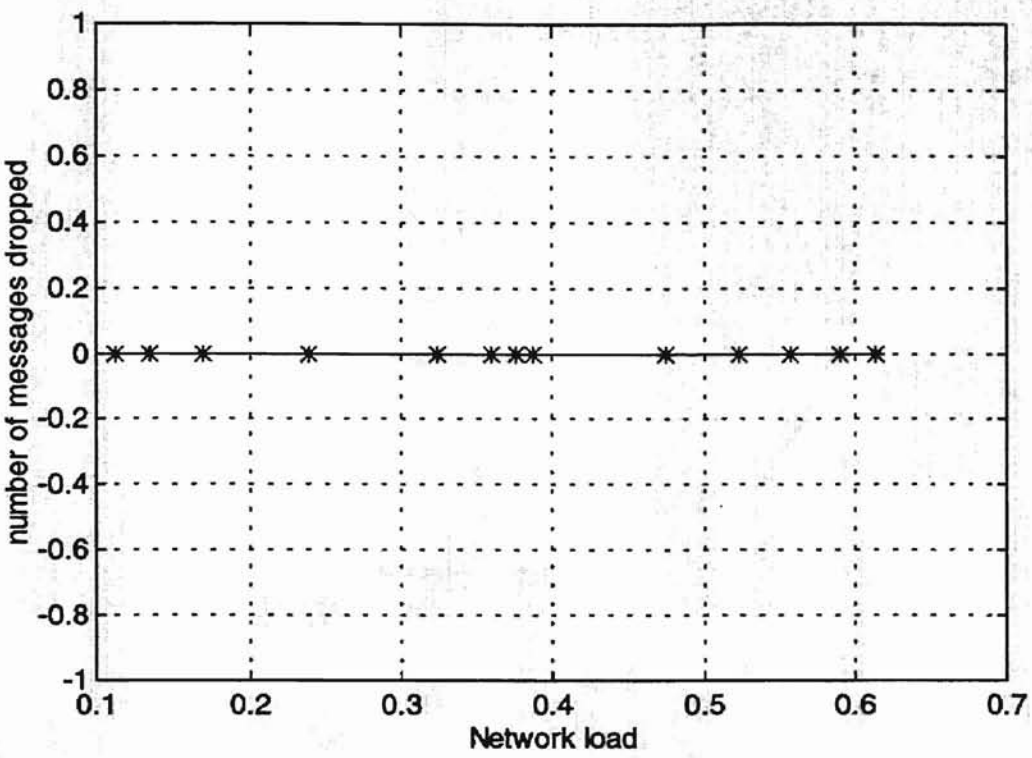


Figure 4.5 Number of message dropped plot

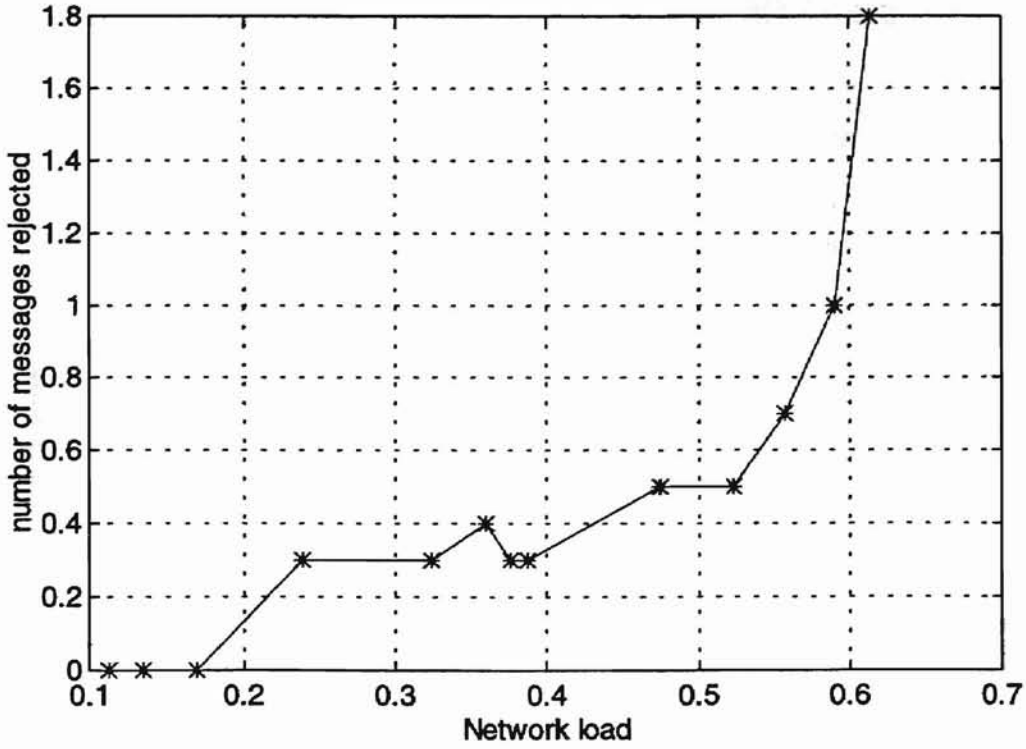


Figure 4.6 Number of message rejected plot

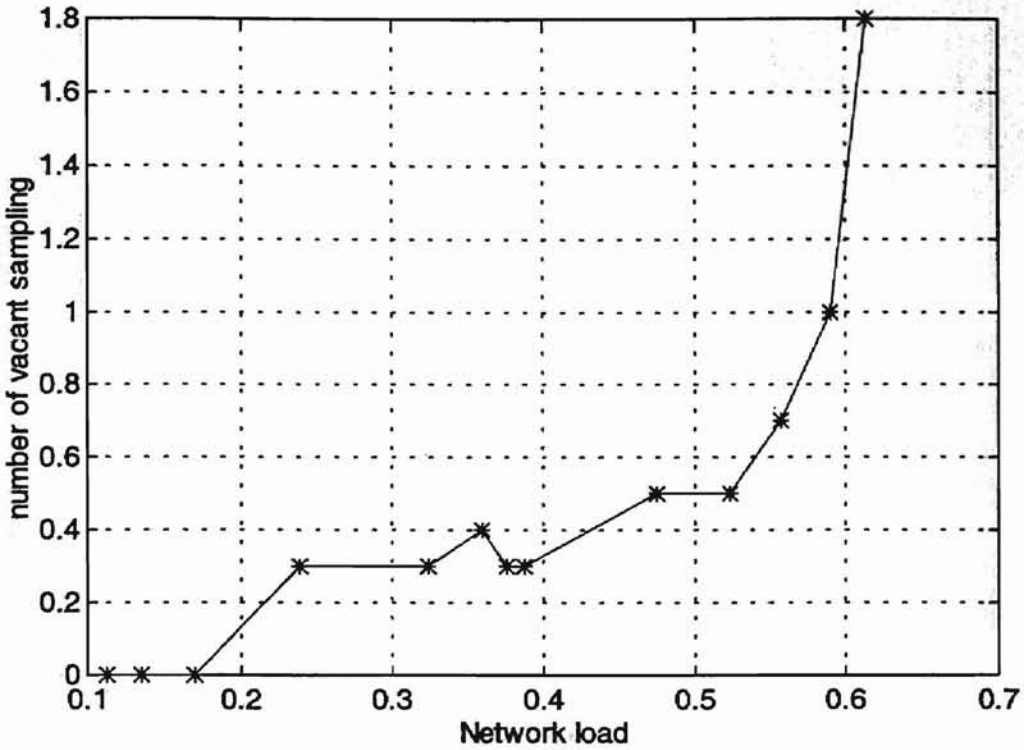


Figure 4.7 Number of vacant sampling plot

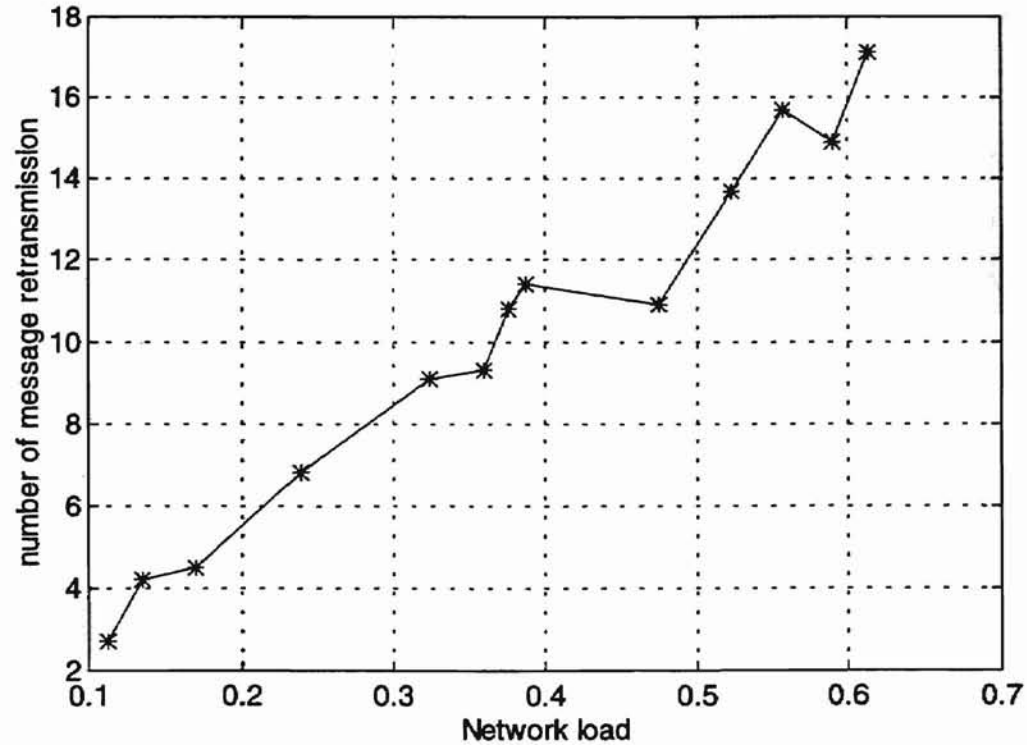


Figure 4.8 Number of message retransmission plot

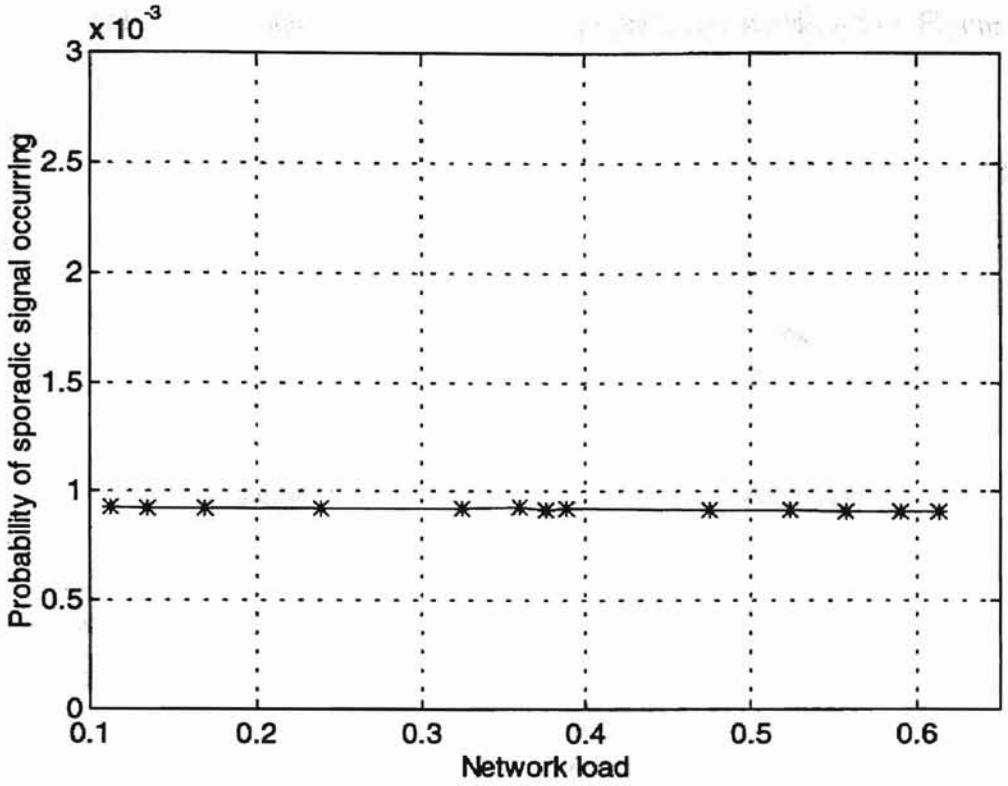


Figure 4.9 Sporadic signal rate plot

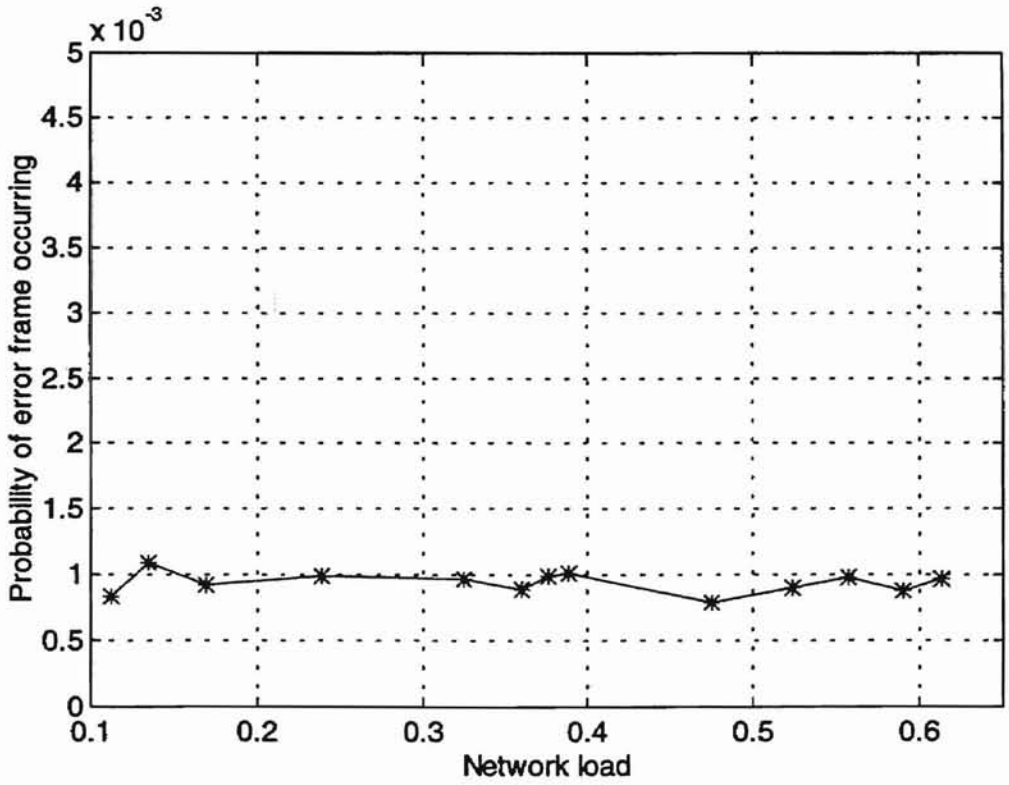
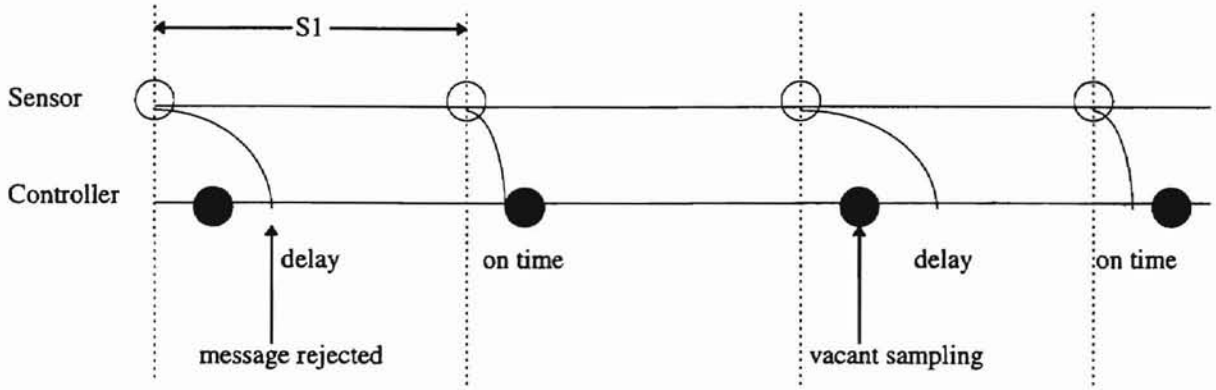


Figure 4.10 Error rate plot

The definitions of vacant sampling and message rejection are shown on Figure 4.11.



○ Sensor sampling instant

● Controller sampling instant

S1 : Sampling interval of the sensor

PS. : Controller has a fixed(constant) sampling period

Figure 4.11 vacant sampling and message rejection

As mentioned before, the total simulation time is divided into each bit time in which the network bus is either busy or idle. The bus busy time includes the time taken to transmit data frames, overload frames, error frames, interframe space, etc. So, the network load is defined as ratio of bus busy time to the total simulation time.

Network load = bus busy time / total simulation time

Analysis and discussion

Figure 4.1 shows that the relation between the network load and the throughput is linear when the network load is under 62%. Figure 4.2 gives us an idea of how the number of message collisions is related to the network load. A message collision happens whenever there is more than one node trying to send a message at the same time. It is related to the transmission rate of each message, the sporadic signal rate, the error rate, and the occurring instances of the sporadic signals and the transmission errors. In Figure 4.3, the idle time curve is linearly decreasing and bus busy time curve is linearly increasing with the increased network load. It is clear that as more messages are sent on the bus, the bus has a smaller amount of free time.

In Figure 4.4, we find the simulation results comply with our intuitive predictions. The number of message delays increases dramatically when the network load is higher than 50%. When comparing the last 6 data points with their corresponding message response time calculations, we find that the number of message delays increases dramatically as the number of the messages whose expected message response times are longer than their time constraints increases. In the last two data points, the expected message response times of all messages are higher than their real-time deadlines. That's why we see the number of message delays reaches 10 and 18 in the twelfth and thirteenth data point, respectively, during the total of 10 simulations, which means that the message delays happened , on average, 1 time during each one of the ten simulations for the twelfth data point and 1.8 times during each one of the ten simulations for the thirteenth data point. The network reached its limitation at 58% bus load.

According to the calculations, there should be no message delay occurring in the simulation results of the first four input data files. But, the number of message delays reaches 0.3 in the simulation results of the fourth data point. The reason for that is clearly from the fact that our expected message response time formulae are based on the assumptions that all messages will be retransmitted once at most, which means the number of transmission errors for each message transmission is less than or equal to 1. However, as mentioned previously, the transmission errors were generated by pseudo-random numbers based on certain error rates in the input data files. If the number of transmission errors occurring for the same message is higher than 1, the real data latency (message response time) may be longer than the expected message response time we calculated. That's why we still can see very few message delays in the simulation results. It is less than 1 because we ran the simulation 10 times and showed the average numbers only. Also, by the calculations, the data latencies of some messages in the input data files (from the seventh to the eleventh) are larger than their deadlines. But, we do not see a lot of message delays in the simulation results. That's because the expected message response times are calculated by using the maximum length of a Data Frame including stuffing bits and we assume that the worst conditions always occur in the real world. However, in the real conditions, it may not be that bad. So, as long as the error rate and sporadic signal rate meets our assumptions, the messages still can meet their time constraints.

Whenever a message is queued in the transmitter and can't be sent successfully because another messsage occupies the bus or transmission errors happen until the next message is ready to be sent, the new message will overwrite the old one. We call this

condition "message dropped". Figure 4.5 shows that no message was dropped during the total simulation time because of the data latencies in the previous message transmissions. The curves of message rejected and vacant sampling in Figures 4.6 and 4.7 are similar to that in Figure 4.4. Figure 4.8 indicates that the number of message re-transmissions increases when the network load is increasing. Actually, it is equal to the number of transmission errors occurring during the simulation time and it is directly related to the throughput and the error rate. Figure 4.9 tells us that the sporadic signal rate during the total simulation time is a constant around 0.00093. It is very close to the value, 0.001, we set in the input data files. Figure 4.10 shows the real error rates occurring during the simulations. It is around 0.001, which is consistent with the values in our input data files.

Further investigation of message response time

In order to further investigate CAN message response time, we perform 8 different experiments and use extended data frames instead of the standard frames in all experiments. The length of the data field in each data frame varies from 1 byte to 8 bytes same as our last simulations. There are 3 nodes in each one of the experiments. They are a message transmitter, a "noise" maker, and a listener. The noise maker generates a given bus load under which the message transmitter tries to send out messages to the listener. The messages generated by the noise maker in the first 4 experiments are periodic signals and those in the last 4 experiments are random signals which are similar to the sporadic signals we previously mentioned. We record the data latencies(message response times) of the messages generated by the message transmitter and show the average values of the

latencies under different bus loads. The definition of the bus load is exactly the same as the network load mentioned before which includes the bus load generated by the noise maker, the message transmitter, and the corresponding error overheads. Table 4.2 shows the specification of each experiment.

Experiment	message transmitter		noise maker		simulation results	
	transmission period	priority	message generated type	priority	latency plot	others
1	100 ms	Low	periodic (uniform)	High	Figure 4.12	*C.1-7
2	100 ms	High	periodic (uniform)	Low	Figure 4.13	*C.8-14
3	10 ms	Low	periodic (uniform)	High	Figure 4.14	*C.15-21
4	10 ms	High	periodic (uniform)	Low	Figure 4.15	*C.22-28
5	100 ms	Low	sporadic (random)	High	Figure 4.16	*C.29-36
6	100 ms	High	sporadic (random)	Low	Figure 4.17	*C.37-44
7	10 ms	Low	sporadic (random)	High	Figure 4.18	*C.45-52
8	10 ms	High	sporadic (random)	Low	Figure 4.19	*C.53-60

* Refer to Appendix C

Table 4.2 Specifications of the experiments

Figures 4.12 and C.1-7 are the simulation results of the first experiment. From the latency plot, the message response time keeps increasing slowly when the bus load is under 80% and it goes up dramatically when the bus load is higher than 85%. It is clear that the messages with low priority from the message transmitter must wait a longer time to be sent as the noise maker generates higher bus loads. This condition is illustrated by the message collision plot (refer to Appendix C) which shows that the number of

collisions increases dramatically when the bus load is higher than 80%. Also, from the messages dropped plot, we find that more and more messages from the noise maker are dropped when the bus load is higher than 80%.

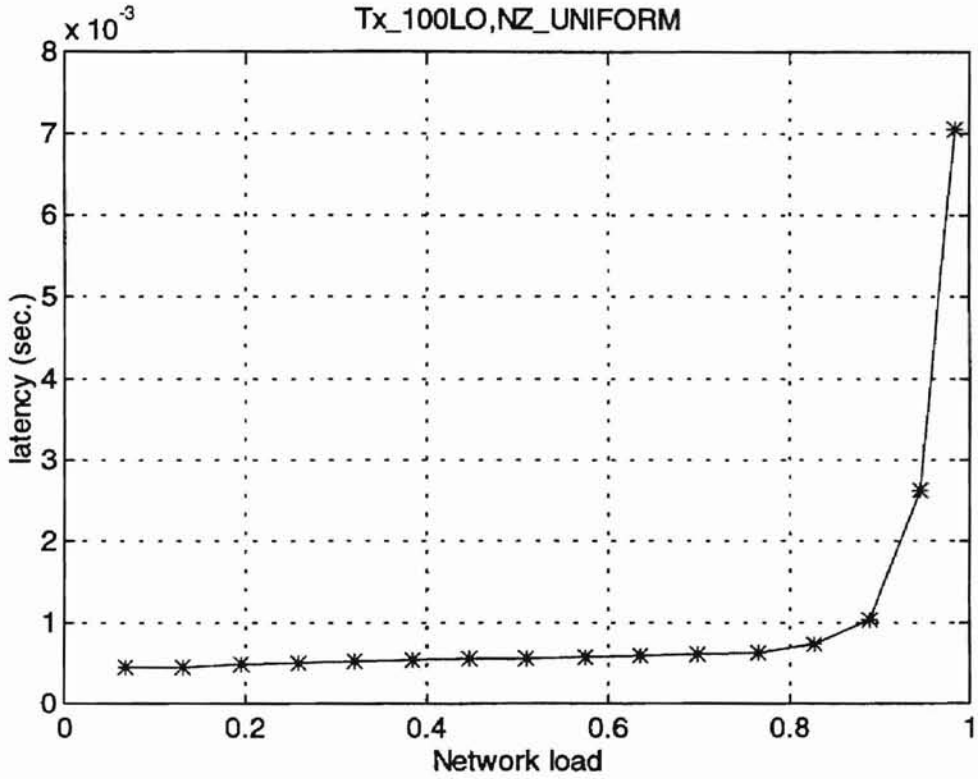


Figure 4.12 Latency plot of experiment 1

Figures 4.13 and C.8-14 show the simulation results of the second experiment where the messages from the message transmitter have higher priority than those from the noise maker. As we can see, the message response time(data latency) increases very slowly even though the network load is getting higher. Actually, it shows that the network load has very little affect for the higher priority message. That's because the messages from

the transmitter have higher priority and CAN guarantees the data latency for the highest priority message.

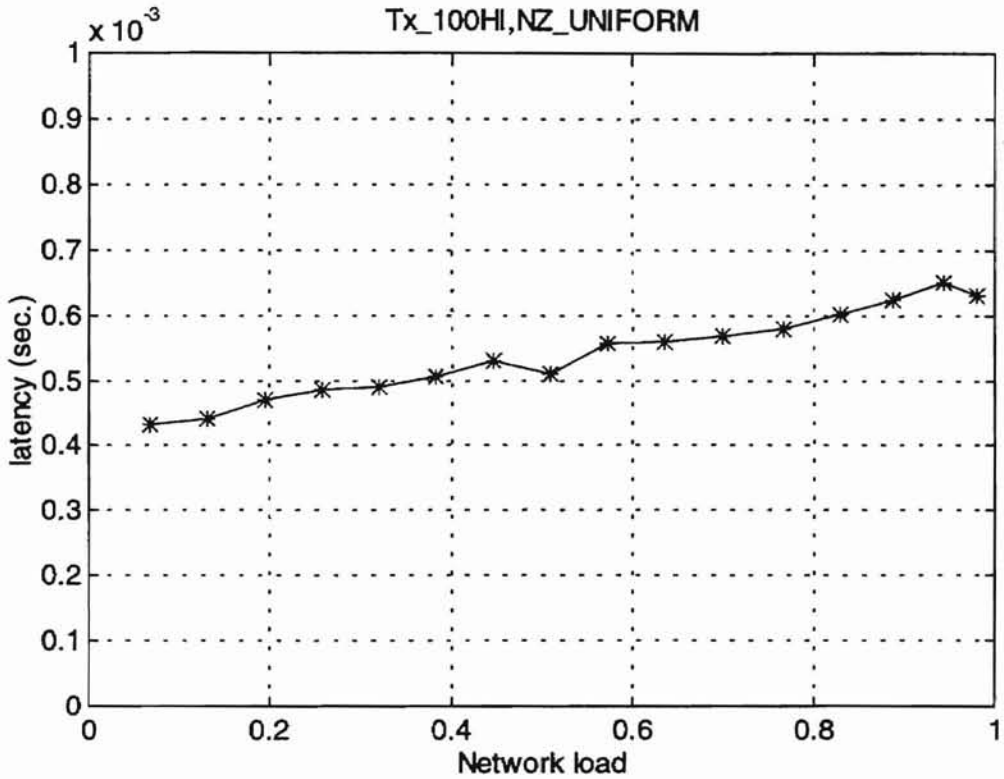


Figure 4.13 Latency plot of experiment 2

In experiments 3 and 4, we adjust the transmission period of the message transmitter from 100 ms to 10 ms. In experiment 3, we notice that the data latencies are similar to those in the experiment 1 when the bus load is under 90%. One interesting difference is that the throughput from the massage transmitter drops dramatically when the bus load is higher than 90%. The results of experiment 4 are almost the same as those of experiment 2. They indicate that the priorities of messages are much more important than the transmission rates of messages in data latencies(message response times).

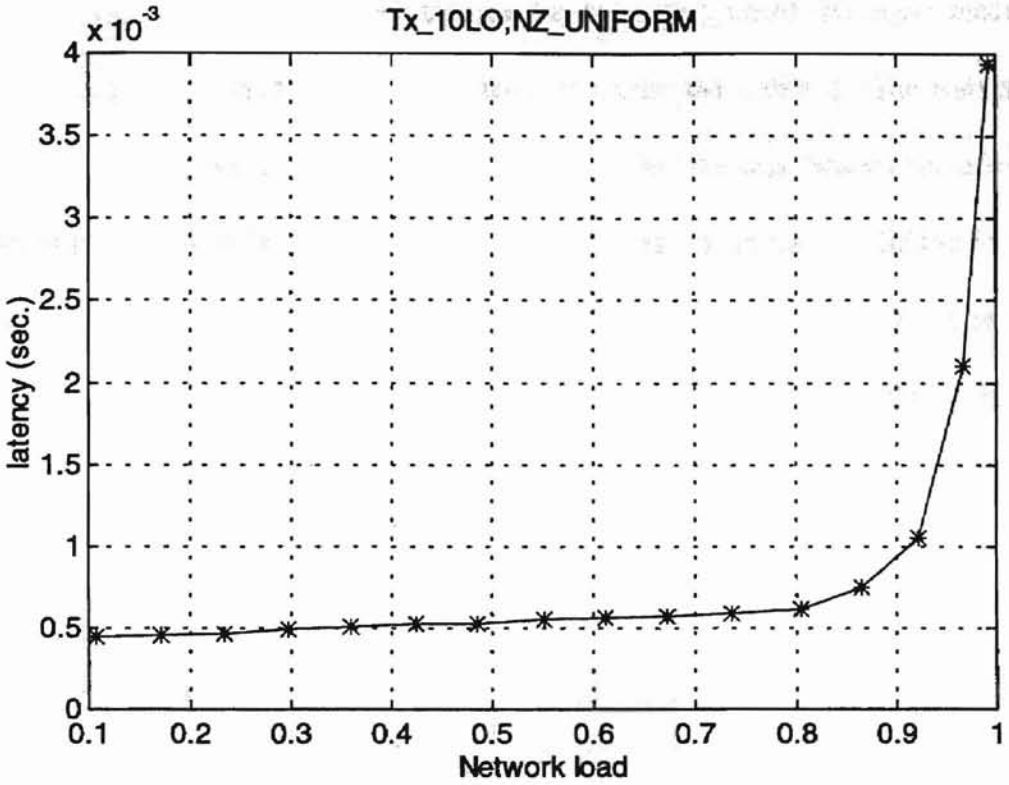


Figure 4.14 Latency plot of experiment 3

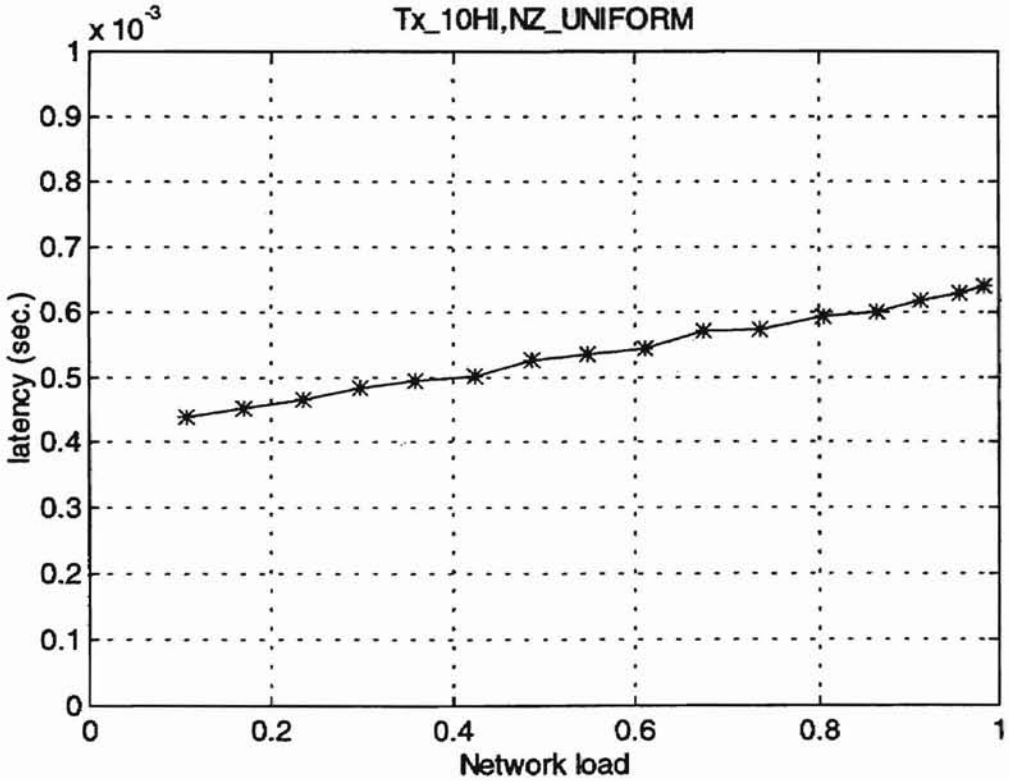


Figure 4.15 Latency plot of experiment 4

From experiments 5 to 8, we change the message type of the noise maker from periodic signals to sporadic signals in order to compare the impacts. The results of the experiment 5 are shown in the Figures 4.16 and C.29-36. The data latency curve oscillates but keeps increasing as the bus load increases. The reason for the oscillation is that the "noise signals" are generated in a sporadic pattern which causes the latencies of the "normal signals" to vary. The overall results are very similar to the results in experiment 1.

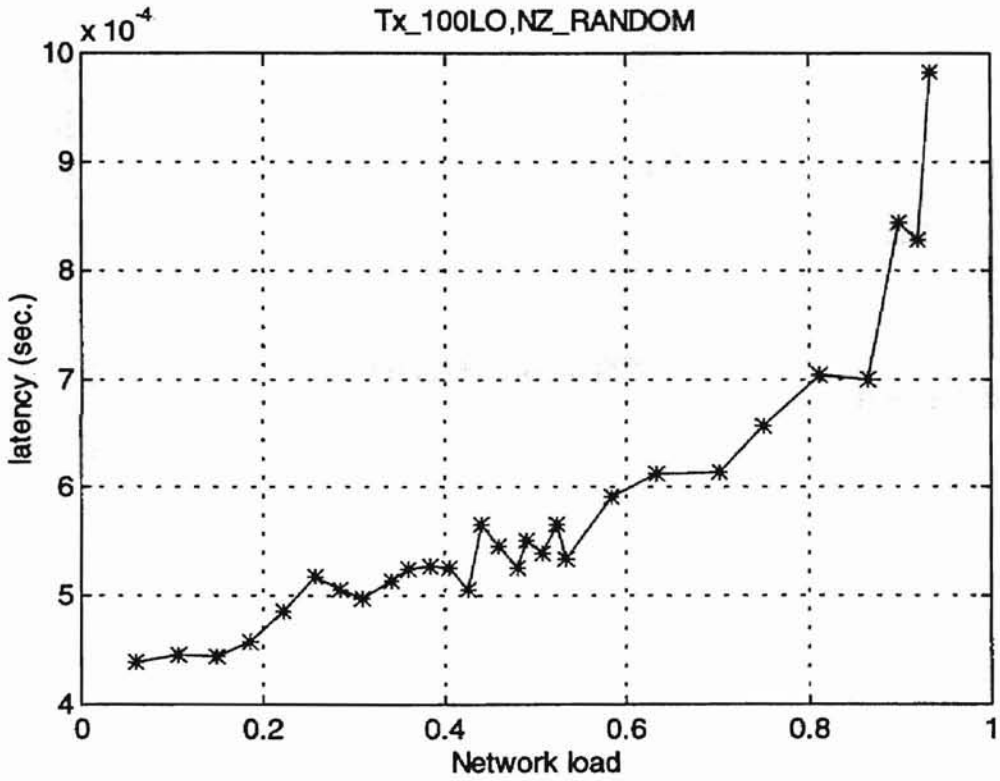


Figure 4.16 Latency plot of experiment 5

Figures 4.17 and C.37-44 show the simulation results of experiment 6, which are like a fuzzy version of experiment 2. Figures 4.18 and C.45-52 and Figures 4.19 and C.53-60 are the results of experiments 7 and 8, respectively. They are very similar to the results of experiments 3 and 4 except for the curve oscillations.

