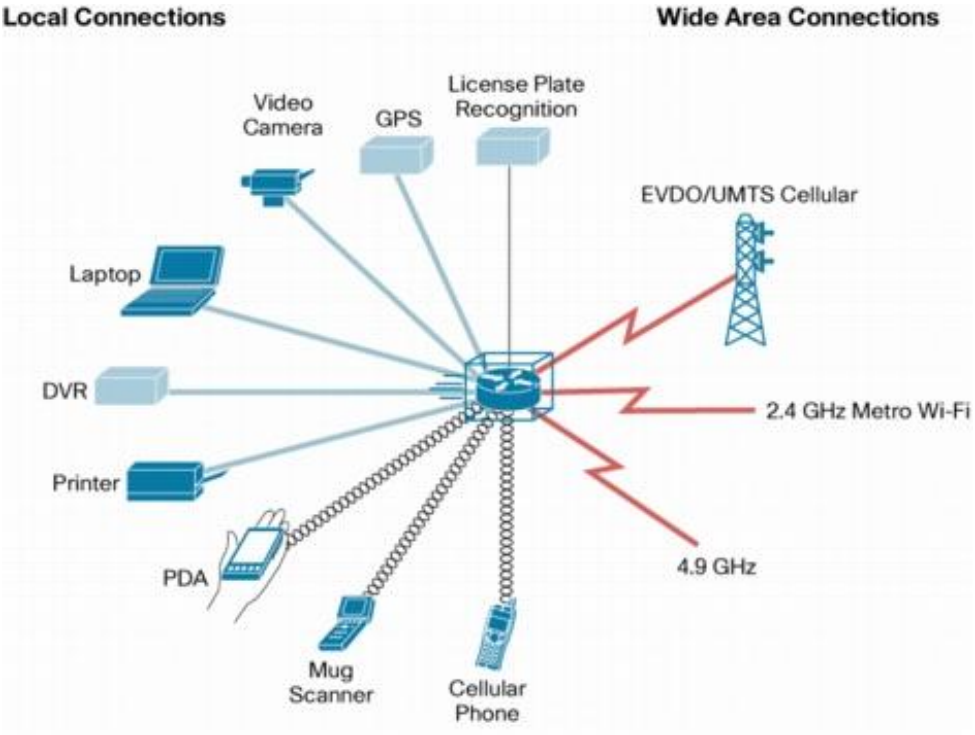
ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΑΤΡΩΝ
ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ
ΤΜΗΜΑ ΕΠΙΧΕΙΡΗΜΑΤΙΚΟΥ ΣΧΕΔΙΑΣΜΟΥ ΚΑΙ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΑΛΓΟΡΙΘΜΟΙ ΔΡΟΜΟΛΟΓΗΣΗΣ ΚΑΙ ΜΕΤΑΔΟΣΗΣ ΜΗΝΥΜΑΤΩΝ ΣΕ AD-HOC ΚΙΝΗΤΑ ΔΙΚΤΥΑ »



<u>Ονόματα σπουδαστών :</u>
Δημακόπουλος Παναγιώτης
Κερδέλας Αναστάσιος
Σέβης Δημήτριος

Εποπτεύων Καθηγητής : Γκουμόπουλος Χρήστος

ΠΑΤΡΑ - 2011

Ευχαριστούμε τον κ.Γκουμόπουλο για την καθοδήγηση του κατά την εκπόνηση της παρούσας πτυχιακής εργασίας.

ΠΡΟΛΟΓΟΣ

Με την εξάπλωση των κινητών συσκευών και την ενσωμάτωσή τους στο διαδίκτυο, δημιουργήθηκε η ανάγκη για την σχεδίαση νέων πρωτοκόλλων δρομολόγησης, τα οποία θα λαμβάνουν υπόψη τη συνεχώς μεταβαλλόμενη τοπολογία ενός δικτύου, κάτι το οποίο δε γίνεται από τα ήδη υπάρχοντα πρωτόκολλα στατικής δρομολόγησης. Μία από τις προκλήσεις, που κάνουν την υλοποίηση πρωτοκόλλων δυναμικής δρομολόγησης αρκετά πιο δύσκολη, είναι η αδυναμία δοκιμών των πρωτοκόλλων αυτών σε πραγματικές συνθήκες. Αν και αυτό το πρόβλημα υπάρχει και στα πρωτόκολλα στατικής δρομολόγησης (και για να είμαστε πιο ακριβείς, σε οποιοδήποτε πρωτόκολλο φεύγει από το στάδιο της ακαδημαϊκής έρευνας και περνάει στο στάδιο της πραγματικής εφαρμογής), είναι αρκετά πιο εύκολο να ελεχθεί ένα στατικό πρωτόκολλο δρομολόγησης σε ακαδημαϊκό περιβάλλον με πραγματικά δεδομένα. Για παράδειγμα μία συνεργασία ακαδημαϊκών ιδρυμάτων με κάποιες επιχειρήσεις, θα μπορούσε να δημιουργήσει ένα περιβάλλον διαδικτύου, στο οποίο θα μπορούσε να ελεγχθεί ένα πρωτόκολλο δρομολόγησης μέσω της κατάλληλης διαμόρφωσης δρομολογητών. Αν το διαδίκτυο είναι αρκετά μεγάλο, θα μπορούσε να θεωρηθεί ως ένα περιβάλλον ελέγχου που προσεγγίζει αρκετά τα πραγματικά δεδομένα όσον αφορά παραμέτρους όπως μέγεθος και πολυπλοκότητα δικτύου. Στην περίπτωση των ad hoc δικτύων ωστόσο κάτι τέτοιο θα ήταν δύσκολο να γίνει. Τα πραγματικά σενάρια ad hoc δικτύων απαιτούν σταθμούς που θα αλλάζουν θέσεις συνέχεια και μάλιστα πιθανόν με αρκετά μεγάλη ταχύτητα. Το σενάριο για παράδειγμα κινητών σταθμών που βρίσκονται σε αεροπλάνο που ταξιδεύει υπερατλαντικά, είναι αδύνατο να αναπαραχθεί εργαστηριακά.

Μία λύση για το παραπάνω πρόβλημα προσφέρουν οι εξομοιωτές δικτύου, και συγκεκριμένα το network simulator 2 (ns2) που θα μελετηθεί στην παρούσα πτυχιακή εργασία. Αν και το ns μπορεί να χρησιμοποιηθεί γα την εξομοίωση οποιουδήποτε δικτύου, είναι στην κατηγορία των ad hoc δικτύων που προσφέρει τα μέγιστα, αφού δίνει τη δυνατότητα να εξομοιωθεί οποιαδήποτε τοπολογία δικτύου. Ο δε ανοιχτός (open) χαρακτήρας του, επιτρέπει τη χρήση του από οποιαδήποτε ερευνητική ομάδα ή ιδιώτη δωρεάν και την συχνή ανανέωση του ns2 με τα τελευταία πρωτόκολλα που χρησιμοποιούνται στην αγορά ή στην έρευνα. Δεν είναι τυχαίο, ότι το ns2 είναι το κύριο εργαλείο που χρησιμοποιείται στην έρευνα πρωτοκόλλων δρομολόγησης ad hoc δικτύων.

Κύριος στόχος της παρούσας πτυχιακής εργασίας είναι να δείξει τη χρήση του ns2, και πώς μπορούν να γραφτούν προγράμματα που να εξομοιώνουν διάφορες τοπολογίες δικτύου. Πιο συγκεκριμένα, περιγράφεται η διαδικασία εγκατάστασης του ns2, δίνεται μία εισαγωγή στο πώς μπορούν να γραφτούν προγράμματα στο ns2, καθώς και η διαδικασία ανάλυσης των αποτελεσμάτων της εξομοίωσης μέσω της γλώσσας awk. Καθώς η αναλυτική παρουσίαση όλων

των πρωτοκόλλων βρίσκεται πέρα από τους στόχους της παρούσας πτυχιακής εργασίας, προτιμήθηκε να γίνει η εξομοίωση σε ένα αντιπροσωπευτικό δείγμα πρωτοκόλλων.

ΠΕΡΙΛΗΨΗ

Στην εισαγωγή της πτυχιακής εργασίας δίνεται μία σύντομη ιστορική αναδρομή στα πρωτόκολλα δρομολόγησης μηνυμάτων που ξεκινάει από τα σήματα καπνού των αρχαίων πολιτισμών και φτάνει μέχρι τα σημερινά πρωτόκολλα για ad hoc δίκτυα. Δίνονται τα γενικά χαρακτηριστικά των πρωτοκόλλων για στατική δρομολόγηση και εξηγείται γιατί αυτά είναι ανεπαρκή για ad hoc δίκτυα.

Στο δεύτερο κεφάλαιο γίνεται μία περίληψη των πρωτοκόλλων DSDV, DSR, TORA, AODV και WRP τα οποία θα χρησιμοποιηθούν για τις εξομοιώσεις

Στο τρίτο κεφάλαιο περιγράφεται η διαδικασία εγκατάστασης του ns2, ο τρόπος λειτουργίας του, καθώς και η διαδικασία εξαγωγής αποτελεσμάτων από τα αρχεία εξόδου του ns2, μέσω της χρήσης της γλώσσας awk.

Στο τέταρτο κεφάλαιο δίνεται το σενάριο εξομοίωσης που χρησιμοποιήθηκε για τις εξομοιώσεις της παρούσας πτυχιακής εργασίας, ένα δίκτυο με εκατό τυχαία κινούμενους κόμβους, από τους οποίους οι 75 στέλνουν δεδομένα στους υπόλοιπους 25, καθώς και μία περιγραφή του κώδικα που αναπτύχθηκε και που εξομοιώνει την παραπάνω τοπολογία.

Στο πέμπτο κεφάλαιο παρουσιάζουμε τα αποτελέσματα των εξομοιώσεων. Οι μετρικές που χρησιμοποιούνται είναι: α) Μέσοι χρόνοι παραλαβής πακέτων, β) αποκλίσεις (δεύτερη ροπή) από τη μέση τιμή των χρόνων και γ) ποσοστό αναμετάδοσης μηνυμάτων (είτε λόγω χαμένων πακέτων πληροφοριών, είτε λόγω ανανέωσης μή έγκυρων μονοπατιών δρομολόγησης).

Τέλος, στο παράρτημα δίνονται οι πλήρεις κώδικες του ns2 καθώς και της awk που χρησιμοποιήθηκαν για την εξομοίωση και την ανάλυση αποτελεσμάτων αντίστοιχα.

Πίνακας Περιεχομένων

1. Εισαγωγή

1.1 Ιστορική Αναδρομή

Η ανάγκη δρομολόγησης μηνυμάτων είναι σχεδόν τόσο παλιά, όσο και η επικοινωνία μεταξύ ανθρώπων από μεγάλη απόσταση. Ήδη από προϊστορικούς χρόνους τα σήματα καπνού χρησιμοποιούνταν για να διαδώσουν επείγοντα νέα σε μακρινές αποστάσεις. Η δρομολόγηση που γινόταν ήταν απλή. Ένας πύργος για παράδειγμα που δεχόταν επίθεση, έκανε σήμα φωτιάς το οποίο το έβλεπε ένας άλλος πύργος με τον οποίο είχε οπτική επαφή. Ο δεύτερος πύργος αναπαρήγαγε το σήμα, και ούτω καθεξής μέχρι το μήνυμα να φτάσει σε όλους τους πύργους. Με την ανάπτυξη και τη διάδοση του ταχυδρομείου αρκετούς αιώνες μετά, δημιούργησε επίσης ανάγκη για αποτελεσματική μέθοδο δρομολόγησης. Αντί ο ταχυδρόμος να μεταφέρει απευθείας τα μηνύματα μεταξύ δέκτη και παραλήπτη, κάτι που θα απαιτούσε πολλά και μεγάλα δρομολόγια για να μην αναφέρουμε και πολλούς ταχυδρόμους, δημιουργήθηκαν οι ταχυδρομικοί σταθμοί (post offices). Κάθε γράμμα πηγαίνει στον ταχυδρομικό σταθμό που είναι υπεύθυνος για την περιοχή. Ο ταχυδρομικός σταθμός με τη σειρά του, προωθεί το γράμμα σε άλλον ταχυδρομικό σταθμό ανάλογα με ταχυδρομικό κώδικα του παραλήπτη. Η διαδικασία επαναλαμβάνεται μέχρι να φτάσει στον παραλήπτη.

Με τις ανακαλύψεις του Graham Bell και Guglielmo Marconi, το τηλέφωνο απέκτησε σημαντική θέση στα μέσα επικοινωνίας μεγάλων αποστάσεων. Και εδώ, η ανάγκη για δρομολόγηση είναι εμφανής, αφού απευθείας συνδέσεις μεταξύ όλων των τηλεφωνικών συσκευών είναι πρακτικά αδύνατο να υλοποιηθούν. Σε αναλογία με το ταχυδρομείο, και στην περίπτωση της τηλεφωνικής επικοινωνίας, υπάρχουν τα λεγόμενα τηλεφωνικά κέντρα τα οποία δημιουργούν (χειροκίνητα παλαιότερα μέσω του τηλεφωνητή, αυτόματα πλέον) τη σύνδεση μεταξύ των δύο ομιλητών.

Ερχόμενοι πλέον στις τελευταίες δεκαετίες του 20^{ου} αιώνα, το διαδίκτυο (internet) ξεκινά ως ένα διαπανεπιστημιακό δίκτυο υπολογιστών για ανταλλαγή πληροφοριών για να φτάσει στις μέρες μας να αποτελεί ένα διεθνές δίκτυο με εκατομμύρια συνδρομητές μέσα από το οποίο οι χρήστες επικοινωνούν, ανταλλάσουν δεδομένα, διαφημίζονται, κάνουν ηλεκτρονικές αγορές και πολλά άλλα. Καθώς το διαδίκτυο σχεδιάστηκε με αρχικό στόχο την ανταλλαγή δεδομένων, το πρωτόκολλο δρομολόγησης διαδικτύου (internet protocol, σε συντομογραφία IP), διαφέρει σε αρκετά σημεία σε σχέση με τη δρομολόγηση τηλεφωνικών συνδέσεων.

Συγκεκριμένα, στο IP η πληροφορία μοιράζεται σε πακέτα, τα οποία δρομολογούνται ξεχωριστά χωρίς καμία εγγύηση ότι θα παραδοθούν σωστά ή σε κάποια συγκεκριμένη σειρά. Ο παραλήπτης αναλαμβάνει την υποχρέωση να

ενημερώσει αν το κάθε πακέτο έχει φτάσει σωστά και να «συναρμολογήσει» τα πακέτα ώστε να ανακτηθεί η αρχική πληροφορία. Αυτή η διαδικασία είναι αρκετά διαφορετική από τη δρομολόγηση τηλεφωνικής επικοινωνίας, στην οποία απαιτείται κάθε πακέτο να φτάνει στη σωστή σειρά (ώστε να μην έχουμε scrambled ομιλία), αλλά και να έχει όσο το δυνατόν χαμηλότερη καθυστέρηση (latency). Παρά τις διαφορές αυτές, και στις δύο περιπτώσεις διατηρείται το μοντέλο του ιεραρχικού routing, αφού κάθε σταθμός, επικοινωνεί με τον αμέσως επόμενο δρομολογητή, προκειμένου να βρεθεί διαδρομή.

Με τον ερχομό και την καθιέρωση των κινητών συσκευών το μοντέλο του ιεραρχικού routing άρχισε να δείχνει τις αδυναμίες του. Αν και στο στατικό ακόμα routing είχαν αναπτυχθεί ad hoc αλγόριθμοι δρομολόγησης για την περίπτωση μεταβαλλόμενης τοπολογίας δικτύου (όπως π.χ. συμβαίνει όταν κάποιος δρομολογητής βγει εκτός λειτουργίας), στα δίκτυα κινητής επικοινωνίας το μεταβαλλόμενο δίκτυο αποτελεί κανόνα και όχι εξαίρεση. Είναι προφανές λοιπόν, ότι έπρεπε να αναπτυχθούν νέοι αλγόριθμοι δρομολόγησης που να λαμβάνουν υπόψη ότι τόσο οι γειτονικοί σταθμοί, όσο και οι σταθμοί δρομολόγησης (π.χ. ένας σταθμός κυψέλης) θα αλλάζουν συνεχώς με το χρόνο.

1.2 Στατική Δρομολόγηση

Στη στατική δρομολόγηση υποτίθεται ότι η τοπολογία του δικτύου είναι σταθερή. Κάθε σταθμός έχει μία διεύθυνση δικτύου τη λεγόμενη διεύθυνση ip. Η ip δηλώνει μοναδικά τον κάθε σταθμό, ενώ τα πρώτα ψηφία της δηλώνουν το δίκτυο στο οποίο ανήκει. Οι δρομολογητές μέσω της διεύθυνσης δικτύου, βρίσκουν τον επόμενο δρομολογητή στον οποίο θα προωθήσουν το μήνυμα. Όταν το μήνυμα φτάσει στο δρομολογητή που ανήκει στο δίκτυο του παραλήπτη, τότε το μήνυμα θα παραδοθεί μέσω του τοπικού δικτύου.

Για τη διευθυνσιοδότηση χρησιμοποιούνται οι λεγόμενες διευθύνσεις IP (IP addresses), οι οποίες αποτελούνται από τέσσερις οκτάμπιτους αριθμούς (παράδειγμα ip διεύθυνσης είναι: 150.142.32.123).

1.3 Ad hoc δρομολόγηση

Για τον συντονισμό μεταξύ των κόμβων ενός ad hoc δικτύου και τη διευκόλυνση της επικοινωνίας μεταξύ οποιονδήποτε ζευγαριών από αυτούς, χρησιμοποιούνται πρωτόκολλα δρομολόγησης, τα οποία ανακαλύπτουν διαδρομές μεταξύ των κόμβων αυτών. Τα ad hoc κινητά δίκτυα, αρκετά ιδιαίτερα χαρακτηριστικά, τα οποία καθιστούν τα παραδοσιακά πρωτόκολλα

δρομολόγησης, που έχουν σχεδιαστεί για ενσύρματα δίκτυα , ακατάλληλα για αυτά .

Τα πρωτόκολλα δρομολόγησης για ad hoc δίκτυα, μπορούν να διαιρεθούν σε δύο βασικές κατηγορίες :

- Table-driven (proactive) πρωτόκολλα
- On-demand (reactive) πρωτόκολλα

Στα table-driven πρωτόκολλα δρομολόγησης , κάθε κόμβος διατηρεί έναν ή περισσότερους πίνακες , οι οποίοι περιέχουν πληροφορίες δρομολόγησης για κάθε άλλο κόμβο στο δίκτυο . Όλοι οι κόμβοι ανανεώνουν αυτούς τους πίνακες , έτσι ώστε να διατηρήσουν μια συνεπή και ενημερωμένη άποψη του δικτύου .
Στα on-demand πρωτόκολλα δρομολόγησης, σε αντίθεση με τα table-driven πρωτόκολλα , όλες οι έγκυρες διαδρομές δεν διατηρούνται σε κάθε κόμβο , αλλά δημιουργούνται όπως και όταν απαιτείται. Όταν μια πηγή θέλει να στείλει πληροφορίες σε έναν προορισμό , καλεί τη διαδικασία ανεύρεσης διαδρομής για να βρει το μονοπάτι για τον προορισμό. Η διαδρομή παραμένει έγκυρη για όσο μπορούμε να φτάσουμε

Στο επόμενο κεφάλαιο παρουσιάζονται συνοπτικά τα ad hoc πρωτόκολλα δρομολόγησης με τα οποία θα ασχοληθεί η παρούσα πτυχιακή

2. Mobile Routing Πρωτόκολα

Στο παρόν κεφάλαιο θα δοθεί μία περιληπτική παρουσίαση των πρωτοκόλλων με τα οποία θα ασχοληθεί η παρούσα πτυχιακή εργασία. Τα πρωτόκολλα αυτά είναι τα:

1. DSDV
2. DSR
3. AODV
4. TORA
5. WRP

2.1 DSDV

Ο DSDV βασίζεται στον κλασσικό αλγόριθμο δρομολόγησης των Bellman-Ford, με κάποιες βελτιώσεις .

Σύμφωνα με το DSDV , κάθε κινητός κόμβος του δικτύου διατηρεί έναν πίνακα δρομολόγησης , στον οποίο αποθηκεύει όλους τους πιθανούς προορισμούς , τον απαιτούμενο αριθμό των hops για κάθε προορισμό και τον sequence number , ο οποίος έχει οριστεί από τον προορισμό . Ο αριθμός αυτός χρησιμοποιείται για να διαχωριστούν οι παλιές διαδρομές από τις νεώτερες , και έτσι αποφεύγεται η δημιουργία loops . Οι κόμβοι μεταδίδουν περιοδικά τους πίνακες δρομολόγησής τους στους άμεσους γείτονές τους , έτσι ώστε να διατηρείται η συνέπεια των πινάκων . Επίσης μεταδίδουν τους πίνακές δρομολόγησής τους αν συμβεί κάποια σημαντική αλλαγή στην τοπολογία του δικτύου (και επομένως στους πίνακες τους) στο χρόνο μεταξύ των περιοδικών μεταδόσεων . Για να μειωθεί η πιθανά μεγάλη κίνηση στο δίκτυο που μπορεί να προκληθεί από τέτοιου είδους ενημερώσεις των πινάκων δρομολόγησης , οι ενημερώσεις αυτές μπορούν να σταλούν με δύο είδη πακέτων .Το πρώτο είδος ,είναι γνωστό σαν “full dump” πακέτα , περιέχουν ολόκληρους τους πίνακες δρομολόγησης και μπορεί να απαιτήσουν πολλαπλές μονάδες δεδομένων του πρωτοκόλλου του δικτύου (NPDUs) . Το δεύτερο είδος ,είναι τα πακέτα επαύξησης (incremental packets), τα οποία χρησιμοποιούνται για να σταλούν μόνο εκείνες οι εγγραφές των πινάκων δρομολόγησης που έχουν αλλάξει από την τελευταία ενημέρωση και πρέπει να χωρούν σε ένα NPDU , και έχουν ως αποτέλεσμα να μειώνεται το ποσό της κίνησης που παράγεται . Αν υπάρχει χώρος στα πακέτα επαύξησης , τότε μπορούν να συμπεριληφθούν και οι εγγραφές εκείνες των οποίων έχει αλλάξει ο sequence number . Όταν το δίκτυο είναι σχετικά σταθερό , στέλνονται πακέτα επαύξησης , έτσι ώστε να αποφευχθεί η επιπλέον κίνηση , ενώ τα πακέτα “full dump” είναι σχετικά σπάνια . Σε ένα δίκτυο που αλλάζει συχνά τα πακέτα επαύξησης μπορεί να μεγαλώσουν , επομένως τα πακέτα “full dump” θα είναι πιο συχνά . Κάθε πακέτο ενημέρωσης , περιέχει τη διεύθυνση του προορισμού , τον αριθμό των hops για να φτάσουμε στον προορισμό αυτό , το sequence number των

πληροφοριών που ελήφθησαν σε σχέση με τον προορισμό αυτό , όπως επίσης και ένα sequence number το οποίο είναι μοναδικό για την εκπομπή . Η διαδρομή με το μεγαλύτερο sequence number , δηλαδή η πιο πρόσφατη , είναι αυτή που χρησιμοποιείται . Στην περίπτωση που δύο διαδρομές έχουν το ίδιο sequence number , τότε η διαδρομή με την καλύτερη μετρική , δηλαδή η μικρότερη διαδρομή , χρησιμοποιείται . Όταν κάποιος κόμβος Α αντιληφθεί ότι η διαδρομή μέχρι τον προορισμό D έχει πάψει να είναι έγκυρη , τότε αυξάνεται ο αριθμός hop-count της διαδρομής αυτής . Έτσι , την επόμενη φορά που ο Α θα κοινοποιήσει στους γείτονες του τον πίνακα δρομολόγησής του , θα δώσει στη διαδρομή προς τον D άπειρο hop-count και ένα sequence number που είναι μεγαλύτερος από πριν .

Οι κόμβοι υπολογίζουν επίσης το χρόνο εγκατάστασης μιας διαδρομής , δηλαδή το μέσο χρόνο κατά τον οποίο κυμαίνονται οι διαδρομές για ένα προορισμό μέχρι να ληφθεί η καλύτερη διαδρομή . Έτσι , καθυστερούν την εκπομπή μιας ενημέρωσης διαδρομής κατά ένα ποσό χρόνου ίσο με το χρόνο εγκατάστασης , μειώνοντας έτσι την κίνηση του δικτύου και βελτιστοποιώντας τις διαδρομές , αφού εξαλείφονται οι εκπομπές αυτές οι οποίες θα συνέβαιναν αν μια καλύτερη διαδρομή βρισκόταν πολύ σύντομα .

Ιδιότητες (Πλεονεκτήματα-Μειονεκτήματα)

Πλεονεκτήματα

- Εγγυάται ότι δεν υπάρχουν loops στους πίνακες δρομολόγησης , χρησιμοποιώντας τα sequence numbers για να διαχωρίσει τις παλιές από τις νέες διαδρομές .
- Ενώ παρέχει μόνο ένα μονοπάτι για κάθε προορισμό , επιλέγει το μικρότερο μονοπάτι βασισμένος στον αριθμό των hops για τον προορισμό .
- Παρέχει δύο είδη πακέτων ενημέρωσης , το ένα από τα οποία είναι σημαντικά μικρότερο από το άλλο και το οποίο μπορεί να χρησιμοποιηθεί για ενημερώσεις επαύξησης έτσι ώστε να μη χρειάζεται να σταλεί ολόκληρος ο πίνακας δρομολόγησης για κάθε αλλαγή στην τοπολογία του δικτύου .
- Διατηρεί ενημερωμένες διαδρομές χρησιμοποιώντας τα sequence numbers .

Μειονεκτήματα

Ο DSDV είναι μη αποδοτικός γιατί :

- Απαιτεί εκπομπή περιοδικών ενημερώσεων ανεξάρτητα από τον αριθμό των αλλαγών στην τοπολογία του δικτύου , το οποίο έχει ως σα συνέπεια να περιορίζεται ο αριθμός των κόμβων που μπορούν να συνδεθούν στο δίκτυο , αφού το συνολικό κόστος του δικτύου αυξάνεται ($O(n^2)$) .

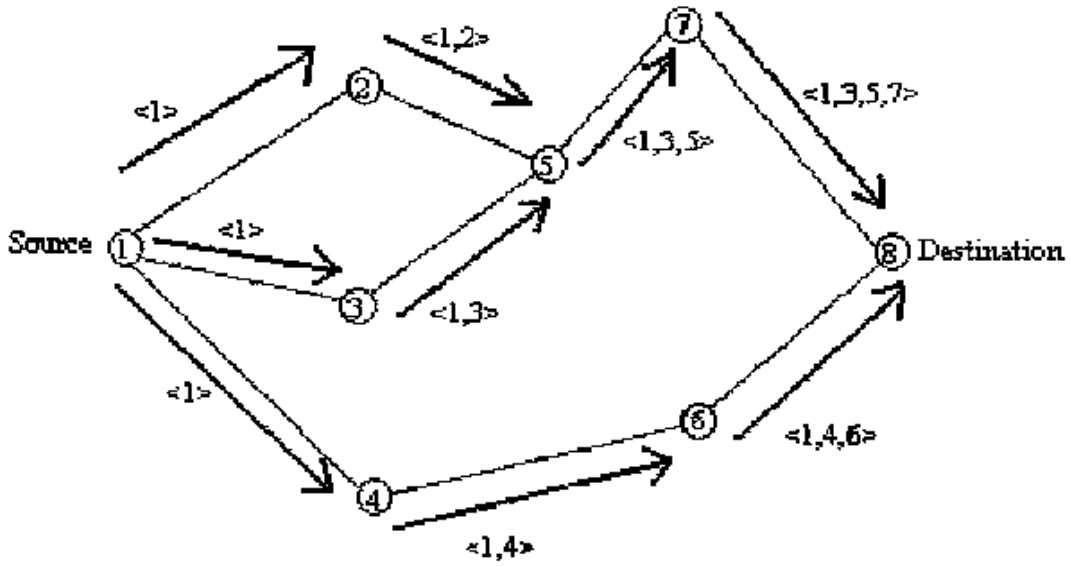
- Χρειάζεται κάποιο χρόνο έτσι ώστε να συγκλίνει πριν χρησιμοποιηθεί κάποια διαδρομή . Αυτός ο χρόνος σύγκλισης μπορεί να θεωρηθεί αμελητέος σε ένα στατικό δίκτυο , όπου η τοπολογία δεν αλλάζει και τόσο συχνά , αλλά στα ad hoc δίκτυα η τοπολογία περιμένουμε να μεταβάλλεται πολύ συχνά .Έτσι ο χρόνος αυτός σύγκλισης μπορεί να σημαίνει ότι ένας μεγάλος αριθμός πακέτων έχουν απορριφθεί προτού βρεθεί μια κατάλληλη διαδρομή .

2.2 DSR

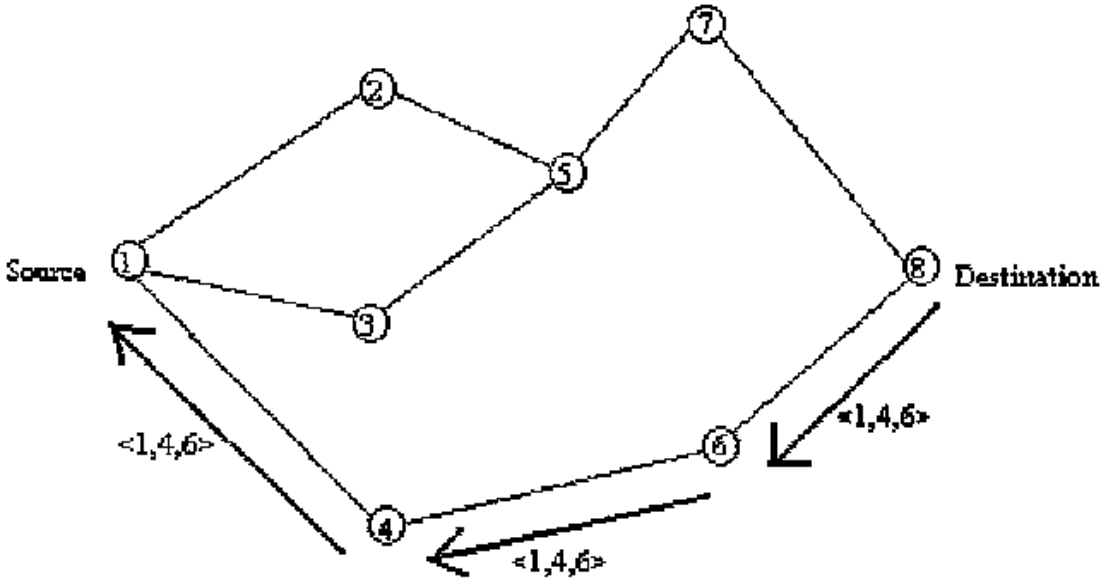
Το DSR πρωτόκολλο , είναι ένα on-demand πρωτόκολλο δρομολόγησης το οποίο βασίζεται στην έννοια της δρομολόγησης πηγής . Κάθε κόμβος χρειάζεται να διατηρεί κρυφές μνήμες διαδρομών , οι οποίες περιέχουν τις διαδρομές πηγής για τις οποίες είναι ενήμερος . Οι εγγραφές στην κρυφή μνήμη διαδρομών ενός κόμβου ενημερώνονται συνεχώς , καθώς αυτός μαθαίνει για νέες διαδρομές . Κάθε εγγραφή στην κρυφή μνήμη διαδρομών έχει συσχετισμένη με αυτή μια περίοδο λήξης , μετά την οποία η εγγραφή διαγράφεται από την κρυφή μνήμη .

