T.C.

ONDOKUZ MAYIS ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Bilgisayar Ağ Yönetimi Laboratuvar Dersi

Final Projesi

Hazırlayanlar

14060292– Tunahan YETİMOĞLU

Danışman

Dr. Öğr. Üyesi – Sercan Demirci

ÖDEV METNİ

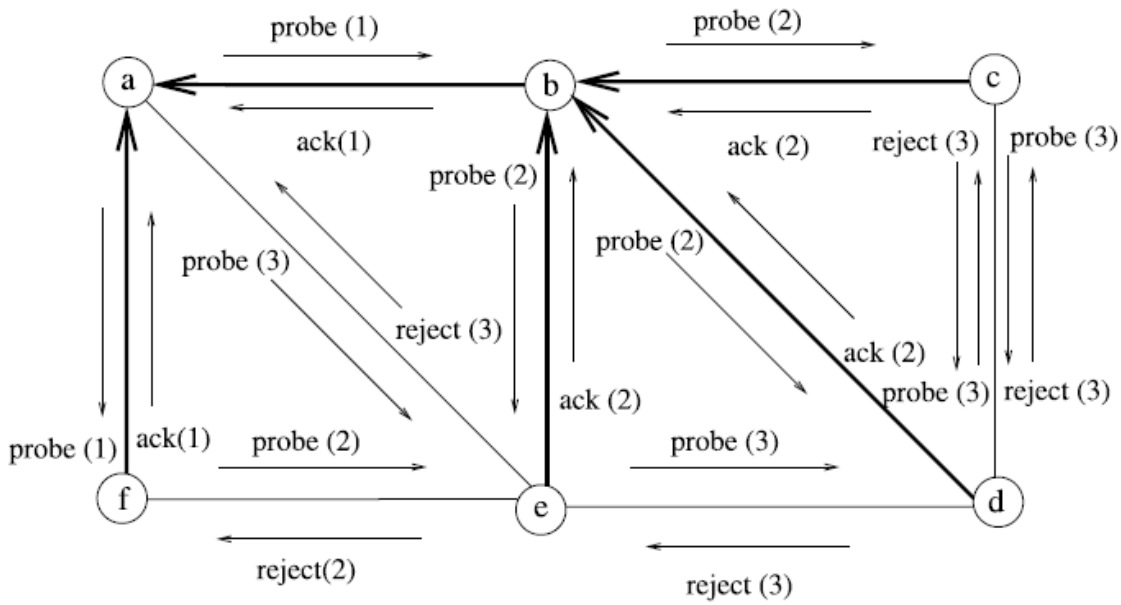
Soru 2 . Sel Basma Tabanlı Asenkron Kapsayan Ağaç Oluşturma

Algorithm 1 *Flood_ST*

```
1:  int parent ← ⊥
2:  set of int childs ← ∅, others ← ∅
3:  message types probe, ack, reject
4:
5:  if i = root then                                   ▷ root initiates tree construction
6:      send probe to Γ(i)
7:      parent ← i
8:  end if
9:
10: while (childs ∪ others) ≠ (Γ(i)\{parent}) do
11:     receive msg(j)
12:     case msg(j).type of
13:             probe:  if parent = ⊥ then              ▷ probe received first time
14:                         parent ← j
15:                         send ack to j
16:                         send probe to Γ(i)\{j}
17:                     else                           ▷ probe received before
18:                         send reject to j
19:             ack:    childs ← childs ∪ {j}          ▷ include j in children
20:             reject: others ← others ∪ {j}          ▷ include j in unrelated neighbors
21: end while
```

Flood_ST olarak adlandırılan yukarıdaki algoritmada kullanılan mesaj tipleri probe, ack ve reject'dir. Bir kapsayan ağaç oluşturmak isteyen herhangi bir düğüm, probe mesajını komşularına göndermektedir. Daha sonra bu mesaj diğer düğümlere aktarılarak algoritma başlamaktadır. Bir düğüm birden fazla probe mesajı alabildiğinden, Algoritma 1'de gösterildiği gibi bir düğümün probe mesajı alınıp alınmadığını kontrol etmek için ack ve reject mesajları gereklidir.

