ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ
ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Μελέτη και Προσομοίωση μηχανισμών υποστήριξης κινητικότητας σε ασύρματα δίκτυα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παύλος Χ.Αντωνίου
Μιχάλης Γ. Καλλίτσης
Παναγιώτης Θ. Καμπανάκης

Επιβλέπων : Ιάκωβος Βενιέρης
Καθηγητής Ε.Μ.Π

Αθήνα, Μάρτιος 2005

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ
ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Μελέτη και Προσομοίωση μηχανισμών υποστήριξης κινητικότητας σε ασύρματα δίκτυα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παύλος Χ. Αντωνίου
Μιχάλης Γ. Καλλίτσης
Παναγιώτης Θ. Καμπανάκης

Επιβλέπων : Ιάκωβος Βενιέρης
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 10η Ιουνίου 2005.

.....	.....	.....
Ι. Βενιέρης	Δ. Κακλαμάνη	Γ. Στασινόπουλος
Καθηγητής Ε.Μ.Π	Αν. Καθηγήτρια Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Μάρτιος 2005

.....

Παύλος Χ. Αντωνίου

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

.....

Μιχάλης Γ. Καλλίτσης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

.....

Παναγιώτης Θ. Καμπανάκης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περιεχόμενα

Πίνακας Πινάκων

Πίνακας Σχημάτων

Περίληψη

Η μεγάλη εξάπλωση των δικτύων ασύρματης πρόσβασης έχει οδηγήσει στην ολοένα και μεγαλύτερη ζήτηση για υπηρεσίες πραγματικού χρόνου (real-time) αλλά και για υπηρεσίες ανεξάρτητες της θέσης του χρήστη την κάθε χρονική στιγμή σε τέτοιου είδους δίκτυα.

Πολλοί μηχανισμοί έχουν προταθεί για τη διαχείριση της κινητικότητας του χρήστη (mobility) με ταυτόχρονη διατήρηση των ενεργών συνδέσεών του, με το πρωτόκολλο Mobile IP να αποτελεί την κλασική μέχρι τώρα λύση. Το γεγονός ότι πρωτόκολλα του ανώτερου υποστρώματος, όπως το TCP, βλέπουν μια σταθερή IP διεύθυνση καθόλη την διάρκεια της κίνησης ενός χρήστη, κάνει το Mobile IP να συνεισφέρει και στην παροχή αδιάλειπτων υπηρεσιών, όπως είναι η μεταφορά αρχείων (FTP) και η πλοήγηση στο διαδίκτυο (WEB browsing).

Ωστόσο, κάποιες εγγενείς αδυναμίες του Mobile IP, όπως η τριγωνική δρομολόγηση και η ενθυλάκωση πακέτου, το κάνουν ακατάλληλο για υποστήριξη εφαρμογών πραγματικού χρόνου μεταξύ κινούμενων χρηστών, όπως το Voice over IP και η ροή video (video streaming). Οι κυριότεροι περιοριστικοί παράγοντες, στους οποίους οφείλεται αυτή η αδυναμία, είναι η καθυστέρηση (delay) παράδοσης πακέτων, η διακύμανση στην καθυστέρηση (jitter) και επιβάρυνση του δικτύου λόγω των ενθυλακωμένων πακέτων.

Αυτοί οι περιορισμοί οδήγησαν στην αναζήτηση νέων τρόπων διαχείρισης κινητικότητας. Έτσι, τα τελευταία χρόνια γίνεται προσπάθεια διαχείρισης της μέσω του στρώματος εφαρμογής (application layer), με χρήση του πρωτοκόλλου SIP (Session Initiation Protocol). Το SIP είναι ένα πρωτόκολλο σηματοδοσίας, το οποίο με κατάλληλες ανταλλαγές μηνυμάτων μπορεί να καταστήσει διαφανή την κινητικότητα σε επίπεδο εφαρμογής.

Σκοπός της διπλωματικής εργασίας είναι η μελέτη των παραπάνω μηχανισμών υποστήριξης κινητικότητας καθώς και η ανάπτυξη μοντέλου για την προσομοίωση και σύγκρισή τους. Το μοντέλο αναπτύχθηκε ως επέκταση του προγράμματος προσομοίωσης Network Simulator 2, ενός από τα πλέον χρησιμοποιούμενα προγράμματα προσομοίωσης ανοικτού κώδικα στο χώρο των δικτύων.

Λέξεις Κλειδιά

Πρωτόκολλο Έναρξης Συνόδου (SIP), Κινητό IP, τριγωνική δρομολόγηση, προσωπική κινητικότητα, κινητικότητα IP, Network Simulator 2 (ns-2), ασύρματα δίκτυα, πράκτορας χρήστη, πληρεξούσιος εξυπηρετητής, εξυπηρετητής εγγραφών.

Abstract

Lately, there has been a great boost in the area of wireless networks that has led to an inceasing demand on real time services and services independent from the location of the mobile user whilst he is moving, in such networks.

There are a bunch of mechanisms that have been developped in order to provide mobility support along with preserving all active connections in wireless networks. The common one until now has been Mobile IP protocol. This is because higher layers' protocols, such as TCP, act as if the IP address of the moving user remains the same while moving, as in Mobile IP, which contributes to providing continuous, unceasing services like FTP or WEB browsing.

On the other hand, there are a few drawbacks that make Mobile IP inappropriate for real time applications like Voice over IP or video streaming. These drawbacks are mainly the routing triangle and IP in IP encapsulation. The factors that are influenced by Mobile IP's disadvantages are packet delay, jitter and header overhead because of IP in IP encapsulation.

Thus, reasearch in the area of mobility support has led to the use of an Application Layer protocol, called SIP (Session Initialization Protocol) which overcomes the flaws of Mobile IP. SIP is a signalling protocol which by exchanging the appropriate messages can provide mobility awareness at application layer.

Our diploma thesis aims to focus on the above mobility support mechanisms and also to develop a model to simulate and compare them. The model will be developped using Network Simulator 2, one of the mostly well-known simulation open-source software in the networks field.

Keywords

Session Initiation Protocol (SIP), Mobile IP, triangular routing, personal mobility, IP mobility, Network Simulator 2 (ns-2), wireless networks, User Agent, Proxy Server, Registrar.

Εισαγωγή

Ζούμε σε μια εποχή όπου η ανάγκη του ανθρώπου για συνεχή και άμεση πληροφόρηση γίνεται ολοένα και πιο επιτακτική. Η επιθυμία του για ενημέρωση και επικοινωνία οπουδήποτε κι αν βρίσκεται, οποιαδήποτε χρονική στιγμή κι αν το θελήσει, φαίνεται να εκπληρώνεται με την αλματώδη ανάπτυξη των προσωπικών κινητών συσκευών, οι οποίες του δίνουν τη δυνατότητα να συνδιαλέγεται, να ενημερώνεται και να κλείνει συμφωνίες ανά πάσα στιγμή, ακόμα κι αν βρίσκεται εν κινήσει.

Αυτό δε θα μπορούσε να καταστεί εφικτό χωρίς την ύπαρξη μηχανισμών οι οποίοι παρέχουν την υποδομή για τη διατήρηση των κλήσεων χωρίς τη διακοπή τους κατά τη διάρκεια της κίνησης ενός χρήστη μέσα σε περιβάλλοντα όπου παρεμβάλλονται διάφορα ασύρματα δίκτυα. Οι δύο κυριότεροι μηχανισμοί υλοποίησης της κινητικότητας αυτή τη στιγμή, είναι το Mobile IP το οποίο είναι ένας μηχανισμός επιπέδου 3 (επίπεδο δικτύου) και το Πρωτόκολλο Έναρξης Συνόδου (Session Initiation Protocol) το οποίο αποτελεί ένα πρωτόκολλο του στρώματος εφαρμογής.

Στη παρούσα διπλωματική εργασία, αντικειμενικός μας στόχος ήταν να μελετήσουμε, να προσομοιώσουμε και να συγκρίνουμε τους δύο προαναφερθέντες μηχανισμούς υποστήριξης κινητικότητας σε ασύρματα δίκτυα. Η προσομοίωση των δύο μηχανισμών έγινε στο ανοικτού κώδικα περιβάλλον εξομοίωσης Network Simulator v2.27 το οποίο υποστήριζε μόνο το Mobile IP, οπότε χρειάστηκε να υλοποιήσουμε το Πρωτόκολλο Έναρξης Συνόδου (SIP).

Παρακάτω γίνεται αρχικά μια εκτενής αναφορά τόσο στο Mobile IP όσο και στο SIP πάνω σε θέματα που άπτονται κυρίως της κινητικότητας καθώς και άλλων συνιστωσών, όπως ανάλυση των προβλημάτων που αντιμετωπίζει ο κάθε μηχανισμός και οι κυριότερες παράμετροι που εμπλέκονται στα προβλήματα αυτά. Ακολουθεί η σχεδίαση και η ανάλυση του κώδικα υλοποίησης του Πρωτοκόλλου Έναρξης Συνόδου στο περιβάλλον εξομοίωσης Network Simulator v2.27. Στη συνέχεια έχουμε την περιγραφή των διαφόρων σεναρίων που υλοποιήθηκαν στο NS με σκοπό τη σύγκριση Mobile IP –

SIP καθώς και των αποτελεσμάτων που προέκυψαν από την εκτέλεσή τους. Καταλήγουμε με την αποτίμηση των αποτελεσμάτων και την εξαγωγή των κυριοτέρων συμπερασμάτων.

Κλείνοντας την σύντομη εισαγωγή, θα θέλαμε να ευχαριστήσουμε θερμά τον σύμβουλό μας Καθηγητή Ιάκωβο Βενιέρη αλλά και τους υποψήφιους διδάκτορες Βαγγέλη Νίκα, Γιώργο Λιουδάκη και Νίκο Δέλλα για την πολύτιμη βοήθεια και συμπαράσταση που μας έδειχναν κατά τη διάρκεια εκπόνησης αυτής της διπλωματικής εργασίας. Πιστεύουμε ότι ένα ιδιαίτερο ευχαριστώ αξίζει ο Νίκος για την υπομονή που υπέδειξε στην απάντηση απορίων σχετιζόμενες με το Network Simulator (ns-2).

1. Κινητικότητα με χρήση Mobile IP

Η κινητικότητα IP, όπως προτείνεται από το Mobile IP, χρησιμοποιεί την τεχνική της σήραγγας (tunneling) πάνω στα IP πακέτα από τον Πράκτορα Οικείων (Home Agent) στον Πράκτορα Επισκεπτών (Foreign Agent). Με αυτόν τον τρόπο επιτυγχάνεται η διαφάνεια της κινητικότητας στα ανώτερα επίπεδα. Παρόλα αυτά, η τεχνική του Mobile IP παρουσιάζει το πρόβλημα της τριγωνικής δρομολόγησης (triangular routing). Έχουν παρουσιαστεί κάποιες ιδέες για βελτίωση του Mobile IP ώστε να ξεπεραστεί το πρόβλημα αυτό ωστόσο δεν λείπουν ούτε από αυτές τα μειονεκτήματα.

1.1 Γενικά

Κύριο χαρακτηριστικό του Mobile IP είναι ότι η IP διεύθυνση παραμένει σταθερή καθώς ένας κινούμενος χρήστης περνά από διαδοχικά υποδίκτυα. Αυτό έχει σαν συνέπεια τα πρωτόκολλα υψηλότερων επιπέδων να βλέπουν μια σταθερή IP διεύθυνση. Έτσι συνδέσεις τύπου TCP μπορούν να διατηρούνται ενεργές σε όλη την διάρκεια της κίνησης. Σε ενδεχόμενες αλλαγές διευθύνσεων IP οι συνδέσεις TCP θα έσπαζαν αφού μια τέτοια σύνδεση προϋποθέτει σταθερές διευθύνσεις πηγής και προορισμού καθώς και αριθμούς θυρών. Αυτό που επιτυγχάνεται στο Mobile IP είναι η δυνατότητα της επικοινωνίας με τον κινούμενο χρήστη μέσω μιας αμετάβλητης IP διεύθυνσης και συγκεκριμένα με τη διεύθυνση που του έχει ανατεθεί από το οικείο του δίκτυο. Ο Πράκτορας Οικείων αναλαμβάνει να στέλνει τα πακέτα που προορίζονται για τον κινούμενο host ενθυλακώνοντάς (encapsulate) τα και προωθώντας τα στον τρέχοντα Πράκτορα Επισκεπτών του κινούμενου χρήστη. Όπως καταλαβαίνουμε, το Mobile IP υποφέρει από το πρόβλημα της τριγωνικής δρομολόγησης· πακέτα που κατευθύνονται στον κινούμενο χρήστη πρέπει να περάσουν αναγκαστικά από τον Πράκτορα Οικείων ενώ τα πακέτα που στέλνει ο κινούμενος χρήστης πάνε κατευθείαν στον προορισμό.

1.2 Περιγραφή Mobile IP

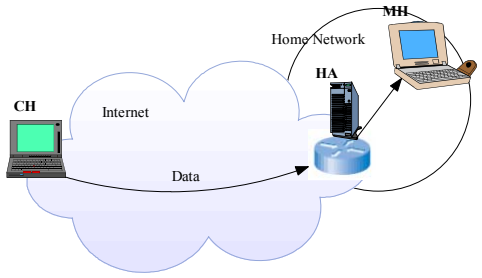
Για την πιο κάτω σύντομη περιγραφή του Mobile IP θα χρησιμοποιήσουμε τις πιο κάτω οντότητες:

- Κινούμενος χρήστης (MH) : Ο χρήστης που κινείται ανάμεσα σε διαφορετικά υποδίκτυα.
- Καλών χρήστης (CH) : Ο χρήστης στο άλλο άκρο της επικοινωνίας.
- Πράκτορας Οικείων (HA) : Δρομολογητής στο οικείο δίκτυο του κινούμενου χρήστη ο οποίος αναλαμβάνει να ενθυλακώνει και να στέλνει με την μέθοδο της σήραγγας τα πακέτα προς τον κινούμενο host.
- Πράκτορας Επισκεπτών (FA) : Δρομολογητής στο δίκτυο που επισκέπτεται ο κινούμενος host που αναλαμβάνει να του παραδίδει τα πακέτα, αφού πρώτα τα αποφλοιώσει (decapsulate).

Η μια περίπτωση που θα παρουσιάσουμε είναι όταν ο κινούμενος host είναι ακόμα στο οικείο του δίκτυο και η δεύτερη όταν ο κινούμενος host βρίσκεται σε ένα ξένο δίκτυο.

1.2.1 Κινούμενος χρήστης σε οικείο δίκτυο

Στο σενάριο αυτό, που αποτελεί και την πιο απλή περίπτωση, ο κινούμενος χρήστης δέχεται τα δεδομένα κατευθείαν από τον καλούντα χρήστη με τη μεσολάβηση του Πράκτορα Οικείων που αποτελεί τον δρομολογητή του οικείου δικτύου του χρήστη. Η διαδικασία φαίνεται στο σχήμα 1.

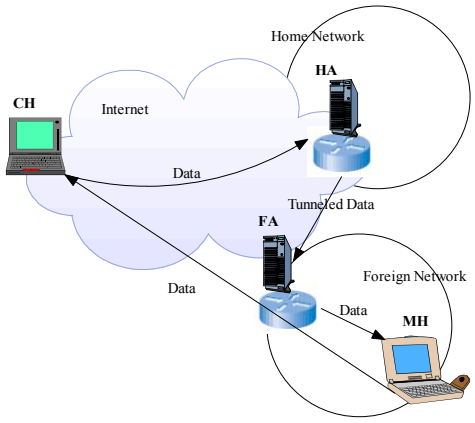


Σχήμα 1: Χρήστης στο οικείο του Δίκτυο.

1.2.2 Κινούμενος χρήστης σε ξένο δίκτυο

Όταν ένας host μετακινηθεί σε ένα άλλο υποδίκτυο, μαθαίνει τον τοπικό Πράκτορα Επισκεπτών και στέλνει ένα μήνυμα εγγραφής (registration message) στον Πράκτορα

Οικείων του για να τον ενημερώσει για την IP διεύθυνση του τοπικού Πράκτορα Επισκεπτών. Παρά το γεγονός ότι στον κινούμενο host έχει ανατεθεί μια καινούργια IP διεύθυνση (λόγω του ότι τώρα ανήκει σε άλλο υποδίκτυο), συνεχίζει να είναι γνωστός με την αρχική του IP διεύθυνση. Έτσι από εκείνη τη στιγμή και πέρα, όταν κάποιος στείλει πακέτα με προορισμό τον κινούμενο χρήστη, θα κατευθυνθούν και πάλι προς τον Πράκτορα Οικείων του, ο οποίος θα αναλάβει να ενθυλακώσει και να στείλει με την τεχνική της σήραγγας τα πακέτα προς τον εκάστοτε Πράκτορα Επισκεπτών του κινούμενου χρήστη. Αυτός θα προωθήσει τα πακέτα στον προορισμό τους, έχοντας πρώτα «ξεφλουδίσει» το πακέτο με την επιβαρύνουσα IP επικεφαλίδα που είχε προσθέσει ο Πράκτορας Οικείων (βλέπε σχήμα2).



Σχήμα 2: Χρήστης σε ξένο Δίκτυο.

1.2.3 Μειονεκτήματα Mobile IP

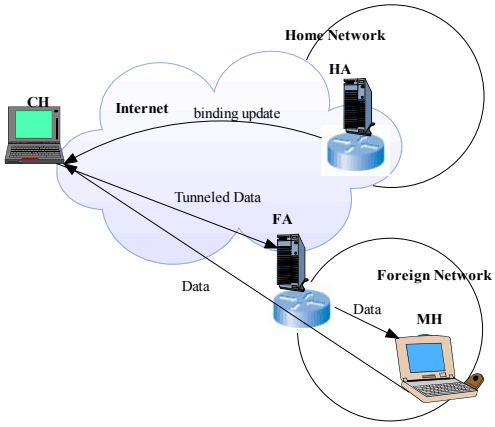
Το χαρακτηριστικό τρίγωνο που δημιουργείται αυξάνει κατά πολύ την καθυστέρηση (latency) του χρόνου παράδοσης των πακέτων στον κινούμενο χρήστη. Αυτό αποτελεί και το κυριότερο μειονέκτημα του Mobile IP. Η επιπρόσθετη καθυστέρηση που παρατηρείται ισούται με το άθροισμα του χρόνου μετάδοσης και διάδοσης του πακέτου από τον καλούντα χρήστη προς τον Πράκτορα Οικείων, του χρόνου που χρειάζεται να γίνει η ενθυλάκωση του πακέτου, του χρόνου μετάδοσης και διάδοσης του πακέτου από τον Πράκτορα Οικείων στον Πράκτορα Επισκεπτών και του χρόνου που παίρνει η αποφλοίωση του πακέτου μείον τον χρόνο μετάδοσης και διάδοσης που θα έκανε το

πακέτο αν πήγαινε κατευθείαν από τον καλούντα χρήστη στον Πράκτορα Επισκεπτών. Όπως γίνεται αντιληπτό ο χρόνος καθυστέρησης γίνεται όλο και μεγαλύτερος όσο αυξάνεται η απόσταση μεταξύ των διαφόρων οντοτήτων. Τέτοιοι χρόνοι είναι ανεπίτρεπτοι για εφαρμογές ευαίσθητες σε καθυστέρηση, όπως Voice over IP (VoIP) και video streaming κάτι που καθιστά το Mobile IP ακατάλληλο για τέτοιες κινήσεις.

Ένα άλλο σημαντικό μειονέκτημα είναι η επιβάρυνση των 20 bytes που δημιουργείται σε κάθε πακέτο που ενθυλακώνεται. Αν αναλογιστεί κανείς το γεγονός ότι το μέγεθος των πακέτων που χρησιμοποιούνται για μετάδοση φωνής είναι πολύ μικρά (αφού πρόκειται συνήθως για πακέτα UDP) καθώς και το γεγονός ότι η μετάδοση γίνεται με ένα ρυθμό της τάξης των 16Kbps καταλαβαίνει πόσο μεγάλο είναι το ποσοστό επιβάρυνσης του δικτύου.

1.2.4 Βελτιώσεις στο Mobile IP

Μια βελτίωση που προτάθηκε για το Mobile IP και είναι γνωστή σαν route optimization λύνει το πρόβλημα της τριγωνικής δρομολόγησης. Για να το πετύχει αυτό ο καλών χρήστης ενημερώνεται από τον Πράκτορα Οικείων του κινούμενου χρήστη για την τρέχουσα IP διεύθυνση του κινούμενου. Η ενημέρωση γίνεται κάθε φορά που ο κινούμενος host μπαίνει σε διαφορετικό υποδίκτυο. Με αυτό τον τρόπο ο καλών χρήστης γνωρίζει πάντα που βρίσκεται ο κινούμενος χρήστης και αναλαμβάνει αυτός να ενθυλακώσει τα πακέτα και να τα στείλει με την μέθοδο της σήραγγας στον Πράκτορα Επισκεπτών του κινούμενου (βλέπε σχήμα3).



Σχήμα 3: Mobile IP και χρήση route optimization.

Ωστόσο, παρόλο που αυτή η μέθοδος μειώνει την καθυστέρηση επικοινωνίας, δεν αποτελεί την ιδανική λύση. Η επιβάρυνση πακέτων (packet overhead) που δημιουργείται με την ενθυλάκωση IP σε IP (επιπρόσθετα 20 bytes για κάθε πακέτο) παραμένει ένα σημαντικό μειονέκτημα. Επιπρόσθετα, οι Πράκτορες Οικείων και Επισκεπτών δύναται να αποτελέσουν την στενωπό (bottleneck) του δικτύου σε περιπτώσεις που πρέπει να χειριστούν σήραγγες για να εξυπηρετήσουν πολλούς κινούμενους χρήστες. Επίσης, κάποια επιπρόσθετα χαρακτηριστικά που πρέπει να υποστηρίζουν οι καλούντες host, όπως οι συνεχείς αλλαγές στην IP τους στοίβα (για να μπορεί να κάνει την ενθυλάκωση), η παρακολούθηση σε συγκεκριμμένη θύρα για να πάρει τυχόν ενημερώσεις αλλά και κάποιες άλλες λειτουργίες κάνουν δύσκολη την ευρεία εφαρμογή αυτής της μεθόδου στην πράξη.

2. Κινητικότητα με χρήση SIP

Το SIP είναι ένα πρωτόκολλο σηματοδοσίας που χρησιμοποιείται για την εγκατάσταση και τον τερματισμό συνόδων (sessions). Συνήθως χρησιμοποιείται για συνδέσεις πολυμέσων (π.χ. video streaming) και για συνδέσεις με έναν ή περισσότερους χρήστες**.** Είναι ένα πρωτόκολλο βασισμένο σε κείμενο (όπως το HTTP) και στο μοντέλο πελάτη-εξυπηρετητή (client-server).

Το SIP υποστηρίζει την προσωπική κινητικότητα (personal mobility), δηλαδή τη δυνατότητα που δίνεται σε ένα χρήστη να εντοπίζεται ανεξαρτήτως της τοποθεσίας που βρίσκεται και της συσκευής που χρησιμοποιεί. Η προσωπική κινητικότητα βασίζεται στην χρήση μιας μοναδικής και σταθερής προσωπικής ταυτότητας (constant identifier). Το SIP URI έχει ακριβώς αυτήν την ιδιότητα. Οι SIP URI διευθύνσεις που ανατίθενται στους χρήστες SIP είναι διευθύνσεις τύπου e-mail που έχουν την μορφή "user@host", όπου user είναι το όνομα του χρήστη ή κάποιος αριθμός τηλεφώνου και host είναι η διεύθυνση ή το όνομα του υποδικτύου. Με αυτόν τον τρόπο οι χρήστες αναγνωρίζονται μοναδικά κατά αναλογία με αυτά που ξέρουμε με τις διευθύνσεις e-mail. Το SIP υποστηρίζει επίσης την κινητικότητα υπηρεσίας δηλαδή την ικανότητα να παρέχει στο χρήστη τις ίδιες και αδιάλειπτες υπηρεσίες όταν αυτός κινείται σε διαφορετικά και ξένα περιβάλλοντα καθώς και την κινητικότητα συνόδου, την ικανότητα δηλαδή να διατηρούνται οι ενεργοποιημένες συνόδοι ενός χρήστη όταν αυτός αλλάζει τερματικό (π.χ. μεταφορά ενεργοποιημένης συνόδου από ένα υπολογιστή σε ένα κινητό τερματικό παλάμης).

Στο πρώτο μέρος της ενότητας αυτής θα παρουσιαστεί αρκετά αναλυτικά το πρωτόκολλο SIP και στη συνέχεια θα περιγράψουμε πως επιτυγχάνεται η κινητικότητα χρηστών με τη βοήθεια μηνυμάτων SIP.

2.1 Το Πρωτόκολλο Έναρξης Συνόδου (SIP – Session Initiation Protocol)

Στην τρέχουσα υποενότητα θα γίνει μια αναλυτική αναφορά στις οντότητες του SIP, στα μηνύματα που ανταλλάζονται μεταξύ αυτών των οντοτήτων και στα πεδία επικεφαλίδας αυτών των μηνυμάτων. Σκοπός μας είναι η εξοικείωση του αναγνώστη με αυτές τις έννοιες ώστε να του δοθεί η δυνατότητα της πλήρους κατανόησης των παραδειγμάτων και των σεναρίων που θα ακολουθήσουν.

2.1.1 Οντότητες του Πρωτοκόλλου Έναρξης Συνόδου (SIP Entities)

Σε ένα δίκτυο σηματοδοσίας που υλοποιεί το Πρωτόκολλο Έναρξης Συνόδου (SIP network) μπορούν να υπάρχουν 4 διαφορετικές λογικές οντότητες. Κάθε οντότητα έχει συγκεκριμένες λειτουργίες και συμμετέχει σε μια αμφίδρομη επικοινωνία σαν εξυπηρετητής (server) ο οποίος απαντά σε αιτήσεις, σαν πελάτης (client) ο οποίος αρχικοποιεί αιτήσεις ή ακόμα μπορεί να εκτελεί και τις δύο λειτουργίες. Μια ή περισσότερες λογικές οντότητες μπορούν να αποτελούν μια φυσική οντότητα η οποία παρέχει τη λειτουργικότητα της κάθε λογικής οντότητας. Παρατίθενται παρακάτω οι 4 λογικές οντότητες ενός δικτύου SIP.

Πράκτορας Χρήστη (User Agent)

Ο πράκτορας χρήστη αποτελεί μια τερματική οντότητα σε ένα δίκτυο SIP. Ένας πράκτορας χρήστη μπορεί να αρχικοποιήσει και να τερματίσει συνόδους ανταλλάζοντας αιτήσεις και αποκρίσεις μεταξύ άλλων οντοτήτων του δικτύου. Όπως υποδηλώνει άλλωστε και το όνομά του, ένας πράκτορας χρήστη καθοδηγούμενος από τις εντολές που δέχεται από το χρήστη, ενεργεί εκ μέρους του εγκαθιδρύοντας ή τερματίζοντας συνόδους για τη μετάδοση κλήσεων πολυμέσων. Στις περισσότερες περιπτώσεις ο χρήστης είναι ένας άνθρωπος όμως υπάρχουν και περιπτώσεις όπου ο χρήστης είναι ένα πρωτόκολλο υψηλότερου επιπέδου.

Ένας πράκτορας χρήστη υλοποιεί λειτουργίες πελάτη (User Agent Client) καθώς και λειτουργίες εξυπηρετητή (User Agent Server). Ο πράκτορας χρήστη – πελάτης (UAC) αρχικοποιεί αιτήσεις ενώ από την άλλη ο πράκτορας χρήστη – εξυπηρετητής (UAS)

απαντά σε αιτήσεις παράγοντας τις κατάλληλες αποκρίσεις. Κατά τη διάρκεια μιας συνόδου, ο πράκτορας χρήστη συνήθως λειτουργεί τόσο σαν εξυπηρετητής όσο και σαν πελάτης.

Μερικές από τις συσκευές που υλοποιούν πράκτορα χρήστη σε δίκτυα σηματοδοσίας που υλοποιούν το SIP είναι IP-τηλέφωνα (υπηρεσία VoIP), πύλες διασύνδεσης του δικτύου IP με το δίκτυο τηλεφωνίας (telephony gateways), σταθμοί εργασίας κ.τ.λ..

Πληρεξούσιος Εξυπηρετητής (Proxy Server)

Ένας πληρεξούσιος εξυπηρετητής είναι μια ενδιάμεση οντότητα που υλοποιείται στο δίκτυο σηματοδοσίας δρώντας τόσο σαν εξυπηρετητής όσο και σαν πελάτης με σκοπό να πραγματοποιήσει αιτήσεις εκ μέρους άλλων πελατών που βρίσκονται στα όρια της περιοχής κάλυψής του.

Ένας πληρεξούσιος εξυπηρετητής, δέχεται αιτήσεις τόσο από άλλους εξυπηρετητές ή από πράκτορες χρήστη και δρα εκ μέρους τους προωθώντας τις αιτήσεις τους σε άλλους ομότιμους εξυπηρετητές ή απαντώντας σε αυτές. Ένας τέτοιος εξυπηρετητής ερμηνεύει και αν είναι αναγκαίο, αναδημιουργεί το μήνυμα αίτησης προτού το προωθήσει. Πρέπει να ληφθεί επίσης υπόψη ότι ένας πληρεξούσιος εξυπηρετητής δε χρειάζεται να καταλαβαίνει το μήνυμα αίτησης για να προωθήσει την αίτηση.

Τυπικά ο εξυπηρετητής έχει πρόσβαση σε μια τοπική βάση δεδομένων με σκοπό να ανακτήσει τον επόμενο κόμβο (next hop) στον οποίο θα προωθήσει κάποια εισερχόμενη αίτηση. Η διεπαφή του εξυπηρετητή με τη βάση δεδομένων δεν καθορίζεται από το Πρωτόκολλο Έναρξης Συνόδου. Ο πληρεξούσιος εξυπηρετητής δεν έχει δικαιοδοσία να τροποποιήσει οποιαδήποτε εγγραφή που βρίσκεται μέσα στη βάση δεδομένων αλλά μπορεί να ανακτήσει οποιαδήποτε διαθέσιμη πληροφορία. Οι βάσεις δεδομένων συνήθως περιέχουν εγγραφές (SIP registrations), πληροφορία παρουσίας (presence information) και πληροφορία για το που βρίσκεται ο προς αναζήτηση χρήστης. Οι καταχωρήσεις αυτές γίνονται από τον πράκτορα – καταχωρητή (Registrar Proxy) ο οποίος παρουσιάζεται πιο κάτω, καθώς ένας κινητός χρήστης εισέρχεται σε μια νέα περιοχή.

Ένας πληρεξούσιος εξυπηρετητής διαφέρει από έναν πράκτορα χρήστη σε κάποια βασικά σημεία όπως:

- Δε παράγει αιτήσεις, απλά αποκρίνεται σε αιτήσεις του πράκτορα χρήστη, ή τις προωθεί εκ μέρους του.
- Δεν επεξεργάζεται το κύριο μέρος ενός μηνύματος (message body) αλλά βασίζει τη προώθηση ενός μηνύματος αποκλειστικά στην επικεφαλίδα του.

Εξυπηρετητής Ανακατεύθυνσης (Redirect Server)

Ένας εξυπηρετητής ανακατεύθυνσης δέχεται αιτήσεις από πράκτορες χρήστη και αποκρίνεται πίσω σε αυτούς χωρίς να προωθεί περαιτέρω τις αιτήσεις όπως θα έκανε ένας πληρεξούσιος εξυπηρετητής. Αυτή είναι και η βασική διαφορά των δύο προαναφερθεισών εξυπηρετητών. Όταν ένας εξυπηρετητής ανακατεύθυνσης λάβει μια αίτηση από ένα πράκτορα χρήστη, χρησιμοποιεί μια βάση δεδομένων για να ανακτήσει την πληροφορία για το που βρίσκεται ο πράκτορα χρήστη προορισμού δηλαδή πιθανές IP διευθύνσεις του. Η πληροφορία αυτή αποστέλνεται πίσω στον πράκτορα χρήστη που είχε αποστείλει την αίτηση.

Εξυπηρετητής Εγγραφών (Registration Server ή Registrar)

Ένας εξυπηρετητής εγγραφών δέχεται μόνο αιτήσεις εγγραφής (μηνύματα REGISTER) απορρίπτοντας οποιαδήποτε άλλα μηνύματα που μπορεί πιθανώς να λάβει (σε αυτά απαντά με το μήνυμα 501 Not Implemented). Καθώς ένας κινητός χρήστης αλλάζει περιοχή αποκτά μια νέα IP διεύθυνση και πρέπει να εγγραφεί στη τοπική βάση δεδομένων καθώς και στην οικεία βάση δεδομένων του, έτσι ώστε να μπορεί να τον εντοπίσει οποιοσδήποτε επιθυμεί να επικοινωνήσει μαζί του. Αυτό γίνεται εφικτό αποστέλλοντας κατάλληλα μηνύματα REGISTER στους δύο αντίστοιχους εξυπηρετητές εγγραφών, οι οποίοι ενημερώνουν κατάλληλα τις αντίστοιχες βάσεις δεδομένων των περιοχών στις οποίες ανήκουν. Ο εξυπηρετητής αυτός αποτελεί τη μοναδική οντότητα που μπορεί να τροποποιεί και να ενημερώνει τις εγγραφές στις βάσεις δεδομένων.

2.1.2 Μηνύματα του Πρωτοκόλλου Έναρξης Συνόδου (SIP Messages)

Υλοποιούνται δύο ήδη μηνυμάτων στο Πρωτόκολλο Έναρξης Συνόδου:

- Μηνύματα αίτησης – αποστέλλονται από τον πελάτη προς τον εξυπηρετητή
- Μηνύματα απόκρισης – αποστέλλονται από τον εξυπηρετητή προς το πελάτη

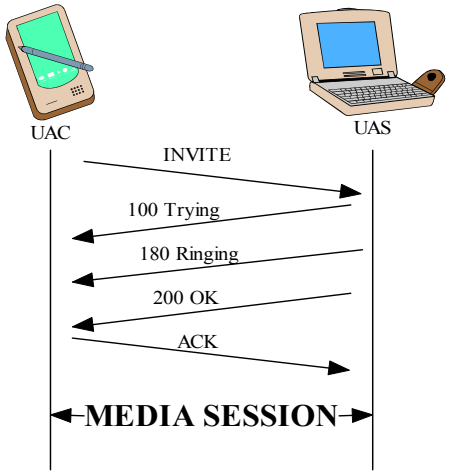
Μηνύματα Αίτησης

Υπάρχουν 13 διαφορετικά μηνύματα αίτησης (6 εξ αυτών περιγράφονται στις προδιαγραφές του πρωτοκόλλου στο RFC 3261 και τα υπόλοιπα 7 σε ξεχωριστά RFCs). Τα μηνύματα INVITE, REGISTER, BYE, ACK, CANCEL και OPTIONS αποτελούν τις αρχικές μεθόδους που είχαν υλοποιηθεί στο SIP. Τα υπόλοιπα μηνύματα REFER, SUBSCRIBE, NOTIFY, MESSAGE, UPDATE, INFO και PRACK περιγράφονται σε ξεχωριστά RFCs. Ο σκοπός της διπλωματικής αυτής είναι η μελέτη μηχανισμών διαχείρισης κινητικότητας σε ασύρματα δίκτυα και κυρίως η υλοποίηση των μηχανισμών αυτών στο περιβάλλον εξομοίωσης Network Simulator πάνω από το Πρωτόκολλο Έναρξης Συνόδου. Έτσι υλοποιήσαμε μόνο 5 από τα 13 μηνύματα αίτησης, αυτά που εμπλέκονται στην κινητικότητα και τα οποία περιγράφονται αναλυτικά παρακάτω.

Μήνυμα INVITE

Το μήνυμα INVITE χρησιμοποιείται για την έναρξη συνόδων μεταξύ δύο πρακτόρων χρήστη. Στο μήνυμα INVITE που αποστέλνεται από κάποιο πράκτορα χρήστη περιέχονται πληροφορίες σχετικά με τη ποιότητα υπηρεσίας της ζεύξης που πρόκειται να εγκατασταθεί (Quality of Service), πληροφορίες ασφαλείας (security information) καθώς και πληροφορίες που αφορούν τα μέσα του καλούντα (media information of the caller). Μια σύνοδος αρχικοποιείται μόνο όταν ο πράκτορας χρήστη του καλούντα λάβει – από τον ομότιμο πράκτορα χρήστη στον οποίο έστειλε το μήνυμα INVITE – το μήνυμα 200 OK και αποστείλει σε αυτόν το μήνυμα ACK. Κατά τη διάρκεια της επικοινωνίας αυτής, ο πράκτορας χρήστη του καλούντα – που αποστέλλει το μήνυμα INVITE – ενεργεί σαν πελάτης (User Agent Client) και ο πράκτορας χρήστη του καλούμενου σαν εξυπηρετητής (User Agent Server). Μια επιτυχημένη αίτηση INVITE εγκαθιστά ένα διάλογο μεταξύ δύο πρακτόρων χρήστη, ο οποίος τερματίζεται με την αποστολή ενός μηνύματος BYE από οποιαδήποτε εκ των δύο πλευρών.

Ένα παράδειγμα ροής μηνυμάτων που αφορούν στο INVITE παρουσιάζεται στο σχήμα 4.



Σχήμα 4: Μήνυμα INVITE.

Ένας UAC που αρχικοποιεί ένα μήνυμα INVITE, δημιουργεί ένα μοναδικό Call-ID το οποίο θα χρησιμοποιείται καθ' όλη τη διάρκεια της κλήσης και αρχικοποιεί το πεδίο CSeq το οποίο θα αυξάνεται για κάθε νέα αίτηση που ανήκει στο ίδιο Call-ID (στην ίδια σύνοδο). Στα πεδία της επικεφαλίδας To και From του εν λόγω μηνύματος περιέχονται οι μοναδικές διευθύνσεις SIP (SIP URIs) των πρακτόρων χρήστη του καλούμενου και του καλούντα αντίστοιχα και στο πεδίο Contact βρίσκεται η διεύθυνση του UAC έτσι ώστε να μπορεί να επικοινωνήσει μαζί του ο UAS. Ο συνδυασμός των πεδίων To, From και Call-ID αποτελούν ένα μοναδικό αναγνωριστικό για κάθε διάλογο.

Εκτός από την περίπτωση Έναρξης μιας συνόδου, ένα μήνυμα INVITE αποστέλνεται στο μέσο μιας κλήσης (mid-call) μεταξύ δύο πρακτόρων χρήστη από τον πράκτορα χρήστη ενός κινούμενου χρήστη ο οποίος αλλάζει περιοχή. Το μήνυμα αυτό περιέχει το ίδιο Call-ID όπως και το αρχικό INVITE που χρησιμοποιήθηκε για την έναρξη της συνόδου αυτής. Με το μήνυμα αυτό, ο αποστολέας του μηνύματος επιθυμεί να ενημερώσει τον παραλήπτη για τη νέα του διεύθυνση την οποία τοποθετεί στο πεδίο Contact.

Τα υποχρεωτικά πεδία του μηνύματος INVITE φαίνονται στον πίνακα 1.

Υποχρεωτικά πεδία του μηνύματος INVITE
Call-ID
CSeq
From
To
Via
Contact
Max-Forwards

Πίνακας 1: Υποχρεωτικά πεδία του μηνύματος INVITE.

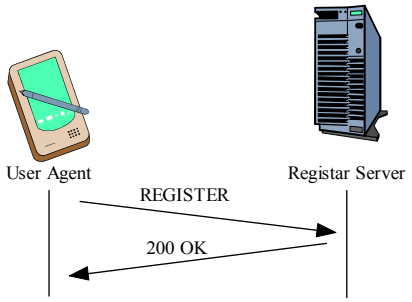
Μήνυμα REGISTER

Το μήνυμα REGISTER αποστέλνεται από ένα πράκτορα χρήστη με σκοπό να κοινοποιήσει στο δίκτυο SIP τη παρούσα διεύθυνση του την οποία τοποθετεί στο πεδίο Contact του μηνύματος. Με το τρόπο αυτό, όλες οι αιτήσεις που απευθύνονται σε αυτόν (στο δικό του SIP URI) θα στέλνονται σε αυτή τη νέα διεύθυνση. Τα μηνύματα REGISTER κατευθύνονται προς τον εξυπηρετητή εγγραφών (Registrar) ο οποίος ενημερώνει τη βάση δεδομένων της περιοχής του. Η εγγραφή του πράκτορα χρήστη δεν απαιτείται για να του επιτραπεί να χρησιμοποιήσει έναν τοπικό πληρεξούσιο εξυπηρετητή για εξερχόμενες κλήσεις αλλά είναι αναγκαία έτσι ώστε να δέχεται εισερχόμενες κλήσεις από κάποιο τοπικό πληρεξούσιο εξυπηρετητή.

Όπως και στο μήνυμα INVITE έτσι και στο REGISTER, το πεδίο CSeq αυξάνεται για κάθε αίτηση REGISTER αλλά η χρήση των πεδίων To, From και Call-ID διαφέρει ελάχιστα από άλλες αιτήσεις. Το πεδίο To περιέχει το SIP URI του πράκτορα χρήστη ο οποίος θα εγγραφεί και το πεδίο From περιέχει το SIP URI του αποστολέα της αίτησης το οποίο συνήθως είναι το ίδιο με εκείνο του πράκτορα χρήστη που πρόκειται να εγγραφεί. Διαφέρει στη περίπτωση που ένας πράκτορας χρήστη εγγράφεται από κάποιο

άλλο πράκτορα χρήστη. Στη περίπτωση αυτή το πεδίο From περιέχει το SIP URI του πράκτορα που κάνει την εγγραφή.

Ένα παράδειγμα ροής μηνυμάτων που αφορούν στο REGISTER φαίνεται στο σχήμα 5.



Σχήμα 5: Μήνυμα REGISTER.

Τα υποχρεωτικά πεδία του μηνύματος REGISTER φαίνονται στο πιο κάτω πίνακα 2.

Υποχρεωτικά πεδία του μηνύματος REGISTER
Call-ID
CSeq
From
To
Via
Max-Forwards

Πίνακας 2: Υποχρεωτικά πεδία του μηνύματος REGISTER.

Μήνυμα BYE

Το μήνυμα BYE χρησιμοποιείται για το τερματισμό μιας συνόδου που έχει εγκατασταθεί εκ των προτέρων μέσω του μηνύματος INVITE. Το μήνυμα BYE αποστέλνεται μόνο από τερματική οντότητα (πράκτορα χρήστη) η οποία επιθυμεί να τερματίσει τη σύνοδο και ποτέ από πληρεξούσιους εξυπηρετητές.

Τα υποχρεωτικά πεδία του μηνύματος BYE φαίνονται στο πιο κάτω πίνακα 3.

Υποχρεωτικά πεδία του μηνύματος BYE
Call-ID
CSeq
From
To
Via
Max-Forwards

Πίνακας 3: Υποχρεωτικά πεδία του μηνύματος BYE.

Μήνυμα ACK

Το μήνυμα ACK αποστέλνεται ως αναφορά λήψης των μηνυμάτων τελικής απόκρισης που αποστέλλονται κατά τη διάρκεια της ροής μηνυμάτων που ακολουθούν το INVITE. Τελικά μηνύματα απόκρισης ορίζονται στο Πρωτόκολλο Έναρξης Συνόδου τα μηνύματα κλάσης 2xx, 3xx, 4xx, 5xx, 6xx τα οποία θα αναφερθούν αργότερα. Η αναφορά λήψης ACK, των μηνυμάτων απόκρισης 2xx (π.χ. του μηνύματος 200 OK) στέλνεται από άκρο σε άκρο, αλλά για οποιαδήποτε άλλα μηνύματα απόκρισης η αποστολή μηνυμάτων γίνεται βήμα – βήμα.

Τα υποχρεωτικά πεδία του μηνύματος ACK φαίνονται στο πιο κάτω πίνακα 4.

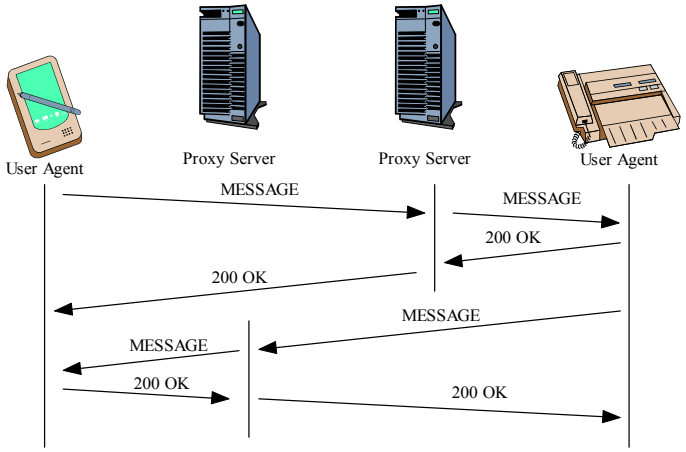
Υποχρεωτικά πεδία του μηνύματος ACK
Call-ID
CSeq
From
To
Via
Max-Forwards

Πίνακας 4: Υποχρεωτικά πεδία του μηνύματος ACK.

Μήνυμα MESSAGE

Το μήνυμα MESSAGE χρησιμοποιείται για τη μεταφορά σύντομων μηνυμάτων (instant messages – IM) πάνω από το πρωτόκολλο σηματοδοσίας SIP. Τα μηνύματα αυτά ανταλλάσσονται σε πραγματικό χρόνο μεταξύ δύο πρακτόρων χρήστη και μπορούν να αποστέλλονται κατά τη παρουσία συνόδου ή χωρίς την παρουσία συνόδου. Ο αποστολέας του μηνύματος MESSAGE δέχεται το μήνυμα απόκρισης 200 OK με το οποίο ο τελικός αποδέκτης πιστοποιεί την ορθή παραλαβή του μηνύματος.

Ένα παράδειγμα ροής μηνυμάτων που αφορούν στο MESSAGE φαίνεται στο σχήμα 6.



Σχήμα 6: Μήνυμα MESSAGE.

Τα υποχρεωτικά πεδία του μηνύματος MESSAGE φαίνονται στον πίνακα 5.

Υποχρεωτικά πεδία του μηνύματος MESSAGE
Call-ID
CSeq
From
To
Via
Max-Forwards

Πίνακας 5: Υποχρεωτικά πεδία του μηνύματος MESSAGE.

Μηνύματα Απόκρισης

Τα μηνύματα απόκρισης είναι μηνύματα που παράγονται από έναν πράκτορα χρήστη εξυπηρετητή (UAS) ή έναν εξυπηρετητή SIP (πληρεξούσιος εξυπηρετητής ή εξυπηρετητής ανακατεύθυνσης) για να αποκριθούν σε κάποια αίτηση ενός πράκτορα χρήστη πελάτη (UAC). Η απόκριση μπορεί να περιέχει κάποια πρόσθετα πεδία επικεφαλίδας που περιέχουν πληροφορία, χρήσιμη για το UAC.

Υπάρχουν 6 κλάσεις μηνυμάτων απόκρισης στο Πρωτόκολλο Έναρξης Συνόδου οι οποίες φαίνονται παρακάτω στον πίνακα 6:

Κλάση	Περιγραφή	Λειτουργία
1xx	Ενημέρωση Informational	Δηλώνει την κατάσταση της κλήσης πριν από την ολοκλήρωση.
2xx	Επιτυχής Έκβαση Success	Επιτυχημένη αίτηση. Αν πρόκειται για απόκριση σε μήνυμα INVITE, το μήνυμα ACK πρέπει να σταλεί.
3xx	Ανακατεύθυνση Redirection	Ο εξυπηρετητής υποδηλώνει πιθανές διευθύνσεις. Ο πελάτης μπορεί να ξαναστείλει εκεί νέα αίτηση.
4xx	Σφάλμα Πελάτη Client Error	Η αίτηση απέτυχε λόγω λάθους που έκανε ο πελάτης. Ο πελάτης μπορεί να ξαναστείλει την αίτηση με βάση την απόκριση που έλαβε.
5xx	Αποτυχία Εξυπηρετητή Server Failure	Η αίτηση απέτυχε λόγω λάθους στον εξυπηρετητή. Η αίτηση μπορεί να σταλεί σε άλλο εξυπηρετητή.
6xx	Γενική Αποτυχία Global Failure	Η αίτηση απέτυχε και δε συνίσταται να αποσταλεί σε αυτό ή άλλο εξυπηρετητή.

Πίνακας 6: Κλάσεις μηνυμάτων απόκρισης.

Μηνύματα Ενημέρωσης – 1xx

- Μήνυμα 100 Trying

Το μήνυμα αυτό αποστέλνεται από έναν πληρεξούσιο εξυπηρετητή ή έναν πράκτορα χρήστη κατά τη διάρκεια της επεξεργασίας μιας κλήσης. Δηλώνει ότι η κλήση τυγχάνει επεξεργασίας και καταβάλλεται προσπάθεια εξεύρεσης ενός χρήστη χωρίς να προσδιορίζεται κατά πόσο έχει βρεθεί. Το μήνυμα αυτό δεν προωθείται και συνήθως δεν περιέχει την επικεφαλίδα To.

- Μήνυμα 180 Ringing

Το μήνυμα αυτό χρησιμοποιείται για να υποδηλώσει ότι ένα μήνυμα INVITE έχει ληφθεί από το πράκτορα χρήστη προορισμού και κάποιος μηχανισμός ειδοποίησης (π.χ. κουδούνισμα) έχει ενεργοποιηθεί για να ενημερώσει τον χρήστη.

Μηνύματα Επιτυχούς Έκβασης – 2xx

- Μήνυμα 200 OK

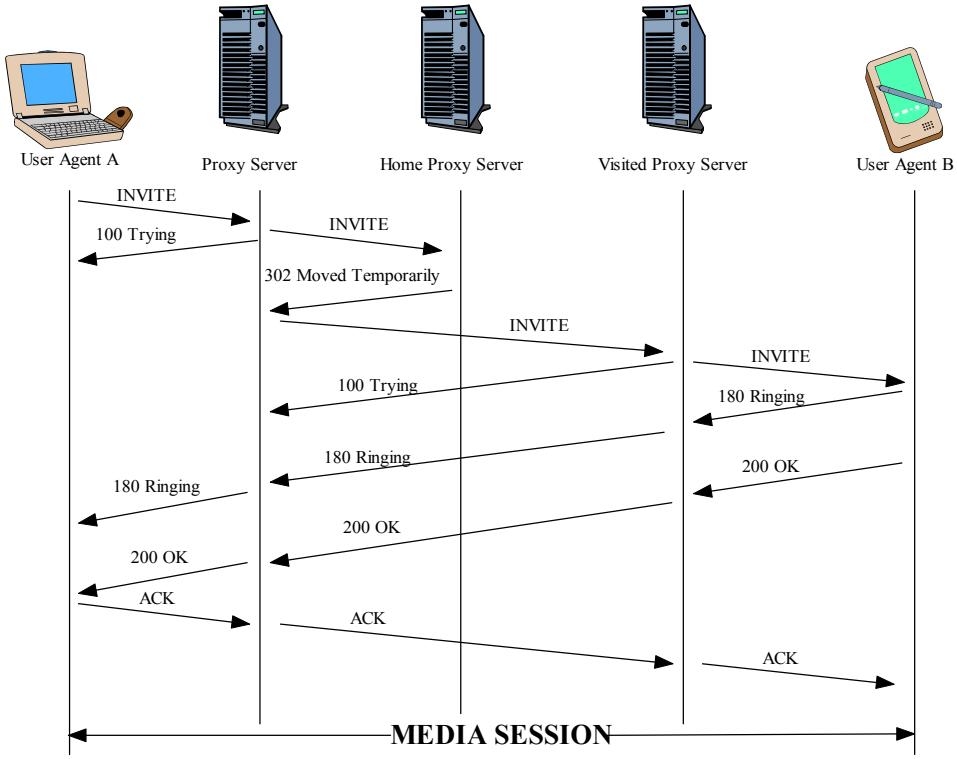
Το μήνυμα αυτό έχει δύο χρήσεις στο Πρωτόκολλο Έναρξης Συνόδου. Πρώτιστα χρησιμοποιείται για την αποδοχή έναρξης συνόδων, οπότε περιέχει στο κυρίως μέρος του σώματός του τα μέσα επικοινωνίας (media properties) του καλούμενου (UAS). Επίσης όταν χρησιμοποιείται σαν απόκριση σε άλλες αιτήσεις δηλώνει επιτυχή τερματισμό ή αποδοχή της αίτησης. Στο πεδίο CSeq του μηνύματος αυτού περιλαμβάνεται το είδος της αίτησης που τερματίζεται με αυτό το μήνυμα 200 OK.

Μηνύματα Ανακατεύθυνσης – 3xx

- Μήνυμα 302 Moved Temporarily

Το μήνυμα αυτό αποστέλνεται από τον τοπικό πληρεξούσιο εξυπηρετητή ενός πράκτορα χρήστη (ο οποίος αναζητήθηκε με βάση το SIP URI του) για να δηλώσει αλλαγή της διεύθυνσης του πράκτορα χρήστη. Το μήνυμα περιέχει στο πεδίο Contact της επικεφαλίδας του, τη νέα διεύθυνση του πράκτορα χρήστη προορισμού που πιθανώς να έχει αλλάξει λόγω της μετακίνησης του σε κάποια νέα περιοχή έτσι ο αποδέκτης του μηνύματος 302 Moved Temporarily να απευθυνθεί στη νέα διεύθυνση του πράκτορα χρήστη προορισμού.

Ένα παράδειγμα ροής μηνυμάτων που αφορούν στο 302 Moved Temporarily φαίνεται στο σχήμα 7.



Σχήμα 7: Μήνυμα 302 Moved Temporarily.

Μηνύματα Σφάλματος Πελάτη – 4xx

- Μήνυμα 404 Not Found

Το μήνυμα Not Found υποδηλώνει ότι ο προς αναζήτηση – με βάση το SIP URI του – πράκτορας χρήστη, δε μπορεί να εντοπιστεί από τον εξυπηρετητή ή ότι ο χρήστης δεν έχει προς το παρόν ενεργοποιήσει το πράκτορα χρήστη του (τη συσκευή του).

2.1.3 Πεδία Επικεφαλίδας των μηνυμάτων του Πρωτοκόλλου Έναρξης Συνόδου

Τα πεδία επικεφαλίδας των μηνυμάτων του SIP μπορούν να διαχωριστούν στα πεδία που ορίζονται από άκρο σε άκρο και στα πεδία που ορίζονται από βήμα σε βήμα. Τα πεδία που ορίζονται από βήμα σε βήμα είναι αυτά τα οποία μπορεί να εισάγει ή να τροποποιήσει – σε μερικές περιπτώσεις – ένας πληρεξούσιος εξυπηρετητής.

Πεδίο Contact

Το πεδίο επικεφαλίδας Contact χρησιμοποιείται για τη μεταβίβαση του SIP URI με το οποίο αναγνωρίζεται ο αποστολέας μιας αίτησης (request) ή μιας απόκρισης (response). Το πεδίο αυτό είναι υποχρεωτικό να υπάρχει σε όλες τις αιτήσεις INVITE και στις αποκρίσεις 200 OK. Όταν ληφθεί ένα μήνυμα που περιέχει το πεδίο Contact, τότε η τιμή του πεδίου αυτού αποθηκεύεται προσωρινά για να χρησιμοποιηθεί σε μελλοντικές αιτήσεις που θα αποσταλούν προς τον πράκτορα χρήστη 5του οποίου το SIP URI βρίσκεται στο Contact.

Πεδίο From

Ένα από τα υποχρεωτικά πεδία σε ένα οποιοδήποτε μήνυμα, είναι το πεδίο From, το οποίο υποδηλώνει αυτόν που έκανε αρχικά την αίτηση. Στο πεδίο From εγγράφεται η μια από τις δυο διευθύνσεις που χρειάζονται για να αναγνωρίζεται μονοσήμαντα μια συνομιλία. Στο πεδίο αυτό τοποθετείται η SIP URI του καλούντα χρήστη.

Πεδίο To

Το πεδίο επικεφαλίδας To υποδεικνύει τον αποδέκτη μιας αίτησης. Ανήκει και αυτό στα υποχρεωτικά πεδία των μηνυμάτων SIP. Μαζί με το πεδίο From καθορίζουν τη συνομιλία και ειδικότερα την κατεύθυνση της συνομιλίας, δηλαδή το ποιος (πεδίο From) έκανε αρχικά την αίτηση και σε ποιον (πεδίο To).

Πεδίο Cseq

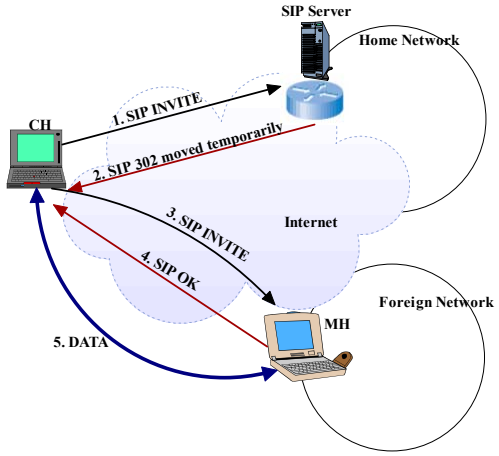
Πρόκειται για άλλο ένα υποχρεωτικό πεδίο επικεφαλίδας. Περιέχει έναν αριθμό που αυξάνεται σε κάθε νέα αίτηση και ένα όνομα μεθόδου (π.χ. INVITE, BYE, REGISTER). Το πεδίο αυτό χρησιμοποιείται από τους UACs για να καταλαβαίνουν σε ποια αίτηση αντιστοιχεί η απόκριση που έχουν πάρει. Για παράδειγμα, ένας UAC που στέλνει μια αίτηση INVITE και στη συνέχεια μια αίτηση CANCEL θα καταλάβει από το όνομα μεθόδου που αναγράφεται στο πεδίο Cseq αν η απόκριση 200 OK που θα πάρει αναφέρεται στο INVITE ή στο CANCEL.

2.2 Διαχείριση Κινητικότητας στο Πρωτόκολλο Έναρξης Συνόδου (Mobility Management in SIP)

Όπως έχει αναφερθεί στην αρχή αυτής της ενότητας το SIP υποστηρίζει διαχείριση προσωπικής κινητικότητας. Με ανταλλαγή μηνυμάτων SIP επιτυγχάνεται και υποστήριξη κινητικότητας IP δηλαδή δυνατότητα να κινείται ένας χρήστης ενόσω μια σύνοδος είναι ενεργή. Θα περιγράψουμε την κινητικότητα με δύο σενάρια· το πρώτο αφορά στην περίπτωση που ένας κινούμενος χρήστης αλλάζει υποδίκτυο πριν να γίνει κάποια αίτηση για εγκατάσταση κλήσης (pre-call mobility) και το δεύτερο αφορά στην περίπτωση που η αλλαγή υποδικτύου γίνεται κατά τη διάρκεια κάποιας κλήσης (mid-call mobility).

2.2.1 Pre-Call Mobility

Σε αυτή την περίπτωση, όταν ο κινούμενος host μετακινείται μεταξύ υποδικτύων εγγράφεται σε ένα Εξυπηρετητή Εγγραφών SIP (SIP Registrar server) του οικείου του δικτύου (home network) ούτως ώστε να μπορεί να βρεθεί. Αυτό αντιστοιχεί στην περίπτωση του Mobile IP που ο κινούμενος host ενημερώνει τον Πράκτορα Οικείων του ότι άλλαξε υποδίκτυο. Όταν ο καλών χρήστης στείλει ένα μήνυμα INVITE στον κινούμενο χρήστη, ο Πληρεξούσιος Εξυπηρετητής του υποδικτύου του κινούμενου host έχει την πληροφορία για το που βρίσκεται, ενημερώνει τον καλούντα προς τα που έχει μετακινηθεί ο προς αναζήτηση χρήστης με τελικό αποτέλεσμα την ανακατεύθυνση (redirect) του αρχικού INVITE (βλέπε σχήμα 8).



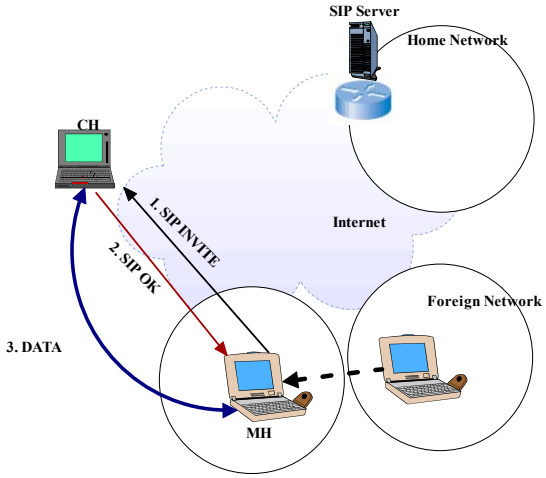
Σχήμα 8: SIP pre-call mobility.

2.2.2 Mid-Call Mobility

Αν ο κινούμενος host μετακινηθεί κατά τη διάρκεια μιας συνόδου, στέλνει ένα νέο INVITE στον καλούντα χρήστη χρησιμοποιώντας τα ίδια χαρακτηριστικά κλήσης (call identifier) με αυτά που χρησιμοποιήθηκαν στην αρχική εγκατάσταση της κλήσης. Τοποθετεί τη νέα του IP διεύθυνση στο πεδίο Contact του μηνύματος SIP έτσι ώστε να ενημερωθεί ο καλών χρήστης για το που να στέλνει τα επόμενα μηνύματα (βλέπε σχήμα9). Το μήνυμα SIP INVITE έχει την ακόλουθη μορφή:

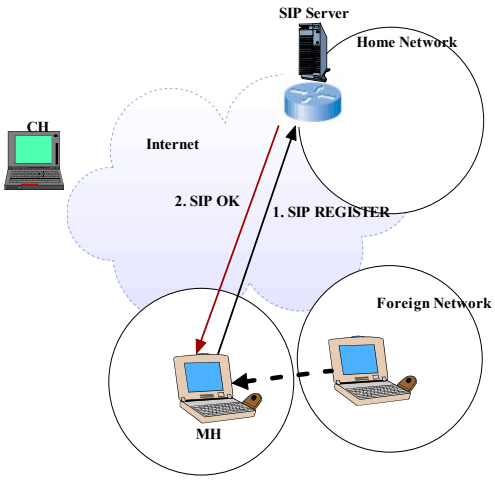
INVITE sip:sender@correspondent.com SIP/2.0
Via: SIP/2.0 /UDP new.ip.address.com:5060
From: sip:mobile@home.ip.address.com
To: sip: sender@correspondent.com
Subject: Changing my IP address
Contact: mobile@new.ip.address.com
Cseq: 123456789 INVITE
Call-ID: …

Όπως βλέπουμε, ο κινούμενος χρήστης με όνομα mobile ενημερώνει τον καλούντα χρήστη με όνομα sender. Η νέα IP διεύθυνση (με το DNS όνομα new.ip.address.com) του mobile τοποθετήθηκε στο πεδίο Contact.



Σχήμα 9: SIP mid-call mobility. Ο κινούμενος host αλλάζει υποδίκτυο.

Τέλος, αφού τελειώσει η πιο πάνω σηματοδοσία και επιτευχθεί η διαπομπή (handoff), ο κινούμενος host ενημερώνει τον οικείο του Εξυπηρετητή Εγγραφών για την νέα του διεύθυνση έτσι ώστε μελλοντικές κλήσεις να ανακατευθύνονται σωστά (βλέπε σχήμα10).



Σχήμα 10: SIP mid-call mobility. Ο κινούμενος host ενημερώνει τον οικείο του Server.

