

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ



Πτυχιακή εργασία

Μελέτη προβλημάτων τοποθέτησης σε ασύρματα δίκτυα αισθητήρων

Κωνσταντίνος Ξυπολιτόπουλος

Επιβλέπων: Κωνσταντίνος Οικονόμου

Κέρκυρα

Φεβρουάριος 2019

Επιβλέπων

Κωνσταντίνος Οικονόμου, Αναπληρωτής Καθηγητής
Ιόνιο Πανεπιστήμιο

Τριμελής Επιτροπή

Μιχαήλ Στεφανιδάκης, Επίκουρος Καθηγητής
Ιόνιο Πανεπιστήμιο

Θεόδωρος Ανδρόνικος, Αναπληρωτής Καθηγητής
Ιόνιο Πανεπιστήμιο

Κωνσταντίνος Οικονόμου, Αναπληρωτής Καθηγητής
Ιόνιο Πανεπιστήμιο

Στους γονείς μου, Άρη και Λίτσα, και στην αδελφή μου,
Μορφούλα.

Περίληψη

Τα ασύρματα δίκτυα αισθητήρων είναι μια περιοχή με μεγάλο ερευνητικό ενδιαφέρον από διάφορες απόψεις, σε αυτή την εργασία θα μελετήσουμε προβλήματα βέλτιστης τοποθέτησης μιας υπηρεσίας σε ασύρματα δίκτυα και επίσης την ενεργειακή συμπεριφορά αυτών των δικτύων. Ξεκινάμε με μια γρήγορη επισκόπηση της θεωρίας των γράφων, αφού άλλωστε ένα δίκτυο μπορεί να αντιπροσωπευθεί από ένα γράφο με τις συνδέσεις του ως ακμές του γράφου, και τα σημεία συνδέσεων (υπολογιστές, αισθητήρες) ως τις κορυφές του γράφου. Συνεχίζουμε παρουσιάζοντας το 1-median πρόβλημα και τι πραγματεύεται σε σχέση με την τοποθέτηση μίας υπηρεσίας πάνω στο δίκτυο, δείχνουμε με αποτελέσματα προσομοιώσεων την χρησιμότητα της λύσης του 1-median προβλήματος σε ένα δίκτυο αναφορικά με το συνολικό κόστος επικοινωνίας των κόμβων με την υπηρεσία. Έπειτα αναλύουμε γιατί η λύση του 1-median προβλήματος που χρειάζεται γενική γνώση της τοπολογίας του δικτύου δεν είναι βιώσιμη στρατηγική σε ασύρματα δίκτυα αισθητήρων που μπορεί να έχουμε χιλιάδες κόμβους και συνεχείς αλλαγές στην τοπολογία (μετακινήσεις, προσθήκες και αφαιρέσεις αισθητήρων από το δίκτυο), και υλοποιούμε μια λειτουργία μετεγκατάστασης της υπηρεσίας σε καλύτερες τοποθετήσεις δυναμικά, με μόνο τοπική πληροφορία σαν το κριτήριο της μετακίνησης. Αφότου υλοποιήσουμε την λειτουργία μετεγκατάστασης, αναλύουμε την συμπεριφορά της σε διάφορες τοπολογίες μέσω προσομοιώσεων. Τελικά, ρίχνουμε μια ματιά στο πώς η ενέργεια καταναλώνεται μέσα στο δίκτυο, και στο πρόβλημα της ενεργειακής οπής γύρω από τον κόμβο υπηρεσίας, και δείχνουμε ότι η βέλτιστη τοποθέτηση ενεργειακά μιλώντας βρίσκεται στην θέση της λύσης του 1-median προβλήματος. Τελειώνουμε με προσομοιώσεις μετεγκατάστασης της υπηρεσίας με διάφορες παραμέτρους καθώς μελετάμε την ενεργειακή συμπεριφορά ολόκληρου του δικτύου, και δίνουμε σκέψεις για μελλοντικές κατευθύνσεις του θέματος.

Λέξεις κλειδιά: θεωρία γράφων, p-median πρόβλημα, πρόβλημα ενεργειακής οπής, μετεγκατάσταση υπηρεσίας, omnet++, προσομοιώσεις δικτύων

Abstract

Wireless sensor networks is an area with a lot of research interest, in this paper we will study problems of optimally placing a service in wireless sensor networks and also the energy behavior of these sensor networks. We begin with a quick overview of some basic graph theory, since a graph is a nice theoretical representation for a network. We move on by presenting the 1-median problem and its involvement in terms of the placing of a service over the network. We show with software simulation results the usefulness of the solution of the 1-median problem in a network regarding the total cost of communication with the service node. Then we analyze why the solution of the 1-median problem requiring general knowledge of network topology is not a viable strategy in wireless sensor networks that can have thousands of nodes and constant changes in topology (node additions, removals, and moving of nodes) and we implement a service relocation mechanism to dynamically move locations, with only local information as the mechanism criterion. Once we implement the so-called migration, we analyze its behavior in various network topologies through simulations. Finally, we take a look at how energy is consumed in the network and at the energy hole problem, and we show that the optimal position for energy starved networks is at the position of the solution of the 1-median problem. We finish off with service migration experiments with various parameters as we study the energy behavior of the entire network, and give thoughts on future directions for the subject.

Keywords: graph theory, p-median problem, energy hole problem, migration, omnet++, network simulation

Πρόλογος και Ευχαριστίες

Η παρούσα πτυχιακή εργασία εκπονήθηκε κατά τα έτη 2018-2019 στο τμήμα Πληροφορικής του Ιονίου Πανεπιστημίου και υποβάλλεται για την συμπλήρωση των προϋποθέσεων απόκτησης του πτυχίου τριτοβάθμιας εκπαίδευσης.

Αρχικά, θα ήθελα να ευχαριστήσω τον κο. Οικονόμου για την μεγάλη βοήθεια και καθοδήγηση του τα τελευταία αυτά δύο χρόνια. Επίσης, ευχαριστώ τους γονείς μου Αριστείδη και Ευαγγελία, την αδελφή μου Ευμορφία, την γιαγιά μου Βέτα που πάντα με κρατάει στις προσευχές της, και την ευρύτερη οικογένεια μου για την αγάπη και στήριξη τους. Ευχαριστώ την οικογένεια Μπίκου για την υποστήριξη των προσπαθειών μου, και τέλος, ευχαριστώ τους φίλους μου, Βαγγέλη, Γεωργία, Θεοδόση, Σαράντη, Μιχάλη, Κοσμά και Bryan.

Κέρκυρα, Φεβρουάριος 2019
Κωνσταντίνος Ξυπολιτόπουλος

Περιεχόμενα

A' Εισαγωγή	1
B' Επισκόπηση Θεωρίας Γράφων	3
B'.1 Γράφοι	3
B'.1.1 Γραφική αναπαράσταση	3
B'.1.2 Γείτονες	4
B'.1.3 Βάρη	4
B'.1.4 Κατεύθυνση	5
B'.2 Μονοπάτια γράφων	5
B'.2.1 Συνεκτικότητα	6
B'.2.2 Ελάχιστα μονοπάτια	6
B'.3 Δέντρα	6
B'.3.1 Κύκλοι	6
B'.3.2 Ρίζα	7
B'.3.3 Γονείς	7
B'.3.4 Παιδιά	8
B'.3.5 Απόγονοι	8
B'.4 Δέντρα επικάλυψης	8
B'.4.1 Ελάχιστα δέντρα επικάλυψης	9
B'.5 Τυχαίοι γεωμετρικοί γράφοι	9
B'.6 Πλεγματικές τοπολογίες	10
Γ' Μετεγκατάσταση Υπηρεσίας	13
Γ'.1 Κόστος τοποθεσίας	13
Γ'.2 Το 1-median πρόβλημα	13
Γ'.3 Στοιχεία υπηρεσίας	16
Γ'.4 Μετεγκατάσταση	16

Γ'.5 Το κριτήριο μετεγκατάστασης	16
Γ'.6 Μελέτη της πολιτικής μετεγκατάστασης	17
Γ'.6.1 Σε γεωμετρικούς τυχαίους γράφους	17
Γ'.6.2 Σε πλεγματική τοπολογία	19
Δ' Ενεργειακή Μελέτη Ασύρματων Δικτύων Αισθητήρων	23
Δ'.1 Ασύρματα δίκτυα αισθητήρων	23
Δ'.1.1 MICAz	24
Δ'.2 Το πρόβλημα της ενεργειακής οπής	24
Δ'.3 Στατική τοποθέτηση υπηρεσίας	26
Δ'.3.1 Μέσος όρος εναπομείνουσας ενέργειας	28
Δ'.3.2 Χρόνος "θανάτου" δικτύου	28
Δ'.3.3 Διακύμανση εναπομείνουσας ενέργειας	28
Δ'.3.4 Κανονικοποιημένα αποτελέσματα	30
Δ'.4 Συμβάντα	31
Δ'.4.1 Στατικά συμβάντα	31
Δ'.4.2 Κινούμενα συμβάντα	31
Δ'.5 Συμβάντα και μετεγκατάσταση υπηρεσίας	36
Δ'.6 Σύγκριση χρονισμών μετεγκατάστασης	38
Δ'.7 Ανοιχτά ερωτήματα για μελλοντική πορεία	40
Ε' Συμπεράσματα	43

Κατάλογος σχημάτων

B'.1	Παράδειγμα γράφου G με $V = \{\text{Κώστας, Bryan, Γιάννης}\}$ και $E = \{\{\text{Κώστας, Bryan}\}, \{\text{Κώστας, Κοσμάς}\}, \{\text{Κοσμάς, Bryan}\}\}$	4
B'.2	Παράδειγμα ελάχιστου μονοπατιού από την κορυφή A στην κορυφή E . Χωρίς βάρη $\{A, E\}$, και με βάρη $\{A, B, \Gamma, E\}$	7
B'.3	Παράδειγμα δέντρου.	9
B'.4	Παράδειγμα τυχαίου γεωμετρικού γράφου.	10
B'.5	Παράδειγμα μη συνεκτικού τυχαίου γεωμετρικού γράφου.	11
B'.6	Παράδειγμα γράφου πλεγματικής τοπολογίας.	11
Γ'.1	Παράδειγμα εύρεσης κόστους ενός κόμβου στο δίκτυο (χρωματισμένος). Στους κόμβους μέσα αναγράφεται η απαίτηση υπηρεσίας, στον κεντρικό κόμβο που στεγάζεται η υπηρεσία, αναγράφεται το συνολικό κόστος που υπολογίστηκε.	14
Γ'.2	Παράδειγμα λύσης του 1-median (ο χρωματισμένος κόμβος). Στους κόμβους μέσα αναγράφεται το C_y όταν τοποθετούμε την υπηρεσία στον συγκεκριμένο κόμβο, δίνονται οι απαιτήσεις υπηρεσίας των κόμβων και τα βάρη των ακμών.	15
Γ'.3	Διάγραμμα μεταβολής του κόστους τοποθεσίας της υπηρεσίας κατά την πάροδο του χρόνου σε τυχαίο γεωμετρικό γράφο 500 κόμβων.	18
Γ'.4	Διάγραμμα μεταβολής του κόστους τοποθεσίας της υπηρεσίας 5 διαφορετικών πειραμάτων με διαφορετικές θέσεις εκκίνησης κατά την πάροδο του χρόνου σε όμοιο τυχαίο γεωμετρικό γράφο 500 κόμβων.	19
Γ'.5	Διάγραμμα μεταβολής του κόστους τοποθεσίας της υπηρεσίας κατά την πάροδο του χρόνου σε τετράγωνο πλεγματικό δίκτυο διαστάσεων 25 επί 25.	20

Γ'.6	Διάγραμμα μεταβολής του κόστους τοποθεσίας της υπηρεσίας κατά την πάροδο του χρόνου σε τετράγωνο πλεγματοτικό δίκτυο διαστάσεων 25 επί 25 100 διαφορετικών πειραμάτων στο ίδιο σύστημα συντεταγμένων.	21
Γ'.7	Διάγραμμα μεταβολής του κόστους τοποθεσίας της υπηρεσίας κατά την πάροδο του χρόνου σε ορθογώνιο πλεγματοτικό δίκτυο διαστάσεων 100 επί 4.	21
Δ'.1	Σχηματική αναπαράσταση του προβλήματος της ενεργειακής οπής με χρωματικό κώδικα για τα επίπεδα της εναπομείνουσας ενέργειας την στιγμή του θανάτου του δικτύου.	25
Δ'.2	Μέσος όρος εναπομείνουσας ενέργειας μπαταρίας στην στιγμή θανάτου του δικτύου σε σύγκριση με την απόσταση της αρχικής τοποθέτησης από τον κόμβο της λύσης του 1-median.	27
Δ'.3	Χρόνος θανάτου δικτύου σε συνάρτηση με την απόσταση της αρχικής τοποθέτησης από τον κόμβο της λύσης του 1-median.	29
Δ'.4	Διακύμανση επιπέδων μπαταρίας σε όλο το δίκτυο στην στιγμή θανάτου σε σύγκριση με την απόσταση της αρχικής τοποθέτησης από τον κόμβο της λύσης του 1-median.	29
Δ'.5	Κανονικοποιημένα αποτελέσματα διακύμανση εναπομείνουσας ενέργειας και μέσου όρου εναπομείνουσας ενέργειας.	30
Δ'.6	Σχηματική απεικόνιση μιας περιοχής συμβάντος, φαίνεται η ακτίνα r και το κέντρο του κύκλου που ορίζει την περιοχή, και οι πιθανότητες δημιουργίας πακέτου ανά χρονική στιγμή του κάθε κόμβου.	32
Δ'.7	(Στατικό συμβάν) Μέσος όρος μπαταρίας την στιγμή θανάτου του δικτύου.	32
Δ'.8	(Στατικό συμβάν) Διακύμανση επιπέδων μπαταρίας σε όλο το δίκτυο την στιγμή θανάτου.	32
Δ'.9	(Στατικό συμβάν) Χρόνος θανάτου του δικτύου.	33
Δ'.10	(Στατικό συμβάν) Κανονικοποιημένα αποτελέσματα διακύμανσης και μέσου όρου εναπομείνουσας ενέργειας.	33
Δ'.11	Μέσος όρος επιπέδων μπαταρίας σε όλο το δίκτυο στην στιγμή θανάτου σε σύγκριση με την απόσταση της αρχικής τοποθέτησης από τον κόμβο της λύσης του 1-median για στατικό (ακίνητο) και δυναμικό (κινούμενο) συμβάν.	33

Δ'.12 Διακύμανση επιπέδων μπαταρίας σε όλο το δίκτυο στην στιγμή θανάτου σε σύγκριση με την απόσταση της αρχικής τοποθέτησης από τον κόμβο της λύσης του 1-median για στατικό (ακίνητο) και δυναμικό (κινούμενο) συμβάν.	34
Δ'.13 Στιγμή θανάτου σε σύγκριση με την απόσταση της αρχικής τοποθέτησης από τον κόμβο της λύσης του 1-median για στατικό (ακίνητο) και δυναμικό (κινούμενο) συμβάν.	35
Δ'.14 (Χωρίς μετεγκατάσταση) Μέσος όρος μπαταρίας.	37
Δ'.15 (Με μετεγκατάσταση) Μέσος όρος μπαταρίας.	37
Δ'.16 (Χωρίς μετεγκατάσταση) Διακύμανση επιπέδων μπαταρίας.	37
Δ'.17 (Με μετεγκατάσταση) Διακύμανση επιπέδων μπαταρίας.	37
Δ'.18 (Χωρίς μετεγκατάσταση) Χρονική στιγμή θανάτου.	38
Δ'.19 (Με μετεγκατάσταση) Χρονική στιγμή θανάτου.	38
Δ'.20 Χρόνος θανάτου δικτύου σε δευτερόλεπτα με τον μηχανισμό μετεγκατάστασης ενεργοποιημένο και απενεργοποιημένο σε κόκκινο και μπλε αντίστοιχα.	39
Δ'.21 Χρόνος θανάτου δικτύου σε δευτερόλεπτα με τον χρονισμό του μηχανισμού μετεγκατάστασης στα 400 δευτερόλεπτα και 5,000 δευτερόλεπτα.	40
Δ'.22 Χρόνος θανάτου δικτύου σε δευτερόλεπτα με τον χρονισμό του μηχανισμού μετεγκατάστασης στα 100 δευτερόλεπτα και 10,000 δευτερόλεπτα.	41

Κεφάλαιο Α΄

Εισαγωγή

Τα δίκτυα είναι η ραχοκοκαλιά της σύγχρονης κοινωνίας της πληροφορίας, αλλά τα δίκτυα στην πράξη είναι καταδικασμένα να μένουν στο υπόστρωμα άλλων τεχνολογιών, έχοντας δευτερεύων υποστηρικτικό ρόλο, όσο καλύτερα κρυμμένες είναι οι εσωτερικές λειτουργίες ενός δικτύου θα λέγαμε ότι είναι και επιτυχές. Αλλά σαν την βάση πολλών συστημάτων τα δίκτυα χρήζουν συνεχής ανάπτυξης και εξέλιξης για να δίνουν τις δυνατότητες σε αυτά ακριβώς τα ανώτερα στρώματα να έχουν την ελευθερία στην ταχύτητα, καθυστερήσεις και συνδεσιμότητα που χρειάζονται για να λειτουργήσουν και να εξελιχθούν και αυτά.

Ένας τομέας ερευνητικού και πρακτικού ενδιαφέροντος είναι τα ασύρματα δίκτυα αισθητήρων, πολλά προγράμματα επιστημονικού και στρατιωτικού ενδιαφέροντος έχουν χρηματοδοτηθεί για την ανάπτυξη τέτοιων δικτύων και την εφαρμογή τους. Τα ασύρματα δίκτυα αισθητήρων είναι δίκτυα δομημένα από πολύ μικρούς αυτόνομους κόμβους που συνήθως διασκορπίζονται σε μια μεγάλη γεωγραφικά περιοχή ενδιαφέροντος για μέτρηση περιβαλλοντολογικών τιμών και παρακολούθηση συμβάντων. Τέτοιου τύπου δίκτυα φέρνουν τις δικές τους δυσκολίες, όπως τον τρόπο δημιουργίας της τοπολογίας του δικτύου, την επικοινωνία και την δρομολόγηση πληροφορίας από το δίκτυο στον εξωτερικό κόσμο, την ενεργειακή απόδοση και συμπεριφορά του δικτύου, και άλλα. Σε αυτή την πτυχιακή εργασία θα αναλύσουμε κάποια από αυτά τα προβλήματα σε διάφορες καταστάσεις και τοπολογίες ασυρμάτων δικτύων αισθητήρων.

Κώδικας

Ο κώδικας που υλοποιήθηκε για την παρούσα πτυχιακή εργασία χρησιμοποιήσε το εργαλείο ελέγχου εκδόσεων κώδικα "git" και μπορεί να βρεθεί ανοιχτά στην

ιστοσελίδα αποθετηρίων git "GitHub" στα παρακάτω τρία σημεία:

<https://github.com/ConstantineXipol/ssm-testing>

<https://github.com/ConstantineXipol/ssm-trees>

<https://github.com/ConstantineXipol/ssm-notebook>

Το αποθετήριο ssm-testing είναι το κύριο και περιέχει τέσσερις διακλαδώσεις του κυρίου κώδικα (git branches), μία για το κάθε κύριο μέρος της πτυχιακής (Migration, Batteries, Events, master). Το αποθετήριο ssm-trees είναι η αρχική δουλειά της πτυχιακής που είναι βασισμένη σε ένα πρωτότυπο μετεγκατάστασης που δουλεύει πάνω σε τοπολογίες δέντρου μόνο. Και τελικά το αποθετήριο ssm-notebook είναι για τον κώδικα παραγωγής όλων των διαγραμμάτων της εργασίας, γραμμένο σε python στο περιβάλλον jupyter notebook. Ο κώδικας επίσης μπορεί να βρεθεί σε εκτυπωμένη μορφή στο παράρτημα της εργασίας χωρισμένος στα αρχεία που τον αποτελούν.

Κεφάλαιο Β'

Επισκόπηση Θεωρίας Γράφων

Β'.1 Γράφοι

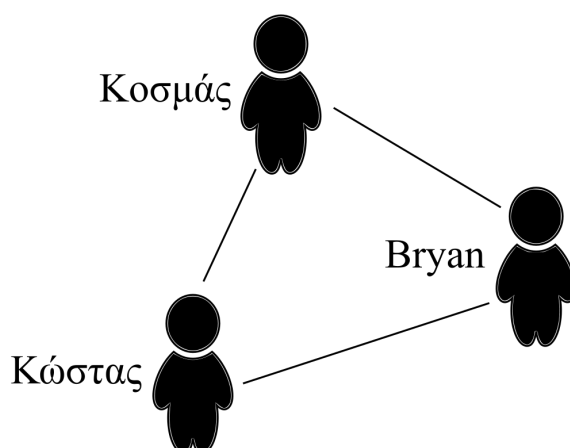
Φανταστείτε τον εαυτό σας και τους φίλους σας σαν ένα δίκτυο από ανθρώπους που ενώνονται με δεσμούς φιλίας, αυτό το αποκαλούμενο κοινωνικό δίκτυο μπορεί όταν το κοιτάξετε συνολικά, να περιέχει άτομα που δεν είναι φίλοι με σας συγκεκριμένα, αλλά μπορεί κάποιος δικός σας φίλος να είναι φίλος αυτών, ή ακόμη και ένας φίλος φίλου να είναι φίλος τους κοκ. Αυτή η δομή που μόλις περιγράψαμε είναι ένας γράφος (επίσης ονομάζεται γράφημα/γραφήματα, αλλά θα χρησιμοποιούμε την λέξη γράφος για να αποφύγουμε τυχών παρανοήσεις για παράδειγμα με γραφήματα των συναρτήσεων), οι γράφοι περιγράφουν τις συνδέσεις μεταξύ διαφόρων εννοιών-οντοτήτων (πχ. άνθρωποι - κοινωνικός γράφος, υπολογιστές - γράφος του διαδικτύου), δηλαδή ποιος συνδέεται με ποιόν και πως. Πηγές για το κεφάλαιο είναι τα βιβλία διακριτών μαθηματικών Liu [5] και Κολουντζάκης [12].

Ένας γράφος $G = (V, E)$ αποτελείται από ένα σύνολο κορυφών V (πχ. άνθρωποι) και ένα σύνολο ακμών E (πχ. μια φιλία μεταξύ δύο ανθρώπων), το οποίο είναι ένα σύνολο αποτελούμενο από πολλές πλειάδες (σύνολα δύο στοιχείων - δι-σύνολα) του V , που αντιπροσωπεύει την σύνδεση των δύο. Γράφουμε $V = V(G)$ και $E = E(G)$ για τα σύνολα των κορυφών και ακμών του γράφου G αντίστοιχα.

Β'.1.1 Γραφική αναπαράσταση

Συνήθως πολλοί γράφοι παριστάνονται γραφικά ως ένα σύνολο τελειών που αντιπροσωπεύουν τις κορυφές V , και αυτές τις τελείες ενώνουν ευθείες γραμμές ή καμπύλες γραμμές ανάλογα με το αν υπάρχει σύνδεση δηλαδή μια ακμή E που τις ενώνει. Αυτή η αναπαράσταση θα χρησιμοποιηθεί και σε αυτή την εργασία

σαν τρόπος διευκόλυνσης και οπτικοποίησης, πάντα όμως πρέπει να θυμόμαστε ότι η χρησιμότητα των γραφικών παραστάσεων γράφων φτάνει μόνο μέχρι ένα σημείο, γράφοι με εκατοντάδες ή χιλιάδες κόμβους και δεκάδες χιλιάδες ακμές δεν είναι νοητό ότι η αναπαράστασή τους γραφικά να δώσει την ουδεμία βοήθεια κατανόησης του γράφου.



Σχήμα Β΄.1: Παράδειγμα γράφου G με $V = \{\text{Κώστας}, \text{Bryan}, \text{Γιάννης}\}$ και $E = \{\{\text{Κώστας}, \text{Bryan}\}, \{\text{Κώστας}, \text{Κοσμάς}\}, \{\text{Κοσμάς}, \text{Bryan}\}\}$.

Β΄.1.2 Γείτονες

Δύο κορυφές οι οποίες συνδέονται με μια ακμή θα λέγονται γειτονικές κορυφές ή απλά γείτονες, από το πάνω παράδειγμα, ο Κώστας είναι γείτονας με τον Bryan και τον Κοσμά, ο Κοσμάς είναι γείτονας με τον Κώστα αλλά δεν είναι γείτονας με τον Bryan αφού δεν τους ενώνει καμία ακμή. Τις διάφορες κορυφές μπορούν να τις συνδέουν πολλαπλές ακμές όχι μόνο μία, αλλά για την διάρκεια της εργασίας θα ασχοληθούμε αποκλειστικά με ενώσεις κορυφών με μία μοναδική ακμή. Επίσης μια κορυφή δεν γειτονεύει με τον εαυτό της ποτέ εκτός αν το αναφέρουμε.

Β΄.1.3 Βάρη

Μια ακμή στον γράφο αντιπροσωπεύει μία ένωση, αυτή η ένωση μπορεί να σημαίνει πολλά διαφορετικά πράγματα σε διαφορετικούς επιστήμονες, κάποιος που

αναλύει δίκτυα μεταφορών μπορεί να βλέπει τις ακμές ως δρόμους που ενώνουν πόλεις και λιμάνια. Στα δίκτυα οι ακμές του γράφου έχουν την έννοια της σύνδεσης ή και της δυνατότητας επικοινωνίας μεταξύ των δύο κόμβων (κορυφών). Και οι δύο αυτές ερμηνείες του κατασκευάσματος της ακμής έχουν την ιδιότητα του βάρους, δηλαδή της αντίστασης του δικτύου σε ενέργειες, του κόστους, υπάρχουν πολλά ονόματα και παραπλήσιες έννοιες με την ίδια βάση, το βάρος, ένας δρόμος μπορεί να είναι πιο δύσβατος από κάποιον άλλον, ή να έχει αυξημένη κίνηση σε σχέση με κάποιον άλλον, ή να είναι πιο μακρύς, άρα το κόστος μεταφοράς από αυτόν τον δρόμο να είναι αυξημένο σε σχέση με κάποιους άλλους. Το ίδιο ισχύει και στα δίκτυα, ένα κανάλι επικοινωνίας μπορεί να είναι κεντρικό, άρα να έχει πάρα πολύ κίνηση και μεγάλες καθυστερήσεις, ενώ ένα κανάλι τοπικό που το χρησιμοποιούν πολύ λίγοι να έχει πολύ χαμηλότερο κόστος μετάδοσης.

B'.1.4 Κατεύθυνση

Ένας γράφος λέγεται κατευθυνόμενος γράφος όταν οι ακμές του έχουν κατεύθυνση στην σύνδεσή τους (οι πλειάδες των ακμών σε αυτή την περίπτωση είναι διατεταγμένες, σε σειρά), δηλαδή όταν μια κορυφή u μπορεί να ενώνεται με κάποια άλλη κορυφή v αλλά να μην ισχύει το αντίθετο. Αν ο γράφος δεν έχει κατευθυνόμενες ακμές, τότε λέγεται μη-κατευθυνόμενος, και αυτός είναι ο τύπος γράφου που θα δούμε σε αυτή την εργασία.

B'.2 Μονοπάτια γράφων

Μονοπάτι ενός γράφου G λέγεται μια ακολουθία από κορυφές $V(G)$ (ή ακμές) η οποία δεν περιέχει επαναλήψεις κορυφών (ή ακμών). Κάθε μονοπάτι στον γράφο έχει κάποιο μήκος, που βρίσκεται μετρώντας τον αριθμό των κορυφών (ή ακμών) που περιέχει, και επίσης έχει και κάποιο κόστος, το οποίο βρίσκεται με δύο τρόπους: αθροίζοντας τα επιμέρους κόστη των ακμών στην περίπτωση όπου το μονοπάτι κατασκευάστηκε από ακμές, ή στην περίπτωση που αποτελείται από κορυφές, από τα κόστη των αντιστοίχων ακμών που ενώνουν τις κορυφές του μονοπατιού ανά δύο.

Παράδειγμα του ίδιου ακριβώς μονοπατιού εκφρασμένου (κατασκευασμένου) με κορυφές και με ακμές: $\{2, 5, 7, 3\}$ (μήκος $n - 1 = 3$) ως κορυφές, και $\{\{2, 5\}, \{5, 7\}, \{7, 3\}\}$ (μήκος 3) ως ακμές.

Κάθε μονοπάτι λέμε ότι "συνδέει" τις δύο κορυφές που βρίσκονται στην αρχή και στο τέλος της ακολουθίας, δηλαδή το μονοπάτι του παραδείγματος συνδέει τις

κορυφές 2 και 3.

Β΄.2.1 Συνεκτικότητα

Ένας γράφος λέγεται συνεκτικός όταν υπάρχει ένα μονοπάτι μεταξύ του κάθε ζεύγους κορυφών. Ένας γράφος που δεν είναι συνεκτικός ονομάζεται μη-συνεκτικός. Ένας γράφος λέγεται ότι είναι μη-συνεκτικός εάν υπάρχουν δύο οποιεσδήποτε κορυφές έτσι ώστε κανένα μονοπάτι να μην υπάρχει που να μπορεί να τις συνδέσει, αυτό σημαίνει στα δίκτυα ότι καμία επικοινωνία δεν μπορεί να υπάρξει μεταξύ κάποιων μερών ενός μη-συνεκτικού γράφου, αντιθέτως σε ένα συνεκτικό γράφο, επικοινωνία μπορεί να υφίσταται μεταξύ οποιωνδήποτε δύο κορυφών. Στην διάρκεια αυτής της εργασίας θα ασχοληθούμε αποκλειστικά με συνεκτικούς γράφους.

Β΄.2.2 Ελάχιστα μονοπάτια

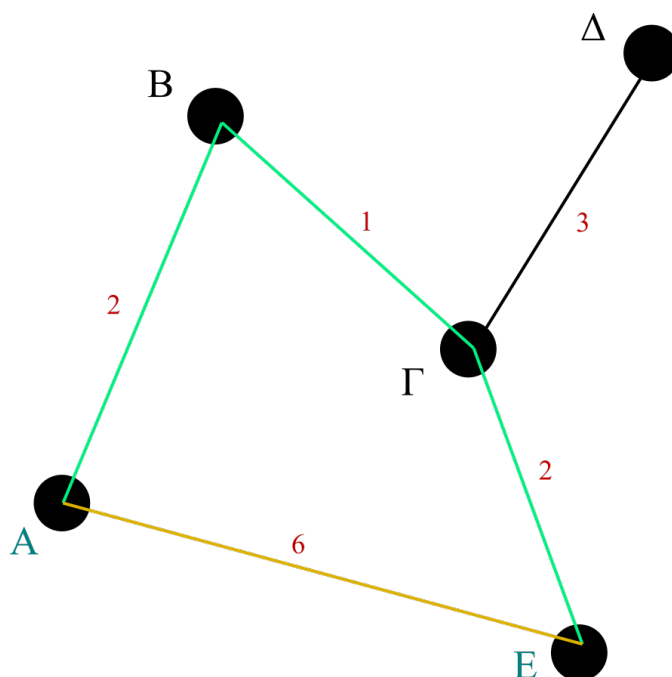
Όταν εισάγαμε την έννοια του μονοπατιού πάνω, δεν δώσαμε κάποιο συγκεκριμένο περιορισμό στην έννοια του, το ελάχιστο μονοπάτι περιορίζει τον ορισμό του μονοπατιού στο ελάχιστο δυνατό μήκος για το συγκεκριμένο ζεύγος κορυφών που συνδέει. Αν προσθέσουμε την έννοια των βαρών ακμών τότε το ελάχιστο μονοπάτι από μία κορυφή $u \in V$ σε μία διαφορετική κορυφή $z \in V$ ορίζεται ως το μονοπάτι μεταξύ των δύο με το ελάχιστο συνολικό βάρος των ακμών που το αποτελούν, δηλαδή υπάρχει η περίπτωση ένα μονοπάτι που περιέχει περισσότερες κορυφές να έχει μικρότερο συνολικό βάρος από κάποιο που έχει λιγότερες κορυφές, δεν είναι το μήκος του μονοπατιού που παίζει τον ρόλο αλλά το άθροισμα των βαρών των συγκεκριμένων ακμών που αποτελούν το μονοπάτι.