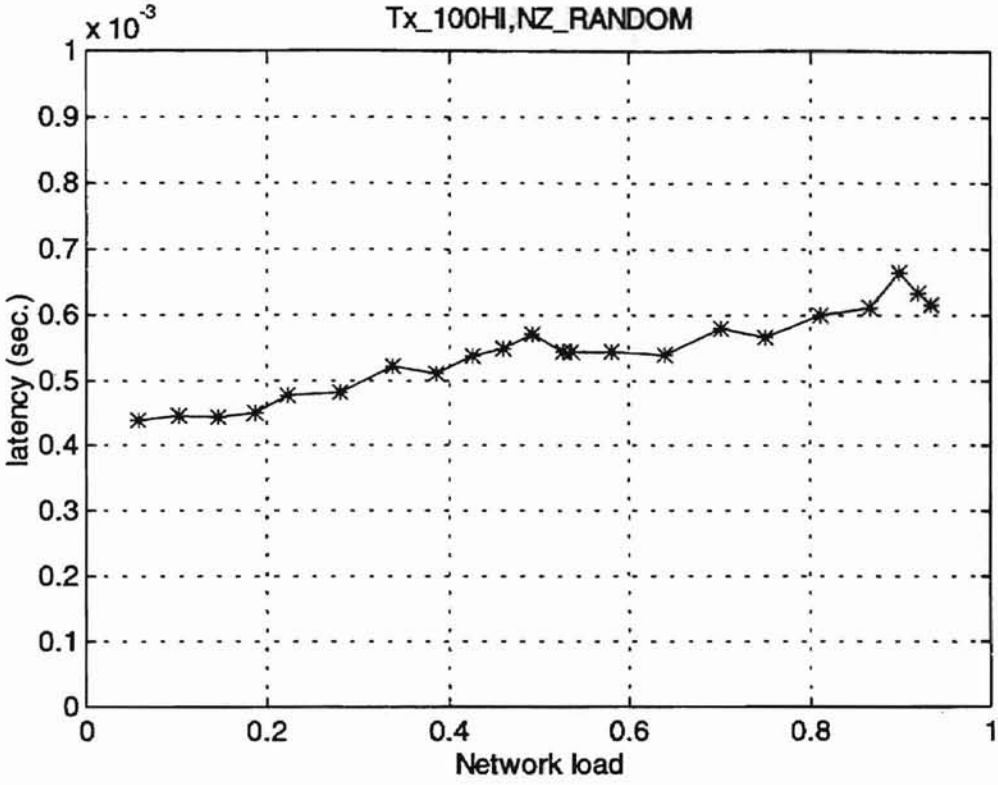


Figure 4.17 Latency plot of experiment 6

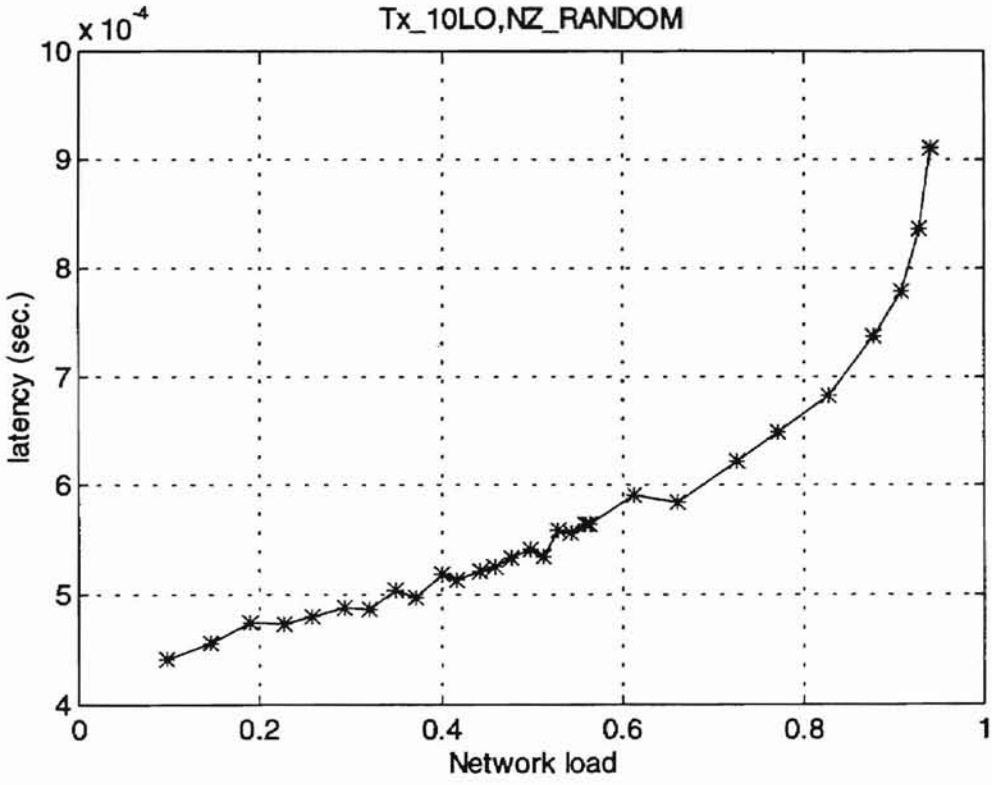


Figure 4.18 Latency plot of experiment 7

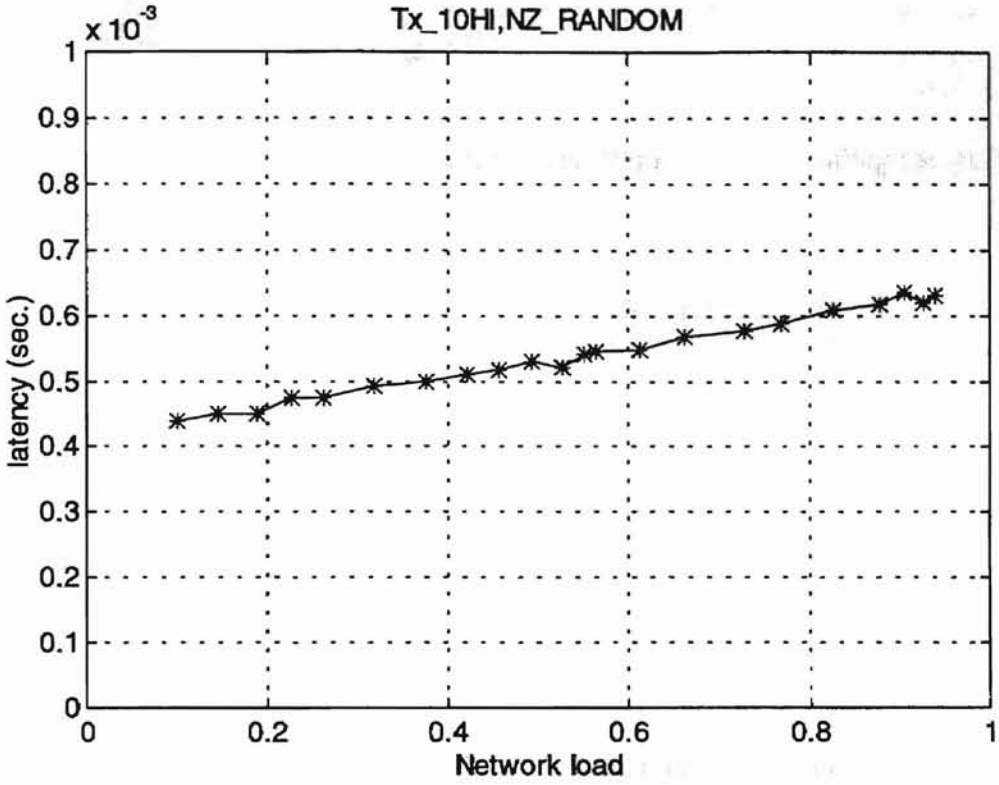


Figure 4.19 Latency plot of experiment 8

Our model can be used to calculate the expected data latencies of the experiments. According to the specifications of the experiments, the model can be simplified as follows.

1. For the "periodic noise signal" experiments where the messages from the message transmitter have lower priority than the "noise messages".

$$T_m = T_{cmsg} + T_{msg_m} + \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i}\right) \times T_{msg_i} \right] + N_{error} \times T_{errorframe+\text{int}ermission} + \sum_{j \in E(m)} T_{msg_j} \qquad (4.6)$$

The sporadic signal term is gone because there is no sporadic signal in the experiments.

$$N = \sum_{i \in p(m)} \left(\text{int}\left(\frac{T_m}{t_i} \right) \right) + 2 \quad \text{and} \quad N_{error} = N \times P$$

The maximum number of bits in an extended data frame is 151 including the stuffing bits. So,

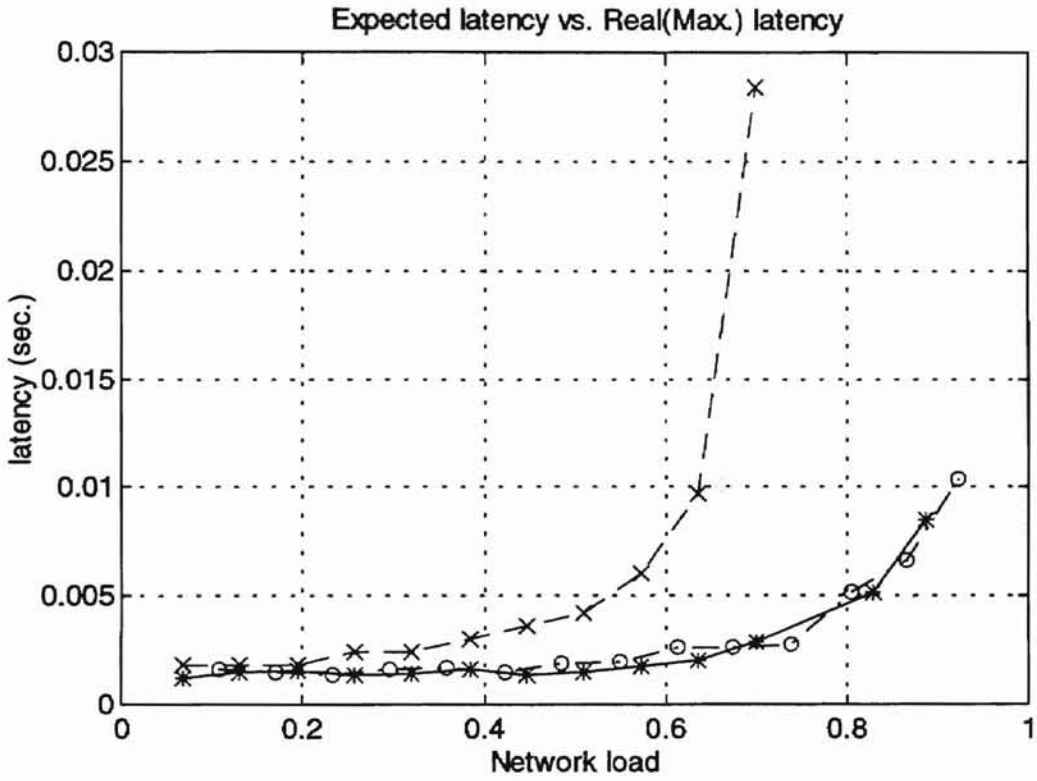
$$T_m = 151 + 151 + \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i} \right) \times 151 \right] + \left(\sum_{i \in p(m)} \left(\text{int}\left(\frac{T_m}{t_i} \right) \right) + 2 \right) \times P \times 23 +$$

$$\left(\sum_{i \in p(m)} \left(\text{int}\left(\frac{T_m}{t_i} \right) \right) + 2 \right) \times P \times 151$$

$$= 302 + \sum_{i \in p(m)} \left[\text{int}\left(\frac{T_m}{t_i} \right) \right] \times 151 + \left(\sum_{i \in p(m)} \left(\text{int}\left(\frac{T_m}{t_i} \right) \right) + 2 \right) \times 174 \times P \qquad (4.7)$$

Figure 4.20 shows the expected data latencies generated by Equation (4.7) and the real maximum data latencies in experiments 1 and 3 under different network loads. Please note that Equation (4.7) can only generate the expected latencies for experiments 1 and 3 under 70% network load because, beyond 70%, some of the input data are not reasonable for a real system. By the specifications of the experiments, there are only three stations on the CAN bus and we use the noise maker to generate different network loads by increasing the transmission rate of the messages. When the network load is higher than 70%, the transmission intervals of messages from the noise maker in experiments 1 and 3 are smaller than the maximum length of a CAN extended data frame, 151 (bit times), which means all lower priority messages will never get a change to be transmitted as long as the lengths of the "noise messages" are larger than their transmission intervals. Under these conditions, Equation (4.7) will never converge. The only reason we designed these unreasonable conditions is to investigate the data latencies. For a real CAN-based control

system, these conditions should never happen. We also notice that the difference between the expected latencies and the real maximum latencies increases when the network load is higher than 40%. The reason for this condition is that we use the maximum data length of every message in our expected latency formula (Equation 4.7).



x - - Expected latency * - Latency from experiment 1 o -. Latency from experiment 3

Figure 4.20 Expected latency/Real (Max.) latency plot (experiments 1 and 3)

2. For the "periodic noise signal" experiments where the messages from the transmitter have higher priority than the "noise messages".

$$T_m = T_{cmsg} + T_{msg_m} + N_{error} \times T_{errorframe+intermission} + \sum_{j \in E(m)} T_{msg_j} \qquad (4.8)$$

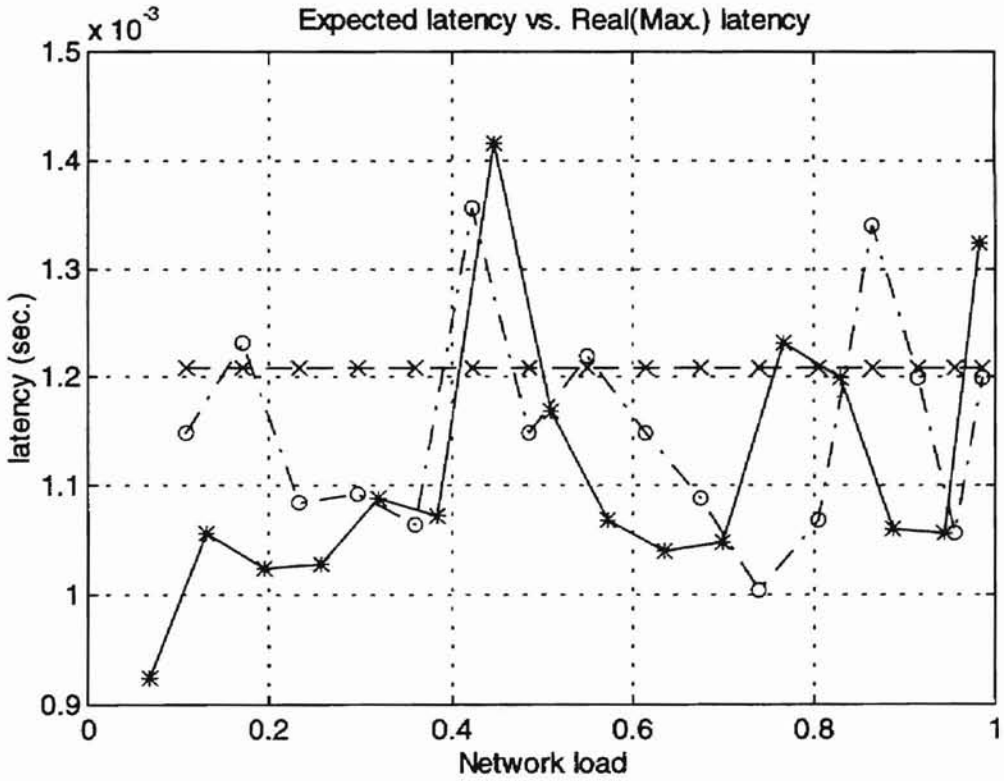
The third term is gone because the messages from the transmitter have the highest priority.

$N = 1$ and $N_{error} = N \times P$

$$T_m = 151+151+P\times 23+P\times 151$$

$$=302+174P \qquad (4.9)$$

From Equation (4.9), we can see that the expected data latency depends only on the transmission error rate. In the experiments, we set the error rate is a constant, 0.001, which means the expected latency curve is a constant line. Figure 4.21 shows the expected latency curve and the real maximum latency curves of experiments 2 and 4 under different network loads. At some points, the real maximum latencies are higher than the expected values. It is the same condition we mentioned previously that the number of real transmission errors occurring during a message transmission is higher than our assumption.



x - - Expected latency * - Latency from experiment 2 o -. Latency from experiment 4

Figure 4.21 Expected latency/Real (Max.) latency plot (experiments 2 and 4)

3. For the "sporadic noise signal" experiments where the messages from the message transmitter have lower priority than the "noise messages".

$$T_m = T_{cmsg} + T_{msg_m} + N_{error} \times T_{errorframe+intermission} + \sum_{j \in E(m)} T_{msg_j} + \sum_{k=1}^{N_{sp}} T_{smsg_k} \tag{4.10}$$

Please note that the third term is gone because the messages from the message transmitter are the only periodic signals in the system.

$$N_{error} = N \times P = (2 + N_{sp}) \times P$$

Assume G is the real number of sporadic signals occurring during T_m and there will not be any sporadic signal generated during the time to transmit the current sporadic signal. (Actually, this is what we define in our simulator.)

Then,

$$(T_m - G \times 151) \times Psp = G$$

$$\rightarrow \quad G = \left(\frac{T_m \times Psp}{1 + 151 \times Psp} \right) \tag{4.11}$$

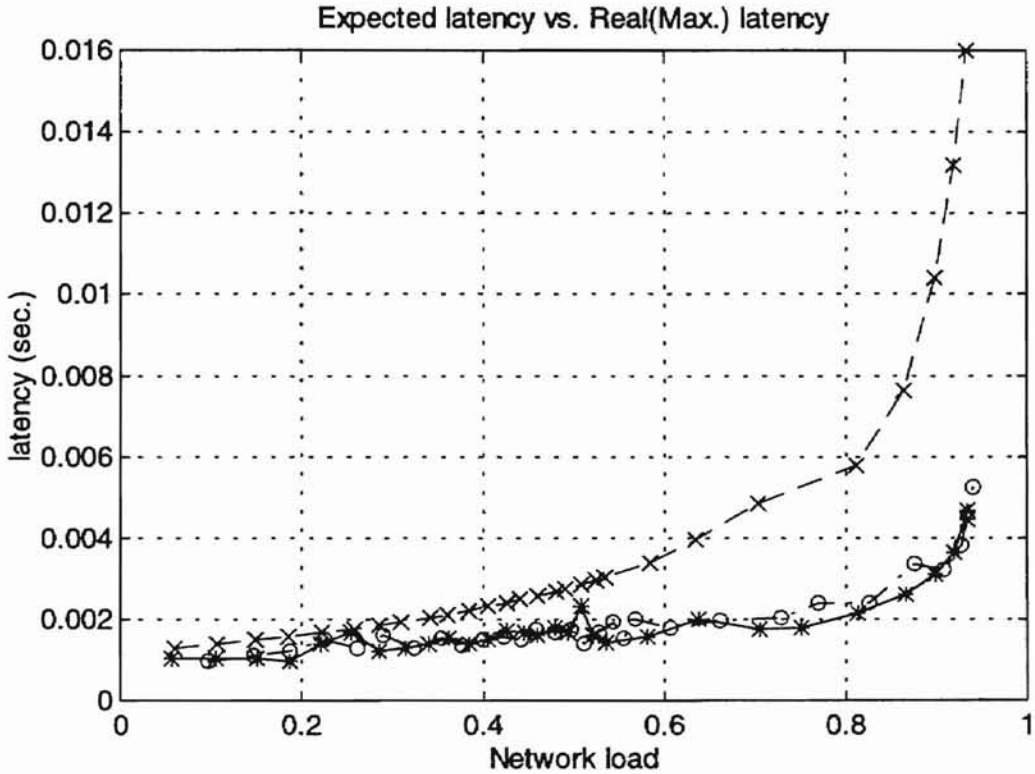
So, $N_{sp} = G$

$$T_m = 151 + 151 + (2 + N_{sp}) \times P \times 23 + (2 + N_{sp}) \times P \times 151 + N_{sp} \times 151$$

$$= 302 + 46P + 302P + 174 \times N_{sp} \times P + N_{sp} \times 151$$

$$= 302 + 348P + 174 \times \left(\frac{T_m \times Psp}{1 + 151 \times Psp} \right) \times P + 151 \times \left(\frac{T_m \times Psp}{1 + 151 \times Psp} \right) \tag{4.12}$$

From Equation (4.12), we can see that the message response time should increase slowly when *Psp* is small and it should increase rapidly as *Psp* increases to make the last two terms large. Figure 4.22 is the expected data latency plot generated by Equation (4.12) versus the real maximum latency plots of experiments 5 and 7. The curves are very close to the expected curve when the network load is under 40%. The differences result from the reason that we used maximum data length and the worst conditions in our formulae.



x - - Expected latency * - Latency from experiment 5 o -. Latency from experiment 7

Figure 4.22 Expected latency/Real (Max.) latency plot (experiments 5 and 7)

4. For the "sporadic noise signal" experiments where the messages from the message transmitter have higher priority than the "noise messages".

$$T_m = T_{cmsg} + T_{msg_m} + N_{error} \times T_{errorframe+intermission} + \sum_{j \in E(m)} T_{msg_j} \qquad (4.13)$$

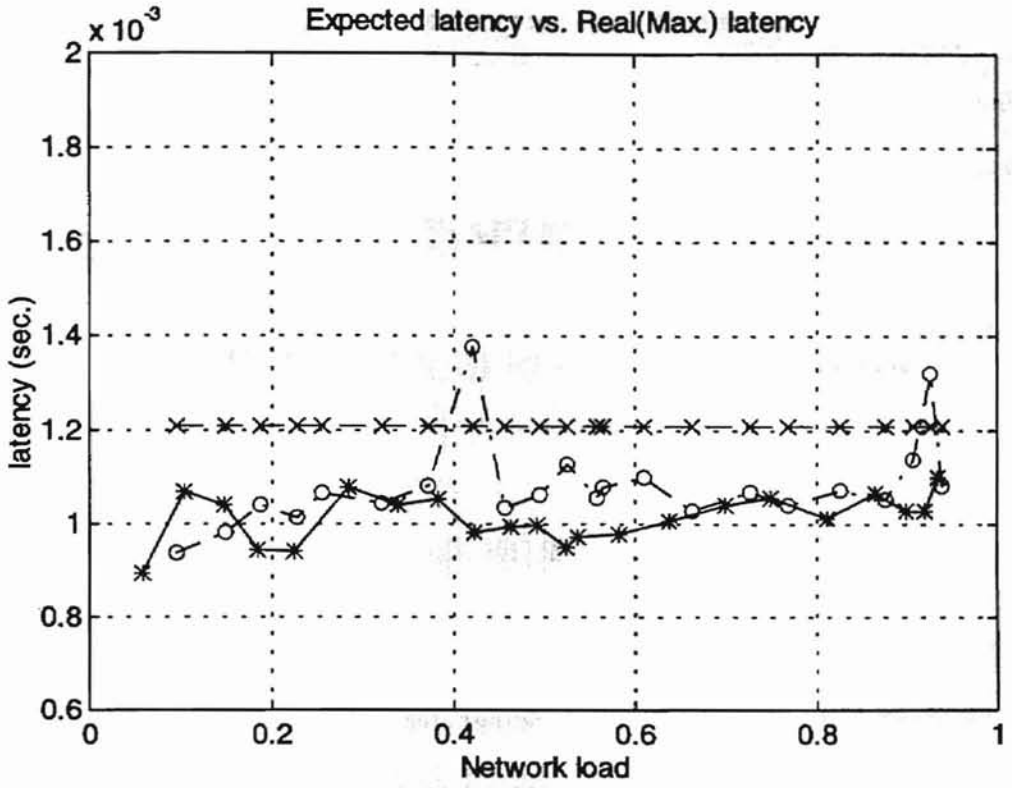
Please note that the sporadic signal term is gone because the messages from the message transmitter have higher priority than the sporadic signals.

$N_{error} = N \times P$ and $N = 1$ because the message m is the highest priority message in the system.

$$T_m = 151 + 151 + N_{error} \times 23 + N_{error} \times 151$$
$$= 302 + P \times 23 + P \times 151$$
$$= 302 + 174P \qquad (4.14)$$

Equation (4.14) is exactly the same as (4.9) which shows that the message response time depends on the value of P only. So, the expected message response time is a constant. This result is supported by the simulations. As we can see from Figures 4.17 and 4.19, both curves are nearly constant lines. Even though the network load almost reaches 95%, we don't see the data latencies increase dramatically. Figure 4.23 shows the expected latencies generated by Equation (4.14) and the real maximum latencies from experiments 6 and 8 under different network loads. Same as in Figure 4.21, the real maximum latencies are larger than the expected latencies at some points. The reason is exactly the same as that in experiments 2 and 4.

Table 4.3 shows the expected data latencies and the real maximum latencies in the experiments under 3 different network loads, 20%, 40%, and 60%.



x - - Expected latency * - Latency from experiment 6 o -. Latency from experiment 8

Figure 4.23 Expected latency/Real (Max.) latency plot (experiments 6 and 8)

Experiment	Network load		
	Expected/Max. latency(in Sec.)		
	20%	40%	60%
1	0.001812 / 0.001514	0.003128 / 0.001528	0.0062 / 0.001844
2	0.0012087 / 0.001026	0.0012087 / 0.00116	0.0012087 / 0.001056
3	0.001812 / 0.001417	0.003128 / 0.00154	0.0062 / 0.002485
4	0.0012087 / 0.001162	0.0012087 / 0.001249	0.0012087 / 0.001163
5	0.001595 / 0.001155	0.002285 / 0.001462	0.0035 / 0.001707
6	0.0012087 / 0.000942	0.0012087 / 0.00102	0.0012087 / 0.000983
7	0.001595 / 0.001322	0.002285 / 0.001493	0.0035 / 0.001805
8	0.0012087 / 0.00103	0.0012087 / 0.001249	0.0012087 / 0.001095

Table 4.3 Expected/Maximum latencies in experiments 1-8

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

Summary and conclusions

As high quality and inexpensive microprocessors are available, network-based control systems become more and more popular. System components are connected by a high-speed communication bus to share the system resource, transmit important messages, and save the cost of cabling and diagnostics. However, inevitable network-induced delays degrade the system performance and may even cause stability problems. Controller Area Network(CAN), a type of Local Area Network designed for use in control systems, was originally developed in the late 1980's. Because of its excellent performance, reliability, and robustness in harsh real-time environments, CAN has been proven to be an efficient network for real-time control systems.

Although it has been only 10 years since CAN was invented, some CAN research has been done. Most of studies focus on CAN protocol and only a general model is available for real-time analysis. In this study, a specific CAN message response time model has been presented from the real-time control point of view. It is well-suited to analyze network data latencies of real-time control systems because the model describes the

message response time by using system parameters of a CAN-based control system, which includes the sampling interval of each signal, the priority of each message, the number of stations or nodes, the transmission error rate, and the sporadic signal rate. The following questions have been addressed.

1. How should the transmission rate(or sampling period) of each message (sensor signal, control signal, etc.) on the CAN bus be chosen ?

   To design a good control system, control engineers need to choose appropriate parameters, like sampling periods of sensors and controllers, because they are directly related to system performance. When the control system works on a network, a more complex problem like network-induced time varying delays are involved. In order to analyze and design a control system to meet the minimum performance requirements, control engineers need to know how the transmission rate or sampling period of each message is related to the time varying delay. In this study, a CAN message response time analysis model has been presented by using system parameters of a CAN-based control system including the transmission rate of each message. Based on the model (Equation 3.2), specific formulae can be derived to calculate the expected message response time (data latency) of each message transmitted on the CAN bus for any specific CAN-based control system. With the information and the timing diagram of the control system, control engineers can analyze and even determine the appropriate sampling period of each message to achieve the minimum real-time performance requirements.

2. How should the priority of each message transmitted on the CAN bus be assigned ?

Both the simulation results and the model (Equation 3.2) show that priority assignment is another important problem of setting up a CAN-based control system. As mentioned previously in Chapter I, many priority assignment algorithms can be used to assign the priority of each message because priority assignment is a system dependent problem and is directly related to the data latency of each message. Control engineers can apply any appropriate algorithm to assign the priorities. However, different priority assignment will influence the data latencies (message delays). The more important thing is to know how the priority of each message is related to the message response time (data latency). The model (Equation 3.2) presented in Chapter III shows data latencies by using the priority of each message as an important system parameter. No matter what kind of priority assignment algorithm was used to assign the priority to each message, the impacts of different priority assignments can be found easily by the derived formulae. With the model and the derived formulae, control engineers can determine the appropriate priority assignment to meet system requirements.

3. How are the system parameters(sampling period, priority, network speed, number of stations, etc.) related to the data latencies ?

   The model (Equation 3.2) consists of all system parameters (sampling period and priority of each message, transmission rate and sporadic signal rate, network speed, number of stations, etc.). For any specific CAN-based control system, based on error rate and sporadic signal rate assumptions, control engineers can derive formulae to represent the data latencies by using the above system parameters. We illustrate the formula derivation in Chapter III (Equation 3.3 - 3.6 for a general control system with

0.001 error rate and 0.001 sporadic signal rate assumptions) and Chapter IV (Equation 4.7, 4.9, 4.12, and 4.14 for a specific system with the same error rate assumptions). The relationship between the parameters and the data latencies are specifically indicated by these formulae. They are also very important information for control system engineers to analyze the system performance.

4. Under what kind of conditions will the network fail to meet the real-time constraints ? Basically, this is a system dependent problem. Control engineers need to have some ideas or assumptions concerning the error rate and sporadic signal rate, and then use the model to derive expected message response time formulae. With all the above information and the formulae, control engineers can analyze the control system to see whether or not the appropriate system parameters have been chosen satisfactorily and whether or not the real-time constraints are too tight for the current system to meet. They may even predict the system performance and the maximum network limitation because all the real-time network behaviors of the CAN-based control system are shown in the derived formulae.

A bitwise CAN simulator has also been presented to simulate the network behaviors of CAN-based control systems. Many simulations have been run and presented in this study. The results support our model and comply with our predictions. The following is a summary and conclusions.

1. The major drawback of CAN protocol is that it fails to guarantee the maximum data latency for a message without the highest priority, which is also a major problem of all networks. Our simulation results support this problem. As we can see from our model (Equation 3.2), if there is no error rate assumption, theoretically, the message

response time can be infinite, which means that the message without the highest priority may never get a chance to be transmitted. This problem shows in the simulation results of the fourth, fifth, the sixth input data, and Figures 4.21 and 4.23. The actual retransmission number of the same message is higher than our assumption, which causes some message delays and in Figures 4.21 and 4.23, some real maximum latencies are larger than the expected latencies. Since the transmission error rate is the main problem which is system dependent, a suggestion is that we should investigate the network hardware before we set up the control system. At the same time, we can adjust the error rate based on our model and recalculate the message response time of each signal transmitted on the network.

2. From the simulation results, we find when the model suggests that the system parameters may be too tight for the system to meet the real-time constraints, it doesn't necessarily mean the network will fail in the real system. That's because our formulae always assume that the worst-case conditions happen in the real world. Although the expected message response times of some messages are larger than their deadlines and they may still meet the time constraints in the real system, for safety reasons, when using the model to analyze data latencies of control systems, we should always keep the real-time constraint of the message larger than its expected message response time.

Recommendations for further research work

As a control system becomes larger and more complex like a heavy duty vehicle which involves many subsystems, a single Controller Area Network is not capable to operate in

a real-time environment because more complex problems like the real-time communication problems between the subsystems become the main issues. The suggested solution of these problems is shown in Figure 5.1.

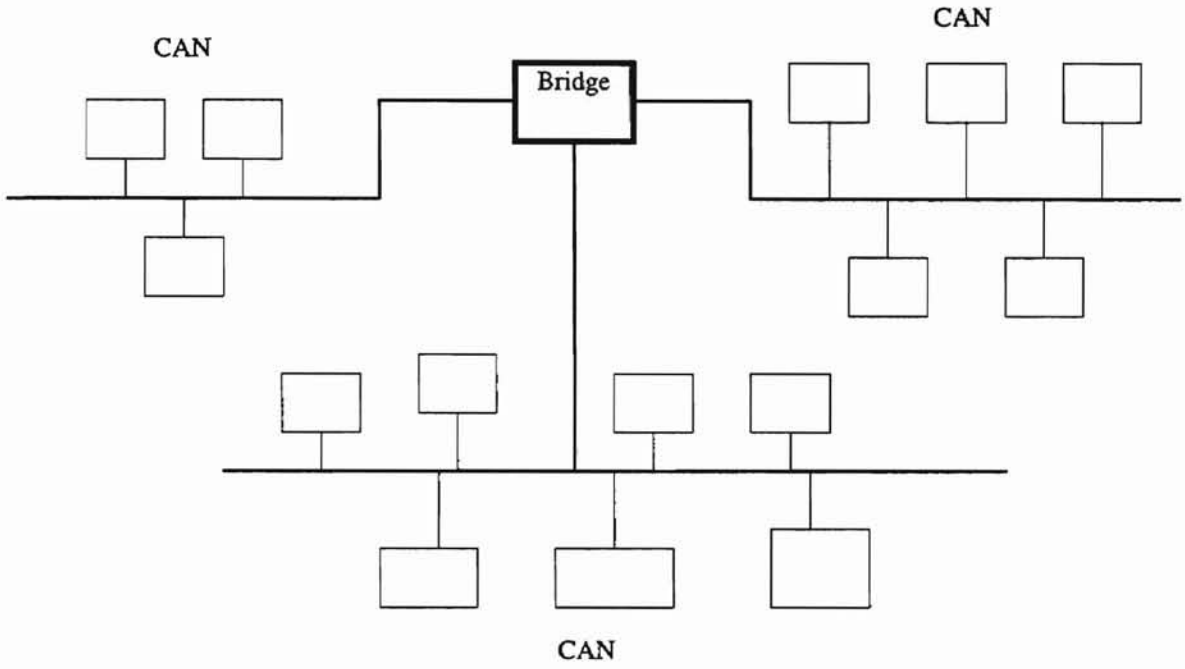


Figure5.1 A solution for a complex network-based control system

A special device called a Bridge is used to connect the subsystems(CAN) together. The Society of Automotive Engineers (SAE) documented a technical paper, SAE J1939, which presents a high speed communication protocol for On-Road heavy vehicles. Based on the CAN protocol, SAE J1939 added some different definitions in the Arbitration Field of a CAN Data Frame in order to specify the source address and destination address of a message and solve the identifier problem between the subsystems. Further research work following this study is to extend the model to accommodate bridges and subsystems for the analysis of complex distributed real-time control systems.

