ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΑΓΡΟΝΟΜΩΝ & ΤΟΠΟΓΡΑΦΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΤΟΠΟΓΡΑΦΙΑΣ

Διπλωματική Εργασία

Προσαρμογή Λογισμικών Προσομοίωσης Δικτύων Για Την Προσομοίωση Ασυρμάτων Δικτύων Αισθητήρων (WSNs) Υψηλής Χωρικής Πυκνότητας

Επιβλέπων Καθηγητής: Βεσκούκης Βασίλειος

Επιτροπή Καθηγητών: Βεσκούκης Βασίλειος
Κορακίτης Ρωμύλος
Τσακίρης Γεώργιος

Βούτος Διονύσης
Μάρτιος 2010

αφιερώνεται στους γονείς μου,
Ανδρέα και Ελένη

Θα ήθελα να εκφράζω τις θερμές μου ευχαριστίες στον επιβλέποντα την εργασία μου Επίκουρο Καθηγητή κ. Βασίλειο Βεσκούκη, τόσο για την πολύτιμη βοήθεια και την καθοδήγηση πάνω στην εργασία, όσο και για την καταρχήν δυνατότητα που μου έδωσε να εργαστώ σε ένα τέτοιο αντικείμενο. Δεν είναι άλλωστε η πρώτη φορά.

Επίσης να ευχαριστήσω την υποψήφια διδάκτορα Κολεγά Εύα, για τη βοήθεια και τις καίριες συμβουλές της πάνω στο θέμα.

Πρόλογος

Η διπλωματική αυτή εργασία αποτελεί μια μικρή μελέτη πάνω στην προσαρμογή λογισμικών προσομοίωσης δικτύων, για την προσομοίωση ασυρμάτων δικτύων αισθητήρων υψηλής χωρικής πυκνότητας. Εργαλείο ανάπτυξης είναι η C++ γλώσσα προγραμματισμού, στην οποία βασίζονται τα λογισμικά που επεξεργάζεται η εργασία. Το αντικείμενο είναι πολύ ενδιαφέρον και επίκαιρο, ειδικά με την ολοένα αυξανόμενη περιβαλλοντική ανησυχία που προκαλούν οι φθορές των ανθρωπίνων κατασκευασμάτων στη φύση. Γίνεται προσπάθεια να αναλυθεί, στο στενό αυτό πλαίσιο που εξετάζεται, όσο καλύτερα γίνεται, ώστε να αποτελέσει κίνητρο για μια ακόμα μεγαλύτερης έκτασης εργασία.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΜΕΡΟΣ 2ο: ΜΕΤΑΔΟΣΗ ΡΑΔΙΟΣΥΧΝΟΤΗΤΩΝ

ΜΕΡΟΣ 3ο: ΥΛΟΠΟΙΗΣΗ

ΠΙΝΑΚΑΣ ΣΧΗΜΑΤΩΝ

ΠΙΝΑΚΑΣ ΠΙΝΑΚΩΝ

Περίληψη

Η παρούσα διπλωματική εργασία εξετάζει την προσομοίωση ασύρματων δικτύων αισθητήρων, μέσω της τροποποίησης και του προγραμματισμού κατάλληλου λογισμικού.Τρεις προσομοιωτές δικτύων έχουν ληφθεί υπόψη: το Shawn, το Omnet++ και το Castalia. Και οι τρεις βασίζονται στη γλώσσα προγραμματισμού C++. Οι δύο πρώτοι δεν αποτελούν προσομοιωτές με τη στενή έννοια του όρου, αλλά περισσότερο βιβλιοθήκες εργαλείων και υποδομών (C++ τάξεις), για να προγραμματίζονται προσομοιωτές. Επιπλέον, το Castalia είναι ένας προσομοιωτής του Omnet++.

Η τροποποίηση του λογισμικού γίνεται με γνώμονα τον προγραμματισμό προσομοιωτών ασυρμάτων δικτύων αισθητήρων μεγάλης χωρικής πυκνότητας. Σε ένα τέτοιο δίκτυο δεχόμαστε δύο ειδών κόμβους: αισθητήρες (sensors) και πύλες (gateways). Οι αισθητήρες είναι συσκευές εκπομπής σήματος και οι πύλες αποδέκτες αυτής της εκπομπής. Επιπλέον, η περιοχή ενδιαφέροντος περιορίζεται σε ανοιχτό χώρο υπαίθρου με διάφορες συνθήκες βλάστησης.

Η εργασία αποτελείται από τέσσερα μέρη:

Στο πρώτο μέρος αναλύεται το θεωρητικό υπόβαθρο της εργασίας, αναπτύσσοντας παράλληλα τις δομές των τριών προσομοιωτών ξεχωριστά. Παρουσιάζεται επίσης η πολιτική τους στη συγκρότηση και εκτέλεση προσομοιώσεων, και παράλληλα καταγράφονται συνοπτικά οι ελλείψεις του καθενός στα πλαίσια των περιοχών ενδιαφέροντος που εξετάζουμε.

Στο δεύτερο μέρος γίνεται ανάλυση της μετάδοσης ραδιοσυχνοτήτων και μοντέλων μεγάλης κλίμακας για τον υπολογισμό της εξασθένισης της ισχύος σήματος, ώστε να εξεταστεί υπό το πρίσμα αυτό η συνδεσιμότητα των κόμβων. Αυτή η ανάλυση αποτελεί το θεωρητικό υπόβαθρο για την ανάπτυξη των αλγορίθμων και των σχέσεων πάνω στις οποίες κάθε προσομοιωτής
θα εξετάζει την επικοινωνία.

Στο τρίτο μέρος παρουσιάζεται η υλοποίηση που πραγματοποιήθηκε για κάθε λογισμικό προσομοίωσης. Αυτή έγινε με σεβασμό στην πολιτική και αρχιτεκτονική που προωθεί ο καθένας από αυτούς. Κύριος άξονας ανάπτυξης κάθε εφαρμογής ήταν η ενσωμάτωση των μοντέλων εξασθένισης ισχύος σήματος στις δομές εξέτασης της επικοινωνίας, και ο διαχωρισμός του ρόλου πομπού και δέκτη για τους αισθητήρες και τις πύλες.

Στο τέταρτο μέρος εκτελείται σενάριο υπολογισμού ισχύος για θέσεις κόμβων σε δύο δίκτυα αισθητήρων. Για τα δίκτυα αυτά έχουμε μετρήσεις ισχύος από ανεξάρτητη εργασία τις οποίες και συγκρίνουμε με τα αποτελέσματα των προσομοιωτών. Στη δοκιμή αυτή παρουσιάζεται χαρακτηριστικά και η ροή μιας εργασίας από την προετοιμασία των δεδομένων μέχρι την τελική παρουσίασή τους.

Τέλος, η εργασία κλείνει με την κριτική των τριών προσομοιωτών, τόσο ανεξάρτητα όσο και σε σύγκριση μεταξύ τους. Δίνονται τα τελικά συμπεράσματα της εκπόνησης της εργασίας με τις εμπειρίες και τα προβλήματα που παρουσιάστηκαν. Αναφέρεται μια προοπτική ανάπτυξης και βελτίωσης των συγκεκριμένων υλοποιήσεων όπως και μια γενικότερη θεώρηση του θέματος .

Abstract

The present diploma thesis examines Wireless Sensor Networks (WSNs) Simulation through modifying and programming simulation software. Three simulators are examined: Shawn, Castalia and Omnet++. All of them are programmed in C++. Shawn and mostly Omnet++ are not considered strictly simulators; rather, simulation frameworks for building simulators. Castalia was built using the Omnet ++ framework, so that makes it an Omnet++ simulator.

Source code modification and coding in general are done under the concept of high density WSNs implementation. In such a network two kinds of nodes are accepted: sensors and gateways. Sensors are devices transmitting radio signals and gateways are the receivers of this transmission. These nodes reside in an open area with various types of vegetation.

This diploma thesis consists of four parts:

Part One analyses the theoretical framework while exploring separately the simulators' structure. Their different policies concerning the formation and performance of simulations are presented, while their inadequacies are stated, based on this thesis's needs.

Part Two supplies a short explanation of the RF Propagation phenomenon and presents a variety of large-scale signal attenuation models, which will be used to calculate node connectivity. This analysis results to the theoretical framework for algorithm and function development. These algorithms are compiled in the communication model of each simulator.

Part Three presents a custom implementation for each simulator, with respect to their particular architectures and policies. The programming techniques used and generated code are presented in detail. The major goal of each simulator was the integration of the signal loss models in the simulation process and the distinction between sensors and gateways over the transmitter/receiver functionality.

Part Four presents a simulation scenario for two WSNs. Each WSN consists of a number of sensors and a gateway. The received signal strength at the gateway of each network for is calculated for all sensors, using the three simulators we built. Independent measurements of signal strength are provided for the sensors of these networks, which we compare with the calculated data of the simulators. The process of simulation is presented from preparation to the final presentation of results.

The simulators are assessed both independently and in relation to one another. The final conclusions of the procedure are given as well as the experiences and the problems that were faced. Ways the matter could be explored further are also mentioned, followed by an analysis of some thesis's issues in a broader perspective.

Εισαγωγή

Τα ασύρματα δίκτυα αισθητήρων είναι μια εφαρμογή με αυξανόμενη ζήτηση και συνεχή διεύρυνση του πεδίου εφαρμογής της. Τα δίκτυα αυτά αποτελούνται από πολλούς κόμβους που συνδέονται μεταξύ τους με ασύρματη τεχνολογία μεταφοράς δεδομένων. Οι κόμβοι αυτοί διαθέτουν συσκευή αισθητήρα η οποία παρακολουθεί τη διακύμανση μιας φυσικής ποσότητας. Όταν η τιμή αυτής περάσει κάποιο κρίσιμο επίπεδο οι κόμβοι ενημερώνουν μια κεντρική βάση για το γεγονός, και από εκεί δρομολογείται η οποιαδήποτε περαιτέρω ενέργεια.

Η ανάπτυξη των ασυρμάτων δικτύων αισθητήρων έχει τις ρίζες τις σε στρατιωτικές έρευνες και εφαρμογές σε παρακολούθηση πεδίου μάχης. Σήμερα ωστόσο χρησιμοποιούνται ευρέως σε εφαρμογές ελέγχου του περιβάλλοντος και προστασίας της φύσης και του ανθρώπου. Ιδιαίτερα διαδεδομένες είναι οι εφαρμογές πυρασφάλειας, ελέγχου του επιπέδου των υδάτων και της μόλυνσής τους, συλλογής ατμοσφαιρικών πληροφοριών (υγρασία, πίεση) και παρακολούθησης άγριων ζώων σε προστατευόμενες περιοχές. Πληθώρα άλλων εφαρμογών μπορούμε να συναντήσουμε σε αστικό περιβάλλον και στη βιομηχανία.

Η υλοποίηση ενός ασύρματου δικτύου αισθητήρων, είναι μια δύσκολη διαδικασία. Αυτό οφείλεται στην φύση της ασύρματης μετάδοσης σήματος, η οποία είναι επιρρεπής σε πολλούς παράγοντες και δεν εξασφαλίζεται τόσο εύκολα όσο η καλωδιακή σύνδεση. Η διάταξη του περιβάλλοντος χώρου, η απόσταση των κόμβων, η ορατότητα μεταξύ τους, οι ατμοσφαιρικές κατακρημνίσεις άλλες ηλεκτρομαγνητικές παρεμβολές και πάρα πολλοί άλλοι παράγοντες καθορίζουν την ποιότητα της σύνδεσης και την επίτευξη επικοινωνίας σε ένα ασύρματο δίκτυο. Είναι απαραίτητο πριν την εγκατάσταση ενός δικτύου, να εξασφαλισθεί σε ένα βαθμό ότι αυτή θα είναι πετυχημένη και λειτουργική, αλλιώς υπάρχει ο κίνδυνος να ξοδευτούν χρόνος και χρήμα χωρίς αντίκρυσμα.

Για το σκοπό αυτό υπάρχουν δύο επιλογές. Η πρώτη είναι η δοκιμή, το πείραμα. Ορίζουμε ένα δίκτυο με ένα πρόχειρο υπολογισμό και αφού το στήσουμε ελέγχουμε τη λειτουργικότητά του. Αν δεν είναι ικανοποιητική προσπαθούμε να εντοπίσουμε το πρόβλημα και με νέα δοκιμή ελέγχουμε την τροποποίηση που πραγματοποιήσαμε. Η τακτική αυτή μπορεί ενδεχομένως να αποδώσει σε ένα απλό δίκτυο μικρής έκτασης και λίγων κόμβων. Σίγουρα όμως δεν θα μπορούσε με κανένα τρόπο να εφαρμοστεί σε δίκτυα υψηλής χωρικής πυκνότητας, δηλαδή δίκτυα με πάρα πολλούς κόμβους αναλογικά με την έκταση που καλύπτουν. Αν υπολογίσει κανείς επιπλέον και τα δίκτυα κινούμενων κόμβων, τότε γίνεται σαφές πως η επιλογή της δοκιμής είναι πολύ περιορισμένη.

Η δεύτερη δυνατότητα που μας δίνεται για το σχεδιασμό ενός ασύρματου δικτύου αισθητήρων είναι η προσομοίωση. Προσομοίωση είναι η προσπάθεια μίμησης μιας πραγματικής κατάστασης, αντικειμένου ή φαινομένου, όσο πιο πιστά γίνεται. Μια προσομοίωση αξιολογείται από το πλήθος, τη λεπτομέρεια και την πιστότητα των χαρακτηριστικών και των λειτουργιών που αναπαριστά. Οι μορφές προσομοίωσης είναι πάρα πολλές. Για υπολογιστικές φαινόμενα τα οποία αναπαριστώνται με μαθηματικά μοντέλα εφαρμόζουμε συνήθως προσομοιώσεις σε ηλεκτρονικό υπολογιστή ή κάποιο άλλο υπολογιστικό σύστημα.

Για τα ασύρματα δίκτυα αισθητήρων το κύριο ενδιαφέρον μας είναι να αναπαραστήσουμε την επικοινωνία ανάμεσα στους κόμβους. Εφόσον αυτή βασίζεται στη μετάδοση ραδιοσυχνοτήτων, το αντικείμενο μεταφέρεται σε αυτή. Χρειάζεται επομένως να υλοποιηθούν μοντέλα επικοινωνίας τα οποία να αναπαριστούν όσο πιο πιστά γίνεται αυτή τη μετάδοση και να εκτιμούν με ακρίβεια την ποιότητα των συνδέσεων.

Πάνω σε αυτό το θέμα έχουν υλοποιηθεί αρκετοί προσομοιωτές ασυρμάτων δικτύων. Για την μελέτη που ακολουθεί επιλέχθηκαν να χρησιμοποιηθούν τρεις προσομοιωτές, το Shawn, το Castalia και το Omnet++. Το Shawn και το Castalia χαρακτηρίζονται ως προσομοιωτές ασυρμάτων δικτύων αισθητήρων, γεγονός που τους κατέστησε αντικείμενα της μελέτης αυτής έναντι άλλων προσομοιωτών οι οποίοι δεν διαφημίζουν κάποιο τέτοιο προσανατολισμό. Η χρήση τους είναι αρκετά διαδεδομένη στην κοινότητα ανάπτυξης εφαρμογών προσομοίωσης δικτύων αισθητήρων, δίνοντας μια αίσθηση επιβεβαίωσης και ασφάλειας ότι θα εξυπηρετήσουν το σκοπό τους. Το Omnet++ είναι μια συλλογή βιβλιοθηκών και άλλων εργαλείων για την ανάπτυξη δικτύων δεδομένων. Δεν έχει κάποιο προσανατολισμό προς τα ασύρματα δίκτυα. Κάθε άλλο η αρχιτεκτονική και η πολιτική του είναι στα πρότυπα της ενσύρματης δικτύωσης. Επειδή όμως το Castalia είναι ανεπτυγμένο βάσει του Omnet++, δίνεται το κίνητρο της διερεύνησης των δυνατοτήτων στο συγκεκριμένο τύπο επικοινωνίας.

Οι τρεις αυτοί προσομοιωτές που χρησιμοποιούνται στην εργασία, βασίζουν τη λειτουργία τους στον προγραμματισμό του πηγαίου κώδικά τους. Δεν πρόκειται δηλαδή για προγράμματα καθοδηγούμενα από ρυθμίσεις και οδηγίες σε κάποιο διαδραστικό περιβάλλον χρήστη, αλλά για συλλογές βιβλιοθηκών και πηγαίου κώδικα σε C++ γλώσσα προγραμματισμού. Ο μελετητής πρέπει να συνδυάσει αυτές τις δομές, να τις τροποποιήσει και να προσθέσει νέες, ώστε τελικά να συγκροτήσει έναν προσομοιωτή που ανταποκρίνεται στις ανάγκες του. Ο προγραμματισμός είναι αναπόφευκτος στο συγκεκριμένο χώρο εφαρμογών. Είναι πάρα πολλά τα μοντέλα και οι αλγόριθμοι που μπορούν να αξιοποιηθούν, και ακόμα περισσότεροι οι πιθανοί συνδυασμοί και οι βελτιώσεις που τυχόν θα προκύψουν. Δεν είναι άλλωστε τυχαίο ότι έχουν δημιουργηθεί πάρα πολλές γλώσσες προγραμματισμού ειδικά για ανάπτυξη προσομοιωτών. Η ίδια η C++, παρότι γλώσσα προγραμματισμού γενικότερου ενδιαφέροντος, βασίζει πολλές δομές της στην Simula γλώσσα προγραμματισμού προσομοιωτών.

Στόχος της εργασίας είναι να προσαρμόσουμε τα λογισμικά προσομοίωσης Shawn, Castalia και Omnet++, ώστε να εξετάζουν επαρκώς την επικοινωνία σε ασύρματα δίκτυα αισθητήρων υψηλής χωρικής πυκνότητας. Η προσαρμογή αυτή περιλαμβάνει τόσο τροποποίηση του αρχικού πηγαίου κώδικα, όσο και τον προγραμματισμό νέων C++ τάξεων και λειτουργιών. Ο νέος κώδικας πρέπει να εναρμονίζεται με την αρχιτεκτονική κάθε λογισμικού και να καλύπτει επαρκώς τις όποιες ελλείψεις για τη σωστή λειτουργία της προσομοίωσης. Οι προσομοιωτές πρέπει να μπορούν να εξετάζουν την μονόδρομη επικοινωνία κόμβου αισθητήρα με ένα κόμβο συλλογής δεδομένων (πύλη) και μόνο αυτή, με γνώμονα μοντέλα εξασθένισης της ισχύος σήματος.

Για το σκοπό αυτό το πρώτο βήμα είναι η μελέτη και ανάλυση των τριών πακέτων λογισμικού. Η μελέτη γίνεται σε επίπεδο πηγαίου κώδικα, ειδικά για τα Shawn και Castalia, δεδομένου ότι η τεκμηρίωση που υπάρχει για αυτά είναι ελάχιστη. Το Omnet++ αντίθετα ως πολύ παλαιότερο και ώριμο λογισμικό έχει πολύ καλή και επαρκή τεκμηρίωση και ως εκ τούτου η ανάπτυξη σε αυτό ήταν πιο εύκολη εξ' αρχής. Η μελέτη των προσομοιωτών συμπληρώνεται από μια ανάλυση των μοντέλων εξασθένισης της ισχύος, στα πρότυπα του τύπου περιοχών που εξετάζονται. Έτσι παρουσιάζονται μοντέλα εξασθένισης λόγω απόστασης ανοιχτού χώρου και αναγλύφου, καθώς και μοντέλα εξασθένισης βλάστησης.

Αφού εντοπίσουμε τις ελλείψεις κάθε πακέτου λογισμικού και συγκεντρώσουμε τα μοντέλα εξασθένισης που θα ενσωματώσουμε στους προσομοιωτές, προχωρούμε στον προγραμματισμό του κατάλληλου κώδικα που θα υλοποιεί τις λειτουργίες αυτών και τις υπόλοιπες δομές που χρειάζονται. Η ανάπτυξη γίνεται σε επίπεδο μοντέλου επικοινωνίας και κόμβων. Οι προσομοιωτές που δημιουργούνται παρουσιάζονται αναλυτικά. Τέλος εκτελείται δοκιμαστική προσομοίωση για δύο περιοχές και συγκρίνονται τα αποτελέσματα με πειραματικές μετρήσεις που προέρχονται από ανεξάρτητες παρατηρήσεις. Στο παράρτημα παρουσιάζεται το εικονικό μηχάνημα (Virtual Machine) στο οποίο εγκαταστάθηκε το λογισμικό και γενικές οδηγίες χρήσης του.

ΜΕΡΟΣ 1ο

ΑΝΑΛΥΣΗ

Shawn

ΑΝΑΛΥΣΗ:

Το Shawn είναι ένας προσομοιωτής ασυρμάτων δικτύων αισθητήρων του οποίου κύριος στόχος είναι η προσομοίωση των αποτελεσμάτων και όχι του ίδιου του δικτύου. Είναι προγραμματισμένο στη C++ γλώσσα προγραμματισμού και οι προσομοιώσεις κατασκευάζονται τροποποιώντας τον πηγαίο κώδικα του λογισμικού και δημιουργώντας νέες δομές.

1. Shawn: Ανάλυση Δομής

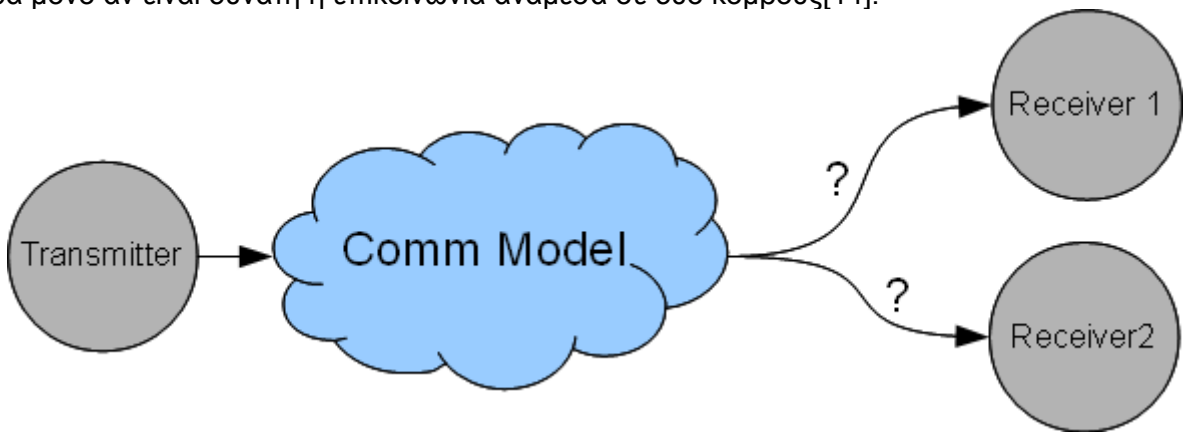
Το Shawn απαρτίζεται από πολλές διαφορετικές λειτουργικές οντότητες, οι οποίες συνεργάζονται για την εκτέλεση προσομοιώσεων. Μερικές εξυπηρετούν την αναπαράσταση φυσικών αντικειμένων, ενώ άλλες συγκροτούν τα φυσικά φαινόμενα στα οποία παίρνουν μέρος τα αντικείμενα αυτά, και εν προκειμένω την ασύρματη επικοινωνία, δηλαδή τη μετάδοση ραδιοκυμάτων. Και οι δύο τύποι είναι εξίσου σημαντικοί για την προσομοίωση.

Στο Shawn τα φυσικά αντικείμενα, οι θέσεις δικτύου, αναπαρίστανται με κόμβους (Nodes). Η λειτουργία αυτών συγκροτείται με επεξεργαστές (Processors) και η λειτουργία και οργάνωση του φυσικού κόσμου και του περιβάλλοντος προσομοίωσης με τα μοντέλα επικοινωνίας (Communication), ακμών (Edge) και εκπομπής (Transmission). Ειδικά τα μοντέλα αυτά είναι που εκφράζουν την πολιτική του Shawn στο θέμα προσομοίωση ασύρματων δικτύων, και είναι το κέντρο ενδιαφέροντος της ανάλυσης.

1.1 Μοντέλα (Models)

1.1.1 Μοντέλο Επικοινωνίας (Communication Model)

Καθορίζει αν επικοινωνούν κατά συνθήκη δύο κόμβοι. Όταν ένας κόμβος αποστέλλει ένα μήνυμα, οι εν δυνάμει παραλήπτες αυτού του μηνύματος αναγνωρίζονται από το μοντέλο επικοινωνίας. Δεν εξετάζει τις ιδιότητες της εκπομπής ενός μηνύματος (πράγμα που κάνει το μοντέλο εκπομπής), παρά μόνο αν είναι δυνατή η επικοινωνία ανάμεσα σε δύο κόμβους[14].



Σχήμα 1.1

Το Shawn διαθέτει αρκετές υλοποιήσεις για το μοντέλο επικοινωνίας:

1.1.1.1 disk_graph_model

Το μοντέλο αυτό, γνωστό ως 'Unit Disk Graph radio model' (UDG), βασίζεται στην παρατήρηση ότι η ισχύς του εκπεμπόμενου σήματος φθίνει ανάλογα με το τετράγωνο της απόστασης από την πηγή εκπομπής. Δεδομένης μιας ελάχιστης απαίτησης ισχύος σήματος για λήψη, δύο κόμβοι μπορούν να επικοινωνούν αν η ευκλείδεια απόσταση ανάμεσά τους είναι μικρότερη από *rmax*, όπου *rmax* είναι η μέγιστη ακτίνα λήψης του σήματος.

Το 'UDG' μπορεί να τροποποιηθεί ώστε να δίνει δύο κατώφλια εκτίμησης της λήψης του σήματος, ένα μέγιστο όπως πριν και ένα ελάχιστο. Εφόσον η απόσταση *r* ανάμεσα σε δύο κόμβους είναι μικρότερη του ελαχίστου (*0* < *r* < *rmin*) ή μεγαλύτερη του μεγίστου (*r* > *rmax*), τότε η συμπεριφορά του μοντέλου είναι η ίδια με του απλού 'UDG'. Αν όμως είναι *rmin* < *r* < *rmax* τότε εισάγεται μια πιθανότητα λήψης του πακέτου η οποία μειώνεται γραμμικά από 1 σε 0 όσο το *r* τείνει στο *rmax*, τιμώντας το γεγονός ότι η πιθανότητα μιας επιτυχημένης λήψης μειώνεται με την αύξηση της απόστασης. Το μοντέλο αυτό καλείται 'Quasi-Unit Disk Graph radio model' (Q-UDG), και στο Shawn υλοποιείται στο 'RIM' μοντέλο επικοινωνίας με αξιοποίηση μόνο των απαραιτήτων παραμέτρων.

1.1.1.2 link_probability_communication_model

Το μοντέλο επικοινωνίας αυτό παρέχει στατικές συνδέσεις ανάμεσα στους κόμβους προκειμένου να προσομοιώσει σταθερά κανάλια επικοινωνίας, εξετάζοντας όμως και μια πιθανότητα προκειμένου να μπορεί να υλοποιηθεί αυτή η επικοινωνία. Κατά την εκκίνηση το μοντέλο εξετάζει αν το σενάριό μας έχει ομαδική ετικέτα (*tag type="group"*) με όνομα "link_probability" στο περιβάλλον προσομοίωσης. Μια τέτοια ετικέτα μπορούμε να έχουμε αποθηκεύσει σε ένα .xml αρχείο ως εξής:

```
<scenario>
  <environment>
    <tag type="group" name="link_probability">
    .
    .
    .
    </tag>
  </environment>
</scenario>
```

Μέσα σ' αυτήν την ετικέτα ομάδας, το μοντέλο αναζητεί ένα StringDoubleMapTag, δηλαδή ετικέτα τύπου χάρτη (*tag type="map"*) που έχει δύο παραμέτρους, ένα string και ένα double. Το όνομα της ετικέτας χάρτη πρέπει να είναι το "node id" (δηλαδή το *Node::label()* και όχι το εσωτερικό node id που χρησιμοποιεί το Shawn) του κόμβου από τον οποίο θα ξεκινούν οι στατικές συνδέσεις. Προφανώς για περισσότερους κόμβους εισάγουμε επιπλέον ετικέτες χάρτη. Κάθε εγγραφή σε μια ετικέτα χάρτη αποτελείται από ένα string (*"entry index"*) που θα αναφέρεται στο "node id" κάθε κόμβου με τον οποίο θέλουμε να κάνουμε στατική σύνδεση, και ένα double (*"value"*) που εκπροσωπεί την πιθανότητα επικοινωνίας μέσα από τη σύνδεση αυτή. Αυτή συγκρίνεται με μια Ομοιόμορφη Κατανομή Τυχαίας Μεταβλητής (UniformRandomVariable), η οποία υπολογίζεται ξεχωριστά για κάθε σύνδεση. Δεν εξετάζεται κάποιο εύρος επικοινωνίας ή η απόσταση των κόμβων. Εφόσον οριστούν οι συνδέσεις η επικοινωνία είναι δεδομένη και εξαρτάται μόνο από την πιθανότητα που αντιστοιχεί σε κάθε σύνδεση. Η σύνταξη μιας ετικέτας χάρτη για το 'Link Probability' μοντέλο επικοινωνίας είναι:

```
<tag type="map-string-double" name="01">
     <entry index="02" value="1.0"/>
     <entry index="03" value="0.5"/>
     <entry index="04" value="0.8"/>
     .
    .
     .
     <entry index="99" value="0.7"/>
</tag>
.
.
.
<tag type="map-string-double" name="02">
     <entry index="01" value="1.0"/>
     .
     .
     .
     <entry index="10" value="0.6"/>
</tag>
```

Για να φορτώσουμε τις ετικέτες στο περιβάλλον της προσομοίωσης ώστε να τις εξετάσει το μοντέλο χρησιμοποιούμε στην εργασία *prepare_world* την παράμετρο *"env_config"* στην οποία υποδεικνύουμε το αρχείο XML όπου έχουμε αποθηκεύσει τις ετικέτες. Είναι επιπλέον δυνατό να τις αποθηκεύσουμε στο ίδιο XML αρχείο που πχ έχουμε αποθηκεύσει τις θέσεις των κόμβων, αρκεί να τηρείται η σωστή σύνταξη.

Η μέθοδος εξετάζει την επικοινωνία μόνο για τα τις ετικέτες που έχουμε δηλώσει, και αποκλειστικά για τις κατευθύνσεις που αυτές υλοποιούν. Όσοι κόμβοι είναι φορτωμένοι στον κόσμο αλλά δεν συμπεριλαμβάνονται στις ετικέτες δεν εξετάζονται από αυτό το μοντέλο. Είναι βέβαια δυνατό υλοποιώντας πολλαπλές επιλογές με το 'Multiple' μοντέλο επικοινωνίας να εφαρμόσουμε άλλο μοντέλο επικοινωνίας για αυτούς τους κόμβους. Για κάθε ετικέτα χάρτη εξετάζεται αν ο κόμβος που είναι στο όνομά της (*"name"*) επικοινωνεί με τους κόμβους που είναι στον κορμό της ετικέτας (*"index"*) για την αντίστοιχη πιθανότητα (*"value"*). Είναι δυνατή και η μελέτη αμφίδρομης επικοινωνίας (*can_commmunicate_bidi()*) για τους κόμβους, δηλαδή να ελεγχθεί και αν οι κόμβοι του ευρετηρίου της ετικέτας χάρτη (*"index"*) επικοινωνούν με τον κόμβο στον οποίο ανήκει η ετικέτα (*"name"*). Κάθε άλλη σχέση επικοινωνίας δεν μελετάται από το μοντέλο.

1.1.1.3 permalink_communication_model

Το μοντέλο αυτό επικοινωνίας παρέχει στατικές συνδέσεις ανάμεσα σε κόμβους προκειμένου να προσομοιώσει σταθερά κανάλια επικοινωνίας. Μοιάζει αρκετά με το 'Link Probability' μοντέλο, με τη διαφορά ότι εδώ δεν υπάρχει πιθανότητα αλλά έλεγχος αμφίδρομης/μονόδρομης επικοινωνίας. Κατά την εκκίνηση το μοντέλο εξετάζει αν το σενάριό μας έχει ετικέτα ομάδας "permalink" στο περιβάλλον προσομοίωσης. Μια τέτοια ετικέτα μπορούμε να έχουμε αποθηκεύσει σε ένα XMLl αρχείο ως εξής:

```
<scenario>
  <environment>
    <tag type="group" name="permalink">
    .
    .
    .
    </tag>
  </environment>
</scenario>
```

Μέσα στην ετικέτα αυτή, το μοντέλο αναζητεί ένα StringBoolMapTag, δηλαδή ετικέτα χάρτη που έχει δύο παραμέτρους, ένα string και ένα bool (true / fasle). Το όνομα της ετικέτας πρέπει να είναι το "node id" (δηλαδή το *Node::label()* και όχι το εσωτερικό node id που χρησιμοποιεί το Shawn) του κόμβου από τον οποίο θα ξεκινούν οι στατικές συνδέσεις. Προφανώς για περισσότερους κόμβους εισάγουμε επιπλέον ετικέτες. Κάθε εγγραφή της ετικέτας χάρτη αποτελείται από ένα string (*"entry index"*), που θα αναφέρεται στο "node id" κάθε κόμβου με τον οποίο θέλουμε να κάνουμε στατική σύνδεση, και ένα bool (*"value"*) που καθορίζει αν η επικοινωνία θα είναι αμφίδρομη ή μονόδρομη. Αν η τιμή είναι "true" τότε έχουμε αμφίδρομη επικοινωνία, ενώ αν είναι "false" έχουμε μονόδρομη. Δεν εξετάζεται κάποιο εύρος επικοινωνίας ή η απόσταση των κόμβων. Η σύνταξη ενός map tag για το permanent_link μοντέλο επικοινωνίας είναι:

```
<tag type="map-string-bool" name="01">
     <entry index="02" value="true"/>
     <entry index="03" value="true"/>
     <entry index="04" value="false"/>
     .
     .
     .
     <entry index="99" value="true"/>
</tag>
.
.
.
<tag type="map-string-bool" name="02">
     <entry index="01" value="true"/>
     .
     .
     .
     <entry index="10" value="false"/>
</tag>
```

Για να φορτώσουμε τις ετικέτες στο περιβάλλον της προσομοίωσης ώστε να τις εξετάσει το μοντέλο χρησιμοποιούμε στην εργασία *prepare_world* την παράμετρο *"env_config"* στην οποία υποδεικνύουμε το αρχείο XML όπου έχουμε αποθηκεύσει τις ετικέτες. Είναι επίσης δυνατό να τις αποθηκεύσουμε στο ίδιο XML που πχ έχουμε αποθηκεύσει τις θέσεις των κόμβων, αρκεί να τηρείται η σωστή σύνταξη [14].

Η μέθοδος εξετάζει την επικοινωνία μόνο για τις ετικέτες χάρτη που έχουμε δηλώσει, και αποκλειστικά για τις κατευθύνσεις που αυτές υλοποιούν. Όσοι κόμβοι είναι φορτωμένοι στον κόσμο αλλά δεν συμπεριλαμβάνονται στις ετικέτες δεν εξετάζονται. Είναι βέβαια δυνατό με το 'Multiple' μοντέλο επικοινωνίας να εφαρμόσουμε άλλο μοντέλο για αυτούς. Για κάθε ετικέτα χάρτη εξετάζεται αν ο κόμβος που είναι στο όνομά του (*"name"*) επικοινωνεί με αυτούς που είναι στο ευρετήριο της ετικέτας (*"index"*) και καθίσταται ανάλογα της τιμής bool (*"value"*) αμφίδρομη ή μονόδρομη επικοινωνία. Κάθε άλλη σχέση επικοινωνίας δεν μελετάται από το μοντέλο.

1.1.1.4 rim_comm_model

Το μοντέλο αυτό προσεγγίζει καλύτερα από όλα τις πραγματικές συνθήκες μετάδοσης του σήματος. Το 'RIM' εφαρμόζει διακύμανση στο εύρος της εκπομπής με βάση την διεύθυνση αναπαραγωγής του σήματος, σε αντίθεση με το 'UDG' που χρησιμοποιεί σταθερό ακτινικό εύρος εκπομπής (*rmax*).

Το 'RIM' μοντέλο επικοινωνίας εκκινείται από ένα απλό 'DOI' (Degree Of Irregularity = βαθμός ανωμαλίας) μοντέλο, το οποίο ορίζει την μέγιστη διακύμανση του εύρους εκπομπής ανά μονάδα γωνίας στην κατεύθυνση αναπαραγωγής του σήματος. Αν ο βαθμός ανωμαλίας (DOI) είναι μηδέν, τότε δεν υπάρχει διακύμανση στο εύρος εκπομπής και το 'RIM' μοντέλο συμπεριφέρεται σαν το 'UDG', με σταθερό εύρος εκπομπής ανά γωνία κατεύθυνσης. Δηλαδή το εύρος εκπομπής είναι τέλειος κύκλος. Όσο όμως αυξάνουμε την τιμή του βαθμού ανωμαλίας τόσο πιο ανώμαλο γίνεται το σχήμα του εύρους εκπομπής.

Το μοντέλο 'DOI' θεωρεί ένα κατώτερο και ένα ανώτερο (ελάχιστο, μέγιστο) όριο, μεταξύ των οποίων κυμαίνεται το εύρος μετάδοσης του σήματος (*min_range*, *max_range*). Πέρα από το ανώτερο όριο δεν υφίσταται επικοινωνία μεταξύ δύο κόμβων, ενώ εντός του κατώτερου η επικοινωνία μεταξύ τους είναι 100% εγγυημένη. Κατά την εκκίνηση του μοντέλου δημιουργούνται 360 προσωρινές τιμές που αντιπροσωπεύουν τη διακύμανση του εύρους μετάδοσης ανά μοίρα γωνίας. Οι τιμές αυτές επηρεάζονται άμεσα από τον παράγοντα 'DOI' και αποδίδονται με κατανομή 'Weibull' τυχαίας μεταβλητής, η οποία επίσης εξαρτάται από δύο παραμέτρους α και β (*alpha* και *beta*) όπου πρέπει να είναι: α, β > 0. Όλες οι τιμές υπολογίζονται στο ενδιάμεσο μεταξύ κατώτερου και ανώτερου ορίου (*min_range* < Weibull τιμές < *max_range*). Από τις 360 προσωρινές αυτές τιμές ένα ποσοστό θα χρησιμοποιηθεί στην τελική συχνότητα διακύμανσης εύρους μέσω της παραμέτρου "granularity", η οποία πρέπει να είναι ακέραιος διαιρέτης του 360:

```
granularity >= 1 && 360 % granularity == 0
```

Οι παράμετροι μέση τιμή (*mean*) και μεταβλητότητα (*variance*) ορίζουν την Κανονική Κατανομή Τυχαίας Μεταβλητής που χρησιμοποιείται για τον προσδιορισμό της μέγιστης ποσοστιαίας μεταβολής της ισχύος εκπομπής (*vsp*). Αυτή χρησιμοποιείται στον τελικό υπολογισμό της δυνατότητας επικοινωνίας μεταξύ δύο κόμβων. Εξετάζεται με διάφορες μετατροπές η εξής σχέση:

Αν η ευκλείδεια απόσταση ανάμεσα σε δύο κόμβους 'u' , 'v' είναι μικρότερη από το γινόμενο

```
range *  min_range * vsp
```

τότε οι κόμβοι επικοινωνούν. Σε αντίθετη περίπτωση δεν επικοινωνούν.

Τέλος βάσει της γωνίας μετάδοσης υπολογίζεται το τελικό εύρος εκπομπής ανά κατεύθυνση.

Όλοι οι κόμβοι χρησιμοποιούν την ίδια τιμή *vsp* καθώς και τις ίδιες τελικές τιμές διακύμανσης του εύρους μετάδοσης. Όλες οι παράμετροι του 'RIM' μοντέλου είναι προαιρετικές.

1.1.1.5 stochastic_comm_model

Το μοντέλο αυτό εκτιμά την επικοινωνία δύο κόμβων βάσει τριών παραγόντων:

1. *range* (double)
2. *max_probability* (double), >0, <=1.0, default 1.0
3. *smooth_factor* (double), >0, <=1.0, default 0.25

Και οι τρεις παράγοντες είναι προαιρετικοί και συνοδεύουν τη δήλωση του μοντέλου. Το 'εύρος' (*range*) αξιοποιείται κατά την κλασσική χρήση. Οι παράγοντες 'μέγιστη πιθανότητα' (*max_probability*) και 'παράγοντας εξομάλυνσης' (*smooth factor*) συνδυάζονται στην μέθοδο '*double communication_probability(double distance)*' της '*void init()*' μεθόδου του μοντέλου, προκειμένου να αποδώσουν μια τελική πιθανότητα επικοινωνίας. Η παράμετρος εισαγωγής απόστασης *distance* είναι προφανώς κάθε φορά διαφορετική για ένα ζεύγος κόμβων, και συνεπώς για κάθε ζεύγος έχουμε διαφορετική πιθανότητα επικοινωνίας. Όσο μεγαλύτερη η απόσταση ανάμεσα σε δύο κόμβους, τόσο μικρότερη είναι η πιθανότητα επικοινωνίας μεταξύ τους. Αυτή συγκρίνεται για κάθε ζεύγος με μια ενιαία για όλους τους κόμβους Ομοιόμορφη Κατανομή Τυχαίας Μεταβλητής (UniformRandomVariable), και αν είναι μεγαλύτερη τότε υφίσταται επικοινωνία για τους κόμβους.

Το στοχαστικό μοντέλο επικοινωνίας είναι ουσιαστικά ένα βελτιωμένο 'UDG' μοντέλο. Η διαφορά είναι ότι ενώ στο απλό 'UDG' το εύρος επικοινωνίας και η απόσταση των κόμβων καθορίζουν χωρίς κανένα άλλο κριτήριο της επικοινωνία, πλέον υπεισέρχεται και η πιθανότητα σαν παράγοντας ώστε για οριακές κυρίως καταστάσεις να προσδώσει ρεαλισμό στο μοντέλο.

1.1.1.6 manual_communication_model

Συνδυάζεται μόνο με το 'Manual' μοντέλο ακμών. Επιτρέπει χειροκίνητη προσθήκη ακμών (*virtual void add_edge(Node&, Node&)*) ανάμεσα σε δύο κόμβους χωρίς κάποια συνθήκη (απόσταση ή οτιδήποτε άλλο). Προκειμένου να επικοινωνούν δύο κόμβοι πρέπει να έχει γίνει εκ των προτέρων χρήση της '*virtual void add_edge(Node&, Node&)*', η οποία καλεί το 'Manual' μοντέλο ακμών, για να προστεθούν συνδέσεις.

1.1.1.7 multiple_communication_model

Το μοντέλο αυτό επιτρέπει τον συνδυασμό πολλαπλών μοντέλων σε μια προσομοίωση. Για παράδειγμα μερικοί κόμβοι μπορούν να χρησιμοποιούν το 'UDG' μοντέλο ενώ άλλοι να έχουν στατικές συνδέσεις μεταξύ τους μέσω του 'Μοντέλου Επικοινωνίας Μόνιμης Σύνδεσης'. Η υλοποίησή του γίνεται ως εξής:

Ορίζουμε λίστα με τα μοντέλα επικοινωνίας που θα χρησιμοποιήσουμε. Μπορούμε προαιρετικά να θέσουμε κάποιο μοντέλο ως προεπιλογή, το οποίο θα εφαρμοστεί εφόσον τα μοντέλα της λίστας δεν μπορούν να υλοποιήσουν επικοινωνία ανάμεσα στους κόμβους. Οι παράμετροι και η σύνταξη της δήλωσής τους είναι:

```
comm_models=$COMM_MODEL1,$COMM_MODEL2,.....
(τα μοντέλα χωρίζονται με κόμμα)

comm_default_model=$COMM_DEFAULT_MODEL
```

Το 'Multiple'' μοντέλο επικοινωνίας εφαρμόζεται ως εξής:

- Για κάθε μοντέλο της λίστας εξετάζεται αν μπορεί να δώσει απάντηση στο ερώτημα αν δύο κόμβοι 'u' και 'v' μπορούν να επικοινωνήσουν. Δηλαδή δεν εξετάζεται αν επικοινωνούν ή όχι οι κόμβοι, αλλά αν μπορούν να δώσουν απάντηση οι μέθοδοι *can_communicate_uni()* και *can_communicate_bidi()*. Αυτό γίνεται με την κλήση της '*virtual bool in_domain(const Node& u, const Node& v)'*, όπως υλοποιείται σε κάθε μοντέλο. Η προεπιλεγμένη ρύθμιση επιστρέφει απάντηση “true”, δηλαδή ουσιαστικά δεν εξετάζει τίποτα, δίνοντας το πράσινο φως για το επόμενο βήμα, και εφαρμόζεται στα 'Disk Graph', 'RIM', 'Stochastic', 'Manual' και 'Multiple' μοντέλα επικοινωνίας. Τα μόνα μοντέλα που αποκλίνουν της προεπιλογής είναι τα 'Link Probability' και 'PermaLink'. Και στα δύο η '*in_domain()*' λειτουργία "μεταφράζεται" στη μέθοδο '*bool can_communicate_uni(const Node& u, const Node& v)*' του εκάστοτε μοντέλου.

- Εφόσον κάποιο από τα μοντέλα της λίστας με τη σειρά που τα έχουμε δηλώσει, επιτρέπει την εξέταση της επικοινωνίας ανάμεσα στους κόμβους, τότε γίνεται το ερώτημα επ' αυτής και επιστρέφεται η απάντηση στο 'Multiple' μοντέλο επικοινωνίας και στην προσομοίωση. Συνεπώς για τους συγκεκριμένους κόμβους 'u' και 'v' η επικοινωνία υλοποιείται βάσει του πρώτου μοντέλου της λίστας που μπορεί να δώσει απάντηση.

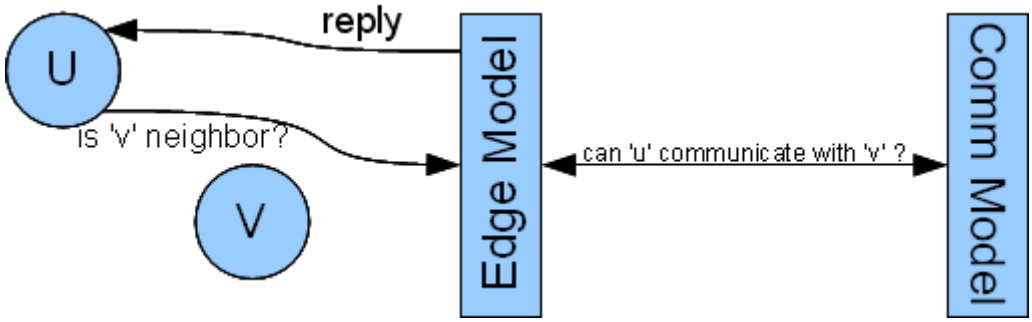
- Αν κανένα από τα μοντέλα της λίστας δεν μπορεί να ικανοποιήσει το ερώτημα για επικοινωνία, τότε εξετάζεται το "comm_default_model". Εδώ δεν εξετάζεται αν μπορεί να απαντήσει το μοντέλο, αλλά γίνεται ευθέως το ερώτημα για την ύπαρξη ή μη επικοινωνίας ανάμεσα στους κόμβους.

Όπως γίνεται κατανοητό αν σε μια λίστα πχ με τρία μοντέλα και μια προεπιλογή, δίνουν απάντηση όλα και λόγου χάρη με το 1ο στην λίστα οι κόμβοι δεν επικοινωνούν ενώ με τα υπόλοιπα επικοινωνούν, η απάντηση του 'Multiple' μοντέλου επικοινωνίας θα είναι ότι δεν επικοινωνούν, καθότι αφού θα πάρει απάντηση από το 1ο μοντέλο της λίστας δεν θα εξετάσει τα υπόλοιπα. Φυσικά είναι δυνατό να τροποποιηθεί η πολιτική αυτή ώστε να επιλέξουμε μοντέλο από την λίστα με διαφορετικά κριτήρια από την θέση του σε αυτή.

Το 'Multiple' μοντέλο έχει μεγάλη χρησιμότητα να εφαρμόζεται όταν θέλουμε να έχουμε στατικές συνδέσεις ανάμεσα σε κάποιους κόμβους, αλλά παράλληλα να εξετάζουμε και την επικοινωνία των υπολοίπων. Αυτό συμβαίνει όταν χρησιμοποιούμε το 'Link Probability' ή το PermaLink' μοντέλο. Συνήθως θα προκύψουν αρκετά ζεύγη κόμβων για τα οποία τα δύο αυτά μοντέλα δε θα μπορούν να δώσουν απάντηση αν επικοινωνούν ή όχι, εκτός βέβαια αν συνδέσουμε στατικά όλα τους κόμβους μεταξύ τους σε ετικέτες χάρτη. Αν επομένως θέλουμε για τα ζεύγη αυτά να εξετάσουμε την επικοινωνία πρέπει να χρησιμοποιήσουμε κάποιο άλλο μοντέλο μέσω της υλοποίησης του 'Multiple' μοντέλου επικοινωνίας.

1.1.2 Μοντέλο Ακμών (Edge Model)

Ενώ το μοντέλο επικοινωνίας παρέχει τις μεθόδους με τις οποίες αποφασίζεται αν δύο κόμβοι μπορούν να επικοινωνήσουν, εν τούτοις δεν διαχειρίζεται αυτή την πληροφορία. Αν για παράδειγμα θέλουμε να βρούμε τους γείτονες ενός κόμβου, δηλαδή τους κόμβους με τους οποίους επικοινωνεί, τότε θα πρέπει χειροκίνητα να εξετάσουμε την επικοινωνία για όλες τις περιπτώσεις με το μοντέλο επικοινωνίας, και με κάποια στατιστική μέθοδο να υπολογίσουμε το σύνολο. Το μοντέλο ακμών παίζει αυτόν ακριβώς το ρόλο, να είναι δηλαδή διαχειριστής του μοντέλου επικοινωνίας και να δίνει μεθόδους με τις οποίες αξιοποιείται η πληροφορία της επικοινωνίας ή μη ανάμεσα στους κόμβους του δικτύου. Μπορούμε να πούμε πως το μοντέλο επικοινωνίας δίνει απάντηση στο ερώτημα αν μπορούν να επικοινωνήσουν δύο κόμβοι, και το μοντέλο ακμών απαντά στο ερώτημα αν επικοινωνούν.



Σχήμα 1.2

Το μοντέλο ακμών παρέχει μια γραφική παράσταση του δικτύου. Οι κορυφές του γραφήματος είναι οι κόμβοι. Δύο κόμβοι συνδέονται μεταξύ τους με ακμή (edge) αν επικοινωνούν. Με το μοντέλο ακμών μπορούμε να βρούμε όλους τους δυνατούς παραλήπτες ενός μηνύματος, είτε αυτοί είναι γείτονες του κόμβου που το στέλνει είτε γείτονες των γειτόνων κτλ. Δηλαδή αυτή η γραφική παράσταση του δικτύου που δημιουργεί το μοντέλο ακμών είναι ουσιαστικά οι δρόμοι που μπορεί να ακολουθήσει ένα μήνυμα στο δίκτυο. Αυτό όμως εξαρτάται από έναν επιπλέον παράγοντα που υπολογίζουμε στην επικοινωνία και που ειδικά αντιμετωπίζει το μοντέλο ακμών, την κατεύθυνσή της [14].

Για δύο κόμβους του δικτύου το μοντέλο ακμών, σε συνεργασία με το μοντέλο επικοινωνίας, αποδίδει την 'δυνατότητα της επικοινωνίας στην εκάστοτε κατεύθυνση'. Το μοντέλο επικοινωνίας με τη μέθοδο '*bool can_communicate_uni(u,v)*' εξετάζει αν μπορεί να επικοινωνήσει ο κόμβος 'u' με τον κόμβο 'v'. Πρόκειται δηλαδή για μονόδρομη επικοινωνία. Αν αυτό είναι αληθές για δύο κόμβους (u, v) αλλά δεν ισχύει το αντίθετο (δηλαδή *can_communicate_uni(v,u) = false*), τότε η επικοινωνία του u με το v για την διεύθυνση "out" είναι δυνατή, ενώ για τη διεύθυνση "in" δεν είναι. Επίσης εννοείται ότι δεν είναι δυνατή σε αυτή την περίπτωση η αμφίδρομη επικοινωνία ("bidi"). Αυτή είναι η λογική με την οποία το μοντέλο ακμών εξετάζει αν γειτονεύουν δύο κόμβοι.

Για δύο κόμβους 'u' και 'v' καλείται η μέθοδος '*bool are_adjacent(const Node& u, const Node& v, CommunicationDirection d)*', αξιοποιώντας τις μεθόδους '*bool can_communicate_uni(const Node&, const Node&)'* και '*bool can_communicate_bidi(const Node&, const Node&)*' του μοντέλου επικοινωνίας. Παράμετροι της μεθόδου είναι οι δύο κόμβοι που εξετάζουμε και η κατεύθυνση επικοινωνίας. Ανάλογα με τη κατεύθυνση επικοινωνίας που εξετάζουμε η μέθοδος καλεί την αντίστοιχη λειτουργία όπως φαίνεται στον κώδικα:

```
bool
EdgeModel::
are_adjacent( const Node& u, const Node& v,
CommunicationDirection d )
const throw()
{
    switch( d )
    {
    case CD_IN:
        return communication_model().can_communicate_uni(v,u);

    case CD_OUT:
         return communication_model().can_communicate_uni(u,v);

    case CD_BIDI:
        return communication_model().can_communicate_bidi(u,v);

    case CD_ANY:
        return communication_model().can_communicate_uni(u,v) ||
        communication_model().can_communicate_uni(v,u);
    }

    return false;

}
```

Ανάλογα με τη κατεύθυνση επικοινωνίας που εξετάζουμε γίνεται η αντίστοιχη εκλογή σύνταξης του ελέγχου (switch statement).

Εξετάζεται με το μοντέλο επικοινωνίας η δυνατότητα επικοινωνίας ανάμεσα σε δύο κόμβους 'u', 'v', σύμφωνα με τη σύνταξη της κατεύθυνσης που εξετάζουμε (χωρίς να παίζει ρόλο η κατεύθυνση στον υπολογισμό, εκτός αν με κάποιο τρόπο την έχουμε υλοποιήσει και στο μοντέλο επικοινωνίας)

Εφόσον μπορούν να επικοινωνήσουν δύο κόμβοι 'u', 'v' βάσει του μοντέλου επικοινωνίας, η κατεύθυνση αυτής ορίζεται ως εξής:

"in" : το 'u' δέχεται μηνύματα από το 'v'
"out" : το 'u' στέλνει μηνύματα στο 'v'
"bidi" : το 'u' δέχεται μηνύματα από το 'v' και το 'u' στέλνει μηνύματα στο 'v'
"any" : το 'u' δέχεται μηνύματα από το 'v' και/ή το 'u' στέλνει μηνύματα στο 'v'

Αν για τον εκάστοτε κατεύθυνση επικοινωνίας ισχύει η αντίστοιχη συνθήκη, τότε οι κόμβοι 'u', 'v' είναι γείτονες (δηλαδή επικοινωνούν). Για παράδειγμα αν επιχειρήσουμε να στείλουμε μήνυμα από το 'u' στο 'v' αλλά για την κατεύθυνση "out" η '*bool can_communicate_uni(u,v)*' επιστρέφει "false", τότε οι κόμβοι δεν είναι γείτονες και δεν επικοινωνούν.

Κάθε μοντέλο επικοινωνίας έχει την δική του υλοποίηση στις μεθόδους εκτίμησης της επικοινωνίας ανάμεσα σε δύο κόμβους. Ομοίως και για το μοντέλο ακμών υπάρχουν αρκετές υλοποιήσεις, κάθε μια από της οποίες εξετάζει διαφορετικά το θέμα της γειτνίασης. Η μέθοδος '*bool are_adjacent(const Node& u, const Node& v, CommunicationDirection d)*' εφαρμόζεται όμοια σε όλα τα ήδη υλοποιημένα μοντέλα. Φυσικά είναι δυνατό να γίνει τροποποίησή της, είτε στην γονική τάξη, είτε ξεχωριστά σε κάποιο μοντέλο. Η άλλη κύρια μέθοδος που χρησιμοποιεί το μοντέλο ακμών είναι η '*virtual int nof_adjacent_nodes(const Node&, EdgeModel::CommunicationDirection d)*', και η οποία επιστρέφει τον αριθμό των κόμβων με τους οποίους επικοινωνεί ένας κόμβος για μια κατεύθυνση *d*. Η παράμετρος *CommunicationDirection* ανήκει στη γονική τάξη του μοντέλου ακμών και υλοποιείται με κοινόχρηστο enum (enumeration), οπότε την χρησιμοποιούμε ευθέως. Η μέθοδος αυτή υλοποιείται διαφορετικά στα διάφορα μοντέλα ακμών, και είναι αυτή που κυρίως ορίζει τον χαρακτήρα τους.

Τα υλοποιημένα Μοντέλα Ακμών του Shawn είναι:

1.1.2.1 Lazy Edge Model

Το 'Lazy' Μοντέλο Ακμών ή αλλιώς απλό (simple) μοντέλο, είναι προσανατολισμένο στην χρήση σε μικρά δίκτυα, δεδομένου ότι δεν αποθηκεύει πληροφορία και κάθε φορά υπολογίζει το γράφημα των ακμών 'στον αέρα' (on the fly) κάνοντας κλήσεις στο αντίστοιχο μοντέλο επικοινωνίας. Αυτό έχει το πλεονέκτημα της πλήρους δυναμικής απόδοσης των ακμών, σε βάρος βέβαια της ταχύτητας. Έτσι αν έχουμε ένα σενάριο με κινούμενους κόμβους ή με μεταβλητές παραμέτρους υπολογισμού επιπλέον της θέσης των κόμβων, το μοντέλο ανταποκρίνεται στις ανάγκες επανυπολογισμού του γραφήματος των ακμών. Για κάθε κόμβο καλούνται και οι δύο θεμελιώδεις μέθοδοι '*virtual int nof_adjacent_nodes(const Node&, EdgeModel::CommunicationDirection d)*' και '*bool are_adjacent(const Node& u, const Node& v, CommunicationDirection d)*'. Η φύση του μοντέλου δημιουργεί μεγάλους χρόνους καθυστέρησης για μεγάλα δίκτυα και ιδίως αν έχουμε και κινούμενους κόμβους, οπότε οι συνεχείς επανυπολογισμοί γίνονται πάρα πολλοί.

1.1.2.2 Grid Edge Model

Το 'Grid' Μοντέλο Ακμών χρησιμοποιεί ένα δισδιάστατο κάναβο, στον οποίο κατανέμει τους κόμβους βάσει της γεωμετρικής τους θέσης. Κάθε κόμβος αποδίδεται στο κελί του κανάβου που τον περιέχει. Το μέγεθος των κελιών (το βήμα του κανάβου) ρυθμίζεται με το εύρος εκπομπής (trasmission range). Κάθε κόμβος εξετάζει για γείτονες μόνο στο δικό του κελί και στα γειτονικά του. Αυτό επιτρέπει πολύ γρήγορους υπολογισμούς σε μεγάλα δίκτυα, διότι δεν απαιτείται να εξετάζουμε τις αποστάσεις όλων των κόμβων μεταξύ τους, αλλά μόνο όσων είναι σε γειτονικά κελιά. Αν στο σενάριό μας έχουμε κινούμενους κόμβους, τότε αυτοί κάθε φορά αποδίδονται στο κελί που βρίσκονται την εκάστοτε χρονική στιγμή [14].

Το μοντέλο δέχεται τρεις προαιρετικές παραμέτρους με τη δήλωσή του:

grid_size (int)

- Οι διαστάσεις του κανάβου: grid = grid_size x grid_size cells. Προεπιλεγμένη τιμή '8' και ελάχιστη '1'.

grid_cell_size (double)

- Η παράμετρος αυτή πολλαπλασιάζεται με το εύρος εκπομπής για να δώσει της διαστάσεις του κελιού. Η παράμετρος δεν μπορεί να είναι μικρότερη από '1.01' (προεπιλεγμένη τιμή '1.5').

grid_closeness_fraction (double)

- Κλάσμα εγγύτητας: Ορίζει τα όρια του κελιού στον υπολογισμό των γειτόνων(!?). Πρέπει να παίρνει τιμές μεταξύ '0' και '1' ($0 < \text{grid\_closeness\_distance} < 1$) και έχει προεπιλεγμένη τιμή '0.01'. Πολλαπλασιάζεται με το μισό της διαφοράς της διάστασης του κελιού μείον το εύρος εκπομπής, και το αποτέλεσμα που είναι απόσταση προστίθεται στο εύρος του κελιού κατά τις τέσσερις διευθύνσεις για τον υπολογισμό των γειτόνων των κόμβων. Δεν αλλάζει ούτε το εύρος εκπομπής ούτε το μέγεθος του κελιού.

1.1.2.3 List Edge Model

Το 'List' Μοντέλο Ακμών αποθηκεύει όλες τις ακμές του δικτύου στη μνήμη του συστήματος, επιτρέποντας έτσι γρηγορότερη προσπέλαση των γειτόνων ενός κόμβου. Η τεχνική της εύρεσης των γειτόνων ενός κόμβου είναι ίδια με αυτή του 'Lazy' μοντέλου ακμών, και εφαρμόζεται μία μόνο φορά κατά την εκκίνηση της προσομοίωσης, οπότε και δομείται η σχετική λίστα. Προφανώς σε μεγάλα δίκτυα είναι μεγάλος και ο χρόνος κατασκευής της λίστας των γειτόνων για κάθε κόμβο, και συνεπακόλουθα μεγάλο το κόστος σε μνήμη για την αποθήκευσή της. Επιπλέον λόγω της στατικότητας αυτής το μοντέλο δεν επιτρέπει κινούμενους κόμβους.

1.1.2.4 Fast List Edge Model

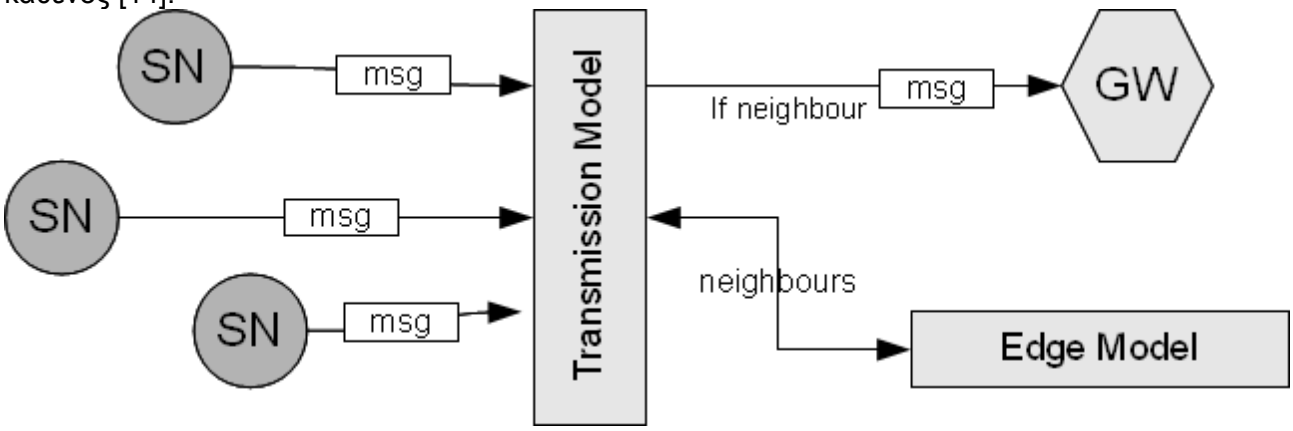
Το 'Fast List' Μοντέλο Ακμών συνδυάζει τη λειτουργικότητα των 'Grid' και 'List' μοντέλων σε ένα μοναδικό μοντέλο. Ουσιαστικά είναι παράγωγη τάξη του 'List' Μοντέλου και χρησιμοποιεί ένα δείκτη στο 'Grid' Μοντέλο για να καλεί τις μεθόδους του ώστε να υπολογίζει τους γείτονες των κόμβων. Επομένως η αρχική κατασκευή της λίστας των γειτόνων για κάθε κόμβο γίνεται από το 'Grid' Μοντέλο και αποθηκεύεται στη μνήμη με το 'List'. Αυτό επιτρέπει ταχύτερη κατασκευή της λίστας, με το κόστος της μεγαλύτερης κατανάλωσης μνήμης κατά τη κατασκευή, λόγω της φύσης του 'Grid' Μοντέλου και των λειτουργιών του.

1.1.2.5 Manual Edge Model

Αποτελεί γονική τάξη για το 'List' Μοντέλο Ακμών και δίνει την δήλωση της μεθόδου '*virtual void add_edge(Node&, Node&)*'. Με αυτή το μοντέλο ελέγχει την επικοινωνία ανάμεσα σε δύο κόμβους και για όλες τις κατευθύνσεις (CD_IN, CD_OUT, CD_ANY, CD_BIDI) και αποθηκεύει τα αποτελέσματα σε ένα bool array.

1.1.3 Μοντέλο Εκπομπής (Transmission Model)

Προσομοιώνει τα χαρακτηριστικά του μέσου/καναλιού επικοινωνίας και καθορίζει πως μεταδίδονται τα μυνήματα. Το μοντέλο εκπομπής για κάθε αποστολή μηνύματος εξετάζει μέσω του μοντέλου ακμών τους γείτονες του αποστολέα. Το μοντέλο ακμών επιστέφει τους γειτονικούς κόμβους και στη συνέχεια το μοντέλο εκπομπής στέλνει το μήνυμα διαδοχικά στους επεξεργαστές καθενός [14].



Σχήμα 1.3

Το μοντέλο εκπομπής είναι άμεσα και αποκλειστικά συνδεδεμένο με τη διαχείριση των μηνυμάτων επικοινωνίας ανάμεσα στου κόμβους του δικτύου. Τόσο το μοντέλο ακμών όσο και το μοντέλο επικοινωνίας δεν έχουν απολύτως καμία επαφή με τα μηνύματα που ανταλλάσσουν οι κόμβοι. Αντίστοιχα όμως το μοντέλο εκπομπής δεν έχει κανένα ρόλο στον καθορισμό της επικοινωνίας ανάμεσα στους κόμβους. Μπορεί μόνο να απορρίπτει μηνύματα με δικά του κριτήρια, τα οποία δεν αλλάζουν την ιδιότητα της επικοινωνίας.

Το μοντέλο εκπομπής βασίζει τη λειτουργία του σε δύο κυρίως μεθόδους:

virtual void send_message(MessageInfo& mi)

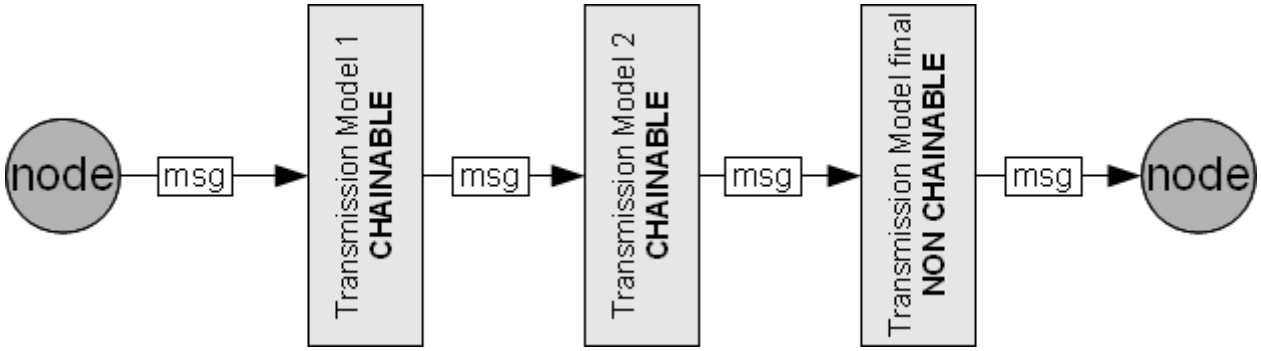
- αποδέχεται μηνύματα προς αποστολή και τα αποθηκεύει σε μια λίστα (queue) για αποστολή στον επόμενο γύρο.

virtual void deliver_messages()

- αποστέλλει όλα τα μηνύματα που είναι στη λίστα.

Οι δύο αυτές μέθοδοι υλοποιούνται διαφορετικά στα διάφορα μοντέλα εκπομπής του shawn. Η αποστολή μηνυμάτων δεν είναι δεσμευμένη στη μέθοδο *deliver_messages()*. Είναι δυνατό να αποσταλούν μηνύματα με οποιαδήποτε μέθοδο, όπως με χρήση πχ του προγραμματιστή γεγονότων (Event Scheduler).

Τα μοντέλα εκπομπής είναι δυνατό να είναι 'αλυσιδωτά' (chainable), δηλαδή να μπορούν να συνδεθούν σε σειρά. Σε αυτή τη περίπτωση τα μηνύματα μεταδίδονται από το ένα μοντέλο στο άλλο πριν καταλήξουν στους κόμβους που πρέπει να τα παραλάβουν. Κάθε μοντέλο διαχειρίζεται τα μηνύματα που παραλαμβάνει με τις δικές του μεθόδους και μετά τα παραδίδει στο επόμενο, μέχρι το τελευταίο στην αλυσίδα να αποστείλει τα μηνύματα που έχουν περάσει όλα τα φιλτραρίσματα στους κόμβους που μπορούν να τα παραλάβουν. Το τελευταίο μοντέλο της αλυσίδας πρέπει οπωσδήποτε να μην είναι αλυσιδωτό [14].



Σχήμα 1.4

Οι κύριες επιπλέον μέθοδοι που υπεισέρχονται στην οικοδόμηση της αλυσίδας και στη ροή των μηνυμάτων είναι:

void append_to_chain(TransmissionModel&)

- προσθέτει ένα μοντέλο στην αλυσίδα. Μόνο το τελευταίο μοντέλο της αλυσίδας μπορεί (και πρέπει) να μην είναι αλυσιδωτό. Αν προσπαθήσουμε να προσαρτήσουμε ένα μοντέλο σε ένα μη αλυσιδωτό, το shawn θα βγάλει σφάλμα.

void pass_to_chain(MessageInfo& mi)

- Η μέθοδος αυτή καλείται από την send_message(TransmissionModel::MessageInfo& mi), και ουσιαστικά αντικαθιστά την deliver_messages(). Αντί να αποστέλλονται τα μηνύματα στους κόμβους, παραδίδονται στο επόμενο μοντέλο για περαιτέρω επεξεργασία.

1.1.3.1 Αλυσιδωτά Μοντέλα Εκπομπής (Chainable Transmission Models)

Το Shawn διαθέτει αρκετά Transmission Models. Τα αλυσιδωτά μοντέλα ταυ είναι:

1.1.3.1.1 stats_chain_transmission_model

Μοντέλο που μετράει τα μηνύματα που αποστέλλονται. Δεν τροποποιεί κατά κανένα τρόπο τα μυνήματα, απλά τα μετράει. Τα στατιστικά εκτυπώνονται με το task *dump_transmission_stats* και έχουν ενδεικτικά την παρακάτω μορφή:

```
Simulation: Running task 'dump_transmission_stats'
---- stats_chain transmission model information -----------------
general all_types messages 112
general all_types size 128
general N10simple_app16SimpleAppMessageE messages 112
general N10simple_app16SimpleAppMessageE size 128
-------------------------------------------------------------
Simulation: Task done 'dump_transmission_stats'
```

Όπως φαίνεται αναφέρεται ο συνολικός αριθμός των μηνυμάτων και το μέγεθός τους, καθώς και τα αντίστοιχα μεγέθη ανά είδος μηνυμάτων. Η εργασία εφαρμόζει την d*ump_stats(std::ostream& os)* μέθοδο του μοντέλου. Υπάρχει και η *dump_node_stats(const Node& node, std::ostream& os)* η οποία εκτυπώνει στατιστικά για κάθε κόμβο και μπορεί εύκολα να ενσωματωθεί στο task.

Το μοντέλο δηλώνεται στο αρχείο ρυθμίσεων ή στη γραμμή εντολών με τη παράμετρο "transm_model=stats_chain". Λόγω του ότι είναι αλυσιδωτό μοντέλο πρέπει οπωσδήποτε να το συνδέσουμε με άλλο μοντέλο, είτε αλυσιδωτό είτε όχι, αρκεί στο τέλος η αλυσίδα να τερματίζεται με μη αλυσιδωτό μοντέλο. Η σύνδεση αυτή γίνεται με το task *chain_transm_model*, το οποίο δέχεται ως παράμετρο το όνομα του μοντέλου που συνδέουμε("name=$tranmission_model").

1.1.3.1.2 random_drop_transmission_model

Μοντέλο που απορρίπτει μήνυμα βάσει μιας 'πιθανότητας απόρριψης'. Ο παράγοντας αυτός ορίζεται με την απαραίτητη παράμετρο "probability" (double), που πρέπει να συνοδεύει την δήλωση του μοντέλου, και πρέπει να παίρνει τιμές μεταξύ 0.0 και 1.0. Επιτρέπονται και τιμές μεγαλύτερες από 1.0, έχοντας το ίδιο αποτέλεσμα, δηλαδή να μην απορρίπτεται κανένα μήνυμα.

Το μοντέλο δηλώνεται με τη παράμετρο "trans_model=random_drop_chain". Περί της αλυσίδας ισχύουν τα ίδια με το stats_chain model.

1.1.3.2 Μη Αλυσιδωτά Μοντέλα Εκπομπής (NON-Chainable Transmission Models)

Πέρα από τα αλυσιδωτά μοντέλα το shawn διαθέτει και αρκετά 'απλά' μοντέλα εκπομπής. Αυτά μπορούν να είναι μέρος αλυσίδας μοντέλων, οπότε και επιβάλλεται να είναι το τελευταίο μοντέλο στην αλυσίδα. Φυσικά μπορούν να χρησιμοποιούνται ως τελείως αυτόνομα μοντέλα. Τα μοντέλα αυτά είναι:

1.1.3.2.1 reliable_transmission_model

Το μοντέλο αυτό αποστέλλει όλα τα μηνύματα χωρίς καμία απώλεια. Δηλώνεται με την παράμετρο "trans_model=reliable" ή αν τερματίζει αλυσίδα μοντέλων με το task *chain_transm_mode*l και παράμετρο "name=reliable". Επίσης δέχεται προαιρετική παράμετρο "άμεση παράδοση" ("immediate_delivery" (bool)) με προεπιλεγμένη τιμή “false”. Αν ενεργοποιήσουμε την άμεση παράδοση τότε τα μηνύματα παραδίδονται αμέσως στα nodes και όχι στην αρχή του επόμενου γύρου που είναι η κανονική συμπεριφορά του μοντέλου. Το μοντέλο υλοποιείται ως εξής:

Όταν ένας κόμβος αποστέλλει μήνυμα, καλείται η *send_messages()* μέθοδος, με την οποία δημιουργείται μια λίστα αναμονής εισερχομένων (queue) στην οποία αποθηκεύεται το μήνυμα. Κατόπιν το μοντέλο ελέγχει αν θέλουμε άμεση παράδοση ή όχι, καθώς και αν το μήνυμα έχει αποσταλεί μέσω του προγραμματιστή γεγονότων (Event Scheduler). Υπάρχουν δύο δυνατές ενέργειες:

- Αν το μήνυμα έχει αποσταλεί μέσω του Event Scheduler ή αν έχουμε άμεση παράδοση ενεργή, τότε καλείται η μέθοδος deliver_messages() η οποία μεταφέρει όλα τα μηνύματα σε μια δεύτερη λίστα αναμονής (queue) εξερχομένων μηνυμάτων και καλεί τη μέθοδο deliver_one_message() για να αρχίσει η αποστολή των μηνυμάτων. Η μέθοδος αυτή πρώτα ελέγχει αν το μήνυμα είναι unicast (δηλαδή να απευθύνεται σε ένα μόνο κόμβο). Αν ναι, τότε ακυρώνεται η μετάδοση γιατί δεν υποστηρίζεται από την υλοποίηση. Αν δεν είναι unicast τότε βάσει των στοιχείων συνδεσιμότητας που μας δίνει το μοντέλο Ακμών, αποστέλλονται τα μηνύματα στους δυνατούς παραλήπτες και επιβεβαιώνεται η παραλαβή τους.

- Στην περίπτωση που δεν θέλουμε άμεση παράδοση, καλείται ένα νέο γεγονός από τον Event Scheduler για χρόνο "current_time() + EPSILON_TIME". Η timeout() μέθοδος του Event Scheduler καλείται στον χρόνο αυτό και καλεί την deliver_messages(), οπότε η συνέχεια είναι ίδια με προηγουμένως. Τροποποιώντας το EPSILON_TIME μπορούμε να δώσουμε την επιθυμητή καθυστέρηση στην αποστολή μηνυμάτων.

Είναι σαφές ότι το reliable μοντέλο είναι ιδανικό για περιπτώσεις που δεν μας ενδιαφέρει να εξετάσουμε τα χαρακτηριστικά της εκπομπής. Η ουδετερότητα του έχει το επιπλέον προσόν ότι το κάνει ιδανικό για να κλείνει αλυσίδες, ιδίως στις περιπτώσεις που θέλουμε να χρησιμοποιήσουμε ένα αλυσιδωτό μοντέλο εντελώς αυτόνομα. Αυτό γίνεται χωρίς την χρησιμοποίηση ενός τελικού μη αλυσιδωτού μοντέλου, και σε αυτό οφείλεται η χρησιμότητα του reliable μοντέλου. Αν πχ θέλουμε να προσθέσουμε μόνο μια τυχαία απόρριψη μηνυμάτων (random_drop chain model) και τίποτα παραπάνω, εισάγουμε ως chain_transm_model το reliable και έχουμε καθαρά random_drop model.

1.1.3.2.2 csma_transmission_model

Το μοντέλο αυτό υλοποιεί το CSMA/CA μοντέλο επικοινωνίας. Ένας επεξεργαστής στέλνει μηνύματα μόνο εφόσον δεν στέλνει κάποιος άλλος ταυτόχρονα και είναι ελεύθερο το κανάλι επικοινωνίας. Αν κάποιος γειτονικός κόμβος στέλνει ήδη κάποιο μήνυμα, ο επεξεργαστής θα περιμένει μέχρι να ολοκληρωθεί η αποστολή συν ένα επιπλέον χρόνο καθυστέρησης για να στείλει το μύνημά του. Οι παράμετροι για αυτό το μοντέλο είναι:

bandwidth (int)	: ο ρυθμός μεταφοράς δεδομένων του μέσου σε bits/sec
backoff (double)	: καθυστέρηση σε sec πριν την αποστολή (σταθερός αριθμός)
sending_jitter	: ορίζει το άνω όριο της τυχαίας διακύμανσης του χρόνου αποστολής
sending_jitter_lb	: ορίζει το κάτω όριο της διακύμανσης του χρόνου αποστολής

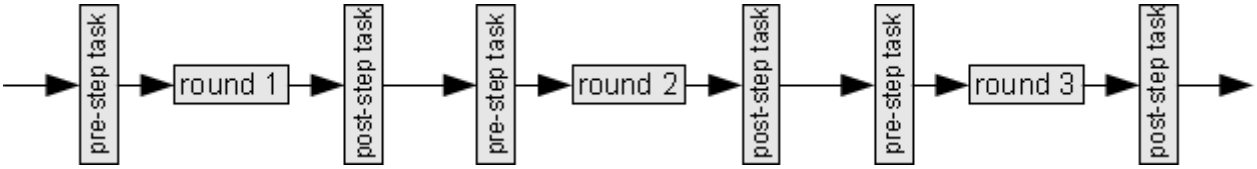
1.2 Προγραμματισμός γεγονότων στο shawn

Στο shawn υπάρχουν δύο τρόποι να προγραμματίσουμε γεγονότα:

1.2.1. Γύροι Προσομοίωσης (Simulation Rounds)

Η πρακτική των γύρων/επαναλήψεων είναι μια απλοϊκή προσέγγιση της εξέλιξης της προσομοίωσης. Σε κάθε γύρο οι κόμβοι μέσω της *work()* μεθόδου μπορούν να αποστέλλουν μηνύματα, και μέσω της *process_message(MessageHandle)* να επεξεργάζονται μηνύματα και τυχόν να αποστέλλουν απαντήσεις κλπ. Είναι σαφές βέβαια ότι όλα αυτά τα γεγονότα γίνονται στους ίδιους χρόνους για όλοι οι κόμβοι. Για παράδειγμα αν σε έναν επεξεργαστή η *work()* μέθοδος περιλαμβάνει εντολή *send(MessageHandle)*, τότε σε κάθε γύρο όλοι οι κόμβοι που έχουν αυτόν τον επεξεργαστή θα εκτελούν στον ίδιο χρόνο την αποστολή μηνύματος.

Ο προσομοιωτής είναι δυνατό να εκτελέσει Simulation Tasks σαν pre-step ή post-step Tasks, πριν και μετά από κάθε επανάληψη. Αυτό δίνει τη δυνατότητα να εξάγουμε δεδομένα από την προσομοίωση χωρίς να απαιτείται η παρεμβολή κώδικα στις άλλες μεθόδους της. Επίσης μπορούμε έτσι να τροποποιούμε είτε με κάποιο script είτε και με χειροκίνητη εισαγωγή διάφορες παραμέτρους του προσομοιωτή [10].



Σχήμα 1.5

1.2.2. Προγραμματιστής Γεγονότων (Event Scheduler)

Σε κάθε γύρο μπορούμε να καταχωρήσουμε γεγονότα σε διάφορες τιμές χρόνου, εντός του ίδιου γύρου. Αυτό βέβαια επιτρέπει ακόμα και την δημιουργία επαναλήψεων εντός των γύρων. Ο Event Scheduler ορίζει χρόνους (timeout time) στους οποίους εκτελούνται κάποια γεγονότα. Όταν η προσομοίωση φτάσει σε χρονικό σημείο που λήγει η προθεσμία για την εκτέλεση ενός γεγονότος, τότε αυτό λαμβάνει χώρα στον καθορισμένο χρόνο, ανεξάρτητα από το στάδιο της προσομοίωσης στο οποίο είμαστε. Η timeout μέθοδος δηλαδή είναι ανεξάρτητη από τις υπόλοιπες κύριες μεθόδους του επεξεργαστή. Ο Event Scheduler είναι δυνατό να κληθεί για χρονισμό γεγονότων σε οποιοδήποτε σημείο της προσομοίωση [10].

1.3 Περιβάλλον Προσομοίωσης (Simulation Environment)

Το περιβάλλον προσομοίωσης είναι ο εικονικός κόσμος στον οποίο βρίσκονται τα αντικείμενα της προσομοίωσης. Αυτά είναι:

1.3.1 Ο Κόσμος (World)

Ο 'Κόσμος' είναι ένα δοχείο (container) που περιέχει τους κόμβους (nodes), τα βασικά Μοντέλα και το μέγεθος της περιοχής που βρίσκονται οι κόμβοι. Διατηρεί στοιχεία της κατάστασης της προσομοίωσης, όπως τον χρόνο και την κατάσταση των κόμβων, και παρέχει πρόσβαση σε αυτούς. Ο 'Κόσμος' διαχειρίζεται όλες τις λειτουργίες της προσομοίωσης [10].

1.3.2 Κόμβοι (Nodes)

Οι κόμβοι έχουν ποικίλες ιδιότητες στην προσομοίωση.

- Έχουν την φυσική ιδιότητα να αντιπροσωπεύουν θέσεις στο δίκτυο
- Αποτελούν δοχεία για τους επεξεργαστές. Κάθε κόμβος μπορεί να περιέχει πολλαπλούς επεξεργαστές
- Αποστέλλουν και λαμβάνουν μηνύματα
- Παρέχουν πρόσβαση στον 'Κόσμο'
- Μόνο ένας κόμβος είναι δυνατό να είναι ειδικός κόμβος (special node)

Η τάξη των κόμβων παρέχει μεθόδους διαχείρισης των επεξεργαστών τους και σωρείας παραμέτρων σχετικές με τη θέση τους, την κατάστασή τους, την αποστολή μηνυμάτων και την διαχείριση των γειτόνων. Μπορεί τα μηνύματα να γεννώνται στους επεξεργαστές και να διαχειρίζονται από αυτούς, η αποστολή τους όμως είναι ευθύνη του κόμβου που τους περιέχει [10].

Όλες οι θέσεις και τα αντικείμενα του δικτύου εκφράζονται με κόμβους. Ανεξάρτητα της εσωτερικής λειτουργίας που υλοποιούν οι επεξεργαστές όλοι οι κόμβοι είναι η ενσάρκωση της ίδιας C++ τάξης.

1.3.3 Επεξεργαστές (Processors)

Στους επεξεργαστές γίνεται ο προγραμματισμός και η εφαρμογή των αλγορίθμων που καθορίζουν την ροή της προσομοίωσης. Σε κάθε κόμβο είναι δυνατό να ενσωματώσουμε πολλούς επεξεργαστές που θα αναλαμβάνουν διαφορετικές εργασίες. Οι επεξεργαστές στέλνουν και επεξεργάζονται μηνύματα.

Ένας επεξεργαστής μπορεί να βρίσκεται σε μια από τις εξής καταστάσεις λειτουργίας [10]:

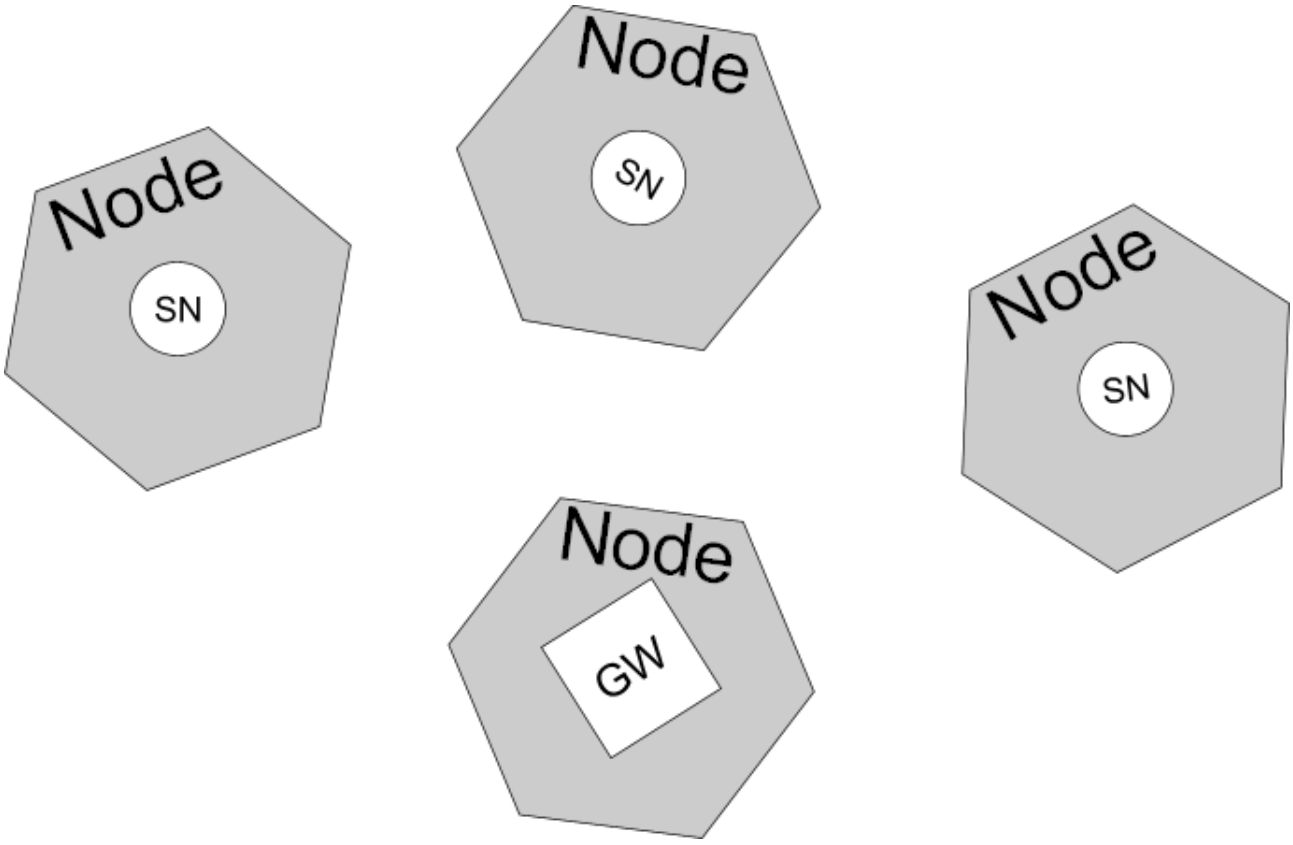
- Ενεργός... πλήρως λειτουργικός
- Σε Καταστολή... δεν λαμβάνει μηνύματα
- Ανενεργός... μπορεί να επανενεργοποιηθεί εξωτερικά

Αν όλοι οι επεξεργαστές σε ένα κόμβο επέλθουν σε ανενεργή κατάσταση, τότε απενεργοποιείται ο κόμβος. Αν όλοι οι κόμβοι είναι ανενεργοί τερματίζεται η προσομοίωση [10].

Σε ένα ασύρματο δίκτυο αισθητήρων συναντάμε κυρίως δύο ειδών κόμβους:

- Αισθητήρες (sensors / SN)
- Πύλες (gateways / GW)

Οι κόμβοι αυτοί δε διαφοροποιούνται σε επίπεδο τάξης κόμβου (Node) αλλά σε επίπεδο επεξεργαστή. Δηλαδή όλοι οι αισθητήρες και πύλες είναι κόμβοι, ως ενσάρκωση της ίδιας C++ τάξης. Καθένας διαχωρίζει το ρόλο και τις ιδιαίτερες λειτουργίες του με τους επεξεργαστές που περιέχει.



Σχήμα 1.6

1.3.3.1 Η Διεπαφή Προγραμματισμού Εφαρμογών (API) του επεξεργαστή

Οι κύριες μέθοδοι που την απαρτίζουν είναι [10]:

void boot()

- Καλείται μία φορά για κάθε επεξεργαστή πριν ξεκινήσει η προσομοίωση. Ουσιαστικά με τη μέθοδο αυτή εκκινεί η εφαρμογή.

void special_boot()

- Καλείται μία φορά για κάθε επεξεργαστή που περιέχεται στον ειδικό κόμβο (scecial node). Αυτή η μέθοδος εκτελείται πριν από την boot().

void work()

- Καλείται στην αρχή κάθε γύρου. Χρησιμοποιείται για να εκτελούνται περιοδικές εργασίες.

bool process_message(MessageHandle)

- Καλείται όταν ο κόμβος λαμβάνει ένα μήνυμα. Το μήνυμα παραδίδεται σε όλους τους επεξεργαστές του κόμβου από το μοντέλο εκπομπής, μέχρι κάποιος να επιστρέψει με τη μέθοδο αυτή “true”.

void send(MessageHandle)

- Με τη μέθοδο αυτή ο επεξεργαστείς αποστέλλει μηνύματα στο δίκτυο.

void set state(Processor::State)

- Ορίζει την κατάσταση λειτουργίας του επεξεργαστή
  - Processor::Active : ενεργός
  - Processor::Sleeping : σε καταστολή
  - Processor::Inactive : ανενεργός

const Node& owner() / Node& owner_w()

- Επιστρέφει τον κόμβο στον οποίο ανήκει ο επεξεργαστής και μας δίνει πρόσβαση στις μεθόδους του.

1.4 Μηνύματα (Messages)

Τα μηνύματα δημιουργούνται από τους επεξεργαστές και αποστέλλονται από τους κόμβους που τους περιέχουν. Η μεταφορά και παράδοση των μηνυμάτων είναι ευθύνη του μοντέλου εκπομπής. Κάθε επεξεργαστής μπορεί να δημιουργεί τα δικά του μηνύματα ως παράγωγες τάξεις της γενικής τάξης 'Message'. Ταυτόχρονα κατά την παραλαβή μηνυμάτων κάθε επεξεργαστής μπορεί να ελέγχει αν το μήνυμα που παρέλαβε είναι του τύπου που επεξεργάζεται [10].

Οι κυριότερες μέθοδοι των μηνυμάτων είναι:

void set_source(Node&)

- Καλείται αυτόματα από τον κόμβο και ενσωματώνει στο μήνυμα το εσωτερικό ID του.

const Node& source(void)

- Μέθοδος για να ανακτήσουμε το εσωτερικό ID του κόμβου που έστειλε το μήνυμα.

virtual void setSize(int)

- Ορισμός του μεγέθους του μηνύματος σε bytes.

virtual int size (void)

- Ανάκτηση του μεγέθους του μηνύματος.

void set_timestamp(int, double)

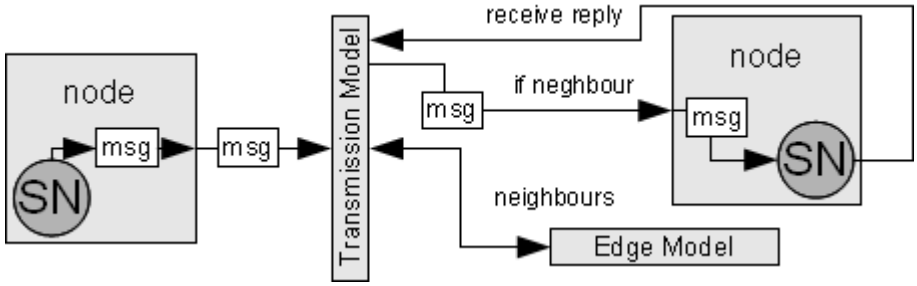
- Ορισμός του χρόνου αποστολής. Καλείται αυτόματα από τον κόμβο.

double timestamp_time(void)

- Ανάκτηση του χρόνου αποστολής.

1.4.1 Διαδικασία Αποστολής Μηνυμάτων

1. Ο επεξεργαστής δημιουργεί ένα μήνυμα και μέσω του κόμβου του το στέλνει στο δίκτυο
2. Ο κόμβος του επεξεργαστή ενσωματώνει στο μήνυμα με τη μέθοδο *set_source(Node&)* το ID του, και με τη μέθοδο *set_timestamp(int, double)* την χρονική στιγμή αποστολής του μηνύματος. Στη συνέχεια παραδίδει το μήνυμα στο μοντέλο εκπομπής.
3. Το μοντέλο εκπομπής (Transmission Model) δημιουργεί για κάθε πιθανό παραλήπτη του μηνύματος αντίγραφα. Παραδίδει τα αντίγραφα αυτά στους γειτονικούς κόμβους που μπορούν να το παραλάβουν. Αυτούς τους εντοπίζει χάρη στο μοντέλο ακμών, με το οποίο στέλνει ερωτήματα στο μοντέλο επικοινωνίας.
4. Σε κάθε κόμβο το μήνυμα παραδίδεται σε όλους τους επεξεργαστές του.
5. Κάθε επεξεργαστής ελέγχει αν το μήνυμα είναι του τύπου που επεξεργάζεται και αναλόγως εκτελεί τους αλγορίθμους του.
6. Αν κανένας επεξεργαστής δεν επεξεργαστεί το μήνυμα, επιστρέφοντας “false” στο μοντέλο εκπομπής, τότε το μήνυμα απορρίπτεται (dropped).



Σχήμα 1.7

1.5 Οπτικοποίηση (Visualization)

Το Shawn διαθέτει εφαρμογή οπτικοποίησης του δικτύου. Η εφαρμογή αυτή αποτυπώνει το δίκτυο όπως προκύπτει από το μοντέλο ακμών . Το μοντέλο επικοινωνίας ενημερώνει το μοντέλο ακμών για τους κόμβους που έχουν δυνατότητα επικοινωνίας. Δεν εξετάζεται πως εφαρμόζεται η μεταφορά των μηνυμάτων με το μοντέλο εκπομπής. Το μοντέλο ακμών για κάθε σύνδεση μεταξύ δύο κόμβων δημιουργεί μια 'ακμή' (edge). Οι εργασίες οπτικοποίησης (visualization tasks) δημιουργούν μια γραφική απεικόνισης της διάταξη του δικτύου και των ακμών που συνδέουν τους γειτονικούς κόμβους, δηλαδή το πόρισμα του μοντέλου ακμών. Συνολικά στο γράφημα απεικονίζονται οι κόμβοι στις σχετικές τους θέσεις και οι ακμές που συνδέουν αυτούς που έχουν επικοινωνία [11].

Η εφαρμογή οπτικοποίησης του Shawn κάνει χρήση της βιβλιοθήκης δισδιάστατων διανυσματικών γραφικών Cairo (2D vector graphics library Cairo). Είναι δυνατό να εξάγει το προϊόν σε διάφορα format όπως pdf, png, ps (postscript). Αποτελείται δε ουσιαστικά από πολλές εργασίες (tasks) οι οποίες εκτελούν τα διάφορα στάδια της συγκρότησης της τελικής παράστασης του δικτύου. Φυσικά είναι δυνατό να προσθέσει κανείς επιπλέον εργασίες ή και παραμέτρους στις ήδη υπάρχουσες, προκείμενου να τροποποιήσει την οπτικοποίηση. Σε αυτό πρέπει να συμπεριλάβουμε και το σύνολο των δυνατοτήτων του Cairo, καθώς δεν αξιοποιείται σε πλήρη έκταση (πχ δεν αξιοποιείται η πολλαπλότητα των επιφανειών και δεν δίνεται προτεραιότητα σε στοιχεία της απεικόνισης κατά την περίπτωση επικάλυψης)

1.6 Εργασίες (common tasks)

Οι εργασίες είναι τάξεις που εκτελούνται είτε στην εκκίνηση της προσομοίωσης, είτε ανάμεσα σε γύρους, είτε στο τέλος αυτής. Είναι δυνατό να εκτελούμε εργασίες χωρίς να τρέξουμε γύρους προσομοίωσης. Η δόμηση του περιβάλλοντος προσομοίωσης γίνεται επίσης με εργασίες. Παρουσιάζονται οι κυριότερες εργασίες του shawn [21] :

- **add_poststep**

Ορίζει σε μια εργασία να εκτελείται μετά από κάθε γύρο προσομοίωσης. Δέχεται ως παράμετρο το όνομα της εργασίας

- **add_prestep**

Ορίζει σε μια εργασία να εκτελείται πριν από κάθε γύρο προσομοίωσης. Δέχεται ως παράμετρο το όνομα της εργασίας

- **chain_transm_model**

Εφόσον έχουμε ορίσει αλυσιδωτό μοντέλο εκπομπής, με τη μέθοδο αυτή προσθέτουμε άλλα μοντέλα στην αλυσίδα.

