T.C.
ONDOKUZ MAYIS ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BİL 465 Bilgisayar Ağ Yönetimi Laboratuvarı dersi Vize Projesi

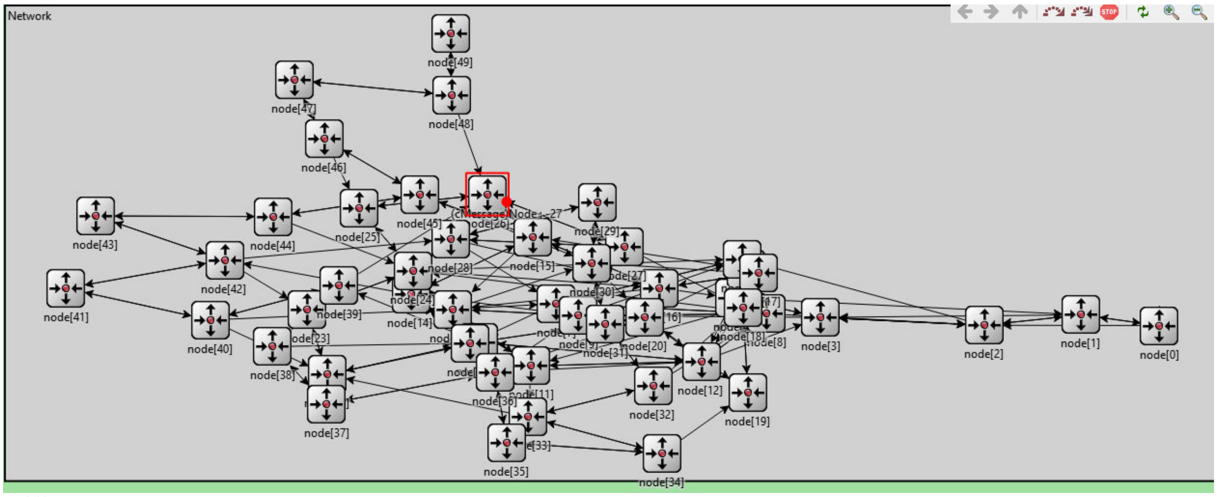
Rastgele oluşturulan 50 düğümlük ağ topolojisi üzerinde Dijkstra Shortest Path algoritmasının Omnet++ ağ benzetim ortamında gerçeklenmesi projesi

