Reliability and Delay Analysis of Control Networks

Fayyaz Ahmed

University of Ontario Institute of Technology,
Oshawa, ON, CANADA

Summary
In this paper we have addressed the delay problem for real time control system. Real time control systems response is critical for any abnormal condition. We have modeled a communication and control system for system delay by using dymonda software that describes both the system hardware and software conditions. We have programmed an echo server and client application on VMware player for a parallelized client server delay computation and a single program, parent child thread application to compute elapsed time for resource allocation for parent and child threads in Matlab. We have modeled a communication network delay for a proposed CANDU SCWR Hydrogen co-generation unit, using state of the art Dynamic Flow Graph (DFM) methodology to analyze the system delays. Model expressions and results yield insight into the design of communication delay model.

Key words:
DFM model, Network Delays, Echo client-server,

1. Introduction

System reliability and safety analysis has always been an important topic in industry and reliable communication network can enhance system safety and reliability.
Echo server is similar to IPCONFIG command that echo calls to all the associated hardware devices. In control network the echo server is an application receiving multiple communications from echo clients. We can assume echo server as a Structured Query Language (SQL) database application accepting connections of echo clients through their unique mac or IP address. Sensors and actuator nodes lack memory and transmit information in the form of unified data and control information on packets with embedded mac address. In our Echo model we have programmed an echo server call and an echo client response for a client server connection for a clustered control network. Our model uses the multiple parallelized thread or hardware clusters using Flynn's taxonomy.
The paper is organized into eight sections. Section two is the background section that develops the DFM methodology knowledge base. Section three defines the network delay model and levels of its control. In section four and five the java coding of echo server and echo client is described. The logic for concurrent
parallelized model is explained in section seven of the paper and total computed delay for context switching and interrupt latency are shown. Section eight discusses the outcomes of the computations.

2. Background

CANDU SCWR is proposed nuclear power plant model that can generate electricity with lower cost and improved efficiency [1]. The Ontario region of Canada is producing 22000 MWe [2] of electricity (on average) from both conventional and non conventional sources and only 68 percent of it is utilized by the consumers. We assume that this energy network has the CANDU SCWR unit and the excess heat and power generated can be integrated to a carbon free hydrogen production unit. This integration not only can improve the power generation throughput but can also produce carbon free hydrogen fuel.
Amongst various risk assessment methodologies the dynamic flowgraph methodology (DFM) [3-6] is considered as an advance approach for a dynamic system reliability analysis. A dynamic system can be modeled and analyzed for reliability and safety through DFM. In a dynamic system DFM model, the variables and control system are represented through a time based logical cause and effect relationship. The results generated through DFM analysis are multi valued discrete events that can be initial, intermediate, and final events. These set of events that generate an unanticipated condition due to software logic errors, hardware failures and environmental conditions or can generate anticipated conditions representing the system behavior.
The DFM model works on two algorithms and it contains various components that are described as below:

2.1 Deductive Algorithm:

It is the backward analysis from effects to causes. The backward analysis finds the set of variables through nodes, edges, transfer and transition boxes that generate a set of prime implicants.

2.2 Inductive Algorithm:

It is the forward cause and effect analysis. For a particular set of inputs, the outputs can be normal outputs or may contain errors. The desired states can verify the system

Manuscript received March 5, 2011
Manuscript revised March 20, 2011

requirements whereas the undesired states can verify the system safety behaviour.

2.3 Prime Implicants:

The DFM system model represents both the hardware and software conditions. All the system components in a DFM model have pre-defined conditions. Similarly the software used for system control and monitoring has initial set points. The combination of both the system and the software conditions produce a set of prime implicants. Based on the system knowledge the user can define the system top events. The system top events are normally critical conditions that may lead to system failure. Prime implicants are output of deductive analysis. The shortest way from fault to an initiating event in a DFM model is called a Minimal Cut Set (MCS) which eventually leads to a top event that can cause system failure. A qualitative analysis of DFM model with unknown event and failure probabilities gives us the MCS. MCS is said to be in disjunctive normal form if either an MCS `A` is in a fundamental conjunction (AND) or in a disjunction (OR) of two or more fundamental conjunctions. As DFM model deals with a set of large size of variable functions, and we get a product form that cannot be reduced by further merging, is called prime implicant.
A DFM model analysis produces a set of prime implicants, and a complete base that contains all of the system prime implicants generates a top event, which is the final state of the DFM model. A DFM model comprises variable and condition nodes; causality and condition edges; and transfer and transition boxes with decision tables.

