



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

«Υλοποίηση Αλγορίθμου Tarry's Traversal»

Για το μάθημα

Κατανεμημένα Δικτυοκεντρικά Συστήματα

Ονοματεπώνυμο: Γρίβας Πασχάλης

Εξάμηνο Φοίτησης: ΣΤ΄

A.M.: Π2017082

Περίληψη

Ο αλγόριθμος του Tarry, ορισμένος το 1895, είναι ένας αλγόριθμος διάσχισης (traversal algorithm) δικτύων με τυχαία τοπολογία κόμβων. Η λειτουργία του αλγορίθμου είναι βασισμένη σε δύο κανόνες: (α) ένας κόμβος δεν προωθεί το μήνυμα (token) δεύτερη φορά μέσω του ίδιου καναλιού και (β) ένας κόμβος ο οποίος δεν έχει οριστεί ως εκκινητής (initiator), προωθεί το μήνυμα στον κόμβο – γονέα του, δηλαδή τον κόμβο από τον οποίο έλαβε πρώτη φορά μήνυμα, μόνο όταν δεν υπάρχει κάποιο άλλο κανάλι μέσω του οποίου δεν έχει στείλει. Έτσι, με την εφαρμογή των δύο αυτών κανόνων, το μήνυμα στέλνεται μόνο μία φορά σε κάθε κατεύθυνση του κάθε καναλιού με αποτέλεσμα να γίνονται συνολικά $2|E|$ επί τον αριθμό των καναλιών - ακμών ($2|E|$) προωθήσεις του μηνύματος έως λήξη του αλγορίθμου.

Στόχος αυτής της εργασίας είναι η υλοποίηση του αλγορίθμου και εκτέλεσή του σε πλήθος προσομοιώσεων με διάφορους τυχαίους γράφους και η καταγραφή και σύγκριση των αποτελεσμάτων με την αντίστοιχη θεωρία και την πολυπλοκότητα του.

Λέξεις Κλειδιά

Tarry's Traversal, Κατανεμημένος αλγόριθμος, 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	6
2. Αποτελέσματα	8
2.1 Παρουσίαση των προσομοιώσεων	8
2.1.1 Απλό δίκτυο παραδείγματος	9
2.1.2 Δίκτυο δυαδικού δέντρου	10
2.1.3 Δίκτυο αλυσίδας	11
2.1.4 Δίκτυο δακτυλίου	12
2.1.5 Πλήρες συνεκτικό δίκτυο	13
2.1.6 Τυχαίος γράφος	14
2.2 Ερμηνεία των αποτελεσμάτων	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, δημιουργήθηκαν και αντικείμενα τύπου network, ένα για κάθε είδος δικτύου (π.χ. Chain.net), τα οποία περιέχουν πληροφορίες σχετικά με την τοπολογία του κάθε δικτύου όπως τους κόμβους τους οποίους θα περιλαμβάνουν, 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, μόνο ο κόμβος ο οποίος έχει οριστεί ως εκκινητής γίνεται κόκκινος και στέλνει ένα μήνυμα στον εαυτό του.

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 είναι ο κόμβος – εκκινητής, γίνεται πορτοκαλί και ορίζει ως πατέρα του τον εαυτό του. Διαφορετικά, αν το μήνυμα προέρχεται από κάποιον άλλο κόμβο, τότε αυτός ο κόμβος ορίζεται ως πατέρας στην περίπτωση που ο τρέχων κόμβος δεν έχει δηλώσει έως τώρα κάποιον κόμβο για πατέρα του.

```
void Node::handleMessage(cMessage *msg)
{
    ...
    if(!msg->isSelfMessage())
    {
        ...
        if(haveParent == false)
        {
            parentId = ((Node*)msg->getSenderModule())->id;
            haveParent = true;
        }
        ...
    }
    else
    {
        ...
        this->setDisplayString(myDispStr.str());
        ...
        parentId = this->id;
        haveParent = true;
    }
}
```

Πρέπει να σημειωθεί ότι στο κομμάτι αυτό του κώδικα δημιουργούνται και τα μηνύματα τα οποία θα αποσταλούν σε γείτονες του κάθε κόμβου. Σύμφωνα με την θεωρία του αλγορίθμου το μήνυμα (token) το οποίο ανταλλάσσεται μεταξύ των κόμβων παραμένει το ίδιο. Ωστόσο, αυτό δε συμβαίνει σε αυτή την υλοποίηση καθώς προκειμένου να εμφανίζεται το σύνολο των προωθήσεων των μηνυμάτων, το κείμενο του μηνύματος έχει το ρόλο ενός μετρητή (counter).