Β΄.3 Δέντρα

Ένας γράφος αποκαλείται και δέντρο όταν είναι συνεκτικός και δεν περιέχει κύκλους. Δηλαδή σε ένα δέντρο υπάρχει ένα μοναδικό μονοπάτι που συνδέει το κάθε ζευγάρι κορυφών.

Β΄.3.1 Κύκλοι

Ένας κύκλος είναι ένα μονοπάτι γράφου όπου συνδέει μια κορυφή με τον εαυτό της. Αφού έχουμε αναφέρει ότι στα μονοπάτια δεν επιτρέπονται οι επαναλήψεις ακμών, για να μπορέσει ένα μονοπάτι να φτάσει την κορυφή εκείνη από όπου ξεκίνησε θα πρέπει να υπάρχει μια ακμή εισόδου, και μια ακμή εξόδου από την κο-



Σχήμα Β'.2: Παράδειγμα ελάχιστου μονοπατιού από την κορυφή Α στην κορυφή Ε. Χωρίς βάρη {Α, Ε}, και με βάρη {Α, Β, Γ, Ε}.

ρυφή, σαν ελάχιστη προϋπόθεση για την ύπαρξη κύκλου, αφού αν δεν υπήρχαν οι δύο αυτές ακμές, δεν θα μπορούσαμε ποτέ να επιστρέψουμε στην κορυφή από την ίδια ακμή από και όπου εξήλθαμε. Στο σχήμα 2.2 το μονοπάτι {Ε, Γ, Β, Α, Ε} είναι ένας κύκλος.

Β'.3.2 Ρίζα

Ο ορισμός σαν ρίζα αποδίδει σε μία μοναδική κορυφή στον γράφο τον ρόλο της αναφοράς ως την πιο "υψηλή" ιεραρχικά κορυφή, σχηματικά είναι είτε η υψηλότερη κορυφή στον γράφο, είτε η χαμηλότερη. Η ρίζα σε ένα δέντρο είναι εντελώς ένα κατασκεύασμα δικό μας για να μας διευκολύνει την δουλεία με τα δέντρα, να μας δώσει ένα κοινό σημείο αναφοράς για τις παρακάτω έννοιες των δέντρων. Για παράδειγμα μπορούμε στο σχήμα 2.3 να θέσουμε ως ρίζα την κορυφή Η.

Β'.3.3 Γονείς

Σε ένα δέντρο που έχει ρίζα, ο γονέας μιας κορυφής είναι η κορυφή που είναι η επόμενη στο μονοπάτι προς τη ρίζα, κάθε κορυφή στο δέντρο εκτός από τη ρίζα

έχει ένα μοναδικό γονέα. Για να ξεκαθαρίσουμε ακόμη πιο πολύ τα δέντρα, καλό είναι να αναφέρουμε ότι ενώ είπαμε πριν ότι η ρίζα του δέντρου δεν έχει γονέα αυτό δεν σημαίνει τίποτα το ιδιαίτερο για τον γράφο, για παράδειγμα, αν επιλέξουμε μια διαφορετική ρίζα για το ίδιο δέντρο τότε η παλαιά ρίζα, που δεν είχε γονέα, θα έχει τώρα κάποιον γονέα. Στο σχήμα 2.3 ο γονέας της κορυφής Z είναι η κορυφή H , και ο γονέας της κορυφής E είναι η κορυφή A .

Β'.3.4 Παιδιά

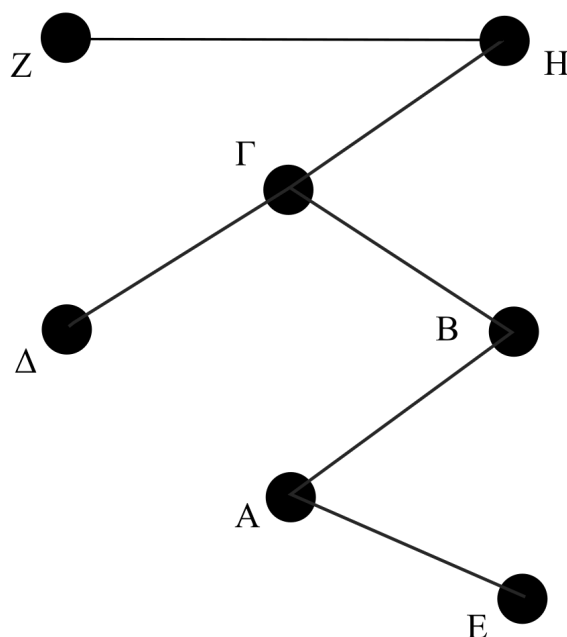
Παιδιά μιας κορυφής ονομάζονται όλες οι κορυφές που έχουν την αρχική κορυφή ως γονέα, δηλαδή μοιράζονται μία ακμή με την κορυφή και είναι σε αντίθετη φορά από την ρίζα του δέντρου. Στο σχήμα 2.3 τα παιδιά της κορυφής Γ είναι οι κορυφές B και Δ .

Β'.3.5 Απόγονοι

Ένας απόγονος κάποιας κορυφής είναι η κάθε κορυφή που είναι είτε παιδί ή είναι (αναδρομικά) παιδί οποιουδήποτε από τα παιδιά της αρχικής κορυφής. Στο σχήμα 2.3 για παράδειγμα βλέπουμε ότι με ρίζα την κορυφή H , η κορυφή A είναι απόγονος των κορυφών B , Γ και H , και η κορυφή B έχει σαν απογόνους τις κορυφές A και E .

Β'.4 Δέντρα επικάλυψης

Παραπάνω αναφέραμε ότι οι προϋποθέσεις για ένα γράφο να είναι δέντρο είναι δύο: να είναι συνεκτικός και να μην παρουσιάζει κύκλους πουθενά. Αυτό σημαίνει ότι σε κάθε συνεκτικό γράφο, μπορούμε να δημιουργήσουμε ένα δέντρο, που περιέχει όλες τις κορυφές, απλώς αν εξαλείψουμε όλους τους κύκλους. Αν το κάνουμε αυτό, τότε εκείνος ο γράφος που δημιουργήσαμε θα αποτελείται από όλες τις κορυφές του αρχικού γράφου, και δεν θα έχει κύκλους, άρα θα είναι ένα δέντρο. Αν αυτό το δέντρο έχει τον ελάχιστο αριθμό ακμών από όλα τα πιθανά δέντρα που μπορούμε να κατασκευάσουμε με αυτόν τον τρόπο τότε αυτό το δέντρο αποκαλείται δέντρο επικάλυψης του αρχικού γράφου, γιατί καλύπτει όλες τις κορυφές του γράφου με τις λιγότερες δυνατές ακμές. Κάποιος συγκεκριμένος γράφος μπορεί να περιέχει πολλαπλά δέντρα επικάλυψης αφού είναι πολύ συχνό να μπορείς να κατασκευάσεις πολλαπλά δέντρα με τον ίδιο ελάχιστο αριθμό ακμών στον ίδιο συνεκτικό γράφο.



Σχήμα B'3: Παράδειγμα δέντρου.

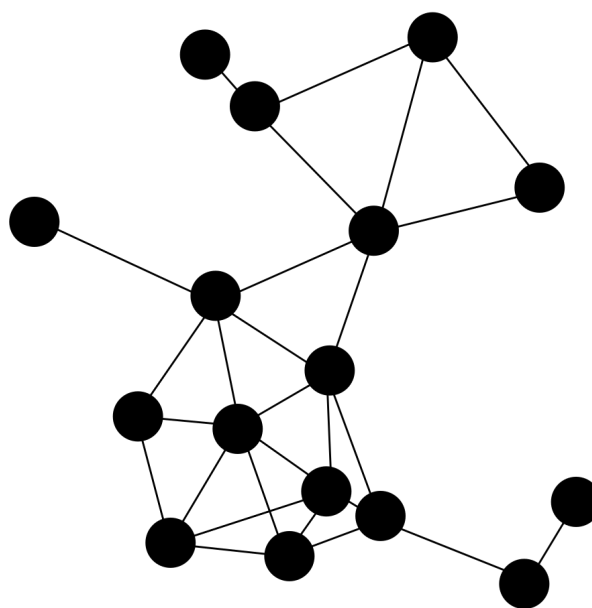
B'4.1 Ελάχιστα δέντρα επικάλυψης

Το ελάχιστο δέντρο επικάλυψης είναι παρόμοιο με το απλό δέντρο επικάλυψης, όμως με την διαφορά ότι το άθροισμα των βαρών των ακμών του είναι το ελάχιστο δυνατό.

B'5 Τυχαίοι γεωμετρικοί γράφοι

Φανταστείτε ότι παίρνετε μια χούφτα από φασόλια, και τα πετάτε όλα μαζί τυχαία στο πάτωμα, αφότου σταματήσουν να κινούνται και κατασταλάζουν στις θέσεις τους, μετράτε τις αποστάσεις μεταξύ κάθε φασολιού, αν είναι κάτω από ένα όριο, σημειώνετε μια γραμμή που αντιπροσωπεύει την σύνδεση (ακμή) μεταξύ τους. Η διαδικασία που μόλις περιγράψαμε είναι μία καλή αναλογία για το πως κατασκευάζονται οι τυχαίοι γεωμετρικοί γράφοι, η τοποθέτηση της κάθε κορυφής (φασολιού) γίνεται βάση κάποιας συνάρτησης γεννήτριας ψευδοτυχαίων αριθμών μαζί με κάποια συγκεκριμένη κατανομή πιθανότητας (πχ. κανονική/γκουσσσιανή, ομοιόμορφη, εκθετική, και άλλες.) για να καθορίσει τις συντεταγμένες των κορυφών, και αφού τελειώσει με την τοποθέτηση των κορυφών, κάνει ένα πέρασμα και

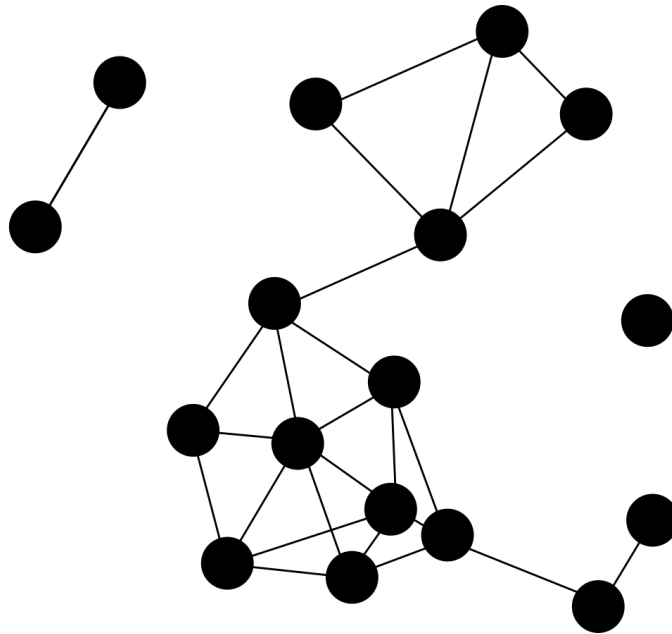
ελέγχει τις ευκλείδειες αποστάσεις όλων των κορυφών, αν είναι κάτω από το όριο που έχει δώσει ο σχεδιαστής του αλγορίθμου, θα δημιουργήσει σύνδεση μεταξύ τους, αλλιώς όχι. Στην διαδικασία δημιουργίας ενός τυχαίου γεωμετρικού γράφου, υπάρχει η πιθανότητα να μην παραχθεί ένα συνεκτικό δίκτυο, σε τέτοιες περιπτώσεις, αν χρειαζόμαστε συνεκτικότητα, πρέπει να ξεκινήσουμε την διαδικασία δημιουργίας του τυχαίου γεωμετρικού γράφου από την αρχή.



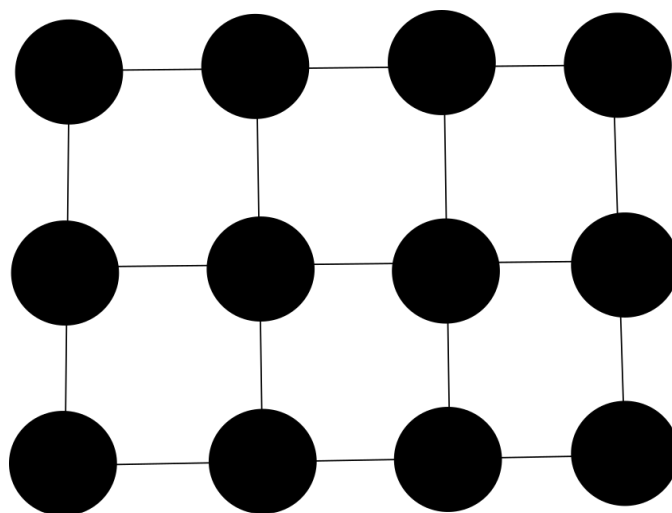
Σχήμα Β΄.4: Παράδειγμα τυχαίου γεωμετρικού γράφου.

Β΄.6 Πλεγματικές τοπολογίες

Η πλεγματική τοπολογία είναι ένα είδος γράφου όπου οι κορυφές διατάσσονται σε γραμμές και στήλες και οι συνδέσεις των κορυφών είναι επίσης ανά γραμμή και ανά στήλη, ο μέγιστος αριθμός γειτονικών κόμβων σε ένα δίκτυο με πλεγματική τοπολογία είναι 4 (πάνω, κάτω, δεξιά, και αριστερά) και ο ελάχιστος (στις γωνίες του ορθογωνίου) είναι 2. Το παράδειγμα Β΄.6 δίνει ένα απλό παράδειγμα αυτής της τοπολογίας γράφων.



Σχήμα Β'.5: Παράδειγμα μη συνεκτικού τυχαίου γεωμετρικού γράφου.



Σχήμα Β'.6: Παράδειγμα γράφου πλεγματικής τοπολογίας.

Κεφάλαιο Γ'

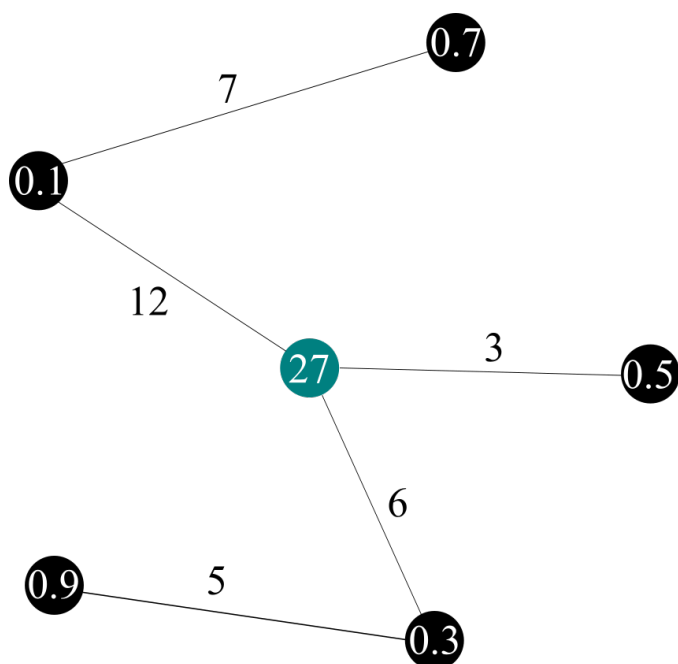
Μετεγκατάσταση Υπηρεσίας

Γ'.1 Κόστος τοποθεσίας

Ως κόστος τοποθεσίας ονομάζουμε το άθροισμα του συνόλου των αποστάσεων επί την απαίτηση υπηρεσίας από όλους τους κόμβους προς τον κόμβο υπηρεσίας. Δηλαδή διαλέγουμε τον κόμβο για τον οποίον θέλουμε να βρούμε το κόστος τοποθεσίας, μετά βρίσκουμε για κάθε άλλο κόμβο του δικτύου το ελάχιστο μονοπάτι προς τον κόμβο υπηρεσίας το πολλαπλασιάζουμε με την απαίτηση του κόμβου, παίρνουμε τις τιμές αυτές από κάθε κόμβο και τις αθροίζουμε μαζί. Το κόστος τοποθεσίας στις πιο πολλές θέλουμε να το ελαχιστοποιήσουμε, γιατί αυτό σημαίνει ότι για όλες τις κινήσεις που θα κάνουμε από όλους τους κόμβους του δικτύου προς την τοποθεσία, θα έχουμε το λιγότερο κόστος συνολικά, το οποίο είναι κάτι που θέλουμε. Στο σχήμα Γ'.1 βλέπουμε έναν απλό υπολογισμό του κόστους της τοποθεσίας του χρωματισμένου κόμβου στο συγκεκριμένο δέντρο.

Γ'.2 Το 1-median πρόβλημα

Το k -median πρόβλημα [3] είναι το πρόβλημα καθορισμού των τοποθεσιών k διαφορετικών εγκαταστάσεων σε διαφορετικές τοποθεσίες έτσι ώστε να ελαχιστοποιηθεί η συνολική βαρυσταθμισμένη απόσταση μεταξύ των κόμβων ζήτησης και της πλησιέστερης από τις k εγκατάστασης υπηρεσίας (η βαρυσταθμισμένη απόσταση βγαίνει από τον πολλαπλασιασμό της απόστασης ελάχιστου μονοπατιού από τον κόμβο προς την υπηρεσία με τον ρυθμό παραγωγής πακέτων του συγκεκριμένου κόμβου). Σε αυτή την εργασία θα ασχοληθούμε αποκλειστικά με το 1-median πρόβλημα, που είναι η πρώτη και απλούστερη περίπτωση του k -median προβλήμα-



Σχήμα Γ'.1: Παράδειγμα εύρεσης κόστους ενός κόμβου στο δίκτυο (χρωματισμένος). Στους κόμβους μέσα αναγράφεται η απαίτηση υπηρεσίας, στον κεντρικό κόμβο που στεγάζεται η υπηρεσία, αναγράφεται το συνολικό κόστος που υπολογίστηκε.

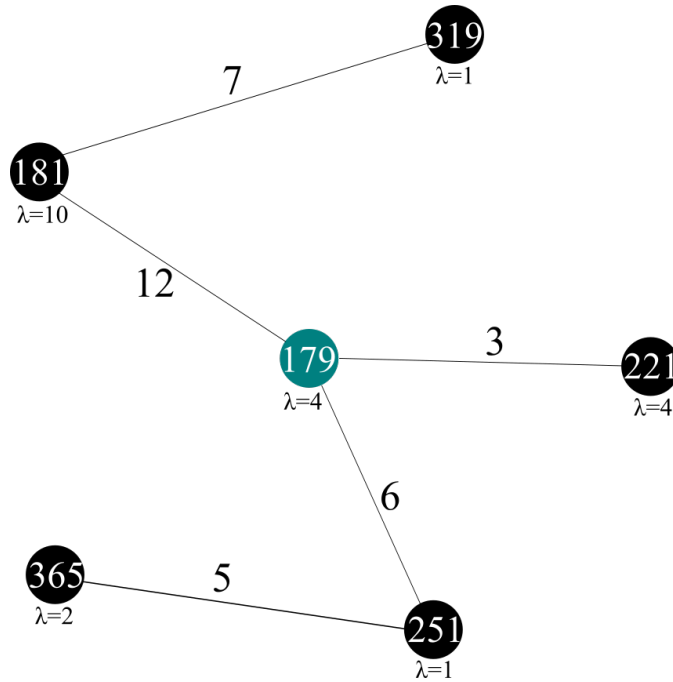
τος, στην οποία ζητείται να τοποθετήσουμε μία μόνο υπηρεσία σε μία τοποθεσία. Επίσης, έχουμε σαν απαίτηση ότι η τοποθεσία μπορεί να είναι μόνο κάποιος από τους ήδη υπάρχοντες κόμβους του γράφου, δηλαδή μιλάμε για διακριτές πιθανές τοποθετήσεις της υπηρεσίας, και σαν γεγονός ότι η υπηρεσία δεν έχει κάποιο συγκεκριμένο όριο επεξεργαστικής ισχύς, άρα δεν υπάρχει όριο στην ταχύτητα εξυπηρέτησης ή σε άλλες μετρικές όπως τον αριθμό ταυτόχρονων πελατών που μπορεί να εξυπηρετήσει.

Η διατύπωση του 1-median προβλήματος είναι η εξής: Σκεφτόμαστε την τοπολογία του δικτύου σαν ένα μη-κατευθυνόμενο γράφο $G(V, E)$ όπου V είναι το σύνολο των κόμβων ως κορυφές του γράφου, και E το σύνολο των συνδέσμων των κόμβων ως ακμές του γράφου. S_v είναι το σύνολο των γειτονικών κόμβων του v , και (u, v) είναι ένας συγκεκριμένος σύνδεσμος (ακμή) μεταξύ δύο κόμβων (κορυφών) u και v (το v είναι μια απλή κορυφή στον γράφο, V είναι το σύνολο των κορυφών), σε κάθε τέτοια σύνδεση αντιστοιχεί κάποιο συγκεκριμένο βάρος (σκεφτόμαστε κόστος, αντίσταση, στην κίνηση πάνω στην ακμή) σαν πραγματικός αριθμός. Ως $d_{(u,v)}$ λέμε την απόσταση των δύο κορυφών, όπου η απόσταση υπολογίζεται

ως το άθροισμα των επιμέρους βαρών των συνδέσμων του ελάχιστου μονοπατιού των δύο κορυφών (προφανές ότι $d_{(v,v)} = 0$). Ο ρυθμός ανά δευτερόλεπτο της παραγωγής πακέτων του κόμβου v προς την υπηρεσία συμβολίζεται με λ_v , και θα αναφέρεται ως απαιτήσεις υπηρεσίας του κόμβου v , αυτά τα πακέτα προοριζόμενα για την υπηρεσία του δικτύου δρομολογούνται από το πρωτόκολλο επικοινωνίας του δικτύου μέσα από ένα ελάχιστο μονοπάτι προς την υπηρεσία. Έστω C_y το συνολικό κόστος υπηρεσίας για όλους τους κόμβους του δικτύου, όταν η υπηρεσία βρίσκεται στην τοποθεσία του κόμβου y .

$$C_y = \sum_{\forall v \in V} \lambda_v d_{(v,y)} \quad (\Gamma'.1)$$

Για να λύσουμε το 1-median πρόβλημα, θα πρέπει να βρούμε τον κόμβο $y \in V$ για τον οποίο το παραπάνω άθροισμα ελαχιστοποιείται. Στο σχήμα Γ'.2 βλέπουμε μια απλή λύση του 1-median προβλήματος.



Σχήμα Γ'.2: Παράδειγμα λύσης του 1-median (ο χρωματισμένος κόμβος). Στους κόμβους μέσα αναγράφεται το C_y όταν τοποθετούμε την υπηρεσία στον συγκεκριμένο κόμβο, δίνονται οι απαιτήσεις υπηρεσίας των κόμβων και τα βάρη των ακμών.

Γ'.3 Στοιχεία υπηρεσίας

Η υπηρεσία που έχουμε αναφέρει πολλές φορές δεν έχει κάποια συγκεκριμένη διαδραστική λειτουργία με το δίκτυο, αλλά δουλεύει ως κάποιου είδους συλλέκτη πακέτων (πχ. τιμών θερμοκρασίας) από όλους τους κόμβους, ο κάθε κόμβος στέλνει ένα πακέτο με προορισμό την υπηρεσία με ρυθμό ανάλογο με την τιμή λ που του αντιστοιχεί.

Γ'.4 Μετεγκατάσταση

Η τοποθέτηση υπηρεσιών σε δίκτυα συνήθως αντιμετωπίζεται ως ένα πρόβλημα κεντριοποιημένης γνώσης, δηλαδή ο αρχιτέκτονας του δικτύου και των υπηρεσιών γνωρίζει τα πάντα για το δίκτυο, σε τέτοιες καταστάσεις η λύση του προαναφερθέντος 1-median προβλήματος δίνει την λύση στην βέλτιστη τοποθέτηση μιας μοναδικής υπηρεσίας στο δίκτυο. Τι γίνεται όμως αν δεν έχουμε γνώση για την τοπολογία του δικτύου, ή το δίκτυο μας βρίσκεται σε κατάσταση συχνών αλλαγών στην τοπολογία του, ή ακόμη και όταν το δίκτυο είναι πάρα πολύ μεγάλο για την λύση του 1-median προβλήματος, δηλαδή περιέχει δεκάδες χιλιάδες ή και εκατοντάδες χιλιάδες κόμβους? Σε πολλές περιπτώσεις, κάποιες που αναφέραμε και πολλές άλλες που δεν αναφέραμε, δεν είναι δυνατό ή βάσιμο να αποφασίζεται η τοποθέτηση της υπηρεσίας μέσω τέτοιων κεντριοποιημένων, χρονοβόρων, και περίπλοκων βημάτων όπως είναι η λύση του 1-median προβλήματος, σε τέτοιες καταστάσεις πρέπει να κοιτάξουμε στην μετεγκατάσταση υπηρεσίας Mohammadi Zanjireh and Larijani [7], δηλαδή στην δυναμική μετακίνησης της τοποθέτησης της υπηρεσίας βασισμένοι πάντα σε τοπικά διαθέσιμη πληροφορία (στον κόμβο υπηρεσίας ή και στους γειτονικούς κόμβους του).

Γ'.5 Το κριτήριο μετεγκατάστασης

Ο κόμβος όπου στεγάζεται η υπηρεσία χρειάζεται απλά να παρακολουθεί τον συνολικό όγκο των δεδομένων προοριζόμενα για την υπηρεσία από τον κάθε γειτονικό κόμβο της, αυτή η τοπική μέτρηση των στατιστικών κίνησης είναι αρκετή για την εφαρμογή του κριτηρίου μετεγκατάστασης και την μετακίνηση της υπηρεσίας σε μονοτονικά μικρότερου κόστους θέσεις μέσα στο δίκτυο.

Πιο συγκεκριμένα, ανά κάποια κανονισμένη χρονική περίοδο, στον κόμβο υπηρεσίας, γίνεται μία στατιστική μέτρηση του αριθμού των πακέτων προοριζόμενα

για την υπηρεσία από κάθε γειτονικό κόμβο της υπηρεσίας που μπορεί να προωθήσει πακέτα προς την υπηρεσία από το ευρύτερο δίκτυο. Στο τέλος αυτής της χρονικής περιόδου γίνεται ο έλεγχος του κριτηρίου της μετεγκατάστασης: αν κάποια ακμή που προσπίπτει στον κόμβο υπηρεσίας έχει υψηλότερο από το διπλάσιο φορτίο πακέτων από το άθροισμα όλων των υπόλοιπων ακμών τότε το κριτήριο ισχύει και η υπηρεσία ξεκινά μια διαδικασία μετεγκατάστασης του εαυτού της στον κόμβο που βρίσκεται στην άλλη άκρη της ακμής.

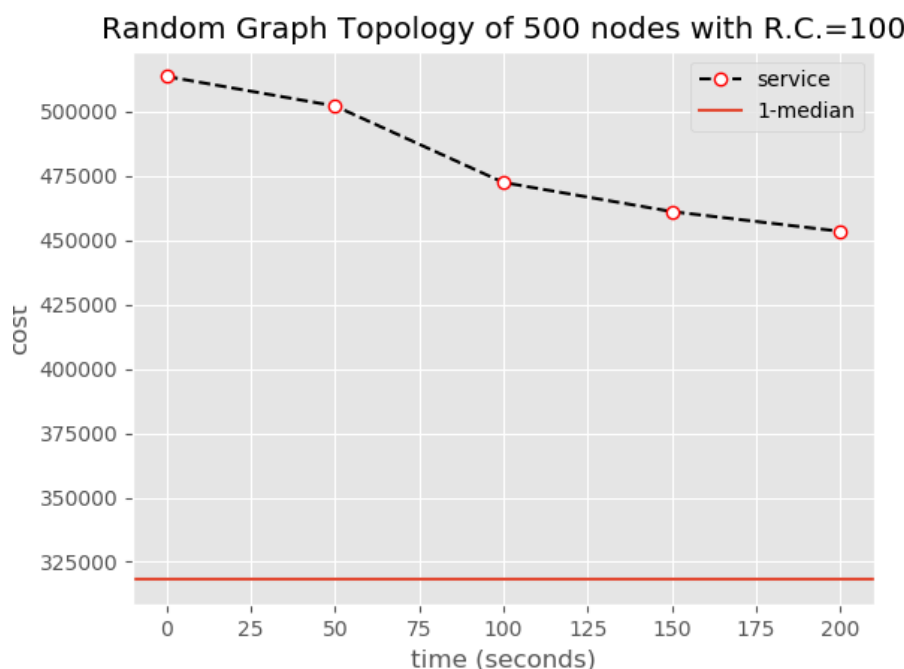
Η εφαρμογή της πολιτικής μετακινεί την υπηρεσία προς τη βέλτιστη θέση σε κάθε βήμα, όμως καθώς η βέλτιστη θέση μπορεί να αλλάξει με την πάροδο του χρόνου και με διάφορα γεγονότα που πιθανώς να επηρεάσουν ένα δυναμικό δίκτυο, τότε πολιτική προσαρμόζει την πορεία της μετακίνησης της υπηρεσίας συνεχώς προς την παρούσα βέλτιστη θέση.

Γ'.6 Μελέτη της πολιτικής μετεγκατάστασης

Σε αυτό το σημείο θα μελετήσουμε στην πράξη τώρα τα αποτελέσματα της πολιτικής μετεγκατάστασης μέσω διαφόρων διαγραμμάτων. Τα αποτελέσματα που χρησιμοποιήθηκαν για την παραγωγή όλων των διαγραμμάτων πάρθηκαν μέσω ειδικού κώδικα με το εργαλείο προσομοιώσεων δικτύων omnet++, για το οποίο κώδικας αναπτύχθηκε με σκοπό την κατασκευή δικτύου τοπολογίας τυχαίου γεωμετρικού γράφου, και προσομοίωσης επικοινωνίας των δικτυακών κόμβων με την υπηρεσία του δικτύου (η οποία είναι μοναδική στο δίκτυο όπως έχουμε αναφέρει).

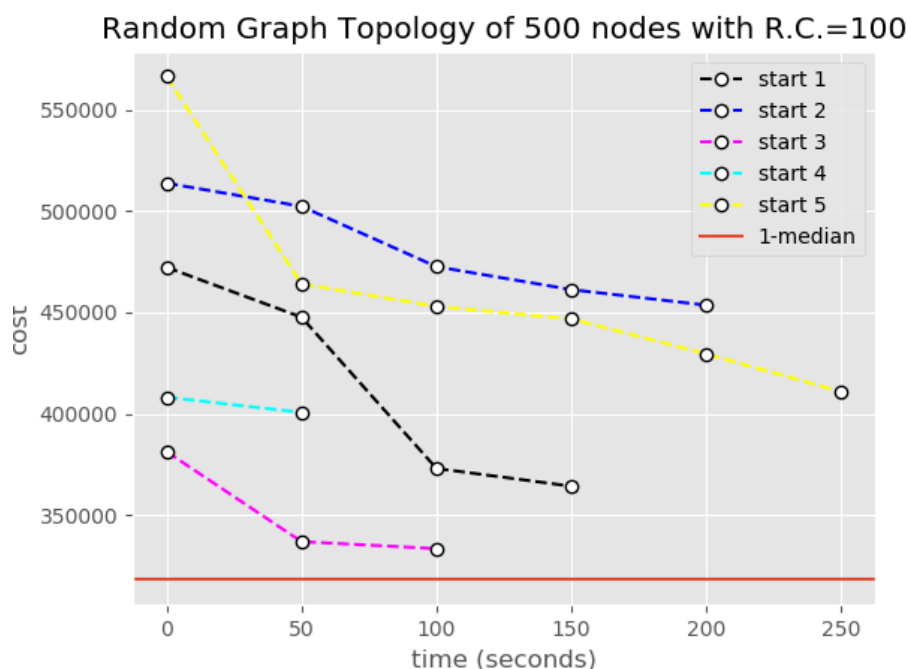
Γ'.6.1 Σε γεωμετρικούς τυχαίους γράφους

Στο διάγραμμα 4.3 βλέπουμε την μεταβολή του κόστους της υπηρεσίας καθώς μετακινείται από κόμβο σε κόμβο, με βάση το κριτήριο μετεγκατάστασης ανά κάποιο χρονικό διάστημα μέτρησης στατιστικών, στο παράδειγμα μας 50 δευτερόλεπτα. Όπως βλέπουμε η υπηρεσία ξεκινά με κόστος 570,000 και μετά από 4 μετακινήσεις το κόστος έχει πέσει σε 455,000, μια μείωση της τάξης του 25%. Σαν ελάχιστη δυνατή τιμή του κόστους έχουμε το κόστος που έχει ο κόμβος που βρίσκεται στην λύση του 1-median προβλήματος, ο οποίος έχει το κόστος 323,000 (σαν μια κόκκινη γραμμή), σημαντικά πιο χαμηλά από ότι βλέπουμε για την υπηρεσία μας. Θα είναι αδύνατο να φτάσουμε το κόστος του 1-median κόμβου με αυτή την πολιτική μετεγκατάστασης, αλλά ξέρουμε ότι τα βήματα που θα κάνουμε μας δίνουν μια μονοτονική μείωση στο κόστος, δηλαδή θα έχουμε σε κάθε βήμα κάποια μετρήσιμη, όσο μικρή και να είναι, μείωση στο κόστος της υπηρεσίας.



