



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

«Υλοποίηση Αλγορίθμου Chang-Roberts»

Για το μάθημα

Κατανεμημένα Δικτυοκεντρικά Συστήματα

Ονοματεπώνυμο: Γρίβας Πασχάλης

Εξάμηνο Φοίτησης: ΣΤ΄

A.M.: Π2017082

Περίληψη

Ο αλγόριθμος των Chang και Roberts αποτελεί μία βελτίωση του αλγορίθμου του LeLann για την εύρεση του μεγαλύτερου (ή του μικρότερου) σε ένα δίκτυο δακτυλίου (ring) από μοναδικά αριθμημένους κόμβους. Η διαδικασία αυτή είναι γνωστή και ως εκλογή αρχηγού κατά την οποία ορίζεται ένας κόμβος ως αρχηγός προκειμένου να εκτελέσει μία συγκεκριμένη λειτουργία (π.χ. λήψη μιας απόφασης). Η λειτουργία του αλγορίθμου βασίζεται στο γεγονός ότι ένας κόμβος ο οποίος επιθυμεί να γίνει αρχηγός στέλνει το μοναδικό αναγνωριστικό (id) του και αν το λάβει πίσω τότε εκλέγεται αρχηγός. Στην περίπτωση κατά την οποία ως αρχηγός ορίζεται ο κόμβος με το μικρότερο αναγνωριστικό, αν κάποιος κόμβος, ο οποίος επιθυμεί να γίνει αρχηγός, λάβει ένα id μικρότερο από το δικό του τότε το προωθεί και χάνει την εκλογή. Διαφορετικά, αν το id το οποίο έχει λάβει είναι μεγαλύτερο από το δικό του τότε το απορρίπτει και προωθεί το δικό του. Οι υπόλοιποι κόμβοι του δικτύου οι οποίοι δεν επιθυμούν να γίνουν αρχηγοί απλά προωθούν τα μηνύματα τα οποία λαμβάνουν. Έτσι, υπό αυτές τις συνθήκες, στην περίπτωση κατά την οποία όλοι οι κόμβοι του δικτύου επιθυμούν να γίνουν αρχηγοί και είναι διατεταγμένοι με αύξουσα σειρά, γίνονται συνολικά $n(n + 1)/2$ προωθήσεις (χειρότερη περίπτωση), ενώ όταν είναι διατεταγμένοι με φθίνουσα σειρά τότε γίνονται συνολικά $2n - 1$ προωθήσεις (καλύτερη περίπτωση), όπου n ο αριθμός των κόμβων του δικτύου. Στην μέση περίπτωση πραγματοποιούνται συνολικά $n \log n$ προωθήσεις.

Λέξεις Κλειδιά

Chang-Robert's algorithm, Κατανεμημένος αλγόριθμος, Αλγόριθμος εκλογής, Κατανεμημένα δικτυοκεντρικά συστήματα, Υλοποίηση αλγορίθμου, C++, OMNet++

Περιεχόμενα

| | |
|--|----|
| Περίληψη | 2 |
| Περιεχόμενα | 3 |
| 1. Υλοποίηση | 4 |
| 1.1 Περιγραφή της υλοποίησης του αλγορίθμου | 4 |
| 1.2 Διάγραμμα ροής του κώδικα | 8 |
| 2. Αποτελέσματα | 10 |
| 2.1 Παρουσίαση των προσομοιώσεων | 10 |
| 2.1.1 Δίκτυα με έναν initiator | 11 |
| 2.1.2 Δίκτυα με τυχαίους initiators | 12 |
| 2.1.3 Δίκτυα με όλους τους κόμβους initiators. | 13 |
| 2.1.4 Δίκτυα με δύο initiators | 14 |
| 2.2 Ερμηνεία των αποτελεσμάτων | 15 |
| 2.2.1 Κατεύθυνση των μηνυμάτων | 15 |
| 2.2.2 Καλύτερη και χειρότερη περίπτωση | 16 |
| Βιβλιογραφία | 17 |

Κεφάλαιο 1

Υλοποίηση

1.1 Περιγραφή της υλοποίησης του αλγορίθμου

Η υλοποίηση του αλγορίθμου έγινε χρησιμοποιώντας το περιβάλλον κατασκευής προσομοιώσεων δικτύων “OMNet++ 5.6.1”. Αρχικά δημιουργήθηκε ένα αντικείμενο Node (Node.net) τύπου simple module το οποίο αναπαριστά την δομή όλων των κόμβων. Παράλληλα, δημιουργήθηκε ένα αντικείμενο τύπου network, το οποίο αναπαριστά το δίκτυο της προσομοίωσης, με τοπολογία κόμβων σε μορφή δακτυλίου (Ring), και περιέχει πληροφορίες σχετικά με το πλήθος των κόμβων, τις συνδέσεις μεταξύ τους κ.τ.λ..

Εν συνεχεία, υλοποιήθηκε η λειτουργικότητα – συμπεριφορά του κάθε κόμβου σε γλώσσα προγραμματισμού C++ στα αρχεία Node.cc και Node.h τα οποία δημιουργήθηκαν κατά την δημιουργία του module Node. Το αρχείο Node.h περιλαμβάνει την δήλωση της κλάσης Node και την δήλωση των μεθόδων `initialize()` και `handleMessage(cMessage *msg)`, οι οποίες αντιπροσωπεύουν τις καταστάσεις στις οποίες μεταβαίνει κάθε κόμβος, δηλαδή αρχικοποίηση και λήψη μηνύματος. Το αρχείο Node.cc περιέχει την υλοποίηση των παραπάνω μεθόδων.