2.4 Process Variable Nodes:

These nodes represent the physical variables and corresponding control system states. Digital control system processes can be represented by software or hardware node. The node can be discrete or continuous with multiple states that can be generated according to the particular variable. It can be failure or working, true or false, normal or exception, low or high, or multiple conditions.

2.5 Causality Edges:

The cause and effect relationship between different variables within the DFM model is defined through causality edges. The edges functional relationship is developed by connecting to a transition box, or to a transfer box with the process variable nodes.

2.6 Transfer Boxes and Associated Decision Tables:

The transfer box can represent the outcome of two or more process variables. The system knowledge can generate a logical relation between the anticipated outputs. The possible combinations that can truly reflect the system behavior can develop the decision table. The table is based on logical operators and the number of inputs associated with it. The Pseudo code forms a logical link between various possible system states and predicts system behavior in normal and abnormal conditions. The decision table entries are crucial for the system model, as improper selection can result in erroneous outcomes. A judicial combination results in stable system model.

2.7 Condition Edge:

These are used to represent the control logic mapping input variable to output variable states. The conditional connection sets possible logical conditions for the associated transfer box. The set of inputs together with specific conditions associated with the transfer box can produce specific outputs.

2.8 Condition Nodes:

The nodes represent the physical or software parameters. The conditions can be failure of component, process changes or software switching mode. The conditions are discrete variable with finite possible outputs.

2.9 Logic Operators:

Dynamic flow graph methodology generates logical decision tables which are combination of Boolean operation of NOT (negation), AND (conjunction), OR (disjunction), XNOR (equivalence), NAND (NOT AND), XNOR (bi-conditional) and XOR (implication).

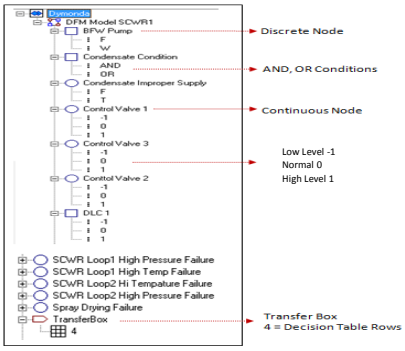

Fig. 1 DFM Model Component of CANDU SCWR Hydrogen Co-generation Model

Fig. 1 shows the DFM model components of the proposed co-generation model.

3. The Network Delay Model:

The network delay model we are assuming for the co-generation model is shown in Fig. 2.

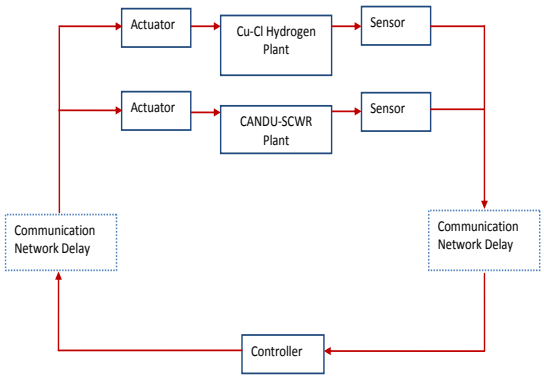

Fig. 2 Co-Generation Plant Delay Model

The DFM Network delay model nomenclature is given as follows:

HCD: Hardware Computation Delay
HRD: Hardware Response Delay
DS: Device Status
HD: Hardware Delay
ALD: Application Layer Delay
MLD: Mac Layer Delay
PLD: Physical Layer Delay
OLD: OSI Layer Delay
SS: Software Status
PRE: Pre-Processing Delay
POST: Post-Processing Delay
DR: Data Rate
DS: Data Size
PF: Protocol Frame
FD: Frame Delay
CMD: Communication Medium Delay
RTD: Retransmission Delay
BD: Back off Delay
QD: Queuing Delay
NS: Network Status
TD: Transmission Delay
CD: Communication Delay
DEL: Total Delay

The control network of the co-generation model has delays between sensor and controller and between controller and actuator. We have formed six levels of controls shown in Fig. 3.

Level 1: The delays can be broadly categorized into as hardware and software delays. This level describes the hardware component computation delay and the response delay that is when a component receives the input it computes the values by a pre-programmed logic and then takes an action. The time delay between the signal received and computational result is computation delay and the time elapsed between the computation and actual response is the response delay.