Ders Hocası: Doktor Öğretim Üyesi Sercan DEMİRCİ

Hazırlayan
14060292 – Tunahan Yetimoğlu
15060009 – Esra Gül

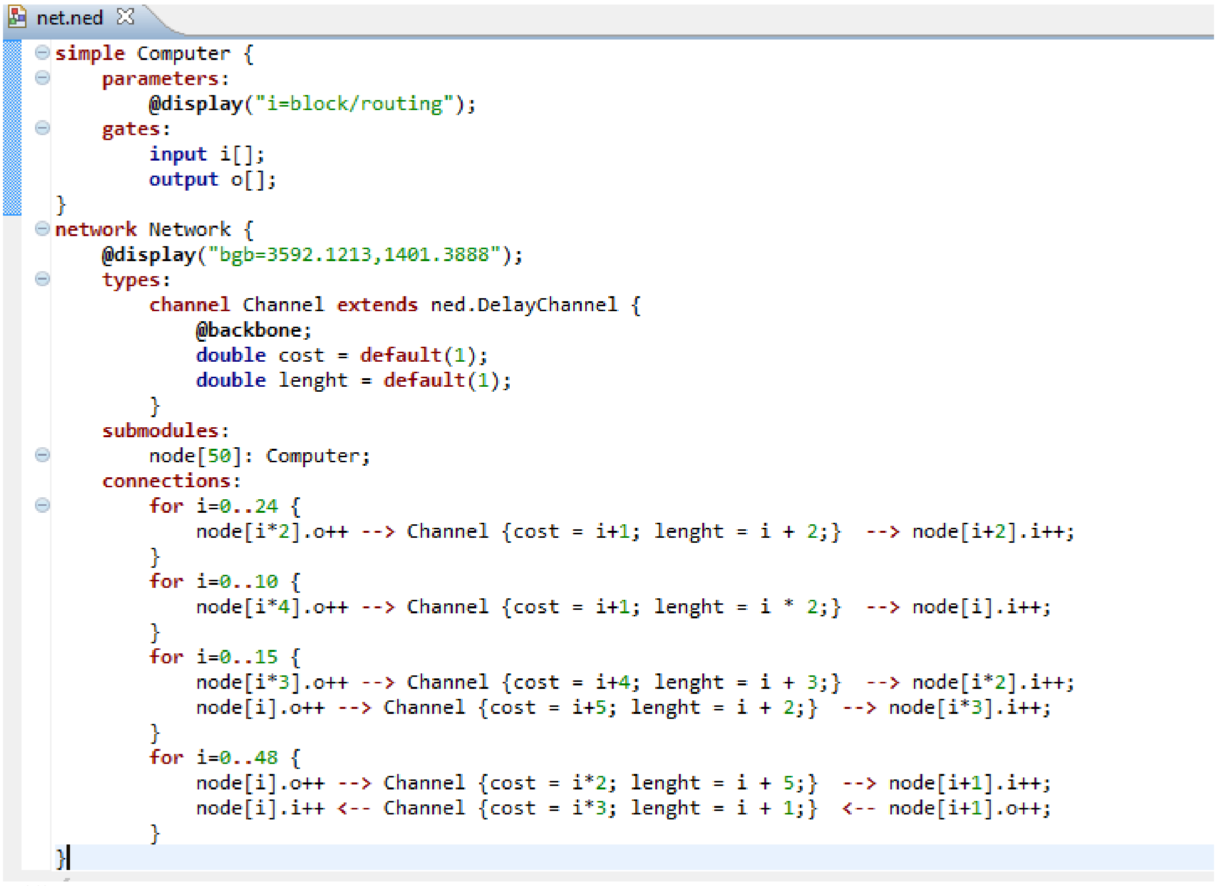
KASIM/2018
SAMSUN

Ağ Topolojisi



Şekil1.0

.Ned File

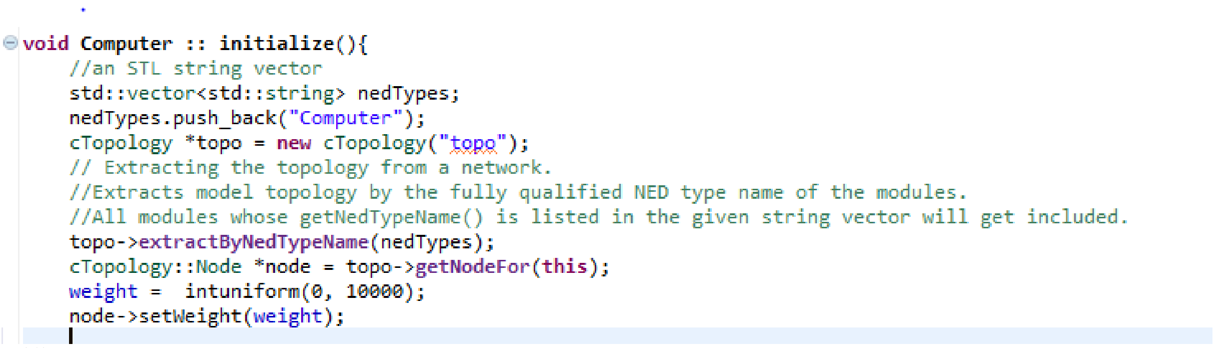


```
simple Computer {
    parameters:
        @display("i=block/routing");
    gates:
        input i[];
        output o[];
}
network Network {
    @display("bgb=3592.1213,1401.3888");
    types:
        channel Channel extends ned.DelayChannel {
            @backbone;
            double cost = default(1);
            double lenght = default(1);
        }
    submodules:
        node[50]: Computer;
    connections:
        for i=0..24 {
            node[i*2].o++ --> Channel {cost = i+1; lenght = i + 2;}  --> node[i+2].i++;
        }
        for i=0..10 {
            node[i*4].o++ --> Channel {cost = i+1; lenght = i * 2;}  --> node[i].i++;
        }
        for i=0..15 {
            node[i*3].o++ --> Channel {cost = i+4; lenght = i + 3;}  --> node[i*2].i++;
            node[i].o++ --> Channel {cost = i+5; lenght = i + 2;}  --> node[i*3].i++;
        }
        for i=0..48 {
            node[i].o++ --> Channel {cost = i*2; lenght = i + 5;}  --> node[i+1].i++;
            node[i].i++ <-- Channel {cost = i*3; lenght = i + 1;}  <-- node[i+1].o++;
        }
}
```