Πιο συγκεκριμένα, στη μέθοδο `initialize()` κάθε κόμβος αρχικοποιεί τα μέλη δεδομένων του (π.χ. id) με τιμές οι οποίες υπάρχουν στα .ned αρχεία. Επίσης, ο κόμβος ο οποίος έχει οριστεί ως εκκινητής και οι κόμβοι οι οποίοι επιθυμούν να εκλεχθούν αρχηγοί, ανάλογα με τις παραμετροποιήσεις του χρήστη, γίνονται κόκκινοι και στέλνουν ένα μήνυμα στον εαυτό τους.

```
void Node::initialize()
{
    id = par("id");
    sizeX = int(getParentModule()->par("nodeSizeX"));
    sizeY = int(getParentModule()->par("nodeSizeY"));
    bool oneTokenMode = bool(getParentModule()->par("oneTokenMode"));
    numberOfGates = gateSize("inoutGateVector");
    myDispStr = this->getDisplayString();

    //if current node is the initiator - wants to be leader
    if(id == int(getParentModule()->par("initiator")))
    {
        myDispStr.setTagArg("b",0,sizeX);
        myDispStr.setTagArg("b",1,sizeY);
        myDispStr.setTagArg("b",3,"red");
        this->setDisplayString(myDispStr.str()); //make node red

        //send a self message
        scheduleAt(simTime()+1,new cMessage("I want to be the leader!"));
        iWantToBeLeader = true;
    }
}
```

```

else
{
    double probabilityToBeLeader = double(getParentModule()->par("probabilityToBeLeader"));

    if(probabilityToBeLeader == 0)
    {
        //no one wants to become leader - except initiator
        iWantToBeLeader = false;
    }
    else if(probabilityToBeLeader == 0.5)
    {
        //there is a 50% probability of the nodes to want the leadership
        if(intuniform(0,1) == 1)
        {
            ...
            this->setDisplayString(myDispStr.str()); //make node red

            if(!oneTokenMode)
                scheduleAt(simTime()+1,new cMessage("I want to be the leader!"));
            iWantToBeLeader = true;
        }
    }
    else if(probabilityToBeLeader == 1)
    {
        //everyone wants to become leaders
        ...
        this->setDisplayString(myDispStr.str()); //make node red

        if(!oneTokenMode)
            //send a self message
            scheduleAt(simTime()+1,new cMessage("I want to be the leader!"));
        iWantToBeLeader = true;
    }
    else if(probabilityToBeLeader == 2)
    {
        //the node before initiator wants to become leader
        if(this->id == int(getParentModule()->par("initiator")) - 1)
        {
            ...
            this->setDisplayString(myDispStr.str()); //make node red

            if(!oneTokenMode)
                //send a self message
                scheduleAt(simTime()+1,new cMessage("I want to be the leader!"));

            iWantToBeLeader = true;
        }
    }
    else if(probabilityToBeLeader == 3)
    {
        //the node after initiator wants to become leader
        if(this->id == int(getParentModule()->par("initiator")) + 1)
        {
            ...
            this->setDisplayString(myDispStr.str()); //make node red

            if(!oneTokenMode)
                //send a self message
                scheduleAt(simTime()+1,new cMessage("I want to be the leader!"));
            iWantToBeLeader = true;
        }
    }
}
}
}

```

Στη μέθοδο **handleMessage(cMessage *msg)**, η οποία εκτελείται κάθε φορά που ένας κόμβος λαμβάνει μήνυμα, ελέγχεται αν το μήνυμα το οποίο έχει λάβει ο κάθε κόμβος προέρχεται από τον εαυτό του (self message). Αν αυτή η συνθήκη ισχύει τότε ο κόμβος επιθυμεί να γίνει αρχηγός και στέλνει το αναγνωριστικό του στον γείτονα δεξιά ή αριστερά του, ανάλογα με το πως το έχει ορίσει ο χρήστης. Διαφορετικά, αν το μήνυμα προέρχεται από κάποιον άλλο κόμβο, τότε ελέγχεται αν ο τρέχον κόμβος επιθυμεί να γίνει αρχηγός.

Στην περίπτωση κατά την οποία ο κόμβος έχει λάβει ένα μήνυμα από τον γείτονά του και δεν επιθυμεί να γίνει αρχηγός, προωθεί το μήνυμα στον επόμενο γείτονα του. Αντιθέτως, αν και ο τρέχον κόμβος επιθυμεί να εκλεχθεί, τότε ελέγχει την τιμή του μηνύματος – χαρακτηριστικού το οποίο έχει λάβει και τη συγκρίνει με το δικό του αναγνωριστικό. Έτσι προκύπτουν οι εξής τρεις περιπτώσεις:

(α) Αν το id του τρέχοντα κόμβου είναι μεγαλύτερο από το id το οποίο έχει λάβει, τότε αυτό σημαίνει ότι υπάρχει κάποιος άλλος κόμβος με μικρότερο id ο οποίος ενδέχεται να γίνει αρχηγός. Επομένως, ο τρέχον κόμβος χάνει την εκλογή, αλλάζει το χρώμα του σε μαύρο και προωθεί το μήνυμα στον επόμενο γείτονά του.

(β) Αλλιώς αν το id του τρέχοντα κόμβου είναι μικρότερο από το id το οποίο έχει λάβει, τότε αυτό σημαίνει ότι ο τρέχον κόμβος είναι πιθανό να γίνει αρχηγός. Έτσι, απορρίπτει το μήνυμα το οποίο έχει λάβει και στέλνει στον επόμενο γείτονά του το δικό του id αν δεν το έχει ήδη κάνει.