Level 3: This level describes the same delays for the receiver node.

Level 2: The Pre-processing delays are normally calculated at the sender end. For communication network between sender and receiver, three OSI model layers are involved. The software is at the application layer and the delays due to software are described as (software) application layer delays. The second OSI layer involved in communication delays is OSI MAC layer. It is part of the data link layer and it forms an interface between the software which is the application layer and the hardware which is the physical layer. . The delays at the physical layer depend on the hardware such as cables, nodes, junctions, repeaters, amplifiers and many other types of equipment. The physical layer converts digital data to analog signals and adds the necessary headers and senses the carrier medium before transmission.

Level 4: This level describes the same type of processing delays for the receiver end.

Level 5: The transmission delay depends on the data rate of the transmission from sender to receiver, the data size or payload that needs to be transmitted between sender and receiver, and finally on the protocol frame format. For example many protocols involve retransmission techniques, error checking headers, acknowledgement, encoding bits, and various other information therefore frame size is a major component in the frame transmission delays.

Level 6: Different medium of transmission have different delays. The medium can be a wired or wireless. The medium delays depend on the distance of transmission, adjacent interference and the given transmission frequency per unit time. For multiple frames competing for a single medium a buffered queue is used to minimize packet drop rate and to optimize the packet transmission. Number of packets arriving at the buffer, number of packets leaving the buffer, and the queue size can play an important role in the buffer delays. There are several queuing techniques that are commonly used in communication channel to cope with congestion. In wireless communication the chances of collision or packet drop are higher therefore a retransmission algorithm is employed by the protocol selected and these retransmissions cause delays. These three delays combined with the frame transmission delay are the actual transmission delays.

For hardware delay, Ds is the device status.Hardware delay control involves device status, sleep, idle and working. The hardwired devices compute the commands and generate response. These delays are also called interrupt latency that involves system main controller BIOS host, dependent slave devices and the bridge that is hardwired interface between interrupt requests (IRQs). IRQs are tri-state mechanism with sample, recovery and turn around phase all depending on the CPU clock cycle.

The CPU clock cycle delay for one clock cycle is assumed as 170 nano seconds for our system. Vendors have specified their system clock cycles, and we are assuming the PCI Bus delays for a 25 MHz CPU clock cycle as 3.84 micro seconds and for 33 MHz CPU clock cycle as 2.88 micro seconds.

OSI layer delay which comprises has node condition of SS, which is the software status with busy, idle and wait states.

Transmission Delay (TD) comprises NC, which is the network condition; and either it is contention free or contention full.

In our dymonda [7] DFM model, we have modelled the communication system delays. Application layer is the upper most layer on OSI model, and we have defined permissible and exception level of delays on the OSI application layer. In concurrent programming

various applications are working in parallel and are assigned thread numbers from 1 to n. These threads can be prioritized or non-prioritized, synchronized or a synchronized.

Every thread is timed in and timed out by thread server application which throws an exception on thread time out and takes in another application in a round robin mechanism. Polling is done through an echo server to all of the client applications.

Parallelization can also be achieved at physical layer through multiple processors, by adding new hardware interfaced through Message Passing Interface (MPI). Parallelization at hardware level is expensive and requires additional interfaces and programming skills to compute. Recent advancements in Very Large Scale Integration (VLSI) micro chip technologies have introduced personal computers that can perform exponential computations. Flynn's taxonomy of parallel programming is a classification method for instructions and data processing. The taxonomy classifies four distinct features for instructions and data. Flynn's taxonomy was initially developed for hardware interfaces but it is recently adapted for single processor multi-data and instructions architecture. Various GUI tools like java and dot net technologies have built in libraries for threads programming. In our model we are assuming software parallelized model which can run multiple context switched programs.

