ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ
ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Σχεδίαση και ανάπτυξη λογισμικού για τη χρήση του Πρωτοκόλλου Έναρξης Συνόδου (SIP) ως γενικευμένου πρωτοκόλλου σηματοδοσίας σε κινητά δίκτυα μετά την τρίτη γενιά (Beyond 3G Networks)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

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

Επιβλέπων : Ιάκωβος Βενιέρης
Καθηγητής Ε.Μ.Π

Αθήνα, Ιούνιος 2005

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ
ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Σχεδίαση και ανάπτυξη λογισμικού για τη χρήση του Πρωτοκόλλου Έναρξης Συνόδου (SIP) ως γενικευμένου πρωτοκόλλου σηματοδοσίας σε κινητά δίκτυα μετά την τρίτη γενιά (Beyond 3G Networks)

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, Ιούνιος 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

Το Πρωτόκολλο Έναρξης Συνόδου (Session Initiation Protocol, SIP) χρησιμοποιείται ευρέως για τον έλεγχο συνόδων υπηρεσιών πολυμέσων. Χαρακτηριστική εφαρμογή του είναι η υιοθέτησή του από την περιοχή IMS (IP Multimedia Subsystem domain) των κινητών δικτύων τρίτης γενιάς για τον έλεγχο κλήσεων που εμπλέκουν πολυμέσα.

Με την εξέλιξη των κινητών δικτύων μετά την τρίτη γενιά (Beyond 3G, B3G δίκτυα) και την ευρύτατη διείσδυση του IP στη στοίβα πρωτοκόλλων, το SIP παρουσιάζεται κατάλληλο ως γενικευμένο πρωτόκολλο σηματοδοσίας σε τέτοιου τύπου δίκτυα. Με άλλα λόγια, εκτός από τον κλασικό από άκρο σε άκρο έλεγχο συνόδου, το SIP καλείται να αναλάβει κάθε μορφής σηματοδοσία, όπως τη σηματοδοσία εντός του δικτύου πυρήνα (διαχείριση κινητικότητας και συνόδου) και τη σηματοδοσία στο σύνορο των δικτύων πυρήνα και πρόσβασης, αντικαθιστώντας το πρωτόκολλο RANAP.

Η παραπάνω πρόταση μπορεί να επιτευχθεί με την ενοποίηση των κόμβων δικτύου κορμού των συστημάτων κινητών επικοινωνιών τρίτης γενιάς GGSN (Gateway GPRS Support Node) και SGSN (Serving GPRS Support Node) σε ένα μοναδικό κόμβο που θα καλείται RASN (Radio Access Support Node) και ο οποίος θα αναλαμβάνει τη διαχείριση κινητικότητας και συνόδου των κινητών τερματικών, αλληλεπιδρώντας κατάλληλα με τα δίκτυα πρόσβασης και τις οντότητες του δικτύου κορμού, με αποκλειστική χρήση μηνυμάτων SIP.

Σκοπός της συγκεκριμένης διπλωματικής εργασίας είναι η διερεύνηση εφαρμογής της συγκεκριμένης πρότασης και των αλλαγών που απαιτούνται στους κόμβους του δικτύου για την υποστήριξή της. Στα πλαίσιά της υλοποιήθηκε ο κόμβος RASN, που αποτελεί βασική οντότητα της αρχιτεκτονικής του δικτύου πυρήνα.

Λέξεις Κλειδιά

Πρωτόκολλο Έναρξης Συνόδου (SIP), GGSN, SGSN, RASN, δίκτυα 3^{ης} γενιάς, UMTS, GPRS, raw socket

Abstract

Session Initiation Protocol (SIP) is widely used for session control in multimedia sessions. A characteristic media session control use of SIP is in the IMS (IP Multimedia Subsystem) domain of 3rd generation (3G) networks.

After the further development of networks beyond 3G (B3G) and the wide spreading of IP in protocol stack, SIP seems to be the most appropriate general signaling protocol in such networks. In other words, beside the classic end-to-end session control, SIP aims to take over any signaling procedure, such as signaling in the core network (mobility and session management) and between the core network and the radio access network by replacing the RANAP protocol.

The above can come true by unifying the nodes of 3G networks GGSN (Gateway GPRS Support Node) and SGSN (Serving GPRS Support Node) in one node called RASN (Radio Access Support Node). RASN node will undertake mobility management and session control of mobile terminals by interacting with radio access and core network nodes using SIP.

The purpose of our diploma thesis is the exploration and implementation of the previous suggestion and of all the necessary changes that have to take place in the network nodes for its support. To cut a long story short, we implemented RASN node which is of essential value for the core network.

Keywords

Session Initiation Protocol (SIP), GGSN, SGSN, RASN, 3rd Generation mobile networks, UMTS, GPRS, raw socket

Εισαγωγή

Παρ' όλη την ραγδαία ανάπτυξη των κινητών επικοινωνιών τα τελευταία χρόνια, οι συνεχώς αυξανόμενες ανάγκες του σύγχρονου απαιτητικού ανθρώπου δεν φαίνεται να καλύπτονται. Οι πρωταρχικές υπηρεσίες φωνής, σύντομων γραπτών μηνυμάτων (SMS) καθώς και μηνυμάτων πολυμέσων (MMS) που παρέχονται από τα δίκτυα 2^{ης} και 2.5 γενιάς (GSM – GPRS) τείνουν να γίνουν ανεπαρκείς καθώς νέες υπηρεσίες πολυμέσων και πρόσβασης στο Διαδίκτυο εισβάλλουν για τα καλά στην σύγχρονη καθημερινότητα μας μέσω των δικτύων 3^{ης} γενιάς και ιδιαίτερα του UMTS.

Κυριότερες οντότητες της νέας αυτής αρχιτεκτονικής των δικτύων 3^{ης} γενιάς, αποτελούν οι κόμβοι GGSN και SGSN, οι οποίοι είναι υπεύθυνοι για το μέρος του δικτύου που εμπλέκει τη μεταγωγή πακέτου. Επίσης η υιοθέτηση της περιοχής IMS (UMTS Έκδοση 5) όσον αφορά τον έλεγχο κλήσεων που εμπλέκουν πολυμέσα αποτελεί μια ουσιαστική καινοτομία της νέας αρχιτεκτονικής.

Στη παρούσα διπλωματική εργασία, αντικειμενικός μας στόχος ήταν ενοποίηση των κόμβων δικτύου κορμού των συστημάτων κινητών επικοινωνιών τρίτης γενιάς SGSN και GGSN σε ένα μοναδικό κόμβο που θα καλείται RASN και ο οποίος θα αναλαμβάνει τη διαχείριση κινητικότητας και συνόδου των κινητών τερματικών, αλληλεπιδρώντας κατάλληλα με τα δίκτυα πρόσβασης και τις οντότητες του δικτύου κορμού, με αποκλειστική χρήση μηνυμάτων SIP. Έτσι, πλησιάζουμε ένα βήμα πιο κοντά στα δίκτυα All-IP, μειώνεται το κόστος σηματοδοσίας, απλοποιούνται οι διαδικασίες εγγραφής και πιστοποίησης αυθεντικότητας, αποφεύγεται η ενθυλάκωση στο δίκτυο πυρήνα και επιταχύνονται όλες οι διαδικασίες χωρίς απώλεια λειτουργικότητας.

Παρακάτω, στο πρώτο κεφάλαιο γίνεται μια εκτενής αναφορά τόσο στην αρχιτεκτονική και τα πρωτόκολλα των δικτύων 3^{ης} γενιάς σύμφωνα με τις συστάσεις της 3GPP όσο και στις απαραίτητες διαδικασίες που λαμβάνουν χώρα σε τέτοια δίκτυα. Στη συνέχεια, το δεύτερο κεφάλαιο αφορά στην προταθείσα αρχιτεκτονική από το SAILOR Project και την υλοποίηση των παραπάνω διαδικασιών. Ακόμα, ακολουθεί ο σχεδιασμός και η εφαρμογή της παραπάνω αρχιτεκτονικής και κλείνουμε με τα συμπεράσματα της διπλωματικής εργασίας.

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 έδιναν κατά τη διάρκεια εκπόνησης αυτής της διπλωματικής εργασίας. Πιστεύουμε ότι ένα ιδιαίτερο ευχαριστώ αξίζει ο Βαγγέλης για την υπομονή που υπέδειξε στην απάντηση απορριών σχετιζόμενες με την υλοποίηση του λογισμικού του κόμβου RASN.

1. Δίκτυα 3G σύμφωνα με τις συστάσεις της 3GPP

Μετά την επανάσταση που έφεραν τα δίκτυα 2ης γενιάς, η ανάγκη για παροχή ευρυζωνικών υπηρεσιών μέσω των κινητών τερματικών, οδήγησε στην ανάπτυξη των δικτύων 3ης γενιάς. Τα δίκτυα 2ης γενιάς έχουν τη δυνατότητα παροχής υπηρεσιών φωνής και σύντομων γραπτών μηνυμάτων μόνο, ενώ αυτά της 3ης, υποστηρίζουν και άλλες υπηρεσίες – όπως το video κατ' απαίτηση (video on demand) – οι οποίες είναι πιο απαιτητικές σε εύρος ζώνης.

1.1 Αρχιτεκτονική δικτύου και πρωτοκόλλων

Ένα αντιπροσωπευτικό παράδειγμα δικτύου 3ης γενιάς είναι το UMTS (Universal Mobile Terrestrial System) του οποίου την αρχιτεκτονική θα αναλύσουμε παρακάτω. Το δίκτυο UMTS χωρίζεται σε 3 βασικά μέρη:

- air interface,
- UMTS Terrestrial Radio Access Network (UTRAN),
- core network (δίκτυο πυρήνα).

1.1.1 Οντότητες στο UMTS

Όσον αφορά στις οντότητες που αποτελούν το δίκτυο πυρήνα του UMTS παρατηρούμε τους κόμβους MSC, G-MSC, HLR, and VLR που προϋπήρχαν στο δίκτυο GSM, που όμως έχουν αυξημένη λειτουργικότητα έτσι ώστε να ανταποκρίνονται στις απαιτήσεις των δικτύων 3ης γενιάς. Οι κόμβοι αυτοί αποτελούν το δίκτυο μεταγωγής κυκλώματος που αποτελεί υποσύνολο του δικτύου πυρήνα.

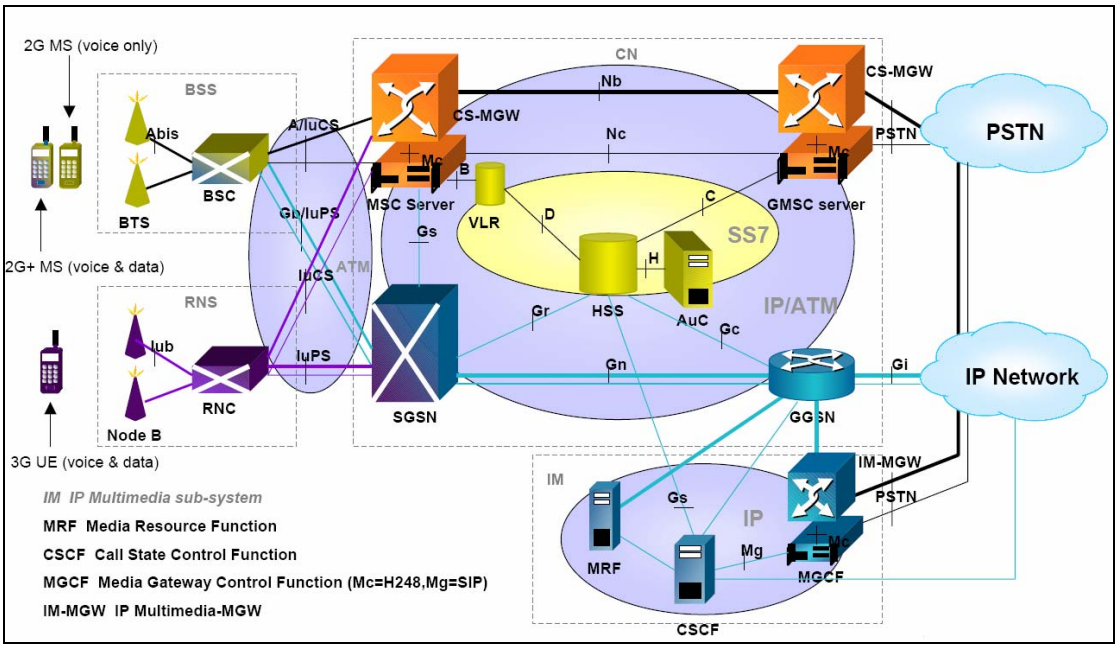
Επιπλέον, στο δίκτυο πυρήνα υπάρχουν και δύο νέες οντότητες· ο κόμβος SGSN (Serving GPRS Support Node) και ο κόμβος GGSN (Gateway GPRS Support Node) που αποτελούν το δίκτυο μεταγωγής κυκλώματος. Στην έκδοση 5 του UMTS (UMTS Release 5) είχαμε την εισαγωγή μιας νέας περιοχής στο δίκτυο πυρήνα η οποία ονομάζεται IMS (IP Multimedia Subsystem). Ο ρόλος της είναι η παροχή νέων υπηρεσιών όπως multimedia conferences (πχ φωνή, video και whiteboard), παιχνίδια με αλληλεπίδραση (interactive games) και άλλα. Με την εμφάνιση της νέας αυτής περιοχής είχαμε την παρουσίαση τριών βασικών καινοτομιών όπως η εγκατάσταση και ο έλεγχος κλήσης/συνόδου, η περιαγωγή (roaming) και η χρήση IPv6. Μέσα στην περιοχή IMS υπάρχουν οντότητες όπως ο εξυπηρετητής CSCF (Call Server Control

Function), ο MGCF (Media Gateway Control Function), ο MRF (Multimedia Resource Function) και ο MGW (Media Gateway).

Το τμήμα UTRAN του δικτύου UMTS περιλαμβάνει τις οντότητες RNC και Node B που αντιστοιχούν στους κόμβους BSC και BTS που υπάρχουν στο GSM.

Η διεπαφή ανάμεσα στο UTRAN και στο δίκτυο κορμού χωρίζεται σε Iu-PS και Iu-CS ανάλογα με το αν αναφερόμαστε στο δίκτυο μεταγωγής πακέτου (packet switched) ή κυκλώματος (circuit switched). Τα πακέτα δρομολογούνται μέσα από το κόμβο SGSN (Serving GPRS Support Node). Ο συνοριακός κόμβος μεταξύ του δικτύου κορμού και του διαδικτύου είναι ο GGSN (Gateway GPRS Support Node) ο οποίος δρομολογεί τα πακέτα από το δίκτυο πυρήνα προς το δίκτυο IP.

Οι διάφορες οντότητες και η αρχιτεκτονική που περιγράψαμε φαίνονται στο παρακάτω σχήμα (βλέπε σχήμα 1).



Σχήμα 1: Αρχιτεκτονική δικτύου UMTS Release 5.

Τα κυριότερα χαρακτηριστικά μερικών από τους κόμβους που έχουμε αναφέρει είναι τα εξής:

- RNC

  Είναι ένας ATM μεταγωγέας που μπορεί να πολυπλέξει/αποπολυπλέξει δεδομένα. Οι RNCs είναι συνδεδεμένοι μεταξύ τους και γι' αυτό μπορούν να διαχειριστούν

αυτόνομα θέματα δέσμευσης ασύρματων πόρων. Πολλοί Node Bs μπορούν να είναι συνδεδεμένοι σε έναν RNC και κάθε RNC ελέγχει τη συμφόρηση και τις ήπιες διαπομπές.

- SGSN

Ο κόμβος αυτός κάνει τη διαχείριση των συνόδων και παράγει πληροφορίες χρέωσης. Επίσης δρομολογεί πακέτα στο σωστό RNC και χειρίζεται διαδικασίες όπως οι attach/detach και η εγκατάσταση συνόδων.

- GGSN

Λειτουργεί όπως ένας συνοριακός δρομολογητής (border router), έχει firewall και μεθόδους διευθυνσιοδότησης. Ακόμα μπορεί να προωθήσει αιτήσεις για υπηρεσίες σε κόντινα τοπικά δίκτυα. Τέλος, ο GGSN μπορεί να παράξει πληροφορίες χρέωσης.

- CSCF

Μια νέα οντότητα που παρουσιάστηκε στην έκδοση 5 είναι ο εξυπηρετητής CSCF (Call Server Control Function) ο οποίος λειτουργεί σαν ένας Πληρεξούσιος Εξυπηρετητής SIP του οποίου οι κύριες λειτουργίες είναι να:

α) Εντοπίζει τους χρήστες – μετάφραση SIP URLs σε διευθύνσεις IP,

β) Ενεργεί εκ μέρους άλλων οντοτήτων προωθώντας τα μηνύματα INVITE,

γ) Διατηρεί πληροφορίες σχετικά με τη κατάσταση μιας συνόδου για να επιτρέπει άλλες ροές πολυμέσων να προστεθούν σε μια ήδη υπάρχουσα σύνοδο καθώς και για τη χρέωση.

Η οντότητα αυτή χωρίζεται σε τρία στοιχεία· τον P-CSCF (Proxy CSCF), τον S-CSCF (Serving CSCF) και τον I-CSCF (Interrogating CSCF).

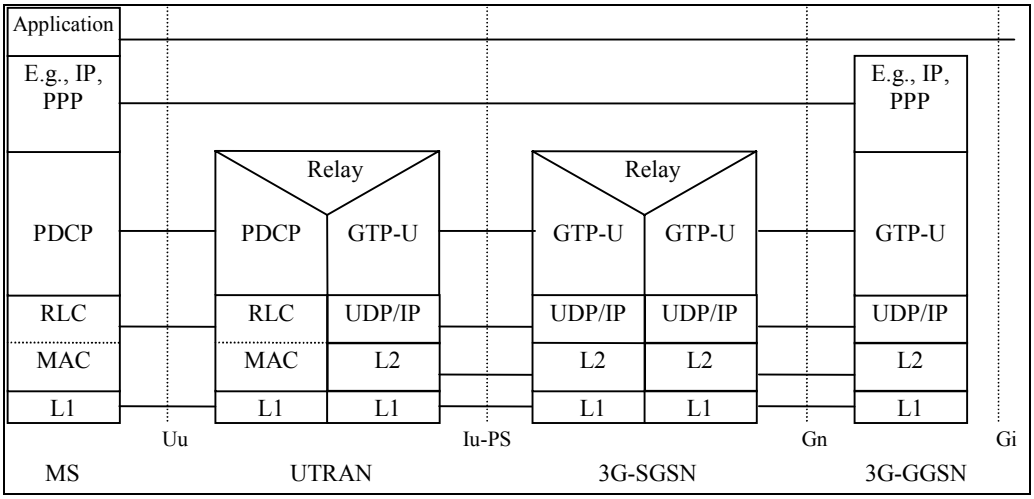
1.1.2 Στοίβα πρωτόκολλων

Από την άλλη, εκτός από τον ορισμό των φυσικών οντοτήτων που απαρτίζουν το UMTS, αναγκαίος είναι και ο ορισμός της στοίβας πρωτοκόλλων που βρίσκεται σε κάθε μία από τις παραπάνω οντότητες καθώς αυτή είναι υπεύθυνη για τη ιεραρχική, διαφανή λειτουργία και «συνεργασία» τους.

Όπως μας είναι ήδη γνωστό, η κάθετη κατάτμηση της ιεραρχίας των πρωτοκόλλων ορίζει το επίπεδο χρήστη (user plane) και το επίπεδο ελέγχου (control plane) που είναι υπεύθυνα για τα δεδομένα του χρήστη και τη σηματοδοσία αντίστοιχα. Τα δύο σχήματα που ακολουθούν παρουσιάζουν την οριζόντια κατάτμηση για κάθε ένα από

τα επίπεδα που αναφέραμε. Αρχικά θα δούμε το επίπεδο χρήστη και ύστερα το επίπεδο ελέγχου με τις απαραίτητες επεξηγήσεις.

Το επίπεδο χρήστη είναι αυτό που αναλαμβάνει τη μεταφορά των δεδομένων από το κινητό τερματικό στον κόμβο GGSN και από εκεί προς όλο το δίκτυο. Αποτελείται από μια σειρά από πρωτόκολλα που παρέχουν τη μεταφορά της πληροφορίας μαζί με μια σειρά από άλλες λειτουργίες ελέγχου, όπως ο έλεγχος ροής, η ανίχνευση και η διόρθωση λαθών. Το σχήμα που ακολουθεί παρουσιάζει τα πρωτόκολλα αυτά. Στη συνέχεια θα αναλυθεί η λειτουργία των κυριότερων.



Σχήμα 2: Επίπεδο χρήστη δικτύου UMTS.

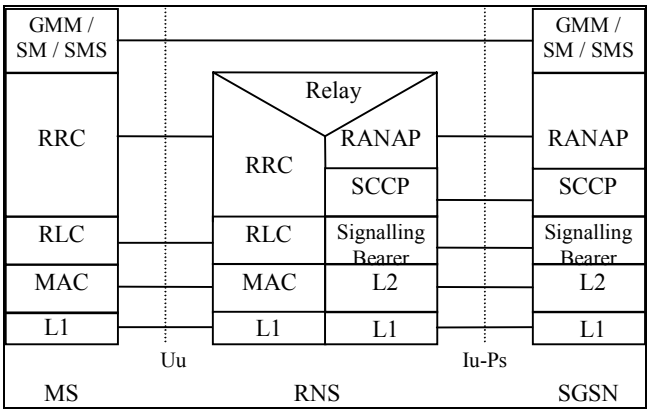
Το PDCP (Packet Data Convergence Protocol) εξασφαλίζει τη διαφάνεια πρωτοκόλλων χαμηλότερων στρωμάτων από άλλα πρωτόκολλα υψηλότερων στρωμάτων. Τα RLC (Radio Link Control) και MAC (Medium Access Control) εξασφαλίζουν μια αξιόπιστη λογική ζεύξη στο ασύρματο μέσο και κάνουν έλεγχο πρόσβασης αντίστοιχα. Σημειωτέον ότι κάθε RLC ζεύξη έχει τη δική της ταυτότητα (bearer id) και ένας κινητός σταθμός μπορεί να έχει πολλές τέτοιες ζεύξεις. Τα UDP/IP είναι τα γνωστά μας πρωτόκολλα που χρησιμοποιούνται για τη δρομολόγηση των δεδομένων και τη σηματοδοσία. Όσον αφορά στα L1, L2 είναι τα γνωστά πρωτόκολλα του φυσικού επίπεδου και ζεύξης δεδομένων αντίστοιχα. Τέλος, το GTP-U (GPRS Tunnelling Protocol for the user plane) είναι ένα πρωτόκολλο που χρησιμοποιεί τη μέθοδο της σήραγγας (tunnelling) για τη μεταφορά δεδομένων από τον κόμβο RNC (του δικτύου UTRAN) στον SGSN και από τον SGSN στον GGSN. Χρησιμοποιώντας το GTP, το UMTS έχει την δυνατότητα να μεταφέρει διάφορα είδη

πακέτων (όπως τα IPv4, IPv6, PPP και X.25) μέσω της ίδιας υποδομής (infrastructure).

Παρατηρώντας το σχήμα (βλέπε σχήμα 2) βλέπουμε και τα αντίστοιχα σημεία αναφοράς (reference points) που συνδέουν τα τμήματα και τους κόμβος του UMTS δικτύου.

Σ' αυτό το σημείο πρέπει να αναφερθεί ότι στα ανώτερα στρώματα η επικοινωνία εξασφαλίζεται από τα κατώτερα διαφανώς, όπως άλλωστε ορίζουν και οι αρχές τις διαστρωμάτωσης. Έτσι, το PPP πρωτόκολλο όπως και το επίπεδο εφαρμογής (application layer) δεν «αντιλαμβάνονται» την ύπαρξη των κατώτερων στρωμάτων. Ακόμα, πρέπει να πούμε ότι το SIP πρωτόκολλο σηματοδοσίας λειτουργεί στο επίπεδο εφαρμογής και τα μηνύματα SIP μεταφέρονται από το επίπεδο χρήστη σαν κανονικά δεδομένα. Το SIP αποτελεί ένα πρωτόκολλο βασισμένο σε κείμενο (text-based) και θα περιγραφεί αναλυτικότερα στο Παράρτημα Β.

Από την άλλη, όπως έχουμε ήδη αναφέρει, υπάρχει και το επίπεδο ελέγχου. Αυτό είναι υπεύθυνο για τον έλεγχο και την υποστήριξη των λειτουργιών του επιπέδου χρήστη. Συνοπτικά, ελέγχει τις λειτουργίες attach και detach (βλέπε §1.2.1), ελέγχει τα χαρακτηριστικά ήδη υπαρχουσών συνδέσεων, εξασφαλίζει την κινητικότητα χρήστη και εκχωρεί πόρους ανάλογα με την κατάσταση του δικτύου. Το σχήμα που ακολουθεί δείχνει την αρχιτεκτονική των πρωτοκόλλων μεταξύ κινητού σταθμού και SGSN καθώς αυτοί οι δύο κόμβοι κάνουν όλη τη σηματοδοσία για τη διαχείριση κινητικότητας χρήστη (mobility management) και συνόδου (session management) και στη συνέχεια επεξήγονται τα σημαντικότερα σημεία.



Σχήμα 3: Επίπεδο ελέγχου MS – SGSN.

Το RANAP είναι το πρωτόκολλο σηματοδοσίας που αναλαμβάνει να κάνει μια σειρά από ενέργειες ελέγχου που είναι απαραίτητες για την ομαλή και αδιάλειπτη λειτουργία του δικτύου. Τέτοιες είναι:

- διαχείριση RAB (Radio Access Bearer),
- απελευθέρωση πόρων,
- μεταφορά πληροφορίας NAS μεταξύ χρήστη και δικτύου κορμού,
- αλλαγή του ήδη επυπηρετούντος RNC (serving RNC relocation).

Ακόμα, το RRC (Radio Resource Control), όπως υποδηλοί και το όνομά του ελέγχει τους ασύρματους πόρους του δικτύου. Τα GMM/SM υποστηρίζουν λειτουργίες διαχείρισης κινητικότητας (πχ attach, detach) και συνόδου (πχ PDP context activation, PDP context deactivation) αντίστοιχα. Τέλος, το SCCP (Signalling Connection Control Part) χρησιμοποιείται για να μεταφέρονται RANAP μηνύματα.

1.2 Διαδικασίες (Procedures in 3GPP)

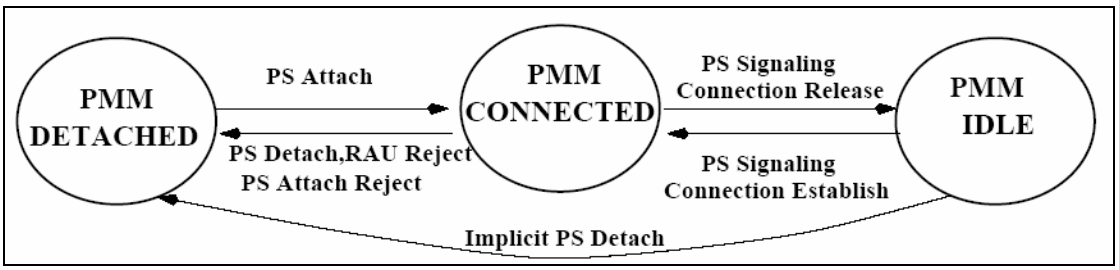
Οι κυριότερες διαδικασίες που ακολουθούνται στα δίκτυα $3^{ης}$ γενιάς αφορούν στη διαχείριση κινητικότητας, στη διαχείριση θέσης καθώς και στην ενεργοποίηση PDP context.

1.2.1 Διαδικασίες διαχείρισης κινητικότητας (Mobility Management Procedures)

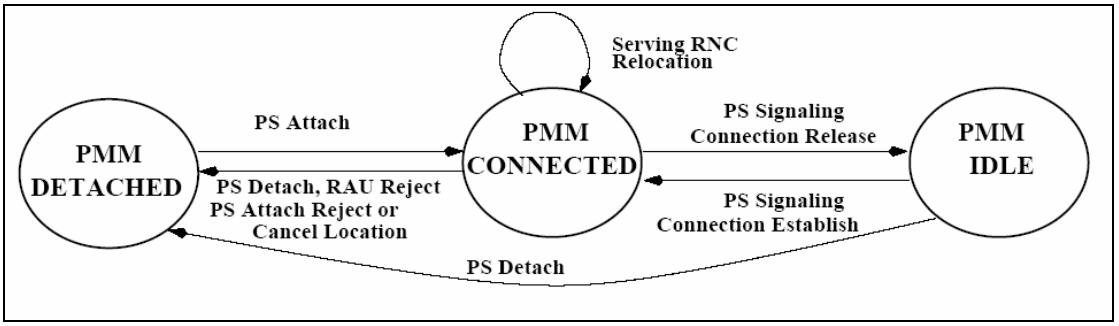
Στο UMTS, τόσο ο κόμβος SGSN όσο και το κινητό τερματικό, χαρακτηρίζονται από μια μηχανή πεπερασμένων καταστάσεων (finite state machine). Οι ενέργειες που γίνονται για την διαχείριση κινητικότητας του κινητού τερματικού εξαρτώνται από την εκάστοτε κατάσταση που βρίσκονται οι δυο αυτές οντότητες. Οι καταστάσεις αυτές είναι η PMM-DETACHED, η PMM-CONNECTED και η PMM IDLE (βλέπε σχήματα 4 και 5). Ακολουθεί μια σύντομη περιγραφή τους:

- PMM-DETACHED. Το δίκτυο UMTS δεν γνωρίζει για το κινητό τερματικό, δηλαδή το κινητό τερματικό δεν είναι προσβάσιμο από το δίκτυο.
- PMM-CONNECTED. Στην κατάσταση αυτή υπάρχει κανάλι σηματοδοσίας για συνδέσεις μεταγωγής πακέτου μεταξύ του κινητού τερματικού και του κόμβου SGSN. Πακέτα μπορούν να παραδίδονται μόνο όταν είμαστε στην κατάσταση αυτή. Η διαδικασία S-RNC Relocation μπορεί επίσης να εκτελεστεί καθώς είμαστε στην κατάσταση αυτή.