(γ) Αλλιώς αν το id του τρέχοντα κόμβου είναι ίσο με το id το οποίο έχει λάβει, τότε ο τρέχον κόμβος γίνεται αρχηγός μιας και το αναγνωριστικό του δεν έχει απορριφθεί από κανέναν άλλον κόμβο – έχει την μικρότερη τιμή. Έτσι, ο κόμβος αλλάζει το χρώμα του σε πράσινο και η προσομοίωση τερματίζει.

```
void Node::handleMessage(cMessage *msg)
{
    int senderId; //the node who send me the msg
    int msgcontent; //the content of the msg converted to int
    int currentNeighborId; //the neighbor node that i am checking in each loop
    int i;

    bubble(msg->getName());

    if(msg->isSelfMessage()) //if msg is a self message from scheduleAt
    {
        std::string newMsgString = std::to_string(this->id); //convert int id to string

        //convert string to char array and create cMessage
        int n = newMsgString.length();
        char newMsgCharArray[n + 1];
        strcpy(newMsgCharArray, newMsgString.c_str());
        cMessage *message = new cMessage(newMsgCharArray);

        if(bool(getParentModule()->par("clockwiseDirection"))) //send your id to your clockwise neighbor
        {
            if(this->id == 1000)
                send(message->dup(), gate("inoutGateVector$0", 0));
            else
                send(message->dup(), gate("inoutGateVector$0", 1));
        }
        else
        {
            if(this->id == 1000)
                send(message->dup(), gate("inoutGateVector$0", 1));
            else
                send(message->dup(), gate("inoutGateVector$0", 0));
        }

        this->iHaveSendMyToken = true;
        delete message;
    }
}
```

```

else //if msg is not a self message
{
    senderId = ((Node*)msg->getSenderModule())->id; //get the id of the sender

    if(iWantToBeLeader) //if i want to become leader
    {
        msgcontent = std::stoi(msg->getName()); //convert msg to int

        if(msgcontent == this->id) //the msg contains my id - i received my id back - i have the
                                                                    smallest id
        {
            bubble("I am the leader");
            ...
            this->setDisplayString(myDispStr.str()); //make node green
        }
        else if (msgcontent < this->id) //there is a node with smaller id
        {
            for(i=0;i<numberOfGates;i++)