- connectivity

Με την μέθοδο αυτή μας επιστρέφονται στατιστικά για την συνδεσιμότητα μεταξύ των κόμβων του δικτύου. Συγκεκριμένα υπολογίζεται η μέση, η ελάχιστη και η μέγιστη τιμή της. Στην ουσία η τιμές αυτές αναφέρουν τον αριθμό των γειτονικών κόμβων. Οι *min* και *max* είναι ακέραιοι αριθμοί ενώ η *mean* είναι δεκαδικός. Ο εντοπισμός των συνδέσεων γίνεται με κλήση της μεθόδου *nof_adjacent_nodes(const Node&)*. Φυσικά η μέθοδος υλοποιείται διαφορετικά σε κάθε edge model, με εξαίρεση το fast list που χρησιμοποιεί την υλοποίηση των grid και list μοντέλων ακμών.

- **degree_sequence**

Η εργασία αυτή απαιτεί παράμετρο ds_out_file=$ds_out_file (string) στην οποία ορίζεται αρχείο εγγραφής εξόδου, και στο οποίο εγγράφονται σε μορφή κειμένου τα εξής στοιχεία:

Node.real_position().x()
Η θέση του κόμβου κατά x

Node.real_position().y()
Η θέση του κόμβου κατά y

Node.degree()
Ο αριθμός των γειτόνων ενός κόμβου

Node.id()
Το εσωτερικό id του κόμβου (μοναδικό για κάθε κόμβο)

Node.label()
Το label του κόμβου (εξωτερικό id)

Τα ανωτέρω στοιχεία δίνονται σε μορφή πίνακα με κάθε γραμμή να αντιστοιχεί σε ένα κόμβο, και αποδίδονται με αύξουσα σειρά Node.id().

- **dump_location**

Εργασία που εκτυπώνει σε αρχείο τις εκτιμημένες ή πραγματικές θέσεις των κόμβων. Η εντολή δέχεται μία υποχρεωτική και έως τέσσερις προαιρετικές παραμέτρους. Η υποχρεωτική παράμετρος είναι το αρχείο εξόδου (dloc_out_file=$dloc_out_file (string)). Οι προαιρετικές είναι:

- dloc_print_estimates (int):
Η προεπιλογή και η τιμή 1 εκτυπώνουν τις θέσεις των κόμβων κατ' εκτίμηση. Αν δεν έχουν εκτιμημένες θέσεις οι κόμβοι, τότε εκτυπώνεται η τιμή της παραμέτρου 'dloc_unpositioned_mark', ώστε να σημάνουμε τους μη τοποθετημένους κόμβους. Δίνοντας τιμή 0 (μηδέν) στην παράμετρο αυτή οδηγούμε την εργασία να εκτυπώσει την πραγματική θέση των κόμβων (*Node.real_position()*).

- dloc_skip_unpositioned (int):
Η παράμετρος αυτή έχει προεπιλογή τιμή 0 και για τιμή διάφορη του μηδενός ορίζει την παράβλεψη των μη τοποθετημένων κόμβων στην εκτύπωση.

- dloc_col_sep (string):
Η παράμετρος αυτή ορίζει το διαχωριστικό ανάμεσα στις συντεταγμένες θέσης. Προεπιλογή είναι το tab ("\t").

- dloc_unpositioned_mark (string):
Η παράμετρος αυτή ορίζει την σήμανση για τους μη τοποθετημένους κόμβους στην εκτύπωση εκτιμημένων θέσεων.

- **dump_transmission_stats**

Για να εκτελεσθεί αυτή η εργασία πρέπει να έχουμε φορτώσει στην αλυσίδα του μοντέλου εκπομπής το stats_chain μοντέλο. Η εντολή δεν δέχεται επιπλέον παραμέτρους. Μετά το πέρας της αποστολής όλων των μηνυμάτων με τις διάφορες μεθόδους, η εργασία αυτή εξάγει στατιστικά για τα μηνύματα, στα οποία αναφέρεται το συνολικό πλήθος και μέγεθός τους, καθώς και τα αναλυτικά μεγέθη ανάλογα με το είδος τους. Αν δηλαδή. έχουμε δύο διαφορετικούς επεξεργαστές οι οποίοι παράγουν διαφορετικού είδους μηνύματα που αποστέλλονται στο δίκτυο (πχ a_message και b_message), τότε θα έχουμε αναλυτικά στατιστικά για κάθε τύπο μηνύματος.

- **load_world**

Με την εργασία αυτή εισάγουμε στον κόσμο που έχουμε δημιουργήσει για την προσομοίωση την τοπολογία του δικτύου από αρχείο δεδομένων XML. Η εργασία αυτή πρέπει να εκτελεσθεί αφού έχει δημιουργηθεί ο κόσμος με την εντολή prepare_world και έχει οριστεί το μοντέλο εκπομπής (Transmission Model). Στην ουσία μας δίνεται η δυνατότητα είτε να επαναφέρουμε μια τοπολογία που έχουμε αποθηκεύσει με τη μέθοδο save_world είτε να φορτώσουμε στην προσομοίωση δεδομένα θέσης που έχουμε από άλλη πηγή και έχουμε προσαρμόσει σε XML format όπως το απαιτεί η μέθοδος.

Η σύνταξη της εντολής είναι η εξής:

```
load_world world_in_file=$PATH/$TOPOLOGY_FILE.xml
snapshot=$SNAPSHOT
```

Η διαδρομή προς το αρχείο μπορεί να είναι απόλυτη ή σχετική. Για σχετική διαδρομή πρόκειται για τη διαδρομή από το shawn εκτελέσιμο και όχι από κάποιο αρχείο ρυθμίσεων conf, σε περίπτωση που χρησιμοποιούμε για να εκτελέσουμε την προσομοίωση.

Ένας κόμβος ορίζεται σε ένα shawn XML αρχείο τοπολογίας ως εξής:

```
<scenario>
<snapshot id="01">
.
.
.
<node id="GateWay">
     <location x="482643.800" y="4201487.850" z="352.672" />
</node>
.
.
.
</scenario>
</snapshot>
```

- **prepare_world**

Πρόκειται για την πιο σύνθετη εργασία του shawn. Με αυτήν δημιουργείται ο κόσμος της προσομοίωσης και ορίζονται τα μοντέλα επικοινωνίας (Comm) και ακμών (Edge), καθώς και οι παράμετροί τους. Η εντολή για την εκτέλεση της εργασίας είναι:

```
prepare_world edge_model=$EDGE_MODEL comm_model=$COMM_MODEL
$COMM_MODEL_PARAMETER1="value"
$COMM_MODEL_PARAMETER2="value" ...
```

Ο ορισμός του μοντέλου ακμών είναι υποχρεωτικός, ενώ για το μοντέλο επικοινωνίας είναι προαιρετικός (προεπιλογή σε 'Unit Disk Graph').

Οι τιμές που αντιστοιχούν στα υπάρχοντα μοντέλα ακμών είναι:

fast_list - Fast List Edge Model
grid - Grid Edge Model
simple - Lazy Edge Model
list - List Edge Model

Το μόνο από αυτά που δέχεται επιπλέον παραμέτρους είναι το Grid Edge Model. Και οι τρεις παράμετροι που δέχεται είναι προαιρετικές και παρατίθενται μαζί με τις προεπιλεγμένες τιμές τους:

grid_size 8 (int)
grid_cell_size 1.5 (double)
grid_closeness_fraction 0.10 (double)

Οι τιμές για τα μοντέλα επικοινωνίας είναι:

disk_graph - Disk Graph Comm Model
link_probability - Link Probability Comm Model
manual - Manual Comm Model
multiple - Multiple Comm Model
permalink - Permanent Link Comm Model
rim - RIM Comm Model
stochastic - Stochastic Comm Model

Οι παραπάνω ονομαστικές τιμές προκύπτουν από τη μέθοδο '*virtual std::string name(void)*' των παραγωγικών τάξεων των μοντέλων (model factories).

Σε περίπτωση που επιλέξουμε Multiple Comm Model πρέπει να ορίσουμε υποχρεωτικά μια λίστα με όλα τα μοντέλα επικοινωνίας που θα χρησιμοποιήσουμε. Επίσης μπορούμε προαιρετικά να θέσουμε κάποιο μοντέλο ως προεπιλογή, το οποίο θα εφαρμοστεί εφόσον τα μοντέλα της λίστας δεν μπορούν να υλοποιήσουν επικοινωνία ανάμεσα στους κόμβους. Οι παράμετροι και η σύνταξη της δήλωσής τους είναι:

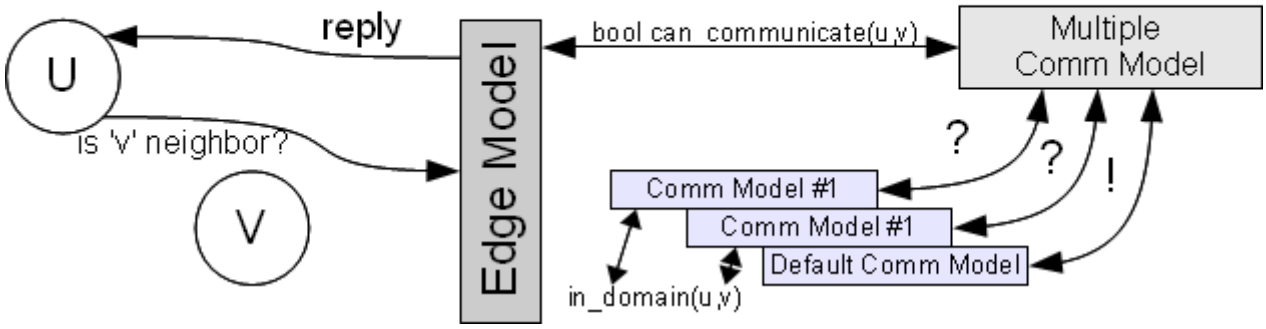
```
comm_models=$COMM_MODEL1,$COMM_MODEL2,.....
(τα μοντέλα χωρίζονται με κόμμα)

comm_default_model=$COMM_DEFAULT_MODEL
```

Το Multiple μοντέλο επικοινωνίας εφαρμόζεται ως εξής:

- Για κάθε μοντέλο της λίστας εξετάζεται αν μπορεί να δώσει απάντηση στο ερώτημα αν δύο κόμβοι 'u' και 'v' μπορούν να επικοινωνήσουν. Δηλαδή δεν εξετάζεται αν επικοινωνούν ή όχι οι κόμβοι, αλλά αν μπορούν να δώσουν απάντηση οι μέθοδοι *can_communicate_uni()* και *can_communicate_bidi()*. Αυτό γίνεται με την κλήση της '*virtual bool in_domain(const Node& u, const Node& v)*', όπως υλοποιείται σε κάθε μοντέλο. Η προεπιλεγμένη ρύθμιση επιστρέφει απάντηση “true”, δηλαδή ουσιαστικά δεν εξετάζει τίποτα δίνοντας το πράσινο φως για το επόμενο βήμα, και εφαρμόζεται στα Unit Disk Graph, RIM, Stochastic, Manual και Multiple μοντέλα επικοινωνίας. Τα μόνα μοντέλα που αποκλίνουν της προεπιλογής είναι τα Link Probability και Permanent Link. Και στα δύο η in_domain() λειτουργία "μεταφράζεται" στη μέθοδο '*bool can_communicate_uni(const Node& u, const Node& v)*' του εκάστοτε μοντέλου (διότι προφανώς σε κάθε μέθοδο υλοποιείται/δύναται να υλοποιηθεί διαφορετικά).

- Εφόσον κάποιο από τα μοντέλα της λίστας με τη σειρά που τα έχουμε δηλώσει, επιτρέπει την εξέταση της επικοινωνίας ανάμεσα στους κόμβους, τότε γίνεται το ερώτημα επ' αυτής και επιστρέφεται η απάντηση στο Multiple μοντέλο επικοινωνίας και στην προσομοίωση. Συνεπώς για τους συγκεκριμένους κόμβους 'u' και 'v' η επικοινωνία υλοποιείται βάσει του πρώτου μοντέλου της λίστας που μπορεί να δώσει απάντηση.

- Αν κανένα από τα μοντέλα της λίστας δεν μπορεί να ικανοποιήσει το ερώτημα για επικοινωνία, τότε εξετάζεται το comm_default_model. Εδώ δεν εξετάζεται αν μπορεί να απαντήσει το μοντέλο, αλλά γίνεται ευθέως το ερώτημα για την ύπαρξη ή μη επικοινωνίας ανάμεσα στους κόμβους.



Σχήμα 1.8

Όπως γίνεται κατανοητό αν σε μια λίστα πχ με 3 μοντέλα και μία προεπιλογή, δίνουν απάντηση όλα και λόγου χάρη με το 1ο στην λίστα οι κόμβοι δεν επικοινωνούν ενώ με τα υπόλοιπα επικοινωνούν, η απάντηση του Multiple μοντέλου θα είναι ότι δεν επικοινωνούν, καθότι αφού θα πάρει απάντηση από το 1ο μοντέλο της λίστας δεν θα εξετάσει τα υπόλοιπα. Φυσικά είναι δυνατό να τροποποιηθεί η πολιτική αυτή ώστε να επιλέξουμε μοντέλο από την λίστα με διαφορετικά κριτήρια από την θέση του σε αυτή.

- **random_seed**

Πολλές παράμετροι στο shawn ρυθμίζονται με τυχαίες μεταβλητές, προσομοιώνοντας έτσι την τυχαιότητα και αβεβαιότητα των γεγονότων, όπως αυτές παρουσιάζονται στον πραγματικό κόσμο. Τις τιμές των τυχαίων αυτών μεταβλητών παράγουν "Γεννήτριες Τυχαίων Μεταβλητών" (Random Variable Generators). Κάθε φορά που εκτελούμε ένα σενάριο που παραμετροποιείται από τυχαίες μεταβλητές έχουμε διαφορετικά αποτελέσματα. Προκειμένου να έχουμε ακριβώς την ίδια συμπεριφορά σε δύο εκτελέσεις της ίδιας προσομοίωσης πρέπει να χρησιμοποιήσουμε τις ίδιες μεταβλητές. Αυτό επιτυγχάνεται με το random_seed task. Η εντολή αυτή έχει τρεις διαφορετικές υλοποιήσεις:

(1) random_seed action=create filename=$filename

Με την εντολή να ακολουθείται από την δράση 'δημιούργησε' (action=create) και το όνομα αρχείου $filename, η εργασία δημιουργεί το αρχείο που ορίσαμε στην εντολή στον τρέχοντα φάκελο και σε αυτό αποθηκεύει ένα τυχαίο seed που δημιουργείται από το χρόνο του συστήματος. Το seed αυτό όχι μόνο αποθηκεύεται, αλλά και ενεργοποιείται για την τρέχουσα προσομοίωση.

(2) random_seed action=load filename=$filename

Όταν η δράση είναι 'φόρτωσε' (action=load) ακολουθούμενη από το όνομα αρχείου που έχουμε αποθηκεύσει κάποιο seed, τότε ο προσομοιωτής φορτώνει το αποθηκευμένο seed και το χρησιμοποιεί στις γεννήτριες τυχαίων μεταβλητών.

(3) random_seed action=set seed=$seed

Με την δράση 'θέσε' (action=set) ορίζουμε ευθέως το seed που θέλουμε να χρησιμοποιούμε στην προσομοίωση μέσα στο αρχείο ρυθμίσεων. Το seed είναι μεγάλος ακέραιος αριθμός (long int).

Προκειμένου να αξιοποιήσουμε το εκάστοτε seed, είτε το δημιουργούμε κατά την προσομοίωση, είτε το φορτώνουμε από κάποιο αρχείο, είτε το θέτουμε ευθέως, πρέπει να εκτελέσουμε την εργασία random_seed πριν από το πρώτο task που θέλουμε να αξιοποιήσει τυχαίες μεταβλητές που θα οριστούν από το seed αυτό.

- **rect_world**

Με την εργασία αυτή, και αφού έχουμε ορίσει το περιβάλλον προσομοίωσης με την εντολή prepare_world, δημιουργούμε ένα ορθογώνιο παραλληλόγραμμο το οποίο θα γεμίσουμε με τυχαίους κόμβους. Στην περίπτωση αυτή απαιτούνται τρεις παράμετροι:

width - πλάτος του εικονικού κόσμου της προσομοίωσης (int)
height - ύψος του εικονικού κόσμου της προσομοίωσης (int)
count - αριθμός των nodes που θα τοποθετηθούν τυχαία στον κόσμο που ορίζουν οι παραπάνω διαστάσεις (int)

Είναι προφανές ότι ο κόσμος που δημιουργείται με τη μέθοδο αυτή είναι δισδιάστατος (2d) και όλοι οι κόμβοι βρίσκονται στο ίδιο επίπεδο. Αν θέλουμε τρισδιάστατο κόσμο πρέπει να χρησιμοποιήσουμε την αντίστοιχη εργασία cuboid_world. Φυσικά αυτή η μέθοδος δημιουργίας κόμβων δεν έχει νόημα όταν φορτώνουμε τοπολογία από κάποιο XML αρχείο.

Συνήθως όταν δημιουργούμε την τοπολογία του δικτύου με κάποια από τις εργασίες rect_world, cuboid_world, flegsensworld, load_world δηλώνουμε και τους προεπιλεγμένους επεξεργαστές που θα έχουν οι κόμβοι με την παράμετρο processors. Αν θέλουμε περισσότερους από έναν default processors τους διαχωρίζουμε με κόμμα.

- **reset_movement**

Η εντολή αυτή εκτελείται συνήθως στα πλαίσια σεναρίου με κινούμενους κόμβους. Καλείται η *reset()* μέθοδος του κόσμου, η οποία επαναφέρει τον κόσμο στην αρχική του κατάσταση χωρίς να αφαιρεί κόμβους. Ωστόσο αν κατά τη διάρκεια της προσομοίωσης έχουμε πχ προσθέσει επεξεργαστές στους κόμβους, μετά το reset αυτοί διατηρούνται και οι κόμβοι συμπεριφέρονται σαν να τους είχαν ενσωματωμένους από την αρχή της προσομοίωσης.

- **save_world**

Εντολή που αποθηκεύει την κατάσταση του κόσμου και τα nodes σε xml αρχείο

- **show_distance_estimates**

Εκτυπώνει τις καταχωρημένες μεθόδους εκτίμησης αποστάσεων ανάμεσα σε nodes.

- **show_edge_models**

Εντολή που εκτυπώνει στην οθόνη όλα τα καταχωρημένα μοντέλα Ακμών (Edge Models).

- **show_processors**

Εντολή που εκτυπώνει στην οθόνη όλους τους καταχωρημένους επεξεργαστές.

- **show_random_variables**

Εντολή που εκτυπώνει στην οθόνη όλες τις καταχωρημένες τυχαίες μεταβλητές.

- **show_tasks**

Εντολή που εκτυπώνει στην οθόνη όλες τις εργασίες προσομοίωσης που έχουν ενσωματωθεί στο εκτελέσιμο του shawn. Επιδέχεται προαιρετική παράμετρο "verbose" (int). Με απλή δήλωση και με verbose=0 απλά τυπώνει τα ονόματα των tasks. Με verbose=1 τυπώνει και την περιγραφή τους αν έχουν.

- **simulation**

Εκτελεί την προσομοίωση. Απαιτείται να έχει οριστεί ο κόσμος με τις αντίστοιχες εργασίες, και προαιρετικά ο αριθμός των γύρων που θα διαρκέσει η προσομοίωση. Αυτός αποδίδεται με την ακέραιη παράμετρο max_iterations (int) (μέγιστος αριθμός επαναλήψεων).

- **tagtest**

Εκτυπώνει τις ετικέτες που έχουμε φορτώσει στο περιβάλλον προσομοίωσης και στους κόμβους, καθώς και το είδος τους.

- **vis tasks**

Για την οπτικοποίηση της προσομοίωσης υπάρχει πληθώρα εργασιών. Παρουσιάζονται αυτές που αξιοποιούνται στο κεφάλαιο του αρχείου ρυθμίσεων [§1.7.2].

1.7 Αρχείο Ρυθμίσεων (config file)

Υπάρχουν δύο τρόποι να εκτελέσουμε προσομοιώσεις στο shawn:

- Εκτέλεση με είσοδο οδηγιών και παραμέτρων στη γραμμή εντολών (interactive input)

Όταν εκτελείται το shawn χωρίς παραμέτρους σε κάποιο κέλυφος, μας εισάγει σε γραμμή εντολών, αναμένοντας εισαγωγή (input). Σε αυτή τη κατάσταση το shawn δεν έχει πραγματοποιήσει ακόμα κανένα στάδιο της προσομοίωσης και αναμένει από τον χρήστη να ορίσει ποιες εργασίες πρέπει να εκτελέσει (common tasks), καθώς και τις διάφορες παραμέτρους της προσομοίωσης (global variables).

- Εκτέλεση με αρχείο ρυθμίσεων Config File

Το shawn είναι δυνατό να εκτελέσει προσομοιώσεις ανακτώντας όλες τις εργασίες και τις παραμέτρους του σεναρίου από αρχεία ρυθμίσεων "*.conf" (config files). Τα αρχεία αυτά είναι αρχεία κειμένου και περιέχουν σε σειρά τις ίδιες ακριβώς εντολές που θα δίναμε στον προσομοιωτή από την γραμμή εντολών. Είναι ένας πολύ βολικός τρόπος να τρέχουμε πολύπλοκες προσομοιώσεις ταχύτατα χωρίς να χρειάζεται κάθε φορά να εισάγουμε όλες τις ρυθμίσεις χειροκίνητα. Μπορούμε επίσης να διατηρούμε διαφορετικές ρυθμίσεις του ίδιου σεναρίου σε διαφορετικά αρχεία.

Το αρχείο εκτελείται γραμμή γραμμή, παρακάμπτοντας όσες εκκινούν με "#". Οι γραμμές αυτές θεωρούνται σχόλια και είναι πολύ χρήσιμες για την τεκμηρίωση των εντολών και των παραμέτρων του αρχείου ρυθμίσεων. Επίσης μπορούμε να διατηρούμε αρκετές εκδόσεις ρυθμίσεων στο ίδιο αρχείο και να έχουμε απενεργοποιημένες όσες δεν χρησιμοποιούμε με το σύμβολο "#" του σχολίου. ΔΕΝ αντιμετωπίζεται σαν αρχή σχολίου το σύμβολο "#" αν δεν είναι στην αρχή μας γραμμής.

Ακολουθεί η παρουσίαση ενός conf αρχείου (τα σχόλια διατηρούνται για καλύτερη κατανόηση):

1.7.1 Γενικές ρυθμίσεις

prepare_world edge_model=simple comm_model=rim range=100 doi=0.5

- Η πρώτη εντολή που αφορά την δόμηση του

transm_model=stats_chain
chain_transm_model name=reliable

- Αμέσως μετά πρέπει να ορίσουμε το μοντέλο εκπομπής (Transmission Model). Αν επιθυμούμε να χρησιμοποιήσουμε πολλαπλά μοντέλα σε αλυσίδα πρέπει να ορίσουμε αλυσιδωτό (chainable) transm_model και με την εντολή chain_transm_model να προσθέσουμε διαδοχικά τα υπόλοιπα μοντέλα. Μόνο το τελευταίο στην αλυσίδα μπορεί να μην είναι αλυσιδωτό (non-chainable).

load_world world_in_file=../src/legacyapps/simple_app/shawn-kaisariani_xyz.xml snapshot=01

- Φορτώνουμε στο περιβάλλον της προσομοίωσης τους κόμβους όπως έχουν αποθηκευθεί σε XML αρχείο.

processors=simple_app

- Αφού έχουμε γεμίσει τον εικονικό κόσμο της προσομοίωσης με nodes, είτε με κάποια γεννήτρια είτε από αρχείο με δεδομένα θέσης, ορίζουμε τους προεπιλεγμένους επεξεργαστές που θα έχουν όλα αυτά τα nodes. Αν ορίσουμε περισσότερους από έναν πρέπει να τους χωρίσουμε με κόμμα.

simulation max_iterations=4

- Για να τρέξει εν τέλει το σενάριο και να εφαρμοστούν οι λειτουργίες των επεξεργαστών μένει να δηλώσουμε τον αριθμό των γύρων/επαναλήψεων που θα διαρκέσει η προσομοίωση. Προσοχή χρειάζεται να μη δηλώσουμε μικρότερο αριθμό από όσο χρειάζεται να εκτελεστούν όλες οι απαιτούμενες λειτουργίες, αφού είναι συχνό φαινόμενο και πρακτική να χρονολογούνται διεργασίες σε μεταγενέστερα στάδια της προσομοίωσης.

connectivity

- Εκτελεί το connectivity task. Δεν είναι απαραίτητο να έχει ξεκινήσει η προσομοίωση. Αρκεί να έχει ολοκληρωθεί η δημιουργία του περιβάλλοντός της και να έχουν δημιουργηθεί/φορτωθεί τα nodes.

dump_transmission_stats

- Εργασία που εκτυπώνει τα στατιστικά του stats_chain Transmission Model. Πρέπει να εκτελείται μετά το πέρας της προσομοίωσης και αφού έχουν αποσταλεί τα μηνύματα που ανταλλάσσουν μεταξύ τους τα nodes, καθότι πάνω σε αυτά αναφέρει.

1.7.2 Ρυθμίσεις Οπτικοποίησης

#Name of visualization
vis=simple_app_vis

- Ονομασία της οπτικοποίησης

#Create visualization
vis_create

- Δημιουργία της οπτικοποίησης

#Group nodes and assign by tag
vis_create_group group=Gateways

- Δημιουργούμε μια ομάδα απεικόνισης για τις Πύλες (Gateways)

vis_create_group group=sensors

- Δημιουργούμε μια ομάδα απεικόνισης για τους Αισθητήρες (Sensors)

*vis_group_add_by_tag group=gateways tag=node_type tag_regex=.*gateway*
*vis_group_add_by_tag group=sensors tag=node_type tag_regex=.*sensor*

- Αναλόγως της τιμής της ετικέτας node_type κάθε κόμβου, τον αντιστοιχούμε στην ομάδα που ανήκει.

#Create edges
vis_create_edges

- Δημιουργούμε ακμές ανάμεσα σε κάθε κόμβο και τους γείτονες του, όπως αυτοί προκύπτουν από τα μοντέλα επικοινωνίας και ακμών της προσομοίωσης.

For color properties COLOR x y z stand for RED GREEN BLUE -> xyz = RGB
values in xyz are between 0 and 1

1.7.2.1 Ρυθμίσεις Οπτικοποίησης Κόμβων

Node properties

#Set border color:
vis_constant_vec x=0 y=0 z=0 elem_regex=node.default. prop=foreground prio=1*

- Ορίζει το χρώμα του περιγράμματος ενός κόμβου. Ισχύει για όλους τους κόμβους, είτε είναι Gateways είτε Sensors.

#Set fill color:
#vis_constant_vec x=1 y=1 z=0 elem_regex=node.default.* prop=background prio=1
vis_constant_vec x=1 y=0 z=1 elem=Gateways prop=background prio=2
vis_constant_vec x=0 y=1 z=1 elem=Sensors prop=background prio=1

- Ορίζει το χρώμα ενός κόμβου. Εδώ υλοποιούμε διαφορετικά το χρώμα για τα Gateways και τα Sensors, αξιοποιώντας τις ομάδες που δημιουργήσαμε προηγουμένως για κάθε τύπο κόμβου.

#Set the size of a node:
#vis_constant_double value=2 elem_regex=node.default.* prop=size prio=1
vis_constant_double value=3 elem=Stations prop=size prio=2
vis_constant_double value=1 elem=Nodes prop=size prio=1

- Ορίζει το μέγεθος ενός κόμβου. Εδώ ορίζουμε το Gateway να είναι μεγαλύτερο από τα Sensors, όπως εύκολα φαίνεται.

#Set the shape of a node (1=circle, 2-quadrat):
#vis_constant_int value=2 elem_regex=node.default.* prop=shape prio=1
vis_constant_int value=1 elem=Stations prop=shape prio=2
vis_constant_int value=2 elem=Nodes prop=shape prio=1

- Ορίζει το σχήμα ενός κόμβου. Εδώ ορίζουμε τα Gateways να παρίστανται με κύκλο και τα Sensors με τετράγωνο.

#Set node blend value
#vis_constant_double value=0.5 elem_regex=node.default. prop=blend prio=1*

- Ορίζει την ανάμιξη του χρώματος των κόμβων με το χρώμα του υπόβαθρου της κάμερας.

1.7.2.2 Ρυθμίσεις Οπτικοποίησης Ακμών

Edge properties

#Set edge color:

vis_constant_vec x=0 y=0 z=0 elem_regex=edge.default. prop=color prio=1*

- Ορίζει το χρώμα των ακμών.

#Set Line-with:
vis_constant_double value=0.05 elem_regex=edge.default. prop=line_width prio=1*

- Ορίζει το πάχος των ακμών.

#Set blend property:
vis_constant_double value=0.5 elem_regex=edge.default. prop=blend prio=1*

- Ορίζει την τιμή της ανάμιξης του χρώματος των ακμών με το χρώμα του υπόβαθρου της κάμερας.

1.7.2.3 Ρυθμίσεις Οπτικοποίησης Κάμερας

CAMERA setup AUTO

#Change Image Resolution (width and height properties of the camera have to be set before calling vis_simple_camera):
vis_constant_double value=1024 elem=cam prio=0 prop=width
vis_constant_double value=800 elem=cam prio=0 prop=height

- Ορίζουν το μέγεθος της εικόνας της οπτικοποίησης. Πρέπει να γίνει πριν την κλήση της εργασίας vis_simple_camera

#Set simple_camera to adjust all other camera properties
vis_simple_camera

- Καλεί την απλή κάμερα. Δεν χρειάζεται να ορίσουμε επιπλέον ρυθμίσεις πλην των διαστάσεων της εικόνας που έχει προηγηθεί. Όλες οι υπόλοιπες παράμετροι ορίζονται αυτόματα

CAMERA setup MANUAL

#Position of the camera in 3D space, (set in WORLD coordinates -> NOT PIXEL!!!):
#vis_constant_vec x=12.5 y=12.5 z=0 elem_regex=cam prop=position prio=1

- Ορίζει την θέση της κάμερας στον τρισδιάστατο χώρο. Οι συντεταγμένες πρέπει να δίνονται στο σύστημα αναφοράς του κόσμου και όχι σε εικονοστοιχεία (pixels).

#Position_shift allows to move the camera horizontally and/or vertically taking PIXEL values:
#vis_constant_vec x=0 y=0 z=0 elem_regex=cam prop=position_shift prio=1

- Ορίζει μετακίνηση της κάμερας κατά τις τρεις διευθύνσεις x,y,z με τιμές σε pixel.

#Set the background color (RGB values):
#vis_constant_vec x=1 y=1 z=1 elem_regex=cam prop=background prio=1

- Ορίζει το χρώμα του υποβάθρου.

#Define Horizontal Resolution:
#vis_constant_double value=1000 elem_regex=cam prop=width prio=1

- Ορίζει το πλάτος της εικόνας (x)

#Define Vertical Resolution:
#vis_constant_double value=1000 elem_regex=cam prop=height prio=1

- Ορίζει το ύψος της εικόνας (y)

#Scale factor:
#vis_constant_double value=38 elem_regex=cam prop=scale prio=1

- Ορίζει την κλίμακα της απεικόνισης

1.7.3 Τελικές Ρυθμίσεις Οπτικοποίησης και εκτύπωση

l*abel_font_size=2*
vis_create_label

- Ορισμός μεγέθους γραμματοσειράς και εκτύπωση σε αυτό το μέγεθος ετικέτας με τον αριθμό του κόμβου

#pdf output
vis_single_snapshot

- Εκτύπωση σε pdf αρχείο της οπτικοποίησης. Το αρχείο θα έχει όνομα snapshot.pdf.

OMNeT++

ΑΝΑΛΥΣΗ:

Το Omnet++ είναι ένα framework δημιουργίας προσομοιωτών δικτύων, βασισμένο στην C++. Ακολουθεί την λογική δημιουργίας αυτόνομων λειτουργικών μονάδων (modular approach) για την αναπαράσταση των στοιχείων του δικτύου, και την εσωτερική θεμελίωση εντός αυτών όλων των παραμέτρων και μεθόδων που υλοποιούν την προσομοίωση. Το Omnet++ δεν περιορίζεται σε ένα τύπο/μοντέλο δικτύου, αλλά είναι ικανό να προσομοιώσει παντός τύπου δίκτυα, ενσύρματα / ασύρματα δίκτυα επικοινωνίας, δίκτυα μεταφορών, δρομολογητές, token rings κλπ.

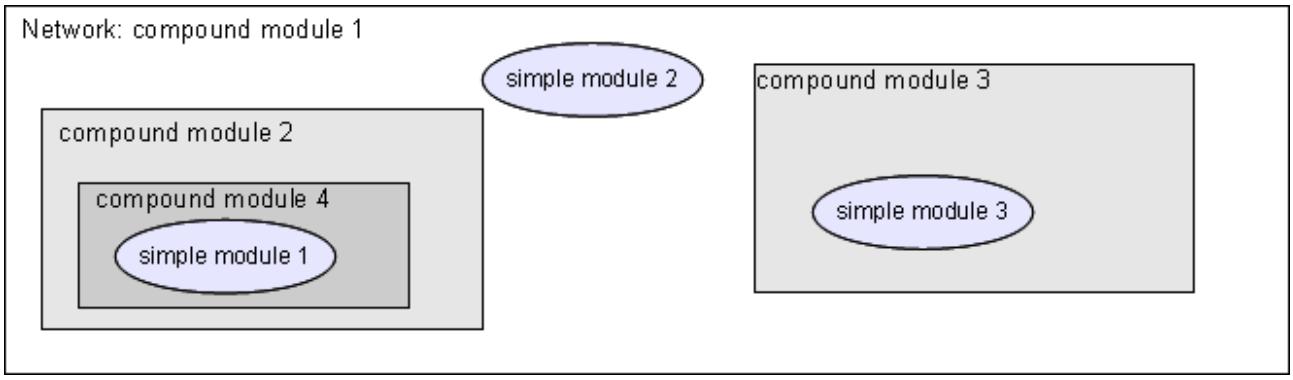
2. Omnet++: Ανάλυση Δομής

2.1 Αρχιτεκτονική του Omnet++

Στο Omnet++ οι προσομοιωτές εξετάζουν διάφορα μοντέλα. Τα μοντέλα αυτά αναφέρονται ως δίκτυα (networks). Κάθε μοντέλο-δίκτυο αποτελείται από ιεραρχικά δομημένες 'μονάδες' (modules). Η κορυφαία στην ιεραρχία μονάδα είναι η 'μονάδα συστήματος' και ισοδυναμεί με το δίκτυο που προσομοιώνουμε ως οντότητα-αντικείμενο. Αυτή περιέχει άλλα υπομέρη, τις λεγόμενες 'υπομονάδες' (submodules), οι οποίες επίσης δύνανται με τη σειρά τους να περιέχουν άλλες υπομονάδες. Δεν υπάρχει περιορισμός στο εύρος της υποδιαίρεσης μιας μονάδας [17].

Η δομή ενός μοντέλου περιγράφεται με την NED γλώσσα προγραμματισμού του Omnet++. Οι μονάδες που περιέχουν άλλες υπομονάδες λέγονται 'σύνθετες μονάδες' (compound modules). Οι μονάδες που βρίσκονται στο κατώτερο επίπεδο της ιεραρχίας καλούνται 'απλές μονάδες' (simple modules). Οι απλές μονάδες είναι αυτές που περιέχουν τους αλγόριθμους του μοντέλου και καθορίζουν τη συμπεριφορά και λειτουργία του προσομοιωτή. Οι απλές μονάδες υλοποιούνται στη C++.

Τόσο οι απλές όσο και οι σύνθετες μονάδες είναι ενσαρκώσεις 'τύπων μονάδων' (module types). Ο χρήστης όταν σχεδιάζει το μοντέλο ορίζει τύπους μονάδων. Ενσαρκώσεις αυτών αποτελούν μέρη πιο σύνθετων τύπων μονάδων, και εν τέλει ο χρήστης σχεδιάζει το δίκτυο/σύστημα σαν ένα τύπο μονάδος στο οποίο όλοι οι υπόλοιποι τύποι ενσαρκώνονται σαν υπομονάδες.



Σχήμα 2.1

Όταν ένας τύπος μονάδος χρησιμοποιείται στο χτίσιμο του δικτύου, δεν εξετάζεται αν αυτή είναι απλή ή σύνθετη, και δεν αντιμετωπίζεται διαφορετικά σε κάθε περίπτωση. Δηλαδή με τον ίδιο τρόπο που χρησιμοποιούμε ένα τύπο σύνθετης χρησιμοποιούμε και ένα τύπο απλής μονάδος. Αυτό επιτρέπει να χωρίζουμε μια απλή μονάδα σε πολλές μικρότερες απλές μονάδες οι οποίες να τη συγκροτούν πλέον ως μια σύνθετη μονάδα Έτσι μπορούμε να οργανώσουμε το σύστημά μας πιο αποδοτικά σε μικρά μέρη με αυτόνομο χαρακτήρα. Φυσικά είναι δυνατό και το αντίθετο, δηλαδή να ενώσουμε μερικές απλές μονάδες που συνιστούν μία σύνθετη σε μια απλή, που να υλοποιεί τις ίδιες λειτουργίες με την πρώτη υλοποίηση, και χωρίς να επηρεάζεται η διαχείριση της μονάδος από το σύστημα.

2.1.1 Γεγονότα στο Omnet++

Το Omnet++ χρησιμοποιεί μηνύματα για να αναπαραστήσει γεγονότα. Κάθε γεγονός είναι μια ενσάρκωση της cMessage τάξης ή κάποιας παράγωγης τάξης της. Τα μηνύματα αποστέλλονται από μονάδα σε μονάδα μέχρι να καταλήξουν στον προορισμό τους. Το γεγονός που ενεργοποιείται από το μήνυμα λαμβάνει χώρα στη μονάδα που παραλαμβάνει το μήνυμα και στον χρόνο προσομοίωσης που το παρέλαβε (arrival time). Τα μηνύματα επομένως εκτελούνται κατά προτεραιότητα παράδοσης στον παραλήπτη. Αν δύο μηνύματα φθάσουν στον ίδιο χρόνο προσομοίωσης στον ίδιο προορισμό, τότε εκτελείται πρώτα αυτό που έχει μικρότερη τιμή στην παράμετρο 'προτεραιότητα' (priority). Αν και αυτή η παράμετρος είναι ίδια για τα δύο μηνύματα τότε πρώτα εκτελείται αυτό που εστάλη πρώτο [17].

2.2 Επικοινωνία Μονάδων (Module Communication)

Οι μονάδες επικοινωνούν στο Omnet++ ανταλλάσσοντας μηνύματα. Τα μηνύματα αυτά αντιπροσωπεύουν τα στοιχεία ροής του δικτύου που προσομοιώνουμε. Πχ για ένα οδικό δίκτυο τα μηνύματα αντιπροσωπεύουν τα οχήματα, για ένα δίκτυο επικοινωνίας δεδομένων τα πακέτα δεδομένων κλπ. Τα μηνύματα μπορούν να περιέχουν σύνθετες δομές δεδομένων.

Ο 'τοπικός χρόνος προσομοίωσης' μιας μονάδος προχωράει όταν αυτή δέχεται ένα μήνυμα. Το μήνυμα αυτό μπορεί να έχει αποσταλεί από κάποια άλλη μονάδα, ή από την ίδια (selfMesssage) ώστε να εκκινήσει κάποιο γεγονός.

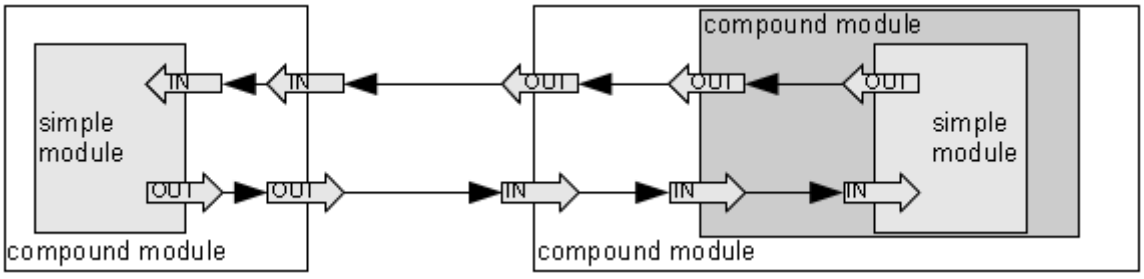
Οι μονάδες μπορούν να στέλνουν μηνύματα κατευθείαν στους προορισμούς τους, ή μέσω θυρών (gates) και συνδέσεων (links).

Οι θύρες (gates) είναι τα στοιχεία εισόδου/εξόδου των μονάδων και είναι δύο ειδών:

- input gates : θύρες εισόδου στις οποίες φτάνουν μηνύματα από άλλες μονάδες
- output gates : θύρες εξόδου από τις οποίες στέλνονται μηνύματα προς άλλες μονάδες

Οι συνδέσεις (links) γίνονται αποκλειστικά στο ίδιο επίπεδο ιεραρχίας ή στα αμέσως γειτονικά, εφόσον συνδέεται σύνθετη μονάδα με εσωτερική της υπομονάδα. Εντός μιας σύνθετης μονάδας είτε θα συνδέονται μεταξύ τους δύο υπομονάδες είτε μια υπομονάδα με τη σύνθετη μονάδα που την περιέχει. Οι συνδέσεις γίνονται στις θύρες των μονάδων και προφανώς στα άκρα μιας διαδρομής πάντα συνδέεται μια θύρα εξόδου σε μια εισόδου.

Μηνύματα δημιουργούν και επεξεργάζονται μόνο οι απλές μονάδες. Οι σύνθετες μονάδες απλά δρομολογούν τα μηνύματα μεταξύ τους στο ίδιο επίπεδο ιεραρχίας και στις υπομονάδες που περιέχουν. Εξαιτίας της συνθήκης ιεραρχίας των συνδέσεων, ένα μήνυμα προκειμένου να φτάσει στον προορισμό του περνάει μέσα από μια σειρά συνδέσεων ανάμεσα σε απλές και σύνθετες μονάδες. Μια τέτοια σειρά συνδέσεων, από απλή μονάδα "πηγή" σε απλή μονάδα "προορισμό", που ακολουθεί ένα μήνυμα καλείται "διαδρομή" (route) [17].



Σχήμα 2.2

Οι συνδέσεις ανάμεσα στις μονάδες μπορούν να ενσωματώνουν τρεις προαιρετικές παραμέτρους, οι οποίες μοντελοποιούν τα χαρακτηριστικά της επικοινωνίας:

1. Propagation Delay - Χρόνος Αναμετάδοσης: αντιπροσωπεύει το χρόνο που καθυστερεί ένα μήνυμα όταν περνάει από τη συγκεκριμένη σύνδεση.

2. Bit Error Rate - Ρυθμός Σφάλματος: καθορίζει την πιθανότητα να μην μεταδίδεται σωστά ένα bit πληροφορίας.

3. Data Rate - Ρυθμός Δεδομένων: ορίζει την ταχύτητα της σύνδεσης σε bits/sec και χρησιμοποιείται για τον υπολογισμό του χρόνου μετάδοσης ενός πακέτου. Ο χρόνος αυτός υπολογίζεται από την αποστολή του 1ου bit ενός πακέτου μέχρι την λήψη του τελευταίου bit από τον παραλήπτη.

Κάθε σύνδεση στο Omnet++ είναι δυνατό να έχει τις δικές τις παραμέτρους και αντίστοιχα τιμές.

2.3 Παράμετροι Μονάδων (Module Parameters)

Οι μονάδες μπορούν να έχουν παραμέτρους διαφόρων τύπων. Είναι δυνατό να τις ορίζουμε σε NED γλώσσα ορισμού στα ned αρχεία των μονάδων, ή σε ini αρχεία ρυθμίσεων. Οι παράμετροι ρυθμίζουν την συμπεριφορά των μοντέλων και των δικτύων και καθορίζουν την εκτέλεση των προσομοιώσεων. Μπορούν να είναι:

- Numeric - Αριθμητικά
- String - Σειρές χαρακτήρων
- Bool - Σωστό/Λάθος
- XML data – δεδομένα XML

2.4 Διεπαφή Χρήστη (User Interface)

Το Omnet++ προσφέρει στο χρήστη δύο διαφορετικούς τρόπους εκτέλεσης προσομοιώσεων:

(a) Σε γραφικό περιβάλλον υλοποιημένο σε Tcl/Tk
(b) Εκτέλεση στη γραμμή εντολών του κελύφους

Κάθε υλοποίηση έχει τα πλεονεκτήματα και μειονεκτήματα της. Σε γενικές γραμμές μπορούμε να πούμε ότι το γραφικό περιβάλλον ενδείκνυται για παρουσιάσεις προσομοιώσεων, για εντοπισμό σφαλμάτων (debugging), αφού επιτρέπει την τροποποίηση μεταβλητών των μονάδων και του δικτύου κατά την εκτέλεση και γενικότερα για την καλύτερη εποπτεία της προσομοίωσης. Από την άλλη η εκτέλεση σε γραμμή εντολών δίνει ταχύτητα στην εξαγωγή αποτελεσμάτων και είναι ιδανική για μαζικές εκτελέσεις σεναρίων [17].

2.5 Απλή Μονάδα (Simple Module)

Τα γεγονότα στο Omnet++ γεννιούνται και συμβαίνουν μέσα σε απλές μονάδες. Αυτές περιέχουν τις μεθόδους που δημιουργούν και επεξεργάζονται μηνύματα και καθορίζουν την συμπεριφορά του μοντέλου. Οι απλές μονάδες είναι παράγωγες τάξεις της cSimpleModule τάξης, που είναι τμήμα της βιβλιοθήκης τάξεων του Omnet++.

Ο ορισμός και η κατασκευή μιας απλής μονάδας υλοποιείται σε δύο φάσεις:
1. Με την περιγραφή της μονάδας στη NED γλώσσα
2. Με την υλοποίηση και τον ορισμό των λειτουργιών της σε C++ κώδικα

2.5.1 Απλή Μονάδα: NED ορισμός

Η περιγραφή της μονάδας στη γλώσσα NED μπορεί να γίνει είτε σε ένα ned αρχείο ορισμού μιας σύνθετης μονάδας που την περιέχει, είτε σε ένα αυτόνομο ned αρχείο. Σημαντικό είναι στη δεύτερη περίπτωση να μην παραλείπεται η οδηγία "import" (εισαγωγή) με την οποία εισάγουμε τον ορισμό μιας υπομονάδας στη σύνθετη μονάδα που την περιέχει, ώστε αυτή να δομηθεί σωστά [17].

Οι απλές μονάδες είναι τα βασικά συστατικά δόμησης για άλλες σύνθετες μονάδες. Οι τύποι των απλών μονάδων αναγνωρίζονται από το όνομά τους. Από σύμβαση τα ονόματα των απλών μονάδων ξεκινούν με κεφαλαία γράμματα (SimpleModuleName).

Μια απλή μονάδα ορίζεται στη NED γλώσσα δηλώνοντας τις παραμέτρους και τις θύρες της. Η σύνταξη της δήλωσης ενός simple module είναι:

```
simple SimpleModuleName
    parameters:
      .
      .
      .
    gates:
      .
      .
      .
endsimple
```

2.5.1.1 Παράμετροι Απλής Μονάδος

Οι παράμετροι είναι μεταβλητές που ανήκουν σε μια μονάδα και αξιοποιούνται από τη μονάδα στις εσωτερικές της λειτουργίες. Οι παράμετροι αναγνωρίζονται από το όνομά τους, το οποίο από σύμβαση ξεκινά με μικρό γράμμα (parameterName), και δηλώνονται με παράθεση του ονόματός τους σε μια λίστα στον τομέα "parameters:" της περιγραφής της μονάδος. Κατά την δήλωση των παραμέτρων μπορεί προαιρετικά να δηλωθεί ο τύπος τους. Αυτός μπορεί να πάρει τις εξής τιμές:

1. numeric - αριθμητικό (προεπιλεγμένος τύπος σε περίπτωσης παράληψης δήλωσης)
2. numeric const / const - αριθμητικό σταθερό
3. string - σειρά χαρακτήρων
3. bool - σωστό/λάθος
4. XML – XML δεδομένα

Παράδειγμα σύνταξης [17]:

```
simple SimpleModuleName
    parameters:
        numericParameter,
        numericConstParameter : const,
        stringParameter : string;
    gates: //...
endsimple
```

Οι παράμετροι απλά δηλώνονται στο NED ορισμό μιας μονάδος, και δεν παίρνουν τιμές. Αυτό γίνεται στη NED γλώσσα για μια μονάδα (απλή ή σύνθετη) μόνο όταν αυτή χρησιμοποιείται σαν δομικό στοιχείο μιας μεγαλύτερης σύνθετης μονάδας που βρίσκεται ψηλότερα στην ιεραρχία. Δηλαδή οι τιμές θα αποδοθούν στη NED δήλωση της σύνθετης μονάδας που περιέχει τη μονάδα στην οποία ανήκουν οι παράμετροι. Για να αποδοθούν ευθέως αρχικές τιμές στις παραμέτρους μιας μονάδος πρέπει αυτές να φορτωθούν κατά την εκκίνηση της προσομοίωσης από ini αρχείο ρυθμίσεων. Όλες οι παράμετροι των μονάδων στο Omnet++ που δηλώνονται στον NED ορισμό πρέπει να πάρουν αρχικές τιμές πριν την προσομοίωση. Αν αυτό δε συμβεί, τότε θα προβάλει αίτημα συμπλήρωσής τους ο προσομοιωτής κατά την εκκίνηση. Σε αυτή τη περίπτωση προεπιλεγμένη τιμή είναι '0' για τα numeric, κενό ("") για τα string και 'false' για τα bool [17].

Οι αριθμητικές παράμετροι μπορούν να πάρουν τιμές και από αριθμητικές συναρτήσεις. Στη NED γλώσσα επιτρέπεται η χρήση αρκετών μαθηματικών συναρτήσεων της C βιβλιοθήκης <math.h>, όπως και αρκετές συναρτήσεις παραγωγής τυχαίων αριθμών (NRGs) που προσφέρει το ίδιο το Omnet++ [6.4.4]. Η απόδοση των συναρτήσεων στις παραμέτρους μιας μονάδος γίνεται, όπως και με την απόδοση σταθερών τιμών, στη NED δήλωση της σύνθετης μονάδος που την περιέχει.

2.5.1.1.1 Σταθερές Τιμές Παραμέτρων (Constants)

Οι σταθερές τιμές που μπορούν να αποδοθούν σε παραμέτρους είναι:

- numeric constants - αριθμητικές σταθερές
  Οι αριθμητικές σταθερές γίνονται δεκτές τόσο σε δεκαδική μορφή όσο και σε επιστημονική σημειογραφία

- string constants - σταθερές σειρές χαρακτήρων
  Πρέπει να εσωκλείονται σε διπλά εισαγωγικά (double quotes)

- time constants - σταθερές χρόνου
  Οι σταθερές αυτές αντιπροσωπεύουν χρόνο σε διάφορες μονάδες υποδιαίρεσης [17]:

```
ns    nanoseconds
us    microseconds
ms    milliseconds
s     seconds
m     minutes (60s)
h     hours (3600s)
d     days (86400s)
```

2.5.1.2 Θύρες Απλής Μονάδας (Simple Module Gates)

Οι θύρες (gates) είναι τα σημεία σύνδεσης των μονάδων. Για κάθε σύνδεση ανάμεσα σε δύο μονάδες, τα σημεία εκκίνησης και προορισμού είναι θύρες. Το Omnet++ δεν υποστηρίζει αμφίδρομες συνδέσεις, γεγονός που οδηγεί στο να υπάρχουν θύρες εισόδου (input gates) και θύρες εξόδου (output gates). Τα μηνύματα στέλνονται από θύρες εξόδου και λαμβάνονται σε θύρες εισόδου [17].

Οι θύρες αναγνωρίζονται με το όνομά τους, το οποίο κατά σύμβαση ξεκινά με μικρό γράμμα. Το Omnet++ υποστηρίζει και διανύσματα θυρών (gate vectors), όπου ένα διάνυσμα έχει μέγεθος που ανταποκρίνεται στον αριθμό θυρών που περιέχει.

Οι θύρες δηλώνονται με το όνομά τους στον τομέα "gates:" της NED δήλωσης της μονάδας. Ένα διάνυσμα θυρών δηλώνεται με το όνομά του ακολουθούμενο από διπλή παρένθεση [ ]. Τα μεγέθη των διανυσμάτων ορίζονται όταν η μονάδα χρησιμοποιείται σαν υπομονάδα σε μια σύνθετη μονάδα, από τη NED δήλωση της τελευταίας. Αυτό επιτρέπει διαφορετικές ενσαρκώσεις του ίδιου μοντέλου να έχουν διαφορετικού μεγέθους διανύσματα θυρών, δηλαδή διαφορετικό αριθμό θυρών.

Παράδειγμα σύνταξης [17]:

```
simple SimpleModuleName
    parameters:
       //...
    gates:
        in:  fromPort, fromHigherLayer;
        out: toPort, toHigherLayer;
endsimple

simple Router
    parameters:
      //...
    gates:
        in:  output[];
        out: input[];
endsimple
```

2.5.2 Απλή Μονάδα: C++ υλοποίηση

Εφόσον έχει γίνει η NED δήλωση μιας απλής μονάδος, επόμενο βήμα είναι να οριστεί η λειτουργικότητά της. Αυτό γίνεται υλοποιώντας τις διάφορες λειτουργίες της σε γλώσσα προγραμματισμού C++. Οι τύποι απλών μονάδων δημιουργούνται ως παράγωγες τάξεις της cSimpleModule τάξης της βιβλιοθήκης τάξεων του Omnet++. Αυτή με τη σειρά της είναι παράγωγη τάξη της cModule βασικής τάξης, όπως είναι και η cCompoundModule που υλοποιεί την στοιχειώδη λειτουργικότητα των σύνθετων μονάδων [17].

Η cSimpleModule τάξη, παρότι είναι εξοπλισμένη με λειτουργίες σχετικές με την προσομοίωση, δεν κάνει κάποια εργασία από μόνη της, εκτός αν αναδιατυπώσουμε κάποιες εικονικές μεθόδους μελών που διαθέτει, ώστε να δώσουμε στη μονάδα κάποια λειτουργικότητα και συμπεριφορά. Οι μέθοδοι αυτοί είναι:

void initialize() // εκκίνησε
*void handleMessage(cMessage *msg)* // διαχειρίσου μήνυμα
void activity() // δραστηριότητα
void finish() // τέλος

- *void initialize()*

Κατά την εκκίνηση της προσομοίωσης το Omnet++ χτίζει το δίκτυο σύμφωνα με τους ορισμούς που έχουμε δώσει στη NED γλώσσα και καλεί την *initialize()* μέθοδο για όλες τις μονάδες. Σε αυτή τη μέθοδο πρέπει να υλοποιήσουμε όλα τα γεγονότα που θέλουμε να εκτελεστούν κατά την οικοδόμηση του δικτύου. Επίσης είναι ιδανικό να διαβάσουμε σε αυτό το σημείο της προσομοίωσης τις παραμέτρους της μονάδας που εκκινεί η μέθοδος και να τις αποδώσουμε σε μεταβλητές στη C++. Αν θέλουμε η μονάδα να διαχειρίζεται selfMessages, πρέπει να στείλουμε το πρώτο σε αυτή τη φάση.

- *void handleMessage (cMessage *msg)*

Η μέθοδος αυτή καλείται όταν σε μια απλή μονάδα φτάσει κάποιο μήνυμα. Στο Omnet++ τα μηνύματα ενεργοποιούν γεγονότα στις μονάδες που τα παραλαμβάνουν, και συνεπώς η μέθοδος αυτή υλοποιεί όλα τα γεγονότα που είναι δυνατό να εκτελέσει μια μονάδα. Τα μηνύματα που διαχειρίζεται η μέθοδος είναι τόσο μηνύματα από άλλες μονάδες όσο και μηνύματα που αποστέλλει μια μονάδα στον εαυτό της (selfMessages) για να εκκινήσει γεγονότα. Η λογική του selfMessage είναι το κύριο υπόβαθρο υλοποίησης λειτουργιών στις απλές μονάδες.

Όταν φτάνει ένα μήνυμα σε μια μονάδα το αναλαμβάνει η handleMessage (cMessage *msg) μέθοδος και αφού το επεξεργαστεί επιστρέφει το όποιο αποτέλεσμα χωρίς να έχει καταναλώσει χρόνο προσομοίωσης. Δηλαδή η μέθοδος δεν εκτελείται σε βάθος χρόνου αλλά στιγμιαία.

- *void activity()*

Η μέθοδος *activity()* είναι ο δεύτερος τρόπος διαχείρισης εισερχόμενων μηνυμάτων που προσφέρει το Omnet++. Οι διαφορές που παρουσιάζει ως προς την *handleMessage(cMessage *msg)* γεννούνται από το γεγονός ότι η *activity()* μέθοδος εφαρμόζεται σαν μια εργασία με αρχή και τέλος, μέσα στην οποία μπορούν να δημιουργούνται και να λαμβάνονται μηνύματα, καθώς και να εκτελούνται διάφορες άλλες λειτουργίες, σε επαναλήψεις. Όταν η μέθοδος επιστρέφει στην προσομοίωση, η μονάδα που ανήκει τερματίζεται και παύει να συμμετέχει στην προσομοίωση.

- *void finish()*

Η λειτουργία *finish()* καλείται κάθε φορά που τερματίζεται επιτυχώς μια προσομοίωση. Η πιο τυπική χρήση της λειτουργίας αυτή είναι η αποθήκευση, αναφορά ή εκτύπωση στατιστικών που συλλέχθηκαν κατά την προσομοίωση.

Στη C++ υλοποίηση των απλών μονάδων, μπορούμε φυσικά να δημιουργήσουμε και επιπλέον μεθόδους από τις προαναφερθείσες, για να υποστηρίξουμε τις ανάγκες της προσομοίωσης. Είναι σημαντικό όμως αυτές πάντα να εκκινούν μέσα από κάποια εκ των βασικών μεθόδων της τάξης.

2.6 Σύνθετη Μονάδα (Compound Module)

Οι σύνθετες μονάδες αποτελούνται από μία ή περισσότερες μονάδες στο κατώτερο επίπεδο ιεραρχίας. Αυτές μπορούν να είναι είτε απλές είτε επίσης σύνθετες. Όπως και οι απλές μονάδες έτσι και οι σύνθετες έχουν παραμέτρους και θύρες. Η λειτουργία και η χρήση τους, όπως και η συμπεριφορά της σύνθετης μονάδος στην προσομοίωση, είναι απολύτως ίδιες με αυτές μιας απλής [17].

Οι σύνθετες μονάδες δεν έχουν ενεργό ρόλο στην προσομοίωση με μεθόδους διαχείρισης και δημιουργίας μηνυμάτων ή άλλες. Ως εκ τούτου η υλοποίηση των σύνθετων μονάδων γίνεται εξολοκλήρου στην NED γλώσσα και αποθηκεύεται σε ned αρχεία. Ουσιαστικά αποτελούν δοχεία (containers) για άλλες μονάδες, ώστε να τις ομαδοποιούν σε μεγαλύτερα στοιχεία. Αυτά μπορούν να αποτελέσουν ένα μοντέλο μιας λειτουργίας ή ενός αντικειμένου, ή να χρησιμοποιηθούν σα δομικό στοιχείο μιας μεγαλύτερης σύνθετης μονάδος. Έτσι οργανώνεται καλύτερα η προσομοίωση σε αυτόνομες λειτουργικές μονάδες και δύναται να αποκτά μια σχεδιαστική δομή που ανταποκρίνεται στο προς προσομοίωση σενάριο.

Οι τύποι σύνθετων μονάδων αναγνωρίζονται με το όνομά τους, το οποίο από σύμβαση ξεκινά με μεγάλο γράμμα. Οι υπομονάδες μπορούν να χρησιμοποιούν παραμέτρους της σύνθετης μονάδας που τις περιέχει. Μπορούν επίσης να συνδέονται μαζί της, όπως και μεταξύ τους.

2.6.1 Σύνθετη Μονάδα: NED ορισμός

Η NED δήλωση μιας σύνθετης μονάδας είναι σχεδόν όμοια με αυτή μιας απλής. Όπως και η απλή μονάδα, έτσι και η σύνθετη έχει παραμέτρους και θύρες. Η διαφορά είναι ότι υπάρχουν δύο επιπλέον τομείς, "submodules:" και "connections:", για τις υπομονάδες και τις συνδέσεις αντίστοιχα. Η σύνταξη για μια σύνθετη μονάδα είναι:

```
module CompoundModule
    parameters:
        //...
    gates:
        //...
    submodules:
        //...
    connections:
        //...
endmodule
```

2.6.1.1 Παράμετροι Σύνθετης Μονάδας

Οι παράμετροι μιας σύνθετης μονάδος δηλώνονται και λειτουργούν με τον ίδιο τρόπο όπως και σε μια απλή. Συχνά οι παράμετροι μια σύνθετης μεταβιβάζονται στις υπομονάδες τους για να αποδώσουν τιμές στις δικές τους παραμέτρους.

Οι παράμετροι μιας σύνθετης μονάδας χρησιμοποιούνται επιπλέον για να ορίσουν την εσωτερική της δομή. Είναι δυνατό να ορίσουμε παράμετρο που θα ορίζει τον αριθμό των υπομονάδων από ένα τύπο μονάδος, να ορίσουμε μεγέθη σε πίνακες θυρών μεταβιβάζοντας παραμέτρους στις υπομονάδες και να καθορίσουμε τις συνδέσεις στο εσωτερικό της σύνθετης μονάδας. Παραμέτρους που επηρεάζουν τις εσωτερικές λειτουργίες μιας σύνθετης μονάδας, όπως και για τις απλές, είναι σημαντικό να εξετάζουμε αν επιβάλλεται να είναι σταθερές, οπότε να τις δηλώνουμε ως τέτοιες. Σε αντίθετη περίπτωση υπάρχει πάντα ο κίνδυνος να μεταβληθεί μια κρίσιμη τιμή από κάποιο λάθος το οποίο δύσκολα θα φαίνεται, αφού θα επιτρέπεται η αλλαγή, και τα αποτελέσματα της προσομοίωσης να είναι τελείως λάθος, χωρίς συχνά να μπορούμε να βρούμε το λόγο. Είναι μια λογική που ακολουθεί την C++ οδηγία για σταθερού τύπους (const types) [17].

Παράδειγμα σύνταξης [17]:

```
module Router
    parameters:
        packetsPerSecond : numeric,
        bufferSize : numeric,
        numOfPorts : const;
    gates:
      //...
    submodules:
      //...
    connections:
      //...
endmodule
```

2.6.1.2 Θύρες Σύνθετης Μονάδας

Οι θύρες μιας σύνθετης μονάδος δηλώνονται με τον ίδιο τρόπο όπως και σε μια απλή. Το ίδιο ισχύει και για τα διανύσματα θυρών. Ενώ και εδώ τα διανύσματα θυρών δηλώνονται εξαρχής στον τομέα "gates:" χωρίς μέγεθος, εν τούτοις οι σύνθετες μονάδες ορίζουν το μέγεθός τους στον τομέα "connections:".

Παράδειγμα σύνταξης [17]:

```
module Router
    parameters:
      //...
    gates:
        in: inputPort[];
        out: outputPort[];
    submodules: //...
    connections: //...
endmodule
```

Είναι εξαιρετικά σημαντικό να τονισθεί το γεγονός ότι οι θύρες των σύνθετων μονάδων είναι στην ουσία πύλες (gateways) για συνδέσεις μεταξύ απλών μονάδων. Δηλαδή μια θύρα σύνθετης μονάδας δεν μπορεί να αποτελεί τελικό προορισμό ενός μηνύματος και τέρμα της διαδρομής που αυτό ακολουθεί. Ως συνέπεια αυτού του γεγονότος οι θύρες των σύνθετων μονάδων είναι σύμφωνες πάντα με την κατεύθυνση επικοινωνίας.

2.6.1.3 Υπομονάδες Σύνθετων Μονάδων

Στην NED δήλωση μιας σύνθετης μονάδος προβλέπεται ο ειδικός τομέας "submodules:" στον οποίο γίνεται ο ορισμός των υπομονάδων που περιέχονται σε αυτό. Οι υπομονάδες ταυτοποιούνται με το όνομά τους, το οποίο από σύμβαση ξεκινά με μικρό γράμμα.

Παράδειγμα σύνταξης [17]:

```
module CompoundModule
    //...
    submodules:
        submodule1: ModuleType1
            parameters:
                //...
            gatesizes:
                //...
        submodule2: ModuleType2
            parameters:
                //...
            gatesizes:
                //...
endmodule
```

Οι υπομονάδες είναι ενσαρκώσεις ενός τύπου απλής ή σύνθετης μονάδας. Δεν υπάρχει καμία διάκριση ανάμεσα σε κάθε περίπτωση. Ο τύπος μονάδας πρέπει να είναι γνωστός στον NED compiler. Αυτό εξασφαλίζεται εφόσον έχει δηλωθεί νωρίτερα στο ίδιο ned αρχείο, ή αν εισαχθεί με την οδηγία "import" από άλλο ned αρχείο όπου υπάρχει η δήλωσή του [17].

Είναι δυνατό να ορίζουμε διανύσματα υπομονάδων (module vectors) εντός μιας σύνθετης μονάδος. Το μέγεθος του διανύσματος, δηλαδή ο αριθμός των υπομονάδων που θα δημιουργηθούν από τον συγκεκριμένο τύπο, θα ορίζεται από παράμετρο η τιμή της οποίας είτε θα αποδίδεται από ini αρχείο ρυθμίσεων, είτε θα καθορίζεται με αριθμητικό στη NED δήλωση, ή θα αποδίδεται από σύνθετη μονάδα ανώτερου επιπέδου ιεραρχίας. Η λογική δήλωσης είναι ίδια με αυτή των διανυσμάτων θυρών, και υλοποιείται με μια έκφραση ανάμεσα σε παρένθεση. Η έκφραση μπορεί να αναφέρεται σε παράμετρο της σύνθετης μονάδας, να έχει θετική ακέραιη αριθμητική τιμή, '0' (μηδέν) ή οποιαδήποτε έγκυρη αριθμητική πράξη επί παραμέτρων και αριθμών που να δίνει σαν αποτέλεσμα θετικό ακέραιο.

Παράδειγμα σύνταξης [17]:

```
module CompoundModule
    parameters:
        size: const;
    submodules:
        submod1: Node[3]
            //...
        submod2: Node[size]
            //...
        submod3: Node[2*size+1]
            //...
endmodule
```

2.6.1.3.1 Παραμετροποίηση Τύπου Μονάδος για Υπομονάδες Σύνθετης Μονάδας

Ο τύπος μονάδος από τον οποίο δημιουργείται μια υπομονάδα, μπορεί να αποτελέσει παράμετρο για τη σύνθετη μονάδα που την περιέχει. Αυτό μας δίνει τη δυνατότητα σε μια υλοποίηση να μπορούμε να δοκιμάζουμε διάφορα μοντέλα, τα οποία θα κάνουν την ίδια εργασία αλλά με διαφορετικό τρόπο. Φυσικά τα κίνητρα μπορούν να είναι άλλα, όπως ευελιξία, προσαρμοστικότητα, συμβατότητα. Η λογική αυτή θυμίζει τον πολυμορφισμό που συναντάμε στις αντικειμενοστραφείς γλώσσες προγραμματισμού [17].

Η διαδικασία της παραμετροποίησης του τύπου μονάδος μιας υπομονάδος είναι η εξής:

- Δηλώνεται η παράμετρος για τον τύπο της μονάδος. Η παράμετρος είναι τύπου 'string'

```
module CompoundModule
     parameters:
         subModType: string;
```

- Κάθε υπομονάδα που θέλει να αξιοποιήσει την παράμετρο για να ορίζει τον τύπο της, την δηλώνει με το όνομά της στην σύνταξη ως εξής:

```
module CompoundModule
     parameters:
         subModType: string;
```

Η έκφραση 'like SimilarModuleType' είναι ένας μηχανισμός ασφαλείας για την δήλωση. Το SimilarModuleType (Παρόμοιος Τύπος Μονάδας) πρέπει να είναι ένας τύπος μονάδας που θα έχει οπωσδήποτε τις ίδιες θύρες και παραμέτρους με τον τύπο που παραμετροποιούμε. Ο NED compiler ελέγχει αυτή τη συνθήκη για κάθε τύπο μονάδας που αποδίδουμε στη παράμετρο 'subModType'. Το SimilarModuleType πρέπει να είναι δηλωμένο στην NED γλώσσα, χωρίς όμως να χρειάζεται να είναι υλοποιημένο σε C++, διότι το ίδιο δεν υλοποιείται πουθενά. Οι τύποι μονάδας που συνδέονται μαζί του, πρέπει στην NED δήλωση να έχουν ακριβώς τις ίδιες θύρες και τουλάχιστον τις ίδιες παραμέτρους. Είναι δηλαδή δυνατό να περιέχουν περισσότερες παραμέτρους όχι όμως και θύρες. Η 'like' φράση ουσιαστικά αποδίδει σε όλους αυτούς τους τύπους την δήλωση του SimilarModuleType και τους οργανώνει σε μια οικογένεια μονάδων με κοινά χαρακτηριστικά.

Παράδειγμα σύνταξης [17]:

```
module WirelessNetwork
    parameters:
        wifiNodeType: string;
    gates:
      //...
    submodules:
        node1: wifiNodeType like WirelessNode;
        node2: wifiNodeType like WirelessNode;
        //...
    connections nocheck:
        node1.out0 --> node2.in0;
        //...
endmodule
```

Στο παραπάνω παράδειγμα οι 'node1' και 'node2' υπομονάδες παίρνουν τον τύπο μονάδος από την παράμετρο 'wifiNodeType'. Ο τύπος αυτός είναι στα πρότυπα του 'WirelessNode' τύπου μονάδος, ο οποίος είναι δηλωμένο στη NED γλώσσα.

Η παραμετροποίηση του τύπου μονάδας είναι μια πολύ σημαντική ιδιότητα της δυναμικής κατασκευής προσομοιώσεων μέσω της NED δήλωσης. Παρότι εισάγονται περιορισμοί στη χρήση τύπου μονάδας, είναι δικαιολογημένοι, προκειμένου να προστατευθεί ο χρήστης από σφάλματα κατά την εκτέλεση. Είναι δυνατό να παρακάμψει κανείς αυτή τη πολιτική και να επέμβει κατευθείαν στον C++ κώδικα ώστε να δημιουργεί το δίκτυο με άλλα κριτήρια ενδεχομένως πιο πολύπλοκα και αποδοτικά. Ωστόσο η NED γλώσσα και οι υποδομές που προσφέρει είναι πολύ απλές στην κατανόηση και την οργάνωση ώστε να κάνουν και προσιτή τη χρήση του Omnet++ για τον προγραμματισμό προσομοιώσεων.

2.6.1.3.2 Απόδοση Τιμών σε Παραμέτρους Υπομονάδων Σύνθετης Μονάδας

Σε μια σύνθετη μονάδα, αν οι τύποι μονάδων που χρησιμοποιούνται σαν υπομονάδες έχουν παραμέτρους, τότε είναι δυνατό να αποδώσουμε τιμές σε αυτές τις παραμέτρους για κάθε υπομονάδα. Αυτή η διαδικασία γίνεται στον τομέα "parameters:" που περιέχεται στη δήλωση της εκάστοτε υπομονάδας. Οι τιμές που μπορούν να αποδοθούν σε μια παράμετρο μπορούν να είναι σταθερά μεγέθη (πχ 0, 10, "string"), άλλες παράμετροι (πχ της σύνθετης μονάδας στην οποία ανήκει η υπομονάδα) και λογικές εκφράσεις / συναρτήσεις που να αξιοποιούν τα παραπάνω. Η σύνταξη των εκφράσεων είναι παρόμοια με αυτή της C γλώσσας προγραμματισμού [17].

Παράδειγμα σύνταξης [17]:

```
module CompoundModule
    parameters:
        param1: numeric,
        param2: numeric,
        useParam1: bool;
    submodules:
        submodule1: Node
            parameters:
                p1 = 10,
                p2 = param1+param2,
                p3 = useParam1==true ? param1 : param2;
endmodule
```

Δεν είναι υποχρεωτικό να δώσουμε τιμή εκκίνησης σε κάθε παράμετρο μιας υπομονάδας. Είναι δυνατό, όπως ισχύει σε κάθε περίπτωση μονάδος (απλής / σύνθετης), να θέσουμε τις τιμές μέσω ini αρχείου ρυθμίσεων ή και διαδραστικά κατά την εκκίνηση του προσομοιωτή. Είναι επιπλέον δυνατό να έχουμε μια προεπιλεγμένη τιμή για μια παράμετρο και να προβάλλουμε αίτημα εισαγωγής άλλης τιμής ή επιβεβαίωσης της προεπιλεγμένης κατά την προσομοίωση. Αυτό υλοποιείται με την "input" παραμετροποίηση της δήλωσης.

Παράδειγμα σύνταξης [17]:

```
   parameters:
      numCPUs = input(10, "Number of processors?"), // προεπιλεγμένη τιμή, κείμενο οδηγίας εισαγωγής
   processingTime = input(10ms), // προεπιλεγμένη τιμή ( time const )
   cacheSize = input; // δήλωση ότι η παράμετρος θα πάρει τιμή με εισαγωγή στην εκκίνηση
```

2.6.1.3.3 Ορισμός Μεγέθους Διανύσματος Θυρών Υπομονάδας

Το μέγεθος ενός διανύσματος θυρών για μια υπομονάδα ορίζεται στον τομέα "gatesizes:" που περιέχεται στην δήλωσή της στον τομέα "submodules:". Για κάθε διάνυσμα θυρών το μέγεθος ορίζεται με σταθερή τιμή, άλλη παράμετρο ή λογική έκφραση, τα οποία πρέπει να περιέχονται σε παρένθεση που ακολουθεί το όνομα του διανύσματος. Δεν είναι υποχρεωτικό να οριστεί εξαρχής το μέγεθος ενός διανύσματος θυρών. Αν δε γίνει αυτό τότε το διάνυσμα έχει μηδενικό μέγεθος.

Παράδειγμα σύνταξης [17]:

```
simple Node
    gates:
        in: inputs[];
        out: outputs[];
endsimple
module CompoundModule
    parameters:
        numPorts: const;
    submodules:
        node1: Node
            gatesizes:
                inputs[2], outputs[2];
        node2: Node
            gatesizes:
                inputs[numPorts], outputs[numPorts];
endmodule
```

Συνήθως το μέγεθος διανυσμάτων θυρών υπομονάδας ορίζεται στη σύνθετη μονάδα που περιέχει την μονάδα αυτή.

2.6.1.3.4 Έλεγχος Συνθήκης στους τομείς "parameters:" και "gatesizes:" Υπομονάδων

Είναι δυνατό στην δήλωση μιας υπομονάδας να ορίσουμε πολλαπλούς τομείς παραμέτρων "parameters:" και μεγέθους διανυσμάτων θυρών "gatesizes:". Σε αυτούς τους τομείς μπορούμε να έχουμε τόσο διαφορετικές παραμέτρους και τύπους θυρών, όσο και κοινές. Η δεύτερη περίπτωση είναι και ο λόγος που υφίσταται αυτή η υλοποίηση.

Εφόσον μια παράμετρος, ή ένα διάνυσμα κάποιου τύπου θύρας, δηλώνεται σε δύο τομείς, τότε σε αυτή την περίπτωση και χωρίς κάποια συνθήκη ισχύει η τελευταία δήλωση. Το Omnet++ όμως επιτρέπει να ενσωματώσουμε έλεγχο συνθήκης σε κάθε τομέα, ώστε να επιλέγεται ανάλογα με την πλήρωσή της η κατάλληλη τιμή για την παράμετρο ή το μέγεθος ενός διανύσματος θυρών. Καλό είναι να έχουμε ένα τομέα ανά κατηγορία χωρίς συνθήκη ώστε να εξασφαλίζουμε για όλες τις παραμέτρους και τα διανύσματα θυρών μια προεπιλεγμένη τιμή. Προκειμένου αυτή η τιμή να αντικαθίσταται από τις τιμές των 'τομέων με συνθήκη', πρέπει αυτοί να έπονται των προεπιλεγμένων τιμών, ώστε να υπερισχύουν αυτών. Παράδειγμα σύνταξης [17]:

```
module Chain
  parameters: count: const;
  submodules:
    node : Node [count]
      parameters:// τομέας παραμέτρων χωρίς συνθήκη
         position = "middle";// προεπιλεγμένη τιμή παραμέτρου position
      parameters if index==0:// τομέας παραμέτρων με συνθήκη #1
         position = "beginning";// ιμή παραμέτρου 'position' για συνθήκη #1
      parameters if index==count-1: // τομέας παραμέτρων με συνθήκη #2
         position = "end"; // τιμή παραμέτρου 'position' για συνθήκη #2
      gatesizes: // τομέας μεγέθους πινάκων θυρών χωρίς συνθήκη
         in[2], out[2]; // προεπιλεγμένα μεγέθη
      gatesizes if index==0 || index==count-1:
            // τομέας μεγέθους διανύσματος θυρών με συνθήκη #3
         in[1], in[1]; // μεγέθη θυρών για συνθήκη #3
  connections:
endmodule
```

2.6.1.4 Σύνθετες Μονάδες: Συνδέσεις

Στην NED δήλωση μιας σύνθετης μονάδος ορίζουμε πως συνδέονται οι θύρες της με αυτές των περιεχομένων της υπομονάδων που βρίσκονται στο αμέσως κατώτερο στάδιο ιεραρχίας, καθώς και οι θύρες αυτών μεταξύ τους. Είναι δηλαδή δυνατά δύο σενάρια συνδέσεων και δηλώσεων συνδέσεων εντός μιας σύνθετης μονάδος:

1. Συνδέσεις ανάμεσα στις υπομονάδες που περιέχει στο αμέσως κατώτερο επίπεδο ιεραρχίας
2. Συνδέσεις της σύνθετης μονάδας με υπομονάδες της στο αμέσως κατώτερο επίπεδο ιεραρχίας

Δεν είναι δυνατό να γίνουν συνδέσεις που παραβιάζουν την NED συνθήκη ιεραρχίας, σύμφωνα με την οποία για να μπορούν να συνδεθούν δύο μονάδες πρέπει να βρίσκονται στο ίδιο επίπεδο ιεραρχίας ή σε γειτονικά εφόσον πρόκειται για σχέση σύνθετη μονάδα --> / <-- υπομονάδα. Είναι επίσης επιτρεπτό να συνδέσουμε δύο θύρες (είσοδο με έξοδο) της ίδιας μονάδας μεταξύ τους. Επιβάλλεται, όπως έχει αναφερθεί και νωρίτερα, κάθε σύνδεση να υλοποιείται με διαδρομή ανάμεσα σε θύρα εξόδου και θύρα εισόδου δύο απλών μονάδων. Οι θύρες των σύνθετων μονάδων που περιέχονται στην διαδρομή αυτή αποτελούν στην ουσία πύλες (gateways) από τις οποίες προωθείται το μήνυμα στην επόμενη σύνδεση της διαδρομής προς τον τελικό προορισμό. Για το λόγο αυτό δεν χρησιμοποιούμε τον όρο 'αποστολή μηνυμάτων' αλλά 'προώθηση μηνυμάτων'. Η κατεύθυνση της θύρας πρέπει πάντα να συμφωνεί με την κατεύθυνση του μηνύματος από τον ένα επίπεδο ιεραρχίας στο άλλο. Μια θύρα που προωθεί μηνύματα από ανώτερο επίπεδο ιεραρχίας είναι πάντα θύρα εισόδου, και μια θύρα που προωθεί μηνύματα από κατώτερο επίπεδο ιεραρχίας είναι θύρα εξόδου. Αντίστοιχα όταν μια θύρα προωθεί μηνύματα προς ανώτερο επίπεδο ιεραρχίας είναι θύρα εξόδου και όταν προωθεί προς κατώτερο είναι θύρα εισόδου.

Οι συνδέσεις που ορίζονται στην NED δήλωση μιας σύνθετης μονάδας πρέπει οπωσδήποτε να είναι συνδέσεις 'ένας προς έναν' (one-to-one). Συνεπώς μία θύρα δεν μπορεί να εξυπηρετεί πολλαπλές συνδέσεις και εν γένει όλες οι θύρες πρέπει να χρησιμοποιούνται σε μία μόνο σύνδεση. Είναι ωστόσο δυνατό να υλοποιηθούν συνδέσεις 'ένας προς πολλούς' (one-to-many) και 'πολλοί προς έναν' (many-to-one) στην C++ υλοποίηση των απλών μονάδων. Η υλοποίηση που παρουσιάζεται στην παρούσα διπλωματική εργασία είναι βασισμένη σε συνδέσεις 'πολλοί προς έναν' [17].

Οι συνδέσεις ορίζονται στον τομέα "connections:" της NED δήλωσης μιας σύνθετης μονάδας. Σε κάθε γραμμή αναφέρεται μια σύνδεση ανάμεσα σε πηγή και προορισμό, με ένα βέλος ανάμεσά τους που δηλώνει την κατεύθυνση επικοινωνίας. Η κατεύθυνση αυτή πρέπει να δείχνει επικοινωνία από θύρα εξόδου προς θύρα εισόδου.

Παράδειγμα σύνταξης [17]:

```
module CompoundModule
    parameters:
      //...
    gates:
      //...
    submodules:
      //...
    connections:
        node1.output --> node2.input;
      //...
endmodule
```

Σε περιπτώσεις που θέλουμε να δηλώσουμε συνδέσεις ανάμεσα σε διανύσματα θυρών είναι δυνατό να χρησιμοποιήσουμε την έκφραση της αύξησης '++' ώστε να αυξάνουμε το μέγεθος του διανύσματος με κάθε σύνδεση που ορίζουμε. Έτσι μπορούμε χωρίς να έχουμε ορίσει το μέγεθος του διανύσματος θυρών στον τομέα "gatesizes:" να πραγματοποιήσουμε πολλαπλές συνδέσεις κατά βούληση. Κάθε σύνδεση βέβαια ορίζεται στατικά και κάθε θύρα χρησιμοποιείται μόνο σε μια σύνδεση. Δηλαδή το κέρδος αυτής της ιδιότητας είναι ότι δεν ορίζουμε το μέγεθος των πινάκων θυρών, και τίποτα παραπάνω.

Παράδειγμα [17]:

```
simple Node
    gates:
        in: in[]; // διάνυσμα θυρών εισόδου, χωρίς αρχικό μέγεθος
        out: out[]; // διάνυσμα θυρών εξόδου, χωρίς αρχικό μέγεθος
endsimple

module SmallNet
    submodules:
        node: Node[2]; // διάνυσμα τριών απλών μονάδων
    connections:
        node[0].out++ --> node[1].in++; // προσθέτει μια θύρα εξόδου στο node[0] και μια εισόδου στο node[1] και τις συνδέει
        node[0].in++ <-- node[2].out++; // προσθέτει μια θύρα εισόδου στο node[0] και μια εξόδου στο node[2] και τις συνδέει

        node[1].out++ --> node[2].in++; // προσθέτει μια θύρα εξόδου στο node[1] και μια εισόδου στο node[2] και τις συνδέει

        node[2].in++ <-- node[0].out++; // προσθέτει μια θύρα εισόδου στο node[2] και μια εξόδου στο node[0] και τις συνδέει
endmodule
```

Στο παραπάνω παράδειγμα οι θύρες που δημιουργούνται είναι:

	node[0]	node[1]	node[2]	Σύνολο
in:	1	1	2	4
out:	2	1	1	4

Δημιουργήσαμε δηλαδή διαφορετικά μεγέθη θυρών για κάθε υπομονάδα, παρότι προέρχονται όλες από τον ίδιο τύπο μονάδας. Η μονάδα node[1] για παράδειγμα έχει δύο θύρες, ενώ οι node[0] και node[2] έχουν από τρεις. Συνολικά οι θύρες εισόδου είναι όσες και οι θύρες εξόδου, κάτι αναμενόμενο. Αυτό που έχει μεγάλη σημασία είναι να είναι όλες οι θύρες συνδεδεμένες μεταξύ τους με συνδέσεις, εκτός αν ορίζεται αλλιώς όπως αναλύεται σε επόμενη παράγραφο [§2.6.1.4.2γ].

2.6.1.4.1 Ιδιότητες Σύνδεσης & Ορισμός Καναλιών

Μια σύνδεση είναι δυνατό να επηρεάζεται από τρεις προαιρετικές παραμέτρους [17]:

(1) Propagation Delay : Χρόνος Αναμετάδοσης: αντιπροσωπεύει το χρόνο που καθυστερεί ένα μήνυμα όταν περνάει από τη συγκεκριμένη σύνδεση.

(2) Bit Error Rate : Ρυθμός Σφάλματος: καθορίζει την πιθανότητα να μην μεταδίδεται σωστά ένα bit πληροφορίας.

(3) Data Rate: Ρυθμός Δεδομένων: ορίζει την ταχύτητα της σύνδεσης σε bits/sec και χρησιμοποιείται για τον υπολογισμό του χρόνου μετάδοσης ενός πακέτου. Ο χρόνος αυτός υπολογίζεται από την αποστολή του 1ου bit ενός πακέτου μέχρι την λήψη του τελευταίου bit από τον παραλήπτη.

Κάθε σύνδεση στο Omnet++ είναι δυνατό να έχει τις δικές τις παραμέτρους και αντίστοιχα τιμές. Αυτές είτε ορίζονται στην δήλωση της σύνδεσης είτε ενσωματώνονται σε μια νέα δήλωση που ονομάζεται Κανάλι Επικοινωνίας (Channel).

α) Ορισμός παραμέτρων στη δήλωσης της σύνδεσης:

Έστω ότι εξετάζουμε την σύνδεση ανάμεσα σε δύο απλές μονάδες, node1 και node2. Η σύνδεση από το node1 προς το node2 χωρίς παραμέτρους ορίζεται [17]:

```
node1.outGate --> node2.inGate;
```

Όταν θέλουμε να ορίσουμε τις παραμέτρους σύνδεσης απευθείας στην δήλωσή της, ακολουθούμε την εξής σύνταξη [17]:

```
node1.outGate --> error 1e-9 delay 0.001 datarate 128000 -->
node2.inGate;
```

Ουσιαστικά περικλείουμε με βέλη τις παραμέτρους και τις παρεμβάλλουμε ανάμεσα στις δύο θύρες επικοινωνίας. Η δήλωση μιας παραμέτρου είναι προαιρετική. Επίσης δεν παίζει κάποιο ρόλο η σειρά με την οποία τις δηλώνουμε.

β) Ορισμός Καναλιού Επικοινωνίας:

Η δήλωση ενός καναλιού επικοινωνίας γίνεται στη NED γλώσσα ως εξής [17]:

```
channel LeasedLine
    delay 0.0018  // sec
    error 1e-8
    datarate 128000  // bit/sec
endchannel
```

Και οι τρεις παράμετροι είναι προαιρετικές. Εφόσον όμως τις δηλώσουμε πρέπει να τους δώσουμε και τιμή. Η δήλωση μπορεί να είναι στο ίδιο αρχείο ned με το σύνθετη μονάδα που θα χρησιμοποιήσει το κανάλι ή και σε διαφορετικό αρχείο, φτάνει να γίνει εισαγωγή του με την οδηγία "import". Το όνομα του καναλιού κατά σύμβαση ξεκινά με κεφαλαίο γράμμα. Ένα κανάλι χρησιμοποιείται στην δήλωση μιας σύνδεσης ως εξής [17]:

```
node1.outGate --> LeasedLine --> node2.inGate;
```

Όπως φαίνεται το όνομα του καναλιού αξιοποιείται στην δήλωση μιας σύνδεσης με τον ίδιο τρόπο που χρησιμοποιούμε και τις παραμέτρους του ανεξάρτητα. Είναι φανερό ότι είναι πολύ πιο βολικό να ορίσουμε πολλαπλά Κανάλια επικοινωνίας με διαφορετικές παραμέτρους, παρά σε κάθε σύνδεση να ορίζουμε εκ νέου τις παραμέτρους.

2.6.1.4.2 Συνδέσεις με επαναλήψεις, με συνθήκη & χωρίς έλεγχο

Είναι δυνατό να δηλώσουμε συνδέσεις κατά πολλούς τρόπους στη NED δήλωση μιας σύνθετης μονάδας. Οι επιλογές που δίνονται ευνοούν την παραμετροποίησης της σύνδεσης και τη δυναμική κατασκευή πολλαπλών συνδέσεων ανάμεσα σε πολλές μονάδες. Επίσης δίνεται και η ελευθερία του ορισμού θυρών χωρίς σύνδεση [17].

α) Συνδέσεις με επανάληψη (loop connections):

Αν έχουμε διανύσματα υπομονάδων ή θυρών σε μια σύνθετη μονάδα, είναι εφικτό να δημιουργήσουμε περισσότερες από μια συνδέσεις με μια μόνο δήλωση. Ο όρος για την δήλωση αυτή είναι 'πολλαπλή δήλωση'. Μια πολλαπλή δήλωση γίνεται με ένα for statement.

Παράδειγμα σύνταξης [17]:

```
for i=0..4 do
    node1.outGate[i] --> node2[i].inGate
endfor;
```

Η παραπάνω δήλωση δημιουργεί πέντε συνδέσεις. Συγκεκριμένα συνδέει τις πέντε εξόδους της μονάδας node1 με τις εισόδους από πέντε μονάδες node2. Είναι δυνατό να δημιουργήσουμε πολλαπλές συνδέσεις με 'φωλιασμένες επαναλήψεις' (nested loops), οι οποίες καθορίζονται από το πλήθος των παραμέτρων σε ένα for statement. Η σειρά των παραμέτρων καθορίζει το φώλιασμα, με την πρώτη να αποτελεί την πιο εξωτερική επανάληψη, και την τελευταία την πιο εσωτερική.

Παράδειγμα σύνταξης [17]:

```
for i=0..4, j=0..3, k=0..5 do
    //...
endfor
```

β) Συνδέσεις με έλεγχο συνθήκης (conditional connections):

Η δημιουργία συνδέσεων μπορεί να γίνει υπό συνθήκες, χρησιμοποιώντας if statement. Η συνθήκη ελέγχεται και αν ισχύει δημιουργείται η σύνδεση.

Παράδειγμα σύνταξης [17]:

```
for i=0..n do
    node1.outGate[i] --> node2[i].inGate if i%2==0;
endfor;
```

γ) Τροποποίηση 'μη ελέγχου' (nocheck modifier):

Κατά σύμβαση ο NED compiler απαιτεί όλες οι θύρες των μονάδων να είναι συνδεδεμένες. Αυτό είναι δυνατό να μην βολεύει σε μερικά σενάρια, όπου τυχόν να θέλουμε να έχουμε ελεύθερες θύρες. Για την περίπτωση αυτή μπορούμε να χρησιμοποιήσουμε μια τροποποιημένη δήλωση για τον τομέα συνδέσεων μιας σύνθετης μονάδας, την "connections nocheck:"

Παράδειγμα σύνταξης [17]:

```
module RandomConnections
    parameters: //..
    gates: //..
    submodules: //..
    connections nocheck:
        for i=0..n-1, j=0..n-1 do
            node[i].out[j] --> node[j].in[i] if
uniform(0,1)<0.3;
        endfor;
endmodule
```

2.7 Ορισμοί Δικτύου (Network Definitions)

Οι προσομοιώσεις στο Omnet++ απαιτούν ένα περιβάλλον προσομοίωσης, το οποίο θα ενσαρκώνει το μοντέλο του δικτύου που θέλουμε να προσομοιώσουμε. Μέσα σε αυτό το μοντέλο θα ορίζονται οι απλές και σύνθετες μονάδες που αντιπροσωπεύουν τα λειτουργικά και εννοιολογικά στοιχεία του δικτύου. Προκειμένου να υλοποιήσουμε κάτι τέτοιο πρέπει να το δηλώσουμε στη NED γλώσσα με τον ορισμό του δικτύου (network definition) [17].

Το δίκτυο είναι η ενσάρκωση ενός ήδη υπαρκτού και ορισμένου τύπου μονάδας. Συνήθως για δίκτυα χρησιμοποιούμε σύνθετες μονάδες, αφού ουσιαστικά θέλουμε να έχουν το ρόλο του 'container' άλλων υπομονάδων και όχι ρόλο λειτουργικό όπως έχει μια απλή μονάδα που υλοποιείται σε C++. Είναι δυνατό να έχουμε πολλαπλούς ορισμούς δικτύων σε ένα ned αρχείο και να επιλέγουμε τον επιθυμητό προς εκτέλεση μέσα από το omnetpp.ini αρχείο ρυθμίσεων. Η σύνταξη της δήλωσης ενός δικτύου είναι παρόμοια με αυτή μιας σύνθετη μονάδας.

Παράδειγμα σύνταξης [17]:

```
network wirelessLAN: WirelessLAN
    parameters:
        numUsers=10,
        httpTraffic=true,
        ftpTraffic=true,
        distanceFromHub=truncnormal(100,60);
endnetwork
```

Στο παραπάνω παράδειγμα WirelessLAN είναι το όνομα ενός ήδη δηλωμένου τύπου σύνθετης μονάδας. Μόνο τύποι μονάδων χωρίς θύρες μπορούν χρησιμοποιηθούν σαν δίκτυα. Επιπλέον στη δήλωση ενός δικτύου αποδίδονται και οι τιμές των παραμέτρων του τύπου μονάδας που ενσαρκώνει κατά τα γνωστά.

2.8 Αρχείο Ρυθμίσεων omnetpp.ini

Οι προσομοιώσεις στο Omnet++ ελέγχονται από το αρχείο ρυθμίσεων omnetpp.ini. Στο αρχείο αυτό αποδίδονται οι τιμές των παραμέτρων των μονάδων και ορίζονται οδηγίες για τον τρόπο που θα εκτελείται η προσομοίωση. Κάθε γραμμή περιέχει μια ρύθμιση. Γραμμές που ξεκινούν με # είναι σχόλια. Σχόλια επιτρέπονται και στο τέλος μιας ρύθμισης.

Οι ρυθμίσεις στο αρχείο omnetpp.ini ομαδοποιούνται σε τομείς. Αυτοί μπορεί να είναι [17]:

- `[General]`
  Περιέχει ρυθμίσεις που εφαρμόζονται για όλους τους τομείς

- `[Run 1],[Run 2],...`
  Τομείς όπου δηλώνονται ξεχωριστές ρυθμίσεις για κάθε εκτέλεση.

- `[Cmdenv]`
  Τομέας που περιέχει ρυθμίσεις για εκτέλεση χωρίς γραφικό περιβάλλον

- `[Tkenv]`
  Τομέας που περιέχει ρυθμίσεις για εκτέλεση στο γραφικό περιβάλλον Tkenv

- `[Parameters]`
  Περιέχει τιμές για τις παραμέτρους που δεν τους έχει αποδοθεί τιμή στο αρχείο NED του ορισμού τους

- `[OutVectors]`
  Περιέχει ρυθμίσεις για τη συλλογή στατιστικών

Ένα αρχείο omnetpp.ini μπορεί να περιλαμβάνει τις εγγραφές ενός άλλου αρχείου με την οδηγία include:

```
include parameters.ini
```

Αυτό είναι ιδιαίτερα βολικό προκειμένου να διατηρούμε οργανωμένες τις ρυθμίσεις της προσομοίωσης κατά κατηγορία. Για παράδειγμα όλες οι τιμές συντεταγμένων για τους κόμβους των δικτύων που εξετάζονται στην εργασία είναι αποθηκευμένες σε ξεχωριστά αρχεία *.ini, ώστε να αξιοποιούνται σε πολλά δίκτυα [17].

Σημαντικό είναι να δηλώνουμε στο αρχείο omnetpp.ini, και συγκεκριμένα στον τομέα `[General]`, τα αρχεία .ned που θέλουμε να χρησιμοποιήσουμε για τον ορισμό του δικτύου:

```
[General]
preload-ned-files = *.ned
```

Αυτή η ενέργεια είναι απαραίτητη, εκτός αν έχουμε δημιουργήσει C++ κώδικα από τα .ned αρχεία ρυθμίσεων και των έχουμε συμπεριλάβει στο χτίσιμο του προσομοιωτή. Είναι δυνατό να φορτώνουμε αυτά τα αρχεία και από λίστα, δηλαδή αρχείο στο οποίο κάθε ned αρχείο ρυθμίσεων είναι τυπωμένο με τη σχετική διαδρομή του από από το omnetpp.ini, ή την απόλυτη διαδρομή στο σύστημα αρχείων [17].

```
 [General]
preload-ned-files = *.ned @../nedfiles.lst
```

Σε περίπτωση που αποδίδονται τιμές σε παραμέτρους και στη NED σύνταξη και στο omnetpp.ini, επικρατεί η πρώτη. Έχει συνεπώς σημασία στους NED ορισμούς να δίνουμε τιμές σε σταθερές παραμέτρους και στο omnetpp.ini σε μεταβλητές της προσομοίωσης.