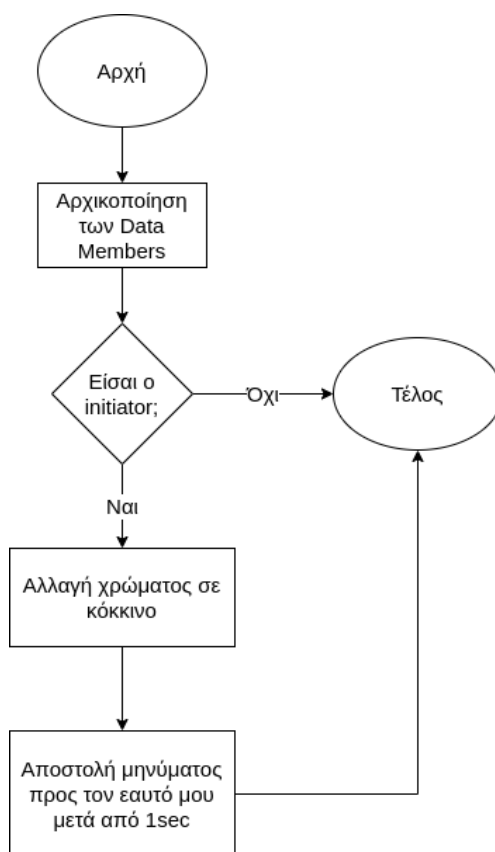
Έπειτα, εντός της ίδιας μεθόδου, εκτελείται μία επαναληπτική ο κώδικας της οποίας επαναλαμβάνεται τόσες φορές όσο το πλήθος των συνδέσεων – γειτόνων του τρέχων κόμβου. Αν ο γειτονικός κόμβος, ο οποίος εξετάζεται στην τρέχουσα επανάληψη, δεν είναι ο κόμβος ο οποίος έχει οριστεί ως πατέρας του τρέχοντος κόμβου τότε αν υπάρχουν γειτονικοί κόμβοι (εκτός από τον πατέρα) στους οποίους ο τρέχων κόμβος δεν έχει στείλει ακόμα μήνυμα τότε αν ο τρέχων κόμβος δεν έχει ήδη στείλει σε αυτόν τον γείτονα [κανόνας (α)] τότε ο τρέχων κόμβος του στέλνει το μήνυμα. Σε κάθε άλλη περίπτωση ο τρέχων κόμβος στέλνει το μήνυμα στον κόμβο τον οποίο έχει ορίσει ως πατέρα του. [κανόνας (β)]

```
void Node::handleMessage(cMessage *msg)
{
    ...
    for (i=0; i<numOfGates; i++)
    {
        ...
        if(currentNeighborId != parentId) //if current output gate not leading to parent
        {
            if(numOfSends < numOfGates) //if i still have neighbors to send except parent
            {
                ...
                if (it != alreadySendTo.end()) //if i have already send to this node
                    continue;
                else
                {
                    ...
                    numOfSends++;
                    send(message->dup(), gate("inoutGateVector$o", i));
                    alreadySendTo.push_back(currentNeighborId);
                    break;
                }
            }
        }
        else
        {
            if(id == int(getParentModule()->par("initiator")))
            {
                ...
                bubble("Finished!");
            }
        }
    }
    else
    {
        if(id != int(getParentModule()->par("initiator")))
        {
            numOfSends++;
            send(message->dup(), gate("inoutGateVector$o", i));
            alreadySendTo.push_back(currentNeighborId);
            break;
        }
    }
    ...
}
```

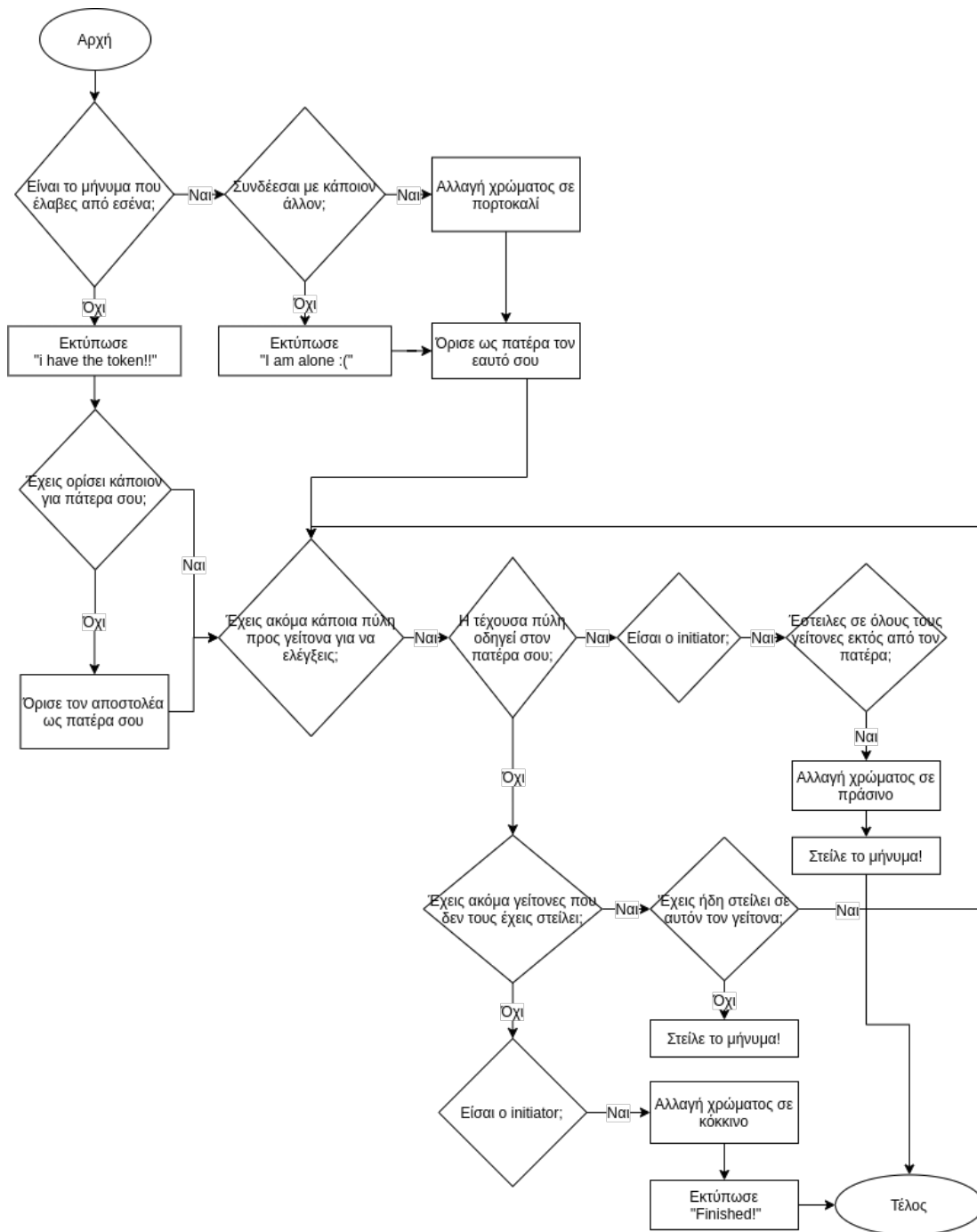
Εδώ πρέπει να αναφερθεί ότι στον πλήρη κώδικα περιλαμβάνετε και η υλοποίηση ενός τρίτου κανόνα ο οποίος τροποποιεί τον αλγόριθμο έτσι ώστε να εκτελεί αναζήτηση κατά βάθος (depth-first search). Αυτός ο κανόνας επιβάλλει κάθε κόμβο ο οποίος λαμβάνει ένα μήνυμα να το στέλνει πίσω σε αυτόν που το έστειλε αν αυτό επιτρέπεται από τους προηγούμενους δύο κανόνες (α) και (β).