2.3 Περιορισμοί στο SIP

Οι μηχανισμοί κινητικότητας του SIP δεν μπορούν να υποστηρίξουν TCP συνδέσεις. Αυτό έγκειται στο γεγονός ότι οι TCP συνδέσεις απαιτούν σταθερά ζεύγη IP διευθύνσεων και θυρών. Έτσι σε ενδεχόμενη είσοδο κάποιου κινούμενου χρήστη σε ένα νέο υποδίκτυο (handoff), κάτι που συνεπάγεται και αλλαγή IP διεύθυνσης, οι TCP συνδέσεις κόβονται. Αυτό που προτείνεται είναι να χρησιμοποιείται κινητικότητα SIP για επικοινωνίες πραγματικού χρόνου (φωνή και video) πάνω από UDP, ενώ σε περιπτώσεις TCP συνδέσεων μακράς διάρκειας (telnet, ftp, irc) να χρησιμοποιείται το Mobile IP.

Ωστόσο, υπάρχουν κάποιες εφαρμογές TCP που μπορούν να τρέχουν σε κινούμενα τερματικά Internet (mobile Internet terminals) και να βασίζονται στην κινητικότητα με χρήση SIP. Πρόκειται για εφαρμογές που απαιτούν μικρής διάρκειας συνδέσεις μειώνοντας έτσι την πιθανότητα κάποια διαπομπή (handoff) να σπάσει την TCP σύνδεση. Στις εφαρμογές αυτές συμπεριλαμβάνονται οι πλοήγηση στο διαδίκτυο (web browsing) καθώς και οι συνδιαλλαγές e-mail (SMTP mail upload, POP/IMAP mail retrieval). Εκτός από τη μικρή τους διάρκεια τα πρωτόκολλα αυτού του είδους έχουν τη δυνατότητα επαναφοράς εφαρμογών (application-layer recovery) γιατί είναι σχεδιασμένα να λειτουργούν σε περιβάλλοντα ευάλωτα σε διακοπές, όπως οι συνδέσεις dial-up.

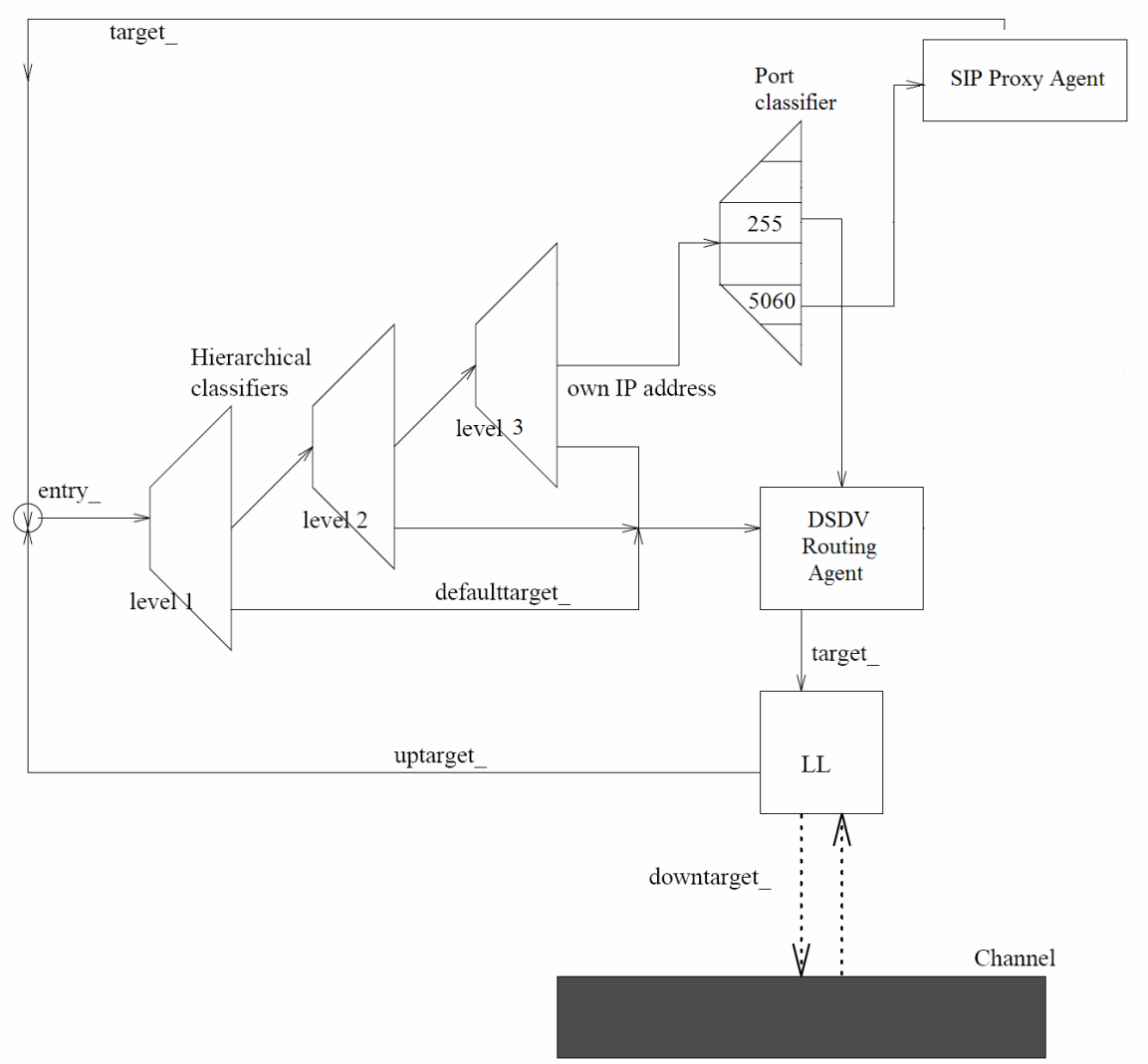
3. Ανάλυση, Σχεδίαση και Υλοποίηση του Πρωτοκόλλου Έναρξης Συνόδου στο περιβάλλον εξομοίωσης δικτύων Network Simulator v2.27

3.1 Ανάλυση και Σχεδίαση

Στην παρούσα διπλωματική εργασία, όπως έχει προαναφερθεί, υλοποιήσαμε τις οντότητες που υπάρχουν σε ένα δίκτυο σηματοδοσίας SIP καθώς και κάποια από τα μηνύματα, ιδιαίτερα αυτά που αποτελούν τη βάση του συστήματος διαχείρισης κινητικότητας σε ασύρματα δίκτυα που χρησιμοποιούν το SIP.

3.1.1 Κόμβοι (Nodes)

Όσον αφορά τις λογικές οντότητες που υπάρχουν στο SIP, υλοποιήσαμε τον πράκτορα χρήστη (User Agent), τον πληρεξούσιο εξυπηρετητή (Proxy Server) και τον εξυπηρετητή καταχωρήσεων (Registrar).

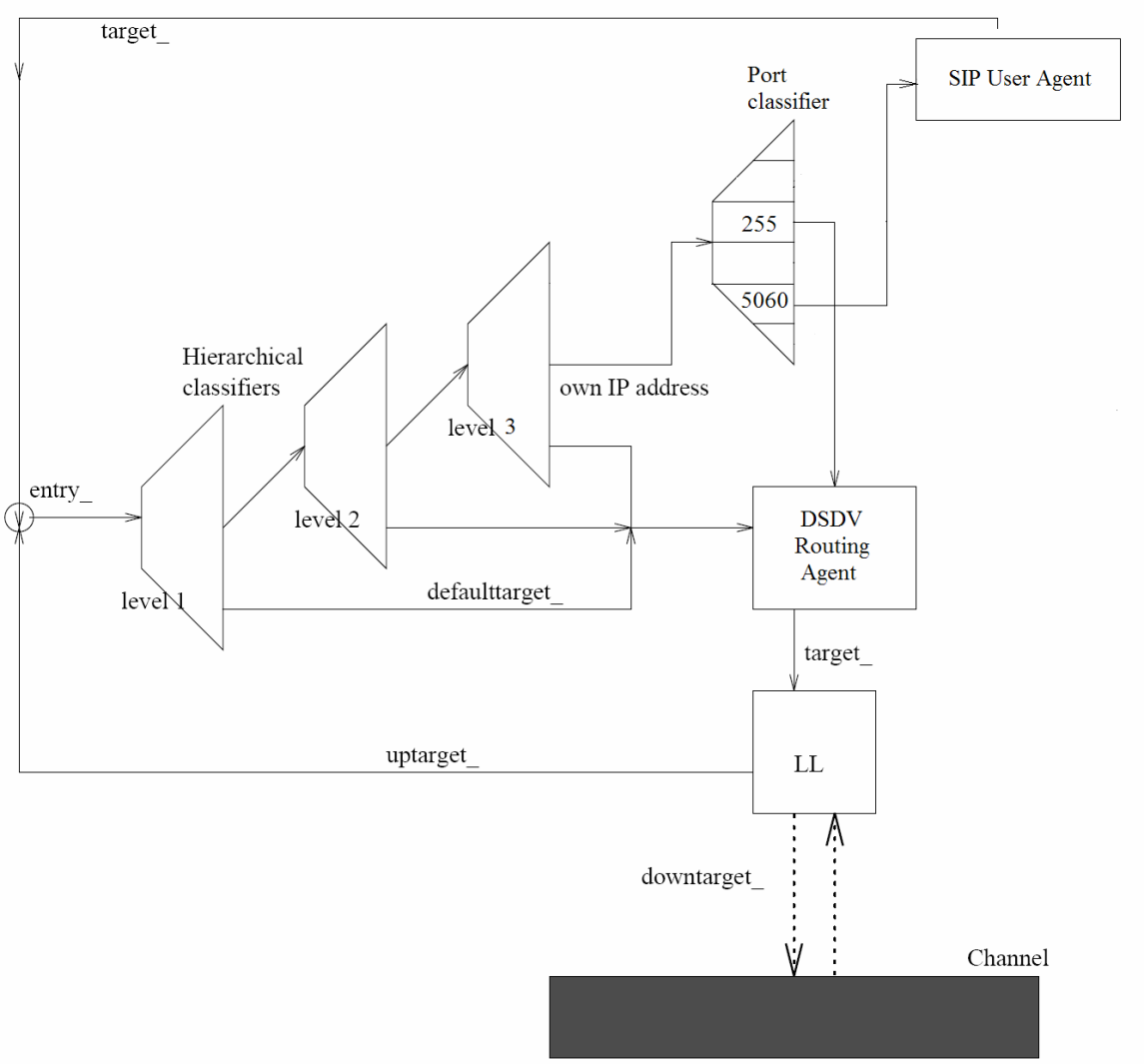


Σχήμα 11: Σταθμός Βάσης σε δίκτυο SIP.

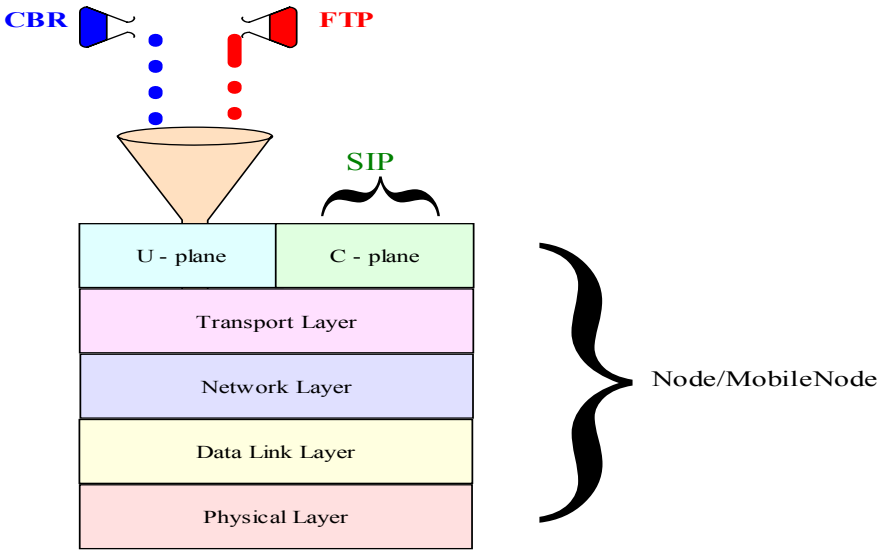
Ειδικότερα υλοποιήσαμε δύο φυσικές οντότητες: έναν πράκτορα χρήστη και άλλη μια φυσική οντότητα η οποία παρέχει τη λειτουργικότητα των δύο λογικών οντοτήτων Proxy Server και Registrar.

Κάθε κόμβος που υπάρχει σε ένα δίκτυο σηματοδοσίας το οποίο λειτουργεί με βάση το Πρωτόκολλο Έναρξης Συνόδου, είτε πρόκειται για κινητό κόμβο (mobilenode) είτε πρόκειται για σταθμό βάσης (base-station node), έχει σχεδιαστεί έτσι ώστε να υλοποιεί τα δύο διαφορετικά επίπεδα (α) σηματοδοσίας – ελέγχου (control plane) και (β) χρήστη (user plane) (βλέπε σχήμα13). Έτσι κάθε κόμβος που υλοποιεί έναν πράκτορα χρήστη ή έναν πληρεξούσιο εξυπηρετητή – εξυπηρετητή εγγραφών, θα έχει τον αντίστοιχο agent

(User Agent ή Proxy Agent αντιστοίχως) να «ακούει» για αιτήσεις ή αποκρίσεις μηνυμάτων σηματοδοσίας, στη θύρα (port) 5060. Στα υπόλοιπές θύρες (ports) του κόμβου, μπορούν να προσδεθούν (bind), διάφορες άλλες εφαρμογές. Στα σχήματα 11 και 12 φαίνονται παραστατικά οι κόμβοι (κινητός κόμβος και σταθμός βάσης αντίστοιχα) και οι οντότητες (υπό μορφή agent) του Πρωτοκόλλου SIP (πληρεξούσιος εξυπηρετητής – εξυπηρετητής εγγραφών) που έχουμε υλοποιήσει στο NS.



Σχήμα 12: Κινητός κόμβος σε δίκτυο SIP.



Σχήμα 13: Node/MobileNode

3.1.2 Βάσεις Δεδομένων Εξυπηρετητή Εγγραφών (Registrar)

Υλοποιήσαμε τις βάσεις δεδομένων του κάθε εξυπηρετητή εγγραφών σαν αρχεία με συγκεκριμένη μορφή. Τα αρχεία αυτά δημιουργούνται από το κάθε Registrar κατά την αρχικοποίηση του κάθε σταθμού βάσης στην Tcl και έχουν δύο στήλες. Στην πρώτη στήλη βρίσκονται οι διαθέσιμες IP διευθύνσεις κάθε περιοχής οι οποίες συνάμα αποτελούν και το SIP URI των χρηστών που ανήκουν – με βάση το IP – στην περιοχή αυτή (είναι η οικεία τους περιοχή). Η βάση δεδομένων στην οικεία περιοχή του κάθε πράκτορα χρήστη, περιέχει στη δεύτερη στήλη, δίπλα από το SIP URI του, τη διεύθυνση που έχει ο κάθε κινούμενος πράκτορας κάθε χρονική στιγμή (σε οποιοδήποτε domain κι αν βρίσκεται). Έτσι ο οικείος πληρεξούσιος εξυπηρετητής μπορεί να ενημερώνει τους εξυπηρετητές/πράκτορες που αιτούνται να επικοινωνήσουν με αυτόν, για τη νέα περιοχή στην οποία βρίσκεται. Επίσης κοιτάζοντας το αρχείο αυτό, ένας πληρεξούσιος εξυπηρετητής μπορεί να ενημερωθεί για το ποιοι πράκτορες χρήστη βρίσκονται μέσα στη περιοχή του.

3.2 Υλοποίηση

Ο κώδικας υλοποίησης των παραπάνω οντοτήτων γράφτηκε στη γλώσσα προγραμματισμού C++ καθώς και στη scripting language OTcl και ενσωματώθηκε στον κώδικα του NS v2.27. Τα αρχεία αυτά θα παρουσιαστούν αναλυτικά παρακάτω, όπως

και μερικές τροποποιήσεις που έγιναν σε διάφορα άλλα αρχεία (C++ και OTcl) που προϋπήρχαν ήδη στο κώδικα του NS v2.27, οι οποίες είχαν καταστεί αναγκαίες για τη λειτουργικότητα και την ορθή εκτέλεση της εφαρμογής μας.

3.2.1 Αρχείο sip.h

Το αρχείο sip.h που παρατίθεται πιο κάτω, περιέχει τους τύπους των μηνυμάτων του Πρωτοκόλλου Έναρξης Συνόδου, τη δήλωση της δομής του κάθε μηνύματος που χρησιμοποιήθηκε στην εφαρμογή καθώς και τις δηλώσεις των κλάσεων των δύο οντοτήτων.

Όπως φαίνεται, έχουν υλοποιηθεί τα μηνύματα αίτησης INVITE, REGISTER, MESSAGE, BYE, ACK και τα μηνύματα απόκρισης 100 Trying, 180 Ringing, 200 OK, 302 Moved Temporarily και 404 Not Found.

```
//Types of messages used in SIP protocol
typedef enum {
INVITE,          //call initialization message
REGISTER,     //USER AGENT -> PROXY/REGISTRAR -- User Agent is registered in the new region
                 //and in its home region
MESSAGE,
BYE,             //call termination message
ACK,             //acknowledgement message
TRYING_100,
RINGING_180,
OK_200,
MOVED_302,       //PROXY -> PROXY -- User Agent is far from its home region (Moved temporarily)
NOT_FOUND_404 //User Agent not found
} SipRegType;
```

Η δομή της επικεφαλίδας του κάθε μηνύματος σηματοδοσίας του πρωτοκόλλου SIP φαίνεται παρακάτω. Τα κυριότερα πεδία που βρίσκονται στην επικεφαλίδα του κάθε μηνύματος και τα οποία είναι απαραίτητα για την υλοποίηση των μηχανισμών κινητικότητας είναι αυτά που φαίνονται πιο κάτω.

```
struct hdr_sip {
  int invite_;            //callee's IP address
  int contact_;           //caller's IP address
  int from_;              //caller's SIP URI;
  int to_;                //callee's SIP URI
```

```
  SipRegType type_; //type of the message

  int index_;           //the index of the application
  bool expires_;        //true for removing binding (default value = false)
  SipRegType cseq_;

  //Header access methods
  static int offset_;    //required by PacketHeaderManager
   inline static hdr_sip* access(const Packet *p) {
       return (hdr_sip*)p->access(offset_);
  }
};
```

Η δήλωση της κλάσης που αφορά στον πράκτορα χρήστη παρουσιάζεται παρακάτω. Η κλάση αυτή συμπεριλαμβάνει τις διάφορες μεθόδους (methods) που χρησιμοποιεί η κλάση αυτή (εκτός από τις standard μεθόδους του NS) οι οποίες είναι:

- sendRegister(int newIp)

Μέθοδος του πράκτορα χρήστη η οποία καλείται όποτε ένας χρήστης εισέρχεται σε μια νέα περιοχή και έχει αποκτήσει μια νέα διεύθυνση IP. Η μέθοδος αυτή, κατασκευάζει ένα μήνυμα REGISTER και το αποστέλλει στον εξυπηρετητή εγγραφών της νέας περιοχής που μόλις έχει εισέλθει ο χρήστης, με σκοπό να γίνει η εγγραφή του χρήστη στη βάση δεδομένων της νέας περιοχής.

- removeBinding()

Μέθοδος του πράκτορα χρήστη η οποία καλείται όποτε ένας χρήστης εξέρχεται από μια περιοχή. Η μέθοδος αυτή, κατασκευάζει ένα μήνυμα REGISTER (με το πεδίο expires = true) και το αποστέλλει στον εξυπηρετητή εγγραφών της παλιάς περιοχής από την οποία που μόλις έχει εξέλθει ο χρήστης, με σκοπό να γίνει σβηστεί η εγγραφή του χρήστη από τη βάση δεδομένων της παλιάς περιοχής.

- isInBounds(double x,double y)

Μέθοδος του πράκτορα χρήστη η οποία καλείται περιοδικά για να ελέγξει εάν ο χρήστης βρίσκεται μέσα στα όρια μιας περιοχής ή αν έχει εισέλθει σε μια νέα. Ο έλεγχος αυτός γίνεται με βάση την απόσταση από τους πληρεξούσιους εξυπηρετητές που τον περιβάλλουν.

```
//USER AGENT
class SipUAgent : public Agent {
 public:
  SipUAgent();
  int command(int argc, const char*const* argv);
  void recv(Packet*, Handler*);
  void timeout(int);

 protected:
  //method that sends REGISTER message whenever a User Agent changes region
  void sendRegister(int newIp);
  //method that removes binding whenever a USER AGENT goes out of bounds of his visited PROXY
  void removeBinding();
  //method that checks if the node is inside the area that a Proxy Agent covers; it is executed
  //periodically
  bool isInBounds(double x,double y);
  SimpleTimer timer_;
  string names[MAX_EL]; //this array holds the name of the node and the names of agents that are
  //attached on this node (except for the sip agent)
  //Proxy Agent coordinates that the User Agent finds from a file
  double xProxy_;
  double yProxy_;
  MobileNode *node_; //ptr to my mobilenode
 private:
  int mySipUri_;         //Well known User Agent's SIP URI
  int dstSipUri_;        //destination's SIP URI
  int communication; //shows the number of applications that are activated on every node
};
```

Η δήλωση της κλάσης που αφορά στον πληρεξούσιο εξυπηρετητή – εξυπηρετητή εγγραφών παρουσιάζεται παρακάτω.

```
//PROXY SERVER - REGISTRAR SERVER
class SipPAgent : public Agent {
 public:
  SipPAgent();   //constructor of the class
  ~SipPAgent(); //destructor of the class
  int command(int argc, const char*const* argv);
  void recv(Packet*, Handler*);
 protected:
  MobileNode *node_;  // pointer to mobilenode
  NsObject *ragent_;   //pointer to the routing agent -- e.g DSDV
};
```

3.2.2 Αρχείο sip.cc

Παρατίθεται τμηματικά πιο κάτω το αρχείο sip.cc. Αρχικά δίνονται επεξηγηματικά σχόλια πάνω στον κώδικα του Πράκτορα Χρήστη (User Agent) και έπειτα πάνω στον

κώδικα του Πληρεξούσιου Εξυπηρετητή – Εξυπηρετητή Εγγραφών (Proxy Server – Registrar).

Πράκτορας Χρήστη (User Agent)

*Μέθοδος command (int argc, const char*const* argv)*

Η μέθοδος command υλοποιεί όλες εκείνες τις ενέργειες που θα γίνουν μόλις ο χρήστης ή κάποια άλλη μέθοδος, δώσει κάποια εντολή στον πράκτορα χρήστη μέσω Tcl. Οι εντολές αυτές είναι οι ακόλουθες:

- *open*

Με την εντολή αυτή ενεργοποιείται ο πράκτορας χρήστη και κάνει αμέσως εγγραφή στη βάση δεδομένων της περιοχής μέσα στην οποία βρίσκεται, στέλνοντας μήνυμα REGISTER στον εξυπηρετητή εγγραφών της περιοχής αυτής.

```
int SipUAgent::command(int argc, const char*const* argv)
{
  if (argc == 2) {
    //this command is executed at the beginning of every User Agent
    if (strcmp(argv[1], "open") == 0) {

      Tcl& tcl = Tcl::instance();

      // Create a new packet
      Packet* pkt = allocpkt();
      // Access the IP header for the new packet:
      hdr_ip* hdrip = hdr_ip::access(pkt);
      // Access the SIP header for the new packet:
      hdr_sip* hdrsip = hdr_sip::access(pkt);

      // Initializing USER AGENT'S SIP URI and IP address
      mySipUri_  = Address::instance().get_nodeaddr(addr());
      dstSipUri_ = Address::instance().get_nodeaddr(daddr());

      if(node_) { // Obtain reference to a MobileNode object so as to use the function base_stn() of
            //the class MobileNode
            // Check if this User Agent's Proxy Agent is the same as its nearest base station
          if(STR2ADDR((GETPROXY(PRINTADDR(addr()))).c_str())==FINDCOORD(node_->X(),node_->Y())-
>ip ) {
         // If User Agent's Proxy = User Agent's base station => IS NOT ASSIGNED a new ip address
                if(FINDCOORD(node_->X(),node_->Y())->ip != node_->base_stn())
                              node_->set_base_stn(FINDCOORD(node_->X(),node_->Y())->ip);
            }
            else {  // User Agent's Proxy != User Agent's base station
                  // Packet to the VISITED PROXY requesting registration
                if(FINDCOORD(node_->X(),node_->Y())->ip != node_->base_stn())
```

```
                        node_->set_base_stn(FINDCOORD(node_->X(),node_->Y())->ip);

                // Changing the nessesary addresses

                string newIp = FINDFREEIP(PRINTADDR(FINDCOORD(node_->X(),node_->Y())->ip));

                addr() = STR2ADDR((newIp).c_str());
                sprintf(tcl.buffer(), "$%s addr %s",(names[0]).c_str(),newIp.c_str());
                tcl.eval();
                sprintf(tcl.buffer(), "[$%s get_ragent] addr %s",(names[0]).c_str(),newIp.c_str());
                tcl.eval();
                sprintf(tcl.buffer(), "[$%s get_ragent] sip %d",(names[0]).c_str(),mySipUri_);
                tcl.eval();
sprintf(tcl.buffer(), "$%s move-dmux %s %s",(names[0]).c_str(), newIp.c_str(), PRINTADDR(addr()));
                tcl.eval();

                //setting the destination of all transport agents that are connected on this node
                for(int i=1; i<MAX_EL; i++) {
                    if(names[i] != "") {
                      sprintf(tcl.buffer(), "$%s set agent_addr_ %d", (names[i]).c_str(), addr() );
                      tcl.eval();
                    }
            }
            }
    }

    // Initializing headers
    hdrip->saddr() = addr();
    hdrip->sport()  = port();
    hdrip->daddr() = node_->base_stn();
    hdrip->dport() = SIP_PORT;
    hdrsip->type_ = REGISTER;
    hdrsip->from_ = mySipUri_; //the sender of the request
    hdrsip->to_  = mySipUri_; //contains the SIP URI of the USER AGENT that is being resistered
    hdrsip->contact_ = addr();

    send(pkt,0);

    xProxy_ = FINDCOORD(node_->X(),node_->Y())->x;
    yProxy_ = FINDCOORD(node_->X(),node_->Y())->y;

    timer_.resched(0.1);

    return(TCL_OK);
   }
  }
```

- *invite*

Με την εντολή αυτή η οποία παίρνει είτε ένα είτε δύο ορίσματα, δίνεται άμεση εντολή στον πράκτορα χρήστη, να αποστείλει μήνυμα INVITE σε ένα ομότιμο πράκτορα χρήστη με τον οποίο έχει εκ των προτέρων ενωθεί (μέσω της εντολής connect σε Tcl), με το οποίο να αιτείται την έναρξη μιας μονόδρομης σύνδεσης ($1^{ο}$ όρισμα) είτε μιας αμφίδρομης σύνδεσης ($1^{ο}$ και $2^{ο}$ όρισμα).

Μήνυμα INVITE με ένα όρισμα

```
else if (argc == 3) {
   if (strcmp(argv[1], "invite") == 0) {

     // Create a new packet
     Packet* pkt = allocpkt();
     // Access the IP header for the new packet:
     hdr_ip* hdrip = hdr_ip::access(pkt);
     // Access the SIP header for the new packet:
     hdr_sip* hdrsip = hdr_sip::access(pkt);

     // IP Header
     hdrip->saddr() = addr();
     hdrip->sport() = port();
     hdrip->daddr() = node_->base_stn(); //Visited Proxy Agent IP address
     hdrip->dport() = SIP_PORT;          //Visited Proxy Agent's well known SIP port

     // SIP Header
     hdrsip->contact_ = addr();      //Caller's IP Address
     hdrsip->from_  = mySipUri_;  //Caller's SIP URI | END to END USER AGENTS
     hdrsip->to_    = dstSipUri_;    //Callee's SIP URI | MUST BE CONNECTED
     hdrsip->invite_= dstSipUri_;   //Callee's IP Address
     hdrsip->type_  = INVITE;        //INVITE message
     hdrsip->index_ = indexx;        //Application index
     appl[indexx][0]  = argv[2];
     appl[indexx++][1]= "none";      //in case of ftp -- simplex connection

     //send the packet
     send(pkt,0);
     cout<<"USER AGENT "<<PRINTADDR(addr())<<" sends an INVITE message at "<<CURR_TIME
<<endl;

     // return TCL_OK, so the calling function knows that the
     // command has been processed
     return(TCL_OK);
}
```

Μήνυμα INVITE με δύο ορίσματα

```
else if ((argc == 4) && (strcmp(argv[2],"agents") != 0)) {
   if (strcmp(argv[1], "invite") == 0) {

     // Create a new packet
     Packet* pkt = allocpkt();
     // Access the IP header for the new packet:
     hdr_ip* hdrip = hdr_ip::access(pkt);
     // Access the SIP header for the new packet:
     hdr_sip* hdrsip = hdr_sip::access(pkt);

     // IP Header
     hdrip->saddr() = addr();
```

```
    hdrip->sport() = port();
    hdrip->daddr() = node_->base_stn(); //Visited Proxy Agent IP address
    hdrip->dport() = SIP_PORT;          //Visited Proxy Agent's well known SIP port


    // SIP Header

    hdrsip->contact_ = addr();       //Caller's IP Address
    hdrsip->from_  = mySipUri_;      //Caller's SIP URI | END to END USER AGENTS
    hdrsip->to_    = dstSipUri_;     //Callee's SIP URI | MUST BE CONNECTED
    hdrsip->invite_= dstSipUri_;     //Callee's IP Address
    hdrsip->type_  = INVITE;         //INVITE message
    hdrsip->index_ = indexx;         //Application index
    appl[indexx][0]  = argv[2];
    appl[indexx++][1]= argv[3];      //in case of cbr -> duplex connection

    //send the packet
    send(pkt,0);
    cout<<"USER AGENT "<<PRINTADDR(addr())<<" sends an INVITE message at "<<CURR_TIME <<
endl;

    // return TCL_OK, so the calling function knows that the
    // command has been processed
    return(TCL_OK);
   }
}
```

- *bye*

Με την εντολή αυτή η οποία παίρνει ένα όρισμα, δίνεται άμεση εντολή στο πράκτορα χρήστη, να τερματίσει μια σύνδεση ($1^{ο}$ όρισμα) αποστέλλοντας μήνυμα BYE σε ένα ομότιμο πράκτορα χρήστη με τον οποίο βρίσκεται συνδεδεμένος.

```
   if (strcmp(argv[1], "bye") == 0) { //it is an end-to-end method

    Tcl& tcl = Tcl::instance();
    // Create a new packet
    Packet* pkt = allocpkt();
    // Access the IP header for the new packet:
    hdr_ip* hdrip = hdr_ip::access(pkt);
    // Access the SIP header for the new packet:
    hdr_sip* hdrsip = hdr_sip::access(pkt);

    // IP Header
    hdrip->saddr() = addr();
    hdrip->sport() = port();
    hdrip->daddr() = daddr();          //Visited Proxy Agent IP address
    hdrip->dport() = SIP_PORT;         //Visited Proxy Agent's well known SIP port

    // SIP Header
    hdrsip->from_  = mySipUri_;
    hdrsip->to_    = dstSipUri_;
    hdrsip->type_  = BYE;              //BYE message
```

```
    // find the index of the application that will be terminated
    for(int i=0; i<MAX_EL; i++) {
        if((appl[i][0] == argv[2]) || (appl[i][1] == argv[2])) {
            if(appl[i][0] == argv[2])
                appl[i][0] = "none";
            else
                appl[i][1] = "none";

            hdrsip->index_ = i;
            break;
        }
    }

    communication--; //a connection will be terminated
    sprintf(tcl.buffer(), "$%s stop", argv[2] );
    tcl.eval();
    cout<<"The session that this SipUAgent has terminated at "<<CURR_TIME<<" is: "<<argv[2] <<
endl;

    //send the packet
    send(pkt,0);
    cout<<"USER AGENT "<<PRINTADDR(addr())<<" sends a BYE message at "<<CURR_TIME<<
endl;

    // return TCL_OK, so the calling function knows that the
    // command has been processed
    return(TCL_OK);
  }
```

- *message*

Με την εντολή αυτή η οποία παίρνει ένα όρισμα, δίνεται άμεση εντολή στον πράκτορα χρήστη, να αποστείλει ένα σύντομο μήνυμα κειμένου (2^o όρισμα) μέσω του μηνύματος MESSAGE σε έναν ομότιμο πράκτορα χρήστη με τον οποίο βρίσκεται συνδεδεμένος.

```
  if(strcmp(argv[1], "message") == 0) {
          int length;
          length=strlen(argv[2]);

          // Create a new packet
          Packet* pkt = allocpkt(length);
          // Access the IP header for the new packet:
          unsigned char *data = pkt->accessdata();
          hdr_ip* hdrip = hdr_ip::access(pkt);
          // Access the SIP header for the new packet:
          hdr_sip* hdrsip = hdr_sip::access(pkt);
          // Access the common header for the new packet:
          hdr_cmn* hdrcmn = hdr_cmn::access(pkt);
          // IP Header
          hdrip->saddr() = addr();
          hdrip->sport() = SIP_PORT;
          hdrip->daddr() = STR2ADDR((GETPROXY(PRINTADDR(daddr()))).c_str());
          hdrip->dport() = SIP_PORT;
```

```
        // SIP Header
        hdrsip->contact_ = addr();       //Caller's IP Address
        hdrsip->from_  = mySipUri_;    //Caller's SIP URI | END to END USER AGENTS
        hdrsip->to_    = dstSipUri_;     //Callee's SIP URI | MUST BE CONNECTED
        hdrsip->invite_= daddr();
        hdrsip->type_  = MESSAGE;      //MESSAGE message
         memcpy (data, argv[2], length);
        hdrcmn->size() += length;

        cout<<"USER AGENT "<<PRINTADDR(addr())<<" sends a MESSAGE at "
<<CURR_TIME<<endl;

         send (pkt, 0);
         return(TCL_OK);
}
```

- *node*

Εντολή η οποία εκτελείται κατά την αρχικοποίηση του πράκτορα μέσω Tcl, για να μας επιστρέψει ένα δείκτη πάνω στο αντικείμενο της κλάσης Node/MobileNode έτσι ώστε να μπορέσουμε να εκτελέσουμε συναρτήσεις της κλάσης αυτής.

```
   else if(strcmp(argv[1], "node") == 0) {
    //obtain reference to a MobileNode object so as to use functions of the class MobileNode -- eg
   //set_base_stn()
    node_ = (MobileNode*)TclObject::lookup(argv[2]);
    if(node_ == 0) {
          fprintf(stderr,"%s: %s lookup of %s failed\n",__FILE__,argv[1],argv[2]);
          return(TCL_ERROR);
    }
   return(TCL_OK);
  }
 }
```

- *[$node_name] agents [$agent_names…]*

Εντολή με πολλά ορίσματα, με την οποία ο χρήστης δίνει σαν πρώτο όρισμα το instance του node πάνω στον οποίο έχει προσαρτηθεί ο πράκτορας χρήστη, σαν δεύτερο όρισμα δίνεται πάντα η λέξη agents και μετά ακολουθούν τα instances των agents (επιπέδου μεταφοράς π.χ. UDP agents, TCP agents, TCPSink agents κ.τ.λ.) τα οποία χρειαζόμαστε μέσα από το πρόγραμμά μας για να μπορούμε να αλλάζουμε – μέσω εντολών Tcl – τις διευθύνσεις IP των προαναφερθέντων agents, όποτε ένας κινούμενος χρήστης αλλάζει περιοχή (και άρα IP διεύθυνση) έτσι ώστε να μπορούν να δρομολογούνται σωστά τα

πακέτα των διαφόρων εφαρμογών (FTP,CBR), ακόμα και μετά την αλλαγή διεύθυνσης IP ενός χρήστη.

```
else { // command with many arguments
    names[0] = argv[1];       //the first argument => name of the node
    for(int i=3; i<argc; i++)  //the rest of the arguments (beyong the word agents) => names of the
                               //agents
            names[i-2] = argv[i];

    return(TCL_OK);

  }
  // If the command hasn't been processed by SipUAgent()::command,
  // call the command() function for the base class
  return (Agent::command(argc, argv));
}
```

Μέθοδος recv(Packet pkt, Handler*)*

Η μέθοδος recv υλοποιεί όλες εκείνες τις ενέργειες που θα γίνουν μόλις ο πράκτορας χρήστη (agent που ακούει στη θύρα 5060) λάβει ένα μήνυμα σηματοδοσίας. Σε κάθε μήνυμα που λαμβάνεται, υπάρχει και το αντίστοιχο μήνυμα απόκρισης γι' αυτό δημιουργείται αρχικά ένα νέο πακέτο προς αποστολή.

```
void SipUAgent::recv(Packet* pkt, Handler*)
{
    //Access the instance -- reference to the instance
    Tcl& tcl = Tcl::instance();

    // Access the IP header for the received packet:
    hdr_ip* hdrip = hdr_ip::access(pkt);
    // Access the SIP header for the received packet:
    hdr_sip* hdrsip = hdr_sip::access(pkt);

    // Create a new packet
    Packet* pktret = allocpkt();
    // Access the IP header for the new packet:
    hdr_ip* hdripret = hdr_ip::access(pktret);
    // Access the SIP header for the new packet:
    hdr_sip* hdrsipret = hdr_sip::access(pktret);

    Packet* pktret2 = allocpkt();
    hdr_ip* hdripret2 = hdr_ip::access(pktret2);
    hdr_sip* hdrsipret2 = hdr_sip::access(pktret2);

    // Access the IP header for the new packet:
    unsigned char *data;
```

Αν το μήνυμα που θα λάβει ένας πράκτορας χρήστη είναι τύπου INVITE, τότε υπάρχουν δύο πιθανότητες.

(α) το εισερχόμενο μήνυμα να προέρχεται από έναn πληρεξούσιο εξυπηρετητή ο οποίος αναμεταδίδει ένα μήνυμα που έλαβε από κάποιο άλλο πληρεξούσιο εξυπηρετητή (που δρα εκ μέρους κάποιου πράκτορα χρήστη) ή από άλλο πράκτορα χρήστη, με σκοπό την ενεργοποίηση κάποιας σύνδεσης. Στη περίπτωση αυτή, ο πράκτορας χρήστη που λαμβάνει το μήνυμα, στέλνει αρχικά πίσω στο πληρεξούσιο εξυπηρετητή το μήνυμα 180 Ringing που υποδηλώνει ότι έχει ενεργοποιηθεί κάποιος μηχανισμός ειδοποίησης του χρήστη. Όταν ο χρήστης απαντήσει τη κλήση, τότε ο πράκτορας χρήστη αποστέλλει το μήνυμα απόκρισης 200 OK πίσω στο πληρεξούσιο εξυπηρετητή.

(β) το εισερχόμενο μήνυμα να προέρχεται άμεσα από έναν πράκτορα χρήστη με τον οποίο υπάρχει ενεργοποιημένη κάποια σύνδεση. Στη περίπτωση αυτή ο πράκτορας που έχει λάβει το μήνυμα (re-)INVITE, στέλνει πίσω το μήνυμα 200 OK.

```
switch(hdrsip->type_) {
        case INVITE:
                if(ISPROXY(PRINTADDR(hdrip->saddr()))) { //if the sender is a proxy send 180
                                                        //RINGING and 200 OK messages
                        cout << "USER AGENT "  << PRINTADDR(addr()) << " received an INVITE
message/sends RINGING message at " <<CURR_TIME<<endl;
                        hdripret->daddr() = hdrip->saddr();
                        hdripret->dport() = hdrip->sport();
                        hdripret->saddr() = addr();
                        hdripret->sport() = port();
                        hdrsipret->from_  = hdrsip->from_;
                        hdrsipret->to_    = hdrsip->to_;
                        hdrsipret->contact_ = addr();
                        hdrsipret->invite_  = hdrsip->contact_;
                        hdrsipret->type_  = RINGING_180;
                        send(pktret,0);

                        //sending OK message
                        cout << "USER AGENT "  << PRINTADDR(addr()) << " sends OK message at
" << CURR_TIME << endl;
                        hdripret2->daddr() = hdrip->saddr();
                        hdripret2->dport() = hdrip->sport();
                        hdripret2->saddr() = addr();
                        hdripret2->sport() = port();
                        hdrsipret2->contact_ = addr();
                        hdrsipret2->from_  = hdrsip->from_;
                        hdrsipret2->to_    = hdrsip->to_;
                        hdrsipret2->invite_= hdrsip->contact_;
                        hdrsipret2->type_  = OK_200;
```

```
                        hdrsipret2->cseq_  = INVITE;
                        hdrsipret2->index_ = hdrsip->index_;
                        send(pktret2,0);
                        //if a mobile that changed region USER AGENT sends INVITE
                        if(daddr() != hdrsip->contact_) {
                                daddr() = hdrsip->contact_;
                                //setting destination address of sip agent and all agents that are
                                //connected on this node
                                for(int i=1; i<MAX_EL; i++) {
                                        if(names[i] != "") {
                    sprintf(tcl.buffer(), "$%s set dst_addr_ %d", (names[i]).c_str(),hdrsip->contact_ );
                                                tcl.eval();
                                        }

                                }

                        }
                }
                else {  //this message re-INVITE comes from a USER AGENT that changed region
                        //and this USER AGENT must send 200 OK
                        //directly to the correspondent USER AGENT
                        cout << "USER AGENT "  << PRINTADDR(addr()) << " has been informed
and sends 200 OK at " << CURR_TIME  << endl;
                        hdripret->daddr() = hdrip->saddr();
                        hdripret->dport() = hdrip->sport();
                        hdripret->saddr() = addr();
                        hdripret->sport() = port();
                        hdrsipret->from_  = hdrsip->from_;
                        hdrsipret->to_    = hdrsip->to_;
                        hdrsipret->type_  = OK_200;

                        hdrsipret->cseq_  = INVITE;
                        send(pktret,0);
                        Packet::free(pktret2);
                }
                Packet::free(pkt);
                break;
```

Αν το μήνυμα που θα λάβει ένας πράκτορας χρήστη είναι τύπου BYE, τότε ο πράκτορας χρήστη, τερματίζει τη σύνδεση για την οποία έλαβε μήνυμα BYE και αποστέλλει σαν απόκριση, το μήνυμα 200 OK.

```
        case BYE:
                cout << "USER AGENT " << PRINTADDR(addr()) << " received BYE message at " <<
CURR_TIME  << endl;
                communication--; //connection will be terminated
                //find out which application will be terminated
                if((appl[hdrsip->index_][0] != "none") || (appl[hdrsip->index_][1] != "none")) {
                        if(appl[hdrsip->index_][0] != "none") {
                                sprintf(tcl.buffer(), "$%s stop", (appl[hdrsip->index_][0]).c_str() );
                                tcl.eval();
                                cout << "The session that this SipUAgent has terminated at " <<
CURR_TIME << " is: " << appl[hdrsip->index_][0] << endl;
                                appl[hdrsip->index_][0] = "none";
```

```
                        }
                        else {
                                sprintf(tcl.buffer(), "$%s stop", (appl[hdrsip->index_][1]).c_str() );
                                tcl.eval();
                                cout << "The session that this SipUAgent has terminated at " <<
CURR_TIME << " is: " << appl[hdrsip->index_][1] << endl;
                                appl[hdrsip->index_][1] = "none";
                        }
                }
                hdripret->daddr() = hdrip->saddr();
                hdripret->dport() = hdrip->sport();
                hdripret->saddr() = addr();
                hdripret->sport() = port();
                hdrsipret->from_  = hdrsip->from_;
                hdrsipret->to_    = hdrsip->to_;

                hdrsipret->type_  = OK_200;

                hdrsipret->cseq_  = BYE;
                send(pktret,0);

                Packet::free(pktret2);
                Packet::free(pkt);
                break;
```

Όσον αφορά στο μήνυμα απόκρισης 200 OK, ο πράκτορας χρήστη εκτελεί διαφορετικές ενέργειες αναλόγως με το ποια τιμή έχει το πεδίο CSeq, δηλαδή αναλόγως με το ποιο μήνυμα αίτησης πυροδότησε το συγκεκριμένο μήνυμα 200 OK. Έτσι διακρίνουμε 4 διαφορετικές περιπτώσεις:

1) REGISTER

Αν το μήνυμα 200 OK είναι απόκριση σε μήνυμα REGISTER που έστειλε ο ίδιος ο πράκτορας χρήστη εισερχόμενος σε μια νέα περιοχή, τότε υπάρχουν και πάλι δύο ενδεχόμενα. Είτε το 200 OK προέρχεται από τον οικείο εξυπηρετητή εγγραφών οπότε δεν υπάρχει καμία απόκριση, είτε προέρχεται από έναν ξένο εξυπηρετητή εγγραφών, οπότε θα πρέπει ο πράκτορας χρήστη να ενημερώσει και τον οικείο του εξυπηρετητή. Η ενημέρωση του οικείου εξυπηρετητή μπορεί να μη γίνει άμεσα αν υπάρχει κάποια ενεργοποιημένη σύνδεση με κάποιο άλλον πράκτορα χρήστη κατά τη διάρκεια της αλλαγής περιοχής. Στη περίπτωση αυτή ο πράκτορας χρήστη θα πρέπει να ενημερώσει τον άλλο πράκτορα για την αλλαγή της IP διεύθυνσης του (λόγω αλλαγής περιοχής) αποστέλλοντας του μήνυμα (re-)INVITE και μετά την ολοκλήρωση της ροής μηνυμάτων του INVITE, θα ενημερώσει και τον οικείο του εξυπηρετητή.

```
        case OK_200:
                switch(hdrsip->cseq_) {
                        case REGISTER:
   cout<<"USER AGENT "<<PRINTADDR(addr())<<" received 200 OK (REG) at "<<CURR_TIME<<endl;
                         //There is two posibilities:
                        //(a)receive 200 OK from HOME PROXY if this USER AGENT re-entered
                        //inside his region => do nothing at all
                        //(b)receive 200 OK from VISITED PROXY => must send REGISTER to his
                        //HOME PROXY to bind his new IP with his SIP URI
                        //but if we are in the middle of a call then firstly USER AGENT must send
                        //INVITE to the correspondent USER AGENT
                        //and sends REGISTER to his HOME PROXY for binding afterwards

                        if(hdrsip->contact_ != 1) {

                                if(communication != 0) { //Inform the Correspondent USER AGENT
                                                         //that this USER AGENT changed region
           cout <<"At " << CURR_TIME <<" USER AGENT'S NEW IP " << PRINTADDR(addr()) << endl;
          cout << "USER AGENT " <<PRINTADDR(addr())<<" informs the correspondent USER
AGENT at "<<CURR_TIME<<endl;
                                hdripret->daddr() = daddr();
                                hdripret->dport() = SIP_PORT;
                                hdripret->saddr() = addr();
                                hdripret->sport() = port();
                                hdrsipret->type_  = INVITE;    //re-INVITE
                                hdrsipret->contact_ = addr();  //USER AGENT'S new IP address
                                send(pktret,0);
                         }
                        else { //if there is no connection inform HOME PROXY
                          cout <<"At " << CURR_TIME <<" USER AGENT'S NEW IP " <<
PRINTADDR(addr()) << endl;
         if(hdrip->saddr() != STR2ADDR((GETPROXY(PRINTADDR(mySipUri_))).c_str())) { //(b)
        cout<<"USER AGENT "<<PRINTADDR(addr())<<" sends REGISTER to his HOME PROXY at
"<<CURR_TIME<<endl;
                                hdripret->daddr() = hdrip->saddr();
                                hdripret->dport() = hdrip->sport();
                                hdripret->saddr() = addr();
                                hdripret->sport() = port();
                                hdrsipret->from_  = mySipUri_;
    hdrsipret->to_    = STR2ADDR((GETPROXY(PRINTADDR(mySipUri_))).c_str()); //to HOME PROXY
                                hdrsipret->type_  = REGISTER
                                hdrsipret->contact_ = addr();  //this is USER AGENT'S new IP that
                                                               //sends to his HOME PROXY
                                send(pktret,0)
                              }
                            }
                          }
                        break;
```

2) INVITE

Αν το μήνυμα 200 OK είναι απόκριση από κάποιο άλλο πράκτορα χρήστη σε μήνυμα INVITE που έστειλε ο ίδιος για να ενεργοποιήσει μια σύνδεση, τότε αποστέλλει το

μήνυμα ACK. Αν το 200 OK είναι απόκριση σε μήνυμα (re-)INVITE τότε ο πράκτορας χρήστη εκτός από την αποστολή του μηνύματος ACK, θα πρέπει να ενημερώσει τον οικείο του εξυπηρετητή εγγραφών (όπως έχει αναφερθεί στην προηγούμενη παράγραφο) αποστέλλοντας του ένα μήνυμα REGISTER.

```
                        case INVITE:
                                cout << "USER AGENT " << PRINTADDR(addr()) << " received 200
OK and sends ACK at " << CURR_TIME << endl;
                                hdripret->daddr() = hdrip->saddr();
                                hdripret->dport() = hdrip->sport();
                                hdripret->saddr() = addr();
                                hdripret->sport() = port();

                                hdrsipret->from_  = hdrsip->from_;

                                hdrsipret->to_    = hdrsip->to_;
                                hdrsipret->contact_ = addr();
                                hdrsipret->invite_= hdrsip->contact_;
                                hdrsipret->index_ = hdrsip->index_;
                                hdrsipret->type_  = ACK;
                                hdrsipret->cseq_  = INVITE;
                                send(pktret,0);

            if(ISPROXY(PRINTADDR(hdrip->saddr()))) { //if it is not an OK 200 after re-INVITE
                    communication++; //new connection will be established
                        if(daddr() != hdrsip->contact_ ) { //if the callee moves in a visited region,
                                                           //caller must change its destination addr
        //setting destination address of sip agent and all agents that are connected on this node
                                for(int i=1; i<MAX_EL; i++) {
                                        if(names[i] != "") {
                                        sprintf(tcl.buffer(),    "$%s    set    dst_addr_    %d",
(names[i]).c_str(),hdrsip->contact_ );
                                        tcl.eval();
                                        }
                                }
                                daddr() = hdrsip->contact_;
                        }
                }
                else { //sends REGISTER to HOME PROXY in order to bind USER's URI
                        //with its new ip address
                            if(     STR2ADDR((GETPROXY(PRINTADDR(addr()))).c_str())     !=
STR2ADDR((GETPROXY(PRINTADDR(mySipUri_))).c_str()) ) {
                                cout << "USER AGENT " << PRINTADDR(addr()) << " sends
REGISTER to his HOME PROXY at "<<CURR_TIME<<endl;
                                hdripret2->daddr() = node_->base_stn();
                                hdripret2->dport() = SIP_PORT;
                                hdripret2->saddr() = addr();
                                hdripret2->sport() = port();
                                hdrsipret2->from_  = mySipUri_;
    hdrsipret2->to_    = STR2ADDR((GETPROXY(PRINTADDR(mySipUri_))).c_str()); //to HOME PROXY
                                hdrsipret2->type_  = REGISTER;
     hdrsipret2->contact_ = addr();  //this is USER AGENT'S new IP that sends to his HOME PROXY
                                send(pktret2,0);
                            }
                        }
```

```
                    break;
```

3-4) BYE/MESSAGE

Αν το μήνυμα 200 OK είναι απόκριση σε μηνύματα BYE ή MESSAGE τότε καμία ενέργεια δεν λαμβάνεται.

```
                case BYE:
                    cout << "USER AGENT " << PRINTADDR(addr()) << " received 200 OK at " << CURR_TIME <<endl;
                    Packet::free(pktret);
                    Packet::free(pktret2);

                    break;

                case MESSAGE:
                    cout<<"USER AGENT "<<PRINTADDR(addr())<<" received an OK_200 \"INSTANT MESSAGE\" MESSAGE at "<<CURR_TIME<<endl;
                    Packet::free(pktret);
                    Packet::free(pktret2);
                    break;
```

Αν το μήνυμα που θα λάβει ένας πράκτορας χρήστη είναι τύπου MESSAGE, τότε τυπώνει το μήνυμα στην οθόνη και αποστέλλει πίσω το μήνυμα 200 OK.

```
    case MESSAGE:
        data = pkt->accessdata();
        data[pkt->datalen()] = '\0';
        printf("USER AGENT %s received MESSAGE %s",PRINTADDR(addr()),data);
        cout << " at " << CURR_TIME << endl;
        hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->contact_))).c_str());
        hdripret->dport() = SIP_PORT;
        hdripret->saddr() = addr();
        hdripret->sport() = SIP_PORT;
        hdrsipret->from_  = hdrsip->from_;
        hdrsipret->to_    = hdrsip->to_;
        hdrsipret->invite_= hdrsip->contact_;
        hdrsipret->contact_ = addr();
        hdrsipret->type_  = OK_200;
        hdrsipret->cseq_  = MESSAGE;
        send(pktret,0);
        Packet::free(pkt);
        Packet::free(pktret2);
        break;
```

Αν το μήνυμα που θα λάβει ένας πράκτορας χρήστη είναι τύπου ACK, τότε δεν αποστέλλει κανένα μήνυμα απόκρισης, αλλά ενημερώνει κάποιες μεταβλητές της εφαρμογής.

```
        case ACK:
                cout << "USER AGENT "  << PRINTADDR(addr()) << " received ACK message at " <<
CURR_TIME << endl;
                if(ISPROXY(PRINTADDR(hdrip->saddr()))) {
                        communication++; //new connection will be established
                        if(appl[hdrsip->index_][1] == "none") {
                                sprintf(tcl.buffer(), "$%s start", (appl[hdrsip->index_][0]).c_str() );
                                tcl.eval();
                                cout << "The session that this SipUAgent has established at "
<<CURR_TIME<<" is: " << appl[hdrsip->index_][0] << endl;
                        }
                        else {
                                sprintf(tcl.buffer(), "$%s start", (appl[hdrsip->index_][0]).c_str() );

                                tcl.eval();

                                sprintf(tcl.buffer(), "$%s start", (appl[hdrsip->index_][1]).c_str() );
                                tcl.eval();
                                cout << "The session that this SipUAgent has established at "
<<CURR_TIME<<" is: " << appl[hdrsip->index_][0] << endl;
                                cout << "The session that this SipUAgent has established at "
<<CURR_TIME<<" is: " << appl[hdrsip->index_][1] << endl;
                        }
                }
                else { //ACK message after re-INVITE
                        //setting destination address of sip agent and all agents that are connected
                        //on this node
                        for(int i=1; i<MAX_EL; i++) {
                                if(names[i] != "") {
                                sprintf(tcl.buffer(), "$%s set dst_addr_ %d", (names[i]).c_str(),hdrsip->contact_ );
                                        tcl.eval();
                                }
                        }
                        daddr() = hdrsip->contact_;
                }
                Packet::free(pkt);
                Packet::free(pktret);
                Packet::free(pktret2);
                break;
```

Όσον αφορά τα μηνύματα 100 Trying, 180 Ringing και 404 Not Found, καμία ενέργεια δε λαμβάνεται από το πράκτορα χρήστη μετά τη λήψη τους.

```
        case TRYING_100:
                cout << "USER AGENT "  << PRINTADDR(addr()) << " received 100 TRYING message
at " << CURR_TIME << endl;
                Packet::free(pkt);
```

```
                Packet::free(pktret);
                Packet::free(pktret2);
                break;
        case RINGING_180:
                cout << "USER AGENT "  << PRINTADDR(addr()) << " received 180 RINGING
message at " << CURR_TIME << endl;
                Packet::free(pkt);
                Packet::free(pktret);
                Packet::free(pktret2);
                break;
        case NOT_FOUND_404:
                cout << "USER AGENT "  << PRINTADDR(addr()) << " received 404 NOT FOUND
message at " << CURR_TIME << endl;
                Packet::free(pkt);
                Packet::free(pktret);
                Packet::free(pktret2);
                break;
```

Μέθοδος isInBounds(double x,double y)

Η μέθοδος αυτή καλείται περιοδικά μέσω της μεθόδου timeout που παρουσιάζεται πιο κάτω και ελέγχει αν ο κινούμενος πράκτορας χρήστη έχει βγει από την εμβέλεια κάποιου σταθμού βάσης οπότε επιστρέφει true αλλιώς αν βρίσκεται ακόμα στην εμβέλεια του ίδιου σταθμού βάσης τότε επιστρέφει false.

```
//Checks if User Agent is in or out of Proxy Agent's range
bool SipUAgent::isInBounds(double x,double y)
{
        //RXThresh_ default value is 3.652e-10 W. Using /indep-utils/propagation/threshold.cc
        //we can evaluate the default range of the base-station which is 250m -- for
        //TwoRayGround model
        if((xProxy_ == 0.0) && (yProxy_ == 0.0)) return true; //initial state
        if( sqrt( pow(xProxy_-x,2) + pow(yProxy_-y,2) ) < RANGE )
                return true;  //User Agent is in bounds
        else
                return false; //User Agent is out of bounds
}
```

Μέθοδος timeout(int)

Η μέθοδος timeout καλείται περιοδικά κάθε 0.1sec και καλεί τη μέθοδο isInBounds για να διαπιστώσει πιθανή αλλαγή περιοχής. Αν δε διαπιστωθεί αλλαγή περιοχής τότε δεν γίνεται καμιά ενέργεια αλλιώς ο πράκτορας χρήστη αποκτά μια νέα διεύθυνση IP και στέλνει μηνύματα REGISTER τόσο στο νέο εξυπηρετητή εγγραφών για να ενημερώσει

τη βάση δεδομένων της νέας περιοχής (βάζοντας στο πεδίο επικεφαλίδας Contact τη νέα του IP διεύθυνση) όσο και στον παλιό εξυπηρετητή εγγραφών για να ακυρώσει την εγγραφή στη παλιά βάση δεδομένων της περιοχής από την οποία έχει μόλις βγει ο χρήστης (βάζοντας στο πεδίο επικεφαλίδας Contact την παλιά του IP διεύθυνση).

```
//Every given period of time this function is invoked
void SipUAgent::timeout(int)
{
        Tcl& tcl = Tcl::instance();
        string newIp, oldIp;
        if(!isInBounds(node_->X(),node_->Y())) {
                cout<<"USER AGENT "<<PRINTADDR(addr())<<" enters in region "<<
PRINTADDR(FINDCOORD(node_->X(),node_->Y())->ip)  << " at " << CURR_TIME <<endl;
                oldIp = PRINTADDR(addr()) ;

                //User agent's new ip address
               //check if the USER AGENT entered inside his home region

             if(STR2ADDR((GETPROXY(PRINTADDR(mySipUri_))).c_str())!=FINDCOORD(node_-
>X(),node_->Y())->ip)

                        newIp = FINDFREEIP(PRINTADDR(FINDCOORD(node_->X(),node_->Y())->ip));
           else
                        newIp = PRINTADDR(mySipUri_);

         //If USER AGENT gets outside of a region which is not his home region
         if(node_->base_stn() != STR2ADDR((GETPROXY(PRINTADDR(mySipUri_))).c_str()))

         //Send REGISTER message with expires_ = true to his old visited PROXY AGENT so as to
        //remove binding
                removeBinding();

                //USER AGENT changes base station
                node_->set_base_stn(FINDCOORD(node_->X(),node_->Y())->ip);
                // Get the coordinates of the closest PROXY AGENT - Base station
                xProxy_ = FINDCOORD(node_->X(),node_->Y())->x;
                yProxy_ = FINDCOORD(node_->X(),node_->Y())->y;
                // Changing the nessesary addresses
                // 1.Changing the addr_ of this sip agent
                addr() = STR2ADDR(newIp.c_str());
                // 2.Changing the address_ of this node
                sprintf(tcl.buffer(), "$%s addr %s",(names[0]).c_str(),newIp.c_str());
                tcl.eval();
                // 3.Changing myaddr_ of DSDV routing agent
                sprintf(tcl.buffer(), "[$%s get_ragent] addr %s",(names[0]).c_str(),newIp.c_str());
                tcl.eval();
                // 4.Sending update to the new base station
                sprintf(tcl.buffer(), "[$%s get_ragent] sip %d",(names[0]).c_str(),mySipUri_);
                tcl.eval();
                // Moving the dmux of this node from the old IP address to the new IP address
           sprintf(tcl.buffer(), "$%s move-dmux %s %s",(names[0]).c_str(), newIp.c_str(), oldIp.c_str());
                tcl.eval();

                //setting the destination of all transport agents that are connected on this node
                for(int i=1; i<MAX_EL; i++) {
                        if(names[i] != "") {
```

```
                                sprintf(tcl.buffer(), "$%s set agent_addr_ %d", (names[i]).c_str(), addr() );
                                tcl.eval();
                        }
                }

                //USER AGENT wants to register with his new PROXY AGENT
                sendRegister(STR2ADDR(newIp.c_str()));
        }
        timer_.resched(0.1);
}
```

Μέθοδος sendRegister(int newIp)

Η μέθοδος αυτή καλείται μέσα από τη μέθοδο timeout για να δημιουργήσει ένα μήνυμα REGISTER και να το αποστείλει στο νέο εξυπηρετητή εγγραφών έτσι ώστε να γίνει η καταχώρηση στη βάση δεδομένων.

```
//Function that sends a REGISTER message whenever the USER AGENT changes region
void SipUAgent::sendRegister(int newIp)
{
        Packet  *pkt    = allocpkt();
        hdr_ip  *hdrip  = hdr_ip::access(pkt);
        hdr_sip *hdrsip = hdr_sip::access(pkt);

        //Find the IP address of the Proxy Agent which is near from this USER AGENT
        hdrip->saddr() = addr();
        hdrip->sport() = port();
        hdrip->daddr() = FINDCOORD(node_->X(),node_->Y())->ip; //function that returns the IP
        //address of the closest PROXY SERVER
        hdrip->dport() = SIP_PORT;   //well-known sip port that PROXY AGENT listens for requests
        hdrsip->contact_ = newIp;    //Register with the new IP address
        hdrsip->from_ = mySipUri_;
        hdrsip->to_   = mySipUri_;
        hdrsip->type_ = REGISTER;

        send(pkt,0);
}
```

Μέθοδος removeBinding()

Η μέθοδος αυτή καλείται επίσης μέσα από τη μέθοδο timeout για να δημιουργήσει ένα μήνυμα REGISTER και να το αποστείλει στον παλιό εξυπηρετητή εγγραφών έτσι ώστε να διαγράψει την παλιά καταχώρηση από την παλιά βάση δεδομένων.

```
void SipUAgent::removeBinding()
{
        Packet  *pkt    = allocpkt();
        hdr_ip  *hdrip  = hdr_ip::access(pkt);
        hdr_sip *hdrsip = hdr_sip::access(pkt);
        //USER AGENT sends a REGISTER message to his visited PROXY AGENT requesting
        //removing of the previous binding
        hdrip->saddr() = addr();
        hdrip->sport() = port();
        hdrip->daddr() = node_->base_stn(); //FINDCOORD(node_->X(),node_->Y())->ip; //function
        //that returns the IP address of the closest PROXY AGENT
        hdrip->dport() = SIP_PORT;
        hdrsip->from_    = mySipUri_;
        hdrsip->contact_ = addr();
        hdrsip->expires_ = true;
        hdrsip->type_    = REGISTER;

//      FREEIP(PRINTADDR(addr()));

        send(pkt,0);
}
```

Πληρεξούσιος Εξυπηρετητής – Εξυπηρετητής Εγγραφών (Proxy Server – Registrar)

Μέθοδος command (int argc, const char*const* argv)

Η μέθοδος command υλοποιεί όλες εκείνες τις ενέργειες που θα γίνουν μόλις ο χρήστης ή κάποια άλλη μέθοδος, δώσει κάποια εντολή στον εξυπηρετητή μέσω Tcl. Οι εντολές αυτές είναι οι ακόλουθες:

- *open*

Με την εντολή αυτή ενεργοποιείται ο εξυπηρετητής.

```
int SipPAgent::command(int argc, const char*const* argv)
{
        // Initialization of coordFl file -- each Proxy appends its coordinates and ip address
        if (argc == 2) {
        if (strcmp(argv[1], "open") == 0) {
          CREATECOORDFILE(node_->X(),node_->Y(),atoi((GETDOMAIN(PRINTADDR(addr())))).c_str()));
          return TCL_OK;
          }
        }
```

- *initialize*

Με την εντολή αυτή αρχικοποιείται ο διαθέσιμος αριθμός IP διευθύνσεων σε κάθε περιοχή ενός εξυπηρετητή καθώς και ποιες θα είναι αυτές.

```
        if (argc == 3) {
           // Initialization of the Registration File with 9 available ip addresses
           if (strcmp(argv[1], "initialize") == 0) {
                CRREGFILE( atoi((GETDOMAIN(argv[2])).c_str()) ,9);
                return TCL_OK;
           }
```

- *node*

Η εντολή αυτή – όπως και στη περίπτωση του πράκτορα χρήστη – εκτελείται κατά την αρχικοποίηση του εξυπηρετητή μέσω Tcl, για να μας επιστρέψει ένα δείκτη πάνω στο αντικείμενο της κλάσης Node/MobileNode έτσι ώστε να μπορέσουμε να εκτελέσουμε συναρτήσεις της κλάσης αυτής.

```
if(strcmp(argv[1], "node") == 0) {
            // Obtain reference to a MobileNode object so as to use functions of the class MobileNode
                  node_ = (MobileNode*)TclObject::lookup(argv[2]);
                  if(node_ == 0) {
                        fprintf(stderr,"%s: %s lookup of %s failed\n",__FILE__,argv[1],argv[2]);
                        return(TCL_ERROR);
                  }
                  return(TCL_OK);
             }
```

- *ragent*

Η εντολή αυτή εκτελείται κατά την αρχικοποίηση του εξυπηρετητή μέσω Tcl, για να μας επιστρέψει ένα δείκτη πάνω στο routing agent.

```
            // Bind routing agent and sip agent
              if(strcmp(argv[1], "ragent") == 0) {
                  ragent_ = (NsObject*)TclObject::lookup(argv[2]);
                  return(TCL_OK);
              }
         }

         // If the command hasn't been processed by SipUAgent()::command,
         // call the command() function for the base class
         return (Agent::command(argc, argv));
}
```

Μέθοδος recv(Packet* pkt, Handler*)

Η μέθοδος recv υλοποιεί όλες εκείνες τις ενέργειες που θα γίνουν μόλις ο εξυπηρετητής (agent που ακούει στη θύρα 5060) λάβει ένα μήνυμα σηματοδοσίας. Σε κάθε μήνυμα που λαμβάνεται, υπάρχει και το αντίστοιχο μήνυμα απόκρισης γι' αυτό δημιουργείται αρχικά ένα νέο πακέτο προς αποστολή όπως και στην περίπτωση του πράκτορα χρήστη.

