**International Islamic University Chittagong**

***Department of Computer Science and Engineering***

***Project Report Submission***

**Course Code:** CSE-3634
**Course Name:** Computer Networks Laboratory

# Implementation of Cyclic Redundancy Check (CRC)

**Submitted By:**
**Group 1: Divergent**

**C201249** Lamisa Mashiat
**C201266** Sohana Tasneem
**C201270** Sadia Haque Chowdhury
**C201284** Mossammat Marowa Khanam
**C193252** Tahnia Binte Shafi

**Submitted To:**
Abdullahil Kafi
Assistant Professor
Department of CSE, IIUC

***Date of Submission**: September 18, 2023*

## Abstract:

In the realm of data communication and networking, ensuring the integrity of transmitted data is paramount. This project explores the implementation of Cyclic Redundancy Check (CRC) at the data link layer using OMNeT++, with the aim of enhancing data integrity in communication protocols. By calculating CRC codes for outgoing messages and enabling error detection at the receiving end, the study employs a meticulously chosen polynomial and overcomes implementation challenges. The objective is to enhance data reliability by calculating CRC codes for transmitted messages and detecting errors at the destination. Through extensive simulations and practical experiments, the report evaluates the effectiveness of CRC in error detection, highlighting its significance in data communication protocols. This project underscores the pivotal role of CRC, offering insights into its real-world applicability within the context of OMNeT++ simulations. Through this project, the significance of CRC in data communication is underscored, and its practical implementation within a simulation environment is explored.

# TABLE OF CONTENTS

# CHAPTER 1
# INTRODUCTION

## 1.1 Introduction:

In the dynamic landscape of data communication, the robustness and reliability of transmitted data hold pivotal importance. This project presents a comprehensive exploration of the Cyclic Redundancy Check (CRC) technique, deployed at the data link layer, realized through the OMNeT++ simulation framework. The primary objective is to enhance data integrity by rigorously calculating CRC codes for outgoing messages and providing a mechanism for error detection at the receiving end.

The project methodically unfolds with an exposition of the chosen methodology, including a detailed description of the tools and software employed in the implementation process. Notably, the project encompasses the intricate computation of CRC codes, featuring a meticulously selected polynomial. Throughout the implementation phase, a series of challenges were encountered, each surmounted with innovative problem-solving strategies, which are thoroughly elucidated in this report.

Subsequently, the project embarks on a compelling presentation of results derived from extensive simulations and practical experiments. These results shed light on the effectiveness of the CRC implementation in the detection of transmission errors, providing empirical insights into its real-world applicability. The ensuing discussion section dissects and interprets the implications of the findings, offering a critical analysis of the CRC implementation's efficacy and reliability. Furthermore, a comparative assessment against anticipated outcomes furnishes a holistic perspective on the project's achievements and potential areas for further refinement.

In summation, this project not only underscores the pivotal role of CRC in ensuring data accuracy in communication protocols but also exemplifies its seamless integration into the OMNeT++ environment. The culmination of this research endeavor accentuates the significance of CRC in the realm of data communication and beckons further exploration into its extended applications and optimizations for future network protocols.

## 1.2 Objectives:

The objective of the project, as described in the provided text, is to implement and simulate the Cyclic Redundancy Check (CRC) mechanism in the OMNeT++ network simulation framework. This project aims to demonstrate the use of CRC as an error detection technique in data communication and networking. The specific objectives of the project can include:

⇨ **CRC Implementation:** This objective involves creating a functional implementation of the Cyclic Redundancy Check (CRC) mechanism within the OMNeT++ network simulation framework. It may include designing algorithms and data structures to perform CRC calculations.

⇨ **Error Detection:** The project aims to demonstrate the effectiveness of CRC as an error detection technique. It involves simulating scenarios where data packets are transmitted through a network, and CRC is used to detect errors in these packets.

⇨ **Network Simulation:** To achieve the above objectives, a network simulation environment must be created within OMNeT++. This includes setting up network nodes, data transmission, and CRC calculation processes.

⇨ **Performance Evaluation:** This objective involves assessing how well CRC performs in terms of error detection. It may include measuring the accuracy of error detection and evaluating the computational overhead introduced by CRC.

⇨ **Comparison:** The project may compare CRC with other error detection methods, such as parity checks. The goal is to understand the relative advantages and disadvantages of using CRC in different network conditions.

⇨ **Documentation:** Comprehensive documentation is essential to explain the project's design, implementation details, and simulation results. Documentation helps users and researchers understand the project and its findings.

⇨ **Educational Purpose:** This project can serve as an educational resource for students, researchers, and network professionals. It provides hands-on experience with error detection techniques and their practical application.

⇨ **Extendibility:** Designing the project with extendibility in mind means making it easy to modify and adapt for different purposes. For example, users might want to test various CRC polynomials or simulate different network configurations.

⇨ **Validation:** To ensure the accuracy and reliability of the simulation results, the project should validate these results against theoretical expectations and known properties of CRC. This step confirms that the simulation faithfully represents CRC's behavior.

Overall, the project's objectives align with creating a valuable tool for exploring error detection in network communication. It combines theory with practical implementation and simulation, making it suitable for both learning and experimentation.

# CHAPTER 2
# REQUIREMENTS OF THE SYSTEM

## 2.1 Language:

❖ C++

## 2.2 Software Specifications:

❖ OMNeT++ IDE

## 2.3 Hardware Specifications:

| | |
|---|---|
| **Processor** | AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz; Installed RAM 8.00 GB |
| **Edition** | Windows 10 Home Single Language |
| **System Type** | 64-bit operating system, x64-based processor |

## 2.4 Functional Requirements:

❖ **Module Creation:** Develop compound modules representing network endpoints (Sender and Receiver) and simple modules representing various layers of the TCP/IP stack (Physical, Data Link, Network, Transport, Application).

❖ **Layered Communication:** Implement communication interfaces (gates) between adjacent layers to facilitate message passing while focusing on CRC error detection and correction.

❖ **Error Simulation:** Simulate communication between Sender and Receiver with a focus on introducing CRC errors into the data transmission process.

## 2.5 Non-Functional Requirements:

- ❖ **Modularity:** The simulator should exhibit modularity to efficiently model communication with CRC error detection, accommodating various error detection algorithms and strategies.
- ❖ **Realism:** Reflect real-world network behavior with a specific emphasis on accurate timing, message passing, and the realistic modeling of CRC error detection and correction processes.
- ❖ **Performance:** Measure and report execution time and resource usage to assess the efficiency of your CRC error detection approach within the simulation.
- ❖ **Documentation:** Provide comprehensive code comments and user documentation, particularly detailing the implementation of CRC error detection mechanisms and their integration into the OMNeT++ simulation.
- ❖ **Portability:** Design your simulation for compatibility across platforms and different versions of OMNeT++, ensuring that your CRC error detection approach can be easily replicated and tested on various setups.

# CHAPTER 3
# SYSTEM DESIGN

## 3.1 Layer Architecture of the System:

In OMNeT++, simple or compound modules represent layers with Input (I) and Output (O) gate pairs. These modules are used to define the behavior and interactions of components within a network simulation. Simple modules typically encapsulate basic functionality, while compound modules can combine several simple modules to form more complex network components. Input and Output gates facilitate communication and data exchange between these modules.