1.2 Διάγραμμα ροής του κώδικα

Στα παρακάτω σχήματα παρουσιάζεται η διαδικασία την οποία ακολουθεί το πρόγραμμα προκειμένου να προσομοιωθεί ο αλγόριθμος διάσχισης. 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



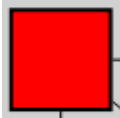


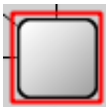
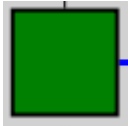


Σχήμα 2: Διάγραμμα Ροής - handleMessage

Κεφάλαιο 2

Αποτελέσματα

2.1 Παρουσίαση των προσομοιώσεων

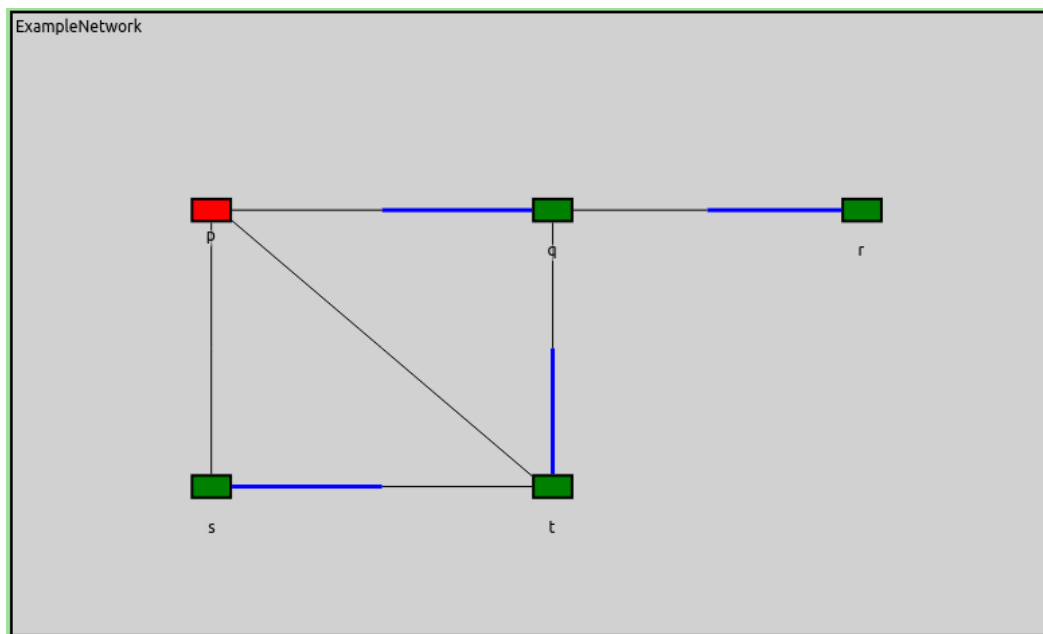
Προκειμένου να εκτελεστεί ο κώδικας ο οποίος αναλύθηκε στο κεφάλαιο 1 είναι απαραίτητο να έχει επιλεχθεί το αρχείο omnetpp.ini το οποίο περιέχει τις ρυθμίσεις των παραμέτρων όλων των δικτύων τα οποία έχουν υλοποιηθεί. Επίσης, δίνεται ο παρακάτω πίνακας ο οποίος περιλαμβάνει τους χρωματισμούς και την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

Απεικόνιση	Στοιχείο	Ερμηνεία
	Κόμβος Εκκινητής – Initiator	Έχει αρχικοποιηθεί ή έχει λάβει πίσω το μήνυμα από όλους τους γείτονές του.
	Κόμβος Εκκινητής – Initiator	Αναμένει να λάβει πίσω το μήνυμα που έστειλε από όλους τους γείτονες.
	Κόμβος (non initiator)	Αδρανής κόμβος του δικτύου – περιμένει να λάβει μήνυμα.
	Ενεργός Κόμβος (non initiator)	Ενεργός κόμβος του δικτύου – έχει λάβει μήνυμα και το προωθεί.
	Κόμβος ο οποίος έχει τερματίσει.	Έχει προωθήσει το μήνυμα σε όλους τους γείτονες του και τέλος στον πατέρα του.
	Κανάλι προς τον κόμβο - πατέρα	-
	Μήνυμα - Token	-

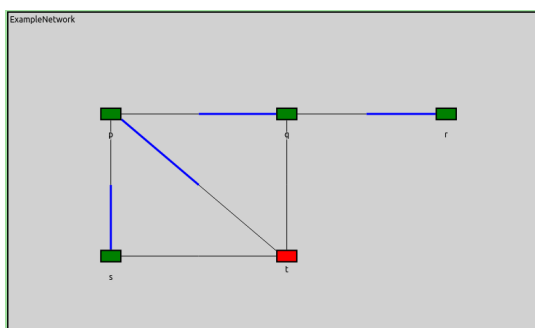
Πίνακας 1: Ερμηνεία των στοιχείων της προσομοίωσης