Σχήμα Γ΄.3: Διάγραμμα μεταβολής του κόστους τοποθεσίας της υπηρεσίας κατά την πάροδο του χρόνου σε τυχαίο γεωμετρικό γράφο 500 κόμβων.

Στο διάγραμμα Γ΄.4 βλέπουμε την μεταβολή του κόστους τοποθεσίας της υπηρεσίας 5 διαφορετικών πειραμάτων με διαφορετικές θέσεις εκκίνησης κατά την πάροδο του χρόνου σε όμοιο τυχαίο γεωμετρικό γράφο 500 κόμβων. Στο διάγραμμα, οι διαφορετικές χρωματισμένες γραμμές αντιπροσωπεύουν ένα συγκεκριμένο μεμονωμένο πείραμα πάνω στον ίδιο γράφο, και οι γραμμές τελειώνουν όταν το κριτήριο σταματάει να ισχύει. Βλέπουμε ότι το ποσοστό μείωσης του κόστους με βάση την μετεγκατάσταση είναι συνδεδεμένο με την αρχική τοποθέτηση της υπηρεσίας, και την ακριβή τοπολογία του δικτύου, και είναι αδύνατο να προβλέψουμε την πορεία της υπηρεσίας πέρα από κάποιες πολύ βασικές, και όχι πάντα σωστές, παρατηρήσεις όπως το γεγονός του ότι όσο πιο "κοντά" βρίσκεται η αρχική τοποθεσία στο κόστος του 1-median κόμβου είναι αναμενόμενο να γίνουν όλο και λιγότερες μετακινήσεις βάση του κριτηρίου, και όσο πιο "μακριά" είναι η αρχική τοποθέτηση, τόσο πιο πολλές κινήσεις θα έχουμε τελικά, αυτό φαίνεται και στο διάγραμμα όπου οι αρχικές τοποθετήσεις που είναι πιο "πάνω" στο διάγραμμα, δηλαδή έχουν μεγαλύτερο κόστος, από την κόκκινη γραμμή του 1-median, έχουν και τις περισσότερες κινήσεις.

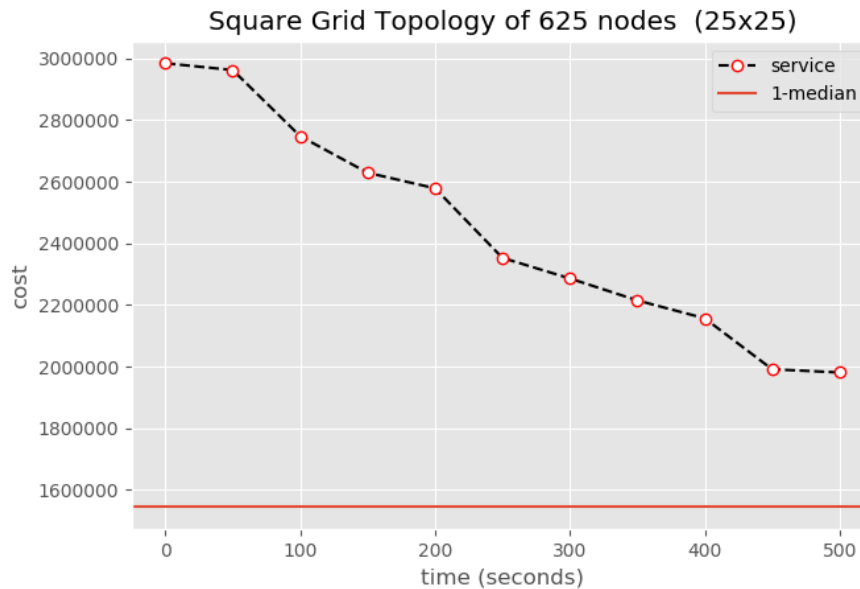


Σχήμα Γ'.4: Διάγραμμα μεταβολής του κόστους τοποθεσίας της υπηρεσίας 5 διαφορετικών πειραμάτων με διαφορετικές θέσεις εκκίνησης κατά την πάροδο του χρόνου σε όμοιο τυχαίο γεωμετρικό γράφο 500 κόμβων.

Γ'.6.2 Σε πλεγματική τοπολογία

Τετράγωνα

Στο διάγραμμα Γ'.5 βλέπουμε την μεταβολή του κόστους της υπηρεσίας κατά την πάροδο του χρόνου σε τετράγωνο πλεγματικό δίκτυο διαστάσεων 25 επί 25. Σε σχέση με τους τυχαίους γεωμετρικούς γράφους έχουμε προφανώς παραπάνω κινήσεις, αλλά γιατί? Η απάντηση βρίσκεται στον αριθμό των γειτόνων που έχει ο κάθε κόμβος, στους τυχαίους γεωμετρικούς γράφους ο κάθε κόμβος έχει μόνο 4 γειτονικούς κόμβους, και έτσι η κίνηση θα είναι σχεδόν πάντα πιο ισχυρή προς τις 2 ακμές προς τους γειτονικούς κόμβους που βρίσκονται πιο κοντά στην θέση της λύσης του 1-median. Άρα το κριτήριο μετεγκατάστασης έχει πολύ πιο "εύκολη" δουλειά σε πλεγματικές τοπολογίες γιατί η κίνηση των πακέτων περνάει από λιγότερες ακμές από ότι σε τυχαίους γεωμετρικούς γράφους, που συνήθως μιλάμε για περισσότερους γειτονικούς κόμβους από τέσσερις, σαν αποτέλεσμα υπάρχει μεγαλύτερη πιθανότητα μία από τις ακμές να έχει το μισό και πάνω της συνολικής

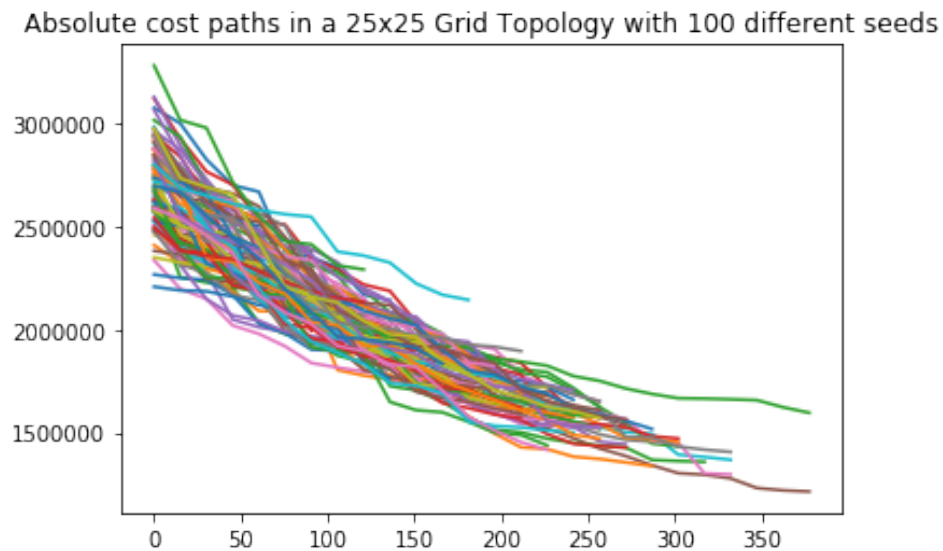


Σχήμα Γ΄.5: Διάγραμμα μεταβολής του κόστους τοποθεσίας της υπηρεσίας κατά την πάροδο του χρόνου σε τετράγωνο πλεγματοικό δίκτυο διαστάσεων 25 επί 25.

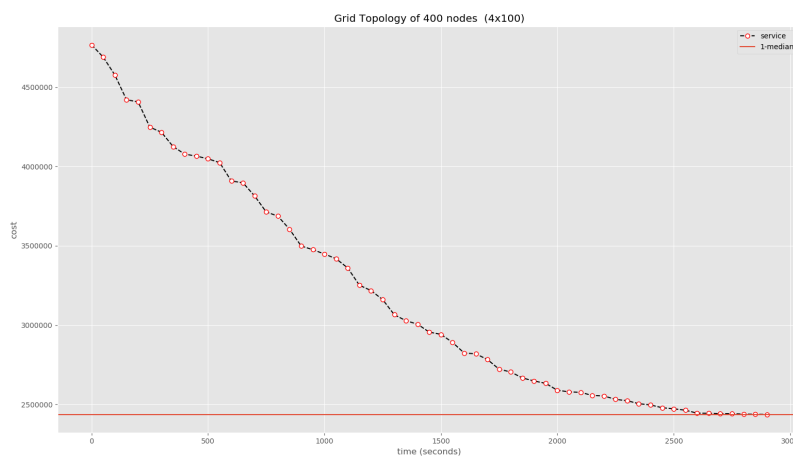
κίνησης προς τον κόμβο και άρα να επαληθεύει το κριτήριο μετεγκατάστασης.

Ορθογώνια

Αυτό γίνεται εντελώς εμφανές στο διάγραμμα Γ΄.7 όπου σε μια ορθογώνια πλεγματοική τοπολογία η υπηρεσία μετά από πενήντα κινήσεις φτάνει στην θέση της λύσης του 1-median, γιατί μέχρι και την τελευταία της κίνηση η υπηρεσία λαμβάνει την πλειοψηφία της κίνησης πακέτων από την ακμή που "δείχνει" προς το κέντρο του ορθογωνίου, το οποίο είναι η θέση της λύσης του 1-median προβλήματος. Αυτό δεν μας εκπλήσσει γιατί μια πλεγματοική τοπολογία με ανάγκες υπηρεσίας του κάθε κόμβου παραγόμενες από πιθανοτική συνάρτηση θα είναι "ομοιόμορφη" και το κέντρο βαρύτητας του ορθογωνίου θα είναι και η βέλτιστη τοποθεσία για την εξυπηρέτηση όλου του ορθογωνίου, άρα και η λύση του 1-median.



Σχήμα Γ'.6: Διάγραμμα μεταβολής του κόστους τοποθεσίας της υπηρεσίας κατά την πάροδο του χρόνου σε τετράγωνο πλεγματο δίκτυο διαστάσεων 25 επί 25 100 διαφορετικών πειραμάτων στο ίδιο σύστημα συντεταγμένων.



Σχήμα Γ'.7: Διάγραμμα μεταβολής του κόστους τοποθεσίας της υπηρεσίας κατά την πάροδο του χρόνου σε ορθογώνιο πλεγματο δίκτυο διαστάσεων 100 επί 4.

Κεφάλαιο Δ΄

Ενεργειακή Μελέτη Ασύρματων Δικτύων Αισθητήρων

Δ΄.1 Ασύρματα δίκτυα αισθητήρων

Ως "ασύρματο δίκτυο αισθητήρων" (wireless sensor network - WSN) ονομάζεται μια ομάδα διασκορπισμένων συσκευών με διάφορες δυνατότητες μετρήσεων περιβάλλοντος και υπολογισμών σε κάποια συγκεκριμένη περιοχή, διασυνδεδεμένα με ασύρματο τρόπο σε κάποια δικτυακή τοπολογία, ο σκοπός των οποίων είναι η παρακολούθηση και καταγραφή διαφόρων μετρήσιμων τιμών από το περιβάλλον τους και η επεξεργασία και συλλογή αυτών σε κάποια κεντρική αρχή ή κόμβο του δικτύου [1].

Τα WSNs είναι αποτελούμενα από αρκετές εκατοντάδες ή και (δεκάδες) χιλιάδες κόμβους, όπου ο καθένας κόμβος έχει πάνω του συνήθως πολλαπλούς αισθητήρες. Κάθε τέτοιος κόμβος ενός WSN έχει τυπικά τα εξής μέρη, έναν πομποδέκτη με εσωτερική ή εξωτερική κεραία, έναν μικρο-ελεγκτή ή ένα πιο μεγάλο και δυνατό επεξεργαστή, κάποια ποσότητα μνήμης για την αποθήκευση των προγραμμάτων ελέγχου, και τέλος μια μικρή μπαταρία. Τα μεγέθη των κόμβων που αποτελούν ένα WSN ποικίλουν πολύ, τα πιο διαδεδομένα μοντέλα διαθέσιμα για πώληση προς ιδιώτες είναι συνήθως της τάξης του μεγέθους μιας μπάρας δημητριακών, και κάποια στρατιωτικά προγράμματα κατασκεύασαν κόμβους μεγέθους ενός κόκκου ρυζιού [9]. Κύρια χαρακτηριστικά των κόμβων ενός WSN είναι η απειροελάχιστη υπολογιστική δυνατότητα σε σχέση με υπολογιστές γραφείου, περιορισμένη και πολλές φορές μη-επαναφορτιζόμενη πηγή ενέργειας ή χωρίς μπαταρία [10], ο ενσωματωμένος προγραμματισμός της πλατφόρμας, η επικοινωνία αποκλειστικά

με ασύρματο τρόπο, η ad-hoc τοπολογία του δικτύου και η εύκολη επέκταση του δικτύου, και άλλα. Τα ασύρματα δίκτυα αισθητήρων έχουν χρησιμοποιηθεί για μέτρηση θερμοκρασίας, επιπέδων ήχου, επίπεδα παρουσίας χημικών ουσιών, υγρασίας, βαρομετρικής πίεσης, σεισμολογικής δραστηριότητας, ηφαιστειακής δραστηριότητας, και άλλα [2].

Δ΄.1.1 MICAZ

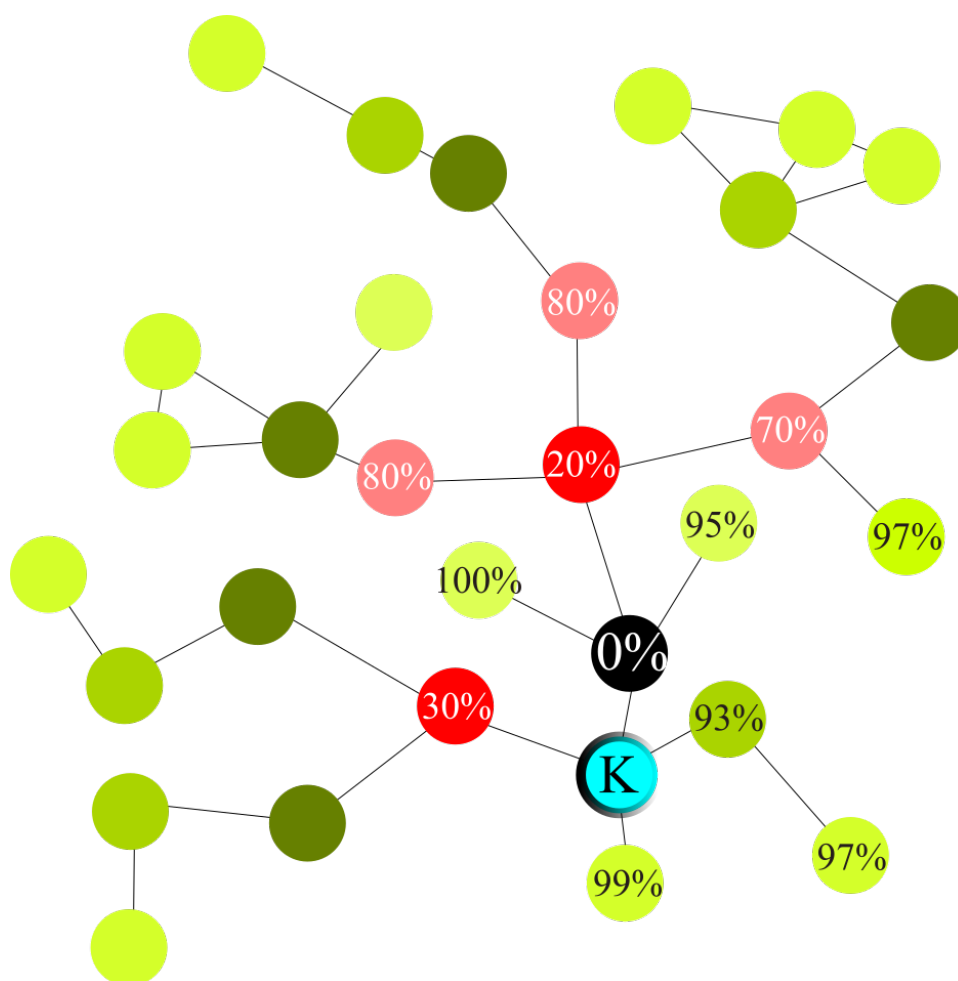
Το "mote" (κόμβος αισθητήρων ενός WSN) MICAZ [4] είναι η βάση για τους κόμβους στα δίκτυα που θα δούμε στα επόμενα κεφάλαια της εργασίας, οπότε ας δούμε τα βασικά στοιχεία του που μας ενδιαφέρουν.

Το MICAZ επικοινωνεί χρησιμοποιώντας το IEEE 802.15.4 πρωτόκολλο ασύρματης επικοινωνίας χαμηλής εμβέλειας (personal area wireless network - PAN) στα 250 kbps (συχνότητα 2.4Ghz μέχρι 2.9Ghz). Το MICAZ βασίζεται στον Atmel ATmega128L μικρο-ελεγκτή χαμηλής ισχύος ο οποίος τρέχει το λειτουργικό σύστημα MoteWorks και τα διάφορα προγράμματα χρήστη από την ενσωματωμένη μνήμη flash (128KB) του. Περιέχει 2 AA τύπου μπαταρίες, ο μικρο-ελεγκτής καταναλώνει 8mA (milliampere) σε ενεργή λειτουργία, 15μA (microampere) σε λειτουργία ύπνου. Για την λειτουργία της κεραίας καταναλώνει 20mA όταν λαμβάνει, 20mA σε λειτουργία idle, 1μA σε λειτουργία ύπνου, και 80mA όταν μεταδίδει.

Ένα από τα μεγαλύτερα θέματα στην έρευνα αυτών των δικτύων, λόγο ότι ο χρόνος ζωής που αποδίδει η μπαταρία στον κάθε κόμβο είναι περιορισμένος είναι η ενεργειακή συμπεριφορά και η χρονική διάρκεια ζωής του δικτύου. Για την πρώτη κατηγορία, οι έρευνες κινούνται γύρω από την βελτιστοποίηση των διαφόρων πρωτοκόλλων επικοινωνίας, ώστε οι κεραίες να δουλεύουν για όσο λιγότερο δυνατό, και ο επεξεργαστής και οι διάφοροι αισθητήρες να είναι όσο πιο ενεργειακά αποδοτικοί γίνεται ώστε να καταναλώνουν ελάχιστη ενέργεια. Για την δεύτερη κατηγορία, αυτή της ενεργειακής συμπεριφοράς του δικτύου σαν ένα σύνολο υπάρχει ένα κύριο πρόβλημα, το πρόβλημα της ενεργειακής οπής (energy hole problem).

Δ΄.2 Το πρόβλημα της ενεργειακής οπής

Σε πολλές περιπτώσεις ένα δίκτυο ασύρματων αισθητήρων θα αποδίδει σε ένα μοναδικό κόμβο του τον ρόλο του κυρίου κόμβου, ο οποίος έχει ειδικές λειτουργίες και κεντρικό ρόλο στην λειτουργικότητα του δικτύου. Το πρωτόκολλο δρομολόγησης πακέτων στέλνει όλα τα πακέτα που δημιουργούνται στο δίκτυο προς αυτό



Σχήμα Δ'.1: Σχηματική αναπαράσταση του προβλήματος της ενεργειακής οπής με χρωματικό κώδικα για τα επίπεδα της εναπομείνουσας ενέργειας την στιγμή του θανάτου του δικτύου.

τον κύριο κόμβο, και ο κύριος κόμβος έχει τον ρόλο της μεταβίβασης πληροφοριών και στατιστικών στοιχείων ή και σημάτων κινδύνου προς κάποιο εξωτερικό παραλήπτη (εμάς) για επεξεργασία. Ένα δίκτυο αισθητήρων μπορεί να έχει πολλά τέτοιους κυρίους κόμβους, ειδικά αν είναι πάρα πολύ μεγάλο και περιέχει χιλιάδες κόμβους, αλλά σε αυτή την εργασία θα κοιτάξουμε στις περιπτώσεις όπου δουλεύουμε με ένα μόνο κύριο κόμβο.

Η κυκλοφορία της πληροφορίας σε ένα δίκτυο ασύρματων αισθητήρων ακολουθεί μια όλο και πιο συγκλίνουσα πορεία προς τον κύριο κόμβο του δικτύου, όπου οι κόμβοι πλησιέστεροι στο κύριο κόμβο αναμεταδίδουν όλο και βαρύτερα φορτία κυκλοφορίας όσο πιο κοντά βρίσκονται στον κόμβο. Δηλαδή προκύπτει, ότι

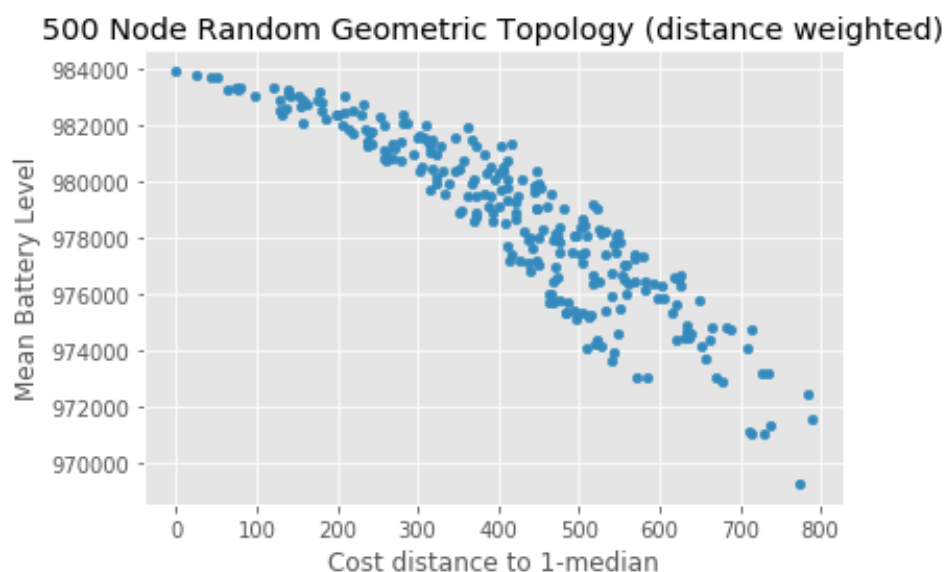
οι κόμβοι που βρίσκονται γύρω από τον κύριο θα εξαντλήσουν την ενέργεια τους πιο γρήγορα από κόμβους που βρίσκονται πιο "μακριά" από τον κύριο κόμβο, και τελικά αυτή η γρήγορη εξάντληση θα αδειάσει την μπαταρία τους και έτσι δεν θα μπορεί να συνεχίσει η επικοινωνία του δικτύου με τον κύριο κόμβο, και δια αυτού με τον εξωτερικό κόσμο, εδώ λέμε ότι το δίκτυο έχει "πεθάνει".

Το σχήμα Δ΄.1 αναπαριστά, χρησιμοποιώντας χρωματισμούς για την ενεργειακή κατανάλωση και δείχνοντας την εναπομείνουσα ενέργεια στον κάθε κόμβο, το πρόβλημα της ενεργειακής οπής, με 'Κ' αναφέρουμε τον κύριο κόμβο ως προς τον οποίο προωθούνται τα πακέτα. Όπως βλέπουμε ο μαύρος κόμβος με επίπεδο ενέργειας 0% είναι ακριβώς δίπλα στον κύριο κόμβο, και όσο πιο "μακριά" από τον κύριο κόμβο τόσο πιο υψηλά και τα επίπεδα ενέργειας.

Γιατί όμως το πρόβλημα της ενεργειακής οπής είναι πρόβλημα? Η παρατήρηση που πρέπει να γίνει είναι ότι ενώ το δίκτυο έχει χάσει την δυνατότητα λειτουργίας του, αφού η επικοινωνία έχει διακοπεί με τον έξω κόσμο του δικτύου, η τεράστια πλειοψηφία των αισθητήρων έχει ακόμη άπλετο απόθεμα ενέργειας, είναι μόνο οι γειτονικοί κόμβοι του κυρίου κόμβου που κλασσικά απενεργοποιούνται πρώτοι. Άρα σαν αποτέλεσμα έχουμε ένα αχρηστευμένο δίκτυο, στο οποίο σχεδόν όλοι οι κόμβοι του είναι λειτουργικοί, και όχι μόνο λειτουργικοί αλλά έχουν σχεδόν όλη την μπαταρία τους γεμάτη, αλλά διακόπτεται η λειτουργία του από κάποιους μετρημένους στα δάχτυλα κόμβους, αυτό είναι το πρόβλημα της ενεργειακής οπής [8].

Δ΄.3 Στατική τοποθέτηση υπηρεσίας

Όταν λέμε "στατική τοποθεσία υπηρεσίας" εννοούμε ότι η τοποθεσία του κυρίου κόμβου, ο οποίος έχει τον ρόλο μιας υπηρεσίας, δεν αλλάζει κατά την διάρκεια του πειράματος από την αρχική της τοποθέτηση. Κατασκευάσαμε πειράματα για να αναλύσουμε την συμπεριφορά του προβλήματος της ενεργειακής οπής χρησιμοποιώντας πάλι το εργαλείο omnet++, στο πείραμα κάθε κόμβος-αισθητήρας του δικτύου με μια σταθερή πιθανότητα παραγάγει ένα πακέτο ανά κάθε χρονική στιγμή, το οποίο προωθείται από το πρωτόκολλο δρομολόγησης προς τον κύριο κόμβο μέσω του ελαχίστου μονοπατιού που ενώνει τους δύο κόμβους. Κάθε φορά που κάποιος κόμβος κάνει κάποια λειτουργία, όπως να ενεργοποιηθεί από την λειτουργία χαμηλής ενέργειας (idle), να κάνει κάποια μέτρηση με τους αισθητήρες του, να μεταδώσει ή να λάβει κάποιο μήνυμα από κάποιον άλλο κόμβο, καταναλώνει κάποιο συγκεκριμένο ποσό ενέργειας. Εδώ να σημειωθεί ότι το συντριπτικό



Σχήμα Δ'.2: Μέσος όρος εναπομείνουσας ενέργειας μπαταρίας στην στιγμή θανάτου του δικτύου σε σύγκριση με την απόσταση της αρχικής τοποθέτησης από τον κόμβο της λύσης του 1-median.

ποσοστό της ενεργειακής κατανάλωσης στους κόμβους είναι αποδεδειγμένα η μετάδοση και λήψη δεδομένων, δηλαδή η λειτουργία της κεραίας. Από αυτό μπορούμε να καταλάβουμε ότι η συμπεριφορά του δικτύου από πλευρά της κατανάλωσης ενέργειας καταλήγει τελικά στο πώς οι διάφοροι κόμβοι επικοινωνούν μεταξύ τους, κάθε πότε, και ποιοι μεταδίδουν πιο συχνά από άλλους, και οι υπόλοιπες λειτουργίες των κόμβων μας είναι σχεδόν ενεργειακά αδιάφορες.

Ας κοιτάξουμε τώρα σε κάποια από τα αποτελέσματα των πειραμάτων στατικής τοποθέτησης της υπηρεσίας σε διάφορες θέσεις μέσα στο δίκτυο ασύρματων αισθητήρων. Σε όλα τα παρακάτω διαγράμματα ο χ-άξονας αντιπροσωπεύει την απόσταση της στατικής αρχικής τοποθέτησης (με απόσταση εννοούμε όχι την ευκλείδεια αλλά το συνολικό κόστος του ελαχίστου μονοπατιού) από την λύση του 1-median για το δίκτυο, ο γράφος του δικτύου παράγεται με ψευδοτυχαίο πιθανοτικό τρόπο με τον ίδιο seed αριθμό κάθε φορά ώστε να έχουμε μια σταθερή όψη του δικτύου για τα διαγράμματα. Για όλα τα διαγράμματα πάρθηκαν υπόψη όλες οι πιθανές τοποθετήσεις, συμπεριλαμβανομένης και της λύσης του 1-median προβλήματος.

Δ΄.3.1 Μέσος όρος εναπομείνουσας ενέργειας

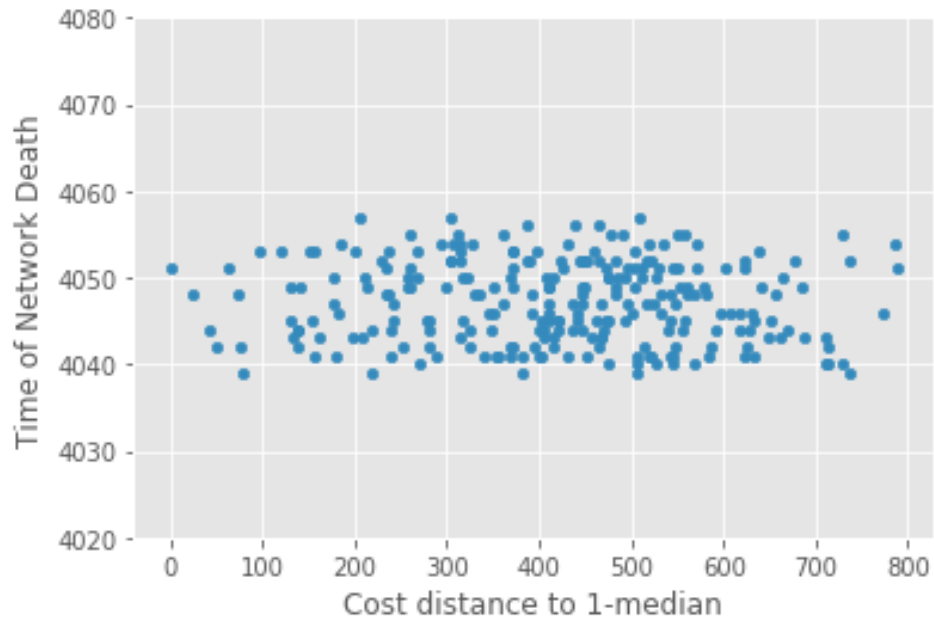
Στο σχήμα Δ΄.2 βλέπουμε το γράφημα του μέσου όρου της εναπομείνουσας ενέργειας της μπαταρίας όλων των κόμβων του δικτύου για όλες τις πιθανές τοποθετήσεις, η αρχική στάθμη της μπαταρίας κάθε κόμβου τέθηκε στις 1 εκατομμύριο μονάδες. Ας αναλύσουμε τι βλέπουμε στο διάγραμμα: γενικά καταλαβαίνουμε ότι ακόμα και στην χειρότερη περίπτωση το δίκτυο σαν μέσο όρο έχει περίπου το 97% της ενέργειας του ανέγγιχτη, στην καλύτερη περίπτωση κοιτάμε στο 98.4%. Επίσης αξιοσημείωτο είναι το γεγονός ότι η μέγιστη σε μέσο όρο ενέργεια παρατηρείται όταν η αρχική τοποθέτηση της υπηρεσίας έχει απόσταση από την λύση του 1-median ίση με μηδέν, δηλαδή όταν βρίσκεται στην λύση του 1-median έχουμε την μεγαλύτερη εναπομείνουσα ενέργεια στο δίκτυο, αυτό εξηγείται εύκολα αν καταλάβουμε ότι η δύναμη του σήματος μετάδοσης είναι ανάλογη της δυσκολίας μετάδοσης από κόμβο σε κόμβο, αφού το συνολικό κόστος όλου του δικτύου είναι ελάχιστο στην λύση του 1-median προβλήματος, στην ίδια τοποθέτηση επίσης θα ελαχιστοποιείται και κατανάλωση ενέργειας του δικτύου. Η προηγούμενη παρατήρηση μπορούμε να δούμε εύκολα πως μπορεί να συνδέεται με την μετεγκατάσταση υπηρεσίας, αφού εκείνη η πολιτική μετακινεί την υπηρεσία προς την λύση του 1-median προβλήματος, όπου και βρίσκεται η βέλτιστη τοποθέτηση από ενεργειακής άποψης.

