ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΑΤΡΑΣ
ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ
ΤΜΗΜΑ ΕΠΙΧΕΙΡΗΜΑΤΙΚΟΥ ΣΧΕΔΙΑΣΜΟΥ ΚΑΙ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ολοκληρωμένα Περιβάλλοντα Προσομοίωσης Δικτύων Υπολογιστών

Σεφερλής Δήμος-Αριστοτέλης
Κούκος Παναγιώτης
Σμυρναίος Φίλιπος

Εποπτεύων καθηγητής: Μάνδαλος Λουκάς

Πάτρα 2011

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

ΠΕΡΙΛΗΨΗ

Η υλοποίηση ενός μεγάλου δικτύου υπολογιστών είναι μια δύσκολη διαδικασία λόγω της διάταξης του περιβάλλοντος χώρου, της απόσταση των κόμβων και πάρα πολλών άλλων παραγόντων που καθορίζουν την ποιότητα της σύνδεσης και την επίτευξη επικοινωνίας σε ένα δίκτυο υπολογιστών. 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 η πιο κατάλληλη επιλογή είναι η προσομοίωση ενός δικτύου μέσω διάφορων πακέτων λογισμικών που ονομάζονται προσομοιωτές. 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.

Στην εργασία αυτή επιλέχθηκαν να χρησιμοποιηθούν τέσσερις προσομοιωτές, το OPNET Modeler, το NS-2, το JSIM και το OMNET++. Η χρήση τους είναι αρκετά διαδεδομένη στην κοινότητα ανάπτυξης εφαρμογών προσομοίωσης δικτύων, δίνοντας μια αίσθηση επιβεβαίωσης και ασφάλειας ότι θα εξυπηρετήσουν το σκοπό τους. Το OMNET++ θεωρήθηκε ο πιο κατάλληλος προσομοιωτής από τους τέσσερις για την εκτέλεση της προσομοίωσης (βλ. Κεφάλαιο 4) και στην εργασία αυτή παρουσιάζονται οι προσομοιώσεις που έγιναν με αυτό το πακέτο λογισμικού.

ΚΕΦΑΛΑΙΟ 1: Εισαγωγή

1.1 Αντικείμενο εργασίας

Η υλοποίηση ενός μεγάλου δικτύου υπολογιστών είναι μια δύσκολη διαδικασία. Η διάταξη του περιβάλλοντος χώρου, η απόσταση των κόμβων και πάρα πολλοί άλλοι παράγοντες καθορίζουν την ποιότητα της σύνδεσης και την επίτευξη επικοινωνίας σε ένα δίκτυο υπολογιστών. 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.

Η δεύτερη δυνατότητα που μας δίνεται για το σχεδιασμό ενός δικτύου υπολογιστών 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 υπολογιστικά φαινόμενα τα οποία αναπαριστώνται με μαθηματικά μοντέλα εφαρμόζουμε συνήθως προσομοιώσεις σε ηλεκτρονικό υπολογιστή ή κάποιο άλλο υπολογιστικό σύστημα.

Πάνω σε αυτό το θέμα έχουν υλοποιηθεί αρκετοί προσομοιωτές δικτύων. Για την μελέτη που ακολουθεί επιλέχθηκαν να χρησιμοποιηθούν τέσσερις προσομοιωτές, το OPNET Modeler, το NS-2, το JSIM και το OMNET++. Η χρήση τους είναι αρκετά διαδεδομένη στην κοινότητα ανάπτυξης εφαρμογών προσομοίωσης δικτύων, δίνοντας μια αίσθηση επιβεβαίωσης και ασφάλειας ότι θα εξυπηρετήσουν το σκοπό τους. Γίνεται μία σύγκριση μεταξύ των προσομοιωτών και παρουσιάζονται τα αποτελέσματα αυτής.

Το OMNET++ θεωρήθηκε ο πιο κατάλληλος προσομοιωτής από τους τέσσερις για την εκτέλεση της προσομοίωσης και στην εργασία αυτή παρουσιάζονται οι προσομοιώσεις που έγιναν με αυτό το πακέτο λογισμικού.

1.2 Διάρθρωση εργασίας

Στο κεφάλαιο 2 παρουσιάζονται μερικές θεμελιώδεις έννοιες των δικτύων υπολογιστών, καθώς και η ανάγκη για τη δημιουργία προσομοιώσεων. Επίσης, παρουσιάζονται μερικές γλώσσες και περιβάλλοντα προσομοίωσης. Στο 3^{ο} κεφάλαιο παρουσιάζονται τα πιο ευρέως διαδεδομένα πακέτα λογισμικού που χρησιμοποιούνται στη μοντελοποίηση και προσομοίωση των δικτύων υπολογιστών. Πιο συγκεκριμένα, τα πακέτα αυτά είναι οι προσομοιωτές OPNET Modeler, NS-2, JSIM και OMNET++. Επίσης, επιχειρείται μία σύγκριση μεταξύ αυτών των πακέτων.

Στο 4^{ο} κεφάλαιο παρουσιάζεται μία σειρά από προσομοιώσεις με το πακέτο λογισμικού OMNET++. Οι προσομοιώσεις που έγιναν αφορούν τόσο μικρά όσο και μεγάλα δίκτυα με την εξαγωγή διάφορων γραφημάτων. Τέλος, στο 5^{ο} κεφάλαιο παρουσιάζονται τα συμπεράσματα της εργασίας.

1.3 Ευχαριστίες

Θα θέλαμε να ευχαριστήσουμε τον καθηγητή μας, κ. Μάνδαλο για την πολύτιμη βοήθεια και καθοδήγηση που μας παρείχε καθώς και για την υπομονή και κατανόηση που έδειξε. Επίσης ένα μεγάλο ευχαριστώ στον φίλο μας Στάθη ή αλλιώς ΄΄Wozniak΄΄ ο οποίος είναι πάντα πρόθυμος να δώσει λύση σε θέματα programming. Τέλος να ευχαριστήσουμε τις οικογένειες μας και ελπίζουμε να ανταποδώσουμε την εμπιστοσύνη που μας έδειξαν όλο αυτό τον καιρό.

ΚΕΦΑΛΑΙΟ 2: Εισαγωγή στα Δίκτυα Υπολογιστών και ανάγκη για προσομοίωσή τους

2.1 Εισαγωγή στα Δίκτυα Υπολογιστών

2.1.1 Ιστορική αναδρομή του Internet

Το σημερινό Internet αποτελεί εξέλιξη του ARPANET, ενός δικτύου που άρχισε να αναπτύσσεται πειραματικά στα τέλη της δεκαετίας του 60 στις ΗΠΑ.

- **Δεκαετία '60: ένα ενδιαφέρον πείραμα ξεκινά**

Στα πανεπιστήμια των ΗΠΑ οι ερευνητές ξεκινούν να πειραματίζονται με τη διασύνδεση απομακρυσμένων υπολογιστών μεταξύ τους. Το δίκτυο ARPANET γεννιέται το 1969 με πόρους του προγράμματος ARPA (Advanced Research Project Agency) του Υπουργείου Άμυνας, με σκοπό να συνδέσει το Υπουργείο με στρατιωτικούς ερευνητικούς οργανισμούς και να αποτελέσει ένα πείραμα για τη μελέτη της αξιόπιστης λειτουργίας των δικτύων. Στην αρχική του μορφή, το πρόγραμμα απέβλεπε στον πειραματισμό με μια νέα τεχνολογία γνωστή σαν μεταγωγή πακέτων (packet switching), σύμφωνα με την οποία τα προς μετάδοση δεδομένα κόβονται σε πακέτα και πολλοί χρήστες μπορούν να μοιραστούν την ίδια επικοινωνιακή γραμμή.

Στόχος ήταν η δημιουργία ενός διαδικτύου που θα εξασφάλιζε την επικοινωνία μεταξύ απομακρυσμένων δικτύων, έστω και αν κάποια από τα ενδιάμεσα συστήματα βρίσκονταν προσωρινά εκτός λειτουργίας. Κάθε πακέτο θα είχε την πληροφορία που χρειάζονταν για να φτάσει στον προορισμό του, όπου και θα γινόταν η επανασύνθεσή του σε δεδομένα τα οποία μπορούσε να χρησιμοποιήσει ο τελικός χρήστης. Το παραπάνω σύστημα θα επέτρεπε σε υπολογιστές να μοιράζονται δεδομένα και σε ερευνητές να υλοποιήσουν το ηλεκτρονικό ταχυδρομείο.

- **Δεκαετία '70: οι πρώτες συνδέσεις**

Το 1973 ξεκινά ένα νέο ερευνητικό πρόγραμμα που ονομάζεται Internetting Project (Πρόγραμμα Διαδικτύωσης) προκειμένου να ξεπεραστούν οι διαφορετικοί τρόποι που χρησιμοποιεί κάθε δίκτυο για να διακινεί τα δεδομένα του. Στόχος είναι η διασύνδεση πιθανώς ανόμοιων δικτύων και η ομοιόμορφη διακίνηση δεδομένων από το ένα δίκτυο στο άλλο. Από την έρευνα γεννιέται μια νέα τεχνική, το Internet Protocol (IP)

(Πρωτόκολλο Διαδικτύωσης), από την οποία θα πάρει αργότερα το όνομά του το Internet. Διαφορετικά δίκτυα που χρησιμοποιούν το κοινό πρωτόκολλο IP μπορούν να συνδέονται και να αποτελούν ένα διαδίκτυο.

Σε ένα δίκτυο IP όλοι οι υπολογιστές είναι ισοδύναμοι, οπότε τελικά όλοι οι υπολογιστές του διαδικτύου μπορούν να επικοινωνούν μεταξύ τους. Επίσης, σχεδιάζεται μια άλλη τεχνική για τον έλεγχο της μετάδοσης των δεδομένων, το Transmission Control Protocol (TCP) (Πρωτόκολλο Ελέγχου Μετάδοσης). Ορίζονται προδιαγραφές για τη μεταφορά αρχείων μεταξύ υπολογιστών (FTP) και για το ηλεκτρονικό ταχυδρομείο (E-mail). Σταδιακά συνδέονται με το ARPANET ιδρύματα από άλλες χώρες, με πρώτα το University College of London (Αγγλία) και το Royal Radar Establishment (Νορβηγία).

- **Δεκαετία '80: ένα παγκόσμιο δίκτυο για την ακαδημαϊκή κοινότητα**

Το 1983, το πρωτόκολλο TCP/IP (δηλ. ο συνδυασμός των TCP και IP) (βλ. 2.1.4) αναγνωρίζεται ως πρότυπο από το Υπουργείο Άμυνας των ΗΠΑ. Η έκδοση του λειτουργικού συστήματος Berkeley UNIX το οποίο περιλαμβάνει το TCP/IP συντελεί στη γρήγορη εξάπλωση της διαδικτύωσης των υπολογιστών. Εκατοντάδες πανεπιστήμια συνδέουν τους υπολογιστές τους στο ARPANET, το οποίο επιβαρύνεται πολύ και το 1983, χωρίζεται σε δύο τμήματα: στο MILNET (για στρατιωτικές επικοινωνίες) και στο νέο ARPANET (για χρήση αποκλειστικά από την πανεπιστημιακή κοινότητα και συνέχιση της έρευνας στη δικτύωση).

Το 1985, το National Science Foundation (NSF) δημιουργεί ένα δικό του γρήγορο δίκτυο, το NSFNET χρησιμοποιώντας το πρωτόκολλο TCP/IP, προκειμένου να συνδέσει πέντε κέντρα υπερ-υπολογιστών μεταξύ τους καθώς και με την υπόλοιπη επιστημονική κοινότητα. Στα τέλη της δεκαετίας του '80, όλο και περισσότερες χώρες συνδέονται στο NSFNET (Καναδάς, Γαλλία, Σουηδία, Αυστραλία, Γερμανία, Ιταλία, κ.α.). Χιλιάδες πανεπιστήμια και οργανισμοί δημιουργούν τα δικά τους δίκτυα και τα συνδέουν πάνω στο παγκόσμιο αυτό δίκτυο το οποίο αρχίζει να γίνεται γνωστό σαν INTERNET και να εξαπλώνεται με τρομερούς ρυθμούς σε ολόκληρο τον κόσμο. Το 1990, το ARPANET πλέον καταργείται.

- **Δεκαετία '90: ένα παγκόσμιο δίκτυο για όλους**

Όλο και περισσότερες χώρες συνδέονται στο NSFNET, μεταξύ των οποίων και η Ελλάδα το 1990. Το 1993, το εργαστήριο CERN στην Ελβετία παρουσιάζει το World Wide Web (WWW) (Παγκόσμιο Ιστό) που αναπτύχθηκε από τον Tim Berners-Lee. Πρόκειται για ένα σύστημα διασύνδεσης πληροφοριών σε μορφή πολυμέσων(multimedia), που βρίσκονται αποθηκευμένες σε χιλιάδες υπολογιστές συνδεδεμένους στο

Internet σε ολόκληρο τον κόσμο και παρουσίασής τους σε ηλεκτρονικές σελίδες, στις οποίες μπορεί να περιηγηθεί κανείς χρησιμοποιώντας ένα φυλλομετρητή (browser).

Το γραφικό αυτό περιβάλλον συνέβαλλε στην εξερεύνηση του Internet από τον απλό χρήστη. Παράλληλα, εμφανίζονται στο Internet διάφορα εμπορικά δίκτυα που ανήκουν σε εταιρίες παροχής υπηρεσιών Internet (Internet Service Providers - ISP) και προσφέρουν πρόσβαση στο Internet για όλους. Οποιοσδήποτε διαθέτει Η/Υ και modem μπορεί να συνδεθεί με το Internet σε τιμές που μειώνονται διαρκώς. Το 1995, το NSFNET καταργείται πλέον επίσημα και το φορτίο του μεταφέρεται σε εμπορικά δίκτυα.

Η ανακάλυψη του WWW σε συνδυασμό με την ευκολία απόκτησης πρόσβασης στο Internet προσέλκυσε έναν μεγάλο αριθμό καινούργιων χρηστών και έφερε την «έκρηξη» που παρακολουθήσαμε τα τελευταία χρόνια.
Σήμερα το μεγαλύτερο μέρος του πληθυσμού της Γης ζει σε χώρες που είναι συνδεδεμένες στο Internet. Παρατηρούμε ότι καθημερινά περιοδικά και εφημερίδες εκδίδονται «on-line» και μας παραπέμπουν στις διευθύνσεις τους, επιχειρήσεις και ιδιώτες φτιάχνουν τις δικές τους σελίδες στο WWW, κλπ. Είναι προφανές ότι το Internet δεν αποτελεί πλέον ένα δίκτυο των φοιτητών και των ερευνητών, αλλά ότι επεκτείνεται και επιδρά στις καθημερινές πρακτικές όλων μας. Ήδη μιλάμε για ηλεκτρονικό εμπόριο, τηλεργασία, τηλεκπαίδευση, τηλεϊατρική, κλπ. μέσα από το Internet.

2.1.2 Το μοντέλο αναφοράς OSI

Το μοντέλο αναφοράς OSI (Open System Interconnection – Διασύνδεση Ανοιχτών Συστημάτων) αναπτύχθηκε από τον Διεθνή Οργανισμό Τυποποίησης (ISO – International Standards Organization) και ασχολείται με συνδέσεις ανοιχτών συστημάτων (αυτά τα οποία είναι ανοιχτά για επικοινωνία με άλλα συστήματα).

Το μοντέλο OSI έχει 7 επίπεδα τα οποία φαίνονται στο παρακάτω σχήμα:

Πίνακας 1: Το Μοντέλο Αναφοράς OSI

7. Επίπεδο Εφαρμογής (Application Layer)
6. Επίπεδο Παρουσίασης (Presentation Layer)
5. Επίπεδο Συνόδου (Session Layer)
4. Επίπεδο Μεταφοράς (Transport Layer)
3. Επίπεδο Δικτύου (Network Layer)
2. Επίπεδο Σύνδεσης Δεδομένων (Data Link Layer)
1. Φυσικό Επίπεδο (Physical Layer)

1. Το φυσικό επίπεδο (physical layer) ασχολείται με τη μετάδοση ακατέργαστων bits σε ένα κανάλι επικοινωνίας.
2. Η κύρια αποστολή του επιπέδου σύνδεσης δεδομένων (data link layer) είναι να μετασχηματίσει το ακατέργαστο μέσο μετάδοσης σε μια γραμμή που εμφανίζεται ελεύθερη από σφάλματα μετάδοσης στο επίπεδο δικτύου. Μερικές από τις βασικές λειτουργίες αυτού του επιπέδου είναι η επιβεβαίωση μετάδοσης και λήψης καθώς και η ανίχνευση λαθών.
3. Το επίπεδο δικτύου (network layer) ασχολείται με τον έλεγχο της λειτουργίας του υποδικτύου. Παρέχει σύνδεση και δρομολόγηση (routing) ανάμεσα σε δύο κόμβους ενός δικτύου.
4. Η βασική λειτουργία του επιπέδου μεταφοράς (transport layer) είναι η αποδοχή δεδομένων από το επίπεδο δικτύου, η διάσπαση αυτών σε μικρότερες μονάδες αν χρειαστεί, η μεταφορά τους στο επίπεδο συνόδου και η διασφάλιση ότι όλα τα τμήματα φτάνουν σωστά στην άλλη πλευρά.
5. Το επίπεδο συνόδου (session layer) επιτρέπει στους χρήστες διαφορετικών μηχανημάτων να εγκαθιστούν συνόδους (sessions) μεταξύ τους. Μία σύνοδος επιτρέπει μια συνήθη μεταφορά δεδομένων, όπως και το επίπεδο μεταφοράς, αλλά παρέχει και μερικές πρόσθετες υπηρεσίες που είναι χρήσιμες σε πολλές εφαρμογές. Μια σύνοδος, μπορεί να χρησιμοποιηθεί για να επιτρέψει τη σύνδεση ενός χρήστη σ’ ένα απομακρυσμένο σύστημα καταμερισμού χρόνου (time-sharing) ή να μεταφέρει ένα αρχείο μεταξύ δύο μηχανών.
6. Το επίπεδο παρουσίασης (presentation layer) εκτελεί συγκεκριμένες λειτουργίες οι οποίες ζητούνται αρκετά συχνά από τους χρήστες, για να εξασφαλίζουν την εύρεση μιας γενικής λύσης γι’ αυτούς, ώστε να μην αφήνεται κάθε χρήστης να λύνει τα προβλήματα μόνος του. Συγκεκριμένα, ενώ όλα τα κατώτερα επίπεδα ενδιαφέρονται μόνο για την αξιόπιστη μεταφορά bits από το ένα μέρος στο άλλο, το επίπεδο παρουσίασης ενδιαφέρεται για το συντακτικό και τη σημασιολογία των πληροφοριών που μεταδίδονται.

7. Το επίπεδο εφαρμογής (application layer) χρησιμοποιεί τις υπηρεσίες του επιπέδου παρουσίασης για την εκτέλεση εφαρμογών των χρηστών. Μερικές χαρακτηριστικές λειτουργίες αυτού του επιπέδου είναι η μεταφορά αρχείων, η εισαγωγή εργασιών από απόσταση, η εμφάνιση καταλόγων (directory) αρχείων, το ηλεκτρονικό ταχυδρομείο κλπ. Σήμερα λίγοι είναι οι υπολογιστές και τα δίκτυα που είναι τελείως συμβατά με όλα τα επίπεδα του μοντέλου αναφοράς OSI.

2.1.3 Εισαγωγή στα δίκτυα

Για να καταλάβει κανείς τον τρόπο με τον οποίο σχεδιάζονται και λειτουργούν τα δίκτυα Η/Υ, θα πρέπει να γνωρίζει τα βασικά χαρακτηριστικά τους. Πολλά από αυτά μπορεί να λειτουργούν αυτόνομα ή σε συνδυασμό με άλλα, προκειμένου να οριοθετήσουν κάποια μορφή ταξινόμησης. Οι σημαντικότερες ταξινομήσεις δικτύων είναι οι ακόλουθες:

- Ταξινόμηση ως προς το μέσο μετάδοσης.
- Ταξινόμησης ως προς το είδος του συνδέσμου.
- Ταξινόμηση ως προς τη γεωγραφική κάλυψη.
- Ταξινόμηση ως προς το είδος της τοπολογίας.

Ταξινόμηση ως προς το μέσο μετάδοσης

Στην απλούστερη μορφή σύνδεσης, οι υπολογιστές συνδέονται απευθείας με κάποιο φυσικό μέσο ή σύνδεσμο, όπως είναι για παράδειγμα, ένα καλώδιο χαλκού, μια οπτική ίνα ή κάποια ασύρματη ζεύξη. Δύο είναι οι κυριότερες μορφές συνδέσμων:

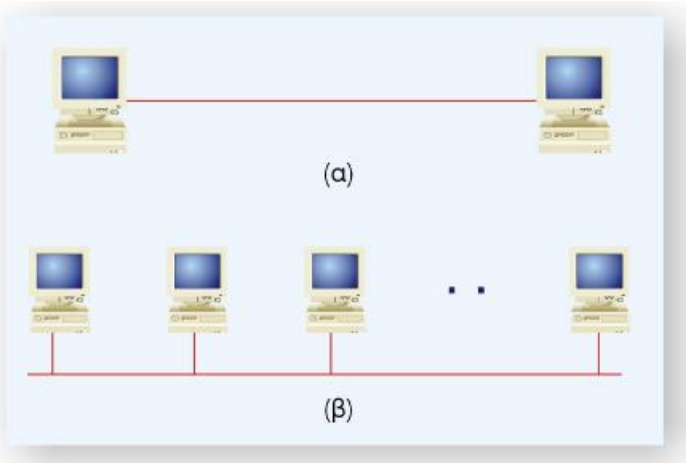
- Καλωδιακή ή ενσύρματη επικοινωνία που περιλαμβάνει όλων των ειδών τις εναέριες, επίγειες ή υπόγειες συνδέσεις αυτού του είδους. Παραδείγματα τέτοιων δικτύων αποτελούν όλα τα χάλκινα καλωδιακά δίκτυα, όπως επίσης και τα δίκτυα οπτικών ινών.
- Ασύρματη επικοινωνία στην οποία το μέσο μετάδοσης είναι η γήινη ατμόσφαιρα ή το διάστημα. Στα δίκτυα αυτά η πληροφορία μεταφέρεται μέσω ηλεκτρομαγνητικών κυμάτων με συχνότητα που εξαρτάται κάθε φορά από το ρυθμό μετάδοσης που επιδιώκεται κάθε φορά να έχει το δίκτυο. Παραδείγματα τέτοιων

δικτύων αποτελούν τα δίκτυα μικροκυματικών ζεύξεων, τα δίκτυα ραδιοεπικοινωνιών, τα δορυφορικά δίκτυα κ.τ.λ.

Ταξινόμηση ως προς το είδος σύνδεσης

Οι συνδέσεις διακρίνονται στις ακόλουθες δύο κατηγορίες:

- Σύνδεση σημείου με σημείο (point-to-point connection), η οποία συνδέει δύο μόνο κόμβους κάθε φορά. Αποτέλεσμα αυτής της απευθείας σύνδεσης είναι η επικοινωνία δύο κόμβων που συνδέονται δια μέσου άλλων κόμβων να γίνεται τμηματικά
- Σύνδεση ανοικτής ακρόασης ή ευρείας εκπομπής (broadcasting), η οποία συνδέει δύο ή και περισσότερους κόμβους ταυτόχρονα. Αποτέλεσμα αυτής της σύνδεσης είναι κάθε μήνυμα που αποστέλλεται από έναν κόμβο να παραλαμβάνεται από όλους ανεξαιρέτως τους κόμβους που βρίσκονται πάνω στο δίκτυο. Για το λόγο αυτό η σύνδεση αυτή λέγεται και σημείου με πολλαπλά σημεία (point – to multipoint connection).

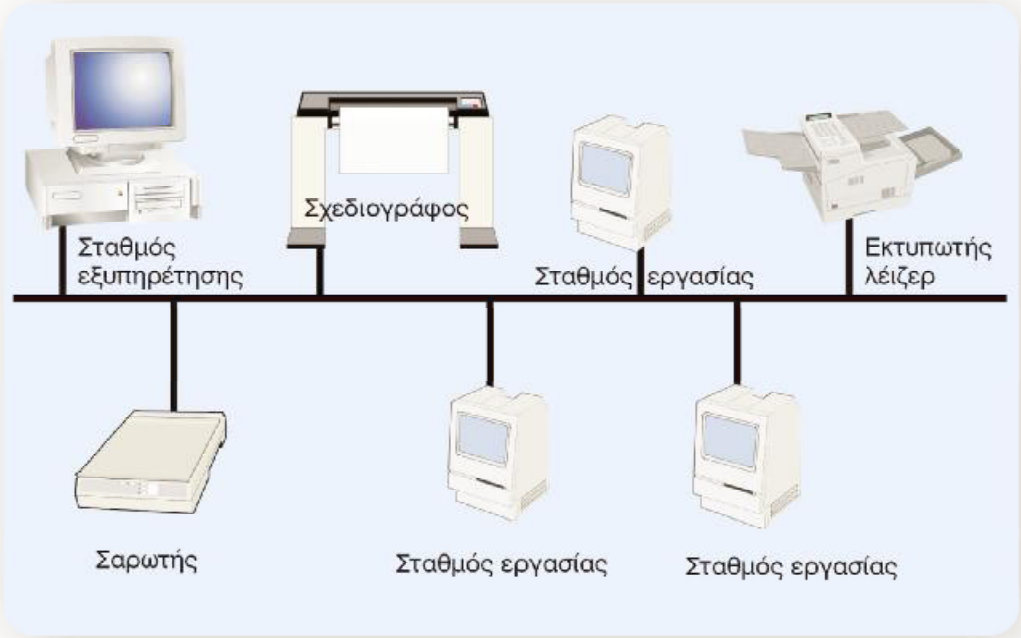


Εικόνα 1: Σύνδεση (α) σημείου με σημείο και (β) σημείου με πολλαπλά σημεία ή ανοικτής ακρόασης

Ταξινόμηση ως προς τη γεωγραφική κάλυψη

Ένας άλλος τρόπος χαρακτηρισμού των δικτύων Η / Υ είναι αυτός που έχει σχέση με την έκταση που καταλαμβάνουν. Έως τώρα, τρεις είναι οι κυρίαρχες μορφές δικτύων αυτής της ταξινόμησης, οι οποίες όμως έχουν ασαφή γεωγραφικά όρια. Οι κατηγορίες αυτές είναι οι ακόλουθες:

- Τοπικά δίκτυα (LAN: Local Area Network), είναι δίκτυα που συνδέουν υπολογιστές σε κοντινές αποστάσεις, π.χ. από υπολογιστές που βρίσκονται σε ένα δωμάτιο μέχρι υπολογιστές που απέχουν μερικά χιλιόμετρα μεταξύ τους. Χρησιμοποιούνται συνήθως για να συνδέουν προσωπικούς υπολογιστές και σταθμούς εργασίας σε γραφεία εταιρειών, εργοστάσια, πανεπιστήμια κ.λπ. Ενσύρματα δίκτυα αυτής της κατηγορίας συνήθως δεν επεκτείνονται πέραν των 100 km.



Εικόνα 2: Τοπικό δίκτυο

- Ένα μητροπολιτικό δίκτυο ή και MAN (metropolitan area network) είναι μια μεγαλύτερη εκδοχή ενός τοπικού δικτύου καθώς καλύπτει μεγαλύτερες αποστάσεις, π.χ. από μια ομάδα γειτονικών γραφείων μιας εταιρείας έως μια πόλη. Τα ενσύρματα δίκτυα αυτής της κατηγορίας συνήθως δεν υπερβαίνουν τα 200 km.
- Τα δίκτυα ευρείας περιοχής ή WAN (wide area network) καλύπτουν μεγάλες γεωγραφικές περιοχές, π.χ. από σύνδεση μεταξύ διαφορετικών πόλεων μέχρι μιας ολόκληρης ηπείρου και μπορούν να συνδέσουν ακόμη και περισσότερα από ένα τοπικά δίκτυα καθώς και ομάδες τοπικών δικτύων. Τα περισσότερα δίκτυα ευρείας περιοχής χρησιμοποιούν τηλεφωνικά δίκτυα ή

τηλεπικοινωνιακούς δορυφόρους. Η γεωγραφική τους έκταση ξεπερνάει τα 200 km.

Εικόνα 3: Δίκτυο ευρείας περιοχής

- Τα διαδίκτυα είναι δίκτυα ευρείας περιοχής τα οποία καλύπτουν γεωγραφικές περιοχές μίας ή περισσοτέρων ηπείρων διασυνδέοντας επιμέρους δίκτυα. Σε ένα διαδίκτυο μπορεί να συνυπάρχουν διασυνδεδεμένοι υπολογιστές και δίκτυα που χρησιμοποιούν διαφορετικές τεχνολογίες και λειτουργικά συστήματα. Το Διαδίκτυο (Internet) είναι το μεγαλύτερο τέτοιου είδους δίκτυο.

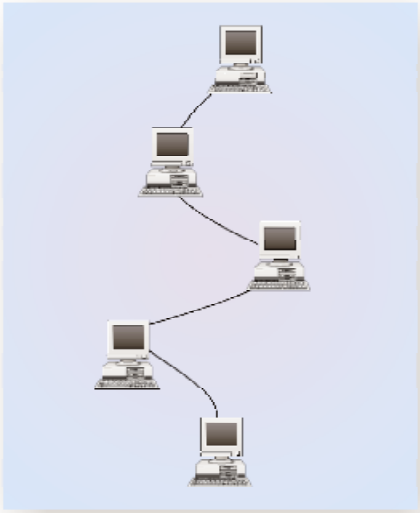
Ταξινόμηση ως προς το είδος της τεχνολογίας

Τοπολογία δικτύου ονομάζεται η μορφή της σύνδεσης μεταξύ των κόμβων ενός δικτύου. Οι τοπολογίες είναι είτε φυσικές είτε λογικές. Τα κυριότερα είδη τοπολογιών είναι:

- τύπου διαύλου,
- τύπου δακτυλίου,

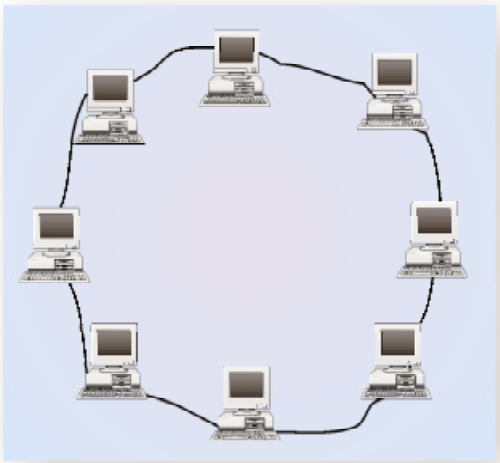
- τύπου αστέρα,
- η κατανεμημένη,
- η πλήρως κατανεμημένη και
- τύπου δένδρου.

Ø Στην τοπολογία διαύλου (bus) όλες οι συσκευές συνδέονται με ένα κεντρικό καλώδιο, το οποίο αποκαλείται bus ή σπονδυλική στήλη. Τα δίκτυα διαύλου είναι σχετικά ανέξοδα και εύκολο να εγκατασταθούν για τα μικρά δίκτυα. Τα συστήματα Ethernet χρησιμοποιούν μια τοπολογία bus.



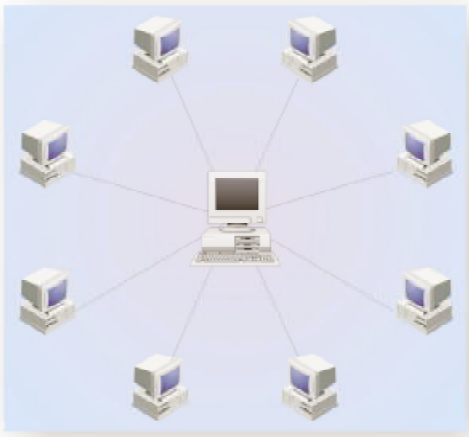
Εικόνα 4: Δίκτυο τοπολογίας διαύλου

Ø Στην τοπολογία δακτυλίου (ring) όλες οι συσκευές συνδέονται η μία με την άλλη με τη μορφή ενός κλειστού βρόχου, έτσι ώστε κάθε συσκευή να συνδέεται άμεσα με δύο άλλες συσκευές, μία από κάθε πλευρά . Οι τοπολογίες δακτυλίων είναι σχετικά ακριβές και δύσκολο να εγκατασταθούν, αλλά προσφέρουν το υψηλό εύρος ζώνης και μπορούν να εκταθούν σε μεγάλες αποστάσεις. Παραδείγματα τέτοιων τοπολογιών αποτελούν το token ring και το FDDI.



