Διατριβή Μάστερ

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΑΣΥΓΧΡΟΝΩΝ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΑΛΓΟΡΙΘΜΩΝ ΠΛΗΡΟΦΟΡΗΣΗΣ

Κωνσταντίνα Σπανούδη

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2011

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΑΣΥΓΧΡΟΝΩΝ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΑΛΓΟΡΙΘΜΩΝ ΠΛΗΡΟΦΟΡΗΣΗΣ

Κωνσταντίνα Σπανούδη

Η Διατριβή αυτή

Υποβλήθηκε προς Μερική Εκπλήρωση των

Απαιτήσεων για την Απόκτηση

Τίτλου Σπουδών Master

στην Επιστήμη της Πληροφορικής

στο

Πανεπιστήμιο Κύπρου

Συστήνεται προς Αποδοχή

από το Τμήμα Πληροφορικής

Μάιος, 2012

ΠΕΡΙΛΗΨΗ

Τα κατανεμημένα συστήματα υπολογιστών διέρχονται μια περίοδο εκρηκτικής εξέλιξης. Οι ανάγκες για πολύπλοκους υπολογισμούς, που δεν μπορούν να διεκπεραιώσουν οι σειριακοί υπολογιστές, αυξάνονται συνεχώς. Το Διαδίκτυο έχει γίνει πλέον μια δημοφιλής πλατφόρμα παράλληλου υπολογισμού. Λόγω της μεγάλης αυτής ανάπτυξης προκύπτει το πρόβλημα της μετάδοσης της πληροφορίας σε τέτοια συστήματα, γνωστό και ως "gossip problem". Δημιουργείται, δηλαδή, η ανάγκη για σχεδιασμό εύρωστων κατανεμημένων αλγορίθμων πληροφόρησης που να μεταδίδουν διάφορα δεδομένα στους κόμβους των συστημάτων αυτών αποτελεσματικά και αποδοτικά.

Στην παρούσα Διατριβή Μάστερ παρουσιάζεται η υλοποίηση και η πειραματική αξιολόγηση δύο εύρωστων κατανεμημένων αλγορίθμων πληροφόρησης, οι οποίοι είναι προς το παρόν οι μοναδικοί που έχουν αναπτυχθεί και αυστηρότυπα αναλυθεί για ασύγχρονα κατανεμημένα συστήματα. Οι αλγόριθμοι αυτοί είναι ο EARS και ο SEARS, δύο εύρωστοι αλγόριθμοι οι οποίοι είναι αποδοτικοί ακόμα και στην παρουσία σφαλμάτων δικτύου.

Αφού μελετήθηκαν αρκετά, οι εν λόγω αλγόριθμοι υλοποιήθηκαν στη γλώσσα προγραμματισμού JAVA με τη χρήση της YALPS, μιας βιβλιοθήκης που υποβοηθά την υλοποίηση κατανεμημένων αλγορίθμων και την προσομοίωση ή εκτέλεσή τους σε πραγματικό υπολογιστικό περιβάλλον. Οι αλγόριθμοι εφαρμόστηκαν πρώτα σε περιβάλλον προσομοίωσης με σκοπό να καταμετρηθούν οι επιδόσεις τους και ακολούθως στο PlanetLab, ένα πραγματικό Διαδικτυακό σύστημα, ως ισχυρό επιχείρημα για να επιβεβαιωθεί η πρακτικότητά τους.

Η εμπειρική αξιολόγηση των αλγορίθμων οδήγησε στο γενικό συμπέρασμα ότι η πειραματική τους απόδοση συνάδει με τη θεωρητική αξιολόγησή τους. Και οι δύο αλγόριθμοι είναι εύρωστοι και πρακτικοί, με τον SEARS, μάλιστα, για μεγάλο αριθμό κόμβων να είναι ένας σταθερού χρόνου αλγόριθμος πληροφόρησης.

ΣΕΛΙΔΑ ΕΓΚΡΙΣΗΣ

Διατριβή Master

ΥΛΟΠΟΙΗΣΗ ΚΑΙ ΠΕΙΡΑΜΑΤΙΚΗ ΑΞΙΟΛΟΓΗΣΗ ΑΣΥΓΧΡΟΝΩΝ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΑΛΓΟΡΙΘΜΩΝ ΠΛΗΡΟΦΟΡΗΣΗΣ

Παρουσιάστηκε από

Κωνσταντίνα Σπανούδη

Ερευνητικός Σύμβουλος _____

Χρύσης Γεωργίου

Μέλος Επιτροπής _____

Άννα Φιλίππου

Μέλος Επιτροπής _____

Νικόλας Νικολάου

Πανεπιστήμιο Κύπρου

Μάιος, 2012

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή της διατριβής μου Δρ Χρύση Γεωργίου, Επίκουρο καθηγητή του τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου, για την καθοδήγηση και τη βοήθειά του σε κάθε φάση της δημιουργίας της.

Θα ήθελα να εκφράσω επίσης την ευγνωμοσύνη μου στην οικογένειά μου για τη διαρκή υποστήριξη, που επέτρεψε την επιτυχή διεκπεραίωση των σπουδών μου.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Κεφάλαιο 1 ..... **1**

Εισαγωγή **1**

1.1 Κίνητρο 1

1.2 Προσφορά Διατριβής 3

1.3 Δομή της Διατριβής 5

Κεφάλαιο 2 ..... **7**

Υπόβαθρο **7**

2.1 Το Πρόβλημα της Πληροφόρησης 7

2.2 Αλγόριθμοι Πληροφόρησης 8

2.3 Μοντέλα Παρεμπόδισης Διάδοσης Πληροφοριών 9

2.4 Ο Μηχανισμός Διάχυσης της Πληροφορίας σε Ασύγχρονο Περιβάλλον 11

2.5 Προηγούμενες Εργασίες 14

2.6 Η Βιβλιοθήκη YALPS 15

Κεφάλαιο 3 ..... **17**

Περιγραφή και Υλοποίηση Αλγορίθμων **17**

3.1 Μοντέλο Συστήματος Υλοποίησης 17

3.2 Υλοποίηση Αλγορίθμων με Χρήση YALPS 19

3.3 Αλγόριθμος EARS 24

3.3.1 Περιγραφή Αλγορίθμου EARS 24

3.3.2 Λεπτομέρειες Υλοποίησης EARS 27

3.4 Αλγόριθμος SEARS 34

3.4.1 Περιγραφή Αλγορίθμου SEARS 35

3.4.2 Λεπτομέρειες Υλοποίησης SEARS 36

Κεφάλαιο 4 ..... **39**

Αξιολόγηση Αλγορίθμων σε Περιβάλλον Προσομοίωσης **39**

4.1 Μετρικές Αξιολόγησης Πολυπλοκότητας 39

4.2 Μεθοδολογία 42

4.3 Πειραματική Αξιολόγηση Αλγορίθμου EARS 45

4.3.1 Περίπτωση με σταθερό d+δ 45

4.3.1.1 Αποτελέσματα εφαρμογής χωρίς σφάλματα 46

4.3.1.2 Αποτελέσματα εφαρμογής με σφάλματα 52

4.3.2 Περίπτωση με μεταβλητό d+δ 56

4.3.2.1 Αποτελέσματα εφαρμογής με και χωρίς σφάλματα 56

4.3.4 Συμπεράσματα 60

4.4 Πειραματική Αξιολόγηση Αλγορίθμου SEARS 61

4.4.1 Περίπτωση με σταθερό d+δ 62

4.4.1.1 Αποτελέσματα εφαρμογής χωρίς σφάλματα 62

4.4.1.2 Αποτελέσματα εφαρμογής με σφάλματα 65

4.4.2 Περίπτωση όπου d+δ = μεταβλητό 69

4.4.2.1 Αποτελέσματα εφαρμογής με και χωρίς σφάλματα 70

4.4.4 Συμπεράσματα 72

4.5 Γενικά Συμπεράσματα 73

Κεφάλαιο 5 ..... **75**

Εφαρμογή Αλγορίθμων σε πραγματικό περιβάλλον **75**

5.1 Το Δικτυακό Σύστημα PlanetLab 75

5.2 Σύντομος Οδηγός Χρήσης του Planet Lab 76

5.3 Μεθοδολογία 78

5.4 Αποτελέσματα Εφαρμογής Αλγορίθμου EARS στο PlanetLab 81

5.5 Αποτελέσματα Εφαρμογής Αλγορίθμου SEARS στο PlanetLab 82

5.6 Συμπεράσματα 85

Κεφάλαιο 6 ..... **86**

Σύγκριση Αλγορίθμων **86**

6.1 Σύγκριση ως Προς τον Αριθμό Μηνυμάτων 86

6.1.1 Περίπτωση με σταθερό d+δ 87

6.1.2 Περίπτωση με μεταβλητό d+δ 88

6.1.3 Περίπτωση εφαρμογής στο Planet Lab 90

6.2 Σύγκριση ως Προς το Χρόνο Ολοκλήρωσης 91

6.2.1 Περίπτωση με σταθερό d+δ 92

6.2.2 Περίπτωση με μεταβλητό d+δ 93

6.2.3 Περίπτωση εφαρμογής στο Planet Lab 95

6.4 Συμπεράσματα 96

Κεφάλαιο 7 ..... **99**

Επίλογος **99**

7.1 Γενικά Συμπεράσματα 99

7.2 Δυσκολίες που παρουσιάστηκαν και πώς ξεπεράστηκαν 101

7.3 Μελλοντική Εργασία 103

Βιβλιογραφία **105**

Παράρτημα Α **Α-1**

Κώδικας Αλγορίθμου EARS σε γλώσσα προγραμματισμού JAVA Α-1

Παράρτημα Β **Β-1**

Κώδικας Αλγορίθμου SEARS σε γλώσσα προγραμματισμού JAVA Β-1

Παράρτημα Γ **Γ-1**

Παραδείγματα Εκτέλεσης Αλγορίθμου EARS Γ-1

ΚΑΤΑΛΟΓΟΣ ΜΕ ΠΙΝΑΚΕΣ

Πίνακας 4.1: Μετρήσεις EARS για f=1 και d+δ σταθερό (χωρίς κατάρρευση κόμβου)..... 47

Πίνακας 4.2: Μετρήσεις EARS για f=n/4 και d+δ σταθερό (χωρίς κατάρρευση κόμβου)..... 47

Πίνακας 4.3: Μετρήσεις EARS για f=1 και d+δ σταθερό (με κατάρρευση κόμβου)..... 52

Πίνακας 4.4: Μετρήσεις EARS για f=n/4 και d+δ σταθερό (με κατάρρευση κόμβου)..... 53

Πίνακας 4.5: Μετρήσεις EARS για f=1 και d+δ μεταβλητό (με και χωρίς σφάλματα) 57

Πίνακας 4.6: Μετρήσεις EARS για f=n/4 και d+δ μεταβλητό (με και χωρίς σφάλματα) 57

Πίνακας 4.7: Μετρήσεις SEARS για d+δ σταθερό (χωρίς σφάλματα)..... 62

Πίνακας 4.8: Μετρήσεις SEARS για f=1 και d+δ σταθερό (με κατάρρευση κόμβου)..... 66

Πίνακας 4.9: Μετρήσεις SEARS για f=n/4 και d+δ σταθερό (με κατάρρευση κόμβου)..... 66

Πίνακας 4.10: Μετρήσεις SEARS για d+δ μεταβλητό (με και χωρίς σφάλματα)..... 70

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

Εικόνα 4.1: Πειραματική και Θεωρητική προσέγγιση EARS για f=1 χωρίς σφάλματα 48

Εικόνα 4.2: Πειραματική και Θεωρητική προσέγγιση EARS για f=n/4 χωρίς σφάλματα 48

Εικόνα 4.3: Σύγκριση αριθμού Μηνυμάτων EARS για f=1 και f=n/4, χωρίς σφάλματα..... 49

Εικόνα 4.4: Πειραματική και Θεωρητική προσέγγιση EARS για f=1 χωρίς αποτυχίες..... 50

Εικόνα 4. 5: Πειραματική και Θεωρητική προσέγγιση EARS για f=n/4 χωρίς αποτυχίες..... 51

Εικόνα 4.6: Σύγκριση Χρόνου Ολοκλήρωσης EARS για f=1 και f=n/4, χωρίς σφάλματα..... 52

Εικόνα 4.7: Πειραματική σύγκριση Μηνυμάτων EARS για f=1 με και χωρίς αποτυχίες..... 53

Εικόνα 4.8: Πειραματική σύγκριση Μηνυμάτων EARS για f=n/4 με και χωρίς αποτυχίες.... 54

Εικόνα 4.9: Πειραματική σύγκριση EARS για f=1 με και χωρίς αποτυχίες..... 55

Εικόνα 4.10: Πειραματική σύγκριση EARS για f=n/4 με και χωρίς αποτυχίες..... 55

Εικόνα 4.11: Πειραματική σύγκριση EARS με σφάλματα f=n/4, d+δ=const και d+δ=var..... 58

Εικόνα 4.12: Πειραματική σύγκριση EARS με σφάλματα f=n/4, d+δ=const και d+δ=var..... 60

Εικόνα 4.13: Πειραματική και Θεωρητική προσέγγιση SEARS χωρίς αποτυχίες 63

Εικόνα 4.14: Πειραματική και Θεωρητική προσέγγιση SEARS χωρίς αποτυχίες 64

Εικόνα 4.15: Πειραματική προσέγγιση SEARS για d+δ = σταθερό, χωρίς αποτυχίες..... 65

Εικόνα 4.16: Πειραματική σύγκριση SEARS για f=1 με και χωρίς αποτυχίες 67

Εικόνα 4.17: Πειραματική σύγκριση SEARS για f=n/4 με και χωρίς αποτυχίες 67

Εικόνα 4.18: Πειραματική σύγκριση SEARS για f=n/4 με και χωρίς αποτυχίες 69

Εικόνα 4.19: Πειραματική σύγκριση SEARS με σφάλματα f=n/4, d+δ=const, d+δ=var..... 71

Εικόνα 4.20: Πειραματική σύγκριση SEARS f=n/4 σφάλματα, d+δ=const, d+δ=var 72

Εικόνα 5.1: Αλγόριθμος EARS – Αριθμός μηνυμάτων (PlanetLab)…… 81

Εικόνα 5.2: Αλγόριθμος EARS – Χρόνος Ολοκλήρωσης (PlanetLab)…… 81

Εικόνα 5.3: Αλγόριθμος SEARS – Αριθμός μηνυμάτων (PlanetLab)……..83

Εικόνα 5.4: Αλγόριθμος SEARS – Χρόνος Ολοκλήρωσης (PlanetLab)……..83

Εικόνα 6.1: Σύγκριση EARS και SEARS – Αριθμός μηνυμάτων (YALPS)……87

Εικόνα 6.2: Σύγκριση EARS και SEARS – Αριθμός μηνυμάτων (YALPS)……88

Εικόνα 6.3: Σύγκριση EARS και SEARS – Αριθμός μηνυμάτων (YALPS)……89

Εικόνα 6.4: Σύγκριση EARS και SEARS – Αριθμός μηνυμάτων (YALPS)……90

Εικόνα 6.5: Σύγκριση EARS και SEARS – Αριθμός μηνυμάτων (PlanetLab)…… 91

Εικόνα 6.6: Σύγκριση EARS και SEARS – Χρόνος Ολοκλήρωσης (YALPS)…… 92

Εικόνα 6.7: Σύγκριση EARS και SEARS – Χρόνος Ολοκλήρωσης (YALPS)…… 93

Εικόνα 6.8: Σύγκριση EARS και SEARS – Χρόνος Ολοκλήρωσης (YALPS)…… 94

Εικόνα 6.9: Σύγκριση EARS και SEARS – Χρόνος Ολοκλήρωσης (YALPS)…… 95

Εικόνα 6.10: Σύγκριση EARS και SEARS – Χρόνος Ολοκλήρωσης (PlanetLab)……...96

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο

Η μετάδοση της πληροφορίας σε σύγχρονα και ασύγχρονα κατανεμημένα συστήματα υπολογιστών αποτελεί έναν πολύ σημαντικό και κρίσιμο παράγοντα έρευνας και μελέτης. Καθώς τα συστήματα διάχυτου υπολογισμού αναπτύσσονται συνεχώς και η ανάγκη για δημιουργία ad-hoc δικτύων όλο και μεγαλώνει, οι έρευνες επικεντρώνονται στον αποτελεσματικότερο και αποδοτικότερο τρόπο μετάδοσης της πληροφορίας σε πληροφοριακά συστήματα.

Με τον όρο κατανεμημένο σύστημα εννοούμε μια συλλογή από γεωγραφικά ανεξάρτητες και αυτόνομες υπολογιστικές οντότητες που επικοινωνούν μεταξύ τους και λειτουργούν συντονισμένα για την επίτευξη ενός κοινού στόχου. Τα πρωτόκολλα πληροφόρησης (gossip protocols) ή επιδημικά μοντέλα (epidemic models) [1] είναι μια ειδική κατηγορία τυχαιοποιημένων αλγορίθμων που χρησιμοποιούνται συχνά για τη διάδοση πληροφοριών (data dissemination) σε κατανεμημένα συστήματα. Οι αλγόριθμοι αυτοί βασίζονται συνήθως σε ένα πολύ απλό τυχαιοποιημένο μηχανισμό επικοινωνίας: ένας κόμβος επιλέγει ένα γειτονικό του βάσει μιας πιθανοτικής κατανομής και μεταδίδει σ' αυτόν όλη την πληροφορία (ή μέρος της) που διαθέτει μέχρι εκείνη τη στιγμή. Η διαδικασία αυτή συνεχίζεται μέχρις ότου ένα ικανοποιητικό ποσοστό κόμβων του συστήματος ή και όλοι έχουν ενημερωθεί για μια συγκεκριμένη πληροφορία.

Βασικό πρόβλημα των επιδημικών αλγορίθμων είναι ότι πολλές φορές, λόγω της τυχαιοποιημένης επικοινωνίας, οι κόμβοι δεν μπορούν να γνωρίζουν ποιες πληροφορίες είναι ήδη γνωστές στο γείτονά τους και ποιες όχι με αποτέλεσμα να οδηγούνται σε άσκοπη μετάδοση πληροφοριών. Σχετικές έρευνες που έχουν γίνει σε επιδημικούς αλγόριθμους [1], έδειξαν ότι το ποσοστό των άσκοπων πληροφοριών που μεταδίδονται μπορεί να περιοριστεί σημαντικά και μάλιστα χωρίς να επηρεάζεται αρνητικά η αποδοτικότητα του αλγορίθμου ούτε η ανεκτικότητά του σε σφάλματα. Συγκεκριμένα, αναπτύχθηκαν επιδημικοί αλγόριθμοι πληροφόρησης οι οποίοι μπορούν να ενημερώνουν όλους τους κόμβους του δικτύου σε O(polylog(n)) γύρους επικοινωνίας και με O(n polylog(n)) μηνύματα, όπου *n* είναι ο αριθμός των κόμβων, διατηρώντας παράλληλα την ευρωστία[1] τους. Σε σχετική διπλωματική εργασία που εκπονήθηκε [2], η οποία αφορούσε στην πειραματική αξιολόγηση τριών τέτοιων αλγορίθμων πληροφόρησης, του PUSH, του PUSH & PULL και του MEDIAN-COUNTER, αποδείχθηκε ότι η θεωρητική προσέγγιση της πολυπλοκότητας των αλγορίθμων επαληθεύεται και σε πρακτικό επίπεδο. Συνεπώς, αφού υπάρχουν επιδημικοί αλγόριθμοι που είναι εύρωστοι τότε μπορούν να υπάρξουν και αποδοτικά κατανεμημένα συστήματα στα οποία όλοι οι κόμβοι θα είναι ενημερωμένοι και συνεπείς ανά πάσα στιγμή.

Οι επιδημικοί αλγόριθμοι, λοιπόν, αποτελούν ένα πολύ ισχυρό εργαλείο μελέτης της διάχυσης της πληροφορίας σε κατανεμημένα συστήματα. Αρκετοί αλγόριθμοι πληροφόρησης έχουν ήδη μελετηθεί σε διάφορα κατανεμημένα περιβάλλοντα, όπως στη συνέπεια των αντιγράφων των βάσεων δεδομένων σε ιστοσελίδες (database consistency) [4], στην ανίχνευση σφαλμάτων (failure detection) [5], στη διάχυση πληροφοριών από ομάδες πολλαπλής διανομής (group multicast) [6], στο πρόβλημα της ομοφωνίας (consensus problem) [7] και στην τοποθέτηση δικτυακών πόρων (resource location) [8].

Ωστόσο, σχεδόν όλες οι θεωρητικές έρευνες έχουν επικεντρωθεί σε σύγχρονα συστήματα, συστήματα δηλαδή που οι κόμβοι λειτουργούν σε συγχρονισμένους γύρους υπολογισμού. Πιο σημαντική, όμως, είναι η μελέτη της περίπτωσης ασύγχρονων

[1] Ο όρος “Εύρωστος Αλγόριθμος” εδώ χρησιμοποιείται για να εκφράσει τον αλγόριθμο εκείνο που είναι ανεκτικός σε σφάλματα και ταυτόχρονα αποδοτικός ως προς την πολυπλοκότητα χρόνου και μηνύματος (βλ. επεξήγηση στα Υποκεφάλαια 2.2 και 4.1) .

κατανεμημένων συστημάτων, δηλαδή συστημάτων όπου δεν μπορεί να διασφαλιστεί ότι η παράδοση των μηνυμάτων διεκπεραιώνεται μέσα σε ένα δεδομένο πεπερασμένο χρονικό διάστημα και οι κόμβοι λειτουργούν σε διαφορετικές ταχύτητες. Ειδικά τα τελευταία χρόνια που η χρήση του Διαδικτύου είναι η πλέον διαδεδομένη, προβάλλεται επιτακτική η ανάγκη για εφαρμογή εύρωστων και αποτελεσματικών αλγορίθμων πληροφόρησης που να διατηρούν την ευρωστία τους ακόμα και σε ασύγχρονα κατανεμημένα συστήματα.

Έρευνα [3] που έχει γίνει σε ασύγχρονους αλγόριθμους πληροφόρησης απέδειξε ότι ακόμη και απέναντι σε σφάλματα κατάρρευσης κόμβων μια πληροφορία μπορεί τελικά να διαδοθεί στους κόμβους του δικτύου χωρίς να επηρεαστεί σημαντικά η αποδοτικότητα των αλγορίθμων. Συγκεκριμένα, σχεδιάστηκαν τρεις εύρωστοι αλγόριθμοι οι οποίοι μπορούν να ενημερώνουν τους κόμβους του δικτύου σε λογαριθμικό ή και σταθερό χρόνο, αποστέλλοντας λογαριθμικό αριθμό μηνυμάτων με μεγάλη πιθανότητα. Οι αλγόριθμοι αυτοί όμως, αν και αναλύθηκαν θεωρητικά [3] εντούτοις δεν έχουν δοκιμαστεί σε πρακτικό επίπεδο και αυτό αποτέλεσε το κίνητρο που οδήγησε στην εκπόνηση της παρούσας εργασίας.

1.2 Προσφορά Διατριβής

Η εργασία αυτή ασχολήθηκε με την υλοποίηση και πειραματική αξιολόγηση των αλγορίθμων πληροφόρησης που αναπτύχθηκαν στο [3]. Συγκεκριμένα, αξιολογήθηκαν πειραματικά ως προς την πολυπλοκότητα του χρόνου που χρειάζεται ώστε να ενημερωθούν τελικά όλοι οι κόμβοι του δικτύου που δεν κατάρρευσαν, αλλά και ως προς τον αριθμό των μηνυμάτων που απαιτείται να σταλούν στο δίκτυο για το σκοπό αυτό. Για την υλοποίηση των αλγορίθμων χρησιμοποιήθηκε η βιβλιοθήκη YALPS [9], μία ανοικτού κώδικα βιβλιοθήκη της Java, η οποία έχει σχεδιαστεί ακριβώς για να διευκολύνει την ανάπτυξη κατανεμημένων εφαρμογών. Η πειραματική αξιολόγησή τους έγινε πρώτα σε επίπεδο προσομοίωσης, ενώ στη συνέχεια σε πραγματικό κατανεμημένο σύστημα και συγκεκριμένα στο PlanetLab [10], μια ανοικτή πλατφόρμα για την ανάπτυξη και τη χρήση υπηρεσιών δικτύου παγκόσμιας

κλίμακας. Από τα αποτελέσματα αποδεικνύεται η πρακτικότητα των αλγορίθμων και διαφαίνεται σε πόσο μεγάλο βαθμό η θεωρητική ανάλυση και πολυπλοκότητα των αλγορίθμων συνάδει με την εφαρμογή τους σε πραγματικά ασύγχρονα κατανεμημένα περιβάλλοντα.

Οι δύο αλγόριθμοι πληροφόρησης με τους οποίους ασχολείται η εργασία αυτή βασίζονται στο επιδημικό μοντέλο. Κάθε κόμβος στο δίκτυο έχει τη δική του στατική πληροφορία η οποία με την ολοκλήρωση του αλγορίθμου πρέπει να έχει μεταδοθεί σε όλους τους υπόλοιπους κόμβους του δικτύου. Δηλαδή αν υπάρχουν *n* κόμβοι στο δίκτυο τότε υπάρχουν και *n* πληροφορίες που πρέπει να μεταδοθούν. Παράλληλα ο κάθε κόμβος διατηρεί μία τοπική λίστα με όλες τις πληροφορίες που έχει συλλέξει και γνωρίζει ανά πάσα στιγμή. Η βασική δομή των αλγορίθμων είναι η ίδια ενώ υπάρχουν διαφοροποιήσεις όσον αφορά στον αριθμό των κόμβων που επιλέγονται τυχαία σε κάθε γύρο επικοινωνίας για να τους σταλεί η πληροφορία καθώς επίσης και στις συνθήκες τερματισμού του κάθε αλγορίθμου.

Ο πρώτος αλγόριθμος που υλοποιείται είναι ο αλγόριθμος EARS (Epidemic Asynchronous Rumor Spreading). Σε κάθε γύρο επικοινωνίας ένας κόμβος επιλέγει τυχαία κάποιο άλλο κόμβο και αποστέλλει σ’ αυτόν ένα μήνυμα – λίστα το οποίο περιέχει όλες τις πληροφορίες που έχει συλλέξει μέχρι εκείνη τη στιγμή. Αν ο κόμβος λάβει ένα μήνυμα από κάποιο άλλο κόμβο τότε ο κόμβος – παραλήπτης ενημερώνει την τοπική του λίστα με τις νέες πληροφορίες που του προσφέρει ο κόμβος – αποστολέας. Η ίδια διαδικασία επαναλαμβάνεται τόσες φορές όσες εκτιμάται ότι χρειάζεται να εφαρμοστεί έτσι ώστε με μεγάλη πιθανότητα οι πληροφορίες να έχουν διασκορπιστεί επαρκώς στο δίκτυο. Αν όμως κατά τη διάρκεια διαπιστωθεί ότι κάποια πληροφορία δεν έχει διαδοθεί ακόμη τότε ξαναρχίζει όλη η διαδικασία από την αρχή.

Ο δεύτερος αλγόριθμος που υλοποιείται είναι ο αλγόριθμος SEARS (Spamming Epidemic Asynchronous Rumor Spreading). Η κύρια διαφορά του από τον αλγόριθμο EARS είναι ότι στον ίδιο γύρο επικοινωνίας επιλέγονται περισσότεροι από ένας κόμβοι για να τους σταλεί ένα μήνυμα. Συνεπώς, στον αλγόριθμο SEARS σε κάθε γύρο επικοινωνίας στέλλονται περισσότερα μηνύματα και όχι μόνο ένα όπως γίνεται στον αλγόριθμο EARS. Η δεύτερη

διαφορά του αλγορίθμου από τον EARS είναι ότι δε λαμβάνεται υπόψη αν κάποια πληροφορία δεν έχει διαδοθεί ακόμη.

Αυτή η εργασία εκτιμάται ότι συνεισφέρει σημαντικά στον τομέα της διάχυσης της πληροφορίας σε ασύγχρονα κατανεμημένα συστήματα, καθ' ότι οι εν λόγω αλγόριθμοι είναι οι μοναδικοί, προς το παρόν, που έχουν αναπτυχθεί και αυστηρότυπα αναλυθεί για ασύγχρονα κατανεμημένα συστήματα. Επομένως το γεγονός αυτό κάνει και την πειραματική αξιολόγησή τους ακόμη πιο σημαντική.

1.3 Δομή της Διατριβής

Η παρούσα διατριβή Μάστερ αποτελείται συνολικά από επτά κεφάλαια ως εξής:

Στο Κεφάλαιο 2 παρουσιάζεται το υπόβαθρο που χρειάζεται να έχει κανείς για να κατανοήσει καλύτερα το πρόβλημα της πληροφόρησης και την υλοποίηση των αλγορίθμων που το επιλύουν. Συγκεκριμένα, ορίζονται κάποιες σχετικές έννοιες, περιγράφονται γενικά οι αλγόριθμοι της πληροφόρησης και το επιδημικό μοντέλο και γίνεται μια σύντομη αλλά περιεκτική αναφορά για τη βιβλιοθήκη YALPS που χρησιμοποιείται στην υλοποίηση των αλγορίθμων.

Στο Κεφάλαιο 3 παρουσιάζονται οι αλγόριθμοι EARS και SEARS οι οποίοι επιλύουν το πρόβλημα μετάδοσης πληροφοριών σε ασύγχρονα κατανεμημένα συστήματα. Δίνεται μια περιεκτική περιγραφή του κάθε αλγορίθμου, ώστε να γίνει κατανοητό το μοντέλο που χρησιμοποιείται σε κάθε περίπτωση, και αναφέρονται λεπτομέρειες υλοποίησης και δομής του κώδικα που δημιουργήθηκε για τον κάθε αλγόριθμο.

Στο Κεφάλαιο 4 παρουσιάζεται η διαδικασία της πειραματικής αξιολόγησης του κάθε αλγορίθμου, οι μετρικές αξιολόγησης που χρησιμοποιήθηκαν και οι γραφικές παραστάσεις των μετρήσεων που πάρθηκαν μετά από εκτελέσεις του κάθε αλγορίθμου σε περιβάλλον προσομοίωσης. Στο τέλος παρατίθεται σχολιασμός των αποτελεσμάτων και εξάγονται

συμπεράσματα που υποδεικνύουν σε ποιο βαθμό η θεωρητική ανάλυση του κάθε αλγορίθμου συνάδει με την πειραματική.

Παρομοίως, στο Κεφάλαιο 5 παρουσιάζεται η διαδικασία της πειραματικής αξιολόγησης του κάθε αλγορίθμου όταν εφαρμοστεί σε ένα πραγματικό κατανεμημένο δίκτυο, όπως είναι το PlanetLab. Λαμβάνονται γραφικές παραστάσεις των αποτελεσμάτων και γίνεται σύγκριση με αυτές του προηγούμενου κεφαλαίου.

Ακολούθως στο Κεφάλαιο 6 γίνεται σύγκριση των δύο αλγορίθμων και καταγράφονται σχετικά συμπεράσματα. Πρώτα γίνεται σύγκριση ως προς τον αριθμό των μηνυμάτων που χρειάζεται να αποσταλούν από κάθε αλγόριθμο προκειμένου να επιλυθεί το πρόβλημα μετάδοσης πληροφοριών σε όλους τους κόμβους του δικτύου. Στη συνέχεια γίνεται σύγκριση ως προς το χρόνο που χρειάζεται ο κάθε αλγόριθμος ξεχωριστά για να επιλύσει το πρόβλημα.

Στο Κεφάλαιο 7 συνοψίζονται τα συμπεράσματα από την εφαρμογή των δύο αλγορίθμων τόσο σε περιβάλλον προσομοίωσης όσο και σε πραγματικό κατανεμημένο περιβάλλον. Αναφέρονται τυχόν δυσκολίες που είχαν παρουσιαστεί κατά την εκπόνηση της εργασίας και διατυπώνονται ιδέες για μελλοντική εργασία και εκμετάλλευση των συμπερασμάτων.

Τέλος, η εργασία κλείνει με τα παραρτήματα Α και Β, στα οποία παρουσιάζεται ο κώδικας υλοποίησης των αλγορίθμων EARS και SEARS αντίστοιχα στο προγραμματιστικό περιβάλλον Java και με τη χρήση της βιβλιοθήκης YALPS, και με το παράρτημα Γ στο οποίο παρουσιάζονται παραδείγματα εκτέλεσης ενός από τους δύο αλγορίθμους, του αλγόριθμου EARS.

Κεφάλαιο 2

Υπόβαθρο

2.1 Το Πρόβλημα της Πληροφόρησης

Σε κάθε κατανεμημένο σύστημα κρίσιμος παράγοντας για τη σωστή διαχείριση και αξιοποίησή του είναι η επίλυση του προβλήματος διάχυσης πληροφοριών στους κόμβους του δικτύου, γνωστό και ως το πρόβλημα της πληροφόρησης (gossip problem) [1, 3]. Κάθε κόμβος αρχικά διαθέτει μια τοπική πληροφορία που ονομάζεται "φήμη" (rumor)[2] και το πρόβλημα που προσπαθεί να επιλύσει είναι να μάθει τις φήμες όλων των υπόλοιπων κόμβων του δικτύου. Η τεχνική που επικράτησε ως η καλύτερη και αποδοτικότερη λύση στο πρόβλημα είναι ο κάθε κόμβος περιοδικά να αποστέλλει τη δική του φήμη – μαζί με οποιεσδήποτε άλλες φήμες έχει μάθει μέχρι εκείνη τη στιγμή – σε έναν άλλο τυχαία επιλεγμένο κόμβο. Ένα μοντέλο που κάνει χρήση μια τέτοιας τεχνικής συχνά καλείται επιδημικό μοντέλο καθώς μιμείται την τεχνική διάδοσης μιας επιδημίας (epidemic) όπως ακριβώς είναι γνωστή από το βιολογικό πεδίο εφαρμογής. Δηλαδή μία πληροφορία μεταδίδεται όπως ένας ιός εξαπλώνεται σε ένα βιολογικό πληθυσμό. Αντίστοιχο και ανάλογο παράδειγμα μπορεί να επικαλεστεί κάποιος για τη διάδοση μίας φήμης μεταξύ ανθρώπων που κουτσομπολεύουν (gossiping).

Υπάρχουν διάφορες κλάσεις πρωτοκόλλων πληροφόρησης που βασίζονται στην επιδημική διάδοση της πληροφορίας (Epidemic Information Dissemination) σε κατανεμημένα συστήματα [11]. Οι πιο γνωστές είναι τα πρωτόκολλα διάχυσης (dissemination ή rumor–

[2] Ο όρος "rumor" χρησιμοποιείται για να εκφράσει όλη την πληροφορία που διαθέτει ένας κόμβος σε τοπικό επίπεδο, και πρέπει να διαδοθεί στους υπόλοιπους.

mongering protocols), τα πρωτόκολλα αντι–εντροπίας και τα πρωτόκολλα συνδυασμένης διάδοσης. Στην πρώτη κλάση κάθε κόμβος που παραλαμβάνει μια νέα πληροφορία επιλέγει περιοδικά και τυχαία έναν άλλο κόμβο στο δίκτυο και την στέλλει σ' αυτόν. Στη δεύτερη κλάση κάθε κόμβος που παραλαμβάνει μια νέα πληροφορία ανανεώνει πρώτα την τοπική του γνώση με αυτήν την πληροφορία, μετά επιλέγει τακτικά και τυχαία έναν άλλο κόμβο στο δίκτυο και στέλλει σ' αυτόν όλη την πληροφορία που γνωρίζει μέχρι εκείνη τη στιγμή. Στην τρίτη και τελευταία κλάση πρωτοκόλλων συλλέγονται πληροφορίες από όλους τους κόμβους για στοιχεία που αφορούν ολόκληρο το δίκτυο και αφού αυτές συνδυαστούν καταλήγουν σε μία πληροφορία που αφορά όλο το δίκτυο.

Η παρούσα εργασία ασχολείται με το πρόβλημα της πληροφόρησης σε ασύγχρονα κατανεμημένα συστήματα και οι αλγόριθμοι που μελετά βασίζονται σε ένα συνδυασμό των πρώτων δύο κλάσεων πρωτοκόλλων πληροφόρησης.

2.2 Αλγόριθμοι Πληροφόρησης

Οι αλγόριθμοι πληροφόρησης έχουν ως τελικό στόχο να ενημερωθούν όλοι οι κόμβοι σε ένα δίκτυο με όλες τις πληροφορίες που υπάρχουν στο δίκτυο, δηλαδή ο κάθε κόμβος να διαδώσει αντίγραφα της τοπικής του πληροφορίας σε όλους τους υπόλοιπους κόμβους του δικτύου. Τέτοιοι αλγόριθμοι είναι από τη φύση τους πιθανοτικοί, αφού ο κάθε κόμβος επιλέγει τυχαία τον κόμβο στον οποίο θα στείλει την τοπική του γνώση.

Οι αλγόριθμοι δουλεύουν σε γύρους επικοινωνίας όπου ο κάθε γύρος αποτελείται από δύο φάσεις. Στην πρώτη φάση ο κάθε κόμβος επιλέγει τυχαία έναν κόμβο από το δίκτυο και αφού εγκαταστήσει τις απαραίτητες συνδέσεις για να μπορέσει να επικοινωνήσει μαζί του, στη συνέχεια στέλλει σ' αυτόν ένα μήνυμα με όλη την πληροφορία που είναι αποθηκευμένη στην τοπική του κρυφή μνήμη (cache) τη συγκεκριμένη στιγμή. Στη δεύτερη φάση κάθε κόμβος που λαμβάνει πληροφορίες από κάποιο κόμβο του δικτύου, τις καταχωρεί στην

τοπική του κρυφή μνήμη και με βάση τα νέα δεδομένα ανανεώνει τις πληροφορίες που θα διαδώσει στον επόμενο γύρο.

Για τη μελέτη της αποδοτικότητας ενός αλγορίθμου πληροφόρησης τα μέτρα αξιολόγησης που συνήθως χρησιμοποιούνται είναι η *πολυπλοκότητα μηνύματος*, δηλαδή ο συνολικός αριθμός των μηνυμάτων που αποστέλλονται κατά την εκτέλεση και η *πολυπλοκότητα χρόνου*, δηλαδή ο χρόνος (γύροι επικοινωνίας) που χρειάζεται ο αλγόριθμος για να ολοκληρώσει τη λειτουργία του. Κάθε αλγόριθμος πληροφόρησης θεωρείται εύρωστος και αποτελεσματικός όταν πετυχαίνει βελτιστοποίηση των πιο πάνω μέτρων.

Υπάρχουν δύο βασικές τεχνικές διάδοσης των φημών που χρησιμοποιούνται για το σκοπό αυτό: στην πρώτη οι διεργασίες – κόμβοι βασίζονται στη διαδοχή μηνυμάτων σε μια προσπάθεια να στείλουν όσο το δυνατό λιγότερα μηνύματα και παράλληλα σε αρκετά ικανοποιητικό χρόνο, ενώ στη δεύτερη οι διεργασίες αποστέλλουν περισσότερα μηνύματα σε μια προσπάθεια να κατανεμηθούν με γοργό ρυθμό οι φήμες τους στο σύστημα. Και οι δύο τεχνικές είναι σημαντικές. Η τεχνική ελαχιστοποίησης των μηνυμάτων οδηγεί σε αλγόριθμους αποδοτικούς ως προς την πολυπλοκότητα των μηνυμάτων που ανταλλάζονται, ενώ η τεχνική της γοργής διάδοσης των φημών οδηγεί σε σταθερού χρόνου αλγόριθμους, δηλαδή αλγόριθμους που ολοκληρώνουν τη διάδοση των φημών σε σταθερό και σχετικά μικρό χρόνο. Στην παρούσα εργασία οι δύο αλγόριθμοι που μελετήθηκαν, ο EARS και ο SEARS, κάνουν χρήση της πρώτης και δεύτερης τεχνικής αντίστοιχα.

2.3 Μοντέλα Παρεμπόδισης Διάδοσης Πληροφοριών

Για την αξιολόγηση ενός αλγορίθμου πληροφόρησης και τη σύγκρισή του με άλλους παρόμοιους αλγορίθμους δεν αρκεί να εξεταστεί μόνο η αποτελεσματικότητά του, δηλαδή σε ποιο βαθμό επιλύει το πρόβλημα πληροφόρησης. Είναι εξίσου σημαντικό να διερευνηθεί η αποδοτικότητά του και ακόμη πιο σημαντικό να δοκιμαστεί ο βαθμός ανοχής του σε σφάλματα δικτύου και άλλους πιθανούς «εχθρούς» του συστήματος. Ένα κατανεμημένο

σύστημα λέγεται ότι ανέχεται σφάλματα αν είναι σχεδιασμένο με τέτοιο τρόπο ώστε να λειτουργεί ορθά ακόμα κι αν κάποια συστατικά του συστήματος αποτύχουν στο να λειτουργούν βάσει της προδιαγραφής τους, χωρίς να επηρεάζεται σοβαρά η συνολική απόδοση.

Υπάρχουν διάφορα είδη σφαλμάτων τα οποία ταξινομούνται ως προς τη δριμύτητά τους (severity taxonomy) και τη διάρκειά τους (temporal taxonomy). Στην πρώτη κατηγορία συγκαταλέγονται τα σφάλματα κατάρρευσης (crash faults), τα σφάλματα παράλειψης (omission faults), τα σφάλματα χρονισμού (timing faults) και τα βυζαντινά σφάλματα (Byzantine faults). Στη δεύτερη κατηγορία ανήκουν τα παροδικά σφάλματα (transient faults), τα διαλείποντα (intermittent faults) και τα μόνιμα (permanent faults) σφάλματα. Ωστόσο η παρούσα εργασία ασχολείται μόνο με τα σφάλματα κατάρρευσης καθώς αυτά παρατηρούνται πιο συχνά σε συστήματα υπολογιστών. Ένα σφάλμα κατάρρευσης συμβαίνει όταν ένας επεξεργαστής σταματά χωρίς προειδοποίηση οποιαδήποτε λειτουργία του. Εάν ο επεξεργαστής αυτός ήταν συνδεδεμένος σε κάποιο δίκτυο τότε ο σύνδεσμος σταματά να παραλαμβάνει, να προωθεί και να παραδίδει μηνύματα. Περισσότερες πληροφορίες για τα σφάλματα κατάρρευσης καθώς και για τα άλλα είδη σφαλμάτων μπορούν να ανακτηθούν από το [12].

Υπάρχουν δύο μοντέλα παρεμπόδισης της διάδοσης των φημών τα οποία μπορούν να εφαρμοστούν κατά την πειραματική αξιολόγηση ενός αλγορίθμου πληροφόρησης για να εξεταστεί κατά πόσο ο υπό μελέτη αλγόριθμος είναι πράγματι εύρωστος ή όχι [3].

Το πρώτο μοντέλο, το οποίο παρέχει την ασθενέστερη παρεμπόδιση, είναι ο αφελής αντίπαλος (oblivious adversary). Έχει γνώση του κώδικα του αλγορίθμου και το τι κάνει αλλά δεν μπορεί να γνωρίζει τα τυχαιοποιημένα αποτελέσματα κατά την εκτέλεσή του. Γι’ αυτό καθορίζει πριν την εκτέλεση του κώδικα μια συγκεκριμένη συμπεριφορά παρεμπόδισης κάποιου αριθμού κόμβων από το να αποστέλλουν μηνύματα, με λίγα λόγια τους καταρρέει.

Το δεύτερο μοντέλο, το οποίο παρέχει την ισχυρότερη παρεμπόδιση, είναι ο δυναμικός αντίπαλος (adaptive adversary). Πρόκειται για έναν πιο ισχυρό αντίπαλο ο οποίος δρα κατά την εκτέλεση του αλγορίθμου αποφασίζοντας δυναμικά εκείνη τη στιγμή πώς να ενεργήσει

ανάλογα με την εξέλιξη του συστήματος. Σε αντίθεση με τον αφελή αντίπαλο, γνωρίζει τα πάντα για το πρόγραμμα που τρέχει και παρακολουθεί συνεχώς την εκτέλεση κάνοντας την εμφάνισή του σε καίρια σημεία για τη διάδοση των φημών.

Στο [3] αποδείχθηκε ότι το πρόβλημα της πληροφόρησης δεν μπορεί να επιλυθεί αποδοτικά σε ασύγχρονα κατανεμημένα συστήματα στην παρουσία ισχυρού αντιπάλου και γι' αυτό το λόγο η παρούσα εργασία ασχολείται μόνο με το πρώτο μοντέλο παρεμπόδισης διάδοσης των φημών και για τους συγκεκριμένους αλγόριθμους κάνει διερεύνηση του βαθμού ανοχής τους στην παρουσία αφελών μόνο αντιπάλων.