REFERENCES

1. Ray, A., "Introduction to Networking for Integrated Control Systems", IEEE Control Systems Magazine, Jan. 1989, pp. 76-78.
2. Omegas Co. UK., "Controller Area Network-Background Information". http:/www.omegas.co.uk/CAN/history.html (Sept. 1996)
3. Ray, A., "Distributed Data Communication Networks for Real-Time Process Control", Chemical Engineering Communications, Vol. 65, Mar. 1988, pp.139-154.
4. Tindell, K., "Analysis of Hard Real-Time Communications", Report of YCS222 in Real-Time Systems, May 1994. ftp://ftp.cs.york.ac.uk/pub/realtime/papers/INDEX
5. Ray, A., "Performance Evaluation of Medium Access Control Protocols for Distributed Digital Avionics", ASME Journal of Dynamic system, Measurement, and Control, Vol. 109, No.4, December 1987, pp.370-377.
6. Ray, A., Ayyagari, A., "Modelling and Analysis of a data Communication Protocol for Integrated Control of Advanced Aircraft", Computer Communication, Vol. 16, No. 6, Jun. 1993, pp. 350-365.
7. Halevi, Y., and Ray, A., "Integrated Communication and Control Systems : Part I - Analysis", Journal of Dynamic system, Measurement, and Control, Vol. 110, December 1988, pp. 367-373.
8. Ray, A., and Halevi, Y., "Integrated Communication and Control Systems : Part II - Design Considerations", Journal of Dynamic system, Measurement, and Control, Vol. 110, December 1988, pp. 374-381.

9. Wang, Z., Stone, M., Lu, H., Hedrick, G. E., "Message Delay Analysis for CAN based Networks", Proceedings of ACM Computer Science Conference, New York, March 3-5, 1992, pp.25-32.

10. Rauchhaupt, L., "Performance Analysis of CAN Based Systems", Proceedings of 1st International CAN Conference, Mainz 1994.
http://pmt05.et.uni-magdeburg.de/NT/Veroeff/CAN/icc94.html

11. Tindell, K., Burns, A., "Guaranteeing Message Latencies on Controller Area Network", Proceedings of the First International CAN Conference, Germany, September 1994.
ftp://ftp.cs.york.ac.uk/pub/realtime/papers/INDEX

12. Tindell, K., Burns, A., "Guaranteed Message Latencies For Distributed Safety-Critical Hard Real-Time Control Networks", Report of YCS229, May 1994.
ftp://ftp.cs.york.ac.uk/pub/realtime/papers/INDEX

13. Tindell, K., Burns, A., Wellings, A., "Calculating Controller Area Network(CAN) Message Response Times", Proceedings 1994 IFAC Workshop on Discributed Control System(DCCS), Toledo, Spain, September 1994.
ftp://ftp.cs.york.ac.uk/pub/realtime/papers/INDEX

14. Cena, G., Valenzano, A., "A Distributed Mechanism to Improve Fairness in CAN Networks", Proceedings of IEEE International Workshop on Factory Communication Systems, Switzerland, 1995. pp.3-11.

15. Wang, Z., Lu, H., Stone, M., "A Message Priority Assignment Algorithm For CAN Based Networks", Proceedings of the 20th Annual Computer Science Conference,

Kansas City, MO, March 3-5, 1992, pp.25-32.

16. Liu, C. L., Layland, J. W., "Scheduling Algorithm for Multiprogramming in a Hard Real Time Environment", Journal of the Association for Computer Machinery, Vol. 20, No. 1, January 1973, pp.46-61.

17. Pennathur, N. S., "A Bitwise Simulation of The Controller Area Network", M.S. Thesis, Department of Computer Science, Oklahoma State University, Stillwater, OK, December 1993.

18. Leu, T. J., "Performance Analysis of A Controller Area Network Subject to Asymmetric Traffic Loads", M.S. Thesis, Department of Computer Science, Oklahoma State University, Stillwater, OK, July 1994.

19. Li, W., "Hard Real-Time Communications in Controller Area Network", M.S. Thesis, Department of Computer Science, Oklahoma State University, Stillwater, OK, December 1995.

20. Robert Bosch GmbH Co., "CAN Specification Version 2.0", September, 1991. http://design-net.com/csic/GLOSSARY/CAN.htm

21. Omegas Co. UK. "Controller Area Network-CAN Application Layers" http:/www.omegas.co.uk/CAN/osimodel.html(Sep. 1996)

22. Omegas Co. UK. "Controller Area Network-Error Handling". http//www.omegas.co.uk/CAN/errors.html (Sep. 1996)

APPENDIXES

APPENDIX A

INPUT DATA FILES

The format of input data file

Simulation time (in seconds)
Network speed (bandwidth in kilobits per second)
The number of stations(nodes) on the CAN bus
Probability of Overload Frames occurring in total simulation time (I)
Probability of Overload Frames occurring in total simulation time (II)
Probability of Form(Frame) errors occurring in total simulation time (I)
Probability of Form(Frame) errors occurring in total simulation time (II)
Probability of Form(Frame) errors occurring in total simulation time (III)
Probability of Form(Frame) errors occurring in total simulation time (IV)
Probability of Bit Stuffing errors occurring in total simulation time (I)
Probability of Bit Stuffing errors occurring in total simulation time (II)
Probability of Bit Stuffing errors occurring in total simulation time (III)
Probability of Bit Stuffing errors occurring in total simulation time (IV)
Probability of CRC errors occurring in total simulation time (I)
Probability of CRC errors occurring in total simulation time (II)
Probability of CRC errors occurring in total simulation time (III)
Probability of CRC errors occurring in total simulation time (IV)
Probability of ACK errors occurring in total simulation time (I)
Probability of ACK errors occurring in total simulation time (II)
Probability of ACK errors occurring in total simulation time (III)
Probability of ACK errors occurring in total simulation time (IV)

(The real probability of Overload Frames occurring is equal to $\frac{(I)}{(II)}$ and the real probability of each Error Frame occurring is $\frac{(I)}{(II)} \times \frac{(III)}{(IV)}$.)

(For periodic signals)
Node name Sampling period(in seconds) Sampling start instant(in seconds) Time skew(in seconds) Message identifier Data frame type The number of messages received Message Masks(id.)

(For sporadic signals)
Node name 0 (Sampling period) Probability of sporadic signal(I) Probability of sporadic signal(II) Time skew(in seconds) Message identifier Data frame type The number of messages received Message Masks(id.)

(The real probability of sporadic signals occurring is equal to $\frac{(I)}{(II)}$.)

The first input data file : (5 nodes)

```
10
250
5
0
10000
119
10000
10
10000
119
10000
10
10000
119
10000
10
10000
119
10000
10
10000
sensr1 0.05 0 0.001164 3 0 1 17
cntrlr1 0.05 0.001164 0.048836 6 0 1 3
sensr2 0.04 0 0.001744 4 0 1 17
cntrlr2 0.04 0.001744 0.038256 5 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

The second input data file : (7 nodes)

```
10
250
7
0
10000
431
10000
2
10000
431
10000
2
10000
431
10000
2
10000
431
10000
2
10000
sensr1 0.05 0 0.001164 3 0 1 17
cntrlr1 0.05 0.001164 0.048836 8 0 1 3
sensr2 0.04 0 0.001744 4 0 1 17
cntrlr2 0.04 0.001744 0.038256 7 0 1 3
sensr3 0.03 0 0.002328 5 0 1 17
cntrlr3 0.03 0.002328 0.027672 6 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

The third input data file : (9 nodes)

```
10
250
9
0
10000
169
10000
4
10000
169
10000
4
10000
169
10000
4
10000
169
10000
4
10000
sensr1 0.05 0 0.001164 3 0 1 17
cntrlr1 0.05 0.001164 0.048836 10 0 1 3
sensr2 0.04 0 0.001744 4 0 1 17
cntrlr2 0.04 0.001744 0.038256 9 0 1 3
sensr3 0.03 0 0.002328 5 0 1 17
cntrlr3 0.03 0.002328 0.027672 8 0 1 3
sensr4 0.02 0 0.002912 6 0 1 17
cntrlr4 0.02 0.002912 0.017088 7 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

The fourth input data file : (11 nodes)

```
10
250
11
0
10000
139
10000
4
10000
139
10000
4
10000
139
10000
4
10000
139
10000
4
10000
sensr1 0.05 0 0.001164 3 0 1 17
cntrlr1 0.05 0.001164 0.048836 12 0 1 3
sensr2 0.04 0 0.001744 4 0 1 17
cntrlr2 0.04 0.001744 0.038256 11 0 1 3
sensr3 0.03 0 0.002328 5 0 1 17
cntrlr3 0.03 0.002328 0.027672 10 0 1 3
sensr4 0.02 0 0.002912 6 0 1 17
cntrlr4 0.02 0.002912 0.017088 9 0 1 3
sensr5 0.01 0 0.003492 7 0 1 17
cntrlr5 0.01 0.003492 0.006508 8 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

The fifth input data file : (13 nodes)

```
10
250
13
0
10000
59
10000
8
10000
59
10000
8
10000
59
10000
8
10000
59
10000
8
10000
sensr1 0.05 0 0.001164 3 0 1 17
cntrlr1 0.05 0.001164 0.048836 14 0 1 3
sensr2 0.04 0 0.001744 4 0 1 17
cntrlr2 0.04 0.001744 0.038256 13 0 1 3
sensr3 0.03 0 0.002328 5 0 1 17
cntrlr3 0.03 0.002328 0.027672 12 0 1 3
sensr4 0.02 0 0.002912 6 0 1 17
cntrlr4 0.02 0.002912 0.017088 11 0 1 3
sensr5 0.01 0 0.003492 7 0 1 17
cntrlr5 0.01 0.003492 0.006508 10 0 1 3
sensr6 0.008 0 0.004076 8 0 1 17
cntrlr6 0.008 0.004076 0.003924 9 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

The sixth input data file : (13 nodes)

```
10
250
13
0
10000
59
10000
8
10000
59
10000
8
10000
59
10000
8
10000
59
10000
8
10000
sensr1 0.05 0 0.001164 3 0 1 17
cntrlr1 0.05 0.001164 0.048836 14 0 1 3
sensr2 0.04 0 0.001744 4 0 1 17
cntrlr2 0.04 0.001744 0.038256 13 0 1 3
sensr3 0.03 0 0.002328 5 0 1 17
cntrlr3 0.03 0.002328 0.027672 12 0 1 3
sensr4 0.01 0 0.002912 6 0 1 17
cntrlr4 0.01 0.002912 0.007088 11 0 1 3
sensr5 0.01 0 0.003492 7 0 1 17
cntrlr5 0.01 0.003492 0.006508 10 0 1 3
sensr6 0.008 0 0.004076 8 0 1 17
cntrlr6 0.008 0.004076 0.003924 9 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

The seventh input data file : (13 nodes)

```
10
250
13
0
10000
59
10000
8
10000
59
10000
8
10000
59
10000
8
10000
59
10000
8
10000
sensr1 0.05 0 0.001164 3 0 1 17
cntrlr1 0.05 0.001164 0.048836 14 0 1 3
sensr2 0.04 0 0.001744 4 0 1 17
cntrlr2 0.04 0.001744 0.038256 13 0 1 3
sensr3 0.03 0 0.002328 5 0 1 17
cntrlr3 0.03 0.002328 0.027672 12 0 1 3
sensr4 0.01 0 0.002912 6 0 1 17
cntrlr4 0.01 0.002912 0.007088 11 0 1 3
sensr5 0.008 0 0.003492 7 0 1 17
cntrlr5 0.008 0.003492 0.004508 10 0 1 3
sensr6 0.008 0 0.004076 8 0 1 17
cntrlr6 0.008 0.004076 0.003924 9 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

The eighth input data file : (13 nodes)

```
10
250
13
0
10000
59
10000
8
10000
59
10000
8
10000
59
10000
8
10000
59
10000
8
10000
sensr1 0.05 0 0.001164 3 0 1 17
cntrlr1 0.05 0.001164 0.048836 14 0 1 3
sensr2 0.04 0 0.001744 4 0 1 17
cntrlr2 0.04 0.001744 0.038256 13 0 1 3
sensr3 0.02 0 0.002328 5 0 1 17
cntrlr3 0.02 0.002328 0.017672 12 0 1 3
sensr4 0.01 0 0.002912 6 0 1 17
cntrlr4 0.01 0.002912 0.007088 11 0 1 3
sensr5 0.008 0 0.003492 7 0 1 17
cntrlr5 0.008 0.003492 0.004508 10 0 1 3
sensr6 0.008 0 0.004076 8 0 1 17
cntrlr6 0.008 0.004076 0.003924 9 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

The ninth input data file : (15 nodes)

```
10
250
15
0
10000
41
10000
10
10000
41
10000
10
10000
41
10000
10
10000
41
10000
10
10000
sensr1 0.05 0 0.001164 3 0 1 17
cntrlr1 0.05 0.001164 0.048836 16 0 1 3
sensr2 0.02 0 0.001744 4 0 1 17
cntrlr2 0.02 0.001744 0.018256 15 0 1 3
sensr3 0.01 0 0.002328 5 0 1 17
cntrlr3 0.01 0.002328 0.007672 14 0 1 3
sensr4 0.01 0 0.002912 6 0 1 17
cntrlr4 0.01 0.002912 0.007088 13 0 1 3
sensr5 0.015 0 0.003492 7 0 1 17
cntrlr5 0.015 0.003492 0.011508 12 0 1 3
sensr6 0.0085 0 0.004076 8 0 1 17
cntrlr6 0.0085 0.004076 0.004424 11 0 1 3
sensr7 0.0085 0 0.00466 9 0 1 17
cntrlr7 0.0085 0.00466 0.00384 10 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

The tenth input data file : (15 nodes)

```
10
250
15
0
10000
41
10000
10
10000
41
10000
10
10000
41
10000
10
10000
41
10000
10
10000
sensr1 0.016 0 0.001164 3 0 1 17
cntrlr1 0.016 0.001164 0.014836 16 0 1 3
sensr2 0.015 0 0.001744 4 0 1 17
cntrlr2 0.015 0.001744 0.013256 15 0 1 3
sensr3 0.012 0 0.002328 5 0 1 17
cntrlr3 0.012 0.002328 0.009672 14 0 1 3
sensr4 0.01 0 0.002912 6 0 1 17
cntrlr4 0.01 0.002912 0.007088 13 0 1 3
sensr5 0.01 0 0.003492 7 0 1 17
cntrlr5 0.01 0.003492 0.006508 12 0 1 3
sensr6 0.009 0 0.004076 8 0 1 17
cntrlr6 0.009 0.004076 0.004924 11 0 1 3
sensr7 0.0085 0 0.00466 9 0 1 17
cntrlr7 0.0085 0.00466 0.00384 10 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

The eleventh input data file : (15 nodes)

```
10
250
15
0
10000
41
10000
10
10000
41
10000
10
10000
41
10000
10
10000
41
10000
10
10000
sensr1 0.016 0 0.001164 3 0 1 17
cntrlr1 0.016 0.001164 0.014836 16 0 1 3
sensr2 0.01 0 0.001744 4 0 1 17
cntrlr2 0.01 0.001744 0.008256 15 0 1 3
sensr3 0.01 0 0.002328 5 0 1 17
cntrlr3 0.01 0.002328 0.007672 14 0 1 3
sensr4 0.01 0 0.002912 6 0 1 17
cntrlr4 0.01 0.002912 0.007088 13 0 1 3
sensr5 0.01 0 0.003492 7 0 1 17
cntrlr5 0.01 0.003492 0.006508 12 0 1 3
sensr6 0.009 0 0.004076 8 0 1 17
cntrlr6 0.009 0.004076 0.004924 11 0 1 3
sensr7 0.0085 0 0.00466 9 0 1 17
cntrlr7 0.0085 0.00466 0.00384 10 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

The twelfth input data file : (15 nodes)

```
10
250
15
0
10000
41
10000
10
10000
41
10000
10
10000
41
10000
10
10000
41
10000
10
10000
sensr1 0.012 0 0.001164 3 0 1 17
cntrlr1 0.012 0.001164 0.010836 16 0 1 3
sensr2 0.0097 0 0.001744 4 0 1 17
cntrlr2 0.0097 0.001744 0.007956 15 0 1 3
sensr3 0.0095 0 0.002328 5 0 1 17
cntrlr3 0.0095 0.002328 0.007172 14 0 1 3
sensr4 0.0093 0 0.002912 6 0 1 17
cntrlr4 0.0093 0.002912 0.006388 13 0 1 3
sensr5 0.0091 0 0.003492 7 0 1 17
cntrlr5 0.0091 0.003492 0.005608 12 0 1 3
sensr6 0.009 0 0.004076 8 0 1 17
cntrlr6 0.009 0.004076 0.004924 11 0 1 3
sensr7 0.0085 0 0.00466 9 0 1 17
cntrlr7 0.0085 0.00466 0.00384 10 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

The thirteenth input data file : (15 nodes)

```
10
250
15
0
10000
41
10000
10
10000
41
10000
10
10000
41
10000
10
10000
41
10000
10
10000
sensr1 0.01 0 0.001164 3 0 1 17
cntrlr1 0.01 0.001164 0.008836 16 0 1 3
sensr2 0.0093 0 0.001744 4 0 1 17
cntrlr2 0.0093 0.001744 0.007556 15 0 1 3
sensr3 0.0091 0 0.002328 5 0 1 17
cntrlr3 0.0091 0.002328 0.006772 14 0 1 3
sensr4 0.009 0 0.002912 6 0 1 17
cntrlr4 0.009 0.002912 0.006088 13 0 1 3
sensr5 0.009 0 0.003492 7 0 1 17
cntrlr5 0.009 0.003492 0.005508 12 0 1 3
sensr6 0.0087 0 0.004076 8 0 1 17
cntrlr6 0.0087 0.004076 0.004624 11 0 1 3
sensr7 0.0085 0 0.00466 9 0 1 17
cntrlr7 0.0085 0.00466 0.00384 10 0 1 3
emg1 0 10 10000 0.0010167 1 0 1 0
```

APPENDIX B

RESULTS OF MESSAGE RESPONSE TIME CALCULATIONS

Expected message response time calculation for the first input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):4
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.05
Input the sampling rate of the priority 2 message (in Sec.):0.04
Input the sampling rate of the priority 3 message (in Sec.):0.04
Input the sampling rate of the priority 4 message (in Sec.):0.05
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected message response time of priority (1):0.0011640
Expected message response time of priority (2):0.0017440
Expected message response time of priority (3):0.0023280
Expected message response time of priority (4):0.0029120

Expected message response time calculation for the second input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):6
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.05
Input the sampling rate of the priority 2 message (in Sec.):0.04
Input the sampling rate of the priority 3 message (in Sec.):0.03
Input the sampling rate of the priority 4 message (in Sec.):0.03
Input the sampling rate of the priority 5 message (in Sec.):0.04
Input the sampling rate of the priority 6 message (in Sec.):0.05
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected bit time of priority (5) : 873
Expected bit time of priority (6) : 1019
Expected message response time of priority (1):0.0011640
Expected message response time of priority (2):0.0017440
Expected message response time of priority (3):0.0023280
Expected message response time of priority (4):0.0029120
Expected message response time of priority (5):0.0034920
Expected message response time of priority (6):0.0040760

Expected message response time calculation for the third input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):8
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.05
Input the sampling rate of the priority 2 message (in Sec.):0.04
Input the sampling rate of the priority 3 message (in Sec.):0.03
Input the sampling rate of the priority 4 message (in Sec.):0.02
Input the sampling rate of the priority 5 message (in Sec.):0.02
Input the sampling rate of the priority 6 message (in Sec.):0.03
Input the sampling rate of the priority 7 message (in Sec.):0.04
Input the sampling rate of the priority 8 message (in Sec.):0.05
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected bit time of priority (5) : 873
Expected bit time of priority (6) : 1019
Expected bit time of priority (7) : 1165
Expected bit time of priority (8) : 1310
Expected message response time of priority (1):0.0011640
Expected message response time of priority (2):0.0017440
Expected message response time of priority (3):0.0023280
Expected message response time of priority (4):0.0029120
Expected message response time of priority (5):0.0034920
Expected message response time of priority (6):0.0040760
Expected message response time of priority (7):0.0046600
Expected message response time of priority (8):0.0052400

Expected message response time calculation for the fourth input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):10
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.05
Input the sampling rate of the priority 2 message (in Sec.):0.04
Input the sampling rate of the priority 3 message (in Sec.):0.03
Input the sampling rate of the priority 4 message (in Sec.):0.02
Input the sampling rate of the priority 5 message (in Sec.):0.01
Input the sampling rate of the priority 6 message (in Sec.):0.01
Input the sampling rate of the priority 7 message (in Sec.):0.02
Input the sampling rate of the priority 8 message (in Sec.):0.03
Input the sampling rate of the priority 9 message (in Sec.):0.04
Input the sampling rate of the priority 10 message (in Sec.):0.05
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected bit time of priority (5) : 873
Expected bit time of priority (6) : 1019
Expected bit time of priority (7) : 1165
Expected bit time of priority (8) : 1310
Expected bit time of priority (9) : 1456
Expected bit time of priority (10) : 1601
Expected message response time of priority (1):0.0011640
Expected message response time of priority (2):0.0017440
Expected message response time of priority (3):0.0023280
Expected message response time of priority (4):0.0029120
Expected message response time of priority (5):0.0034920
Expected message response time of priority (6):0.0040760
Expected message response time of priority (7):0.0046600
Expected message response time of priority (8):0.0052400
Expected message response time of priority (9):0.0058240
Expected message response time of priority (10):0.0064040

Expected message response time calculation for the fifth input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):12
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.05
Input the sampling rate of the priority 2 message (in Sec.):0.04
Input the sampling rate of the priority 3 message (in Sec.):0.03
Input the sampling rate of the priority 4 message (in Sec.):0.02
Input the sampling rate of the priority 5 message (in Sec.):0.01
Input the sampling rate of the priority 6 message (in Sec.):0.008
Input the sampling rate of the priority 7 message (in Sec.):0.008
Input the sampling rate of the priority 8 message (in Sec.):0.01
Input the sampling rate of the priority 9 message (in Sec.):0.02
Input the sampling rate of the priority 10 message (in Sec.):0.03
Input the sampling rate of the priority 11 message (in Sec.):0.04
Input the sampling rate of the priority 12 message (in Sec.):0.05
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected bit time of priority (5) : 873
Expected bit time of priority (6) : 1019
Expected bit time of priority (7) : 1165
Expected bit time of priority (8) : 1310
Expected bit time of priority (9) : 1456
Expected bit time of priority (10) : 1601
Expected bit time of priority (11) : 1747
Expected bit time of priority (12) : 1893
Expected message response time of priority (1):0.0011640
Expected message response time of priority (2):0.0017440
Expected message response time of priority (3):0.0023280
Expected message response time of priority (4):0.0029120
Expected message response time of priority (5):0.0034920
Expected message response time of priority (6):0.0040760
Expected message response time of priority (7):0.0046600
Expected message response time of priority (8):0.0052400
Expected message response time of priority (9):0.0058240
Expected message response time of priority (10):0.0064040
Expected message response time of priority (11):0.0069880
Expected message response time of priority (12):0.0075720

Expected message response time calculation for the sixth input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):12
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.05
Input the sampling rate of the priority 2 message (in Sec.):0.04
Input the sampling rate of the priority 3 message (in Sec.):0.03
Input the sampling rate of the priority 4 message (in Sec.):0.01
Input the sampling rate of the priority 5 message (in Sec.):0.01
Input the sampling rate of the priority 6 message (in Sec.):0.008
Input the sampling rate of the priority 7 message (in Sec.):0.008
Input the sampling rate of the priority 8 message (in Sec.):0.01
Input the sampling rate of the priority 9 message (in Sec.):0.01
Input the sampling rate of the priority 10 message (in Sec.):0.03
Input the sampling rate of the priority 11 message (in Sec.):0.04
Input the sampling rate of the priority 12 message (in Sec.):0.05
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected bit time of priority (5) : 873
Expected bit time of priority (6) : 1019
Expected bit time of priority (7) : 1165
Expected bit time of priority (8) : 1310
Expected bit time of priority (9) : 1456
Expected bit time of priority (10) : 1601
Expected bit time of priority (11) : 1747
Expected bit time of priority (12) : 1893
Expected message response time of priority (1): 0.0011640
Expected message response time of priority (2): 0.0017440
Expected message response time of priority (3): 0.0023280
Expected message response time of priority (4): 0.0029120
Expected message response time of priority (5): 0.0034920
Expected message response time of priority (6): 0.0040760
Expected message response time of priority (7): 0.0046600
Expected message response time of priority (8): 0.0052400
Expected message response time of priority (9): 0.0058240
Expected message response time of priority (10): 0.0064040
Expected message response time of priority (11): 0.0069880
Expected message response time of priority (12): 0.0075720

Expected message response time calculation for the seventh input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):12
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.05
Input the sampling rate of the priority 2 message (in Sec.):0.04
Input the sampling rate of the priority 3 message (in Sec.):0.03
Input the sampling rate of the priority 4 message (in Sec.):0.01
Input the sampling rate of the priority 5 message (in Sec.):0.008
Input the sampling rate of the priority 6 message (in Sec.):0.008
Input the sampling rate of the priority 7 message (in Sec.):0.008
Input the sampling rate of the priority 8 message (in Sec.):0.008
Input the sampling rate of the priority 9 message (in Sec.):0.01
Input the sampling rate of the priority 10 message (in Sec.):0.03
Input the sampling rate of the priority 11 message (in Sec.):0.04
Input the sampling rate of the priority 12 message (in Sec.):0.05
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected bit time of priority (5) : 873
Expected bit time of priority (6) : 1019
Expected bit time of priority (7) : 1165
Expected bit time of priority (8) : 1310
Expected bit time of priority (9) : 1456
Expected bit time of priority (10) : 1601
Expected bit time of priority (11) : 1747
Expected bit time of priority (12) : 1893
Expected message response time of priority (1): 0.0011640
Expected message response time of priority (2): 0.0017440
Expected message response time of priority (3): 0.0023280
Expected message response time of priority (4): 0.0029120
Expected message response time of priority (5): 0.0034920
Expected message response time of priority (6): 0.0040760
Expected message response time of priority (7): 0.0046600
Expected message response time of priority (8): 0.0052400
Expected message response time of priority (9): 0.0058240
Expected message response time of priority (10): 0.0064040
Expected message response time of priority (11): 0.0069880
Expected message response time of priority (12): 0.0075720

Expected message response time calculation for the eighth input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):12
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.05
Input the sampling rate of the priority 2 message (in Sec.):0.04
Input the sampling rate of the priority 3 message (in Sec.):0.02
Input the sampling rate of the priority 4 message (in Sec.):0.01
Input the sampling rate of the priority 5 message (in Sec.):0.008
Input the sampling rate of the priority 6 message (in Sec.):0.008
Input the sampling rate of the priority 7 message (in Sec.):0.008
Input the sampling rate of the priority 8 message (in Sec.):0.008
Input the sampling rate of the priority 9 message (in Sec.):0.01
Input the sampling rate of the priority 10 message (in Sec.):0.02
Input the sampling rate of the priority 11 message (in Sec.):0.04
Input the sampling rate of the priority 12 message (in Sec.):0.05
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected bit time of priority (5) : 873
Expected bit time of priority (6) : 1019
Expected bit time of priority (7) : 1165
Expected bit time of priority (8) : 1310
Expected bit time of priority (9) : 1456
Expected bit time of priority (10) : 1601
Expected bit time of priority (11) : 1747
Expected bit time of priority (12) : 1893
Expected message response time of priority (1): 0.0011640
Expected message response time of priority (2): 0.0017440
Expected message response time of priority (3): 0.0023280
Expected message response time of priority (4): 0.0029120
Expected message response time of priority (5): 0.0034920
Expected message response time of priority (6): 0.0040760
Expected message response time of priority (7): 0.0046600
Expected message response time of priority (8): 0.0052400
Expected message response time of priority (9): 0.0058240
Expected message response time of priority (10): 0.0064040
Expected message response time of priority (11): 0.0069880
Expected message response time of priority (12): 0.0075720

Expected message response time calculation for the ninth input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):14
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.05
Input the sampling rate of the priority 2 message (in Sec.):0.02
Input the sampling rate of the priority 3 message (in Sec.):0.01
Input the sampling rate of the priority 4 message (in Sec.):0.01
Input the sampling rate of the priority 5 message (in Sec.):0.015
Input the sampling rate of the priority 6 message (in Sec.):0.0085
Input the sampling rate of the priority 7 message (in Sec.):0.0085
Input the sampling rate of the priority 8 message (in Sec.):0.0085
Input the sampling rate of the priority 9 message (in Sec.):0.0085
Input the sampling rate of the priority 10 message (in Sec.):0.015
Input the sampling rate of the priority 11 message (in Sec.):0.01
Input the sampling rate of the priority 12 message (in Sec.):0.01
Input the sampling rate of the priority 13 message (in Sec.):0.02
Input the sampling rate of the priority 14 message (in Sec.):0.05
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected bit time of priority (5) : 873
Expected bit time of priority (6) : 1019
Expected bit time of priority (7) : 1165
Expected bit time of priority (8) : 1310
Expected bit time of priority (9) : 1456
Expected bit time of priority (10) : 1601
Expected bit time of priority (11) : 1747
Expected bit time of priority (12) : 1893
Expected bit time of priority (13) : 2038
Expected bit time of priority (14) : 3349
Expected message response time of priority (1): 0.0011640
Expected message response time of priority (2): 0.0017440
Expected message response time of priority (3): 0.0023280
Expected message response time of priority (4): 0.0029120
Expected message response time of priority (5): 0.0034920
Expected message response time of priority (6): 0.0040760
Expected message response time of priority (7): 0.0046600
Expected message response time of priority (8): 0.0052400
Expected message response time of priority (9): 0.0058240
Expected message response time of priority (10): 0.0064040
Expected message response time of priority (11): 0.0069880
Expected message response time of priority (12): 0.0075720

Expected message response time of priority (13): 0.0081520
Expected message response time of priority (14): 0.0133960

Expected message response time calculation for the tenth input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):14
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.016
Input the sampling rate of the priority 2 message (in Sec.):0.015
Input the sampling rate of the priority 3 message (in Sec.):0.012
Input the sampling rate of the priority 4 message (in Sec.):0.01
Input the sampling rate of the priority 5 message (in Sec.):0.01
Input the sampling rate of the priority 6 message (in Sec.):0.009
Input the sampling rate of the priority 7 message (in Sec.):0.0085
Input the sampling rate of the priority 8 message (in Sec.):0.0085
Input the sampling rate of the priority 9 message (in Sec.):0.009
Input the sampling rate of the priority 10 message (in Sec.):0.01
Input the sampling rate of the priority 11 message (in Sec.):0.01
Input the sampling rate of the priority 12 message (in Sec.):0.012
Input the sampling rate of the priority 13 message (in Sec.):0.015
Input the sampling rate of the priority 14 message (in Sec.):0.016
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected bit time of priority (5) : 873
Expected bit time of priority (6) : 1019
Expected bit time of priority (7) : 1165
Expected bit time of priority (8) : 1310
Expected bit time of priority (9) : 1456
Expected bit time of priority (10) : 1601
Expected bit time of priority (11) : 1747
Expected bit time of priority (12) : 1893
Expected bit time of priority (13) : 2038
Expected bit time of priority (14) : 3641
Expected message response time of priority (1): 0.0011640
Expected message response time of priority (2): 0.0017440
Expected message response time of priority (3): 0.0023280
Expected message response time of priority (4): 0.0029120
Expected message response time of priority (5): 0.0034920
Expected message response time of priority (6): 0.0040760
Expected message response time of priority (7): 0.0046600
Expected message response time of priority (8): 0.0052400
Expected message response time of priority (9): 0.0058240
Expected message response time of priority (10): 0.0064040
Expected message response time of priority (11): 0.0069880
Expected message response time of priority (12): 0.0075720

Expected message response time of priority (13): 0.0081520
Expected message response time of priority (14): 0.0145640

Expected message response time calculation for the eleventh input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):14
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.016
Input the sampling rate of the priority 2 message (in Sec.):0.01
Input the sampling rate of the priority 3 message (in Sec.):0.01
Input the sampling rate of the priority 4 message (in Sec.):0.01
Input the sampling rate of the priority 5 message (in Sec.):0.01
Input the sampling rate of the priority 6 message (in Sec.):0.009
Input the sampling rate of the priority 7 message (in Sec.):0.0085
Input the sampling rate of the priority 8 message (in Sec.):0.0085
Input the sampling rate of the priority 9 message (in Sec.):0.009
Input the sampling rate of the priority 10 message (in Sec.):0.01
Input the sampling rate of the priority 11 message (in Sec.):0.01
Input the sampling rate of the priority 12 message (in Sec.):0.01
Input the sampling rate of the priority 13 message (in Sec.):0.01
Input the sampling rate of the priority 14 message (in Sec.):0.016
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected bit time of priority (5) : 873
Expected bit time of priority (6) : 1019
Expected bit time of priority (7) : 1165
Expected bit time of priority (8) : 1310
Expected bit time of priority (9) : 1456
Expected bit time of priority (10) : 1601
Expected bit time of priority (11) : 1747
Expected bit time of priority (12) : 1893
Expected bit time of priority (13) : 2038
Expected bit time of priority (14) : 3932
Expected message response time of priority (1): 0.0011640
Expected message response time of priority (2): 0.0017440
Expected message response time of priority (3): 0.0023280
Expected message response time of priority (4): 0.0029120
Expected message response time of priority (5): 0.0034920
Expected message response time of priority (6): 0.0040760
Expected message response time of priority (7): 0.0046600
Expected message response time of priority (8): 0.0052400
Expected message response time of priority (9): 0.0058240
Expected message response time of priority (10): 0.0064040
Expected message response time of priority (11): 0.0069880
Expected message response time of priority (12): 0.0075720

Expected message response time of priority (13): 0.0081520
Expected message response time of priority (14): 0.0157280

Expected message response time calculation for the twelfth input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):14
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.012
Input the sampling rate of the priority 2 message (in Sec.):0.0097
Input the sampling rate of the priority 3 message (in Sec.):0.0095
Input the sampling rate of the priority 4 message (in Sec.):0.0093
Input the sampling rate of the priority 5 message (in Sec.):0.0091
Input the sampling rate of the priority 6 message (in Sec.):0.009
Input the sampling rate of the priority 7 message (in Sec.):0.0085
Input the sampling rate of the priority 8 message (in Sec.):0.0085
Input the sampling rate of the priority 9 message (in Sec.):0.009
Input the sampling rate of the priority 10 message (in Sec.):0.0091
Input the sampling rate of the priority 11 message (in Sec.):0.0093
Input the sampling rate of the priority 12 message (in Sec.):0.0095
Input the sampling rate of the priority 13 message (in Sec.):0.0097
Input the sampling rate of the priority 14 message (in Sec.):0.012
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected bit time of priority (5) : 873
Expected bit time of priority (6) : 1019
Expected bit time of priority (7) : 1165
Expected bit time of priority (8) : 1310
Expected bit time of priority (9) : 1456
Expected bit time of priority (10) : 1601
Expected bit time of priority (11) : 1747
Expected bit time of priority (12) : 1893
Expected bit time of priority (13) : 2038
Expected bit time of priority (14) : 4077
Expected message response time of priority (1): 0.0011640
Expected message response time of priority (2): 0.0017440
Expected message response time of priority (3): 0.0023280
Expected message response time of priority (4): 0.0029120
Expected message response time of priority (5): 0.0034920
Expected message response time of priority (6): 0.0040760
Expected message response time of priority (7): 0.0046600
Expected message response time of priority (8): 0.0052400
Expected message response time of priority (9): 0.0058240
Expected message response time of priority (10): 0.0064040
Expected message response time of priority (11): 0.0069880
Expected message response time of priority (12): 0.0075720

Expected message response time of priority (13): 0.0081520
Expected message response time of priority (14): 0.0163080

Expected message response time calculation for the thirteenth input data file :