Εικόνα 5: Δίκτυο τοπολογίας δακτυλίου

- Στην τοπολογία αστέρα (star) όλες οι συσκευές συνδέονται με μια κεντρική πλήμνη (hub). Τα δίκτυα αστεριών είναι σχετικά εύκολο να εγκατασταθούν και να διαχειριστούν, αλλά οι δυσχέρειες μπορούν να εμφανιστούν επειδή όλα τα στοιχεία πρέπει να περάσουν μέσω του hub.



Εικόνα 6: Δίκτυο τοπολογίας αστέρα

- Η τοπολογία δένδρου είναι μια γενίκευση της τοπολογίας διαύλου. Το μέσο μετάδοσης είναι ένα διακλαδιζόμενο καλώδιο χωρίς κλειστούς βρόγχους. Το σχεδιάγραμμα της τοπολογίας δένδρου ξεκινάει από ένα σημείο γνωστό σαν "ρίζα" (headend). Ένα ή περισσότερα καλώδια ξεκινούν από αυτό το σημείο και καθένα από αυτά μπορεί να έχει διακλαδώσεις. Οι διακλαδώσεις αυτές μπορούν με τη σειρά τους να έχουν και άλλες διακλαδώσεις επιτρέποντας με αυτόν τον τρόπο πολύπλοκα σχεδιαγράμματα. Και εδώ η μετάδοση από κάποιο σταθμό διαδίδεται διαμέσου του

φυσικού μέσου και «ακούγεται» από όλους τους άλλους σταθμούς. Μια τοπολογία δέντρου συνδυάζει τα χαρακτηριστικά των γραμμικών τοπολογιών διαύλου και αστέρα. Αυτές οι τοπολογίες μπορούν επίσης να αναμιχθούν. Παραδείγματος χάριν, ένα δίκτυο διαύλου-αστέρα αποτελείται από ένα bus υψηλής-εύρους ζώνης, αποκαλούμενο σπονδυλική στήλη, η οποία συνδέει τις συλλογές των τμημάτων αστεριών αργές-εύρους ζώνης.

2.1.4 Η οικογένεια πρωτοκόλλων TCP/IP

Η οικογένεια πρωτοκόλλων TCP/IP (Transmission Control Protocol / Internet Protocol) ξεκίνησε στις αρχές του 1970 και χρησιμοποιήθηκε για τη διασύνδεση κεντρικών υπολογιστών (hosts) στο ARPANET, στο PRNET (packet radio) και στο SATNET (packet satellite). Αρχικά ο σχεδιασμός της έγινε λόγω του γεγονότος ότι τα τρία παραπάνω δίκτυα ήταν ετερογενή μεταξύ τους. Σήμερα, παρόλο που τα παραπάνω δίκτυα έχουν αποσυρθεί, τα TCP/IP πρωτόκολλα είναι τα πιο διαδεδομένα παγκοσμίως.

Τα διάφορα επίπεδα του TCP/IP σε σχέση με το μοντέλο αναφοράς OSI φαίνονται στον παρακάτω πίνακα:

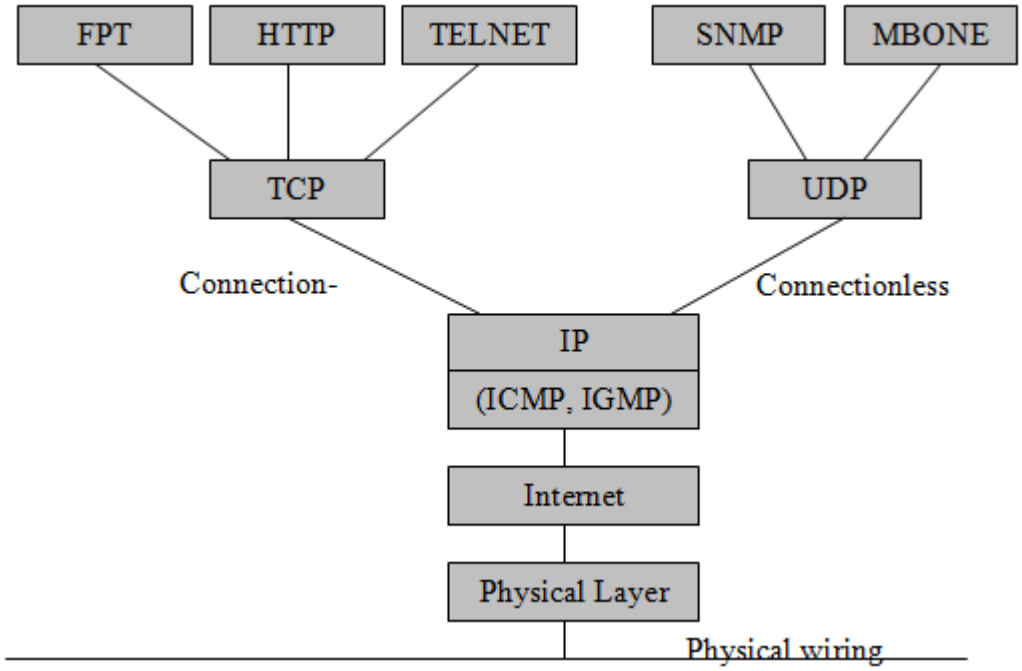
Πίνακας 2: Τα επίπεδα του OSI και του TCP/IP

OSI Layering		TCP Layering
7. Application Layer		Application or process layer
6. Presentation Layer		
5. Session Layer		
4. Transport Layer		Host-to-host transport layer
3. Network Layer		Internetwork (IP)
2. Data Link Layer		Network Interface
1. Physical Layer		Physical Layer

1. Το φυσικό επίπεδο (physical layer) αναλαμβάνει τη διαχείριση του φυσικού μέσου, όπως μία γραμμή Ethernet.
2. Το επίπεδο δικτύου (network layer) αναλαμβάνει τη διευθυνσιοδότηση (IP addressing) και το Domain Name Service (DNS).
3. Το Internetwork επίπεδο παρέχει τη βασική υπηρεσία μεταγωγής αυτοδύναμων πακέτων (datagrams) στον τελικό τους προορισμό.

4. Το επίπεδο μεταφοράς (host-to-host transport layer) παρέχει υπηρεσίες οι οποίες απαιτούνται από διάφορες εφαρμογές.
5. Το επίπεδο εφαρμογής (application or process layer) είναι ένα πρωτόκολλο εφαρμογών, όπως το ηλεκτρονικό ταχυδρομείο.

Η λογική δομή της οικογένειας πρωτοκόλλων TCP/IP φαίνεται στην παρακάτω εικόνα:



Εικόνα 7: ***Η λογική δομή της οικογένειας πρωτοκόλλων διαδικτύου***

2.2 Ανάγκη για προσομοίωση δικτύων

Τα δίκτυα συνεχίζουν να αναπτύσσονται όλο και πιο πολύπλοκα όσο η βιομηχανία παράγει τόσο ενσύρματες όσο και ασύρματες τεχνολογίες σε μεγάλης κλίμακας αρχιτεκτονικές δικτύων, αλλά και όσο οι εφαρμογές των χρηστών και η «κίνηση» σε αυτά συνεχίζουν να εξελίσσονται. Στις μέρες μας ελάχιστες εταιρίες μπορούν να ανεχτούν το κόστος για να χαραμίζουν επιπλέον megabits το δευτερόλεπτο σε ένα ερευνητικό έργο, όταν ο προϋπολογισμός της εταιρίας προστάζει για ακριβή, άρτια και οικονομικά σχέδια δικτύων από την αρχή.

Αντιμέτωποι με αυτή την τεράστια ανάπτυξη τα τελευταία χρόνια οι τεχνικοί και οι ερευνητές των δικτύων σχεδόν καθολικά χρησιμοποιούν την προσομοίωση με σκοπό να προβλέψουν την αναμενόμενη απόδοση πολύπλοκων

δικτύων και να κατανοήσουν τη συμπεριφορά των υπαρχόντων πρωτοκόλλων δικτύων που δεν έχουν στηριχτεί σε πρότυπα που λειτουργούν στα σημερινά δίκτυα. Η προσομοίωση χρησιμοποιείται επίσης διαρκώς όλο και πιο πολύ για να προβλέψει την ορθότητα και τις επιδόσεις των νέων σχεδίων πρωτοκόλλων για τα δίκτυα. Επιπλέον η χρησιμοποίηση της προσομοίωσης πλέον εμφανίζεται σαν μια αυστηρή απαίτηση στην διαδικασία που οδηγεί στην διεθνή πιστοποίηση.

Βέβαια αυτή η αναπτυσσόμενη εμπιστοσύνη στην προσομοίωση αυξάνει τις απαιτήσεις σε ότι αφορά την εξασφάλιση της ορθότητας και της πρόβλεψης των σφαλμάτων σε συγκεκριμένα μοντέλα προσομοίωσης. Παρόλα αυτά όμως δεν υπάρχουν ευρέως αποδεκτές πρακτικές και τεχνικές για να βοηθήσουν στην αξιολόγηση των προσομοιώσεων των δικτύων και στην αποτίμηση εμπιστοσύνης στα αποτελέσματά τους.

Οι πρώτες εργασίες που αφορούσαν την τεχνική σχεδίαση και έρευνα των δικτύων περιείχαν τόσο πειραματικά όσο και μαθηματικά μοντέλα για να αποδείξουν την επίτευξη και την εξασφάλιση των ορίων για την προβλεπόμενη απόδοση. Τα τελευταία δέκα χρόνια καθώς τα δίκτυα έγιναν τόσο πολύ μεγάλα ώστε να επιτρέπουν εύκολο πειραματισμό, αλλά και τόσο πολύπλοκα ώστε να επιτρέπουν μια αρκετά εύκολη μαθηματική ανάλυση, η προσομοίωση των δικτύων ήρθε να συμπληρώσει ένα πολύ σημαντικό ρόλο, βοηθώντας τους ερευνητές και τους σχεδιαστές να αντιληφθούν την συμπεριφορά και την απόδοση των πρωτοκόλλων και των δικτύων. Σήμερα η προσομοίωση συχνά χρησιμοποιείται (Λογοθέτης, 2001):

- για την απόδοση των τωρινών δικτύων και πρωτοκόλλων με σκοπό να βοηθήσει στον καθορισμό της τεχνολογίας, να προγραμματίσει τις παραγόμενες δυνατότητες και να επιδείξει την εκπλήρωση των απαιτήσεων του πελάτη
- για να προβλέψει την αναμενόμενη συμπεριφορά των νέων δικτυακών πρωτοκόλλων και σχεδίων μέσα από ποιοτικές και ποσοτικές εκτιμήσεις απόδοση ή ορθότητας
- για να εξερευνήσει γρήγορα τα όρια και τις δυνατότητες ενδεχόμενων σχεδίων πρωτοκόλλων μέσα από σύντομη αποτίμηση και επανάληψη

Λόγω αυτής της μεγάλης εξάπλωσης της προσομοίωσης των δικτύων έχει δημιουργηθεί ένας μεγάλος αριθμός εργαλείων με τα οποία μπορεί να γίνει η προσομοίωση σε διάφορες τοπολογίες ή σε διάφορα πρωτόκολλα και τα οποία μέσω των γραφικών που διαθέτουν να καταδείξουν τα αποτελέσματα και τις επιδόσεις που προκύπτουν από την λειτουργία ενός δικτύου κάτω από συγκεκριμένες συνθήκες και παραμέτρους. Τα εργαλεία αυτά που έχουν δημιουργηθεί αναφέρονται σε διάφορες πλατφόρμες και υποστηρίζονται από συγκεκριμένα λειτουργικά συστήματα. Πολλά αυτά λειτουργούν μόνο σε

περιβάλλοντα Linux ή Unix, κάποια άλλα σε περιβάλλοντα Windows, μερικά σε Solaris, ενώ λίγα σχετικά προσφέρονται και είναι συμβατά σε όλα τα λειτουργικά συστήματα.

Όλα σχεδόν τα εργαλεία προσομοίωσης δικτύων απαιτούν αρκετά μεγάλες προγραμματιστικές ικανότητες σε συνδυασμό με άριστη γνώση των δικτύων και των πρωτοκόλλων τους, καθώς για να προκύψουν σωστά, αλλά κυρίως χρήσιμα αποτελέσματα τα οποία θα χρησιμοποιηθούν για την αποφυγή ανεπιθύμητων σφαλμάτων, θα πρέπει να μπορεί ο προγραμματιστής να εκτιμήσει και να δώσει τις σωστές παραμέτρους που θα εξαντλούν όλες τις πιθανές περιπτώσεις χρήσης ενός δικτύου. Είναι άσκοπο να ξοδεύεις πολλές ώρες στην προσομοίωση μιας δικτυακής εφαρμογής αν στο τέλος δε μπορείς να βγάλεις ασφαλή και τεκμηριωμένα αποτελέσματα για την αξιοπιστία, τις επιδόσεις, την ασφάλεια και γενικά την άρτια λειτουργία της εφαρμογής. Όπως προαναφέρθηκε λοιπόν είναι απαραίτητο να δοθεί έμφαση στην αξιολόγηση των αποτελεσμάτων που προκύπτουν από την προσομοίωση.

2.3 Σκοποί της προσομοίωσης

Μερικές φορές, οι επακριβείς αναλυτικές μέθοδοι στην θεωρία τηλεπικοινωνιακής κινήσεως παρουσιάζουν αδυναμία να αναλύσουν ικανοποιητικά δίκτυα επικοινωνιών ευρείας κλίμακας. Ως εναλλακτική λύση χρησιμοποιείται η μέθοδος της προσομοίωσης στον υπολογιστή. Στον πίνακα που ακολουθεί γίνεται σύγκριση ανάμεσα σε τρεις μεθόδους επίλυσης προβλημάτων τηλεπικοινωνιακής κινήσεως.

Πίνακας 3: Λύσεις προβλημάτων τηλεπικοινωνιακής κίνησης

Λύση	**Εύρος εφαρμογών**	**Ακρίβεια αποτελέσματος**	**Υπολογιστικός χρόνος**	**Απαιτούμενη μνήμη**	**Μέγεθος προγράμματος**
Αναλυτική μέθοδος	Μικρό	Υψηλή	Μικρός	Μικρή	Μικρό
Αριθμητική ανάλυση	Μεσαίο	Μεσαία	Μεσαίος	Μεσαία	Μεσαίο

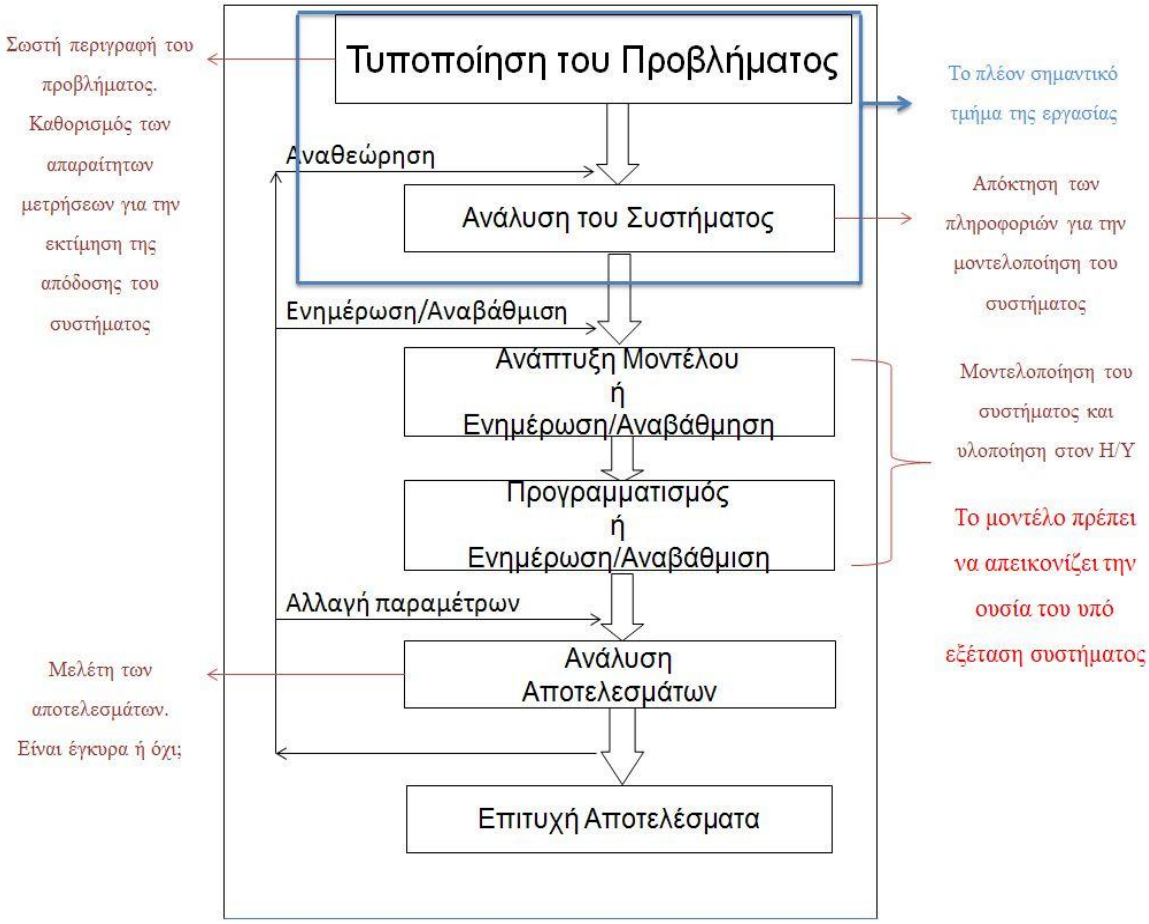
Προσομοίωση	Μεγάλο	Χαμηλή	Μεγάλος	Μεγάλη	Μεγάλο

Η προσομοίωση είναι χρήσιμη όχι μόνο στην περίπτωση που η αναλυτική επίλυση δεν είναι εφικτή, αλλά και όταν ο υπολογισμός της αναλυτικής λύσης με αριθμητικές μεθόδους είναι δύσκολος ή πρέπει να πιστοποιήσουμε την εγκυρότητα προσεγγιστικών λύσεων. Επίσης, η προσομοίωση χρησιμοποιείται σε διάφορα στάδια σχεδιασμού και λειτουργίας τηλεπικοινωνιακών συστημάτων όπως (Πομπόρτσης et. al., 2001):

1. Στην διασαφήνιση προβλημάτων τηλεπικοινωνιακής κινήσεως, ώστε εν συνεχεία να ανατροφοδοτηθεί ο σχεδιασμός του υπό μελέτη συστήματος
2. Στην εκτίμηση της καλής λειτουργίας ενός συστήματος
3. Στην εκτίμηση της απόκρισης του συστήματος σε σφάλματα και υπερφόρτωση, για τον καθορισμό των ενδεδειγμένων μέτρων

Στο θέμα (1.), είναι σημαντικότερο να έχουμε τα αποτελέσματα γρήγορα και έγκαιρα για τον σχεδιασμό του συστήματος, παρά να επιδιώξουμε μεγαλύτερη ακρίβεια. Έτσι, χρησιμοποιούνται συχνά προσομοιώσεις περιορισμένης κλίμακας, οι οποίες κατασκευάζονται γρήγορα και μπορούν να τροποποιηθούν εύκολα. Για τα θέματα (2.) και (3.), απαιτούνται πλήρεις προσομοιώσεις, ώστε, βρίσκοντας επακριβώς τα χαρακτηριστικά της κίνησης, να υπολογίσουμε τις διαστάσεις του συστήματος.

Η διαδικασία της προσομοίωσης μπορεί να αναπαρασταθεί από την παρακάτω εικόνα:



Εικόνα 8: Η διαδικασία της προσομοίωσης

Τα πλεονεκτήματα της προσομοίωσης μπορούν να συνοψιστούν στα παρακάτω (Πομπόρτσης et. al., 2001):

1) Συμβάλλει στην μελέτη νέων τρόπων λειτουργίας ενός συστήματος χωρίς να χρειάζεται να διακοπεί η λειτουργία του πραγματικού συστήματος.
2) Συμβάλλει στην μελέτη σύνθετων συστημάτων με στοχαστικά στοιχεία τα οποία δεν μπορούν να περιγραφούν με μαθηματικά μοντέλα.
3) Συμβάλλει στην κατανόηση της σχέσης μεταξύ των μεταβλητών ενός μοντέλου και στο πως οι μεταβλητές αυτές επηρεάζουν την απόδοση του συστήματος.
4) Συμβάλλει στην σύγκριση εναλλακτικών τρόπων λειτουργίας ή σχεδιασμού του συστήματος.

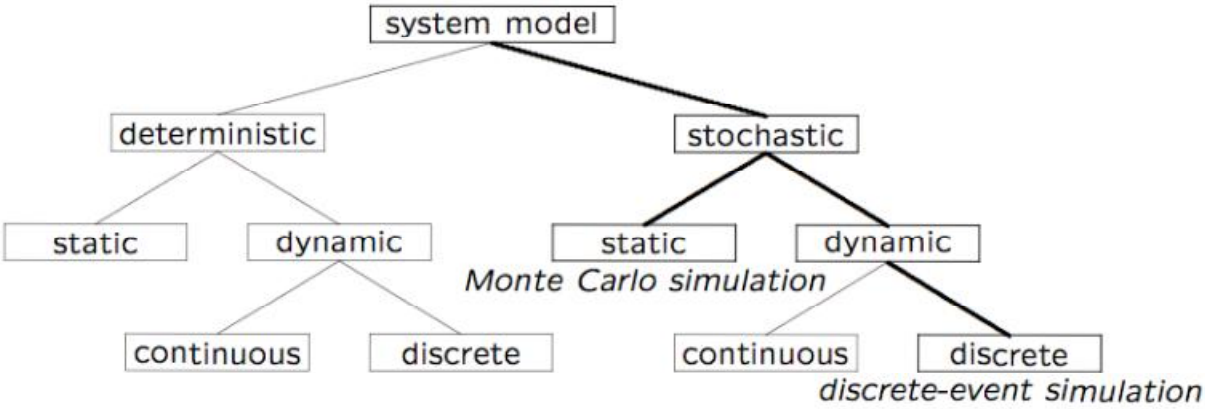
Τα μειονεκτήματα της προσομοίωσης μπορούν να συνοψιστούν στα παρακάτω (Πομπόρτσης et. al., 2001):

1) Η κατασκευή ενός μοντέλου προσομοίωσης μπορεί να γίνει μια δύσκολη διαδικασία που απαιτεί ειδικές γνώσεις και εξάσκηση.

2) Σε κάθε τρέξιμο (run) ενός στοχαστικού μοντέλου προσομοίωσης παράγονται αποτελέσματα για ένα συγκεκριμένο σύνολο παραμέτρων εισόδου. Απαιτείται λοιπόν να μελετηθούν πολλά ανεξάρτητα τρεξίματα του μοντέλου για κάθε σύνολο παραμέτρων εισόδου.

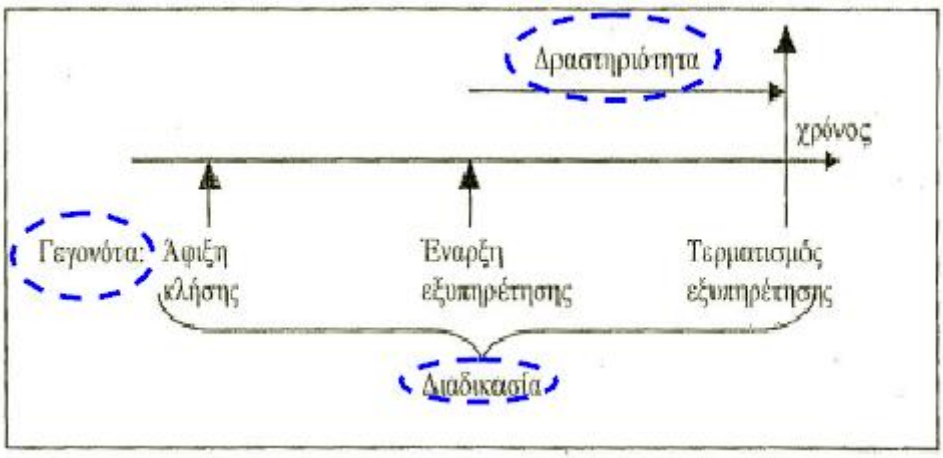
2.4 Μέθοδοι προσομοίωσης

Στην παρακάτω εικόνα αναπαρίστανται όλοι οι μέθοδοι προσομοίωσης:



Εικόνα 9: Μέθοδοι προσομοίωσης

Η διακριτή προσομοίωση λαμβάνει χώρα όταν η προσομοίωση της τηλεπικοινωνιακής κινήσεως ταξινομείται στην προσομοίωση διακριτών γεγονότων. Αυτό σημαίνει πως η διακριτή κατάσταση (πλήθος κλήσεων) του συστήματος μεταβάλλεται από μια έναρξη ή τερματισμό μιας κλήσης.



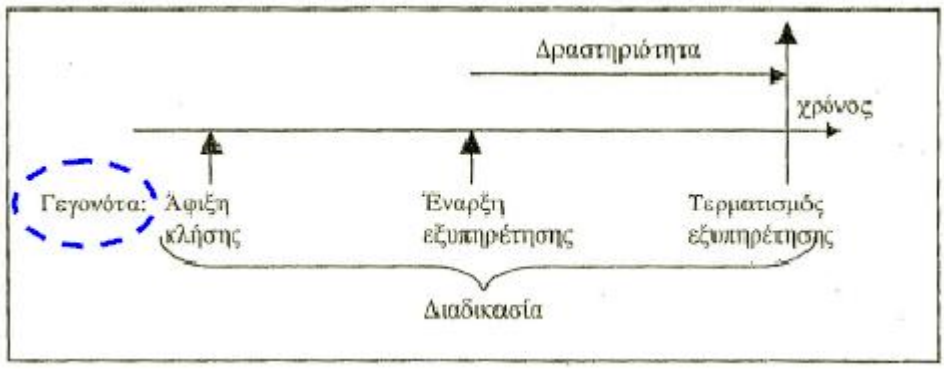
Εικόνα 10: Προσομοίωση βάσει διακριτών γεγονότων

Υπάρχουν τρεις παράμετροι βάσει των οποίων κατασκευάζουμε ένα μοντέλο προσομοίωσης διακριτών γεγονότων:

- το γεγονός (event),
- τη διαδικασία (process) και
- τη δραστηριότητα (activity).

Ø Μοντελοποίηση βάσει γεγονότων

Με την μέθοδο της μοντελοποίησης γεγονότων (event-oriented modelling), ο προγραμματιστής ορίζει τα γεγονότα (π.χ. η έναρξη και ο τερματισμός μιας κλήσης) και γράφει ρουτίνες οι οποίες «καλούνται» κάθε φορά που εμφανίζεται ένα γεγονός (event). Προτιμούμε αυτή την μέθοδο όταν χρησιμοποιούμε γλώσσες προγραμματισμού γενικού σκοπού όπως η FORTRAN, C, C++, Java.



Εικόνα 11: Μοντελοποίηση βάσει γεγονότων

Ø Μοντελοποίηση βάσει διαδικασιών

Με αυτή την μέθοδο προσπαθούμε να περιγράψουμε την συμπεριφορά των οντοτήτων (entities) του συστήματος. Γι' αυτό και ονομάζεται μοντελοποίηση οντοτήτων (entity-oriented modelling).

Οι οντότητες μπορεί να είναι προσωρινές (π.χ. κλήσεις ή μηνύματα που φτάνουν στο σύστημα με βάση κάποια στοχαστική διαδικασία) ή μόνιμες (π.χ. servers, routers που εξυπηρετούν τις προσωρινές οντότητες).

Η συμπεριφορά των οντοτήτων περιγράφεται μέσω (υπό)ρουτινών.

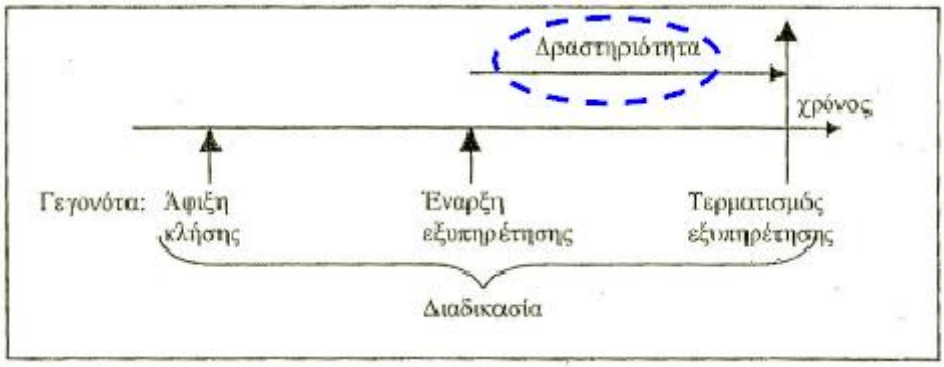
Η μέθοδος αυτή εφαρμόζεται στις περιπτώσεις που χρησιμοποιούνται γλώσσες προσομοίωσης, όπως General Purpose Simulation System (GPSS) ή Simulation Language for Alternative Modelling (SLAM). (βλ. 2.5)

Εικόνα 12: Μοντελοποίηση βάσει διαδικασιών

Ø Μοντελοποίηση βάσει δραστηριοτήτων

Με την μέθοδο της μοντελοποίησης δραστηριοτήτων (activity-oriented modeling) προσπαθούμε να περιγράψουμε τις χρονικές στιγμές της αρχής και του τερματισμού των δραστηριοτήτων, όπως είναι η διάρκεια κλήσης, η αρχή και το τέλος της κατάληψης μιας ζεύξης.

Αυτή η μέθοδος ταιριάζει στην μοντελοποίηση συστημάτων στα οποία ο χρόνος αναμονής εξαρτάται από την κατάσταση του συστήματος (πλήθος υπαρχουσών κλήσεων). Ωστόσο, επειδή η προσομοίωση επιτελείται ανιχνεύοντας τις δραστηριότητες του συστήματος, ο χρόνος εκτέλεσης είναι μεγαλύτερος απ' ότι στη μοντελοποίηση γεγονότων.



Εικόνα 13: Μοντελοποίηση βάσει δραστηριοτήτων

Ας θεωρήσουμε την προσομοίωση ενός συστήματος αναμονής με μοντελοποίηση βάσει δραστηριοτήτων.

Σ' αυτή την περίπτωση θα χωρίζαμε τον χρόνο της προσομοίωσης σε μικρά διαστήματα. Αν για παράδειγμα η μέση τιμή του χρόνου μεταξύ δύο διαδοχικών αφίξεων (mean interarrival time) ήταν 20 δευτερόλεπτα θα μπορούσαμε να χωρίσουμε τον χρόνο σε διαστήματα των 0.001 δευτερολέπτων. Σε κάθε τέτοιο μικρό διάστημα ο κώδικας της προσομοίωσης θα έψαχνε για όλες τις δραστηριότητες (π.χ. κλήσεις που εξυπηρετούνται, τερματισμός μιας κλήσης) που θα μπορούσαν να ξεκινήσουν άμεσα.

Μειονέκτημα της μοντελοποίησης είναι ο μεγάλος χρόνος εκτέλεσης καθώς και το γεγονός ότι σε πολλά διαστήματα (π.χ. των 0.001 δευτερολέπτων) το σύστημα δεν θα παρουσίαζε καμία αλλαγή.

2.5 Γλώσσες προγραμματισμού για προσομοίωση

Υπάρχουν δύο είδη γλωσσών προγραμματισμού που χρησιμοποιούνται στην προσομοίωση:

- οι γλώσσες προσομοίωσης
- οι γλώσσες γενικού σκοπού.

Στην πρώτη κατηγορία ανήκουν οι GPSS, SIMSCRIPT και SLAM II ενώ στη δεύτερη οι FORTRAN, C, C++, Java κ.τ.λ. Χαρακτηριστικά των συνήθων ειδικών γλωσσών προσομοίωσης φαίνονται στον παρακάτω πίνακα.