2.4 Ο Μηχανισμός Διάχυσης της Πληροφορίας σε Ασύγχρονο Περιβάλλον

Όπως έχει αναφερθεί και προηγουμένως, ο μηχανισμός διάχυσης της πληροφορίας (gossiping) έχει χρησιμοποιηθεί σε πολλές διαφορετικές περιοχές των κατανεμημένων συστημάτων [4, 5, 6, 7, 8] και έχουν αναπτυχθεί αρκετοί αλγόριθμοι πληροφόρησης για την κάθε περιοχή. Ωστόσο σχεδόν όλες οι σχετικές έρευνες που έχουν γίνει εστιάζονται μόνο στα σύγχρονα συστήματα. Παρόλο που θα μπορούσε να ισχυριστεί κάποιος ότι οι κατανεμημένες εφαρμογές τις περισσότερες φορές συμπεριφέρονται με σύγχρονο τρόπο τελικά, εντούτοις είναι σαφώς πιο επιθυμητό να σχεδιάζονται αλγόριθμοι που να μπορούν να ανεχτούν και τη χείριστη περίπτωση και να είναι ταυτόχρονα αποδοτικοί (βλ. επεξήγηση για τα *μέτρα αξιολόγησης αποδοτικότητας* στο Υποκεφάλαιο 2.2). Με άλλα λόγια, επιβάλλεται να αναπτυχθούν αλγόριθμοι πληροφόρησης που να μπορούν να διατηρούν την αποδοτικότητά τους ακόμα και σε ασύγχρονα συστήματα, δηλαδή σε συστήματα όπου δεν υπάρχουν προκαθορισμένα όρια στην καθυστέρηση αποστολής ενός μηνύματος ή στην ταχύτητα του κάθε επεξεργαστή – κόμβου του δικτύου [3].

Έστω ένα ασύγχρονο κατανεμημένο σύστημα αποτελούμενο από n επεξεργαστές – κόμβους με άγνωστη καθυστέρηση d στην αποστολή ενός μηνύματος σε κάποιο κόμβο και άγνωστη καθυστέρηση δ στην ταχύτητα του επεξεργαστή. Σημειώνεται ότι για σκοπούς

ανάλυσης, θεωρούνται κάποια d και δ τα οποία όμως δε γνωρίζουν οι κόμβοι του συστήματος. Ο κάθε κόμβος δύναται να καταρρεύσει οποιαδήποτε χρονική στιγμή διακόπτοντας οριστικά τη λειτουργία του και κατά συνέπεια σταματώντας να αποστέλλει και να λαμβάνει μηνύματα. Το πρόβλημα θεωρείται ότι έχει λυθεί ορθά στην παρουσία σφαλμάτων κατάρρευσης, όταν μετά το τέλος της εκτέλεσης όλοι οι κόμβοι που δεν κατάρρευσαν έχουν μάθει τις φήμες όλων των υπόλοιπων κόμβων που επίσης δεν κατάρρευσαν. Ένα απλό μοντέλο, εμπνευσμένο από το επιδημικό μοντέλο, που επιλύει ορθά το πρόβλημα της πληροφόρησης θα μπορούσε να είναι το εξής: κάθε κόμβος στέλλει περιοδικά (περίοδος πληροφόρησης) τη δική του φήμη, μαζί με όλες τις φήμες που έχει μάθει μέχρι στιγμής, σε έναν άλλο κόμβο του δικτύου τον οποίο έχει επιλέξει τυχαία.

Αμέσως προκύπτουν δύο ερωτήματα: πόσο συχνά πρέπει ο κόμβος να αποστέλλει τις φήμες του και πότε πρέπει να σταματήσει τη διάδοση. Στα σύγχρονα συστήματα η απάντηση στα δύο ερωτήματα μπορεί να δοθεί εύκολα: κάθε κόμβος στέλλει ένα μήνυμα σε κάθε γύρο επικοινωνίας και διακόπτει τη διάδοση μετά από έναν επαρκή αριθμό γύρων επικοινωνίας. Στα ασύγχρονα συστήματα όμως για να δουλέψει το πιο πάνω μοντέλο μία λύση θα ήταν η περίοδος πληροφόρησης να μη βασίζεται στις καθυστερήσεις d και δ αλλά σε ένα τοπικό μετρητή που θα καθόριζε σε κάθε πόσα βήματα να διαδίδεται η πληροφορία. Ωστόσο το ερώτημα για το πότε πρέπει να διακόπτεται οριστικά η διάδοση, παραμένει. Ο λόγος είναι ότι, επειδή το σύστημα είναι ασύγχρονο, υπάρχει δυσκολία στη διάκριση μεταξύ των κόμβων που έχουν καταρρεύσει και αυτών που είναι αυθαίρετα αργοί. Από την άλλη, σε αντίθεση με τα σύγχρονα συστήματα, δεν είναι αρκετό να καθοριστεί ένας ικανοποιητικός αριθμός γύρων επικοινωνίας διότι, λόγω των μη σταθερών καθυστερήσεων d και δ, υπάρχει ο κίνδυνος να μην διαδοθούν επαρκώς τα δεδομένα και κατά συνέπεια να μην επιλυθεί ορθά το πρόβλημα της πληροφόρησης. Στην παρούσα εργασία οι αλγόριθμοι που μελετούνται επιλύουν ορθά το πρόβλημα της πληροφόρησης και υποθέτουν ένα πλήρως ασύγχρονο κατανεμημένο περιβάλλον με πιθανότητα εμφάνισης σφαλμάτων κατάρρευσης,

Όσον αφορά στην αποδοτικότητα του μηχανισμού διάχυσης της πληροφορίας, σχετική έρευνα που διεξήχθη [3] απέδειξε ότι στην παρουσία δυναμικού αντιπάλου κανένας

τυχαιοποιημένος αλγόριθμος πληροφόρησης δεν μπορεί να είναι ταυτόχρονα αποδοτικός ως προς τον αριθμό των μηνυμάτων που ανταλλάζονται και αποδοτικός ως προς το χρόνο ολοκλήρωσης της διάδοσης των φημών. Αυτό είναι το κόστος του ασυγχρονισμού: ένας ασύγχρονος αλγόριθμος πληροφόρησης κάτω από ένα δυναμικό εχθρό, ο οποίος μπορεί να προκαλέσει μέχρι και $f < n$ σφάλματα κατάρρευσης, επιλύει ορθά το πρόβλημα με μεγάλη πιθανότητα είτε σε χρόνο $\Omega(f(d+\delta))$ είτε αποστέλλοντας $\Omega(n+f^2)$ μηνύματα. Δηλαδή ένας ασύγχρονος αλγόριθμος πληροφόρησης, όσο και να βελτιωθεί σε σχέση με τον τετριμμένο[3] αλγόριθμο, πάλι θα απαιτεί χρόνο γραμμικό στο f, όπου f ο αριθμός των πιθανών σφαλμάτων. Αντιθέτως οι ντετερμινιστικοί σύγχρονοι αλγόριθμοι ολοκληρώνουν τη διάδοση των rumors σε μόνο O(polylog(n)) γύρους επικοινωνίας αποστέλλοντας μόνο O(n polylog(n)) μηνύματα παρόλο που ανέχονται μέχρι και n-1 σφάλματα [13].

Η ίδια έρευνα [3] όμως έδειξε ότι απέναντι σε έναν αφελή αντίπαλο τα πράγματα είναι διαφορετικά. Υπάρχουν ασύγχρονοι αλγόριθμοι πληροφόρησης οι οποίοι μπορούν να είναι αποδοτικοί ως προς το χρόνο ολοκλήρωσής τους (αριθμό γύρων επικοινωνίας) και ταυτόχρονα αποδοτικοί ως προς τον αριθμό των μηνυμάτων που ανταλλάζονται. Συγκεκριμένα, υπάρχει ο αλγόριθμος EARS ο οποίος συνδυάζει το επιδημικό μοντέλο με το μοντέλο ελέγχου προόδου (progress control scheme) και συλλέγει επιπρόσθετες πληροφορίες οι οποίες είναι απαραίτητες για να αποφασιστεί πότε να τερματιστεί η διάδοση των φημών. Έτσι αποφεύγει την άσκοπη μετάδοση μηνυμάτων. Ο αλγόριθμος αυτός ολοκληρώνει τη διάδοση σε χρόνο $O\left(\frac{n}{n-f}\log^2 n(d+\delta)\right)$ αποστέλλοντας $O(n \log^3 n\,(d+\delta))$ μηνύματα με μεγάλη πιθανότητα. Έχει, δηλαδή, αποδοτικότητα συγκρίσιμη με αυτήν των καλύτερων σύγχρονων αλγορίθμων πληροφόρησης. Επιπλέον, υπάρχει ο αλγόριθμος SEARS ο οποίος αποκλίνει από το επιδημικό μοντέλο στέλλοντας περισσότερα μηνύματα σε κάθε περίοδο πληροφόρησης κι όχι μόνο ένα όπως στο κλασικό μοντέλο. Αυτός πετυχαίνει σταθερό χρόνο ολοκλήρωσης της διάδοσης των φημών της τάξεως $O\left(\frac{n}{\varepsilon(n-f)}(d+\delta)\right)$ ενώ αποστέλλει αριθμό μηνυμάτων της

[3] Ο «τετριμμένος αλγόριθμος πληροφόρησης» είναι ο πιο απλός αλγόριθμος πληροφόρησης. Σύμφωνα με αυτόν, ο κάθε κόμβος στέλλει τη δική του φήμη απευθείας σε κάθε άλλο κόμβο. Ο αλγόριθμος αυτός ολοκληρώνει τη λειτουργία του σε χρόνο $O(d+\delta)$ αποστέλλοντας $\Theta(n^2)$ μηνύματα όπου d, δ οι καθυστερήσεις και n ο αριθμός των κόμβων του συστήματος.

τάξεως $O\left(\frac{n^{2+\varepsilon}}{\varepsilon(n-f)}\log n\,(d+\delta)\right)$ όπου ε < 1. Τέλος, υπάρχει ο αλγόριθμος TEARS ο οποίος επιλύει μια πιο αδύνατη εκδοχή του μηχανισμού διάδοσης των φημών, το λεγόμενο πρόβλημα της πλειοψηφικής πληροφόρησης (majority gossip). Σύμφωνα με αυτό δε χρειάζεται να επιλυθεί το πρόβλημα της πληροφόρησης πλήρως, δηλαδή να ενημερωθούν όλοι οι κόμβοι με όλες τις φήμες αλλά αρκεί ο καθένας να γνωρίζει την πλειοψηφία των φημών. Ο αλγόριθμος αυτός έχει επίσης σταθερό χρόνο όπως ο SEARS, της τάξεως O(d+δ) και αποστέλλει αριθμό μηνυμάτων ανεξάρτητο των d και δ, της τάξεως $O\left(n^{7/4}log^2 n\right)$. Για να είναι κατορθωτή η επίλυση του προβλήματος της πλειοψηφικής πληροφόρησης απαιτείται ο αριθμός των πιθανών καταρρεύσεων να μην ξεπερνά το $\frac{n}{2}$, όπου *n* ο συνολικός αριθμός των κόμβων του συστήματος. Για σκοπούς απλοποίησης της ανάλυσης γίνεται η επιπλέον υπόθεση ότι το *n* είναι αρκετά μεγάλο. Όμως το γεγονός αυτό δεν επέτρεψε την πειραματική αξιολόγηση του εν λόγω αλγόριθμου καθώς απαιτούσε αρκετά μεγάλο αριθμό επεξεργαστών στην περίπτωση εφαρμογής του στο Planet-Lab και επίσης επεξεργαστή με μεγάλη υπολογιστική δύναμη στην περίπτωση προσομοίωσής του τοπικά. Έτσι η εργασία αυτή ασχολήθηκε μόνο με τους δύο πρώτους αλγόριθμους και περισσότερες λεπτομέρειες για αυτούς θα αναφερθούν στα επόμενα κεφάλαια.

2.5 Προηγούμενες Εργασίες

Στο [1] έχει αποδειχθεί ότι σε ένα σύγχρονο κατανεμημένο σύστημα μία φήμη μπορεί να διαδοθεί σε O(log *n*) γύρους επικοινωνίας αποστέλλοντας O(*n* loglog *n*) μηνύματα με μεγάλη πιθανότητα. Μία επέκταση του αποτελέσματος αυτού στο [13] απέδειξε ότι μπορούν να υπάρξουν πρωτόκολλα πληροφόρησης τα οποία χρειάζονται μόνο O(polylog (*n*)) γύρους επικοινωνίας και μόνο O(*n* polylog(*n*)) μηνύματα ακόμα και στην παρουσία μέχρι και *n-1* καταρρεύσεων.

Μια πρώτη προσπάθεια μελέτης της ασύγχρονης περίπτωσης έγινε στο [14]. Θεωρήθηκε ένα μερικώς σύγχρονο σύστημα το οποίο όμως έκανε την υπόθεση μίας, γνωστής από την αρχή, πιθανοτικής κατανομής των καθυστερήσεων στην επικοινωνία μεταξύ των κόμβων καθώς επίσης ενός συστήματος χωρίς σφάλματα κατάρρευσης. Μια καλύτερη προσπάθεια έγινε στο [15] όταν θεωρήθηκαν πρωτόκολλα πληροφόρησης σε ένα "ασύγχρονο" περιβάλλον, όπου τα τοπικά ρολόγια μοντελοποιήθηκαν σαν κατανομές Poisson, όμως ούτε εκεί έγινε οποιαδήποτε υπόθεση για πιθανά σφάλματα κατάρρευσης.

Σε αντίθεση με τα [14] και [15], οι αλγόριθμοι πληροφόρησης οι οποίοι αναλύονται αυστηρότυπα στο [3] και τους οποίους αξιολογεί πειραματικά η παρούσα εργασία, υποθέτουν ένα πλήρως ασύγχρονο περιβάλλον με πιθανή την εμφάνιση σφαλμάτων κατάρρευσης κατά την εκτέλεση. Περισσότερες πληροφορίες για προηγούμενες σχετικές εργασίες μπορούν να ανακτηθούν από τα [16] και [17].

2.6 Η Βιβλιοθήκη YALPS

Η βιβλιοθήκη YALPS [6] είναι μια ανοικτού κώδικα βιβλιοθήκη της Java η οποία σχεδιάστηκε πρόσφατα από μια ομάδα ερευνητών του ιδρύματος INRIA και σκοπό έχει τη διευκόλυνση της ανάπτυξης, δοκιμής και ελέγχου κατανεμημένων εφαρμογών. Περιλαμβάνει δύο πλατφόρμες υλοποίησης, τον προσομοιωτή SimulatedNodeStarter και τον εκτελεστή ExecutorNodeStarter. Έτσι οι εφαρμογές που γράφονται με τη χρήση YALPS μπορούν να τρέξουν τόσο σε περιβάλλον προσομοίωσης όσο και σε πραγματικό περιβάλλον και μάλιστα χωρίς να χρειάζεται καμία αλλαγή στον κώδικα εφαρμογής. Μ' αυτόν τον τρόπο δίνεται η δυνατότητα στους ερευνητές, αφού δημιουργήσουν το πρωτότυπο μιας εφαρμογής, να πειραματιστούν και να δοκιμάσουν τα διάφορα σενάριά τους στον προσομοιωτή προτού τα εφαρμόσουν σε πραγματικό σύστημα.

Μία εφαρμογή YALPS αποτελείται από μια συλλογή εργασιών (Tasks) τις οποίες διαχειρίζεται ο διαχειριστής εργασιών (Task Manager). Ο διαχειριστής είναι υπεύθυνος να

καθορίζει με ποιο τρόπο θα εκτελεστεί κάθε μια από τις εργασίες. Υπάρχουν συνολικά τρεις τρόποι εκτέλεσης για μια εργασία: η άμεση εκτέλεση κατά την οποία η εργασία εκτελείται αμέσως μετά την καταχώρησή της, η αναβεβλημένη εκτέλεση κατά την οποία η εργασία δεν εκτελείται άμεσα αλλά μετά από συγκεκριμένο χρονικό διάστημα και η περιοδική εκτέλεση κατά την οποία η εργασία εκτελείται περιοδικά, δηλαδή ανά τακτά χρονικά διαστήματα. Για καλύτερη διαχείριση των εργασιών, η βιβλιοθήκη YALPS προσφέρει επιπλέον τη δυνατότητα οργάνωσης των εργασιών σε ομάδες (TaskGroups) έτσι ώστε οι εργασίες να μπορούν να τυγχάνουν διαχείρισης σαν ομάδα αντί σαν ατομικές εργασίες. Αυτό είναι σημαντικό διότι με μια απλή διαφοροποίηση σε μια ομάδα εργασιών μπορεί να αλλάξει τελείως η εκτέλεση της εφαρμογής.

Οι εφαρμογές YALPS επικοινωνούν μέσω σειριοποιήσιμων μηνυμάτων, δηλαδή μηνυμάτων σε κατάλληλη μορφή που να επιτρέπει την αποθήκευση ή μετάδοσή τους μέσω μιας σύνδεσης ενός δικτύου. Χρησιμοποιούν ένα στρώμα επικοινωνίας (Communication Layer) το οποίο έχει δύο μορφές, μια μορφή κατάλληλη για προσομοίωση και μια μορφή κατάλληλη για εφαρμογή σε πραγματικό σύστημα. Για προσομοίωση υπάρχει μια ειδική υποδομή επικοινωνίας της βιβλιοθήκης YALPS ενώ για εφαρμογή σε πραγματικό σύστημα τα μηνύματα δρομολογούνται μέσω των γνωστών πρωτοκόλλων UDP/TCP.

Στην παρούσα εργασία χρησιμοποιείται η βιβλιοθήκη YALPS με έτοιμα όλα τα απαραίτητα στοιχεία μιας κατανεμημένης εφαρμογής και με τη βοήθειά της υλοποιούνται δύο ασύγχρονοι αλγόριθμοι gossip οι οποίοι προσομοιώνονται και εφαρμόζονται σε πραγματικό κατανεμημένο σύστημα χωρίς την ανησυχία για τα πρακτικά θέματα δικτύων ή και άλλες τεχνικές δυσκολίες. Περισσότερες πληροφορίες για την YALPS μπορούν να ανακτηθούν από τα [9] και [2].

Κεφάλαιο 3

Περιγραφή και Υλοποίηση Αλγορίθμων

3.1 Μοντέλο Συστήματος Υλοποίησης

Οι δύο αλγόριθμοι με τους οποίους ασχολείται η εργασία αυτή υλοποιούνται με βάση ένα πολύ απλό μοντέλο: θεωρείται ένα κατανεμημένο σύστημα το οποίο αποτελείται από n ασύγχρονους επεξεργαστές – κόμβους οι οποίοι ανταλλάζουν μηνύματα μεταξύ τους. Ο κάθε επεξεργαστής έχει ένα μοναδικό αναγνωριστικό σε ένα σταθερό σύνολο [n] = {1, 2, …, n} και δύναται να παρουσιάσει σφάλμα κατάρρευσης σε οποιαδήποτε χρονική στιγμή διακόπτοντας οριστικά την εκτέλεσή του. Ο μέγιστος αριθμός σφαλμάτων που πιθανό να συμβούν σε μια εκτέλεση του αλγορίθμου είναι μέχρι και f = n-1 επεξεργαστές να καταρρεύσουν. Οι κόμβοι επικοινωνούν άμεσα μεταξύ τους χωρίς την ανάγκη ύπαρξης κάποιου ενδιάμεσου κεντρικού κόμβου και τα μηνύματα που ανταλλάζονται δεν αλλοιώνονται ούτε χάνονται κατά τη μετάδοση.

Για σκοπούς ανάλυσης και υλοποίησης θεωρείται ότι ο κάθε κόμβος εκτελεί τοπικά βήματα, ασυγχρόνιστα από τους υπόλοιπους. Ένα τοπικό βήμα αποτελείται από δύο φάσεις. Στην πρώτη φάση ο κόμβος επιλέγει τυχαία έναν άλλο κόμβο και αποστέλλει σ’ αυτόν όλη την πληροφορία που έχει συλλέξει μέχρι εκείνη τη στιγμή ενώ στη δεύτερη φάση όταν ο κόμβος λάβει ένα μήνυμα από κάποιον άλλο κόμβο, ενημερώνει τη συλλογή του με τις νέες πληροφορίες έτσι ώστε στο επόμενο βήμα να μεταδώσει τη νέα ενημερωμένη συλλογή του. Σε κάθε τοπικό βήμα κάθε κόμβος – διεργασία[4] δύναται να εκτελέσει τρεις ενέργειες:

[4] Οι όροι «διεργασία», «επεξεργαστής» και «κόμβος» εκφράζουν το ίδιο νόημα.

(1) να λάβει ένα σύνολο μηνυμάτων που της έστειλε κάποια άλλη διεργασία.

(2) να διεξάγει κάποιο υπολογισμό.

(3) να αποστείλει ένα ή περισσότερα μηνύματα σε άλλη/ες διεργασία/ες.

Για κάθε εκτέλεση ορίζονται δύο χρονικοί παράμετροι, ο μέγιστος χρόνος καθυστέρησης *d* που χρειάζεται κάθε μήνυμα για να φτάσει στον προορισμό του και ο μέγιστος χρόνος καθυστέρησης *δ* που χρειάζεται κάθε διεργασία για να εκτελέσει τις τρεις ενέργειες. Κατά την εκτέλεση θεωρείται ότι υπάρχει η πιθανότητα εμφάνισης κάποιου αντιπάλου ο οποίος προσπαθεί να παρεμποδίσει τη διάδοση των πληροφοριών – φημών στους κόμβους του δικτύου. Ο αντίπαλος, δηλαδή, αντιπροσωπεύει τα σφάλματα που ενδέχεται να συμβούν στον υπολογισμό (για σκοπούς ανάλυσης). Γίνεται η υπόθεση, λοιπόν, ότι ένας τέτοιος αντίπαλος καθορίζει το σύνολο διεργασιών που είναι προγραμματισμένες να εκτελεστούν σε κάθε βήμα καθώς επίσης και το σύνολο των διεργασιών οι οποίες καταρρέουν στο κάθε βήμα.

Για την επίλυση του προβλήματος της πληροφόρησης κάθε διεργασία p ξεκινά την εκτέλεσή της αρχικά γνωρίζοντας μόνο τη δική της φήμη r_p η οποία είναι άγνωστη στις υπόλοιπες διεργασίες του συστήματος και υποστηρίζει μια συλλογή από τις φήμες τις οποίες έχει λάβει μέχρι στιγμής. Ένα πρωτόκολλο πληροφόρησης πρέπει να ικανοποιεί τρεις σημαντικές απαιτήσεις:

(1) Συλλογή Rumor (Rumor gathering): τελικά κάθε διεργασία να έχει στη συλλογή της rumors, καθένας από τους οποίους είχε χρησιμοποιηθεί ως αρχικός rumor σε κάποια άλλη διεργασία.

(2) Αξιοπιστία (Validity): αν κάποιος rumor προστεθεί στη συλλογή κάποιας διεργασίας τότε αυτός να είναι ο αρχικός rumor κάποιας διεργασίας.

(3) Απραξία (Quiescence): τελικά κάθε διεργασία σταματά για πάντα να στέλλει μηνύματα.

Θωρείται ότι το πρόβλημα της πληροφόρησης έχει «επιλυθεί ορθά» ή διαφορετικά ότι ο αλγόριθμος έχει «ολοκληρωθεί» όταν κάθε διεργασία είτε έχει καταρρεύσει είτε έχει λάβει

τη φήμη κάθε άλλης ορθής[5] διεργασίας του συστήματος και παράλληλα έχει σταματήσει να στέλλει μηνύματα.

3.2 Υλοποίηση Αλγορίθμων με Χρήση YALPS

Οι δύο αλγόριθμοι, ο EARS και ο SEARS, υλοποιούνται με τη βοήθεια της βιβλιοθήκης YALPS της Java και αρκετών έτοιμων υλοποιημένων μεθόδων και διαδικασιών που παρέχει για το στήσιμο μιας κατανεμημένης εφαρμογής. Επειδή οι αλγόριθμοι που υλοποιούνται είναι αλγόριθμοι πληροφόρησης, η λειτουργία τους βασίζεται σε δύο κύριες μεθόδους: την "ενεργή" μέθοδο που επιλέγει τυχαία και περιοδικά ένα κόμβο και στέλλει σ' αυτόν ένα μήνυμα με όλες τις φήμες που γνωρίζει και την "παθητική" μέθοδο που λαμβάνει ένα μήνυμα από ένα κόμβο και με βάση το μήνυμα αυτό ανανεώνει τη συλλογή με τις δικές του φήμες.

Αρχικά δηλώνονται και αρχικοποιούνται οι n κόμβοι του συστήματος. Οι κόμβοι στη YALPS δηλώνονται ως αντικείμενα της διεπαφής NodeID η οποία υλοποιείται μέσω δύο κλάσεων, της S_NodeID και της E_NodeID οι οποίες καθορίζουν αν οι κόμβοι ανήκουν σε περιβάλλον προσομοίωσης ή σε πραγματικό περιβάλλον αντίστοιχα. Σε μια εφαρμογή YALPS ένας κόμβος αρχικοποιείται με την ακόλουθη εντολή:

```
NodeID MyNode=cl.getNodeIdFromString(NodeName);
```

όπου `cl` είναι το αντικείμενο που αντιπροσωπεύει το υπόστρωμα επικοινωνίας που χρησιμοποιείται και `getNodeIdFromString(NodeName)` είναι η συνάρτηση που παίρνει σαν είσοδο το όνομα ενός κόμβου και επιστρέφει στη μεταβλητή `MyNode` το μοναδικό αναγνωριστικό του.

Οι `n` κόμβοι αρχικοποιούνται στη μέθοδο `startNode()` η οποία κληρονομείται από την κλάση `AbstractYalpsNode`. Η κλάση αυτή επιτρέπει σε ένα κόμβο να εκτελέσει όλες τις

[5] Ο όρος «ορθή διεργασία» αναφέρεται στη διεργασία η οποία δεν παρουσιάζει σφάλμα κατά την εκτέλεσή της, δηλαδή εξακολουθεί χωρίς κανένα πρόβλημα να λαμβάνει και να αποστέλλει μηνύματα από και προς άλλες επίσης «ορθές» διεργασίες αντίστοιχα.

απαραίτητες αρχικοποιήσεις που αφορούν τον προγραμματισμό των εργασιών του κόμβου. Μια εργασία (task) αποτελεί το "ενεργό" μέρος της εφαρμογής YALPS. Πρόκειται ουσιαστικά για ένα αντικείμενο που υλοποιεί τη διεπαφή Task. Για την ολοκλήρωση της υλοποίηση της διεπαφής αυτής πρέπει να προστεθεί κώδικας στη "ενεργή" μέθοδο `run()` του αντικειμένου ο οποίος θα περιγράφει τη λειτουργία της εργασίας.

Όπως αναφέρθηκε και προηγουμένως (βλ. Υποκεφάλαιο 2.6) ανάλογα με τον τρόπο εκτέλεσης της εργασίας (άμεση, αναβεβλημένη, περιοδική) η YALPS προσφέρει τρεις διαφορετικές μεθόδους για τον προγραμματισμό της εκτέλεσης μιας εργασίας. Η μέθοδος που είναι υπεύθυνη για την άμεση εκτέλεση εργασιών είναι:

```
void registerTask(Task t, TaskGroup g);
```

όπου `registerTask()` είναι η μέθοδος που καταχωρεί την άμεσης εκτέλεσης εργασία `t` και την συσχετίζει με την ομάδα εργασιών `g`.

Η μέθοδος που είναι υπεύθυνη για την αναβεβλημένη εκτέλεση εργασιών είναι:

```
void registerTask(Task t, TaskGroup g, long millis);
```

όπου η μέθοδος `registerTask` καταχωρεί την εργασία `t` η οποία είναι αναβεβλημένης εκτέλεσης, την συσχετίζει με την ομάδα εργασιών `g` και καθορίζει την εκτέλεσή της να γίνει μετά από χρονικό διάστημα `millis` χιλιοστών δευτερολέπτου.

Τέλος, η μέθοδος που είναι υπεύθυνη για την περιοδική εκτέλεση εργασιών είναι:

```
void registerPeriodicTask(PeriodicTask t, TaskGroup g, long millis);
```

όπου η μέθοδος `registerPeriodicTask`, σε αντίθεση με τις προηγούμενες, καταχωρεί την `t` ως περιοδική εργασία (PeriodicTask) αντί σαν απλή εργασία. Η PeriodicTask είναι μία διεπαφή η οποία έχει κάποιες επιπλέον μεθόδους τις οποίες χρειάζονται αυτού του είδους οι εργασίες όπως είναι η μέθοδος `getPeriod()`. Η μέθοδος αυτή καθορίζει τη χρονική περίοδο

που πρέπει να μεσολαβεί κάθε φορά πριν την επανεκτέλεση της εργασίας, η οποία δίνεται από την παράμετρο `long millis`.

Αφού δηλωθούν και αρχικοποιηθούν οι n κόμβοι του συστήματος, στη συνέχεια καθορίζεται η μορφή των μηνυμάτων που θα διακινούνται καθώς και το υπόστρωμα επικοινωνίας μεταξύ των κόμβων. Η επικοινωνία γίνεται με αποστολή και λήψη μηνυμάτων. Για την αποστολή μηνυμάτων υπάρχουν δύο διαφορετικές μέθοδοι αποστολής οι οποίες ανήκουν στην κλάση `AbstractYalpsNode`. Η μία είναι η `sendTCP()` η οποία χρησιμοποιεί το πρωτόκολλο TCP και η δεύτερη είναι η `sendUDP()` η οποία χρησιμοποιεί το πρωτόκολλο UDP. Και οι δύο μέθοδοι παίρνουν σαν παράμετρο το μήνυμα που πρέπει να αποσταλεί καθώς και τον κόμβο – παραλήπτη στον οποίο πρέπει να παραδοθεί το μήνυμα και το αποστέλλουν σ' αυτόν χρησιμοποιώντας το ανάλογο πρωτόκολλο. Στην υλοποίηση των δύο αλγορίθμων για τη διακίνηση των μηνυμάτων χρησιμοποιείται το πρωτόκολλο TCP ώστε να διασφαλίζεται ότι τα μηνύματα που αποστέλλονται δεν αλλοιώνονται ούτε χάνονται κατά τη μετάδοση κάτι το οποίο προϋποθέτουν οι δύο αλγόριθμοι.

Αφού καθοριστεί και το υπόστρωμα επικοινωνίας απομένει η υλοποίηση των δύο βασικών μεθόδων, της "ενεργής" μεθόδου `run()`που αναφέρθηκε πιο πάνω και της "παθητικής" μεθόδου `receive()`. Η μέθοδος `receive()` ορίζεται στη διεπαφή Receiver και μέσω αυτής οι κόμβοι της εφαρμογής παραλαμβάνουν όλα τα μηνύματα που προέρχονται από άλλους κόμβους και προορίζονται γι' αυτούς. Ενεργοποιείται αμέσως μόλις παραληφθεί ένα μήνυμα με κάποια επικεφαλίδα από τον κόμβο – αποστολέα και επιστρέφει true αν το μήνυμα έχει παραληφθεί με επιτυχία και false στην αντίθετη περίπτωση.

Αφού υλοποιηθούν και οι δύο αυτές μέθοδοι δημιουργείται το αρχείο διαμόρφωσης (configuration file) το οποίο δίνεται σαν είσοδος στον προσομοιωτή SimulatedNodeStarter ή στον εκτελεστή ExecutorNodeStarter (ανάλογα με το πού θα τρέξει η εφαρμογή). Το αρχείο διαμόρφωσης καθορίζει ποιοι κόμβοι της εφαρμογής θα αρχικοποιηθούν και ποιοι θα εκτελεστούν. Παράδειγμα αρχείου διαμόρφωσης που χρησιμοποιείται στη συγκεκριμένη υλοποίηση για εφαρμογή σε περιβάλλον προσομοίωσης είναι το εξής (για τον αλγόριθμο EARS και n=10):

```
sim.numNodes=10
node.CLASS=EarsNode
#parameters defined in AbstractYalpsNode
node.outFileName=outfile.log
node.rnd='node'
#parameters defined in Ears
node.nodeList=1;2;3;4;5;6;7;8;9;10;
```

Το αρχείο αποτελείται από δηλώσεις αντικειμένων με τις ιδιότητές τους. Πρώτα καθορίζεται ο συνολικός αριθμός των κόμβων του συστήματος (`sim.numNodes`). Στο πιο πάνω παράδειγμα καθορίζονται 10 κόμβοι. Έτσι χρειάζεται μόνο ένα αρχείο διαμόρφωσης το οποίο είναι κοινό για όλους τους κόμβους του συστήματος. Ακολουθεί η αρχικοποίηση του αντικειμένου node της εφαρμογής με τη δήλωση της κλάσης στην οποία ανήκει (`node.CLASS=EarsNode`) ενώ τελικά δηλώνεται η ιδιότητα nodeList του αντικειμένου node η οποία στο παράδειγμα παίρνει τις τιμές 1 μέχρι 10 ορίζοντας τους 10 κόμβους του συστήματος. Αξίζει να σημειωθεί ότι για να μπορεί να υπάρξει μια τέτοια δήλωση στο αρχείο διαμόρφωσης, δηλαδή δήλωση της μορφής <object name>.<property name> = <value> πρέπει η κλάση του αντικειμένου που δηλώνεται να περιέχει μία μέθοδο τύπου void, το όνομα της οποίας να αποτελεί τη συνένωση της συμβολοσειράς “set” με το όνομα της ιδιότητας. Συνεπώς στο παράδειγμα θα πρέπει στην κλάση EarsNode να υπάρχει μία μέθοδος τύπου void με την ονομασία setNodeList() η οποία καθορίζει το σύνολο των κόμβων του συστήματος.

Σε περίπτωση εφαρμογής σε πραγματικό σύστημα το αρχείο διαμόρφωσης έχει παρόμοια μορφή όπως πιο πάνω αλλά με κάποιες διαφορές κυρίως στο σημείο όπου δηλώνονται οι κόμβοι της εφαρμογής. Οι κόμβοι στην προσομοίωση δεν είναι πραγματικοί οπότε δε χρειάζεται να δηλωθεί η πραγματική διεύθυνση δικτύου ούτε ο αριθμός θύρας για τον κάθε κόμβο (port number). Αντιθέτως σε πραγματικό σύστημα τα στοιχεία αυτά θα πρέπει να δηλωθούν. Έτσι ουσιαστικά η κύρια διαφοροποίηση που παρατηρείται στο αρχείο

διαμόρφωσης είναι στη δήλωση της ιδιότητας `nodeList`. Σε πραγματικό σύστημα η αντίστοιχη δήλωση είναι:

```
node.nodeList=hostname:port;hostname2:port2;……;……
```

Επιπλέον, επειδή στο πραγματικό σύστημα ο κάθε κόμβος έχει τη δική του διεύθυνση και αριθμό θύρας, χρειάζεται ο κάθε κόμβος να διαθέτει το δικό του ξεχωριστό αρχείο διαμόρφωσης με τα στοιχεία διεύθυνσης και θύρας και όχι ένα κοινό αρχείο όπως συμβαίνει στον προσομοιωτή. Έτσι στο αρχείο υπάρχουν οι ακόλουθες επιπλέον πληροφορίες:

```
executor.virtualAddress= . . . .
executor.port= . . . .
```

όπου `virtualAddress` είναι η διεύθυνση του κόμβου και `port` είναι ο αριθμός θύρας.

Παράδειγμα αρχείου διαμόρφωσης για πραγματικό σύστημα βρίσκεται στο Παράρτημα Γ.

Αφού δημιουργηθεί και το αρχείο διαμόρφωσης η εφαρμογή είναι πλέον έτοιμη είτε να τρέξει σε περιβάλλον προσομοίωσης είτε σε πραγματικό περιβάλλον. Το μεγάλο πλεονέκτημα των εφαρμογών YALPS είναι ότι ο ίδιος κώδικας, σχεδόν χωρίς καμιά αλλαγή, μπορεί να τρέξει απευθείας από περιβάλλον προσομοίωσης σε πραγματικό περιβάλλον αλλάζοντας μόνο την κλήση μιας συνάρτησης. Συγκεκριμένα, καλώντας τη συνάρτηση `SimulatedNodeStarter()` η εφαρμογή τρέχει σε περιβάλλον προσομοίωσης ενώ καλώντας τη συνάρτηση `ExecutorNodeStarter()` η εφαρμογή τρέχει σε πραγματικό περιβάλλον.

Παράδειγμα εντολής για προσομοίωση της εφαρμογής:

```
java –cp /yalps.jar launchers.SimulatedNodeStarter myExample-sim.conf
```

Παράδειγμα εντολής για τρέξιμο της εφαρμογής σε πραγματικό σύστημα:

```
java –cp /yalps.jar launchers.ExecutorNodeStarter myExample-exec.conf
```

Όπου `myExample-sim.conf` και `myExample-exec.conf` είναι τα αντίστοιχα αρχεία διαμόρφωσης.

3.3 Αλγόριθμος EARS

Στο υποκεφάλαιο αυτό θα γίνει αρχικά μια σύντομη αλλά περιεκτική περιγραφή του αλγορίθμου EARS [3], του πρώτου αλγορίθμου με τον οποίο ασχολείται η παρούσα εργασία και στη συνέχεια θα δοθούν σημαντικές λεπτομέρειες σε θέματα υλοποίησης του αλγορίθμου αυτού.

3.3.1 Περιγραφή Αλγορίθμου EARS

Ο αλγόριθμος EARS (Epidemic Asynchronous Rumor Spreading) βασίζεται στο κλασικό επιδημικό μοντέλο και προσπαθεί να επιλύσει πλήρως το πρόβλημα της πληροφόρησης. Δηλαδή ενώ αρχικά κάθε κόμβος γνωρίζει μόνο μία φήμη, τη δική του, τελικός του στόχος είναι να μάθει όλες τις υπόλοιπες φήμες όλων των υπόλοιπων κόμβων του δικτύου. Έχει αποδειχθεί [3] ότι ο EARS μπορεί να ανεχτεί μέχρι και *n-1* αποτυχίες (failures), υπό τον αφελή αντίπαλο, ενώ η αποδοτικότητά του μπορεί να φτάσει σε επίπεδα συγκρίσιμα με αυτά των καλύτερων σύγχρονων αλγορίθμων πληροφόρησης. Συγκεκριμένα, αποδείχθηκε ότι ο αλγόριθμος αυτός με μεγάλη πιθανότητα μπορεί να ολοκληρώνει τη διάδοση σε χρόνο $O\left(\frac{n}{n-f}\log^2 n(d+\delta)\right)$ αποστέλλοντας $O(n \log^3 n (d+\delta))$ μηνύματα, όπου *n* ο συνολικός αριθμός των κόμβων του δικτύου, *f* ο συνολικός αριθμός των αποτυχιών / σφαλμάτων που μπορεί να συμβούν κατά την εκτέλεση και *d, δ* οι μέγιστες χρονικές καθυστερήσεις στην αποστολή ενός μηνύματος και στην ταχύτητα των υπολογισμών αντίστοιχα. Υπενθυμίζεται ότι τα d και δ δεν τα γνωρίζει ο αλγόριθμος. Την τιμή του f όμως τη γνωρίζει γι' αυτό στην υλοποίηση καθορίζεται ένα άνω φράγμα για το πλήθος των σφαλμάτων που μπορεί να

ανεχτεί ο αλγόριθμος. Επειδή στη χείριστη περίπτωση ο αλγόριθμος μπορεί να ανεχτεί μέχρι και *n-1* σφάλματα, συνεπάγεται ότι η μέγιστη τιμή που μπορεί να πάρει το f είναι *n-1*.

Εκείνο που διαφοροποιεί τον αλγόριθμο EARS από τους άλλους αλγόριθμους πληροφόρησης είναι ότι σε αντίθεση με το επιδημικό μοντέλο, κατά τη μετάδοση των φημών μεταδίδεται και κάποια επιπλέον πληροφορία για το βαθμό διάδοσης των υπόλοιπων φημών στο δίκτυο μέχρι εκείνη τη στιγμή. Κεντρική ιδέα του αλγορίθμου είναι: σε κάθε βήμα μία διεργασία – κόμβος επιλέγει τυχαία ένα στόχο (ένα άλλο κόμβο – διεργασία) και στέλλει σ' αυτόν όλη την πληροφορία που έχει συγκεντρώσει μέχρι εκείνη τη στιγμή. Αυτή η διαδικασία είναι σχεδιασμένη να ικανοποιεί τρεις ιδιότητες:

(1) Συλλογή (gathering), δηλαδή μετά από κάποια χρονική περίοδο κάθε φήμη, που αρχικά ήταν γνωστή σε μια ορθή διεργασία, είναι γνωστή σε κάθε άλλη διεργασία που δεν έχει καταρρεύσει.

(2) Κυνήγι (shooting), δηλαδή σε κάθε ένα μη σταθερό χρονικό διάστημα κάθε φήμη που είναι γνωστή σε ένα μεγάλο αριθμό ορθών διεργασιών αποστέλλεται σε κάθε άλλη ορθή διεργασία.

(3) Ανταλλαγή (exchange), δηλαδή σε κάθε ένα μη σταθερό χρονικό διάστημα κάθε ζεύγος διεργασιών που δεν έχουν καταρρεύσει ανταλλάζουν πληροφορίες για το ποιος είναι γνωστός σε μεγάλο αριθμό διεργασιών.

Η διαδικασία του αλγορίθμου σε περισσότερη λεπτομέρεια είναι η εξής: Κάθε διεργασία p, καθ' όλη τη διάρκεια της εκτέλεσης διατηρεί τρεις λίστες :

(1) Τη λίστα V(p) στην οποία περιλαμβάνονται όλες οι φήμες που γνωρίζει η διεργασία p από το σύνολο των διεργασιών [n] και η οποία αρχικά περιέχει μόνο τη δική της τοπική φήμη.

(2) Τη λίστα I(p) – Informed List η οποία περιλαμβάνει όλα τα ζεύγη (r,q) όπου r είναι η φήμη που στάλθηκε στη διεργασία q από κάποια διεργασία και η οποία αρχικά είναι άδεια.

(3) Τη λίστα L(p) η οποία περιλαμβάνει όλες τις διεργασίες για τις οποίες η p δεν μπορεί να εξακριβώσει μέσω της λίστας I(p) αν τους έχουν σταλεί όλες οι φήμες που περιέχονται στη λίστα V(p), δηλαδή $L(p)=\{q: \exists\, r \in V(p),\ (r,q) \notin I(p)\}$. Αρχικά L(p)=[n].

Σε κάθε βήμα μια διεργασία p αποστέλλει ένα μήνυμα το οποίο αποτελείται από τις τοπικές της λίστες V(*p*) και I(*p*), σε μια διεργασία q η οποία επιλέγεται τυχαία από το [*n*]. Τότε η p προσθέτει όλα τα ζεύγη (*r*,*q*) όπου $r \in V(p)$, στην τοπική της λίστα I(*p*). Από την άλλη όταν η q λαμβάνει ένα νέο μήνυμα από την *p*, ανανεώνει ομοίως τα δικά της V(*q*) και I(*q*). Και στις δύο περιπτώσεις (αποστολή και λήψη μηνύματος) ακολουθεί η ενημέρωση της λίστας L με βάση τις λίστες V και I. Για την ενημέρωση της λίστας L(*p*) η διαδικασία είναι η εξής (ομοίως για L(*q*) και για οποιαδήποτε διεργασία q): γίνεται σύγκριση μεταξύ του V(*p*) και του I(*p*) και ελέγχεται αν για κάθε φήμη $r \in V(p)$ υπάρχει ζεύγος (*r*,*q*) όπου $(r,q) \in I(p)$. Αν δεν υπάρχει τότε σημαίνει ότι ακόμη δεν υπάρχει επαρκής απόδειξη ότι η φήμη *r* έχει διαδοθεί. Αν υπάρχει τότε σημαίνει ότι η φήμη *r* έχει διαδοθεί σε τουλάχιστον μία διεργασία και συνεπώς αφαιρείται από τη λίστα L(*p*).

Αν η διαδικασία επαναλαμβάνεται συνεχώς, σε κάποια φάση η L(*p*) θα μείνει κενή οπότε θα σημαίνει ότι όλες οι φήμες της *p* έχουν σταλεί σε όλες τις άλλες διεργασίες. Τότε η *p* είναι έτοιμη να περάσει στη φάση του "shut-down" η οποία διαρκεί το πολύ $\Theta\left(\frac{n}{n-f}\log n\right)$ βήματα. Για το σκοπό αυτό η κάθε διεργασία διατηρεί έναν τοπικό μετρητή που είναι υπεύθυνος για τη ρύθμιση της διάρκειας της "shut-down" φάσης της διεργασίας. Σύμφωνα με τη φάση αυτή, η *p* συνεχίζει όπως πριν να λαμβάνει μηνύματα από άλλες διεργασίες και να στέλλει "shut-down" μήνυμα σε μια διεργασία (έστω *q*) η οποία επιλέγεται τυχαία από το [*n*]. Μετά τη φάση του "shut-down" η *p* σταματά να αποστέλλει μηνύματα και "κοιμάται". Το χαρακτηριστικό του αλγορίθμου σ' αυτό το σημείο είναι ότι κατά τη φάση του "shut-down" και ενόσω η *p* βρίσκεται στο "sleep mode", μπορεί να συνεχίσει να λαμβάνει μηνύματα από άλλες διεργασίες οι οποίες ακόμη δεν έχουν "κοιμηθεί". Εάν η *p* λάβει μία νέα φήμη η οποία ακόμη δεν έχει σταλεί σε κάποια διεργασία, δηλαδή εάν η λίστα L(*p*) γίνει μη κενή τότε

διακόπτει τη φάση του “shut-down”, “ξυπνάει” και ξαναρχίζει τη διαδικασία μέχρις ότου η L(*p*) γίνει και πάλι κενή.