root, algoritmayı başlatır ve i düğümü bir probe mesajı aldığında, gönderici j'yi parent olarak işaretler ve j'ye bir ack mesajı gönderir. parent j, bir ack mesajı aldıktan sonra, çocuklarından birisini çocuğu olarak işaretler. Sonrasında düğüm i, parent j haricinde tüm komşularına probe mesajını gönderir. Bir düğüm, komşu bir düğümden probe mesajı aldığında zaten bir parent'a sahipse, komşu düğümüne reject mesajı gönderir. Algoritmanın sonlandırma koşulu, çocukların ve bir i düğümün ilişkisiz komşularının sayısının, ebeveyn dışındaki komşularına eşit olduğunda ortaya çıkmasıdır. Bu durum algoritmanın 10. satırında işaret edilmiştir. Dikkat edilmelidir ki algoritmanın 10. ve 21. satırları arasındaki ana gövdesi de root tarafından yürütülmektedir



Şekil 1. Flood_ST algoritması tarafından oluşturulan bir örnek kapsayan ağaç

Şekil 1'de, a, b, c, d, e ve f olmak üzere altı düğümden oluşan bir ağ üzerinde Flood_ST algoritması tarafından oluşturulan bir örnek kapsayan ağaç görülmektedir. Düğüm a root'dur ve komşuları b, e ve f'ye probe mesajları göndererek ağacın inşasına başlar. Aynı etiketli mesajlar aynı zaman çerçevesinde eş zamanlı olarak gerçekleştiğinden her bir mesaj zaman çerçevesi numarası ile etiketlenir. Bir başka ifadeyle aynı etiketli mesaj varsa bunlar aynı zaman çerçevesi içerisinde gerçekleşmiş demektir. a ve e arasındaki bağlantıdaki gecikmeden dolayı, ilk önce b'den e'ye probe mesajı ulaşmaktadır. Daha sonra root'dan iki zıplama (hop) uzaklıkta olmasına rağmen b düğümü e düğümünün parent'ı olmaktadır. En son oluşan kapsayan ağaç Şekil 1 üzerinde koyu çizgilerle gösterilmiştir.

RAPOR

İstenen algoritma, kaynak kod yazımı ve simülasyonunu “OMNeT++” üzerinde gerçekleştirilmiştir.

Algoritmada ki sabit terimler kaynak kodun en başında tanımlanmıştır.

```
#define ROOT "a"
#define GATENAME "gate$o"
#define GATENAME_i "gate$i"

#define PROBECODE 2
#define ACKCODE 1
#define REJECTCODE 0
```

Ekran Görüntüsü 1.0. Sabit Tanımları

Buradaki “ROOT“algoritmanın başlangıç düğümü belirtmektedir. “GATENAME” ve “GATENAME_i “ kodun içerisinde çok fazla tekrar ettiği için “define” ile tanımlanmıştır. Tanımlanan “PROBECODE,ACKCODE,REJETCODE” ise programda ki mesajların beklenen geri dönüş değerleridir.

```
class FST : public cSimpleModule
{
private:
    bool hasParent;
    int gateIndex;
    int gateId;
    int messageCode;
    std::vector<std::string> children;

protected:
    virtual Message *generateMessage(const char *name,short int kind);
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;

};
```

Ekran Görüntüsü 1.1. Sınıf Tanımı

OMNeT++ bize C++ ile kaynak kod yazmamıza izin veriyor. Programın içerisinde nesnelerimize ait (Node,Düğüm) parametreleri ve sınıfa ait metodları burada tanımladık.

```
void FST::initialize()
{
    hasParent = false;
    if (strcmp(ROOT,getName()) == 0) {
        hasParent = true;
        Message *msg = generateMessage("Probe", 2);
        scheduleAt(0.0, msg);
    }
}
```

Ekran Görüntüsü 1.2. initialize() metodu

Bu metot, simülasyon başladığında çalışan metottur. Simülasyon başladığında .”ned” klasöründe tanımladığımız “network” bu metodla birlikte “initialize” edilir. Her nodun bir “hasParent” diye parentı olup olmaması durumunu belirten parametresi mevcuttur. “if” bloğunda ise eğer tanımlanan sıradaki node, “ROOT” olarak seçtiğim node ise, “hasParent” değerini “true” yapıp, kendisine “Probe” mesajı göndermesini sağlıyor.

```
Message *FST::generateMessage(const char *name,short int kind)
{

     Message *msg = new Message(name,kind);
    if(strcmp(name, "Probe") == 0){
        msg->setMessageCode(2);
    }else if(strcmp(name,"Ack") == 0){
        msg->setMessageCode(1);
    }else if(strcmp(name,"Reject") == 0){
        msg->setMessageCode(0);
    }
    return msg;
}
```