            {
                //get the id of the node at the current gate
                currentNeighborId = ((Node*)gate("inoutGateVector$o",i)->getPathEndGate()
                                                                    ->getOwnerModule())->id;

                if(currentNeighborId != senderId)
                {
                    bubble("I lost :(");
                    ...
                    this->setDisplayString(myDispStr.str()); //make node black

                    send(msg->dup(),gate("inoutGateVector$o",i)); //just forward the msg
                }
            }
        }
        else //my id is smaller than the one in the token
        {
            std::string newMsgString = std::to_string(this->id); //convert int id to string

            //convert string to char array and create cMessage
            int n = newMsgString.length();
            char newMsgCharArray[n + 1];
            strcpy(newMsgCharArray, newMsgString.c_str());
            cMessage *message = new cMessage(newMsgCharArray);

            for(i=0;i<numberOfGates;i++)
            {
                //get the id of the node at the current gate
                currentNeighborId = ((Node*)gate("inoutGateVector$o",i)->getPathEndGate()
                                                                    ->getOwnerModule())->id;

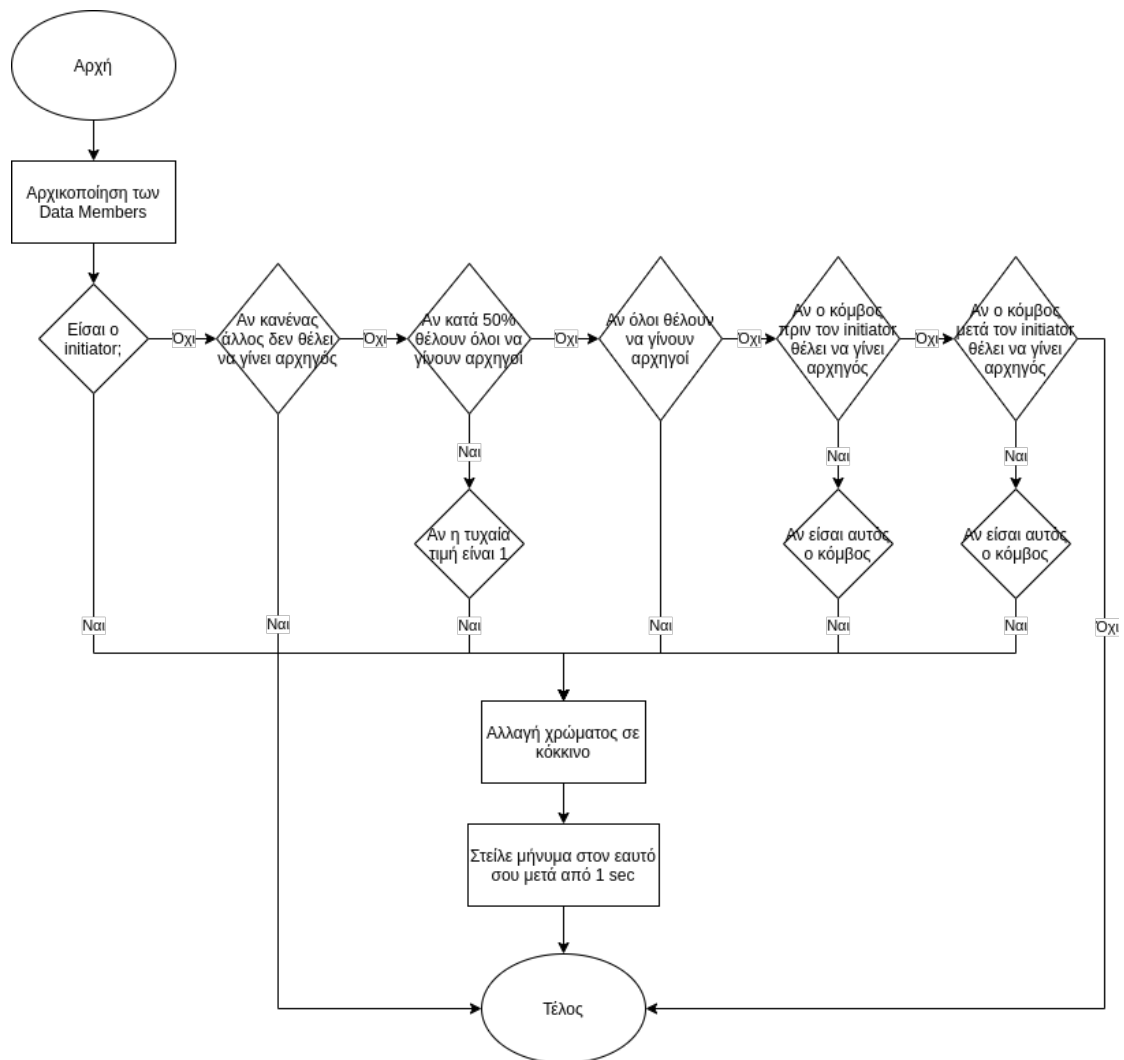
                if(currentNeighborId != senderId)
                {
                    if(!iHaveSendMyToken) //if i already sent my token
                    {
                        //send my id
                        send(message->dup(),gate("inoutGateVector$o",i));
                        delete message;
                    }
                }
            }
        }
    }
}
else //if i don't want to become leader
{
    for(i=0;i<numberOfGates;i++)
    {
        //get the id of the node at the current gate
        currentNeighborId = ((Node*)gate("inoutGateVector$o",i)->getPathEndGate()
                                                                    ->getOwnerModule())->id;

        if(currentNeighborId != senderId)
            send(msg->dup(),gate("inoutGateVector$o",i)); //just forward the msg
    }
}
delete msg;
}

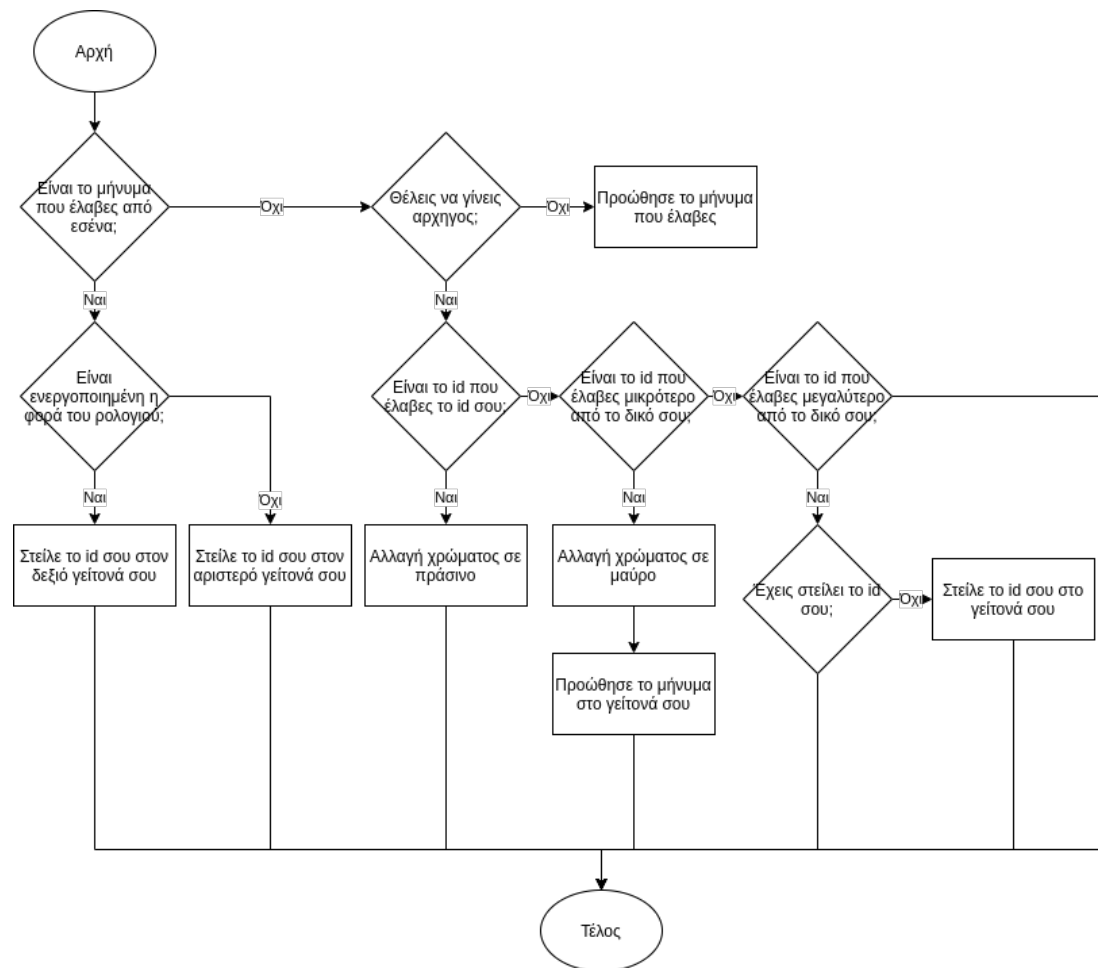
```