Το πρωτόκολλο αποτελείται από δύο κύριες φάσεις : την ανακάλυψη διαδρομής και τη διατήρηση διαδρομής . Όταν ένας κινητός κόμβος θέλει να στείλει ένα πακέτο σε κάποιον προορισμό , ελέγχει την κρυφή μνήμη διαδρομών του για να καθορίσει αν ήδη έχει μια διαδρομή για τον προορισμό αυτό . Αν βρει ότι υπάρχει μια διαδρομή για τον προορισμό που δεν έχει λήξει, χρησιμοποιεί τη διαδρομή αυτή για να στείλει το πακέτο . Αν όμως ο κόμβος δεν έχει μια τέτοια διαδρομή , τότε ξεκινάει τη διαδικασία ανακάλυψης διαδρομής εκπέμποντας ένα πακέτο αίτησης διαδρομής . Αυτό το πακέτο αίτησης διαδρομής , περιέχει τη διεύθυνση της πηγής και του προορισμού και ένα μοναδικό αριθμό αναγνώρισης ταυτότητας . Κάθε ενδιάμεσος κόμβος που λαμβάνει το πακέτο αυτό , ελέγχει αν ξέρει μια διαδρομή για τον προορισμό . Αν δεν ξέρει μια τέτοια διαδρομή , προσαρτά τη διεύθυνσή του στο αρχείο διαδρομής του πακέτου και στη συνέχεια προωθεί το πακέτο στους γείτονές του . Για να μειωθεί ο αριθμός των αιτήσεων διαδρομής που μεταδίδονται , ένας κόμβος προωθεί το πακέτο αίτησης διαδρομής μόνο αν δεν έχει δει ήδη το πακέτο αυτό και η διεύθυνσή του δεν εμφανίζεται ήδη στο αρχείο διαδρομής του πακέτου .

Μια απάντηση διαδρομής παράγεται όταν το πακέτο αίτησης διαδρομής φτάσει είτε στον ίδιο τον προορισμό , είτε σε έναν ενδιάμεσο κόμβο που περιέχει στην κρυφή μνήμη διαδρομών του μια διαδρομή για τον προορισμό που δεν έχει λήξει . Ένα πακέτο αίτησης διαδρομής που φτάνει σε έναν από αυτούς τους κόμβους , ήδη περιέχει στο αρχείο διαδρομής του την ακολουθία των βημάτων (κόμβων) που έγιναν από την πηγή μέχρι τον κόμβο αυτό .



Σχήμα 2.1α. Κατασκευή του αρχείου διαδρομής κατά τη διάρκεια της ανακάλυψης διαδρομής



Σχήμα 2.1β. Μετάδοση της απάντησης διαδρομής με το αρχείο διαδρομής

Καθώς το πακέτο αίτησης διαδρομής μεταδίδεται διαμέσου του δικτύου , το αρχείο διαδρομής σχηματίζεται όπως φαίνεται στο Σχήμα 2.1α . Αν η απάντηση διαδρομής παράγεται από τον προορισμό , τότε αυτός τοποθετεί το αρχείο διαδρομής , που περιέχεται στο πακέτο αίτησης διαδρομής , στο πακέτο

απάντησης διαδρομής . Αν όμως ο κόμβος που παράγει την απάντηση διαδρομής είναι ένας ενδιάμεσος κόμβος , τότε αυτός προσαρτά την αποθηκευμένη του διαδρομή για τον προορισμό στο αρχείο διαδρομής του πακέτου αίτησης διαδρομής και το τοποθετεί στη συνέχεια (το αρχείο διαδρομής) στο πακέτο απάντησης διαδρομής . Το Σχήμα 2.1β δείχνει το πακέτο απάντησης διαδρομής (με το συσχετισμένο αρχείο διαδρομής) που στέλνεται από τον ίδιο τον προορισμό πίσω στον κόμβο-πηγή .

Για να επιστρέψει το πακέτο απάντησης διαδρομής , ο κόμβος που απαντά θα πρέπει να έχει μια διαδρομή για την πηγή . Αν έχει μια τέτοια διαδρομή στην κρυφή μνήμη διαδρομών του , μπορεί να τη χρησιμοποιήσει . Αλλιώς , αν υποστηρίζονται οι συμμετρικές συνδέσεις , μπορεί να χρησιμοποιήσει την αντίστροφη διαδρομή του αρχείου διαδρομής . Στην περίπτωση που δεν υποστηρίζονται οι συμμετρικές συνδέσεις , ο κόμβος μπορεί να ξεκινήσει τη δική του διαδικασία ανακάλυψης διαδρομής και να «φορτώσει» την απάντηση διαδρομής στη νέα αίτηση διαδρομής .

Η διατήρηση διαδρομής επιτυγχάνεται μέσω της χρήσης δύο ειδών πακέτων : τα πακέτα Λάθους Διαδρομής και τις Αναγνωρίσεις . Ένα πακέτο Λάθους Διαδρομής παράγεται σε ένα κόμβο , όταν το στρώμα Ζεύξης Δεδομένων αντιμετωπίσει ένα μοιραίο πρόβλημα μετάδοσης . Όταν ένας κόμβος λάβει ένα πακέτο Λάθους Διαδρομής , μετακινεί το hop για το λάθος αυτό από την κρυφή μνήμη διαδρομών του και όλες οι διαδρομές που περιέχουν το βήμα αυτό προς το λάθος , περικόπτονται σε αυτό το σημείο . Επιπρόσθετα με τα πακέτα Λάθους Διαδρομής , χρησιμοποιούνται και πακέτα Αναγνώρισης για την επιβεβαίωση της σωστής λειτουργίας των συνδέσεων των διαδρομών .Τέτοιες Αναγνωρίσεις περιλαμβάνουν και παθητικές αναγνωρίσεις , στις οποίες ο κόμβος είναι σε θέση να «ακούσει» τον επόμενο κόμβο να προωθεί το πακέτο κατά μήκος της διαδρομής .

Ιδιότητες (Πλεονεκτήματα-Μειονεκτήματα)

Πλεονεκτήματα

- Ένα πλεονέκτημά του σε σχέση με τα υπόλοιπα on-demand πρωτόκολλα , είναι ότι δε χρησιμοποιεί περιοδικά μηνύματα δρομολόγησης , και έτσι μειώνεται το πρόσθετο κόστος στο δίκτυο , εξοικονομώντας ενέργεια στους κόμβους καθώς και πολύτιμο εύρος ζώνης επικοινωνίας . Έτσι το πρωτόκολλο δεν επιφέρει οποιοδήποτε overhead όταν δεν υπάρχουν αλλαγές στην τοπολογία του δικτύου και επιπλέον μπορεί να τεθεί σε κατάσταση sleep mode .
- Χρησιμοποιεί το σημαντικό πλεονέκτημα της δρομολόγησης από την πηγή . Έτσι , οι ενδιάμεσοι κόμβοι δεν χρειάζεται να διατηρούν ενημερωμένες πληροφορίες για τις διαδρομές έτσι ώστε να δρομολογούν τα πακέτα που προωθούν .

- Η μάθηση των διαδρομών γίνεται με τον έλεγχο της πληροφορίας που περιέχεται στα πακέτα που λαμβάνει κάθε κόμβος . Αυτή η μορφή ενεργούς μάθησης είναι πολύ χρήσιμη , αφού μειώνει το πρόσθετο κόστος του δικτύου .
- Παρέχει υποστήριξη για αμφίδρομες συνδέσεις , με τη χρήση νέων αιτήσεων διαδρομής από τον τελικό προς τον αρχικό κόμβο .
- Δεν απαιτεί τη χρήση συμμετρικών συνδέσεων και μπορεί να χρησιμοποιήσει μη-συμμετρικές συνδέσεις όταν δεν είναι διαθέσιμες συμμετρικές .
- Επιτρέπει στους κόμβους να διατηρούν πολλαπλές διαδρομές για έναν προορισμό στην κρυφή τους μνήμη . Έτσι , όταν σπάσει μια σύνδεση σε μια διαδρομή , η πηγή μπορεί να ελέγξει την κρυφή της μνήμη για μια άλλη έγκυρη διαδρομή .
- Είναι loop-free πρωτόκολλο .
- Το πρωτόκολλο είναι κατανεμημένο , αφού δεν εξαρτάται από κάποιον κεντρικοποιημένο κόμβο .

Μειονεκτήματα

- Κάθε πακέτο στον DSR έχει ένα μικρό πρόσθετο κόστος , αφού πρέπει να περιέχει τη διαδρομή μέχρι τον αρχικό κόμβο που έστειλε το πακέτο. Το πρόσθετο αυτό κόστος αυξάνεται όταν το πακέτο πρέπει να περάσει από πολλά hops μέχρι να φτάσει στον προορισμό του .
- Τα πακέτα απάντησης διαδρομής είναι επίσης μεγαλύτερα (σε σχέση με τον AODV) , αφού περιέχουν τη διεύθυνση κάθε κόμβου κατά μήκος της διαδρομής , με αποτέλεσμα να παράγεται μεγαλύτερο overhead ελέγχου .
- Το overhead μνήμης είναι μεγαλύτερο στον DSR (σε σχέση με τον AODV) , αφού κάθε κόμβος πρέπει να θυμάται ολόκληρες διαδρομές .
- Λόγω της υπόθεσης που έγινε , ότι η διάμετρος του δικτύου είναι σχετικά μικρή , και της απαίτησης της δρομολόγησης από την πηγή , ο DSR δεν κλιμακώνεται σε μεγάλα δίκτυα .
- Αν προκύψει μια αποτυχία σύνδεσης κατά μήκος ενός μονοπατιού , ο αλγόριθμος ανακάλυψης διαδρομής πρέπει να ξανακληθεί από την πηγή για να βρεθεί ένα νέο μονοπάτι για τον προορισμό . Δε γίνεται καμιά προσπάθεια να χρησιμοποιηθεί τμηματική ανάκτηση διαδρομής , δηλαδή να επιτραπεί στους ενδιάμεσους κόμβους να προσπαθήσουν να ξαναφτιάξουν μόνοι τους τη διαδρομή . Αυτό μπορεί να οδηγήσει σε μεγαλύτερους χρόνους ανακατασκευής διαδρομής . [11]

    - Όμως η προσπάθεια και η αποτυχία ενός ενδιάμεσου κόμβου να ξαναφτιάξει μια διαδρομή , θα προκαλέσει μεγαλύτερη καθυστέρηση από ότι αν ο κόμβος-πηγή είχε προσπαθήσει να την ξαναφτιάξει αμέσως μόλις αντιλήφθηκε τη σπασμένη σύνδεση .

Performance Evaluation Measures

- Ο λόγος του συνολικού αριθμού μεταδόσεων πακέτων προς το βέλτιστο αριθμό μεταδόσεων πακέτων .
- Ο λόγος του μέσου μήκους διαδρομής προς το βέλτιστο μήκος διαδρομής .

Οι μεταβλητές που χρησιμοποιούνται είναι οι εξής :

- Pause time μεταξύ δύο διαδοχικών κινήσεων (καθορίζει την κινητικότητα των κόμβων) .
- Ο αριθμός των κόμβων .

2.3TORA

Ο TORA είναι ένας υψηλά προσαρμόσιμος , αποδοτικός , loop-free , κλιμακωτός , κατανεμημένος αλγόριθμος δρομολόγησης , ο οποίος βασίζεται στην έννοια της αντιστροφής σύνδεσης Προτείνεται για υψηλά δυναμικά κινητά , πολλαπλών βημάτων , ασύρματα δίκτυα . Είναι ένα on-demand πρωτόκολλο δρομολόγησης που ξεκινά από την πηγή . Βρίσκει πολλαπλές διαδρομές από μια πηγή για έναν προορισμό .Το κύριο χαρακτηριστικό του TORA είναι ότι τα μηνύματα ελέγχου περιορίζονται τοπικά σε ένα πολύ μικρό σύνολο κόμβων κοντά στο σημείο εμφάνισης μιας τοπολογικής αλλαγής . Για να το πετύχουν αυτό , οι κόμβοι πρέπει να διατηρούν πληροφορίες δρομολόγησης για τους γειτονικούς (one-hop) κόμβους . Το TORA παρέχει μόνο το μηχανισμό δρομολόγησης και βασίζεται πάνω στο IMEP (Internet MANET Encapsulation Protocol) για τις χαμηλότερου επιπέδου λειτουργίες. Το πρωτόκολλο εκτελεί τρεις βασικές λειτουργίες :

- Δημιουργία Διαδρομής
- Διατήρηση Διαδρομής
- Διαγραφή Διαδρομής

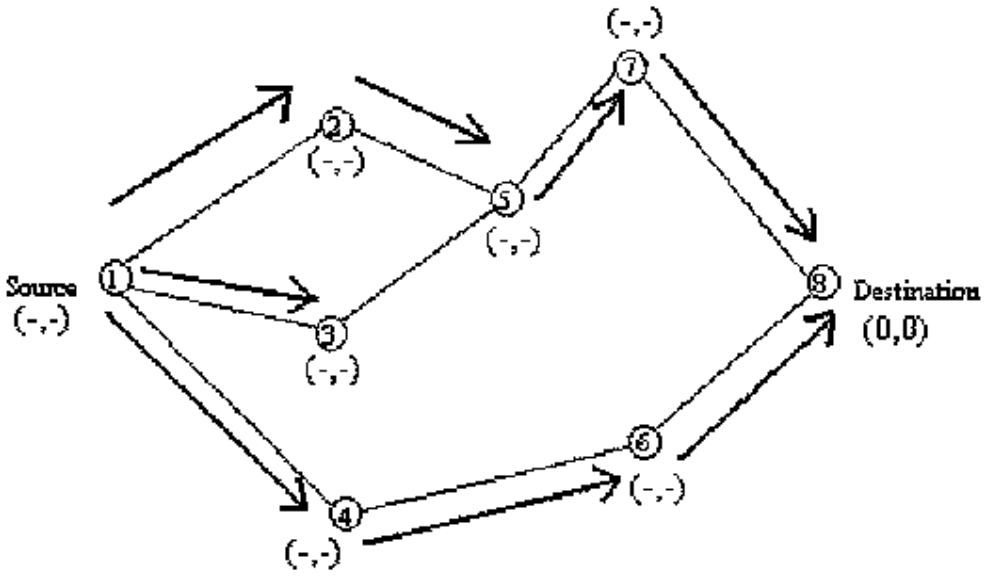
Κάθε κόμβος έχει συσχετισμένη με αυτόν μια πεντάδα :

- Το λογικό χρόνο μιας αποτυχίας σύνδεσης
- Το μοναδικό ID του κόμβου το οποίο καθόρισε το νέο επίπεδο αναφοράς
- Ένα bit ένδειξης αντανάκλασης
- Μια παράμετρο σειράς μετάδοσης
- Το μοναδικό ID του κόμβου

Τα πρώτα τρία στοιχεία επιλεκτικά αντιπροσωπεύουν το επίπεδο αναφοράς . Ένα νέο επίπεδο αναφοράς ορίζεται κάθε φορά που ένας κόμβος χάνει τον τελευταίο επόμενο του (downstream) γείτονα εξαιτίας μιας αποτυχίας σύνδεσης

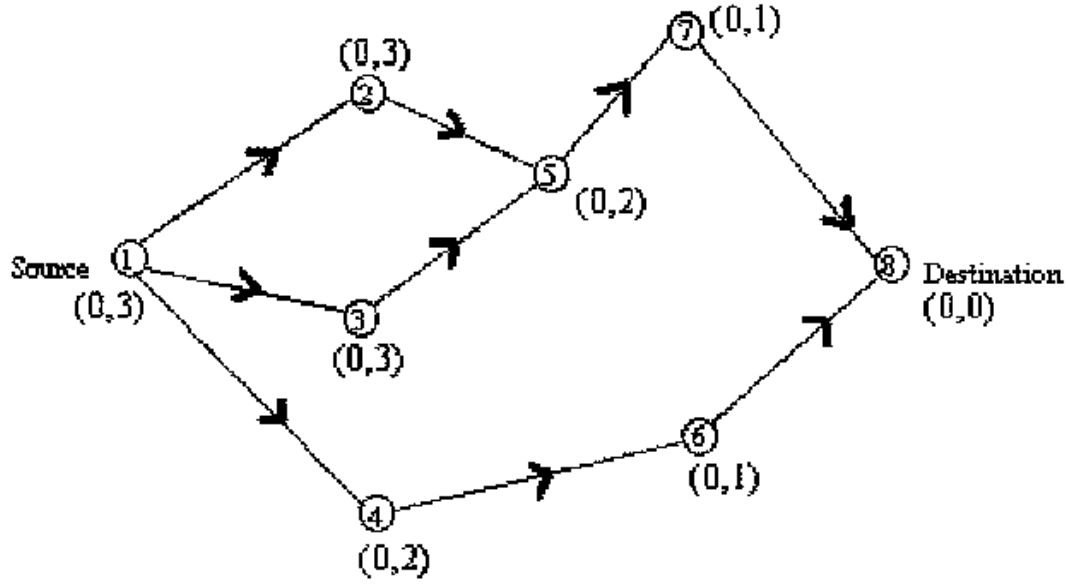
. Τα τελευταία δύο στοιχεία καθορίζουν ένα δέλτα σε σχέση με το επίπεδο αναφοράς .

Η δημιουργία Διαδρομής πραγματοποιείται χρησιμοποιώντας QRY και UPD πακέτα . Ο αλγόριθμος δημιουργίας διαδρομής ξεκινά με το "ύψος" (η παράμετρος σειράς μετάδοσης στην πεντάδα) του προορισμού να είναι 0 και το "ύψος" όλων των υπόλοιπων κόμβων να είναι NULL (δηλαδή απροσδιόριστο) . Όταν ένας κόμβος χρειάζεται μια διαδρομή για ένα προορισμό εκπέμπει ένα QRY πακέτο με το ID του προορισμού . Το πακέτο αυτό διαδίδεται μέσω του δικτύου μέχρι να φτάσει σε ένα κόμβο που να έχει διαδρομή προς τον προορισμό ή στον ίδιο τον προορισμό . Ο κόμβος αυτός (του οποίου το "ύψος" δεν είναι NULL) απαντά με ένα UPD πακέτο που περιέχει το "ύψος" του κόμβου . Κάθε κόμβος που λαμβάνει αυτό το UPD πακέτο , θέτει το "ύψος" του σε μια τιμή μεγαλύτερη κατά ένα από αυτή του κόμβου που παρήγαγε το UPD πακέτο . Στη συνέχεια , ο κόμβος αυτός εκπέμπει το δικό του UPD πακέτο . Ένας κόμβος με μεγαλύτερο "ύψος" θεωρείται upstream ενώ ένας κόμβος με μικρότερο "ύψος" θεωρείται downstream . Στη συνέχεια , ανατίθεται μια κατεύθυνση στις συνδέσεις (upstream ή downstream) βασισμένη στην σχετική μετρική "ύψος" των γειτονικών κόμβων . Με τον τρόπο αυτό δημιουργείται ένας κατευθυνόμενος άκυκλος γράφος (DAG) από την πηγή στον προορισμό (δηλαδή με ρίζα τον προορισμό) . Όλα τα μηνύματα του δικτύου ρέουν προς τα κάτω , από ένα κόμβο με μεγαλύτερο "ύψος" σε έναν άλλο με μικρότερο "ύψος" .



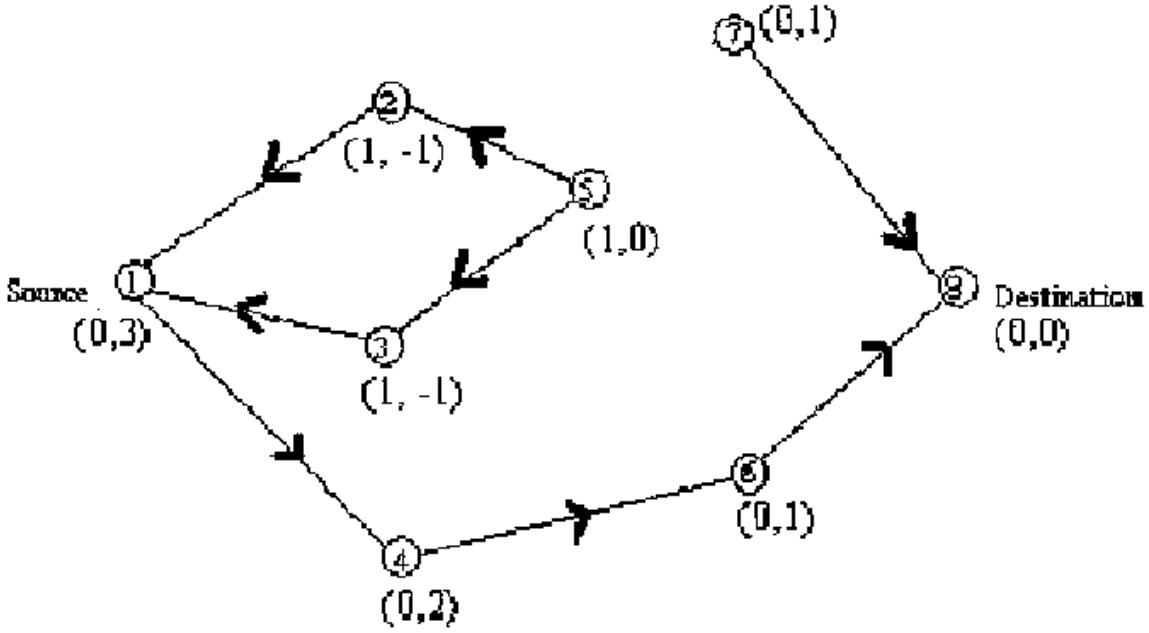
Σχήμα 2.2α. Μετάδοση του QRY μηνύματος διαμέσου του δικτύου

Στο Σχήμα 2.2 απεικονίζεται μια διαδικασία δημιουργίας διαδρομής στον TORA . Όπως φαίνεται στο Σχήμα 2.2α , ο κόμβος 5 δεν μεταδίδει το QRY από τον κόμβο 3 , αφού έχει ήδη δει και μεταδώσει το QRY από τον κόμβο 2 . Στο Σχήμα 2.2β , η πηγή (δηλαδή ο κόμβος 1) μπορεί να έχει λάβει ένα UPD πακέτο και από τον κόμβο 2 και από τον κόμβο 3, αλλά μιας και ο κόμβος 4 του δίνει μικρότερο “ύψος” , διατηρεί αυτό το “ύψος” .



Σχήμα 2.2β. Το «ύψος» του κάθε κόμβου ενημερώνεται σαν αποτέλεσμα των UPD μηνυμάτων

Όταν ένας κόμβος μετακινηθεί , η διαδρομή στο DAG σπάει και χρειάζεται η διατήρηση Διαδρομής για να ξαναδημιουργήσει ένα DAG για τον ίδιο προορισμό . Όταν η τελευταία downstream σύνδεση ενός κόμβου αποτύχει , τότε ο κόμβος παράγει ένα νέο επίπεδο αναφοράς , το οποίο έχει ως αποτέλεσμα τη μετάδοση αυτού του επιπέδου αναφοράς από τους γειτονικούς του κόμβους , συντονίζοντας έτσι αποδοτικά μια δομημένη αντίδραση στην αποτυχία , όπως φαίνεται στο Σχήμα 2.2γ . Οι συνδέσεις αντιστρέφονται για αν ανταποκρίνονται στην αλλαγή λόγω της προσαρμογής στο νέο επίπεδο αναφοράς . Αυτό έχει την ίδια επίδραση με την αντιστροφή της κατεύθυνσης ενός ή περισσοτέρων συνδέσεων όταν ένας κόμβος δεν έχει downstream συνδέσεις .



Σχήμα 2.2γ. Επανεγκατάσταση διαδρομής μετά από αποτυχία σύνδεσης της σύνδεσης 5-7 . Νέο επίπεδο αναφοράς στον κόμβο 5 .

Ο συγχρονισμός είναι ένας σημαντικός παράγοντας στον TORA , γιατί η μετρική "ύψος" εξαρτάται από το λογικό χρόνο μιας αποτυχίας σύνδεσης . Ο TORA θεωρεί ότι όλοι οι κόμβοι έχουν συγχρονισμένα ρολόγια .

Στη φάση διαγραφής Διαδρομής , ο TORA πλημμυρίζει το δίκτυο με ένα πακέτο CLR για να διαγράψει τις άκυρες διαδρομές .

Στον TORA , υπάρχει πιθανότητα να συμβούν ταλαντώσεις , κυρίως όταν πολλαπλά σύνολα συντονισμένων κόμβων ανιχνεύουν διαιρέσεις, διαγράφουν διαδρομές και κατασκευάζουν νέες διαδρομές βασισμένοι ο ένας στον άλλο , ταυτόχρονα . Επειδή ο TORA χρησιμοποιεί τον συντονισμό μεταξύ των κόμβων , το πρόβλημα της αστάθειάς του είναι παρόμοιο με το πρόβλημα του «μετρήματος στο άπειρο» των distance-vector πρωτοκόλλων δρομολόγησης , εκτός από το ότι τέτοιες ταλαντώσεις είναι προσωρινές και η σύγκλιση διαδρομής θα συμβεί τελικά .

Ιδιότητες (Πλεονεκτήματα-Μειονεκτήματα)

Πλεονεκτήματα

- Εγγυάται ότι όλες οι διαδρομές είναι χωρίς βρόχους (μόνο προσωρινοί βρόχοι μπορούν να σχηματιστούν) .

- Όπως όλα τα πρωτόκολλα που βασίζονται στην έννοια της αντιστροφής συνδέσεων , αντιδρά στις αλλαγές των συνδέσεων μέσω ενός απλού τοπικού περάσματος του κατανεμημένου αλγορίθμου .
- Υποστηρίζει πολλαπλές διαδρομές για κάθε ζεύγος αποστολέα-παραλήπτη . Έτσι δεν είναι απαραίτητη η ανακατασκευή μιας διαδρομής μέχρις ότου όλες οι γνωστές διαδρομές για έναν προορισμό θεωρηθούν άκυρες , και επομένως το εύρος ζώνης μπορεί να διατηρηθεί δυναμικά (εξαιτίας της ανάγκης για λιγότερες ανακατασκευές διαδρομών) .
- Τα μηνύματα ελέγχου περιορίζονται σε ένα μικρό αριθμό γειτονικών κόμβων και επομένως μειώνεται το overhead επικοινωνίας .
- Σε συνδυασμό με τον LAM (Lightweight Adaptive Multicast Algorithm) παρέχουν υποστήριξη multicast .
- Το πρωτόκολλο είναι κατανεμημένο , αφού δεν εξαρτάται από κάποιον κεντρικοποιημένο κόμβο και οι κόμβοι χρειάζεται να διατηρούν πληροφορίες μόνο για τους γειτονικούς τους κόμβους .
- Είναι υψηλά προσαρμόσιμο και αποδοτικό πρωτόκολλο , και κλιμακώνεται σε μεγάλα δίκτυα .
- Έχει την ικανότητα να ξαναρχίζει τη λειτουργία του και να αντιδρά στις τοπολογικές αλλαγές σπάνια , με αποτέλεσμα να μειώνεται το overhead επικοινωνίας .
- Όταν συμβούν τοπολογικές αλλαγές που απαιτούν αντίδραση , το πρωτόκολλο επανεγκαθιστά γρήγορα έγκυρες διαδρομές .
- Στην περίπτωση της διαίρεσης του δικτύου , το πρωτόκολλο ανιχνεύει τη διαίρεση και διαγράφει όλες τις άκυρες διαδρομές μέσα σε ένα πεπερασμένο χρονικό διάστημα .
- Έχει το πλεονέκτημα της δρομολόγησης από την πηγή και δημιουργεί γρήγορα ένα σύνολο διαδρομών για ένα δεδομένο προορισμό , μόνο όταν ζητηθεί .

Μειονεκτήματα

- Ενώ το γεγονός ότι ο TORA στηρίζεται στα συγχρονισμένα ρολόγια είναι μια καινοφανής ιδέα , έμφυτα περιορίζει την προσαρμοστικότητά του .
- Η επανακατασκευή μιας διαδρομής στον TORA μπορεί να μη συμβεί τόσο γρήγορα όσο σε άλλους αλγορίθμους , λόγω της πιθανότητας ταλαντώσεων κατά τη διάρκεια αυτής της περιόδου . Αυτό μπορεί να οδηγήσει σε πιθανά μεγάλες καθυστερήσεις , ενώ περιμένουμε να καθοριστούν οι νέες διαδρομές .
- Με την πάροδο του χρόνου και καθώς προχωρά η διαδικασία αντιστροφής των συνδέσεων , το προσανατολισμένο στον προορισμό DAG μπορεί να γίνει λιγότερο βέλτιστα κατευθυνόμενο , από ότι ήταν όταν δημιουργήθηκε .

- Το γράφημα έχει ρίζα τον προορισμό , ο οποίος έχει το μικρότερο “ύψος” . Εντούτοις , ο αρχικοποιητής του QRY μηνύματος (κόμβος - πηγή) δεν έχει απαραίτητα το μεγαλύτερο δυνατό “ύψος”. Αυτό μπορεί να οδηγήσει στην κατάσταση , όπου μπορεί να είναι δυνατές πολλές διαδρομές από την πηγή στον προορισμό , αλλά μόνο μια να ανακαλύπτεται . Ο λόγος που οδηγεί σε αυτό είναι ότι το “ύψος” βασίζεται αρχικά στην απόσταση σε hops από τον προορισμό .
- Το πρόσθετο κόστος του TORA εξαιτίας της χρήσης του **IMEP** είναι μεγάλο .

2.4 AODV

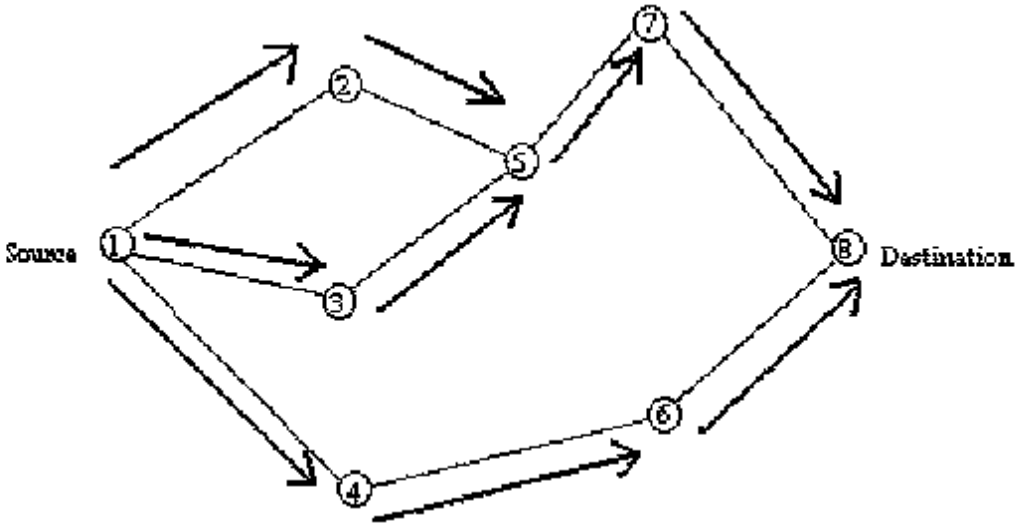
Το AODV πρωτόκολλο δρομολόγησης δημιουργείται με βάση τον DSDV αλγόριθμο . Το AODV είναι μια βελτίωση του DSDV , αφού τυπικά ελαχιστοποιεί τον αριθμό των εκπομπών που απαιτούνται , δημιουργώντας διαδρομές όταν απαιτούνται , σε αντίθεση με τον DSDV που διατηρεί μια πλήρη λίστα των διαδρομών . Οι συγγραφείς του AODV το κατατάσσουν σαν ένα απλό σύστημα απόκτησης διαδρομής όταν απαιτείται , αφού οι κόμβοι που δε βρίσκονται στο επιλεγμένο μονοπάτι δε διατηρούν πληροφορίες δρομολόγησης , ούτε συμμετέχουν σε ανταλλαγές πινάκων δρομολόγησης .