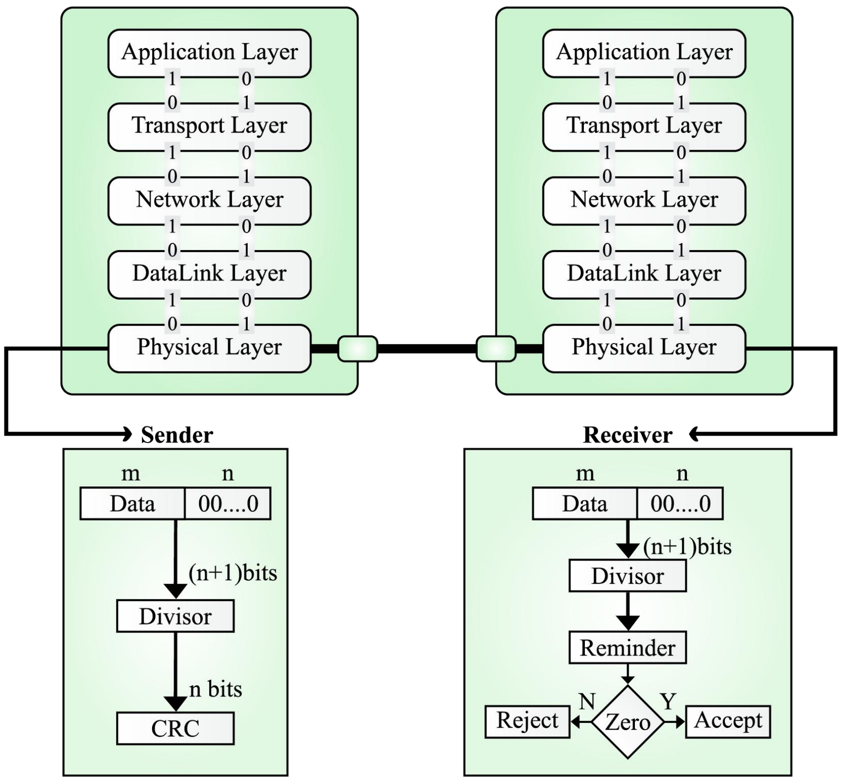
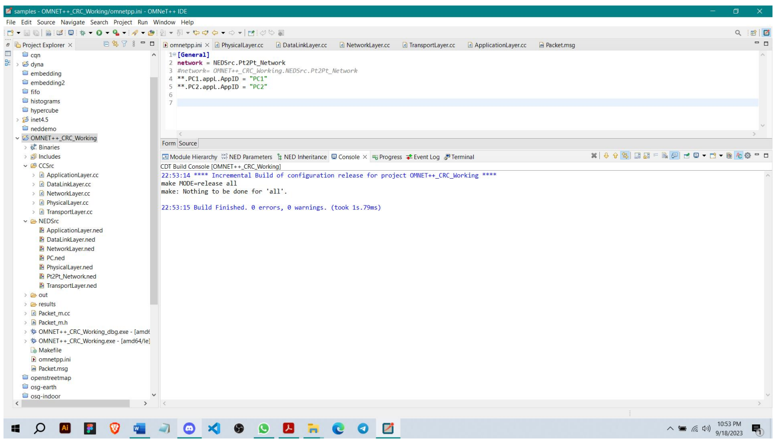


**Fig-1:** Simple or compound modules representing layers with Input (I)/Output (O) gate pairs in OMNeT++

# CHAPTER 4
# INTERFACES

## 4.1 Simulation and Result

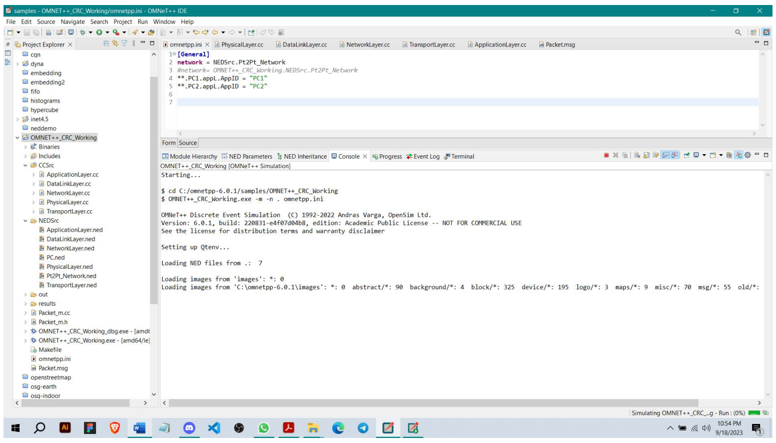
In this section, we delve into the heart of our project, where theory meets practical implementation. Utilizing the OMNeT++ simulation framework, we meticulously executed the CRC implementation on the data link layer. This involved calculating CRC codes for a range of transmitted messages and subjecting them to simulated network conditions to assess their accuracy. Through a series of controlled experiments and simulations, we examined the capability of the CRC mechanism to detect errors effectively. The results obtained provide empirical evidence of CRC's prowess in enhancing data integrity within a networked environment. This section presents a detailed account of our simulation setup, the parameters used, and, most importantly, the quantitative and qualitative outcomes of our experiments, shedding light on the real-world applicability and benefits of CRC in error detection.



```
[General]
network = NEDSrc.Pt2Pt_Network
#network= OMNET++_CRC_Working.NEDSrc.Pt2Pt_Network
**.PC1.appL.AppID = "PC1"
**.PC2.appL.AppID = "PC2"
```

```
CDT Build Console [OMNET++_CRC_Working]
22:53:14 **** Incremental Build of configuration release for project OMNET++_CRC_Working ****
make MODE=release all
make: Nothing to be done for 'all'.

22:53:15 Build Finished. 0 errors, 0 warnings. (took 1s.79ms)
```

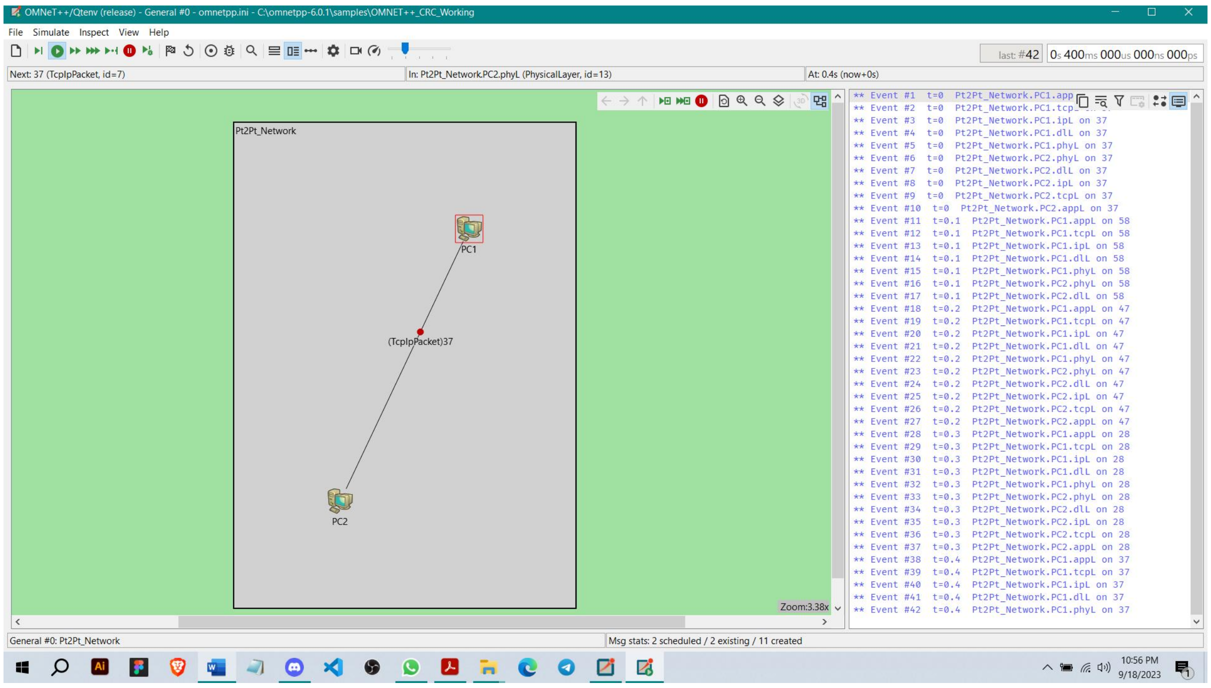
**Fig-2:** Successfully finished Building



```
[General]
network = NEDSrc.Pt2Pt_Network
#network= OMNET++_CRC_Working.NEDSrc.Pt2Pt_Network
**.PC1.appL.AppID = "PC1"
**.PC2.appL.AppID = "PC2"
```

```
OMNET++_CRC_Working [OMNeT++ Simulation]
Starting...

$ cd C:/omnetpp-6.0.1/samples/OMNET++_CRC_Working
$ OMNET++_CRC_Working.exe -m -n . omnetpp.ini

OMNeT++ Discrete Event Simulation  (C) 1992-2022 Andras Varga, OpenSim Ltd.
Version: 6.0.1, build: 220831-e4f07d04b8, edition: Academic Public License -- NOT FOR COMMERCIAL USE
See the license for distribution terms and warranty disclaimer

Setting up Qtenv...

Loading NED files from .:  7

Loading images from 'images': *: 0
Loading images from 'C:\omnetpp-6.0.1\images': *: 0  abstract/*: 90  background/*: 4  block/*: 325  device/*: 195  logo/*: 3  maps/*: 9  misc/*: 70  msg/*: 55  old/*:
```