1.2 Διάγραμμα ροής του κώδικα

Στα παρακάτω σχήματα παρουσιάζεται η διαδικασία την οποία ακολουθεί το πρόγραμμα προκειμένου να προσομοιωθεί ο αλγόριθμος εκλογής. Πιο συγκεκριμένα, φαίνονται οι αποφάσεις τις οποίες κάθε κόμβος παίρνει κατά την αρχικοποίηση του (σχήμα 1) και όταν λαμβάνει ένα μήνυμα (σχήμα 2). Τα σχήματα αποτελούν την διαγραμματική απεικόνιση του κώδικα ο οποίος αναλύθηκε στην ενότητα 1.1.



Σχήμα 1: Διάγραμμα Ροής - *initialize()*



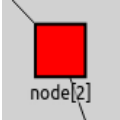

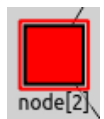

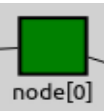
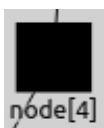

Σχήμα 2: Διάγραμμα Ροής - **handleMessage(cMessage *msg)**

Κεφάλαιο 2

Αποτελέσματα

2.1 Παρουσίαση των προσομοιώσεων

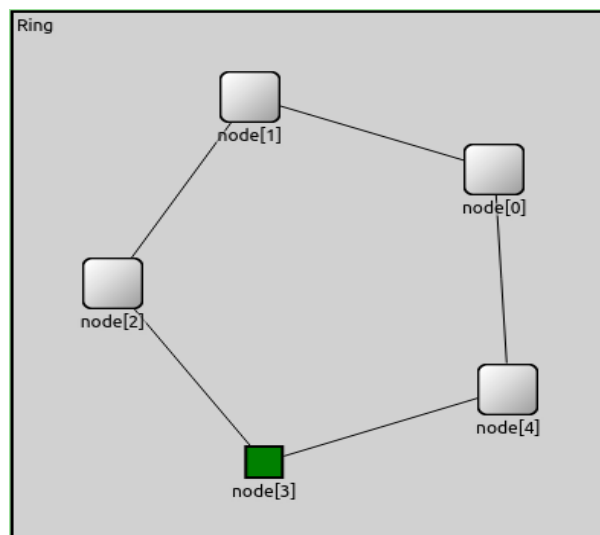
Προκειμένου να εκτελεστεί ο κώδικας ο οποίος αναλύθηκε στο κεφάλαιο 1 είναι απαραίτητο να έχει επιλεχθεί το αρχείο omnetpp.ini το οποίο περιέχει τις ρυθμίσεις των παραμέτρων όλων των δικτύων τα οποία έχουν υλοποιηθεί. Επίσης, δίνεται ο παρακάτω πίνακας ο οποίος περιλαμβάνει τους χρωματισμούς και την ερμηνεία του καθενός, των στοιχείων του γραφικού αποτελέσματος της προσομοίωσης:

| Απεικόνιση | Στοιχείο | Ερμηνεία |
|---|--------------------------------|---|
|  | Κόμβος Εκκινητής – Initiator | Έχει αρχικοποιηθεί - θέλει να εκλεχθεί αρχηγός και στέλνει το id του. |
|  | Κόμβος (non initiator) | Αδρανής κόμβος του δικτύου – προωθεί τα μηνύματα που λαμβάνει. |
|  | Ενεργός Κόμβος (initiator) | Ενεργός κόμβος – εκκινητής - συγκρίνει τα id που λαμβάνει με το δικό του. |
|  | Ενεργός Κόμβος (non initiator) | Ενεργός κόμβος του δικτύου – έχει λάβει μήνυμα και το προωθεί. |
|  | Κόμβος Αρχηγός | Έχει το μικρότερο id, το έχει λάβει πίσω και έχει εκλεχθεί αρχηγός. |
|  | Χαμένος Κόμβος | Θέλει να γίνει αρχηγός αλλά έχει λάβει ένα id μικρότερο από το δικό του και χάνει την εκλογή. |
|  | Μήνυμα - Token | - |

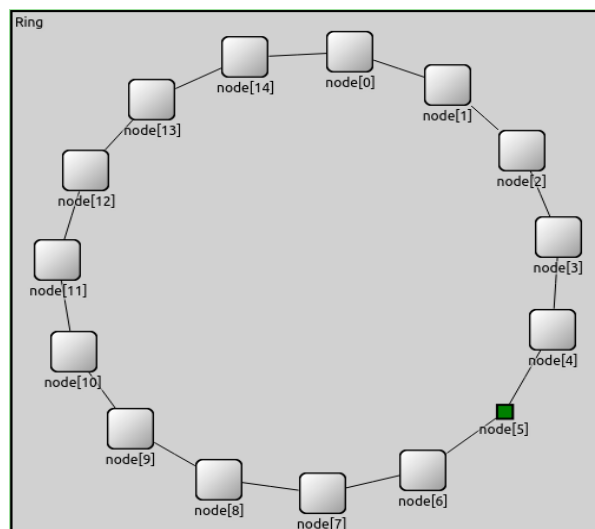
Πίνακας 1: Ερμηνεία των στοιχείων της προσομοίωσης

2.1.1 Δίκτυα με έναν initiator

Στα σχήματα παρακάτω φαίνεται η εκτέλεση του αλγορίθμου σε δίκτυα δακτυλίου 5 και 15 κόμβων αντίστοιχα. Στα δίκτυα αυτά ορίστηκε κάθε φορά ένας κόμβος εκκινήτης, ο οποίος και επιθυμεί να γίνει αρχηγός, και όλοι οι υπόλοιποι κόμβοι να μη συμμετάσχουν στην εκλογή. Αυτό επιτεύχθηκε επιλέγοντας, κατά την έναρξη της προσομοίωσης, η τιμή της μεταβλητής “`prob`” να είναι μηδέν. Στο σχήμα 3 κόμβος εκκινήτης είναι ο 1003 ενώ στο σχήμα 4 ο 1005. Στο δίκτυο του πρώτου σχήματος έγιναν συνολικά 5 και στο δίκτυο του δεύτερου 15 προωθήσεις του μηνύματος μεταξύ των κόμβων έως τον τερματισμό του αλγορίθμου, ανεξάρτητα από την κατεύθυνση μετάβασης του μηνύματος – τιμή της μεταβλητής “`clockwiseDirection`”.



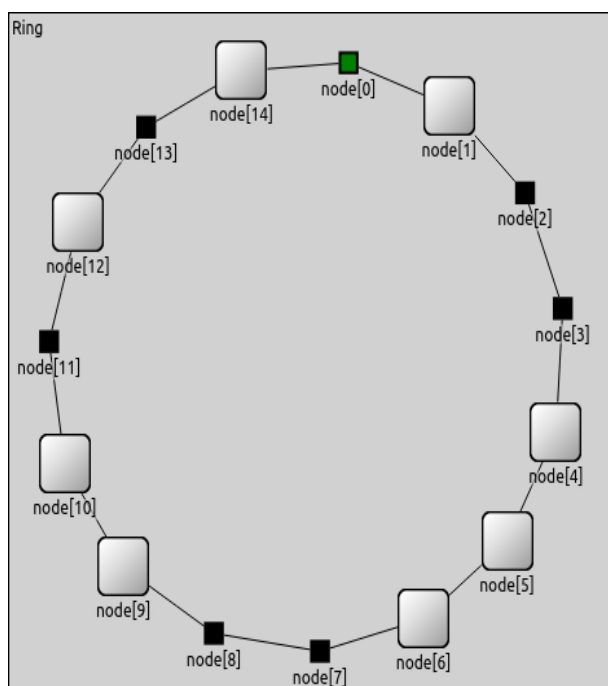
Σχήμα 3: Δακτύλιος - N:5 Init:1003 Prob:0