Şekil 1.1

Rastgele oluşturulması beklenen 50 düğümlük ağ topolojisini(Şekil 1.0) , .Ned dosyasındaki (Şekil 1.1) kod ile oluşturduk.Rastgele bağlantılar oluşturduk. Node'lara cost(maliyet) ve lenght(mesafe) değerleri atadık.

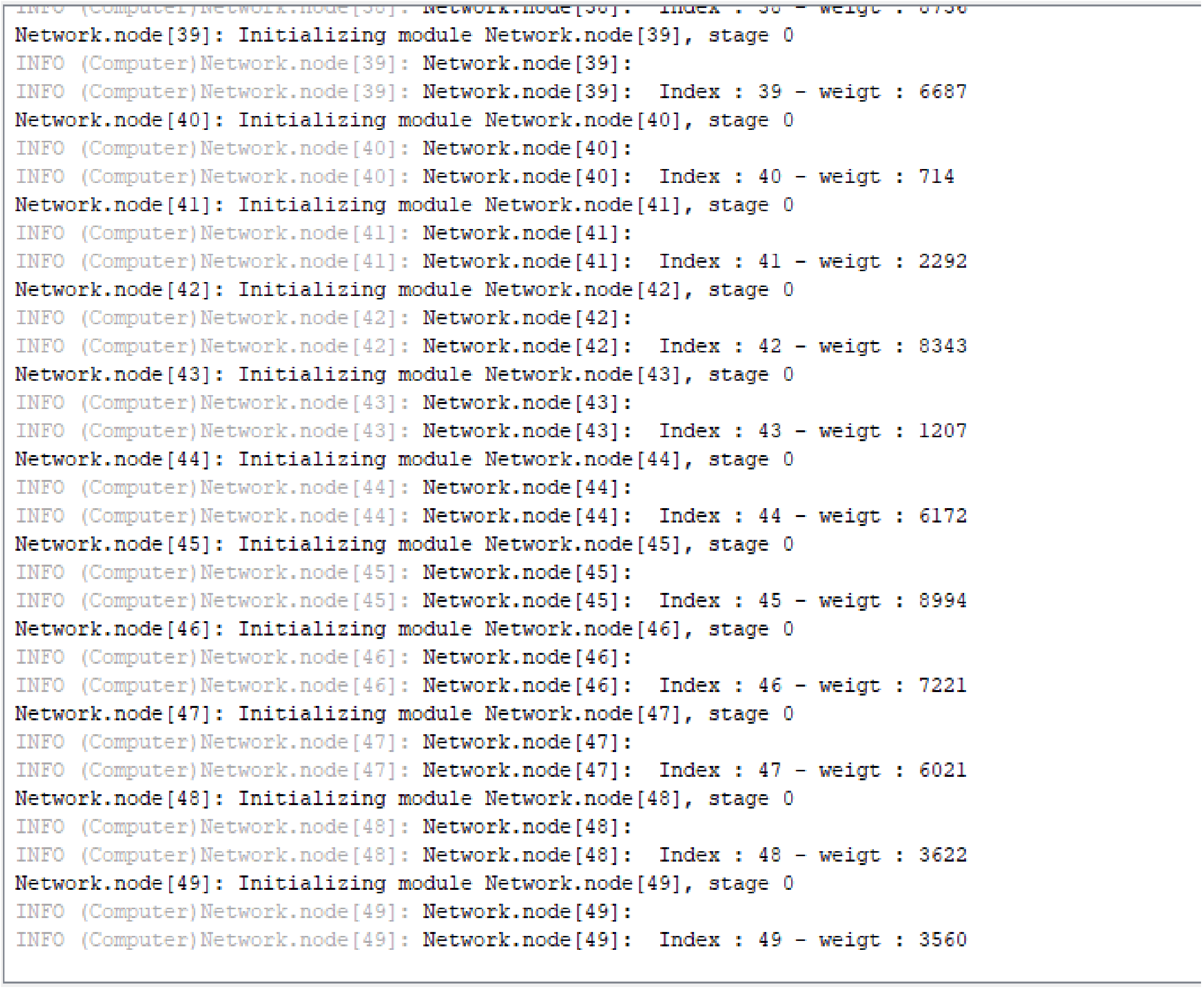
Source Code



```
void Computer :: initialize(){
    //an STL string vector
    std::vector<std::string> nedTypes;
    nedTypes.push_back("Computer");
    cTopology *topo = new cTopology("topo");
    // Extracting the topology from a network.
    //Extracts model topology by the fully qualified NED type name of the modules.
    //All modules whose getNedTypeName() is listed in the given string vector will get included.
    topo->extractByNedTypeName(nedTypes);
    cTopology::Node *node = topo->getNodeFor(this);
    weight =  intuniform(0, 10000);
    node->setWeight(weight);
```