Οι τιμές παραμέτρων των μονάδων αποδίδονται στον τομέα `[Parameters]`. Δηλώνονται με την ιεραρχία τους στο δίκτυο ή τη πλήρη διαδρομή (full path), όπως αλλιώς λέγεται. Αυτή αποτελείται από τα ονόματα των γονικών μονάδων (από το κατώτερο επίπεδο ιεραρχίας μέχρι και τη μονάδα που ανήκει η παράμετρος) χωρισμένα με τελεία, συν το όνομα της παραμέτρου [17]:

```
[Parameters]
sim.numNodes = 15
sim.node[*].bandwidth = 100
sim.node[5].label = "home"
```

Σε περίπτωση που μια παράμετρος δηλώνεται περισσότερες από μία φορές η πρώτη τιμή είναι αυτή που θα υπερισχύσει. Αυτό συναντάται συνήθως όταν δηλώνουμε μαζικά την τιμή μιας παραμέτρου διανύσματος μονάδων με αστερίσκο (wildcard), και ειδικές τιμές της με τη πλήρη διαδρομή. Οι ειδικές τιμές πρέπει να προηγούνται των μαζικών, αλλιώς αγνοούνται. Για τα διανύσματα μονάδων επιτρέπεται επίσης και η δήλωση αριθμητικού εύρους για την απόδοση τιμών[17]:

```
[Parameters]
sim.node[0].bandwidth = 100
sim.node[1..3].bandwidth = 20
sim.node[8..].bandwidth = 50
sim.node[*].bandwidth = 10 #the rest of the nodes
```

Πέρα από τον απλό μονό αστερίσκο (*) υπάρχει και ο διπλός (**). Ενώ ο μονός αντικαθιστά μια μονάδα ή παράμετρο στη διαδρομή απόδοσης μια τιμής, ο διπλός αντικαθιστά ολόκληρη τη διαδρομή μέχρι την πρώτη στατική δήλωση που τον ακολουθεί [17].

```
[Parameters]
sim.subnet1.node[*].bandwidth = 100
*.subnet1.node[*].bandwidth = 100 #all networks
**.node[*].bandwidth = 100 #all networks and subnets
```

Για να τρέχουμε διαφορετικά σενάρια στην προσομοίωση, διαχωρίζουμε τις ξεχωριστές τους ρυθμίσεις σε `[Run ?]` τομείς. Αυτοί έχουν τον ίδιο ρόλο με το τομέα παραμέτρων `[Parameters]`, μόνο που φορτώνεται ένας σε κάθε προσομοίωση. Στην εκκίνηση ο προσομοιωτής μας ρωτάει πιο σενάριο να εκτελέσει [17].

Οι παραπάνω ρυθμίσεις είναι οι κυριότερες και αυτές που αξιοποιούνται στην εργασία. Πιο αναλυτική παρουσίαση του omnetpp.ini διατίθεται στο εγχειρίδιο του Omnet++ [17]

Castalia

ΑΝΑΛΥΣΗ:

Το Castalia είναι ένας προσομοιωτής ασυρμάτων δικτύων αισθητήρων (WSN) που υλοποιείται βάσει του Omnet++ framework, είναι δηλαδή ένας προσομοιωτής του Omnet++. Το Castalia είναι ένας προσομοιωτής εξειδικευμένος στη μελέτη ασυρμάτων δικτύων αισθητήρων, σε αντίθεση με το Omnet++ που είναι ένα σύνολο βιβλιοθηκών για τη κατασκευή παντός τύπου προσομοιωτών δικτύων.

3. Castalia: Ανάλυση Δομής

Ως ένας προσομοιωτής εξειδικευμένος στο αντικείμενο των ασυρμάτων δικτύων αισθητήρων, το Castalia διαθέτει μονάδες για την αναπαράσταση κάθε επί μέρους ενέργειας/στοιχείου ενός τέτοιου δικτύου. Επιπλέον προσεγγίζει το φαινόμενο της επικοινωνίας χωρίς να περιορίζεται στην ασύρματη μετάδοση και μόνο, αλλά δίνοντας μεγάλη σημασία στην υλοποίηση της σε MAC επίπεδο, διαθέτοντας ξεχωριστή μονάδα για το σκοπό αυτό. Επίσης αναλύει τη διαδικασία αποστολής μηνυμάτων σε φυσικό επίπεδο, υλοποιώντας για κάθε κόμβο ένα σύνολο μονάδων που αντιπροσωπεύουν τα λειτουργικά του μέρη. Έτσι υπεισέρχεται στην προσομοίωση ο ίδιος ο αισθητήρας της συσκευής, η μονάδα δρομολόγησης της πληροφορίας, η κεντρική μονάδα λειτουργίας, η μονάδα εκπομπής ραδιοκυμάτων κλπ.

Στο Castalia είναι αξιοσημείωτος ο τρόπος με τον οποίο δρουν τα μηνύματα στην προσομοίωση και την αλληλεπίδραση των μονάδων. Αυτά χωρίζονται σε γενικές γραμμές σε δύο ειδών μηνύματα:

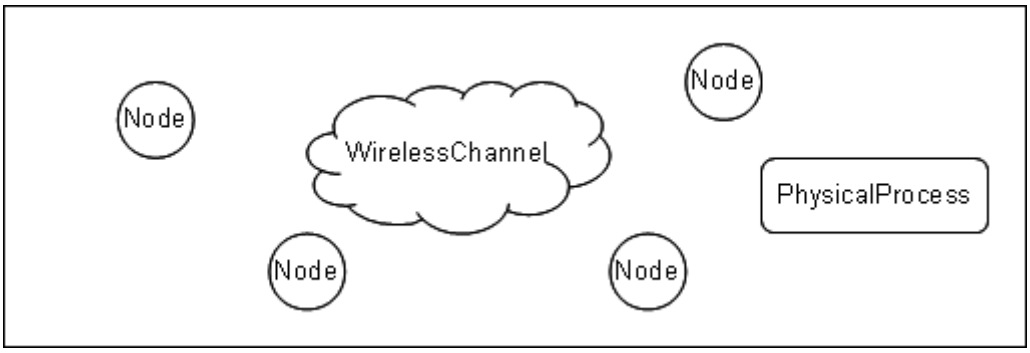
- ➔ Μηνύματα Δεδομένων
- ➔ Μηνύματα Ελέγχου

Τα μηνύματα δεδομένων είτε αντιπροσωπεύουν το πραγματικό μήνυμα που αποστέλλεται από ένα κόμβο σε ένα άλλο, είτε είναι μήνυμα που ενσωματώνει το πρώτο σε πακέτο δεδομένων δικτύου. Αντίστοιχα και στα μηνύματα ελέγχου έχουμε πληθώρα κατηγοριών ελέγχων, όπως έλεγχος λειτουργίας μιας μονάδας από μια άλλη, προγραμματισμός εσωτερικής λειτουργίας μονάδας, ανταπόκριση σε μηνύματα ελέγχου κλπ. Η πολυδιάστατη φύση των μηνυμάτων στο Castalia γίνεται πιο κατανοητή στην ανάλυση των μονάδων του και των αλληλεπιδράσεών τους στις επόμενες παραγράφους.

3.a Δίκτυο και Μονάδες

Το Castalia ως προσομοιωτής του Omnet++ ακολουθεί την ίδια δομή και οργάνωση των μερών του σε σύνθετες και απλές μονάδες. Το μοντέλο της προσομοίωσης υλοποιείται στο δίκτυο 'SN', το οποίο βασίζεται στη σύνθετη μονάδα 'SensorNetwork'. Αυτή με τη σειρά της περιέχει τρεις υπομονάδες:

- Ασύρματο Κανάλι (WirelessChannel simple module)
- Φυσική Διεργασία (PhysicalProcess simple module)
- Κόμβος (Node compound module)



Σχήμα 3.1

Όπως εύκολα μπορεί κανείς να συμπεράνει η μονάδα Ασύρματο Κανάλι (WirelessChannel) προσομοιώνει το μέσο επικοινωνίας μεταξύ των μονάδων του δικτύου. Μπορούμε να πούμε ότι αποτελεί το μοντέλο επικοινωνίας του προσομοιωτή. Η μονάδα Φυσικής Διεργασίας (PhysicalProcess) αντιπροσωπεύει την φυσική ποσότητα που παρακολουθούν οι μονάδες (θερμοκρασία, υγρασία κλπ) και τις μεθόδους μετάδοσης της πληροφορίας της. Τέλος η σύνθετη μονάδα Node αντιπροσωπεύει τον κάθε κόμβο, δηλαδή την θέση και τη συσκευή του αισθητήρα. Οι κόμβοι ενός δικτύου συνήθως ορίζονται ως διάνυσμα της σύνθετης μονάδας Node.

3.1 Κόμβος (Node compound module)

Η σύνθετη μονάδα Node αντιπροσωπεύει τους κόμβους του δικτύου, δηλαδή τους αισθητήρες του, σαν συσκευές/αντικείμενα σε κάποια θέση. Ως σύνθετη μονάδα δεν φέρει λειτουργικό ρόλο, παρά μόνο αποτελεί δοχείο για τις υπομονάδες που αναπαριστούν τα λειτουργικά μέρη μιας συσκευής αισθητήρα. Ένας κόμβος αποτελείται από απλές και σύνθετες υπομονάδες:

- Μονάδα Εφαρμογής : Application (simple module)
- Μονάδα Συσκευής Αισθητήρα : Sensor_Device_Manager (simple module)
- Μονάδα Επικοινωνίας : Communication (compound module)
- Μονάδα Διαχείρισης Πόρων : Resource_Manager (simple module)

Αυτές ορίζονται στη NED δήλωση των υπομονάδων του κόμβου ως εξής:

```
submodules:
 nodeResourceMgr: ResourceGenericManager
 nodeSensorDevMgr : SensorDevMgrModule
   gatesizes:
     fromNodeContainerModule[sizeof(toPhysicalProcess)],
     toNodeContainerModule[sizeof(toPhysicalProcess)];
 networkInterface : CommunicationModule;
 nodeApplication : appModuleName like  ApplicationGenericModule;
```

[Castalia-1.3/src/Node/Node.ned]

Από τις υπομονάδες του κόμβου όλες ορίζονται στατικά εκτός από τη μονάδα Εφαρμογής, η οποία ορίζεται με παράμετρο τύπου μονάδας. Επιπλέον ορίζεται το μέγεθος του διανύσματος θυρών εισόδου και εξόδου της μονάδας συσκευής αισθητήρα. Η υπομονάδα αυτή έχει ένα ζεύγος θυρών για κάθε φυσική διεργασία που υλοποιείται στο δίκτυο.

Κάθε κόμβος έχει τέσσερις κύριες παραμέτρους:

```
          xCoor: x συντεταγμένη θέσης
          yCoor: y συντεταγμένη θέσης
          zCoor: z συντεταγμένη θέσης
appModuleName: ο τύπος μονάδας που θα υλοποιεί την υπομονάδα Εφαρμογής
```

[Castalia-1.3/src/Node/Node.ned]

Οι κόμβοι συνδέονται μόνο με το Ασύρματο Κανάλι Επικοινωνίας και τις μονάδες Φυσικής Διεργασίας του δικτύου. Για το σκοπό αυτό ορίζονται οι εξής θύρες:

```
gates:
   out:    toWirelessChannel, toPhysicalProcess[];
    in:    fromWirelessChannel, fromPhysicalProcess[];
```

[Castalia-1.3/src/Node/Node.ned]

Οι θύρες αυτές ουσιαστικά είναι πύλες μετάβασης από το ένα επίπεδο ιεραρχίας στο άλλο. Συγκεκριμένα τα μηνύματα διοχετεύονται από τις υπομονάδες του δικτύου στις υπομονάδες κάθε κόμβου και το αντίθετο. Επομένως οι θύρες αυτές συνδέονται οι μεν εξόδου με θύρες εξόδου υπομονάδων και οι εισόδου με θύρες εισόδου υπομονάδων. Αυτό φαίνεται χαρακτηριστικά στις παρακάτω συνδέσεις που ορίζονται στη μονάδα του κόμβου:

```
connections nocheck:
       networkInterface.toNodeContainerModule  -->
     toWirelessChannel;
       fromWirelessChannel  -->
     networkInterface.fromNodeContainerModule;
       nodeApplication.toCommunicationModule  -->
     networkInterface.fromApplicationModule;
       nodeApplication.toSensorDeviceManager  -->
     nodeSensorDevMgr.fromApplicationModule;
       networkInterface.toApplicationModule  -->
     nodeApplication.fromCommunicationModule;
     //...
     for i=0..sizeof(toPhysicalProcess)-1 do
       fromPhysicalProcess[i] -->
     nodeSensorDevMgr.fromNodeContainerModule[i];
       nodeSensorDevMgr.toNodeContainerModule[i] -->
     toPhysicalProcess[i];
     endfor;
     //...
```

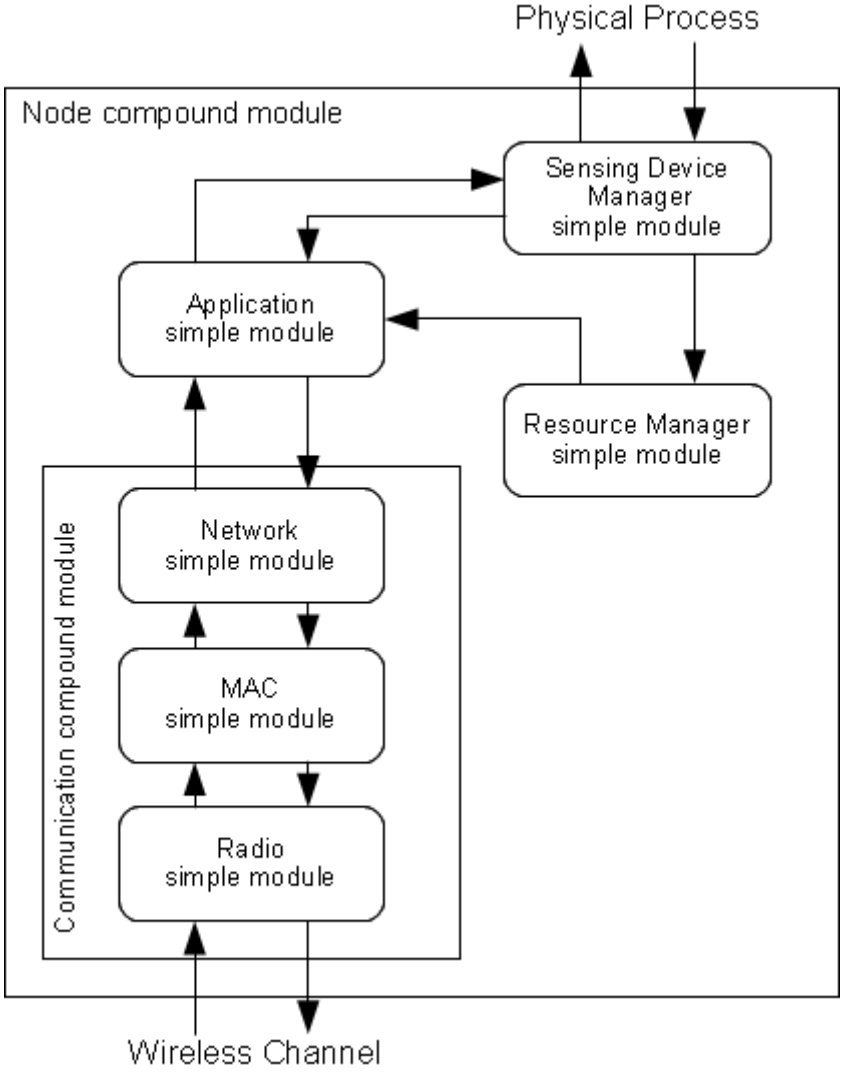
[Castalia-1.3/src/Node/Node.ned]

Χαρακτηριστική είναι η δήλωση των συνδέσεων ανάμεσα στις θύρες της υπομονάδας Συσκευή Αισθητήρα (SensorDevMgr) και τις θύρες του κόμβου που συνδέονται με τις Φυσικές Διεργασίες. Δημιουργείται με μια for επανάληψη ένα ζεύγος θυρών για κάθε Φυσική Διεργασία. Τα ζεύγη είναι για ίδιου τύπου θύρες, δηλαδή:

```
θύρα εισόδου υπομονάδας <-- θύρα εισόδου κόμβου
θύρα εξόδου υπομονάδας --> θύρα εξόδου κόμβου
```

Έτσι όταν η Συσκευή Αισθητήρα στέλνει αιτήματα στη Φυσική Διεργασία, αυτά περνάνε από την θύρα εξόδου του κόμβου για τη συγκεκριμένη μονάδα, και η απάντηση επιστρέφεται από την αντίστοιχη θύρα εισόδου του.

Η διάταξη των μονάδων και οι συνδέσεις εντός του κόμβου φαίνονται χαρακτηριστικά στο παρακάτω σχήμα [9]:



Σχήμα 3.2

3.1.2 Μονάδα Εφαρμογής (Application simple module)

Η μονάδα εφαρμογής είναι αυτή που καθορίζει την λειτουργία κάθε κόμβου. Ορίζεται με παράμετρο τύπου μονάδας, και συνεπώς μπορούμε να ενσωματώνουμε διαφορετική μονάδα σε κάθε κόμβο. Αυτή εκκινεί την λειτουργία του μέσω των selfMessages, στέλνει μηνύματα στις άλλες μονάδες του και επεξεργάζεται τις απαντήσεις που λαμβάνει. Επίσης είναι ο τελικός αποδέκτης των πακέτων δεδομένων που αποστέλλονται μέσω του δικτύου από κόμβο σε κόμβο.

Η μονάδα επικοινωνεί απευθείας με τις υπόλοιπες του κόμβου, και μόνο με αυτές. Οι θύρες που υποστηρίζουν τις συνδέσεις της είναι:

```
gates:
   out:    toCommunicationModule, toSensorDeviceManager;
    in:    fromCommunicationModule, fromSensorDeviceManager,
           fromResourceManager;
```

[Castalia-1.3/src/Node/Application/ApplicationGenericModule.ned]

Μηνύματα προς άλλους κόμβους ή μηνύματα από την φυσική διεργασία που παρακολουθεί ο αισθητήρας, δρομολογούνται μέσω των Communication και Sensor_Device_Manager μονάδων αντίστοιχα. Κατά την εκκίνηση της μονάδας μέσω της *initialize()* μεθόδου αποστέλλονται και τα μηνύματα εκκίνησης στις υπόλοιπες υπομονάδες του κόμβου.

Η μονάδα Εφαρμογής είναι απλή μονάδα και ως εκ τούτου υλοποιείται σε C++. Οι υπάρχουσες υλοποιήσεις μονάδων Εφαρμογής διαθέτουν τις εξής κύριες μεθόδους:

```
virtual void initialize();
virtual void handleMessage(cMessage *msg);
virtual void finish();

void send2NetworkDataPacket(const char *destID, int data,
                            int pckSeqNumber);
void requestSampleFromSensorManager();
```

[Castalia-1.3/src/Node/Application/templateApplication/template_ApplicationModule.h]

3.1.2.1 *void initialize()*

Στη μέθοδο εκκίνησης γίνεται ανάκτηση των παραμέτρων της και των παραμέτρων θέσης της γονικής μονάδας του κόμβου και, αποδίδεται η τιμή τους σε μεταβλητές της τάξης της μονάδας.

```
int self; // the node's ID
double self_xCoo;
double self_yCoo;
//...
```

[Castalia-1.3/src/Node/Application/templateApplication/template_ApplicationModule.h]

```
self = parentModule()->index();
self_xCoo = parentModule()->par("xCoor");
self_yCoo = parentModule()->par("yCoor");
//...
```

[Castalia-1.3/src/Node/Application/templateApplication/template_ApplicationModule.cc]

Επίσης στέλνονται τα μηνύματα εκκίνησης (STARTUP) προς την μονάδα Συσκευή Αισθητήρα και την υπομονάδα Δικτύωση (Network) της σύνθετης υπομονάδας Επικοινωνίας του κόμβου. Αποστέλλεται επίσης και το πρώτο selfMessage προς την ίδια την Εφαρμογή.

3.1.2.2 *void handleMessage(cMessage *msg)*

Η μέθοδος διαχείρισης μηνυμάτων της Εφαρμογής, εξετάζει το είδος τους μηνύματος και αναλόγως ενεργεί. Το είδος ενός μηνύματος ορίζεται με ακέραιο και επιστρέφεται με τι μέθοδο *kind()* της τάξης cMessage:

cMessage msg;*
int msgKind = msg->kind();

Οι κύριοι τύποι μηνύματος που διαχειρίζεται η Εφαρμογή και οι αντίστοιχες δράσεις είναι:

➔ APP_NODE_STARTUP

Το μήνυμα αυτού του τύπου είναι selfMessage που στέλνει η ίδια η μονάδα στον εαυτό της για να κληθεί η *void handleMessage(cMessage *msg)* μέθοδος και να εκκινήσει η μονάδα. Η προτεινόμενη λύση είναι να στέλνεται μήνυμα αίτησης μέτρησης φυσικής ποσότητας στη μονάδα Συσκευής Αισθητήρα. Άλλη συνηθισμένη πρακτική είναι να στέλνονται νέα selfMessages με τύπο APP_TIMER_*, τα οποία να ενεργοποιούν διάφορες δράσεις και λειτουργίες κατά την παραλαβή τους από την *void handleMessage(cMessage *msg)*.

➔ APP_SELF_REQUEST_SAMPLE

Το μήνυμα αυτό είναι selfMessage το οποίο έχει σταλεί από τον κόμβο προκειμένου να γίνει αίτηση δείγματος των φυσικών ποσοτήτων που παρακολουθεί ο κόμβος, από τη Συσκευή Αισθητήρα. Αυτή η αίτηση ενεργοποιείται με τη μέθοδο *requestSampleFromSensorManager()*.

➔ APP_DATA_PACKET

Όταν ένας κόμβος στέλνει μηνύματα σε έναν άλλο, στην ουσία το μήνυμα που στέλνει δεν φτάνει το ίδιο στον παραλήπτη. Όλα τα μηνύματα, αφού διαχειριστούν από τη *void handleMessage(cMessage *msg)* μέθοδο κάθε κόμβου, στο τέλος καταστρέφονται. Το μήνυμα που κάθε φορά φτάνει στη θύρα μιας μονάδας, είναι είτε τελείως καινούργιο, είτε αντίγραφο του αρχικού μηνύματος με κάποια τροποποίηση, συνήθως στο είδος του. Ένα μήνυμα τύπου APP_DATA_PACKET έχει ουσιαστικά αποσταλεί στην Εφαρμογή από τη μονάδα Δικτύωσης (Network), η οποία το δημιούργησε αφού έλαβε μήνυμα από το δίκτυο. Το μήνυμα περιλαμβάνει συν τοις άλλοις τα στοιχεία του αποστολέα και την λαμβανόμενη ισχύ της εκπομπής στο δέκτη. Στην διαχείριση αυτού του πακέτου είναι δυνατό να υλοποιηθεί καταγραφή γειτονικών κόμβων και ποιότητα σύζευξης.

➔ APP_TIMER_*

Τα μηνύματα με τίτλο "χρονοδιακόπτη", είναι μια επιλογή για επιπλέον λειτουργίες αυτοδιαχείρισης της Εφαρμογής. Μπορούμε να εξετάζουμε διαφορετικές περιπτώσεις για κάθε μήνυμα τύπου 'χρονοδιακόπτη' (APP_TIMER_1, APP_TIMER_2, ...). Τα μηνύματα αυτά συνήθως προγραμματίζονται μέσα σε άλλες λειτουργίες διαχείρισης της *void handleMessage(cMessage *msg)* μεθόδου. Η χρήση και διαχείριση των μηνυμάτων αυτών δεν είναι υποχρεωτική.

➔ SDM_2_APP_SENSOR_READING

Το μήνυμα αυτό παραλαμβάνεται από την Εφαρμογή απευθείας από την Συσκευή Αισθητήρα (SensorDevMgr), σε απάντηση προηγούμενης αίτησης για δείγμα φυσικής ποσότητας. Εδώ υλοποιείται συλλογή στατιστικών για τα μηνύματα αυτού του τύπου.

➔ RESOURCE_MGR_OUT_OF_ENERGY

Το μήνυμα αυτό αποστέλλεται από τον Διαχειριστή ενέργειας προς όλες τις απλές μονάδες του κόμβου, εφόσον αυτός ξεμείνει από ενέργεια. Το μήνυμα οδηγεί κάθε μονάδα στο να απενεργοποιείται.

➔ RESOURCE_MGR_DESTROY_NODE

Αυτό το μήνυμα αποστέλλεται από τον Διαχειριστή ενέργειας προς όλες τις απλές μονάδες του κόμβου, και ορίζει διακοπή της λειτουργίας σε κάθε μονάδα.

➔ NETWORK_2_APP_FULL_BUFFER

Το μήνυμα αυτό αποστέλλεται στην Εφαρμογή από τη μονάδα Δικτύωσης (Network), όταν η προσωρινή μνήμη αποθήκευσης μηνυμάτων (buffer) στη μονάδα Ελέγχου Προσβασιμότητας Μέσου (MAC) είναι γεμάτη.

Αφού διαχειριστεί το μήνυμα η void handleMessage(cMessage *msg) το καταστρέφει και επιστρέφει στην προσομοίωση.

3.1.2.3 *void finish()*

Στη μέθοδο τερματισμού συνηθίζεται η εκτύπωση στατιστικών για τα δεδομένα που συνέλεξε η μονάδα Εφαρμογής.

3.1.2.4 *void send2NetworkDataPacket(const char *destID, int data, int pckSeqNumber)*

Η μέθοδος αποστολής μηνυμάτων στο δίκτυο των κόμβων. Στοιχεία εισόδου στη πρότυπη διατύπωσή της είναι η διεύθυνση του παραλήπτη (const char *destID), τα δεδομένα του μηνύματος (int data), και ο αριθμός σειράς (int pckSeqNumber). Η παράμετρος data μπορεί να έχει άλλο τύπο, αναλόγως του τύπου μηνύματος που χρησιμοποιούμε. Τα στοιχεία αυτά ενσωματώνονται στο μήνυμα και αυτό προωθείται στη μονάδα Δικτύωσης (Network) προκειμένου να ακολουθήσει την πορεία προς την παραλήπτη. Ενδεικτική χρήση της μεθόδου γίνεται στην *void handleMessage(cMessage *msg)* μέθοδο της Εφαρμογής connectivityMap:

```
case APP_TIMER_1:
{
      simtime_t now = simTime();
      int currentNodeTx = ((int)floor(now / txInterval_perNode))
                          %totalSNnodes;
      if( (self == currentNodeTx) && (packetsSent <
         SENT_PACKETS_PER_NODE_PER_LEVEL) )
      {
            send2NetworkDataPacket(BROADCAST, serialNumber, 1);
            serialNumber++;
            packetsSent++;
      }
      scheduleAt(now + DRIFTED_TIME(INTER_PACKET_SPACING),
                  new App_ControlMessage("timer1", APP_TIMER_1));
      break;
}
```

[Castalia-1.3/src/Node/Application /connectivityMapApplication/connectivityMap_ApplicationModule.cc]

Η αποστολή μηνυμάτων στο δίκτυο ενεργοποιείται εφόσον η Εφαρμογή παραλάβει μήνυμα τύπου "APP_TIMER_1" το οποίο αντιστοιχεί στον ακέραιο '1017'. Παρατηρούμε πως αφού σταλεί το μήνυμα προγραμματίζεται νέο μήνυμα ελέγχου APP_TIMER_1, για την επόμενη αποστολή στο δίκτυο.

3.1.2.5 void requestSampleFromSensorManager()

Η μέθοδος αυτή εκτελείται στην *void handleMessage(cMessage *msg)*, όταν παραλάβει μήνυμα τύπου APP_SELF_REQUEST_SAMPLE. Τότε αυτή δημιουργεί καί στέλνει νέο μήνυμα τύπου APP_2_SDM_SAMPLE_REQUEST απευθείας στη Συσκευή Αισθητήρα (SensorDevMgr), προκειμένου αυτή να απαντήσει στέλνοντας μέτρηση της Φυσικής Διεργασίας.

3.1.3 Μονάδα Συσκευής Αισθητήρα (Sensor_Device_Manager simple module)

Η μονάδα αυτή είναι ο διαμεσολαβητής ανάμεσα στην μονάδα Εφαρμογής και την μονάδα της φυσικής ποσότητας που παρακολουθεί ο αισθητήρας (PhysicalProcess module). Η τελευταία δεν ανήκει στη σύνθετη μονάδα του κόμβου, αλλά είναι στο ίδιο επίπεδο ιεραρχίας με αυτή και ανήκει στο δίκτυο. Η μονάδα Συσκευής Αισθητήρα εκκινείται από τη μονάδα Εφαρμογής, από την οποία δέχεται μηνύματα αίτησης δειγμάτων της φυσικής διεργασίας που παρακολουθεί ο κόμβος. Τότε στέλνει μήνυμα στη μονάδα της Φυσικής Διεργασίας ζητώντας μέτρησης της τιμής της φυσικής ποσότητας, και επιστρέφει την απάντηση στη μονάδα Εφαρμογής.

Η μονάδα συνδέεται με θύρες με τη μονάδα Εφαρμογής, με τη μονάδα Διαχείρισης Πόρων και με την σύνθετη μονάδα Node του κόμβου:

```
gates:
   out:    toApplicationModule, toNodeContainerModule[];
   in:     fromNodeContainerModule[], fromApplicationModule,
           fromResourceManager;
```

[Castalia-1.3/src/Node/Sensor_Device_Manager/SensorDevMgrModule.ned]

Η *void handleMessage(cMessage *msg)* μέθοδος διαχειρίζεται αρκετά είδη μηνυμάτων. Τα σημαντικότερα είναι:

➔ APP_2_SDM_SAMPLE_REQUEST

Το μήνυμα αυτό λαμβάνεται από τη μονάδα Εφαρμογής και αιτείται δείγμα από τη Φυσική Διεργασία. Η διαχείριση δημιουργεί μήνυμα τύπου PHY_SAMPLE_REQ, και το στέλνει στην θύρα εξόδου του κόμβου που συνδέεται με τη μονάδα της Φυσικής Διεργασίας.

➔ PHY_SAMPLE_REPLY

Εφόσον η μονάδα Φυσικής Διεργασίας (PhyProcess) λάβει μήνυμα τύπου PHY_SAMPLE_REQ, δημιουργεί μέτρηση για το φυσικό μέγεθος και το ενσωματώνει σε μήνυμα τύπου PHY_SAMPLE_REPLY, το οποίο και επιστρέφεται στη μονάδα Συσκευής Αισθητήρα. Όταν το μήνυμα ληφθεί στη *void handleMessage(cMessage *msg)*, διαβάζονται οι τιμές του και ενσωματώνονται σε νέο μήνυμα τύπου SDM_2_APP_SENSOR_READING, το οποίο και αποστέλλεται ευθέως στη μονάδα Εφαρμογής.

Προκειμένου να επικοινωνήσει η Συσκευή Αισθητήρα με τη μονάδα Φυσικής Διεργασίας, συνδέεται με τη σύνθετη μονάδα του κόμβου στα διανύσματα θυρών *toNodeContainerModule[]* και *fromNodeContainerModule[]*. Τα μεγέθη των διανυσμάτων αυτών ορίζονται βάσει του αριθμού των μονάδων Φυσικής Διεργασίας, ώστε για κάθε τέτοια μονάδα να υλοποιείται ένα ζεύγος θυρών εισόδου/εξόδου και οι αντίστοιχες συνδέσεις:

```
module Node
      //...
   gates:
      out: toWirelessChannel, toPhysicalProcess[];
      in: fromWirelessChannel, fromPhysicalProcess[];

   submodules:
      //...
      nodeSensorDevMgr: SensorDevMgrModule
         gatesizes:
            fromNodeContainerModule[sizeof(toPhysicalProcess)],
            toNodeContainerModule[sizeof(toPhysicalProcess)];
            //...

   connections nocheck:
      //...
      for i=0..sizeof(toPhysicalProcess)-1 do
         fromPhysicalProcess[i] -->
          nodeSensorDevMgr.fromNodeContainerModule[i];
         nodeSensorDevMgr.toNodeContainerModule[i] -->
          toPhysicalProcess[i];
      endfor;
      //...
endmodule:
```

[Castalia-1.3/src/Node/Node.ned]

3.1.4 Μονάδα Επικοινωνίας (Communication compound module)

Η μονάδα Επικοινωνίας των κόμβων αναλαμβάνει την επικοινωνία της μονάδας Εφαρμογής του κόμβου με το υπόλοιπο δίκτυο. Ως σύνθετη μονάδα δεν ενσωματώνει κάποια λειτουργία, και το ρόλο αυτό αναλαμβάνουν οι τρεις υπομονάδες του:

- Μονάδα Δικτύωσης (Network simple module)
- Μονάδα Ελέγχου Προσβασιμότητας Μέσου (MAC simple module)
- Μονάδα Ραδιοεπικοινωνίας (Radio simple module)

Η μονάδα Radio δηλώνεται στατικά, ενώ οι μονάδες Network και MAC με τη χρήση παραμέτρου τύπου υπομονάδας:

```
parameters:
      macModuleName: string,
      networkModuleName: string;

submodules:
      Radio: RadioModule;
      MAC: macModuleName like MacGenericModule;
      Network: networkModuleName like NetworkGenericModule
```

[Castalia-1.3/src/Node/Communication/CommunicationModule.ned]

Η μονάδα Network συνδέει τη μονάδα Εφαρμογής με την υπομονάδα MAC. Η MAC με τη σειρά της συνδέεται μόνο με την Radio υπομονάδα, η οποία συνδέεται μέσω της σύνθετης μονάδας επικοινωνίας και του κόμβου με το Ασύρματο Κανάλι (WChannel).

3.1.4.1 Μονάδα Δικτύωσης (Network simple module)

Η υπομονάδα Δικτύωσης της σύνθετης μονάδας Επικοινωνίας, περιλαμβάνει δυνατότητες δρομολόγησης (προαιρετικό στοιχείο) καθώς και το network frame, δηλαδή. τα στοιχεία αποστολέα/παραλήπτη, την κεφαλίδα του μηνύματος, το προηγούμενο και επόμενο βήμα κλπ. Δημιουργήθηκε από την ομάδα προγραμματιστών του Castalia όχι τόσο για τις ανάγκες των προσομοιώσεων με τις οποίες ασχολείται κυρίως αυτό το πρόγραμμα, αλλά για να υπάρχει σε όποιον θέλει να αξιοποιήσει τις δυνατότητες δρομολόγησης που εισάγει [9]. Ως εκ τούτου υπάρχουν τριών ειδών μονάδες:

- BypassRouting
- multipathRingsRouting
- simpleTreeRouting

Οι τύποι μονάδας Δικτύωσης multipathRingsRouting και simpleTreeRouting, υλοποιούν πρωτόκολλα δρομολόγησης, ενώ ο τύπος BypassRouting, επιτρέπει να εκτελούμε προσομοιώσεις παραβλέποντας αυτό το στοιχείο. Ακόμα και έτσι η μονάδα υφίσταται στην υλοποίηση του κόμβου και των συνδέσεων και εκτελεί χρέη δρομολόγησης, δεδομένου ότι τα μηνύματα πρέπει να προωθηθούν μέσω αυτής στο MAC και από εκεί στο Radio και το δίκτυο. Στην παρούσα εργασία δεν εφαρμόζεται δρομολόγηση, οπότε εξετάστηκε μόνο η μονάδα BypassRouting.

Οι κυριότεροι τύποι μηνυμάτων που διαχειρίζεται η μονάδα Δικτύωσης BypassRouting είναι:

➔ APP_DATA_PACKET

Ο τύπος αυτός μηνύματος στέλνεται από το την υπομονάδα Εφαρμογής του κόμβου με προορισμό το δίκτυο, δηλαδή γειτονικούς κόμβους. Η *void handleMessage(cMessage *msg)* μέθοδος της μονάδας δικτύωσης εξετάζει πρώτα την προσωρινή μνήμη (buffer) της μονάδας:

i. Αν αυτή δεν είναι γεμάτη (!BUFFER_IS_FULL):

Διαβάζονται τα στοιχεία του μηνύματος και ενσωματώνονται σε μήνυμα *Network_GenericFrame *newDataFrame* 50 χαρακτήρων (προεπιλογή). Αν το μήνυμα ενσωματωθεί σωστά, δηλαδή χωρέσει το APP_DATA_PACKET στο **newDataFrame*, τότε αποθηκεύεται σε προσωρινή μνήμη και προγραμματίζεται μήνυμα ελέγχου NETWORK_SELF_CHECK_TX_BUFFER για την ίδια μονάδα. Αν η ενσωμάτωση αποτύχει, τότε το μήνυμα ακυρώνεται και ενημερώνεται ο χρήστης για το γεγονός.

ii. Αν είναι γεμάτη (BUFFER_IS_FULL):

Δημιουργείται μήνυμα αναφοράς NETWORK_2_APP_FULL_BUFFER και στέλνεται στη μονάδα Εφαρμογής για να ενημερωθεί ότι δεν μπορεί να γίνει αποστολή και να μπει σε κατάσταση αναμονής και ακρόασης

➔ NETWORK_SELF_CHECK_TX_BUFFER

Όταν το μήνυμα αυτό παραληφθεί από τη μονάδα Δικτύωσης, γίνεται έλεγχος στην προσωρινή μνήμη. Αν αυτή δεν είναι άδεια (!BUFFER_IS_EMPTY), τότε στέλνεται το μήνυμα τύπου NETWORK_FRAME (Network_GenericFrame) που βρίσκεται σε αυτή στη MAC υπομονάδα και αδειάζει η προσωρινή μνήμη.

➔ MAC_2_NETWORK_FULL_BUFFER

Το μήνυμα αυτό επιστρέφεται στη Δικτύωση από τη MAC μονάδα, αν η προσωρινή μνήμη της MAC είναι γεμάτη. Όπως και σε παρόμοια περίπτωση προηγουμένως, στέλνεται μήνυμα αναφοράς στη μονάδα Εφαρμογής και ενημερώνεται ο χρήστης.

➔ NETWORK_FRAME

Το μήνυμα αυτό λαμβάνεται στη μονάδα Network από τη MAC. Στην ουσία πρόκειται για μήνυμα που παρέλαβε η μονάδα Radio από το Ασύρματο Κανάλι και έχει προωθηθεί από την MAC στη μονάδα Network. Το μήνυμα αυτό έχει ενσωματωμένο το αυθεντικό μήνυμα APP_DATA_PACKET, που εστάλη από την μονάδα Εφαρμογής του αποστολέα. Αυτό εξάγεται και προωθείται στη μονάδα Εφαρμογής του κόμβου, που είναι και ο τελικός παραλήπτης.

3.1.4.2 Μονάδα Ελέγχου Προσβασιμότητας Μέσου (MAC simple module)

Η μονάδα αυτή ελέγχει τη μετάδοση των μηνυμάτων στη μονάδα Ραδιοεπικοινωνίας από τη μονάδα Δικτύωσης και το αντίθετο. Διαχειρίζεται δύο ειδών μηνύματα δεδομένων:

➔ MAC_FRAME

Στέλνεται από τη μονάδα Ραδιοεπικοινωνίας, όταν αυτή λάβει μήνυμα από το Ασύρματο Κανάλι (Wchannel). Ενσωματωμένο στο μήνυμα αυτό είναι το APP_DATA_PACKET μήνυμα της μονάδας Εφαρμογής του αποστολέα.

➔ NETWORK_FRAME

Στέλνεται από τη μονάδα δικτύωσης, προς αποστολή στο δίκτυο. Έχει ενσωματωμένο το αυθεντικό μήνυμα APP_DATA_PACKET της μονάδας Εφαρμογής του κόμβου, προς αποστολή στο δίκτυο.

Για τους δύο αυτούς τύπους μηνυμάτων, υπάρχει πληθώρα ρυθμίσεων και ελέγχων που καθορίζουν τη μετάβαση από τον ένα τύπο στον άλλο. Δηλαδή το εισερχόμενο MAC_FRAME ελέγχεται για τον τύπο του και εφόσον είναι πακέτο δεδομένων (Data Frame) εξάγεται από αυτό το ενσωματωμένο APP_DATA_PACKET και ενσωματώνεται σε NETWORK_FRAME, το οποίο στέλνεται στη μονάδα δικτύωσης. Αυτή με τη σειρά της θα διαχειριστεί ομοίως και θα στείλει το APP_DATA_PACKET στην μονάδα Εφαρμογής. Παρόμοια διαδικασία ακολουθείται για την μετάβαση από NETWORK_FRAME σε MAC_FRAME.

Η μονάδα MAC κατ' επιλογή υλοποιείται σε δύο τύπους μονάδων:

- BypassMac
- TunableMac

Ο πρώτος τύπος, BypassMac, υλοποιεί την μετατροπή NETWORK_FRAME<-->MAC_FRAME με τις αντίστοιχες αποστολές/παραλαβές, χωρίς όμως τους ελέγχους του μέσου. Η πραγματική MAC υλοποίηση είναι η TunableMac, η οποία περιλαμβάνει διαχείριση και αποστολή μεγάλου όγκου μηνυμάτων ελέγχου του Radio, τόσο για την αποστολή όσο και για την λήψη μηνυμάτων δεδομένων και λοιπόν πακέτων αίσθησης του μέσου. Παρατίθεται ενδεικτικά το σύνολο των τύπων των μηνυμάτων ελέγχου που χρησιμοποιεί η μονάδα TunableMac και παραβλέπονται στην BypassMac:

```
enum MAC_ContorlMessageType
{
      //...
      //directive MAC-->Radio to perform carrier sense
      MAC_2_RADIO_SENSE_CARRIER = 2005;
      //self messages MAC-->MAC
      MAC_SELF_EXIT_EXPECTING_RX = 2007;
      MAC_SELF_PERFORM_CARRIER_SENSE = 2008;
      MAC_SELF_EXIT_CARRIER_SENSE = 2009;
      MAC_SELF_SET_RADIO_SLEEP = 2010;
      MAC_SELF_WAKEUP_RADIO = 2011;
      MAC_SELF_INITIATE_TX = 2012;
      MAC_SELF_POP_BUFFER_AND_SEND_DATA = 2013;
      MAC_2_NETWORK_FULL_BUFFER = 2014;
      //...
};
```

[Castalia-1.3/src/Node/Communication/Mac/MacControlMessage.msg]

Σε αυτά πρέπει να συνυπολογίσουμε και τα μηνύματα της μονάδας ραδιοεπικοινωνίας προς την MAC:

```
enum Radio_ContorlMessageType
{
      //...
      RADIO_2_MAC_SENSED_CARRIER = 4005;
      RADIO_2_MAC_FULL_BUFFER = 4006;
      RADIO_2_MAC_STARTED_TX = 4007;
      RADIO_2_MAC_STOPPED_TX = 4008;
};
```

[Castalia-1.3/src/Node/Communication/Radio/RadioControlMessage.msg]

Γίνεται αντιληπτό ότι η TunableMac μονάδα είναι ένα πολύ δυνατό εργαλείο του Castalia, για ανάλυση επικοινωνίας σε αυτό το επίπεδο. Όπως γίνεται εύκολα αντιληπτό και από τους τίτλους των τύπων των μηνυμάτων, η *void handleMessage(cMessage *msg)*:

- 'Αφουγκράζεται' το κανάλι (MAC_SELF_PERFORM_CARRIER SENSE)
- Μπαίνει σε κατάσταση παραλαβής από το Radio (RADIO_2_MAC_STARTED_TX)
- Εξέρχεται της παραλαβής αν δεν έρθει εν τέλει το πακέτο μηνύματος (MAC_SELF_EXIT_EXPECTING_RX)
- Σταματάει τη παραλαβή από το Radio (RADIO_2_MAC_STOPPED_TX)
- Στέλνει το Radio σε κατάσταση καταστολή (MAC_SELF_SET_RADIO_SLEEP)

Η διαφορά ανάμεσα στις μονάδες TunableMac και BypassMac φαίνεται εξίσου έντονα και στις παραμέτρους τους. Παρουσιάζονται ενδεικτικά, όπως είναι στα αντίστοιχα αρχεία ned:

```
simple BypassMacModule
   parameters:
      printDebugInfo: bool,
      maxMACFrameSize : const,
      macFrameOverhead: const,
      macBufferSize   : const;
      .
      .
      .
endsimple
```

[Castalia-1.3/src/Node/Communication/Mac/BypassMac/BypassMacModule.ned]

```
simple TunableMacModule
   parameters:
      printDebugInfo: bool,
      printPotentiallyDroppedPacketsStatistics: bool,
      printStateTransitions: bool,
      dutyCycle: numeric,
      listenInterval: numeric,
      beaconIntervalFraction: numeric,
      probTx: numeric,
      numTx : numeric,
      randomTxOffset: numeric,
      reTxInterval: numeric,
      maxMACFrameSize: const,
      macFrameOverhead: const,
      beaconFrameSize: const,
      ACKFrameSize: const,
      macBufferSize: const,
      carrierSense: bool,
      backoffType: const,
      backoffBaseValue: const,
      randomBackoff: bool;
      .
      .
      .
endsimple
```

[Castalia-1.3/src/Node/Communication/Mac/TunableMac/TunableMacModule.ned]

3.1.4.3 Μονάδα Ραδιοεπικοινωνίας (Radio simple module)

Η μονάδα Ραδιοεπικοινωνίας προσπαθεί να προσομοιώσει τα χαρακτηριστικά της εκπομπής/λήψης ραδιοκυμάτων, όπως τα συναντάμε σε διάφορες υλοποιήσεις ασύρματων δικτύων αισθητήρων. Η μονάδα υποστηρίζεται από πολλές παραμέτρους διαμόρφωσης της συμπεριφοράς στη μετάδοση ραδιοκυμάτων.

Οι σημαντικότερες παράμετροι είναι:

datarate:	ο ρυθμός μεταφοράς δεδομένων σε kbps
noiseBandwidth:	η συχνότητα θορύβου σε KHz
noiseFloor:	το επίπεδο θορύβου (thermal noise)
modulationType:	διαμόρφωση (0 --> MODULATION_IDEAL)
encodingType:	κωδικοποίηση (0 --> NRZ δεν υποστηρίζεται άλλη τιμή)
receiverSensitivity:	η ευαισθησία του δέκτη σε dBm
txPowerLevels:	τα επίπεδα εκπομπής σε dBm. Ορίζεται μέγιστος αριθμός από 15 τιμές ισχύος εκπομπής. Η πρώτη είναι η προεπιλογή
rxPower:	η ισχύς λήψης σε mW
listenPower:	η ισχύς ακρόασης του καναλιού σε mW
sleepPower:	η ισχύς σε κατάσταση καταστολής σε mW
txPowerConsumptionPerLevel:	κατανάλωση ισχύος σε mW για τα επίπεδα εκπομπής που ορίστηκαν στη παράμετρο txPowerlLevels
txPowerLevelUsed:	το επίπεδο εκπομπής που χρησιμοποιείται
txModeUsed:	επιλογή μεθόδου αποστολής πακέτου: 0 για αποστολή χωρίς έλεγχο του μέσου (CARRIER_SENSE_NONE) 1 για έλεγχο του καναλιού και αποστολή αν είναι ελεύθερο (CARRIER_SENSE_ONCE_CHECK) 2 για επίμονο έλεγχο του καναλιού μέχρι να είναι ελεύθερο (CARRIER_SENSE_PERSISTENT)
bufferSize:	το μέγεθος της προσωρινής μνήμης σε bytes
maxPhyFrameSize:	το μέγιστο μέγεθος του πακέτου αποστολής στο κανάλι σε bytes
phyFrameOverhead:	το επιπρόσθετο κόστος του πακέτου σε bytes

[Castalia-1.3/src/Node/Communication/Radio/RadioModule.ned]

Πολλές από τις παραμέτρους αυτές τις χρησιμοποιεί η μονάδα WChannel, δηλαδή το Ασύρματο Κανάλι Επικοινωνίας.

Η μονάδα Ραδιοεπικοινωνίας, επεξεργάζεται δύο ήδη εισερχομένων μηνυμάτων δεδομένων:

➔ WC_PKT_END_TX
➔ MAC_FRAME

Τα μηνύματα τύπου WC_PKT_END_TX είναι εισερχόμενα από το Ασύρματο Κανάλι (WChannel). Αυτά είτε περιέχουν μήνυμα τύπου MAC_FRAME, το οποίο και εξάγεται και αποστέλλεται στη MAC μονάδα, αν είναι πακέτο δεδομένων, ή απλώς δηλώνεται η παραλαβή του αν είναι πακέτο σήμανσης.

Όταν λαμβάνεται μήνυμα τύπου MAC_FRAME, η μονάδα ελέγχει το μέσο (WChannel) και την προσωρινή μνήμη (buffer) και ενημερώνει τη MAC μονάδα αν υπάρχει δυνατότητα αποστολής. Αυτή επιστρέφει μετά από όλους τους ελέγχους μήνυμα τύπου MAC_2_RADIO_ENTER_TX, το οποίο εκκινεί τη διαδικασία αποστολής του πακέτου στο δίκτυο. Πρώτα προγραμματίζεται ένα μήνυμα τύπου RADIO_SELF_ENTER_TX_NOW, το οποίο στην δική του διαχείριση εκτελεί την εντολή *popAndSendToChannel()*. Η λειτουργία αυτή βγάζει από την προσωρινή μνήμη το MAC_FRAME και ακολουθεί την εξής διαδικασία:

- Ενσωματώνει το MAC_FRAME, που είναι μήνυμα MAC_GenericFrame, σε μήνυμα τύπου WChannel_GenericMessage.

  i. Αν η ενσωμάτωση πετύχει, δηλαδή χωρέσει το ένα μέσα στο άλλο τότε:

     Δημιουργείται μήνυμα τύπου WC_PKT_BEGIN_TX το οποίο και στέλνεται στο Ασύρματο Κανάλι για να σημάνει την εκκίνηση εκπομπής πακέτων. Στο μήνυμα ενσωματώνεται η διεύθυνση του αποστολέα (ο κόμβος), η διεύθυνση του παραλήπτη (το Ασύρματο Κανάλι Επικοινωνίας) και το επίπεδο ισχύος εκπομπής όπως ορίζεται στη σχετική παράμετρο. Τέλος το μήνυμα σήμανσης στέλνεται στην θύρα εξόδου της σύνθετης μονάδας επικοινωνίας, από όπου θα προωθηθεί διαδοχικά μέχρι το Κανάλι Επικοινωνίας. Το κυρίως μήνυμα ορίζεται να είναι τύπου WC_PKT_END_TX. Σε αυτό ορίζεται καθυστέρηση αποστολής και στέλνεται με τον ίδιο τρόπο προς το Ασύρματο Κανάλι Επικοινωνίας.

  ii. Αν δεν πετύχει η ενσωμάτωση:

     Το μήνυμα ενσωμάτωσης ακυρώνεται, και το MAC_FRAME απορρίπτεται, Ο χρήστης ενημερώνεται μέσω της αποσφαλμάτωσης.

Η διαδικασία φαίνεται χαρακτηριστικά στο απόσπασμα κώδικα:

```
MAC_GenericFrame *poppedMacFrame = popBuffer();
WChannel_GenericMessage *end = new
WChannel_GenericMessage("Frame to be sent to the Wireless
                         Channel");
if(encapsulateMacFrame(poppedMacFrame, end))
//the "end" message holds the encapsulated Mac-frame
{
      //Generate and send "packet begin message"
      WChannel_GenericMessage *begin = new
WChannel_GenericMessage("Radio begins Tx", WC_PKT_BEGIN_TX);
      begin->setSrcAddress(self);
      begin->setDestAddress(end->getDestAddress());
      begin->setPowerLevel(txPowerLevelUsed);
      send(begin, "toCommunicationModule");
      //send "packet end message"
      end->setKind(WC_PKT_END_TX);
      txTime = ((double)(end->byteLength() * 8.0f)) / (1000.0f *
                dataRate);
      //calculate the TX time for the Length of frame2send frame
      sendDelayed(end, txTime, "toCommunicationModule");
}
else
{
      cancelAndDelete(end);
      cancelAndDelete(poppedMacFrame);
      end = NULL;
      poppedMacFrame = NULL;
}
```

[Castalia-1.3/src/Node/Communication/Radio/RadioModule.cc]

Στις παραπάνω διαδικασίες συνεισφέρουν και αρκετά μηνύματα ελέγχου του μέσου και επικοινωνίας με τη MAC μονάδα, όπως και πληθώρα selfMessages.

3.1.5 Μονάδα Διαχείρισης Πόρων (Resource_Manager simple module)

Η μονάδα Διαχείρισης Πόρων προσθέτει άλλο ένα επίπεδο προσομοίωσης στο Castalia, αυτό του περιορισμού στην ποσότητα και το μέγεθος των φυσικών πόρων του κόμβου.

Οι ασύρματοι αισθητήρες στην πραγματικότητα είναι μικρές συσκευές που λειτουργούν υπό ελάχιστες συνθήκες κατανάλωσης ενέργειας και κύκλων εργασίας, προκειμένου να έχουν μεγάλη αυτονομία και γρήγορη ανταπόκριση. Αυτήν ακριβώς την ιδιότητα προσπαθεί να προσομοιώσει η μονάδα αυτή.

Η Διαχείριση Πόρων είναι συνδεδεμένη με όλες τις υπόλοιπες υπομονάδες του κόμβου:

```
module Node
          //...
     connections nocheck:
       //...
     nodeResourceMgr.toSensorDevManager  -->
          nodeSensorDevMgr.fromResourceManager;
     nodeResourceMgr.toApplication -->
          nodeApplication.fromResourceManager;
     nodeResourceMgr.toNetwork -->
          networkInterface.fromResourceManager2Net;
     nodeResourceMgr.toMac  -->
          networkInterface.fromResourceManager2Mac;
     nodeResourceMgr.toRadio  -->
          networkInterface.fromResourceManager2Radio;
       //...
endmodule
```

[Castalia-1.3/src/Node/Node.ned]

Οι συνδέσεις είναι μόνο εξερχόμενες προς τις υπόλοιπες υπομονάδες. Αυτό γεννά το ερώτημα πώς εκτελεί εργασίες η μονάδα αφού δεν μπορεί να δεχθεί μηνύματα προκειμένου να εκτελεστεί η *void handleMessage(cMessage *msg)*. Η λογική πίσω από αυτή τη μονάδα είναι να χρησιμοποιούνται οι μέθοδοί της από τις άλλες μονάδες ευθέως. Για το λόγο αυτό η μονάδα υλοποιεί όλες τις υπόλοιπες μεθόδους της ως δημόσιες:

```
class ResourceGenericManager : public cSimpleModule
{
    //...
    public:
     double getCPUClockDrift(void);
     void consumeEnergy(double amount);
     double getSpentEnergy(void);
     void destroyNode(void);
     int RamStore(int numBytes);
     void RamFree(int numBytes);
};
```

[Castalia-1.3/src/Node/Resource_Manager/ResourceGenericManager.h]

Οι μέθοδοι επιστρέφουν το αποτέλεσμα τους στην μονάδα που τους κάλεσε με εξαίρεση τις *void consumeEnergy(double amount)* και *void destroyNode(void)*. Αυτές επειδή έχουν ως αποτέλεσμα την απενεργοποίηση ολόκληρου του κόμβου, αποστέλλουν μηνύματα τύπου RESOURCE_MGR_OUT_OF_ENERGY και RESOURCE_MGR_DESTROY_NODE αντίστοιχα προς όλες τις υπομονάδες του κόμβου.

Η λειτουργία των μεθόδων είναι κατανοητή από τον τίτλο τους. Αν για παράδειγμα θέλουμε να δοκιμάσουμε τη διαχείριση μνήμης του κόμβου, τότε με τη i*nt RamStore(int numBytes)*, αποθηκεύουμε bytes στη μνήμη του κόμβου. Αν η μνήμη γεμίσει, χρησιμοποιούμε την *void RamFree(int numBytes)* για να απελευθερώσουμε χώρο.

Οι παραπάνω εργασίες χρειάζονται υποστήριξη από τις αντίστοιχες μεταβλητές τιμών. Οι κυριότερες ορίζονται στη NED γλώσσα ως εξής:

```
ramSize: const, // RAM size in kbytes
sigmaCPUClockDrift: numeric,  // the standard deviation of the
                              // Drift of the CPU
initialEnergy: numeric;// energy of the node in Joules
```

[Castalia-1.3/src/Node/Resource_Manager/ResourceGenericManager.ned]

3.2 Φυσική Διεργασία (Physical_Process simple module)

Η απλή αυτή μονάδα βρίσκεται στο ίδιο επίπεδο ιεραρχίας με τον κόμβο. Ορίζεται συνήθως σε διάνυσμα, το μέγεθος του οποίου αντιπροσωπεύει το πλήθος των διαφορετικών φυσικών φαινομένων που παρακολουθούν οι αισθητήρες. Επιπλέον κάθε μονάδα Φυσικής Διεργασίας, συνδέεται με όλους τους κόμβους του δικτύου με θύρες εισόδου και εξόδου. Παρατίθεται χαρακτηριστικά ο σχετικός NED κώδικας σε υλοποίηση με ένα μόνο τύπο μονάδας Φυσικής Διεργασίας:

```
simple PhysicalProcessGenericModule
       //...
    gates:
        out: toNode[];
        in: fromNode[];
endsimple
```

[Castalia-1.3/src/Physical_Process/PhysicalProcessGenericModule.ned]

```
module SensorNetwork
    parameters:
      //...
      numNodes: const,
      physicalProcessModuleName: string,
      numPhysicalProcesses: const,
      //...
    submodules:
      //...
      physicalProcess:
            physicalProcessModuleName[numPhysicalProcesses] like
            PhysicalProcessGenericModule
         gatesizes:
            toNode[numNodes],
            fromNode[numNodes];

      node: Node[numNodes]
      //...
         gatesizes:
            toPhysicalProcess[numPhysicalProcesses],
            fromPhysicalProcess[numPhysicalProcesses];

    connections nocheck:
      //...
      for i=0..numNodes-1, j=0..numPhysicalProcesses-1 do
node[i].toPhysicalProcess[j] --> physicalProcess[j].fromNode[i];
node[i].fromPhysicalProcess[j] <-- physicalProcess[j].toNode[i];
      endfor;
endmodule
```

[Castalia-1.3/src/sensorNetwork.ned]

Όπως φαίνεται, η μονάδα ορίζεται με παράμετρο τύπου μονάδας physicalProcessModuleName. Βάσει του τύπου αυτού δημιουργείται διάνυσμα μονάδων, σύμφωνα με τη παράμετρο numPhysicalProcesses. Κάθε μονάδα του διανύσματος αυτού έχει διανύσματα θυρών εισόδου και εξόδου, τα οποία έχουν μέγεθος ίδιο με τον αριθμό των κόμβων (Nodes) του δικτύου. Συνεπακόλουθα κάθε κόμβος έχει διανύσματα θυρών εισόδου και εξόδου με μέγεθος ίσο με τον αριθμό των μονάδων Φυσικής Διεργασίας. Έτσι αυτές συνδέονται με όλους του κόμβους και στις δύο κατευθύνσεις επικοινωνίας, όπως ορίζεται με το σχετικό for statement. Οι κόμβοι έχουν φυσικά και τις στατικές θύρες προς και από το Ασύρματο Κανάλι Επικοινωνίας. Ορίζονται δηλαδή σε αυτούς θύρες σε δύο επίπεδα.

Η μονάδα Φυσικής Διεργασίας, μέσω των θυρών της σύνθετης μονάδας του κόμβου, επικοινωνεί με την υπομονάδα Συσκευής Αισθητήρα (SensorDevMgr) του κόμβου. Με αυτή ανταλλάσσει μηνύματα δεδομένων και ελέγχου.

Για την μονάδα αυτή υπάρχει μία λοποίηση τύπου μονάδας, η CustomizablePhysicalProcess. Η *void handleMessage(cMessage *msg)* αυτής διαχειρίζεται τα μηνύματα τύπου PHY_SAMPLE_REQ που λαμβάνει από τη Συσκευή Αισθητήρα του κόμβου, και επιστρέφει μέτρηση φυσικής ποσότητας με μήνυμα τύπου PHY_SAMPLE_REPLY.

3.3 Ασύρματο Κανάλι Επικοινωνίας (Wireless Channel simple module)

Η απλή αυτή μονάδα υλοποιεί το μοντέλο επικοινωνίας του μέσου, δηλαδή το Κανάλι Επικοινωνίας. Βρίσκεται στο ίδιο επίπεδο ιεραρχίας με τους κόμβους και συνδέεται με αυτούς με ζεύγη θυρών εισόδου/εξόδου:

```
simple WirelessChannel
       //...
    gates:
        out: toNode[];
        in: fromNode[];
endsimple
```

[Castalia-1.3/src/Wireless_Channel/WirelessChannel.ned]

```
module SensorNetwork
    parameters:
     //...
     numNodes: const,
     wirelessChannelModuleName: string,
     //...
    submodules:
     //...
     wirelessChannel: wirelessChannelModuleName like
                      WirelessChannel
        gatesizes:
          toNode[numNodes],
          fromNode[numNodes];

     node: Node[numNodes]
     //...

    connections nocheck:
     //...
        for i=0..numNodes-1 do
node[i].toWirelessChannel[j] --> wirelessChannel[j].fromNode[i];
node[i].fromWirelessChannel[j] <-- wirelessChannel[j].toNode[i];
        endfor;
endmodule
```

[Castalia-1.3/src/sensorNetwork.ned]

Όπως φαίνεται από τον NED κώδικα το Ασύρματο Κανάλι ορίζεται με παράμετρο τύπου μονάδας wirelessChannelModuleName και οι θύρες του με διανύσματα θυρών. Το μέγεθός τους αποδίδεται στη δήλωση του δικτύου και είναι ίσο με τον αριθμό των κόμβων. Τέλος ορίζονται οι συνδέσεις για κάθε κόμβο. Το Ασύρματο Κανάλι επικοινωνεί μέσω αυτών των συνδέσεων με την υπομονάδα Ραδιοεπικοινωνίας (Radio) του κόμβου.

Το κανάλι επικοινωνίας είναι αυτό που αποφασίζει αν τελικά επικοινωνούν δύο κόμβοι μεταξύ τους ή όχι. Το Castalia περιλαμβάνει μόνο έναν τύπο απλής μονάδας, το WChannel_Interference. Σε αυτό η επικοινωνία δύο κόμβων βασίζεται στην εκτίμηση των απωλειών ισχύος στο κανάλι σύμφωνα με το μοντέλο της Εξασθένισης Διαδρομής Λογαριθμικής Απόστασης (Log-Distance path loss model) [§4.1.2.2]. Η μονάδα για να υποστηρίξει το μοντέλο αυτό ορίζει τις εξής παραμέτρους:

pathLossExponent: ο εκθέτης της εξασθένισης
PLd0: η εξασθένισης αναφοράς για απόσταση d0 σε dBm
d0: η απόσταση αναφοράς σε μέτρα

collisionModel: το μοντέλο σύγκρουσης μηνυμάτων (παρεμβολή ραδιοκυμάτων: radio Interference)
// 0 --> No Collisions (no Interference)
// 1 --> Simple Collision Model
// 2 --> Additive interference model
// 3 --> Complex interference model
sigma: η τυπική απόκλιση της μέσης τιμής της εξασθένισης για κανονικοποιημένη κανονική κατανομή σε dBm
allBidirectionalLinks: ορίζει αν όλες οι συνδέσεις είναι αμφίδρομες (bool)
PRR_ConnectivityMap: χάρτης πιθανοτήτων παραλαβής μηνύματος (0<=PRR<=1)
rxSignal_ConnectivityMap: χάρτης συνδεσιμότητας με δείγματα ισχύος σε dBm

[Castalia-1.3/src/Wireless_Channel/WChannel_Interference/WChannel_Interference.ned]

Οι χάρτες συνδεσιμότητας ορίζονται με σύνθετη δήλωση, στην οποία περιλαμβάνονται τα ζεύγη αποστολέα και δέκτη, και η τιμή της μεταβλητής:

```
rxSignal_ConnectivityMap: node1-->node2(txlevel)=value,
                          node1-->node3(txlevel)=value,...
     PRR_ConnectivityMap: <packet size>node1-->node3=value,...
```

Οι τιμές για την εξασθένιση αναφοράς PLd0 και την απόσταση αναφοράς d0 είναι ενιαίες για όλες τις ζεύξεις.

Η τάξη WChannel_Interference δανείζεται πολλές από τις παραμέτρους της Radio μονάδας των κόμβων:

```
//Radio module parameters
double dataRate;
double noiseBandwidth;
int modulationType;
int encodingType;
double receiverSensitivity;
int numTxPowerLevels;
double txPowerLevels[15];
```

[Castalia-1.3/src/Wireless_Channel/WChannel_Interference/WChannel_Interference.h]

Επιπλέον ορίζονται στην τάξη οι εξής παράμετροι:

---

double thresholdSNRdB: κάτω όριο SNR για επίτευξη σύνδεσης
double goodLinkSNRdB: άνω όριο SNR για επίτευξη σύνδεσης
double ***rxSignal: τρισδιάστατο διάνυσμα που αποθηκεύει την λαμβανόμενη ισχύ k σήματος που φτάνει σε ένα κόμβο i, και εκπέμπεται από κόμβο j. Η ισχύς αυτή σε dBm ορίζεται ως rxSignal[i][j][k].
double **maxInterferenceForCurrentTx: μέγιστη ισχύ παρεμβολής σε dBm. Ορίζει δισδιάστατο διάνυσμα για τον δέκτη i και τον πομπό j maxInterferenceForCurrentTx[i][j].
ListElement *activeNodes: λίστα με ενεργούς κόμβους
ListElement *carrierSensingNodes: λίστα με κόμβους που βρίσκονται σε κατάσταση αίσθησης του μέσου
int **stats: δισδιάστατος πίνακας στατιστικών για κάθε κόμβο. Για ένα κόμβο i είναι:
stats[i][0]: οι εκπομπές του κόμβου i
stats[i][1]: οι πιθανές παραλαβές για τις εκπομπές του i
stats[i][2]: οι παρεμβαλλόμενες παραλαβές για τις εκπομπές του i
stats[i][3]: οι πραγματικές παραλαβές των εκπομπών του i
stats[i][4]: οι παραλαβές του κόμβου i
const char * rxSignal_ConnectivityMap: χάρτης δειγμάτων συνδεσιμότητας se dBm
const char * PRR_ConnectivityMap: χάρτης πιθανότητας συνδεσιμότητας (0,1)
int defaultPowerLevel: προεπιλεγμένη ισχύς εκπομπής σε dBm
double minumumPacketSize: ελάχιστο μέγεθος πακέτου σε bytes

[Castalia-1.3/src/Wireless_Channel/WChannel_Interference/WChannel_Interference.h]

Η λειτουργία της μονάδας του Ασύρματου Καναλιού, υλοποιείται στις κύριες μεθόδους της ως εξής:

3.3.1 *void initialize()*

Στην εκκίνηση η μονάδα ορίζει τις μεταβλητές thresholdSNRdB και goodLinkSNRdB, με τις οποίες κρίνεται η σύνδεση δύο κόμβων. Η πρώτη αντιπροσωπεύει το κατώτερο όριο αναλογίας σήματος /θορύβου, ενώ η δεύτερη το ανώτερο. Η απόδοση τιμών σε αυτές βασίζεται στη τιμή της παραμέτρου modulationType (τύπος διαμόρφωσης):

```
if(modulationType == MODULATION_FSK)
{
thresholdSNRdB = 10.0 * log10((-2.0*(dataRate/noiseBandwidth))
                      * log(2.0 * (1.0 -  pow(THRESHOLD_PROB,
                        1.0/(8.0*minumumPacketSize))) ));

goodLinkSNRdB = 10.0 * log10((-2.0*(dataRate/noiseBandwidth))
                     * log( 2.0 * (1.0 -  pow(GOOD_LINK_PROB,
                       1.0/(8.0*minumumPacketSize))) ));
}
else if(modulationType == MODULATION_PSK)
{
thresholdSNRdB = 10.0 * log10((dataRate/noiseBandwidth)
                      * pow(erfcInv(2.0 * ( 1.0 -
                        pow(THRESHOLD_PROB, 1.0/
                       (8.0*minumumPacketSize)) )), 2.0));

goodLinkSNRdB = 10.0 * log10((dataRate/noiseBandwidth)
                     * pow(erfcInv(2.0 * ( 1.0 -
                       pow(GOOD_LINK_PROB, 1.0/
                      (8.0*minumumPacketSize)) )), 2.0));
}
else //(modulationType == MODULATION_IDEAL)
{
thresholdSNRdB = IDEAL_MODULATION_THRESHOLD_SNR_DB;
goodLinkSNRdB = IDEAL_MODULATION_GOOD_LINK_SNR_DB;
}
```

[Castalia-1.3/src/Wireless_Channel/WChannel_Interference/WChannel_Interference.cc]

όπου:

```
IDEAL_MODULATION_THRESHOLD_SNR_DB = 5.0
IDEAL_MODULATION_GOOD_LINK_SNR_DB = 8.5
```

Στη συνέχεια υπολογίζεται η εξασθένιση διαδρομής Λογαριθμικής Απόστασης PLd για όλα ζεύγη κόμβων, και η λαμβανόμενη ισχύς rxSignal[i][j][k] σε κάθε πιθανή περίπτωση σύνδεσης:

```
for (int i = 0; i < numNodes; i++)
{
   x1 = topo->node(i)->module()->par("xCoor");
   y1 = topo->node(i)->module()->par("yCoor");

   for (int j = 0; j < numNodes; j++)
   {
      if (i == j)
      {
         for (int k = 0; k < numTxPowerLevels; k++)
         rxSignal[i][j][k] = 100.0;
      }
      else
      {
         x2 = topo->node(j)->module()->par("xCoor");
         y2 = topo->node(j)->module()->par("yCoor");
         dist = sqrt((x2 - x1)*(x2 - x1) + (y2 - y1)*(y2 - y1));

         PLd = PLd0 + 10.0 * pathLossExponent * log10(dist/d0) +
               normal(0, sigma);

         for (int k = 0; k < numTxPowerLevels; k++)
         rxSignal[i][j][k] = txPowerLevels[k] - PLd;
      }
   }
}
```

[Castalia-1.3/src/Wireless_Channel/WChannel_Interference/WChannel_Interference.cc]

Όπως φαίνεται οι τιμές που αποδίδονται στην εξασθένιση αλλά και την λαμβανόμενη ισχύ, συνυπολογίζουν τη διασπορά. Επίσης για τον υπολογισμό της απόστασης ανάμεσα σε δύο κόμβους χρησιμοποιείται μόνο η οριζόντια έννοια, δεδομένου ότι δεν χρησιμοποιείται z συντεταγμένη για την τρίτη διάσταση.

Στη συνέχεια ελέγχεται η παράμετρος allBidirectionalLinks και αν είναι αληθής υπολογίζονται οι σχέσεις θορύβου και ισχύος για τον πομπό και τον δέκτη. Βάσει αυτών υπολογίζεται η μέγιστη ισχύς στα άκρα μιας σύνδεσης, και αποδίδεται και στα δύο. Δηλαδή με αυτό τον τρόπο εξασφαλίζουμε ότι η αποστολή μηνυμάτων ανάμεσα σε δύο κόμβους θα γίνεται με τον ίδιο απολύτως τρόπο και προς τις δύο κατευθύνσεις:

```
double snrA, snrB;
double rndVar;
snrA = (rxSignal[i][j][k] - noiseFloor);
snrB = (rxSignal[j][i][k] - noiseFloor);
if( (snrA > goodLinkSNRdB) || (snrB > goodLinkSNRdB) )
{
   if(snrA > snrB) rxSignal[j][i][k] = rxSignal[i][j][k];
   else            rxSignal[i][j][k] = rxSignal[j][i][k];
}
else
   rndVar = genk_dblrand(0);
if(rndVar < 0.5) rxSignal[j][i][k] = rxSignal[i][j][k];
else             rxSignal[i][j][k] = rxSignal[j][i][k];
```

[Castalia-1.3/src/Wireless_Channel/WChannel_Interference/WChannel_Interference.cc]

Ορίζονται οι μεταβλητές snrA και snrB. Αποδίδονται οι πρώτες τιμές:

- snrA: η ισχύς k που λαμβάνει ο κόμβος j από τον i μείον το επίπεδο θορύβου
- snrB: η ισχύς k που λαμβάνει ο κόμβος i από τον j μείον το επίπεδο θορύβου

Οι τιμές αυτές συγκρίνονται αμφότερες με το ανώτερο κατώφλι SNR goodLinkSNRdB:

Αν έστω μία από τις δύο είναι μεγαλύτερη από το κατώφλι, τότε συγκρίνονται και μεταξύ τους, διότι μπορεί και οι δύο να είναι μεγαλύτερες από το κατώφλι. Αν είναι μεγαλύτερη η snrA, τότε η λαμβανόμενη ισχύς rxSignal[i][j][k] αποδίδεται και στην αντίθετη κατεύθυνση. Το αντίθετο γίνεται αν η snrB είναι μεγαλύτερη της snrA. Φροντίζουμε δηλαδή να υπάρχει η μέγιστη λαμβανόμενη ισχύς και στα δύο άκρας της σύνδεσης "`i<-->j`."

Στην περίπτωση που οι snrA και snrB είναι μικρότερες και οι δύο από το κατώφλι goodLinkSNRdB, τότε ορίζεται τυχαία μεταβλητή με τη λειτουργία *genk_dblrand(int k)* του Omnet++, η οποία χρησιμοποιεί την γεννήτρια k για να επιστρέψει τυχαίο δεκαδικό στο διάστημα [0,1). Αν αυτός είναι μικρότερος από 0.5, ορίζεται η rxSignal[i][j][k] ως μέγιστη ισχύς στη σύνδεση, και στην αντίθετη περίπτωση μέγιστη ορίζεται η rxSignal[j][i][k].

Παρατηρούμε ότι στην μέθοδο εκκίνησης της μονάδας του Ασύρματου καναλιού ορίζονται όλες οι τιμές ισχύος για όλες τις πιθανές περιπτώσεις. Δεν εξετάζεται αν μπορούν να επικοινωνήσουν βάσει άλλων συνθηκών οι κόμβοι. Σε αυτή τη μέθοδο επεμβαίνουμε στην υλοποίηση.

3.3.2 *void handleMessage(cMessage* msg)*

Η μέθοδος διαχείρισης μηνυμάτων του ασύρματου καναλιού διαχειρίζεται μηνύματα ελέγχου και μηνύματα αποστολής. Τα σημαντικότερα μηνύματα και οι αντίστοιχες δράσεις είναι:

➔ WC_CARRIER_SENSE_BEGIN

Το μήνυμα αυτό προέρχεται από την υπομονάδα Radio κάποιου κόμβου, προκειμένου να εξετάσει αν μπορεί να σταλεί κάποιο μήνυμα στο δίκτυο. Ο κόμβος προστίθεται στην λίστα των κόμβων βρίσκονται σε κατάσταση αίσθησης του μέσου:

```
node = new ListElement;    node->index = srcAddr;
carrierSensingNodes = addToFront(carrierSensingNodes, node);
```

[Castalia-1.3/src/Wireless_Channel/WChannel_Interference/WChannel_Interference.cc]

Στη συνέχεια ελέγχεται το επίπεδο ισχύος εκπομπής του, powerLevel. Αν αυτό είναι 0 ή μεγαλύτερο από τη numTxPowerLevels μεταβλητή, τότε ορίζεται ως 0. Δηλαδή αποδίδεται στο μήνυμα του αποστολέα η πρώτη τιμή ισχύος στη σειρά που ορίζεται από το διάνυσμα txPowerLevels[15]. Στη συνέχεια ελέγχεται για όλους τους υπόλοιπους ενεργούς κόμβους αν η λαμβανόμενη ισχύς στον καθένα τους είναι μεγαλύτερη από την ευαισθησία της κεραίας τους. Αν ναι, τότε επιστρέφεται στο Radio του κόμβου εκπομπής μήνυμα WC_CARRIER_SENSED, προκειμένου να γίνει η αποστολή του πακέτου δεδομένων.

➔ WC_CARRIER_SENSE_END

Το μήνυμα αυτό στέλνεται πάντα με κάποια καθυστέρηση από το Radio του κόμβου, προκειμένου να αφαιρεθεί αυτός από την λίστα των κόμβων που βρίσκονται σε κατάσταση αίσθησης του μέσου:

```
carrierSensingNodes = deleteListElement(carrierSensingNodes,
                                                    srcAddr);
```

[Castalia-1.3/src/Wireless_Channel/WChannel_Interference/WChannel_Interference.cc]

➔ WC_PKT_BEGIN_TX

Ο τύπος μηνύματος αυτός αποστέλλεται στο Ασύρματο κανάλι από την υπομονάδα Radio ενός κόμβου, για να ειδοποιήσει ότι θα ξεκινήσεις αποστολή πακέτου δεδομένων. Ακολουθείται εκ νέου η διαδικασία ελέγχου που εφαρμόσθηκε κατά την παραλαβή μηνύματος τύπου WC_CARRIER_SENSE_BEGIN, μόνο που πλέον ο κόμβος που έστειλε το μήνυμα προστίθεται στη λίστα ενεργών κόμβων και ενημερώνονται τα στατιστικά του:

```
node = new ListElement;
node->index = srcAddr;
node->powerLevel = message->getPowerLevel();
activeNodes = addToFront(activeNodes, node);
stats[srcAddr][0]++;
```

[Castalia-1.3/src/Wireless_Channel/WChannel_Interference/WChannel_Interference.cc]

Κατόπιν γίνεται ο έλεγχος ισχύος λήψης / ευαισθησίας δέκτη, και στέλνεται μήνυμα ειδοποίησης WC_CARRIER_SENSED στην υπομονάδα Radio του δέκτη:

```
If ( rxSignal[srcAddr][tempAddr][powerLevel] >
     receiverSensitivity )
sendDelayed( new cMessage("carrier signal sensed",
             WC_CARRIER_SENSED), 0.0001, "toNode", tempAddr )
```

[Castalia-1.3/src/Wireless_Channel/WChannel_Interference/WChannel_Interference.cc]

όπου:

srcAddr: η διεύθυνση του αποστολέα
powerLevel: το επίπεδο ισχύος εκπομπής
tempAddr: η διεύθυνση του παραλήπτη (node->index)

Ο δέκτης μπαίνει σε κατάσταση ακρόασης μετά τους σχετικούς ελέγχους, προκειμένου να παραλάβει το μήνυμα που θα στείλει ο πομπός.

➔ WC_PKT_END_TX

Το μήνυμα αυτό στέλνεται με μια μικρή καθυστέρηση από το WC_PKT_BEGIN_TX, ώστε να προετοιμαστεί η μονάδα και ο παραλήπτης κόμβος για την παραλαβή του κυρίως μηνύματος, του πακέτου δεδομένων. Η *handleMessage(cMessage *msg)* εξετάζει το μήνυμα βάσει του μοντέλου παρεμβολής (collision model). Οι διαφορετικές προσεγγίσεις είναι:

I. Αποστολή Χωρίς Παρεμβολές:
(NO_INTERFERENCE_NO_COLLISIONS)

Συγκρίνεται η ισχύς λήψης στο δέκτη με την ευαισθησία του, όπως και στις προηγούμενες περιπτώσεις. Αν είναι μεγαλύτερη υπολογίζεται η αναλογία σήματος και θορύβου SNR_dB, ως η διαφορά της ισχύος στο δέκτη με τον θόρυβο σε αυτόν. Αν το SNR_dB είναι μεγαλύτερο από το κάτω όριο thresholdSNRdB εξετάζεται η πιθανότητα παραλαβής του μηνύματος από τον παραλήπτη, βάσει του τύπου της διαμόρφωσης (modulatioType), όπως ορίζεται στην υπομονάδα Radio:

```
if(modulationType == MODULATION_FSK)
   prob = pow( 1.0 - 0.5 *
          exp(((-0.5) * noiseBandwidth / dataRate)*
          pow(10.0, (SNR_dB/10.0))), (8.0*packetByteSize) );
else
if(modulationType == MODULATION_PSK)
   prob = pow( 1.0 - 0.5 * erfc( sqrt(pow(10.0,(SNR_dB/10.0)) *
          noiseBandwidth / dataRate)  ),  (8.0*packetByteSize));
else //(modulationType == MODULATION_IDEAL)
   prob = 1.0;
```

[Castalia-1.3/src/Wireless_Channel/WChannel_Interference/WChannel_Interference.cc]

Αν η πιθανότητα prob προκύψει ίση με 1 ή μεγαλύτερη από ένα τυχαίο αριθμό στο διάστημα [0,1) που γεννάται με την genk_dblrand() γεννήτρια τυχαίων αριθμών του Omnet++, τότε το μήνυμα εγκρίνεται για αποστολή. Σε αυτή τη περίπτωση δημιουργείται αντίγραφό του με τη μέθοδο dup(), ενσωματώνεται σε αυτό η ισχύς λήψης στη παράμετρο Rssi, και τέλος στέλνεται στον παραλήπτη. Αφού σταλεί ένα μήνυμα σε ένα κόμβο, ο δεύτερος αφαιρείται από την λίστα των ενεργών κόμβων (activeNodes)

II. Αποστολή με Παρεμβολές - Απλό Μοντέλο Συγκρούσεων:
(ADDITIVE_INTERFERENCE_MODEL - SIMPLE_COLLISION_MODEL)

Με το Απλό Μοντέλο Συγκρούσεων, υπολογίζεται η παρεμβολή σε κάθε εκπομπή από όλες τις υπόλοιπες και συγκρίνεται με τη μέγιστη που έχει παρατηρηθεί. Αν είναι μεγαλύτερη ορίζεται ως η νέα μέγιστη. Στη συνέχει συγκρίνεται με την ευαισθησία του δέκτη και το επίπεδο θορύβου σε αυτόν.

- Αν είναι μεγαλύτερη και από τους δύο ορίζει αναλογία σήματος / θορύβου:

  SNR_dB = -1

- Αν είναι μικρότερη:
  SNR_dB = rxSignal[srcAddr][i][srcPowerLevel]
  - addPower_dBm(noiseFloor, maxInterferenceForCurrentTx[srcAddr][i])

[Castalia-1.3/src/Wireless_Channel/WChannel_Interference/WChannel_Interference.cc]

όπου:

SNR_dB: αναλογία σήματος / θορύβου
rxSignal[srcAddr][i][srcPowerLevel]: ισχύς λήψης srcPowerLevel, σήματος που εστάλει από τον κόμβο srcAddr στον κόμβο i
noiseFloor: το επίπεδο θορύβου
maxInterferenceForCurrentTx[srcAddr][i]: μέγιστη παρεμβολή εκπομπής σήματος από τον κόμβο srcAddr στον κόμβο i
addPower_dBm(double, double): λειτουργία μερικής αύξησης ισχύος σε dBm

---

Το SNR_dB συγκρίνεται με το κάτω όριο thresholdSNRdB, και αν είναι μεγαλύτερο εξετάζεται η πιθανότητα παραλαβής του μηνύματος από τον παραλήπτη, βάσει του τύπου της διαμόρφωσης (modulatioType), με τον ίδιο τρόπο όπως και στην περίπτωση της απουσίας παρεμβολών.

Αν η πιθανότητα prob προκύψει ίση με 1 ή μεγαλύτερη από ένα τυχαίο αριθμό στο διάστημα [0,1) που γεννάται με την *genk_dblrand()* γεννήτρια τυχαίων αριθμών του Omnet++, τότε το μήνυμα εγκρίνεται για αποστολή. Σε αυτή τη περίπτωση δημιουργείται αντίγραφό του με τη μέθοδο *dup()*, ενσωματώνεται σε αυτό η ισχύς λήψης στη παράμετρο Rssi, και τέλος στέλνεται στον παραλήπτη. Αφού σταλεί ένα μήνυμα σε ένα κόμβο, ο δεύτερος αφαιρείται από την λίστα των ενεργών κόμβων (activeNodes). Επιπλέον ελαχιστοποιείται η μέγιστη παρεμβολή για εκ νέου υπολογισμό σε νέα αποστολή μεγίστων τιμών.

Στο τέλος της διαχείρισης καταστρέφεται το αρχικό μήνυμα που παρέλαβε το ασύρματο κανάλι, δεδομένου ότι τα μηνύματα που έστειλε είναι αντίγραφα του πρωτότυπου.

3.3.3 *void finish()*

Η μέθοδος χρησιμοποιείται για την εκτύπωση των στατιστικών αποστολής των κόμβων που διαχειρίζεται η μονάδα στο δισδιάστατο πίνακα int **stats.

3.b Ορισμός Δικτύου (SensorNetwork)

Η ανάλυση των μονάδων του δικτύου και η θέση τους στην προσομοίωση, ολοκληρώνεται με τον ορισμό του ίδιου του περιβάλλοντος της προσομοίωσης, δηλαδή της σύνθετης μονάδας του δικτύου. Σε αυτό οι απλές υπομονάδες Ασύρματο Κανάλι (wirelessChannel) και Φυσική Διεργασία (physicalProcess) ορίζονται με παράμετρο τύπου μονάδας. Οι παράμετροι του δικτύου συμπληρώνονται από τον αριθμό των κόμβων του, των αριθμό των Φυσικών Διεργασιών και φυσικά τις διαστάσεις τις απεικόνισής του. Επιπλέον ορίζεται και μια ακέραιη παράμετρος deploymentType για τον τρόπο τοποθέτησης των κόμβων στο δίκτυο.

```
module SensorNetwork
    parameters:
      field_x:        const,
      field_y:                const,
         numNodes:                          const,
         deploymentType:                    numeric,
      physicalProcessModuleName:            string,
      wirelessChannelModuleName:            string,
      numPhysicalProcesses:                 const,
      debugInfoFilename:                    string;
      //...
endmodule
```

[Castalia-1.3/src/SensorNetwork.ned]

Κατόπιν ορίζονται οι υπομονάδες βάσει των τύπων τους, και ορίζονται τα μεγέθη των διανυσμάτων σύμφωνα με τις σχετικές παραμέτρους:

```
submodules:
   wirelessChannel: wirelessChannelModuleName like
                    WirelessChannel
               gatesizes:
                      toNode[numNodes],
                      fromNode[numNodes];
   physicalProcess:
           physicalProcessModuleName[numPhysicalProcesses]
           like PhysicalProcessGenericModule
               gatesizes:
                      toNode[numNodes],
                      fromNode[numNodes];
   node: Node[numNodes]
               parameters if deploymentType == 0:
                       xCoor = uniform(0, field_x),
                       yCoor = uniform(0, field_y);
                           //...
               gatesizes:
                      toPhysicalProcess[numPhysicalProcesses],
                      fromPhysicalProcess[numPhysicalProcesses];
```

[Castalia-1.3/src/SensorNetwork.ned]

Παρατηρούμε πως στους κόμβους ορίζεται η θέση με if statements. Παρατίθεται μόνο η πρώτη περίπτωση όπου πραγματοποιείται τυχαία ομοιόμορφη κατανομή τους στις διαστάσεις του δικτύου. Οι θύρες των Κόμβων προς το Ασύρματο Κανάλι, ορίζονται στην NED δήλωση της Node μονάδας.

Τέλος ορίζονται οι συνδέσεις των υπομονάδων:

```
connections nocheck:

    for i=0..numNodes-1 do
node[i].toWirelessChannel --> wirelessChannel.fromNode[i];
node[i].fromWirelessChannel <-- wirelessChannel.toNode[i];
    endfor;

    for i=0..numNodes-1, j=0..numPhysicalProcesses-1 do
node[i].toPhysicalProcess[j] --> physicalProcess[j].fromNode[i];
node[i].fromPhysicalProcess[j] <-- physicalProcess[j].toNode[i];
    endfor;
```

[Castalia-1.3/src/SensorNetwork.ned]

Για να μπορεί η σύνθετη μονάδα SensorNetwork να χρησιμοποιηθεί ως δίκτυο προσομοίωσης μένει να ορισθεί και ως τέτοιο:

```
network SN : SensorNetwork

endnetwork
```

[Castalia-1.3/src/SensorNetwork.ned]

3.4 Αρχείο Ρυθμίσεων omnetpp.ini και Ειδικά Αρχεία Ρυθμίσεων Παραμέτρων Μονάδων

Το Castalia ως προσομοιωτής του Omnet++ βασίζεται στις ρυθμίσεις του αρχείου omnetpp.ini, προκειμένου να εκτελεί προσομοιώσεις. Η σύνταξη του αρχείου διέπεται από τις ίδιες αρχές όπως και για κάθε άλλο προσομοιωτή που κατασκευάζουμε στο Omnet++ [§2.8].

Για το Castalia, στο omnetpp.ini καθορίζονται κυρίως οι παράμετροι των κόμβων [§3.1] και του δικτύου προσομοίωσης [§3.b]. Οι παράμετροι αυτοί είναι οι διαστάσεις της σύνθετης μονάδας του δικτύου, οι συντεταγμένες των κόμβων, ο χρόνος που θα διαρκέσει η προσομοίωση κλπ. Οι συντεταγμένες των κόμβων συνηθίζεται να περιλαμβάνονται σε ξεχωριστό αρχείο ρυθμίσεων. Για τυχαία δημιουργία και κατανομή κόμβων, το δίκτυο διαθέτει την παράμετρο `deploymentType` η οποία και παίρνει τη τιμή της σε αυτό το αρχείο. Ακόμα φορτώνονται τα NED αρχεία ορισμού των μονάδων και ορίζονται τα αρχεία συλλογής στατιστικών

Στο αρχείο αυτό ορίζεται και η μονάδα εφαρμογής των κόμβων, δηλαδή η τιμή της παραμέτρου του τύπου μονάδας εφαρμογής. Επιπλέον αποδίδονται τιμές στις παραμέτρους της [§3.1.2].

Οι παράμετροι για τις υπόλοιπες υπομονάδες των κόμβων [§3.1.3 – §3.1.5] όπως και για το Ασύρματο Κανάλι [§3.3] και την Φυσική Διεργασία [§3.2], αποδίδονται σε ξεχωριστά αρχεία. Αυτά βρίσκονται στον υποκατάλογο “Simulations/Parameter_Include_Files” του κεντρικού καταλόγου του Castalia, και μοιράζονται σε επιπλέον υποκαταλόγους, βάσει της μονάδας για την οποία αποδίδουν τιμές στις ρυθμίσεις της. Παρατίθεται απόσπασμα της οργάνωσης των αρχείων αυτών:

```
|-- Parameter_Include_Files
|-- MAC_Bypass.ini
|-- MAC_Tunable.ini
|-- MAC_just_carrierSense.ini
|-- Radio
|   |-- Custom_IDEALmodulation.ini
|-- Routing_bypass.ini
|-- WChannel
|   |-- Additive_Interference_Model
|   |   `-- WChannel_yesInterference_yesSigma_yesAllBidirectional.ini
|   |-- No_Interference
|   |   |-- WChannel_Ideal.ini
|   `-- Simple_Interference_Model
|       |-- WChannel_yesInterference_noSigma_yesAllBidirectional.ini
|-- extractOmnetppINI
|-- general_and_RNGs.ini
|-- nodeSensorDevMgr_Temperature.ini
|-- omnet_cmdenv_reporting.ini
|-- physicalProcess_0_node6_asssignedValue40.ini
|-- physicalProcess_0_one_source_at_2_38.ini
`-- resourceMgr_2AAbatteries.ini
```

Οι παράμετροι στα αρχεία αυτά μπορούν να δηλωθούν και στο κυρίως omnetpp.ini, αρκεί πάντα να έχουμε υπόψη ότι η πρώτη δήλωση τιμής σε μια παράμετρο είναι και αυτή που θα εφαρμοστεί. Θα πρέπει δηλαδή να ορίσουμε τιμή πριν φορτώσουμε το εκάστοτε ειδικό αρχείο [§2.8].

Επιπλέον για κάθε σενάριο προσομοίωσης δηλώνονται στο omnetpp.ini οι κατατακτήριοι αριθμοί (seed) ελέγχου των γεννητριών τυχαίων αριθμών (RNGs). Αυτό έχει νόημα όταν θέλουμε να ελέγχουμε τις τιμές που παράγουν οι γεννήτριες αυτές, ώστε να εξετάζουμε τα ιδια αποτελέσματα σε πολλές προσομοιώσεις. Η προεπιλογή είναι τυχαία δημιουργία αριθμών κατάταξης.

ΜΕΡΟΣ 2ο

ΜΕΤΑΔΟΣΗ ΡΑΔΙΟΣΥΧΝΟΤΗΤΩΝ

Μοντέλα Εξασθένισης Σήματος Μεγάλης Κλίμακας

4. Μετάδοση Ραδιοσυχνοτήτων (RF Propagation)

Στο κεφάλαιο αυτό γίνεται μια μικρή θεωρητική ανάλυση της μετάδοσης ραδιοσυχνοτήτων και των απωλειών που υφίσταται η ισχύς του εκπεμπόμενου σήματος στο διάστημα ανάμεσα στον πομπό και τον δέκτη. Για τις απώλειες αυτές παρουσιάζονται διάφορα εμπειρικά μοντέλα μεγάλης κλίμακας, κατάλληλα για τον τύπο της περιοχής που εξετάζουμε, δηλαδή τον υπαίθριο χώρο. Για κάθε μοντέλο δίνονται οι εξισώσεις υπολογισμού της εξασθένισης, οι οποίες και θα χρησιμοποιηθούν στην κατασκευή των μοντέλων επικοινωνίας στους προσομοιωτές που εξετάζονται. Το κύριο κίνητρο για την ανάλυση αυτή ήταν η παντελής έλλειψη σχετικών υποδομών για την εξέταση της επικοινωνίας στα Shawn και Omnet++. Το Castalia αντίθετα χρησιμοποιεί μοντέλο υπολογισμού εξασθένισης. Κρίθηκε ωστόσο σκόπιμο να εξεταστούν περισσότερα του ενός μοντέλα και συνδυασμοί.

4.1 Συστήματα Επικοινωνιών και Προϋπολογισμός Σύνδεσης

Η σχεδίαση συστημάτων επικοινωνιών για ένα συγκεκριμένο περιβάλλον απαιτεί τον υπολογισμό όλων εκείνων των παραγόντων που επηρεάζουν την επικοινωνία των μονάδων του και την εν γένει απόδοση του συστήματος, προκειμένου η τελική υλοποίηση να είναι επαρκής, λειτουργική και οικονομική. Σε αυτό οφείλουμε να υπολογίσουμε τις παραμέτρους που καθορίζουν την απόδοση κάθε σύνδεσης του συστήματος. Στην περίπτωση των ασυρμάτων δικτύων, η σχεδίαση τους βασίζεται στην συμπεριφορά των ραδιοσυχνοτήτων στο εκάστοτε περιβάλλον μετάδοσης (RF planning), και οι κρίσιμες παράμετροι που επηρεάζουν την επικοινωνία είναι [5]:

I. Η ισχύς σήματος που φτάνει στον δέκτη
II. Ο θόρυβος που συνοδεύει το σήμα
III. Η διασπορά των τιμών

Για κάθε σύνδεση πραγματοποιείται ένας προϋπολογισμός σύνδεσης ο οποίος χρησιμοποιώντας την εκπεμπόμενη ισχύ του σήματος και όλες τις απώλειες μέχρι τον δέκτη υπολογίζει την ισχύ που φθάνει σε αυτόν. Κατόπιν εάν γνωρίζουμε την ευαισθησία της κεραίας του δέκτη μπορούμε να αποφανθούμε για τον αν λαμβάνει το σήμα ή όχι, και εφόσον το λαμβάνει πιο είναι το 'περιθώριο συνδεσιμότητας' (link margin) για την συγκεκριμένη σύνδεση. Στην περίπτωση που δεν γνωρίζουμε την ευαισθησία στο δέκτη αλλά γνωρίζουμε την σχέση της με το θόρυβο (SNR), πρέπει να υπολογίσουμε την του θορύβου στο δέκτη προκειμένου να εξετάσουμε αν μπορεί να αντιληφθεί το σήμα που φθάνει σε αυτόν [5].

Στα ασύρματα συστήματα επικοινωνίας, στην περίπτωση που έχουμε πολυκατευθυντική μετάδοση σήματος, χρησιμοποιούμε συνήθως ως κεραία αναφοράς μια ιδεατή κεραία, τον 'ισότροπο πομπό', που εκπέμπει ισχύ με ενιαίο μοναδιαίο κέρδος (Gain = 1) όμοια σε όλες τις κατευθύνσεις. Η αποδοτική ισχύς ισότροπης μετάδοσης (effective isotropic radiated power) $EIRP$ ενός πομπού ορίζεται ως το γινόμενο της εκπεμπόμενης ισχύος P_{Tx} επί το κέρδος της κεραίας G_{Tx} του πομπού [4]:

$$EIRP = P_t \cdot G_t \quad (4.1)$$

Το κέρδος της κεραίας μπορεί να περιγραφεί ως η μέγιστη εκπεμπόμενη ισχύς σε μια κατεύθυνση συγκριτικά με την ισχύ που θα μετέδιδε προς όλες τις κατευθύνσεις ένας 'ισότροπος πομπός' [5]. Αντίστοιχα η "EIRP" αντιπροσωπεύει την μέγιστη εκπεμπόμενη ισχύ που εκπέμπει ένας πομπός στην κατεύθυνση που έχει το μέγιστο κέρδος της κεραίας [4].