**Fig-3:** Execution of the project


OMNeT++/Qtenv (release) - General #0 - omnetpp.ini - C:\omnetpp-6.0.1\samples\OMNET++_CRC_Working
File
Simulate
Inspect
View
Help
last: #42
0s 400ms 000us 000ns 000ps
Pt2Pt_Network
PC1
(TcpIpPacket)37
PC2
Zoom:3.38x
10:56 PM
9/18/2023

Next: 37 (TcpIpPacket, id=7)

In: Pt2Pt_Network.PC2.phyL (PhysicalLayer, id=13)

At: 0.4s (now+0s)

```
** Event #1   t=0   Pt2Pt_Network.PC1.app
** Event #2   t=0   Pt2Pt_Network.PC1.tcp
** Event #3   t=0   Pt2Pt_Network.PC1.ipL on 37
** Event #4   t=0   Pt2Pt_Network.PC1.dlL on 37
** Event #5   t=0   Pt2Pt_Network.PC1.phyL on 37
** Event #6   t=0   Pt2Pt_Network.PC2.phyL on 37
** Event #7   t=0   Pt2Pt_Network.PC2.dlL on 37
** Event #8   t=0   Pt2Pt_Network.PC2.ipL on 37
** Event #9   t=0   Pt2Pt_Network.PC2.tcpL on 37
** Event #10  t=0   Pt2Pt_Network.PC2.appL on 37
** Event #11  t=0.1  Pt2Pt_Network.PC1.appL on 58
** Event #12  t=0.1  Pt2Pt_Network.PC1.tcpL on 58
** Event #13  t=0.1  Pt2Pt_Network.PC1.ipL on 58
** Event #14  t=0.1  Pt2Pt_Network.PC1.dlL on 58
** Event #15  t=0.1  Pt2Pt_Network.PC1.phyL on 58
** Event #16  t=0.1  Pt2Pt_Network.PC2.phyL on 58
** Event #17  t=0.1  Pt2Pt_Network.PC2.dlL on 58
** Event #18  t=0.2  Pt2Pt_Network.PC1.appL on 47
** Event #19  t=0.2  Pt2Pt_Network.PC1.tcpL on 47
** Event #20  t=0.2  Pt2Pt_Network.PC1.ipL on 47
** Event #21  t=0.2  Pt2Pt_Network.PC1.dlL on 47
** Event #22  t=0.2  Pt2Pt_Network.PC1.phyL on 47
** Event #23  t=0.2  Pt2Pt_Network.PC2.phyL on 47
** Event #24  t=0.2  Pt2Pt_Network.PC2.dlL on 47
** Event #25  t=0.2  Pt2Pt_Network.PC2.ipL on 47
** Event #26  t=0.2  Pt2Pt_Network.PC2.tcpL on 47
** Event #27  t=0.2  Pt2Pt_Network.PC2.appL on 47
** Event #28  t=0.3  Pt2Pt_Network.PC1.appL on 28
** Event #29  t=0.3  Pt2Pt_Network.PC1.tcpL on 28
** Event #30  t=0.3  Pt2Pt_Network.PC1.ipL on 28
** Event #31  t=0.3  Pt2Pt_Network.PC1.dlL on 28
** Event #32  t=0.3  Pt2Pt_Network.PC1.phyL on 28
** Event #33  t=0.3  Pt2Pt_Network.PC2.phyL on 28
** Event #34  t=0.3  Pt2Pt_Network.PC2.dlL on 28
** Event #35  t=0.3  Pt2Pt_Network.PC2.ipL on 28
** Event #36  t=0.3  Pt2Pt_Network.PC2.tcpL on 28
** Event #37  t=0.3  Pt2Pt_Network.PC2.appL on 28
** Event #38  t=0.4  Pt2Pt_Network.PC1.appL on 37
** Event #39  t=0.4  Pt2Pt_Network.PC1.tcpL on 37
** Event #40  t=0.4  Pt2Pt_Network.PC1.ipL on 37
** Event #41  t=0.4  Pt2Pt_Network.PC1.dlL on 37
** Event #42  t=0.4  Pt2Pt_Network.PC1.phyL on 37
```