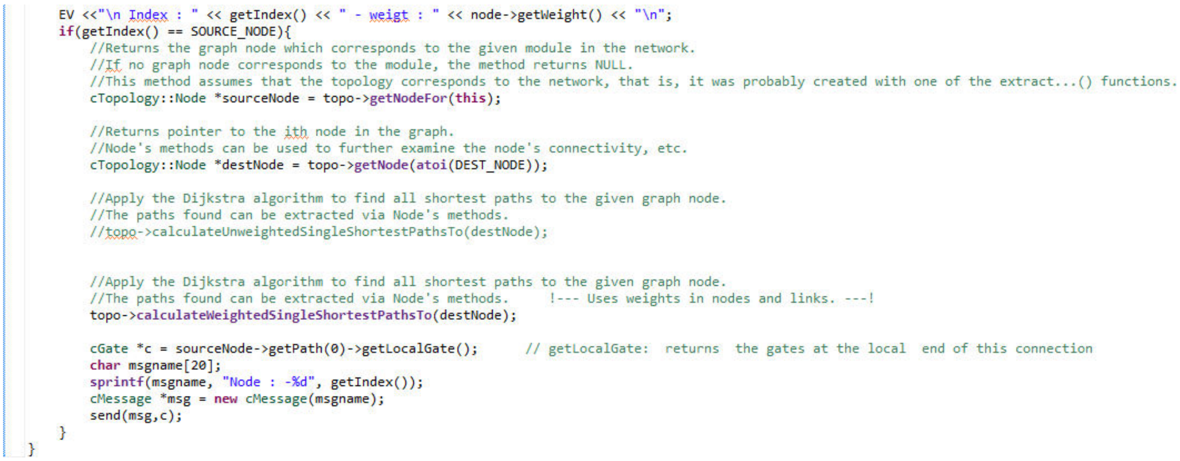
Şekil 1.2

Yukarıdaki kod parçası ile düğümlere(node) intuniform metodunu kullanarak rastgele ağırlık(weight) değerleri atadık.Rastgele atanmış ağırlık değerlerininin bir kısmını aşağıdaki şekilde (Şekil 1.3) görebiliriz.



```
INFO (Computer)Network.node[38]: Network.node[38]:  Index : 38 - weigt : [illegible]
Network.node[39]: Initializing module Network.node[39], stage 0
INFO (Computer)Network.node[39]: Network.node[39]:
INFO (Computer)Network.node[39]: Network.node[39]:  Index : 39 - weigt : 6687
Network.node[40]: Initializing module Network.node[40], stage 0
INFO (Computer)Network.node[40]: Network.node[40]:
INFO (Computer)Network.node[40]: Network.node[40]:  Index : 40 - weigt : 714
Network.node[41]: Initializing module Network.node[41], stage 0
INFO (Computer)Network.node[41]: Network.node[41]:
INFO (Computer)Network.node[41]: Network.node[41]:  Index : 41 - weigt : 2292
Network.node[42]: Initializing module Network.node[42], stage 0
INFO (Computer)Network.node[42]: Network.node[42]:
INFO (Computer)Network.node[42]: Network.node[42]:  Index : 42 - weigt : 8343
Network.node[43]: Initializing module Network.node[43], stage 0
INFO (Computer)Network.node[43]: Network.node[43]:
INFO (Computer)Network.node[43]: Network.node[43]:  Index : 43 - weigt : 1207
Network.node[44]: Initializing module Network.node[44], stage 0
INFO (Computer)Network.node[44]: Network.node[44]:
INFO (Computer)Network.node[44]: Network.node[44]:  Index : 44 - weigt : 6172
Network.node[45]: Initializing module Network.node[45], stage 0
INFO (Computer)Network.node[45]: Network.node[45]:
INFO (Computer)Network.node[45]: Network.node[45]:  Index : 45 - weigt : 8994
Network.node[46]: Initializing module Network.node[46], stage 0
INFO (Computer)Network.node[46]: Network.node[46]:
INFO (Computer)Network.node[46]: Network.node[46]:  Index : 46 - weigt : 7221
Network.node[47]: Initializing module Network.node[47], stage 0
INFO (Computer)Network.node[47]: Network.node[47]:
INFO (Computer)Network.node[47]: Network.node[47]:  Index : 47 - weigt : 6021