Το 'περιθώριο συνδεσιμότητας' υπολογίζεται συγκρίνοντας την λαμβανόμενη ισχύ σήματος στο δέκτη με την ευαισθησία του δέκτη. Είναι ένα μέτρο ποιότητας της σύνδεσης και δίνει το περιθώριο που υπάρχει ανάμεσα στην κατάσταση λειτουργίας της σύνδεσης και στο οριακό σημείο αποτυχίας αυτής. Το 'περιθώριο συνδεσιμότητας' δίνεται από τον εξής τύπο [5]:

$$Link\ Margin = EIRP - L_{Path} + G_{Rx} - TH_{Rx} \quad (4.2)$$

όπου

$EIRP$: αποδοτική ισχύς ισότροπης μετάδοσης σε dBm ή dBW
L_{Path} : η συνολική εξασθένισης σήματος στη διαδρομή
G_{Rx} : το κέρδος της κεραίας του δέκτη
TH_{Rx} : το κατώφλι εισερχόμενου σήματος / ευαισθησία του δέκτη

4.1.2 Απώλεια Διαδρομής (Path Loss)

Ο προσδιορισμός / πρόβλεψη της εξασθένισης του σήματος στη διαδρομή από τον πομπό στο δέκτη, είναι ένας πολύ σημαντικός παράγοντας κατά τον σχεδιασμό ασύρματων δικτύων επικοινωνιών. Αποτελεί το κύριο αντικείμενο ενδιαφέροντος του προϋπολογισμού σύνδεσης καθότι συνδέεται άμεσα με την μετάδοση των ραδιοσυχνοτήτων. Η απώλεια διαδρομής περιλαμβάνει όλες τις απώλειες που υφίσταται το μεταδιδόμενο σήμα στην πορεία του από τον πομπό στον δέκτη. Η εξασθένιση που προκύπτει για την ισχύ από τα μοντέλα 'Απώλειας Διαδρομής' είναι η μέση τιμή γύρω από την οποία βρίσκονται οι απώλειες των λαμβανόμενων δειγμάτων στο δέκτη. Οι επιπλέον απώλειες εξαρτώνται κυρίως από την συχνότητα του ραδιοκύματος [5][2].

4.1.2.1 Μοντέλο Μετάδοσης Ελεύθερου Χώρου (Free Space Propagation Model)

Σε κάθε τύπο σύνδεσης αποδίδεται η άρχουσα διαδρομή επικοινωνίας. Αυτή για τον ανοιχτό χώρο είναι η γραμμή οπτικής επαφής (Line Of Sight) "LOS". Δηλαδή ο κύριος τρόπος μετάδοσης των ραδιοκυμάτων είναι ευθέως από τον πομπό στο δέκτη. Αυτό δεν σημαίνει ότι δεν υπάρχουν φαινόμενα πχ ανακλάσεων ή σκέδασης και διασποράς, αλλά αν υπάρχουν, τότε είναι δευτερευούσης σημασίας και δεν κρίνουν την επικοινωνία στον ίδιο βαθμό με την ευθεία επαφή. Η μετάδοση κατά την οπτική γραμμή επαφής μοντελοποιείται από το 'μοντέλο μετάδοσης ελεύθερου χώρου' (free space propagation model) "FSP" και η εξασθένηση που αυτό ορίζει καλείται 'Απώλεια Ελεύθερου Χώρου' (free space loss) L_{FS} .

Το "FSP" μοντέλο ανήκει στην κατηγορία των μοντέλων μεγάλης κλίμακας. Ως εκ τούτου προβλέπει ότι η εκπεμπόμενη ισχύς φθίνει σε συνάρτηση με το αντίστροφο της απόστασης πομπού - δέκτη, υψωμένη σε μια δύναμη α . Στην απλή περίπτωση του ελεύθερου χώρου η ισχύς ελαττώνεται σε σχέση με το αντίστροφο του τετραγώνου της απόστασης ($\alpha = 2$). Η σχέση ανάμεσα στην ισχύ εκπομπής P_t και την ισχύ λήψης P_r στον ελεύθερο χώρο δίνονται από την εξής σχέση [4][2]:

$$\frac{P_{Rx}}{P_{Tx}} = G_{Tx} \cdot G_{Rx} \cdot \left(\frac{\lambda}{4 \cdot \pi \cdot d} \right)^2 \quad (4.3)$$

όπου:

G_{Tx} : το κέρδος της κεραίας του πομπού
G_{Rx} : το κέρδος της κεραίας του δέκτη
λ : το μήκος κύματος του εκπεμπόμενου σήματος σε μέτρα ($\lambda = c / f$, όπου c η ταχύτητα του φωτός στο κενό $(3 \cdot 10^8 \ m/s)$ και f η συχνότητα του ραδιοκύματος)
d : η απόσταση ανάμεσα σε πομπό και δέκτη σε μέτρα

Η παραπάνω σχέση ονομάζεται εξίσωση εξασθένισης ελεύθερου χώρου ή *απλή εξίσωση εκπομπής του Friis*. Ο παράγοντας της παρένθεσης αποτελεί στη σχέση αυτή την εξασθένιση. Σε decibels (dB) η παραπάνω εξίσωση παίρνει την ακόλουθη μορφή:

$$10 \cdot \log(P_{Rx}) = 10 \cdot \log(P_{Tx}) + G_{Rx} + G_{Tx} + 20 \cdot \log(\lambda) - 20 \cdot \log(4 \cdot \pi) - 20 \cdot \log(d) \quad (4.4)$$

όπου πλέον τα κέρδη για τις κεραίες εκφράζονται σε dB.

Αν στην πρώτη εξίσωση θέσουμε για απόσταση τιμή ένα (1), τότε έχουμε την σχέση [2]:

$$P_{R0} = P_{Tx} \cdot G_{Tx} \cdot G_{Rx} \cdot \left(\frac{\lambda}{4 \cdot \pi} \right)^2 \quad (4.5)$$

όπου P_0 είναι η ισχύς που λαμβάνουμε στο ένα μέτρο από την κεραία του πομπού. Επομένως για την ισχύ στο δέκτη μπορούμε να γράψουμε:

$$P_{Rx} = \frac{P_{R0}}{d^2} \quad (4.5)$$

και σε decibels:

$$10 \cdot \log(P_{Rx}) = 10 \cdot \log(P_{R0}) - 20 \cdot \log(d) \quad (4.6)$$

Η παραπάνω σχέση είναι μια δεύτερη έκφραση της ισχύος λήψης συναρτήσει της απόστασης και της ισχύος σε μια γνωστή απόσταση. Αν αντί για ένα μέτρο είχαμε κάποια άλλη γνωστή απόσταση d_0 και την ισχύ σε αυτή τότε η εξίσωση γίνεται:

$$10 \cdot \log(P_{Rx}) = 10 \cdot \log(P_{R0}) - 20 \cdot \log\left(\frac{d}{d_0} \right) \quad (4.7)$$

Στις παραπάνω σχέσεις δεν περιλαμβάνονται άλλες απώλειες. Αν θέλουμε απαραίτητα να εισάγουμε όλες τις επιπλέον απώλειες του συστήματος στην εξίσωση του Friis, τότε αυτές μπορούν στην πιο απλή μορφή να αντιπροσωπευθούν από έναν παράγοντα στον παρονομαστή του κλάσματος. Συμβολίζοντας με L_S τον παράγοντα αυτό (L_S >= 1) η παραπάνω γενική εξίσωση γίνεται [4]:

$$\frac{P_{Rx}}{P_{Tx}} = G_{Tx} \cdot G_{Rx} \cdot \left(\frac{\lambda}{4 \cdot \pi \cdot d} \right)^2 \cdot \frac{1}{L_S} \quad (4.8)$$

Από την εξίσωση του Friis η εξασθένηση της διαδρομής εκφράζεται ως η διαφορά σε decibels της εκπεμπόμενης ισχύος από τον πομπό μείον την ισχύ που φτάνει στο δέκτη. Ως εκ τούτου η τιμή της εξασθένησης είναι θετική, αφού πάντα η ισχύς εκπομπής είναι μεγαλύτερη της ισχύος λήψης. Η σχέση είναι [4]:

$$PL(db) = 10 \cdot \log\left(\frac{P_{Tx}}{P_{Rx}} \right) \quad (4.9)$$

Σε αρκετές βιβλιογραφίες θεωρείται η εξασθένηση αρνητική, ως η αντίθετη διαφορά. Αναλόγως της προτίμησης πρέπει να ακολουθούμε μια μόνο παραδοχή προκειμένου να έχουμε σταθερό πρόσημο στις σχέσεις. Στην παρούσα εργασία η εξασθένηση κάθε μορφής σε dB θεωρείται θετική και συνεπώς αφαιρείται στις εξισώσεις που συμμετέχει.

4.1.2.2 Απώλεια Διαδρομής Λογαριθμικής Απόστασης (Log-Distance Path Loss Model)

Τόσο από θεωρητικής άποψης, όσο και με την εξέταση μετρήσεων, τα μοντέλα μετάδοσης ραδιοσυχνοτήτων καταδεικνύουν ότι η μέση εισερχόμενη ισχύς στο δέκτη φθίνει λογαριθμικά με την απόσταση από τον πομπό. Αυτό ισχύει τόσο σε επικοινωνία εσωτερικού χώρου όσο και εξωτερικού [4].

Η μέση εξασθένιση για μια απόσταση πομπού - δέκτη d εκφράζεται συναρτήσει αυτής χρησιμοποιώντας έναν εκθέτη εξασθένισης διαδρομής n :

$$L(d)=\left(\frac{d}{d_0}\right)^n \quad (4.10)$$

$$L(dB)=L_0(dB)+n\cdot 10\cdot \log\left(\frac{d}{d_0}\right) \quad (4.11)$$

όπου:

L : η μέση τιμή της εξασθένισης για απόσταση d
d : η απόσταση πομπού δέκτη
L_0 : η μέση τιμή της εξασθένισης σε μια γνωστή απόσταση d_0
d_0 : μια γνωστή απόσταση για την οποία γνωρίζουμε την μέση εξασθένιση L_0
n : ο εκθέτης εξασθένισης διαδρομής

Από την λογαριθμική έκφραση του μοντέλου φαίνεται ότι ο εκθέτης n εκφράζει τον ρυθμό με τον οποίο η εξασθένιση αυξάνει λόγω απόστασης. Η απόσταση d_0 όπως και η εξασθένιση L_0 είναι τα μεγέθη αναφοράς και το ιδανικό είναι να έχουν υπολογιστεί από μετρήσεις ακριβείας στην διαδρομή μετάδοσης, δηλαδή στο ίδιο περιβάλλον [4].

Η τιμή του εκθέτη εξαρτάται από το περιβάλλον. Η απόσταση αναφοράς και η αντίστοιχη εξασθένιση πρέπει να αναφέρονται σε μετάδοση ελεύθερου χώρου στο κατάλληλο περιβάλλον. Η απόσταση αναφοράς πρέπει οπωσδήποτε να βρίσκεται στην περιοχή μακρινού πεδίο της κεραίας, ώστε οι επιδράσεις του εγγύτερου πεδίου να μην αλλοιώνουν την εξασθένιση αναφοράς [4].

Η εξασθένιση αναφοράς L_0 υπολογίζεται με το μοντέλο 'Μετάδοσης Ελεύθερου Χώρου' για απόσταση d_0, ή με μετρήσεις στο πεδίο. Τυπικές τιμές του εκθέτη εξασθένισης διαδρομής για διάφορα περιβάλλοντα μετάδοσης ραδιοκυμάτων παρουσιάζονται στον παρακάτω πίνακα [4]:

Περιβάλλον	Εκθέτης Εξασθένισης Διαδρομής n
Ελεύθερος Χώρος	2
Αστικός Χώρος	2.7 - 3.5
Αστικός Χώρος σε 'Σκίαση'	3 - 5
Μέσα σε Κτίρια με LOS	1.6 – 1.8
Μέσα σε Κτίρια με Εμπόδια	4 - 6
Μέσα σε Εργοστάσια με Εμπόδια	2 - 3

Πίνακας 4.1

4.1.3 Επίγεια Μοντέλα Μετάδοσης (Near-Earth Propagation Models)

Πολλές εφαρμογές συστημάτων ασύρματης επικοινωνίας απαιτούν μετάδοση ραδιοκυμάτων ή και μικροκυμάτων ανάμεσα σε σημεία που βρίσκονται πολύ κοντά στην επιφάνεια της Γης. Ως εκ τούτου υπεισέρχονται επιπλέον απώλειες στην μετάδοση εξαιτίας του αναγλύφου, της βλάστησης και άλλων εμποδίων που συναντάμε σε αυτό το περιβάλλον επικοινωνίας. Οι απώλειες αυτές οφείλονται σε διάφορα φαινόμενα που παραποιούν τη μετάδοση, τα κυριότερα των οποίων είναι η ανάκλαση, η περίθλαση, η σκέδαση, η διασπορά η απορρόφηση [5].

Στο σενάριο που εξετάζουμε οι κόμβοι του δικτύου βρίσκονται εξολοκλήρου σε δασική έκταση με έντονες υψομετρικές διαφορές. Ως εξ τούτου θα υπολογίσουμε επιπλέον απώλειες λόγω βλάστησης και θα εξετάσουμε μοντέλα αναγλύφου μεγάλης κλίμακας.

Προκειμένου να ενσωματώσουμε τις απώλειες αυτές στον συνολικό υπολογισμό, εισάγονται νέα μοντέλα για κάθε ξεχωριστή περίπτωση, τα οποία είτε προσθέτουν εξασθένηση σε ένα κύριο μοντέλο, είτε εξολοκλήρου ορίζουν την εξασθένιση υπό τις συνθήκες που εκπροσωπούν.

4.1.3.1 Μοντέλα Εξασθένισης Βλάστησης (Foliage Models)

Πολλές έρευνες έχουν γίνει για τον χαρακτηρισμό και την μοντελοποίηση της επίδρασης της βλάστησης στην μετάδοση των ραδιοκυμάτων και ως εκ τούτου έχουν δημιουργηθεί αρκετά μοντέλα που εξετάζουν και διαφορετικές συνθήκες βλάστησης. Τα μοντέλα αυτά παρέχουν μια εκτίμηση της επιπλέον εξασθένησης σήματος που οφείλεται στην παρεμβολή βλάστησης, η οποία προστίθεται στην κύρια εξασθένηση της μετάδοσης [5].

4.1.3.1.1 Μοντέλο του Weissberger

Το μοντέλο εκθετικής φθοράς σήματος του Weissberger δίνεται από τους εξής τύπους:

$$L(db) = 1.33 \cdot F^{0.284} \cdot d_f^{0.588} \quad (4.12) \qquad 14 < df \leq 400\,m$$

$$L(db) = 0.45 \cdot F^{0.284} \cdot d_f \quad (4.13) \qquad 0 < df \leq 14\,m$$

όπου:

d_f : το βάθος της βλάστησης κατά μήκος της γραμμής οπτικής επαφής σε μέτρα
F : η συχνότητα σε GHz

Ο Weissberger εξέτασε αρκετά μοντέλα εκθετικής εξασθένισης σήματος σε όρους dB ανά μέτρο και χρησιμοποίησε για σύγκριση ένα ευρύ σετ πειραματικών δεδομένων για συχνότητες από 23 MHz έως 95 GHz. Εκ της μελέτης του αυτής κατέληξε στο συμπέρασμα ότι το εύρος της εξασθένισης εξαρτάται από την εποχή του χρόνου, δηλαδή από την κατάσταση της βλάστησης και την υγρασία που περιέχει, τη συχνότητα μετάδοσης του σήματος και την απόσταση την μετάδοσης που βρίσκεται εντός της βλάστησης. Το μοντέλο του Weissberger εφαρμόζεται σε περιπτώσεις όπου η οπτική επαφή εμποδίζεται από πυκνή και στεγνή βλάστηση, η οποία φέρει φυλλώματα (in leaf) [3][5].

4.1.3.1.2 Early ITU-Recommendation

Το πρώιμο μοντέλο βλάστησης ITU είναι ένα εμπειρικό μοντέλο. Παρότι υπάρχουν νεότερες εκδόσεις του μοντέλου η πρώιμη έκδοση εφαρμόζεται εύκολα και δίνει αποτελέσματα αρκετά κοντά στο Weissberger μοντέλο. Το μοντέλο υλοποιείται με την παρακάτω σχέση [3][5]:

$$L(db) = 0.2 \cdot F^{0.3} \cdot d_f^{0.6} \quad (4.14)$$

όπου:

d_f : το βάθος της βλάστησης κατά μήκος της γραμμής οπτικής επαφής σε μέτρα
F : η συχνότητα σε MHz

Η παραπάνω σχέση έχει φανεί ότι συμφωνεί με πειραματικές μετρήσεις στα 1800 MHz [Parson 53].

4.1.3.1.3 COST235 Μοντέλο Βλάστησης

Το μοντέλο εξασθένισης COST235 έχει δύο εκφράσεις για την απώλεια σήματος λόγω βλάστησης, προσαρμόζοντας τον υπολογισμό στην κατάσταση του φυλλώματος των δέντρων. Έτσι διακρίνονται οι περιπτώσεις στις οποίες τα δέντρα έχουν φύλλωμα (in leaf) και εκείνες στις οποίες δεν έχουν (out of leaf) [3]. Οι σχετικές εξισώσεις είναι:

a) $L(db) = 15.6 \cdot F^{-0.009} \cdot d_f^{0.26} \quad (4.15)$: δέντρα με φύλλωμα (in leaf)

b) $L(db) = 26.6 \cdot F^{-0.2} \cdot d_f^{0.5} \quad (4.16)$: δέντρα χωρίς φύλλωμα (out of leaf)

όπου:

d_f : το βάθος της βλάστησης κατά μήκος της γραμμής οπτικής επαφής σε μέτρα
F : η συχνότητα σε MHz

Οι "εποχικές" διαφορές που εμφανίζουν οι δύο σχέσεις είναι της τάξεως των 4-6 dB [3].

4.1.3.2 Μοντέλα Αναγλύφου (Terrain Models)

Η επικοινωνία μόνο κατά την γραμμή οπτικής επαφής 'LOS', είναι ένα ιδεατό σενάριο για επίγειες μεταδόσεις ραδιοσυχνοτήτων. Συνεπώς το απλό μοντέλο 'Εξασθένισης Ελεύθερης Διαδρομής' είναι στις περισσότερες αυτών των περιπτώσεων ανακριβές και ανεπαρκές [4]. Για τα επίγεια συστήματα επικοινωνιών, σημαντική επηρεή στην μετάδοση των ραδιοκυμάτων ασκεί ο τύπος του εδάφους πάνω από το οποίο μεταδίδονται. Ως έδαφος σε αυτή την προσέγγιση θεωρούμε όλα τα γεωγραφικά χαρακτηριστικά της γης πάνω από την οποία πραγματοποιείται η μετάδοση. Εξαιρούνται αυτών η βλάστηση και οι ανθρώπινες, κυρίως οικοδομικές, κατασκευές.

Όταν το έδαφος είναι επίπεδο και δεν παρουσιάζει μεγάλη τραχύτητα, αρκεί να λάβουμε υπόψη ανακλάσεις και περίθλαση επί του ορίζοντα αν οι αποστάσεις που εξετάζουμε είναι μεγάλες. Στην περίπτωση που έχουμε ωστόσο αρκετά τραχύ και μεταβαλλόμενο ανάγλυφο, τότε αυτό μπορεί να παράγει ανακλάσεις, περίθλαση, διάχυση, αποκλεισμό μετάδοσης, απορρόφηση, σκίαση και άλλα φαινόμενα επί του μεταδιδόμενου κύματος, ακόμα και σε πολύ κοντινές αποστάσεις.

Τα μοντέλα εδάφους επιδιώκουν συνήθως να δώσουν μια εκτίμηση για τη ενδιάμεση τιμή εξασθένησης (50%) συναρτήσει της απόστασης και της τραχύτητα του εδάφους. Ως ενδιάμεση τιμή θεωρείται αυτή που δεν θα ξεπεραστεί στο 50% των θέσεων στην περιοχή ή στο 50% των περιπτώσεων που εξετάζουμε [3]. Αν για κάποιο μοντέλο μπορεί να προβλεφθεί η εξασθένηση για ποσοστό πχ 90%, τότε θα μιλάμε για μια τιμή που θα υπερβαίνεται μόνο στο 10% των περιπτώσεων. Η μεταβλητότητα γύρω από κάθε πρόβλεψη λόγω άλλων φαινομένων πρέπει να εξετάζεται ξεχωριστά [5].

4.1.3.2.1 Μοντέλο Εδαφικής Ανάκλασης Δύο Δρόμων

Για αποστάσεις μικρότερες μερικών δεκάδων χιλιομέτρων είναι επιτρεπτό να παραδεχόμαστε την γήινη επιφάνεια ως λεία και επίπεδη, παραβλέποντας έτσι την καμπυλότητα της Γης [3]. Το μοντέλο 'Εδαφικής Ανάκλασης Δύο Δρόμων' (Plane Earth Propagation Model / Ground Reflection 2-Ray Model / Ground Bounce Model / Multipath Model) συνυπολογίζει την άμεση διαδρομή μετάδοσης κατά την ευθεία οπτικής επαφής 'LOS', και μια δεύτερη διαδρομή λόγω ανάκλασης στην επιφάνεια της γης, ανάμεσα στον πομπό και τον δέκτη [4]. Το μοντέλο αυτό έχει φανεί εξαιρετικά ακριβές για την πρόβλεψη σε μεγάλη κλίμακα της ισχύος μεταδιδόμενου σήματος σε αποστάσεις μερικών χιλιομέτρων.

Στον δέκτη φτάνουν δύο ραδιοκύματα, το ευθέως μεταδιδόμενο που καλύπτει την ευθεία απόσταση ανάμεσα σε πομπό και δέκτη, και το ανακλώμενο στην επιφάνεια της γης.

Η σχέση που εκφράζει το μοντέλο είναι [3]:

$$P_{Rx} = P_{Tx} \cdot G_{Tx} \cdot G_{Rx} \cdot \left(\frac{h_{Tx} \cdot h_{Rx}}{d^2} \right)^2 \quad (4.17)$$

που είναι απλούστευση της παρακάτω σχέσης για πολύ μεγάλο d :

$$P_{Rx} = 4 \cdot P_{Tx} \cdot \left(\frac{\lambda}{4 \cdot \pi \cdot d} \right)^2 \cdot G_{Tx} \cdot G_{Rx} \cdot \sin^2 \left(\frac{2 \cdot \pi \cdot h_{Tx} \cdot h_{Rx}}{\lambda \cdot d} \right) \quad (4.18)$$

Η εξίσωση αυτή αποκαλείται 'εξίσωση μετάδοσης ομαλής γης' (plane earth propagation equation). Διαφέρει δε από την εξίσωση μετάδοσης σε ελεύθερο χώρο κατά τα εξής [3]:

- Υπολογίζει τα ύψη των κεραιών πομπού και δέκτη και κάνει την παραδοχή ότι $d \gg h_{Tx}$, h_{Rx} ώστε να απλουστευθεί η εξίσωση
- Δίνει εκτίμηση βάσει του αντιστρόφου της τέταρτης δύναμης της απόστασης και όχι της δεύτερης. Αυτό καταδεικνύει μια πιο ραγδαία ελάττωση της ισχύος συναρτήσει της απόστασης.

Η συνθήκη $d \gg h_{Tx}$, h_{Rx} είναι αναγκαία προκειμένου να εφαρμόσουμε την απλή σχέση του μοντέλου. Σε πιο κοντινές αποστάσεις στον δέκτη πρέπει να εξετάσουμε με την σύνθετη εξίσωση, η οποία δίνει ακρότατα για την εξασθένηση [3].

4.1.3.2.2 Μοντέλο του Egli

Ο Egli παρατήρησε σε μετρήσεις μεταξύ 90 και 1000 MHz πάνω σε ανώμαλο ανάγλυφο, ότι υπήρχε μια τάση για τη μέση τιμή της ισχύος του σήματος σε μια μικρή περιοχή να είναι αντιστρόφως ανάλογη της τέταρτης δύναμης της απόστασης από τον πομπό. Αυτό τον οδήγησε στο να δημιουργήσει ένα μοντέλο εδάφους βασισμένο στη θεωρία μετάδοσης ραδιοσυχνοτήτων πάνω σε επίπεδη γη. Το μοντέλο που δημιούργησε ο Egli είναι μια τροποποίηση του μοντελου 'Εδαφικής Ανάκλασης Δύο Δρόμων'. Η δε τροποποίηση οφείλεται στο γεγονός ότι παρατήρησε σημαντικές αποκλίσεις του μοντέλου για διαφορετικούς τύπου εδάφους και διαφορετικές συχνότητες. Εισήγαγε έτσι έναν παράγοντα που να αντιπροσωπεύει αυτή την διακύμανση, ως πολλαπλασιαστή στην 'εξίσωση μετάδοσης ομαλής γης'. Η εξίσωση του Egli για την ενδιάμεση (50%) εξασθένηση είναι [3]:

$$L_{50} = G_{Tx} \cdot G_{Rx} \cdot \left(\frac{h_{Tx} \cdot h_{Rx}}{d^2} \right)^2 \cdot \beta \quad (4.19)$$

όπου: $$\beta = \left(\frac{40}{f} \right)^2 \quad (4.20)$$

ο παράγοντας που εισάγει ο Egli για να διορθώσει τις αποκλίσεις του απλού μοντέλου και f η συχνότητα σε Mhz. [3]

Από την έκφραση του παράγοντα β είναι προφανές ότι η συχνότητα των 40 MHz χρησιμοποιείται ως η συχνότητα αναφοράς, στην οποία η ενδιάμεση εξασθένιση που συμφωνεί με το μοντέλο του Egli τείνει να είναι ίδια με την τιμή που δίνει η 'εξίσωση μετάδοσης ομαλής γης' του μοντέλου 'Εδαφικής Ανάκλασης Δύο Δρόμων', ανεξάρτητα της ανωμαλίας του αναγλύφου [3].

Στην πράξη ο Egli ανακάλυψε ότι ο παράγοντας β είναι συνάρτηση της ανωμαλίας του αναγλύφου, ώστε η ενδιάμεση τιμή της να προκύπτει σύμφωνα με την εξίσωση $\beta = (40/f)^{2.}$

4.2 Διασπορά Μεγάλης Κλίμακας (Large Scale Fading / Slow Fading / Shadowing)

Τα περισσότερα μοντέλα υπολογισμού της εξασθένισης της ισχύος μετάδοσης σήματος που εφαρμόζονται στα ασύρματα δίκτυα επικοινωνιών, όπως και αυτά που παρουσιάστηκαν προηγουμένως, συνήθως υπολογίζουν μια μέση τιμή για την εξασθένιση ή τη μεσαία τιμή αυτής σε ένα εύρος μιας κατανομής (πχ Egli). Δίνουν δηλαδή μοναδικό αποτέλεσμα για τις ίδιες συνθήκες μετάδοσης (απόσταση, κέρδος κεραίας, ύψος κεραίας, συχνότητα, κλπ). Ωστόσο, σε ρεαλιστικά περιβάλλοντα, το μεταδιδόμενο σήμα από ένα πομπό σε ένα δέκτη δεν φτάνει ποτέ με την ίδια τιμή ισχύος στον δέκτη, ακόμα και με τις ίδιες τελείως συνθήκες επικοινωνίας. Αυτό οφείλεται με μια λέξη στην τυχαιότητα των φαινομένων στην φύση.

Στην μετάδοση ραδιοκυμάτων σε επίγεια συστήματα η τυχαιότητα αυτή γεννάται από τις επιδράσεις του περιβάλλοντος και τον εμποδίων που αυτό προβάλλει στα εκπεμπόμενα κύματα. Τα εμπόδια αυτά μπορεί να είναι η τραχύτητας του εδάφους, η φυτοκάλυψης, οι ατμοσφαιρικές συνθήκες, τεχνητά εμπόδια κλπ. Τα ραδιοκύματα υπό την επίδραση αυτών υφίστανται περίθλαση, διάχυση, ανακλάσεις, διασπορά κλπ, με αποτέλεσμα κάθε κύμα, και παραγόμενο αυτού κύμα, να ακολουθούν διαφορετική πορεία προς τον δέκτη, αν τελικά φτάσουν με την εκάστοτε διαδρομή. Ο κάθε παλμός δηλαδή που μεταδίδεται από την κεραία του πομπού μεταλλάσσεται από τα εμπόδια της μετάδοσης και διαφοροποιείται από τους υπόλοιπους. Το φαινόμενο αυτό καλείται διασπορά παλμού.

Εν τέλει το κύμα που λαμβάνει ο δέκτης κάθε χρονική στιγμή είναι στην πραγματικότητα υπέρθεση όλων των κυμάτων που προέρχονται από όλες τις κατευθύνσεις και διαφορετικές διαδρομές και φθάνουν ταυτόχρονα στο δέκτη την συγκεκριμένη χρονική στιγμή. Ο τρόπος αυτός μετάδοσης ραδιοκυμάτων καλείται 'Μετάδοση Πολλαπλής Διαδρομής' (Multipath Propagation). Σύμφωνα με αυτή το λαμβανόμενο σήμα στην κεραία του δέκτη είναι ένα διανυσματικό άθροισμα, θεωρητικά άπειρων, αντιγράφων του εκπεμπόμενου σήματος, γεννημένων από τους μηχανισμούς μετάδοσης στο συγκεκριμένο περιβάλλον. Η ισχύς του τελικού σήματος, δηλαδή η ισχύς του διανυσματικού αθροίσματος των εισερχόμενων σημάτων μια δεδομένη στιγμή, είναι μεταβλητή αναλόγως της κατανομής της φάσης ανάμεσα στους προσθετέους. Αυτή η μεταβλητότητα στην ισχύ του σήματος καλείται Διασπορά και παίρνει τόσο θετικές όσο και αρνητικές τιμές (Fading)[3].

Η Διασπορά, όπως και τα περισσότερα φαινόμενα στην μετάδοση ραδιοσυχνοτήτων, παρουσιάζεται ποικιλότροπα, σύμφωνα με το εύρος της περιοχής που εμφανίζεται και την έκταση των φαινομένων που την προκαλούν. Ως εκ τούτου ορίζονται και διαφορετικά μοντέλα 'Διασποράς' για όλες τις περιπτώσεις. Χονδρικά αυτά ομαδοποιούνται σε δύο κατηγορίες [Seybold 163]:

- Διασπορά Μεγάλης Κλίμακας / Σκίαση / Αργή Διασπορά (Large Scale Fading / Shadowing / Slow Fading)

Η Διασπορά Μεγάλης Κλίμακας σχετίζεται με την θέση και την απόσταση και ουσιαστικά εφαρμόζεται στην μέσες / ενδιάμεσες τιμές ισχύος / εξασθένισης που υπολογίζουμε με τα μοντέλα υπολογισμού εξασθένισης διαδρομής. Έχει δηλαδή εφαρμογή όταν οι απόσταση ανάμεσα στον πομπό και το δέκτη είναι αρκετά μεγάλη ώστε να παρουσιάζονται μεγάλες διακυμάνσεις στη συνολική διαδρομή μετάδοσης των ραδιοκυμάτων [3]. Επειδή συνήθως οι διακυμάνσεις αυτές παρατηρούνται σε κινούμενους δέκτες όταν βρίσκονται στην σκιά κτιρίων ή λόφων, δηλαδή πολύ κοντά σε αυτούς, το φαινόμενο ονομάζεται συχνά και Σκίαση [3][2]. Ακόμα αποκαλείται συχνά και ως Αργή Διασπορά επειδή οι μεταβολές στην ισχύ του σήματος λόγω εξασθένισης απόστασης είναι πιο αργές από άλλα φαινόμενα όπως της εξασθένισης Πολλαπλής Διαδρομής[2]. Ωστόσο ο όρος Αργή Διασπορά δεν είναι αποκλειστικός για τη Διασπορά Μεγάλης Κλίμακας.

Δεν υπάρχει αυτόνομο πλήρες μοντέλο για την Σκίαση [3]. Μετρήσεις ωστόσο έχουν δείξει ότι η μέση εξασθένιση διαδρομής εφαρμόζεται πολύ καλά σε μια Λογαριθμική Κανονικά Κατανομή με τυπική απόκλιση που εξαρτάται από τη συχνότητα μετάδοσης και το περιβάλλον. Ως εκ τούτου συχνά χρησιμοποιείται και ο όρος Λογαριθμο-Κανονική Διασπορά για το φαινόμενο [3].

- Διασπορά Μικρής Κλίμακας (Small Scale Fading)

Η Διασπορά Μικρής Κλίμακας σχετίζεται με πολύ μικρό χώρο αναφοράς και οφείλεται σχεδόν αποκλειστικά σε αλλαγές στην γεωμετρία της Πολλαπλής Διαδρομής (σε τοπική κλίμακα και όχι στο σύνολο της διαδρομής μακροσκοπικά). Ως εκ τούτου οι μεταβολές στην ισχύ συμβαίνουν συχνά και ταχύτατα. Το φαινόμενο γενικά μπορεί να χαρακτηριστεί από μια Συνάρτηση Πυκνότητας Πιθανότητας Rayleigh ή Rician [5].

Συνεπώς για την μελέτη σεναρίων συνδεσιμότητας σε ασύρματα δίκτυα επικοινωνιών, η μέση / μεσαία τιμή της εξασθένισης δεν μπορεί σε καμία περίπτωση να είναι μοναδικό μέτρο εκτίμησης της συνδεσιμότητας δύο κόμβων ή της ποιότητας αυτής. Η ανάλυση / πρόβλεψη της συνολικής εξασθένισης σε ένα επίγειο σύστημα ασύρματων επικοινωνιών μπορεί να εξεταστεί σε τρία στάδια[5]:

I. Καταρχήν εκτιμούμε την μέση τιμή ή την ενδιάμεση τιμή της εξασθένισης της διαδρομής (δηλαδή λόγω απόστασης), χρησιμοποιώντας κάποιο μοντέλο κατάλληλο για το περιβάλλον που εξετάζουμε. Σε αυτή πρέπει να συνυπολογίσουμε και τυχόν επιπλέον εξασθένιση λόγω βλάστησης ή κάθε άλλης μορφής της οποία την μέση / μεσαία τιμή μπορούμε να υπολογίσουμε με κάποιο μοντέλο.

II. Υπολογίζουμε την Διασπορά Μεγάλης Κλίμακας γύρω από τη μέση τιμή της συνολικής εξασθένισης, η οποία οφείλεται σε εδαφικές ανωμαλίες και άλλα εμπόδια ή και μετακίνηση των κόμβων. Αυτή μοντελοποιείται με τη συνάρτηση πυκνότητας πιθανότητας της λογαριθμικής κανονικής κατανομής.

III. Υπολογίζουμε τη Διασποράς Μικρής Κλίμακας.

Στην παρούσα μελέτη εξετάζουμε μακροσκοπικά το φαινόμενο της μετάδοσης ραδιοσυχνοτήτων, και ως εκ τούτου αναλύουμε μόνο την Διασπορά Μεγάλης Κλίμακας και παραλείπουμε το βήμα III της παραπάνω ανάλυσης. Επιπλέον προς τον υπολογισμό αυτής θεωρούμε την επίδραση μόνο των επίγειων στοιχείων και όχι ατμοσφαιρικών.

4.2.1 Στατιστική Ανάλυση Διασποράς

Με όρους στατιστικής η εξασθένιση είναι μια τυχαία μεταβλητή L. Η μέση τιμή της που υπολογίζουμε στο εκάστοτε μοντέλο είναι η "αληθής τιμή" L_X της τυχαίας μεταβλητής L. Η αληθής τιμή είναι έννοια θεωρητική που αναφέρεται στην πραγματική τιμή ενός μεγέθους, πλήρως απαλλαγμένη από σφάλματα. Από μετρήσεις δεν είναι δυνατό να υπολογίσουμε την αληθή τιμή ενός μεγέθους. Μπορούμε να υπολογίσουμε μόνο την καλλίτερη τιμή τιμή του, και αν οι μετρήσεις τείνουν στο άπειρο, να προσεγγίσουμε την ακριβή τιμή αυτού..

Η ίδια υπόθεση μπορεί να γίνει και για την ισχύ P_r που φτάνει στον δέκτη. Η τιμή της ισχύος στο δέκτη που υπολογίζεται από το μοντέλο αποτελεί την "αληθή τιμή" P_{rX} της τυχαίας μεταβλητής P_r. Αν πάρουμε μεγάλο αριθμό δειγμάτων ισχύος εισερχόμενου σήματος, η μέση τιμή αυτού του δείγματος Α, αποτελεί την καλλίτερη εκτίμηση της ακριβούς τιμής P_r μ της ισχύος. Αν το δείγμα τείνει στο άπειρο, τότε η μέση τιμή αυτού είναι και η ακριβής τιμή του.

Η Διασπορά Μεγάλης Κλίμακας εκφράζει την μεταβλητότητα της τυχαίας μεταβλητής P_r. Τα σενάρια που ενδέχεται να εξετάσουμε είναι κυρίως τα εξής:

(a) Μας δίνεται δείγμα μετρήσεων ισχύος και ζητείται η μέση τιμή αυτών και η τυπική απόκλιση.

(b) Μας δίνεται η αληθής τιμή και μια τυπική απόκλιση και ζητούνται τιμές ισχύος (δείγματα). Είναι το αντίστροφο πρόβλημα του (a) και πρόκειται για γεννήτρια τυχαίων δειγμάτων γύρω από την αληθή τιμή που δίνει ένα μοντέλο εξασθένισης διαδρομής.

(c) Μας δίνεται δείγμα μετρήσεων και ζητείται η σύγκριση της μέσης τιμής αυτών με την αληθή τιμή του μεγέθους.

(d) Μας δίνεται η μέση τιμή και η τυπική απόκλιση και ζητείται η συνάρτηση κατανομής της ισχύος στην τιμή της ευαισθησίας του δέκτη, δηλαδή η πιθανότητα η ισχύς που φτάνει στο δέκτη να είναι μεγαλύτερη από την ευαισθησία του δέκτη. Πρόκειται ουσιαστικά για την πιθανότητα να φτάσει ένα σήμα στο δέκτη (πιθανότητα συνδεσιμότητας).

(e) Για μια μέση τιμή ισχύος και μια πιθανότητα συνδεσιμότητας ζητείται η ελάχιστη τυπική απόκλιση γύρω από τη μέση τιμή, ώστε η συνάρτηση κατανομής της ισχύος στην τιμή ευαισθησίας του δέκτη να είναι ίση η μεγαλύτερη με την απαιτούμενη πιθανότητα συνδεσιμότητας.

ΜΕΡΟΣ 3ο

ΥΛΟΠΟΙΗΣΗ

SHAWN

ΥΛΟΠΟΙΗΣΗ:

Η μελέτη του πηγαίου κώδικα στο Shawn, και ειδικά των μοντέλων επικοινωνίας και ακμών, οδήγησε στις αποφάσεις για την τροποποίησή του και την εισαγωγή νέων τάξεων και συναρτήσεων. Σε αυτές συγκαταλέγεται η ενσωμάτωση του τύπου των κόμβων στην Node τάξη, και η αξιοποίηση της πληροφορίας αυτής για την φόρτωση επεξεργαστών και τον χαρακτηρισμό των κόμβων. Τροποποιείται το μοντέλο ακμών 'Lazy' και δημιουργούνται δύο καινούργια μοντέλα επικοινωνίας, 'UDG NT' και 'RF NT', ώστε να υποστηρίζεται η εξέταση της επικοινωνίας από τον τύπο των κόμβων. Το 'RF NT' ενσωματώνει επιπλέον και όλη την ανάλυση της μετάδοσης ραδιοσυχνοτήτων σε ένα πολυσύνθετο μοντέλο, το οποίο αποτελείται από πολλά επιμέρους μοντέλα υπολογισμού εξασθένισης ισχύος. Τέλος δημιουργούνται οι κατάλληλοι επεξεργαστές και εργασίες για να υποστηρίξουν τις υποδομές αυτές.

5. Shawn: Υλοποίηση Προσομοίωσης

Το μοντέλο επικοινωνίας στο Shawn είναι υπεύθυνο για την απάντηση στο ερώτημα αν δύο κόμβοι μπορούν να επικοινωνήσουν μεταξύ τους. Η προεπιλεγμένη υλοποίηση σε όλα τα ανεπτυγμένα μοντέλα επικοινωνίας του Shawn δεν εξετάζει το είδος των κόμβων και πως αυτό παίζει ρόλο στην δυνατότητα επικοινωνίας μεταξύ τους [§1.1.1].

Στο σενάριο που εξετάζουμε θέλουμε να προσομοιώσουμε δύο διαφορετικά είδη κόμβων:

- Αισθητήρες (sensors)
- Πύλες (gateways)

Οι αισθητήρες είναι συσκευές οι οποίες παρακολουθούν τις τιμές που λαμβάνει μια φυσική ποσότητα από το περιβάλλον (πχ θερμοκρασία, υγρασία, πίεση κλπ) και για κάποια κρίσιμη τιμή εκπέμπουν ασύρματα κάποιο σήμα. Οι πύλες είναι συσκευές που ανιχνεύουν το ραδιοφάσμα για σήματα από αισθητήρες, προκειμένου να αποκτήσουμε γνώση ότι υπάρχει κρίσιμη κατάσταση σε κάποια περιοχή και να ενεργήσουμε. Είναι σύνηθες να συνδέονται με άλλα δίκτυα προκειμένου να ενημερώνουν για τα δεδομένα που λαμβάνουν, και γι' αυτό τους αποδίδεται η ονομασία 'Πύλη' δεδομένου ότι δρομολογούν δεδομένα από ένα δίκτυο (αυτό των αισθητήρων) σε ένα άλλο (αυτό των σταθμών ελέγχου και επεξεργασίας δεδομένων).

Οι αισθητήρες δεν έχουν λόγο να λαμβάνουν μηνύματα από το δίκτυο που βρίσκονται, παρά μόνο να στέλνουν. Είναι δηλαδή κόμβοι με μονόδρομη κατεύθυνση επικοινωνίας και μάλιστα εξερχόμενη. Στο μοντέλο ακμών του Shawn αυτό μεταφράζεται σε "CommunidationDirection CD_OUT". Αντίστοιχα οι Πύλες για το δίκτυο των αισθητήρων δεν έχουν λόγο να στέλνουν μηνύματα, παρά μόνο να λαμβάνουν αυτά που αποστέλλουν οι αισθητήρες. Συνεπώς έχουν δυνατότητα μόνο εισερχόμενης επικοινωνίας στο δίκτυο των αισθητήρων. Στο μοντέλο ακμών του Shawn αυτό μεταφράζεται σε "CommunicationDirection CD_IN". Αυτός ο χαρακτηρισμός της κατεύθυνσης επικοινωνίας για κάθε κόμβο μέλος του δικτύου των αισθητήρων είναι κρίσιμος προκειμένου να υλοποιήσουμε σωστά τα μοντέλα ελέγχου της επικοινωνίας.

Με την απαίτηση η κατεύθυνση επικοινωνίας στο ασύρματο δίκτυο των αισθητήρων να είναι για τους αισθητήρες πάντα εξερχόμενη (CD_OUT) και για τις πύλες εισερχόμενη (CD_IN), τροποποιούμε την μέθοδο *bool can_communicate_uni(const Node& u, const Node& v)* ώστε να συμμορφώνεται με τις ανάγκες του δικτύου μας. Αυτό είναι απαραίτητο διότι η προαναφερθείσα μέθοδος αξιοποιείται από το μοντέλο ακμών για να δώσει τους γείτονες ενός κόμβου, και προφανώς δεν θέλουμε οι αισθητήρες να αναφέρουν ως γείτονές τους άλλους αισθητήρες, ούτε μια πύλη να δηλώνει ότι γειτονεύει με αισθητήρες στους οποίους στέλνει και μηνύματα. Πριν κάνουμε οποιαδήποτε προσομοίωσης για να εξάγουμε αποτελέσματα στην εξέταση ενός δικτύου αισθητήρων πρέπει να εξασφαλίσουμε ότι:

- Οι αισθητήρες μόνο αποστέλλουν μηνύματα και δεν δέχονται καμία εισερχόμενη επικοινωνία. Ένας αισθητήρας μπορεί να επικοινωνήσει με κάποιον άλλο κόμβο στο δίκτυο των αισθητήρων μόνο εφόσον ο δεύτερος μπορεί να δέχεται εισερχόμενη επικοινωνία.

- Οι πύλες δεν αποστέλλουν μηνύματα στο δίκτυο των αισθητήρων παρά μόνο δέχονται. Για να μπορεί να επικοινωνήσει μια πύλη με έναν κόμβο πρέπει ο δεύτερος να υλοποιεί εξερχόμενη επικοινωνία και να στέλνει μηνύματα στο δίκτυο.

Συνεπώς οι αισθητήρες πρέπει να επικοινωνούν μόνο με πύλες, και μάλιστα με μονόδρομη κατεύθυνση. Δηλαδή στην λογική του Shawn δεν ισχύει το αντίθετο, δηλαδή δεν επικοινωνεί η πύλη με τον αισθητήρα.

5.1 Τροποποίηση της τάξης των κόμβων (node class)

Πρώτο μέλημα για να αξιοποιήσουμε τον διαφορετικό χαρακτήρα που έχει ένας αισθητήρας και μια πύλη στο θέμα της επικοινωνίας είναι να μπορούμε να τον διαχειριστούμε. Για το σκοπό αυτό στον ορισμό του node class προσθέτουμε τα εξής επιπλέον δημόσια και ιδιωτικά μέλη:

```
public:
     virtual void set_node_type( const std::string& ) throw();
     const std::string& node_type( void ) const throw();
     bool has_node_type( void ) const throw();
private:
     std::string node_type_;
     bool has_node_type_;
```

[src/sys/node.h]

Το string 'node_type' είναι μια μεταβλητή στην οποία κάθε κόμβος θα αποθηκεύει τον τύπο του. Τον ορισμό της τιμής της node_type μεταβλητής αναλαμβάνει η εικονική λειτουργία '*virtual void set_node_type(const std::string&)*' και την ανάκτηση της τιμής αυτής η '*const std::string& node_type(void)*'. Η λειτουργία 'bool has_node_type(void)' ελέγχει την μεταβλητή 'has_node_type_' και μας επιτρέπει να γνωρίζουμε αν ένας κόμβος έχει κάποιο τύπο ή όχι. Ο κώδικας των τριών λειτουργιών είναι:

```
   void
   Node::
   set_node_type( const std::string& nt )
      throw()
   {
       has_node_type_ = true;
       node_type_=nt;
   }
   const std::string&
   Node::
   node_type( void )
      const throw()
   {
      return node_type_;
   }
   bool
   Node::
   has_node_type( void )
      const throw()
   {
      return has_node_type_;
   }
```

[src/sys/node.cpp]

Με τον τρόπο αυτό κάθε κόμβος έχει όλες τις απαραίτητες μεθόδους διαχείρισης του τύπου του, μεθόδους που μπορούμε να καλέσουμε κάθε φορά που θέλουμε να τον ανακτήσουμε σε οποιοδήποτε στάδιο της προσομοίωσης.

5.1.1 Τύπος Κόμβου σε XML ετικέτα

Ένας βολικός τρόπος να έχουμε αποθηκευμένο το είδος των κόμβων είναι στο ίδιο XML αρχείο που έχουμε αποθηκευμένο το όνομα και τις συντεταγμένες τους. Για τον σκοπό αυτό δημιουργούμε μια κλειστή ετικέτα τύπου 'string' με όνομα 'node_type' και τιμή το αντίστοιχο είδος του κάθε κόμβου [14]. Η ετικέτα αυτή πρέπει να περιέχεται στην ετικέτα του κάθε κόμβου. Ακολουθεί απόσπασμα από το XML αρχείο των κόμβων της Καισαριανής:

```
<scenario>
<snapshot id="01">
      <node id="130">
            <location x="482643.800" y="4201487.850" z="352.672" />
            <tag type="string" name="node_type" value="gateway"/>
      </node>
      <node id="131">
            <location x="482643.140" y="4201487.600" z="352.672" />
            <tag type="string" name="node_type" value="sensor"/>
         </node>
      <node id="132">
            <location x="482670.810" y="4201489.210" z="358.672" />
            <tag type="string" name="node_type" value="sensor"/>
      </node>
      //...
</snapshot>
</scenario>
```

Φυσικά είναι δυνατό να δημιουργήσουμε άλλου τύπου ετικέτα, ή να εφαρμόσουμε και εντελώς διαφορετική υλοποίηση. Είναι δυνατό για παράδειγμα να δημιουργήσουμε μια ετικέτα χάρτη (map tag) που να περιλαμβάνει τον τύπο όλων των κόμβων. Η λύση αυτή είναι ενδεικτική στην περίπτωση που έχουμε πάρα πολύ μεγάλο αριθμό κόμβων ή πολλά αρχεία με πληροφορία για αυτούς, και θέλουμε να συγκεντρώσουμε σε ένα σημείο τη διαχείριση του τύπου τους. Βέβαια η επιλογή της ενσωμάτωσης στο αρχείο συντεταγμένων συνδυάζει όλη την πληροφορία του δικτύου σε ένα αρχείο και είναι εξαιρετικά βολική. Μένει οι ετικέτες να αποδοθούν στη μεταβλητή 'node_type' κάθε κόμβου.

5.1.2 Εργασία Φόρτωσης του Τύπου των Κόμβων

Όταν εκτελούμε την εργασία 'load_world' για να φορτώσουμε έναν έτοιμο κόσμο από XML αρχείο στο περιβάλλον προσομοίωσης, ο φορτωτής των ρυθμίσεων περιβάλλοντος καλεί έναν αναλυτή σύνταξης ετικετών XML και αποδίδει κάθε ετικέτα στον κόμβο που ανήκει [21]. Το Shawn διαθέτει μεθόδους για την διαχείριση των ετικετών, με τις οποίες μπορούμε να ανακτήσουμε τις τιμές από τις ετικέτες και να τις διαχειριστούμε με οποιοδήποτε τρόπο. Προκειμένου να μην καλούμε κάθε φορά αυτές τις μεθόδους τις οργανώνουμε σε μια απλή εργασία (common task) την οποία εκτελούμε μετά την φόρτωση του XML αρχείου ρυθμίσεων. Επιπλέον με τον τρόπο αυτό διατηρούμε τον κώδικά μας απομονωμένο και εύκολα επεξεργάσιμο. Η εργασία ονομάστηκε node_type_load_task και η κύρια λειτουργία της είναι:

```
   void
   NodeTypeLoadTask::
   run( shawn::SimulationController& sc )
      throw( std::runtime_error )
   {
      require_world( sc );
      for( shawn::World::node_iterator
               it = sc.world_w().begin_nodes_w();
               it != sc.world_w().end_nodes_w();
               ++it )
      {
      ConstTagHandle th = it->find_tag("node_type");
      const StringTag* st = dynamic_cast<const
StringTag*>( th.get() );
      std::string nt = st->value();
      it->set_node_type( st->value() );
      std::cout << "NODE '" << it->label() << "' is of type '"<<
it->node_type() << "' \n";
      }
   }
```

[src/legacyapps/tasks/node_type_load_task/node_type_load_task.cpp]

Όπως φαίνεται στον κώδικα εφαρμόζουμε επαναλήψεις με 'for' δήλωση για να αποδώσουμε δείκτη με δυνατότητες εγγραφής σε κάθε κόμβο. Στη συνέχεια βρίσκουμε την ετικέτα 'node_type' σε κάθε κόμβο και εξετάζουμε αν είναι του τύπου που θέλουμε (δηλαδή σε αυτή την περίπτωση 'string'). Τέλος ανακτούμε την τιμή της ετικέτας και με την λειτουργία s*et_node_type(const std::string&)* την αποθηκεύουμε στην ιδιωτική μεταβλητή 'node_type'. Για έλεγχο εκτυπώνουμε στην κύρια έξοδο τον τύπο κάθε κόμβου. Πλέον δε χρειάζεται κάθε φορά να ανακτούμε την τιμή από την ετικέτα με την παραπάνω διαδικασία, αλλά μόνο με τη μέθοδο *Node::node_type()*.

Φυσικά είναι δυνατό η παραπάνω εργασία να αποθηκευτεί σε μια λειτουργία και να κληθεί από οπουδήποτε. Όπως όμως προαναφέρθηκε δίνεται με αυτό τον τρόπο τάξη και οργάνωση στην προσομοίωση και τον κώδικα, και επιπλέον έχουμε την δυνατότητα εφόσον δεν θέλουμε να αξιοποιήσουμε την συγκεκριμένη πληροφορία να μην την φορτώσουμε, οπότε απλά δεν εκτελούμε την εργασία. Αυτή ακριβώς είναι η ουσία της οργάνωσης λειτουργιών σε εργασίες του Shawn, να μπορούμε εύκολα με μια ρύθμιση στο αρχείο ρυθμίσεων να καθορίζουμε ποιες θα εκτελεστούν, πότε θα εκτελεστούν, πόσες φορές και με τι ρυθμίσεις. Δεν πρέπει βέβαια να παραλείπουμε να εκτελούμε την συγκεκριμένη εργασία κάθε φορά που θέλουμε να χρησιμοποιήσουμε το είδος των κόμβων.

Μια άλλη εναλλακτική λύση της παραπάνω υλοποίησης είναι να ενσωματώσουμε τον παραπάνω κώδικα χωρίς την επανάληψη στο σύνολο των κόμβων, σε έναν επεξεργαστή και συγκεκριμένα στην *boot()* μέθοδο ενός προεπιλεγμένου επεξεργαστή. Επομένως πριν ξεκινήσει οποιαδήποτε διαδικασία θα μπορούμε να έχουμε ανακτήσει την τιμή του 'node_type' για κάθε κόμβο. Το μειονέκτημα αυτής της υλοποίησης είναι ότι απαιτεί την εκτέλεση γύρων προσομοίωσης, ενώ η εργασία δεν απαιτεί κάτι τέτοιο.

5.2 Τροποποίηση Μοντέλου Επικοινωνίας: Unit Disk Graph NT

Η *bool can_communicate_uni(const Node& u, const Node& v)* για το 'Unit Disk Graph' μοντέλο επικοινωνίας είναι:

```
 bool
   DiskGraphNTModel::
   can_communicate_uni( const Node& u,
                        const Node& v )
      const throw()
   {
      return euclidean_distance( u.real_position(),
v.real_position() ) <= range_;
   }
```

[src/sys/comm_models/disk_graph.cpp]

Όπως παρατηρούμε μοναδικό κριτήριο για τη δυνατότητα επικοινωνίας ανάμεσα σε δύο κόμβους 'u' και 'v' είναι η ευκλείδεια απόσταση που τους συνδέει να είναι μικρότερη του εύρους εκπομπής (range_ = transmission range). Αυτή η απλή συνθήκη βαδίζει στη λογική του Shawn να εξετάζει τα αποτελέσματα των φαινομένων και όχι τις αιτίες τους. Δηλαδή δεν μας ενδιαφέρει πως προκύπτει το εύρος εκπομπής, αρκεί να το γνωρίζουμε για να το αξιοποιήσουμε. Η μέθοδος δίνει απάντηση στο ερώτημα 'αν μπορούν δύο κόμβοι να επικοινωνήσουν' και όχι αν τελικά επικοινωνούν [§1.1.1.1].

Βλέποντας την υλοποίηση διαπιστώνει κανείς πως η μέθοδος δίνει τα ίδια αποτελέσματα είτε την καλέσουμε ως *can_communicate_uni(u, v)* είτε ως *can_communicate_uni(v, u)*. Δηλαδή ενώ υποτίθεται ότι ελέγχει μονοσήμαντα την δυνατότητα επικοινωνίας ουσιαστικά την ελέγχει αμφιμονοσήμαντα. Αυτό είναι σοβαρό πρόβλημα για το σενάριο που εξετάζουμε δεδομένου ότι δεν γίνεται να έχουμε σε καμία περίπτωση αμφίδρομη επικοινωνία στα δύο είδη κόμβων που εξετάζουμε. Η παραπάνω υλοποίηση δίνει βέβαιη αμφίδρομη επικοινωνία που είναι και η προεπιλεγμένη ρύθμιση για το UDG μοντέλο επικοινωνίας.

Για να μπορέσουμε επομένως να αξιοποιήσουμε το είδος των κόμβων στην μέθοδο αυτή πρέπει να εξετάσουμε όλα τα πιθανά σενάρια για δύο κόμβους 'u' και 'v' και να εξασφαλίσουμε ότι η *can_communicate_uni(u, v)* εξετάζει οπωσδήποτε την κατεύθυνση επικοινωνίας. Για το σκοπό αυτό υλοποιήθηκε μια τροποποιημένη μορφή του UDG μοντέλου επικοινωνίας, το 'Disk Graph NT', όπου ο όρος NT προκύπτει από τα αρχικά της φράσης 'Node Type', και θα χρησιμοποιείται και σε άλλα μοντέλα και εφαρμογές κατ'αυτό τον τρόπο.

Με το ενδεχόμενα ένας κόμβος να μην έχει κάποιο γνωστό τύπο ('has_node_type = false') διακρίνουμε τις εξής περιπτώσεις για την μέθοδο '*bool can_communicate_uni (const Node& u, const Node& v)*':

(1) Και οι δύο κόμβοι να ανήκουν σε κάποιο τύπο. Τότε εφαρμόζουμε την εξής συνθήκη:

Αν ο 'u' κόμβος είναι αισθητήρας και ο 'v' πύλη, τότε εξετάζεται η επικοινωνία με τη σύγκριση ευκλείδειας απόστασης και εύρους εκπομπής κατά την γενική μέθοδο. Σε κάθε άλλη περίπτωση ο κόμβος 'u' δεν μπορεί να επικοινωνήσει με τον 'v'.

(2) Και οι δύο κόμβοι δεν έχουν τύπο.

Τότε εξετάζεται η γενική μέθοδος και μιλάμε για δυνατότητα αμφίδρομης επικοινωνίας αφού και η can_communicate(v, u) θα δώσει το ίδιο αποτέλεσμα.

(3) Ο κόμβος 'u' έχει τύπο ενώ ο 'v' όχι. Τότε εφαρμόζουμε την εξής συνθήκη:

Αν ο κόμβος 'u' είναι αισθητήρας τότε εξετάζεται η γενική μέθοδος. Σε κάθε άλλη περίπτωση δεν είναι δυνατή η επικοινωνία.

(4) Ο κόμβος 'v' έχει τύπο ενώ ο 'u' όχι. Τότε εφαρμόζουμε την εξής συνθήκη:

Αν ο κόμβος 'v' είναι πύλη τότε εξετάζεται η γενική μέθοδος. Σε κάθε άλλη περίπτωση δεν είναι δυνατή η επικοινωνία.

Οι περίπτωση (4) είναι το αντίστροφο της (3), δηλαδή ό,τι ισχύει στην (3) για ένα ζεύγος κόμβων (u, v), ισχύει στην (4) για το ζεύγος (v, u)

Ο νέος ορισμός για την *bool can_communicate_uni (const Node& u, const Node& v)* είναι:

```
bool
DiskGraphNTModel::
can_communicate_uni( const Node& u,
                     const Node& v )
const throw()
{
   if(u.has_node_type() && v.has_node_type())
   {
     if(u.node_type()=="sensor" && v.node_type()=="gateway")
      return euclidean_distance( u.real_position(),
                                 v.real_position() ) <= range_;
   else
     return false;
   };
   if(!u.has_node_type() && !v.has_node_type())
   {
      return euclidean_distance( u.real_position(),
                                 v.real_position() ) <= range_;
   };
   if(u.has_node_type() && !v.has_node_type())
   {
   if (u.node_type()=="sensor")
     return euclidean_distance( u.real_position(),
                                 v.real_position() ) <= range_;
   else
     return false;
   };
   if(!u.has_node_type() && v.has_node_type())
   {
   if (v.node_type()=="gateway")
     return euclidean_distance( u.real_position(),
                                 v.real_position() ) <= range_;
   else
     return false;
   };
}
```

[src/sys/comm_models/disk_graph_nt_model.cpp]

Με το νέο μοντέλο επικοινωνίας και την παραπάνω υλοποίηση έχουμε καλύψει την εκτίμηση της δυνατότητας επικοινωνίας ανάμεσα σε δύο κόμβους ανάλογα με τον τύπο του καθενός. Επιπλέον διατηρήσαμε την δυνατότητα να υπάρχουν κόμβοι που να μην έχουν τύπο και να συμπεριφέρονται κατά το προεπιλεγμένο μοντέλο ως αμφίδρομης επικοινωνίας. Τέτοιοι κόμβοι μεταξύ τους μπορούν να υλοποιήσουν αμφίδρομη επικοινωνία, ενώ με αισθητήρες και πύλες μόνο μονόδρομη, εισερχόμενη στην πρώτη περίπτωση και εξερχόμενη στην δεύτερη. Το είδος της επικοινωνίας και οι μέθοδοι αξιοποίησής της είναι αντικείμενο του μοντέλου ακμών.

5.3 Τροποποίηση Μοντέλου Επικοινωνίας: RF NT

Το Shawn είναι ένας προσομοιωτής ασύρματων δικτύων αισθητήρων που δεν εξετάζει τις αιτίες των φαινομένων, αλλά μόνο τα αποτελέσματά τους. Ως εκ τούτου το θέμα της εκτίμησης της επικοινωνίας δύο κόμβων το αντιμετωπίζει με μοναδικό κριτήριο τη σχέση της απόστασής τους και του εύρους της επικοινωνίας (distance < transmission range) [§1.1.1]. Δεν εξετάζονται τα φυσικά φαινόμενα και οι αιτίες που καθορίζουν το εύρος επικοινωνίας, αλλά αυτό ορίζεται ρητά και αξιοποιείται χωρίς περαιτέρω ανάλυση. Ακόμα και στο 'RIM' μοντέλο επικοινωνίας, όπου ορίζεται διακύμανση στο εύρος και προσδίδεται τυχαιότητα στην επίτευξη επικοινωνίας, το εύρος εξακολουθεί να είναι αυθαίρετα ορισμένο, χωρίς κάποια φυσική εξήγηση [§1.1.1.4].

Η πρακτική αυτή του Shawn δεν είναι αδικαιολόγητη, ούτε εσφαλμένη. Είναι μια απλουστευμένη προσέγγιση στην επικοινωνία, η οποία δεν εξετάζει το φυσικό επίπεδο αυτής (physical layer). Υπάρχουν στοιχεία αξιοποίησης φαινομένων σε φυσικό επίπεδο, όπως πχ η απόρριψη μηνυμάτων στο μοντέλο εκπομπής 'random_drop' [§1.1.3.1.2]. Σε αυτό χρησιμοποιείται ένας παράγοντας πιθανότητας σύμφωνα με τον οποίων απορρίπτονται μηνύματα, θεωρητικά λόγω διακυμάνσεων στην εκπομπή. Ακόμα όμως και αυτή η προσέγγιση είναι απλοϊκή και δεν εξετάζει τις αιτίες της απόρριψης, παρά μόνο το γεγονός.

Στα πλαίσια της ανάλυσης της παρούσας μελέτης κρίθηκε σκόπιμο να αναλυθεί η επικοινωνία των κόμβων σε φυσικό επίπεδο και να ενσωματωθεί στην προσομοίωση. Για το σκοπό αυτό δημιουργήθηκε ένα επιπλέον μοντέλο επικοινωνίας, το 'RF NT'. Ο τίτλος οφείλεται στο ότι το μοντέλο αυτό εξετάζει την μετάδοση των ραδιοσυχνοτήτων (RF Propagation) βάσει του είδους τον κόμβων.

5.3.1 Υλοποίηση RF NT Μοντέλου

Η υλοποίηση του μοντέλου 'RF NT' βασίστηκε στην ανάλυση της μετάδοσης ραδιοσυχνοτήτων που πραγματοποιήθηκε στο κεφάλαιο 4. Είναι ως εκ τούτου προσαρμοσμένο στις συνθήκες της περιοχής που εξετάζουμε και εφαρμόζει πρακτικές και μοντέλα ανάλυσης μεγάλης κλίμακας. Ο κώδικας του ακολουθεί το πρότυπο κώδικα των μοντέλων επικοινωνίας και εν προκειμένω χρησιμοποιήθηκε ως βάση ο κώδικας του 'UDG' μοντέλου [§1.1.1.1], όπως έγινε και στην περίπτωση του 'Disk Graph NT'. Το μοντέλο υλοποιείται με την τάξη 'RFNTModel' που είναι παράγωγη της τάξης 'CommunicationModel'.

Η υλοποίηση του μοντέλου χωρίζεται σε τρεις ενότητες:

(1) 'RFNTModel'

- (a) Μέθοδοι: Οι λειτουργίες που εξετάζουν την επικοινωνία των κόμβων.
- (b) Παράμετροι: Οι παράμετροι εισόδου που χρησιμοποιεί το μοντέλο και ανήκουν στην τάξη του

(3) Config File & XML Data: Πως πρέπει να δομείται ένα αρχείο ρυθμίσεων για να αξιοποιεί το μοντέλο και πως πρέπει οι διάφοροι παράμετροι να ενσωματώνονται σε XML αρχείο δεδομένων

(2) 'RFNTCommunicationModelFactory'

(a) Παραγωγή μοντέλου: Ανάλυση της τάξης 'RFNTCommunicationModelFactory' που καλεί ο ελεγκτής της προσομοίωσης όταν αυτή εκκινεί, προκειμένου να δημιουργηθεί το μοντέλο σαν αντικείμενο αυτής.

Η παραπάνω ταξινόμηση είναι η ακριβώς αντίθετη της εμφάνισης και αξιοποίησης κάθε στοιχείου του μοντέλου στην οικοδόμηση της προσομοίωσης. Δηλαδή πρώτα δομούνται τα αρχεία ρυθμίσεων και XML δεδομένων (3) που περιλαμβάνουν και τις τιμές των παραμέτρων του μοντέλου. Στη συνέχει εκτελείται η 'Παραγωγός' τάξη του μοντέλου (2), η οποία συγκεντρώνει τις τιμές των παραμέτρων από τις ρυθμίσεις και τα XML δεδομένα και καθορίζει πως αυτές θα ενσωματωθούν στην δημιουργία του μοντέλου. Τέλος δημιουργεί το μοντέλο (1), και αποδίδει σε αυτό τις τιμές των παραμέτρων. Η παρουσίαση γίνεται ανάποδα προκειμένου να παρουσιαστούν επαρκώς οι αιτίες όλων των στοιχείων που το απαρτίζουν και να γίνει κατανοητή η λειτουργία του.

5.3.2 'RFNTModel' τάξη Μοντέλου Επικοινωνίας RF NT

5.3.2.a Μοντέλο RF NT: Μέθοδοι τάξης 'RFNTModel'

Οι μέθοδοι που υλοποιήθηκαν για το μοντέλο εφαρμόζουν όλα τα μοντέλα πρόβλεψης απώλειας διαδρομής που παρουσιάστηκαν, καθώς και βοηθητικές λειτουργίες για αυτά. Επιπλέον δημιουργήθηκαν μέθοδοι που αξιοποιούν τα αποτελέσματα αυτών και πραγματοποιήθηκε εκτενέστατη τροποποίηση της *can_communicate_uni(u, v)*. Όλες οι μέθοδοι προγραμματίστηκαν ως εικονικές λειτουργίες της 'RFNTModel' τάξης.

Οι μέθοδοι του μοντέλου οι σχετικές με την μετάδοση ραδιοσυχνοτήτων είναι οι εξής:

- 5.3.2.a.1 Μετατροπή της ισχύος εκπομπής σε dBm (βοηθητική μέθοδος)

```
virtual double transmit_power_dBm()
```

Με τη μέθοδο αυτή μετατρέπουμε την ισχύ εκπομπής Pt σε dBm. Δίνεται δηλαδή η δυνατότητα να χρησιμοποιήσουμε διαφορετικές μονάδες κατά την δήλωση της ισχύος εκπομπής. Το μοντέλο επιτρέπει η ισχύς εκπομπής να εισάγεται είτε σε milliWatts, είτε σε Watts είτε σε dBm. Η μέθοδος αξιοποιεί μια επιπλέον παράμετρο Pt_unit που δηλώνει τη μονάδα της ισχύος. Οι αντίστοιχες τιμές είναι "mW", "W" και "dBm", με ευαισθησία κεφαλαίων. Η μέθοδος εφαρμόζει λογαρίθμιση στις δύο πρώτες περιπτώσεις ώστε να προκύψει η τιμή της ισχύος σε dBm και σε κάθε περίπτωση αναθέτει την ισχύ στη μεταβλητή Pt_dBm. Αυτή γίνεται κατά τις εξής εξισώσεις:

Pt_dBm = 10.0 * log10(Pt_mW) (5.1)

Pt_dBm = 10.0 * log10(Pt_W * 1000) (5.2)

- 5.3.2.a.2 Υπολογισμός 'Εξασθένισης Ελεύθερου Χώρου' (μέθοδος υπολογισμού μέσης εξασθένισης διαδρομής) [§4.1.2.1]

```
virtual double free_space_PL_dB(const Node&, const Node&)
```

Η μέθοδος αυτή υλοποιεί το μοντέλο Εξασθένισης Ελεύθερου Χώρου (free space loss model) για τον υπολογισμό της απώλειας διαδρομής ανάμεσα σε δύο κόμβους. Είναι βασισμένη στην εξίσωση του Friis σε λογαριθμική μορφή και επιστρέφει την τιμή της εξασθένισης σε dBm. Αυτή διατυπώνεται ως εξής:

Lfs = - Gt_dB_ - Gr_dB_ - 20.0 * log10(wl_m) + 20.0 * log10(d_m) + 20.0 * log10(4 * PI) (5.3)

όπου:

Lfs: η εξασθένιση σε dBm με θετικό πρόσημο
Gt_dB: το κέρδος της κεραίας του πομπού σε dB
Gr_dB: το κέρδος της κεραίας του δέκτη σε dB
wl_m: το μήκος κύματος του εκπεμπόμενου σήματος σε μέτρα
d_m: η ευκλείδεια απόσταση των δύο κόμβων σε μέτρα
PI: ο αριθμός 'π' σε δεκαδική μορφή

- Υπολογισμός 'Εξασθένισης Λογαριθμικής Απόστασης' (μέθοδος υπολογισμού μέσης εξασθένισης διαδρομής) [§4.1.2.2] : d0=1m

```
virtual double log_distance_1m_PL_dB(const Node&, const Node&)
```

Η μέθοδος αυτή εφαρμόζει το μοντέλο Εξασθένισης Λογαριθμικής Απόστασης για να προβλέψει την απώλεια διαδρομής ανάμεσα σε δύο κόμβους. Το μοντέλο προκειμένου να εφαρμοστεί απαιτεί μια εξασθένιση αναφοράς L0 για μια γνωστή απόσταση d0. Στην πρώτη αυτή υλοποίηση του μοντέλου χρησιμοποιείται η εξασθένισης αναφοράς L0 που υπολογίζεται με το μοντέλο Ελεύθερου Χώρου για απόσταση d0 = 1m (1 μέτρο). Η εξίσωση υπολογισμού είναι:

Ld = L0 + n * 10.0 * log10(d_m) (5.4)

όπου:

Ld: Η εξασθένιση σε dBm με θετικό πρόσημο
L0: Η εξασθένιση αναφοράς στο 1 μέτρο, όπως υπολογίζεται με το μοντέλο FSP
n: Ο εκθέτης της εξασθένισης
d_m: Η ευκλείδεια απόσταση ανάμεσα σε πομπό και δέκτη

- Υπολογισμός 'Εξασθένισης Λογαριθμικής Απόστασης' (μέθοδος υπολογισμού μέσης εξασθένισης διαδρομής) [§4.1.2.2] : d0=variable

```
virtual double log_distance_PL_dB( const Node&, const Node& )
```

Αυτή είναι η δεύτερη μέθοδος που υλοποιεί το μοντέλο Εξασθένισης Λογαριθμικής Απόστασης για να προβλέψει την απώλεια διαδρομής ανάμεσα σε δύο κόμβους. Στην περίπτωση αυτή χρησιμοποιούμε ως απόσταση αναφοράς d0 κάποια γνωστή απόσταση από το δέκτη. Η εξίσωση που δίνει την εξασθένιση είναι:

Ld = L0 + n * 10.0 * log10(d_m / d0) (5.5)

όπου:

Ld: Η εξασθένιση σε dBm με θετικό πρόσημο
L0: Η εξασθένιση αναφοράς
n: Ο εκθέτης της εξασθένισης
d_m: Η ευκλείδεια απόσταση ανάμεσα σε πομπό και δέκτη
d0: Η απόσταση αναφοράς

Οι επιλογές για τον υπολογισμό της εξασθένισης αναφοράς είναι δύο:

(a) Η L0 να υπολογίζεται ως προϊόν μετρήσεων ακριβείας σε απόσταση d0 από το δέκτη

(b) Η L0 να υπολογίζεται για την απόσταση d0 με το μοντέλο 'Ελεύθερου Χώρου' και την εξίσωση του Friis

Στην μέθοδο εξετάζονται και οι δύο επιλογές. Η τακτική είναι η εξής:

Για κάθε πύλη (gateway) ελέγχεται αν ενσωματώνει ετικέτα με δείγματα μετρήσεων.

- Αν δεν περιλαμβάνει τότε:

1. Υπολογίζεται μια τιμή αναφοράς L0fs για κάθε απόσταση d μεταξύ της πύλης και των κόμβων του δικτύου

2. Υπολογίζεται η εξασθένισης Λογαριθμικής Απόστασης για κάθε ζεύγος αναφοράς L0fs - d, με την εξίσωση του Friis (β)

3. Υπολογίζεται ο αριθμητικός μέσος Ldfs_mean των τιμών εξασθένισης και επιστρέφεται στην προσομοίωση ως θετική τιμή

- Αν περιλαμβάνει τότε:

1. Πραγματοποιούνται τα βήματα 1, 2 της προηγούμενης περίπτωσης

2. Για κάθε ζεύγος δείγματος L0 και απόστασης d υπολογίζεται η Εξασθένιση Λογαριθμικής Απόστασης

3. Υπολογίζεται ο αριθμητικός μέσος Ld_mean των τιμών εξασθένισης που υπολογίστηκαν βάσει μετρήσεων

4. Συγκρίνεται η τιμή των Ldfs_mean και Ld_mean και επιστρέφεται στην προσομοίωση η μεγαλύτερη τιμή, δηλαδή η μεγαλύτερη εξασθένιση, ως θετική τιμή

Η επιλογή να επιστρέφεται η δυσμενέστερη τιμή μεταξύ των Ldfs_mean και Ld_mean δεν είναι ακριβώς σωστή αλλά αυστηρή. Είναι όμως υπέρ της λειτουργικότητας του δικτύου, διότι επιστρέφει το ανώτερο κατώφλι στην εκτίμηση της επικοινωνίας.

Οι τιμές των Ldfs_mean και Ld_mean υπολογίζονται χωρίς εκτίμηση βαρών.

- 5.3.2.a.3 Υπολογισμός 'Μεσαίας (50%) Εξασθένισης Αναγλύφου' (μέθοδος υπολογισμού μεσαίας εξασθένισης διαδρομής) [§4.1.3.2.2]

```
virtual double Egli_median_terrain_model_PL_dB( const Node&,
                                                const Node& )
```

Το μοντέλο του Egli δίνει την εξασθένηση που δε θα υπερβεί το 50% των περιπτώσεων επικοινωνίας ανάμεσα στο πομπό και το δέκτη. Η εξασθένηση αποδίδεται ως εξής:

L50 = - Gt_dB_ - Gr_dB_ - 20 * log10(hr_m_) - 20 * log10(ht_m_)
- 10 * log10(b) + 40 * log10 (d_m) (5.6)

όπου:

L50: η εξασθένιση σε dBm με θετικό πρόσημο
Gt_dB: το κέρδος της κεραίας του πομπού σε dB
Gr_dB: το κέρδος της κεραίας του δέκτη σε dB
hr_m: το ύψος της κεραίας του δέκτη σε μέτρα
ht_m: το ύψος της κεραίας του πομπού σε μέτρα
wl_m: το μήκος κύματος του εκπεμπόμενου σήματος σε μέτρα
b: ο παράγοντας εδάφους 'β' συναρτήσει της συχνότητας σε Mhz
b=pow((40 / (f_GHz_ * 1000)), 2)
d_m: η ευκλείδεια απόσταση των δύο κόμβων σε μέτρα

Παρότι το μοντέλο του Egli δίνει εξασθένηση με πιθανότητα, η εκτίμησή του προκύπτει εμπειρικά και δεν οφείλεται σε διασπορά εξασθένησης. Δεν πρόκειται δηλαδή για πιθανότητα ως συνάρτηση κατανομής της εξασθένησης. Συνεπώς η εξασθένηση που προβλέπεται από το μοντέλου υπόκειται σε επιπλέον ανάλυση για την εκτίμηση της διασποράς, όπως και στα μοντέλα που υπολογίζουν μέση εξασθένιση.

- 5.3.2.a.4 Υπολογισμός Εξασθένισης Εδαφικής Ανάκλασης Δύο Δρόμων (μέθοδος υπολογισμού μέσης εξασθένισης διαδρομής) [§4.1.3.2.1]

```
      virtual double
ground_bounce_multipath_Approximation_PL_dB( const Node&,
                                             const Node& )
      virtual double
ground_bounce_multipath_Analysis_PL_dB( const Node&,
                                        const Node& )
```

Το μοντέλο εξασθένισης 'Εδαφικής Ανάκλασης Δύο Δρόμων' υλοποιείται με δύο μεθόδους, εκ των οποίων η πρώτη εφαρμόζει την προσεγγιστική σχέση (υπό την θεώρηση ότι η απόσταση είναι πολύ μεγαλύτερη του ύψους των κεραιών), και η δεύτερη την αναλυτική σχέση:

Lmp = - Gt_dB_ - Gr_dB_ - 20.0 * log10(hr_m_) (5.7)
- 20.0 * log10(ht_m_) + 40.0 * log10(d_m)

Lmp = - 10 * log10(4) - Gt_dB_ - Gr_dB_ - 20 * log10(wl_m)
- 20 * log10(sin((2 * PI * hr_m_ * ht_m_) / (d_m * wl_m))) (5.8)
+ 20.0 * log10(d_m * 4 * PI)

όπου:

Lmp: η εξασθένιση σε dBm με θετικό πρόσημο
Gt_dB: το κέρδος της κεραίας του πομπού σε dB
Gr_dB: το κέρδος της κεραίας του δέκτη σε dB
hr_m: το ύψος της κεραίας του δέκτη σε μέτρα
ht_m: το ύψος της κεραίας του πομπού σε μέτρα
wl_m: το μήκος κύματος του εκπεμπόμενου σήματος σε μέτρα
d_m: η ευκλείδεια απόσταση των δύο κόμβων σε μέτρα
PI: ο αριθμός 'π' σε δεκαδική μορφή

Η επιλογή μιας εκ των δύο εκφράσεων της μεθόδου έχει αφεθεί σε χειροκίνητο έλεγχο μέσω των παραμέτρων της προσομοίωσης. Είναι δυνατό σε κάποια άλλη υλοποίηση να επιλέγεται μια σχέση απόστασης / ύψους κεραιών, ώστε να αποφασίζεται η χρήση της προσεγγιστικής ή της αναλυτικής εξίσωσης.

- 5.3.2.a.5 Υπολογισμός Μέσης Εξασθένισης Βλάστησης (μέθοδοι υπολογισμού εξασθένισης βλάστησης) [§4.1.3.1]

```
virtual
double weissberger_foliage_PL_dB( const Node&, const Node& )
```

Η μέθοδος αυτή εφαρμόζει το μοντέλο του Weissberger [§4.1.3.1.1]. Η σχέση που δίνει την εξασθένηση είναι:

Για απόσταση μέχρι και τα 14 μέτρα

LdB = 0.45 * pow(f_GHz_, 0.284) * df_m (5.9)

Για απόσταση μεταξύ 14 και 400 μέτρων

LdB = 1.33 * pow(f_GHz_, 0.284) * pow(df_m, 0.588) (5.10)

όπου:

LdB: Η εξασθένιση σε dBm
f_GHz: Η συχνότητα σε GHz
df_m: Το βάθος της φυτοκάλυψης σε μέτρα

```
virtual double early_ITU_foliage_PL_dB( const Node&, const Node& )
```

Η μέθοδος αυτή εφαρμόζει το πρώιμο μοντέλο ITU [§4.1.3.1.2]. Η εξασθένιση δίνεται από την εξής σχέση:

LdB = 0.2 * pow((f_GHz_ * 1000) , 0.3) * pow(df_m, 0.6) (5.11)

όπου:

LdB: Η εξασθένιση σε dBm
f_GHz: Η συχνότητα σε GHz
df_m: Το βάθος της φυτοκάλυψης σε μέτρα

```
virtual double COST235_in_leaf_foliage_PL_dB( const Node&,
                                              const Node& )
virtual double COST235_out_of_leaf_foliage_PL_dB( const Node&,
                                                  const Node& )
```

Με τις δύο αυτές μεθόδους εφαρμόζεται το μοντέλο βλάστησης COST235 [§4.1.3.1.3]. Η πρώτη μέθοδος εξετάζει βλάστηση με φύλλωμα, ενώ η δεύτερη βλάστηση χωρίς φύλλωμα. Οι σχέσεις που δίνουν την εξασθένηση είναι αντίστοιχα:

LdB = 15.6 * pow ((f_GHz_ * 1000) , -0.009) * pow(df_m, 0.36) (5.12)

LdB = 15.6 * pow ((f_GHz_ * 1000) , -0.2) * pow(df_m, 0.5) (5.13)

όπου:

LdB: Η εξασθένιση σε dBm
f_GHz: Η συχνότητα σε GHz
df_m: Το βάθος της φυτοκάλυψης σε μέτρα

- 5.3.2.a.6 Υπολογισμός Σημείου Διασταύρωσης 'crossover point' (βοηθητική μέθοδος)

```
virtual double crossover_point_m( const Node&, const Node& )
```

Η μέθοδος αυτή υπολογίζει το σημείο 'διασταύρωσης' για τα μοντέλα εξασθένισης 'Ελεύθερου Χώρου' και 'Ανάκλασης Δύο Δρόμων'. Το σημείο αυτό αντιστοιχεί στην απόσταση εκείνη η οποία δίνει ίδια τιμή εξασθένισης και με τα δύο μοντέλα. Η απόσταση αυτή υπολογίζεται εξισώνοντας τη σχέση υπολογισμού εξασθένισης για το μοντέλο 'Ελεύθερου Χώρου' με την προσεγγιστική σχέση του μοντέλου 'Ανάκλασης Δύο Δρόμων'. Η εξίσωση που προκύπτει είναι:

dx = 4 * PI * hr_m_ * ht_m_ / wl_m (5.14)

όπου:

dx: το σημείο διασταύρωσης σε μέτρα (απόσταση)
PI: ο αριθμός 'π' σε δεκαδική μορφή
hr_m: το ύψος της κεραίας του δέκτη σε μέτρα
ht_m: το ύψος της κεραίας του πομπού σε μέτρα
wl_m: το μήκος κύματος σε μέτρα

Αν η απόσταση ανάμεσα σε δύο κόμβους είναι πέρα από αυτό το σημείο, τότε το μοντέλο 'Ελεύθερου Χώρου' δίνει μεγαλύτερη εξασθένιση. Στην αντίθετη περίπτωση, δηλαδή κάτω από το σημείο αυτό, μεγαλύτερη εξασθένηση δίνει το μοντέλο 'Ανάκλασης Δύο Δρόμων'.

■ 5.3.2.a.7 Επιλογή Μοντέλων Εξασθένισης και Υπολογισμός

Δεδομένου ότι επιθυμούμε να υπάρχει η δυνατότητα χρήσης πολλών διαφορετικών μοντέλων για τον υπολογισμό της ίδιας ποσότητας, είτε πρόκειται για την εξασθένισης διαδρομής είτε για την εξασθένιση λόγω βλάστησης, πρέπει δημιουργήσουμε τις κατάλληλες μεθόδους που θα επιτρέπουν την επιλογή με κριτήρια.

```
virtual double path_loss_dBm( const Node&, const Node& )
```

Η μέθοδος αυτή επιλέγει ποια μέθοδος υπολογισμού εξασθένισης διαδρομής θα χρησιμοποιηθεί από το μοντέλο επικοινωνίας. Είναι δηλαδή το σημείο στο οποίο ορίζουμε ποιο μοντέλο εξασθένισης διαδρομής θα αξιοποιήσουμε, από όσα έχουν υλοποιηθεί. Ο ορισμός της μεθόδους είναι:

```
path_loss_dBm( const Node& u, const Node& v )
{
  double PL; // signal attenutation due to path loss ropagation.
             // foliage is not accounted for here
  switch( pl_model_ )
  {
  case 0:
     PL = free_space_PL_dB( u, v );
  break;

  case 1:
     PL = log_distance_PL_dB( u, v );
  break;

  case 2:
     PL = Egli_median_terrain_model_PL_dB( u, v );
  break;

  case 3:
     PL = ground_bounce_multipath_Approximation_PL_dB( u, v );
  break;

  case 4:
     PL = ground_bounce_multipath_Analysis_PL_dB( u, v );
  break;

  default:
     PL = free_space_PL_dB( u, v );
  break;
  }
  return PL;
}
```

Η μέθοδος είναι εξαιρετικά απλή στην υλοποίηση, χωρίς σύνθετες δομές αποφάσεων. Μια ακέραιη παράμετρος 'pl_model_' ελέγχεται με μια δήλωση switch και αναλόγως της τιμής της εκτελείται η μέθοδος του επιλεγμένου μοντέλου για δύο κόμβους 'u', 'v', και επιστρέφεται το αποτέλεσμα στην προσομοίωση.

```
virtual double foliage_loss_dBm( const Node&, const Node& )
```

Αντίστοιχα με την προηγούμενη μέθοδο, σε αυτή τη περίπτωση επιλέγεται ποια μέθοδος υπολογισμού εξασθένισης βλάστησης θα χρησιμοποιηθεί από το μοντέλο επικοινωνίας. Ο ορισμός της μεθόδους είναι:

```
foliage_loss_dBm( const Node& u, const Node& v )
{
  double FL; // signal attenutation due to foliage.

 switch( f_model_ )
 {
 case 0:
     FL = early_ITU_foliage_PL_dB( u, v );
 break;

 case 1:
     FL = weissberger_foliage_PL_dB( u, v );
 break;

 case 2:
     FL = COST235_in_leaf_foliage_PL_dB( u, v );
 break;

 case 3:
     FL = COST235_out_of_leaf_foliage_PL_dB( u, v );
 break;

 default:
     FL = early_ITU_foliage_PL_dB( u, v );
 break;
 }

  return FL;
}
```

Η ακέραιη παράμετρος 'fl_model_' ελέγχεται με μια δήλωση switch και αναλόγως της τιμής της εκτελείται η μέθοδος του επιλεγμένου μοντέλου για δύο κόμβους 'u', 'v', και επιστρέφεται το αποτέλεσμα στην προσομοίωση.

■ 5.3.2.a.8 Υπολογισμός Μέσης Λαμβανόμενης Ισχύος Σήματος Στο Δέκτη

```
virtual double received_power_dBm( const Node&, const Node& )
```

Η μέθοδος αυτή υπολογίζει την μέση τιμή της λαμβανόμενης ισχύος σήματος στο δέκτη σε dBm. Αυτή είναι ίση με την τιμή της εκπεμπόμενης ισχύος σε dBm μείον την συνολική μέση εξασθένιση σε dBm. Αναφορικά με την μέση εξασθένηση εξετάζονται δύο περιπτώσεις αναλόγως την τιμή μιας bool μεταβλητής 'foliage_'. Αν η τιμή είναι αληθής τότε υπολογίζουμε την μέση εξασθένηση ως άθροισμα της μέσης εξασθένησης διαδρομής και της μέσης εξασθένησης βλάστησης. Αν είναι ψευδής, τότε εφαρμόζουμε μόνο εξασθένηση διαδρομής. Σε κάθε περίπτωση επιστρέφεται η μέση λαμβανόμενη ισχύς σε dBm. Ο ορισμός της μεθόδου είναι:

```
received_power_dBm( const Node& u, const Node& v ) const throw()
{
      double Pr_dBm;
     if(!foliage_)
       Pr_dBm = transmit_power_dBm() - path_loss_dBm( u ,v );
      else if(foliage_)
       Pr_dBm = transmit_power_dBm() - path_loss_dBm( u ,v )
              - foliage_loss_dBm( u, v );
      return Pr_dBm;
}
```

■ 5.3.2.a.9 Υπολογισμός Πιθανότητας Συνδεσιμότητας

```
virtual double log_normal_P_connectivity_probability
                                (const Node&, const Node& )
```

Η μέθοδος αυτή εκφράζει το φαινόμενο της διασποράς μεγάλης κλίμακας [§4.2] και υπολογίζει την πιθανότητα η λαμβανόμενη ισχύς να είναι μεγαλύτερη της ευαισθησίας του δέκτη για μια δεδομένη τυπική απόκλιση. Η πιθανότητα αυτή είναι η συμπληρωματική της συνάρτησης κατανομής της κανονικής κατανομής, την οποία ακολουθούν οι τιμές ισχύος γύρω από την μέση τιμή. Είναι δηλαδή η F(-x) = 1 - F(x). Η μέθοδος ακολουθεί τα εξής βήματα:

(1) Υπολογίζεται η μέση τιμή mean της ισχύος στο δέκτη με τη μέθοδο '*received_power_dBm(const Node&, const Node&)*'

(2) Για την μέση τιμή mean και την τυπική απόκλιση εισόδου σε dBm std_dev_dBm, υπολογίζεται η τιμή τη ευαισθησίας του δέκτη κατά την τυποποιημένη κανονική κατανομή. Αυτή ορίζεται για μέση τιμή 0 και τυπική απόκλιση 1 και δίνεται από την σχέση:

z_var = (RS_dBm_ - mean) / abs(std_dev_dBm_) (5.15)

όπου:

z_var: η τιμή της ευαισθησίας του δέκτη κατά την τυποποιημένη κανονική κατανομή
RS_dBm_: η τιμή της ευαισθησίας του δέκτη κατά την κανονική κατανομή με μέση τιμή mean και τυπική απόκλιση std_dev_dBm
mean: η μέση τιμή ισχύος
std_dev_dBm: η τυπική απόκλιση εισόδου της κατανομής της ισχύος

Φυσικά γίνεται έλεγχος για την τιμή της τυπικής απόκλισης εισόδου. Αν είναι 0, τότε προφανώς δεν έχουμε κατανομή αλλά μόνο μέση τιμή που είναι και η ακριβής τιμή της ισχύος, οπότε η μέθοδος δεν έχει κάποιο ρόλο να διαδραματίσει. Σε αυτή την περίπτωση επιστρέφει τιμή '-1', η οποία αξιοποιείται από τη μέθοδο που την κάλεσε ώστε να ακολουθήσει άλλη τακτική. Σε κάθε άλλη περίπτωση το πρόσημο της τυπικής απόκλισης εξασφαλίζεται θετικό με την abs() λειτουργία της C++.

(3) Η τιμή της μεταβλητής z_var τυποποιημένης κανονικής κατανομής μπορεί να προκύψει είτε θετική είτε αρνητική, αναλόγως αν η ευαισθησία του δέκτη είναι μικρότερη από την ισχύ εκπομπής ή μεγαλύτερη αντίστοιχα.

Για την συνάρτηση κατανομής ισχύει F(-z) = 1 - F(z). Δημιουργούμε συνεπώς δύο βοηθητικές μεθόδους τις F_fn και Q_fn που υλοποιούν τις F(z) και F(-z) αντίστοιχα:

```
virtual double F_fn( double z )
virtual double Q_fn( double z )
```

Η F_fn υλοποιεί την συνάρτηση κατανομής της μεταβλητής z δηλαδή την πιθανότητα P(z < Z), και η Q_fn υλοποιεί την συμπληρωματική της, δηλαδή την πιθανότητα P(z > Z). Επομένως αν η z_var προκύψει θετική χρησιμοποιούμε την Q_fn, και αν προκύψει αρνητική την F_fn. Δημιουργούμε δύο μεθόδους αντί για μία για λόγους τάξης και δεδομένου ότι έτσι συναντούνται σε μεγάλη έκταση της βιβλιογραφίας και σε προγραμματιστικές τεχνικές.

Η συνάρτηση κατανομής της κανονικής κατανομής υλοποιείται στην C++ με την βοήθεια της συνάρτησης σφάλματος erf(). Η σχέση που εκφράζει την F(z) = F_fn με την βοήθεια της erf() είναι:

F_fn = 0.5 * (1 + (double)erf(z / (double)sqrt(2))) (5.16)

και για την συμπληρωματική F(-z) = Q_fn:

Q_fn = 0.5 * (1 - (double)erf(z / (double)sqrt(2))) (5.17)

Εν τέλει επιστρέφεται η πιθανότητα στην προσομοίωση σε δεκαδική μορφή.

■ 5.3.2.a.10 Παραγωγή τυχαίας διασποράς

```
virtual double normal_random_variance_generator()
```

Οι μέθοδοι υπολογισμού εξασθένισης που εφαρμόζονται στο μοντέλο υπολογίζουν μια σταθερή μέση τιμή για κάθε ζεύξη κόμβων, εφόσον τα στοιχεία υπολογισμού είναι σταθερά. Η παρούσα μέθοδος εισάγει διασπορά σε αυτή τη μέση τιμή δημιουργώντας εικονικά τυχαία δείγματα μετρήσεων.

Η εξασθένιση και επακόλουθα η ισχύς σήματος ακολουθούν την κανονική κατανομή. Σκοπός της μεθόδου είναι να παράγουμε τυχαίες τιμές με σεβασμό στη κατανομή. Πρέπει δηλαδή για θεωρητικά άπειρο αριθμό τυχαίων τιμών να εφαρμόζεται σε αυτές η κανονική κατανομή. Η μέθοδος που εφαρμόζεται για τον σκοπό αυτό είναι ο βασικός μετασχηματισμός "Box-Muller" των George Edward Pelham Box και Mervin Edgar Muller [23]. Ο μετασχηματισμός αυτός δημιουργεί τυχαία δειγμάτα τυποποιημένης κανονικής κατανομής (μέση τιμή = 0, τυπική απόκλιση = 1), με δεδομένο δείγματα τιμών που ακολουθούν την ομοιόμορφη κατανομή.

Στο Shawn ο μετασχηματισμός είναι υλοποιημένος στον πηγαίο κώδικα ως εξής:

```
NormalRandomVariable::
   operator double( void )
      const throw()
   {
      assert( initialized_ );
      return
         mean_ +
         std_dev_
         * sqrt( -2.0 * log(uniform_random_0e_1i()) )
         * cos( 2.0 * M_PI * uniform_random_0e_1i());
   }
```

[shawn/sys/misc/random/normal_random_variable.cpp]

Όπως παρατηρούμε η υλοποίηση του shawn προσαρμόζει την μέθοδο στην εκάστοτε κατανομή μέσης τιμής mean_ και τυπικής απόκλισης std_dev_. Προκειμένου να έχουμε καλύτερο έλεγχο ορίζουμε εκ νέου τη μέθοδο χωρίς την προσαρμογή στην κατανομή, προκειμένου να την κάνουμε χειροκίνητα:

```
double
RFNTModel::
normal_random_variance_generator() const throw()
{
     return  sqrt( -2.0 * log(uniform_random_0e_1i()) ) *
             cos( 2.0 * PI * uniform_random_0e_1i());
}
```

[shawn/sys/comm_models/rf_nt_model.cpp]

Η μέθοδος *uniform_random_0e_1i()* επιστρέφει μια τιμή μεταξύ 0 και 1 (0,1], ώστε η τιμή του λογαρίθμου να είναι αρνητική ή μηδέν και να μην προκύπτει αρνητική ποσότητα στην ρίζα. Χρησιμοποιεί την *rand()* λειτουργία της C++:

```
double uniform_random_0e_1i( void ) throw()
{ return (double(rand())+1.0) / (double(RAND_MAX)+1.0); }
```

[shawn/sys/misc/random/basic_random.cpp]

όπου RAND_MAX είναι η μέγιστη τιμή που μπορεί να αποδώσει η *rand()* και ορίζεται στην βιβλιοθήκη <cstdlib>.

Η random_variance_generator() επιστρέφει συνεπώς τυχαίες τιμές διασποράς, οι οποίες προστίθενται στην μέση τιμή για να δώσουν εικονικά τυχαία δείγματα. Αποκαλούμε τα δείγματα εικονικά διότι δεν είναι προϊόντα μετρήσεων αλλά υπολογισμού.

Η διαδικασία θα μπορούσε να υλοποιηθεί και με την μέθοδο 'αντίστροφου μετασχηματισμού δειγμάτων' που βασίζεται στην αντίστροφη συνάρτηση κατανομής $F^{-1}(y)$, όπου y είναι μια μεταβλητή ομοιόμορφης κατανομής στο διάστημα [0, 1]. Είναι δηλαδή:

$$Y = Fx(X)$$
$$X = Fx^{-1}(Y)$$

- 5.3.2.a.11 Εξέταση της Επικοινωνίας ανάμεσα σε δύο κόμβους

```
virtual bool can_communicate_uni( const Node&, const Node& )
```

Τελευταίο στάδιο στην δόμηση του μοντέλου 'RF NT' δεν μπορούσε να είναι άλλο από την τροποποίηση της τελευταίας μεθόδου στην αλυσίδα της λειτουργικής του εφαρμογής, δηλαδή της *can_communicate_uni(const Node&, const Node&)*. Η τροποποίηση της μεθόδου χρησιμοποιεί ως πρότυπο τον ορισμό της στο μοντέλο 'Unit Disk Graph NT' [§5.2] προκειμένου να παρέχει υποδομή για εξέταση της επικοινωνίας βάσει του τύπου του κόμβου. Από αυτή την προσέγγιση τα δύο μοντέλα έχουν ακριβώς την ίδια συμπεριφορά.