Αν το μήνυμα που θα λάβει ένας εξυπηρετητής είναι τύπου REGISTER και έχει το πεδίο επικεφαλίδας expires_ ενεργοποιημένο τότε ο εξυπηρετητής σβήνει την καταχώρηση για το συγκεκριμένο IP που βρίσκεται στο πεδίο Contact του παραληφθέντος μηνύματος από τη βάση δεδομένων του. Αν δεν είναι ενεργοποιημένο το πεδίο expires_ τότε ο

εξυπηρετητής που λαμβάνει το μήνυμα ελέγχει αν απευθύνεται σε αυτόν οπότε εκτελεί τις απαραίτητες ενέργειες (ενημέρωση της βάσης δεδομένων που συντηρεί και αποστολή του μηνύματος 200 OK προς τον αποστολέα του μηνύματος) ή αλλιώς επαναμεταδίδει το μήνυμα προς τον τελικό αποδέκτη.

```
        switch(hdrsip->type_) {
                case REGISTER:
                        if(hdrsip->expires_)   { //Removing binding
                                FREEIP(PRINTADDR(hdrsip->contact_));
                              cout<<"PROXY "<<PRINTADDR(addr())<<" is removing binding"<<endl;
                                Packet::free(pktret);
                                Packet::free(pkt);
                                break;
                        }
                        cout << "PROXY " << PRINTADDR(addr())  << " received a REGISTER
message at " << CURR_TIME << endl;
                        //preparing the new packet to be sent
                        hdripret->saddr() = addr();
                        hdripret->sport() = port();
                        if(hdrsip->from_ == hdrsip->to_) {
                                //PROXY AGENT updates the registration file - database
                                GETFREEIP(PRINTADDR(hdrsip->contact_));
                                hdripret->daddr() = hdrip->saddr();
                                 hdripret->dport() = SIP_PORT;
                                hdrsipret->type_  = OK_200;
                                hdrsipret->cseq_  = REGISTER;
                                send(pktret,0);
                        }
                        else {
                                if(hdrsip->to_ == addr()) {
                                        //PROXY AGENT binds the new ip of the USER AGENT
                                        // with his SIP URI -- registration on behalf of visited Proxy
                        INSMOVEDAG(PRINTADDR(hdrsip->from_),PRINTADDR(hdrsip->contact_));
                        hdripret->daddr() = hdrip->saddr(); //send REG_OK to visited PROXY
                                        hdripret->dport() = hdrip->sport();
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;
                                        hdrsipret->type_  = OK_200;
                                        hdrsipret->cseq_  = REGISTER;
                                }
                                else {
                                        //send the REGISTER message that has just arrived from
                                        //new USER AGENT to USER AGENT's HOME PROXY
                                        hdripret->daddr()   = hdrsip->to_;
                                        hdripret->dport()   = SIP_PORT;
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_    = hdrsip->from_;
                                        hdrsipret->to_      = hdrsip->to_;
                                        hdrsipret->type_    = REGISTER;

                                }
                                send(pktret,0);
                        }
```

```
                    Packet::free(pkt);
                    Packet::free(pktret2);
                    break;
```

Αν το μήνυμα που θα λάβει ένας εξυπηρετητής είναι τύπου INVITE ελέγχει αν απευθύνεται σε κάποιο πράκτορα χρήστη του οποίου ο οικείος εξυπηρετητής είναι αυτός. Αν το μήνυμα απευθύνεται σε πράκτορα χρήστη του οποίου είναι ο οικείος εξυπηρετητής, τότε ελέγχει – εκτελώντας query προς τη βάση δεδομένων της περιοχής του – αν ο προς αναζήτηση πράκτορας χρήστη βρίσκεται εντός της εμβέλειας του για να του αποστείλει το μήνυμα INVITE. Αν ο προς αναζήτηση χρήστης έχει απενεργοποιημένη τη συσκευή του, τότε ο εξυπηρετητής αποστέλλει πίσω το μήνυμα 404 Not Found.

```
            case INVITE:
                    cout << "PROXY " << PRINTADDR(addr())  << " received an INVITE message
at " << CURR_TIME << endl;
                    //Proxy must check if the callee belongs to its region and if it is inside
                    if (STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str()) == addr()) {
                    //Callee belongs to this Proxy
                    //check if callee has his device closed - not registered to the network
                            if(GETVISPROXYADDR(PRINTADDR(hdrsip->invite_))=="not_found"){
                                    hdripret->daddr() = hdrip->saddr();
                                    hdripret->dport() = SIP_PORT;
                                    hdripret->saddr() = addr();
                                    hdripret->sport() = port();
                                    hdrsipret->contact_ = hdrsip->contact_;
                                    hdrsipret->from_  = hdrsip->from_;
                                    hdrsipret->to_    = hdrsip->to_;
                                    hdrsipret->invite_= hdrsip->invite_;
                                    hdrsipret->type_  = NOT_FOUND_404;
                                    send(pktret,0);
                            }
```

Αν ο προς αναζήτηση χρήστης έχει ενεργοποιημένη τη συσκευή του και βρίσκεται εντός της εμβέλειας του οικείου του εξυπηρετητή, τότε ο εξυπηρετητής του επαναμεταδίδει το μήνυμα INVITE και στέλνει προς τα πίσω το μήνυμα 100 Trying.

```
else if(STR2ADDR((GETVISPROXYADDR(PRINTADDR(hdrsip->invite_))).c_str())  == addr()) {
                                    //Callee is inside Proxy's region
                                    //preparing to send the new packet to the User Agent
                                    //that is inside Proxy's region
```

```
                                        cout << "PROXY " << PRINTADDR(addr()) << " sends
INVITE to " << PRINTADDR(hdrsip->invite_) << " at " << CURR_TIME << endl;

                                        hdripret->daddr() = hdrsip->invite_;

                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;  //caller's SIP URI
                                        hdrsipret->to_     = hdrsip->to_;       //callee's SIP URI
                                        hdrsipret->invite_= hdrsip->invite_;
                                        hdrsipret->type_  = INVITE;
                                        hdrsipret->index_ = hdrsip->index_;
                                        send(pktret,0);
                                        //Send Trying_100 to caller or to caller's Proxy
                                        cout << "PROXY " << PRINTADDR(addr()) << " sends 100
TRYING to " << PRINTADDR(hdrip->saddr())<< " at " << CURR_TIME <<endl;
                                        hdripret2->daddr() = hdrip->saddr();
                                        hdripret2->dport() = SIP_PORT;
                                        hdripret2->saddr() = addr();
                                        hdripret2->sport() = port();
                                        hdrsipret2->type_  = TRYING_100;
                                        send(pktret2,0);
}
```

Αν ο προς αναζήτηση χρήστης έχει ενεργοποιημένη τη συσκευή του και βρίσκεται εκτός της εμβέλειας του οικείου του εξυπηρετητή, τότε ο εξυπηρετητής ελέγχει αν το μήνυμα INVITE προήλθε από κάποιο άλλο πληρεξούσιο εξυπηρετητή ή από άλλο πράκτορα χρήστη που βρίσκεται εντός της περιοχής εμβέλειας του. Αν προήλθε από ένα πληρεξούσιο εξυπηρετητή τότε του αποστέλλει πίσω το μήνυμα 302 Moved Temporarily τοποθετώντας στο πεδίο επικεφαλίδας Contact το νέο IP του προς αναζήτηση πράκτορα χρήστη. Αν προήλθε από ένα πράκτορα χρήστη τότε αυτός αναλαμβάνει να επαναμεταδώσει το μήνυμα INVITE στον πληρεξούσιο εξυπηρετητή της περιοχής που βρίσκεται ο προς αναζήτηση χρήστης αποστέλλοντας ταυτοχρόνως πίσω στον πράκτορα χρήστη το μήνυμα 100 Trying.

```
else  { //Callee is inside other Proxy's region
                                        //The sender of the INVITE is a UA or a PA?
                                        //(a) PA then send the new packet MOVED to the caller's
                                        //Proxy Agent
                                        //(b) UA relay the INVITE message to callee's Proxy Agent
                                        if( ISPROXY(PRINTADDR(hdrip->saddr())) ) { //(a)
                                                hdripret->daddr() = hdrip->saddr();
                                                hdripret->dport() = hdrip->sport();
                                                hdripret->saddr() = addr();
                                                hdripret->sport() = port();
```

```
                hdrsipret->contact_ = STR2ADDR((GETVISADDR(PRINTADDR(hdrsip->invite_))).c_str());
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;
                                        hdrsipret->invite_= hdrsip->contact_;

                                        hdrsipret->type_  = MOVED_302;

                                        hdrsipret->index_ = hdrsip->index_;
                                        send(pktret,0);
                                }
                                else { //(b)

        hdripret->daddr() = STR2ADDR((GETVISPROXYADDR(PRINTADDR(hdrsip->invite_))).c_str());
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;
                hdrsipret->invite_= STR2ADDR((GETVISADDR(PRINTADDR(hdrsip->invite_))).c_str());
                                        hdrsipret->type_  = INVITE;
                                        hdrsipret->index_ = hdrsip->index_;
                                        send(pktret,0);
                                        //Send Trying_100 to caller
                                        cout << "PROXY " << PRINTADDR(addr()) << "
sends 100 TRYING to " << PRINTADDR(hdrip->saddr())<< " at " <<CURR_TIME<<endl;
                                        hdripret2->daddr() = hdrip->saddr();
                                        hdripret2->dport() = SIP_PORT;
                                        hdripret2->saddr() = addr();
                                        hdripret2->sport() = port();
                                        hdrsipret2->type_  = TRYING_100;
                                        send(pktret2,0);
                                }
                        }
}
```

Αν το εισερχόμενο μήνυμα απευθύνεται σε πράκτορα χρήστη του οποίου δεν είναι ο οικείος εξυπηρετητής, επαναμεταδίδει το μήνυμα προς τον οικείο εξυπηρετητή του τελικού αποδέκτη ελέγχοντας κάποιο από τα πεδία της επικεφαλίδας. Επίσης αποστέλλει προς τα πίσω μήνυμα 100 Trying.

```
else {  //The callee does not belong to Proxy's region
      //Send the new packet to the Proxy Server of the callee
                        cout<<"PROXY    "<<PRINTADDR(addr())<<"  sends  INVITE  to
"<<GETPROXY(PRINTADDR(hdrsip->invite_))<< " at " << CURR_TIME<<endl;
                hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str());
                        hdripret->dport() = SIP_PORT;
                        hdripret->saddr() = addr();
                        hdripret->sport() = port();
                        hdrsipret->contact_ = hdrsip->contact_;
                        hdrsipret->from_  = hdrsip->from_;
                        hdrsipret->to_    = hdrsip->to_;
                        hdrsipret->invite_= hdrsip->invite_;
```

```
                                hdrsipret->type_  = INVITE;
                                hdrsipret->index_ = hdrsip->index_;
                                send(pktret,0);
                                //send TRYING to caller
                                cout << "PROXY " << PRINTADDR(addr()) << " sends 100 TRYING to
" << PRINTADDR(hdrip->saddr())<< " at " << CURR_TIME<< endl;

                                hdripret2->daddr() = hdrip->saddr();

                                hdripret2->dport() = SIP_PORT;
                                hdripret2->saddr() = addr();
                                hdripret2->sport() = port();
                                hdrsipret2->type_  = TRYING_100;
                                send(pktret2,0);
                        }
                        Packet::free(pkt);
                        break;
```

Αν το μήνυμα που θα λάβει ένας εξυπηρετητής είναι τύπου 180 Ringing, τότε το επαναμεταδίδει προς τα πίσω (αντίστροφη κατεύθυνση από το μήνυμα INVITE) μέχρι να φτάσει στο τελικό αποδέκτη που είναι ένας πράκτορας χρήστη ο οποίος είχε αποστείλει μήνυμα INVITE.

```
                case RINGING_180:
                        cout << "PROXY " << PRINTADDR(addr())  << " received 180 RINGING
message at " << CURR_TIME << endl;
                        if(STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str())  == addr()) {
                                hdripret->daddr() = hdrsip->invite_;
                                hdripret->dport() = SIP_PORT;
                                hdripret->saddr() = addr();
                                hdripret->sport() = port();
                                hdrsipret->contact_ = hdrsip->contact_;
                                hdrsipret->from_  = hdrsip->from_;
                                hdrsipret->to_    = hdrsip->to_;
                                hdrsipret->invite_= hdrsip->invite_;
                                hdrsipret->type_  = RINGING_180;
                                hdrsipret->index_ = hdrsip->index_;
                                send(pktret,0);
                        }
                        else {
                hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str());
                                hdripret->dport() = SIP_PORT;
                                hdripret->saddr() = addr();
                                hdripret->sport() = port();
                                hdrsipret->contact_ = hdrsip->contact_;
                                hdrsipret->from_  = hdrsip->from_;
                                hdrsipret->to_    = hdrsip->to_;
                                hdrsipret->invite_= hdrsip->invite_;
                                hdrsipret->type_  = RINGING_180;
                                hdrsipret->index_ = hdrsip->index_;
                                send(pktret,0);
                        }
                        Packet::free(pkt);
```

```
                    break;
```

Αν το μήνυμα που θα λάβει ένας εξυπηρετητής είναι τύπου 302 Moved Temporarily, τότε στέλνει προς τον αποστολέα το μήνυμα ACK και επίσης επαναμεταδίδει το αρχικό μήνυμα INVITE προς τον πληρεξούσιο εξυπηρετητή της νέας περιοχής μέσα στην οποία βρίσκεται ο προς αναζήτηση πράκτορας χρήστη.

```
                case MOVED_302:
                        if(STR2ADDR((GETPROXY(PRINTADDR(hdrsip->contact_))).c_str()) != addr()){
                                cout << "PROXY " << PRINTADDR(addr())  << " received a 302
MOVED TEMPORARILY message at " << CURR_TIME << endl;
                         hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->contact_))).c_str());
                                hdripret->dport() = SIP_PORT;
                                hdripret->saddr() = addr();
                                hdripret->sport() = port();
                                hdrsipret->contact_ = hdrsip->invite_;
                                hdrsipret->from_  = hdrsip->from_;
                                hdrsipret->to_    = hdrsip->to_;
                                hdrsipret->invite_= hdrsip->contact_;
                                hdrsipret->type_  = INVITE;
                                hdrsipret->index_ = hdrsip->index_;
                                send(pktret,0);
                                //send ACK to HOME PROXY
                                hdripret2->daddr() = hdrip->saddr();
                                hdripret2->dport() = SIP_PORT;
                                hdripret2->saddr() = addr();
                                hdripret2->sport() = port();
                                hdrsipret2->contact_ = hdrsip->invite_;
                                hdrsipret2->type_  = ACK;
                                hdrsipret2->cseq_  = MOVED_302;
                                send(pktret2,0);
                        }
                        else { //the callee is inside this PROXY's region
                                hdripret->daddr() = hdrsip->to_;
                                hdripret->dport() = SIP_PORT;
                                hdripret->saddr() = addr();
                                hdripret->sport() = port();
                                hdrsipret->contact_ = hdrsip->invite_;
                                hdrsipret->from_  = hdrsip->from_;  //caller's SIP URI
                                hdrsipret->to_    = hdrsip->to_;       //callee's SIP URI
                                hdrsipret->invite_= hdrsip->contact_;
                                hdrsipret->type_  = INVITE;
                                hdrsipret->index_ = hdrsip->index_;
                                send(pktret,0);          //send the packet through the routing agent
                        }
                        Packet::free(pkt);
                        break;//preparing to send the new packet MOVED to caller's Proxy Server
```

Αν το μήνυμα που θα λάβει ένας εξυπηρετητής είναι τύπου 200 OK, ο εξυπηρετητής εκτελεί διαφορετικές ενέργειες αναλόγως με το ποια τιμή έχει το πεδίο CSeq, δηλαδή

αναλόγως με το ποιο μήνυμα αίτησης πυροδότησε το συγκεκριμένο μήνυμα 200 OK. Έτσι διακρίνουμε 3 διαφορετικές περιπτώσεις:

1) REGISTER

Το μήνυμα 200 OK είναι απόκριση σε μήνυμα REGISTER που έλαβε ένας πληρεξούσιος εξυπηρετητής από κάποιο πράκτορα χρήστη που άλλαξε περιοχή και έστειλε ενημέρωση στον οικείο εξυπηρετητή εγγραφών του. Ο πληρεξούσιος εξυπηρετητής που λαμβάνει το μήνυμα 200 OK, το προωθεί προς τον πράκτορα χρήστη στον οποίο απευθύνεται.

```
case OK_200:
                        switch(hdrsip->cseq_) {
                                case REGISTER:
                                        cout << "PROXY " << PRINTADDR(addr())  << " received a
200 OK (REG) message at " << CURR_TIME << endl;
                                        hdripret->daddr() = hdrsip->contact_;
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->type_  = OK_200;
                                        hdrsipret->cseq_  = REGISTER;
                                        hdrsipret->contact_ = 1;//flag in order for the USER
                                        //AGENT to understand that the received OK comes from
                                        //its HOME PROXY
                                        send(pktret,0);
                                        Packet::free(pkt);
                                        break;
```

2) INVITE

Το μήνυμα 200 OK μπορεί να είναι απόκριση σε μήνυμα INVITE οπότε ο πληρεξούσιος εξυπηρετητής που λαμβάνει το μήνυμα 200 OK, το προωθεί προς τον πράκτορα χρήστη στον οποίο απευθύνεται.

```
                case INVITE:
                        cout << "PROXY " << PRINTADDR(addr())  << " received a 200 OK (INVITE)
message at " << CURR_TIME << endl;
                         //Proxy must check if the caller belongs to its region and if it is inside
                        if (STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str()) == addr()) {
                        //Caller belongs to this Proxy
                                        hdripret->daddr() = hdrsip->invite_;
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;
```

```
                                        hdrsipret->to_    = hdrsip->to_;
                                        hdrsipret->invite_= hdrsip->invite_;
                                        hdrsipret->type_  = OK_200;
                                        hdrsipret->cseq_  = INVITE;
                                        hdrsipret->index_ = hdrsip->index_;
                                        send(pktret,0);
                                }
                        else {  //The caller does not belong to Proxy's region
```

```
                                //Send the new packet to the Proxy Agent of the caller
                        hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str());
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;
                                        hdrsipret->invite_= hdrsip->invite_;
                                        hdrsipret->type_  = OK_200;
                                        hdrsipret->cseq_  = INVITE;
                                        hdrsipret->index_ = hdrsip->index_;
                                        send(pktret,0);
                                }
                                Packet::free(pkt);
                                break;
```

3) MESSAGE

Το μήνυμα 200 OK μπορεί να είναι επίσης απόκριση σε μήνυμα INVITE οπότε ο πληρεξούσιος εξυπηρετητής που λαμβάνει το μήνυμα 200 OK, το προωθεί προς τον πράκτορα χρήστη στον οποίο απευθύνεται (χωρίς να εμπλέκεται στη διαδρομή κάποιος άλλος πληρεξούσιος εξυπηρετητής).

```
case MESSAGE:
// forward the message OK_200 to the instant message sender
cout<<"PROXY " << PRINTADDR(addr())  << " received 200 OK message at " << CURR_TIME << endl;
                                        hdripret->daddr() = hdrsip->invite_;
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = SIP_PORT;
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;
                                        hdrsipret->type_  = hdrsip->type_;
                                        hdrsipret->cseq_  = hdrsip->cseq_;
                                        send(pktret,0);
                                        Packet::free(pkt);
                                        break;
```

Αν το μήνυμα που θα λάβει ένας εξυπηρετητής είναι τύπου ACK, τότε το επαναμεταδίδει προς τον πληρεξούσιο εξυπηρετητή της περιοχής που βρίσκεται κάποιος πράκτορας

χρήστη στον οποίο απευθύνεται το μήνυμα ACK. Στη περίπτωση που το ACK είναι απόκριση σε μήνυμα 302 Moved Temporarily, ο πληρεξούσιος εξυπηρετητής που λαμβάνει ACK, δεν εκτελεί καμία ενέργεια.

```
case ACK:
cout<< "PROXY " << PRINTADDR(addr())  << " received an ACK message at " << CURR_TIME << endl;
                        switch(hdrsip->cseq_) {
                                case INVITE:
                                if(STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str()) == addr()){
                                                hdripret->daddr() = hdrsip->invite_;
                                                hdripret->dport() = SIP_PORT;
                                                hdripret->saddr() = addr();
                                                hdripret->sport() = port();
                                                hdrsipret->contact_ = hdrsip->contact_;
                                                hdrsipret->from_  = hdrsip->from_;
                                                hdrsipret->to_    = hdrsip->to_;
                                                hdrsipret->invite_= hdrsip->invite_;
                                                hdrsipret->index_ = hdrsip->index_;
                                                hdrsipret->type_  = ACK;
                                                send(pktret,0);
                                        }
                                        else {
                                hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str());
                                                hdripret->dport() = SIP_PORT;
                                                hdripret->saddr() = addr();
                                                hdripret->sport() = port();
                                                hdrsipret->contact_ = hdrsip->contact_;
                                                hdrsipret->from_  = hdrsip->from_;
                                                hdrsipret->to_    = hdrsip->to_;
                                                hdrsipret->invite_= hdrsip->invite_;
                                                hdrsipret->index_ = hdrsip->index_;
                                                hdrsipret->type_  = ACK;
                                                hdrsipret->cseq_  = INVITE;
                                                send(pktret,0);
                                        }
                                        break;
                                case MOVED_302:
                                        Packet::free(pktret);
                                        break;
```

Στη περίπτωση που ο πληρεξούσιος εξυπηρετητής λάβει μήνυμα MESSAGE, τότε το επαναμεταδίδει προς τον πράκτορα χρήστη στον οποίο απευθύνεται το μήνυμα (χωρίς να εμπλέκεται στη διαδρομή κάποιος άλλος πληρεξούσιος εξυπηρετητής).

```
case MESSAGE:
                        data=pkt->accessdata();
                        length=pkt->datalen();
                        pktret3 = allocpkt(length);
                        hdripret3 = hdr_ip::access(pktret3);
                        hdrsipret3 = hdr_sip::access(pktret3);
```

```
                        hdrcmnret3 = hdr_cmn::access(pktret3);
                        data2=pktret3->accessdata();
                        // IP Header
                        hdripret3->saddr() = addr();
                        hdripret3->sport() = SIP_PORT;
                        hdripret3->daddr() = hdrsip->invite_;
                        hdripret3->dport() = SIP_PORT;
                        // SIP Header

                        hdrsipret3->contact_ = hdrsip->contact_;//Caller's IP Address
                        hdrsipret3->from_=hdrsip->from_;//Caller's SIP URI | E2E USER AGENTS
                        hdrsipret3->to_= hdrsip->to_; //Callee's SIP URI | MUST BE CONNECTED
                        hdrsipret3->type_ = MESSAGE;          //MESSAGE message
                        memcpy (data2, (char*)data, length);
                        hdrcmnret3->size() = hdrcmn->size();
                        cout <<"PROXY " << PRINTADDR(addr()) <<" has received the
message.(forwarding...) at " << CURR_TIME << endl;
                        send(pktret3,0);
                        Packet::free(pkt);
                        Packet::free(pktret);
                        Packet::free(pktret2);
                        break;
```

Ένας εξυπηρετητής που λαμβάνει μήνυμα 100 Trying δεν εκτελεί καμία ενέργεια.

```
case TRYING_100:
                        cout<<"PROXY "<<PRINTADDR(addr())<<" received 100 TRYING message at
"<<CURR_TIME<<endl;
                    Packet::free(pkt);
                    Packet::free(pktret);
                    Packet::free(pktret2);
                    break;
```

Καταλήγοντας, ένας εξυπηρετητής που λαμβάνει το μήνυμα 404 Not Found, το επαναμεταδίδει προς τον πράκτορα χρήστη ο οποίος είχε ζητήσει να επικοινωνήσει με κάποιο άλλο πράκτορα χρήστη που πιθανώς να έχει απενεργοποιημένη τη συσκευή του.

```
case NOT_FOUND_404:
                        cout << "PROXY "  << PRINTADDR(addr()) << " received 404 NOT FOUND
message at " << CURR_TIME << endl;
                        hdripret->daddr() = hdrsip->invite_;
                        hdripret->dport() = SIP_PORT;
                        hdripret->saddr() = addr();
                        hdripret->sport() = port();
                        hdrsipret->type_  = NOT_FOUND_404;
                        send(pktret,0);
                        Packet::free(pkt);
                        Packet::free(pktret2);
                        break;
```

3.2.3 Αρχείο methods.h

Η ανάγκη για την επεξεργασία των αρχείων που θα έχουν την πληροφορία για τις διευθύνσεις των περιοχών (domain) και τις συντεταγμένες των πληρεξούσιων εξυπηρετητών (Proxy Server) μας οδήγησαν στη δημιουργία μιας επιπλέον κλάσης η οποία θα περιέχει τις απαραίτητες μεθόδους για την επεξεργασία αυτή.

Μέσα στο αρχείο methods.h ορίζεται η δομή coordInfo η οποία θα περιέχει τις συντεταγμένες καθώς και η σειρά των συναρτήσεων που αναφέραμε παραπάνω και θα αναλυθούν πιο κάτω.

```
#include <template.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string.h>
#include <cmath>

/* the struct that will include the information for the closest proxy */
struct coordInfo {
                        double x;
                        double y;
                         int ip;
};

class AgentMethods {
 public:
       static AgentMethods& instance() { return (*instance_); }
       AgentMethods();
       /*****************************
          COMMONLY USED FUNCTIONS
            -- DEFINITIONS --
        *****************************/
       void crRegFile(int,int);
       string getDomain(string);
       void writeIP(string,int,string);
       string getProxy(string);
       string getVisProxyAdd(string);
       string getVisAdd(string);
       string findfreeIp(string);
       void getfreeIp(string);
      void insMovedAg(string,string);
      void freeIp(string);
      coordInfo* findCoord(double,double); // returns the coordinates of the closest proxy
      bool isProxy(string);
      void crCoordFile(double,double,int);
      void eraseCoordFile(int);
      void eraseRegFile(int);
      //*****************************
 protected:
            static AgentMethods* instance_;
};
```

3.2.4 Αρχείο methods.cc

Παρακάτω παρατίθεται τμηματικά το αρχείο methods.cc και οι συναρτήσεις που περιέχει.

Μέθοδος crRegFile(int domain,int addrNo)

Η μέθοδος αυτή κατασκευάζει το αρχείο με τις διαθέσιμες διευθύνσεις της κάθε περιοχής (domain) και του δίνει το όνομα RegFl_x, όπου x είναι ο αριθμός της περιοχής (domain).

```
/*********************************************************
It creates the registar files with addrNo free ips.
At first it has the ips(k.0.m) that are given to noone(x)
*********************************************************/
void AgentMethods::crRegFile(int domain,int addrNo)
{
 ostringstream hlpStream;
 string flNam,domString,fileLnString,iString;
// the file name is RegFil_<domain>
 hlpStream << domain;
 domString=hlpStream.str();
 flNam="RegFl_"+domString;

// it opens the file empty for reading and writing
 ofstream fl(flNam.c_str(),ios_base::trunc|ios_base::in|ios_base::out);
  for (int i=1;i<addrNo+1;i++)
          {
// empry the stream first
           hlpStream.str("");
           hlpStream << i;
           iString=hlpStream.str();
// make the string that will be put in the file
           fileLnString=domString+".0."+iString+";x\n";
// writes in the file
           fl.write(fileLnString.c_str(),fileLnString.length());
           }
// close the stream,thus the file
 fl.close();
}
```

Μέθοδος getVisProxyAdd(string homeIp)

Αναζητεί μέσα στο αρχείο του οικείου πληρεξούσιου εξυπηρετητή (Home Proxy) κάποιου πράκτορα χρήστη και βρίσκει τη διεύθυνση του ξένου πληρεξούσιου εξυπηρετητή (Visited Proxy) στην περιοχή του οποίου βρίσκεται ο πράκτορας χρήστη.

```
/*************************************************************************
It finds the proxy server of the assigned ip given the home ip address
in some registar file.If it finds x, it returns "closed"
*************************************************************************/
string AgentMethods::getVisProxyAdd(string homeIp)
{
 char buff[32];
 string lnString;

// open the relevant file of the home ip's registar

 string flNam="RegFl_"+getDomain(homeIp);
 ifstream regFl(flNam.c_str(),ios_base::in);
// it searches all the lines
 while (!regFl.getline(buff, 32).eof())
 {
  lnString.assign(buff);
// if it finds the ip
  if ( lnString.find(homeIp, 0 ) != string::npos )
  {
// it keeps the assigned ip in the visited registar
          lnString=lnString.substr(lnString.find(';',0)+1,lnString.length()-lnString.find(';',0)-1);
    break;
  }
}
 regFl.close();
//returns error if it didn't find a valid ip
 if (lnString.find('.', 0 ) == string::npos)
 return "not_found";
//else it returns the domain proxy's ip
 else
 return getProxy(lnString);
}
```

Μέθοδος getVisAdd(string homeIp)

Βρίσκει και επιστρέφει τη διεύθυνση IP που έχει ένας πράκτορας χρήστη που έχει κινηθεί σε άλλη περιοχή (domain) διαφορετική από αυτή του οικείου πληρεξούσιου εξυπηρετητή του (Home Proxy).

```
/*************************************************************************
It finds the foreign assigned ip given the home ip address
in some registar file.If it finds x, it returns "closed"
*************************************************************************/
string AgentMethods::getVisAdd(string homeIp)
{
 char buff[32];
 string lnString;
// open the relevant file of the home ip's registar
 string flNam="RegFl_"+getDomain(homeIp);
```

```
 ifstream regFl(flNam.c_str(),ios_base::in);
// it searches all the lines
 while (!regFl.getline(buff, 32).eof())
 {
  lnString.assign(buff);
// if it finds the ip
  if ( lnString.find(homeIp, 0 ) != string::npos )
  {
// it keeps the assigned ip in the visited registar
       lnString=lnString.substr(lnString.find(';',0)+1,lnString.length()-lnString.find(';',0)-1);
    break;
  }

}

 regFl.close();
//returns error if it didn't find a valid ip
 if (lnString.find('.', 0 ) == string::npos)
 return "not_found";
//else it returns the domain proxy's ip
 else
 return lnString;
}
```

Μέθοδος getProxy(string ip)

Η μέθοδος getProxy επιστρέφει τη διεύθυνση του πληρεξούσιου εξυπηρετητή (Proxy Server) στον οποίο ανήκει κάποιος πράκτορας χρήστη δεδομένης της διεύθυνσης του.

```
/*********************************************************
It gets the proxy's ip given the ip address,ex 2.0.9->2.0.0
*********************************************************/
string AgentMethods::getProxy(string ip)
{
        return getDomain(ip)+".0.0";
}
```

Μέθοδος findfreeIp(string proxyIp)

Η μέθοδος findfreeIp βρίσκει την πρώτη διαθέσιμη διεύθυνση κάποιας περιοχής.

```
/*************************************************************************
It finds the first free ip of the registar of a domain given the relevant
proxy's ip
*************************************************************************/
string AgentMethods::findfreeIp(string proxyIp)
{
```

```
 char buff[37];
 string lnString;
// open the relevant file of the home ip's registar
 string flNam="RegFl_"+getDomain(proxyIp);
 ifstream regFl(flNam.c_str(),ios_base::in|ios_base::out);
// it searches all the lines
 while (!regFl.getline(buff, 37).eof())
 {
  lnString.assign(buff);
// if it finds the first free ip,thus the ip with 'x' beside the ';'
  if ( lnString.find('x', 0 ) != string::npos )
  {
// it keeps the free ip

            lnString=lnString.substr(0,lnString.find(';',0));

    break;
  }
}
 regFl.close();
 return lnString;
}
```

Μέθοδος getfreeIp(string newIp)

Η getfreeIp δίνει μια διαθέσιμη διεύθυνση κάποιας περιοχής φροντίζοντας να ενημερώσει και το αρχείο (βάση δεδομένων) της νέας περιοχής.

```
/************************************************************
It gets the free ip given updating the relevant registar file
************************************************************/
void AgentMethods::getfreeIp(string newIp)
{
 int line;
 int lncount=1;
 char buff[32];
 string lnString;
// open the relevant file of the home ip's registar
 string flNam="RegFl_"+getDomain(newIp);
 ifstream regFl(flNam.c_str(),ios_base::in);
// it searches all the lines
 while (!regFl.getline(buff, 32).eof())
 {
  lnString.assign(buff);
// if it finds the first free ip,thus the ip with 'x' beside the ';'
  if ( lnString.find(newIp, 0 ) != string::npos )
            line=lncount;
  lncount++;
 }
 regFl.close();
 writeIP(flNam,line,newIp);
}
```

Μέθοδος getDomain(string IP)

Επιστρέφει τη διεύθυνση του πληρεξούσιου εξυπηρετητή (Proxy Server) στου οποίου τη περιοχή βρίσκεται ένας πράκτορας χρήστη, δεδομένης της διεύθυνσης που αυτός έχει.

```
/*********************************************************************
It gets the domain number of an ip address. It's onle defined to be used
in the getVisProxyAdd&findfreeIp functions
*********************************************************************/
string AgentMethods::getDomain(string Ip)
{
 string dom;
// keep the substring until the first '.'
 dom=Ip.substr(0,Ip.find_first_of('.',0));
 return dom;
}
```

Μέθοδος insMovedAg(string homeIP,string nIP)

Ενημερώνει το αρχείο (βάση δεδομένων) του οικείου καταχωρητή εγγραφών (Home Registrar), στον οποίο ανήκει ο πράκτορας χρήστη που άλλαξε διεύθυνση, με τη νέα του διεύθυνση IP.

```
/*******************************************************
It informs the home registar file about the new ip address
assigned from the visited registar
*******************************************************/
void AgentMethods::insMovedAg(string homeIp,string nIp)
{
  int line;
  int lncount=1;
  char buff[32];
  string lnString;
 // open the relevant file of the home ip's registar
  string flNam="RegFl_"+getDomain(homeIp);
  ifstream regFl(flNam.c_str(),ios_base::in);
 // it searches all the lines
  while (!regFl.getline(buff, 32).eof())
  {
   lnString.assign(buff);
 // if it finds the first free ip,thus the ip with 'x' beside the ';'
   if ( lnString.find(homeIp, 0 ) != string::npos )
                        line=lncount;
```

```
   lncount++;
  }
 regFl.close();
 writeIP(flNam,line,nIp);
}
```

Μέθοδος freeIp(string hIP)

Η freeIp ελευθερώνει (διαγράφει) τη δεδομένη διεύθυνση IP από το αρχείο στο οποίο βρίσκεται.

```
/*****************************************
It frees one of the given ips in a reg file
*****************************************/
void AgentMethods::freeIp(string hIp)
{
 insMovedAg(hIp,"x");
}
```

Μέθοδος writeIP(string file,int lin,string IP)

Η μέθοδος writeIP γράφει μια διεύθυνση στην κατάλληλη θέση ενός αρχείου που καθορίζεται από τα ορίσματά της.

```
/*****************************************************************
It writes in a specified file, to the line told the ip given in
the position of assigned ip.It is only defined to be used by
getfreeIP&insMovedAg
*****************************************************************/
void AgentMethods::writeIP(string file,int lin,string ip)
{
  char c;
  int line=1;
  string leadString,leftString;
// first get the string that will be kept,following the written string
  ifstream iregFl(file.c_str(),ios_base::in);
// keep the leading string
            while (line<lin)
            {
             iregFl.get(c);
             leadString+=c;
             if (c=='\n') line++;
            }
            while (c!=';')
            {
```

```
            iregFl.get(c);
            leadString+=c;
           }
// ignore until you get to the next line
           while (c!='\n')
            iregFl.get(c);
  leftString+='\n';
// keep the left string
  while (iregFl.get(c))
   leftString+=c;
  iregFl.close();

// write to the positions what you want
  fstream oregFl(file.c_str(),ios_base::out|ios_base::out);
//write thea leading string
  oregFl.write(leadString.c_str(),leadString.length());
// write the ip
  oregFl.write(ip.c_str(),ip.length());
// write the string you have kept
  oregFl.write(leftString.c_str(),leftString.length());
  oregFl.close();
}
```

Μέθοδος isProxy(string IPstr)

Εξακριβώνει αν μια διεύθυνση είναι διεύθυνση πληρεξούσιου εξυπηρετητή ή απλού πράκτορα χρήστη.

```
/*****************************************
Function that decides if the given ip address
belongs to a Proxy Agent
*****************************************/
bool AgentMethods::isProxy(string ipstr)
{
        int last = ipstr.size();
        int loc  = ipstr.find_last_of(".");

        if(ipstr.substr(loc+1,last-(loc+1)) == "0")
                return true;
        else false;
}
```

Μέθοδος crCoordFile (double x,double y,int domain)

Κατασκευάζει το αρχείο CoordFl εισάγοντας τις συντεταγμένες του πληρεξούσιου εξυπηρετητή σε αυτό.

```
/*********************************************************
 Given the coordinates of the proxy an its ip it adds this
information in the coordFl file
*********************************************************/
void AgentMethods::crCoordFile(double x,double y,int domain)
{
 ostringstream hlpStream;
 string domString,xString,yString, fileLnString;
 string flNam="coordFl";
 // the file name is RegFil_<domain>
 hlpStream << domain;
 domString=hlpStream.str()+".0.0";
 // empty the stream
 hlpStream.str("");
 hlpStream << x;
 xString=hlpStream.str();

  // empty the stream

 hlpStream.str("");
 hlpStream << y;
 yString=hlpStream.str();
 // the line to be inserted in the file
 fileLnString=xString+','+yString+';'+domString+'\n';
 // open the Coord File for appending
 ofstream fl(flNam.c_str(),ios_base::app|ios_base::out);
 // writes in the file
 fl.write(fileLnString.c_str(),fileLnString.length());
 // close the stream,thus the file
 fl.close();
}
```

Μέθοδος findCoord(double x,double y)

Βρίσκει τις συντεταγμένες και τη διεύθυνση του πληρεξούσιου εξυπηρετητή που βρίσκεται πιο κοντά στις συντεταγμένες που δίνονται ως όρισματά της.

```
/*********************************************************
 Given the coordinates of the mobile node it looks in
the coordFl file and retrns the ip of the closest proxy
*********************************************************/
coordInfo* AgentMethods::findCoord(double x,double y)
{
// the output
 coordInfo *out;
 coordInfo coinfo;
 out=&coinfo;

 char buff[100];
 string lnString,x0Str,y0Str;
 string flNam="coordFl";
 // the minimum distance is huge in the beginning
 double minDist=1000000;
```

```
 double x0,y0;
// open the file for reading
 ifstream coordFl(flNam.c_str(),ios_base::in);
 while (!coordFl.eof())
 {
 // read the nect line of the file
  coordFl.getline(buff,100);
  lnString.assign(buff);
  // get the coordinates
  x0Str=lnString.substr(0,lnString.find(',',0));
  x0=atof(x0Str.c_str());
  y0Str=lnString.substr(lnString.find(',',0)+1,lnString.find(';',0)-lnString.find(',',0)-1);
  y0=atof(y0Str.c_str());
  // if the new proxy is closer
  if (minDist>(x-x0)*(x-x0)+(y-y0)*(y-y0))
        {
         // udate the closest proxy's info and the minimum distance
         minDist=(x-x0)*(x-x0)+(y-y0)*(y-y0);

            out->x=x0;

            out->y=y0;
            out->ip=   Address::instance().str2addr((lnString.substr(lnString.find(';',0)+1,lnString.length()-
lnString.find(';',0)-1)).c_str());
            }
 }
// close the stream,thus the file
 coordFl.close();
 return out;
};
```

Μέθοδος eraseCoordFile(int domain)

Η μέθοδος αυτή σβήνει το RegFl (αρχείο – βάση δεδομένων) της περιοχής που δέχεται ως όρισμα. Καλείται στο destructor της κλάσης, όταν δε χρειάζεται πια το RegFl αυτό.

```
/*************************************************************
 Given the coordinates of the proxy it erases its record from
 the Coord File
*************************************************************/
 void AgentMethods::eraseCoordFile(int domain)
{
 char buff[100];
 ostringstream hlpStream;
 string domString,lnString,x0Str,y0Str;
 string outStr="";
 string flNam="coordFl";
 // the file name is RegFil_<domain>
 hlpStream << domain;
 domString=hlpStream.str()+".0.0";
 // the minimum distance is huge in the beginning
// open the file for reading
 ifstream coordFl(flNam.c_str(),ios_base::in);
```

```
 while (!coordFl.getline(buff,100).eof())
 {
  lnString.assign(buff);
  // get the coordinates
 // x0Str=lnString.substr(0,lnString.find(',',0));
 // y0Str=lnString.substr(lnString.find(',',0)+1,lnString.find(';',0)-lnString.find(',',0)-1);
  if (!(lnString.substr(lnString.find(';',0)+1,lnString.length()-lnString.find(';',0)-1)==domString))
   outStr+=lnString+'\n';
 }
  coordFl.close();
  // open the file for output to the positions what you want
  ofstream oregFl(flNam.c_str(),ios_base::out);
  //write the string
  oregFl.write(outStr.c_str(),outStr.length());
  oregFl.close();
}
```

Μέθοδος eraseRegFile(int domain)

Τέλος η eraseRegFile σβήνει την εγγραφή που αντιστοιχεί στη περιοχή που δέχεται ως όρισμα από το coordFl. Καλείται στο destructor της κλάσης, όταν δε χρειάζεται πια η εγγραφή αυτή.

```
/************************************************
 Erase the Reg File of the proxy given its domain
************************************************/
void AgentMethods::eraseRegFile(int domain)
{
 ostringstream hlpStream;
 string flNam,domString;
 // the file name is RegFil_<domain>
 hlpStream << domain;
 domString=hlpStream.str();
 flNam="RegFl_"+domString;
 remove(flNam.c_str());
}
```

3.2.5 Αρχείο ns-sip.tcl

Στο αρχείο αυτό γίνεται η αρχικοποίηση των αντικειμένων του πληρεξούσιου εξυπηρετητή και του πράκτορα χρήστη μέσω Tcl. Όπως φαίνεται, κατά την αρχικοποίηση του αντικειμένου του πληρεξούσιου εξυπηρετητή – εξυπηρετητή εγγραφών, καλούνται οι εντολές node και initialize που είχαμε προαναφέρει κατά την παρουσίαση της μεθόδου command της κλάσης του πληρεξούσιου εξυπηρετητή –

εξυπηρετητή εγγραφών. Επίσης κατά την αρχικοποίηση του αντικειμένου του πράκτορα χρήστη καλείται η εντολή node.

```
#Sip Proxy Agent initialization
Agent/SipPA instproc init { node args } {
        eval $self next $args

        $self node $node                    ;# pointer to MobileNode
        $self initialize [$node node-addr]  ;# file initialization
}

#Sip User Agent initialization
Agent/SipUA instproc init { node args } {
        eval $self next $args
        $self node $node  ;# pointer to MobileNode
}
```

Στη συνέχεια παρουσιάζονται τα αρχεία που τροποποιήσαμε κατά τη διάρκεια της διπλωματικής αυτής δίνοντας τα απαραίτητα επεξηγηματικά σχόλια ιδιαίτερα στα σημεία όπου έγιναν οι τροποποιήσεις.

3.2.6 Αρχείο dsdv.h

Στο αρχείο αυτό που περιέχει τις δηλώσεις των συναρτήσεων και των μεταβλητών της κλάσης του DSDV routing agent προσθέσαμε τη δήλωση μιας νέας συνάρτησης που εισαγάγαμε στο κώδικα του routing agent.

```
  //**********************
 void sipUpdate(int dst);
  //**********************
```

3.2.7 Αρχείο dsdv.cc

Στο αρχείο αυτό που περιέχει το πλήρη κώδικα του DSDV routing agent, προσθέσαμε τη συνάρτηση sipUpdate η οποία καλείται μέσα από το αρχείο sip.cc όταν ο πράκτορας χρήστη διαπιστώνει ότι ο χρήστης εισέρχεται σε μια νέα περιοχή. Η συνάρτηση αυτή χρησιμοποιείται για να αλλάξουμε τη διεύθυνση IP του routing agent του πράκτορα χρήστη. Όταν ο εξυπηρετητής λάβει το μήνυμα ενημέρωσης τότε εισάγει μια νέα καταχώρηση στο πίνακα δρομολόγησής του έτσι ώστε όταν ο πληρεξούσιος

εξυπηρετητής λάβει ένα μήνυμα που απευθύνεται προς το νεοεισελθόντα πράκτορα χρήστη να μπορεί να το δρομολογήσει άμεσα.

```
/*****************************************
 Function that changes variables myaddr_
here_.addr_
*****************************************/
void DSDV_Agent::sipUpdate(int dst)
{

  rtable_ent *prte;
  rtable_ent  rte;

  prte = table_->GetEntry(dst);
  rte = *prte;
  rte.dst = myaddr_;
  rte.hop = myaddr_;
  table_->AddEntry(rte);


  addr() = myaddr_;

}
/*************************************/
```

Στο ίδιο αρχείο, εισάγουμε στη συνάρτηση command του DSDV_Agent τις παρακάτω γραμμές κώδικα, έτσι ώστε να μπορούμε να καλούμε την εντολή sip με όρισμα τη νέα διεύθυνση του πράκτορα χρήστη, μέσα από το αρχείο sip.cc και να αλλάζουμε τη διεύθυνση IP του routing agent.

```
int
DSDV_Agent::command (int argc, const char *const *argv)
{
    ...

        else if (argc == 3)   {
            //*************************************************
            if (strcasecmp (argv[1], "sip")  == 0) {
                        sipUpdate(atoi(argv[2]));
                        return TCL_OK;
            }
             //*************************************************
```

3.2.8 Αρχείο arp.cc

Στο αρχείο αυτό που περιέχει τον κώδικα που υλοποιεί το πρωτόκολλο ARP, κάναμε μια μικρή τροποποίηση για να δώσουμε μεγαλύτερη προτεραιότητα στα πακέτα σηματοδοσίας του πρωτοκόλλου SIP. Μέχρι τώρα το πρωτόκολλο ARP λειτουργούσε με

τον εξής τρόπο στο περιβάλλον εξομοίωσης NS: Όταν σε ένα κόμβο Α φτάσει ένα πακέτο για το κόμβο Β και ο κόμβος Α δε γνωρίζει τη διεύθυνση MAC του προορισμού (κόμβος Β) παρά μόνο τη διεύθυνση IP του προορισμού, τότε φυλάει το προς αποστολή πακέτο και αποστέλλει πακέτο ARP request (broadcast) με σκοπό να μάθει τη διεύθυνση MAC του κόμβου Β. Αν στο κόμβο Α φτάσει ακόμα ένα πακέτο που κατευθύνεται προς το κόμβο Β προτού φτάσει το πακέτο ARP reply, τότε ο κόμβος Α πετάει το πακέτο που είχε φυλάξει προηγουμένως, φυλάει το νέο πακέτο προς αποστολή και ξαναστέλλει ARP request. Η διαδικασία αυτή είναι επιζήμια για μας διότι υπάρχει ενδεχόμενο να χαθεί κάποιο πακέτο σηματοδοσίας. Οπότε προσθέσαμε τις παρακάτω εντολές για να κρατάμε τα πακέτα SIP έτσι ώστε να μην υπάρχει ενδεχόμενο να τερματιστεί απρόσμενα η ροή πακέτων σηματοδοσίας.

```
int
ARPTable::arpresolve(nsaddr_t dst, Packet *p, LL *ll)
{
  …
        if(llinfo->count_ >= ARP_MAX_REQUEST_COUNT)
         {
           …
            //for SIP protocol
            hdr_cmn* hdr = HDR_CMN(t);
            if(hdr->ptype_ == PT_SIP) {
//                      printf("TRYING TO DROP PT_SIP at %.5f\n",Scheduler::instance().clock());
                        return 0;
            }
           …
          }

        /*
         * If we have SIP signalling and traffic running on a single node we don't want to lose the
         * first SIP packet because of arp request failure, so we keep the first SIP packet
          * and drop the rest of the packets
         */
        hdr_cmn* hdrcmn;
        llinfo->count_++;
        if(llinfo->hold_) { //drop pending packet if it is not SIP packet
            hdrcmn = HDR_CMN(llinfo->hold_);
            if(hdrcmn->ptype_ == PT_SIP) {/* do nothing at all -- hold the first SIP packet */}
            else {
                        drop(llinfo->hold_, DROP_IFQ_ARP_FULL);
//                     printf("Dropping packet from node %d at %.5f\n",node_->
address(),Scheduler::instance().clock());
            }
        }
        hdr_cmn* hdrcmn2 = HDR_CMN(p);
        if(hdrcmn2->ptype_ == PT_SIP)
            llinfo->hold_ = p;
        else {
            if((hdrcmn) && (hdrcmn->ptype_ == PT_SIP)) {
```

```
                drop(p, DROP_IFQ_ARP_FULL);
        }
        else {
                llinfo->hold_ = p;
        }
    }
     …
```

3.2.9 Αρχείο priqueue.cc

Στο αρχείο αυτό προσδιορίζεται η προτεραιότητα των πακέτων που εισέρχονται στην ουρά IFq. Λόγω του γεγονότος ότι τα πακέτα σηματοδοσίας δε πρέπει να καθυστερούν αρκετό χρονικό διάστημα μέσα στην ουρά αυτή γιατί θα αργούν να γίνονται οι ενημερώσεις (όπως π.χ. στη περίπτωση διαπομπής θα αργήσει να γίνει η ενημέρωση των εμπλεκομένων οντοτήτων και θα χαθούν πολλά πακέτα άσκοπα). Οπότε με την προσθήκη μιας εντολής στο αρχείο αυτό όπως φαίνεται παρακάτω, δίνουμε υψηλή προτεραιότητα στα πακέτα σηματοδοσίας.

```
void
PriQueue::recv(Packet *p, Handler *h)
{
        struct hdr_cmn *ch = HDR_CMN(p);

        if(Prefer_Routing_Protocols) {
              switch(ch->ptype()) {
            …
            case PT_SIP:  // high priority for signalling packets (sip)
                    recvHighPriority(p, h);
                    break;
             …
```

3.2.10 Αρχείο packet.h

Στο αρχείο αυτό ορίζουμε το πακέτο sip (PT_SIP). Κάθε μήνυμα σηματοδοσίας που αποστέλνεται τόσο από τους πράκτορες χρήστη όσο και από τους εξυπηρετητές θα είναι τύπου PT_SIP. Αυτό διευκολύνει και την αναγνώριση των πακέτων σηματοδοσίας μέσα στα αρχεία trace.

```
enum packet_t {
            …

            //SIP packet
            PT_SIP,

            PT_NTYPE // This MUST be the LAST one
```

```
};
class p_info {
public:
        p_info() {
                ...
                // SIP packet
                        name_[PT_SIP]= "sip";
```

3.2.11 Αρχείο ns-mobilenode.tcl

Στο αρχείο αυτό δημιουργούνται (όταν τοποθετηθεί η εντολή $ns node-config –SIP ON σε κάποιο αρχείο Tcl που περιέχει κάποιο σενάριο κίνησης) δύο νέοι κόμβοι (Nodes) που ανήκουν στη κλάση Node/MobileNode. Ο πρώτος κόμβος αποτελεί το σταθμό βάσης πάνω στον οποίο προσαρτάται ένας πληρεξούσιος εξυπηρετητής (SipPA) που «ακούει» στη θύρα 5060. Ο δεύτερος κόμβος αποτελεί τον κινητό σταθμό πάνω στον οποίο προσαρτάται ένας πράκτορας χρήστη (SipUA) που «ακούει» και αυτός στη θύρα 5060.

```
#**********************************************************************
#                       SIP CODE
#**********************************************************************
Node/MobileNode instproc makemip-NewSipPA {} {
        $self instvar regagent_ agents_ id_

        set dmux [new Classifier/Port/Reserve]
        $dmux set mask_ 0x7fffffff
        $dmux set shift_ 0
        $self install-demux $dmux

        set regagent_ [new Agent/SipPA $self]
        $self attach $regagent_ [Node/MobileNode set SIP_PORT];# attach proxy agent on port 5060
}

Node/MobileNode instproc makemip-NewSipUA {} {
        $self instvar regagent_

        set dmux [new Classifier/Port/Reserve]
        $dmux set mask_ 0x7fffffff
        $dmux set shift_ 0
        $self install-demux $dmux

        set regagent_ [new Agent/SipUA $self]
        $self attach $regagent_ [Node/MobileNode set SIP_PORT]  ;# attach user agent in port 5060
        $regagent_ set mask_ [AddrParams NodeMask 1]
        $regagent_ set shift_ [AddrParams NodeShift 1]
#       $regagent_ set dst_addr_ [expr (~0) << [AddrParams NodeShift 1]]
#       $regagent_ set dst_port_ 0
        $regagent_ node $self
}
```

Στο ίδιο αρχείο τοποθετούμε και τις παρακάτω συναρτήσεις. Η πρώτη συνάρτηση καλείται μέσα από το αρχείο ns-lib.tcl. Η συνάρτηση αυτή καλεί την εντολή ragent της κλάσης SipPA για να επιστρέψει ένα δείκτη πάνω στο routing agent.

```
#Bind routing agent and sip agent if existing basestation address setting
Node/MobileNode instproc sip-call {ragent} {
        $self instvar regagent_
        if [info exists regagent_] {
            $regagent_ ragent $ragent   ;# command ragent in SipPAs
        }
}
```

Η παρακάτω συνάρτηση καλείται μέσα από αρχεία Tcl που περιέχουν σενάρια κίνησης, με σκοπό να επιστρέψει ένα instance του sip agent, έτσι ώστε ο χρήστης να μπορεί να χρησιμοποιήσει κάποιες εντολές των sip agents (αυτές που βρίσκονται στις εκάστοτε συναρτήσεις command.

```
Node/MobileNode instproc sipag {} {
        return [$self set regagent_]        ;# returns an instance to the sip agent
}
```

Οι επόμενες δύο συναρτήσεις ορίζουν και επιστρέφουν αντίστοιχα ένα instance του routing agent ο οποίος είναι προσαρτημένος στο κάθε κόμβο.

```
Node/MobileNode instproc set_ragent {ragent} {
        $self instvar my_ragent_

        $self set my_ragent_ $ragent
}
Node/MobileNode instproc get_ragent {} {
        return [$self set my_ragent_]
}
```

Η τελευταία αυτή συνάρτηση καλείται μέσα από το αρχείο sip.cc όταν ένας πράκτορας χρήστη διαπιστώνει αλλαγή περιοχής και αφότου έχει αλλάξει διεύθυνση IP. Με τη συνάρτηση αυτή, γίνεται μια εσωτερική αλλαγή μέσα στον κόμβο για να μπορεί να δέχεται μηνύματα στη νέα αυτή διεύθυνση IP.

```
#Add a new route from the new address to the dmux_ and delete route
#from the old address to the dmux_
Node/MobileNode instproc move-dmux {newaddr oldaddr} {
        $self instvar dmux_
        $self delete-route $oldaddr $dmux_
        $self add-route $newaddr $dmux_
}
#*********************************************************************
#                   end of SIP CODE
#*********************************************************************
```

3.2.12 Αρχείο ns-default.tcl

Στο αρχείο αυτό γίνονται οι αναγκαίες αρχικοποιήσεις των μεταβλητών που ορίζονται σε Tcl.

```
Node/MobileNode set SIP_PORT                    5060
#Sip packet size
Agent/SipUA set packetSize_ 48
Agent/SipPA set packetSize_ 48
```

3.2.13 Αρχείο ns-packet.tcl

Στο αρχείο αυτό βάλαμε την πληροφορία σχετικά με το νέο πακέτο τύπου sip που θα χρησιμοποιούν οι δύο οντότητες του SIP.

```
foreach prot {
        …
        #for SIP application
        #-------------------
        Sip
        #-------------------
        …
```

3.2.14 Αρχείο ns-lib.tcl

Στο αρχείο αυτό, εισάγουμε αρχικά τις παρακάτω γραμμές, έτσι ώστε ο χρήστης να μπορεί να χρησιμοποιήσει το Πρωτόκολλο Έναρξης Συνόδου, βάζοντας στο αρχείο Tcl που θα περιγράφει το σενάριο κίνησης, την εντολή $ns node-config –SIP ON.

```
…
source ns-sip.tcl
```

```
…
#******************************************************************************
Simulator instproc SIP {val} {$self set SIP_ $val}
#******************************************************************************
…
```

Με τη παρακάτω συνάρτηση ορίζονται οι μεταβλητές SipUA για κινητούς κόμβους και SipPA για σταθμούς βάσης, αναλόγως με το αν είναι ενεργοποιημένο το wiredRouting_ ή όχι.

```
Simulator instproc get-nodetype {} {
        …
        # Sip agents
        if { [Simulator set sip_] } {
            if { $val == "Base" && $wiredRouting_ == "ON" } {
                    set val SipPA
            }

            if { $val == "Base" && $wiredRouting_ == "OFF" } {

                    set val SipUA
            }
        }
        …
```

Αν είναι ενεργοποιημένη η μεταβλητή SIP_ (ο χρήστης έδωσε την εντολή $ns node-config –SIP ON) τότε τίθεται η μεταβλητή sip_ ίση με 1 αλλιώς τίθεται ίση με 0.

```
Simulator instproc node-config args {
        …
        $self instvar addressType_ … SIP_ …
        …
        # set SIP flag
        if { [info exists SIP_] && $SIP_ == "ON"} {
            Simulator set sip_  1

        } else {
            if { [info exists SIP_] } {
                    Simulator set sip_ 0
            }
        }
        …
```

Η εντολή $node sip-call $ragent ενεργοποιεί τη συνάρτηση sip-call που ορίζεται μέσα στο αρχείο ns-mobilenode.tcl και η οποία καλεί την εντολή ragent της κλάσης SipPA για να επιστρέψει ένα δείκτη πάνω στο routing agent έτσι ώστε να μπορούμε να καλέσουμε μέσα από τη κλάση αυτή, συναρτήσεις ή μεταβλητές του routing agent.

```
Simulator instproc create-wireless-node args {
        …
        if { [info exist wiredRouting_] && $wiredRouting_ == "ON" } {
            if { $routingAgent_ != "DSR" } {
                        $node mip-call $ragent
                        $node sip-call $ragent
            }
        }
        …
Simulator instproc create-dsdv-agent { node } {
        …
        # SIP protocol

        if [Simulator set sip_] {
            $ragent port-dmux [$node demux]

        } …
```

4. Περιγραφή Σεναρίων Κίνησης

4.1 Κλάσεις Κίνησης

Κατά τη διάρκεια της υλοποίησης των σεναρίων κίνησης χρησιμοποιήθηκαν 3 διαφορετικές κλάσεις κίνησης (Traffic Classes TC), έτσι ώστε να παρατηρήσουμε τη συμπεριφορά των δύο μηχανισμών κινητικότητας (SIP και Mobile IP) σε σχέση με το είδος της κίνησης.

Οι 3 κλάσεις κίνησης που χρησιμοποιήθηκαν είναι:

Α) TC1: Διαλογική επικοινωνία (conversational) σταθερού ρυθμού μετάδοσης (Constant Bit Rate – CBR) όπως για παράδειγμα η εφαρμογή Voice over IP (VoIP) η οποία έχει τα εξής χαρακτηριστικά:

- bi-directional link
- full duplex
- downlink bandwidth (DL) 16Kbps
- uplink bandwidth (UP) 16Kbps

Οι κυριότερες παράμετροι που ενδιαφέρουν τις κινήσεις που ανήκουν στην κλάση αυτή είναι:

- καθυστέρηση (delay),
- διακύμανση καθυστέρησης (jitter),
- απώλειες λόγω σφαλμάτων.

Οι εφαρμογές που ανήκουν στη κατηγορία αυτή είναι εφαρμογές πραγματικού χρόνου όπως τηλεφωνία και εξομοίωση κυκλώματος. Εφαρμογές της κατηγορίας αυτής έχουν αυστηρές απαιτήσεις στην καθυστέρηση καθώς και στη διακύμανση της καθυστέρησης μεταξύ διαδοχικών πακέτων αλλά είναι ανεκτικές ως προς κάποιο αριθμό σφαλμάτων.

Β) TC2: Επικοινωνία ροής (streaming) σταθερού ρυθμού μετάδοσης (CBR) όπως για παράδειγμα μια εφαρμογή video streaming η οποία έχει τα εξής χαρακτηριστικά:

- uni-directional link,
- semi-duplex,
- downlink bandwidth (DL) ~100Kbps,
- uplink bandwidth (UP) 2Kbps.

Οι κυριότερες παράμετροι που ενδιαφέρουν τις κινήσεις που ανήκουν στη κλάση αυτή είναι:

- καθυστέρηση (delay),
- διακύμανση καθυστέρησης (jitter),
- απώλειες λόγω σφαλμάτων.