2.1.1 Απλό δίκτυο παραδείγματος

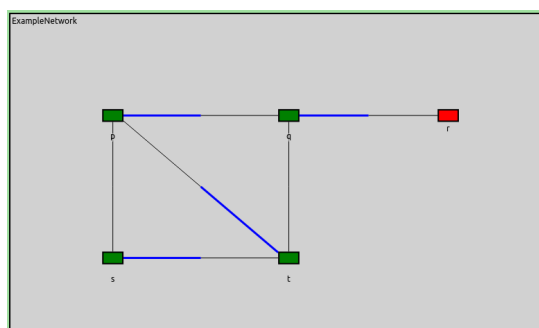
Στα σχήματα παρακάτω φαίνεται η εκτέλεση του αλγορίθμου σε ένα δίκτυο 5 κόμβων και 6 ακμών το οποίο περιλαμβάνεται στα παραδείγματα της παρουσίασης του Dr. Borzoo Bonakdarpour - Distributed Algorithms (CAS 769) - McMaster University. Κατά την έναρξη της προσομοίωσης πρέπει να επιλεγθεί το “ExampleNetwork5Nodes” και στη συνέχεια το id του κόμβου ο οποίος πρόκειται να γίνει εκκινήτης (1000-1004). Στο σχήμα 3 κόμβος εκκινήτης είναι ο “p”, στο σχήμα 4 ο “t” και στο σχήμα 5 ο “r”. Και στις τρεις εκτελέσεις έγιναν συνολικά 12 προωθήσεις του μηνύματος μεταξύ των κόμβων έως τον τερματισμό του αλγορίθμου.



Σχήμα 3: Δίκτυο Παραδείγματος - Initiator p



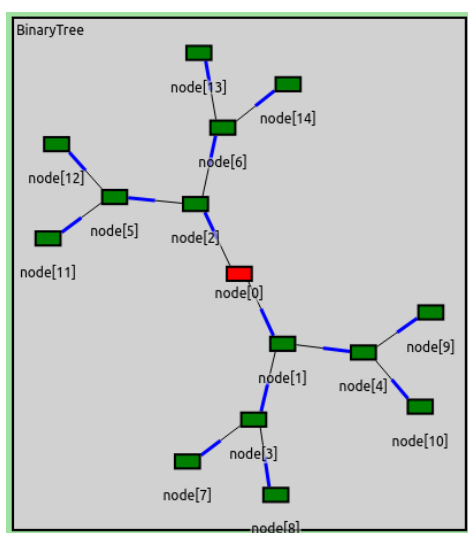
Σχήμα 4: Δίκτυο Παραδείγματος - Initiator t



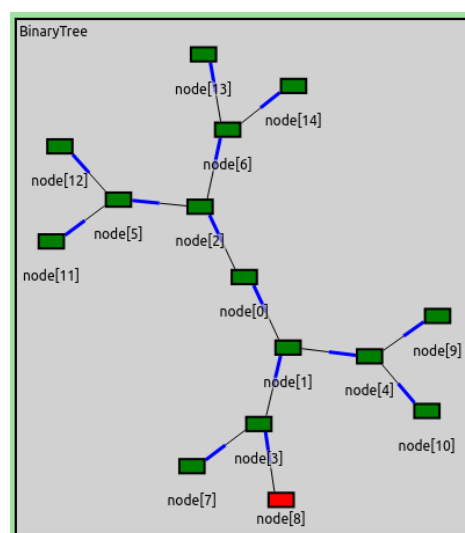
Σχήμα 5: Δίκτυο Παραδείγματος - Initiator r

2.1.2 Δίκτυο δυαδικού δέντρου

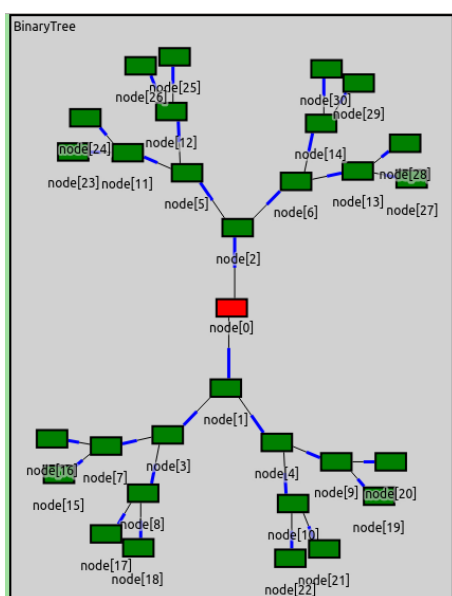
Κατά την έναρξη της προσομοίωσης αν επιλεγθεί το “BinaryTree” θα προσομοιωθεί η λειτουργία του αλγορίθμου σε ένα δίκτυο με την μορφή ενός δυαδικού δέντρου. Το ύψος του δέντρου μπορεί να οριστεί από τον χρήστη όπως και το id του κόμβου - εκκινήτη. Στα σχήματα 6 και 7 δημιουργήθηκαν δυαδικά δέντρα με ύψος 3 και initiators τους κόμβους 1000 και 1008 αντίστοιχα. Και στις δύο εκτελέσεις έγιναν συνολικά 28 προωθήσεις του μηνύματος μεταξύ των κόμβων έως τον τερματισμό του αλγορίθμου. Από την άλλη, στα σχήματα 8 και 9 δημιουργήθηκαν δέντρα ύψους 4 και initiators τους 1000 και 1018 αντίστοιχα. Οι συνολικές προωθήσεις του μηνύματος ήταν 60.