Ωστόσο εδώ σταματούν και οι ομοιότητες για τα δύο μοντέλα. Το 'Unit Disk Graph NT' μοντέλο επικοινωνίας, στα πρότυπα του απλού UDG, δίνει απάντηση στο ερώτημα της επικοινωνίας δύο κόμβων 'u', 'v', συγκρίνοντας την ευκλείδεια απόσταση μεταξύ τους με το εύρος επικοινωνίας της προσομοίωσης, σύμφωνα με την εξής απλή σχέση:

```
return
euclidean_distance( u.real_position(), v.real_position() )
<= range_;
```

[shawn/sys/comm_models/disk_graph_nt_model.cpp]

Στο 'RF NT' μοντέλο η σχέση αυτή αντικαθίσταται σε πρώτο στάδιο από δύο βοηθητικές μεθόδους της *can_communicate_uni(const Node&, const Node&):*

- can_communicate_mean_range(const Node&, const Node&)
- can_communicate_RNG(const Node&, const Node&).

```
virtual
bool can_communicate_mean_range( const Node&, const Node& )
```

Η μέθοδος αυτή εξετάζει την επικοινωνία με μια πιθανότητα συνδεσιμότητας. Η πιθανότητα αυτή εκφράζει την απαιτούμενη ελάχιστη τιμή της συνάρτησης κατανομής της ισχύος σήματος που φτάνει στο δέκτη και ορίζεται με την παράμετρο εισόδου P_RS. Μπορούμε δε να πούμε ότι ορίζει ένα επίπεδο εμπιστοσύνης για τη συνδεσιμότητα των δύο κόμβων.

Αν η πιθανότητα P_RS είναι μεταξύ 0 και 1 { (0, 1) ή (0%, 100%)} τότε υπολογίζουμε για την ευαισθησία του δέκτη την πιθανότητα συνδεσιμότητας CP και συγκρίνουμε τις δύο τιμές. Η τελευταία υπολογίζεται με τη μέθοδο *log_normal_P_connectivity_probability(const Node&, const Node&)* και εφόσον είναι μεγαλύτερη ή ίση της απαίτησης, τότε οι κόμβοι επικοινωνούν. Αν όμως η CP είναι μικρότερη του μηδενός, αυτό σημαίνει ότι έχουμε ορίσει τυπική απόκλιση μηδέν, και συνεπώς δεν ορίζεται διασπορά. Σε αυτή τη περίπτωση συγκρίνεται η μέση τιμής της ισχύος με την ευαισθησία στο δέκτη και αν είναι μεγαλύτερη επιστρέφει true.

```
if(P_RS_>0 && P_RS_<1
{
    double CP;
    CP = log_normal_P_connectivity_probability( u, v );
    if(CP < 0)
      return received_power_dBm( u, v ) >= RS_dBm_;
    else if (CP >=0 )
      return CP >=P_RS_;
}
```

[shawn/sys/comm_models/rf_nt_model.cpp]

Αν η πιθανότητα είναι 100% ή αρνητική ή μηδέν, τότε η επικοινωνία εξετάζεται συγκρίνοντας την μέση τιμή της λαμβανόμενης ισχύος στο δέκτη με την ευαισθησία αυτού. Αν η ισχύς είναι μεγαλύτερη επιστρέφει θετικό:

```
else if (P_RS_<=0 || P_RS_>=1)
  return received_power_dBm( u, v ) >= RS_dBm_;
```

[shawn/sys/comm_models/rf_nt_model.cpp]

```
can_communicate_RNG(const Node& u, const Node& v) const throw()
```

Η μέθοδος αυτή δημιουργεί εικονικά τυχαία δείγματα ισχύος με την μέθοδο normal_random_variance_generator() και τα συγκρίνει με την ευαισθησία του δέκτη. Επειδή η normal_random_variance_generator() δίνει διασπορά, πολλαπλασιάζουμε με την τυπική απόκλιση και προσθέτουμε στη μέση ισχύ για να προκύψει το τυχαίο δείγμα:

```
return
normal_random_variance_generator() * std_dev_dBm_ +
received_power_dBm( u,v ) >= RS_dBm_;
```

[shawn/sys/comm_models/rf_nt_model.cpp]

Με ένα if statement η *can_communicate_uni(const Node&, const Node&)* επιλέγει ποια από τις δύο μεθόδους θα αξιοποιήσει για να εξετάσει την επικοινωνία. Το κριτήριο είναι η παράμετρος εισόδου bool RNG. Αν η παράμετρος είναι true τότε εκτελείται η *can_communicate_RNG(const Node&, const Node&)*, αλλιώς η *can_communicate_mean_range(const Node&, const Node&)*:

```
if(RNG_==false)
  return can_communicate_mean_range( u, v );
else if(RNG_==true)
  return can_communicate_RNG( u, v );
```

[shawn/sys/comm_models/rf_nt_model.cpp]

Πάντα προηγείται ο έλεγχος του τύπου των κόμβων πριν εκτιμηθεί η επικοινωνία σε RF επίπεδο.

Εν τέλει η *can_communicate_uni(const Node&, const Node&)* επιστρέφει στην προσομοίωση τη τιμή που επιστρέφεται σε αυτή από τις δύο βοηθητικές μεθόδους.

Η μέθοδος *can_communicate_mean_range(const Node&, const Node&)* ενδείκνυται για χρήση σε εργασία (common task) προκειμένου να εξετάσουμε την συνδεσιμότητα σε ένα δίκτυο χωρίς να τρέξουμε προσομοίωση αποστολής/παραλαβής μηνυμάτων. Είναι η ιδανική μέθοδος για την εργασία 'connectivity'.

Αντίθετα η μέθοδος *can_communicate_mean_range(const Node&, const Node&)* είναι ιδανική για εκτέλεση προσομοίωσης και συλλογή στατιστικών μέσω αποστολής/παραλαβής μηνυμάτων. Σε αυτή τη περίπτωση θέλουμε να προσομοιώσουμε την τυχαιότητα στην ισχύ σήματος σε κάθε εκπομπή, τηρώντας πάντα τη συνθήκη της κανονικής κατανομής. Η μέθοδος αυτή εξυπηρετεί άριστα τον σκοπό αυτό.

5.3.2.b Μοντέλο RF NT: Παράμετροι τάξης 'RFNTModel'

Στην προηγούμενη ενότητα παρουσιάστηκαν οι μέθοδοι του μοντέλου, καθώς και οι παράμετροι που χρησιμοποιεί η κάθε μία κατά περίπτωση. Σε αυτή τη μικρή ενότητα παρουσιάζονται συγκεντρωτικά οι παράμετροι αυτοί και εξηγείται η χρήση της κάθε μιας.

bool use_RF

Παράμετρος που ουσιαστικά ενεργοποιεί το μοντέλο. Η τιμή 'false' το υποβαθμίζει σε απλό 'UDG'

bool RNG

Η παράμετρος αυτή καθορίζει σε πρώτο βαθμό την συμπεριφορά την μεθόδου *can_communicate_uni(const Node&, const Node&)* σε επίπεδο RF. Για τιμή true εξετάζεται η επικοινωνία με την *can_communicate_RNG(const Node&, const Node&*), ενώ για τιμή false με την *can_communicate_mean_range(const Node&, const Node&)*.

int pl_model

Ακέραια παράμετρος που ορίζει το μοντέλο εξασθένισης διαδρομής που θα χρησιμοποιηθεί για τον υπολογισμό της ισχύος σήματος στο δέκτη. Πιθανές τιμές είναι:

0 : Free Space Propagation Model - Μοντέλο Ελεύθερου Χώρου
1 : Log-Distance Model - Μοντέλο Λογαριθμικής Απόστασης (d0=1m)
2 : Log-Distance Model - Μοντέλο Λογαριθμικής Απόστασης (d0=variable)
3 : Egli Terrain Model - Μοντέλο Αναγλύφου του Egli
4 : Multipath Simple Model - Μοντέλο Ανακλάσεων Δύο Δρόμων (προσεγγιστική σχέση)
5 : Multipath Analytic Model - Μοντέλο Ανακλάσεων Δύο Δρόμων (αναλυτική σχέση)

Για κάθε άλλη τιμή χρησιμοποιείται το μοντέλο Ελεύθερου Χώρου (FSP).

double f_GHz

Η συχνότητα σε GHz.

double Pt

Η ισχύς εκπομπής. Μπορεί να δίνεται είτε σε milliWatts είτε σε Watts είτε σε dBm.

std::string Pt_unit

Η παράμετρος αυτή ορίζει τη μονάδα της ισχύος εκπομπής εισόδου, δηλαδή της Pt. Αποδεκτές τιμές είναι:

"mW": milliWatts
"W": Watts
"dBm": decibels

double Receiver_Sensitivity_dBm

Η ευαισθησία στο δέκτη σε dBm.

double Gt_dB

Το κέρδος της κεραίας του πομπού σε dB.

double Gr_dB

Το κέρδος της κεραίας του δέκτη σε dB.

bool foliage

Παράμετρος που ορίζει αν θα υπολογίσουμε εξασθένιση λόγω βλάστησης στον υπολογισμό της συνολικής εξασθένισης ή όχι.

int f_model

Εφόσον εξετάζουμε εξασθένιση βλάστησης η παράμετρος αυτή ορίζει το μοντέλο που θα χρησιμοποιηθεί. Οι πιθανές επιλογές είναι:

0 : Early ITU Recomendation - Πρώιμο Μοντέλο ITU-R
1 : Weissberger Model - Μοντέλο του Weissberger
2 : COST235 'in leaf' vegetation - Μοντέλο COST235 σε βλάστηση με φύλλωμα
3 : COST235 'out of leaf' vegetation - Μοντέλο COST235 σε βλάστηση χωρίς φύλλωμα

double foliage_path_factor

Παράγοντας που ορίζει το ποσοστό της απόστασης στη διαδρομή επικοινωνίας για το οποίο θεωρούμε ότι καλύπτεται από βλάστηση. Πρέπει να έχει τιμή μεταξύ 0 και 1 { [0,1] }.

double hr_m

Το ύψος της κεραίας του δέκτη σε μέτρα.

double ht_m

Το ύψος της κεραίας του πομπού σε μέτρα

double n

Ο εκθέτης του μοντέλου εξασθένισης διαδρομής 'Λογαριθμικής Απόστασης'. Παίρνει τιμές αναλόγως του περιβάλλοντος [Πίνακας 4.1]

double std_dev_dBm

Η τυπική απόκλιση της μέσης τιμής της ισχύος σε dBm.

double P_RS

Η κατώτερη τιμή της πιθανότητας συνδεσιμότητας για την οποία γίνεται αποδεκτή η επικοινωνία δύο κόμβων. Πρέπει να παίρνει τιμές μεταξύ 0 και 1 (0,1). Για κάθε άλλη τιμή εξετάζεται η επικοινωνία σε επίπεδο μέσης τιμής της ισχύος.

double P_CI

Παράμετρος πιθανότητας που ορίζει το επίπεδο εμπιστοσύνης γύρω από την μέση τιμής της ισχύος για πραγματικά δείγματα / πραγματικές μετρήσεις. Η πιθανότητα μπορεί να αξιοποιηθεί σε κάποια μέθοδο που εξετάζει της αποδοτικότητα του μοντέλου βάσει πραγματικών μετρήσεων ισχύος στο πεδίο.

5.3.3 Μοντέλο RF NT: Παραγωγή Μοντέλου

Στο Shawn όλα τα μοντέλα και όλες οι οντότητες τις προσομοίωσης είναι ενσαρκώσεις C++ τάξεων που γεννιούνται με την οδηγία 'new'. Ομοίως και το 'RF NT' μοντέλο επικοινωνίας. Η γέννεση των αντικειμένων γίνεται από τις αντίστοιχες τάξεις "Παραγωγούς" (factories) που ορίζονται για αυτά. Κάθε μοντέλο επικοινωνίας δηλαδή κατά την προσομοίωση συγκροτείται και παραμετροποιείται από την "Παραγωγική" του τάξη. Αυτή είναι δυνατό ακόμα και να καλέσει μεθόδους του μοντέλου αν το επιθυμήσουμε. Κύρια λειτουργία της ωστόσο είναι να εξετάζει το περιβάλλον της προσομοίωσης και να συλλέγει τις τιμές των παραμέτρων που θέλουμε να περάσουμε στο μοντέλο.

Για το 'Disk Graph NT' μοντέλο η "Παραγωγός" τάξη δημουργήθηκε με πρότυπο την αντίστοιχη του UDG μοντέλου χωρίς να γίνει προσθήκη επιπλέον μεθόδων ή παραμέτρων. Ο τύπος των κόμβων είναι παράμετρος της τάξης των κόμβων και όχι του μοντέλου επικοινωνίας. Στην περίπτωση του 'RF NT' μοντέλου επικοινωνίας, όπως είδαμε παραπάνω έχουμε σωρεία επιπλέον παραμέτρων, οπότε πρέπει να τροποποιήσουμε κατάλληλα την "Παραγωγική" του τάξη. Αυτή ονομάζεται "RFNTCommunicationModelFactory".

Οι παράμετροι που διαχειρίζεται και αποδίδει στο μοντέλο η Παραγωγός τάξη είναι όλες οι παράμετροι που αναφέρθηκαν στο προηγούμενο κεφάλαιο. Ονομαστικά μαζί με τον τύπο τους και μικρή περιγραφή στα αγγλικά είναι:

```
bool use_RF; // bool to use or not RF analysis. NO results in plain UDG model
bool RNG; // random signal power generator (normal distribution around mean)
int pl_model; // main path loss propagation model to use
double f_GHz; // frequency of the carrier in GHz
double Pt; // transmitter power
std::string Pt_unit; // transmitter power unit
double Receiver_Sensitivity_dBm; // receiver sensitivity in dBm
double Gt_dB; // transmitter antenna Gain in dB
double Gr_dB; // receiver antenna Gain in dB
bool foliage; // bool to consider vegetation attenuation
int f_model; // foliage model to use. adds loss to main path loss model
double foliage_path_factor; // factor to specify the foliage path compared to total path (m)
double hr_m; // universal receiver antenna height (m)
double ht_m; // universal transmitter height
double n; // log-distance model exponent
double std_dev_dBm; // standar deviation of the received signal strength
double P_RS; // required conectivity probability at the given receiver sensitivity / connectivity
             // threshold
double P_CI; // confidence interval threshold -> to compute confidence space around mean
             // value and see if samples are within
int verbosiry; // verbosity level
```

[shawn/sys/comm_models/rf_communication_model_nt_factory.cpp]

Η τάξη υλοποιείται ώστε να ελέγχει για τιμές παραμέτρων σε δύο διαφορετικά μέρη:

(a) Στην δήλωση του μοντέλου, είτε στη γραμμή εντολών είτε μέσω αρχείου ρυθμίσεων.

(b) Σε αρχείο XML και συγκεκριμένα σε ετικέτα περιβάλλοντος (<environment/> tag) το οποίο πρέπει να έχει φορτωθεί μαζί με την εργασία 'prepeare world' που ετοιμάζει το περιβάλλον προσομοίωσης και συγκροτεί τα μοντέλα του.

5.3.3.a Παράμετροι Ρυθμίσεων

Το ενεργό μοντέλο επικοινωνίας δηλώνεται με την εκτέλεση της εργασίας 'prepare world'. Αυτή εκτελείται στην αρχή της προσομοίωσης ώστε να ετοιμαστεί το περιβάλλον της και να συγκροτηθούν τα μοντέλα επικοινωνίας / ακμών και εκπομπής. Η δήλωση του μοντέλου επικοινωνίας ακολουθεί την δήλωση του μοντέλου ακμών και την συνοδεύουν οι παράμετροί του. Οι παράμετροι αυτοί προκειμένου να φορτωθούν στο μοντέλο, πρέπει να έχουν οριστεί στην "Παραγωγική" του τάξη και να αποδοθεί σωστά η τιμή τους στην δήλωση του μοντέλου με την 'prepare world' εργασία. Ο ορισμός τους γίνεται με απευθείας οδηγία στο περιβάλλον προσομοίωσης με κάποια από τις παρακάτω μεθόδους:

```
const std::string& required_string_param( const std::string& )
const throw( std::runtime_error );

const std::string& optional_string_param( const std::string&,
const std::string&, bool* is_set=NULL )
const throw( std::runtime_error );

int required_int_param( const std::string& )
const throw( std::runtime_error );

int optional_int_param( const std::string&, int, bool*
is_set=NULL )
const throw( std::runtime_error );

double required_double_param( const std::string& )
const throw( std::runtime_error );

double optional_double_param( const std::string&, double, bool*
is_set=NULL )
const throw( std::runtime_error );

bool required_bool_param( const std::string& )
const throw( std::runtime_error );

bool optional_bool_param( const std::string&, bool, bool*
is_set=NULL )
const throw( std::runtime_error );
```

[shawn/sys/simulation/simulation_environment.h]

Με μια αναφορά στο περιβάλλον προσομοίωσης (SimulationEnvironment&) μέσω του δείκτη που παρέχει σε αυτό ο ελεγκτής προσομοίωσης, αποκτούμε πρόσβαση στις παραπάνω μεθόδους που ορίζουν παραμέτρους εισαγωγής. Όπως φαίνεται και από το όνομα των μεθόδων αυτές είναι δύο ειδών για όλους τους τύπους δεδομένων:

I) Υποχρεωτικής εισόδου
II) Προαιρετικής εισόδου

Στην πρώτη περίπτωση η παράμετρος πρέπει να πάρει οπωσδήποτε τιμή κατά την δήλωση του μοντέλου, αλλιώς θα οδηγηθούμε σε σφάλμα κατά την εκτέλεση. Στην δεύτερη περίπτωση η παράμετρος είναι προαιρετική δεδομένου ότι κατά την δήλωσή της δίνουμε και μια προεπιλεγμένη τιμή που θα ισχύσει σε περίπτωση που δεν εισάγουμε εμείς άλλη τιμή.

Οι δηλώσεις για όλες τις παραμέτρους του μοντέλου είναι:

```
const SimulationEnvironment& se = sc.environment();

bool config_over_xml = se.optional_bool_param("config_over_xml", true);
int verbosity = se.optional_int_param("verbosity", 0);

use_RF = se.optional_bool_param("use_RF", true);
RNG = se.optional_bool_param("RNG", false);
pl_model = se.optional_int_param("path_loss_model", 0);
f_GHz = se.required_double_param("f_GHz");
Pt = se.required_double_param("Pt");
Pt_unit = se.required_string_param("Pt_unit");
Receiver_Sensitivity_dBm = se.required_double_param("Receiver_Sensitivity_dBm");
Gt_dB = se.optional_double_param("Gt_dB", 0);
Gr_dB = se.optional_double_param("Gr_dB", 0);
foliage = se.optional_bool_param("foliage", false);
f_model = se.optional_int_param("foliage_model", 0);
foliage_path_factor = se.optional_double_param("foliage_path_factor", 1 );
hr_m = se.optional_double_param("hr_m", 1.5 );
ht_m = se.optional_double_param("ht_m", 1.5 );
ht_m = se.optional_double_param("n", 2 );
std_dev_dBm = se.optional_double_param("std_dev_dBm", 0);
P_RS = se.optional_double_param("Connectivity_Probability_Threshold", -1);
P_CI = se.optional_double_param("Confidence_Interval_Threshhold", -1);
```

[shawn/sys/comm_models/rf_communication_model_nt_factory.cpp]

Όπως φαίνεται για αρκετές παραμέτρους υλοποιείται η προαιρετική εισαγωγή. Οι παράμετροι μπορούν να δηλωθούν είτε σε αρχείο ρυθμίσεων, από το οποίο εκτελείται η προσομοίωση, είτε στη γραμμή εντολών διαδραστικά. Προτιμάται η πρώτη λύση καθότι το μοντέλο έχει πολλές παραμέτρους και θα είναι εξαιρετικά κουραστικό για τον χρήστη να τις εισάγει χειροκίνητα. Ακολουθεί παράδειγμα δήλωσης:

prepare_world env_config=shawn-kaisariani_xyz_.xml edge_model=simple_nt comm_model=rf_nt path_loss_model=1 foliage_model=1 std_dev_dBm=5

5.3.3.b XML Παράμετροι

Με την εκτέλεση της εργασίας 'prepare world' είναι δυνατό να ενσωματώσουμε στην προσομοίωση παραμέτρους από XML αρχείο δεδομένων. Αυτό πραγματοποιείται μη την παράμετρο 'env_config' της εργασίας. Σε αυτή δηλώνεται XML αρχείο στο οποίο πρέπει να υπάρχει ετικέτα περιβάλλοντος (<environment/> tag), το οποίο εφόσον βρεθεί φορτώνεται στο περιβάλλον προσομοίωσης και είναι προσβάσιμο, μαζί με τις απλές ετικέτες που περιέχει.

Για την προσομοίωση που παρουσιάζεται στην παρούσα μελέτη δημιουργήθηκε μια ετικέτα στο ίδιο XML αρχείο που περιέχει τις συντεταγμένες και τα στοιχεία των κόμβων. Η ετικέτα περιλαμβάνει ομαδική ετικέτα (group tag) "RF" μέσα στην οποία βρίσκονται οι απλές ετικέτες των παραμέτρων, Ακολουθεί δείγμα της ετικέτας:

```
<scenario>
  <environment>
    <!-- ================================================
    ########## RF Environment and node info ##############
    ==> In case u don't want a value comment whole line
    ==> Zero values are take into account normally
    ==> No value results in runtime error !!!
    ================================================ -->
      <tag type="group" name="RF">
      <tag type="bool" name="use_RF" value="true"/>

      <!-- ========================================== -->
      <tag type="bool" name="RNG" value="true"/>
      <!-- ==========================================
      This option decides the function of the comm model:

      true: The comm model compares a random power value
      with the receiver sensitivity. This value is generated
      by the normal distribution with mean value the mean
      path loss and standard daviation the respective value.

      false: The comm model compares the received power due
      to mean path loss with the receiver sensitivity, or
      computes the probability that a random value in its
      distribution can be higher tha the receiver sensitivity

      ========================================== -->
      <tag type="double" name="Connectivity_Probability_Threshold" value="0.95"/>
      <tag type="double" name="Confidence_Interval_Threshhold" value="1"/>
      .
      .
      .
      </tag>
  </environment>
</scenario>
```

[/shawn/buildfiles/shawn-kaisariani_xyz_.xml]

Στην Παραγωγό τάξη του μοντέλου επικοινωνίας αποκτούμε πρόσβαση στην ετικέτα "RF" με τη μέθοδο find_tag(const std::string&). Εφόσον την βρούμε ορίζουμε δείκτη σε αυτή και μπορούμε πλέον να επεξεργαστούμε τα περιεχόμενά της:

```
ConstTagHandle rf_tag_group = sc.environment().find_tag("RF");
const GroupTag* rf_gt = rf_tag_group.is_not_null() ?
dynamic_cast<const GroupTag*>(rf_tag_group.get()) : NULL;
```

[shawn/sys/comm_models/rf_communication_model_nt_factory.cpp]

Με τον ίδιο τρόπο ορίζουμε δείκτες και στα περιεχόμενά της, προσέχοντας πάντα τον τύπο της ετικέτας στον ορισμό του δείκτη. Γι' αυτό επιπλέον γίνεται έλεγχος τους τύπου της ετικέτας (dynamic_cast) προκειμένου να εξασφαλίσουμε ότι έχουμε αυτόν που αναμένουμε:

```
ConstTagHandle use_RF_th = rf_gt->find_tag("use_RF");
const BoolTag* use_RF_bt = use_RF_th.is_not_null() ?
dynamic_cast<const BoolTag*> (use_RF_th.get()) : NULL;
```

[shawn/sys/comm_models/rf_communication_model_nt_factory.cpp]

Έχοντας πρόσβαση στις ετικέτες μπορούμε πλέον να διαβάσουμε τις τιμές τους και να τις αποδώσουμε σε μεταβλητές:

bool use_RF_value = use_RF_bt->value();

Από όλες τις παραμέτρους ρυθμίσεων που αναφέρθηκαν, είναι δύο που δεν υλοποιούνται στην XML πρακτική:

bool config_over_xml = se.optional_bool_param("config_over_xml", true);

int verbosity = se.optional_int_param("verbosity", 0);

Η πρώτη εξετάζει την ιεραρχία των δύο μεθόδων και η δεύτερη το επίπεδο αναφοράς του μοντέλου.

5.3.3.1 Υλοποίηση ιεραρχίας και επιλογής

Εφόσον έχουμε πρόσβαση στις ίδιες παραμέτρους με δύο διαφορετικούς τρόπους, τίθεται πλέον το θέμα της ιεραρχίας αυτών. Σκοπός είναι να έχει ο χρήστης την δυνατότητα να ορίζει παραμέτρους και με τις δύο μεθόδος που αναλύθηκαν και να ορίζει ποιες θα έχουν προτεραιότητα έναντι των άλλων στην περίπτωση επικάλυψης. Επίσης πρέπει να ορίσουμε μια προεπιλογή στην περίπτωση που δεν ορίζεται μια παράμετρος σε καμία από τις δύο περιπτώσεις.

Οι παράμετροι είναι δύο ειδών όπως είδαμε στην περίπτωση (α), προαιρετικές και υποχρεωτικές. Επειδή η υποδομή είναι έτοιμη με τις παραμέτρους ρυθμίσεων του περιβάλλοντος προσομοίωσης επιλέγουμε αυτή τη τακτική ως προεπιλογή στην περίπτωση που μια παράμετρος δεν δηλώνεται με κάθε τρόπο.

Αφού τακτοποιήθηκε το θέμα της προεπιλογής μένει να οριστεί η προτεραιότητα των δύο μεθόδων. Σε αυτή την προσέγγιση δεν υπάρχει λόγος να έχουμε ρητή σειρά προτεραιότητας και συνεπώς δημιουργούμε επιλογή. Ορίζεται προαιρετική παράμετρος ρυθμίσεων bool "config_over_xml" με προεπιλογή 'true':

```
const SimulationEnvironment& se = sc.environment();
bool config_over_xml = se.optional_bool_param("config_over_xml",
true);
```

[shawn/sys/comm_models/rf_communication_model_nt_factory.cpp]

Εφόσον δεν ορίσουμε διαφορετικά οι τιμές ρυθμίσεων θα επικρατούν πάντα αυτών της ετικέτας. Για κάποια παράμετρο όμως που δεν δίνουμε τιμή με ρύθμιση αλλά δίνεται στην ετικέτα, θα ορίζεται από αυτή. Όλη η διαδικασία ορισμού παρουσιάζεται στο παρακάτω παράδειγμα κώδικα:

```
ConstTagHandle rf_tag_group = sc.environment().find_tag("RF"); // find group tag "RF"
const GroupTag* rf_gt = rf_tag_group.is_not_null() ?
                        dynamic_cast<const GroupTag*> (rf_tag_group.get()) : NULL;
// cast to the expected type and assign a pointer to it if the tag exists, else assign to pointer NULL

if( rf_gt != NULL ) // if the group tag "RF" exists -> so the pointer ain't NULL
{
    std::cout << "group tag \"RF\" found"<< std::endl;

    ConstTagHandle use_RF_th = rf_gt->find_tag("use_RF"); // find tag RF
    const BoolTag* use_RF_bt = use_RF_th.is_not_null() ?
                               dynamic_cast<const BoolTag*> (use_RF_th.get()) : NULL;
// cast to expected type and add a pointer to it if the tag exists, else assign to pointer NULL

    if( use_RF_bt != NULL ) // if we found the tag and got a pointer to it
    {
       bool use_RF_XML = use_RF_bt->value(); // get the value and add to local parameter
       XML_RF_map[ "use_RF" ] = true;
       XML_RF_bool_map[ use_RF_bt->name() ] = use_RF_bt->value();
       //CHECK PRIORITY OF VALUES BETWEEN CONFIG AND XML FILES
       if(config_over_xml) // if true then config rules over XML
            use_RF = se.optional_bool_param("use_RF", use_RF_XML);
            // get value from config, else the default value is the value of the XML (if existed).
             //Assign the value to the class global variable use_RF
       else // if XML rules over config
            use_RF = use_RF_XML;
            // get value from XML and assign to use_RF class global variable
    }
    else // if we didn't find the tag or is of wrong type
    {
       XML_RF_map[ "use_RF" ] = false;
       use_RF = se.optional_bool_param("use_RF", true);
        // use parameter of config with default value true
    }
    .
```

```
    .
    .
 else // if group tag "RF" doesn't exist or whole <environment/> tag doesn't exist or isn't loaded
 {
    /*-----------------------------------------------------------
    ASSIGN MODEL PARAMETERS FROM CONFIG FILE ONLY!!! THIS SCENARIO IS
    EXECUTED IN CASE THE GROUP TAG "RF" ISN'T LOADED FROM AN XML FILE
    -----------------------------------------------------------*/
    std::cout << "group tag \"RF\" not found!!!!\n"
            << "using parameters from config file only if available\n";

    //-------------------------------------------------------------
    // PARAMETER ASSIGNMENT
    //-------------------------------------------------------------
    use_RF = se.optional_bool_param("use_RF", true);
     // get value from config or assign defalut value true
    .
    .
    .
 }
```

[shawn/sys/comm_models/rf_communication_model_nt_factory.cpp]

Συνοψίζοντας τον παραπάνω κώδικα έχουμε τα εξής βήματα:

<<GROUP TAG RF>>

- Ψάχνουμε την ετικέτα ομάδας "RF".
  - Αν υπάρχει ελέγχουμε τον τύπο και εφόσον είναι ο επιθυμητός ορίζουμε ένα δείκτη 'rf_gt' στην ετικέτα αλλιώς ορίζουμε στον δείκτη αυτό τιμή μηδέν (NULL).
- Αν ο δείκτης υπάρχει και δεν είναι μηδενικός:

<<SIMPLE TAG PARAMETER>>

- Αναζητούμε την ετικέτα κάθε παραμέτρου.
  - Αν υπάρχει ελέγχουμε τον τύπο και εφόσον είναι ο επιθυμητός ορίζουμε ένα δείκτη στην ετικέτα αλλιώς ορίζουμε στον δείκτη αυτό τιμή μηδέν (NULL).

- Αν ο δείκτης υπάρχει και δεν είναι μηδενικός:
  - Αναζητούμε για κάθε παράμετρο την αντίστοιχη ετικέτα.
  - Αποδίδουμε την τιμή της ετικέτας σε τοπική μεταβλητή του ιδίου τύπου.
  - Εξετάζεται η ιεραρχία μεταξύ παραμέτρων ρυθμίσεων και ετικέτας.

- Αν έχει προτεραιότητα η ρύθμιση:
  - Ορίζεται η τιμή της παραμέτρου προαιρετικά από τη ρύθμιση, με προεπιλογή την τιμή της παραμέτρου από την ετικέτα.

- Αν έχει προτεραιότητα η ετικέτα:
  - Ορίζεται η τιμή της παραμέτρου από την τοπική παράμετρο που πήρε την τιμή της ετικέτας.

- Αν ο δείκτης της απλής ετικέτας είναι μηδενικός:
  - Η παράμετρος ορίζεται από την ρύθμιση, είτε με υποχρεωτική δήλωση είτε προαιρετικά με προεπιλογή, αναλόγως της περίπτωσης.

<<GROUP TAG RF>>
- Αν ο δείκτης της ετικέτας ομάδας "RF" είναι μηδενικός:
  - Η τιμή της παραμέτρου ορίζεται από την ρύθμιση, είτε με υποχρεωτική δήλωση είτε προαιρετικά με προεπιλογή.

Η διαδικασία επαναλαμβάνεται για όλες τις παραμέτρους και εξασφαλίζει ότι δεν θα υπάρχει λάθος τύπος δεδομένων σε κάποια παράμετρο, όπως και ότι δεν θα αξιοποιηθεί παράμετρος που δεν έχει οριστεί.

5.3.3.2 Έλεγχος Λογικών Σφαλμάτων Στις Τιμές Εισόδου των Παραμέτρων

Πριν την δημιουργία του μοντέλου είναι συνετό να ελέγξουμε τις τιμές των παραμέτρων όσον αφορά την ορθότητά τους στη λογική της κάθε παραμέτρου.

Για κάθε παράμετρο υπάρχει ένα εύρος αποδεκτών τιμών. Το εύρος αυτό είτε ορίζεται ρητά σαν απαίτηση, είτε είναι θεωρητικό, λόγω της φυσικής ιδιότητας της εκάστοτε παραμέτρου. Για παράδειγμα το μοντέλο της εξασθένισης διαδρομής ορίζεται ρητά στο εύρος ακεραίων 0-4. Κάθε άλλη τιμή αποτελεί λογικό σφάλμα. Από την άλλη το ύψος των κεραιών των κόμβων δεν είναι δυνατό να είναι αρνητικό λόγω του φυσικού ορισμού του ύψους, που πρόκειται για την απόσταση από το έδαφος, και συνεπώς για αρνητικές τιμές έχουμε λογικό σφάλμα. Προκειμένου να συμπληρώνονται σωστά οι παράμετροι στα αρχεία ρυθμίσεων και XML δεδομένων, υπάρχει εκτενής τεκμηρίωση για τον ορισμό των τιμών τους.

Αν παρά τις συστάσεις ορισθεί μή επιθυμητή τιμή σε τέτοια παράμετρο, υπάρχουν διάφορες δράσεις που ακολουθούνται κατά περίπτωση. Για παράδειγμα στην περίπτωση του εύρους βλάστησης στο μοντέλο βλάστησης, αν η τιμή του παράγοντα 'foliage_path_factor' είναι μη επιθυμητή, ορίζεται από την Παραγωγό τάξη του 'RF NT' μοντέλου τιμή 50% (0.5). Στην περίπτωση όμως της μονάδας ισχύος εκπομπής, η προσομοίωση οδηγείται σε σφάλμα. Ο έλεγχος των λογικών αυτών σφαλμάτων εφόσον εντοπίσει παράλογες τιμές εισόδου ενημερώνει το χρήστη για το λογικό σφάλμα και τερματίζει τη προσομοίωση 'βίαια' με την abort() λειτουργία της C++. Σε μερικές περιπτώσεις αν απλά δίνεται πολύ μεγάλη ή πολύ μικρή τιμή σε ένα μέγεθος, τιμή που θεωρητικά δεν συναντάται στην φύση, ο έλεγχος λογικών σφαλμάτων ενημερώνει για το γεγονός και επιτρέπει την εκτέλεση της προσομοίωσης.

Ακολουθεί παράδειγμα διαχείρισης λογικού σφάλματος για τη μονάδα ισχύος:

```
if(Pt_unit!="mW" && Pt_unit!="W" && Pt_unit!="dBm")
{
    LOGICAL_ERROR_REPORT(Pt_unit);
    abort();
}
```

όπου:

LOGICAL_ERROR_REPORT(x) είναι η εξής μακροεντολή:

```
#define LOGICAL_ERROR_REPORT(x) std::cout << "LOGICAL ERROR: Input variable \"" << #x
<< "\" has illegal value \"" << (x) << "\" !!!\n"
```

Με την εντολή αυτή ενημερώνεται ο χρήστης για το όνομα της παραμέτρου που φέρει μη λογική τιμή, καθώς και για την τιμή αυτή, πριν ακυρώσει την εκτέλεση.

Αν σε μια προσομοίωση έχουμε εισάγει σωρεία παράλογων τιμών σε μεταβλητές, κάθε φορά η προσομοίωση θα ακυρώνεται στην πρώτη που θα εντοπίζεται το σφάλμα.

Ο έλεγχος λογικών σφαλμάτων γίνεται πριν την δημιουργία του μοντέλου επικοινωνίας, και πάνω στην εκάστοτε τοπική μεταβλητή της παραμέτρου. Συνεπώς δεν μπορούμε να γνωρίζουμε αν το σφάλμα προέρχεται από το αρχείο ρυθμίσεων ή από XML ετικέτα. Πρέπει πάντα να ελέγχονται και τα δύο σε περίπτωση λογικού σφάλματος.

5.3.3.3 Δημιουργία & Παραμετροποίηση Μοντέλου Επικοινωνίας 'RF NT'

Εφόσον η 'Παραγωγός' τάξη 'RFNTCommunicationModelFactory' έχει αντλήσει όλες τις παραμέτρους που απαιτούνται για τη δημιουργία του μοντέλου, είτε από ετικέτες αρχείου XML είτε από παραμέτρους ρύθμισης, επόμενο βήμα είναι η δημιουργία του μοντέλου επικοινωνίας 'RF NT' και η μετάδοση των τιμών των παραμέτρων σε αυτό. Αυτό πραγματοποιείται με την παρακάτω εντολή:

```
RFNTModel* rfm = new RFNTModel(
                                use_RF,
                                RNG,
                                pl_model,
                                f_GHz,
                                Pt,
                                Pt_unit,
                                Receiver_Sensitivity_dBm,
                                Gt_dB,
                                Gr_dB,
                                foliage,
                                f_model,
                                foliage_path_factor,
                                hr_m,
                                ht_m,
                                n,
                                std_dev_dBm,
                                P_RS,
                                P_CI,
                                verbosity
                               );
return rfm;
```

shawn/sys/comm_models/rf_communication_model_nt_factory.cpp]

Αφού το μοντέλο δημιουργηθεί επιστρέφεται ένας δείκτης σε αυτό στην προσομοίωση.

5.3.4 Μοντέλο RF NT: Αρχείο Ρυθμίσεων & XML δεδομένα

Τα αρχεία ρυθμίσεων και XML που μπορούμε να αξιοποιήσουμε για να παραμετροποιήσουμε την προσομοίωση, ακολουθούν κάποιες γενικές οδηγίες και πρότυπα, όπως παρουσιάστηκαν στα αντίστοιχα κεφάλαια. Εξαιτίας όμως των αναγκών του μοντέλου 'RF NT' προέκυψαν επιπλέον ρυθμίσεις και δεδομένα, και συνεπώς η ανάγκη για επιπλέον προτυποποίηση και ανάλυση.

5.3.4.1 Αρχείο Ρυθμίσεων

Στο αρχείο ρυθμίσεων η παράμετρος που χρίζει ιδιαίτερης προσοχής είναι η bool config_over_xml, καθότι ορίζει ποια από τις δύο επιλογές (config / XML) θα υπερισχύει της άλλης. Αν δεν οριστεί ισχύει η προεπιλογή 'true'. Είναι πολύ σημαντικό η επιλογή αυτή να είναι η πρώτη που θα οριστεί στο αρχείο ρυθμίσεων.

Επίσης εφόσον θέλουμε να αντλήσουμε τιμές παραμέτρων περιβάλλοντος από XML αρχείο, πρέπει να το δηλώσουμε στην εργασία 'prepare world' με την παράμετρο 'env_config', στην οποία αποδίδουμε ως τιμή το όνομα του XML αρχείου και αν χρειάζεται τη διαδρομή προς αυτό από το εκτελέσιμο του shawn.

Όλες οι παράμετροι πρέπει να δηλωθούν αμέσως μετά την δήλωση του μοντέλου κατά το πρότυπο "όνομα παραμέτρου = τιμή παραμέτρου". Οι παράμετροι χωρίζονται μεταξύ τους με απλό διάστημα. Ακολουθεί παράδειγμα δήλωσης:

prepare_world env_config=shawn-kaisariani_xyz_.xml edge_model=simple_nt comm_model=rf_nt config_over_xml=true path_loss_model=1 foliage_model=1 std_dev_dBm=5 ht_m=1 hr_m=3

5.3.4.2 XML Δεδομένα

Στην περίπτωση του XML αρχείου ρυθμίσεων έχουμε δύο ειδών παρεμβάσεις:

(1) Την δημιουργία στην ετικέτα περιβάλλοντος της ομαδικής ετικέτας "RF" με τις παραμέτρους του μοντέλου σαν ετικέτες - μέλη της.

(2) Την δημιουργία ετικέτας χάρτη στην πύλη της κάθε περιοχής που εξετάζουμε, όπου καταγράφονται πραγματικές μετρήσεις / δείγματα λαμβανόμενης στο δέκτη ισχύος.

5.3.4.2.1 "RF" Group Tag

Η ομαδική ετικέτα που περιλαμβάνει τις παραμέτρους για το μοντέλο "RF NT" είναι η εξής:

```
<scenario>
  <environment>
    <!-- ===============================================
    ########## RF Environment and node info ##############
    ==> In case u don't want a value comment whole line
    ================================================ -->
      <tag type="group" name="RF">
      <tag type="bool" name="use_RF" value="true"/>

      <!-- ========================================== -->
      <tag type="bool" name="RNG" value="true"/>
      <!-- ===========================================
      This option decides the function of the comm model:

      true: The comm model compares a random power value
      with the receiver sensitivity. This value is generated
      by the normal distribution with mean value the mean
      path loss and standard daviation the respective value.
```

```
false: The comm model compares the received power due
to mean path loss with the receiver sensitivity, or
computes the probability that a random value in its
distribution can be higher tha the receiver sensitivity

============================================== -->

<tag type="double" name="Connectivity_Probability_Threshold" value="0.95"/>
<tag type="double" name="Confidence_Interval_Threshhold" value="1"/>

<!-- ========================================== -->
<tag type="int" name="path_loss_model" value="2"/>
<!-- ==========================================
"path_loss_model" defines the main model to use for
attenuation calculation due to propagation path.

 Available Options Are:

0 - Free Space Propagation Model
1 - Log-Distance Model (d0=1m)
2 - Log-Distance Model (d0=variable)
3 - Egli Terrain Model
4 - Multipath Simple Model
5 - Multipath Analytic Model

default value is FSP if not defined here or in config

Foliage Model are added using the "foliage_model" tag

============================================== -->
<tag type="double" name="f_GHz" value="0.433"/>
<!-- ========================================== -->
<tag type="double" name="Pt" value="10"/>
<!-- ==========================================
Pt_Unit must take one of the following values:

"mW" for milliWatts
"W" for Watts
"dBm" for decibels

ANY OTHER value will result in error
CASE MATTERS!!!
============================================== -->
<tag type="string" name="Pt_unit" value="mW"/>
<tag type="double" name="Receiver_Sensitivity_dBm" value="-112"/>
<tag type="double" name="Gt_dB" value="2"/>
<tag type="double" name="Gr_dB" value="2"/>
<!-- ========================================== -->
<tag type="bool" name="foliage" value="true"/>
<tag type="int" name="foliage_model" value="0"/>
<tag type="double" name="foliage_path_factor" value="0.6"/>
```

```
<!-- ===============================================
  FOLIAGE MODEL PARAMETERS

  "foliage" decides weather we use or not a foliage model
  "foliage_model" explicitly defines foliage model to use

   Available options are:

  0 - Early ITU Recomendation
  1 - Weissberger Model
  2 - COST235 'in leaf' vegetation
  3 - COST235 'out of leaf' vegetation

  default value if not defined here or in config is 0

  "foliage_path_factor" is used to compute the depth
  of vegetation across the propagation path. MUST be a
  value between 1 and 0. In any other case or with negative
  values a 50% path is used.

  ================================================ -->

  <tag type="double" name="hr_m" value="1.5"/>
  <tag type="double" name="ht_m" value="1.5"/>
  <tag type="double" name="n" value="2"/>
  <tag type="double" name="std_dev_dBm" value="5"/>
 </tag>
</environment>
</scenario>
```

Κάθε ετικέτα ορίζεται από τον τύπο της, το όνομα της παραμέτρου που εκπροσωπεί και την τιμή που δίνουμε σε αυτή. Σε περιπτώσεις παραμέτρων που απαιτούνται συγκεκριμένες τιμές ή συγκεκριμένο εύρος τιμών, υπάρχει εκτενής σχολιασμός πώς πρέπει να οριστεί σωστά η τιμή.

Η ετικέτα περιβάλλοντος είναι δυνατό να είναι ενσωματωμένη σε ένα αυτόνομο αρχείο XML, ή σε αρχείο με άλλες ετικέτες όπως το αρχείο XML των συντεταγμένων των κόμβων. Δεν παύει στην δεύτερη περίπτωση να ισχύει η απαίτηση της παραμέτρους 'env_config'.

5.3.4.2.2 Ετικέτα Χάρτη μετρήσεων λαμβανόμενης ισχύος

Σε κάθε πύλη για την οποία έχουμε μετρήσεις ισχύος από γειτονικούς κόμβους - αισθητήρες, μπορούμε να περιλάβουμε τις μετρήσεις αυτές σε μια ετικέτα 'χάρτη' εντός της ετικέτας κόμβου της πύλης. Για την πύλη του δικτύου της Καισαριανής που εξετάζεται στην μελέτη, η ετικέτα είναι:

```
<node id="130">
  <location x="482643.800" y="4201487.850" z="352.672" />
  <tag type="string" name="node_type" value="gateway"/>
  <tag type="map-string-double" name="Pr_dBm_samples">
  <!-- ================================================
  map tag containing sample power levels for sensor nodes
  ================================================ -->
       <entry index="131" value="-54"/>
       <entry index="132" value="-64"/>
       <entry index="133" value="-60"/>
       <entry index="134" value="-83"/>
       <entry index="135" value="-85"/>
       <entry index="136" value="-91"/>
       <entry index="137" value="-95"/>
       <entry index="138" value="0"/>
       <entry index="139" value="0"/>
       <entry index="140" value="0"/>
       <entry index="141" value="-86"/>
       <entry index="142" value="-66"/>
       <entry index="143" value="-72"/>
       <entry index="144" value="-61"/>
       <entry index="145" value="-65"/>
       <entry index="146" value="-57"/>
  </tag>
</node>
```

Το κλειδί κάθε εγγραφής είναι ο αριθμός του κόμβου και η τιμή είναι η λαμβανόμενη ισχύς στην πύλη σε dBm. Τις τιμές αυτές αξιοποιεί το μοντέλο εξασθένισης Λογαριθμικής Απόστασης για να υπολογίσει την εξασθένιση αναφοράς. Επίσης μπορούμε να εκτιμήσουμε το επίπεδο εμπιστοσύνης με το οποίο κάθε δείγμα προσεγγίζει την αντίστοιχη μέση τιμή ισχύος υπολογισμού. Είναι πολύ σημαντικό να αποδοθεί τιμή σε κάθε εγγραφή, ακόμα και στις περιπτώσεις που δεν έχουμε δείγμα. Σε αυτές εισάγουμε την τιμή '0'. Αν δεν αποδοθεί τιμή σε κάποια εγγραφή, για κάθε τύπου ετικέτα, ο προσομοιωτής θα επιστρέψει σφάλμα και θα τερματίσε.

5.4 Τροποποίηση Μοντέλου Ακμών: Lazy Edge NT

Το μοντέλο ακμών είναι υπεύθυνο για τη διαχείριση της επικοινωνίας ανάμεσα στους κόμβους, πάντα σε συνεργασία με το μοντέλο επικοινωνίας [§1.1.2]. Η κύρια μέθοδος που εφαρμόζει το μοντέλο ακμών είναι η '*virtual bool are_adjacent(const Node& u, const Node& v, CommunicationDirection d) const throw()*', η οποία δίνει απάντηση στο ερώτημα αν δύο κόμβοι 'u' και 'v' επικοινωνούν κατά της κατεύθυνση επικοινωνίας 'd'. Αυτή η λειτουργία δηλώνεται και ορίζεται στην γονική τάξη των μοντέλων επικοινωνίας. Ως παραμέτρους εισόδου δέχεται τους δύο κόμβους 'u' και 'v', και την κατεύθυνση επικοινωνίας 'CommunicationDirection' που θέλουμε να εξετάσουμε.

Η διεύθυνση επικοινωνίας για την '*are_adjacent(u,v)*' μπορεί να είναι:

- εισερχόμενη (CD_IN)
- εξερχόμενη (CD_OUT)
- αμφίδρομη (CD_BIDI)
- οποιαδήποτε (CD_ANY)

Ο ορισμός της μεθόδου με τις συνθήκες που εξετάζονται για κάθε κατεύθυνση επικοινωνίας είναι:

```
bool
EdgeModel::
are_adjacent( const Node& u, const Node& v, CommunicationDirection d )
const throw()
{
      switch( d )
      {
      case CD_IN:
            return communication_model().can_communicate_uni(v,u);
      case CD_OUT:
            return communication_model().can_communicate_uni(u,v);
      case CD_BIDI:
            return communication_model().can_communicate_bidi(u,v);
      case CD_ANY:
            return communication_model().can_communicate_uni(u,v) ||
            communication_model().can_communicate_uni(v,u);
      }
      return false;
 }
```

[src/sys/edge_model.cpp]

Όπως παρατηρούμε η μέθοδος είναι ήδη συμβατή και επαρκής για χρήση με τα τροποποιημένα μοντέλα επικοινωνίας που δημιουργήθηκαν για τις ανάγκες της μελέτης. Αρκεί για κάθε περίπτωση να εξετάζουμε την εκάστοτε κατεύθυνση. Για τους αισθητήρες που δεν λαμβάνουν σήμα αλλά μόνο στέλνουν θα εκτελούμε την μέθοδο για την κατεύθυνση 'CD_OUT'. Ομοίως για τον κόμβο πύλη θα εφαρμόζουμε την περίπτωση 'CD_IN'. Η μέθοδος δεν μπορεί να διαχωρίσει το είδος των κόμβων, και επομένως θα εξετάζει και ζεύγη αισθητήρων, όπως και κάθε άλλου τύπου. Εφόσον όμως έχουμε ορίσει την σωστή κατεύθυνση, τότε το μοντέλο επικοινωνίας θα ακυρώσει αδύνατες συνδέσεις.

Η εφαρμογή στα μοντέλα ακμών 'Lazy' και 'Grid' εμφανίζει σωστά αποτελέσματα και ανταποκρίνεται πλήρως. Τα 'List' και 'Fast List' μοντέλα δεν εξάγουν σωστά αποτελέσματα, γεγονός που οφείλεται στις μεθόδους δόμησης της λίστας των κόμβων και τον γειτόνων τους. Συν τοις άλλοις, λόγω του ότι το μοντέλο 'RF NT' εισάγει τυχαιότητα στον υπολογισμό της επικοινωνίας σε κάθε αίτημα, με την μέθοδο *can_communicate_RNG(const Node&, const Node&)*, δεν μπορούμε να χρησιμοποιήσουμε τα μοντέλα αυτά, διότι εξετάζουν μόνο μια φορά την επικοινωνία και την αποθηκεύουν για μελοντική χρήση. Επιπλέον η αντικατάσταση της απλής παραμέτρους 'εύρος επικοινωνίας' (range) με την ανάλυση μετάδοσης ραδιοσυχνοτήτων, δεν έχει ανάγκη το μοντέλο 'Grid', καθότι αυτό ορίζει μοναδική εννιαία διάσταση για όλα τα κελιά και εξετάζει βάσει αωτών την επικοινωνία, γεγονός που δεν αναταποκρίνεται στην λογαριθμική φύση της εξέτασης που υλοποιεί το μοντέλο 'RF NT'. Το μόνο μοντέλο ακμών που σε κάθε περίπτωση εξετάζει την επικοινωνία καθαρά και μόνο με την μέθοδο *can_communicate_uni(const Node&, const Node&)* είναι το απλό 'Lazy' μοντέλο, και πάνω σε αυτό εφαρμόζουμε τις προσομοιώσεις με το μοντέλο 'RF NT'. Το Disk Graph NT μοντέλο μπορεί να συνεργαστεί όπως το απλό UDG και με το 'Grid' μοντέλο.

Παρά την επάρκεια από πλευράς δυνατοτήτων και υποστήριξης ελέγχου κατεύθυνσης, το μοντέλο ακμών στην περίπτωση της απλής έκδοσης (Lazy) που κυρίως εξετάζουμε, δεν αξιοποιεί τις κατευθύνσεις ανά περίπτωση για να θέσει το ερώτημα της επικοινωνίας δύο κόμβων στο μοντέλο επικοινωνίας. Η προεπιλεγμένη υλοποίηση χρησιμοποιεί μία μόνο κατεύθυνση, αυτή που συνοδεύει την δήλωση της μεθόδου στην γονική τάξη. Προκειμένου επομένως να αξιοποιήσουμε ουσιαστικά το μοντέλο, ώστε το μοντέλο επικοινωνίας να μπορεί να 'δει' σωστά τις συνδέσεις, πρέπει να κατευθύνουμε τον έλεγχο σύμφωνα με τις περιπτώσεις κόμβων που αντιμετωπίζουμε. Αυτό είναι ιδιαίτερα κρίσιμο γιατί αλλιώς το μοντέλο εκπομπής δεν θα έχει σωστή ενημέρωση για τους παραλήπτες των μηνυμάτων. Η επέμβαση γίνεται στον κάτωθι κώδικα:

```
 while(
(world_it_!=world_end_it_) &&
!edge_model_.are_adjacent(node_, *world_it_, base_type::direction_))
++world_it_;
```

[src/sys/edge_models/lazy_edge_model.cpp]

Πάνω στην παραπάνω επανάληψη βασίζεται ο εντοπισμός των γειτόνων ενός κόμβου με το μοντέλο ακμών 'Lazy'. Όπως φαίνεται δεν υπάρχει κάποια συνθήκη εξέτασης της κατεύθυνσης επικοινωνίας. Αν για παράδειγμα αν η βασική κατεύθυνση είναι εξερχόμενη και εξετάσουμε τους γείτονες μιας πύλης, τότε θα λάβουμε απάντηση μηδέν, διότι το μοντέλο επικοινωνίας, όπως το τροποποιήσαμε, δεν επιτρέπει αυτή τη κατεύθυνση για μια πύλη. Επομένως εισάγουμε συνθήκη εξέτασης του τύπου του κόμβου και ορίζουμε την κατεύθυνση επικοινωνίας αναλόγως.

Ο νέος κώδικας είναι:

```
//---------------------------------------------------------
// Communication direction for node type "gateway"
//---------------------------------------------------------
   if( node_.has_node_type() && node_.node_type()=="gateway" )
     while( (world_it_!=world_end_it_) &&
     !edge_model_.are_adjacent(node_, *world_it_, CD_IN))
          ++world_it_;
//---------------------------------------------------------
//Communication direction for node type "sensor"
//---------------------------------------------------------
   if( node_.has_node_type() && node_.node_type()=="sensor" )
     while( (world_it_!=world_end_it_) &&
     !edge_model_.are_adjacent(node_, *world_it_, CD_OUT))
          ++world_it_;
//---------------------------------------------------------
//Communication direction for nodes with NO type -  DEFAULT
//---------------------------------------------------------
   if( !node_.has_node_type() )
     while( (world_it_!=world_end_it_) &&
     !edge_model_.are_adjacent(node_, *world_it_, CD_ANY))
          ++world_it_;
//---------------------------------------------------------
//Communication direction for nodes with UNKNOWN type - SAFETY
//---------------------------------------------------------
   if( node_.has_node_type() &&
      node_.node_type()!="gateway" && node_.node_type()!="sensor")
     while( (world_it_!=world_end_it_) &&
     !edge_model_.are_adjacent(node_, *world_it_, base_type::direction_))
          ++world_it_;
```

[src/sys/edge_models/lazy_edge_model_nt.cpp]

5.5 Εργασία Συνδεσιμότητας (connectivity task)

Το Shawn συνοδεύεται από εργασίες, ανάμεσα στις οποίες υπάρχει μια εργασία συνδεσιμότητας με όνομα "connectivity". Η εργασία αυτή υπολογίζει το μέσο όρο και τις ακρότατες τιμές συνδεσιμότητας των κόμβων του δικτύου και τις τυπώνει στις οθόνη.

Το σκεπτικό της εργασίας προσφέρεται για την εξέταση των συνδέσεων του δικτύου και τον υπολογισμό στατιστικών συνδεσιμότητας. Πρέπει πρώτα όμως να εξασφαλισθεί η αξιοποίηση της επιπλέον πληροφορίας του είδους κάθε κόμβου, ώστε η εργασία να εξυπηρετεί το σενάριο του δικτύου αισθητήρων που εξετάζουμε. Προς την κατεύθυνση αυτή δημιουργούμε ένα κλώνο της πρωτότυπης εργασίας με τίτλο “connectivity_nt” .

Δημιουργούμε ένα χάρτη (std::map) για να αποθηκεύσουμε τα στατιστικά των κόμβων. Η οργάνωση ακολουθεί την αρχική υλοποίηση, μόνο που διαχωρίζει κάθε είδος κόμβου. Δημιουργούνται κατά σύμβαση τέσσερις περιπτώσεις:

1. Κόμβος 'Πύλη' (gateway)
2. Κόμβος 'Αισθητήρας (sensor)
3. Κόμβος αγνώστου τύπου (uknown)
4. Κόμβος χωρίς τύπο (no-type)

Παρότι δεν εξετάζεται στο σενάριο προβλέφθηκε χώρος για κόμβους αγνώστων στοιχείων. Θα ήταν ίσως ενδεικνυόμενη η ένωση των περιπτώσεων 3 και 4, εν τούτοις η αναλυτική παρουσίαση μπορεί να βοηθήσει καλύτερα στον έλεγχο σφαλμάτων.

Για κάθε κατηγορία κόμβων καταγράφονται τέσσερα στατιστικά στοιχεία:

1. Ελάχιστη Τιμή (min) : ο μικρότερος αριθμός γειτόνων/συνδέσεων για έναν κόμβο
2. Μέγιστη Τιμή (max) : ο μεγαλύτερος αριθμός γειτόνων/συνδέσεων για έναν κόμβο
3. Συνδεσιμότητα (con) : το σύνολο όλων των γειτόνων/συνδέσεων στους κόμβους
4. Πλήθος (count) : το σύνολο των κόμβων στην κατηγορία

Τα δεδομένα για τις παραπάνω τιμές προκύπτουν από την '*int nof_adjacent_nodes(const Node&, shawn::EdgeModel::CommunicationDirection)*'. Για κάθε κόμβο αναζητούνται οι γείτονες στην κατεύθυνση επικοινωνίας που υποστηρίζει. Στον πρώτο κόμβο που θα εξετάσουμε η απάντηση θα αποδοθεί σε όλες τι μεταβλητές εκτός του πλήθους κόμβων, που θα πάρει τιμή '1'. Για τους επόμενους κόμβους συγκρίνεται κάθε φορά ο αριθμός συνδέσεων με τις τιμές ακροτάτων της κατηγορίας στον χάρτη, που έχουν αποδοθεί σε προηγούμενες καταγραφές. Αν είναι μικρότερος της ελάχιστης ή μεγαλύτερος της μέγιστης παίρνει την θέση της αντίστοιχης τιμής. Επιπλέον προστίθεται στη τιμή συνδεσιμότητας και αυξάνεται κατά ένα το πλήθος των κόμβων στην καταγραφή του χάρτη. Τα τελικά στοιχεία που εκτυπώνει η εργασία περιλαμβάνουν την ελάχιστη και μέγιστη τιμή της συνδέσεων σε ένα κόμβο της κατηγορίας, καθώς και τη μέση τιμή που ισούται με το λόγο της συνδεσιμότητας δια το πλήθος των κόμβων.

5.6 Εργασία Αναφοράς RF Εξασθένισης (rf_nt_attenuation task)

Η εργασία αυτή δημιουργήθηκε αποκλειστικά για το σκοπό της συλλογής και καταγραφής των εξασθενίσεων στην ισχύ εκπομπής, όπως υλοποιούνται στα μοντέλα του Shawn. Αυτό που εφαρμόζει συγκεκριμένα είναι να καλεί με τη σειρά όλες τις μεθόδους υπολογισμού εξασθένισης που έχουν υλοποιηθεί στο 'RF NT' μοντέλο [§5.3.2.α] και να υπολογίζει για κάθε ζεύγος πομπού-δέκτη, όλες τι εξασθενίσεις με όλα τα μοντέλα.

Στην ετικέτα περιβάλλοντος του XML αρχείου δεδομένων δημιουργούμε ομαδική ετικέτα με όνομα “RF_REPORT”. Σε αυτή εισάγουμε bool εγγραφές με τα ονόματα όλων των μοντέλων υπολογισμού εξασθένισης. Κάθε φορά που θα εκτελείται η εργασία θα ελέγχει ποια από αυτά είναι ενεργά (bool = true) και θα εκτυπώνει την εξασθένισή τους στο αρχείο εξόδου.

Η διαδικασία ξεκινάει εντοπίζοντας τους κόμβους που έχουν επεξεργαστή gateway. Στη συνέχεια για κάθε κόμβο υπολογίζονται οι εξασθενίσεις των ενεργών μοντέλων για όλους του αισθητήρες, ανεξάρτητα αν επικοινωνούν ή όχι. Δεν εξετάζεται η συνδεσιμότητα των κόμβων, ούτε εκτελείται η *bool can_communicate_uni(const Node&, const Node&)*. Μόνο οι μέθοδοι υπολογισμού της εξασθένισης καλούνται. Η κλίση του μοντέλου γίνεται με αναφορά (by reference). Για κάθε πύλη τυπώνεται αρχείο με τις τιμές αυτές, του οποίου το όνομα ξεκινάει με το όνομα του κόμβου και συμπληρώνεται με την κατάληξη “_attenuation_report.txt”. Στο αρχείο τυπώνονται και οι παράμετροι του μοντέλου επικοινωνίας, οι σχετικές με τη μετάδοση ραδιοσυχνοτήτων. Οι εγγραφές για κάθε νέα εκτέλεση τυπώνονται στο τέλος του αρχείου, χωρίς να σβήνουν παλιά δεδομένα.

Παρουσιάζεται μια εκτύπωση αναφοράς για την περιοχή της Καισαριανής. Οι τιμές της εξασθένισης για κάθε μοντέλο έχουν εισαχθεί σε πίνακα για την καλύτερη εποπτεία τους. Η αναφορά έχει αποθηκευθεί στο αρχείο “130_attenuation_report.txt”, αφού ο κόμβος 130 είναι η πύλη στο δίκτυο αυτό:

```
------------------------------------------------------------
Simulation RF paramater values:
Frequency: 0.433 Ghz
Transmit Power: 10.00 mW
Receiver Sensitivity: -112 dBm
Transmitter Antenna Gain: 2.0 dB
Receiver Antenna Gain: 2.0 dB
Foliage Path percent: 100 %
Receiver Antenna Height: 1.50 m
Transmitter Antenna Height: 1.50 m
Log-Distance Path Loss Exponent: 2.00
------------------------------------------------------------
```

130	FSP	LD	LD_1m	L50	LMP1	LMP2	W	ITU_R	C235_IN	C235_OUT
131	18.1507	44.8748	18.1507	6.6181	-17.0973	nan	0.2504	1.0027	13.0291	3.8919
132	50.0277	76.7518	50.0277	38.4951	46.6567	47.4579	7.3928	9.0674	48.8314	24.3827
133	57.3137	84.0378	57.3137	45.7811	61.2286	61.3761	12.1065	14.9989	66.0460	37.0878
134	60.1651	86.8892	60.1651	48.6325	66.9315	67.0078	14.6842	18.2643	74.3314	43.7035
135	62.7512	89.4753	62.7512	51.2186	72.1037	72.1457	17.4936	21.8367	82.7413	50.7188
136	63.6741	90.3982	63.6741	52.1415	73.9495	73.9835	18.6214	23.2741	85.9675	53.4861
137	64.3283	91.0524	64.3283	52.7957	75.2579	75.2872	19.4647	24.3501	88.3305	55.5389
138	66.5377	93.2618	66.5377	55.0051	79.6768	79.6943	22.6050	28.3650	96.8010	63.0714
139	66.6279	93.3520	66.6279	55.0953	79.8571	79.8743	22.7434	28.5422	97.1634	63.3995
140	67.7204	94.4445	67.7204	56.1878	82.0421	82.0555	24.4892	30.7796	101.6640	67.5147
141	64.9908	91.7149	64.9908	53.4582	76.5829	76.6080	20.3575	25.4903	90.7894	57.6977
142	62.1858	88.9099	62.1858	50.6532	70.9729	71.0208	16.8367	21.0002	80.8248	49.0946
143	59.6788	86.4029	59.6788	48.1462	65.9589	66.0443	14.2086	17.6610	72.8483	42.4971
144	58.0230	84.7471	58.0230	46.4904	62.6473	62.7725	12.7020	15.7522	68.0166	38.6335
145	51.7661	78.4901	51.7661	40.2334	50.1334	50.6670	8.3160	10.2243	52.4795	26.9488
146	48.5317	75.2558	48.5317	36.9991	43.6647	44.8045	6.6808	8.1771	45.8956	22.3708

Πίνακας 5.1

5.7 Επεξεργαστές (Processors)

Στο Shawn οι εφαρμογές υλοποιούνται ως επί το πλείστον με δύο τρόπους, με εργασίας (common tasks [§1.6]) και με επεξεργαστές (processors [§1.3.3]). Η υλοποίηση στην πρώτη περίπτωση αφορά κυρίως εφαρμογές που θέλουμε να γίνουν καθολικά στο περιβάλλον της προσομοίωσης, ενώ στη δεύτερη αποδίδουμε τις εφαρμογές εκείνες που θέλουμε να είναι συνδεδεμένες με τον εκάστοτε κόμβο και να εκτελούνται σε αυτόν αποκλειστικά. Επίσης οι επεξεργαστές είναι η ιδανική πλατφόρμα για την αποστολή και παραλαβή μηνυμάτων, καθώς και για τη επανάληψη εφαρμογών.

Για το σενάριό μας δημιουργήσαμε τρεις επεξεργαστές:

1. proc_manager : Διαχειριστής Επεξεργαστών
2. sensor1 : Επεξεργαστής Αισθητήρα
3. gateway1 : Επεξεργαστής Πύλης

5.7.1 Διαχειριστής Επεξεργαστών: proc_manager

Επεξεργαστής διαχείρισης. Είναι ο μοναδικός που φορτώνεται στον κόσμο της προσομοίωσης ως προεπιλογή. Εργασία του είναι να εξετάσει το είδος των κόμβων και αναλόγως να φορτώσει στον καθένα τους επιπλέον επεξεργαστές. Ως εκ τούτου απαραίτητη είναι μόνο η μέθοδος εκκίνησης *void boot()* κατά την οποία αφού εκτελέσει το έργο του μπορούμε και να τον απενεργοποιήσουμε.

Δεδομένου ότι είναι ο πρώτος επεξεργαστής που φορτώνεται στην λίστα επεξεργαστών κάθε κόμβου είναι πολύ σημαντικό να μην υλοποιήσουμε την '*bool process_message()*' μέθοδο επεξεργασίας μηνυμάτων, ή αν το κάνουμε να εξασφαλίσουμε ότι επιστρέφει τιμή 'false'. Αυτό είναι απαραίτητο διότι η μέθοδος παράδοσης μηνυμάτων, που είναι υλοποιημένη στα μοντέλα επικοινωνίας, παραδίδει ένα μήνυμα στους επεξεργαστές ενός κόμβου με τη σειρά που αυτοί έχουν φορτωθεί, μέχρι κάποιος να επιστρέψει 'true'. Επομένως αν γίνει αυτό για ένα μήνυμα στον πρώτο επεξεργαστή δεν θα πάει στους υπόλοιπους. Μιας και στο παράδειγμά μας δεν θέλουμε να επεξεργάζονται πολλοί επεξεργαστές ένα μήνυμα ώστε να τροποποιήσουμε την υλοποίηση του μοντέλου επικοινωνίας, εξασφαλίζουμε ότι ο επεξεργαστής διαχείρισης δεν θα παρεμβαίνει σε αυτή την επιλογή, απενεργοποιώντας τελείως την μέθοδο.

Καθώς δεν έχει λόγο να στέλνει μηνύματα δεν χρειάζεται να υλοποιήσουμε κάποια τάξη μηνυμάτων για τον επεξεργαστή αυτό. Επιπλέον μπορούμε να μην υλοποιήσουμε ούτε την 'work()' μέθοδο που εκτελείται σε κάθε γύρο προσομοίωσης.

Ο επεξεργαστής εξετάζει στην μέθοδο εκκίνησης την παράμετρο 'std::string node_type' για κάθε κόμβο και αναλόγως φορτώνει τον κατάλληλο επεξεργαστή. Ο κώδικας είναι:

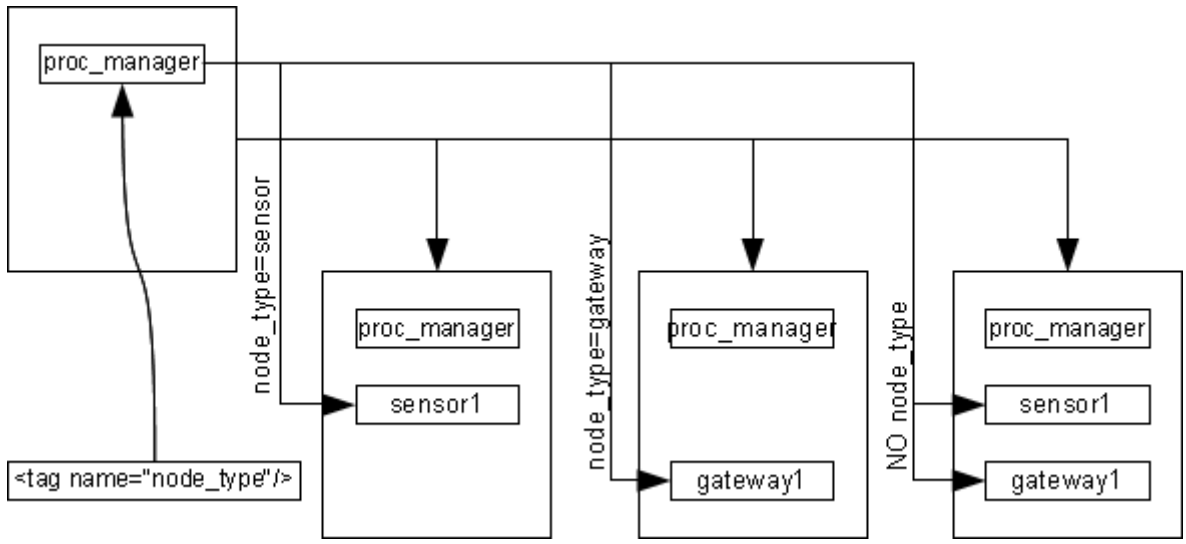
```
if(owner().has_node_type())
{
  if( owner().node_type()=="gateway")
  {
     ProcessorFactoryHandle pfh =
     owner_w().world_w().simulation_controller_w().processor_keeper_w().find_w( "gateway1" );
     if ( pfh.is_not_null() )
        owner_w().add_processor( pfh->create() );
  }
  else if( owner().node_type()=="sensor")
  {
     ProcessorFactoryHandle pfh =
     owner_w().world_w().simulation_controller_w().processor_keeper_w().find_w( "sensor1" );
     if ( pfh.is_not_null() )
       owner_w().add_processor( pfh->create() );
   }
   else
      return;
}
```

```
else if(!owner().has_node_type())
{
  ProcessorFactoryHandle pfh1 =
  owner_w().world_w().simulation_controller_w().processor_keeper_w().find_w( "gateway1" );
  ProcessorFactoryHandle pfh2 =
  owner_w().world_w().simulation_controller_w().processor_keeper_w().find_w( "sensor1" );
  if ( pfh1.is_not_null() )
    owner_w().add_processor( pfh1->create() );
  if ( pfh2.is_not_null() )
    owner_w().add_processor( pfh2->create() );
}
```

[src/legacyapps/processors/proc_manager/proc_manager_processor.cpp]

Η λειτουργία '*owner()*' επιστρέφει τον κόμβο που ανήκει ο επεξεργαστής. Όπως φαίνεται από τον κώδικα υπάρχουν τέσσερα πιθανά σενάρια:

1. Ο κόμβος να είναι Πύλη, οπότε φορτώνεται ένας επεξεργαστής 'gateway1'
2. Ο κόμβος να είναι Αισθητήρας, οπότε φορτώνεται ένας επεξεργαστής 'sensor1'
3. Ο κόμβος έχει τύπο, αλλά δεν ανήκει σε κάποιο από τους δύο προαναφερθέντες. Σε αυτή τη περίπτωση δεν φορτώνουμε επιπλέον επεξεργαστές και ο κόμβος δεν θα εκτελεί καμιά εργασία κατά την προσομοίωση.
4. Ο κόμβος δεν έχει κάποιο τύπο. Προκειμένου να ελέγχουμε σενάρια όπου όλοι στέλνουν και λαμβάνουν μηνύματα φορτώνουμε σε αυτή την περίπτωση και τους δύο επιπλέον επεξεργαστές. Πρώτα φορτώνουμε τον 'gateway1' προκειμένου να προηγείται του 'sensor1' στην ιεραρχία παραλαβής μηνυμάτων από το μοντέλο επικοινωνίας. Βέβαια δεν είναι αυστηρό αυτό αφού στον επεξεργαστή 'sensor1' δεν υλοποιούμε την '*bool process_message()*'.



Σχήμα 5.1

Όλοι οι επεξεργαστές που τυχόν θα δημιουργηθούν στην '*boot()*' μέθοδο του επεξεργαστή διαχείρισης, θα εκτελέσουν αμέσως και τις δικές τους '*boot()*' μεθόδους, όπως δημιουργείται ο καθένας. Δηλαδή η συγκεκριμένη υλοποίηση δεν παρεμβαίνει στον χρόνο της προσομοίωσης. Η συμπεριφορά του προσομοιωτή είναι η ίδια με το να φορτώναμε τους επεξεργαστές στην δόμηση του κόσμου ως προεπιλογή ή με κάποιο άλλο τρόπο. Με την υλοποίηση αυτή όμως έχουμε καλύτερο έλεγχο στην δόμηση του δικτύου και φυσικά μπορούμε να τροποποιήσουμε τις παραπάνω συνθήκες εισάγοντας περισσότερους επεξεργαστές και επιπλέον τύπους κόμβων.

Αφού ο επεξεργαστής εκτελέσει την αποστολή τους μπορούμε να τον απενεργοποιήσουμε με την λειτουργία '*set_state(state)*' στο τέλος της '*boot()*' μεθόδου:

```
set_state( Inactive );
```

[src/legacyapps/processors/proc_manager/proc_manager_processor.cpp]

5.7.2 Επεξεργαστής Αισθητήρα sensor1

Ο επεξεργαστής αυτός υλοποιείται στις περιπτώσεις που ένας κόμβος αποστέλλει μηνύματα. Η ονομασία δεν έχει καμία σχέση με την εργασία της συγκομιδής πληροφορίας από το περιβάλλον για κάποιο φυσικό μέγεθος. Απλά αντιπροσωπεύει τις επικοινωνιακές δυνατότητες ενός αισθητήρα, δηλαδή να στέλνει μηνύματα στο δίκτυο που ανήκει. Η τάξη του επεξεργαστή sensor1 είναι παράγωγη των τάξεων shawn::Processor και shawn::EventScheduler::EventHandler, δηλαδή ενσωματώνει και τις λειτουργίες του EventHandler που διαχειρίζεται προθεσμίες γεγονότων που έχουν οριστεί στον EventScheduler.

Στην περίπτωση του επεξεργαστή αυτού υλοποιούμε τέσσερις μεθόδους:

(1) virtual void boot(void)

Η μέθοδος *void boot()* εκτελείται κατά την φόρτωση του επεξεργαστή στον κόμβο που τον περιέχει, δηλαδή μόνο μία φορά. Είναι δυνατό εδώ να εκτελέσουμε εργασίες εκκίνησης και να στείλουμε τα πρώτα μηνύματα προς το δίκτυο ή να προγραμματίσουμε γεγονότα στον 'Προγραμματιστή Γεγονότων' (EventScheduler) ώστε να εκτελεστούν στον χρόνο που ορίζουμε με την μέθοδο *timeout()*.

(2) virtual void work(void)

Η μέθοδος *void work()* εκτελείται στο τέλος κάθε γύρου προσομοίωσης. Σε αυτή προγραμματίζουμε εργασίες που θέλουμε να εκτελούνται περιοδικά. Για τον σκοπό αυτό χρησιμοποιούμε την μέθοδο *send(const shawn::MessageHandle&)*. Η μέθοδος υπερφορτώνεται με νέο ορισμό στον επεξεργαστή. Η *work()* μέθοδος του αισθητήρα δεν υλοποιεί κάποια άλλη λειτουργία.

(3) virtual void send(const shawn::MessageHandle&)

Η μέθοδος αυτή υπερφορτώνεται με νέο ορισμό στον επεξεργαστή. Στον ορισμό αυτό ορίζεται στο μήνυμα που αποστέλλουμε η επιπλέον παράμετρος του επεξεργαστή που το στέλνει, καθώς και το μέγεθός του σε 'bytes'. Κατά τα άλλα είναι η ίδια μέθοδος με την γενική. Η αποστολή μηνυμάτων υλοποιείται στις μεθόδους *boot()* και *work()* και ορίζεται ως εξής:

send(new Sensor1Message ()

όπου Sensor1Message είναι η τάξη μηνυμάτων που υλοποιείται για τον επεξεργαστή sensor1. Δεν είναι υποχρεωτικό οι τάξεις μηνυμάτων να ορίζονται για κάθε επεξεργαστή διαφορετικά, είναι όμως πρακτικό σαν ένα επιπλέον μέτρο ελέγχου. Το μήνυμα με την εντολή *send()* αποστέλλεται στο μοντέλο εκπομπής. Αυτό εξετάζει μέσω του μοντέλου ακμών και του μοντέλου επικοινωνίας ποιοι κόμβοι είναι οι δυνατοί παραλήπτες του. Τέλος αποστέλλει αντίγραφο του μηνύματος στους επεξεργαστές κάθε κόμβου που μπορεί να το παραλάβει.

(4) void timeout(shawn::EventScheduler & event_scheduler,
shawn::EventScheduler::EventHandle event_handle, double time,
shawn::EventScheduler::EventTagHandle & event_tag_handle)

Η μέθοδος αυτή εκτελείται όταν λήξει η χρονική προθεσμία (time) ενός γεγονότος (shawn::EventScheduler::EventHandle event_handle) που έχει ορίσει ο επεξεργαστής στον EventScheduler του shawn. Συνήθως προγραμματίζουμε νέο γεγονός στον EventScheduler πριν την έξοδο από την μέθοδο προκειμένου να επαναληφθεί στο μέλλον.

Στον επεξεργαστή η αποστολή μηνυμάτων υλοποιείται και με την *work()* μέθοδο και με την *timeout()*. Η επιλογή κάποιας εξ των δύο γίνεται με την παράμετρο 'msg_proc'. Για τιμή 0 εφαρμόζεται η *work()* μέθοδος, ενώ για οποιαδήποτε άλλη η *timeout()*.

Όπως μπορεί κανείς να παρατηρήσει δεν υλοποιείται η *virtual bool process_message(const shawn::ConstMessageHandle&)* μέθοδος αφού οι αισθητήρες δεν λαμβάνουν μηνύματα, παρά μόνο στέλνουν.

5.7.3 Επεξεργαστής Κόμβου gateway1

Ο επεξεργαστής αυτός υλοποιείται στις περιπτώσεις που ένας κόμβος λαμβάνει μηνύματα. Σε αυτόν υλοποιούνται τρεις μέθοδοι:

(1) virtual void boot(void)

Η *boot()* μέθοδος, όπως και στην περίπτωση του sensor1 επεξεργαστή, εκτελείται μια φορά κατά την φόρτωση του επεξεργαστή στον κόμβο που ανήκει. Η μόνη εργασία που εκτελεί σε αυτό τον επεξεργαστή είναι να ορίσει τον τελευταίο γύρο προσομοίωσης, βάσει της προαιρετικής παραμέτρου των συνολικών γύρων της προσομοίωσης:

final_round = owner().world().simulation_controller().environment().optional_int_param
("max_iterations", std::numeric_limits<int>::max()) - 1;

Αφαιρούμε 1 από τον αριθμό των γύρων, καθότι η αρίθμησή τους ξεκινάει από το 0 και όχι από το 1. Δηλαδή όταν ορίζουμε μέγιστο αριθμό γύρων 2, σημαίνει ότι ο τελευταίος γύρος είναι ο 1 και όχι ο 2. Η παράμετρος int final_round αξιοποιείται στη *work()* μέθοδο του επεξεργαστή για τυπώσει τα τελικά στατιστικά της προσομοίωσης στον τελευταίο γύρο αυτής.

(2) virtual bool process_message(const shawn::ConstMessageHandle&)

Η μέθοδος αυτή επεξεργάζεται τα μηνύματα που αποστέλλει το μοντέλο εκπομπής στον κόμβο και τον επεξεργαστή. Στην περίπτωση αυτή έχει δημιουργηθεί υποδομή προκειμένου ο κόμβος να εξετάζει μόνο τα μηνύματα που αποστέλλονται από αισθητήρες, αλλιώς ενημερώνει ότι δεν μπορεί να διαχειριστή το μήνυμα και επιστρέφει τιμή 'false'. Εφόσον το μήνυμα που εξετάζει είναι μήνυμα αισθητήρα (Sensor1Message) αποθηκεύει τον αριθμό του μηνύματος για τον αισθητήρα που το έστειλε σε std::map< const shawn::Node*,int > [7] και επιστρέφει την τιμή true στο μοντέλο εκπομπής. Επιλέγεται η χρήση map διότι επιβάλει την μοναδικότητα του κλειδιών του, τα οποία εν προκειμένω είναι δείκτες σε κόμβους (const::Node*). Έτσι κάθε φορά που λαμβάνει μήνυμα από έναν αισθητήρα ο επεξεργαστής αυξάνει κατά μια μονάδα την τιμή που αντιστοιχεί στο κλειδί του αισθητήρα στο map.

Ακολουθεί ο κώδικας που υλοποιεί την τακτική αυτή:

```
std::map<const shawn::Node*, int> neighbours_msgs_ //unique container with pointer to Nodes as
                                                   //key and number of msgs as value
```

[shawn/legacyapps/processors/gateway1/gateway1_processor.h]

```
process_message( const ConstMessageHandle& mh ) // process message mh
{
   const sensor1::Sensor1Message* hmsg =
   dynamic_cast<const sensor1::Sensor1Message*> ( mh.get() );
   //cast to expected type and assign pointer to it
   if ( hmsg != NULL ) //if pointer is not NULL
   {
      neighbours_msgs_[ &hmsg->source() ]++;
      //increase by one the number of messages that this gateway
      // has received from the source of this message
      .
      .
      .
      return true;
   }
   else
   {
      return false;
    }
```

[shawn/legacyapps/processors/gateway1/gateway1_processor.cpp]

Η μέθοδος ενημερώνει κάθε φορά που λαμβάνει μήνυμα για τον αριθμό μηνυμάτων που έχει λάβει συνολικά από το συγκεκριμένο κόμβο

(3) virtual void work(void)

Η *work()* μέθοδος στον επεξεργαστή gateway1 χρησιμοποιείται αποκλειστικά για αναφορά παραλαβής μηνυμάτων και γειτόνων:

ΓΕΙΤΟΝΕΣ:

Σε κάθε γύρο ο επεξεργαστής ενημερώνει στο τέλος του για τον συνολικό αριθμό των γειτόνων του σε αυτό το γύρο και την ταυτότητα καθενός από αυτούς. Οι γείτονες αυτοί ορίζονται βάσει των μηνυμάτων που έλαβε στο γύρο αυτό ο κόμβος στην μέθοδο *process_message(const ConstaMessageHandle& mh)*. Στη μέθοδο αυτή, πέρα από το map με τις εγγραφές των μηνυμάτων, ορίζεται και ένα set neighbours_, το οποίο σε κάθε γύρο καταγράφει την ταυτότητα των αποστολέων των μηνυμάτων που έλαβε ο κόμβος:

```
process_message( const ConstMessageHandle& mh )
{
    //...
    neighbours_.insert( &hmsg->source() )
    //...
}
```

[shawn/legacyapps/processors/gateway1/gateway1_processor.cpp]

Όπως και η δομή map έτσι και το set [7] έχει μοναδικές εγγραφές. Συνεπώς αν λάβουμε περισσότερα από ένα μηνύματα από τον ίδιο κόμβο θα δημιουργηθεί μόνο μια εγγραφή. Επιπλέον οι εγγραφές είναι ταξινομημένες κατά όνομα. Ανακτώντας επομένως το μέγεθος του set και τις εγγραφές του έχουμε των αριθμό των γειτόνων του κόμβου και τις ταυτότητές τους.

Στον τελευταίο γύρο της προσομοίωσης εκτυπώνεται και ο συνολικός αριθμός των κόμβων από τους οποίους παρέλαβε μηνύματα ο επεξεργαστής και η ταυτότητα αυτών. Αυτό γίνεται με το μέγεθος του map neighbours_msgs_ και με επανάληψη στα κλειδιά του.

ΜΥΝΗΜΑΤΑ:

Η μέθοδος *process_message(const ConstMessageHandle& mh)* για κάθε μήνυμα που επεξεργάζεται σε ένα γύρο δίνει μοναδιαία αύξηση στην τιμή της μεταβλητής int round_msgs, η οποία δηλώνει τον αριθμό των μηνυμάτων που παρέλαβε σε αυτό το γύρο η μέθοδος. Η μεταβλητή είναι παγκόσμια και όχι τοπική, οπότε η τιμή της διατηρείται και εκτός της μεθόδου. Η *work()* μέθοδος δεδομένου ότι εκτελείται πάντα στο τέλος του γύρου, διαβάζει την τιμή αυτή και αναφέρει τον συνολικό αριθμό των μηνυμάτων που παρέλαβε ο επεξεργαστής στο γύρο. Πριν τερματίσει ορίζει τιμή 0 στην round_msgs μεταβλητή, προκειμένου στον επόμενο γύρο η αρίθμηση να αρχίσει πάλι από την αρχή.

Στον τελευταίο γύρο αναφέρεται ο συνολικός αριθμός μηνυμάτων που παρέλαβε ο κόμβος κατά την διάρκεια της προσομοίωσης. Αυτό γίνεται αθροίζοντας τους μετρητές μηνυμάτων από το map neighbours_msgs_.

OMNeT++

ΥΛΟΠΟΙΗΣΗ:

Στο Omnet++ κατασκευάζουμε ένα προσομοιωτή από το μηδέν, σε αντίθεση με τα άλλα δύο λογισμικά όπου πραγματοποιούμε τόσο κατασκευή όσο και τροποποίηση. Ο προγραμματισμός των μοντέλων, των αντικειμένων και των σχέσεών τους είναι τελείως στη κρίση μας. Μοναδική υποδομή η βιβλιοθήκη τάξεων και λειτουργιών του Omnet++. Η αρχιτεκτονική του Shawn και ο τρόπος λειτουργίας των κόμβων και των μοντέλων του άσκησαν μεγάλη επιρροή στις αποφάσεις που πάρθηκαν για την κατασκευή του προσομοιωτή του Omnet++. Ο προγραμματισμός στο Omnet++ έγινε με απόλυτο σεβασμό στη δική του αρχιτεκτονική σχεδιασμού προσομοιώσεων. Το μοντέλο επικοινωνίας βασίζεται στα ίδια μοντέλα εξασθένισης σήματος που χρησιμοποιούνται στο Shawn και αναλύθηκαν στο κεφάλαιο 4. Ενσωματώνονται στην προσομοίωση σύμφωνα με τις αρχές σχεδιασμού του Omnet++.

6. Omnet++: Υλοποίηση Προσομοίωσης

Η υλοποίηση μιας προσομοίωσης στο Omnet++ είναι αρκετά διαφορετική σε σχέση με το Shawn. Αυτό οφείλεται κυρίως στην παντελή έλλειψη μοντέλων και υποδομών σε επίπεδο βάσης, για την εξέταση ασύρματης επικοινωνίας. Τα εργαλεία που προσφέρει ωστόσο είναι υπεραρκετά για την υλοποίησή τους σε μορφή απλής ή σύνθετης μονάδος (module), αρκεί να μπορεί αυτή να εκφρασθεί στο πλαίσιο της εκτέλεσης διακριτών γεγονότων και της εκκίνησής τους μέσω μηνυμάτων.

Πριν τον οποιοδήποτε σχεδιασμό και την υλοποίηση των μοντέλων επικοινωνίας και των κόμβων ενός δικτύου, απαραίτητη προϋπόθεση είναι να υπάρχει μια ξεκάθαρη εικόνα για τη σύνθεση του δικτύου αυτού. Η εικόνα αυτή μπορεί να παρουσιασθεί σε ένα οργανόγραμμα. Η σχεδίαση του πρέπει να διέπει τις αρχές οργάνωσης μιας προσομοίωσης, όπως αυτές παρουσιάσθηκαν στην ανάλυση της NED γλώσσας προγραμματισμού.

6.1 Οργάνωση της Προσομοίωσης: Οργανόγραμμα

Στο Omnet++ οι οντότητες ενός δικτύου υλοποιούνται σε μονάδες (modules). Οντότητες δεν αποτελούν μόνο οι τερματικοί κόμβοι, αλλά κάθε ενεργή διαδικασία και αντικείμενο που συμμετέχει στο φαινόμενο της επικοινωνίας.

Σε ένα ενσύρματο δίκτυο δεδομένων για παράδειγμα, οφείλουμε να υλοποιήσουμε σε απλές μονάδες (simple modules) όλες τις οντότητες που διαχειρίζονται ένα πακέτο δεδομένων που αποστέλλεται από έναν κόμβο σε έναν άλλο, όπως δρομολογητές, 'modems', 'hubs', κλπ. Όλες δε οι οντότητες ανήκουν συνήθως σε διαφορετικά επίπεδα ιεραρχίας (οικιακό, τοπικό, ... περιφερειακό, εθνικό, διεθνές), τα οποία μπορούμε να υλοποιήσουμε επαρκώς με τις σύνθετες μονάδες (compound modules) και τις αρχές ιεραρχίας του Omnet++. Οι συνδέσεις (connections) για τις μονάδες αυτές, μοντελοποιούν την ενσύρματη επικοινωνία επαρκώς [17].

Στην περίπτωση όμως των ασυρμάτων δικτύων δεν παρεμβάλλεται τίποτα στην επικοινωνία ανάμεσα σε δύο κόμβους, πλην του περιβάλλοντος χώρου και της ατμόσφαιρας. Τις επιδράσεις αυτών είναι που πρέπει να μοντελοποιήσουμε σε απλές μονάδες, ώστε να προσομοιώσουμε τις συνθήκες επικοινωνίας ανάμεσα στους κόμβους. Οι απλές αυτές μονάδες εν τέλει θα συγκροτούν μια σύνθετη, η οποία θα αναπαριστά το μοντέλο επικοινωνίας του μέσου.

Στο Omnet++ οι μονάδες ενεργοποιούν τις λειτουργίες/μεθόδους τους με την παραλαβή και αποστολή μηνυμάτων [17]. Είτε αυτά τα μηνύματα θα είναι selfMessages που θα στέλνει η ίδια η μονάδα στον εαυτό της με την scheduleAt(double time, cMessage *selfMessage), είτε θα είναι μηνύματα που ανταλλάσσουν διαφορετικές μονάδες μεταξύ τους με τις send(cMessage *msg, const char *gateName) και sendDirect(cMessage *msg, double delay, cGate *gate). Τα μηνύματα λαμβάνονται στην handleMessage(cMessage*) μέθοδο και εκτελούνται οι όποιες λειτουργίες της μονάδας [17].

Η οργάνωση της προσομοίωσης πρέπει συνεπώς να βασίζεται στην ροή μηνυμάτων μεταξύ των μονάδων του δικτύου. Στο πρώτο επίπεδο ιεραρχίας βρίσκονται οι κόμβοι και το μοντέλο επικοινωνίας. Πρέπει να εξεταστεί το είδος της υλοποίησης αυτών των μονάδων, αν θα πρέπει να ορισθούν σαν απλές μονάδες ή σύνθετες, και αναλόγως να ορισθούν παράμετροι, συνδέσεις και υπομονάδες. Αυτό γίνεται στον ορισμό του δικτύου στη NED γλώσσα προγραμματισμού. Αυτός βασίζεται στον ορισμό των υπομονάδων του, των κόμβων και του μοντέλου επικοινωνίας, που είναι και η αφετηρία για τη δόμηση της οργάνωσης της προσομοίωσης.

6.2 NED Ορισμός του Δικτύου Προσομοίωσης

Μια προσομοίωση στο Omnet++ υλοποιείται σε ένα δίκτυο. Ο ορισμός του δικτύου στην NED γλώσσα, είναι το πρώτο βήμα για την υλοποίησή της, και περιγράφει την οργάνωση των λειτουργικών του μονάδων και τις σχέσεις μεταξύ τους [§2.7]. Το δίκτυο ορίζεται σαν σύνθετη μονάδα, η οποία περιέχει ως υπομονάδες τους κόμβους του και το μέσο επικοινωνίας. Επομένως είναι απαραίτητο πρώτα να ορισθούν οι υπομονάδες αυτές πριν τον ορισμό του συνολικού δικτύου.

6.2.1 NED Ορισμός Κόμβων

Οι κόμβοι του δικτύου είναι δύο ειδών:

- Αισθητήρες - Sensors
- Πύλες - Gateways

Σε ένα δίκτυο οι κόμβοι μπορούν είτε να ορίζονται ως διάνυσμα μονάδας κόμβου, είτε να ορίζεται ο καθένας ξεχωριστά σαν τελείως διαφορετική μονάδα. Η δεύτερη λύση δεν είναι σε καμία περίπτωση αποδεκτή, οπότε καταφεύγουμε απαραίτητα σε διάνυσμα μονάδας.

Για να υλοποιήσουμε μια μονάδα που να ορίζεται σαν κόμβος και να υλοποιεί διαφορετικές λειτουργίες αναλόγως με το είδους του, υπάρχουν δύο πρώτες σκέψεις. Η πρώτη είναι να ορίσουμε τους κόμβους σαν απλές μονάδες, ενώ η δεύτερη σαν σύνθετες. Κριτήριο στην επιλογή είναι το γεγονός ότι πρέπει να έχουμε δύο διαφορετικές λειτουργίες βάσει του είδους του κόμβου, και ότι ο ορισμός τους πρέπει να γίνει σε διάνυσμα.

Εξαιτίας αυτού του γεγονότος, η λύση της απευθείας δήλωσης σε απλή μονάδα δεν είναι η καλλίτερη και απορρίπτεται, και αυτό οφείλεται στο ότι τα μέλη ενός διανύσματος μονάδας είναι ακριβώς ίδια μεταξύ τους, δηλαδή του ίδιου τύπου μονάδας. Αν αυτή είναι απλή (simple module), τότε όλοι οι κόμβοι θα είναι ενσάρκωση της ίδιας C++ τάξης. Σε αυτή τη περίπτωση, προκειμένου να εκτελούμε διαφορετικές λειτουργίες αναλόγως του είδους του κόμβου, θα πρέπει να εισάγουμε δηλώσεις επιλογής σε όλες τις μεθόδους της τάξης. Τελικά δηλαδή πομπός και δέκτης θα είναι η ίδια απλή μονάδα, με διαφοροποίηση στην εκτέλεση των μεθόδων της βάσει του ελέγχου του τύπου της. Η υλοποίηση αυτή δεν είναι ιδιαίτερα κομψή ούτε εκφράζει την λογική υλοποίησης προσομοιώσεων στο Omnet++, καθότι δεν εκμεταλλεύεται τις δυνατότητες της NED οργάνωσης.

Στην περίπτωση μιας σύνθετης μονάδας, ο κόμβος θα αποτελεί ένα 'δοχείο', το οποίο μπορεί να περιέχει άλλες μονάδες στο εσωτερικό του. Φυσικά σαν μέλη διανύσματος σύνθετης μονάδας όλοι οι κόμβοι πρέπει να περιέχουν τον ίδιο ακριβώς αριθμό υπομονάδων στο εσωτερικό τους και τις ίδιες θύρες και συνδέσεις. Δεν είναι όμως απαραίτητο οι υπομονάδες αυτές να είναι οι ίδιες ακριβώς μονάδες, δηλαδή του ίδιου τύπου, αλλά να είναι παρόμοιου τύπου μονάδας, με τις ίδιες θύρες και συνδέσεις. Αυτό υποστηρίζεται από την αξιοποίηση του τύπου μονάδας σαν παράμετρο στις σύνθετες μονάδες. Έτσι, ορίζοντας την σύνθετη μονάδα κόμβου "Node", μας δίνεται η δυνατότητα να προσφέρουμε διαφορετικές λειτουργίες σε κάθε κόμβο, χωρίς αυτοί να είναι διαφορετικές μονάδες, αλλά χάρη σε διαφορετικές υπομονάδες τους, παρόμοιου τύπου μονάδας [§2.6.1.3.1].