Γλώσσα	GPSS	SIMSCRIPT	SLAM II
Προαπαιτούμενες Γνώσεις	Καμία	FORTRAN ή C	Καμία (διασύνδεση με FORTRAN)
Σύμβολα Διαγράμματος Ροής	Ειδικά	Μη Ειδικά	Ειδικά
Μέθοδος Μοντελοποίησης	Βάσει των Διαδικασιών	Βάσει των Γεγονότων	Βάσει των Διαδικασιών ή Βάσει των Γεγονότων
Μονάδα Προγραμματισμού	Block	Πρόταση SIMSCRIPT, Ρουτίνα Γεγονότων	Δραστηριότητα Κόμβου
Οντότητα Προσομοίωσης	Συναλλαγή (transaction)	Προσωρινή Οντότητα	Οντότητα

Εικόνα 14: Τυπικές Ειδικές Γλώσσες Προσομοίωσης

Οι γλώσσες προσομοίωσης συμπεριλαμβάνουν τις ακόλουθες συναρτήσεις:

1. Δημιουργία τυχαίων αριθμών,
2. Εκτέλεση χρονοδιαγραμμάτων.
3. Σώσιμο δεδομένων, στατιστική ανάλυση, ποικιλία παρουσίασης αποτελεσμάτων.

Το πλεονέκτημα της χρήσης ειδικών γλωσσών προσομοίωσης είναι ότι δεν χρειάζεται να προγραμματίσουμε συναρτήσεις όπως αυτές που προαναφέραμε, κι ότι διευκολύνεται ο προγραμματισμός απ' την στιγμή που το μοντέλο έχει τυποποιηθεί.

Απ' την άλλη πλευρά, οι γλώσσες γενικού σκοπού είναι πιο ευπροσάρμοστες από τις ειδικές γλώσσες προσομοίωσης. Επιπλέον μπορούν να επιτύχουν ταχύτερη εκτέλεση. Η επιλογή του είδους της γλώσσας εξαρτάται από πολλούς παράγοντες όπως η ευκολία χρήσης, οι γνώσεις του προγραμματιστή, τα χαρακτηριστικά των μοντέλων κ.λ.π.

2.6 Ανάλυση αποτελεσμάτων προσομοίωσης

Για να υπολογίσουμε την μέση τιμή X των δειγμάτων που παίρνουμε από τα αποτελέσματα της προσομοίωσης εφαρμόζουμε πολύ συχνά την μέθοδο των διαστημάτων εμπιστοσύνης (confidence intervals). Όσο μικρότερο είναι το διάστημα τόσο ακριβέστερος είναι ο υπολογισμός της μέσης τιμής.

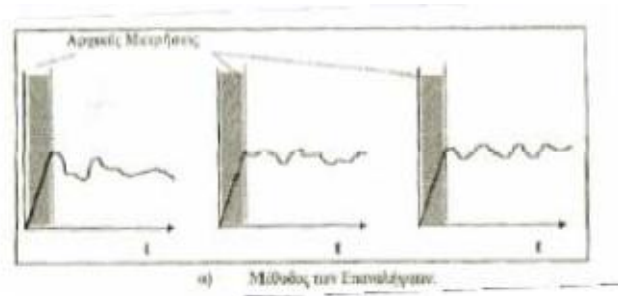
Ένα διάστημα εμπιστοσύνης υπολογίζεται με ταξινόμηση των αποτελεσμάτων σε κατάλληλα σύνολα βάσει μεθόδων όπως:

- Η μέθοδος των επαναλήψεων (η προσομοίωση «τρέχει» πολλές φορές)

  Με την μέθοδο των επαναλήψεων πραγματοποιείται ένας αριθμός ανεξαρτήτων προσομοιώσεων και υπολογίζονται η μέση τιμή και το διάστημα εμπιστοσύνης, αφού απορριφθούν οι αρχικές μετρήσεις (της μεταβατικής κατάστασης) σύμφωνα με τα ακόλουθα (σχήμα):

  1. Η προσομοίωση ξεκινάει από μία καθορισμένη αρχική κατάσταση ισορροπίας, την οποία βρίσκουμε από θεωρητική μελέτη, οπότε ενδεχομένως να μη χρειάζεται να απορρίψουμε τις αρχικές μετρήσεις.
  2. Η προσομοίωση ξεκινάει από την μηδενική αρχική κατάσταση, όπου το σύστημα είναι κενό, και απορρίπτονται σι αρχικές μετρήσεις μέχρις ενός προκαθορισμένου χρόνου T.

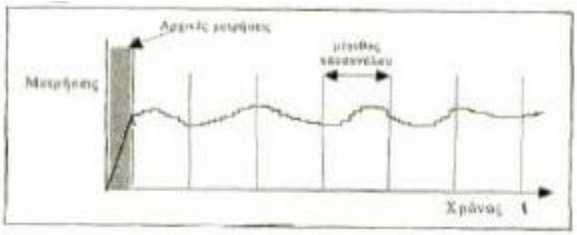
  Αν και ο καθορισμός του T είναι προτιμότερο να γίνεται έπειτα από δοκιμαστικό έλεγχο, στην πράξη γίνεται ευρετικά επιθεωρώντας τις μετρήσεις.



Εικόνα 15: Μέθοδος των επαναλήψεων

- Η μέθοδος της μέσης τιμής υποσυνόλων (η προσομοίωση «τρέχει» μια φορά)

  Στην μέθοδο αυτή, n δείγματα τα οποία έχουν ληφθεί από την εκτέλεση μιας προσομοίωσης χωρίζονται σε Μ υποσύνολα που το καθένα περιλαμβάνει n'= n/M δείγματα.



Εικόνα 16: Μέθοδος της μέσης τιμής υποσυνόλων

2.7 Συχνά λάθη στις προσομοιώσεις

Τα πιο συχνά λάθη που γίνονται στις προσομοιώσεις είναι τα εξής:

1) Ανάρμοστο επίπεδο λεπτομέρειας: Η προσομοίωση επιτρέπει την μελέτη ενός συστήματος με περισσότερες λεπτομέρειες απ' ό,τι η αναλυτική μέθοδος. Το πόσο λεπτομερειακό θα είναι ένα μοντέλο προσομοίωσης εξαρτάται μόνο από τον διαθέσιμο χρόνο για την ανάπτυξη της προσομοίωσης. Όσο περισσότερο λεπτομερειακή απαιτούμε να είναι η προσομοίωση, τόσο περισσότερος χρόνος απαιτείται για την ανάπτυξη της, και αυξάνεται η πιθανότητα ύπαρξης ατελειών. Επίσης αυξάνεται ο χρόνος για την απομάκρυνση αυτών των ατελειών (debugging time). Μια προσομοίωση με αρκετές λεπτομέρειες απαιτεί επίσης πολύ υπολογιστικό χρόνο (CPU time) για να «τρέξει» (ώρες ή μέρες ή χρόνια).

2) Ακατάλληλη γλώσσα προγραμματισμού: Η επιλογή της γλώσσας προγραμματισμού έχει σημαντικό αντίκτυπο στην γρήγορη ανάπτυξη του μοντέλου. Οι γλώσσες προσομοίωσης ειδικού σκοπού απαιτούν λιγότερο χρόνο για την ανάπτυξη του μοντέλου και διευκολύνουν διάφορα γενικά καθήκοντα όπως η επαλήθευση (verification) και η στατιστική ανάλυση. Οι γλώσσες γενικού σκοπού είναι περισσότερο οικείες στον προγραμματιστή και παρέχουν καλύτερο έλεγχο στην αποτελεσματικότητα της προσομοίωσης και στον χρόνο εκτέλεσης.

3) Ακατάλληλη αντιμετώπιση αρχικών συνθηκών: Το αρχικό στάδιο μιας προσομοίωσης δεν είναι αντιπροσωπευτικό της συμπεριφοράς ενός συστήματος σε μια κατάσταση ισορροπίας. Συνήθως είναι το μεταβατικό στάδιο και επομένως επιβάλλεται να απορριφθεί (να μη προσμετρηθεί στα αποτελέσματα).

4) Προσομοιώσεις πολύ μικρής διάρκειας: Οι αναλυτές συχνά προσπαθούν να εξοικονομήσουν χρόνο «τρέχοντας» πολύ μικρές προσομοιώσεις. Τα αποτελέσματα σε αυτές τις περιπτώσεις εξαρτώνται κατά μεγάλο βαθμό από τις αρχικές συνθήκες και μπορεί να μην αντιπροσωπεύουν το πραγματικό σύστημα.

5) Πτωχές γεννήτριες τυχαίων αριθμών: Τα μοντέλα προσομοίωσης απαιτούν τυχαίες ποσότητες και οι διαδικασίες για την παραγωγή τυχαίων αριθμών καλούνται γεννήτριες τυχαίων αριθμών (random munber generators). Είναι ασφαλέστερο να χρησιμοποιηθεί μια γνωστή γεννήτρια η οποία έχει διεξοδικά αναλυθεί παρά να αναπτύξει κάποιος την δική του.

6) Ακατάλληλη επιλογή των αριθμών «seeds»: Οι γεννήτριες τυχαίων αριθμών είναι υπολογιστικές διαδικασίες στις οποίες αν δοθεί ένας τυχαίος αριθμός παράγουν έναν άλλο. Ο πρώτος τυχαίος αριθμός σ' αυτή την ακολουθία αριθμών ονομάζεται «seed» και παρέχεται από τον αναλυτή. Ο αριθμός «seed» για διαφορετικές ακολουθίες τυχαίων αριθμών πρέπει να επιλέγεται με πολύ προσοχή για να διατηρηθεί η στατιστική ανεξαρτησία μεταξύ των ακολουθιών. Συχνά οι αναλυτές είτε διανέμουν μια ακολουθία τυχαίων αριθμών σε διαφορετικές διαδικασίας (processes) είτε χρησιμοποιούν το ίδιο «seed» για όλα τις ακολουθίες τυχαίων αριθμών. Αυτό εισάγει μια συσχέτιση μεταξύ των διαφόρων διαδικασιών στο

σύστημα και οδηγεί σε συμπεράσματα που μπορεί να μην είναι αντιπροσωπευτικά του πραγματικού συστήματος.

ΚΕΦΑΛΑΙΟ 3: Παρουσίαση ολοκληρωμένων πακέτων λογισμικού που χρησιμοποιούνται στην μοντελοποίηση και προσομοίωση δικτύων υπολογιστών

Στο κεφάλαιο αυτό θα παρουσιαστούν μερικά πακέτα λογισμικού που χρησιμοποιούνται στην μοντελοποίηση και προσομοίωση δικτύων, όπως το OMNET++, ο NS-2, το JSIM και το OPNET Modeler. Επίσης, θα σχολιαστούν τα πλεονεκτήματα και τα μειονεκτήματα των πακέτων αυτών, καθώς και οι χρήσεις τους. Να σημειωθεί πως ένα tutorial χρήσης των προσομοιωτών ξεφεύγει από τους σκοπούς της εργασίας, μιας και το μέγεθος τους είναι μεγάλο και αφετέρου υπάρχει διαθέσιμο μεγάλο πλήθος αυτών στο Internet. Τέλος, θα γίνει και μια συγκριτική αξιολόγηση.

3.1 Προσομοιωτής OPNET Modeler

Ο προσομοιωτής OPNET Modeler (OPNET Technologies Inc., 2011) είναι ένας εμπορικός προσομοιωτής που μπορεί να χρησιμοποιηθεί δωρεάν για εκπαιδευτικούς σκοπούς. Τα αρκτικόλεξο OPNET σημαίνει Optimized Network Engineering Tools. Η πρώτη του έκδοση δημιουργήθηκε το 1986 από τον συνιδρυτή και πρόεδρο της εταιρίας, Alain Cohen, στα πλαίσια μιας εργασίας με τη γλώσσα προγραμματισμού C++ στο Massachusetts Institute of Technology.

Ο προσομοιωτής OPNET Modeler (εφεξής OPNET) είναι ένα εργαλείο της σουίτας εφαρμογών OPNET Technologies Suite που επιταχύνει την διαδικασία ανάλυσης και σχεδιασμού τηλεπικοινωνιακών δικτύων, συσκευών, πρωτοκόλλων και εφαρμογών. Σε συνδυασμό με άλλες υπηρεσίες-εφαρμογές της εταιρίας επιτρέπει:

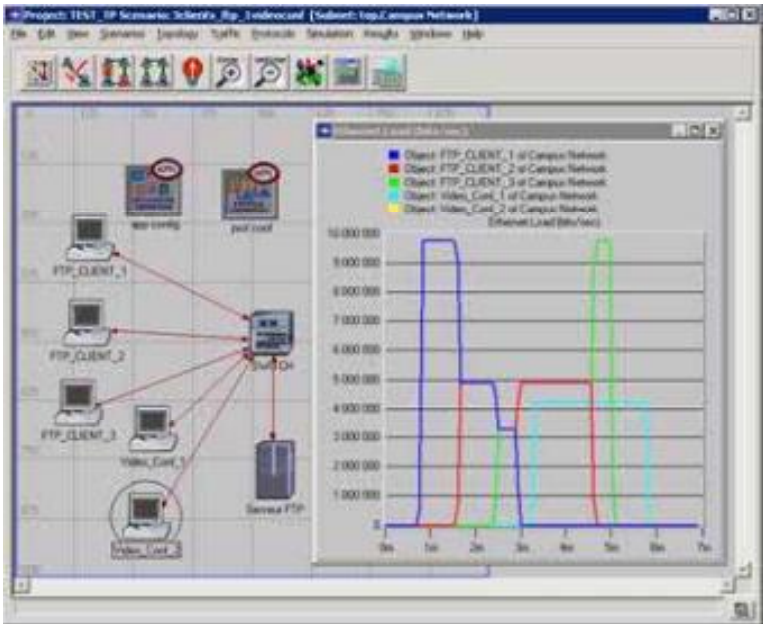
- Εκτίμηση και ενίσχυση ασύρματων πρωτοκόλλων
- Σχεδιασμό πρωτοκόλλων δρομολόγησης
- Μελέτη νέων σεναρίων ενεργειακής διαχείρισης για δίκτυα
- Έρευνα και βελτίωση τεχνολογιών δικτύου π.χ. IPv6, MPLS, κτλ.
- Ανάλυση οπτικών σχεδίων δικτύων

Ο προσομοιωτής διαθέτει μια ευρεία γκάμα από πρωτόκολλα και τεχνολογίες και επιτρέπει την μοντελοποίηση όλων των τύπων δικτύων και τεχνολογιών, όπως:

- VoIP
- TCP
- OSPFv3
- MPLS
- IPv6
- και άλλα

Ο χρήστης έχει την δυνατότητα να αναλύσει τις προσομοιώσεις και να συγκρίνει την επίδραση των διάφορων τεχνολογικών λύσεων. Το Contributed Papers Library είναι ένα ανοιχτό αρχείο από OPNET μελέτες και επιστημονικών εργασιών από χρήστες OPNET σε εκπαιδευτικό, κυβερνητικό και βιομηχανικό επίπεδο όπου γίνεται το “ανέβασμα” (upload) και η σύγκριση των αποτελεσμάτων.

Ένα στιγμιότυπο από το γραφικό περιβάλλον του OPNET φαίνεται στην παρακάτω εικόνα.

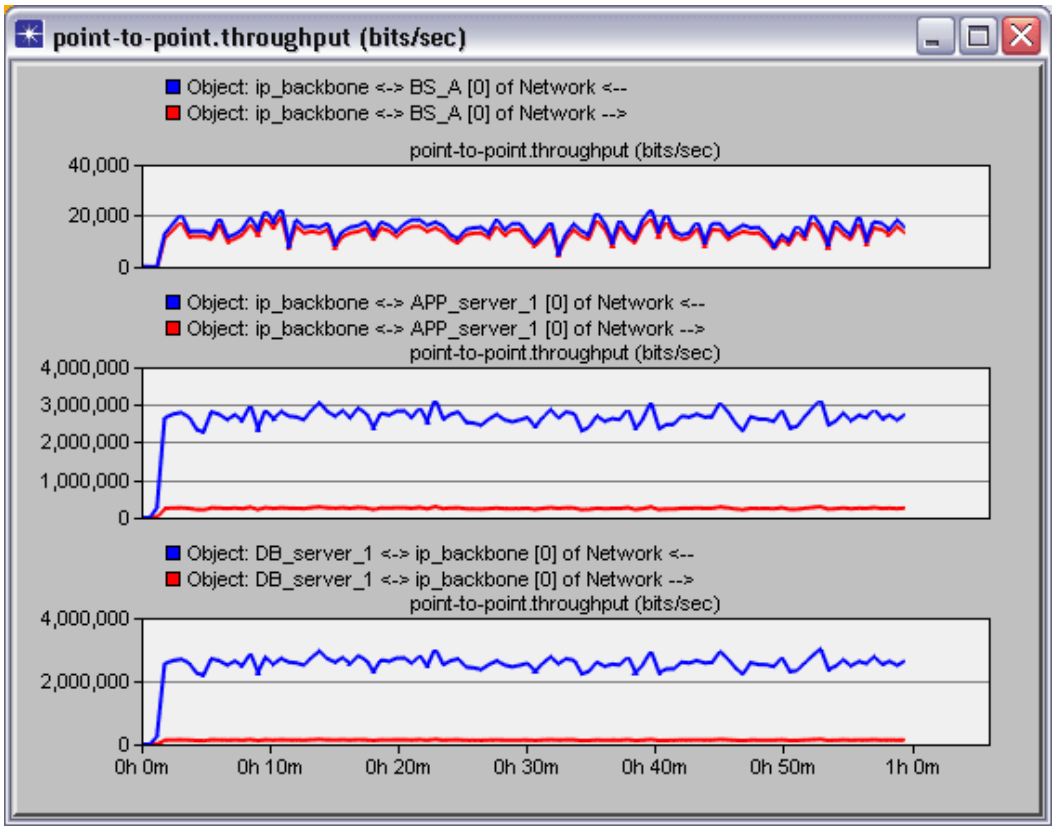


Εικόνα 17: Γραφικό περιβάλλον του προσομοιωτή OPNET

Ο προσομοιωτής έχει τρεις κύριες λειτουργίες:

- Μοντελοποίηση
- Προσομοίωση και
- Ανάλυση.

Ο OPNET διαθέτει μία εύχρηστη γραφική διεπιφάνεια (GUI) με την ονομασία Project για τη δημιουργία, τροποποίηση και επαλήθευση διάφορων μοντέλων σε περιπτώσεις σεναρίων ‘‘τι και αν;’’ Για την προσομοίωση παρέχονται διάφορα μοντέλα ανάλογα με τις συνθήκες και ανάγκες του εκάστοτε σεναρίου όπου ο χρήστης καθορίζει διεργασίες μέσα από ένα σύνολο 400 διαθέσιμων συναρτήσεων μοντελοποίησης (OPNET Manual, 2011). Για την ανάλυση, τα αποτελέσματα της προσομοίωσης μπορούν να αναλυθούν και να προβληθούν εύκολα με τη βοήθεια γράφων και animation.



Εικόνα 18: Απεικόνιση αποτελεσμάτων προσομοίωσης με OPNET

Οι διεργασίες που καθορίζει ο χρήστης εκτελούνται πάνω από ένα μεταφραστή της γλώσσας προγραμματισμού C (C compiler) και C++ σε συνδυασμό με μία τεράστια βιβλιοθήκη συγκεκριμένων συναρτήσεων της OPNET (VTT, 2006). Ο OPNET δημιουργεί τα μοντέλα προσομοίωσης με ιεραρχική δομή χρησιμοποιώντας editors, οι οποίοι παραλληλίζουν την δομή των πραγματικών δικτύων, πρωτοκόλλων και εφαρμογών. Τα μοντέλα μπορούν να δημιουργηθούν από υπάρχοντα συστατικά (components) είτε με top-down ανάλυση είτε με bottom-up ανάλυση.

Παρακάτω περιγράφεται τι εξυπηρετεί ο κάθε editor:

- Project Editor

  Στο επίπεδο αυτό λαμβάνει χώρα η μοντελοποίηση της τοπολογίας του δικτύου και η συνολική παραμετροποίηση. Τα στοιχεία του δικτύου, όπως γραμμές σύνδεσης και δικτυακές συσκευές, χρησιμοποιούνται στη δημιουργία του μοντέλου.

- Node Editor

  Στο επίπεδο αυτό μοντελοποιείται η εσωτερική δομή των δικτυακών συσκευών ή συστημάτων απεικονίζοντας την ροή των δεδομένων μεταξύ λειτουργικών μοντέλων, τα modules. Τα στοιχεία που μπορούν να χρησιμοποιηθούν στη μοντελοποίηση είναι: πομποί, δέκτες και queue modules. Τα στοιχεία αυτά συνδέονται είτε με καλώδια είτε ασύρματα.

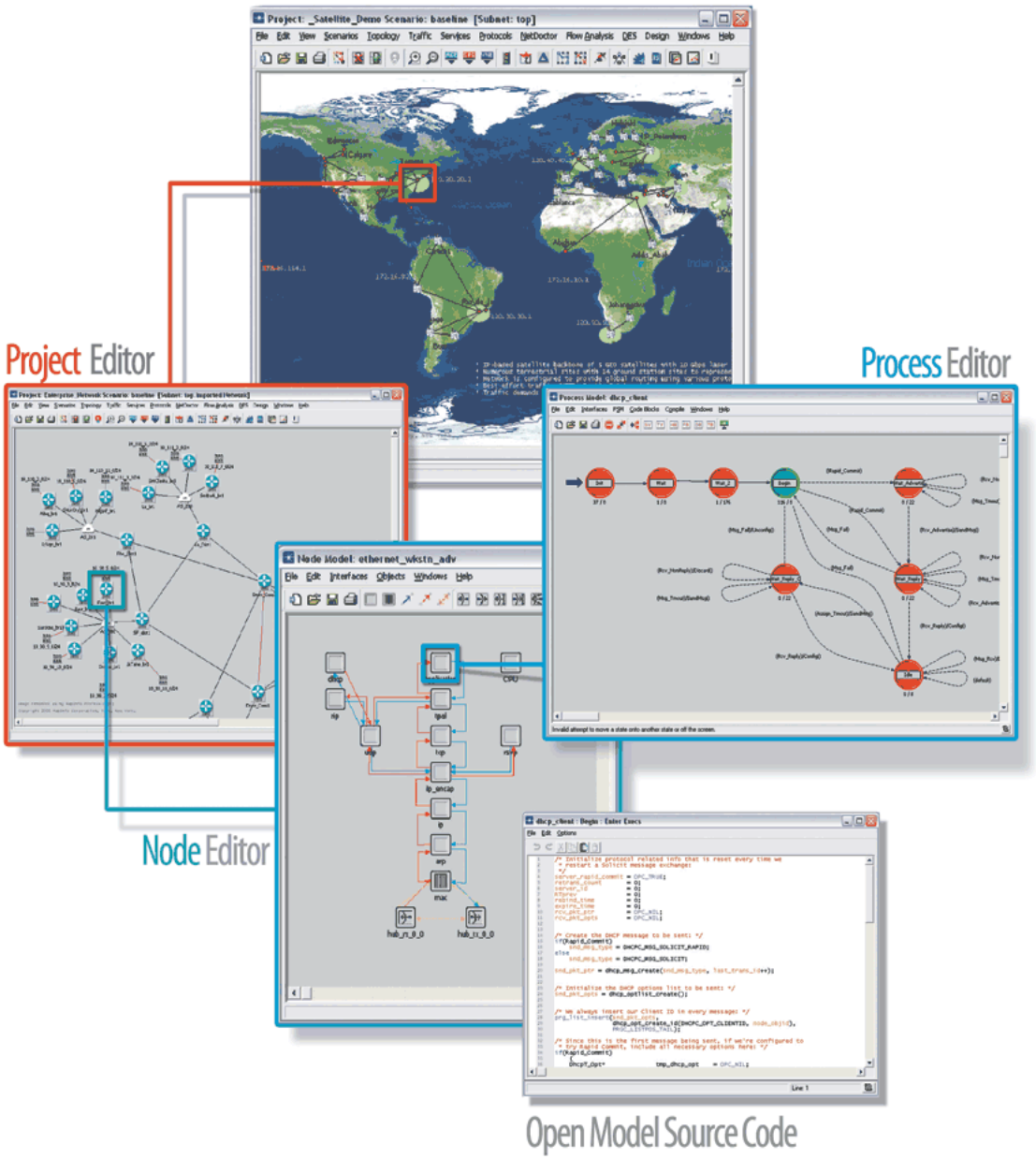
- Process Editor

  Στο επίπεδο αυτό καθορίζονται οι λειτουργίες των δικτυακών συσκευών, πρωτοκόλλων, διαθέσιμοι πόροι, εφαρμογές και αλγόριθμοι.

- Επίπεδο προτυποποίησης

  Το χαμηλότερο επίπεδο ονομάζεται proto-C level. Η proto-C είναι μια επέκταση της γλώσσας προγραμματισμού C, που διαθέτει ένα μεγάλο αριθμό διαδικασιών πυρήνα.

Το ιεραρχικό επίπεδο των επιπέδων αποτυπώνεται στην παρακάτω εικόνα.



Εικόνα 19: Η ιεραρχική δομή των μοντέλων του προσομοιωτή OPNET

Τα μοντέλα μπορούν να τροποποιηθούν σε κάθε επίπεδο. Επίσης ένας μεγάλος αριθμός από μοντέλα πρωτοκόλλων και συσκευές είναι διαθέσιμος, αφού με την αγορά αδειών άλλων προϊόντων της εταιρείας OPNET Technologies μπορούν να ενσωματωθούν στον προσομοιωτή. Προαιρετικά για τον OPNET προσφέρονται τα ακόλουθα μοντέλα:

- ACE Module
- Flow Analysis Module
- High Level Architecture Module (μόνο για 32-bit προσομοιώσεις)

- Pv6 Planning and Operations Module
- NetDoctor Module

Η εισαγωγή των δεδομένων μπορεί να γίνει είτε με τη χρήση του γραφικού περιβάλλοντος είτε από τη γραμμή εντολών. Παρατίθεται ένα παράδειγμα εισαγωγής CBR traffic (Siteceerx.edu).

1. Εισαγωγή μετρήσεων κίνησης ως trace αρχεία
2. Δημιουργία μιας μη προεπιλεγμένης διεργασίας για εισαγωγή των trace αρχείων
3. Εισαγωγή των trace αρχείων ως πηγή κίνησης

Πριν την εκτέλεση της προσομοίωσης, ο χρήστης πρέπει να καθορίσει τις παραμέτρους εκείνες που θα βελτιστοποιήσουν τα αποτελέσματα. Κάθε κόμβος τίθεται σε λειτουργία θέτοντας τις παραμέτρους του, που ορίζουν την εσωτερική του δομή. Η επικοινωνία μεταξύ των κόμβων επιτυγχάνεται με την ανταλλαγή μηνυμάτων. Οι χρήστες μπορούν να παραμετροποιήσουν τις εφαρμογές που είναι εγκατεστημένες σε κάθε κόμβο και να θέσουν τους κόμβους και τις επικοινωνιακές γραμμές να ανακτώνται κατά τη διάρκεια της προσομοίωσης σε καθορισμένες χρονικές στιγμές. Αναφέρονται ενδεικτικά κάποιες παράμετροι και ο σκοπός χρήσης τους όπως:

- Version of TCP

  Βελτιώνει την δυνατότητα γρήγορης ανάκτησης χαμένων πακέτων

- Window Scaling

  Επιτρέπει την αναγγελία μεγαλύτερου μεγέθους παραθύρων

- Time-Stamp

  Μιμείται τον χρονικό απόηχο των μετρήσεων

Κατά τη διαδικασία της προσομοίωσης, τα αποτελέσματα αποθηκεύονται σε αρχεία προσπελάσιμα από τον χρήστη ώστε να εξεταστεί η πρόοδος της προσομοίωσης σε διαφορετικά στάδια. Επίσης είναι δυνατό ο χρήστης να καθορίσει ένα σύνολο από προσομοιώσεις, καθορίζοντας τις παραμέτρους εισόδου ή δημιουργώντας διάφορα σενάρια. Τα στατιστικά για την απόδοση του προσομοιωμένου δικτύου μπορούν να συλλεχθούν κατά την εκτέλεση της προσομοίωσης.

Η σωστή καταγραφή και εισαγωγή των δεδομένων, η παραμετροποίηση και η μοντελοποίηση στα πρότυπα του πραγματικού σεναρίου μπορεί να έχει καταλυτική σημασία ως προς τα αποτελέσματα της προσομοίωσης.

Ο προσομοιωτής έχει βελτιστοποιηθεί για να τρέχει το ίδιο αποδοτικά σε υπολογιστές με ένα επεξεργαστή και σε υπολογιστές με πολλούς επεξεργαστές εκμεταλλευόμενος διάφορες τεχνολογίες (multi-threads, virtualization). Αναφέρονται οι απαιτήσεις συστήματος της τελευταίας έκδοσης (v16.0):

- Λειτουργικό σύστημα

MS Windows 2000, XP, Vista, 7, Server 2000,2003,2008 (32-64 bit)

Red Hat Enterprise Linux 4, Linux 5

Fedora Linux 6

- CPU

2.0GHz για MS Windows (προτείνεται 3+ GHz)

1.0GHz για Linux

- RAM

512 MB (προτείνεται 1-2 GB)

- Αποθήκευση

3 GB ελεύθερου χώρου

Σύμφωνα με την OPNET Technologies, τα κυριότερα χαρακτηριστικά του προσομοιωτή OPNET είναι:

1) Γρήγορη μηχανή προσομοίωσης
2) Πληθώρα βιβλιοθηκών με πηγαίο κώδικα
3) Αντικειμενοστραφής μοντελοποίηση
4) Ιεραρχικό περιβάλλον μοντελοποίησης
5) Υποστήριξη ασύρματων προσομοιώσεων
6) Γραφικό περιβάλλον τόσο για υπολογιστές 32 bit όσο και 64 bit.
7) Υποστήριξη κατανεμημένων προσομοιώσεων.
8) Ανοικτή διεπαφή για ενσωμάτωση εξωτερικών στοιχείων.

Στα αρνητικά συγκαταλέγεται το γεγονός ότι ο OPNET δεν είναι ανοιχτού κώδικα και δεν είναι εύκολη η παραμετροποίηση των προσφερόμενων συναρτήσεων πέρα από το όριο που η OPNET Technologies επιτρέπει. Σε αντίθεση με άλλους προσομοιωτές ο OPNET μπορεί να χαρακτηριστεί σαν ‘‘βαριά’’ εφαρμογή με αυξημένες υπολογιστικές ανάγκες.