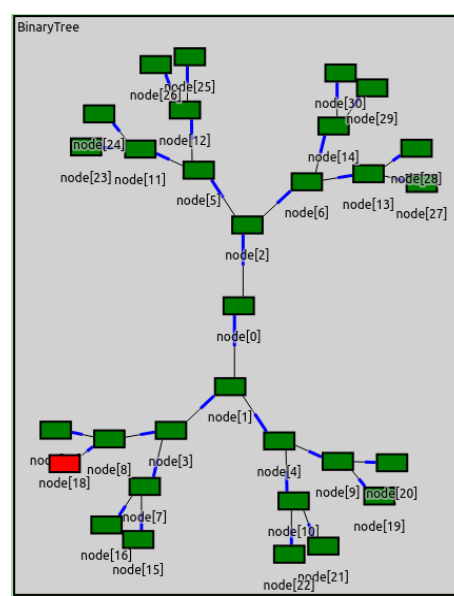
Σχήμα 6: Δυαδικό Δέντρο Υ:3 - Init:1000



Σχήμα 7: Δυαδικό Δέντρο Υ:3 - Init:1008



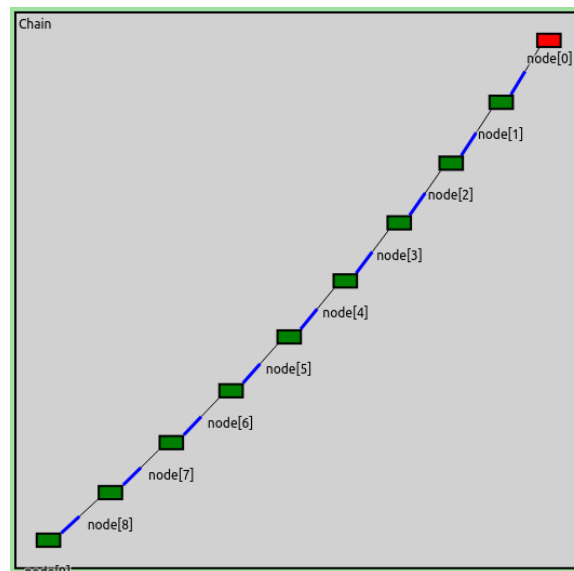
Σχήμα 8: Δυαδικό Δέντρο Υ:4 - Init:1000



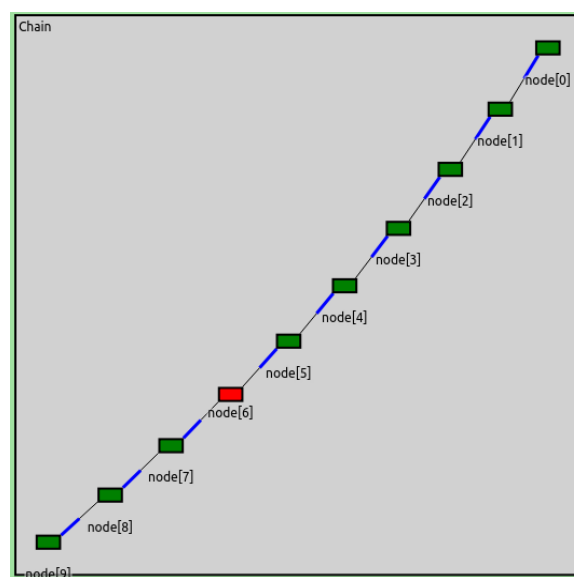
Σχήμα 9: Δυαδικό Δέντρο Υ:4 - Init:1008

2.1.3 Δίκτυο αλυσίδας

Ένα άλλο δίκτυο το οποίο παρέχεται και μπορεί να τρέξει ο αλγόριθμος είναι το δίκτυο αλυσίδας, αν επιλεγθεί το “Chain” στην έναρξη της προσομοίωσης. Την ίδια στιγμή μπορεί να οριστεί από τον χρήστη το πλήθος των κόμβων του δικτύου όπως και το id του initiator. Στα σχήματα 10 και 11 δημιουργήθηκαν δίκτυα με 10 κόμβους και 9 ακμές όπου στο πρώτο initiator είναι ο κόμβος 1000, στην αρχή της αλυσίδας και στο δεύτερο ο κόμβος 1006 στη μέση. Και στις δύο εκτελέσεις έγιναν συνολικά 18 προωθήσεις του μηνύματος μεταξύ των κόμβων έως τον τερματισμό του αλγορίθμου.