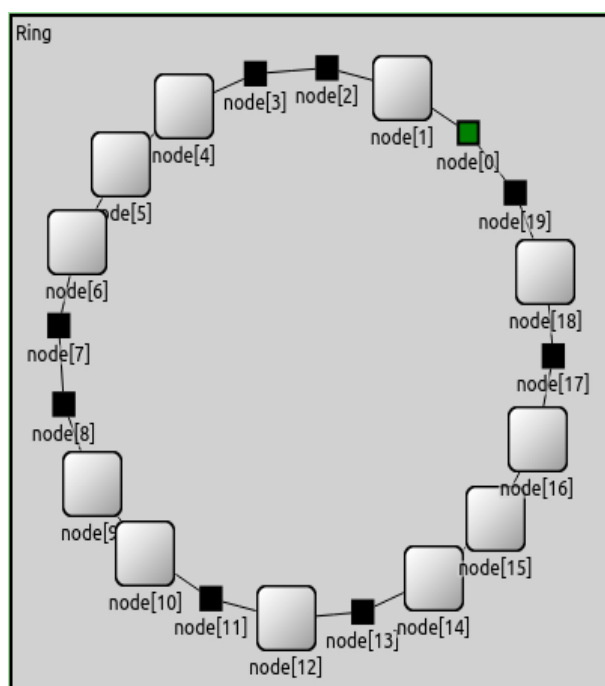
Σχήμα 4: Δακτύλιος - N:15 Init:1005 Prob:0

2.1.2 Δίκτυα με τυχαίους initiators

Κατά την έναρξη της προσομοίωσης αν οριστεί η τιμή της μεταβλητής “prob” σε 0.5, δημιουργείται δίκτυο στο οποίο κάθε κόμβος θέλει να γίνει αρχηγός με πιθανότητα 50%. Έτσι, δημιουργήθηκαν δύο ring δίκτυα με 15 (σχήμα 5) και 20 (σχήμα 6) κόμβους αντίστοιχα, στα οποία ορίστηκαν εκκινήτες ο κόμβος 1008 και τυχαία κάποιοι από τους υπόλοιπους. Στο πρώτο δίκτυο επιλέγοντας τα μηνύματα να κατευθύνονται από τον μικρότερο στον μεγαλύτερο κόμβο (`clockwiseDirection=true`), έγιναν συνολικά 61 προωθήσεις των μηνυμάτων ενώ αντίστροφα (`clockwiseDirection=false`), έγιναν 28. Αντίστοιχα, στο δεύτερο δίκτυο επιλέγοντας τα μηνύματα να κατευθύνονται από τον μικρότερο στον μεγαλύτερο κόμβο (`clockwiseDirection=true`), έγιναν συνολικά 100 προωθήσεις των μηνυμάτων ενώ αντίστροφα (`clockwiseDirection=false`), έγιναν 39. Σε κάθε περίπτωση, αρχηγός εκλέχθηκε ο κόμβος 1000 μιας και ήθελε να γίνει αρχηγός και είχε το μικρότερο αναγνωριστικό.



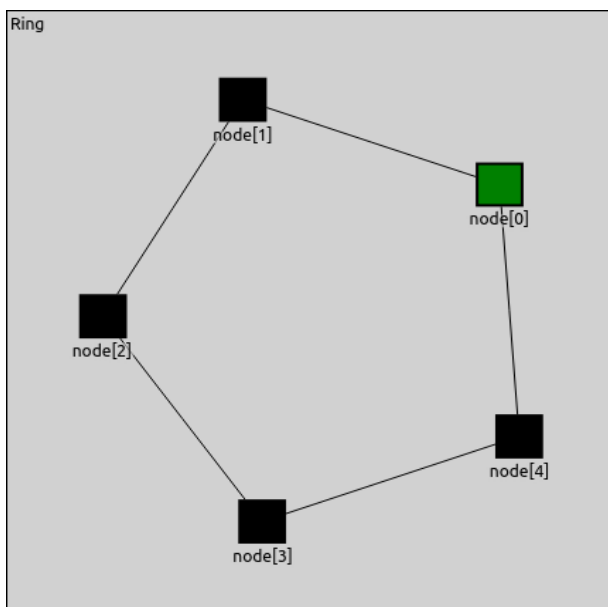
Σχήμα 5: Δακτύλιος - N:15 Init:1008 Prob:0.5



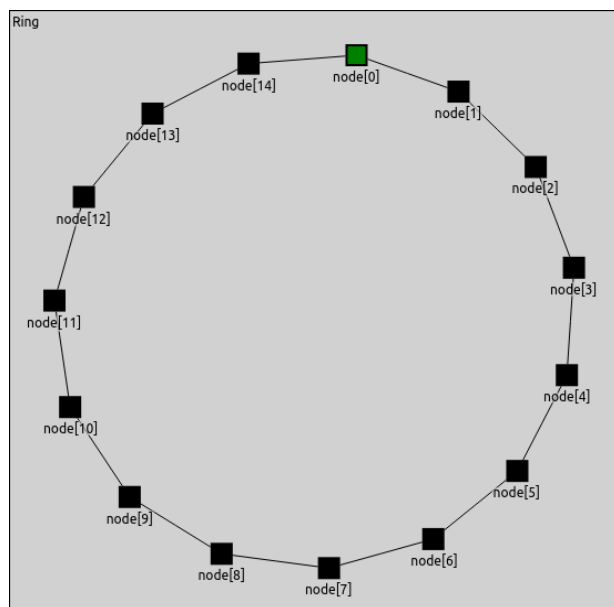
Σχήμα 6: Δακτύλιος - N:20 Init:1008 Prob:0.5