Ekran Görüntüsü 1.3. generateMessage() metodu

Bu metot, mesaj adı ve türü adında 2 adet parametre almaktadır. Gelen bu parametrelerle sonradan tanımlanan “Message” paketine göre yeni bir mesaj oluşturuyor.Mesajın ismine göre, mesaja ait bir “messageCode” u “setMessageCode” ile tanımlıyor ve bu mesajı “return” ediyor.

```
L6 packet Message
L7 {
L8     int messageCode;
L9 }
```

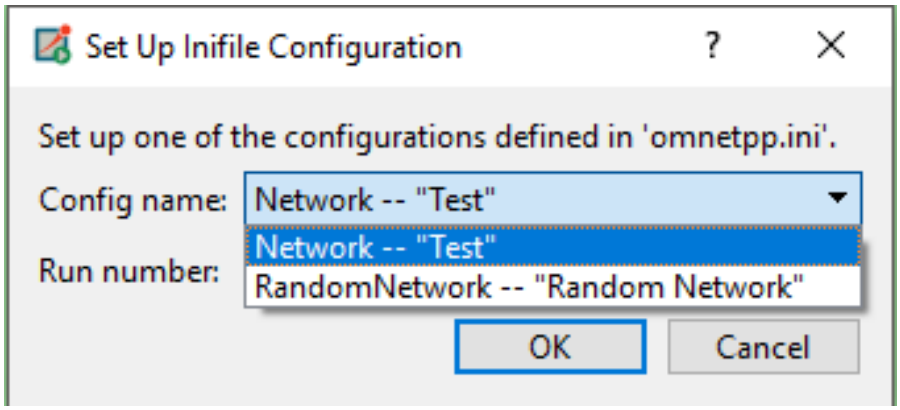
Ekran Görüntüsü 1.4. Message Paketi

```
Message *mmsg = check_and_cast<Message *>(msg);
messageCode = mmsg->getMessageCode();

if(messageCode == 0){
    cDisplayString& arrivalDispStr = gate(GATENAME, mmsg->getArrivalGate()->getIndex())->getDisplayString();
    arrivalDispStr.parse("ls=,0");
}
else{
     switch(messageCode){
    case (PROBECODE):
        gateId =  msg->getArrivalGateId();
        if( gateId != -1){
            gateIndex = msg->getArrivalGate()->getIndex();
            if(hasParent == false){
                hasParent = true;
                send(generateMessage("Ack", 1), GATENAME, gateIndex);
                cDisplayString& arrivalDispStr = gate(GATENAME, gateIndex)->getDisplayString();
                arrivalDispStr.parse("ls=green,3");

                for(int i = 0; i < this->gateCount() / 2; i++){
                    if(msg->arrivedOn(GATENAME_i, i)){continue;}
                    send(generateMessage("Probe", 2), GATENAME, i);
                }
            }else{
                send(generateMessage("Reject", 0), GATENAME, gateIndex);
            }
         }else{
                for(int i = 0; i < this->gateCount() / 2; i++){
                    if(msg->arrivedOn(GATENAME_i, i)){continue;}
                    send(generateMessage("Probe", 2),GATENAME, i);
                }
            }
            break;
    case (ACKCODE):
            children.push_back(msg->getArrivalGate()->getPreviousGate()->getOwner()->getName());
            cDisplayString& arrivalDispStr = gate(GATENAME, msg->getArrivalGate()->getIndex())->getDisplayString();
            arrivalDispStr.parse("ls=blue,5");

            EV << getName() << "'s childs: ";
            for(int i = 0; i < children.size(); i++){
                EV << children.at(i) << " , ";
            }
            EV << endl;
            break;
    }
}
delete mmsg;
```