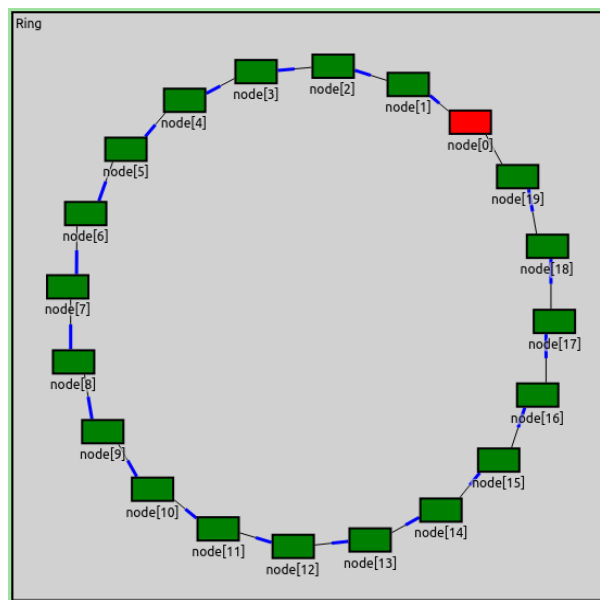
Σχήμα 10: Αλυσίδα - Init:1000



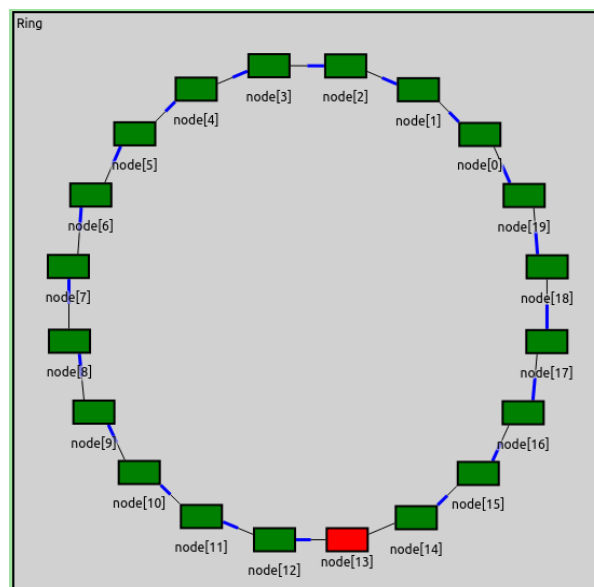
Σχήμα 11: Αλυσίδα - Init:1006

2.1.4 Δίκτυο δακτυλίου

Παρόμοια με το δίκτυο αλυσίδας υπάρχει και το δακτυλίου, αν επιλεγθεί το “Ring” στην έναρξη της προσομοίωσης. Αντίστοιχα, μπορεί να οριστεί από τον χρήστη το πλήθος των κόμβων του δικτύου όπως και το id του initiator. Στα σχήματα 12 και 13 δημιουργήθηκαν δίκτυα με 20 κόμβους και 20 ακμές όπου στο πρώτο initiator είναι ο κόμβος 1000 και στο δεύτερο ο κόμβος 1013. Και στις δύο εκτελέσεις έγιναν συνολικά 40 προωθήσεις του μηνύματος μεταξύ των κόμβων έως τον τερματισμό του αλγορίθμου.



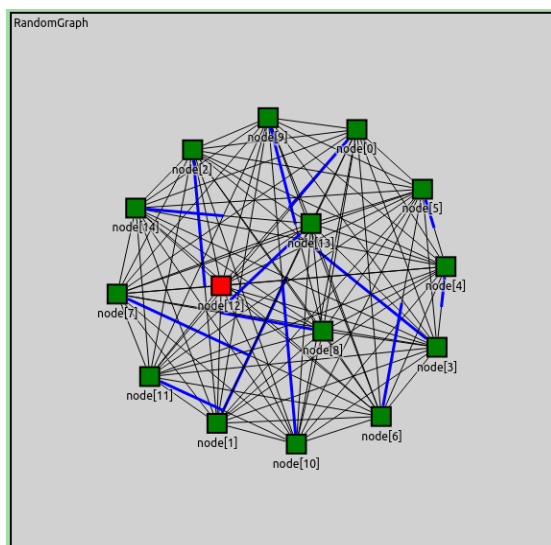
Σχήμα 12: Δακτύλιος - Init:1000



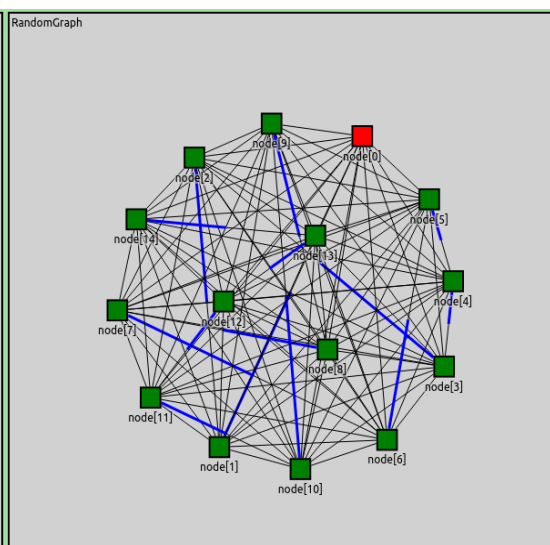
Σχήμα 13: Δακτύλιος - Init:1013

2.1.5 Πλήρες συνεκτικό δίκτυο

Στην έναρξη της προσομοίωσης, αν επιλεγθεί το “RandomGraph” δίνεται η δυνατότητα να δημιουργηθεί ένα δίκτυο στο οποίο οι θέσεις των κόμβων και οι συνδέσεις μεταξύ τους είναι τυχαίες. Ωστόσο, μπορεί να οριστεί από τον χρήστη το πλήθος των κόμβων του δικτύου, το id του initiator και η πιθανότητα να δημιουργείται ή όχι σύνδεση μεταξύ δύο κόμβων. Ορίζοντας την τιμή αυτής της πιθανότητας στο 1 σχηματίζεται ένας πλήρης συνεκτικός γράφος καθώς όλοι οι κόμβοι συνδέονται με όλους τους υπόλοιπους κόμβους του δικτύου. Στα σχήματα 14 και 15 δημιουργήθηκαν πλήρη συνεκτικά δίκτυα με 15 κόμβους και 105 ακμές ($15 \times 14 / 2$) όπου στο πρώτο initiator είναι ο κόμβος 1000 και στο δεύτερο ο κόμβος 1012. Και στις δύο εκτελέσεις έγιναν συνολικά 210 προωθήσεις του μηνύματος μεταξύ των κόμβων έως τον τερματισμό του αλγορίθμου.



