The LNM Institute of Information Technology, Jaipur

Computer Networks Lab

Lab Assignment 3 Solution

Tasks 1: Behavioural Design: Update behavioral design of network implemented in Layered Architecture Assignment with the following functionality:

a) Network Layer: define *the number of packets* as a state variable
(Hint: TicToc Tutorial: Step 3)

b) Data-Link Layer: Include 10% packet loss at a destination
(Hint: TicToc Tutorial: Step 7)

c) Data-Link Layer: Retransmission of packet if packet loss i.e. After receiving a data PDU, the destination sends an ACK PDU. If the ACK PDU does not reach the source before a certain time period, known as the timeout, the source sends the same data PDU again. The timeout event is reset after each data PDU transmission. (Hint: TicToc Tutorial: Step 8 and 9)

Solution:

```
simple Nl_node

{

    parameters:

        int nl_id;

        int remaining_packets;

        int source;



       gates:

           input nl_in;

           output nl_out;
```

```
}
simple Dll_node
{
    parameters:
        int dll_id;

    gates:
        input dlln_in;
        output dlln_out;
        input dllc_in;
        output dllc_out;
}

module Comp
{
    parameters:
        int c_id;

    gates:
        input c_in;
        output c_out;

    submodules:
        Nl: Nl_node{nl_id=c_id;}
        Dll: Dll_node{dll_id=c_id;}
```

```
    connections:
        Nl.nl_in <-- Dll.dlln_out;
        Nl.nl_out --> Dll.dlln_in;
        Dll.dllc_in <-- c_in;
        Dll.dllc_out --> c_out;
}
network Comp_network
{
    parameters:
        int source;
        int dest;

    submodules:
        C1: Comp{c_id=1;}
        C2: Comp{c_id=2;}

    connections:
        C1.c_in <-- {delay=100ms;} <-- C2.c_out;
        C1.c_out --> {delay=100ms;} --> C2.c_in;
}

packet Nl_pkt{
```

```
    int Nl_pkt_id;
    int Nl_pkt_type;

}
packet Dll_pkt {
    int Dll_pkt_id;
    int Dll_pkt_type;
}
#ifndef __NLDLL_NL_NODE_H_
#define __NLDLL_NL_NODE_H_

#include <omnetpp.h>
#include <nl_pkt_m.h>

using namespace omnetpp;

/**
 * TODO - Generated class
 */

class Nl_node : public cSimpleModule
{
  protected:
    int id;
    cGate* in;
```

```
    cGate* out;
    int remaining_packets;
    int source;
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};


#endif
#include "nl_node.h"


Define_Module(Nl_node);


void Nl_node::initialize()
{
    remaining_packets = par("remaining_packets");
    // TODO - Generated method body
    id = par("nl_id");
    in = gate("nl_in");
    source = par("source");
    out = gate("nl_out");
    if (id==source && remaining_packets!=0){
        cMessage* event = new cMessage();
        scheduleAt(0, event);
    }
```

```
}


void Nl_node::handleMessage(cMessage *msg)
{
    // TODO - Generated method body
    if (msg->isSelfMessage()){
        Nl_pkt* data = new Nl_pkt();
        if (remaining_packets == 0){
            return;
        }
        data->setNl_pkt_id(remaining_packets--);
        data->setNl_pkt_type(1);

        send(data, out);
        cMessage* event = new cMessage();
        scheduleAt(simTime()+200, event);

    }
    else{
        delete(msg);
    }

}

#ifndef __NLDLL_DLL_NODE_H_
```

```
#define __NLDLL_DLL_NODE_H_


#include <omnetpp.h>
#include <dll_pkt_m.h>
#include <nl_pkt_m.h>


using namespace omnetpp;


/**
 * TODO - Generated class
 */
class Dll_node : public cSimpleModule
{
  protected:
    int id;
    cGate* in_n;
    cGate* out_n;
    cGate* in_c;
    cGate* out_c;
    Dll_pkt* copy_message;
    cMessage* timeoutEvent;
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
```

```
#endif

#include "dll_node.h"



Define_Module(Dll_node);


void Dll_node::initialize()

{

    // TODO - Generated method body

    id = par("dll_id");

    in_n = gate("dlln_in");

    out_n = gate("dlln_out");

    in_c = gate("dllc_in");

    out_c = gate("dllc_out");

    copy_message = new Dll_pkt();

    timeoutEvent = new cMessage("timeoutEvent");



}


void Dll_node::handleMessage(cMessage *msg)

{

    // TODO - Generated method body

    if(msg==timeoutEvent){

        send(copy_message, out_c);

        cancelEvent(timeoutEvent);
```

```
        scheduleAt(simTime()+200, timeoutEvent);
    }


    else if (msg->getArrivalGate()==in_n)
    {
        Dll_pkt* data = new Dll_pkt();
        Nl_pkt* data_to_encapsulate = check_and_cast<Nl_pkt*>(msg);
        data->encapsulate(data_to_encapsulate);
        data->setDll_pkt_type(1);
        send(data, out_c);
        copy_message = data->dup();
        cancelEvent(timeoutEvent);
        scheduleAt(simTime()+200, timeoutEvent);


    }
    else if(msg->getArrivalGate()==in_c)
    {
        Dll_pkt* message = check_and_cast<Dll_pkt*>(msg);
        if (message->getDll_pkt_type()==1)
        {
            message->decapsulate();
            Dll_pkt* ack = new Dll_pkt();
            ack->setDll_pkt_type(0);
            if (uniform(0, 1) <= 0.8) {
                            delete (message);
```

```
                        }
            else{
            send(message, out_n);
            send(ack, out_c);
            }

        }
        else{
            cancelEvent(timeoutEvent);
            delete(msg);
        }
    }


}
[General]
network = Comp_network
record-eventlog = true
**.remaining_packets = 10
**.source = 1
**.dest = 2
```