Input the network speed in kbps :250
Input the number of messages transmitted via CAN bus (<=100):14
Input the probability of error occurring per message transmission :0.001
Input the probability of sporadic signal occurring :0.001
Input the sampling rate of the priority 1 message (in Sec.):0.01
Input the sampling rate of the priority 2 message (in Sec.):0.0093
Input the sampling rate of the priority 3 message (in Sec.):0.0091
Input the sampling rate of the priority 4 message (in Sec.):0.009
Input the sampling rate of the priority 5 message (in Sec.):0.009
Input the sampling rate of the priority 6 message (in Sec.):0.0087
Input the sampling rate of the priority 7 message (in Sec.):0.0085
Input the sampling rate of the priority 8 message (in Sec.):0.0085
Input the sampling rate of the priority 9 message (in Sec.):0.0087
Input the sampling rate of the priority 10 message (in Sec.):0.009
Input the sampling rate of the priority 11 message (in Sec.):0.009
Input the sampling rate of the priority 12 message (in Sec.):0.0091
Input the sampling rate of the priority 13 message (in Sec.):0.0093
Input the sampling rate of the priority 14 message (in Sec.):0.01
Expected bit time of priority (2) : 436
Expected bit time of priority (3) : 582
Expected bit time of priority (4) : 728
Expected bit time of priority (5) : 873
Expected bit time of priority (6) : 1019
Expected bit time of priority (7) : 1165
Expected bit time of priority (8) : 1310
Expected bit time of priority (9) : 1456
Expected bit time of priority (10) : 1601
Expected bit time of priority (11) : 1747
Expected bit time of priority (12) : 1893
Expected bit time of priority (13) : 2038
Expected bit time of priority (14) : 4077
Expected message response time of priority (1): 0.0011640
Expected message response time of priority (2): 0.0017440
Expected message response time of priority (3): 0.0023280
Expected message response time of priority (4): 0.0029120
Expected message response time of priority (5): 0.0034920
Expected message response time of priority (6): 0.0040760
Expected message response time of priority (7): 0.0046600
Expected message response time of priority (8): 0.0052400
Expected message response time of priority (9): 0.0058240
Expected message response time of priority (10): 0.0064040
Expected message response time of priority (11): 0.0069880
Expected message response time of priority (12): 0.0075720

Expected message response time of priority (13): 0.0081520
Expected message response time of priority (14): 0.0163080