- PMM-IDLE. Το κινητό τερματικό έχει εκτελέσει τη διαδικασία GPRS Attach (η οποία παρουσιάζεται πιο κάτω). Στην κατάσταση αυτή το κινητό τερματικό μπορεί να εκτελέσει τη διαδικασία GPRS Detach.



Σχήμα 4: Καταστάσεις Διαχείρισης Κινητικότητας που αφορούν το κινητό τερματικό.



Σχήμα 5: Καταστάσεις Διαχείρισης Κινητικότητας που αφορούν τον κόμβο SGSN.

Αφού έχουμε αναφερθεί στις καταστάσεις διαχείρισης κινητικότητας θα περιγράψουμε παρακάτω τις κυριότερες διαδικασίες διαχείρισης κινητικότητας που συναντούμε στα δίκτυα UMTS. Θα εξετάσουμε τις διαδικασίες GPRS Attach και GPRS Detach οι οποίες χρησιμοποιούν το πρωτόκολλο RANAP για τη μετάδοση των μηνυμάτων στη διεπαφή Iu.

Διαδικασία GPRS Attach

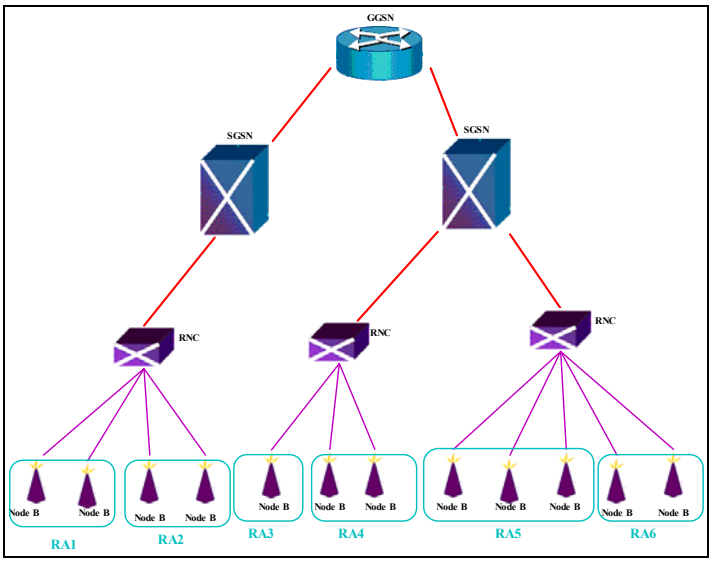
Με τη διαδικασία Attach το κινητό τερματικό ενημερώνει το δίκτυο (συγκεκριμένα τον κόμβο SGSN) για την παρουσία του έτσι ώστε να μπορεί να αποκτήσει πρόσβαση στις υπηρεσίες GPRS. Ειδικότερα γνωστοποιεί τη ταυτότητα της περιοχής δρομολόγησής του (Routing Area Identifier) καθώς και την προσωρινή του ταυτότητα P-TMSI.

Διαδικασία GPRS Detach

Με τη διαδικασία Detach το κινητό τερματικό σταματά να έχει πρόσβαση στις προσφερόμενες από τον SGSN υπηρεσίες.

Διαδικασίες διαχείρισης θέσης (Location Management Procedures)

Οι κυριότερες διαδικασίες διαχείρισης θέσης είναι οι Routing Area Update (RAU) και Intra & Inter- S-RNC Relocation (active/no active sessions).



Σχήμα 6: Περιοχές δρομολόγησης (Routing Areas – RAs).

Routing Area Update

Η διαδικασία RAU εκτελείται είτε περιοδικά είτε όταν ένα κινητό τερματικό αλλάξει περιοχή δρομολόγησης (βλέπε σχήμα 6). Η διαδικασία αυτή έχει σκοπό την ενημέρωση του δικτύου για την τρέχουσα θέση του κινητού τερματικού. Διακρίνουμε δύο περιπτώσεις:

- Intra SGSN RAU

  Πρόκειται για την περίπτωση όπου το κινητό τερματικό εγκαταλείπει μια RA για να εισέλθει σε μια νέα που ελέγχεται από τον ίδιο κόμβο SGSN. Τότε δεν είναι απαραίτητο ο κόμβος SGSN να ενημερώσει τους κόμβους GGSN και HLR για τη νέα RA του κινητού τερματικού. Σημειωτέον ότι τα περιοδικά μηνύματα RAU ανήκουν στην κατηγορία αυτή.

- Inter SGSN RAU

  Πρόκειται για την περίπτωση όπου το κινητό τερματικό εγκαταλείπει μια RA για να εισέλθει σε μια νέα που δεν ελέγχεται από τον ίδιο κόμβο SGSN. Τότε οι κόμβοι GGSN και HLR ενημερώνονται.

Intra & Inter- S-RNC Relocation (active/no active sessions)

Η διαδικασία αυτή λαμβάνει χώρα μόνο όταν το κινητό βρίσκεται στην κατάσταση PMM-CONNECTED όπου στη διεπαφή Iur μεταφέρονται τόσο μηνύματα ελέγχου σηματοδοσίας όσο και δεδομένα χρήστη, στη περίπτωση που το κινητό τερματικό αλλάζει σημείο σύνδεσης στο δίκτυο μετακινούμενο από τον source RNC (παλιός RNC) στον target RNC (νέος RNC). Αν ο target RNC και ο source RNC είναι συνδεδεμένοι στον ίδιο κόμβο SGSN τότε πραγματοποιείται η διαδικασία Intra-SGSN S-RNC Relocation. Αν έχουμε ταυτοχρόνως αλλαγή της RA τότε ακολουθεί η διαδικασία Intra SGSN RAU την οποία περιγράψαμε πιο πάνω. Από την άλλη αν ο target RNC και ο source RNC δεν είναι συνδεδεμένοι στον ίδιο κόμβο SGSN τότε πραγματοποιείται η διαδικασία Inter-SGSN S-RNC Relocation η οποία ακουθείται από τη διαδικασία Inter SGSN RAU.

1.2.2 Διαδικασία ενεργοποίησης PDP context (PDP Context Activation Procedure)

Για να μπορεί ένα κινητό τερματικό να στείλει ή να λάβει πακέτα σε μια σύνδεση μεταγωγής πακέτου, θα πρέπει να εγκαταστήσει ένα PDP context (Packet Data Protocol context). Μέσω της διαδικασίας αυτής το τερματικό ανακτά μια διεύθυνση IP. Το PDP context περιγράφει τη σύνδεση προς ένα εξωτερικό δίκτυο πακέτου όπως είναι το Διαδίκτυο. Τα βήματα που ακολουθούνται για την εγκατάσταση ενός PDP context είναι τα ακόλουθα:

α) Το τερματικό αιτείται την ενεργοποίηση του PDP context (PDP context activation).

β) Ο κόμβος SGSN εξακριβώνει την εγκυρότητα της αίτησης συγκρίνοντας την με κάποιες πληροφορίες που λαμβάνει από τον HLR.

γ) Ο κόμβος SGSN στέλνει μια αίτηση στον εξυπηρετητή DNS έτσι ώστε να μάθει τη διεύθυνση IP του κόμβου GGSN.

δ) Ο κόμβος SGSN προσπαθεί να εγκαταστήσει τα κανάλια επικοινωνίας RAB (Radio Access Bearers) σύμφωνα με τη ποιότητα υπηρεσίας (QoS) που έχει τύχει διαπραγμάτευσης.

ε) Ο κόμβος SGSN στέλνει μήνυμα δημιουργίας PDP context στον κόμβο GGSN (αίτημα που μπορεί να γίνει ή να μη γίνει δεκτό λόγω υπερφόρτωσης του κόμβου GGSN).

στ) Μια σήραγγα IP (IP tunnel) εγκαθίσταται μεταξύ των κόμβων SGSN και GGSN.

ζ) Μια διεύθυνση PDP εκχωρείται στο τερματικό.

η) Το PDP context αποθηκεύεται στους κόμβους SGSN, GGSN και HLR.

1.3 Παροχή υπηρεσιών πολυμέσων (Service execution)

Η προσθήκη της περιοχής IMS στα δίκτυα 3ης γενιάς αποτέλεσε την απαρχή για την παροχή υπηρεσιών πολυμέσων. Μια περιγραφή των κυριοτέρων στοιχείων που ανήκουν στην περιοχή IMS δίνεται παρακάτω σε συνδυασμό με τις διαδικασίες που ακολουθούνται για την εγγραφή ενός κινητού τερματικού καθώς και την ενεργοποίηση υπηρεσίας.

1.3.1 Περιοχή IMS (IMS domain)

Όπως έχουμε αναφέρει στην §1.1.1, μια σημαντική καινοτομία που παρουσιάστηκε στην έκδοση 5 του UMTS ήταν η εισαγωγή μιας νέας περιοχής που ονομάζεται IMS (IP Multimedia Subsystem) και κυρίως των στοιχείων P-CSCF, S-CSCF και I-CSCF που την απαρτίζουν. Η λειτουργία των στοιχείων αυτών βασίζεται στο Πρωτόκολλο Έναρξης Συνόδου (SIP).

Ο P-CSCF (Proxy CSCF) αποτελεί το πρώτο σημείο επαφής του κινητού τερματικού με τη περιοχή IMS. Μόλις λάβει μήνυμα REGISTER από το κινητό τερματικό (συνδρομητής), το προωθεί στον I-CSCF του οικείου δικτύου του συνδρομητή (ο P-CSCF μπορεί να προσδιορίσει το οικείο δίκτυο του συνδρομητή χρησιμοποιώντας το IMSI ή το SIP URI του).

Ο S-CSCF (Serving CSCF) χειρίζεται τις διάφορες καταστάσεις στις οποίες μπορεί να βρεθεί μια σύνοδος (session) μέσα σε ένα δίκτυο. Το στοιχείο αυτό έχει μεγαλύτερη λειτουργικότητα από ότι ο Proxy CSCF και ο Interrogating CSCF γιατί έχει πρόσβαση σε όλους τους πόρους που απαιτούνται για τη παροχή υπηρεσιών πχ εξυπηρετητές video και media gateways.

Τέλος, ο I-CSCF (Interrogating CSCF) αποτελεί το σημείο επαφής με το δίκτυο κάποιου παρόχου για όλες τις συνδέσεις IMS οι οποίες προορίζονται για ένα συνδρομητή αυτού του παρόχου καθώς επίσης και για όλες τις συνδέσεις IMS οι οποίες προέρχονται από ένα συνδρομητή αυτού του παρόχου (του οποίου αποτελεί το οικείο δίκτυο) που βρίσκεται στην περιοχή εξυπηρέτησης άλλου παρόχου (εκτελεί υπηρεσίες περιαγωγής). Ο I-CSCF ενός οικείου δικτύου, ανακτά πληροφορίες για

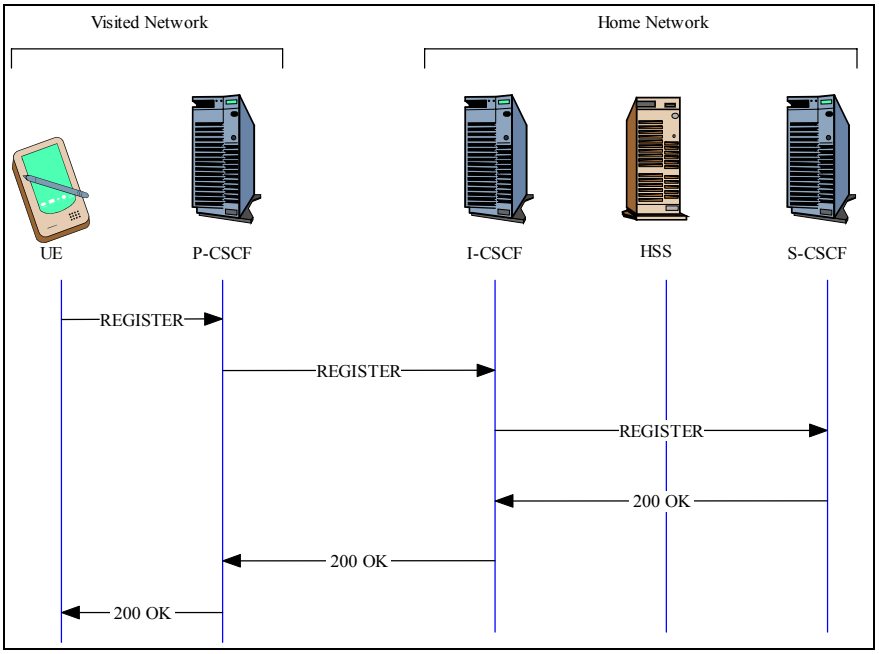
κάποιο συνδρομητή από το κόμβο HSS, χρησιμοποιώντας LDAP και τις κατανέμει στους P-CSCF και S-CSCF.

1.3.2 P-CSCF Discovery

Για να μπορεί ένας χρήστης να χρησιμοποιήσει υπηρεσίες της περιοχής IMS ή για να έχει την ικανότητα να δέχεται κλήσεις/συνόδους IM πρέπει πρώτα να εγγραφεί στο δίκτυο. Αυτό γίνεται εφικτό μέσω ενός πληρεξούσιου εξυπηρετητή (P-CSCF) είτε ο χρήστης βρίσκεται στο οικείο του δίκτυο είτε όχι. Ο P-CSCF παρέχει υποστήριξη συνόδων πολυμέσων αλλά επίσης ενεργεί και σαν firewall προς την περιοχή IMS. Το κινητό τερματικό αρχικά ενεργοποιεί ένα PDP context για σκοπούς σηματοδοσίας και εγγραφής στο δίκτυο. Με τον τρόπο αυτό ανακτά – στατικά ή δυναμικά – μια διεύθυνση IP και στέλνει μια αίτηση DNS στον τοπικό DNS εξυπηρετητή (που βρίσκεται στον κόμβο GGSN) ο οποίος του αποκρίνεται με τη διεύθυνση IP του P-CSCF.

1.3.3 Εγγραφή στην περιοχή IMS σε επίπεδο εφαρμογής (Application level Registration with the IMS)

Με τη διαδικασία αυτή το κινητό τερματικό επιθυμεί να ενημερώσει την περιοχή IMS για την διεύθυνση IP που του έχει εκχωρηθεί κατά τη διάρκεια ενεργοποίησης του PDP context. Η διαδικασία της εγγραφής επιτυγχάνεται μέσω μηνύματος SIP-REGISTER το οποίο μεταφέρεται μέσω του επιπέδου χρήστη (user plane) που έχει εγκατασταθεί μετά την ενεργοποίηση του PDP context προς τον S-CSCF ο οποίος παίζει το ρόλο της οντότητας SIP Registrar. Το παρακάτω σχήμα (βλέπε σχήμα 7) απεικονίζει τα διάφορα στάδια μέχρι την ολοκλήρωση της εγγραφής του κινητού τερματικού στην περιοχή IMS.

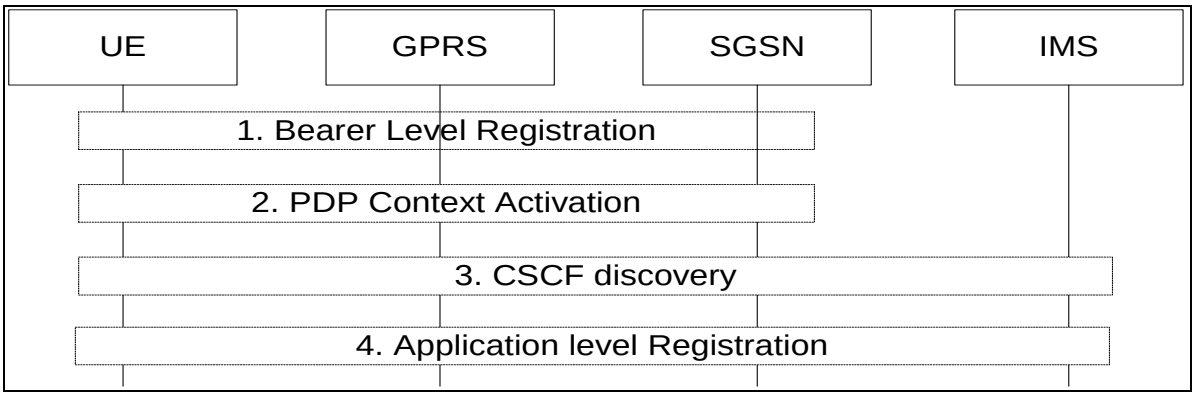


Σχήμα 7: Εγγραφή στην περιοχή IMS σε επίπεδο στρώματος εφαρμογής.

1.3.4 Ενεργοποίηση υπηρεσίας (Service activation)

Μετά την επιτυχή ολοκλήρωση των βημάτων που αναφέρθηκαν στην προηγούμενη παράγραφο, ο χρήστης μπορεί να αρχικοποιήσει ή να αποδεκτεί μια σύνοδο SIP αποστέλλοντας ή λαμβάνοντας μήνυμα SIP-INVITE αντίστοιχα. Αξίζει να αναφερθεί ότι ο P-CSCF παίζει το ρόλο του SIP Proxy Server.

Στο παρακάτω σχήμα (βλέπε σχήμα 8) παρουσιάζονται τα βήματα που χρειάζεται να εκτελεστούν έτσι ώστε το κινητό τερματικό να εγκαταστήσει μια επιτυχημένη σύνοδο SIP, για να μπορεί να αποκτήσει πρόσβαση σε υπηρεσίες που προσφέρονται από την περιοχή IMS.



Σχήμα 8: Βήματα που απαιτούνται για την εγκατάσταση μιας συνόδου SIP στα δίκτυα UMTS

2. Ανάλυση και Σχεδίαση προταθείσας αρχιτεκτονικής

Το ευρωπαϊκό πρόγραμμα SAILOR πρότεινε το 2002 την ενοποίηση των κόμβων SGSN και GGSN σε μια ενιαία οντότητα που θα ονομάζεται κόμβος RASN. Η πρόταση αυτή στόχευε στην εξάλειψη κάποιων αδυναμιών που αντιμετώπιζαν τα δίκτυα 3^{ης} γενιάς. Στο κεφάλαιο αυτό γίνεται μια εκτενής αναφορά στην προταθείσα αρχιτεκτονική καθώς και στα πλεονεκτήματα που αυτή συνεπάγεται.

2.1 Μειονεκτήματα δικτύων UMTS

Ένα από τα κυριότερα μειονεκτήματα που αντιμετωπίζουν τα δίκτυα UMTS είναι η αναγκαστική δρομολόγηση των δεδομένων του δικτύου μεταγωγής πακέτου (packet switched network) μέσω του κόμβου GGSN (GGSN anchoring). Αυτό έχει ως αποτέλεσμα την αύξηση του χρόνου διαπομπής κατά τη διαδικασία Inter- S-RNC Relocation (βλέπε §1.2.1) καθώς και την αύξηση της καθυστέρησης διάδοσης των δεδομένων (end to end delay) προς το διαδίκτυο στην περίπτωση που ένας χρήστης βρίσκεται μακριά από τον κόμβο GGSN (λόγω περιορισμένου αριθμού κόμβων GGSN ανά πάροχο). Το μειονέκτημα αυτό είναι ισοδύναμο με το πρόβλημα που αντιμετωπίζει το Mobile IP λόγω της αναγκαστικής δρομολόγησης των δεδομένων μέσω του οικείου πράκτορα (Home Agent).

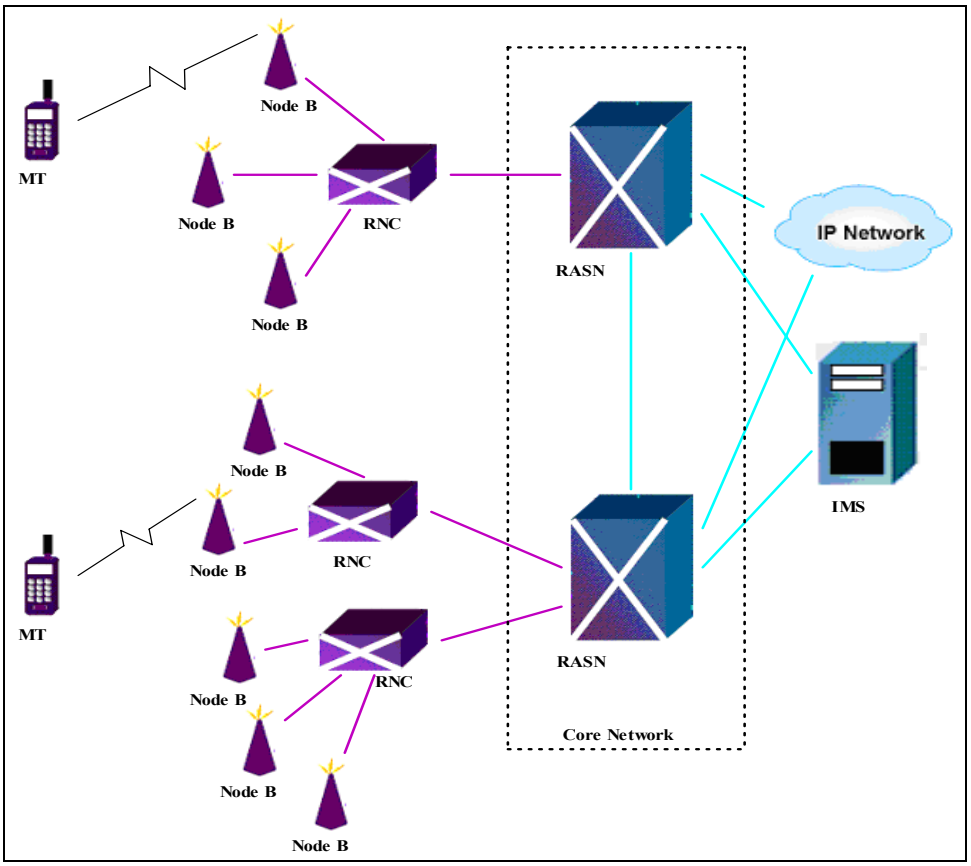
Ένα άλλο μειονέκτημα αποτελεί η επιβάρυνση (8 -12 bytes overhead) που εισάγεται λόγω της χρήσης του πρωτοκόλλου GTP-U στο δίκτυο πυρήνα (μεταξύ του SGSN και του GGSN). Αυτό οφείλεται στην πρόσθετη επικεφαλίδα που εισάγει το εν λόγω πρωτόκολλο για την ενθυλάκωση πακέτων IPv4, IPv6, PPP και X.25.

Θα ήταν παράλειψη να μην αναφέρουμε την ύπαρξη δύο ξεχωριστών διαδικασιών σηματοδοσίας για τον έλεγχο των υπηρεσιών που αφορούν στη μεταγωγή πακέτου. Αρχικά μέσω των μηνυμάτων NAS (που ακολουθούν τη διαδρομή MT-RNC-SGSN-GGSN) γίνονται οι διαδικασίες εγγραφής/διαγραφής (GPRS Attach/Detach) ενός κινητού τερματικού και έπειτα ακολουθεί η διαδικασία ενεργοποίησης PDP context για τη διάνοιξη καναλιού για τη μεταφορά δεδομένων. Εκτός της σηματοδοσίας αυτής υπάρχει και η σηματοδοσία επιπέδου εφαρμογής μέσω των μηνυμάτων του πρωτοκόλλου SIP (που ακολουθούν τη διαδρομή TE-IMS-called Party) για την ενεργοποίηση υπηρεσιών πολυμέσων (application level signalling). Τα δύο

διαφορετικά είδη σηματοδοσίας οδηγούν στην άσκοπη επανάληψη λειτουργιών που αφορούν στην ποιότητα υπηρεσίας (QoS), διαχείριση κινητικότητας (Mobility Management – MM) καθώς και στην εκτέλεση διαδικασιών ασφάλειας.

2.2 Αρχιτεκτονική Δικτύου

Στην νέα αρχιτεκτονική (βλέπε σχήμα 9) ουσιαστικά προτείνεται η αντικατάσταση των κόμβων SGSN και GGSN – που ανήκουν στο δίκτυο πυρήνα – με ένα νέο κόμβο ο οποίος θα ονομάζεται RASN (Radio Access Support Node) ο οποίος θα υλοποιεί τις λειτουργίες των κόμβων SGSN και GGSN. Με την εισαγωγή της καινοτομίας αυτής, το δίκτυο πυρήνα (όσον αφορά το κομμάτι που ασχολείται με τη μεταγωγή πακέτου) θα βασίζεται εξ' ολοκλήρου στο πρωτόκολλο IP αφού τα μέχρι πρότινως χρησιμοποιούμενα μηνύματα NAS (Non Access Stratum) πρόκειται να αντικατασταθούν από μηνύματα SIP.



Σχήμα 9: Προταθείσα αρχιτεκτονική δικτύου.

Ο νέος κόμβος RASN επικοινωνεί με τις παρακάτω οντότητες:

α) Radio Access Network (RAN) για μηνύματα σηματοδοσίας καθώς και για δεδομένα χρήστη.

Στην περίπτωση μηνυμάτων σηματοδοσίας προβλέπονται τα ακόλουθα μηνύματα:

- μηνύματα RANAP για εγκατάσταση και απόλυση καναλιών RAB μέσω άμεσης επικοινωνίας που υπάρχει μεταξύ του κόμβου RASN και του RAN (κόμβοι RNC).
- μηνύματα SIP_{RAN} (βλέπε §2.3) μεταφέρονται μέσω του επιπέδου σηματοδοσίας δηλαδή πάνω από το πρωτόκολλο RANAP. Αφού πρόκειται για μηνύματα SIP_{RAN} τότε απαιτείται η χρήση parsers οι οποίοι να μετατρέπουν τα μηνύματα SIP σε μηνύματα SIP_{RAN} και αντίστροφα για τη προώθηση μηνυμάτων που ανταλλάζονται μεταξύ κινητού τερματικού και κόμβου P-CSCF (περιοχή IMS). Οι parsers αυτοί επικοινωνούν με ένα χώρο φύλαξης δεδομένων (data depository) για την εισαγωγή και ανάκτηση απαιτούμενων πληροφοριών όπως είναι το περιεχόμενο NAS (NAS context). Τονίζεται ότι η σηματοδοσία SIP είναι διαφανής στο δίκτυο πρόσβασης (Access Network) αφού αυτό προωθεί τα μηνύματα από και προς το κινητό τερματικό.

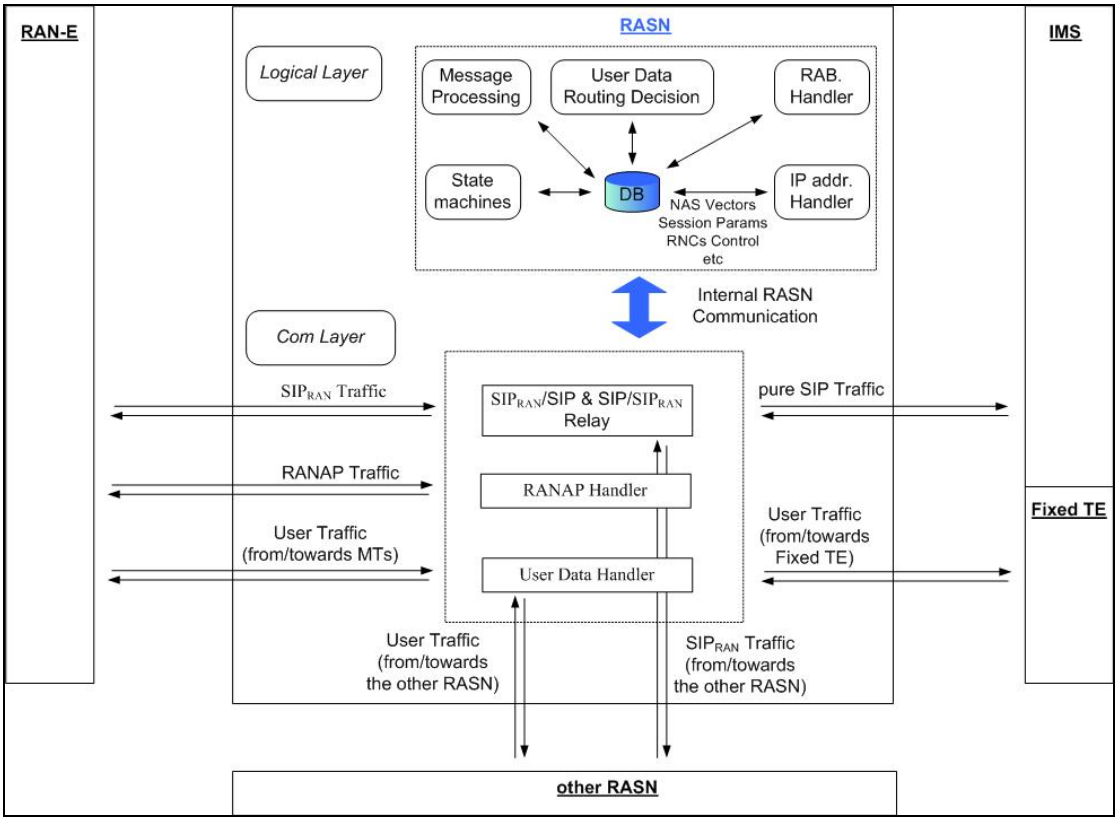
Στην περίπτωση δεδομένων χρήστη, αυτά μεταφέρονται μέσω GTP-U από RAN προς RASN και μέσω IP από τον RASN προς το διαδίκτυο.