Δ΄.3.2 Χρόνος "θανάτου" δικτύου

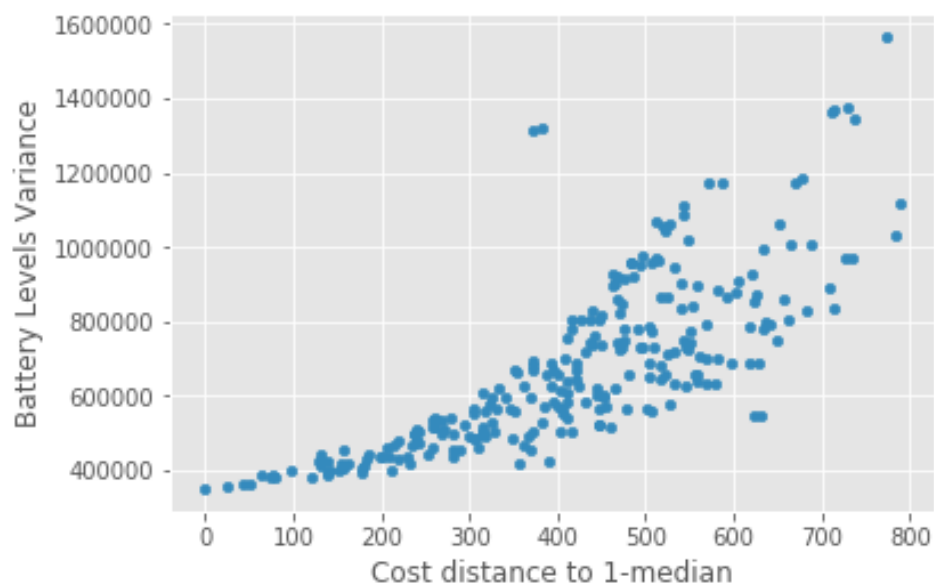
Στο σχήμα Δ΄.3 μετράμε στον χ-άξονα την απόσταση των τοποθετήσεων από την λύση του 1-median, και στον ψ-άξονα μετράμε χρόνο σε δευτερόλεπτα. Η κύρια παρατήρηση εδώ είναι ότι η μέγιστη τιμή του γραφήματος με την ελάχιστη διαφέρουν μόλις 1%, δηλαδή η τοποθέτηση της υπηρεσίας πάνω στο δίκτυο δεν παίζει κανένα ρόλο στο πότε θα πεθάνει το δίκτυο βάσει του προβλήματος της ενεργειακής οπής. Αυτό ακριβώς το γεγονός θα προσπαθήσουμε να πολεμήσουμε στην επόμενη ενότητα του κεφαλαίου με την χρήση της μετεγκατάστασης υπηρεσίας, με μικτά αποτελέσματα. Τελικά, το συγκεκριμένο διάγραμμα δεν μας δίνει κάποια περαιτέρω πολύτιμη πληροφορία πέρα από αυτή την παρατήρηση, αφού θυμίζει θόρυβο και όχι κάτι με κάποια δομημένη μορφή.

Δ΄.3.3 Διακύμανση εναπομείνουσας ενέργειας

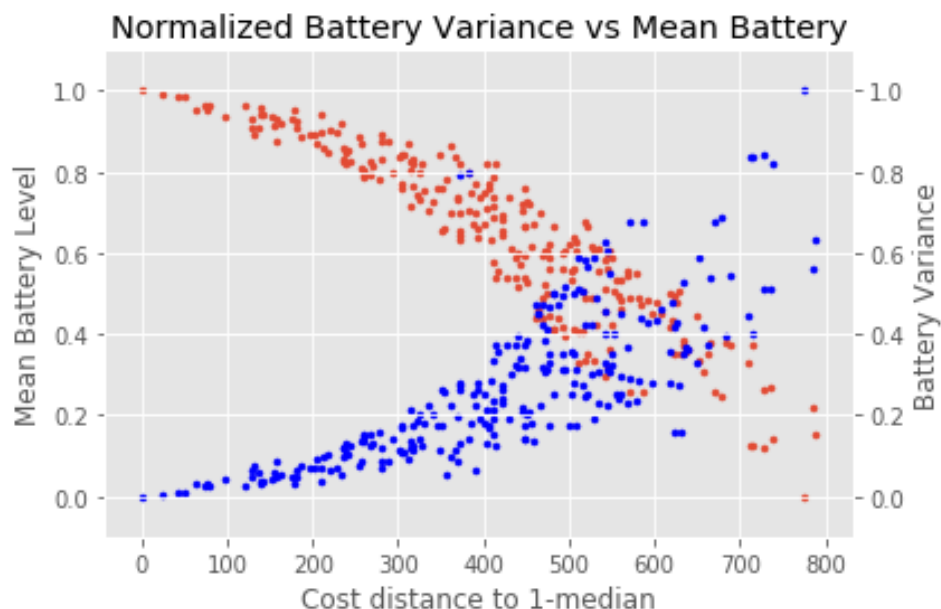
Η διακύμανση είναι το τετράγωνο της τυπικής απόκλισης, δηλαδή το άθροισμα των τετράγωνων της διαφοράς των τιμών κάθε επιπέδου μπαταρίας όλων των



Σχήμα Δ'3: Χρόνος θανάτου δικτύου σε συνάρτηση με την απόσταση της αρχικής τοποθέτησης από τον κόμβο της λύσης του 1-median.



Σχήμα Δ'4: Διακύμανση επιπέδων μπαταρίας σε όλο το δίκτυο στην στιγμή θανάτου σε σύγκριση με την απόσταση της αρχικής τοποθέτησης από τον κόμβο της λύσης του 1-median.



Σχήμα Δ΄.5: Κανονικοποιημένα αποτελέσματα διακύμανση εναπομείνουσας ενέργειας και μέσου όρου εναπομείνουσας ενέργειας.

κόμβων από τον μέσο όρο τους. Αν λάβουμε υπόψη το διάγραμμα Δ΄.2 του μέσου όρου μπαταρίας τότε θα περιμέναμε να δούμε μια αντίθετη πορεία της διακύμανσης από αυτή του μέσου όρου, αφού στο διάγραμμα Δ΄.3 έχουμε ότι ο χρόνος ζωής του δικτύου είναι περίπου ο ίδιος σε όλες τις τοποθετήσεις. Άρα αν έχουμε σταθερό χρόνο θανάτου, και όλο μειωμένο μέση τιμή μπαταρίας, το άθροισμα της διαφοράς των τιμών με τον μέσο όρο θα αυξάνει άρα θα αυξάνει και η διακύμανση όσο απομακρυνόμαστε από την λύση του 1-median.

Δ΄.3.4 Κανονικοποιημένα αποτελέσματα

Τελικά ας δούμε μια αντιπαράθεση των δύο πιο σημαντικών αποτελεσμάτων, τον μέσο όρο ενέργειας, και την διακύμανση ενέργειας. Έχουμε κοινωνικοποιήσει και τα δύο σετ αποτελεσμάτων και τα αντιπαραθέτουμε σε ένα διάγραμμα μαζί. Με αποτέλεσμα να έχουμε πιο καθαρά σε αυτό το γράφημα την συσχέτιση των δύο αυτών τιμών.

Δ'.4 Συμβάντα

Προσπαθούμε να καταλάβουμε τι επίδραση θα είχε ένα δυναμικό γεγονός στον πραγματικό κόσμο (πχ. μια δασική πυρκαγιά [11]) πάνω στο δίκτυο αισθητήρων. Θα προσημειώσουμε αυτό το γεγονός με τον εξής τρόπο, σε κάποια χρονική στιγμή t μικρότερη από τον χρόνο θανάτου του δικτύου μια κυκλική περιοχή μέσα στο δίκτυο με κάποια ακτίνα r θα διαλεχτεί ως την περιοχή του συμβάντος. Το συμβάν θα αντιπροσωπευθεί σαν την ραγδαία αύξηση της ροής πακέτων προερχόμενα από την περιοχή του συμβάντος. Τα πακέτα αυτά δεν είναι ιδιαίτερα, απλά είναι περισσότερα από τον κανονικό αριθμό που παράγει το δίκτυο σε μέσο όρο, αυτό θα το πετύχουμε αυξάνοντας την πιθανότητα δημιουργίας πακέτου ανά κάθε χρονική στιγμή σε 1 (βλέπε σχήμα Δ'.6), δηλαδή κάθε χρονική στιγμή που περνά στο δίκτυο όλοι οι κόμβοι που βρίσκονται στην περιοχή θα παράξουν από ένα πακέτο που κατευθύνεται προς την υπηρεσία/κύριο κόμβο.

Δ'.4.1 Στατικά συμβάντα

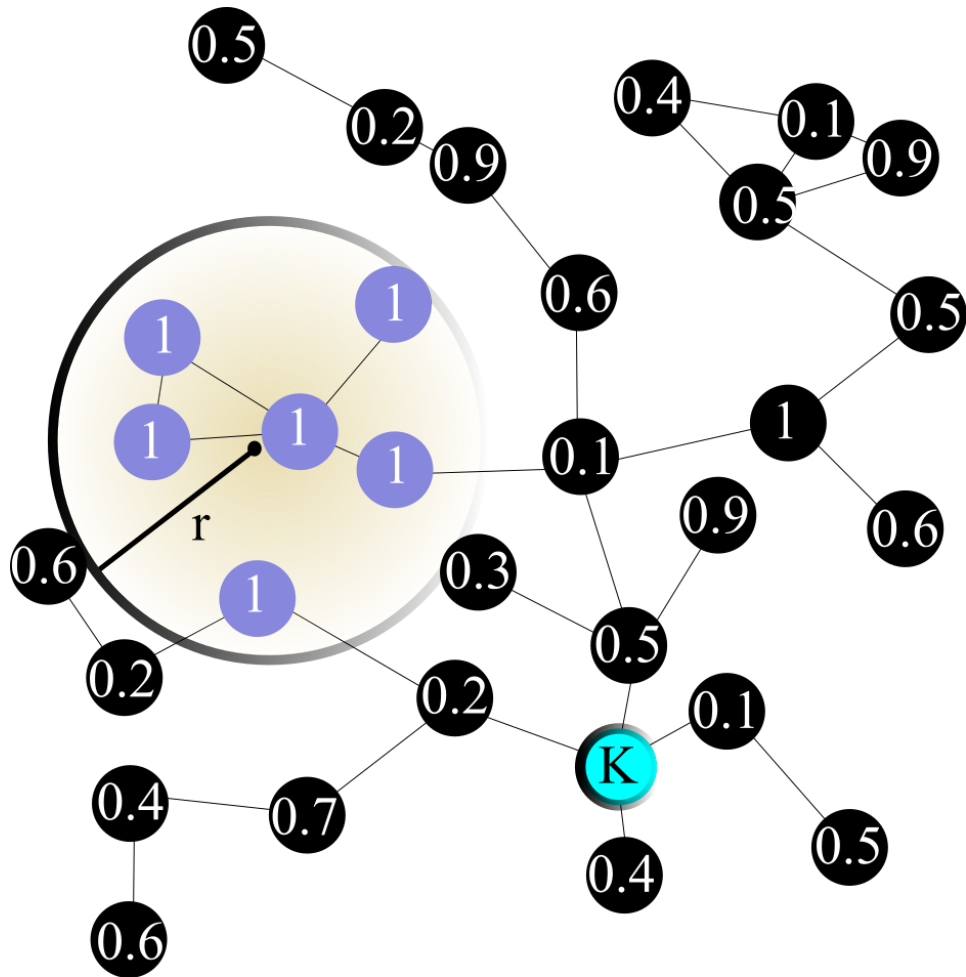
Στατικό ονομάζουμε ένα συμβάν όταν η περιοχή από όπου ξεκινά και επηρεάζει παραμένει σταθερή σε ισχύ, τοποθεσία, και ακτίνα του γεγονότος. Είναι η πιο απλή περίπτωση ενός συμβάντος και είναι κάπως ιδανική, τα πραγματικά γεγονότα συμπεριφέρονται δυναμικά, εξελίσσονται και αλλάζουν συνεχώς, το οποίο είναι και κάτι που θα αναλύσουμε στο επόμενο τμήμα του κεφαλαίου.

Όπως βλέπουμε στα διαγράμματα Δ'.7, Δ'.8, Δ'.9, και Δ'.10 ένα στατικό συμβάν δεν επηρεάζει σχεδόν καθόλου τα αποτελέσματα που είχαμε για την έκδοση χωρίς την περιοχή συμβάντος.

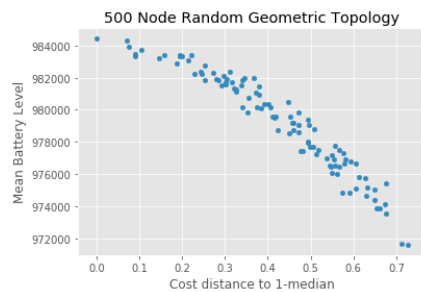
Δ'.4.2 Κινούμενα συμβάντα

Πολύ πιο ενδιαφέρον έχουν τα κινούμενα συμβάντα, σε αυτή την περίπτωση το συμβάν ξεκινά όπως στα στατικά συμβάντα αλλά ξεκινά μια πορεία κίνησης και ενεργοποιεί τις επιδράσεις του συμβάντος σε όλους τους κόμβους που συναντά. Σε αυτή τη φάση η κίνηση είναι μια απλή κυκλική πορεία πάνω από τα εξωτερικά όρια του δικτύου. Το σύστημα των δυναμικών συμβάντων υλοποιήθηκε με επεκτάσιμο τρόπο έτσι ώστε με μια απλή γεννήτρια πορείας να μπορεί ο καθένας να δημιουργεί την δική του πορεία και εξέλιξη της περιοχής του συμβάντος.

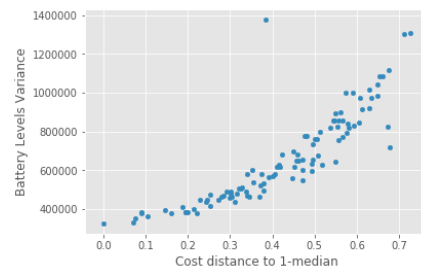
Στα διαγράμματα Δ'.11 και Δ'.12, Δ'.13 σαν στατικά αποτελέσματα χρησιμοποιούμε τα αποτελέσματα από τα διαγράμματα Δ'.7, Δ'.8, και Δ'.9 αντίστοιχα. Θέτουμε επί τίνος αποτελεσμάτων τα καινούργια αποτελέσματα των πειραμάτων με



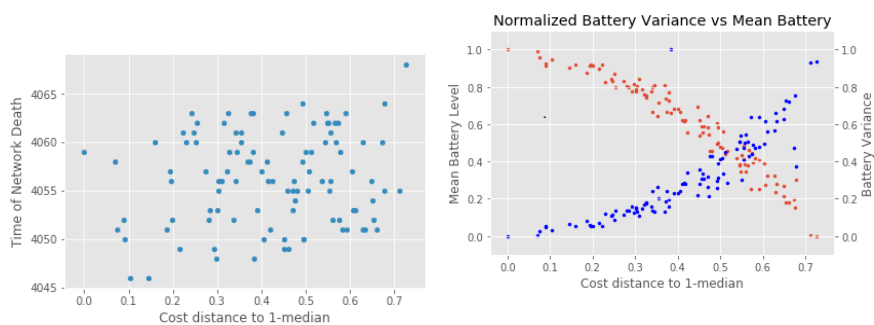
Σχήμα Δ΄.6: Σχηματική απεικόνιση μιας περιοχής συμβάντος, φαίνεται η ακτίνα r και το κέντρο του κύκλου που ορίζει την περιοχή, και οι πιθανότητες δημιουργίας πακέτου ανά χρονική στιγμή του κάθε κόμβου.



Σχήμα Δ΄.7: (Στατικό συμβάν) Μέσος όρος μπαταρίας την στιγμή θανάτου του δικτύου.

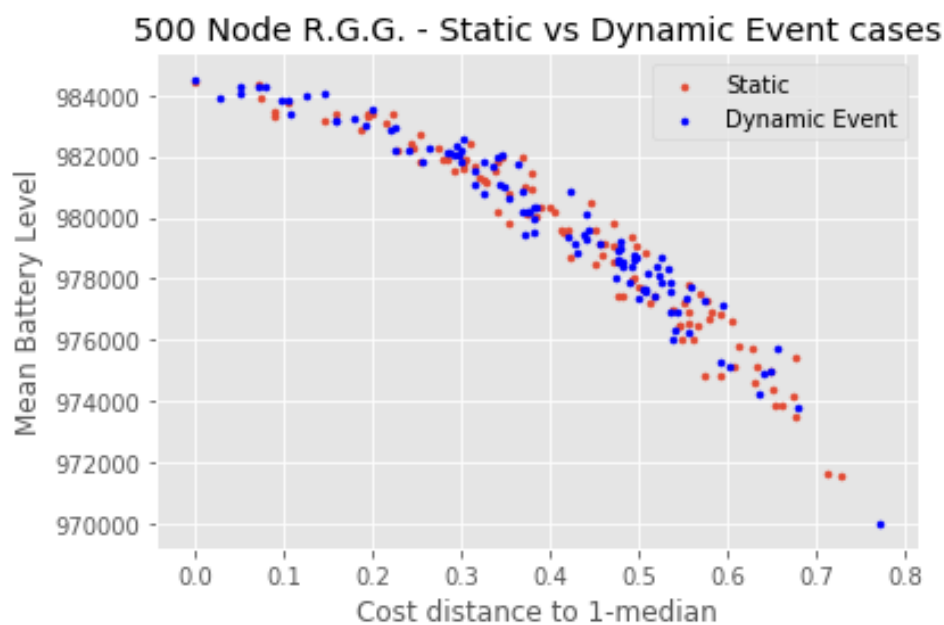


Σχήμα Δ΄.8: (Στατικό συμβάν) Διακύμανση επιπέδων μπαταρίας σε όλο το δίκτυο την στιγμή θανάτου.

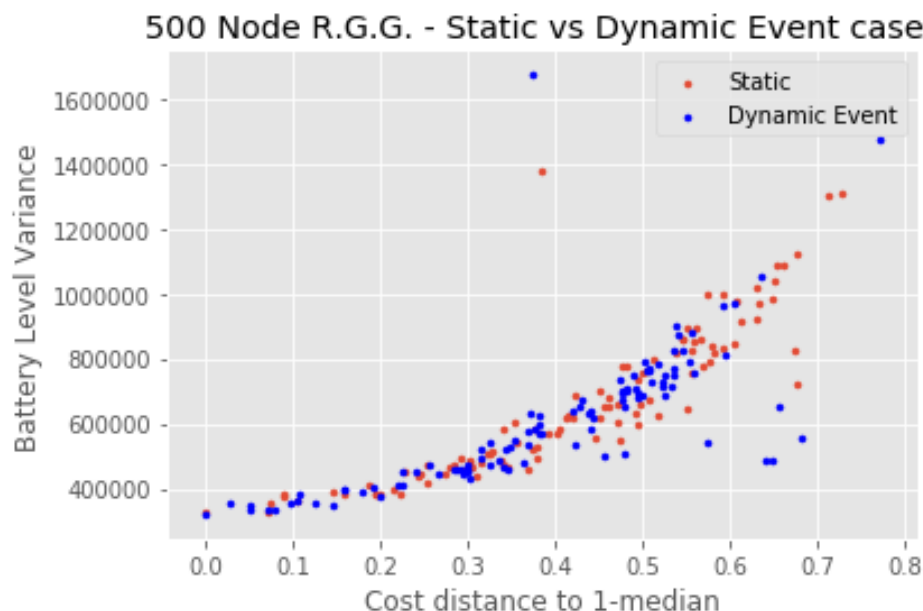


Σχήμα Δ'.9: (Στατικό συμβάν) Χρόνος θανάτου του δικτύου.

Σχήμα Δ'.10: (Στατικό συμβάν) Κανονικοποιημένα αποτελέσματα διακύμανσης και μέσου όρου εναπομείνουσας ενέργειας.



Σχήμα Δ'.11: Μέσος όρος επιπέδων μπαταρίας σε όλο το δίκτυο στην στιγμή θανάτου σε σύγκριση με την απόσταση της αρχικής τοποθέτησης από τον κόμβο της λύσης του 1-median για στατικό (ακίνητο) και δυναμικό (κινούμενο) συμβάν.

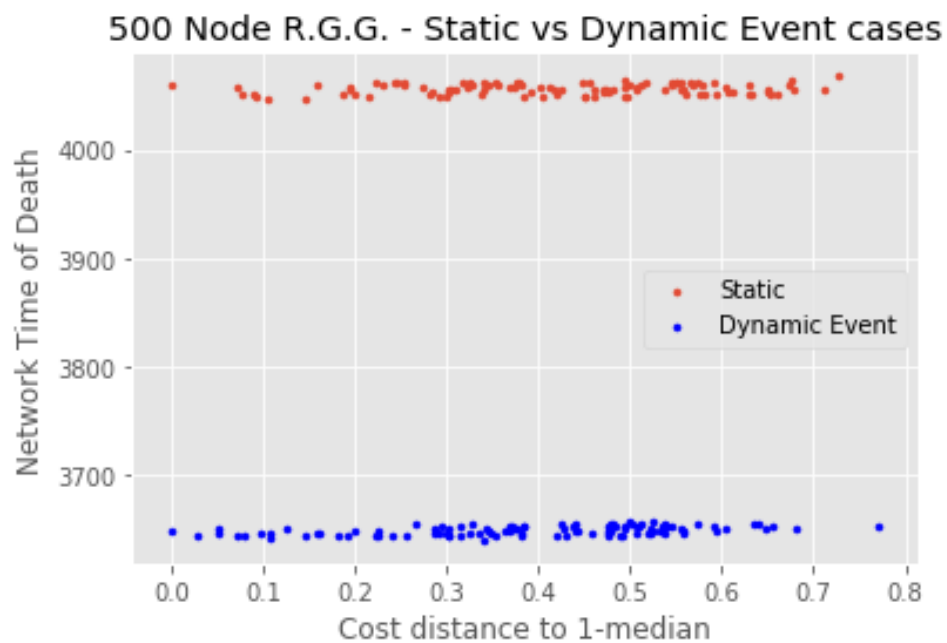


Σχήμα Δ΄.12: Διακύμανση επιπέδων μπαταρίας σε όλο το δίκτυο στην στιγμή θανάτου σε σύγκριση με την απόσταση της αρχικής τοποθέτησης από τον κόμβο της λύσης του 1-median για στατικό (ακίνητο) και δυναμικό (κινούμενο) συμβάν.

το δυναμικό συμβάν για εύκολη σύγκριση. Τα δύο πρώτα διαγράμματα δεν μας δίνουν κάποια διαφοροποίηση μεταξύ των δύο οπότε μπορούμε να τα αγνοήσουμε. Το διάγραμμα Δ΄.13 όμως δείχνει κάτι διαφορετικό, για την στατική έκδοση του πειράματος, ο χρόνος θανάτου είναι γύρω στα 4,070 δευτερόλεπτα, ενώ με την δυναμική έκδοση του πειράματος, κινείται γύρω στα 3,630 δευτερόλεπτα.

Ερμηνεία των αποτελεσμάτων δυναμικού συμβάντος

Το πρόβλημα με αυτά τα αποτελέσματα είναι ότι αν τα πάρουμε ξεχωριστά δεν διαφέρουν, δηλαδή και τα δύο μας λένε την ίδια ιστορία, ότι όπου και να τοποθετήσουμε την υπηρεσία πάνω στο δίκτυο, ο χρόνος θανάτου του δικτύου θα είναι σταθερός, ακόμα και αν έχουμε κάποιο συμβάν, στατικό ή δυναμικό, ή και ακόμα χωρίς κάποιο συμβάν όπως είδαμε πιο πάνω στο διάγραμμα Δ΄.3. Εδώ η σύγκριση μπορεί με μια πρώτη ματιά να μας δείχνει ότι έχουμε μειώσει ή αυξήσει με κάποιον τρόπο την ζωή του δικτύου, αλλά αυτό είναι απλά λάθος, γιατί είναι το είδος του συμβάντος που άλλαξε και όχι κάτι που κάναμε εμείς, αυτό που μας λέει το σχήμα είναι ότι και στις δύο περιπτώσεις είμαστε το ίδιο. Άλλους τρόπους αύξησης της ζωής του δικτύου θα κοιτάξουμε στην επόμενη ενότητα του κεφαλαίου.



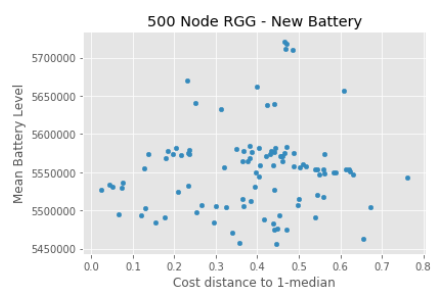
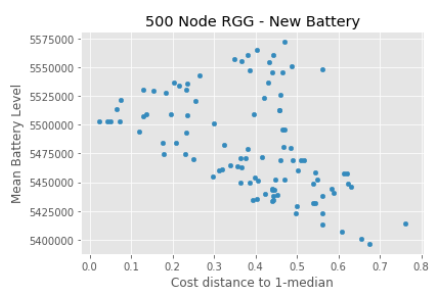
Σχήμα Δ'.13: Στιγμή θανάτου σε σύγκριση με την απόσταση της αρχικής τοποθέτησης από τον κόμβο της λύσης του 1-median για στατικό (ακίνητο) και δυναμικό (κινούμενο) συμβάν.

Δ΄.5 Συμβάντα και μετεγκατάσταση υπηρεσίας

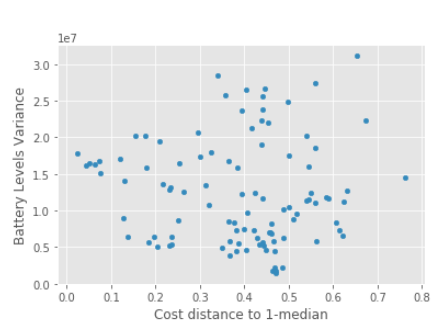
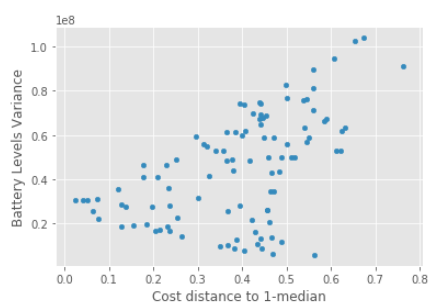
Ας επεκτείνουμε τις προσομοιώσεις που κάναμε στην προηγούμενη ενότητα στο να περιέχουν και τον μηχανισμό της μετεγκατάστασης της υπηρεσίας, άρα τώρα δεν έχουμε μόνο μια περιοχή αυξημένης δραστηριότητας, αλλά έχουμε και κίνηση της υπηρεσίας. Στο προηγούμενο κεφάλαιο εισάγαμε την έννοια του μηχανισμού μετεγκατάστασης υπηρεσίας, εκεί είχαμε αναφέρει ότι σε ένα κανονικό δίκτυο η πολιτική μετεγκατάστασης θα μετακινεί την υπηρεσία σε τοποθετήσεις μειωμένου κόστους, και θα το κάνει μονοτονικά. Σε γενικές γραμμές αν μπορούμε να πούμε ότι η πολιτική μετεγκατάστασης μετακινεί την υπηρεσία προς την λύση του 1-median προβλήματος, τότε σε αυτή την παραλλαγή του δικτύου με την περιοχή του συμβάντος, μπορούμε να πούμε ότι γενικά, και ανάλογα με την ισχύ και ακτίνα του συμβάντος, η πολιτική θα μετακινήσει την υπηρεσία προς την περιοχή της αυξημένης δραστηριότητας. Αυτή η παρατήρηση δεν είναι κάτι το παράξενο αν σκεφτούμε ότι αν υπολογίσουμε ξανά την λύση του 1-median προβλήματος μετά την έναρξη του συμβάντος, η λύση θα έχει σχεδόν σίγουρα αλλάξει θέση, και σίγουρα θα έχει προσέλθει πιο κοντά στην περιοχή του συμβάντος, άρα πάλι η πολιτική προς την λύση του 1-median μετακινεί την υπηρεσία, απλά αυτή η θέση τείνει να φτάνει όλο και πιο κοντά στην περιοχή του συμβάντος όσο αυξάνει η ισχύς του ή και η ακτίνα επιρροής του.

Θα κοιτάξουμε την συμπεριφορά του δικτύου σε μια καινούργια πορεία με ένα δυναμικό κινούμενο συμβάν. Ξεκινάμε με τα διαγράμματα Δ΄.14 και Δ΄.15, όπου το πρώτο είναι με απενεργοποιημένο τον μηχανισμό μετεγκατάστασης, και το δεύτερο με τον μηχανισμό ενεργοποιημένο. Αντίστοιχα είναι τα διαγράμματα Δ΄.16 και Δ΄.17 για τις διακυμάνσεις, όπως και τα Δ΄.18 και Δ΄.19 για τους χρόνους θανάτου του δικτύου.

Στους μέσους όρους εναπομείνας ενέργειας (σχήμα Δ΄.14 και σχήμα Δ΄.15) βλέπουμε ότι η κίνηση της περιοχής του συμβάντος σε συνδυασμό με τον μηχανισμό μετεγκατάστασης της υπηρεσίας μας δίνουν τυχαία αποτελέσματα, δεν μπορούμε να τα χρησιμοποιήσουμε για κάποιο συμπέρασμα σε αυτή την φάση. Ακόμα και στο πρώτο σχήμα που η λειτουργία της μετεγκατάστασης δεν έχει ενεργοποιηθεί τα αποτελέσματα δεν μας οδηγούν σε κανένα συμπέρασμα. Το ίδιο μπορούμε να πούμε και για τις διακυμάνσεις των ενεργειών (σχήμα Δ΄.16 και σχήμα Δ΄.17) αφού οι τιμές των διακυμάνσεων βασίζονται στους μέσους όρους που θα παράγει το δίκτυο στην στιγμή του θανάτου. Και τελικά ούτε τα αποτελέσματα των χρόνων ζωής των δικτύων μας φέρνουν πιο κοντά σε κάποιο συμπέρασμα για την οποία-

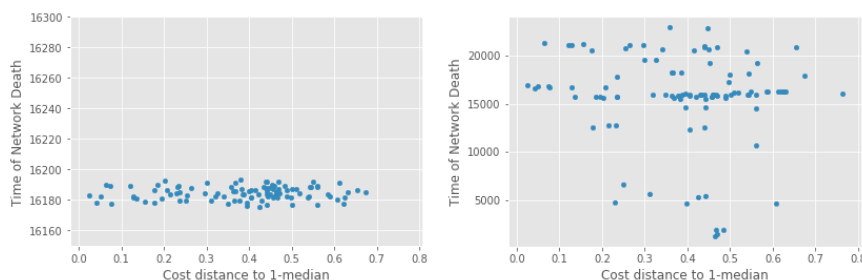


Σχήμα Δ'.14: (Χωρίς μετεγκατά- Σχήμα Δ'.15: (Με μετεγκατάσταση) Μέσος όρος μπαταρίας. Μέσος όρος μπαταρίας.