2.1.3 Δίκτυα με όλους τους κόμβους initiators

Αντίστοιχα, στην έναρξη της προσομοίωσης αν οριστεί η τιμή της μεταβλητής “**prob**” σε 1, δημιουργείται δίκτυο στο οποίο όλοι οι κόμβοι του θέλουν να εκλεχθούν αρχηγοί. Έτσι, δημιουργήθηκαν δύο ring δίκτυα με 5 (σχήμα 7) και 15 (σχήμα 8) κόμβους αντίστοιχα, στα οποία ορίστηκαν εκκινητές όλοι οι κόμβοι. Στο πρώτο δίκτυο επιλέγοντας τα μηνύματα να κατευθύνονται από τον μικρότερο στον μεγαλύτερο κόμβο (**clockwiseDirection=true**), έγιναν συνολικά 15 προωθήσεις των μηνυμάτων ενώ αντίστροφα (**clockwiseDirection=false**), έγιναν 9. Αντίστοιχα, στο δεύτερο δίκτυο επιλέγοντας τα μηνύματα να κατευθύνονται από τον μικρότερο στον μεγαλύτερο κόμβο (**clockwiseDirection=true**), έγιναν συνολικά 120 προωθήσεις των μηνυμάτων ενώ αντίστροφα (**clockwiseDirection=false**), έγιναν 29. Σε κάθε περίπτωση, αρχηγός εκλέχθηκε ο κόμβος 1000 καθώς είχε το μικρότερο αναγνωριστικό.



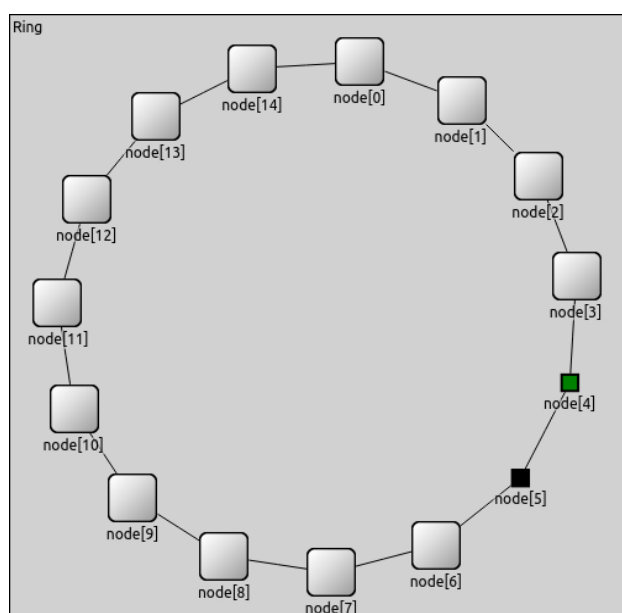
Σχήμα 7: Δακτύλιος - N:5 Init:all Prob:1



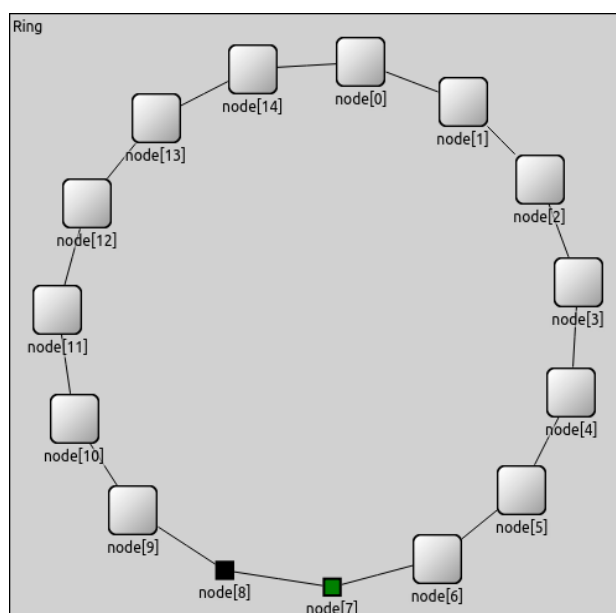
Σχήμα 8: Δακτύλιος - N:15 Init:all Prob:1

2.1.4 Δίκτυα με δύο initiators

Τέλος, κατά την έναρξη της προσομοίωσης, υπάρχουν οι επιλογές η μεταβλητή “**prob**” να πάρει τις τιμές 2 και 3. Στην πρώτη περίπτωση (σχήμα 9), δημιουργείται ένα ring δίκτυο στο οποίο θέλουν να γίνουν αρχηγοί ο κόμβος ο οποίος έχει οριστεί ως εκκινήτης (1005) και ο κόμβος πριν από αυτόν. Στην δεύτερη περίπτωση (σχήμα 10), οι κόμβοι οι οποίο επιθυμούν να εκλεχθούν αρχηγοί είναι ο κόμβος ο οποίος έχει οριστεί ως εκκινήτης (1007) και ο κόμβος μετά από αυτόν. Ανεξάρτητα από αυτήν την παράμετρο και στις δύο περιπτώσεις όταν τα μηνύματα κατευθύνονται από τον μικρότερο στον μεγαλύτερο κόμβο (**clockwiseDirection=true**), γίνονται συνολικά 29 προωθήσεις ενώ αντίστροφα (**clockwiseDirection=false**), γίνονται 16. Επίσης, σε κάθε προσομοίωση, εκλέγεται αρχηγός ο κόμβος με το μικρότερο id δηλαδή ο 1004 και ο 1007 αντίστοιχα.