β) Περιοχή IMS για σκοπούς σηματοδοσίας (μηνύματα SIP μέσα στο ωφέλιμο φορτίο των πακέτων IP).

γ) Correspondent Host για μηνύματα σηματοδοσίας καθώς και δεδομένα χρήστη. Αυτά περιλαμβάνουν μηνύματα SIP re-INVITE (continuous Inter-RASN mobility) καθώς και δεδομένα επιπέδου χρήστη (uplink, downlink).

δ) Άλλους κόμβους RASN για σκοπούς σηματοδοσίας (μέσω διεπαφής SIP_{RAN} πάνω από IP) στην περίπτωση που το τερματικό αλλάζει κόμβο RASN κατά τη διαρκεία της κίνησής του.

Ακολουθεί το μπλοκ διάγραμμα (βλέπε σχήμα 10) που δείχνει τη λειτουργία και την επικοινωνία του κόμβου RASN με τις άλλες οντότητες.

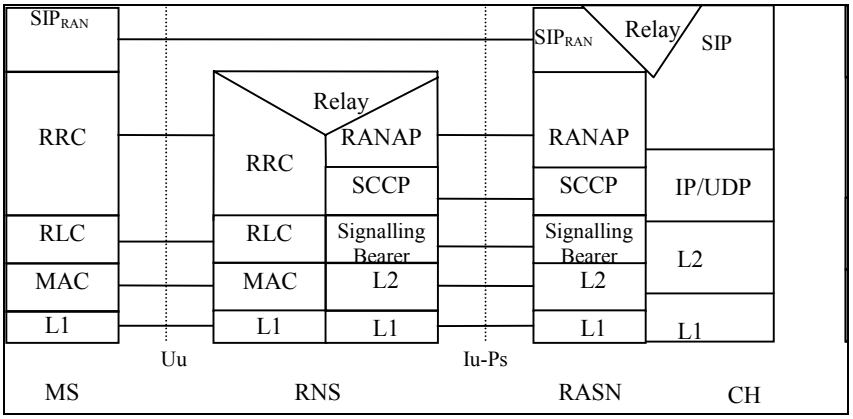


Σχήμα 10: Μπλοκ διάγραμμα κόμβου RASN.

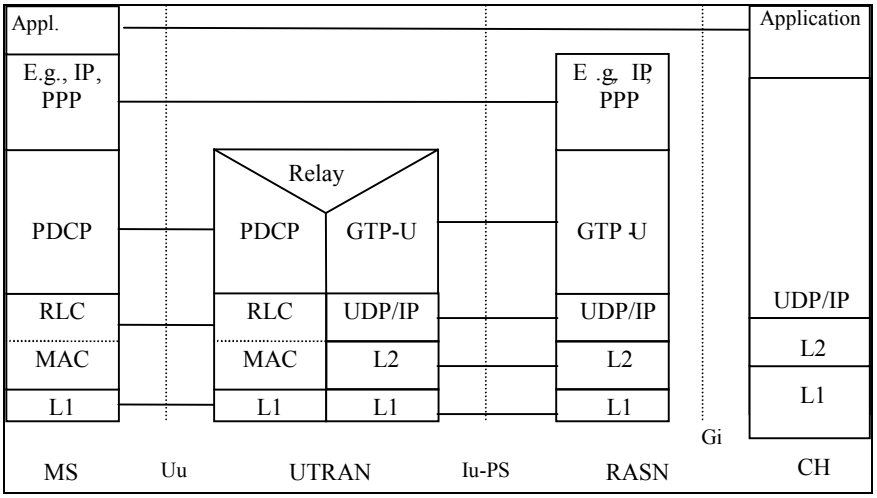
2.3 Πρωτόκολλο SIP_{RAN}

Σύμφωνα με τη προταθείσα αρχιτεκτονική, τα μηνύματα NAS πρόκειται να αντικατασταθούν από μηνύματα SIP_{RAN}. Αυτό θα έχει ως αποτέλεσμα την αποφυγή της εκτέλεσης διαδικασιών που ανήκουν σε διαφορετικά πρωτόκολλα και εξυπηρετούν τον ίδιο σκοπό όπως πιστοποίηση αυθεντικότητας (authentication), ανταλλαγή πληροφοριών κινητικότητας κτλ. Με τη προσέγγιση αυτή θα έχουμε μια σημαντική μείωση του φορτίου σηματοδοσίας το οποίο αντικατοπτρίζεται σε μείωση του αριθμού των μηνυμάτων που στέλνονται καθώς του συνολικού αριθμού bytes που ανταλλάζονται στη διεπαφή air interface.

Το σχήμα που παρουσιάζεται πιο κάτω παρουσιάζει τη στοίβα πρωτοκόλλων που αφορά τόσο στο επίπεδο ελέγχου (βλέπε σχήμα 11) όσο και στο επίπεδο χρήστη (βλέπε σχήμα 12).



Σχήμα 11: Στοίβα πρωτοκόλλων επιπέδου ελέγχου προταθείσας αρχιτεκτονικής.



Σχήμα 12: Στοίβα πρωτοκόλλων επιπέδου χρήστη προταθείσας αρχιτεκτονικής.

Τα μηνύματα SIP θα περνούν από το επίπεδο ελέγχου (control plane) και θα εκτελούν διαδικασίες που αφορούν στην κινητικότητα UMTS και διαχείριση συνόδου (μηνύματα NAS) καθώς επίσης τις εγγενείς λειτουργίες ελέγχου συνόδων του πρωτοκόλλου SIP (πχ για εφαρμογές πραγματικού χρόνου).

Ακόμα, επειδή τα μηνύματα SIP που φτάνουν στον κόμβο RASN περιέχουν και επιπρόσθετες πληροφορίες εκτός από αυτές που ορίζει η προτυποποίηση του πρωτοκόλλου, ο RASN πρέπει να εξασφαλίζει ότι τα μηνύματα που προωθεί προς το internet πρέπει να είναι «καθαρά» μηνύματα SIP.

Προκειμένου το SIP να μπορεί να λειτουργεί σαν ένα γενικευμένο πρωτόκολλο σηματοδοσίας γίνεται μια επέκταση του σε SIP_{RAN} το οποίο θα περιλαμβάνει και NAS πληροφορίες. Όσον αφορά στα SIP μηνύματα, ενώ ως τώρα μεταφέρονταν σε επίπεδο χρήστη αυτό δε συμβαίνει πια. Η διαδικάσια είχε ως εξής: η NAS

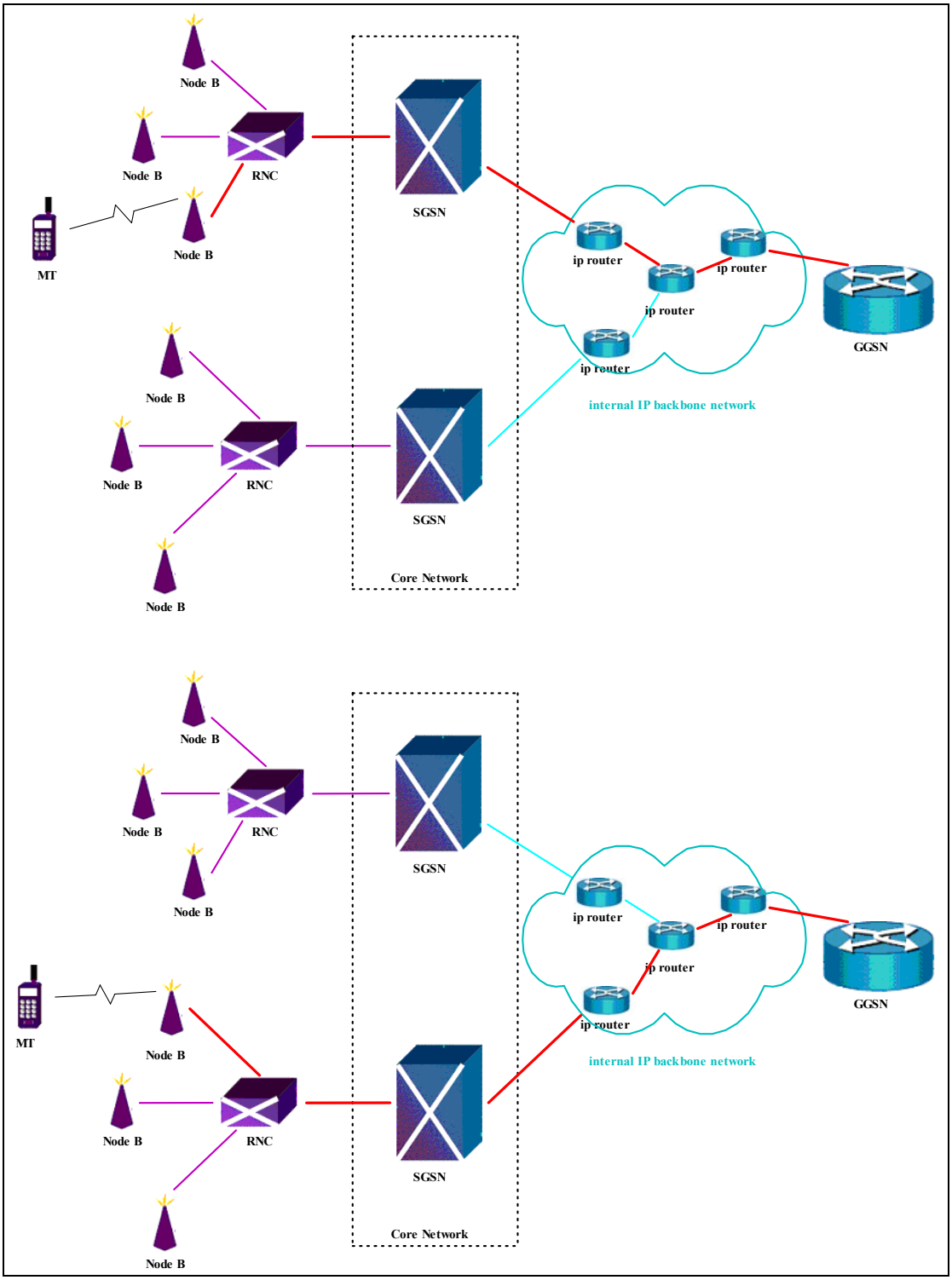
σηματοδοσία «φρόντιζε» πρώτα σε επίπεδο ελέγχου να δημιουργήσει κάποιο κανάλι (radio acess bearer) και ύστερα μεταφέρονταν τα μηνύματα SIP σε επίπεδο χρήστη. Με τη νέα πρόταση το επεκτεταμένο SIP (SIP_{RAN}) μεταφέρεται σε επίπεδο ελέγχου αφού στην ουσία η NAS σηματοδοσία αντικαταστάθηκε από τη SIP.

2.4 Πλεονεκτήματα

Αρκετά είναι τα πλεονεκτήματα που προκύπτουν από την προταθείσα αρχιτεκτονική τα οποία παρατίθενται πιο κάτω.

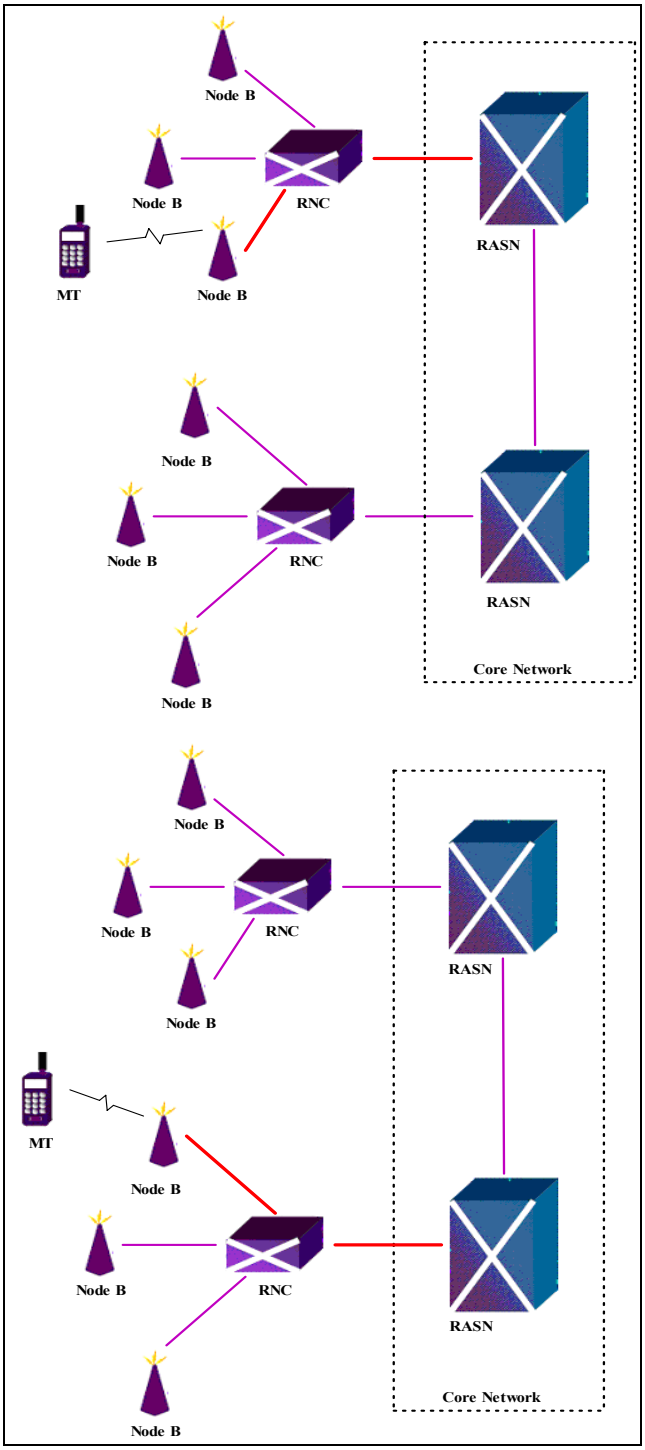
1. Το δίκτυο πυρήνα και ειδικότερα το κομμάτι που ασχολείται με τη μεταγωγή πακέτου θα βασίζεται εξ' ολοκλήρου στο πρωτόκολλο IP (fully IP based CN) μετά την τοποθέτηση του κόμβου RASN στη θέση των κόμβων SGSN και GGSN, αφού τα μέχρι πρότινος μηνύματα NAS (Non Access Stratum) πρόκειται να αντικατασταθούν από μηνύματα SIP. Αυτό αποτελεί ένα βήμα προς την αρχιτεκτονική All-IP (4G). Ένα δίκτυο 3^{ης} γενιάς που στηρίζεται στο IP (δηλαδή χρήση των ήδη εγκαταστημένων δρομολογητών και πρωτοκόλλων της στοίβας IP) έχει τα εξής πλεονεκτήματα:
    - Υποστήριξη νέων λειτουργιών επιπέδου IP όπως multicast (πολλαπλή διανομή) και anycast πιο οικονομικά χωρίς τη χρήση γεφυρών (bridges).
    - Η σηματοδοσία για την διαχείριση κινητικότητας, την εγκατάσταση συνόδων και για την ρύθμιση των παραμέτρων του QoS γίνεται πολύ πιο απλά και πιο εύκολα σε δίκτυα IP.
    - Παρέχεται πιο εύκολη επέκταση του δικτύου γιατί αυτό θα στηρίζεται στα ήδη υπάρχοντα σταθερά δίκτυα IP. Να σημειωθεί επίσης ότι τα δίκτυα IP δεν προϋποθέτουν κάποια πολύπλοκη αρχιτεκτονική αφού βασίζονται κυρίως στην χρήση IP δρομολογητών.
    - Παρέχεται στους παρόχους η δυνατότητα πιο οικονομικής συντήρησης ενός ομογενούς δικτύου IP λόγω του λιγότερου εξοπλισμού που χρειάζεται εν συγκρίσει με ένα ετερογενές δίκτυο όπως είναι το υπάρχον δίκτυο UMTS.
    - Παρέχεται επίσης στους παρόχους η ευκολία σύγκλισης τεχνολογιών πρόσβασης (όπως ασύρματα τοπικά δίκτυα Wi-Fi, Bluetooth) και κυψελωτών δικτύων ευρείας περιοχής.

2. Η αντικατάσταση των μηνυμάτων NAS από μηνύματα SIP έχει ως αποτέλεσμα την αποφυγή της εκτέλεσης διαδικασιών που ανήκουν σε διαφορετικά πρωτόκολλα και εξυπηρετούν τον ίδιο σκοπό όπως αναφέραμε πιο πάνω. Έτσι γίνεται εφικτή μια σημαντική μείωση του φορτίου σηματοδοσίας (μείωση αριθμού μηνυμάτων/bytes προς αποστολή).
3. Η δυνατότητα για πιο γρήγορες διαπομπές είναι εφικτή σύμφωνα με τη νέα αρχιτεκτονική. Όπως φαίνεται στο παρακάτω σχήμα (βλέπε σχήμα 13) όταν ένα κινητό τερματικό καθώς κινείται εξέρχεται από την εμβέλεια ενός κόμβου Node B που ανήκει σε ένα SGSN και εισέλθει στην εμβέλεια ενός άλλου Node B που ανήκει σε νέο SGSN τότε έχουμε διαπομπή Inter-SGSN. Για να γίνει εφικτή η διαπομπή θα πρέπει να ενημερωθεί και ο κόμβος GGSN του παρόχου, ο οποίος μπορεί να βρίσκεται αρκετά μακρυά οπότε η διαδικασία γίνεται χρονοβόρα.



Σχήμα 13: Διαδικασία σκληρής διαπομπής σε δίκτυα UMTS.

Εν αντιθέσει με τη κατάσταση αυτή, η διαπομπή στην περίπτωση που υπάρχουν μόνο οι κόμβοι RASN είναι μια διαδικασία αρκετά πιο γρήγορη γιατί δεν εμπλέκει κόμβους οι οποίοι να βρίσκονται σε μακρινή απόσταση ο ένας από τον άλλο όπως φαίνεται και από το πιο κάτω σχήμα (βλέπε σχήμα 14).



Σχήμα 14: Διαδικασία σκληρής διαπομπής σύμφωνα με την προταθείσα αρχιτεκτονική.

4. Με την απαλοιφή του κόμβου GGSN το δίκτυο γίνεται πιο αποδοτικό. Ως γνωστό, στον εν λόγω κόμβο πρέπει να συνδέονται όλοι οι κόμβοι SGSN ενός παρόχου κάτι που οδηγεί σε αργές διαπομπές όπως αναφέραμε πιο πάνω. Επίσης ο κόμβος GGSN αποτελεί τη στενωπό του συστήματος αφού όλη η κίνηση δεδομένων χρήστη θα πρέπει να δρομολογείται μέσω αυτού για να βγει στο διαδίκτυο (GGSN anchoring). Όταν όμως στο δίκτυο πυρήνα

τοποθετήσουμε το κόμβο RASN, το μονοπάτι που ακολουθούν τα δεδομένα του χρήστη είναι πιο αποδοτικό καθώς δεν συγκλίνουν σε ένα μοναδικό κόμβο (όπως τον κόμβο GGSN που μπορεί να βρίσκεται μακρυά) αλλά δρομολογούνται μέσω του κάθε RASN απευθείας στο διαδίκτυο.

5. Τέλος, με την κατάργηση των δύο κόμβων αποφεύγεται η χρήση του πρωτοκόλλου GTP στο δίκτυο πυρήνα δηλαδή η χρήση σήραγγας για μεταφορά δεδομένων που υπήρχε μεταξύ των κόμβων SGSN και GGSN. Αυτό έχει ως αποτέλεσμα την αποφυγή της πρόσθετης επιβάρυνσης λόγω επικεφαλίδας (overhead).

2.5 Διαδικασίες Mobility Management (MM) / Session Management (SM) με βάση το SIP_{RAN}/SIP

Σκοπός της διαχείρισης κινητικότητας είναι η αδιάλειπτη ενημέρωση του δικτύου για τη παρούσα θέση ενός κινητού τερματικού έτσι ώστε να είναι προσβάσιμο ανά πάσα στιγμή. Οι διαδικασίες MM υλοποιούνται με τη χρήση του μηνύματος SIP REGISTER.

Σκοπός της διαχείρισης συνόδου είναι να εξασφαλίσει στο κινητό τερματικό τη δέσμευση των απαιτούμενων καναλιών (radio bearer) για την παροχή της αιτούμενης ποιότητας υπηρεσίας μιας συνόδου. Δύναται επίσης να αρνηθεί ή να τροποποιήσει την αίτηση αν δεν υπάρχουν επαρκείς αδέσμευτοι πόροι. Οι διαδικασίες SM υλοποιούνται με τη χρήση του μηνύματος SIP INVITE.

Στην παράγραφο αυτή παραθέτουμε αρχικά τα νέα μηνύματα που θα χρησιμοποιηθούν στη θέση των διαδικασιών NAS (Attach/Detach) για την εγγραφή/διαγραφή από το δίκτυο. Στη συνέχεια αναφέρουμε τις διαδικασίες ενεργοποίησης/τερματισμού συνόδων και τα διάφορα γεγονότα που αφορούν στην κινητικότητα τα οποία εμπλέκονται στη νέα αρχιτεκτονική.

2.5.1 Διαδικασίες εγγραφής/διαγραφής

Όπως προαναφέραμε στην §2.3, οι διαδικασίες NAS πρόκειται να αντικατασταθούν από μηνύματα SIP (τα οποία με την προσθήκη NAS context μετατρέπονται σε μηνύματα SIP_{RAN}). Τα μηνύματα NAS σε αντιστοιχία με τα υποψήφια πιθανά μηνύματα SIP που πρόκειται να χρησιμοποιηθούν φαίνονται στον παρακάτω πίνακα

(βλέπε πίνακα 1). Σημειώνεται πως στον πίνακα αυτό περιλαμβάνονται μόνο τα μηνύματα τα οποία υλοποιήσαμε στην παρούσα διπλωματική εργασία.

Μηνύματα 3GPP SM/MM	Πιθανά μηνύματα SIP
GPRS Attach Procedure	**SIP Registration Procedure**
ATTACH REQUEST (up)	REGISTER
ATTACH ACCEPT (down)	200 OK
GPRS Detach Procedure	**SIP De-registration Procedure**
DETACH REQUEST (up)	REGISTER, exp.time = 0
DETACH ACCEPT (down)	200 OK
DETACH REQUEST (down)	REGISTER, exp.time = 0
DETACH ACCEPT (up)	200 OK

Πίνακας 1: Σύγκριση προτυποποιημένων διαδικασιών NAS και μηνυμάτων SIP.

2.5.2 Διαδικασίες ενεργοποίησης/τερματισμού συνόδων

Η ενεργοποίηση μιας συνόδου γίνεται με το μήνυμα SIP INVITE το οποίο αποστέλνεται από ένα κινητό τερματικό (καλών) προς μια ομότιμη οντότητα (καλούμενος). Το μήνυμα INVITE φτάνει στον κόμβο RASN και στη συνέχεια προωθείται στην περιοχή IMS όπου θα γίνουν οι απαραίτητες διαδικασίες για τη δέσμευση των πόρων που ικανοποιούν τη απαιτούμενη ποιότητα υπηρεσίας (QoS). Στη συνέχεια το μήνυμα INVITE προωθείται προς τον καλούμενο. Κατά τη διάρκεια της ενεργοποίησης συνόδου, γίνεται και η εγκατάσταση των απαραίτητων καναλιών (RAB setup) μέσω των οποίων θα μεταφέρονται τα δεδομένα χρήστη από το κινητό τερματικό στον κόμβο RASN διαμέσου ενός κόμβου RNC. Η διαδικασία εγκατάστασης των καναλιών RAB αρχικοιείται από τον κόμβο RASN.

Ο τερματισμός μιας συνόδου γίνεται με το μήνυμα SIP BYE το οποίο αποστέλνεται από οποιαδήποτε από τις δύο ομότιμες οντότητες που λαμβάνουν μέρος σε μια σύνοδο. Κατά τη διάρκεια του τερματισμού της συνόδου γίνονται οι απαραίτητες ενέργειες για την αποδέσμευση των πόρων (περιοχή IMS) και των καναλιών που είχαν δεσμευτεί κατά την ενεργοποίηση της συνόδου.

2.5.3 Διαδικασίες διαχείρισης θέσης

Οι διαδικασίες διαχείρισης θέσης περιλαμβάνουν τη διαδιακασία ενημέρωσης θέσης (Routing Area Update) καθώς και τις διαδικασίες Intra/Inter-RASN SRNC Relocation.

- Routing Area Update (RAU)

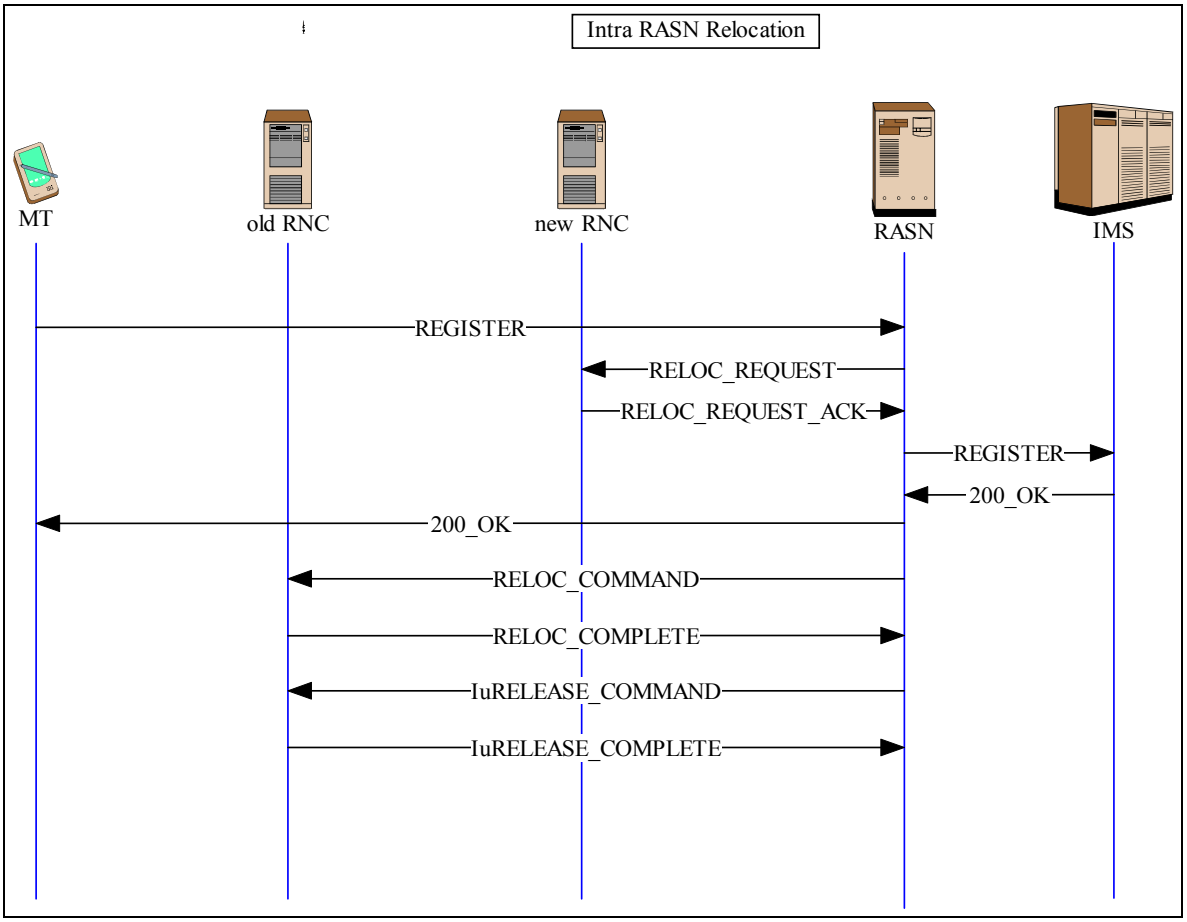
Το μήνυμα SIP REGISTER προτείνεται για την υλοποίηση της διαδικασίας RAU. Με το μήνυμα αυτό, το κινητό τερματικό ενημερώνει το δίκτυο για την αλλαγή της περιοχής δρομολόγησης του (βλέπε §1.2.1). Τα μηνύματα που ανταλλάζονται κατά τη διάρκεια της διαδικασίας RAU σε αντιστοιχία με τα υποψήφια πιθανά μηνύματα SIP που πρόκειται να χρησιμοποιηθούν φαίνονται στον παρακάτω πίνακα (βλέπε πίνακα 2).

RA Update	Πιθανά μηνύματα SIP
ROUTING AREA UPDATE REQUEST (up)	REGISTER, contact field incl. RAI
ROUTING AREA UPDATE ACCEPT (down)	200 OK
ROUTING AREA UPDATE COMPLETE (up)	INFO or ACK
ROUTING AREA UPDATE REJECT (down)	RESPONSE CODES to REGISTER

Πίνακας 2: Μηνύματα RAU με τη χρήση SIP.