Σχήμα Δ'.16: (Χωρίς μετεγκατά- Σχήμα Δ'.17: (Με μετεγκατάσταση) Διακύμανση επιπέδων μπα- Διακύμανση επιπέδων μπαταρίας. ταρίας.

δήποτε βελτιστοποίηση των δικτύων.



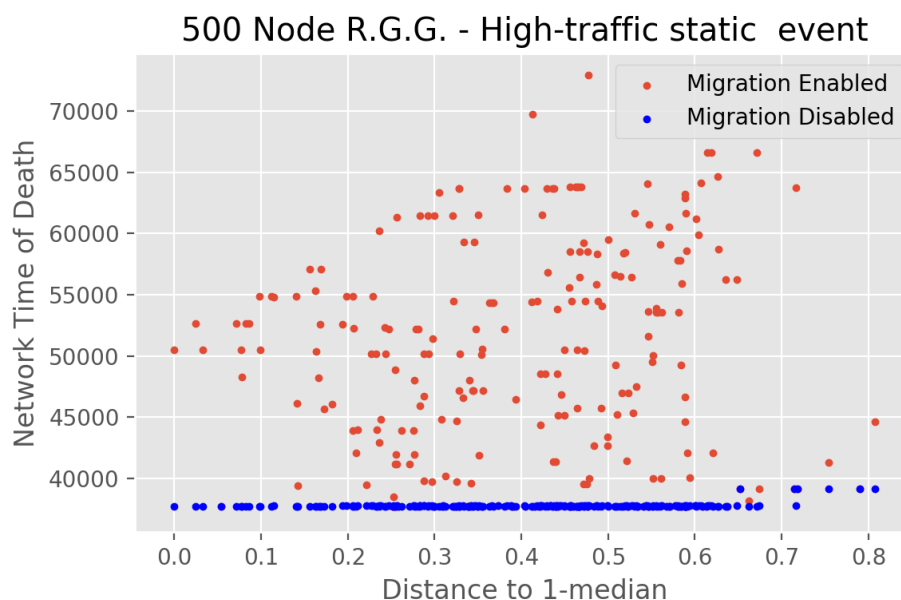
Σχήμα Δ΄.18: (Χωρίς μετεγκατάσταση) Χρονική στιγμή θανάτου. Σχήμα Δ΄.19: (Με μετεγκατάσταση) Χρονική στιγμή θανάτου.

Δ΄.6 Σύγκριση χρονισμών μετεγκατάστασης

Στην προηγούμενη ενότητα, μια από τις παραμέτρους του μηχανισμού μετεγκατάστασης παρέμενε σταθερή, η χρονική διάρκεια συλλογής στατιστικών μετρήσεων της κίνησης στην υπηρεσία μέχρι να παρθεί η απόφαση για μετεγκατάσταση δια μέσω της επαλήθευσης του κριτηρίου μετεγκατάστασης. Σε αυτή την ενότητα θα αναλύσουμε κάποιες διάφορες τιμές σε αυτή την παράμετρο, και θα δούμε πως αυτές οι τιμές επηρεάζουν την λειτουργία και κατάληξη του δικτύου.

Σύγκριση με ή χωρίς μετεγκατάσταση

Αρχικά δοκιμάσαμε δύο διαφορετικές εκδοχές, μία με την λειτουργία της μετεγκατάστασης ενεργοποιημένη και μία με την μετεγκατάσταση απενεργοποιημένη. Το διάγραμμα Δ΄.20 δείχνει τα αποτελέσματα και των δύο προσομοιώσεων, στην περίπτωση που ο μηχανισμός μετεγκατάστασης είναι απενεργοποιημένος έχουμε τα αποτελέσματα που είδαμε τουλάχιστον δύο φορές πριν, δηλαδή όπου και να τοποθετήσουμε την υπηρεσία αρχικά δεν βλέπουμε καμία διαφορά στον χρόνο ζωής του δικτύου. Σε κόκκινο χρώμα έχουμε τα δεδομένα των αποτελεσμάτων από τις προσομοιώσεις όπου έχουμε ενεργοποιήσει την μετεγκατάσταση υπηρεσίας. Παρατηρούμε ότι σε όλες τις περιπτώσεις ο χρόνος ζωής του δικτύου βελτιώνεται, σε κάποιες περιπτώσεις έχουμε και διπλασιασμό της ζωής του δικτύου. Αρχικά θα λέγαμε ότι καταφέραμε αυτό που θέλαμε, δηλαδή να αυξήσουμε την ζωή του δικτύου, αλλά με λίγο παραπάνω σκέψη φτάνουμε στο συμπέρασμα ότι αφού δεν βλέπουμε κάποιο συγκεκριμένο μοτίβο στον χρόνο ζωής με την μετεγκατάσταση ενεργοποιημένη, ότι η αύξηση της ζωής προέρχεται απλά και μόνο από την ενεργοποίηση του μηχανισμού της μετεγκατάστασης, και από την φυσική κίνηση της υπηρεσίας σε διαφορετικούς κόμβους διαμοιράσουμε το βάρος σε πολλούς διαφο-



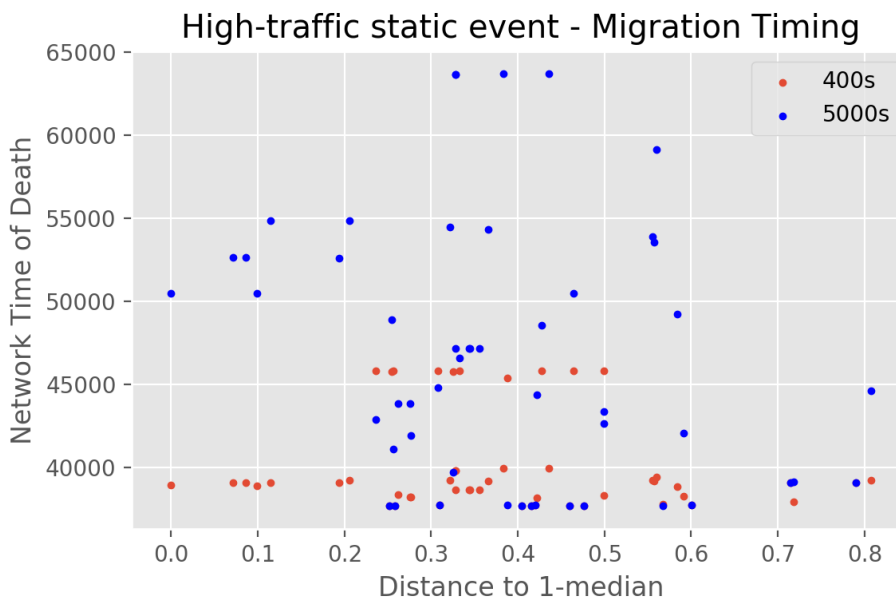
Σχήμα Δ'20: Χρόνος θανάτου δικτύου σε δευτερόλεπτα με τον μηχανισμό μετεγκατάστασης ενεργοποιημένο και απενεργοποιημένο σε κόκκινο και μπλε αντίστοιχα.

ρετικούς κόμβους και κατά συνέπεια αυξάνουμε την ζωή του δικτύου.

Μελέτη χρονισμών μετεγκατάστασης

Στα διαγράμματα Δ'21 και Δ'22 θέτουμε την παράμετρο του χρονισμού του μηχανισμού της μετεγκατάστασης σε 400 δευτερόλεπτα και 5,000 δευτερόλεπτα, και σε 100 δευτερόλεπτα και 10,000 δευτερόλεπτα αντίστοιχα. Το συμβάν θέτεται σαν την αρχική του διάταξη, δηλαδή μια σταθερή περιοχή αυξημένης δραστηριότητας χωρίς να κινείται και χωρίς αλλαγές στην ισχύ της. Βλέπουμε ότι σαν κανόνα όσο παραπάνω περιμένουμε να κάνουμε την μετεγκατάσταση της υπηρεσίας κάθε φορά που είναι να ελέγξουμε το κριτήριο, είναι ευεργετικό να περιμένουμε ένα μεγαλύτερο χρονικό διάστημα πάρα μικρότερο, εκτός αν αυτό το χρονικό διάστημα ξεπερνά τον χρόνο ζωής του δικτύου στην οποία περίπτωση δεν θα προλάβει το δίκτυο να εκτελέσει καμία μετεγκατάσταση πριν την στιγμή του θανάτου του.

Γενικά βλέπουμε ότι όταν έχουμε σχετικά μικρούς χρονισμούς στον μηχανισμό μετεγκατάστασης σε σύγκριση με τον χρόνο ζωής του δικτύου, η μετεγκατάσταση δεν βοηθά στην παράταση του χρόνου ζωής του δικτύου σε κάποια βαρυσήμαντη χρονική διάρκεια. Ο λόγος για το προηγούμενο είναι απλά ότι η πολιτική μετεγκα-



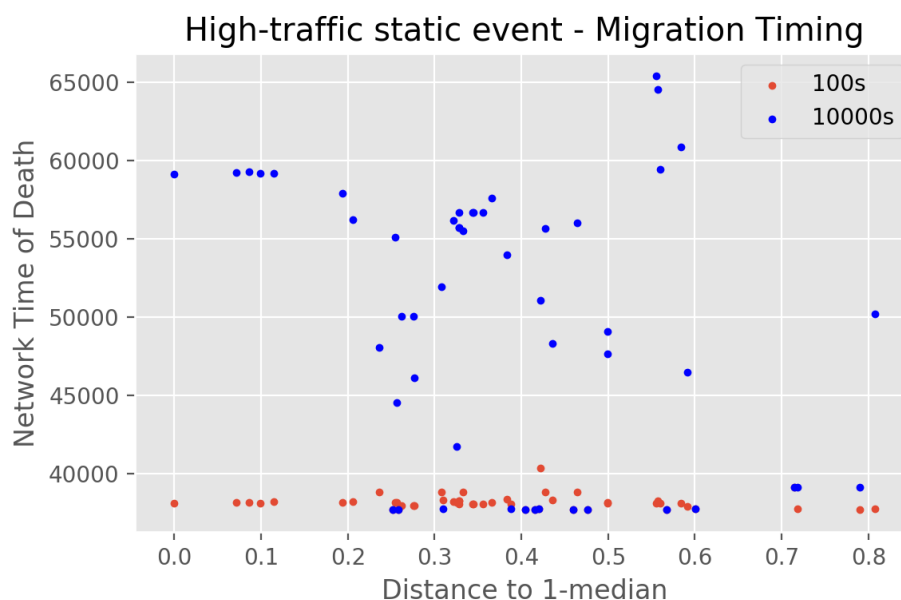
Σχήμα Δ΄.21: Χρόνος θανάτου δικτύου σε δευτερόλεπτα με τον χρονισμό του μη-χανισμού μετεγκατάστασης στα 400 δευτερόλεπτα και 5,000 δευτερόλεπτα.

τάστασης σε ένα σταθερό δίκτυο έχει κάποιο συγκεκριμένο αριθμό μετακινήσεων που μπορεί να εκτελέσει μέχρι το κριτήριο της μετεγκατάστασης να σταματήσει να πληρείτε και η υπηρεσία να μείνει στάσιμη σε εκείνο τον κόμβο μέχρι το πέρας του δικτύου. Άρα αν οι κινήσεις όλες εκτελεστούν σε σχετικά μικρό χρονικό διάστημα σε σχέση με την συνολική ζωή του δικτύου τότε θα έχουν πολύ μικρή επίδραση πάνω σε αυτή.

Δ΄.7 Ανοιχτά ερωτήματα για μελλοντική πορεία

Δυναμικά συμβάντα και οι εφαρμογές στην πραγματικότητα

Μεγάλη προσοχή δώσαμε στα δυναμικά γεγονότα διότι είναι βασισμένα στην πραγματικότητα, μια δασική πυρκαγιά δεν παραμένει σε ένα σημείο αλλά εξαπλώνεται και μετακινείται, αλλάζει πορείες ακολουθώντας τον άνεμο, ένα πολεμικό άρμα μετακινείται, παίρνει θέση και στοχεύει, μετά αλλάζει την θέση του στο πεδίο της μάχης. Είναι η φύση των πραγμάτων να είναι δυναμικά και όχι στατικά, και αυτό πρέπει να ανακλαστεί στην έρευνα κάπως και να μελετηθεί, πολλές πιθανότητες για βελτιστοποίηση ασυρμάτων δικτύων αισθητήρων υπάρχουν γύρω από



Σχήμα Δ'22: Χρόνος θανάτου δικτύου σε δευτερόλεπτα με τον χρονισμό του μηχανισμού μετεγκατάστασης στα 100 δευτερόλεπτα και 10,000 δευτερόλεπτα.

αυτόν τον τομέα.

Παράταση χρόνου ζωής ασυρμάτων δικτύων αισθητήρων

Ο χρόνος ζωής ενός δικτύου αισθητήρων είναι ίσως το πιο σημαντικό στοιχείο βελτιστοποίησης του που υπάρχει [6], στην εργασία αυτή είδαμε ότι οι αρχικές τοποθετήσεις να μην βοηθούν στα ενεργειακά αποθέματα του δικτύου σαν σύνολο, αλλά το πρόβλημα της ενεργειακής οπής καθιστά αυτό το γεγονός άχρηστο πρακτικά αφού το δίκτυο πεθαίνει πολύ πιο γρήγορα από ότι καταναλώνεται η συνολική ενέργεια που υπάρχει σε όλους τους κόμβους του. Ο μηχανισμός μετεγκατάστασης αυξάνει τον χρόνο ζωής του δικτύου αλλά το ίδιο θα έκανε και η τυχαία κίνηση της υπηρεσίας, άρα βρισκόμαστε πάλι από εκεί που αρχίσαμε, με το πρόβλημα της ενεργειακής οπής να βρίσκεται μπροστά μας. Ειδικά αν μιλήσουμε για πολιτικές επαναφόρτισης, πολλαπλές υπηρεσίες, αλλαγές τοπολογίας, κτλ, τότε πρόβλημα γίνεται πολύ πιο περίπλοκο αλλά χρήζει λύσης.

Κεφάλαιο Ε΄

Συμπεράσματα

Σε αυτή την πτυχιακή εργασία καταφέραμε να λύσουμε το 1-median πρόβλημα σε τυχαίους γεωμετρικούς γράφους, πλεγματικές τοπολογίες, και δέντρα, χρησιμοποιώντας την βιβλιοθήκη προσομοιώσεων Omnet++. Τα ασύρματα δίκτυα αισθητήρων είναι η αρχέτυπη μορφή ενός δυναμικού δικτύου με πάρα πολλούς κόμβους, όμως η δυσκολία και η μη πρακτικότητα στην λύση του 1-median προβλήματος σε πολύ μεγάλα και δυναμικά δίκτυα όπως αυτά, μας έθεσαν σε πορεία να βρούμε άλλους τρόπους βελτιστοποίησης της τοποθέτησης της υπηρεσίας σε ένα δίκτυο. Υλοποιήσαμε λοιπόν τον μηχανισμό της μετεγκατάστασης υπηρεσίας, ο οποίος μέσω στατιστικών μετρήσεων της εισερχόμενης κίνησης προς τον κόμβο της υπηρεσίας ελέγχει το κριτήριο μετεγκατάστασης και μετακινεί την υπηρεσία από κόμβο σε κόμβο δυναμικά και ανάλογα με την κίνηση των πακέτων, σε μια μονοτονικά φθίνουσα καμπύλη κόστους τοποθεσίας για την υπηρεσία. Τρέξαμε πειράματα του μηχανισμού μετεγκατάστασης σε διάφορες τοπολογίες ασύρματων δικτύων και βρήκαμε ότι όντως ο μηχανισμός και το κριτήριο αποδίδουν δραματικές μειώσεις στο κόστος της υπηρεσίας με την οποιαδήποτε πιθανή αρχική τοποθέτηση και ειδικά σε πλεγματικές τοπολογίες.

Έπειτα κοιτάξαμε από τι αποτελείται ένα ασύρματο δίκτυο αισθητήρων, και τα προβλήματα που αντιμετωπίζουν οι σχεδιαστές τους σε σχέση με την κατανάλωση ενέργειας των κόμβων και το πρόβλημα της ενεργειακής οπής και το οποίο προσομοιώσαμε και προσπαθήσαμε να καταλάβουμε. Τρέξαμε πειράματα για την μελέτη του προβλήματος της ενεργειακής οπής αρχικά με απλές στατικές τοποθετήσεις και παρατηρήσαμε ότι όταν η τοποθέτηση της υπηρεσίας γίνεται στον κόμβο της λύσης του 1-median προβλήματος, η εναπομείνουσα ενέργεια και η διακύμανση της ήταν η μέγιστη και ελάχιστη αντίστοιχα. Όποτε μας έβαλε στο μονοπάτι του

να προσπαθήσουμε να ενώσουμε την προηγούμενη δουλειά πάνω στον μηχανισμό μετεγκατάστασης της υπηρεσίας και στην ενεργειακή μελέτη του δικτύου αισθητήρων, με μικτά αποτελέσματα.

Τελικά υλοποιήσαμε ένα μηχανισμό παραγωγής ”συμβάντων”, δηλαδή δυναμικών αλλαγών στην δραστηριότητα συγκεκριμένων περιοχών του δικτύου, τα οποία είναι ελεύθερα να κινηθούν και να αλλάξουν ισχύ κατά την διάρκεια της προσομοίωσης, και βρήκαμε δεδομένα για όλους τους συνδυασμούς των περιπτώσεων στατικών και δυναμικών γεγονότων, αλλά και στατικών τοποθετήσεων και ενεργοποιημένης της μετεγκατάστασης, αλλά και διαφορετικών χρονισμών του μηχανισμού της μετεγκατάστασης. Κλείσαμε με τα δύο πιο σημαντικά ανοιχτά ερωτήματα της εργασίας, αυτά του χρόνου ζωής του δικτύου και των δυναμικών συμβάντων γενικά, που χρήζουν περαιτέρω ανάπτυξης και ερευνητικής προσπάθειας ώστε να μπορέσουν να κατανοηθούν σε μεγαλύτερο βάθος.

Βιβλιογραφία

- [1] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422.
- [2] Arampatzis, T., Lygeros, J., and Manesis, S. (2005). A survey of applications of wireless sensors and wireless sensor networks. In *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, pages 719–724. IEEE.
- [3] Daskin, M. S. and Maass, K. L. (2015). The p-median problem. In *Location science*, pages 21–45. Springer.
- [4] Inc., M. (2006). *MicaZ Datasheet, Rev. A*. MEMSIC Inc.
- [5] Liu, C. L. (1987). *Elements of discrete mathematics*. Tata McGraw-Hill Education.
- [6] Manjeshwar, A. and Agrawal, D. P. (2001). Teen: a routing protocol for enhanced efficiency in wireless sensor networks. In *null*, page 30189a. IEEE.
- [7] Mohammadi Zanjireh, M. and Larijani, H. (2015). A survey on centralised and distributed clustering routing algorithms for wsns. volume 2015.
- [8] Tsoumanis, G., Oikonomou, K., Koufoudakis, G., and Aïssa, S. (2018). Energy-efficient sink placement in wireless sensor networks. *Computer Networks*.
- [9] Warneke, B. A., Scott, M. D., Leibowitz, B. S., Zhou, L., Bellew, C. L., Chediak, J. A., Kahn, J. M., Boser, B. E., and Pister, K. S. (2002). An autonomous 16 mm/sup 3/solar-powered node for distributed wireless sensor networks. In *Sensors, 2002. Proceedings of IEEE*, volume 2, pages 1510–1515. IEEE.

- [10] Wu, X., Lee, I., Dong, Q., Yang, K., Kim, D., Wang, J., Peng, Y., Zhang, Y., Saliganc, M., Yasuda, M., et al. (2018). A 0.04 mm 3 16nw wireless and batteryless sensor system with integrated cortex-m0+ processor and optical communication for cellular temperature measurement. In *2018 IEEE Symposium on VLSI Circuits*, pages 191–192. IEEE.
- [11] Yu, L., Wang, N., and Meng, X. (2005). Real-time forest fire detection with wireless sensor networks. In *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on*, volume 2, pages 1214–1217. IEEE.
- [12] Κολουντζάκης (2015). *Διακριτά μαθηματικά*. Σύνδεσμος Ελληνικών Ακαδημαϊκών Βιβλιοθηκών.

1 NODE.CC

```

#include <omnetpp.h>
#include <stdio.h>
#include <string.h>
#include <vector>

#include "OVERSEER.h"

#define TRAFFIC_MSG 40          // Traffic packets
#define TRAFFIC_GEN 41         // Timer self message for generating new traffic
#define RELOCATE 52            // Timer self message for service migration
#define CALCULATEROUTING 147

using namespace omnetpp;

class Node : public cSimpleModule
{
public:
    Node();
    virtual ~Node();
    double batteryLevel();
    void changeDemand(double);
    void activateService(Node*);
    void shutdownService();
    void findRouting(Node* serviceNode = nullptr);
    void scheduleMigration();
    bool serviceActive;
    cGate* routingCache;

protected:
    virtual void initialize(int) override;
    virtual int numInitStages() const override { return 4;};
    virtual void handleMessage(cMessage *) override;

private:
    OVERSEER *The_Overseer;    // Personal pointer to the overseer module
    cTopology *overseerTopo;    // Overseer's topology object reference for weighted routing

    double timeToCheckMigrate;
    int totalTrafficCount;
    std::vector<int> trafficStats;
    cMessage* SrvDemandMsg;
    cMessage* relocationMsg;

    double demandProbability;
    double battery;

    bool migrationLogs;
    bool batteryLogs;
}; Define_Module(Node);

```

```

void Node::initialize(int stage)
{
    if (stage == 0){
        migrationLogs = getAncestorPar("migrationLogs").boolValue();
        batteryLogs = getAncestorPar("batteryLogs").boolValue();
        timeToCheckMigrate = getAncestorPar("timeToCheckMigrate").doubleValue();
        battery = getAncestorPar("battery").doubleValue(); WATCH(battery);
        demandProbability = par("mydem").doubleValue();    WATCH(demandProbability);
        totalTrafficCount = 0;                               WATCH(totalTrafficCount);

        serviceActive = false; WATCH(serviceActive);
        WATCH_VECTOR(trafficStats);
        routingCache = nullptr;

        relocationMsg = new cMessage("relocation self event", RELOCATE);
        if(par("init").boolValue()){
            getDisplayString().setTagArg("i", 0, "misc/town2");
            serviceActive = true;
            if ( getAncestorPar("migrationActive").boolValue()){
                scheduleAt(simTime()+timeToCheckMigrate, relocationMsg);
            }
        }
        SrvDemandMsg = new cMessage("Traffic generation timer", TRAFFIC_GEN);
        scheduleAt(simTime()+1, SrvDemandMsg);
    }
    if (stage == 2){
        // Overseer has finished building its topology object, and found the 1-median
        The_Overseer = check_and_cast<OVERSEER *>(getParentModule()->getSubmodule("The_Overseer"));
        overseerTopo = &The_Overseer->topo;
    }
    if (stage == 3){
        // Find initial routing tables (only called once by init node)
        if(par("init").boolValue())
            scheduleAt(simTime()+0.01, new cMessage("routing initialize", CALCULATEROUTING));
    }
}

```

```

Node::Node()
{
    SrvDemandMsg = nullptr;
    relocationMsg = nullptr;
}

```

```

Node::~~Node()
{
    cancelAndDelete(SrvDemandMsg);
    cancelAndDelete(relocationMsg);
}

```

```

void Node::handleMessage(cMessage *msg) {
    if (msg->getKind() == TRAFFIC_GEN) {
        scheduleAt(simTime()+1, SrvDemandMsg);

        if (serviceActive)
            return;

        if ( !(uniform(0,1) < demandProbability))
            return;

        send(new cMessage("Traffic", TRAFFIC_MSG), routingCache);
        if ( battery-- < 0){ // decrement battery and check if lower than 0
            if(getAncestorPar("batteryLogs").boolValue()){
                // If migration is active, call the scalar function with pointer as the last position of the server
                if(getAncestorPar("migrationActive").boolValue())
                    The_Overseer->writeBatteryScalars(); // TODO this
                else
                    The_Overseer->writeBatteryScalars();
            }
            endSimulation();
        }
    }

    else if (msg->getKind() == TRAFFIC_MSG){
        if (serviceActive){
            // --- POLICY STATISTICS GATHERING ---
            totalTrafficCount++;
            bool incremented = false;
            for(int i = 0; i < trafficStats.size(); i+=2){
                // 0, 2, 4... gateIds / 1, 3, 5... message count
                if (msg->getArrivalGateId() == trafficStats[i]){
                    // we have recorded traffic from a gate already in the vector
                    trafficStats[i+1]++;
                    incremented = true;
                }
            }
            if (!incremented) {
                // we didn't find the gate in the vector so we have to document it
                trafficStats.push_back(msg->getArrivalGateId());
                trafficStats.push_back(1);
            }
            // --- END POLICY STATISTICS GATHERING ---
            delete msg;
        }
        else {
            bubble("routing message");
            send(msg, routingCache);
        }
        if ( battery-- < 0){ // decrement battery and check if lower than 0
            if(getAncestorPar("batteryLogs").boolValue()){
                if(getAncestorPar("migrationActive").boolValue())
                    The_Overseer->writeBatteryScalars(); // TODO this
                else

```

```

        The_Overseer->writeBatteryScalars();
    }

    endSimulation();
}

}

else if (msg->getKind() == RELOCATE){
    if (serviceActive){
        int maxIndex = -1;
        int maxTraffic = -1;
        int maxGateId = -1;
        for(int i = 0; i < trafficStats.size(); i+=2){
            EV << "GateID: " << trafficStats[i] << ", Traffic Counts: " << trafficStats[i+1] << endl;
            if( trafficStats[i+1] > maxTraffic ){
                maxTraffic = trafficStats[i+1];
                maxIndex = i;
                maxGateId = trafficStats[i];
            }
        }

        int outGateId = gateBaseId("port$o") + (maxGateId - gateBaseId("port$i"));
        if ( (2 * maxTraffic) < totalTrafficCount){
            scheduleAt(simTime()+timeToCheckMigrate, relocationMsg);
            totalTrafficCount = 0;
            trafficStats.clear();
        }
        else{
            Node* neighbour = check_and_cast<Node *>(gate(outGateId)->getNextGate()->getOwnerModule());
            shutdownService();
            activateService(neighbour);
        }
    }
    else
        throw cRuntimeError("Non-service node tried to relocate");
}

else if (msg->getKind() == CALCULATEROUTING){
    Node* tempNode = check_and_cast<Node *>(getParentModule()->getSubmodule("node", getAncestorPar("starter"
    findRouting(tempNode);

}

}

void Node::activateService(Node* node){
    node->serviceActive = true;
    node->getDisplayString().setTagArg("i", 0, "misc/town2");
    if ( node->getAncestorPar("migrationActive").boolValue())
        node->scheduleMigration();
    findRouting(node);
}

void Node::shutdownService(){
    serviceActive = false;

```

```

totalTrafficCount = 0;
trafficStats.clear();
getDisplayString().setTagArg("i", 0, "misc/node_vs");
The_Overseer->hop();
}

void Node::findRouting(Node* serviceNode){
    // service is an optional argument
    int numNodes = getParentModule()->getSubmodule("node", 0)->getVectorSize();
    Node* service = serviceNode;
    if (service == nullptr){
        for(int i = 0; i < numNodes; i++){
            service = check_and_cast<Node *>(getParentModule()->getSubmodule("node", i));
            if (service->serviceActive)
                break;
        }
    }
    overseerTopo->calculateWeightedSingleShortestPathsTo(overseerTopo->getNodeFor(service)); // calculate only o
    Node* node = nullptr;
    cTopology::Node *topoNode = nullptr;
    for(int i = 0; i < numNodes; i++){
        node = check_and_cast<Node *>(getParentModule()->getSubmodule("node", i));
        if ( node == service ){
            node->routingCache = nullptr;
            continue; // getPath errors if you try to get the path to yourself (service)
        }
        topoNode = overseerTopo->getNodeFor(node);
        node->routingCache = topoNode->getPath(0)->getLocalGate();
    }
}

}

double Node::batteryLevel(){
    return battery;
}

void Node::changeDemand(double newDemand){
    if( newDemand < 0 || newDemand > 1)
        throw cRuntimeError("(changeDemand) Invalid demand value range.");
    demandProbability = newDemand;
}

void Node::scheduleMigration(){
    Enter_Method("scheduleMigration()");
    scheduleAt(simTime()+timeToCheckMigrate, relocationMsg);
}

```

2 NODE.h

```

/*
 * NODE.h
 *
 * Created on: Oct 2, 2018

```

```

*      Author: Constantine
*/

#ifndef NODE_H_
#define NODE_H_

#include <omnetpp.h>

using namespace omnetpp;

class Node : public cSimpleModule
{
public:
    Node();
    virtual ~Node();
    double batteryLevel();
    void changeDemand(double);
};
#endif /* NODE_H_ */

```

3 OVERSEER.cc

```

#include <omnetpp.h>
#include <cmath>
#include <vector>

using namespace omnetpp;

#include "NODE.h"

class OVERSEER : public cSimpleModule
{
public:
    OVERSEER();
    virtual ~OVERSEER();

    // 1-median functions
    cModule* findMedian();
    cModule* findMedian_old();

    // Scalar & Vector functions
    void writeScalars(double);
    void writeVectorCost(double);
    void writeBatteryScalars(cModule*);
    void writeBatteryScalars();
    void hop();

    // Event functions
    void calculateSteps(double, double);
    void eventCalcAffected();
    void eventRestoreAffected();
    void eventStep();

```



```

    currentX = 1;
    currentY = 1;
    scheduleAt(simTime()+300, new cMessage("Event", 707));
    ///////////////////////////////////

    //calculateSteps(pathPoints[movingToPoint*2], pathPoints[movingToPoint*2 +1]);
    //eventMoveMsg = new cMessage("Event", 707);
    //scheduleAt(simTime()+eventMoveFreq+0.2, eventMoveMsg); // TODO UNCOMMENT FOR EVENT
}
if (stage == 1){
    findMedian();
}
}

void OVERSEER::handleMessage(cMessage *msg)
{
    if ( msg->isSelfMessage() && msg->getKind() == 808){
        findMedian();
        //scheduleAt(simTime()+10, msg);
        //dynamicNodeCreate(1);
    }
    if (msg->getKind() == 707){
        eventCalcAffected(); // manual mode

        // MANUAL MODE ACTIVATED
        /*
        if(numSteps == 0){
            movingToPoint += 1; // since we finished the move to this point, increment the point we're moving to
            calculateSteps(pathPoints[movingToPoint*2], pathPoints[movingToPoint*2 +1]); // calculate the steps
        }
        eventStep();
        numSteps--;
        scheduleAt(simTime()+eventMoveFreq, msg);
        */
    }
}

void OVERSEER::eventStep(){
    currentX = currentX + stepX;
    currentY = currentY + stepY;
    eventRestoreAffected();
    eventCalcAffected();
}

void OVERSEER::calculateSteps(double targX, double targY){
    int numStepsY = int(round(abs((targY-currentY)/eventSpeed)));
    int numStepsX = int(round(abs((targX-currentX)/eventSpeed)));
    if (numStepsX == numStepsY) numSteps = numStepsX;
    else if (numStepsX < numStepsY) numSteps = numStepsY;
    else numSteps = numStepsX;

    stepX = (targX - currentX)/numSteps;
    stepY = (targY - currentY)/numSteps;
}

