Experiment 4
Implement secure data transmission in cloud

Aim: To Implement secure data transmission in Cloud using OMNET++.

Software Required: OMNeT++ 6.0.1

Operating System Required: Windows OS

Algorithm:

To build a "network" that consists of two nodes, where one of the nodes will create a packet and the two nodes will keep passing the same packet back and forth.

Let's call the nodes tic and toc.

STEP 1: Start the OMNeT++ IDE by typing omnetpp in your terminal.

In the IDE, choose *New -> OMNeT++ Project* from the menu.

STEP 2: A wizard dialog will appear. Enter 'tictoc' as project name, choose *Empty project* when asked about the initial content of the project, then click *Finish*. An empty project will be created.

The project will hold all files that belong to our simulation

STEP 3: Adding the NED file

To add the file to the project,

Switch into *Source* mode, and enter the following: App.cc

```
#ifdef _MSC_VER
#pragma warning(disable:4786)
#endif

#include <vector>
#include <omnetpp.h>
#include "Packet_m.h"

using namespace omnetpp;

/**
 * Generates traffic for the network.
```

```
 */
class App : public cSimpleModule
{
  private:
    // configuration
    int myAddress;
    std::vector<int> destAddresses;
    cPar *sendIATime;
    cPar *packetLengthBytes;

    // state
    cMessage *generatePacket = nullptr;
    long pkCounter;

    // signals
    simsignal_t endToEndDelaySignal;
    simsignal_t hopCountSignal;
    simsignal_t sourceAddressSignal;

  public:
    virtual ~App();

  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(App);

App::~App()
{
    cancelAndDelete(generatePacket);
}

void App::initialize()
{
    myAddress = par("address");
    packetLengthBytes = &par("packetLength");
    sendIATime = &par("sendIaTime");  // volatile parameter
    pkCounter = 0;

    WATCH(pkCounter);
    WATCH(myAddress);

    const char *destAddressesPar = par("destAddresses");
    cStringTokenizer tokenizer(destAddressesPar);
    const char *token;
    while ((token = tokenizer.nextToken()) != nullptr)
        destAddresses.push_back(atoi(token));

    if (destAddresses.size() == 0)
        throw cRuntimeError("At least one address must be specified in
the destAddresses parameter!");

    generatePacket = new cMessage("nextPacket");
    scheduleAt(sendIATime->doubleValue(), generatePacket);
```

```
    endToEndDelaySignal = registerSignal("endToEndDelay");
    hopCountSignal = registerSignal("hopCount");
    sourceAddressSignal = registerSignal("sourceAddress");
}

void App::handleMessage(cMessage *msg)
{
    if (msg == generatePacket) {
        // Sending packet
        int destAddress = destAddresses[intuniform(0,
destAddresses.size()-1)];

        char pkname[40];
        sprintf(pkname, "pk-%d-to-%d-#%ld", myAddress, destAddress,
pkCounter++);
        EV << "generating packet " << pkname << endl;

        Packet *pk = new Packet(pkname);
        pk->setByteLength(packetLengthBytes->intValue());
        pk->setKind(intuniform(0, 7));
        pk->setSrcAddr(myAddress);
        pk->setDestAddr(destAddress);
        send(pk, "out");

        scheduleAt(simTime() + sendIATime->doubleValue(),
generatePacket);
        if (hasGUI())
            getParentModule()->bubble("Generating packet...");
    }
    else {
        // Handle incoming packet
        Packet *pk = check_and_cast<Packet *>(msg);
        EV << "received packet " << pk->getName() << " after " << pk-
>getHopCount() << "hops" << endl;
        emit(endToEndDelaySignal, simTime() - pk->getCreationTime());
        emit(hopCountSignal, pk->getHopCount());
        emit(sourceAddressSignal, pk->getSrcAddr());
        delete pk;

        if (hasGUI())
            getParentModule()->bubble("Arrived!");
    }
}
```

STEP 4: Adding the NED file

To implement the functionality of the Txc1 simple module in C++.

STEP 5: Adding the omnetpp.ini file:

To be able to run the simulation, we need to create an omnetpp.ini file. omnetpp.ini tells the simulation program which network you want to simulate.

We are now done with creating the model, and ready to compile and run it.

```
[Net60CutThrough]
network = networks.Net60
description = "60 node network with cut-through switching"
**.app.packetLength = 32768 bytes
**.useCutThroughSwitching = true  # let packets flow through the routers
**.destAddresses = "1 50"
**.sendIaTime = uniform(1ms, 5ms)

[Net60StoreAndForward]
network = networks.Net60
description = "60 node network with store-and-forward switching"
**.app.packetLength = 32768 bytes
**.destAddresses = "1 50"
**.sendIaTime = uniform(1ms, 5ms)
```

Explanation:

Store and Forward Switching

Store − and − forward packet switching is a technique where the data packets are stored in each intermediate node, before they are forwarded to the next node. The intermediate node checks whether the packet is error−free before transmitting, thus ensuring integrity of the data packets. In general, the network layer operates in an environment that uses store and forward packet switching.

he node which has a packet to send, delivers it to the nearest node, i.e. router. The packet is stored in the router until it has fully arrived and its checksum is verified for error detection. Once, this is done, the packet is transmitted to the next router. The same process is continued in each router until the packet reaches its destination.