Ο αλγόριθμος ολοκληρώνει τη λειτουργία του όταν κάθε διεργασία είτε έχει καταρρεύσει είτε έχει μάθει όλες τις φήμες του συστήματος και παράλληλα έχει σταματήσει να αποστέλλει μηνύματα σε άλλες διεργασίες, χωρίς να έχει καταρρεύσει.

3.3.2 Λεπτομέρειες Υλοποίησης EARS

Παρακάτω φαίνεται ο ψευδοκώδικας στον οποίο βασίζεται η υλοποίηση του αλγορίθμου (Εικόνα 3.1). Ο ψευδοκώδικας αφορά μια διεργασία *p* η οποία γνωρίζει αρχικά μόνο τη φήμη r_p. Κάθε φορά η *p* προγραμματίζεται να εκτελεί ένα τοπικό βήμα που αποτελείται από τρεις ενέργειες:

(1) Παραλαμβάνει ένα σύνολο μηνυμάτων τα οποία στάλθηκαν σ’ αυτήν.

(2) Ανανεώνει τις τοπικές λίστες που διατηρεί από την έναρξη της διαδικασίας.

(3) Αποστέλλει ένα μήνυμα σε κάποια άλλη διεργασία η οποία επιλέγεται τυχαία.

Το βήμα αυτό στον ψευδοκώδικα δηλώνεται με την εκτέλεση μιας επανάληψης του κυρίως βρόγχου (main loop).

```
EARS(r_p)
 1  ▷ Initialization:
 2  V(p) ← {r_p}            ▷ a set of processes
 3  I(p) ← ∅                ▷ a set of pairs ⟨r, p⟩
 4  L(p) ← [n]              ▷ a set of processes
 5  sleep_cnt ← 0           ▷ an integer
 6
 7  repeat
 8     for every message m = ⟨V, I⟩ received do
 9           V(p) ← V(p) ∪ m.V
10           I(p) ← I(p) ∪ m.I
11     Update L(p) based on V(p) and I(p).
12     if L(p) = ∅
13           then    sleep_cnt ← sleep_cnt + 1
14           else    sleep_cnt ← 0
15     if sleep_cnt < Θ((n/(n−f)) log n) then
16           ▷ Epidemic Transmission Mode:
17           Choose q uniformly at random from [n].
18           Send m = ⟨V(p), I(p)⟩ to process q.
19           for every r ∈ V(p) do
20              I(p) ← I(p) ∪ (r, q)
21           Update L(p) based on V(p) and I(p).
```

Εικόνα 3.1: Ψευδοκώδικας Αλγορίθμου EARS για μια διεργασία p με αρχική φήμη r_p [3]

Αρχικά δηλώνονται οι διεργασίες του συστήματος με τις τρεις τοπικές τους λίστες, την Vp, την Ip και την Lp και στην κάθε διεργασία *p* ανατίθεται μια διαφορετική πληροφορία (φήμη) η οποία θα πρέπει με το τέλος του αλγορίθμου να έχει μεταδοθεί σε όλες τις υπόλοιπες διεργασίες. Αρχικά η Vp περιέχει την τοπική φήμη της κάθε διεργασίας, η Ip είναι άδεια, ενώ η Lp περιλαμβάνει το σύνολο όλων των διεργασιών του συστήματος. Επιπλέον δηλώνεται και μία ακέραια μεταβλητή sleep_cnt η οποία αρχικά παίρνει την τιμή μηδέν και μετρά τη χρονική διάρκεια της "shut-down" φάσης της διεργασίας καθορίζοντας έμμεσα και τον "τερματισμό" του αλγορίθμου. Στο πιο κάτω κομμάτι κώδικα φαίνεται ο τρόπος που οργανώνει τα περιεχόμενά της η κάθε διεργασία – κόμβος:

```
public class MyEarsNode extends AbstractYalpsNode{
    ArrayList<NodeID> DestinatioList = new ArrayList<NodeID>();
    ArrayList<String> Vp;
    ArrayList<RumorProcess> Ip;
    ArrayList<String> Lp;
    int sleep_cnt;
. . .
}
```

Όπου "`DestinationList`" είναι η λίστα με όλους τους κόμβους και "`RumorProcess`" είναι μια κλάση αποτελούμενη από τα ζεύγη <`φήμη, διεργασία`> που υποδηλώνουν ότι η "`φήμη`" έχει σταλεί στη "`διεργασία`".

Ο αλγόριθμος εκτελείται σε τοπικά βήματα και κάθε βήμα αποτελείται από την ενεργή μέθοδο MyMsgSenderTask() η οποία διαχειρίζεται τη φάση αποστολής ενός μηνύματος και την παθητική μέθοδο receive() η οποία διαχειρίζεται τη φάση της λήψης ενός μηνύματος. Η ενεργή μέθοδος η οποία είναι υπεύθυνη για τη φάση της αποστολής μηνυμάτων ορίζεται ως εξής:

```
protected void MyMsgSenderTask(){
    .
    .
}
```

Στην αρχή της φάσης αυτής ελέγχεται κατά πόσο η λίστα L(p) είναι άδεια. Εάν δεν είναι άδεια τότε η τιμή της μεταβλητής “sleep_cnt” παραμένει μηδενική υποδηλώνοντας πως ακόμη να διαδοθούν όλες οι φήμες του συστήματος σε όλες τις διεργασίες που αποτελούν την L(p) λίστα και συνεπώς δεν υπάρχει λόγος να ξεκινήσει η φάση του “shut-down”. Μόλις η λίστα μείνει κενή όμως, η τιμή του sleep_cnt αυξάνεται κάθε φορά κατά μία μονάδα, σηματοδοτώντας την έναρξη της “shut-down” φάσης της διεργασίας και ταυτόχρονα μετρώντας το χρόνο διάρκειας αυτής της φάσης. Πιο κάτω φαίνεται ο τρόπος με τον οποίο γίνεται ο έλεγχος αυτός:

```
if (Lp.size() == 0)
    sleep_cnt = sleep_cnt + 1;
else
    sleep_cnt = 0;
```

Αφού γίνει ο έλεγχος για τα περιεχόμενα της λίστας L(p) στη συνέχεια ελέγχεται η τιμή της μεταβλητής sleep_cnt. Εάν αυτή ξεπεράσει ένα συγκεκριμένο όριο τότε η διεργασία σταματά οριστικά να αποστέλλει μηνύματα σε άλλες διεργασίες και ο αλγόριθμος διακόπτει τη διάδοση των φημών. Εάν αυτή βρίσκεται εντός των προκαθορισμένων ορίων και η sleep_cnt είναι μηδέν, τότε η διεργασία αποστέλλει κανονικά μηνύματα επιλέγοντας κάθε φορά μία διεργασία – παραλήπτη με τυχαίο τρόπο. Εάν η L(p) μείνει κενή τότε η μεταβλητή sleep_cnt αυξάνει την τιμή της κατά μία μονάδα εκκινώντας παράλληλα τη “shut-down” φάση της διεργασίας. Με την έναρξη της φάσης αυτής η μεταβλητή “sleep_cnt” αυξάνεται συνεχώς μέχρις ότου η τιμή της ξεπεράσει το προκαθορισμένο όριο. Η περίοδος αυτή που μεσολαβεί είναι ακριβώς η “shut-down” φάση της διεργασίας και διαρκεί $\Theta(\frac{n}{n-f}\log n)$ βήματα, όπου *n* ο αριθμός των διεργασιών του συστήματος και *f* ο αριθμός των πιθανών σφαλμάτων. Αν κατά τη διάρκεια της “shut-down” φάσης διαπιστωθεί ότι σε κάποια διεργασία κάποια φήμη δεν έχει διαδοθεί ακόμη, τότε αυτόματα η λίστα L(p) από κενή γίνεται μη κενή και συνεπώς η μεταβλητή sleep_cnt μηδενίζεται ενεργοποιώντας ξανά την πιο πάνω διαδικασία. Πιο κάτω φαίνεται ο τρόπος με τον οποίο γίνεται ο έλεγχος της μεταβλητής sleep_cnt:

```
if (sleep_cnt < 2*(n/(n-f)*(StrictMath.log(n)/StrictMath.log(2)))){
. . .
}
```

Αξίζει να σημειωθεί ότι για σκοπούς υλοποίησης για να ικανοποιείται το θεωρητικό όριο $\Theta(\frac{n}{n-f}\log n)$ έπρεπε να καθοριστεί κάποιο άνω φράγμα αυτής της ποσότητας ως η ακριβής διάρκεια της "shut-down" φάσης. Αφού δοκιμάστηκαν διάφορες τιμές, ανάμεσά τους, οι ποσότητες $2\frac{n}{n-f}\log n$ και $4\frac{n}{n-f}\log n$, και διαπιστώθηκε ότι δεν επηρέαζαν σημαντικά τα γενικότερα συμπεράσματα, αποφασίστηκε τελικά στα πειράματα να χρησιμοποιηθεί η ποσότητα $2\frac{n}{n-f}\log n$.

Επιστρέφοντας τώρα στον έλεγχο της μεταβλητής sleep_cnt, στην περίπτωση που αυτή βρίσκεται εντός των προκαθορισμένων ορίων τότε η διεργασία εκτελεί τρεις βασικές ενέργειες:

(1) Επιλέγει τυχαία και ομοιόμορφα μία διεργασία από το σύνολο διεργασιών.

(2) Δημιουργεί και αποστέλλει ένα μήνυμα στην επιλεγείσα διεργασία.

(3) Ανανεώνει τις λίστες I(p) και L(p).

Για την τυχαία επιλογή διεργασίας χρησιμοποιείται η κλάση Random της βιβλιοθήκης Java και το κομμάτι κώδικα που αφορά το σημείο αυτό είναι:

```
int rand;
int size = DestinationList.size();
NodeID neighbor;
rand = new Random().nextInt(size);
neighbor = DestinationList.get(rand);
```

όπου `DestinationList` είναι η λίστα με τους κόμβους – διεργασίες του συστήματος, `DestinationList.size()` η συνάρτηση που επιστρέφει τον αριθμό των κόμβων του συστήματος, `DestinationList.get(rand)` η συνάρτηση που επιστρέφει τον κόμβο που βρίσκεται στη θέση της λίστας με τους κόμβους και `Random().nextInt(size)` η συνάρτηση – γεννήτρια ψευδοτυχαίων ακέραιων αριθμών μεταξύ του 0 και του `size-1`. Για

τη δημιουργία ενός μηνύματος ορίζεται η κλάση MySerializableMsg και η δομή του κάθε μηνύματος φαίνεται στο παρακάτω τμήμα κώδικα:

```
class MySerializableMsg implements Serializable{
    private ArrayList<String> V = null;
    private ArrayList<RumorProcess> I = null;
    Message (ArrayList<String> Vp , ArrayList<RumorProcess> Ip){
       this.V = Vp;
       this.I = Ip;
    }
}
```

Δηλαδή το κάθε μήνυμα που διακινείται στο σύστημα έχει τη μορφή <V, I> όπου V η λίστα με όλες τις προς αποστολή φήμες και I η λίστα με όλα τα ζεύγη (φήμη, διεργασία). Όπως αναφέρθηκε και προηγουμένως για τα ζεύγη αυτά στην υλοποίηση υπάρχει μία κλάση ειδικά ορισμένη για τη λίστα I η οποία καθορίζει τον τύπο των περιεχομένων της λίστας. Η δήλωση της εν λόγω κλάσης γίνεται ως εξής:

```
class RumorProcess implements Serializable{
    private static final long serialVersionUID = 1L;
    String r;
    String q;
}
```

Όπου r και q αποτελούν το ζεύγος (r,q) που υποδεικνύει ότι η φήμη r έχει σταλεί στη διεργασία q.

Για την αποστολή ενός μηνύματος στην διεργασία που επιλέγεται τυχαία χρησιμοποιείται το πρωτόκολλο TCP και η εντολή που εκτελείται είναι:

```
getCommLayer().sendTCP(m, neighbor);
```

Όπου `m` είναι το μήνυμα που αποστέλλεται και `neighbor` είναι ο γείτονας – κόμβος στον οποίο αποστέλλεται το μήνυμα m.

Τέλος, για να ολοκληρωθεί η ***ενεργή μέθοδος*** (βλ. επεξήγηση στο Υποκεφάλαιο 3.2) του αλγορίθμου απομένει η διεργασία να ενημερώσει – ανανεώσει τα περιεχόμενα των δύο

λιστών της I(p) και L(p). Για την ενημέρωση της λίστας I(p) ακολουθείται η εξής διαδικασία: για την κάθε φήμη r που αποστέλλει η διεργασία p στην επιλεγείσα διεργασία q, και κατ' επέκταση για την κάθε φήμη r που περιέχεται στη V(p) λίστα, προστίθενται στη λίστα I(p) όλα τα ζεύγη (r,q) σηματοδοτώντας ότι όλες οι φήμες r της διεργασίας p έχουν σταλεί στη διεργασία q. Το κομμάτι κώδικα που αφορά το κομμάτι αυτό είναι:

```
for (int r=0; r<m.getRumorVlist().size(); r++){
    RumorProcess rp = new RumorProcess();
    rp.setRumor(m.getRumorVlist().get(r));
    rp.setProcess("Node"+neighbor.getName());
    if (!findPairIp(rp) && !(rp.q.equals("Node"+nodeId.getName()))){
       Ip.add(rp);
    }
}
```

Όπου `findPairIp(rp)` είναι η συνάρτηση που παίρνει ως παράμετρο ένα ζεύγος (r,q) και ελέγχει αν αυτό υπάρχει ήδη στη λίστα I(p). Αν υπάρχει ήδη τότε επιστρέφει την τιμή True διαφορετικά την τιμή False. Παρακάτω φαίνεται ο κώδικας της συνάρτησης αυτής:

```
public boolean findPairIp(RumorProcess rp){
    for (int i=0; i<Ip.size(); i++){
       if ((Ip.get(i).r).equals(rp.r) && (Ip.get(i).q).equals(rp.q)){
             return true;
       }
    }
    return false;
}
```

Για την ενημέρωση της λίστας L(p) η διαδικασία που ακολουθείται βασίζεται στις δύο λίστες V(p) και I(p) και είναι η εξής: για κάθε διεργασία q που περιέχεται στην L(p) γίνεται έλεγχος αν υπάρχουν στην I(p) όλα τα ζεύγη (r,q) όπου r οι φήμες της p (δηλαδή αυτοί που περιέχονται στη V(p) λίστα). Η συνάρτηση που υλοποιεί τη διαδικασία αυτή είναι:

```
public void updateLpList(){
    for (int i=0; i<this.Lp.size(); i++){
       if (hasAllRumors(this.Lp.get(i))){
             this.Lp.remove(i);
       }
    }
}
```

Όπου hasAllRumors(this.Lp.get(i)) είναι η συνάρτηση που κάνει τον έλεγχο βάσει των I(p) και V(p) λιστών και ο κώδικάς της φαίνεται πιο κάτω:

```
public boolean hasAllRumors(String q){
  int totalRumors = Vp.size();
  for (int i=0; i<Ip.size(); i++){
     for (int rumor=0; rumor<Vp.size(); rumor++){
        if(Ip.get(i).r.equals(Vp.get(rumor))&&Ip.get(i).q.equals(q)){
            totalRumors--;
        }
      }
      if(totalRumors==0)
            break;
    }
    if(totalRumors==0)
       return true;
    else
       return false;
}
```

Η ***παθητική μέθοδος*** η οποία είναι υπεύθυνη για τη φάση της λήψης μηνυμάτων ορίζεται ως εξής:

```
public boolean receive(short header, Object msg, NodeID sender) {
}
```

Ένας κόμβος p εισέρχεται αυτόματα στην παθητική μέθοδο όταν παραλάβει κάποιο μήνυμα. Τότε βάσει του νέου μηνύματος που έχει λάβει ενημερώνει τις λίστες V(p), I(p) και L(p). Οι δύο τελευταίες λίστες ανανεώνονται με τον ίδιο τρόπο όπως αναφέρθηκε πιο πάνω. Η ενημέρωση της V(p) λίστας είναι μια απλή διαδικασία σύγκρισης της λίστας V που περιέχεται στο μήνυμα με τη λίστα V(p) που διαθέτει ο παραλήπτης του μηνύματος. Αν στο μήνυμα που παραλήφθηκε διαπιστωθεί κάποια φήμη η οποία δεν περιέχεται στην V(p) τότε η V(p) ενημερώνεται με τη νέα αυτή φήμη. Παρακάτω φαίνεται η υλοποίηση της μεθόδου receive():

```
public boolean receive(short header, Object msg, NodeID sender) {
      MySerializableMsg m=(MySerializableMsg)msg;
      updateAllLists(m);
      return true;
}
```

Όπου `updateAllLists(m)` η συνάρτηση που ενημερώνει και τις τρεις λίστες V(p), I(p) και L(p) με βάση τις διαδικασίες που περιγράφτηκαν πιο πάνω.

Αξίζει να σημειωθεί ότι τα όσα περιγράφηκαν πιο πάνω δεν λαμβάνουν υπόψη την πιθανότητα να συμβούν σφάλματα κατάρρευσης κατά την εκτέλεση. Επειδή όμως σε ένα πραγματικό ασύγχρονο περιβάλλον υπάρχει πάντα η πιθανότητα να συμβούν σφάλματα, προστέθηκε στο περιβάλλον προσομοίωσης και υλοποίηση για σφάλματα η οποία περιγράφεται αναλυτικά στο Κεφάλαιο 4. Ο κώδικας για τον αλγόριθμο EARS επισυνάπτεται στο τέλος της εργασίας στο Παράρτημα Α.

3.4 Αλγόριθμος SEARS

Στο υποκεφάλαιο αυτό θα γίνει αρχικά μια σύντομη αλλά περιεκτική περιγραφή του αλγορίθμου SEARS [3], του δεύτερου αλγορίθμου με τον οποίο ασχολείται η παρούσα

εργασία και στη συνέχεια θα δοθούν σημαντικές λεπτομέρειες σε θέματα υλοποίησης του αλγορίθμου αυτού.

3.4.1 Περιγραφή Αλγορίθμου SEARS

Ο αλγόριθμος SEARS (Spamming Epidemic Asynchronous Rumor Spreading) επιλύει ορθά το πρόβλημα gossip με μεγάλη πιθανότητα βασιζόμενος σε μια τεχνική γοργής διάδοσης των φημών. Αποκλίνει από το επιδημικό μοντέλο στέλλοντας περισσότερα μηνύματα σε κάθε βήμα και προσπαθώντας να διαδοθούν οι φήμες σε όσο το δυνατό πιο σύντομο χρονικό διάστημα. Έχει αποδειχθεί [3] ότι ο SEARS είναι ένας σταθερού – χρόνου αλγόριθμος πληροφόρησης ο οποίος μπορεί να ανεχτεί μέχρι και *n-1* αποτυχίες (failures), υπό τον αφελή αντίπαλο. Συγκεκριμένα, αποδείχθηκε ότι ο αλγόριθμος αυτός με μεγάλη πιθανότητα ολοκληρώνει τη διάδοση των φημών σε σταθερό χρόνο της τάξεως $O\left(\frac{n}{\varepsilon(n-f)}(d+\delta)\right)$ αποστέλλοντας $O\left(\frac{n^{2+\varepsilon}}{\varepsilon(n-f)}\log n\,(d+\delta)\right)$ μηνύματα, όπου $\varepsilon < 1$ όπου *n* ο συνολικός αριθμός των κόμβων του δικτύου, *f* ο συνολικός αριθμός των σφαλμάτων κατάρρευσης κόμβων που συμβαίνουν κατά την εκτέλεση και *d, δ* οι μέγιστες χρονικές καθυστερήσεις στην αποστολή ενός μηνύματος και στην ταχύτητα των υπολογισμών αντίστοιχα.

Η δομή του αλγορίθμου σε γενικές γραμμές είναι όμοια με τον EARS με τη διαφορά ότι σε κάθε βήμα, η διεργασία αποστέλλει μηνύματα σε $\Theta(n^{\varepsilon}\log n)$ διεργασίες που επιλέγονται τυχαία, και όχι σε μόνο μία όπως γίνεται στον EARS. Επιπλέον, σε αντίθεση με τον EARS, κάθε διεργασία μπορεί να περάσει στη φάση του "shut-down" μόνο μία φορά και συνεπώς δεν ελέγχεται αν κάποια φήμη δεν έχει διαδοθεί ακόμη. Όμως το πρόβλημα πάλι λύνεται τελικά διότι η κάθε διεργασία προνοεί σε κάθε βήμα να στέλλει μηνύματα σε περισσότερες από μία διεργασίες οπότε σε κάθε βήμα αυξάνονται οι διεργασίες που μαθαίνουν μία νέα φήμη.

Η ορθότητα του αλγορίθμου φαίνεται και από την παρακάτω αλληλουχία γεγονότων: έστω *r* κάποια συγκεκριμένη φήμη η οποία αποτελεί την αρχική φήμη μιας διεργασίας *p*.

Μέσα σε χρόνο $O\left(\frac{n}{n-f}(d+\delta)\right)$ η διεργασία *p* έχει στείλει τη φήμη της σε τουλάχιστον $\Theta(n^{\varepsilon})$ άλλες ορθές διεργασίες (με μεγάλη πιθανότητα). Σε κάθε $O\left(\frac{n}{n-f}(d+\delta)\right)$ χρόνο, ο αριθμός των ορθών διεργασιών που μαθαίνουν τον r αυξάνονται με συντελεστή αύξησης n^{ε}. Μετά από $\frac{1}{\varepsilon}$ βήματα, πλέον μια σταθερή ομάδα από διεργασίες γνωρίζουν την *r* (με μεγάλη πιθανότητα) οπότε σε επόμενο $O\left(\frac{n}{n-f}(d+\delta)\right)$ χρόνο, αυτές οι ορθές διεργασίες έχουν στείλει συλλογικά την *r* σε κάθε άλλη διεργασία στο σύστημα (πάλι με μεγάλη πιθανότητα). Τότε, με τον ίδιο τρόπο που διαδίδεται η φήμη *r*, διαδίδεται και το γεγονός ότι "η *r* έχει σταλεί σε κάθε ορθή διεργασία" και όλα αυτά μέσα σε χρόνο $O\left(\frac{n}{\varepsilon(n-f)}(d+\delta)\right)$ μετά από τον οποίο ακολουθεί μία φάση αποστολής "shut-down" μηνυμάτων που οδηγεί τελικά στον τερματισμό της διαδικασίας (με μεγάλη πιθανότητα).

3.4.2 Λεπτομέρειες Υλοποίησης SEARS

Παρακάτω φαίνεται ο ψευδοκώδικας στον οποίο βασίζεται η υλοποίηση του αλγορίθμου (Εικόνα 3.2). Όπως στον EARS έτσι και στον SEARS κάθε φορά που η διεργασία *p* προγραμματίζεται για ένα τοπικό βήμα, στον ψευδοκώδικα δηλώνεται με την εκτέλεση μιας επανάληψης του κυρίως βρόγχου (main loop). Ο κώδικας του αλγορίθμου βρίσκεται στο τέλος της εργασίας στο Παράρτημα Β.

```
SEARS(r_p)
 1  ▷ Initialization:
 2  V(p) ← {r_p}        ▷ a set of rumors
 3  I(p) ← ∅            ▷ a set of pairs ⟨r, p⟩
 4  L(p) ← [n]          ▷ a set of processes
 5  sleep_cnt ← 0       ▷ an integer
 6
 7  repeat
 8    for every message m = ⟨V, I⟩ received do
 9        V(p) ← V(p) ∪ m.V
10        I(p) ← I(p) ∪ m.I
11    L(p) ← {q : ∃r ∈ V(p), (r, q) ∉ I(p)}
12    if L(p) = ∅
13        then  sleep_cnt ← sleep_cnt + 1
14        else  sleep_cnt ← 0
15    if sleep_cnt ≤ 1 then
16        ▷ Do epidemic transmission:
17        repeat Θ(max{n^ε, 1} log n) times:
18          Choose q uniformly at random from [n].
19          Send ⟨V(p), I(p)⟩ to process q.
20          for every r ∈ V(p) do
21              I(p) ← I(p) ∪ (r, q)
22          L(p) ← {q : ∃r ∈ V(p), (r, q) ∉ I(p)}
```

Εικόνα 3.2: Ψευδοκώδικας Αλγορίθμου SEARS για μια διεργασία p με αρχική φήμη r_p [3]

Η υλοποίηση του αλγορίθμου είναι ακριβώς η ίδια με αυτή του EARS με κάποιες διαφορές. Η πρώτη διαφορά είναι ότι η "shut-down" φάση συμβαίνει μία φορά μόνο. Δηλαδή κάθε διεργασία είναι προγραμματισμένη να παίρνει μόνο ένα "shut-down" βήμα. Πιο κάτω φαίνεται πώς διαφοροποιείται ο τρόπος με τον οποίο γίνεται ο έλεγχος της ακέραιας μεταβλητής sleep_cnt σε αντίθεση με τον EARS:

```
if (sleep_cnt <= 1){
. . .
}
```

Άλλη διαφοροποίηση είναι στην περίπτωση που η sleep_cnt βρίσκεται εντός των προκαθορισμένων ορίων. Τότε η διεργασία εκτελεί τρεις βασικές ενέργειες:

(1) Επιλέγει τυχαία και ομοιόμορφα τουλάχιστον $\max\{n^{\varepsilon}, 1\} \log n$ διεργασίες από το σύνολο διεργασιών και όχι μία όπως συμβαίνει στον EARS.

(2) Δημιουργεί και αποστέλλει ένα μήνυμα στις επιλεγείσες διεργασίες.

(3) Ανανεώνει τις λίστες I(p) και L(p). Για την τυχαία επιλογή διεργασίας χρησιμοποιείται η κλάση Random της βιβλιοθήκης Java. Το κομμάτι κώδικα που αφορά το σημείο αυτό είναι:

```
. . .
double numOfProcesses = StrictMath.max(StrictMath.pow(n, E), 1.0) *
(StrictMath.log(n) / StrictMath.log(2));
. . .
for (int i=0; i<numOfProcesses; i++){
    . . .
    int rand;
    int size = DestinationList.size();
    NodeID neighbor;
    rand = new Random().nextInt(size);
    neighbor = DestinationList.get(rand);
    . . .
}
```

όπου `numOfProcesses` είναι ο αριθμός των διεργασιών που επιλέγονται τυχαία σε κάθε βήμα, δηλαδή ο αριθμός $\max\{n^{\varepsilon}, 1\} \log n$, `DestinationList` είναι η λίστα με τους κόμβους – διεργασίες του συστήματος, `DestinationList.size()` η συνάρτηση που επιστρέφει τον αριθμό των κόμβων του συστήματος και `Random().nextInt(size)` η συνάρτηση – γεννήτρια ψευδοτυχαίων ακέραιων αριθμών μεταξύ του 0 και του `size-1`.

Όπως αναφέρθηκε και προηγουμένως για τον EARS, έτσι και για τον SEARS προστέθηκε στο περιβάλλον προσομοίωσης και υλοποίηση για σφάλματα η οποία περιγράφεται αναλυτικά στο Κεφάλαιο 4.

Κεφάλαιο 4

Αξιολόγηση Αλγορίθμων σε Περιβάλλον Προσομοίωσης

4.1 Μετρικές Αξιολόγησης Πολυπλοκότητας

Μετά τη φάση της υλοποίησης των δύο αλγορίθμων ακολουθεί η πειραματική αξιολόγησή τους. Απώτερος στόχος της αξιολόγησης αυτής είναι να διαφανεί σε ποιο βαθμό οι αλγόριθμοι είναι αποτελεσματικοί, αποδοτικοί και ανεκτικοί σε σφάλματα όχι μόνο στη θεωρία αλλά και στην πράξη.

Για μια πιο σωστή και μεθοδική πειραματική ανάλυση πρώτα καθορίζονται με ακρίβεια και σαφήνεια οι μετρικές αξιολόγησης που θα χρησιμοποιηθούν στη μελέτη των αποτελεσμάτων εφαρμογής των αλγορίθμων. Επειδή η αξιολόγηση εστιάζεται στην ευρωστία του κάθε αλγορίθμου και όχι στο κόστος ή στη δυσκολία υλοποίησής του, ως μετρικές αξιολόγησης επιλέγηκαν παράμετροι που δύναται να επηρεάσουν την ιδιότητα αυτή όπως ο χρόνος και τα μηνύματα που ανταλλάζονται. Συνεπώς για την πειραματική μελέτη των δύο αλγορίθμων καθορίστηκαν δύο μετρικές αξιολόγησης: η *πολυπλοκότητα μηνύματος* και η *πολυπλοκότητα χρόνου*.

Η πρώτη μετρική αφορά το συνολικό αριθμό των μηνυμάτων που αποστέλλονται από τις διεργασίες καθ’ όλη τη διάρκεια της εκτέλεσης. Η καταμέτρηση αρχίζει από το πρώτο μήνυμα που αποστέλλει η πρώτη διεργασία που συμμετέχει στη διαδικασία διάχυσης της πληροφορίας και τερματίζεται με το τελευταίο μήνυμα που αποστέλλεται από την τελευταία διεργασία πριν σταματήσει και αυτή οριστικά την αποστολή και παραλαβή μηνυμάτων. Ένας

απλός ψευδοκώδικας για την υλοποίηση του μετρητή μηνυμάτων για μια διεργασία *p* είναι ο εξής:

```
Για κάθε διεργασία έστω έχω μια ακέραια μεταβλητή msg_cnt. Τότε
msg_cnt=0;
for each msg sent{
    num_msg++;
}
```

Αφού καταμετρηθεί ο αριθμός των μηνυμάτων που αποστέλλει η κάθε ορθή διεργασία, στη συνέχεια καταγράφεται το άθροισμα όλων αυτών των μηνυμάτων ως εξής:

msg_complexity = $\sum_{i\in[n]} msg\_cnt_i$

Η δεύτερη μετρική αφορά το χρόνο ολοκλήρωσης του αλγορίθμου. Συγκεκριμένα μετριέται ο χρόνος ολοκλήρωσης της κάθε διεργασίας ξεχωριστά και από τους χρόνους αυτούς λαμβάνεται ο μέγιστος ως χρόνος ολοκλήρωσης του αλγορίθμου. Η καταμέτρηση γίνεται για κάθε διεργασία ξεχωριστά, από την αρχή της εκτέλεσης μέχρι το σημείο όπου αποστέλλεται και το τελευταίο μήνυμα. Για σκοπούς προσομοίωσης ο αλγόριθμος θεωρείται ότι εκτελείται σε χρονικά βήματα / γύρους. Σε κάθε βήμα εκτελείται η εργασία αποστολής (ενεργή μέθοδος) και παραλαβής (παθητική μέθοδος) ενός μηνύματος από κάθε έναν από τους κόμβους (για την ενεργή και την παθητική μέθοδο βλ. Υποκεφάλαιο 3.2.2). Η βιβλιοθήκη YALPS προσφέρει, ανάμεσα σε άλλες, μια βοηθητική μέθοδο του διαχειριστή εργασιών (Task Manager), την "getCurrentTimeMillis()" η οποία ανάλογα με την εφαρμογή, αν είναι προσομοίωση ή υλοποίηση, επιστρέφει σε χιλιοστά δευτερολέπτου τον τρέχοντα χρόνο προσομοίωσης ή τον πραγματικό χρόνο αντίστοιχα. Ο διαχειριστής εργασιών είναι ένα αντικείμενο της κλάσης `AbstractYalpsNode`. Έτσι με τη βοήθειά της μεθόδου αυτής υπολογίζεται το χρονικό διάστημα μετά από το οποίο μπορεί να εκτελεστεί ένα βήμα. Στην περίπτωση της προσομοίωσης εφόσον όλες οι διεργασίες ξεκινούν τη λειτουργία τους την ίδια στιγμή, η "getCurrentTimeMillis()" μπορεί να λειτουργήσει ως ένα εικονικό εξωτερικό ρολόι για όλους τους κόμβους το οποίο να μετρά αυτό το πλήθος των βημάτων που χρειάζεται ο αλγόριθμος για την ολοκλήρωση της λειτουργίας του. Αξίζει να σημειωθεί ότι

στην περίπτωση περιοδικών εργασιών η μέθοδος αυτή μπορεί να αντικατασταθεί με τη μέθοδο “getPeriod()” η οποία ουσιαστικά κάνει το ίδιο πράγμα. Περισσότερες πληροφορίες για το πώς λειτουργούν ακριβώς η “getCurrentTimeMillis()” και η “getPeriod()” υπάρχουν στα [2] και [9]. Ένας απλός ψευδοκώδικας για την υλοποίηση του μετρητή του χρόνου ολοκλήρωσης για μια διεργασία *p* είναι ο εξής:

```
Για κάθε διεργασία έχω δύο μεταβλητές start_clock και elapsed_time.
Τότε
start_clock = tm.getCurrentTimeMillis();
for each msg sent{
    elapsed_time = tm.getCurrentTimeMillis() – start_clock;
}
```

Όπου `tm` είναι το αντικείμενο της κλάσης `AbstractYalpsNode` που αντιπροσωπεύει το διαχειριστή εργασιών (Task Manager).

Αφού καταμετρηθεί ο χρόνος ολοκλήρωσης της κάθε διεργασίας, στη συνέχεια καταγράφεται ο μέγιστος από αυτούς χρόνος ως εξής:

$$time\_complexity = max_{i \epsilon [n]}(elapsed\_time_i)$$

Αξίζει να σημειωθεί ότι για τη μέτρηση της πολυπλοκότητας μηνύματος γίνεται καταμέτρηση μόνο του συνολικού αριθμού των μηνυμάτων που αποστέλλονται και όχι του συνολικού αριθμού των bits που μεταφέρονται ο οποίος εξαρτάται από το μέγεθος του μηνύματος. Αυτό γίνεται διότι ο αριθμός των bits θα περιέγραφε την απόδοση των αλγορίθμων ως προς συγκεκριμένα μηνύματα και τα πειραματικά αποτελέσματα που θα λαμβάνονταν δεν θα έδιναν πιο γενικά συμπεράσματα. Αντιθέτως, ο αριθμός των μηνυμάτων είναι μια πιο αντικειμενική μετρική και δεν εξαρτάται από το μέγεθος της πληροφορίας που θα ανατεθεί στους κόμβους. Επομένως, τα συμπεράσματα που θα ληφθούν θα είναι πιο αξιόπιστα και εν τέλει πάλι θα δίνουν μια εικόνα για την απόδοση του συστήματος.

4.2 Μεθοδολογία

Όπως ήταν φυσικό υπήρξαν κάποιοι περιορισμοί κατά την προσομοίωση της εφαρμογής. Κατά την εκτέλεση στο περιβάλλον προσομοίωσης, όλοι οι κόμβοι του συστήματος έτρεχαν τις λειτουργίες τους πάνω στην ίδια μηχανή. Συνεπώς ο αριθμός των κόμβων που μπορούσαν να χρησιμοποιηθούν στα πειράματα περιορίζεται λόγω της περιορισμένης μνήμης της μηχανής. Ο μέγιστος αριθμός κόμβων που ήταν τελικά εφικτό να χρησιμοποιηθούν στην προσομοίωση ήταν 128 κόμβοι. Έγιναν προσομοιώσεις για αυξανόμενο αριθμό κόμβων από 2 μέχρι 128 ανά δύναμη του 2, δηλαδή για 2, 4, 8, 16, 32, 64 και 128 κόμβους. Αυτό επαναλήφθηκε πέντε φορές για να παρθεί στο τέλος ο μέσος όρος των αντίστοιχων περιπτώσεων προσομοίωσης έτσι ώστε οι τελικές μετρήσεις της προσομοίωσης να είναι πιο ρεαλιστικές.

Λήφθηκαν ποικίλες μετρήσεις για διαφορετικά σενάρια. Πρώτα, εξετάστηκαν ξεχωριστά οι περιπτώσεις όπου οι χρονικές καθυστερήσεις d και δ είναι σταθερές καθ’ όλη τη διάρκεια της προσομοίωσης. Μ’ αυτόν τον τρόπο μελετήθηκε η ειδική περίπτωση που οι κόμβοι στο δίκτυο είναι συγχρονισμένοι ώστε να διαφανεί κατά πόσο ο αλγόριθμος είναι εύρωστος και σε σύγχρονες εκτελέσεις. Διευκρινίζεται βέβαια ότι οι κόμβοι δεν γνωρίζουν αν είναι συγχρονισμένοι ή όχι, δηλαδή δεν γνωρίζουν τις τιμές των d και δ. Η περίπτωση αυτή υλοποιήθηκε εύκολα με την προσθήκη μιας σταθερής συνολικής καθυστέρησης d+δ πριν την αποστολή ενός μηνύματος. Έτσι κάθε φορά που θα αποστέλλεται οποιοδήποτε μήνυμα θα καθυστερείται η αποστολή του κατά χρόνο ίσο με d+δ. Επειδή στο [3] γίνεται η υπόθεση ότι $d, \delta \geq 1$ για τις σύγχρονες εκτελέσεις θεωρείται ότι η ελάχιστη συνολική καθυστέρηση d+δ που μπορεί να συμβεί είναι ίση με 2.

Έπειτα διερευνήθηκε η περίπτωση όπου οι χρονικές καθυστερήσεις d και δ δεν είναι σταθερές αλλά μεταβάλλονται χωρίς να είναι γνωστές από την αρχή. Η περίπτωση αυτή υλοποιήθηκε με την προσθήκη τυχαίας καθυστέρησης πριν την αποστολή κάθε μηνύματος. Για σκοπούς υλοποίησης έπρεπε να καθοριστεί από την αρχή κάποιο επιτρεπτό διάστημα καθυστέρησης. Επειδή, όπως θα φανεί στο επόμενο κεφάλαιο, στο PlanetLab ο μέγιστος

χρόνος που χρειάζεται να σταλεί ένα μήνυμα δεν είναι μεγαλύτερος από 200 ms επιλέχθηκε ένα σχετικά μικρό διάστημα καθυστέρησης, μεταξύ 2 και 100 ms, ώστε τα αποτελέσματα της προσομοίωσης να είναι όσο το δυνατό πιο ρεαλιστικά.

Ένας άλλος σημαντικός παράγοντας που λήφθηκε υπόψη κατά την αξιολόγηση ήταν τα σφάλματα κατάρρευσης[6] που συμβαίνουν στους κόμβους. Αρχικά ο αλγόριθμος εφαρμόστηκε με πιθανότητα 1 να μην εμφανιστεί κανένα σφάλμα κατάρρευσης. Έτσι παρόλο που στην υλοποίηση αρχικοποιήθηκε μια μεταβλητή f η οποία υποδηλώνει το συνολικό αριθμό σφαλμάτων που ενδέχεται να συμβούν κατά την εκτέλεση του αλγορίθμου, εντούτοις διασφαλίστηκε ότι κανένας κόμβος δεν καταρρέει. Στη συνέχεια, για σκοπούς σύγκρισης και περαιτέρω μελέτης προστέθηκε και μια πιθανότητα πραγματικής εμφάνισης f σφαλμάτων κατά την εκτέλεση, η οποία υπολογίστηκε με βάση το θεωρητικό άνω φράγμα για την πολυπλοκότητα του χρόνου και με βάση το μέγιστο αριθμό σφαλμάτων που μπορεί να συμβούν.

Συγκεκριμένα, για τον αλγόριθμο EARS η πιθανότητα p ο κάθε κόμβος να καταρρεύσει στη διάρκεια της προσομοίωσης είναι $p < \frac{f}{\frac{n}{n-f}\log^2 n(d+\delta)}$ όπου $\frac{n}{n-f}\log^2 n(d+\delta)$ είναι το θεωρητικό άνω φράγμα για το χρόνο ολοκλήρωσης του αλγορίθμου και f ο αριθμός των πιθανών σφαλμάτων κατάρρευσης. Αυτό προκύπτει από τον ακόλουθο συλλογισμό: έστω ότι υπάρχουν n κόμβοι στο σύστημα, f πιθανές αποτυχίες και x ο αναμενόμενος αριθμός των γύρων επικοινωνίας που αποτελούν το συνολικό χρόνο ολοκλήρωσης του αλγορίθμου. Τότε η πιθανότητα κατάρρευσης ενός κόμβου σε κάθε γύρο επικοινωνίας είναι $\frac{p}{n}$ όπου p η ζητούμενη πιθανότητα. Επειδή ο αναμενόμενος αριθμός των γύρων επικοινωνίας είναι x, υπάρχει πιθανότητα $\frac{p}{n}x$ να καταρρεύσει ο κάθε κόμβος κατά τη διάρκεια της προσομοίωσης. Όμως από τη θεωρητική ανάλυση, αν σε κάθε προσομοίωση υπάρχει η πιθανότητα να καταρρεύσουν μέχρι και f κόμβοι από τους n, τότε η πιο πάνω πιθανότητα πρέπει να είναι μικρότερη από την πιθανότητα να συμβούν f σφάλματα. Συνεπώς πρέπει να ισχύει $\frac{p}{n}x < \frac{f}{n}$,

[6] Υπενθυμίζεται ότι ένα «σφάλμα κατάρρευσης κόμβου» συμβαίνει όταν ο κόμβος σε κάποια χρονική στιγμή και για κάποιο λόγο διακόπτει οριστικά τη λειτουργία αποστολής και λήψης μηνυμάτων. Για περαιτέρω εξήγηση βλ. Υποκεφάλαιο 3.3

δηλαδή p < $\frac{f}{\mathrm{x}}$. Μετά από αρκετές δοκιμές για την εύρεση της καλύτερης τιμής πιθανότητας που θα μπορούσε να χρησιμοποιηθεί στα πειράματα αποφασίστηκε τελικά να εφαρμοστεί η τιμή $\frac{f}{n\mathrm{x}}$ η οποία πάλι ικανοποιεί το άνω φράγμα $\frac{f}{\mathrm{x}}$.

Παρόμοια καθορίστηκε η πιθανότητα και στον αλγόριθμο SEARS. Όμως επιπλέον έπρεπε να αποφασιστεί η τιμή της σταθεράς *ε*. Κατά τη διάρκεια των πειραμάτων δοκιμάστηκαν διάφορες τιμές για τη σταθερά *ε<1* ανάμεσά τους οι τιμές *ε* = $\frac{1}{2}$, $\frac{1}{10}$ και $\frac{1}{100}$. Και για τις τρεις τιμές τα αποτελέσματα ήταν εξίσου ικανοποιητικά, όμως τελικά επιλέχθηκε η τιμή *ε* = $\frac{1}{100}$ διότι θεωρήθηκε ότι αυτή δίνει τα πιο καλά και πιο ενδιαφέροντα αποτελέσματα. Συνεπώς τα πειράματα που έγιναν για τον αλγόριθμο SEARS αφορούν την περίπτωση εφαρμογής της τιμής *ε* = $\frac{1}{100}$.

Όσον αφορά σε θέματα υλοποίησης των σφαλμάτων, στην αρχή της ενεργής συνάρτησης ελέγχεται κατά πόσο ο κόμβος έχει καταρρεύσει ή όχι. Σε περίπτωση κατάρρευσης επιστρέφει από τη συνάρτηση χωρίς να εκτελεί καμία λειτουργία. Ο έλεγχος γίνεται χρησιμοποιώντας μια τοπική μεταβλητή του κόμβου με την ονομασία *failure* η οποία αρχικοποιείται με την τιμή *false* και παίρνει την τιμή *true* μόνο όταν ο κόμβος καταρρέει. Στο τέλος της ενεργής συνάρτησης γίνεται η πιθανή κατάρρευση του κόμβου. Για την αυθαίρετη κατανομή πιθανοτήτων χρησιμοποιείται μια γεννήτρια τυχαίων πραγματικών αριθμών μεταξύ 0 και 1. Παράγεται ένας τέτοιος αριθμός και ελέγχεται κατά πόσο είναι μικρότερος από $\frac{f}{n\mathrm{x}}$. Αν ναι τότε ο κόμβος θα καταρρεύσει και η μεταβλητή *failure* θα πάρει την τιμή *true* διαφορετικά θα παραμείνει η τιμή *false*.