- Intra-RASN SRNC Relocation

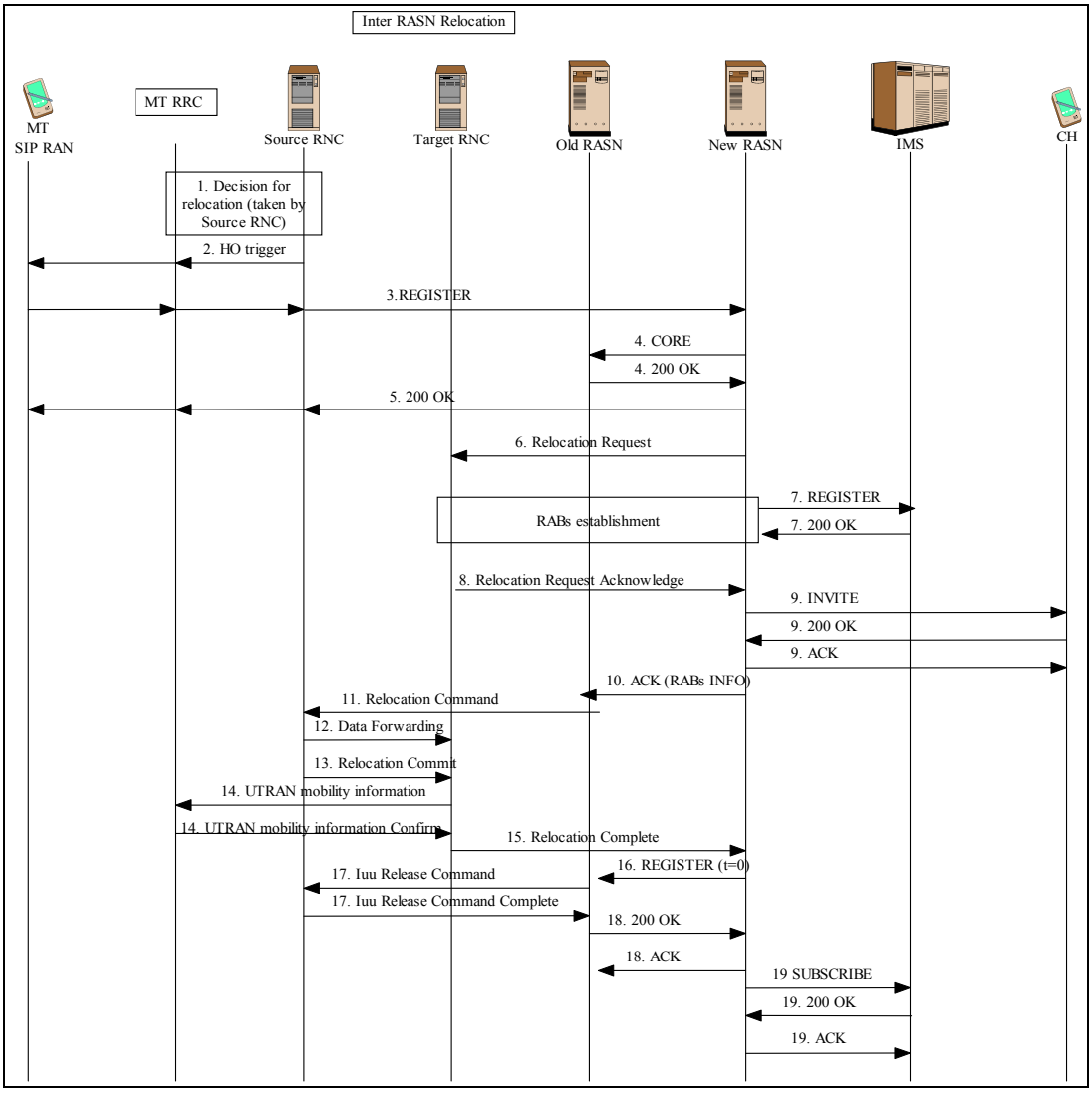
Καθώς το κινητό τερματικό μετακινείται από την περιοχή κάλυψης ενός κόμβου RNC (παλιός RNC) στην περιοχή ενός άλλου RNC (νέος RNC) στέλνει ένα μήνυμα REGISTER για να ενημερώσει το δίκτυο (κόμβος P-CSCF). Αν τόσο ο παλιός κόμβος RNC όσο και ο νέος ανήκουν στον ίδιο κόμβο RASN τότε πρόκειται για την περίπτωση Intra-RASN SRNC Relocation. Η αλληλουχία μηνυμάτων που ανταλλάζονται κατά τη διαδικασία αυτή φαίνονται στο επόμενο σχήμα. (βλέπε σχήμα 15).



Σχήμα 15: Intra-RASN Relocation.

- Inter-RASN SRNC Relocation

Αν ο παλιός κόμβος RNC ανήκει σε διαφορετικό κόμβο RASN από ότι ο νέος RNC τότε πρόκειται για την περίπτωση Inter-RASN SRNC Relocation. Η αλληλουχία μηνυμάτων που ανταλλάζονται κατά τη διαδικασία αυτή φαίνονται στο επόμενο σχήμα. (βλέπε σχήμα 16).



Σχήμα 16: Inter-RASN Relocation.

3. Εφαρμογή προταθείσας αρχιτεκτονικής

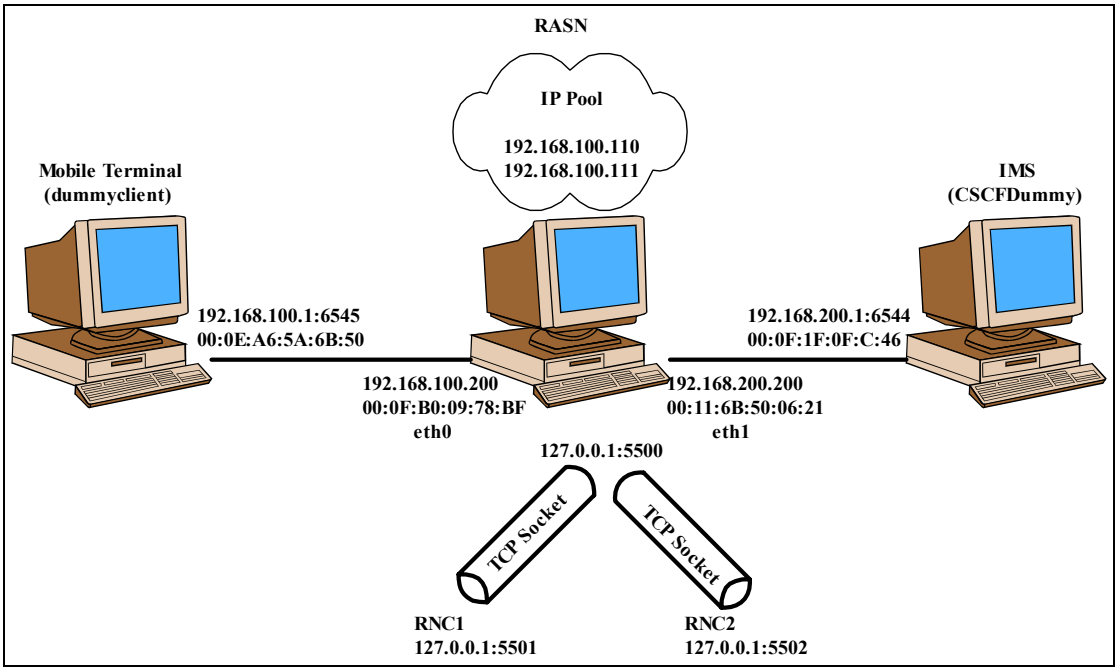
Στην παρούσα διπλωματική εργασία, όπως έχει προαναφερθεί, υλοποιήσαμε τον κόμβο RASN, ο οποίος πρόκειται να αντικαταστήσει τις οντότητες SGSN και GGSN που υπάρχουν σε ένα δίκτυο 3^{ης} γενιάς (UMTS). Ο κόμβος αυτός θα δέχεται κάποια μηνύματα SIP_{NAS} από το κινητό τερματικό (mobile terminal) από τα οποία θα αφαιρεί το NAS context δηλαδή τα δεδομένα που καθορίζουν σε ποιον RNC ανήκει το συγκεκριμένο κινητό τερματικό και θα αποστέλλει το καθαρό μήνυμα SIP προς τον κόμβο IMS ο οποίος θα τα επεξεργάζεται και θα απαντά αναλόγως. Θα εκτελείται επίσης η αντίστροφη διαδικασία από τον κόμβο RASN για τα μηνύματα που προέρχονται από τον κόμβο IMS και κατευθύνονται προς το κινητό τερματικό.

Ο κώδικας υλοποίησης του κόμβου RASN γράφτηκε στη γλώσσα προγραμματισμού C++. Τα αρχεία αυτά θα παρουσιαστούν αναλυτικά παρακάτω.

3.1 Δίκτυο Εξομοίωσης

Όπως φαίνεται από το παρακάτω σχήμα (βλέπε σχήμα 17) στο οποίο απεικονίζεται η διάταξη εξομοίωσης ενός υποτυπώδους δικτύου 3^{ης} γενιάς, ο κόμβος RASN επικοινωνεί μέσω κάρτας δικτύου Ethernet με ένα υπολογιστή ο οποίος εξομοιώνει τον κόμβο IMS, καθώς και με έναν άλλον υπολογιστή μέσω δεύτερης κάρτας δικτύου ο οποίος εξομοιώνει ένα κινητό τερματικό (mobile terminal).

Η επικοινωνία του κόμβου RASN με τους αντίστοιχους RNC κόμβους για την ανταλλαγή μηνυμάτων RANAP επιτυγχάνεται μέσω TCP Socket. Συγκεκριμένα ο κόμβος RASN λειτουργεί σαν server ο οποίος «ακούει» στη θύρα 5500 για αιτήσεις αποκατάστασης επικοινωνίας με καθένα από τους κόμβους RNC οι οποίοι παίζουν το ρόλο του client.



Σχήμα 17: Διάταξη εξομοίωσης δικτύου 3ης γενιάς.

3.2 Περιγραφή λειτουργίας κόμβου RASN

Κύριο χαρακτηριστικό του κόμβου RASN είναι ότι εκτελεί λειτουργίες δρομολόγησης. Το κινητό τερματικό (εφαρμογή dummyclient) στέλνει μηνύματα SIP_{RAN} προς το κόμβο IMS αγνοώντας τη παρουσία του κόμβου RASN. Ο κόμβος RASN εκτελεί λειτουργία sniffing συλλαμβάνοντας (capture) όσα πακέτα φτάνουν στη κάρτα δικτύου eth0 (βλέπε σχήμα 17). Η συλλογή των πακέτων από τη κάρτα δικτύου Ethernet eth0 γίνεται χρησιμοποιώντας raw socket. Στη συνέχεια ο κόμβος RASN αφού επεξεργαστεί κατάλληλα το κάθε πακέτο, το αποστέλλει στον IMS μέσω του raw socket που αντιστοιχεί στη κάρτα δικτύου eth1. Οι ίδιες διαδικασίες ακολουθούνται κατά την αποστολή πακέτων από τον IMS (εφαρμογή CSCFDummy) προς το κινητό τερματικό.

3.2.1 Συλλαμβάνοντας τα πακέτα (Capturing packets)

Όπως έχουμε ήδη αναφέρει η σύλληψη των πακέτων γίνεται με τη βοήθεια μιας ειδικής κατηγορίας socket που ονομάζονται raw socket. Χρησιμοποιώντας raw socket μπορούμε να πάρουμε τα πακέτα που καταφθάνουν στην Ethernet κάρτα με ή χωρίς να έχει αφαιρεθεί η επικεφαλίδα του επιπέδου 2. Εμείς χρησιμοποιήσαμε την πιο κάτω συνάρτηση, η οποία δημιουργεί ένα raw socket μέσω του οποίου μπορούμε να λαμβάνουμε πακέτα χωρίς την επικεφαλίδα του επιπέδου MAC.

```
sd = socket(PF_PACKET, SOCK_DGRAM, htons(ETH_P_ALL))
```

Η παραπάνω συνάρτηση καλείται μέσα από τη συνάρτηση create_raw_socket(). Αφού ρυθμίσουμε κάποιες άλλες παραμέτρους μέσω των συναρτήσεων set_if_index(), set_sockaddr_ll_values() και set_dst_mac_addr() οι οποίες ορίζονται στο αρχείο rawsockutils.cpp (βλέπε Παράρτημα Α) χρησιμοποιούμε τη συνάρτηση bind_raw_socket(). Η συνάρτηση αυτή συνδέει ένα δεδομένο raw socket με μια συγκεκριμένη κάρτα δικτύου από την οποία θα συλλαμβάνει τα εισερχόμενα πακέτα, καλώντας τη παρακάτω συνάρτηση.

```
bind(sd, (struct sockaddr *)&device, sizeof(device))
```

Η διαδικασία που αναλύθηκε πιο πάνω φαίνεται στο παρακάτω κώδικα, ο οποίος καλείται για την αρχικοποίηση των καναλιών επικοινωνίας, προτού αρχίσει η αλληλουχία ανταλλαγής πακέτων.

```
/*****************************************************************
*       Create Raw Sockets and bind them with ethernet interfaces                *
*****************************************************************/
        //creating a raw socket and binding it with eth0 interface CLIENT -> RASN
        if((rawfd0 = RawSockUtilities::Instance().create_raw_socket()) == FAIL )
                exit(0);
        if((RawSockUtilities::Instance().set_if_index("eth0", rawfd0, &ifr0)) == FAIL)
                exit(0);
        RawSockUtilities::Instance().set_sockaddr_ll_values(dev0, &ifr0);
        if((RawSockUtilities::Instance().bind_raw_socket(rawfd0, dev0, &ifr0)) == FAIL)
                exit(0);
        RawSockUtilities::Instance().set_dst_mac_addr(dev0, MT);

        //creating a raw socket and binding it with eth1 interface RASN -> IMS
        if((rawfd1 = RawSockUtilities::Instance().create_raw_socket()) == FAIL)
                exit(0);
        if((RawSockUtilities::Instance().set_if_index("eth1", rawfd1, &ifr1)) == FAIL)
                exit(0);
        RawSockUtilities::Instance().set_sockaddr_ll_values(dev1, &ifr1);
        if((RawSockUtilities::Instance().bind_raw_socket(rawfd1, dev1, &ifr1)) == FAIL)
                exit(0);
        RawSockUtilities::Instance().set_dst_mac_addr(dev1, IMS);
```

3.2.2 Επικοινωνία μέσω νημάτων

3.2.2.1 Κανάλι επικοινωνίας μεταξύ Κινητού Τερματικού και κόμβου RASN

Για να γίνει εφικτή η επικοινωνία του κόμβου RASN με το κινητό τερματικό, τον κόμβο IMS και τους κόμβους RNC υλοποιήσαμε 3 νήματα (threads). Το πρώτο νήμα ασχολείται με το raw socket που αντιστοιχεί στη κάρτα δικτύου eth0 δηλαδή με το κανάλι επικοινωνίας του κινητού τερματικού με το κόμβο RASN. Παρακάτω παρουσιάζεται ο κώδικας που εκτελεί το νήμα αυτό.

Αρχικά γίνεται έλεγχος αν το πακέτο που συλλαμβάνεται, περιέχει μήνυμα SIP στο ωφέλιμο φορτίο του UDP πακέτου. Αν το payload του πακέτου UDP δεν περιέχει

μήνυμα SIP και το πακέτο προορίζεται για τον κόμβο IMS με διεύθυνση IP 192.168.200.200 (βλέπε σχήμα 17) τότε προωθείται στο raw socket που αντιστοιχεί στη κάρτα δικτύου eth1. Με αυτόν τον τρόπο μπορούμε να μεταδίδουμε δεδομένα από το κινητό τερματικό προς τον κόμβο IMS. Ο κόμβος RASN σε αυτή την περίπτωση εκτελεί λειτουργίες δρομολόγησης επιπέδου IP, αφού ελέγχει κάθε φορά τη διεύθυνση IP του προορισμού και κρίνει σε ποιο raw socket (με άλλα λόγια σε ποια κάρτα δικτύου) θα προωθήσει το πακέτο.

```
while(1) {
        // raw socket that accepts packets from MTs and route them to IMS
        if( (fromMT = recvfrom(rawfd0, recvbuff, sizeof(recvbuff), 0, from, &len)) < 0) {
                perror("RECV\n");
                exit(0);
        }
        if(fromMT > 0) {
                ip = (struct ip_packet *)recvbuff;
                inaddr.s_addr = ip->ip_dest;
                sipmes = (char *)malloc( (fromMT-28)*sizeof(char) );
                memcpy((void *)sipmes, (void *)(recvbuff+28), fromMT-28);
                // check for SIP message
                if (strstr(sipmes, "sip") == NULL) {
                 // it is not a SIP message, so forward it to the IMS node if this is the destination of the packet
                ip = (struct ip_packet *)recvbuff;
                if(((ntohl(ip->ip_dest) & 0xffffff00) == 0xc0a8c800)) {
                        addrlen = sizeof(struct sockaddr_ll);
                        if( sendto(rawfd1, recvbuff, fromMT, 0, (struct sockaddr *) &dev1, (socklen_t)addrlen) <= 0) {
                                perror("SEND TO");
                                exit(0);
                        }
                }
                continue;
        }
```

Αν το payload του πακέτου UDP περιέχει μήνυμα SIP τότε καλείται η μέθοδος parser() (βλέπε §3.2.3), η οποία αναλαμβάνει να κάνει parsing στο μήνυμα SIP και να αποσπάσει κάποια χρήσιμα δεδομένα όπως το τύπο του μηνύματος που έχει ληφθεί (πεδίο msg), τη ταυτότητα (id) του RNC στην περιοχή του οποίου βρίσκεται το κινητό τερματικό, την ταυτότητα (id) του κινητού τερματικού που έστειλε το μήνυμα και άλλα. Τα δεδομένα αυτά τοποθετούνται σε μια μεταβλητή της δομής Sip_parsed_msg η οποία ορίζεται μέσα στο αρχείο sailorCommonTypes.h.

Με βάση το πεδίο msg της μεταβλητής αυτής γίνεται έλεγχος σχετικά με το ποιο μήνυμα SIP έχει ληφθεί και ακολούθως γίνονται οι κατάλληλες ενέργειες. Ο κόμβος RASN μπορεί να δεχθεί μόνο τα μηνύματα REGISTER και 200 OK από τον κινητό κόμβο.

```
// it is a SIP message so parse it
parser(function_info,sipmes,fromMT-28);
// pass to a function all the information needed
function_info.bytes_read = fromMT;                // number of bytes read from raw socket
```

```
function_info.rawfd = rawfd1;                    // raw socket descriptor
switch(function_info.msg) {
        case REGISTER:
                cout<<"REGISTER MESSAGE RECEIVED from MT: "<< function_info.from << endl;
                if(mt_rnc[function_info.from] == -1) {
                        // current RNC gateway of the mobile terminal
                        mt_rnc[function_info.from] = function_info.rnc_gw;
                        // new RASN tcp socket descriptor
                        function_info.new_tcpfd = rnc[mt_rnc[function_info.from]];
                        mt[function_info.from-1]->SendRegister(recvbuff, sipmes, &function_info, &dev1);
                }
                else {      // it is already registered and sends REGISTER message,
                        //having in RNC-ID_GW field the same RNC ID
                        //as the last REGISTER message => INTRA RASN ROUTING UPDATE
                        if(mt_rnc[function_info.from] == function_info.rnc_gw) {
                                cout << "INTRA RASN ROUTING UPDATE NOT IMPLEMENTED" << endl;
                                continue;
                        }
                        else {      // mobile terminal informs IMS that it has changed
                                //serving RNC node
                                // old RASN tcp socket descriptor
                                function_info.old_tcpfd = rnc[mt_rnc[function_info.from]];
                                mt_rnc[function_info.from] = function_info.rnc_gw;
                                function_info.new_tcpfd = rnc[mt_rnc[function_info.from]];
                                mt[function_info.from-1]->RelocReq(recvbuff, sipmes, &function_info, &dev1);
                        }
                }
                break;
        case _200_OK:
                cout<<"200 OK MESSAGE RECEIVED from MT: " <<function_info.from<< endl;
                mt[function_info.from-1]->Send200_OK(recvbuff, sipmes, &function_info, &dev1);
                break;
        default:
                cout << "WRONG MESSAGE RECEIVED!!!" << endl;
                break;
}
```

3.2.2.2 Κανάλι επικοινωνίας μεταξύ κόμβου RASN και κόμβου IMS

Το δεύτερο νήμα ασχολείται με το raw socket που αντιστοιχεί στη κάρτα δικτύου eth1 δηλαδή με το κανάλι επικοινωνίας του κόμβου RASN με τον κόμβο IMS. Το νήμα αυτό δημιουργείται από το κύριο νήμα (πρώτο νήμα) μέσω της παρακάτω συνάρτησης.

```
// thread that handles incoming messages from IMS to RASN (raw socket)
pthread_create(&tid1, NULL, handle_ims, &thread_info);
```

Η συνάρτηση pthread_create(), που ανήκει στη βιβλιοθήκη των POSIX threads, δημιουργεί το νέο νήμα στο οποίο αναθέτει την εκτέλεση της συνάρτησης handle_ims().

Οι λειτουργίες που αποδίδονται στο νήμα αυτό είναι σχεδόν οι ίδιες με αυτές που εκτελεί το πρώτο νήμα με τη διαφορά όμως ότι αναμένονται διαφορετικά μηνύματα SIP από τον κόμβο IMS προς το κινητό τερματικό τα οποία είναι το INVITE και το 200 OK.

```
void *handle_ims(void *infoptr)
{
```

```
        char recvbuff[BUFFSIZE];
        struct sockaddr *from;
        char *sipmes;
        socklen_t len,addrlen;
        struct ip_packet *ip;
        ssize_t fromIMS;
        struct info *iptr;      // struct that contains the required information (i.e socket descriptors)
        Sip_parsed_msg function_info;

        iptr = (struct info *)infoptr;

        while(1) {
                // raw socket that accepts packets from MTs and route them to IMS
                if( (fromIMS = recvfrom(iptr->rawfd_ims, recvbuff, sizeof(recvbuff), 0, from, &len)) < 0) {
                        perror("RECV\n");
                        exit(0);
                }
                 if(fromIMS > 0) {
                        sipmes = (char *)malloc( (fromIMS-28)*sizeof(char) );
                        memcpy((void *)sipmes, (void *)(recvbuff+28), fromIMS-28);
                         if (strstr(sipmes, "sip") == NULL) {
                                // it is not a SIP message, so forward it to the desirable destination if available
                                ip = (struct ip_packet *)recvbuff;
                                if((ntohl(ip->ip_dest) & 0xffffff00) == 0xc0a86400) {
                                        addrlen = sizeof(struct sockaddr_ll);
                if( sendto(iptr->rawfd_mt, recvbuff, fromIMS, 0, (struct sockaddr *) iptr->dev0, (socklen_t)addrlen) <= 0)
                                        {
                                                perror("SEND TO");
                                                exit(0);
                                        }
                                }
                                continue;
                        }
                        // it is a SIP message so parse it
                        parser(function_info,sipmes,fromIMS-28);
                                // pass to a function all the information needed
                                function_info.bytes_read = fromIMS;     // number of bytes read from raw socket
                                function_info.rawfd = iptr->rawfd_mt;   // raw socket descriptor
                                // RNC gateway of the mobile terminal
                                function_info.rnc_gw = mt_rnc[function_info.from];
                                switch(function_info.msg) {
                                        case _200_OK:
                                                cout << "200 OK MESSAGE RECEIVED from IMS"<< endl;
                                mt[function_info.from-1]->Send200_OK(recvbuff, sipmes, &function_info, iptr->dev0);
                                                break;
                                        case INVITE:
                                                cout << "INVITE MESSAGE RECEIVED from IMS" << endl;
                                                function_info.new_tcpfd = rnc[mt_rnc[function_info.from]];
                                                // used while forwarding INVITE message (for NAS context)
                                                function_info.rnc_gw = mt_rnc[function_info.from];
                                  mt[function_info.from-1]->RABAssReq(recvbuff, sipmes, &function_info, iptr->dev0);
                                                break;
                                        default:
                                                cout << "WRONG MESSAGE RECEIVED!!!" << endl;
                                                break;
                }

              }
        }
        return (NULL);
}
```

3.2.2.3 Κανάλι επικοινωνίας μεταξύ κόμβου RASN και κόμβων RNC

Το τρίτο νήμα ασχολείται με το TCP socket στο οποίο ο RASN ακούει αιτήσεις αποκατάστασης επικοινωνίας καθώς και μηνύματα RANAP από τους κόμβους RNC. Το νήμα αυτό δημιουργείται από το κύριο νήμα (πρώτο νήμα) μέσω της παρακάτω συνάρτησης.

```
// thread that handles TCP connections from RNCs to RASN
pthread_create(&tid0, NULL, connect_rnc, &thread_info);
```

Η συνάρτηση pthread_create() δημιουργεί το νέο νήμα στο οποίο αναθέτει την εκτέλεση της συνάρτησης connect_rnc().

Ο κόμβος RASN που παίζει το ρόλο του server στην επικοινωνία με τους κόμβους RNC, δέχεται στη θύρα 5500 αιτήσεις από τους RNCs για αποκατάσταση επικοινωνίας με σκοπό την ανταλλαγή μηνυμάτων RANAP. Όταν επιτευχθεί επικοινωνία με κάποιο RNC, τότε αποθηκεύεται ο socket descriptor έτσι ώστε μέσω του νέου TCP socket descriptor να ανταλλάζονται τα μηνύματα RANAP. Τα μηνύματα RANAP που είναι δυνατό να λάβει ο κόμβος RASN από ένα RNC είναι το RELOCATION_REQUEST_ACK, το RELOCATION_COMPLETE, το Iu_RELEASE_COMPLETE και το RAB_ASSIGNMENT_RESPONSE.

```
void *connect_rnc(void *arg)
{
	int		i, maxi, maxfd, listenfd, connfd, sockfd;
	int		nready;
	ssize_t	n;
	fd_set	rset, allset;
	char	buff[MAXLINE];
	socklen_t	rnclen;
	struct sockaddr_in rncaddr, rasnaddr;
	if( (listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
		perror("SOCKET\n");
		exit(1);
	}

	int enable = 1;
	setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int));
	bzero(&rasnaddr, sizeof(rasnaddr));
	rasnaddr.sin_family	= AF_INET;
	rasnaddr.sin_addr.s_addr = htonl(INADDR_ANY);
	rasnaddr.sin_port	= htons(RNC_RASN_PORT);

	if ( (bind(listenfd, (sockaddr *) &rasnaddr, sizeof(rasnaddr))) < 0) {
		perror("BIND\n");
		exit(1);
	}

	listen(listenfd, LISTENQ);

	maxfd = listenfd;			// initialize
	maxi = -1;				// index into rnc[] array
	for (i = 0; i < N_RNC_GWS; i++)
		rnc[i] = -1;			// -1 indicates available entry
	FD_ZERO(&allset);
	FD_SET(listenfd, &allset);
	while (1) {
		rset = allset;			// structure assignment
		// select waits for something to happen (establishment of new rnc connection or the arrival of data)
		nready = select(maxfd+1, &rset, NULL, NULL, NULL); // returns positive count of ready discriptors

		if (FD_ISSET(listenfd, &rset)) {  // new RNC connection
			rnclen = sizeof(rncaddr);
			connfd = accept(listenfd, (sockaddr *) &rncaddr, &rnclen);

			printf("RNC %d from: %s:%d CONNECTED\n", ntohs(rncaddr.sin_port) - 5500,
```

```
                                        inet_ntop(AF_INET, &rncaddr.sin_addr, buff, sizeof(buff)),
                                        ntohs(rncaddr.sin_port));

                        rnc[ntohs(rncaddr.sin_port)-5500] = connfd; // save descriptor
                        FD_SET(connfd, &allset);        // add new descriptor to set
                        if (connfd > maxfd)
                                maxfd = connfd;         // for select
                        if (--nready <= 0)
                                continue;               // no more readable descriptors
                }

                for (i = 0; i <= N_RNC_GWS; i++) {      // check all RNCs for RANAP messages
                        if ( (sockfd = rnc[i]) < 0)
                                continue;
                        if (FD_ISSET(sockfd, &rset)) {
                                if ( (n = read(sockfd, buff, MAXLINE)) == 0) {
                                        // connection closed by RNC
                                        close(sockfd);
                                        FD_CLR(sockfd, &allset);
                                        rnc[i] = -1;
                                        printf("CONNECTION LOST from RNC %d\n",i);
                                }
                                else {  // all cases are inside this else
                                        int nbytes, index=0;
                                        int msg_type;
                                        memcpy((char *)&msg_type, buff, sizeof(msg_type));
                                        switch(msg_type) {
                                                case RELOCATION_REQUEST_ACK:
                                        cout << "RELOCATION REQUEST ACK MESSAGE RECEIVED" << endl;
                                        struct Reloc_Req_Ack reloc_req_ack;
                                        memcpy((char *)&reloc_req_ack, buff, sizeof(reloc_req_ack));
                                        mt[reloc_req_ack.mtID-1]->SendRegister(NULL, NULL, NULL, NULL);
                                        continue;
                                        break;
                                                case RELOCATION_COMPLETE:
                                        cout << "RELOCATION COMPLETE MESSAGE RECEIVED" << endl;
                                        struct Reloc_Complete reloc_complete;
                                        memcpy((char *)&reloc_complete, buff, sizeof(reloc_complete));
                                        index += sizeof(reloc_complete);
                                        struct Iu_Release_Command iu_release_command;
                                        iu_release_command.msgID = Iu_RELEASE_COMMAND;
                                        iu_release_command.mtID = reloc_complete.mtID;
                                       memcpy((char *)buff, (char *)&iu_release_command, sizeof(iu_release_command));
                                        if ( (nbytes=write(sockfd, buff, index)) != index )
                                        {
                                                perror("SEND");
                                        }
                                        //printf("SENT REPLY %d BYTES\n", nbytes);
                                        break;
                                                case Iu_RELEASE_COMPLETE:
                                        cout << "Iu RELEASE COMPLETE MESSAGE RECEIVED" << endl;
                                        // do nothing at all
                                        break;
                                                case RAB_ASSIGNMENT_RESPONSE:
                                        cout << "RAB ASSIGNMENT RESPONSE MESSAGE RECEIVED" << endl;
                                        struct RAB_Assign_Resp rab_assign_resp;
                                        memcpy((char *)&rab_assign_resp, buff, sizeof(rab_assign_resp));
                                        mt[rab_assign_resp.mtID-1]->SendInvite(NULL, NULL, NULL, NULL);
                                        continue;
                                        break;
                                                default:
                                        cout << "WRONG MESSAGE RECEIVED" << endl;
                                        break;
                                }
                        }
                        if (--nready <= 0)
                                break;  // no more readable descriptors
                        }
                    }
                }
                return (NULL);
}
```