3.2 Προσομοιωτής NS-2

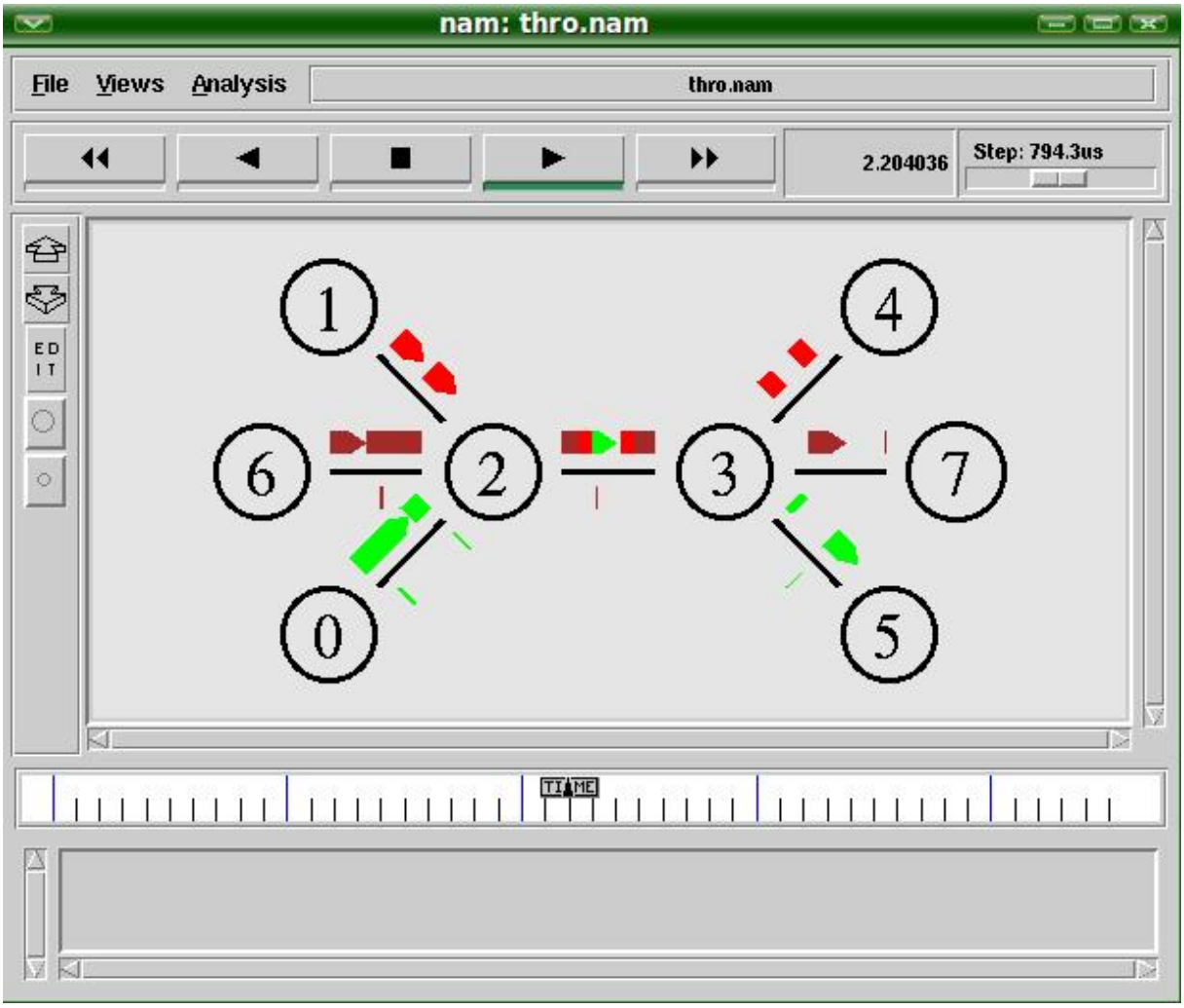
Ο προσομοιωτής NS-2 (NS-2, 2011) είναι η δεύτερη έκδοση ενός δικτυακού προσομοιωτή που έχει δημιουργηθεί από το project Virtual InterNetwork Testbed (Bajaj et al., 1999) στο Berkeley University of California. Ο προσομοιωτής αυτός είναι ο πιο συχνά χρησιμοποιούμενος σε ερευνητικά έργα σχετικά με δίκτυα IP. Υπάρχουν πολλοί καταξιωμένοι επιστήμονες που έχουν συμμετάσχει στο project του NS-2 και για αυτό ο προσομοιωτής διαθέτει πολλές σύγχρονες λειτουργίες που σχετίζονται με τα δίκτυα IP.

Είναι ιδιαίτερα δημοφιλής στην ερευνητική κοινότητα και παρέχεται εντελώς δωρεάν τόσο για εκπαιδευτικούς όσο και για εμπορικούς σκοπούς. Ο προσομοιωτής κατεβαίνει με τον πλήρη πηγαίο κώδικα, ένα μεγάλο πλήθος παραδειγμάτων και ένα αρκετά καλό εγχειρίδιο χρήστη αφού μπορεί να δυσκολέψει κάποιους αρχάριους χρήστες, για αυτό τον λόγο παρέχεται βοήθεια από την κοινότητα των χρηστών του μέσω μιας mailing list που υπάρχει.

Ο προσομοιωτής αυτός ενεργεί σε διακριτά γεγονότα και είναι γραμμένος σε C++ και OTcl. Ο NS-2 διαθέτει μια πληθώρα μοντέλων από ευρέως διαδεδομένα πρωτόκολλα του Διαδικτύου, αλλά και νεότερα που έχουν κάνει την εμφάνισή τους (Fall, 1999) και είναι κυρίως χρήσιμος για την προσομοίωση τοπικών και ευρείας περιοχής δικτύων. Συγκεκριμένα επιτρέπει την προσομοίωση (isi.edu):

- Επίγειων, ασύρματων και δορυφορικών δικτύων με διάφορους αλγόριθμους δρομολόγησης (DV, LS, PIM-DM, PIM-SM, AODV, DSR)
- Συμπεριφορών πηγών κίνησης πακέτων όπως το web, ftp, telnet, cbr, vbr στοχαστική κίνηση
- Αποτυχημένων συνδέσεων, βλαβών, λαθών πιθανοτήτων
- Πρωτοκόλλων TCP, δρομολόγησης, πολύ-εκπομπής (multicast)

Η γραφική διεπιφάνεια (GUI) που χρησιμοποιείται ονομάζεται NAM (Bajaj et al., 1999), η οποία παρέχει γραφήματα στο επίπεδο πακέτου του δικτύου και διάφορα εργαλεία για τη δημιουργία και τη διόρθωση σφαλμάτων στα επίπεδα του δικτύου. Το αρκτικόλεξο NAM σημαίνει Network Animator και οπτικά μοιάζει με ένα περιβάλλον χρήσης ενός cd player όπως φαίνεται και στην παρακάτω εικόνα.



Εικόνα 20: Γραφική διεπιφάνεια NAM

Μέσα από τον NAM μπορούν να απεικονιστούν διάφορα πρωτόκολλα, στοιχεία της προσομοίωσης και πραγματικά πακέτα ως γραφικές απεικονίσεις. Υποστηρίζει σχεδιαγράμματα τοπολογίας, απεικόνιση επιπέδου πακέτου και διάφορα εργαλεία για τον έλεγχο της προσομοίωσης. Μερικές από τις απεικονίσεις αυτές είναι:

- Ροή και απώλεια πακέτων, συσσώρευση ουράς
- Συμπεριφορά πρωτοκόλλων: αργή εκκίνηση TCP, χρονισμός και έλεγχος κίνησης, γρήγορη αναμετάδοση και ανάκτηση

- Θέσεις κόμβων σε ασύρματα δίκτυα
- Δημιουργία σημειώσεων και σχολίων για σημαντικά γεγονότα
- Κατάσταση πρωτοκόλλων (π.χ. TCP cwnd)

Υπάρχουν διαθέσιμες προ-τυποποιημένες απεικονίσεις και σενάρια (scripts) σε διάφορες βάσεις δεδομένων και φυσικά υπάρχει δυνατότητα "ανεβάσματος" (uploading) στο Διαδίκτυο.

Ο NS-2 είναι ένας αντικειμενοστραφής (Object oriented-OTcl) μεταφραστής (interpreter) σεναρίων (script). Στην ουσία ο NS-2 είναι ένας προσομοιωτής batch-type που διαθέτει σενάρια που σταματούν και ελέγχουν την προσομοίωση, ενώ στο τέλος παρέχουν χρήσιμες πληροφορίες με κατάλληλη μορφοποίηση χρησιμοποιώντας το εργαλείο NAM. Διαθέτει μία βιβλιοθήκη αντικειμένων που αποτελείται από (NS by example, Jae Choung and Mark Claypool):

- Προγραμματιστή γεγονότων

  Ένα γεγονός στον NS-2 είναι ένα πακέτο ταυτότητας (packet ID) το οποίο είναι μοναδικό για κάθε πακέτο με προγραμματισμένο χρόνο εκτέλεσης και συσχετισμένο με το αντικείμενο που διαχειρίζεται το γεγονός. Ο προγραμματιστής γεγονότων παρακολουθεί και κρατάει στοιχεία της προσομοίωσης και εκκινεί όλα τα γεγονότα που βρίσκονται στην ουρά γεγονότων την συγκεκριμένη χρονική στιγμή με την επίκληση κατάλληλων σύνθετων αντικειμένων δικτύων.

- Σύνθετα αντικείμενα δικτύων

  Τα σύνθετα αντικείμενα δικτύων είναι εκείνα που συνήθως καλούν τα γεγονότα, επιτρέποντας τα να εκτελέσουν την κατάλληλη λειτουργία κάνοντας χρήση του πακέτου που είναι συσχετισμένο με το γεγονός. Η επικοινωνία επιτυγχάνεται με την μεταφορά πακέτων, τα οποία δεν επηρεάζουν τον διαθέσιμο χρόνο προσομοίωσης, ενώ σε περίπτωση που κάτι τέτοιο είναι επιθυμητό γίνεται χρήση του προγραμματιστή γεγονότων.

- Βοηθητικά μοντέλα ρύθμισης (setup) δικτύου

  Χαρακτηρίζονται από τον όρο plumbing, δηλαδή ''σωληνωτά'' μοντέλα γιατί ένα δίκτυο μπορεί να παρομοιαστεί οπτικά με υδραυλικές σωληνώσεις. Εφαρμόζονται σαν μέρη συναρτήσεων στην βάση της προσομοίωσης.

Για να γίνει πιο κατανοητό θα παρουσιάσουμε τα στάδια για το ‘‘τρέξιμο’’ (run) μίας προσομοίωσης:

1. Σύνθεση σεναρίου σε γλώσσα OTcl
2. Εκκίνηση προγραμματιστή γεγονότων
3. Ρύθμιση της τοπολογίας δικτύου χρησιμοποιώντας την βιβλιοθήκη αντικειμένων
4. Έναρξη και τερματισμός μετάδοσης πακέτων μέσω του προγραμματιστή γεγονότων

Για λόγους αναφοράς παρακάτω δίνεται μία εικόνα που δείχνει ένα σενάριο γραμμένο σε γλώσσα OTcl:

```
# Writing a procedure called "test"
proc test {} {
    set a 43
    set b 27
    set c [expr $a + $b]
    set d [expr [expr $a - $b] * $c]
    for {set k 0} {$k < 10} {incr k} {
        if {$k < 5} {
            puts "k < 5, pow = [expr pow($d, $k)]"
        } else {
            puts "k >= 5, mod = [expr $d % $k]"
        }
    }
}

# Calling the "test" procedure created above
test
```

Εικόνα 21: Script γραμμένο σε γλώσσα OTcl

Μερικές από τις παραμέτρους που μπορεί να θέσει ο χρήστης που αποσκοπούν σε συγκεκριμένες συμπεριφορές είναι:

- Link Latency (Boolean)
- Max Segment Size (π.χ. 1000/1500)
- Window Size (π.χ. 20/100)

Όπως έχουμε πει ο NS-2 χρησιμοποιεί δύο γλώσσες προγραμματισμού:

- C++

  Χρησιμοποιείται από τον NS-2 για μηχανισμούς προσομοίωσης χαμηλού επιπέδου.

- OTcl (Objective Tool Commanf Language)

  Χρησιμοποιείται από τον NS-2 για μηχανισμούς προσομοίωσης υψηλού επιπέδου (OTcl, 2011).

Ο συνδυασμός αυτών των δύο γλωσσών είναι εξαιρετικά χρήσιμος, γιατί ακόμα και πολύπλοκές ενέργειες μπορούν να γίνουν εύκολα με τη γλώσσα OTcl.

Ο χρήστης γράφει ένα σενάριο σε γλώσσα OTcl, όπου καθορίζει τις παραμέτρους του δικτύου και τα πρωτόκολλα που θα χρησιμοποιήσει στην προσομοίωση του. Το σενάριο αυτό χρησιμοποιείται από τον προσομοιωτή κατά τη διάρκεια της προσομοίωσης. Το αποτέλεσμα αυτής της προσομοίωσης είναι ένα ή περισσότερα αρχεία καταγραφής, τα οποία μπορούν να χρησιμοποιηθούν για την επεξεργασία των δεδομένων της προσομοίωσης. Στις περισσότερες περιπτώσεις, ακόμα και σε προχωρημένες, είναι αρκετό να δημιουργηθεί OTcl κώδικας (υψηλού επιπέδου), αλλά αν υπάρχει η ανάγκη χαμηλού επιπέδου χειρισμού των πακέτων, τότε απαιτείται η δημιουργία C++ κώδικα.

Ο λόγος που χρησιμοποιεί δύο διαφορετικές γλώσσες προγραμματισμού είναι γιατί στην πραγματικότητα ο προσομοιωτής έχει δύο διαφορετικά πράγματα να κάνει:

1. Λεπτομερής προσομοίωση των πρωτοκόλλων του δικτύου

   Απαιτεί μία γλώσσα προγραμματισμού συστήματος η οποία μπορεί να χειριστεί αποτελεσματικά byte, επικεφαλίδες πακέτων και να υλοποιηθούν αλγόριθμοι που εκτελούνται για μεγάλα σύνολα δεδομένων. Η γλώσσα αυτή είναι η C++.

2. Έρευνα και εξέταση δικτύων

   Ένα μεγάλο κομμάτι της έρευνας δικτύων περιλαμβάνει εξέταση πολλών παραμέτρων, αλλά και σεναρίων, τα οποία μπορούν να επιτευχθούν με τη γλώσσα σεναρίων OTcl.

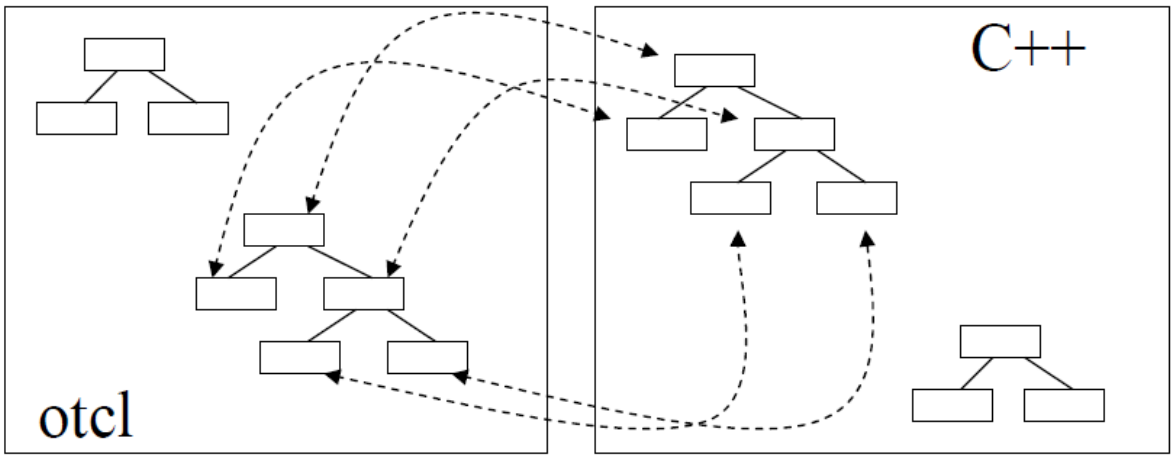
Ο NS-2 επιτυγχάνει και τους δύο στόχους. Η C++, ως γλώσσα προγραμματισμού, είναι γρήγορη στην εκτέλεση αλλά αργή στην αλλαγή του κώδικα, κάτι που την κάνει κατάλληλη για λεπτομερή υλοποίηση πρωτοκόλλων. Αντίθετα, η OTcl, ως γλώσσα κωδικοποίησης σεναρίων, εκτελείται πολύ πιο αργά αλλά είναι ιδανική για γρήγορες αλλαγές στον κώδικα, κάτι που την κάνει ιδανική για παραμετροποίηση και έλεγχο των προσομοιώσεων. Δηλαδή έχουμε έναν συμβιβασμό ανάμεσα στην μεγάλη ταχύτητα εκτέλεσης που παρέχεται από μια γλώσσα όπως η C++ και στους μικρούς χρόνους γραφής ενός σεναρίου σε

μια γλώσσα όπως η OTcl. Αυτή η διαλειτουργικότητα υλοποιείται με την παρουσία δύο ιεραρχιών κλάσεων:

- Μία μεταγλωττισμένη ιεραρχία από ομάδες αντικειμένων σε C++
- Μία διερμηνευμένη ιεραρχία από αντικείμενα OTcl.

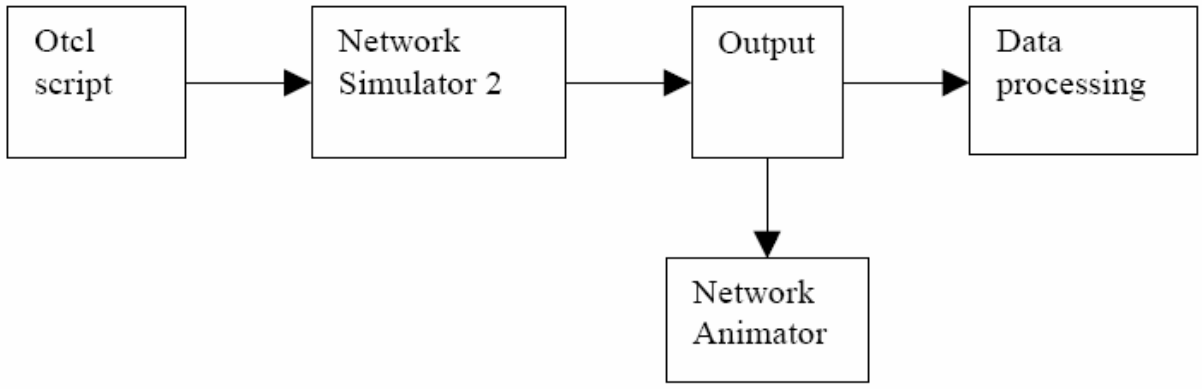
μεταξύ των οποίων υπάρχει μία προς μία αντιστοιχία κλάσεων.

Η παρακάτω εικόνα δείχνει αυτήν την αντιστοιχία.



Εικόνα 22: Αντιστοιχία των κλάσεων της ιεραρχίας της C++ με την ιεραρχία της OTcl

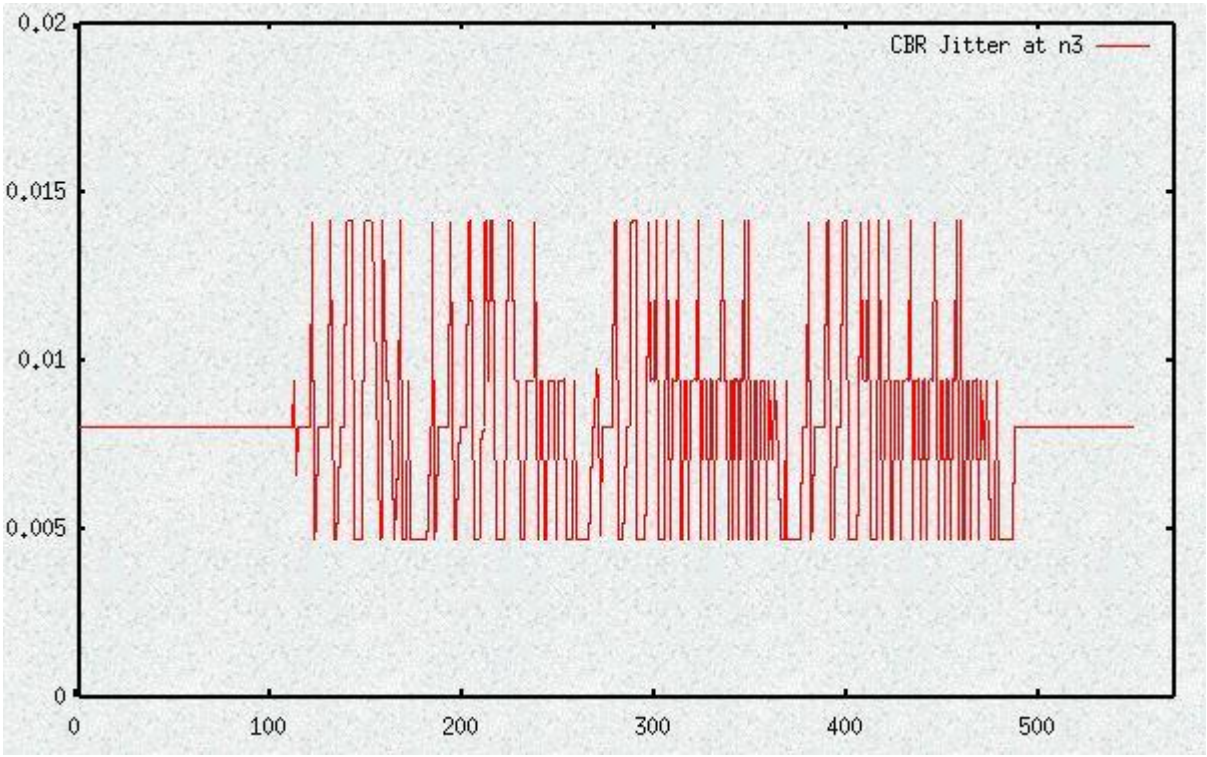
Μια γενική άποψη της δομής του NS-2 φαίνεται στην εικόνα που ακολουθεί.



Εικόνα 23: Γενική άποψη της δομής του προσομοιωτή NS-2

Η ανάλυση των δεδομένων γίνεται με την χρήση διάφορων πακέτων-εργαλείων. Υπάρχει μία παράλληλη και κατανεμημένη έκδοση διαθέσιμη (PDNS, 2011) και δίνεται η επιλογή να γίνει λήψη μεμονωμένων εργαλείων ή όλου του πακέτου λογισμικού. Αυτό είναι ιδιαίτερα χρήσιμο καθότι κυκλοφορεί μεγάλο πλήθος από τέτοια πακέτα-εργαλεία. Για παράδειγμα μπορεί να χρειαζόμαστε μόνο τον προσομοιωτή (NS-2), να γράψουμε ένα σενάριο σε OTcl και να αναλύσουμε τα δεδομένα με τον gnuplot.

Στην παρακάτω εικόνα δίνεται ένα γράφημα που δημιουργήθηκε με την χρήση του gnuplot και προσομοιώνει την λήψη πακέτων στον κόμβο 3. Ο άξονας των x αντιπροσωπεύει την συχνότητα της λήψης πακέτων και ο άξονας των y τον χρόνο προσομοίωσης.



Εικόνα 24: Παρουσίαση αποτελεσμάτων με χρήση gnuplot

Μερικά από τα εργαλεία που διατίθενται είναι τα ακόλουθα:

- Tcl/Tk
- OTcl
- TclCL (γνωστό ως libTcl)
- NS-2
- NAM

- Xgraph
- Tcl-debug
- Dmalloc
- Gnuplot

Εξαιτίας της σχεδίασης του NS-2 η προσομοίωση πολύ μεγάλων δικτύων είναι δύσκολη και χρονοβόρα, ειδικά αν ο χρήστης δεν είναι έμπειρος. Υπάρχουν διάφορα εργαλεία (π.χ. Dmalloc) που βοηθάνε στον έλεγχο και στην διαχείριση της μνήμης RAM, που συνήθως κρίνεται ανεπαρκής σε προσομοιώσεις τέτοιου μεγέθους. Σε μερικές περιπτώσεις μάλιστα η προσομοίωση είναι αδύνατη εξαιτίας των υπερβολικών απαιτήσεων σε υπολογιστικούς πόρους. Για τον λόγο αυτό υπάρχει μία παράλληλη και μία κατανεμημένη έκδοση διαθέσιμη (PDNS, 2011) που επιτρέπει την προσομοίωση να τρέχει σε ξεχωριστούς σταθμούς εργασίας (workstations).

Ο NS-2 είναι διαθέσιμος σε λειτουργικά που βασίζονται στο UNIX (FreeBSD, Linux, SunOS, Solaris, Mac OS X) και η ανάπτυξη του γίνεται σε αυτό το περιβάλλον. Εξάλλου είναι γνωστή η προτίμηση της ερευνητικής κοινότητας σε λύσεις ανοιχτού κώδικα. Για την χρήση του σε MS Windows δημιουργήθηκε το Cygwin port, που προσομοιώνει το περιβάλλον Linux στα MS Windows, με ότι αρνητικά αποτελέσματα μπορεί να έχει η χρήση μίας εφαρμογής που δεν είναι αποκλειστικά γραμμένη για ένα συγκεκριμένο λειτουργικό σύστημα. Το όλο πακέτο χρειάζεται ελάχιστο χώρο στον δίσκο για τα σημερινά δεδομένα (320MB) και χαρακτηρίζεται σαν “ελαφριά” εφαρμογή.

3.3 Προσομοιωτής JSIM

Το JSIM (JSIM, 2011) είναι ένα περιβάλλον προσομοίωσης διακριτών γεγονότων που διατίθεται δωρεάν και είναι βασισμένο σε συστατικά που έχουν αναπτυχθεί με τη γλώσσα προγραμματισμού Java. Δημιουργήθηκε από την ομάδα του Jarda Kacer στο Distributed Realtime Computing Laboratory του Ohio University.

Μερικά από τα κύρια χαρακτηριστικά του είναι:

- Χρησιμοποιείται ανεξάρτητου λειτουργικού συστήματος
- Προσομοίωση σε πραγματικό χρόνο

- Εφαρμογή ενός συνόλου από πρωτόκολλα του Διαδικτύου και Best Effort υπηρεσίες
- “Χαλαρός” ορισμός μοντέλων
- Χρήση περιβάλλοντος που επιτρέπει την αυτόματη παραμετροποίηση και on-line παρακολούθηση

Το γραφικό περιβάλλον του JSIM ονομάζεται gEditor, το οποίο είναι ένα πακέτο κλάσεων Java και λειτουργεί ως γραφικός επεξεργαστής αλλά και ως η διεπαφή επικοινωνίας με το JSIM. Με την χρήση plugins μπορούν να επεκταθούν οι βασικές δυνατότητες, όπως η χρησιμοποίηση εξειδικευμένων εργαλείων απεικόνισης, η μορφή των δεδομένων κ.α. Συγκεκριμένα οι τύποι των plugins που υποστηρίζονται είναι:

- DataFormat

  Επιτρέπει την εισαγωγή και εξαγωγή δεδομένων που δεν υποστηρίζονται από τον JSIM

- Graphic

  Επιτρέπει την χρήση ειδικών γραφικών απεικονίσεων για την αναπαράσταση των ιδιοτήτων, συσχετίσεων και δεδομένων ενός μοντέλου του JSIM

- ModelEasel

  Επιτρέπει την δημιουργία μοντέλων με χρήση της τεχνικής point-and-click (χρησιμοποίηση δείκτη ποντικιού)

- Optimizer

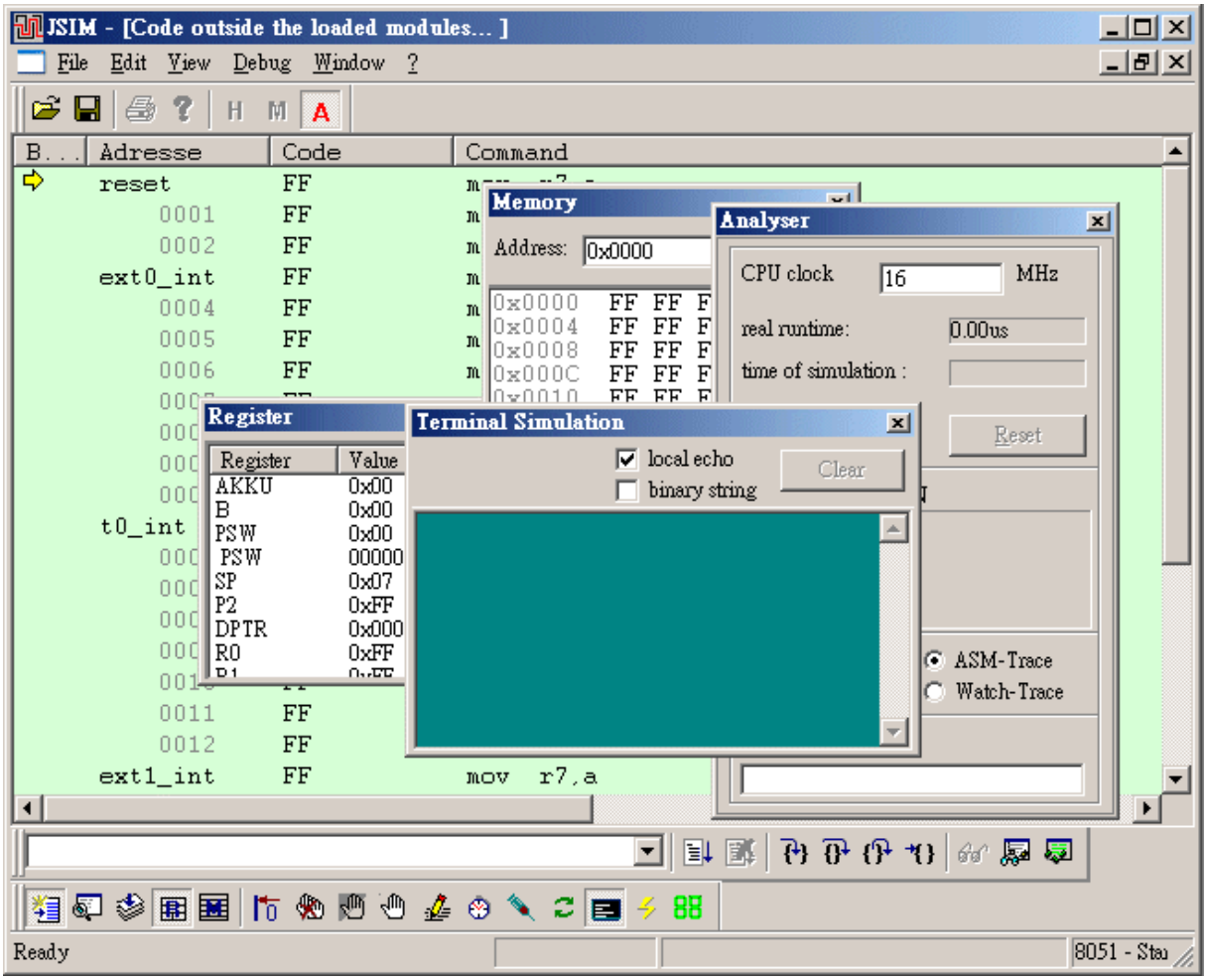
  Επιτρέπει την χρήση βελτιστοποιημένων αλγόριθμων οι οποίοι δεν υποστηρίζονται από το JSIM

Μέσα από το γραφικό περιβάλλον, με την χρήση ή όχι plugins, ο χρήστης έχει την δυνατότητα να χρησιμοποιήσει διάφορες εντολές:

- Σετ από τιμές παραμέτρων του μοντέλου
- Υπολογιστικά μοντέλα
- Πειραματισμό των αποτελεσμάτων και αρχείων αναφοράς
- Σημειώσεις και επικύρωση της μέχρι τότε προόδου
- Γραφική αναπαράσταση των δεδομένων

Ο gEditor χρησιμοποιεί την μηχανή προσομοίωσης του JSIM για να εκτελέσει τις προσομοιώσεις. Επειδή ο προσομοιωτής εκτελεί βήμα-βήμα τις διεργασίες, ο χρήστης μπορεί να ασκήσει έλεγχο στην προσομοίωση, ακόμα και

να προσθέσει μια διεργασία καθώς αυτή εκτελείται, στέλνοντας εντολές προς το JSIM μέσω της κονσόλας εντολών.



Εικόνα 25: Γραφικό περιβάλλον του JSIM

Η γλώσσα για την ανταλλαγή μηνυμάτων με το JSIM ονομάζεται Jacl (Jacl, 2011) και επιτρέπει την παραμετροποίηση αντικειμένων JAVA σε περιβάλλον Tcl. Ο χρήστης έχει την επιλογή να μην την χρησιμοποιήσει μιας και όλα γίνονται με το γραφικό περιβάλλον που παρέχει ο gEditor. Να σημειωθεί πως υπάρχει η δυνατότητα να χρησιμοποιηθούν scripts σε γλώσσες όπως Perl, Python ή Tcl για την παραμετροποίηση και τον ορισμό των ιδιοτήτων του συστήματος.

Ο προσομοιωτής JSIM είναι μία πλατφόρμα προσομοίωσης που βασίζεται στην αρχιτεκτονική ACA (Autonomous Component-based Architecture) και βασίζεται κυρίως σε διεργασίες. Ωστόσο, υποστηρίζει και χαρακτηριστικά μοντελοποίησης και χειρισμού συμβάντων.

Οι βασικές οντότητες της αρχιτεκτονικής ACA είναι τα συστατικά (components), τα οποία επικοινωνούν το ένα με το άλλο μέσω αποστολής και λήψης δεδομένων στις θύρες τους. Κάθε οντότητα δικτύου, π.χ. ένας δρομολογητής, ένα πρωτόκολλο, το δίκτυο είναι ένα συστατικό. Κάθε συστατικό μπορεί να αποτελείται από ένα συνδυασμό άλλων συστατικών και στα πλαίσια της προσομοίωσης δικτύων, ένα δίκτυο αποτελείται από συνδυασμούς συστατικών όπως κόμβοι, συνδέσεις, πρωτόκολλα και κατάλληλα μοντέλα. Ο τρόπος συμπεριφοράς των συστατικών καθορίζεται στον σχεδιασμό του συστήματος με τη δημιουργία συμβολαίων, αλλά η σύνδεση μεταξύ τους επιτυγχάνεται κατά τη σύνθεση του συστήματος. Με αυτόν τον τρόπο είναι εύκολο να εισάγονται νέα συστατικά στο JSIM με τρόπο άμεσης τοποθέτησης και λειτουργίας.