```

```

EV << "Number of Steps: " << numSteps << "\nStep X: " << stepX << "\nStep Y: " << stepY << endl;
}

void OVERSEER::eventCalcAffected(){
    /*
     * Calculates the affected nodes in the area of the event and sets their demand to 1
     * (event should already be initialized)
     */
    double X = currentX;
    double Y = currentY;
    int N = getAncestorPar("numNodes").intValue();
    Node* node = nullptr;
    for(int i = 0; i < N; i++){
        node = check_and_cast<Node*>(getParentModule()->getSubmodule("node", i));
        if ( sqrt(pow((X - node->par("xcoord").doubleValue()), 2) + pow((Y - node->par("ycoord").doubleValue()), 2) ) < 2 )
            affectedNodes.push_back(i);
        node->changeDemand(1);
        node->getDisplayString().setTagArg("i", 1, "gold");
    }
}

void OVERSEER::eventRestoreAffected(){
    /*
     * Return the affected nodes's demand to their original value (that they get from the .ini on network initia
     */
    int vSize = affectedNodes.size();
    Node* node = nullptr;
    for (int i = 0; i < vSize; i++){
        node = check_and_cast<Node*>(getParentModule()->getSubmodule("node", affectedNodes[i]));
        node->changeDemand(node->par("mydem").doubleValue());
        node->getDisplayString().setTagArg("i", 1, "white");
    }
    affectedNodes.clear();
}

void OVERSEER::writeBatteryScalars(cModule* caller){ // for migration
    /*
     * Calculates Mean Battery level, and Battery Variance and stores them in Scalar output files. (called by th
     */
    recordScalar("Time of Network Death", simTime());
    int numNodes = getParentModule()->getSubmodule("node", 0)->getVectorSize();
    double batterySum = 0;
    Node* tempnode = nullptr;
    for(int n = 0; n < numNodes; n++){
        tempnode = check_and_cast<Node*>(getParentModule()->getSubmodule("node", n));
        batterySum += tempnode->batteryLevel();
    }
    double Mean = batterySum / numNodes;
    recordScalar("Network battery average", Mean);
}

```

```

double meanDiffSum = 0;
tempnode = nullptr;
for(int n = 0; n < numNodes; n++){
    tempnode = check_and_cast<Node *>(getParentModule()->getSubmodule("node", n));
    meanDiffSum = pow((tempnode->batteryLevel() - Mean), 2);
}

recordScalar("Battery level variance", meanDiffSum / (numNodes - 1)); // TODO Bessel's Correction n-1 or not?

topo.calculateWeightedSingleShortestPathsTo(topo.getNodeFor(median));
recordScalar("Weighted (Cost) Distance to Median", topo.getNodeFor(caller)->getDistanceToTarget());

topo.calculateUnweightedSingleShortestPathsTo(topo.getNodeFor(median));
recordScalar("Hop Distance to Median", topo.getNodeFor(caller)->getDistanceToTarget());
}

void OVERSEER::writeBatteryScalars(){
    /*
     * Calculates Mean Battery level, and Battery Variance and stores them in Scalar output files. (called by th
     */
    recordScalar("Time of Network Death", simTime());
    int numNodes = getParentModule()->getSubmodule("node", 0)->getVectorSize();
    double batterySum = 0;
    Node* tempnode = nullptr;
    for(int n = 0; n < numNodes; n++){
        tempnode = check_and_cast<Node *>(getParentModule()->getSubmodule("node", n));
        batterySum += tempnode->batteryLevel();
    }
    double Mean = batterySum / numNodes;
    recordScalar("Network battery average", Mean);

    double meanDiffSum = 0;
    for(int n = 0; n < numNodes; n++){
        tempnode = check_and_cast<Node *>(getParentModule()->getSubmodule("node", n));
        meanDiffSum = pow((tempnode->batteryLevel() - Mean), 2);
    }

    recordScalar("Battery level variance", meanDiffSum / (numNodes - 1)); // TODO Bessel's Correction n-1 or not?

    cModule* originalPos = getParentModule()->getSubmodule("node", this->par("starter").intValue()); // the node

    topo.calculateWeightedSingleShortestPathsTo(topo.getNodeFor(median));
    recordScalar("Weighted (Cost) Distance to Median", topo.getNodeFor(originalPos)->getDistanceToTarget());

    topo.calculateUnweightedSingleShortestPathsTo(topo.getNodeFor(median));
    recordScalar("Hop Distance to Median", topo.getNodeFor(originalPos)->getDistanceToTarget());
}

void OVERSEER::writeScalars(double final_cost){
    recordScalar("Hops", hops);
}

```

```

// Absolute cost reduction
double initCost = getParentModule()->getSubmodule("node", this->par("starter").intValue()->par("cost").doubleValue();
recordScalar("Absolute Cost Reduction", (initCost - final_cost));

// Percentage cost reduction
recordScalar("Percent Cost Reduction", ((initCost - final_cost)/initCost)*100);

recordScalar("Time to descend", simTime());
}

void OVERSEER::writeVectorCost(double cost){
    costVector.recordWithTimestamp(simTime(), cost);
}

void OVERSEER::hop(){
    hops++;
}

cModule* OVERSEER::findMedian(){ // NEW PERFORMANCE ORIENTED 1-MEDIAN CALCULATOR.
    /*
     * Calculates the 1-median in a general network topology with vector gates
     * Returns the module which is the median as a cModule pointer
     */

    topo.extractByProperty("topoIncl", "yes");
    int numNodes = topo.getNumNodes();
    double sumPathCost = 0;
    double minPathCost = INFINITY;
    int minN = -1;

    // TOPOLOGY BUILD
    cTopology::Node *current_node = nullptr;
    for( int n = 0; n < numNodes; n++){
        current_node = topo.getNodeFor(getParentModule()->getSubmodule("node", n));
        //current_node->setWeight(1); // Initialize Node weights (node weights = 1, calculate weighted paths only)

        // Initialize link weights
        // Note: You can access channels from "out"--> gates only, so we only work with those here, even though
        double tmpw = 0;
        for (int i = 0; i < current_node->getNumInLinks(); i++) {
            if (current_node->getLinkIn(i)->isEnabled()){
                tmpw = current_node->getLinkIn(i)->getRemoteGate()->getChannel()->par("weight").doubleValue(); //
                current_node->getLinkIn(i)->setWeight(tmpw);
            }
        }
        for (int i = 0; i < current_node->getNumOutLinks(); i++) {
            if (current_node->getLinkOut(i)->isEnabled()){
                tmpw = current_node->getLinkOut(i)->getLocalGate()->getChannel()->par("weight").doubleValue(); //
                current_node->getLinkOut(i)->setWeight(tmpw);
            }
        }
    }
}

```

```

// 1-MEDIAN CALCULATION
cModule* Temp;
cModule* Temp2;
cTopology::Node *targetnode = nullptr;
cTopology::Node *node = nullptr;
double demand = 0;
for( int n = 0; n < numNodes; n++){
    Temp = getParentModule()->getSubmodule("node", n);
    targetnode = topo.getNodeFor(Temp);
    topo.calculateWeightedSingleShortestPathsTo(targetnode);
    for (int i = 0; i < numNodes; i++) {
        Temp2 = getParentModule()->getSubmodule("node", i);
        demand = Temp2->par("mydem").doubleValue();
        node = topo.getNodeFor(Temp2);
        if(node->getNumPaths() == 0){
            if(i == n) continue;
            throw cRuntimeError("Critical Error! Disconnected Network.");
        }
        sumPathCost += (node->getDistanceToTarget() * demand);
        //sumPathCost += node->getDistanceToTarget();
    }

    if (sumPathCost < minPathCost){ // minPathCost is the minimum cost of the minimum spanning tree create
        minPathCost = sumPathCost; // sumPathCost is the sum of the costs of each weighted shortest pat
        minN = n;
    }
    Temp->par("cost").setDoubleValue(sumPathCost);
    Temp->getDisplayString().setTagArg("t", 0, sumPathCost);
    sumPathCost = 0;
}

if( minPathCost > 0 ){
    recordScalar("Median", minN);
    median = getParentModule()->getSubmodule("node", minN);
    median->getDisplayString().setTagArg("i", 0, "misc/house");
    return (median);
}
throw cRuntimeError("Critical Error! Couldn't calculate median.");
}

cModule* OVERSEER::findMedian_old(){ // OLD GENERAL PURPOSE 1MEDIAN FINDER
/*
 * Calculates the 1-median in a general network topology with vector gates
 * Returns the module which is the median as a cModule pointer
 */
if ( median != nullptr ){
    prevMedian = median;
    prevMedian->getDisplayString().setTagArg("i", 0, "misc/node_vs");
}

topo.extractByProperty("topoIncl", "yes"); // EV << topo.str() << endl;
int numNodes = topo.getNumNodes();
double sumPathCost = 0;

```

```

double minPathCost = INFINITY;
int minN = -1;

cTopology::Node *current_node = nullptr;
for( int n = 0; n < numNodes; n++){
    if ( getParentModule()->getSubmodule("node", n) == nullptr ){
        // we are most likely dealing with a runtime dynamically created node that has no index z
        std::string strTmp = "node["+ std::to_string(n) +"]";
        const char* c = strTmp.c_str();
        if ( getParentModule()->getSubmodule(c) == nullptr ){
            cRuntimeError("(FindMedian) Cannot find Module");
        }
        else{
            current_node = topo.getNodeFor(getParentModule()->getSubmodule(c));
        }
    }else{
        current_node = topo.getNodeFor(getParentModule()->getSubmodule("node", n));
    }
    // current_node->setWeight(1); //(current_node->getModule()->par("node_demand").doubleValue()); // Initia

    // Initialize link weights
    // Note: You can access channels from "out"--> gates only, so we only work with those here, even tho it'
    double tmpw = 0;
    for (int i = 0; i < current_node->getNumInLinks(); i++) {
        if (current_node->getLinkIn(i)->isEnabled()){
            tmpw = current_node->getLinkIn(i)->getRemoteGate()->getChannel()->par("weight").doubleValue(); /
            current_node->getLinkIn(i)->setWeight(tmpw);
            //EV << "IN LINK " << i <<" - weight: " << current_node->getLinkIn(i)->getWeight() << " // comin
        }
    }
    for (int i = 0; i < current_node->getNumOutLinks(); i++) {
        if (current_node->getLinkOut(i)->isEnabled()){
            tmpw = current_node->getLinkOut(i)->getLocalGate()->getChannel()->par("weight").doubleValue(); /
            current_node->getLinkOut(i)->setWeight(tmpw);
            //EV << "OUT LINK " << i <<" - weight: " << current_node->getLinkOut(i)->getWeight() << " // goi
        }
    }
}
for( int n = 0; n < numNodes; n++){
    // Calculating Shortest Weighted Paths for each node n to every other node i in the network
    cModule *Temp = nullptr;
    for (int i = 0; i < numNodes; i++) {
        cModule *target_module = getParentModule()->getSubmodule("node", i);
        if ( target_module == nullptr ){
            std::string strTmp = "node["+ std::to_string(i) +"]";
            const char* c = strTmp.c_str();
            target_module = getParentModule()->getSubmodule(c);
            if ( target_module == nullptr )
                cRuntimeError("(FindMedian) Cannot find Module");
        }
        cTopology::Node *targetnode = topo.getNodeFor(target_module);

        topo.calculateWeightedSingleShortestPathsTo(targetnode);
    }
}

```

```

/* This performs the Dijkstra algorithm and stores the result in the cTopology object.
* The result can then be extracted using cTopology and cTopology::Node methods. */

/*
* The purpose of the getDistanceToTarget() member function of a node is self-explanatory.
* In the unweighted case, it returns the number of hops.
* The getNumPaths() member function returns the number of edges which are part of a shortest path,
* and path(i) returns the ith edge of them as cTopology::LinkOut.
* If the shortest paths were created by the ...SingleShortestPaths() function,
* getNumPaths() will always return 1 (or 0 if the target is not reachable),
* that is, only one of the several possible shortest paths are found.
* The ...MultiShortestPathsTo() functions find all paths, at increased run-time cost.
* The cTopology's getTargetNode() function returns the target node of the last shortest path search.
*/
cTopology::Node *node = nullptr;
if ( getParentModule()->getSubmodule("node", n) == nullptr ){
    std::string strTmp = "node["+ std::to_string(n) +"]";
    const char* c = strTmp.c_str();
    if (getParentModule()->getSubmodule(c) == nullptr ){
        cRuntimeError("(FindMedian) Cannot find Module");
    }
    else{
        node = topo.getNodeFor(getParentModule()->getSubmodule(c));
    }
}
else{
    node = topo.getNodeFor(getParentModule()->getSubmodule("node", n));
}

if (node == nullptr){
    EV << "This module: (" << getFullPath() << ") is not included in the topology.\n";
}
else if (node->getNumPaths() == 0){
    if( i == n ){ // don't print "unreachable" because we are the same node..
        continue;
    }
    EV << "No path to destination. (Unreachable)\n";
    throw cRuntimeError("Critical Error! UNCONNECTED NETWORK!!!");
    endSimulation();
}
else{
    Temp = node->getModule(); // this is the current "node" (the root node of the min spanning tree
    double demand = topo.getTargetNode()->getModule()->par("mydem").doubleValue(); // the demand val

    sumPathCost += (node->getDistanceToTarget() * demand) ; // Distance in unweighted means "hops",
    // Debug outputs, not needed for function's goal
    EV << "TRAVERSING PATH=: " << node->getModule()->getFullName() << " to " << topo.getTargetNode()
    EV << node->getModule()->getFullName();
    while (node != topo.getTargetNode()){
        /* OLD EVs
        EV << "We are in " << node->getModule()->getFullPath() << endl;
        EV << node->getDistanceToTarget() << " hops to go\n"; // for unweighted case should be integ
        EV << "There are " << node->getNumPaths() << " equally good directions, taking the first one

```

```

        cTopology::LinkOut *path = node->getPath(0);
        EV << "Taking gate " << path->getLocalGate()->getFullName() << " we arrive in " << path->get
        /*
        cTopology::LinkOut *path = node->getPath(0);
        EV << " -> " << path->getRemoteNode()->getModule()->getFullName();
        node = path->getRemoteNode();
    }
    EV << endl;
}

if ( sumPathCost < minPathCost ){ // minPathCost is the minimum cost of the minimum spanning tree create
// sumPathCost is the sum of the costs of each weighted shortest path
    minPathCost = sumPathCost;
    minN = n;
    EV << "minN: " << minN << ".. minPathCost: " << minPathCost << endl;
}
EV<< "COST OF NODE[" << n << "] = " << sumPathCost << endl;
Temp->par("cost").setDoubleValue(sumPathCost); // setting the node's cost parameter value
Temp->getDisplayString().setTagArg("t", 0, sumPathCost);
sumPathCost = 0;
}
if( minPathCost > 0.1 ){
    recordScalar("Median", minN);
    if ( getParentModule()->getSubmodule("node", minN) == nullptr ){
        std::string strTmp = "node["+ std::to_string(minN) +"]";
        const char* c = strTmp.c_str();
        if ( getParentModule()->getSubmodule(c) == nullptr ){
            cRuntimeError("(FindMedian) Cannot find Module");
        }
        else{
            median = getParentModule()->getSubmodule(c);
            median->getDisplayString().setTagArg("i", 0, "misc/house");
            return (median);
        }
    }
    else{
        median = getParentModule()->getSubmodule("node", minN);
        median->getDisplayString().setTagArg("i", 0, "misc/house");
        return (median);
    }
}
return (nullptr);
}

OVERSEER::OVERSEER()
{
    eventMoveMsg = nullptr;
}
OVERSEER::~OVERSEER()
{
    cancelAndDelete(eventMoveMsg);
}

```

4 OVERSEER.h

```
/*
 * OVERSEER.h
 *
 * Created on: Aug 30, 2018
 * Author: Constantine
 */

#ifndef OVERSEER_H_
#define OVERSEER_H_

#include <omnetpp.h>

using namespace omnetpp;
class OVERSEER : public cSimpleModule
{
public:
    OVERSEER();
    virtual ~OVERSEER();

    void dynamicNodeCreate(int);
    int getCreatedNodes();
    void setCreatedNodes(int);
    void addCreatedNodes(int);
    void subtractCreatedNodes(int);
    void incrementCreatedNodes();

    cModule* findMedian();
    cModule* returnMedian();
    void writeScalars(double);
    void writeVectorCost(double);
    void hop();
    void writeBatteryScalars(cModule*);
    void writeBatteryScalars();

    cTopology topo;

protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

#endif /* OVERSEER_H_ */
```

5 Node.ned

```
simple Node{
    parameters:
        @topoIncl("yes");
        double xcoor = default(0);
        double ycoor = default(0);
        bool init = default(false);
        double cost = default(0);
```

```

double mydem = default(0);
gates:
    inout port[];
}

```

6 OVERSEER.ned

```

simple OVERSEER
{
    parameters:
        int starter = default(-1);
        @display("p=0,0;i=abstract/penguin");
}

```

7 RandomGraph.ned

```

import Node;
import OVERSEER;
import ned.DelayChannel;

```

```

network RandomGraph
{
    parameters:
        int starter;
        int numNodes;
        double rc;
        double timeToCheckMigrate;
        bool migrationActive;
        bool eventActive;
        bool movingEvent;

        bool migrationLogs;
        bool batteryLogs;

        double battery;
    types:
        channel Channel extends ned.DelayChannel{
            delay = 1000ms;
            double weight = default(0);
            bool weightSet = default(true); // if set default to weight set = true, skip custom random weight
            @display("tt=$weight;");
        }
    submodules:
        node[numNodes]: Node{
            @display("p=$xcoor,$ycoor;i=misc/node_vs;");
        }
        The_Overseer: OVERSEER;
    connections allowunconnected:
        for i = 0 .. numNodes - 1, for j = i + 1 .. numNodes - 1{
            node[i].port++ <--> Channel {weight = sqrt((node[i].xcoor - node[j].xcoor)^2 + (node[i].ycoor - node[j].ycoor)^2) / rc;
            if ( sqrt((node[i].xcoor - node[j].xcoor)^2 + (node[i].ycoor - node[j].ycoor)^2) <= rc );
        }
}

```

```
}
```

```
}
```

8 omnetpp.ini

```
[Config RandomGraph]
network = RandomGraph
seed-set = ${seed=6}
**.timeToCheckMigrate = 5000 # how many seconds to gather traffic data for migration
**.xcoor = uniform(0, 1)
**.ycoor = uniform(0, 1)
**.numNodes = 500
**.rc = 0.08
**.mydem = uniform(0, 0.05) # demand of each node
**.starter = ${initN = 0..250} # used in Overseer module for cost calculations (eg. final cost)
**.init = (index == ${initN}) ? true : false # where is service initially
**.migrationLogs = false # record vectors/scalars about migration (descending cost p
**.batteryLogs = true # record scalars about mean battery left, network
**.migrationActive = ${migration = true, false}
**.eventActive = false
**.movingEvent = false
**.battery = 1000000

# MICA2 (MPR400CB) mote
###.battery = 5760000 # 800 mAh * 3,600 = 2,880,000 mAs per AA battery, X2 = 5,760,000
###.msgSize = 30720 # binary picture resolution 100x100p = 30KB
###.datarate = 19660 # bits per second. (38.4 kbaud * 1024) / 2 = 19660 (manchester encoding)
###.transTime = (IN NED) # the delay of the channel ( in milliseconds )
###.transmitCost = 42.2 # (msgSize / datarate) * cost of 1 sec of transmission => 6400 / 19,660 * 27 mA = 8.8
###.receiveCost = 15 # same as above, but * 10 mA
###.processingCost = 0 # 48 mA for 1s camera usage of waspMote camera module
###.activeCost = 8 # per time unit ( second )
```

9 BinaryTree.ned

```
import Node;
import OVERSEER;

import ned.DelayChannel;

// binary tree topology.

network BinaryTree
{
    parameters:
        int levels = default(6); // 1 = root only

    types:
        channel Channel extends ned.DelayChannel{
            delay = 100ms;
            double weight = default(0);
            bool weightSet = default(false);
        }
}
```

```

submodules:
    The_Overseer: OVERSEER;
    node[2^levels-1]: Node { gates: port[3]; };

connections allowunconnected:
    for i=1..sizeof(node)-1 {
        node[i].port[0] <--> Channel <--> node[int(floor((i-1)/2))].port[1+i%2];
    }
}

```

10 Grid.ned

```

import Node;
import OVERSEER;

import ned.DelayChannel;

network Grid
{
    parameters:
        int columns;
        int rows;
        double rc;
        bool performanceMode = default(false);
        bool knownMedianMode = default(false);
        int medianNodeNum = default(-1);
        double timeToCheckMigrate = default(100000);
        double battery;
        double msgSize;
        double datarate;
        double transmitCost;
        double receiveCost;
        double processingCost;
        double activeCost;

    types:
        channel Channel extends ned.DelayChannel{
            delay = 100ms;
            double weight = default(0);
            bool weightSet = default(false);
        }

    submodules:
        The_Overseer: OVERSEER;
        node[columns*rows]: Node {
            @display("p=,,matrix,$columns");
            gates: port[4];
        }

    connections allowunconnected:
        for i=0..rows-1, for j=0..columns-1 {

```

```

        node[i*columns+j].port[1] <--> Channel <--> node[(i+1)*columns+j].port[0] if i!=rows-1;
        node[i*columns+j].port[3] <--> Channel <--> node[i*columns+j+1].port[2] if j!=columns-1;
    }
}

```

11 (old) NODE.CC

```

#include <omnetpp.h>
#include <stdio.h>
#include <string.h>
#include <vector>

#include "OVERSEER.h"
#include "targ_msg_m.h"

#define TRAFFIC_MSG 40          // "Artificial" traffic message always bound towards a service point
#define TRAFFIC_GEN 41         // Self message for generating new traffic
#define SERVICE_MIGRATION 50   // Message containing a service currently in migration
#define RELOCATE 52            // Self message for service relocation
#define POLICY_STAT 51         // Self message for statistics tracking timing for the migration policy

using namespace omnetpp;

class Node : public cSimpleModule
{
public:
    Node();
    virtual ~Node();
    double batteryLevel();
    void changeDemand(double);

protected:
    virtual void initialize(int) override;
    virtual void handleMessage(cMessage *) override;
    virtual int numInitStages() const override { return 4;};
    Targ_msg* generateTrafficMessage(int mode, int source = -1); // mode 0: random, mode 1: to service node

private:
    OVERSEER *The_Overseer;          // Personal pointer to the overseer module

    cTopology topo;                  // Local topology object for routing traffic
    cTopology *overseerTopo = nullptr; // Overseer's topology object reference for weighted routing

    bool serviceActive = false;
    double timeToCheckMigrate;
    int totalTrafficCount;
    std::vector<int> trafficStats; // have to use the std:: namespace to use the vector class
    cMessage* SrvDemandMsg;

    cGate* routingCache;
    int servicePosforMsgGenerator;

```

```

double demandProbability; // probability
double costToTransmit;    // cost (for battery) to transmit message
double batteryCap;        // battery total capacity
long numRouted;           // number of routed packages in the module's life
double battery;           // battery current level
double rc;
double msgSize;
double datarate;
double transmitCost;
double receiveCost;
double processingCost;
double activeCost;

cModule* median1;          // 1-median module pointer (taken from overseer's similar pointer)
}; Define_Module(Node);

void Node::initialize(int stage)
{
    if (stage == 0){
        timeToCheckMigrate = getAncestorPar("timeToCheckMigrate").doubleValue();
        battery = getAncestorPar("battery").doubleValue(); WATCH(battery);
        batteryCap = battery;
        msgSize = getAncestorPar("msgSize").doubleValue();
        datarate = getAncestorPar("datarate").doubleValue();
        transmitCost = getAncestorPar("transmitCost").doubleValue();
        receiveCost = getAncestorPar("receiveCost").doubleValue();
        processingCost = getAncestorPar("processingCost").doubleValue();
        activeCost = getAncestorPar("activeCost").doubleValue();

        rc = getAncestorPar("rc").doubleValue();
        numRouted = 0; WATCH(numRouted);
        totalTrafficCount = 0;
        WATCH(totalTrafficCount); WATCH(serviceActive); WATCH_VECTOR(trafficStats); WATCH(demandProbability);
        routingCache = nullptr;
        SrvDemandMsg = new cMessage("demand timer", TRAFFIC_GEN);
        demandProbability = par("mydem").doubleValue();

        servicePosforMsgGenerator = -1;

        if(par("init").boolValue()){
            getDisplayString().setTagArg("i", 0, "misc/town2");
            serviceActive = true;
            scheduleAt(simTime()+timeToCheckMigrate, new cMessage("relocation self event", RELOCATE)); // TODO S
        }
        else{
            scheduleAt(simTime()+1, SrvDemandMsg); // TODO UNCOMMENT FOR TRAFFIC
        }
    }
    if (stage == 1){
        // Setting random bi-directional channel weights irrespective of coordinate distance
        cGate *gate = nullptr;
        for (cModule::GateIterator i(this); !i.end(); i++) {
            gate = *i;

```

```

if (gate->getChannel() == nullptr || gate->getType() == cGate::INPUT){ // we will only deal with the
    continue;
}
else{
    if(!gate->getChannel()->par("weightSet").boolValue()){
        // for vector gates only ([] in NED)
        double rdn = uniform(3, 100);
        std::string str;
        str = "ls=white,2;t="+ std::to_string(int(rdn));
        const char* c = str.c_str();

        // ** Output gate towards neighbor node **
        cChannel *tempchannel = gate->getChannel();
        tempchannel->setDisplayString(c);
        tempchannel->par("weight").setDoubleValue(rdn);
        tempchannel->par("weightSet").setBoolValue(true);

        // ** Output gate from neighbor node towards this one **
        // Method: Go to the next gate: means go to the input gate of the module that this output gate
        // then search for the portOutput gate with the same index as the portInput gate that we got
        // after we got the respective output gate of the input gate, we get its channel which this
        // and do the same tasks to it as we did in the easy output channel.
        tempchannel = gate->getNextGate()->getOwnerModule()->gate("port$o", gate->getNextGate()->get
        tempchannel->par("weight").setDoubleValue(rdn);
        tempchannel->par("weightSet").setBoolValue(true);
        tempchannel->setDisplayString("ls=white, 2");
    }
}
}
}

if (stage == 3){
    // Overseer has finished building its topology object, and found the 1-median
    The_Overseer = check_and_cast<OVERSEER *>(getParentModule()->getSubmodule("The_Overseer"));
    overseerTopo = &The_Overseer->topo;
}
}

```

```

Node::Node()
{
    SrvDemandMsg = nullptr;
}

```

```

Node::~~Node()
{
    cancelAndDelete(SrvDemandMsg);
}

```

```

void Node::handleMessage(cMessage *msg) {
    if ( msg->getKind() == TRAFFIC_GEN ) {
        EV << "TRAFFIC_GEN SELF MESSAGE RECEIVED \n";
    }
}

```

```

battery -= activeCost;

if ( serviceActive ){
    EV << "Active Service so no real traffic generation, returning. \n";
    if ( uniform(0,1) < demandProbability){
        battery -= processingCost + transmitCost;
    }
    scheduleAt(simTime()+1, SrvDemandMsg);
    return;
}

if ( !(uniform(0,1) < demandProbability) ){
    scheduleAt(simTime()+1, SrvDemandMsg); // try again at the next time, only the "active mode" battery
    return;
}
battery -= processingCost; // message created, lost processing cost power

Targ_msg *trafficMsg = generateTrafficMessage(1); // NEW TRAFFIC MESSAGE
if ( trafficMsg->getDestination() == 99999){
    // service node is in the process of migrating
    EV << "[Can't find service node]  DELETING TRAFFIC PACKET... \n";
    delete trafficMsg;
    routingCache = nullptr; // reset cache
    scheduleAt(simTime()+1, SrvDemandMsg); // try again later.... i guess...
    return;
}

////////////////////////////////////

// ROUTING using WEIGHTED topology and topology algorithms using the Observer topo object.
if (routingCache == nullptr){
    // there exists no cached routing gate so we have to find it anew (or for the first time)
    cModule *targetModule = getParentModule()->getSubmodule("node", int(trafficMsg->getDestination()));
    if (targetModule == nullptr) {
        // get submodule returns nullptr when we go above the vector size of the original network
        std::string str = "node["+ std::to_string(trafficMsg->getDestination()) +"]";
        const char* c = str.c_str();
        targetModule = getParentModule()->getSubmodule(c);
    }

    cTopology::Node *targetNode = overseerTopo->getNodeFor(targetModule);
    overseerTopo->calculateWeightedSingleShortestPathsTo(targetNode);
    cTopology::Node *node = overseerTopo->getNodeFor(this);

    if (node == nullptr) {
        EV << "ROUTING ERROR: This node (" << getFullPath() << ") is not included in this topology.\n";
        throw cRuntimeError("ROUTING ERROR: This node is not included in this topology.");
    } else if (node->getNumPaths() == 0) {
        EV << "ROUTING ERROR: There exists no path to the destination from this node.\n";
        throw cRuntimeError("ROUTING ERROR: There exists no path to the destination from this node.");
    } else {
        routingCache = node->getPath(0)->getLocalGate(); // SETTING CACHED GATE
        send(trafficMsg, routingCache);
    }
}

```

```

}
else {
    // cached gate routing exists, send traffic msg to that gate
    EV << " GENERATION - ROUTING WITH CACHE " << endl;
    send(trafficMsg, routingCache);
}

////////////////////////////////////

battery -= transmitCost;
numRouted++;
if ( battery < 1 ){
    battery = 0;
    The_Overseer->writeBatteryScalars(this);
    endSimulation();
}
scheduleAt(simTime()+1, SrvDemandMsg); // generate continuous probabilistic traffic by scheduling a TRAF
}

else if ( msg->getKind() == TRAFFIC_MSG ){
    battery -= receiveCost;

    Targ_msg *trafficMsg = check_and_cast<Targ_msg *>(msg);
    if ( serviceActive ){
        bubble("Reached!");
        // --- POLICY MIGRATION STATISTICS ---
        totalTrafficCount++;
        bool incremented = false;
        int vSize = trafficStats.size();
        for(int i = 0; i < vSize; i+=2){
            // 0, 2, 4... gateIds / 1, 3, 5... message count
            if (int(trafficMsg->getArrivalGateId()) == trafficStats[i]){
                // we got traffic from a gate already in the vector array
                trafficStats[i+1]++;
                incremented = true;
            }
        }
        if (!incremented) {
            // we didn't find the gate in the vector so we have to document it
            trafficStats.push_back( int(trafficMsg->getArrivalGateId()) );
            trafficStats.push_back(1); // this is the first time we see this gate
        }
        // --- END POLICY MIGRATION STATISTICS ---
        /*
        if (trafficMsg->getDestination() == getIndex()){
            EV << "Case 1, the service node is currently active, and the message was routed successfully to
        }
        else {
            std::string strTmp = "node["+ std::to_string(trafficMsg->getDestination()) +"]";
            const char* c = strTmp.c_str();
            if (this == getParentModule()->getSubmodule(c))
                EV << "Case 1, the service node is currently active, and the message was routed successfully
            else

```

```

        EV << "Case 2, the service is active, but the message hasn't had an update in its routing yet";
    }
    */
    numRouted++; // service(sink) node stats (simulating sending a message to the internet for example)
    battery -= transmitCost;
    if(battery < 0){
        battery = 0;
        The_Overseer->writeBatteryScalars(this);
        endSimulation();
    }
    delete trafficMsg;
}
else {
    std::string strTmp = "node["+ std::to_string(trafficMsg->getDestination()) +"]";
    const char* c = strTmp.c_str();
    if (this == getParentModule()->getSubmodule(c)){
        EV << "we used to be an active service point, but this message didn't get rerouted in time" << endl;
        delete trafficMsg;
        return;
    }

    // Case 4, we are neither an active service, nor the target of the traffic, re-routing towards the c
    cModule *targetModule = getParentModule()->getSubmodule("node", int(trafficMsg->getDestination()));
    if ( targetModule == nullptr ){
        std::string strTmp = "node["+ std::to_string(trafficMsg->getDestination()) +"]";
        const char* c = strTmp.c_str();
        targetModule = getParentModule()->getSubmodule(c);
        if ( targetModule == nullptr )
            throw cRuntimeError("Cannot find Module");
    }

    // -----
    // checking if service node's position has changed since the message's destination was set
    Node* target = check_and_cast<Node *>(targetModule);

    if (target->serviceActive == false){
        routingCache = nullptr;
        int oldSource = trafficMsg->getSource();
        delete trafficMsg; // deleting old obsolete message

        // creating a trafficmsg with updated destination but the old source id
        trafficMsg = check_and_cast<Targ_msg *>(generateTrafficMessage(1, oldSource));
        if ( trafficMsg->getDestination() == 99999){
            delete trafficMsg;
            return;
        }
        targetModule = getParentModule()->getSubmodule("node", int(trafficMsg->getDestination())); // ne
        if ( targetModule == nullptr ){
            std::string strTmp = "node["+ std::to_string(trafficMsg->getDestination()) +"]";
            const char* c = strTmp.c_str();
            targetModule = getParentModule()->getSubmodule(c);
            if ( targetModule == nullptr )