APPENDIX C

SIMULATION RESULTS OF EXPERIMENTS 1-8

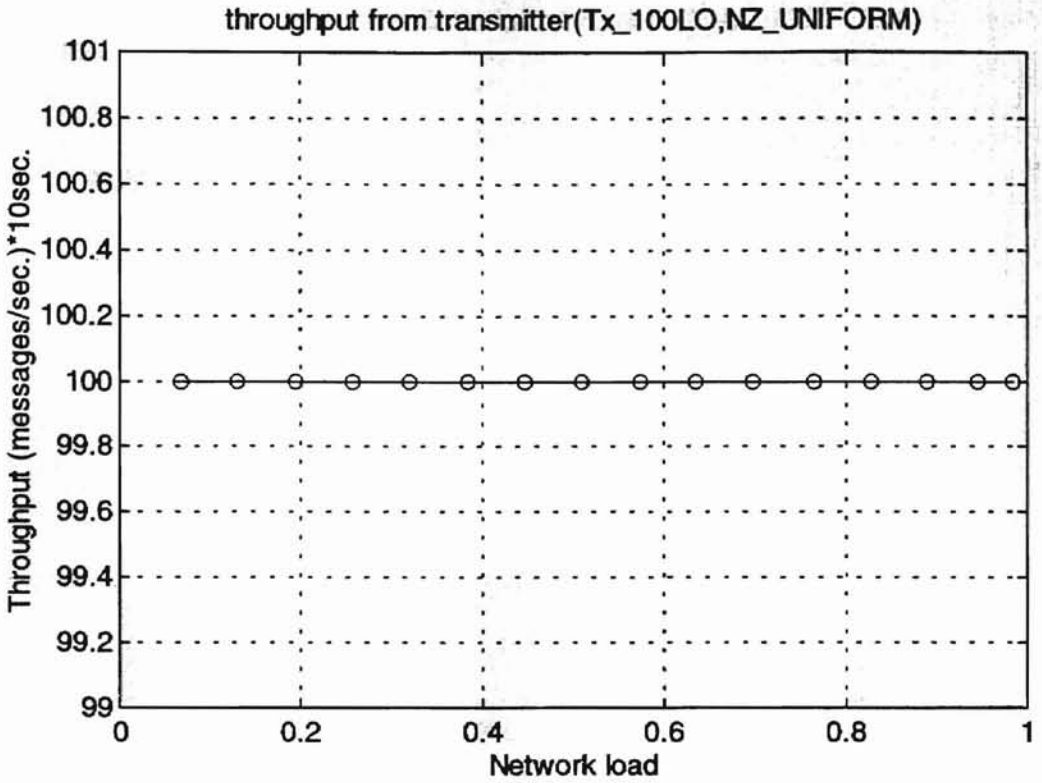


Figure C.1 Simulation result of experiment 1

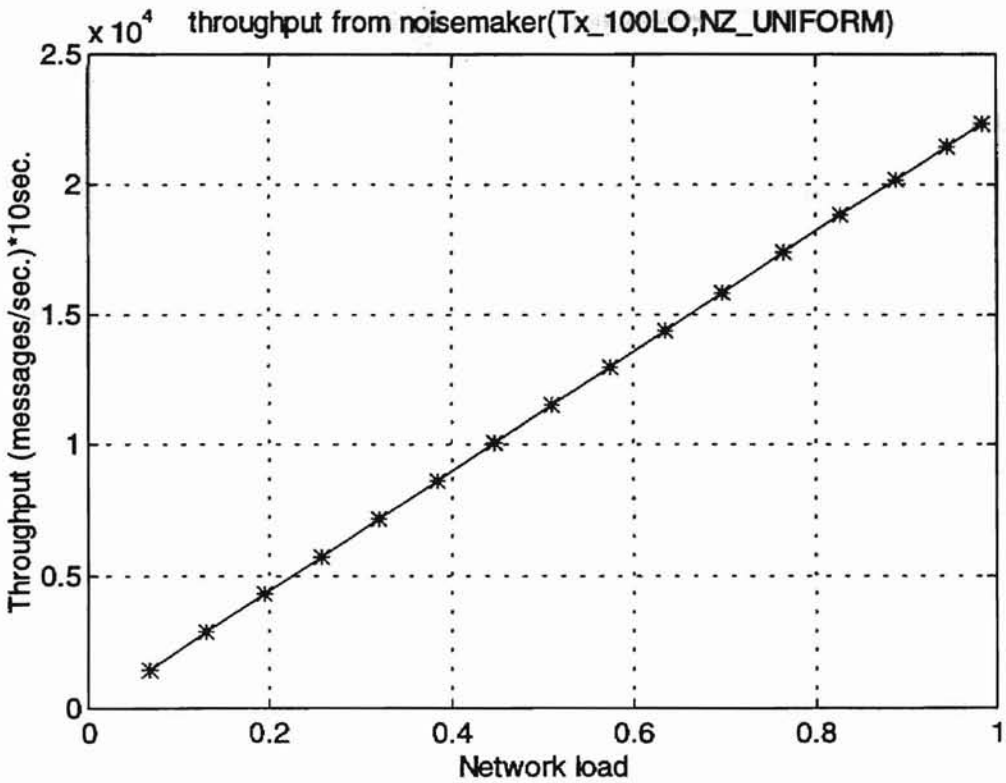


Figure C.2 Simulation result of experiment 1

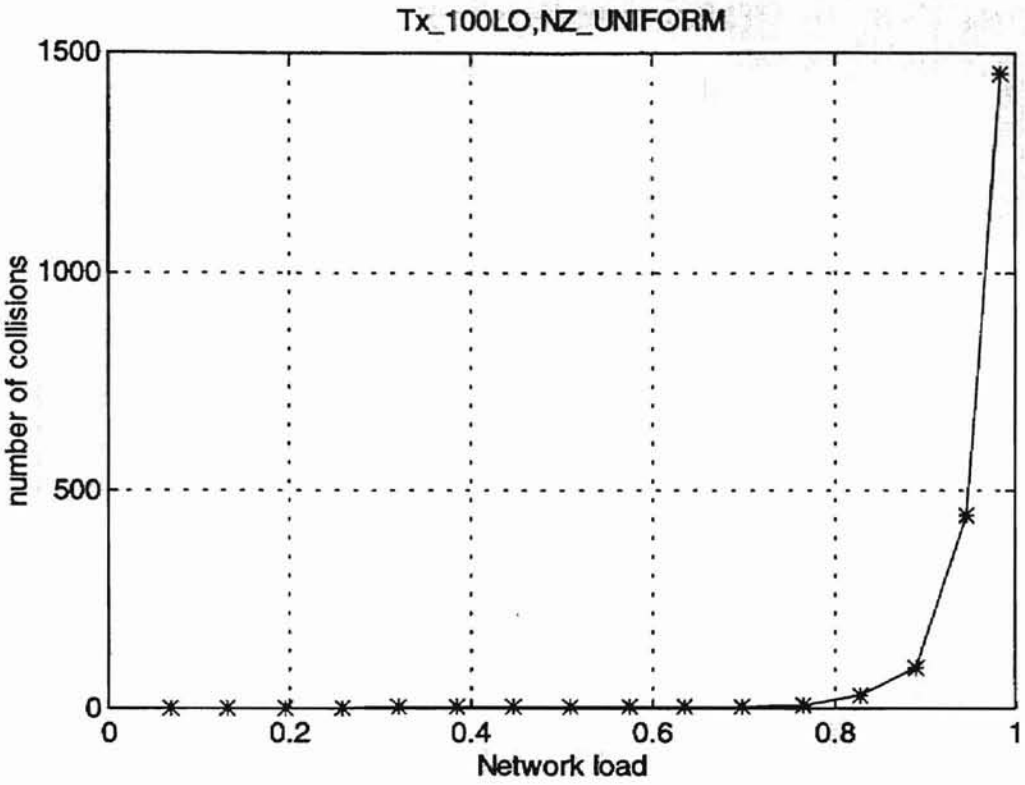


Figure C.3 Simulation result of experiment 1

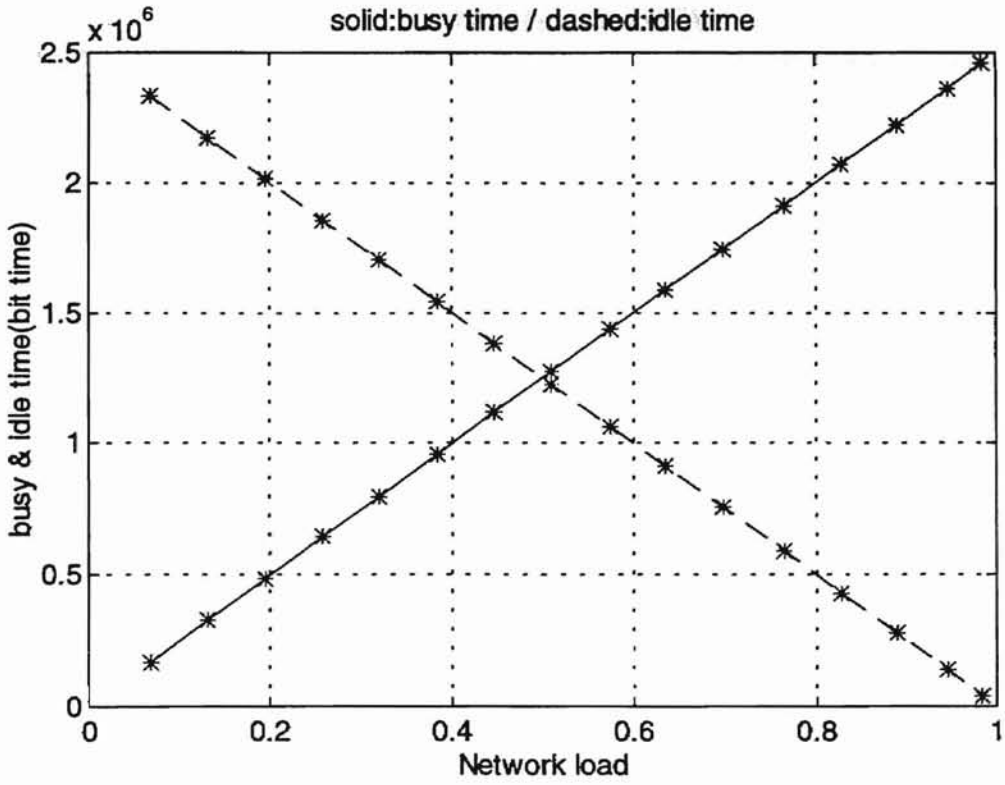


Figure C.4 Simulation result of experiment 1

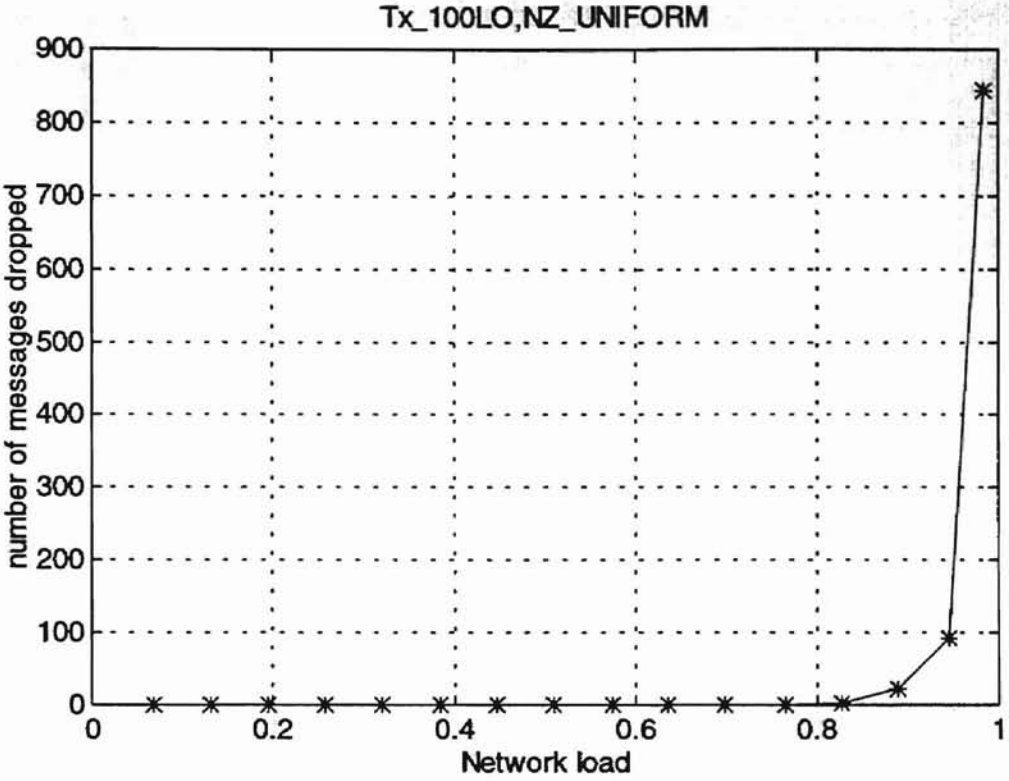


Figure C.5 Simulation result of experiment 1

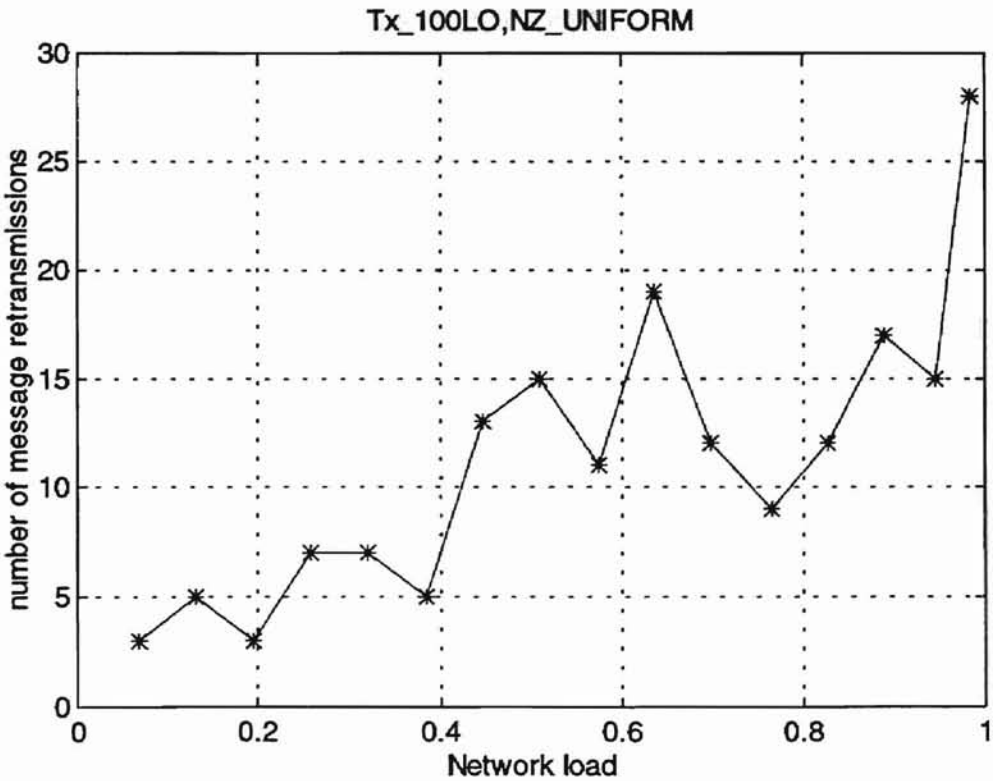


Figure C.6 Simulation result of experiment 1

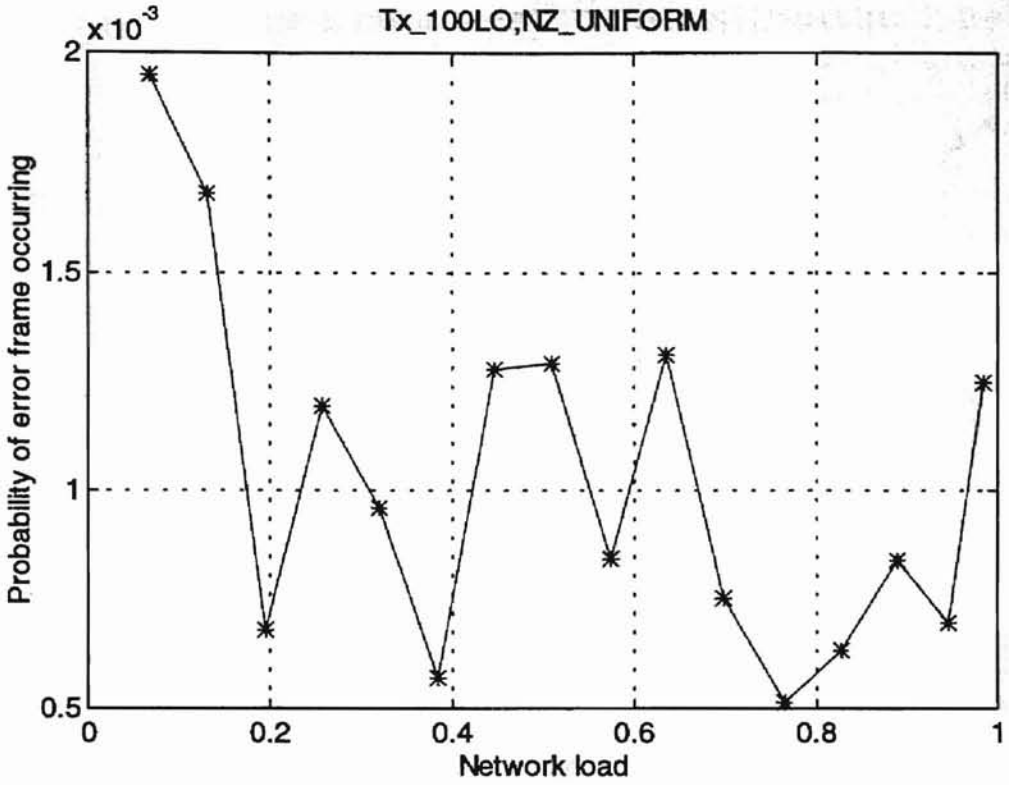


Figure C.7 Simulation result of experiment 1

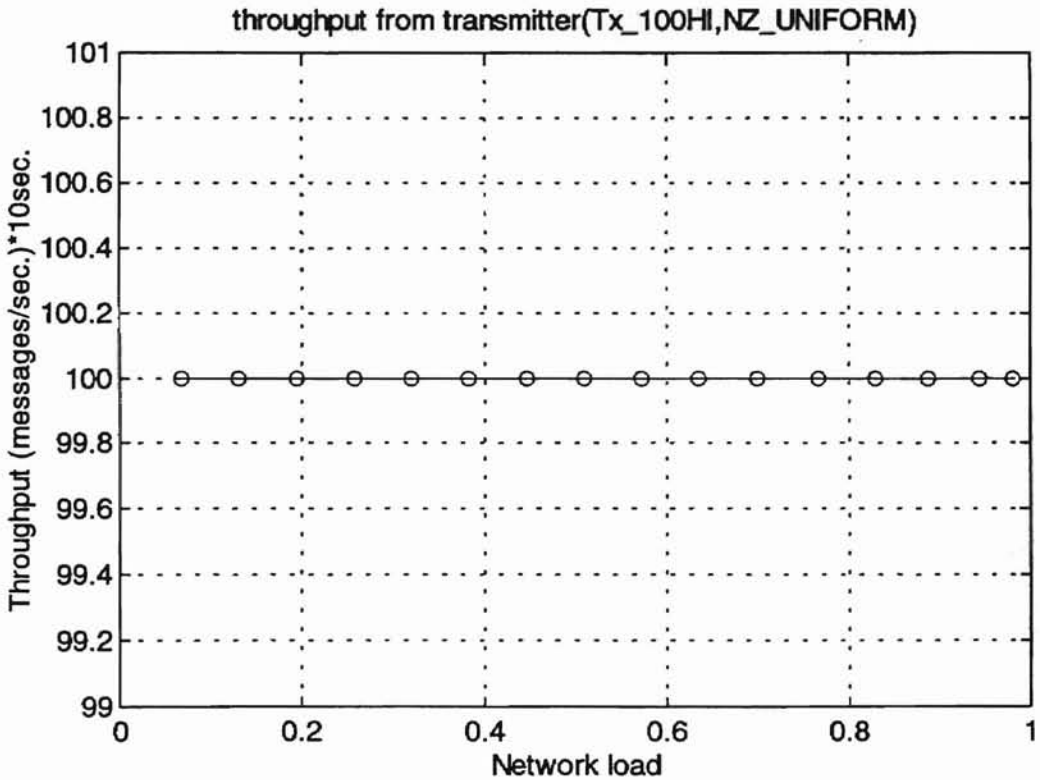


Figure C.8 Simulation result of experiment 2

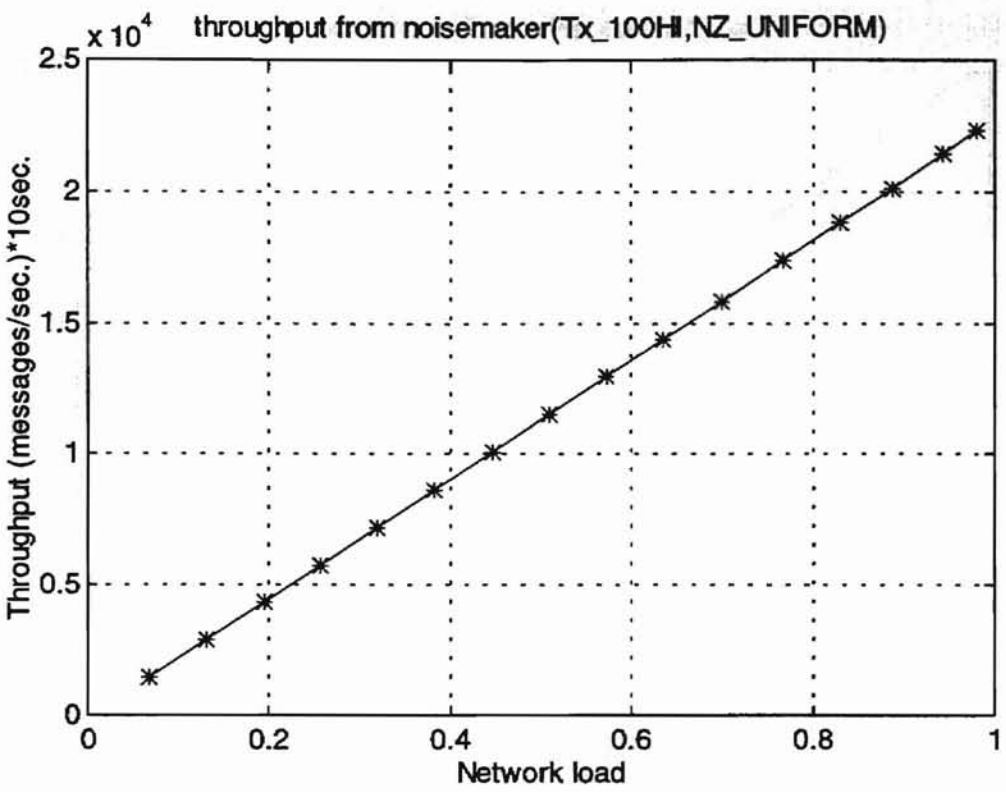


Figure C.9 Simulation result of experiment 2

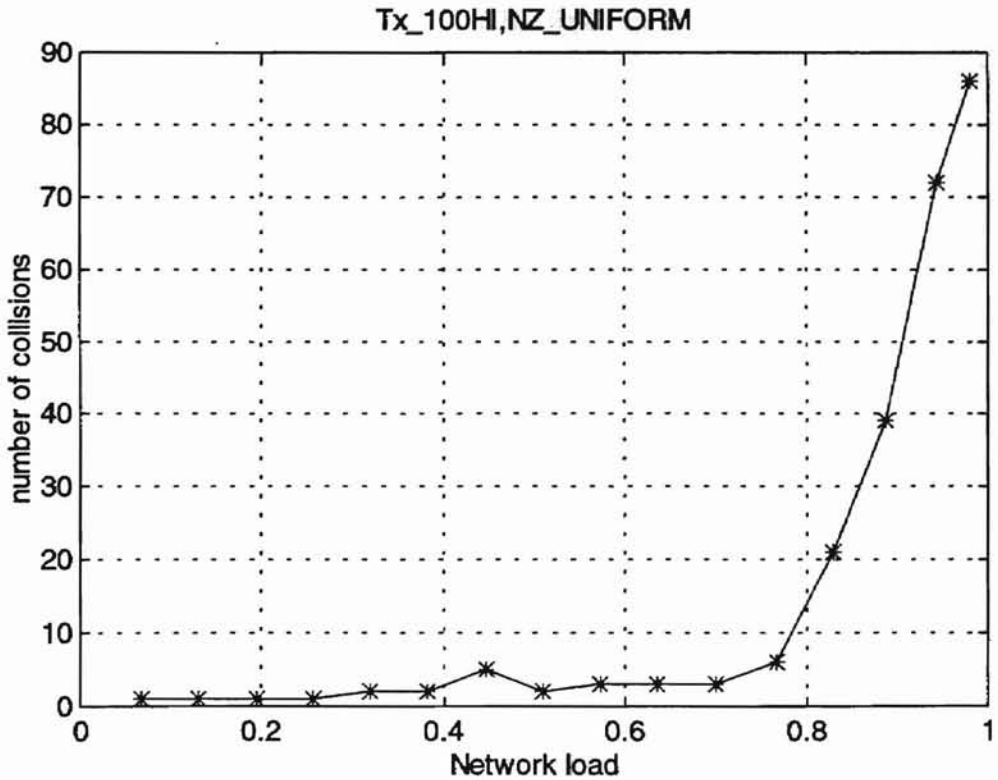


Figure C.10 Simulation result of experiment 2

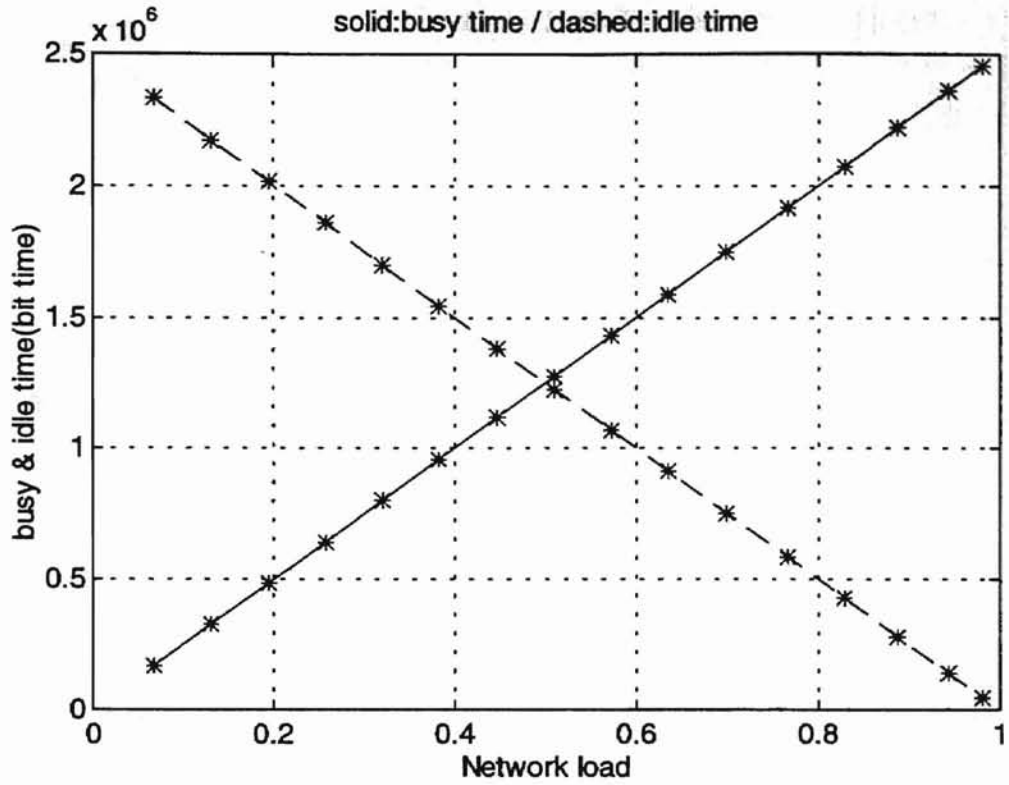


Figure C.11 Simulation result of experiment 2

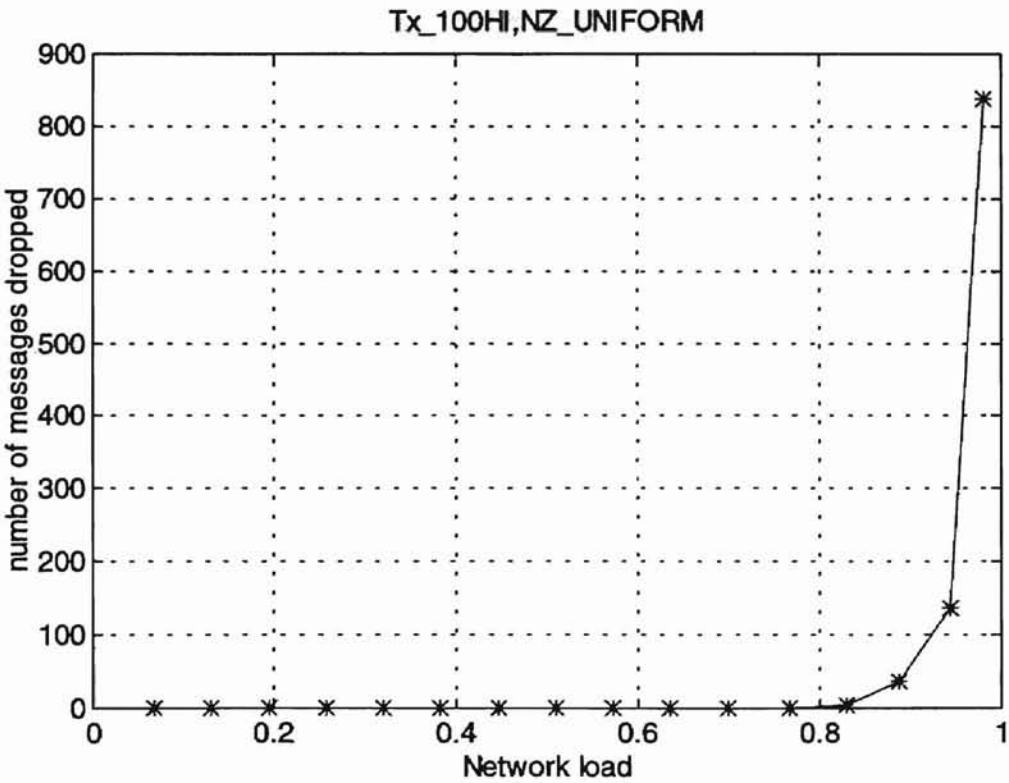


Figure C.12 Simulation result of experiment 2

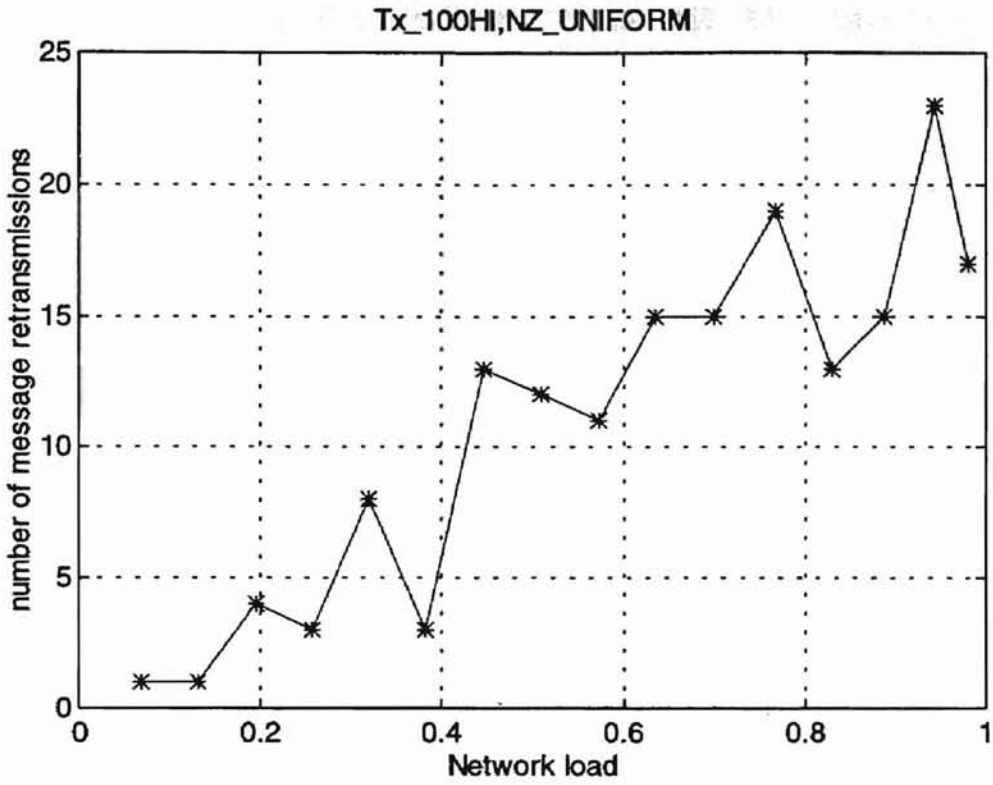


Figure C.13 Simulation result of experiment 2

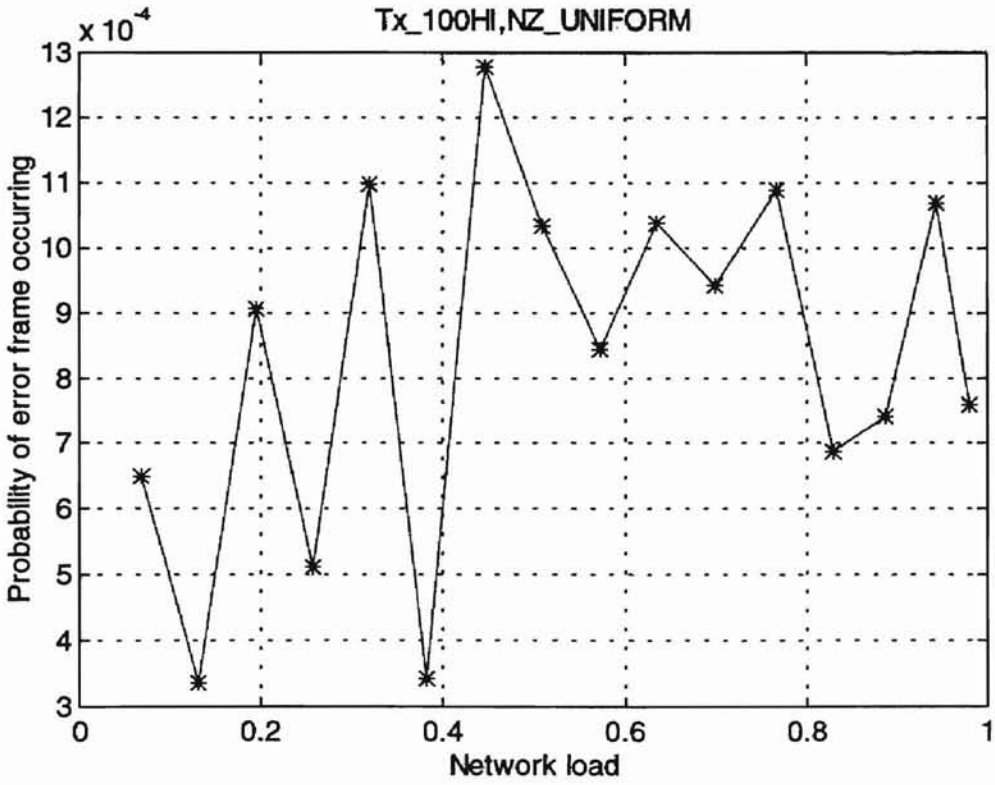


Figure C.14 Simulation result of experiment 2

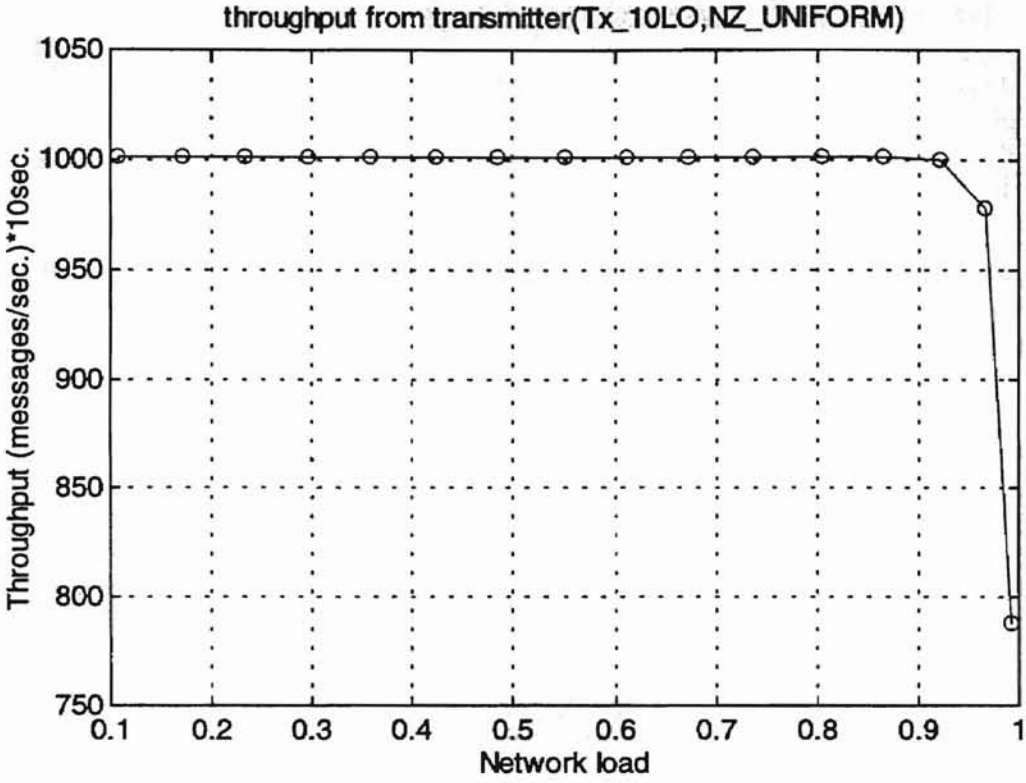


Figure C.15 Simulation result of experiment 3

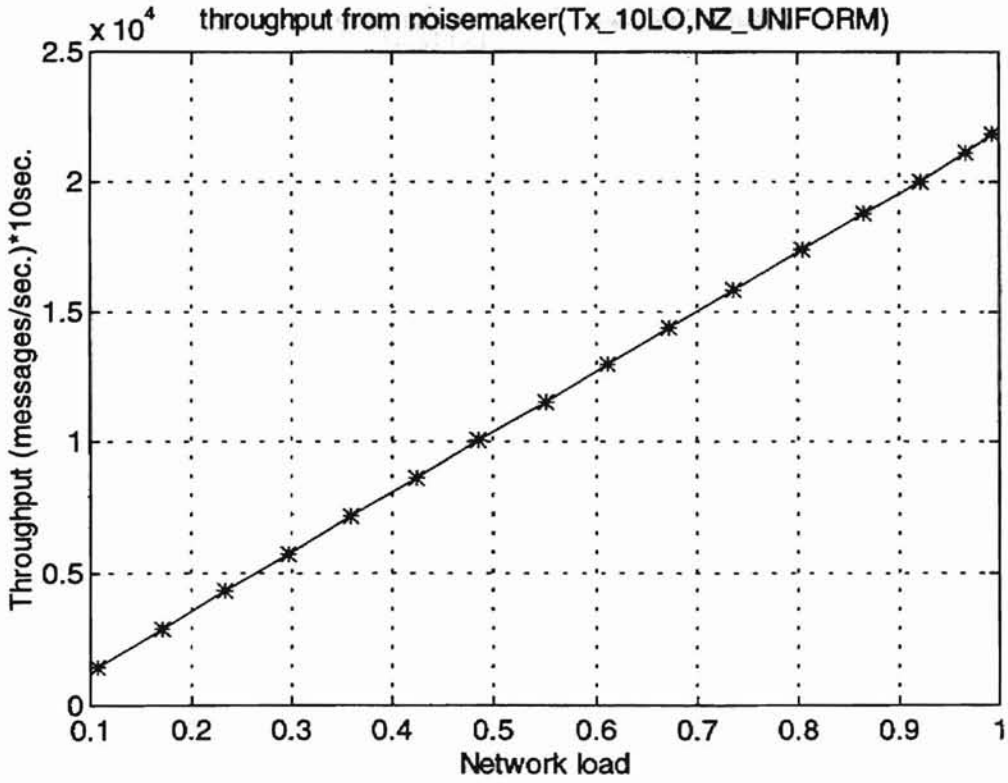


Figure C.16 Simulation result of experiment 3

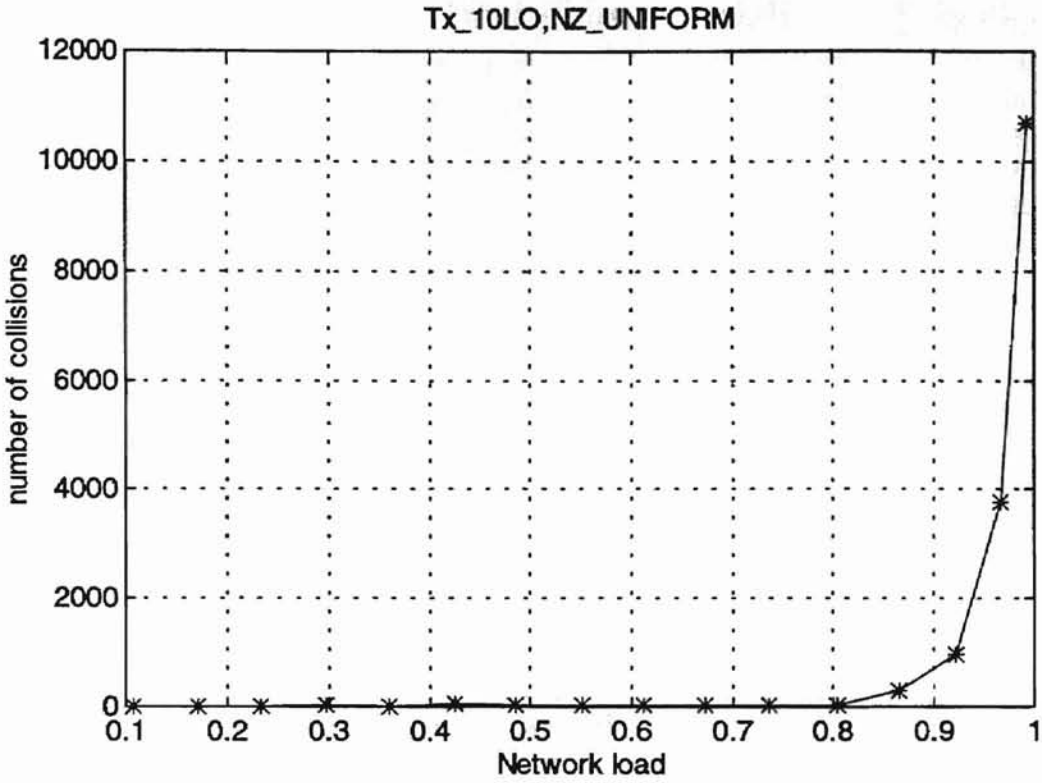


Figure C.17 Simulation result of experiment 3

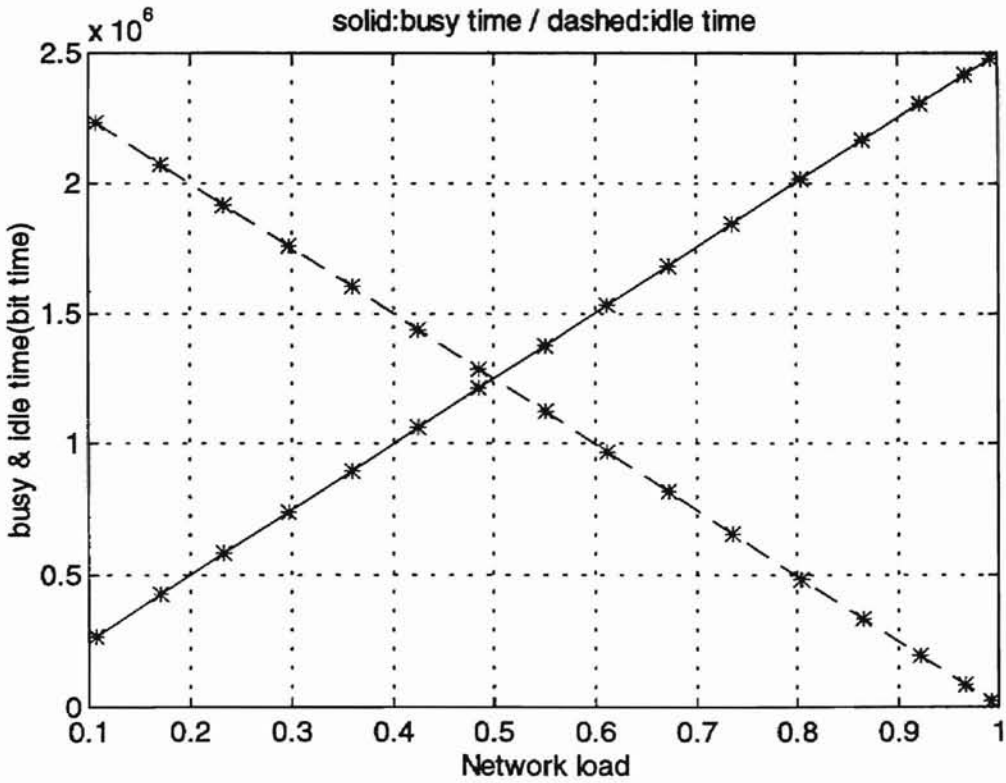


Figure C.18 Simulation result of experiment 3

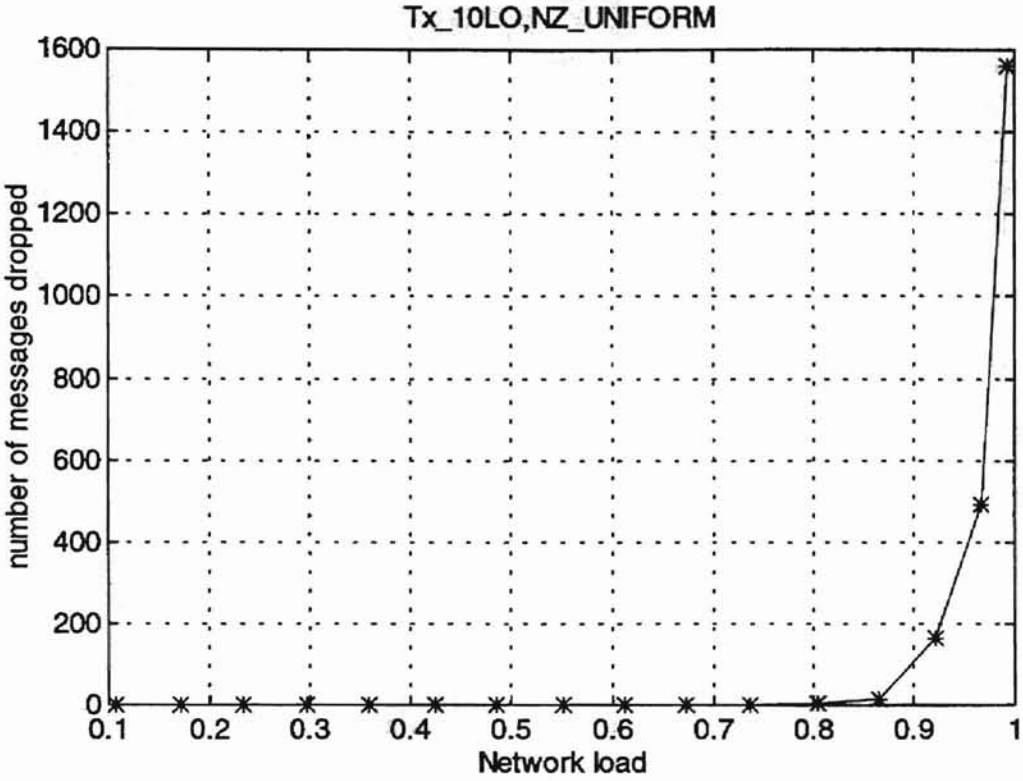


Figure C.19 Simulation result of experiment 3

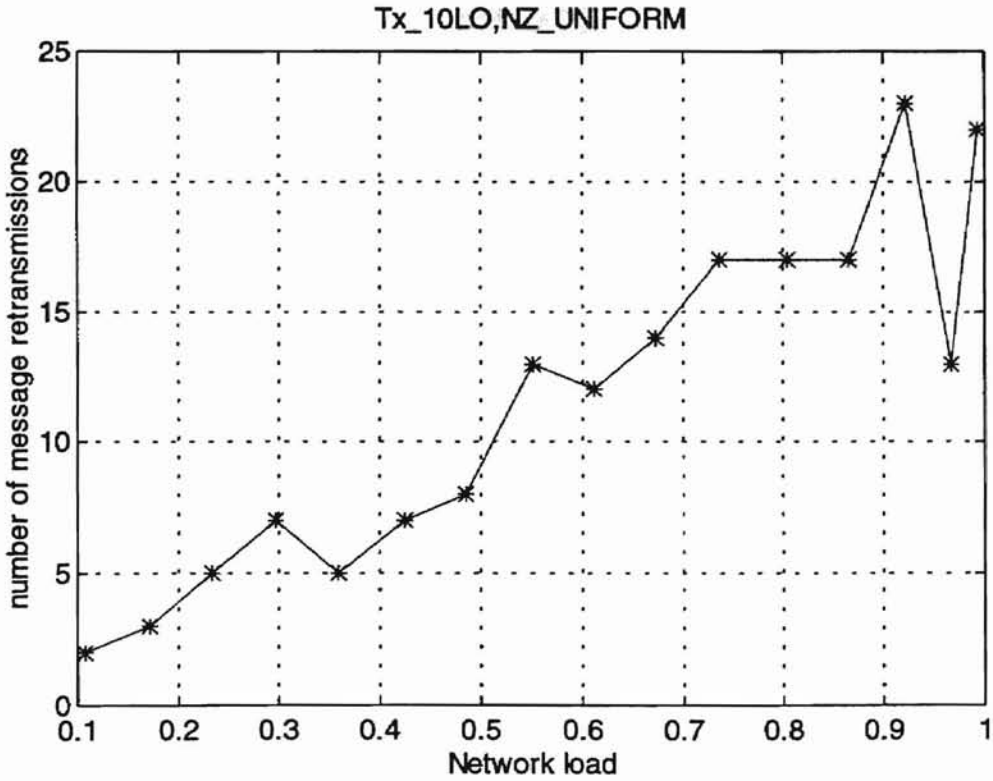


Figure C.20 Simulation result of experiment 3

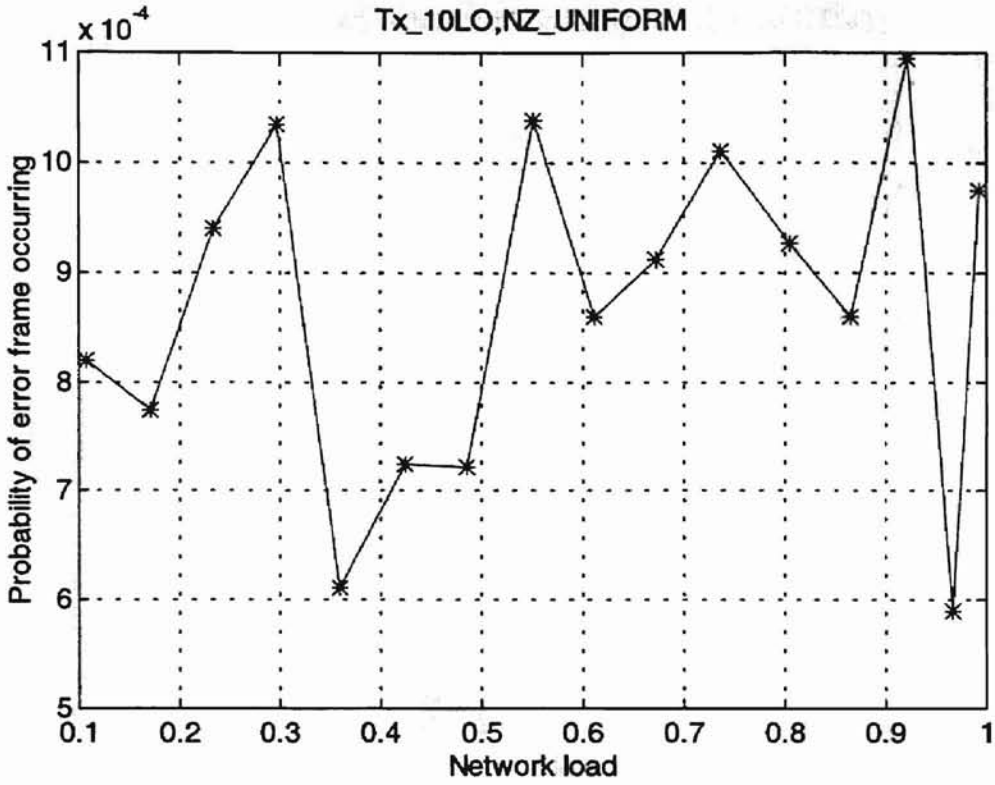


Figure C.21 Simulation result of experiment 3

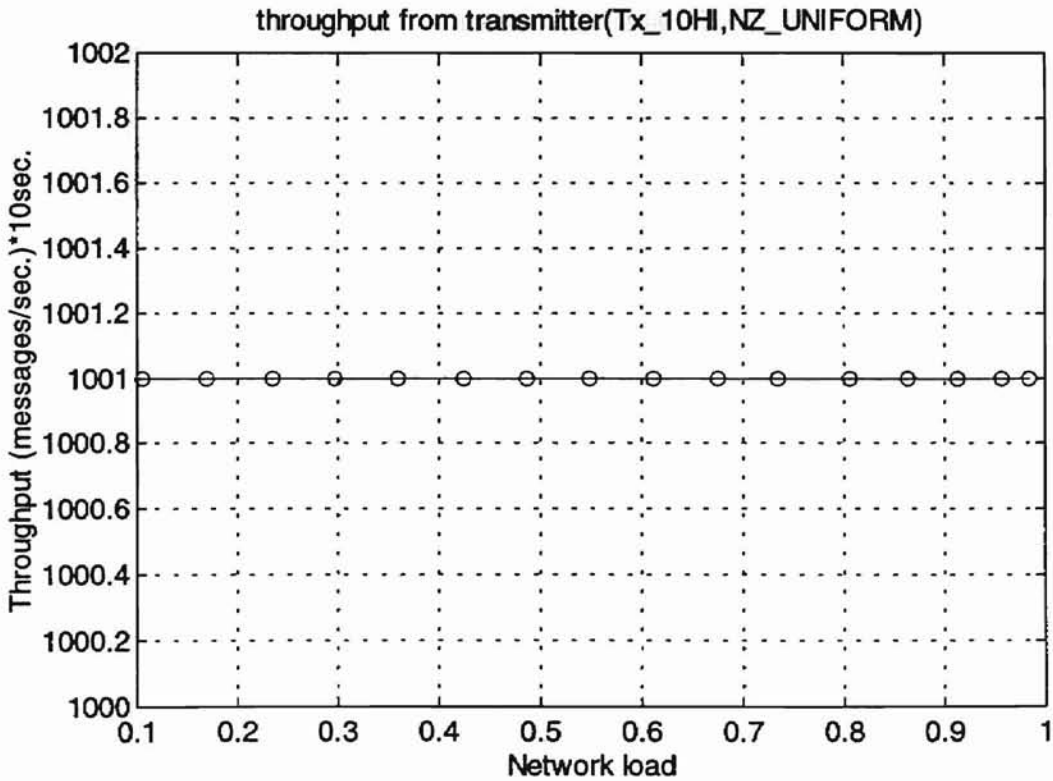


Figure C.22 Simulation result of experiment 4

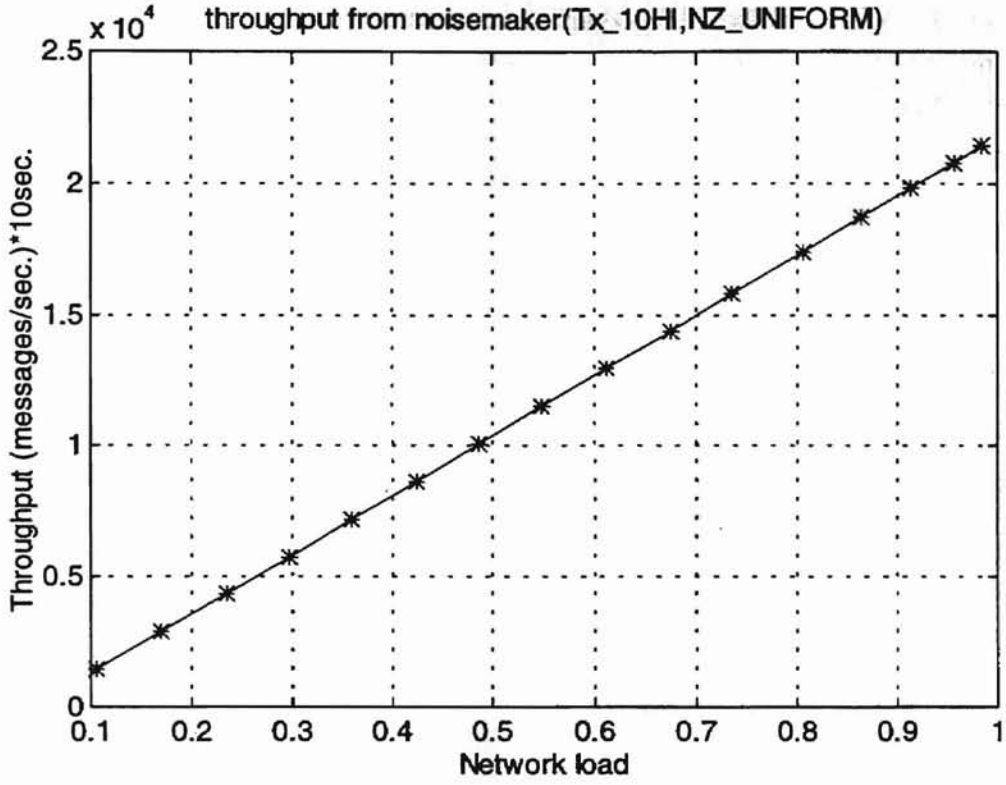


Figure C.23 Simulation result of experiment 4

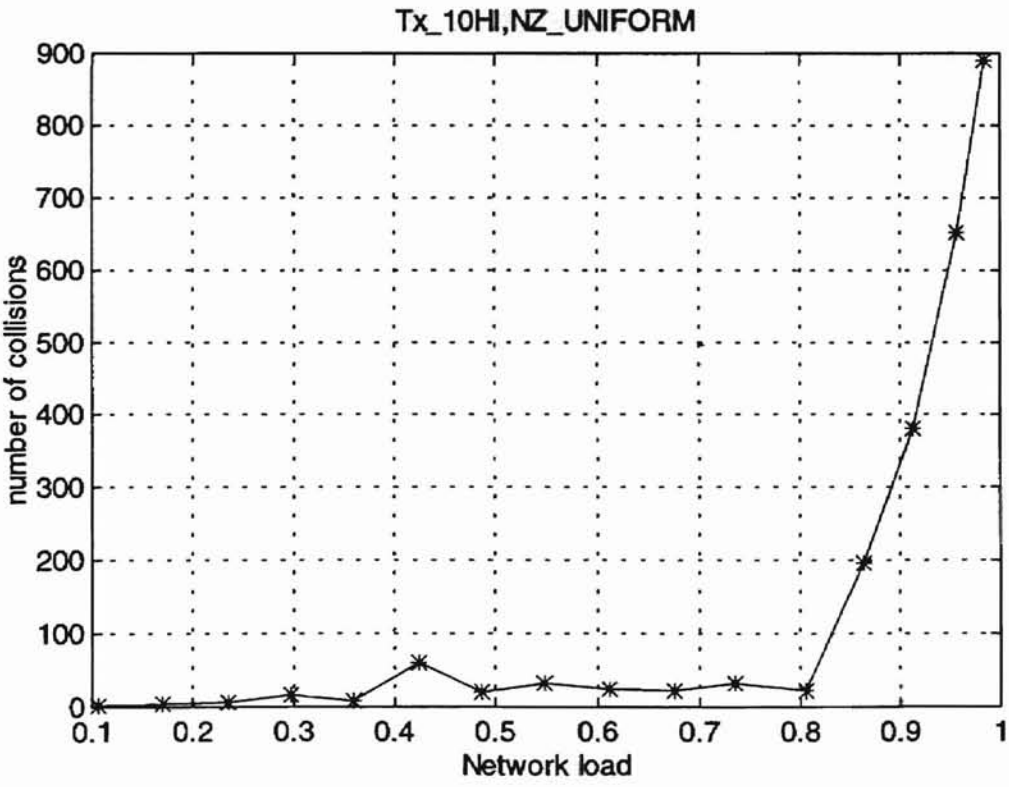


Figure C.24 Simulation result of experiment 4

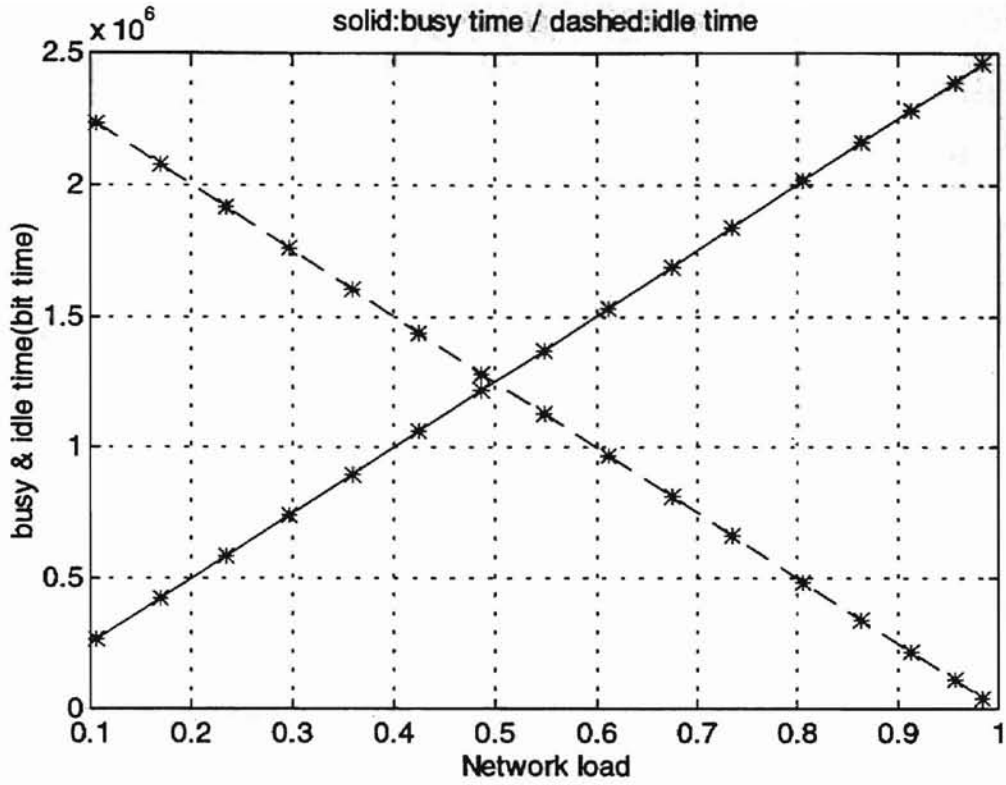


Figure C.25 Simulation result of experiment 4

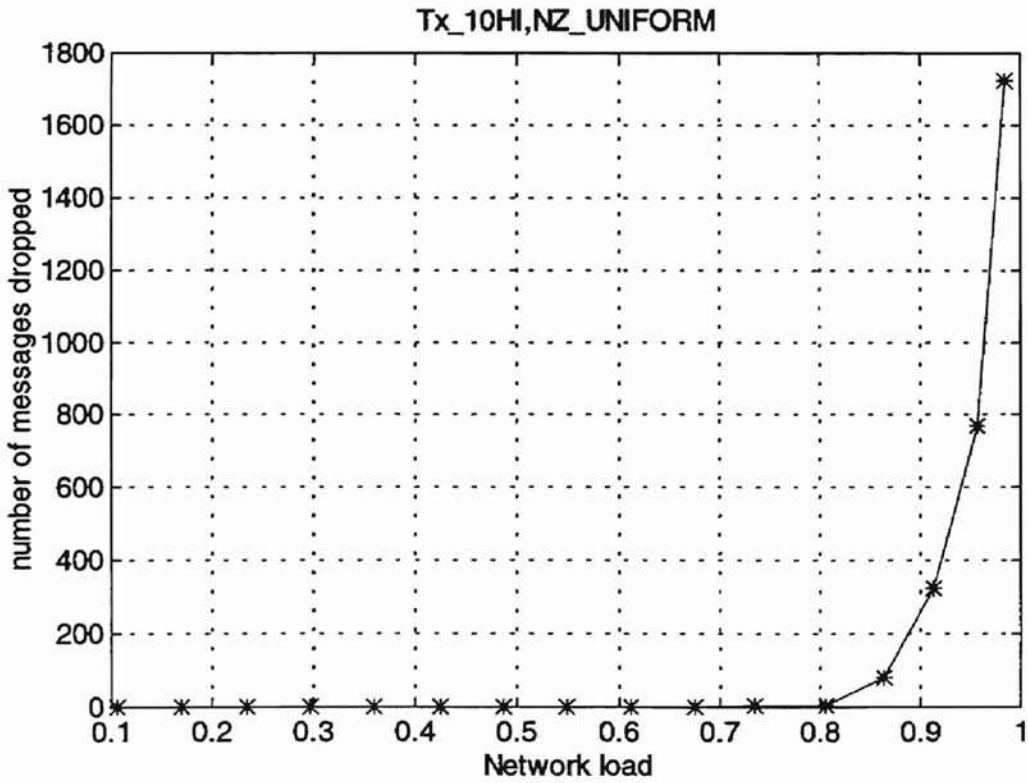


Figure C.26 Simulation result of experiment 4

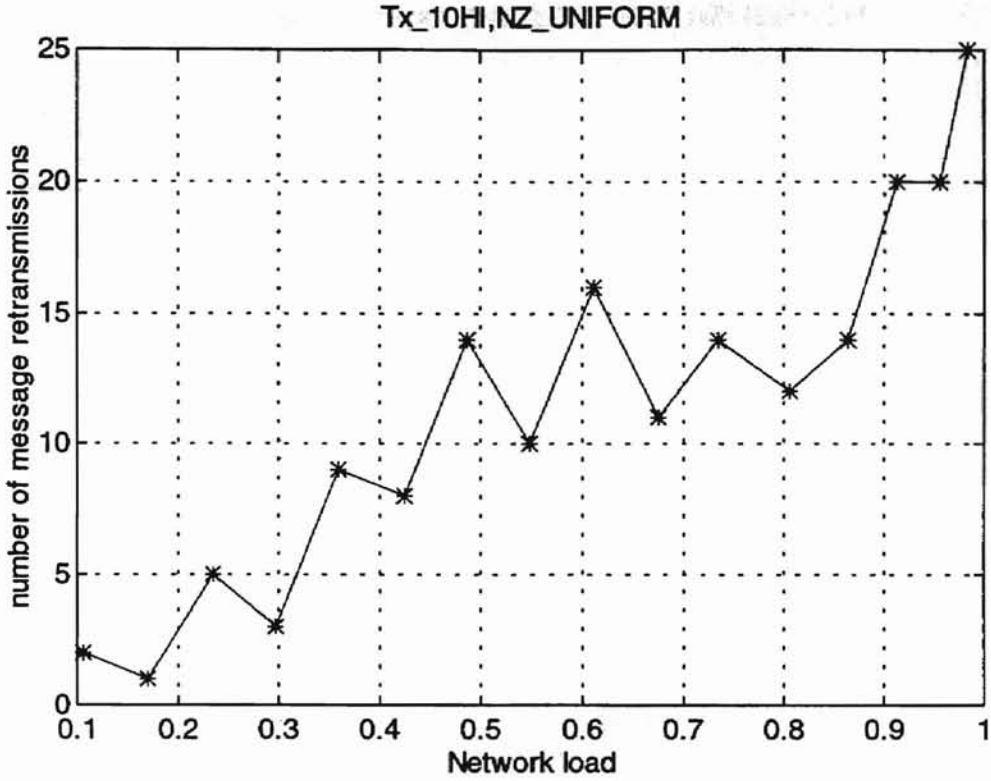


Figure C.27 Simulation result of experiment 4

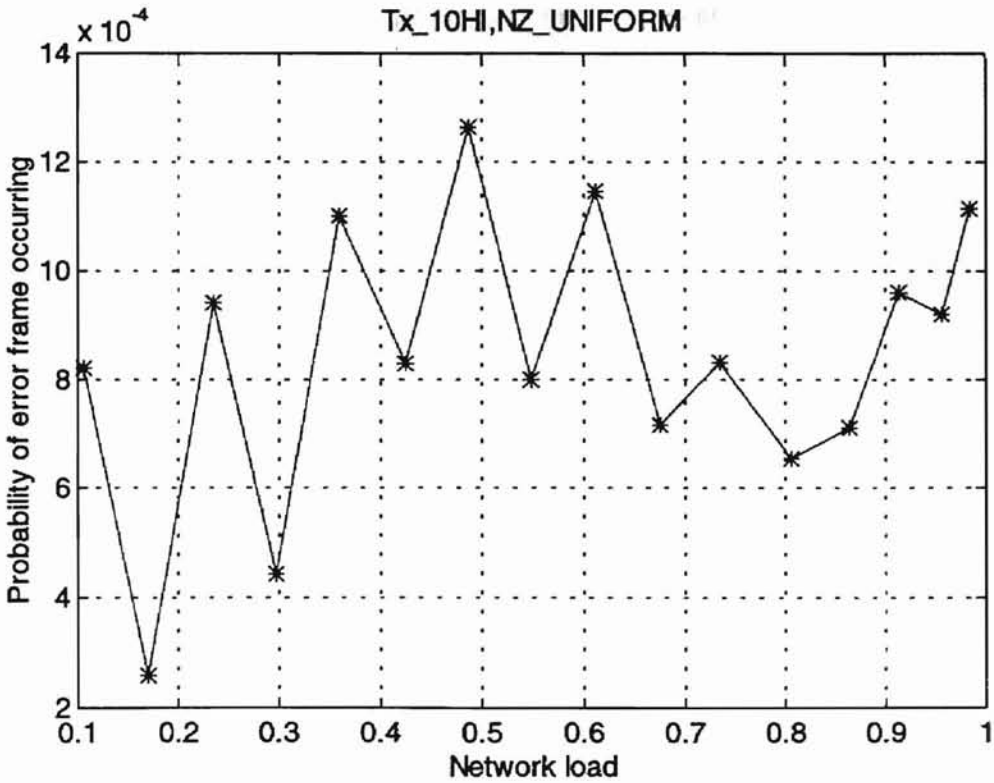


Figure C.28 Simulation result of experiment 4

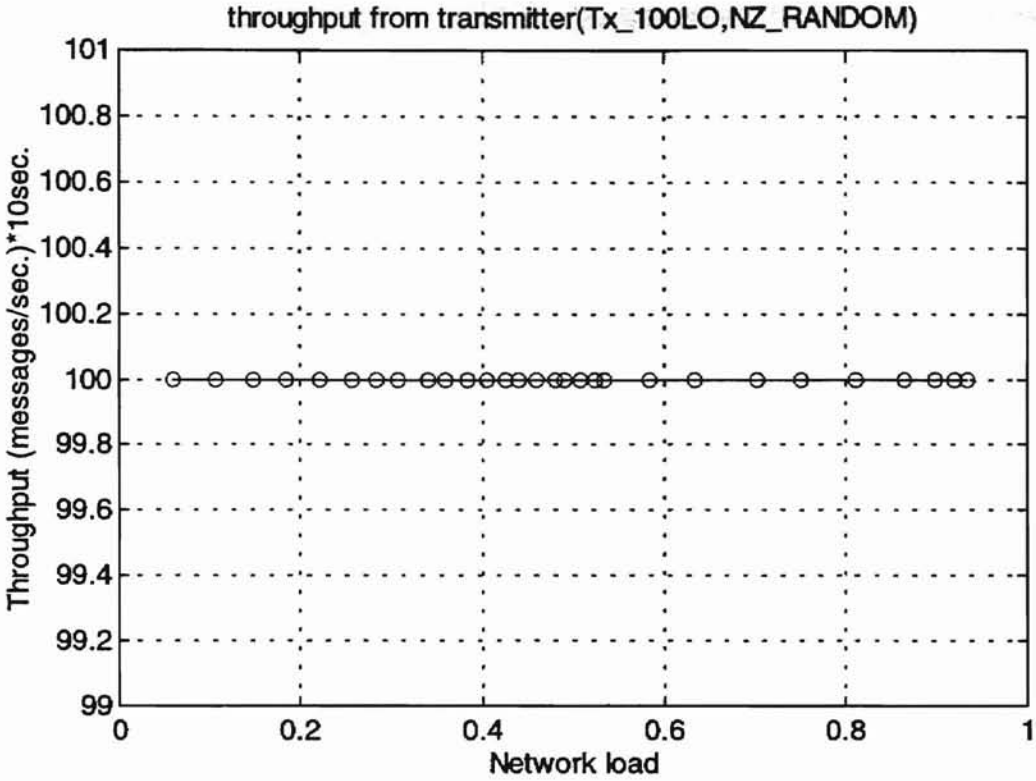


Figure C.29 Simulation result of experiment 5

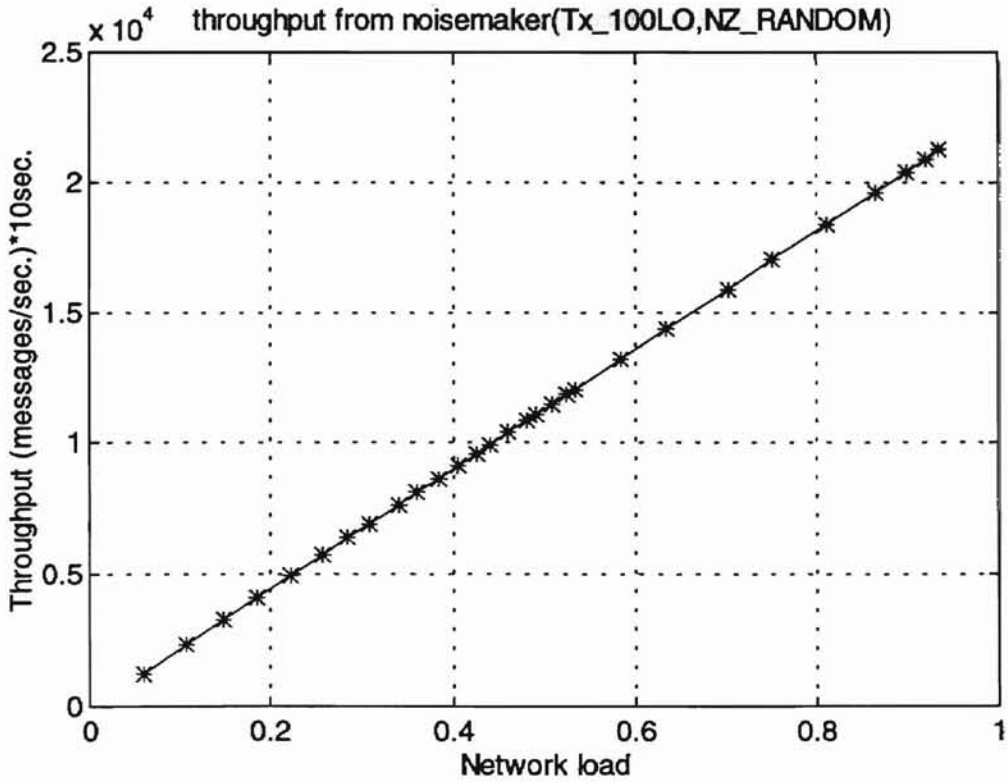


Figure C.30 Simulation result of experiment 5

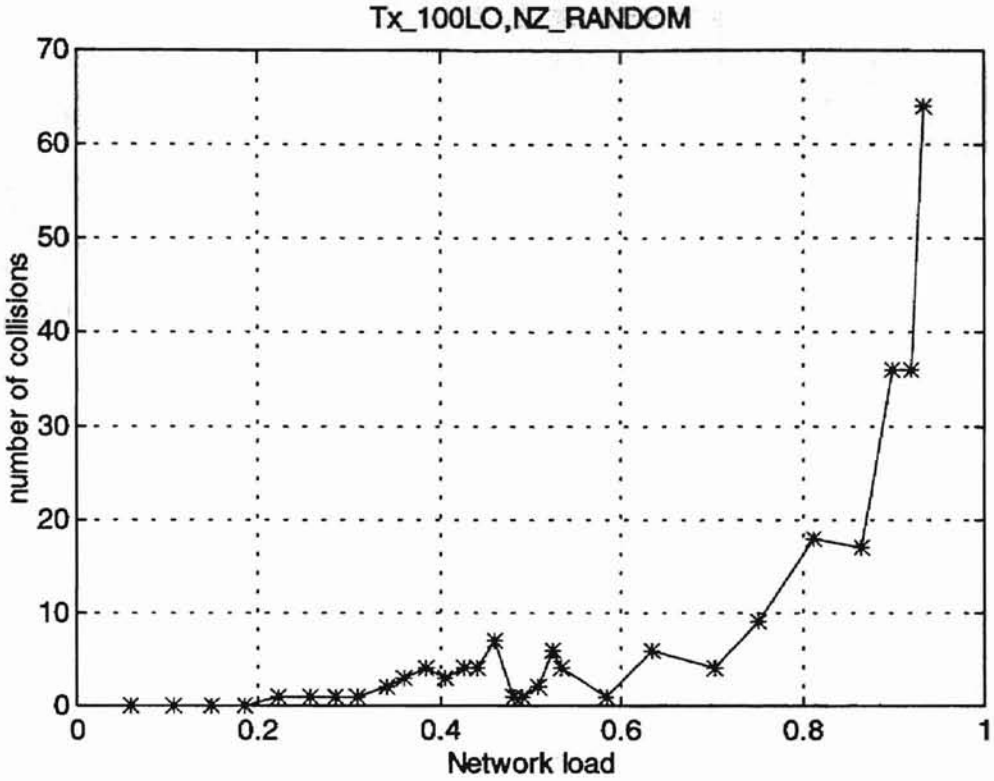


Figure C.31 Simulation result of experiment 5

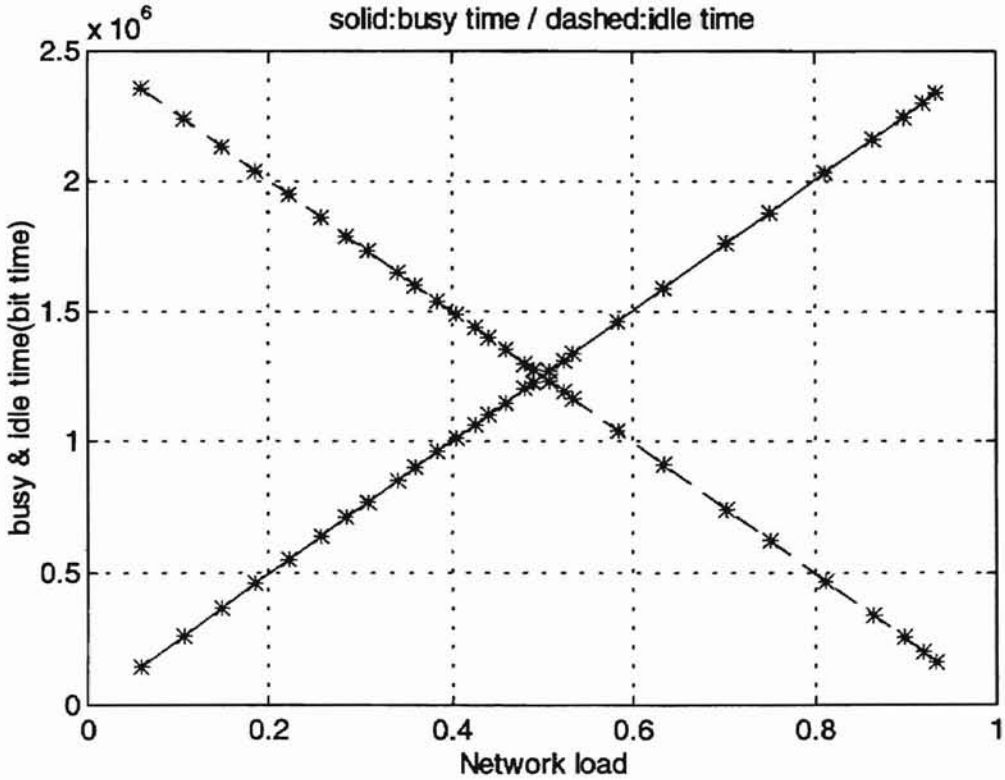


Figure C.32 Simulation result of experiment 5

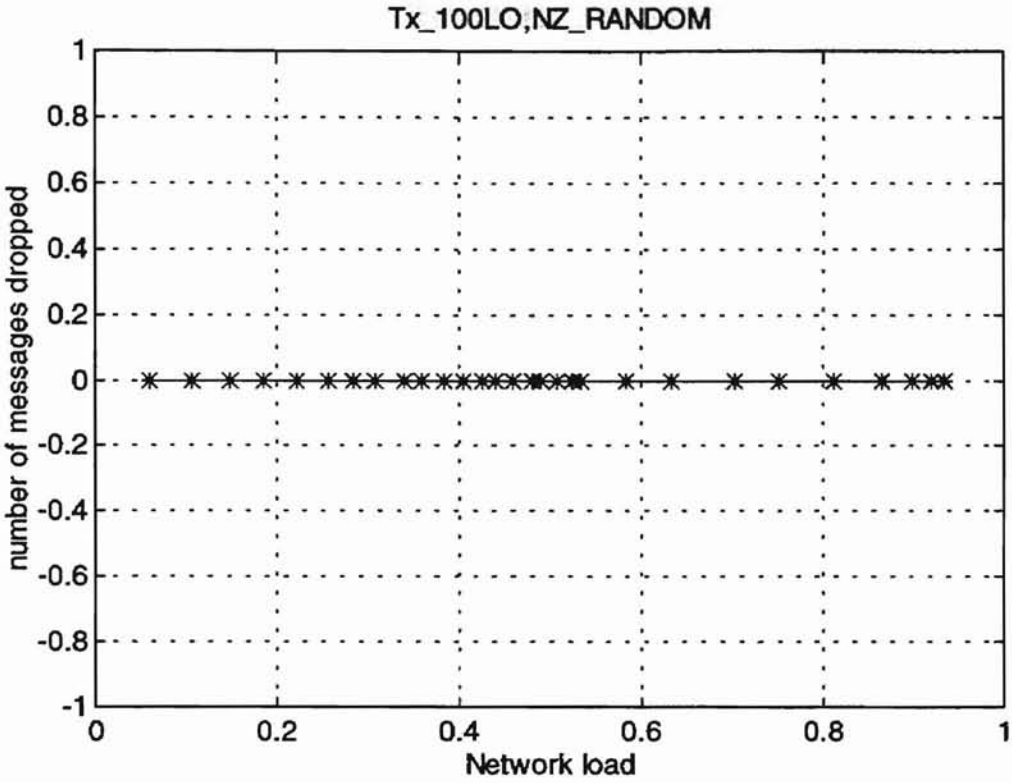


Figure C.33 Simulation result of experiment 5

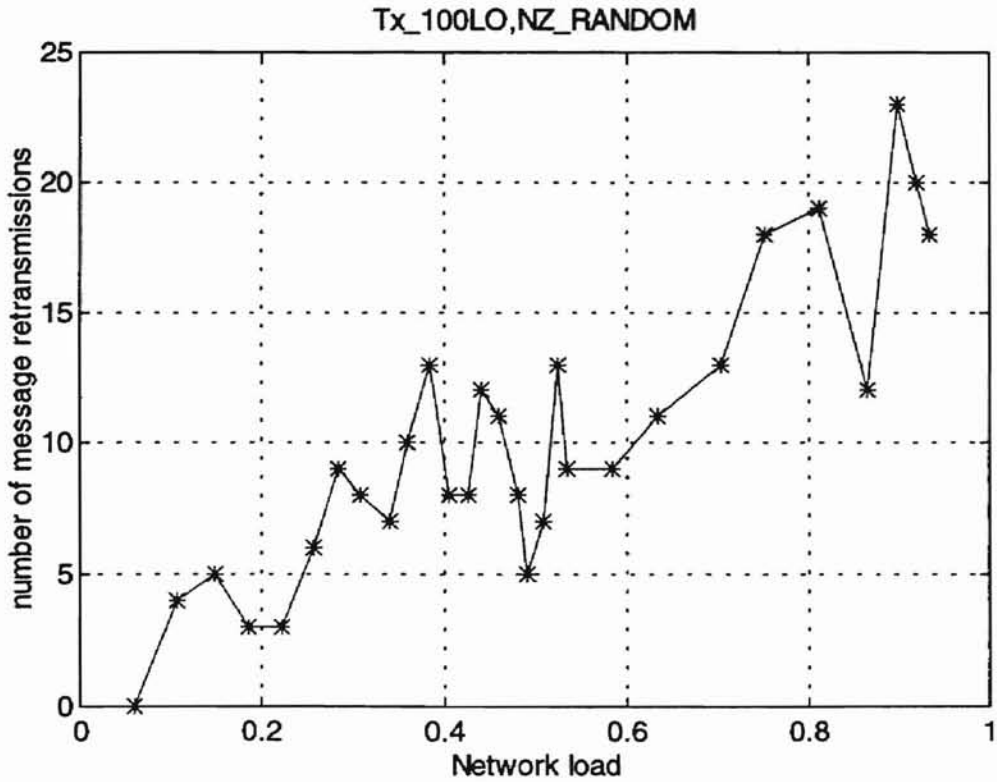


Figure C.34 Simulation result of experiment 5

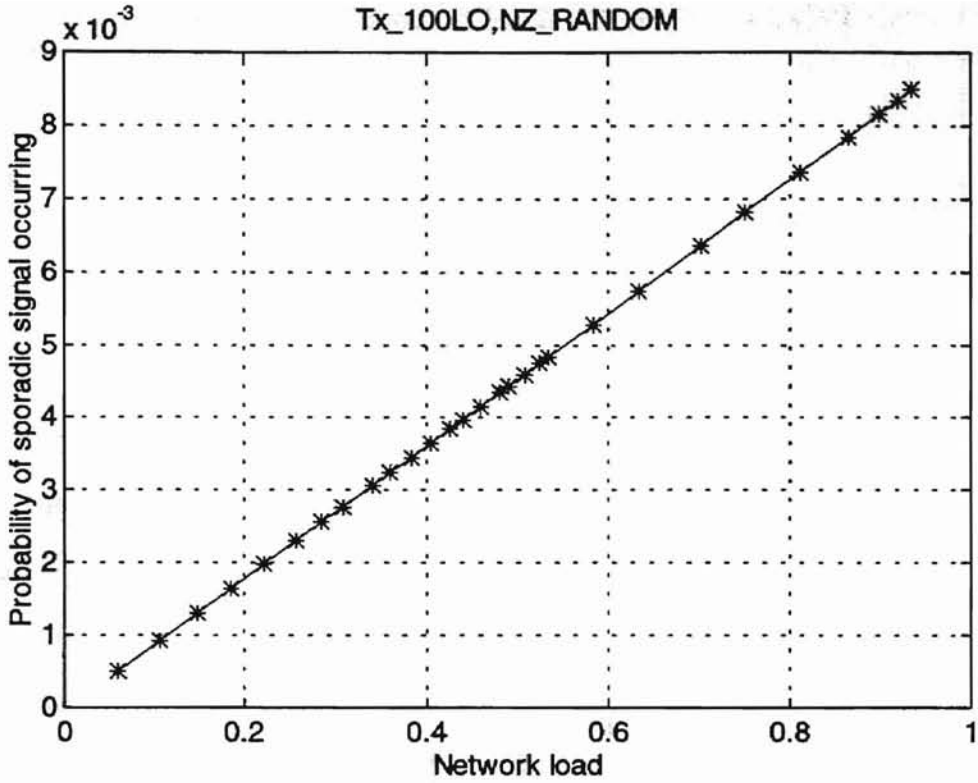


Figure C.35 Simulation result of experiment 5

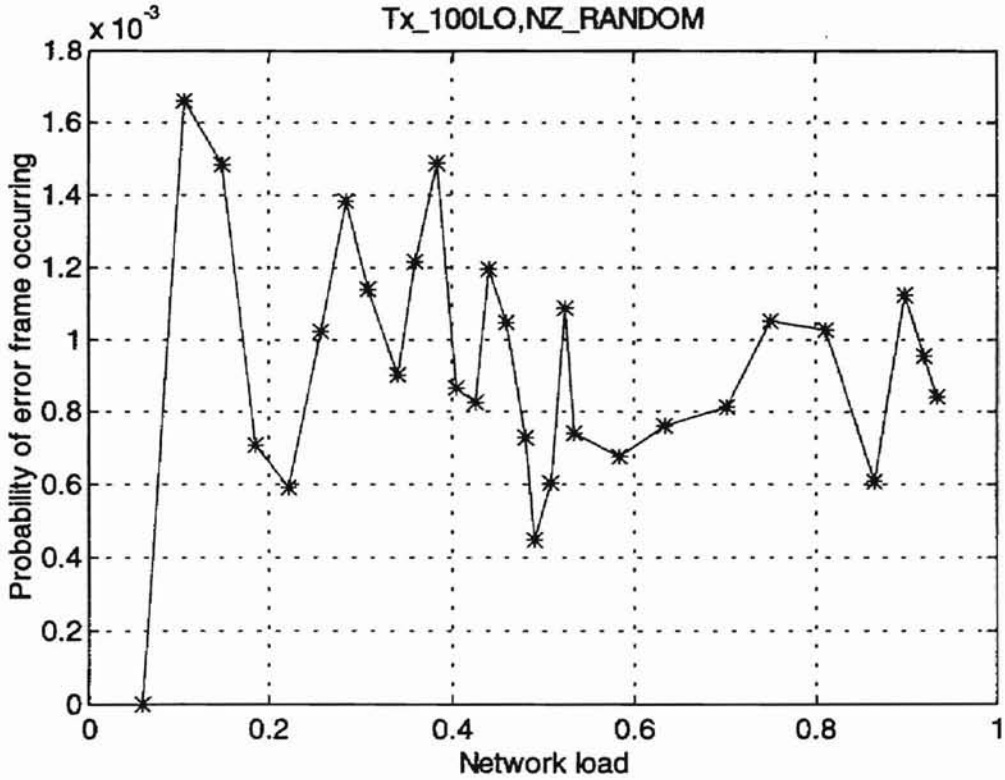


Figure C.36 Simulation result of experiment 5

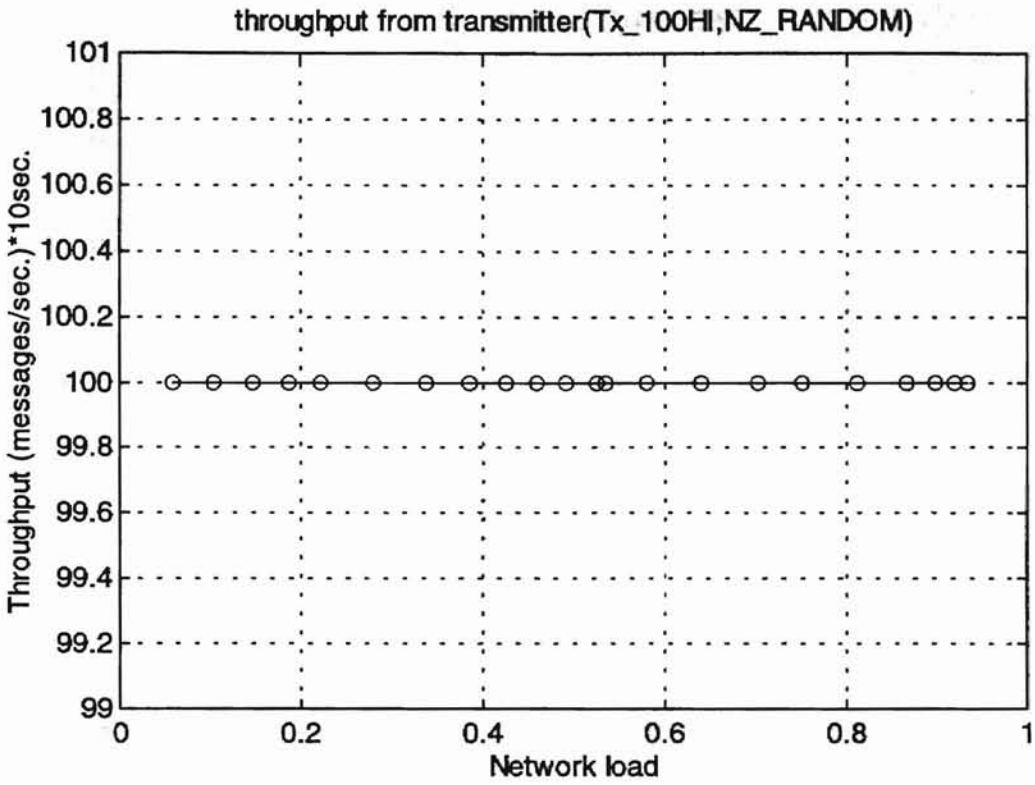


Figure C.37 Simulation result of experiment 6

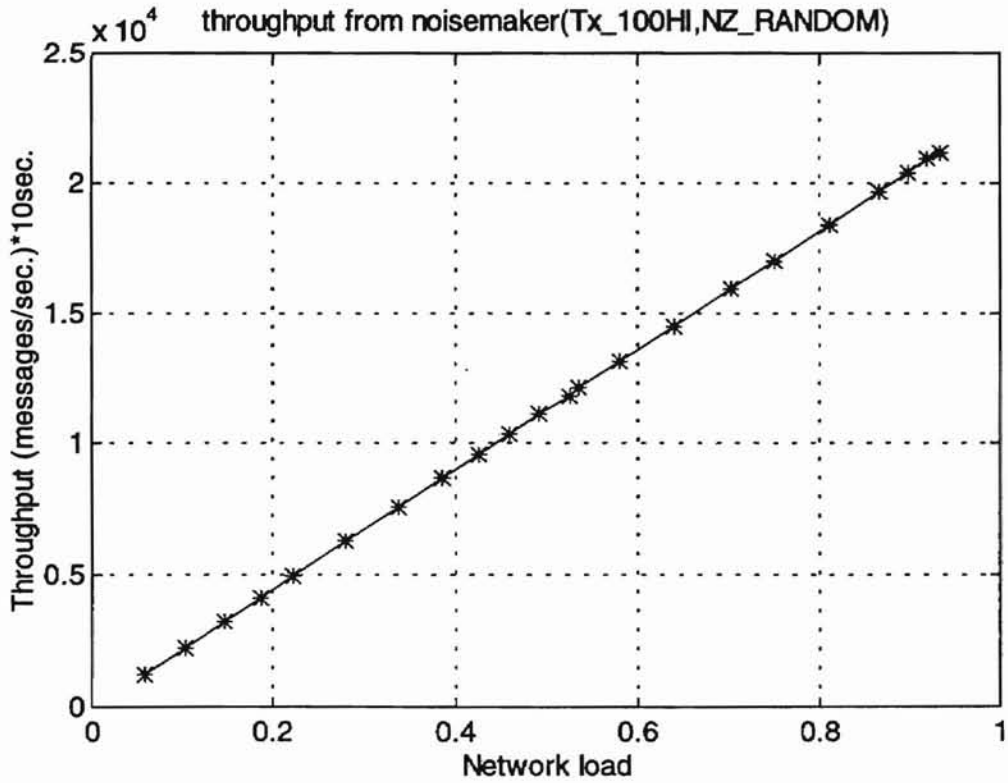


Figure C.38 Simulation result of experiment 6

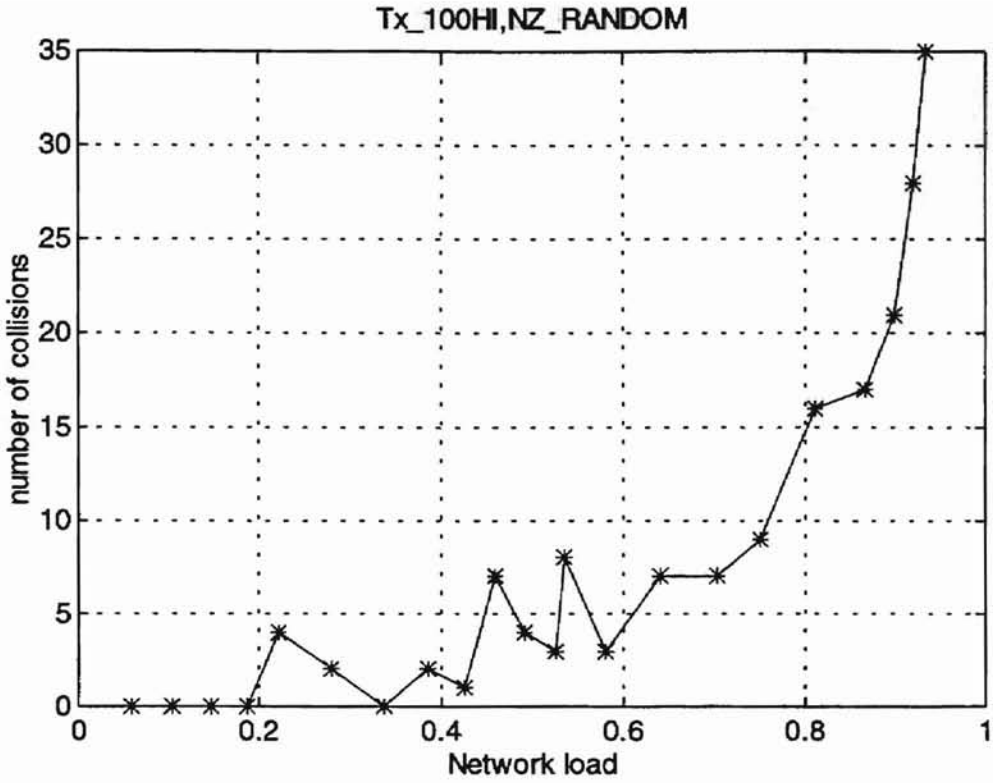


Figure C.39 Simulation result of experiment 6

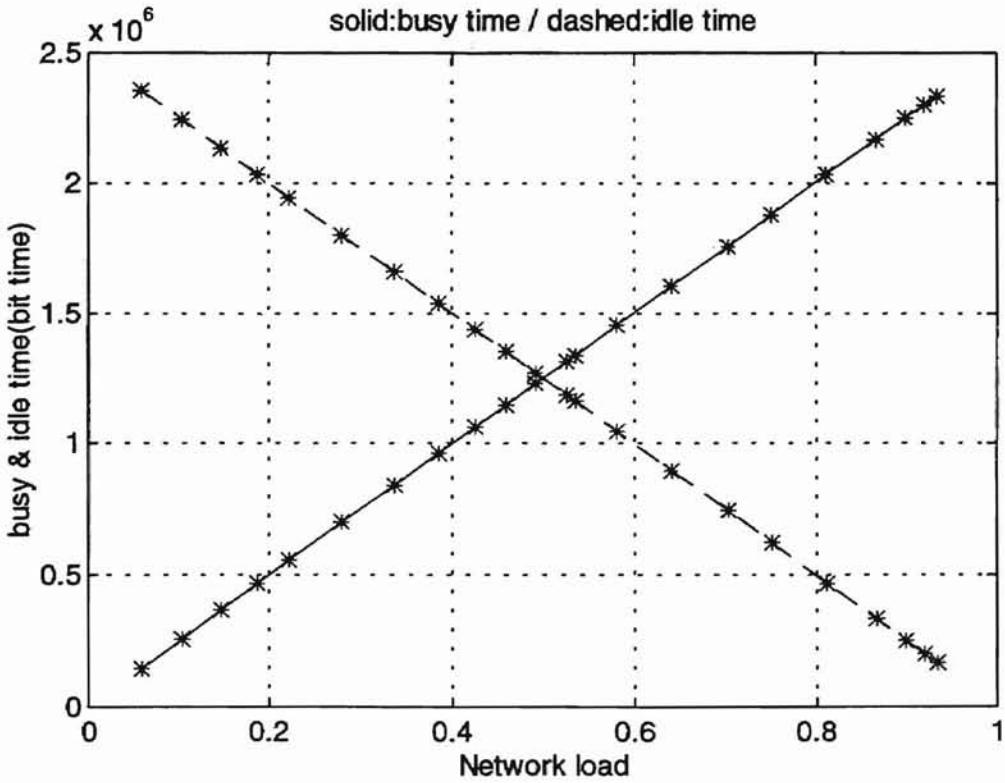


Figure C.40 Simulation result of experiment 6

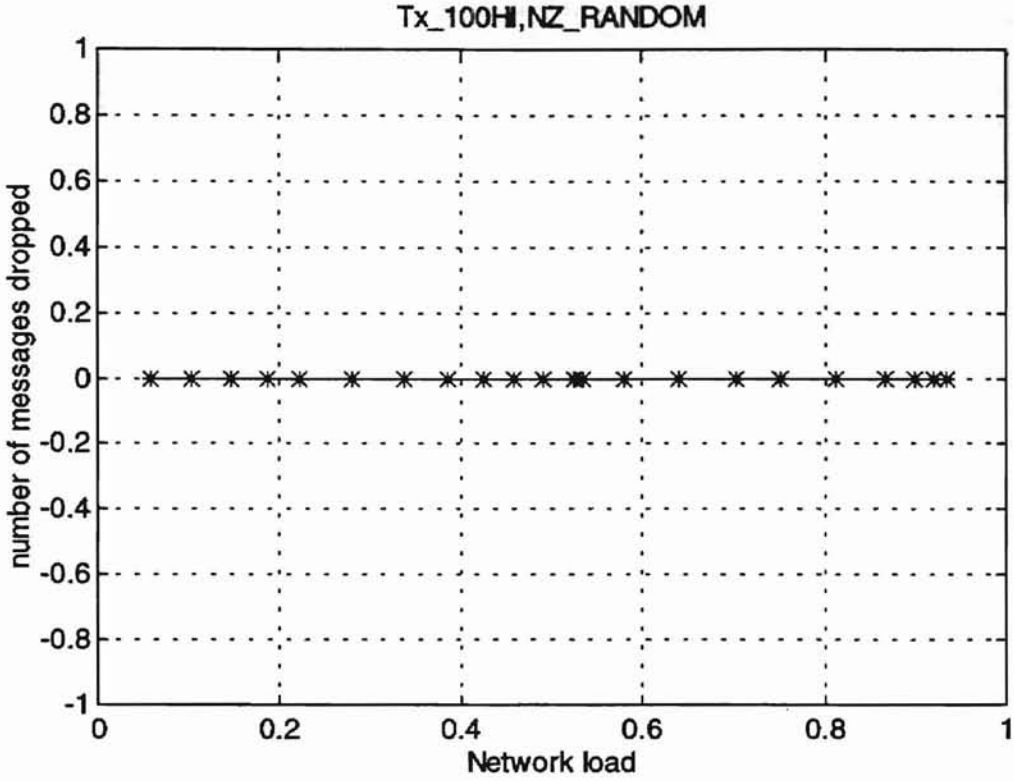


Figure C.41 Simulation result of experiment 6

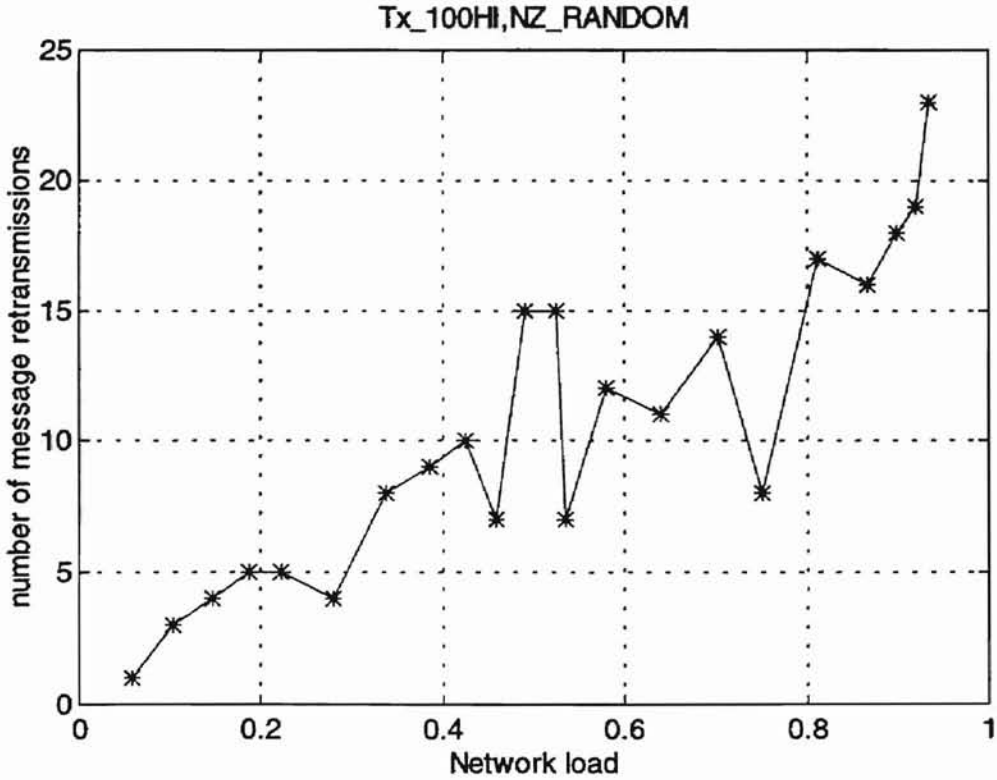


Figure C.42 Simulation result of experiment 6

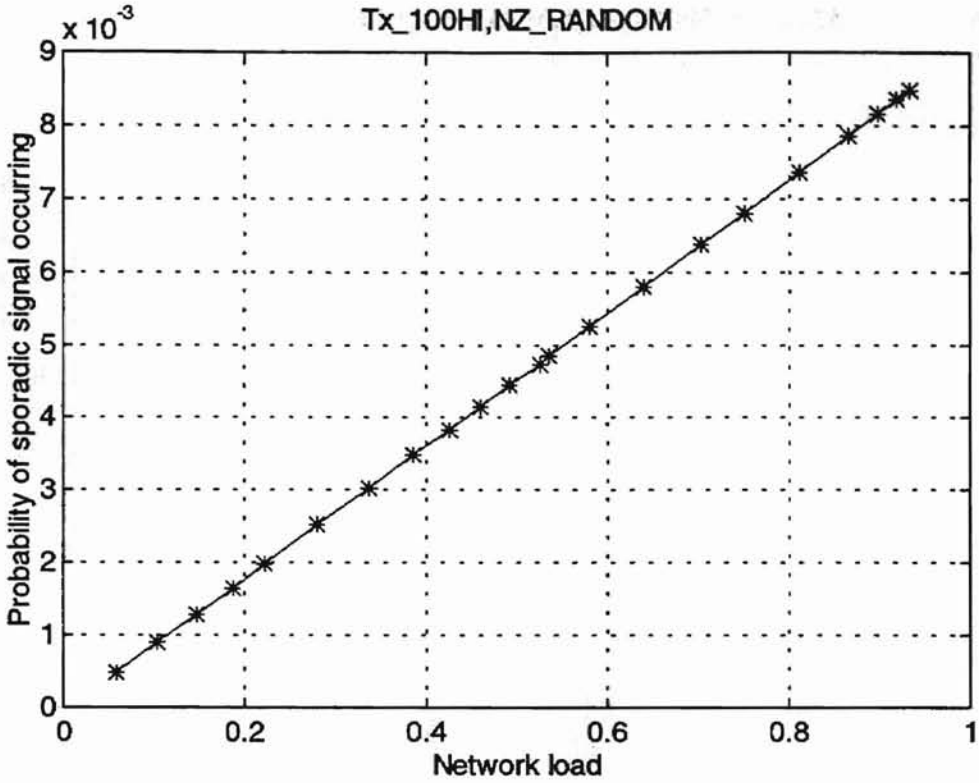


Figure C.43 Simulation result of experiment 6

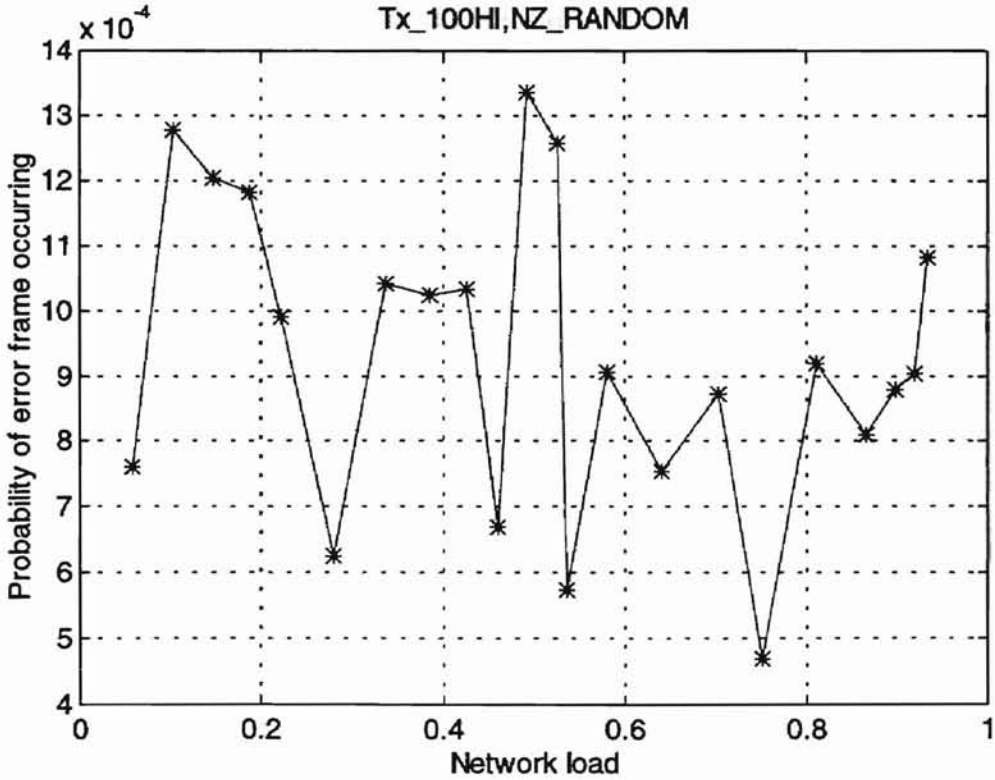


Figure C.44 Simulation result of experiment 6

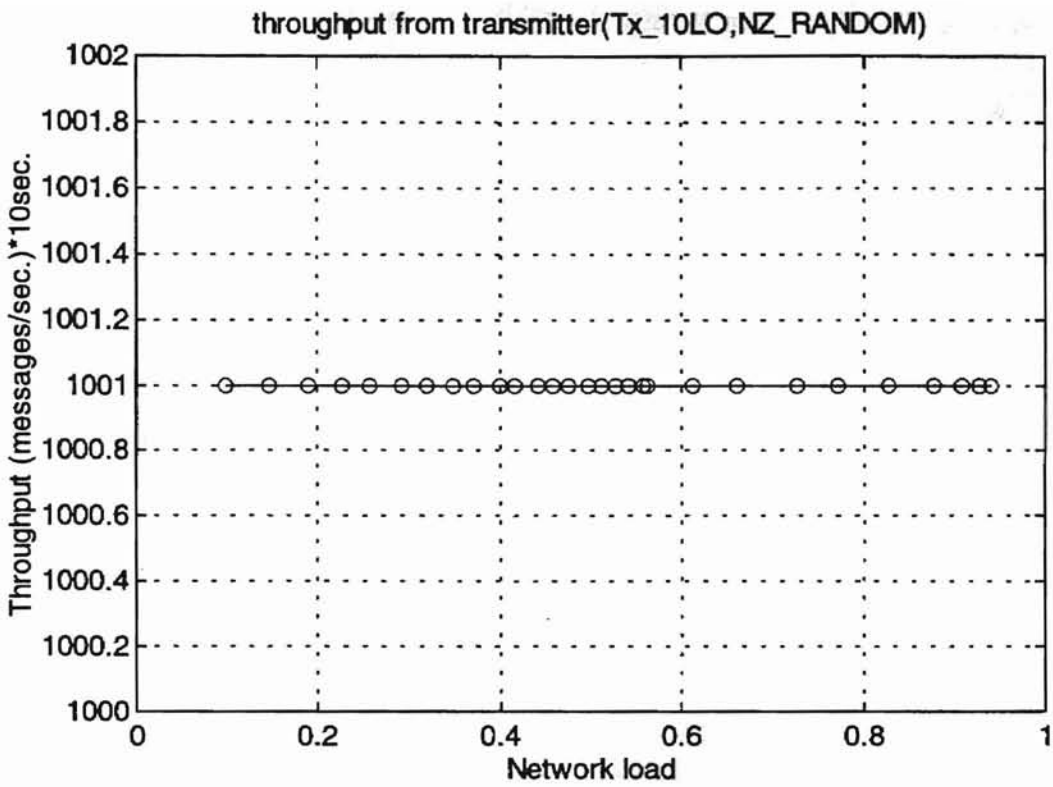


Figure C.45 Simulation result of experiment 7

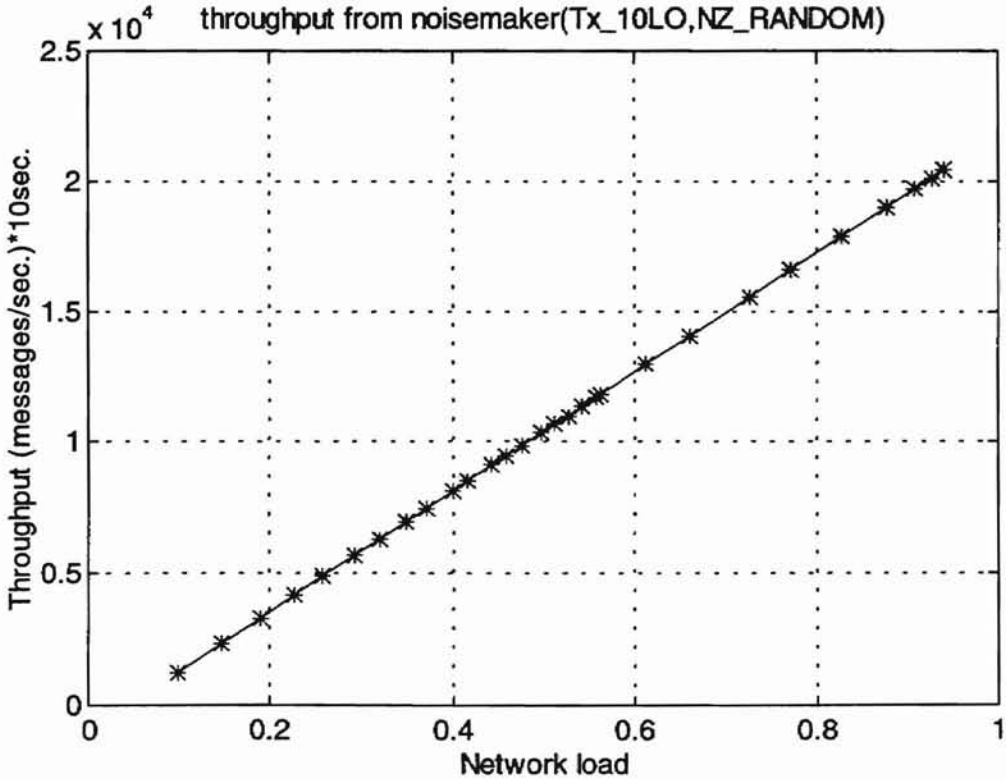


Figure C.46 Simulation result of experiment 7

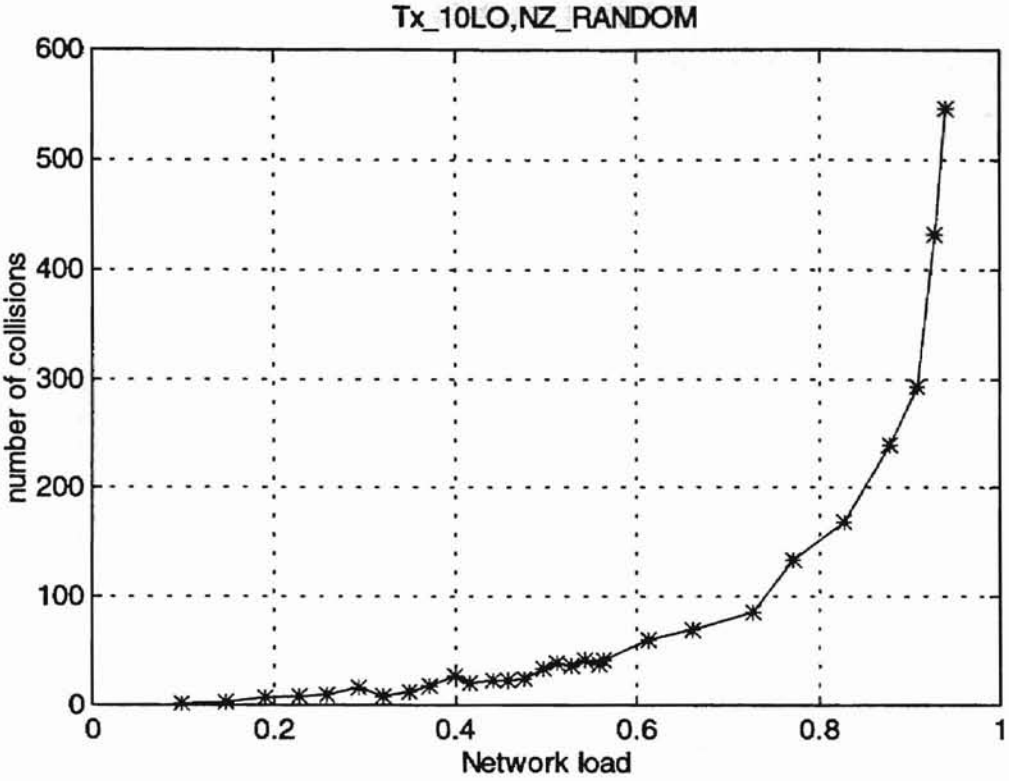


Figure C.47 Simulation result of experiment 7

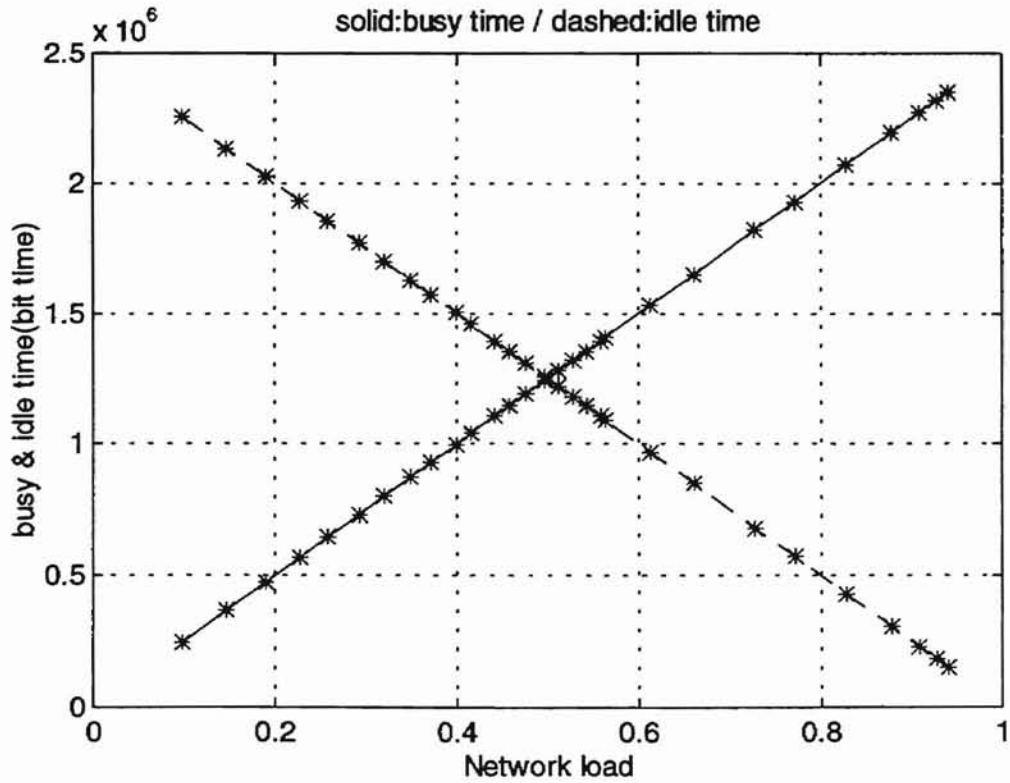


Figure C.48 Simulation result of experiment 7

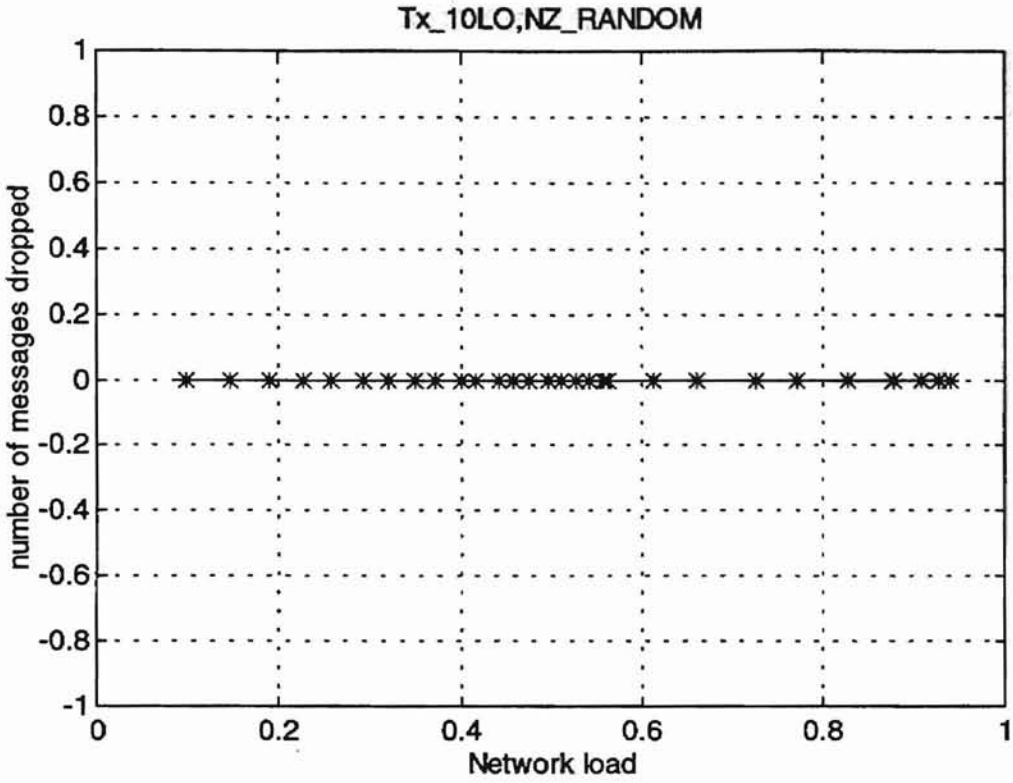


Figure C.49 Simulation result of experiment 7

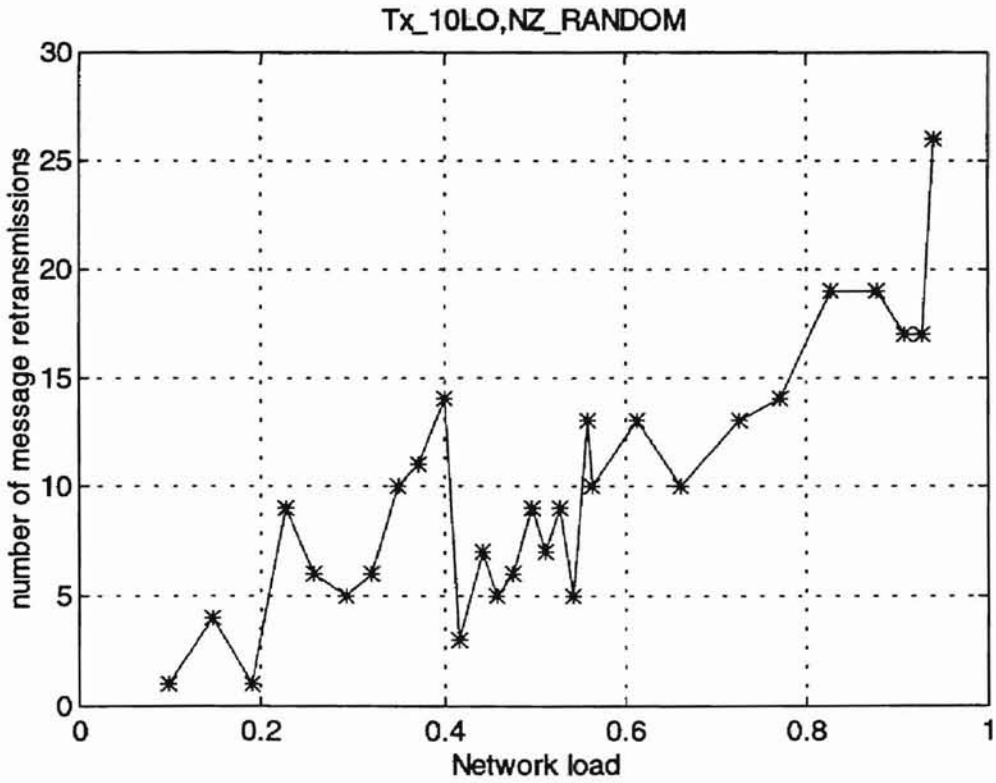


Figure C.50 Simulation result of experiment 7

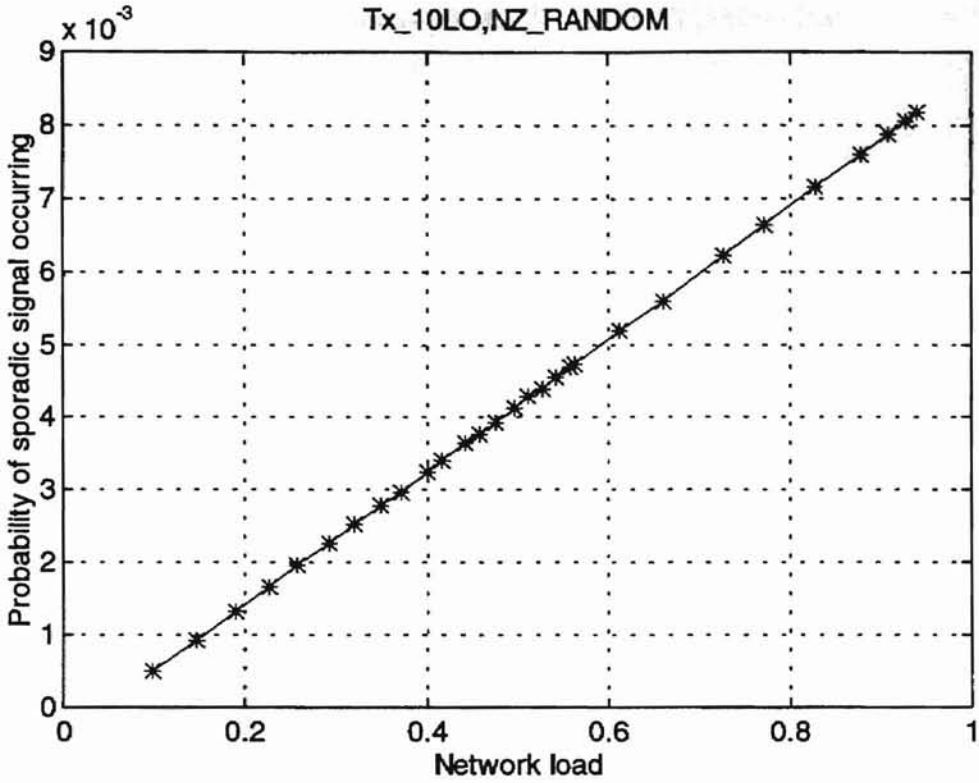


Figure C.51 Simulation result of experiment 7

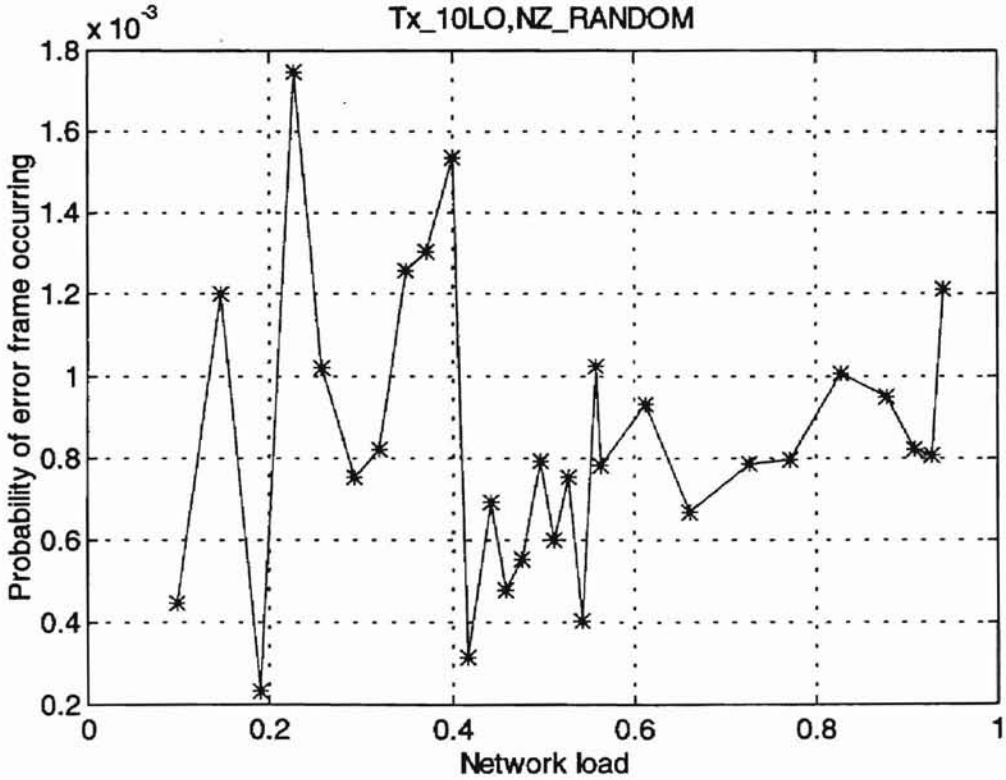


Figure C.52 Simulation result of experiment 7

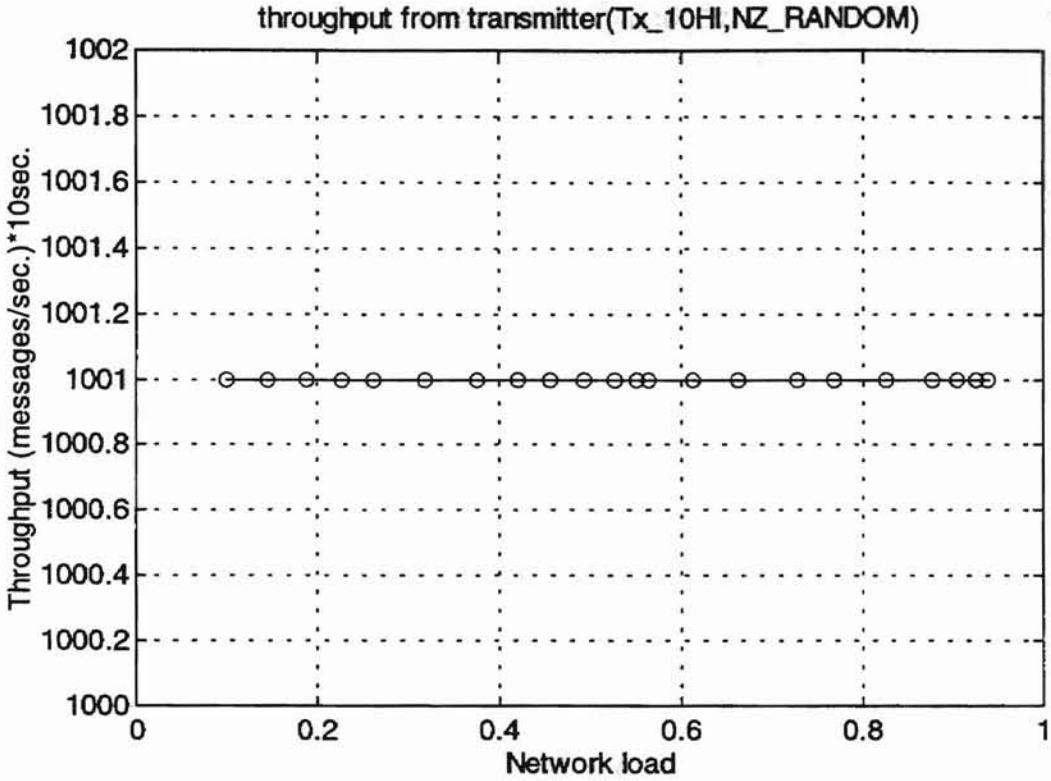


Figure C.53 Simulation result of experiment 8

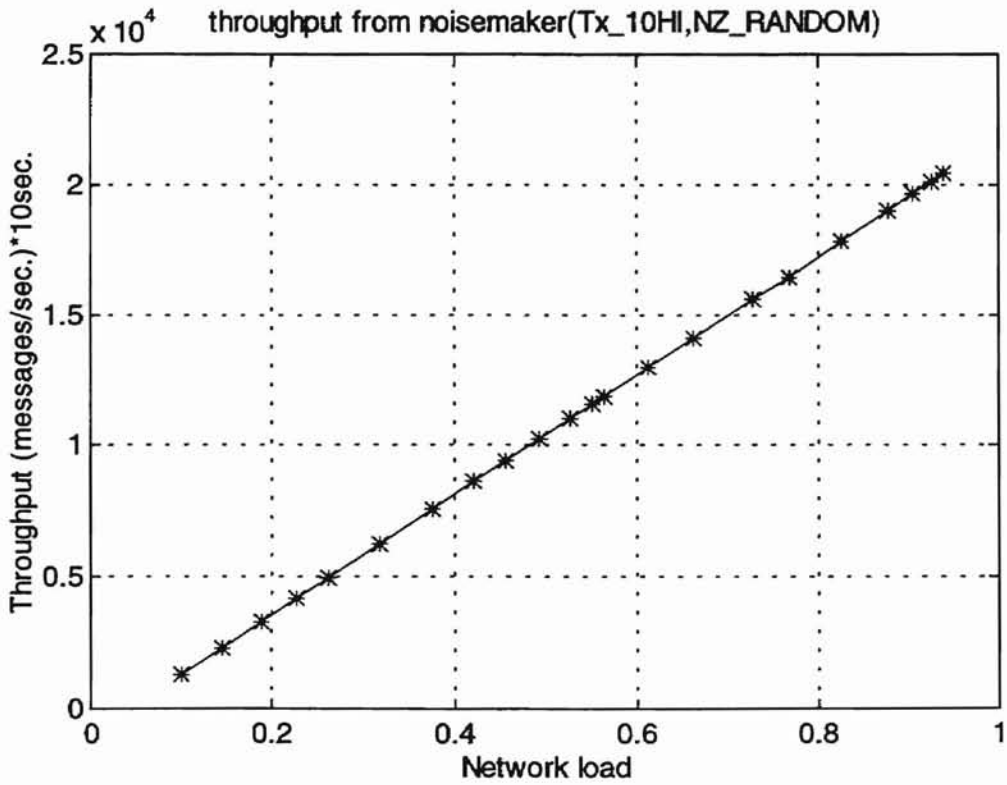


Figure C.54 Simulation result of experiment 8

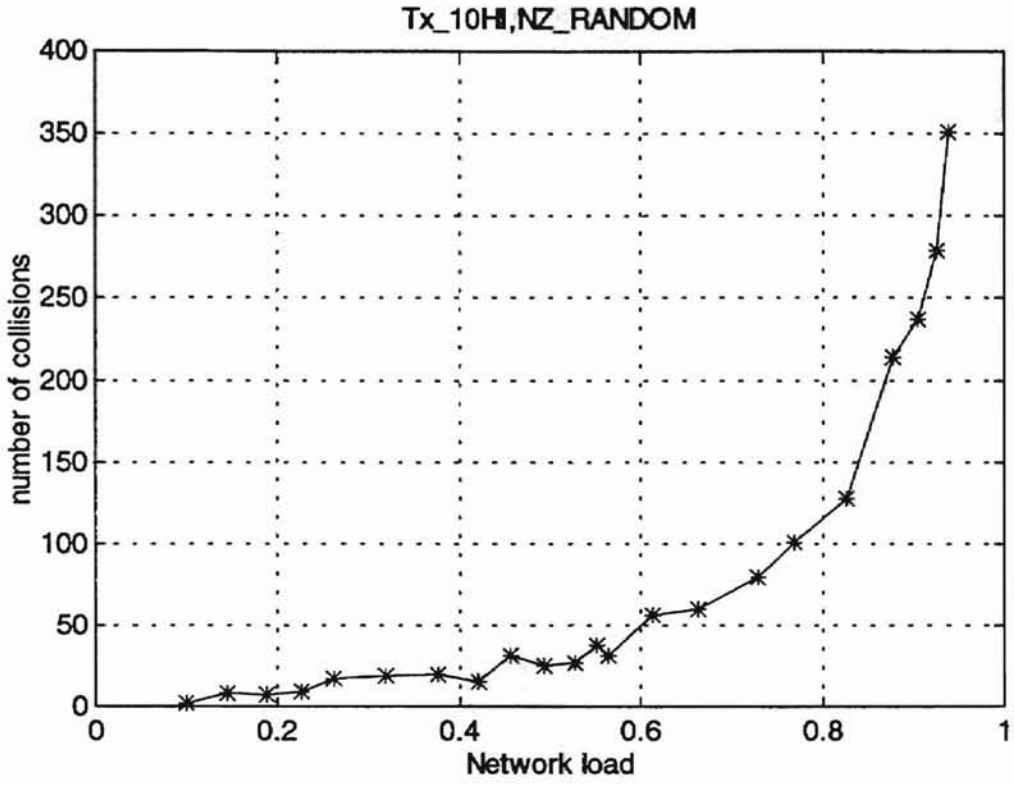


Figure C.55 Simulation result of experiment 8

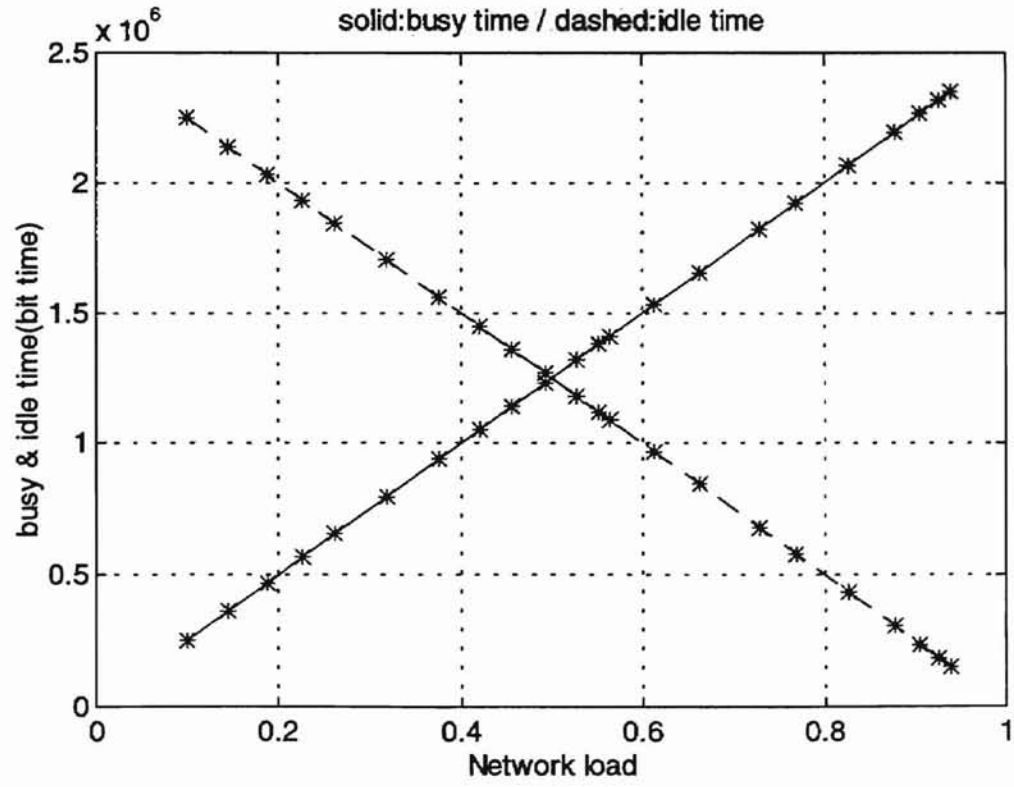


Figure C.56 Simulation result of experiment 8

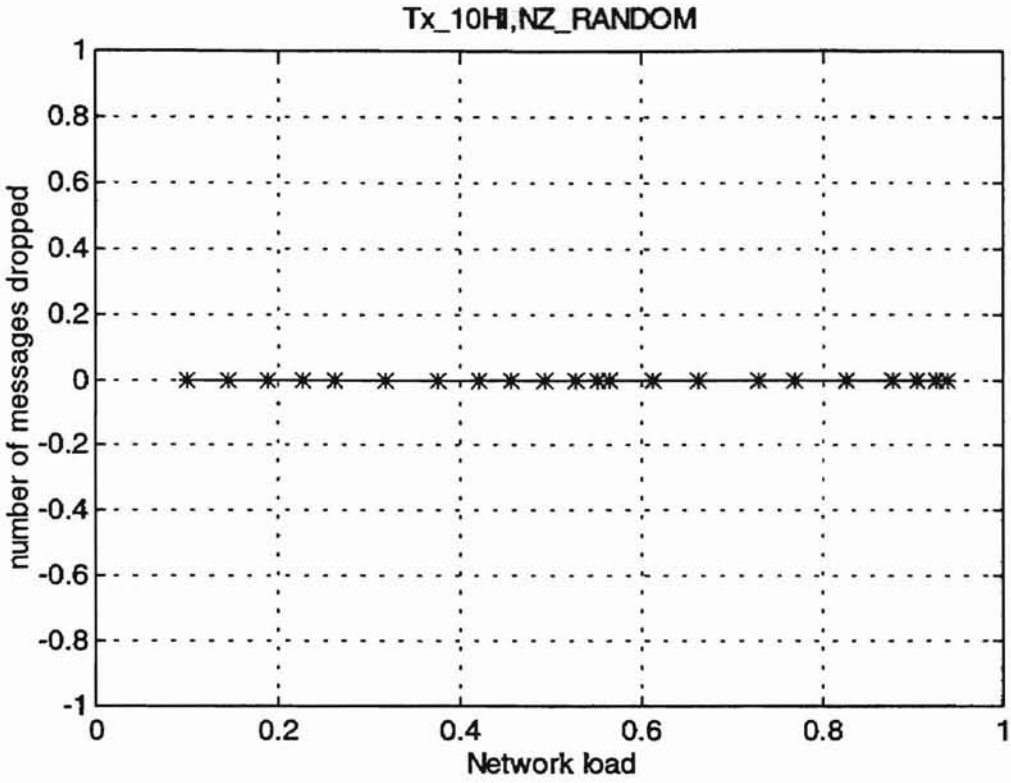


Figure C.57 Simulation result of experiment 8

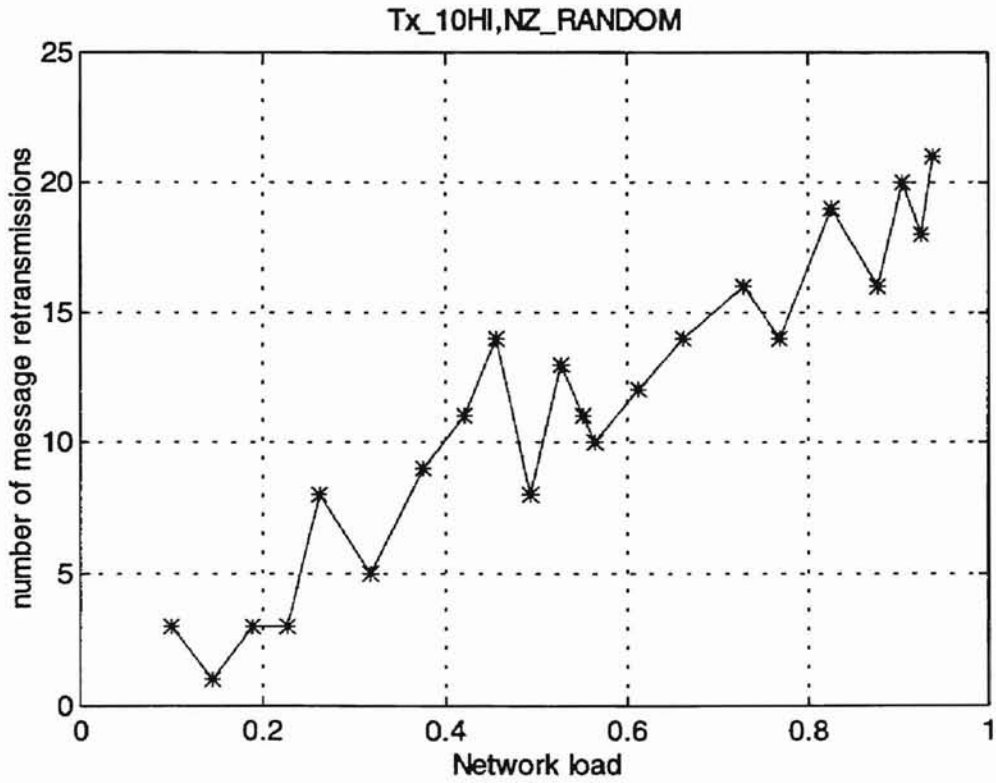


Figure C.58 Simulation result of experiment 8

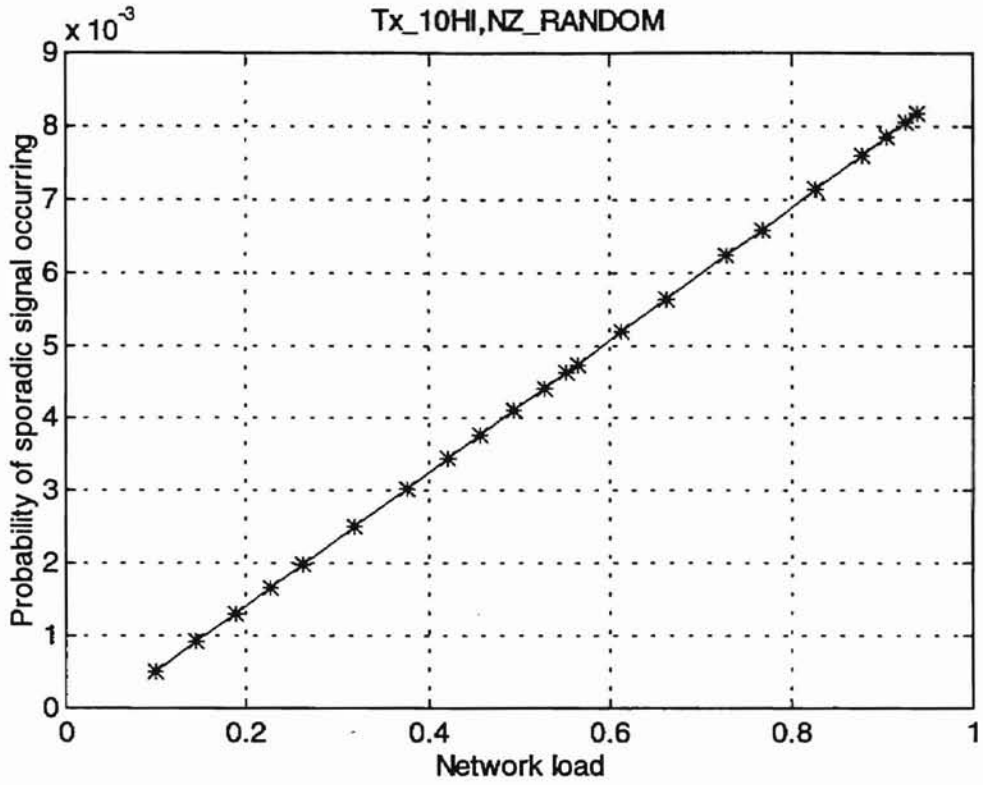


Figure C.59 Simulation result of experiment 8

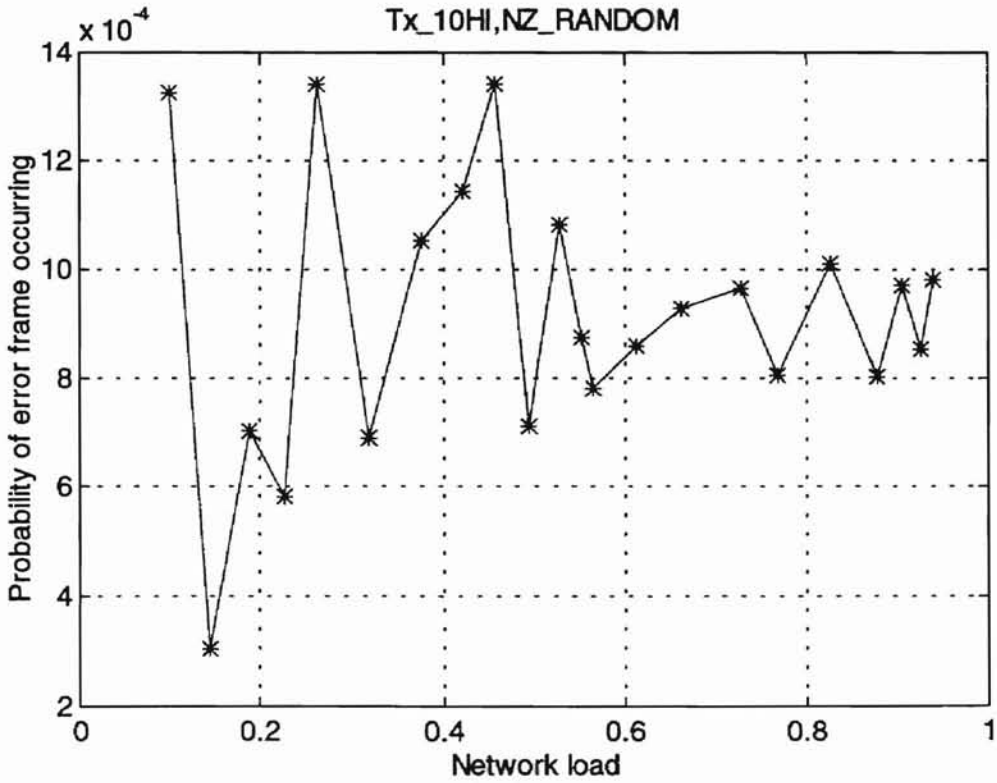


Figure C.60 Simulation result of experiment 8

APPENDIX D

CALCULATION PROGRAM OF CAN MESSAGE RESPONSE TIME

```
/* Calculate expected message response time */
/* Maximum number of message preset is 100 */
/* the formula is based on the assumption that p is small enough to make */
/* p*p=0 */
/* p = probability of an error occurring in a message transmission */
/* psp = probability of a sporadic signal occurring */

void main()
{int i,j,k,old;
 float tm[100],gg;
 int bittm[100]={0};
 int samptm[100];
 int network_speed,num,sumint;
 float p;
 float psp;
 float samprate[100];

 printf("Input the network speed in kbps :");
 scanf("%d",&network_speed);
 printf("\nInput the number of messages transmitted via CAN bus (<=100):");
 scanf("%d",&num);
 printf("\Input the probability of error occurring per message transmission :");
 scanf("%f",&p);
 printf("Input the probability of sporadic signal occurring :");
 scanf("%f",&psp);

 for (i=0;i<num;i++)
  {printf("\nInput the sampling rate of the priority %d message (in Sec.):",i+1);
   scanf("%f",&samprate[i]);
  }

  for (i=0;i<num;i++)
   samptm[i]=samprate[i]*network_speed*1000;

 bittm[0]=(254+300*p)/(1-127*psp);
 for (i=1;i<num;i++)
  {do
        {sumint=0;
         for (k=i-1;k>=0;k--)
          {j=bittm[i]/samptm[k];
           j++;
           sumint+=j;
          }
         old=bittm[i];
         gg=p*150*(2+sumint)+psp*127*bittm[i];
```

```
        gg+=254+127*sumint;
        bittm[i]=gg;
        }
 while (old!=bittm[i]);
 printf("Expected bit time of priority (%d) : %d\n",i+1,bittm[i]);
 }
 for (i=0;i<num;i++)
 {tm[i]=bittm[i];
        gg=network_speed;
        tm[i]/=gg;
        tm[i]/=1000;
        printf("Expected message response time of priority (%d): %8.7f\n",i+1,tm[i]);
 }
}
```

APPENDIX E

CAN SIMULATOR PROGRAM

```
#include "stdio.h"
#include "time.h"
#include "stdlib.h"

#define MAX_NODES 100
#define MAX_SAMPS 10

typedef struct
{char nodename[20];
 long samprate;     /* in bit time */
 long sampstart;    /* in bit time */
 long timescrew;    /* in bit time */
 long deadline;
 int messageid;