Ekran Görüntüsü 1.5. handleMessage() metodu

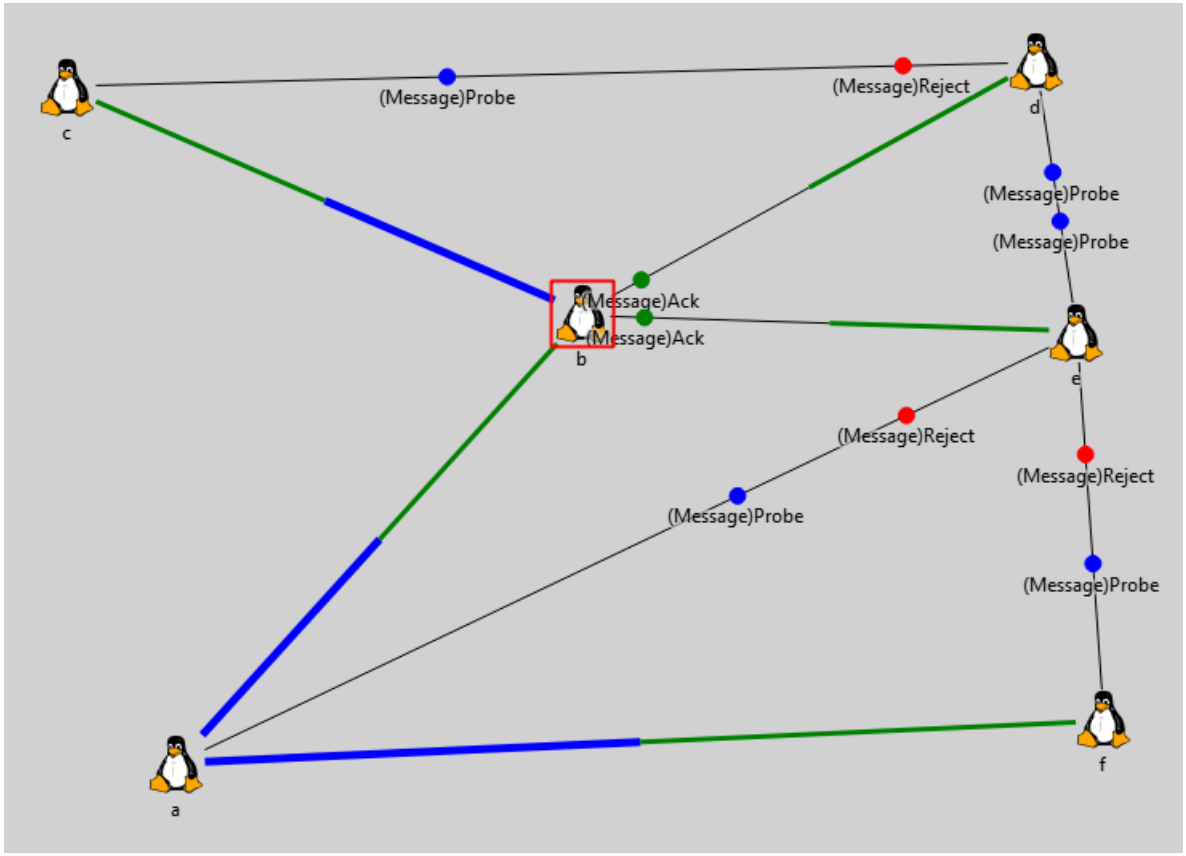
handleMessage() metotu, düğümlere mesaj geldiğinde çalışır.Algoritmanın asıl çalıştığı yer denebilir. Gelen mesajı, oluşturduğum “Message” türüne “check_and_cast”, yani tip dönüşümü, çeviriyorum. Gelen mesajın mesajın kodu “PROBECODE” ise algoritmanın “13 - 18 “ arasındaki adımlar, “ACKCODE” ise 19. Satırı, “REJECTCODE” ise 20. Satırı çalışmaktadır.