Στο παρακάτω τμήμα κώδικα φαίνονται οι διαφοροποιήσεις που έγιναν στην υλοποίηση ώστε να προσομοιώνεται η εμφάνιση πιθανών σφαλμάτων κατάρρευσης κόμβων:

```
protected void myMsgSenderTask() {
    if (failure)
       return;
    . . .
    double fail = Math.random();
    x=n/(n-f)*StrictMath.pow(StrictMath.log(n)/StrictMath.log(2.0), 2.0)*2;
    if (fail < f/(n*x)){
       failure = true;
    }
}
```

Όσον αφορά στην παθητική συνάρτηση η διαφοροποίησή της γίνεται μόνο στην αρχή και είναι η ίδια με την ενεργή συνάρτηση. Δηλαδή υπάρχει η εξής διαφοροποίηση:

```
public boolean receive(short header, Object msg, NodeID sender) {
    if (failure)
       return true;
    . . .
}
```

4.3 Πειραματική Αξιολόγηση Αλγορίθμου EARS

Ακολουθεί η πειραματική αξιολόγηση του αλγορίθμου EARS στο περιβάλλον προσομοίωσης που προφέρει η βιβλιοθήκη YALPS.

4.3.1 Περίπτωση με σταθερό d+δ

Στο υποκεφάλαιο που ακολουθεί εξετάζεται ο αλγόριθμος EARS στην περίπτωση όπου οι χρονικές καθυστερήσεις d και δ είναι γνωστές από την αρχή και παραμένουν σταθερές καθ’

όλη τη διάρκεια της προσομοίωσης. Συγκεκριμένα εξετάζονται δύο περιπτώσεις, η πρώτη χωρίς εμφάνιση πραγματικών αποτυχιών και η δεύτερη με πιθανές αποτυχίες. Σε πρώτη φάση τα αποτελέσματα τυγχάνουν σύγκρισης με τα αντίστοιχα θεωρητικά για να αποφανθεί κατά πόσο οι θεωρητικές μετρήσεις επιβεβαιώνονται και στην πράξη. Ύστερα συγκρίνονται οι διάφορες περιπτώσεις μεταξύ τους και σημειώνονται σημαντικά συμπεράσματα. Όλες οι συγκρίσεις γίνονται με μετρικές αξιολόγησης τον αριθμό των μηνυμάτων που αποστέλλονται κατά την εκτέλεση και το χρόνο ολοκλήρωσης του αλγορίθμου.

4.3.1.1 Αποτελέσματα εφαρμογής χωρίς σφάλματα

Ακολουθούν πίνακες μετρήσεων που πάρθηκαν από το **μέσο όρο πέντε προσομοιώσεων** του αλγορίθμου για κάθε μια από τις περιπτώσεις που εξετάστηκαν. Οι μετρήσεις χαρακτηρίζουν δύο παραμέτρους αξιολόγησης:

(1) το συνολικό αριθμό μηνυμάτων που στάλθηκαν κατά τη διάρκεια της προσομοίωσης από όλους τους κόμβους του συστήματος που δεν κατέρρευσαν

(2) το μέγιστο χρόνο (σε τοπικά βήματα) μέσα στον οποίο ολοκληρώνει τη λειτουργία του κάθε κόμβος που δεν κατέρρευσε

Από τους πίνακες αυτούς πάρθηκαν γραφικές παραστάσεις των δύο παραμέτρων σε σχέση με τον αριθμό των κόμβων. Ως σταθερή καθυστέρηση στα πειράματα και στις συγκρίσεις με τη θεωρία ορίστηκε η τιμή 2ms για πιο ρεαλιστικά αποτελέσματα.

Πιο κάτω φαίνονται οι μετρήσεις για f=1 και f=n/4 στην περίπτωση που δεν σημειώνεται καμιά κατάρρευση κόμβου κατά την προσομοίωση (Πίνακες 4.1 και 4.2):

	Κόμβοι	Μηνύματα	Χρόνος Ολοκλήρωσης
EARS **d+δ**=σταθερό **f=1** χωρίς καταρρεύσεις κόμβων	2	5,00	3,67
	4	31,67	10,67
	8	119,00	17,00
	16	332,67	26,33
	32	853,67	31,67
	64	2039,67	36,00
	128	4694,67	41,67

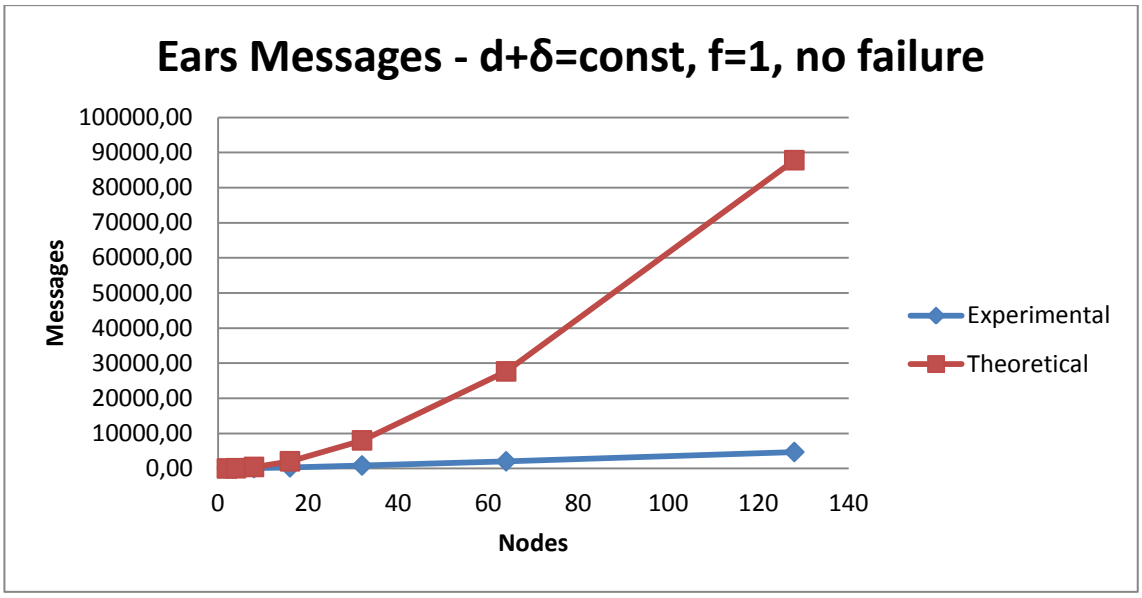
Πίνακας 4.1: Μετρήσεις EARS για f=1 και d+δ σταθερό (χωρίς κατάρρευση κόμβου)

	Κόμβοι	Μηνύματα	Χρόνος Ολοκλήρωσης
EARS **d+δ**=σταθερό **f=n/4** χωρίς καταρρεύσεις κόμβων	2	3,33	5,33
	4	34,33	12,00
	8	118,33	21,00
	16	352,67	27,00
	32	939,00	33,33
	64	2160,00	38,67
	128	5160,00	46,00

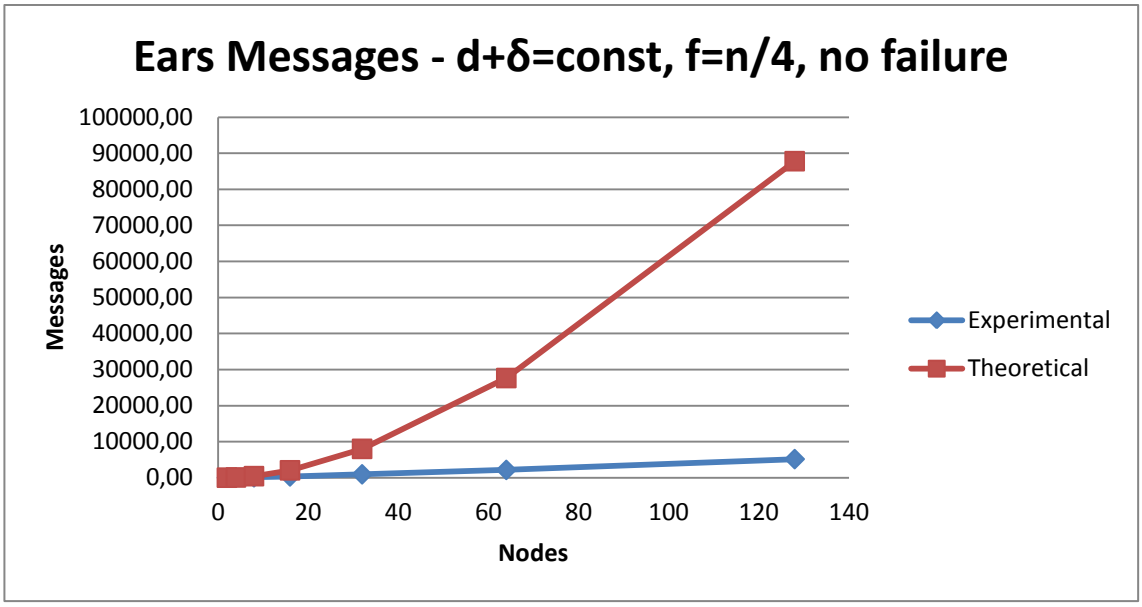
Πίνακας 4.2: Μετρήσεις EARS για f=n/4 και d+δ σταθερό (χωρίς κατάρρευση κόμβου)

Για τα μηνύματα θεωρητικά έχει αποδειχθεί [3] ότι ο συνολικός αριθμός των μηνυμάτων που αποστέλλονται κατά την εκτέλεση του αλγορίθμου είναι της τάξεως $O(n \log^3 n (d+\delta))$ όπου n ο αριθμός των κόμβων και d, δ οι καθυστερήσεις. Δηλαδή ο αριθμός των μηνυμάτων εξαρτάται από τον αριθμό των κόμβων και όσο αυξάνονται οι κόμβοι στο σύστημα αυξάνονται και τα μηνύματα που αποστέλλονται. Αυτό είναι προφανές και αναμενόμενο αφού καθώς αυξάνονται οι κόμβοι στο σύστημα, αυξάνεται και η πληροφορία που πρέπει να διαδοθεί ώστε να ενημερωθούν όλοι. Κατά συνέπεια αυξάνονται και τα μηνύματα που πρέπει να σταλούν.

Παρακάτω φαίνονται οι γραφικές παραστάσεις του αριθμού των μηνυμάτων για f=1 και f=n/4 σε σύγκριση με τις αντίστοιχες θεωρητικές καμπύλες (Εικόνες 4.1 και 4.2):



Εικόνα 4.1: Πειραματική και Θεωρητική προσέγγιση EARS για f=1 χωρίς σφάλματα



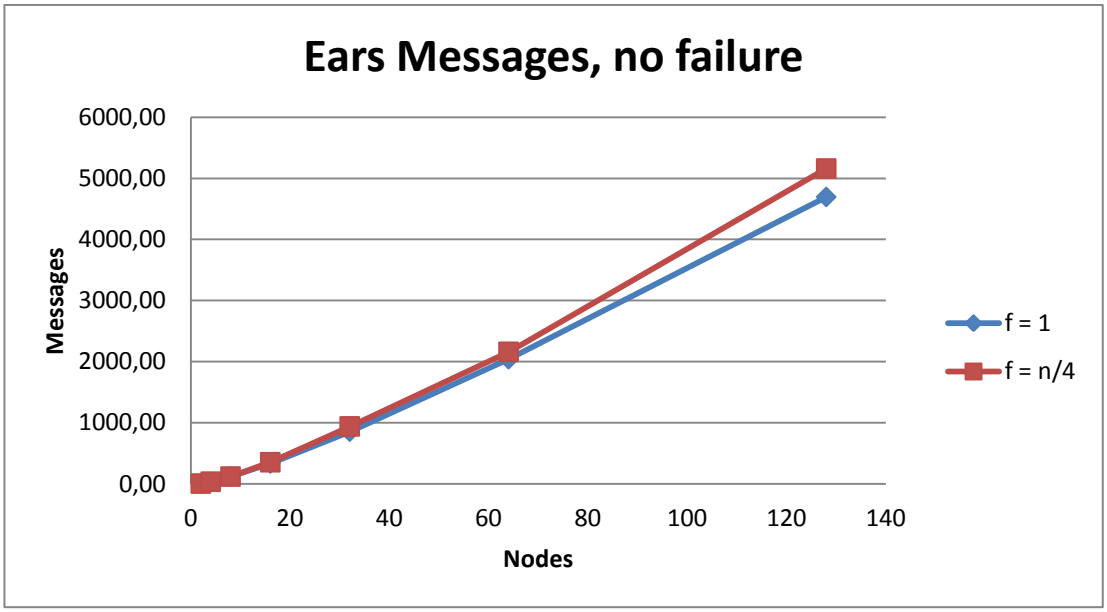
Εικόνα 4.2: Πειραματική και Θεωρητική προσέγγιση EARS για f=n/4 χωρίς σφάλματα

Από τα πειράματα φαίνεται ότι πράγματι ο αριθμός των μηνυμάτων μεγαλώνει καθώς μεγαλώνει ο αριθμός των κόμβων. Αρχικά σχηματίζεται μια μικρή καμπύλη, η οποία δικαιολογεί το *n log n* στο θεωρητικό άνω φράγμα, ενώ στη συνέχεια φαίνεται μια πιο γραμμική συμπεριφορά που πλησιάζει αρκετά κοντά στη βέλτιστη τιμή (*O(n)*). Αυτό είναι

λογικό αφού η θεωρητική ανάλυση είναι χείριστης περίπτωσης. Με λίγα λόγια τα πειράματα υποδεικνύουν ότι ο αλγόριθμος αποδίδει καλύτερα από το αναμενόμενο στην πράξη.

Συγκρίνοντας τα πειραματικά αποτελέσματα με τα θεωρητικά επιβεβαιώνεται η ορθότητα του αλγορίθμου και επαληθεύεται το θεωρητικό άνω φράγμα. Δεν υπάρχει πρόβλημα αν τα πειραματικά αποτελέσματα είναι πολύ χαμηλότερα από τα αντίστοιχα θεωρητικά, το αντίθετο μάλιστα. Η όλη ανάλυση του αλγορίθμου έγινε με γνώμονα τη χείριστη περίπτωση και στην πράξη, τα αποτελέσματα σίγουρα πρέπει να είναι καλύτερα.

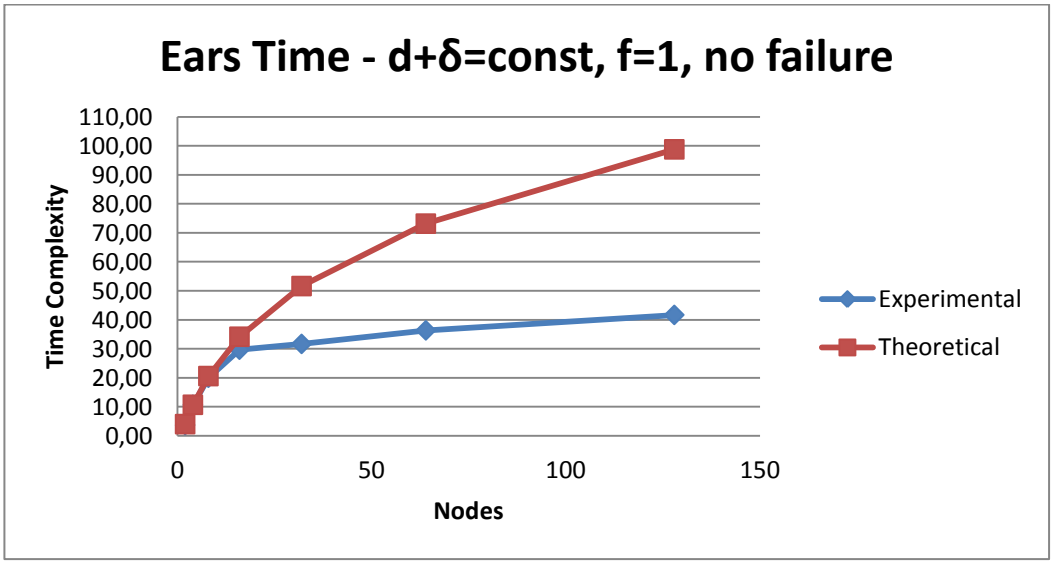
Τέλος, παρατηρώντας τους πίνακες μετρήσεων για f=1 και f=n/4 αλλά και τη γραφική παράσταση που ακολουθεί ο αριθμός των μηνυμάτων στις αντίστοιχες περιπτώσεις δεν διαφέρει σημαντικά. Το γεγονός αυτό συνάδει με τη θεωρητική ανάλυση, σύμφωνα με την οποία ο αριθμός των μηνυμάτων είναι ανεξάρτητος από την επιλογή της τιμής του f. Οι μικροδιαφορές που εντοπίζονται στην πράξη ίσως να οφείλονται στην τυχαία επιλογή του κάθε κόμβου πριν αποσταλεί σ' αυτόν ένα νέο μήνυμα καθώς επίσης και στα όρια που καθορίζονται στην υλοποίηση τα οποία εξαρτώνται από την παράμετρο f (βλ. επεξήγηση για μεταβλητή *sleep_cnt* στο Κεφάλαιο 3).



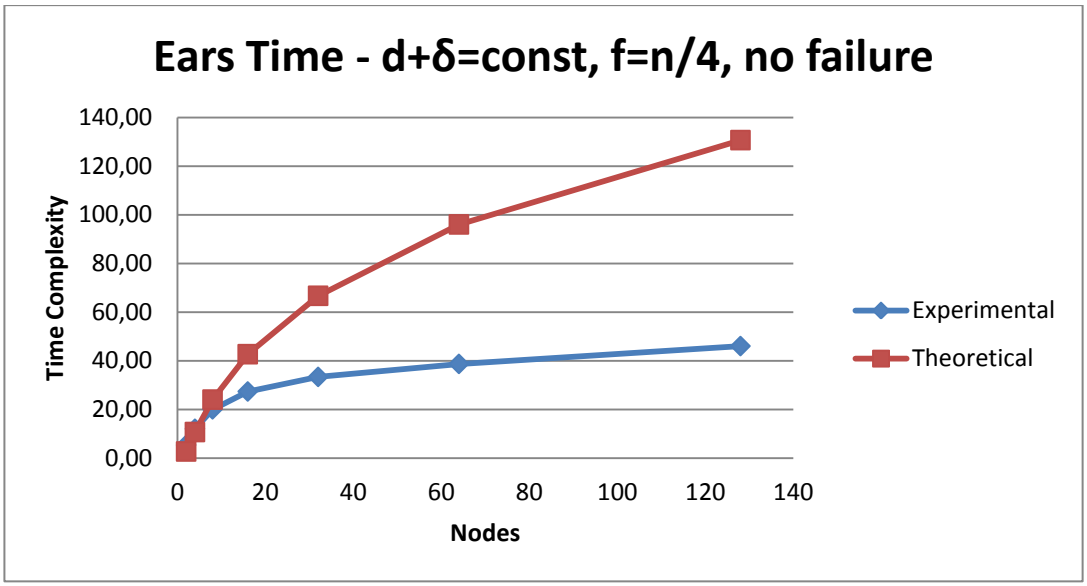
Εικόνα 4.3: Σύγκριση αριθμού Μηνυμάτων EARS για f=1 και f=n/4, χωρίς σφάλματα

Για το χρόνο ολοκλήρωσης θεωρητικά έχει αποδειχθεί [3] ότι κάθε κόμβος που δεν έχει καταρρεύσει κατά την εκτέλεση ολοκληρώνει τη λειτουργία του μέσα σε χρόνο τάξεως $O\left(\frac{n}{n-f}\log^2 n(d+\delta)\right)$ όπου n ο αριθμός των κόμβων και d, δ οι καθυστερήσεις. Το γεγονός αυτό οδηγεί στο συμπέρασμα ότι ο χρόνος ολοκλήρωσης και ο αριθμός των κόμβων συνδέονται μεταξύ τους μέσω μίας σχέσης εξάρτησης κατά την οποία καθώς αυξάνεται ο αριθμός των κόμβων αυξάνεται και ο χρόνος που χρειάζεται μέχρι να ενημερωθούν όλοι οι "ζωντανοί" κόμβοι του συστήματος. Αυτό είναι λογικό, αφού όταν αυξάνονται οι κόμβοι στο σύστημα πρέπει να ενημερωθούν περισσότεροι. Κατά συνέπεια αυξάνονται τα τοπικά βήματα που απαιτείται να εκτελεστούν για το σκοπό αυτό άρα και ο χρόνος. Παρατηρώντας τους αντίστοιχους πίνακες μετρήσεων, πράγματι ο χρόνος ολοκλήρωσης μεγαλώνει όσο μεγαλώνει ο αριθμός των κόμβων και παράλληλα μεγαλώνει και ο ρυθμός αύξησης. Γενικά μια τέτοια συμπεριφορά θυμίζει παρόμοια με λογαριθμική κι αυτό ίσως να δικαιολογεί την ύπαρξη του λογαρίθμου στο θεωρητικό άνω φράγμα.

Παρακάτω φαίνονται οι γραφικές παραστάσεις του χρόνου ολοκλήρωσης συναρτήσει του αριθμού των κόμβων του συστήματος για f=1 και f=n/4 στην περίπτωση που δε σημειώνεται καμιά αποτυχία (κατάρρευση) καθ' όλη τη διάρκεια της προσομοίωσης. Αυτές αντιπαραβάλλονται με τις αντίστοιχες θεωρητικές γραφικές παραστάσεις:



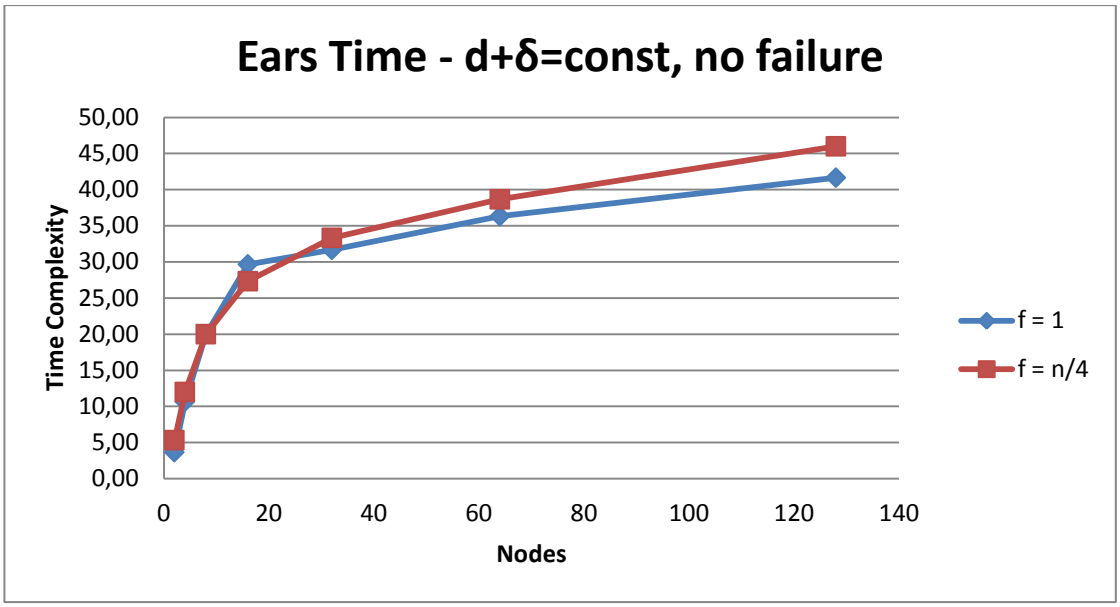
Εικόνα 4.4: Πειραματική και Θεωρητική προσέγγιση EARS για f=1 χωρίς αποτυχίες



Εικόνα 4. 5: Πειραματική και Θεωρητική προσέγγιση EARS για f=n/4 χωρίς αποτυχίες

Συγκρίνοντας τα πειραματικά αποτελέσματα με τα αντίστοιχα θεωρητικά, επιβεβαιώνεται η ορθότητα του αλγορίθμου και επαληθεύεται το θεωρητικό άνω φράγμα που εκφράζει το συνολικό χρόνο που χρειάζεται ο αλγόριθμος για να ολοκληρώσει την εκτέλεσή του και να ενημερωθούν όλοι οι κόμβοι του συστήματος.

Τέλος, παρατηρώντας τους πίνακες μετρήσεων για f=1 και f=n/4 αλλά και τη γραφική παράσταση που ακολουθεί ο χρόνος ολοκλήρωσης στις αντίστοιχες περιπτώσεις διαφέρει. Όπως ήταν αναμενόμενο ο χρόνος στην περίπτωση που f=n/4 είναι μεγαλύτερος από την αντίστοιχη περίπτωση όπου f=1. Αυτό οφείλεται στο γεγονός ότι παρόλο που δε σημειώνεται κανένα σφάλμα κατά την εκτέλεση, εντούτοις ο αλγόριθμος θεωρεί ότι πιθανό να συμβούν μέχρι f σφάλματα (είτε 1 είτε n/4). Οπότε καθορίζει τη διάρκεια της "shut-down" φάσης, λαμβάνοντας υπόψη στους υπολογισμούς την τιμή αυτή. Υπενθυμίζεται ότι στην υλοποίηση η διάρκεια ορίστηκε ως η ποσότητα $2\frac{n}{n-f}\log n$. Επειδή για f=n/4 όπου n αρκετά μεγάλο, η ποσότητα $\frac{n}{n-f}$ είναι πάντα μεγαλύτερη από την αντίστοιχη ποσότητα για f=1, έπεται ότι για τον ίδιο αριθμό κόμβων και f=n/4 η "shut-down" φάση θα έχει πάντα μεγαλύτερη διάρκεια με αποτέλεσμα ο αλγόριθμος να σημειώνει μεγαλύτερο χρόνο ολοκλήρωσης.



Εικόνα 4.6: Σύγκριση Χρόνου Ολοκλήρωσης EARS για f=1 και f=n/4, χωρίς σφάλματα

4.3.1.2 Αποτελέσματα εφαρμογής με σφάλματα

Ενώ προηγουμένως μελετήθηκαν οι περιπτώσεις όπου f=1 και f=n/4 χωρίς σφάλματα κατάρρευσης, στη συνέχεια μελετήθηκαν οι ίδιες περιπτώσεις αλλά με πιθανότητα να συμβούν τέτοια σφάλματα. Ακολουθούν πίνακες μετρήσεων για f=1 και f=n/4 με πιθανότητα κατάρρευσης ίση με $\frac{f}{nx}$ (βλ. επεξήγηση για την πιθανότητα στο Κεφάλαιο 4).

	Κόμβοι	Μηνύματα	Χρόνος Ολοκλήρωσης
EARS d+δ=σταθερό f=1 κατάρρευση κόμβου	2	2,60	3,20
	4	18,00	14,00
	8	97,00	17,00
	16	297,00	26,33
	32	847,67	32,33
	64	1968,00	36,00
	128	3941,00	41,47

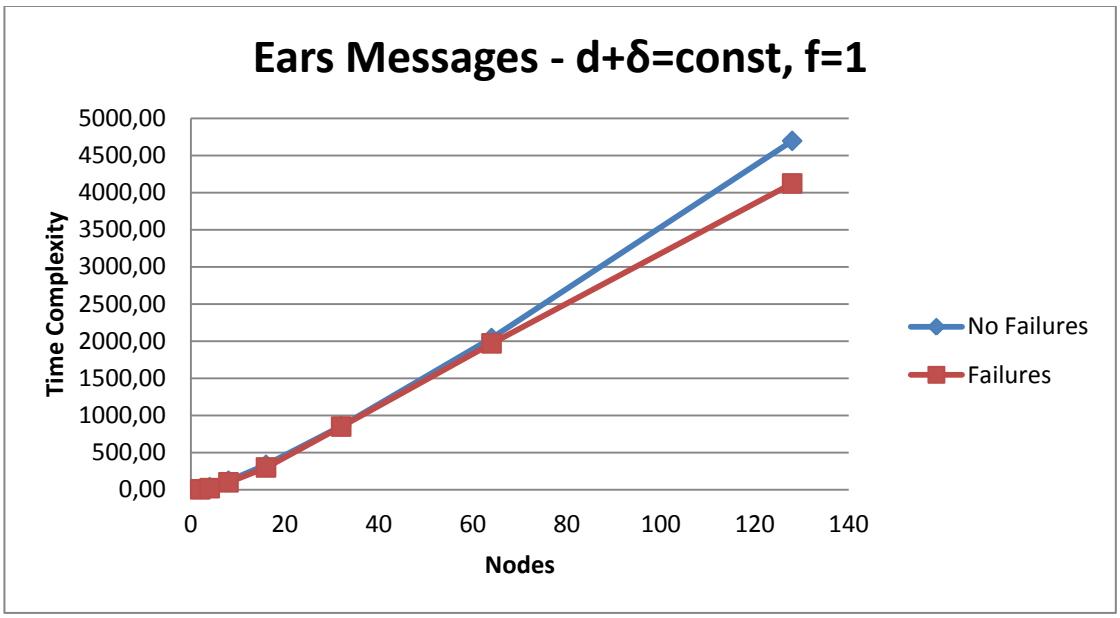
Πίνακας 4.3: Μετρήσεις EARS για f=1 και d+δ σταθερό (με κατάρρευση κόμβου)

	Κόμβοι	Μηνύματα	Χρόνος Ολοκλήρωσης
EARS d+δ=σταθερό f=n/4 καταρρεύσεις κόμβων	2	1,67	2,00
	4	28,67	14,00
	8	104,00	19,00
	16	313,33	27,00
	32	877,67	35,33
	64	2042,00	40,00
	128	4052,00	53,00

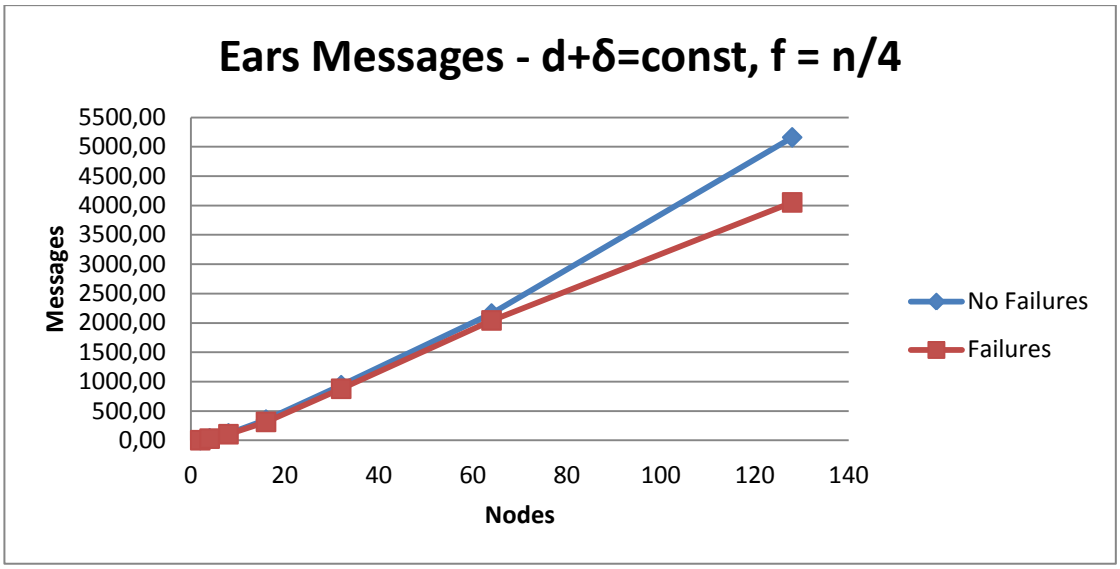
Πίνακας 4.4: Μετρήσεις EARS για f=n/4 και d+δ σταθερό (με κατάρρευση κόμβου)

Ως προς τον αριθμό μηνυμάτων τα αποτελέσματα είναι εντελώς παρόμοια με τα αντίστοιχα χωρίς σφάλματα. Ο αριθμός μηνυμάτων εξακολουθεί να είναι ανεξάρτητος από τον αριθμό των σφαλμάτων επιβεβαιώνοντας το γεγονός πως ακόμα και αν πραγματικά παρουσιαστούν καταρρεύσεις κόμβων ο αλγόριθμος δεν θα χάσει την ευρωστία του.

Παρακάτω φαίνονται οι γραφικές παραστάσεις του αριθμού των μηνυμάτων που αποστέλλει ο αλγόριθμος συναρτήσει του αριθμού των κόμβων του συστήματος για f=1 και f=n/4 σε σύγκριση με τις αντίστοιχες χωρίς σφάλματα:



Εικόνα 4.7: Πειραματική σύγκριση Μηνυμάτων EARS για f=1 με και χωρίς αποτυχίες



Εικόνα 4.8: Πειραματική σύγκριση Μηνυμάτων EARS για f=n/4 με και χωρίς αποτυχίες

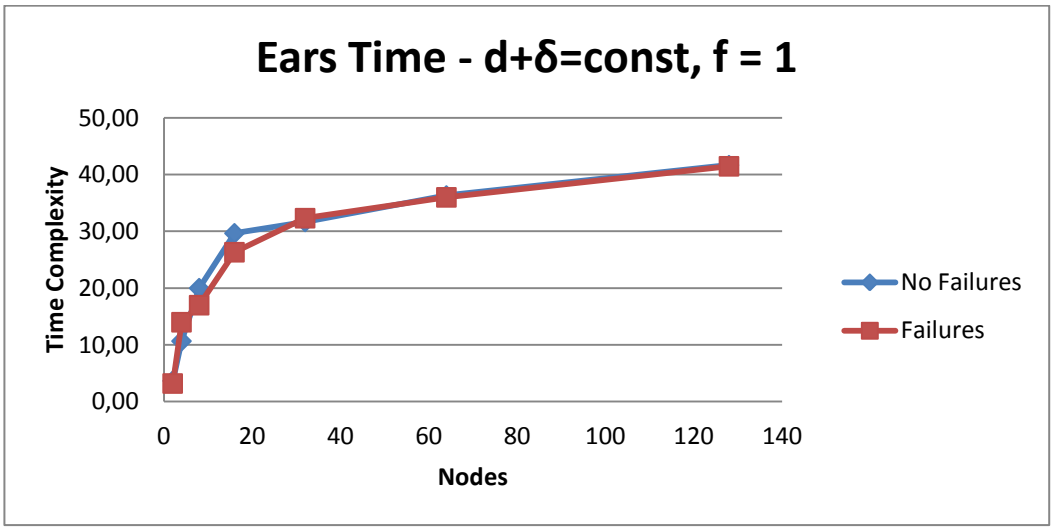
Από τα πειραματικά αποτελέσματα για f=1 φαίνεται ότι ένα σφάλμα δεν αποτελεί τόσο μεγάλη απώλεια στο σύστημα που να είναι ικανή να επηρεάσει σημαντικά την πολυπλοκότητα του αλγορίθμου. Γι' αυτό και είτε συμβεί ένα σφάλμα είτε όχι, τα αποτελέσματα είναι αρκετά κοντινά.

Τώρα για f=n/4 θα ανέμενε κανείς ότι τα πράγματα θα είναι διαφορετικά. Από τη στιγμή που καταρρέουν κάποιοι κόμβοι και άρα σταματούν να στέλλουν μηνύματα, είναι λογικό να μειωθεί ο συνολικός αριθμός των μηνυμάτων του αλγορίθμου. Πράγματι, όπως δείχνουν τα αποτελέσματα συγκριτικά, μειώνονται τα μηνύματα που αποστέλλει ο αλγόριθμος όμως η διαφορά δεν είναι πολύ μεγάλη. Αυτό οδηγεί στο γενικότερο συμπέρασμα ότι τελικά είτε συμβούν μέχρι και n/4 σφάλματα είτε όχι, η πολυπλοκότητα του αλγορίθμου δεν επηρεάζεται σε μεγάλο βαθμό. Ο λόγος είναι η δομή του αλγορίθμου που εξαρτάται από την τιμή του f και όχι από τον πραγματικό αριθμό σφαλμάτων που συμβαίνουν. Από αυτό έπεται ότι ο αλγόριθμος είναι κατασκευασμένος με τέτοιο τρόπο ώστε, ανάλογα με την τιμή του f, στέλλει περισσότερα μηνύματα από όσα πραγματικά χρειάζεται να στείλει για να ενημερωθούν οι κόμβοι. Έτσι ακόμη κι αν συμβούν μέχρι f σφάλματα, τα επιπλέον μηνύματα «αντικαθιστούν» τα χαμένα μηνύματα κι έτσι τελικά οι επιζώντες κόμβοι ενημερώνονται

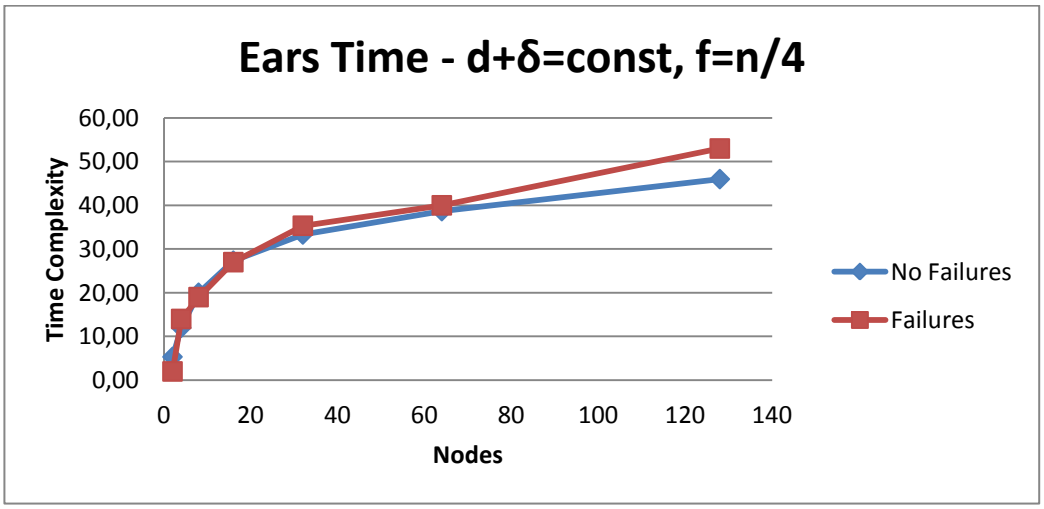
ορθά ακόμη κι αν άλλοι γείτονές τους έχουν καταρρεύσει χωρίς να επηρεάζεται η πολυπλοκότητα του αλγορίθμου.

Ως προς το χρόνο ολοκλήρωσης του αλγορίθμου τα αποτελέσματα είναι κι εδώ παρόμοια με τα αντίστοιχα χωρίς σφάλματα. Ο χρόνος ολοκλήρωσης για f=1 εξακολουθεί να είναι μικρότερος σε σχέση με το χρόνο για f=n/4.

Παρακάτω φαίνονται οι γραφικές παραστάσεις του χρόνου ολοκλήρωσης του αλγορίθμου συναρτήσει του αριθμού των κόμβων του συστήματος για f=1 και f=n/4 σε σύγκριση με τις αντίστοιχες χωρίς σφάλματα:



Εικόνα 4.9: Πειραματική σύγκριση EARS για f=1 με και χωρίς αποτυχίες



Εικόνα 4.10: Πειραματική σύγκριση EARS για f=n/4 με και χωρίς αποτυχίες

Όπως και στα μηνύματα έτσι και στο χρόνο για f=1 δεν αλλάζει τίποτα ουσιαστικά. Ένα μόνο σφάλμα στο σύστημα δεν μπορεί να επηρεάσει την πολυπλοκότητα του αλγορίθμου. Γι' αυτό είτε συμβεί ένα σφάλμα είτε όχι, τα αποτελέσματα στις αντίστοιχες περιπτώσεις είναι πολύ κοντινά.

Για f=n/4 τα αποτελέσματα πάλι δεν διαφέρουν πολύ σε σχέση με την αντίστοιχη περίπτωση χωρίς σφάλματα. Από τη στιγμή που καταρρέουν κάποιοι κόμβοι και άρα σταματούν να στέλλουν μηνύματα, είναι λογικό να καθυστερούν την ενημέρωση των υπόλοιπων κόμβων κι έτσι να αυξάνεται ο χρόνος ολοκλήρωσης του αλγορίθμου. Πράγματι, όπως δείχνουν τα αποτελέσματα συγκριτικά, ο χρόνος είναι αυξημένος όμως η διαφορά δεν είναι πολύ μεγάλη. Αυτό οδηγεί στο ίδιο συμπέρασμα όπως με τα μηνύματα, ότι τελικά είτε συμβούν μέχρι και n/4 σφάλματα είτε όχι, η πολυπλοκότητα του αλγορίθμου ουσιαστικά δεν επηρεάζεται. Είναι επομένως σημαντικό να γίνεται μια καλή εκτίμηση της τιμής του f την οποία χρησιμοποιεί ο αλγόριθμος.

4.3.2 Περίπτωση με μεταβλητό d+δ

Στο υποκεφάλαιο που ακολουθεί εξετάζονται οι ίδιες ακριβώς περιπτώσεις όπως στο προηγούμενο υποκεφάλαιο με τη διαφορά ότι οι χρονικές καθυστερήσεις d και δ δεν είναι σταθερές αλλά μεταβάλλονται καθ' όλη τη διάρκεια της προσομοίωσης.

4.3.2.1 Αποτελέσματα εφαρμογής με και χωρίς σφάλματα

Η εφαρμογή τώρα εξετάζεται για μη σταθερές καθυστερήσεις, όπως άλλωστε συμβαίνει στα ασύγχρονα συστήματα. Η μέγιστη καθυστέρηση ορίστηκε η τιμή 100 ms ενώ η ελάχιστη 2 ms. Για τις θεωρητικές προσεγγίσεις που αποτέλεσαν το κριτήριο σύγκρισης θεωρήθηκε η χείριστη περίπτωση δηλαδή ορίστηκε καθυστέρηση σταθερή και ίση με 100 ms. Οι μετρήσεις

που ακολουθούν πάρθηκαν από το **μέσο όρο πέντε προσομοιώσεων** του αλγορίθμου για f=1 και f=n/4.

Πιο κάτω φαίνονται οι πίνακες μετρήσεων στις περιπτώσεις που σημειώνεται και που δε σημειώνεται καμιά κατάρρευση κόμβου κατά την προσομοίωση:

	Κόμβοι	**Μηνύματα**		**Χρόνος Ολοκλήρωσης**	
		Χωρίς Σφάλματα	**Με Σφάλματα**	**Χωρίς Σφάλματα**	**Με Σφάλματα**
EARS d+δ=μεταβλητό **f=1**	2	5,20	2,25	5,00	2,50
	4	33,80	28,33	12,60	16,00
	8	110,50	96,00	19,50	22,33
	16	318,00	315,67	24,50	26,67
	32	856,50	817,33	31,00	33,25
	64	2014,00	1986,33	37,60	37,69
	128	5166,00	4658,00	42,00	44,45

Πίνακας 4.5: Μετρήσεις EARS για f=1 και d+δ μεταβλητό (με και χωρίς σφάλματα)

	Κόμβοι	**Μηνύματα**		**Χρόνος Ολοκλήρωσης**	
		Χωρίς Σφάλματα	**Με Σφάλματα**	**Χωρίς Σφάλματα**	**Με Σφάλματα**
EARS d+δ=μεταβλητό **f= n/4**	2	4,67	2,00	5,33	2,75
	4	33,50	29,67	13,00	17,52
	8	120,50	98,33	20,50	25,44
	16	363,50	343,00	28,00	29,98
	32	935,33	906,33	34,00	37,65
	64	2171,67	2159,33	39,67	45,15
	128	5189,00	5166,00	45,00	58,97

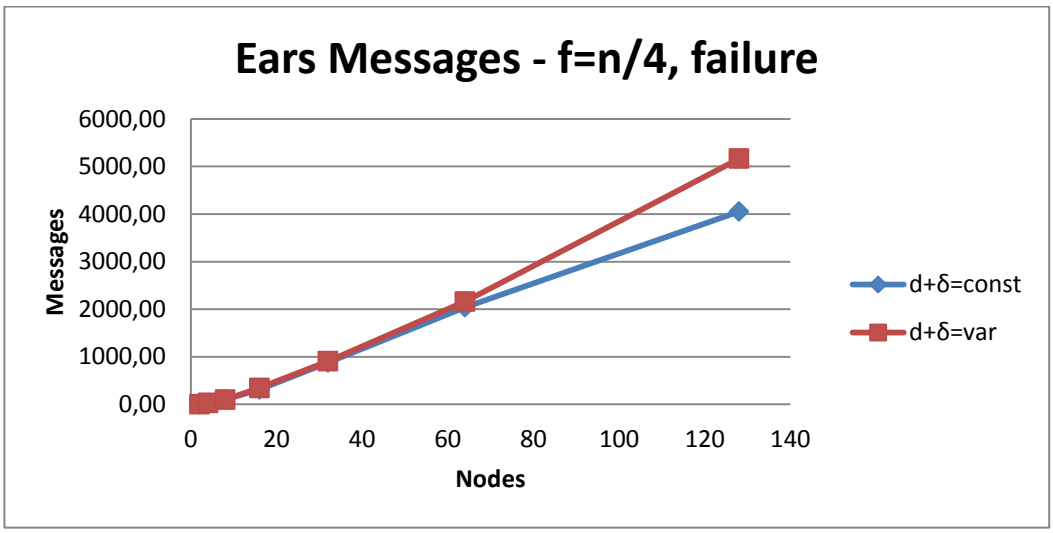
Πίνακας 4.6: Μετρήσεις EARS για f=n/4 και d+δ μεταβλητό (με και χωρίς σφάλματα)

Τόσο ως προς τον αριθμό των μηνυμάτων όσο και ως προς το χρόνο ολοκλήρωσης, από τις μετρήσεις φαίνεται ότι γενικά για f=1 και f=n/4 οι γραφικές παραστάσεις του αριθμού των μηνυμάτων και του χρόνου ολοκλήρωσης σε σχέση με τον αριθμό των κόμβων, είτε

συμβαίνουν σφάλματα είτε όχι, θα έχουν την ίδια συμπεριφορά όπως ακριβώς συμβαίνει στην περίπτωση που το d+δ ήταν σταθερό.