```

```

        throw cRuntimeError("Cannot find Module");
    }
} // else no change is needed, can be routed safely as is..
// -----

if ( routingCache == nullptr ){
    cTopology::Node *targetNode = overseerTopo->getNodeFor(targetModule);
    overseerTopo->calculateWeightedSingleShortestPathsTo(targetNode);
    cTopology::Node *node = overseerTopo->getNodeFor(this);
    if (node == nullptr) {
        EV << "ERROR: This node (" << getFullPath() << ") is not included in this topology.\n";
        //delete trafficMsg;
    } else if (node->getNumPaths()==0) {
        EV << "ERROR: There exists no path to the destination from this node.\n";
        //delete trafficMsg;
    } else {
        bubble("routing..");
        //cTopology::LinkOut *path = node->getPath(0);
        routingCache = node->getPath(0)->getLocalGate();
        send(trafficMsg, routingCache);
    }
}
else{
    bubble("routing..");
    EV << " ROUTING WITH CACHE " << endl;
    send(trafficMsg, routingCache);
}

numRouted++;
battery -= transmitCost;
if(battery < 0){
    battery = 0;
    The_Overseer->writeBatteryScalars(this);
    endSimulation();
}
}
}

else if ( msg->getKind() == RELOCATE ){
    if ( serviceActive ){
        // Testing valid status of migration POLICY A (tree topology)
        EV << "RELOCATING" << endl;
        int maxIndex = -1;
        int maxTraffic = -1;
        int maxGateId = -1;
        int vSize = trafficStats.size();
        EV << "TrafficStats vector size: " << vSize << endl;
        for(int i = 0; i < vSize; i+=2){
            EV << "GateID: " << trafficStats[i] << ", Traffic Counts: " << trafficStats[i+1] << endl;
            if( trafficStats[i+1] > maxTraffic ){
                maxTraffic = trafficStats[i+1];
                maxIndex = i;
                maxGateId = trafficStats[i];
            }
        }
    }
}

```

```

}
The_Overseer->writeVectorCost(par("cost").doubleValue()); // writing cost to vector file

int outGateId = gateBaseId("port$o") + (maxGateId - gateBaseId("port$i"));

// if the traffic incoming from the top traffic neighbor is 10% larger than of the combined traffic
// if ( false) { // use this to ignore policy and keep migrating to the highest traffic neighboring
if ( 2 *maxTraffic < totalTrafficCount){ // if maxtraffic is smaller than 50% of total traffic
    // clear statistics and try to relocate after another 14 seconds
    EV << "MIGRATION POLICY NOT MET. WILL NOT MIGRATE. ATTEMPT AGAIN IN X SECONDS. STATISTICS RESET
    scheduleAt(simTime()+timeToCheckMigrate, new cMessage("relocation self event", RELOCATE));
    totalTrafficCount = 0;
    trafficStats.clear();

    // we have reached final service placement, call recording function and terminate simulation
    The_Overseer->writeScalars(this->par("cost").doubleValue()); // sending our service cost (last l

    delete msg;
    return;
    // endSimulation();
}
else{
    The_Overseer->hop();
    // carry out the migration
    cModule *neighbour = gate(outGateId)->getNextGate()->getOwnerModule();
    EV << "MOVING TOWARDS NODE: " << neighbour->getFullName() << endl;
    cMessage* srvcMove = new cMessage("migrating..", SERVICE_MIGRATION);
    send(srvcMove, outGateId);

    /* Gate IDs are input vector /output vector contiguous in vector gates..
    * and fully contiguous between input/output gates in scalar (non-vector) gates.
    *
    * Network.node[0]: parentLi: id=0, scalar gate, type:input
    * Network.node[0]: parentLo: id=1, scalar gate, type:output
    * Network.node[0]: leftLi: id=2, scalar gate, type:input
    * Network.node[0]: leftLo: id=3, scalar gate, type:output
    *
    * To adapt this code for general random topologies or any topology which requires a vector in o
    * considerable changes need to be made in the way you calculate the corresponding output gate t
    */

    // -- DEACTIVATING SERVICE NODE --
    serviceActive = false;
    routingCache = nullptr;
    totalTrafficCount = 0; // resetting statistics gathering
    cDisplayString& dispStr = getDisplayString();
    dispStr.setTagArg("i", 0, "misc/node_vs");
    trafficStats.clear(); // clearing the stat tracking vector
    // -- DEACTIVATING SERVICE NODE --

    // EV << "Restarting traffic from this node." << endl;
    // scheduleAt(simTime()+1, SrvDemandMsg);
}
}

```

```

else EV << "ERROR: Service already deactivated." << endl;

delete msg;
}

else if ( msg->getKind() == SERVICE_MIGRATION ){
    bubble("MIGRATION COMPLETE");

    // -- ACTIVATING SERVICE NODE --
    if ( !serviceActive ) {
        servicePosforMsgGenerator = this->getIndex();
        routingCache = nullptr;
        serviceActive = true;
        cDisplayString& dispStr = getDisplayString();
        dispStr.setTagArg("i", 0, "misc/town2");
    } else
    {
        EV << "ERROR: Service already active.\n";
    }
    // -- ACTIVATING SERVICE NODE --

    // Canceling traffic generation of this node

    // again relocate after 25 seconds
    scheduleAt(simTime()+timeToCheckMigrate, new cMessage("relocation self event", RELOCATE));

    delete msg;
}
}

```

```

Targ_msg* Node::generateTrafficMessage(int mode, int source)
{

```

```

    /*
     * Generates a traffic message, automatically targets it towards the current active service node.
     */
    // Generate SOURCE ADDRESS
    int src;
    if ( source == -1)
        src = getIndex(); // no source was provided, using current node as source
    else
        src = source; // source was provided
    bool activeService1 = false;
    if ( servicePosforMsgGenerator > 0 ){ // cleared value for servicePosforMsgGenerator is -1
        Node* target = check_and_cast<Node *>(getParentModule()->getSubmodule("node", servicePosforMsgGenerator))
        if (target->serviceActive)
            activeService1 = true; // else stays false
    }

    // Generate DESTINATION ADDRESS
    int dest = 99999;

    if ( mode == 1){ // TO THE ACTIVE SERVICE NODE

```

```

    if (activeService1){
        dest = servicePosforMsgGenerator;
    }else{
        // searching the currently active service node
        int numNodes = getParentModule()->getSubmodule("node", 0)->getVectorSize();
        // numNodes += The_Overseer->getCreatedNodes();
        for(int i = 0; i < numNodes; i++){
            Node* target = check_and_cast<Node *>(getParentModule()->getSubmodule("node", i));
            if (target->serviceActive){
                servicePosforMsgGenerator = i;
                dest = i; // tempmodule get index?
                break; // break since we found the active service node
            }
        }
    }
}

}else throw cRuntimeError("wrong message generation code.");

char msgname[100]; sprintf(msgname, "route n[%d]->n[%d]", src, dest);

// Create message object and set source and destination field.
Targ_msg *msg = new Targ_msg(msgname, TRAFFIC_MSG);
msg->setSource(src);
msg->setDestination(dest);
return (msg);
}

double Node::batteryLevel(){
    /*
     * Returns the node's current battery level.
     */
    return battery;
}

void Node::changeDemand(double newDemand){
    /*
     * Change the node's demand probability (from 0 to 1), does NOT change the demand probability parameter of t
     */
    if( newDemand < 0 || newDemand > 1){
        throw cRuntimeError("Invalid Demand value range.");
    }
    demandProbability = newDemand;
}

```

12 (old) OVERSEER.cc

```

#include <omnetpp.h>
#include <cmath>
#include <vector>

using namespace omnetpp;

#include "NODE.h"

```

```

class OVERSEER : public cSimpleModule
{
    public:
        OVERSEER();
        virtual ~OVERSEER();

        // Dynamic Node Creation
        void dynamicNodeCreate(int);
        int getCreatedNodes();
        void setCreatedNodes(int);
        void addCreatedNodes(int);
        void subtractCreatedNodes(int);
        void incrementCreatedNodes();

        // 1-median functions
        cModule* findMedian();
        cModule* findMedian_old();
        cModule* returnMedian();
        void buildTopology();

        // Scalar/Vector writing functions
        void writeScalars(double);
        void writeVectorCost(double);
        void hop();
        void writeBatteryScalars(cModule*);

        // Event functions
        void eventStep();
        void eventStop();
        void eventStart(double, double);
        void eventSetSpeed(double);
        void eventSetMovementFreq(double);
        void eventSetRadius(double);

        void eventCalcAffected();
        void eventRestoreAffected();

        void calculateSteps(double, double);

        cTopology topo;

protected:
    virtual void initialize(int) override;
    virtual int numInitStages() const override { return 4;};
    virtual void handleMessage(cMessage *msg) override;

private:
    int createdNodes;
    int hops;
    cModule* median1;
    cModule* prevMedian1;
    cOutVector costVector;

```

```
// "Event" variables
int pointsOfMovement;
int movingToPoint;
double currentX;
double currentY;
int numSteps;
double stepX;
double stepY;
double eventSpeed;           // speed, how much do you want to move per movement [0,1]
double eventMoveFreq;        // how often does it move ( in seconds )
double eventRadius;           // the radius of the affected area, center is X,Y
cMessage* eventMoveMsg;       // msg timing the movements of the event area
std::vector<int> affectedNodes; // vector containing the currently affected nodes's indexes
std::vector<double> pathPoints;
```

```
ERSEER::initialize(int stage)
```

```
if ( stage == 0){
    hops = 0;
    createdNodes = 0; WATCH(createdNodes);
    median1 = prevMedian1 = nullptr;

    double myPoints[] = {1.0, 0.5, 0, 0.9, 0.35, 0.1, 1.0, 0.5, 0, 0.9, 0.35, 0.1, 1.0, 0.5, 0, 0.9, 0.35, 0};
    pointsOfMovement = (sizeof(myPoints)/sizeof(*myPoints)) /2;
    std::copy(&myPoints[0], &myPoints[sizeof(myPoints)/sizeof(*myPoints)], back_inserter(pathPoints));

    EV << "POINTS OF MOVEMENT: " << pointsOfMovement << endl;
    movingToPoint = 0;
    eventMoveMsg = nullptr;
    currentX = 0;
    currentY = 0;
    eventRadius = 0.1;
    eventMoveFreq = 8.0;
    eventSpeed = 0.01;
    ////////////////////////////////////
    // MANUAL EVENT MODE //
    //currentX = 1;
    //currentY = 1;
    //scheduleAt(simTime()+300, new cMessage("Event", 707));
    ////////////////////////////////////

    calculateSteps(pathPoints[movingToPoint*2], pathPoints[movingToPoint*2+1]);
    eventMoveMsg = new cMessage("Event", 707);
    scheduleAt(simTime()+eventMoveFreq+0.2, eventMoveMsg); // TODO UNCOMMENT FOR EVENT
}

if ( stage == 2){
    if(getAncestorPar("performanceMode").boolValue()){
        // PERFORMANCE MODE - Disables 1-median Calculation (!! WARNING !! Does not check for network connection)
        buildTopology();
    }
}
```

```

    else{
        // NORMAL MODE
        findMedian(); // builds weighted topology and finds 1-median. (Topology object should remain correct
        //scheduleAt(simTime()+30, new cMessage("OVERSEER EVENT", 808));
    }
}

}

void OVERSEER::handleMessage(cMessage *msg)
{
    if ( msg->isSelfMessage() && msg->getKind() == 808){
        //findMedian();
        //scheduleAt(simTime()+10, msg);
        //dynamicNodeCreate(1);
    }
    if (msg->getKind() == 707){
        //eventCalcAffected(); // manual mode

        // MANUAL MODE ACTIVATED

        if(numSteps == 0){
            movingToPoint += 1; // since we finished the move to this point, increment the point we're moving to
            calculateSteps(pathPoints[movingToPoint*2], pathPoints[movingToPoint*2 +1]); // calculate the steps
        }
        eventStep();
        numSteps--;
        scheduleAt(simTime()+eventMoveFreq, msg);
    }
}

OVERSEER::OVERSEER()
{
    eventMoveMsg = nullptr;
}

OVERSEER::~OVERSEER()
{
    cancelAndDelete(eventMoveMsg);
}

void OVERSEER::eventStep(){
    currentX = currentX + stepX;
    currentY = currentY + stepY;
    eventRestoreAffected();
    eventCalcAffected();
}

void OVERSEER::calculateSteps(double targX, double targY){
    int numStepsY = int(round(abs((targY-currentY)/eventSpeed)));
    int numStepsX = int(round(abs((targX-currentX)/eventSpeed)));

```

```

    if (numStepsX == numStepsY) numSteps = numStepsX;
    else if (numStepsX < numStepsY) numSteps = numStepsY;
    else numSteps = numStepsX;

    stepX = (targX - currentX)/numSteps;
    stepY = (targY - currentY)/numSteps;
    EV << "Number of Steps: " << numSteps << "\nStep X: " << stepX << "\nStep Y: " << stepY << endl;
}

void OVERSEER::eventCalcAffected(){
    /*
     * Calculates the affected nodes in the area of the event and sets their demand to 1
     * (event should already be initialized)
     */
    double X = currentX;
    double Y = currentY;
    int N = getAncestorPar("numNodes").intValue();
    Node* node = nullptr;
    for(int i = 0; i < N; i++){
        node = check_and_cast<Node*>(getParentModule()->getSubmodule("node", i));
        if ( sqrt(pow((X - node->par("xcoor").doubleValue()), 2) + pow((Y - node->par("ycoor").doubleValue()), 2) ) < 2 )
            affectedNodes.push_back(i);
        node->changeDemand(1);
        node->getDisplayString().setTagArg("i", 1, "gold");
    }
}

void OVERSEER::eventRestoreAffected(){
    /*
     * Return the affected nodes's demand to their original value (that they get from the .ini on network initia
     */
    int vSize = affectedNodes.size();
    Node* node = nullptr;
    for (int i = 0; i < vSize; i++){
        node = check_and_cast<Node*>(getParentModule()->getSubmodule("node", affectedNodes[i]));
        node->changeDemand(node->par("mydem").doubleValue());
        node->getDisplayString().setTagArg("i", 1, "white");
    }
    affectedNodes.clear();
}

void OVERSEER::writeBatteryScalars(cModule* caller){
    /*
     * Calculates Mean Battery level, and Battery Variance and stores them in Scalar output files. (called by th
     */
    recordScalar("Time of Network Death", simTime());
    int numNodes = getParentModule()->getSubmodule("node", 0)->getVectorSize();
    double batterySum = 0;
    Node* tempnode = nullptr;

```

```

for(int n = 0; n < numNodes; n++){
    tempnode = check_and_cast<Node *>(getParentModule()->getSubmodule("node", n));
    batterySum += tempnode->batteryLevel();
}
double Mean = batterySum / numNodes;
recordScalar("Network battery average", Mean);
EV << "Network battery average: " << batterySum << endl;

double meanDiffSum;
for(int n = 0; n < numNodes; n++){
    tempnode = check_and_cast<Node *>(getParentModule()->getSubmodule("node", n));
    meanDiffSum = pow((tempnode->batteryLevel() - Mean), 2);
}

// TODO Bessel's Correction n-1 or not?
recordScalar("Battery level variance", meanDiffSum / numNodes - 1);
/*
if (serviceActive && getAncestorPar("knownMedianNode").boolValue()){
    // PERFORMANCE CHEAT: calculating distance to median when we already know who the 1-median is (same seed)
    overseerTopo->calculateWeightedSingleShortestPathsTo(overseerTopo->getNodeFor(getParentModule()->getSubmodule("knownMedianNode")));
    cTopology::Node *node = overseerTopo->getNodeFor(this); // this is the current "node" (the root node of the min spanning tree)
    recordScalar("Weighted Distance to Median", node->getDistanceToTarget());

    overseerTopo->calculateUnweightedSingleShortestPathsTo(overseerTopo->getNodeFor(getParentModule()->getSubmodule("knownMedianNode")));
    recordScalar("Unweighted Distance to Median", node->getDistanceToTarget());
}
else{
    */

topo.calculateWeightedSingleShortestPathsTo(topo.getNodeFor(median1));
cTopology::Node *node = topo.getNodeFor(caller); // this is the current "node" (the root node of the min spanning tree)
double omnetWeightDistance = node->getDistanceToTarget();
topo.calculateUnweightedSingleShortestPathsTo(topo.getNodeFor(median1));
double omnetNumOfHops = node->getDistanceToTarget();
recordScalar("Hop Distance to Median", omnetNumOfHops);

recordScalar("Weighted (Cost) Distance to Median", omnetWeightDistance);

// (ignore this) COMMENT: Omnet's calcWeightPaths function, adds 1.0 for every "hop" the path has, in addition to the weighted distance.
// this interferes a lot when we are working with small distances, so we subtract it from the weighted distance.
/*
if (omnetNumOfHops==0){
    recordScalar("Weighted (Cost) Distance to Median", omnetWeightDistance);
}else{
    recordScalar("Weighted (Cost) Distance to Median", omnetWeightDistance - (omnetNumOfHops - 1));
}
*/
}

void OVERSEER::writeScalars(double final_cost){
    recordScalar("Hops", hops);

    // Absolute cost reduction

```

```

double initCost = getParentModule()->getSubmodule("node", this->par("starter").intValue()->par("cost").doubleValue();
recordScalar("Absolute Cost Reduction", (initCost - final_cost));

// Percentage cost reduction
recordScalar("Percent Cost Reduction", ((initCost - final_cost)/initCost)*100);

recordScalar("Time to descend", simTime());
}

void OVERSEER::writeVectorCost(double cost){
    costVector.recordWithTimestamp(simTime(), cost);
}

void OVERSEER::hop(){
    hops++;
}

int OVERSEER::getCreatedNodes(){
    /**
     * Returns the number of nodes that were created dynamically since the start of the current simulation.
     */
    if (createdNodes >= 0){
        return (createdNodes);
    }
    else
        throw cRuntimeError("Major Error! Created nodes counter is negative!");
}

void OVERSEER::setCreatedNodes(int value){
    /**
     * Sets the created nodes counter to the given non-negative value.
     */
    if ( value > 0){
        createdNodes = value;
    }
    else
        throw cRuntimeError("Error! Cannot set created nodes counter to a negative value!", value);
}

void OVERSEER::addCreatedNodes(int value){
    /**
     * Sums the given integer value to the created nodes counter.
     */
    if(value > 0){
        createdNodes += value;
    }
    else
        throw cRuntimeError("Invalid value %d. Must be a positive integer, use subtractCreatedNodes for subtract");
}

```

```

}

void OVERSEER::subtractCreatedNodes(int value){
    /**
     * Subtracts the given integer value from the created nodes counter.
     */
    if(value < 0){
        if(createdNodes - value > 0){
            createdNodes -= value;
        }
        else
            throw cRuntimeError("Error! Subtracting %d from createdNodes puts it below zero!", value);
    }
    else
        throw cRuntimeError("Invalid value %d. Must be a negative integer, use addCreatedNodes for subtraction o
}

void OVERSEER::incrementCreatedNodes(){
    createdNodes++;
}

void OVERSEER::buildTopology(){
    topo.extractByProperty("topoIncl", "yes"); // EV << topo.str() << endl;
    int numNodes = topo.getNumNodes();
    double sumPathCost = 0;
    double minPathCost = INFINITY;
    int minN = -1;

    cTopology::Node *current_node = nullptr;
    for( int n = 0; n < numNodes; n++){
        if ( getParentModule()->getSubmodule("node", n) == nullptr ){
            // we are most likely dealing with a runtime dynamically created node that has no index z
            std::string strTmp = "node["+ std::to_string(n) +"]";
            const char* c = strTmp.c_str();
            if ( getParentModule()->getSubmodule(c) == nullptr ){
                cRuntimeError("(FindMedian) Cannot find Module");
            }
            else{
                current_node = topo.getNodeFor(getParentModule()->getSubmodule(c));
            }
        }else{
            current_node = topo.getNodeFor(getParentModule()->getSubmodule("node", n));
        }
        //current_node->setWeight(1); //(current_node->getModule()->par("node_demand").doubleValue()); // Initia
        //cDisplayString& dispStr = getDisplayString();
        //dispStr.setTagArg("t", 0, current_node->getWeight());

        // Initialize link weights
        // Note: You can access channels from "out"--> gates only, so we only work with those here, even tho it'
        double tmpw = 0;

```

```

    for (int i = 0; i < current_node->getNumInLinks(); i++) {
        if (current_node->getLinkIn(i)->isEnabled()){
            tmpw = current_node->getLinkIn(i)->getRemoteGate()->getChannel()->par("weight").doubleValue();
            current_node->getLinkIn(i)->setWeight(tmpw);
            //EV << "IN LINK " << i <<" - weight: " << current_node->getLinkIn(i)->getWeight() << " // coming in
        }
    }
    for (int i = 0; i < current_node->getNumOutLinks(); i++) {
        if (current_node->getLinkOut(i)->isEnabled()){
            tmpw = current_node->getLinkOut(i)->getLocalGate()->getChannel()->par("weight").doubleValue();
            current_node->getLinkOut(i)->setWeight(tmpw);
            //EV << "OUT LINK " << i <<" - weight: " << current_node->getLinkOut(i)->getWeight() << " // going out
        }
    }
}
}
}

```

```

cModule* OVERSEER::findMedian(){ // NEW PERFORMANCE ORIENTED 1-MEDIAN CALCULATOR. NOT FOR DYNAMIC NETWORKS (adding nodes)
/*
 * Calculates the 1-median in a general network topology with vector gates
 * Returns the module which is the median as a cModule pointer
 */

```

```

topo.extractByProperty("topoIncl", "yes");
int numNodes = topo.getNumNodes();
double sumPathCost = 0;
double minPathCost = INFINITY;
int minN = -1;

```

// TOPOLOGY BUILD

```

cTopology::Node *current_node = nullptr;
for( int n = 0; n < numNodes; n++){
    current_node = topo.getNodeFor(getParentModule()->getSubmodule("node", n));
    //current_node->setWeight(1); // Initialize Node weights (node weights = 1, calculate weighted paths only)

    // Initialize link weights
    // Note: You can access channels from "out"--> gates only, so we only work with those here, even though
    double tmpw = 0;
    for (int i = 0; i < current_node->getNumInLinks(); i++) {
        if (current_node->getLinkIn(i)->isEnabled()){
            tmpw = current_node->getLinkIn(i)->getRemoteGate()->getChannel()->par("weight").doubleValue();
            current_node->getLinkIn(i)->setWeight(tmpw);
        }
    }
    for (int i = 0; i < current_node->getNumOutLinks(); i++) {
        if (current_node->getLinkOut(i)->isEnabled()){
            tmpw = current_node->getLinkOut(i)->getLocalGate()->getChannel()->par("weight").doubleValue();
            current_node->getLinkOut(i)->setWeight(tmpw);
        }
    }
}
}

```

// 1-MEDIAN CALCULATION

```

cModule* Temp;
cModule* Temp2;
cTopology::Node *targetnode = nullptr;
cTopology::Node *node = nullptr;
double demand = 0;
for( int n = 0; n < numNodes; n++){
    Temp = getParentModule()->getSubmodule("node", n);
    targetnode = topo.getNodeFor(Temp);
    topo.calculateWeightedSingleShortestPathsTo(targetnode);
    for (int i = 0; i < numNodes; i++) {
        Temp2 = getParentModule()->getSubmodule("node", i);
        demand = Temp2->par("mydem").doubleValue();
        node = topo.getNodeFor(Temp2);
        if(node->getNumPaths() == 0){
            if(i == n) continue;
            throw cRuntimeError("Critical Error! Disconnected Network.");
        }
        sumPathCost += (node->getDistanceToTarget() * demand);
        //sumPathCost += node->getDistanceToTarget();
    }

    if (sumPathCost < minPathCost){ // minPathCost is the minimum cost of the minimum spanning tree create
        minPathCost = sumPathCost; // sumPathCost is the sum of the costs of each weighted shortest pat
        minN = n;
    }
    Temp->par("cost").setDoubleValue(sumPathCost);
    Temp->getDisplayString().setTagArg("t", 0, sumPathCost);
    sumPathCost = 0;
}

if( minPathCost > 0 ){
    recordScalar("Median", minN);
    median1 = getParentModule()->getSubmodule("node", minN);
    median1->getDisplayString().setTagArg("i", 0, "misc/house");
    return (median1);
}
throw cRuntimeError("Critical Error! Couldn't calculate median.");
}

cModule* OVERSEER::returnMedian(){
    return median1;
}

cModule* OVERSEER::findMedian_old(){ // OLD GENERAL PURPOSE 1MEDIAN FINDER
    /*
     * Calculates the 1-median in a general network topology with vector gates
     * Returns the module which is the median as a cModule pointer
     */
    if ( median1 != nullptr ){
        prevMedian1 = median1;
        prevMedian1->getDisplayString().setTagArg("i", 0, "misc/node_vs");
    }
}

```

```

topo.extractByProperty("topoIncl", "yes"); // EV << topo.str() << endl;
int numNodes = topo.getNumNodes();
double sumPathCost = 0;
double minPathCost = INFINITY;
int minN = -1;

cTopology::Node *current_node = nullptr;
for( int n = 0; n < numNodes; n++){
    if ( getParentModule()->getSubmodule("node", n) == nullptr ){
        // we are most likely dealing with a runtime dynamically created node that has no index z
        std::string strTmp = "node["+ std::to_string(n) +"]";
        const char* c = strTmp.c_str();
        if ( getParentModule()->getSubmodule(c) == nullptr ){
            cRuntimeError("(FindMedian) Cannot find Module");
        }
        else{
            current_node = topo.getNodeFor(getParentModule()->getSubmodule(c));
        }
    }else{
        current_node = topo.getNodeFor(getParentModule()->getSubmodule("node", n));
    }
    // current_node->setWeight(1); //(current_node->getModule()->par("node_demand").doubleValue()); // Initia

    // Initialize link weights
    // Note: You can access channels from "out"---> gates only, so we only work with those here, even tho it'
    double tmpw = 0;
    for (int i = 0; i < current_node->getNumInLinks(); i++) {
        if (current_node->getLinkIn(i)->isEnabled()){
            tmpw = current_node->getLinkIn(i)->getRemoteGate()->getChannel()->par("weight").doubleValue(); /
            current_node->getLinkIn(i)->setWeight(tmpw);
            //EV << "IN LINK " << i <<" - weight: " << current_node->getLinkIn(i)->getWeight() << " // comin
        }
    }
    for (int i = 0; i < current_node->getNumOutLinks(); i++) {
        if (current_node->getLinkOut(i)->isEnabled()){
            tmpw = current_node->getLinkOut(i)->getLocalGate()->getChannel()->par("weight").doubleValue(); /
            current_node->getLinkOut(i)->setWeight(tmpw);
            //EV << "OUT LINK " << i <<" - weight: " << current_node->getLinkOut(i)->getWeight() << " // goi
        }
    }
}
for( int n = 0; n < numNodes; n++){
    // Calculating Shortest Weighted Paths for each node n to every other node i in the network
    cModule *Temp = nullptr;
    for (int i = 0; i < numNodes; i++) {
        cModule *target_module = getParentModule()->getSubmodule("node", i);
        if ( target_module == nullptr ){
            std::string strTmp = "node["+ std::to_string(i) +"]";
            const char* c = strTmp.c_str();
            target_module = getParentModule()->getSubmodule(c);
            if ( target_module == nullptr )
                cRuntimeError("(FindMedian) Cannot find Module");
        }
        cTopology::Node *targetnode = topo.getNodeFor(target_module);

```

```

topo.calculateWeightedSingleShortestPathsTo(targetnode);
/* This performs the Dijkstra algorithm and stores the result in the cTopology object.
* The result can then be extracted using cTopology and cTopology::Node methods. */

/*
* The purpose of the getDistanceToTarget() member function of a node is self-explanatory.
* In the unweighted case, it returns the number of hops.
* The getNumPaths() member function returns the number of edges which are part of a shortest path,
* and path(i) returns the ith edge of them as cTopology::LinkOut.
* If the shortest paths were created by the ...SingleShortestPaths() function,
* getNumPaths() will always return 1 (or 0 if the target is not reachable),
* that is, only one of the several possible shortest paths are found.
* The ...MultiShortestPathsTo() functions find all paths, at increased run-time cost.
* The cTopology's getTargetNode() function returns the target node of the last shortest path search.
*/
cTopology::Node *node = nullptr;
if ( getParentModule()->getSubmodule("node", n) == nullptr ){
    std::string strTmp = "node["+ std::to_string(n) +"]";
    const char* c = strTmp.c_str();
    if (getParentModule()->getSubmodule(c) == nullptr ){
        cRuntimeError("(FindMedian) Cannot find Module");
    }
    else{
        node = topo.getNodeFor(getParentModule()->getSubmodule(c));
    }
}
else{
    node = topo.getNodeFor(getParentModule()->getSubmodule("node", n));
}

if (node == nullptr){
    EV << "This module: (" << getFullPath() << ") is not included in the topology.\n";
}
else if (node->getNumPaths() == 0){
    if( i == n ){ // don't print "unreachable" because we are the same node..
        continue;
    }
    EV << "No path to destination. (Unreachable)\n";
    throw cRuntimeError("Critical Error! UNCONNECTED NETWORK!!!");
    endSimulation();
}
else{
    Temp = node->getModule(); // this is the current "node" (the root node of the min spanning tree
    double demand = topo.getTargetNode()->getModule()->par("mydem").doubleValue(); // the demand val

    sumPathCost += (node->getDistanceToTarget() * demand) ; // Distance in unweighted means "hops",
    // Debug outputs, not needed for function's goal
    EV << "=TRAVERSING PATH=: " << node->getModule()->getFullName() << " to " << topo.getTargetNode(
    EV << node->getModule()->getFullName();
    while (node != topo.getTargetNode()){
        /* OLD EVs

```

```

        EV << "We are in " << node->getModule()->getFullPath() << endl;
        EV << node->getDistanceToTarget() << " hops to go\n"; // for unweighted case should be integ
        EV << "There are " << node->getNumPaths() << " equally good directions, taking the first one

        cTopology::LinkOut *path = node->getPath(0);
        EV << "Taking gate " << path->getLocalGate()->getFullName() << " we arrive in " << path->get
        */
        cTopology::LinkOut *path = node->getPath(0);
        EV << " -> " << path->getRemoteNode()->getModule()->getFullName();
        node = path->getRemoteNode();
    }
    EV << endl;
}

}

if ( sumPathCost < minPathCost ){ // minPathCost is the minimum cost of the minimum spanning tree create
                                // sumPathCost is the sum of the costs of each weighted shortest path
    minPathCost = sumPathCost;
    minN = n;
    EV << "minN: " << minN << ".. minPathCost: " << minPathCost << endl;
}
EV<< "COST OF NODE[" << n << "] = " << sumPathCost << endl;
Temp->par("cost").setDoubleValue(sumPathCost); // setting the node's cost parameter value
Temp->getDisplayString().setTagArg("t", 0, sumPathCost);
sumPathCost = 0;
}
if( minPathCost > 0.1 ){
    recordScalar("Median", minN);
    if ( getParentModule()->getSubmodule("node", minN) == nullptr ){
        std::string strTmp = "node["+ std::to_string(minN) +"]";
        const char* c = strTmp.c_str();
        if ( getParentModule()->getSubmodule(c) == nullptr ){
            cRuntimeError("(FindMedian) Cannot find Module");
        }
        else{
            median1 = getParentModule()->getSubmodule(c);
            median1->getDisplayString().setTagArg("i", 0, "misc/house");
            return (median1);
        }
    }
    else{
        median1 = getParentModule()->getSubmodule("node", minN);
        median1->getDisplayString().setTagArg("i", 0, "misc/house");
        return (median1);
    }
}
return (nullptr);
}

void OVERSEER::dynamicNodeCreate(int numToCreate){
    cModuleType *moduleType = cModuleType::get("Node"); // Find the factory object (created by Define_Module and