Programdan Görüntüler

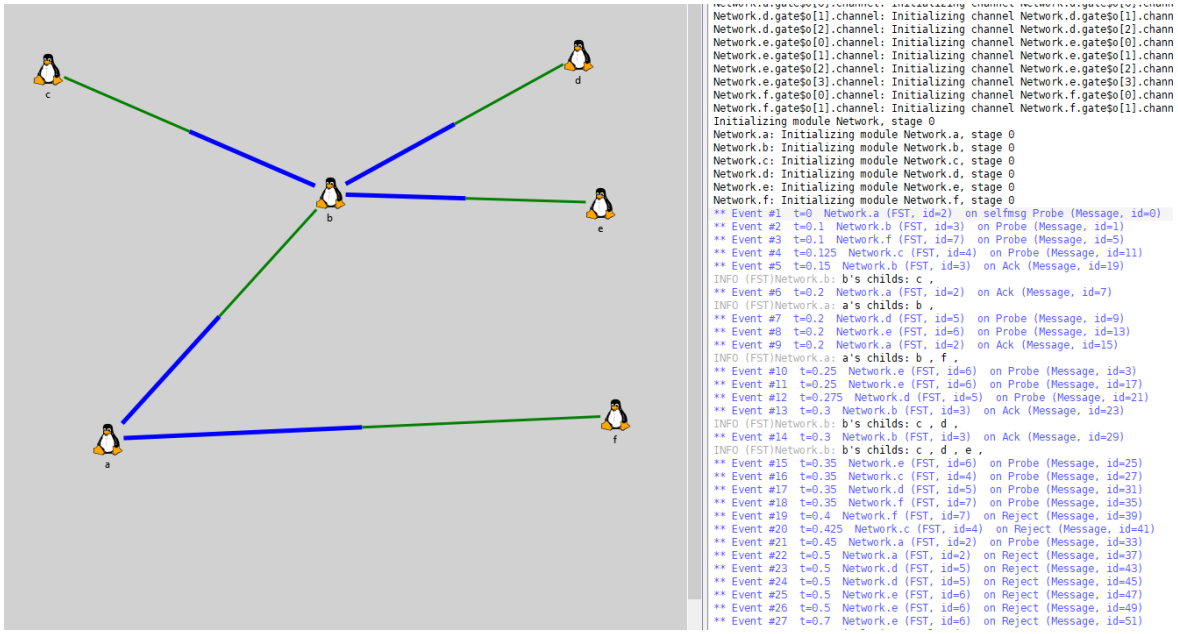


Ekran Görüntüsü 2.0

Simülasyon ilk çalıştığında “config name” seçilmesi bekleniyor. Ödev metninde verilen 6 node luk örneğin simülasyonu için “Test”, Rastgele oluşturulan bir “Network” için RandomNetwork seçilmesi bekleniyor.



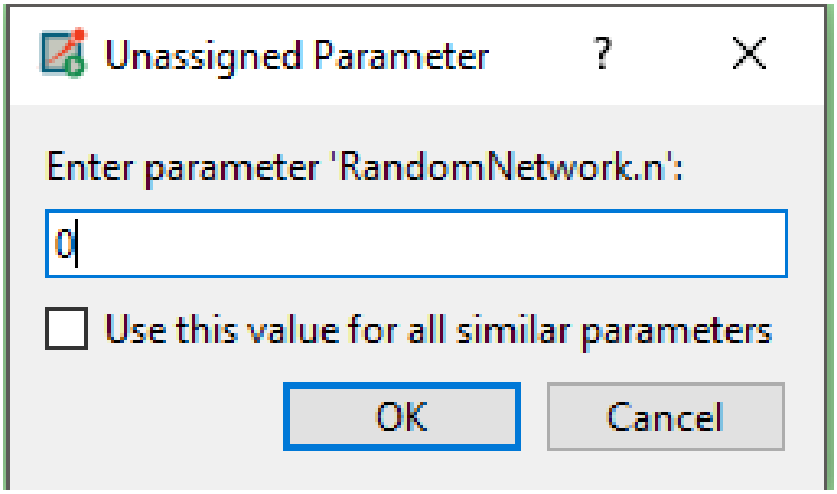
Ekran Görüntüsü 2.1 Test Simülasyonu çalışma anı



Ekran Görüntüsü 2.1 Test Simülasyonu son durum

Mavi çizginin başladığı taraf “probe” mesajını gönderen “parent”, Yeşil çizginin başladığı yer “child” çocuk düğüm.

RandomNetwork seçildiğinde bizden node sayısı istenir.