General #0: Pt2Pt_Network

Msg stats: 2 scheduled / 2 existing / 11 created

**Fig-4:** Running project of OMNeT++

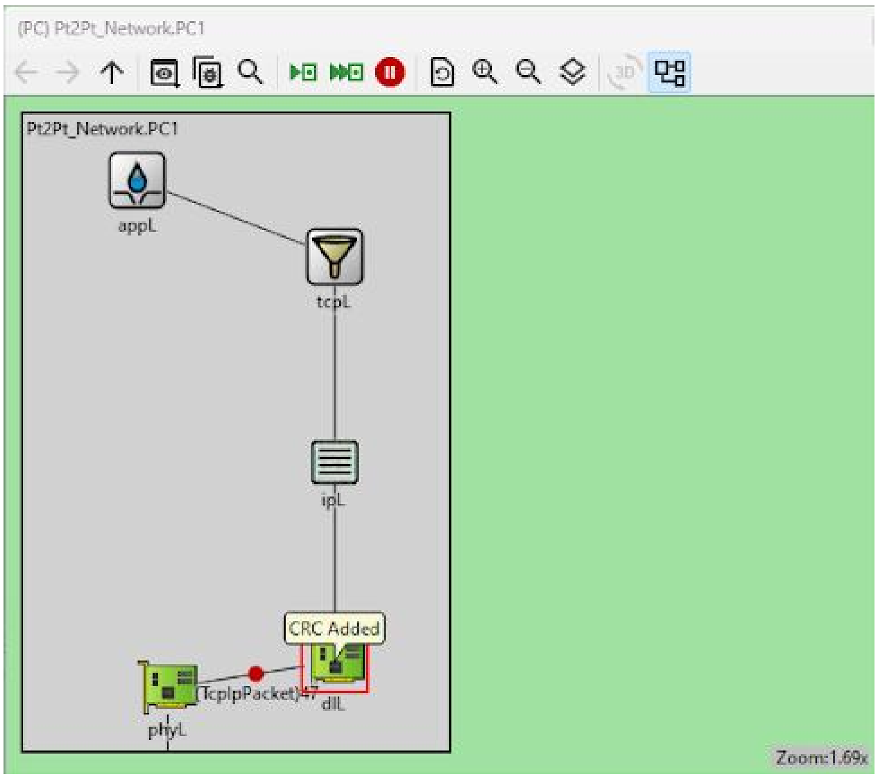
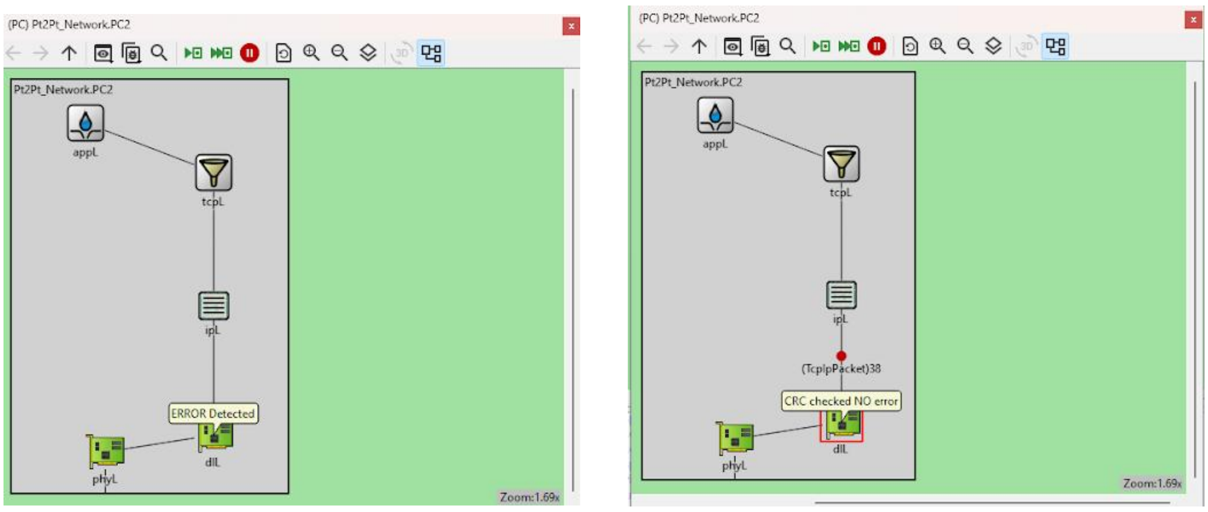
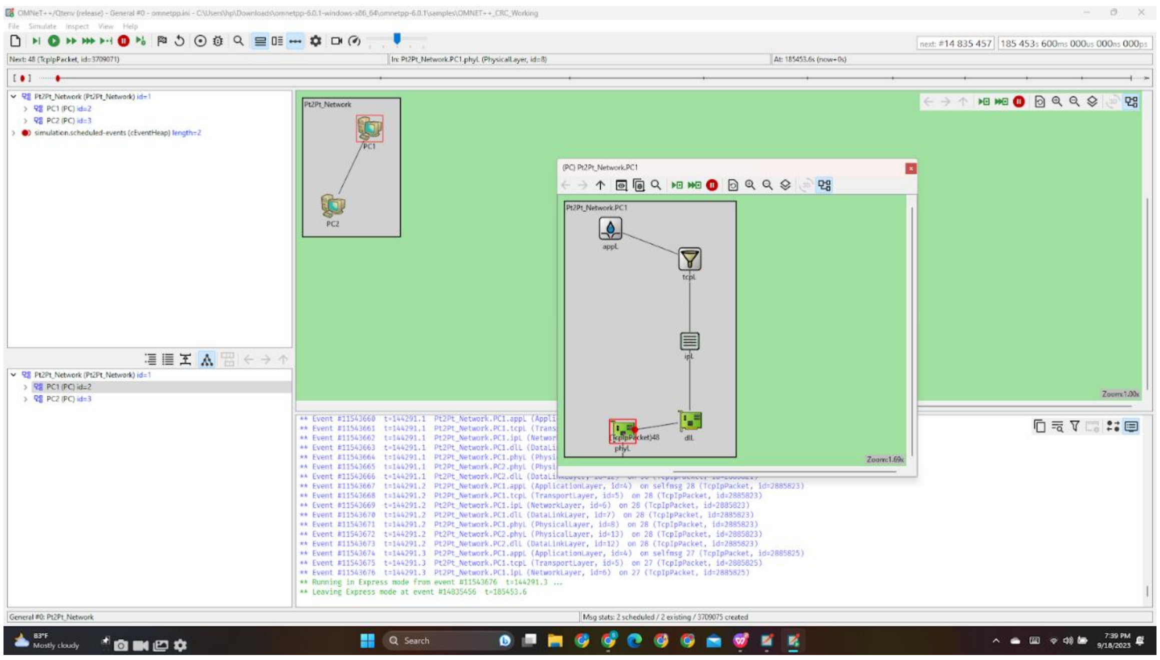


**Fig-5:** Snapshot of adding CRC



(a) Error Detected (b) No Error Detected

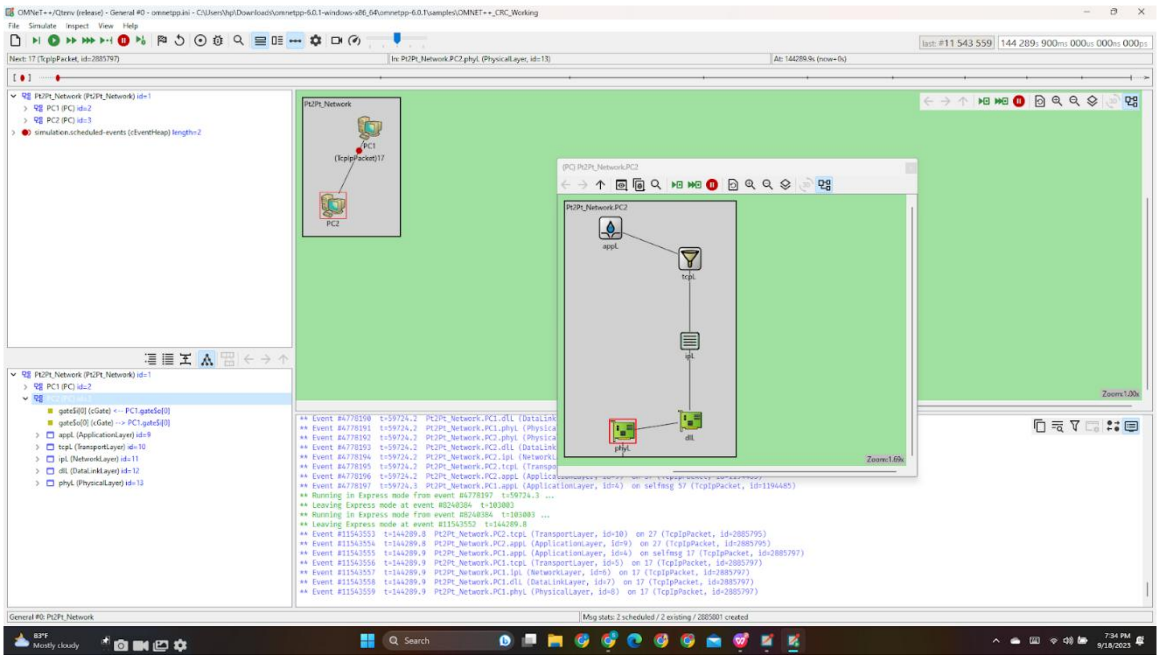
**Fig. 6:** (a,b) Error detection Demonstration



```
** Event #11543660  t=144291.1  Pt2Pt_Network.PC1.appL (Appli
** Event #11543661  t=144291.1  Pt2Pt_Network.PC1.tcpL (Trans
** Event #11543662  t=144291.1  Pt2Pt_Network.PC1.ipL (Networ
** Event #11543663  t=144291.1  Pt2Pt_Network.PC1.dlL (DataLi
** Event #11543664  t=144291.1  Pt2Pt_Network.PC1.phyL (Physi
** Event #11543665  t=144291.1  Pt2Pt_Network.PC2.phyL (Physi
** Event #11543666  t=144291.1  Pt2Pt_Network.PC2.dlL (DataLi
** Event #11543667  t=144291.2  Pt2Pt_Network.PC1.appL (ApplicationLayer, id=4)  on selfmsg 28 (TcpIpPacket, id=2885823)
** Event #11543668  t=144291.2  Pt2Pt_Network.PC1.tcpL (TransportLayer, id=5)  on 28 (TcpIpPacket, id=2885823)
** Event #11543669  t=144291.2  Pt2Pt_Network.PC1.ipL (NetworkLayer, id=6)  on 28 (TcpIpPacket, id=2885823)
** Event #11543670  t=144291.2  Pt2Pt_Network.PC1.dlL (DataLinkLayer, id=7)  on 28 (TcpIpPacket, id=2885823)
** Event #11543671  t=144291.2  Pt2Pt_Network.PC1.phyL (PhysicalLayer, id=8)  on 28 (TcpIpPacket, id=2885823)
** Event #11543672  t=144291.2  Pt2Pt_Network.PC2.phyL (PhysicalLayer, id=13)  on 28 (TcpIpPacket, id=2885823)
** Event #11543673  t=144291.2  Pt2Pt_Network.PC2.dlL (DataLinkLayer, id=12)  on 28 (TcpIpPacket, id=2885823)
** Event #11543674  t=144291.3  Pt2Pt_Network.PC1.appL (ApplicationLayer, id=4)  on selfmsg 27 (TcpIpPacket, id=2885825)
** Event #11543675  t=144291.3  Pt2Pt_Network.PC1.tcpL (TransportLayer, id=5)  on 27 (TcpIpPacket, id=2885825)
** Event #11543676  t=144291.3  Pt2Pt_Network.PC1.ipL (NetworkLayer, id=6)  on 27 (TcpIpPacket, id=2885825)
** Running in Express mode from event #11543676  t=144291.3 ...
** Leaving Express mode at event #14835456  t=185453.6
```