3.2.3 Αναλύοντας το μήνυμα SIP (Parsing the SIP Message)

Μια σημαντική λειτουργία που επιτελεί ο κόμβος RASN είναι η ανάλυση ενός μηνύματος SIP όταν αυτό συλληφθεί από την κάρτα δικτύου. Η κυριότερη ενέργεια που εκτελείται από τη συνάρτηση parser() (η οποία ορίζεται στο αρχείο parser.cpp που παρουσιάζεται πιο κάτω) είναι η μετατροπή των μηνυμάτων SIP_{NAS} που αποστέλλονται από το κινητό τερματικό σε καθαρά μηνύματα SIP έτσι ώστε να μπορεί να τα επεξεργαστεί ο κόμβος IMS. Στην ουσία το NAS context που αφαιρείται, είναι το id του κόμβου RNC στου οποίου την περιοχή βρίσκεται το κινητό τερματικό. Η συνάρτηση αυτή όπως είχαμε προαναφέρει, αποσπά κάποια χρήσιμα δεδομένα από το μήνυμα SIP τα οποία τοποθετεί σε μια μεταβλητή της δομής Sip_parsed_msg.

```
/***************************************************************************
                        parser.cpp  -  description
                        -------------------
   begin                : Sat Apr 16 2005
   copyright            : (C) 2005 by tkampanakis, pantoniou, mkallitsis
   email                : {el00129, el00600, el00602}@mail.ntua.gr
 ***************************************************************************/

#include <iostream>
#include <string.h>
#include "sailorCommonTypes.h"

using namespace std;
using namespace SailorTypes;

// function that parses the SIP message
void parser(Sip_parsed_msg &out,char *buff,int len)
{
        string st;
        st.assign(buff);
        string tmpStr;
        string st2=st;

        // The RNC line is deleted from the sip message if it is present and the results are stored in st2
        if ( st2.find("RNC_GW-ID: ", 0 ) != string::npos )
        st2.erase(st2.find("RNC_GW-ID: ", 0),st2.find("Via: ", 0)-st2.find("RNC_GW-ID: ", 0));
        // erase all useless characters at the end of the string
        st2 = st2.substr(0,st2.find_last_of(":")+5);

        // cout<<"------- WITHOUT SIPRANAP -----------"<<endl;
        // cout<<st2<<endl;
        // Erase the SIP-RAN message that is stored in buff
        for(int i=0; i<len; i++) buff[i] = '\0';
        // and copy the new pure SIP message in buff
        memcpy((void *)buff,(void *)st2.c_str(),st2.length());
        out.length = st2.length();
        // cout<<"-------------------------------------"<<endl;

        // The method is stored in msg field
        tmpStr=st.substr(0,st.find(' ',2));
        if(tmpStr == "REGISTER")                out.msg = REGISTER;
        else if(tmpStr == "INVITE")             out.msg = INVITE;
        else if(st.substr(8,3) == "200")        out.msg = _200_OK;

        // The message is parsed and all the needed fields are collected in a dedicated special structure
        // that is defined in sailorCommonTypes.h
        while (1) {
                tmpStr=st.substr(0,st.find('\n',0));
                if ( tmpStr.find("Contact: ", 0 ) != string::npos ) {
                        out.contact = true;
```

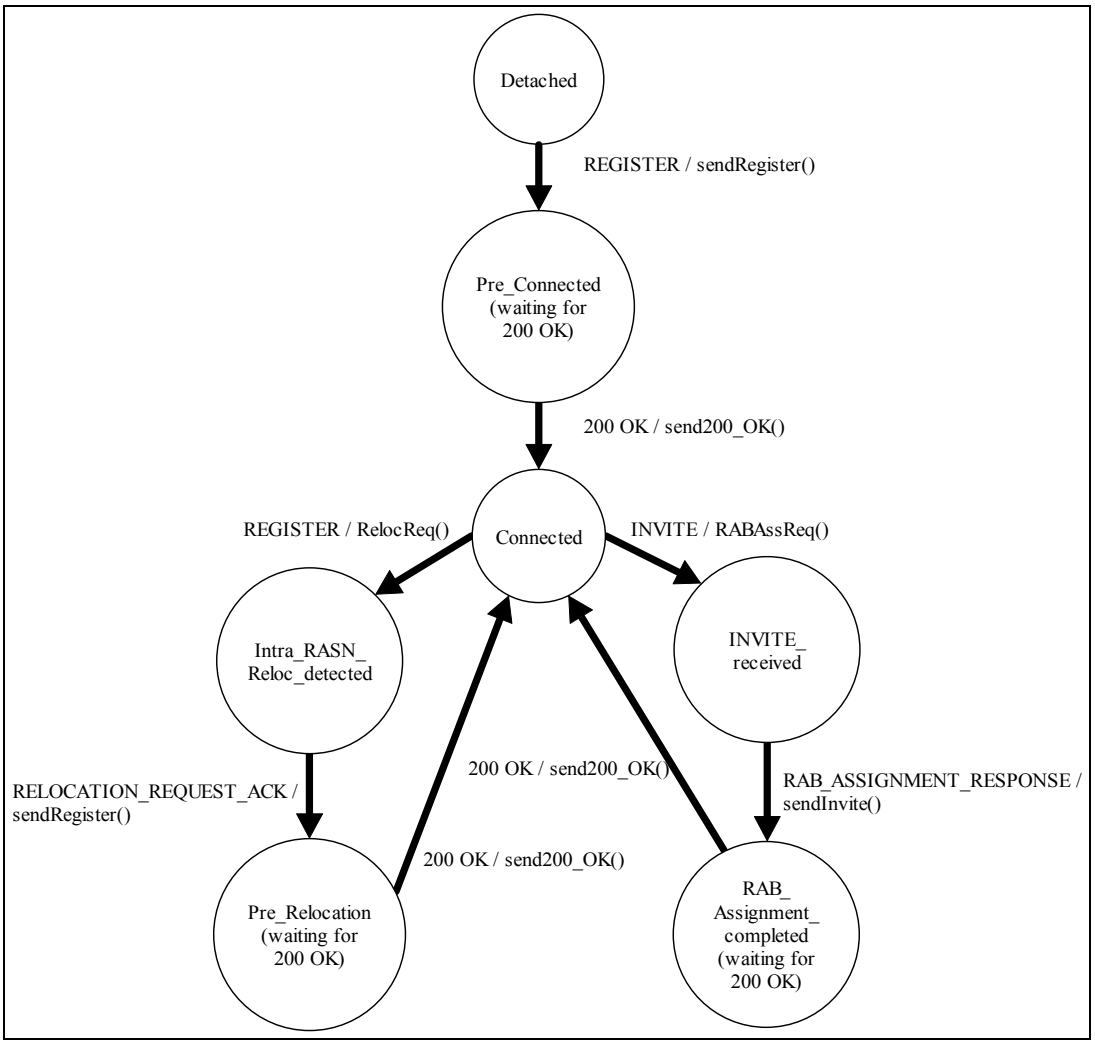
```
                                tmpStr=tmpStr.substr(tmpStr.find("Contact: ",0)+9,tmpStr.length()-tmpStr.find("Contact: ",0)-9);
                                string tmpStr2 = tmpStr;
                                int position = tmpStr2.find(" ");
                                while(position!=string::npos) {
                                        tmpStr2.replace(position, 1, ".");
                                        position = tmpStr2.find(" ", position + 1);
                                }
        if(((tmpStr2.find_last_of(".")-tmpStr2.find_first_of(".")+1) == (tmpStr.length()-1)) || (tmpStr.length() == 1))
                                        out.contact = false;

                        }
                        else if ( tmpStr.find("To: ", 0 ) != string::npos ) {
                                tmpStr=tmpStr.substr(tmpStr.find("To: ",0)+4,tmpStr.find(';',0)-4);
                        }
                        else if ( tmpStr.find("From: ", 0 ) != string::npos ) {
                                tmpStr=tmpStr.substr(tmpStr.find("From: ",0)+6, tmpStr.find(';',0)-6);
                                tmpStr=tmpStr.substr(tmpStr.find("<",0)+1, tmpStr.find(">",0) - tmpStr.find("<",0)-1);
                                if(tmpStr == SIP_UA_1)
                                        out.from = 1;
                                else if(tmpStr == SIP_UA_2)
                                        out.from = 2;
                        }
                        else if ( tmpStr.find("Call-ID: ", 0 ) != string::npos ) {
                                tmpStr=tmpStr.substr(tmpStr.find("Call-ID: ",0)+9,tmpStr.length()-tmpStr.find("Call-ID: ",0)-9);
                        }
                        else if ( tmpStr.find("CSeq: ", 0 ) != string::npos ) {
                                tmpStr=tmpStr.substr(tmpStr.find("CSeq: ",0)+6,tmpStr.length()-tmpStr.find("CSeq: ",0)-6);
                                  tmpStr=tmpStr.substr(tmpStr.find(' ',0)+1,tmpStr.length()-tmpStr.find(' ',0)-2);
                                  // return the message type that is contained in CSeq field
                                  if(tmpStr == "REGISTER")        out.cseq = REGISTER;
                                  else if(tmpStr == "INVITE")     out.cseq = INVITE;
                        }
                        else if ( tmpStr.find("RNC_GW-ID: ", 0 ) != string::npos ) {
        tmpStr=tmpStr.substr(tmpStr.find("RNC_GW-ID: ",0)+11,tmpStr.length()-tmpStr.find("RNC_GW-ID: ",0)-11);
                                  // return the id of the RNC gateway
                                  out.rnc_gw = atoi(tmpStr.c_str());
                        }
                        if (st.find('\n',0)==string::npos) break;
                                else st=st.substr(st.find('\n',0)+1,st.length());
                        }

}
```

3.2.4 Διάγραμμα Καταστάσεων κόμβου RASN

Το διάγραμμα καταστάσεων του κόμβου RASN φαίνεται στο πιο κάτω σχήμα (βλέπε σχήμα 18). Οι κύκλοι περιγράφουν τις διάφορες καταστάσεις που μπορεί να βρεθεί ο κόμβος RASN. Σημειωτέον ότι ο κόμβος RASN διατηρεί μια ξεχωριστή μηχανή καταστάσεων για κάθε ένα από τα κινητά τερματικά τα οποία βρίσκονται στην περιοχή του. Με τον τρόπο αυτό μπορούμε να προσδιορίσουμε ποια ενέργεια θα εκτελεστεί από τον κόμβο RASN με την έλευση κάποιου μηνύματος SIP ή RANAP που φτάνει από κάποιο συγκεκριμένο κινητό τερματικό. Για μεταφερθούμε από μια κατάσταση σε μια άλλη, πρέπει να προηγηθεί ένα συμβάν (event) καθώς και να εκτελεστεί μια αντίστοιχη ενέργεια διαφορετική για κάθε συμβάν. Το συμβάν και η ενέργεια που ακολουθεί το συμβάν παρουσιάζονται στο πιο κάτω σχήμα δίπλα από το αντίστοιχο βέλος, στη μορφή συμβάν/ενέργεια. Η σχεδίαση των καταστάσεων έγινε με βάση τη τεχνική των design patterns [1] και ο εν λόγω κώδικας βρίσκεται στα αρχεία patterns.h και patterns.cpp.



Σχήμα 18: Διάγραμμα καταστάσεων κόμβου RASN.

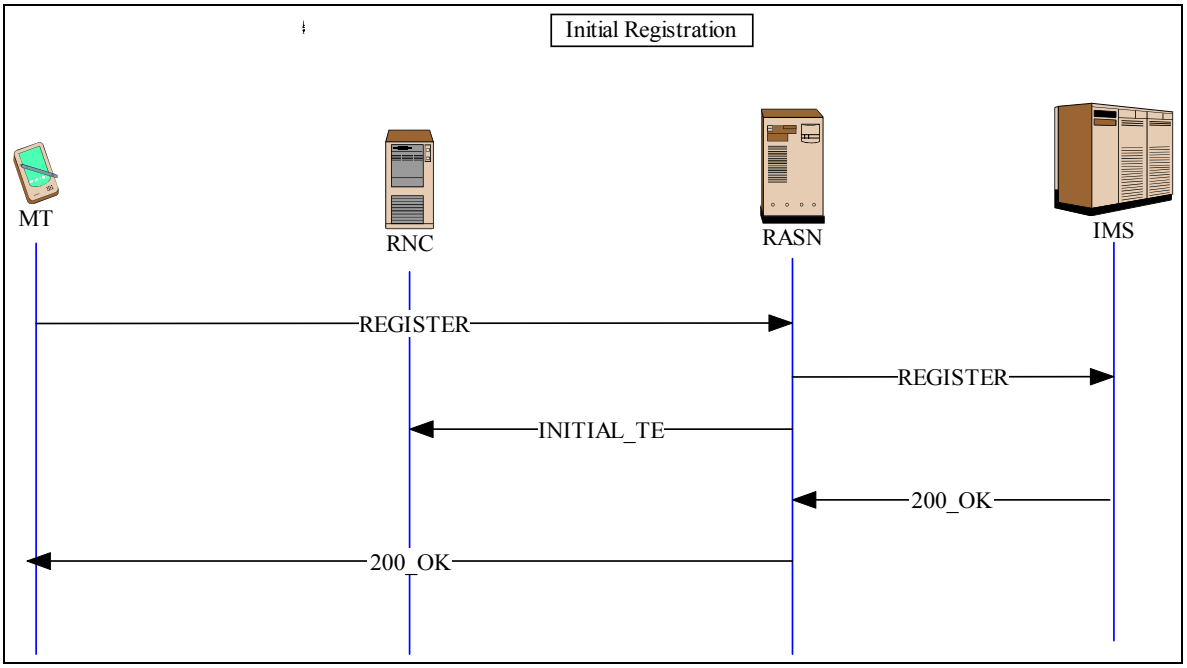
Οι διάφορες καταστάσεις που προσδιορίζονται στο σχήμα 18 θα αναλυθούν πιο κάτω με ενδελεχή αναφορά στα μηνύματα SIP και RANAP που φτάνουν στον κόμβο RASN καθώς και στις ενέργειες που λαμβάνονται άμα τη αφίξη των εν λόγω μηνυμάτων.

Στο διάγραμμα καταστάσεων παρουσιάζονται τρία διαφορετικά σενάρια:

- αρχική εγγραφή του κινητού τερματικού στο δίκτυο (Initial Registration),
- σενάριο μεταγωγής από τον παλιό RNC κόμβο με τον οποίο επικοινωνεί ένα κινητό τερματικό σε ένα νέο RNC κόμβο, καθώς αυτό μετακινείται, παραμένοντας όμως στην περιοχή εξυπηρέτησης του ίδιου κόμβου RASN (Intra – RASN Relocation),
- εγκαθίδρυση media session (μήνυμα INVITE).

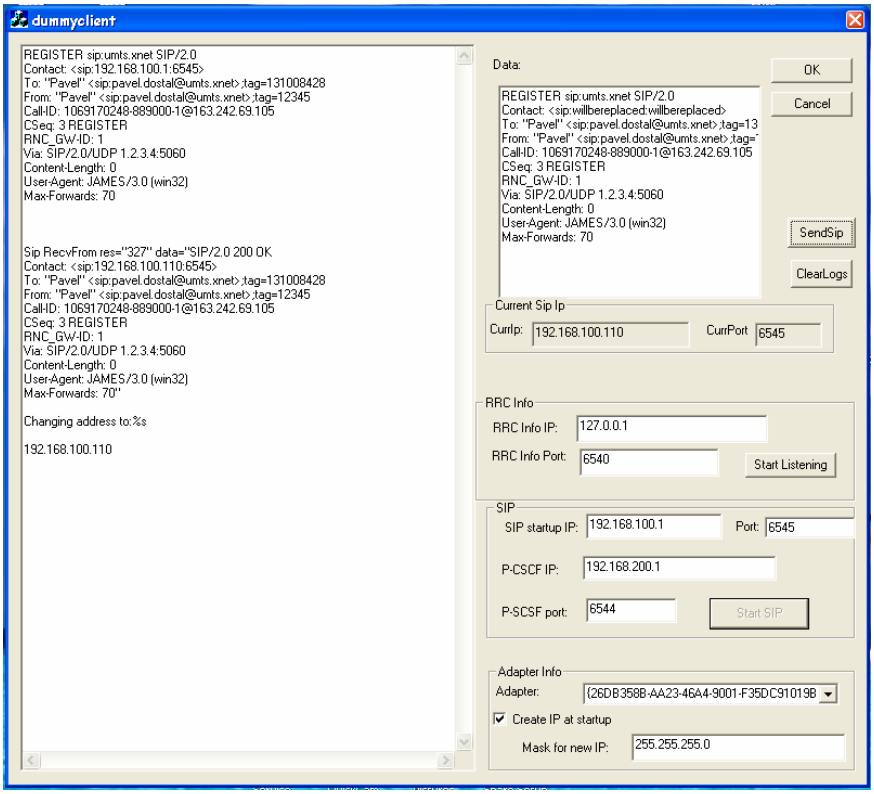
3.2.4.1 Αρχική Εγγραφή κινητού τερματικού (Initial Registration)

Αρχικά ένα κινητό τερματικό είναι απενεργοποιημένο οπότε η κατάσταση του κόμβου RASN για το εν λόγω τερματικό είναι η Detached. Όταν θέλουμε να ενεργοποιήσουμε το κινητό τερματικό, στέλνουμε μήνυμα REGISTER μέσω της εφαρμογής dummyclient με προορισμό τον κόμβο IMS. Ο κόμβος RASN που παρεμβάλλεται στη διαδρομή, λαμβάνει το μήνυμα αυτό και αφαιρεί το NAS context. Ακολούθως δεσμεύει μια νέα διεύθυνση IP από το IP pool – το οποίο υλοποιείται σαν ένας πίνακας από διαθέσιμες διευθύνσεις (βλέπε αρχείο patterns.cpp) – και την τοποθετεί στο πεδίο Contact καθώς και στο πεδίο της διεύθυνσης προορισμού της επικεφαλίδας IP. Το νέο αυτό πακέτο προωθείται προς τον κόμβο IMS μέσω του raw socket που αντιστοιχεί στην κάρτα δικτύου eth1 και ταυτοχρόνως στέλνει μήνυμα INITIAL_TE στον κόμβο RNC που εξυπηρετεί το κινητό τερματικό, μέσω του TCP socket που αντιστοιχεί στην επικοινωνία με τον συγκεκριμένο κόμβο RNC. Η κατάσταση του κόμβου RASN για το συγκεκριμένο τερματικό μεταβαίνει στη Pre_Connected. Ο κόμβος IMS, δηλαδή η εφαρμογή CSCFDummy, λαμβάνοντας το καθαρό μήνυμα SIP REGISTER αποκρίνεται με το μήνυμα 200 OK. Ο κόμβος RASN λαμβάνοντας το μήνυμα αυτό από τον IMS τοποθετεί το NAS context παράγοντας το μήνυμα $SIP_{NAS.}$ Επίσης τοποθετεί στο πεδίο της διεύθυνσης προορισμού της επικεφαλίδας IP του εν λόγω πακέτου, τη διεύθυνση που είχε αρχικά το τερματικό έτσι ώστε να δρομολογηθεί ορθά το πακέτο. Το πακέτο αυτό αποστέλλεται προς το κινητό τερματικό μέσω του raw socket που αντιστοιχεί στην κάρτα δικτύου eth0 και ταυτοχρόνως η κατάσταση του κόμβου RASN για το συγκεκριμένο τερματικό μεταβαίνει στη Connected. Το κινητό τερματικό – δηλαδή η εφαρμογή dummyclient – λαμβάνοντας το μήνυμα 200 OK, διαβάζει τη διεύθυνση IP που βρίσκεται στο πεδίο Contact του μηνύματος 200 OK την οποία θα χρησιμοποιεί από εδώ και πέρα σαν διεύθυνση του κινητού τερματικού. Η αλληλουχία μηνυμάτων που έχει περιγραφεί φαίνεται στο πιο κάτω σχήμα (βλέπε σχήμα 19).

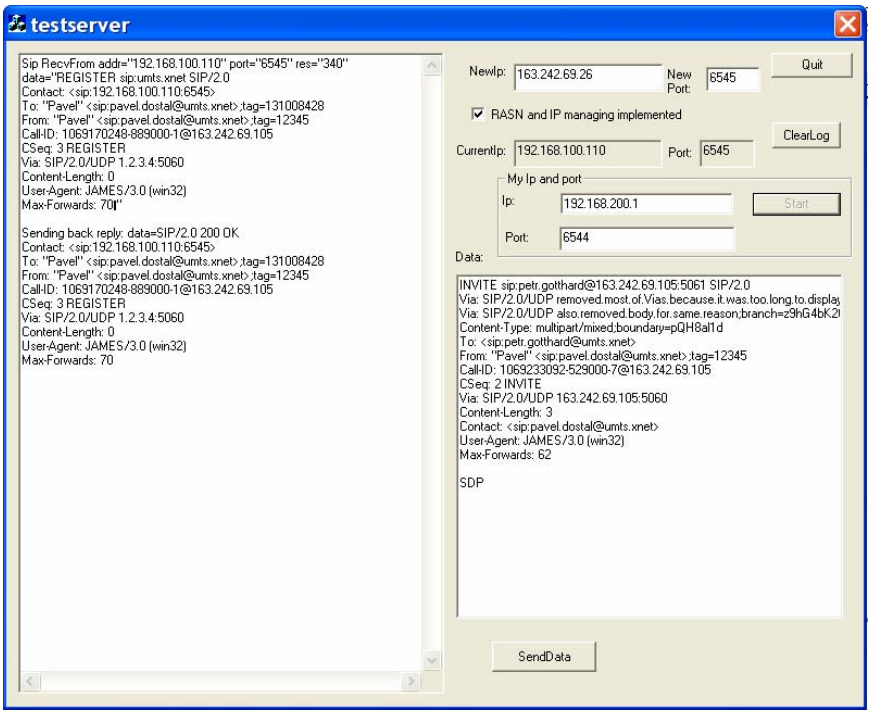


Σχήμα 19: Αρχική εγγραφή (Initial Registration)**.**

Το στιγμιότυπο της εφαρμογής dummyclient (κινητό τερματικό) που φαίνεται στο σχήμα 20 καθώς και το στιγμιότυπο της εφαρμογής CSCFDummy (κόμβος IMS) που φαίνεται στο σχήμα 21, απεικονίζουν το σενάριο που μόλις περιγράψαμε.



Σχήμα 20: Αρχική εγγραφή (εφαρμογή dummyclient).



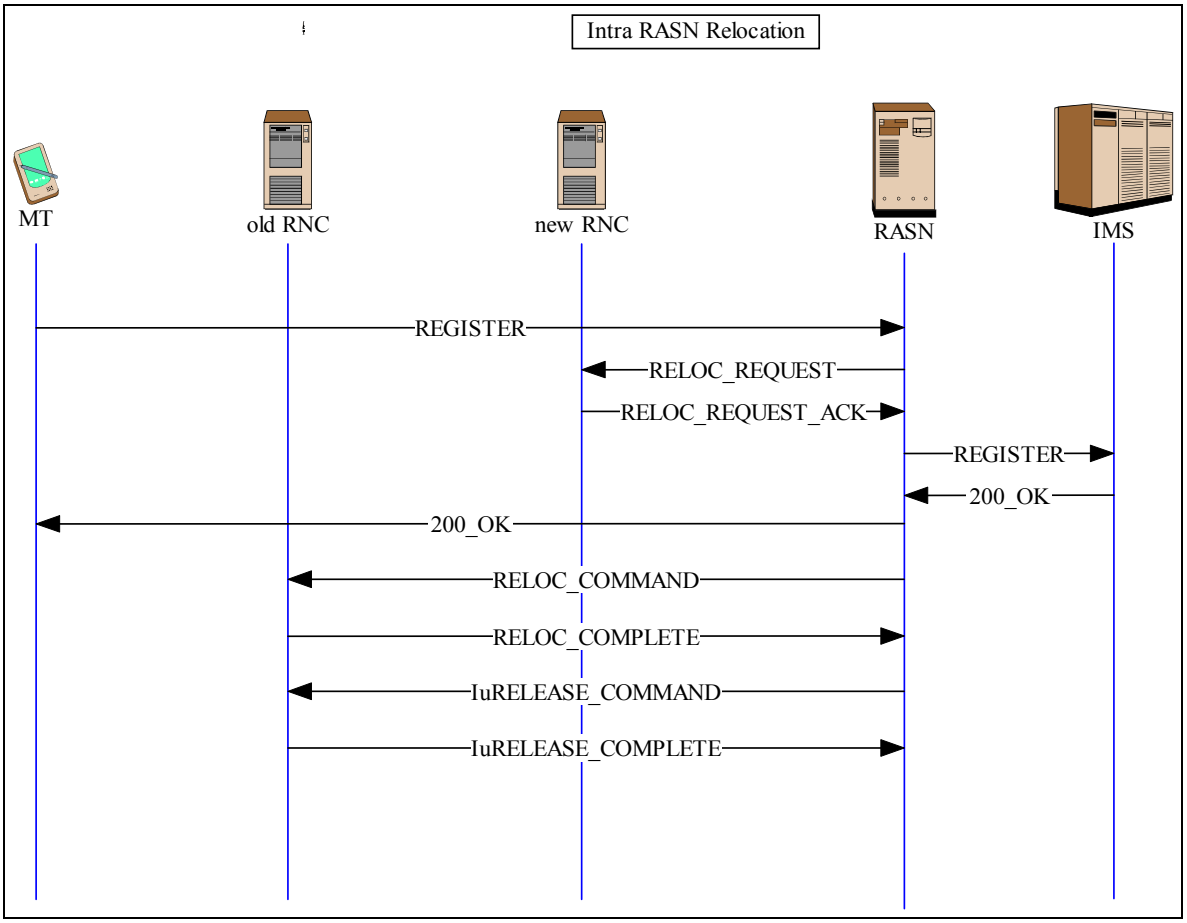
Σχήμα 21: Αρχική εγγραφή (εφαρμογή CSCFDummy).

3.2.4.2 Intra – RASN Relocation

Όταν ο κόμβος RASN βρίσκεται στην κατάσταση Connected μπορούν να συμβούν δύο τινά. Η πρώτη περίπτωση είναι να λάβει μήνυμα REGISTER από το κινητό τερματικό την οποία εξετάζουμε στην παράγραφο αυτή. Η άλλη περίπτωση που αφορά στη λήψη μηνύματος INVITE από τον κόμβο IMS, εξετάζεται στην επόμενη παράγραφο.