Κάθε κόμβος του δικτύου διατηρεί ένα πίνακα Δρομολόγησης , κάθε εγγραφή του οποίου περιέχει τις ακόλουθες πληροφορίες :

- Την IP διεύθυνση του προορισμού
- Το sequence number του προορισμού
- Τον αριθμό των hops μέχρι τον προορισμό
- Το επόμενο βήμα-κόμβο , το οποίο έχει επιλεγεί για την αποστολή πακέτων στον προορισμό μέσω αυτής της διαδρομής
- Το χρόνο για τον οποίο η διαδρομή θεωρείται έγκυρη (lifetime)
- Τους γειτονικούς κόμβους , οι οποίοι χρησιμοποιούν ενεργά αυτή τη διαδρομή
- Ένα buffer (Request Buffer) , ο οποίος εξασφαλίζει ότι μια αίτηση επεξεργάζεται μόνο μια φορά

Όταν ένας κόμβος-πηγή επιθυμεί να στείλει ένα μήνυμα σε κάποιον κόμβο-προορισμό , για να βρει ένα μονοπάτι για τον προορισμό αυτό , ξεκινά μια διαδικασία ανεύρεσης μονοπατιού για να τον εντοπίσει . Εκπέμπει ένα πακέτο αίτησης διαδρομής (RREQ) στους γείτονες του . Αυτοί με τη σειρά τους προωθούν την αίτηση στους γείτονές τους και ούτω καθεξής , μέχρι να φτάσει σε έναν ενδιάμεσο κόμβο που έχει μια πρόσφατη διαδρομή για τον προορισμό , ή μέχρι να φτάσει στον προορισμό (Σχήμα 2.3α) . Ένας κόμβος πετά ένα πακέτο αίτησης διαδρομής που έχει ξαναδεί . Ο AODV χρησιμοποιεί τους sequence

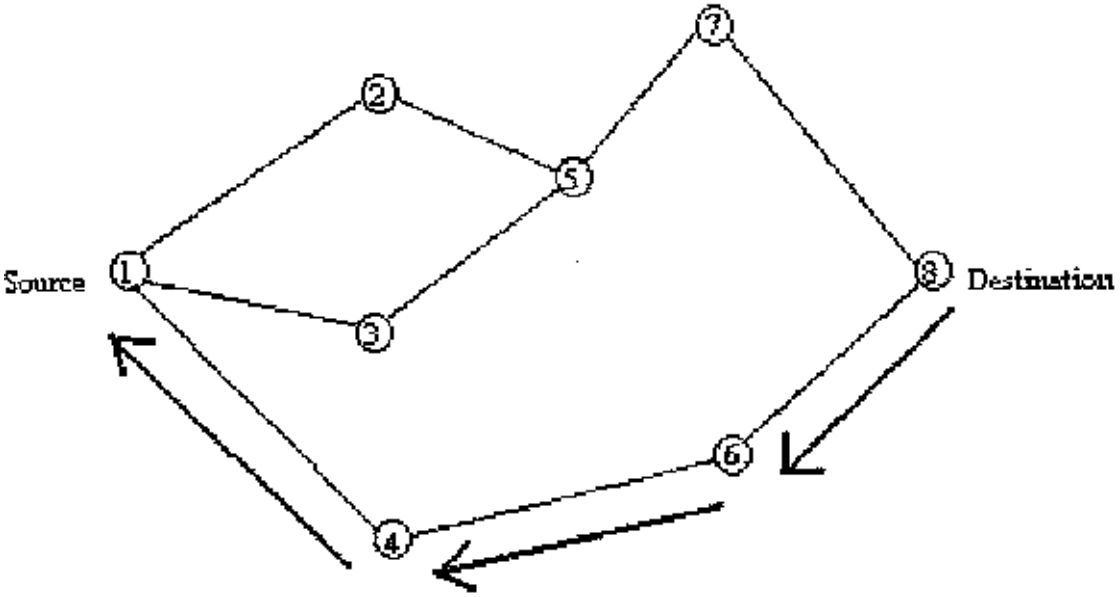
numbers των προορισμών (οι οποίοι αλλάζουν όταν συμβεί κάτι στην περιβάλλουσα περιοχή) για να εξασφαλίσει ότι όλες οι διαδρομές δεν περιέχουν βρόχους (loop-free) και περιέχουν τις πιο πρόσφατες πληροφορίες διαδρομών . Κάθε κόμβος διατηρεί ένα δικό του sequence number , καθώς και ένα ID εκπομπής .Το ID αυτό αυξάνεται για κάθε RREQ που ξεκινά ο κόμβος , και μαζί με την IP διεύθυνση του κόμβου , προσδιορίζουν μοναδικά ένα RREQ. Μαζί με το δικό του sequence number και το ID εκπομπής , ο κόμβος-πηγή συμπεριλαμβάνει στο RREQ τον πιο πρόσφατο sequence number που έχει για τον προορισμό . Ενδιάμεσοι κόμβοι μπορούν να απαντήσουν στο RREQ μόνο αν έχουν μια διαδρομή για τον προορισμό , της οποίας ο αντίστοιχος sequence number του προορισμού είναι μεγαλύτερος ή ίσος με αυτόν που περιέχεται στο RREQ .



Σχήμα 2.3α. Μετάδοση του πακέτου αίτησης διαδρομής (RREQ)

Κατά τη διάρκεια της διαδικασίας της προώθησης του RREQ , οι ενδιάμεσοι κόμβοι καταγράφουν στον πίνακα Δρομολόγησής τους τη διεύθυνση του γείτονα από τον οποίο ήρθε το πρώτο αντίγραφο της αίτησης. Η πληροφορία αυτή χρησιμοποιείται για την κατασκευή του αντίστροφου μονοπατιού για το πακέτο απάντησης διαδρομής (RREP) . Όταν το RREQ φτάσει στον προορισμό ή σε έναν ενδιάμεσο κόμβο με μια αρκετά «φρέσκια» διαδρομή , ο προορισμός ή ο ενδιάμεσος κόμβος απαντά μεταδίδοντας ένα RREP πακέτο πίσω προς το γείτονα από τον οποίο έλαβε αρχικά το RREQ (Σχήμα 2.3β) . Ενόσω το RREP προωθείται προς τα πίσω κατά μήκος του αντιστρόφου μονοπατιού , οι κόμβοι κατά μήκος του μονοπατιού αυτού δημιουργούν στους πίνακες Δρομολόγησής τους εγγραφές διαδρομών προς τα μπρος , που δείχνουν προς τον κόμβο από τον

οποίο ήρθε το RREP . Αυτές οι προς τα μπρος εγγραφές διαδρομών δείχνουν την ενεργή διαδρομή προς τα μπρος . Με κάθε εγγραφή διαδρομής είναι συσχετισμένο ένα χρονόμετρο διαδρομής , το οποίο θα προκαλέσει τη διαγραφή της αντίστοιχης εγγραφής , αν αυτή δε χρησιμοποιείται μέσα στον προκαθορισμένο χρόνο ζωής . Επειδή το RREP προωθείται κατά μήκος του μονοπατιού που έχει εγκατασταθεί από το RREQ, ο AODV υποστηρίζει μόνο τη χρήση συμμετρικών συνδέσεων . Αν η απάντηση (RREP) δεν φτάσει μέσα σε ένα συγκεκριμένο χρονικό διάστημα , ο κόμβος μπορεί να επαναλάβει την αποστολή του RREQ μηνύματος , ή να υποθέσει ότι δεν υπάρχει κάποια διαδρομή προς τον απαιτούμενο προορισμό.



Σχήμα 2.3β. Το μονοπάτι που ακολουθείται από το πακέτο απάντησης διαδρομής (RREP)

Οι διαδρομές διατηρούνται ως εξής : Αν ένας κόμβος-πηγή μετακινηθεί , τότε μπορεί να ξαναρχίσει το μηχανισμό ανακάλυψης διαδρομής , για να βρει μια νέα διαδρομή για τον προορισμό . Αν ένας ενδιάμεσος κόμβος κατά μήκος της διαδρομής μετακινηθεί , τότε ο προηγούμενος γείτονάς του παρατηρεί τη μετακίνηση (την αποτυχία της σύνδεσης) και μεταδίδει ένα μήνυμα ειδοποίησης αποτυχίας σύνδεσης (ένα RREP με άπειρη μετρική) σε κάθε έναν από τους ενεργούς προηγούμενους γείτονές του , για να τους ενημερώσει για την εξάλειψη αυτού του τμήματος της διαδρομής . Αυτοί οι κόμβοι με τη σειρά τους , μεταδίδουν το μήνυμα ειδοποίησης στους προηγούμενους γείτονές τους και ούτω καθεξής μέχρι να φτάσει στον κόμβο-πηγή . Η πηγή μπορεί τότε να

επιλέξει να ξαναρχίσει την ανακάλυψη διαδρομής για τον προορισμό αυτό , αν η διαδρομή είναι ακόμα επιθυμητή .

Μια επιπλέον άποψη του πρωτοκόλλου είναι η χρήση των “hello” μηνυμάτων (ένα ειδικό τύπο RREP μηνύματος) , τα οποία στέλνονται περιοδικά από ένα κόμβο προς όλους τους άμεσους γείτονές του . Αυτά τα μηνύματα έχουν σα στόχο τη διαρκή ενημέρωση κάθε κόμβου για άλλους κόμβους που βρίσκονται στη γειτονιά του . Οι γείτονες που χρησιμοποιούν διαδρομές μέσω του συγκεκριμένου κόμβου θα εξακολουθήσουν να θεωρούν τις διαδρομές σαν έγκυρες .Τα “hello” μηνύματα μπορούν να χρησιμοποιηθούν για να διατηρηθεί η τοπική συνδετικότητα ενός κόμβου . Παρόλα αυτά , η χρήση τους δεν απαιτείται . Οι κόμβοι ελέγχουν τη μετάδοση πακέτων δεδομένων , έτσι ώστε να επιβεβαιώσουν ότι μπορούν ακόμη να φτάσουν στον επόμενο κόμβο . Αν μια τέτοια μετάδοση δεν «ακουστεί» , ο κόμβος μπορεί να χρησιμοποιήσει οποιαδήποτε από έναν αριθμό τεχνικών , συμπεριλαμβανομένης και της λήψης “hello” μηνυμάτων , έτσι ώστε να καθορίσει αν ο επόμενος κόμβος βρίσκεται εντός της ακτίνας επικοινωνίας . Αν τα “hello” μηνύματα σταματήσουν να φτάνουν από ένα συγκεκριμένο κόμβο , τότε οι γείτονές του μπορούν να υποθέσουν ότι έχει απομακρυνθεί εκτός ακτίνας επικοινωνίας , και να σημαδέψουν τη σύνδεση αυτή σα σπασμένη . Ταυτόχρονα , θα πρέπει να γνωστοποιηθεί η αποτυχία της σύνδεσης αυτής σε όλους τους επηρεαζόμενους κόμβους . Τα “hello” μηνύματα μπορούν να δημιουργήσουν μια λίστα των κόμβων από τους οποίους έχει ακούσει ένας κόμβος , αποδίδοντας έτσι μεγαλύτερη γνώση της συνδετικότητας του δικτύου .

Ιδιότητες (Πλεονεκτήματα-Μειονεκτήματα)

Πλεονεκτήματα

- Πλεονεκτεί σε σχέση με τους κλασσικούς αλγόριθμους δρομολόγησης , όπως ο Distance Vector και ο Link State , στο ότι έχει περιορίσει σημαντικά τον αριθμό των μηνυμάτων δρομολόγησης μέσα στο δίκτυο.
- Με τη χρήση των sequence numbers εξασφαλίζεται ότι μια διαδρομή είναι πρόσφατη και δεν περιέχει βρόχους (loops) .
- Προσθέτει τη δυνατότητα multicast , η οποία αυξάνει την απόδοση σημαντικά όταν ένας κόμβος επικοινωνεί με πολλούς .
- Το πρωτόκολλο είναι κατανεμημένο , αφού δεν εξαρτάται από κάποιον κεντρικοποιημένο κόμβο .
- Διατηρεί χαμηλό το overhead δρομολόγησης , αφού τόσο τα πακέτα αναζήτησης διαδρομής , όσο και τα πακέτα απάντησης διαδρομής , περιέχουν μικρό όγκο πληροφοριών .
- Με την αποστολή “hello” μηνυμάτων μπορεί να διατηρηθεί η τοπική συνδετικότητα του κάθε κόμβου .

Μειονεκτήματα

- Η χρήση των sequence numbers μπορεί να δημιουργήσει προβλήματα, αν για παράδειγμα πάψουν να είναι συγχρονισμένοι .
- Υποστηρίζει μόνο μια διαδρομή για κάθε προορισμό .
- Απαιτεί συμμετρικές συνδέσεις μεταξύ των κόμβων και για το λόγο αυτό δε μπορεί να χρησιμοποιήσει διαδρομές με μη-συμμετρικές συνδέσεις .
- Αν προκύψει μια αποτυχία σύνδεσης κατά μήκος ενός μονοπατιού , ο αλγόριθμος ανακάλυψης διαδρομής πρέπει να ξανακληθεί από την πηγή για να βρεθεί ένα νέο μονοπάτι για τον προορισμό . Δε γίνεται καμιά προσπάθεια να χρησιμοποιηθεί τμηματική ανάκτηση διαδρομής , δηλαδή να επιτραπεί στους ενδιάμεσους κόμβους να προσπαθήσουν να ξαναφτιάξουν μόνοι τους τη διαδρομή . Αυτό μπορεί να οδηγήσει σε μεγαλύτερους χρόνους ανακατασκευής διαδρομής . [11]

  - Όμως η προσπάθεια και η αποτυχία ενός ενδιάμεσου κόμβου να ξαναφτιάξει μια διαδρομή , θα προκαλέσει μεγαλύτερη καθυστέρηση από ότι αν ο κόμβος-πηγή είχε προσπαθήσει να την ξαναφτιάξει αμέσως μόλις αντιλήφθηκε τη σπασμένη σύνδεση .

Performance Evaluation Measures

- Κλάσμα παράδοσης πακέτων (Packet Delivery fraction) .
- Μέση καθυστέρηση πακέτων από άκρη σε άκρη (Average end-to-end packet delay) .
- Ομαλοποιημένο φορτίο δρομολόγησης (Normalized routing load) .
- Throughput .

Οι μεταβλητές που χρησιμοποιούνται είναι οι εξής :

- Κινητικότητα .
- Αριθμός των ζευγαριών κόμβων που επικοινωνούν .
- Προσφερόμενο φορτίο (Offered load).

2.5 WRP

Το WRP είναι ένα πρωτόκολλο το οποίο βασίζεται σε πίνακες και έχει σαν σκοπό τη διατήρηση πληροφοριών δρομολόγησης μεταξύ όλων των κόμβων του δικτύου. Κάθε κόμβος στο δίκτυο είναι υπεύθυνος για τη διατήρηση τεσσάρων πινάκων :

- Πίνακας αποστάσεων (Distance Table)
- Πίνακας δρομολόγησης (Routing Table)
- Πίνακας κόστους συνδέσεων (Link-cost table)
- Λίστα επαναμετάδοσης μηνυμάτων (Message Retransmission list)

Ο πίνακας αποστάσεων ενός κόμβου x περιέχει την απόσταση κάθε κόμβου προορισμού y μέσω κάθε γείτονα z του x . Επίσης περιέχει τον επόμενο γείτονα του z μέσω του οποίου πραγματοποιείται αυτό το μονοπάτι .

Ο πίνακας δρομολόγησης ενός κόμβου x περιέχει την απόσταση κάθε προορισμού y από τον x καθώς και τον προηγούμενο και τον επόμενο κόμβο του κόμβου x σε αυτό το μονοπάτι . Επίσης περιέχει μια ετικέτα η οποία προσδιορίζει αν η εγγραφή είναι ένα απλό μονοπάτι , ένα loop ή άκυρη .Η αποθήκευση του προηγούμενου και του επόμενου κόμβου στον πίνακα είναι ωφέλιμη στην ανίχνευση βρόγχων και στο να αποφεύγουμε προβλήματα «μετρήματος στο άπειρο» .

Ο πίνακας κόστους συνδέσεων περιέχει το κόστος της σύνδεσης με κάθε γείτονα του κόμβου και τον αριθμό των timeouts που συνέβησαν από τότε που ένα μήνυμα χωρίς λάθη ελήφθη από αυτό το γείτονα .

Η λίστα επαναμετάδοσης μηνυμάτων περιέχει πληροφορίες έτσι ώστε κάθε κόμβος να ξέρει ποιοι από τους γείτονές του δεν έχουν αναγνωρίσει το μήνυμα ενημέρωσής του και επομένως να επαναμεταδώσει το μήνυμα ενημέρωσης σε αυτούς τους γείτονες . Πιο συγκεκριμένα , κάθε εγγραφή της λίστας επαναμετάδοσης μηνυμάτων περιέχει τον sequence number του μηνύματος ενημέρωσης , ένα μετρητή επαναμετάδοσης , ένα διάνυσμα flags, με μια εγγραφή για κάθε γείτονα ,το οποίο απαιτεί αναγνώριση και μια λίστα με τις ενημερώσεις που εστάλησαν με το μήνυμα ενημέρωσης . Η MRL καταγράφει ποιες ενημερώσεις σε ένα μήνυμα ενημέρωσης θα πρέπει να επαναμεταδοθούν και ποιοι γείτονες θα πρέπει να αναγνωρίσουν την επαναμετάδοση .

Οι κόμβοι ανταλλάσσουν τους πίνακες δρομολόγησής τους με τους γείτονές τους χρησιμοποιώντας μηνύματα ενημέρωσης περιοδικά , καθώς και σε περιπτώσεις αλλαγής συνδέσεων . Ένα μήνυμα ενημέρωσης στέλνεται μόνο μεταξύ γειτονικών κόμβων και περιέχει μια λίστα ενημερώσεων (τον προορισμό, την απόσταση για τον προορισμό , και τον προηγούμενο κόμβο του προορισμού) , καθώς επίσης και μια λίστα ανταποκρίσεων . Κάθε κόμβος στη λίστα αυτή (η οποία δημιουργείται χρησιμοποιώντας την MRL) πρέπει να αναγνωρίσει τη λήψη του μηνύματος ενημέρωσης . Οι κόμβοι ενημερώνονται για την ύπαρξη των γειτόνων τους από τη λήψη μηνυμάτων αναγνώρισης ή από άλλα μηνύματα . Αν δεν υπάρχει καμία αλλαγή στον πίνακα δρομολόγησης του μετά την τελευταία ενημέρωση , κάθε κόμβος θα πρέπει να στείλει ένα κενό “Hello” μήνυμα μέσα σε μια καθορισμένη χρονική περίοδο, έτσι ώστε να επιβεβαιώσει την ύπαρξη σύνδεσης . Αλλιώς η έλλειψη μηνυμάτων από ένα κόμβο , θα είναι ενδεικτική της αποτυχίας αυτής της σύνδεσης , γεγονός που μπορεί να οδηγήσει σε εσφαλμένο «συναγερμό» . Όταν ένας κόμβος δέχεται ένα “Hello” μήνυμα από ένα νέο κόμβο , τότε ο κόμβος αυτός προστίθεται στον πίνακα δρομολόγησης του αρχικού κόμβου , ο οποίος με τη σειρά του στέλνει στο νέο αυτό κόμβο ένα αντίγραφο των πληροφοριών του πίνακα δρομολόγησής του .

Κάθε κόμβος , όταν δέχεται ένα μήνυμα ενημέρωσης , τροποποιεί κατάλληλα τον πίνακα αποστάσεων και ψάχνει για καλύτερα μονοπάτια χρησιμοποιώντας τις νέες πληροφορίες. Κάθε νέο μονοπάτι που βρίσκεται έτσι ,στέλνεται πίσω στους αρχικούς κόμβους έτσι ώστε να μπορούν να ενημερώσουν και αυτοί τους πίνακές τους . Ο κόμβος ενημερώνει επίσης και τον πίνακα δρομολόγησής του , αν το νέο μονοπάτι είναι καλύτερο από το υπάρχον σε αυτόν μονοπάτι .Όταν δέχεται ένα ACK , ο πίνακας ενημερώνει την MRL του .

Ένα μοναδικό χαρακτηριστικό αυτού του αλγορίθμου, είναι ότι κάθε κόμβος επαληθεύει τη συνέπεια όλων των γειτόνων του , κάθε φορά που ανιχνεύει μια αλλαγή σε σύνδεση οποιουδήποτε από τους γείτονές του . Η κατά αυτόν τον τρόπο επαλήθευση της συνέπειας βοηθάει στην εξάλειψη των βρόχων κατά έναν καλύτερο τρόπο και παρέχει πιο γρήγορη σύγκλιση σε μια διαδρομή , όταν συμβαίνει μια αποτυχία σύνδεσης .

Ιδιότητες (Πλεονεκτήματα-Μειονεκτήματα)

Πλεονεκτήματα

- Αν και ανήκει στην τάξη των αλγορίθμων ανεύρεσης μονοπατιού , έχει ένα πλεονέκτημα σε σχέση με αυτούς , αφού αποφεύγει το πρόβλημα της δημιουργίας προσωρινών βρόχων δρομολόγησης (loops) , μέσω της επαλήθευσης της συνέπειας όλων των γειτόνων του κάθε κόμβου , κάθε φορά που αυτός ανιχνεύει μια αλλαγή σε σύνδεση οποιουδήποτε από τους γείτονές του .
- Οδηγεί σε πιο γρήγορη σύγκλιση .
- Αποφεύγει το πρόβλημα του «μετρήματος στο άπειρο» με την αποθήκευση του προηγούμενου και του επόμενου κόμβου ενός κόμβου x , στο μονοπάτι προς ένα κόμβο y , στον πίνακα δρομολόγησης του x .

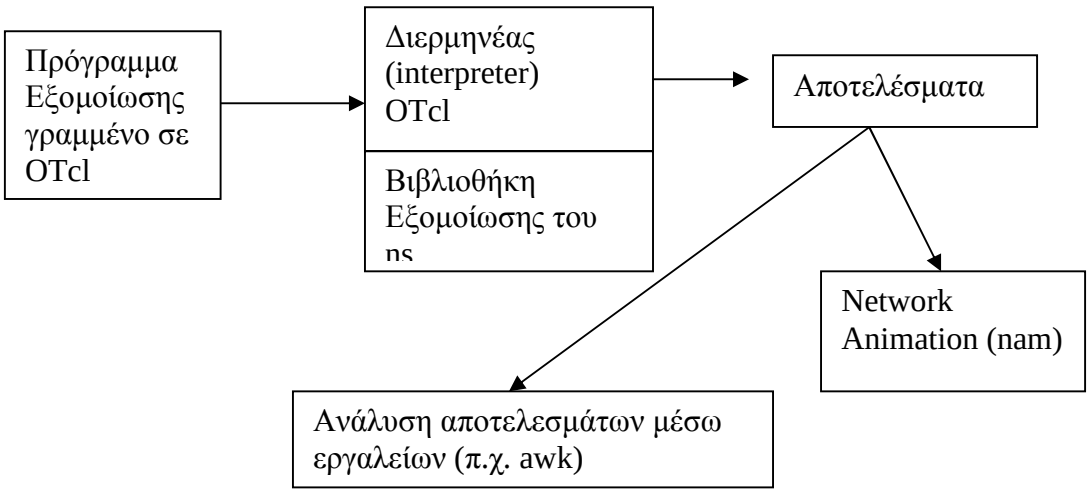
Μειονεκτήματα

- Απαιτεί κάθε κόμβος να διατηρεί τέσσερις πίνακες , το οποίο μπορεί να οδηγήσει σε σημαντικές απαιτήσεις μνήμης , ειδικά όταν ο αριθμός των κόμβων του δικτύου είναι μεγάλος .
- Απαιτεί τη χρήση μηνυμάτων “Hello” , όταν δεν υπάρχουν πρόσφατες μεταδόσεις μηνυμάτων από ένα δεδομένο κόμβο . Αυτά τα μηνύματα καταναλώνουν εύρος ζώνης και δεν επιτρέπουν σε έναν κόμβο να εισέλθει σε sleep mode .

3. Network Simulator 2 (ns2)

3.1 Περιγραφή του ns

Το ns είναι ένας εξομοιωτής δικτύων υλοποιήθηκε για πρώτη φορά το 1989. Η τωρινή του έκδοση, ns2, χρησιμοποιείται ευρέως για ερευνητικούς σκοπούς ιδιαίτερα όσον αφορά το mobile routing. Το ns2 είναι γραμμένο σε C++ και OTcl. Η βασική δομή του ns δίνεται στην παρακάτω εικόνα.



Εικόνα: Βασική Δομή του ns.

Όπως φαίνεται στην παραπάνω εικόνα, ο χρήστης τροφοδοτεί το ns με ένα αρχείο εισόδου γραμμένο σε OTcl (OTcl script). Το script περιγράφει την τοπολογία του δικτύου, αρχικοποιεί τους χειριστές συμβάντων (κώδικας δηλαδή που τρέχει όταν ενεργοποιηθούν διάφορα συμβάντα, όπως π.χ. η έναρξη αποστολής πακέτων) και «πληροφορεί» το διερμηνέα για τις στιγμές εκκίνησης αποστολής πακέτων των κόμβων. Το ns, το οποίο βασικά είναι ένας διερμηνέας (interpreter) της γλώσσας OTcl, δέχεται το αρχείο εισόδου και το εκτελεί χρησιμοποιώντας τις βιβλιοθήκες που έχει για να κληθούν οι ενσωματωμένες συναρτήσεις τις οποίες χρησιμοποιεί ο χρήστης στο script εισόδου.

Ένα βασικό συστατικό του ns είναι ο χειριστής συμβάντων (event scheduler). Ένα συμβάν για το ns είναι ένα αναγνωριστικό (id) πακέτου το οποίο είναι μοναδικό για ένα πακέτο με προγραμματισμένο χρόνο κι ένας δείκτης σε ένα αντικείμενο το οποίο είναι υπεύθυνο (διαχειρίζεται) το συμβάν. Ένας χειριστής συμβάντων παρακολουθεί τον χρόνο κατά τη διάρκεια της

εξομοίωσης και πυροδοτεί όλα τα συμβάντα που είναι προγραμματισμένα για τη συγκεκριμένη χρονική στιγμή και που βρίσκονται στην ουρά (queue) συμβάντων.

Το ns2 είναι υλοποιημένο για την πλατφόρμα Unix/Linux. Προκειμένου να τρέξει σε άλλα λειτουργικά συστήματα, όπως π.χ. MS Windows, είναι απαραίτητο να υπάρχει ένα πρόγραμμα εξομοίωσης των παραπάνω αυτών λειτουργικών. Για τα MS Windows, το πρόγραμμα εξομοίωσης ονομάζεται cygwin. Στη συνέχεια περιγράφεται η διαδικασία εγκατάστασης του ns2, καθώς και ο τρόπος με τον οποίο εκτελείται

3.2 Εγκατάσταση του ns2

Στο παρόν κεφάλαιο θα δοθεί η διαδικασία με την οποία γίνεται εγκατάσταση του ns2 σε περιβάλλον MS Windows. Στο κάθε υποκεφάλαιο θα δοθεί η διαδικασία για κάθε βήμα της εγκατάστασης ξεχωριστά

3.2.1 Εγκατάσταση του Cygwin

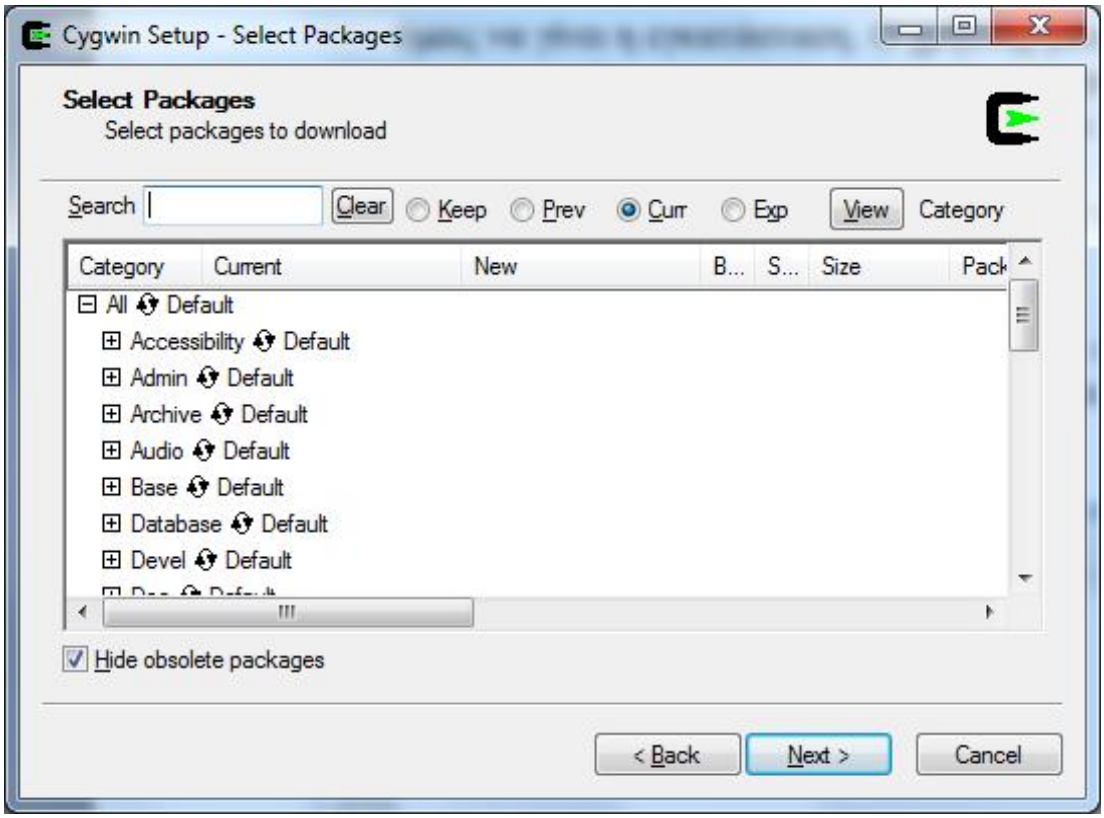
Κατ'αρχάς, χρειάζεται να εγκατασταθεί η τελευταία έκδοση του cygwin. Το Cygwin διατίθεται δωρεάν στην ηλεκτρονική διεύθυνση:
http://www.cygwin.com/
Επιλέγοντας το σύνδεσμο για εγκατάσταση, ξεκινά το πρόγραμμα εγκατάστασης του Cygwin:

Εικόνα 3.1 Αρχικό πλαίσιο του προγράμματος εγκατάστασης του Cygwin

Πατώντας «Επόμενο» ο χρήστης μπορεί να επιλέξει τον τρόπο με τον οποίο επιθυμεί να εγκαταστήσει το cygwin. Οι δυνατοί τρόποι είναι τρεις:

1. Εγκατάσταση από το internet. Το πρόγραμμα εγκατάστασης του Cygwin θα κατεβάσει τα απαραίτητα αρχεία από το διαδίκτυο και θα εγκαταστήσει αυτόματα την εφαρμογή στον υπολογιστή
2. Κατέβασμα των αρχείων χωρίς εγκατάσταση. Θα κατέβουν τα αρχεία εγκατάσταστης του cygwin χωρίς όμως να γίνει η εγκατάσταση. Ο χρήστης μπορεί αργότερα να εκγαταστήσει το cygwin από αυτά τα αρχεία.
3. Εγκατάσταση από τοπικό κατάλογο. Γίνεται εγκατάσταση του Cygwin από τοπικό κατάλογο στον οποίο έχουν αποθηκευτεί τα αρχεία εγκατάστασης με τη διαδικασία του τρόπου 2.