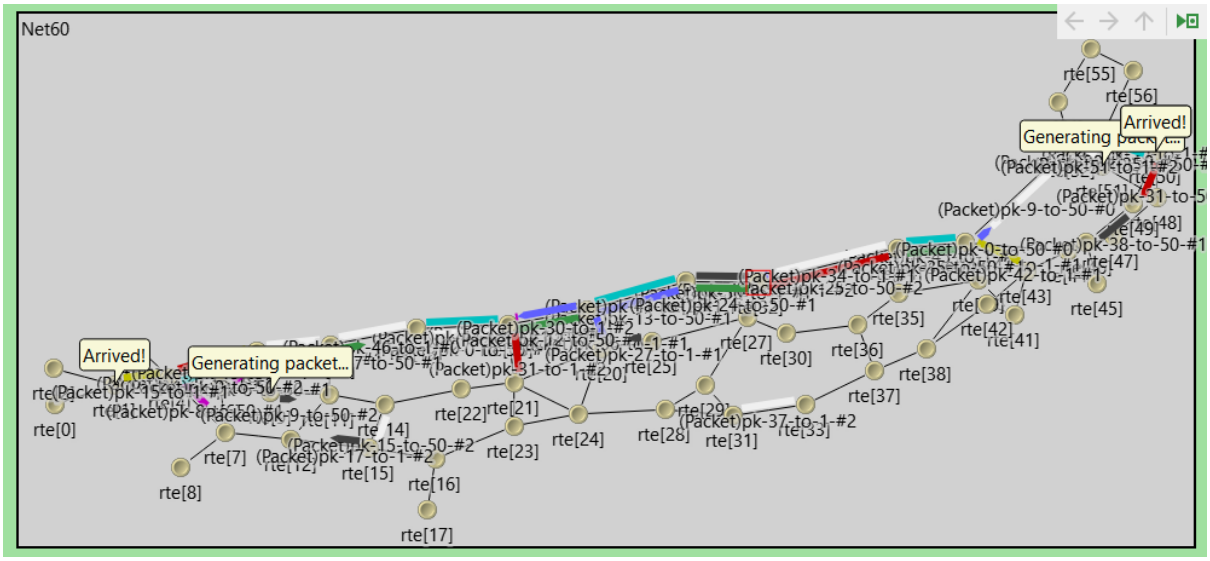
Store and Cut Through Switching

Cut-through switching is a method of switching data packets by the switching device that forwards the packets as soon as the destination address is available without waiting for the rest of the data to arrive. It supports low latency and high-speed transmission and requires less storage space. It is used in fiber channel transmission, SCSI traffic transmission, etc.

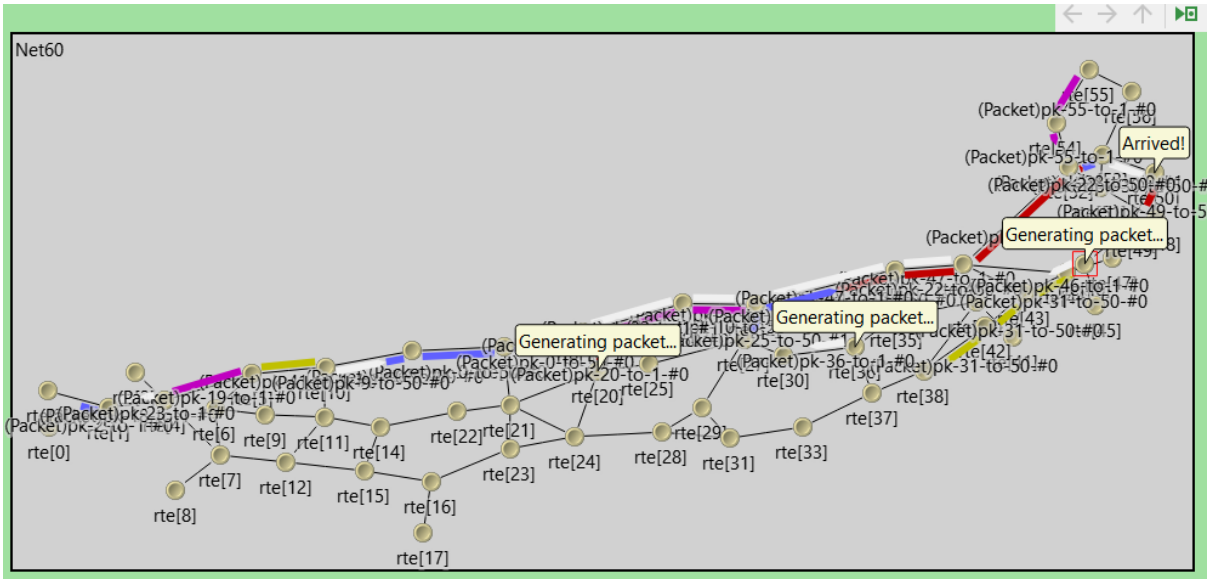
In cut−through switching, data transmission starts as soon as the destination address field arrives at the switching device. Then the device performs a lookup operation to

check whether the destination address is valid or not. If the address is found valid and the link to the destination is available then the switching device starts to transmit the packets to the destination without waiting for the rest of the frame to arrive.

Output: **Store and Forward Switching**



Store and Cut Through Switching



Result:

Thus, the formation of secure data transmission in Cloud using OMNET++ was implemented and executed successfully.