Msg stats: 2 scheduled / 2 existing / 3709075 created

**Fig. 7:** Pt2Pt Network PC-1 Layering



OMNeT++/Qtenv (release) - General #0 - omnetpp.ini - C:\Users\hp\Downloads\omnetpp-6.0.1-windows-x86_64\omnetpp-6.0.1\samples\OMNET++_CRC_Working

Next: 17 (TcpIpPacket, id=2885797)
In: Pt2Pt_Network.PC2.phyL (PhysicalLayer, id=13)
At: 144289.9s (now+0s)

```
Pt2Pt_Network (Pt2Pt_Network) id=1
  PC1 (PC) id=2
  PC2 (PC) id=3
simulation.scheduled-events (cEventHeap) length=2
```

```
Pt2Pt_Network (Pt2Pt_Network) id=1
  PC1 (PC) id=2
  PC2 (PC) id=3
    gate$i[0] (cGate) <-- PC1.gate$o[0]
    gate$o[0] (cGate) --> PC1.gate$i[0]
    appL (ApplicationLayer) id=9
    tcpL (TransportLayer) id=10
    ipL (NetworkLayer) id=11
    dlL (DataLinkLayer) id=12
    phyL (PhysicalLayer) id=13
```

```
** Event #4778190  t=59724.2  Pt2Pt_Network.PC1.dlL (DataLink
** Event #4778191  t=59724.2  Pt2Pt_Network.PC1.phyL (Physica
** Event #4778192  t=59724.2  Pt2Pt_Network.PC2.phyL (Physica
** Event #4778193  t=59724.2  Pt2Pt_Network.PC2.dlL (DataLink
** Event #4778194  t=59724.2  Pt2Pt_Network.PC2.ipL (NetworkL
** Event #4778195  t=59724.2  Pt2Pt_Network.PC2.tcpL (Transpo
** Event #4778196  t=59724.2  Pt2Pt_Network.PC2.appL (Applica
** Event #4778197  t=59724.3  Pt2Pt_Network.PC1.appL (ApplicationLayer, id=4)  on selfmsg 57 (TcpIpPacket, id=1194485)
** Running in Express mode from event #4778197  t=59724.3 ...
** Leaving Express mode at event #8240384  t=103003
** Running in Express mode from event #8240384  t=103003 ...
** Leaving Express mode at event #11543552  t=144289.8
** Event #11543553  t=144289.8  Pt2Pt_Network.PC2.tcpL (TransportLayer, id=10)  on 27 (TcpIpPacket, id=2885795)
** Event #11543554  t=144289.8  Pt2Pt_Network.PC2.appL (ApplicationLayer, id=9)  on 27 (TcpIpPacket, id=2885795)
** Event #11543555  t=144289.9  Pt2Pt_Network.PC1.appL (ApplicationLayer, id=4)  on selfmsg 17 (TcpIpPacket, id=2885797)
** Event #11543556  t=144289.9  Pt2Pt_Network.PC1.tcpL (TransportLayer, id=5)  on 17 (TcpIpPacket, id=2885797)
** Event #11543557  t=144289.9  Pt2Pt_Network.PC1.ipL (NetworkLayer, id=6)  on 17 (TcpIpPacket, id=2885797)
** Event #11543558  t=144289.9  Pt2Pt_Network.PC1.dlL (DataLinkLayer, id=7)  on 17 (TcpIpPacket, id=2885797)
** Event #11543559  t=144289.9  Pt2Pt_Network.PC1.phyL (PhysicalLayer, id=8)  on 17 (TcpIpPacket, id=2885797)
```

General #0: Pt2Pt_Network
Msg stats: 2 scheduled / 2 existing / 2885801 created

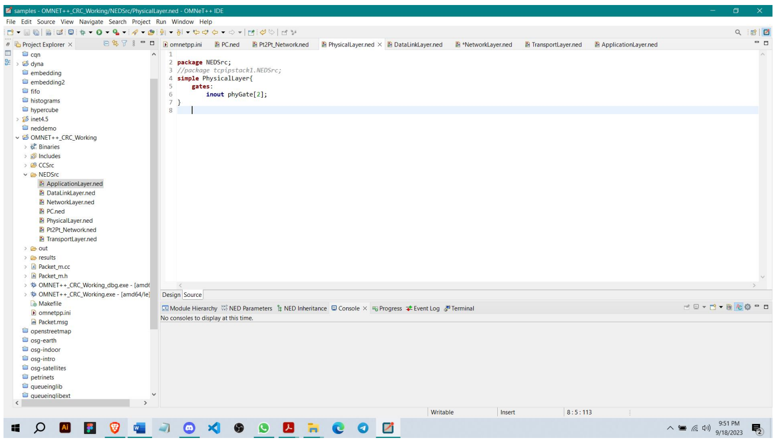
**Fig 8:** Pt2Pt Network PC-2 Layering

# CHAPTER 5
# PROGRAM FILES

## 5.1 NED files:

### 1. Physical Layer:

```
package NEDSrc;
//package tcpipstack1.NEDSrc;
simple PhysicalLayer{
    gates:
        inout phyGate[2];
}
```



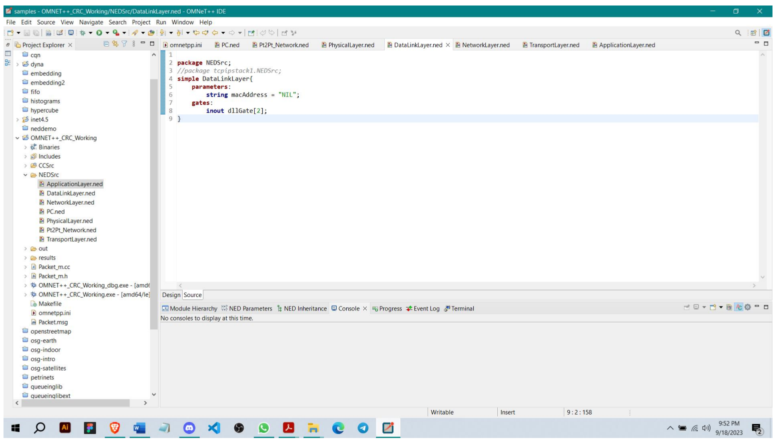
```
package NEDSrc;
//package tcpipstack1.NEDSrc;
simple PhysicalLayer{
    gates:
        inout phyGate[2];
}
```

No consoles to display at this time.