Αφού ο χρήστης επιλέξει σε ποιον κατάλογο θα εγκατασταθεί το cygwin, η εφαρμογή εγκατάστασης θα ζητήσει ποια πακέτα θα πρέπει να εγκατασταθούν:



Εικόνα 3.2. Επιλογή πακέτων για εγκατάσταση

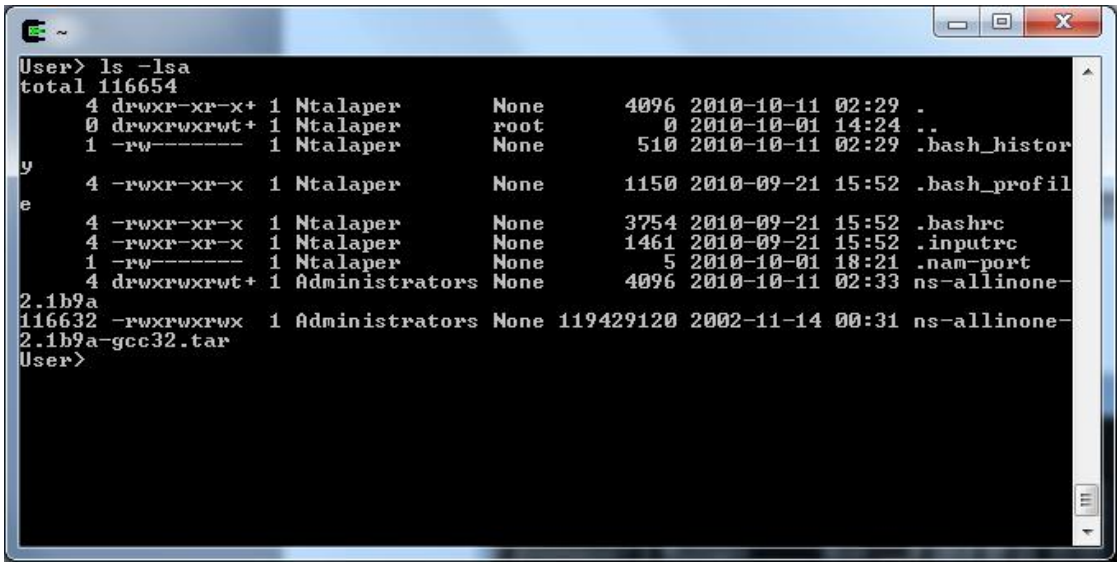
Μία πλήρης εγκατάσταση του cygwin απαιτεί πολύ χώρο στο σκληρό δίσκο καθώς και αρκετό χρόνο εγκατάστασης. Καλό είναι λοιπόν, να επιλεχθούν μόνο αυτά τα πακέτα που απαιτούνται για την εγκατάσταση του ns2. Τα πακέτα αυτά είναι τα εξής:

- gcc
- gcc-g++
- gawk
- tar
- gzip
- make
- patch
- perl
- w32api
- diff
- X

Αφού επιλεγούν τα παραπάνω πακέτα, ο χρήστης μπορεί να ξεκινήσει την διαδικασία εγκατάστασης. Η εγκατάσταση, υποθέτοντας ότι έχουν χρησιμοποιηθεί οι προεπιλεγμένες τιμές, θα εγκαταστήσει το cygwin στην τοποθεσία C:\Cygwin. Σ'αυτή την τοποθεσία βρίσκεται και το αρχείο

Cygwin.bat με το οποίο ξεκινά το cygwin. Με διπλό κλικ πάνω στο συγκεκριμένο αρχείο ο χρήστης οδηγείται στην αρχική οθόνη του cygwin

Εδώ ο χρήστης μπορεί να πληκτρολογήσει οποιαδήποτε εντολή unix. Για παράδειγμα με την εντολή ls μπορεί να εμφανίσει τα περιεχόμενα του τρέχοντα καταλόγου:

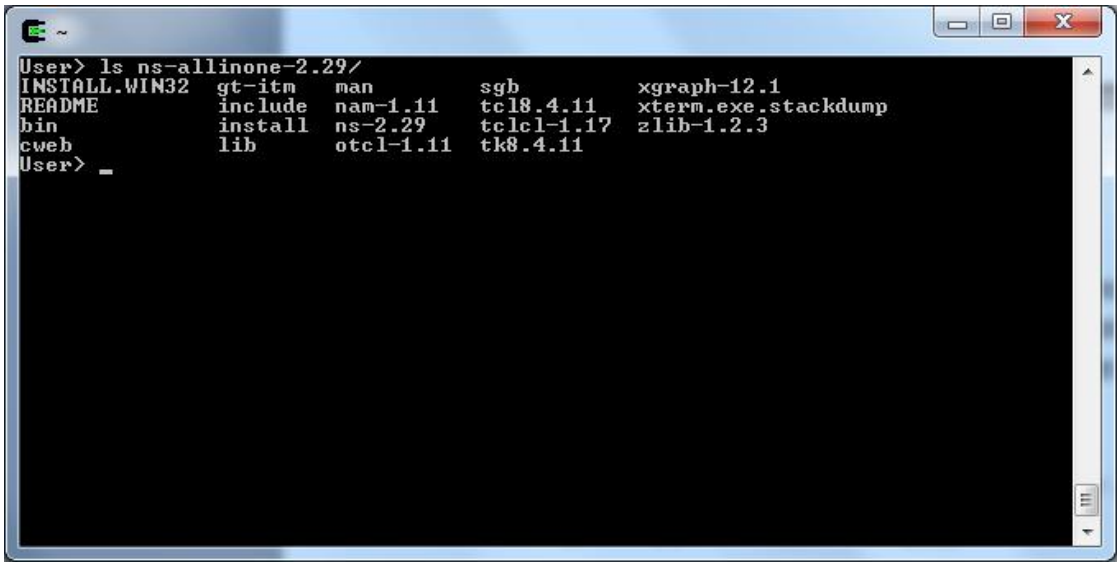


Εικόνα 3.3. Το περιβάλλον κελύφους του Cygwin. Δίνεται για παράδειγμα η εντολή ls η οποία εμφανίζει τα περιεχόμενα ενός καταλόγου.

3.2.2 Εγκατάσταση του ns2.

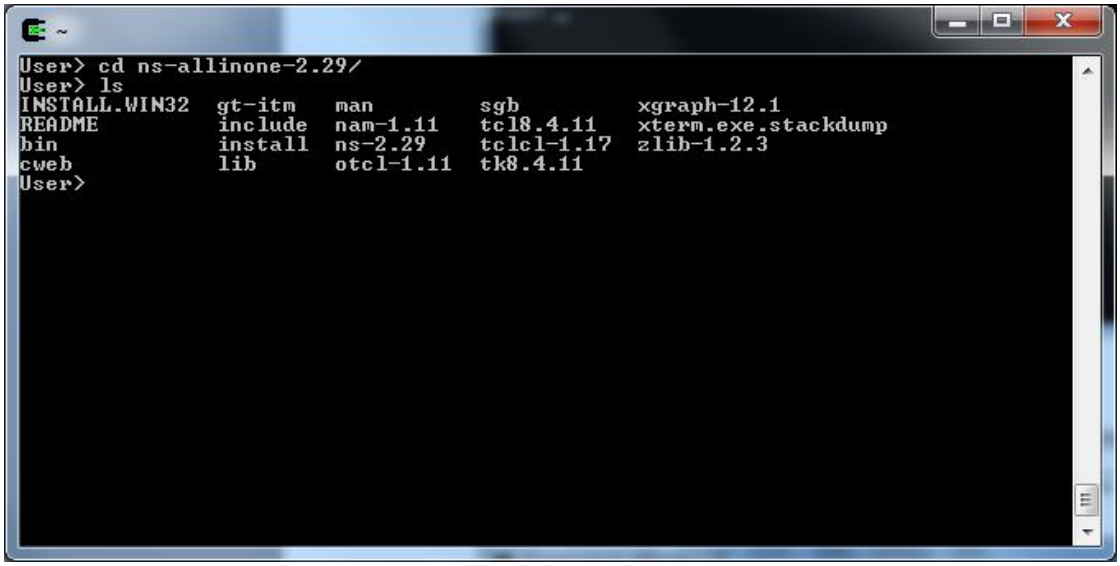
Ολόκληρη η εφαρμογή του ns2 διατίθεται δωρεάν από την παρακάτω διεύθυνση:
http://sourceforge.net/projects/nsnam/files/
Το αρχείο ns-allinone-2.34.tar.gz πρέπει να αποσυμπιεστεί με μία κατάλληλη εφαρμογή (π.χ. 7zip) στον κατάλογο χρήστη του cygwin. Αν το όνομα του χρήστη είναι «user» τότε ο κατάλογος αυτός βρίσκεται στο C:\cygwin\home\user

Από το cygwin γίνεται να επιβεβαιωθεί αν έχει γίνει σωστά η αποσυμπίεση μέσω της εντολής ls



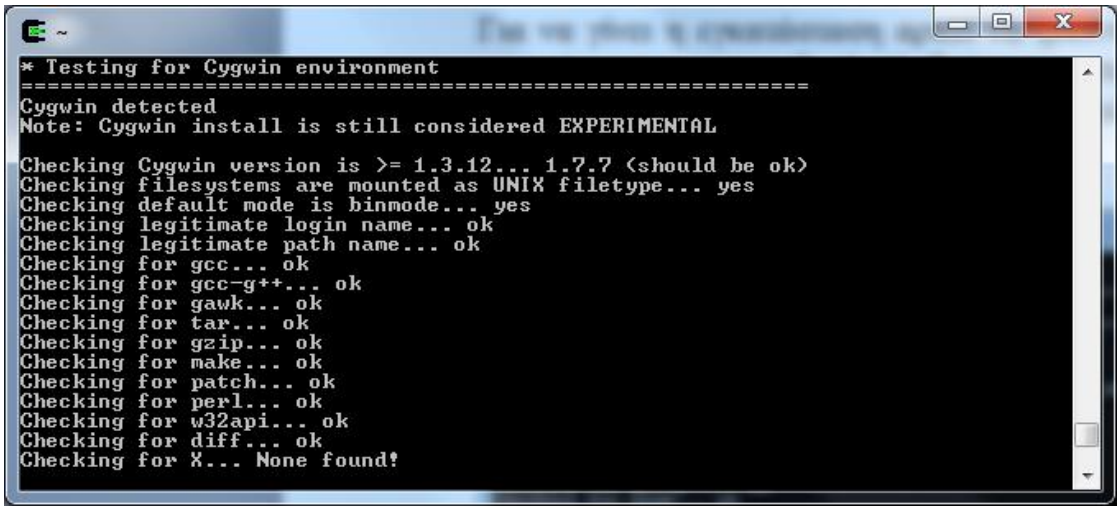
Εικόνα 3.4. Περιεχόμενα του φακέλου της εφαρμογής ns*, αμέσως μετά την αποσυμπίεση των αρχείων εγκατάστασης.*

Ο χρήστης μπορεί να μπει στον κατάλογο του ns που έχει αποσυμπιεστεί στον κατάλογό του, μέσω της εντολής cd.



Εικόνα 3.5. Περιήγηση στον φάκελο της εφαρμογής ns*. Από εδώ μπορούν να εκτελεστούν τα διάφορα* scripts *εγκατάστασης.*

Για να γίνει η εγκατάσταση αρκεί να ξεκινήσει το πρόγραμμα εγκατάστασης του ns2. Αυτό βρίσκεται στο αρχείο “install”. Άρα πληκτρολογώντας την εντολή “install”, ξεκινά και η εγκατάσταση του ns2.



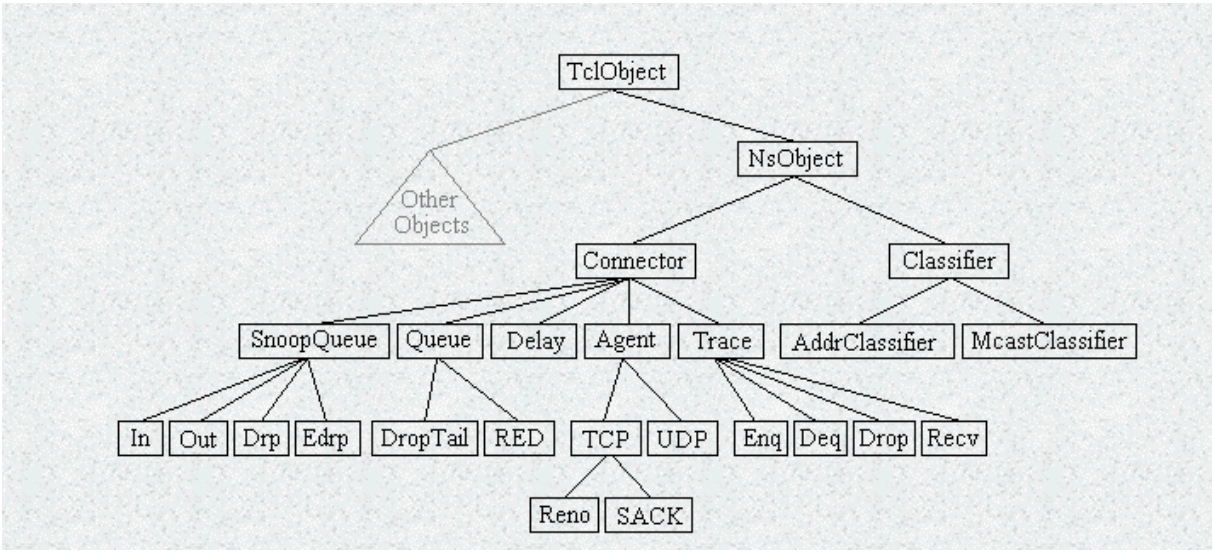
Εικόνα 3.5. Εκτέλεση του προγράμματος εγκατάστασης install. *Το* script *αρχικά ελέγχει αν ικανοποιούνται οι απαιτήσεις εγκατάστασης και μετά προχωρά στην ίδια την εγκατάσταση.*

Το πρόγραμμα εγκατάστασης ελέγχει αρχικά αν όλα τα προαπαιτούμενα πακέτα του cygwin έχουν εγκατασταθεί. Αν οποιοδήποτε πακέτο λείπει, ο χρήστης πρέπει να διακόψει τη διαδικασία εγκατάστασης πατώντας το πλήκτρο «n» και να εγκαταστήσει τα συγκεκριμένα πακέτα πριν ξαναδοκιμάσει την εγκατάσταση.

Αφού τελειώσει η εγκατάσταση, θα έχουν δημιουργηθεί διάφοροι υποκατάλογοι στον αρχικό κατάλογο του ns2. Το ίδιο το ns2 βρίσκεται στον κατάλογο «ns-2.29». Στον συγκεκριμένο κατάλογο βρίσκεται και το πρόγραμμα «validate», το οποίο ελέγχει αν το ns2 έχει εγκατασταθεί σωστά. Αν και το *validate* μπορεί να απαιτεί πολύ χρόνο μέχρι να ελέγξει όλες τις παραμέτρους του ns2, καλό είναι να εκτελεσθεί μία φορά ώστε να είναι βέβαιο ότι η εγκατάσταση έχει γίνει χωρίς προβλήματα.

3.2 Δομή του ns

Στην εικόνα 3.6 φαίνεται μια μερική ιεραρχία των Otcl κλάσεων που χρησιμοποιούνται στο ns. Όλες οι κλάσεις προέρχονται από την υπερκλάση *NsObject* από την οποία προέρχονται μέσω κληρονομικότητας όλες οι βασικές κλάσεις που χρησιμοποιούνται στο *ns* (*Agent, Queue κλπ).*



Εικόνα 3.6 Μερική ιεραρχία κλάσεων του ns.

3.2 Εισαγωγή στην OTcl

Ένα πρόγραμμα σε NS είναι στην ουσία ένα πρόγραμμα γραμμένο σε OTcl το οποίο συνδέεται με τις κατάλληλες βιβλιοθήκες εξομοίωσης δικτύου. Σ' αυτή την παράγραφο θα επιχειρηθεί μία εισαγωγή στην OTcl. Η εισαγωγή αυτή θα γίνει μέσω παραδειγμάτων κώδικα.

Σαν πρώτο παράδειγμα παρουσιάζονται τα βασικά στοιχεία της γλώσσας (συναρτήσεις, μεταβλητές, βρόχοι):

```
#definition of procedure myproc
proc myproc{}  {
        set a 6;
        set b 9;
        set s [expr $a + $b];
        set m [expr[exrp[$a - $b] * $s];
        for {set i 0} {$i < 10} {incr $i} {
                if {$i < 5} {
                        puts "i < 5, sum = [expr $s + $i]"
                } else {
                        puts "i>=5, diff = [expr $s-$m]"
                }
        }
}

#calling myproc
myproc
```

Μία συνάρτηση δηλώνεται με τη λέξη κλειδί *proc*. Ακολουθεί το όνομα της συνάρτησης και τυχόν ορίσματα που δέχεται μέσα σε αγκύλες (στο συγκεκριμένο παράδειγμα η συνάρτηση δε δέχεται ορίσματα). Η λέξη κλειδί *set* χρησιμοποείται για να δώσει τιμή σε μία μεταβλητή. Η τιμή μπορεί να είναι σταθερά, όπως στις μεταβλητές *a* και *b* του παραδείγματος οι οποίες παίρνουν τιμή *6* και *9* αντίστοιχα, ή τιμές που προκύπτουν από υπολογισμό εκφράσεων. Στη δεύτερη περίπτωση χρησιμοποιείται η λέξη κλειδί *expr* μέσα σε άγκιστρα, η οποία δίνει την οδηγία στον ερμηνευτεί (*interpreter)* να υπολογίζει την έκφραση που ακολουθεί. Για να πάρουμε την τιμή μίας μεταβλητής πρέπει να μπει σαν πρόθεμα ο χαρακτήρας *$* πριν από το αναγνωριστικό της. Η εντολή *puts* βγάζει στην έξοδο τη συμβολοσειρά που βρίσκεται ανάμεσα στα διπλά εισαγωγικά που την ακολουθούνε.

Παρακάτω φαίνεται η έξοδος του προγράμματος του παραδείγματος.

i<5,sum=15
i<5,sum=16
i<5,sum=76
i<5,sum=18
i<5,sum=19
i>=5,diff=150
i>=5,diff=150
i>=5,diff=150
i>=5,diff=150
i>=5,diff=150

Η OTcl είναι μία αντικειμενοστραφής επέκταση της παλαιότερης Tcl κι άρα τα περισσότερα από τα δυνατά της χαρακτηριστικά σχετίζονται με τις ιδιότητες μία αντικειμενοστραφούς γλώσσας (κλάσεις, κληρονομικότητα κλπ.) Στο παρακάτω παράδειγμα φαίνονται τα κυριώτερα αντικειμενοστραφή χαρακτηριστικά της γλώσσας

```
#base class definition
Class person
person instproc introduce {} {
        $self instvar _name
        puts “Hello, my name is $_name”
}

#employee inherits person
Class employee –superclass person
employee instproc introduce {} {
        $self instvar _name
        puts “Hello, my name is $_name and I am an employee”
```

```
}

#create objects of classes
set a [new person]
$a set _name “Nikos”
set b [new employee]
$b set _age “Petros”

#call object methods
$a introduce
$b introduce
```

Η λέξη κλειδί *class* ακολουθούμενη από το όνομα της κλάσης, ορίζει μία κλάση με το συγκεκριμένο όνομα. Η λέξη κλειδί *instproc* χρησιμοποιείται για να οριστεί μία μέθοδος μέλος για την κλάση. Η κληρονομικότητα επιτυγχάνεται με τη λέξη κλειδί –*superclass* και το όνομα της κλάσης την οποία θέλουμε να κληρονομήσει η υπο ορισμό κλάση. Η λέξη κλειδί *$self* παίζει τον ίδιο ρόλο που παίζει και η λέξη κλειδί *this* άλλων γνωστών αντικειμενοστραφών γλωσσών (π.χ. C, Java) και αναφέρεται στο τρέχον αντικείμενο. Η λέξη κλειδή *instvar* χρησιμοποιείται για να ελεχθεί αν η μεταβλητή έχει ήδη ορισθεί σε κάποια υπερκλάση. Εαν πράγματι χρησιμοποιείται, τότε γίνεται αναφορά στη μεταβλητή της υπερκλάσης, διαφορετικά δηλώνεται μία καινούργια. Η λέξη κλειδί *new* δημιουργεί ένα αντικείμενο της κλάσης.

Το πρόγραμμα του παραδείγματος βγάζει την παρακάτω έξοδο:

```
Hello, my name is Nikos
Hello, my name is Petros and I am an employee
```

3.4 Τρόπος χρήσης του ns2.

Όπως αναφέρθηκε και σε προηγούμενο κεφάλαιο, το ns δέχεται ως είσοδο αρχεία που περιγράφουν την τοπολογία ενός δικτύου σε κώδικα OTcl. Ο ίδιος ο εξομοιωτής τρέχει με τις παραμέτρους που ορίζονται από το χρήστη και εξομοιώνει την εν λόγω τοπολογία. Ας δούμε για παράδειγμα το αρχείο wireless1.tcl που βρίσκεται στον κατάλογο ns-tutorial/examples

```
# ======================================================================
# Define options
# ======================================================================
```

```
set val(chan)       Channel/WirelessChannel
set val(prop)       Propagation/TwoRayGround
set val(netif)      Phy/WirelessPhy
set val(mac)        Mac/802_11
set val(ifq)        Queue/DropTail/PriQueue
set val(ll)         LL
set val(ant)        Antenna/OmniAntenna
set val(x)            670   ;# X dimension of the topography
set val(y)            670   ;# Y dimension of the topography
set val(ifqlen)        50           ;# max packet in ifq
set val(seed)          0.0
set val(adhocRouting)   DSR
set val(nn)            3            ;# how many nodes are simulated
set val(cp)           "tcl/mobility/scene/cbr-3-test"
set val(sc)           "tcl/mobility/scene/scen-3-test"
set val(stop)          400.0          ;# simulation time

#
# ============================================================
# Main Program
#
# ============================================================

#
# Initialize Global Variables
#

# create simulator instance

set ns_                [new Simulator]

# setup topography object

set topo       [new Topography]

# create trace object for ns and nam

set tracefd    [open wireless1-out.tr w]
set namtrace    [open wireless1-out.nam w]
```

```
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)

# define topology
$topo load_flatgrid $val(x) $val(y)

#
# Create God
#
set god_ [create-god $val(nn)]

#
# define how node should be created
#

#global node setting

$ns_ node-config -adhocRouting $val(adhocRouting) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $val(ifq) \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                 -channelType $val(chan) \
                     -topoInstance $topo \
                     -agentTrace ON \
                 -routerTrace OFF \
                 -macTrace OFF

#
#  Create the specified number of nodes [$val(nn)] and "attach" them
#  to the channel.

for {set i 0} {$i < $val(nn) } {incr i} {
        set node_($i) [$ns_ node]
        $node_($i) random-motion 0              ;# disable random motion
}


#
# Define node movement model
```

```
#
puts "Loading connection pattern..."
source $val(cp)

#
# Define traffic model
#
puts "Loading scenario file..."
source $val(sc)

# Define node initial position in nam

for {set i 0} {$i < $val(nn)} {incr i} {

   # 20 defines the node size in nam, must adjust it according to your scenario
   # The function must be called after mobility model is defined

   $ns_ initial_node_pos $node_($i) 20
}


#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at $val(stop).0 "$node_($i) reset";
}

$ns_ at  $val(stop).0002 "puts \"NS EXITING...\" ; $ns_ halt"

puts $tracefd "M 0.0 nn $val(nn) x $val(x) y $val(y) rp $val(adhocRouting)"
puts $tracefd "M 0.0 sc $val(sc) cp $val(cp) seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) ant $val(ant)"

puts "Starting Simulation..."
$ns_ run
```

Το παραπάνω πρόγραμμα φτιάχνει έναν αριθμό κόμβων που καθορίζονται από την παράμετρο *nn* και που στο συγκεκριμένο παράδειγμα ισούται με 3. Αφού ορίσει τη θέση τους, τρέχει την εξομοίωση σύμφωνα με το πρωτόκολλο dsr που έχει οριστεί στην παράμετρο *adHocRouting*. Γενικώς με την εντολή *set val* μπορούν να οριστούν οι τιμές των καθολικών μεταβλητών. Η γενική μορφή της εντολής είναι:

set val(param) value

όπου *param* η μεταβλητή στην οποία θέτουμε την τιμή *value*.

Το πρόγραμμα δημιουργεί δύο αρχεία με τα αποτελέσματα της εξομοίωσης: το wireless1-out.tr και το wireless1-out.nam. Το πρώτο αρχείο περιέχει πληροφορίες για κάθε πακέτο ξεχωριστά. Μπορούμε να δούμε τα περιεχόμενά του με την εντολή *cat*. Οι εντολές με τις οποίες ορίζονται τα δύο παραπάνω αρχεία είναι οι εξής:

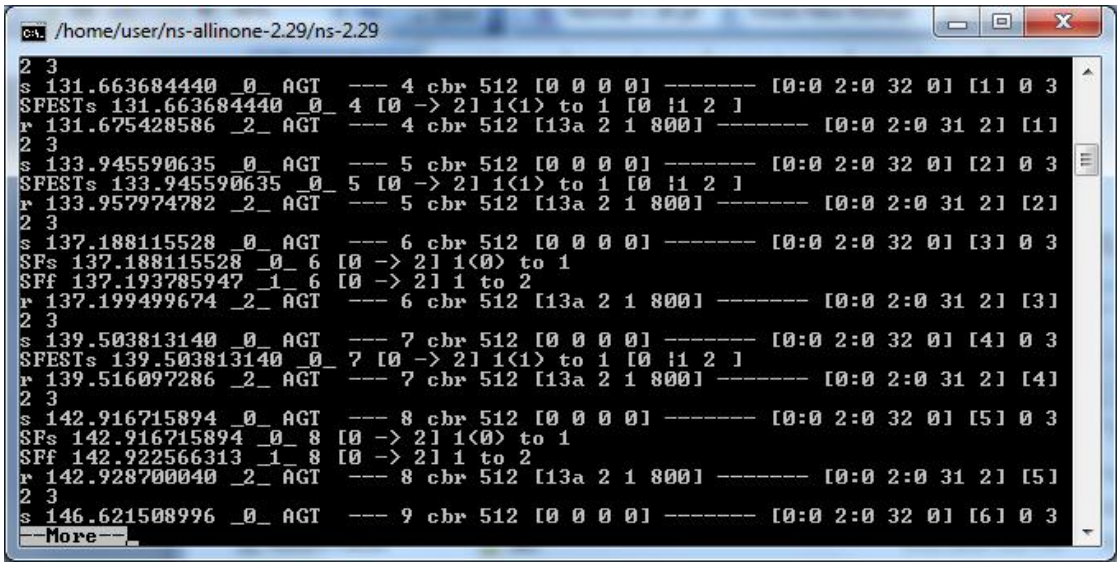
set tracefd [open wireless1-out.tr w]
set namtrace [open wireless1-out.nam w]

Οι παραπάνω εντολές σχετίζουν τα αρχεία *wireless1-out.tr* και *wireless1-out.nam* με τις μεταβλητές *tracefd* και *namtrace* αντίχτοιχα. Η παράμετρος *w* στην εντολή *open* λέει στο πρόγραμμα να δημιουργήσει τα αρχεία για γράψιμο. Οτιδήποτε γράφεται στις παραπάνω μεταβλητές, γράφεται και στα αρχεία με τα οποία αυτές σχετίζονται. Για παράδειγμα η εντολή:

puts $tracefd "M 0.0 sc $val(sc) cp $val(cp) seed $val(seed)"

λέει στο πρόγραμμα να δώσει ως έξοδο μία γραμματοσειρά (εντολή *puts*). Η γραμματοσειρά δίνεται από το κείμενο που βρίσκεται μέσα στα διπλά εισαγωγικά και θα γραφτεί στο αρχείο *wireless1-out.tr* το οποίο, όπως είδαμε αντιχτοιχεί στη μεταβλητή *tracefd*. Ομοίως και για το αρχείο *wireless1-out.nam*.

Ας δούμε τη μορφή και τη χρήση των δύο αυτών αρχείων πιο αναλυτικά. Συγκεκριμένα, η μορφή ενός τυπικού *.tr* αρχείου φαίνεται στην παρακάτω εικόνα.