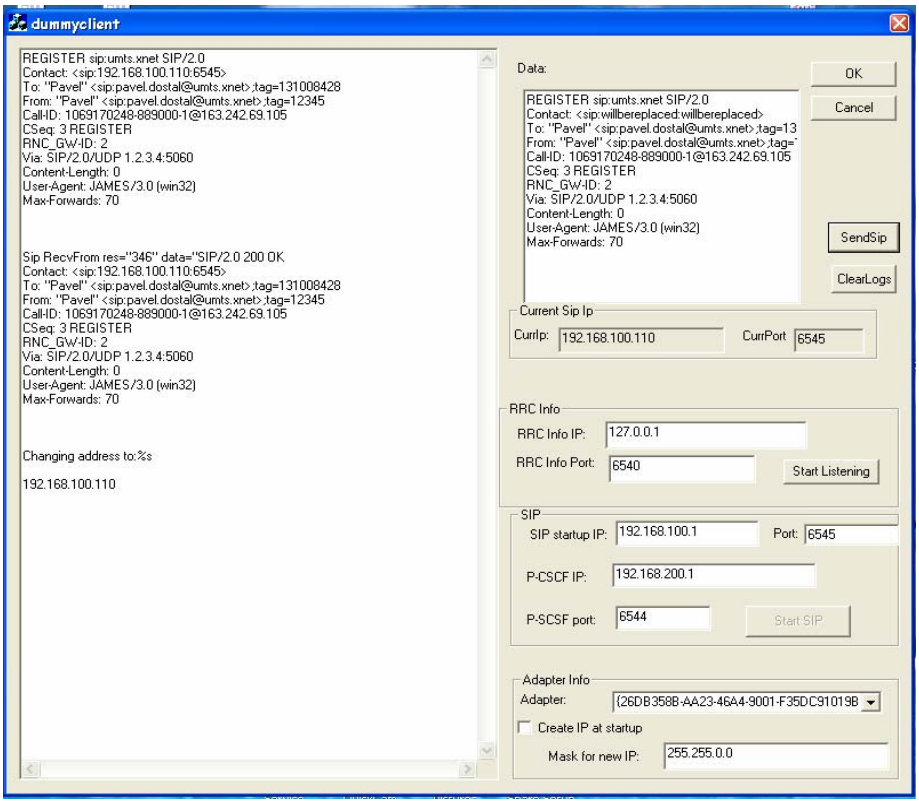
Όταν ο κόμβος RASN λάβει μήνυμα REGISTER από το κινητό τερματικό (εφαρμογή dummyclient) ευρισκόμενος στην κατάσταση Connected, στέλνει μήνυμα RELOC_REQUEST προς τον κόμβο RNC στου οποίου την περιοχή έχει μόλις εισέλθει το τερματικό. Το μήνυμα RELOC_REQUEST – όπως και όλα τα μηνύματα RANAP – υλοποιείται σαν μια δομή που ορίζεται μέσα στο αρχείο sailorCommonTypes.h. Στην περίπτωση αυτή ο κόμβος RASN μεταβαίνει στην κατάσταση Intra_RASN_Reloc_detected. Μόλις ο κόμβος RNC λάβει το πιο πάνω μήνυμα αποκρίνεται με το μήνυμα RELOC_REQUEST_ACK. Λαμβάνοντας το μήνυμα αυτό ο RASN, προωθεί το μήνυμα REGISTER (που είχε λάβει προηγουμένως από το κινητό τερματικό) στον κόμβο IMS αφαιρώντας όπως πάντα το NAS context και μεταβαίνει στην κατάσταση Pre_Relocation. Λαμβάνοντας η εφαρμογή CSCFDummy του κόμβου IMS το μήνυμα REGISTER, αποκρίνεται με το μήνυμα 200 OK. Ο κόμβος RASN λαμβάνει το μήνυμα αυτό και το προωθεί στο

κινητό τερματικό προσθέτοντας το NAS context έτσι το καθαρό μήνυμα SIP να μετατραπεί σε μήνυμα SIP_{NAS}. Ακολουθεί μια ανταλλαγή μηνυμάτων RANAP μεταξύ του κόμβου RASN και του παλιού κόμβου RNC από του οποίου την εμβέλεια έχει βγει το τερματικό, που έχουν ως αποτέλεσμα την αποδέσμευση των πόρων που είχαν δεσμευτεί για την εξυπηρέτηση του κινητού τερματικού. Τα μηνύματα που ανταλλάζονται φαίνονται στο πιο κάτω σχήμα (βλέπε σχήμα 22). Ο κόμβος RASN μεταβαίνει ξανά στην κατάσταση Connected.

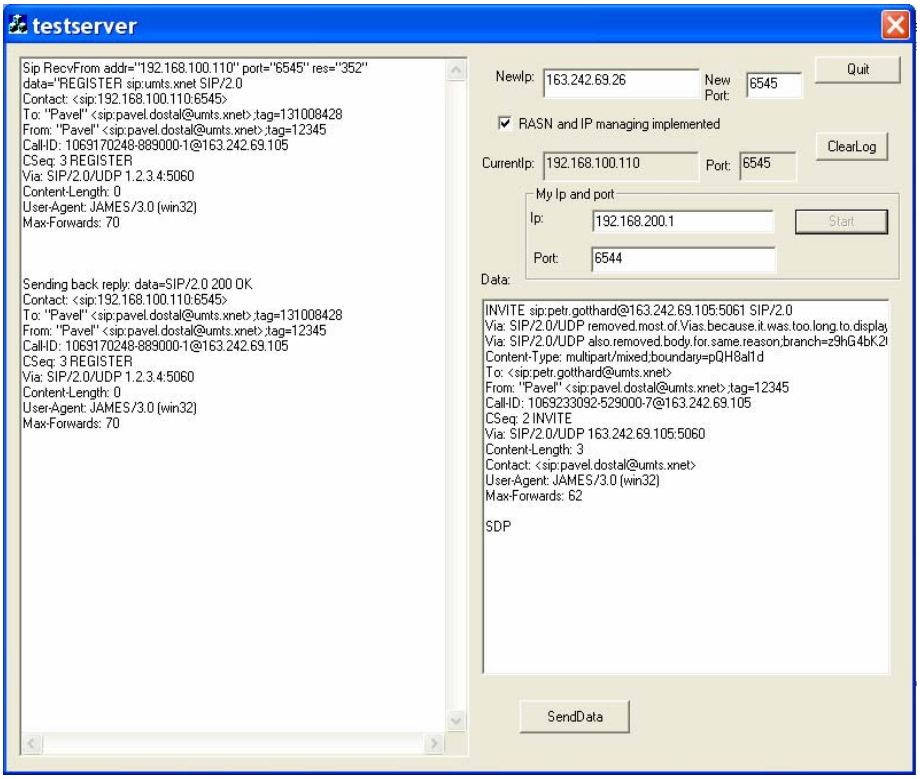


Σχήμα 22: Intra-RASN Relocation.

Το στιγμιότυπο της εφαρμογής dummyclient (κινητό τερματικό) που φαίνεται στο σχήμα 23 καθώς και το στιγμιότυπο της εφαρμογής CSCFDummy (κόμβος IMS) που φαίνεται στο σχήμα 24, απεικονίζουν το σενάριο που μόλις περιγράψαμε το οποίο αφορά στο Intra-RASN Relocation.



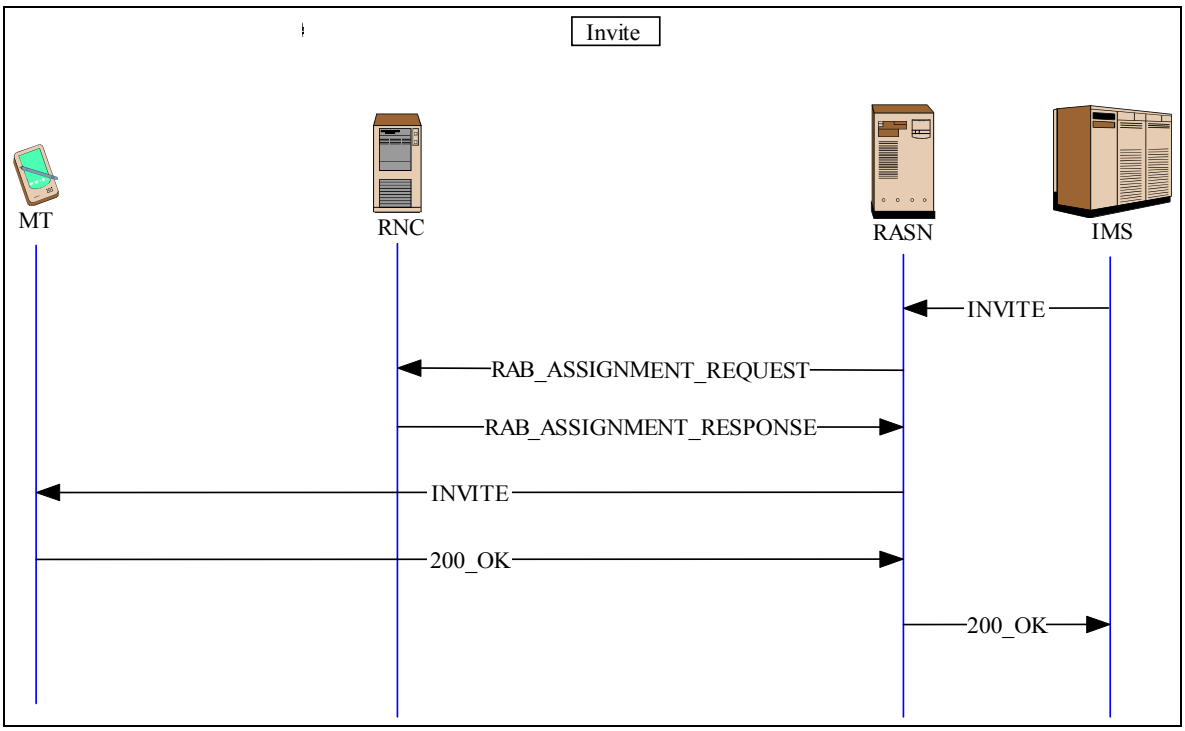
Σχήμα 23: Intra-RASN Relocation (εφαρμογή dummyclient).



Σχήμα 24: Intra-RASN Relocation (εφαρμογή CSCFDummy).

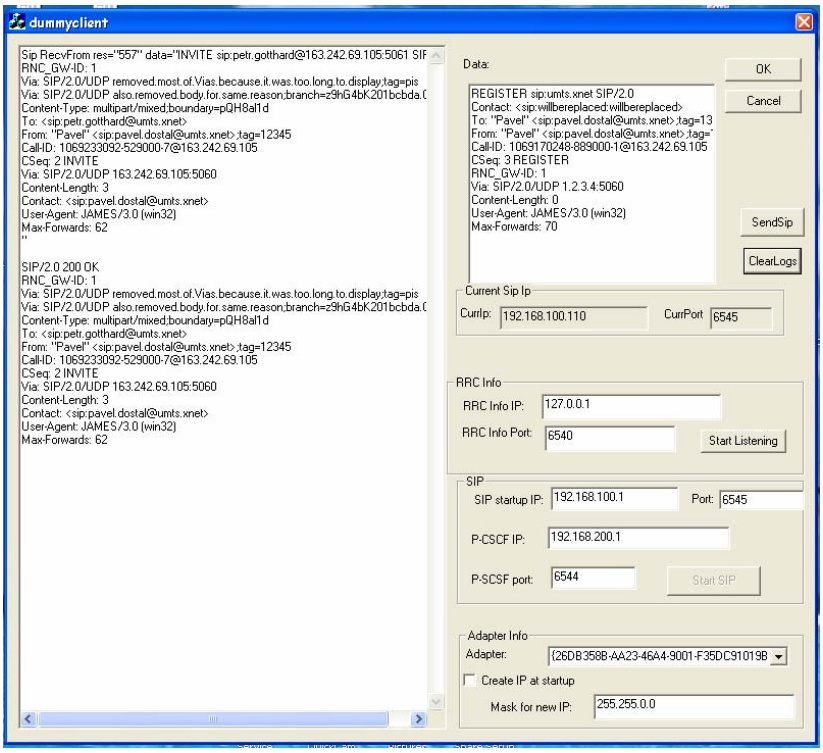
3.2.4.3 Εγκαθίδρυση media session

Όταν ο κόμβος RASN λάβει μήνυμα INVITE από τον κόμβο IMS (εφαρμογή CSCFDummy) ευρισκόμενος στην κατάσταση Connected, στέλνει μήνυμα RAB_ASSIGNMENT_REQUEST στον κόμβο RNC που εξυπηρετεί το κινητό τερματικό και μεταβαίνει στην κατάσταση INVITE_Received. Ο κόμβος RNC αποκρίνεται με το μήνυμα RAB_ASSIGNMENT_RESPONSE. Μόλις ο κόμβος RASN λάβει το μήνυμα αυτό, προωθεί το μήνυμα INVITE – προσθέτοντας το NAS context – στο κινητό τερματικό (εφαρμογή dummyclient) και μεταβαίνει στην κατάσταση RAB_Assignment_completed. Το κινητό τερματικό αποκρίνεται με το μήνυμα 200 OK το οποίο ο κόμβος RASN προωθεί στον κόμβο IMS αφαιρώντας το NAS context. Ο κόμβος RASN μεταβαίνει στην κατάσταση Connected. Η αλληλουχία των μηνυμάτων αυτών παρουσιάζεται στο πιο κάτω σχήμα (βλέπε σχήμα 25).

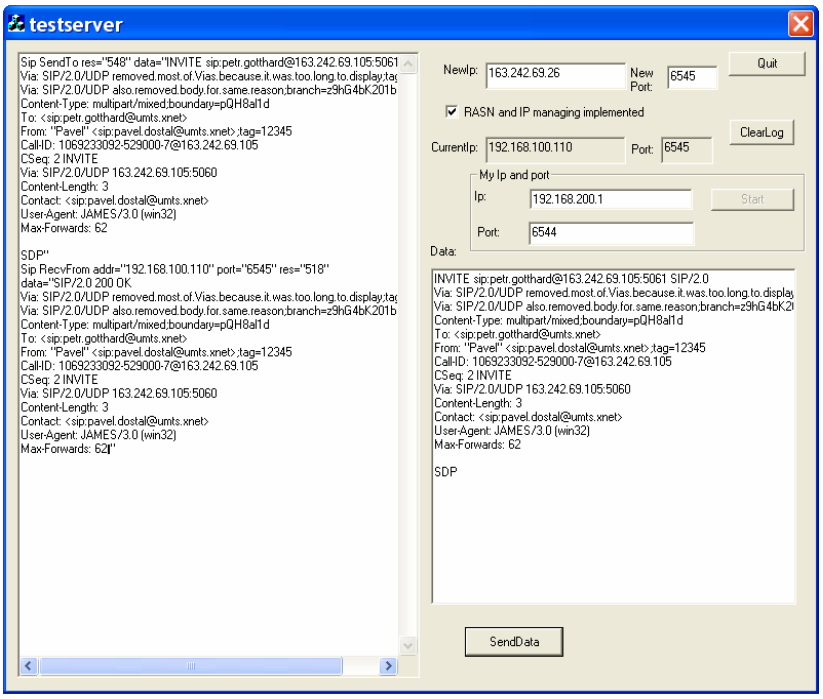


Σχήμα 25: Εγκαθίδρυση media session.

Το στιγμιότυπο της εφαρμογής dummyclient (κινητό τερματικό) που φαίνεται στο σχήμα 26 καθώς και το στιγμιότυπο της εφαρμογής CSCFDummy (κόμβος IMS) που φαίνεται στο σχήμα 27, απεικονίζουν το σενάριο που μόλις περιγράψαμε το οποίο αφορά στην εγκαθίδρυση media session.



Σχήμα 26: Εγκαθίδρυση media session (εφαρμογή dummyclient).



Σχήμα 27: Εγκαθίδρυση media session (εφαρμογή CSCFDummy).

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.

Όπως αναμένεται, η ενοποίηση των κόμβων SGSN και GGSN σε μια ενιαία οντότητα που ονομάζεται RASN αποτελεί μια καινοτομία πολλά υποσχόμενη για τα μελλοντικά δίκτυα. Η εφαρμογή της συνεπάγεται μια σειρά από πλεονεκτήματα σε σχέση με την ήδη υπάρχουσα προσέγγιση, τα οποία έχουμε προαναφέρει.

Παρόλα αυτά, στα πλαίσια αυτής της διπλωματικής εμείς υλοποιήσαμε μόνο ένα κύριο τμήμα της προσέγγισης του SAILOR Project αφού όλες οι διαδικασίες που προτείνονται από το εν λόγω πρόγραμμα δεν μπορούν να υλοποιηθούν μέσα στο χρονικό διάστημα που διαρκεί μια διπλωματική εργασία. Έτσι, δεν είμαστε σε θέση να αποφανθούμε με σιγουριά για τη λειτουργικότητα και τα προβλήματα που πιθανόν να παρουσιαστούν κατά την πλήρη εφαρμογή της. Το σίγουρο είναι ότι στο μέλλον θα συνεχιστούν οι προσπάθειες καθώς ήδη βρίσκεται σε πιλοτικό στάδιο με πολλούς συνεργαργαζόμενους οργανισμούς πανευρωπαϊκώς και ίσως αποτελέσει ένα ακόμα σημαντικό βήμα προς το όραμα της ενοποίησης των δικτύων κάτω από το πρίσμα του IP, με όποια πλεονεκτήματα και μειονεκτήματα αυτό συνεπάγεται.

Ωστόσο, σε γενικές γραμμές μπορούμε να βγάλουμε και μερικά χρήσιμα συμπεράσματα για τη νέα αρχιτεκτονική, τα οποία εξάγονται και από τα test-beds της ομάδας του προγράμματος SAILOR. Τέτοια είναι:

- η μείωση του αριθμού των μηνυμάτων και εξοικονόμηση πόρων καθώς το SIPRAN επιτελεί και τις NAS λειτουργίες και τις MM/SM λειτουργίες,
- η μείωση του χρόνου διαπομπής (handover) όταν ο κινητός κόμβος αλλάζει RASN,
- η πιο αποδοτική δρομολόγηση καθώς τώρα πια δεν υπάρχει ένας μοναδικός GGSN (avoid GGSN anchoring),
- η αποφυγή της GTP επικεφαλίδας μέσα στο δίκτυο πυρήνα κάτι που οδηγεί στην μείωση της επιβάρυνσης,

- η ελάττωση της καθυστέρηση από άκρο σε άκρο (end-to-end delay) διότι το μονοπάτι όπου διέρχεται η κίνηση δεδομένων είναι μικρότερο.

Φυσικά στο μέλλον απομένουν μια σειρά από βελτιώσεις και διορθώσεις που θα κάνουν την προσέγγιση πιο αποδοτική, όπως για παράδειγμα η υποστήριξη κινητικότητας σε TCP συνδέσεις και η σμίκρυνση τους μεγέθους των SIPRAN μηνυμάτων.

Παράρτημα Α – Παρουσίαση Κώδικα

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

Αρχείο: rasn.h

```
#ifndef		__rasn_h
#define		__rasn_h

#include "rawsockutils.h"

#include	<iostream>
#include	<sys/types.h>  /* basic system data types */
#include	<sys/socket.h>/* basic socket definitions */
#if TIME_WITH_SYS_TIME
#include	<sys/time.h>   /* timeval{} for select() */
#include	<time.h>	/* timespec{} for pselect() */
#else
#if HAVE_SYS_TIME_H
#include	<sys/time.h>   /* includes <time.h> unsafely */
#else
#include	<time.h>	/* old system? */
#endif
#endif
#include	<netinet/in.h> /* sockaddr_in{} and other Internet defns */
#include	<arpa/inet.h>  /* inet(3) functions */
#include	<errno.h>
#include	<fcntl.h>	/* for nonblocking */
#include	<netdb.h>
#include	<signal.h>
#include	<stdio.h>
#include	<stdlib.h>
#include	<string.h>
#include	<sys/stat.h>    /* for S_xxx file mode constants */
#include	<sys/uio.h>     /* for iovec{} and readv/writev */
#include	<unistd.h>
#include	<sys/wait.h>
#include	<sys/un.h>      /* for Unix domain sockets */
#include      <pthread.h>

#ifdef		HAVE_SYS_SELECT_H
# include	<sys/select.h> /* for convenience */
#endif

#ifdef		HAVE_SYS_SYSCTL_H
#ifdef		HAVE_SYS_PARAM_H
# include	<sys/param.h>/* OpenBSD prereq for sysctl.h */
#endif
# include	<sys/sysctl.h>
#endif

#ifdef		HAVE_POLL_H
# include	<poll.h>	/* for convenience */
#endif

#ifdef		HAVE_SYS_EVENT_H
# include	<sys/event.h> /* for kqueue */
#endif

#ifdef		HAVE_STRINGS_H
# include	<strings.h>     /* for convenience */
#endif

/* Three headers are normally needed for socket/file ioctl's:
 * <sys/ioctl.h>, <sys/filio.h>, and <sys/sockio.h>.
 */
#ifdef		HAVE_SYS_IOCTL_H
# include	<sys/ioctl.h>
#endif
#ifdef		HAVE_SYS_FILIO_H
# include	<sys/filio.h>
#endif
#ifdef		HAVE_SYS_SOCKIO_H
# include	<sys/sockio.h>
#endif

#ifdef		HAVE_PTHREAD_H
# include	<pthread.h>
#endif
```

```
#ifdef HAVE_NET_IF_DL_H
# include       <net/if_dl.h>
#endif

/* Define bzero() as a macro if it's not in standard C library. */
#ifndef         HAVE_BZERO
#define         bzero(ptr,n)                    memset(ptr, 0, n)
#endif

/* Following could be derived from SOMAXCONN in <sys/socket.h>, but many
   kernels still #define it as 5, while actually supporting many more */
#define         LISTENQ                 1024            /* 2nd argument to listen() */

/* Miscellaneous constants */
#define         MAXLINE                 4096            /* max text line length */
#define         BUFFSIZE    8192                        /* buffer size for reads and writes */

/* Define some port number that can be used for our examples */
#define         RNC_RASN_PORT                           5500                                    /* TCP socket ports --> RNC */
#define IMS_IP                                          "192.168.200.1"
#endif          /* __rasn_h */
```

Αρχείο: rasn.cpp

```
#include "rasn.h"
#include "patterns.h"

using namespace SailorTypes;
using namespace RawSockTypes;
using namespace std;

struct info {
                int rawfd_mt;                           // raw socket descriptor (between the mobile terminal and RASN)
                int rawfd_ims;                          // raw socket descriptor (between RASN and IMS)
                struct sockaddr_ll *dev0;
                struct sockaddr_ll *dev1;
} thread_info;

void *connect_rnc(void *);                              // function that handles the TCP sockets that connect RNCs with RASN
void *handle_ims(void *);                               // function that handles incoming messages from IMS to RASN (through raw socket)
void parser(Sip_parsed_msg &,char *,int);               // function that parses SIP messages -- defined in parser.cpp

int  rnc[N_RNC_GWS];                                    // array that stores new descriptors when connection is established
int  mt_rnc[N_TERMINALS];                               // array that stores the number of the RNC that every mobile terminal is connected to

SIPConnection *mt[N_TERMINALS];

// function that executes the main thread
int main(int argc,char **argv)
{
                pthread_t tid0,tid1;
                Sip_parsed_msg function_info;

                int rawfd0,rawfd1;
                struct ifreq ifr0,ifr1;
                struct sockaddr_ll dev0,dev1;
                char recvbuff[BUFFSIZE];
                struct sockaddr *from;
                char *sipmes;
                socklen_t len,addrlen;
                struct ip_packet *ip;
                struct in_addr inaddr;
                ssize_t fromMT;

                mt[0] = new SIPConnection();
                mt[1] = new SIPConnection();

                /*****************************************************************************
                *       Create Raw Sockets and bind them with ethernet interfaces                *
                *****************************************************************************/
                //creating a raw socket and binding it with eth0 interface CLIENT -> RASN
                if((rawfd0 = RawSockUtilities::Instance().create_raw_socket()) == FAIL )
                                exit(0);
                if((RawSockUtilities::Instance().set_if_index("eth0", rawfd0, &ifr0)) == FAIL)
                                exit(0);
                RawSockUtilities::Instance().set_sockaddr_ll_values(dev0, &ifr0);
                if((RawSockUtilities::Instance().bind_raw_socket(rawfd0, dev0, &ifr0)) == FAIL)
                                exit(0);
                RawSockUtilities::Instance().set_dst_mac_addr(dev0, MT);
                //creating a raw socket and binding it with eth1 interface RASN -> IMS
                if((rawfd1 = RawSockUtilities::Instance().create_raw_socket()) == FAIL)
                                exit(0);
                if((RawSockUtilities::Instance().set_if_index("eth1", rawfd1, &ifr1)) == FAIL)
                                exit(0);
                RawSockUtilities::Instance().set_sockaddr_ll_values(dev1, &ifr1);
                if((RawSockUtilities::Instance().bind_raw_socket(rawfd1, dev1, &ifr1)) == FAIL)
                                exit(0);
                RawSockUtilities::Instance().set_dst_mac_addr(dev1, IMS);
                //*****************************************************************************


                // raw socket info that are sent to every thread
                thread_info.rawfd_mt = rawfd0;
                thread_info.rawfd_ims = rawfd1;
                thread_info.dev0 = &dev0;
                thread_info.dev1 = &dev1;
```

```
		// thread that handles TCP connections from RNCs to RASN
		pthread_create(&tid0, NULL, connect_rnc, &thread_info);
		// thread that handles incoming messages from IMS to RASN (raw socket)
		pthread_create(&tid1, NULL, handle_ims, &thread_info);

		for(int i=0; i<N_TERMINALS; i++)
			mt_rnc[i] = -1;

		while(1) {
			// raw socket that accepts packets from MTs and route them to IMS
			if( (fromMT = recvfrom(rawfd0, recvbuff, sizeof(recvbuff), 0, from, &len)) < 0) {
				perror("RECV\n");
				exit(0);
			}
			if(fromMT > 0) {
				ip = (struct ip_packet *)recvbuff;
				inaddr.s_addr = ip->ip_dest;
				sipmes = (char *)malloc( (fromMT-28)*sizeof(char) );
				memcpy((void *)sipmes, (void *)(recvbuff+28), fromMT-28);

				if (strstr(sipmes, "sip") == NULL) {
					// it is not a SIP message, so forward it to the desirable destination if available
					 ip = (struct ip_packet *)recvbuff;
					if(((ntohl(ip->ip_dest) & 0xffffff00) == 0xc0a8c800)) {
						addrlen = sizeof(struct sockaddr_ll);
						if( sendto(rawfd1, recvbuff, fromMT, 0, (struct sockaddr *) &dev1, (socklen_t)addrlen) <= 0) {
							perror("SEND TO");
							exit(0);
						}

					}
				continue;
			}
			// it is a SIP message so parse it
			parser(function_info,sipmes,fromMT-28);
			// pass to a function all the information needed
			function_info.bytes_read = fromMT;				// number of bytes read from raw socket
			function_info.rawfd = rawfd1;					// raw socket descriptor

			switch(function_info.msg) {
				case REGISTER:
					cout << "REGISTER MESSAGE RECEIVED from MT: " << function_info.from << endl;
					// check if the mobile terminal that sends this message is already registered
					if(mt_rnc[function_info.from] == -1) {
						// current RNC gateway of the mobile terminal
						mt_rnc[function_info.from] = function_info.rnc_gw;
						// new RASN tcp socket descriptor
						function_info.new_tcpfd = rnc[mt_rnc[function_info.from]];
						mt[function_info.from-1]->SendRegister(recvbuff, sipmes, &function_info, &dev1);
					}
					else {// it is already registered and sends REGISTER message, having in RNC-ID_GW field the same RNC ID
						// as the last REGISTER message => INTRA RASN ROUTING UPDATE
						if(mt_rnc[function_info.from] == function_info.rnc_gw) {
							cout << "INTRA RASN ROUTING UPDATE NOT IMPLEMENTED" << endl;
							continue;
						}
						else {		// mobile terminal informs IMS that it has changed serving RNC node
							// old RASN tcp socket descriptor
							function_info.old_tcpfd = rnc[mt_rnc[function_info.from]];
							mt_rnc[function_info.from] = function_info.rnc_gw;
							// new RASN tcp socket descriptor
							function_info.new_tcpfd = rnc[mt_rnc[function_info.from]];
							mt[function_info.from-1]->RelocReq(recvbuff, sipmes, &function_info, &dev1);
						}
					}
					break;
				case _200_OK:
					cout << "200 OK MESSAGE RECEIVED from MT: " << function_info.from << endl;
					mt[function_info.from-1]->Send200_OK(recvbuff, sipmes, &function_info, &dev1);
					break;
				default:
					cout << "WRONG MESSAGE RECEIVED!!!" << endl;
					break;
			}

		}
	}
	return 0;
}

void *connect_rnc(void *arg)
{
	int		i, maxi, maxfd, listenfd, connfd, sockfd;
	int		nready;
	ssize_t		n;
	fd_set		rset, allset;
	char		buff[MAXLINE];
	socklen_t rnclen;
	struct sockaddr_in rncaddr, rasnaddr;

	if( (listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
		perror("SOCKET\n");
		exit(1);
	}

	int enable = 1;
	setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int));

	bzero(&rasnaddr, sizeof(rasnaddr));
```

```
rasnaddr.sin_family      = AF_INET;
rasnaddr.sin_addr.s_addr = htonl(INADDR_ANY);
rasnaddr.sin_port        = htons(RNC_RASN_PORT);

if ( (bind(listenfd, (sockaddr *) &rasnaddr, sizeof(rasnaddr))) < 0) {
        perror("BIND\n");
        exit(1);
}

listen(listenfd, LISTENQ);

maxfd = listenfd;                                       // initialize
maxi = -1;                                              // index into rnc[] array
for (i = 0; i < N_RNC_GWS; i++)
        rnc[i] = -1;                                    // -1 indicates available entry
FD_ZERO(&allset);
FD_SET(listenfd, &allset);
while (1) {
        rset = allset;                  // structure assignment
        // select waits for something to happen (establishment of new rnc connection or the arrival of data)
        nready = select(maxfd+1, &rset, NULL, NULL, NULL); // returns positive count of ready discriptors

        if (FD_ISSET(listenfd, &rset)) {        // new RNC connection
                rnclen = sizeof(rncaddr);
                connfd = accept(listenfd, (sockaddr *) &rncaddr, &rnclen);

                printf("RNC %d from: %s:%d CONNECTED\n", ntohs(rncaddr.sin_port) - 5500,
                                inet_ntop(AF_INET, &rncaddr.sin_addr, buff, sizeof(buff)),
                                ntohs(rncaddr.sin_port));


                rnc[ntohs(rncaddr.sin_port)-5500] = connfd; // save descriptor

                FD_SET(connfd, &allset);    // add new descriptor to set
                if (connfd > maxfd)
                        maxfd = connfd;                         // for select

                if (--nready <= 0)
                        continue;                       // no more readable descriptors
        }

        for (i = 0; i <= N_RNC_GWS; i++) {      // check all RNCs for RANAP messages
                if ( (sockfd = rnc[i]) < 0)
                        continue;
                if (FD_ISSET(sockfd, &rset)) {
                        if ( (n = read(sockfd, buff, MAXLINE)) == 0) {
                                                // connection closed by RNC
                                close(sockfd);
                                FD_CLR(sockfd, &allset);
                                rnc[i] = -1;
                                printf("CONNECTION LOST from RNC %d\n",i);
                        }
                        else {          // all cases are inside this else
                                int nbytes, index=0;
                                int msg_type;

                                memcpy((char *)&msg_type, buff, sizeof(msg_type));
                                switch(msg_type) {
                                        case RELOCATION_REQUEST_ACK:
                                        cout << "RELOCATION REQUEST ACK MESSAGE RECEIVED" << endl;
                                                struct Reloc_Req_Ack reloc_req_ack;
                                                memcpy((char *)&reloc_req_ack, buff, sizeof(reloc_req_ack));
                                        mt[reloc_req_ack.mtID-1]->SendRegister(NULL, NULL, NULL, NULL);
                                                continue;
                                                break;
                                        case RELOCATION_COMPLETE:
                                        cout << "RELOCATION COMPLETE MESSAGE RECEIVED" << endl;
                                                struct Reloc_Complete reloc_complete;
                                                 memcpy((char *)&reloc_complete, buff, sizeof(reloc_complete));
                                                index += sizeof(reloc_complete);
                                                struct Iu_Release_Command iu_release_command;
                                                iu_release_command.msgID = Iu_RELEASE_COMMAND;
                                                iu_release_command.mtID = reloc_complete.mtID;
                                        memcpy((char *)buff, (char *)&iu_release_command, sizeof(iu_release_command));
                                                if ( (nbytes=write(sockfd, buff, index)) != index ) {
                                                        perror("SEND");
                                                }
                                                //printf("SENT REPLY %d BYTES\n", nbytes);
                                                break;
                                        case Iu_RELEASE_COMPLETE:
                                        cout << "Iu RELEASE COMPLETE MESSAGE RECEIVED" << endl;
                                                // do nothing at all
                                                break;
                                        case RAB_ASSIGNMENT_RESPONSE:
                                        cout << "RAB ASSIGNMENT RESPONSE MESSAGE RECEIVED" << endl;
                                                struct RAB_Assign_Resp rab_assign_resp;
                                                memcpy((char *)&rab_assign_resp, buff, sizeof(rab_assign_resp));
                                        mt[rab_assign_resp.mtID-1]->SendInvite(NULL, NULL, NULL, NULL);
                                                continue;
                                                break;
                                        default:
                                                cout << "WRONG MESSAGE RECEIVED" << endl;
                                                break;
                                }
                        }

                        if (--nready <= 0)
                                break;          // no more readable descriptors
                }
        }
```

```
		}
		return (NULL);
}

void *handle_ims(void *infoptr)
{
		char recvbuff[BUFFSIZE];
		struct sockaddr *from;
		char *sipmes;
		socklen_t len,addrlen;
		struct ip_packet *ip;
		ssize_t fromIMS;
		struct info *iptr;		// struct that contains the required information (i.e socket descriptors)
		Sip_parsed_msg function_info;

		iptr = (struct info *)infoptr;

		while(1) {
			// raw socket that accepts packets from MTs and route them to IMS
			if( (fromIMS = recvfrom(iptr->rawfd_ims, recvbuff, sizeof(recvbuff), 0, from, &len)) < 0) {
				perror("RECV\n");
				exit(0);
			}
			if(fromIMS > 0) {
				sipmes = (char *)malloc( (fromIMS-28)*sizeof(char) );
				memcpy((void *)sipmes, (void *)(recvbuff+28), fromIMS-28);
				if (strstr(sipmes, "sip") == NULL) {
					// it is not a SIP message, so forward it to the desirable destination if available
					ip = (struct ip_packet *)recvbuff;
					if((ntohl(ip->ip_dest) & 0xffffff00) == 0xc0a86400) {
						addrlen = sizeof(struct sockaddr_ll);
					if( sendto(iptr->rawfd_mt, recvbuff, fromIMS, 0, (struct sockaddr *) iptr->dev0, (socklen_t)addrlen) <= 0) {
							perror("SEND TO");
							exit(0);
					}

					}
					continue;
				}
				// it is a SIP message so parse it
				parser(function_info,sipmes,fromIMS-28);

				// pass to a function all the information needed
				function_info.bytes_read = fromIMS;			// number of bytes read from raw socket
				function_info.rawfd = iptr->rawfd_mt;			// raw socket descriptor
				function_info.rnc_gw = mt_rnc[function_info.from];	// RNC gateway of the mobile terminal

				switch(function_info.msg) {
					case _200_OK:
						cout << "200 OK MESSAGE RECEIVED from IMS"<< endl;
						 mt[function_info.from-1]->Send200_OK(recvbuff, sipmes, &function_info, iptr->dev0);
						break;
					case INVITE:
						cout << "INVITE MESSAGE RECEIVED from IMS" << endl;
						function_info.new_tcpfd = rnc[mt_rnc[function_info.from]];
						// used while forwarding INVITE message (for NAS context)
						function_info.rnc_gw = mt_rnc[function_info.from];
						mt[function_info.from-1]->RABAssReq(recvbuff, sipmes, &function_info, iptr->dev0);
						break;
					default:
						cout << "WRONG MESSAGE RECEIVED!!!" << endl;
						break;
				}

			}
		}
		return (NULL);
}
```