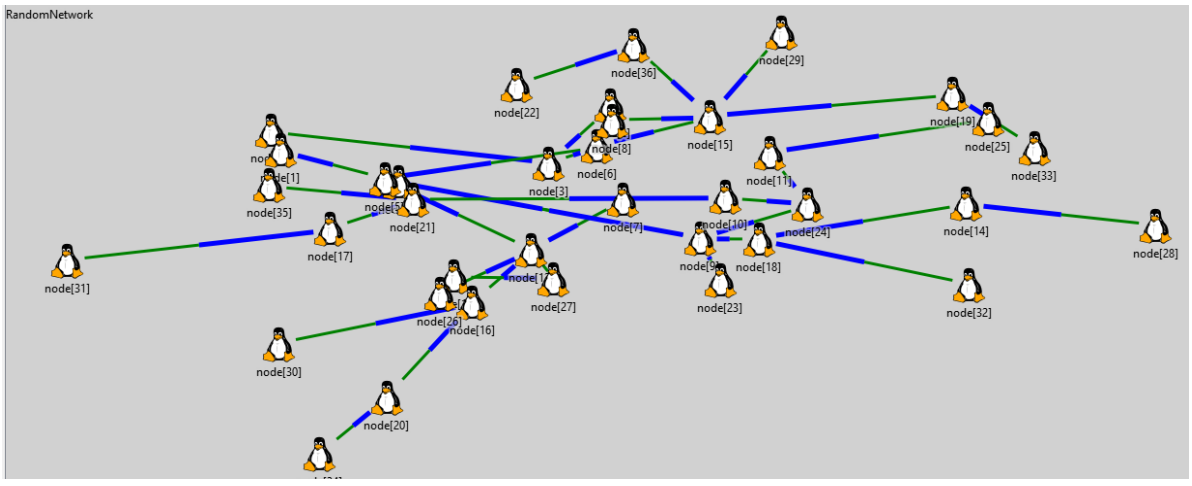


Ekran Görüntüsü 2.2. RandomNetwork için Parametre

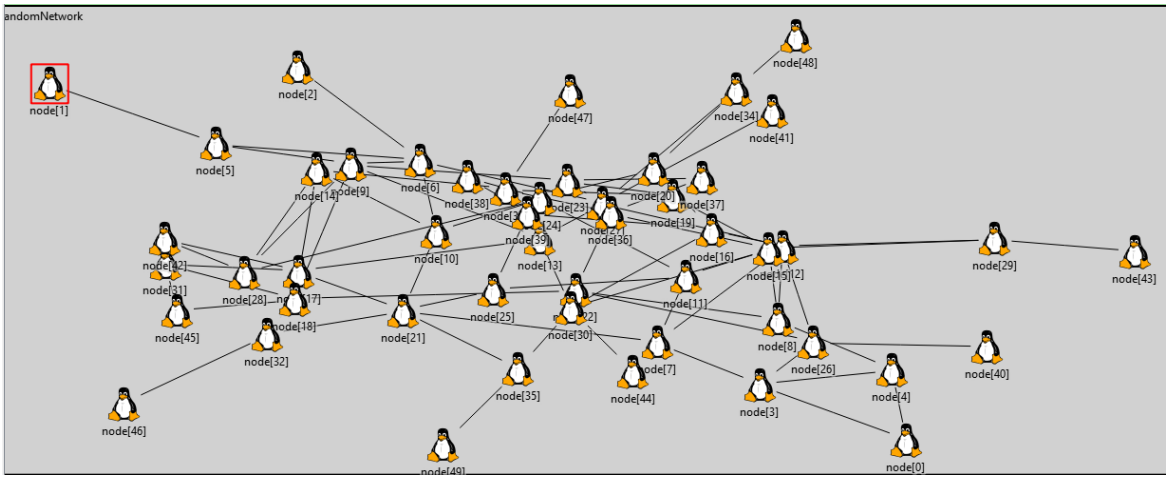
```
simple Node
{
        @display("i=abstract/penguin");
    gates:
        inout g[];
}

network RandomNetwork
{
    parameters:
        int n;  // Number of Nodes
    @display("bgb=3592.1213,1401.3888");
    types:
        channel Channel extends ned.DelayChannel
        {
            delay = default(1ms);
        }
    submodules:
        node[n]: Node;
    connections:
        for i=0..int(n/2)-1 {
            node[i].g++ <--> Channel {delay = intuniform(0ms, 10000ms); }  <--> node[i+4].g++;
        }
        for i=0..int(n/5)+5{
            node[i*3].g++ <--> Channel {delay = intuniform(0ms,1000ms); } <--> node[i+3].g++;
        }
        for i=1..int(n/3)-3{
            node[i*3].g++ <--> Channel {delay = intuniform(0ms, 1000ms); } <--> node[i+1].g++;
        }
        for i=7..int(n-15){
            node[i].g++ <--> Channel {delay = intuniform(0ms,1000ms); } <--> node[i+14].g++;
        }
}
```

Ekran Görüntüsü 2.3 Random Topology .ned file

Node sayısı >= 37 için her durumda çalışır. Döngüler neticesinde 37 dan küçük bir değer girersek bazı komşuluklar tekrar ettiği için, bazı düğümler komşusuz kaldığı için veya verilen sayıdan daha büyük bir düğüm değeri atamaya çalıştığı için hata verir. Örnek; n = 30 için 2. döngü de node[33] e bir komşuluk atmaya çalışıyor ama değerimiz 30 olduğu için hata verir.



Ekran Görüntüsü 2.4 node[37] için son durum



Ekran Görüntüsü 2.5 node[50] için ilk durum