Οι εφαρμογές που ανήκουν στη κατηγορία αυτή είναι εφαρμογές πραγματικού χρόνου όπως η τηλεδιάσκεψη (video-conference). Εφαρμογές της κατηγορίας αυτής έχουν όχι και τόσο αυστηρές απαιτήσεις στη καθυστέρηση όπως οι εφαρμογές της προηγούμενης κλάσης αλλά έχουν πολύ αυστηρούς περιορισμούς στη διακύμανση της καθυστέρησης μεταξύ διαδοχικών πακέτων. Επίσης είναι ανεκτικές ως προς κάποιον αριθμό σφαλμάτων. Αυτό που κυρίως ενδιαφέρει είναι η αναπαραγωγή μιας ροής video (χωρίς να ενοχλεί κάποια μικρή καθυστέρηση) χωρίς διακυμάνσεις της καθυστέρησης που θα αλλοιώνουν την κινούμενη εικόνα.

Γ) TC3: Επικοινωνία μεταβλητού ρυθμού μετάδοσης (Variable Bit Rate – VBR) όπως για παράδειγμα οι εφαρμογές WEB, FTP οι οποίες έχουν εκρηκτικά χαρακτηριστικά κίνησης και όχι αυστηρούς περιορισμούς στην καθυστέρηση και στη διακύμανση καθυστέρησης πακέτων. Από την άλλη πλευρά όμως, οι εφαρμογές αυτές έχουν μηδενική ανοχή σε σφάλματα, γιαυτό άλλωστε λειτουργούν πάνω απο το πρωτόκολλο TCP το οποίο αποτελεί μια αξιόπιστη υπηρεσία μεταφοράς.

4.2 Παράμετροι επεξεργασίας (metrics)

Οι κύριες παράμετροι που μας ενδιαφέρουν στην κατά την επεξεργασία των αποτελεσμάτων που έδωσαν οι εξομοιώσεις των σεναρίων στο Network Simulator v2.27 είναι οι ακόλουθες:

- Στιγμιαία και μέση καθυστέρηση από άκρο σε άκρο (instant and mean end-to-end delay in seconds)

Λαμβάνοντας δεδομένα από το trace file (χρονική στιγμή αποστολής και λήψης κάθε πακέτου) μπορούμε να υπολογίσουμε τόσο την στιγμιαία όσο και την μέση καθυστέρηση από άκρο σε ακρο.

- Στιγμιαία και μέση διακύμανση καθυστέρησης (instant and mean end-to-end delay variation – jitter in seconds)

  Λαμβάνοντας δεδομένα από το trace file όπως παραπάνω, μπορούμε να υπολογίσουμε τόσο τη στιγμιαία όσο και τη μέση διακύμανση καθυστέρηση από άκρο σε ακρο.

- Μέση επιβάρυνση λόγω σηματοδοσίας σε Kbps (mean signalling overhead in Kbps)

  Βρίσκοντας πόσα πακέτα έχουν ενθυλακωθεί, πολλαπλασιάζοντας με το 20 (bytes IP πακέτου ενθυλάκωσης) και διαιρώντας με το χρόνο της εξομοίωσης, βρίσκουμε τη μέση επιβάρυνση λόγω σηματοδοσίας σε bps.

- Μέση τιμή καθυστέρησης λόγω διαπομπής (mean handoff delay in seconds)

  Α] Περίπτωση Mobile IP: Ως χρόνο διαπομπής στο Mobile IP υπολογίσαμε το χρονικό διάστημα από τη στιγμή που ο κινητός κόμβος αντιλαμβάνεται ότι έχει εισέλθει σε μια νέα περιοχή μέχρι να ενημερώσει τον Πράκτορα Επισκεπτών και τον Πράκτορα Οικείων για την αλλαγή διεύθυνσης IP.

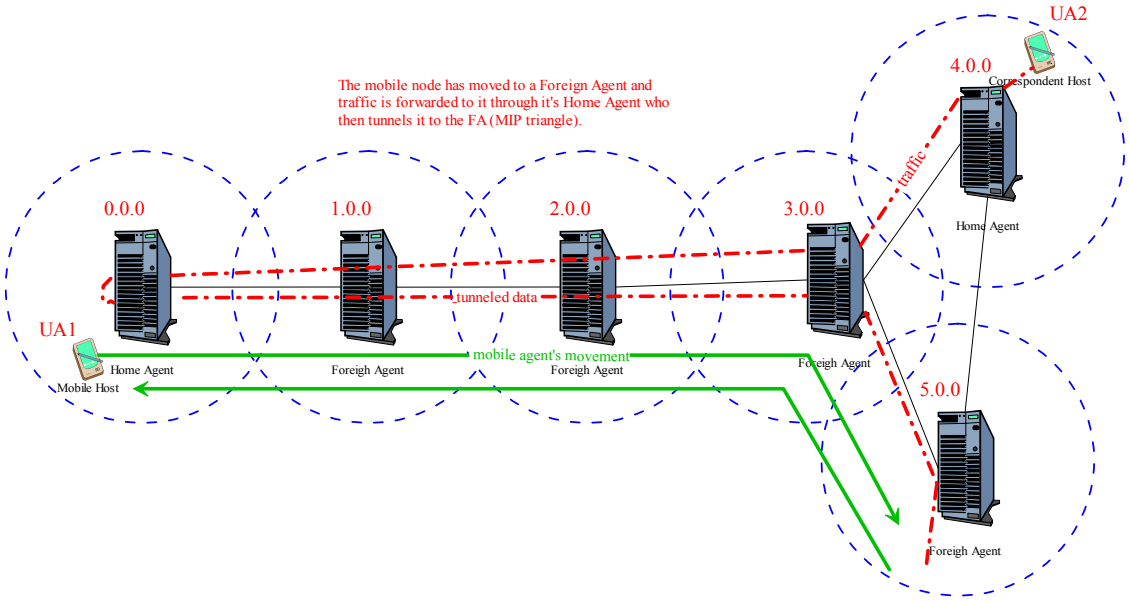
  Β] Περίπτωση SIP: Ως χρόνο διαπομπής στο SIP υπολογίσαμε το χρονικό διάστημα από τη στιγμή που ο κινητός κόμβος αντιλαμβάνεται ότι έχει εισέλθει σε μια νέα περιοχή μέχρι να ενημερώσει τον κόμβο με τον οποίο διατηρεί μια σύνδεση για την αλλαγή διεύθυνσης IP (δηλαδή μέχρι να λάβει ο correspondent host το μήνυμα (re-)INVITE).

  Η καθυστέρηση λόγω διαπομπής στο SIP εξαρτάται από την απόσταση των δύο τερματικών συσκευών που λαμβάνουν μέρος σε μια σύνδεση από άκρο σε άκρο, ενώ στο Mobile IP, εξαρτάται από την απόσταση του κόμβου – προορισμού από τον Πράκτορα Οικείων του.

4.3 Τοπολογία σεναρίου κίνησης

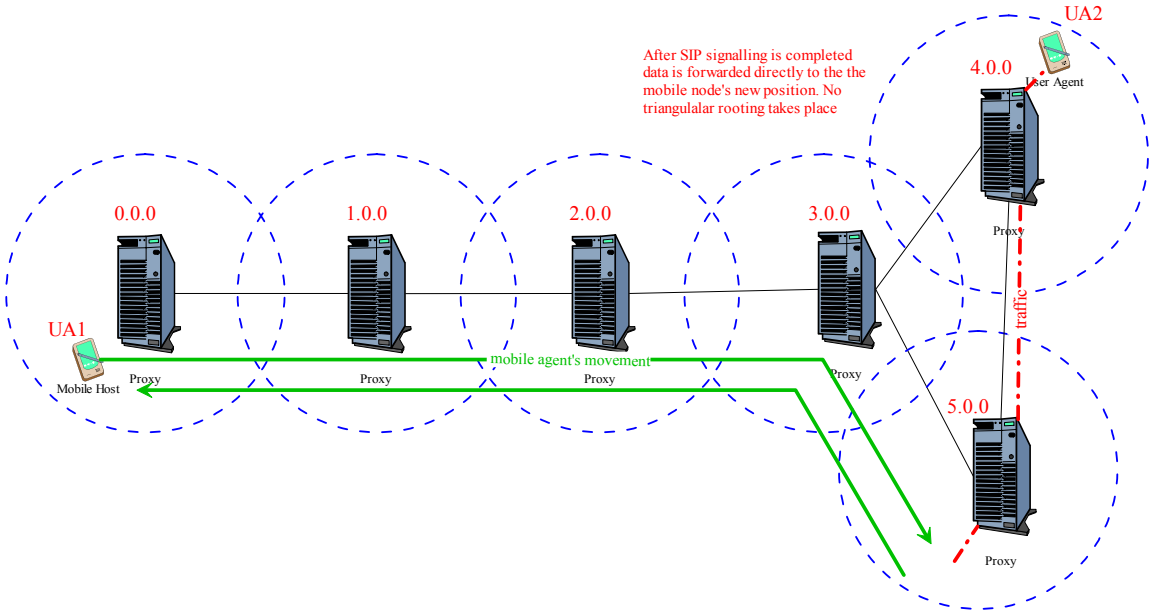
Η τοπολογία πάνω στην οποία βασίστηκαν οι εξομοιώσεις τόσο στο Mobile IP όσο και στο Πρωτόκολλο Έναρξης Συνόδου παρουσιάζεται τα πιο κάτω σχήματα ξεχωριστά για το σενάριο Mobile IP και ξεχωριστά για το SIP.

M I P



Σχήμα 14: Σενάριο Mobile IP.

S I P



Σχήμα 15: Σενάριο SIP.

Επιλέξαμε την πιο πάνω τοπολογία για το λόγο ότι μπορεί να παρουσιαστεί με τον πιο παραστατικό τρόπο η διαφορά ανάμεσα στους 2 μηχανισμούς υποστήριξης κινητικότητας. Θεωρούμε ότι ο ασύρματος κινητός κόμβος UA1 ξεκινά την κίνησή του από την περιοχή 0.0.0 (η οποία αποτελεί και την οικεία περιοχή του) και κατευθύνεται προς την περιοχή 5.0.0 στη πορεία που καθορίζεται από την πράσινη διαδρομή. Όταν φτάσει στην περιοχή 5.0.0, τότε γυρίζει πίσω στην οικεία του περιοχή. Επίσης θεωρούμε ότι ο ασύρματος ακίνητος κόμβος UA2 βρίσκεται εντός της οικείας περιοχής του η οποία είναι η 4.0.0. Κατά τη διάρκεια που ο κόμβος UA1 βρίσκεται μέσα στην περιοχή 2.0.0 αρχίζουν οι αιτήσεις (είτε από το UA1 είτε από το UA2) για την ενεργοποίηση κάποιων εφαρμογών οι οποίες ανήκουν σε κάποια από τις προαναφερθείσες κλάσεις, ανάλογα με το σενάριο. Οι εφαρμογές αυτές είναι ενεργοποιημένες μέχρι ο κόμβος UA1 επιστρέψει στην οικεία περιοχή του. Οι διαφορές ανάμεσα στους δύο μηχανισμούς αναμένεται να γίνουν ορατές ιδιαίτερα στην περίπτωση που ο UA1 θα βρίσκεται στην περιοχή 5.0.0 όπου στην περίπτωση του Mobile IP η κίνηση προερχόμενη από τον UA2 θα αναγκάζεται να ακολουθεί την κόκκινη διακεκομμένη διαδρομή (βλέπε σχήμα 14) ενώ στην περίπτωση του SIP η κίνηση από τον UA2 θα δρομολογείται κατευθείαν μέσω της κόκκινης διακεκομμένης διαδρομής (βλέπε σχήμα 15).

Τα σενάρια που υλοποιήσαμε βασιζόμενοι στην πιο πάνω τοπολογία φαίνονται στον παρακάτω πίνακα.

Σενάριο	Κλάση Κίνησης
Σενάριο 1	TC1 – Mobile IP
Σενάριο 2	TC1 – SIP
Σενάριο 3	TC2 – Mobile IP
Σενάριο 4	TC2 – SIP
Σενάριο 5	TC3 – Mobile IP
Σενάριο 6	2 TC1 – Mobile IP
Σενάριο 7	2 TC1 – SIP
Σενάριο 8	2 TC2 – Mobile IP
Σενάριο 9	2 TC2 – SIP
Σενάριο 10	2 TC3 – Mobile IP
Σενάριο 11	TC1 + TC3 – Mobile IP
Σενάριο 12	TC2 + TC3 – Mobile IP
Σενάριο 13	TC1 + TC2 – Mobile IP
Σενάριο 14	TC1 + TC2 – SIP

Πίνακας 7: Σενάρια και κλάσεις κίνησης.

Μερικές σημαντικές παράμετροι των παραπάνω σεναρίων είναι:

- Ακτίνα κάλυψης σταθμών βάσης : 250m
- Link delay : 400msec
- Link Bandwidth : 200 Mbps
- Movement of UA1 : (171,205) → (1600,1) και αντίστροφα
- Μήκος ουράς iFqueue : 100

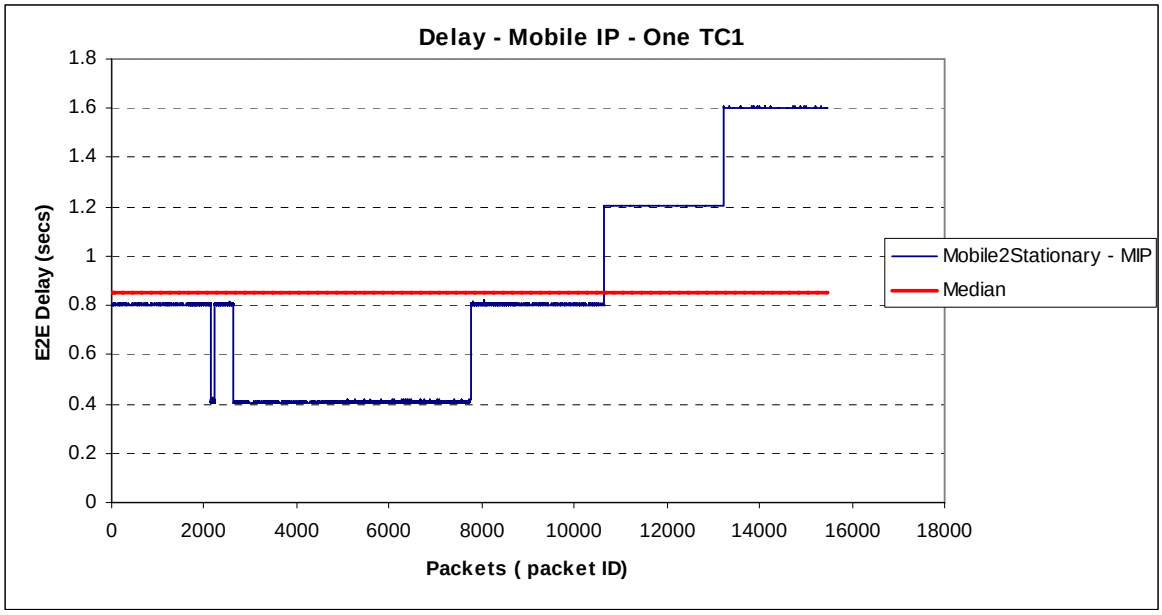
5. Αποτίμηση Αποτελεσμάτων

5.1 Σενάρια 1 – 2

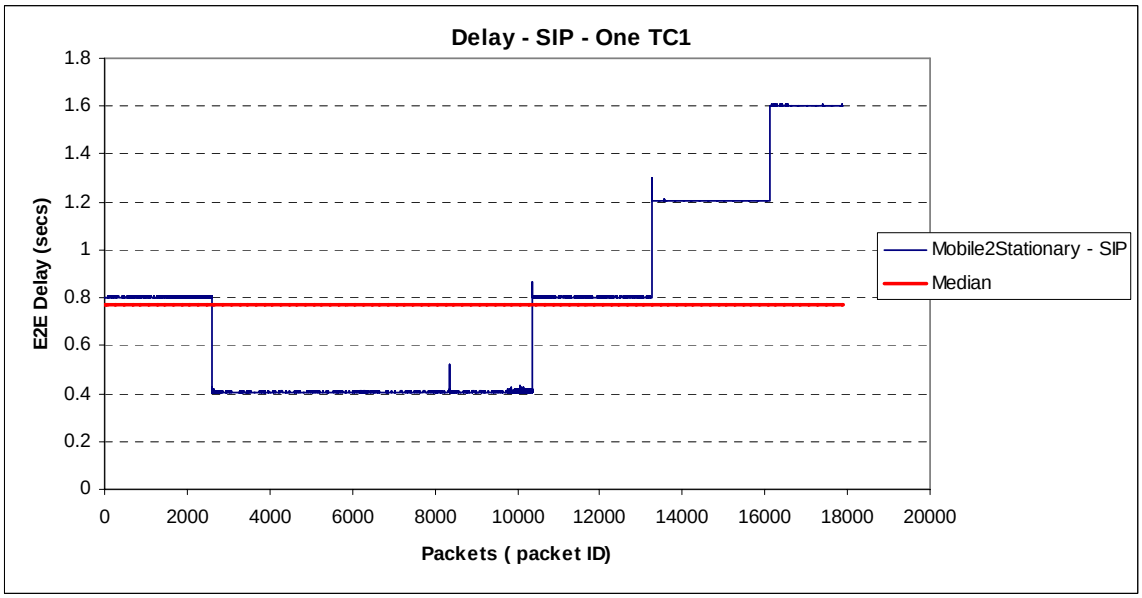
Στο σενάριο αυτό εγκαθιδρύεται μια διαλογική επικοινωνία μεταξύ των δύο κόμβων UA1 και UA2, δηλαδή μια εφαρμογή VoIP η οποία ανήκει στην κλάση TC1. Στην πρώτη περίπτωση το σενάριο υλοποιείται για το Mobile IP και στη δεύτερη περίπτωση για το Πρωτόκολλο Έναρξης Συνόδου (SIP).

Στα παρακάτω σχήματα (σχήμα 16 και 17) – στα οποία παρατίθενται οι δύο μηχανισμοί υποστήριξης κινητικότητας σε αντιδιαστολή – παρουσιάζονται οι στιγμιαίες τιμές και η μέση τιμή της καθυστέρησης όσον αφορά τη ζεύξη από τον κινητό κόμβο UA1 (mobile) προς τον ακίνητο κόμβο UA2 (stationary).

Μπορούμε να παρατηρήσουμε ότι η στιγμιαία και μέση καθυστέρηση παίρνει τις ίδιες τιμές και στις δύο περιπτώσεις (Mobile IP και SIP) για το λόγο ότι τα πακέτα ακολουθούν τη συντομότερη διαδρομή από τον UA1 προς το UA2. Η κλιμακωτή μορφή των γραφικών παραστάσεων οφείλεται στο γεγονός ότι καθώς ο UA1 κινείται από τη περιοχή 2.0.0 προς τη περιοχή 3.0.0 και αργότερα τη 5.0.0, πλησιάζει προς τον ακίνητο κόμβο UA2 οπότε η καθυστέρηση στις περιοχές 3.0.0 και 5.0.0 παίρνει την ελάχιστη τιμή της αφού τα πακέτα έχουν να διασχίσουν μόνο μια ενσύρματη ζεύξη για να φτάσουν στον τελικό προορισμό. Εν συνεχεία καθώς ο κινητός κόμβος UA1 επιστρέφει στην οικεία περιοχή του (0.0.0), απομακρύνεται από τον κόμβο UA2 και έτσι οι τιμές της στιγμιαίας καθυστέρησης αυξάνονται ολοένα και περισσότερο με αποτέλεσμα να φτάσει τη μέγιστη τιμή της όταν ο UA1 μπει στην οικεία περιοχή του.



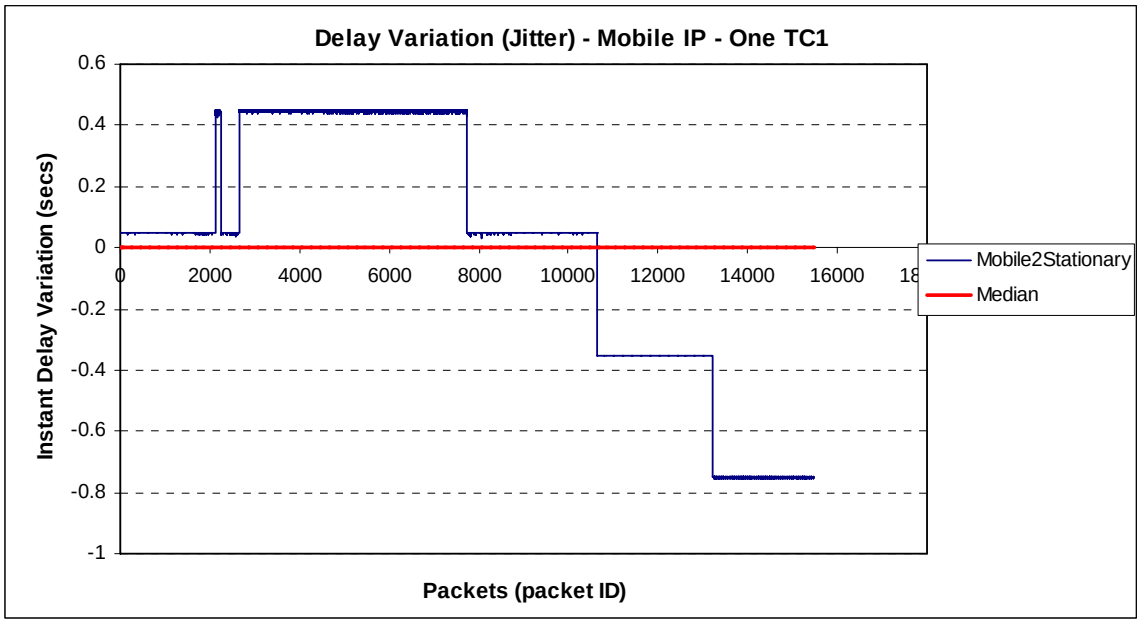
Σχήμα 16: Καθυστέρηση σε σενάριο μιας σύνδεσης TC1 - Κινητός προς σταθερό - Mobile IP.



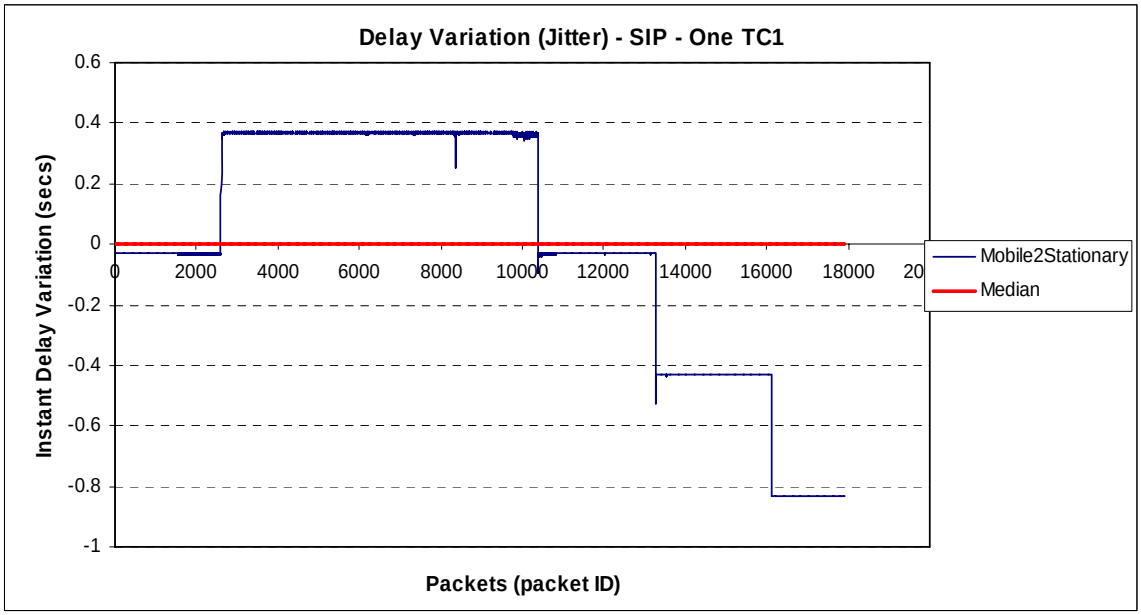
Σχήμα 17: Καθυστέρηση σε σενάριο μιας σύνδεσης TC1 - Κινητός προς σταθερό – SIP.

Στις δύο επόμενες γραφικές παραστάσεις (σχήμα18 και σχήμα19) παρουσιάζονται οι στιγμιαίες τιμές και η μέση τιμή της διακύμανσης της καθυστέρησης (jitter) για τις δύο περιπτώσεις (Mobile IP και SIP) όσον αφορά στη ζεύξη από τον UA1 προς το UA2. Είναι εμφανές ότι οι τιμές αυτές είναι περίπου ίσες κάτι που οφείλεται στους ίδιους λόγους που αναφέραμε πιο πάνω. Η μέση τιμή της διακύμανσης της καθυστέρησης είναι

επιθυμητό να κυμαίνεται γύρω στο μηδέν, χωρίς οι στιγμιαίες τιμές της διακύμανσης να παρουσιάζουν μεγάλες αποκλίσεις από τη μέση τιμή.



Σχήμα 18: Διακύμανση Καθυστέρησης σε σενάριο μιας σύνδεσης TC1 – Κινητός προς σταθερό – Mobile IP.

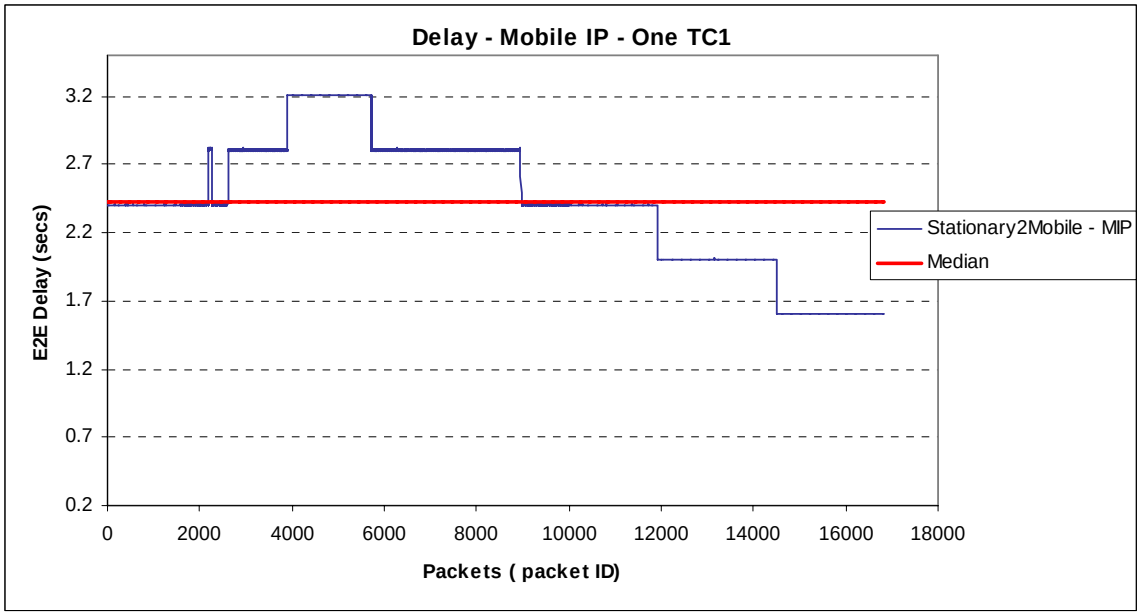


Σχήμα 19: Διακύμανση Καθυστέρησης σε σενάριο μιας σύνδεσης TC1 – Κινητός προς σταθερό – SIP.

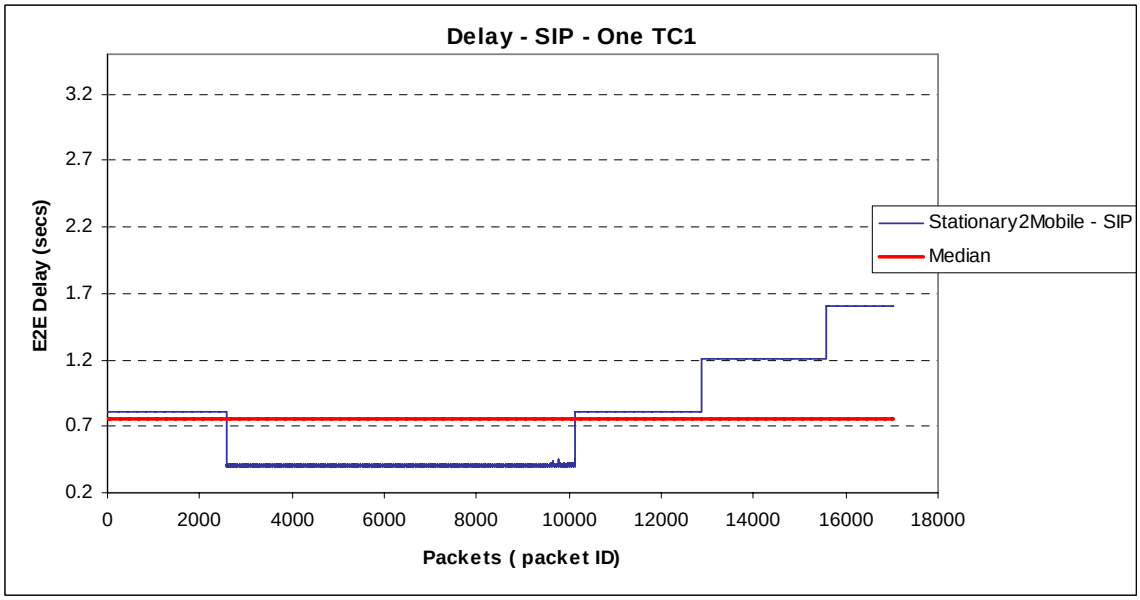
Στα επόμενα σχήματα (σχήμα 20 και σχήμα 21) παρουσιάζονται οι στιγμιαίες τιμές και μέση τιμή της καθυστέρησης όσον αφορά στη ζεύξη από τον ακίνητο κόμβο UA2 προς τον κινητό κόμβο UA1.

Όπως μπορούμε εύκολα να δούμε, όσον αφορά στο Mobile IP, η καθυστέρηση (μέση και στιγμιαίες τιμές) έχουν αρκετά μεγαλύτερες τιμές από αυτές που παρατηρούμε για την ίδια ζεύξη στο SIP. Αυτό ήταν αναμενόμενο από τη στιγμή που στο Mobile IP έχουμε τριγωνική δρομολόγηση πακέτων. Έτσι ο κόμβος UA2 στέλνει τα πακέτα που απευθύνονται στον UA1, προς τον οικείο πράκτορα του UA1 που είναι ο κόμβος με IP 0.0.0 και αυτός με τη σειρά του αφού ενθυλακώσει το κάθε πακέτο, το δρομολογεί προς τον πράκτορα επισκεπτών της περιοχής που βρίσκεται κάθε χρονική στιγμή ο κινούμενος κόμβος UA1. Η διαδικασία αυτή αυξάνει την καθυστέρηση μεταφοράς ενός πακέτου από την πηγή προς τον προορισμό. Καθώς ο κινούμενος κόμβος πλησιάζει στην περιοχή του, μπορούμε να παρατηρήσουμε ότι η καθυστέρηση μειώνεται φτάνοντας στην ελάχιστη τιμή της όταν ο UA1 εισέλθει στην οικεία περιοχή του.

Από την άλλη, όσον αφορά το SIP, η κατάσταση είναι εντελώς διαφορετική. Οι στιγμιαίες τιμές της καθυστέρησης καθώς και η μέση τιμή της, είναι αρκετά μικρότερες από τις αντίστοιχες τιμές στην περίπτωση του Mobile IP για το λόγο ότι τα πακέτα ακολουθούν πάντοτε τη συντομότερη διαδρομή από τον UA1 προς το UA2. Αυτό γίνεται γιατί στο SIP, ο κόμβος που κινείται και αλλάζει περιοχή, ενημερώνει τον άλλο κόμβο για τη νέα διεύθυνση IP στην οποία θα δέχεται τα πακέτα της κίνησης, οπότε τα πακέτα δρομολογούνται απευθείας χωρίς να πηγαίνουν προς το οικείο δίκτυο. Οπότε στην περίπτωση αυτή του SIP, η κλιμακωτή μορφή της γραφικής έχει αντίστροφη συμπεριφορά αφού η καθυστέρηση μειώνεται στην ελάχιστη τιμή της καθώς ο UA1 βρίσκεται στις περιοχές των κόμβων 3.0.0 και 5.0.0, και αυξάνεται καθώς επιστρέφει στην οικεία περιοχή του, χωρίς όμως να φτάνει τη μέγιστη τιμή της καθυστέρησης που ειχαμε στο Mobile IP.



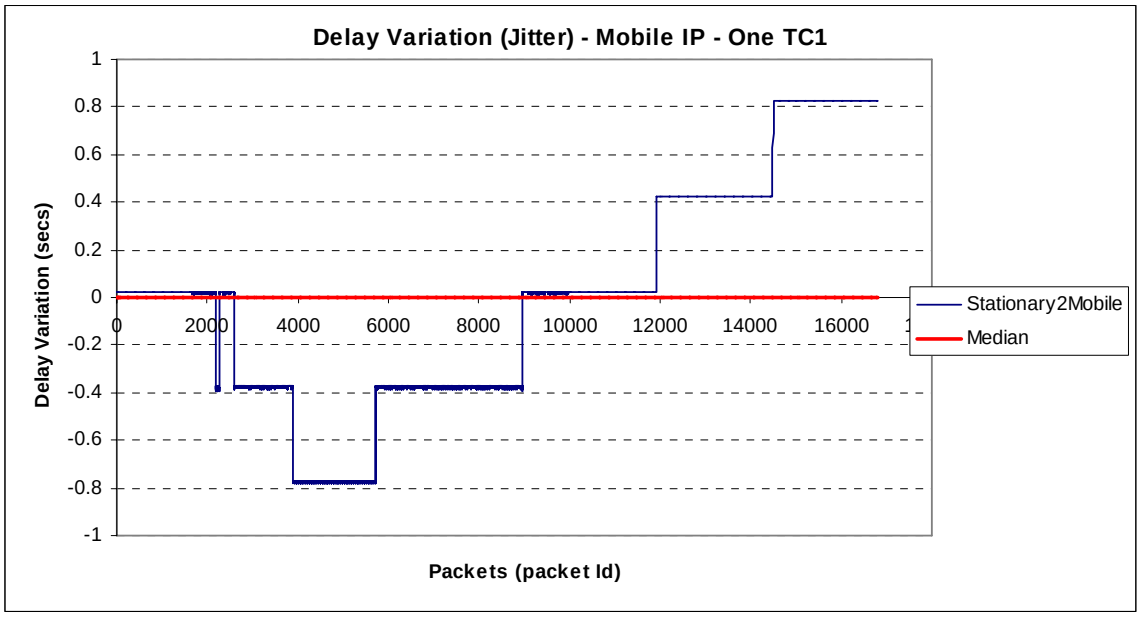
Σχήμα 20: Καθυστέρηση σε σενάριο μιας σύνδεσης TC1 – Σταθερός προς κινητό – Mobile IP.



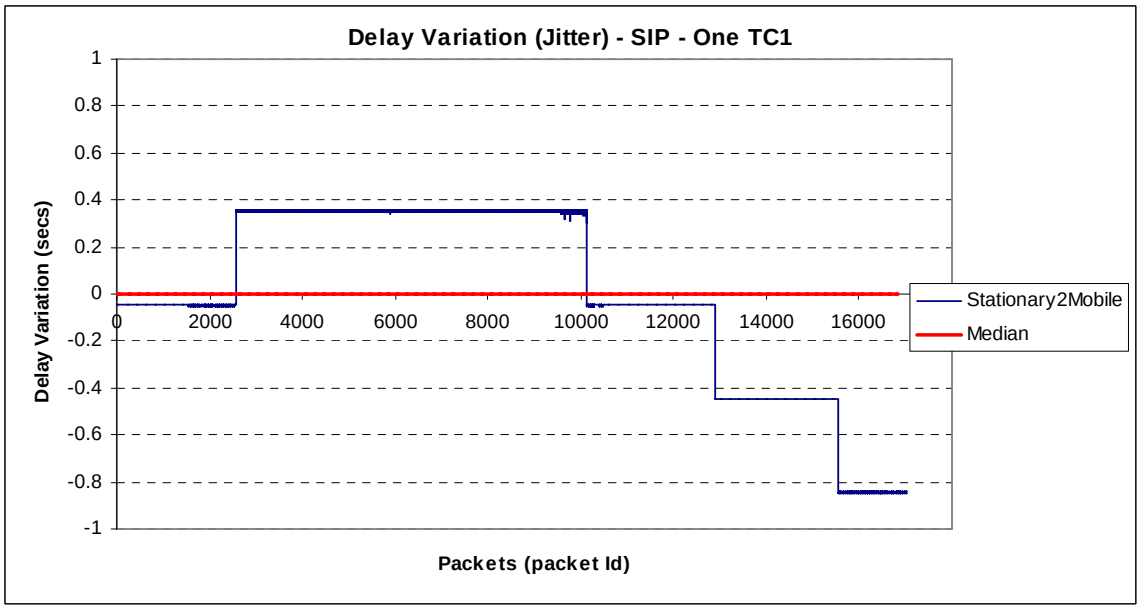
Σχήμα 21: Καθυστέρηση σε σενάριο μιας σύνδεσης TC1 – Σταθερός προς κινητό – SIP.

Όσον αφορά στη διακύμανση της καθυστέρησης (σχήμα 22 και σχήμα 23) για την ίδια ζεύξη, μπορούμε κατ' αρχήν να παρατηρήσουμε οτι και στις δύο περιπτώσεις η μέση τιμή είναι μηδέν. Αυτό δεν είναι κατ' ανάγκη καλό γιατί πρέπει να έχουμε μικρές αποκλίσεις των στιγμιαίων τιμών από τη μέση τιμή, κάτι που όπως φαίνεται συμβαίνει μόνο στην περίπτωση του SIP. Στη περίπτωση του Mobile IP, οι αποκλίσεις από τη μέση

τιμή φτάνουν ακόμα και τα 0.8 δευτερόλεπτα αν αντιθέσει με το SIP όπου οι αποκλίσεις κυμαίνονται στα 0.4 δευτερόλεπτα.



Σχήμα 22: Διακύμανση Καθυστέρησης σε σενάριο μιας σύνδεσης TC1 – Σταθερός προς κινητό – Mobile IP.

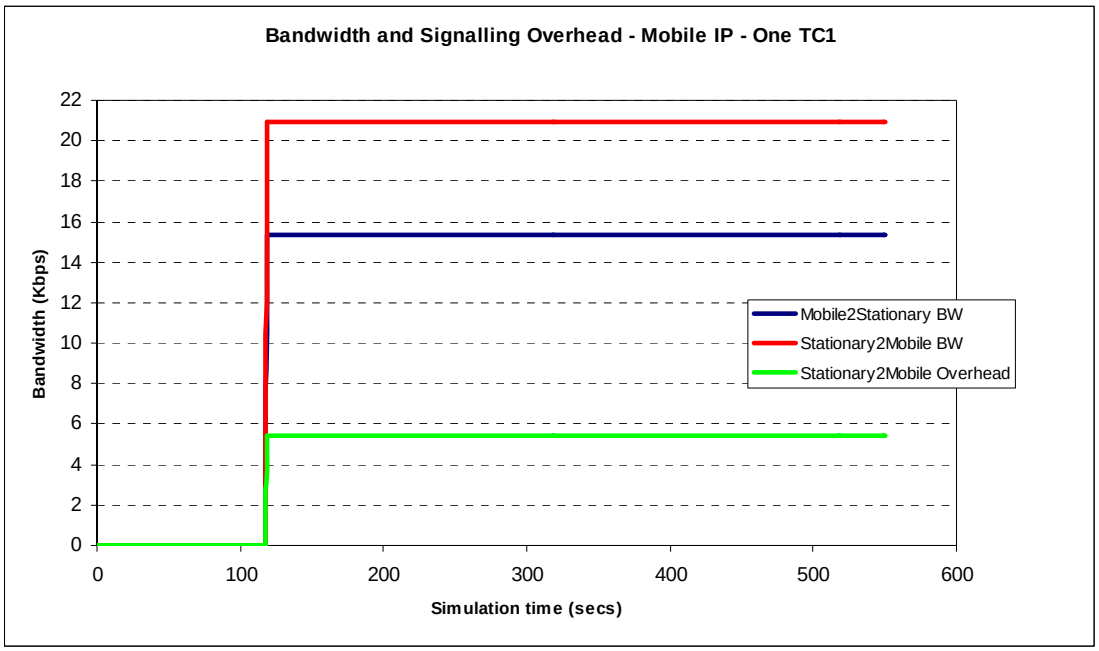


Σχήμα 23: Διακύμανση Καθυστέρησης σε σενάριο μιας σύνδεσης TC1 – Σταθερός προς κινητό – SIP.

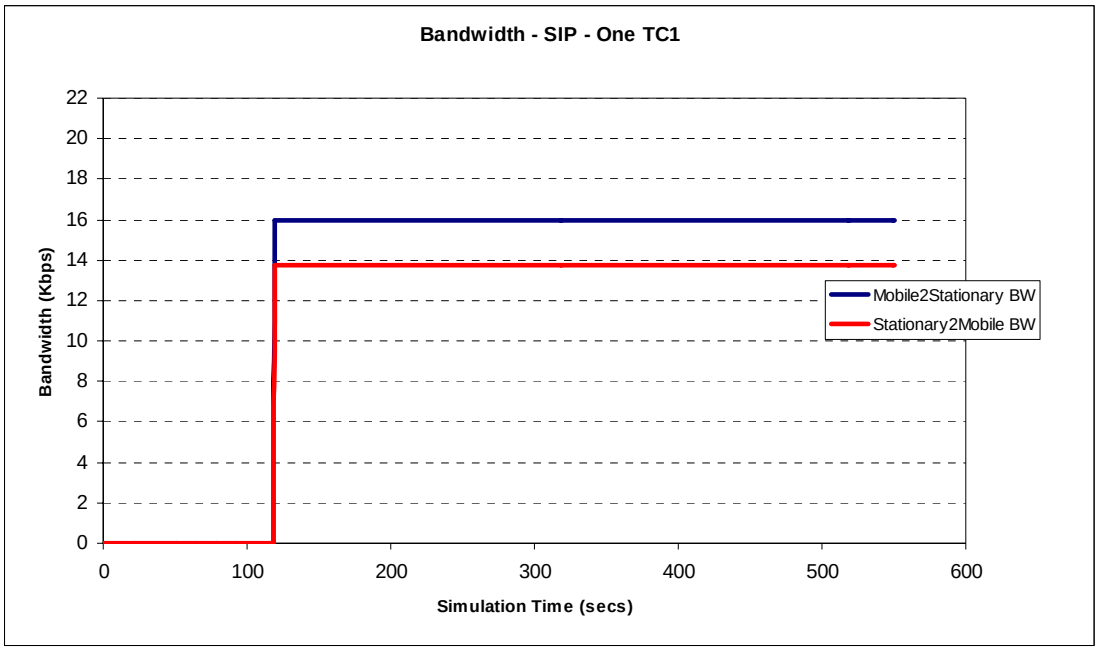
Όσον αφορά το σενάριο 1 που εμπλέκει το Mobile IP, υπάρχει επιβάρυνση λόγω σηματοδοσίας, αφού στο κάθε πακετό που φτάνει στον οικείο πράκτορα από τον κόμβο

UA2 με προορισμό τον κόμβο UA1, προστίθενται 20bytes (επικεφαλίδα πακέτου IP) λόγω ενθυλάκωσης κάτι που δεν συμβαίνει στο SIP. Στα παρακάτω σχήματα (σχήμα 24 και σχήμα 25) παρουσιάζεται ο ρυθμός μετάδοσης ανά ζεύξη σε συνδυασμό με την επιβάρυνση σε Kbps για το Mobile IP καθώς και ο ρυθμός μετάδοσης για το SIP.

Σημειώνεται ότι η ροή πληροφορίας ξεκινά τη χρονική στιγμή 120sec και τερματίζεται τη χρονική στιγμή 550sec.



Σχήμα 24: Ρυθμός μετάδοσης και επιβάρυνση σηματοδοσίας σε σενάριο μιας σύνδεσης TC1 – Mobile IP.



Σχήμα 25: Ρυθμός μετάδοσης σε σενάριο μιας σύνδεσης TC1 – SIP.

Όπως παρατηρούμε στο σχήμα 24 που αφορά την περίπτωση Mobile IP, ο ρυθμός μετάδοσης πληροφορίας στη ζεύξη από τον κόμβο UA2 (σταθερός) προς το κόμβο UA1 (κινητός), είναι γύρω στα 21Kbps δηλαδή μεγαλύτερο από το ρυθμό με τον οποίο εκπέμπεται πληροφορία από τη πηγή (UA1) που είναι 16Kbps. Αυτό οφείλεται στο γεγονός ότι ο ρυθμός των 21Kbps δεν είναι ρυθμός μετάδοσης καθαρής πληροφορίας αλλά συμπεριλαμβάνει και την πρόσθετη επιβάρυνση που εισάγεται σε κάθε πακέτο που δρομολογείται προς τον UA1 όταν αυτός βρίσκεται εκτός της οικείας περιοχής του. Στο ίδιο σχήμα παρατηρούμε ότι η επιβάρυνση σηματοδοσίας (20 bytes/packet όταν ο κινητός κόμβος βρίσκεται εκτός της οικείας περιοχής του) είναι περίπου 5.4Kbps κατά μέσο όρο. Στη περίπτωση συνδέσεων κλάσης TC1 όπου τα πακέτα UDP που αποστέλλονται έχουν μέγεθος 48 bytes, η προσθήκη άλλων 20 bytes προσθέτει αρκετά μεγάλη επιβάρυνση εν αντιθέσει με την κλάση TC3 όπου τα πακέτα TCP που αποστέλλονται έχουν μέγεθος 1000bytes, οπότε η προσθήκη 20bytes δεν τα επιβαρύνει τόσο.

Αν αφαιρέσουμε τους δύο αυτούς ρυθμούς, αυτό που προκύπτει είναι ο μέσος ρυθμός μετάδοσης καθαρής πληροφορίας και είναι περίπου 15.6Kbps. Ο μέσος ρυθμός μετάδοσης πληροφορίας στη ζεύξη από το σταθερό κόμβο UA2 στον κινητό κόμβο UA1

είναι ελάχιστα μεγαλύτερος από τον αντίστοιχο μέσο ρυθμό στη ζεύξη από τον κόμβο UA1 στο UA2 που προσεγγίζει τα 15.4Kbps. Αυτό πιθανώς να οφείλεται στο γεγονός ότι η ζεύξη από σταθερό προς κινητό είναι πιο ευάλωτη σε απώλειες λόγω του ότι κινείται ο προορισμός (κινητός κόμβος).

Στη περίπτωση του SIP (βλέπε σχήμα 25) δεν υπάρχει επιβάρυνση σηματοδοσίας και ο μέσος ρυθμός μετάδοσης στη ζεύξη του κινητού κόμβου προς το σταθερό, προσεγγίζει τον ρυθμό αποστολής πληροφορίας που είναι 16Kbps. Στην αντίθετη ζεύξη ο ρυθμός πληροφορίας είναι γύρω στα 13.7Kbps. Παρατηρούμε ότι στη ζεύξη αυτή, από τον ακίνητο κόμβο UA2 προς τον κινητό κόμβο UA1 είναι μικρότερος ο μέσος ρυθμός μετάδοσης πληροφορίας. Αυτό οφείλεται πιθανόν στο γεγονός ότι χάνονται πολύ περισσότερα πακέτα στη ζεύξη αυτή γιατί ο κόμβος UA1 κινείται συνεχώς αλλάζοντας περιοχές, ενώ ο κόμβος UA2 παραμένει ακίνητος (σταθερός).

Όσον αφορά το μέσο χρόνο διαπομπής, στο SIP είναι περίπου 4.2 δευτερόλεπτα ενώ στο Mobile IP είναι 7.7 δευτερόλεπτα. Ο χρόνος διαπομπής υπολογίστηκε με το τρόπο που αναφέραμε στο προηγούμενο κεφάλαιο.

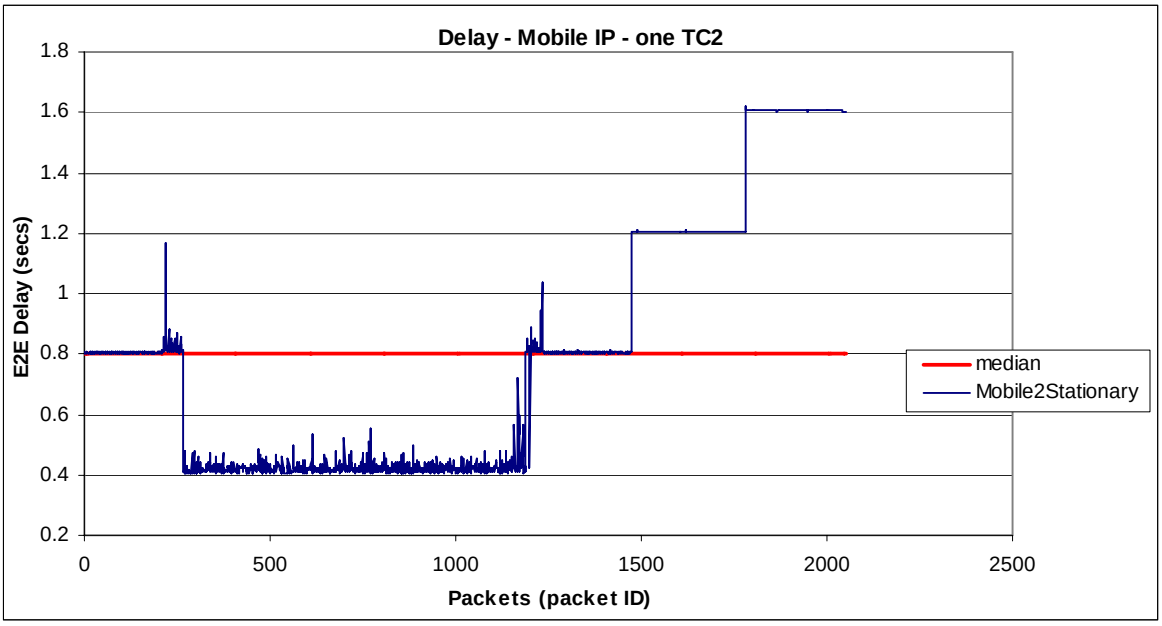
Το ποσοστό απώλειας πακέτων στο SIP είναι 0.06% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 13.8% για την αντίθετη ζεύξη κάτι που ήταν αναμενόμενο μιας και ο κόμβος UA1 κινείται συνεχώς αλλάζοντας περιοχές με αποτέλεσμα τα πακέτα που κατευθύνονται προς αυτόν να κινδυνεύουν περισσότερο να απολεσθούν. Αυτό ήταν εμφανές και από το μικρότερο ρυθμό πληροφορίας πάνω στη ζεύξη αυτή όπως επισημάναμε στην προηγούμενη παράγραφο.

Από την άλλη πλευρά το ποσοστό απώλειας πακέτων στο Mobile IP είναι 3.88% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 3.01% για την αντίθετη ζεύξη. Παρατηρούμε αρκετά μικρότερο ποσοστό απώλειας πακέτων στην ζεύξη από τον σταθερό προς τον κινητό κόμβο. Αυτό συμβαίνει γιατί στο Mobile IP, η αλλαγή σταθμού βάσης γίνεται στο μέσο της επικαλυπτόμενης περιοχής μεταξύ δύο σταθμών βάσης, έτσι τα πακέτα που κατευθύνονται στον κινητό κόμβο μέσω του παλιού σταθμού βάσης

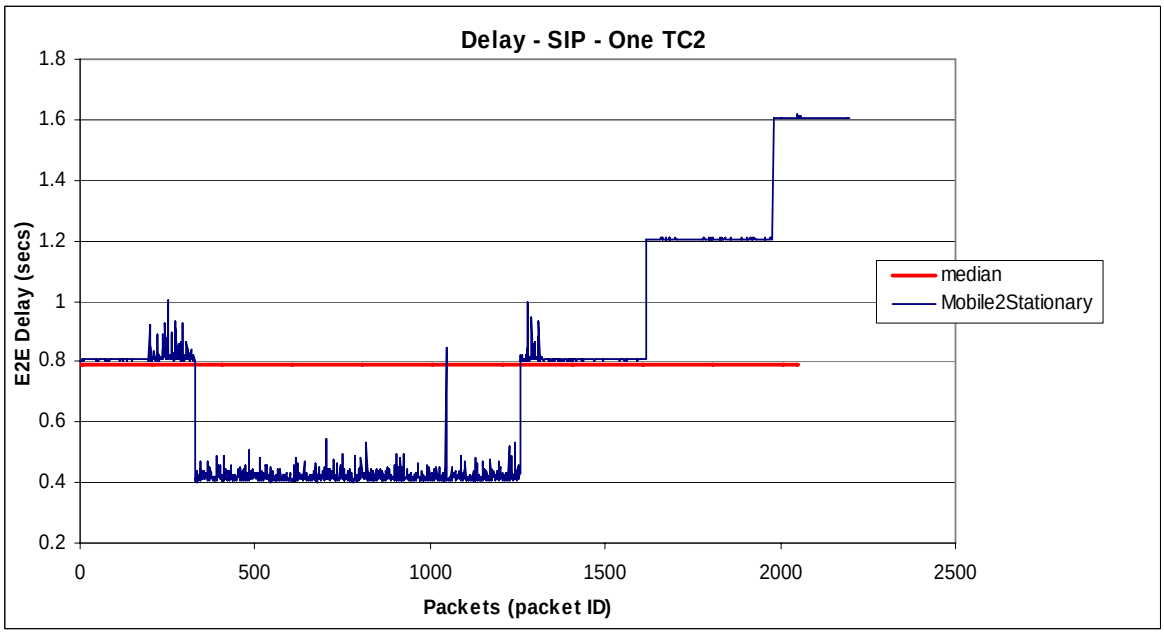
καταφέρνουν να φτάσουν σε αυτόν γιατί συνεχίζει να βρίσκεται εντός της εμβέλειας του. Από την άλλη στο SIP, η αλλαγή σταθμού βάσης και άρα η ενημέρωση του correspondent κόμβου γίνεται αφότου βγει ο κινητός κόμβος έξω από τη περιοχή του παλιού σταθμού βάσης. Οπότε τα πακέτα που απευθύνονται προς τον κινητό κόμβο μέσω του παλιού σταθμού βάσης (προτού ενημερωθεί ο correspondent κόμβος), χάνονται και άρα οδηγούμαστε σε μεγάλο ποσοστό απώλειας στη ζεύξη αυτή.

5.2 Σενάρια 3 – 4

Στα συγκεκριμένα σενάρια εγκαθιδρύεται μια video streaming εφαρμογή, η οποία ανήκει στην κλάση TC2. Και πάλι στην πρώτη περίπτωση το σενάριο υλοποιείται για το Mobile IP και στη δεύτερη περίπτωση για το Πρωτόκολλο Έναρξης Συνόδου (SIP). Στα παρακάτω σχήματα (βλέπε σχήμα 26 και σχήμα 27) – στα οποία παρατίθενται οι δύο μηχανισμοί υποστήριξης κινητικότητας σε αντιδιαστολή – παρουσιάζονται οι στιγμιαίες τιμές και η μέση τιμή της καθυστέρησης όσον αφορά τη ζεύξη από τον κινητό κόμβο UA1 (mobile) προς τον ακίνητο κόμβο UA2 (stationary), η οποία είναι αυτή με το χαμηλό ρυθμό 2Kbps (uplink). Όπως εύκολα βλέπει κανείς η στιγμιαία και μέση καθυστέρηση παίρνουν σχεδόν τις ίδιες τιμές και στις δύο περιπτώσεις (Mobile IP και SIP). Ο λόγος είναι και πάλι ότι τα πακέτα ακολουθούν τη συντομότερη διαδρομή από τον UA1 προς το UA2. Ενώ η κλιμάκωση των γραφικών παραστάσεων οφείλεται και πάλι στην αλλαγή περιοχών από τον κινητό κόμβο.



Σχήμα 26: Καθυστέρηση σε σενάριο μιας σύνδεσης TC2 – Uplink (2Kbps) από κινητό προς σταθερό – Mobile IP.

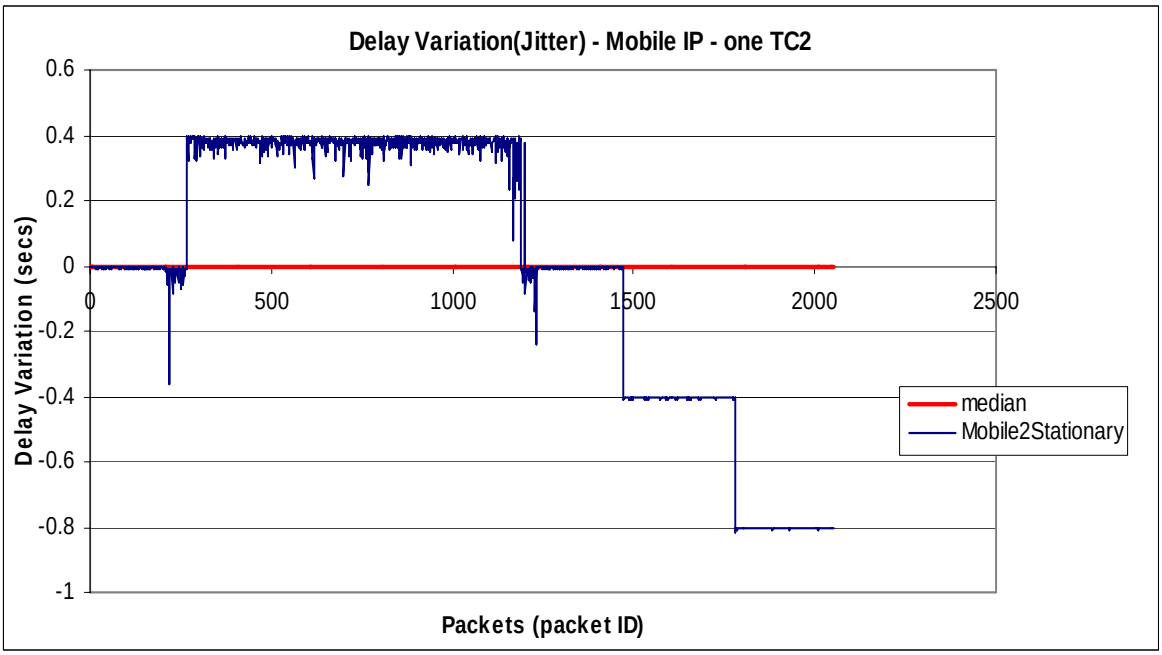


Σχήμα 27: Καθυστέρηση σε σενάριο μιας σύνδεσης TC2 – Uplink (2Kbps) από κινητό προς σταθερό – SIP.

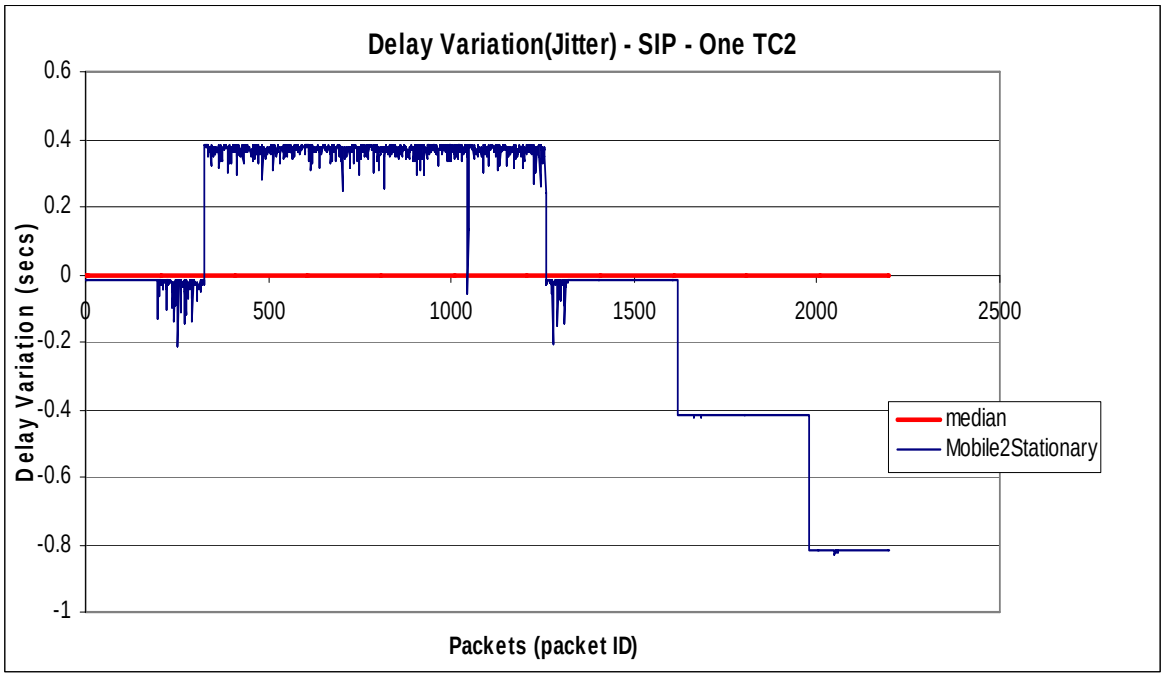
Οι δύο επόμενες γραφικές παραστάσεις (βλέπε σχήμα 28 και σχήμα 29) όπου παρουσιάζονται οι στιγμιαίες τιμές και η μέση τιμή της διακύμανσης της καθυστέρησης (jitter) για τις δύο περιπτώσεις (Mobile IP και SIP) όσον αφορά τη ζεύξη από τον UA1

προς το UA2 έχουν και αυτές ίσες τιμές κάτι που οφείλεται στους ίδιους λόγους που αναφέραμε πιο πάνω. Η μέση τιμή της διακύμανσης της καθυστέρησης είναι επιθυμητό να κυμαίνεται γύρω στο μηδέν, χωρίς οι στιγμιαίες τιμές της διακύμανσης να παρουσιάζουν μεγάλες αποκλίσεις από τη μέση τιμή.

Επιπλέον, μια αξιοσημείωτη διαφοροποίηση είναι ο σημαντικά μεγαλύτερος αριθμός πακέτων που παρατηρείται στο SIP και οφείλεται όπως γίνεται αντιληπτό και από όλες τις προσομοιώσεις στο ότι τα χαμένα πακέτα είναι πολύ λιγότερα όταν χρησιμοποιείται το SIP.

