Experiment 5
Implement access management in cloud environment using OMNET++

Aim: To Implement access management in cloud environment 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: Routing.cc

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

using namespace omnetpp;
class L2Queue : public cSimpleModule
{
  private:
    intval_t frameCapacity;

    cQueue queue;
    cMessage *endTransmissionEvent = nullptr;
    bool isBusy;
```

```
    simsignal_t qlenSignal;
    simsignal_t busySignal;
    simsignal_t queueingTimeSignal;
    simsignal_t dropSignal;
    simsignal_t txBytesSignal;
    simsignal_t rxBytesSignal;

  public:
    virtual ~L2Queue();

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

Define_Module(L2Queue);

L2Queue::~L2Queue()
{
    cancelAndDelete(endTransmissionEvent);
}

void L2Queue::initialize()
{
    queue.setName("queue");
    endTransmissionEvent = new cMessage("endTxEvent");

    if (par("useCutThroughSwitching"))
        gate("line$i")->setDeliverImmediately(true);

    frameCapacity = par("frameCapacity");

    qlenSignal = registerSignal("qlen");
    busySignal = registerSignal("busy");
    queueingTimeSignal = registerSignal("queueingTime");
    dropSignal = registerSignal("drop");
    txBytesSignal = registerSignal("txBytes");
    rxBytesSignal = registerSignal("rxBytes");

    emit(qlenSignal, queue.getLength());
    emit(busySignal, false);
    isBusy = false;
}

void L2Queue::startTransmitting(cMessage *msg)
{
    EV << "Starting transmission of " << msg << endl;
    isBusy = true;
    int64_t numBytes = check_and_cast<cPacket *>(msg)->getByteLength();
    send(msg, "line$o");

    emit(txBytesSignal, numBytes);

    // Schedule an event for the time when last bit will leave the gate.
    simtime_t endTransmission = gate("line$o")->getTransmissionChannel()-
>getTransmissionFinishTime();
    scheduleAt(endTransmission, endTransmissionEvent);
}
```

```
void L2Queue::handleMessage(cMessage *msg)
{
    if (msg == endTransmissionEvent) {
        // Transmission finished, we can start next one.
        EV << "Transmission finished.\n";
        isBusy = false;
        if (queue.isEmpty()) {
            emit(busySignal, false);
        }
        else {
            msg = (cMessage *)queue.pop();
            emit(queueingTimeSignal, simTime() - msg->getTimestamp());
            emit(qlenSignal, queue.getLength());
            startTransmitting(msg);
        }
    }
    else if (msg->arrivedOn("line$i")) {
        // pass up
        emit(rxBytesSignal, (intval_t)check_and_cast<cPacket *>(msg)-
>getByteLength());
        send(msg, "out");
    }
    else {  // arrived on gate "in"
        if (endTransmissionEvent->isScheduled()) {
            // We are currently busy, so just queue up the packet.
            if (frameCapacity && queue.getLength() >= frameCapacity) {
                EV << "Received " << msg << " but transmitter busy and queue full:
discarding\n";
                emit(dropSignal, (intval_t)check_and_cast<cPacket *>(msg)-
>getByteLength());
                delete msg;
            }
            else {
                EV << "Received " << msg << " but transmitter busy: queueing
up\n";
                msg->setTimestamp();
                queue.insert(msg);
                emit(qlenSignal, queue.getLength());
            }
        }
        else {
            // We are idle, so we can start transmitting right away.
            EV << "Received " << msg << endl;
            emit(queueingTimeSignal, SIMTIME_ZERO);
            startTransmitting(msg);
            emit(busySignal, true);
        }
    }
}

void L2Queue::refreshDisplay() const
{
    getDisplayString().setTagArg("t", 0, isBusy ? "transmitting" : "idle");
    getDisplayString().setTagArg("i", 1, isBusy ? (queue.getLength() >= 3 ? "red"
: "yellow") : "");
}
```

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.

```
[Net60Bursty]
network = networks.Net60
**.appType = "BurstyApp"  # override "App" in [General]
**.app.packetLength = intuniform(2048 byte, 16384 byte)
**.destAddresses = "1 50"