 int maskreg[MAX_SAMPS];
 int numsamps;
 int datatype;
 char arb[40];
 int prob;
 int ready;
 int status;
 int arbnum;
 int stuff;
 int state;
 int intermission;
 int delimiter;
 int overload;
 int error;
} NODE;

long tic=0;          /* in bit time */
long simtime;       /* in bit time */
long networkspeed;  /* unit : kbps  */
long collision=0;
long busytime=0;
long idletime=0;
long throughput=0;
long nummretrans=0;
long nmsgd=0;
long nmsgdrop=0;
long nmsgrej=0;
long nmsgvac=0;
long numsporadic=0;
long numferror=0;
```

```
long numbstuff=0;
long numCRCerror=0;
long numbmerror=0;
long numACKerror=0;
long numoverload3=0;
long numoverload2=0;
NODE node[MAX_NODES];
char dlc[8][4]={{'0','0','0','1'},{'0','0','1','0'},{'0','0','1','1'}, \
                                {'0','1','0','0'},{'0','1','0','1'},{'0','1','1','0'}, \
                                {'0','1','1','1'},{'1','0','0','0'}};
char fields[120],bus,prebus;
int totalnodes,datalen,data_state;
int bitstuff=0;
int overload,overload_happen;
int frame_prob1,frame_err1,frame_prob2,frame_err2;
int bstuff_prob1,bstuff_prob2,bstuff1,bstuff2;
int ACK_prob1,ACK_prob2,ACK_err1,ACK_err2;
int CRC_prob1,CRC_err1,CRC_prob2,CRC_err2;

int gen_fields(int);
int gen_arb(int);
int prob_error(int,int,int,int);
void updata(int);
void sporadic(int);
void bstuff_check(int);
void bstuff_error(void);

void main()
{ void get_input_data(void);
 void msg_cycle();

 FILE *fp;
 int i;
 float k,l;

 get_input_data();
 for (i=0;i<totalnodes;i++)
  {printf("node %d :\n",i);
        printf("samprate=%ld\n",node[i].samprate);
        printf("sampstart=%ld\n",node[i].sampstart);
        printf("timescrew=%ld\n",node[i].timescrew);
        printf("prob=%d \n",node[i].prob);
  }

 printf("simtime=%ld\n",simtime);
```

```
msg_cycle();
l=simtime;
busytime=simtime-idletime;
k=busytime;
nummretrans+=numoverload3;
printf("Network load = %f\n",k/l);
printf("Throughput = %ld\n",throughput);
printf("Bus busy time = %ld\n",busytime);
printf("Bus idle time = %ld\n",idletime);
printf("Number of collision happen = %ld\n",collision);
printf("Number of message retransmitted = %ld\n",nummretrans);
printf("Number of Message delay = %ld\n",nmsgd);
printf("Number of Message dropped = %ld\n",nmsgdrop);
printf("Number of Message rejected = %ld\n",nmsgrej);
printf("Number of vacant sampling = %ld\n",nmsgvac);
printf("Number of sporadic signal happened=%ld\n",numsporadic);

if ((fp=fopen("csoutput.dat","a+"))==NULL)
 {printf("Error opening file : csoutput.dat \n");
       exit(0);
 }
fprintf(fp,"%f ",k/l);
k=numsporadic;
printf("Probability of sporadic signal occurring=%f\n",k/l);
printf("Number of frame error occurring=%ld\n",numferror);
printf("Number of bit stuffing error occurring=%ld\n",numbstuff);
printf("Number of CRC error occurring=%ld\n",numCRCerror);
printf("Number of bit monitoring error occurring=%ld\n",numbmerror);
printf("Number of ACK error occurring=%ld\n",numACKerror);
printf("Number of overload frame occurred in first 2 bit time=%ld\n",numoverload2);
printf("Number of overload frame occurred in 3rd bit time=%ld\n",numoverload3);

fprintf(fp,"%ld %ld %ld %ld ",throughput,collision,busytime,idletime);
fprintf(fp,"%ld %ld %ld %ld %ld ",nmsgd,nmsgdrop,nmsgrej,nmsgvac,nummretrans);
fprintf(fp,"%f ",k/l);      /* Probability of sporadic signal occurring */
k=nummretrans;l=throughput;
fprintf(fp,"%f\n",k/l);     /* Probability of error frame occurring */
                                                    /* per message transmission */
fclose(fp);
}

/* data_state=0 ----> send start of frame */
/* data_state=1 ----> generate arbitration field & send the first bit of it */
/* data_state=2 ----> during arbitration (competition) */
/* data_state>2 ----> after arbitration */
```

```
/* node[i].ready=1 -> node i has message to send */
/* node[i].ready=2 -> node i is in competition */
/* node[i].ready=3 -> node i is the current message transmitter */
/* node[i].state=1 -> node i is in intermission status */
/* node[i].state=2 -> node i is in overload frame status */
/* node[i].state=3 -> node i is in error frame status */
/* node[i].state=4 -> node i is in data frame status */
/* node[i].state=5 -> node i is in idle status */
/* Bit stuffing, frame(form) error is generated by message transmitter */
/* but detected by all nodes  */
/* CRC, ACK error is generated by nontransmitter and detected by transmitter */
/* Bit monitoring error is detected by message transmitter */

void msg_cycle(void)
{int checknode(int);
 FILE *fp;
 int i,r,ggyy=0;
 int fieldnum,flag;
 int data_type;

 randomize();
 do {tic++;
        if (data_state>1 && bitstuff!=1)
         flag++;
        if (bitstuff!=1)
         prebus=bus;
        else
         {bitstuff=0;
          prebus='*';
         }
        bus='1';

        /* Simulate what each node will do */
        /* during the first part (transmission) of each bit time */

 for (i=0;i<totalnodes;i++)
  {if (node[i].ready<2)
        {updata(i);
         sporadic(i);
        }
   switch(node[i].state)
        {case 1:node[i].intermission++;
                        if (i==totalnodes-1)       /* generate overload frame */
                         {r=random(overload)+1;
                          if (r<=overload_happen)
```

```
                    {bus='0';
                          if (node[i].intermission==3)
                           {r=random(totalnodes);
                            node[r].state=2;
                            node[r].intermission=0;
                            numoverload3++;
                           }
                    }
                   }
                   break;
case 2:if (node[i].overload<6)
                   {bus='0';
                    node[i].overload++;
                   }
                   if (node[i].delimiter>0)
                    node[i].delimiter++;
                   break;
case 3:if (node[i].error<6)
                   {bus='0';
                    node[i].error++;
                   }
                   if (node[i].delimiter>0)
                    node[i].delimiter++;
                   break;
case 4:if (data_state==0 && node[i].ready==1)
                   {bus='0';
                    break;
                   }
                  if (data_state==1 && node[i].ready==1)
                   {ggyy++;
                    node[i].arb[0]='0';
                    node[i].arbnum=gen_arb(i);      /* Generate arbitration field */
                    flag=1;
                    r=checknode(i);
                    if (r)
                     {if (ggyy>1)
                           collision++;          /*  count message collision occurring */
                          ggyy=0;
                     }
                    if (bus=='1' && node[i].arb[flag]=='0')
                     bus='0';
                    node[i].ready++;
                    break;
                   }
                  if (node[i].ready==2 && data_state==2)
```

```
                        {if (node[i].stuff==5)
                         {if (prebus=='1')
                                bus='0';
                               bitstuff=1;
                               break;
                         }
                         if (bus=='1' && node[i].arb[flag]=='0')
                          bus='0';
                         break;
                        }
                        if (data_state>2 && flag==8*datalen+22 && node[i].ready!=3)
                         {r=prob_error(ACK_prob1,ACK_err1,ACK_prob2,ACK_err2);