```

```

int createdSoFar = getCreatedNodes();
int originalNumberOfNodes = getParentModule()->par("numNodes").intValue();
double radiusOfConnectivity = getParentModule()->par("rc").doubleValue();

for (int i = originalNumberOfNodes+createdSoFar; i < (createdSoFar + originalNumberOfNodes + numToCreate); i
incrementCreatedNodes(); // counter if we want to create more later...
// Figure out new node's index, and name and creating name string (because create function requires cons

std::string str = "node["+ std::to_string(i) +"]";
const char* c = str.c_str();
// Create module (node)
cModule* module = moduleType->create(c, getParentModule()); // getParentModule returns the top level "Ne

// Setting up parameters and gate sizes before we set up its submodules
// module->par("service") = false; // don't need this because of =default(false) (I THINK)
double xcoor = uniform(0,1000);
double ycoor = uniform(0,1000);
module->par("xcoor").setDoubleValue(xcoor);
module->par("ycoor").setDoubleValue(ycoor);
module->par("init").setBoolValue(false);

module->finalizeParameters();

// Setting up gates and connections
// find channel factory object and create a channel
cChannelType *channelType = cChannelType::get("Network.Channel");
cChannel *channel = nullptr;

// Creating connections
cModule* tM = nullptr;
int numConn = 0;

for ( int z = 0; z < originalNumberOfNodes+createdSoFar; z++){
    tM = getParentModule()->getSubmodule("node", z);
    if ( tM == nullptr ){
        // we are most likely dealing with a runtime dynamically created node that has no index z
        std::string strTmp = "node["+ std::to_string(z) +"]";
        const char* c = strTmp.c_str();
        tM = getParentModule()->getSubmodule(c);
        if ( tM == nullptr )
            cRuntimeError("Cannot find Module");
    }

    if ( sqrt(std::pow((xcoor - tM->par("xcoor").doubleValue()), 2) + std::pow((ycoor - tM->par("ycoor")
EV << "CONNECTING!" << endl;
    numConn++;

    /*
    // Making the gate vectors names for the newly created node
    std::string strIn = "portIn["+ std::to_string(numConn-1) +"]";
    std::string strOut = "portOut["+ std::to_string(numConn-1) +"]";
    const char* cIn = strIn.c_str();
    const char* cOut = strOut.c_str();

```

```

// Making the gate vector names for the neighbor node
std::string strnIn = "portLi["+ std::to_string(tM->gateSize("port") + 1) +"]";
std::string strnOut = "portLo["+ std::to_string(tM->gateSize("port") + 1) +"]";
const char* cnIn = strnIn.c_str();
const char* cnOut = strnOut.c_str();
*/

// Setting the gate vector size of the new node
module->setGateSize("port", numConn);

// Changing the gate vector size of the neighboring node that we are creating a connection to
tM->setGateSize("port", tM->gateSize("port") + 1);

// Finally, creating the connections
module->gate("port$o", numConn -1 )->connectTo(tM->gate("port$i", tM->gateSize("port") -1), chan
tM->gate("port$o", tM->gateSize("port") -1)->connectTo(module->gate("port$i", numConn -1), chann

```

```

}
}

```

```

// Create internals, and schedule it
module->buildInside();
module->callInitialize(); // call initializer of the module
module->scheduleStart(simTime());
}

```

```

}

```

```

void cTopology::calculateWeightedSingleShortestPathsTo(Node *_target)
{
    EV << "Weighted Shortest Path To: " << target->getModule()->getFullName() << endl;
    if (!_target)
        throw cRuntimeError(this, "..ShortestPathTo(): Target node is nullptr");
    target = _target;
    for (auto & node : nodes) { // nodes is vector of Node*
        node->dist = INFINITY;
        node->outPath = nullptr;
    }
    target->dist = 0;
    std::list<Node *> q;
    q.push_back(target);

    while (!q.empty()) {
        Node *dest = q.front();
        q.pop_front();
        ASSERT(dest->getWeight() >= 0.0); // If getWeight negative or not a number, throw runtime error

        // for each neighbor of the dest node
        for (int i = 0; i < dest->getNumInLinks(); i++) {

            // For every link: if link is disabled skip this loop

```

```

if (!(dest->getLinkIn(i)->isEnabled()))
    continue;

// If link is enabled, check if the node it connects to is enabled, if not, skip this loop
Node *src = dest->getLinkIn(i)->getRemoteNode();
if (!src->isEnabled())
    continue;

double linkWeight = dest->getLinkIn(i)->getWeight(); //->getRemoteGate()->getChannel()->par("weight")
ASSERT(linkWeight > 0.0);
EV << "Link weight " << dest->getModule()->getFullName() << " --> " << dest->getLinkIn(i)->getRemoteNode()
    << " weight: " << linkWeight << endl;

double newdist = dest->dist + linkWeight;
if (dest != target){
    newdist += dest->getWeight(); // dest is not the target, uses weight of dest node as price of r
    EV << "dest != target: newdist = " << newdist << endl;
}

if (newdist != INFINITY && src->dist > newdist) { // it's a valid shorter path from src to target n
    EV << "it's a valid shorter path from src to target node" << endl;
    if (src->dist != INFINITY)
        q.remove(src); // src is in the queue
    src->dist = newdist;
    src->outPath = dest->inLinks[i];

    // insert src node to ordered list
    std::list<Node *>::iterator it;
    for (it = q.begin(); it != q.end(); ++it)
        if ((*it)->dist > newdist)
            break;

    q.insert(it, src);
}
}
}
}
}

```

ssm-testing

13 NODE.CC

```

#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include <vector>

#include "targ_msg_m.h"

#include "OVERSEER.h"

#define TRAFFIC_MSG 40
#define TRAFFIC_GEN 41
#define SERVICE_MIGRATION 50
#define RELOCATE 52 // self message testing service relocation - placeholder for policy service migration

```

```

#define POLICY_STAT 51 // self message to time stat tracking for migration policy

using namespace omnetpp;

class Node: public cSimpleModule {
public:
    Node();
    virtual ~Node();
    cTopology topo;
    bool serviceActive = false;

protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *) override;
    Targ_msg* generateTrafficMessage(int mode, int source = -1); // mode 0: random, mode 1: to service node
    void dynamicNodeCreate(int numToCreate);
    //virtual void initialize(int) override;
    //virtual int numInitStages() const override { return 3;}; // 3 stage initialization

private:
    OVERSEER *The_Overseer; //personal pointer to the overseer module
    cMessage* SrvDemandMsg;
    int totalTrafficCount;
    std::vector<int> trafficStats; // have to use the std:: namespace to use the vector class
};

Define_Module(Node);

void Node::initialize() {
    SrvDemandMsg = new cMessage("demand timer", TRAFFIC_GEN);
    totalTrafficCount = 0; WATCH(totalTrafficCount);
    WATCH(serviceActive);
    WATCH_VECTOR(trafficStats);

    if( par("service") ) {
        serviceActive = true;
        cDisplayString& dispStr = getDisplayString();
        dispStr.setTagArg("i", 0, "misc/town2");
        scheduleAt(simTime()+15.3, new cMessage("relocation self event", RELOCATE));
    }
    else scheduleAt(simTime()+5.0, SrvDemandMsg);

    // OVERSEER METHODS
    cModule *targetModule = getParentModule()->getSubmodule("The_Overseer");
    The_Overseer = check_and_cast<OVERSEER *>(targetModule);

    for (cModule::GateIterator i(this); !i.end(); i++) {
        cGate *gate = *i;
        EV << gate->getFullName() << ": " << "id=" << gate->getId() << ", ";

        if (!gate->isVector())
            EV << "scalar gate, ";
        else
            EV << "gate " << gate->getIndex() << " in vector " << gate->getName() << " of size " << gate->getVecSize() << ", ";
            EV << "type:" << cGate::getTypeName(gate->getType()) << endl;
    }
}

```

```

    }
}

Node::Node() {
    SrvDemandMsg = nullptr;
}

Node::~Node() {
    cancelAndDelete(SrvDemandMsg);
}

void Node::handleMessage(cMessage *msg) {
    if ( msg->getKind() == TRAFFIC_GEN ) {
        EV << "TRAFFIC_GEN SELF MESSAGE RECEIVED \n";
        if ( serviceActive ){
            EV << "Active Service so no traffic generation, returning. \n";
            return;
        }
        Targ_msg *trafficMsg = generateTrafficMessage(1); // NEW TRAFFIC MESSAGE IN THE NETWORK
        if ( trafficMsg->getDestination() == 99999){
            // service node is in the process of migrating
            EV << "[Can't find service node] ABORTING... \n";
            delete trafficMsg;
            scheduleAt(simTime()+5.0, SrvDemandMsg); // try again later.... i guess...
            return;
        }

        topo.extractByProperty("topoIncl", "yes");
        cModule *targetModule = getParentModule()->getSubmodule("node", int(trafficMsg->getDestination()));
        cTopology::Node *targetNode = topo.getNodeFor(targetModule);
        topo.calculateUnweightedSingleShortestPathsTo(targetNode);
        cTopology::Node *node = topo.getNodeFor(this);
        if (node == nullptr) {
            EV << "ERROR: This node (" << getFullPath() << ") is not included in this topology.\n";
        } else if (node->getNumPaths()==0) {
            EV << "ERROR: There exists no path to the destination from this node.\n";
        } else {
            cTopology::LinkOut *path = node->getPath(0);
            send(trafficMsg, path->getLocalGate());
        }

        scheduleAt(simTime()+5.0, SrvDemandMsg); // generate continuous traffic by scheduling a TRAFFIC_GEN mess
    }

    else if ( msg->getKind() == TRAFFIC_MSG ){
        Targ_msg *trafficMsg = check_and_cast<Targ_msg *>(msg);

        if ( serviceActive ){
            bubble("Reached!");
            // --- POLICY MIGRATION STATISTICS ---
            totalTrafficCount++;
            bool incremented = false;
            int vSize = trafficStats.size();
            for(int i = 0; i < vSize; i+=2){

```

```

    // 0, 2, 4... gateIds / 1, 3, 5... message count
    if (int(trafficMsg->getArrivalGateId()) == trafficStats[i]){
        // we got traffic from a gate already in the vector array
        trafficStats[i+1]++;
        incremented = true;
    }
}
if (!incremented) {
    // we didn't find the gate in the vector so we have to document it
    trafficStats.push_back( int(trafficMsg->getArrivalGateId()) );
    trafficStats.push_back(1); // this is the first time we see this gate
}
// --- POLICY MIGRATION STATISTICS ---

if (trafficMsg->getDestination() == getIndex()){
    EV << "Case 1, the service node is currently active, and the message was routed successfully to
}
else if (trafficMsg->getDestination() != getIndex()){
    EV << "Case 2, the service is active, but the message hasn't had an update in its routing yet" << endl;
}
delete trafficMsg;
} else
{
    if (trafficMsg->getDestination() == getIndex()){
        EV << "we used to be an active service point, but this message didn't get rerouted in time" << endl;
        delete trafficMsg;
        return;
    }
    // Case 4, we are neither an active service, nor the target of the traffic, re-routing towards the c
    topo.extractByProperty("topoIncl", "yes");
    cModule *targetModule = getParentModule()->getSubmodule("node", int(trafficMsg->getDestination()));

    // -----
    // checking if service node's position has changed since the message's destination was set
    Node* target = check_and_cast<Node *>(targetModule);

    if (target->serviceActive == false){

        int oldSource = trafficMsg->getSource();
        int oldServiceNode = trafficMsg->getDestination();
        delete trafficMsg; // deleting old obsolete message

        // creating a trafficmsg with updated destination but the old source id
        trafficMsg = check_and_cast<Targ_msg *>(generateTrafficMessage(1, oldSource));
        if ( trafficMsg->getDestination() == 99999){
            // if service node is in the process of migrating, route towards the last known service node
            EV << "[Can't find service node] Setting Destination to: " << oldServiceNode << endl;
            trafficMsg->setDestination(oldServiceNode);
        }
        targetModule = getParentModule()->getSubmodule("node", int(trafficMsg->getDestination())); // ne
    } // else no change is needed, can be routed safely as is..
    // -----

    cTopology::Node *targetNode = topo.getNodeFor(targetModule);

```

```

topo.calculateUnweightedSingleShortestPathsTo(targetNode);
cTopology::Node *node = topo.getNodeFor(this);
if (node == nullptr) {
    EV << "ERROR: This node (" << getFullPath() << ") is not included in this topology.\n";
    //delete trafficMsg;
} else if (node->getNumPaths()==0) {
    EV << "ERROR: There exists no path to the destination from this node.\n";
    //delete trafficMsg;
} else {
    bubble("routing..");
    cTopology::LinkOut *path = node->getPath(0);
    send(trafficMsg, path->getLocalGate());
}
}
}

else if ( msg->getKind() == RELOCATE ){
    if ( serviceActive ){
        // Testing valid status of migration POLICY A (tree topology)
        EV << "RELOCATING" << endl;
        int maxIndex = -1;
        int maxTraffic = -1;
        int maxGateId = -1;
        int vSize = trafficStats.size();
        EV << "TrafficStats vector size: " << vSize << endl;
        for(int i = 0; i < vSize; i+=2){
            EV << "GateID: " << trafficStats[i] << ", Traffic Counts: " << trafficStats[i+1] << endl;
            if( trafficStats[i+1] > maxTraffic ){
                maxTraffic = trafficStats[i+1];
                maxIndex = i;
                maxGateId = trafficStats[i];
            }
        }

        // if the traffic incoming from the top traffic neighbor is 10% larger than of the combined traffic
        if ( maxTraffic < totalTrafficCount-maxTraffic + 10*((totalTrafficCount-maxTraffic)/100)) {
            // try to relocate after another 14 seconds
            scheduleAt(simTime()+15.3, new cMessage("relocation self event", RELOCATE));
        }
        else{
            // carry out the migration
            cModule *neighbour = gate(maxGateId + 1)->getNextGate()->getOwnerModule();
            EV << "MOVING TOWARDS NODE: " << neighbour->getFullName() << endl;
            cMessage* srvcMove = new cMessage("migrating..", SERVICE_MIGRATION);
            send(srvcMove, maxGateId + 1); // +1 because we are keeping the gateIDs of the in gates, +1 gives
            // Gate IDs are input vector /output vector contiguous in vector gates..
            // * and fully contiguous between input/output gates in scalar (non-vector) gates.
            // *
            // * Network.node[0]: parentIi: id=0, scalar gate, type:input
            // * Network.node[0]: parentIo: id=1, scalar gate, type:output
            // * Network.node[0]: leftIi: id=2, scalar gate, type:input
            // * Network.node[0]: leftIo: id=3, scalar gate, type:output
            // *

```

```

    * To adapt this code for general random topologies or any topology which requires a vector ino
    * considerable changes need to be make in the way you calculate the corresponding output gate t
    */

```

```

// -- DEACTIVATING SERVICE NODE --
serviceActive = false;
totalTrafficCount = 0; // resetting statistics gathering
cDisplayString& dispStr = getDisplayString();
dispStr.setTagArg("i", 0, "misc/node_vs");
trafficStats.clear(); // clearing the stat tracking vector
// -- DEACTIVATING SERVICE NODE --

```

```

EV << "Restarting traffic from this node" << endl;
scheduleAt(simTime()+5.0, SrvDemandMsg);

```

```

}

```

```

}

```

```

else EV << "ERROR: Service already deactivated.\n";

```

```

delete msg;

```

```

}

```

```

else if ( msg->getKind() == SERVICE_MIGRATION ){
    bubble("MIGRATION COMPLETE!!!");

```

```

// -- ACTIVATING SERVICE NODE --

```

```

if ( !serviceActive ) {
    serviceActive = true;
    cDisplayString& dispStr = getDisplayString();
    dispStr.setTagArg("i", 0, "misc/town2");
} else

```

```

{
    EV << "ERROR: Service already active.\n";
}

```

```

// -- ACTIVATING SERVICE NODE --

```

```

// Canceling traffic generation of this node

```

```

// again relocate after 14 seconds

```

```

scheduleAt(simTime()+15.3, new cMessage("relocation self event", RELOCATE));

```

```

delete msg;

```

```

dynamicNodeCreate(2);

```

```

}

```

```

}

```

```

Targ_msg* Node::generateTrafficMessage(int mode, int source) // Custom Function
{

```

```

    // Generate SOURCE ADDRESS

```

```

    int src;

```

```

    if ( source == -1){

```

```

    // no source was provided, using current node as source
    src = getIndex();
}
else{
    // source was provided
    src = source;
}

// Generate DESTINATION ADDRESS
int dest = 99999;
if ( mode == 0 ){ // RANDOM DESTINATION
    int n = 62;
    do {
        dest = intuniform(0, n);
    } while(dest == src);
}
else if ( mode == 1){ // TO THE ACTIVE SERVICE NODE
    // generating traffic message towards the currently active service node
    int numNodes = getParentModule()->getSubmodule("node", 0)->getVectorSize();
    numNodes += The_Overseer->getCreatedNodes();
    for(int i = 0; i < numNodes; i++){
        cModule *tempModule = getParentModule()->getSubmodule("node", i); // TODO possible problem with dyna
        if (tempModule == nullptr) {
            // get submodule returns nullptr when we go above the vector size of the original network
            std::string str;
            str = "node["+ std::to_string(i) +"]";
            const char* c = str.c_str();
            tempModule = getParentModule()->getSubmodule(c);
        }
        Node* target = check_and_cast<Node *>(tempModule);
        //EV << "CHECKING SUBMODULE Node " << target->getFullPath() << "w/ index: " << target->getIndex() <<
        if (target->serviceActive){
            dest = target->getIndex();
            //break; // break since we found the active service node
        }
    }
}

char msgname[100]; sprintf(msgname, "route n[%d]->n[%d]", src, dest);

// Create message object and set source and destination field.
Targ_msg *msg = new Targ_msg(msgname, TRAFFIC_MSG);
msg->setSource(src);
msg->setDestination(dest);
return (msg);
}

void Node::dynamicNodeCreate(int numToCreate){ // Custom Function

    // Find the factory object (created by Define_Module and taken from the NED type)
    cModuleType *moduleType = cModuleType::get("Node");

    int levels = int(getParentModule()->par("levels"));

```

```

int originalNumberOfNodes = int(pow(2, levels)) -1; // -2 gives us the index of the last node (since root =

int createdSoFar = The_Overseer->getCreatedNodes();
for (int i = originalNumberOfNodes+createdSoFar; i < (createdSoFar + originalNumberOfNodes + numToCreate); i
The_Overseer->incrementCreatedNodes(); // counter if we want to create more later...
// Figure out new node's index, and name
EV << "levels: " << levels << endl;
EV << "OriginalNumberOfNodes: " << originalNumberOfNodes << endl;

// creating name string (because create function requires const char * and not a string)
std::string str;
str = "node["+ std::to_string(i) +"]";
const char* c = str.c_str();
// Create module (node)
cModule *module = moduleType->create(c, getParentModule()); // getParentModule returns the top level "Ne

// Set up parameters and gate sizes before we set up its submodules
module->par("service") = false; // don't need this because of =default(false) (I THINK)
module->finalizeParameters();

// Setting up gates and connections
// find channel factory object and create a channel
cChannelType *channelType = cChannelType::get("Network.Channel");
cChannel *channel = nullptr;

// create connections
cModule *tempModule = getParentModule()->getSubmodule("node", int(floor((i-1)/2)));
if ( int(i%2) == 0 ){// connect left
    EV << "CONNECTING LEFT! \n";
    channel = channelType->create("Channel");
    module->gate("parent$o")->connectTo(tempModule->gate("left$i"), channel);
    channel = channelType->create("Channel");
    // module->gate("parent$i")->connectTo(tempModule->gate("left$o"), channel); DOESNT WORK
    tempModule->gate("left$o")->connectTo(module->gate("parent$i"), channel);
}
else if ( int(i%2) == 1 ){// connect right
    EV << "CONNECTING RIGHT! \n";
    channel = channelType->create("Channel");
    module->gate("parent$o")->connectTo(tempModule->gate("right$i"), channel);
    cChannel *channel = channelType->create("Channel");
    tempModule->gate("right$o")->connectTo(module->gate("parent$i"), channel);
}

// Create internals, and schedule it
module->buildInside();
module->callInitialize(); // call initializer of the module
module->scheduleStart(simTime());
}
}

```

14 OVERSEER.CC

```
#include <omnetpp.h>

using namespace omnetpp;

class OVERSEER : public cSimpleModule
{
public:
    OVERSEER();
    virtual ~OVERSEER();

    int getCreatedNodes();           // returns the number of dynamically created nodes so far
    void setCreatedNodes(int);       // sets number of created nodes
    void addCreatedNodes(int);       // adds a specific number of created nodes
    void subtractCreatedNodes(int);  // subtracts a specific number of created nodes
    void incrementCreatedNodes();

protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);

private:
    int numNodes;
    int createdNodes;
    cTopology overseer_topo;
}; Define_Module(OVERSEER);

void OVERSEER::initialize()
{
    createdNodes = 0; WATCH(createdNodes);
    /*
    overseer_topo.extractByParameter("topo_incl");
    numNodes = int(topo.getNumNodes()); WATCH(numNodes);

    for (int i = 0; i < numNodes; i++) {
        cTopology::Node *node = overseer_topo.getNode(i);

        EV << "Node i=" << i << " is " << node->getModule()->getFullPath() << endl;
        EV << " It has " << node->getNumOutLinks() << " conns to other nodes\n";
        EV << " and " << node->getNumInLinks() << " conns from other nodes\n";
        EV << " Connections to other modules are:\n";

        for (int j = 0; j < node->getNumOutLinks(); j++) {
            cTopology::Node *neighbour = node->getLinkOut(j)->getRemoteNode();
            cGate *gate = node->getLinkOut(j)->getLocalGate();
            EV << " " << neighbour->getModule()->getFullPath()
                << " through gate " << gate->getFullName() << endl;
        }
    }
    */
}
```

```

}

void OVERSEER::handleMessage(cMessage *msg)
{
    delete msg;
}

OVERSEER::OVERSEER()
{

}

OVERSEER::~OVERSEER()
{

}

int OVERSEER::getCreatedNodes(){
    /**
     * Returns the number of nodes that were created dynamically since the start of the current simulation.
     */
    if (createdNodes >= 0){
        return (createdNodes);
    }
    else
        throw cRuntimeError("Major Error! Created nodes counter is negative!");
}

void OVERSEER::setCreatedNodes(int value){
    /**
     * Sets the created nodes counter to the given non-negative value.
     */
    if ( value > 0){
        createdNodes = value;
    }
    else
        throw cRuntimeError("Error! Cannot set created nodes counter to a negative value!", value);
}

void OVERSEER::addCreatedNodes(int value){
    /**
     * Sums the given integer value to the created nodes counter.
     */
    if(value > 0){
        createdNodes += value;
    }
    else
        throw cRuntimeError("Invalid value %d. Must be a positive integer, use subtractCreatedNodes for subtract");
}

void OVERSEER::subtractCreatedNodes(int value){

```

```

/**
 * Subtracts the given integer value from the created nodes counter.
 */
if(value < 0){
    if(createdNodes - value > 0){
        createdNodes -= value;
    }
    else
        throw cRuntimeError("Error! Subtracting %d from createdNodes puts it below zero!", value);
}
else
    throw cRuntimeError("Invalid value %d. Must be a negative integer, use addCreatedNodes for subtraction o
}

void OVERSEER::incrementCreatedNodes(){
    createdNodes++;
}

```

15 OVERSEER.h

```

/*
 * OVERSEER.h
 *
 * Created on: Aug 30, 2018
 * Author: Constantine
 */

#ifndef OVERSEER_H_
#define OVERSEER_H_

#include <omnetpp.h>

using namespace omnetpp;

class OVERSEER : public cSimpleModule
{
public:
    OVERSEER();
    virtual ~OVERSEER();

    int getCreatedNodes();           // returns the number of dynamically created nodes so far
    void setCreatedNodes(int);       // sets number of created nodes
    void addCreatedNodes(int);       // adds a specific number of created nodes
    void subtractCreatedNodes(int);  // subtracts a specific number of created nodes
    void incrementCreatedNodes();

protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);

private:
    int numNodes;

```

```

    int createdNodes;
    cTopology overseer_topo;

}; // Define_Module(OVERSEER);

#endif /* OVERSEER_H_ */

```

16 Network.ned

```

import ned.DelayChannel;

network Network
{
    parameters:
        int levels = default(4);
    types:
        channel Channel extends ned.DelayChannel{
            delay = 100ms;
            //double weight;
            //@display("ls=white,2;t=weight");
        }
    submodules:
        node[int(2^levels-1)]: Node;
        The_Overseer: OVERSEER;

    connections allowunconnected:
        for i=1..sizeof(node)-1 {
            node[i].parent <--> Channel <--> node[int(floor((i-1)/2))].left if int(i%2)==0;
            node[i].parent <--> Channel <--> node[int(floor((i-1)/2))].right if int(i%2)==1;
        }
}

simple Node {
    parameters:
        bool service = default(false);
        @topoIncl("yes");
        @display("i=misc/node_vs");
    gates:
        inout parent;
        inout left;
        inout right;
}

simple OVERSEER
{
    @display("p=1000,0;i=abstract/penguin");
    gates:
        input in @directIn;
}

```

17 omnetpp.ini

```
[General]
network = Network
seed-set = 1112
Network.levels = 4
**.service = (index == 2^(levels)-2 ) ? true : false
```

18 targmsg.msg

```
message Targ_msg {
    int source;
    int destination;
    int hopCount = 0;
}
```

19 Python Notebook

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

def parse_if_number(s):
    ''' Returns float64 pandas value from string,
        else returns boolean values if string is a
        representation of a boolean value. '''
    try: return float(s)
    except: return True if s=="true" else False if s=="false" else s if s else None

def parse_ndarray(s):
    ''' Returns numpy array into the pandas dataframe
        from space seperated values in strings. '''
    return np.fromstring(s, sep=' ') if s else None

def mycsvread(s):
    ''' Custom pandas .csv reading function, uses the above two parsers '''
    tmp = pd.read_csv(s, converters = {
        'attrvalue': parse_if_number,
        'binedges': parse_ndarray,
        'binvalues': parse_ndarray,
        'vectime': parse_ndarray,
        'vecvalue': parse_ndarray})

    return tmp

# V E C T O R S
# Vectors have type 'vector'
# Reading all the scalar and vector values into two pandas dataframes
vec = mycsvread('vec0Results.csv')
vectors = vec[vec['type'] == 'vector']

#vectors[['vectime', 'vecvalue']]
for index, vector in vectors.iterrows():
```

```

plt.plot(vector['vectime'], vector['vecvalue'])
plt.title('Absolute cost paths in a 25x25 Grid Topology with 100 different seeds')
#plt.plot(vectors.iloc[3]['vectime'], vectors.iloc[3]['vecvalue'])

# find the biggest 'trail' of value
vectors['vecvalue'].map(len).max()

avg = [0] * vectors['vecvalue'].map(len).max() # creating a list of zeroes the size of the biggest vector
nval = [0] * vectors['vecvalue'].map(len).max()

for index, vector in vectors.iterrows():
    i = 0
    for value in vector['vecvalue']:
        avg[i] += value
        nval[i] += 1
        i += 1
#print(avg)
#print(nval)

for i in range(0, len(avg)):
    avg[i] = avg[i] / nval[i]

#print(avg)

plt.style.use('ggplot')
plt.title('Average absolute cost path in a 25x25 Grid Topology (Node 0)')
plt.plot(range(0, 50*len(avg), 50), avg, marker='o', markerfacecolor='white', markeredgecolor='red', linestyle='solid')

# check for errors in the vectors (should always be monotonically descending)
count = 0
for vector in vectors['vecvalue']:
    prev = 10000000000
    for i in vector:
        if (i > prev):
            count += 1
            print('error: ' + str(int(i)) + '>' + str(int(prev)))
        prev = i

print('errors:', count)

#S C A L A R S
sca = mycsvread('batteryScalars2.csv')

# Keeping only the rows that either have a scalar value, or are .ini iteration values (eg. seed, rc)
sca = sca[(sca['type'] == 'scalar') | (sca['type'] == 'itervar')]

# Scalar names (eg. Hops, % Cost Reduction) are in 'module' and 'name' columns. Names of attribute values are in 'attrname'
# Since attrname is not filled in for Scalar rows, we use attrname.combine_first(module+name) to fill in the attrname
sca = sca.assign(qname = sca['attrname'].combine_first(sca['module'] + '.' + sca['name']))

# Value column is empty for itervars, respectively, attrvalue is empty for scalars. Again merging the two into value
sca['value'] = sca['value'].combine_first(sca.attrvalue.astype('float64'))

# Deleting the attrvalue since we merged it into value

```

```

del sca['attrname']
del sca['attrvalue']
del sca['module']
del sca['name']

sca['qname'].unique()

# Pivot the above table so each run and every value produced by it is one row
sca_wide = sca.pivot('run', columns='qname', values='value')
sca_wide.head()
#sca_wide[['rc', 'seed', 'RandomGraph.The_Overseer.Percent Cost Reduction']].head(25)

sca_wide.plot.scatter('RandomGraph.The_Overseer.Weighted (Cost) Distance to Median', 'RandomGraph.The_Overseer.N
plt.title('500 Node Random Geometric Topology (distance weighted)')
plt.ylabel('Mean Battery Level')
plt.xlabel('Cost distance to 1-median')
plt.style.use('ggplot')
plt.show()

sca_wide.plot.scatter('RandomGraph.The_Overseer.Weighted (Cost) Distance to Median', 'RandomGraph.The_Overseer.B
plt.ylabel('Battery Levels Variance')
plt.xlabel('Cost distance to 1-median')
#plt.yscale("log")
#plt.ylim([0, 1300000])
plt.style.use('ggplot')
plt.show()

sca_wide.plot.scatter('RandomGraph.The_Overseer.Weighted (Cost) Distance to Median', 'RandomGraph.The_Overseer.T
plt.ylabel('Time of Network Death')
plt.xlabel('Cost distance to 1-median')
plt.ylim([4020,4080])
plt.style.use('ggplot')
plt.show()

# Normalizing data
# (df-df.min())/(df.max()-df.min())
s1 = 'RandomGraph.The_Overseer.Network battery average'
sca_wide[s1] = (sca_wide[s1]-sca_wide[s1].min())/(sca_wide[s1].max() - sca_wide[s1].min())
s2 = 'RandomGraph.The_Overseer.Battery level variance'
sca_wide[s2] = (sca_wide[s2]-sca_wide[s2].min())/(sca_wide[s2].max() - sca_wide[s2].min())

# NORMALIZED MEAN BATTERY, BATTERY VARRIANCE / DISTANCE SCATTER
ax1 = plt.plot()
plt.scatter(sca_wide['RandomGraph.The_Overseer.Weighted (Cost) Distance to Median'], sca_wide['RandomGraph.The_0
plt.scatter(sca_wide['RandomGraph.The_Overseer.Weighted (Cost) Distance to Median'], sca_wide['RandomGraph.The_0
plt.title('Normalized Battery Variance vs Mean Battery')
plt.ylabel('Mean Battery Level')
plt.xlabel('Cost distance to 1-median')
plt.ylim([-0.1,1.1])

ax2 = plt.gca().twinx()
plt.ylabel('Battery Variance')
plt.ylim([-0.1,1.1])

```

```

plt.style.use('ggplot')
plt.show()

temppivot = sca_wide.pivot_table(index='rc', columns='seed', values='RandomGraph.The_Overseer.Network battery av
temppivot.head()

temppivot.plot.line()
plt.show()

# Javascript Pivot Table Interface
import pivottablejs as pj
pj.pivot_ui(sca)

grid35[grid35['name'] == 'Percent Cost Reduction'][['name', 'value']].head(10)
grid0[grid0['name'] == 'Percent Cost Reduction']['value'].describe()
grid0[grid0['name'] == 'Hops']['value'].describe()
#grid[grid['name'] == 'Hops']
grid35[grid35['name'] == 'Hops']['value'].describe()
scalars = grid0[grid0['name'] == 'Percent Cost Reduction']['value']
scalars2 = grid35[grid35['name'] == 'Percent Cost Reduction']['value']

```