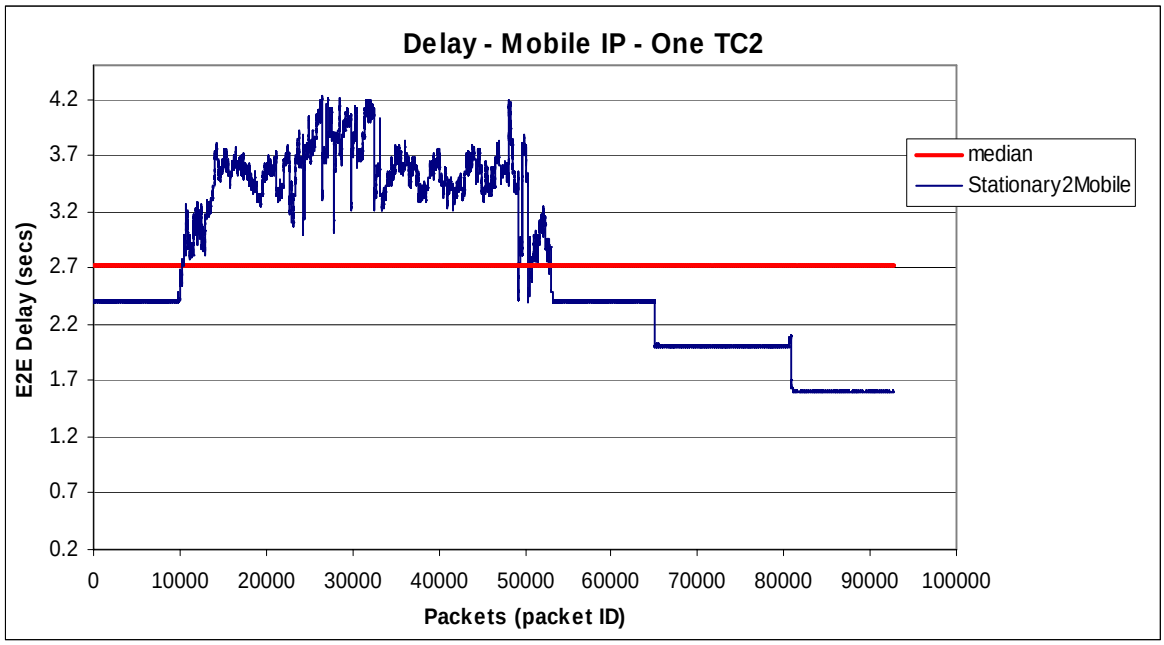


Σχήμα 28: Διακύμανση της καθυστέρησης σε σενάριο μιας σύνδεσης TC2 – Uplink (2Kbps) από κινητό προς σταθερό – Mobile IP

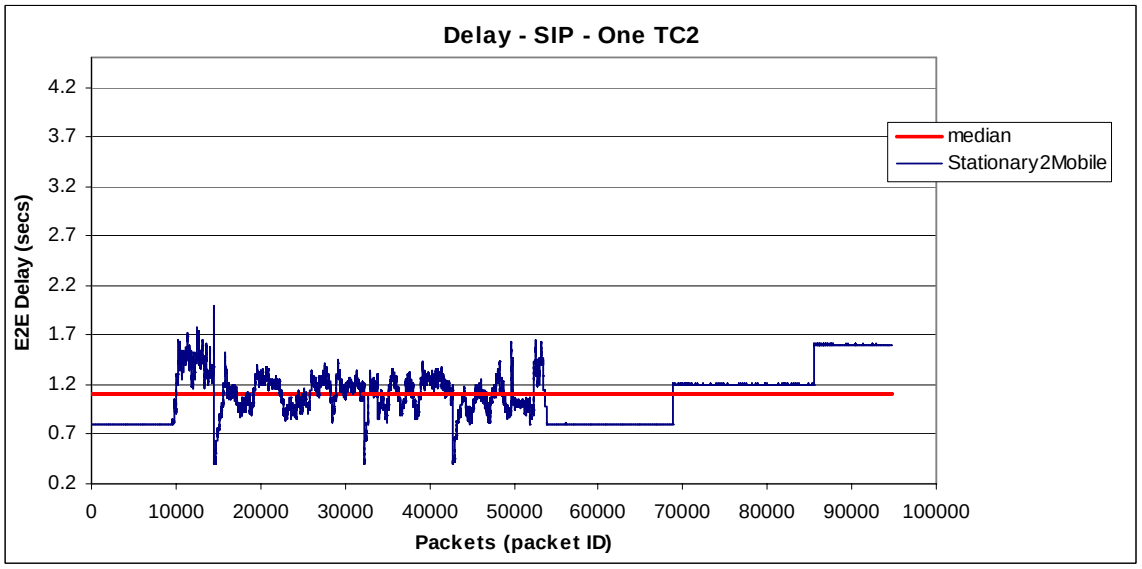


Σχήμα 29: Διακύμανση της καθυστέρησης σε σενάριο μιας σύνδεσης TC2 – Uplink (2Kbps) από κινητό προς σταθερό - SIP.

Στα σχήματα που ακολουθούν (σχήμα 30 και σχήμα 31) παρουσιάζονται τα ίδια μεγέθη όσον αφορά τη ζεύξη από τον ακίνητο κόμβο UA2 προς το κινητό κόμβο UA1, δηλαδή αυτή με την υψηλή ταχύτητα 100Kbps (downlink). Όπως μπορούμε εύκολα να δούμε, όσον αφορά στο Mobile IP, η καθυστέρηση (μέση και στιγμιαίες τιμές) έχουν αρκετά μεγαλύτερες τιμές από αυτές που παρατηρούμε για την ίδια ζεύξη στο SIP. Η αιτία είναι και πάλι η επιπλέον καθυστέρηση που επιβάλλει η τριγωνική δρομολόγηση. Από την άλλη, όσον αφορά το SIP, οι στιγμιαίες τιμές της καθυστέρησης καθώς και η μέση τιμή της, είναι αρκετά μικρότερες από τις αντίστοιχες τιμές του Mobile IP διότι τα πακέτα ακολουθούν πάντοτε τη συντομότερη διαδρομή. Ενώ βέβαια η κλιμάκωση οφείλεται και πάλι στις αλλαγές περιοχών. Στο μεν Mobile IP η κλιμάκωση γίνεται προς τα πάνω όσο απομακρυνόμαστε από τον οικείο πληρεξούσιο εξυπηρετητή, στο δε SIP όσο απομακρύνονται οι κόμβοι μεταξύ τους, προς οποιαδήποτε κατεύθυνση.



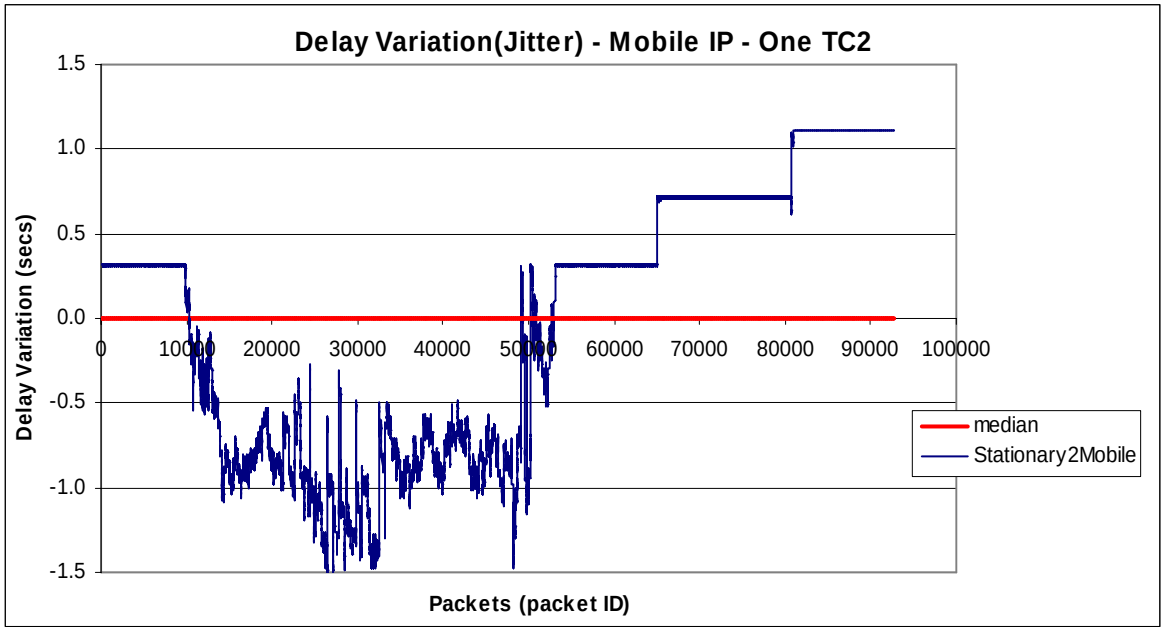
Σχήμα 30: Καθυστέρηση σε σενάριο μιας σύνδεσης TC2 – Downlink (100Kbps) από κινητό προς σταθερό – Mobile IP



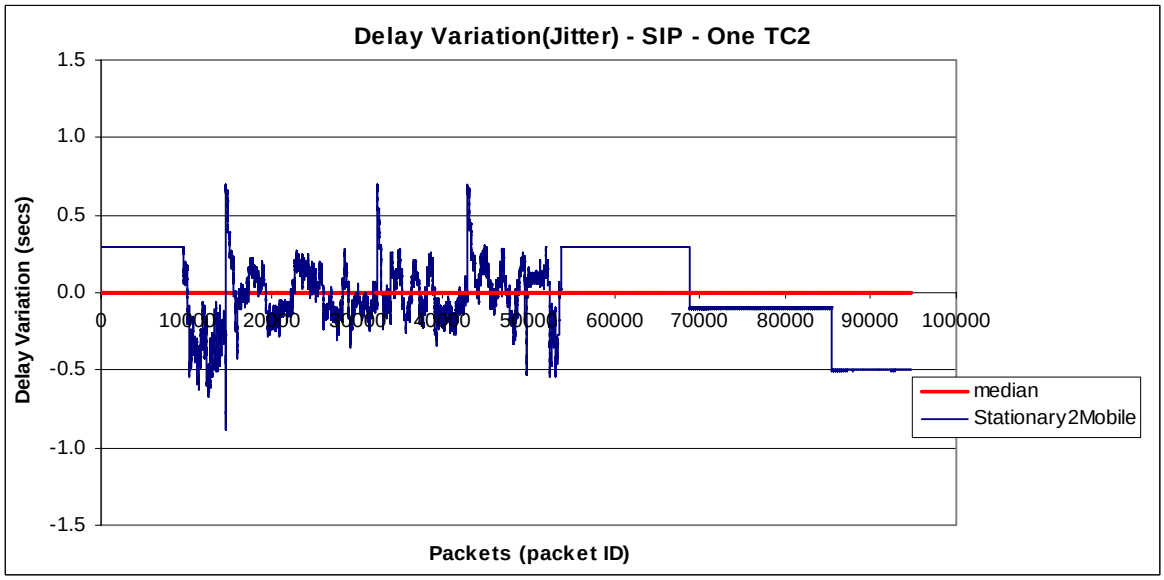
Σχήμα 31: Καθυστέρηση σε σενάριο μιας σύνδεσης TC2 – Downlink (100Kbps) από κινητό προς σταθερό – SIP.

Βέβαια συγκρίνοντας και τις γραφικές παραστάσεις και για τη διακύμανση της καθυστέρησης (βλέπε σχήμα 32 και σχήμα 33) παρατηρούμε ότι το SIP υπερτερεί. Αυτό γιατί παρόλο που και στις δύο περιπτώσεις η μέση τιμή της διακύμανσης είναι περίπου μηδενική στην περίπτωση του Mobile IP, οι αποκλίσεις από τη μέση τιμή φτάνουν ακόμα

και τα 1,5sec ενώ στο SIP οι αποκλίσεις κυμαίνονται στα 0.5sec. Και στη συγκεκριμένη περίπτωση ροής video (video streaming) καταλαβαίνουμε ότι πιο σημαντικό είναι να μην «ταλαντώνεται» σημαντικά η καθυστέρηση παρά να είναι μικρή. Οπότε καταλαβαίνουμε γιατί το SIP θα πλεονεκτούσε σημαντικά σε μια τέτοια περίπτωση.

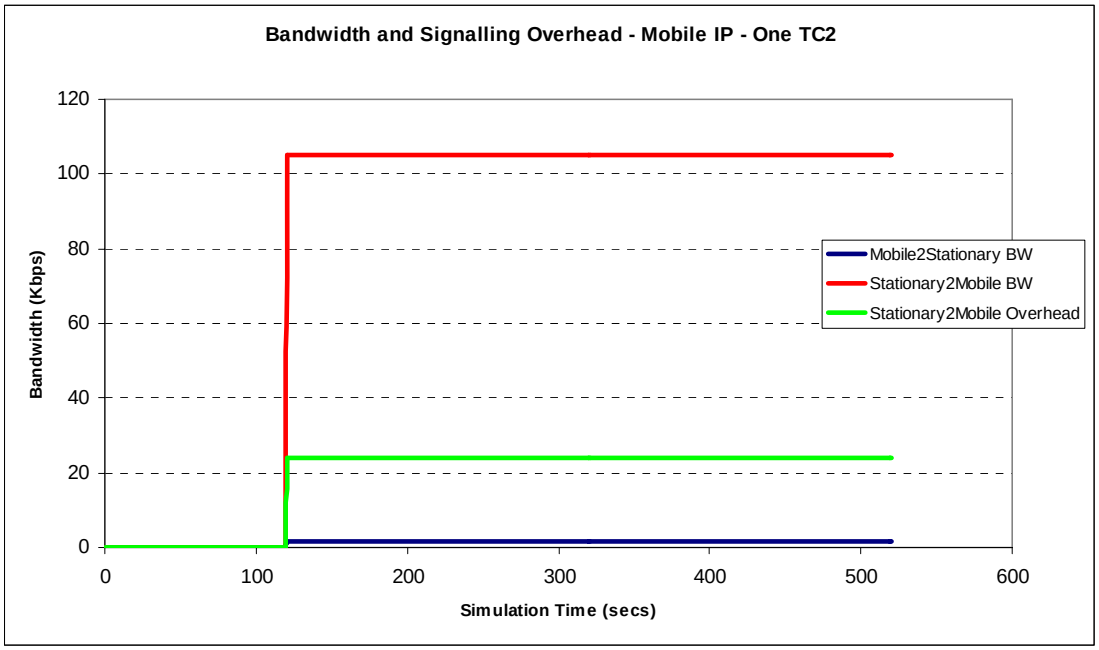


Σχήμα 32: Διακύμανση της καθυστέρησης σε σενάριο μιας σύνδεσης TC2 – Downlink (100Kbps) από κινητό προς σταθερό – Mobile IP.

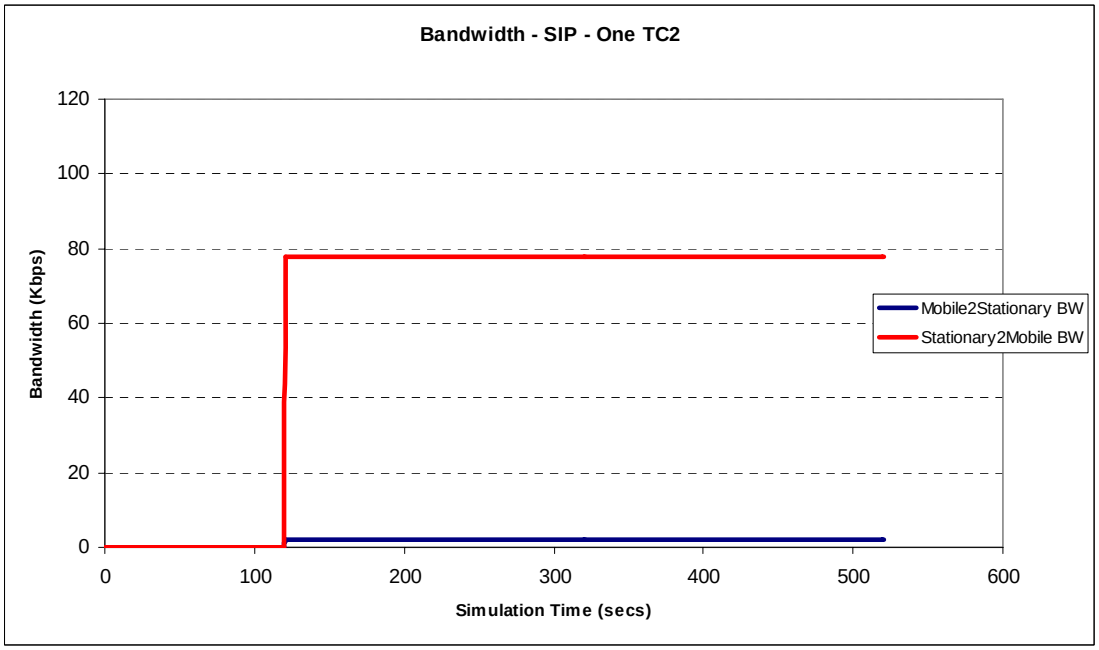


Σχήμα 33: Διακύμανση της καθυστέρησης σε σενάριο μιας σύνδεσης TC2 – Downlink (100Kbps) από κινητό προς σταθερό – SIP.

Στα παρακάτω σχήματα (σχήμα 34 και σχήμα 35) παρουσιάζεται ο ρυθμός μετάδοσης ανά ζεύξη σε συνδυασμό με την επιβάρυνση σε Kbps για το Mobile IP καθώς και ο ρυθμός μετάδοσης για το SIP για την περίπτωση μιας σύνδεσης TC2.



Σχήμα 34: Ρυθμός μετάδοσης και επιβάρυνση σηματοδοσίας σε σενάριο μιας σύνδεσης TC2 – Mobile IP.



Σχήμα 35: Ρυθμός μετάδοσης σε σενάριο μιας σύνδεσης TC2 – SIP.

Όπως παρατηρούμε στο σχήμα 34 που αφορά την περίπτωση Mobile IP, ο ρυθμός μετάδοσης πληροφορίας στη ζεύξη από το σταθερό κόμβο UA2 προς τον κινητό κόμβο UA1 (downlink), είναι γύρω στα 106Kbps δηλαδή μεγαλύτερο από το ρυθμό με τον οποίο εκπέμπεται πληροφορία από την πηγή (UA1) που είναι 100Kbps. Αυτό οφείλεται όπως έχουμε προαναφέρει στην επιβάρυνση σηματοδοσίας του Mobile IP. Στο ίδιο σχήμα παρατηρούμε ότι η επιβάρυνση σηματοδοσίας είναι περίπου 24Kbps κατά μέσο όρο. Αν αφαιρέσουμε τους δύο αυτούς ρυθμούς, αυτό που προκύπτει είναι ο μέσος ρυθμός μετάδοσης καθαρής πληροφορίας και είναι περίπου 82Kbps. Στη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό κόμβο UA2 (uplink), ο μέσος ρυθμός μετάδοσης πληροφορίας είναι 1.7Kbps.

Στη περίπτωση του SIP (βλέπε σχήμα 35) δεν υπάρχει επιβάρυνση σηματοδοσίας και ο μέσος ρυθμός μετάδοσης σε κάθε ζεύξη προσεγγίζει τα 78Kbps ενώ στη ζεύξη από τον ακίνητο κόμβο UA2 προς τον κινητό κόμβο UA1, ο μέσος ρυθμός μετάδοσης πληροφορίας είναι 1.96Kbps. Παρατηρούμε ότι ο ρυθμός μετάδοσης πληροφορίας στο downlink είναι ελάχιστα μικρότερος από ότι στο Mobile IP, αλλά στην περίπτωση του SIP, οι αποκλίσεις των στιγμιαίων τιμών της διακύμανσης της καθυστέρησης (jitter) από τη μέση τιμή είναι πολύ μικρότερες όπως είδαμε πιο πάνω, και αυτή είναι πιο κρίσιμη παράμετρος σε τέτοιες εφαρμογές.

Όσον αφορά το μέσο χρόνο διαπομπής, στο SIP είναι περίπου 6.5 δευτερόλεπτα. Στη περίπτωση του Mobile IP, ο μέσος χρόνος διαπομπής είναι 4.8 δευτερόλεπτα.

Το ποσοστό απώλειας πακέτων στο SIP είναι 1.8% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 21.72% για την αντίθετη ζεύξη κάτι που ήταν αναμενόμενο μιας και ο κόμβος UA1 κινείται συνεχώς αλλάζοντας περιοχές με αποτέλεσμα τα πακέτα που κατευθύνονται προς αυτόν να κινδυνεύουν περισσότερο να απωλεσθούν.

Από την άλλη πλευρά το ποσοστό απώλειας πακέτων στο Mobile IP είναι 15% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 18.42% για την αντίθετη ζεύξη.

Η εμφανώς μεγαλύτερη απώλεια πακέτων κατά το downlink (ζεύξη 100Kbps) τόσο στην περίπτωση του SIP όσο και στην περίπτωση του Mobile IP δικαιολογείται από το γεγονός ότι στέλνεται μεγαλύτερη ποσότητα πακέτων στη μονάδα του χρόνου από ότι στο uplink (ζεύξη 2Kbps), γι' αυτό άλλωστε υπάρχει μεγαλύτερη πιθανότητα απώλειας πακέτων.

5.3 Σενάριο 5

Στο σενάριο αυτό οι δύο χρήστες εγκαθιδρύουν μια κίνηση μεταβλητή ρυθμού (VBR). Συγκεκριμένα, η εξομοίωση αυτή αφορά FTP πάνω από TCP. Ο μηχανισμός υποστήριξης κινητικότητας που μελετάται είναι το Mobile IP. Ο ένας από τους δύο χρήστες κάνει αίτηση για μεταφορά ενός αρχείου την χρονική στιγμή 120. Όπως φαίνεται στην παρακάτω γραφική παράσταση (βλέπε σχήμα 36) αρχικά ο ρυθμός μεταφοράς (bandwidth) είναι αρκετά υψηλός. Ωστόσο, παρατηρούμε μια σημαντική επιβράδυνση του ρυθμού δεδομένων που προκαλείται από τις διαπομπές που συμβαίνουν καθώς κινείται ο ένας χρήστης. Αυτό είναι αναμενόμενο γιατί έτσι απαιτεί η λειτουργία του TCP· όταν παρουσιαστεί μια εκπνοή χρόνου (δηλ. τα πακέτα ACK δεν έχουν φτάσει στην ώρα τους) μειώνεται ο ρυθμός που ο αποστολέας στέλνει τα δεδομένα του (μείωση παραθύρου συμφόρησης σύμφωνα με τον αλγόριθμο της αργής αρχής).

Σημειώνεται ότι σε τέτοιου είδους κινήσεις που τρέχουν στο παρασκήνιο (WEB, FTP) δεν ενδιαφερόμαστε για την καθυστέρηση (delay) και την διακύμανση καθυστέρησης (jitter) σε αντίθεση με τις κινήσεις πραγματικού χρόνου που τέτοια πειραματικά δεδομένα είναι πολύ χρήσιμα. Για αυτό το λόγο, στα σενάρια που αφορούν κινήσεις TC3 δεν έχει νόημα η παρουσίαση των μετρήσεων αυτών σε γραφικές παραστάσεις.

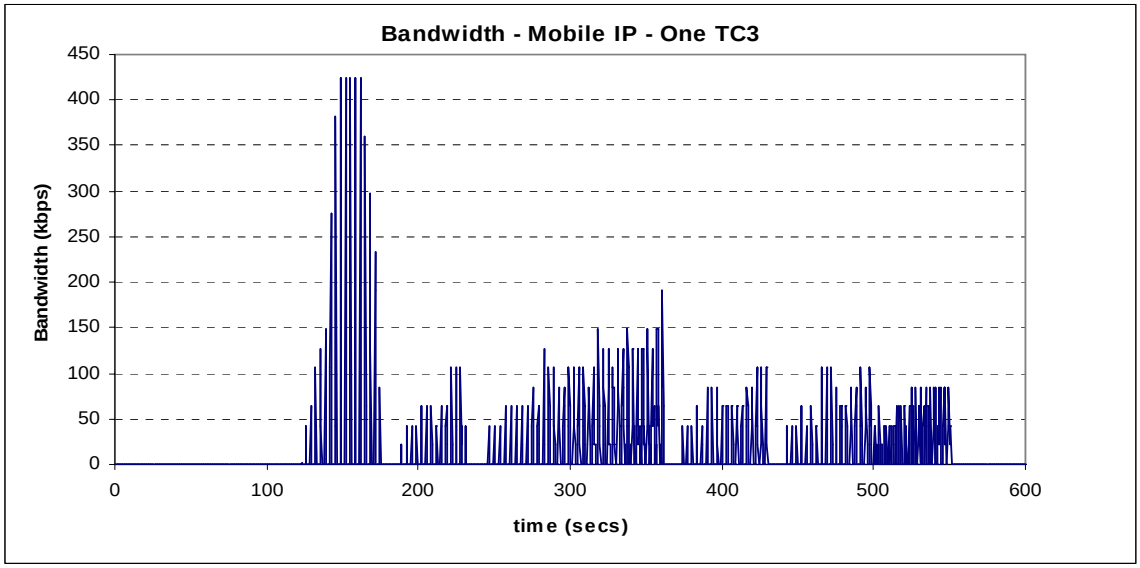
Επίσης, αχρείαστη είναι και η πληροφορία για το ποσοστό χαμένων πακέτων αφού το TCP λειτουργεί με επιβεβαιώσεις (πακέτα ACK) με αποτέλεσμα να αποτελεί ένα

αξιόπιστο πρωτόκολλο με το οποίο δεν χάνονται δεδομένα. Παρά το γεγονός όμως ότι δεν χάνεται πληροφορία, τα πακέτα που χάνονται σε ένα περιβάλλον ασύρματου δικτύου και πρέπει να επαναμεταδοθούν οδηγούν, όπως έχουμε ήδη πει, στην μείωση του ρυθμού μεταφοράς.

Η επιβάρυνση σηματοδοσίας είναι ελάχιστη αφού τα πακέτα TCP που μεταδίδονται έχουν μέγεθος 1000bytes οπότε η προσθήκη άλλων 20bytes κατά την ενθυλάκωση δε δημιουργεί τόσο μεγάλο πρόβλημα όπως στη περίπτωση πακέτων UDP τα οποία έχουν μέγεθος 48bytes.

Ο μέσος χρόνος διαπομπής στη περίπτωση αυτή είναι 7.1 δευτερόλεπτα.

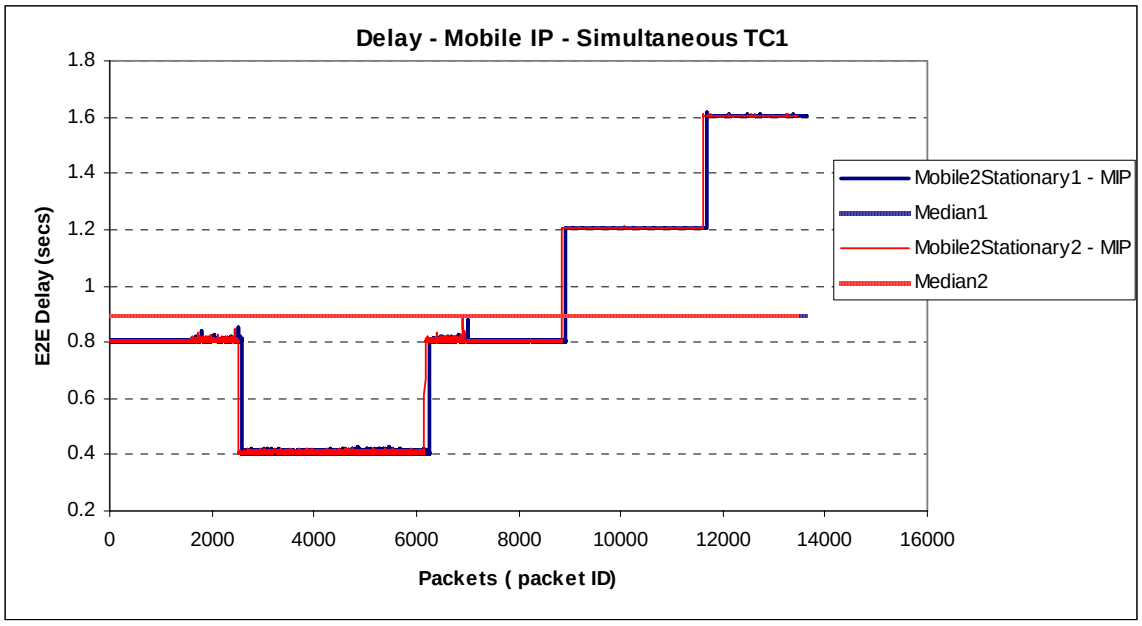
Τέλος, αναφέρουμε ότι δεν τρέξαμε κάποιο σενάριο με κίνηση TC3 και μηχανισμό κινητικότητας SIP, γιατί οι TCP συνδέσεις κόβονται όταν αλλάζει η IP διεύθυνση κάποιου χρήστη.



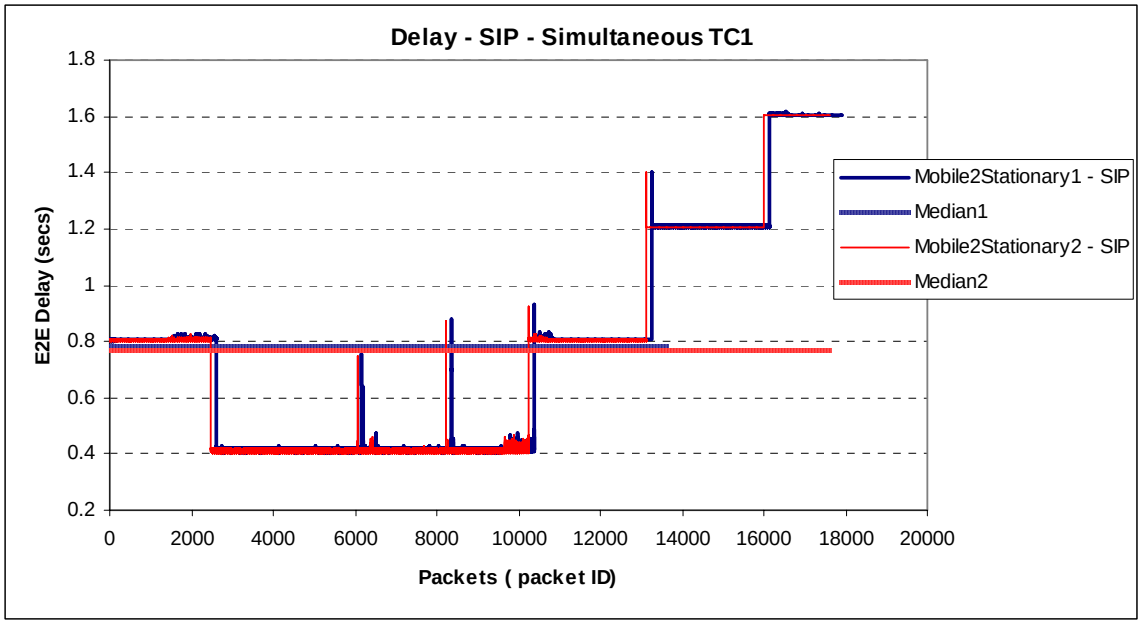
Σχήμα 36: Ρυθμός μετάδοσης σε FTP κίνηση.

5.4 Σενάρια 6 – 7

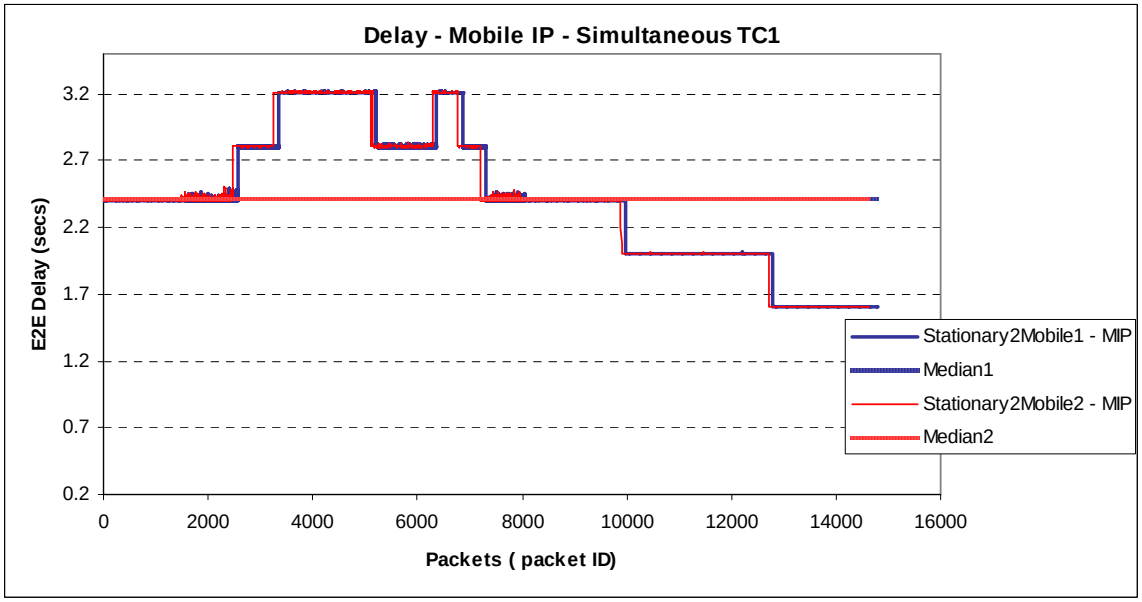
Στο σενάριο αυτό εγκαθιδρύονται δύο ταυτόχρονες συνδέσεις τύπου TC1. Όπως φαίνεται από τις πιο κάτω γραφικές παραστάσεις (βλέπε σχήμα 37 και σχήμα 38) που απεικονίζουν την καθυστέρηση των πακέτων στη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό κόμβο UA2, δεν παρατηρούμε μεγάλες διαφορές στις τιμές διότι τα πακέτα ακολουθούν πάντα τη συντομότερη διαδρομή προς τον προορισμό και στις δύο περιπτώσεις (MIP,SIP).



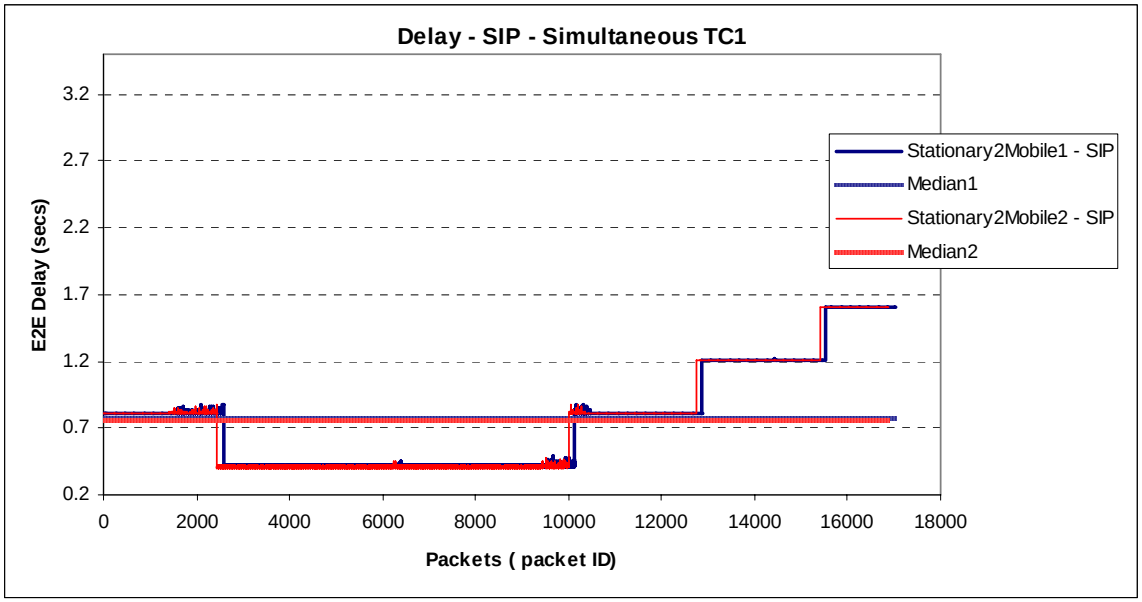
Σχήμα 37: Καθυστέρηση σε σενάρια ταυτόχρονης TC1 – Κινητός προς σταθερό – Mobile IP.



Σχήμα 38: Καθυστέρηση σε σενάρια ταυτόχρονης TC1 – Κινητός προς σταθερό – SIP.



Σχήμα 39: Καθυστέρηση σε σενάρια ταυτόχρονης TC1 – Σταθερός προς κινητό – Mobile IP.



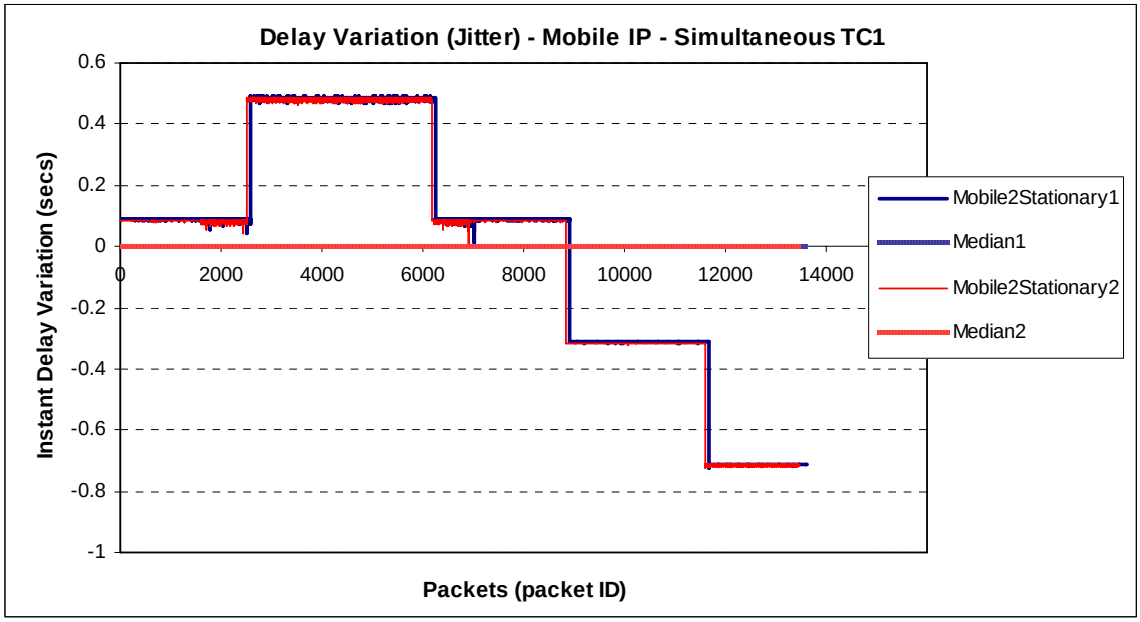
Σχήμα 40: Καθυστέρηση σε σενάρια ταυτόχρονης TC1 – Σταθερός προς κινητό – SIP.

Όσον αφορά την αντίστροφη ζεύξη από το σταθερό κόμβο προς τον κινητό κόμβο (βλέπε σχήμα 39 και σχήμα 40) παρατηρούμε μεγαλύτερες στιγμιαίες τιμές στην περίπτωση του Mobile IP από ότι στο SIP. Συγκεκριμένα, η αυξημένη καθυστέρηση που έχουμε στο Mobile IP οφείλεται κατά κύριο λόγο στην αποστολή δεδομένων από τον UA2 (ακίνητος κόμβος) στον UA1 (κινητός κόμβος) μέσω του οικείου πράκτορα του UA1. Η κίνηση του UA1 σε ξένα υποδίκτυα προκαλεί την ενθυλάκωση των πακέτων καθώς αυτά περνούν από τον οικείο πράκτορα του UA1 κάτι που αυξάνει και την επιβάρυνση του δικτύου (βλέπε σχήμα 39).

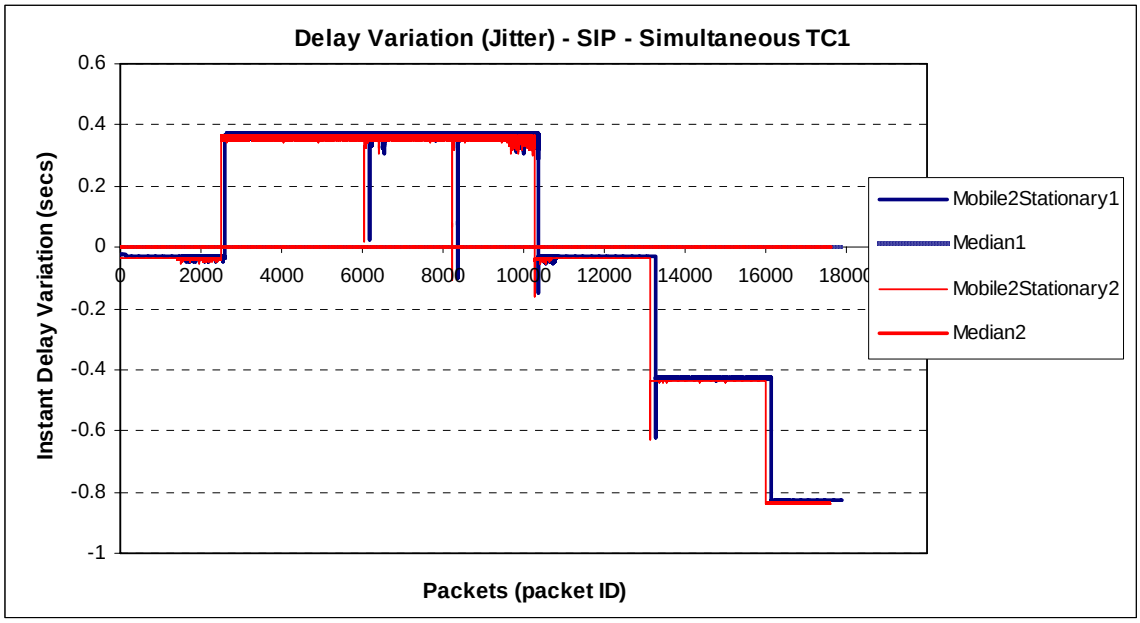
Από τις γραφικές παραστάσεις παρατηρούμε επίσης τη χαρακτηριστική κλιμάκωση καθώς ο κινούμενος χρήστης μπαινοβγαίνει σε ξένα υποδίκτυα.

Όσον αφορά τη διακύμανση καθυστέρησης (jitter), στο SIP (βλέπε σχήμα 42) μετρήσαμε παρόμοιες τιμές όπως και στο Mobile IP (βλέπε σχήμα41) όσον αφορά στην περίπτωση της ζεύξης από το κινητό προς τον σταθερό κόμβο. Όσον αφορά τη ζεύξη του σταθερού προς τον κινητό κόμβο παρατηρούμε μικρότερες καθυστερήσεις στο SIP (βλέπε σχήμα 44) από ότι στο Mobile IP (βλέπε σχήμα 43). Αυτό είναι αναμενόμενο και γίνεται καλύτερα αντιληπτό αν σκεφτούμε το σενάριο (σε Mobile IP) στο οποίο ένας κινούμενος

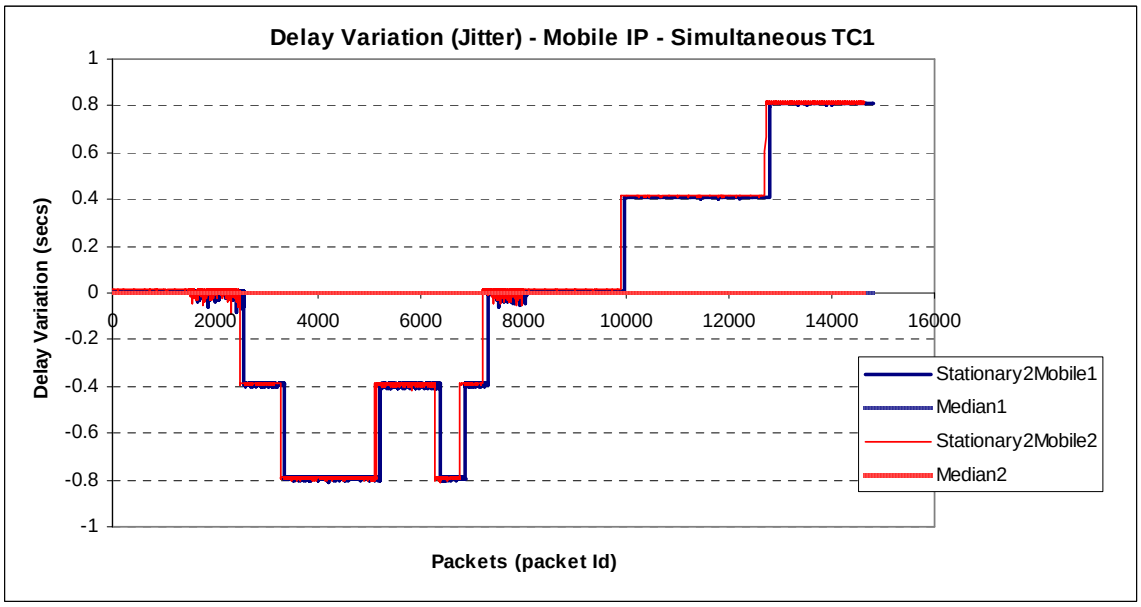
χρήστης πηγαινοέρχεται από το οικείο του δίκτυο σε ένα ξένο υποδίκτυο. Στην περίπτωση αυτή η καθυστέρηση παράδοσης πακέτων θα είναι πολύ μεγαλύτερη στην περίπτωση που ο χρήστης βρίσκεται στην ξένη περιοχή. Θα παρατηρούσαμε έτσι μια συνεχή εναλλαγή ανάμεσα σε μεγάλες και μικρές καθυστερήσεις πράγμα που ισοδυναμεί με μεγαλύτερη διακύμανση (jitter) από αυτήν που θα παρατηρούσαμε σε ένα πανομοιότυπο σενάριο SIP.



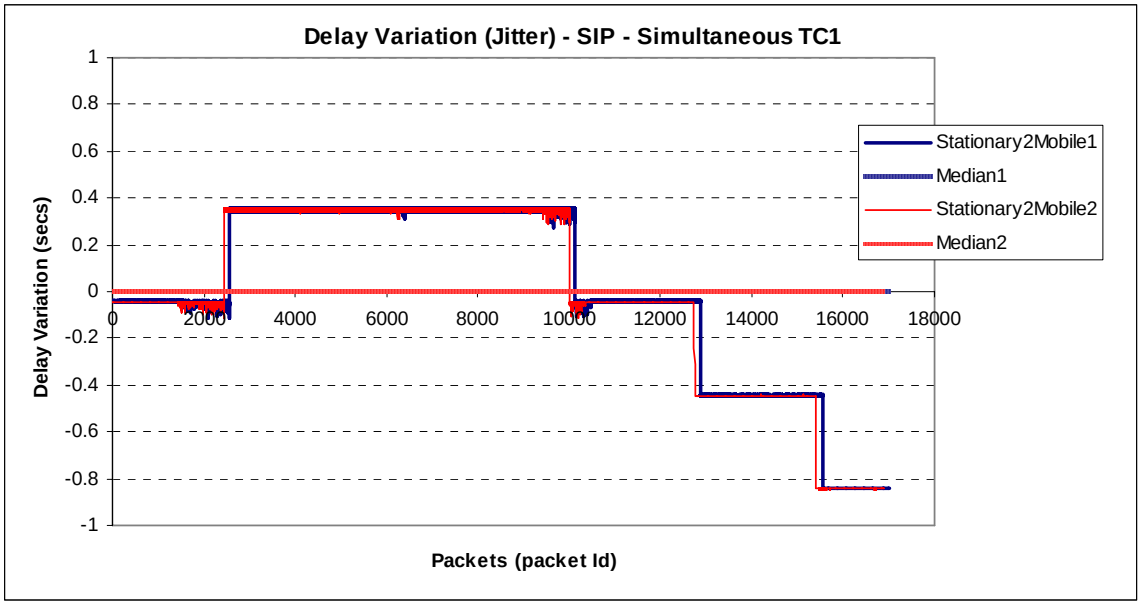
Σχήμα 41: Διακύμανση καθυστέρησης σε σενάρια ταυτόχρονης TC1 - Κινητός προς σταθερό - Mobile IP.



Σχήμα 42: Διακύμανση καθυστέρησης σε σενάρια ταυτόχρονης TC1 - Κινητός προς σταθερό – SIP.

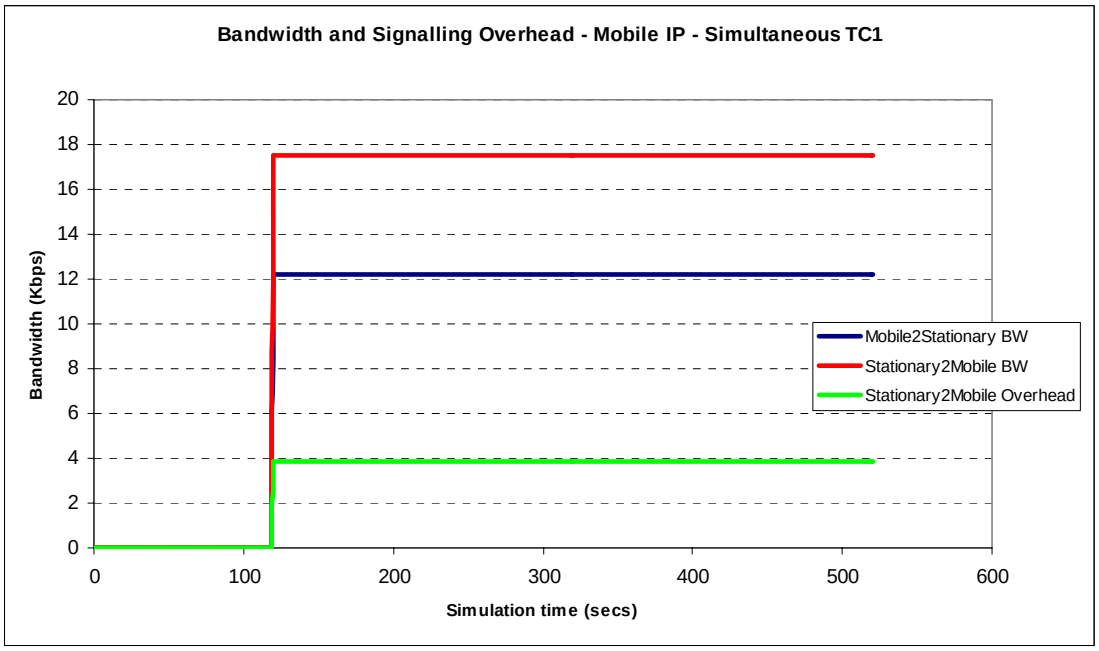


Σχήμα 43: Διακύμανση καθυστέρησης σε σενάρια ταυτόχρονης TC1 – Σταθερός προς κινητό - Mobile IP.

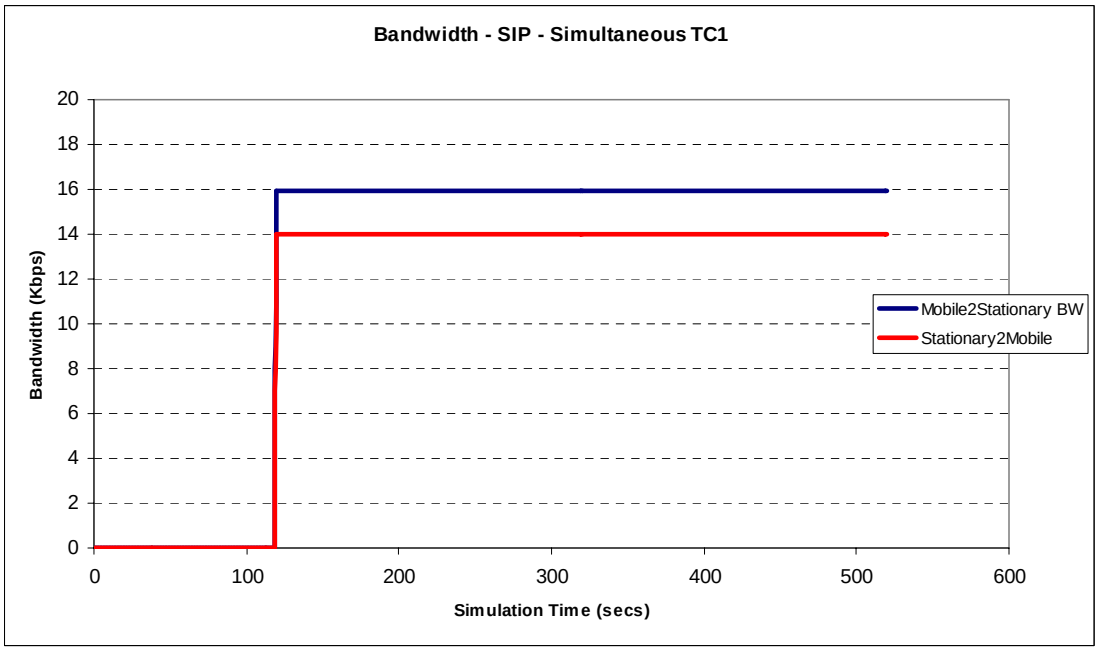


Σχήμα 44: Διακύμανση καθυστέρησης σε σενάρια ταυτόχρονης TC1 – Σταθερός προς κινητό – SIP.

Όσον αφορά στο σενάριο 6 που εμπλέκει το Mobile IP, υπάρχει επιβάρυνση λόγω σηματοδοσίας, κάτι που δεν συμβαίνει στο SIP όπως είδαμε και στην παράγραφο 5.1. Στα παρακάτω σχήματα (σχήμα 45 και σχήμα 46) παρουσιάζεται ο ρυθμός μετάδοσης ανά ζεύξη σε συνδυασμό με την επιβάρυνση σε Kbps για το Mobile IP καθώς και ο ρυθμός μετάδοσης για το SIP για τη μια μόνο σύνδεση μιας και τα ίδια αποτελέσματα ισχύουν και για την άλλη σύνδεση.



Σχήμα 45: Ρυθμός μετάδοσης και επιβάρυνση σηματοδοσίας σε σενάριο ταυτόχρονων συνδέσεων TC1 – Mobile IP.



Σχήμα 46: Ρυθμός μετάδοσης σε σενάριο ταυτόχρονων συνδέσεων TC1 – SIP.

Παρατηρώντας τα πιο πάνω σχήματα μπορούμε να διαπιστώσουμε ισχύουν περίπου τα ίδια συμπεράσματα όπως και στην περίπτωση μιας σύνδεσης TC1 που αναλύθηκε διεξοδικά στη παράγραφο 5.1.

Υπάρχουν όμως και κάποιες μικρές διαφορές κυρίως όσον αφορά το Mobile IP όπως το γεγονός ότι ο συνολικός μέσος ρυθμός πληροφορίας πάνω στη ζεύξη από τον σταθερό κόμβο UA2 προς το UA1 (κόκκινη παράσταση στο σχήμα 45) είναι αισθητά μικρότερος από τον αντίστοιχο συνολικό μέσο ρυθμό πληροφορίας στη περίπτωση μιας σύνδεσης TC1 (βλέπε σχήμα 24). Αυτό πιθανώς να οφείλεται στην αυξημένη επιβάρυνση του δικτύου λόγω της ταυτόχρονης μετάδοσης δυο συνδέσεων πάνω από το ίδιο κανάλι.

Όσον αφορά στο μέσο χρόνο διαπομπής, στο SIP είναι περίπου 7.5 δευτερόλεπτα δηλαδή ελάχιστα μεγαλύτερος από το μέσο χρόνο διαπομπής που είχαμε στην περίπτωση μιας σύνδεσης TC1 ενώ στο Mobile IP είναι 6.4 δευτερόλεπτα.

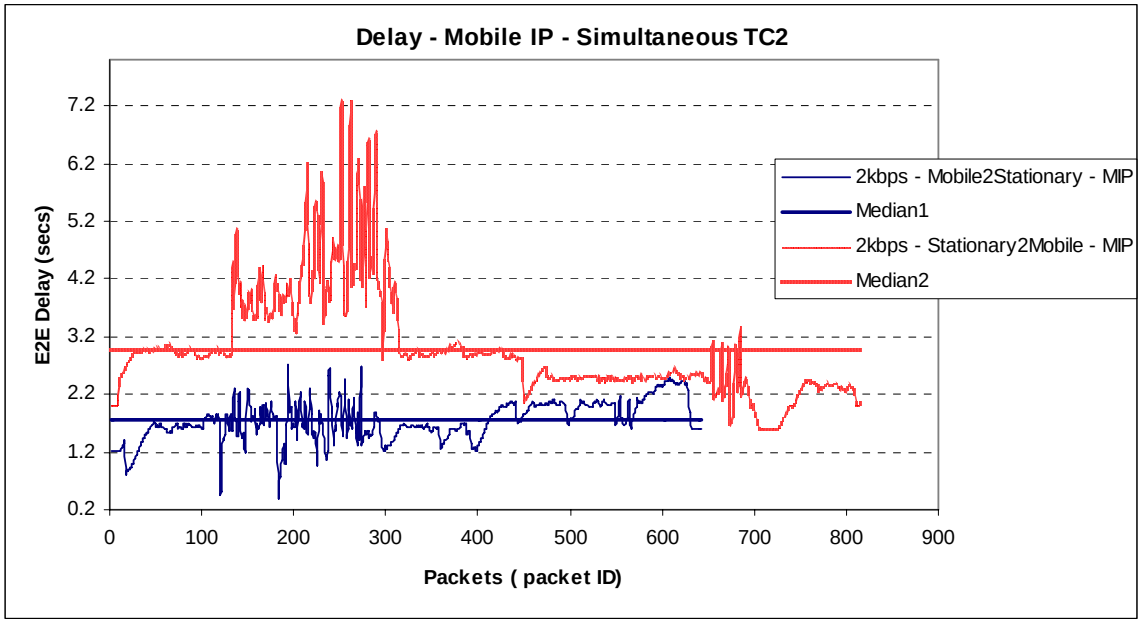
Όσον αφορά τη πρώτη σύνδεση, το ποσοστό απώλειας πακέτων στο Mobile IP είναι 23.75% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 14.9% για την αντίθετη ζεύξη. Για τη δεύτερη σύνδεση έχουμε ποσοστό απώλειας πακέτων 23.97% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 15.04% για την αντίθετη ζεύξη. Οι τιμές αυτές είναι αισθητά μεγαλύτερες από αυτές που ίσχυαν στην περίπτωση μιας σύνδεσης TC1 (βλέπε §5.1) και μπορούν να δικαιολογηθούν από το γεγονός ότι το κανάλι είναι περισσότερο επιβαρημένο λόγω των 2 ταυτόχρονων συνδέσεων.

Από την άλλη πλευρά το ποσοστό απώλειας πακέτων στο SIP είναι 0.078% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 12.02% για την αντίθετη ζεύξη. Για τη δεύτερη σύνδεση έχουμε ποσοστό απώλειας πακέτων 0.085% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 11.8% για την αντίθετη ζεύξη. Οι τιμές αυτές είναι περίπου οι ίδιες όπως στη περίπτωση απλής σύνδεσης TC1. Επίσης η ζεύξη σταθερού κόμβου προς κινητό κόμβο παρουσιάζει αισθητά μεγαλύτερες απώλειες όπως αναφέραμε προηγουμένως.

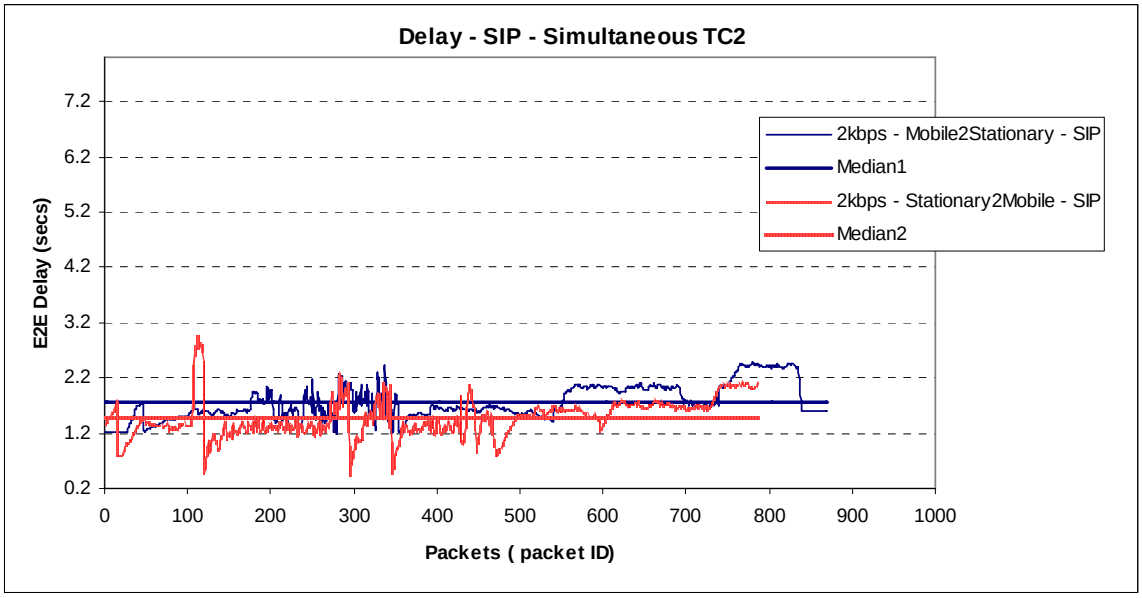
5.5 Σενάρια 8 – 9

Στο σενάριο αυτό εγκαθιδρύονται δύο ταυτόχρονες συνδέσεις της κλάσης TC2 μεταξύ των δύο κόμβων UA1 και UA2 όπου ο κάθε κόμβος κάνει μια αίτηση προς τον άλλο για ενεργοποίηση ροής video (video streaming). Οι δύο συνδέσεις ξεκινούν την ίδια χρονική στιγμή.

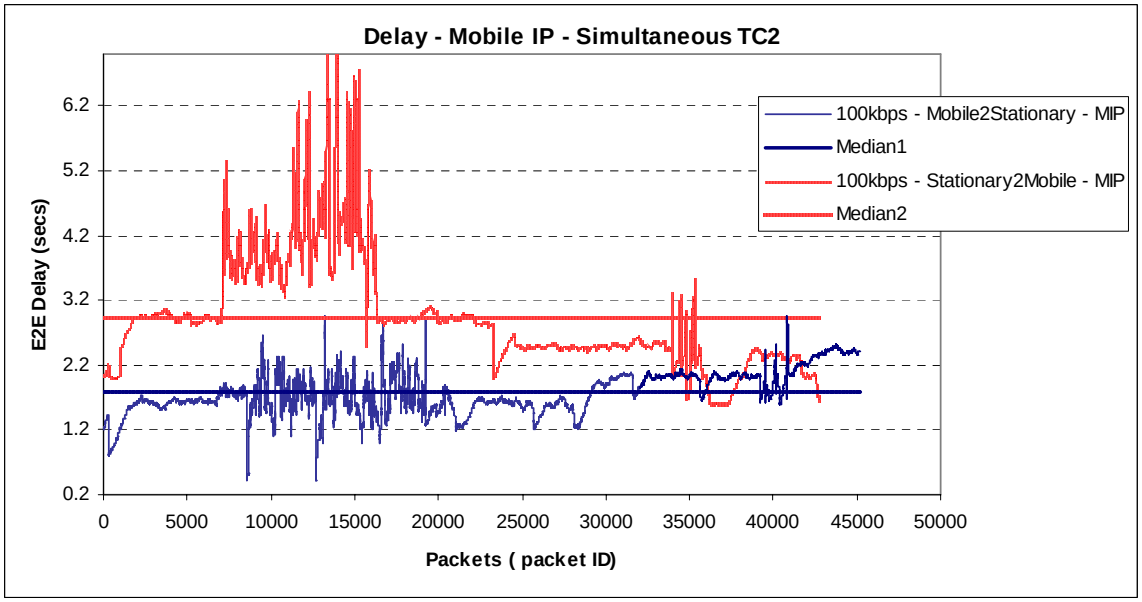
Όπως φαίνεται από τις γραφικές παραστάσεις που ακολουθούν, η καθυστέρηση που εμφανίζεται στην περίπτωση του SIP είναι πολύ πιο μικρή από αυτήν που παρατηρείται στο Mobile IP τόσο στην περίπτωση του downlink (100Kbps) όσο και στην περίπτωση του uplink (2Kbps). Αν συγκρίνουμε προσεκτικά τα σχήμα 47 – 50 θα δούμε ότι η μέση τιμή της καθυστέρησης στο Mobile IP είναι σχεδόν διπλάσια από αυτήν του SIP.