                          if (r)              /* generate ACK error */
                           node[i].state=6;
                          else
                           bus='0';
                          break;
                         }
                        if (node[i].ready==3 && data_state>2)
                         {if (node[i].stuff==5 && flag<=20+8*datalen)
                           {bstuff_error();
                                 bitstuff=1;
                           }
                          else
                           {if (fields[flag]=='0')
                                  bus=fields[flag];
                           }
                         }
        case 5:break;
        }
}

/* Simulate what each node will do */
/* during the second part (listening) of each bit time */

for (i=0;i<totalnodes;i++)
 {switch(node[i].state)
        {case 1:if (bus=='0')
                        {if (node[i].intermission<=2)
                         {node[i].state=2;
                          node[i].intermission=0;
                          if (i==totalnodes-1)
                                numoverload2++;
                         }
                        else
```

```
                    {node[i].state=4;
                     data_state=1;
                     node[i].intermission=0;
                     node[i].stuff=1;
                     bitstuff=0;
                    }
                   }
                  else
                   {if (node[i].intermission==3)
                    {node[i].intermission=0;
                          if (node[i].ready==0)
                           node[i].state=5;
                          else
                           {node[i].state=4;
                            data_state=0;
                            node[i].stuff=0;
                           }
                    }
                   }
          break;
case 2:if (bus=='1' && node[i].delimiter==0)
                   node[i].delimiter=1;
                  if (node[i].delimiter==8)
                   {node[i].state=1;
                    node[i].delimiter=0;
                    node[i].overload=0;
                   }
                  break;
case 3:if (bus=='1' && node[i].delimiter==0)
                   node[i].delimiter=1;
                  if (node[i].delimiter==8)
                   {node[i].state=1;
                    node[i].delimiter=0;
                    node[i].error=0;
                   }
                  break;
case 4:if (bus=='0'&& data_state==0)
                   {node[i].stuff=1;
                    r=checknode(i);
                    if (r)
                     data_state++;
                    break;
                   }
                   if (data_state==1)
                    {if (node[i].ready==2 && node[i].arb[flag]!=bus)
```

```
        node[i].ready--;
  bstuff_check(i);
  r=checknode(i);
  if (r)
        data_state++;
  break;
  }
 if (data_state==2)
  {bstuff_check(i);
  if (node[i].stuff==6)
        {node[i].state=3;
         if (node[i].ready==2)
          node[i].ready--;
         break;
        }
  if (bitstuff)
         break;
  if (node[i].ready==2)
        {if (node[i].arb[flag]!=bus)
         node[i].ready--;
        else
         {if (node[i].arbnum==flag)
                {node[i].ready++;
                 data_state++;
                 data_type=node[i].datatype;
                 fieldnum=gen_fields(i);
                 flag=-1;
                }
         }
        }
  break;
 }
if (data_state>2)
 {if (flag<=20+8*datalen)
  {bstuff_check(i);
        if (node[i].stuff==6)          /* check bit stuffing error */
         {node[i].state=3;
         if (node[i].ready==3)
          {node[i].ready=1;
                nummretrans++;
                if (flag<=8*datalen+20 && flag>=6+8*datalen)
                 numCRCerror++;
          }
         break;
         }
```

```
        if (bitstuff)
         break;
 }
if (node[i].ready!=3)
 {if (flag==0 || flag==1)          /* check frame error */
        {if (data_type==1 && bus=='0')
          node[i].state=3;
         if (data_type==0 && bus=='1')
          node[i].state=3;
         break;
        }
       if (flag==8*datalen+21 || flag==8*datalen+23)
        {if (bus=='0')
          node[i].state=3;
         break;
        }
       if (flag<=20+8*datalen && flag>=6+8*datalen)
   {r=prob_error(CRC_prob1,CRC_err1,CRC_prob2,CRC_err2);
        if (r)        /* generate CRC error */
         node[i].state=3;
         break;
        }
 }
if (node[i].ready==3)          /* check bit monitoring error */
 {if (flag!=8*datalen+22)
       {if (bus!=fields[flag])
         {node[i].state=3;
               node[i].ready=1;
               numbmerror++;
               nummretrans++;
               break;
         }
       }
       if (flag==0 || flag==1)          /* check frame error */
        {if (data_type==1 && bus=='0')
          {node[i].state=3;nummretrans++;node[i].ready=1;break;}
         if (data_type==0 && bus=='1')
          {node[i].state=3;nummretrans++;node[i].ready=1;break;}
        }
       if (flag==8*datalen+21 || flag==8*datalen+23)
        if (bus=='0')
         {node[i].state=3;nummretrans++;node[i].ready=1;break;}
       if (flag==8*datalen+22 && bus=='1')
        {node[i].ready=1;                          /* Check ACK error*/
         node[i].state=3;
```

```
                    nummretrans++;
                    numACKerror++;
                    break;
                   }
                 }
                 if (flag>8*datalen+23 && flag<8*datalen+30)
                 {if (bus=='0')                    /* check end of frame */
                    {node[i].state=3;
                     if (node[i].ready==3)
                      {node[i].ready=1;nummretrans++;}
                     break;
                    }
                 }
                }
                if (data_state>2 && flag==fieldnum)
                 {node[i].state=1;
                  if (node[i].ready==3)
                   {if (tic<=node[i].deadline)              /* check msg condition */
                      {if (node[i].status==1) nmsgrej++;   /* status=0 --> O.K.*/
                       node[i].status=0;                    /* status=1 --> delay */
                      }                          /* from 0 to 1 --> vacant sampling */
                     else {nmsgd++;              /* from 1 to 0 --> msg rejectted */
                           if (node[i].status==0) nmsgvac++;
                            node[i].status=1;
                           }
                     node[i].ready=0;
                     throughput++;
                   }
                  }
                 break;
case 5:if (bus=='0' || node[i].ready==1)
                 {node[i].state=4;
                  if (bus!='0')
                   {data_state=0;
                      node[i].stuff=0;
                   }
                  else
                   {node[i].stuff=1;
                      bitstuff=0;
                   }
                 }
                if (i==totalnodes-1 && bus=='1')
                 idletime++;
                break;
case 6:node[i].state=3;
```

```
        }
 }
  }
 while (tic<=simtime);
}
int checknode(int i)
{int r;
 for (r=i+1;r<totalnodes;r++)
  {if (node[r].state==4 && node[r].ready==1)
        return 0;
  }
 return 1;
}
int prob_error(int prob1,int err1,int prob2,int err2)
{int r;
 r=random(err1)+1;
 if (r<=prob1)
  {r=random(err2)+1;
   if (r<=prob2)