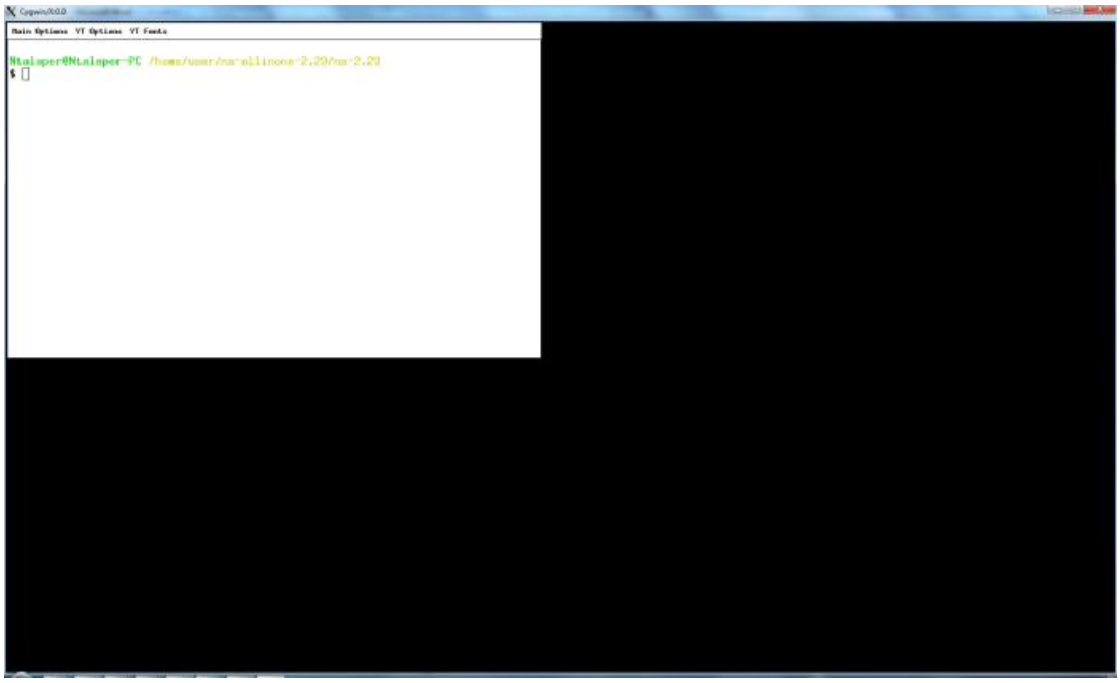
Εικόνα 3.6. Στιγμιότυπο ενός αρχείου ***trace****.*

Κάθε γραμμή περιγράφει κι από ένα γεγονός. Ο πρώτος χαρακτήρας κάθε γραμμής δείχνει τι είδους γεγονότος συνέβη και μπορεί να πάρει τις παρακάτω τιμές:

- “+”. Είσοδος στην ουρά αναμονής
- “-“, αποχώρηση από την ουρά αναμονής
- “h”, προώθηση πακέτου
- “r”, επιτυχημένη λήψη πακέτου
- “d”, απόρριψη πακέτου

Ακολουθεί η χρονική στιγμή κατά την οποία συνέβη το γεγονός. Οι επόμενες δύο λέξεις αναφέρουν τους κόμβους μεταξύ των οποίων βρίσκεται το πακέτο. Ακολουθεί το είδος του πακέτου και το μέγεθός του. Το αρχείο αυτό μπορεί να χρησιμοποιηθεί για εξαγωγή στατιστικών και συγκεντρωτικών στοιχείων. Αργότερα θα δούμε πώς μπορεί να γίνει αυτό μέσω του προγράμματος ***awk****.*

Το δεύτερο αρχείο, ***wireless1-out.nam*** μπορεί να διαβαστεί από το πρόγραμμα ***nam*** ώστε να δώσει μία γραφική αναπαράσταση σε πραγματικό χρόνο του αποτελέσματος της εξομοίωσης. Επειδή το ***nam*** απαιτεί γραφικά, είναι απαραίτητο να σηκωθεί πρώτα ένας x client. Αυτό μπορεί να γίνει με την εντολή ***xinit****.*



Εικόνα 3.7. Το περιβάλλον παραθύρων X.

Από το γραφικό περιβάλλον μπορεί να ξεκινήσει η εφαρμογή *nam*. Μέσω του μενού της *nam* μπορεί να επιλεγεί το αρχείο που θα διαβαστεί, στην περίπτωσή μας το wireless1-out.nam

Η τοπολογία του δικτύου ορίζεται μέσω των εντολών *set node* και η εξομοίωση τρέχει μέσω της εντολής *ns run*. Στα παρακάτω υποκεφάλαια δίνουμε μία πιο αναλυτική περιγραφή του *ns2*.

3.2.1 Βασική δομή ενός προγράμματος σε ns2

Όπως αναφέρθηκε, ένα πρόγραμμα σε ns2 είναι στην ουσία ένα πρόγραμμα (*script)* γραμμένο σε *Tcl*. Η βασική εντολή είναι η

Set ns [new Simulator]

η οποία δημιουργεί ένα αντικείμενο εξοιμοίωσης και πρέπει να υπάρχει σε κάθε *tcl script*. Επίσης κάθε *script* πρέπει να έχει και μία διαδικασία (*procedure*), η οποία θα καλείται όταν η εξομοίωση τελείωνει. Η διαδικασία αυτή ονομάζεται *finish*

```
proc finish {
        $ns flush-trace
        …
        exit 0
```

}

Η διαδικασία αυτή τυπικά γράφει στα αρχεία ό,τι δεδομένα δεν έχουν γραφεί ακόμα μέσω της εντολής ***flush-trace*** και επιστρέφει έναν κωδικό εξόδου στο λειτουργικό σύστημα μέσω της ***exit*** (0 για κανονική, χωρίς λάθη, έξοδο). Για να τελειώσει η εξομοίωση, πρέπει να δηλωθεί ρητά η χρονική στιγμή κατά την οποία γίνεται αυτό. Π.χ. μέσω της εντολής:

ns at 6.0 "finish"

δηλώνεται ότι η εξομοίωση σταματά στο $6^{ο}$ δευτερόλεπτο. Μόλις σταματάει η εξομοίωση, καλείται η διαδικασία ***finish*** που περιγράψαμε παραπάνω. Με την εντολή

ns run

τέλος, ξεκινάει η εξομοίωση αναφέρθηκε και στο προηγούμενο υποκεφάλαιο. Η ***ns run*** θα πρέπει να είναι η τελευταία εντολή της εξομοίωσης. Θα πρέπει να καλείται αφού οριστεί η τοπολογία, οι συνδέσεις μεταξύ κόμβων και γενικώς όλα τα στοιχεία της εξομοίωσης. Οι κόμβοι του δικτύου ορίζονται επίσης μέσω της εντολής ***set***.

set n0 [$ns node]
set n1 [$ns node]

Ο παραπάνω κώδικας δηλώνει δύο μεταβλητές (χειριστές ή ***handlers*** ακριβέστερα) οι οποίες αναφέρονται σε δύο κόμβους. Οι κόμβοι προσπελαύνονται στον κώδικα μέσω το αναγνωριστικό του ***handler*** τους (***n0*** και ***n1*** στο τρέχον παράδειγμα). Οι κόμβοι αυτοί δε συνδέονται μεταξύ τους. Για να υπάρξει οποιαδήποτε μεταφορά δεδομένων μεταξύ τους, θα πρέπει να δηλωθεί ρητά ότι υπάρχει ***link*** μεταξύ τους.

$ns duplex-link $n0 $n1 1Mb 10ms DropTail

Με την παραπάνω εντολή ορίζεται ότι οι δύο κόμβοι μπορούν να επικοινωνήσουν μέσω μίας γραμμής διπλής κατεύθυνσης, χωρητικότητας ***1Mb*** και καθυστέρηση ***10ms***. Το ***DropTail*** στο τέλος, ορίζει ότι η ουρά αναμονής θα απορρίπτει πακέτα από την ουρά (δηλαδή από το τέλος), όταν υπάρξει συμφόρηση.

Μέχρι τώρα δεν έχει οριστεί τι είδους κίνηση θα υπάρχει μεταξύ των κόμβων. Για να οριστεί η κίνηση, θα πρέπει πρώτα να οριστούν ***agents***. Στο ***ns*** πάντα η κίνηση ξεκινάει από έναν ***agent*** και καταλήγει σε άλλον. Με άλλα λόγια η κίνηση γίνεται μόνο μεταξύ ***agents*** οι οποίοι με τη σειρά τους

αντιστοιχούν σε κόμβους. Έστω ότι θέλουμε να στέλνονται δεδομένα του πρωτοκόλλου *udp* από τον κόμβο *n0*.

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

Η πρώτη γραμμή δημιουργεί τον *agent* και του δίνει το όνομα *udp0*. Στη δεύτερη γραμμή ο *agent udp0* αντιστοιχίζεται στον κόμβο *n0*. Μένει να δοθούν τα ακριβή χαρακτηριστικά της κίνησης (ρυθμός αποστολή πακέτων, μέγεθος πακέτων κοκ).

set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

Η πρώτη γραμμή δημιουργεί μία γεννήτρια πακέτων τύπου *CBR* (*Constant Bit Rate*, δηλαδή σταθερού ρυθμού), με το όνομα *cbr0*. Οι επόμενες δύο γραμμές θέτουν τις παραμέτρους *packetSize_* και *interval_* του αντικειμένου *cbr0*, οι οποίες αντιστοιχούν στο μέγεθος πακέτου και στο διάστημα αναμετάδοσης αντίστοιχα. Η τελευταία γραμμή συνδέει τη γεννήτρια με τον *agent udp0* που έχει ορισθεί παραπάνω.

Μένει να δηλωθεί *agent* για τον κόμβο *n1*. Στο συγκεκριμένο παράδειγμα ο *n1* απλά θα λαμβάνει δεδομένα, άρα ο τύπος του θα είναι *sink*.

Set null0 [new Agent/Null]
$ns attach-agent $n1 $null0

Κατά αντιστοιχία με τον προηγούμενο *agent*, ο καινούργιος *agent* δηλώνεται με το όνομα *null0* και είναι τύπου *Null*, δηλαδή δεν παράγει πακέτα. Τα χαρακτηριστικά της κίνησης για αυτόν τον *agent* επομένως, δε χρειάζεται να ορισθούν. Μένει μόνο να συνδεθούν οι δύο *agents* ώστε να ο πρώτος να στέλνει δεδομένα στον δεύτερο μόλις ξεκινήσει η εξομοίωση.

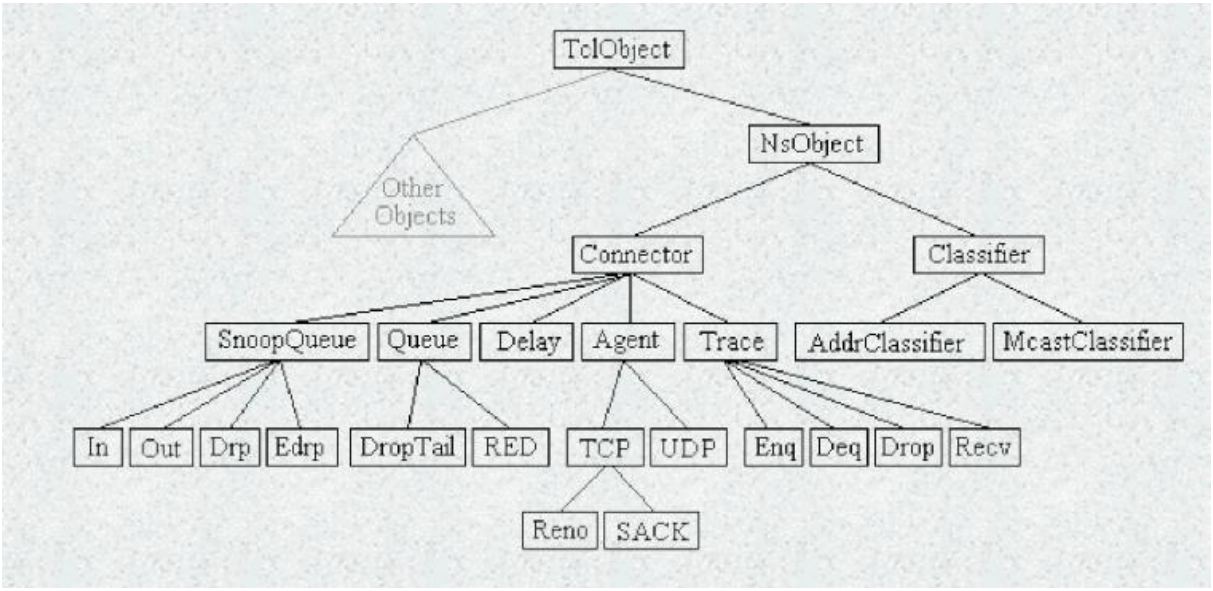
$ns connect $udp0 $null0

Για να γίνει η εξομοίωση θα πρέπει να δοθεί εντολή εκκίνησης, καθώς και τερματισμού:

$ns at 1.0 “cbr0 start”
$ns at 5.0 “cbr0 stop”

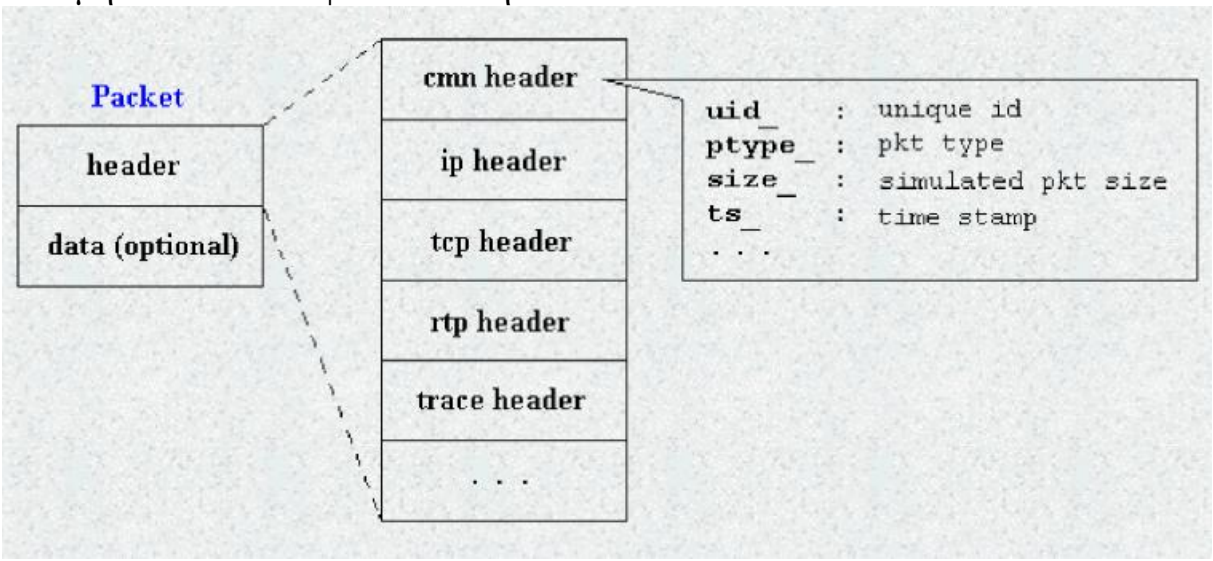
Με τις παραπάνω δύο γραμμές ξεκινάει η γεννήτρια κίνησης *cbr0* να παράγει δεδομένα τη χρονική στιγμή *1.0s*, και τερματίζει τη χρονική στιγμή *5.0s*.

3.2.2 Συστατικά Δικτύου (network components).



3.2.2 Πακέτα (packets).

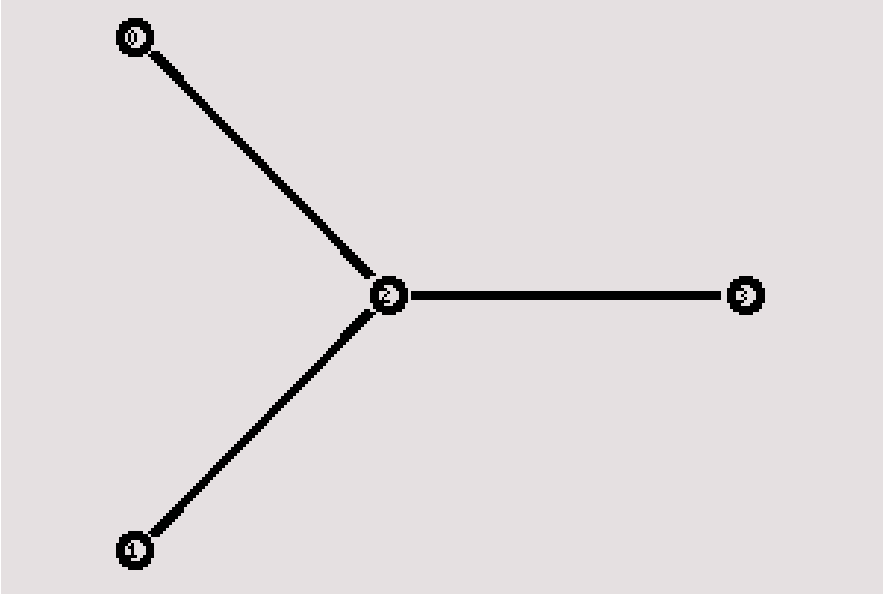
Ένα πακέτο στο ns αποτελείται από τις επικεφαλίδες κι από το χώρο δεδομένων. Η δομή του πακέτου φαίνεται στη εικόνα 3.6



Εικόνα 3.6. Δομή πακέτου

3.2.2 Τοπολογία Δικτύου και συμβάντα.

Ένα τυπικό σενάριο εξομοίωσης περιλαμβάνει πολλούς κόμβους και πιθανόν πολύπλοκη τοπολογία. Στο παρόν υποκεφάλαιο θα δούμε πώς γίνεται, τροποποιώντας τη βασική δομή του κώδικα που παρουσιάστηκε στο προηγούμενο υποκεφάλαιο, να οριστούν πιο πολύπλοκες τοπολογίες. Ο βασικός κώδικας παραμένει ο ίδιος. Θα πρέπει να οριστεί ένα αντικείμενο τύπου ***Simulator*** όπως και πριν. Η διαφορά είναι στην τοπολογία των κόμβων και στη μεταξύ τους συνδεσιμότητα. Έστω για παράδειγμα ότι έχουμε την τοπολογία που φαίνεται στο παρακάτω σχήμα:



Εικόνα 3.8. Απλή τοπολογία τεσσάρων κόμβων. Ένας μοναδικός κόμβος, ο κόμβος «2», δρα ως γέφυρα που συνδέει όλους τους άλλους κόμβους μεταξύ τους.

Το πρώτο βήμα, όπως και πριν, είναι να ορίσουμε τους κόμβους του δικτύου:

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

Κατόπιν, θα πρέπει να ορίσουμε τις συνδέσεις μεταξύ των κόμβων. Όπως φαίνεται και από την εικόνα, υπάρχει τρεις συνδέσεις μεταξύ των κόμβων 2 και 1, 2 και 0, και 2 και 3. Υποθέτοντας ότι είναι διπλής κατεύθυνσης, ο κώδικας που δημιουργεί τις παρακάτω συνδέσεις, είναι ο παρακάτω:

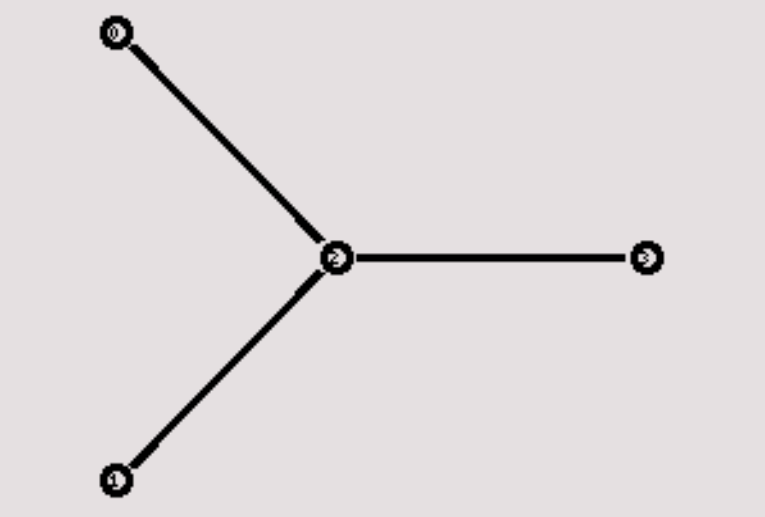
```
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms DropTail
```

Σ’αυτό το σημείο αξίζει να αναφερθεί, ότι ενώ ορίζεται η τοπολογία του δικτύου, εντούτοις δεν έχει ορισθεί η θέση των κόμβων μέσα στο πλέγμα (grid). Αυτό, όσον αφορά τα αποτελέσματα της εξομοίωσης είναι αδιάφορο, θα ήταν επιθυμητό παρ’όλ’αυτά να μπορεί να οριστεί η θέση των κόμβων ώστε το αποτέλεσμα να είναι οπτικά αρεστό σε περίπτωση που η εξομοίωση τρέξει σε γραφικό περιβάλλον (π.χ nam).

Αυτό μπορεί να γίνει εύκολα:

$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

Με τις παραπάνω εντολές δίνεται η θέση των συνδέσεων, όπως αυτή θα εμφανιστεί από το nam.



Εικόνα 3.9. Απλή τοπολογία τεσσάρων κόμβων όπως αυτή εμφανίζεται από το «nam». Οι σχετικές θέσεις μεταξύ των κόμβων έχουν δωθεί ρητά με τη δήλωση «orient», η οποία ακολουθεί κάθε αρχικοποίηση κόμβου.

Προχορώντας με παρόμοιο τρόπο με το προηγούμενο υποκεφάλαιο, μπορούν να ορισθούν οι agents μεταξύ των κόμβων. Στο συγκεκριμένο παράδειγμα οι κόμβοι *0* και *1* θα παράγουν κίνηση δεδομένων, ενώ ο κόμβος *3* θα δέχεται δεδομένα. Προφανώς, η κίνηση θα δρομολογηθεί μέσω του κόμβου *2*, αφού μέσω αυτού μόνο υπάρχει μονοπάτι μεταξύ των κόμβων *1* και *2* και του κόμβου *3*.

#Δημιουργία UPD agent στον κόμβο n0
set udp0 [new Agent/UDP]

```
$ns attach-agent $n0 $udp0

#Δημιουργία κίνηση σταθερού ρυθμού αποστολής στον agent #udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

# Δημιουργία UPD agent στον κόμβο 1
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1

#Δημιουργία κίνηση σταθερού ρυθμού αποστολής στον agent #udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

#Ο κόμβος n3 δέχεται δεδομένα και δεν παράγει κίνηση
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
```

Μένει να ξεκινήσει η διαδικασία εξομοίωσης:

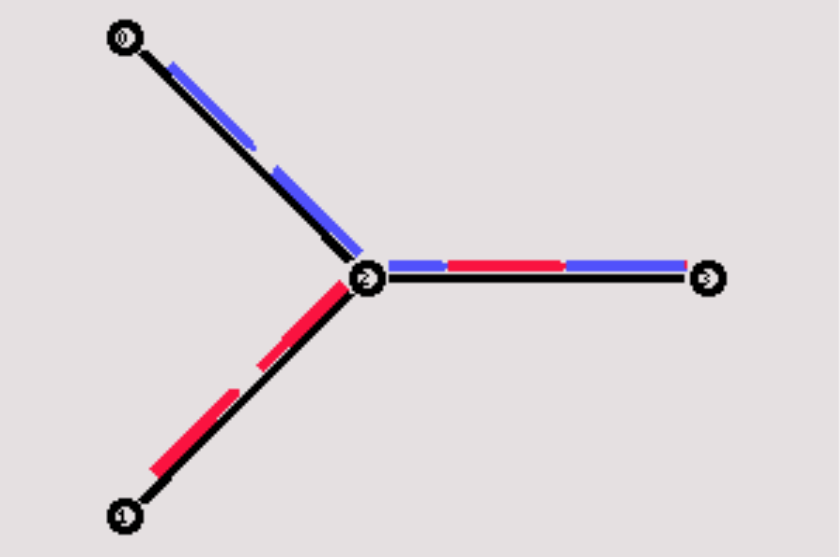
```
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
```

Τρέχοντας την παραπάνω εξομοίωση και παρατηρώντας το αρχείο εξόδου μέσω της *awk* ή άλλου εργαλείου, παρατηρείται ότι υπάρχει απώλεια δεδομένων. Αυτό μπορεί να εξηγηθεί εύκολα: Τη χρονική στιγμή *1.0* που ξεκινάει η κίνηση μεταξύ των κόμβων *1* και *3* ο ρυθμός που δέχεται δεδομένα ο κόμβος *2* που δρα ως δρομολογητής, ξεπερνά τη χωρητικότητα της διαδρομής μεταξύ του *2* και *3*. Πράγματι, ο αριθμός των πακέτων που στέλνονται σε ένα δευτερόλεπτο ισούται με *1/interval*. Στην περίπτωση του παραδείγματος, *interval=0.005*, άρα σε κάθε σύνδεση μεταξύ των κόμβων *0,1* και 2, στέλνονται 200 πακέτα. Το κάθε πακέτο έχει μέγεθος *500*, άρα σε κάθε δευτερόλεπτο, στέλνεται 1Mb δεδομένων. Συνολικά δηλαδή στον κόμβο-δρομολογητή *2* φτάνουν *2Mb* το δευτερόλεπτο. Αναγκαστικά, αφού η χωρητικότητα της σύνδεσης μεταξύ *2* και *3* είναι 1Mb/sec, κάποια πακέτα θα απορριφθούν.

Σε τέτοιες περιπτώσεις θα θέλαμε να μπορούμε να δούμε ποια σύνδεση έχει το μεγαλύτερο ποσοστό απόρριψης, ώστε να φανεί κατά πόσο η δρομολόγηση είναι «δίκαια» ή όχι. Σε περίπτωση που θέλουμε να το δούμε αυτό «οπτικά», μέσω του *nam*, υπάρχει ένα πρόβλημα. Και οι δύο ροές έχουν το ίδιο χρώμα (μαύρο είναι το προκαθορισμένο), άρα δε θα φαίνεται από ποια ροή χάνονται τα περισσότερα πακέτα. Αυτό μπορεί να επιλυθεί δηλώνοντας διαφορετικά χρώματα του προκαθορισμένου, σε κάθε ροή.

```
$udp0 set class_ 1
$udp1 set class_ 2

$ns color 1 Blue
$ns color 2 Red
```

Με τις 2 πρώτες εντολές δίνεται ένα *id* στις ροές. Με τις δύο τελευταίες αντιστοιχίζεται ένα χρώμα σε κάθε ροή. Ένα στιγμιότυπο της εξομοίωσης στο *nam* φαίνεται στην παρακάτω εικόνα:

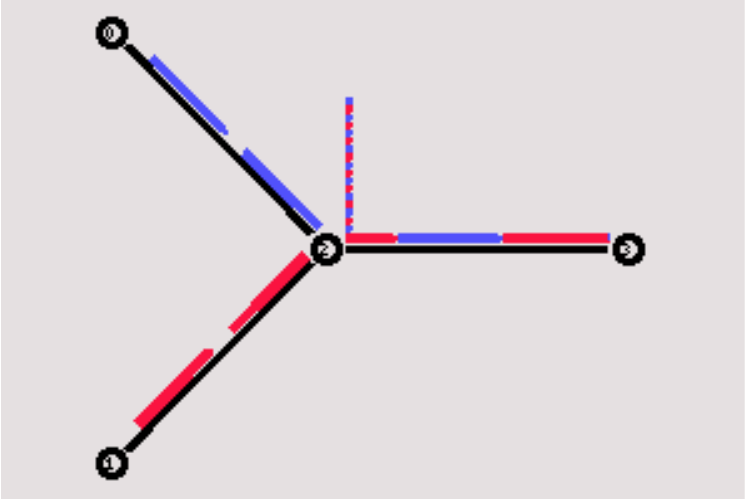


Εικόνα 3.10. Στιγμιότυπο μίας απλής ροής, όπως αυτή φαίνεται από το «nam».