**Fig-9:** NED file of Physical Layer

### 2. Data Link Layer:

```
package NEDSrc;
//package tcpipstack1.NEDSrc;
simple DataLinkLayer{
    parameters:
        string macAddress = "NIL";
            gates:
                inout dllGate[2];
}
```



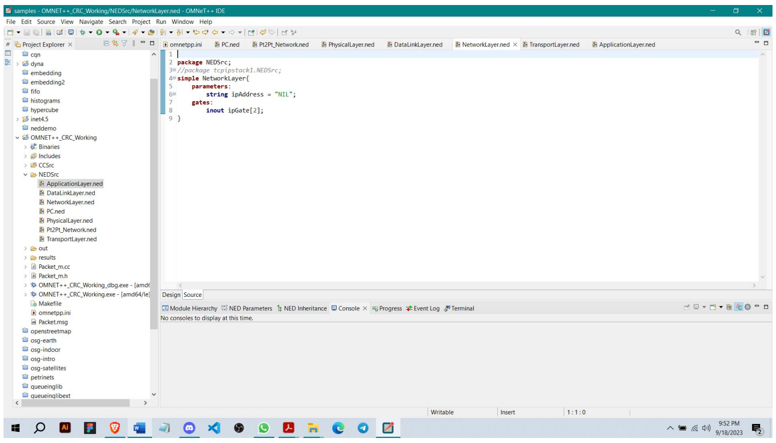
```
package NEDSrc;
//package tcpipstack1.NEDSrc;
simple DataLinkLayer{
    parameters:
        string macAddress = "NIL";
    gates:
        inout dllGate[2];
}
```

No consoles to display at this time.

**Fig-10:** NED file of DataLink Layer

## 3. Network Layer:

```
package NEDSrc;
//package tcpipstack1.NEDSrc;
simple NetworkLayer{
    parameters:
        string ipAddress = "NIL";
            gates:
                    inout ipGate[2];
}
```



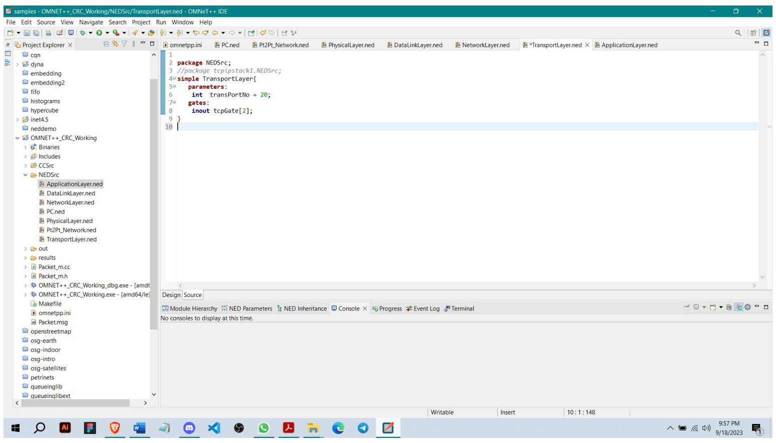
```
package NEDSrc;
//package tcpipstack1.NEDSrc;
simple NetworkLayer{
    parameters:
        string ipAddress = "NIL";
    gates:
        inout ipGate[2];
}
```

No consoles to display at this time.

**Fig-11:** NED file of Network Layer

## 4. Transport Layer:

```
package NEDSrc;
//package tcpipstack1.NEDSrc;
simple TransportLayer{
   parameters:
    int  transPortNo = 20;
   gates:
          inout tcpGate[2];
}
```



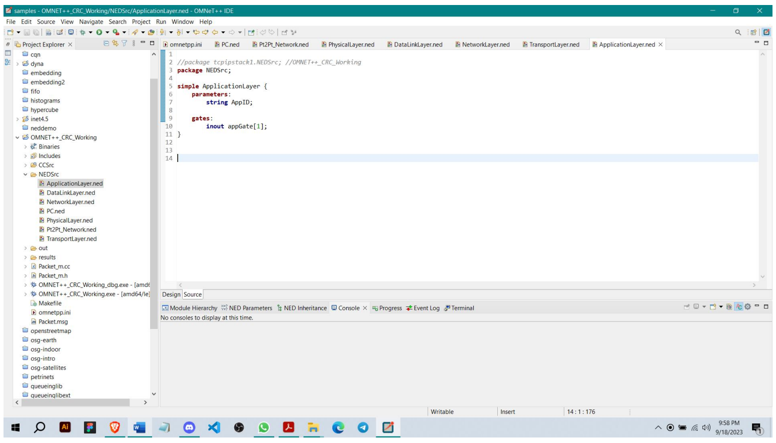
```
package NEDSrc;
//package tcpipstack1.NEDSrc;
simple TransportLayer{
    parameters:
    int  transPortNo = 20;
    gates:
    inout tcpGate[2];
}
```

No consoles to display at this time.

**Fig-12:** NED file of Transport Layer

## 5. Application Layer:

```
//package tcpipstack1.NEDSrc; //OMNET++_CRC_Working
package NEDSrc;

simple ApplicationLayer {
   parameters:
      string AppID;

   gates:
      inout appGate[1];
}
```



```

//package tcpipstack1.NEDSrc; //OMNET++_CRC_Working
package NEDSrc;

simple ApplicationLayer {
    parameters:
        string AppID;

    gates:
        inout appGate[1];
}



```

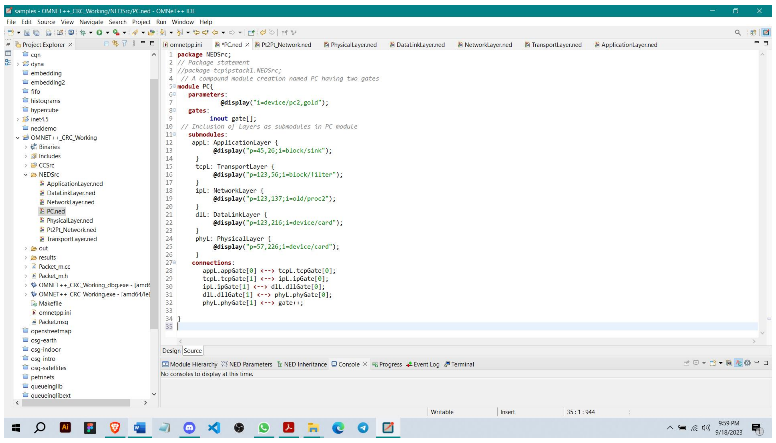
No consoles to display at this time.