Αρχείο: parser.cpp

```
/***************************************************************************
                parser.cpp  -  description
                -------------------
   begin        : Sat Apr 16 2005
   copyright    : (C) 2005 by tkampanakis, pantoniou, mkallitsis
   email        : {el00129, el00600, el00602}@mail.ntua.gr
 ***************************************************************************/

#include <iostream>
#include <string.h>
#include "sailorCommonTypes.h"

using namespace std;
using namespace SailorTypes;

// function that parses the SIP message
void parser(Sip_parsed_msg &out,char *buff,int len)
{

                                string st;
                                st.assign(buff);
                                string tmpStr;
                                string st2=st;
```

```
                                        // The RNC line is deleted from the sip message if it is present and the results are stored in st2
                                        if ( st2.find("RNC_GW-ID: ", 0 ) != string::npos )
                                          st2.erase(st2.find("RNC_GW-ID: ", 0),st2.find("Via: ", 0)-st2.find("RNC_GW-ID: ", 0));
                                        // erase all useless characters at the end of the string
                                        st2 = st2.substr(0,st2.find_last_of(":")+5);

                                        // cout<<"------- WITHOUT SIPRANAP -----------"<<endl;
                                        // cout<<st2<<endl;
                                        // Erase the SIP-RAN message that is stored in buff
                                        for(int i=0; i<len; i++) buff[i] = '\0';
                                        // and copy the new pure SIP message in buff
                                        memcpy((void *)buff,(void *)st2.c_str(),st2.length());
                                        out.length = st2.length();
                                        // cout<<"------------------------------------"<<endl;

                                        // The method is stored in msg field
                                        tmpStr=st.substr(0,st.find(' ',2));
                                        if(tmpStr == "REGISTER")                        out.msg = REGISTER;
                                        else if(tmpStr == "INVITE")                     out.msg = INVITE;
                                        else if(st.substr(8,3) == "200")    out.msg = _200_OK;

                                        // The message is parsed and all the needed fields are collected in a dedicated special structure
                                        // that is defined in sailorCommonTypes.h
                                        while (1) {
                                                tmpStr=st.substr(0,st.find('\n',0));
                                                if ( tmpStr.find("Contact: ", 0 ) != string::npos ) {
                                                out.contact = true;
                                                tmpStr=tmpStr.substr(tmpStr.find("Contact: ",0)+9,tmpStr.length()-tmpStr.find("Contact: ",0)-9);
                                                string tmpStr2 = tmpStr;
                                                int position = tmpStr2.find(" ");
                                                while(position!=string::npos) {
                                                        tmpStr2.replace(position, 1, ".");
                                                        position = tmpStr2.find(" ", position + 1);
                                                }
                                                if(((tmpStr2.find_last_of(".")-tmpStr2.find_first_of(".")+1) == (tmpStr.length()-1)) || (tmpStr.length() == 1))
                                                        out.contact = false;
                                                }
                                                else if ( tmpStr.find("To: ", 0 ) != string::npos ) {
                                                        tmpStr=tmpStr.substr(tmpStr.find("To: ",0)+4,tmpStr.find(';',0)-4);
                                                }
                                                else if ( tmpStr.find("From: ", 0 ) != string::npos ) {
                                                        tmpStr=tmpStr.substr(tmpStr.find("From: ",0)+6, tmpStr.find(';',0)-6);
                                                        tmpStr=tmpStr.substr(tmpStr.find("<",0)+1, tmpStr.find(">",0) - tmpStr.find("<",0)-1);
                                                        if(tmpStr == SIP_UA_1)
                                                                out.from = 1;
                                                        else if(tmpStr == SIP_UA_2)
                                                                out.from = 2;
                                                }
                                                else if ( tmpStr.find("Call-ID: ", 0 ) != string::npos ) {
                                                        tmpStr=tmpStr.substr(tmpStr.find("Call-ID: ",0)+9,tmpStr.length()-tmpStr.find("Call-ID: ",0)-9);
                                                }
                                                else if ( tmpStr.find("CSeq: ", 0 ) != string::npos ) {
                                                        tmpStr=tmpStr.substr(tmpStr.find("CSeq: ",0)+6,tmpStr.length()-tmpStr.find("CSeq: ",0)-6);
                                                        //int k=atoi(tmpStr.substr(0,tmpStr.find(' ',0)).c_str());
                                                        tmpStr=tmpStr.substr(tmpStr.find(' ',0)+1,tmpStr.length()-tmpStr.find(' ',0)-2);
                                                        // return the message type that is contained in CSeq field
                                                        if(tmpStr == "REGISTER")   out.cseq = REGISTER;
                                                        else if(tmpStr == "INVITE")     out.cseq = INVITE;
                                                }
                                                else if ( tmpStr.find("RNC_GW-ID: ", 0 ) != string::npos ) {
                                                tmpStr=tmpStr.substr(tmpStr.find("RNC_GW-ID: ",0)+11,tmpStr.length()-tmpStr.find("RNC_GW-ID: ",0)-11);
                                                        // return the id of the RNC gateway
                                                        out.rnc_gw = atoi(tmpStr.c_str());
                                                }
                                                if (st.find('\n',0)==string::npos) break;
                                                else st=st.substr(st.find('\n',0)+1,st.length());
                                        }

}
```

Αρχείο: patterns.h

```
/***************************************************************************
                        patterns.h  -  description
                        -------------------
    begin               : Sun Apr 17 2005
    copyright           : (C) 2005 by pantoniou, tkampanakis, mkallitsis
    email               : {el00600, el00129, el00602}@mail.ntua.gr
 ***************************************************************************/

#ifndef PATTERNS_H
#define PATTERNS_H

#include "rasn.h"

class RASNState;

class SIPConnection {
public:
                                SIPConnection();
                                // SIP messages
                                void SendRegister(char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
                                void SendInvite(char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
                                void SendAck(char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
                                void Send200_OK(char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
```

```
                                // RANAP messages

                                void RABAssReq(char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
                                void RelocReq(char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
protected:
                                friend class RASNState;
                                void ChangeState(RASNState *);
private:
                                RASNState* _state;
};

class RASNState {
public:
                                // SIP messages
                                virtual void SendRegister(SIPConnection *, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *) = 0;
                                virtual void SendInvite(SIPConnection *, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *) = 0;
                                virtual void SendAck(SIPConnection *, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *) = 0;
                                virtual void Send200_OK(SIPConnection *, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *) = 0;
                                // RANAP messages
                                virtual void RABAssReq(SIPConnection *, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *) = 0;
                                virtual void RelocReq(SIPConnection *, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) = 0 ;
protected:
                                void ChangeState(SIPConnection *,RASNState *);
};

class Detached : public RASNState {
public:
                                static Detached *Instance();
                                virtual void SendRegister(SIPConnection *, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
                                virtual void SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
                                virtual void SendAck(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
                                virtual void Send200_OK(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
                                virtual void RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
                                virtual void RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
private:
                                static Detached* instance;
};

class Pre_Connected : public RASNState {
public:
                                static Pre_Connected *Instance();
                                virtual void SendRegister(SIPConnection *, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
        virtual void SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void SendAck(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void Send200_OK(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
        virtual void RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
private:
                                static Pre_Connected* instance;
};

class Connected : public RASNState {
public:
        static Connected *Instance();
        virtual void SendRegister(SIPConnection *, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
        virtual void SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void SendAck(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void Send200_OK(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
        virtual void RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
private:
        static Connected* instance;
};

class Intra_RASN_Reloc_detected : public RASNState {
public:
        static Intra_RASN_Reloc_detected *Instance();
        virtual void SendRegister(SIPConnection *, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
        virtual void SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void SendAck(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void Send200_OK(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
        virtual void RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
private:
        static Intra_RASN_Reloc_detected* instance;
};

class Pre_Relocation : public RASNState {
public:
        static Pre_Relocation *Instance();
        virtual void SendRegister(SIPConnection *, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
        virtual void SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void SendAck(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void Send200_OK(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
        virtual void RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
private:
        static Pre_Relocation* instance;
};

class INVITE_received : public RASNState {
public:
        static INVITE_received *Instance();
        virtual void SendRegister(SIPConnection *, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
        virtual void SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void SendAck(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void Send200_OK(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
        virtual void RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
        virtual void RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
private:
```

```
	static INVITE_received* instance;
};

class RAB_Assignment_completed : public RASNState {
public:
	static RAB_Assignment_completed *Instance();
	virtual void SendRegister(SIPConnection *, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
	virtual void SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
	virtual void SendAck(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
	virtual void Send200_OK(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
	virtual void RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg*, struct sockaddr_ll *);
	virtual void RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);
private:
	static RAB_Assignment_completed* instance;
};

#endif
```

Αρχείο: patterns.cpp

```
/***************************************************************************
                          patterns.cpp  -  description
                          ----------
    begin                : Sun Apr 17 2005
    copyright            : (C) 2005 by pantoniou, tkampanakis, mkallitsis
    email                : {el00600, el00129, el00602}@mail.ntua.gr
 ***************************************************************************/
#include <iostream>
#include <sstream>
#include "patterns.h"

using namespace std;
using namespace SailorTypes;

// available IP addresses that are given to every mobile terminal which enters in a  RASN region
char *ip_pool[IP_POOL_SIZE][2] =
{
                                        {"192.168.100.110", "free"},
                                        {"192.168.100.111", "free"}
};

// structure that stores the information about a SIP message that will be forwarded to its final destination (IMS or MT) after RANAP messages
struct inter {
                                        char recvbuff[BUFFSIZE];
                                        char sipmes[BUFFSIZE];
                                        Sip_parsed_msg *function_info;
                                        struct sockaddr_ll *dev;
} inter_thread_info;

// variable that stores the source IP address of the initial REGISTER message
char *sourceip;

/*****************************************************
*          SIPConnection class member-function definition            *
*****************************************************/

// default constructor
SIPConnection::SIPConnection () {
                                        _state = Detached::Instance();
}

// set/change the state of the connection
void SIPConnection::ChangeState (RASNState* s) {
                                        _state = s;
}

void SIPConnection::SendRegister ( char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev) {
                                        _state->SendRegister(this, recvbuff, sipmes, function_info, dev);
}

void SIPConnection::SendInvite (char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev) {
                                        _state->SendInvite(this, recvbuff, sipmes, function_info, dev);
}

void SIPConnection::SendAck (char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev) {
                                        _state->SendAck(this, recvbuff, sipmes, function_info, dev);
}

void SIPConnection::Send200_OK (char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev) {
                                        _state->Send200_OK(this, recvbuff, sipmes, function_info, dev);
}

void SIPConnection::RABAssReq (char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev) {
                                        _state->RABAssReq(this, recvbuff, sipmes, function_info, dev);
}

void SIPConnection::RelocReq (char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev) {
                                        _state->RelocReq(this, recvbuff, sipmes, function_info, dev);
}

/*****************************************************
*  RASNState class member-function definition                *
*****************************************************/

void RASNState::SendRegister(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void RASNState::SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
```

```
void RASNState::SendAck(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void RASNState::Send200_OK(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void RASNState::RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void RASNState::RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void RASNState::ChangeState(SIPConnection* c, RASNState* s) {
                                        c->ChangeState(s);
}

/*****************************************************
*   Detached class member-function definition    *
*****************************************************/
Detached* Detached::instance = 0;

Detached *Detached::Instance() {
                                        if(instance == 0)
                                            instance = new Detached();
                                        return instance;
}

void Detached::SendRegister(SIPConnection *c, char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev) {

                                        struct ip_packet *ip;
                                        struct in_addr inaddr;
                                        struct Init_TE_Address te;
                                        int i,nbytes;
                                        char buff[MAXLINE];
                                        string newsip,newsip1,newsip2,temp;

                                        // pure SIP REGISTER message is sent to IMS
                                        ip = (struct ip_packet *)recvbuff;
                                        inaddr.s_addr = ip->ip_source;
                                        sourceip = inet_ntoa(inaddr);
                                        printf("Initial srcIP = %s\n", sourceip);
                                        for(i=0; i<IP_POOL_SIZE; i++) {
                                            if(strcmp(ip_pool[i][1],"free") == 0) {
                                                    ip->ip_source = inet_addr(ip_pool[i][0]);     // returns 32-bit binary network byte ordered IPv4 address
                                                    ip_pool[i][1] = "given";
                                                    printf("New srcIP = %s\n", ip_pool[i][0]);
                                                    break;
                                            }
                                        }

                                        newsip.assign(sipmes);
                                        newsip1 = newsip.substr(0, newsip.find("<sip", 0) + 5);
                                        newsip2 = newsip.substr(newsip.find(":", newsip.find("<sip:",0) + 6));
                                        temp.assign(ip_pool[i][0]);
                                        newsip1.append(temp);
                                        newsip1.append(newsip2);

                                        function_info->length = newsip1.length();

                                        RawSockUtilities::Instance().Sendto(recvbuff, (char *)newsip1.c_str(), function_info, dev);

                                        // INITIAL_TE_ADDRESS message is sent to the proper RNC
                                        te.msgID = INITIAL_TE_ADDRESS;
                                        te.mtID  = function_info->from;
                                        //te.newIP[] = "192.168.100.110";

                                        memcpy( (char *)buff, (char *)&te, sizeof(te) );
                                        if ( (nbytes=write(function_info->new_tcpfd, buff, sizeof(te))) != sizeof(te) )
                                        {
                                            perror("SEND");
                                        }
                                        ChangeState(c, Pre_Connected::Instance());
}

// override base class functions
void Detached::SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { cout << "HELLO_0" << endl; }
void Detached::SendAck(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void Detached::Send200_OK(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void Detached::RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { cout << "Detached" << endl; }
void Detached::RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }

/*****************************************************
*   Pre_Connected class member-function definition    *
*****************************************************/
Pre_Connected* Pre_Connected::instance = 0;

Pre_Connected *Pre_Connected::Instance() {
    if(instance == 0)
        instance = new Pre_Connected();
    return instance;
}

void Pre_Connected::Send200_OK(SIPConnection *c, char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev) {

    struct ip_packet *ip;
                                        string newsip,newsip1,newsip2,temp;
                                        ostringstream o;

                                        ip = (struct ip_packet *)recvbuff;
                                        ip->ip_dest = inet_addr(sourceip);

                                        newsip.assign(sipmes);
                                        newsip1 = newsip.substr(0,newsip.find("Via:", 0));
                                        newsip2 = newsip.substr(newsip.find("Via:", 0));
                                        o << function_info->rnc_gw;
                                        temp = "RNC_GW-ID: "+o.str();                   // insert NAS context => construct SIP-RAN message
                                        newsip1.append(temp+"\n");
                                        newsip1.append(newsip2);
```

```
                                        function_info->length = newsip1.length();

                                        RawSockUtilities::Instance().Sendto(recvbuff, (char *)newsip1.c_str(), function_info, dev);
                                        ChangeState(c, Connected::Instance());
}

// override base class functions
void Pre_Connected::SendRegister(SIPConnection*, char *, char *, Sip_parsed_msg *out, struct sockaddr_ll *) { }
void Pre_Connected::SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { cout << "HELLO_1" << endl; }
void Pre_Connected::SendAck(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void Pre_Connected::RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { cout << "Pre_Connected" << endl; }
void Pre_Connected::RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }

/*************************************************
*   Connected class member-function definition    *
*************************************************/
Connected* Connected::instance = 0;

Connected *Connected::Instance() {
      if(instance == 0)
            instance = new Connected();
      return instance;
}

void Connected::RelocReq(SIPConnection *c, char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev) {
                                        struct Reloc_Req reloc_req;
                                        char buf[MAXLINE];
                                        int nbytes;

                                        // store the REGISTER message. This message will be sent after the message RELOCATION_COMPLETE has received
                                        memcpy((char *)(inter_thread_info.recvbuff), (char *)recvbuff, function_info->bytes_read) ;
                                        memcpy((void *)(inter_thread_info.sipmes), (void *)sipmes, function_info->length);
                                        inter_thread_info.function_info = function_info;
                                        inter_thread_info.dev = dev;

                                        // send RELOCATION REQUEST message to the new RNC
                                        reloc_req.msgID = RELOCATION_REQUEST;
                                        reloc_req.mtID  = function_info->from;

                                        memcpy( (char *)buf, (char *)&reloc_req, sizeof(reloc_req) );
                                        if ( (nbytes=write(function_info->new_tcpfd, buf, sizeof(reloc_req))) != sizeof(reloc_req) )
                                        {
                                              perror("SEND");
                                        }

                                        ChangeState(c, Intra_RASN_Reloc_detected::Instance());
}

void Connected::RABAssReq(SIPConnection *c , char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev) {
                                        struct RAB_Assign_Req rab_assign_req;
      char buff[MAXLINE];
      int nbytes;

      // store the INVITE message. This message will be sent after the message RAB_ASSIGNMENT_RESPONSE has received
      memcpy((char *)(inter_thread_info.recvbuff), (char *)recvbuff, function_info->bytes_read) ;
      memcpy((void *)(inter_thread_info.sipmes), (void *)sipmes, function_info->length);
      inter_thread_info.function_info = function_info;
      inter_thread_info.dev = dev;

      // send RAB ASSIGNMENT REQUEST to the MT's serving RNC
      rab_assign_req.msgID = RAB_ASSIGNMENT_REQUEST;
      rab_assign_req.mtID  = function_info->from;
      //rab_assign_req.add_rabs[0].RABID = 0;

      memcpy((char *)buff, (char *)&rab_assign_req, sizeof(rab_assign_req));
      if ( (nbytes=write(function_info->new_tcpfd, buff, sizeof(rab_assign_req))) != sizeof(rab_assign_req) )
      {
            perror("SEND");
      }

      ChangeState(c, INVITE_received::Instance());
}


// override base class functions
void Connected::SendRegister(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void Connected::SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { cout << "HELLO_2" << endl; }
void Connected::SendAck(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void Connected::Send200_OK(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }


/*****************************************************************
*   Intra_RASN_Reloc_detected class member-function definition    *
*****************************************************************/
Intra_RASN_Reloc_detected* Intra_RASN_Reloc_detected::instance = 0;

Intra_RASN_Reloc_detected *Intra_RASN_Reloc_detected::Instance() {
      if(instance == 0)
            instance = new Intra_RASN_Reloc_detected();
      return instance;
}

void Intra_RASN_Reloc_detected::SendRegister(SIPConnection *c, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) {
                              // forward REGISTER message to the IMS
                              RawSockUtilities::Instance().Sendto(inter_thread_info.recvbuff,inter_thread_info.sipmes,inter_thread_info.function_info,inter_thread_info.dev);
                              ChangeState(c, Pre_Relocation::Instance());
}

// override base class functions
```

```
void Intra_RASN_Reloc_detected::Send200_OK(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void Intra_RASN_Reloc_detected::SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { cout << "HELLO_3" << endl; }
void Intra_RASN_Reloc_detected::SendAck(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void Intra_RASN_Reloc_detected::RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { cout << "State 3" << endl; }
void Intra_RASN_Reloc_detected::RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }

/*********************************************************
*            Pre_Relocation class member-function definition              *
*********************************************************/
Pre_Relocation* Pre_Relocation::instance = 0;

Pre_Relocation *Pre_Relocation::Instance() {
      if(instance == 0)
            instance = new Pre_Relocation();
      return instance;
}

void Pre_Relocation::Send200_OK(SIPConnection *c, char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev) {

      struct Reloc_Command reloc_command;
      char buff[MAXLINE];
      int nbytes;
      string newsip,newsip1,newsip2,temp;
      ostringstream o;

      newsip.assign(sipmes);
      newsip1 = newsip.substr(0,newsip.find("Via:", 0));
      newsip2 = newsip.substr(newsip.find("Via:", 0));
      o << function_info->rnc_gw;
      temp = "RNC_GW-ID: "+o.str();          // insert NAS context => construct SIP-RAN message
      newsip1.append(temp+"\n");
      newsip1.append(newsip2);

      function_info->length = newsip1.length();

      RawSockUtilities::Instance().Sendto(recvbuff, (char *)newsip1.c_str(), function_info, dev);

      // send RELOCATION COMMAND message to the old RNC
      reloc_command.msgID = RELOCATION_COMMAND;
      reloc_command.mtID  = function_info->from;
      reloc_command.rasnID= function_info->rnc_gw;

      memcpy((char *)buff, (char *)&reloc_command, sizeof(reloc_command));
      if ( (nbytes=write(inter_thread_info.function_info->old_tcpfd, buff, sizeof(reloc_command))) != sizeof(reloc_command) )       {
            perror("SEND");
      }

       // go to Connected state
       ChangeState(c, Connected::Instance());
}

// override base class functions
void Pre_Relocation::SendRegister(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void Pre_Relocation::SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { cout << "HELLO_4" << endl; }
void Pre_Relocation::SendAck(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void Pre_Relocation::RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { cout << "State 4" << endl; }
void Pre_Relocation::RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }


/*********************************************************
*   INVITE_received class member-function definition    *
*********************************************************/
INVITE_received* INVITE_received::instance = 0;

INVITE_received *INVITE_received::Instance() {
      if(instance == 0)
            instance = new INVITE_received();
      return instance;
}

void INVITE_received::SendInvite(SIPConnection *c, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) {

      // forward INVITE message to the IMS, adding NAS context (RNC)
      string newsip,newsip1,newsip2,temp;
      ostringstream o;

      newsip.assign(inter_thread_info.sipmes);
      newsip1 = newsip.substr(0,newsip.find("Via:", 0));
      newsip2 = newsip.substr(newsip.find("Via:", 0));
      o << inter_thread_info.function_info->rnc_gw;                        // insert NAS context => construct SIP-RAN message
      temp = "RNC_GW-ID: "+o.str();
      newsip1.append(temp+"\n");
      newsip1.append(newsip2);

      inter_thread_info.function_info->length = newsip1.length();

      RawSockUtilities::Instance().Sendto(inter_thread_info.recvbuff, (char *)newsip1.c_str(), inter_thread_info.function_info, inter_thread_info.dev);
      ChangeState(c, RAB_Assignment_completed::Instance());
}

// override base class functions
void INVITE_received::SendRegister(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void INVITE_received::RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { cout << "State 5" << endl; }
void INVITE_received::SendAck(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void INVITE_received::Send200_OK(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void INVITE_received::RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }


/*****************************************************************
```

```
*   RAB_Assignment_completed class member-function definition    *
*****************************************************************/
RAB_Assignment_completed* RAB_Assignment_completed::instance = 0;


RAB_Assignment_completed *RAB_Assignment_completed::Instance() {
	if(instance == 0)
		instance = new RAB_Assignment_completed();
	return instance;
}

void RAB_Assignment_completed::Send200_OK(SIPConnection *c, char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev) {

	// simply forward 200 OK message without NAS context
	RawSockUtilities::Instance().Sendto(recvbuff, sipmes, function_info, dev);
	ChangeState(c, Connected::Instance());
}

// override base class functions
void RAB_Assignment_completed::SendRegister(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void RAB_Assignment_completed::RABAssReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { cout << "State 6" << endl; }
void RAB_Assignment_completed::SendAck(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
void RAB_Assignment_completed::SendInvite(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { cout << "HELLO_6" << endl; }
void RAB_Assignment_completed::RelocReq(SIPConnection*, char *, char *, Sip_parsed_msg *, struct sockaddr_ll *) { }
```