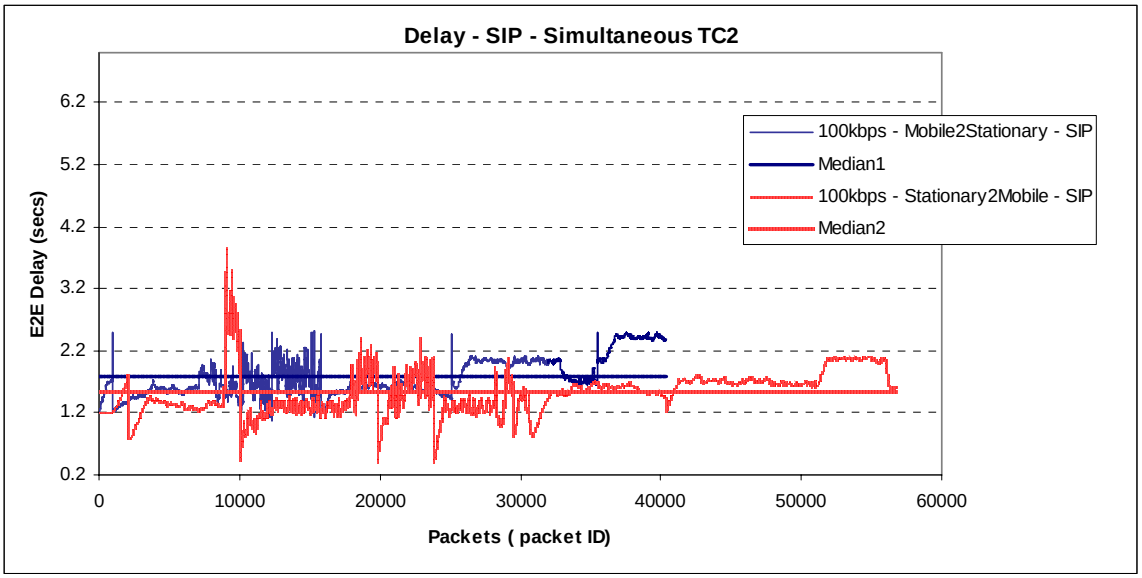
Σχήμα 47: Καθυστέρηση σε σενάρια ταυτόχρονης TC2 – Uplink (2Kbps) από κινητό προς σταθερό και το ανάποδο – Mobile IP.



Σχήμα 48: Καθυστέρηση σε σενάρια ταυτόχρονης TC2 – Uplink (2Kbps) από κινητό προς σταθερό και το ανάποδο –SIP.

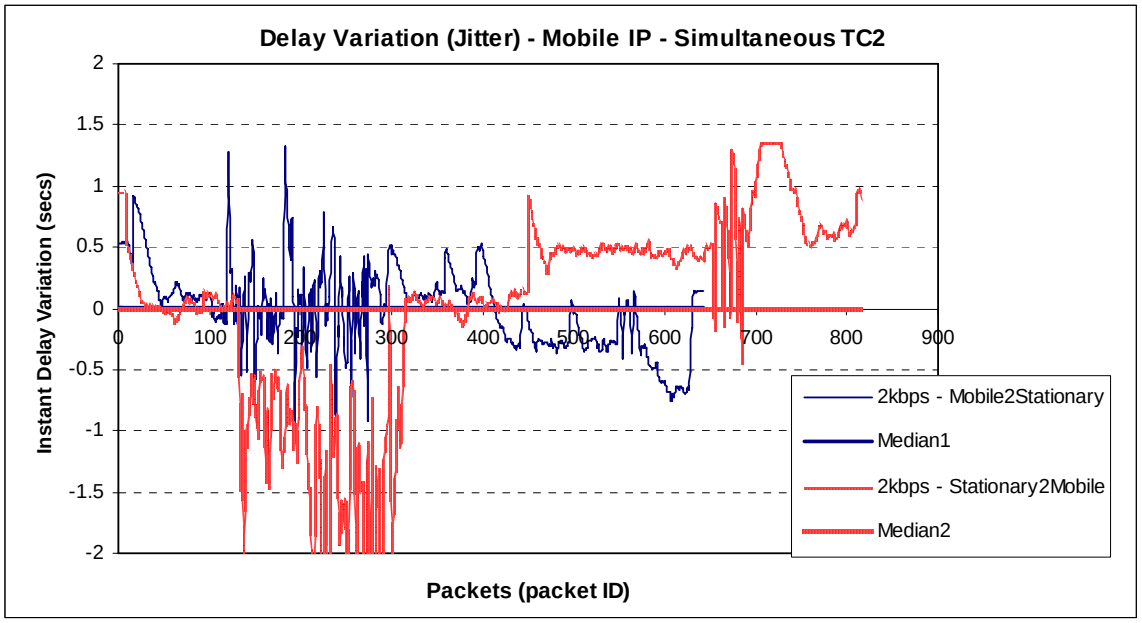


Σχήμα 49: Καθυστέρηση σε σενάρια ταυτόχρονης TC2 – Downlink (100Kbps) από κινητό προς σταθερό και το ανάποδο – Mobile IP.

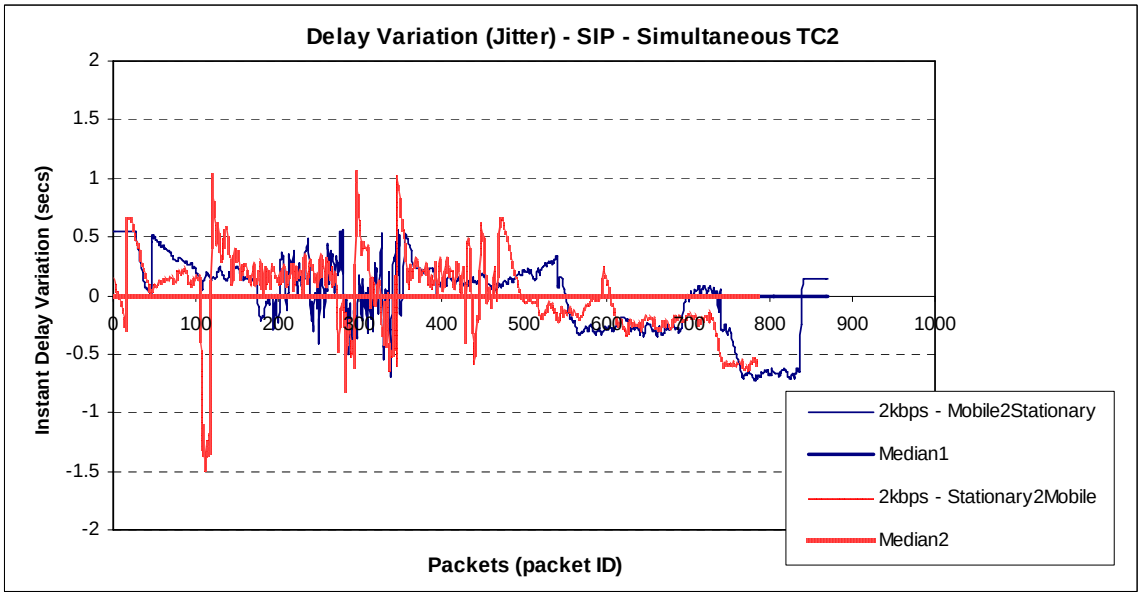


Σχήμα 50: Καθυστέρηση σε σενάρια ταυτόχρονης TC2 – Downlink (100Kbps) από κινητό προς σταθερό και το ανάποδο – SIP.

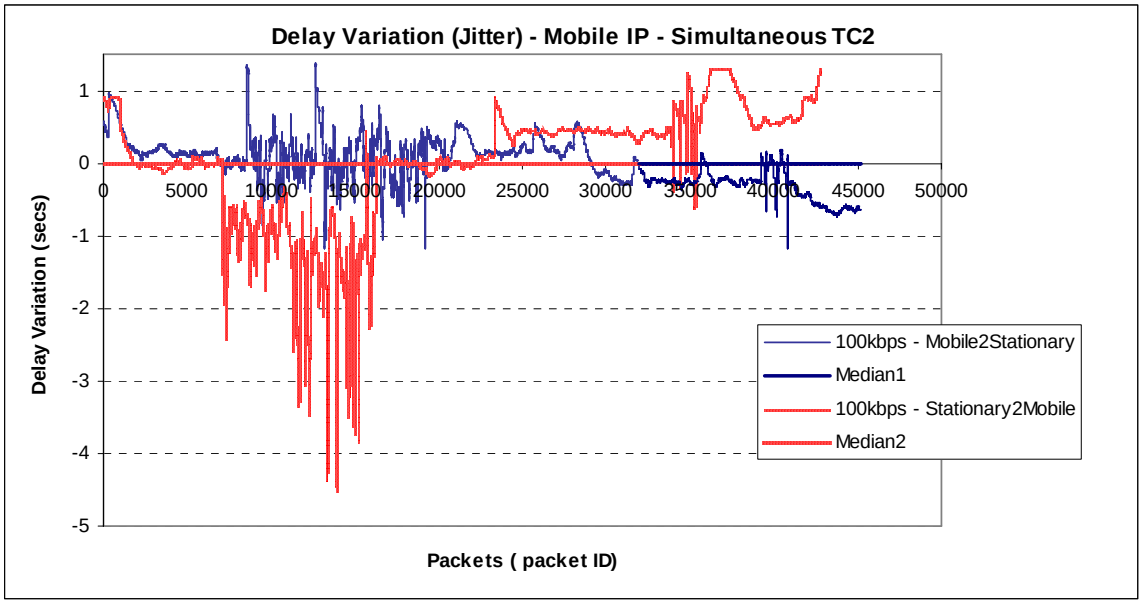
Επιπρόσθετα, στο Mobile IP υπάρχει μεγαλύτερη διακύμανση καθυστέρησης (βλέπε σχήμα 51 – 54). Κάτι τέτοιο κάνει το Mobile IP ακατάλληλο για εφαρμογές πραγματικού χρόνου και ιδιαίτερα video streaming.



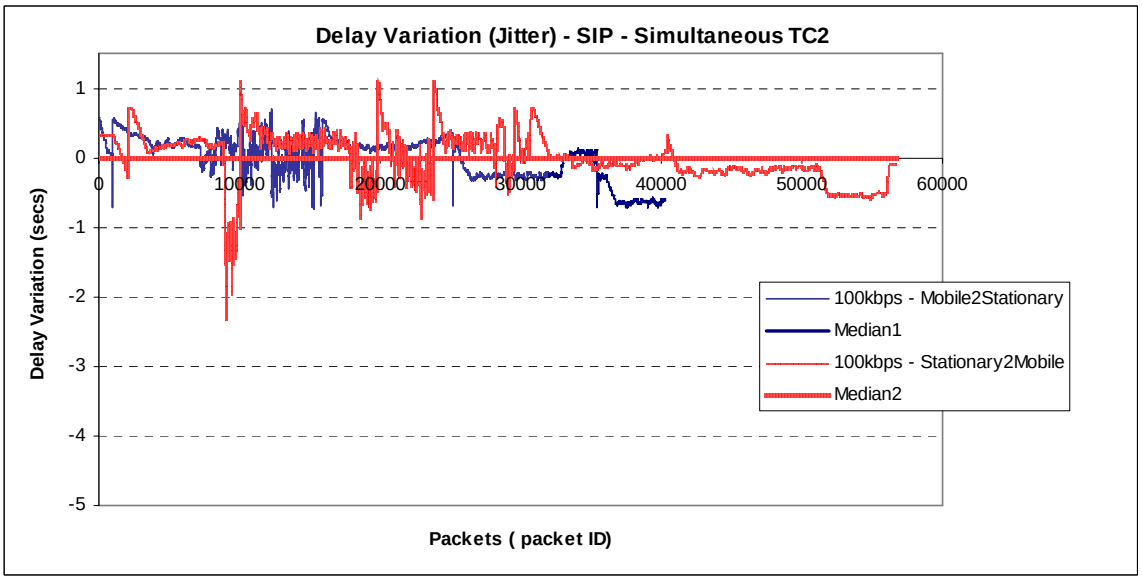
Σχήμα 51: Διακύμανση καθυστέρησης σε σενάρια ταυτόχρονης TC2 – Uplink (2Kbps) από κινητό προς σταθερό και το ανάποδο – Mobile IP.



Σχήμα 52: Διακύμανση καθυστέρησης σε σενάρια ταυτόχρονης TC2 – Uplink (2Kbps) από κινητό προς σταθερό και το ανάποδο – SIP.



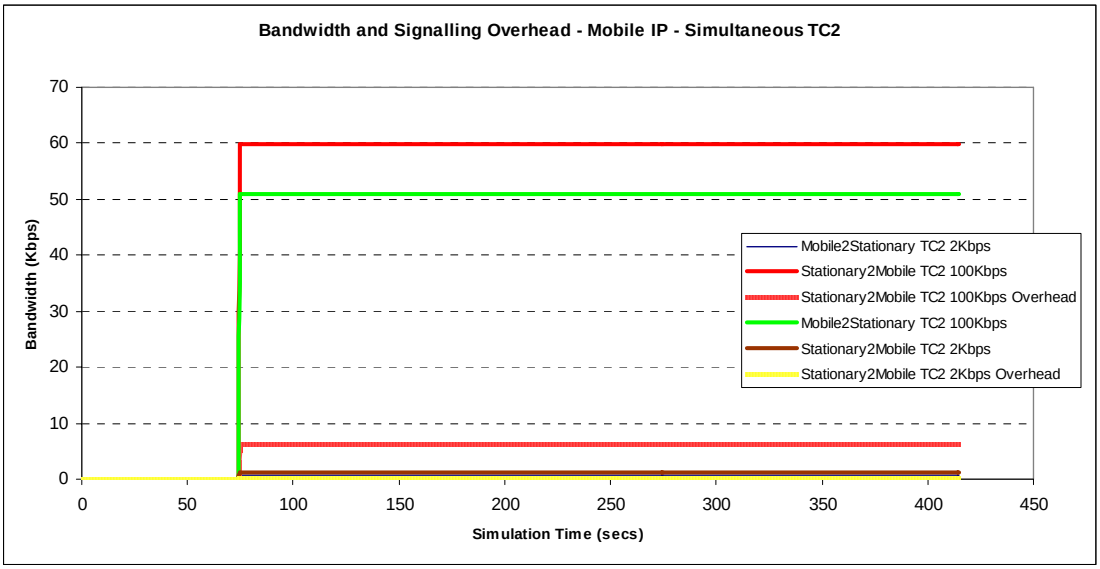
Σχήμα 53: Διακύμανση καθυστέρησης σε σενάρια ταυτόχρονης TC2 – Downlink (100Kbps) από κινητό προς σταθερό και το ανάποδο – Mobile IP.



Σχήμα 54: Διακύμανση καθυστέρησης σε σενάρια ταυτόχρονης ταυτόχρονης TC2 – Downlink (100Kbps) από κινητό προς σταθερό και το ανάποδο - SIP.

Τέλος, μια άλλη σημαντική παρατήρηση είναι ότι έχουμε υψηλότερη μέση τιμή καθυστέρησης (τόσο στο downlink όσο και στο uplink) σε σχέση με αυτή που είχαμε στο

σενάριο με μονή TC2. Αυτό οφείλεται στην αυξημένη επιβάρυνση του δικτύου λόγω της ταυτόχρονης μετάδοσης δυο ροών video.



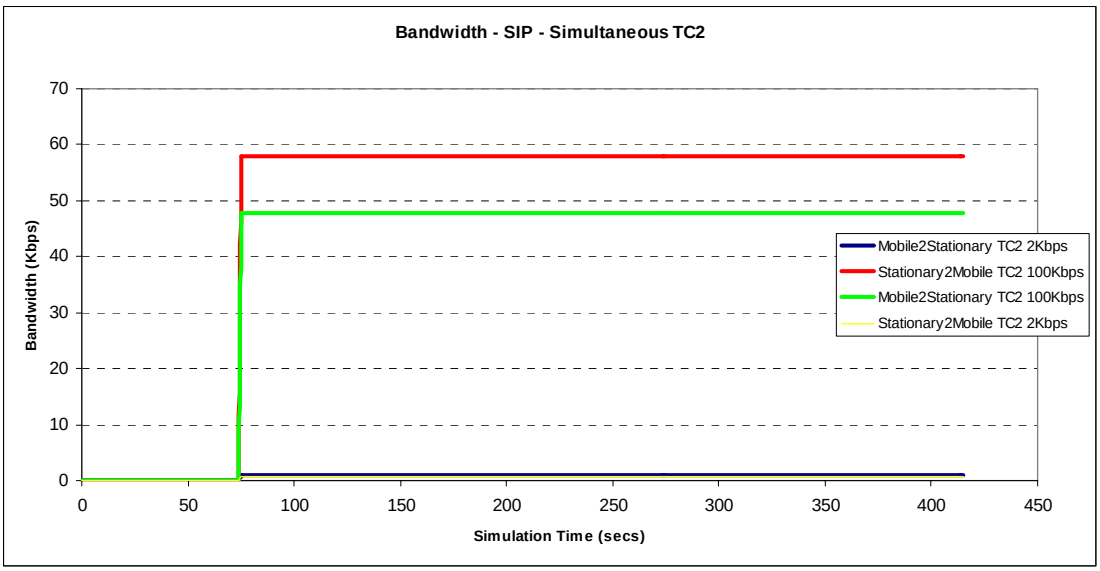
Σχήμα 55: Ρυθμός μετάδοσης και επιβάρυνση σηματοδοσίας σε σενάριο ταυτόχρονων συνδέσεων TC2 – Mobile IP.

Παρατηρώντας το πιο πάνω σχήμα (βλέπε σχήμα 55) μπορούμε να διαπιστώσουμε ισχύουν περίπου τα ίδια συμπεράσματα όπως και στην περίπτωση μιας σύνδεσης TC2 που αναλύθηκε διεξοδικά στην παράγραφο 5.2 για το Mobile IP.

Υπάρχουν όμως και κάποιες μικρές διαφορές κυρίως όσον αφορά στο Mobile IP όπως το γεγονός ότι ο συνολικός μέσος ρυθμός πληροφορίας πάνω στη ζεύξη από το σταθερό κόμβο UA2 προς το UA1 (κόκκινη παράσταση στο σχήμα 55) καθώς και η μέση επιβάρυνση (σε Kbps) έχουν αισθητά μικρότερες τιμές από τις αντίστοιχες τιμές στην περίπτωση μιας σύνδεσης TC2 (βλέπε σχήμα 34). Αυτό πιθανώς να οφείλεται στην αυξημένη επιβάρυνση του δικτύου λόγω της ταυτόχρονης μετάδοσης δυο συνδέσεων ροής video πάνω από το ίδιο κανάλι.

Επίσης μπορούμε να επισημάνουμε ότι ο συνολικός μέσος ρυθμός μετάδοσης πληροφορίας στην περίπτωση που η σύνδεση των 100Kbps (downlink) βρίσκεται πάνω στη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό κόμβο UA2, είναι μικρότερος από τον αντίστοιχο μέσο ρυθμό στην περίπτωση που η σύνδεση των 100Kbps βρίσκεται

στη ζεύξη από το σταθερό κόμβο UA2 στον κινητό κόμβο UA1 που πιθανώς να οφείλεται στο γεγονός ότι καθώς ο κινητός κόμβος αλλάζει περιοχές και κάνει διαπομπές από τον παλιό σταθμό βάσης στο νέο, χάνει περισσότερα πακέτα με αποτέλεσμα να μειώνεται ο μέσος ρυθμός πληροφορίας.



Σχήμα 56: Ρυθμός μετάδοσης σε σενάριο ταυτόχρονων συνδέσεων TC2 – SIP.

Παρατηρώντας το πιο πάνω σχήμα (βλέπε σχήμα 56) για τη περίπτωση του SIP, μπορούμε να διαπιστώσουμε ότι ισχύουν τα ίδια συμπεράσματα όπως και στην περίπτωση του Mobile IP όσον αφορά στη σύγκριση με το σενάριο μιας κίνησης TC2 (βλέπε §5.2). Εκτός αυτού, μπορούμε να δούμε ότι ο μέσος ρυθμός μετάδοσης πληροφορίας στην περίπτωση της σύνδεσης των 100Kbps από το σταθερό κόμβο στον κινητό κόμβο (~58Kbps) στο SIP είναι μεγαλύτερος από το μέσο ρυθμό μετάδοσης καθαρής πληροφορίας στο Mobile IP (60 – 6.4 = 53.6Kbps), ενώ στην άλλη σύνδεση των 100Kbps από τον κινητό κόμβο προς το σταθερό κόμβο έχουμε μεγαλύτερο ρυθμό μετάδοσης καθαρής πληροφορίας στην περίπτωση του Mobile IP (51Kbps) έναντι του SIP (47.7Kbps).

Όσον αφορά το μέσο χρόνο διαπομπής, στο SIP είναι περίπου 4.8 δευτερόλεπτα δηλαδή ελάχιστα μεγαλύτερος από το μέσο χρόνο διαπομπής που είχαμε στην περίπτωση μιας σύνδεσης TC2 ενώ στο Mobile IP είναι 8.1 δευτερόλεπτα.

Όσον αφορά στην πρώτη σύνδεση, το ποσοστό απώλειας πακέτων στο Mobile IP είναι 64.25% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 (σύνδεση 2Kbps) και 51.28% (σύνδεση 100Kbps) για την αντίθετη ζεύξη. Για τη δεύτερη σύνδεση έχουμε ποσοστό απώλειας πακέτων 48.26% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 (σύνδεση 100Kbps) και 52.49% για την αντίθετη ζεύξη (σύνδεση 2Kbps). Οι τιμές αυτές είναι αισθητά μεγαλύτερες από αυτές που ίσχυαν στη περίπτωση μιας σύνδεσης TC2 (βλέπε §5.2) και μπορούν να δικαιολογηθούν από το γεγονός ότι το κανάλι είναι περισσότερο επιβαρημένο λόγω των 2 ταυτόχρονων συνδέσεων.

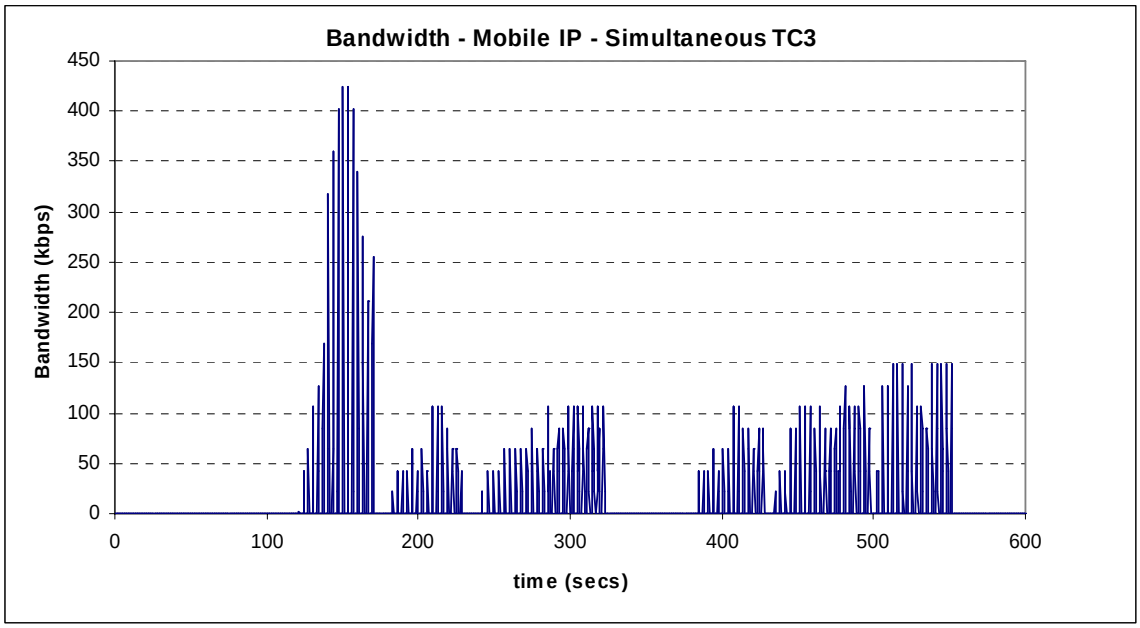
Από την άλλη πλευρά το ποσοστό απώλειας πακέτων στο SIP είναι 49.9% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 (σύνδεση 2Kbps) και 41.52% για την αντίθετη ζεύξη (σύνδεση 100Kbps). Για τη δεύτερη σύνδεση έχουμε ποσοστό απώλειας πακέτων 50.3% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 (σύνδεση 100Kbps) και 61.6% για την αντίθετη ζεύξη (σύνδεση 2Kbps). Οι τιμές αυτές είναι αισθητά μεγαλύτερες από τις αντίστοιχες στη περίπτωση απλής σύνδεσης TC2.

5.6 Σενάριο 10

Το σενάριο αυτό μοιάζει αρκετά με το σενάριο 5 (VBR) μόνο που στην συγκεκριμένη περίπτωση έχουμε δύο κινήσεις μεταβλητού ρυθμού, μία από και μία προς τον κινητό κόμβο. Ως γνωστόν ο ρυθμός δεδομένων δεν είναι σταθερός καθώς τέτοιου είδους εφαρμογές (π.χ. FTP) είναι ιδιαίτερα εκρηκτικές με μεταβλητό ρυθμό μετάδοσης. Εδώ πάλι μελετούμε μόνο το Mobile IP γιατί στο SIP θα έπρεπε να γίνεται διακοπή της κίνησης κάθε φορά που ο κινητός χρήστης αλλάζει περιοχή, επειδή σπάνε οι TCP συνδέσεις λόγω αλλαγής IP διεύθυνσης προορισμού και αποστολής μηνύματος re-INVITE, οπότε θα ήταν ανούσιο να το μελετήσουμε. Και τέλος, κατά τις διαπομπές, όπως είδαμε και στο σενάριο 5, ο ρυθμός μειώνεται σημαντικά γιατί γίνονται αναμεταδόσεις.

Γνωρίζοντας ότι η πληροφορία για καθυστέρηση, διακύμανση καθυστέρησης και ποσοστό χαμένων πακέτων είναι ανούσια για τέτοιου είδος εφαρμογές, παρακάτω

παρουσιάζεται μόνο ο ρυθμός δεδομένων και αυτός της επιβάρυνσης από το MIP λόγω ενθυλάκωσης.

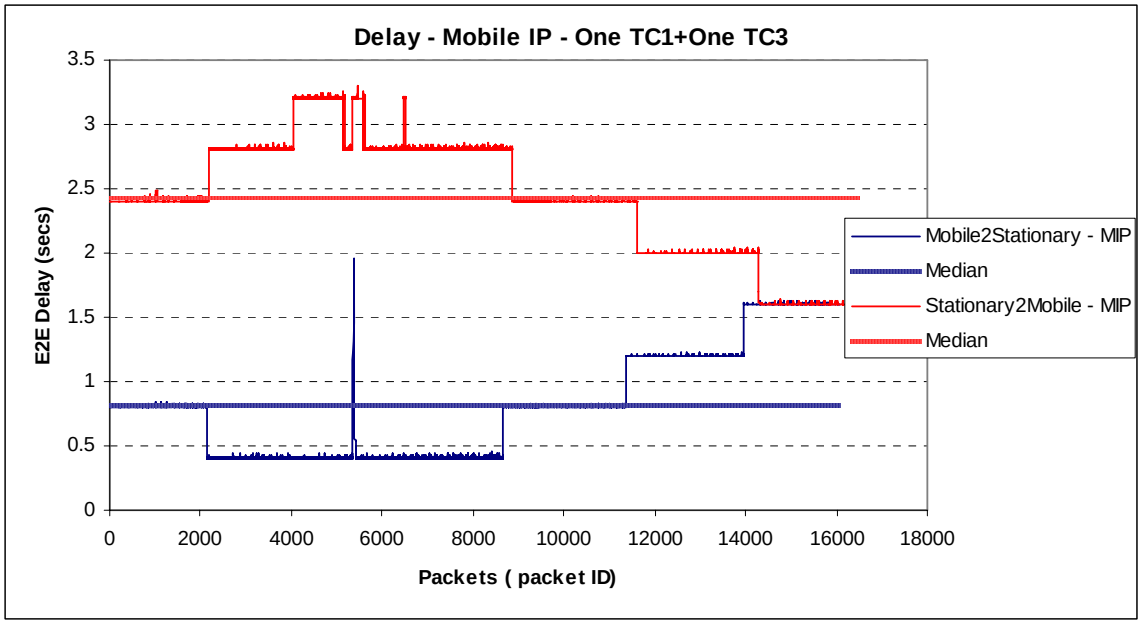


Σχήμα 57: Ρυθμός μετάδοσης και overhead σε FTP κίνηση.

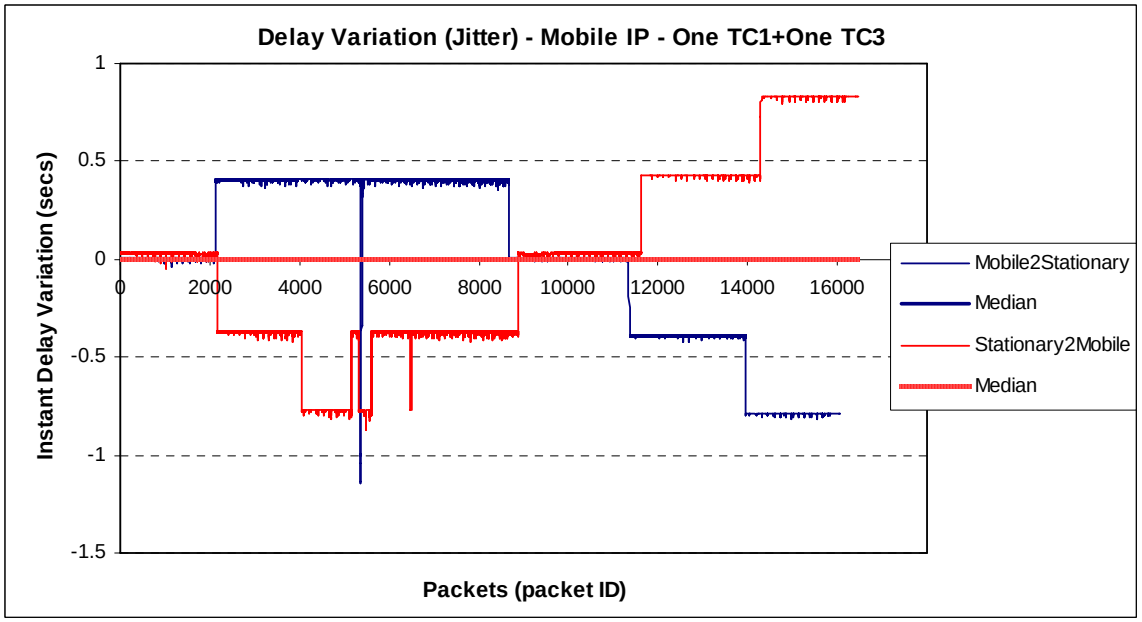
Στην περίπτωση αυτή όπως και στο σενάριο 5, επιβάρυνση σηματοδοσίας είναι ελάχιστη και ο μέσος χρόνος διαπομπής είναι 8.1 δευτερόλεπτα.

5.7 Σενάριο 11

Στο παρόν σενάριο εγκαθιδρύεται ταυτόχρονα με την κίνηση της κλάσης TC1 (π.χ. VoIP) και μια κίνηση της κλάσης TC3 (π.χ. FTP). Για ακόμη μια φορά παρατηρούμε μεγαλύτερη καθυστέρηση στα πακέτα που μεταδίδονται από τον UA2 στον UA1 (βλέπε σχήμα58). Αυτό οφείλεται στην τριγωνική δρομολόγηση που δημιουργείται στο Mobile IP όταν ο UA2 (σταθερός χρήστης) στέλνει πακέτα στον Οικείο Πράκτορα του UA1 (κινητός χρήστης) και αυτός στη συνέχεια τα προωθεί ενθυλακωμένα προς τον UA1.



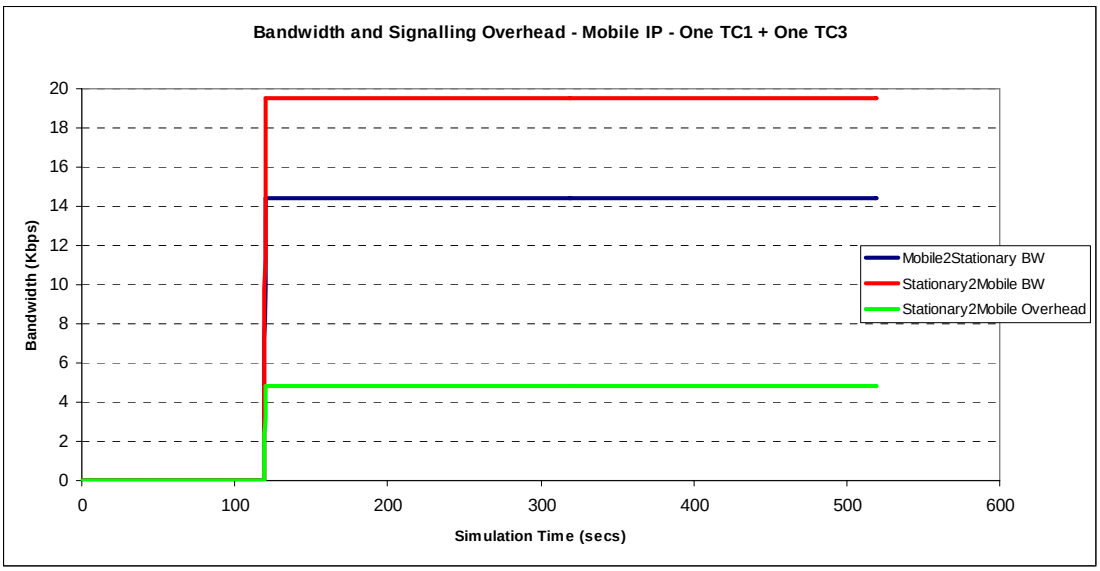
Σχήμα 58: Καθυστέρηση της κίνησης TC1 σε σενάρια TC1 και TC3 – Κινητός προς σταθερό και αντίστροφα – Mobile IP.



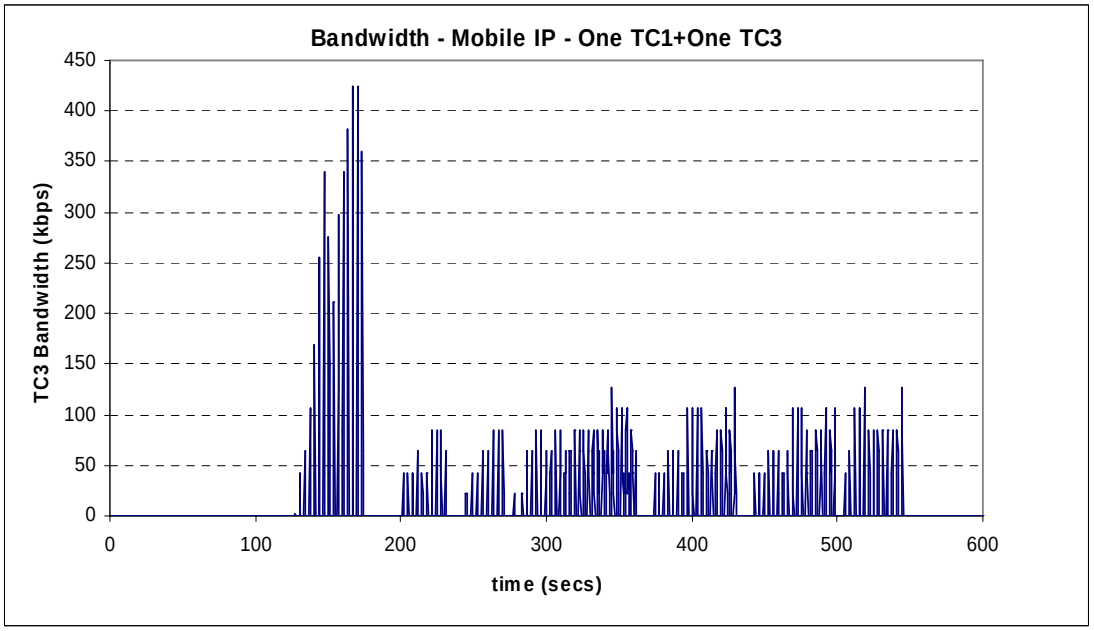
Σχήμα 59: Διακύμανση της καθυστέρηση της κίνησης TC1 σε σενάρια TC1 και TC3 – Κινητός προς σταθερό και αντίστροφα – Mobile IP.

Για την κίνηση FTP και τη συμπεριφορά της ως προς το ρυθμό μετάδοσης διαπιστώνουμε ότι μετά την αρχική προσπάθεια για μετάδοση σε υψηλούς ρυθμούς

παρατηρείται μια σημαντική μείωση του ρυθμού αυτού (βλέπε σχήμα 61). Αυτό, όπως έχουμε ξαναπεί, συμβαίνει λόγω της εκπνοής του χρόνου παραλαβής των πακέτων επιβεβαίωσης (πακέτα ACK – αλγόριθμος αργής αρχής) από την πλευρά του αποστολέα. Τέτοιες εκπνοές γίνονται συνήθως κατά τη διάρκεια διαπομπής από μια περιοχή σε άλλη.



Σχήμα 60: Ρυθμός μετάδοσης και επιβάρυνση σηματοδοσίας σε σενάριο μιας σύνδεσης TC1 και μιας σύνδεσης TC3.



Σχήμα 61: Ρυθμός μετάδοσης FTP κίνησης - Σενάριο TC1 και TC3.

Όσον αφορά στη γραφική παράσταση του μέσου ρυθμού μετάδοσης συνολικής πληροφορίας στη ζεύξη από τον σταθερό κόμβο UA2 προς το κινητό κόμβο UA1 (σχήμα 60), παρατηρούμε υπερβαίνει τον αντίστοιχο μέσο ρυθμό στη περίπτωση που είχαμε δύο ταυτόχρονες συνδέσεις TC1 (βλέπε §5.4 σχήμα 45) ενώ υπολείπεται του αντίστοιχου μέσου ρυθμού στην περίπτωση που είχαμε μόνο μια TC1 σύνδεση (βλέπε §5.1 σχήμα 24) κάτι που είναι αναμενόμενο μιας και στην πρώτη περίπτωση (ταυτόχρονες TC1) συνυπάρχουν καθ' όλη τη διάρκεια της σύνδεσης οι δύο κινήσεις κάτι που επιβαρύνει περισσότερο τις ζεύξεις ενώ στην δεύτερη περίπτωση (μια TC1) υπάρχει μόνο μια κίνηση. Στη περίπτωση που μελετούμε εδώ, υπάρχουν διαστήματα κατά τα οποία συνυπάρχουν δύο είδη κινήσεων – όταν έχουμε ριπές δεδομένων FTP – ενώ υπάρχουν διαστήματα κατά τα οποία υπάρχει μόνο η κίνηση TC1.

Η τιμή του μέσου χρόνου διαπομπής στη περίπτωση αυτή είναι 8.5 δευτερόλεπτα.

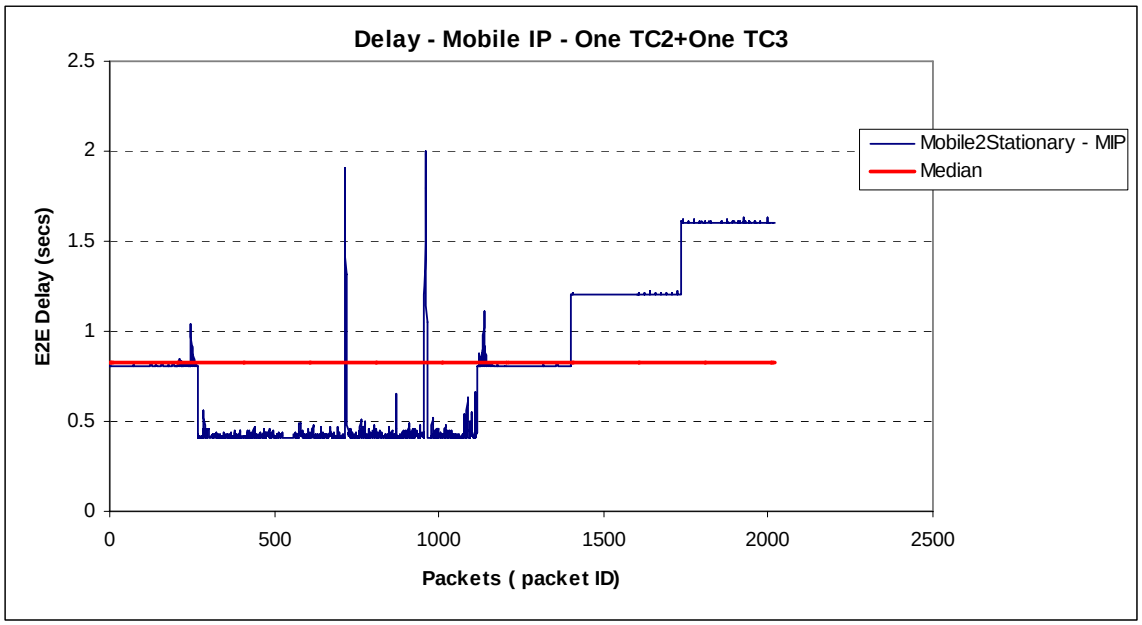
Όσον αφορά στη σύνδεση TC1, το ποσοστό απώλειας πακέτων είναι 10.1% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 7.97% για την αντίθετη ζεύξη. Οι τιμές αυτές είναι αισθητά μικρότερες από αυτές που ίσχυαν στη περίπτωση 2 ταυτόχρονων συνδέσεων TC1 (βλέπε §5.4) και μπορούν να δικαιολογηθούν από το γεγονός ότι στη περίπτωση των ταυτόχρονων συνδέσεων διακινείται μεγαλύτερη ποσότητα πληροφορίας οπότε είναι αυξημένη και η πιθανότητα απώλειας. Αν επιχειρήσουμε μια σύγκριση με την περίπτωση όπου είχαμε μόνο μια σύνδεση TC1 (βλέπε §5.1) θα διαπιστώσουμε ότι τα ποσοστά απώλειας πακέτων ήταν μικρότερα (κοντά στο 3% στη κάθε ζεύξη) από αυτά που υπολογίσαμε στην παράγραφο αυτή. Αυτό ήταν καθόλα αναμενόμενο μιας και στην περίπτωση που μελετούμε τώρα, ιδιαίτερα στη ζεύξη από σταθερό προς κινητό κόμβο διακινούνται περισσότερα πακέτα (αφού έχουμε και FTP σύνδεση) από ότι στην περίπτωση μιας σύνδεσης TC1 οπότε αυξημένη θα είναι και η πιθανότητα απώλειας πακέτων.

Από την άλλη πλευρά το ποσοστό απώλειας πακέτων που ανήκουν στη σύνδεση TC3 είναι μηδενικό μιας και τα χαμένα (ή μη επιβεβαιωμένα πακέτα) επαναμεταδίδονται. Λόγω όμως της μεγάλης απώλειας πακέτων γίνονται πολλές επαναμεταδόσεις των ιδίων

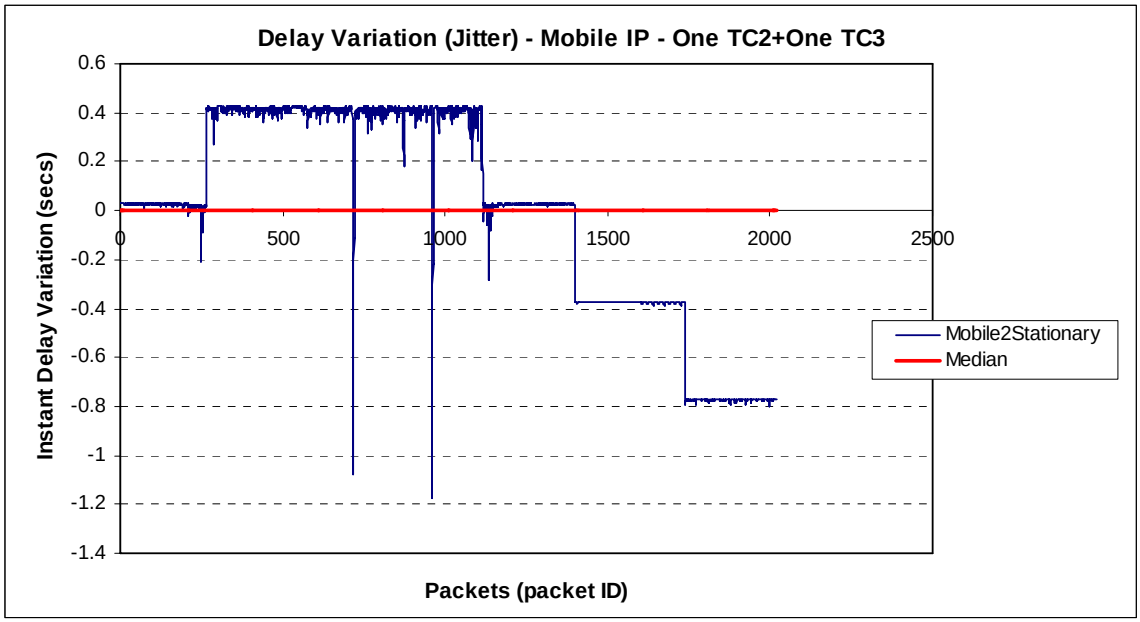
πακέτων με αποτέλεσμα να καθυστερεί αρκετά η μετάδοση ενός συγκεκριμένου όγκου πληροφορίας.

5.8 Σενάριο 12

Στο παρόν σενάριο εγκαθιδρύεται ταυτόχρονα με την κίνηση της κλάσης TC2 (π.χ. video streaming) και μια κίνηση της κλάσης TC3 (π.χ. FTP). Παρατηρούμε μεγαλύτερη καθυστέρηση στα πακέτα που μεταδίδονται από τον UA2 στον UA1 (βλέπε σχήμα 62). Αυτό οφείλεται στην τριγωνική δρομολόγηση που δημιουργείται στο Mobile IP όταν ο UA2 (σταθερός χρήστης) στέλνει πακέτα στον Οικείο Πράκτορα του UA1 (κινητός χρήστης) και αυτός στη συνέχεια τα προωθεί ενθυλακωμένα προς τον UA1.

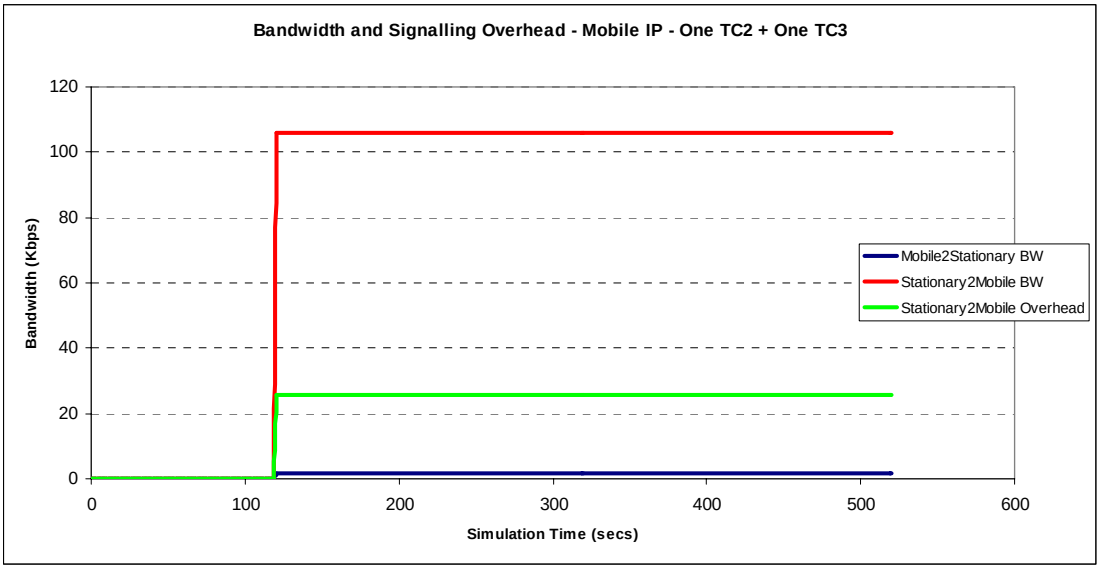


Σχήμα 62: Καθυστέρηση της κίνησης TC2 σε σενάριο TC2 και TC3 - Κινητός προς σταθερό και αντίστροφα – Mobile IP.

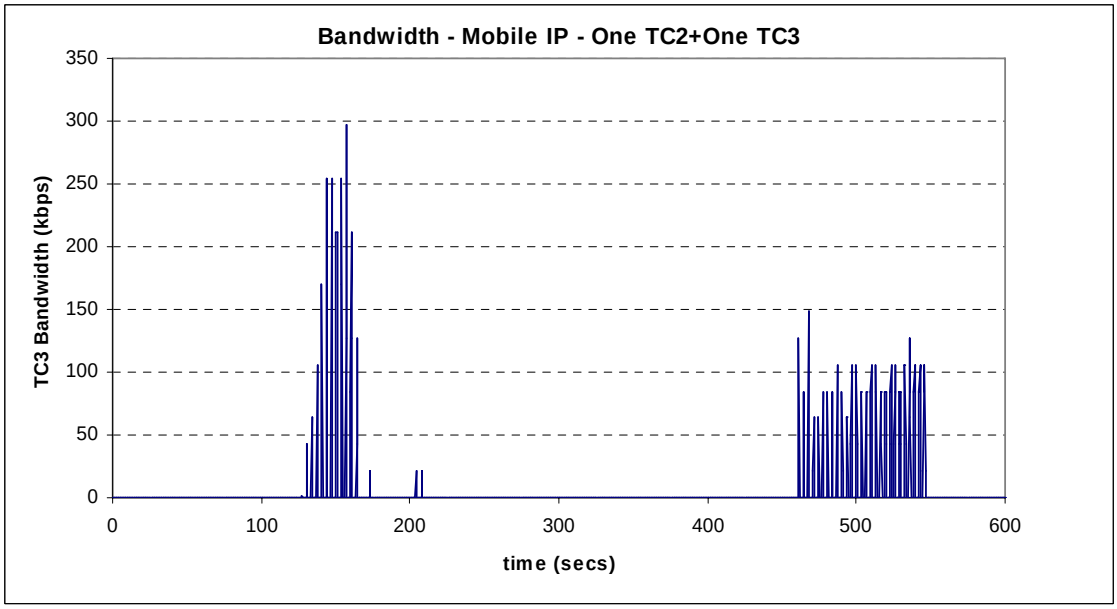


Σχήμα 63: Διακύμανση της καθυστέρηση της κίνησης TC2 σε σενάριο TC2 και TC3 - Κινητός προς σταθερό και αντίστροφα – Mobile IP.

Το ενδιαφέρον αποτέλεσμα που αντλούμε από αυτό το σενάριο είναι η ανικανότητα του Mobile IP να υποστηρίξει και τις δυο κινήσεις ταυτόχρονα. Όπως διακρίνουμε από το παρακάτω σχήμα (βλέπε σχήμα 65), ο ρυθμός μετάδοσης της FTP κίνησης είναι μηδενικός στο μεγαλύτερο μέρος του χρονικού διαστήματος στο οποίο οι δυο κινήσεις συνυπάρχουν. Αιτία για αυτό είναι το γεγονός ότι οι επιβεβαιώσεις (πακέτα ACK) που στέλνονται από τον παραλήπτη UA1 δεν φθάνουν στον UA2 με αποτέλεσμα αυτός να αναβάλλει συνεχώς την αποστολή πακέτων κίνησης FTP. Ο πιθανότερος λόγος που προκαλεί την απώλεια πακέτων ACK είναι η συμφόρηση που συμβαίνει στο ασύρματο μέρος του δικτύου εξαιτίας και της κίνησης TC2.



Σχήμα 64: Ρυθμός μετάδοσης και επιβάρυνση σηματοδοσίας σε σενάριο μιας σύνδεσης TC2 και μιας σύνδεσης TC3.



Σχήμα 65: Ρυθμός μετάδοσης FTP κίνησης - Σενάριο TC2 και TC3.

Όσον αφορά στη γραφική παράσταση του μέσου ρυθμού μετάδοσης συνολικής πληροφορίας στη ζεύξη από τον σταθερό κόμβο UA2 προς τον κινητό κόμβο UA1 (σχήμα 64), παρατηρούμε υπερβαίνει τον αντίστοιχο μέσο ρυθμό στην περίπτωση που είχαμε δύο ταυτόχρονες συνδέσεις TC2 (βλέπε §5.5 σχήμα 55) ενώ έχει περίπου τις ίδιες

τιμές με την περίπτωση που είχαμε μόνο μια TC2 σύνδεση (βλέπε §5.2 σχήμα 34) κάτι που είναι αναμενόμενο μιας και στην πρώτη περίπτωση (ταυτόχρονες TC2) συνυπάρχουν καθ' όλη τη διάρκεια της σύνδεσης οι δύο κινήσεις κάτι που επιβαρύνει περισσότερο τις ζεύξεις ενώ στην δεύτερη περίπτωση (σύγκριση με μια TC2) βλέπουμε παρόμοια συμπεριφορά λόγω του ότι στην περίπτωση που μελετάμε τώρα υπάρχει πολύ μεγάλο χρονικό διάστημα στο οποίο η σύνδεση FTP είναι ουσιαστικά αδρανής και στη ζεύξη υπάρχει μόνο η σύνδεση TC2.

Η τιμή του μέσου χρόνου διαπομπής στη περίπτωση αυτή είναι 3.9 δευτερόλεπτα.

Όσον αφορά τη σύνδεση TC2, το ποσοστό απώλειας πακέτων είναι 12.77% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 (σύνδεση uplink 2Kbps) και 18.33% για την αντίθετη ζεύξη (σύνδεση downlink 100Kbps). Οι τιμές αυτές είναι αισθητά μικρότερες από αυτές που ίσχυαν στη περίπτωση 2 ταυτόχρονων συνδέσεων TC2 (βλέπε §5.5) και μπορούν να δικαιολογηθούν από το γεγονός ότι στην περίπτωση των ταυτόχρονων συνδέσεων διακινείται μεγαλύτερη ποσότητα πληροφορίας οπότε είναι αυξημένη και η πιθανότητα απώλειας. Αν επιχειρήσουμε μια σύγκριση με την περίπτωση όπου είχαμε μόνο μια σύνδεση TC2 (βλέπε §5.2) θα διαπιστώσουμε ότι τα ποσοστά απώλειας και στις δύο ζεύξεις είναι σχεδόν ταυτόσημα (15% στο uplink και 18.42% downlink). Στην περίπτωση που μελετάμε εδώ, οι απώλειες στο downlink είναι ελαφρώς αυξημένες για το λόγο ότι υπάρχουν και δεδομένα από τη σύνδεση FTP που ταξιδεύουν στη ζεύξη αυτή.

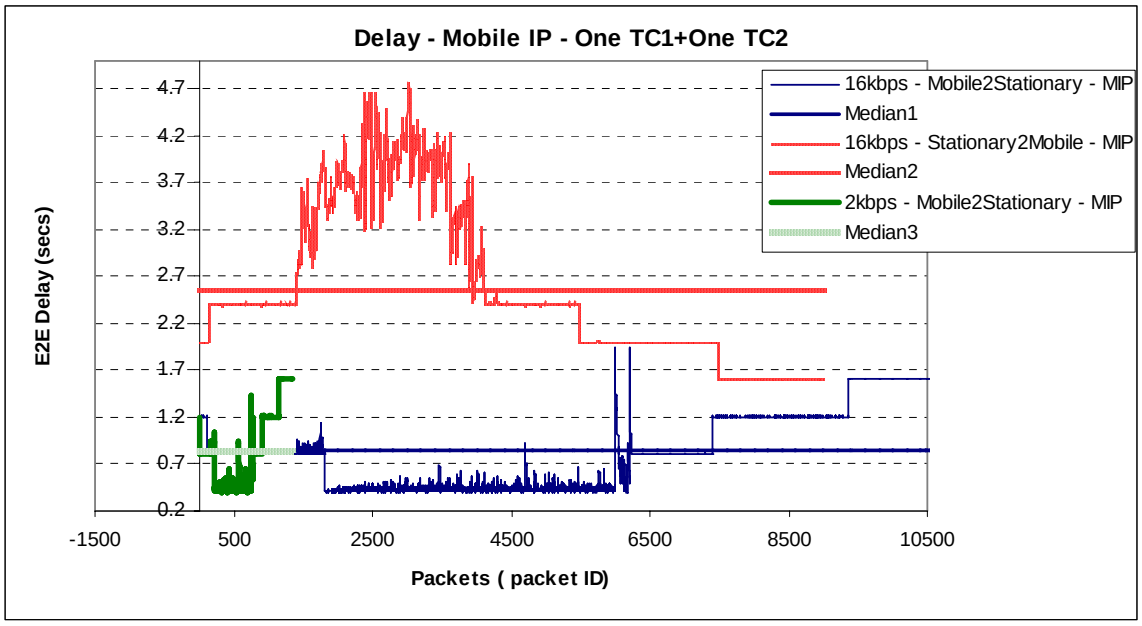
Από την άλλη πλευρά το ποσοστό απώλειας πακέτων που ανήκουν στη σύνδεση TC3 είναι μηδενικό όπως προαναφέραμε στη προηγούμενο παράγραφο.

5.9 Σενάρια 13 – 14

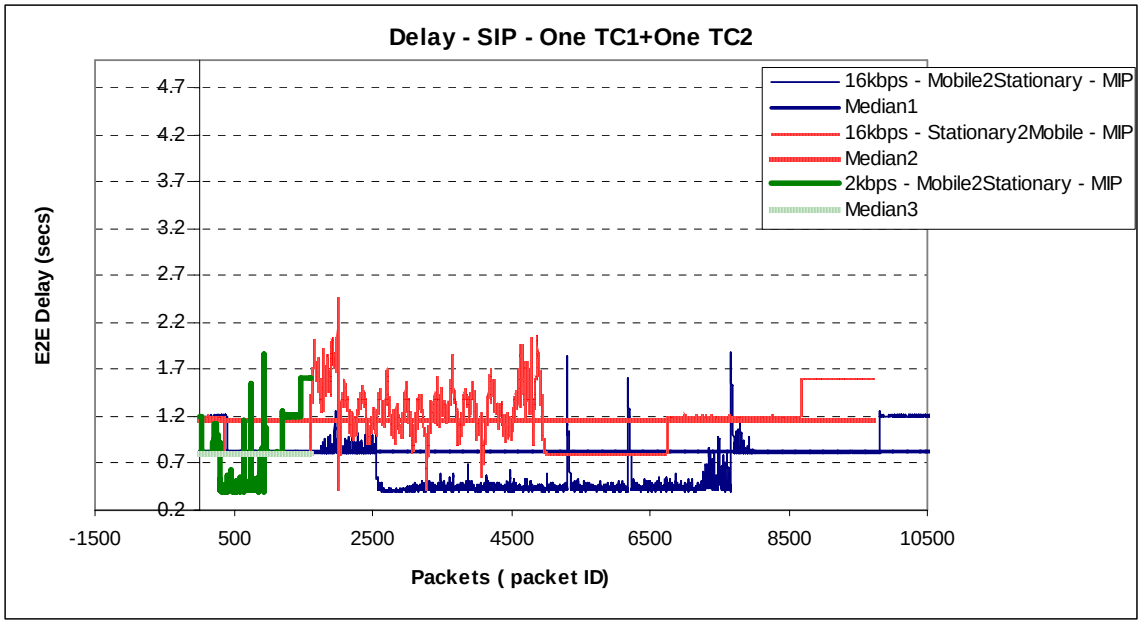
Στα τελευταία αυτά σενάρια εγκαθιδρύεται ταυτόχρονα με μια κίνηση της κλάσης TC1 και μια κίνηση της κλάσης TC2. Όσον αφορά την καθυστέρηση, παρατηρούμε ότι και πάλι στην περίπτωση του Mobile IP που ο UA2 (σταθερός κόμβος) στέλνει στον UA1 (κινητός κόμβος) η καθυστέρηση που εισάγεται είναι πολύ μεγαλύτερη σε σχέση με αυτή

που έχουμε στο SIP. Η καθυστέρηση είναι ίδια και στις δυο περιπτώσεις μόνο όταν πρόκειται για αποστολή πακέτων από τον UA1 (κινητός κόμβος) στον UA2 (σταθερός κόμβος) (βλέπε σχήμα 66 – 69). Αυτή η παρατήρηση αφορά τόσο την κίνηση της κλάσης TC1 όσο και την κίνηση της κλάσης TC2.

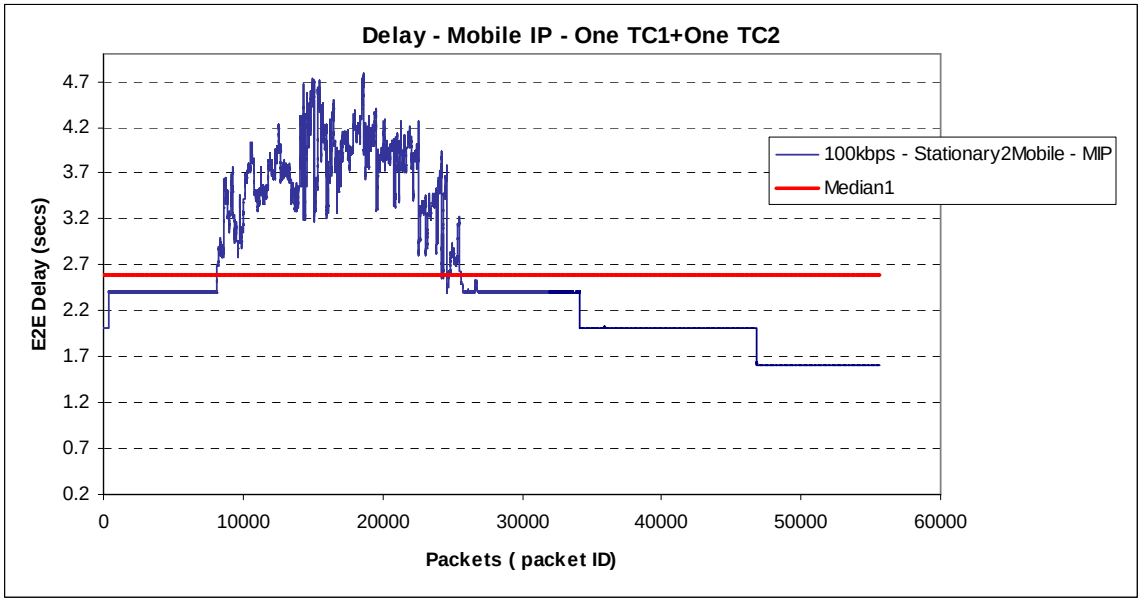
Επίσης, στα σενάρια αυτά διακρίνουμε μια μικρή αύξηση της μέσης τιμής της καθυστέρησης, τόσο στο Mobile IP όσο και στο SIP, στην κατεύθυνση ροής πακέτων της κλάσης TC1 από τον UA2 προς τον UA1 σε σχέση με αυτή που είχαμε στα σενάρια 1 και 2. Αιτία για αυτό είναι η αποστολή σε αυτή την κατεύθυνση του video streaming.