Network.node[48]: Initializing module Network.node[48], stage 0
INFO (Computer)Network.node[48]: Network.node[48]:
INFO (Computer)Network.node[48]: Network.node[48]:  Index : 48 - weigt : 3622
Network.node[49]: Initializing module Network.node[49], stage 0
INFO (Computer)Network.node[49]: Network.node[49]:
INFO (Computer)Network.node[49]: Network.node[49]:  Index : 49 - weigt : 3560
```

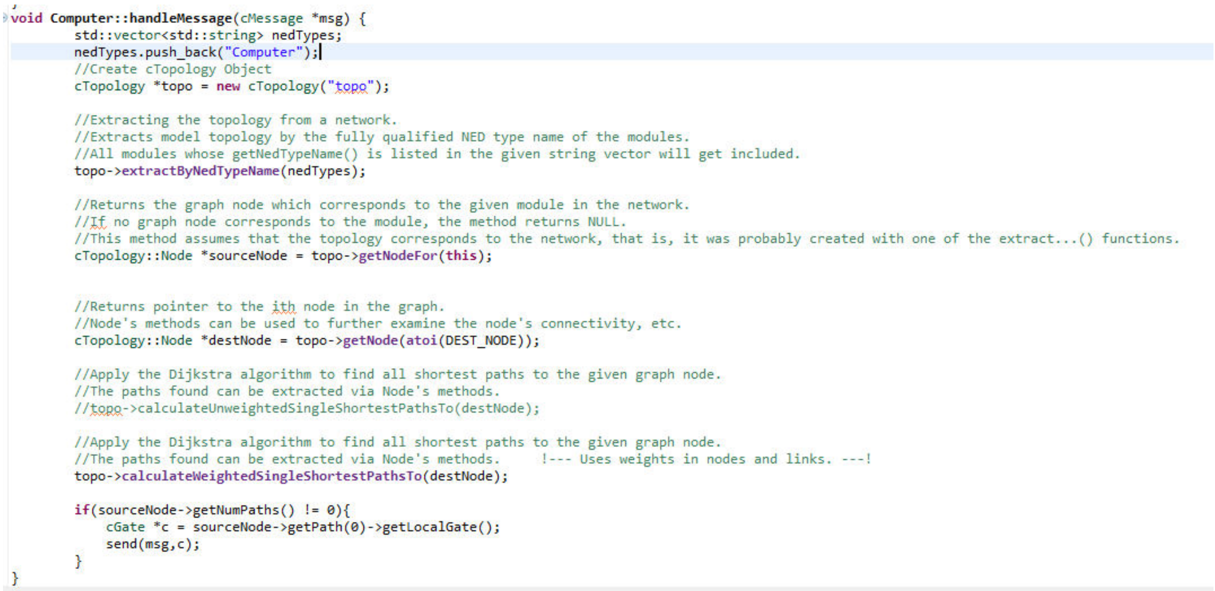
Şekil1.3



```
        EV <<"\n Index : " << getIndex() << " - weigt : " << node->getWeight() << "\n";
        if(getIndex() == SOURCE_NODE){
            //Returns the graph node which corresponds to the given module in the network.
            //If no graph node corresponds to the module, the method returns NULL.
            //This method assumes that the topology corresponds to the network, that is, it was probably created with one of the extract...() functions.
            cTopology::Node *sourceNode = topo->getNodeFor(this);

            //Returns pointer to the ith node in the graph.
            //Node's methods can be used to further examine the node's connectivity, etc.
            cTopology::Node *destNode = topo->getNode(atoi(DEST_NODE));

            //Apply the Dijkstra algorithm to find all shortest paths to the given graph node.
            //The paths found can be extracted via Node's methods.
            //topo->calculateUnweightedSingleShortestPathsTo(destNode);


            //Apply the Dijkstra algorithm to find all shortest paths to the given graph node.
            //The paths found can be extracted via Node's methods.     !--- Uses weights in nodes and links. ---!
            topo->calculateWeightedSingleShortestPathsTo(destNode);

            cGate *c = sourceNode->getPath(0)->getLocalGate();      // getLocalGate:  returns  the gates at the local  end of this connection
            char msgname[20];
            sprintf(msgname, "Node : -%d", getIndex());
            cMessage *msg = new cMessage(msgname);
            send(msg,c);
        }
    }
```