Ένα συστατικό έχει τις ακόλουθες ιδιότητες:

1) Ενεργοποιείται παθητικά από εισερχόμενα δεδομένα
2) Είναι δυνατός ο μηδενισμός και εκ νέου ορισμός του
3) Είναι επιτρεπτή η κλωνοποίηση του
4) Είναι δυνατός ο καθορισμός της κατάστασης του
5) Μπορεί να ενεργεί σαν πηγή δεδομένων

Η δημιουργία ενός συστατικού-πρωτοκόλλου γίνεται με τα ακόλουθα βήματα:

1. Ορισμός της κλάσης πακέτου για το πρωτόκολλο
2. Γράψιμο πρωτοκόλλου
3. Έλεγχος του πρωτοκόλλου με ένα σενάριο σε Tcl

Σε συνέχεια γίνεται η σύναψη συμβολαίου βάση σχεδιασμού του συστήματος

Στον προσομοιωτή JSIM υπάρχουν τριών ειδών κόμβοι:

- Κόμβοι στόχοι

  Οι κόμβοι στόχοι παράγουν τα διάφορα φυσικά συμβάντα του περιβάλλοντος, τα οποία αντιλαμβάνονται οι αισθητήριοι κόμβοι και κατόπιν επεξεργασίας τα στέλνουν ως δεδομένα προς τους συγκεντρωτές κόμβους, οι οποίοι τα συγκεντρώνουν για να τα παραδώσουν στον τελικό χρήστη.

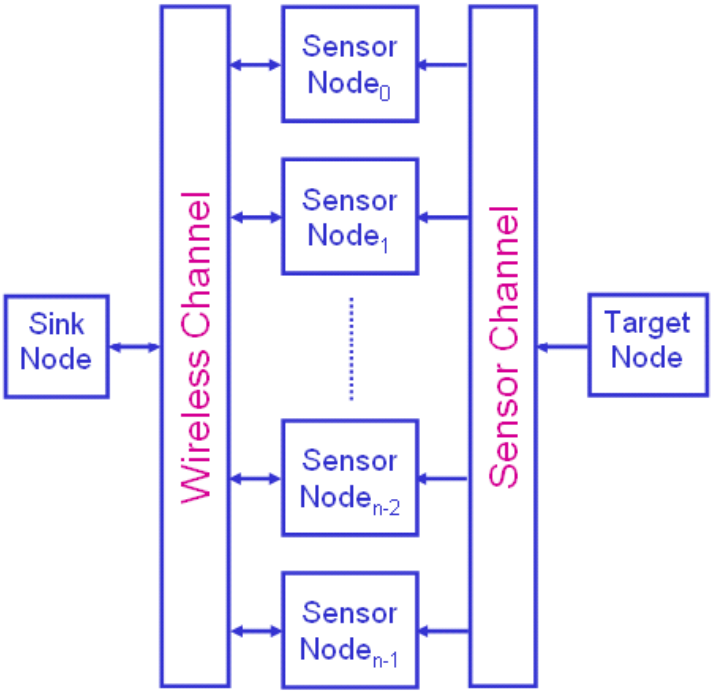
- Αισθητήριοι κόμβοι

  Αισθητήρια κανάλια και κανάλια ασύρματης επικοινωνίας

- Συγκεντρωτές κόμβοι

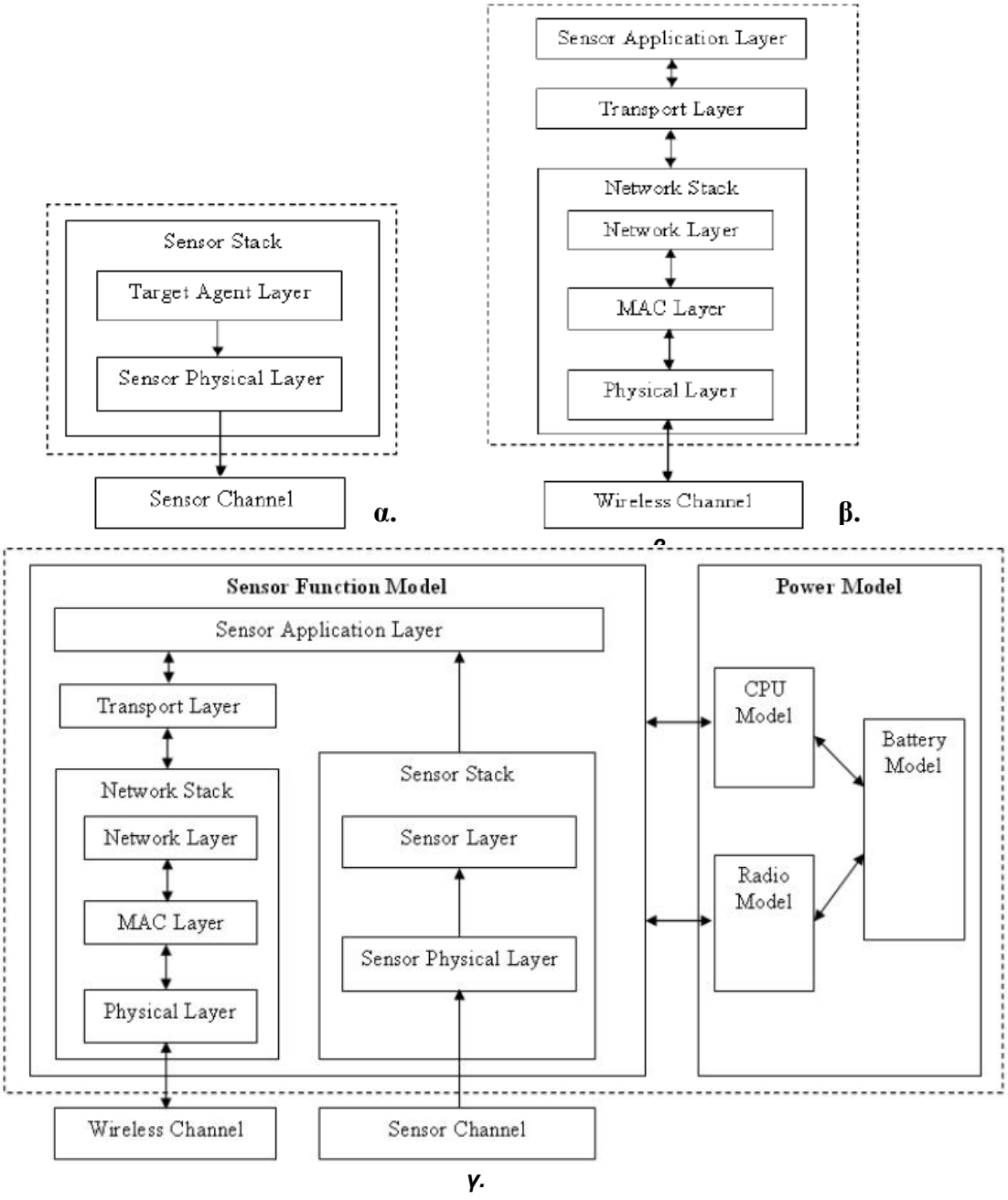
  Φυσικά μέσα όπως είναι τα κανάλια απόδοσης των σεισμικών κυμάτων, μοντέλο κινητικότητας και μοντέλο ενέργειας.

Στην παρακάτω εικόνα απεικονίζεται μια υψηλού επιπέδου όψη της εν λόγω αρχιτεκτονικής.



Εικόνα 26: Μοντέλο αρχιτεκτονικής ACA

Στην παρακάτω εικόνα απεικονίζεται το εσωτερικό των τριών ειδών κόμβων.



Εικόνα 27: Εσωτερικό των α) κόμβων στόχων, β) συγκεντρωτών κόμβων και γ) των αισθητήριων κόμβων

Η αρχιτεκτονική ACA και το γεγονός ότι το JSIM είναι αναπτυγμένο εξ ολοκλήρου σε JAVA του επιτρέπει να εκτελείται ανεξάρτητα από το λειτουργικό σύστημα και έτσι είναι δυνατή η χρήση του από πάρα πολλά συστήματα. Το μόνο που χρειάζεται είναι ένα περιβάλλον εκτέλεσης JAVA. Ενδεικτικά αναφέρουμε σε ποια λειτουργικά μπορεί να χρησιμοποιηθεί:

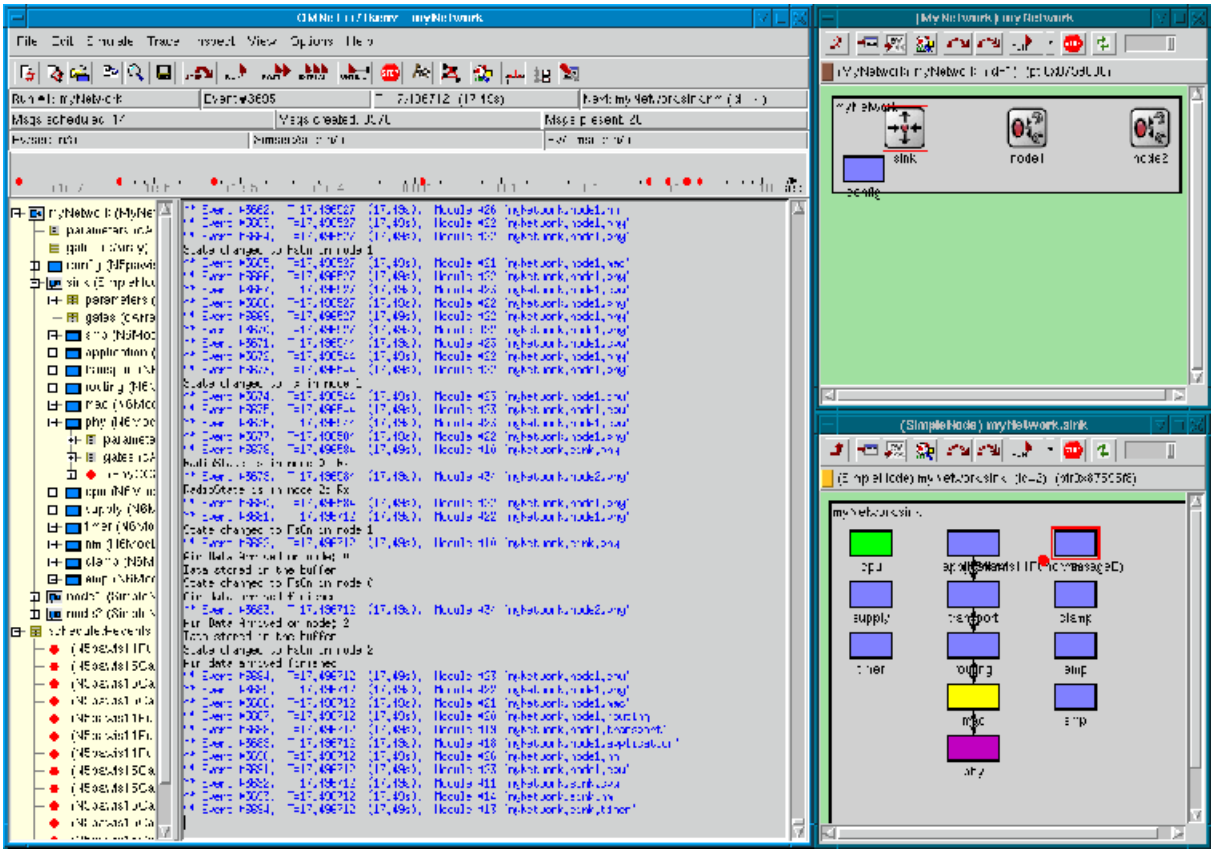
- Solaris, OS/2, eComStation, Windows XP, Linux

Ο προσομοιωτής είναι γραμμένος ώστε να μπορεί να χρησιμοποιηθεί σε σταθμούς εργασίας με πολλούς επεξεργαστές αν και οι δημιουργοί του

υποστηρίζουν πως τρέχει ικανοποιητικά σε συστήματα με χαμηλές δυνατότητες (JSIM Official, 2006).

3.4 Προσομοιωτής OMNET++

Το OMNET++ (OMNET, 2011) είναι ένα πακέτο προσομοίωσης διακριτών γεγονότων βασισμένο στη γλώσσα C++ και αναπτύχθηκε για ακαδημαϊκούς σκοπούς στο Τεχνικό Πανεπιστήμιο της Βουδαπέστης από τον Andras Varga (Varga, 2011). Αρχικός τομέας εφαρμογών του OMNET++ είναι η προσομοίωση δικτύων υπολογιστών και άλλων κατανεμημένων συστημάτων. Το αρκτικόλεξο OMNET++ σημαίνει Objective Modular Network Test-bed in C++.



Εικόνα 28: OMNET++

Είναι ευρέως γνωστό ότι αυτό το εργαλείο προσομοίωσης είναι κατάλληλο για την προσομοίωση σύνθετων συστημάτων όπως τους κόμβους Διαδικτύου. Το OMNET++ αν και μπορεί από μόνο του να χρησιμοποιηθεί για

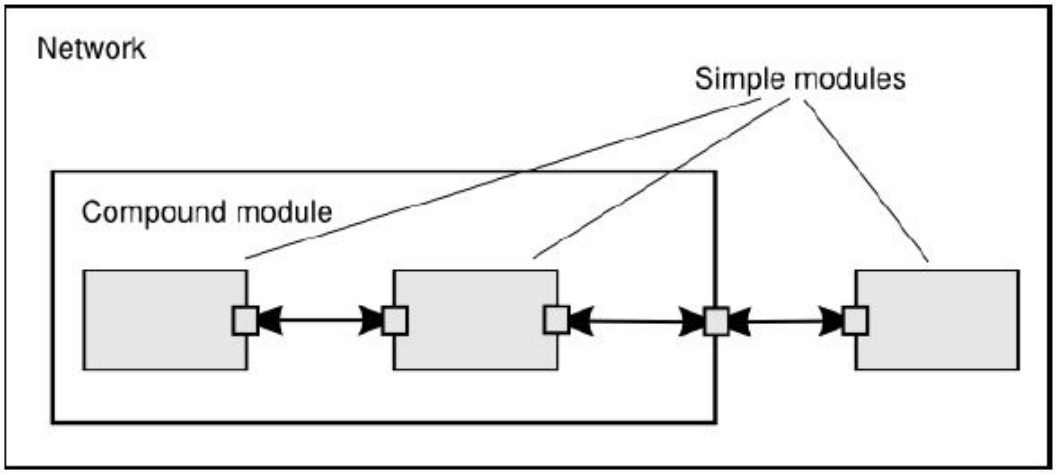
εφαρμογές προσομοίωσης χρησιμοποιείται ευρέως ως πλατφόρμα πάνω στην οποία έχουν αναπτυχθεί άλλα προγράμματα προσομοίωσης που αφορούν σε συγκεκριμένες κατηγορίες και τύπους δικτύων.

Ο προσομοιωτής OMNET++ μπορεί να χρησιμοποιηθεί για:

1) Μοντελοποίηση κυκλοφορίας των δικτύων τηλεπικοινωνιών
2) Μοντελοποίηση κυκλοφορίας των δικτύων τηλεπικοινωνιών
3) Μοντελοποίηση πρωτοκόλλων
4) Μοντελοποίηση δικτύων ουρών αναμονής
5) Μοντελοποίηση πολυεπεξεργαστών και άλλων κατανεμημένων hardware συστημάτων
6) Επικύρωση αρχιτεκτονικών υλικού
7) Αξιολόγηση των πτυχών απόδοσης των σύνθετων συστημάτων λογισμικού
8) Μοντελοποίηση οποιουδήποτε άλλου συστήματος όπου η προσέγγιση διακριτών γεγονότων είναι κατάλληλη.

Το OMNET++ έχει πολύ καλό γραφικό περιβάλλον, το GNED, το οποίο υπάρχει ελεύθερο στο Διαδίκτυο ώστε να μπορεί ο οποιοσδήποτε χρήστης να το κατεβάσει και να το τρέξει στον υπολογιστή του.

Το OMNET++ αποτελείται από απλές και σύνθετες ενότητες που οι λειτουργίες τους περιγράφονται με τη βοήθεια της γλώσσας προγραμματισμού C++. Δηλαδή υπάρχουν βασικές ενότητες και υπο-ενότητες από τις οποίες οικοδομούνται οι βασικές ενότητες, οι οποίες χρησιμοποιούν την γλώσσα υψηλού επιπέδου NED. Οι ενότητες επικοινωνούν μεταξύ τους μέσω μηνυμάτων τα οποία μπορούν να περιέχουν και δεδομένα. Διαγραμματικά, οι απλές και οι σύνθετες ενότητες απεικονίζονται στην παρακάτω εικόνα:



Εικόνα 29: Ενότητες στο OMNET++

Ένα μοντέλο στο OMNET++ αποτελείται από τα ακόλουθα μέρη:

- Τη γλώσσα NED (.ned αρχεία) που περιγράφει την τοπολογία, την δομή του μοντέλου με τις παραμέτρους, τις πύλες κ.τ.λ. Τα αρχεία .ned μπορεί να γραφτούν σε οποιονδήποτε επεξεργαστή κειμένου ή με τον GNED γραφικό επεξεργαστή.
- Τους ορισμούς των μηνυμάτων (msg. αρχεία). Μπορεί ο προγραμματιστής να ορίσει διάφορους τύπους μηνυμάτων και να προσθέσει πεδία δεδομένων σε αυτά. Το OMNET++ θα μεταφράσει τους ορισμούς των μηνυμάτων σε πλήρως δομημένες κλάσεις της C++.
- Τα αρχεία από τις απλές ενότητες. Είναι αρχεία C++ με κατάληξη .h/.cc .

Το σύστημα προσομοίωσης αποτελείται από τα ακόλουθα μέρη:

- Ο πυρήνας προσομοίωσης.

  Αυτός περιέχει τον κώδικα που διαχειρίζεται την προσομοίωση και την βιβλιοθήκη της κλάσης προσομοίωσης. Είναι γραμμένος σε C++ και μαζί με την μεταγλώττιση δημιουργεί μια βιβλιοθήκη (ένα αρχείο με κατάληξη .a ή .lib).

- Διεπαφή με τον χρήστη

  Το OMNET++ παρέχει διεπαφή με τον χρήστη που χρησιμοποιείται κατά την εκτέλεση της προσομοίωσης για να εξηγήσει τον έλεγχο των λαθών, για επίδειξη των αποτελεσμάτων ή

για παράλληλη εκτέλεση προσομοιώσεων. Είναι γραμμένη σε C++ και μαζί με την μεταγλώττιση δημιουργεί μια βιβλιοθήκη (ένα αρχείο με κατάληξη .a ή .lib).

Το αποτέλεσμα της προσομοίωσης γράφεται σε αρχεία εξόδου, σε ενδιάμεσα αρχεία εξόδου, σε κλιμακωτά αρχεία εξόδου, και ενδεχομένως σε αρχεία εξόδου του χρήστη. Το OMNET++ παρέχει ένα ξεχωριστό εργαλείο GUI που ονομάζεται Plove για να βλέπει και να σχεδιάζει το περιεχόμενο των αρχείων εξόδου των αποτελεσμάτων της προσομοίωσης.

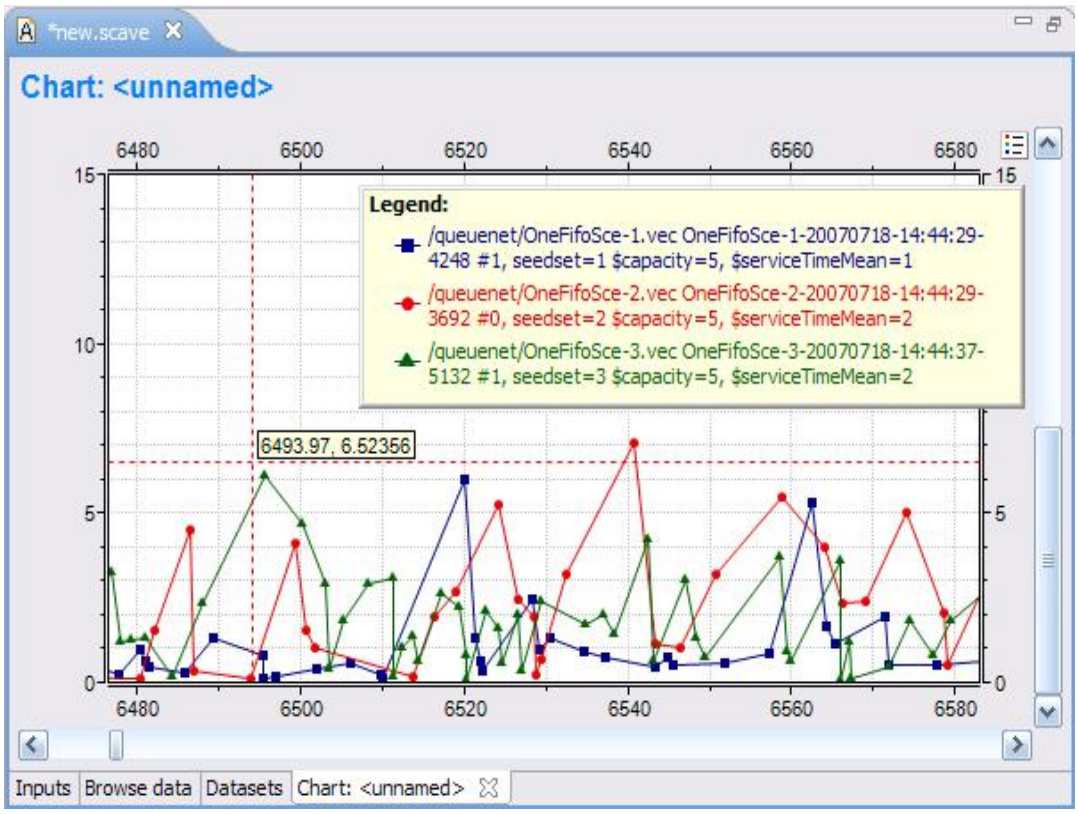
Ο αρχικός σκοπός αυτής της διεπαφής με τον χρήστη είναι να καταστούν τα εσωτερικά στοιχεία του μοντέλου ορατά στο χρήστη, να ελεγχθεί η εκτέλεση προσομοίωσης και να επιτραπεί ενδεχομένως στον χρήστη να επέμβει με την αλλαγή των μεταβλητών και των αντικειμένων μέσα στο μοντέλο. Αυτό είναι πολύ σημαντικό στη φάση της ανάπτυξης και διόρθωσης του προγράμματος προσομοίωσης. Εξίσου σημαντικό είναι και το γεγονός ότι ο χρήστης αποκτά μια οπτική εμπειρία που τον διευκολύνει να πάρει μια αίσθηση της συμπεριφοράς του μοντέλου. Το γραφικό ενδιάμεσο με τον χρήστη μπορεί επίσης να χρησιμοποιηθεί για να καταδείξει τη λειτουργία ενός προτύπου.

Δίνονται συνοπτικά τα βήματα που ακολουθούνται για το τρέξιμο μίας προσομοίωσης (OMNET++, 2011):

1. Ορισμός ιεραρχίας ενοτήτων
2. Ορισμός δομής του μοντέλου με χρήση της γλώσσας NED (είναι δυνατή η επεξεργασία από ένα πρόγραμμα επεξεργασίας κειμένου ή μέσω GNED)
3. Οι απλές ενότητες πρέπει να προγραμματιστούν με χρήση της C++
4. Χρησιμοποίηση ενός αρχείου omnetpp.ini για κράτηση παραμέτρων και ρυθμίσεων μοντέλου
5. Δημιουργία του προγράμματος προσομοίωσης και τρέξιμο
6. Εξαγωγή αποτελεσμάτων σε αρχεία output vector (.vec) και output scalar (.sca) (για την δυνατότητα καταγραφής των αποτελεσμάτων πρέπει να γίνει η κατάλληλη ρύθμιση)

Το ίδιο μοντέλο προσομοίωσης μπορεί να εκτελεσθεί με διαφορετική διεπαφή με τον χρήστη, χωρίς οποιαδήποτε αλλαγή στα αρχεία του μοντέλου από τον ίδιο. Ο χρήστης θα μπορεί να εξετάζει και να διορθώνει την προσομοίωση με ένα ισχυρό γραφικό ενδιάμεσο με τον χρήστη, και τελικά να την τρέχει με ένα απλό και γρήγορο γραφικό περιβάλλον που υποστηρίζει εκτέλεση batch.

Για την ανάλυση των αποτελεσμάτων χρησιμοποιείται το εργαλείο Scave και η λειτουργία του είναι να βοηθήσει τον χρήστη στην αποτύπωση των αποτελεσμάτων από τα αρχεία Vector και Scalar αντίστοιχα. Το Scave είναι σχεδιασμένο έτσι ώστε ο χρήστης να μπορεί να επεξεργαστεί τα αποτελέσματα από μία προσομοίωση (ένα ή δύο αρχεία), όσο και από ένα πλήθος προσομοιώσεων (πιθανόν εκατοντάδες αρχεία). Παρακάτω δίνεται μια εικόνα του Scave.



Εικόνα 30: Εργαλείο ανάλυσης Scave

Εργαλεία και βοηθητικά προγράμματα που είναι διαθέσιμα για τον OMNET++:

- iNoc
- HTTP Tools
- Google Earth Demo
- Ethernet Passive Optical Network
- VoIP Tool for INET
- Energy Framework

- Queues

καθώς και πλήθος άλλων τα οποία ο χρήστης μπορεί να κατεβάσει από τον επίσημο δικτυακό τόπο του ONET++.

Ο OMNET++ είναι διαθέσιμος για όλα τα δημοφιλή λειτουργικά συστήματα, όπως:

- MS Windows (WIN2K, XP, Vista, 7), Linux, Mac OS X και άλλα λειτουργικά βασισμένα στο UNIX

3.5 Σύγκριση προσομοιωτών

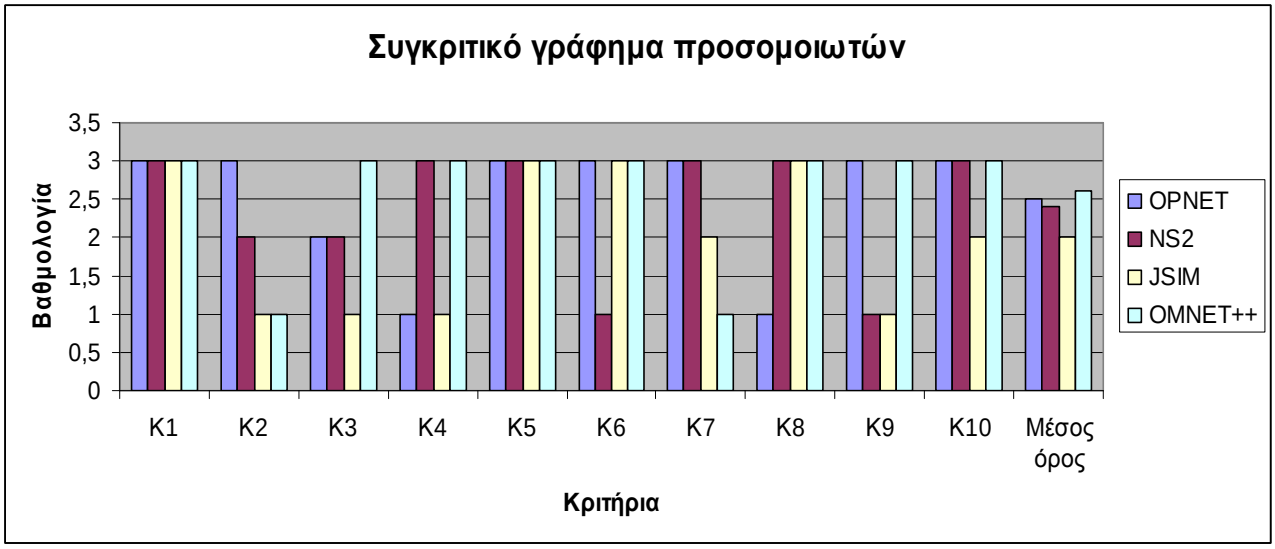
Στον παρακάτω πίνακα παρουσιάζεται μια σύγκριση των τεσσάρων προσομοιωτών που αναλύσαμε στις προηγούμενες ενότητες βασισμένη σε κριτήρια που βρέθηκαν σε αντίστοιχες μελέτες στη βιβλιογραφία (Lucio et al., 2003; Nicol, 2003; Lessmann et al., 2008; Begg et al., 2006; Duflos et al., 2006; Fleury et al., 2003). Η βαθμολογία που δίνεται για κάθε κριτήριο ακολουθεί την παρακάτω κλίμακα:

- 1: Ελλιπής
- 2: Μέτριο
- 3: Επαρκής

Πίνακας 4: Συγκριτικός πίνακας προσομοιωτών

Κριτήριο	OPNET	NS2	JSIM	OMNET++
Μοντέλα από διάφορες αρχιτεκτονικές δικτύων	3	3	3	3
Απόδοση	3	2	1	1
Αξιοπιστία προσομοιώσεων	2	2	1	3
Ανάλυση των δεδομένων προσομοίωσης	1	3	1	3
Δυνατότητα επέκτασης των μοντέλων προσομοίωσης	3	3	3	3
Γραφική διεπιφάνεια χρήστη	3	1	3	3
Εγχειρίδιο χρήσης και αρχεία βοήθειας	3	3	2	1
Κόστος	1	3	3	3
Δυνατότητα προσομοίωσης πολύ μεγάλων δικτύων	3	1	1	3
Δυνατότητα παράλληλης εξομοίωσης	3	3	2	3
Μέσος όρος	2,5	2,4	2	2,6

Τα αποτελέσματα της σύγκρισης παρουσιάζονται και διαγραμματικά στο παρακάτω γράφημα:



Εικόνα 31: Συγκριτικό γράφημα προσομοιωτών

Από την παραπάνω σύγκριση μπορούμε να εξάγουμε το συμπέρασμα ότι ο προσομοιωτής OMNET++ είναι καλύτερος σε μερικά σημεία από ότι οι άλλοι προσομοιωτές, όπως η αξιοπιστία των προσομοιώσεων (με γενικά κριτήρια), ενώ ο JSIM είναι ο χειρότερος προσομοιωτής στα περισσότερα κριτήρια, όπως το ότι δεν προσφέρει την δυνατότητα προσομοίωσης πολύ μεγάλων δικτύων (με γενικά κριτήρια). Βέβαια, αυτό δε σημαίνει ότι ο OMNET++ είναι καλύτερος για την προσομοίωση οποιουδήποτε δικτύου. Γίνεται σαφές ότι κάθε διαφορετικό σενάριο απαιτεί διαφορετική προσέγγιση και για αυτό ο κάθε χρήστης πρέπει να ελέγξει ενδελεχώς τις δυνατότητες του κάθε προσομοιωτή και να επιλέξει αυτόν που ικανοποιεί τις ανάγκες τους.

Όπως έχει ήδη διατυπωθεί, με την κατάλληλη παραμετροποίηση και καθορισμό των διάφορων λειτουργιών και συναρτήσεων τα αποτελέσματα μπορεί να αλλάξουν.

Ίσως το περισσότερο διαδεδομένο εργαλείο προσομοίωσης, από αυτά που περιγράψαμε, να θεωρείται από πολλούς το NS2, το οποίο παρέχει μια μεγάλη γκάμα τοπολογιών, πρωτοκόλλων και εφαρμογών που μπορεί να προσομοιώσει. Εξίσου διαδεδομένο είναι και το OMNET++ διότι εκτός του ότι από μόνο του μπορεί να χρησιμοποιηθεί για οποιαδήποτε προσομοίωση, έχουν αναπτυχθεί με βάση αυτό και άλλα εργαλεία τα οποία εξειδικεύονται σε τοπολογίες ή συγκεκριμένα πρωτόκολλα δικτύων και τα οποία στη συνέχεια μπορούν να συνδυαστούν άψογα με οποιαδήποτε άλλη εφαρμογή έχει αναπτυχθεί σε

OMNET++. Το OPNET παρόλο που είναι ένα πολύ ισχυρό εργαλείο παρέχεται δωρεάν μόνο για εκπαιδευτικούς σκοπούς και το JSIM μολονότι απαιτεί λιγότερο χρόνο εκμάθησής περιορίζεται από κάποιες από τις δυνατότητες του.