        return 1;
   else
        return 0;
  }
 else
  return 0;
}
void bstuff_error(void)      /* Generate bit stuffing error */
{int r;
 r=prob_error(bstuff_prob1,bstuff1,bstuff_prob2,bstuff2);
 if (r)
  {numbstuff++;
   if (prebus=='0')
        bus='0';
  }
 else
  {if (prebus=='1')
        bus='0';
  }
}
void bstuff_check(int i)          /* Check bit stuffing */
{if (bus==prebus)
 node[i].stuff++;
 else
 node[i].stuff=1;
}
```

```
int gen_arb(int i)          /* generate arbitration field */
{char temp[35];
 int k,num,j=1,stuff;
 temp[0]='0';
 num=node[i].messageid;
 do {if (num%2==0)
          temp[j]='0';
         else
          temp[j]='1';
         num/=2;
         j++;
         if (node[i].datatype==1 && j==19)
          {temp[j++]='1';
           temp[j++]='1';
          }
         }
 while (num!=0);
 while (node[i].datatype==0 && j<12)
  temp[j++]='0';
 while (node[i].datatype==1 && j<32)
  temp[j++]='0';
 if (node[i].datatype==1)
  {temp[19]='1';
   temp[20]='1';
  }
 k=1;
 for (j=j-1;j>=0;j--)
  node[i].arb[k++]=temp[j];
 return k-1;
}
int gen_fields(int i)          /* generate control, data, CRC, end of frame fields */
{char CRC_GEN[15]={'1','0','0','1','1','0','0','1','1','0','1','0','0','0','1'};
 char temp[15]={'0'};
 int CRCNXT,r,j,k,n,l,m,frameflag=0;
 r=prob_error(frame_prob1,frame_err1,frame_prob2,frame_err2);
 if (node[i].datatype==1)              /* generate control field */
  {if (r)
         {fields[0]='0';   /* generate frame error for extended data frame */
          fields[1]='1';
          frameflag++;
         }
   else
         {fields[0]='1';
          fields[1]='1';}
  }
```

```
else
 {if (r)              /* generate frame error for standard data frame */
        {fields[0]='0';
         fields[1]='1';
         frameflag++;
        }
 else
        {fields[0]='0';
         fields[1]='0';}
 }
datalen=random(8)+1;
for (j=0;j<4;j++)
 fields[j+2]=dlc[datalen-1][j];
j+=2;
for (k=0;k<8*(datalen);k++)          /* generate data field */
 {n=random(2);
 if (n==1)
        fields[j]='1';
 else
        fields[j]='0';
 j++;
 }
for (k=0;k<=node[i].arbnum+j;k++)      /* generate CRC SEQUENCE */
 {if (k<=node[i].arbnum)
        {if (node[i].arb[k]!=temp[14])
         CRCNXT=1;
        else
         CRCNXT=0;
        if (k==node[i].arbnum)
         m=k;
        }
 else
        {if (fields[k-m-1]!=temp[14])
         CRCNXT=1;
        else
         CRCNXT=0;
        }
 for (l=13;l>=0;l--)
        temp[l+1]=temp[l];
 temp[0]='0';
 if (CRCNXT)
        {for (l=0;l<15;l++)
         {if (temp[l]!=CRC_GEN[l])
                temp[l]='1';
          else
```

```
                temp[l]='0';
            }
          }
 }
 for (k=0;k<15;k++)
  fields[j++]=temp[k];
 r=prob_error(frame_prob1,frame_err1,frame_prob2,frame_err2);;
 if (r)                          /* generate frame error */
  {fields[j++]='0';frameflag++;}
 else
  fields[j++]='1';               /* CRC delimiter */
 fields[j++]='1';                /* ACK slot */
 r=prob_error(frame_prob1,frame_err1,frame_prob2,frame_err2);;
 if (r)                          /* generate frame error */
  {fields[j++]='0';frameflag++;}
 else
  fields[j++]='1';               /* ACK delimiter */
 for (k=0;k<7;k++)                                  /* End of frame */
  {fields[j]='1';
       j++;
  }
 if (frameflag>0)
  numferror++;
 return j-1;
}
void updata(int i)
{FILE *fp;
 int p;
 if (tic>=node[i].sampstart && node[i].samprate!=0 && node[i].timescrew!=0)
  {if (node[i].ready==1)
         {if ((fp=fopen(node[i].nodename,"a+"))==NULL)
                {printf("Error opening file :%s\n",node[i].nodename);
                 exit(0);
                }
          fprintf(fp,"%s : %d message dropped before processing\n",
                   node[i].nodename,node[i].numsamps);
          fprintf(fp,"at tic time =%ld\n",node[i].sampstart);
          nmsgdrop++;
          fclose(fp);
          }
         node[i].ready=1;
         node[i].deadline=node[i].sampstart+node[i].timescrew;
   node[i].sampstart+=node[i].samprate;
  }
}
```

```
void sporadic(int i)          /* generate sporadic signals  */
{int j;
 if (node[i].samprate==0 && node[i].ready==0)
  {j=random(node[i].prob)+1;
   if (j<=node[i].sampstart)
          {node[i].ready=1;numsporadic++;
           node[i].deadline=tic+node[i].timescrew;
          }
  }
}

void get_input_data(void)
{ FILE *fp;
  int i,j;
  float f;
  char filename[20];
  char input_filename[24];

  fprintf(stdout,"\nEnter File name (No extension):");
  fscanf(stdin,"%s",filename);
  sprintf(input_filename,"%s.dat",filename);
  if ((fp=fopen(input_filename,"r"))==NULL)
   {fprintf(stderr,"Unable to open input file !\n");
          exit(-1);
   }

  fscanf(fp,"%ld",&simtime);
  fscanf(fp,"%ld",&networkspeed);
  simtime*=networkspeed*1000;  /* convert to bit time */
  fscanf(fp,"%d",&totalnodes);
  fscanf(fp,"%d",&overload_happen);
  fscanf(fp,"%d",&overload);
  fscanf(fp,"%d",&frame_prob1);
  fscanf(fp,"%d",&frame_err1);
  fscanf(fp,"%d",&frame_prob2);
  fscanf(fp,"%d",&frame_err2);
  fscanf(fp,"%d",&bstuff_prob1);
  fscanf(fp,"%d",&bstuff1);
  fscanf(fp,"%d",&bstuff_prob2);
  fscanf(fp,"%d",&bstuff2);
  fscanf(fp,"%d",&CRC_prob1);
  fscanf(fp,"%d",&CRC_err1);
  fscanf(fp,"%d",&CRC_prob2);
  fscanf(fp,"%d",&CRC_err2);
  fscanf(fp,"%d",&ACK_prob1);
```

```
fscanf(fp,"%d",&ACK_err1);
fscanf(fp,"%d",&ACK_prob2);
fscanf(fp,"%d",&ACK_err2);

/* unit of the sampling rate in input file is sec. per time */
/* unit of the time screw in input file is sec. per time */

for (i=0;i<totalnodes;i++)
 {fscanf(fp,"%s",node[i].nodename);
        fscanf(fp,"%f",&f);
        node[i].samprate=networkspeed*1000*f;  /* convert to bit time */
        if (node[i].samprate!=0)
         {fscanf(fp,"%f",&f);
          node[i].sampstart=networkspeed*1000*f; /* convert to bit time */
         }
        else
         {fscanf(fp,"%d",&node[i].sampstart);
          fscanf(fp,"%d",&node[i].prob);
         }
        fscanf(fp,"%f",&f);
        node[i].timescrew=networkspeed*1000*f; /* convert to bit time */
        fscanf(fp,"%d",&node[i].messageid);
        fscanf(fp,"%d",&node[i].datatype);
        fscanf(fp,"%d",&node[i].numsamps);
        for (j=0;j<node[i].numsamps;j++)
         fscanf(fp,"%d",&node[i].maskreg[j]);
        node[i].state=1;
        node[i].ready=0;
        node[i].status=-1;
        node[i].intermission=0;
        node[i].delimiter=0;
        node[i].overload=0;
        node[i].error=0;
 }
 fclose(fp);
}
```

VITA

Liang-Wei Ho

Candidate for the Degree of

Master of Science

Thesis : MESSAGE RESPONSE TIME ANALYSIS OF CAN-BASED CONTROL SYSTEMS

Major Field : Mechanical Engineering

Biographical :

Personal Data : Born in Taipei, Taiwan, On April 26, 1969

Education : Graduate from Chung Cheng High School, Taipei, Taiwan in May 1988; received Bachelor of Science degree in Mechanical Engineering from Tamkang University, Taipei, Taiwan in January, 1993. Completed the requirements for the Master of Science degree with a major in Mechanical Engineering at Oklahoma State University in July, 1997.

Experience : Employed by Combustion and Heat Transfer Co., Taipei, Taiwan, R.O.C. as a Mechanical Engineer from January, 1993 to May, 1994; employed as a Sale & Mechanical Engineer by Great Wall Industrial Co., Taipei, Taiwan, R.O.C. from May, 1994 to July, 1995; employed by Oklahoma State University, School of Mechanical and Aerospace Engineering as a Teaching Assistant from Jan. 1997 to present.