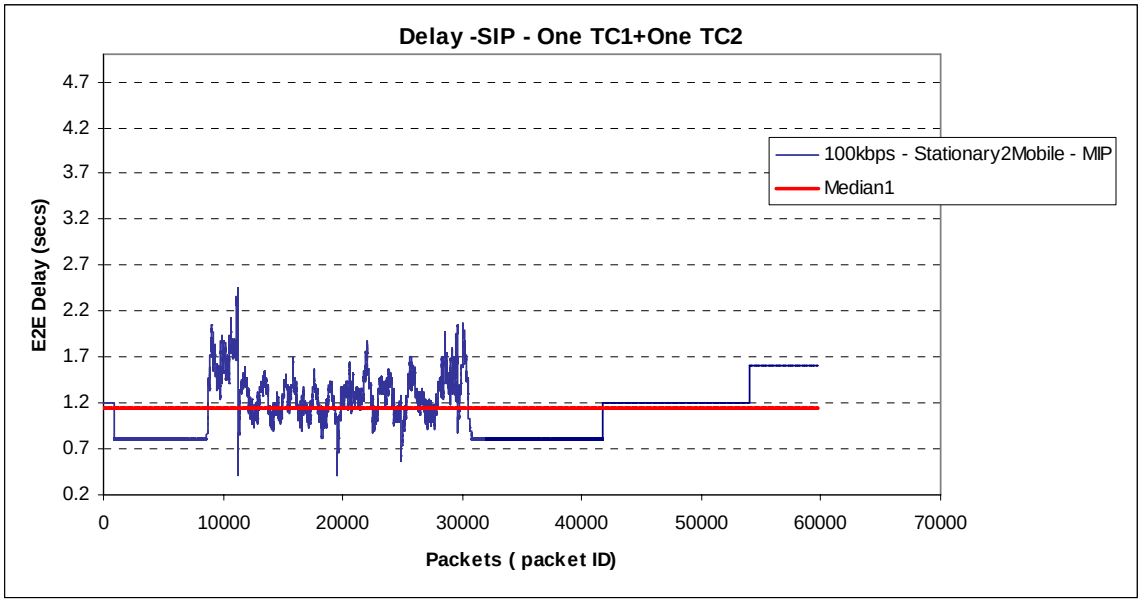
Σχήμα 66: Καθυστέρηση σε σενάρια TC1 και TC2 – Κίνηση TC1 και uplink (2Kbps) της TC2 – Mobile IP.



Σχήμα 67: Καθυστέρηση σε σενάρια TC1 και TC2 – Κίνηση TC1 και uplink (2Kbps) της TC2 – SIP.

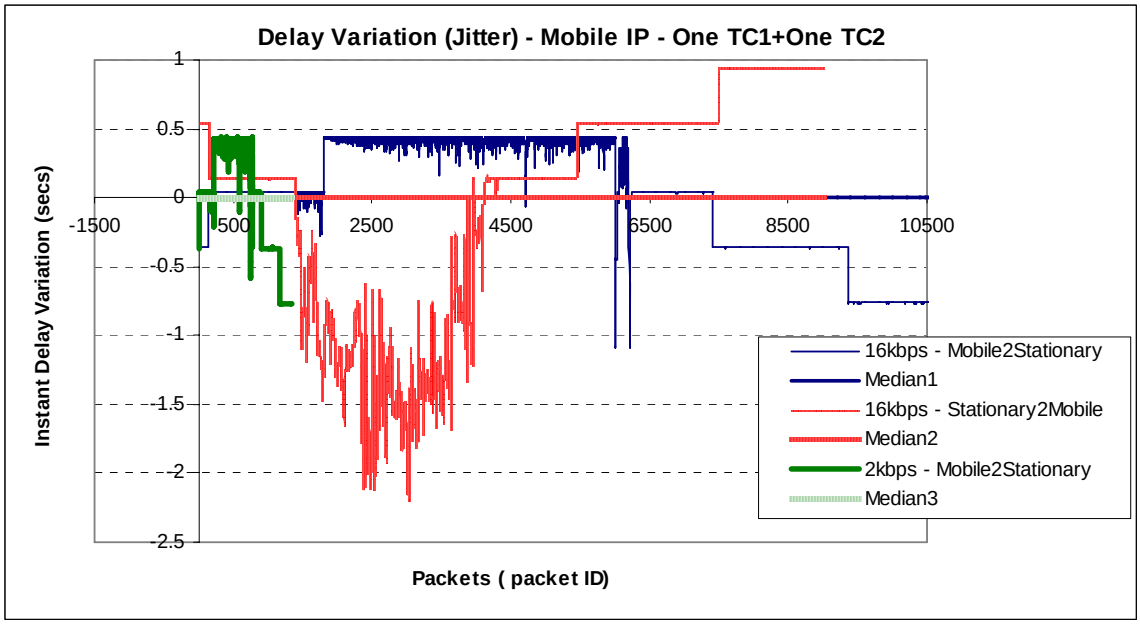


Σχήμα 68: Καθυστέρηση σε σενάρια TC1 και TC2 – Downlink (100Kbps) της TC2 – Mobile IP.

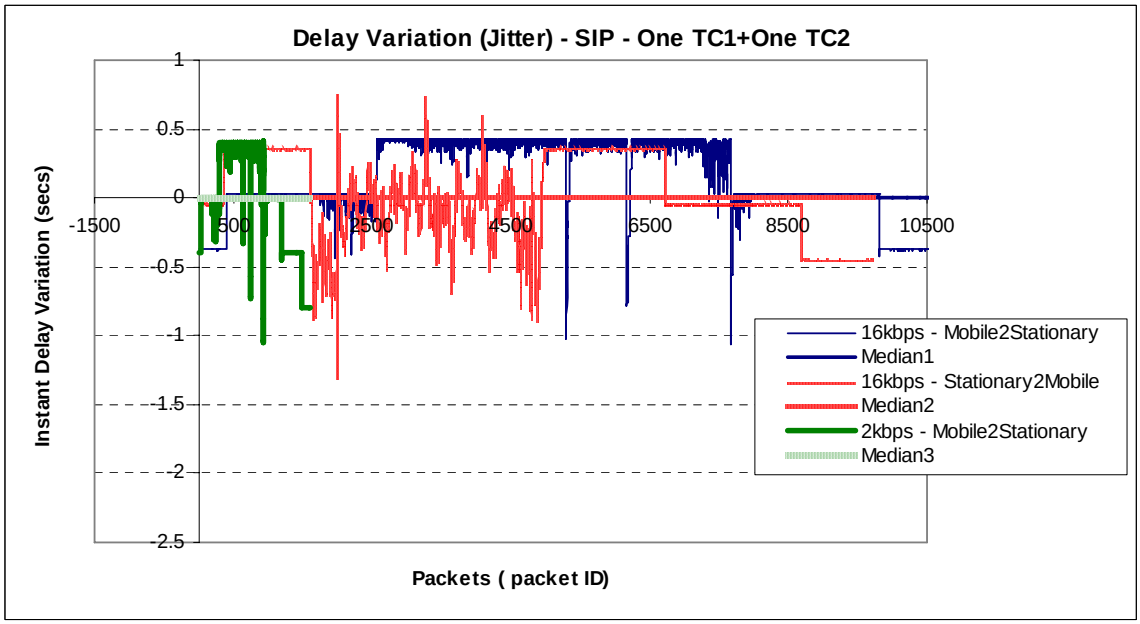


Σχήμα 69: Καθυστέρηση σε σενάρια TC1 και TC2 – Downlink (100Kbps) της TC2 – SIP.

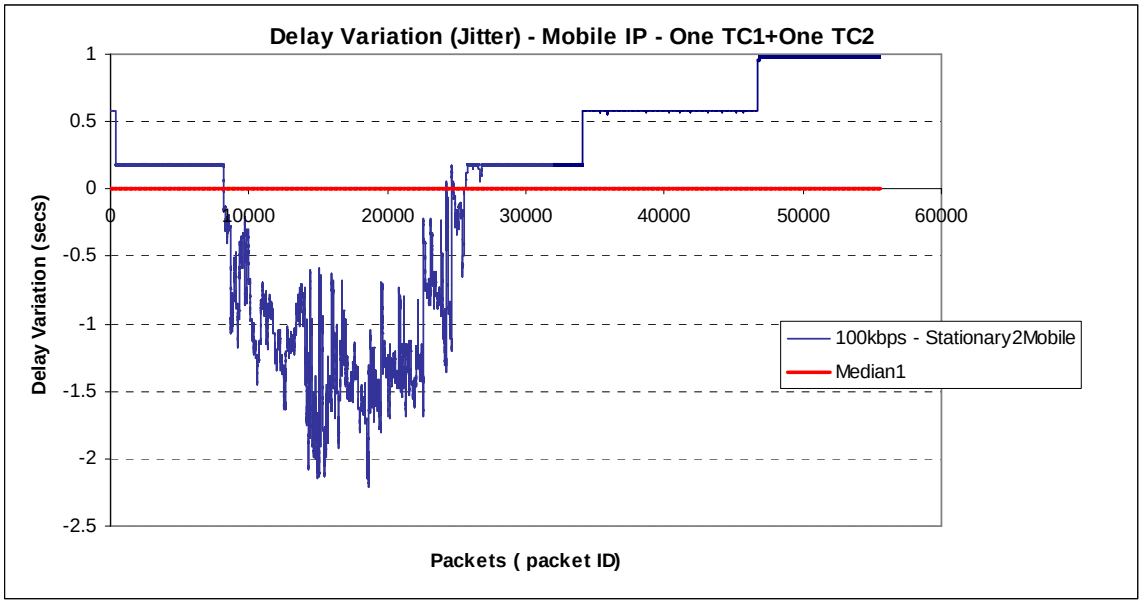
Από τις γραφικές παραστάσεις της διακύμανσης καθυστέρησης (βλέπε σχήματα 70 – 73) παρατηρούμε ότι στο SIP έχουμε μικρότερες τιμές σε σχέση με αυτές που έχουμε στο Mobile IP. Ένα σημαντικότερο συμπέρασμα είναι μεγαλύτερο jitter που παρατηρείται στο Mobile IP σε αυτά τα σενάρια σε σχέση με το σενάριο 3. Μια τέτοια διακύμανση στην τιμή της καθυστέρησης υποδηλώνει ότι το Mobile IP δε μπορεί να υποστηρίξει ποιοτικά εφαρμογές video streaming και φωνής ταυτόχρονα.



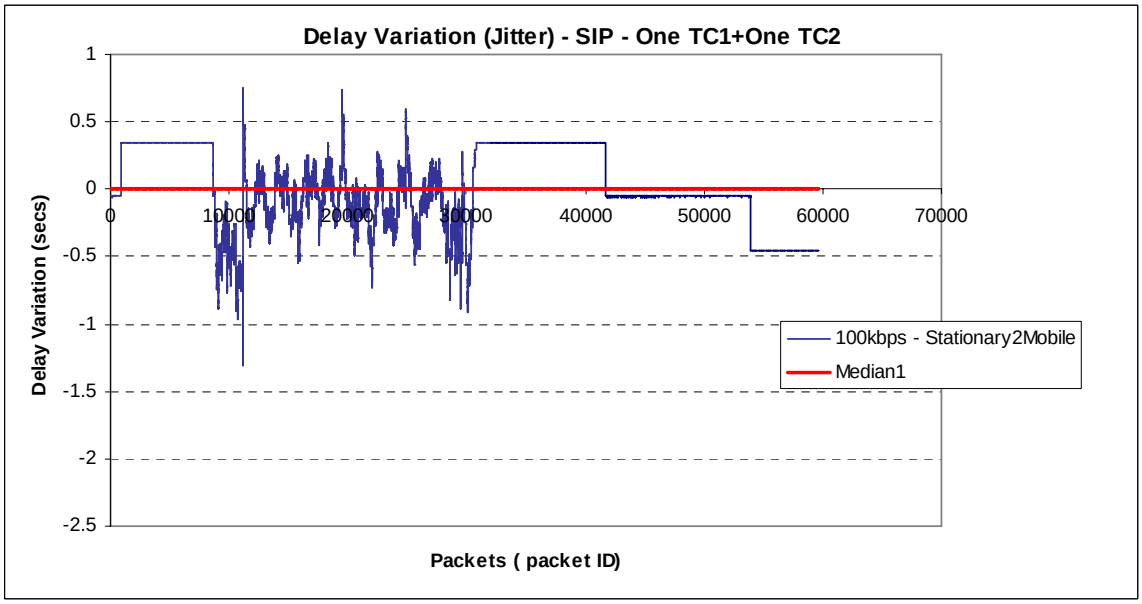
Σχήμα 70: Διακύμανση καθυστέρησης σε σενάρια TC1 και TC2 – Κίνηση TC1 και uplink (2Kbps) της TC2 – Mobile IP.



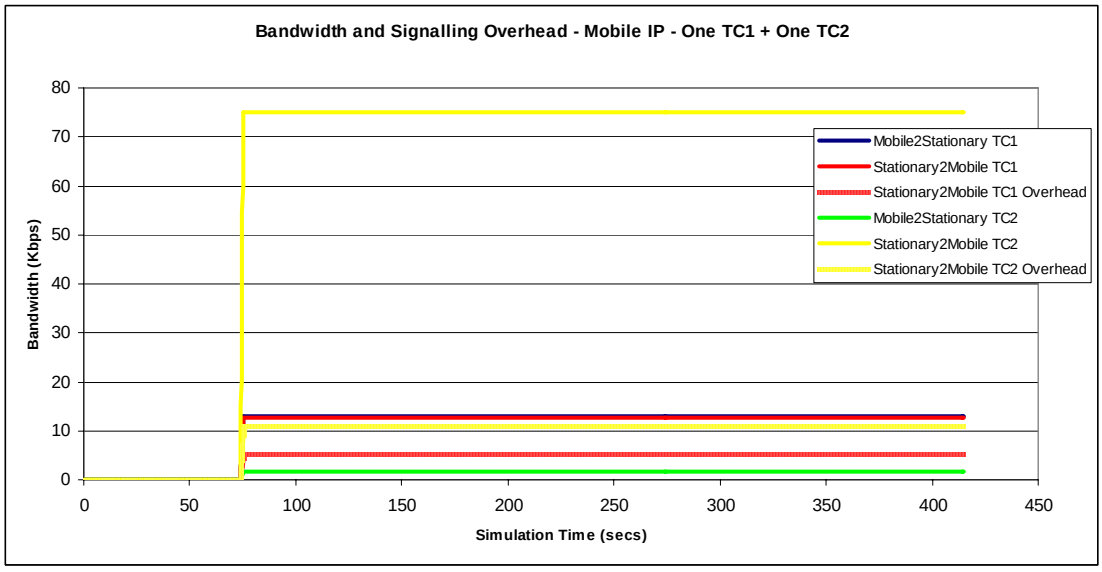
Σχήμα 71: Διακύμανση καθυστέρησης σε σενάρια TC1 και TC2 – Κίνηση TC1 και uplink (2Kbps) της TC2 – SIP.



Σχήμα 72: Διακύμανση καθυστέρησης σε σενάρια TC1 και TC2 – Downlink (100Kbps) της TC2 – Mobile IP.



Σχήμα 73: Διακύμανση καθυστέρησης σε σενάρια TC1 και TC2 – Downlink (100Kbps) της TC2 – SIP.



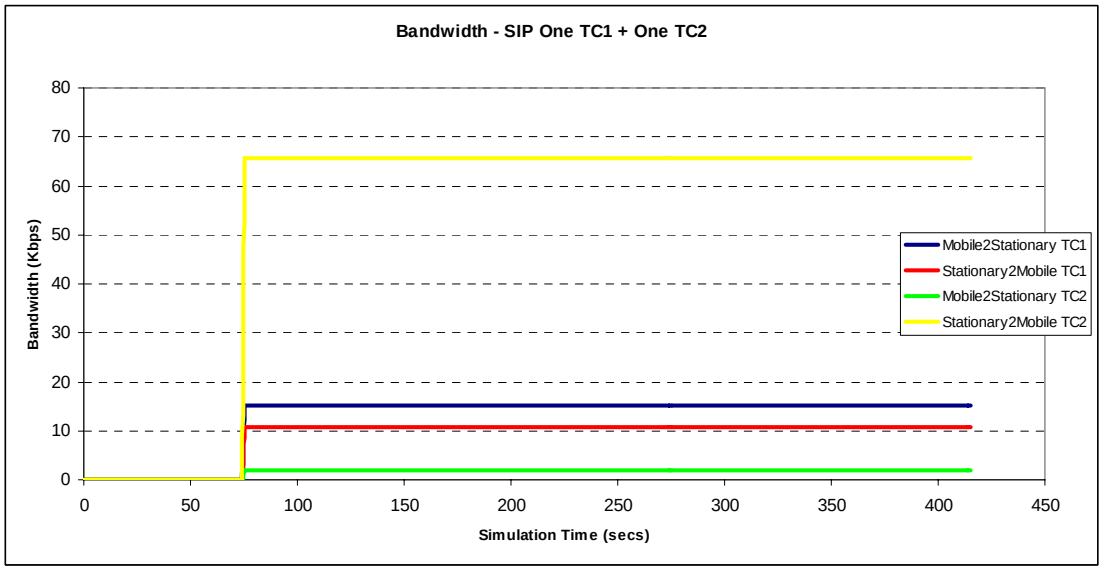
Σχήμα 74: Ρυθμός μετάδοσης και επιβάρυνση σηματοδοσίας σε σενάριο μιας σύνδεσης TC1 και μιας σύνδεσης TC2.

Όσον αφορά στη γραφική παράσταση του μέσου ρυθμού μετάδοσης καθαρής πληροφορίας για τη σύνδεση TC1 (σχήμα 74), παρατηρούμε ότι είναι μικρότερος (και στις δύο ζεύξεις) από τον αντίστοιχο που είδαμε στην περίπτωση μιας σύνδεσης TC1 (βλέπε § 5.1) αλλά συγκρινόμενος με την περίπτωση δύο ταυτόχρονων συνδέσεων TC1 (βλέπε §5.4) δε παρουσιάζονται μεγάλες διαφορές. Ειδικότερα μπορούμε να δούμε ότι ο μέσος ρυθμός μετάδοσης καθαρής πληροφορίας στην περίπτωση που μελετάμε τώρα, είναι μεγαλύτερος από ότι στην περίπτωση δύο TC1 όσον αφορά τη ζεύξη από τον κινητό κόμβο προς το σταθερό γιατί πάνω στην ίδια ζεύξη (κινητό προς το σταθερό κόμβο) υπάρχει και το uplink της σύνδεση TC2, της οποίας ο ρυθμός μετάδοσης πληροφορίας είναι αρκετά μικρός (~1.6Kbps) εν αντιθέσει με την περίπτωση 2 TC1 συνδέσεων όπου ο ρυθμός μετάδοσης και των δύο συνδέσεων πάνω στην ίδια ζεύξη είναι μεγαλύτερος (12 – 13Kbps). Παρατηρώντας τη ζεύξη από σταθερό προς κινητό κόμβο, βλέπουμε ότι στην περίπτωση μας ο ρυθμός μετάδοσης καθαρής πληροφορίας είναι περίπου 7.6Kbps δηλαδή αρκετά μικρότερος από την περίπτωση ταυτόχρονων συνδέσεων TC1 όπου ο αντίστοιχος ρυθμός είναι περίπου ο διπλάσιος (13.6Kbps). Αυτό οφείλεται στο γεγονός ότι στην περίπτωσή μας, υπάρχει σύνδεση 64Kbps καθαρής πληροφορίας (λόγω ροής video) πάνω στην ίδια ζεύξη κάτι που δε συμβαίνει όταν έχουμε ταυτόχρονες συνδέσεις TC1.

Παρατηρώντας τη γραφική παράσταση του μέσου ρυθμού μετάδοσης καθαρής πληροφορίας για τη σύνδεση TC2 (σχήμα 74), παρατηρούμε ότι είναι μικρότερος (και στις δύο ζεύξεις) από τον αντίστοιχο που είδαμε στη περίπτωση μιας σύνδεσης TC2 (βλέπε §5.2) αλλά συγκρινόμενος με τη περίπτωση δύο ταυτόχρονων συνδέσεων TC2 (βλέπε §5.5) είναι αρκετά μεγαλύτερος.

Όσον αφορά στη σύνδεση TC1, το ποσοστό απώλειας πακέτων είναι 19.62% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 34.64% για την αντίθετη ζεύξη. Οι τιμές αυτές είναι αισθητά μεγαλύτερες από αυτές που ίσχυαν στη περίπτωση 2 ταυτόχρονων συνδέσεων TC1 (βλέπε §5.4) ιδιαίτερα στη ζεύξη από σταθερό προς κινητό κόμβο και μπορούν να δικαιολογηθούν από το γεγονός ότι στην περίπτωση αυτή διακινείται μεγαλύτερη ποσότητα πληροφορίας στην προαναφερθείσα ζεύξη, οπότε είναι αυξημένη και η πιθανότητα απώλειας. Αν επιχειρήσουμε μια σύγκριση με την περίπτωση όπου είχαμε μόνο μια σύνδεση TC1 (βλέπε §5.1) θα διαπιστώσουμε ότι τα ποσοστά απώλειας πακέτων ήταν μικρότερα (κοντά στο 3% στη κάθε ζεύξη) από αυτά που υπολογίσαμε στην παράγραφο αυτή. Αυτό ήταν καθόλα αναμενόμενο μιας και στην περίπτωση που μελετούμε τώρα, ιδιαίτερα στη ζεύξη από σταθερό προς κινητό κόμβο διακινούνται περισσότερα πακέτα (αφού έχουμε και ροή video) από ότι στην περίπτωση μιας σύνδεσης TC1 οπότε αυξημένη θα είναι και η πιθανότητα απώλειας πακέτων.

Από την άλλη πλευρά το ποσοστό απώλειας πακέτων που ανήκουν στη σύνδεση TC2 είναι 20.28% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 35% για την αντίθετη ζεύξη λόγω του ότι διακινείται μεγαλύτερη ποσότητα πληροφορίας στη ζεύξη αυτή. Οι τιμές αυτές είναι αισθητά μικρότερες από αυτές που ίσχυαν στην περίπτωση 2 ταυτόχρονων συνδέσεων TC2 (βλέπε §5.5) αλλά μεγαλύτερες από ότι είχαμε στην περίπτωση μιας σύνδεσης TC2 (βλέπε §5.2).



Σχήμα 75: Ρυθμός μετάδοσης σε σενάριο μιας σύνδεσης TC1 και μιας σύνδεσης TC2.

Παρατηρώντας το πιο πάνω σχήμα (βλέπε σχήμα 75) για τη περίπτωση του SIP, μπορούμε να διαπιστώσουμε ότι ισχύουν τα ίδια συμπεράσματα όπως και στην περίπτωση του Mobile IP όσον αφορά τις συγκρίσεις με τα άλλα σχετικά σενάρια. Εκτός αυτού, μπορούμε να δούμε ότι ο μέσος ρυθμός μετάδοσης πληροφορίας στην περίπτωση της σύνδεσης των 100Kbps από τον σταθερό κόμβο στο κινητό κόμβο (~66Kbps) στο SIP είναι λίγο μεγαλύτερος από το μέσο ρυθμό μετάδοσης καθαρής πληροφορίας στο Mobile IP (75 – 11.2 = 63.8Kbps). Όπως επίσης και στη σύνδεση των 64Kbps από τον σταθερό κόμβο προς τον κινητό έχουμε μεγαλύτερο ρυθμό μετάδοσης καθαρής πληροφορίας στην περίπτωση του SIP (10.5Kbps) έναντι του Mobile IP (12-4=8Kbps).

Όσον αφορά στο μέσο χρόνο διαπομπής, στο SIP είναι περίπου 5.6 δευτερόλεπτα δηλαδή ελάχιστα μεγαλύτερος από το μέσο χρόνο διαπομπής που είχαμε στην περίπτωση μιας σύνδεσης TC2 ενώ στο Mobile IP είναι 6.3 δευτερόλεπτα.

Παρατηρώντας τη σύνδεση TC1, βλέπουμε ότι το ποσοστό απώλειας πακέτων είναι 5.05% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 31.67% για την αντίθετη ζεύξη. Οι τιμές αυτές είναι αισθητά μεγαλύτερες από αυτές που ίσχυαν στην περίπτωση 2 ταυτόχρονων συνδέσεων TC1 (βλέπε §5.4) ιδιαίτερα στη ζεύξη από

σταθερό προς κινητό κόμβο όπως προηγουμένως, Τα ίδια ισχύουν και για την περίπτωση όπου είχαμε μόνο μια σύνδεση TC1 (βλέπε §5.1).

Από την άλλη πλευρά το ποσοστό απώλειας πακέτων που ανήκουν στη σύνδεση TC2 είναι 5.11% για τη ζεύξη από τον κινητό κόμβο UA1 προς το σταθερό UA2 και 31.87% για την αντίθετη ζεύξη λόγω του ότι διακινείται μεγαλύτερη ποσότητα πληροφορίας στη ζεύξη αυτή. Οι τιμές αυτές είναι αισθητά μικρότερες από αυτές που ίσχυαν στην περίπτωση 2 ταυτόχρονων συνδέσεων TC2 (βλέπε §5.5) αλλά μεγαλύτερες από ότι είχαμε στην περίπτωση μιας σύνδεσης TC2 (βλέπε §5.2) κάτι που είχαμε παρατηρήσει και στο Mobile IP.

6. Συμπεράσματα

Φτάνοντας στο τέλος της διαδρομής θα θέλαμε να αποτυπώσουμε τα συμπεράσματα και τις γενικές παρατηρήσεις που κάναμε κατά τη διάρκεια της εκπόνησης αυτής της διπλωματικής εργασίας.

Αρχικά θα μπορούσαμε να πούμε ότι το Πρωτόκολλο Έναρξης Συνόδου αποτελεί έναν αξιόπιστο μηχανισμό υποστήριξης κινητικότητας στο επίπεδο εφαρμογής, ο οποίος συγκρινόμενος με το Mobile IP παρουσιάζει καλύτερες επιδόσεις όσον αφορά στην καθυστέρηση μετάδοσης πακέτων και τη διακύμανση της καθυστέρησης από άκρο σε άκρο. Αυτό άλλωστε αποδεικνύεται από τα αποτελέσματα εκτέλεσης των σεναρίων που παρουσιάσαμε στην προηγούμενη παράγραφο. Οι ιδιότητες αυτές του Πρωτοκόλλου Έναρξης Συνόδου το κάθιστουν κατάλληλο για εφαρμογές πραγματικού χρόνου όπως είναι το Voice over IP.

Από την άλλη, το Mobile IP προσφέρεται για υπηρεσίες που απαιτούν τη διατήρηση ενεργών συνδέσεων (πάνω από το αξιόπιστο πρωτόκολλο TCP) καθ' όλη τη διάρκεια της κίνησης ενός χρήστη διαμέσου διαφορετικών δικτύων όπως είναι η μεταφορά αρχείων (FTP). Η δυνατότητα αυτή του Mobile IP οφείλεται στο ότι η αλλαγή της διεύθυνσης IP είναι διαφανής στα επίπεδα που βρίσκονται πάνω από το επίπεδο 3. Τέτοιες συνδέσεις δύναται να υποστηριχθούν και από το SIP με την προϋπόθεση ότι θα υλοποιηθεί πάνω από το πολλά υποσχόμενο πρωτόκολλο SCTP, το οποίο υποστηρίζει την ύπαρξη συνδέσεων ακόμα κι αν οι χρήστες κινούνται και αλλάζουν διευθύνσεις IP.

Στο σημείο αυτό θα ήταν παράλειψη να μην αναφερθούμε στο Network Simulator, το οποίο σαν ένα αρθρωτό και επεκτάσιμο περιβάλλον προσομοίωσης δικτύων αποδείχθηκε η καλύτερη επιλογή για την ενσωμάτωση μιας νέας επέκτασης όπως αυτής του SIP. Παρόλα αυτά επειδή είναι λογισμικό ανοικτού κώδικα «νοσεί» αρκετά στο documentation.

Παράρτημα Α – Παρουσίαση Κώδικα

Ο πλήρης κώδικας όλων των αρχείων που δημιουργήθηκαν ή τροποποιήθηκαν κατά τη διάρκεια αυτής της διπλωματικής εργασίας παρατίθεται με εκτενή σχόλια ανά αρχείο, στο παράρτημα αυτό.

Αρχικά παρατίθενται τα αρχεία που δημιουργήθηκαν κατά τη διάρκεια αυτής της διπλωματικής εργασίας.

Αρχείο: sip.h
Ευρετήριο: ~/ns2.27/apps/

```
#ifndef ns_sip_h
#define ns_sip_h

#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"
#include "udp.h"
#include <string>
#include <assert.h>
#include "classifier-addr.h"
#include "methods.h"

#define MAX_EL            30        // Maximum elements of every array

//Types of messages
typedef enum {
INVITE,           //call initialization
REGISTER,     //USER AGENT -> PROXY/REGISTRAR  -- User Agent is registered in the new region
MESSAGE,
BYE,              //call termination
ACK,
TRYING_100,
RINGING_180,
OK_200,
MOVED_302,    //PROXY -> PROXY -- User Agent is far from its home region (Moved temporarily)
NOT_FOUND_404
} SipRegType;

struct hdr_sip {
  int invite_;        //callee's IP address
  int contact_;     //caller's IP address
  int from_;         //caller's SIP URI;
  int to_;            //callee's SIP URI
  SipRegType type_; //type of the message
  int index_;       //the index of the application
  bool expires_;   //true for removing binding (default value = false)
  SipRegType cseq_;

  //Header access methods
  static int offset_; //required by PacketHeaderManager
  inline static hdr_sip* access(const Packet *p) {
                return (hdr_sip*)p->access(offset_);
  }
};

//TIMER
```

```
class SimpleTimer : public TimerHandler {

 public:

  SimpleTimer(Agent *a) : TimerHandler() { a_ = a; }
 protected:
  inline void expire(Event *) { a_->timeout(0); }
  Agent *a_;
};

//USER AGENT
class SipUAgent : public Agent {
 public:
  SipUAgent();
  int command(int argc, const char*const* argv);
  void recv(Packet*, Handler*);
  void timeout(int);

 protected:
  //method that sends REGISTER message whenever a User Agent changes region
  void sendRegister(int newIp);
  //method that removes binding whenever a USER AGENT goes out of bounds of his visited PROXY
  void removeBinding();
  //method that checks if the node is inside the area that a Proxy Agent covers; it is executed periodically
  bool isInBounds(double x,double y);
  SimpleTimer timer_;
  string names[MAX_EL]; //this array holds the name of the node and the names of agents that are attached on this node (except for the sip agent)
  //Proxy Agent coordinates that the User Agent gets as soon as it receives the REG_OK
  double xProxy_;
  double yProxy_;
  MobileNode *node_; //ptr to my mobilenode
 private:
  int mySipUri_;       //Well known User Agent's SIP URI
  int dstSipUri_;      //destination's SIP URI
  int communication;
};

//PROXY AGENT - REGISTRAR AGENT
class SipPAgent : public Agent {
 public:
  SipPAgent();  //constructor
  ~SipPAgent(); //destructor
  int command(int argc, const char*const* argv);
  void recv(Packet*, Handler*);
 protected:
  MobileNode *node_;
  NsObject *ragent_; //pointer to the routing agent -- e.g DSDV
};
```

Αρχείο: sip.cc

Ευρετήριο: ~/ns2.27/apps/

```
#include <template.h>
#include <random.h>
#include <mobilenode.h>
#include <iostream>
#include <sip.h>
#include <address.h>
#include <fstream>
#include <sstream>
#include <string.h>
#include <cmath>
#include <random.h>

#define SIP_PORT 5060       // well-known SIP port
#define RANGE    250        // Base station range of coverage
#define CURR_TIME                   Scheduler::instance().clock()  // Current time

#define CRREGFILE(a,b)          AgentMethods::instance().crRegFile(a,b)
#define GETDOMAIN(a)            AgentMethods::instance().getDomain(a)
#define WRITEIP(a)              AgentMethods::instance().writeIP(a,b)
#define GETPROXY(a)             AgentMethods::instance().getProxy(a)
#define GETVISPROXYADDR(a)  AgentMethods::instance().getVisProxyAdd(a)
#define GETVISADDR(a)           AgentMethods::instance().getVisAdd(a)
#define FINDFREEIP(a)           AgentMethods::instance().findfreeIp(a)
#define GETFREEIP(a)            AgentMethods::instance().getfreeIp(a)
#define INSMOVEDAG(a,b)         AgentMethods::instance().insMovedAg(a,b)
```

```
#define FREEIP(a)                       AgentMethods::instance().freeIp(a)
#define FINDCOORD(a,b)                  AgentMethods::instance().findCoord(a,b)
#define CREATECOORDFILE(a,b,c)  AgentMethods::instance().crCoordFile(a,b,c)

#define ERASECOORDFILE(a)     AgentMethods::instance().eraseCoordFile(a)

#define ERASEREGFILE(a)         AgentMethods::instance().eraseRegFile(a)
#define ISPROXY(a)                      AgentMethods::instance().isProxy(a)

#define PRINTADDR(a)                Address::instance().print_nodeaddr(a)
#define STR2ADDR(a)                 Address::instance().str2addr(a)

using namespace std;

string appl[MAX_EL][2];
//the application that the user wants to start
int indexx=0; //related with index_ of hdr_sip

int hdr_sip::offset_;
static class SipHeaderClass : public PacketHeaderClass {
public:
  SipHeaderClass() : PacketHeaderClass("PacketHeader/Sip",sizeof(hdr_sip)) {
                bind_offset(&hdr_sip::offset_);
  }
} class_siphdr;


static class SipUAClass : public TclClass {
public:
  SipUAClass() : TclClass("Agent/SipUA") {}
  TclObject* create(int, const char*const*) {
    return (new SipUAgent());
  }
} class_sipua;


SipUAgent::SipUAgent() : Agent(PT_SIP),timer_(this),
 node_(0),mySipUri_(0),dstSipUri_(0),xProxy_(0.0),yProxy_(0.0),communication(0)
{
                bind("packetSize_", &size_);
}

int SipUAgent::command(int argc, const char*const* argv)
{
  if (argc == 2) {
    //this command is executed at the beginning of every User Agent
    if (strcmp(argv[1], "open") == 0) {

      Tcl& tcl = Tcl::instance();

      // Create a new packet
      Packet* pkt = allocpkt();
      // Access the IP header for the new packet:
      hdr_ip* hdrip = hdr_ip::access(pkt);
      // Access the SIP header for the new packet:
      hdr_sip* hdrsip = hdr_sip::access(pkt);

      // Initializing USER AGENT'S SIP URI and IP address
      mySipUri_  = Address::instance().get_nodeaddr(addr());
      dstSipUri_ = Address::instance().get_nodeaddr(daddr());

      if(node_) {
                if(STR2ADDR((GETPROXY(PRINTADDR(addr()))).c_str())==FINDCOORD(node_->X(),node_->Y())->ip ) {
                        if(FINDCOORD(node_->X(),node_->Y())->ip != node_->base_stn())
                                node_->set_base_stn(FINDCOORD(node_->X(),node_->Y())->ip);
                }
                else {  // User Agent's Proxy != User Agent's base station
                        // Packet to the VISITED PROXY requesting registration
                        if(FINDCOORD(node_->X(),node_->Y())->ip != node_->base_stn())
                                node_->set_base_stn(FINDCOORD(node_->X(),node_->Y())->ip);

              // Changing the nessesary addresses
              string newIp = FINDFREEIP(PRINTADDR(FINDCOORD(node_->X(),node_->Y())->ip));
                        addr() = STR2ADDR((newIp).c_str());
                        sprintf(tcl.buffer(), "$%s addr %s",(names[0]).c_str(),newIp.c_str());
              tcl.eval();
              sprintf(tcl.buffer(), "[$%s get_ragent] addr %s",(names[0]).c_str(),newIp.c_str());
              tcl.eval();
              sprintf(tcl.buffer(), "[$%s get_ragent] sip %d",(names[0]).c_str(),mySipUri_);
              tcl.eval();
              sprintf(tcl.buffer(),"$%s move-dmux %s %s",(names[0]).c_str(),newIp.c_str(),PRINTADDR(addr()));
              tcl.eval();

              //setting the destination of all transport agents that are connected on this node
              for(int i=1; i<MAX_EL; i++) {
                   if(names[i] != "") {
                        sprintf(tcl.buffer(), "$%s set agent_addr_ %d", (names[i]).c_str(), addr() );
                        tcl.eval();
                   }
              }
```

```
        }
    }

    // Initializing headers
    hdrip->saddr() = addr();

    hdrip->sport() = port();

    hdrip->daddr() = node_->base_stn();
    hdrip->dport() = SIP_PORT;
    hdrsip->type_  = REGISTER;
    hdrsip->from_  = mySipUri_;       //the sender of the request
    hdrsip->to_    = mySipUri_;       //contains the SIP URI of the USER AGENT that is being resistered
    hdrsip->contact_ = addr();

    send(pkt,0);

    xProxy_ = FINDCOORD(node_->X(),node_->Y())->x;
    yProxy_ = FINDCOORD(node_->X(),node_->Y())->y;

    timer_.resched(0.1);

    return(TCL_OK);
  }
}
else if (argc == 3) {
  if (strcmp(argv[1], "invite") == 0) {

    // Create a new packet
    Packet* pkt = allocpkt();
    // Access the IP header for the new packet:
    hdr_ip* hdrip = hdr_ip::access(pkt);
    // Access the SIP header for the new packet:
    hdr_sip* hdrsip = hdr_sip::access(pkt);

    // IP Header
    hdrip->saddr() = addr();
    hdrip->sport() = port();
    hdrip->daddr() = node_->base_stn(); //Visited Proxy Agent IP address
    hdrip->dport() = SIP_PORT;              //Visited Proxy Agent's well known SIP port

    // SIP Header
    hdrsip->contact_ = addr();       //Caller's IP Address
    hdrsip->from_  = mySipUri_;   //Caller's SIP URI | END to END USER AGENTS
    hdrsip->to_    = dstSipUri_;     //Callee's SIP URI | MUST BE CONNECTED
    hdrsip->invite_= dstSipUri_;    //Callee's IP Address
    hdrsip->type_  = INVITE;        //INVITE message
    hdrsip->index_ = indexx;        //Application index
    appl[indexx][0]  = argv[2];
    appl[indexx++][1]= "none";      //in case of ftp -- simplex connection

    //send the packet
    send(pkt,0);
    cout << "USER AGENT "  << PRINTADDR(addr()) << " sends an INVITE message at " << CURR_TIME << endl;

    // return TCL_OK, so the calling function knows that the
    // command has been processed
    return(TCL_OK);
  }
  if (strcmp(argv[1], "bye") == 0) { //it is an end-to-end method

    Tcl& tcl = Tcl::instance();
    // Create a new packet
    Packet* pkt = allocpkt();
    // Access the IP header for the new packet:
    hdr_ip* hdrip = hdr_ip::access(pkt);
    // Access the SIP header for the new packet:
    hdr_sip* hdrsip = hdr_sip::access(pkt);

    // IP Header
    hdrip->saddr() = addr();
    hdrip->sport() = port();
    hdrip->daddr() = daddr();             //Visited Proxy Agent IP address
    hdrip->dport() = SIP_PORT;          //Visited Proxy Agent's well known SIP port

    // SIP Header
    hdrsip->from_  = mySipUri_;
    hdrsip->to_    = dstSipUri_;
    hdrsip->type_  = BYE;               //BYE message

    // find the index of the application that will be terminated
    for(int i=0; i<MAX_EL; i++) {
            if((appl[i][0] == argv[2]) || (appl[i][1] == argv[2])) {
                    if(appl[i][0] == argv[2])
                            appl[i][0] = "none";
                    else
                            appl[i][1] = "none";
                    hdrsip->index_ = i;
                    break;
```

```
            }
    }

    communication--; //a connection will be terminated
    sprintf(tcl.buffer(), "$%s stop", argv[2] );
    tcl.eval();
    cout << "The session that this SipUAgent has terminated at " << CURR_TIME << " is: " << argv[2] << endl;



    //send the packet
    send(pkt,0);
    cout << "USER AGENT "  << PRINTADDR(addr()) << " sends a BYE message at " << CURR_TIME << endl;

    // return TCL_OK, so the calling function knows that the
    // command has been processed
    return(TCL_OK);
  }
  if(strcmp(argv[1], "message") == 0) {
       int length;
       length=strlen(argv[2]);

        // Create a new packet
        Packet* pkt = allocpkt(length);
        // Access the IP header for the new packet:
        unsigned char *data = pkt->accessdata();
        hdr_ip* hdrip = hdr_ip::access(pkt);
        // Access the SIP header for the new packet:
        hdr_sip* hdrsip = hdr_sip::access(pkt);
       // Access the common header for the new packet:
       hdr_cmn* hdrcmn = hdr_cmn::access(pkt);

      // IP Header
       hdrip->saddr() = addr();
       hdrip->sport() = SIP_PORT;
       hdrip->daddr() = STR2ADDR((GETPROXY(PRINTADDR(daddr()))).c_str());
                hdrip->dport() = SIP_PORT;
         // SIP Header
       hdrsip->contact_ = addr();     //Caller's IP Address
       hdrsip->from_  = mySipUri_;    //Caller's SIP URI | END to END USER AGENTS
       hdrsip->to_    = dstSipUri_;   //Callee's SIP URI | MUST BE CONNECTED
       hdrsip->invite_= daddr();
       hdrsip->type_  = MESSAGE;      //MESSAGE message
                memcpy (data, argv[2], length);
       hdrcmn->size() += length;
       cout << "USER AGENT "  << PRINTADDR(addr()) << " sends a MESSAGE at " <<CURR_TIME<<endl;
                send (pkt, 0);

                return(TCL_OK);

  }
  else if(strcmp(argv[1], "node") == 0) {
    //obtain reference to a MobileNode object so as to use functions of the class MobileNode -- eg set_base_stn()
    node_ = (MobileNode*)TclObject::lookup(argv[2]);
    if(node_ == 0) {
              fprintf(stderr,"%s: %s lookup of %s failed\n",__FILE__,argv[1],argv[2]);
              return(TCL_ERROR);
    }

    return(TCL_OK);
  }
}
else if ((argc == 4) && (strcmp(argv[2],"agents") != 0)) {
  if (strcmp(argv[1], "invite") == 0) {

    // Create a new packet
    Packet* pkt = allocpkt();
    // Access the IP header for the new packet:
    hdr_ip* hdrip = hdr_ip::access(pkt);
    // Access the SIP header for the new packet:
    hdr_sip* hdrsip = hdr_sip::access(pkt);

    // IP Header
    hdrip->saddr() = addr();
    hdrip->sport() = port();
    hdrip->daddr() = node_->base_stn(); //Visited Proxy Agent IP address
    hdrip->dport() = SIP_PORT;              //Visited Proxy Agent's well known SIP port

    // SIP Header
    hdrsip->contact_ = addr();       //Caller's IP Address
    hdrsip->from_  = mySipUri_;    //Caller's SIP URI | END to END USER AGENTS
    hdrsip->to_    = dstSipUri_;     //Callee's SIP URI | MUST BE CONNECTED
    hdrsip->invite_= dstSipUri_;     //Callee's IP Address
    hdrsip->type_  = INVITE;          //INVITE message
    hdrsip->index_ = indexx;         //Application index
    appl[indexx][0]  = argv[2];
    appl[indexx++][1]= argv[3];     //in case of cbr -> duplex connection

    //send the packet
    send(pkt,0);
```

```
      cout << "USER AGENT "  << PRINTADDR(addr()) << " sends an INVITE message at " << CURR_TIME << endl;

      // return TCL_OK, so the calling function knows that the
      // command has been processed
      return(TCL_OK);
    }
  }
  else { // command with many arguments
    names[0] = argv[1];           //the first argument => name of the node

    for(int i=3; i<argc; i++)     //the rest of the arguments (beyong the word agents) => names of the agents

                names[i-2] = argv[i];
    return(TCL_OK);
  }
  // If the command hasn't been processed by SipUAgent()::command,
  // call the command() function for the base class
  return (Agent::command(argc, argv));
}

void SipUAgent::recv(Packet* pkt, Handler*)
{
    //Access the instance -- reference to the instance
    Tcl& tcl = Tcl::instance();

    // Access the IP header for the received packet:
    hdr_ip* hdrip = hdr_ip::access(pkt);
    // Access the SIP header for the received packet:
    hdr_sip* hdrsip = hdr_sip::access(pkt);

    // Create a new packet
    Packet* pktret = allocpkt();
    // Access the IP header for the new packet:
    hdr_ip* hdripret = hdr_ip::access(pktret);
    // Access the SIP header for the new packet:
    hdr_sip* hdrsipret = hdr_sip::access(pktret);

    Packet* pktret2 = allocpkt();
    hdr_ip* hdripret2 = hdr_ip::access(pktret2);
    hdr_sip* hdrsipret2 = hdr_sip::access(pktret2);

    // Access the IP header for the new packet:
    unsigned char *data;

    switch(hdrsip->type_) {
                case INVITE:
                                if(ISPROXY(PRINTADDR(hdrip->saddr()))) { //if the sender is a proxy send 180 RINGING and 200 OK messages
                                cout << "USER AGENT "  << PRINTADDR(addr()) << " received an INVITE message/sends RINGING message at "
<<CURR_TIME<<endl;
                                        hdripret->daddr() = hdrip->saddr();
                                        hdripret->dport() = hdrip->sport();
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;
                                        hdrsipret->contact_ = addr();
                                        hdrsipret->invite_  = hdrsip->contact_;
                                        hdrsipret->type_  = RINGING_180;
                                        send(pktret,0);

                                        //sending OK message
                                        cout << "USER AGENT "  << PRINTADDR(addr()) << " sends OK message at " << CURR_TIME << endl;
                                        hdripret2->daddr() = hdrip->saddr();
                                        hdripret2->dport() = hdrip->sport();
                                        hdripret2->saddr() = addr();
                                        hdripret2->sport() = port();
                                        hdrsipret2->contact_ = addr();
                                        hdrsipret2->from_  = hdrsip->from_;
                                        hdrsipret2->to_    = hdrsip->to_;
                                        hdrsipret2->invite_= hdrsip->contact_;
                                        hdrsipret2->type_  = OK_200;
                                        hdrsipret2->cseq_  = INVITE;
                                        hdrsipret2->index_ = hdrsip->index_;
                                        send(pktret2,0);

                                        //if a mobile that changed region USER AGENT sends INVITE
                                        if(daddr() != hdrsip->contact_) {
                                                daddr() = hdrsip->contact_;

                                                //setting destination address of sip agent and all agents that are connected on this node
                                                for(int i=1; i<MAX_EL; i++) {
                                                        if(names[i] != "") {
                                                        sprintf(tcl.buffer(), "$%s set dst_addr_ %d", (names[i]).c_str(),hdrsip->contact_ );
                                                        tcl.eval();
                                                        }
                                                }

                                        }
                                }
                               else { //this message re-INVITE comes from a USER AGENT that changed region and this USER AGENT must send 200 OK
```

```
                        //directly to the correspondent USER AGENT
                       cout << "USER AGENT "  << PRINTADDR(addr()) << " has been informed and sends 200 OK at " << CURR_TIME  << endl;
                                hdripret->daddr() = hdrip->saddr();
                                hdripret->dport() = hdrip->sport();
                                hdripret->saddr() = addr();
                                hdripret->sport() = port();
                                hdrsipret->from_  = hdrsip->from_;
                                hdrsipret->to_    = hdrsip->to_;
                                hdrsipret->type_  = OK_200;
                                hdrsipret->cseq_  = INVITE;
                                send(pktret,0);


                                Packet::free(pktret2);
                        }
                        Packet::free(pkt);
                        break;
        case BYE:
                        cout << "USER AGENT " << PRINTADDR(addr()) << " received BYE message at " << CURR_TIME  << endl;
                        communication--; //connection will be terminated
                        //find out which application will be terminated
                        if((appl[hdrsip->index_][0] != "none") || (appl[hdrsip->index_][1] != "none")) {
                                if(appl[hdrsip->index_][0] != "none") {
                                        sprintf(tcl.buffer(), "$%s stop", (appl[hdrsip->index_][0]).c_str() );
                                        tcl.eval();
                        cout << "The session that this SipUAgent has terminated at " << CURR_TIME << " is: " << appl[hdrsip->index_][0] << endl;
                                        appl[hdrsip->index_][0] = "none";
                                }
                                else {
                                        sprintf(tcl.buffer(), "$%s stop", (appl[hdrsip->index_][1]).c_str() );
                                        tcl.eval();
                         cout << "The session that this SipUAgent has terminated at " << CURR_TIME << " is: " << appl[hdrsip->index_][1] << endl;
                                        appl[hdrsip->index_][1] = "none";
                                }
                        }
                        hdripret->daddr() = hdrip->saddr();
                        hdripret->dport() = hdrip->sport();
                        hdripret->saddr() = addr();
                        hdripret->sport() = port();
                        hdrsipret->from_  = hdrsip->from_;
                        hdrsipret->to_    = hdrsip->to_;
                        hdrsipret->type_  = OK_200;
                        hdrsipret->cseq_  = BYE;
                        send(pktret,0);

                        Packet::free(pktret2);
                        Packet::free(pkt);
                        break;
        case OK_200:
                        switch(hdrsip->cseq_) {
                                case REGISTER:
                                        cout << "USER AGENT " <<PRINTADDR(addr())<<" received 200 OK (REG) at "<<CURR_TIME<<endl;
                         //There are two posibilities:
                        //(a)receive 200 OK from HOME PROXY if this USER AGENT re-entered inside his region => do nothing at all
                        //(b)receive 200 OK from VISITED PROXY => must send REGISTER to his HOME PROXY to bind his new IP with his SIP URI
                        //but if we are in the middle of a call then firstly USER AGENT must send INVITE to the correspondent USER AGENT
                        //and sends REGISTER to his HOME PROXY for binding afterwards
                        if(hdrsip->contact_ != 1) {
                                if(communication != 0) { //Inform the Correspondent USER AGENT that this USER AGENT changed region
                                  cout <<"At " << CURR_TIME <<" USER AGENT'S NEW IP " << PRINTADDR(addr()) << endl;
                                  cout << "USER AGENT " <<PRINTADDR(addr())<<" informs the correspondent USER AGENT at "<<CURR_TIME<<endl;
                                   hdripret->daddr() = daddr();
                                   hdripret->dport() = SIP_PORT;
                                   hdripret->saddr() = addr();
                                  hdripret->sport() = port();
                                  hdrsipret->type_  = INVITE;
                                  hdrsipret->contact_ = addr();  //this is USER AGENT'S new IP address
                                  send(pktret,0);
                        }
                        else { //if there is no connection
                           cout <<"At " << CURR_TIME <<" USER AGENT'S NEW IP " << PRINTADDR(addr()) << endl;
                           if(hdrip->saddr() != STR2ADDR((GETPROXY(PRINTADDR(mySipUri_)).c_str())) { //(b)
                                   cout << "USER AGENT " << PRINTADDR(addr()) << " sends REGISTER to his HOME PROXY at "<<CURR_TIME<<endl;
                                    hdripret->daddr() = hdrip->saddr();
                                    hdripret->dport() = hdrip->sport();
                                    hdripret->saddr() = addr();
                                    hdripret->sport() = port();
                                    hdrsipret->from_  = mySipUri_;
                                    hdrsipret->to_    = STR2ADDR((GETPROXY(PRINTADDR(mySipUri_))).c_str()); //to HOME PROXY
                                    hdrsipret->type_  = REGISTER;
                                    hdrsipret->contact_ = addr();  //this is USER AGENT'S new IP that sends to his HOME PROXY
                                    send(pktret,0);
                                      }
                           }
                         }
                         break;
                         case INVITE:
                            cout << "USER AGENT " << PRINTADDR(addr()) << " received 200 OK and sends ACK at " << CURR_TIME << endl;
                            hdripret->daddr() = hdrip->saddr();
```

```
                        hdripret->dport() = hdrip->sport();
                        hdripret->saddr() = addr();
                        hdripret->sport() = port();
                        hdrsipret->from_  = hdrsip->from_;
                        hdrsipret->to_    = hdrsip->to_;
                        hdrsipret->contact_ = addr();
                        hdrsipret->invite_= hdrsip->contact_;
                        hdrsipret->index_ = hdrsip->index_;
                        hdrsipret->type_  = ACK;
                        hdrsipret->cseq_  = INVITE;
                        send(pktret,0);
                        if(ISPROXY(PRINTADDR(hdrip->saddr()))) { //if it is not an OK 200 after re-INVITE
                            communication++; //new connection will be established

                           if(daddr() != hdrsip->contact_ ) {    //if the callee moves in a visited region, caller must change its destination addr

                           //setting destination address of sip agent and all agents that are connected on this node
                                for(int i=1; i<MAX_EL; i++) {
                                        if(names[i] != "") {
                                            sprintf(tcl.buffer(), "$%s set dst_addr_ %d", (names[i]).c_str(),hdrsip->contact_ );
                                            tcl.eval();
                                        }
                                  }
                                  daddr() = hdrsip->contact_;
                                }
                             }
                           else { //sends REGISTER to HOME PROXY in order to bind USER's URI with its new ip address
                           if( STR2ADDR((GETPROXY(PRINTADDR(addr()))).c_str()) != STR2ADDR((GETPROXY(PRINTADDR(mySipUri_))).c_str())) {
                              cout << "USER AGENT " << PRINTADDR(addr()) << " sends REGISTER to his HOME PROXY at "<<CURR_TIME<<endl;
                              hdripret2->daddr() = node_->base_stn();
                              hdripret2->dport() = SIP_PORT;
                              hdripret2->saddr() = addr();
                              hdripret2->sport() = port();
                              hdrsipret2->from_  = mySipUri_;
                              hdrsipret2->to_    = STR2ADDR((GETPROXY(PRINTADDR(mySipUri_))).c_str()); //to HOME PROXY
                              hdrsipret2->type_  = REGISTER;
                              hdrsipret2->contact_ = addr();  //this is USER AGENT'S new IP that sends to his HOME PROXY
                             send(pktret2,0);
                           }
                        }
                        break;
                        case BYE:
                                        cout << "USER AGENT " << PRINTADDR(addr()) << " received 200 OK at " << CURR_TIME <<endl;
                                        Packet::free(pktret);
                                        Packet::free(pktret2);
                        break;
                        case MESSAGE:
                        cout<<"USER AGENT "<<PRINTADDR(addr())<<" received an OK_200 \"INSTANT MESSAGE\" MESSAGE at
"<<CURR_TIME<<endl;
                        Packet::free(pktret);
                        Packet::free(pktret2);
                        break;
                        default:
                                        Packet::free(pktret);
                                        Packet::free(pktret2);
                                        break;
                        }
                        Packet::free(pkt);
                        break;
                case MESSAGE:
                        data = pkt->accessdata();
                        data[pkt->datalen()] = '\0';
                        printf("USER AGENT %s received MESSAGE %s",PRINTADDR(addr()),data);
                        cout << " at " << CURR_TIME << endl;
                        hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->contact_))).c_str());
                        hdripret->dport() = SIP_PORT;
                        hdripret->saddr() = addr();
                        hdripret->sport() = SIP_PORT;
                        hdrsipret->from_  = hdrsip->from_;
                        hdrsipret->to_    = hdrsip->to_;
                        hdrsipret->invite_= hdrsip->contact_;
                        hdrsipret->contact_ = addr();
                        hdrsipret->type_  = OK_200;
                        hdrsipret->cseq_  = MESSAGE;
                        send(pktret,0);
                        Packet::free(pkt);
                        Packet::free(pktret2);
                        break;
                case TRYING_100:
                        cout << "USER AGENT "  << PRINTADDR(addr()) << " received 100 TRYING message at " << CURR_TIME << endl;
                        Packet::free(pkt);
                        Packet::free(pktret);
                        Packet::free(pktret2);
                        break;
                case RINGING_180:
                        cout << "USER AGENT "  << PRINTADDR(addr()) << " received 180 RINGING message at " << CURR_TIME << endl;
                        Packet::free(pkt);
                        Packet::free(pktret);
                        Packet::free(pktret2);
                        break;
```

```
		case ACK:
			cout << "USER AGENT "  << PRINTADDR(addr()) << " received ACK message at " << CURR_TIME << endl;
			if(ISPROXY(PRINTADDR(hdrip->saddr()))) {
				communication++; //new connection will be established
				if(appl[hdrsip->index_][1] == "none") {
					sprintf(tcl.buffer(), "$%s start", (appl[hdrsip->index_][0]).c_str() );
					tcl.eval();
			cout << "The session that this SipUAgent has established at " <<CURR_TIME<<" is: " << appl[hdrsip->index_][0] << endl;
				}
				else {
					sprintf(tcl.buffer(), "$%s start", (appl[hdrsip->index_][0]).c_str() );
					tcl.eval();
					sprintf(tcl.buffer(), "$%s start", (appl[hdrsip->index_][1]).c_str() );
					tcl.eval();
			cout << "The session that this SipUAgent has established at " <<CURR_TIME<<" is: " << appl[hdrsip->index_][0] << endl;

			cout << "The session that this SipUAgent has established at " <<CURR_TIME<<" is: " << appl[hdrsip->index_][1] << endl;

				}
			}
			else { //ACK message after re-INVITE
			//setting destination address of sip agent and all agents that are connected on this node
			for(int i=1; i<MAX_EL; i++) {
			    if(names[i] != "") {
				sprintf(tcl.buffer(), "$%s set dst_addr_ %d", (names[i]).c_str(),hdrsip->contact_ );
				tcl.eval();
			    }
			 }
			daddr() = hdrsip->contact_;
			}
			Packet::free(pkt);
			Packet::free(pktret);
			Packet::free(pktret2);
			break;
		case NOT_FOUND_404:
			cout << "USER AGENT "  << PRINTADDR(addr()) << " received 404 NOT FOUND message at " << CURR_TIME << endl;
			Packet::free(pkt);
			Packet::free(pktret);
			Packet::free(pktret2);
			break;
		default:
			Packet::free(pkt);
			Packet::free(pktret);
			Packet::free(pktret2);
			break;
	}

}

//Checks if User Agent is in or out of Proxy Agent's range
bool SipUAgent::isInBounds(double x,double y)
{
		//RXThresh_ default value is 3.652e-10 W. Using /indep-utils/propagation/threshold.cc we
		//can evaluate the default range of the base-station which is 250m -- for TwoRayGround model
		if((xProxy_ == 0.0) && (yProxy_ == 0.0)) return true; //initial state
		if( sqrt( pow(xProxy_-x,2) + pow(yProxy_-y,2) ) < RANGE )
			return true;  //User Agent is in bounds
		else
			return false; //User Agent is out of bounds
}

//Every given period of time this function is invoked
void SipUAgent::timeout(int)
{
		Tcl& tcl = Tcl::instance();
		string newIp, oldIp;

		if(!isInBounds(node_->X(),node_->Y())) {
		cout << "USER AGENT "<<PRINTADDR(addr())<<" enters in region "<< PRINTADDR(FINDCOORD(node_->X(),node_->Y())->ip)   << " at "
<< CURR_TIME <<endl;
			oldIp = PRINTADDR(addr()) ;

			//User agent's new ip address
			//check if the USER AGENT entered inside his home region
			if(STR2ADDR((GETPROXY(PRINTADDR(mySipUri_))).c_str()) != FINDCOORD(node_->X(),node_->Y())->ip)
				newIp = FINDFREEIP(PRINTADDR(FINDCOORD(node_->X(),node_->Y())->ip));
			else
				newIp = PRINTADDR(mySipUri_);

		    //If USER AGENT gets outside of a region which is not his home region
		    if(node_->base_stn() != STR2ADDR((GETPROXY(PRINTADDR(mySipUri_))).c_str()))
		    //Send REGISTER message with expires_ = true to his old visited PROXY AGENT so as to remove binding
				removeBinding();

		//USER AGENT changes base station
		node_->set_base_stn(FINDCOORD(node_->X(),node_->Y())->ip);
		// Get the coordinates of the closest PROXY AGENT - Base station
		xProxy_ = FINDCOORD(node_->X(),node_->Y())->x;
		yProxy_ = FINDCOORD(node_->X(),node_->Y())->y;
		// Changing the nessesary addresses
```

```
                // 1.Changing the addr_ of this sip agent
                addr() = STR2ADDR(newIp.c_str());
                // 2.Changing the address_ of this node
                sprintf(tcl.buffer(), "$%s addr %s",(names[0]).c_str(),newIp.c_str());
                tcl.eval();
                // 3.Changing myaddr_ of DSDV routing agent
                sprintf(tcl.buffer(), "[$%s get_ragent] addr %s",(names[0]).c_str(),newIp.c_str());
                 tcl.eval();
                // 4.Sending update to the new base station
                sprintf(tcl.buffer(), "[$%s get_ragent] sip %d",(names[0]).c_str(),mySipUri_);
                 tcl.eval();
                // Moving the dmux of this node from the old IP address to the new IP address
                 sprintf(tcl.buffer(), "$%s move-dmux %s %s",(names[0]).c_str(), newIp.c_str(), oldIp.c_str());
                 tcl.eval();

                //setting the destination of all transport agents that are connected on this node
                for(int i=1; i<MAX_EL; i++) {

                   if(names[i] != "") {

                        sprintf(tcl.buffer(), "$%s set agent_addr_ %d", (names[i]).c_str(), addr() );
                        tcl.eval();
                   }
               }

                //USER AGENT wants to register with his new PROXY AGENT
                            sendRegister(STR2ADDR(newIp.c_str()));
                }
                timer_.resched(0.1);
}

//Function that sends a REGISTER message whenever the USER AGENT changes region
void SipUAgent::sendRegister(int newIp)
{
                Packet  *pkt    = allocpkt();
                hdr_ip  *hdrip  = hdr_ip::access(pkt);
                hdr_sip *hdrsip = hdr_sip::access(pkt);

                //Find the IP address of the Proxy Agent which is near from this USER AGENT
                hdrip->saddr() = addr();
                hdrip->sport() = port();
                hdrip->daddr() = FINDCOORD(node_->X(),node_->Y())->ip; //function that returns the IP address of the closest PROXY AGENT
                hdrip->dport() = SIP_PORT; //well-known sip port that the PROXY AGENT listens for requests

                hdrsip->contact_ = newIp;       //Register with the new IP address
                hdrsip->from_ = mySipUri_;
                hdrsip->to_   = mySipUri_;
                hdrsip->type_ = REGISTER;

                send(pkt,0);
}

void SipUAgent::removeBinding()
{
        Packet  *pkt    = allocpkt();
        hdr_ip  *hdrip  = hdr_ip::access(pkt);
        hdr_sip *hdrsip = hdr_sip::access(pkt);
        //USER AGENT sends a REGISTER message to his visited PROXY AGENT requesting removing of the previous binding
        hdrip->saddr() = addr();
        hdrip->sport() = port();
        hdrip->daddr() = node_->base_stn(); //FINDCOORD(node_->X(),node_->Y())->ip;
        hdrip->dport() = SIP_PORT;

         hdrsip->from_    = mySipUri_;
         hdrsip->contact_ = addr();
         hdrsip->expires_ = true;
         hdrsip->type_    = REGISTER;

         FREEIP(PRINTADDR(addr()));

//       send(pkt,0);
}

//PROXY AGENT - REGISTRAR AGENT
static class SipPAClass : public TclClass {
public:
  SipPAClass() : TclClass("Agent/SipPA") {}
  TclObject* create(int, const char*const*) {
    return (new SipPAgent());
  }
} class_sippa;


SipPAgent::SipPAgent() : Agent(PT_SIP),node_(0),ragent_(0)
{
  bind("packetSize_", &size_);
}

SipPAgent::~SipPAgent()
{               // Each Proxy erases its RegFl_ and its coordinates and ip address from coordFl file
```

```
                ERASEREGFILE(atoi((GETDOMAIN(PRINTADDR(addr()))).c_str()));
                ERASECOORDFILE(atoi((GETDOMAIN(PRINTADDR(addr()))).c_str()));
}

int SipPAgent::command(int argc, const char*const* argv)
{
      // Initialization of coordFl file -- each Proxy appends its coordinates and ip address
       if (argc == 2) {
                  if (strcmp(argv[1], "open") == 0) {
                    CREATECOORDFILE(node_->X(),node_->Y(),atoi((GETDOMAIN(PRINTADDR(addr()))).c_str()));
                    return TCL_OK;
                  }
       }

       if (argc == 3) {
                    // Initialization of the Registration File with 9 available ip addresses
                    if (strcmp(argv[1], "initialize") == 0) {
                                CRREGFILE( atoi((GETDOMAIN(argv[2])).c_str()) ,9);
                                return TCL_OK;


                    }

                    if(strcmp(argv[1], "node") == 0) {
                                // Obtain reference to a MobileNode object so as to use functions of the class MobileNode
                                node_ = (MobileNode*)TclObject::lookup(argv[2]);
                                if(node_ == 0) {
                                            fprintf(stderr,"%s: %s lookup of %s failed\n",__FILE__,argv[1],argv[2]);
                                            return(TCL_ERROR);
                                }
                                return(TCL_OK);
                    }
                     // Bind routing agent and sip agent
                     if(strcmp(argv[1], "ragent") == 0) {
                                ragent_ = (NsObject*)TclObject::lookup(argv[2]);
                                return(TCL_OK);
                    }
        }

                // If the command hasn't been processed by SipUAgent()::command,
                // call the command() function for the base class
                return (Agent::command(argc, argv));
}

void SipPAgent::recv(Packet* pkt, Handler*)
{
                Tcl& tcl = Tcl::instance();

                // Access the IP header for the received packet:
                hdr_ip* hdrip = hdr_ip::access(pkt);
                // Access the SIP header for the received packet:
                hdr_sip* hdrsip = hdr_sip::access(pkt);
                // Access the Common header for the received packet:
                hdr_cmn* hdrcmn = hdr_cmn::access(pkt);

                // Create a new packet
                 Packet* pktret = allocpkt();
                // Access the IP header for the new packet:
                 hdr_ip* hdripret = hdr_ip::access(pktret);
                // Access the SIP header for the new packet:
                hdr_sip* hdrsipret = hdr_sip::access(pktret);

       Packet* pktret2 = allocpkt();
       hdr_ip* hdripret2 = hdr_ip::access(pktret2);
       hdr_sip* hdrsipret2 = hdr_sip::access(pktret2);

       Packet* pktret3;
       hdr_ip* hdripret3;
       hdr_sip* hdrsipret3;
       hdr_cmn* hdrcmnret3;

                // Access the IP header for the new packet:
                unsigned char *data,*data2;
                int length;

                switch(hdrsip->type_) {
                            case REGISTER:
/*                                      if(hdrsip->expires_)   { //Removing binding
                                                if(STR2ADDR((GETPROXY(PRINTADDR(hdrsip->from_))).c_str()) == addr()) {
                                                            FREEIP(PRINTADDR(hdrsip->contact_));
                                                            cout << "PROXY " << PRINTADDR(addr())  << " is removing binding" << endl;
                                                            Packet::free(pktret);
                                                            Packet::free(pkt);
                                                            break;
                                                }
                                                else { // REGISTER packet was sent to the visited PROXY thus send REGISTER with expires_ to
                                                       //HOME PROXY
                                                            hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->from_))).c_str());
                                                            hdripret->dport() = SIP_PORT;
                                                            hdripret->saddr() = addr();
                                                            hdripret->sport() = port();
```

```
                                        hdrsipret->from_     = hdrsip->from_;
                                        hdrsipret->contact_  = hdrsip->contact_;
                                        hdrsipret->expires_  = hdrsip->expires_;
                                        hdrsipret->type_  = REGISTER;
                                        send(pktret,0);
                                        Packet::free(pkt);

                                }
                        } */
                        cout << "PROXY " << PRINTADDR(addr())  << " received a REGISTER message at " << CURR_TIME << endl;
                        //preparing the new packet to be sent
                        hdripret->saddr() = addr();
                        hdripret->sport() = port();
                        if(hdrsip->from_ == hdrsip->to_) {
                                //PROXY AGENT updates the registration file - database
                                GETFREEIP(PRINTADDR(hdrsip->contact_));
                                hdripret->daddr() = hdrip->saddr();
                                hdripret->dport() = SIP_PORT;
                                hdrsipret->type_  = OK_200;
                                hdrsipret->cseq_  = REGISTER;
                                send(pktret,0);

                        }

                        else {
                                if(hdrsip->to_ == addr()) {
                                        //PROXY AGENT binds the new ip of the USER AGENT with his SIP URI –
                                        //registration on behalf of visited Proxy
                                        INSMOVEDAG(PRINTADDR(hdrsip->from_),PRINTADDR(hdrsip->contact_));
                                        hdripret->daddr() = hdrip->saddr(); //send REG_OK to visited PROXY
                                        hdripret->dport() = hdrip->sport();
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;
                                        hdrsipret->type_  = OK_200;
                                        hdrsipret->cseq_  = REGISTER;
                                }
                                else {
                                        //send the REGISTER message that has just arrived from new USER AGENT to
                                        //USER AGENT's HOME PROXY
                                        hdripret->daddr()   = hdrsip->to_;
                                        hdripret->dport()   = SIP_PORT;
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_    = hdrsip->from_;
                                        hdrsipret->to_      = hdrsip->to_;
                                        hdrsipret->type_    = REGISTER;

                                }
                                send(pktret,0);

                        }
                        Packet::free(pkt);
                        Packet::free(pktret2);
                        break;
                case INVITE:
                        cout << "PROXY " << PRINTADDR(addr())  << " received an INVITE message at " << CURR_TIME << endl;
                        //Proxy must check if the callee belongs to its region and if it is inside
                        if (STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str()) == addr()) { //Callee belongs to this Proxy
                                //check if callee has his device closed - not registered to the network
                                if(GETVISPROXYADDR(PRINTADDR(hdrsip->invite_)) == "not_found") {
                                        hdripret->daddr() = hdrip->saddr();
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;
                                        hdrsipret->invite_= hdrsip->invite_;
                                        hdrsipret->type_  = NOT_FOUND_404;
                                        send(pktret,0);
                                }
                                else if(STR2ADDR((GETVISPROXYADDR(PRINTADDR(hdrsip->invite_))).c_str())  == addr()) {
                                        //Callee is inside Proxy's region
                                        //preparing to send the new packet to the User Agent that is inside Proxy's region
        cout << "PROXY " << PRINTADDR(addr()) << " sends INVITE to " << PRINTADDR(hdrsip->invite_) << " at " << CURR_TIME << endl;
                                        hdripret->daddr() = hdrsip->invite_;
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;  //caller's SIP URI
                                        hdrsipret->to_    = hdrsip->to_;    //callee's SIP URI
                                        hdrsipret->invite_= hdrsip->invite_;
                                        hdrsipret->type_  = INVITE;
                                        hdrsipret->index_ = hdrsip->index_;
                                        send(pktret,0);
                                        //Send Trying_100 to caller or to caller's Proxy
        cout << "PROXY " << PRINTADDR(addr()) << " sends 100 TRYING to " << PRINTADDR(hdrip->saddr())<< " at " << CURR_TIME <<endl;
                                        hdripret2->daddr() = hdrip->saddr();
                                        hdripret2->dport() = SIP_PORT;
```

```
                                                hdripret2->saddr() = addr();
                                                hdripret2->sport() = port();
                                                hdrsipret2->type_  = TRYING_100;
                                                send(pktret2,0);

                                        }
                                        else  { //Callee is inside other Proxy's region
                                                //The sender of the INVITE is a UA or a PA?
                                                //(a) PA then send the new packet MOVED to the caller's Proxy Agent
                                                //(b) UA relay the INVITE message to callee's Proxy Agent
                                                if( ISPROXY(PRINTADDR(hdrip->saddr())) ) { //(a)
                                                        hdripret->daddr() = hdrip->saddr();
                                                        hdripret->dport() = hdrip->sport();
                                                        hdripret->saddr() = addr();
                                                        hdripret->sport() = port();
                                        hdrsipret->contact_ = STR2ADDR((GETVISADDR(PRINTADDR(hdrsip->invite_))).c_str());
                                                        hdrsipret->from_  = hdrsip->from_;
                                                        hdrsipret->to_    = hdrsip->to_;
                                                        hdrsipret->invite_= hdrsip->contact_;
                                                        hdrsipret->type_  = MOVED_302;
                                                        hdrsipret->index_ = hdrsip->index_;
                                                        send(pktret,0);
                                                }

                                                else { //(b)

                                                        hdripret->daddr() =
                                                        STR2ADDR((GETVISPROXYADDR(PRINTADDR(hdrsip->invite_))).c_str());
                                                        hdripret->dport() = SIP_PORT;
                                                        hdripret->saddr() = addr();
                                                        hdripret->sport() = port();
                                                        hdrsipret->contact_ = hdrsip->contact_;
                                                        hdrsipret->from_  = hdrsip->from_;
                                                        hdrsipret->to_    = hdrsip->to_;
                                                hdrsipret->invite_= STR2ADDR((GETVISADDR(PRINTADDR(hdrsip->invite_))).c_str());
                                                        hdrsipret->type_  = INVITE;
                                                        hdrsipret->index_ = hdrsip->index_;
                                                        send(pktret,0);
                                                        //Send Trying_100 to caller
cout << "PROXY " << PRINTADDR(addr()) << " sends 100 TRYING to " << PRINTADDR(hdrip->saddr())<< " at " <<CURR_TIME<<endl;
                                                        hdripret2->daddr() = hdrip->saddr();
                                                        hdripret2->dport() = SIP_PORT;
                                                        hdripret2->saddr() = addr();
                                                        hdripret2->sport() = port();
                                                        hdrsipret2->type_  = TRYING_100;
                                                        send(pktret2,0);

                                                }
                                        }
                                }
                                else {  //The callee does not belong to Proxy's region
                                        //Send the new packet to the Proxy Agent of the callee
cout<<"PROXY "<<PRINTADDR(addr())<<" sends INVITE to "<<GETPROXY(PRINTADDR(hdrsip->invite_))<< " at " << CURR_TIME<<endl;
                                        hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str());
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;
                                        hdrsipret->invite_= hdrsip->invite_;
                                        hdrsipret->type_  = INVITE;
                                        hdrsipret->index_ = hdrsip->index_;
                                        send(pktret,0);
                                        //send TRYING to caller
cout << "PROXY " << PRINTADDR(addr()) << " sends 100 TRYING to " << PRINTADDR(hdrip->saddr())<< " at " << CURR_TIME<< endl;
                                        hdripret2->daddr() = hdrip->saddr();
                                        hdripret2->dport() = SIP_PORT;
                                        hdripret2->saddr() = addr();
                                        hdripret2->sport() = port();
                                        hdrsipret2->type_  = TRYING_100;
                                        send(pktret2,0);
                                }
                                Packet::free(pkt);
                                break;
                        case RINGING_180:
                                cout << "PROXY " << PRINTADDR(addr())  << " received 180 RINGING message at " << CURR_TIME << endl;
                                if(STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str())  == addr()) {
                                        hdripret->daddr() = hdrsip->invite_;
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;
                                        hdrsipret->invite_= hdrsip->invite_;
                                        hdrsipret->type_  = RINGING_180;
                                        hdrsipret->index_ = hdrsip->index_;
                                        send(pktret,0);
                                }
```

```
                                else {
                                        hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str());
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;
                                        hdrsipret->invite_= hdrsip->invite_;
                                        hdrsipret->type_  = RINGING_180;
                                        hdrsipret->index_ = hdrsip->index_;
                                        send(pktret,0);
                                }
                                Packet::free(pkt);
                                break;
                        case MOVED_302:
                                if(STR2ADDR((GETPROXY(PRINTADDR(hdrsip->contact_))).c_str()) != addr()) {
        cout << "PROXY " << PRINTADDR(addr())  << " received a 302 MOVED TEMPORARILY message at " << CURR_TIME << endl;
                                        hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->contact_))).c_str());
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->contact_ = hdrsip->invite_;
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;

                                        hdrsipret->invite_= hdrsip->contact_;

                                        hdrsipret->type_  = INVITE;
                                        hdrsipret->index_ = hdrsip->index_;
                                        send(pktret,0);
                                        //send ACK to HOME PROXY
                                        hdripret2->daddr() = hdrip->saddr();
                                        hdripret2->dport() = SIP_PORT;
                                        hdripret2->saddr() = addr();
                                        hdripret2->sport() = port();
                                        hdrsipret2->contact_ = hdrsip->invite_;
                                        hdrsipret2->type_  = ACK;
                                        hdrsipret2->cseq_  = MOVED_302;
                                        send(pktret2,0);
                                }
                                else { //the callee is inside this PROXY's region
                                        hdripret->daddr() = hdrsip->to_;
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->contact_ = hdrsip->invite_;
                                        hdrsipret->from_  = hdrsip->from_;  //caller's SIP URI
                                        hdrsipret->to_    = hdrsip->to_;    //callee's SIP URI
                                        hdrsipret->invite_= hdrsip->contact_;
                                        hdrsipret->type_  = INVITE;
                                        hdrsipret->index_ = hdrsip->index_;
                                         send(pktret,0); //send the packet through the routing agent
                                }
                                Packet::free(pkt);
                                break;//preparing the send the new packet MOVED to the caller's Proxy Agent
                        case OK_200:
                                switch(hdrsip->cseq_) {
                                        case REGISTER:
                                cout << "PROXY " << PRINTADDR(addr())  << " received a 200 OK (REG) message at " << CURR_TIME << endl;
                                                hdripret->daddr() = hdrsip->contact_;
                                                hdripret->dport() = SIP_PORT;
                                                hdripret->saddr() = addr();
                                                hdripret->sport() = port();
                                                hdrsipret->type_  = OK_200;
                                                hdrsipret->cseq_  = REGISTER;
                                                hdrsipret->contact_ = 1;//flag in order for the USER AGENT to understand that the
                                                //received a OK from his HOME PROXY
                                                send(pktret,0);
                                                Packet::free(pkt);
                                                break;
                                        case INVITE:
                        cout << "PROXY " << PRINTADDR(addr())  << " received a 200 OK (INVITE) message at " << CURR_TIME << endl;
                                                //Proxy must check if the caller belongs to its region and if it is inside
                                                if (STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str()) == addr()) {
                                                        //Caller belongs to this Proxy
                                                        hdripret->daddr() = hdrsip->invite_;
                                                        hdripret->dport() = SIP_PORT;
                                                        hdripret->saddr() = addr();
                                                        hdripret->sport() = port();
                                                        hdrsipret->contact_ = hdrsip->contact_;
                                                        hdrsipret->from_  = hdrsip->from_;
                                                        hdrsipret->to_    = hdrsip->to_;
                                                        hdrsipret->invite_= hdrsip->invite_;
                                                        hdrsipret->type_  = OK_200;
                                                        hdrsipret->cseq_  = INVITE;
                                                        hdrsipret->index_ = hdrsip->index_;
                                                        send(pktret,0);
                                                }
                                                else {  //The caller does not belong to Proxy's region
```

```
                                        //Send the new packet to the Proxy Agent of the caller
                                        hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str());
                                                hdripret->dport() = SIP_PORT;
                                                hdripret->saddr() = addr();
                                                hdripret->sport() = port();
                                                hdrsipret->contact_ = hdrsip->contact_;
                                                hdrsipret->from_  = hdrsip->from_;
                                                hdrsipret->to_    = hdrsip->to_;
                                                hdrsipret->invite_= hdrsip->invite_;
                                                hdrsipret->type_  = OK_200;
                                                hdrsipret->cseq_  = INVITE;
                                                hdrsipret->index_ = hdrsip->index_;
                                                send(pktret,0);
                                        }
                                        Packet::free(pkt);
                                        break;
                                case MESSAGE:
                                        // forward the message OK_200 to the instant message sender
                        cout << "PROXY " << PRINTADDR(addr())  << " received 200 OK message at " << CURR_TIME << endl;
                                        hdripret->daddr() = hdrsip->invite_;
                                        hdripret->dport() = SIP_PORT;
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = SIP_PORT;
                                        hdrsipret->contact_ = hdrsip->contact_;
                                        hdrsipret->from_  = hdrsip->from_;
                                        hdrsipret->to_    = hdrsip->to_;
                                        hdrsipret->type_  = hdrsip->type_;

                                        hdrsipret->cseq_  = hdrsip->cseq_;

                                        send(pktret,0);
                                        Packet::free(pkt);
                                        break;
                                default:
                                        Packet::free(pktret);
                                        Packet::free(pkt);
                                        break;
                        }
                        break;
        case ACK:
                        cout << "PROXY " << PRINTADDR(addr())  << " received an ACK message at " << CURR_TIME << endl;
                        switch(hdrsip->cseq_) {
                                case INVITE:
                                        if(STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str()) == addr()) {
                                                hdripret->daddr() = hdrsip->invite_;
                                                hdripret->dport() = SIP_PORT;
                                                hdripret->saddr() = addr();
                                                hdripret->sport() = port();
                                                hdrsipret->contact_ = hdrsip->contact_;
                                                hdrsipret->from_  = hdrsip->from_;
                                                hdrsipret->to_    = hdrsip->to_;
                                                hdrsipret->invite_= hdrsip->invite_;
                                                hdrsipret->index_ = hdrsip->index_;
                                                hdrsipret->type_  = ACK;
                                                send(pktret,0);
                                        }
                                        else {
                                        hdripret->daddr() = STR2ADDR((GETPROXY(PRINTADDR(hdrsip->invite_))).c_str());
                                                hdripret->dport() = SIP_PORT;
                                                hdripret->saddr() = addr();
                                                hdripret->sport() = port();
                                                hdrsipret->contact_ = hdrsip->contact_;
                                                hdrsipret->from_  = hdrsip->from_;
                                                hdrsipret->to_    = hdrsip->to_;
                                                hdrsipret->invite_= hdrsip->invite_;
                                                hdrsipret->index_ = hdrsip->index_;
                                                hdrsipret->type_  = ACK;
                                                hdrsipret->cseq_  = INVITE;
                                                send(pktret,0);
                                        }
                                        break;
                                case MOVED_302:
                                        Packet::free(pktret);
                                        break;
                                default:
                                        Packet::free(pktret);
                                        break;
                        }
                        Packet::free(pkt);
                        break;
        case TRYING_100:
                        cout << "PROXY "  << PRINTADDR(addr()) << " received 100 TRYING message at " << CURR_TIME << endl;
                        Packet::free(pkt);
                        Packet::free(pktret);
                        Packet::free(pktret2);
                        break;
        case NOT_FOUND_404:
        cout << "PROXY "  << PRINTADDR(addr()) << " received 404 NOT FOUND message at " << CURR_TIME << endl;
                        hdripret->daddr() = hdrsip->from_;
                        hdripret->dport() = SIP_PORT;
```

```
                                        hdripret->saddr() = addr();
                                        hdripret->sport() = port();
                                        hdrsipret->type_  = NOT_FOUND_404;
                                        send(pktret,0);
                                        Packet::free(pkt);
                                        Packet::free(pktret2);
                                        break;
                        case MESSAGE:
                                        data=pkt->accessdata();
                                        length=pkt->datalen();
                                        pktret3 = allocpkt(length);
                                        hdripret3 = hdr_ip::access(pktret3);
                                        hdrsipret3 = hdr_sip::access(pktret3);
                                        hdrcmnret3 = hdr_cmn::access(pktret3);
                                        data2=pktret3->accessdata();
                                        // IP Header
                                        hdripret3->saddr() = addr();
                                        hdripret3->sport() = SIP_PORT;
                                        hdripret3->daddr() = hdrsip->invite_;
                                        hdripret3->dport() = SIP_PORT;
                                        // SIP Header
                                        hdrsipret3->from_  = hdrsip->from_;       //Caller's SIP URI | END to END USER AGENTS
                                        hdrsipret3->to_    = hdrsip->to_;       //Callee's SIP URI | MUST BE CONNECTED
                                        hdrsipret3->type_  = MESSAGE;           //MESSAGE message
                                        memcpy (data2, (char*)data, length);
                                        hdrcmnret3->size() = hdrcmn->size();
                        cout <<"PROXY " << PRINTADDR(addr()) <<" has received the message.(forwarding...) at " << CURR_TIME << endl;
                                        send(pktret3,0);
                                        Packet::free(pkt);

                                        Packet::free(pktret);

                                        Packet::free(pktret2);
                                        break;
                        default:
                                        Packet::free(pkt);
                                        Packet::free(pktret);
                                        Packet::free(pktret2);
                                        break;
                }
}
```