ΚΕΦΑΛΑΙΟ 4: Προσομοίωση με χρήση του λογισμικού OMNET++

Στο κεφάλαιο αυτό θα παρουσιαστεί η προσομοίωση που θα γίνει με τον προσομοιωτή OMNET++. Θα πραγματοποιηθούν μερικές απλές και μερικές πιο σύνθετες προσομοιώσεις.

4.1 Προσομοίωση με δύο κόμβους

Στο αρχικό μας παράδειγμα δημιουργούμε ένα δίκτυο που θα αποτελείται από δύο κόμβους. Οι κόμβοι αυτοί θα ανταλλάσουν μεταξύ τους ένα πακέτο. Για τη δημιουργία της προσομοίωσης αυτής πρέπει αρχικά να δημιουργήσουμε ένα αρχείο που θα περιγράφει την τοπολογία του δικτύου. Στο αρχείο αυτό θα ορίσουμε ουσιαστικά τους κόμβους του δικτύου και τις συνδέσεις μεταξύ αυτών. Το αρχείο package.ned παρατίθεται παρακάτω:

Πίνακας 5: Αρχείο package.ned 1ης προσομοίωσης

```
package my_project;

//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
```

```
// along with this program.  If not, see http://www.gnu.org/licenses/.
//

package my_project;

simple Txc1
{
    gates:
        input in;
        output out;
}

//
// Δύο στιγμιότυπα Txcl συνδεδεμένα μεταξύ τους αμφίδρομα.
// Ο κόμβος 1 και ο κόμβος 2 ανταλλάσσουν ένα μήνυμα συνεχώς.
//
network Simple1
{
    submodules:
        node1: Txc1;  //ο πρώτος κόμβος
        node2: Txc1;  //ο δεύτερος κόμβος
    connections:
        //δημιουργία αμφίδρομης σύνδεσης
        node1.out -->
        {
            delay = 100ms;
        } --> node2.in;

        node1.in <--
```

```
    {
      delay = 100ms;
    } <-- node2.out;
}
```

Το δίκτυο Simple1 είναι ένα δίκτυο που περιέχει δύο submodules, τα node1 και node2. Τα node1 και node2 είναι στιγμιότυπα του ίδιου module που ονομάζεται Txcl. Οι δύο κόμβοι συνδέονται αμφίδρομα μεταξύ τους με μία καθυστέρηση μετάδοσης 100 ms. Το Txcl είναι ένα απλό module που διαθέτει μία είσοδο και μία έξοδο.

Στη συνέχεια πρέπει να υλοποιήσουμε τη λειτουργικότητα του απλού module Txcl. Το αρχείο Txcl.cc παρατίθεται παρακάτω:

Πίνακας 6: Αρχείο Txcl.cc 1ης προσομοίωσης

```
#include <string.h>
#include <omnetpp.h>

//δήλωση κλάσης - κληρονομεί το cSimpleModule
class Txc1 : public cSimpleModule
{
        protected:
                virtual void initialize();
                virtual void handleMessage(cMessage *msg);
};

//δήλωση του module
Define_Module(Txc1);

void Txc1::initialize()
```

```
{

        // Η μέθοδος αυτή καλείται στην αρχή της προσομοίωσης.
        // Κάποιος από τους δύο κόμβους πρέπει να αποστείλει το
        // πρώτο μήνυμα. Αυτός ο κόμβος είναι ο node1.

   // Εύρεση του κόμβου node1
   if (strcmp("node1", getName()) == 0)
   {
        // δημιουργεί και αποστέλλει το πρώτο μήνυμα στη θύρα out.
        // Το μήνυμα είναι το Hello World
      cMessage *msg = new cMessage("Hello World");
      send(msg, "out");
   }
}

void Txc1::handleMessage(cMessage *msg)
{
        // Η μέθοδος αυτή καλείται όταν ένα μήνυμα καταφτάνει σε κάποιο
        // module. Ουσιαστικά με τη μέθοδο αυτή μόλις φτάνει κάποιο μήνυμα
        // στέλνει πίσω ένα άλλο μήνυμα.
   send(msg, "out");
}
```

Η κλάση Txcl υλοποιεί το Txcl module που πρέπει να κληρονομεί την κλάση cSimpleModule και να εγγραφεί στο OMNeT++ με τη μακροεντολή Define_Module(). Οι δύο μέθοδοι που δηλώνονται ξανά στην κλάση Txcl είναι οι:

- initialize(): Η μέθοδος αυτή δημιουργεί ένα μήνυμα στον κόμβο 1 και το αποστέλλει στον κόμβο 2.

- handleMessage(): Μόλις ένα μήνυμα φτάνει σε κάποιον κόμβο, αυτός απαντάει με ένα άλλο μήνυμα.

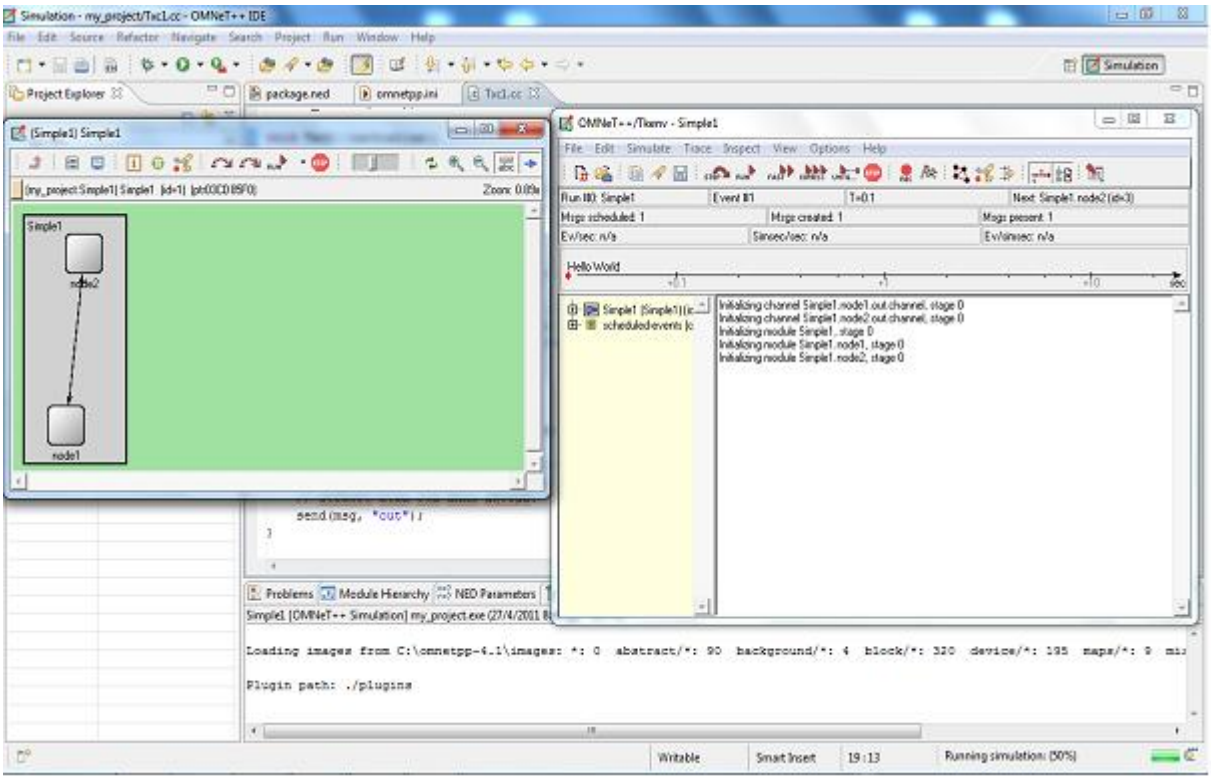
Οπότε γίνεται εύκολα κατανοητό ότι οι δύο κόμβοι θα ανταλλάσουν επ' άπειρον μηνύματα μεταξύ τους.

Κάθε αρχείο προσομοίωσης πρέπει να διαθέτει ένα αρχείο omnetpp.ini που να αρχικοποιεί το δίκτυο. Το αρχείο omnetpp.ini παρατίθεται παρακάτω:

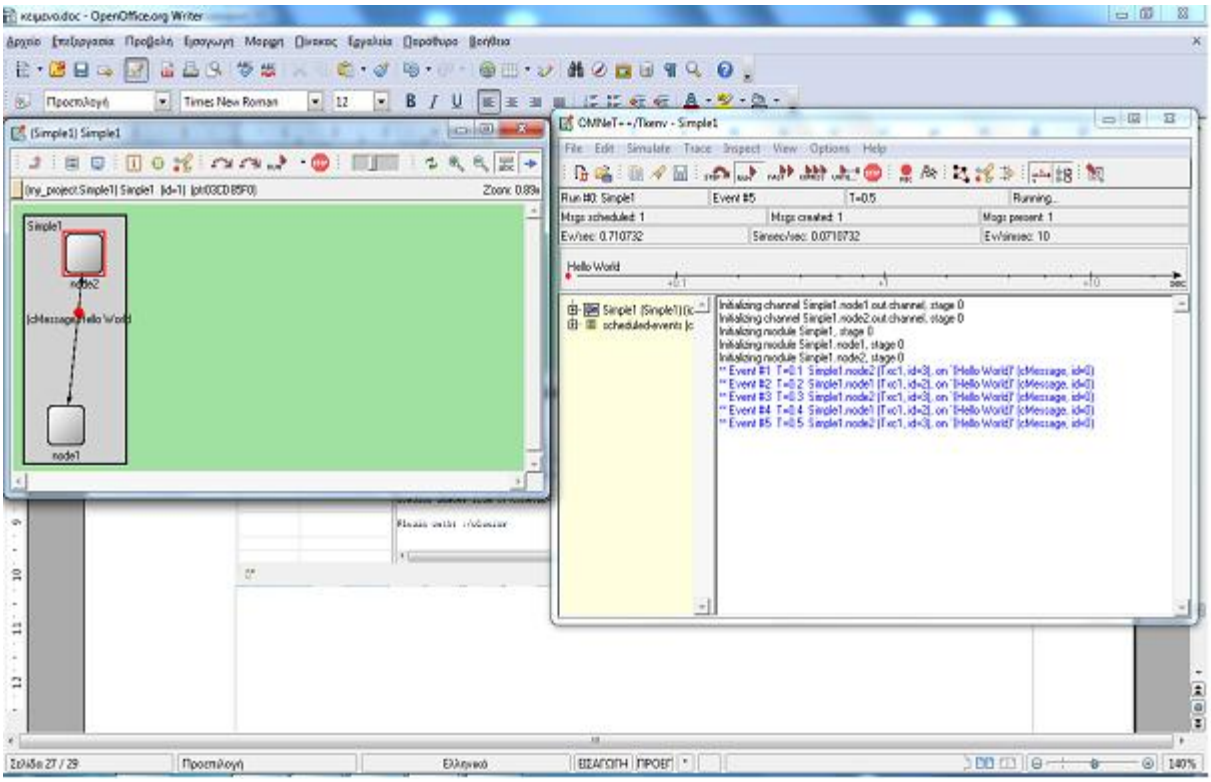
Πίνακας 7: Αρχείο omnetpp.ini 1ης προσομοίωσης

```
[General]

network = tictoc1
```

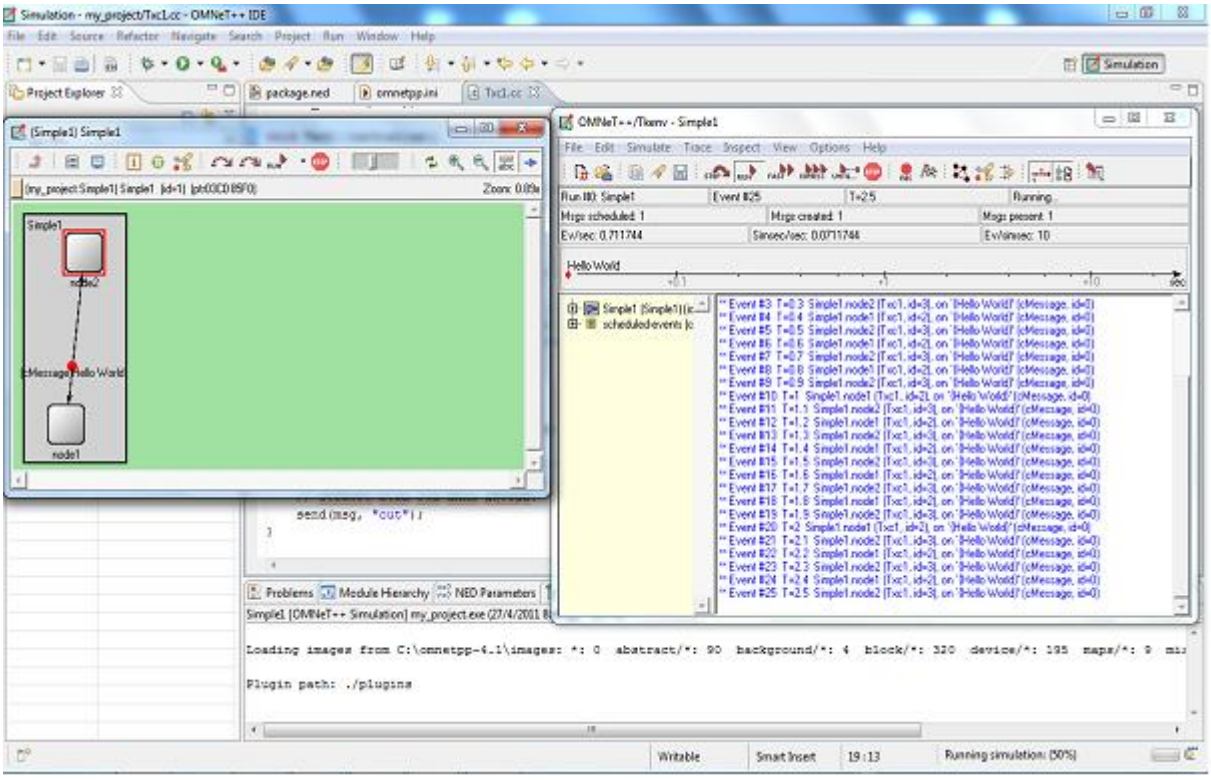
Στη συνέχεια μπορούμε να εκτελέσουμε το πρόγραμμα. Μερικά στιγμιότυπα από την εκτέλεση του φαίνεται στις παρακάτω εικόνες:



Εικόνα 32: Αρχικό στιγμιότυπο 1ης προσομοίωσης



Εικόνα 33: Ενδιάμεσο στιγμιότυπο 1ης προσομοίωσης



Εικόνα 34: Τελικό στιγμιότυπο 1ης προσομοίωσης

4.2 Περαιτέρω ανάπτυξη της προσομοίωσης με δύο κόμβους

Στο παράδειγμα αυτό θα μετατρέψουμε το προηγούμενο δίκτυο προσθέτοντας κάποιες παραμέτρους που θα κάνουν την προσομοίωση πιο ενδιαφέρουσα. Αρχικά, προσθέτουμε κάποια εικόνα στους δύο κόμβους για να γίνουν διακριτοί. Αυτό επιτυγχάνεται κάνοντας τις εξής αλλαγές στο αρχείο package.ned:

Πίνακας 8: Αρχείο package.ned 2ης προσομοίωσης

```
package my_project;


//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with this program.  If not, see http://www.gnu.org/licenses/.
//


//Κόμβος 1
simple Node1
{
   parameters:
      @display("i=block/routing");
```

```
    gates:
        input in;
        output out;
}

//Κόμβος 2
simple Node2
{
    parameters:
        @display("i=block/process");
    gates:
        input in;
        output out;
}



//
// Δίκτυο
//
network ExtensionSimple1
{
    submodules:
        //πρώτος κόμβος
        node1: Node1 {
            parameters:
                @display("i=,cyan");
        }

        //δεύτερος κόμβος
```

```
        node2: Node2 {
            parameters:
                @display("i=,gold");
        }
    connections:
        //δημιουργία συνδέσεων
        node1.out -->
        {
            delay = 100ms;
        } --> node2.in;

        node1.in <--
        {
            delay = 100ms;
        } <-- node2.out;
}
```

Επίσης, τροποποιούμε το αρχείο Txcl.cc εισάγοντας κάποια μηνύματα που θα βοηθήσουν στην αποσφαλμάτωση. Τα μηνύματα αυτά είναι τα εξής:

Πίνακας 9: Προσθήκη μηνυμάτων αποσφαλμάτωσης στο αρχείο Txl.cc

```
EV << "Sending initial message\n";

EV << "Received message `" << msg->getName() << "', sending it out again\n";
```

Επίσης, προσθέτουμε στο δίκτυο τυχαίες τιμές καθυστέρησης διάδοσης και δημιουργούμε ξεχωριστές κλάσεις για τους κόμβους 1 και 2, αντίστοιχα. Με αυτήν την προσθήκη θα φέρουμε το παράδειγμά μας πιο κοντά στα σημερινά πρωτόκολλα. Πάλι οι δύο κόμβοι θα εναλλάσσουν μεταξύ τους ένα μήνυμα, αλλά αυτή τη φορά ο κόμβος 2 μπορεί να χάσει το μήνυμα με κάποια

πιθανότητα και αν συμβεί αυτό, τότε ο κόμβος 1 πρέπει να το ξαναστείλει. Για την επίτευξη αυτού του στόχου, ο κόμβος 1 θα ξεκινάει ένα χρονοδιακόπτη, ο οποίος αν λήξει, τότε ξαναστέλνει το μήνυμα, ενώ αν ο κόμβος 2 στείλει απάντηση, τότε θα πρέπει να σταματάει ο χρονοδιακόπτης. Με όλες αυτές τις αλλαγές το αρχείο Txcl.cc παρατίθεται παρακάτω:

Πίνακας 10: Αρχείο Txcl.cc 2ης προσομοίωσης

```
#include <stdio.h>

#include <string.h>

#include <omnetpp.h>


//δήλωση κόμβου 1

class Node1 : public cSimpleModule

{

  private:

    simtime_t timeout;  // timeout

    cMessage *timeoutEvent;  // για να γνωρίζουμε αν συνέβη το timeout

    int seq;  // αριθμός μηνύματος

    cMessage *message;  // το μήνυμα που πρέπει να μεταδοθεί


  public:

    Node1();

    virtual ~Node1();


  protected:

    virtual cMessage *generateNewMessage();

    virtual void sendCopyOf(cMessage *msg);

    virtual void initialize();

    virtual void handleMessage(cMessage *msg);

};
```

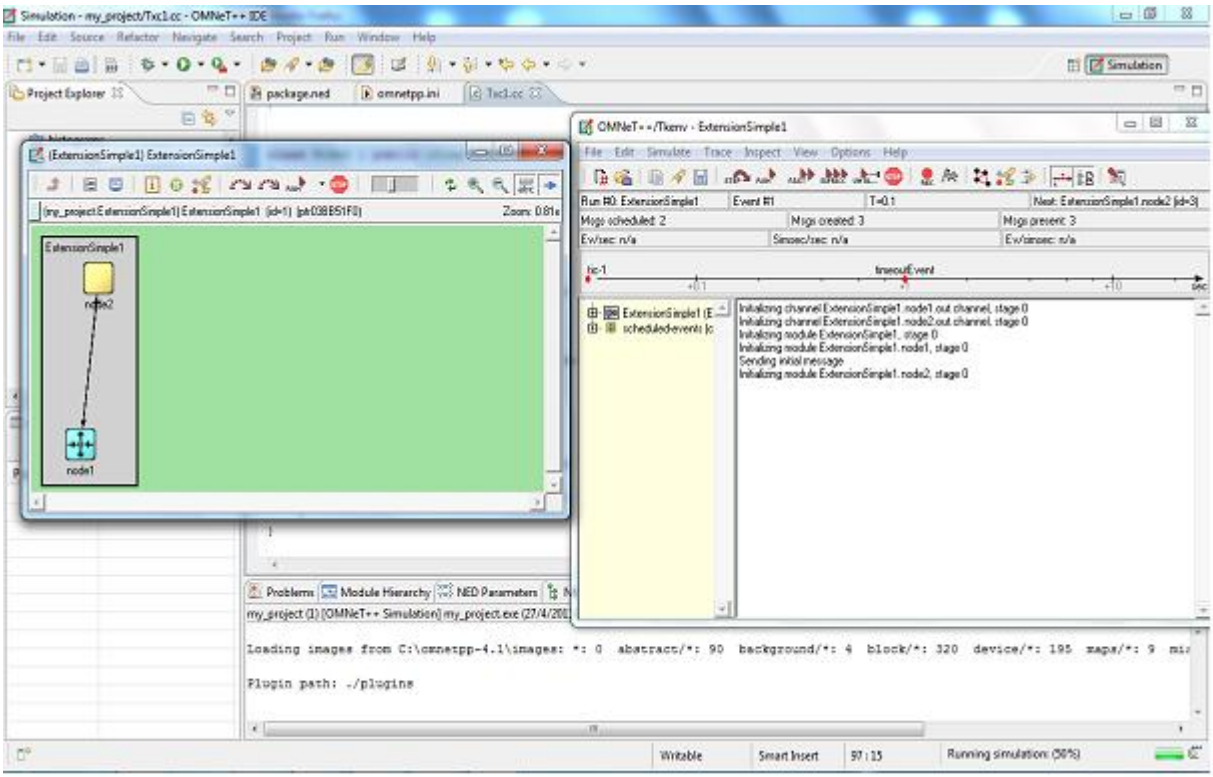
```
Define_Module(Node1);

Node1::Node1()
{
    timeoutEvent = message = NULL;
}

Node1::~Node1()
{
    cancelAndDelete(timeoutEvent);
    delete message;
}

void Node1::initialize()
{
    //Αρχικοποίηση μεταβλητών
    seq = 0;
    timeout = 1.0;
    timeoutEvent = new cMessage("timeoutEvent");

    // Δημιουργία αρχικού μηνύματος
    EV << "Sending initial message\n";
    message = generateNewMessage();
    sendCopyOf(message);
    scheduleAt(simTime()+timeout, timeoutEvent);
}

void Node1::handleMessage(cMessage *msg)
```

```
{
   if (msg==timeoutEvent)
   {
         //Αν συμβεί το timeout, τότε το πακέτο δεν έχει φτάσει μέσα
         //στο χρόνο, οπότε το μεταδίδουμε πάλι
      EV << "Timeout expired, resending message and restarting timer\n";
      sendCopyOf(message);
      scheduleAt(simTime()+timeout, timeoutEvent);
   }
   else // το μήνυμα έφτασε μέσα στο χρόνο
   {
      EV << "Received: " << msg->getName() << "\n";
      delete msg;

      //διαγράφουμε το αποθηκευμένο μήνυμα και μηδενίζουμε το χρονοδιακόπτη
      EV << "Timer cancelled.\n";
      cancelEvent(timeoutEvent);
      delete message;

      // Στέλνουμε το επόμενο μήνυμα
      message = generateNewMessage();
      sendCopyOf(message);
      scheduleAt(simTime()+timeout, timeoutEvent);
   }
}

cMessage *Node1::generateNewMessage()
{
   // Δημιουργούμε ένα νέο μήνυμα με διαφορετικό σφραγίδα χρόνου
```

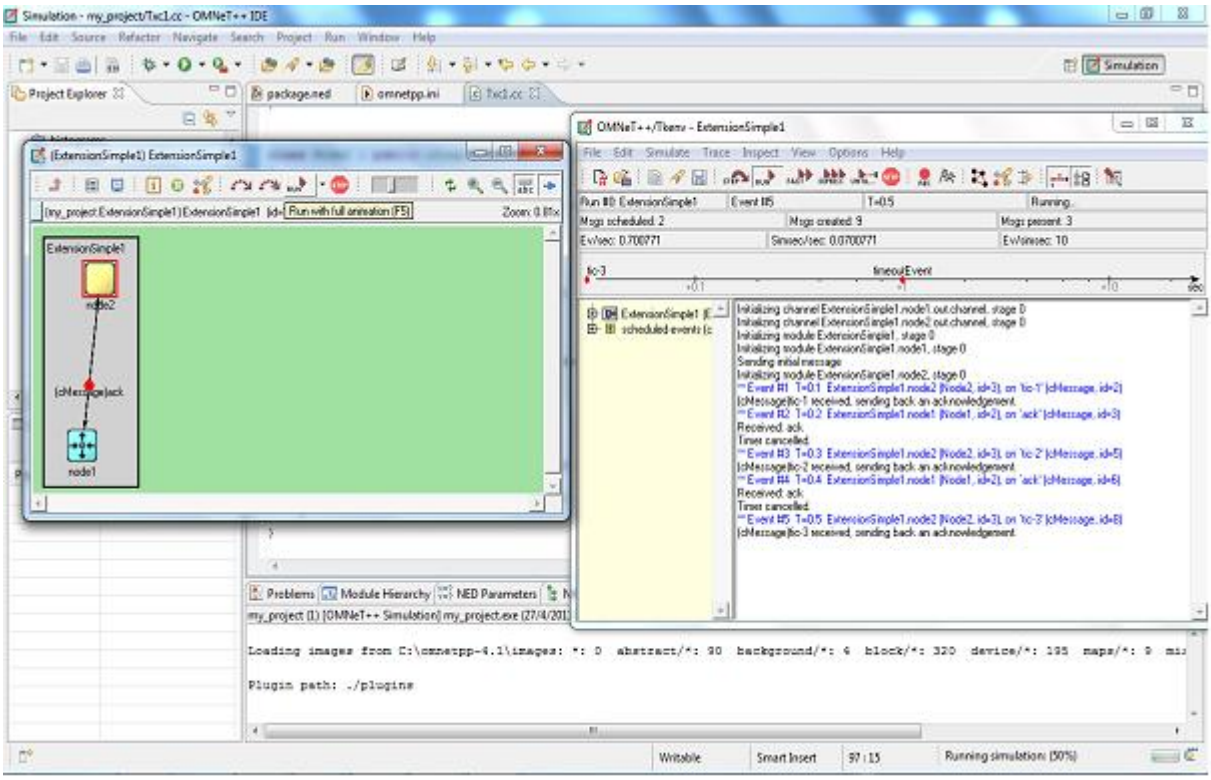
```
    char msgname[20];
    sprintf(msgname, "tic-%d", ++seq);
    cMessage *msg = new cMessage(msgname);
    return msg;
}

void Node1::sendCopyOf(cMessage *msg)
{
    // Αντιγράφουμε το μήνυμα και το ξαναστέλνουμε
    cMessage *copy = (cMessage *) msg->dup();
    send(copy, "out");
}


class Node2 : public cSimpleModule
{
  protected:
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Node2);

void Node2::handleMessage(cMessage *msg)
{
    if (uniform(0,1) < 0.1)
    {
        EV << "\"Losing\" message " << msg << endl;
        bubble("message lost");
        delete msg;
```

```
    }
    else
    {
        EV << msg << " received, sending back an acknowledgement.\n";
        delete msg;
        send(new cMessage("ack"), "out");
    }
}
```

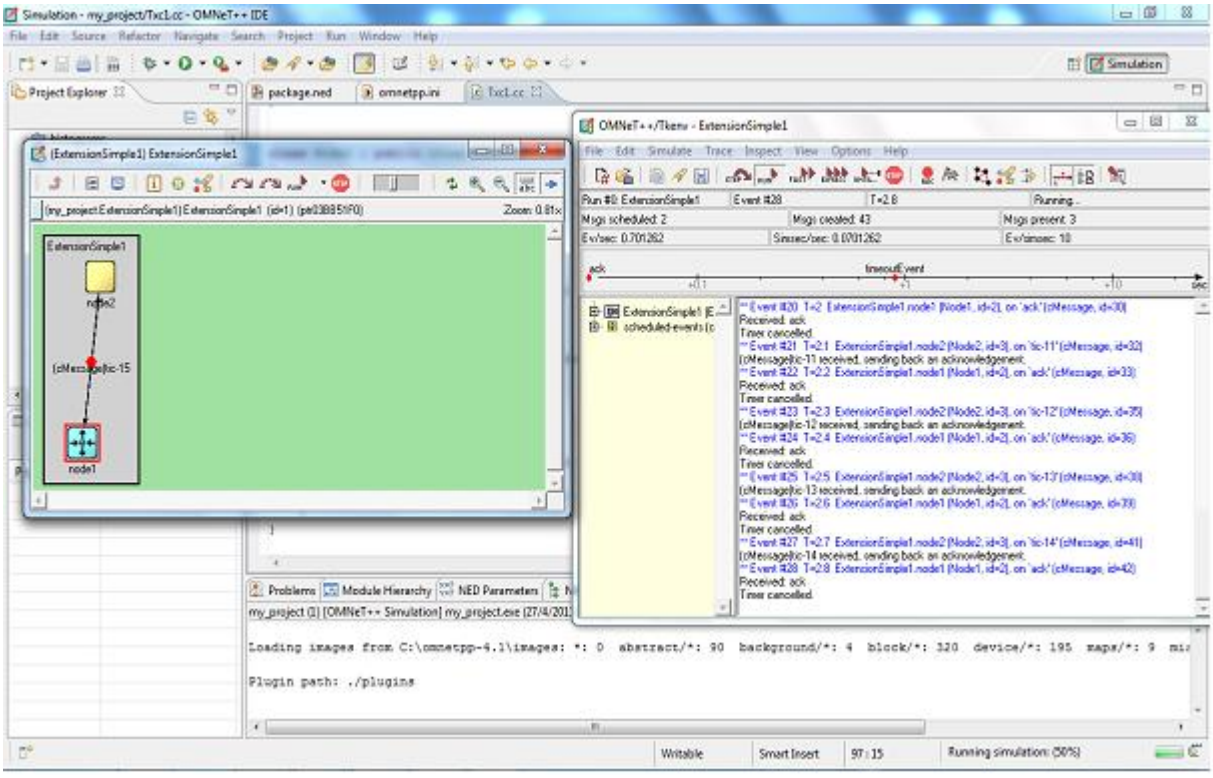
Στη συνέχεια μπορούμε να εκτελέσουμε το πρόγραμμα. Μερικά στιγμιότυπα από την εκτέλεση του φαίνονται στις παρακάτω εικόνες:



Εικόνα 35: Αρχικό στιγμιότυπο 2ης προσομοίωσης



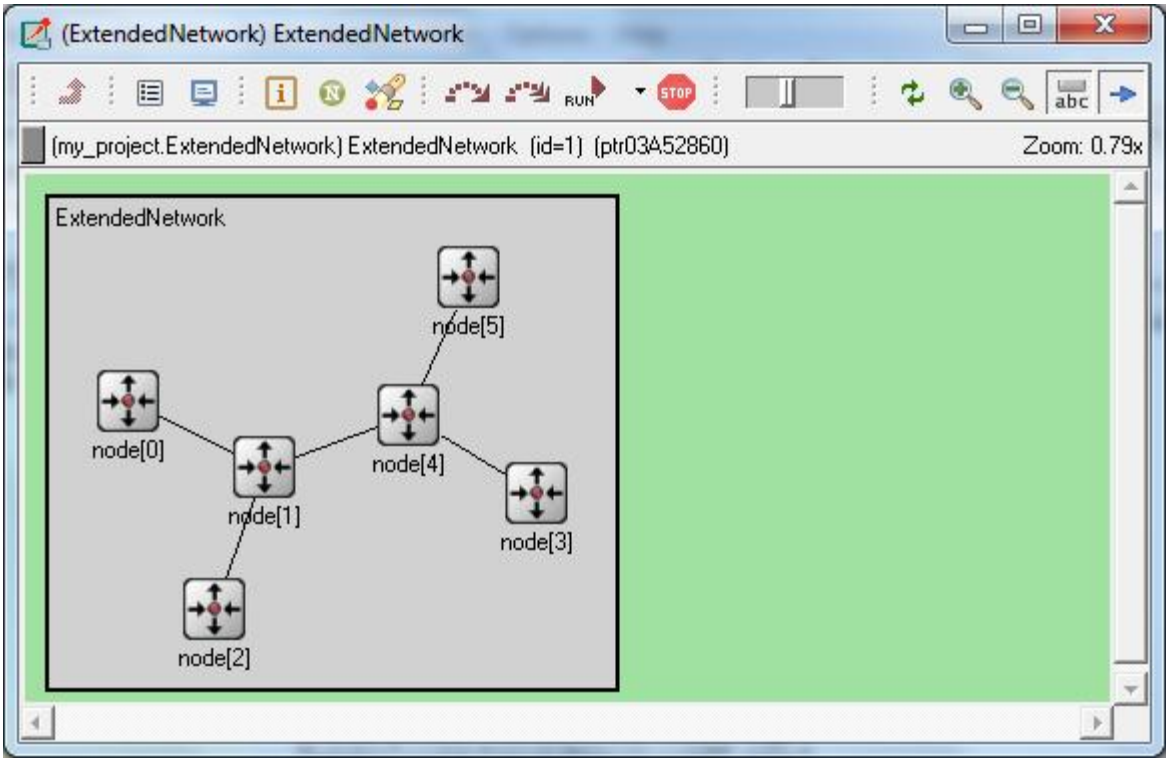
Εικόνα 36: Ενδιάμεσο στιγμιότυπο 2ης προσομοίωσης



Εικόνα 37: Τελικό στιγμιότυπο 2ης προσομοίωσης

4.3 Προσομοίωση με πολλούς κόμβους σε ένα πραγματικό δίκτυο

Στην προσομοίωση αυτή θα κατασκευάσουμε ένα δίκτυο με έξι κόμβους. Η τοπολογία του δικτύου φαίνεται στην παρακάτω εικόνα.