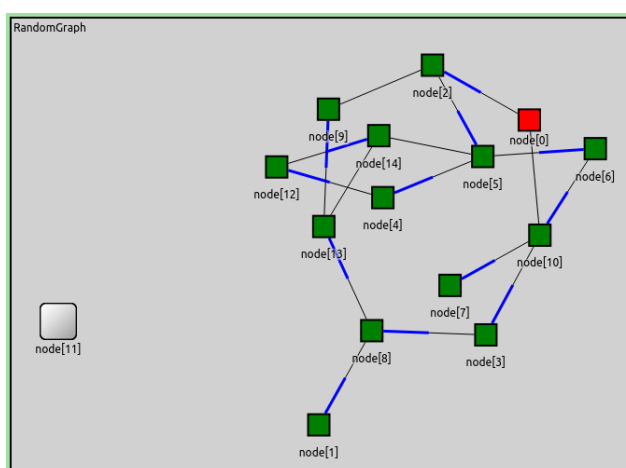
Σχήμα 14: Πλήρες - Init:1012



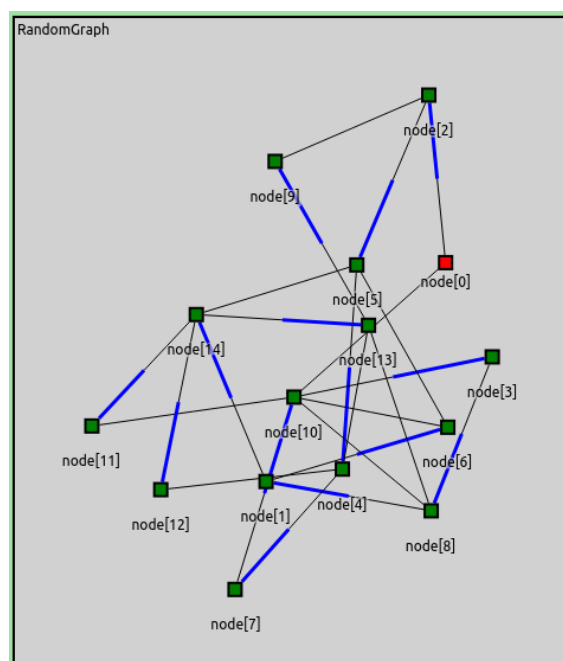
Σχήμα 15: Πλήρες - Init:1000

2.1.6 Τυχαίος γράφος

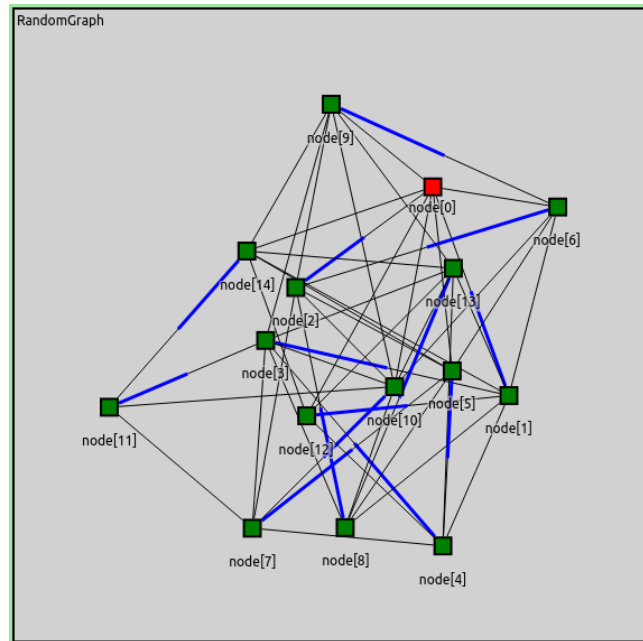
Τέλος, τρέχοντας την προσομοίωση με την επιλογή “RandomGraph” και αλλάζοντας την τιμή της πιθανότητας να δημιουργείται ή όχι σύνδεση μεταξύ δύο κόμβων, δίνεται η δυνατότητα να προσομοιωθεί ο αλγόριθμος σε διάφορους τυχαίους γράφους. Επίσης, από το omnetpp.ini αρχείο, αλλάζοντας την τιμή της μεταβλητής “seed-set” αλλάζουν οι τυχαίοι αριθμοί οι οποίοι παράγονται στο πρόγραμμα και καθορίζουν την μορφή του δικτύου. Έτσι, ορίζοντας την τιμή του παράγοντα τυχαιότητας σε 5, στα σχήματα 16, 17 και 18 δημιουργήθηκαν δίκτυα με 15 κόμβους, initiator τον κόμβο 1000 και τιμές της πιθανότητας να δημιουργείται ή όχι σύνδεση μεταξύ δύο κόμβων 0.15, 0.2 και 0.5 αντίστοιχα. Οι συνολικές προωθήσεις των μηνυμάτων στην πρώτη περίπτωση ήταν 34, στη δεύτερη 48 και στην τρίτη 102.