Το πιο ενδιαφέρον εδώ, λοιπόν, είναι να γίνει μια σύγκριση των προηγούμενων αποτελεσμάτων όπου το d+δ ήταν σταθερό και ίσο με 2 ms με τα αντίστοιχα αποτελέσματα όπου το d+δ είναι μεταβλητό. Έτσι θα διερευνηθεί κατά πόσο επηρεάζεται η αποδοτικότητα του αλγόριθμου από τις καθυστερήσεις d+δ. Θα εξεταστεί η περίπτωση όπου f=n/4 (όμοια εξετάζεται και η περίπτωση f=1).

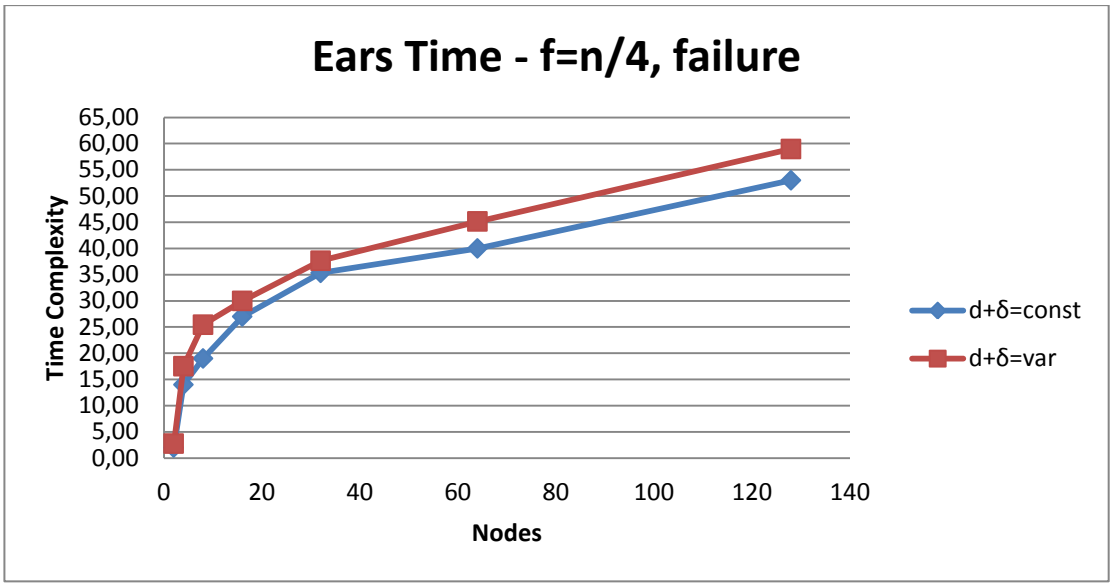
Ως προς τον αριθμό των μηνυμάτων οι παρακάτω γραφικές παραστάσεις δείχνουν συγκριτικά τον αριθμό των μηνυμάτων που αποστέλλονται κατά την προσομοίωση όταν το d+δ είναι σταθερό και όταν είναι μεταβλητό. Θεωρητικά αφού ο συνολικός αριθμός μηνυμάτων που αποστέλλονται κατά την εκτέλεση του αλγορίθμου είναι της τάξεως $O(n \log^3 n (d+\delta))$ όπου n ο αριθμός των κόμβων και d, δ οι καθυστερήσεις, εξάγεται το συμπέρασμα ότι ο αριθμός των μηνυμάτων εξαρτάται από δύο παραμέτρους, τον αριθμό των κόμβων του συστήματος και την καθυστέρηση d+δ. Για την ίδια εκτέλεση του αλγορίθμου με τον ίδιο αριθμό κόμβων ουσιαστικά ο αριθμός μηνυμάτων εξαρτάται μόνο από την καθυστέρηση d+δ. Πιο κάτω παρουσιάζεται η γραφική παράσταση του αριθμού μηνυμάτων σε σχέση με τον αριθμό των κόμβων στην περίπτωση που συμβαίνουν μέχρι f=n/4 σφάλματα (όμοια για f=1 και για την περίπτωση όπου δε συμβαίνουν σφάλματα):



Εικόνα 4.11: Πειραματική σύγκριση EARS με σφάλματα f=n/4, d+δ=const και d+δ=var

Όπως φαίνεται από τα πειραματικά αποτελέσματα ο αριθμός μηνυμάτων στην περίπτωση όπου d+δ είναι μεταβλητό είναι μεγαλύτερος σε σχέση με την περίπτωση όπου d+δ είναι σταθερό. Αυτό ίσως είναι λογικό διότι όταν το d+δ είναι μεταβλητό, είναι πολύ πιθανό να συμβεί μια αλληλουχία γεγονότων σαν αυτή που ακολουθεί: μια διεργασία Α να καθυστερήσει περισσότερο από τις άλλες να στείλει ένα μήνυμα και συνεπώς να καθυστερήσει να στείλει τη φήμη της σε μια διεργασία Β. Η διεργασία Β όμως, που ακόμα δεν γνωρίζει τη φήμη της Α μπορεί εν τω μεταξύ, λόγω μικρότερων καθυστερήσεων, να συνεχίσει να στέλλει και να λαμβάνει μηνύματα με την ελπίδα να μάθει και τη φήμη της Α. Έτσι, λόγω των μεγαλύτερων καθυστερήσεων, η Β "αναγκάζεται" κατά κάποιο τρόπο να στείλει περισσότερα μηνύματα. Στην αντίθετη περίπτωση όπου το d+δ είναι σταθερό, οι διεργασίες είναι μερικώς συγχρονισμένες κι έτσι όλα τα μηνύματα στέλλονται σχεδόν ταυτόχρονα χωρίς την "αναγκαστική" αποστολή επιπλέον μηνυμάτων.

Ως προς χρόνο ολοκλήρωσης οι παρακάτω γραφικές παραστάσεις δείχνουν συγκριτικά το χρόνο που απαιτείται μέχρι να ολοκληρωθεί ο αλγόριθμος, όταν το d+δ είναι σταθερό και όταν είναι μεταβλητό. Θεωρητικά αφού ο χρόνος είναι της τάξεως $O\left(\frac{n}{n-f}\log^2 n(d+\delta)\right)$ όπου n ο αριθμός των κόμβων και d, δ οι καθυστερήσεις, για την ίδια εκτέλεση του αλγορίθμου με τον ίδιο αριθμό κόμβων και f=1 (όμοια για f=n/4 και οποιοδήποτε άλλο f) ουσιαστικά ο χρόνος εξαρτάται μόνο από την καθυστέρηση d+δ. Πιο κάτω παρουσιάζονται οι γραφικές παραστάσεις του χρόνου ολοκλήρωσης σε σχέση με τον αριθμό των κόμβων για f= n/4 στην περίπτωση που στην περίπτωση που συμβαίνουν μέχρι f=n/4 σφάλματα (όμοια για f=1 και για την περίπτωση όπου δε συμβαίνουν σφάλματα):



Εικόνα 4.12: Πειραματική σύγκριση EARS με σφάλματα f=n/4, d+δ=const και d+δ=var

Όπως ήταν αναμενόμενο, στην περίπτωση όπου η καθυστέρηση d+δ είναι σταθερή και ίση με 2 ms ο χρόνος είναι μικρότερος σε σχέση με την περίπτωση όπου η καθυστέρηση d+δ είναι μεταβλητή. Αυτό είναι λογικό και οφείλεται κατά κύριο λόγο στις μη σταθερές καθυστερήσεις. Επειδή αυτές κυμαίνονται από 2 ms μέχρι 100 ms τότε υπάρχει μεγάλη πιθανότητα κατά τη διάρκεια της προσομοίωσης να συμβούν και πιο μεγάλες καθυστερήσεις κι όχι πάντα 2 ms όπως συμβαίνει στην προσομοίωση όπου οι καθυστερήσεις είναι σταθερές. Συνεπώς είναι φυσικό ανάλογα με τις τυχαίες τιμές των καθυστερήσεων να αυξάνεται και ο χρόνος ολοκλήρωσης του αλγορίθμου. Παρόλα αυτά όμως, όπως φαίνεται και από τις γραφικές παραστάσεις η διαφορά στο χρόνο δεν είναι τόσο μεγάλη που να επηρεάσει την ευρωστία του αλγορίθμου.

4.3.4 Συμπεράσματα

Ανακεφαλαιώνοντας μετά την πειραματική αξιολόγηση του αλγορίθμου EARS σε περιβάλλον προσομοίωσης εξάγονται τα ακόλουθα συμπεράσματα:

Καθώς αυξάνεται ο αριθμός των κόμβων στο σύστημα, αυξάνεται και ο συνολικός αριθμός των μηνυμάτων που πρέπει να αποστείλει ο αλγόριθμος για να ολοκληρώσει τη λειτουργία του. Για μεγάλο αριθμό κόμβων μπορεί να πει κανείς ότι η αύξηση είναι σχεδόν γραμμική, το πλήθος των μηνυμάτων είναι της τάξεως $O(n \log^3 n (d+\delta))$ όπου n ο αριθμός των κόμβων και d, δ οι μέγιστες καθυστερήσεις και είναι ανεξάρτητο του αριθμού των σφαλμάτων. Αυτό είναι θετικό διότι σημαίνει πως ακόμα και αν παρουσιαστούν καταρρεύσεις κόμβων δεν επηρεάζεται η πολυπλοκότητα του αλγορίθμου. Επιπλέον, σημαντικό ρόλο στην ευρωστία του αλγορίθμου έχουν οι καθυστερήσεις. Όταν αυτές είναι σταθερές ο αλγόριθμος απαιτεί την αποστολή λιγότερων μηνυμάτων ενώ όταν είναι μεταβλητές απαιτεί περισσότερα μηνύματα.

Καθώς αυξάνεται ο αριθμός των κόμβων αυξάνεται και ο χρόνος που απαιτείται για να ολοκληρώσει ο αλγόριθμος τη λειτουργία του. Ο χρόνος αυτός είναι της τάξεως $O\left(\frac{n}{n-f}\log^2 n(d+\delta)\right)$ όπου n ο αριθμός των κόμβων και d, δ οι μέγιστες καθυστερήσεις. Για μεγάλο αριθμό κόμβων η συμπεριφορά αυτή θυμίζει παρόμοια με λογαριθμική κι αυτό σημαίνει ότι ο αλγόριθμος δε χάνει την ευρωστία του. Επιπρόσθετα, τα πειράματα έδειξαν ότι είτε συμβούν σφάλματα είτε όχι, η πολυπλοκότητα του αλγορίθμου δεν επηρεάζεται σε μεγάλο βαθμό. Ο λόγος είναι η δομή του αλγορίθμου που εξαρτάται από την τιμή του f και όχι από τον πραγματικό αριθμό σφαλμάτων που συμβαίνουν. Τέλος, σημαντικό ρόλο στην ευρωστία του αλγορίθμου έχουν οι καθυστερήσεις. Όταν αυτές είναι σταθερές απαιτούν λιγότερο χρόνο ενώ όταν είναι μεταβλητές απαιτούν περισσότερο χρόνο.

4.4 Πειραματική Αξιολόγηση Αλγορίθμου SEARS

Ακολουθεί η πειραματική αξιολόγηση του αλγορίθμου SEARS στο περιβάλλον προσομοίωσης που προφέρει η βιβλιοθήκη YALPS.

4.4.1 Περίπτωση με σταθερό d+δ

Στο υποκεφάλαιο που ακολουθεί εξετάζεται ο αλγόριθμος SEARS στην περίπτωση όπου οι χρονικές καθυστερήσεις d και δ είναι σταθερές καθ' όλη τη διάρκεια της προσομοίωσης. Οι περιπτώσεις που εξετάστηκαν είναι οι ίδιες με αυτές που εξετάστηκαν στον EARS. Επαναλήφθηκαν τα ίδια πειράματα και πάρθηκαν παρόμοια συμπεράσματα.

4.4.1.1 Αποτελέσματα εφαρμογής χωρίς σφάλματα

Πιο κάτω φαίνονται οι μετρήσεις στην περίπτωση που δεν σημειώνεται καμιά κατάρρευση κόμβου (f=0) κατά την προσομοίωση (πίνακας 4.7).

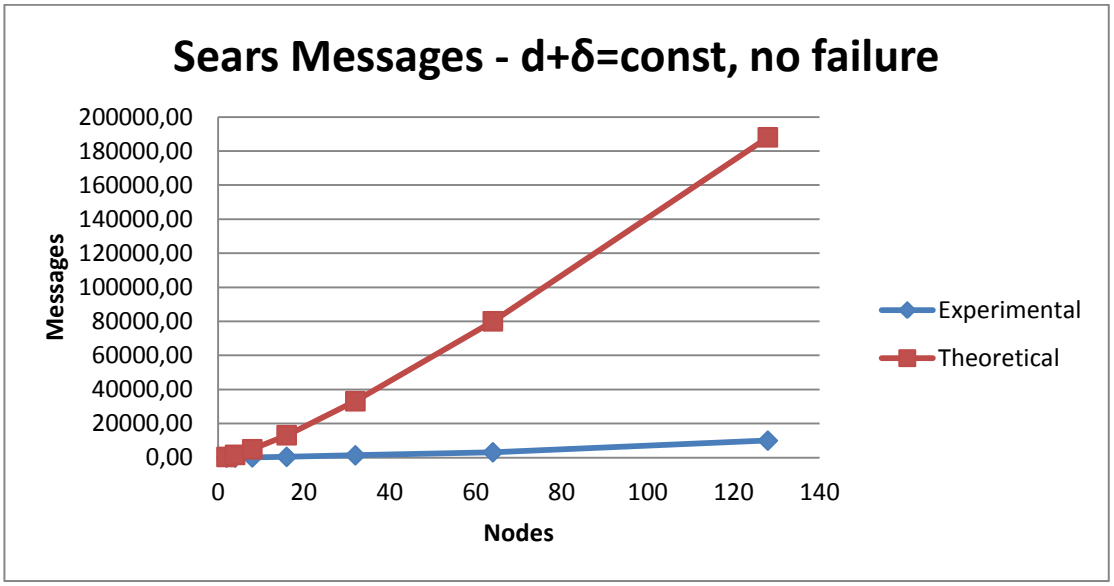
	Κόμβοι	**Μηνύματα**	**Χρόνος Ολοκλήρωσης**
SEARS d+δ=σταθερό χωρίς καταρρεύσεις κόμβων	2	5,00	2,33
	4	31,33	4,67
	8	135,33	5,67
	16	438,33	6,00
	32	1346,33	7,00
	64	3136,67	8,00
	128	9976,00	8,67

Πίνακας 4.7: Μετρήσεις SEARS για d+δ σταθερό (χωρίς σφάλματα)

Ως προς τον αριθμό μηνυμάτων θεωρητικά έχει αποδειχθεί [3] ότι ο συνολικός αριθμός μηνυμάτων που αποστέλλονται κατά την εκτέλεση του αλγορίθμου είναι της τάξεως $O\left(\frac{n^{2+\varepsilon}}{\varepsilon(n-f)}\log n\,(d+\delta)\right)$ όπου n ο αριθμός των κόμβων, ε<1 μια οποιαδήποτε σταθερά, d, δ οι μέγιστες καθυστερήσεις και f ο αριθμός των σφαλμάτων. Επειδή στο συγκεκριμένο πείραμα εξετάζεται η περίπτωση όπου δεν συμβαίνει καμιά κατάρρευση (f=0), η θεωρητική προσέγγιση για τον αριθμό των μηνυμάτων που αποστέλλονται παίρνει τη μορφή

$O\left(\frac{n^{1+\varepsilon}}{\varepsilon}\log n\,(d+\delta)\right)$. Υπενθυμίζεται ότι για τη σταθερά ε<1 αφού δοκιμάστηκαν αρκετές τιμές, τελικά στα πειράματα επιλέχθηκε το $\varepsilon = \frac{1}{100}$ (βλ. λεπτομέρειες στο Υποκεφάλαιο 4.2).

Παρακάτω φαίνεται η γραφική παράσταση που προκύπτει από τη θεωρητική ανάλυση του αλγορίθμου σε σύγκριση με τη γραφική παράσταση του αριθμού των μηνυμάτων συναρτήσει του αριθμού των κόμβων του συστήματος για την περίπτωση όπου δε σημειώνεται καμιά αποτυχία (κατάρρευση) καθ' όλη τη διάρκεια της προσομοίωσης:



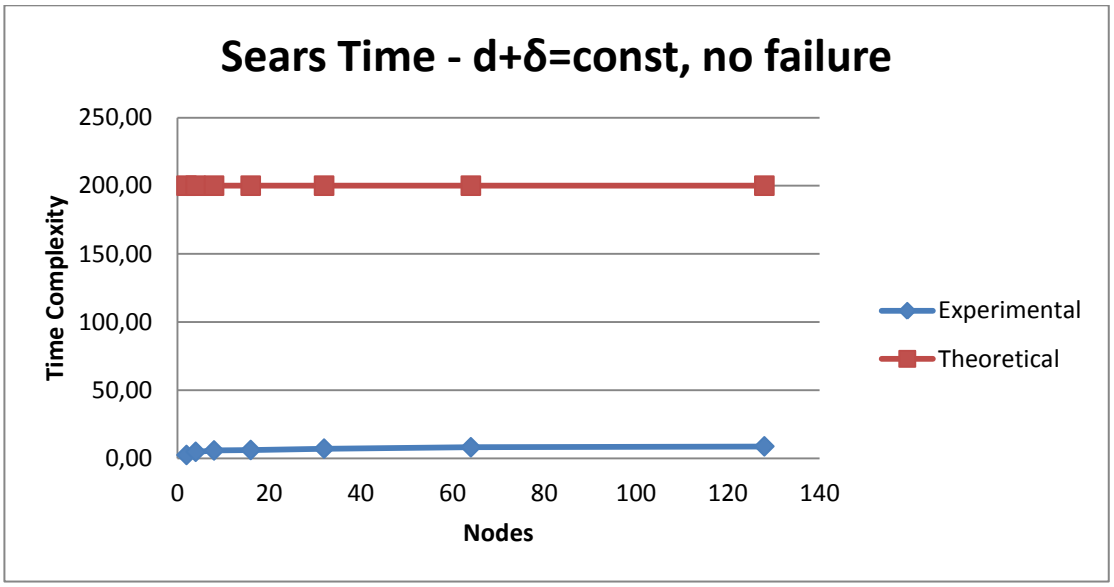
Εικόνα 4.13: Πειραματική και Θεωρητική προσέγγιση SEARS χωρίς αποτυχίες

Από τα πειραματικά αποτελέσματα φαίνεται ότι ως προς τα μηνύματα ο αλγόριθμος SEARS έχει όμοια συμπεριφορά με τον αλγόριθμο EARS, δηλαδή ο αριθμός των μηνυμάτων αυξάνεται καθώς αυξάνεται ο αριθμός των κόμβων για τους ίδιους λόγους που αναφέρθηκαν προηγουμένως (βλ. επεξήγηση αποτελεσμάτων EARS στο Υποκεφάλαιο 4.3). Συγκρίνοντας τα πειραματικά αποτελέσματα με τα θεωρητικά επιβεβαιώνεται και η ορθότητα του SEARS και επαληθεύεται το θεωρητικό άνω φράγμα για το συνολικό αριθμό μηνυμάτων που χρειάζεται να σταλούν κατά την εκτέλεση του αλγορίθμου.

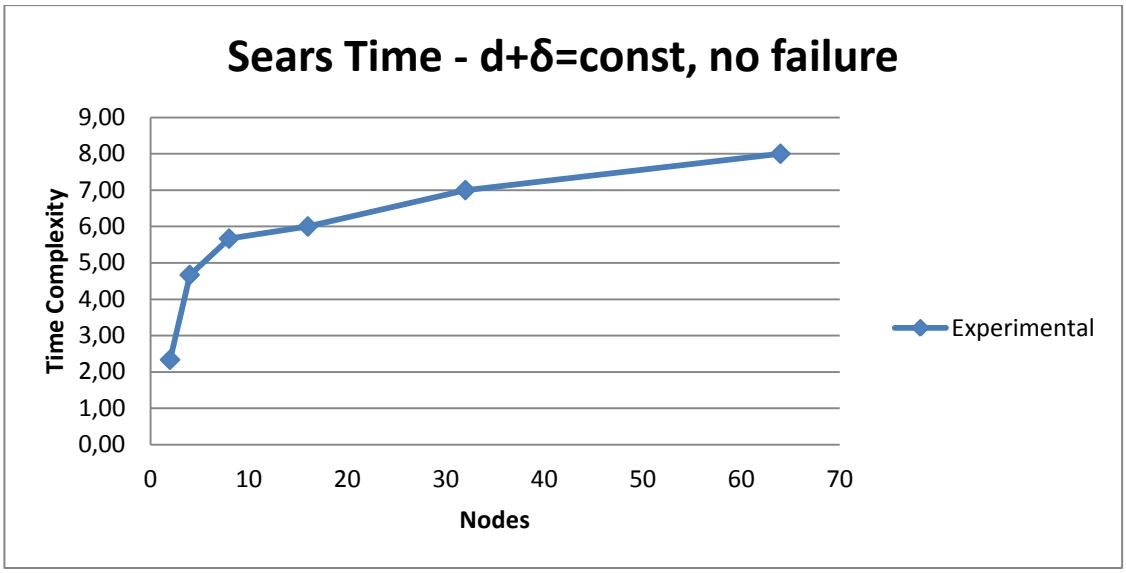
Ως προς το χρόνο ολοκλήρωσης του αλγορίθμου, θεωρητικά έχει αποδειχθεί [3] ότι κάθε κόμβος που δεν έχει καταρρεύσει κατά την εκτέλεση ολοκληρώνει τη λειτουργία του μέσα σε

χρόνο τάξεως $O\left(\frac{n}{\varepsilon(n-f)}(d+\delta)\right)$ όπου n ο αριθμός των κόμβων και d, δ οι καθυστερήσεις. Επειδή στο συγκεκριμένο πείραμα εξετάζεται η περίπτωση όπου δεν συμβαίνει καμιά κατάρρευση (f=0), η θεωρητική προσέγγιση για το χρόνο ολοκλήρωσης του αλγορίθμου παίρνει τη μορφή $O\left(\frac{1}{\varepsilon}(d+\delta)\right)$. Το γεγονός αυτό οδηγεί στο συμπέρασμα ότι ο χρόνος ολοκλήρωσης είναι ανεξάρτητος από τον αριθμό των κόμβων και για σταθερές καθυστερήσεις d και δ παραμένει σταθερός. Αυτό είναι αναμενόμενο αφού από την κατασκευή του ο αλγόριθμος αποστέλλει περισσότερα μηνύματα σε κάθε βήμα, επομένως είναι λογικό στο ίδιο βήμα να ενημερώνονται περισσότεροι κόμβοι. Άμεση συνέπεια είναι ότι το πρόβλημα λύνεται σε όσο το δυνατό λιγότερο χρόνο και μάλιστα για μεγάλο αριθμό κόμβων (>10) παρατηρείται σχεδόν σταθερός χρόνος.

Παρακάτω (Εικόνα 4.14) φαίνεται η γραφική παράσταση του χρόνου ολοκλήρωσης συναρτήσει του αριθμού των κόμβων του συστήματος σε σύγκριση με την αντίστοιχη γραφική παράσταση που προκύπτει από τη θεωρητική ανάλυση του αλγορίθμου για την περίπτωση όπου δε σημειώνεται καμιά αποτυχία (κατάρρευση) καθ' όλη τη διάρκεια της προσομοίωσης. Για να φανεί καλύτερα η μορφή της γραφικής παράστασης δίνεται από κάτω ξανά το ίδιο γράφημα χωρίς το αντίστοιχο θεωρητικό (Εικόνα 4.15).



Εικόνα 4.14: Πειραματική και Θεωρητική προσέγγιση SEARS χωρίς αποτυχίες



Εικόνα 4.15: Πειραματική προσέγγιση SEARS για d+δ = σταθερό, χωρίς αποτυχίες

Από τα πειραματικά αποτελέσματα φαίνεται ότι πράγματι ο χρόνος δεν επηρεάζεται πολύ από τον αριθμό των κόμβων. Μπορεί στην αρχή που ο αριθμός των κόμβων είναι μικρός ο χρόνος να είναι μικρότερος και να αυξάνεται με μεγαλύτερο ρυθμό καθώς αυξάνονται οι κόμβοι στο σύστημα, όμως καθώς οι κόμβοι ξεπερνούν τους 10 περίπου, μειώνεται ο ρυθμός αύξησης του χρόνου και φαίνεται να συγκλίνει σε ένα σταθερό αριθμό. Συγκρίνοντας τα πειραματικά αποτελέσματα με τα θεωρητικά επιβεβαιώνεται και ως προς το χρόνο η ορθότητα του αλγορίθμου και επαληθεύεται το θεωρητικό άνω φράγμα.

4.4.1.2 Αποτελέσματα εφαρμογής με σφάλματα

Ενώ προηγουμένως μελετήθηκε η περίπτωση όπου δεν παρουσιάζεται κανένα σφάλμα κατάρρευσης κόμβου, στη συνέχεια μελετήθηκε η περίπτωση εμφάνισης μέχρι f σφαλμάτων κατάρρευσης με κάποια πιθανότητα. Ακολουθούν πίνακες μετρήσεων για f=1 και f=n/4 με πιθανότητα κατάρρευσης ίση με $\frac{f}{nx}$ (βλ. επεξήγηση για την πιθανότητα στο Κεφάλαιο 4).

	Κόμβοι	Μηνύματα	Χρόνος Ολοκλήρωσης
SEARS **d+δ**=σταθερό **f=1** κατάρρευση κόμβου	2	1,67	1,67
	4	23,67	5,00
	8	112,00	6,33
	16	379,33	7,33
	32	1108,67	8,00
	64	2907,67	8,67
	128	9457,67	8,33

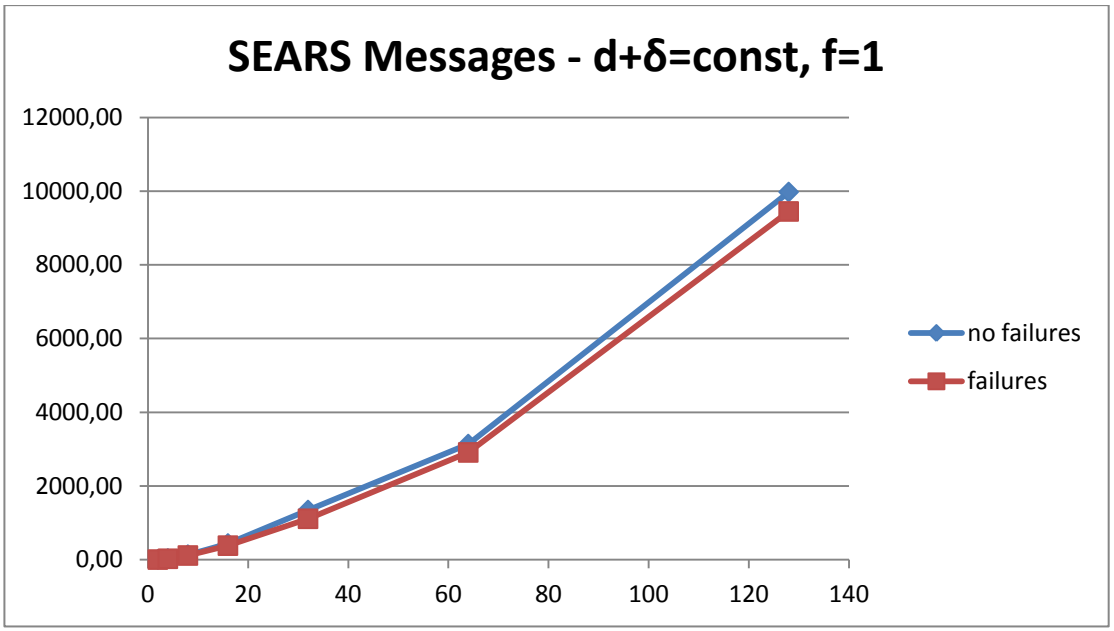
Πίνακας 4.8: Μετρήσεις SEARS για f=1 και d+δ σταθερό (με κατάρρευση κόμβου)

	Κόμβοι	Μηνύματα	Χρόνος Ολοκλήρωσης
SEARS **d+δ**=σταθερό **f=n/4** καταρρεύσεις κόμβων	2	2,00	1,67
	4	24,60	5,60
	8	116,40	6,60
	16	388,00	7,20
	32	1109,80	8,00
	64	2929,67	8,67
	128	10280,00	9,00

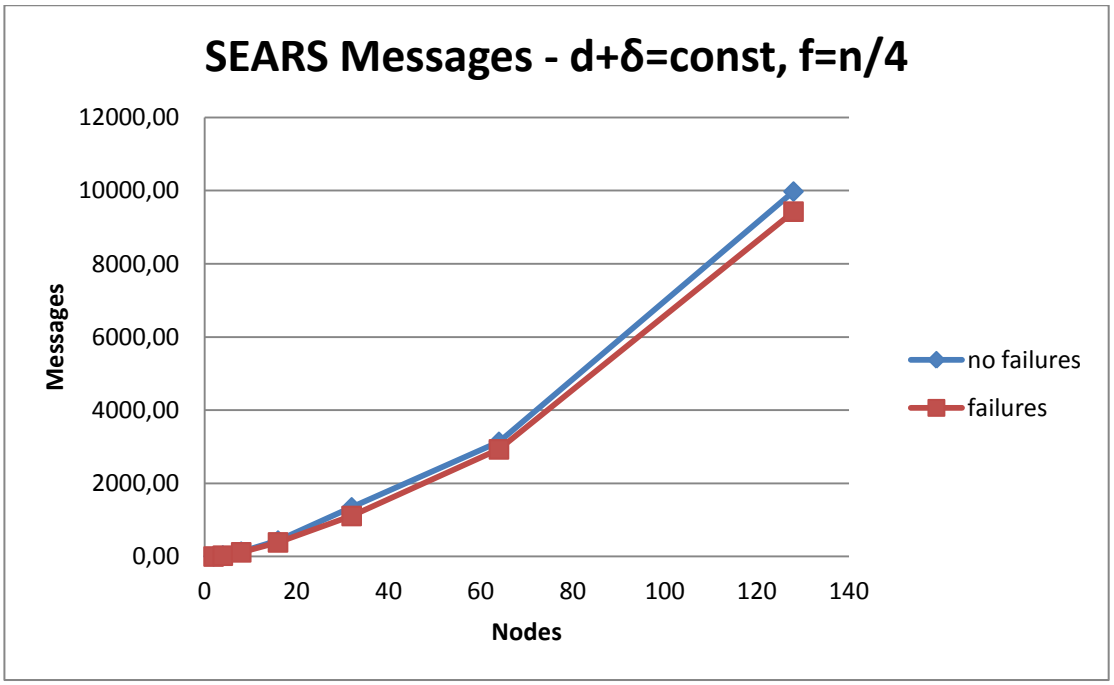
Πίνακας 4.9: Μετρήσεις SEARS για f=n/4 και d+δ σταθερό (με κατάρρευση κόμβου)

Ως προς τον αριθμό μηνυμάτων τα αποτελέσματα είναι εντελώς παρόμοια με τα αντίστοιχα χωρίς σφάλματα. Θεωρητικά ο αριθμός των μηνυμάτων καθώς επίσης και ο χρόνος εξαρτώνται από τη σταθερά ε<1, τον αριθμό των κόμβων και τις καθυστερήσεις. Επειδή τόσο για την περίπτωση όπου f=1 όσο και για την περίπτωση όπου f=n/4 η σταθερά ε<1 είναι η ίδια και οι καθυστερήσεις θεωρούνται σταθερές, για τον ίδιο αριθμό κόμβων οι δύο περιπτώσεις διαφέρουν ουσιαστικά μόνο στην τιμή του f.

Παρακάτω φαίνονται οι γραφικές παραστάσεις του αριθμού των μηνυμάτων που αποστέλλει ο αλγόριθμος συναρτήσει του αριθμού των κόμβων του συστήματος για f=1 και f=n/4 σε σύγκριση με την αντίστοιχη χωρίς σφάλματα (f=0):



Εικόνα 4.16: Πειραματική σύγκριση SEARS για f=1 με και χωρίς αποτυχίες



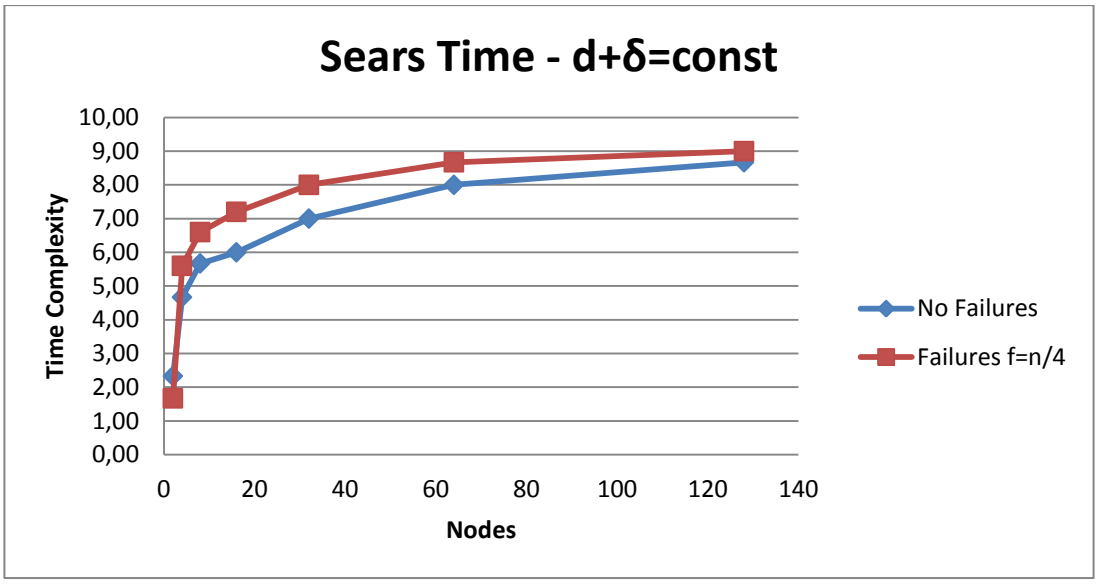
Εικόνα 4.17: Πειραματική σύγκριση SEARS για f=n/4 με και χωρίς αποτυχίες

Παρατηρείται ότι για f=1 ή για f=n/4 ή για οποιοδήποτε άλλο f τα αποτελέσματα είναι πολύ κοντινά. Οι μόνες διαφορές που υπάρχουν πρέπει να οφείλονται στην τιμή της πιθανότητας κατάρρευσης κόμβου η οποία εξαρτάται από την τιμή του f καθώς επίσης και

από τον αριθμό των βημάτων που εκτελεί ο αλγόριθμος κατά την εκτέλεση. Στην περίπτωση αυτή η πιθανότητα να καταρρεύσουν f=n/4 κόμβοι κατά τη διάρκεια της προσομοίωσης αυξάνεται σε σύγκριση με την προηγούμενη περίπτωση όπου f=1. Επομένως αναμένεται μεγαλύτερη αύξηση στον αριθμό των βημάτων που εκτελούνται και μεγαλύτερη μείωση στον αριθμό των μηνυμάτων καθ' ότι θα σταματήσουν την αποστολή μηνυμάτων περισσότεροι κόμβοι και θα προκληθεί έτσι μεγαλύτερη καθυστέρηση στην ενημέρωση των υπόλοιπων κόμβων του συστήματος.

Ως προς το χρόνο ολοκλήρωσης του αλγορίθμου, αναμένεται ότι για σταθερές καθυστερήσεις, όσο μεγαλώνει ο αριθμός των κόμβων στο σύστημα ο αριθμός των βημάτων δεν εξαρτάται τόσο από το πλήθος των κόμβων και για αρκετά μεγάλο αριθμό κόμβων τείνει να παραμένει σταθερός. Αυτό είναι λογικό αφού από την κατασκευή του ο αλγόριθμος αποστέλλει περισσότερα μηνύματα σε κάθε βήμα και άρα στο ίδιο βήμα ενημερώνονται περισσότεροι κόμβοι. Κατά συνέπεια, το πρόβλημα λύνεται σε όσο το δυνατό λιγότερα βήματα και μάλιστα για μεγάλο αριθμό κόμβων (>10) εκτιμάται ότι θα παρατηρείται σχεδόν σταθερός αριθμός βημάτων.

Παρακάτω φαίνεται η γραφική παράσταση του χρόνου ολοκλήρωσης του αλγορίθμου συναρτήσει του αριθμού των κόμβων του συστήματος για την περίπτωση όπου συμβαίνουν μέχρι f=n/4 καταρρεύσεις κόμβων (όμοια για f=1) σε σύγκριση με την περίπτωση όπου δεν συμβαίνει καμία κατάρρευση:



Εικόνα 4.18: Πειραματική σύγκριση SEARS για f=n/4 με και χωρίς αποτυχίες

Συγκρίνοντας τα πειραματικά αποτελέσματα με και χωρίς σφάλματα κατάρρευσης, παρατηρείται ότι στην περίπτωση των σφαλμάτων κατάρρευσης ο χρόνος ολοκλήρωσης είναι μεγαλύτερος από αυτόν στην περίπτωση όπου δεν υπάρχουν σφάλματα. Ο λόγος είναι διότι τυχόν αποτυχία ενός κόμβου οδηγεί στη μείωση του αριθμού των μηνυμάτων που ανταλλάζονται μεταξύ των κόμβων και συνεπώς στην καθυστέρηση στην ενημέρωσή τους. Επιπλέον, όταν καταρρεύσει ένας κόμβος, ναι μεν σταματά να αποστέλλει μηνύματα αλλά οι υπόλοιποι κόμβοι που δεν έχουν καταρρεύσει δεν γνωρίζουν ότι οι γείτονές τους έχουν καταρρεύσει. Αυτό έχει σαν αποτέλεσμα να στέλλονται μηνύματα και σ' αυτούς τους κόμβους καθυστερώντας την ενημέρωση των κόμβων που πραγματικά δεν έχουν καταρρεύσει.

4.4.2 Περίπτωση όπου d+δ = μεταβλητό

Στο υποκεφάλαιο που ακολουθεί εξετάζονται οι ίδιες ακριβώς περιπτώσεις όπως στο προηγούμενο υποκεφάλαιο με τη διαφορά ότι οι χρονικές καθυστερήσεις d και δ δεν είναι σταθερές αλλά μεταβάλλονται καθ' όλη τη διάρκεια της προσομοίωσης.

4.4.2.1 Αποτελέσματα εφαρμογής με και χωρίς σφάλματα

Η εφαρμογή τώρα εξετάζεται για μη σταθερές καθυστερήσεις, όπως ακριβώς εξετάστηκε στον EARS (βλ. Υποκεφάλαιο 4.3). Τόσο ως προς τον αριθμό των μηνυμάτων όσο και ως προς το χρόνο ολοκλήρωσης, από τις μετρήσεις φαίνεται ότι γενικά για f=1 και f=n/4 οι γραφικές παραστάσεις του αριθμού των μηνυμάτων και του χρόνου ολοκλήρωσης σε σχέση με τον αριθμό των κόμβων, είτε συμβαίνουν σφάλματα είτε όχι, θα έχουν την ίδια συμπεριφορά όπως ακριβώς συμβαίνει στην περίπτωση που το d+δ ήταν σταθερό.

Όπως ακριβώς με τον EARS έτσι και με τον SEARS αρκεί να γίνει μια σύγκριση των προηγούμενων αποτελεσμάτων όπου το d+δ ήταν σταθερό και ίσο με 2 ms με τα αντίστοιχα αποτελέσματα όπου το d+δ είναι μεταβλητό και κυμαίνεται από 2 έως και 100 ms. Θα εξεταστεί η περίπτωση όπου f=n/4 (όμοια εξετάζεται και η περίπτωση f=1).

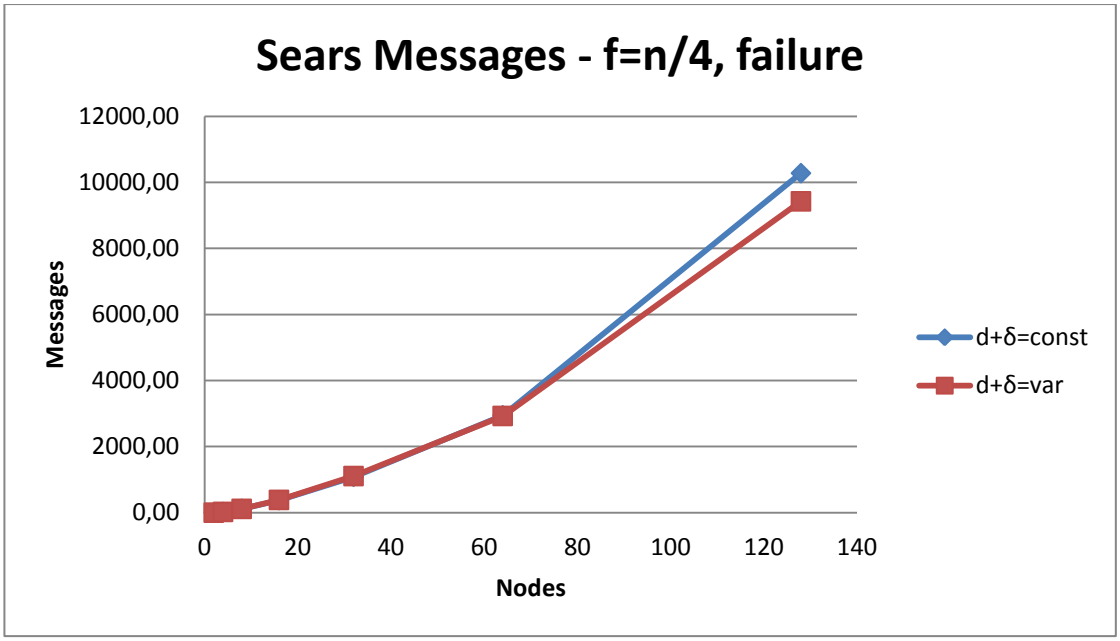
Πιο κάτω φαίνονται οι πίνακες μετρήσεων για f=n/4 (ανάλογες είναι και οι μετρήσεις για f=1) στις περιπτώσεις που σημειώνεται και που δε σημειώνεται καμιά κατάρρευση κόμβου κατά την προσομοίωση:

	Κόμβοι	**Μηνύματα**		**Χρόνος Ολοκλήρωσης**	
		Χωρίς Σφάλματα	**Με Σφάλματα**	**Χωρίς Σφάλματα**	**Με Σφάλματα**
SEARS d+δ=μεταβλητό **f= n/4**	2	2,80	2,00	2,00	1,40
	4	27,20	24,60	5,20	5,80
	8	117,60	116,40	7,00	6,60
	16	390,60	388,00	7,40	7,50
	32	1109,00	1109,80	8,00	8,00
	64	2910,80	2929,67	8,20	9,00
	128	10343,00	9426,67	7,50	9,50

Πίνακας 4.10: Μετρήσεις SEARS για d+δ μεταβλητό (με και χωρίς σφάλματα)

Ως προς τον αριθμό των μηνυμάτων όπως αναφέρθηκε και προηγουμένως ο αριθμός μηνυμάτων εξαρτάται μόνο από την καθυστέρηση d+δ. Η γραφική παράσταση που ακολουθεί

απεικονίζει τον αριθμό των μηνυμάτων σε σχέση με τον αριθμό των κόμβων στην περίπτωση που συμβαίνουν μέχρι f=n/4 σφάλματα (όμοια για f=1 και για την περίπτωση όπου δε συμβαίνουν σφάλματα):

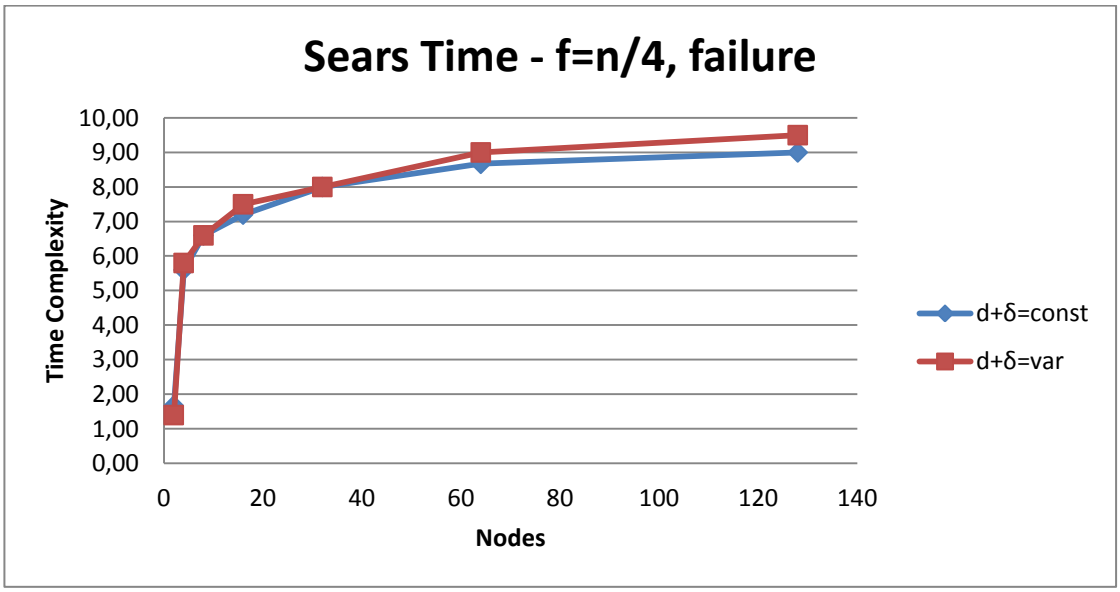


Εικόνα 4.19: Πειραματική σύγκριση SEARS με σφάλματα f=n/4, d+δ=const, d+δ=var

Όπως φαίνεται από τα πειραματικά αποτελέσματα, σε αντίθεση με τον EARS ο αριθμός μηνυμάτων στην περίπτωση όπου d+δ είναι μεταβλητό είναι λίγο μικρότερος σε σχέση με την περίπτωση όπου d+δ είναι σταθερό. Οι διαφορές αυτές οφείλονται ακριβώς στο είδος των καθυστερήσεων. Επειδή στη μια περίπτωση οι καθυστερήσεις ποικίλουν και είναι τυχαίες και άγνωστες είναι πιθανό να παρουσιαστούν και μεγαλύτερες καθυστερήσεις (κοντά στα 100ms) οπότε να καθυστερεί αρκετά ένας κόμβος να αποστείλει ένα μήνυμα. Στην αντίστοιχη περίπτωση όμως όπου οι καθυστερήσεις είναι σταθερές, ο ίδιος κόμβος ναι μεν θα καθυστερήσει να αποστείλει το μήνυμα αλλά ο χρόνος αυτός θα είναι μικρός και σταθερός στα 2ms. Επομένως, θα αποστείλει το μήνυμα πολύ πιο γρήγορα και θα είναι έτοιμος να αποστείλει το επόμενο και δεν θα υπάρχει περίπτωση να καθυστερεί περισσότερο.