Εικόνα 38: Τοπολογία δικτύου 3ης προσομοίωσης

Όλοι οι κόμβοι συνδέονται αμφίδρομα μεταξύ τους. Για να το επιτύχουμε αυτό τροποποιήσαμε το αρχείο package.ned ως εξής:

Πίνακας 11: Αρχείο package.ned 3ης προσομοίωσης

```
package my_project;


//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
```

```
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with this program.  If not, see http://www.gnu.org/licenses/.
//

simple Txc1
{
    parameters:
        @display("i=block/routing");
    gates:
        inout gate[];
}

//
// Δήλωση δικτύου
//
network ExtendedNetwork
{
    types:
        channel Channel extends ned.DelayChannel
        {
            delay = 100ms;
        }
```

```
    submodules:
        node[6]: Txc1;
    connections:
        node[0].gate++ <--> Channel <--> node[1].gate++;
        node[1].gate++ <--> Channel <--> node[2].gate++;
        node[1].gate++ <--> Channel <--> node[4].gate++;
        node[3].gate++ <--> Channel <--> node[4].gate++;
        node[4].gate++ <--> Channel <--> node[5].gate++;
}
```

Σε αυτήν την προσομοίωση ο κόμβος node[0] θα δημιουργήσει το μήνυμα το οποίο θα στέλνεται στο δίκτυο. Αυτό γίνεται στη μέθοδο initialize() με τη βοήθεια της ενσωματωμένης μεθόδου getIndex() που επιστρέφει το δείκτη του module στον πίνακα. Ο κώδικας της μεθόδου αυτής παρατίθεται παρακάτω:

Πίνακας 12: Μέθοδος initialize

```
void Txc1::initialize()
{
    // Το Module 0 στέλνει το 1ο μήνυμα
    if (getIndex()==0)
    {
            // Ξεκινά τη διαδικασία στέλντοντας το μήνυμα στο εαυτό του
        Msg *msg = generateMessage();
        scheduleAt(0.0, msg);
    }
}
```

Η μέθοδος forwardMessage() καλείται από τη μέθοδο handleMessage() όταν καταφτάνει ένα μήνυμα στον κόμβο. Η μέθοδος αυτή επιλέγει μια τυχαία

θύρα και στέλνει το μήνυμα στη θύρα αυτή. Ο κώδικας της μεθόδου forwardMessage() παρατίθεται παρακάτω:

Πίνακας 13: Μέθοδος forwardMessage

```
void Txc1::forwardMessage(Msg *msg)
{
   //Αύξηση του μετρητή αλμάτων
   msg->setHopCount(msg->getHopCount()+1);

   //Επιλογή τυχαία θύρα
   int n = gateSize("gate");
   int k = intuniform(0,n-1);

   EV << "Forwarding message " << msg << " on gate[" << k << "]\n";
   send(msg, "gate$o", k);
}
```

Επίσης, δημιουργούμε μία κλάση τύπου message μέσω του οδηγού του OMNet. Το αρχείο Msg.msg παρατίθεται παρακάτω:

Πίνακας 14: Αρχείο Msg.msg 3ης προσομοίωσης

```
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with this program.  If not, see http://www.gnu.org/licenses/.
//


//
// TODO generated message class
//
message Msg
{
    int source;
    int destination;
    int hopCount = 0;
}
```

Κατά τη μεταγλώττιση του προγράμματος δημιουργούνται αυτόματα από τον μεταγλωττιστή τα απαραίτητα αρχεία C++, Msg_m.cc και Msg_h.cc. Αρχικά, παρατίθεται ο κώδικας του αρχείου Msg_m.cc:

Πίνακας 15: Αρχείο Msg_m.cc

```
//
// Generated file, do not edit! Created by opp_msgc 4.1 from Msg.msg.
//


// Disable warnings about unused variables, empty switch stmts, etc:
#ifdef _MSC_VER
#  pragma warning(disable:4101)
```

```
#  pragma warning(disable:4065)
#endif

#include <iostream>
#include <sstream>
#include "Msg_m.h"

// Template rule which fires if a struct or class doesn't have operator<<
template<typename T>
std::ostream& operator<<(std::ostream& out,const T&) {return out;}

// Another default rule (prevents compiler from choosing base class' doPacking())
template<typename T>
void doPacking(cCommBuffer *, T& t) {
   throw cRuntimeError("Parsim error: no doPacking() function for type %s or its base class (check .msg and _m.cc/h files!)",opp_typename(typeid(t)));
}

template<typename T>
void doUnpacking(cCommBuffer *, T& t) {
   throw cRuntimeError("Parsim error: no doUnpacking() function for type %s or its base class (check .msg and _m.cc/h files!)",opp_typename(typeid(t)));
}



Register_Class(Msg);

Msg::Msg(const char *name, int kind) : cMessage(name,kind)
```

```
{
    this->source_var = 0;
    this->destination_var = 0;
    this->hopCount_var = 0;
}

Msg::Msg(const Msg& other) : cMessage()
{
    setName(other.getName());
    operator=(other);
}

Msg::~Msg()
{
}

Msg& Msg::operator=(const Msg& other)
{
    if (this==&other) return *this;
    cMessage::operator=(other);
    this->source_var = other.source_var;
    this->destination_var = other.destination_var;
    this->hopCount_var = other.hopCount_var;
    return *this;
}

void Msg::parsimPack(cCommBuffer *b)
{
    cMessage::parsimPack(b);
```

```
    doPacking(b,this->source_var);
    doPacking(b,this->destination_var);
    doPacking(b,this->hopCount_var);
}

void Msg::parsimUnpack(cCommBuffer *b)
{
    cMessage::parsimUnpack(b);
    doUnpacking(b,this->source_var);
    doUnpacking(b,this->destination_var);
    doUnpacking(b,this->hopCount_var);
}

int Msg::getSource() const
{
    return source_var;
}

void Msg::setSource(int source_var)
{
    this->source_var = source_var;
}

int Msg::getDestination() const
{
    return destination_var;
}

void Msg::setDestination(int destination_var)
```

```
{
    this->destination_var = destination_var;
}

int Msg::getHopCount() const
{
    return hopCount_var;
}

void Msg::setHopCount(int hopCount_var)
{
    this->hopCount_var = hopCount_var;
}

class MsgDescriptor : public cClassDescriptor
{
  public:
    MsgDescriptor();
    virtual ~MsgDescriptor();

    virtual bool doesSupport(cObject *obj) const;
    virtual const char *getProperty(const char *propertyname) const;
    virtual int getFieldCount(void *object) const;
    virtual const char *getFieldName(void *object, int field) const;
    virtual int findField(void *object, const char *fieldName) const;
    virtual unsigned int getFieldTypeFlags(void *object, int field) const;
    virtual const char *getFieldTypeString(void *object, int field) const;
    virtual const char *getFieldProperty(void *object, int field, const char *propertyname) const;
    virtual int getArraySize(void *object, int field) const;
```

```
    virtual std::string getFieldAsString(void *object, int field, int i) const;
    virtual bool setFieldAsString(void *object, int field, int i, const char *value) const;

    virtual const char *getFieldStructName(void *object, int field) const;
    virtual void *getFieldStructPointer(void *object, int field, int i) const;
};

Register_ClassDescriptor(MsgDescriptor);

MsgDescriptor::MsgDescriptor() : cClassDescriptor("Msg", "cMessage")
{
}

MsgDescriptor::~MsgDescriptor()
{
}

bool MsgDescriptor::doesSupport(cObject *obj) const
{
    return dynamic_cast<Msg *>(obj)!=NULL;
}

const char *MsgDescriptor::getProperty(const char *propertyname) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    return basedesc ? basedesc->getProperty(propertyname) : NULL;
}
```

```
int MsgDescriptor::getFieldCount(void *object) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    return basedesc ? 3+basedesc->getFieldCount(object) : 3;
}

unsigned int MsgDescriptor::getFieldTypeFlags(void *object, int field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldTypeFlags(object, field);
        field -= basedesc->getFieldCount(object);
    }
    static unsigned int fieldTypeFlags[] = {
        FD_ISEDITABLE,
        FD_ISEDITABLE,
        FD_ISEDITABLE,
    };
    return (field>=0 && field<3) ? fieldTypeFlags[field] : 0;
}

const char *MsgDescriptor::getFieldName(void *object, int field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldName(object, field);
        field -= basedesc->getFieldCount(object);
```

```
    }
    static const char *fieldNames[] = {
        "source",
        "destination",
        "hopCount",
    };
    return (field>=0 && field<3) ? fieldNames[field] : NULL;
}

int MsgDescriptor::findField(void *object, const char *fieldName) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    int base = basedesc ? basedesc->getFieldCount(object) : 0;
    if (fieldName[0]=='s' && strcmp(fieldName, "source")==0) return base+0;
    if (fieldName[0]=='d' && strcmp(fieldName, "destination")==0) return base+1;
    if (fieldName[0]=='h' && strcmp(fieldName, "hopCount")==0) return base+2;
    return basedesc ? basedesc->findField(object, fieldName) : -1;
}

const char *MsgDescriptor::getFieldTypeString(void *object, int field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldTypeString(object, field);
        field -= basedesc->getFieldCount(object);
    }
    static const char *fieldTypeStrings[] = {
        "int",
```

```
        "int",
        "int",
    };
    return (field>=0 && field<3) ? fieldTypeStrings[field] : NULL;
}

const char *MsgDescriptor::getFieldProperty(void *object, int field, const char *propertyname) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldProperty(object, field, propertyname);
        field -= basedesc->getFieldCount(object);
    }
    switch (field) {
        default: return NULL;
    }
}

int MsgDescriptor::getArraySize(void *object, int field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getArraySize(object, field);
        field -= basedesc->getFieldCount(object);
    }
    Msg *pp = (Msg *)object; (void)pp;
    switch (field) {
```

```
        default: return 0;
    }
}

std::string MsgDescriptor::getFieldAsString(void *object, int field, int i) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldAsString(object,field,i);
        field -= basedesc->getFieldCount(object);
    }
    Msg *pp = (Msg *)object; (void)pp;
    switch (field) {
        case 0: return long2string(pp->getSource());
        case 1: return long2string(pp->getDestination());
        case 2: return long2string(pp->getHopCount());
        default: return "";
    }
}

bool MsgDescriptor::setFieldAsString(void *object, int field, int i, const char *value) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->setFieldAsString(object,field,i,value);
        field -= basedesc->getFieldCount(object);
    }
```

```
    Msg *pp = (Msg *)object; (void)pp;
    switch (field) {
        case 0: pp->setSource(string2long(value)); return true;
        case 1: pp->setDestination(string2long(value)); return true;
        case 2: pp->setHopCount(string2long(value)); return true;
        default: return false;
    }
}

const char *MsgDescriptor::getFieldStructName(void *object, int field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldStructName(object, field);
        field -= basedesc->getFieldCount(object);
    }
    static const char *fieldStructNames[] = {
        NULL,
        NULL,
        NULL,
    };
    return (field>=0 && field<3) ? fieldStructNames[field] : NULL;
}

void *MsgDescriptor::getFieldStructPointer(void *object, int field, int i) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
```

```
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldStructPointer(object, field, i);
        field -= basedesc->getFieldCount(object);
    }
    Msg *pp = (Msg *)object; (void)pp;
    switch (field) {
        default: return NULL;
    }
}
```

Στη συνέχεια παρατίθεται ο κώδικας του αρχείου Msg_m.h:

Πίνακας 16: Αρχείο Msg_m.h

```
//
// Generated file, do not edit! Created by opp_msgc 4.1 from Msg.msg.
//

#ifndef _MSG_M_H_
#define _MSG_M_H_

#include <omnetpp.h>

// opp_msgc version check
#define MSGC_VERSION 0x0401
#if (MSGC_VERSION!=OMNETPP_VERSION)
#    error Version mismatch! Probably this file was generated by an earlier version of opp_msgc: 'make clean' should help.
#endif
```

```
/**
 * Class generated from <tt>Msg.msg</tt> by opp_msgc.
 * <pre>
 * message Msg
 * {
 *     int source;
 *     int destination;
 *     int hopCount = 0;
 * }
 * </pre>
 */
class Msg : public ::cMessage
{
  protected:
    int source_var;
    int destination_var;
    int hopCount_var;

    // protected and unimplemented operator==(), to prevent accidental usage
    bool operator==(const Msg&);

  public:
    Msg(const char *name=NULL, int kind=0);
    Msg(const Msg& other);
    virtual ~Msg();
    Msg& operator=(const Msg& other);
    virtual Msg *dup() const {return new Msg(*this);}
```

```
    virtual void parsimPack(cCommBuffer *b);
    virtual void parsimUnpack(cCommBuffer *b);

    // field getter/setter methods
    virtual int getSource() const;
    virtual void setSource(int source_var);
    virtual int getDestination() const;
    virtual void setDestination(int destination_var);
    virtual int getHopCount() const;
    virtual void setHopCount(int hopCount_var);
};

inline void doPacking(cCommBuffer *b, Msg& obj) {obj.parsimPack(b);}
inline void doUnpacking(cCommBuffer *b, Msg& obj) {obj.parsimUnpack(b);}

#endif // _MSG_M_H_
```

Επίσης, στη μέθοδο generateMessage() δημιουργείται το μήνυμα προς αποστολή. Ο κώδικας της μεθόδου αυτής παρατίθεται παρακάτω:

Πίνακας 17: Μέθοδος generateMessage

```
Msg *Txc1::generateMessage()
{
    //Δημιουργεί τις διευθύνσεις αποστολέα και παραλήπτη
    int src = getIndex();   //ο δείκτης του τρέχοντος κόμβου
    int n = size();      //το μέγεθος του κόμβου
    int dest = intuniform(0,n-2);
    if (dest>=src) dest++;
```

```
    char msgname[20];
    sprintf(msgname, "send-%d-to-%d", src, dest);

    //Δημιουργία του μηνύματος και τοποθέτηση των πεδίων αποστολέα και παραλήπτη
    Msg *msg = new Msg(msgname);
    msg->setSource(src);
    msg->setDestination(dest);
    return msg;
}
```

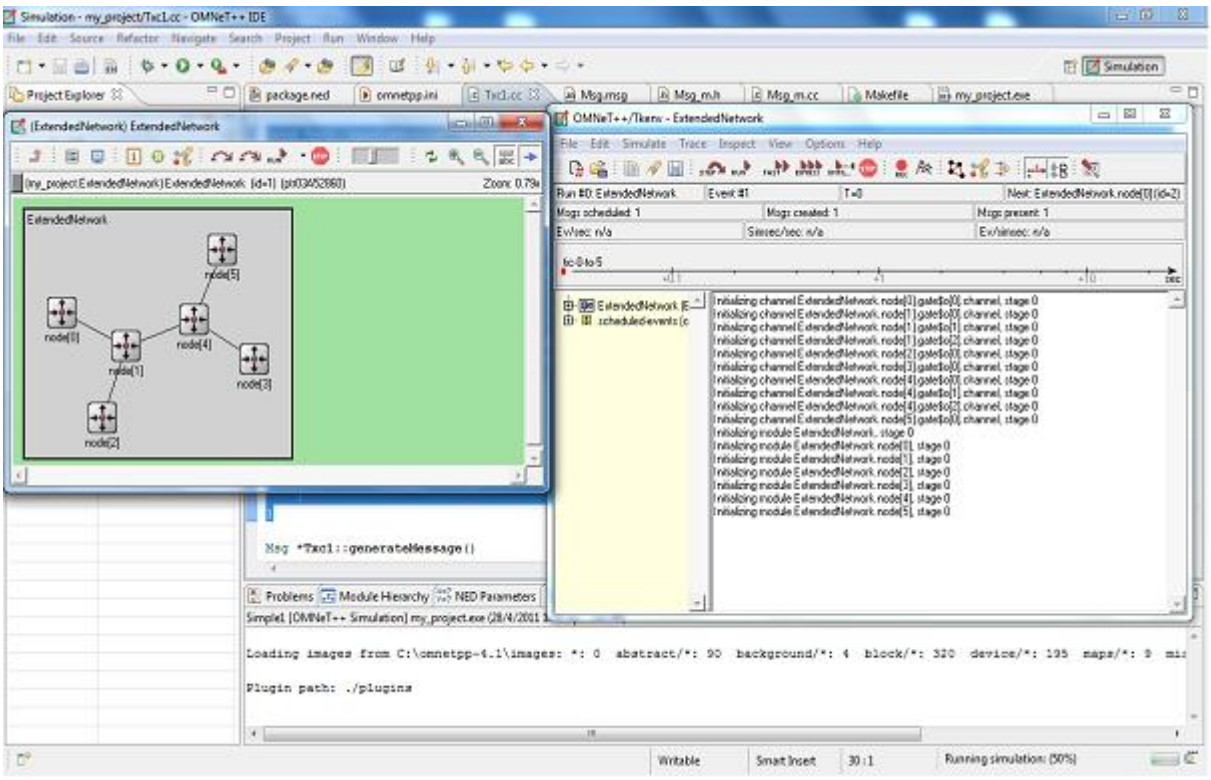
Τέλος, παρατίθεται ο κώδικας της μεθόδου handleMessage() που εκτελείται όταν αποσταλεί κάποιο μήνυμα στον κόμβο:

Πίνακας 18: Μέθοδος handleMessage

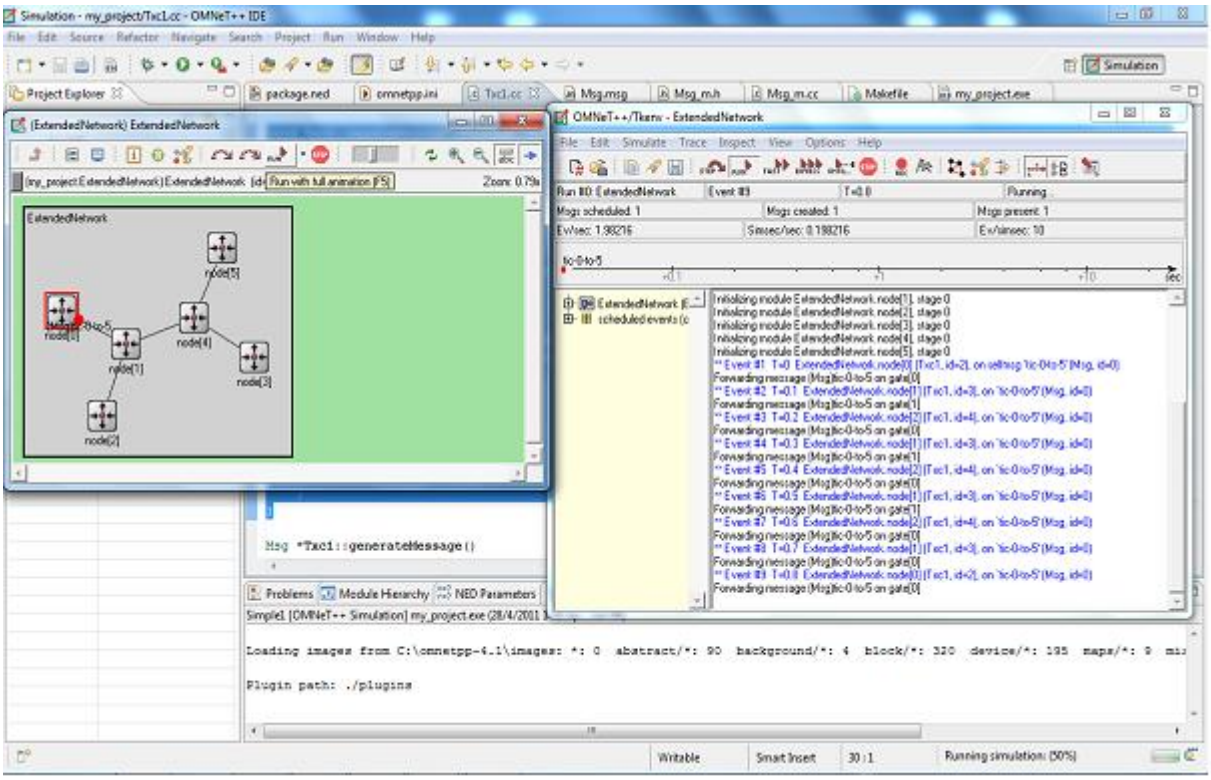
```
void Txc1::handleMessage(cMessage *msg)
{
    Msg *ttmsg = check_and_cast<Msg *>(msg);

    if (ttmsg->getDestination()==getIndex())
    {
        //Το μήνυμα έφτασε
        EV << "Message " << ttmsg << " arrived after " << ttmsg->getHopCount() << " hops.\n";
        bubble("ARRIVED, starting new one!");
        delete ttmsg;

        //Δημιουργία νέου μηνύματος
        EV << "Generating another message: ";
        Msg *newmsg = generateMessage();
        EV << newmsg << endl;
```

```
        forwardMessage(newmsg);
    }
    else
    {
        //Προώθηση μηνύματος
        forwardMessage(ttmsg);
    }
}
```

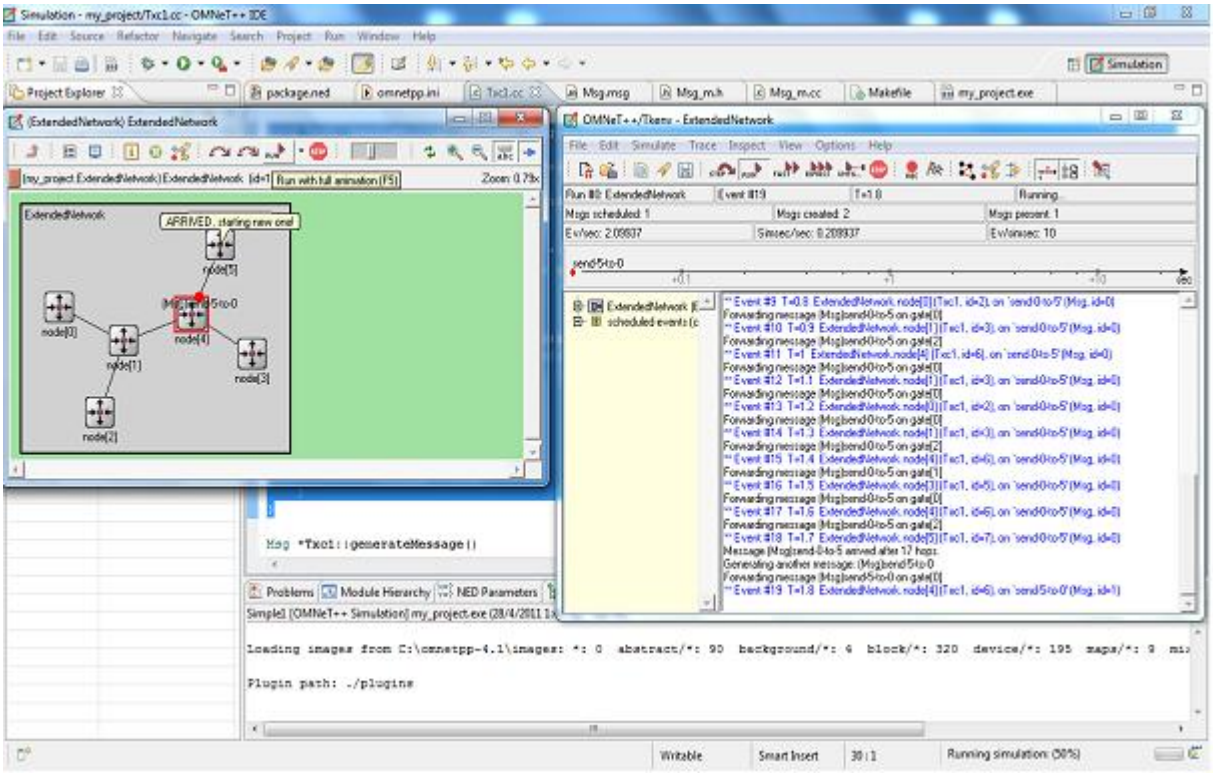
Στη συνέχεια μπορούμε να εκτελέσουμε το πρόγραμμα. Μερικά στιγμιότυπα από την εκτέλεση του φαίνεται στις παρακάτω εικόνες:



Εικόνα 39: Αρχικό στιγμιότυπο 3ης προσομοίωσης



Εικόνα 40: Ενδιάμεσο στιγμιότυπο 3ης προσομοίωσης



Εικόνα 41: Τελικό στιγμιότυπο 3ης προσομοίωσης

4.4 Συλλογή στατιστικών στοιχείων προσομοίωσης με πολλούς κόμβους σε ένα πραγματικό δίκτυο

Στην προηγούμενη προσομοίωση θα προσθέσουμε κάποια στατιστικά στοιχεία. Το OMNet++ προσφέρει ένα μηχανισμό για την καταγραφή τιμών και γεγονότων. Κάθε μοντέλο μπορεί να έχει σήματα που φέρουν μία τιμή ή ένα αντικείμενο. Στην προσομοίωση αυτή χρειαζόμαστε ένα μόνο σήμα που να φέρει τον μετρητή αλμάτων του μηνύματος, ούτως ώστε να γνωρίζουμε τα άλματα που χρειάστηκαν για να φτάσει το μήνυμα στον προορισμό του. Οπότε αρχικά μέσα στην κλάση ορίζουμε ένα σήμα με όνομα arrivalSignal. Επίσης, πρέπει να τροποποιήσουμε τη μέθοδο handleMessage, ούτως ώστε όταν καταφτάνει ένα μήνυμα να στέλνεται ένα σήμα. Ο κώδικας του τροποποιημένου αρχείου Txcl.cc αποτυπώνεται παρακάτω:

Πίνακας 19: Αρχείο Txcl.cc για συλλογή στατιστικών στοιχείων

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

#include "Msg_m.h"


class Txc1 : public cSimpleModule
{
  private:
      simsignal_t arrivalSignal;

  protected:
    virtual Msg *generateMessage();
    virtual void forwardMessage(Msg *msg);
    virtual void initialize();
```

```
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc1);

void Txc1::initialize()
{
    arrivalSignal = registerSignal("arrival");
    //Το module 0 στέλνει το πρώτο μήνυμα
    if (getIndex()==0)
    {
        //Ξεκινά τη διαδικασία δημιουργώντας ένα μήνυμα στον εαυτό του
        Msg *msg = generateMessage();
        scheduleAt(0.0, msg);
    }
}

void Txc1::handleMessage(cMessage *msg)
{
    Msg *ttmsg = check_and_cast<Msg *>(msg);

    if (ttmsg->getDestination()==getIndex())
    {
        //Το μήνυμα έφτασε
        int hopcount = ttmsg->getHopCount();
        //Αποστολή σήματος
        emit(arrivalSignal, hopcount);

        EV << "Message " << ttmsg << " arrived after " << hopcount << " hops.\n";
```

```
        bubble("ARRIVED, starting new one!");

        delete ttmsg;

        //Δημιουργία νέου μηνύματος
        EV << "Generating another message: ";
        Msg *newmsg = generateMessage();
        EV << newmsg << endl;
        forwardMessage(newmsg);
    }
    else
    {
        //προώθηση μηνύματος
        forwardMessage(ttmsg);
    }
}

Msg *Txc1::generateMessage()
{
    //Δημιουργία διευθύνσεων αποστολέα και παραλήπτη
    int src = getIndex();
    int n = size();
    int dest = intuniform(0,n-2);
    if (dest>=src) dest++;

    char msgname[20];
    sprintf(msgname, "send-%d-to-%d", src, dest);

    //Δημιουργία μηνύματος και παραγωγή διευθύνσεων αποστολέα και παραλήπτη
```

```
    Msg *msg = new Msg(msgname);
    msg->setSource(src);
    msg->setDestination(dest);
    return msg;
}

void Txc1::forwardMessage(Msg *msg)
{
    //Αύξηση μετρητή άλματος
    msg->setHopCount(msg->getHopCount()+1);

    //Επιλογή τυχαίας θύρας
    int n = gateSize("gate");
    int k = intuniform(0,n-1);

    EV << "Forwarding message " << msg << " on gate[" << k << "]\n";
    send(msg, "gate$o", k);
}
```

Επίσης, πρέπει να ορίσουμε το σήμα στο αρχείο ned. Ο τροποποιημένος κώδικας του αρχείου package.ned παρατίθεται παρακάτω:

Πίνακας 20: Αρχείο package.ned για συλλογή στατιστικών στοιχείων

```
Package my_project;

//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
```

```
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with this program.  If not, see http://www.gnu.org/licenses/.
//

simple Txc1
{
    parameters:
        @signal[arrival](type="int");
        @statistic[hopCount](title="hop count"; source="arrival"; record=vector,stats; interpolationmode=none);