Προκειμένου να θεωρούνται δύο μονάδες παρόμοιες με ένα τύπο μονάδας, πρέπει η NED δήλωσή τους να περιλαμβάνει τις ίδιες ακριβώς θύρες και όλες τις παραμέτρους που περιέχει ο τύπος αυτός. Είναι δηλαδή δυνατό να περιέχουν περισσότερες παραμέτρους όχι όμως και θύρες. Επομένως ορίζουμε τύπο απλής μονάδας “Node_Processor”, ο οποίος θα είναι και η παράμετρος τύπου μονάδας για την επιλογή μεταξύ αισθητήρα και πύλης. Επίσης ορίζουμε τύπο απλής μονάδας για τον αισθητήρα "sensor" και για την πύλη "gateway":

```
simple Node_Processor
    parameters:
      verbosity: numeric const,
      numNodes: numeric const;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/processor.ned]

Με τον ίδιο τρόπο μπορούμε να κάνουμε τον ορισμό των απλών μονάδων "sensor" και "gateway":

```
simple sensor
    parameters:
      verbosity: numeric const,
      numNodes: numeric const;
      msg_rng: bool;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/sensor.ned]

```
simple gateway
    parameters:
      verbosity: numeric const,
      numNodes: numeric const;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/gateway.ned]

Όπως φαίνεται ο επεξεργαστής “sensor” περιέχει στον ορισμό του τύπου της μονάδας του την επιπλέον παράμετρο “msg_rng” που ορίζει την τυχαιότητα στη δημιουργία μηνυμάτων. Αυτό επιτρέπεται από τον NED compiler [17]. Αυτό που δεν επιτρέπεται είναι να δηλώνουμε επιπλέον θύρες ή παραμέτρους σε υπομονάδες στον ορισμό της σύνθετης μονάδας που τις περιέχει, και εν προκειμένω του κόμβου "Node" .

Από εδώ και στο εξής όταν θα αναφερόμαστε στον 'επεξεργαστή' ενός κόμβου θα αναφερόμαστε στην υπομονάδα "gateway" / "sensor" που θα περιέχει η σύνθετη μονάδα κόμβου "Node", και που είναι παρόμοιου τύπου με τον τύπο μονάδας "Node_Processor".

Σε κάθε κόμβο θα υλοποιείται, μέσω της NED δήλωσης της σύνθετης μονάδας "Node", υπομονάδα "processor", δηλαδή επεξεργαστής. Ο επεξεργαστής κάθε κόμβου θα ορίζεται σύμφωνα με παράμετρο "nodeType" της σύνθετης μονάδας, η οποία θα περιέχει το όνομα τύπου απλής μονάδας, παρόμοιου τύπου με τον τύπο "Node_Processor". Αν το "nodeType" έχει τιμή "sensor", τότε ο επεξεργαστής του κόμβου θα είναι αισθητήρας. Αν έχει τιμή "gateway", τότε ο επεξεργαστής θα είναι πύλη.

Η επιλογή του ονόματος για τον τύπο υπομονάδας "processor" είναι επηρεασμένη από την προσέγγιση του θέματος των κόμβων στο Shawn. Ουσιαστικά η σχέση κόμβου - επεξεργαστή στην παρούσα υλοποίηση είναι ακριβώς ίδια με την λογική του Shawn στην δημιουργία κόμβων [§1.3.2 - §1.3.3].

Ορίζεται η σύνθετη μονάδα κόμβου "Node" ως εξής:

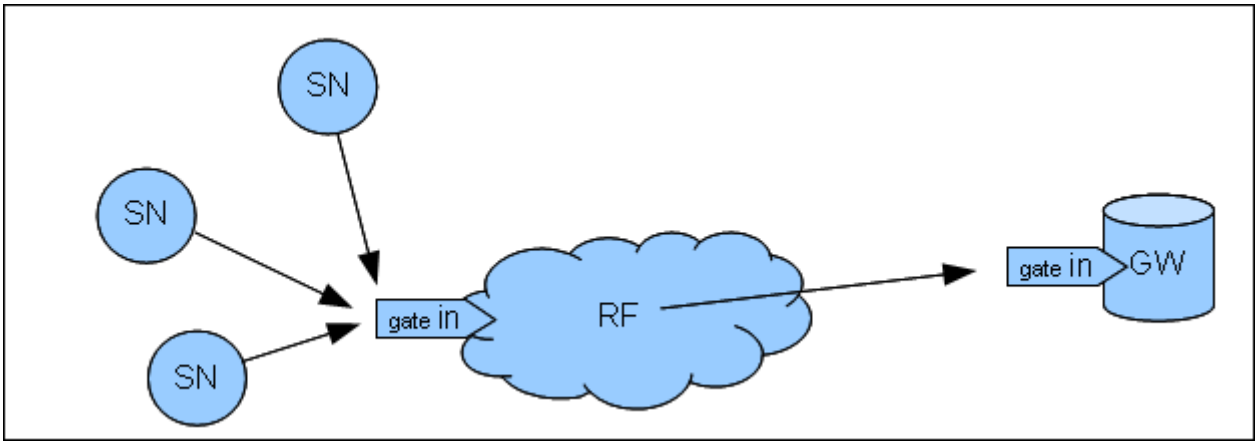
```
import "processor";

module Node
    parameters:
        verbosity: numeric const,
        numNodes: numeric const;
         nodeID: numeric const, // original external node ID
        nodeType: string, // node type -> Sensor/Gateway
        xCoor: numeric const, // X
        yCoor: numeric const, // Y
        zCoor: numeric const; // Z
    submodules:
         processor: nodeType like Node_Processor;
    parameters:
         verbosity = verbosity,
        numNodes = numNodes;
endmodule
```

[omnetpp-3.3p1/mysims/sim4/node.ned]

Είναι απαραίτητο να υπάρχει ο ορισμός του Node_Processor διαθέσιμος στη μονάδα Node, γι' αυτό και εισάγεται με την οδηγία 'import' πριν τη δήλωσή της. Οι παράμετροι της μονάδας είναι η αυθεντική 'εξωτερική' ταυτότητα του κόμβου, οι συντεταγμένες του και ο τύπος του επεξεργαστή που περιέχει ως υπομονάδα. Επιπλέον το επίπεδο αναφοράς (verbosity) και ο συνολικός αριθμός των κόμβων (numNodes).

Μετά τον ορισμό των παραμέτρων του κόμβου, ορίζεται η υπομονάδα επεξεργαστή που περιέχει, σύμφωνα με τη παράμετρο nodeType. Επιπλέον αποδίδονται τιμές στις παραμέτρους verbosity και numNodes του επεξεργαστή. Αυτές κληρονομούνται από τις αντίστοιχες παραμέτρους της γονικής σύνθετης μονάδας Node.

Όπως παρατηρεί κανείς εύκολα στον NED κώδικα των κόμβων δεν ορίζονται θύρες και συνδέσεις για τον επεξεργαστή τους. Οι θύρες εξυπηρετούν την ύπαρξη συνδέσεων ανάμεσα σε μονάδες. Ωστόσο είναι δυνατό να στέλνουμε μηνύματα χωρίς θύρα εξόδου, αλλά δεν γίνεται να λαμβάνουμε χωρίς θύρα εισόδου. Η θύρα εισόδου μπορεί να υπάρχει χωρίς να είναι συνδεδεμένη. Όλα αυτά υποστηρίζονται στο πλαίσιο της μεθόδου αποστολής μηνυμάτων s*endDirect(cMessage *msg, double delay, cGate *gate)* [17]. Με αυτή είναι δυνατό να στείλει μια μονάδα μήνυμα σε μια 'ανοιχτή' θύρα άλλης μονάδας απευθείας. Με αυτό τον τρόπο υλοποιείται η επικοινωνίας ανάμεσα στους αισθητήρες και τις πύλες. Παρεμβάλλοντας το μοντέλο επικοινωνίας ανάμεσά τους διατηρούμε αυτή τη μορφή επικοινωνίας. Η τακτική αναλύεται στην υλοποίηση των επεξεργαστών στη C++ [§6.4.1].



Σχήμα 6.1

Η σύνθετη μονάδα κόμβου "Node" παραμετροποιείται περαιτέρω κατά την δήλωσή της ως υπομονάδας στον NED ορισμό της σύνθετης μονάδας του δικτύου. Οι υπομονάδες "sensor" και "gateway" υλοποιούνται σε C++ κώδικα.

6.2.2 NED Ορισμός Μοντέλου Επικοινωνίας (RF_Propagation_Model)

Η επικοινωνία των κόμβων σε ένα ασύρματο δίκτυο σαν αυτά που εξετάζουμε (Καισαριανής, Πεντέλης) προτυποποιείται σε μεγάλη κλίμακα σύμφωνα με την ανάλυση της μετάδοσης ραδιοσυχνοτήτων του 4ου κεφαλαίου. Αυτή εξετάζει τις απώλειες στην ισχύ του εκπεμπόμενου σήματος ως αποτέλεσμα δύο κυρίως φαινομένων:

- Εξασθένιση Διαδρομής
- Εξασθένιση Βλάστησης

Για τις δύο αυτές πηγές εξασθένισης παρουσιάσθηκε πληθώρα μοντέλων, τα οποία μπορούν να συνδυαστούν με πολλούς τρόπους, ανάλογα με τις συγκεκριμένες συνθήκες επικοινωνίας και τα δεδομένα που έχουμε για αυτή. Είναι συνεπώς σημαντικό να επιτρέπουμε τον συνδυασμό αυτών κατά τρόπο που να τηρείται η λογική σχεδιασμού του Omnet++.

Σε αυτή τη βάση ορίζουμε το μοντέλο επικοινωνίας ως σύνθετη μονάδα με όνομα "RF_Propagation_Model". Η μονάδα αυτή θα περιέχει στο εσωτερικό της υπομονάδες για τα μοντέλα εξασθένισης διαδρομής και βλάστησης, όπως και επιπλέον υπομονάδες για την διαχείριση των αποτελεσμάτων αυτών και των μηνυμάτων των κόμβων. Όλες αυτές οι μονάδες θα ορίζονται με παράμετρο τύπου μονάδας, προκειμένου να ενσωματώσουμε όλες τις υλοποιήσεις σε κάθε επίπεδο λειτουργίας του μοντέλου επικοινωνίας. Οι παράμετροι των τύπων των υπομονάδων είναι:

(1) comm_in_module - παράμετρος τύπου Μονάδας Επικοινωνίας Εισερχομένων

(2) core_module - παράμετρος τύπου Μονάδας Κεντρικής Διαχείρισης

(3) fl_module - παράμετρος τύπου Μονάδας Υπολογισμού Εξασθένισης Βλάστησης

(4) pl_module - παράμετρος τύπου Μονάδας Υπολογισμού Εξασθένισης Διαδρομής

(5) comm_out_module - παράμετρος τύπου Μονάδας Επικοινωνίας Εξερχομένων

Ορίζονται αντίστοιχα με τις παραμέτρους και πέντε τύποι απλών μονάδων ως πρότυπα για τις υπομονάδες του "RF_Propagation_Model". Η κατάταξη συμφωνεί με τη σειρά που ακολουθεί ένα μήνυμα από την είσοδό του στο μοντέλο μέχρι την έξοδο από αυτό:

(1) Comm_IN_Module - πρότυπο τύπου Μονάδας Επικοινωνίας Εισερχομένων

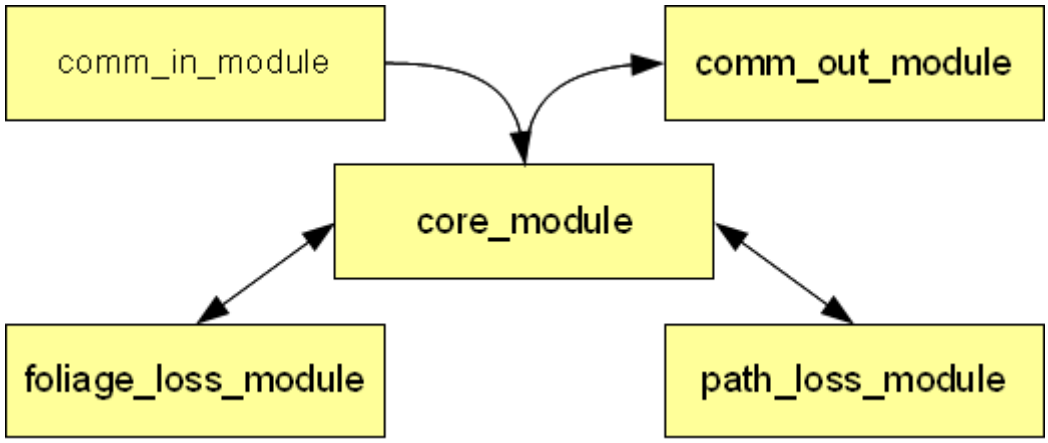
(2) Core_Module - πρότυπο τύπου Μονάδας Κεντρικής Διαχείρισης

(3) Foliage_Loss_Module - πρότυπο τύπου Μονάδας Υπολογισμού Εξασθένισης Βλάστησης

(4) Path_Loss_Module - πρότυπο τύπου Μονάδας Υπολογισμού Εξασθένισης Διαδρομής

(5) Comm_OUT_Module - πρότυπο τύπου Μονάδας Επικοινωνίας Εξερχομένων

Οι τύποι μονάδων του RF μοντέλου συνδέονται μεταξύ τους σύμφωνα με το παρακάτω διάγραμμα:



Σχήμα 6.2

Οι συνδέσεις αυτές ορίζονται στον τομέα 'connections:', στη δήλωση της γονικής σύνθετης μονάδας "RF_Propagation_Model". Αυτή δεν συνδέεται με τις υπομονάδες της, παρά μόνο τις περιέχει και ορίζει τις συνδέσεις και τις παραμέτρους τους.

Η εκπομπή ενός κόμβου υλοποιείται στο Omnet++ με την αποστολή μηνυμάτων. Όταν ένας κόμβος στέλνει ένα μήνυμα, αυτό περνάει πρώτα από το μοντέλο μετάδοσης ραδιοσυχνοτήτων, το οποίο αφού το επεξεργαστεί στέλνει αντίγραφά του στους πιθανούς παραλήπτες. Η διαδικασία σε βήματα είναι η εξής:

1. Ο κόμβος 'x' (Node[x]) στέλνει ένα μήνυμα στο μοντέλο επικοινωνίας (RF_Propagation_Model). Ουσιαστικά ο επεξεργαστής "sensor" στέλνει το μήνυμα στην υπομονάδα comm_in_module της σύνθετης μονάδας RF_Propagation_Model. Το μήνυμα περιέχει μόνο δύο παραμέτρους:

   srcAddress_internal: η διεύθυνση του κόμβου 'x' σε τιμές εσωτερικής οργάνωσης
   srcAddress_external: η ταυτότητα του κόμβου σύμφωνα με την δική μας αρίθμηση

2. Η μονάδα comm_in_module αφού λάβει ένα μήνυμα εξετάζει πόσοι κόμβοι είναι πιθανοί παραλήπτες του. Αυτό συνεπάγεται να εντοπίσει ποιοι κόμβοι είναι πύλες, δηλαδή έχουν επεξεργαστή "gateway". Για κάθε κόμβο (Node) 'y', με επεξεργαστή πύλη (gateway), δημιουργείται ένα αντίγραφό του μηνύματος και προστίθενται σε αυτό παράμετροι με την διεύθυνση του παραλήπτη και την απόστασή του από τον αποστολέα:
   destAddress_internal: η διεύθυνση του κόμβου 'y' παραλήπτη, σε τιμές εσωτερικής οργάνωσης
   destAddress_external: η ταυτότητα του κόμβου 'y' σύμφωνα με την δική μας αρίθμηση
   distance_m: η απόσταση ανάμεσα στον αποστολέα και τον παραλήπτη (x, y) σε μέτρα

   Επιπλέον σε κάθε αντίγραφο προστίθεται και η ισχύς εκπομπής του μηνύματος σε dBm ("transmitPower_dBm"), και όλα τα αντίγραφα στέλνονται στο core_module. Το αυθεντικό μήνυμα καταστρέφεται (delete).

3. Το core_module στέλνει τα μηνύματα στο fl_module

4. Το fl_module προσθέτει παράμετρο ("mean_foliage_loss_dBm") σε κάθε μήνυμα που λαμβάνει. Σε αυτήν αποδίδεται η μέση τιμή της εξασθένισης λόγω βλάστησης σε dBm. Κάθε μήνυμα στέλνεται πίσω στο core_module.

5. Το core_module στέλνει τα μηνύματα στο pl_module

6. Το fl_module προσθέτει παράμετρο ("mean_path_loss_dBm") σε κάθε μήνυμα που λαμβάνει. Σε αυτήν αποδίδεται η μέση τιμή της εξασθένισης διαδρομής σε dBm. Κάθε μήνυμα στέλνεται πίσω στο core_module.

7. Το core_module αφού κάνει έλεγχο για την ύπαρξη των παραμέτρων εξασθένισης ισχύος, ισχύος εκπομπής και απόστασης, στέλνει τα μηνύματα στο comm_out_module.

8. Το comm_out_module εξετάζει για τις παραμέτρους κάθε μηνύματος για την ισχύ και την εξασθένιση της εκπομπής, αν μπορεί να το λάβει ο παραλήπτης του. Σε περίπτωση που μπορεί τότε στέλνει το μήνυμα και κλείνει ο κύκλος της αποστολής επιτυχώς. Αν όχι τότε το μήνυμα καταστρέφεται.

9. Αν στον κόμβο 'y' (Node[y]) που έχει επεξεργαστή πύλη (gateway) φτάσει μήνυμα από την υπομονάδα comm_out_module του RF_Propagation_Model, ο επεξεργαστής ενημερώνει την λίστα των γειτόνων του και τα στατιστικά ληφθέντων μηνυμάτων, και τέλος καταστρέφει το μήνυμα.

Κάθε τύπος υπομονάδας του RF μοντέλου υλοποιείται με αρκετές διαφορετικές εκφράσεις, σύμφωνα με την ανάλυση της μετάδοσης ραδιοσυχνοτήτων [Κεφ 4]. Ακολουθεί η παρουσίαση της NED υλοποίηση των πρότυπων τύπων μονάδων, και οι διαφορετικές υλοποιήσεις που εφαρμόστηκαν για κάθε παράμετρο τύπου μονάδας του RF_Propagation_Model.

6.2.2.1 NED Ορισμός Υπομονάδων Μοντέλου Επικοινωνίας

Για όλες τις υπομονάδες ορίζεται παράμετρος “verbosity” που καθορίζει το επίπεδο αναφοράς για τις μεθόδους της μονάδας. Οι υπόλοιπες παράμετροι και οι θύρες των μονάδων διαφέρουν σε κάθε υπομονάδα, με αρκετές πάντως παραμέτρους να ορίζονται σε περισσότερες από μία.

6.2.2.1.1 Μονάδα Επικοινωνίας Εισερχομένων comm_in_module

Η μονάδα comm_in_module είναι υπεύθυνη για την παραλαβή μηνυμάτων από τους κόμβους. Όταν λάβει ένα μήνυμα δημιουργεί αντίγραφα για κάθε πιθανό παραλήπτη τους και τα προωθεί στο core_module. Έτσι θέλουμε να έχει τουλάχιστον μία θύρα εισόδου και μία εξόδου:

```
comm_in_module <-- sensor        : θύρα εισόδου “fromNetwork”
comm_in_module --> core_module   : θύρα εξόδου “toCore”
```

Η θύρα εξόδου ορίζεται στην NED δήλωση με τίτλο “toCore” (προς το core_module), ενώ η θύρα εισόδου με τίτλο “fromNetwork” (από το δίκτυο). Στη μονάδα πραγματοποιούμε επιπλέον τις μετατροπές μονάδων ισχύος εισερχόμενου σήματος σε dBm και παράγουμε αντίγραφα κάθε εισερχόμενου μηνύματος για κάθε πιθανό παραλήπτη βάσει τύπου κόμβου. Πιθανοί παραλήπτες είναι όλα τα gateway. Ορίζονται συνεπώς και τρεις παράμετροι:

transmitPower : ισχύς εκπομπής
transmitPowerUnit : μονάδα ισχύος εκπομπής
numNodes : αριθμός κόμβων δικτύου

Ο NED ορισμός της πρότυπης απλής μονάδας “Comm_IN_Module” είναι:

```
simple Comm_IN_Module // MODULE TYPE
    parameters:
        verbosity: numeric const,
        transmitPower: numeric const,
        transmitPowerUnit: string,
        numNodes: numeric const;
    gates:
        in: fromNetwork;
        out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/comm_in.ned]

Στην προσομοίωση που παρουσιάζεται έχει υλοποιηθεί μόνο μία μονάδα εισόδου, η comm_in1. Ο NED ορισμός της είναι:

```
simple comm_in1
    parameters:
        verbosity: numeric const,
        transmitPower: numeric const,
        transmitPowerUnit: string,
        numNodes: numeric const;
    gates:
        in: fromNetwork;
        out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/comm_in1.ned]

6.2.2.1.2 Μονάδα Κεντρικής Διαχείρισης (core_module)

Η μονάδα core_module αφού παραλάβει ένα μήνυμα από τη μονάδα comm_in_module, αναλαμβάνει τη διανομή του στις υπόλοιπες:

- Πρώτα στέλνει το μήνυμα στη μονάδα fl_module
- Αφού επιστραφεί το προωθεί στο pl_module
- Αφού επιστραφεί για δεύτερη φορά ελέγχει τα περιεχόμενά του και το στέλνει στη μονάδα comm_out_module για προώθηση πίσω στο δίκτυο

Συνολικά χρειάζονται έξι θύρες για να υλοποιηθούν οι συνδέσεις του core_module με τις υπόλοιπες μονάδες:

```
core_module <-- comm_in_module  : θύρα εισόδου fromCommIn
core_module --> fl_module       : θύρα εξόδου toFL
core_module <-- fl_module       : θύρα εισόδου fromFL
core_module --> pl_module       : θύρα εξόδου toPL
core_module <-- pl_module       : θύρα εισόδου fromPL
core_module --> comm_out_module : θύρα εξόδου toCommOut
```

Ο NED ορισμός του προτύπου απλής μονάδας Κεντρικής Διαχείρισης “Core_Module” είναι:

```
simple Core_Module // MODULE TYPE
parameters:
        verbosity: numeric const;
    gates:
        in: fromPL, fromFL, fromCommIn;
        out: toPL, toFL, toCommOut;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/core.ned]

Στην εφαρμογή δημιουργήθηκε μόνο μία μονάδα κεντρικής διαχείρισης η core1. Ο NED ορισμός της είναι:

```
simple core1
parameters:
        verbosity: numeric const;
    gates:
        in: fromPL, fromFL, fromCommIn;
        out: toPL, toFL, toCommOut;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/core1.ned]

6.2.2.1.3 Μονάδα Εξασθένισης Βλάστησης (fl_module)

Η μονάδα fl_module αφού λάβει ένα μήνυμα από την κεντρική διαχείριση (core_module) υπολογίζει την εξασθένιση βλάστησης, την ενσωματώνει στο μήνυμα και το στέλνει πίσω στη διαχείριση. Χρειάζεται συνεπώς δύο θύρες, μία εισόδου και μία εξόδου, από και προς το core_module αντίστοιχα:

```
fl_module <-- core_module    : θύρα εισόδου fromCore
fl_module --> core_module    : θύρα εξόδου toCore
```

Επιπλέον, δεδομένου τα μοντέλα εξασθένισης που εξετάσαμε στην ανάλυση μετάδοσης ραδιοσυχνοτήτων [Κεφ. 4] χρησιμοποιούν ως παραμέτρους την συχνότητα και το εύρος της βλάστησης, εισάγουμε τις εξής παραμέτρους στην NED δήλωση:

```
   frequency_GHz : η συχνότητα σε GHz
foliagePathFactor : ο παράγοντας εύρους της βλάστησης
```

Ο παράγοντας εύρους της βλάστησης πολλαπλασιάζεται με το μήκος της διαδρομής για να δώσει το εύρος της βλάστησης βάσει του οποίου υπολογίζουμε την εξασθένιση βλάστησης.

Ο ορισμός του προτύπου απλής μονάδας εξασθένισης βλάστησης “Foliage_Loss_Module” είναι:

```
simple Foliage_Loss_Module // MODULE TYPE
    parameters:
        verbosity: numeric const,
        frequency_GHz: numeric,
        foliagePathFactor: numeric;
    gates:
        in: fromCore;
        out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/foliage_loss.ned]

Στην υλοποίηση του Omnet δημιουργήθηκαν τύποι μονάδων και για τα τέσσερα μοντέλα εξασθένισης βλάστησης που εξετάστηκαν [§4.1.3.1]. Αυτοί και τα αντίστοιχα μοντέλα που εφαρμόζουν είναι:

- cost235_in_leaf - COST235 μοντέλο εξασθένισης βλάστησης, για βλάστηση με φύλλωμα
- cost235_out_of_leaf - COST235 μοντέλο εξασθένισης βλάστησης, για βλάστηση χωρίς φύλλωμα
- itu_r - πρώιμο μοντέλο εξασθένισης βλάστησης ITU
- weissberger - μοντέλο εξασθένισης βλάστησης του Weissberger

Επειδή είναι υποχρεωτικό πάντα να φορτώνουμε μονάδα εξασθένισης βλάστησης στο RF_Propagation_Model, δημιουργήθηκε και μια μονάδα που αποδίδει πάντα μηδενική εξασθένιση με τίτλο no_foliage. Αν συνεπώς δεν θέλουμε να υπολογίσουμε εξασθένισης βλάστησης χρησιμοποιούμε αυτή τη μονάδα. Συνολικά ορίζουμε σε C++ κώδικα πέντε μονάδες βλάστησης.

Οι ορισμοί των μονάδων υπολογισμού εξασθένισης βλάστησης στη NED γλώσσα είναι:

```
simple cost235_in_leaf
    parameters:
        verbosity: numeric const,
        frequency_GHz: numeric,
        foliagePathFactor: numeric;
    gates:
      in: fromCore;
     out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/cost235_in_leaf.ned]

```
simple cost235_out_of_leaf
    parameters:
        verbosity: numeric const,
        frequency_GHz: numeric,
        foliagePathFactor: numeric;
    gates:
      in: fromCore;
     out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/cost235_out_of_leaf.ned]

```
simple itu_r
    parameters:
        verbosity: numeric const,
        frequency_GHz: numeric,
        foliagePathFactor: numeric;
    gates:
      in: fromCore;
     out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/itu_r.ned]

```
simple weissberger
    parameters:
        verbosity: numeric const,
        frequency_GHz: numeric,
        foliagePathFactor: numeric;
    gates:
      in: fromCore;
     out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/weissberger.ned]

```
simple no_foliage
    parameters:
        verbosity: numeric const,
        frequency_GHz: numeric,
        foliagePathFactor: numeric;
    gates:
      in: fromCore;
     out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/no_foliage.ned]

6.2.2.1.4 Μονάδα Εξασθένισης Διαδρομής (pl_module)

Η μονάδα pl_module είναι σχεδόν ίδια με την fl_module, τόσο στην NED δήλωση, όσο και στην C++ υλοποίηση [§6.2.2.1.3]. Όταν λαμβάνει ένα μήνυμα από την κεντρική διαχείριση (core_module) υπολογίζει την εξασθένιση διαδρομής, την ενσωματώνει στο μήνυμα και το στέλνει πίσω στη διαχείριση. Χρειάζεται συνεπώς δύο θύρες, μία εισόδου και μία εξόδου, από και προς το core_module αντίστοιχα:

```
pl_module <-- core_module   : θύρα εισόδου fromCore
pl_module --> core_module   : θύρα εξόδου toCore
```

Επιπλέον, τα μοντέλα εξασθένισης που εξετάσαμε στην ανάλυση μετάδοσης ραδιοσυχνοτήτων [§4.1.2, §4.1.3.2] χρησιμοποιούν διάφορες παραμέτρους. Κοινές για όλα τα μοντέλα είναι η απόσταση των κόμβων, η οποία υπολογίζεται στο comm_in_module και υπάρχει στη παράμετρο "distance_m" κάθε μηνύματος, και η συχνότητα. Άλλα όμως χρησιμοποιούν το ύψος των κεραιών και άλλα όχι. Δεν διαχωρίζουμε αυτές τις διαφορές και εισάγουμε στην NED δήλωση όλες τις παραμέτρους που είναι απαραίτητες για να εφαρμόσει κανείς όλα τα μοντέλα. Αυτές είναι:

frequency_GHz: η συχνότητα σε GHz
transmitterAntennaGain_dB: το κέρδος της κεραίας του πομπού σε dB
receiverAntennaGain_dB: το κέρδος της κεραίας του δέκτη σε dB
receiverHeight_m: το ύψος της κεραίας του δέκτη σε μέτρα
transmitterHeight_m: το ύψος της κεραίας του πομπού σε μέτρα

Εν τέλει ο ορισμός του προτύπου απλής μονάδας Εξασθένισης Διαδρομής “Path_Loss_Module” είναι:

```
simple Path_Loss_Module // MODULE TYPE
    parameters:
        verbosity: numeric const,
        frequency_GHz: numeric,
        transmitterAntennaGain_dB: numeric,
        receiverAntennaGain_dB: numeric,
        receiverHeight_m: numeric,
        transmitterHeight_m: numeric;
    gates:
        in: fromCore;
        out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/path_loss.ned]

Στην υλοποίηση του Omnet++ δημιουργήθηκαν τύποι μονάδων και για τα τέσσερα μοντέλα εξασθένισης διαδρομής που εξετάστηκαν [§4.1.2, §4.1.3.2]. Αυτοί και τα αντίστοιχα μοντέλα που εφαρμόζουν είναι:

- free_space - μοντέλο Εξασθένισης Διαδρομής Ελεύθερου Χώρου
- log_distance - μοντέλο Εξασθένισης Διαδρομής Λογαριθμικής Απόστασης
- egli - μοντέλο Εξασθένισης Αναγλύφου του Egli
- multipath_analysis - μοντέλο Εξασθένισης Ανακλάσεων Δύο Δρόμων (αναλυτική σχέση)
- multipath_approximation - μοντέλο Εξασθένισης Ανακλάσεων Δύο Δρόμων (προσεγγιστική σχέση)

Προέκυψαν πέντε τύποι για τέσσερα μοντέλα, διότι όπως και στο Shawn έτσι και εδώ υλοποιούμε το μοντέλο Ανακλάσεων Δύο Δρόμων (Multipath 2-Ray Model) και με τον προσεγγιστικό τύπο (multipath_approximation) και με τον αναλυτικό (multipath_analysis). Σε αντίθεση με το μοντέλο εξασθένισης βλάστησης, εδώ δεν υπάρχει ενδεχόμενο μηδενικών απωλειών, οπότε και δεν δημιουργείται σχετική μονάδα.

Οι δηλώσεις στη NED γλώσσα για τις μονάδες αυτές είναι:

```
simple free_space
    parameters:
        verbosity: numeric const,
        frequency_GHz: numeric,
        transmitterAntennaGain_dB: numeric,
        receiverAntennaGain_dB: numeric,
        receiverHeight_m: numeric,
        transmitterHeight_m: numeric;
    gates:
        in: fromCore;
        out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/free_space.ned]

```
simple log_distance
    parameters:
        verbosity: numeric const,
        frequency_GHz: numeric,
        transmitterAntennaGain_dB: numeric,
        receiverAntennaGain_dB: numeric,
        receiverHeight_m: numeric,
        transmitterHeight_m: numeric,
        log_distance_exponent: numeric;
    gates:
        in: fromCore;
        out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/log_distance.ned]

```
simple egli
    parameters:
        verbosity: numeric const,
        frequency_GHz: numeric,
        transmitterAntennaGain_dB: numeric,
        receiverAntennaGain_dB: numeric,
        receiverHeight_m: numeric,
        transmitterHeight_m: numeric;
    gates:
        in: fromCore;
       out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/egli.ned]

```
simple multipath_analysis
    parameters:
        verbosity: numeric const,
        frequency_GHz: numeric,
        transmitterAntennaGain_dB: numeric,
        receiverAntennaGain_dB: numeric,
        receiverHeight_m: numeric,
        transmitterHeight_m: numeric;
    gates:
        in: fromCore;
       out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/multipath_analysis.ned]

```
simple multipath_approximation
    parameters:
        verbosity: numeric const,
        frequency_GHz: numeric,
        transmitterAntennaGain_dB: numeric,
        receiverAntennaGain_dB: numeric,
        receiverHeight_m: numeric,
        transmitterHeight_m: numeric;
    gates:
        in: fromCore;
       out: toCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/multipath_approximation.ned]

Στην παραπάνω δήλωση ξεχωρίζει ο log_distance τύπος μονάδας. Σε αυτόν υπάρχει επιπλέον παράμετρος "log_distance_exponent", που αντιστοιχεί στον εκθετικό παράγοντα του μοντέλου Εξασθένισης Λογαριθμικής Απόστασης. Η παράμετρος αυτή ορίζεται απευθείας για τη μονάδα στο ini αρχείο ρυθμίσεων.

6.2.2.1.5 Μονάδα Επικοινωνίας Εξερχομένων (comm_out_module)

Η μονάδα comm_out_module είναι υπεύθυνη για την αποστολή των μηνυμάτων στους κόμβους, αφού έχουν προστεθεί σε αυτούς οι παράμετροι εξασθένισης. Όταν λάβει ένα μήνυμα η μονάδα εξετάζει με τις παραμέτρους αυτές αν μπορεί να το λάβει ο παραλήπτης. Αν μπορεί τότε το στέλνει στον επεξεργαστή gateway του κόμβου του παραλήπτη, αλλιώς το καταστρέφει. Χρειάζεται συνεπώς δύο θύρες, μία εισόδου και μία εξόδου:

```
comm_out_module <-- core_module : θύρα εισόδου fromCore
comm_in_module --> gateway      : θύρα εξόδου toNetwork
```

Ο NED ορισμός του προτύπου της απλής μονάδας εξόδου “Comm_OUT_Module” είναι:

```
simple Comm_OUT_Module // MODULE TYPE
    parameters:
       verbosity: numeric const,
       receiverSensitivity_dBm: numeric,
       std_dev_dBm: numeric const,
       foliage: bool;
    gates:
        in: fromCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/comm_out.ned]

Για την έξοδο έχουν δημιουργηθεί δύο τύποι μονάδας:

- comm_out_rng
- comm_out_probability

Η πρώτη μονάδα δημιουργεί τυχαία δείγματα γύρω από τη μέση τιμή της εξασθένισης για μια δεδομένη τυπική απόκλιση και συγκρίνει τη τιμή τους με την ευαισθησίας του δέκτη για κάθε παραλήπτη. Η δεύτερη συγκρίνει μια δεδομένη πιθανότητα συνδεσιμότητας με την πιθανότητα η λαμβανόμενη ισχύς στο δέκτη να είναι μεγαλύτερη της ευαισθησίας του, για τη μέση τιμή της ισχύος και μια δεδομένη τυπική απόκλιση. Η πιθανότητα-κριτήριο της μονάδας είναι και επιπρόσθετη παράμετρος σε αυτή. Οι μέθοδοι αναλύονται ευρύτερα στην C++ υλοποίησή τους [[§6.4.2.5]. Για την εφαρμογή τους ορίζονται από κοινού οι εξής παράμετροι:

receiverSensitivity_dBm: ευαισθησία του δέκτη σε dBm
std_dev_dBm: τυπική απόκλιση σε dBm
foliage: bool που καθορίζει τον συνυπολογισμό της απώλειας βλάστησης

Η μονάδα comm_out_probability δέχεται και την επιπλέον παράμετρο της πιθανότητας επικοινωνίας εισόδου, η οποία συγκρίνεται με την υπολογισμού για να δώσει αποδοχή ή όχι της επικοινωνίας:

communicationProbability: πιθανότητα συνδεσιμότητας εισόδου

Οι NED ορισμοί των μονάδων αυτών είναι:

```
simple comm_out_rng
 parameters:
       verbosity: numeric const,
       receiverSensitivity_dBm: numeric,
       std_dev_dBm: numeric const,
       foliage: bool;
    gates:
      in: fromCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/comm_out_rng.ned]

```
simple comm_out_probability
 parameters:
       verbosity: numeric const,
       receiverSensitivity_dBm: numeric,
       std_dev_dBm: numeric const,
       foliage: bool;
       communicationProbability: numeric,
    gates:
      in: fromCore;
endsimple
```

[omnetpp-3.3p1/mysims/sim4/comm_out_probability.ned]

6.2.2.2 NED Ορισμός Σύνθετης Μονάδας RF_Propagation_Model

Εφόσον έχουν ορισθεί τα πρότυπα των τύπων μονάδων που θα έχει ως υπομονάδες η σύνθετη μονάδα επικοινωνίας RF_Propagation_Model, είναι δυνατό πλέον να γίνει και ο δικός της ορισμός. Αυτός περιλαμβάνει τα εξής μέρη:

1 - Παράμετροι Σύνθετης Μονάδας
2 - Υπομονάδες & Απόδοση τιμών στις παραμέτρους τους
3 - Συνδέσεις Υπομονάδων

Πριν από τα βήματα αυτά προέχει να εισάγουμε τους ορισμούς των προτύπων υπομονάδων στη δήλωση με την οδηγία import:

```
import "comm_in"
       "comm_out"
       "core",
       "path_loss",
       "foliage_loss";
```

[omnetpp-3.3p1/mysims/sim4/rf_model.ned]

6.2.2.2.1 Παράμετροι Σύνθετης Μονάδας RF_Propagation_Model

Οι παράμετροι της σύνθετης μονάδας βρίσκονται στον τομέα "parameters:". Αυτές που πρέπει καταρχήν να περιλαμβάνονται είναι όλες οι παράμετροι που χρησιμοποιούνται στα μοντέλα εξασθένισης και στους βοηθητικούς υπολογισμούς που υλοποιούν οι υπόλοιπες υπομονάδες του μοντέλου επικοινωνίας. Αυτές είναι:

```
parameters:
        useRF: numeric const,
        frequency_GHz: numeric,
        transmitPower: numeric,
        transmitPowerUnit: string,
        receiverSensitivity_dBm: numeric,
        transmitterAntennaGain_dB: numeric,
        receiverAntennaGain_dB: numeric,
        foliage: numeric,
        foliagePathFactor: numeric,
        receiverHeight_m: numeric,
        transmitterHeight_m: numeric;
```

[omnetpp-3.3p1/mysims/sim4/rf_model.ned]

Πλέον αυτών σημαντικό είναι να δηλώσουμε τις παραμέτρους για τους τύπους μονάδων που θα αποτελούν τις υπομονάδες του μοντέλου [§2.6.1.3.1]:

```
parameters:
     //...
     core_ModuleType: string,
     path_loss_ModuleType: string,
     foliage_loss_ModuleType: string,
     comm_in_ModuleType: string,
     comm_out_ModuleType: string,
```

[omnetpp-3.3p1/mysims/sim4/rf_model.ned]

Τέλος κρίνεται χρήσιμο να έχουμε και τον αριθμό των κόμβων του δικτύου, που αξιοποιεί η μονάδα εισερχόμενης επικοινωνίας, αλλά και η μονάδα υπολογισμού Εξασθένισης Διαδρομής Λογαριθμικής Απόστασης. Όπως και σε όλες τις μονάδες υπάρχει και μια παράμετρος verbosity για το επίπεδο αναφορών:

```
parameters:
     //...
     verbosity: numeric const,
     numNodes: numeric const,
```

[omnetpp-3.3p1/mysims/sim4/rf_model.ned]

Οι παράμετροι του μοντέλου, οι σχετικές με την επικοινωνία, αποδίδονται ευθέως σε αυτό στο αρχείο omnetpp.ini, ή σε άλλο σχετικό, και όχι μέσω του ορισμού του δικτύου. Μόνο οι numNodes και verbosity κληρονομούνται από το δίκτυο.

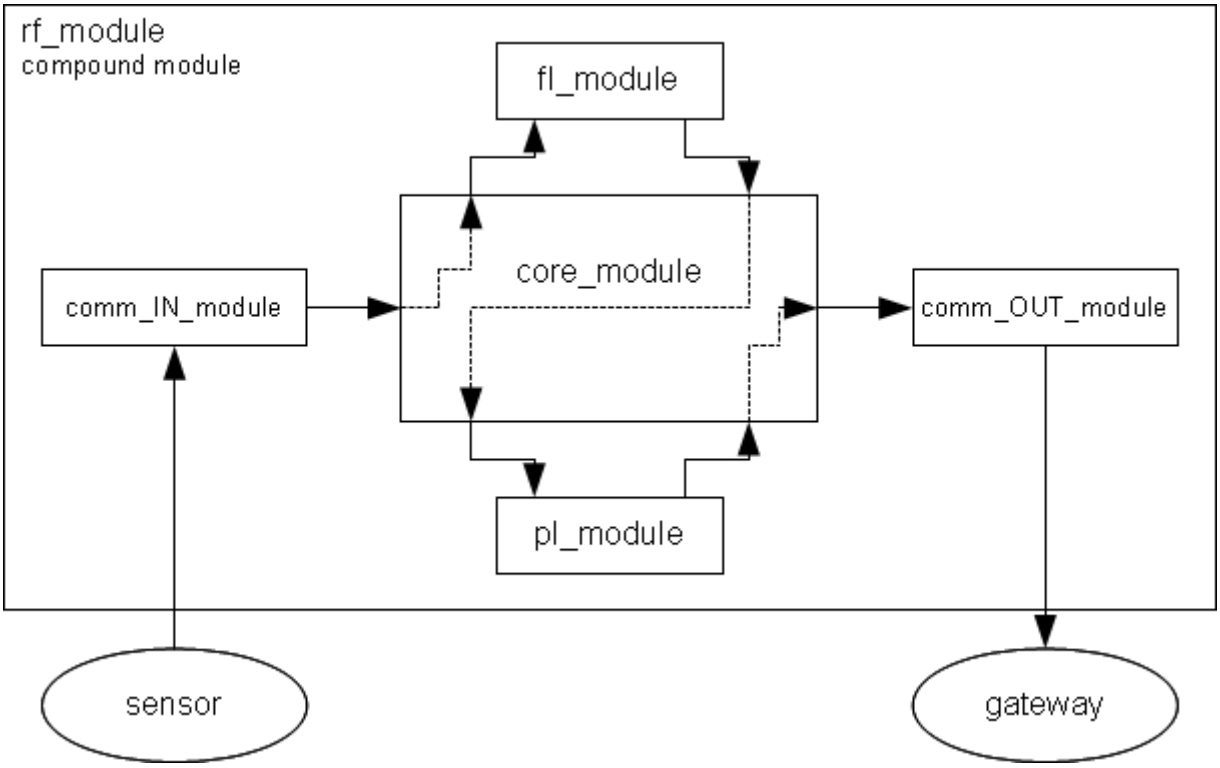
6.2.2.2.2 Υπομονάδες & Απόδοση τιμών στις παραμέτρους τους

Αμέσως μετά τον ορισμό των παραμέτρων της σύνθετης μονάδας RF_Propagation_Model, ορίζονται οι υπομονάδες της. Αυτό γίνεται στον τομέα submodules, και ορίζονται όπως αναφέρθηκε στην αρχή της ανάλυσης με παράμετρο τύπου μονάδας, όπως αυτός δηλώθηκε στον προηγούμενο τομέα των παραμέτρων. Σε κάθε υπομονάδα αποδίδεται ως τιμή των παραμέτρων της, η αντίστοιχη παράμετρος της σύνθετης μονάδας του μοντέλου, υλοποιώντας έτσι κληρονομικότητα. Με αυτό το τρόπο δεν χρειάζεται να δηλώσουμε ευθέως τιμή για τις παραμέτρους των υπομονάδων σε κάποιο ini αρχείο, ή στην εκκίνηση της προσομοίωσης. Σημασία έχει να δηλώσουμε τιμές μόνο για το μοντέλο, δηλαδή τη σύνθετη μονάδα RF_Propagation_Model.

Επιπλέον μετά την απόδοση των παραμέτρων, γίνεται και δήλωση γραφικής απεικόνισης για κάθε υπομονάδα. Η δήλωση περιλαμβάνει θέση και εικόνα αναπαράστασης. Οι εικόνες που χρησιμοποιήθηκαν βρίσκονται στον υποφάκελο 'omnetpp-3.3p1/bitmaps/' και σε κατώτερους φακέλους. Οι θέσεις ορίστηκαν με αποκλειστικό κριτήριο την αποδοτική απεικόνιση και χωρίς κάποια φυσική σημασία. Για τις εικόνες έγινε προσπάθεια να είναι σχετικές με την μονάδα που αναπαριστούν. Η δήλωση των υπομονάδων είναι [§2.6.1.3]:

```
submodules:
   core_module: core_ModuleType like Core_Module
     parameters:
        verbosity = verbosity;
     display: "p=150,150;i=abstract/router_l";
   pl_module: path_loss_ModuleType like Path_Loss_Module
     parameters:
        verbosity = verbosity,
        frequency_GHz = frequency_GHz,
        transmitterAntennaGain_dB = transmitterAntennaGain_dB,
        receiverAntennaGain_dB = receiverAntennaGain_dB,
        receiverHeight_m = receiverHeight_m,
        transmitterHeight_m = transmitterHeight_m;
     display: "p=250,250;i=block/cogwheel,blue";
   fl_module: foliage_loss_ModuleType like Foliage_Loss_Module
     parameters:
        verbosity = verbosity,
        frequency_GHz = frequency_GHz,
        foliagePathFactor = foliagePathFactor;
     display: "p=50,250;i=block/cogwheel,green";
   comm_in_module: comm_in_ModuleType like Comm_IN_Module
     parameters:
        verbosity = verbosity,
        transmitPower = transmitPower,
        transmitPowerUnit = transmitPowerUnit,
        numNodes = numNodes;
     display: "p=50,50;i=block/arrival";
   comm_out_module: comm_out_ModuleType like Comm_OUT_Module
     parameters:
        verbosity = verbosity;
        receiverSensitivity_dBm: numeric,
        std_dev_dBm: numeric const,
        foliage: bool;
     display: "p=250,50;i=block/departure";
```

[omnetpp-3.3p1/mysims/sim4/rf_model.ned]



Σχήμα 6.3

Στο παραπάνω σχήμα φαίνεται η διάταξη των υπομονάδων του RF_Propagation_Model, καθώς και οι συνδέσεις του και η πορεία που ακολουθεί ένα μήνυμα από έναν αισθητήρα ως μια πύλη, διαμέσου του μοντέλου επικοινωνίας. Στο επόμενο κεφάλαιο αναλύονται οι συνδέσεις του μοντέλου και οι θύρες που τις υποστηρίζουν

6.2.2.2.3 Συνδέσεις Υπομονάδων

Οι υπομονάδες του μοντέλου επικοινωνίας συνδέονται μόνο μεταξύ τους και όχι με την σύνθετη μονάδα του μοντέλου. Αυτή αποτελεί απλό 'δοχείο' για τις υπομονάδες που υλοποιούν τις λειτουργίες του. Οι συνδέσεις σύμφωνα με τις θύρες που παρουσιάστηκαν στην ανάλυση των υπομονάδων είναι:

```
    connections nocheck:
        core_module.toPL --> pl_module.fromCore;
        core_module.fromPL <-- pl_module.toCore;

        core_module.toFL --> fl_module.fromCore;
        core_module.fromFL <-- fl_module.toCore;

        core_module.toCommOut --> comm_out_module.fromCore;
        core_module.fromCommIn <-- comm_in_module.toCore;
```

[omnetpp-3.3p1/mysims/sim4/rf_model.ned]

Στο Σχήμα 6.3 αναπαρίστανται χαρακτηριστικά οι συνδέσεις αυτές, διαγράφοντας την πορεία ενός μηνύματος μέσα στο μοντέλο επικοινωνίας.

Η θύρα comm_in_module.fromNetwork δεν είναι συνδεδεμένη με καμία θύρα, ούτε εντός ούτε εκτός της σύνθετης μονάδας RF_Propagation_Model. Η δήλωση 'nocheck' [§2.6.1.4.2] στο όνομα του τομέα "connections:" μας το επιτρέπει αυτό, και είναι σημαντικό σχετικά με τον τρόπο που υλοποιείται η επικοινωνία ανάμεσα στο μοντέλο και τους κόμβους. Στον ορισμό τον μονάδων στη C++ γίνεται πλήρης ανάλυση [§6.4].

Τέλος μπορούμε να ορίσουμε την γραφική παράσταση της σύνθετης μονάδας στο τέλος της NED δήλωσης, χωρίς να είναι απαραίτητο μιας και αυτό μπορεί να γίνει στον ορισμό του δικτύου:

```
display: "p=400,300;b=20,20,oval"; // RF model display
```

[omnetpp-3.3p1/mysims/sim4/rf_model.ned]

Όπου "p=400,300" δηλώνει την θέση του κέντρου της απεικόνισης σε εικονοστοιχεία (pixels), και "b=20,20,oval" δηλώνει ότι η μονάδα παριστάνεται στη θέση αυτή με ένα κύκλο διαμέτρου 20 εικονοστοιχείων.

6.2.3 NED Ορισμός Σύνθετης Μονάδας Δικτύου

Εφόσον υπάρχουν οι ορισμοί στην NED γλώσσα των κόμβων και του μοντέλου επικοινωνίας, είναι πλέον δυνατό να ορισθεί και το δίκτυο, ως σύνθετη μονάδα με υπομονάδες του κόμβους και το μοντέλο επικοινωνίας. Ως εκ τούτου ο ορισμός του ξεκινάει με την οδηγία εισαγωγής των ορισμών τους:

```
import "node",
       "rf_model";
```

[omnetpp-3.3p1/mysims/sim4/sim4.ned]

Ο ορισμός του δικτύου, όπως και κάθε σύνθετης μονάδας, χωρίζεται στα εξής στάδια:

1 - Παράμετροι Σύνθετης Μονάδας Δικτύου
2 - Υπομονάδες & Απόδοση τιμών στις παραμέτρους τους
3 - Συνδέσεις Υπομονάδων

Στο παρόν δίκτυο της προσομοίωσης δεν ορίζονται συνδέσεις για τις υπομονάδες, διότι η αποστολή μηνυμάτων πραγματοποιείται ανάμεσα στους κόμβους και το μοντέλο επικοινωνίας με απευθείας αποστολή σε θύρα, όπως αναλύεται στην C++ υλοποίηση [§6.4.1, §6.4.2]. Επομένως έχουμε μόνο τα στάδια 1 και 2:

6.2.3.1 Παράμετροι σύνθετης μονάδας Δικτύου

Οι παράμετροι που ορίζονται για το δίκτυο είναι οι εξής:

```
  parameters:
      verbosity: numeric const,
      numNodes: numeric const, //number of nodes
      display_X_size: numeric const,
      display_Y_size: numeric const;
```

[omnetpp-3.3p1/mysims/sim4/sim4.ned]

όπου:

- verbosity - το επίπεδο λεπτομέρειας αναφορών της προσομοίωσης
- numNodes - ο αριθμός των κόμβων του δικτύου: χρησιμοποιείται για τον ορισμό του διανύσματος των κόμβων
- display_X_size - η διάσταση της απεικόνισης του δικτύου κατά την οριζόντια έννοια σε εικονοστοιχεία: χρησιμοποιείται από τους κόμβους για την τροποποίηση της απεικόνισης της θέσης τους
- display_Y_size - η διάσταση της απεικόνισης του δικτύου κατά την κατακόρυφη έννοια σε εικονοστοιχεία: χρησιμοποιείται από τους κόμβους για την τροποποίηση της απεικόνισης της θέσης τους

6.2.3.2 Υπομονάδες & Απόδοση τιμών στις παραμέτρους τους

Υπομονάδες του μοντέλου είναι η σύνθετη μονάδα κόμβου (Node) και η σύνθετη μονάδα του μοντέλου επικοινωνίας (RF_Propagation_Model). Η δήλωσή τους στον τομέα "submodules:" είναι:

```
submodules:
     node: Node[numNodes]; //compound module
       parameters:
           numNodes = numNodes,
           verbosity = verbosity;
       display: "b=30,30,oval"; // NODE compound module display

     rf_module: RF_Propagation_Model //compound module
       parameters:
           core_ModuleType = input("core1",
 "assign default core moule of RF Propagation module"),
           path_loss_ModuleType = input("free_space",
 "assign default path loss module"),
           foliage_loss_ModuleType = input("itu_r",
 "assign default foliage loss module"),
           comm_in_ModuleType = input("comm_in1",
 "assign default msg handler for RF module"),
           comm_out_ModuleType = input("comm_out_rng",
 "assign default msg sender for RF module"),
           verbosity = verbosity,
           numNodes = numNodes;
        display: "p=400,300;b=30,30,oval";
```

[omnetpp-3.3p1/mysims/sim4/sim4.ned]

Στην περίπτωση των κόμβων, τους ορίζουμε όλους μαζί ως διάνυσμα του τύπου σύνθετης μονάδας Node. Κάθε κόμβος είναι μονάδα node[x], όπου 0 < x < numNodes - 1. Φυσικά όπως αναλύσαμε προηγουμένως αυτό δεν τους υποχρεώνει να έχουν και τις ίδιες υπομονάδες, λογική πάνω στην οποία υλοποιήθηκε ο διαχωρισμός αισθητήρων και πύλης. Στην δήλωση των κόμβων ορίζεται οι παράμετροι numNodes και verbosity να κληρονομούν τις τιμές τους από τις αντίστοιχες παραμέτρους του δικτύου. Τέλος δίνεται και μια πρώτη έκφραση απεικόνισης για τους κόμβους στον τομέα "display:". Οι κόμβοι παρίστανται με τετράγωνο πλευράς 30 εικονοστοιχείων. Δεν ορίζεται η θέση τους στην δήλωση αυτή, διότι αυτό υλοποιείται στην C++ [§6.4.1.1].

Για το rf_module που ορίζεται με το RF_Propagation_Model, έχουμε ως παραμέτρους πέρα από τον αριθμό των κόμβων και το επίπεδο αναφοράς, τις προεπιλεγμένες τιμές εισαγωγής για τις παραμέτρους των υπομονάδων του. Με την input(...) σύνταξη δίνουμε οδηγία στο Omnet++ για το ενδεχόμενο να μην έχουμε ορίσει κάποια παράμετρο [§2.6.1.3.2]. Το πρώτο μέρος της οδηγίας στην παρένθεση είναι η προεπιλεγμένη τιμή για την παράμετρο, ενώ το δεύτερο η οδηγία εισαγωγής. Εφόσον οι παράμετροι έχουν ήδη ορισθεί σε κάποιο ini αρχείο, η σύνταξη input δεν έχει κανένα αποτέλεσμα. Επίσης ορίζεται και η απεικόνιση της σύνθετης μονάδας rf_module.

```
display: "p=400,300;b=30,30,oval"; // RF model display
```

[omnetpp-3.3p1/mysims/sim4/sim4.ned]

Όπου "p=400,300" δηλώνει την θέση του κέντρου της απεικόνισης σε εικονοστοιχεία (pixels), και "b=20,20,oval" δηλώνει ότι η μονάδα παριστάνεται στη θέση αυτή με ένα κύκλο διαμέτρου 20 εικονοστοιχείων.

6.2.3.3 Απεικόνιση Δικτύου

Μετά και τον ορισμό των υπομονάδων, δηλώνεται η απεικόνιση για τη σύνθετη μονάδα του δικτύου. Η απεικόνιση αυτή είναι:

```
display: "p=10,10;b=800,600,rect"; // NETWORK display
```

[omnetpp-3.3p1/mysims/sim4/sim4.ned]

δηλαδή ορθογώνιο παραλληλόγραμμο 800x600 οικονοστοιχεία με εκκίνηση στο εικονοστοιχείο '10,10' του κανάβου. Μέσα σε αυτή τη διάσταση τοποθετούνται οι υπομονάδες του, οι κόμβοι και το μοντέλο επικοινωνίας. Είναι δυνατό να τοποθετηθούν και έξω από τα όρια του. Συνήθως δίνουμε στο δίκτυο διαστάσεις λίγο μικρότερες από το μέγεθος της οθόνης που παρακολουθούμε την προσομοίωση για να έχουμε πιο άνετη οπτική θεώρηση.

6.2.4 NED Ορισμός Δικτύου

Αφού έχει ολοκληρωθεί ο ορισμός της σύνθετης μονάδας του δικτύου της προσομοίωσης, πρέπει να δηλώσουμε ότι αυτή αποτελεί το 'Δίκτυο' (Network), δηλαδή το μοντέλο του δικτύου που θέλουμε να προσομοιώσουμε. Η δήλωση είναι:

```
network sim4 : Sim4
endnetwork
```

[omnetpp-3.3p1/mysims/sim4/sim4.ned]

6.3 Ορισμός Μηνύματος

Στην προσομοίωση όλες οι μονάδες ανταλλάσσουν μεταξύ τους μηνύματα, η παραλαβή των οποίων πυροδοτεί γεγονότα με την C++ μέθοδο *void handlemessage(cMessage*)*. Το μήνυμα είναι μια πολύ σημαντική λειτουργική οντότητα στο Omnet++. Είναι η κινητήριος δύναμη της προσομοίωσης [§2.1.1].

Στο δίκτυο προσομοίωσης που δημιουργήθηκε για τις ανάγκες της μελέτης, τα μηνύματα γεννούνται στους αισθητήρες (sensor επεξεργαστής). Κατόπιν κλωνοποιούνται στο μοντέλο επικοινωνίας και αποδίδονται σε αυτά παράμετροι για την εξασθένισης προς κάθε παραλήπτη. Τέλος παραδίδονται στις πύλες (gateway επεξεργαστές) [Σχήμα 6.3].

Είναι αντιληπτό ότι μόνο ένα μήνυμα, ένας τύπος μηνύματος, διακινείται στο δίκτυο. Ορίζουμε το μήνυμα αυτό σαν radio_msg:

```
message radio_msg
{};
```

[omnetpp-3.3p1/mysims/sim4/radio_msg.msg]

Ο ορισμός είναι κενός. Όλες οι παράμετροι προστίθενται στο μήνυμα σε κάθε μονάδα που υπολογίζει τις τιμές τους, με την '*void addPar(const char *s)*' μέθοδο της τάξης μηνυμάτων cMessage [17]. Αφού προστεθεί η παράμετρος ορίζεται η τιμή της με την '*void par(const char *s)*'. Το μήνυμα γεννάται κενό και καταλήγει σε κάθε gateway γεμάτο πληροφορίες από όλες τις μονάδες της προσομοίωσης. Όταν τα μηνύματα αντιγράφονται στην μονάδα comm_in του rf_model, κάθε κλώνος έχει τις ίδιες παραμέτρους με το αυθεντικό μήνυμα.

6.4 C++ Ορισμός των Απλών Μονάδων Προσομοίωσης

Εφόσον το δίκτυο έχει δομηθεί με τη δήλωση στη NED γλώσσα όλων των μονάδων του και των θυρών και συνδέσεων αυτών, απομένει ο ορισμός των απλών μονάδων του (simple modules), και των λειτουργιών τους στη C++ γλώσσα προγραμματισμού.

Η δομή μιας απλής μονάδας στη C++ περιλαμβάνει οπωσδήποτε τρεις μεθόδους:

(1) initialize()
(2) handleMessage(cMessage*)
(3) finish()

Πλην αυτών σε κάθε απλή μονάδα δημιουργούνται κατάλληλες βοηθητικές μέθοδοι για να εξυπηρετήσουν τις λειτουργίες της. Αυτές είναι ενσωματωμένες σε κάθε περίπτωση στο ίδιο αρχείο κώδικα για λόγους τάξης και οργάνωσης. Θα ήταν δυνατό σε όμοιες μονάδες να τοποθετηθούν οι λειτουργίες αυτές σε ξεχωριστό αρχείο που να ενσωματώνεται με την οδηγία #include, ή και ακόμα να ορισθεί κάθε μονάδα σαν παράγωγη τάξη μιας γενικής μονάδας κάθε τύπου, και να ορίζονται στις παράγωγε τάξεις μόνο οι διαφορετικές μέθοδοι αυτών. Λόγω του μικρού όγκου κώδικα δεν εφαρμόστηκε τέτοια οργάνωση, και κάθε τάξη ορίζεται τελείως αυτόνομα.

Οι απλές μονάδες παρόμοιου τύπου έχουν παρόμοιες τάξεις. Για παράδειγμα οι μονάδες που υλοποιούν την εξασθένισης διαδρομής διαφέρουν μόνο ως προς μια μέθοδο. Σε τέτοιες περιπτώσεις όλες οι τάξεις παρουσιάζονται μαζί και αναλύεται η διαφοροποίηση.

6.4.1 Επεξεργαστές sensor & gateway: C++ Ορισμοί

Οι επεξεργαστές που υλοποιήθηκαν για την προσομοίωση είναι δύο ειδών:

- sensor : Αισθητήρας
- gateway : Πύλη

Δεν μοιράζονται τις ίδιες ακριβώς παραμέτρους, ούτε έχουν τελείως ίδιες μεθόδους. Οι ομοιότητές τους έχουν να κάνουν αποκλειστικά με τον ορισμό της απεικόνισης των κόμβων τους και τον μετασχηματισμό των συντεταγμένων τους. Οι εργασίες αυτές εκτελούνται στην *initialize()* μέθοδο.

6.4.1.1 *void initialize()*

Οι επεξεργαστές δεν ορίζουν απεικόνιση στη δική τους μονάδα, αλλά στον κόμβο που τους περιέχει. Η εντολή για τους πύλες και αισθητήρες είναι αντίστοιχα:

```
   parentModule()->displayString().setTagArg("i",0,"abstract/accesspoint");
 //display image for gateway
   parentModule()->displayString().setTagArg("i",0,"misc/node_s");
//display image for sensor
```

Επίσης εκτελείται η μέθοδος display_generator(cModule*), η οποία έχει ως μονάδα εισόδου τη σύνθετη μονάδα του κόμβου που περιέχει τον επεξεργαστή. Η εντολή είναι:

```
display_generator(parentModule())
```

Η μέθοδος *display_generator(cModule*)* αναλύεται στην παράγραφο §6.4.1.3, και είναι υπεύθυνη για τον μετασχηματισμό των πραγματικών συντεταγμένων των κόμβων σε συντεταγμένες οθόνης, ώστε να διατηρούνται οι σχετικές θέσεις και να βρίσκονται όλοι οι κόμβοι στο πεδίο απεικόνισης της προσομοίωσης, όπως αυτό ορίζεται στην δήλωση των διαστάσεων του δικτύου.

Ο υπόλοιπος κώδικας διαχωρίζεται για τις περιπτώσεις πύλης (gateway) και αισθητήρα (sensor)

6.4.1.1.1 sensor

Στην περίπτωση των αισθητήρων, επιθυμούμε στη μέθοδο αυτή να στέλνεται το πρώτο μήνυμα στον ίδιο τον αισθητήρα (selfMessage) [17], ώστε να ενεργοποιήσει τις λειτουργίες της *handleMessage(cMessage*)* μεθόδου. Η αποστολή του μηνύματος γίνεται με την *scheduleAt(double time, cMessage* selfMessage)* [17], και πραγματοποιείται με δύο διαφορετικούς τρόπους. Κριτήριο είναι η τυχαιότητα του χρόνου αποστολής των μηνυμάτων. Οι εκφράσεις είναι:

```
if(!msg_rng)
    scheduleAt(simTime()+0.1, selfMsg);
    //schedule msg for sending after 100ms
else if(msg_rng)
    scheduleAt(simTime()+dblrand(), selfMsg);
    //send message after random time less than a sec
```

[omnetpp-3.3p1/mysims/sim4/sensor.cc]

όπου msg_rng είναι bool παράμετρος που δηλώνεται ευθέως για τους αισθητήρες στο omnetpp.ini αποδίδεται σε ομότιτλη παράμετρο στην τάξη του sensor επεξεργαστή:

```
msg_rng = par("msg_rng");
```

[omnetpp-3.3p1/mysims/sim4/sensor.cc]

```
sim4.node[*].processor.msg_rng = true
```

[omnetpp-3.3p1/mysims/sim4/omnetpp.ini]

Στην περίπτωση που δεν έχουμε τυχαιότητα στον χρόνο αποστολής, το μήνυμα στέλνεται για

όλους τους κόμβους σε χρόνο '*simTime()*+0.1' δευτερόλεπτα. Για την εκκίνηση της προσομοίωσης το *simTime()* είναι μηδέν. Άρα το πρώτο μήνυμα θα φύγει σε 0.1 δευτερόλεπτα. Δεδομένου ότι δεν ορίζεται καθυστέρηση στα selfMessages, στον χρόνο αυτό θα εκτελεστεί και η *handleMessage(cMessage*)* μέθοδος του επεξεργαστή. Οι χρόνοι '*simTime()*+0.1' είναι κοινοί για όλους τους κόμβους. Δηλαδή η αποστολή μηνυμάτων γίνεται ταυτόχρονα από όλους. Εφόσον επομένως δεν ορίζεται καθυστέρηση σε καμία σύνδεση, τα μηνύματα φτάνουν ταυτόχρονα και στους τελικούς προορισμούς.

Αν έχουμε τυχαίο χρόνο αποστολής, τότε κάθε μήνυμα στέλνεται σε διαφορετικό χρόνο από τα υπόλοιπα. Ο χρόνος αυτός είναι πάντα ίσος με τον χρόνο της προσομοίωσης συν ένα τυχαίο δεκαδικό στο διάστημα [0,1). Δηλαδή η μέθοδος *handleMessage(cMessage*)* θα εκτελεστεί σε λιγότερο από ένα δευτερόλεπτο από τον χρόνο αποστολής του μηνύματος.

6.4.1.1.2 gateway

Όπως είδαμε στον ορισμό των επεξεργαστών στη NED γλώσσα, αυτοί δεν έχουν θύρες, ούτε εισόδου ούτε εξόδου. Χωρίς θύρες εξόδου είναι δυνατό να στέλνουμε μηνύματα με τη μέθοδο *sendDirect(cMessage *msg, double delay, cGate *gate)*. Δεν γίνεται όμως να λαμβάνουμε μηνύματα που δεν είναι selfMessages χωρίς την ύπαρξη έστω μίας θύρας εισόδου.

Στη μέθοδο *initialize()* του gateway λύνεται αυτό το πρόβλημα προσθέτοντας μια θύρα εισόδου στην εκκίνηση της προσομοίωσης, όπου δεν έχουν ακόμα αποσταλεί μηνύματα. Η σύνταξη είναι:

```
char I;
addGate("fromNetwork",I);
```

[omnetpp-3.3p1/mysims/sim4/gateway.cc]

όπου ο χαρακτήρας 'I' δηλώνει θύρα εισόδου. Αν θέλαμε να δηλώσουμε προσθήκη θύρας εξόδου θα υποδεικνύαμε την κατεύθυνση με τον χαρακτήρα 'O'.

6.4.1.2 *void handleMessage(cMessage*)*

6.4.1.2.1 sensor

Η μέθοδος αυτή ανταποκρίνεται σε εισερχόμενα μηνύματα. Παρότι δεν έχουμε θύρες εισόδου για να λαμβάνει μηνύματα από άλλους κόμβους ένας αισθητήρας, ελέγχουμε το εισερχόμενο για τον τύπο του. Αυτή είναι πρακτική σε κόμβους με αμφίδρομη επικοινωνία. Εδώ αναμένουμε μόνο selfMessage. Για να λειτουργήσει η μέθοδος πρέπει η msg->isSelfMessage() να επιστρέψει "true".

Εφόσον το εισερχόμενο μήνυμα είναι selfMessage, δημιουργείται ένα νέο μήνυμα radio_msg και επιστρέφεται δείκτης σε αυτό. Στο μήνυμα αυτό προστίθενται και αποδίδονται δύο παράμετροι:

scrAddress_internal : η διεύθυνση του κόμβου του sensor στο διάνυσμα των κόμβων
srcAddress_external : η εξωτερική αρίθμηση που έχουμε για τον κόμβο

```
cMessage *msg = new radio_msg("hello");
msg->addPar("srcAddress_internal");
msg->par("srcAddress_internal") = NODE->index();;
msg->addPar("srcAddress_external");
msg->par("srcAddress_external") = NODE->par("nodeID");
```

[omnetpp-3.3p1/mysims/sim4/sensor.cc]

Τέλος το μήνυμα αποστέλλεται στη θύρα "fromNetwork" του comm_in_module του rf_module (RF_Propagation_Model). Η αποστολή γίνεται με τη μέθοδο *sendDirect(cMessage *msg, double delay, cGate *gate)*, καθότι δεν έχουμε και δεν θέλουμε θύρες εξόδου στους επεξεργαστές. Η εντολή είναι:

```
sendDirect( msg, 0, RADIO, "fromNetwork");
```

[omnetpp-3.3p1/mysims/sim4/sensor.cc]

όπου RADIO είναι δείκτης στο comm_in_module και ορίζεται με το παρακάτω macro:

#define RADIO
parentModule()->parentModule()->submodule("rf_module")->submodule("comm_in_module")

Πριν επιστρέψει η μέθοδος στην προσομοίωση δημιουργείται νέο selfMessage με τον ίδιο απολύτως τρόπο όπως και στην μέθοδο *initialize()* [§6.4.1.1.1].

6.4.1.2.2 gateway

Το gateway με το που θα λάβει ένα μήνυμα από το comm_out_module, θα διαβάσει τις παραμέτρους srcAddress_external και receveidPower και θα τις αποθηκεύσει σε multimap [8]. Το multimap αυτό έχει εγγραφές με τον αποστολέα κάθε μηνύματος και την ισχύ εκπομπής που το παρέλαβε το gateway. Ο τύπος αυτός container, σε αντίθεση με τα set και map επιτρέπει πολλαπλές εμφανίσεις του ίδιου κλειδιού, που εδώ είναι το nodeID των κόμβων. Τα δεδομένα αυτά αξιοποιούνται στην finish() μέθοδο για εξαγωγή στατιστικών. Πριν επιστρέψει η μέθοδος καταστρέφει το μήνυμα.

6.4.1.3 *void display_generator(cModule*)*

Το Omnet++ δεν έχει κάποια λειτουργία αυτόματης προσαρμογής των συντεταγμένων στις διαστάσεις της οθόνης. Εφόσον κάποιος ορίσει τις συντεταγμένες ως ετικέτα απεικόνισης στη NED γλώσσα (p=x,y)[17] τότε αυτές μεταφράζονται σε εικονοστοιχεία οθόνης. Κάτι τέτοιο δεν είναι αποδεκτό και δεν έχει καμία χρησιμότητα.

Η μέθοδος αυτή δημιουργήθηκε εκ του μηδενός, προκειμένου να παραστήσουμε τους κόμβους με τις σχετικές τους θέσεις στην απεικόνιση. Ως δεδομένα εισόδου αξιοποιεί τις συντεταγμένες X, Y των κόμβων και τις διαστάσεις της απεικόνισης της σύνθετης μονάδας του δικτύου. Οι τελευταίες δίνονται στις βοηθητικές παραμέτρους του, display_X_size και display_Y_size.

Η μέθοδος εφαρμόζει τα εξής βήματα:

- Βρίσκει τις μέγιστες και ελάχιστες τιμές των X και Y συντεταγμένων (max(X), min(X), max(Y), min(Y))

- Για τις παραπάνω ακρότατες τιμές υπολογίζονται οι μέγιστες διαφορές συντεταγμένων dX και dY:

  dX = max(X) - min(X) και dy = max(Y) - min(Y)

- Υπολογίζεται η σχέση των dX και dY με τις διαστάσεις της απεικόνισης του δικτύου. Από αυτές αφαιρείται ένα μέγεθος (πχ 100 pixels) προκειμένου στην προσαρμογή να μην αγγίζουν τα όρια της απεικόνισης του δικτύου οι κόμβοι:

  scale_X = (display_X_size - 100) / dX (κλίμακα pixels/μέτρα)
  scale_Y = (display_Y_size - 100) / dY

- Βρίσκουμε τη μεγαλύτερη από τις δύο κλίμακες και την ορίζουμε ως κλίμακα μετασχηματισμού των συντεταγμένων.

- Μεταθέτουμε τις νέες συντεταγμένες ώστε να βρίσκονται στο κέντρο της απεικόνισης του δικτύου.

- Τέλος προσθέτουμε ετικέτα θέσης στον κόμβο κάθε επεξεργαστή, και αποδίδουμε τις τιμές των συντεταγμένων οθόνης:

```
NODE->displayString().insertTag("p");
NODE->displayString().setTagArg("p",0,screen_X);
NODE->displayString().setTagArg("p",1,screen_Y);
```

6.4.1.4 *void finish()*

6.4.1.4.1 sensor

Η μέθοδος δεν αξιοποιείται

6.4.1.4.1 gateway

Στη μέθοδο αυτή υλοποιείται αναφορά στατιστικών στοιχείων από τα μηνύματα που παρέλαβε ο επεξεργαστής κατά τη διάρκεια της προσομοίωσης. Για το σκοπό αυτό δημιουργείται ofstream [8] για εγγραφή σε αρχείο με το nodeID του κόμβου του επεξεργαστή:

```
char filename[8];
int nodeID=NODE->par("nodeID");
sprintf(filename, "%d", nodeID);
std::ofstream ost(filename);
```

[omnetpp-3.3p1/mysims/sim4/gateway.cc]

Η μέθοδος χρησιμοποιεί επαναλήψεις για να εξάγει από το multimap συλλογής στατιστικών [§6.4.1.2.2] το συνολικό αριθμό μηνυμάτων και την μέση τιμή ισχύος λήψης που έλαβε από κάθε αποστολέα. Με το ofstream τυπώνει τα στοιχεία αυτά σε αρχείο εξόδου "*_report.txt", του οποίου τον όνομα συμπληρώνεται από την διεύθυνση του κόμβου (nodeID).

6.4.2 Υπομονάδες Μοντέλου Επικοινωνίας: C++ Ορισμοί

6.4.2.1 comm_in_module

Ο τύπος μονάδας comm_in_module είναι υπεύθυνος για την υποδοχή των μηνυμάτων που στέλνουν οι επεξεργαστές sensor των κόμβων στο rf_module. Για την τύπο αυτό έχει ορισθεί μόνο μία μονάδα, η comm_in1. Η δήλωσή της (declaration) είναι:

```
class comm_in1 : public cSimpleModule
{
    public:
      comm_in1();
      virtual ~comm_in1();

    protected:
      virtual void initialize();
      virtual void finish();
      virtual void send_to_core(cMessage *radio_msg);
      virtual void handleMessage(cMessage *msg);
      virtual double distance_v1( cModule *node1,
                                  cModule *node2 ) throw();
      virtual double transmit_power_dBm() throw();

    private:
      int          verbosity;
      double       Pt;
      std::string  Pt_unit;
 };
```

[omnetpp-3.3p1/mysims/sim4/comm_in1.cc]

Η τάξη καταχωρείται στο Omnet++ με την μακροεντολή Define_Module(comm_in1).

Από τη δήλωση της μεθόδου διακρίνει κανείς τρεις βοηθητικές, πέρα από τις απαιτούμενες, τις *void send_to_core(cMessage*)*, *double distance_v1(cModule*, cModule*)* και *double transmit_power_dBm()*. Αυτές συνεργάζονται στην *handlemessage (cMessage*)* μέθοδο.

6.4.2.1.1 *void initialize()*

Στη μέθοδος εκκίνησης αποδίδονται τιμές στις μεταβλητές Pt και verbosity:

```
     verbosity=par("verbosity");
     Pt = par("transmitPower");
```

6.4.2.1.2 void handleMessage (cMessage*)

Η μέθοδος αυτή με το που θα λάβει ένα μήνυμα εξετάζει αν έχει παράμετρο για την ισχύ σήματος. Αν όχι την προσθέτει στο μήνυμα και της αποδίδει την τιμή που έχει κληρονομήσει από το rf_module. Πριν γίνει αυτό όμως πρέπει να εξεταστεί η τιμή αυτή σε τι μονάδες έχει ορισθεί και να μετατραπεί σε dBm. Αυτό γίνεται με τη μέθοδο *double transmit_power_dBm()*:

```
if(!msg->hasPar("transmitPower_dBm"))
{
    msg->addPar("transmitPower_dBm");
    msg->par("transmitPower_dBm") = transmit_power_dBm();
}
```

[omnetpp-3.3p1/mysims/sim4/comm_in1.cc]

6.4.2.1.3 ***double transmit_power_dBm()***

Η *double transmit_power_dBm()* υλοποιείται όπως και στο Shawn [§5.3.2.a.1]. Ελέγχεται η παράμετρος "transmitPowerUnit" που ορίζεται στο ini αρχείο για το rf_module. Αν έχει μια τιμή μεταξύ "W", "mW" και "dBm", χωρίς έλεγχο κεφαλαίων, κάνει το κατάλληλο μετασχηματισμό στις δύο πρώτες περιπτώσεις και σε κάθε περίπτωση αποδίδει την τιμή στο μήνυμα. Αν η τιμή της μονάδας ισχύος είναι άκυρη, το πρόγραμμα τερματίζεται και ενημερώνει το χρήστη για το σφάλμα και τις αναμενόμενες τιμές.

Εφόσον αποδοθεί η ισχύς εκπομπής σε dBm στο μήνυμα, η *void handleMessage (cMessage*)* εκτελεί την *void send_to_core (cMessage*)* με παράμετρο δείκτη προς το μήνυμα που επεξεργάζεται.

6.4.2.1.4 ***void send_to_core (cMessage*)***

Η μέθοδος αυτή εξετάζει για όλους τους κόμβους ποιοι έχουν επεξεργαστή gateway, δηλαδή αναζητεί τους πιθανούς παραλήπτες του μηνύματος που έλαβε η μονάδα. Για κάθε πιθανό παραλήπτη δημιουργεί ένα αντίγραφο του πρωτότυπου μηνύματος, το οποίο έχει και όλες τις παραμέτρους του μηνύματος. Αυτό γίνεται με τη μέθοδο *dup()* της τάξης μηνυμάτων cMessage:

```
cMessage *msg = (cMessage *) radio_msg->dup();
```

[omnetpp-3.3p1/mysims/sim4/comm_in1.cc]

το μήνυμα που αντιστοιχεί στον δείκτη msg είναι radio_msg μήνυμα, ίδιο με το πρωτότυπο. Στη συνέχεια για κάθε μήνυμα προστίθενται οι εξής παράμετροι:

"destAddress_internal"
"destAddress_external"
"distance_m"

Για κάθε παραλήπτη αποδίδεται στην "destAddress_internal" η θέση του κόμβου που ανήκει στο διάνυσμα των κόμβων. Η παράμετρος αυτή δηλαδή συγκρατεί την εσωτερική διεύθυνση του παραλήπτη στο περιβάλλον της προσομοίωσης. Στην παράμετρο "destAddress_external" αποδίδεται η διεύθυνση ή η ταυτότητα/αρίθμηση του κόμβου, σύμφωνα με την δική μας οργάνωση κατά τον ορισμό του δικτύου. Η τιμή αυτή είναι δηλαδή η παράμετρος nodeID των κόμβων.

Η εσωτερική αρίθμηση χρησιμοποιείται για την καλύτερη διαχείριση των μηνυμάτων στην προσομοίωση, ενώ η εξωτερική κατά την εκτύπωση στατιστικών και αναφορών, προκειμένου να αναγνωρίζουμε τις οντότητες όπως τις έχουμε καταγράψει.

Η παράμετρος "distance_m" αντιστοιχεί στην απόσταση του κόμβου που έστειλε το αυθεντικό μήνυμα από τον παραλήπτη του εκάστοτε κλώνου του μηνύματος. Υπολογίζεται με την μέθοδο 'double distance_v1(cModule*, cModule*)', που δέχεται ως παραμέτρους δείκτες προς τους δύο κόμβους.

Τέλος κάθε αντίγραφο του αρχικού μηνύματος προωθείται στη μονάδα κεντρικής διαχείρισης του μοντέλου επικοινωνίας, μέσω της θύρας "toCore":

```
send(msg, "toCore"); //forward message to core module
```

[omnetpp-3.3p1/mysims/sim4/comm_in1.cc]

6.4.2.1.5 *void finish()*

Η μέθοδος δεν αξιοποιείται.

6.4.2.2 core_module

Ο τύπος μονάδας core_module είναι υπεύθυνος για την δρομολόγηση των μηνυμάτων ανάμεσα στις υπόλοιπες υπομονάδες του μοντέλου επικοινωνίας. Θα μπορούσαμε να τον αποκαλούμε "Δρομολογητή" αντί για "Κεντρική Διαχείριση". Για τον τύπο αυτό έχει ορισθεί μία μόνο μονάδα, η core1. Η C++ δήλωσή της τάξης της είναι:

```
class core1 : public cSimpleModule
{
  public:
    core1();
    virtual ~core1();
  protected:
    virtual void initialize();
    virtual void finish();
    virtual void handleMessage(cMessage *msg);
  private:
    int          verbosity;
};
```

[omnetpp-3.3p1/mysims/sim4/core1.cc]

Η τάξη καταχωρείται στο Omnet++ με την μακροεντολή Define_Module(core1).

6.4.2.2.1 *void initialize()*

Στη μέθοδος εκκίνησης αποδίδεται τιμή στη μεταβλητή verbosity:

```
verbosity=par("verbosity");
```

6.4.2.2.2 *void handleMessage (cMessage*)*

Η διαχείριση των μηνυμάτων στην κεντρική μονάδα υλοποιείται με τον παρακάτω κώδικα:

```
if(!msg->hasPar("mean_foliage_loss_dBm"))
{
    send(msg, "toFL");
}
    else if(!msg->hasPar("mean_path_loss_dBm"))
{
    send(msg, "toPL");
}
    else
{
    send(msg, "toCommOut");
}
```

[omnetpp-3.3p1/mysims/sim4/core1.cc]

Κάθε μήνυμα ελέγχεται αν διαθέτει παράμετρο για την μέση απώλεια βλάστησης ("mean_foliage_loss_dBm"). Αν όχι στέλνεται στην θύρα "toFL" που συνδέει τη μονάδα με την μονάδα fl_module που είναι υπεύθυνη για τον υπολογισμό της απώλειας βλάστησης. Όταν το fl_module θα επιστρέψει το μήνυμα στην κεντρική διαχείριση, η *handleMessage(cMessage*) θα βρει την παράμετρο "mean_foliage_loss_dBm" και θα περάσει στον επόμενο έλεγχο, όπου ελέγχει αν υπάρχει παράμετρος στο μήνυμα για την απώλεια διαδρομής ("mean_path_loss_dBm"). Αν όχι το μήνυμα στέλνεται στην θύρα εξόδου "toPL" που συνδέει την κεντρική διαχείριση core_module με τη μονάδα pl_module. Αυτή επιστρέφει το μήνυμα με την απώλεια διαδρομής και εν τέλει η *handleMessage(cMessage*)* αφού βρίσκει και τις δύο παραμέτρους στο μήνυμα, το προωθεί μέσω της θύρας εξόδου "toCommOut" στη μονάδα επικοινωνίας εξερχομένων comm_out_module.

6.4.2.2.3 *void finish()*

Η μέθοδος δεν αξιοποιείται.

6.4.2.3 fl_module

Ο τύπος μονάδας fl_module υπολογίζει και αποδίδει στα μηνύματα την εξασθένιση στην ισχύ εκπομπής λόγω βλάστησης [§4.1.3.1]. Η εξασθένιση είναι θετική ποσότητα και αφαιρείται από την ισχύ εκπομπής. Για τον τύπο αυτής της μονάδας ορίσθηκαν πέντε απλές μονάδες:

- weissberger
- itu_r
- cost235_in_leaf
- cost235_out_of_leaf
- no_foliage

Οι τάξεις των μονάδων αυτών έχουν την ίδια ακριβώς δήλωση, δηλαδή τις ίδιες μεθόδους και παραμέτρους. Εξαίρεση αποτελεί η μονάδα no_foliage, η οποία δεν διαθέτει την μέθοδο *virtual double mean_foliage_loss_dBm(cMessage *msg)*. Τα ονόματα των τάξεων και των constructors/destructors είναι ανάλογα του ονόματος της μονάδας. Παρουσιάζεται η C++ δήλωση της μονάδας του μοντέλου του Weissberger:

```
class weissberger : public cSimpleModule
{
   public:
     weissberger();
     virtual ~weissberger();
   protected:
     virtual void initialize();
     virtual void handleMessage(cMessage *msg) throw();
     virtual double mean_foliage_loss_dBm(cMessage *msg) throw();
   private:
     int verbosity;
     double f_GHz;
     double fpf;
};
```

[omnetpp-3.3p1/mysims/sim4/weissberger.cc]

Κάθε τάξη καταχωρείται στο Omnet++ με την μακροεντολή Define_Module("Module Name").

6.4.2.3.1 *void initialize()*

Στη μέθοδο εκκίνησης αποδίδονται τιμές στις μεταβλητές f_GHz, verbosity και fpf:

```
verbosity=par("verbosity");
f_GHz = par("frequency_GHz");
fpf = par("foliagePathFactor");
```

6.4.2.3.2 *void handleMessage (cMessage*)*

Η διαχείριση μηνυμάτων σε αυτό τον τύπο μονάδας, προσθέτει σε κάθε μήνυμα παράμετρο για τη μέση εξασθένιση βλάστησης ("mean_foliage_loss_dBm"), και αποδίδει σε αυτή τιμή βάσει της μεθόδου *double mean_foliage_loss_dBm(cMessage *msg)*. Η τελευταία αντιστοιχεί στην εφαρμογή του μοντέλου εξασθένιση βλάστησης, που είναι διαφορετικό σε κάθε τύπο μονάδας fl_module. Η μονάδα no_foliage διαθέτει μέθοδο *virtual double mean_foliage_loss_dBm(cMessage *msg)*, και αποδίδει ευθέως τιμή μηδέν (0) στην παράμετρο της εξασθένισης.

Παρουσιάζονται οι σχέσεις υπολογισμού όπως υλοποιούνται σε κάθε μονάδα που αντιστοιχεί σε μοντέλο εξασθένισης βλάστησης [§4.1.3.1]:

- weissberger:

```
if( 0 < df_m && df_m <= 14 )
{
    LdB = 0.45 * pow(f_GHz, 0.284) * df_m;
}
else if( 14 < df_m && df_m <= 400 )
{
    LdB = 1.33 * pow(f_GHz, 0.284) * pow(df_m, 0.588);
}
else
    // ERROR HANDLE FOR ILLEGAL DISTANCE
```

- itu_r:

```
LdB = 0.2 * pow((f_GHz * 1000) , 0.3) * pow(df_m, 0.6);
```

- cost235_in_leaf:

```
LdB = 15.6 * pow((f_GHz * 1000) , -0.009) * pow(df_m, 0.36);
```

- cost235_out_of_leaf:

```
LdB = 15.6 * pow((f_GHz * 1000) , -0.2) * pow(df_m, 0.5);
```

όπου:

LdB: Η μέση εξασθένιση βλάστησης σε dBm (θετική ποσότητα)
df_m: Το εύρος βλάστησης σε μέτρα
f_GHz: Η συχνότητα εκπομπής σε GHz

Το εύρος βλάστησης υπολογίζεται σε κάθε περίπτωση από την παράμετρο "distance_m" που φέρει κάθε μήνυμα (ορίσθηκε στο comm_in_module) και την παράμετρο "foliagePathFactor" που ορίζεται στο ini αρχείο παραμέτρων για το rf_module.

Στην περίπτωση της μονάδας weissberger, που υλοποιεί το αντίστοιχο μοντέλο, για τιμή εύρους πέρα από τα 400 μέτρα η μέθοδος τερματίζει την προσομοίωση και ενημερώνει τον χρήστη για το σφάλμα.

6.4.2.3.2 *void finish()*

Η μέθοδος δεν αξιοποιείται

6.4.2.4 pl_module

Ο τύπος μονάδας pl_module υπολογίζει και αποδίδει στα μηνύματα την εξασθένιση στην ισχύ εκπομπής λόγω διαδρομής. Η εξασθένιση είναι θετική ποσότητα και αφαιρείται από την ισχύ εκπομπής. Για τον τύπο αυτής της μονάδας ορίσθηκαν πέντε απλές μονάδες:

- free_space
- log_distance
- egli
- multipath_approximation
- mulitpath_analysis

Από τις μονάδες αυτές ξεχωρίζει ιδίως η log_distance. Οι υπόλοιπες έχουν παρόμοια δήλωση. Παρουσιάζεται η δήλωση της free_space μονάδας στη C++:

```
class free_space : public cSimpleModule
{
   public:
      free_space();
      virtual ~free_space();
   protected:
      virtual void initialize();
      virtual void handleMessage(cMessage *msg) throw();
      virtual double mean_path_loss_dBm( cMessage *msg ) throw();
   private:
      int verbosity;
      double f_GHz;
      double Gt_dB;
      double Gr_dB;
};
```

[omnetpp-3.3p1/mysims/sim4/free_space.cc]

Στην παραπάνω δήλωση για την περίπτωση της log_distance, έχουμε μια επιπλέον μέθοδο, την *double distance_v1(cModule*, cModule*)*, που συναντήσαμε και στη μονάδα comm_in. Επιπλέον κάθε μονάδα έχει δηλωμένες ακριβώς τις απαραίτητες παραμέτρους για την εκτέλεση των λειτουργιών της, παρά το γεγονός ότι σε όλες έχουν αποδοθεί οι ίδιες παράμετροι μέσω της NED δήλωσης. Εξαίρεση αποτελεί πάλι η log_distance μονάδα, καθότι έχει επιπλέον ορισμένη παράμετρο "log_distance_exponent".

Κάθε τάξη καταχωρείται στο Omnet++ με την μακροεντολή Define_Module("Module Name").

6.4.2.4.1 *void initialize()*

Στη μέθοδο εκκίνησης αποδίδονται τιμές στις απαραίτητες για την λειτουργία του μοντέλου εξασθένισης μεταβλητές. Για την περίπτωση του μοντέλου Εξασθένισης Λογαριθμικής Απόστασης, οι ορισμοί είναι:

```
      f_GHz = par("frequency_GHz");
      Gt_dB = par("transmitterAntennaGain_dB");
      Gr_dB = par("receiverAntennaGain_dB");
      hr_m = par("receiverHeight_m");
      ht_m = par("transmitterHeight_m");
      n = par("log_distance_exponent");
      numNodes = NETWORK->par("numNodes");
      verbosity = par("verbosity");
```

[omnetpp-3.3p1/mysims/sim4/log_distance.cc]

Ο αριθμός των κόμβων (numNodes) ορίζεται με την παράμετρο του δικτύου. Η NETWORK έκφραση είναι μακροεντολή που αντικαθιστά δείκτη προς την σύνθετη μονάδα του δικτύου:

#define NETWORK parentModule()->parentModule()

6.4.2.4.2 *void handleMessage (cMessage*)*

Η μέθοδος αυτή είναι στα ίδια πρότυπα με την αντίστοιχη του fl_module [§6.4.2.3.2]. Η διαφορά είναι στο όνομα της μεθόδου υπολογισμού της εξασθένισης και φυσικά της παραμέτρου αυτής που ορίζεται σε κάθε μήνυμα. Επομένως προστίθεται παράμετρος "mean_path_loss_dBm", η οποία παίρνει την τιμή της από την μέθοδο *double mean_path_loss_dBm(cMessage *msg)*. Να τονίσουμε εδώ ότι στη περίπτωση της μονάδας egli η μέθοδος έχει τίτλο *double median_path_loss_dBm(cMessage *msg)*, καθότι υπολογίζει την μεσαία τιμή εξασθένισης στην περιοχή ή στο δείγμα [§4.1.3.2.2].

Οι σχέσεις υπολογισμού της μέσης εξασθένισης διαδρομής για τις μονάδες pl_module είναι:

- free_space:

```
Lfs = - Gt_dB - Gr_dB - 20.0 * log10(wl_m)
      + 20.0 * log10(d_m) + 20.0 * log10(4 * PI);
```

[omnetpp-3.3p1/mysims/sim4/free_space.cc]

- log_distance:

```
Ldfs = Ld0 + n * 10.0 * log10(d_m / d0);
```

[omnetpp-3.3p1/mysims/sim4/log_distance.cc]

- egli:

```
L50 = - Gt_dB - Gr_dB - 20 * log10(hr_m) - 20 * log10(ht_m)
      - 10 * log10(b) + 40 * log10(d_m);
```

[omnetpp-3.3p1/mysims/sim4/egli.cc]

- multipath_approximation:

```
Lmp = - Gt_dB - Gr_dB - 20.0 * log10(hr_m) - 20.0 * log10(ht_m)
      + 40.0 * log10(d_m);
```

[omnetpp-3.3p1/mysims/sim4/multipath_approximation.cc]

- mulitpath_analysis:

```
Lmp = - 10 * log10(4) - Gt_dB - Gr_dB - 20 * log10(wl_m)
      - 20 * log10(sin((2  * PI * hr_m * ht_m)/(d_m * wl_m)))
      + 20.0 * log10(d_m * 4 * P );
```

[omnetpp-3.3p1/mysims/sim4/multipath_analysis.cc]

όπου:

Lfs: μέση εξασθένιση διαδρομής Ελεύθερου Χώρου
Ldfs: μέση εξασθένιση διαδρομής Λογαριθμικής Απόστασης
L50: ενδιάμεση εξασθένιση Egli
Lmp: μέση εξασθένιση διαδρομής Ανακλάσεων Δύο Δρόμων

και
Gt_dB: κέρδος κεραίας πομπού σε dB
Gr_dB: κέρδος κεραίας δέκτη σε dB
hr_m: ύψος κεραίας δέκτη σε μέτρα
ht_m: ύψος κεραίας πομπού σε μέτρα
wl_m: μήκος κύματος σήματος σε μέτρα
PI: ο αριθμός 'π'
d_m: απόσταση ανάμεσα σε πομπό και δέκτη σε μέτρα
d0: απόσταση αναφοράς σε μέτρα
Ld0: μέση εξασθένιση αναφοράς (υπολογίζεται με το μοντέλο ελεύθερου χώρου) σε dBm

Εφόσον υπολογιστεί η εξασθένιση διαδρομής, αποδίδεται η τιμή στην παράμετρο "mean_path_loss_dBm" του μηνύματος, και αυτό στέλνεται πίσω στο core_module μέσω της θύρας εξόδου "toCore".

6.4.2.4.3 *void finish()*

Η μέθοδος δεν αξιοποιείται

6.4.2.5 comm_out_module

Η μονάδα αυτή λαμβάνει τα μηνύματα από την κεντρική διαχείριση, εφόσον έχουν προστεθεί στα περιεχόμενά τους όλες οι εξασθενίσεις σήματος, και υλοποιεί την τελική απόφαση για την αποστολή κάθε μηνύματος στον παραλήπτη του. Σε αυτή τη περίπτωση έχουν υλοποιηθεί δύο τύποι μονάδων:

- comm_out_rng
- comm_out_probability

Οι μονάδες αυτές είναι αρκετά διαφορετικές ως προς την δήλωσή τους και γι' αυτό παρουσιάζονται τελείως ξεχωριστά.

6.4.2.5.a comm_out_rng

Η μονάδα comm_out_rng είναι η πιο χρήσιμη για την περίπτωση της εκτέλεσης προσομοιώσεων σε γραφικό περιβάλλον και με ροή χρόνου. Για την μέση συνολική εξασθένιση δημιουργεί τυχαία δείγματα ισχύος που φτάνει στο δέκτη, βάσει μιας τυπικής απόκλισης. Η δήλωση της C++ τάξης της μονάδας είναι:

```
class comm_out_rng : public cSimpleModule
{
   public:
      comm_out_rng();
      virtual ~comm_out_rng();
   protected:
      virtual void initialize();
      virtual void finish();
      virtual void handleMessage(cMessage *msg);
      virtual double received_power_dBm( cMessage *msg ) throw();
      virtual bool delivery_authorization( cMessage* msg )
                                                        throw();
      virtual double normal_random_variance_generator() throw();
   private:
      int          verbosity;
      double       RS_dBm;
      double       std_dev_dBm;
      bool         foliage;
};
```

[omnetpp-3.3p1/mysims/sim4/comm_out_rng.cc]

Η τάξη καταχωρείται στο Omnet++ με την μακροεντολή Define_Module(comm_out_rng).

6.4.2.5.a.1 *void initialize()*

Στην εκκίνηση της προσομοίωσης αποδίδονται μόνο οι τιμές των μεταβλητών της τάξης:

```
        verbosity = par("verbosity");
        std_dev_dBm = par("std_dev_dBm");
        RS_dBm = par("receiverSensitivity_dBm");
        foliage = par("foliage");
```

[omnetpp-3.3p1/mysims/sim4/comm_out_rng.cc]

6.4.2.5.a.2 *void handleMessage(cMessage *msg)*

Όταν η μονάδα παραλάβει ένα μήνυμα από την κεντρική διαχείριση, αποφασίζει την αποστολή με βάση την απάντηση της μεθόδου *bool delivery_authorization(cMessage* msg)*. Αυτή ελέγχει την εξής συνθήκη:

```
return received_power_dBm( msg )
+ std_dev_dBm * normal_random_variance_generator() > RS_dBm;
```

[omnetpp-3.3p1/mysims/sim4/comm_out_rng.cc]

Το αριστερό μέρος της παραπάνω ανισότητας είναι η έκφραση της τυχαίας τιμής ισχύος που ακολουθεί κανονική κατανομή με μέση τιμή *received_power_dBm(msg)* και με τυπική απόκλιση std_dev_dBm. Η μέθοδος *double received_power_dBm(cMessage*)*, υπολογίζει τη μέση τιμή της ισχύος σύμφωνα με την παρακάτω έκφραση:

```
if(!foliage)
    Pr_dBm = Pt_dBm - pl_mean_dBm;
else if(foliage)
    Pr_dBm = Pt_dBm - pl_mean_dBm - fl_mean_dBm;
```

όπου:

```
Pt_dBm = msg->par("transmitPower_dBm");
pl_mean_dBm = msg->par("mean_path_loss_dBm");
fl_mean_dBm = msg->par("mean_foliage_loss_dBm");
```

[omnetpp-3.3p1/mysims/sim4/comm_out_rng.cc]

Η *double normal_random_variance_generator()* είναι μια μέθοδος που εφαρμόζει τον μετασχηματισμό “Box-Muller” [23] που συναντήσαμε και στην υλοποίηση του Shawn [§5.3.2.a.10]. Εδώ χρησιμοποιείται η λειτουργία *uniform(0,1)* του Omnet++ [17], προκειμένου να παίρνουμε ζεύγη τιμών ομοιόμορφης κατανομής στο διάστημα (0,1). Ο μετασχηματισμός στη βασική του μορφή είναι:

```
sqrt(-2.0 * log(uniform(0,1))) * cos(2.0 * PI * uniform(0,1));
```

[omnetpp-3.3p1/mysims/sim4/comm_out_rng.cc]

Αν τελικά η μέθοδος *bool delivery_authorization(cMessage* msg)* δώσει θετικό αποτέλεσμα, τότε το μήνυμα στέλνεται στον παραλήπτη του. Η αποστολή υλοποιείται με την μέθοδο *sendDirect(cMessage* msg, double delay, cGate* gate)*, που χρησιμοποιήθηκε και για την αποστολή μηνυμάτων από τους αισθητήρες στο rf_module. Αν όμως η απάντηση είναι αρνητική και η ευαισθησία του δέκτη είναι μεγαλύτερη από την ισχύ εκπομπής, τότε το μήνυμα καταστρέφεται. Η μέθοδος *void handleMessage(cMessage*)* υλοποιείται συνολικά στον εξής κώδικα:

```
void comm_out_rng::handleMessage(cMessage *msg)
{
    if(delivery_authorization( msg ))
    {
        int destID = msg->par("destAddress_internal");
        cModule* node[destID];
        node[destID] = NETWORK->submodule("node",destID);
        sendDirect(msg, 0, node[destID]->submodule("processor")-
>gate("fromNetwork"));
    }
    else if(!delivery_authorization(msg ))
        delete msg;
}
```

[omnetpp-3.3p1/mysims/sim4/comm_out_rng.cc]

Όπως φαίνεται η αποστολή γίνεται χωρίς καθυστέρηση στην θύρα ¨fromNetwork" που έχουν προσθέσει οι μονάδες gateway στην εκκίνησή τους.

Η μέθοδος συλλέγει επιπλέον και στατιστικά ροής μηνυμάτων. Δημιουργεί δύο map [8], ένα για τα μηνύματα που απορρίπτονται και ένα για αυτά που αποστέλλονται. Σε αυτά ο κλειδί είναι ο αποστολέας και η τιμή είναι μετρητής μηνυμάτων. Τα στοιχεία χρησιμοποιούνται στην *finish()* μέθοδο για εκτύπωση.

6.4.2.5.a.3 *finish()*

Σε αυτή τη μέθοδο αξιοποιούνται τα στατιστικά για τα μηνύματα που συλλέχθηκαν σε map container στην *void handleMessage(cMessage*).* Αυτά δίνουν τον συνολικό αριθμό μηνυμάτων που έστειλε κάθε κόμβος προς όλους τους πιθανούς παραλήπτες, τον αριθμό αυτών που εστάλησαν επιτυχώς και τον αριθμό όσων απέτυχαν να σταλούν και καταστράφηκαν στο μοντέλο. Τα μηνύματα αυτά δεν είναι τα μηνύματα που δημιούργησε κάθε κόμβος, αλλά τα αντίγραφα αυτών που δημιουργήθηκαν στην comm_in μονάδα.

6.4.2.5.b comm_out_probability

Η μονάδα comm_out_probability δεν ενδείκνυται για διαρκή προσομοίωση, αλλά για μοναδική εξέταση κάθε σύνδεσης σε ένα πέρασμα. Η μονάδα δεν εισάγει κάποια τυχαιότητα όπως η comm_out_rng. Αντίθετα υπολογίζει την πιθανότητα η ισχύς που φτάνει στο δέκτη να είναι μεγαλύτερη της ευαισθησίας αυτού. Η πιθανότητα αυτή προκύπτει με τη συνάρτηση κατανομής της ισχύος (F(-x) = 1 - F(x) = Q(x)), με μέση τιμή την μέση ισχύ σήματος που φτάνει στο δέκτη, και τυπική απόκλιση που ορίζουμε στο ini αρχείο παραμέτρων. Η δήλωση της C++ τάξης της μονάδας είναι:

```
class comm_out_probability : public cSimpleModule
{
   public:
     comm_out_probability();
     virtual ~comm_out_probability();
   protected:
     virtual void initialize();
     virtual void handleMessage(cMessage *msg);
     virtual double received_power_dBm( cMessage *msg ) throw();
     virtual bool delivery_authorization( cMessage *msg )
throw();
     virtual double log_normal_P_shadowing_probability( cMessage
*msg ) throw();
     virtual double F_fn( double z ) throw();
     virtual double Q_fn( double z ) throw();
   private:
     int          verbosity;
     double       std_dev_dBm;
     double       RS_dBm;
     double       P_RS;
     double       CP;
     bool         foliage;
};
```

[omnetpp-3.3p1/mysims/sim4/comm_out_probability.cc]

Η τάξη καταχωρείται στο Omnet++ με την μακροεντολή Define_Module(comm_out_probability).

6.4.2.5.b.1 *void initialize()*

Στην εκκίνηση της προσομοίωσης αποδίδονται μόνο οι τιμές των μεταβλητών της τάξης και ελέγχονται κρίσιμες μη λογικές τιμές:

```
verbosity = par("verbosity");
if(std_dev_dBm==0)
   opp_error("ILLEGAL std_dev_dBm value %s", std_dev_dBm);
   std_dev_dBm = par("std_dev_dBm");
   RS_dBm = par("receiverSensitivity_dBm");
   CP = par("communicationProbability");
if( CP<0 || CP > 1)
   //ERROR HANDLE
foliage = par("foliage");
```

[omnetpp-3.3p1/mysims/sim4/comm_out_probability.cc]

Αν η τυπική απόκλιση είναι μηδενική ή αν η πιθανότητα CP έχει τιμές εκτός του εύρους [0,1], τότε η προσομοίωση τερματίζεται. Η τυπική απόκλιση πρέπει πάντα να είναι μη μηδενική διότι συμμετέχει σε διαιρέσεις ως παρονομαστής. Η παράμετρος "communicationProbability" ορίζεται απευθείας στη μονάδα από το ini αρχείο ρυθμίσεων:

```
sim4.rf_module.comm_out_module.communicationProbability = 0.9
```

[omnetpp-3.3p1/mysims/sim4/omnetpp.ini]

6.4.2.5.b.2 *void handleMessage(cMessage *msg)*

Η *void handleMessage(cMessage*)* μέθοδος διαχείρισης εισερχομένων μηνυμάτων, έχει την ίδια ακριβώς έκφραση με τη μονάδα comm_out_rng που παρουσιάστηκε προηγουμένως. Αυτό που διαφέρει είναι η υλοποίηση της *bool delivery_authorization(cMessage *msg)*. Αυτή ορίζεται ως εξής:

```
bool comm_out_probability::
delivery_authorization(cMessage* msg) throw()
    return log_normal_P_shadowing_probability( msg ) > CP;
```

[omnetpp-3.3p1/mysims/sim4/comm_out_probability.cc]

Συγκρίνεται δηλαδή το μέγεθος της πιθανότητας εισόδου CP="communicationProbability" με την πιθανότητα υπολογισμού της *double log_normal_P_shadowing_probability(cMessage*)*. Αυτή υλοποιείται με τον ίδιο τρόπο που αναλύθηκε και στην υλοποίηση του Shawn [§5.3.2.a.11].

Εν τέλει αν η πιθανότητα υπολογισμού είναι μεγαλύτερη της πιθανότητας εισόδου, το μήνυμα αποστέλλεται στον παραλήπτη του και πραγματοποιείται η σύνδεση sensor --> gateway. Αν όχι καταστρέφεται.

Όμοια με την comm_out_rng μονάδα, έτσι και εδώ συλλέγονται στατιστικά ροής μηνυμάτων με τον ίδιο ακριβώς τρόπο [§6.4.2.5.a.2]. Τα στοιχεία χρησιμοποιούνται στην *finish()* μέθοδο για εκτύπωση. Αλλάζει φυσικά το κριτήριο αποδοχής/απόρριψης μηνυμάτων.

6.4.2.5.b.3 *finish()*

Η μέθοδος υλοποιείται όμοια με την comm_out_rng [§6.4.2.5.a.3]

6.5 omnetpp.ini

Ένα αρχείο omnetpp.ini περιέχει τις τιμές των παραμέτρων της προσομοίωσης [2.8] και για την παρούσα υλοποίηση συντάσσεται με τις ακόλουθες κύριες παραμέτρους:

```
#omnetpp.ini

[General]
ini-warnings = yes
preload-ned-files=*.ned
network = sim4

[Parameters]
sim4.verbosity=2

#----------------------
#  PENTELI PARAMETERS
#----------------------
#include node_locations_penteli.ini
#sim4.numNodes = 32
#----------------------
# KAISARIANI PARAMETERS
#----------------------
include node_locations_kaisariani.ini
sim4.numNodes = 17
#----------------------

sim4.display_X_size = 800
sim4.display_Y_size = 600
```

[omnetpp-3.3p1/mysims/sim4/omnetpp.ini]

Όπως βλέπουμε, ανάμεσα στα άλλα, φορτώνονται όλα τα ned αρχεία του καταλόγου εργασίας, δηλώνεται το δίκτυο και συμπεριλαμβάνεται στην παραμετροποίηση της προσομοίωσης ένα αρχείο συντεταγμένων κόμβων. Τα αρχεία node_locations_*.ini είναι αρχεία με παραμέτρους θέσεις για τους κόμβους. Ακολουθεί απόσπασμα από το node_locations_penteli.ini για τον κόμβο 701:

```
*.node[1].nodeID = 701
*.node[1].nodeType = "sensor"
*.node[1].xCoor = 487104.900
*.node[1].yCoor = 4213917.470
*.node[1].zCoor = 464.005
```

[omnetpp-3.3p1/mysims/sim4/node_locations_penteli.ini]

Ο αστερίσκος στην αρχή της διαδρομής των παραπάνω παραμέτρων μας επιτρέπει να χρησιμοποιούμε το αρχείο αυτό σε όλα τα δίκτυα που έχουν μονάδες κόμβων node. Μπορούμε φυσικά να δηλώσουμε και ρητά το όνομα του δικτύου στη διαδρομή της κάθε παραμέτρου. Επιπλέον συμπεριλαμβάνεται στο αρχείο συντεταγμένων και η παράμετρος τύπου κόμβου 'nodeType', όπως και η ταυτότητα κάθε κόμβου 'nodeID', σύμφωνα με τη δική μας αρίθμηση.

Επιπλέον το omnetpp.ini περιέχει και τις τιμές για τις επιπλέον παραμέτρους που εισάγει το μοντέλο επικοινωνίας rf_module. Ανάμεσα σε αυτές είναι και οι τιμές των παραμέτρων των τύπων υπομονάδων που θα χρησιμοποιηθούν. Παρουσιάζονται με τις τιμές του μοντέλου επικοινωνίας που εξετάζουμε στην εκτέλεση του 8ου Κεφαλαίου:

```
#-------------
#RF MODEL PARAMETERS
#-------------
sim4.rf_module.useRF = 1
sim4.rf_module.frequency_GHz = 0.433
sim4.rf_module.transmitPower = 10
sim4.rf_module.transmitPowerUnit = "mW"
sim4.rf_module.receiverSensitivity_dBm = -112
sim4.rf_module.transmitterAntennaGain_dB = 2
sim4.rf_module.receiverAntennaGain_dB = 2
sim4.rf_module.foliage = true
sim4.rf_module.leaf = 1
sim4.rf_module.foliagePathFactor = 0.5
sim4.rf_module.receiverHeight_m = 1.5
sim4.rf_module.transmitterHeight_m = 1.5
sim4.rf_module.std_dev_dBm = 0

#RF SUBMODULE TYPE PARAMETER ASSIGNMENT
sim4.rf_module.core_ModuleType = "core1"
sim4.rf_module.path_loss_ModuleType = "multipath_approximation"
sim4.rf_module.foliage_loss_ModuleType = "weissberger"
sim4.rf_module.comm_in_ModuleType = "comm_in1"
sim4.rf_module.comm_out_ModuleType = "comm_out_rng"

#RF SUBMODULES PARAMETERS
sim4.rf_module.pl_module.log_distance_exponent = 2.1
sim4.rf_module.comm_out_modul.communicationProbability = 0.90
```

[omnetpp-3.3p1/mysims/sim4/omnetpp.ini]

Castalia

ΥΛΟΠΟΙΗΣΗ:

Το Castalia είναι ένας ολοκληρωμένος προσομοιωτής ασυρμάτων δικτύων αισθητήρων, σε αντίθεση με το Omnet++ [Κεφ. 2] και το Shawn [Κεφ. 1], που αποτελούν ένα σύνολο εργαλείων και βιβλιοθηκών για την κατασκευή προσομοιωτών. Ενώ δηλαδή στην περίπτωση του Omnet++ έπρεπε εξολοκλήρου να κατασκευαστούν όλες οι λειτουργικές μονάδες του δικτύου από το μηδέν, το Castalia διαθέτει έτοιμες υλοποιήσεις σε όλα τα επίπεδα. Από την άλλη στο Shawn είναι απαραίτητο να υλοποιήσει κανείς εργασίες και επεξεργαστές για κάθε συσκευή ενός κόμβου, λειτουργίες που ήδη διαθέτει σε απλές μονάδες το Castalia. Η επέμβαση στο Castalia έχει σχεδόν αποκλειστικά τον χαρακτήρα της τροποποίησης.

7. Τροποποίηση Castalia

Στα δίκτυα που εξετάζονται στην εργασία αυτή, απαιτείται η συνθήκη να στέλνουν μηνύματα επικοινωνίας μόνο οι αισθητήρες (sensors), και να παραλαμβάνουν μόνο οι πύλες (gateways). Κάθε μορφή αμφίδρομης επικοινωνίας είναι μη ρεαλιστικό σενάριο. Στο Castalia οι υπάρχουσες μονάδες είναι προγραμματισμένες στα πρότυπα αμφίδρομης επικοινωνίας. Όλοι οι κόμβοι δηλαδή είναι αποστολείς και παραλήπτες μηνυμάτων. Η πολιτική αυτή, πέρα από δεν ευνοεί τη μελέτη και συνεπώς είναι απαραίτητο να τροποποιήσουμε την λειτουργία του προσομοιωτή, σχετικά με την διαφοροποίηση του ρόλου αισθητήρα/πύλης. Η επέμβαση θα γίνει σε δύο μονάδες:

- Ασύρματο Κανάλι (Wireless_Channel simple module)
- Μονάδα Εφαρμογής (Application simple module)

Όπως και στην υλοποίηση του Shawn [Κεφ. 5], όπου τροποποιήθηκε το μοντέλο επικοινωνίας [§5.3] και δημιουργήθηκαν δύο ήδη επεξεργαστή [§5.6], ένα για τους αισθητήρες και ένα για τις πύλες, με τον ίδιο τρόπο προσαρμόζουμε το Castalia στις ανάγκες των δικτύων που μελετάμε. Το Ασύρματο Κανάλι τροποποιείται ώστε να μην προωθεί μηνύματα προς κόμβους-αισθητήρες, η μονάδα Αισθητήρα προγραμματίζεται ώστε να μην δέχεται μηνύματα δεδομένων από το δίκτυο, και η μονάδα Πύλης ώστε να μη στέλνει μηνύματα δεδομένων στο δίκτυο. Μπορεί κανείς εύλογα να θεωρήσει ως περιττή την τροποποίηση της μονάδας Αισθητήρα, δεδομένου ότι το Ασύρματο Κανάλι αποκλείει την περίπτωση αποστολής μηνύματος δεδομένων σε αυτήν. Πραγματοποιείται ούτως ή άλλως για να υπάρχει η δυνατότητα αξιοποίησης και του προεπιλεγμένου μοντέλου, για εξέταση φαινομένων παρεμβολής (INTERFERENCE & COLLISION).

7.1 Τροποποίηση Μονάδας Εφαρμογής - Application simple module Modification

Η μονάδα εφαρμογής ορίζεται με παράμετρο τύπου μονάδας [§2.6.1.3.1]. Επομένως έχουμε τη δυνατότητα να δημιουργήσουμε δύο παρόμοιες μονάδες στη πλαίσια του ορισμού τους στη NED γλώσσα, με διαφορετική όμως λειτουργία και ορισμό τάξης στη C++. Ορίζουμε τη μονάδα και τάξη sensor για την υλοποίηση της εφαρμογής των αισθητήρων, και αντίστοιχα την gateway για τις πύλες. Κριτήριο επιλογής είναι η παράμετρος appModuleName

7.1.1 Μονάδα Εφαρμογής Αισθητήρα - sensor simple module

Για την προσομοίωση των αισθητήρων δημιουργούμε απλή μονάδα Εφαρμογής, με τίτλο sensor. Η μονάδα δημιουργείται στα πρότυπα της μονάδας εφαρμογής connectivityMap. Στην ουσία πρόκειται για την ίδια μονάδα, χωρίς τις υποδομές διαχείρισης μηνυμάτων δεδομένων από το δίκτυο. Οι κύριες τροποποιήσεις είναι:

- msg->kind() == APP_DATA_PACKET

Η *void handleMessage(cMessage* msg)* της μονάδας εφαρμογής, όταν παραλαμβάνει ένα τέτοιο μήνυμα, συνήθως καταγράφει τα στοιχεία αποστολής και τα δεδομένα που περιέχει. Επειδή θέλουμε και με το προεπιλεγμένο κανάλι να έχουμε απόρριψη αυτών των μηνυμάτων στους αισθητήρες, θέτουμε ως αντίδραση την καταστροφή του εισερχόμενου μηνύματος:

```
ase APP_DATA_PACKET:
{
      break;
}
//...
delete msg;
```

[Castalia-1.3/src/Node/Application/sensor/sensor_ApplicationModule.cc]

- APP_TIMER_1

Κατά την εκκίνηση της μονάδας αποστέλλεται μήνυμα αυτοελέγχου APP_TIMER_1 (selfMessage) [§3.1.2.2], προκειμένου να ξεκινήσει ο κύκλος εργασιών αποστολής μηνυμάτων:

```
case APP_NODE_STARTUP:
{
      scheduleAt(simTime(), new App_ControlMessage("timer 1",
                                                   APP_TIMER_1));
      //...
      break;
}
```

[Castalia-1.3/src/Node/Application/sensor/sensor_ApplicationModule.cc]

Με την παραλαβή του μηνύματος ελέγχου αυτού, η μονάδα εκκινεί τη διαδικασία αποστολής και αριθμεί κάθε αποστολή με αύξουσα σειρά (serialNumber). Ο αριθμός της αποστολής καταγράφεται στο μήνυμα με τη μέθοδο *send2NetworkDataPacket(const char *destID, int data, int pckSeqNumber)* [§3.1.2.4] στη παράμετρο δεδομένων (data). Τέλος προγραμματίζεται νέο μήνυμα ελέγχου APP_TIMER_1 για την επόμενη αποστολή:

```
simtime_t now = simTime();
int currentNodeTx = ((int)floor(now / txInterval_perNode))
                    %totalSNnodes;
if( (self == currentNodeTx) && (packetsSent <
     SENT_PACKETS_PER_NODE_PER_LEVEL) )
{
      send2NetworkDataPacket(BROADCAST, serialNumber, 1);
      serialNumber++;
      packetsSent++;
}
scheduleAt(now + DRIFTED_TIME(INTER_PACKET_SPACING),
           new App_ControlMessage("timer1", APP_TIMER_1));
```

[Castalia-1.3/src/Node/Application/sensor/sensor_ApplicationModule.cc]

- *void send2NetworkDataPacket(const char *destID, int data, int pckSeqNumber)*

Η μέθοδος αυτή χρησιμοποιείται για την αποστολή μηνυμάτων στο δίκτυο [§3.1.2.4]. Εδώ ενσωματώνουμε επιπλέον σε κάθε μήνυμα την διεύθυνση/ταυτότητα του κόμβου, σύμφωνα με την δική μας αρίθμηση. Για το σκοπό αυτό προσθέτουμε παράμετρο srcID στο μήνυμα sensor_DataPacket:

```
cplusplus {{
#include "App_GenericDataPacket_m.h"
}};
class App_GenericDataPacket;
message sensor_DataPacket extends App_GenericDataPacket
{
   fields:
      int data;
      int srcID;
};
```

[Castalia-1.3/src/Node/Application/sensor/sensor_DataPacket.msg]

Στη παράμετρο αυτή αποδίδουμε τιμή με την *setSrcID(int)* μέθοδο της cMesage τάξης, η οποία κατασκευάζεται αυτόματα με το εργαλείο opp_msgc του Omnet++. Την τιμή την αντλούμε από τη σύνθετη μονάδα του κόμβου που περιέχει τον επεξεργαστή:

```
self_nodeID = parentModule()->par("nodeID");

sensor_DataPacket *packet2Net;
packet2Net = new sensor_DataPacket("Application Packet
Application->Mac", APP_DATA_PACKET);

packet2Net->setSrcID(self_nodeID);
```

[Castalia-1.3/src/Node/Application/sensor/sensor_ApplicationModule.cc]

Η μονάδα δεν τροποποιείται περαιτέρω, διατηρώντας τις υποδομές επικοινωνίας με τη Συσκευή Αισθητήρα, καθώς και με τη Διαχείριση Πόρων. Η λειτουργία της συνοψίζεται στα εξής βήματα:

- Εκκίνηση μονάδας: Αποστολή μηνυμάτων εκκίνησης (selfMessage APP_NODE_STARTUP)
- Παραλαβή μηνύματος εκκίνησης και προγραμματισμός χρονοδιακόπτη APP_TIMER_1
- Παραλαβή APP_TIMER_1: αποστολή μηνύματος στο δίκτυο και προγραμματισμός νέου APP_TIMER_1

7.1.2 Μονάδα Εφαρμογής Πύλης - gateway simple module

Η απλή μονάδα Πύλη (gateway) είναι και αυτή βασισμένη στην μονάδα Εφαρμογής connectivityMap, με τη διαφορά ότι αφαιρείται η δυνατότητα αποστολής μηνυμάτων στο δίκτυο, και τροποποιείται η διαχείριση των εισερχομένων μηνυμάτων δεδομένων:

- APP_NODE_STARTUP

Τα μηνύματα εκκίνησης [§3.1.2.2] είναι περιττά σε αυτή τη μονάδα, εφόσον έχει παθητικό ρόλο. Συνεπώς δεν δημιουργούνται, δεν προγραμματίζονται και δεν αξιοποιούνται κατά κανένα τρόπο. Σε περίπτωση όμως που η μονάδα για κάποιο λόγο έχει απενεργοποιηθεί και θέλουμε να εκκινήσει πάλι, επιτρέπουμε την διαχείριση του τύπου αυτού μηνύματος ώστε να γίνει πάλι ενεργή:

```
case APP_NODE_STARTUP:
{
      disabled = 0;
      break;
}
```

[Castalia-1.3/src/Node/Application/gateway/gateway_ApplicationModule.cc]

- APP_DATA_PACKET

Τα μηνύματα δεδομένων [§3.1.2.2] που παραλαμβάνει ο κόμβος από το δίκτυο, είναι η πρώτη ύλη της μονάδας αυτής. Η *void handleMessage(cMessage* msg)* εφόσον παραλάβει ένα μήνυμα εκτελεί τις εξής λειτουργίες:

I. Εξετάζεται αν το μήνυμα είναι τύπου sensor_DataPacket, δηλαδή μήνυμα δεδομένων της μονάδας Εφαρμογής Αισθητήρα.

```
sensor_DataPacket *rcvPacket;
rcvPacket = check_and_cast<sensor_DataPacket *>(msg);
```

[Castalia-1.3/src/Node/Application/gateway/gateway_ApplicationModule.cc]

II. Αποδίδονται σε τοπικές μεταβλητές τα δεδομένα (data) του μηνύματος:

```
int theData = rcvPacket->getData();
```

[Castalia-1.3/src/Node/Application/gateway/gateway_ApplicationModule.cc]

Όπως παρουσιάστηκε στην υλοποίηση του sensor [§7.1.1], η τιμή της παραμέτρου (data) είναι ο αύξων αριθμός μηνύματος προς το δίκτυο, για τον κάθε αποστολέα.

III. Αποδίδεται σε τοπική μεταβλητή η ισχύς σήματος που είναι ενσωματωμένη στο μήνυμα (rssi) και η διεύθυνση του (srcID):

```
double rssi = rcvPacket->getRssi();
int msgSenderID = rcvPacket->getSrcID();
```

[Castalia-1.3/src/Node/Application/gateway/gateway_ApplicationModule.cc]

IV. Ενημερώνεται ο πίνακας γειτόνων με τα δεδομένα και τον αποστολέα αυτών

```
updateNeighborTable(msgSenderID, theData);
```

[Castalia-1.3/src/Node/Application/gateway/gateway_ApplicationModule.cc]

- *void updateNeighborTable(int msgSenderID, int theData)*

Ο πίνακας γειτόνων δημιουργείται σε διάνυσμα:

```
vector <neighborRecord> neighborTable;

struct neighborRecord
{
      int id;
      int timesRx;
      int receivedPackets;
};
```

[Castalia-1.3/src/Node/Application/gateway/gateway_ApplicationModule.h]

Σε αυτόν καταγράφεται η ταυτότητα κάθε γείτονα και ο αριθμός μηνυμάτων που παρέβαλε από αυτόν το gateway. Επίσης για κάθε γείτονα καταγράφεται και ο αύξων αριθμός μηνύματος, για το σύνολο των μηνυμάτων που έχει στείλει:

```
if( (theData >= 0) && (theData <
     SENT_PACKETS_PER_NODE_PER_LEVEL) )
neighborTable[pos].receivedPackets++;
```

[Castalia-1.3/src/Node/Application/gateway/gateway_ApplicationModule.cc]

όπου:

SENT_PACKETS_PER_NODE_PER_LEVEL=100

- void finish()

Στη μέθοδο τερματισμού της μονάδας εκτυπώνονται όλα τα στοιχεία του κύκλου εργασίας της. Συγκεκριμένα τυπώνονται οι γείτονες κάθε κόμβου, ο αριθμός των μηνυμάτων που έστειλαν, και η μέση τιμής της ισχύος λήψης από τον καθένα τους.

7.2 Τροποποιημένο Ασύρματο Κανάλι - WChannel_Custom simple module

Το υπάρχον Ασύρματο Κανάλι επικοινωνίας [§3.3] μπορεί να συνεργαστεί με τις νέες μονάδες Εφαρμογής, παρότι δεν εξετάζει το είδος των κόμβων. Ωστόσο δεν είναι ορθή αυτή η πολιτική, διότι δημιουργούνται και αποστέλλονται πολλαπλάσια μηνύματα από αυτά που αξιοποιούνται από τους μοναδικούς πραγματικούς παραλήπτες, δηλαδή τις πύλες.

Όπως αναπτύχθηκε στην παράγραφο §7.1.1, οι μονάδες Εφαρμογής sensor δεν αξιοποιούν εισερχόμενα μηνύματα APP_DATA_PACKET. Όμως η προώθηση ενός τέτοιου μηνύματος σε ένα κόμβο περιλαμβάνει και τις εξής ενέργειες:

- Έλεγχος του μέσου από το Radio (WC_CARRIER_SENSED)
- Ενημέρωση του Radio για αποστολή από το Wireless_Channel (WC_PKT_BEGIN_TX)
- Απάντηση του Radio στο Wireless_Channel ότι είναι έτοιμο για παραλαβή

- Εκκίνηση Αποστολής WC_PKT_END_TX
- Παραλαβή από το Radio του WC_PKT_END_TX
- Ενημέρωση του MAC από το Radio για μεταβίβαση του πακέτου
  -
  -
  -
- Ενημέρωση του sensor από το Network για το πακέτο APP_DATA_PACKET
- Παραλαβή και καταστροφή

Όπως είναι εμφανές χρησιμοποιώντας το αρχικό Ασύρματο Κανάλι (WChannel_Interference) γίνεται τεράστια κατασπατάληση πόρων με την αποστολή μηνυμάτων σε κόμβους όπου η μονάδα εφαρμογής είναι sensor. Επιπλέον κάθε αποστολή καταγράφεται ως γεγονός από τη μονάδα που έστειλε μήνυμα προς sensor, παρότι δεν το υπολογίζουμε. Είναι επομένως απαραίτητο να τροποποιηθεί το Ασύρματο Κανάλι ώστε να μην αποστέλλονται μηνύματα προς κόμβους που είναι τύπου αισθητήρα (nodeType==sensor).

Δεδομένου ότι το Ασύρματο Κανάλι ορίζεται με παράμετρο τύπου μονάδας, δημιουργούμε ένα αντίγραφο του WChannel_Interference, το οποίο ονομάζουμε WChannel_Custom. Σε αυτό επεμβαίνουμε και ορίζουμε εκτίμηση της παραμέτρου nodeType κάθε κόμβου πριν την αποστολή. Όταν εξετάζονται οι παραλήπτες μηνυμάτων από την λίστα ενεργών κόμβων, περιορίζεται η επιλογή σε αυτούς που είναι τύπου πύλης (nodeType==gateway).

Επιπλέον υλοποιούμε το μοντέλο υπολογισμού εξασθένισης Ελεύθερου Χώρου (FSP) [§4.1.2.1], για να υπολογίζουμε την εξασθένιση αναφοράς L0 βάσει των εκάστοτε συνθηκών μετάδοσης. Για το σκοπό αυτό εισάγονται και οι παράμετροι της συχνότητας μετάδοσης και τα κέρδη των κεραιών πομπού και δέκτη:

```
simple WChannel_Custom

   parameters:
     printDebugInfo:bool,
     printStatistics:bool,
     useNodeType: bool,
     pathLossExponent: const,// how fast is the signal
                             // strength fading
     PLd0: const, // path loss at reference distance d0 (in dBm)
     d0:const, // reference distance d0 (in meters)
     Pld0_use_FSP: bool, // calculate reference loss with free
                            space propagation model
     f_GHz: const, // frequency in GHz for FSP calculation
     Gt_dB: const, // transmitter antenna gain
     Gr_dB: const, // receiver antenna gain
```

[Castalia-1.3/src/Wireless_Channel/WChannel_Custom/WChannel_Custom.h]

Η αρχική στατική δήλωση δεν καταργείται, αλλά εισάγεται παράμετρος επιλογής ανάμεσα σε στατικό ορισμό της εξασθένισης αναφοράς και στον υπολογισμό της με το FSP μοντέλο

7.3. omnetpp.ini

Στο omnetpp.ini του νέου δικτύου, ορίζουμε τις νέες μονάδες εφαρμογής, και τις επιπλέον παραμέτρους του ασύρματου καναλιού. Στις περιοχές που εξετάζουμε ο πρώτος κόμβος είναι η πύλη και οι υπόλοιποι οι αισθητήρες. Οι ρυθμίσεις αυτές είναι:

```
#Application Module
SN.node[0].appModuleName = "gateway_ApplicationModule"
SN.node[1..].appModuleName = "sensor_ApplicationModule"
#...
SN.node[0].nodeApplication.applicationID = "gateway"
SN.node[1..].nodeApplication.applicationID = "sensor"

#Wireless Channel extra parameters
SN.wirelessChannel.PLd0_use_FSP = true
SN.wirelessChannel.f_GHz = 0.433
SN.wirelessChannel.Gr_dB = 2
SN.wirelessChannel.Gt_dB = 2
```

Όπως βλέπουμε ορίζεται για το node[0] Εφαρμογή gateway και για τα υπόλοιπα sensor [§2.8].

Πέρα από αυτές τις παραμέτρους, οι υπόλοιπες ρυθμίσεις γίνονται στα σχετικά αρχεία αναλόγως των επιπλέον αναγκών [§3.4]. Το αρχείο συντεταγμένων που χρησιμοποιείται στο Castalia είναι ακριβώς το ίδιο που χρησιμοποιείται και στο Omnet++.

Προσομοίωση & Εκτέλεση Σεναρίων

ΑΞΙΟΛΟΓΗΣΗ & ΑΠΕΙΚΟΝΙΣΗ:

Στο κεφάλαιο αυτό εκτελούνται σενάρια προσομοίωσης για δύο περιοχές στη Καισαριανή και Πεντέλη. Για τις περιοχές αυτές διαθέτουμε ανεξάρτητα δεδομένα μετρήσεων ισχύος λήψης, μεταξύ ασυρμάτων συσκευών γνωστών ονομαστικών στοιχείων. Είναι επίσης γνωστές οι θέσεις στις οποίες έγιναν οι παρατηρήσεις των μετρήσεων και σχόλια για αυτές. Τα λίγα αυτά δεδομένα θα χρησιμοποιήσουμε για να αξιολογήσουμε, στο βαθμό που επιτρέπεται, τους τρεις προσομοιωτές που υλοποιήθηκαν στο Shawn, το Omnet++ και το Castalia. Επίσης συγκρίνονται οι διαφορετικές προσεγγίσεις μεταξύ τους και αναλύονται τα αποτελέσματα σε επίπεδο κόμβου. Τέλος χρησιμοποιούνται τα αποτελέσματα που δίνουν καλύτερη σύγκλιση ανάμεσα σε τιμές υπολογισμού και μετρήσεις, για την πραγματοποίηση οπτικής απεικόνισης στα Shawn και Omnet++.

8. Εκτέλεση Σεναρίων & Αξιολόγηση Προσομοιωτών

Στα προηγούμενα κεφάλαια παρουσιάστηκαν και αναλύθηκαν τα πακέτα λογισμικού προσομοίωσης Shawn, Omnet++ και Castalia [Κεφ. 1, 2, 3]. Εν συνεχεία δημιουργήθηκαν δομές και προσομοιωτές ώστε να μπορούμε να εξετάσουμε την ασύρματη επικοινωνία σε ένα ασύρματο δίκτυο αισθητήρων [Κεφ 5,6,7]. Το κριτήριο εκτίμησης της επικοινωνίας είναι η εξασθένιση της ισχύος σήματος στη μετάδοση ραδιοκυμάτων, όπως αναλύθηκε στο κεφάλαιο 4. Στο σημείο αυτό βάζουμε σε δοκιμή τους προσομοιωτές αυτούς, πάνω σε τύπους δεδομένων που εναρμονίζονται με το πλαίσιο υλοποίησης.

Τα δίκτυα αισθητήρων μπορούν να είναι πολλών ειδών. Μπορούμε να έχουμε αισθητήρες ατμοσφαιρικών συνθηκών (πίεση, υγρασία, θερμοκρασία), αισθητήρες κίνησης, αισθητήρες σεισμικών δονήσεων κλπ. Κάθε ποσότητα που παρατηρείται σε τυχαίες τιμές πάνω σε μεγάλη έκταση, ή και μικρή κατά περιπτώσεις, είναι υποψήφια ποσότητα μέτρησης. Συνήθως αυτό που καθορίζει την προσέγγιση στην υλοποίηση δεν είναι τόσο η ποσότητα όσο η ίδια η έκταση. Διαφορετικά θα εξετάζαμε για παράδειγμα την προσομοίωση αισθητήρων πυρασφάλειας σε εσωτερικούς χώρους, και διαφορετικά στο ύπαιθρο, παρότι πρόκειται ακριβώς για την ίδια φυσική ποσότητα, δηλαδή τη ραγδαία αύξηση της θερμοκρασίας ή την παρατήρηση ακραίων τιμών.

Χώρος αναφοράς για τα δίκτυα αισθητήρων που εξετάζουν οι προσομοιωτές που δημιουργήθηκαν, είναι το ύπαιθρο και ειδικότερα οι εκτάσεις με κυριαρχία βλάστησης. Τα μοντέλα εξασθένισης που εξετάζουν την επικοινωνία είναι ακριβώς επιλεγμένα για το σκοπό αυτό. Δεν προσφέρονται για μελέτη αστικού χώρου ή κλειστών χώρων.

8.1 Δοκιμή & Σύγκριση με Πειραματικά Δεδομένα Μετρήσεων

Προκειμένου να ολοκληρωθεί η εικόνα για τους προσομοιωτές πρέπει να εξετάσουμε ένα σενάριο εκτέλεσης σε μια τέτοια περιοχή, ανάλογη των προδιαγραφών. Για το σκοπό αυτό διατέθηκαν δεδομένα μέτρησης λαμβανόμενης ισχύος σήματος για δασική έκταση στις περιοχές της Πεντέλης και της Καισαριανής. Σκοπός της δοκιμής είναι να προσομοιώσουμε όσο γίνεται πιο πιστά τις συνθήκες της περιοχής και της μετάδοσης σήματος ανάμεσα στους κόμβους, ώστε να εξετάσουμε κατά πόσο οι υλοποιήσεις μας είναι αποδοτικές για τη μελέτη τέτοιων περιοχών και δικτύων.

Τα δεδομένα αυτά συλλέχθηκαν με ζεύγος πομπού-δέκτη, γνωστών στοιχείων εκπομπής και λοιπών προδιαγραφών. Οι τιμές της λαμβανόμενης ισχύος για κάθε θέση, μαζί με τα σχόλια που παρέχονταν στα δεδομένα, παρουσιάζονται στους παρακάτω πίνακες:

ΚΑΙΣΑΡΙΑΝΗ		
ΣΗΜΕΙΟ	ΛΗΨΗ ΣΗΜΑΤΟΣ (dBm)	ΠΑΡΑΤΗΡΗΣΕΙΣ
130	-20	Θέση Δέκτη
131	-54	Δίπλα από τον δέκτη
132	-64	
133	-60	Οπτική επαφή πομπού - δέκτη
134	-83	
135	-85	Υψηλή τιμή εξασθένησης μετά τα 100μ.
136	-91	Εξαιρετικά ασθενές σήμα
137	-95	Εξαιρετικά ασθενές σήμα - οριακό
138	ΟΧΙ	Μεγάλη υψομετρική διαφορά - απόσταση
139	ΟΧΙ	Μεγάλη υψομετρική διαφορά - απόσταση
140	ΟΧΙ	Μέγιστη υψομετρική διαφορά - απόσταση
141	-86	Μέγιστη απόσταση λήψης σήματος
142	-66	

143	-72	
144	-61	Οπτική επαφή πομπού - δέκτη
145	-65	Χωρίς οπτική επαφή πομπού - δέκτη
146	-57	Χωρίς υψομετρική διαφορά

Πίνακας 8.1

ΠΕΝΤΕΛΗ		
ΣΗΜΕΙΟ	ΛΗΨΗ ΣΗΜΑΤΟΣ (dBm)	ΠΑΡΑΤΗΡΗΣΕΙΣ
700	-33	Θέση Δέκτη
701	-64	Δίπλα από τον δέκτη
702	-78	Καθόλου βλάστηση - δρόμος
703	-86	Χαμηλή βλάστηση
704	-90	Το πιο ασθενές σήμα
705	ΌΧΙ	Κοντά στον πυλώνα
706	-85	
707	-86	Μέγιστη απόσταση λήψης σήματος
708	ΌΧΙ	Πέρα από τις Γ.Υ.Τ. - Καμία λήψη
709	ΌΧΙ	Πέρα από τις Γ.Υ.Τ. - Καμία λήψη
710	ΌΧΙ	Πέρα από τις Γ.Υ.Τ. - Καμία λήψη
711	ΌΧΙ	Πέρα από τις Γ.Υ.Τ. - Καμία λήψη
712	ΌΧΙ	Πέρα από τις Γ.Υ.Τ. - Καμία λήψη
713	ΌΧΙ	Πέρα από τις Γ.Υ.Τ. - Καμία λήψη
714	-88	Θέση ακριβώς κάτω από πυλώνα
715	-80	Καθόλου βλάστηση
716	-87	Καθόλου βλάστηση - κοντά στις Γ.Υ.Τ.
717	ΌΧΙ	Καθόλου βλάστηση - κοντά στις Γ.Υ.Τ.
718	-87	Ασθενές σήμα λόγω Γ.Υ.Τ. και απόστασης
719	-86	Ασθενές σήμα λόγω Γ.Υ.Τ. και απόστασης
720	-80	
721	-83	
722	-79	Μικρή υψομετρική διαφορά
723	-79	Μέσα σε όρυγμα
724	-89	Μέσα σε όρυγμα - κοντά σε Γ.Υ.Τ.
725	-88	Μέσα σε όρυγμα - κοντά σε Γ.Υ.Τ.
726	-81	Χαμηλή βλάστηση
727	-75	
728	-80	Ελάχιστη υψομετρική διαφορά
729	-75	Καθόλου βλάστηση - δρόμος
730	-58	Οπτική επαφή πομπού - δέκτη
731	-54	Οπτική επαφή πομπού - δέκτη

Πίνακας 8.2

Για τις περιοχές των παραπάνω δειγμάτων μας δίνονται οι εξής επιπλέον πληροφορίες:

- Καισαριανή (θέση “Καλοπούλα”)

Περιοχή με έντονη πυκνή βλάστηση και μεγάλες υψομετρικές διαφορές

- Πεντέλη

Μικρές υψομετρικές διαφορές, χαμηλή και αραιή βλάστηση. Πολύ χαμηλά επίπεδα λαμβανόμενου σήματος, πιθανόν λόγω των καλωδίων υψηλής τάσης που βρίσκονται στην περιοχή.

Επομένως έχουμε δύο αρκετά διαφορετικές περιοχές τόσο από πλευράς μορφολογίας όσο και υφής του εδάφους. Περιμένουμε συνεπώς να παρουσιάζον σύγκλιση με τις μετρήσεις τα αντίστοιχα μοντέλα που προτείνονται βάσει των χαρακτηριστικών της περιοχής. Για την Καισαριανή που έχει έντονο ανάγλυφο, και υψηλά επίπεδα βλάστησης, αναμένουμε θεωρητικά την μέγιστη σύγκλιση με κάποιο συνδυασμό μοντέλου αναγλύφου [§4.1.3.2] με κάποιο μοντέλο βλάστησης, για υψηλή τιμή εύρους βλάστησης [§4.1.3.1]. Αντίθετα για την Πεντέλη είναι υποψήφιο το μοντέλο Εξασθένισης Διαδρομής Λογαριθμικής Απόστασης [§4.1.2.2] με εξασθένιση βλάστησης για μικρό εύρος. Βέβαια επιφύλαξη διατηρούμε σχετικά με την πληροφορία της παρεμβολής των γραμμών υψηλής τάσης στην εκπομπή.

8.2 Προετοιμασία των Δεδομένων

Τα δεδομένα μετρήσεων μαζί με τα δεδομένα θέσης των κόμβων σε κάθε περιοχή πρέπει να εισαχθούν με κατάλληλη σύνταξη σε αρχεία δεδομένων θέσης και παραμέτρων προσομοίωσης, και για τους τρεις προσομοιωτές. Η διαδικασία χωρίζεται σε δύο στάδια:

1.Μετατροπή συντεταγμένων
2.Σύνταξη αρχείων ρυθμίσεων και XML δεδομένων

8.2.1 Μετατροπή GPS δεδομένων σε ΕΓΣΑ'87

Τα δεδομένα θέσης προέρχονται από αρχεία συντεταγμένων Garmin, στο παγκόσμιο σύστημα αναφοράς WGS '84. Σε πρώτη φάση εξάγουμε από τα αρχεία αυτά τα δεδομένα σε μορφή κειμένου με το λογισμικό μετασχηματισμού αρχείων GPS δεδομένων GPSBabel [19]. Αυτά μετασχηματίστηκαν στο Ελληνικό Γεωδαιτικό Σύστημα Αναφοράς ΕΓΣΑ '87 με χρήση του λογισμικού μετατροπής χαρτογραφικών προβολών και συστημάτων αναφοράς PROJ.4 [20]. Οι συντεταγμένες X,Y και το υψόμετρο h όπως υπολογίστηκαν, παρουσιάζονται στους παρακάτω πίνακες:

ΚΑΙΣΑΡΙΑΝΗ				ΠΕΝΤΕΛΗ			
ΣΗΜΕΙΟ	X	Y	h	ΣΗΜΕΙΟ	X	Y	h
130	482643.80	4201487.85	352.672	700	487105.65	4213915.99	463.005
131	482643.14	4201487.60	352.672	701	487104.90	4213917.47	464.005
132	482670.81	4201489.21	358.672	702	487128.00	4213951.40	466.002
133	482704.89	4201483.99	371.671	703	487118.42	4213980.01	467.001
134	482729.11	4201492.08	377.670	704	487109.99	4214008.41	468.000
135	482756.46	4201510.63	386.669	705	487117.43	4214046.83	468.998
136	482767.32	4201491.29	402.669	706	487109.60	4214085.40	465.997
137	482771.09	4201458.61	412.671	707	487142.09	4214123.69	471.994
138	482798.62	4201415.71	424.672	708	487161.35	4214147.64	474.993
139	482780.99	4201374.82	411.674	709	487202.48	4214183.48	484.990
140	482760.69	4201319.26	407.677	710	487241.74	4214158.39	498.990
141	482728.24	4201365.04	395.676	711	487267.72	4214149.66	502.990
142	482687.42	4201391.20	389.676	712	487287.36	4214147.72	504.990
143	482683.18	4201417.81	377.675	713	487270.11	4214101.55	498.992
144	482668.24	4201426.22	373.675	714	487236.15	4214008.95	488.997
145	482650.15	4201457.26	365.674	715	487231.26	4213983.97	490.999
146	482631.60	4201479.43	370.673	716	487235.45	4213967.84	486.999
				717	487251.63	4213951.73	483.000
				718	487268.46	4213935.86	482.000

Πίνακας 8.3

719	487258.19	4213928.48	476.000
720	487256.01	4213918.43	471.001
721	487263.77	4213895.52	472.002
722	487277.09	4213889.89	477.002
723	487260.79	4213875.99	470.003
724	487235.89	4213875.16	464.003
725	487222.67	4213881.47	462.003
726	487207.04	4213889.66	462.003
727	487200.56	4213903.54	469.003
728	487180.54	4213908.14	468.003
729	487169.85	4213900.72	464.004
730	487140.57	4213901.88	467.004
731	487129.54	4213907.81	467004

Πίνακας 8.4

8.2.2 Αρχεία Συντεταγμένων Προσομοιωτών

Οι συντεταγμένες των κόμβων πρέπει να εισαχθούν σε αρχεία δεδομένων θέσης, σύμφωνα με τις απαιτήσεις κάθε προσομοιωτή.

Το Shawn διαβάζει τις θέσεις των κόμβων από XML αρχεία δεδομένων. Αυτές περιλαμβάνονται σε ετικέτα <location..../> εντός της ετικέτας κάθε κόμβου. Παρατίθεται δείγμα για μερικούς κόμβους της Πεντέλης:

```
<node id="700">
     <location x="487105.650" y="4213915.990" z="463.005" />
     <tag type="string" name="node type" value="station"/>
</node>
<node id="701">
     <location x="487104.900" y="4213917.470" z="464.005" />
     <tag type="string" name="node type" value="node"/>
</node>
<node id="702">
     <location x="487128.000" y="4213951.400" z="466.002" />
     <tag type="string" name="node type" value="node"/>
</node>
```

Το Omnet++ και το Castalia βασίζονται στην ίδια απολύτως σύνταξη στον ορισμό παραμέτρων, μιας και πρόκειται για ίδιους προσομοιωτές. Για να χρησιμοποιήσουμε όμως τα ίδια ακριβώς αρχεία ορισμού συντεταγμένων, θα πρέπει η θέση των κόμβων στην ιεραρχία του δικτύου να είναι η ίδια για τους δύο προσομοιωτές. Στην αντίθετη περίπτωση θα προκύψει σφάλμα.

Έστω για παράδειγμα ότι σε μια προσομοίωση ο κόμβος node βρίσκεται στο 1ο επίπεδο ιεραρχίας, κάτω από το δίκτυο, και οι συντεταγμένες του ορίζονται με παραμέτρους X,Y,Z. Αυτές θα αποδίδονται στο ini αρχείο ρυθμίσεων ως εξής:

network.node.X=value_x
network.node.Y=value_y
network.node.Z=value_z

Αν ο ίδιος κόμβος ανήκει σε υπομονάδα του δικτύου, για παράδειγμα subnet, τότε οι συντεταγμένες του θα δηλώνονται βάσει της νέας ιεραρχίας:

network.subnet.node.X=value_x
network.subnet.node.Y=value_y
network.subnet.node.Z=value_z

Όπως γίνεται αντιληπτό αν χρησιμοποιήσουμε τα δεδομένα της πρώτης περίπτωσης στην δεύτερη δε θα αποδοθούν οι συντεταγμένες στους κόμβους και θα μας ζητήσει το πρόγραμμα να τις εισάγουμε χειροκίνητα.

Και στους δύο προσομοιωτές που προγραμματίστηκαν στο Castalia και το Omnet, οι κόμβοι είναι στο ίδιο επίπεδο ιεραρχίας, οπότε μπορούμε να χρησιμοποιήσουμε τις ίδιες παραμέτρους θέσεις. Ακολουθεί ένα δείγμα αυτών για την περιοχή της Καισαριανής:

```
SN.node[0].xCoor = 482643.800
SN.node[0].yCoor = 4201487.850
SN.node[0].zCoor = 352.672

SN.node[1].xCoor = 482643.140
SN.node[1].yCoor = 4201487.600
SN.node[1].zCoor = 352.672

SN.node[2].xCoor = 482670.810
SN.node[2].yCoor = 4201489.210
SN.node[2].zCoor = 358.672
```

8.2.3 Αρχεία Ρυθμίσεων Προσομοιωτών & Τιμές Παραμέτρων

Για να εκτελέσουμε τη προσομοίωση πρέπει να ετοιμαστούν τα αρχεία ρυθμίσεων για τους τρεις προσομοιωτές. Αυτά περιλαμβάνουν τις ιδιαίτερες παραμέτρους κάθε προσομοιωτή και τις τιμές του μέσου και τις προδιαγραφές των συσκευών με τις οποίες καταγράφηκαν οι μετρήσεις ισχύος.

Οι προδιαγραφές που έχουμε στη διάθεσή μας για τις συσκευές των μετρήσεων είναι:

Ισχύς Εκπομπής : 10 mW
Συχνότητα Ραδιομετάδοσης : 433MHz
Κέρδος Κεραίας Πομπού : 2 dB
Κέρδος Κεραίας Δέκτη : 2 dB
Ραδιοευαισθησία Δέκτη : -112 dBm

Τις παραμέτρους εισάγουμε σε αρχεία ρυθμίσεων:

α) Shawn

Στο Shawn δημιουργήθηκε υποδομή ώστε οι παράμετροι να εισάγονται είτε στο αρχείο ρυθμίσεων *.conf είτε σε “RF” XML ετικέτα περιβάλλοντος [§5.3.4.], μαζί με τα υπόλοιπα στοιχεία του μοντέλου επικοινωνίας. Προτιμάται γενικά η δεύτερη επιλογή για την καλύτερη εποπτεία των παραμέτρων. Είναι προτιμότερο το αρχείο ρυθμίσεων να το τροποποιούμε πάνω στις εργασίες και τον αριθμό των γύρων της προσομοίωσης. Η σύνταξη των παραμέτρων στην σχετική ετικέτα του XML αρχείου δεδομένων περιβάλλοντος στο Shawn είναι:

```
<scenario>
   <environment>
     <tag type="group" name="RF">
         <tag type="double" name="f_GHz" value="0.433"/>
         <tag type="double" name="Pt" value="10"/>
         <tag type="string" name="Pt_unit" value="mW"/>
         <tag type="double" name="Receiver_Sensitivity_dBm"
              value="-112"/>
         <tag type="double" name="gain_t_dB" value="2"/>
         <tag type="double" name="gain_r_dB" value="2"/>
         <!-- . . . -->
      </tag>
   </environment>
</scenario>
```

Οι παράμετροι αυτοί ανήκουν στο περιβάλλον επικοινωνίας και όχι στους κόμβους.

Επιπλέον συντάσσουμε αρχείο ρυθμίσεων με τις απαραίτητες εργασίες της προσομοίωσης. Το αρχείο αυτό θα συνοδεύει το εκτελέσιμο στην γραμμή εντολών, ώστε να συγκροτηθεί ο κόσμος και το περιβάλλον προσομοίωσης και να εκτελεστούν οι εργασίες που έχουμε ορίσει.

```
prepare_world env_config=shawn-kaisariani_xyz_.xml
edge_model=simple_nt comm_model=rf_nt verbosity=0
transm_model=reliable

load_world world_in_file=shawn-kaisariani_xyz_.xml snapshot=01
processors=proc_manager

node_type_load_task

simulation max_iterations=2

connectivity_nt verbosity=0 output=false
```

β) Omnet++

Στο Omnet++ οι παράμετροι ορίζονται στο omnetpp.ini αρχείο ρυθμίσεων, και αποδίδονται ευθέως στο RF_Propagation_Module, δηλαδή στο μοντέλο επικοινωνίας, και όχι στους κόμβους. Θα μπορούσαμε με χρήση αστερίσκων (wildcard *) [§2.8] να επιτρέψουμε την χρήση από κάθε μονάδα στο ίδιο επίπεδο ιεραρχίας, δηλαδή και τους κόμβους. Η σύνταξη κατά τους δύο τρόπους είναι:

```
sim4.rf_module.frequency_GHz = 0.433
sim4.rf_module.transmitPower = 10
sim4.rf_module.transmitPowerUnit = "mW"
sim4.rf_module.receiverSensitivity_dBm = -112
sim4.rf_module.transmitterAntennaGain_dB = 2
sim4.rf_module.receiverAntennaGain_dB = 2
sim4.rf_module.receiverHeight_m = 1.5
sim4.rf_module.transmitterHeight_m = 1.5
```

```
sim4.*.frequency_GHz = 0.433
sim4.*.transmitPower = 10
sim4.*.transmitPowerUnit = "mW"
sim4.*.receiverSensitivity_dBm = -112
sim4.*.transmitterAntennaGain_dB = 2
sim4.*.receiverAntennaGain_dB = 2
sim4.*.receiverHeight_m = 1.5
sim4.*.transmitterHeight_m = 1.5
```

γ) Castalia

Στο Castalia οι παράμετροι μπορούν είτε να οριστούν στο omnetpp.ini, είτε σε κάποιο αρχείο παραμέτρων αντίστοιχο της μονάδας για την οποία προορίζονται. Σε κάθε περίπτωση πρέπει να αποδοθούν τιμές στις παραμέτρους βάσει της μονάδας που τις αξιοποιεί. Οι μονάδες αυτές για τις παραμέτρους της εκπομπής είναι το Ασύρματο Κανάλι (wirellessChannel) και η μονάδα Ραδιοεπικοινωνίας (radio). Η σύνταξη για την απόδοση των τιμών στις παραμέτρους είναι:

```
SN.node[*].networkInterface.Radio.receiverSensitivity = -112
SN.node[*].networkInterface.Radio.txPowerLevels = "1 -1 -3 -5 -7
                                                  -10 -15 -25" # in dBm
SN.node[*].networkInterface.Radio.txPowerLevelUsed = 0
SN.wirelessChannel.f_GHz = 0.433
SN.wirelessChannel.Gr_dB = 2
SN.wirelessChannel.Gt_dB = 2
```

Στην περίπτωση του Castalia μιας και χρησιμοποιείται μόνο το μοντέλο Εξασθένισης Λογαριθμικής Απόστασης [§4.1.2.2], δεν αξιοποιούμε το ύψος κεραιών. Επιπλέον στην αρχική υλοποίηση του ασύρματου καναλιού χρησιμοποιείται στατική δήλωση για την εξασθένιση αναφοράς, ενώ στην τροποποίηση έχουμε εισάγει τον υπολογισμό της με το μοντέλο Εξασθένισης Ελεύθερου Χώρου [§4.1.2.1], για απόσταση αναφοράς ένα μέτρο (1m).

Στο Castalia ορίζονται και διάφορες άλλοι παράμετροι για τη μονάδα ραδιοεπικοινωνίας [§3.1.4.3] τις οποίες δεν τροποποιούμε διότι δεν χρησιμοποιούνται στην τροποποίηση του ασύρματου καναλιού επικοινωνίας.

8.3 Εκτέλεση Προσομοιώσεων & Σύγκριση Με Τιμές Μετρήσεων

8.3.1 Shawn

Στο Shawn εφαρμόζουμε προσομοιώσεις με κλήση του εκτελέσιμου. Συνήθως δίνουμε και ως παράμετρο το αρχείο ρυθμίσεων της προσομοίωσης, εκτός αν θέλουμε να βρεθούμε σε γραμμή εντολών και να εισάγουμε χειροκίνητα όλες τις εντολές.

```
root:/usr/local/src/shawn/buildfiles/# ./shawn -f rf_report.conf
```

Το αρχείο ρυθμίσεων rf_report.conf, έχει την ακόλουθη δομή:

```
prepare_world env_config=shawn-kaisariani_xyz_.xml
edge_model=simple_nt comm_model=rf_nt verbosity=3

transm_model=reliable

load_world world_in_file=shawn-kaisariani_xyz_.xml snapshot=01
processors=proc_manager

node_type_load_task

rf_nt_attenuation
```

Στην τελευταία γραμμή του αρχείου διακρίνεται η εργασία rf_nt_attenuation που δημιουργήθηκε για το σκοπό αυτό [§5.6]. Αυτή καλεί με τη σειρά όλες τις μεθόδους υπολογισμού εξασθένισης που έχουν υλοποιηθεί στο 'RF NT' μοντέλο [§5.3.2.α] και υπολογίζει για κάθε ζεύγος αισθητήρα-πύλη όλες τι εξασθενίσεις με όλα τα μοντέλα. Η διαδικασία ξεκινάει βρίσκοντας όλους τους κόμβους που έχουν επεξεργαστή πύλη. Στη συνέχεια για κάθε κόμβο-πύλη υπολογίζονται οι εξασθενίσεις για όλους του αισθητήρες, ανεξάρτητα αν συνδέονται ή όχι. Δεν εξετάζεται η συνδεσιμότητα των κόμβων. Τέλος για κάθε πύλη τυπώνεται αρχείο με τις τιμές αυτές, του οποίου το όνομα ξεκινάει με το όνομα του κόμβου και συμπληρώνεται με την κατάληξη "_attenuation_report.txt". Στο αρχείο τυπώνονται και οι παράμετροι του μοντέλου επικοινωνίας οι σχετικές με τη μετάδοση ραδιοσυχνοτήτων. Οι εγγραφές για κάθε νέα εκτέλεση γράφονται στο τέλος του αρχείου, χωρίς να σβήνουν παλιά δεδομένα.

Παρουσιάζονται τα αποτελέσματα στην περιοχή της Καισαριανής για μία εκτέλεση:

```
------------------------------------------------------------
Simulation RF paramater values:
Frequency: 0.433 Ghz
Transmit Power: 10.00 mW
Receiver Sensitivity: -112 dBm
Transmitter Antenna Gain: 2.0 dB
Receiver Antenna Gain: 2.0 dB
Foliage Path percent: 100 %
Receiver Antenna Height: 1.50 m
Transmitter Antenna Height: 1.50 m
Log-Distance Path Loss Exponent: 2.00
```

[130_attenuation_report.txt]

130	FSP	LD	LD_1m	L50	LMP1	LMP2	W	ITU_R	C235_IN	C235_OUT
131	18.1507	44.8748	18.1507	6.6181	-17.0973	nan	0.2504	1.0027	13.0291	3.8919
132	50.0277	76.7518	50.0277	38.4951	46.6567	47.4579	7.3928	9.0674	48.8314	24.3827
133	57.3137	84.0378	57.3137	45.7811	61.2286	61.3761	12.1065	14.9989	66.0460	37.0878
134	60.1651	86.8892	60.1651	48.6325	66.9315	67.0078	14.6842	18.2643	74.3314	43.7035
135	62.7512	89.4753	62.7512	51.2186	72.1037	72.1457	17.4936	21.8367	82.7413	50.7188
136	63.6741	90.3982	63.6741	52.1415	73.9495	73.9835	18.6214	23.2741	85.9675	53.4861
137	64.3283	91.0524	64.3283	52.7957	75.2579	75.2872	19.4647	24.3501	88.3305	55.5389
138	66.5377	93.2618	66.5377	55.0051	79.6768	79.6943	22.6050	28.3650	96.8010	63.0714
139	66.6279	93.3520	66.6279	55.0953	79.8571	79.8743	22.7434	28.5422	97.1634	63.3995
140	67.7204	94.4445	67.7204	56.1878	82.0421	82.0555	24.4892	30.7796	101.6640	67.5147
141	64.9908	91.7149	64.9908	53.4582	76.5829	76.6080	20.3575	25.4903	90.7894	57.6977
142	62.1858	88.9099	62.1858	50.6532	70.9729	71.0208	16.8367	21.0002	80.8248	49.0946
143	59.6788	86.4029	59.6788	48.1462	65.9589	66.0443	14.2086	17.6610	72.8483	42.4971
144	58.0230	84.7471	58.0230	46.4904	62.6473	62.7725	12.7020	15.7522	68.0166	38.6335
145	51.7661	78.4901	51.7661	40.2334	50.1334	50.6670	8.3160	10.2243	52.4795	26.9488
146	48.5317	75.2558	48.5317	36.9991	43.6647	44.8045	6.6808	8.1771	45.8956	22.3708

Πίνακας 8.5

Εδώ πρέπει να τονισθεί πως ο κόμβος 131 βρίσκεται δίπλα στον κόμβο 130, δηλαδή στο εγγύς πεδίο της κεραίας του με αποτέλεσμα τα περισσότερα μοντέλα να δίνουν τελείως λαθεμένη εξασθένιση. Για το λόγο αυτό το σημείο αυτό εξαιρείται της αξιολόγησης. Το ίδιο εφαρμόζεται και για τον κόμβο 701 στην περιοχής της Πεντέλης, για τον ίδιο ακριβώς λόγο. Επίσης βλέπουμε και τις τελείως άστοχες τιμές που εμφανίζει σε αυτές τις περιπτώσεις το μοντέλο Ανάκλασης Δύο Δρόμων, λόγω της πολύς κοντινής απόστασης, στην οποία το μοντέλο δεν έχει εφαρμογή.

Η εργασία 'rf_nt_attenuation' εκτελείται αρκετές φορές καλύπτοντας ένα ευρύ φάσμα περιπτώσεων. Σε κάθε επανάληψη μεταβάλλονται δύο παράγοντες:

α) Το εύρος βλάστησης
β) Ο εκθέτης της Εξασθένισης Λογαριθμικής Απόστασης

Το εύρος της βλάστησης εξαρτάται από τον παράγοντα foliage_path_factor [§5.3.3] ο οποίος πολλαπλασιάζεται με την απόσταση των κόμβων. Εφόσον για την Περιοχή της Καισαριανή έχουμε πυκνή βλάστηση, εφαρμόζουμε τιμές από 0.5 έως 1. Ο εκθέτης του μοντέλου Log-Distance παίρνει διαδοχικά τις εξής τιμές:

```
N: 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 3.0, 3.5
```

Τα αποτελέσματα εισάγονται σε λογιστικό φύλλο όπου υπολογίζεται η συνολική μέση εξασθένισης σήματος για κάθε πιθανό συνδυασμό εξασθένισης λόγω απόστασης και εξασθένισης βλάστησης. Τα αποτελέσματα αυτά συγκρίνονται με τις ανεξάρτητες μετρήσεις που έχουμε για την περιοχή.

Οι τιμές που υπολογίζονται από την προσομοίωση είναι δεκαδικοί με ακρίβεια 4ου ψηφίου (στο Πίνακα 8.5 για λόγους παρουσίασης τυπώνονται με την ακρίβεια υπολογισμού τεσσάρων ψηφίων), ενώ οι τιμές ισχύος στα δείγματα είναι ακέραιοι, με ακρίβεια μονάδας (1). Δεν έχει νόημα να συγκρίνουμε δεκαδικούς με ακεραίους. Η σύγκριση πρέπει να γίνει στο μικρότερο επίπεδο ακρίβειας, δηλαδή με ακρίβεια μονάδας ακεραίου. Επομένως όλα τα αποτελέσματα εξασθένισης, σαν αυτά του Πίνακα 8.5, μετατρέπονται σε ακέραιη μορφή με στρογγυλοποίηση στον εγγύτερο ακέραιο αριθμό. Όλα τα γραφήματα και οι σχετικοί πίνακες που παρουσιάζουν τις διαφορές αυτές βασίζονται σε αυτή τη στρογγυλοποίηση.

Η σύγκριση των αποτελεσμάτων της προσομοίωσης με τις ανεξάρτητες μετρήσεις, δεν είναι κάποιο ισχυρό κριτήριο αξιολόγησης. Δεν γνωρίζουμε τίποτα παραπάνω για τις μετρήσεις πέρα από την τιμή της ισχύος και τις παρατηρήσεις της μέτρησης. Δεν έχουμε καν δείγμα μετρήσεων, παρά μόνο μία μέτρηση για κάθε σύνδεση. Οι δε μετρήσεις αυτές είναι τελείως ανεξάρτητες μεταξύ τους, διότι παρότι έγιναν με το ίδιο ζεύγος οργάνων, έγιναν υπό διαφορετικές συνθήκες. Ακόμα χειρότερα όπως αναφέρθηκε προηγουμένως έχουν ακρίβεια ακεραίου, γεγονός που οφείλεται εν μέρη στην ακρίβεια των οργάνων παρατήρησης, και κατά δεύτερο στη μία και μοναδική μέτρηση και την έλλειψη δείγματος από το οποίο θα εξάγαμε μέση τιμή και τυπική απόκλιση.

Σύμφωνα με τα παραπάνω η μόνη ασφαλής προσέγγιση στη σύγκριση των τιμών ισχύος είναι να δείξουμε πόσο συγκλίνουν οι μετρήσεις παρατήρησης με τις στρογγυλοποιημένες σε ακέραιο μετρήσεις του προσομοιωτή. Υπολογίζουμε συνεπώς τις διαφορές των αντιστοίχων τιμών με σήμανση θετικής/αρνητικής απόκλισης (τιμή δείγματος – τιμή υπολογισμού). Για τις αποκλίσεις αυτές υπολογίζεται ο μέσος όρος (MEAN/AVERAGE) των απολύτων τιμών τους, και η μέση διασπορά τους από αυτόν (AVERAGE DEVIATION). Για κάθε περίπτωση εξασθένισης λόγω απόστασης, εντοπίζουμε την προσομοίωση με τη μεγαλύτερη σύγκλιση, καθώς και την καλύτερη συνολικά. Εξετάζονται όλοι οι συνδυασμοί με τις εξασθενίσεις βλάστησης. Ζητούμενο είναι χαμηλός μέσος όρος αποκλίσεων και μικρή διασπορά. Η μελέτη γίνεται ξεχωριστά για την περιοχή της Καισαριανής και της Πεντέλης.

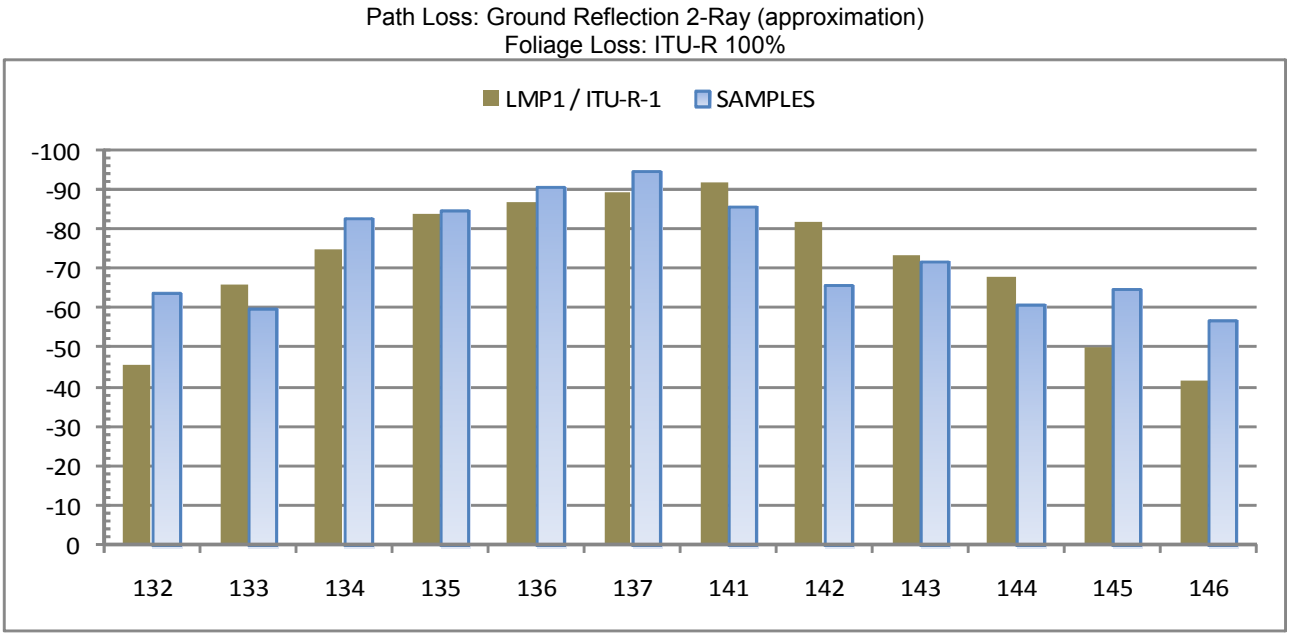
Η σύγκλιση τιμών οπτικοποιείται σε διαγράμματα στα οποία παρουσιάζονται σε μπάρες οι τιμές ισχύος σε dBm. Όσο μικρότερη είναι η τιμή της ισχύος τόσο μεγαλύτερη είναι η μπάρα της απεικόνισης, διότι είναι σε λογαριθμική μορφή. Κάθε διάγραμμα συνοδεύεται από τον πίνακα αποκλίσεων της προσομοίωσης από τα δείγματα:

8.3.1.1 Shawn: Καισαριανή

Για την περιοχή της Καισαριανής αναμένουμε κανονικά υψηλό εύρος βλάστησης για τις συγκλίνουσες προσομοιώσεις. Πέρα του κόμβου 131 που βρίσκεται πολύ κοντά στον 130, εξαιρούμε της σύγκρισης και τις τιμές των 138, 139, 140 για τους οποίους δεν έχουμε μέτρηση:

- Εξασθένιση Διαδρομής Ανακλάσεων Δύο Δρόμων (Ground Reflection 2-Ray)

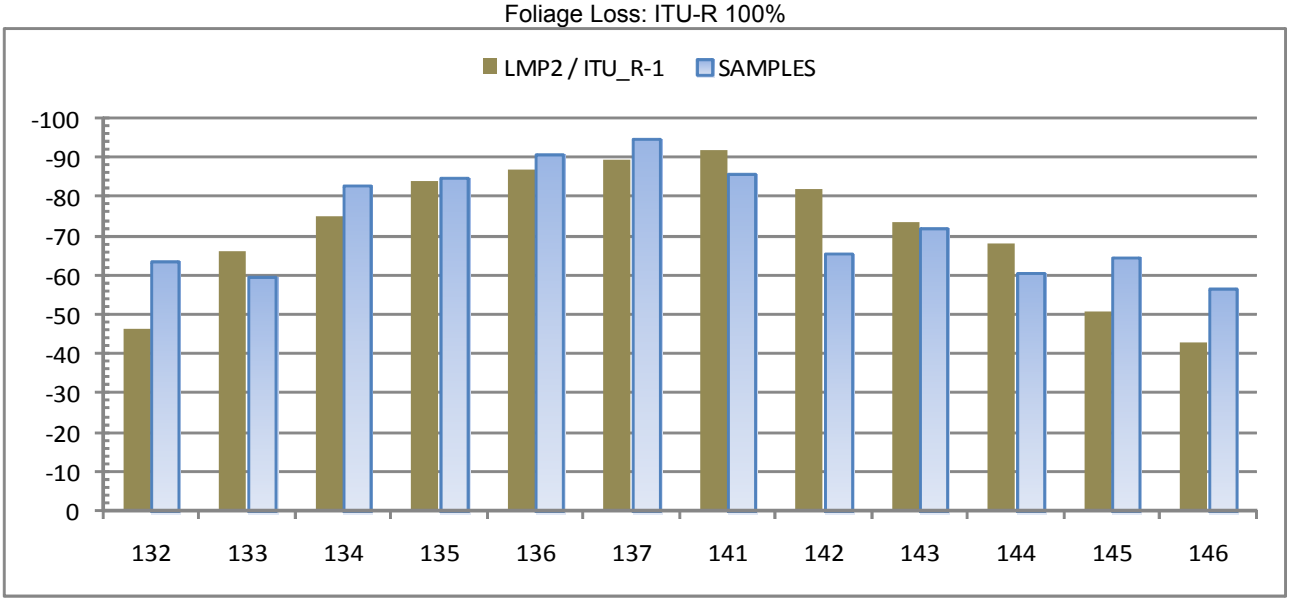
Στην πρώτη περίπτωση της Καισαριανής εξετάζουμε το μοντέλο εξασθένισης διαδρομής Ανακλάσεων Δύο Δρόμων. Αυτό εμφανίζει τη μεγαλύτερη σύγκλιση με το μοντέλο βλάστησης ITU-R στο 100% της διαδρομής, γεγονός που παρότι ακραίο προσεγγίζει την φυσιογνωμία της περιοχής. Η σύγκλιση παρουσιάζεται για το προσεγγιστικό και το αναλυτικό μοντέλο στα παρακάτω γραφήματα και πίνακες:



Γράφημα 8.1

nodeID:	132	133	134	135	136	137	141	142	143	144	145	146
sim RX (dBm):	-46	-67	-76	-84	-88	-90	-93	-82	-74	-69	-51	-42
sample RX (dBm):	-64	-60	-83	-85	-91	-95	-86	-66	-72	-61	-65	-57
difference (dBm):	-18	7	-7	-1	-3	-5	7	16	2	8	-14	-15
											MEAN of absolute difference (dBm):	8.58
											AVERAGE DEVIATION of absolute difference (dBm):	8.42

Πίνακας 8.6



Γράφημα 8.2

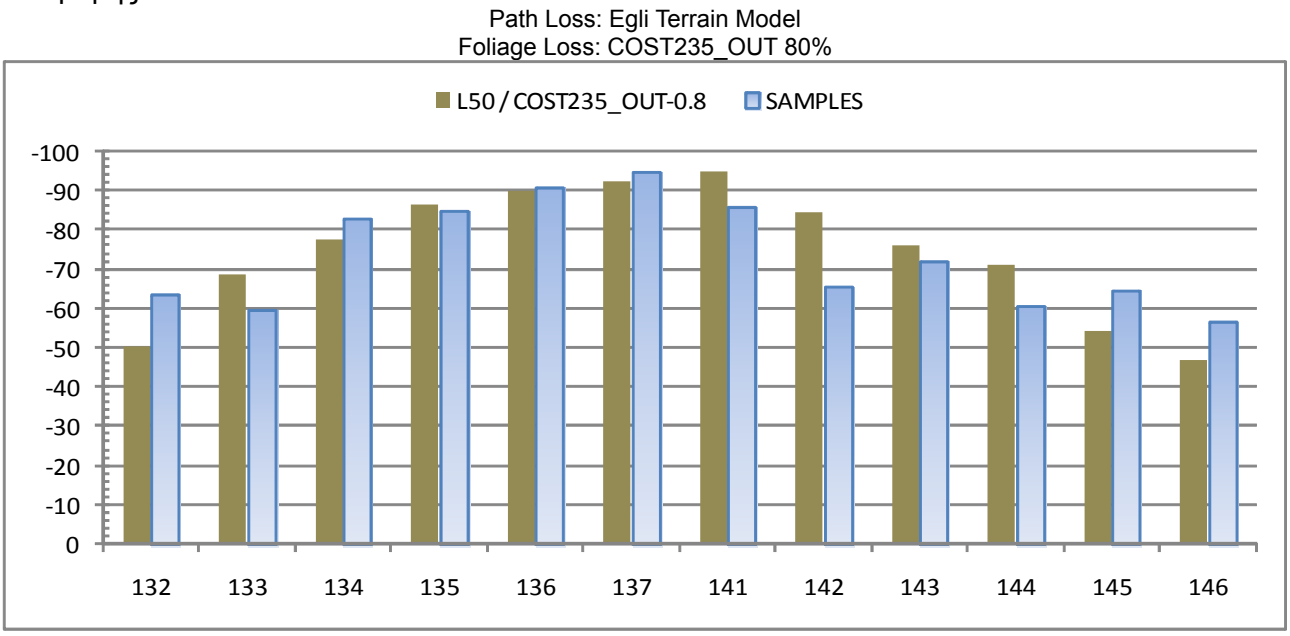
nodeID:	132	133	134	135	136	137	141	142	143	144	145	146
sim RX (dBm):	-47	-67	-76	-84	-88	-90	-93	-83	-74	-69	-51	-43
sample RX (dBm):	-64	-60	-83	-85	-91	-95	-86	-66	-72	-61	-65	-57
difference (dBm):	-17	7	-7	-1	-3	-5	7	17	2	8	-14	-14
											MEAN of absolute difference (dBm):	8.50
											AVERAGE DEVIATION of absolute difference (dBm):	8.33

Πίνακας 8.7

Οι κόμβοι που εμφανίζουν μεγάλες διαφορές από τις τιμές μέτρησης στην προσεγγιστική σχέση [Πίνακας 8.6], είναι οι πιο κοντινοί (132,145,146) και ο 142. Οι 145,146 δεν έχουν οπτική επαφή σύμφωνα με τις παρατηρήσεις, ενώ ο 142 δίνει πολύ καλή ισχύ στις μετρήσεις παρότι απέχει απόσταση μεγαλύτερη πχ του 143 που έχει μικρότερο σήμα. Αυτό μας δίνει μια ένδειξη ότι ο κόμβος 142 έχει κάποιον επιπλέον παράγοντα εξασθένισης που δεν καταγράφεται στις παρατηρήσεις. Το μοντέλο εμφανίζει μεγαλύτερη σύγκλιση στις μεγάλες αποστάσεις κόμβων. Στην αναλυτική σχέση βλέπουμε λίγο καλύτερη σύγκλιση στους κοντινούς κόμβους 132 και 146, αλλά λόγω της στρογγυλοποίησης που πραγματοποιήσαμε δε θεωρούμε διαφορές.

- Εξασθένιση Αναγλύφου Egli

Το μοντέλο αναγλύφου του Egli δίνει καλύτερη σύγκλιση από το μοντέλο Ανακλάσεων Δύο Δρόμων, συνδυαζόμενο όμως με τελείως διαφορετικής φύσης μοντέλο εξασθένισης βλάστησης. Ενώ στην προηγούμενη περίπτωση είχαμε σύγκλιση για μοντέλο ITU-R και 100% κάλυψης της έκτασης, το μοντέλο του Egli συγκλίνει με το COST235 για βλάστηση χωρίς φύλλωμα στο 80% της διαδρομής:



Γράφημα 8.3

nodeID:	132	133	134	135	136	137	141	142	143	144	145	146
sim RX (dBm):	-51	-69	-78	-87	-90	-93	-96	-85	-77	-72	-55	-48
sample RX (dBm):	-64	-60	-83	-85	-91	-95	-86	-66	-72	-61	-65	-57
difference (dBm):	-13	9	-5	2	-1	-2	10	19	5	11	-10	-9

MEAN of absolute difference (dBm):	8.00
AVERAGE DEVIATION of absolute difference (dBm):	8.00

Πίνακας 8.8

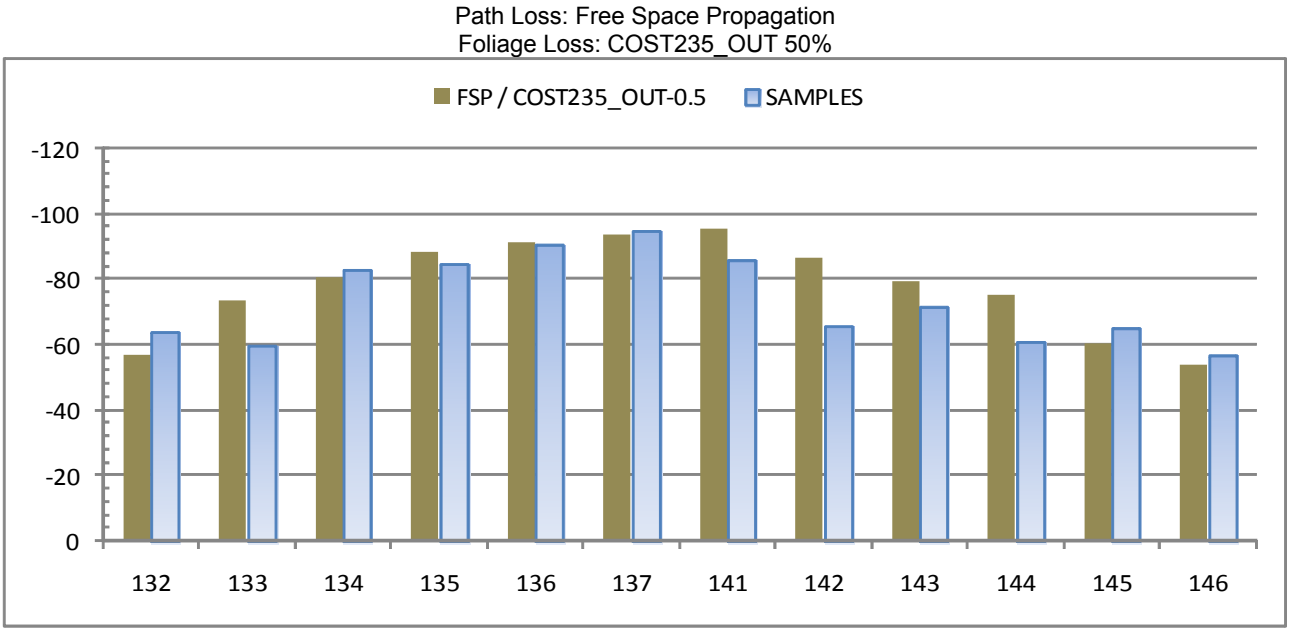
Και σε αυτή τη περίπτωση έχουμε μεγάλες αποκλίσεις στις κοντινές αποστάσεις, όμως πολύ μικρότερες σε σχέση με το μοντέλο ανακλάσεων δύο δρόμων. Το μοντέλο του Egli δίνει την ενδιάμεση τιμή εξασθένισης, γεγονός που σημαίνει ότι η σύγκλιση αυτή ισχύει στο 50% των θέσεων στην περιοχή. Ο κόμβος 142 και εδώ εμφανίζει μεγάλη απόκλιση, όπως και στα επόμενα μοντέλα που ακολουθούν, συνεισφέροντας σε μεγάλο βαθμό στη μεγάλη τιμή του μέσου όρου.

Αυτό που κάνει αίσθηση είναι ο συνδυασμός με το μοντέλο βλάστησης COST235 για βλάστηση χωρίς φύλλωμα. Η περιοχή που εξετάζουμε είναι δασική έκταση με μεγάλη ποικιλία δέντρων, όπως πεύκα (διάφορα είδη), κουκουναριές, κυπαρίσσια, ευκάλυπτοι, βελανιδιές κ.α. Προκειμένου να ταιριάζει βλάστηση χωρίς φύλλωμα στην περιοχή, θα πρέπει είτε να έχουμε πολύ αραιό φύλλωμα είτε τα δέντρα να είναι φυλλοβόλα και να βρισκόμαστε σε εποχή που αυτά έχουν απολέσει το φύλλωμά τους. Ωστόσο δεν ισχύει ούτε η πρώτη συνθήκη για όλα τα δέντρα ούτε και η δεύτερη, διότι πρόκειται για μετρήσεις που έγιναν τέλη Μαρτίου, εποχή που η βλάστηση έχει από καιρό αρχίσει να επανέρχεται από τη φθορά του χειμώνα.

Μια πιο λογική εξήγηση είναι οι φυλλωσιές των δέντρων να βρίσκονται σε μεγαλύτερο ύψος από το έδαφος σε σχέση με τους παρατηρητές, οπότε και για αυτή τη κατάσταση έχουμε συνθήκες χωρίς φύλλωμα. Στα επόμενα μοντέλα ενισχύεται αυτή η άποψη, χωρίς να αποτελεί όμως σε καμία περίπτωση τεκμήριο.

- Εξασθένιση Διαδρομής Ελευθέρου Χώρου (FSP)

Ο τύπος αυτός εξασθένισης δίνει πολύ καλή σύγκλιση στους κοντινούς κόμβους, και την καλύτερη συνολικά από όλα τα μοντέλα. Αυτή προκύπτει σε συνδυασμό με μοντέλο εξασθένισης βλάστησης COST235 για βλάστηση χωρίς φύλλωμα, σε εύρος 50% της διαδρομής:



Γράφημα 8.4

nodeID:	132	133	134	135	136	137	141	142	143	144	145	146
sim RX (dBm):	-58	-74	-82	-89	-92	-94	-96	-87	-80	-76	-61	-55
sample RX (dBm):	-64	-60	-83	-85	-91	-95	-86	-66	-72	-61	-65	-57
difference (dBm):	-6	14	-1	4	1	-1	10	21	8	15	-4	-2

MEAN of absolute difference (dBm):	7.25
AVERAGE DEVIATION of absolute difference (dBm):	7.24

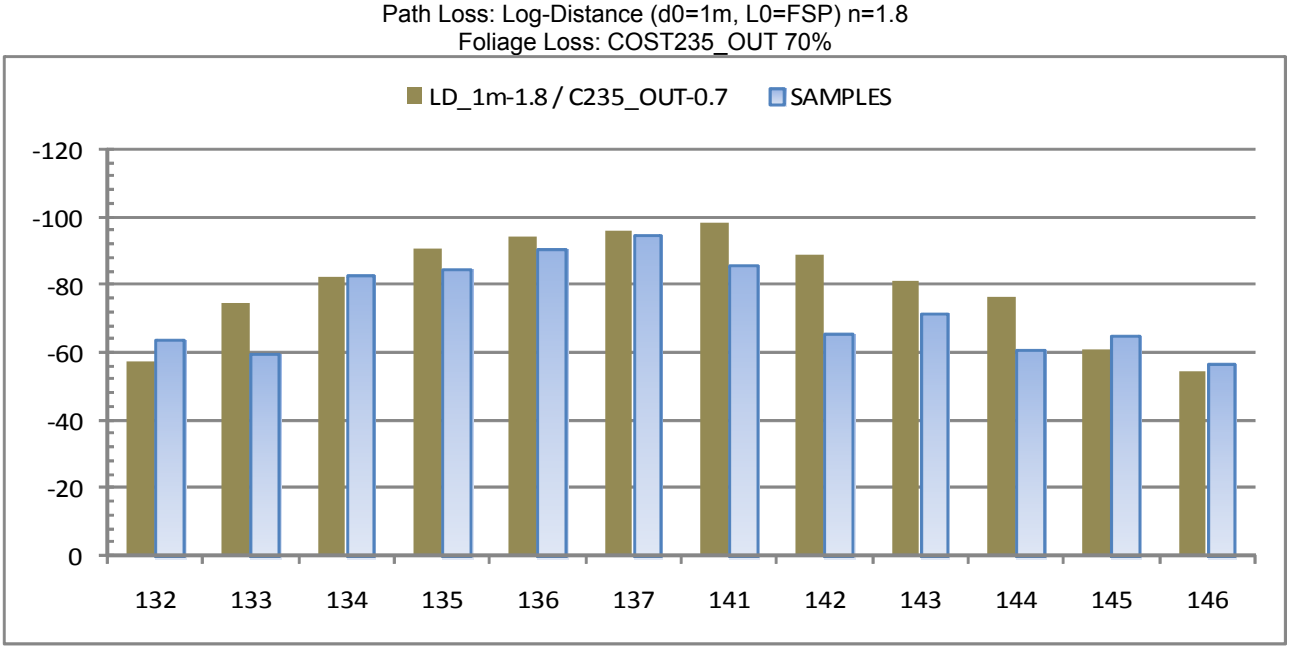
Πίνακας 8.9

Η προσομοίωση αυτή όπως φαίνεται και από τον Πίνακα 8.9 συγκλίνει πάρα πολύ σε κάποιους κόμβους που τα προηγούμενα μοντέλα απέκλιναν, όπως οι 132,134,145 και 146. Σε αντάλλαγμα όμως δίνει ακόμα μεγαλύτερη απόκλιση στον κόμβο 142, και επιπλέον στους 133 και 144. Οι συγκεκριμένοι κόμβοι είναι πολύ κοντά στον δέκτη και έχουν οπτική επαφή και πολύ υψηλή τιμή σήματος σύμφωνα με τις μετρήσεις. Σε αυτές τις τιμές προφανώς το μοντέλο βλάστησης είναι ο λόγος της μεγάλης απόκλισης. Μια μικρότερη τιμή εύρους θα εξαφάνιζε την απόκλιση στους κόμβους αυτούς, αλλά θα προσέθετε στους υπολοίπους που συγκλίνουν.

Γίνεται κατανοητό με αυτό τον τρόπο ότι με διαφορετική εκτίμηση της εξασθένισης βλάστησης σε κάθε σύνδεση θα μπορούσαμε να έχουμε την ιδανική σύγκλιση. Ο σκοπός όμως είναι να μην εξαρτόμαστε τελείως από κάθε παρατήρηση, αλλά με συνολική αποτίμηση του μοντέλου να προβλέπουμε όσο καλύτερα γίνεται την συμπεριφορά μιας περιοχής.

- Εξασθένισης Διαδρομής Λογαριθμικής Απόστασης (Log-Distance)

Το μοντέλο αυτό δίνει την καλύτερη σύγκλιση με εκθέτη 2, δηλαδή όταν μεταπίπτει σε μοντέλο Εξασθένισης Ελευθέρου Χώρου. Πλην αυτής της περίπτωσης, η αμέσως επόμενη που εμφανίζει την 2η καλύτερη σύγκλιση τιμών είναι για εκθέτη 1.8, σε συνδυασμό με COST235 μοντέλο εξασθένισης βλάστησης χωρίς φύλλωμα και εύρους 70% της διαδρομής:



Γράφημα 8.5

nodeID:	132	133	134	135	136	137	141	142	143	144	145	146
sim RX (dBm):	-58	-75	-83	-92	-95	-97	-99	-90	-82	-77	-62	-55
sample RX (dBm):	-64	-60	-83	-85	-91	-95	-86	-66	-72	-61	-65	-57
difference (dBm):	-6	15	0	7	4	2	13	24	10	16	-3	-2

MEAN of absolute difference (dBm):	8.50
AVERAGE DEVIATION of absolute difference (dBm):	7.50

Πίνακας 8.10

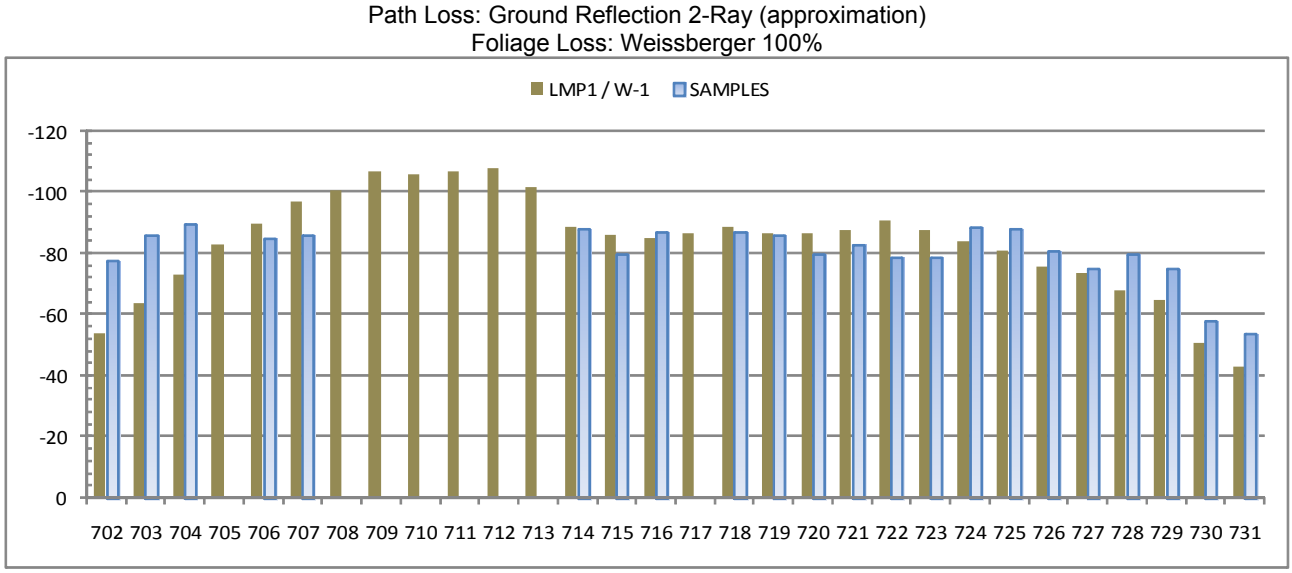
Ο εκθέτης 1.8, παραπέμπει σε μετάδοση μέσα σε κτίρια με οπτική επαφή. Η σύγκλιση είναι πολύ κοντά σε αυτή του μοντέλου ελευθέρου χώρου, χειρότερη ιδίως για τους κόμβους 142 και 144, όπου όπως αναφέρθηκε και προηγουμένως έχουμε πολύ υψηλές τιμές ισχύος σε σχέση με την απόσταση.

8.3.1.2 Shawn: Πεντέλη

Για τη περιοχή αυτή έχουμε πολλούς κόμβους στους οποίους δεν ελήφθη καθόλου σήμα στη σειρά των μετρήσεων. Λόγω της φύσεως της βλάστησης της περιοχής (αραιή/χαμηλή) εξετάζουμε το εύρος της για τιμές από 10% έως 100%. Εξετάζουμε με τον ίδιο τρόπο τα εξαγόμενα της εργασίας rf_attenuation_report και παρουσιάζουμε τις συγκλίνουσες τιμές σε πίνακες. Προκειμένου να δούμε οπτικά το θέμα των κόμβων χωρίς σήμα λήψης, ενσωματώνουμε στο γράφημα τις τιμές τους, χωρίς τη συνοδεία μέτρησης παρατήρησης. Οι τιμές αυτές δεν συμμετέχουν στην στατιστική ανάλυση, αλλά παρατίθενται προς αξιολόγηση της κατάστασης και της περιοχής. Ακολουθούν οι βέλτιστες τιμές προσέγγισης σε γραφήματα όπως και στη προηγούμενη ενότητα:

- Εξασθένιση Διαδρομής Ανακλάσεων Δύο Δρόμων (Ground Reflection 2-Ray)

Το μοντέλο αυτό εμφανίζει τη μικρότερη σύγκλιση από όλα τα υπόλοιπα μοντέλα εξασθένησης διαδρομής. Την καλύτερη προσαρμογή στο δείγμα την παρουσιάζει για μοντέλο βλάστησης Weissberger στο 100% της διαδρομής.

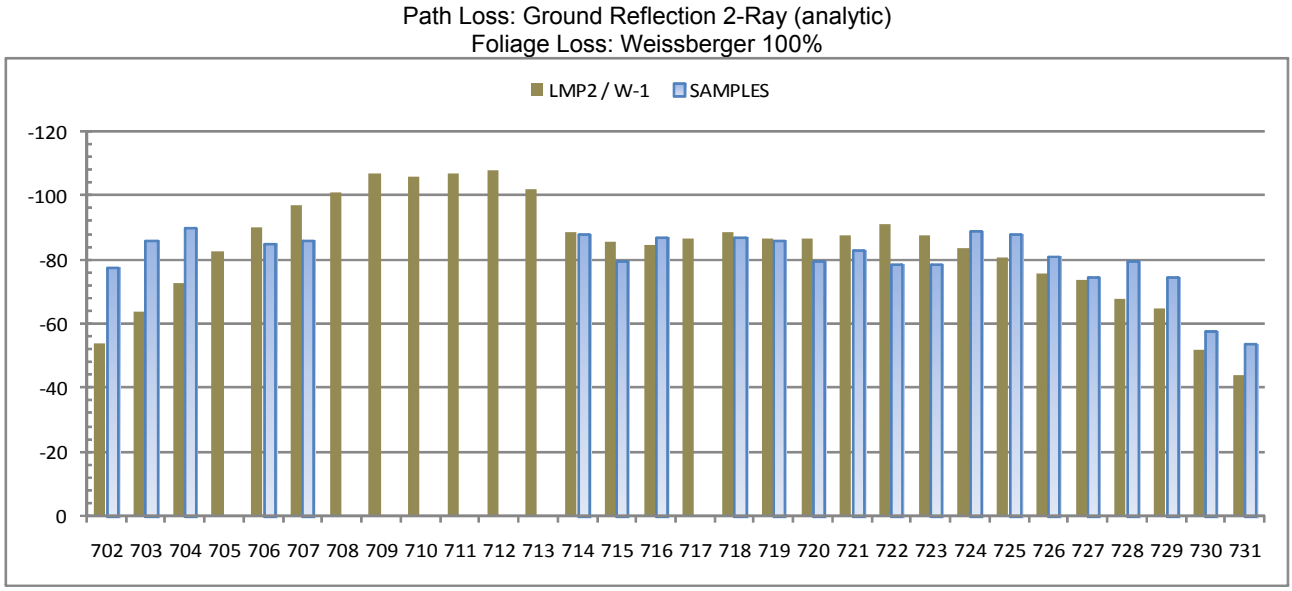


Γράφημα 8.6

nodeID:	702	703	704	706	707	714	715	716	718	719	720	721	722	723	724	725	726	727	728	729	730	731
sim RX (dBm):	-54	-64	-73	-90	-97	-89	-86	-85	-89	-87	-87	-88	-91	-88	-84	-81	-76	-74	-68	-65	-51	-43
sample RX (dBm):	-78	-86	-90	-85	-86	-88	-80	-87	-87	-86	-80	-83	-79	-79	-89	-88	-81	-75	-80	-75	-58	-54
difference (dBm):	-24	-22	-17	5	11	1	6	-2	2	1	7	5	12	9	-5	-7	-5	-1	-12	-10	-7	-11

MEAN of absolute difference (dBm):	8.3
AVERAGE DEVIATION of absolute difference (dBm):	4.9

Πίνακας 8.11



Γράφημα 8.7

nodeID:	702	703	704	706	707	714	715	716	718	719	720	721	722	723	724	725	726	727	728	729	730	731
sim RX (dBm):	-54	-64	-73	-90	-97	-89	-86	-85	-89	-87	-87	-88	-91	-88	-84	-81	-76	-74	-68	-65	-52	-44
sample RX (dBm):	-78	-86	-90	-85	-86	-88	-80	-87	-87	-86	-80	-83	-79	-79	-89	-88	-81	-75	-80	-75	-58	-54
difference (dBm):	-24	-22	-17	5	11	1	6	-2	2	1	7	5	12	9	-5	-7	-5	-1	-12	-10	-6	-10

MEAN of absolute difference (dBm):	8.2
AVERAGE DEVIATION of absolute difference (dBm):	4.9

Πίνακας 8.12

Εξίσου καλή σύγκλιση κάνει και με το μοντέλο ITU-R επίσης στο 100% της διαδρομής. Η συμπεριφορά αυτή είναι ίδια με την αντίστοιχη περίπτωση που εξετάστηκε για την περιοχή της Καισαριανής. Εδώ όμως έχουμε την πληροφορία για χαμηλή και αραιή βλάστηση, γεγονός που δε δικαιολογεί το ποσοστό αυτό.

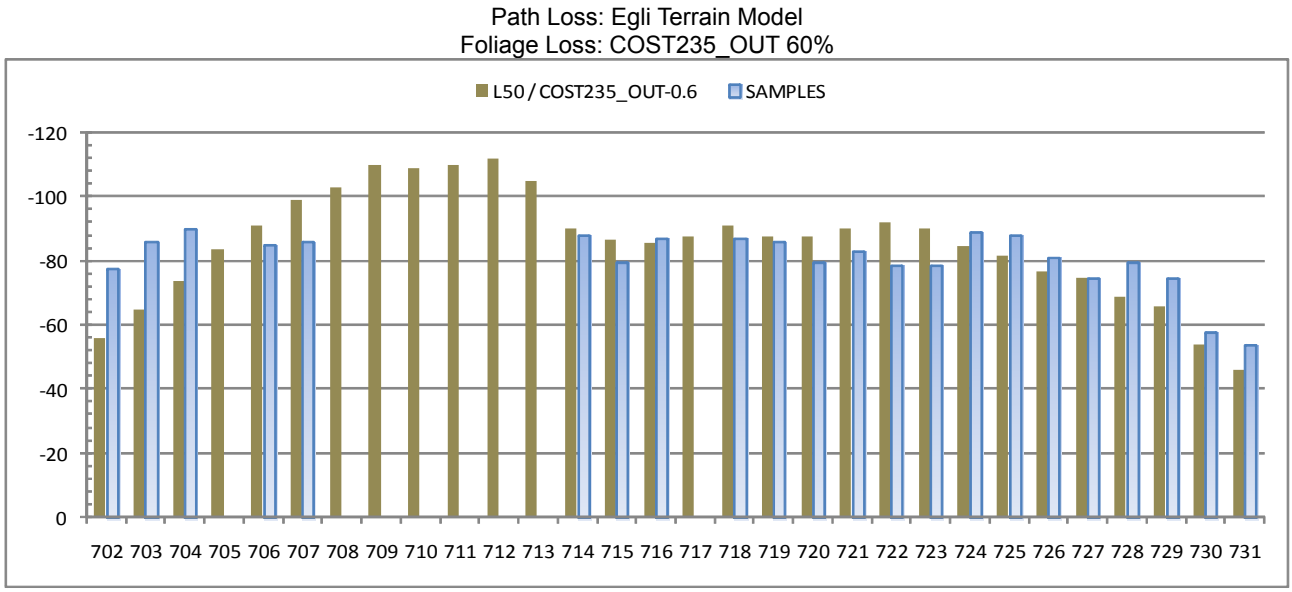
Αυτό που κάνει ιδιαίτερη αίσθηση στα Γραφήματα 8.7 και 8.8 είναι οι τιμές ισχύος για τους κόμβους 708-713, όπως υπολογίζονται από τον προσομοιωτή. Αυτές είναι πάρα πολύ μικρές, ειδικά αν τις συγκρίνουμε με τις τιμές ισχύος για τους υπόλοιπους κόμβους. Άρα βάσει μοντέλου και προσομοίωσης αναμένουμε ελάχιστο έως και μηδενικό σήμα από την μέτρηση, όπως και συμβαίνει. Αυτό όμως που απορρίπτεται είναι η δικαιολογία της έλλειψης σήματος που σημαίνεται στις παρατηρήσεις των μετρήσεων.

Σύμφωνα με αυτή οι κόμβοι αυτοί δεν έχουν σήμα λόγω των γραμμών υψηλής τάσης που υπάρχουν στην περιοχή. Ο προσομοιωτής χωρίς να λαμβάνει κάτι τέτοιο υπόψη ήδη αποδίδει πολύ χαμηλές τιμές στη λαμβανόμενη ισχύ από τους κόμβους αυτούς. Ο λόγος είναι καθαρά η απόσταση. Πρόκειται για τους πιο απομακρυσμένους κόμβους από το δέκτη, με αποστάσεις από 236 έως 293 μέτρα, οι οποίες και αναγράφονται στις παρατηρήσεις. Το αν συνεισφέρουν οι γραμμές υψηλής τάσεις στην εξασθένιση είναι ένα ερώτημα που αφορά όλους τους κόμβους και όχι μόνο τους απομακρυσμένους. Για τους τελευταίους μια σίγουρη δικαιολογία έλλειψης σήματος στις μετρήσεις είναι η απόσταση.

Αναφορικά με τους υπόλοιπους κόμβους του δικτύου, ενδιαφέρον παρουσιάζουν οι 702, 703 και 705. Αυτοί παρότι από τους πιο κοντινούς παρουσιάζουν πολύ χαμηλά επίπεδα ισχύος λήψης. Οι κόμβοι δεν έχουν κάποια παρατήρηση σχετική με τις γραμμές υψηλής τάσης.

- Εξασθένιση Αναγλύφου Egli

Με το μοντέλο του Egli έχουμε αρκετά καλή σύγκλιση, ακόμα και σε κοντινές αποστάσεις, με εξαίρεση φυσικά τους κόμβους 702, 703 και 704, που όπως είδαμε παρουσιάζουν εξαιρετικά χαμηλή τιμή ισχύος λήψης στις μετρήσεις. Η προσομοίωση αξιοποιεί επιπλέον μοντέλο βλάστησης COST235 χωρίς φύλλωμα για το 60% της διαδρομής:



Γράφημα 8.8

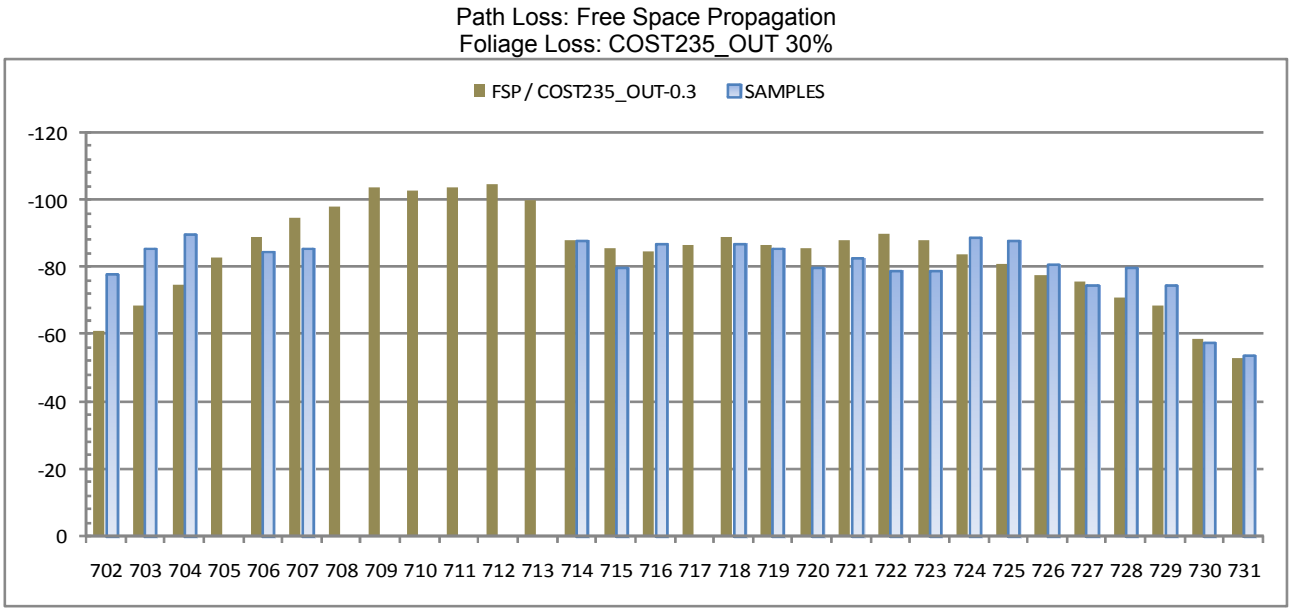
nodeID:	702	703	704	706	707	714	715	716	718	719	720	721	722	723	724	725	726	727	728	729	730	731
sim RX (dBm):	-56	-65	-74	-91	-99	-90	-87	-86	-91	-88	-88	-90	-92	-90	-85	-82	-77	-75	-69	-66	-54	-46
sample RX (dBm):	-78	-86	-90	-85	-86	-88	-80	-87	-87	-86	-80	-83	-79	-79	-89	-88	-81	-75	-80	-75	-58	-54
difference (dBm):	22	21	16	6	13	2	7	1	4	2	8	7	13	11	4	6	4	0	11	9	4	8
																					MEAN of absolute difference (dBm):	8.1
																					AVERAGE DEVIATION of absolute difference (dBm):	4.6

Πίνακας 8.13

Όπως και στην αντίστοιχη περίπτωση της Καισαριανής, έτσι και εδώ το μοντέλο COST235 χωρίς φύλλωμα ταιριάζει με την περιοχή, αυτή τη φορά στο 60% της διαδρομής, δηλαδή για μεγάλο ποσοστό συναρτήσει της φύσεως της περιοχής. Και εδώ διακρίνεται πολύ έντονα στο Γράφημα 8.9 η ζώνη χαμηλών τιμών ισχύος (708-713). Γίνεται αισθητή και η απόκλιση για τους κόμβους 722 και 723 οι οποίοι βρίσκονται μέσα σε όρυγμα, αλλά οι μετρήσεις τους δίνουν καλή ισχύ σε σχέση με την προσομοίωση.

- Εξασθένιση Διαδρομής Ελευθέρου Χώρου (FSP)

Το μοντέλο αυτό δίνει πολύ καλή σύγκλιση, σε συνδυασμό με COST235 μοντέλο βλάστησης χωρίς φύλλωμα για 30% της διαδρομής:



Γράφημα 8.9

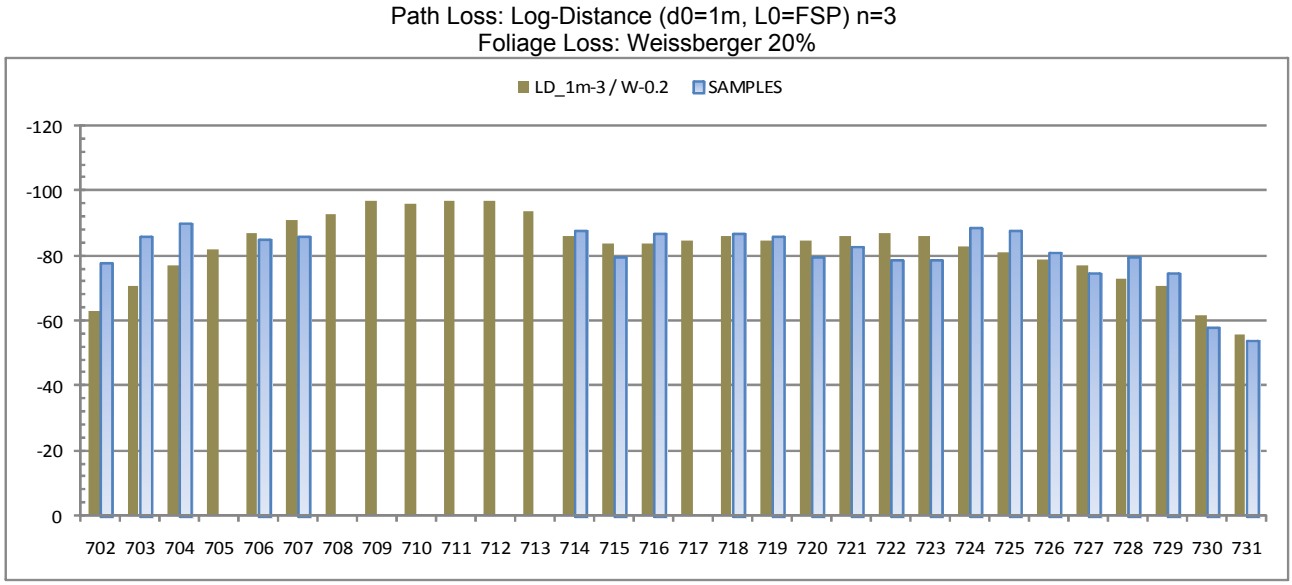
nodeID:	702	703	704	706	707	714	715	716	718	719	720	721	722	723	724	725	726	727	728	729	730	731
sim RX (dBm):	-61	-69	-75	-89	-95	-88	-86	-85	-89	-87	-86	-88	-90	-88	-84	-81	-78	-76	-71	-69	-59	-53
sample RX (dBm):	-78	-86	-90	-85	-86	-88	-80	-87	-87	-86	-80	-83	-79	-79	-89	-88	-81	-75	-80	-75	-58	-54
difference (dBm):	-17	-17	-15	4	9	0	6	-2	2	1	6	5	11	9	-5	-7	-3	1	-9	-6	1	-1
																					MEAN of absolute difference (dBm):	6.2
																					AVERAGE DEVIATION of absolute difference (dBm):	4.0

Πίνακας 8.14

Είναι χαρακτηριστική η βελτίωση στην σύγκλιση για τις κοντινές αποστάσεις, ακόμα και στους προβληματικούς κόμβους 702, 703, και 704. Το ποσοστό 30% για το εύρος βλάστησης είναι αρκετά πιο ρεαλιστικό από τα προηγούμενα για την περιοχή.

- Εξασθένισης Διαδρομής Λογαριθμικής Απόστασης (Log-Distance)

Το μοντέλο Log-Distance, δίνει την καλύτερη σύγκλιση τιμών σε αυτή την περιοχή.



Γράφημα 8.10

nodeID:	702	703	704	706	707	714	715	716	718	719	720	721	722	723	724	725	726	727	728	729	730	731
sim RX (dBm):	-63	-71	-77	-87	-91	-86	-84	-84	-86	-85	-85	-86	-87	-86	-83	-81	-79	-77	-73	-71	-62	-56
sample RX (dBm):	-78	-86	-90	-85	-86	-88	-80	-87	-87	-86	-80	-83	-79	-79	-89	-88	-81	-75	-80	-75	-58	-54
difference (dBm):	-15	-15	-13	2	5	-2	4	-3	-1	-1	5	3	8	7	-6	-7	-2	2	-7	-4	4	2
																					MEAN of absolute difference (dBm):	5.4
																					AVERAGE DEVIATION of absolute difference (dBm):	3.2

Πίνακας 8.15

Πέρα των κόμβων 702-704, σε όλες τις άλλες περιπτώσεις οι διαφορές κυμαίνονται σε αρκετά χαμηλά επίπεδα. Και σε αυτή την προσομοίωση είναι χαρακτηριστικές οι μικρές τιμές ισχύος που προβλέπονται για τους κόμβους 708-713.

8.3.2 Castalia

Το Castalia υπολογίζει μόνο εξασθένιση απόστασης, με χρήση του μοντέλου Log-Distance. Δεν τροποποιήθηκε αυτή η πολιτική, προκειμένου να συγκρίνουμε τα αποτελέσματα με τα αντίστοιχα του σύνθετου υπολογισμού με μοντέλα βλάστησης, όπως παρουσιάστηκε στο Shawn.

Το Castalia εκκινεί την προσομοίωση με απλή κλήση του εκτελέσιμου στην γραμμή εντολών, χωρίς παραμέτρους. Τα αποτελέσματα της προσομοίωσης εξάγονται σε δύο αρχεία κειμένου:

Castalia-Debug.txt
Castalia-Primay-Output.txt

Το πρώτο καταγράφει μηνύματα αποσφαλμάτωσης, σύμφωνα με οδηγίες που είναι ενσωματωμένες σε διάφορα σημεία του κώδικα. Οι περισσότερες αναφορές που καταγράφονται σε αυτό το αρχείο προέρχονται από το Wireless_Channel. Για κάθε μονάδα είναι δυνατό να ενεργοποιούμε ή να απενεργοποιούμε την εγγραφή σε αυτό το αρχείο με τη παράμετρο 'bool printDebugInfo'.

Στο Castalia-Primary-Output.txt καταγράφονται όλες οι ρυθμίσεις της προσομοίωσης και τα αποτελέσματα της *finish()* μεθόδου των κόμβων. Σε αυτή συνήθως δίνονται εντολές καταγραφής στατιστικών παραλαβής μηνυμάτων. Στην υλοποίηση των μονάδων gateway εκτυπώνεται κάθε παραλήπτης, ο αριθμός μηνυμάτων που λάβαμε από αυτόν σε σχέση με πόσα έστειλε συνολικά, και η μέση ισχύς λήψης, όπως ενσωματώνεται στα μηνύματα από το μοντέλο επικοινωνίας. Μια τυπική μορφή αυτών των στοιχείων φαίνεται στο παρακάτω απόσπασμα από το Castalia-Primary-Output.txt για τα δεδομένα της Καισαριανής:

```
Calling finish() at end of Run #1...

** Node [130] received from:
[130<--131]  -->   100   out of     100    mean rssi: -20.6373
[130<--132]  -->   100   out of     100    mean rssi: -68.4528
[130<--133]  -->   100   out of     100    mean rssi: -79.3818
[130<--134]  -->   100   out of     100    mean rssi: -83.6589
[130<--135]  -->   100   out of     100    mean rssi: -87.538
[130<--136]  -->   100   out of     100    mean rssi: -88.9224
[130<--137]  -->   100   out of     100    mean rssi: -89.9037
[130<--138]  -->   100   out of     100    mean rssi: -93.2178
[130<--139]  -->   100   out of     100    mean rssi: -93.3531
[130<--140]  -->   100   out of     100    mean rssi: -94.9918
[130<--141]  -->   100   out of     100    mean rssi: -90.8975
[130<--142]  -->   100   out of     100    mean rssi: -86.6899
[130<--143]  -->   100   out of     100    mean rssi: -82.9295
[130<--144]  -->   100   out of     100    mean rssi: -80.4458
[130<--145]  -->   100   out of     100    mean rssi: -71.0603
[130<--146]  -->   100   out of     100    mean rssi: -66.2088
```

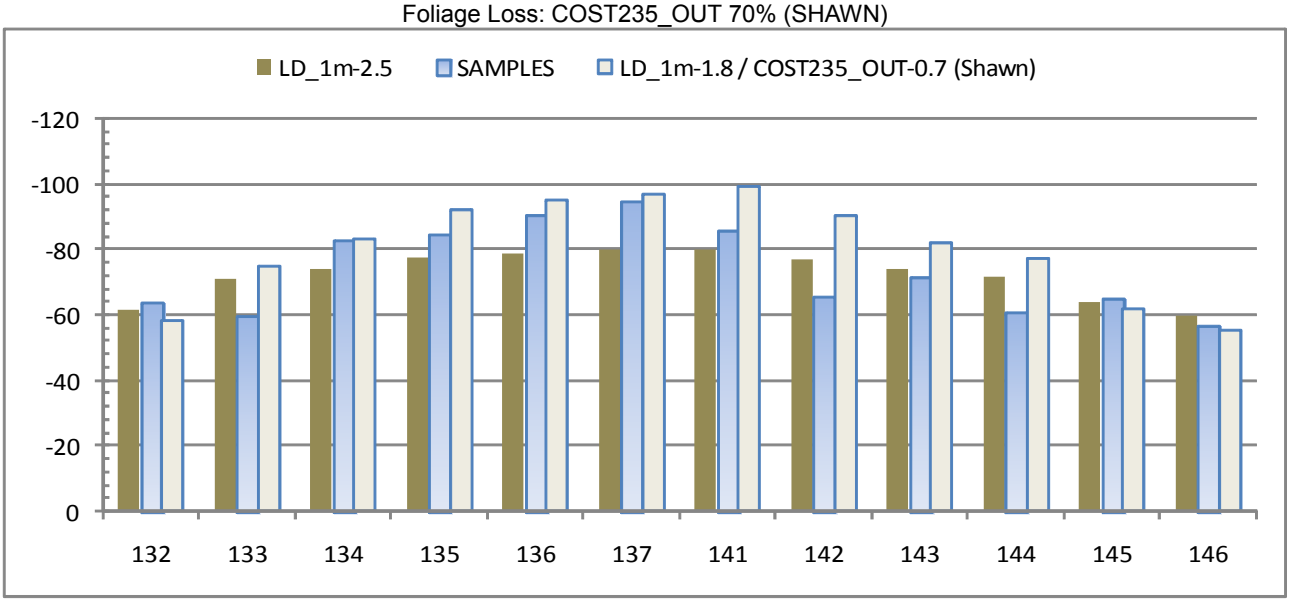
[Castalia-Primary-Output.txt]

Φαίνεται χαρακτηριστικά η απόλυτη επιτυχία στη λήψη των πακέτων. Αυτό οφείλεται στην μη χρήση του μοντέλου παρεμβολής. Επιπλέον παρότι η ονομαστική ευαισθησία του δέκτη είναι στα -122 dBm, είναι μάλλον αρκετά μεγαλύτερη, δεδομένου ότι δεν έχουμε στις μετρήσεις ανιχνεύσιμη ισχύ σήματος μικρότερη από -95 dBm.

Εξετάζουμε με το Castalia την περίπτωση της Καισαριανής. Για την Πεντέλη η διαδικασία είναι παρόμοια.

Καλύτερη σύγκλιση στα δεδομένα της Καισαριανής το Castalia πετυχαίνει για τιμή 2.5 στον εκθέτη της λογαριθμικής εξασθένισης. Ο εκθέτης 2.5 είναι προτεινόμενος για χρήση σε εργοστάσια με εμπόδια, σύμφωνα με τον Πίνακα 4.21[§4.1.2.2]. Αυτά τα δεδομένα παρουσιάζονται σε γράφημα μαζί με τις τιμές που δίνει το μοντέλο επικοινωνίας του Shawn για το ίδιο μοντέλο εξασθένισης, με εκθέτη 2.1, και σε συνδυασμό με μοντέλο εξασθένισης βλάστησης Weissberger για εύρος 50% της διαδρομής. Το μοντέλο αυτό είδαμε να παρουσιάζει την καλύτερη σύγκλιση, με εξαίρεση αυτό με εκθέτη 2 που πρόκειται για το FSP μοντέλο. Επίσης παρουσιάζονται σε πίνακα οι διαφορές των μετρήσεων με τις τιμές υπολογισμού κάθε προσομοιωτή, όπως και οι διαφορά μεταξύ των τιμών υπολογισμού των ιδίων:

Path Loss: Log-Distance (d0=1m, L0=FSP) n=2.5 (CASTALIA)
vs
Path Loss: Log-Distance (d0=1m, L0=FSP) n=1.8 (SHAWN)
Foliage Loss: COST235_OUT 70% (SHAWN)



Γράφημα 8.11

nodeID:	132	133	134	135	136	137	141	142	143	144	145	146
Castalia RX (dBm):	-62	-71	-74	-78	-79	-80	-80	-77	-74	-72	-64	-60
sample RX (dBm):	-64	-60	-83	-85	-91	-95	-86	-66	-72	-61	-65	-57
Shawn (dBm):	-58	-75	-83	-92	-95	-97	-99	-90	-82	-77	-62	-55
sample - Castalia difference (dBm):	-2	11	-9	-7	-12	-15	-6	11	2	11	-1	3
sample - Shawn difference (dBm):	-6	15	0	7	4	2	13	24	10	16	-3	-2
Shawn - Castalia difference (dBm):	4	-4	-9	-14	-16	-17	-19	-13	-8	-5	2	5

Castalia : MEAN of absolute difference (dBm):	7.50
Castalia : AVERAGE DEVIATION of absolute difference (dBm):	4.00
Shawn : MEAN of absolute difference (dBm):	8.50
Shawn : AVERAGE DEVIATION of absolute difference (dBm):	5.92
Shawn - Castalia : MEAN of absolute difference (dBm):	10.1
Shawn - Castalia : AVERAGE DEVIATION of absolute difference (dBm):	5.19

Πίνακας 8.16

Στη σύγκριση των δύο αποτελεσμάτων, βλέπουμε πως η σύγκλιση είναι καλύτερη στο Castalia. Η ψιλή τιμή του εκθέτη της λογαριθμικής εξασθένισης αναπληρώνει την απώλεια που εισάγει το μοντέλο βλάστησης. Ουσιαστικά οι τιμές του γραφήματος στο Castalia είναι πιο ομαλοποιημένες σε σχέση με τις αντίστοιχες του Shawn. Σε αυτό παρατηρούμε πολύ μεγαλύτερες τιμές ισχύος για τις μακρινές αποστάσεις. Αυτό οφείλεται καθαρά στο μοντέλο εξασθένισης βλάστησης, το οποίο εισάγει για δεύτερη φορά την απόσταση ως παράγοντα της εξασθένισης. Σε αυτές τις θέσεις το μοντέλο Log-Distance μόνο, όπως εφαρμόζεται στο Castalia, δεν αποδίδει. Φυσικά αυτό οφείλεται στην προσαρμογή στα πιο κοντινά σημεία, χάρη στα οποία και πετυχαίνει μεγαλύτερη σύγκλιση από το συνδυασμένο μοντέλο της προσομοίωσης του Shawn.

Παρά την υπεροχή από πλευράς αριθμών της προσομοίωσης του Castalia, ο συνδυασμός μοντέλου απόσταση και βλάστησης στο Shawn, όπως και στο Omnet++, δίνει πιο ρεαλιστικά αποτελέσματα, αν λάβει κανείς υπόψη και τις παρατηρήσεις που έχουμε για τις μετρήσεις. Το Log-Distance μοντέλο χωρίς βλάστηση, δίνει σε γενικές γραμμές τιμές μοιρασμένες πάνω και κάτω από τις αντίστοιχες των μετρήσεων. Αντίθετα ο συνδυασμός με μοντέλο βλάστησης δίνει μεγαλύτερη ισχύ από τις μετρήσεις για μόλις τρεις κόμβους.

8.3.3 Omnet++

Στο Omnet++ υλοποιούνται οι ίδιοι ακριβώς αλγόριθμοι με αυτούς που συγκροτούν το μοντέλο επικοινωνίας του Shawn. Τα αποτελέσματα που εξάγουν είναι απολύτως τα ίδια, όπως και η σύγκλιση με τις πειραματικές τιμές. Για το λόγο αυτό δεν παρουσιάζονται συγκρίσεις τιμών σε αυτό το εδάφιο. Ο αναγνώστης μπορεί να συμβουλευθεί την προηγούμενη ανάλυση του Shawn [§8.3.1] σαν να επρόκειτο για αποτελέσματα του Omnet++.

Ο τρόπος εξαγωγής τους όμως διαφέρει στο Omnet++ σε σχέση με το Shawn. Ενώ για το Shawn δημιουργήθηκε εργασία rf_attenuation_report, η οποία εξάγει μαζικά τιμές εξασθένισης από όλα τα μοντέλα, στο Omnet++ κάτι τέτοιο δεν ήταν δυνατό. Αυτό οφείλεται στον τρόπο που κατασκευάστηκε ο προσομοιωτής στα πλαίσια της αρχιτεκτονικής του Omnet++. Εφόσον κάθε μοντέλο υλοποιήθηκε σε ξεχωριστή μονάδα και σε κάθε προσομοίωση φορτώνεται μόνο ένα εξασθένισης διαδρομής και ένα εξασθένισης βλάστησης, δεν γίνεται να τα καλέσουμε όλα στην ίδια εκτέλεση.

Όμως στο Omnet++ έχουμε ένα στοιχείο που λείπει από το Shawn και αξιοποιούμε για την εξαγωγή στατιστικών της προσομοίωσης. Το στοιχείο αυτό είναι η διαχείριση των μηνυμάτων από το μοντέλο επικοινωνίας.

Στο Shawn τα μηνύματα τα διαχειρίζεται αποκλειστικά το μοντέλο εκπομπής. Αυτό τα μεταφέρει από τον ένα κόμβο στον άλλο ή τα απορρίπτει, χωρίς να επεμβαίνει πουθενά το μοντέλο επικοινωνίας του Shawn. Αντίθετα στο Omnet++ όλα τα μηνύματα περνάνε μέσα από το μοντέλο επικοινωνίας πριν πάνε από τον αποστολέα στον παραλήπτη. Δηλαδή αυτό κατέχει διπλό ρόλο σύμφωνα με την αρχιτεκτονική του Shawn, ή καλύτερα ακόμα τριπλό, ενσωματώνοντας σε μια σύνθετη μονάδα και τα τρία μοντέλα διαχείρισης της επικοινωνίας τους Shawn (Communication, Edge, Transmission Models).

Το μοντέλο επικοινωνίας προσθέτει σε κάθε μήνυμα παράμετρο με την ισχύ σήματος λήψης. Ο κόμβος πύλη που θα παραλάβει το μήνυμα αυτό τυπώνει στη μέθοδο *finish()* το σύνολο των μηνυμάτων που παρέλαβε από κάθε κόμβο και τη μέση ισχύ λήψης. Επιπλέον η υπομονάδα comm_out, που είναι και αυτή που καθορίζει την τελική απόφαση για την αποστολή ενός μηνύματος στον παραλήπτη του ή την απόρριψή του, διατηρεί στατιστικά για όλες τις αποστολές και απορρίψεις από κάθε αποστολέα. Αυτά τα στατιστικά εκτυπώνονται στη μέθοδο *finish().*

Παρουσιάζονται τα περιεχόμενα των αρχείων εξόδου “comm_out_rng_attenuation_report.txt” και “130_report.txt”, για την περιοχή της Καισαριανής, για μοντέλο διαδρομής FSP και μοντέλο Βλάστησης COST235 για βλάστηση χωρίς φύλλωμα:

```
------------------------------
Sensor     Sent  Dropped    Total

------------------------------
131         499   0           499
132         479   0           479
133         467   0           467
134         497   0           497
135         486   0           486
136         510   0           510
137         481   0           481
138         485   0           485
139         493   0           493
140         491   0           491
141         465   0           465
142         468   0           468
143         472   0           472
144         500   0           500
145         484   0           484
146         500   0           500
```

[comm_out_rng_attenuation_report.txt]

```
130  --------------------
sensor     count mean
131         499   -10.9027
132         479   -57.2688
133         467   -73.5387
134         497   -81.0682
135         486   -88.6148
136         510   -91.4945
137         481   -93.6002
138         485   -101.136
139         493   -101.458
140         491   -105.461
141         465   -95.7893
142         468   -86.9009
143         472   -79.7288
144         500   -75.3411
145         484   -60.8218
146         500   -54.3502
```

[130_report.txt]

Με διαδοχικές εκτελέσεις μπορεί κανείς να εξάγει στατιστικά για όλους τους συνδυασμούς μοντέλων. Γίνεται φανερό ότι η υλοποίηση του Omnet++ υστερεί σε αυτό το κομμάτι έναντι αυτής του Shawn. Βέβαια κάθε μοντέλο στο Omnet++ είναι από μόνο του πολύ πιο 'ελαφρύ' σαν κώδικας και καταναλωτής πόρων συστήματος, κυρίως δηλαδή μνήμης, σε σχέση με το μοντέλο επικοινωνίας του Shawn. Αν πάλι όμως κάποιος επιθυμεί τέτοια λειτουργικότητα μπορεί να δημιουργήσει μια μονάδα που να τα συνδυάζει όλα και να κάνει τον υπολογισμό με τον ίδιο τρόπο που γίνεται στο Shawn.

8.4 Οπτικοποίηση & Γραφικό Περιβάλλον

Στο μικρό αυτό εδάφιο παρουσιάζονται τα προϊόντα οπτικοποίησης του Shawn και η απεικόνιση του δικτύου στο γραφικό περιβάλλον του Omnett++. Το Castalia είναι σχεδιασμένο για να εκτελείται στην γραμμή εντολών, και ενώ δύναται να χρησιμοποιεί το γραφικό περιβάλλον Tkenv του Omnet++ δεν έχει καλή προσαρμογή και χωροθέτηση, με αποτέλεσμα η απεικόνιση ων κόμβων να είναι τελείως δυσνόητη και δύσχρηστη, αν όχι τελείως άχρηστη. Ένας λόγος που δεν υποστηρίζει με σθένος την υλοποίηση αυτή το Castalia είναι τα πάρα πολλά μηνύματα ελέγχου τα οποία στην γραφική απεικόνιση θα καθυστερούσαν αισθητά την προσομοίωση. Επίσης υποστηρίζεται από τον δημιουργό του ότι ετοιμάζεται και θα κυκλοφορήσει ειδικό εργαλείο για την οπτικοποίηση του προσομοιωτή.

8.4.1 Shawn: εκτύπωση του δικτύου και των συνδέσεων των κόμβων σε αρχείο pdf

Για το Shawn χρησιμοποιούμε την εργασία vis_single_snapshot προκειμένου να εκτυπώσουμε το δίκτυο σε αρχείο pdf [§1.7.3]. Πριν βέβαια προχωρήσουμε στην τελική εκτύπωση πρέπει να έχουμε ρυθμίσει όλες τις παραμέτρους αυτής σύμφωνα με τη παράγραφο §1.7.2. Αυτό που θα παίξει ρόλο στο τελικό προϊόν είναι το μοντέλο επικοινωνίας και οι παράμετροί του. Το μοντέλο ακμών θα δημιουργήσει ακμές στην απεικόνιση ανάμεσα σε δύο κόμβους 'u' και 'v', βάσει της απάντησης του μοντέλου επικοινωνίας στην ερώτηση '*bool can_communicate_uni(const Node& u, const Node& v)'*.

Όπως φάνηκε από την αξιολόγηση των προσομοιώσεων και τη σύγκλιση τιμών με τα δεδομένα παρατηρήσεων, η ονομαστική ευαισθησία του δέκτη δεν ανταποκρίνεται στις μετρήσεις. Αυτό φαίνεται και από την προσομοίωση, που ακόμα και σε περιπτώσεις μεγάλης σύγκλισης, δεν εξασφαλίζει τη μη συνδεσιμότητα των κόμβων που βάσει μετρήσεων δεν λαμβάνονται από το δέκτη.

Παρουσιάζεται για παράδειγμα στην Εικόνα 8.1 η απεικόνιση που προκύπτει για την ονομαστική ευαισθησία δέκτη και το μοντέλο διαδρομής FSP με μοντέλο βλάστησης COST235 χωρίς φύλλωμα για 50% της διαδρομής. Ο συνδυασμός αυτός παρουσίασε τη μεγαλύτερη σύγκλιση με τις μετρήσεις στην προηγούμενη ανάλυση [§8.3.1]. Όπως εύκολα παρατηρεί κανείς έχουμε συνδεσιμότητα ακόμα και για κόμβους που οι μετρήσεις έδειξαν πως δεν λαμβάνεται σήμα.

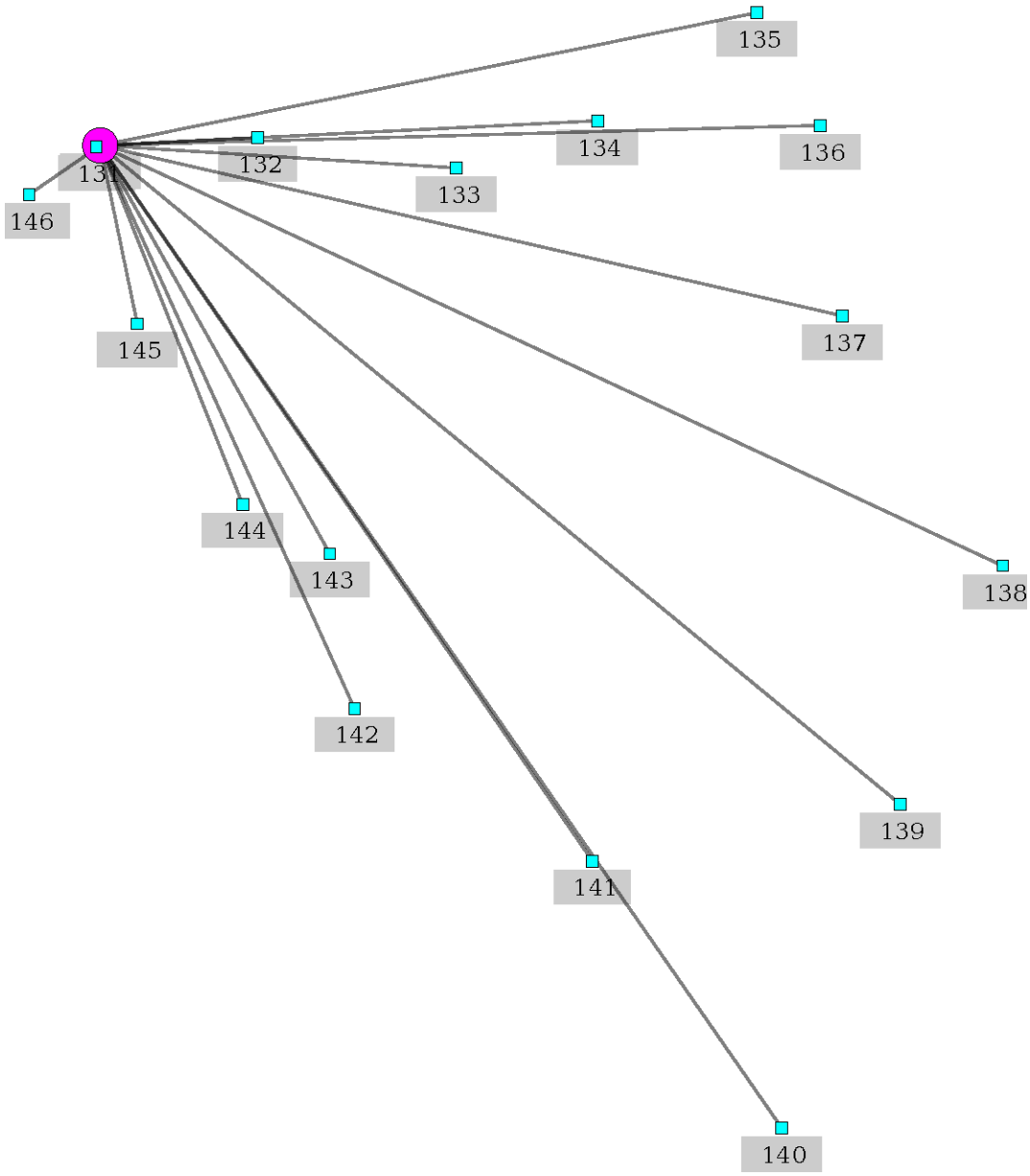
Για να δούμε πραγματικά την σύγκλιση του προσομοιωτή στις μετρήσεις, χρησιμοποιούμε δοκιμαστική ευαισθησία για τον δέκτη ίση με την αμέσως μικρότερη κατά το βήμα διαβάθμισης ισχύ από την μικρότερη των μετρήσεων. Για την Καισαριανή η μικρότερη ισχύς που καταγράφεται είναι -96 dBm. Θέτουμε ως δοκιμαστική ευαισθησία -97 dBm. Τα αποτελέσματα φαίνονται στην Εικόνα 8.2. Όπως παρατηρούμε οι κόμβοι που αναμένουμε να μην συνδέονται παρίστανται χωρίς σύνδεση.

Η ευαισθησία δοκιμής δεν είναι απαραίτητα σωστή, ούτε είναι τρόπος αυτός για να υπολογίσουμε την πραγματική ευαισθησία του δέκτη. Μέχρι και την τιμή -101 dBm παίρνουμε αποτέλεσμα ανάλογο των μετρήσεων στην απεικόνιση. Η πραγματική ευαισθησία υπολογίζεται με βαθμονόμηση των συσκευών σε περιβάλλον παρόμοιο με αυτό των μετρήσεων αν είναι δυνατό, ή στο εργαστήριο.

Την ίδια δοκιμαστική τιμή για την ευαισθησία του δέκτη ελέγχουμε στην περιοχή της Πεντέλης. Εφαρμόζουμε μοντέλο εξασθένισης διαδρομής FSP με εξασθένιση βλάστησης COST235 χωρίς φύλλωμα στο 30% της διαδρομής. Ο συνδυασμός αυτό παρουσίασε τη δεύτερη μεγαλύτερη σύγκλιση με τις μετρήσεις στην προηγούμενη ανάλυση [§8.3.1]. Παρατηρούμε την εικόνα που αναμένεται από τις μετρήσεις παρατήρησης, με εξαίρεση τον κόμβο 717, που εμφανίζει συνδεσιμότητα.

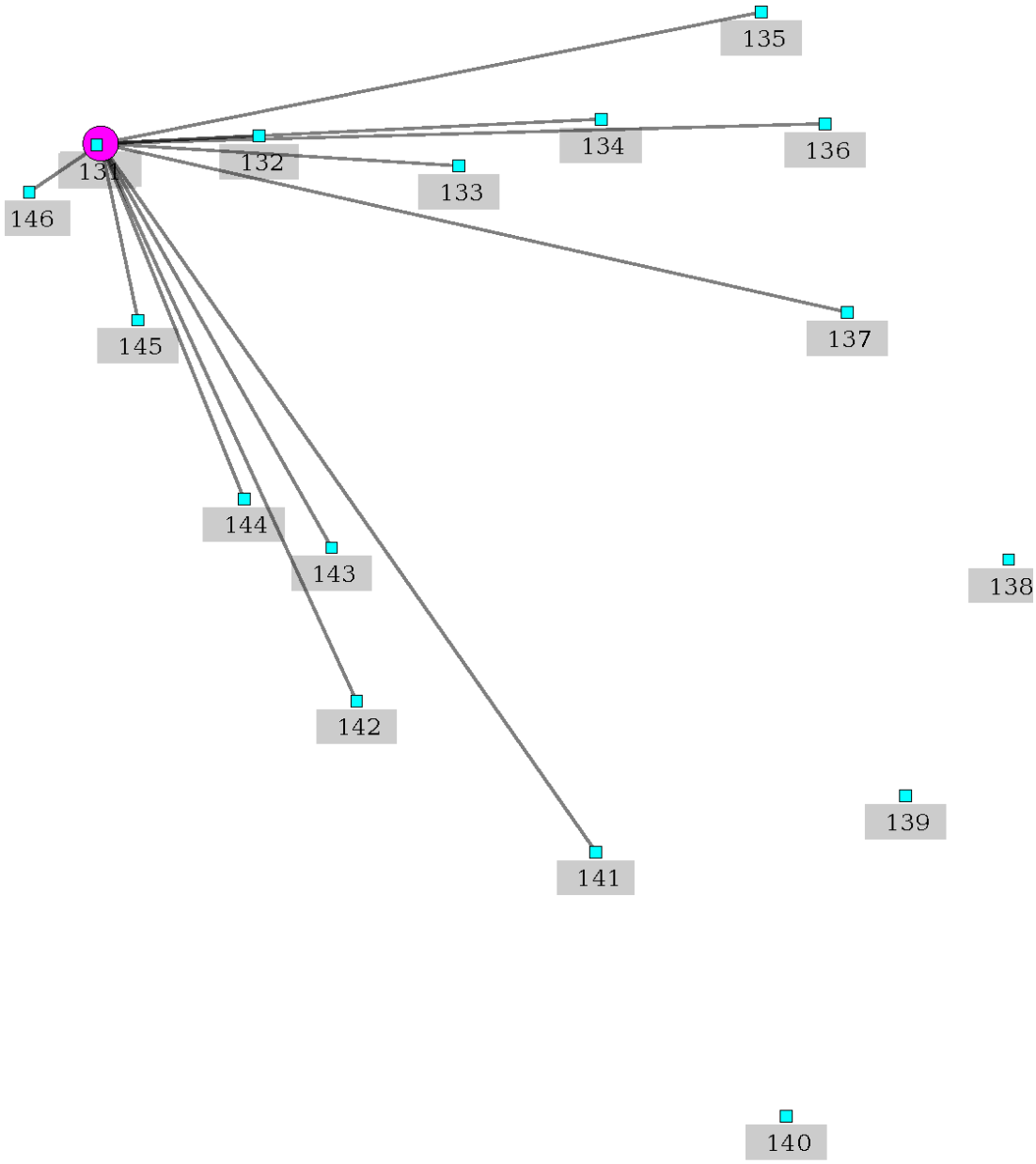
Μπορούμε να πούμε ότι με τη προσεγγιστική τιμής της ευαισθησίας του δέκτη και τα μοντέλα που συγκλίνουν με τις μετρήσεις, επιβεβαιώνουμε αυτές και διαθέτουμε λύσεις στην πρόβλεψη σεναρίων και συνδεσιμότητας κόμβων σε αυτές τις περιοχές. Προκειμένου να δώσουμε πραγματική αξιοπιστία και κάποιο επίπεδο εμπιστοσύνης για τα μοντέλα αυτά, πρέπει να διαθέτουμε ένα ικανό δείγμα μετρήσεων που να επιτρέπει αυτή την ανάλυση.

Path Loss: Free Space Propagation
Foliage Loss: COST235_OUT 50%
NOMINAL/FACTORY Receiver_Sensitivity -112 dBm

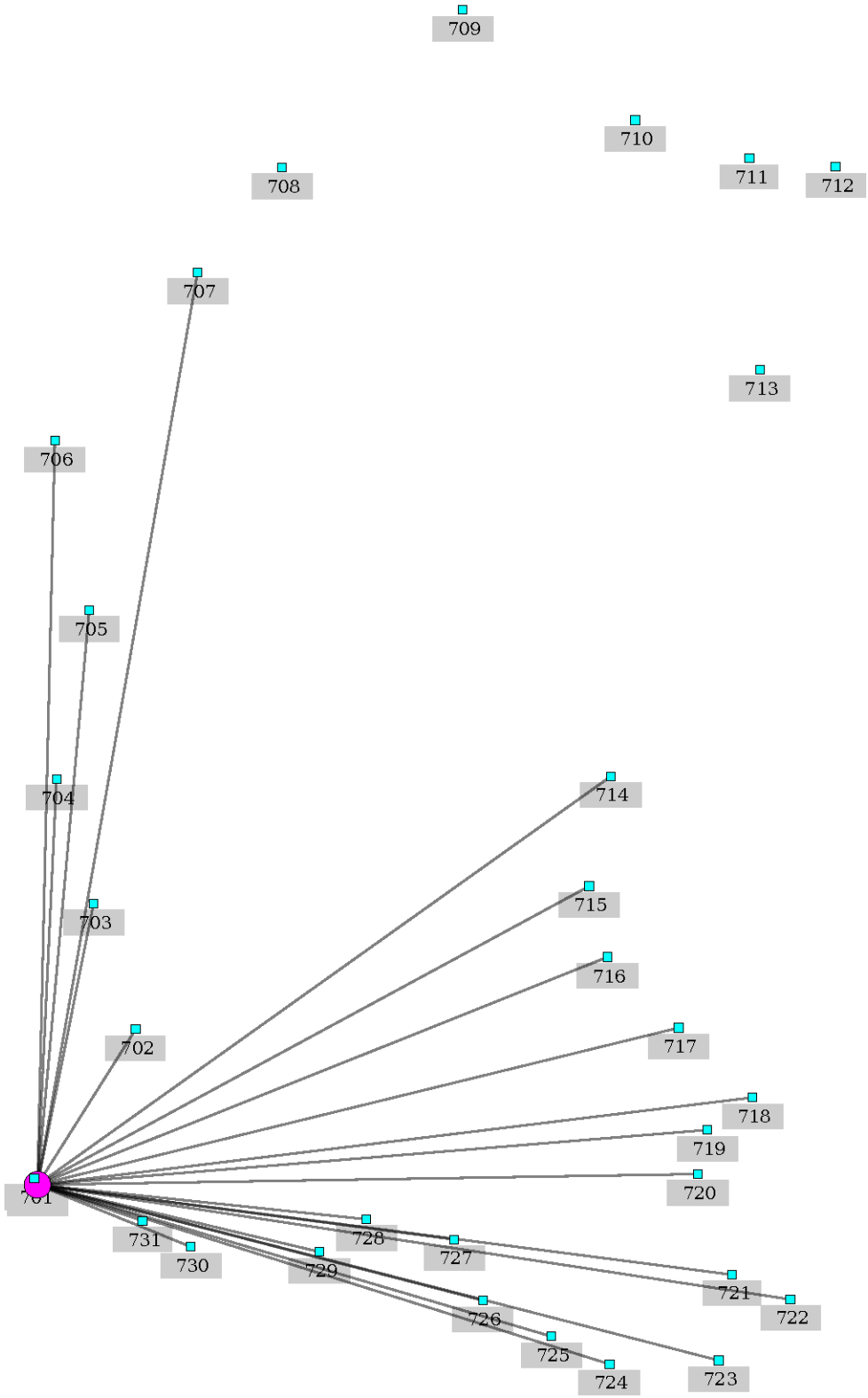


Εικόνα 8.1

Path Loss: Free Space Propagation
Foliage Loss: COST235_OUT 50%
EVALUATION Receiver_Sensitivity -97 dBm



Εικόνα 8.2


Path Loss: Free Space Propagation
Foliage Loss: COST235_OUT 30%
EVALUATION Receiver_Sensitivity -97 dBm
709
710
708
711
712
707
713
706
705
704
714
715
703
716
702
717
718
719
720
731
728
727
730
729
721
722
726
725
724
723

Εικόνα 8.3

8.4.2 Omnet++: Γραφικό Περιβάλλον Tkenv

Το Omnet++ έχει το μεγάλο πλεονέκτημα της δυνατότητας να εκτελεί προσομοιώσεις τόσο σε γραμμή εντολών όσο και σε γραφικό περιβάλλον. Το δεύτερο είναι υλοποιημένο σε tcl/tk γλώσσα προγραμματισμού και παρέχει πληθώρα εργαλείων ελέγχου της προσομοίωσης. Δεν είναι στους στόχους της εργασίας η παρουσίαση αυτών, τα οποία συν τοις άλλοις καλύπτονται πληρέστατα από το εγχειρίδιο του Omnet++ [17].

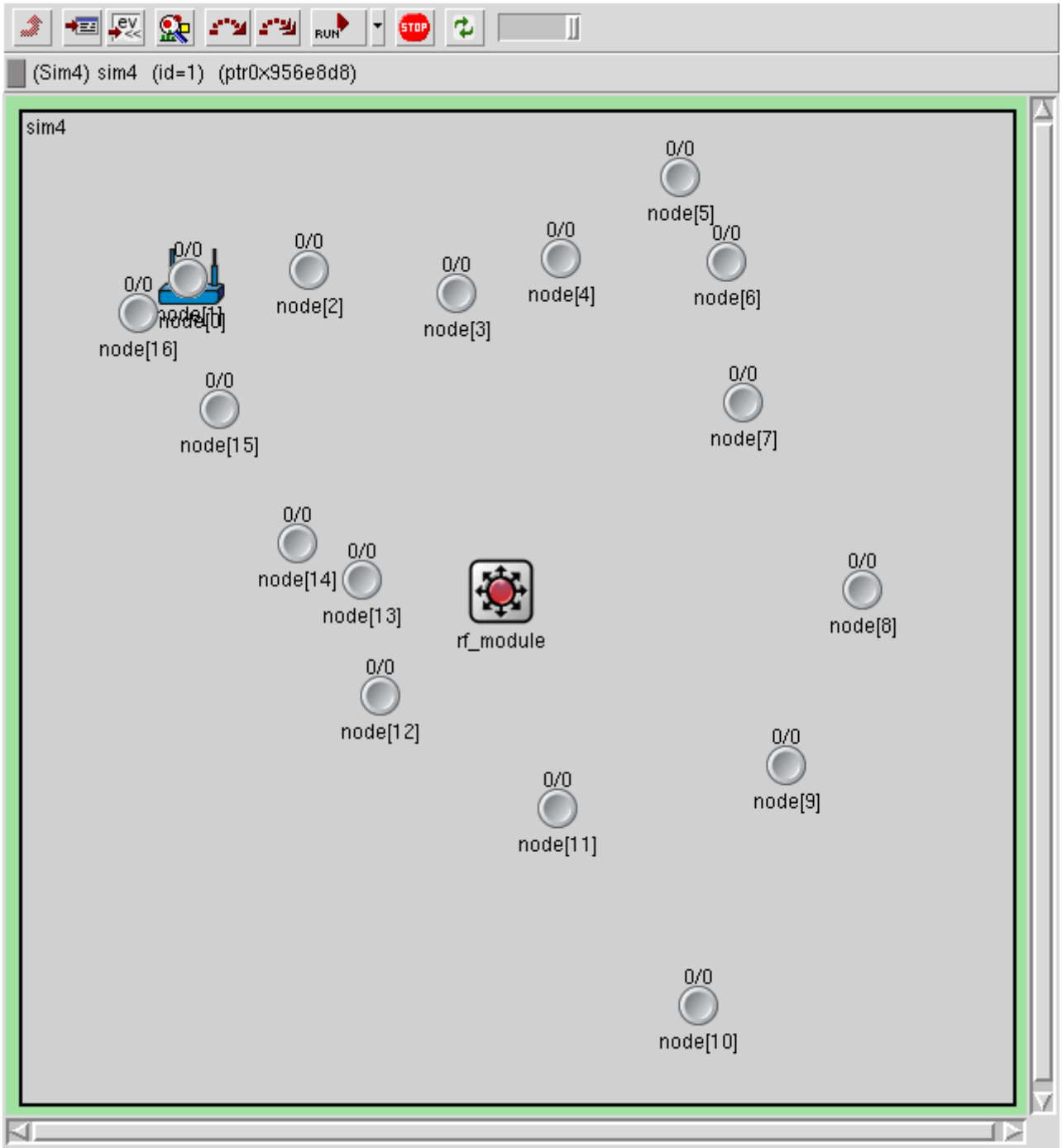
Στις επόμενες εικόνες παρουσιάζονται τα δίκτυα Καισαριανής και Πεντέλης που μελετώνται στην εργασία, και η δομή του μοντέλου επικοινωνίας, όπως δημιουργήθηκε για το Omnet++. Επίσης αποδίδονται και δύο απεικονίσεις προσομοίωσης σε εξέλιξη για κάθε δίκτυο. Σε αυτές διακρίνονται μηνύματα που κινούνται από τι μοντέλο επικοινωνίας σε κόμβους και το αντίθετο. Το μοντέλο επικοινωνίας είναι στο κέντρο κάθε απεικόνισης και συμβολίζεται με ένα εικονίδιο που παραπέμπει σε δρομολογητή. Τα δίκτυα χρησιμοποιούν τα ίδια μοντέλα που εφαρμόσαμε στην απεικόνιση του Shawn[§8.4.1].

Για τις απεικονίσεις ενεργής προσομοίωσης χρησιμοποιείται φαινόμενο διασποράς (fading) με τυπική απόκλιση 6 dBm. Με τον τρόπο αυτό προστίθεται και κάποια τυχαιότητα στην επικοινωνία, γεγονός που καταδεικνύεται από την απεικόνιση. Πάνω από κάθε κόμβο αναγράφεται ο αριθμός μηνυμάτων που απέστειλε επιτυχώς και ο συνολικός αριθμός που έχει στείλει μέχρι αυτή τη στιγμή. Ο διαφορετικός χρωματισμός αποδίδει την υπεροχή της αποστολής έναντι της απόρριψης μηνυμάτων.

Τέλος παρατίθενται και μια εικόνα του κυρίως περιβάλλοντος, όπου διακρίνονται οι αναφορές της προσομοίωσης, τα μηνύματα και οι μονάδες με τις παραμέτρους τους.

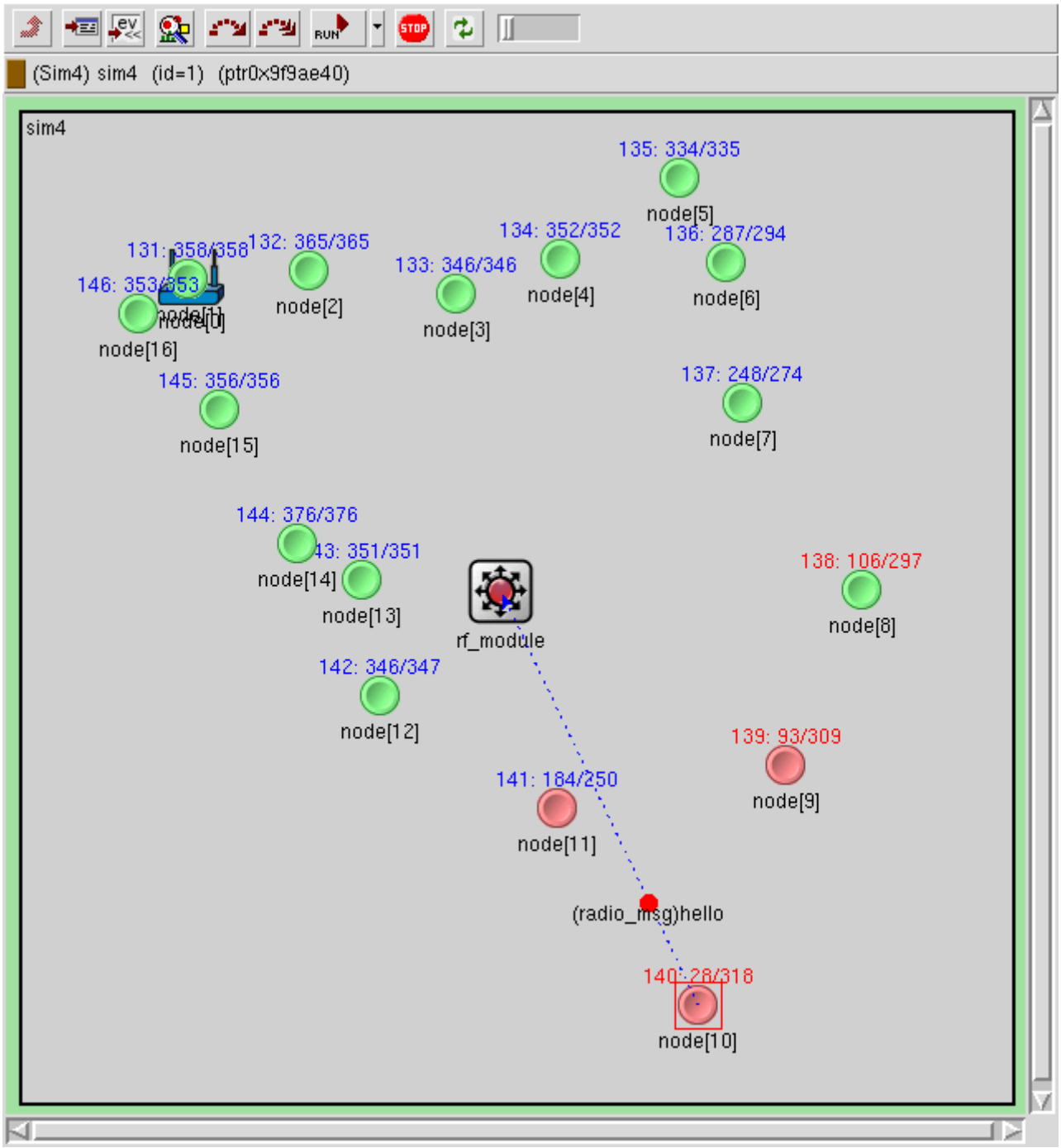
Δίκτυο Καισαριανής (πριν την προσομοίωση)

Path Loss: Free Space Propagation
Foliage Loss: COST235_OUT 50%
EVALUATION Receiver_Sensitivity -97 dBm



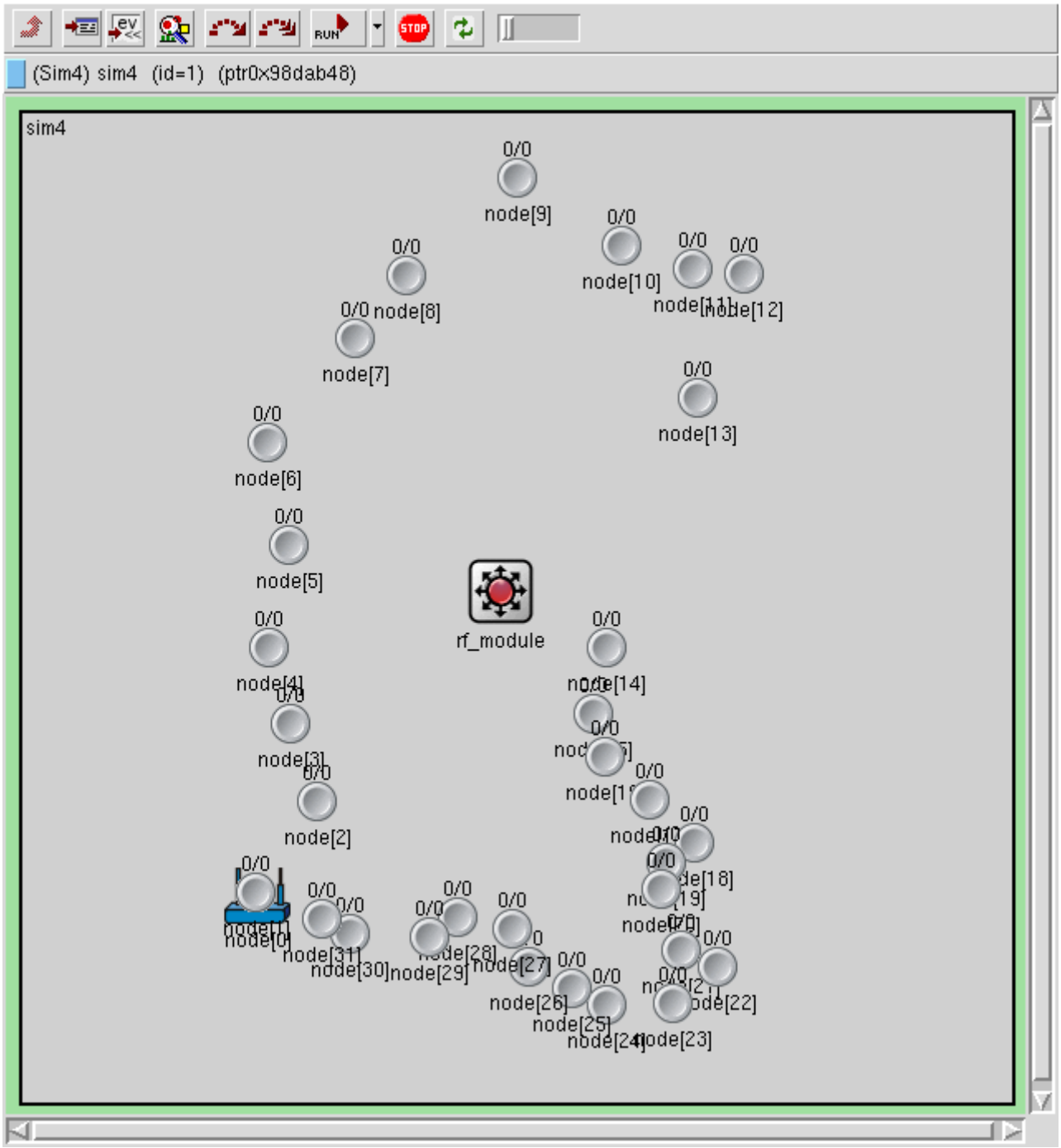
Εικόνα 8.4

Δίκτυο Καισαριανής (προσομοίωση σε εξέλιξη)
Path Loss: Free Space Propagation
Foliage Loss: COST235_OUT 50%
EVALUATION Receiver_Sensitivity -97 dBm



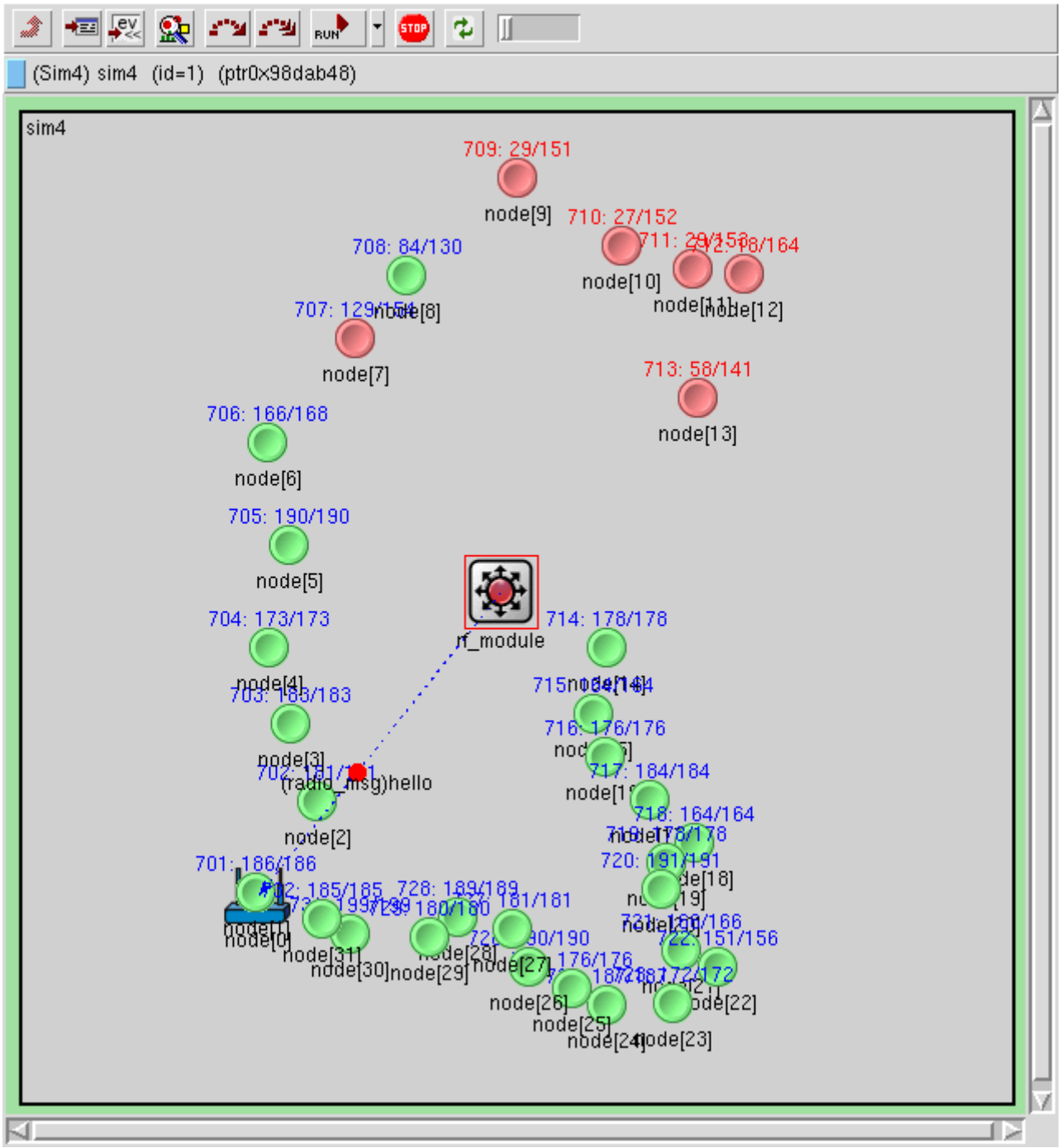
Εικόνα 8.5

Δίκτυο Πεντέλης (πριν την προσομοίωση)
Path Loss: Free Space Propagation
Foliage Loss: COST235_OUT 30%
EVALUATION Receiver_Sensitivity -97 dBm



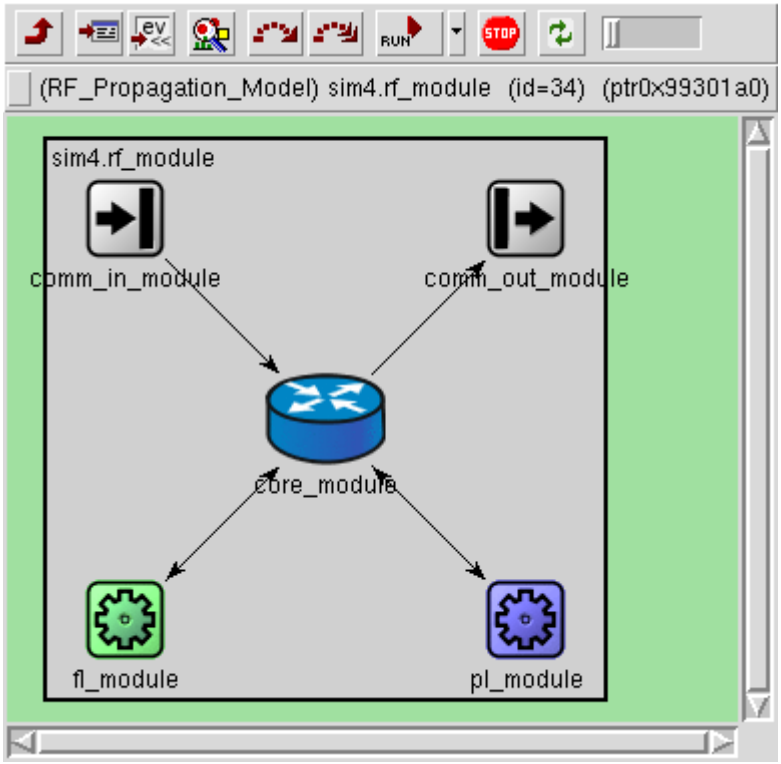
Εικόνα 8.6

Δίκτυο Πεντέλης (προσομοίωση σε εξέλιξη)
Path Loss: Free Space Propagation
Foliage Loss: COST235_OUT 30%
EVALUATION Receiver_Sensitivity -97 dBm



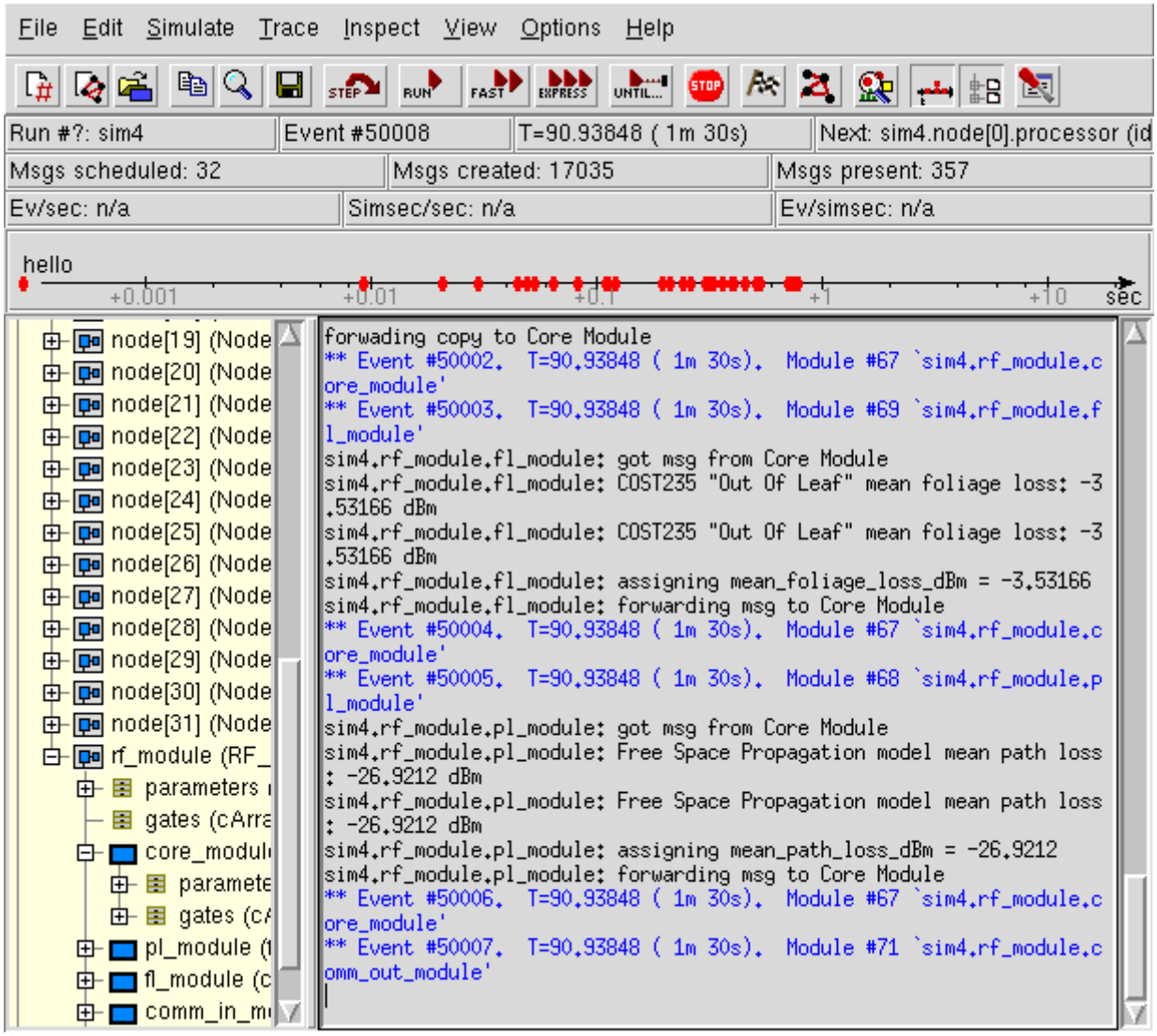
Εικόνα 8.7

RF_Propagation Model: απεικόνιση



Εικόνα 8.8

Omnet++ Tkenv: κύριο παράθυρο



Εικόνα 8.9

Συμπεράσματα

& Προτάσεις

Η επαφή με το αντικείμενο των ασυρμάτων δικτύων αισθητήρων, και ακόμα πιο ειδικά με τη διαδικασία προσομοίωσής τους, ήταν ιδιαίτερα ενδιαφέρουσα και ωφέλιμη. Αναπτύχθηκε σε βαθμό αρκετά αναλυτικό η αντίληψη και λειτουργία των προσομοιωτών αυτών, και αναδείχθηκαν θέματα που αντιμετωπίζουν και οι δυνατότητες εφαρμογής και ανάπτυξης που έχουν. Ακόμα μεγαλύτερη είναι η γέννεση ιδεών που επιφέρει μια τέτοια μελέτη, καθώς η επαφή με τα προβλήματα και τις πιθανές λύσεις τους δίνει πάντα περισσότερους από ένα δρόμους επίλυσης.

Προσομοιωτές Ασυρμάτων Δικτύων Αισθητήρων

Οι προσομοιωτές που εξετάστηκαν στην εργασία αυτή δεν είναι οι μόνοι διαθέσιμοι για τον συγκεκριμένο τύπο προσομοίωσης, δηλαδή ασυρμάτων δικτύων αισθητήρων. Ενδεχομένως να μην είναι οι καλύτεροι για τη συγκεκριμένη εφαρμογή. Σίγουρα όμως η ανάλυσή τους, η ανάπτυξη εφαρμογών και η υλοποίηση επιτυχών προσομοιώσεων, ήταν μια εργασία πολύ ευχάριστη, που προσέφερε πάρα πολλές νέες γνώσεις, ιδέες και ένα αίσθημα ικανοποίησης αλλά και ανησυχίας για περαιτέρω ανάπτυξη. Είναι τέτοια η φύση του προγραμματισμού που δεν υπάρχει στην πραγματικότητα η έννοια της τελικής έκδοσης, παρά πιο σωστά της τελευταίας.

Για κάθε λογισμικό προσομοίωσης που εξετάστηκε στην εργασία, τα προβλήματα που αντιμετωπίστηκαν και οι αντίστοιχες λύσεις ήταν αρκετά διαφορετικά, όπως διαφορετικές είναι και οι σκέψεις για την συνέχεια που αυτά μπορούν να έχουν. Αν και ο τελικός στόχος ήταν κοινός, και εν τέλει εξετάστηκαν σε αυτόν στο Κεφ. 8, η πορεία που ακολουθήθηκε στην κάθε περίπτωση ήταν διαφορετική.

Shawn

Το Shawn από την αρχή φάνηκε ιδιαίτερα ελκυστικό σαν διανομή κώδικα. Και η αλήθεια είναι ότι είναι οργανωμένο σε εξαιρετικά καλογραμμένη μορφή και διάταξη κώδικα. Αυτό σε ένα βαθμό βοήθησε να ξεπεραστεί το πρόβλημα της ελάχιστης τεκμηρίωσης για το λογισμικό αυτό. Παρότι και στον κώδικα υπάρχει αρκετή τεκμηρίωση για τις πιο σημαντικές δομές, όπως για τα μοντέλα επικοινωνίας ακμών και εκπομπής, εν τούτοις δεν καλύπτονται όλες οι λειτουργίες τους και άλλες δομές που αξιοποιούν, αφήνοντας σε πολλά σημεία αμφιβολίες για τον τρόπο που εκτελούνται διάφορες διαδικασίες της προσομοίωσης.

Χάρη στην οργανωμένη και καλογραμμένη διάταξη του πηγαίου κώδικα το πρόβλημα αυτό ξεπεράστηκε. Η ανάλυση της λειτουργίας και των δομών του Shawn κατ' αυτόν τον τρόπο δεν ήταν καθόλου κουραστική, αλλά αντίθετα βοήθησε στην καλύτερη κατανόησή του και την προσαρμογή στις ανάγκες της μελέτης. Δεν είναι τυχαίο ότι η υλοποίηση που έγινε στο Shawn και ο κώδικας που γράφτηκε και χρησιμοποιήθηκε, είναι πολύ μεγαλύτερα από τα αντίστοιχα μεγέθη των Omnet++ και Castalia. Είναι ενδεικτική για παράδειγμα η σύνθετη δόμηση της Παραγωγικής τάξης του RF NT μοντέλου επικοινωνίας, η οποία σε κώδικα είναι εξίσου μεγάλη με το ίδιο το μοντέλο [§5.3.3]. Στην οργάνωση αυτή οδήγησε αποκλειστικά η δελεαστική δυνατότητα που έδινε το λογισμικό να υλοποιήσει κανείς την συλλογή τιμών παραμέτρων με δύο διαφορετικούς τρόπους.

Προϊόν της πολύ καλής αποδοχής που είχε το Shawn ήταν ένα σύνολο εργασιών [§5.5-§5.6] επεξεργαστών [§5.7] και μοντέλων επικοινωνίας [§5.3], με ναυαρχίδα όλων αυτών των νέων τάξεων το μοντέλο επικοινωνίας RF NT. Το μοντέλο αυτό, ως προϊόν της ανάλυσης της μετάδοσης ραδιοσυχνοτήτων με μεγάλης κλίμακας μοντέλα εξασθένισης της ισχύος, είναι η απάντηση στην απλοϊκή προεπιλεγμένη προσέγγιση του Shawn να εξετάζει την επικοινωνία δύο κόμβων βάσει ενός στατικού εύρους επικοινωνίας, κοινό για όλες τις συνδέσεις. Παρότι η λογική αυτή είχε και πιο αναλυτικές εκφράσεις, δεν έφτανε σε ανάλυση της επικοινωνίας σε φυσικό επίπεδο.

Το Shawn αποτέλεσε τον καθοδηγητή της εργασίας αυτής. Κάθε ενέργεια που έγινε στα άλλα δύο λογισμικά είναι με κάποιο τρόπο συνδεδεμένη με κάποια υλοποίηση που είχε προηγηθεί στο Shawn. Ίσως αυτό να ήταν άδικο για το Omnet++ και το Castalia, αλλά είναι και η πραγματικότητα. Ενδεικτικό παράδειγμα είναι το μοντέλο επικοινωνίας που αναπτύχθηκε στο Omnet++ [§6.4.2], το οποίο δεν είναι παρά ένα τεμαχισμένο σε μικρότερες μονάδες RF NT μοντέλο επικοινωνίας του Shawn.

Μέσα στα άλλα το Shawn αφήνει και τους περισσότερους ανοιχτούς λογαριασμούς. Λόγω των απαιτήσεων της εργασίας δεν έγινε εμβάθυνση στις ανώτερες τεχνικές τοπολογίας που διαθέτει, όπως δεν έγινε και για την κινητικότητα των κόμβων και σωρεία άλλων πολύ ενδιαφερουσών δυνατοτήτων. Οι περισσότερες από αυτές είναι ανεπτυγμένες έξω από τα πλαίσια του κυρίως συστήματος του προσομοιωτή, και συνιστούν μεγάλο πεδίο εφαρμογών.

Η εφαρμογή των σεναρίων επίσης ανέδειξε τις δυνατότητες του Shawn. Η εργασία που πραγματοποίησε την συλλογή και εκτύπωση όλων των στατιστικών εξασθένισης [§5.6] κατασκευάστηκε πολύ εύκολα σε σύντομο χρόνο, χάρη στη σαφή δομή και τις δυνατότητες του λογισμικού. Το γεγονός ότι δεν χρειάζεται να εκτελεστεί καν προσομοίωση, είναι μοναδικό στοιχείο του Shawn και δεν συναντάται στους άλλους δύο προσομοιωτές. Πρόκειται για ένα από τα μεγαλύτερα πλεονεκτήματά του, ανάμεσα στα άλλα.

Μέσα από όλες τις εργασίες αυτές, της ανάλυσης, της ανάπτυξης και της εφαρμογής, το Shawn απέδειξε ότι είναι ένα ικανό σύνολο κώδικα για την μελέτη και προσομοίωση ασυρμάτων δικτύων αισθητήρων. Ως τέτοιο μπορεί να βελτιωθεί ακόμα περισσότερο, εισάγοντας στοιχεία που του λείπουν, όπως ένα γραφικό περιβάλλον διαχείρισης των προσομοιώσεων.

Omnet++

Το Omnet++ είναι το μοναδικό από τα τρία πακέτα λογισμικού που συνοδευόταν από πλήρη τεκμηρίωση. Ένα πληρέστατο εγχειρίδιο [17], κώδικας με εσωτερική τεκμηρίωση όλων των μελών του, και μια μεγάλη κοινότητα [22] με πολύτιμη συνεισφορά επιπλέον βοήθειας, καθιστούν το Omnet++ ένα από τα πιο ελκυστικά λογισμικά προσομοίωσης δικτύων γενικότερα. Ίσως αυτό να λειτούργησε αρνητικά στην παρούσα εργασία, δεδομένου ότι η πιο έντονη ενασχόληση με το Shawn προέκυψε λόγω έλλειψης τεκμηρίωσης στη δική του περίπτωση. Έτσι η υλοποίηση στο Omnet++ έγινε αφού είχε ήδη επικρατήσει το μοντέλο ανάπτυξης του Shawn στην τακτική ανάπτυξης προσομοιωτών.

Το γεγονός αυτό όμως με κανένα τρόπο δεν υποβαθμίζει τις δυνατότητες του λογισμικού και την υλοποίηση που εν τέλει δημιουργήθηκε. Σε αυτό πρέπει να συνυπολογιστεί το γεγονός της 'λευκής σελίδας' πάνω στην οποία σχεδιάστηκε ο προσομοιωτής στο Omnet++. Είναι ο μόνος που κατασκευάστηκε από το μηδέν, σε αντίθεση με το Shawn όπου χρησιμοποιούμε πολλές έτοιμες δομές. Παρά τις επιρροές από το Shawn ο προσομοιωτής του Omnet++ δομήθηκε με απόλυτο σεβασμό στην αρχιτεκτονική υλοποίησης που προωθεί το λογισμικό.

Πολύ μεγάλη σημασία για το Omnet++ έχει και η δυνατότητα εκτέλεσης προσομοιώσεων με δύο διαφορετικούς τρόπους: στη γραμμή εντολών και σε γραφικό περιβάλλον εργασίας [§2.4]. Η πρώτη περίπτωση όπως χαρακτηριστικά φαίνεται στο Castalia είναι πολύ χρήσιμη για γρήγορη εκτέλεση μεγάλων σεναρίων προσομοίωσης, ή και την εκτέλεση πολλών προσομοιώσεων σε σειρά. Από την άλλη το γραφικό περιβάλλον είναι ένα πολύ χρήσιμο εργαλείο για την παρακολούθηση της εξέλιξης μιας προσομοίωσης και τον εντοπισμό σφαλμάτων. Επιπλέον δίνει την δυνατότητα επίδειξης ζωντανών προσομοιώσεων, πακέτο πολύ πιο ελκυστικό από την προβολή στατιστικών δεδομένων.

Παρότι δεν έχει τον χαρακτήρα του προσομοιωτή ασυρμάτων δικτύων αισθητήρων, αλλά αντίθετα βασίζεται σε μια πιο ενσύρματη λογική, το Omnet++ στέκεται πολύ δυνατό απέναντι σε αυτή τη πρόκληση. Το απέδειξε τόσο ο νέος προσομοιωτής που κατασκευάστηκε, όσο και το Castalia, που είναι εδώ και καιρό εδραιωμένο στην ασύρματη κοινότητα σαν ικανό λογισμικό σε αυτή τη κατεύθυνση. Αν δε η πρώτη προσπάθεια που παρουσιάστηκε σε αυτή την εργασία είναι πετυχημένη, αναρωτιέται κανείς πόσο καλύτερη μπορεί να είναι μια μελλοντική βελτιωμένη υλοποίηση. Το Omnet++ διαθέτει πολλά εργαλεία, και ακόμα πιο πολλές βοήθειες, ώστε να θεωρείται πληρέστατο για την υλοποίηση προσομοιωτών ασυρμάτων δικτύων αισθητήρων πολλές τάξεις ανώτερους από αυτόν που παρουσιάστηκε σε αυτήν την εργασία.

Castalia

Το Castalia μπορεί κανείς να το δει στην εργασία αυτή σαν τον παραμελημένο φτωχό συγγενή των άλλων δύο λογισμικών. Είναι ενδεχομένως ένα λάθος συμπέρασμα που μπορεί κανείς να βγάλει συγκρίνοντας το μέγεθος της επέμβασης σε αυτό σε σχέση με το Shawn και το Omnet++. Το Castalia ήταν σχεδόν 'έτοιμο' για τρέξιμο. Ακόμα και οι επεμβάσεις που έγιναν μπορούσαν να μείνουν σε πιο μικρή έκταση. Το ασύρματο κανάλι επικοινωνίας για παράδειγμα τροποποιήθηκε κυρίως για λόγους τάξης και αφαίρεσης περιττών ενεργειών στην απλοϊκή προσομοίωση που υλοποιήθηκε.

Μπορεί βέβαια και το Castalia να μην είχε κάποια αναλυτική τεκμηρίωση, αλλά σαν προσομοιωτής του Omnet++ επωφελούνταν της δικής του. Πέρα από το σημείο αυτό, και στο Castalia αναλύθηκε ο πηγαίος κώδικας προκειμένου να γίνουν κατανοητές οι δομές του και ο τρόπος λειτουργίας των προσομοιώσεων. Δυστυχώς η οργάνωση και ο τρόπος γραφής δεν ήταν ιδιαίτερα ευανάγνωστος και βολικός, γεγονός που είναι και το μόνο μελανό σημείο του λογισμικού.

Κατά τα άλλα είναι το μόνο που εισάγει την ανάλυση στο MAC επίπεδο της επικοινωνίας. Παρότι δεν εστιάσαμε σε αυτή και εν τέλει την προσπεράσαμε χάρη της απλής προσέγγισης που υλοποιήσαμε, είναι το μεγαλύτερο εργαλείο του Castalia και το τεράστιο πλεονέκτημά του. Αν υπάρχει κάτι που αξίζει να αναλύσει κανείς στο Castalia αυτό είναι η MAC μονάδα σε συνεργασία με τις γειτονικές της. Είναι πολύ σημαντικό να αναφερθεί αυτό, ώστε να μην υπάρχει η παραμικρή αμφιβολία για την ικανότητα και την αξία του λογισμικού. Απλά αδικείται από τους απλούς σκοπούς αυτής της εργασίας.

Είναι βέβαιο ότι σε μια πιο σύνθετη και αναλυτική μελέτη, η οποία θα εξετάζει σε βάθος το MAC επίπεδο επικοινωνίας, το Castalia θα φανεί εξαιρετικό βοήθημα και πλατφόρμα υλοποίησης. Είναι δε υπό διαρκή ανάπτυξη, οπότε αξίζει να αναμένει κανείς και ακόμα πιο βελτιωμένες λειτουργίες σε αυτό το επίπεδο.

Επίλογος

Οι τρεις προσομοιωτές, Shawn, Omnet++ και Castalia, αναλύθηκαν σε επίπεδο κώδικα, προσαρμόστηκαν στις ανάγκες της εργασίας μέσω του προγραμματισμού στη C++ γλώσσα προγραμματισμού, και εκτέλεσαν τα σενάρια αξιολόγησης για τις περιοχές της Καισαριανής και Πεντέλης που εξετάστηκαν στο τελευταίο κεφάλαιο. Πολλά συμπεράσματα βγάζει κανείς από όλη αυτή την ανάλυση. Κύρια και πάνω από όλα όμως δημιουργήθηκε μια βάση και ένα υπόβαθρο για πιο μεγάλες επεμβάσεις και αναλυτικότερες υλοποιήσεις. Το τελικό συμπέρασμα είναι ότι δεν υπάρχει επίλογος για την ανάπτυξη και την πρόοδο.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Έντυπη Βιβλιογραφία:

1. **Nikopolitidis P.** [et. al.], Wireless networks, John Wiley and Sons Ltd, 2003
2. **Pahlavan Kaveh & Krishnamurthy Prashant,** Principles of wireless networks : a unified approach, Prentice Hall, 2002
3. **Parsons J.D.**, The Mobile Radio Propagation Channel, Second Edition, John Wiley & Sons Ltd, 2000
4. **Rappaport Theodore S.**, Wireless communications : principles and practice, Prentice Hall, 1996
5. **Seybold John S.**, Introduction to RF propagation, Wiley-Interscience, 2005
6. **Stallings William**, Wireless communications and networking, Prentice Hall, 2002
7. **Stroustrup Bjarne**, Programming Principles and Practice Using C++, Pearson Education, Inc., 2009
8. **Stroustrup Bjarne**, The C++ Programming Language, Special Edition, AT&T, 2000

Ηλεκτρονική Βιβλιογραφία

9. **Boulis Athanassios**, Castalia: A simulator for Wireless Sensor Networks Version 1.3, User's Manual, National ICT Australia, Jan 2008
10. **Baumgartner Tobias**, Shawn: Distributed Programming, University of Lubeck, Germany, Nov. 2008 (pdf document)
11. **Baumgartner Tobias**, Shawn: Visualization and Mobility, University of Lubeck, Germany, Nov. 2008 (pdf document)
12. **Becker Claudia**, Shawn: Configuration, Institute of Telematics, University of Lubeck, Germany, Nov. 2008 (pdf document)
13. **Becker Claudia**, Shawn: Installation (on Windows), Institute of Telematics, University of Lubeck, Germany, Nov. 2008 (pdf document)
14. **Brinkmeier Ingo** Shawn: Controlling the Simulation, University of Lubeck, Germany, Nov. 2008 (pdf document)
15. **Gabor Tabi**, Get into GNED: An itroduction to the GNED editor of OMNEST/OMNET++ (pdf document)
16. **Kroller Alex**, Shawn: Physical Environment... and Logging, University of Lubeck, Germany, Nov. 2008 (pdf document)
17. **Varga Andras**, OMNeT++ Discrete Event Simulation System Version 3.2, User Manual, 2005 (pdf document)

Διαδικτυακοί τόποι:

18. **Castalia Homepage**, http://castalia.npc.nicta.com.au/ (26/02/10)

19. **Robert Lipe, et al.** GPSBabel: GPS Data Converter, http://www.gpsbabel.org/ (26/02/10)
20. **Gerald Evenden, et al.**, PROJ.4: Cartographic Projections Library , http://trac.osgeo.org/proj/ (26/02/10)

21. **"Shawn Web Page"**, http://shawnwiki.coalesenses.com/index.php?title=Main_Page (26/02/10)

22. **Varga Andras, et el.** Omnet++ Community Site, http://www.omnetpp.org/ (26/02/10)

23. **"Wikipedia",** Box–Muller transform, http://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform (26/02/10)

ΠΑΡΑΡΤΗΜΑ

Οδηγίες Χρήσης της εικονικής μηχανής

Για τις ανάγκες της διπλωματικής εργασίας δημιουργήθηκε εικονικό μηχάνημα (virtual machine) στο οποίο και εγκαταστάθηκαν τα τρία πακέτα λογισμικού, Shawn, Omnet++ και Castalia. και αναπτύχθηκε ο νέος κώδικας. Για το σκοπό αυτό χρησιμοποιήθηκε η πλατφόρμα εικονικών συστημάτων VirtualBox {4} της Oracle Corporation, η οποία διατίθεται δωρεάν ως ελεύθερο λογισμικό σύμφωνα με τη γενική δημόσια άδεια GNU GPL (General Public Lisence) {3}. Στη πλατφόρμα αυτή δημιουργήθηκε εικονικό μηχάνημα με λειτουργικό σύστημα Debian GNU/Linux{1} το οποίο είναι επίσης ανοιχτού κώδικα και ελεύθερης διανομής. Επιλέχθηκε αυτή η υλοποίηση κυρίως για τις δυνατότητες των δύο συστημάτων (Εικονική Πλατφόρμα και Λειτουργικό Σύστημα) και ειδικά για το Debian λόγω της πληρότητας σε έτοιμα πακέτα που παρέχει και της δυνατότητας εύκολης εγκατάστασης σε μινιμαλιστική λογική. Κάθε άλλη διανομή Linux/Unix μπορεί να χρησιμοποιηθεί κατά προτίμηση, όπως και λειτουργικά συστήματα Windows της Microsoft ή και MacOS της Apple.

Η εικονική μηχανή εγκαθίσταται σύμφωνα με τις οδηγίες του διανομέα {5}. Σε αυτή φορτώνουμε το αρχείο “debian-WSN.vdi” που περιέχει το εικονικό μηχάνημα. Το αρχείο αυτό βρίσκεται στο DVD οπτικό δίσκο που συνοδεύει την εργασία. Στο λειτουργικό σύστημα έχει καταχωρηθεί μόνο ένας χρήστης, ο root. Ο κωδικός είναι επίσης root. Αν εγκαταστήσετε το μηχάνημα σε υπολογιστή με πρόσβαση στο δίκτυο αλλάξτε οπωσδήποτε τον κωδικό σε κάτι πιο δύσκολο, ή ακόμα καλύτερα δημιουργήστε άλλο χρήστη και δώστε του δικαιώματα χρήσης των προσομοιωτών.

Οι προσομοιωτές βρίσκονται στη διαδρομή "/usr/local/src/". Στον ίδιο κατάλογο υπάρχουν και τα συμπιεσμένα αρχεία κώδικα για το Omnet++ v3.3p1 και το Castalia v1.3. Το Shawn εγκαταστάθηκε από SVN αποθήκη. Επιπλέον σε αυτόν τον κατάλογο υπάρχει και το ένα bash script με όνομα vars.bash-3.3p1 το οποίο περιέχει εντολές για την προσθήκη μεταβλητών περιβάλλοντος με διαδρομές προς το εκτελέσιμο και βιβλιοθήκες του Omnet++. Επίσης ορίζει και τη διαδρομή προς την TCL βιβλιοθήκη. Για να εργαστούμε με το Omnet++ ή το Castalia, πρέπει πρώτα να εκτελέσουμε το αρχείο ώστε να ορισθούν οι μεταβλητές.

```
#!/bin/bash
export
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
     /bin:/usr/local/src/omnetpp-3.3p1/bin
export LD_LIBRARY_PATH=/usr/local/src/omnetpp-3.3p1/lib
export TCL_LIBRARY=/usr/share/tcltk/tcl8.4
```

Οι προσομοιωτές του Omnet++ βρίσκονται στη διαδρομή "/usr/local/src/omnetpp-3.3p1/mysims". Σε κάθε υποκατάλογο υπάρχει μια έκδοση προσομοιωτή από τις 5 που δημιουργήθηκαν μέχρι την τελική. Η πρώτη και δεύτερη έκδοση είναι τελείως διαφορετικές από τις υπόλοιπες τρεις. Οι τρεις τελευταίες έχουν μικροδιαφορές. Οι εκδόσεις 4 και 5 είναι οι πλήρως λειτουργικές με την 5η να έχει την υλοποίηση εκτύπωσης στατιστικών και καλύτερη οπτική διαμόρφωση στο γραφικό περιβάλλον.

Ο προσομοιωτή του Castalia βρίσκεται στη διαδρομή "/usr/local/src/Castalia-1.3/Simulations/sim1". Στον ίδιο κατάλογο βρίσκεται το omnetpp.ini, τα αρχεία συντεταγμένων και τα αρχεία αναφοράς και στατιστικών.

Το εκτελέσιμο του Shawn βρίσκεται στη διαδρομή "/usr/local/src/shawn/buildfiles". Στον ίδιο κατάλογο βρίσκονται και τα αρχεία ρυθμίσεων, τα αρχεία XML δεδομένων και οι αρχεία αναφορών και εκτυπώσεις.

Η προβολή και το γράψιμο κώδικα έγιναν με το πρόγραμμα "fte".

ΑΝΑΦΟΡΕΣ:

1. **“Free Software Foundation, Inc.”**, GPL http://www.gnu.org/copyleft/gpl.html (28/02/2010)

2. **“Debian Project Homepage”**, **http://www.debian.org/** (28/02/2010)

3. **“Oracle”**, VirtualBox WebSite **http://www.virtualbox.org/** (28/02/2010)

4. **“Sun Microsystems, Inc.”**, VirualBox UserManual, **http://www.virtualbox.org/manual/UserManual.html** (28/02/2010)