Ως προς το χρόνο ολοκλήρωσης πάλι ισχύουν τα ίδια συμπεράσματα με τον EARS. Πιο κάτω παρουσιάζεται η γραφική παράσταση του χρόνου ολοκλήρωσης σε σχέση με τον αριθμό

των κόμβων στην περίπτωση που συμβαίνουν μέχρι f=n/4 σφάλματα (όμοια για f=1 και για την περίπτωση όπου δε συμβαίνουν σφάλματα):



Εικόνα 4.20: Πειραματική σύγκριση SEARS f=n/4 σφάλματα, d+δ=const, d+δ=var

Όπως ήταν αναμενόμενο, στην περίπτωση όπου η καθυστέρηση d+δ είναι σταθερή και ίση με 2 ms ο χρόνος είναι μικρότερος σε σχέση με την περίπτωση όπου η καθυστέρηση d+δ είναι μεταβλητή για τους ίδιους λόγους που συνέβαινε αυτό και στον EARS.

4.4.4 Συμπεράσματα

Ανακεφαλαιώνοντας μετά την πειραματική αξιολόγηση του αλγορίθμου SEARS σε περιβάλλον προσομοίωσης εξάγονται τα ακόλουθα συμπεράσματα:

Καθώς αυξάνεται ο αριθμός των κόμβων στο σύστημα, αυξάνεται και ο συνολικός αριθμός των μηνυμάτων που πρέπει να αποστείλει ο αλγόριθμος για να ολοκληρώσει τη λειτουργία του. Η αύξηση είναι σχεδόν γραμμική, το πλήθος των μηνυμάτων είναι της τάξεως $O\left(\frac{n^{2+\varepsilon}}{\varepsilon(n-f)} \log n\,(d+\delta)\right)$ όπου $\varepsilon < 1$, $f < \frac{n}{2}$, n ο συνολικός αριθμός των κόμβων του δικτύου, f ο συνολικός αριθμός των σφαλμάτων που συμβαίνουν με κάποια πιθανότητα κατά την

εκτέλεση και d, δ οι μέγιστες χρονικές καθυστερήσεις. Αυτό είναι θετικό διότι σημαίνει πως ακόμα και αν παρουσιαστούν καταρρεύσεις κόμβων ο αλγόριθμος δεν θα χάσει την ευρωστία του.

Καθώς αυξάνεται ο αριθμός των κόμβων αυξάνεται και ο χρόνος που απαιτείται για να ολοκληρώσει ο αλγόριθμος τη λειτουργία του. Ο χρόνος αυτός είναι της τάξεως $O\left(\frac{n}{\varepsilon(n-f)}(d+\delta)\right)$ όπου ε < 1, n ο συνολικός αριθμός των κόμβων του δικτύου, f ο συνολικός αριθμός των σφαλμάτων που πιθανό να συμβούν κατά την εκτέλεση και d, δ οι μέγιστες χρονικές καθυστερήσεις. Αυτό είναι θετικό διότι σημαίνει πως ο αλγόριθμος δε χάνει την ευρωστία του. Ενώ αρχικά σημειώνεται μια αύξηση στο χρόνο ολοκλήρωσης του αλγορίθμου, στη συνέχεια καθώς αυξάνονται οι κόμβοι στο σύστημα, ο ρυθμός αύξησης μειώνεται και για μεγάλο αριθμό κόμβων τείνει να παραμένει σταθερός. Έτσι ακόμα και για αρκετά μεγάλο αριθμό κόμβων ο χρόνος ολοκλήρωσης δεν είναι πολύ μεγάλος και το γεγονός αυτό κάνει τον αλγόριθμο πιο αποδοτικό. Αξίζει να σημειωθεί πως ο αριθμός των γύρων επικοινωνίας επηρεάζεται σε κάποιο βαθμό και από τις καθυστερήσεις d+δ είτε αυτές μεταβάλλονται κατά τη διάρκεια της προσομοίωσης είτε παραμένουν σταθερές. Όμως αυτές οι καθυστερήσεις δεν είναι πολύ μεγάλες κι έτσι δεν μπορούν να βλάψουν την ευρωστία του αλγορίθμου.

4.5 Γενικά Συμπεράσματα

Στο κεφάλαιο αυτό αξιολογήθηκε η ευρωστία των αλγορίθμων EARS και SEARS σε περιβάλλον προσομοίωσης με χρήση της πλατφόρμας YALPS. Αρχικά οι αλγόριθμοι αξιολογήθηκαν κάτω από την υπόθεση σταθερών καθυστερήσεων στην αποστολή μηνυμάτων σε μια προσπάθεια προσομοίωσης του σύγχρονου περιβάλλοντος και στη συνέχεια αξιολογήθηκαν κάτω από την υπόθεση μη σταθερών καθυστερήσεων σε μια προσπάθεια προσομοίωσης του ασύγχρονου περιβάλλοντος. Και στις δύο εκδοχές εξετάστηκε πρώτα η

περίπτωση όπου το περιβάλλον δεν υπόκειται σε σφάλματα και ακολούθως η περίπτωση όπου το περιβάλλον υπόκειται σε σφάλματα με κάποια πιθανότητα κατάρρευσης κόμβων.

Το γενικό συμπέρασμα που εξάγεται από τα πειραματικά αποτελέσματα είναι ότι τόσο ο αλγόριθμος EARS όσο και ο αλγόριθμος SEARS είναι αποδοτικοί αφού ενημερώνουν τους κόμβους του συστήματος σε χρόνο τάξεως $O\left(\frac{n}{n-f}\log^2 n(d+\delta)\right)$ και $O\left(\frac{n}{\varepsilon(n-f)}(d+\delta)\right)$ αντίστοιχα αποστέλλοντας πλήθος μηνυμάτων της τάξεως $O(n\ \log^3 n\ (d+\delta))$ και $O\left(\frac{n^{2+\varepsilon}}{\varepsilon(n-f)}\log n\,(d+\delta)\right)$ επίσης αντίστοιχα.

Το πιο σημαντικό συμπέρασμα όμως είναι ότι και οι δύο αλγόριθμοι είναι ταυτόχρονα ανεκτικοί σε σφάλματα κατάρρευσης κόμβων. Ακόμη και στην παρουσία σφαλμάτων οι αλγόριθμοι εξακολουθούν να ενημερώνουν ικανοποιητικά τους κόμβους του συστήματος και μάλιστα χωρίς να επηρεάζεται σημαντικά ούτε ο αριθμός των μηνυμάτων που αποστέλλονται, ούτε ο αριθμός των γύρων επικοινωνίας που απαιτούνται ούτε ο χρόνος εκτέλεσής τους.

Τελειώνοντας, και ο αλγόριθμος EARS και ο αλγόριθμος SEARS επιβεβαιώνουν στην πράξη τα θεωρητικά συμπεράσματα και έτσι μπορεί άνετα να πει κάποιος πως πρόκειται για δύο εύρωστους ασύγχρονους αλγόριθμους οι οποίοι επιλύουν με επιτυχία το πρόβλημα.

Κεφάλαιο 5

Εφαρμογή Αλγορίθμων σε πραγματικό περιβάλλον

5.1 Το Δικτυακό Σύστημα PlanetLab

Το PlanetLab[10] είναι μια ανοικτή πλατφόρμα για την ανάπτυξη και τη χρήση υπηρεσιών δικτύου παγκόσμιας κλίμακας. Είναι ένα γεωγραφικά κατανεμημένο δίκτυο που ξεκίνησε αρχικά σαν μια ερευνητική κοινότητα με συνδέσεις μεταξύ διαφόρων πανεπιστημίων υπό την εποπτεία του Πανεπιστημίου Princeton. Είναι σχεδιασμένο έτσι ώστε να επιτρέπει στους επιστήμονες – απ' όπου κι αν βρίσκονται – να δημιουργούν και να δοκιμάζουν νέου τύπου λογισμικά τα οποία δεν περιορίζονται σε έναν και μοναδικό υπολογιστή αλλά «τρέχουν» σε πολλούς υπολογιστές ταυτόχρονα χρησιμοποιώντας το παγκόσμιο δίκτυο σαν ένα μεγάλο υπολογιστή. Το Planet Lab, δηλαδή, δημιουργεί ένα μοναδικό περιβάλλον το οποίο επιτρέπει την πραγματοποίηση πειραμάτων σε παγκόσμια κλίμακα.

Αυτή τη στιγμή το PlanetLab αποτελείται από 1077 κόμβους σε 531 τοποθεσίες. Συνεργάζεται με το Παρατηρητήριο Abilene, ένα πρόγραμμα που υποστηρίζει την συλλογή πληροφοριών που αφορά την κίνηση δεδομένων στο Διαδίκτυο και που σχετίζεται με το δίκτυο Internet2. Με την συνεργασία αυτή το PlanetLab έχει πρόσβαση σε στοιχεία που αφορούν στατιστικά στοιχεία για την κυκλοφορία στους συνδέσμους του δικτύου, τη ροή πληροφοριών μέσα στο δίκτυο, τη δρομολόγηση δεδομένων, δηλαδή σε ποιο σημείο οδηγούνται τα πακέτα μέσα στο δίκτυο, τους δρομολογητές, την καθυστέρηση των

δεδομένων, δηλαδή πόσο χρόνο χρειάζονται τα πακέτα για να φτάσουν στον προορισμό τους και το ρυθμό μετάδοσης των δεδομένων.

Στο φιλόδοξο αυτό πρόγραμμα λαμβάνουν μέρος τα μεγαλύτερα ινστιτούτα (εκπαιδευτικά κυρίως) ενώ στους χορηγούς του συμπεριλαμβάνονται δύο από τις μεγαλύτερες εταιρείες στον χώρο της τεχνολογίας, η Intel και η HP. Αυτή τη στιγμή υπάρχουν εκατοντάδες εφαρμογές που τρέχουν στο PlanetLab οι οποίες δημιουργήθηκαν από διάφορα πανεπιστήμια και άλλους οργανισμούς. Υπάρχουν όμως και απλές εφαρμογές που γίνονται από φοιτητές και μικρές ομάδες ερευνητών, οι οποίες μπορούν να χρησιμοποιηθούν από οποιονδήποτε, για ερευνητικούς και μη κερδοσκοπικούς σκοπούς.

Όραμα των δημιουργών του PlanetLab, οι οποίοι το αποκαλούν ως την «Νέα Γενεά του Διαδικτύου», είναι να πετύχουν μια αναβάθμιση του συστήματος, η οποία θα είναι ικανή να αντικαταστήσει ολόκληρο το διαδίκτυο.

Το PlanetLab είναι το καταλληλότερο περιβάλλον για την εφαρμογή των αλγορίθμων EARS και SEARS. Πρόκειται για ένα δικτυακό σύστημα που βρίσκεται στο Διαδίκτυο, είναι από τη φύση του ασύγχρονο και ανά πάσα στιγμή μπορούν να συμβούν σφάλματα στους κόμβους και στις συνδέσεις. Είναι δηλαδή ιδανικό περιβάλλον για την πρακτική αξιολόγηση των αλγορίθμων EARS και SEARS.

5.2 Σύντομος Οδηγός Χρήσης του Planet Lab

Κάθε οργανισμός (site) που συμμετέχει στο PlanetLab έχει στη διάθεσή του τουλάχιστον δύο μηχανές οι οποίες μπορούν να χρησιμοποιηθούν από χρήστες που ανήκουν σε άλλα site. Για να δικαιούται κάποιος να χρησιμοποιήσει το PlanetLab πρέπει πρώτα να γίνει μέλος του κάνοντας αίτηση εγγραφής σε κάποιο site. Αφού η αίτησή του εγκριθεί από τους PIs (Principal Investigators) του site, οι οποίοι είναι υπεύθυνοι για τη διαχείριση των νέων μελών του site και των διαθέσιμων πόρων (slices), πλέον ο αιτητής γίνεται επίσημο μέλος με

προσωπικό λογαριασμό (account), αποκτά πρόσβαση στις μηχανές του Planet Lab και μπορεί να τρέξει ελεύθερα τις εφαρμογές του.

Οι πόροι του PlanetLab είναι οργανωμένοι σε μερίσματα (slices). Κάθε οργανισμός μπορεί να έχει μέχρι και 10 μερίσματα και κάθε μέρισμα έχει δικαίωμα χρήσης μέχρι και σε 32 μηχανές. Ένα μέρισμα, δηλαδή, είναι μια συλλογή από εικονικές μηχανές διαθέσιμες για μια υπηρεσία. Δεσμεύει ένα μέρος των πόρων (εύρος ζώνης δικτύου, μνήμη, χώρος δίσκου) το οποίο περιορίζεται ανάλογα με την κατάσταση δικτύου για κάθε χρήστη. Ο κάθε χρήστης μπορεί να δουλεύει προσωπικά και ανενόχλητα σε κάθε μηχανή μέσω του μερίσματός του χωρίς να επηρεάζει ή να επηρεάζεται από τους άλλους χρήστες που χρησιμοποιούν την ίδια μηχανή αλλά όλοι μπορούν να επηρεάζουν την απόδοση της μηχανής και του δικτύου. Αρχικά το μέρισμα είναι κενό χωρίς επεξεργαστές κειμένου ή κάποιο μεταγλωττιστή παρά μόνο τις βασικές εντολές UNIX. Η κάθε μηχανή μπορεί ανά πάσα στιγμή να μην είναι διαθέσιμη χωρίς να χαθούν οποιαδήποτε δεδομένα στο δίσκο και συμπεριφέρεται απρόσμενα ώστε να είναι μη αξιόπιστη.

Πρώτο βήμα του χρήστη είναι να αναθέσει μηχανές στο μέρισμά του δημιουργώντας το προσωπικό του δίκτυο για την εφαρμογή του. Από την στιγμή που ανατίθεται μια μηχανή από το PlanetLab χρειάζεται κάποιος χρόνος για να εγκριθεί από τον οργανισμό στον οποίο ανήκει η μηχανή, προτού ο χρήστης να μπορεί να την διαχειριστεί. Ο χρήστης δεν μπορεί να επιλέξει μηχανές οι οποίες ανήκουν στον οργανισμό του. Όπως αναφέρθηκε και πιο πάνω κάθε χρήστης μπορεί να αναθέσει μέχρι και 32 μηχανές στο μέρισμά του.

Για πρόσβαση σε οποιαδήποτε μηχανή πρέπει πρώτα να δημιουργηθεί ένα ζευγάρι κλειδιών μέσω SSH για σκοπούς επαλήθευσης της ταυτότητας. Το ζευγάρι αυτό αποτελείται από το Public Key και το Private Key. Το πρώτο γνωστοποιείται στο μέρισμα του χρήστη μέσω του λογαριασμού του από την επίσημη ιστοσελίδα του PlanetLab [10] ενώ το δεύτερο δίνεται ως στοιχείο επαλήθευσης κατά τη διαδικασία σύνδεσης με την απομακρυσμένη μηχανή. Τα κλειδιά πρέπει να είναι κρυπτογραφημένα σύμφωνα με κωδική φράση που εισάγει ο χρήστης. Η δημιουργία των δύο κλειδιών μπορεί να γίνει με την ακόλουθη εντολή:

```
ssh-keygen -t rsa -f ~/.ssh/id_rsa
```

Αφού γνωστοποιηθεί το Public Key στη σελίδα του PlanetLab, τότε ο χρήστης μπορεί να ενωθεί με κάποια μηχανή του Planet Lab εκτελώντας την εντολή:

```
ssh -l slice_name -i ~/.ssh/id_rsa node_address
```

όπου είναι `slice_name` είναι το όνομα του μερίσματος του χρήστη το οποίο του έχει ανατεθεί από τον *PI*, `id_rsa` είναι το private κρυπτογραφημένο αρχείο που έχει ο χρήστης αποθηκευμένο στον τοπικό του χώρο και `node_address` είναι η διεύθυνση της μηχανής με την οποία θα γίνει η σύνδεση. Όλες οι διευθύνσεις των μηχανών βρίσκονται στην ιστοσελίδα του Planet Lab [10]. Με την εκτέλεση της εντολής θα ζητηθεί από το χρήστη η κωδική φράση για πιστοποίηση της ταυτότητας του μερίσματος.

Αφού γίνει η σύνδεση με επιτυχία, πλέον ο χρήστης είναι έτοιμος να ξεκινήσει και να χρησιμοποιήσει τη μηχανή. Μπορεί να μεταφέρει είτε τα αρχεία του στο μέρισμα είτε απλά τα εκτελέσιμα αρχεία του στις μηχανές και να ξεκινήσει να τρέχει την εφαρμογή του. Αυτό μπορεί να γίνει μέσω κάποιου ειδικού λογισμικού (όπως winscp) με την παρακάτω εντολή:

```
scp -i ~/.ssh/id_rsa -r filename slice_name@node:
```

η οποία μεταφέρει το αρχείο με το όνομα `filename` από τον τρέχων κατάλογο στον προσωπικό χώρο του μερίσματος `slice_name` στη μηχανή `node` που ανήκει στο μέρισμα αυτό. Κατά την ολοκλήρωση της μεταφοράς ζητείται η κωδική φράση για πιστοποίηση της ταυτότητας του χρήστη.

Αναλυτικότερα εγχειρίδια χρήσης του PlanetLab μπορεί να βρει κάποιος στην επίσημη ιστοσελίδα του Planet Lab [10] και μάλιστα με οπτικοακουστικά μέσα.

5.3 Μεθοδολογία

Κατά την εκτέλεση της εφαρμογής στο PlanetLab, υπήρξαν κάποιοι περιορισμοί. Όπως αναφέρθηκε και πιο πάνω κάθε χρήστης μπορεί να αναθέσει περιορισμένο αριθμό μηχανών στο μέρισμά του. Γι’ αυτό το λόγο στα πειράματα χρησιμοποιηθήκαν μόνο 25 μηχανές. Αυτό δεν αποτέλεσε πρόβλημα στην αξιολόγηση των αλγορίθμων διότι είχε ήδη δοκιμαστεί

μεγάλος αριθμός κόμβων στο περιβάλλον προσομοίωσης οπότε πιστεύουμε ότι τα αποτελέσματα δεν θα διέφεραν ιδιαίτερα. Ο κύριος λόγος που οι αλγόριθμοι δοκιμάζονται και σε ένα πραγματικό περιβάλλον όπως είναι το PlanetLab είναι για να αποδειχτεί ότι οι αλγόριθμοι αυτοί είναι εφαρμόσιμοι και πρακτικά, δηλαδή σε ένα πραγματικά ασύγχρονο κατανεμημένο σύστημα και όχι μόνο στη θεωρία ή στην προσομοίωση.

Έγιναν, λοιπόν, πειράματα για 8, 16 και 25 κόμβους. Επιλέχθηκαν αυτές οι τιμές για μια καλύτερη και σωστότερη σύγκριση με τα αντίστοιχα πειράματα στην προσομοίωση. Αυτό επαναλήφθηκε τρεις φορές για να παρθεί στο τέλος ο μέσος όρος των αντίστοιχων περιπτώσεων. Όπως και στην προσομοίωση έτσι κι εδώ πάρθηκαν μετρήσεις για το συνολικό αριθμό των μηνυμάτων που αποστάληκαν και για το χρόνο ολοκλήρωσης των αλγορίθμων. Στον αλγόριθμο EARS ορίστηκε επιπλέον μια τιμή για το f, δηλαδή για τον αριθμό των πιθανών σφαλμάτων κατάρρευσης που μπορεί να συμβούν κατά την εκτέλεση του αλγορίθμου στο PlanetLab. Καθορίστηκε η τιμή f = 1 για να είναι καλύτερη και πιο σωστή η σύγκριση με τα αντίστοιχα αποτελέσματα της προσομοίωσης αλλά και διότι στο PlanetLab κατά μέσο όρο το 10% των κόμβων ενδέχεται να καταρρεύσει ανά πάσα στιγμή. Στον αλγόριθμο SEARS, για τη σταθερά ε ορίστηκε η ίδια τιμή με την προσομοίωση, δηλαδή η τιμή $\varepsilon = \frac{1}{100}$ για μια πιο σωστή σύγκριση με τα αντίστοιχα αποτελέσματα της προσομοίωσης. Επιπλέον η ποσότητα αυτή μαζί με τον αριθμό των κόμβων n καθορίζουν έμμεσα τον αριθμό των μηνυμάτων που αποστέλλονται σε κάθε τοπικό βήμα (βλ. ψευδοκώδικα SEARS στο Υποκεφάλαιο 3.4.2). Επειδή η τιμή $\frac{1}{100}$ είναι μια μικρή ποσότητα, συνεπάγεται ότι και η ποσότητα $\Theta(\max\{n^{\varepsilon}, 1\}\log n)$ που καθορίζει πόσα μηνύματα αποστέλλονται σε κάθε βήμα, θα είναι επίσης μικρή. Έτσι θεωρήθηκε καλό να χρησιμοποιηθεί στο PlanetLab η ίδια τιμή, τόσο για λόγους σύγκρισης με την προσομοίωση όσο και για να διαφανεί κατά πόσο το $\frac{1}{100}$ αποτελεί μια καλή εκτίμηση για το ε. Τέλος, αξίζει να σημειωθεί ότι επειδή το πρόβλημα της πληροφόρησης σε ασύγχρονα συστήματα δεν μπορεί να λυθεί αν δεν υπάρχει αξιόπιστη επικοινωνία, στο PlanetLab, όπως και στην προσομοίωση, χρησιμοποιήθηκε το πρωτόκολλο TCP για την αποστολή των μηνυμάτων στους κόμβους.

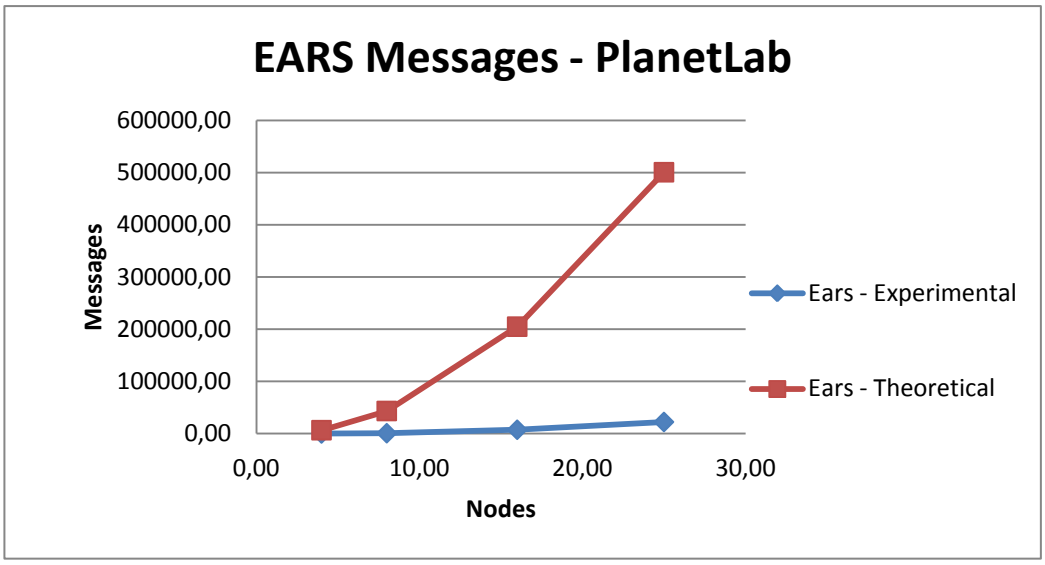
Για τη σύγκριση των αποτελεσμάτων με τα θεωρητικά έπρεπε να προηγηθεί μια εκτίμηση του d+δ. Καθώς το PlanetLab είναι ασύγχρονο σύστημα οι καθυστερήσεις δεν είναι γνωστές και σίγουρα ποικίλουν. Έτσι για να παρθεί μια εκτίμηση των καθυστερήσεων χρησιμοποιήθηκε η εντολή "ping" [19]. Η εντολή αυτή χρησιμοποιείται για τον εντοπισμό της διαθεσιμότητας και της απόδοσης ενός απομακρυσμένου πόρου του δικτύου. Με λίγα λόγια αποστέλλεται στην απομακρυσμένη μηχανή ένα πακέτο δεδομένων και στη συνέχεια η μηχανή που έστειλε το πακέτο, περιμένει απάντηση γι' αυτό το πακέτο δεδομένων. Το χρονικό διάστημα που μεσολαβεί από την αποστολή μέχρι την απάντηση δίνει μια αίσθηση για το πόσο χρόνο χρειάζεται ένα μήνυμα να αποσταλεί από τη μια μηχανή στην άλλη. Έτσι κάνοντας "ping" στις μηχανές που συμμετείχαν στα πειράματα εκτιμήθηκε ότι το d+δ είναι γύρω στα 200 ms. Περισσότερες πληροφορίες για την τεχνική του "ping" μπορούν να ανακτηθούν στα [19] και [20].

Τα πειράματα στο PlanetLab ήταν ιδιαίτερα χρονοβόρα. Το PlanetLab, ως ρεαλιστικό και ασύγχρονο δίκτυο, επέτρεπε την καθυστέρηση στην παράδοση των μηνυμάτων η οποία διέφερε από στιγμή σε στιγμή και από κόμβο σε κόμβο. Επιπλέον, ο κάθε κόμβος ανάλογα με την τοποθεσία και τα χαρακτηριστικά του είχε το δικό του ρυθμό λειτουργίας και η συμφόρηση του δικτύου προκαλούσε περαιτέρω καθυστέρηση. Για τους υπό μελέτη αλγόριθμους αυτό ήταν το ιδανικό περιβάλλον όμως είναι αλήθεια ότι κάποιοι κόμβοι ήταν πολύ πιο αργοί από κάποιους άλλους καθυστερώντας σημαντικά την εκτέλεση των πειραμάτων.

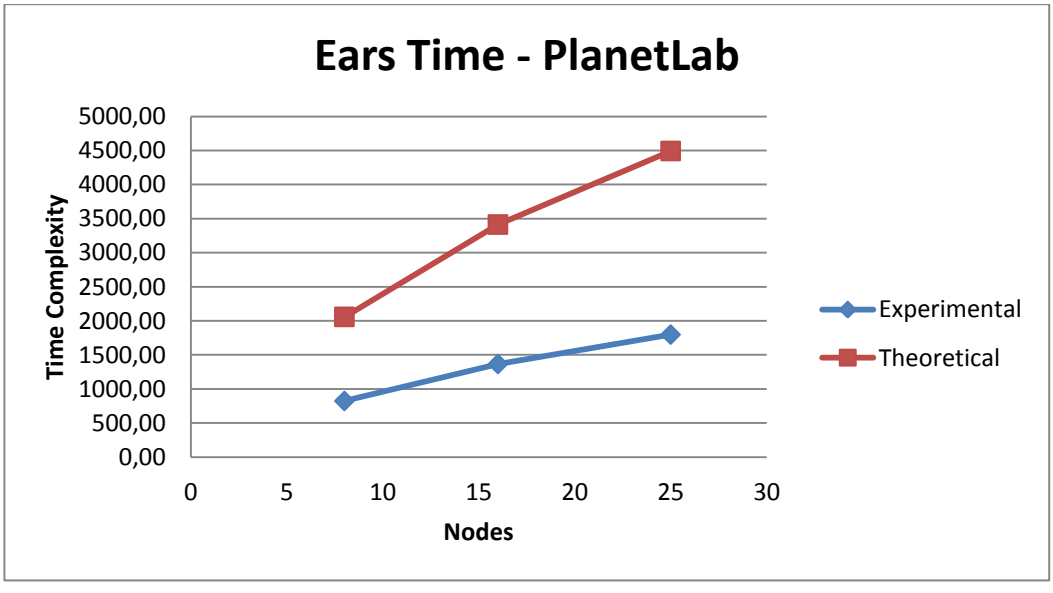
Άλλη δυσκολία που παρουσιαζόταν κατά τη διάρκεια εκτέλεσης των πειραμάτων και καθυστερούσε την όλη διαδικασία ήταν το γεγονός ότι πριν την εκτέλεση των πειραμάτων έπρεπε να ελεγχθεί κατά πόσο ήταν εφικτή η ένωση με κάθε μια από τις μηχανές που θα συμμετείχαν στα πειράματα. Υπάρχουν διάφοροι λόγοι για τους οποίους κάποιος να μην μπορεί να ενωθεί σε μια μηχανή του PlanetLab, όπως η μη μετάδοση του δημόσιου κλειδιού στη μηχανή που επιδιώκεται η ένωση, η μηχανή να μην είναι διαθέσιμη εκείνη τη χρονική περίοδο ή ακόμη να μην επιτρέπει ο ίδιος ο PI (Principal Investigator) της μηχανής την ένωση για λόγους ασφαλείας.

5.4 Αποτελέσματα Εφαρμογής Αλγορίθμου EARS στο PlanetLab

Παρακάτω φαίνονται οι γραφικές παραστάσεις του αριθμού μηνυμάτων και του χρόνου ολοκλήρωσης που πάρθηκαν για τον αλγόριθμο EARS όταν αυτός έτρεξε στο PlanetLab. Αυτές αντιπαραβάλλονται με τις αντίστοιχες θεωρητικές με d+δ ίσο με 200 ms.



Εικόνα 5.1: Αλγόριθμος EARS – Αριθμός μηνυμάτων (PlanetLab)



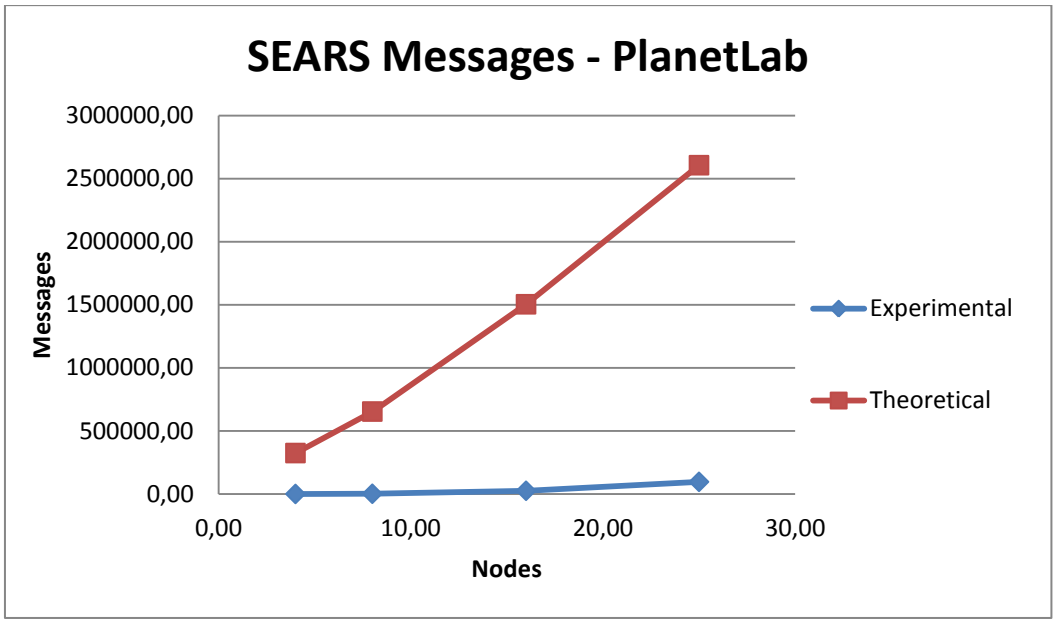
Εικόνα 5.2: Αλγόριθμος EARS – Χρόνος Ολοκλήρωσης (PlanetLab)

Όπως φαίνεται καθαρά, οι γραφικές παραστάσεις τόσο του αριθμού μηνυμάτων όσο και του χρόνου ολοκλήρωσης έχουν την ίδια μορφή με τις αντίστοιχες θεωρητικές γραφικές παραστάσεις. Αυτό είναι σημαντικό διότι αποδεικνύει ότι και στο PlanetLab που είναι πραγματικό σύστημα ο αλγόριθμος έχει την ίδια συμπεριφορά με τη θεωρητική προσέγγιση επιβεβαιώνοντας και έμπρακτα την ορθότητα της θεωρητικής ανάλυσης.

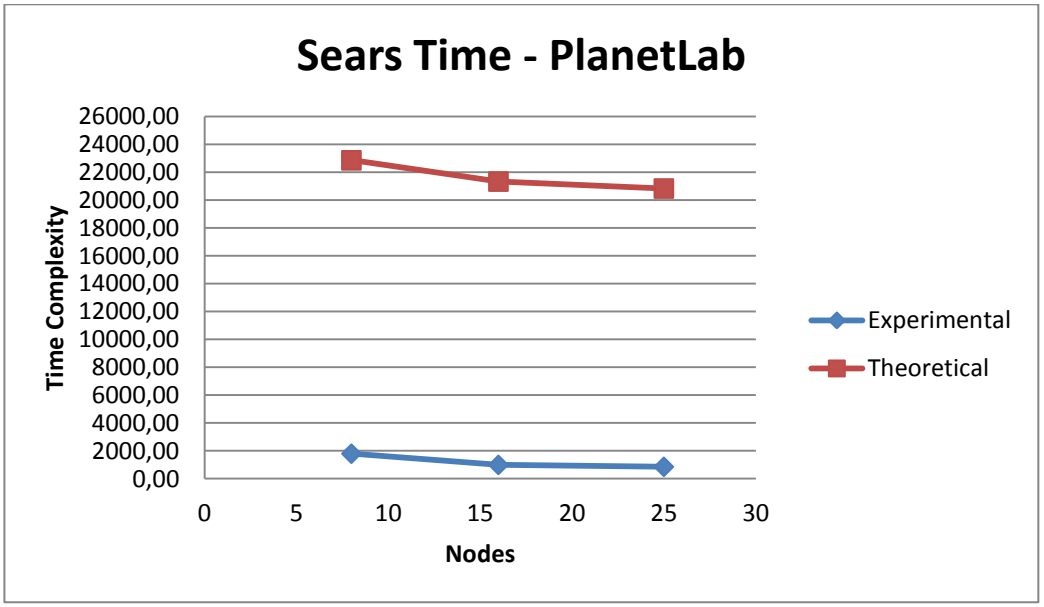
Συγκρίνοντας τα πειραματικά αποτελέσματα με τα αντίστοιχα της προσομοίωσης παρατηρείται ότι τα αποτελέσματα είναι ακόμη καλύτερα στην πράξη ως προς το χρόνο ολοκλήρωσης. Βέβαια αυτό, όπως αναφέρθηκε και προηγουμένως, οφείλεται στο ότι στην προσομοίωση οι υπολογισμοί εκτελούνταν τοπικά στην ίδια μηχανή γεγονός που επέφερε κόστος σε χρόνο. Αντιθέτως στο PlanetLab η κάθε μηχανή εκτελεί τους δικούς της υπολογισμούς ανεξάρτητα από τις υπόλοιπες και γι’ αυτό οι χρόνοι που σημειώνονται είναι μικρότεροι. Ως προς τον αριθμό των μηνυμάτων όμως παρατηρείται ότι στο PlanetLab αποστέλλονται περισσότερα μηνύματα σε σύγκριση με την προσομοίωση. Αυτό ίσως να οφείλεται στο ότι επειδή ακριβώς οι καθυστερήσεις στην παράδοση των μηνυμάτων δεν είναι σταθερές ενδέχεται κάποιοι κόμβοι να είναι πολύ πιο γρήγοροι από κάποιους άλλους. Τότε αυτοί ενδέχεται να εκτελούν περισσότερα τοπικά βήματα από τους υπόλοιπους και άρα να αποστέλλουν περισσότερα μηνύματα κατά τη διάρκεια της εκτέλεσης.

5.5 Αποτελέσματα Εφαρμογής Αλγορίθμου SEARS στο PlanetLab

Παρακάτω φαίνονται οι γραφικές παραστάσεις του αριθμού μηνυμάτων και του χρόνου ολοκλήρωσης που πάρθηκαν για τον αλγόριθμο SEARS όταν αυτός έτρεξε στο PlanetLab. Αυτές αντιπαραβάλλονται με τις αντίστοιχες θεωρητικές με d+δ ίσο με 200 ms.



Εικόνα 5.3: Αλγόριθμος SEARS – Αριθμός μηνυμάτων (PlanetLab)



Εικόνα 5.4: Αλγόριθμος SEARS – Χρόνος Ολοκλήρωσης (PlanetLab)

Όπως και στην περίπτωση του EARS έτσι και εδώ, οι γραφικές παραστάσεις του αριθμού μηνυμάτων και του χρόνου ολοκλήρωσης έχουν την ίδια μορφή με τις αντίστοιχες θεωρητικές γραφικές παραστάσεις. Αποδεικνύεται δηλαδή ότι και στο PlanetLab που είναι πραγματικό

σύστημα ο αλγόριθμος έχει την ίδια συμπεριφορά με τη θεωρητική προσέγγιση επιβεβαιώνοντας και έμπρακτα την ορθότητα της θεωρητικής ανάλυσης.

τα αποτελέσματα είναι ακόμη καλύτερα στην πράξη, ως προς το χρόνο ολοκλήρωσης. Οι λόγοι είναι παρόμοιοι με πριν και ο αλγόριθμος αποδεικνύεται για ακόμη μια φορά ότι είναι ένας σταθερού χρόνου αλγόριθμος.

Συγκρίνοντας τα αποτελέσματα με τα αντίστοιχα στο περιβάλλον προσομοίωσης, παρατηρείται ότι σε αντίθεση με τον EARS ο αριθμός των μηνυμάτων καθώς και ο χρόνος είναι μεγαλύτεροι στο PlanetLab. Αυτό ίσως είναι λογικό διότι οι πραγματικές καθυστερήσεις στην αποστολή ενός μηνύματος μπορεί να είναι μεγαλύτερες από τις καθυστερήσεις που θεωρήθηκαν στην προσομοίωση και ίσως αυτό να δικαιολογεί τις διαφορές που παρατηρούνται στις μετρήσεις. Οι μεγαλύτερες τιμές στον αριθμό των μηνυμάτων που λήφθηκαν ίσως να οφείλονται και στην τιμή του ε. Υπενθυμίζεται ότι επιλέγηκε η ίδια τιμή με την προσομοίωση, δηλαδή $\varepsilon = \frac{1}{100}$. Ίσως τελικά, δεδομένου και των καθυστερήσεων που μεταβάλλονται ανά πάσα στιγμή, να χρειαζόταν μικρότερη τιμή για το ε ώστε ο αριθμός των μηνυμάτων που αποστέλλεται σε κάθε βήμα να είναι μικρότερος.

Αξίζει να σημειωθεί ότι κατά την προσομοίωση του SEARS οι γραφικές παραστάσεις που λήφθηκαν για το χρόνο ολοκλήρωσης ήταν αύξουσες συναρτήσεις του αριθμού των κόμβων κι όχι φθίνουσες όπως φαίνεται παραπάνω. Αυτό ίσως να είναι λογικό λόγω της περιορισμένης υπολογιστικής δύναμης του προσομοιωτή. Επειδή στην προσομοίωση ένας μόνο υπολογιστής καλείται να μοιράσει τη μνήμη του ώστε να μπορέσει να εκτελέσει όλους τους υπολογισμούς ταυτόχρονα, είναι λογικό να χρειάζεται περισσότερο χρόνο να ολοκληρωθεί ο αλγόριθμος απ’ ότι σ’ ένα πραγματικό σύστημα όπου η κάθε μηχανή δρα ανεξάρτητα. Έτσι ίσως δικαιολογείται το γεγονός ότι όσο αυξάνονται οι κόμβοι (και κατ’ επέκταση οι υπολογισμοί), αυξάνεται και ο χρόνος ολοκλήρωσης αντί να μειώνεται. Ωστόσο στην πράξη όπως φάνηκε τα αποτελέσματα συμφωνούν με τα αντίστοιχα θεωρητικά, δηλαδή τελικά έχουν την ίδια συμπεριφορά.

Η σημαντικότερη παρατήρηση ως προς το χρόνο ολοκλήρωσης είναι ότι όπως στην προσομοίωση έτσι και στο PlanetLab καθώς αυξάνονται οι κόμβοι ο ρυθμός μεταβολής του χρόνου του αλγορίθμου μειώνεται και για μεγάλο αριθμό κόμβων (>15) ο χρόνος τείνει να είναι σταθερός όπως θα ήθελε η θεωρητική προσέγγιση.

5.6 Συμπεράσματα

Από την εφαρμογή των αλγορίθμων στο PlanetLab αποδεικνύεται η ορθότητα και η πρακτικότητά τους και σε πραγματικό περιβάλλον. Συγκρίνοντας τα αποτελέσματα με τα αντίστοιχα της προσομοίωσης παρατηρείται ότι ως προς το χρόνο ολοκλήρωσης τα αποτελέσματα του EARS στο PlanetLab είναι πολύ καλύτερα. Αυτό είναι ίσως λογικό εξαιτίας του ότι ο προσομοιωτής τρέχει σε μια μηχανή μόνο και συνεπώς ο φόρτος εργασίας είναι πολύ μεγάλος. Αντιθέτως στο PlanetLab η κάθε μηχανή τρέχει ανεξάρτητα από τις υπόλοιπες χωρίς να χρειάζεται να μοιράζει την τοπική της μνήμη όπως χρειάζεται να κάνει ο προσομοιωτής για την κάθε εικονική μηχανή.