Απ'ότι φαίνεται η ροή μεταξύ *2* και *3* έχει περισσότερα κόκκινα πακέτα. Η δρομολόγηση δεν είναι «δίκαια» στο συγκεκριμένο παράδειγμα. Αυτό μπορεί να φανεί και καλύτερα αν δοθεί στο *ns* η οδηγία να παρακολουθείται και να καταγράφεται και η ουρά δρομολόγησης:

```
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

Τρέχοντας πάλι την εξομοίωση και προβάλλοντας τα αποτελέσματα στο *nam*, φαίνεται η ουρά δρομολόγησης καθώς και ποια πακέτα απορρίπτονται από τον κόμβο-δρομολογητή *2*.



Εικόνα 3.11. Στιγμιότυπο μίας απλής ροής, όπως αυτή φαίνεται από το «nam». Εκτός από τις ροές μεταξύ των κόμβων φαίνεται και η ουρά δρομολόγησης στον κόμβο 2.

Τα αποτελέσματα της συγκεκριμένης εξομοίωσης, δείχνουν ότι ο δρομολογητής απορρίπτει μόνο μπλε πακέτα. Αυτό συμβαίνει γιατί ο μηχανισμός απόρριψης πακέτων είναι η απόρριψη του τελευταίου πακέτου που βρίσκεται στην ουρά (*Droptail)*. Καθώς ο δρομολογητής απορρίπτει ένα πακέτο για κάθε πακέτο που προωθεί (η χωρητικότητα της ροής εξόδου είναι μισή της χωρητικότητας ροής εισόδου), το πακέτο που βρίσκεται στο τέλος (*ουρά)* θα έρχεται συνέχεια από την ίδια ροή.

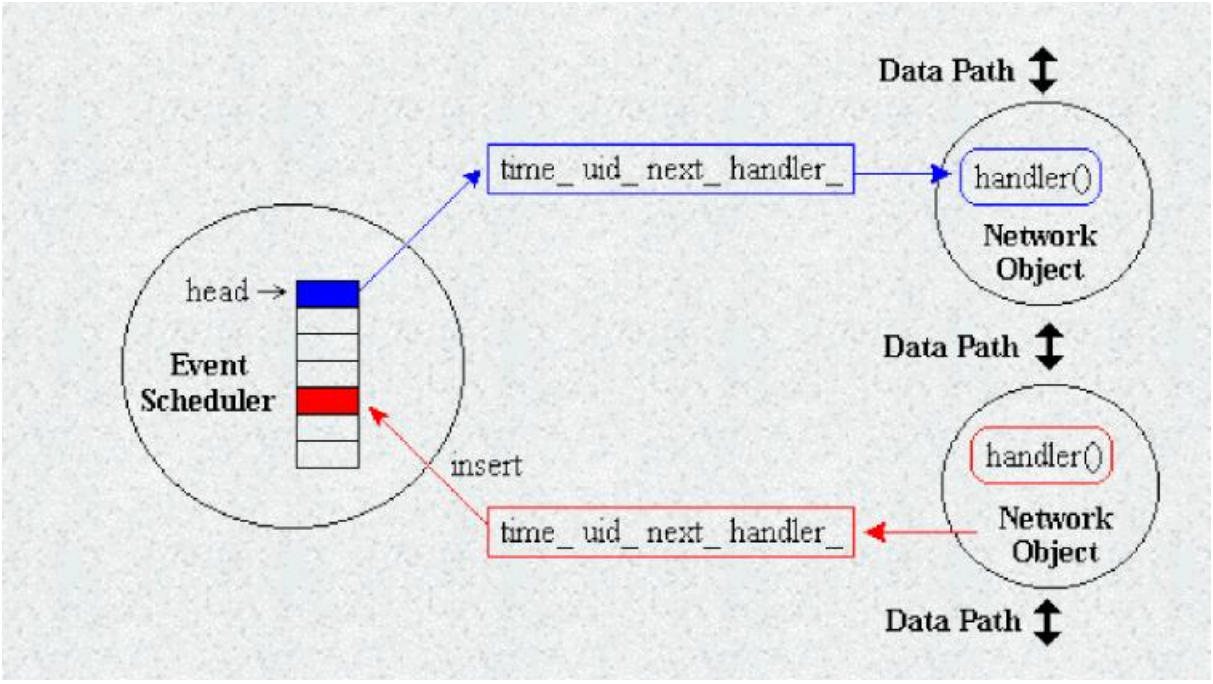
Ένας τρόπος για μία ποιο δίκαια κατανομή, θα ήταν να απορρίπτει ο δρομολογητής πακέτα τυχαία, μέσω κάποιας στοχαστικής διεργασίας. Αυτό μπορεί να γίνει χρησιμοποιώντας μία *στοχαστικά δίκαια ροή* (*Stochastic fairness queuing* ή *SFQ)*.

$ns duplex-link $n3 $n2 1Mb 10ms SFQ

3.2.5 Δρομολογητής Συμβάντων (Event Scheduler)

Ο δρομολογητής συμβάντων χρησιμοποιείται από τα στοιχεία δικτύου (network components) τα οποία τον “πυροδοτούν” όταν χρειαστεί κάποιος χειρισμός πακέτου η σε συγκεκριμένες χρονικές στιγμές. Στην εικόνα 3.12

φαίνεται η αλληλεπίδραση μεταξύ των στοιχείων δικτύου κι ενός δρομολογητή συμβάντων.



3.12 Τα network components εισάγουν δεδομένα (δείκτες σε συναρτήσεις χειρισμού ή αλλιώς handlers) στη στοίβα του δρομολογητή συμβάντων. Στην περίπτωση "πυροδότησης" του συμβάντος, ο δείκτης αυτός περνάει πίσω στο component το οποίο ειδοποείται έτσι για το συγκεκριμένο συμβάν και το χειρίζεται ανάλογα.

3.2.4 Δυναμικά Δίκτυα

Το παρόν υποκεφάλαιο εξετάζει μία ενδιάμεση κατάσταση μεταξύ της στατικής και της *ad hoc* δρομολόγησης. Εξετάζονται τα δυναμικά δίκτυα των οποίων η τοπολογία μπορεί να αλλάξει, λόγω μίας βλάβης μεταξύ της ροής δύο κόμβων και όχι λόγω κίνησης των κόμβων.

Για το παράδειγμα, θα φτιαχτεί μία τοπολογία 7 κόμβων σε σχηματισμό δαχτυλιδιού (*ring)*. Κάθε κόμβος δηλαδή θα γειτνιάζει με δύο ακριβούς κόμβους, με κάθε κόμβο να έχει το πολύ έναν κοινό γείτονα με κάθε άλλο κόμβο. Οι κόμβοι μπορούν να δηλωθούν ένας ένας, όπως και στα προηγούμενα παραδείγματα. Παρ'ολ'αυτά, θα εξεταστεί ένας πιο εύκολος τρόπος, που θα δείξει κιόλας πόσο δυνατή είναι η χρήση της *tcl*, για τη δημιουργία σεναρίων εξομοίωσης στο *ns*. Ο κώδικας για τη δημιουργία 7 κόμβων είναι ο παρακάτω:

```
for {set i 0} {$i < 7} {incr i} {
  set n($i) [$ns node]
}
```

Αποτελεί μία κλασσική περίπτωση βρόχου *for* που συναντάται και σε άλλες γλώσσες προγραμματισμού. Ο βρόχος *for* είναι μία εντολή επανάληψης,

δηλώνει δηλαδή ότι το σώμα τους θα εκτελεστεί παραπάνω από μία φορά, ανάλογα με τις συνθήκη τερματισμού. Ας δούμε πιο αναλυτικά τα μέρη της *for*:

- *{set i 0}* Εδώ δίνεται η αρχικοποίηση. Δηλώνεται μία μεταβλητή που θα χρησιμεύσει σαν μετρητής και αρχικοποιείται με την τιμή 0.
- *{$i < 7}* Η συνθήκη τερματισμού. Ο βρόχος θα τρέχει μέχρι η τιμή της μεταβλητής *i* γίνει 7.
- *{incr i}* Το βήμα του βρόχου. Σε κάθε επανάληψη η μεταβλητή *i* θα αυξάνεται κατά ένα.

Είναι φανερό ότι ο συγκεκριμένος βρόχος θα τρέξει 7 φορές. Το τι ενέργεια θα εκτελεστεί σε κάθε επανάληψη καθορίζεται από το σώμα του βρόχου. Στο παράδειγμα, το σώμα του βρόχου είναι ο εξής κώδικας:

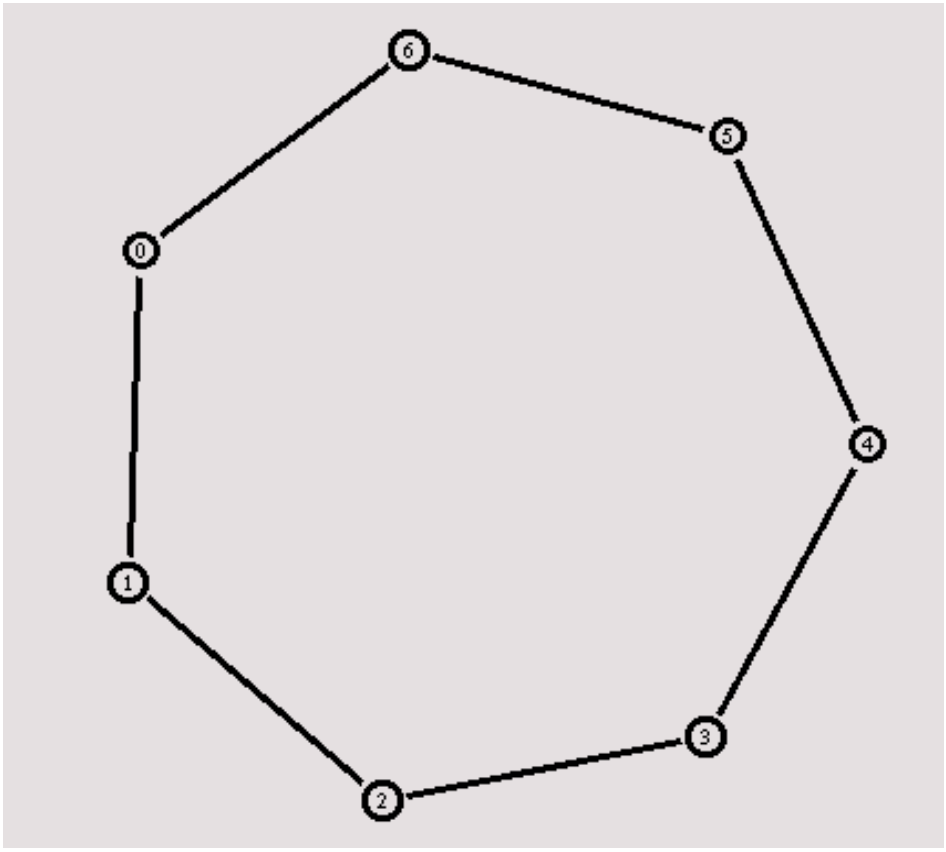
```
set n($i) [$ns node]
```

Εδώ φαίνεται ένα ακόμα από τα χαρακτηριστικά της *tcl*, η δυνατότητα δημιουργίας πινάκων. Αντί να φτιαχτούν 7 μεταβλητές, δημιουργείται ένας πίνακας από μεταβλητές που αντιστοιχούν σε κόμβους. Το νούμερο που βρίσκεται μέσα σε παρενθέσεις υποδεικνύει τη θέση του στοιχείου μέσα στον πίνακα. Έτσι, η παραπάνω εντολή, δηλώνει τη δημιουργία ενός κόμβου στη θέση του πίνακα που δηλώνει η μεταβλητή *i*. Καθώς ο βρόχος τρέχει 7 φορές (από 0 μέχρι 6) με το *i* να διατρέχει τις τιμές 0,1...6, θα δημιουργηθούν 7 κόμβοι, οι κόμβοι *n(0),n(1)…n(6)*.

Για τη δημιουργία συνδέσεων μεταξύ των κόμβων, θα χρησιμοποιηθεί πάλι ένας βρόχος *for*.

```
for {set i 0} {$i < 7} {incr i} {
  $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
}
```

Ο παραπάνω βρόχος συνδέει κάθε κόμβο με τον επόμενό του. Καθώς ο τελευταίος κόμβος πρέπει να συνδεθεί με τον πρώτο, χρησιμοποιείται *modulo* αριθμητική για τον δείκτη θέσης του δεύτερου κόμβου. Συγκεκριμένα, η έκφραση: *expr ($i+1)%7* θα δώσει τις τιμές 1,2,...6 για τις τιμές του *i=0,1,…5*, ενώ μόλις το *i* γίνει 6, θα δώσει την τιμή 0, συνδέοντας έτσι τον κόμβο *n(6)* με τον κόμβο *n(0)*. Η τοπολογία που δημιουργείται με τον παραπάνω κώδικα φαίνεται στην παρακάτω εικόνα του *nam*.



Εικόνα 3.12. Απλή τοπολογία «δακτυλίου», όπως αυτή φαίνεται στο «nam».

Στο επόμενο βήμα, δημιουργείται η κίνηση του σεναρίου. Στο παράδειγμα που μελετάται, η κίνηση θα είναι μεταξύ του κόμβου *0* και *3*.

```
#Create a UDP agent and attach it to node n(0)
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

$ns connect $udp0 $null0
```

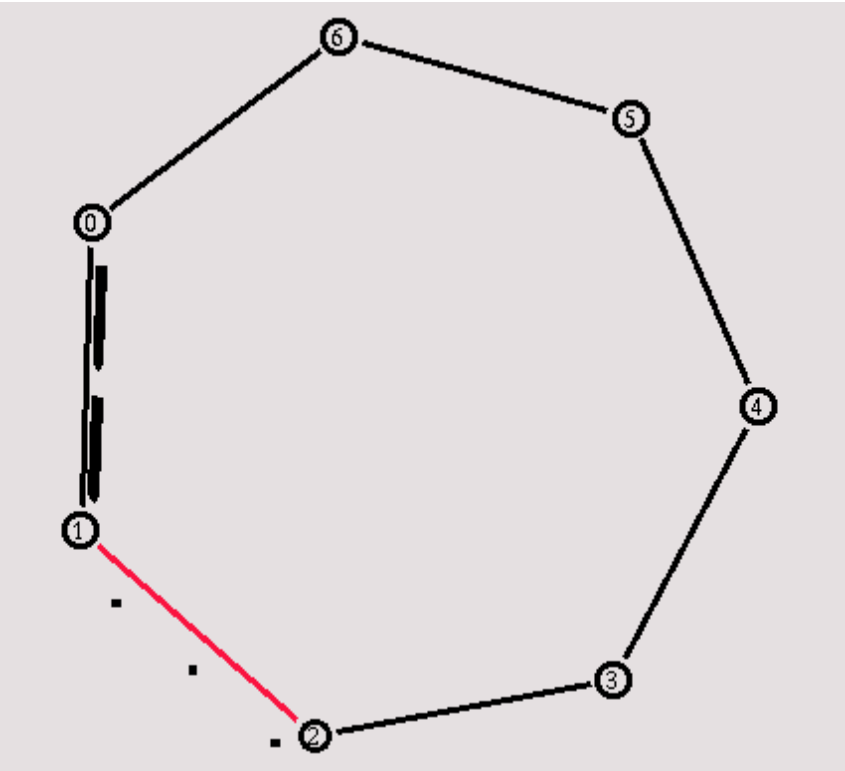
```
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
```

Είναι φανερό ότι η δρομολόγηση θα γίνει από τον κόμβο *0* στον κόμβο *1*, από εκεί στον κόμβο *2* και τέλος στον κόμβο-προορισμό *3*.

Έστω τώρα ότι προστίθενται οι παρακάτω 2 γραμμές κώδικα στο σενάριο:

```
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
```

Η πρώτη γραμμή λέει, ότι η σύνδεση μεταξύ του *$n(1)* και *$n(2)* θα «πέσει» τη χρονική στιγμή 1.0. Η δεύτερη γραμμή αποκαθιστά τη γραμμή τη χρονική στιγμή 2.0. Μεταξύ των δύο αυτών χρονικών στιγμών η διαδρομή που περιγράφτηκε μεταξύ των κόμβων *0* και *3* δε θα είναι έγκυρη συνεπώς. Τρέχοντας το παραπάνω σενάριο και προβάλλοντας τα αποτελέσματα στο *nam* μπορούμε να παρατηρήσουμε τι συμβαίνει μεταξύ των δύο αυτών χρονικών στιγμών.

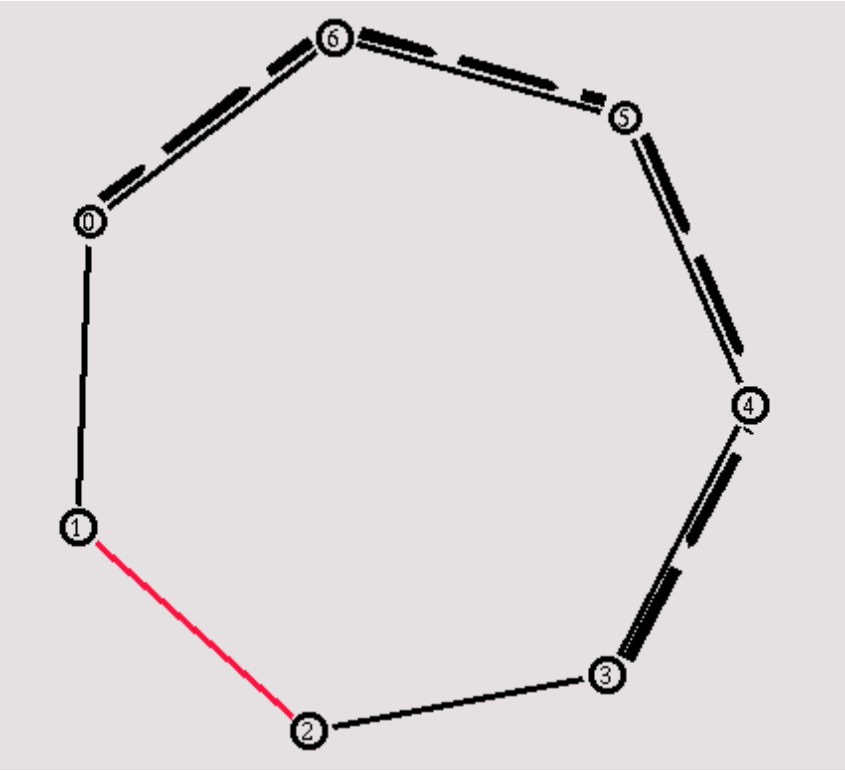


Εικόνα 3.13. Απλή τοπολογιά «δακτυλίου», με το σύνδεσμο μεταξύ των κόμβων 1 και 2 να βρίσκεται εκτός λειτουργίας. Απεικονίζεται η ροή δεδομένων από τον κόμβο 0 στον κόμβο 1. Τα δεδομένα χάνονται καθώς γίνεται η δρομολόγησή τους μέσω του «νεκρού» συνδέσμου.

Είναι φανερό ότι τα πακέτα χάνονται μιας και πάνε να δρομολογηθούν μέσω διαδρομής που δεν υπάρχει πλέον. Παρ'όλ'αυτά, υπάρχει μία εναλλακτική διαδρομή μέσω των κόμβων *0,6,5,...,2*. Στη δυναμική δρομολόγηση, όταν μία διαδρομή δεν είναι πλέον έγκυρη, ανακαλύπτεται ένα νέο μονοπάτι, όχι κατ'ανάγκη βέλτιστο, ώστε τα πακέτα να δρομολογηθούν μέσω αυτού. Η παρακάτω εντολή λέει στο *ns* να χρησιμοποιήσει δυναμική δρομολόγηση σε περίπτωση που κάποια διαδρομή παύει να ισχύει, και συγκεκριμένα να χρησιμοποιήσει τον αλγόριθμο του *Distance Vector (DV)* για την εύρεση εναλλακτικών μονοπατιών.

$ns rtproto DV

Τρέχοντας πάλι την εξομοίωση, φαίνεται ότι η κίνηση ανακατευθύνεται μέσω της εναλλακτικής διαδρομής που βρήκε ο *DV* μεταξύ των χρονικών στιγμών που η ροή *1->2* είναι εκτός λειτουργίας



Εικόνα 3.14. Απλή τοπολογιά «δακτυλίου», με το σύνδεσμο μεταξύ των κόμβων 1 και 2 να βρίσκεται εκτός λειτουργίας. Απεικονίζεται η ροή δεδομένων από τον κόμβο 0 στον κόμβο 1, όπως αυτή έχει επαναδρομολογηθεί προκειμένου να παρακαμφθεί ο «νεκρός» σύνδεσμος 1->2.

3.2.4 Σύνδεση με OTcl

Σ'αυτήν την παράγραφο θα εξεταστεί η βασική διαδικασία με την οποία μπορεί να επεκτείνει ο προγραμματιστής το ns, ώστε να δημιουργεί δικές του

κλάσεις. Ο σκελετός αυτού του υποκεφαλαίου θα χρησιμοποιηθεί στο επόμενο, ώστε να παρουσιαστεί ο τρόπος με τον οποίο μπορεί να υλοποιηθούν καινούργια πρωτόκολλα. Προς το παρόν, θα αναλύσουμε τα βασικά βήματα δημιουργίας νέας κλάσης, υλοποιώντας ένα dummy agent (έναν agent δηλαδή που δε δέχεται ούτε στέλνει πακέτα).

Ο τρόπος με τον οποίο γίνεται η παραπάνω διαδικασία είναι μέσω της διασύνδεσης C++ με OTcl. Συγκεκριμένα, υλοποιούμε τις κλάσεις C++ που υλοποιούν τη λειτουργικότητα του agent. Στη συνέχεια «εξάγουμε» (export) των κώδικα σε OTcl. Παρουσιάζονται τα βήματα αναλυτικά:

- Εξαγωγή κλάσης C++ σε OTcl
  Έστω ότι θέλουμε να υλοποιήσουμε μία κλάση τύπου agent με την ονομασία DummyAgent ώστε να μπορούμε να δημιουργούμε αντικείμενα τύπου DummyAgent σε κώδικα OTcl. Μπορούμε να κληρονομήσουμε τη βασική κλάση agent που υπάρχει ήδη στη βιβλιοθήκη. Ο κώδικας δίνεται παρακάτω:

```
class DummyAgent: public Agent {
public:
        DummyAgent();
protected:
        int command(int argc, char **argv);
private:
        int dummy_var_1;
        int dummy_var_2;
        void somefunction();
}
```

  Ως εδώ έχει δημιουργηθεί η κλάση για τον agent σε C++. Για να γίνει η διασύνδεση όμως με OTcl (ώστε να μπορούν να δημιουργηθούν αντικείμενα σε OTcl scripts) θα πρέπει να δημιουργηθεί κι ένα αντικείμενο διασύνδεσης. Το αντικείμενο αυτό, έστω με το όνομα *DummyAgentClass* πρέπει να κληρονομεί την κλάση *TclClass* (σημ. το αντικείμενο αυτό είναι στην ουσία μία στατική κλάση, γι'αυτό ακριβώς το λόγο χρησιμοποιούνται όροι όπως «κληρονομεί» που έχουν νόημα για κλάσεις κι όχι για αντικείμενα). Ο *constructor* της κλάσεις *TclClass* δέχεται ως όρισμα το όνομα του αντικειμένου, όπως αυτό θα χρησιμοποιηθεί στα OTcl scripts. Τα δύο αυτά αντικείμενα συνδέονται μέσω της συνάρτησης *create* η οποία στην ουσία επιστρέφει έναν δείκτη σε αντικείμενο της C++ κλάσης (*DummyAgent* στο παράδειγμά μας).. Ακολουθεί ο κώδικας:

```
static class DummyAgentClass : public TclClass {
    public:
```

```
        DummyAgentClass : TclClass("Agent/DummyAgent") {}
        TclObject* create(int, const char* const*) {
        return (new DummyAgent());
  }
} class_dummy_agent;
```

Όταν ξεκινάει το *ns* εκτελείται ο *constructor* για τη στατική μεταβλητή *class_dummy_agent* κι αυτό προκαλεί τη δημιουργία ενός αντικειμένου τύπου *DummyAgentClass*. Στο χώρο δεδομένων του OTcl script η δημιουργία του αντικειμένου τύπου *DummyAgentClass* δημιουργεί με τη σειρά της την κλάση *Agent/DummyAgent*, όπως φαίνεται και στον κώδικα. Από δω και στο εξής, κάθε φορά που ο χρήστης θα ορίζει μία μεταβλητή τύπου *Agent/DummyAgent* (με την εντολή new Agent/DummyAgent), καλείται η συνάρτηση *create* η οποία δημιουργεί ένα αντικείμενο τύπου *DummyAgent* και επιστρέφει τη διεύθυνση .

- Εξαγωγή μεταβλητών κλάσεων C++ σε OTcl.
  Το αντικείμενο *DummyAgent* περιέχει δύο μεταβλητές τις *dummy_var_1* και *dummy_var_2* τύπου ακεραίου. Προκειμένου να προσπελαστούν αυτές οι μεταβλητές για αλλαγή ή ανάγνωση από την OTcl θα πρέπει να συνδεθούν (bind) μέσω μία συνάρτησης σύνδεσης (binding function). Η συνάρτηση αυτή είναι η *bind* και πέρνει δύο ορίσματα. Το πρώτο όρισμα είναι το όνομα της μεταβλητής που θα δημιουργηθεί στο χώρο μεταβλητών της OTcl και το δεύτερο είναι η διεύθυνση της μεταβλητής με την οποία θα συνδεθεί. Ο κώδικας είναι ο παρακάτω:

```
DummyAgent::DummyAgent() : Agent(PT_UDP)  {
    bind("dummy_var_1_otcl", &dummy_var_1);
    bind("dummy_var_2_otcl", &dummy_var_2);
}
```

  Οι κλήσεις των συναρτήσεων διασύνδεσης τοποθετούνται στον *constructor* του *DummyAgent* ώστε να γίνουν οι διασυνδέσεις με το που δημιουργείται το αντικείμενο.

3.2.5 Δημιουργία Πρωτοκόλλων

Σ'αυτό το υποκεφάλαιο θα εξεταστεί το κύριο ίσως χαρακτηριστικό του *ns*, που το έχει κάνει τόσο δημοφιλές στην έρευνα πρωτοκόλλων δρομολόγησης. Το χαρακτηριστικό αυτό είναι η δυνατότητα δημιουργίας καινούργιων πρωτοκόλλων. Στο *ns* ο ερευνητής/προγραμματιστής μπορεί να

αλλάξει ή να προσθέσει τμήματα κώδικα που υλοποιούν καινούργιες λειτουργίες ή τροποποιούν ήδη υπάρχουσες. Για παράδειγμα, να κάποιος ερευνητής μελετά έναν νέο μηχανισμό απόρριψης πακέτων και θέλει να τον συγκρίνει με τον *SFQ* που είδαμε σε προηγούμενο υποκεφάλαιο, μπορεί να προσθέσει το νέο μηχανισμό γράφοντας κώδικα σε tcl ή/και C++. Σε ποιο επίπεδο θα προγραμματίσει (Tcl ή C++), εξαρτάται από το πεδίο εφαρμογής. Όπως αναφέρθηκε, η Tcl χρησιμοποιείται για τη συγγραφή προγραμμάτων που βασίζονται σε ήδη υπάρχουσες δομές, οι οποίες συνήθως έχουν υλοποιηθεί ως κλάσεις σε C++. Αν λοιπόν ο προγραμματιστής δεν επιθυμεί να αλλάξει αυτές τις δομές, τότε η Tcl αρκεί. Στην πιο συνηθισμένη όμως περίπτωση οι λειτουργίες που πρέπει να υλοποιηθούν σε επίπεδο έρευνας, απαιτούν την τροποποίηση των υπαρχουσών δομών ή την δημιουργία καινούργιων. Σ'αυτή την περίπτωση η C++ είναι μονόδρομος. Σ'αυτό το κεφάλαιο θα αναπτυχθεί ένα υποτυπώδες πρωτόκολλο τύπου *ping*. Στο πρωτόκολλο, ένας κόμβος θα στέλνει ένα πακέτο σε έναν άλλο κόμβο. Ο κόμβος-παραλήπτης θα επιστρέφει το πακέτο και θα υπολογίζεται ο ολικός χρόνος από τη στιγμή που το πακέτο έφυγε από τον πρώτο κόμβο, μέχρι τη στιγμή που γύρισε πίσω.

Για αρχή, ορίζονται οι δομές που θα χρησιμοποιηθούν από το πρωτόκολλο.

```
struct hdr_ping {
char ret;
double send_time;
};
```

Η πρώτη δομή με το όνομα *hdr_ping* περιέχει τα δεδομένα της επικεφαλίδας (*header)*. Η μεταβλητή *ret* είναι τύπου *char* και έχει τις τιμές '0' αν το πακέτο είναι στο δρόμο από τον αποστολέα στον παραλήπτη και '1' διαφορετικά. Το πρωτόκολλο θα υλοποιηθεί ως μία υποκλάση της ήδη υπάρχουσας κλάσης *Agent* του *ns*. Αυτός είναι ένας τυπικός τρόπος προγραμματισμού στις αντικειμενοστραφής γλώσσας. Αντί να υλοποιήσουμε το πρωτόκολλο ως μία ξεχωριστή κλάση, κάτι το οποίο θα απαιτούσε να ξαναγράψουμε μεγάλο μέρος ήδη υπαρχουσών λειτουργιών της κλάσης *Agent* που είναι κοινός για πολλά πρωτόκολλα, προτιμούμε να κληρονομήσουμε την ήδη υπάρχουσα κλάση. Έτσι ο κώδικας των *Agent* που είναι κοινός με το πρωτόκολλο θα κληρονομηθεί και στην υποκλάση αυτούσιος, ενώ ότι καινούργιο πρέπει να υλοποιηθεί, θα γίνει μέσω της προσθήκης νέων δεδομένων μελών, ή μέσω της τροποποίησης μελών της κλάσης *Agent* μέσω του μηχανισμού *override*. Ο ορισμός της κλάσης *PingAgent* του πρωτοκόλλου που υλοποιείται, είναι ο παρακάτω:

```
class PingAgent : public Agent {
    public:
     PingAgent();
```

```
 int command(int argc, const char*const* argv);
 void recv(Packet*, Handler*);
protected:
 int off_ping_;
};
```

Η κλάση έχει ως μέλος έναν ακέραιο, τη μεταβλητή *off_ping_* που χρησιμεύει για προσπέλαση της επικεφαλίδας, όπως θα φανεί αργότερα. Τα σώματα των συναρτήσεων *command* και *recv* θα πρέπει βεβαίως να δοθούν, αλλά πρώτα θα δοθεί ο κώδικας διασύνδεσης μεταξύ της *tcl* και της *C++*.

```
static class PingHeaderClass : public PacketHeaderClass {
public:
  PingHeaderClass() : PacketHeaderClass("PacketHeader/Ping",
                                        sizeof(hdr_ping)) {}
} class_pinghdr;


static class PingClass : public TclClass {
public:
  PingClass() : TclClass("Agent/Ping") {}
  TclObject* create(int, const char*const*) {
    return (new PingAgent());
  }
} class_ping;
```

Η κλάση *PingClass* είναι τυπικό παράδειγμα των κλάσεων που κληρονομούν την κλάση *TclClass*. Η κλάση *TclClass* είναι μία διεπαφή (*pure virtual class)*, μία κλάση δηλαδή για την οποία δε γίνεται να δημιουργηθεί αντικείμενο, και απλά προσφέρεται για να κληρονομηθεί από άλλες κλάσεις στην ιεραρχία. Η κλάση είναι στατική, δηλαδή δε χρειάζεται να δημιουργηθεί αντικείμενο ρητά με κώδικα του χρήστη (για την ακρίβεια δεν επιτρέπεται καν μία τέτοια λειτουργία), αλλά το αντικείμενο δημιουργείται αυτόματα με τη δήλωση της κλάσης. Το αντικείμενο έχει το όνομα που δηλώνεται στο τέλος, στη συγκεκριμένη περίπτωση *class_ping*. Η μέθοδος *create* δημιουργεί το αντικείμενο, στην περίπτωση που εξετάζεται δημιουργεί απλά ένα νέο δείκτη σε αντικείμενο τύπου *TclObject* καλώντας τον *constructor* της κλάσης *PingAgent*.

Ομοίως, δημιουργείται και το στατικό αντικείμενο *_pinghdr* το οποίο είναι τύπου *PingHeaderClass.* Η *PingHeaderClass* δημιουργεί και διαχειρίζεται την επικεφαλίδα του πρωτόκολλου και κληρονομεί την κλάσση *PacketHeaderClass* που είναι ήδη υλοποιημένη στο *ns*.

Η συνάρτηση *command* της κλάσης *PingAgent* καλείται όταν εκτελέσουμε μία εντολή για αντικείμενο τύπου *PingAgent*. Αν για παράδειγμα

μία μεταβλητή *$pa* είναι τύπου *PingAgent* (η πιο σωστά τύπου *Agent/Ping*, όπως έχει οριστεί στην αντιστοιχία μεταξύ *Tcl* και *C++* στην κλάση *PingClass*) τότε για παράδειγμα η έκφραση σε κώδικα *tcl*

$pa put 1 2

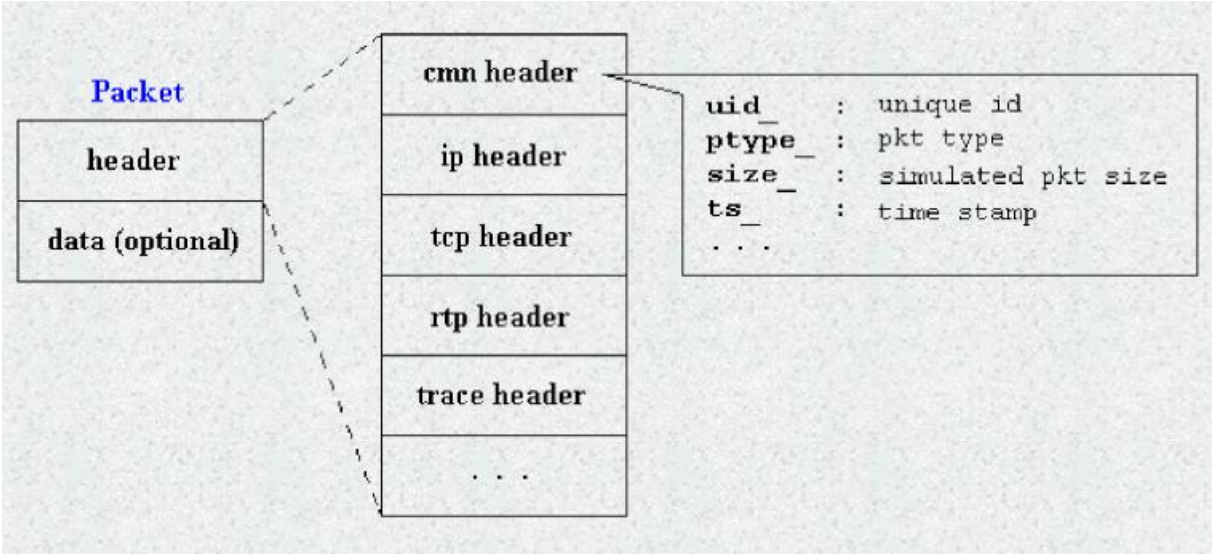
θα προκαλέσει την κλήση της *command* με ορίσματα τις συμβολοσειρές *"put", "1"* και *"2"*. Στο σενάριο που μελετάται, η εντολή για να στείλει πακέτο ο αποστολέας θα είναι η *send*. Ο παρακάτω κώδικας υλοποιεί την *command*

```
int PingAgent::command(int argc, const char*const* argv)
{
  if (argc == 2) {
    if (strcmp(argv[1], "send") == 0) {
      // Δημιουργία νέου πακέτου
      Packet* pkt = allocpkt();
      // Δημιουργία δείκτη στην επικεφαλίδα του νέου πακέτου
      hdr_ping* hdr = (hdr_ping*)pkt->access(off_ping_);
      // Set the 'ret' field to 0, so the receiving node knows
      // that it has to generate an echo packet
      // Το πεδίο ret παίρνει την τιμή 0, ώστε ο παραλήπτης να
      //γνωρίζει ότι πρέπει να στείλει πίσω απάντηση
      hdr->ret = 0;
      //Αποθήκευση της τρέχουσας ώρας
      hdr->send_time = Scheduler::instance().clock();
      // Αποστολή πακέτου
      send(pkt, 0);
      // return TCL_OK, so the calling function knows that the
      // command has been processed
      // επιστροφή του κωδικού TCL_OK ώστε η καλούσα
      // συνάρτηση να γνωρίζει ότι η εντολή έχει εκτελεσθεί
      return (TCL_OK);
    }
  }
  // Κάλεσε την command της κλάσης πατέρα, αν η εντολή
  // δεν είναι η send
  return (Agent::command(argc, argv));
}
```

Η συνάρτηση recv καλείται όταν λαμβάνεται ένα πακέτο που ανήκει στο πρωτόκολλό μας.