**Fig-13:** NED file of Application Layer

## 6. PC Layer:

```
package NEDSrc;
// Package statement
//package tcpipstack1.NEDSrc;
 // A compound module creation named PC having two gates
module PC{
  parameters:
         @display("i=device/pc2,gold");
  gates:
       inout gate[];
 // Inclusion of Layers as submodules in PC module
  submodules:
   appL: ApplicationLayer {
        @display("p=45,26;i=block/sink");
    }
    tcpL: TransportLayer {
        @display("p=123,56;i=block/filter");
    }
    ipL: NetworkLayer {
        @display("p=123,137;i=old/proc2");
    }
```

```
        dlL: DataLinkLayer {
            @display("p=123,216;i=device/card");
        }
        phyL: PhysicalLayer {
            @display("p=57,226;i=device/card");
        }
       connections:
          appL.appGate[0] <--> tcpL.tcpGate[0];
          tcpL.tcpGate[1] <--> ipL.ipGate[0];
          ipL.ipGate[1] <--> dlL.dllGate[0];
          dlL.dllGate[1] <--> phyL.phyGate[0];
          phyL.phyGate[1] <--> gate++;
  }
```



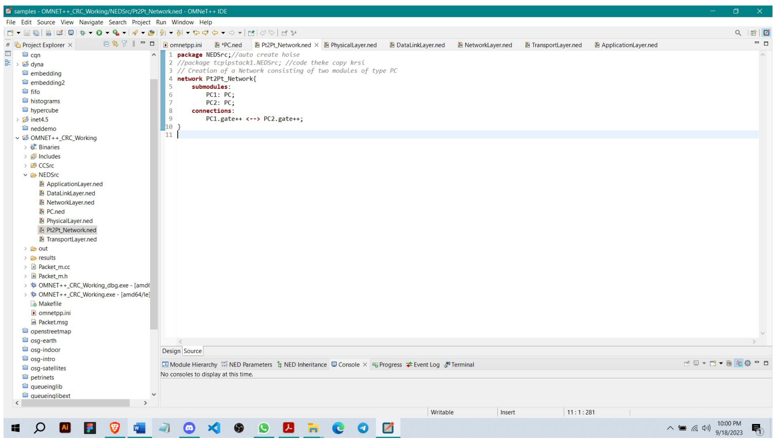
```
package NEDSrc;
// Package statement
//package tcpipstack1.NEDSrc;
 // A compound module creation named PC having two gates
module PC{
   parameters:
           @display("i=device/pc2,gold");
   gates:
         inout gate[];
 // Inclusion of Layers as submodules in PC module
   submodules:
    appL: ApplicationLayer {
          @display("p=45,26;i=block/sink");
     }
     tcpL: TransportLayer {
          @display("p=123,56;i=block/filter");
     }
     ipL: NetworkLayer {
          @display("p=123,137;i=old/proc2");
     }
     dlL: DataLinkLayer {
          @display("p=123,216;i=device/card");
     }
     phyL: PhysicalLayer {
          @display("p=57,226;i=device/card");
     }
    connections:
       appL.appGate[0] <--> tcpL.tcpGate[0];
       tcpL.tcpGate[1] <--> ipL.ipGate[0];
       ipL.ipGate[1] <--> dlL.dllGate[0];
       dlL.dllGate[1] <--> phyL.phyGate[0];
       phyL.phyGate[1] <--> gate++;

}
```

No consoles to display at this time.

**Fig-14:** NED file of PC Layer

## 7.Point to Point Network:

```
package NEDSrc;
//package tcpipstack1.NEDSrc;
//Creation of a Network consisting of two modules of type PC
network Pt2Pt_Network{
   submodules:
       PC1: PC;
           PC2: PC;
   connections:
           PC1.gate++ <--> PC2.gate++;
}
```



```
package NEDSrc;//auto create hoise
//package tcpipstack1.NEDSrc; //code theke copy krsi
// Creation of a Network consisting of two modules of type PC
network Pt2Pt_Network{
    submodules:
        PC1: PC;
        PC2: PC;
    connections:
        PC1.gate++ <--> PC2.gate++;
}
```

**Fig-15:** NED file of Point to Point Network Layer

## 5.2 CC files

### 1.Physical Layer:

```
#include<stdio.h>
#include<string.h>
#include<omnetpp.h>
#include"Packet_m.h"
using namespace omnetpp;

class PhysicalLayer : public cSimpleModule{
   protected:
      virtual void initialize() override;
      virtual void handleMessage(cMessage *msg) override;
};
Define_Module(PhysicalLayer);
void PhysicalLayer::initialize(){  }
void PhysicalLayer::handleMessage(cMessage *msg){
   TcpIpPacket *pkt = check_and_cast<TcpIpPacket*>(msg);
   //For packet receiving from Data Link Layer
   if(pkt->arrivedOn("phyGate$i",0)){
      char packetInfo [40];
      //For setting and sending of IP Packet
      pkt->setPhyHeader (uniform(1,6));
      sprintf(packetInfo,"PHY Header [%d] Added ",
      pkt->getPhyHeader());
      pkt->setPacketFormat(packetInfo);
      bubble(pkt->getPacketFormat());
      //Send packet on PC port
      send(pkt, "phyGate$o",1);
   }
   //Packet reception on PC port
   else if(pkt->arrivedOn("phyGate$i",1)){
      pkt->setPhyHeader(0);
      //Send packet to Data Link layer
      send(pkt, "phyGate$o",0);
   }
}
```

**2. Data Link Layer:**

```
#include<string.h>
#include<omnetpp.h>
#include"Packet_m.h"
using namespace omnetpp;

class DataLinkLayer : public cSimpleModule{
    private:
        int headersConcatenation;
        const int crcDivider =13;
        int crcValue;
    protected:
        virtual void initialize() override;
        virtual void handleMessage(cMessage *msg) override;
};
Define_Module(DataLinkLayer);
void DataLinkLayer::initialize(){}
void DataLinkLayer::handleMessage(cMessage *msg){
    TcpIpPacket *pkt = check_and_cast<TcpIpPacket *>(msg);
     //For packet receiving from Network Layer
    if(pkt->arrivedOn("dllGate$i",0)){
    //Setting of Data Link layer header
        pkt->setDllHeader(uniform(1,6));
         /* Concatenation of all header values+data for CRC code Calculation */
        headersConcatenation =(pkt->getAppHeader()+ pkt->getData()*10+ pkt-
>getTcpHeader()*100+ pkt->getIpHeader()*1000+  pkt->getDllHeader()*10000);
         //Calculation of CRC code
        crcValue = crcDivider -((headersConcatenation *100)%crcDivider);
        //Attach CRC code in packet before transmission
        pkt->setCrc(crcValue);
        bubble("CRC Added");
         //Send packet to Physical Layer
        send(pkt, "dllGate$o",1);
}
    //For packet receiving from Physical Layer
    if (pkt->arrivedOn("dllGate$i",1)){
        /* Artificial random error creation in received packets for
        Error Detection through CRC */
        int error = uniform(1,4);
        if(error!= 2)
        pkt->setAppHeader(7);
        /* At receiver side, concatenation of all header values + data
        for CRC */
        headersConcatenation = ( pkt->getAppHeader() + pkt->getData()*10 + pkt->
getTcpHeader()*100 + pkt->getIpHeader()*1000 + pkt->getDllHeader()*10000 );
         //Applying CRC check on received frame
        if((headersConcatenation *100 +  pkt->getCrc())%crcDivider== 0){
         //No Error Message Display
           bubble("CRC checked NO error");
         //Send packet to Network layer
           send(pkt, "dllGate$o",0);
       }
        else{
           bubble("ERROR Detected");
           delete pkt;      }}}
```

**3.Network Layer:**

```
#include<stdio.h>
#include<string.h>
#include<omnetpp.h>
#include"Packet_m.h"
using namespace omnetpp;

class NetworkLayer : public cSimpleModule{
    protected:
        virtual void initialize() override;
        virtual void handleMessage(cMessage *msg) override;
};
Define_Module(NetworkLayer);
void NetworkLayer::initialize(){ }
void NetworkLayer::handleMessage(cMessage *msg){
    TcpIpPacket *pkt = check_and_cast<TcpIpPacket *>(msg);
//For packet receiving from Transport Layer
    if(pkt->arrivedOn("ipGate$i",0)){
        char packetInfo[40];
//For setting and sending of IP Packet
        pkt->setIpHeader(uniform(1,6));
        sprintf(packetInfo,"IP Header [%d] Added ", pkt->getIpHeader());
        pkt->setPacketFormat(packetInfo);
        bubble(pkt->getPacketFormat());
//Sending packet to Data Link layer
        send(pkt, "ipGate$o",1);
    }
    if(pkt->arrivedOn("ipGate$i",1)){
        pkt->setIpHeader(0);
        //Sending packet to Transport layer
        send(pkt, "ipGate$o",0);
    }
}
```