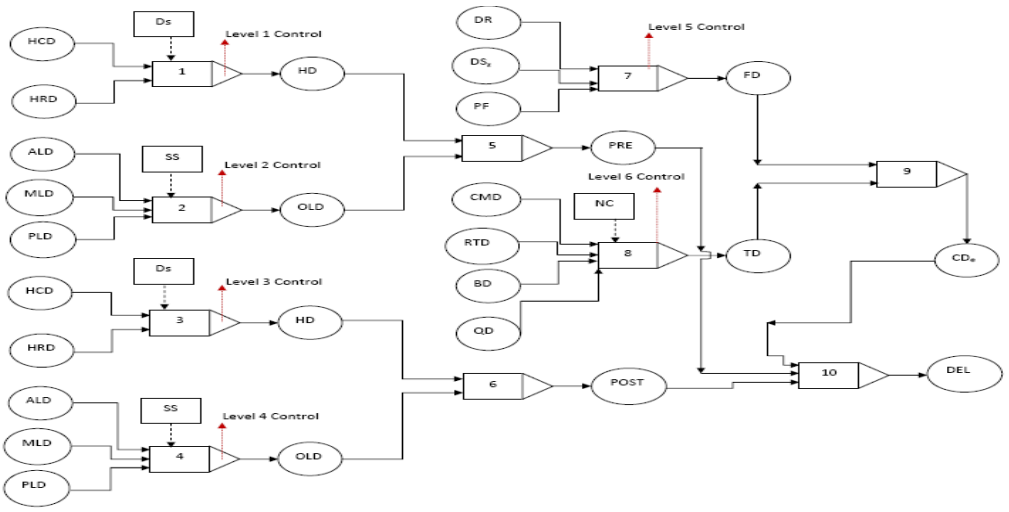

Fig. 3 Co-Generation Plant Delay Model

We have analyzed the entire model with multiple conditions that can generate more than hundred combinations. The discrete values produce top events that are critical for the network system.

For software delay we have defined the conditions as threshold minimum, threshold maximum and normal. The communication between software and hardware takes place through media access layer. The Mac layer delay is commonly associated with the communication between the concurrent programs and protocols.

Real time control systems must respond within specific time limit and any high order delay can throw an exception for a timed out thread. Software and hardware interface for controller, sensor or actuator can be interfaced through network interface card. The command signals coming in from the sensor, controller or actuator and the signals going out from the software are time in and time out entities respectively. Software delays are associated with application, Mac and physical layer.

We have programmed client server environment for echo-server connection application for echoing server ports to the clients. The port we assumed for broadcast is 69. We have designed both the client and server application as multi-thread client and server echo application, so that

multiple client programs can run concurrently using port 69. The advantages of thread based communication between server and client is that threads share same memory address space within the program, context switching and communication between threads is inexpensive and CPU cycles are efficiently utilized by the thread programming. Fig. 4a and b shows the coding in java on JCreator LE tool [8]. The results show that server echoes the port number ready for client connection. The client echoes their identification to the server and a multi-threaded connection is established through the server virtual port and client virtual IP address.

Fig. 4 and 5 shows the output of server and client echo-server applications, The Server was run in windows 7 and client was run in VM-ware player with Linux (Ubuntu) platform.

4. The Echo Server Code:

```
import java.io.*;
import java.net.*;
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
public class EchoServer extends Thread {
static final String APP_NAME = "EchoServer";
static final int PORT = 6991;
static ServerSocket serverSocket;
static int port = PORT;
Socket clientSocket;
BufferedReader is = null;
BufferedWriter os = null;
public EchoServer(Socket cs) {
clientSocket = cs;}
public static void main(String args[]) {
if (usageOnly(args)) {
System.exit(0);}
initialize(args);
printMsg("EchoServer running on port: " + port +
"...Ready to accept connections...");
while (true) {
try {
Socket clientSocket = serverSocket.accept();
EchoServer es = new EchoServer(clientSocket);
es.start();
} catch (IOException e) {
printMsg("Cannot accept client connection.");} } }
public void run() {
processClientRequest();}
private static boolean usageOnly(String args[]) {
if (args.length > 1 || (args.length == 1
&& (args[0].equalsIgnoreCase("-usage")
|| args[0].equalsIgnoreCase("-help")
|| args[0].equalsIgnoreCase("-h")))) {
System.out.println("Usage: java " + APP_NAME + "
[<port>]");
System.out.println(" The default port is " + port + ".");
return true;} else { return false;} }
private static void initialize(String args[]) {
processCommandLine(args);
try {serverSocket = new ServerSocket(port); } catch
(IOException e) {
printMsg("Cannot create server socket " + "on port: " +
port + ". Exiting...");
 System.exit(0);}}
private void processClientRequest() {
try {os = new BufferedWriter(
new
OutputStreamWriter(clientSocket.getOutputStream()));
is = new BufferedReader(
new InputStreamReader(clientSocket.getInputStream()));}
catch (IOException e) {
printMsg("Cannot handle client connection.");
cleanup();
return;}
try {String input = is.readLine();
if (input != null) {
input = APP_NAME + ": " + input + " " + getDateTime();
os.write(input, 0, input.length());
os.flush();}
} catch (IOException e) {
printMsg("I/O error while processing client's print file.");}
cleanup();}
private void cleanup() {
try {if (is != null) {
is.close();}
if (os != null) {
os.close();}
if (clientSocket != null) {
clientSocket.close();}
} catch (IOException e) {
printMsg("I/O error while closing connections.");}}
private static void processCommandLine(String args[]) {if
(args.length != 1) {return;}
port = Integer.parseInt(args[0]);
if (port < 1 || port > 6881) {port = PORT;
printMsg("Using port " + port + " instead.");}}
private static void printMsg(String msg) {
System.out.println(APP_NAME + ": " + msg);}
private String getDateTime() {DateFormat dateFormat =
new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
Date date = new Date();
return dateFormat.format(date);}}
```