Σχήμα 16: Τυχαίος Γράφος - Seed:5 C:0.15 Init:1000

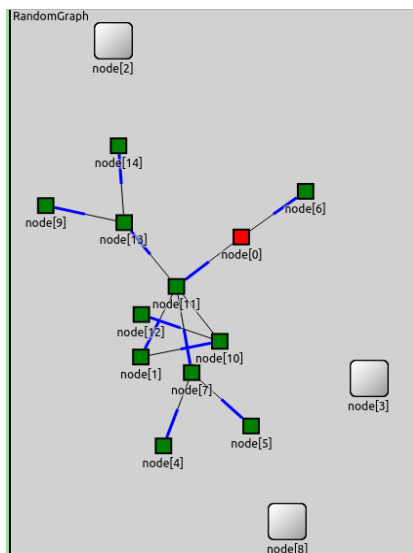


Σχήμα 17: Τυχαίος Γράφος - Seed:5 C:0.2 Init:1000

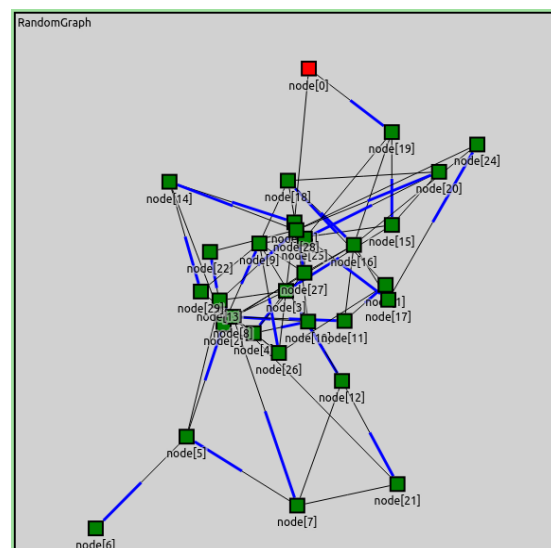


Σχήμα 18: Τυχαίος Γράφος - Seed:5 C:0.5 Init:1000

Με τον ίδιο τρόπο δημιουργήθηκαν και τα δίκτυα 15 και 30 κόμβων τα οποία φαίνονται στα σχήματα 19 και 20 αντίστοιχα. Με παράγοντα τυχαιότητας στο πρώτο 25 και στο δεύτερο 80, πιθανότητα των συνδέσεων 0,15 και initiator τον κόμβο 1000, έγιναν 24 και 126 προωθήσεις του μηνύματος μέχρι την λήξη της προσομοίωσης.



Σχήμα 19: Πλήρες - Init:1012

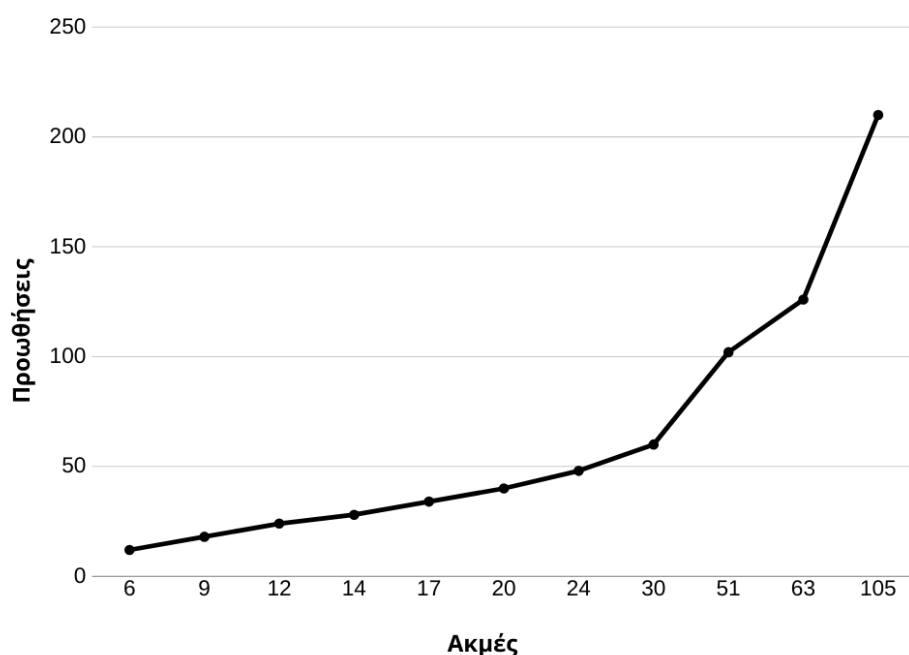


Σχήμα 20: Πλήρες - Init:1012

2.2 Ερμηνεία των αποτελεσμάτων

Τα αποτελέσματα των προσομοιώσεων τα οποία παρουσιάστηκαν στην προηγούμενη ενότητα, αποδεικνύουν ότι η διάσχιση των κόμβων με τον αλγόριθμο του Tarry δημιουργεί ένα από τα spanning trees του δικτύου. Ως spanning tree ορίζεται ένα δέντρο (υπογράφος) το οποίο περιλαμβάνει όλους τους κόμβους του δικτύου με τις λιγότερες δυνατές ακμές. Ρίζα του δέντρου αυτού είναι ο κόμβος – εκκινητής και επεκτείνεται με τις ακμές μπλε χρώματος (δηλαδή παιδιά του είναι οι κόμβοι οι οποίοι τον έχουν ορίσει ως πατέρα κ.ο.κ) όπως φαίνεται στα σχήματα των προσομοιώσεων.

Ένα άλλο γεγονός, το οποίο μπορεί να παρατηρηθεί και στο διάγραμμα 1 παρακάτω με τα αποτελέσματα των προσομοιώσεων συνολικά, είναι ότι ο αλγόριθμος σε κάθε προσομοίωση ανεξάρτητα από το είδος του δικτύου στο οποίο έτρεχε πραγματοποιούσε 2Ε προωθήσεις του μηνύματος μέχρι την λήξη του καθώς το μήνυμα προωθείται μόνο μία φορά σε κάθε κατεύθυνση του κάθε καναλιού. Για παράδειγμα, στο απλό δίκτυο 5 κόμβων της ενότητας 2.1.1 γίνονται συνολικά 12 προωθήσεις, δηλαδή οι διπλάσιες από το σύνολο των καναλιών. Επίσης, παρατηρείται ότι οι αναμενόμενες συνολικές προωθήσεις του μηνύματος δεν αλλάζουν ανεξάρτητα από το πλήθος των κόμβων (π.χ. σχήμα 6 και 8), το ποιος κόμβος έχει οριστεί ως εκκινητής (π.χ. σχήμα 14 και 15) και αν το δίκτυο είναι ένας πλήρης συνεκτικός γράφος (σχήμα 15) ή ένας τυχαίος μη συνεκτικός γράφος (σχήμα 19).



Γράφημα 1: Αποτελέσματα προσομοιώσεων

Βιβλιογραφία

- Tel, G. (2000). *Introduction to Distributed Algorithms*. 2nd ed. Cambridge, United Kingdom: Cambridge University Press.
- Dr. Bonakdarpour, B. *Distributed Algorithms (CAS 769) - Week 2: Graph Algorithms*. Department of Computing and Software, McMaster University
<http://web.cs.iastate.edu/~borzoo/teaching/15/CAS769/lectures/week1.pdf>