```
void PingAgent::recv(Packet* pkt, Handler*)
{

  hdr_ip* hdrip = (hdr_ip*)pkt->access(off_ip_);
  hdr_ping* hdr = (hdr_ping*)pkt->access(off_ping_);
  // Is the 'ret' field = 0 (i.e. the receiving node is being pinged)?
 // Αν ret == 0 τότε ο κόμβος έχει λάβει ένα πακέτο ping
  if (hdr->ret == 0) {
    // Σώσε τον χρόνο αποστολής του πακέτου
    double stime = hdr->send_time;
    // Πέτα το πακέτο
    Packet::free(pkt);
    // Δημιούργησε νέο πακέτο
    Packet* pktret = allocpkt();
    // Δημιούργησε δείκτη για την επικεφαλίδα του νέου πακέτου
    hdr_ping* hdrret = (hdr_ping*)pktret->access(off_ping_);
    // Θέσε ret=1 ώστε το πακέτο να μην ξανααποσταλεί από
   // τον κόμβο που έστειλε το ping
    hdrret->ret = 1;
    // Άλλαξε την ώρα
    hdrret->send_time = stime;
    // Στείλε το πακέτο
    send(pktret, 0);
  } else {
    // Ένα ping γύρισε πίσω στον αποστολέα
    char out[100];
    // Υπολόγισε τον συνολικό χρόνο
    sprintf(out, "%s recv %d %3.1f", name(),
          hdrip->src_.addr_ >> Address::instance().NodeShift_[1],
          (Scheduler::instance().clock()-hdr->send_time) * 1000);
    Tcl& tcl = Tcl::instance();
    tcl.eval(out);
    // Πέτα το πακέτο
    Packet::free(pkt);
  }
}
```

3.4Ανάλυση αποτελεσμάτων μέσω της awk.



3.4.1 Ανάλυση αποτελεσμάτων μέσω της awk.

Τα αποτελέσματα των εξομοιώσεων αναλύθηκαν μέσω προγραμμάτων που γράφτηκαν στη γλώσσα awk. Η awk είναι μία γλώσσα που επιτρέπει τη γρήγορη κι εύκολη διαχείριση γραμματοσειρών. Βασίζεται στη χρήση κανονικών εκφράσεων (regular expressions) σύμφωνα με τις οποίες η είσοδος αναλύεται. Για κάθε regular expression μπορούμε να ορίσουμε τον κώδικα ο οποίος θα εκτελεσθεί. Για παράδειγμα ο κανόνας:

```
/^r/ {
        data+=$6;
        packets++;
        sumDelay += $2
}
```

Ορίζει ότι σε περίπτωση που η τρέχουσα γραμμή ξεκινάει με το γράμμα *r* (που όπως είδαμε σημαίνει επιτυχής λήψη πακέτου), τότε θα προστεθεί στη μεταβλητή *data* η τιμή που βρίσκεται στην έκτη στήλη (*data+=$6)*, ο αριθμός πακέτων αυξάνεται κατά ένα, και προστίθεται η τιμή της δεύτερης στήλης στη συνολική καθυστέρηση.

Στο επόμενο υποκεφάλαιο δίνεται μία πιο λεπτομερής εισαγωγή στα κύρια χαρακτηριστικά της awk.

Εισαγωγή στη γλώσσα *awk*

Η *awk* είναι ένα εργαλείο που επιτρέπει την αναζήτηση κανόνων εκφράσεων (*patterns)* σε ένα κείμενο. Έχει αρκετές ομοιότητες με τη γλώσσα προγραμματισμού C. Η γενική σύνταξη της εντολής *awk* είναι:

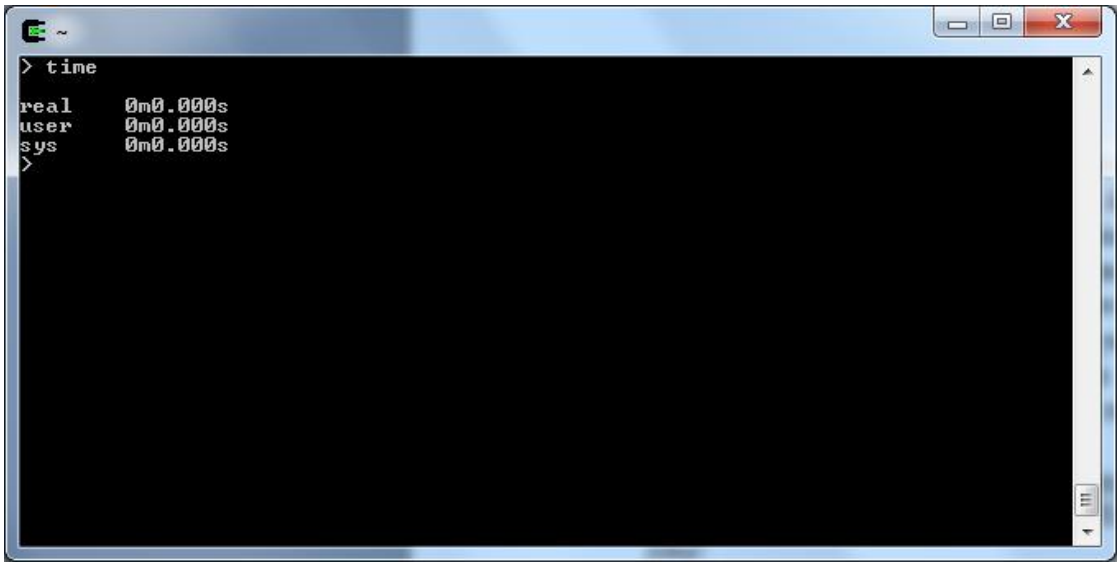
awk '*προγραμμα*' <*ονοματα_αρχείων*>

Το μη τερματικό σύμβολο '*πρόγραμμα*' αναλύεται ως εξής:

pattern {action}
pattern {action}
…

Ώστε ένα '*πρόγραμμα*' είναι μία σειρά από *patterns* ακολουθούμενο από μία σειρά ενεργειών (*actions)*. Η *awk* διαβάζει τα αρχεία που δίνονται από το όρισμα <*ονόματα_αρχείων*> μία γραμμή τη φορά και για κάθε *pattern* που ταιριάζει με τη γραμμή, εκτελείται η αντίστοιχη ενέργεια. Κάθε *pattern* μπορεί να είναι μία κανονική έκφραση. Αν το *pattern* παραλείπεται, τότε ο κανόνας ισχύει για κάθε γραμμή. Αντιθέτως, αν η ενέργεια παραλείπεται, τότε εκτελείται η προεπιλεγμένη ενέργεια που είναι η *print*, η απλή εκτύπωση δηλαδή της γραμμής, η οποία ταιριάζει με το *pattern*.

Η *awk* χωρίζει την είσοδο σε πεδία (*fields)*. Τα πεδία χωρίζονται με χαρακτήρες κενού η στηλοθέτησης. Έστω για παράδειγμα η εντολή *time*:

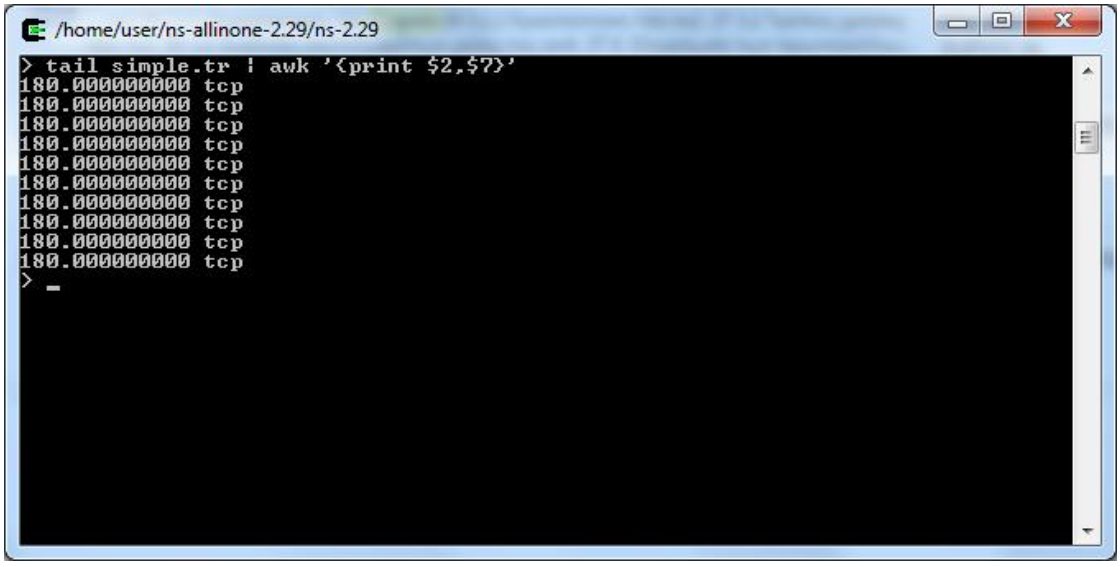


Εικόνα 3.15. Έξοδος της εντολής time.

Κάθε γραμμή έχει δύο πεδία, η 1η γραμμή για παράδειγμα έχει στο 1ο πεδίο την τιμή «real» και στο 2ο την τιμή «0m0.000s». Οι τιμές των πεδίων κάθε γραμμής προσπελαύνονται μέσω των μεταβλητών *$1,$2,…$NF*, όπου *NF* είναι η μεταβλητή που δηλώνει το πλήθος των πεδίων (Σημείωση: Οι μεταβλητές *NF* και *$NF* είναι διαφορετικές. Η 1η είναι το πλήθος των πεδίων, ενώ η 2η η τιμή του τελευταίου πεδίου. Π.χ. για το παράδειγμά μας, για την πρώτη γραμμή ισχύει *$NF=0M0.000s*, ενώ *NF=2)*. Έστω για παράδειγμα, ότι θέλουμε να εμφανίσουμε μόνο τους χρόνους και το είδος πρωτοκόλλου ενός *trace file*, τα οποία βρίσκονται στα πεδία 2 και 7 αντίστοιχα. Με την *awk* αυτό γίνεται πολύ εύκολα μέσω της *print*. Καλούμε απλά την εντολή:

awk '{print $2, $7}'

με είσοδο την εντολή εμφάνισης του αρχείου (στο παράδειγμα χρησιμποιήθηκε η *tail* αντί της *cat* ώστε να εμφανιστούν μόνο οι τελευταίες γραμμές για χάρη ευκρίνειας).



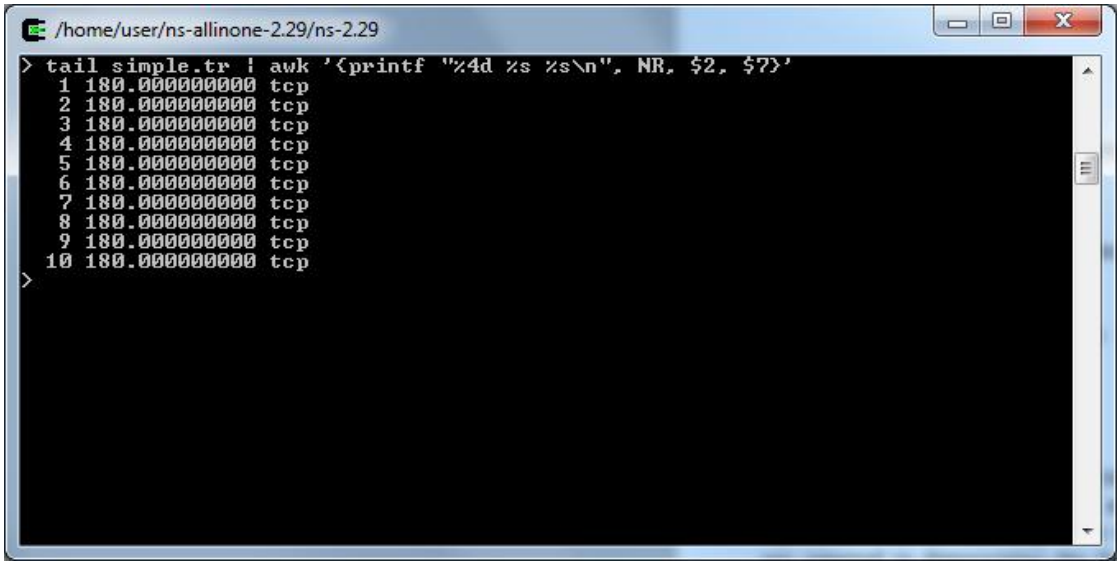
Εικόνα 3.16. Εκτέλεση της awk *με όρισμα τις τελευταίες γραμμές του αρχείου* simple.tr. *Η εντολή* tail *έχει ως έξοδο τις τελευταίες γραμμές του αρχείου-ορίσματος. Η έξοδός της ανατροφοδοτείται ως είσοδος στην* awk *η οποία απλά εκτυπώνει για κάθε γραμμή τα στοιχεία της 2ης και 7ης στήλης.*

Σε περίπτωση που ο διαχωριστικός χαρακτήρας πεδίων δεν είναι οι κενοί χαρακτήρες (κάτι που μπορεί να συμβεί αν ο χρήστης θέλει ειδική διαμόρφωση των αρχείων *trace)*, γίνεται να ορισθεί μέσω της παραμέτρου *–F* ο εναλλακτικός διαχωριστικός χαρακτήρας. Π.χ. η εντολή:

awk –F: '{print $2, $7}'

λέει στην *awk* να εκτυπώση πάλι το 2ο και 7ο πεδίο κάθε γραμμής, με τη διαφορά ότι τώρα τα πεδία χωρίζονται με τον χαρακτήρα ':'.

Η μεταβλητή *NR* κρατάει τον τρέχοντα αριθμό γραμμής. Η εκτύπωση των αποτελεσμάτων μπορεί να διαμορφωθεί μέσω της *printf* αν δεν είναι επιθυμητή η προκαθορισμένη διαμόρφωση. Η *printf*, η οποία μοιάζει με την συνάρτηση της καθιερωμένης βιβλιοθήκης εισόδου/εξόδου της C, επιτρέπει στον χρήστη να καθορίσει συγκεκριμένη διαμόρφωση:



Εικόνα 3.16. Η δεσμευμένη μεταβλητή NR έχει ως τιμή τον αριθμό της τρέχουσας γραμμής.

Στο παραπάνω παράδειγμα η *awk* εκτυπώνει ως αλφαριθμητικά τις μεταβλητές *$2* και *$7*, όπως και πριν. Η διαμόρφωση ως αλφαριθμητικό γίνεται μέσω του ορίσματος *%s* της *printf*. Επιπλέον εκτυπώνει τον αριθμό γραμμής που περιέχεται στη μεταβλητή *NR*, αφήνοντας διάστημα τεσσάρων ψηφίων (παράμετρος *%d*).

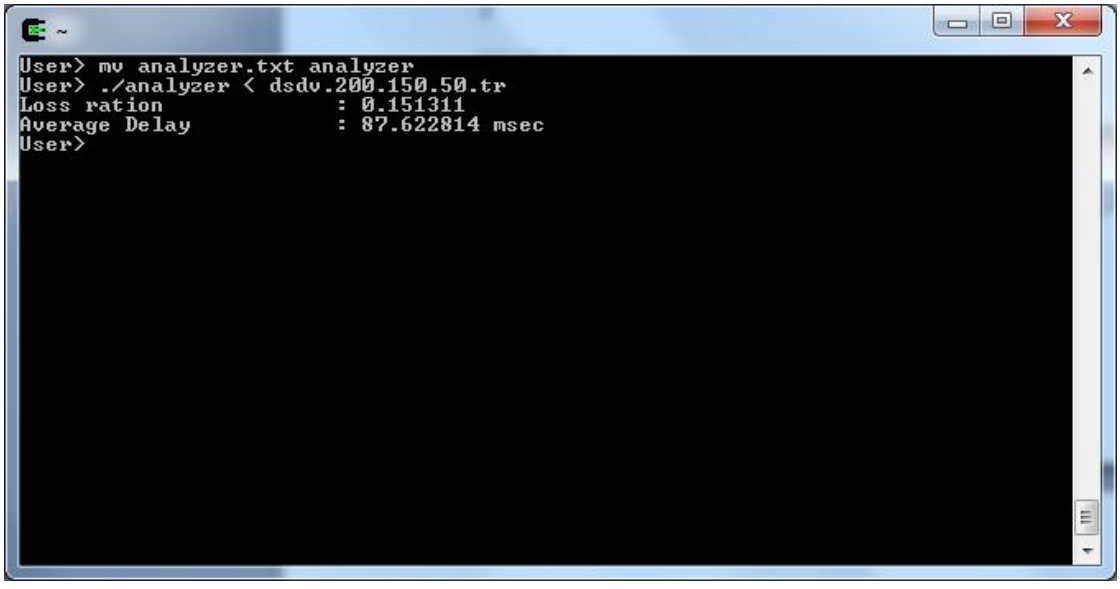
Κανόνες Έκφρασης (*Patterns)* στην *awk*

Παρακάτω δίνεται ο κώδικας ενός ολοκληρωμένου προγράμματος γραμμένου στην awk, το οποίο υπολογίζει τη μέση καθυστέρηση πακέτων:

```
awk '
BEGIN {
        data=0;
        packets=0;
        totalpackets=0;
```

```
        totalDelay=0;
}
        /^s/&&/cbr/ {
        sendtimes[$5]=$2
}
/^r/ {
        totalpackets++;
        packets++;
        sumDelay[$5] = $2 - sendtimes[$5]
        totalDelay += sumDelay[$5]
}
/^D/ {
        totalpackets++;
}
END{
printf("Loss ration\t\t: %f\n", 1.0- packets/totalpackets)
printf("Average Delay\t\t: %f  msec\n", (1.0 * totalDelay )/ packets);
}'
```

Ο κώδικας αυτός αρκεί να αποθηκευτεί σε ένα αρχείο στο οποίο ο χρήστης έχει εξουσίες εκτέλεσης. Εκτελείται μέσω της ανακατεύθηνσης της καθιερωμένη εισόδου, που στα unix-like λειτουργικά συστήματα μπορεί να επιτευχθεί με το σύμβολο ‘<’.



Εικόνα 3.17. Τυπική έξοδος του προγράμματος «analyzer» που υλοποιήθηκε σε awk.

4. Εξομοίωση των πρωτοκόλλων

Για την εξομοίωση θα χρησιμοποιηθεί μία τοπολογία που αποτελείται από ένα grid (πλέγμα) *n* κόμβων με *n=200*. Το grid (ο πίνακας που περιγράφει τον χώρο) θα είναι διαστάσεων 500x500. Αυτό σημαίνει ότι οι κόμβοι θα μπορούν να κινηθούν εντός του ορίου του πίνακα. Αν κάθε νούμερο της κλίμακας αντιστοιχεί στην ελάχιστη διακριτή απόσταση που μπορεί να διανύσει ένας κόμβος, τότε το grid αυτό περιγράφει έναν χώρο 500x500 μονάδων απόστασης.
Από τους 200 κόμβους οι 150 θα εκπέμπουν και οι 50 θα λαμβάνουν. Αυτό γίνεται ώστε να δημιουργηθεί ένα μη ιδανικό σενάριο, στο οποίο θα δημιουργηθούν κόμβοι συμφόρησης λόγω της ασυμμετρίας του αριθμού δεκτών και ληπτών.
Το είδος της κίνησης θα είναι πακέτα tcp, και συγκεκριμένα του πρωτοκόλλου ftp. Οι κόμβοι θα κινούνται τυχαία στο πλέγμα.
Ο κώδικας που περιγράφει την τοπολογία του δικτύου βρίσκεται στο παράρτημα. Για κάθε πρωτόκολλο, αλλάζει την τιμή της μεταβλητής *adHocRouting*.
Τα αποτελέσματα αναλύονται μέσω της awk και υπολογίζεται η μέση καθυστέρηση, η διασπορά της καθυστέρησης καθώς και το ποσοστό απώλειας (loss ratio). Η μέση καθυστέρηση υπολογίζεται ως εξής: Για κάθε πακέτο, μέσω του id του, κρατάμε τον χρόνο κατά τον οποίο αυτό στάλθηκε. Αυτό γίνεται βρίσκοντας τις γραμμές που ξεκινάνε με το γράμμα ‘s’ και αποθηκεύοντας το χρόνο αποστολής στην αντίστοιχη θέση ενός πίνακα. Αν π.χ. τα πακέτα με id 78 και 45 στάλθηκαν στις χρονικές στιγμές 102 και 108 αντίστοιχα τότε ο πίνακας θα έχει τις τιμές:

Sendtimes[78]=102
Sendtimes[45]=108

Αν για κάθε πακέτο αφαιρέσουμε αυτόν τον χρόνο από το χρόνο άφιξης, λαμβάνουμε τον συνολικό χρόνο που έκανε το πακέτο για να φτάσει.
Υπολογίζουμε τα μεγέθη $E(X)$ και $Var(X) = E(X^2)-E(X)^2$. Η μέση τιμή δίνει την μέση καθυστέρηση λήψης κάθε πακέτου σε δευτερόλεπτα. Υπολογίζουμε και τη διασπορά ως μέτρο του κατά πόσο ο μέσος «πελάτης» θα βλέπει χρόνους κοντά στη μέση τιμή. Χαμηλή διασπορά σημαίνει ότι ο κάθε κόμβος θα λαμβάνει τα μηνύματα περίπου στο χρόνο που δίνεται από τη μέση τιμή, ενώ αντίθετα μεγάλη διασπορά σημαίνει ότι θα υπάρχουν μεγάλες διακυμάνσεις στους χρόνους απόκρισης γύρω από τη μέση τιμή.

Στον παρακάτω πίνακα φαίνονται τα αποτελέσματα της εξομοίωσης:

	DSDV	DSR	AODV	TORA	WRP
E(t)	0.0087	0.0074	0.0056	0.0091	0.0080

Var(t)	0.2790	0.3105	0.1441	0.2012	0.122
Loss Ratio	0.151	0.005	0.888	0.1801	0.015

5.Συζήτηση αποτελεσμάτων

Τα αποτελέσματα της εξομοίωσης θα αναλυθούν για κάθε πρωτόκολλο ξεχωριστά:

DSDV
Το *DSDV* έχει σχετικά μεγάλη μέση καθυστέρηση και μάλιστα με μεγάλη διασπορά. Αυτό οφείλεται κατά πάσα πιθανότητα στο γεγονός ότι το *DSDV* αν και αρχικά βρίσκει πιθανότατα βέλτιστες διαδρομές, καθώς η τοπολογία αλλάζει, οι διαδρομές δεν παραμένουν βέλτιστες.

DSR
Το *DSR* φαίνεται να έχει αρκετά μικρό ποσοστό απόρριψης πακέτων, κάτι που φαίνεται ενδιαφέρον, αν αναλογισθεί κανείς ότι τα on-demand πρωτόκολλα γενικώς απορρίπτουν αρκετά πακέτα κατά τη διαδικασία εύρεσης διαδρομής. Πιθανόν αυτό να οφείλεται στο γεγονός ότι το *DSR* ελέγχει αν οι διαδρομές που έχουν βρεθεί παραμένουν έγκυρες με αποτέλεσμα να αποφεύγονται άκυρες διαδρομές. Αν κι έχει καλή συμπεριφορά όσον αφορά τη μέση καθυστέρηση, η διασπορά είναι αρκετά μεγάλη.

AODV
Το *AODV* έχει πολύ μεγάλο ποσοστό απόρριψης. Αυτό πρέπει να προέρχεται από τις απορρίψεις πακέτων κατά την εύρεση διαδρομής, καθώς οι χρόνοι του *AODV* δε φαίνεται να είναι πολύ καλοί σε σχέση με το ποσοστό απόρριψης πακέτων.

TORA
Το *TORA* έχει τη χειρότερη συμπεριφορά όσον αφορά τους χρόνους απόκρισης. Αυτό οφείλεται μάλλον στο γεγονός ότι έχουμε ταχέως μεταβαλλόμενη τοπολογία δικτύου (high mobility) λόγω της τυχαίας κίνησης των κόμβων. Σ'αυτή την περίπτωση πρωτόκολλα όπως το DSR, που έχουν γρηγορότερη ανεύρεση μονοπατιού, φαίνεται να λειτουργούν καλύτερα. Ίσως αν υπήρχε μεγαλύτερη κίνηση στο δίκτυο το TORA να είχε καλύτερη συμπεριφορά.

WRP
Το WRP φαίνεται να έχει γενικώς καλή συμπεριφορά. Όπως και στο DSR αυτή η συμπεριφορά δικαιολογείται μάλλον από την μεταβαλλόμενη τοπολογία, η οποία απαιτεί πρωτόκολλα που έχουν γρήγορη ανεύρεση μονοπατιού.

Συμπεράσματα

Από τα αποτελέσματα των εξομοιώσεων, προκύπτει ότι τα πρωτόκολλα που έχουν ταχείς αλγορίθμους ανεύρεσης διαδρομής, έχουν καλύτερη συμπεριφορά όσον αφορά τους χρόνους απόκρισης. Το ποσοστό απόρριψης πακέτων δε φαίνεται να έχει μεγάλη συσχέτιση με τους χρόνους απόρριψης, με την έννοια ότι πρωτόκολλα με χειρότερο ποσοστό απόρριψης μπορεί να έχουν καλύτερους μέσους χρόνους. Αυτό συμβαίνει κυρίως στα on-demand πρωτόκολλα δρομολόγησης, στα οποία κατά την διαδικασία ανεύρεσης διαδρομής, μπορεί να απορριφθούν πολλά πακέτα. Για τα πρωτόκολλα αυτά επομένως, προκύπτει το συμπέρασμα ότι το ποσοστό απόρριψης πακέτων δεν αποτελεί ασφαλή μετρική.

Έγινε προσπάθεια ώστε το σενάριο της εξομοίωσης να έχει τόσο μεγάλη μεταβλητότητα τοπολογίας δικτύου (μέσω της τυχαίας κίνησης των κόμβων), όσο και πολλές συνδέσεις σχετικά με τον αριθμό των κόμβων. Με το συγκεκριμένο σενάριο, 200 τυχαία κινούμενων κόμβων με τους 150 να στέλνουν δεδομένα στους υπόλοιπους 50, φάνηκε μία ξεκάθαρη υπεροχή των πρωτοκόλλων που έχουν γρήγορους χρόνους εύρεσης διαδρομής.

Βιβλιογραφία

[1] Introduction to Network Simulator NS2 ,Teerawat Issariyakul, Ekram Hossain

[2] Unix Programming Environment (Prentice-Hall Software Series), **Brian W. Kernighan, Rob Pike**

[3] The Network Simulator ns-2: Documentation, **http://www.isi.edu/nsnam/ns/ns-documentation.html**

[4] Ad Hoc Wireless Networks: Architectures and Protocols
C. Siva Ram MurthyB.S. Manoj

[5] Cygwin User's Guide, **http://www.cygwin.com/cygwin-ug-net/cygwin-ug-net.html**

[6] An Introduction to NS, Nam and OTcl
Scripting, Paul Meeneghan and Declan Delaney
http://www.cygwin.com/cygwin-ug-net/cygwin-ug-net.html

[7] Ns by Example, Jae Chung and Mark Claypool
http://nile.wpi.edu/NS/

Παράρτημα - Κώδικας Εξομοίωσης

```
#===================================================================
# Define options
#===================================================================
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(ant) Antenna/OmniAntenna ;# Antenna type
set val(ll) LL ;# Link layer type
set val(ifq) Queue/DropTail/PriQueue ;# Interface queue type
#set val(ifq) CMUPriQueue
set val(ifqlen) 50 ;# max packet in ifq
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac/802_11 ;# MAC type
set val(rp) DSDV;# ad-hoc routing protocol
set val(nn) 200;# number of mobilenodes
set val(np) 200; #number of mobile nodes exchanging data
set val(tc) 150;
set val(ts) 50;
```

```
set ns_ [new Simulator]

set tracefd [open simple.tr w]
$ns_ trace-all $tracefd

set topo [new Topography]
$topo load_flatgrid 500 500

create-god $val(nn)

# Configure nodes
$ns_ node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-channelType $val(chan) \
-agentTrace OFF \
-routerTrace ON \
-macTrace ON \
-movementTrace OFF
for {set i 0} {$i < $val(tc) } {incr i} {
set node_t($i) [$ns_ node ]
#set node_t($i) [new Node/MobileNode]
$node_t($i) random-motion 1 ;# enable random motion
}

for {set i 0} {$i < $val(ts)} {incr i} {
set node_s($i) [$ns_ node ]
#set node_s($i) [new Node/MobileNode ]
$node_s($i) random-motion 1 ;# enable random motion
}

#$node_(0) set X_ 5.0
#$node_(0) set Y_ 2.0
#$node_(0) set Z_ 0.0
#$node_(1) set X_ 390.0
#$node_(1) set Y_ 385.0
#$node_(1) set Z_ 0.0
```

```
# Node_(1) starts to move towards node_(0)
#$ns_ at 50.0 "$node_(1) setdest 25.0 20.0 15.0"
#$ns_ at 10.0 "$node_(0) setdest 20.0 18.0 1.0"
# Node_(1) then starts to move away from node_(0)
#$ns_ at 100.0 "$node_(1) setdest 490.0 480.0 15.0"

for {set i 0} {$i < $val(tc) } {incr i} {
set tcp_($i) [new Agent/TCP]
$ns_ attach-agent $node_t($i) $tcp_($i)
}

for {set i 0} {$i < $val(ts) } {incr i} {
#set sink_($i) [new Agent/TCPSink]
set sink_($i) [new Agent/LossMonitor]
$ns_ attach-agent $node_s($i) $sink_($i)
}

for {set i 0} {$i < $val(tc) } {incr i} {

for {set j 0} {$j < $val(ts) } {incr j} {
$ns_ connect $tcp_($i) $sink_($j)
}
}


for {set i 0} {$i < $val(tc) } {incr i} {
set ftp_($i) [new Application/FTP]
$ftp_($i) attach-agent $tcp_($i)
$ns_ at 10.0 "$ftp_($i) start"
}

for {set i 0} {$i < $val(tc) } {incr i} {
$ns_ at 180.0 "$node_t($i) reset";
}
for {set i 0} {$i < $val(ts) } {incr i} {
$ns_ at 180.0 "$node_s($i) reset";
}