Σχήμα 9: Δακτύλιος - N:15 Init:1005 Prob:2



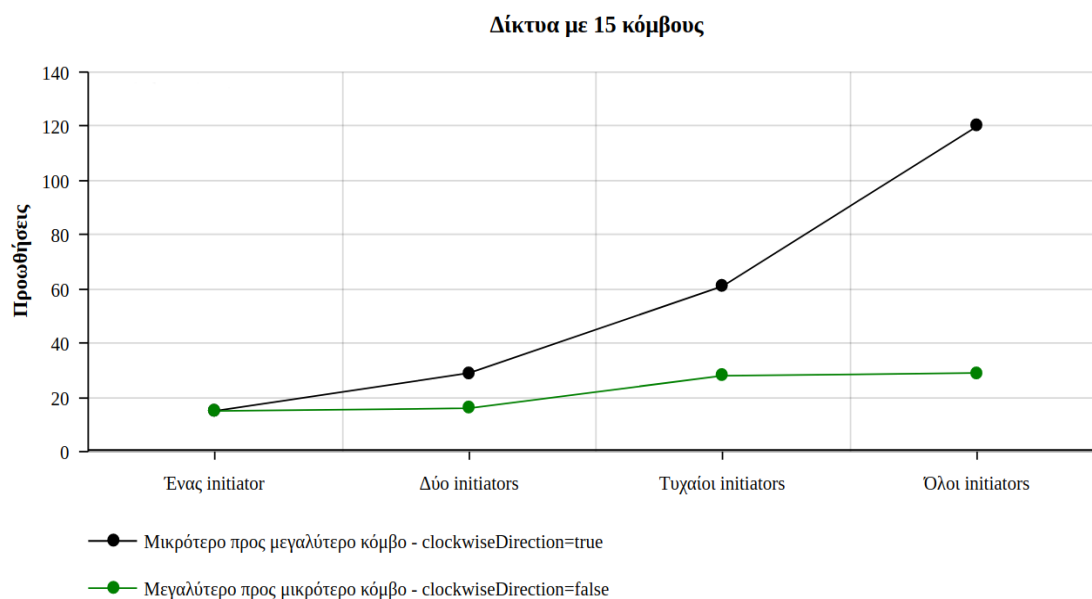
Σχήμα 10: Δακτύλιος - N:15 Init:1007 Prob:3

2.2 Ερμηνεία των αποτελεσμάτων

Τα αποτελέσματα των προσομοιώσεων, τα οποία παρουσιάστηκαν στην προηγούμενη ενότητα, αποδεικνύουν ότι στον αλγόριθμο των Chang και Roberts, ο αριθμός των προωθήσεων των μηνυμάτων, εξαρτάται από το πλήθος όλων των κόμβων του δικτύου, από το πλήθος και την θέση των κόμβων, οι οποίοι διεκδικούν την αρχηγία και από την κατεύθυνση με την οποία προωθούνται τα μηνύματα μέσα στο δίκτυο.

2.2.1 Κατεύθυνση των μηνυμάτων

Ένα συμπέρασμα το οποίο προκύπτει, όταν τα μηνύματα κατευθύνονται από το μικρότερο προς τον μεγαλύτερο κόμβο, είναι ότι γίνονται περισσότερες προωθήσεις μεταξύ των κόμβων. Αυτό συμβαίνει, διότι τα μηνύματα μετακινούνται μέχρι να βρεθεί ένας κόμβος, ο οποίο θέλει να γίνει αρχηγός και έχει μικρότερο αναγνωριστικό, ώστε να τα απορρίψει. Αντίθετα, όταν τα μηνύματα κατευθύνονται προς την αντίθετη κατεύθυνση, πραγματοποιούνται λιγότερες προωθήσεις. Οι παρατηρήσεις αυτές δεν ισχύουν στην περίπτωση κατά την οποία υπάρχει μόνο ένας κόμβος, ο οποίος επιδιώκει να εκλεχθεί, καθώς ο αριθμός των προωθήσεων παραμένει σταθερός, όπως φαίνεται και στα δίκτυα της ενότητας 2.1.1 (σχήμα 3 και 4). Στο διάγραμμα 1, το οποίο ακολουθεί, φαίνονται οι συνολικές προωθήσεις των μηνυμάτων, οι οποίες πραγματοποιήθηκαν στις προσομοιώσεις δικτύων με 15 κόμβους της ενότητας 2.1, όταν αυτά κατευθύνονταν από τον μικρότερο προς τον μεγαλύτερο κόμβο (**μαύρη γραμμή**) και αντίστροφα (**πράσινη γραμμή**).



Γράφημα 1: Αποτελέσματα προσομοιώσεων με δίκτυα 15 κόμβων

2.2.2 Καλύτερη και χειρότερη περίπτωση

Από τις προσομοιώσεις της ενότητας 2.1, υπάρχει επίσης η δυνατότητα να διακριθούν η καλύτερη και η χειρότερη περίπτωση εκτέλεσης του αλγορίθμου όταν όλοι οι κόμβοι του δικτύου επιθυμούν να γίνουν αρχηγοί. Πιο συγκεκριμένα, στις προσομοιώσεις της ενότητας 2.1.3 (σχήμα 7 και 8), όταν τα μηνύματα προωθούνται από τον μικρότερο προς τον μεγαλύτερο κόμβο, γίνονται συνολικά 15 και 120 προωθήσεις στα δίκτυα με 5 και 15 κόμβους αντίστοιχα. Η περίπτωση αυτή, αποτελεί και την χειρότερη περίπτωση εκτέλεσης του αλγορίθμου καθώς πραγματοποιεί:

$$n(n + 1)/2$$

$$5(5 + 1)/2 = 15$$

και

$$15(15 + 1)/2 = 120$$

προωθήσεις έως την εκλογή του αρχηγού – τον τερματισμό του αλγορίθμου. Από την άλλη μεριά, όταν τα μηνύματα προωθούνται προς την αντίθετη κατεύθυνση του δακτυλίου, πραγματοποιούνται συνολικά 9 και 29 προωθήσεις στα δίκτυα με 5 και 15 κόμβους αντίστοιχα. Η περίπτωση αυτή, αποτελεί και την βέλτιστη περίπτωση εκτέλεσης του αλγορίθμου καθώς γίνονται:

$$n + n - 1 = 2n - 1$$

$$2 * 5 - 1 = 9$$

και

$$2 * 15 - 1 = 29$$

προωθήσεις έως την εκλογή του αρχηγού – τον τερματισμό του αλγορίθμου.

Βιβλιογραφία

- Tel, G. (2000). *Introduction to Distributed Algorithms*. 2nd ed. Cambridge, United Kingdom: Cambridge University Press.

- Chang, E. Roberts, R. (1979). *An Improved Algorithm for Decentralized Extrema-Finding in Circular Configurations of Processes*. University of Toronto. University of Waterloo.

<https://dl.acm.org/doi/abs/10.1145/359104.359108>