        @display("i=block/routing");
    gates:
        inout gate[];
}

//
// Δήλωση δικτύου
//
network ExtendedGraphNetwork
{
    types:
        channel Channel extends ned.DelayChannel
```

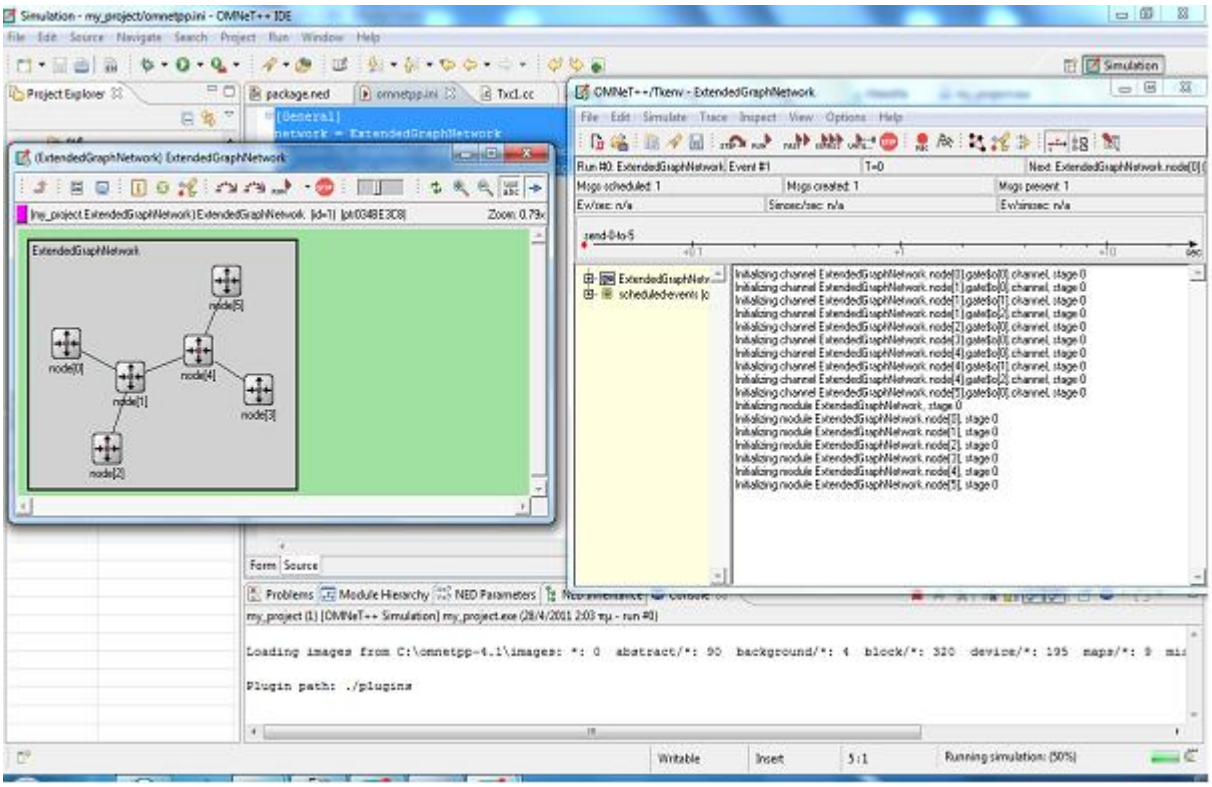
```
        {
            delay = 100ms;
        }
    submodules:
        node[6]: Txc1;
    connections:
        node[0].gate++ <--> Channel <--> node[1].gate++;
        node[1].gate++ <--> Channel <--> node[2].gate++;
        node[1].gate++ <--> Channel <--> node[4].gate++;
        node[3].gate++ <--> Channel <--> node[4].gate++;
        node[4].gate++ <--> Channel <--> node[5].gate++;
}
```

Υπάρχουν επίσης κάποιες περαιτέρω ρυθμίσεις που μπορούν να γίνουν στο αρχείο omnetpp.ini. Αν για παράδειγμα θέλουμε να εξάγουμε ένα ιστόγραμμα για το μετρητή αλμάτων στον κόμβο 1 και επίσης δε θέλουμε να καταγράφουμε τις πληροφορίες για τους κόμβους 0, 1 και 2, τότε μπορούμε να τροποποιήσουμε το αρχείο omnetpp.ini ως εξής:

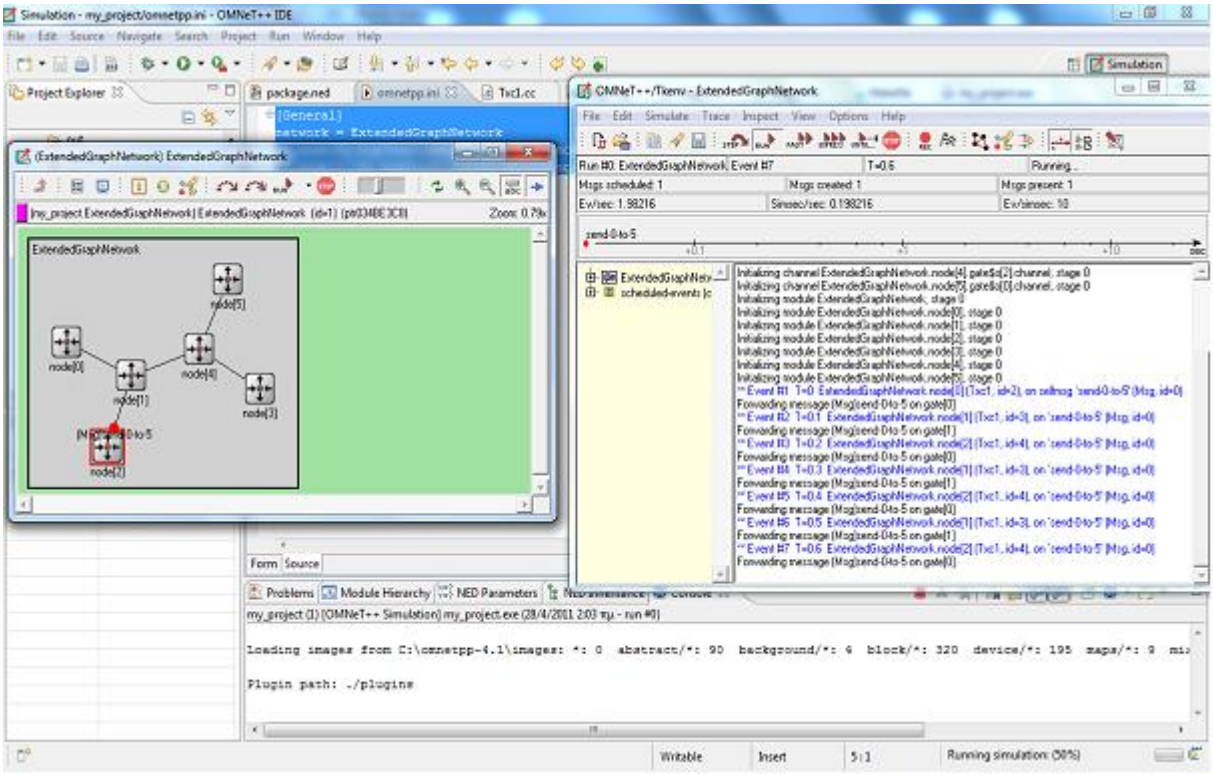
Πίνακας 21: Αρχείο omnetpp.ini για συλλογή στατιστικών στοιχείων

```
[General]
network = ExtendedGraphNetwork
**.node[1].hopCount.result-recording-modes = +histogram
**.node[0..2].hopCount.result-recording-modes = -vector
```

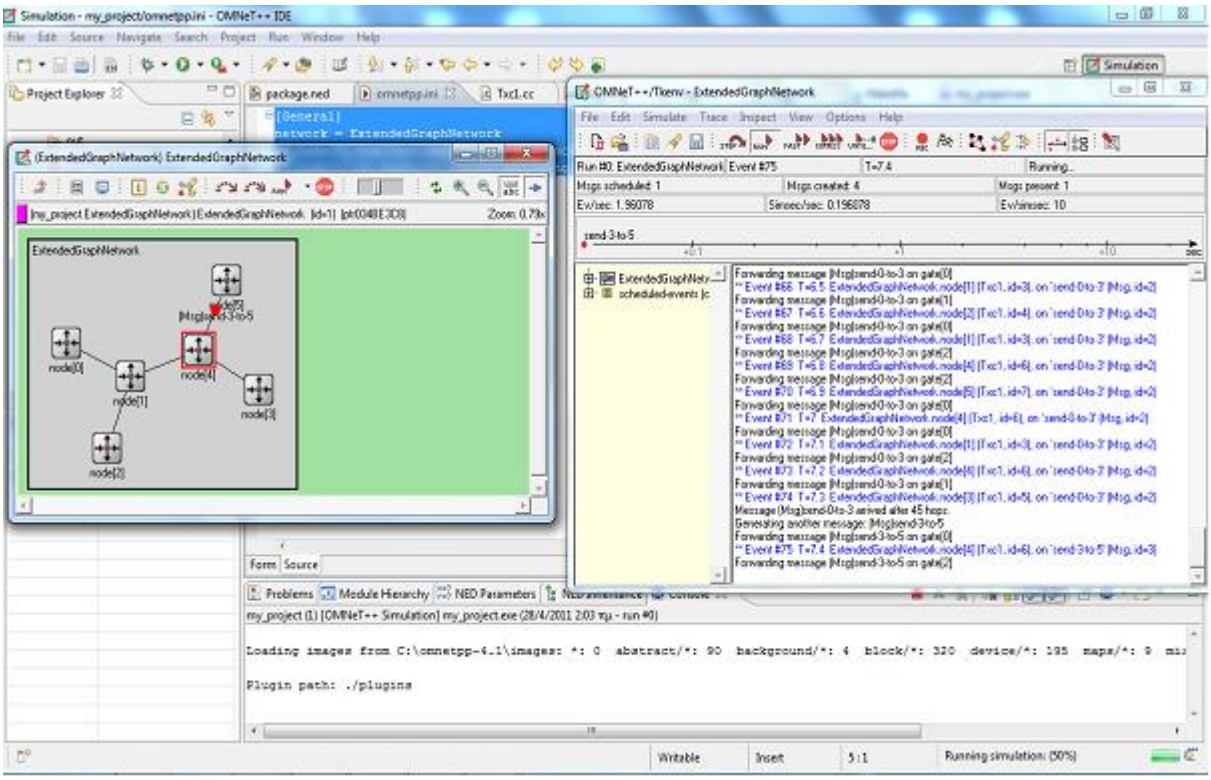
Στη συνέχεια μπορούμε να εκτελέσουμε το πρόγραμμα. Μερικά στιγμιότυπα από την εκτέλεση του φαίνεται στις παρακάτω εικόνες:



Εικόνα 42: Αρχικό στιγμιότυπο προσομοίωσης για συλλογή στατιστικών στοιχείων

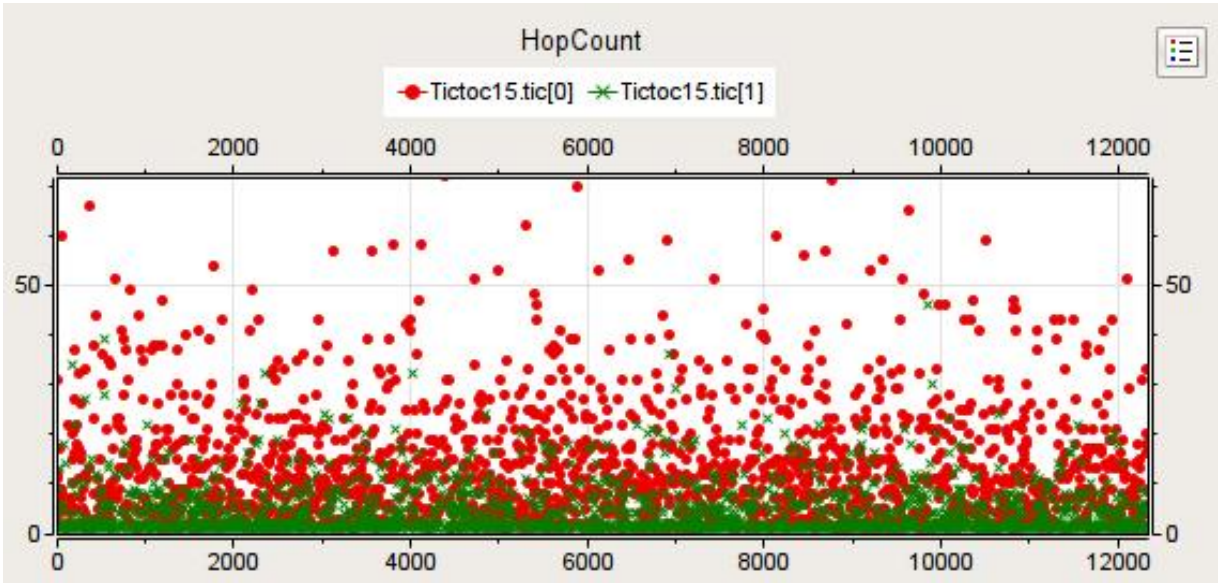


Εικόνα 43: Ενδιάμεσο στιγμιότυπο προσομοίωσης για συλλογή στατιστικών στοιχείων



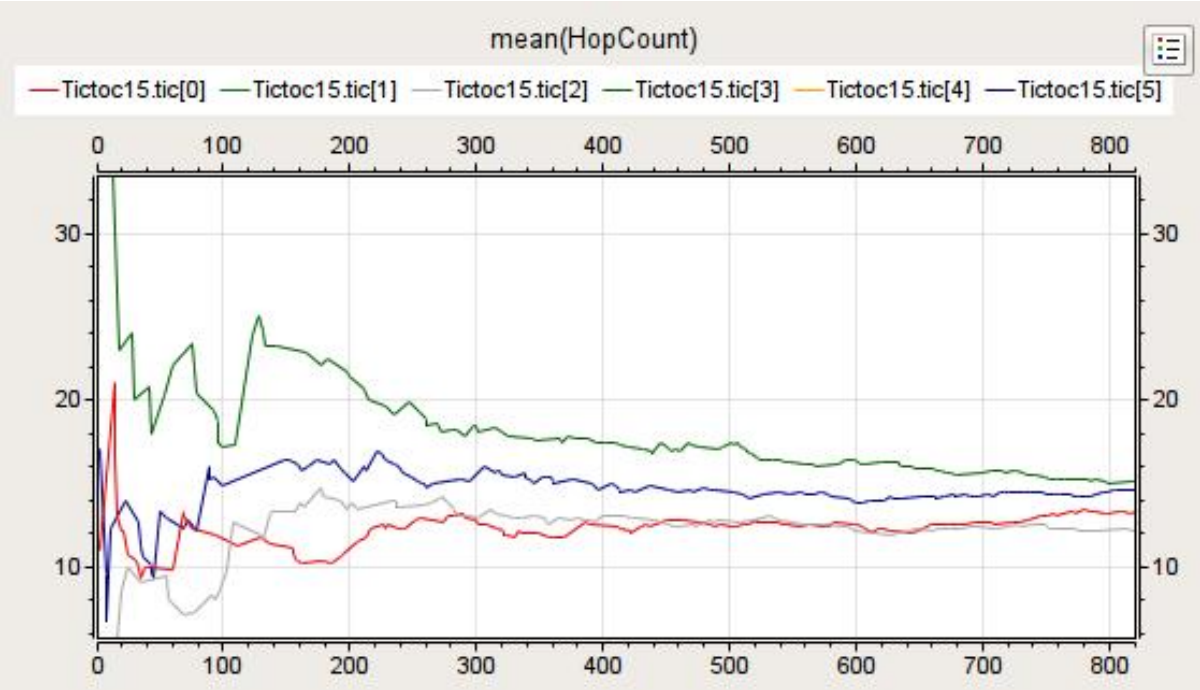
Εικόνα 44: Τελικό στιγμιότυπο προσομοίωσης για συλλογή στατιστικών στοιχείων

Το πρόγραμμά μας καταγράφει το μετρητή αλμάτων του μηνύματος κάθε φορά που φτάνει στον προορισμό. Στην παρακάτω εικόνα φαίνεται το γράφημα που δείχνει αυτούς τους πίνακες για τους κόμβους 0 και 1.



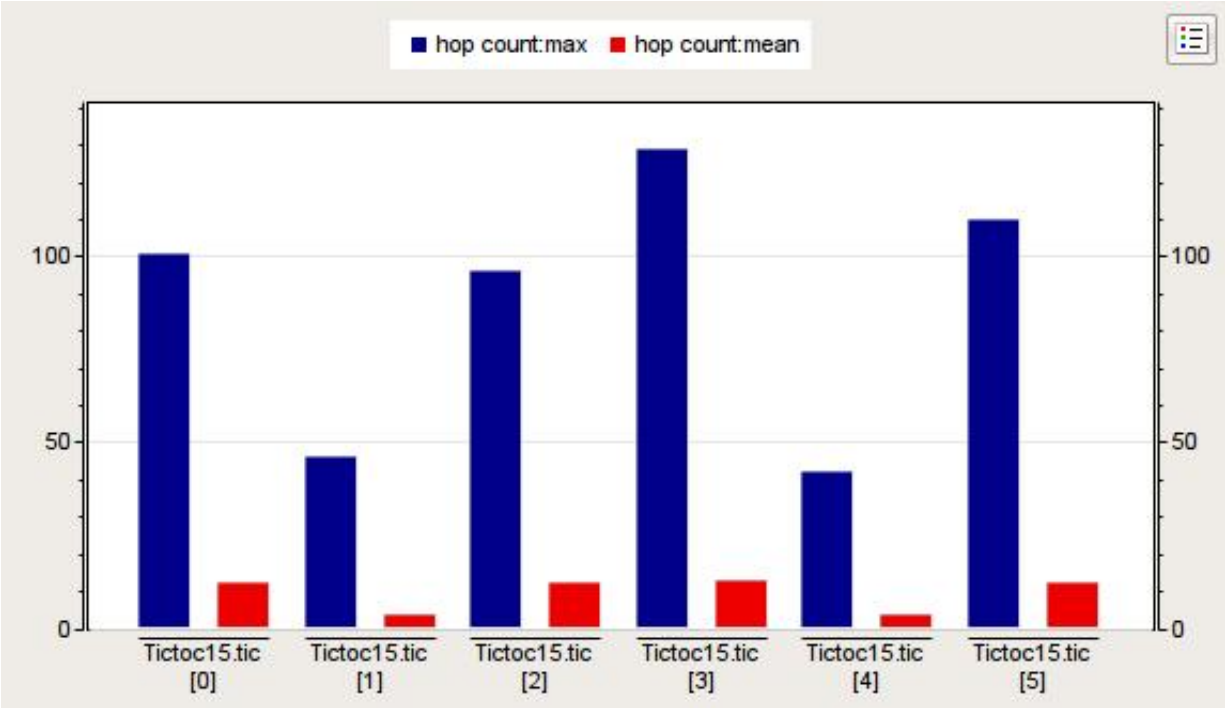
Εικόνα 45: Γράφημα μετρητή αλμάτων

Αν επιλέξουμε το φίλτρο mean, τότε μπορούμε να δούμε πόσο ο μετρητής αλμάτων σε κάθε κόμβο απέχει από το μέσο όρο:



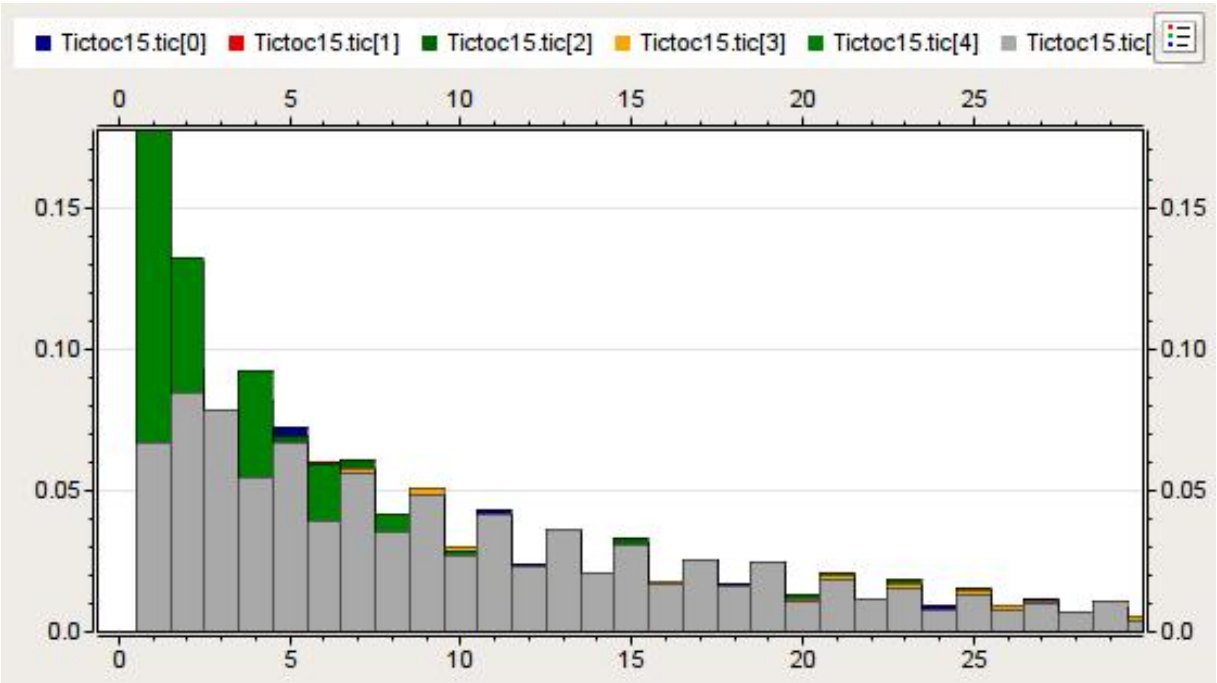
Εικόνα 46: Μέσος όρος μετρητή αλμάτων

Στο παρακάτω γράφημα μπορούμε να δούμε το μέσο και το μέγιστο μετρητή άλματος σε κάθε κόμβο:



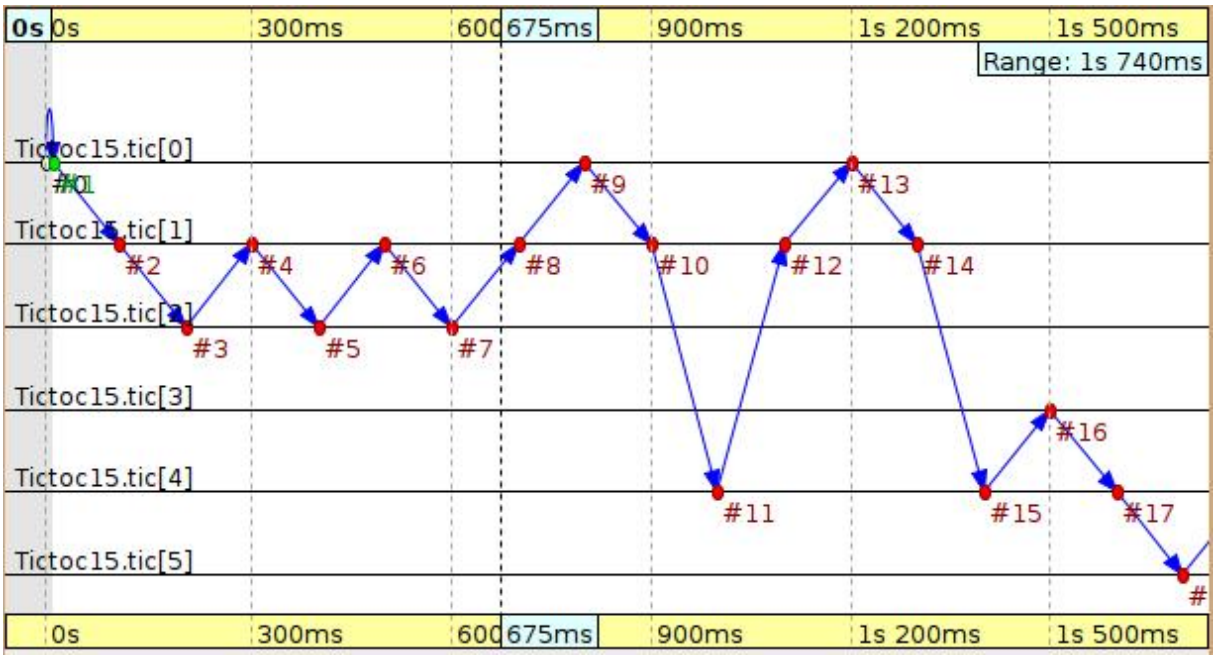
Εικόνα 47: Μέση και μέγιστη τιμή μετρητή αλμάτων

Επίσης, μπορούμε να εξάγουμε το ιστόγραμμα για κάθε κόμβο ξεχωριστά:



Εικόνα 48: Ιστόγραμμα για κάθε κόμβο

Τέλος, μπορούμε να δούμε και ένα γράφημα με τη δρομολόγηση ενός μηνύματος στους διάφορους κόμβους:



Εικόνα 49: Γράφημα δρομολόγησης μηνυμάτων στους διάφορους κόμβους

ΚΕΦΑΛΑΙΟ 5: Συμπεράσματα

Η υλοποίηση ενός μεγάλου δικτύου υπολογιστών είναι μια δύσκολη και χρονοβόρα διαδικασία. Η διάταξη του περιβάλλοντος χώρου, η απόσταση των κόμβων και πάρα πολλοί άλλοι παράγοντες καθορίζουν την ποιότητα της σύνδεσης και την επίτευξη επικοινωνίας σε ένα δίκτυο υπολογιστών. Έτσι 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. Η τακτική αυτή μπορεί ενδεχομένως να αποδώσει σε ένα απλό δίκτυο μικρής έκτασης και λίγων κόμβων όπου μπορούμε να αναγνωρίσουμε και να θέσουμε εύκολα μεταβλητές και προϋποθέσεις. 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. Η δεύτερη δυνατότητα που μας δίνεται για το σχεδιασμό ενός δικτύου υπολογιστών είναι η προσομοίωση. Μια προσομοίωση αξιολογείται από το πλήθος, τη λεπτομέρεια και την πιστότητα των χαρακτηριστικών και των λειτουργιών που αναπαριστά. Οι μέθοδοι προσομοίωσης είναι πάρα πολλές και καλύπτουν διάφορα φαινόμενα.

Πάνω σε αυτό το θέμα έχουν υλοποιηθεί αρκετοί προσομοιωτές δικτύων. Για την μελέτη που προηγήθηκε επιλέχθηκαν να χρησιμοποιηθούν τέσσερις προσομοιωτές, ο OPNET Modeler, ο NS-2, ο JSIM και ο OMNET++. Η χρήση τους είναι αρκετά διαδεδομένη στην κοινότητα ανάπτυξης εφαρμογών προσομοίωσης δικτύων και καλύπτουν ένα ευρύ φάσμα λειτουργιών δίνοντας μια αίσθηση επιβεβαίωσης και ασφάλειας ότι θα εξυπηρετήσουν το σκοπό τους.

Ίσως το περισσότερο διαδεδομένο εργαλείο προσομοίωσης, από αυτά που περιγράψαμε, να θεωρείται από πολλούς το NS-2, το οποίο παρέχει μια μεγάλη γκάμα τοπολογιών, πρωτοκόλλων και εφαρμογών που μπορεί να προσομοιώσει και χρησιμοποιείται εκτενώς από την ερευνητική κοινότητα. Εξίσου διαδεδομένο είναι και το OMNET++ διότι εκτός του ότι από μόνο του μπορεί να χρησιμοποιηθεί για οποιαδήποτε προσομοίωση, έχουν αναπτυχθεί με βάση αυτό και άλλα εργαλεία τα οποία εξειδικεύονται σε τοπολογίες ή συγκεκριμένα πρωτόκολλα δικτύων και τα οποία στη συνέχεια μπορούν να συνδυαστούν άψογα με οποιαδήποτε άλλη εφαρμογή έχει αναπτυχθεί σε OMNET++. Βέβαια, το JSIM απαιτεί λιγότερο χρόνο εκμάθησής και είναι δωρεάν τόσο για εκπαιδευτικούς σκοπούς όσο και για εμπορικούς. Το OPNET είναι μια ολοκληρωμένη λύση κλειστού κώδικα και επιτρέπει ένα μεγάλο φάσμα προσομοιώσεων.

Το OMNET++ θεωρήθηκε ο πιο κατάλληλος προσομοιωτής για την προσομοίωση του παραδείγματός μας από τους τέσσερις προσομοιωτές. Μετά από τη χρήση του πακέτου αυτού διαπιστώσαμε ότι το OMNET++ είναι ένα πολύ εύκολο στη χρήση λογισμικό προσομοιώσεων με το οποίο μπορούμε να εξάγουμε αρκετά στατιστικά στοιχεία γύρω από τη λειτουργία του προσομοιωμένου δικτύου. Για την εξαγωγή συμπερασμάτων για την χρήση των άλλων προσομοιωτών στηριζόμαστε στις μαρτυρίες άλλων χρηστών. Ο NS-2 έχει την φήμη του εύκολου στην χρήση προσομοιωτή αλλά αυτό είναι παραπλανητικό γιατί αυτήν η άποψη προέρχεται από ήδη έμπειρους χρήστες. Για τον αρχάριο χρήστη υπάρχουν αρκετές επιλογές από βοηθήματα και εγχειρίδια χρήσης. Το OPNET καθότι είναι εμπορικό προϊόν και έρχεται με την επαγγελματική υποστήριξη της εταιρίας προσφέρει ένα εύκολο στην χρήση γραφικό περιβάλλον. Τέλος το JSIM έχει το πλεονέκτημα ότι εκτελείται σε περιβάλλον JAVA και δεν περιορίζεται από την επιλογή του λειτουργικού συστήματος.

Σε αυτό το σημείο πρέπει να πούμε ότι η επιλογή ενός προσομοιωτή έναντι κάποιου άλλου δεν μπορεί να γίνει αν δεν έχει προηγηθεί ακριβής έρευνα και προσεκτική ανάλυση των αναγκών που θέτουμε. Φτάνουμε σε ένα παράδοξο αποτέλεσμα και αυτό φαίνεται από την σύγκριση που διενεργήθηκε και εξάγαμε το συμπέρασμα ότι ο προσομοιωτής OMNET++ είναι καλύτερος σε μερικά σημεία από ότι οι άλλοι προσομοιωτές, ενώ ο JSIM είναι ο χειρότερος προσομοιωτής στα περισσότερα κριτήρια, αλλά παρόλα αυτά, αυτό δε σημαίνει ότι ο OMNET++ είναι καλύτερος για την προσομοίωση οποιουδήποτε δικτύου και για αυτό ο κάθε χρήστης πρέπει να ελέγξει ενδελεχώς τις δυνατότητες του κάθε προσομοιωτή και να επιλέξει αυτόν που ικανοποιεί τις ανάγκες του.

Κλείνοντας θα θέλαμε να επισημάνουμε το γεγονός ότι παρόλο που έχουν γίνει μελέτες σύγκρισης προσομοιωτών, αυτές επικεντρώνονται σε συγκεκριμένα σενάρια τα οποία δεν είναι αντιπροσωπευτικά των πραγματικών φαινομένων που μπορεί να χρειαστεί να αντιμετωπίσει ο χρήστης και δεν δίνεται η δυνατότητα να προβούμε σε ασφαλή συμπεράσματα ως προς την τελική καταλληλότητα ενός προσομοιωτή μόνο και μόνο εξετάζοντας τα χαρακτηριστικά του ή το σύνηθες πεδίο χρήσης του.

Βιβλιογραφία

S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, M. Haldar, P. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejajie, S. Punnet, K. Varadhan, X. Ya, H. Yu, and D. Zappala (1999), Improving simulation for network research. Technical Report 99-702b, USC Computer Science Department.

L. Begg, W. Liu, K. Pawlikowski, S. Perera, and H. Sirisena (2006), Survey of Simulators of Next Generation Networks for Studying Service Availability and Resilience. Technical Report TR-COSC 05/06, Department of Computer Science & Software Engineering, University of Canterbury, Christchurch, New Zealand.

S. Duflos, G. L. Grand, A. A. Diallo, C. Chaudet, A. Hecker, C. Balducelli, F. Flentge, C. Schwaegerl, and O. Seifert (2006), Deliverable d 1.3.2: List of Available and Suitable Simulation Components. Technical report, Ecole Nationale Supieure des Tommunications (ENST).

Jacl, Java Command Language (2011), http://dev.scriptics.com/software/java/

JSIM (2011), http://sites.google.com/site/jsimofficial/

J. Lessmann, P. Janacik, L. Lachev, and D. Orfanus (2008), Comparative Study of Wireless Network Simulators. The Seventh International Conference on Networking, pages 517-523, IEEE.

D. M. Nicol (2003), Scalablity of Network Simulators Revisited, SCS SNDS, Orlando, USA

D. M. Nicol (2003), Utility Analysis of Network Simulators, International Journal of Simulation: Systems, Science, and Technology.

NS-2 (2011), http://www.isi.edu/nsnam/ns/

NAM (2011), http://www.isi.edu/nsnam/nam/

K. Fall (1999), Network emulation in the VINT/ns simulator. In Fourth IEEE Symposium on Computers and Communications (ISCC'99), pages 59-67.

NS by Example, Jae Chung and Mark Claypool, Worcester Polytechnic Institute, http://perform.wpi.edu/NS/

Information Sciences Institute, University of Southern California - http://www3.isi.edu/home

M. Fleury, G. F. Lucio and M. J. Reed (2003), Clarification of the "OPNET NS-2 Comparison" Paper with regards to OPNET Modeler.

OPNET (2011), http://www.opnet.com/

OPNET Users' Manual (2011), OPNET Architecture, OV.415.http://forums.opnet.com.

Simulation Tools and Communications, OPNET-Network Simulator, Jarmo Prokkola (2006), VTT Technical Research Centre of Finland

Flores Lucio, G., Paredes-Farrera, M., Jammeh, E., Fleury, M., and J Reed, M. (2003), Opnet modeler and ns-2 - comparing the accuracy of network simulators for packet-level analysis using a network testbed. *WSEAS Transactions on Computers 2*, 3, pp. 700 - 707.

OTcl Tutorial (2011), http://otcl-tclcl.sourceforge.net/otcl/

PDNS - Parallel/Distributed NS (2011),

http://www.cc.gatech.edu/computing/compass/pdns/index.html

The Network Simulator - ns-2 (2011), http://www.isi.edu/nsnam/ns/index.html.

A.Varga (2011), OMNET++ documentation, http://www.omnetpp.org/documentation

OMNET++ (2011), http://www.omnetpp.org/

Wikipedia, http://www.wikipedia.com

Siteceerx, http://www.siteceerx.edu

Μ. Δ. Λογοθέτης (2001), ΘΕΩΡΙΑ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΗΣ ΚΙΝΗΣΕΩΣ ΚΑΙ ΕΦΑΡΜΟΓΕΣ, Εκδόσεις Παπασωτηρίου, Αθήνα 2001.

Α. Πομπόρτσης, Α. Τσουλφάς (2001), ΠΡΟΣΟΜΟΙΩΣΗ ΔΙΚΤΥΩΝ ΥΠΟΛΟΓΙΣΤΩΝ, Εκδόσεις Τζιόλα, Αθήνα 2001.