Şekil 1.4

Dijkstra Algoritmasının amacı, en az maliyetli yoldan hedefe ulaşmaktır. Biz projemizde Omnet++ ın bize sağladığı özellikleri ve metodları kullanarak bunu gerçekleştirdik. Oluşturduğumuz ağ topolojisini cTopology nesnesine atayıp bunun üzerinden işlem yaptık. Bu cTopology'nin içerdiği "**calculateWeightSingleShortestPathsTo** " methodundan faydalandık. (Şekil 1.4)Bu method kaynak düğümden hedef düğüme olan en az maliyetli yolu bize geri döndürüyor. Hop sayısına bakmıyor.

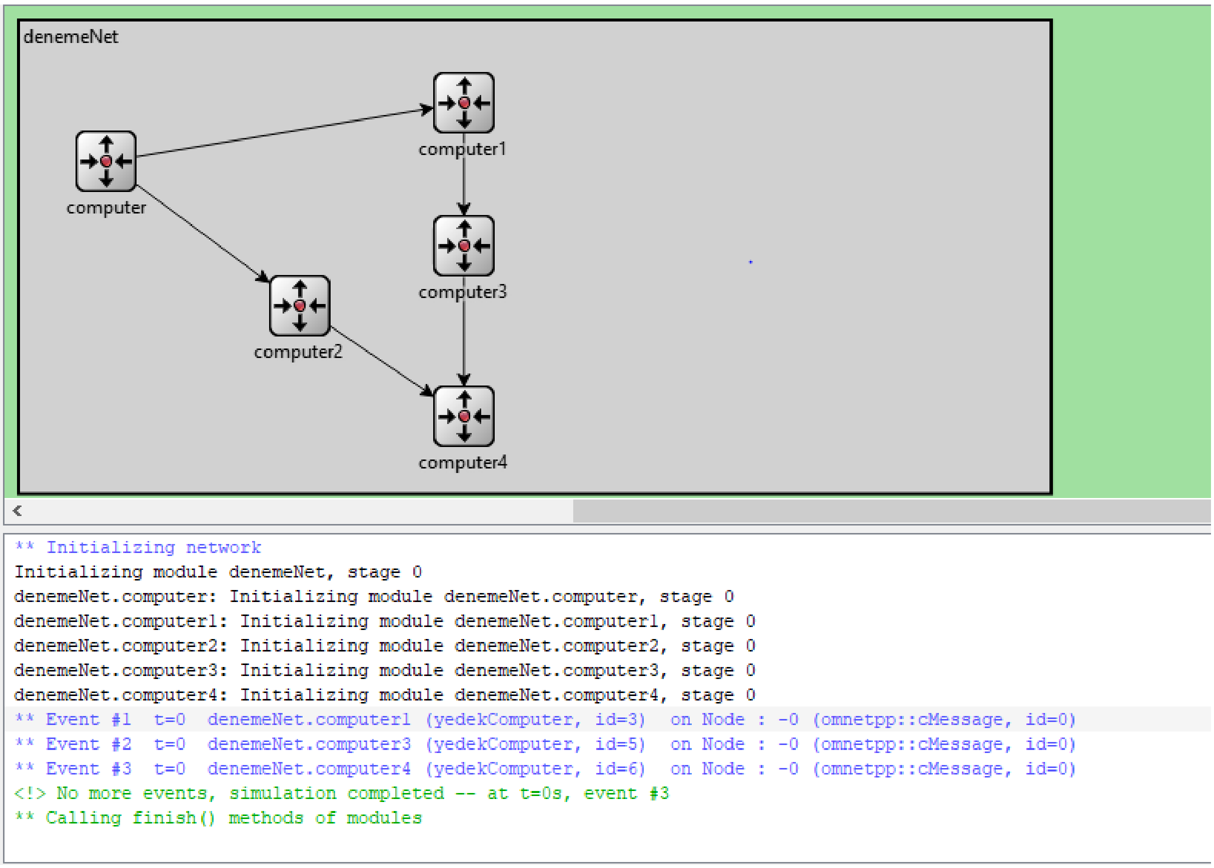


```
void Computer::handleMessage(cMessage *msg) {
        std::vector<std::string> nedTypes;
        nedTypes.push_back("Computer");
        //Create cTopology Object
        cTopology *topo = new cTopology("topo");

        //Extracting the topology from a network.
        //Extracts model topology by the fully qualified NED type name of the modules.
        //All modules whose getNedTypeName() is listed in the given string vector will get included.
        topo->extractByNedTypeName(nedTypes);

        //Returns the graph node which corresponds to the given module in the network.
        //If no graph node corresponds to the module, the method returns NULL.
        //This method assumes that the topology corresponds to the network, that is, it was probably created with one of the extract...() functions.
        cTopology::Node *sourceNode = topo->getNodeFor(this);


        //Returns pointer to the ith node in the graph.
        //Node's methods can be used to further examine the node's connectivity, etc.
        cTopology::Node *destNode = topo->getNode(atoi(DEST_NODE));

        //Apply the Dijkstra algorithm to find all shortest paths to the given graph node.
        //The paths found can be extracted via Node's methods.
        //topo->calculateUnweightedSingleShortestPathsTo(destNode);