Αρχείο: methods.cc

Ευρετήριο: ~/ns2.27/apps/

```
#include "methods.h"
#include "address.h"

AgentMethods* AgentMethods::instance_;

//Empty Constructor
AgentMethods::AgentMethods() {
}

/*************************************************************
      C O M M O N L Y   U S E D   F U N C T I O N S
      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*************************************************************/

/*************************************************************
It creates the registar files with addrNo free ips.
At first it has the ips(k.0.m) that are given to noone(x)
*************************************************************/
void AgentMethods::crRegFile(int domain,int addrNo)
{
 ostringstream hlpStream;
 string flNam,domString,fileLnString,iString;
// tha file name is RegFil_<domain>
 hlpStream << domain;
 domString=hlpStream.str();
 flNam="RegFl_"+domString;

// it opens the file empty for reading and writing
 ofstream fl(flNam.c_str(),ios_base::trunc|ios_base::in|ios_base::out);
  for (int i=1;i<addrNo+1;i++)
                {
                // empry the stream first
```

```
                hlpStream.str("");
                hlpStream << i;
                iString=hlpStream.str();
               // make the string that will be put in the file
                fileLnString=domString+".0."+iString+";x\n";
               // writes in the file
                fl.write(fileLnString.c_str(),fileLnString.length());
               }
// close the stream,thus the file
 fl.close();
}

/*************************************************************************
It finds the proxy server of the assigned ip given the home ip address
in some registar file.If it finds x, it returns "closed"
*************************************************************************/
string AgentMethods::getVisProxyAdd(string homeIp)
{
 char buff[32];
 string lnString;
// open the relevant file of the home ip's registar
 string flNam="RegFl_"+getDomain(homeIp);
 ifstream regFl(flNam.c_str(),ios_base::in);
// it searches all the lines
 while (!regFl.getline(buff, 32).eof())
 {
  lnString.assign(buff);
// if it finds the ip
  if ( lnString.find(homeIp, 0 ) != string::npos )
  {
                // it keeps the assigned ip in the visited registar

                lnString=lnString.substr(lnString.find(';',0)+1,lnString.length()-lnString.find(';',0)-1);

                break;
  }
}
 regFl.close();
//returns error if it didn't find a valid ip
 if (lnString.find('.', 0 ) == string::npos)
 return "not_found";
//else it returns the domain proxy's ip
 else
 return getProxy(lnString);
}

/*************************************************************************
It finds the foreign assigned ip given the home ip address
in some registar file.If it finds x, it returns "closed"
*************************************************************************/
string AgentMethods::getVisAdd(string homeIp)
{
 char buff[32];
 string lnString;
// open the relevant file of the home ip's registar
 string flNam="RegFl_"+getDomain(homeIp);
 ifstream regFl(flNam.c_str(),ios_base::in);
// it searches all the lines
 while (!regFl.getline(buff, 32).eof())
 {
  lnString.assign(buff);
// if it finds the ip
  if ( lnString.find(homeIp, 0 ) != string::npos )
  {
// it keeps the assigned ip in the visited registar
       lnString=lnString.substr(lnString.find(';',0)+1,lnString.length()-lnString.find(';',0)-1);
    break;
  }
}
 regFl.close();
//returns error if it didn't find a valid ip
 if (lnString.find('.', 0 ) == string::npos)
 return "not_found";
//else it returns the domain proxy's ip
 else
 return lnString;
}


/*************************************************************
It gets the proxy's ip given the ip address,ex 2.0.9->2.0.0
*************************************************************/
string AgentMethods::getProxy(string ip)
{
                return getDomain(ip)+".0.0";
}

/*************************************************************************
It finds the first free ip of the registar of a domain given the relevant
proxy's ip
```

```
*************************************************************************/
string AgentMethods::findfreeIp(string proxyIp)
{
 char buff[37];
 string lnString;
// open the relevant file of the home ip's registar
 string flNam="RegFl_"+getDomain(proxyIp);
 ifstream regFl(flNam.c_str(),ios_base::in|ios_base::out);
// it searches all the lines
 while (!regFl.getline(buff, 37).eof())
 {
  lnString.assign(buff);
// if it finds the first free ip,thus the ip with 'x' beside the ';'
  if ( lnString.find('x', 0 ) != string::npos )
  {
                // it keeps the free ip
                lnString=lnString.substr(0,lnString.find(';',0));
                break;
  }
}
 regFl.close();
 return lnString;
}


/*************************************************************
It gets the free ip given updating the relevant registar file
*************************************************************/
void AgentMethods::getfreeIp(string newIp)
{
 int line;
 int lncount=1;
 char buff[32];

 string lnString;

// open the relevant file of the home ip's registar
 string flNam="RegFl_"+getDomain(newIp);
 ifstream regFl(flNam.c_str(),ios_base::in);
// it searches all the lines
 while (!regFl.getline(buff, 32).eof())
 {
  lnString.assign(buff);
// if it finds the first free ip,thus the ip with 'x' beside the ';'
  if ( lnString.find(newIp, 0 ) != string::npos )
                line=lncount;
                lncount++;
 }
 regFl.close();
 writeIP(flNam,line,newIp);
}

/*********************************************************************
It gets the domain number of an ip address. It's onle defined to be used
in the getVisProxyAdd&findfreeIp functions
*********************************************************************/
string AgentMethods::getDomain(string Ip)
{
 string dom;
// keep the substring until the first '.'
 dom=Ip.substr(0,Ip.find_first_of('.',0));
 return dom;
}

/*************************************************************
It informs the home registar file about the new ip address
assigned from the visited registar
*************************************************************/
void AgentMethods::insMovedAg(string homeIp,string nIp)
{
  int line;
  int lncount=1;
  char buff[32];
  string lnString;
 // open the relevant file of the home ip's registar
  string flNam="RegFl_"+getDomain(homeIp);
  ifstream regFl(flNam.c_str(),ios_base::in);
 // it searches all the lines
  while (!regFl.getline(buff, 32).eof())
  {
   lnString.assign(buff);
 // if it finds the first free ip,thus the ip with 'x' beside the ';'
   if ( lnString.find(homeIp, 0 ) != string::npos )
                                line=lncount;
                                lncount++;
  }
 regFl.close();
 writeIP(flNam,line,nIp);
}
```

```
/*******************************************
It frees one of the given ips in a reg file
*******************************************/
void AgentMethods::freeIp(string hIp)
{
 insMovedAg(hIp,"x");
}

/*****************************************************************
It writes in a specified file, to the line told the ip given in
the position of assigned ip.It is only defined to be used by
getfreeIP&insMovedAg
*****************************************************************/
void AgentMethods::writeIP(string file,int lin,string ip)
{
  char c;
  int line=1;
  string leadString,leftString;
// first get the string that will be kept,following the written string
  ifstream iregFl(file.c_str(),ios_base::in);
// keep the leading string
while (line<lin) {
                  iregFl.get(c);
                  leadString+=c;
                  if (c=='\n') line++;
 }
 while (c!=';') {
                  iregFl.get(c);
                  leadString+=c;
}
// ignore until you get to the next line
while (c!='\n')
                  iregFl.get(c);
   leftString+='\n';
// keep the left string

   while (iregFl.get(c))

    leftString+=c;
  iregFl.close();

// write to the positions what you want
  fstream oregFl(file.c_str(),ios_base::out|ios_base::out);
//write thea leading string
  oregFl.write(leadString.c_str(),leadString.length());
// write the ip
  oregFl.write(ip.c_str(),ip.length());
// write the string you have kept
  oregFl.write(leftString.c_str(),leftString.length());
  oregFl.close();
}

/*************************************************************
 Given the coordinates of the mobile node it looks in
the coordFl file and retrns the ip of the closest proxy
*************************************************************/
coordInfo* AgentMethods::findCoord(double x,double y){
// the output
 coordInfo *out;
 coordInfo coinfo;
 out=&coinfo;

 char buff[100];
 string lnString,x0Str,y0Str;
 string flNam="coordFl";
 // the minimum distance is huge in the beginning
 double minDist=1000000;
 double x0,y0;
// open the file for reading
 ifstream coordFl(flNam.c_str(),ios_base::in);
 while (!coordFl.eof())
 {
 // read the nect line of the file
  coordFl.getline(buff,100);
  lnString.assign(buff);
  // get the coordinates
  x0Str=lnString.substr(0,lnString.find(',',0));
  x0=atof(x0Str.c_str());
  y0Str=lnString.substr(lnString.find(',',0)+1,lnString.find(';',0)-lnString.find(',',0)-1);
  y0=atof(y0Str.c_str());
  // if the new proxy is closer
  if (minDist>(x-x0)*(x-x0)+(y-y0)*(y-y0))
  {
                  // udate the closest proxy's info and the minimum distance
                  minDist=(x-x0)*(x-x0)+(y-y0)*(y-y0);
                  out->x=x0;
                  out->y=y0;
        out->ip=Address::instance().str2addr((lnString.substr(lnString.find(';',0)+1,lnString.length()-lnString.find(';',0)-1)).c_str());
  }
 }
```

```
// close the stream,thus the file
 coordFl.close();
 return out;
};

/*********************************************
Function that decides if the given ip address
belongs to a Proxy Agent
*********************************************/
bool AgentMethods::isProxy(string ipstr)
{
        int last = ipstr.size();
        int loc  = ipstr.find_last_of(".");

        if(ipstr.substr(loc+1,last-(loc+1)) == "0")
                return true;
        else false;
}
/************************************************************
 Given the coordinates of the proxy an its ip it adds this
information in the coordFl file
************************************************************/
void AgentMethods::crCoordFile(double x,double y,int domain)
{
 ostringstream hlpStream;
 string domString,xString,yString, fileLnString;
 string flNam="coordFl";
 // the file name is RegFil_<domain>
 hlpStream << domain;
 domString=hlpStream.str()+".0.0";
 // empty the stream
 hlpStream.str("");
 hlpStream << x;
 xString=hlpStream.str();
  // empty the stream
 hlpStream.str("");
 hlpStream << y;

 yString=hlpStream.str();

 // the line to be inserted in the file
 fileLnString=xString+','+yString+';'+domString+'\n';
 // open the Coord File for appending
 ofstream fl(flNam.c_str(),ios_base::app|ios_base::out);
 // writes in the file
 fl.write(fileLnString.c_str(),fileLnString.length());
 // close the stream,thus the file
 fl.close();
}

/************************************************************
 Given the coordinates of the proxy it erases its record from
 the Coord File
************************************************************/
 void AgentMethods::eraseCoordFile(int domain)
{
 char buff[100];
 ostringstream hlpStream;
 string domString,lnString,x0Str,y0Str;
 string outStr="";
 string flNam="coordFl";
 // the file name is RegFil_<domain>
 hlpStream << domain;
 domString=hlpStream.str()+".0.0";
 // the minimum distance is huge in the beginning
// open the file for reading
 ifstream coordFl(flNam.c_str(),ios_base::in);
 while (!coordFl.getline(buff,100).eof())
 {
  lnString.assign(buff);
  // get the coordinates
 // x0Str=lnString.substr(0,lnString.find(',',0));
 // y0Str=lnString.substr(lnString.find(',',0)+1,lnString.find(';',0)-lnString.find(',',0)-1);
  if (!(lnString.substr(lnString.find(';',0)+1,lnString.length()-lnString.find(';',0)-1)==domString))
    outStr+=lnString+'\n';
 }
  coordFl.close();
  // open the file for output to the positions what you want
  ofstream oregFl(flNam.c_str(),ios_base::out);
  //write the string
  oregFl.write(outStr.c_str(),outStr.length());
  oregFl.close();
}

/************************************************
 Erase the Reg File of the proxy given its domain
************************************************/
void AgentMethods::eraseRegFile(int domain)
{
 ostringstream hlpStream;
```

```
 string flNam,domString;
 // the file name is RegFil_<domain>
 hlpStream << domain;
 domString=hlpStream.str();
 flNam="RegFl_"+domString;
 remove(flNam.c_str());
}
```

Αρχείο: methods.cc

Ευρετήριο: ~/ns2.27/apps/

```
#include <template.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string.h>
#include <cmath>

/* the struct that will include the information for the closest proxy */
struct coordInfo {
                                double x;
                                double y;
                                 int ip;
};
class AgentMethods {
 public:
                static AgentMethods& instance() { return (*instance_); }
                AgentMethods();
                /*******************************
                COMMONLY USED FUNCTIONS
              -- DEFINITIONS --
                 *******************************/

                void crRegFile(int,int);

                string getDomain(string);
                 void writeIP(string,int,string);
                string getProxy(string);
                string getVisProxyAdd(string);
                string getVisAdd(string);
                string findfreeIp(string);
                void getfreeIp(string);
                void insMovedAg(string,string);
                void freeIp(string);
                coordInfo* findCoord(double,double); // returns the coordinates of the closest proxy
                bool isProxy(string);
                void crCoordFile(double,double,int);
                void eraseCoordFile(int);
                void eraseRegFile(int);
                //*******************************
 protected:
                static AgentMethods* instance_;
};
```

Αρχείο: ns-sip.tcl

Ευρετήριο: ~/ns2.27/tcl/lib/

```
#Sip Proxy Agent initialization
Agent/SipPA instproc init { node args } {
          eval $self next $args

          $self node $node                      ;# pointer to MobileNode
          $self initialize [$node node-addr]   ;# file initialization
}

#Sip User Agent initialization
Agent/SipUA instproc init { node args } {
          eval $self next $args
          $self node $node  ;# pointer to MobileNode
}
```

Τα παρακάτω αρχεία, προϋπήρχαν στο NSv2.27, αλλά τροποποιήθηκαν έτσι ώστε να υποστηρίζουν τη λειτουργικότητα και την ορθή εκτέλεση της εφαρμογής μας. Δε παρατίθεται ο πλήρης κώδικας των αρχείων αυτών παρά μόνο οι γραμμές που προστέθηκαν σε αυτά για λόγους απλότητας.

Αρχείο: dsdv.cc

Ευρετήριο: ~/ns2.27/dsdv/

```
/*************************************
 Function that changes variables myaddr_
here_.addr_ and makes a periodic update
*************************************/
void DSDV_Agent::sipUpdate(int dst)
{
   int update_type;

 rtable_ent *prte;
 rtable_ent  rte;

 prte = table_->GetEntry(dst);
 rte = *prte;
 rte.dst = myaddr_;
 rte.hop = myaddr_;

 table_->AddEntry(rte);


 addr() = myaddr_;

 // make a periodic update
 update_type = 1;
 Packet *p = makeUpdate(update_type);
 target_->recv(p, (Handler *)0);
}
/*************************************/

int
DSDV_Agent::command (int argc, const char *const *argv)
{
    ...

          else if (argc == 3)   {
               //*********************************************************
               if (strcasecmp (argv[1], "sip")  == 0) {
                              sipUpdate(atoi(argv[2]));
                              return TCL_OK;
               }
               //*********************************************************

    ...
```

Αρχείο: dsdv.h

Ευρετήριο: ~/ns2.27/dsdv/

```
...
  //*************************************
  // fuction declaration
   void sipUpdate(int dst);
  //*************************************
...
```

Αρχείο: arp.cc

Ευρετήριο: ~/ns2.27/mac/

```
int
ARPTable::arpresolve(nsaddr_t dst, Packet *p, LL *ll)
{
  …
          if(llinfo->count_ >= ARP_MAX_REQUEST_COUNT)
           {
          …
           //for SIP protocol
                hdr_cmn* hdr = HDR_CMN(t);
                if(hdr->ptype_ == PT_SIP) {
//                              printf("TRYING TO DROP PT_SIP at %.5f\n",Scheduler::instance().clock());
                              return 0;
                }
           …
          }

           /*
            * If we have SIP signalling and traffic running on a single node we don't want to lose the
            * first SIP packet because of arp request failure, so we keep the first SIP packet
          * and drop the rest of the packets
            */
          hdr_cmn* hdrcmn;
          llinfo->count_++;
          if(llinfo->hold_) { //drop pending packet if it is not SIP packet
                hdrcmn = HDR_CMN(llinfo->hold_);
                if(hdrcmn->ptype_ == PT_SIP) {/* do nothing at all -- hold the first SIP packet */}
                else {
                              drop(llinfo->hold_, DROP_IFQ_ARP_FULL);
//                            printf("Dropping packet from node %d at %.5f\n",node_->address(),Scheduler::instance().clock());
                }
          }

          hdr_cmn* hdrcmn2 = HDR_CMN(p);

          if(hdrcmn2->ptype_ == PT_SIP)
                llinfo->hold_ = p;
          else {
                if((hdrcmn) && (hdrcmn->ptype_ == PT_SIP)) {
                              drop(p, DROP_IFQ_ARP_FULL);
                }
                else {
                              llinfo->hold_ = p;
                }
          }
         …
```

Αρχείο: priqueue.cc

Ευρετήριο: ~/ns2.27/queue/

```
void
PriQueue::recv(Packet *p, Handler *h)
{
      struct hdr_cmn *ch = HDR_CMN(p);

      if(Prefer_Routing_Protocols) {

            switch(ch->ptype()) {
              …
              case PT_SIP:  // high priority for signalling packets (sip)
                    recvHighPriority(p, h);
                    break;
              …
```

Αρχείο: packet.h

Ευρετήριο: ~/ns2.27/common/

```
enum packet_t {
        ...

        //SIP packet
        PT_SIP,

        PT_NTYPE // This MUST be the LAST one
};
class p_info {
public:
        p_info() {
                ...
                // SIP packet
                name_[PT_SIP]= "sip";
                ...
```

Αρχείο: ns-mobilenode.tcl

Ευρετήριο: ~/ns2.27/tcl/lib/

```
...

#******************************************************************************
#                           SIP CODE
#******************************************************************************
Node/MobileNode instproc makemip-NewSipPA {} {
        $self instvar regagent_ agents_ id_


        set dmux [new Classifier/Port/Reserve]
        $dmux set mask_ 0x7fffffff
        $dmux set shift_ 0
        $self install-demux $dmux

        set regagent_ [new Agent/SipPA $self]
        $self attach $regagent_ [Node/MobileNode set SIP_PORT]  ;# attach sip proxy agent in port 5060
}

Node/MobileNode instproc makemip-NewSipUA {} {
        $self instvar regagent_

        set dmux [new Classifier/Port/Reserve]
        $dmux set mask_ 0x7fffffff
        $dmux set shift_ 0
        $self install-demux $dmux

        set regagent_ [new Agent/SipUA $self]
        $self attach $regagent_ [Node/MobileNode set SIP_PORT]  ;# attach sip user agent in port 5060
        $regagent_ set mask_ [AddrParams NodeMask 1]
        $regagent_ set shift_ [AddrParams NodeShift 1]
#       $regagent_ set dst_addr_ [expr (~0) << [AddrParams NodeShift 1]]
#       $regagent_ set dst_port_ 0
        $regagent_ node $self
}
#Bind routing agent and sip agent if existing basestation address setting
Node/MobileNode instproc sip-call {ragent} {
        $self instvar regagent_
        if [info exists regagent_] {
            $regagent_ ragent $ragent   ;# command ragent in SipPAs
        }
}
Node/MobileNode instproc sipag {} {
        return [$self set regagent_]        ;# returns an instance to the sip agent
}
Node/MobileNode instproc set_ragent {ragent} {
        $self instvar my_ragent_

        $self set my_ragent_ $ragent
}
Node/MobileNode instproc get_ragent {} {
        return [$self set my_ragent_]
}
#Add a new route from the new address to the dmux_ and delete route from the old address to the dmux_
```

```
Node/MobileNode instproc move-dmux {newaddr oldaddr} {
        $self instvar dmux_
        $self delete-route $oldaddr $dmux_
        $self add-route $newaddr $dmux_
}
#**************************************************************************
#                   end of SIP CODE
#**************************************************************************
...
```

Αρχείο: ns-default.tcl

Ευρετήριο: ~/ns2.27/tcl/lib/

```
Node/MobileNode set SIP_PORT                    5060
#Sip packet size
Agent/SipUA set packetSize_ 48
Agent/SipPA set packetSize_ 48
```

Αρχείο: ns-packet.tcl

Ευρετήριο: ~/ns2.27/tcl/lib/

```
foreach prot {
        ...
        #for SIP application
        #--------------------

        Sip

        #--------------------
        ...
```

Αρχείο: ns-lib.tcl

Ευρετήριο: ~/ns2.27/tcl/lib/

```
...
source ns-sip.tcl
...
#*********************************************************************************
Simulator instproc SIP {val} {$self set SIP_ $val}
#*********************************************************************************
...
Simulator instproc get-nodetype {} {
        ...
        # Sip agents
        if { [Simulator set sip_] } {
            if { $val == "Base" && $wiredRouting_ == "ON" } {
                    set val SipPA
            }
            if { $val == "Base" && $wiredRouting_ == "OFF" } {
                    set val SipUA
            }
```

```
        }
        …

Simulator instproc node-config args {
        …
        $self instvar addressType_ … SIP_ …
        …
        # set SIP flag
        if { [info exists SIP_] && $SIP_ == "ON"} {
            Simulator set sip_  1

        } else {
            if { [info exists SIP_] } {
                        Simulator set sip_ 0
            }
        }
        …

Simulator instproc create-wireless-node args {
        …
        if { [info exist wiredRouting_] && $wiredRouting_ == "ON" } {
            if { $routingAgent_ != "DSR" } {
                        $node mip-call $ragent
                        $node sip-call $ragent
            }
        }
        …

Simulator instproc create-dsdv-agent { node } {
        …
        # SIP protocol
        if [Simulator set sip_] {
            $ragent port-dmux [$node demux]
        }
        …
```

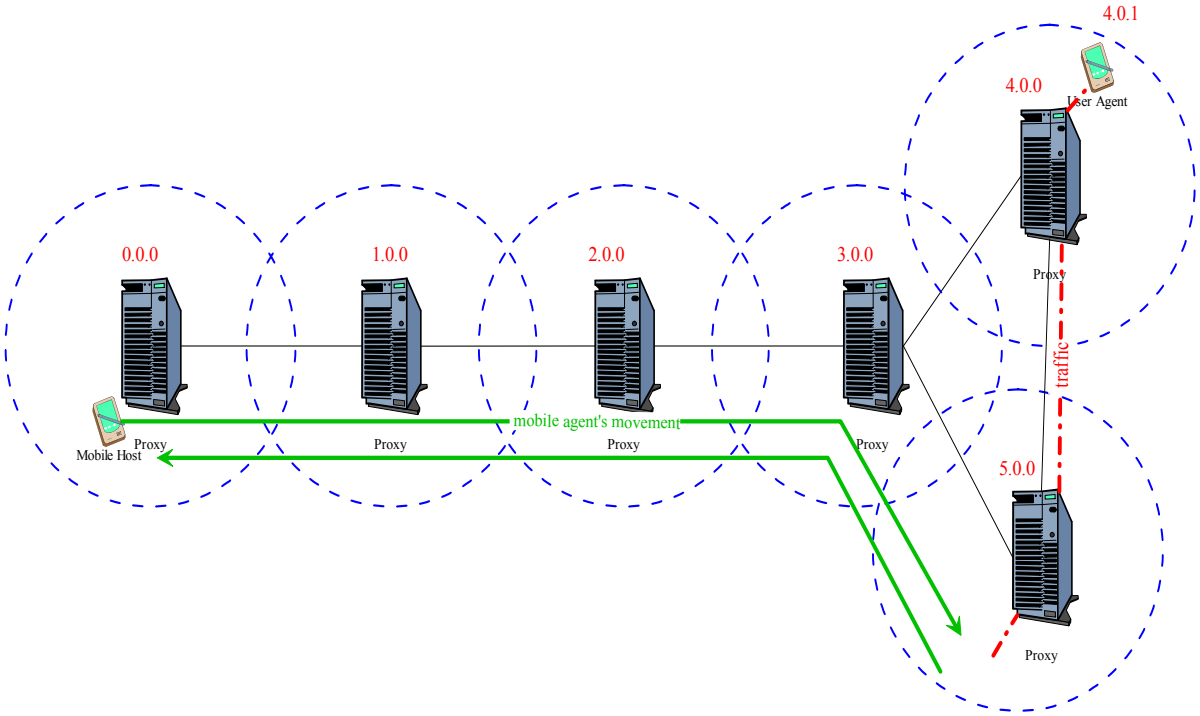
Εκτός από τις τροποποιήσεις στα παραπάνω αρχεία, για να ολοκληρωθεί η διαδικασία εισαγωγής της εφαρμογής μας στο NSv2.27, θα πρέπει να τροποποιηθεί το αρχείο Makefile.in που βρίσκεται στο ευρετήριο ~/ns2.27/ όπως φαίνεται παρακάτω και μετά να εκτελεστούν οι εντολές ./configure και make για να γίνει compile όλο το NS.

```
…
OBJ_CC = \
        …
        apps/methods.o apps/sip.o \
        …

NS_TCL_LIB = \
        …
        tcl/lib/ns-sip.tcl \
        …
```

Παράρτημα Β – Παράδειγμα σε Tcl

Ένα παράδειγμα τυπικού σεναρίου που υλοποιήθηκε σε Tcl στο NS παρατίθεται πιο κάτω. Στο παράδειγμα αυτό δημιουργείται ένα δίκτυο στο οποίο η σηματοδοσία γίνεται με βάση το Πρωτόκολλο Έναρξης Συνόδου (SIP). Όπως φαίνεται και στο πιο κάτω σχήμα, υπάρχουν δύο κινητοί κόμβοι οι οποίοι έχουν ενεργοποιήσει τους πράκτορες χρήστη που ακούνε στη θύρα 5060 για πιθανές αιτήσεις από άλλους πράκτορες χρήστη ή από πληρεξούσιους εξυπηρετητές. Εκτός από τους κινητούς κόμβους, υπάρχουν και 6 σταθμοί βάσης οι οποίοι λειτουργούν τόσο σαν πληρεξούσιοι εξυπηρετητές όσο και σαν εξυπηρετητές εγγραφών στις περιοχές μέσα στις οποίες εδρεύουν. Οι διευθύνσεις που βρίσκονται πάνω από κάθε κόμβο, αντιπροσωπεύουν τις διευθύνσεις IP του κάθε κόμβου. Ο ένας χρήστης UA1 (με IP 0.07) ο οποίος βρίσκεται κάτω αριστερά στο σχήμα, ξεκινά να κινείται κατά μήκος της πράσινης διαδρομής ενώ ο άλλος UA2 (με IP 4.0.1) παραμένει μέσα στην περιοχή του. Καθώς ο UA1 εισέρχεται στην περιοχή του πληρεξούσιου εξυπηρετητή με IP 1.0.0 (στα 75 sec) στέλνει μια αίτηση προς τον UA2 (μήνυμα INVITE) με σκοπό να ενεργοποιήσει μια αμφίδρομη (bi-directional) σύνδεση σταθερού ρυθμού 16Kbps (π.χ. κλήση VoIP) και επίσης (στο 80 sec) στέλνει μια άλλη αίτηση προς το UA2 με σκοπό να ενεργοποιήσει μια σύνδεση video streaming (100Kbps download και 2Kbps upload) οι οποίες γίνονται αποδεκτές και έτσι αρχίζουν οι συνδέσεις οι οποίες διαρκούν μέχρι ο UA1 επανέλθει μέσω της αντίστροφης διαδρομής στην περιοχή του πληρεξούσιου εξυπηρετητή με IP 0.0.0.



Σχήμα 76: Παράδειγμα τοπολογίας.

```
# ======================================================================
# Define options
# ======================================================================
set opt(chan)           Channel/WirelessChannel     ;# channel type
set opt(prop)           Propagation/TwoRayGround    ;# radio-propagation model
set opt(netif)          Phy/WirelessPhy             ;# network interface type
set opt(mac)            Mac/802_11                  ;# MAC type
set opt(ifq)            Queue/DropTail/PriQueue     ;# interface queue type
set opt(ll)             LL                          ;# link layer type
set opt(ant)            Antenna/OmniAntenna         ;# antenna model
set opt(ifqlen)         100                 ;# max packet in ifq
set opt(nn)             2                   ;# number of mobilenodes
set opt(adhocRouting)   DSDV            ;# routing protocol
set opt(x)      3300                    ;# x coordinate of topology
set opt(y)      3300                    ;# y coordinate of topology
set opt(seed)   0.0                     ;# seed for random number gen.
set opt(stop)   420                     ;# time to stop simulation
set num_bs_nodes        6
# ============================================================================
# check for boundary parameters and random seed
if { $opt(x) == 0 || $opt(y) == 0 } {
        puts "No X-Y boundary values given for wireless topology\n"
}
if {$opt(seed) > 0} {
        puts "Seeding Random number generator with $opt(seed)\n"
        ns-random $opt(seed)
}
# create simulator instance
set ns_   [new Simulator]
# set up for hierarchical routing
$ns_ node-config -addressType hierarchical
AddrParams set domain_num_ 6            ;# number of domains
lappend cluster_num 1 1 1 1 1 1         ;# number of clusters in each domain
AddrParams set cluster_num_ $cluster_num
lappend eilastlevel 8 5 5 5 10 3                ;# number of nodes in each cluster
```

```
AddrParams set nodes_num_ $eilastlevel  ;# of each domain

set tracefd  [open scenario.tr w]
$ns_ trace-all $tracefd
set namtrace [open scenario.nam w]
$ns_ namtrace-all-wireless $namtrace $opt(x) $opt(y)
# Create topography object
set topo   [new Topography]
# define topology
$topo load_flatgrid $opt(x) $opt(y)
# create God
create-god [expr $opt(nn) + $num_bs_nodes]
# configure for base-station node
$ns_ node-config -adhocRouting $opt(adhocRouting) \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propType $opt(prop) \
                 -phyType $opt(netif) \
                 -channelType $opt(chan) \
                 -topoInstance $topo \
                 -SIP ON \
                 -wiredRouting ON \
                 -agentTrace ON \
                 -routerTrace OFF \
                 -macTrace OFF \
                 -movementTrace OFF
#################################################################
#create base-station node
set BS(0) [$ns_ node 0.0.0]
$BS(0) random-motion 0              ;# disable random motion
#provide some co-ord (fixed) to base station node
$BS(0) set X_ 171.0
$BS(0) set Y_ 201.0
$BS(0) set Z_ 0.0
#################################################################
#create base-station node
set BS(1) [$ns_ node 1.0.0]
$BS(1) random-motion 0              ;# disable random motion
#provide some co-ord (fixed) to base station node
$BS(1) set X_ 586.0
$BS(1) set Y_ 201.0
$BS(1) set Z_ 0.0
#################################################################
#create base-station node
set BS(2) [$ns_ node 2.0.0]
$BS(2) random-motion 0              ;# disable random motion
#provide some co-ord (fixed) to base station node
$BS(2) set X_ 1001.0
$BS(2) set Y_ 201.0
$BS(2) set Z_ 0.0
#################################################################
#create base-station node
set BS(3) [$ns_ node 3.0.0]
$BS(3) random-motion 0              ;# disable random motion
#provide some co-ord (fixed) to base station node
$BS(3) set X_ 1416.0
$BS(3) set Y_ 201.0
$BS(3) set Z_ 0.0
#################################################################
#create base-station node
set BS(4) [$ns_ node 4.0.0]
$BS(4) random-motion 0              ;# disable random motion
#provide some co-ord (fixed) to base station node
$BS(4) set X_ 1616.0
$BS(4) set Y_ 401.0
$BS(4) set Z_ 0.0
#################################################################
#create base-station node
set BS(5) [$ns_ node 5.0.0]
$BS(5) random-motion 0              ;# disable random motion
#provide some co-ord (fixed) to base station node
$BS(5) set X_ 1616.0
```

```
$BS(5) set Y_ 1.0
$BS(5) set Z_ 0.0
#############################################################
# create mobilenodes
#configure for mobilenodes
$ns_ node-config -wiredRouting OFF

set node_(0) [$ns_ node 0.0.7]
$node_(0) base-station [AddrParams addr2id [$BS(0) node-addr]]
set node_(1) [$ns_ node 4.0.1]
$node_(1) base-station [AddrParams addr2id [$BS(4) node-addr]]

#create links between BS nodes
$ns_ duplex-link $BS(0) $BS(1) 200Mb 400ms DropTail
$ns_ duplex-link $BS(1) $BS(2) 200Mb 400ms DropTail
$ns_ duplex-link $BS(2) $BS(3) 200Mb 400ms DropTail
$ns_ duplex-link $BS(3) $BS(4) 200Mb 400ms DropTail
$ns_ duplex-link $BS(3) $BS(5) 200Mb 400ms DropTail
$ns_ duplex-link $BS(4) $BS(5) 200Mb 400ms DropTail
#***********************************
#Get instances of sip agents
for {set i 0} {$i < $opt(nn) } {incr i} {
   set sipU($i)  [$node_($i) sipag]
}
for {set i 0} {$i < $num_bs_nodes } {incr i} {
   set sipP($i)  [$BS($i) sipag]
}
#***********************************
#           First Traffic
#***********************************
set udp0 [new Agent/UDP]
set null0 [new Agent/Null]
set udp1 [new Agent/UDP]
set null1 [new Agent/Null]
$ns_ attach-agent $node_(0) $udp0
$ns_ attach-agent $node_(0) $null1
$ns_ attach-agent $node_(1) $udp1
$ns_ attach-agent $node_(1) $null0
#Create a CBR traffic source and attach it to a node
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 48
$cbr0 set rate_ 16kb
$cbr0 attach-agent $udp0
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 48
$cbr1 set rate_ 16kb
$cbr1 attach-agent $udp1
$ns_ connect $udp0 $null0
$ns_ connect $udp1 $null1
#***********************************
#       Second Traffic
#***********************************
set udp2 [new Agent/UDP]
set null2 [new Agent/Null]
set udp3 [new Agent/UDP]
set null3 [new Agent/Null]
$ns_ attach-agent $node_(0) $udp2
$ns_ attach-agent $node_(0) $null3
$ns_ attach-agent $node_(1) $udp3
$ns_ attach-agent $node_(1) $null2
#Create a CBR traffic source and attach it to a node
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 48
$cbr2 set rate_ 2kb
$cbr2 attach-agent $udp2
set cbr3 [new Application/Traffic/CBR]
$cbr3 set packetSize_ 48
$cbr3 set rate_ 100kb
$cbr3 attach-agent $udp3
$ns_ connect $udp2 $null2
$ns_ connect $udp3 $null3

$ns_ connect $sipU(0) $sipU(1)

$udp0 set class_ 1
```

```
$udp1 set class_ 2
$udp2 set class_ 3
$udp3 set class_ 4
$ns_ color 1 Blue
$ns_ color 2 Red
$ns_ color 3 Green
$ns_ color 4 Yellow
#************************************
#  Agents names on single node
#************************************
$sipU(0) node_(0) agents udp0 null1 udp2 null3
$sipU(1) node_(1) agents null0 udp1 null2 udp3
#************************************
#  SIP agents initialization
#************************************
for {set i 0} {$i < $num_bs_nodes} {incr i} {
    $ns_ at 0.0 "$sipP($i) open"
}
for {set i 0} {$i < $opt(nn) } {incr i} {
        $ns_ at [expr $i + 1] "$sipU($i) open"
}
#************************************
#   Traffic Scenario
#************************************
$ns_ at 75.0  "$sipU(0) invite cbr0 cbr1"
$ns_ at 80.0  "$sipU(0) invite cbr2 cbr3"
$ns_ at 410.0  "$sipU(0) bye cbr2"
$ns_ at 415.0  "$sipU(1) bye cbr1"
#$ns_ at 19.0  "$sipU(0) message THRYLE_THEE_MOU"
#$ns_ at 200.0 "$sipU(1) message OLYMPIAKE_MOU"
#************************************
$ns_ at 5.000000000000 "$node_(0) setdest 1416.00 205.00 8.0"
$ns_ at 170.0000000000 "$node_(0) setdest 1600.00 1.0000 7.0"
$ns_ at 210.0000000000 "$node_(0) setdest 1416.00 205.00 7.0"
$ns_ at 250.0000000000 "$node_(0) setdest 171.000 205.00 8.0"
$node_(0) set Z_ 0.0000
$node_(0) set Y_ 205.00
$node_(0) set X_ 171.00
$node_(1) set Z_ 0.0000
$node_(1) set Y_ 501.00
$node_(1) set X_ 1716.0
# Define initial node position in nam
for {set i 0} {$i < $opt(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
}
# Sip agents and nodes' termination
for {set i 0} {$i < $opt(nn) } {incr i} {
    $ns_ at $opt(stop).0 "$node_($i) reset";
    $ns_ at $opt(stop).0001 "delete $sipU($i)"
}
for {set i 0 } {$i < [expr $num_bs_nodes + 0 ]} {incr i} {
    $ns_ at $opt(stop).0 "$BS($i) reset";
    $ns_ at $opt(stop).0001 "delete $sipP($i)"
}
$ns_ at $opt(stop).0003 "puts \"NS EXITING...\" ; $ns_ halt"
$ns_ at $opt(stop).0002 "stop"

proc stop {} {
    global ns_ tracefd namtrace
    $ns_ flush-trace
    close $tracefd
    close $namtrace
    exec nam scenario.nam &
    exit 0
}
# informative headers for CMUTracefile
puts $tracefd "M 0.0 nn $opt(nn) x $opt(x) y $opt(y) rp $opt(adhocRouting)"
puts $tracefd "M 0.0 prop $opt(prop) ant $opt(ant)"
puts "Starting Simulation..."
$ns_ run
```

Παράρτημα Γ – Χρήσιμες Οδηγίες

Σε αυτό το παράρτημα δίνονται οδηγίες για το πώς ένας χρήστης μπορεί να χρησιμοποιήσει και να δημιουργήσει σε script TCL τους SIP Agents που έχουμε ενσωματώσει στο περιβάλλον προσομοίωσης του NS-2.

- Αρχικά πρέπει να ορίσουμε σε κάθε προς δημιουργία κόμβο ότι θα χρησιμοποιεί το πρωτόκολλο SIP. Αυτό γίνεται θέτοντας την επιλογή SIP σε ON, μαζί με τα άλλα χαρακτηριστικά των κόμβων, όπως φαίνεται πιο κάτω:

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

- Χρησιμοποιώντας την tcl συνάρτηση *sipag* δίνουμε ένα οποιοδήποτε όνομα σε ένα SIP Agent για να μπορέσουμε στη συνέχεια να χρησιμοποιήσουμε τις συναρτήσεις του. Η εντολή που βλέπουμε παρακάτω δίνει το όνομα sipU0 στον SIP Agent που δημιουργήθηκε στον κόμβο node(0).

```
set sipU0  [$node_(0) sipag]
```

- Χρησιμοποιώντας την tcl συνάρτηση *agents* ορίζουμε για ποιους Agents κίνησης (π.χ. UDP agent, TCP agent) είναι υπεύθυνος ο κάθε SIP Agent. Όπως έχουμε αναφέρει έχουμε διαχωρίσει το επίπεδο σηματοδοσίας (Control Plane) από το επίπεδο χρήστη (User Plane) σε κάθε κόμβο. Έτσι η πιο κάτω εντολή ορίζει ότι ο SIP Agent με όνομα sipU0 (σηματοδοσία) που βρίσκεται πάνω στον κόμβο node_(0) είναι υπεύθυνος για τους agent udp0 και null1(κίνηση).

```
$sipU0 node_(0) agents udp0 null1
```

- Για να θέσουμε σε λειτουργία τον κάθε SIP Agent, είτε πρόκειται για Proxy είτε User, χρειάζεται να εκτελέσουμε την tcl συνάρτηση *open*.

```
#agent initialization
$ns_ at 0.0 "$sipP0 open"
```

- Για να ξεκινήσουμε κάποια κίνηση χρησιμοποιούμε την tcl συνάρτηση *invite.* Στο παρακάτω παράδειγμα στο χρόνο 120.0 αρχίζει η σηματοδοσία SIP για την κίνηση cbr1. Όταν ολοκληρωθεί η σηματοδοσία θα αρχίσει η κίνηση cbr1.

```
$ns_ at 120.0 "$sipU0 invite cbr1"
```

- Για να τερματίσουμε μια κίνηση χρησιμοποιούμε την tcl συνάρτηση *bye.* Στο παρακάτω παράδειγμα στο χρόνο 200.0 αρχίζει η σηματοδοσία SIP για να τερματίσει την κίνηση cbr1.

```
$ns_ at 200.0 "$sipU0 bye cbr1"
```

- Για να στείλουμε κάποιο στιγμιαίο μήνυμα (Instance Message) χρησιμοποιούμε την tcl συνάρτηση *message.*

```
$ns_ at 200.0 "$sipU1 message Hello_World"
```

- Ο τερματισμός των SIP Agents γίνεται με την tcl συνάρτηση *delete*.

```
#SIP agents termination
$ns_ at 250.0 "delete $sipP0"
```

- Όσον αφορά τις διευθύνσεις που μπορούμε να δίνουμε στους διάφορους κόμβους σημειώνουμε τα εξής:
    1. Ακολουθούμε ιεραρχική τριών επιπέδων διευθυνσιδιοδότηση (hierarchical addressing) τριών επιπέδων, δηλαδή της μορφής X.Y.Z. Το X είναι για τα domains, το Y αφορά τα clusters που βρίσκονται μέσα σε ένα συγκεκριμένο domain και το Z χαρακτηρίζει τους host μέσα σε ένα cluster.
    2. Οι διευθύνσεις που πρέπει να δίνονται στους Proxy-Registrar Servers πρέπει να είναι της μορφής X.0.0. Κάναμε αυτή τη σύμβαση (που θυμίζει τις γνωστές μας διευθύνσεις υποδικτύων που τελειώνουν με μηδενικά στο κλασικό Internet) για να έχουμε πάντα υπόψη ότι ένας Proxy-Registrar Server είναι υπεύθυνος για όλους τους χρήστες στην περιοχή-υποδίκτυό του.
    3. Επίσης, για τις διευθύνσεις που μπορούν να έχουν οι SIP User Agents ισχύει ότι $0<Z<10$. Αντίστοιχα, ένας ενδιάμεσος κόμβος που απλά κάνει δρομολόγηση IP (δε θα ενσωματώνει SIP User Agent ή Proxy Server) μπορεί να έχει οποιαδήποτε διεύθυνση για την οποία $Z>10$ ή δεν ανήκει στο domain κάποιου υπάρχοντος SIP Proxy Server.

- Τέλος, όλες οι προσομοιώσεις SIP οφείλουν να χρησιμοποιούν DSDV Routing Agent.

Σημειώσεις

Ευρετήριο

Παραπομπές

[1] Alan B. Johnston , SIP: Understanding the Session Initiation Protocol. -2nd ed.- (Artech House telecommunication library).

[2] C.Perkins, "IP mobility support", Request for Comments (Proposed Standard) 2002, Internet Engineering Task Force, Oct. 1996.

[3] C. Perkins and D. Johnson, "Route optimization in mobile IP", Internet Draft, Internet Engineering Task Force, Feb 1999

[4] E.Wedlund and H. Schulzrinne, "Mobility support using SIP," in Second ACM/IEEE International Conference on Wireless and Mobile Multimedia (WoWMoM'99), (Seattle,Washington), Aug. 1999.

[5] H. Schulzrinne, J. Rosenberg, "The Session Initiation Protocol: Internet – Centric Signalling", IEEE Communications Magazine, October 2000.

[6] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments 2543, Internet Engineering Task Force, Mar.1999.

[7] E.Wedlund and H. Schulzrinne, "Application-Layer Mobility Using SIP", Mobile Computing and Communications Review, Volume 1, Number 2.

[8] RADVISION, Understanding SIP Servers.

[9] RADVISION, SIP Protocol Overview.

[10] UC Berkeley, The ns Manual (formerly ns Notes and Documentation), VINT Project, December 2003.