$ns_ at 180.0 "finish"
#$ns_ at 150.0002 "puts \"NS EXITING...\" ;
#$ns_ halt
proc stop {} {
```

```
        global ns_ tracefd
        $ns_ flush-trace
        close $tracefd
}



proc finish {} {
 $ns_ flush-trace
 close $tracefd
}

puts "Starting Simulation..."
$ns_ run
```

Κώδικας awk για την ανάλυση του trace file

```
awk '
BEGIN {
data=0;
packets=0;
totalpackets=0;
totalDelay=0;
}
/^s/&&/cbr/ {
sendtimes[$5]=$2
}
/^r/ {
totalpackets++;
packets++;
sumDelay[$5] = $2 - sendtimes[$5]
totalDelay += sumDelay[$5]
totalDelaySqr += sumDelay[$5]^2
}
/^D/ {
        totalpackets++;
}
END{
printf("Loss ration\t\t: %f\n", 1.0- packets/totalpackets)
printf("Average Delay\t\t: %f msec\n", (1.0 * totalDelay )/ packets);
printf("Delay Variance\t\t: %f \n", totalDelaySqr/packets- ((1.0 * totalDelay )/
packets)^2);
}
```

'

A simple wireless example file that simulates a 22-mobilenode topology.
#Traffic used are CBR flows .

#
Default Script Options
#

```
set opt(chan)           Channel/WirelessChannel
set opt(prop)           Propagation/TwoRayGround
set opt(netif)          Phy/WirelessPhy
set opt(mac)            Mac/802_11
set opt(ifq)        Queue/DropTail/PriQueue
set opt(ll)         LL
set opt(ant)            Antenna/OmniAntenna

set opt(x)          50     ;# X dimension of the tography
set opt(y)          50    ;# Y dimension of the topography
set opt(cp)         "../ns-2.1b9a/tcl/mobility/scene/cbr-22-test"
set opt(sc)         "../ns-2.1b9a/tcl/mobility/scene/scen-22-test"

set opt(ifqlen)         50              ;# max packet in ifq
set opt(nn)         22              ;# number of nodes
set opt(seed)           0.0
set opt(stop)           500.0       ;# simulation time
set opt(tr)         out-sen22.tr  ;# trace file
set opt(rp)          aodv         ;# routing protocol script
set opt(lm)          "off"         ;# log movement

#
set AgentTrace              ON
set RouterTrace             ON
set MacTrace                OFF
```

```
LL set mindelay_        50us
LL set delay_           25us
LL set bandwidth_       0       ;# not used

Agent/Null set sport_       0
Agent/Null set dport_       0

Agent/CBR set sport_        0
Agent/CBR set dport_        0

Agent/TCPSink set sport_    0
Agent/TCPSink set dport_    0

Agent/TCP set sport_        0
Agent/TCP set dport_        0
Agent/TCP set packetSize_   1460

Queue/DropTail/PriQueue set Prefer_Routing_Protocols    1

# unity gain, omni-directional antennas
# set up the antennas to be centered in the node and 1.5 meters above it
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

# Initialize the SharedMedia interface with parameters to make
# it work like the 914MHz Lucent WaveLAN DSSS radio interface
Phy/WirelessPhy set CPThresh_ 10.0
Phy/WirelessPhy set CSThresh_ 1.559e-11
Phy/WirelessPhy set RXThresh_ 3.652e-10
Phy/WirelessPhy set Rb_ 2*1e6
Phy/WirelessPhy set Pt_ 0.2818
Phy/WirelessPhy set freq_ 914e+6
Phy/WirelessPhy set L_ 1.0

#
==========================================================
==============

proc usage { argv0 }  {
        puts "Usage: $argv0"
```

```
        puts "\tmandatory arguments:"
        puts "\t\t\[-x MAXX\] \[-y MAXY\]"
        puts "\toptional arguments:"
        puts "\t\t\[-cp conn pattern\] \[-sc scenario\] \[-nn nodes\]"
        puts "\t\t\[-seed seed\] \[-stop sec\] \[-tr tracefile\]\n"
}


proc getopt {argc argv} {
        global opt
        lappend optlist cp nn seed sc stop tr x y

        for {set i 0} {$i < $argc} {incr i} {
                set arg [lindex $argv $i]
                if {[string range $arg 0 0] != "-"} continue

                set name [string range $arg 1 end]
                set opt($name) [lindex $argv [expr $i+1]]
        }
}


proc cmu-trace { ttype atype node } {
        global ns_ tracefd

        if { $tracefd == "" } {
                return ""
        }
        set T [new CMUTrace/$ttype $atype]
        $T target [$ns_ set nullAgent_]
        $T attach $tracefd
       $T set src_ [$node id]

       $T node $node

        return $T
}


proc create-god { nodes } {
        global ns_ god_ tracefd

        set god_ [new God]
```

```
        $god_ num_nodes $nodes
}

proc log-movement {} {
    global logtimer ns_ ns

    set ns $ns_
    source ../ns-2.1b9a/tcl/mobility/timer.tcl
    Class LogTimer -superclass Timer
    LogTimer instproc timeout {} {
        global opt node_;
        for {set i 0} {$i < $opt(nn)} {incr i} {
            $node_($i) log-movement
        }
        $self sched 0.1
    }

    set logtimer [new LogTimer]
    $logtimer sched 0.1
}

#
# Main Program
#
getopt $argc $argv

#
# Source External TCL Scripts
#
source ../ns-2.1b9a/tcl/lib/ns-mobilenode.tcl
source ../ns-2.1b9a/tcl/mobility/$opt(rp).tcl
source ../ns-2.1b9a/tcl/lib/ns-cmutrace.tcl

# do the get opt again incase the routing protocol file added some more
# options to look for

getopt $argc $argv

if { $opt(x) == 0 || $opt(y) == 0 } {
```

```
        usage $argv0
        exit 1
}

if {$opt(seed) > 0} {
        puts "Seeding Random number generator with $opt(seed)\n"
        ns-random $opt(seed)
}

#
# Initialize Global Variables
#
set ns_      [new Simulator]
set chan     [new $opt(chan)]
set prop     [new $opt(prop)]
set topo     [new Topography]
set tracefd  [open $opt(tr) w]

set nf [open sen22.nam w]
$ns_ namtrace-all-wireless $nf $opt(x) $opt(y)


$topo load_flatgrid $opt(x) $opt(y)

$prop topography $topo

#
# Create God
#
create-god $opt(nn)

#
# log the mobile nodes movements if desired
#
if { $opt(lm) == "on" } {
   log-movement
}

#
#  Create the specified number of nodes $opt(nn) and "attach" them
#  the channel.
#  Each routing protocol script is expected to have defined a proc
#  create-mobile-node that builds a mobile node and inserts it into the
```

```
#  array global $node_($i)
#

if { [string compare $opt(rp) "dsr"] == 0} {
        for {set i 0} {$i < $opt(nn) } {incr i} {
                dsr-create-mobile-node $i
        }
} elseif { [string compare $opt(rp) "dsdv"] == 0} {
        for {set i 0} {$i < $opt(nn) } {incr i} {
                dsdv-create-mobile-node $i
        }
} elseif { [string compare $opt(rp) "aodv"] == 0} {
        for {set i 0} {$i < $opt(nn) } {incr i} {
                create-mobile-node $i
        }
} elseif { [string compare $opt(rp) "tora"] == 0} {
        for {set i 0} {$i < $opt(nn) } {incr i} {
                create-mobile-node $i
        }
}


#enable node trace in nam

for {set i 0} {$i < $opt(nn)} {incr i} {
    $node_($i) namattach $nf
#4 defines the node size in nam, must adjust it according to your scenario
   $ns_ initial_node_pos $node_($i) 4
}

#
# Source the Connection and Movement scripts
#
if { $opt(cp) == "" } {
        puts "*** NOTE: no connection pattern specified."
       set opt(cp) "none"
} else {
        puts "Loading connection pattern..."
        source $opt(cp)
}

#
```

```
# Tell all the nodes when the simulation ends
#
for {set i } {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).0 "$node_($i) reset";
}
$ns_ at $opt(stop).001 "finish"
$ns_ at $opt(stop).002 "puts \"NS EXITING...\" ; $ns_ halt"


if { $opt(sc) == "" } {
        puts "*** NOTE: no scenario file specified."
        set opt(sc) "none"
} else {
        puts "Loading scenario file..."
        source $opt(sc)
        puts "Load complete..."
}

puts $tracefd "M 0.0 nn $opt(nn) x $opt(x) y $opt(y) rp $opt(rp)"
puts $tracefd "M 0.0 sc $opt(sc) cp $opt(cp) seed $opt(seed)"
puts $tracefd "M 0.0 prop $opt(prop) ant $opt(ant)"


proc finish {} {
    global ns_ nf
    $ns_ flush-trace
    close $nf
    exec ./nam sen22.nam &
}

puts "Starting Simulation..."
$ns_ run
```

- **Κώδικας του πρωτοκόλλου δρομολόγησης AODV**

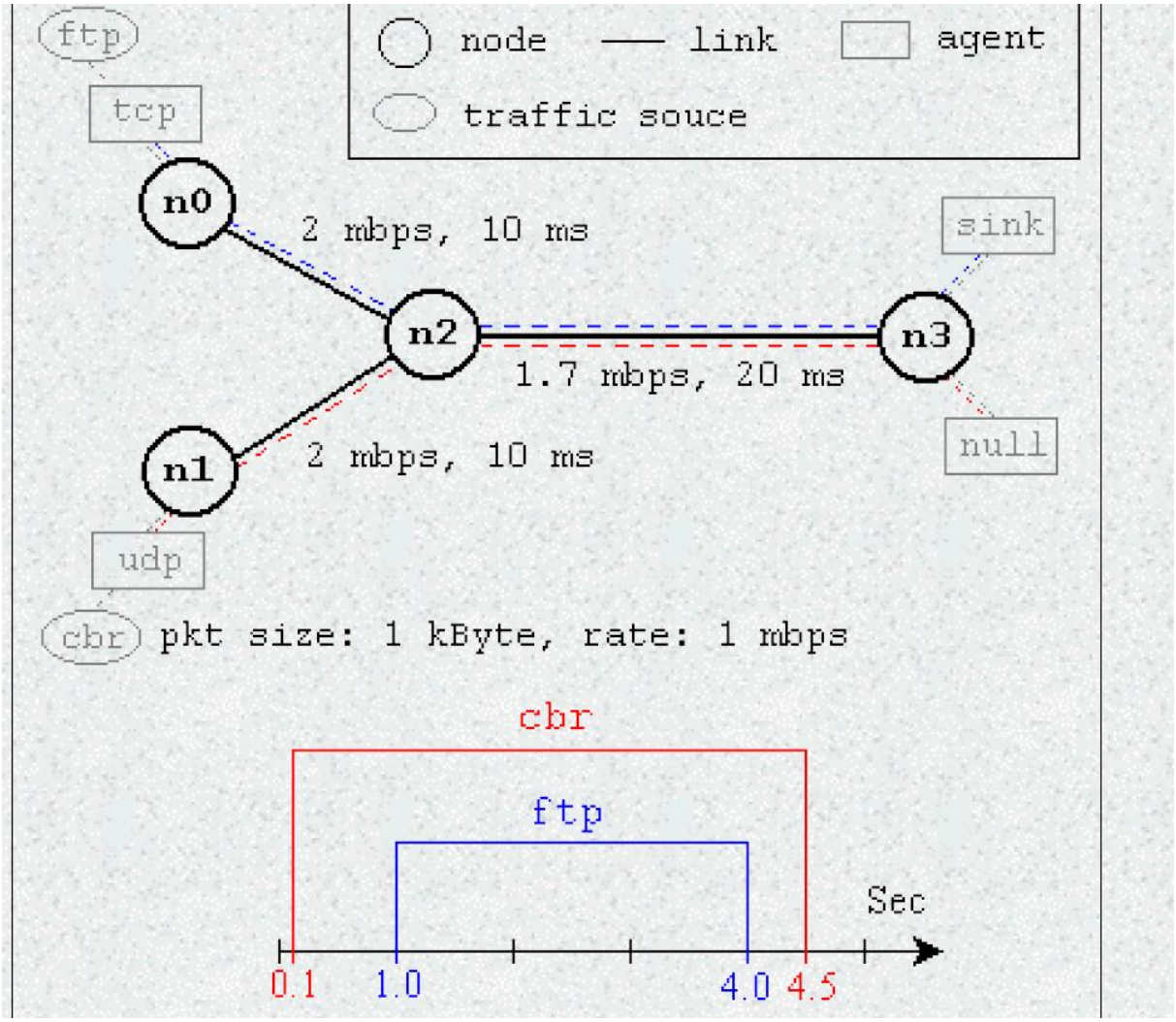
```
# ======================================================================
# Default Script Options
# ======================================================================
```

```
set opt(ragent)             Agent/AODV
set opt(pos)                NONE

if { $opt(pos) != "NONE" } {
        puts "*** WARNING: AODV using $opt(pos) position
configuration..."
}

#
==========================================================
==============
Agent instproc init args {
        $self next $args
}
Agent instproc init args {
        $self next $args
}
Agent/AODV instproc init args {
        $self next $args
}

Agent/AODV set sport_         0
Agent/AODV set dport_         0

#
==========================================================
==============

proc create-routing-agent { node id } {
        global ns_ ragent_ tracefd opt

        #
        #  Create the Routing Agent and attach it to port 255.
        #
        set ragent_($id) [new $opt(ragent) $id]
          set ragent $ragent_($id)
        $node attach $ragent 255

        $ragent if-queue [$node set ifq_(0)]        ;# ifq between LL and
MAC
        $ns_ at 0.$id "$ragent_($id) start"  ;#       start       BEACON/HELLO
Messages
```

```
        #
        # Drop Target (always on regardless of other tracing)
        #
        set drpT [cmu-trace Drop "RTR" $node]
        $ragent drop-target $drpT

        #
        # Log Target
        #
        set T [new Trace/Generic]
        $T target [$ns_ set nullAgent_]
        $T attach $tracefd
        $T set src_ $id
        $ragent log-target $T
}


proc create-mobile-node { id } {
        global ns_ chan prop topo tracefd opt node_
        global chan prop tracefd topo opt

        set node_($id) [new Node/MobileNode]

        set node $node_($id)
        $node random-motion 0               # disable random motion
        $node topography $topo

        #
        # This Trace Target is used to log changes in direction
        # and velocity for the mobile node.
        #
        set T [new Trace/Generic]
        $T target [$ns_ set nullAgent_]
        $T attach $tracefd
        $T set src_ $id
        $node log-target $T

       if ![info exist inerrProc_] {
               set inerrProc_ ""
        }
        if ![info exist outerrProc_] {
                set outerrProc_ ""
        }
```

```
        if ![info exist FECProc_] {
                set FECProc_ ""
        }

       $node add-interface $chan $prop $opt(ll) $opt(mac)  \
                $opt(ifq)  $opt(ifqlen)  $opt(netif)  $opt(ant)  $inerrProc_
$outerrProc_ $FECProc_
        #
        # Create a Routing Agent for the Node
        #
        create-routing-agent $node $id

        #
==========================================================
=====

        if { $opt(pos) == "Box" } {

          set spacing 200
          set maxrow 3
          set col [expr ($id - 1) % $maxrow]
          set row [expr ($id - 1) / $maxrow]
          $node set X_ [expr $col * $spacing]
          $node set Y_ [expr $row * $spacing]
          $node set Z_ 0.0
          $node set speed_ 0.0

          $ns_ at 0.0 "$node_($id) start"

        } elseif { $opt(pos) == "Random" } {

          $node random-motion 1

          $ns_ at 0.0 "$node_($id) start"
       }

}
```

Κώδικας Απλού εισαγωγικού σενάριου εξομοίωσης (πηγή http://nile.wpi.edu/NS/)

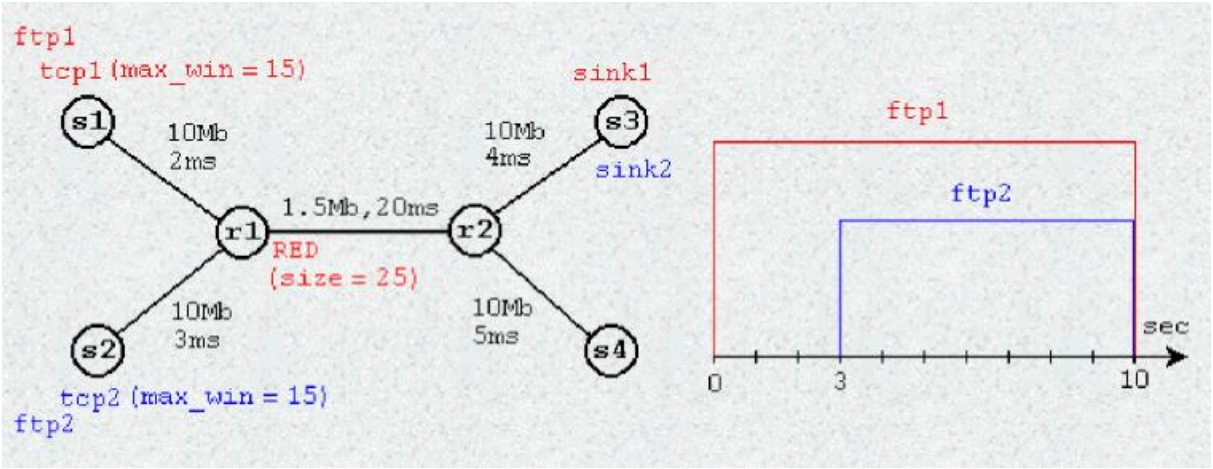



```
#Create a simulator object
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
        global ns nf
```

```
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5


#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

```
$ftp set type_ FTP


#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false


#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

#Run the simulation
$ns run
```

Κώδικας Παρακολούθηση (monitoring) ουράς σε ροή ftp δεδομένων (πηγή http://nile.wpi.edu/NS/, συγγραφέας Polly Huang)

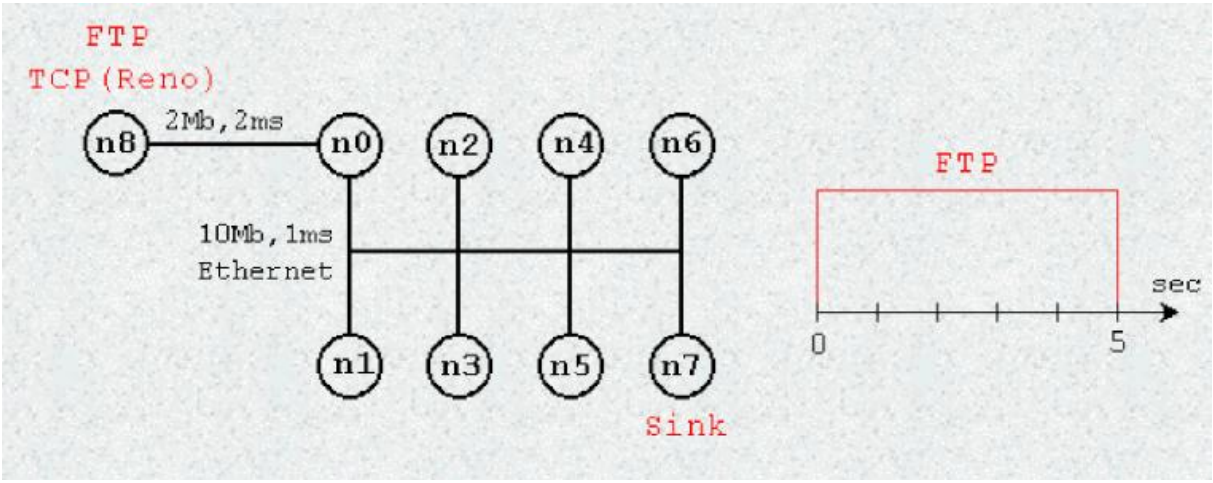



```
set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(r1) [$ns node]
set node_(r2) [$ns node]
set node_(s3) [$ns node]
set node_(s4) [$ns node]

$ns duplex-link $node_(s1) $node_(r1) 10Mb 2ms DropTail
$ns duplex-link $node_(s2) $node_(r1) 10Mb 3ms DropTail
$ns duplex-link $node_(r1) $node_(r2) 1.5Mb 20ms RED
$ns queue-limit $node_(r1) $node_(r2) 25
$ns queue-limit $node_(r2) $node_(r1) 25
$ns duplex-link $node_(s3) $node_(r2) 10Mb 4ms DropTail
$ns duplex-link $node_(s4) $node_(r2) 10Mb 5ms DropTail

$ns duplex-link-op $node_(s1) $node_(r1) orient right-down
$ns duplex-link-op $node_(s2) $node_(r1) orient right-up
$ns duplex-link-op $node_(r1) $node_(r2) orient right
$ns duplex-link-op $node_(r1) $node_(r2) queuePos 0
$ns duplex-link-op $node_(r2) $node_(r1) queuePos 0
$ns duplex-link-op $node_(s3) $node_(r2) orient left-down
$ns duplex-link-op $node_(s4) $node_(r2) orient left-up
```

```
set tcp1 [$ns create-connection TCP/Reno $node_(s1) TCPSink $node_(s3) 0]
$tcp1 set window_ 15
set tcp2 [$ns create-connection TCP/Reno $node_(s2) TCPSink $node_(s3) 1]
$tcp2 set window_ 15
set ftp1 [$tcp1 attach-source FTP]
set ftp2 [$tcp2 attach-source FTP]

# Tracing a queue
set redq [[$ns link $node_(r1) $node_(r2)] queue]
set tchan_ [open all.q w]
$redq trace curq_
$redq trace ave_
$redq attach $tchan_


$ns at 0.0 "$ftp1 start"
$ns at 3.0 "$ftp2 start"
$ns at 10 "finish"

# Define 'finish' procedure (include post-simulation processes)
proc finish {} {
   global tchan_
   set awkCode {
        {
           if ($1 == "Q" && NF>2) {
                print $2, $3 >> "temp.q";
                set end $2
           }
           else if ($1 == "a" && NF>2)
           print $2, $3 >> "temp.a";
        }
   }
   set f [open temp.queue w]
   puts $f "TitleText: red"
   puts $f "Device: Postscript"

   if { [info exists tchan_] } {
        close $tchan_
   }
   exec rm -f temp.q temp.a
   exec touch temp.a temp.q
```

```
    exec awk $awkCode all.q

    puts $f \"queue
    exec cat temp.q >@ $f
    puts $f \n\"ave_queue
    exec cat temp.a >@ $f
    close $f
    exec xgraph -bb -tk -x time -y queue temp.queue &
    exit 0
}

$ns run
```

Κώδικας εξομοίωσης τοπικού δικτύου (LAN) (πηγή http://nile.wpi.edu/NS/**)**




```
set opt(tr)     "out.tr"
set opt(namtr)      "out.nam"
set opt(seed)0
set opt(stop) 5
set opt(node)       8

set opt(qsize)      100
set opt(bw)   10Mb
set opt(delay)      1ms
set opt(ll)     LL
set opt(ifq)    Queue/DropTail
set opt(mac) Mac/Csma/Ca
```

```
set opt(chan)         Channel
set opt(tcp)  TCP/Reno
set opt(sink) TCPSink

set opt(app)  FTP


proc finish {} {
        global ns opt trfd ntrfd

        $ns flush-trace
        close $trfd
      close $ntrfd
        exec nam $opt(namtr) &
        exit 0
}

proc create-trace {} {
        global ns opt

        set trfd [open $opt(tr) w]
        $ns trace-all $trfd
        return $trfd
}

proc create-namtrace {} {
      global ns opt

      set ntrfd [open $opt(namtr) w]
      $ns namtrace-all $ntrfd
}

proc create-topology {} {
        global ns opt
        global lan node source node0

        set num $opt(node)
        for {set i 0} {$i < $num} {incr i} {
                set node($i) [$ns node]
                lappend nodelist $node($i)
        }

        set lan [$ns newLan $nodelist $opt(bw) $opt(delay) \
```

```
                -llType $opt(ll) -ifqType $opt(ifq) \
                -macType $opt(mac) -chanType $opt(chan)]

        set node0 [$ns node]
        $ns duplex-link $node0 $node(0) 2Mb 2ms DropTail

        $ns duplex-link-op $node0 $node(0) orient right

}
```

Κώδικας εξομοίωσης SRM Scalable Reliable Multicast Transport Protocol (πηγή http://nile.wpi.edu/NS/**)**

```
## MAIN ##

set ns [new Simulator]
set trfd [create-trace]
set ntrfd [create-namtrace]

create-topology

set tcp0 [$ns create-connection TCP/Reno $node0 TCPSink $node(7) 0]
$tcp0 set window_ 15

set ftp0 [$tcp0 attach-app FTP]

$ns at 0.0 "$ftp0 start"
$ns at $opt(stop) "finish"

$ns run
```

```
# STAR TOPOLOGY
source /usr/lib/ns-allinone-2.1b4/ns-2/tcl/mcast/srm-nam.tcl ;# to separate $
source /usr/lib/ns-allinone-2.1b4/ns-2/tcl/mcast/srm-debug.tcl ;# to trace del$
Simulator set NumberInterfaces_ 1
set ns [new Simulator]
Simulator set EnableMcast_ 1
$ns trace-all [open out.tr w]
$ns namtrace-all [open out.nam w]
set srmSimType Probabilistic
$ns color 0 red ;#data source
$ns color 40 blue ;#session
$ns color 41 green ;#request
$ns color 42 white ;#repair
$ns color 4 red ;#source node
# Creating The Nodes
set nmax 8
for {set i 0} {$i <= $nmax} {incr i} {
set n($i) [$ns node]
}
$n(1) color "red"
# Creating The Links
for {set i 1} {$i <= $nmax} {incr i} {
$ns duplex-link $n($i) $n(0) 1.5Mb 10ms DropTail
}
# Orienting The Links
$ns duplex-link-op $n(0) $n(1) orient right
$ns duplex-link-op $n(0) $n(2) orient right-up
$ns duplex-link-op $n(0) $n(3) orient up
$ns duplex-link-op $n(0) $n(4) orient left-up
$ns duplex-link-op $n(0) $n(5) orient left
$ns duplex-link-op $n(0) $n(6) orient left-down
$ns duplex-link-op $n(0) $n(7) orient down
$ns duplex-link-op $n(0) $n(8) orient right-down
set group 0x8000
set cmc [$ns mrtproto CtrMcast {}]
$ns at 0.3 "$cmc switch-treetype $group"
# SRM TRACE EVENTS
set srmStats [open srmStats.tr w]
set srmEvents [open srmEvents.tr w]
set fid 0
for {set i 1} {$i <= $nmax} {incr i} {
set srm($i) [new Agent/SRM/$srmSimType]
$srm($i) set dst_ $group
```

```
$srm($i) set fid_ [incr fid]
$srm($i) log $srmStats
$srm($i) trace $srmEvents
$ns at 0.5 "$srm($i) start"
$ns attach-agent $n($i) $srm($i)
}
# Attach a CBR Agent to srm(1)
set packetSize 800
set s [new Application/Traffic/CBR]
$s set packet_size_ $packetSize
$s set interval_ 0.02
$s attach-agent $srm(1)
$srm(1) set tg_ $s
$srm(1) set app_fid_ 0
$srm(1) set packetSize_ $packetSize
$ns at 2.0 "$srm(1) start-source"
set loss_module [new SRMErrorModel]
$loss_module drop-packet 2 10 1
$loss_module drop-target [$ns set nullAgent_]
$ns at 0.75 "$ns lossmodel $loss_module $n(1) $n(0)"
$ns at 4.0 "finish $s" proc distDump interval {
global ns srm
foreach i [array names srm] {
set dist [$srm($i) distances?]
if {$dist != ""} {
puts "[format %7.4f [$ns now]] distances $dist"
}
}
$ns at [expr [$ns now] + $interval] "distDump $interval"
}
proc finish src {
global prog ns env srmStats srmEvents srm nmax
$src stop
$ns flush-trace
close $srmStats
close $srmEvents
puts "converting output to nam format..."
if [info exists env(DISPLAY)] {
puts "running nam..."
exec nam out.nam &
} else {
exec cat srmStats.tr >@stdout
```

```
}
exit 0
}
$ns run

# Author: Jae Chung
# Date:   7/17/99
#
#
#       s1               s3
#        \               /
# 5Mb,3ms \    2Mb,10ms   / 5Mb,3ms
#          r1 --------- r2
# 5Mb,3ms /              \ 5Mb,3ms
#        /               \
#       s2               s4
#

set ns [new Simulator]

#Define different colors for data flows
$ns color 1 Red
$ns color 2 Blue


#Open the nam trace file
set nf [open out.nam w]
set tf [open out.tr w]
$ns namtrace-all $nf
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        #Close the trace file
        close $nf
        close $tf
        #Execute nam on the trace file
        exec nam out.nam &
        exit 0
}
```

```
set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(r1) [$ns node]
set node_(r2) [$ns node]
set node_(s3) [$ns node]
set node_(s4) [$ns node]

$ns duplex-link $node_(s1) $node_(r1) 5Mb 3ms DropTail
$ns duplex-link $node_(s2) $node_(r1) 5Mb 3ms DropTail
$ns duplex-link $node_(r1) $node_(r2) 2Mb 10ms RED
$ns duplex-link $node_(s3) $node_(r2) 5Mb 3ms DropTail
$ns duplex-link $node_(s4) $node_(r2) 5Mb 3ms DropTail

#Setup RED queue parameter
$ns queue-limit $node_(r1) $node_(r2) 20
Queue/RED set thresh_ 5
Queue/RED set maxthresh_ 10
Queue/RED set q_weight_ 0.002
Queue/RED set ave_ 0

$ns duplex-link-op $node_(r1) $node_(r2) queuePos 0.5

$ns duplex-link-op $node_(s1) $node_(r1) orient right-down
$ns duplex-link-op $node_(s2) $node_(r1) orient right-up
$ns duplex-link-op $node_(r1) $node_(r2) orient right
$ns duplex-link-op $node_(s3) $node_(r2) orient left-down
$ns duplex-link-op $node_(s4) $node_(r2) orient left-up


#Setup a MM UDP connection
set udp_s [new Agent/UDP/UDPmm]
set udp_r [new Agent/UDP/UDPmm]
$ns attach-agent $node_(s1) $udp_s
$ns attach-agent $node_(s3) $udp_r
$ns connect $udp_s $udp_r
$udp_s set packetSize_ 1000
$udp_r set packetSize_ 1000
$udp_s set fid_ 1
$udp_r set fid_ 1

#Setup a MM Application
set mmapp_s [new Application/MmApp]
set mmapp_r [new Application/MmApp]
```

```
$mmapp_s attach-agent $udp_s
$mmapp_r attach-agent $udp_r
$mmapp_s set pktsize_ 1000
$mmapp_s set random_ false

#Setup a TCP connection
set tcp [$ns create-connection TCP/Reno $node_(s2) TCPSink $node_(s4) 0]
$tcp set window_ 15
$tcp set fid_ 2

#Setup a FTP Application
set ftp [$tcp attach-source FTP]

#Simulation Scenario
$ns at 0.0 "$ftp start"
$ns at 1.0 "$mmapp_s start"
$ns at 7.0 "finish"

$ns run
```