[Net5SaturatedQueue]  # Note: this config is used by the
Python/Pandas tutorial -- do not touch!
network = networks.Net5
sim-time-limit = 200s
**.frameCapacity = 10
**.destAddresses = "1 4"
**.sendIaTime = uniform(100ms, 500ms)  # high traffic
**.app.packetLength = intuniform(50 bytes, 4096 bytes)
**.channel.*.result-recording-modes = -vector  # turn off vectors
from channels
**.result-recording-modes = all  # turn on recording of optional
vectors
```

Explanation

Bursty Framing

Frame bursting is a transmission technique used at the data link layer of the OSI model to increase the rate of transmission of data frames. It can be effectively deployed in Gigabit Ethernets to increase network throughput. It is specified in the draft 802.11e QoS specification.

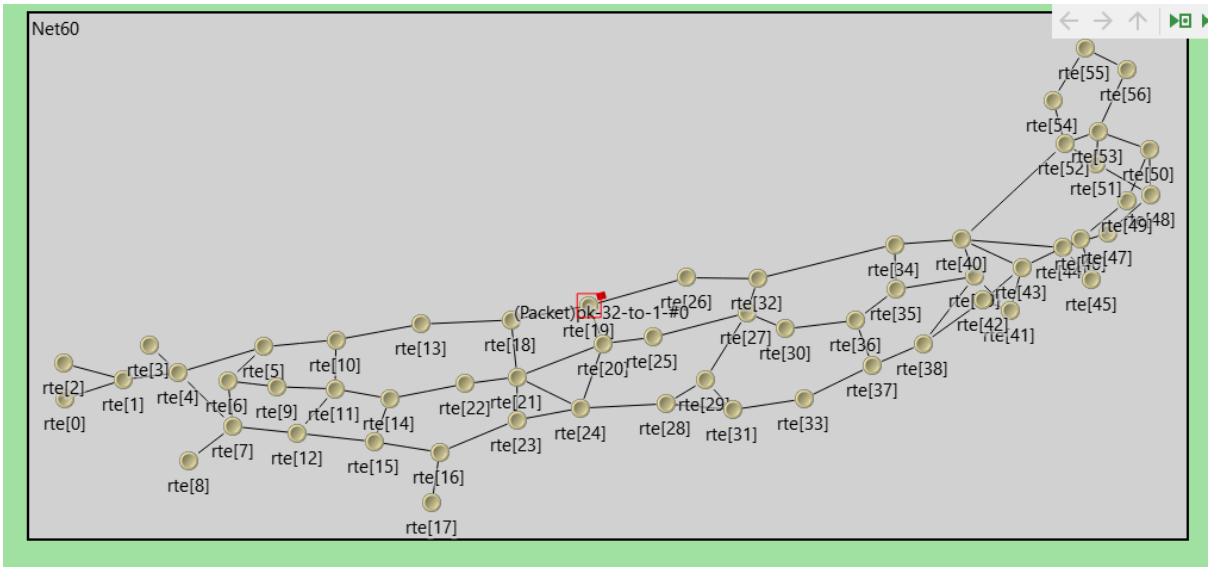
By this technique, a sender can transmit a series of frames in succession, without surrendering control on the transmission medium. A set of smaller frames may be concatenated to form a large frame that is transmitted at one go.

Priority Queue

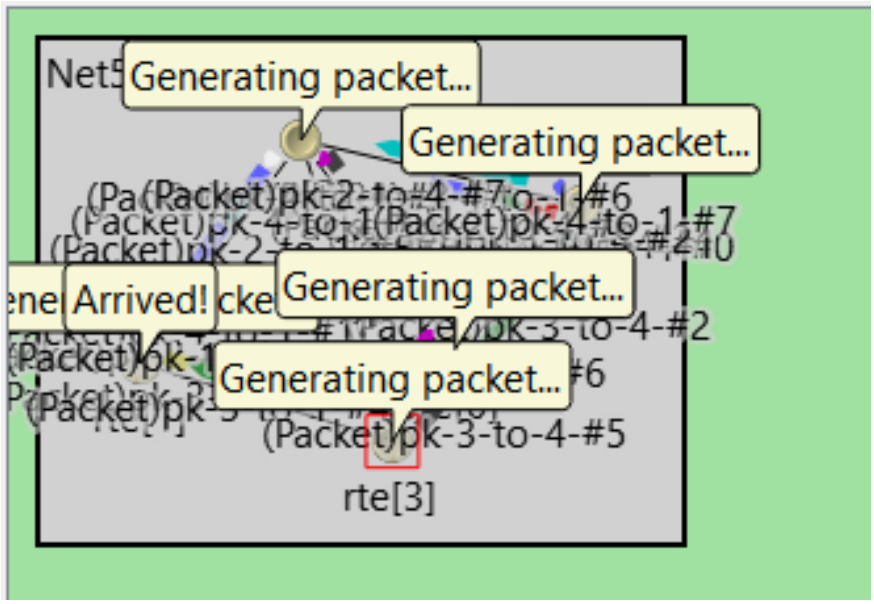
In Priority Queuing, instead of using a single queue, the router bifurcates the memory into multiple queues, based on some measure of priority. After this, each queue is handled in a FIFO manner while cycling through the queues one by one. The queues are marked as **High**, **Medium**, or **Low** based on priority. Packets from the High queue are always processed before packets from the Medium

queue. Likewise, packets from the Medium queue are always processed before packets in the Normal queue, etc. As long as some packets exist in the High priority queue, no other queue's packets are processed. Thus, high priority packets cut to the front of the line and get serviced first. Once a higher priority queue is emptied, *only then* is a lower priority queue serviced.

Output: **Net60 bursty**



Prioriity Queue



Result:

Thus, the formation of access management in Cloud Environment using OMNET++ was implemented and executed successfully.