Bilgisayar Ağ Yönetim Laboratuvarı Final Ödevi

Soru 2) Kapsayan Ağaç Oluşturma

Projeye başlarken vize ödevimde kullandığım n node sayılı random ağ oluşturan .ned dosyasını kullandım. Projenin büyük bir kısmı gelen mesaj tiplerine göre yapılacak işlemleri oluşturduğu için öncelikle handleMessage komutunun içini doldurdum.

Başlangıç noktasını seçerken dinamik bir arayüz kullanmadım. Projedeki .cc dosyasında aşağıda açıkladığım gibi bir değişiklik yapmak yeterli oluyor.

Kodlara geçmeden son olarak bitirme koşulunu kontrol edemedim ve yazdığım bu kodda bitme koşulu herhangi bir nokta başka bir noktaya ack ya da probe mesajı gönderememesi oluyor. Bu program biterken çok hafif bir hantallık yaratsa da program doğru çalışıyor.

Program Kodlarını aşağıya ekledim en karışık olan cc uzantolı dosya üzerinde yaptığım işlemleri detaylıca anlattım.

Programın .ini uzantılı dosyası

```
[General]
network = agDenemesi
```

Programın .ned uzantılı dosyası

```
simple Node
{
    parameters:
        @display("i=abstract/router_vs");
        int usuless = default(0);
        //        volatile double delay = uniform(1,20);
    gates:
        //      input in[];
        //      output out[];
        inout g[];
}

network agDenemesi
{
    parameters:
        int n @prompt("Number of nodes") = default(8);
        //        int baslangic @promt("Başlangıç noktasını giriniz") =
default(0);
        //        int bitis @promt("Başlangıç noktasını giriniz") = default(1);
        volatile int posX = intuniform (0,50);
        volatile int posY = intuniform (0,50);
        double connectedness; // 0.0<x<1.0
        @display("bgb=134,129");
    submodules:
        node[n]: Node {
            parameters:
                @display("p=$posX,$posY");
                //                usuless;
        }

    connections allowunconnected:
        for i=0..n-1, for j=0..n-1 {
                node[i].g++ <--> { delay = 100ms;} <--> node[j].g++ if i != j &&
uniform(0,1)< connectedness;
        }
}
```

Programın .cc uzantılı dosyası

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


using namespace omnetpp;

class Node : public cSimpleModule
{
private:
    cMessage *probe;
    cMessage *act;
    cMessage *reject;
    cMessage *baslangic;
    std::vector<int> parents;
    std::vector<int> childs;
    //    std::vector<int>::iterator it;
protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
    virtual void forwardMessage(cMessage *msg);
};

Define_Module(Node);

//BU KISIM İLERİDE PARENTS VE CHİLDS VEKTÖRLERİNDEKİ ELEMANLARI YAZDIRMAK İÇİN
YAZILMIŞ BİR FONSİYON
void print(std::vector<int> const &input)
{
    for (int i = 0; i < input.size(); i++) {
        EV << input.at(i) << ' ';
    }
}

//HER NODE'UN BAŞLANGIÇTA SAHİP OLACAĞI NİTEKLİKLERİ INITIALIZE ETTİĞİMİZ YER
//AYRICA PROGRAMIN BAŞLANGIÇ NOKTASINI getIndex() KISMINDA SEÇİYORUZ
void Node::initialize()
{
    std::vector<int> parents;
    std::vector<int> childs;
    //    std::vector<int>::iterator it;

    if (getIndex() == 0) {
        baslangic = new cMessage("baslangic");
        EV << "Program baslayacak \n";
        scheduleAt(0.0, baslangic);

    }
}

//İşlerin neredeyse tamamının yapıldığı handleMessage kısmı.
void Node::handleMessage(cMessage *msg)
```

```
{

    //gelen mesaj başlangıç ise tüm komşu noktalara kontrol etmeksizin probe
mesajı gönderiyoruz.
    if(strcmp("baslangic", msg->getName()) == 0 ){
        int n = gateSize("g");
        for (int i = 0; i < n; i++)
        {
            probe = new cMessage("probe");
            send(probe, "g$o", i);
        }
    }

    //  Gelen mesaj probe mesajı ise öncelikle daha önce parent var mı diye
kontrol ediyoruz.
    //  Eğer parent yoksa mesajın geldiği yerin id'sini parentsa iletiyoruz ve
gelen yere
    //  bir adet ack mesajı, gelen yer hariç diğer tüm noktalara probe mesajı
gönderiyoruz.
    //  Eğer daha önce bir parent varsa mesajın geldiği yere reject mesajı
gönderiyoruz
    else if (strcmp("probe", msg->getName()) == 0 ) {
        if(parents.empty() == true){
            parents.push_back(msg->getSenderModuleId());
            EV << "parents = " ;
            print(parents);
            // Mesajın geldiği yere ack mesajı yolladığımız kısım
            cGate * sender = msg->getSenderGate();
            for (cModule::GateIterator i(this); !i.end(); i++)
            {
                cGate *gate = i();
                std::string gateStr = gate->getName();
                if (gateStr == "g$o" && gate->getPathEndGate()->getOwnerModule()
== sender->getOwnerModule() )
                {
                    int senderId = gate->getIndex();

                    act = new cMessage("act");
                    send(act, "g$o", senderId);
                }
            }
            // Mesajın geldiği yere ack mesajı yolladığımız kısım

            // Mesajın geldiği yer hariç her yere probe mesajı yolladığımzı kısım
            for (cModule::GateIterator i(this); !i.end(); i++)
            {
                cGate *gate = i();
                std::string gateStr = gate->getName();
                if (gateStr == "g$o" && gate->getPathEndGate()->getOwnerModule()
== sender->getOwnerModule() )
                {
                    int senderId = gate->getIndex();
                    int n = gateSize("g");
                    for (int i = 0; i < n; i++)
                    {
                        if( i == senderId){
                        }
                        else{probe = new cMessage("probe");
                        send(probe, "g$o", i);
```

```
                        }
                    }
                }
            }
            // Mesajın geldiği yer hariç her yere probe mesajı yolladığımzı kısım
        }
        else {
            cGate * sender = msg->getSenderGate();
            for (cModule::GateIterator i(this); !i.end(); i++)
            {
                cGate *gate = i();
                std::string gateStr = gate->getName();
                if (gateStr == "g$o" && gate->getPathEndGate()->getOwnerModule()
== sender->getOwnerModule() )
                {
                    int senderId = gate->getIndex();
                    reject = new cMessage("reject");
                    send(reject, "g$o", senderId);
                }
            }
        }
    }

    //Eğer bir mesaj probe yada reject değilse ack'tır
    // Ack mesajı alan bir nokta mesajın geldiği yeri çocuğu olarak işaretler.
    else if(strcmp("act", msg->getName()) == 0) {
        childs.push_back(msg->getSenderModuleId());
        EV << "childs = " ;
        print(childs);

    }
}

//Mesaj gönderme dahil tüm işlemleri handleMessage kısmında yaptığım için bu
kısıma birşey yazma gereği duymadım.
void Node::forwardMessage(cMessage *msg){
}
```