Στον SEARS όμως παρατηρείται το αντίθετο. Ο χρόνος ολοκλήρωσης είναι μεγαλύτερος στο PlanetLab όμως καθώς μεγαλώνει ο αριθμός των κόμβων ο χρόνος μειώνεται και για μεγάλο αριθμό κόμβων τείνει να είναι σταθερός. Ως προς τον αριθμό των μηνυμάτων παρατηρείται ότι και στους δυο αλγόριθμους τα μηνύματα που αποστέλλονται είναι περισσότερα σε σύγκριση με την προσομοίωση. Αυτό ίσως να οφείλεται κατά κύριο λόγο στο ότι κάποιοι από τους κόμβους είναι πιο γρήγοροι από κάποιους άλλους εκτελώντας έτσι περισσότερα τοπικά βήματα από και κατά συνέπεια αποστέλλοντας περισσότερα μηνύματα κατά τη διάρκεια της εκτέλεσης. Αυτό ίσως είναι λογικό διότι οι πραγματικές καθυστερήσεις στην αποστολή ενός μηνύματος ενδέχεται να είναι πολύ μεγαλύτερες από τις καθυστερήσεις που θεωρήθηκαν στην προσομοίωση και ίσως αυτό να δικαιολογεί τις διαφορές που παρατηρούνται στις μετρήσεις.

Κεφάλαιο 6

Σύγκριση Αλγορίθμων

Αφού μελετήθηκαν και αξιολογήθηκαν ξεχωριστά οι δύο αλγόριθμοι EARS και SEARS ως προς τις δύο παραμέτρους, αριθμό μηνυμάτων που αποστέλλονται και χρόνο ολοκλήρωσης, ακολουθεί η μεταξύ τους σύγκριση ως προς τις ίδιες παραμέτρους για να διαπιστωθεί εάν και κατά πόσο είναι καλύτερος ο ένας αλγόριθμος από τον άλλο, ποια τα πλεονεκτήματα και ποια τα μειονεκτήματα του καθενός και αν όλα αυτά συνάδουν με τα θεωρητικά συμπεράσματα. Πρώτα η σύγκριση γίνεται σε περιβάλλον προσομοίωσης μέσω του YALPS και ακολούθως σε πραγματικό περιβάλλον μέσω του Planet Lab.

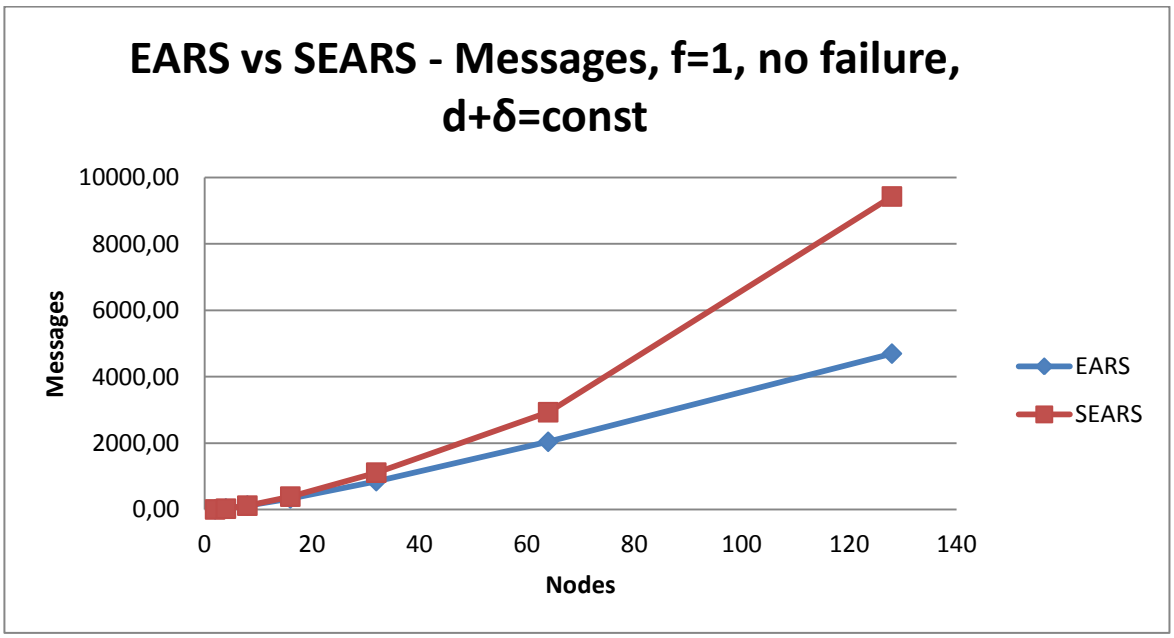
6.1 Σύγκριση ως Προς τον Αριθμό Μηνυμάτων

Ο συνολικός αριθμός μηνυμάτων που ανταλλάζονται μεταξύ των κόμβων του συστήματος εξαρτάται από τον τρόπο κατασκευής του κάθε αλγορίθμου. Ίσως να ισχυριζόταν κανείς ότι αφού ο SEARS είναι ένας σταθερού χρόνου αλγόριθμος τότε θα χρειάζεται λιγότερα μηνύματα σε σύγκριση με τον EARS που τρέχει για περισσότερο χρόνο. Όμως στην πραγματικότητα συμβαίνει το αντίθετο και αυτό οφείλεται στον τρόπο κατασκευής του κάθε αλγορίθμου. Στον EARS, σε κάθε βήμα ο κάθε κόμβος επιλέγει τυχαία έναν άλλο κόμβο και αποστέλλει σ' αυτόν ένα μήνυμα. Αντιθέτως στον SEARS, σε κάθε βήμα ο κάθε κόμβος επιλέγει τυχαία $\max\{n^{\varepsilon},1\}\log n$ άλλους κόμβους και τους στέλλει ένα μήνυμα. Δηλαδή στο ίδιο βήμα αποστέλλει περισσότερα μηνύματα και είναι αυτή η τεχνική που οδηγεί σε σταθερού χρόνου αλγόριθμο.

Επιπλέον, στον EARS, όταν ένας κόμβος θεωρήσει ότι έχει στείλει όλη την πληροφορία του σε όλους τους υπόλοιπους κόμβους, περνά στη φάση του "shut-down" και καθ' όλη τη διάρκεια της φάσης αυτής σταματά να στέλλει μηνύματα. Έτσι μειώνεται σημαντικά ο αριθμός των μηνυμάτων που αποστέλλονται. Το ίδιο συμβαίνει και στον SEARS, όμως τότε ο κάθε κόμβος περνά στη "shut-down" φάση μόνο μια φορά και κατά συνέπεια ο αριθμός των μηνυμάτων δε μειώνεται τόσο πολύ. Άρα, λοιπόν, σε κάθε προσομοίωση ο αριθμός των μηνυμάτων που ανταλλάζονται στον αλγόριθμο EARS θα είναι αρκετά μικρότερος από τον αριθμό των μηνυμάτων που ανταλλάζονται στον αλγόριθμο SEARS.

6.1.1 Περίπτωση με σταθερό d+δ

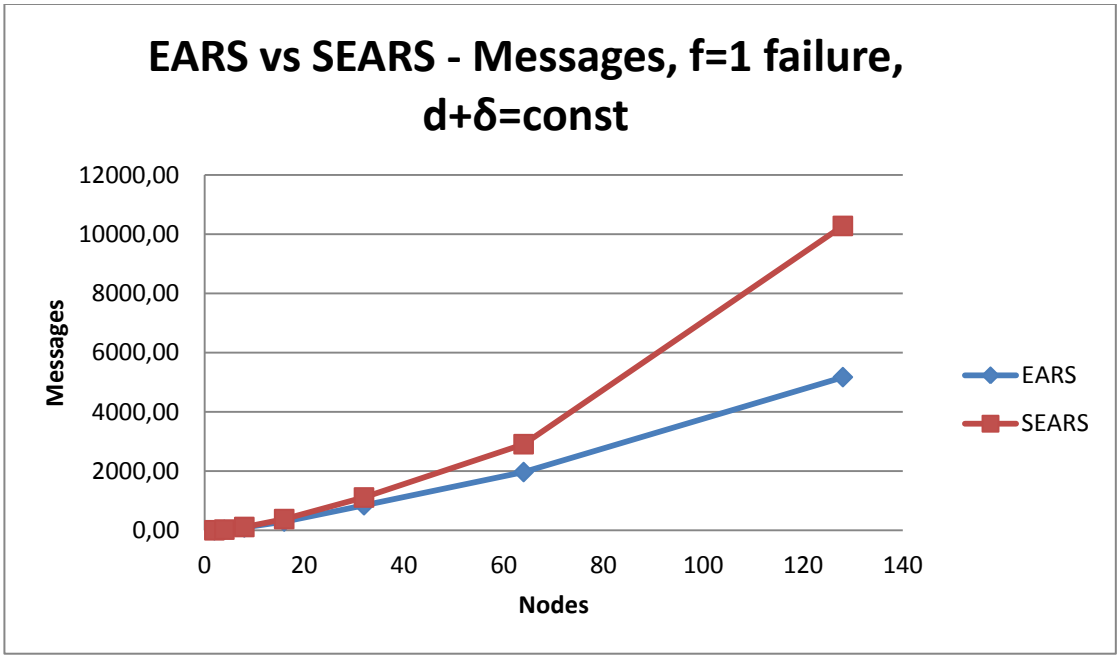
Εξετάζεται πρώτα η περίπτωση όπου η συνολική καθυστέρηση d+δ είναι σταθερή καθ' όλη τη διάρκεια της προσομοίωσης και ίση με 2 ms. Παρακάτω φαίνεται η γραφική παράσταση του αριθμού των μηνυμάτων συναρτήσει του αριθμού των κόμβων του συστήματος για τους δύο αλγορίθμους σε περιβάλλον προσομοίωσης στην περίπτωση όπου f=1 αλλά δε σημειώνεται κανένα σφάλμα κατάρρευσης (ομοίως για f=n/4):



Εικόνα 6.1: Σύγκριση EARS και SEARS – Αριθμός μηνυμάτων (YALPS)

Όπως φαίνεται και από τα πειραματικά αποτελέσματα, όντως είναι τα αναμενόμενα. Ο αριθμός μηνυμάτων που ανταλλάζονται μεταξύ των κόμβων κατά την προσομοίωση του αλγορίθμου EARS είναι πολύ μικρότερος από τον αριθμό των μηνυμάτων που ανταλλάζονται κατά την προσομοίωση του αλγορίθμου SEARS για τους λόγους που αναφέρθηκαν πιο πάνω.

Παρόμοια συμπεράσματα εξάγονται και όταν κατά την προσομοίωση συμβαίνουν f καταρρεύσεις κόμβων. Όπως δείχνει και η σχετική γραφική παράσταση που ακολουθεί (όμοια για f=n/4), παρόλο που μέχρι ένας κόμβος καταρρέει, πάλι στον αλγόριθμο SEARS λόγω τέτοιας κατασκευής, τα μηνύματα που αποστέλλονται είναι περισσότερα από αυτά που αποστέλλονται στον αλγόριθμο EARS.

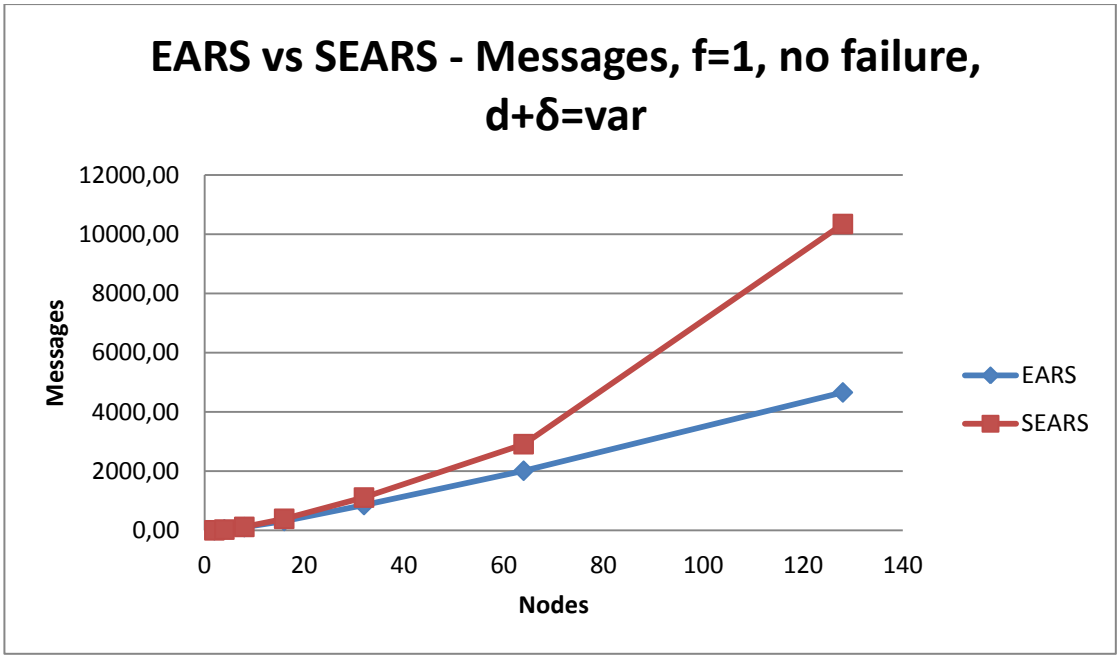


Εικόνα 6.2: Σύγκριση EARS και SEARS – Αριθμός μηνυμάτων (YALPS)

6.1.2 Περίπτωση με μεταβλητό d+δ

Εξετάζεται τώρα η περίπτωση όπου η συνολική καθυστέρηση d+δ δεν είναι σταθερή καθ' όλη τη διάρκεια της προσομοίωσης αλλά μεταβάλλεται τυχαία μεταξύ 2 και 10 ms. Παρακάτω φαίνεται η γραφική παράσταση του αριθμού των μηνυμάτων συναρτήσει του

αριθμού των κόμβων του συστήματος για τους δύο αλγορίθμους σε περιβάλλον προσομοίωσης στην περίπτωση όπου f=1 (όμοια για f=n/4) αλλά δε σημειώνεται κανένα σφάλμα κατάρρευσης:

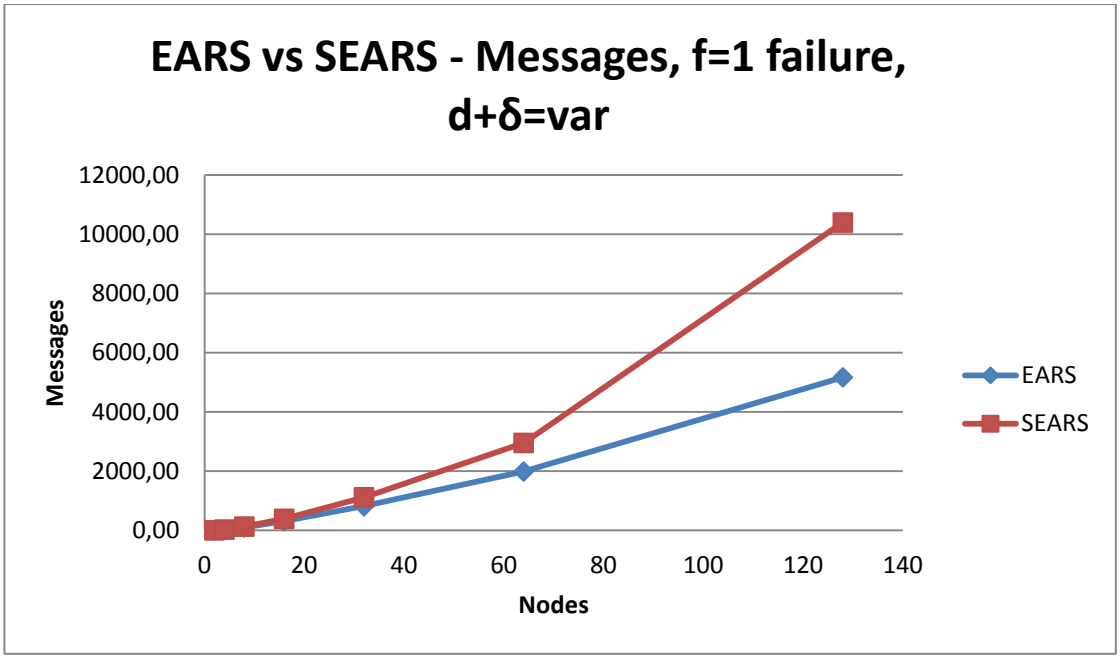


Εικόνα 6.3: Σύγκριση EARS και SEARS – Αριθμός μηνυμάτων (YALPS)

Όπως φαίνεται και από τα πειραματικά αποτελέσματα, όντως είναι τα αναμενόμενα. Ο αριθμός μηνυμάτων που ανταλλάζονται μεταξύ των κόμβων κατά την προσομοίωση του αλγορίθμου EARS είναι πολύ μικρότερος από τον αριθμό των μηνυμάτων που ανταλλάζονται κατά την προσομοίωση του αλγορίθμου SEARS για τους ίδιους λόγους που αναφέρθηκαν πιο πάνω για την περίπτωση όπου d+δ είναι σταθερό. Συγκρίνοντας τα αποτελέσματα με τα αντίστοιχα όπου d+δ είναι σταθερό παρατηρείται ότι η διαφορά των δύο αλγορίθμων στον αριθμό των μηνυμάτων είναι μεγαλύτερη όταν το d+δ είναι μεταβλητό. Αυτό οφείλεται στις μη σταθερές καθυστερήσεις οι οποίες είναι τυχαίες και άγνωστες σε κάθε εκτέλεση. Είναι, λοιπόν, φυσικό οι καθυστερήσεις να μην είναι ακριβώς οι ίδιες στις δύο εκτελέσεις που συγκρίνονται κι αυτό ίσως δικαιολογεί τη διαφορά που παρατηρείται.

Όμοια συμπεράσματα εξάγονται και όταν κατά την προσομοίωση συμβαίνει μέχρι f=1 κατάρρευση κόμβου (όμοια για f=n/4). Όπως δείχνει και η σχετική γραφική παράσταση που

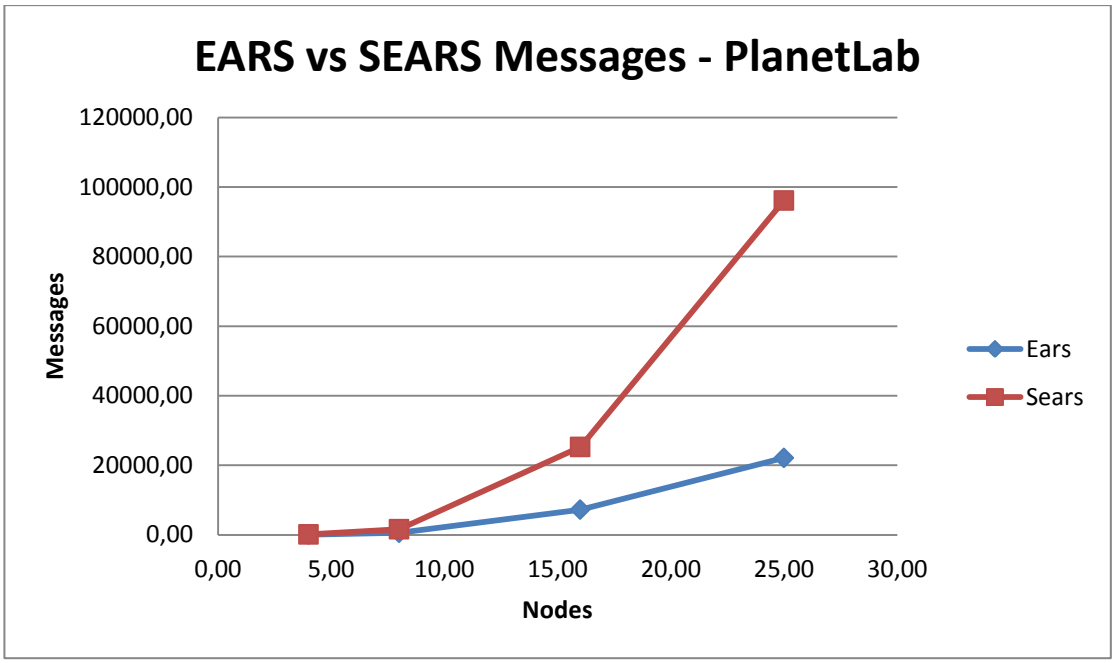
ακολουθεί, παρόλο που κάποιοι κόμβοι καταρρέουν, πάλι στον αλγόριθμο SEARS τα μηνύματα που αποστέλλονται είναι περισσότερα από αυτά που αποστέλλονται στον αλγόριθμο EARS.



Εικόνα 6.4: Σύγκριση EARS και SEARS – Αριθμός μηνυμάτων (YALPS)

6.1.3 Περίπτωση εφαρμογής στο Planet Lab

Εντελώς παρόμοια συμπεράσματα εξάγονται και από τα αντίστοιχα αποτελέσματα της εφαρμογής στο PlanetLab. Όπως δείχνει και η σχετική γραφική παράσταση που ακολουθεί, στον αλγόριθμο SEARS ο αριθμός μηνυμάτων είναι κατά πολύ μεγαλύτερος από αυτόν στον αλγόριθμο EARS για τους ίδιους λόγους που περιγράφηκαν πιο πάνω.



Εικόνα 6.5: Σύγκριση EARS και SEARS – Αριθμός μηνυμάτων (PlanetLab)

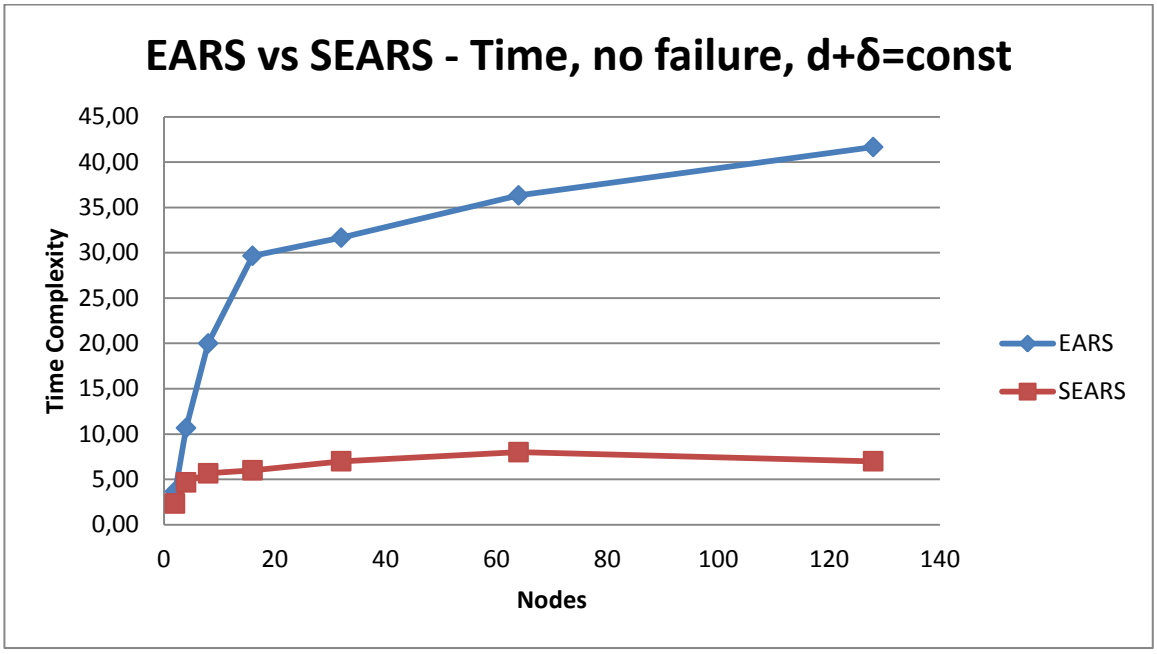
6.2 Σύγκριση ως Προς το Χρόνο Ολοκλήρωσης

Ο χρόνος που απαιτείται για την ορθή επίλυση του προβλήματος πληροφόρησης εξαρτάται από τον τρόπο κατασκευής του κάθε αλγορίθμου. Στον πρώτο αλγόριθμο που αναλύθηκε, τον αλγόριθμο EARS, όταν ένας κόμβος θεωρήσει ότι έχει στείλει όλη την πληροφορία του σε όλους τους υπόλοιπους κόμβους τότε περνά στη φάση του "shut-down" κατά την οποία σταματά να στέλλει μηνύματα όμως όχι και να λαμβάνει. Έτσι υπάρχει η πιθανότητα κατά τη φάση αυτή να λάβει κάποια νέα πληροφορία η οποία δεν έχει ακόμη διαδοθεί σε κάποιο κόμβο. Σε μια τέτοια περίπτωση, αμέσως διακόπτεται η φάση του "shut-down" και αρχίζει ολόκληρη η διαδικασία από την αρχή. Αυτό έχει ως συνέπεια να χρειαστούν επιπλέον βήματα πριν ολοκληρωθεί οριστικά ο αλγόριθμος άρα και μεγαλύτερος χρόνος. Αντιθέτως στο δεύτερο αλγόριθμο που αναλύθηκε, τον αλγόριθμο SEARS, είναι προγραμματισμένο κάθε κόμβος να περνά στη "shut-down" φάση μόνο μια φορά, ασχέτως αν υπάρχει κάποια πληροφορία που να μην έχει διαδοθεί ακόμη σε κάποιο κόμβο. Είναι συνεπώς, λογικό ότι σ' αυτή την περίπτωση ο χρόνος θα είναι μικρότερος. Σε κάθε

προσομοίωση, λοιπόν, αναμένεται ότι ο χρόνος ολοκλήρωσης του αλγορίθμου EARS θα είναι μεγαλύτερος από το χρόνο ολοκλήρωσης του αλγορίθμου SEARS.

6.2.1 Περίπτωση με σταθερό d+δ

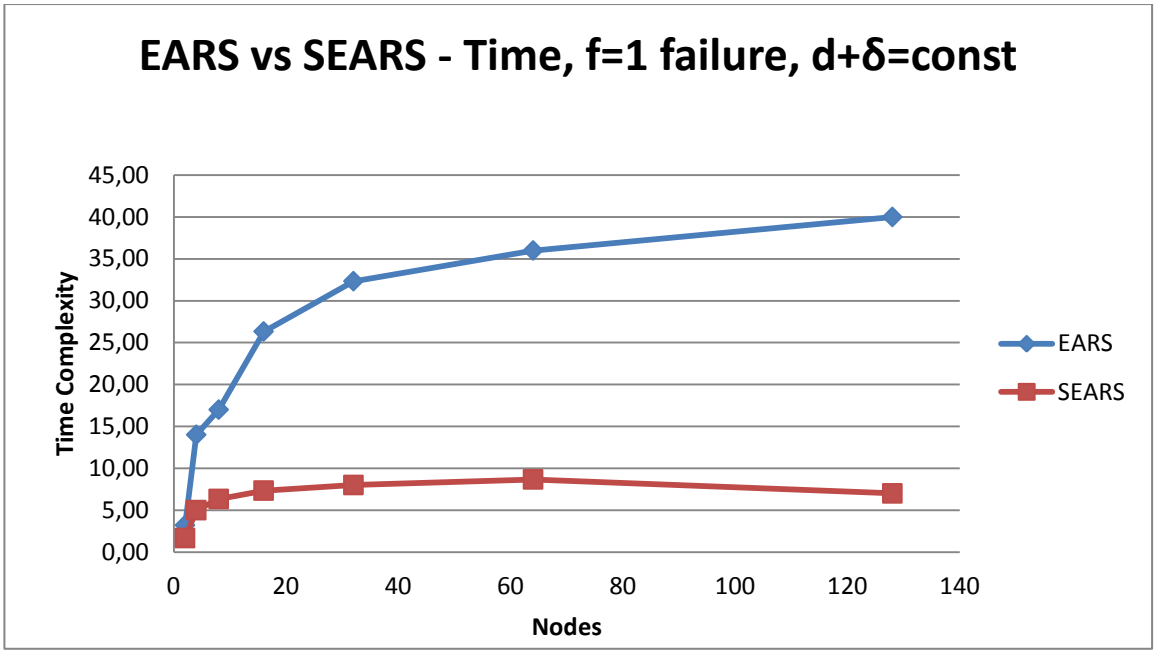
Εξετάζεται πρώτα η περίπτωση όπου η συνολική καθυστέρηση d+δ είναι σταθερή καθ' όλη τη διάρκεια της προσομοίωσης και ίση με 2 ms. Παρακάτω φαίνεται η γραφική παράσταση του χρόνου ολοκλήρωσης συναρτήσει του αριθμού των κόμβων του συστήματος για τους δύο αλγορίθμους σε περιβάλλον προσομοίωσης στην περίπτωση όπου f=1 αλλά δε σημειώνεται κανένα σφάλμα κατάρρευσης (όμοια για f=n/4):



Εικόνα 6.6: Σύγκριση EARS και SEARS – Χρόνος Ολοκλήρωσης (YALPS)

Όπως φαίνεται και από τα πειραματικά αποτελέσματα, όντως είναι τα αναμενόμενα. Ο χρόνος που απαιτούνται κατά την προσομοίωση του αλγορίθμου EARS είναι πολύ μεγαλύτερος από τον αριθμό των γύρων που απαιτούνται κατά την προσομοίωση του αλγορίθμου SEARS.

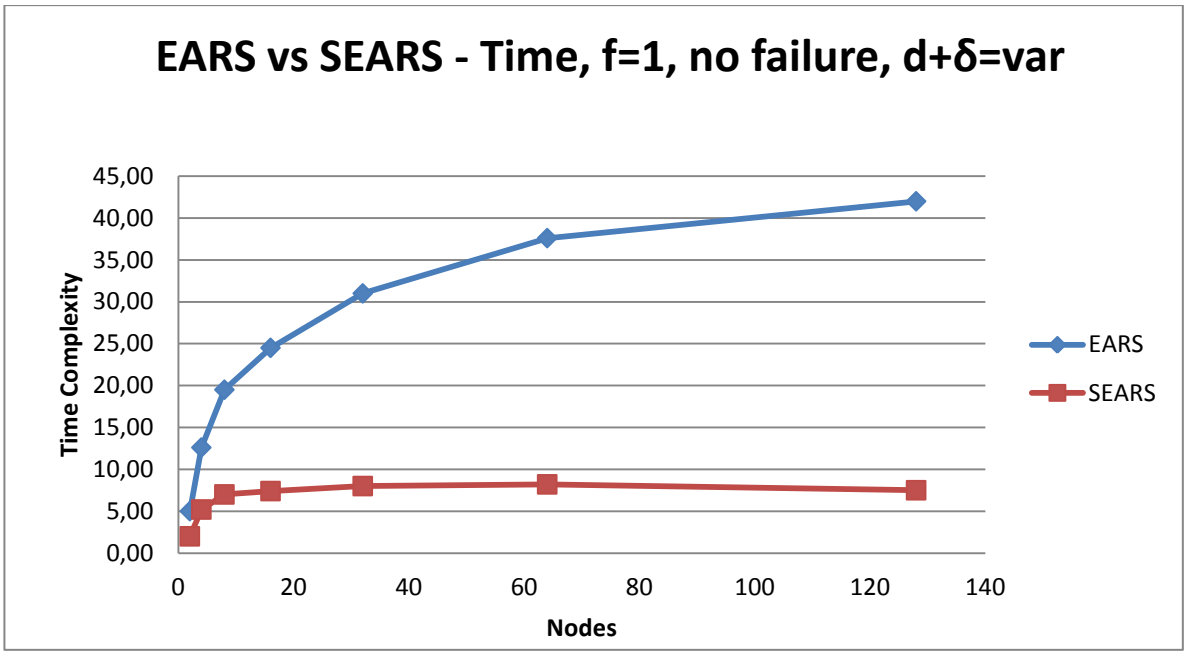
Όμοια συμπεράσματα εξάγονται και όταν κατά την προσομοίωση συμβαίνει μέχρι f=1 κατάρρευση κόμβου (όμοια για f=n/4). Όπως δείχνει και η σχετική γραφική παράσταση που ακολουθεί, παρόλο που κάποιοι κόμβοι καταρρέουν, πάλι στον αλγόριθμο SEARS ο χρόνος είναι καθαρά μικρότερος από αυτόν του αλγορίθμου EARS και μάλιστα για μεγάλο αριθμό κόμβων συγκλίνει σε ένα σταθερό αριθμό.



Εικόνα 6.7: Σύγκριση EARS και SEARS – Χρόνος Ολοκλήρωσης (YALPS)

6.2.2 Περίπτωση με μεταβλητό d+δ

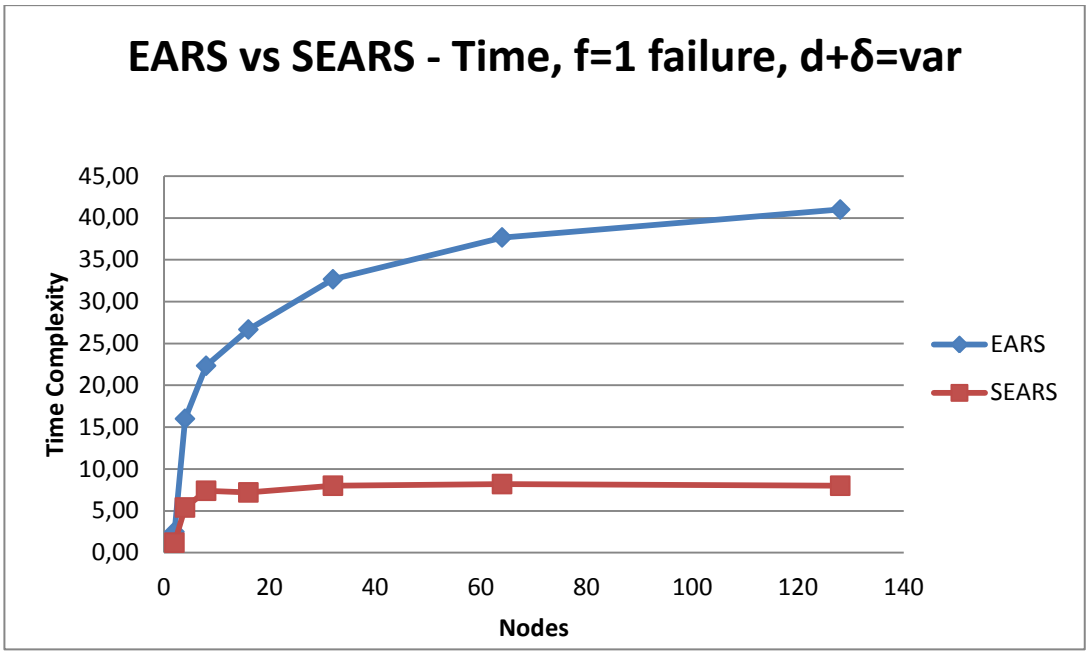
Εξετάζεται τώρα η περίπτωση όπου η συνολική καθυστέρηση d+δ δεν είναι σταθερή καθ' όλη τη διάρκεια της προσομοίωσης αλλά μεταβάλλεται τυχαία μεταξύ 2 και 10 ms. Παρακάτω φαίνεται η γραφική παράσταση του χρόνου ολοκλήρωσης συναρτήσει του αριθμού των κόμβων του συστήματος για τους δύο αλγορίθμους σε περιβάλλον προσομοίωσης στην περίπτωση όπου f=1 αλλά δε σημειώνεται κανένα σφάλμα κατάρρευσης (όμοια για f=n/4):



Εικόνα 6.8: Σύγκριση EARS και SEARS – Χρόνος Ολοκλήρωσης (YALPS)

Όπως φαίνεται και από τα πειραματικά αποτελέσματα, όντως είναι τα αναμενόμενα. Ο χρόνος που απαιτείται για την ολοκλήρωση του αλγορίθμου EARS είναι πολύ μεγαλύτερος από το χρόνο που απαιτείται για την ολοκλήρωση του αλγορίθμου SEARS για τους ίδιους λόγους που αναφέρθηκαν πιο πάνω για την περίπτωση όπου d+δ είναι σταθερό.

Όμοια συμπεράσματα εξάγονται και όταν κατά την προσομοίωση συμβαίνει μέχρι f=1 κατάρρευση κόμβου (όμοια για f=n/4). Η γραφική παράσταση που ακολουθεί, δείχνει ότι παρόλο που κάποιοι κόμβοι καταρρέουν, πάλι στον αλγόριθμο SEARS ο χρόνος είναι καθαρά μικρότερος από αυτόν που απαιτείται στον αλγόριθμο EARS και μάλιστα για μεγάλο αριθμό κόμβων συγκλίνει σε σταθερό αριθμό.

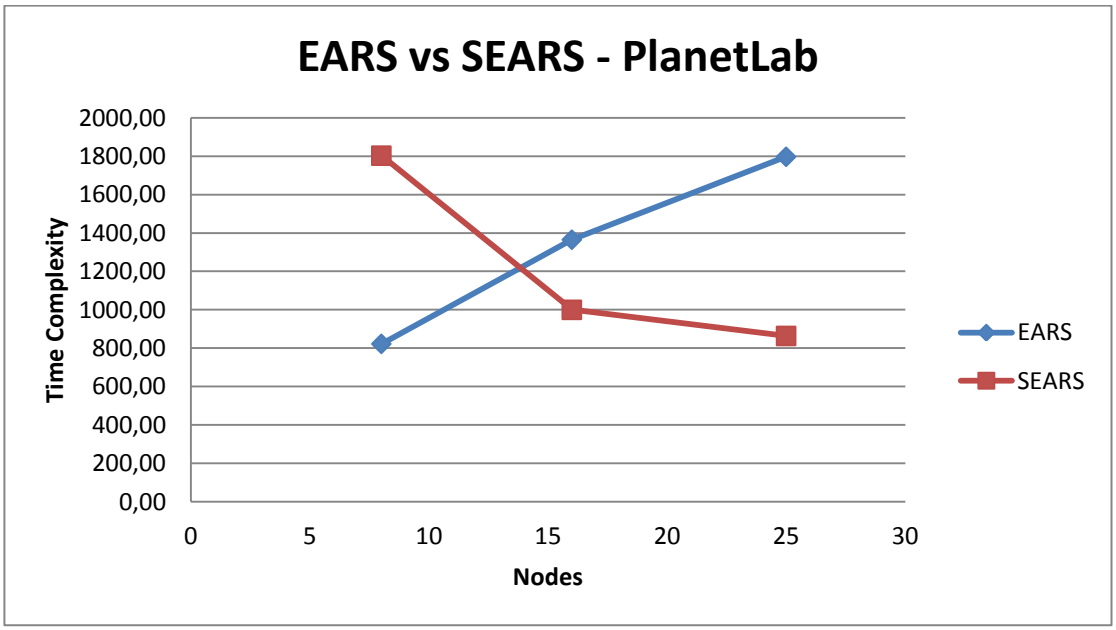


Εικόνα 6.9: Σύγκριση EARS και SEARS – Χρόνος Ολοκλήρωσης (YALPS)

6.2.3 Περίπτωση εφαρμογής στο Planet Lab

Συγκρίνοντας τα αποτελέσματα από την εφαρμογή των δύο αλγορίθμων στο PlanetLab οι παρατηρήσεις οδηγούν σε ένα ενδιαφέρον συμπέρασμα. Όπως δείχνει η γραφική παράσταση που ακολουθεί, αρχικά, για μικρό αριθμό κόμβων (μέχρι 13 κόμβους περίπου) ο χρόνος ολοκλήρωσης του SEARS είναι μεγαλύτερος από αυτόν του EARS. Δηλαδή μέχρι και για 13 κόμβους ο EARS φαίνεται να είναι πιο αποδοτικός από τον SEARS. Καθώς αυξάνεται όμως ο αριθμός των κόμβων (>13) η συμπεριφορά των αλγορίθμων είναι η γνωστή και συμφωνεί με τη θεωρητική προσέγγιση και τα αποτελέσματα της προσομοίωσης. Ο λόγος που συμβαίνει κάτι τέτοιο είναι πιθανότατα το γεγονός ότι για μικρά δίκτυα η τεχνική της αποστολής περισσότερων μηνυμάτων για εξασφάλιση καλύτερου χρόνου επιφέρει μεγαλύτερη πολυπλοκότητα μηνύματος η οποία είναι ουσιαστικά αχρείαστη, αφού όταν οι κόμβοι σε ένα δίκτυο είναι λιγότεροι σίγουρα μπορούν να ενημερωθούν με πολύ λιγότερα μηνύματα και μάλιστα σημειώνοντας όχι κακούς χρόνους. Αντιθέτως για μεγάλα δίκτυα ο

SEARS σημειώνει σχεδόν σταθερούς χρόνους και αρκετά μικρότερους από τους αντίστοιχους του EARS για τους ίδιους λόγους που περιγράφηκαν στο Υποκεφάλαιο 6.2.



Εικόνα 6.10: Σύγκριση EARS και SEARS – Χρόνος Ολοκλήρωσης (PlanetLab)

6.4 Συμπεράσματα

Από τη σύγκριση των δύο αλγορίθμων ως προς τον αριθμό των μηνυμάτων και το χρόνο ολοκλήρωσης λαμβάνεται το συμπέρασμα ότι τόσο ο αλγόριθμος EARS όσο και ο αλγόριθμος SEARS είναι γενικά αποδοτικοί στην ενημέρωση των κόμβων του συστήματος. Και οι δύο αλγόριθμοι επιβεβαιώνουν και στην πράξη τα όσα έχουν αποδειχθεί θεωρητικά. Ακόμα και αν κατά την εκτέλεση των αλγορίθμων συμβαίνουν σφάλματα κατάρρευσης κόμβων, αποδεικνύεται και πειραματικά ότι ο EARS προσπαθεί να κάνει εξισορρόπηση μεταξύ του αριθμού των μηνυμάτων που αποστέλλονται και του χρόνου που απαιτείται πετυχαίνοντας έτσι τελικά την ενημέρωση των κόμβων με λιγότερα μηνύματα και σε λογαριθμικό χρόνο. Από την άλλη ο SEARS προσπαθεί, αποστέλλοντας λογαριθμικό αριθμό μηνυμάτων σε κάθε βήμα, να μειώσει σε μεγάλο βαθμό το χρόνο που απαιτείται για την

ενημέρωση των κόμβων του συστήματος. Έτσι τελικά πετυχαίνει την ενημέρωση των κόμβων με λογαριθμικό αριθμό μηνυμάτων σε σχεδόν σταθερό χρόνο.

Τα πειραματικά αποτελέσματα από την εφαρμογή στο PlanetLab, ως προς τον αριθμό των μηνυμάτων επιβεβαιώνουν τη θεωρητική ανάλυση των αλγορίθμων. Πράγματι, ο αριθμός των μηνυμάτων που ανταλλάζονται στον αλγόριθμο EARS είναι αρκετά μικρότερος από τον αριθμό των μηνυμάτων που ανταλλάζονται στον αλγόριθμο SEARS, αφού η διάρκεια της "shut-down" φάσης κατά την οποία οι αλγόριθμοι σταματούν να στέλλουν μηνύματα είναι μεγαλύτερη στον EARS απ' ότι στον SEARS.

Τα πειραματικά αποτελέσματα ως προς τον χρόνο ολοκλήρωσης επιβεβαιώνουν τη θεωρητική ανάλυση των αλγορίθμων. Πράγματι, ο χρόνος που απαιτείται για την ολοκλήρωση του αλγορίθμου EARS είναι μεγαλύτερος από το χρόνο που απαιτείται για την ολοκλήρωση του αλγορίθμου SEARS. Στον SEARS ο χρόνος είναι ιδιαίτερα μειωμένος κι αυτό εξηγείται τόσο από το γεγονός ότι στον ίδιο βήμα επιλέγονται τυχαία περισσότεροι από ένας κόμβοι για να τους σταλούν μηνύματα – άρα ενημερώνονται περισσότεροι κόμβοι πιο γρήγορα – όσο και από το γεγονός ότι ο κάθε κόμβος περνά στη φάση του "shut-down" μόνο μια φορά. Η βασικότερη διαφορά των δύο αλγορίθμων όμως είναι στο ρυθμό αύξησης του χρόνου ως προς τον αριθμό των κόμβων. Στον SEARS παρατηρείται ότι ο ρυθμός αύξησης μειώνεται όσο αυξάνονται οι κόμβοι στο σύστημα και για αρκετά μεγάλο αριθμό κόμβων ο χρόνος παραμένει σχεδόν σταθερός, ενώ στον EARS ο χρόνος αυξάνεται λογαριθμικά σε σχέση με τον αριθμό των κόμβων. Ωστόσο, τελικά και οι δύο αλγόριθμοι επιβεβαιώνουν το θεωρητικό τους άνω φράγμα ως προς το χρόνο ολοκλήρωσης, δηλαδή τα πειραματικά αποτελέσματα συνάδουν με τα αντίστοιχα θεωρητικά.

Συμπερασματικά, το πιο σημαντικό πλεονέκτημα των δύο αλγορίθμων, τόσο ως προς τον αριθμό των μηνυμάτων που αποστέλλονται, όσο και ως προς το χρόνο ολοκλήρωσης, είναι η ευρωστία τους. Ακόμη και στην παρουσία σφαλμάτων τόσο ο EARS όσο και ο SEARS παραμένουν ανεκτικοί, και κατορθώνουν τελικά να ενημερώνουν ορθά τους κόμβους του συστήματος χωρίς να επηρεάζεται σε μεγάλο βαθμό η απόδοσή τους.