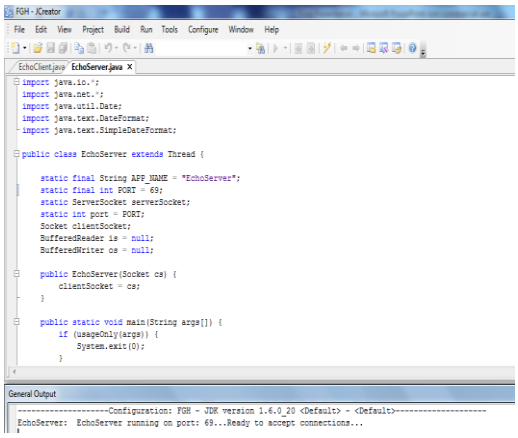

Fig. 4 Echo Server Output

5. The Echo Client Code:

```
import java.net.*;
import java.io.*;
public class EchoClient
{
public static void main(String[] args) throws IOException
{
if (args.length < 2) {
System.err.println("Usage: java EchoClient
<99.253.76.32> <255.255.255.0>");
System.exit(0);}
BufferedReader in = null;
PrintWriter out = null;
BufferedReader fromUser = null;
Socket sock = null;
try {
sock = new Socket(args[0], Integer.parseInt(args[1]));
// set up the necessary communication channels
in = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
fromUser = new BufferedReader(new
InputStreamReader(System.in));
out = new PrintWriter(sock.getOutputStream(),true);
while (true) {
String line = fromUser.readLine();
if (line == null || line.equals("bye"))
{
out.println("bye");
break;
}
out.println(line);
System.out.println(in.readLine());
}
}
catch (IOException ioe) {
System.err.println(ioe);
}
finally {
if (in != null)
in.close();
if (fromUser != null)
fromUser.close();
if (out != null)
out.close();
if (sock != null)
sock.close();}}}
```

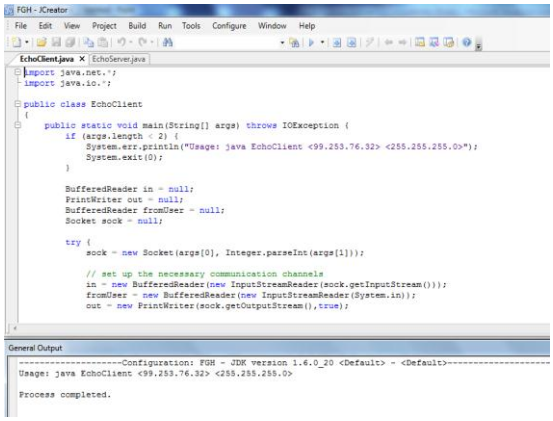

Fig. 5 Echo Client Output

In parallel programmed client/server, multithreaded socket server application accepts multiple client connections and creates thread to service every new client. Socket server identifies each client connection with a unique ID through its IP address. For any residual thread, public void flush() flushes the stream. This is done by writing any buffered output bytes to the underlying output stream and then flushing that stream. The public interface flushable is a destination of the data that can be flushed. Socket server on receiving quit command quits residual threads. Normally running both server and client on the same system, gets 0 second because the communication is less than 1 ms. For real time system we can measure the delay by System.nanoTime() command to measure delay in nano seconds.