        //Apply the Dijkstra algorithm to find all shortest paths to the given graph node.
        //The paths found can be extracted via Node's methods.     !--- Uses weights in nodes and links. ---!
        topo->calculateWeightedSingleShortestPathsTo(destNode);

        if(sourceNode->getNumPaths() != 0){
            cGate *c = sourceNode->getPath(0)->getLocalGate();
            send(msg,c);
        }
}
```

Şekil 1.5

Test Süreci

Bunu test edebilmek için küçük bir ağ topolojisi oluşturup rastgele olmayan değerler atadık(Şekil 1.6).



```
** Initializing network
Initializing module demeneNet, stage 0
demeneNet.computer: Initializing module demeneNet.computer, stage 0
demeneNet.computer1: Initializing module demeneNet.computer1, stage 0
demeneNet.computer2: Initializing module demeneNet.computer2, stage 0
demeneNet.computer3: Initializing module demeneNet.computer3, stage 0
demeneNet.computer4: Initializing module demeneNet.computer4, stage 0
** Event #1  t=0  demeneNet.computer1 (yedekComputer, id=3)  on Node : -0 (omnetpp::cMessage, id=0)
** Event #2  t=0  demeneNet.computer3 (yedekComputer, id=5)  on Node : -0 (omnetpp::cMessage, id=0)
** Event #3  t=0  demeneNet.computer4 (yedekComputer, id=6)  on Node : -0 (omnetpp::cMessage, id=0)
<!> No more events, simulation completed -- at t=0s, event #3
** Calling finish() methods of modules
```

Şekil 1.6

Yukarıdaki topoloji inceleyelim. Kaynak düğümümüz “computer”. Hedef düğümümüz ise “computer4”.Hop sayısına göre baksaydı computer – computer2 – compute4 şekilde ilerlemesi gerekiyordu.Ama kullandığımız method en az maliyetli yolu seçtiği için computer – computer1 computer3 – computer4 sırasında ilerlemiştir.

El ile atanan değer;
computer.setWeight(5);
computer1.setWeight(1);
computer2.setWeight(15);
computer3.setWeight(2);
computer4.setWeight(3);