Αν έπρεπε κάποιος να διαλέξει έναν από τους δύο αλγόριθμους για να χρησιμοποιήσει σε μια κατανεμημένη εφαρμογή, πριν αποφασίσει θα έπρεπε πρώτα να εξετάσει τις συνθήκες κάτω από τις οποίες θα εφαρμοστεί ο αλγόριθμος. Αν το δίκτυο στο οποίο θα εφαρμοστεί είναι μικρό, δηλαδή αποτελείται από καμιά δεκαριά κόμβους μόνο, τότε πιθανότατα να προτιμούσε τη χρήση του EARS, καθώς παρατηρήθηκε από τα πειραματικά αποτελέσματα στο PlanetLab (βλ. Εικόνα 6.10) ότι για αριθμό κόμβων όχι μεγαλύτερο από 13 κόμβους ο EARS σημειώνει μικρότερους χρόνους σε σύγκριση με τον SEARS ενώ παράλληλα αποστέλλει μικρότερο αριθμό μηνυμάτων από αυτόν (βλ. Εικόνα 6.5). Αξίζει να σημειωθεί όμως ότι γενικά από τα πειραματικά αποτελέσματα ως προς τα μηνύματα και το χρόνο, για μικρό αριθμό κόμβων θα μπορούσαν να χρησιμοποιηθούν και οι δύο αλγόριθμοι χωρίς σοβαρό κόστος σε μηνύματα και χρόνο. Αντιθέτως αν το δίκτυο στο οποίο θα εφαρμοστεί ο αλγόριθμος είναι μεγαλύτερο τότε θα πρέπει να λάβει υπόψη του και άλλες συνθήκες εκτός από το μέγεθος του δικτύου. Αν επιβάλλεται κατά την εφαρμογή του αλγορίθμου να σημειώνεται όσο το δυνατό μικρότερη συμφόρηση στο δίκτυο, δηλαδή όσο το δυνατό λιγότερη διακίνηση μηνυμάτων στους κόμβους, τότε θα προτιμούσε τη χρήση του EARS αφού αυτός όπως φάνηκε από τα πειραματικά αποτελέσματα (βλ. Εικόνες 6.1–6.5) πετυχαίνει ελαχιστοποίηση του αριθμού των μηνυμάτων που αποστέλλονται ολοκληρώνοντας τη λειτουργία του μάλιστα σε χρόνο λογαριθμικό (βλ. Εικόνες 6.6–6.9). Αν όμως το σημαντικότερο σε μία εφαρμογή είναι η ενημέρωση των κόμβων με όλες τις πληροφορίες σε βέλτιστο χρόνο, τότε θα προτιμούσε τη χρήση του SEARS ο οποίος, παρόλο που αποστέλλει περισσότερα μηνύματα σε σύγκριση με τον EARS (βλ. Εικόνες 6.1–6.5), όπως φάνηκε από τα πειραματικά αποτελέσματα (βλ. Εικόνες 6.6–6.9) για μεγάλο αριθμό κόμβων σημειώνει καλύτερους χρόνους από αυτόν και πετυχαίνει την ενημέρωση των κόμβων σε σχεδόν σταθερό χρόνο.

Κεφάλαιο 7

Επίλογος

7.1 Γενικά Συμπεράσματα

Το πρόβλημα της μετάδοσης των πληροφοριών μεταξύ των κόμβων ενός ασύγχρονου κατανεμημένου συστήματος, γνωστό ως πρόβλημα της πληροφόρησης, έχει απασχολήσει πολλούς ερευνητές οι οποίοι προσπάθησαν να κατασκευάσουν διάφορους αλγόριθμους που να επιλύουν το ασύγχρονο πρόβλημα της πληροφόρησης και ταυτόχρονα να είναι αποδοτικοί. Η επιλογή ενός εύρωστου αλγορίθμου, δηλαδή αλγορίθμου αποδοτικού και συνάμα ανεκτικού σε σφάλματα, είναι καθοριστική για τη γενικότερη απόδοση ενός συστήματος.

Στην εργασία αυτή υλοποιήθηκαν και αξιολογήθηκαν δύο αλγόριθμοι πληροφόρησης οι οποίοι αν και είχαν αρκετά κοινά στοιχεία ως προς την κατασκευή τους, εντούτοις κάθε ένας από αυτούς έδινε διαφορετικά αποτελέσματα ως προς τον αριθμό των μηνυμάτων που ανταλλάζονταν και το χρόνο ολοκλήρωσης. Η πειραματική αξιολόγηση του καθενός ξεχωριστά οδήγησε σε διαφορετικά συμπεράσματα που όμως όλα κατέληγαν ότι ακόμα κι αν κατά την εκτέλεση συμβαίνουν καταρρεύσεις κόμβων, οι αλγόριθμοι εξακολουθούν να είναι αποδοτικοί και αρκετά αποτελεσματικοί ως προς την επίλυση του προβλήματος της πληροφόρησης.

Ο μηχανισμός τερματισμού που χρησιμοποιεί ο αλγόριθμος EARS, τον κάνει πιο ευέλικτο ως προς το ποια στιγμή η πληροφορία θα σταματήσει να μεταδίδεται. Δηλαδή το γεγονός ότι πριν τερματίσει οριστικά λαμβάνει υπόψη του αν μια πληροφορία δεν έχει διαδοθεί ακόμη σε κάποιο γύρο έτσι ώστε να επαναλάβει τη διαδικασία διάδοσής του, τον

κάνει πιο ανεκτικό σε σφάλματα. Αυτό είναι θετικό διότι κάνει τον αλγόριθμο να είναι εύρωστος και αποτελεσματικός. Από την άλλη όμως τον οδηγεί σε μεγαλύτερο χρόνο ολοκλήρωσης. Ωστόσο αυτό το επιπλέον κόστος μπορεί να θεωρηθεί ασήμαντο σε σχέση με την ευρωστία που παρέχει ο αλγόριθμος. Όπως φάνηκε και από την πειραματική αξιολόγηση ακόμα κι αν καταρρέουν κόμβοι στο σύστημα ο αλγόριθμος εξακολουθεί να ενημερώνει τους κόμβους του συστήματος σε λογαριθμικό χρόνο και με λογικό επιπλέον κόστος. Ο αλγόριθμος EARS είναι χωρίς αμφιβολία κατάλληλος για εφαρμογή σε πραγματικά περιβάλλοντα αφού σε πραγματικά δίκτυα τα σφάλματα είναι αναπόφευκτα.

Σε αντίθεση με τον αλγόριθμος EARS, ο αλγόριθμος SEARS χρησιμοποιεί έναν πιο αυστηρό μηχανισμό τερματισμού ο οποίος καθορίζει την ακριβή στιγμή στην οποία θα τερματίσει ο αλγόριθμος. Δε λαμβάνει υπόψη του αν υπάρχει κάποια πληροφορία που να μην έχει διαδοθεί ακόμη σε κάποιο γύρο. Θα μπορούσε να πει κανείς ότι αυτό θα καθιστά τον αλγόριθμο μη εύρωστο αφού σε περίπτωση εμφάνισης σφαλμάτων ο αλγόριθμος δεν θα τα λάβει υπόψη του, θα τερματίσει κανονικά και πιθανότατα να μην προλάβουν να ενημερωθούν πλήρως οι κόμβοι του συστήματος. Υπάρχει η πιθανότητα να συμβεί κι αυτό όμως ο SEARS χρησιμοποιεί έναν επιπλέον μηχανισμό που μειώνει σημαντικά την πιθανότητα να συμβεί κάτι τέτοιο. Σε κάθε γύρο επικοινωνίας δεν επιλέγει τυχαία μόνο ένα κόμβο στον οποίο αποστέλλει ένα μήνυμα αλλά επιλέγει ένα λογαριθμικό αριθμό κόμβων. Έτσι στον ίδιο γύρο ενημερώνονται περισσότεροι από ένας κόμβοι με κάποια επιπλέον πληροφορία, συνεπώς μειώνεται σημαντικά η πιθανότητα να μην ενημερωθεί κάποιος κόμβος. Εξάλλου και να καταρρεύσει κάποιος κόμβος κατά την εκτέλεση, οι πληροφορίες που πρόλαβε να μεταδώσει στο μεταξύ, δεν χάνονται οπότε μπορούν να μεταδοθούν στους υπόλοιπους κόμβους που είναι ακόμη ζωντανοί. Ως αποτέλεσμα αυτού του μηχανισμού όμως είναι να αυξηθεί αρκετά ο αριθμός των μηνυμάτων που διακινούνται στο σύστημα. Ωστόσο αυτό το επιπλέον κόστος μπορεί να θεωρηθεί ασήμαντο σε σχέση με την ευρωστία που παρέχει ο αλγόριθμος. Όπως φάνηκε και από την πειραματική αξιολόγηση, το σημαντικότερο πλεονέκτημα του αλγορίθμου SEARS είναι ότι για μεγάλο αριθμό κόμβων ο αριθμός των γύρων επικοινωνίας παραμένει σχεδόν σταθερός και ο χρόνος εκτέλεσης είναι μικρότερος σε σύγκριση με τον

προηγούμενο αλγόριθμο. Επιπλέον, ακόμα κι αν καταρρέουν κόμβοι στο σύστημα, ο αλγόριθμος εξακολουθεί να ενημερώνει τους κόμβους σε σταθερό χρόνο και με λογικό επιπρόσθετο κόστος. Ο αλγόριθμος SEARS είναι χωρίς αμφιβολία κατάλληλος για εφαρμογή σε πραγματικά περιβάλλοντα αφού σε πραγματικά δίκτυα τα σφάλματα είναι αναπόφευκτα.

Συνοψίζοντας, γενικά είναι δύσκολο ένας αλγόριθμος να προσφέρει όλα τα πλεονεκτήματα που μπορεί να έχει ένας αλγόριθμος πληροφόρησης χωρίς να αυξάνει το κόστος του αλγορίθμου είτε σε χρόνο είτε σε μηνύματα. Ανάλογα, λοιπόν, με τις ανάγκες και απαιτήσεις του κάθε ασύγχρονου κατανεμημένου συστήματος θα πρέπει κάθε φορά να επιλέγεται εκείνος ο αλγόριθμος που το κόστος του δεν ξεπερνά το ελάχιστο κόστος που απαιτεί η εφαρμογή. Για παράδειγμα αν μια κατανεμημένη εφαρμογή απαιτούσε οι κόμβοι της να ενημερώνονται για όλες τις πληροφορίες του δικτύου προκαλώντας την ελάχιστη δυνατή συμφόρηση στο δίκτυο, τότε θα ήταν καλύτερο να χρησιμοποιηθεί ο αλγόριθμος EARS ο οποίος από τη φύση του ελαχιστοποιεί τα μηνύματα που αποστέλλονται κατά τη διάδοση των πληροφοριών στους κόμβους. Μπορεί αυτός να είναι πιο αργός από τον SEARS όμως ολοκληρώνει τη λειτουργία του σε χρόνο λογαριθμικό. Αν όμως, για παράδειγμα, μια κατανεμημένη εφαρμογή απαιτούσε την ενημέρωση μεγάλου αριθμού κόμβων σε όσο το δυνατό μικρότερο χρόνο, χωρίς να υπάρχει μεγάλο πρόβλημα αν στέλλονται περισσότερα μηνύματα κατά τη διάδοση, τότε θα ήταν καταλληλότερη η χρήση του αλγορίθμου SEARS, ο οποίος ενημερώνει τους κόμβους σε χρόνο σταθερό και μικρότερο από τον EARS κι ας αποστέλλει περισσότερα μηνύματα κατά την εκτέλεση.

7.2 Δυσκολίες που παρουσιάστηκαν και πώς ξεπεράστηκαν

Στα πλαίσια της εργασίας αυτής παρουσιάστηκαν διάφορες δυσκολίες τόσο στην υλοποίηση όσο και στην προσομοίωση και εφαρμογή των αλγορίθμων στο PlanetLab οι οποίες ωστόσο ξεπεράστηκαν αλλά χρειάστηκαν αρκετό χρόνο. Οι πρώτες δυσκολίες εμφανίστηκαν όταν η υλοποίηση έπρεπε να γίνει με χρήση της γλώσσας προγραμματισμού

Java. Όντας πτυχιούχος του κλάδου των Μαθηματικών και όχι του κλάδου της Πληροφορικής και παρόλο που υπήρχε προηγούμενη εμπειρία σε προγραμματισμό στη γλώσσα C# και μερική εμπειρία σε αντικειμενοστραφή προγραμματισμό στη γλώσσα C++, η εκμάθηση της γλώσσας Java ήταν κάτι καινούργιο. Έννοιες όπως η κληρονομικότητα και ο πολυμορφισμός, έπρεπε πρώτα να γίνουν κατανοητές για να χρησιμοποιηθούν σωστά στη μετέπειτα υλοποίηση των αλγορίθμων.

Επιπλέον, για την υλοποίηση των αλγορίθμων ήταν αναγκαία η αυτοδίδακτη εκμάθηση της βιβλιοθήκης YALPS. Επειδή η βιβλιοθήκη αυτή έχει δημιουργηθεί πρόσφατα, η μόνη βοήθεια ήταν ένα πρόχειρο εγχειρίδιο χρήσης που είχε αναρτηθεί στην επίσημη σελίδα του YALPS [10] και μια διπλωματική εργασία που ασχολήθηκε επίσης με την υλοποίηση αλγορίθμων μέσω YALPS [2]. Ωστόσο το υλικό αυτό ήταν αρκετά βοηθητικό για να γίνει κατανοητή η βιβλιοθήκη και συνεπώς να γίνει δυνατή η υλοποίηση και η εκτέλεση των αλγορίθμων σε αυτή. Περιλάμβανε επεξήγηση των βασικών μεθόδων της βιβλιοθήκης καθώς και σαφή περιγραφή του τρόπου με τον οποίο μπορεί να εκτελεστεί μία εφαρμογή της YALPS.

Ένα άλλο πρόβλημα που παρουσιάστηκε ήταν κατά την πειραματική αξιολόγηση των αλγορίθμων που υλοποιήθηκαν. Τόσο στην προσομοίωση όσο και στο PlanetLab ο αριθμός των κόμβων που μπορούσαν να χρησιμοποιηθούν ήταν περιορισμένος. Στην προσομοίωση, επειδή γινόταν χρήση εικονικών επεξεργαστών και οι υπολογισμοί ουσιαστικά γίνονταν τοπικά στον ίδιο υπολογιστή, ήταν αδύνατο να εκτελεστούν οι αλγόριθμοι για πολύ μεγάλο αριθμό κόμβων. Τελικά ο μέγιστος αριθμός κόμβων που έγινε κατορθωτό να χρησιμοποιηθούν ήταν 128. Από την άλλη στο PlanetLab, λόγω περιορισμένου επιτρεπτού αριθμού κόμβων στο slice, ο μέγιστος αριθμός κόμβων που ήταν δυνατό να χρησιμοποιηθούν ήταν μόνο 25. Όμως παρόλο που η αξιολόγηση των αλγορίθμων έγινε κάτω από τέτοιους περιορισμούς, εντούτοις δεν εμπόδισε καθόλου την εξαγωγή συμπερασμάτων ως προς την απόδοσή τους. Βέβαια, αν μπορούσαν να χρησιμοποιηθούν περισσότεροι κόμβοι, σίγουρα τα αποτελέσματα της αξιολόγησης θα ήταν πιο ακριβή και πιο γενικά.

Όταν έπρεπε να τρέξουν οι αλγόριθμοι στο PlanetLab παρουσιάστηκε ένα πρόβλημα με την παθητική μέθοδο receive() που χρησιμοποιούσε η εφαρμογή YALPS. Ενώ οι αλγόριθμοι εγκαθίδρυαν ορθά τη σύνδεσή τους με τους υπόλοιπους κόμβους και έστελλαν κανονικά τα μηνύματα στους άλλους κόμβους εντούτοις οι κόμβοι – παραλήπτες δεν έδειχναν να λάμβαναν τα μηνύματα αυτά. Γι' αυτό το λόγο ζητήθηκε βοήθεια από ένα μέλος της ερευνητικής ομάδας του ιδρύματος INRIA. Το πρόβλημα τελικά επιλύθηκε όταν το μέλος αυτό έστειλε τη νέα εκδοχή της YALPS.

7.3 Μελλοντική Εργασία

Οι στόχοι της εργασίας γενικά έχουν επιτευχθεί. Ο μόνος περιορισμός, όπως αναφέρθηκε και πιο πάνω, ήταν στον αριθμό των κόμβων που ήταν δυνατό να χρησιμοποιηθεί για την αξιολόγηση των αλγορίθμων. Και στους δύο αλγόριθμους ο μέγιστος αριθμός κόμβων που χρησιμοποιήθηκαν στην προσομοίωση ήταν μόνο 128 και αυτό ήταν εφικτό με χρήση μηχανής που διέθετε 4GB RAM.

Κατά την πειραματική αξιολόγηση των αλγορίθμων, ανάμεσα στις μετρικές αξιολόγησης που χρησιμοποιήθηκαν ήταν και η μέτρηση της πολυπλοκότητας μηνύματος, δηλαδή γινόταν καταμέτρηση του συνολικού αριθμού μηνυμάτων που αποστέλλονταν κατά την εκτέλεση. Παρόλο που ήταν μια αρκετά καλή μετρική αξιολόγησης, καθώς έδινε χρήσιμα αποτελέσματα για την εξαγωγή συμπερασμάτων ως προς την αποδοτικότητα των αλγορίθμων, εντούτοις γινόταν καταμέτρηση μόνο του συνολικού αριθμού μηνυμάτων που αποστέλλονταν και όχι του συνολικού αριθμού των bits που μεταφέρονταν ο οποίος εξαρτάται από το μέγεθος του μηνύματος. Ο λόγος που προτιμήθηκε ο αριθμός των μηνυμάτων αντί ο αριθμός των bits ήταν διότι ο αριθμός των bits θα περιέγραφε την απόδοση των αλγορίθμων ως προς κάποια συγκεκριμένα μηνύματα και πιθανότατα τα αποτελέσματα που θα λαμβάνονταν να μην ήταν αρκετά αξιόπιστα ούτε θα οδηγούσαν σε πιο γενικά συμπεράσματα. Κάτι τέτοιο όμως θα μπορούσε να πραγματοποιηθεί σε κάποια μελλοντική εργασία η οποία θα τρέξει τους

αλγορίθμους που υλοποιήθηκαν στην εργασία αυτή και θα προσθέσει κι άλλα συμπεράσματα ως προς την απόδοση των αλγορίθμων, λαμβάνοντας υπόψη συγκεκριμένες μετρήσεις των δεδομένων της πληροφορίας που πρέπει να διαδοθεί. Θα ήταν μάλιστα ενδιαφέρον να μελετηθεί κατά πόσο μπορεί να μειωθεί το μέγεθος των μηνυμάτων χωρίς να επηρεαστεί η ορθότητα των αλγορίθμων.

Επίσης, στην εργασία αυτή οι αλγόριθμοι εφαρμόστηκαν στο PlanetLab μόνο για 8, 16 και 25 κόμβους. Κάτι που θα μπορούσε να γίνει ενδεχομένως σε μια μελλοντική εργασία είναι η αξιολόγηση των αλγορίθμων για μεγαλύτερο αριθμό κόμβων στο PlanetLab ώστε να μελετηθούν οι αλγόριθμοι και σε μεγαλύτερα δίκτυα και μάλιστα χρησιμοποιώντας και άλλες μετρικές αξιολόγησης εκτός από την πολυπλοκότητα χρόνου και μηνύματος, όπως είναι για παράδειγμα ο αριθμός των πακέτων – μηνυμάτων που διακινούνται στο δίκτυο ανά δευτερόλεπτο (thoughput) και άλλες παρόμοιες μετρικές δικτύων. Επειδή ο κάθε χρήστης του PlanetLab μπορεί να έχει μέρισμά του μέχρι και 25 κόμβους μόνο πάνω κάτω, θα μπορούσε η αξιολόγηση να γίνει συνδυάζοντας πολλά μερίσματα μαζί.

Τέλος, θα μπορούσε μελλοντικά να μελετηθεί κατά πόσο θα μπορούσαν να βελτιστοποιηθούν τα θεωρητικά άνω όρια για τη διάρκεια της “shut-down” φάσης του κάθε αλγόριθμου στο PlanetLab. Θα μπορούσε κάποιος να ξεκινήσει με τα πειραματικά αποτελέσματα της παρούσας εργασίας στην περίπτωση που οι αλγόριθμοι εφαρμόστηκαν στο PlanetLab και κοιτάζοντας προσεκτικά σε ποιο ακριβώς βήμα ολοκληρώνονται οι αλγόριθμοι, να πειραματιστεί αυξομειώνοντας τα όρια των αλγορίθμων μέχρι να πετύχει εκείνα τα όρια που κάνουν τους αλγόριθμους να εκτελούν όσο το δυνατό λιγότερα βήματα.

Βιβλιογραφία

[1] R. Karp, C. Schindelhauser, S. Shenker, B. Vocking, *"Randomized Rumor Spreading"*, In proc. of the 41st Annual Symposium on Foundations of Computer Science, 2000, pp.565-574.

[2] Χ. Σοφοκλέους, *"Υλοποίηση και Πειραματική Αξιολόγηση κατανεμημένων αλγορίθμων πληροφόρησης με τη χρήση των βιβλιοθηκών GossipLib και Yalps"*, Ατομική Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, 2011.

[3] C. Georgiou, S. Gilbert, R. Guerraoui, D. Kowalski, *"On the Complexity of Asynchronous Gossip"*, in Proc. of ACM PODC 2008, pp. 135–144.

[4] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D.Swinehart, and D. Terry, *"Epidemic algorithms for replicated database maintenance"*, in Proc. of the 6th ACM Symposium on Principles of Distributed Computing (PODC), 1987, pp. 1–12.

[5] R. van Renesse, Y. Minsky, and M. Hayden, *"A gossip-style failure detection service"*, in Proc. of IFIP Int-l Conference on Distributed Systems Platforms and Open Distributed Processing, 1998, pp. 55–70.

[6] I. Gupta, A.M. Kermarrec, and A.J. Ganesh, *"Efficient epidemic-style protocols for reliable and scalable multicast"*, in Proc. of 21st IEEE Symposium on Reliable Distributed Systems (SRDS), 2002, pp. 593–605.

[7] B.S. Chlebus and D.R. Kowalski, *"Robust gossiping with an application to consensus"*, Journal of Computer and System Sciences, 72 (2006), pp. 1262–1281.

[8] D. Kempe, J. Kleinberg, and A. Demers, "*Spatial gossip and resource location protocols"*, Journal of ACM, 51 (2004), pp. 943–967.

[9] Βιβλιοθήκη YALPS, [Online]. Available: http://yalps.gforge.inria.fr/

[10] Πλατφόρμα PlanetLab, [Online]. Available: http://www.planet-lab.eu/

[11] Πρωτόκολλα πληροφόρησης, [Online]. Available: http://en.wikipedia.org/wiki/Gossip_protocol

[12] A.S. Tanenbaum, and M.van Steen, *"Distributed Systems, Principles and Paradigms"*, 2002.

[13] B.S. Chlebus and D.R. Kowalski, "*Time and communication efficient consensus for crash failures"*, in Proc. of 20th International Symposium on Distributed Computing (DISC), 2006, pp. 314–328.

[14] S. Verma and W.T. Ooi, "*Controlling gossip protocol infection pattern using adaptive fanout*", in Proc. of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS), 2005, pp. 665–674.

[15] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "*Randomized gossip algorithms, IEEE Transactions on Information Theory"*, 52(6) (2006), pp. 2508–2530.

[16] A. Pelc, "*Fault-tolerant broadcasting and gossiping in communication networks, Networks"*, 28 (1996), pp. 143–156.

[17] J. Hromkovic, R. Klasing, A. Pelc, P. Ruzika, and W. Unger, *"Dissemination of Information in Communication Networks: Broadcasting, Gossiping, Leader Election, and Fault-Tolerance"*, Springer-Verlag, 2005.

[18] INRIA, [Online]. Available: http://www.inria.fr/

[19] PING, [Online]. Available: http://en.wikipedia.org/wiki/Ping

[20] PING command, [Online]. Available: http://www.computerhope.com/pinghlp.htm

Παράρτημα Α

Κώδικας Αλγορίθμου EARS σε γλώσσα προγραμματισμού JAVA

```
import java.io.IOException;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.Random;
import yalps.NodeID;
import yalps.Task;
import yalps.executor.E_NodeID;
import yalps.launchers.AbstractYalpsNode;

public class MyEarsNode  extends AbstractYalpsNode{
    protected ArrayList<E_NodeID> DestinationList = new ArrayList<E_NodeID>();
    /*List of all rumors known to p. Initially Vp = rumor of p*/
    ArrayList<String> Vp;
    /*Informed List contains pairs (r,q) which mean that rumor r has been sent to process q*/
    ArrayList<RumorProcess> Ip;
    /*List of all process q that does not exist any pair (r,q) in the Ip list*/
    ArrayList<String> Lp;
    /*Sleep Count*/
    int sleep_cnt;
    E_NodeID nodeId;
    Random rand;
```

```
    MyFile output;
    double n, f;
    int timeDelay;
    boolean failure, nodeidFlag;

public void initialize() throws IOException{
    rand = new Random();
    nodeId = (E_NodeID) getCommLayer().getNodeId();
    Vp = new ArrayList<String>();
    Ip = new ArrayList<RumorProcess>();
    Lp = new ArrayList<String>();
    Vp.add(nodeId.toString());
    for (E_NodeID node : DestinationList)
        Lp.add(node.toString());
    sleep_cnt = 0;
    output = new MyFile(nodeId, "MyEarsNode");
    n = (double)DestinationList.size() ;
    f = 1.0;
    nodeidFlag = true;
    failure = false;
}

@Override
public void startNode() {
    System.out.println("[Application] - Starting node.");
    try {
        initialize();
        MyMsgSenderTask MyTask = new MyMsgSenderTask();
```

```
            while(true){
                    if (!failure){
                            timeDelay = rand.nextInt(99) + 2;          //   timeDelay = 2;
                            Thread.currentThread().sleep(0, timeDelay);
                            tm.registerTask(MyTask, tm.getSystemTaskGroup());
                    }
            }
            } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
            } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
            }
}

public class MyMsgSenderTask implements Task{
    MyMsgSenderTask(){          }

@Override
public void run() {
    try {
            if (Lp.isEmpty())
               sleep_cnt++;
            else
               sleep_cnt = 0;

            if (sleep_cnt < 2.0*(n/(n-f)*(StrictMath.log(n)/StrictMath.log(2.0)))){
```

```
                    // Choose a process uniformly at random from [n]
                    E_NodeID neighbor;
                     neighbor = chooseRandomDestination();
                     if (nodeidFlag){
                         RumorProcess rp = new RumorProcess();
                         rp.setRumor(nodeId.toString());
                         rp.setProcess(nodeId.toString());
                         if (!findPairIp(rp))
                               Ip.add(rp);
                          nodeidFlag = false;
                     }
                     // Send Serializable
                     if (!neighbor.toString().equals(nodeId.toString())){
                         MySerializableMSG myMSG  = new MySerializableMSG(Vp, Ip);
                         System.out.println("[Application] - Sending SERIALIZABLE msg [" +
"V, I" + "] to " + neighbor);
                         cl.sendTCP(myMSG, neighbor);
                         //For every r in V(p) do update I(p) with (r,neighbor)/
                         for (int r=0; r<Vp.size(); r++){
                               RumorProcess rp = new RumorProcess();
                               rp.setRumor(Vp.get(r));
                               rp.setProcess(neighbor.getNodeId().toString());
                               if (!findPairIp(rp) ) {
                                        Ip.add(rp);
                               }
                          }
                          updateLpList();
                           printAllLists();
```

```
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

public boolean receive(short header, Object msg, NodeID snd) {
        E_NodeID sender = null;
        try {
            if (failure)       return;
            MySerializableMSG arrivingMSG1 = (MySerializableMSG) msg;
            sender = (E_NodeID) snd;
            System.out.println("[Application] - ***Received MSG: <V,I>" +  " from " + sender);
            System.out.println("[Application] - ***MSG type: STANDARDSERIALIZABLE
header: " + header);
            getOut().println("[Application] - ***Received MSG 1: <V,I>" +  " from " + sender);
            getOut().println("[Application] - ***MSG type 1: STANDARDSERIALIZABLE
header: " + header);
            updateVpList(arrivingMSG1.getRumorVlist());
            updateIpList(arrivingMSG1.getRumorProcessPairList());
            updateLpList();
            printAllLists();
            return true;
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return false;
```

```
    }
}

public void setDestination(String destinationNode) {
    try {
        String[] nodeStrings=destinationNode.split(";");
        for (String node : nodeStrings)
                DestinationList.add((E_NodeID) cl.getNodeIdFromString(node));
     } catch (UnknownHostException e) {
        e.printStackTrace();
     }
}

public E_NodeID chooseRandomDestination(){
    int randnum;
    int size = DestinationList.size();
    randnum = rand.nextInt(size);
    return DestinationList.get(randnum);
}

public void printVp() throws IOException{
    output.out.write("-----------------------------"+'\n');
    output.out.write("Current Vp list of Node :"+nodeId+'\n');
    for(String v : Vp)
        output.out.write("Rumor:"+v+'\n');
        output.out.flush();
}
```

```
public void printIp() throws IOException{
    output.out.write("-----------------------------"+'\n');
    output.out.write("Current Ip list of Node :"+nodeId+'\n');
    for(RumorProcess rp : Ip)
        output.out.write("Pair:<"+rp.getRumor()+" , "+rp.getProcess()+">"+'\n');
        output.out.flush();
    }

public void printLp() throws IOException{
     output.out.write("-----------------------------"+'\n');
     output.out.write("Current Lp list of Node :"+nodeId+'\n');
     for(String l : Lp)
        output.out.write("Process:"+l+'\n');
        output.out.flush();
}

public void printAllLists() throws IOException{
    output.out.write("====================================="+'\n');
    output.out.write("New Information for Node:"+nodeId+'\n');
    printVp();
    printIp();
    printLp();
    output.out.write("MESSAGES: "+msg_cnt+'\n');
    output.out.write("====================================="+'\n');
    output.out.write("TIME: "+total_delays+" msec"+'\n');
    output.out.write("====================================="+'\n');
    output.out.flush();
}
```

```
public void updateVpList(ArrayList<String> V){
    for (int i=0; i<V.size(); i++){
        if (!this.Vp.contains(V.get(i))){
                this.Vp.add(V.get(i));
        }
    }
}

public void updateIpList(ArrayList<RumorProcess> I) throws IOException{
    for (int i=0; i<I.size(); i++){
        if ((!findPairIp(I.get(i))) ){
                this.Ip.add(I.get(i));
        }
    }
}

public void updateLpList() throws IOException{
    while(Lp.size()!=0){
    Lp.remove(Lp.size()-1);
    }
    for (E_NodeID node : DestinationList)
        if (!hasAllRumors(node.toString()))
                Lp.add(node.toString());
}

public boolean findPairIp(RumorProcess rp) throws IOException{
    for (int i=0; i<Ip.size(); i++){
```

```
            if ((Ip.get(i).r).equals(rp.r) && (Ip.get(i).q).equals(rp.q)){
                return true;
            }
        }
        return false;
}

public boolean hasAllRumors(String q){
        int totalRumors = Vp.size();
        for (int i=0; i<Ip.size(); i++){
            for (int rumor=0; rumor<Vp.size(); rumor++){
                if (Ip.get(i).r.equals(Vp.get(rumor)) && Ip.get(i).q.equals(q)){
                    totalRumors--;
                }
            }
            if(totalRumors==0)
                break;
            }
            if(totalRumors==0){
                return true;
            }
            else
                return false;
}
```

Παράρτημα Β

Κώδικας Αλγορίθμου SEARS σε γλώσσα προγραμματισμού JAVA

```
import java.io.IOException;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.Random;
import yalps.NodeID;
import yalps.Task;
import yalps.executor.E_NodeID;
import yalps.launchers.AbstractYalpsNode;

public class MySearsNode  extends AbstractYalpsNode{
    protected ArrayList<E_NodeID> DestinationList = new ArrayList<E_NodeID>(
    /*List of all rumors known to p. Initially Vp = rumor of p*/
    ArrayList<String> Vp;
    /*Informed List contains pairs (r,q) which mean that rumor r has been sent to process q*/
    ArrayList<RumorProcess> Ip;
    /*List of all process q that does not exist any pair (r,q) in the Ip list*/
    ArrayList<String> Lp;
    /*Sleep Count*/
    int sleep_cnt;
    E_NodeID nodeId;
    Random rand;
    MyFile output;
```

```
    double n, E, numOfProcesses;
    int timeDelay;
    boolean failure, nodeidFlag;

public void initialize() throws IOException{
    rand = new Random();
    nodeId = (E_NodeID) getCommLayer().getNodeId();
    Vp = new ArrayList<String>();
    Ip = new ArrayList<RumorProcess>();
    Lp = new ArrayList<String>();
    Vp.add(nodeId.toString());
    for (E_NodeID node : DestinationList)
        Lp.add(node.toString());
        sleep_cnt = 0;
        output = new MyFile(nodeId, "MySearsNode");
        n = (double)DestinationList.size() ;//+ 1.0;
        E = 1.0/100.0;
        numOfProcesses = StrictMath.max(StrictMath.pow(n, E),
1.0)*(StrictMath.log(n)/StrictMath.log(2.0));
        nodeidFlag = true;
        failure = false;
}

@Override
public void startNode() {
    System.out.println("[Application] - Starting node.");
    try {
        initialize();
```

```
            MyMsgSenderTask MyTask = new MyMsgSenderTask();
            while(true){
                    if (!failure){
                            timeDelay = rand.nextInt(99) + 2;          //  timeDelay = 2;
                            Thread.currentThread().sleep(0, timeDelay);
                           tm.registerTask(MyTask, tm.getSystemTaskGroup());
                    }
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
}

public class MyMsgSenderTask implements Task{
    MyMsgSenderTask(){ }
    @Override
    public void run() {
     try {
        if (Lp.isEmpty())
            sleep_cnt++;
        else
            sleep_cnt = 0;
        if (sleep_cnt <= 1 ){
            for (int i=0; i<2*numOfProcesses; i++){
```

```
                    // Choose a process uniformly at random from [n]
                    E_NodeID neighbor;
                    neighbor = chooseRandomDestination();
                    if (nodeidFlag){
                        RumorProcess rp = new RumorProcess();
                        rp.setRumor(nodeId.toString());
                        rp.setProcess(nodeId.toString());
                        if (!findPairIp(rp))
                                Ip.add(rp);
                        nodeidFlag = false;
                    }
                    // Send Serializable
                    if (!neighbor.toString().equals(nodeId.toString())){
                        MySerializableMSG myMSG  = new MySerializableMSG(Vp, Ip);
                        System.out.println("[Application] - Sending SERIALIZABLE msg ["
+ "V, I" /*msg1.msg*/ + "] to " + neighbor);
                        cl.sendTCP(myMSG, neighbor);
                        //For every r in V(p) do update I(p) with (r,neighbor)/
                        for (int r=0; r<Vp.size()/*myMSG.getRumorVlist().size()*/; r++){
                            RumorProcess rp = new RumorProcess();
                            rp.setRumor(Vp.get(r)/*myMSG.getRumorVlist().get(r)*/);
                            rp.setProcess(neighbor.getNodeId().toString());
                            if (!findPairIp(rp)) {
                                Ip.add(rp);
                            }
                        }
                        updateLpList();
                        printAllLists();
```

```
                    }
                }
            }
             /*-------------------------------------------------------------*/
            /*Random Node Failure with probability = failures/bound*/
            /*double fail = Math.random();
            double rounds = n/(E*(n-f))*timeDelay;
            if (fail < f/(n*rounds)){
                failure = true;
            }*/
            /*-------------------------------------------------------------*/
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public boolean receive(short header, Object msg, NodeID snd) {
    E_NodeID sender = null;
    try {
    if (failure)   return true;
            MySerializableMSG arrivingMSG1 = (MySerializableMSG) msg;
            sender = (E_NodeID) snd;
            System.out.println("[Application] - ***Received MSG: <V,I>" +
/*arrivingMSG1.msg +*/ " from " + sender);
            System.out.println("[Application] - ***MSG type: STANDARDSERIALIZABLE
header: " + header);
```

```
            getOut().println("[Application] - ***Received MSG 1: <V,I>" +
/*arrivingMSG1.msg +*/ " from " + sender);
            getOut().println("[Application] - ***MSG type 1: STANDARDSERIALIZABLE
header: " + header);
            updateVpList(arrivingMSG1.getRumorVlist());
            updateIpList(arrivingMSG1.getRumorProcessPairList());
            updateLpList();
            printAllLists();
            return true;
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return false;
        }
}

public void setDestination(String destinationNode) {
    try {
            String[] nodeStrings=destinationNode.split(";");
            for (String node : nodeStrings)
                if (!node.toString().equals(getCommLayer().getNodeId().toString()))
                    DestinationList.add((E_NodeID) cl.getNodeIdFromString(node));
            } catch (UnknownHostException e) {
                    e.printStackTrace();
            }
}
```

```
public E_NodeID chooseRandomDestination(){
    int rand;
    int size = DestinationList.size();
    rand = new Random().nextInt(size);
    return DestinationList.get(rand);
}

public void printVp() throws IOException{
    output.out.write("----------------------------"+'\n');
    output.out.write("Current Vp list of Node :"+nodeId+'\n');
    for (String v : Vp)
        output.out.write("Rumor:"+v+'\n');
    output.out.flush();
}

public void printIp() throws IOException{
    output.out.write("----------------------------"+'\n');
    output.out.write("Current Ip list of Node :"+nodeId+'\n');
    for(RumorProcess rp : Ip)
        output.out.write("Pair:<"+rp.getRumor()+" , "+rp.getProcess()+">"+'\n');
    output.out.flush();
}

public void printLp() throws IOException{
    output.out.write("----------------------------"+'\n');
    output.out.write("Current Lp list of Node :"+nodeId+'\n');
    for(String l : Lp)
        output.out.write("Process:"+l+'\n');
```

```
        output.out.flush();
    }

    public void printAllLists() throws IOException{
        output.out.write("==================================="+'\n');
        output.out.write("New Information for Node:"+nodeId+'\n');
        printVp();
        printIp();
        printLp();
    }

    public void updateVpList(ArrayList<String> V){
        for (int i=0; i<V.size(); i++){
            if (!this.Vp.contains(V.get(i))){
                    this.Vp.add(V.get(i));
            }
        }
    }

    public void updateIpList(ArrayList<RumorProcess> I) throws IOException{
        for (int i=0; i<I.size(); i++){
            if (!findPairIp(I.get(i))){
                this.Ip.add(I.get(i));
            }
        }
    }
```

```
public void updateLpList() throws IOException{
    while(Lp.size()!=0){
        Lp.remove(Lp.size()-1);
    }
    for (E_NodeID node : DestinationList)
        if (!hasAllRumors(node.toString()))
                Lp.add(node.toString());
}

public boolean findPairIp(RumorProcess rp) throws IOException{
    for (int i=0; i<Ip.size(); i++){
        if ((Ip.get(i).r).equals(rp.r) && (Ip.get(i).q).equals(rp.q)){
                return true;
        }
    }
    return false;
}

public boolean hasAllRumors(String q){
    int totalRumors = Vp.size();
    for (int i=0; i<Ip.size(); i++){
        for (int rumor=0; rumor<Vp.size(); rumor++){
            if (Ip.get(i).r.equals(Vp.get(rumor)) && Ip.get(i).q.equals(q)){
                totalRumors--;
            }
        }
        if(totalRumors==0)
                break;
```

```
            }
        if(totalRumors==0){
            return true;
        }
        else
            return false;
        }
}
```

Παράρτημα Γ

Παραδείγματα Εκτέλεσης Αλγορίθμου EARS

Παράδειγμα Αρχείου Διαμόρφωσης Αλγορίθμου EARS

node.CLASS=MyEarsNode

node.outFileName=logfiles/MyEarsNode10000_app.log

node.rnd='node'

node.bootStrap=false

node.nylon=false

node.destination=127.0.0.1:10000;127.0.0.1:10001;127.0.0.1:10002;127.0.0.1:10003;127.0.0.1:10004;127.0.0.1:10005;127.0.0.1:10006;127.0.0.1:10007;

executor.cheapDeserializationMap=MyDeserializationMap

executor.virtualAddress=127.0.0.1

executor.port=10000

executor.debug=true

executor.tokenbucket=true

executor.burstSize=1000000

executor.rnd='comm'

executor.lossRate=0

executor.maxQueueLength=10000000

executor.maxSendDelay=0

executor.uploadBandwidth=0

Αποτελέσματα εκτέλεσης ενός κόμβου μετά την εφαρμογή του αλγορίθμου EARS σε ένα δίκτυο από 8 κόμβους χωρίς καμιά κατάρρευση κόμβου

New Information for Node:[127.0.0.1,10000]

Current Vp list of Node :[127.0.0.1,10000]

Rumor:[127.0.0.1,10000]

Rumor:[127.0.0.1,10007]

Rumor:[127.0.0.1,10006]

Rumor:[127.0.0.1,10002]

Rumor:[127.0.0.1,10001]

Rumor:[127.0.0.1,10003]

Rumor:[127.0.0.1,10004]

Rumor:[127.0.0.1,10005]

Current Ip list of Node :[127.0.0.1,10000]

Pair:<[127.0.0.1,10000] , [127.0.0.1,10000]>

Pair:<[127.0.0.1,10000] , [127.0.0.1,10006]>

Pair:<[127.0.0.1,10000] , [127.0.0.1,10001]>

Pair:<[127.0.0.1,10000] , [127.0.0.1,10003]>

Pair:<[127.0.0.1,10000] , [127.0.0.1,10007]>

Pair:<[127.0.0.1,10000] , [127.0.0.1,10002]>

Pair:<[127.0.0.1,10000] , [127.0.0.1,10005]>

Pair:<[127.0.0.1,10000] , [127.0.0.1,10004]>

Pair:<[127.0.0.1,10007] , [127.0.0.1,10007]>

Pair:<[127.0.0.1,10007] , [127.0.0.1,10001]>

Pair:<[127.0.0.1,10007] , [127.0.0.1,10006]>

Pair:<[127.0.0.1,10007] , [127.0.0.1,10005]>

.

.

.

Current Lp list of Node :[127.0.0.1,10000]

==========

Αποτελέσματα εκτέλεσης τριών κόμβων μετά την εφαρμογή του αλγορίθμου EARS σε ένα δίκτυο από 8 κόμβους με 2 από αυτούς να έχουν καταρρεύσει κατά την εκτέλεση

Κόμβος 1

New Information for Node:[127.0.0.1,10006]

Current Vp list of Node :[127.0.0.1,10006]

Rumor:[127.0.0.1,10006]

Rumor:[127.0.0.1,10002]

Rumor:[127.0.0.1,10005]

Rumor:[127.0.0.1,10004]

Rumor:[127.0.0.1,10001]

Rumor:[127.0.0.1,10007]

Rumor:[127.0.0.1,10000]

Current Ip list of Node :[127.0.0.1,10006]

Pair:<[127.0.0.1,10006] , [127.0.0.1,10006]>

Pair:<[127.0.0.1,10006] , [127.0.0.1,10004]>

Pair:<[127.0.0.1,10006] , [127.0.0.1,10000]>

.

.

.

Current Lp list of Node :[127.0.0.1,10006]

==========

Κόμβος 2

New Information for Node:[127.0.0.1,10000]

Current Vp list of Node :[127.0.0.1,10000]

Rumor:[127.0.0.1,10000]

Rumor:[127.0.0.1,10001]

Rumor:[127.0.0.1,10002]

Rumor:[127.0.0.1,10007]

Rumor:[127.0.0.1,10004]

Rumor:[127.0.0.1,10005]

Current Ip list of Node :[127.0.0.1,10000]

Pair:<[127.0.0.1,10000] , [127.0.0.1,10000]>

Pair:<[127.0.0.1,10000] , [127.0.0.1,10001]>

Pair:<[127.0.0.1,10000] , [127.0.0.1,10003]>

.

.

.

Current Lp list of Node :[127.0.0.1,10000]

==========

Κόμβος 3

New Information for Node:[127.0.0.1,10007]

Current Vp list of Node :[127.0.0.1,10007]

Rumor:[127.0.0.1,10007]

Rumor:[127.0.0.1,10004]

Rumor:[127.0.0.1,10001]

Rumor:[127.0.0.1,10005]

Rumor:[127.0.0.1,10002]

Rumor:[127.0.0.1,10000]

Current Ip list of Node :[127.0.0.1,10007]

Pair:<[127.0.0.1,10007] , [127.0.0.1,10007]>

Pair:<[127.0.0.1,10007] , [127.0.0.1,10003]>

Pair:<[127.0.0.1,10007] , [127.0.0.1,10000]>

.

.

.

Current Lp list of Node :[127.0.0.1,10007]

==========