6. The Process Time Output:

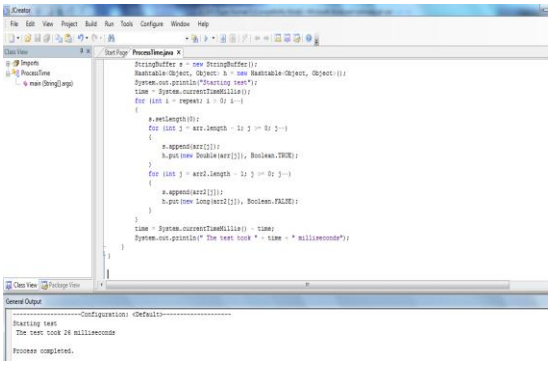

Fig. 6 Accumulated Delays for Latency and Context Switching

Our test for latency and context switching on VMware player generates a delay of 26 milliseconds. In all our computations and programs we do not consider the system idle state processes. System in idle state is still doing some processing and delays can be computed for system idle state when no application is running and no program is computed. It can be understood by the example that a node while not transmitting can be sensing the medium and receiving beacons and broadcast traffic for the network. We assume zero delays for system idle state.

7. Concurrent Programming Logic

The logic behind the concurrent programming model is based on Flynn's taxonomy classification. The single program single data model is omitted as it does not satisfy the basic parallelized program model. The three logics for sensor, controller and actuator model are:

- SPMD = Single Program Multiple Data
- MPMD = Multiple Program Multiple Data
- MPSD = Multiple Program Single Data

The logics are defined below:

- If Logic = SPMD
  Initialize Sensor1 Data, Sensor 2 Data
  Initialize Actuator1 Data, Actuator 2 Data
  Start thread
  Program (try, catch)
  Output result

- If Logic = MPSD
  Initialize Sensor Data
  Start thread
  Program1 (try, catch)
  Program2 (try, catch)
  Program3 (try, catch)
  Program4 (try, catch)
  Output result

- If Logic = MPMD
  Initialize Sensor1 Data, Sensor 2 Data
  Initialize Actuator1 Data, Actuator 2 Data
  Start thread
  Program1 (try, catch)
  Program2 (try, catch)
  Program3 (try, catch)
  Program4 (try, catch)
  Output result

We have implemented the MPMD logic and plot the delays of MPMD for three programs i.e. program 1 with 10 threads, program 2 with 20 threads and program 3 with 40 threads. The delays are shown in micro seconds for a number of iterations. The simulation results show that up to 10000 iterations the delays are much higher for the program 2 and 3 as compared to program 1 delays. For further iterations, the delays of program 1 and program 2 are more or less similar. The output for multithread control network shows that delays of lower order programs (10 to 20 threads) are much lower than delays of higher order programs (40 threads).
For a multi-thread delay we have used matlab code to show the elapsed time between the threads, the total time taken by the threads iterations and time taken by the parent and child threads as shown in Fig. 8. The graphical representation in Fig. 9 shows the parent thread which is the main program delay with two child threads delay. The program threads occupying the CPU cycle release the resources and the CPU cycle time seems to go in negative, it is called the wrap around time to reset systems resources for the next run. The time in nano seconds appear as zero seconds.

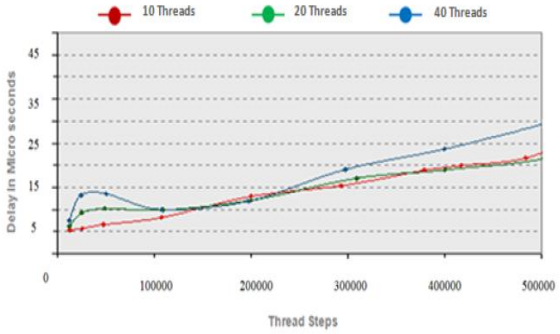

Fig. 7 MPMD Logic output for a Control Network

The resources are assigned to the program main thread that is the parent thread, with a delay of e seconds. The elapsed time is for time switching between threads from parent to child or from child to child. The resources allocation and deallocation creates a negative void for CPU cycles which is actually the wrap around time for the control switch

between threads. Its actual value is zero yet it seems negative during simulation.