**4.Transport Layer:**

```
#include<stdio.h>
#include<string.h>
#include<omnetpp.h>
#include"Packet_m.h"
using namespace omnetpp;

class TransportLayer : public cSimpleModule{
    protected:
        virtual void initialize() override;
        virtual void handleMessage(cMessage *msg) override;
};
Define_Module(TransportLayer);

void TransportLayer::initialize(){  }
void TransportLayer::handleMessage(cMessage *msg){
    TcpIpPacket *pkt = check_and_cast<TcpIpPacket *>(msg);
    //For packet receiving from Application Layer
    if(pkt->arrivedOn("tcpGate$i",0)){
    char packetInfo [40];
    //For Generation and setting of TCP header
    pkt->setTcpHeader(uniform(1,6));
    sprintf(packetInfo,"TCP Header [%d] added", pkt->getTcpHeader());
    pkt->setPacketFormat(packetInfo);
    *pkt = TcpIpPacket(packetInfo,0);
    bubble(packetInfo);
    send(pkt, "tcpGate$o",1);
}
    //For packet receiving from Network Layer
    if(pkt->arrivedOn("tcpGate$i",1)){
        pkt->setTcpHeader(0);
        send(pkt, "tcpGate$o",0);
    }
}
```

**5.Application Layer:**

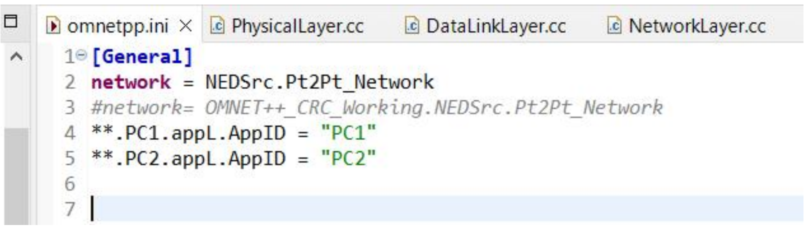
```
#include<stdio.h>
#include<string.h>
#include<omnetpp.h>
#include"Packet_m.h"
```

```
using namespace omnetpp;
class ApplicationLayer : public cSimpleModule{
   protected:
      virtual void initialize() override;
      virtual void handleMessage(cMessage *msg) override;
      virtual TcpIpPacket* generateNextMessage();
};
Define_Module(ApplicationLayer);
void ApplicationLayer::initialize(){
//To initialize Message Passing from PC1
   if(strcmp(par("AppID").operator const char *(),"PC1")==0){
      scheduleAt (0.0, generateNextMessage());
   }
}
void ApplicationLayer::handleMessage(cMessage *msg){
   TcpIpPacket* pkt = check_and_cast<TcpIpPacket *>(msg);
//On message arrival at input port of Application Layer
  if(pkt->arrivedOn("appGate$i",0)){ pkt->setAppHeader(0);
      delete pkt;
}
//For initialized or resend packet scenario
  if(msg->isSelfMessage()){
      //To display packet parameters
      char packetInfo[20];
      //To store Application Header & Data in packet
      sprintf(packetInfo,"AppHeader= %d\nData= %d",
      pkt->getAppHeader(),pkt->getData());
      send(pkt,"appGate$o",0);
      //To reschedule the Self message after every 100ms Delay
      scheduleAt(simTime()+.1, generateNextMessage());
      //To show packet details
      bubble (packetInfo);
  }
}
//For random massage generation
 TcpIpPacket* ApplicationLayer:: generateNextMessage(){
     char nextData[20];
//To generate random data as an integer value
     int generatedData = uniform(6,9);
     //To assign app Header
     int appHdr = uniform(1,6);
     sprintf(nextData,"%d%d",appHdr, generatedData);
     //To create new packet of created data
     TcpIpPacket* nextPkt = new TcpIpPacket(nextData);
     //Settings of Packet Parameters
     nextPkt->setData(generatedData);
     nextPkt->setAppHeader(appHdr);
```

```
    nextPkt->setPacketFormat(nextData);
    return nextPkt;
}
```

## 5.3 INI file:

```
[General]
network = NEDSrc.Pt2Pt_Network
#network= OMNET++_CRC_Working.NEDSrc.Pt2Pt_Network
**.PC1.appL.AppID = "PC1"
**.PC2.appL.AppID = "PC2"
```

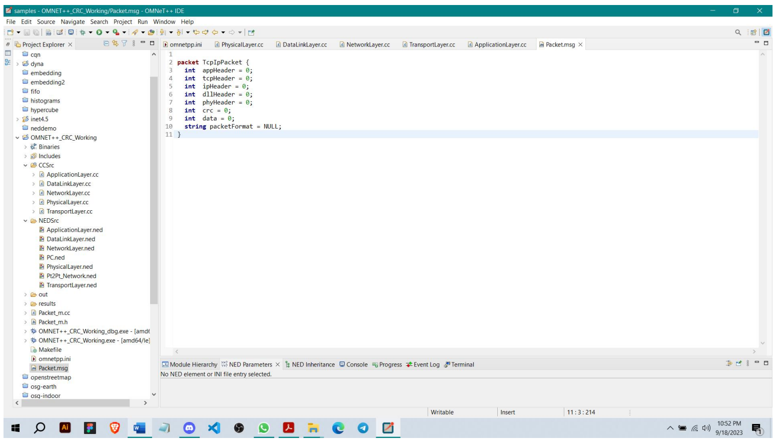


```
[General]
network = NEDSrc.Pt2Pt_Network
#network= OMNET++_CRC_Working.NEDSrc.Pt2Pt_Network
**.PC1.appL.AppID = "PC1"
**.PC2.appL.AppID = "PC2"
```

**Fig. 16:** Ini file

## 5.4 Message file:

```
packet TcpIpPacket {
  int  appHeader = 0;
  int  tcpHeader = 0;
  int  ipHeader = 0;
  int  dllHeader = 0;
  int  phyHeader = 0;
  int  crc = 0;
  int  data = 0;
  string packetFormat = NULL;
}
```



```
packet TcpIpPacket {
  int  appHeader = 0;
  int  tcpHeader = 0;
  int  ipHeader = 0;
  int  dllHeader = 0;
  int  phyHeader = 0;
  int  crc = 0;
  int  data = 0;
  string packetFormat = NULL;
}
```

**Fig. 17:** Message file

# CHAPTER 6
# CONCLUSION

## 6.1 Conclusion:

We have implemented CRC code generation for error detection at receiving end. The simulation shows that CRC code is generated at data link layer based on the packet received from network layer and both are forwarded to other PC. At receiving end CRC code is evaluated at data link layer, in the case of match packet forwarded to upper layers otherwise data link layer reports the error message.

This project has demonstrated the practical implementation and effectiveness of the Cyclic Redundancy Check (CRC) mechanism at the data link layer within the OMNeT++ simulation framework. By calculating CRC codes for transmitted messages and rigorously verifying their accuracy at the receiving end, we have showcased the pivotal role that CRC plays in enhancing data integrity in communication protocols. The successful execution of this project underscores the significance of CRC in safeguarding data against errors and reaffirms its relevance in modern data communication networks. As we move forward, further refinements and optimizations of CRC implementations hold promise for even more robust error detection and correction mechanisms in the evolving landscape of network protocols.