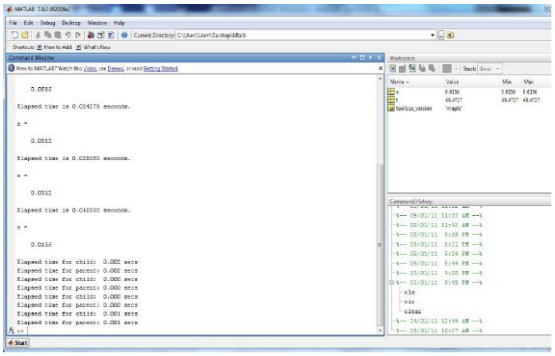

Fig. 8 MatLab Delay for Parent Child threads and Elapsed time

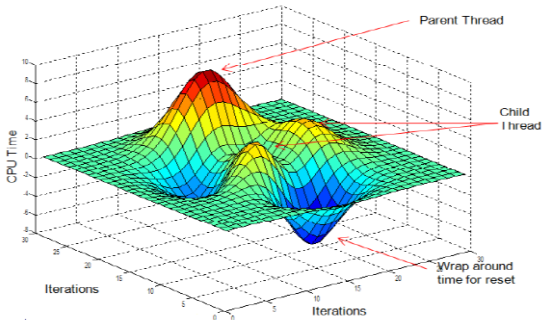

Fig. 9 MatLab Delay for Parent Child threads and Wrap around time

8. Conclusion

We have generated a communication delay model using the dynamic flow graph methodology. We have considered the delays of the co-generation hydrogen production communication model. An echo server application is programmed that calls the clients connection and establish a parallelized connection between the server and the client. Real time Network control system reliability increases with decrease in delays and we have analyzed that concurrent programmed thread applications have higher throughput and lesser delays. These delays are associated with control levels 1 to 4 of the DFM model. In future research we would analyze the DFM control levels 5 and 6 for queuing and scheduling delays.

Acknowledgments

The research is funded by AECL, Canada research funding for the clean energy research labs at UOIT, Ontario, Canada.

References

[1] Duffey R. B., I. Pioro and X. Zhou, "Supercritical Water-Cooled Nuclear Reactors (SCWRs): Current and Future Concepts - Steam-Cycle Options", Proc. ICONE-16, Orlando, FL, USA, May 11-15, Paper #48869, 2008, 9 pages.

[2] www.weathernetwork.com, Accessed on January 2011.

[3] Garrett, S., S. Guarro and G. Apostolakis: "The Dynamic Flowgraph Methodology for Assessing the Dependability of Software Systems," *IEEE Transactions on Systems, Man and Cybernetics* 25, pp.824-840, 1995.

[4] Yau, M., S. Guarro and G. Apostolakis: "Demonstration of the Dynamic Flowgraph Methodology using the Titan II Space Launch Vehicle Digital Flight Control Software," *Reliability Engineering and System Safety* 49, pp.335-353, 1995.

[5] Garrett, S., M. Yau, S. Guarro and G. Apostolakis: "Assessing the Dependability of Embedded Software Systems Using the Dynamic Flowgraph Methodology," in *Dependable Computing and Fault-Tolerant Systems Vol. 9*, F. Cristian, G. Le Lann, T. Lunt (eds.), Springer-Verlag Wien, New York, 1995.

[6] Houtermans M., G. Apostolakis, A. Brombacher and D. Karydas. "Programmable electronic system design & verification utilizing DFM." In *SAFECOMP 2000* (F. Koornneef and M. van der Meulen, eds.), Lecture Notes in Computer Science 1943 (2000), 275-285.

[7] www. Asecanic.com, accessed on January 2011.

[8] www.jcreator.org, accessed on May 2011.

Fayyaz Ahmed is currently Research Assistant at University of Ontario Institute of Technology, Oshawa, Ontario, Canada. Previously he has been teaching assistant at McMaster University, Hamilton, Ontario, Canada for two years. He has also taught at Institute of Business Administration Karachi and Hamdard University Karachi, Sind, Pakistan. Before moving to Canada, he has worked as Nuclear Engineer in Pakistan Atomic Energy Commission for twenty years. He completed his BA Computer Applications from Allama Iqbal Open University, Islamabad, Pakistan in 1996. He graduated in Electrical Engineering from University of Engineering and Technology, Lahore, Pakistan in 1998. He completed his MSc in Computer Sciences, from Al-Khair University, AJK, Pakistan, MS in Information Technology, from Hamdard University, Karachi, Pakistan and MSc in Computational Engineering from McMaster University, Hamilton, Canada in 2001, 2004 and 2009 respectively. Currently he is a PhD student at UOIT, Oshawa, ON, Canada. He has worked for Sympatico Bell and Alcatel Lucent Canada and is certified with Microsoft, Oracle and IBM programming and networking technologies since 2000.