Αρχείο: rawsockutils.h

```
/***************************************************************************
                          rawsockutils.h  -  description
                          -------------------
    begin                : Tue Mar 2 2004
    copyright            : (C) 2004 by enikas
    email                : enikas@siva
 ***************************************************************************/

#ifndef RAWSOCKUTILITIES_H
#define RAWSOCKUTILITIES_H

#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <net/if.h>
#include <linux/if_ether.h>
#include <linux/if_packet.h>
#include <features.h>
#include <asm/types.h>
#include <sys/ioctl.h>

#include <signal.h>
#include <sys/signal.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

#include "sailorCommonTypes.h"
#include "rawsocktypes.h"

#define MAC_ADDR_LEN              6
#define IP_ADDR_LEN               20
#define ARPHRD_ETHER              1

//=============================================================================================================
const unsigned char ims_eth[MAC_ADDR_LEN] =  {0x00, 0x0F, 0x1F, 0x0F, 0x7C, 0x46};              // IMS eth MAC address
const unsigned char rasn_eth[MAC_ADDR_LEN] = {0x00, 0x11, 0x6B, 0x50, 0x06, 0x21};              // RASN eth MAC address -- for RASN to RASN connection
const unsigned char fixedclient_eth[MAC_ADDR_LEN] = {0x00, 0x0E, 0xA6, 0x5A, 0x6B, 0x50};      // MT eth MAC address
//=============================================================================================================

const unsigned char rnc1_eth[MAC_ADDR_LEN] = {0x00, 0x50, 0xDA, 0xCF, 0x5E, 0xD8};
const unsigned char rnc2_eth[MAC_ADDR_LEN] = {0x00, 0x50, 0xDA, 0xCF, 0x5E, 0xD8};
const unsigned char rnc3_eth[MAC_ADDR_LEN] = {0x00, 0x50, 0xDA, 0xCF, 0x5E, 0xD8};
const unsigned char rnc4_eth[MAC_ADDR_LEN] = {0x00, 0x50, 0xDA, 0xCF, 0x5E, 0xD8};
const unsigned char rnc5_eth[MAC_ADDR_LEN] = {0x00, 0x50, 0xDA, 0xCF, 0x5E, 0xD8};
const unsigned char rnc6_eth[MAC_ADDR_LEN] = {0x00, 0x50, 0xDA, 0xCF, 0x5E, 0xD8};
const unsigned char rnc7_eth[MAC_ADDR_LEN] = {0x00, 0x50, 0xDA, 0xCF, 0x5E, 0xD8};

#define RASN_SERVER_PORT          5500
#define RASN1_LOCAL_PORT          5510
#define RNC_PORT_1                5501
#define RNC_PORT_2                5502
#define RNC_PORT_3                5503
#define RNC_PORT_4                5504
#define RNC_PORT_5                5505
#define RNC_PORT_6                5506
#define RNC_PORT_7                  5507

#define IMS_IP_ADDRESS            "192.168.100.2"
#define IMS_LISTEN_PORT           9054
#define CLIENT_LISTEN_PORT        5060

#define SUCCESS                                         1
#define FAIL                                            0

using namespace RawSockTypes;
```

```
using namespace SailorTypes;

class RawSockUtilities
{
public:
protected:
                                        RawSockUtilities();
                                        ~RawSockUtilities();

public:
                                        static RawSockUtilities& Instance() { return (*instance_); }
                                        static int          create_raw_socket();
                                        static void         set_sockaddr_ll_values(struct sockaddr_ll &, struct ifreq *);
                                        static void         set_dst_mac_addr(struct sockaddr_ll &, int);
                                        static int          set_if_index(const char *, int, struct ifreq *);
                                        static int          set_if_in_promisc_mode(const char *, int, struct ifreq *);
                                        static int          bind_raw_socket(int, struct sockaddr_ll &, struct ifreq *);
                                        static void         print_ip_info(const struct ip_packet *);
                                        static void         print_tcp_info(const struct tcp_packet *);
                                        static void         print_nw_info(const struct ip_packet *, const struct udp_packet *);
                                        static void         cksum(u_short *);
                                        static bool         do_ipaddr_match(unsigned long, unsigned long);
                                        static bool         do_ipaddr_match(char *, char *);
                                        static bool         compare_with_source_ip(const struct ip_packet *, char *);
                                        static bool         compare_with_dest_ip(const struct ip_packet *, char *);
                                        static int          obt_num_ip(char* ip_cad,u_char* ip_num);
                                        static int          match_ipadd(char *ipcad, u_char *ipnum2);
                                        static int          inet_ntoa2(u_long, char *);
                                        static void         Sendto( char *, char *, Sip_parsed_msg *, struct sockaddr_ll *);

private:
                                        static RawSockUtilities* instance_;
};
#endif
```

Αρχείο: rawsockutils.cpp

```
/***************************************************************************
                          rawsockutils.cpp  -  description
                          -------------------
    begin                : Tue Mar 2 2004
    copyright            : (C) 2004 by enikas
    email                : enikas@siva
 ***************************************************************************/

#include "rawsockutils.h"

RawSockUtilities* RawSockUtilities::instance_;
//-------------------------------------------------------------------------
RawSockUtilities::RawSockUtilities(){}
//-------------------------------------------------------------------------
RawSockUtilities::~RawSockUtilities(){}
//-------------------------------------------------------------------------
int RawSockUtilities::inet_ntoa2(u_long n_ip_addr, char * a_ip_addr)
{
     memset((void *)a_ip_addr, '\0', IP_ADDR_LEN);
     struct in_addr ip_addr;
     ip_addr.s_addr = n_ip_addr;
     char * temp = (char *)malloc(IP_ADDR_LEN);
     temp = inet_ntoa(ip_addr);
     strncpy(a_ip_addr, temp, strlen(temp));
     return SUCCESS;
}
//-------------------------------------------------------------------------
bool RawSockUtilities::compare_with_source_ip(const struct ip_packet * ip_pkt, char * ip)
{
    //char * ip1 = (char *)malloc(IP_ADDR_LEN);
    //inet_ntoa2((u_long)ip_pkt->ip_source, ip1);

     u_long ip1 = ip_pkt->ip_source;
     u_long ip2 = inet_addr(ip);
    //printf("\ncompare_with_src_ip::ADDR1 = %s\n", ip1);
    //printf("compare_with_src_ip::ADDR2 = %s\n", ip);

    return do_ipaddr_match(ip1, ip2);
}
//-------------------------------------------------------------------------
bool RawSockUtilities::compare_with_dest_ip(const struct ip_packet * ip_pkt, char * ip)
{
    //char * ip1 = (char *)malloc(IP_ADDR_LEN);
    //inet_ntoa2((u_long)ip_pkt->ip_dest, ip1);

    u_long ip1 = ip_pkt->ip_dest;
    u_long ip2 = inet_addr(ip);
    //printf("\ncompare_with_dst_ip::ADDR1 = %d\n", ip1);
    //printf("compare_with_dst_ip::ADDR2 = %d\n", ip2);

    return do_ipaddr_match(ip1, ip2);
}
//-------------------------------------------------------------------------
bool RawSockUtilities::do_ipaddr_match(unsigned long ipaddr1, unsigned long ipaddr2)
{
                                        if (ipaddr1 == ipaddr2)
                                             return true;
                                        else
                                             return false;
```

```
}
//---------------------------------------------------------------------------
bool RawSockUtilities::do_ipaddr_match(char * ipaddr1, char * ipaddr2)
{
   if (strlen(ipaddr1) != strlen(ipaddr2))
      return false;
   else
   {
                                          if ( strncmp(ipaddr1,ipaddr2, strlen(ipaddr2)) == 0 )
                                              return true;
                                          else
                                            return false;
   }
}
//---------------------------------------------------------------------------
void RawSockUtilities::print_nw_info(const struct ip_packet * ip, const struct udp_packet *udp)
{
                                          char * src_addr = (char *)malloc(IP_ADDR_LEN);
                                          char * dest_addr = (char *)malloc(IP_ADDR_LEN);

                                          inet_ntoa2((u_long)ip->ip_source, src_addr);
                                          inet_ntoa2((u_long)ip->ip_dest, dest_addr);

                                          printf("NW LAYER INFO : <%s:%d> -> <%s:%d>\n",
                                          src_addr, ntohs((u_short)udp->udp_source_port),
                                          dest_addr, ntohs((u_short)udp->udp_dest_port) );
}
//---------------------------------------------------------------------------
void RawSockUtilities::print_ip_info(const struct ip_packet * ip)
{
                                          char * src_addr = (char *)malloc(IP_ADDR_LEN);
                                          char * dest_addr = (char *)malloc(IP_ADDR_LEN);

                                          inet_ntoa2((u_long)ip->ip_source, src_addr);
                                          inet_ntoa2((u_long)ip->ip_dest, dest_addr);

                                          printf("IP INFO: <%s> -> <%s>\n", src_addr, dest_addr);
}
//---------------------------------------------------------------------------
void RawSockUtilities::print_tcp_info(const struct tcp_packet * tcp)
{
                                          printf("TCP INFO: <%d> -> <%d> - ", ntohs((u_short)tcp->tcp_source_port), ntohs((u_short)tcp->tcp_dest_port));
                                          //printf("TCP HEADER LEN: %d\n",   tcp->tcp_hlen );


                                          char tcp_flag[15]= "[";
                                          bool add_comma = false;
                                          if (tcp->tcp_fin !=0)
                                          {
                                               strcat(tcp_flag, "FIN");
                                               add_comma = true;
                                          }
                                          if (tcp->tcp_syn !=0)
                                          {
                                               if (add_comma)
                                                         strcat(tcp_flag, ", ");

                                               strcat(tcp_flag, "SYN");
                                               add_comma = true;
                                          }
                                          if (tcp->tcp_psh !=0)
                                          {
                                               if (add_comma)
                                                         strcat(tcp_flag, ", ");
                                               strcat(tcp_flag, "PSH");
                                               add_comma = true;
                                          }
                                          if (tcp->tcp_ack !=0)
                                          {
                                               if (add_comma)
                                               {
                                                         strcat(tcp_flag, ", ");
                                               }
                                               strcat(tcp_flag, "ACK");
                                          }
                                          printf("%s]", tcp_flag);

   printf("=============================================");
}
//---------------------------------------------------------------------------
int RawSockUtilities::create_raw_socket()
{
                                          int _sd=-1;
                                          //get the complete link-layer packet
                                          if ( (_sd = socket(PF_PACKET, SOCK_DGRAM, htons(ETH_P_ALL)) ) < 0)
                                          {
                                               perror("CREATE RAW SOCKET");
                                          }

                                          int val=1;
                                          setsockopt(_sd, IPPROTO_IP, IP_HDRINCL, &val, sizeof(val));

                                          if (_sd < 0)
                                               return FAIL;

                                          return _sd; // return the socket descriptor
}
//---------------------------------------------------------------------------
int RawSockUtilities::set_if_index(const char * _ifr_name, int sd, struct ifreq * ifr)
{
```

```
                                        int return_val = SUCCESS;

                                        strncpy((char *)ifr->ifr_name, _ifr_name, strlen(_ifr_name)+1);

                                        if(ioctl(sd, SIOCGIFINDEX, ifr) < 0)
                                        {
                                            perror("ioctl SIOCGIFINDEX");
                                            return_val = FAIL;
                                        }

                                        return return_val;
}
//---------------------------------------------------------------------------
void RawSockUtilities::set_sockaddr_ll_values(struct sockaddr_ll & device, struct ifreq * ifr)
{
                                        memset((char *) &device, '\0', sizeof(device));
                                        device.sll_family   = AF_PACKET;
                                        device.sll_protocol = htons(ETH_P_IP);
                                        device.sll_hatype   = ARPHRD_ETHER;
                                        device.sll_halen    = MAC_ADDR_LEN;
                                        device.sll_ifindex  = ifr->ifr_ifindex;
                                        //device.sll_pkttype                    = 0;//PACKET_OTHERHOST;
}
//---------------------------------------------------------------------------
void RawSockUtilities::set_dst_mac_addr(struct sockaddr_ll & device, int dest)
{
   //printf("DESTINATION =  %d(%s)\n", dest, ModuleIDType_STR[dest]);
   // CLIENT -> IMS
   if (dest == IMS)
      for (int i=0;i<MAC_ADDR_LEN;i++)
         device.sll_addr[i] = ims_eth[i];
   // IMS - > CLIENT (one of the RNCs)
   else if (dest == RNC1)
      for (int i=0;i<MAC_ADDR_LEN;i++)
         device.sll_addr[i] = rnc1_eth[i];
   else if (dest == RNC2)
      for (int i=0;i<MAC_ADDR_LEN;i++)
         device.sll_addr[i] = rnc2_eth[i];
   else if (dest == RNC3)
      for (int i=0;i<MAC_ADDR_LEN;i++)
         device.sll_addr[i] = rnc3_eth[i];
   else if (dest == RNC4)
      for (int i=0;i<MAC_ADDR_LEN;i++)
         device.sll_addr[i] = rnc4_eth[i];
   else if (dest == RNC5)
      for (int i=0;i<MAC_ADDR_LEN;i++)
         device.sll_addr[i] = rnc5_eth[i];
   else if (dest == RNC6)
      for (int i=0;i<MAC_ADDR_LEN;i++)
         device.sll_addr[i] = rnc6_eth[i];
   else if (dest == RNC7)
      for (int i=0;i<MAC_ADDR_LEN;i++)
         device.sll_addr[i] = rnc7_eth[i];
   // RASN <-> RASN
   else if (dest == RASN)
      for (int i=0;i<MAC_ADDR_LEN;i++)
         device.sll_addr[i] = rasn_eth[i];
   // RASN - > CLIENT
   else if (dest == MT)
      for (int i=0;i<MAC_ADDR_LEN;i++)
         device.sll_addr[i] = fixedclient_eth[i];
   else
   {
      printf("DESTINATION =  %d(%s)", dest, ModuleIDType_STR[dest]);
      fflush(stdout);
      assert(false);
   }
}
//---------------------------------------------------------------------------
int RawSockUtilities::set_if_in_promisc_mode(const char * _ifr_name, int sd, struct ifreq * ifr)
{
                                        int return_val = SUCCESS;

                                        strncpy((char *)ifr->ifr_name, _ifr_name, strlen(_ifr_name)+1);

                                        if ( ( ioctl(sd, SIOCGIFFLAGS, ifr) ) < 0)
                                        {
                                            perror("ioctl SIOCGIFFLAGS");
                                            return_val = FAIL;
                                        }

                                        ifr->ifr_flags |= IFF_PROMISC;

                                        if ( (ioctl(sd, SIOCSIFFLAGS, ifr)) < 0)
                                        {
                                            perror("ioctl SIOCSIFFLAGS");
                                            return_val = FAIL;
                                        }

                                        printf("Interface %s set in promiscuous mode\n", ifr->ifr_name);

                                        return return_val;

}
//---------------------------------------------------------------------------
int                                     RawSockUtilities::bind_raw_socket(int sd, struct sockaddr_ll & device, struct ifreq * ifr)
{
                                        if ( bind(sd, (struct sockaddr *)&device, sizeof(device)) < 0 )
                                        {
                                            perror("BIND RAW SOCKET");
```

```
                                        return FAIL;
                                    }

                                    //  printf("PF_PACKET socket listening on interface %s (index %d) ", ifr->ifr_name, ifr->ifr_ifindex);

                                    return SUCCESS;

}
//-------------------------------------------------------------------------
void RawSockUtilities::cksum(u_short * pdata)
{
  u_char *                          ptr;
  int                                       lon_cab, i, count;
  unsigned short                    checksum;
  register long                     sum = 0;

  ptr                               = (u_char *)pdata;
  lon_cab                           = (ptr[0] & 15)*4;
  count                             = lon_cab - 2;

  for(i=0; i<5; i++)
    sum+= *pdata++;

  pdata++;
  count-=10;

  while(count > 1)
  {
    sum+= *pdata++;
    count-=2;
  }

  if(count)
    sum+= *(unsigned char *) pdata;
  while(sum>>16)
    sum=(sum & 0xffff)+(sum>>16);

  checksum = ~sum;
  memcpy(ptr + 10, &checksum, 2);
}
//-------------------------------------------------------------------------
int RawSockUtilities::obt_num_ip(char* ip_cad,u_char* ip_num)
{
 char *ptr,*aux2;
 char aux[80];
 int c,i;

    strcpy(aux,ip_cad);
    aux2=aux;

    if( (ptr=(char *)strstr(aux2, "."))!=NULL)
    {
                                    *ptr='\0';
                                    sscanf(aux2,"%d",&c);
                                     if( c<0 || c>255)
                                    return 0;
                                    ip_num[0]=c;
                                     aux2=ptr+1;

                                     for(i=1;i<=2;i++)
                                     {
                                      if( (ptr=(char *)strstr(aux2, "."))==NULL)
                                    return 0;
                                      *ptr='\0';
                                      sscanf(aux2, "%d", &c);
                                      if( c<0 || c>255)
                                       return 0;
                                      ip_num[i]=c;
                                    aux2=ptr+1;
                                    }
                                    sscanf(aux2, "%d", &c);
                                    if( c<0 || c>255)
                                     return 0;
                                    ip_num[3]=c;

    }
    else return 0;
    return 1;

}
//-------------------------------------------------------------------------
int RawSockUtilities::match_ipadd(char *ipcad, u_char *ipnum2)
{
  int i;
  unsigned char ipnum[4];
  obt_num_ip(ipcad,ipnum);
  for( i=0; i<4; i++)
  {
    if(ipnum[i] != ipnum2[i])
    return 0;
  }
  return 1;
}
//-------------------------------------------------------------------------
void RawSockUtilities::Sendto( char *recvbuff, char *sipmes, Sip_parsed_msg *function_info, struct sockaddr_ll *dev)
{

      struct ip_packet *ip;
      struct udp_packet *udp;
      socklen_t addrlen;
```

```
    // change the total length of the whole datagram including IP and UDP headers
     ip = (struct ip_packet *)recvbuff;
     ip->ip_total_length = htons(function_info->length+28);

     bzero(recvbuff+28,function_info->bytes_read-28);
     //for(int i=28; i<fromMT; i++) recvbuff[i] = '\0';
     memcpy((void *)(recvbuff+28), (void *)sipmes, function_info->length);
     // evaluating checksum
     cksum((u_short *)recvbuff);

     //printf("THIS IS MESSAGE:\n%s\n",recvbuff+28);
     udp = (struct udp_packet *)(recvbuff+20);
     udp->udp_len = htons(function_info->length+8);
     udp->udp_cksum = 0x00;

     addrlen = sizeof(struct sockaddr_ll);
     if( sendto(function_info->rawfd, recvbuff, function_info->length+28, 0, (struct sockaddr *) dev, (socklen_t)addrlen) <= 0) {
                                        perror("SEND TO");
          exit(0);
     }
}
```

Αρχείο: rawsocktypes.h

```
/***************************************************************************
                    rawsocktypes.h  -  description
                    ------------------
   begin            : Tue Mar 2 2004
   copyright        : (C) 2004 by enikas
   email            : enikas@siva
 ***************************************************************************/

#ifndef RAWSOCKTYPES_H
#define RAWSOCKTYPES_H

namespace RawSockTypes
{
                                        struct ip_packet
                                        {
                                             unsigned int    ip_length:4;       // byte  0 (1/2)
                                             unsigned int    ip_version:4;      // byte  0 (1/2)
                                             unsigned char   ip_tos;            // byte  1
                                             unsigned short  ip_total_length;   // byte  2-3
                                             unsigned short  ip_id;             // byte  4-5
                                             unsigned short  ip_flags;          // byte  6-7
                                             unsigned char   ip_ttl;            // byte  8
                                             unsigned char   ip_protocol;       // byte  9
                                             unsigned short  ip_cksum;          // bytes 10-11
                                             unsigned long   ip_source;         // bytes 12-13-14-15
                                             unsigned long   ip_dest;           // bytes 16-17-18-19
                                        };

                                        struct tcp_packet
                                        {
                                             unsigned short  tcp_source_port;
                                             unsigned short  tcp_dest_port;
                                             unsigned int    tcp_seqno;
                                             unsigned int    tcp_ackno;
                                             unsigned int    tcp_res1:4,
                                                             tcp_hlen:4,
                                                             tcp_fin:1,
                                                             tcp_syn:1,
                                                             tcp_rst:1,
                                                             tcp_psh:1,
                                                             tcp_ack:1,
                                                             tcp_urg:1,
                                                             tcp_res2:2;
                                             unsigned short  tcp_winsize;
                                             unsigned short  tcp_cksum;
                                             unsigned short  tcp_urgent;
                                        };

                                        struct udp_packet
                                        {
                                             unsigned short  udp_source_port;
                                             unsigned short  udp_dest_port;
                                              unsigned short  udp_len;
                                             unsigned short  udp_cksum;

                                        };
};
#endif
```

Αρχείο: sailorCommonTypes.h

```
/***************************************************************************
                    sailorCommonTypes.h  -  description
                    ------------------
   begin            : Tue Mar 2 2004
   copyright        : (C) 2004 by enikas
```

```
   email                : enikas@siva
 ********************************************************************************/

#ifndef SAILORCOMMONTYPES_H
#define SAILORCOMMONTYPES_H

#define RASN_ID                         0
#define N_TERMINALS                     10              // Number of terminal nodes
#define SESSIONS_PER_TERMINAL           5               // Number of terminal nodes
#define N_RASNS                         2       // Number of RASN nodes
#define N_RNC_GWS                       7                       // Number of RNC/GW nodes
#define MAX_RABs                        10                      // Number of RABs

#define                         RASN_SERVER_PORT        5500
#define                         RASN2_PORT              5510
#define                         RNC_PORT_1              5501
#define                         RNC_PORT_2              5502
#define                         RNC_PORT_3              5503
#define                         RNC_PORT_4              5504
#define                         RNC_PORT_5              5505
#define                         RNC_PORT_6              5506
#define                         RNC_PORT_7              5507

#define IMS_SIGNALLING_PORT     6544

#define SIP_UA_1                "sip:pavel.dostal@umts.xnet"
#define SIP_UA_2                "sip:pavel2.dostal@umts.xnet"
//#define SIP_UA_3              "sip:te3@sailor.net"

#define IP_POOL_SIZE                    2
//#define RASN_IPPOOL_1                 "192.168.100.110"
//#define RASN_IPPOOL_2                 "192.168.100.111"

#define SIP_MES_NUM                     8       // Number of SIP messages


namespace SailorTypes
{
   typedef enum forward_type
   {
      TCP_SOCK = 0,
      RAW_SOCK = 1
   };

   const char forward_type_STR[][20] =
   {
      "TCP_SOCK",
      "RAW_SOCK"
   };

   typedef enum ho_type
   {
      INTRA_RASN = 0,
      INTER_RASN = 1
   };

   const char ho_type_STR[][20] =
   {
      "INTRA_RASN",
      "INTER_RASN"
   };

   enum ModuleIDType
   {
      RASN    = 0,
      IMS     = 1,
      SIP_UA  = 2,
      ANE     = 3,
      RNC1    = 4,
      RNC2    = 5,
      RNC3    = 6,
      RNC4    = 7,
      RNC5    = 8,
      RNC6    = 9,
      RNC7    = 10,
      MT = 11
   };

   const char ModuleIDType_STR[][20] =
   {
                                        "RASN",
                                        "IMS",
                                        "SIP_UA",
                                        "ANE",
                                        "RNC1",
                                        "RNC2",
                                        "RNC3",
                                        "RNC4",
                                        "RNC5",
                                        "RNC6",
                                        "RNC7"
   };

   enum boolean_data
   {
       OK     = 0,
       FAILED = 1
   };

   enum SIP_msg
```

```
  {
    REGISTER   = 10,
    INVITE     = 11,
    _200_OK    = 12,
    _401_ANAUTH = 13,
    ACK        = 14,
    CORE       = 15,
    CORE_OK    = 16,
    CORE_ACK   = 17
  };

  const char SIP_msg_STR[][40] =
  {
                                        "REGISTER",
                                        "INVITE",
                                        "200_OK",
                                        "401_ANAUTH",
                                       "ACK",
                                        "CORE",
                                        "CORE_OK",
                                        "CORE_ACK"
  };


  enum RANAP_msg
  {
                                        RAB_ASSIGNMENT_REQUEST  = 0,
                                        RAB_ASSIGNMENT_RESPONSE = 1,
                                        RELOCATION_REQUEST      = 2,
                                        RELOCATION_REQUEST_ACK  = 3,
                                        RELOCATION_COMMAND      = 4,
                                        RELOCATION_COMPLETE     = 5,
                                        Iu_RELEASE_COMMAND      = 6,
                                        Iu_RELEASE_COMPLETE     = 7,
                                        INITIAL_TE_ADDRESS      = 8
  };

  const char RANAP_msg_STR[][40] =
  {
                                        "RAB_ASSIGNMENT_REQUEST",
                                        "RAB_ASSIGNMENT_RESPONSE",
                                        "RELOCATION_REQUEST",
                                        "RELOCATION_REQUEST_ACK",
                                        "RELOCATION_COMMAND",
                                        "RELOCATION_COMPLETE",
                                        "Iu_RELEASE_COMMAND",
                                        "Iu_RELEASE_COMPLETE",
                                        "INITIAL_TE_ADDRESS"
  };

// RRC info
  enum Client_ANE_msg                           // Type of messages carried by RRC info
  {
                                        REL_TRIG   = 0,
                                        UTRAN_AVAI = 1
  };

  enum UTRAN_status
  {
                                        UNAVAILABLE = 0,
                                        AVAILABLE   = 1,
                                        NA          = 2
  };

  struct RRC_info
  {
                                        enum Client_ANE_msg msgID;          // either REL_TRIG or UTRAN_AVAI
                                        int mtID;                           // from 1 to N_TERMINALS
                                        int rasnID;                         // from 1 to N_RASNS; -1 if not applicable
                                        int rnc_gwID;                       // from 1 to N_RNC_GWS; -1 if not applicable
                                        enum UTRAN_status utran_status;
  };

//-----------------------------------------------------------------

// RAB assigment request
                                    struct RABdef
                                    {
                                        int RABID;                          // from 1 to MAX_RABs
                                        int RAB_delay;                      // QoS parameter
                                        enum boolean_data forward_RAB;      // RAB subject to data forwarding in relocation
                                    };

                                    struct RAB_Assign_Req
                                    {
                                        enum RANAP_msg msgID;
                                        int mtID;                                       // from 1 to N_TERMINALS
                                        struct RABdef add_rabs[MAX_RABs];               // from 1 to MAX_RABs
                                        struct RABdef del_rabs[MAX_RABs];               // from 1 to MAX_RABs
                                    };

                                    // RAB assigment response
                                    struct RABrep
                                    {
                                        int RABID;                                       // from 1 to MAX_RABs
                                        enum boolean_data result;
                                    };

                                    struct RAB_Assign_Resp
                                    {
```

```
                                    enum RANAP_msg msgID;
                                    int mtID;                                   // from 1 to N_TERMINALS
                                    struct RABrep add_rabs[MAX_RABs];           // from 1 to MAX_RABs

                                    struct RABrep del_rabs[MAX_RABs];           // from 1 to MAX_RABs
                                };

                                // Initial TE Address
                                struct Init_TE_Address
                                {
                                    enum RANAP_msg msgID;
                                    int  mtID;                                  // from 1 to N_TERMINALS;
                                    char newIP[16];
                                };


                                // Relocation Request
                                struct Reloc_Req
                                {
                                    enum RANAP_msg msgID;
                                    int mtID;                                   // from 1 to N_TERMINALS
                                    char newIP[16];
                                    struct RABdef setup_rabs[MAX_RABs];         // from 1 to MAX_RABs
                                };

                                // Relocation Request Acknowledge
                                struct Reloc_Req_Ack
                                {
                                    enum RANAP_msg msgID;
                                    int mtID;                                   // from 1 to N_TERMINALS
                                    struct RABrep setup_rabs[MAX_RABs];         // from 1 to MAX_RABs
                                };

                                // Relocation Command
                                struct Reloc_Command
                                {
                                    enum RANAP_msg msgID;
                                    int mtID;                                   // from 1 to N_TERMINALS
                                    int del_RAB[MAX_RABs];                      // from 1 to MAX_RABs
                                    int rasnID;                         // from 1 to N_RASNS; -1 if controlling RASN (intra-RASN relocation)
                                };

                                // Relocation Complete
                                struct Reloc_Complete
                                {
                                    enum RANAP_msg msgID;
                                    int mtID;                                   // from 1 to N_TERMINALS
                                };

                                // Iu Release Command
                                struct Iu_Release_Command
                                {
                                    enum RANAP_msg msgID;
                                    int mtID;                                   // from 1 to N_TERMINALS
                                };

                                // Iu Release Complete
                                struct Iu_Release_Complete
                                {
                                    enum RANAP_msg msgID;
                                    int mtID;                                   // from 1 to N_TERMINALS
                                    enum boolean_data result;
                                };

// ----------------------------------------
// SIP_ran Register information for RASN
// ----------------------------------------

                                struct Sip_ran_Register
                                {
                                    int rnc_gwID;                       // target RNC/GW; from 1 to N_RNC_GWS; -1 if not applicable
                                    int oldrasnID;                      // from 1 to N_RASNS
                                };

                                struct Sip_parsed_msg
                                {
                                    enum SIP_msg msg;                   // type of message
                                    bool contact;                       // contact field is present => true, else false (Inital Registration)
                                    enum SIP_msg cseq;                  // type of CSeq file
                                    int rnc_gw;                         // RNC gateway id
                                    int length;                         // SIP message length
                                    int from;                           // identifies mobile terminal id
                                    int bytes_read;                     // number of bytes read from raw socket
                                    int rawfd;                          // raw socket descriptor
                                    int new_tcpfd;                      // new RASN tcp socket descriptor
                                    int old_tcpfd;                      // old RASN tcp socket descriptor
                                };

};
#endif
```

Αρχείο: rnc.h

```
#include <sys/types.h>
#include <sys/socket.h>
#include <time.h>
```

```
#include <sys/time.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include "sailorCommonTypes.h"

#define BUFSIZE                                                 1024

#define                 LOCAL_ADDR                      "127.0.0.1"
#define                 RASN0_REMOTE_ADDR               "192.168.100.200"
#define                 RASN1_REMOTE_ADDR               "127.0.0.1"
#define                 REMOTE_PORT                     5500
```

Αρχείο: rnc.cpp

```
#include "rnc.h"

using namespace SailorTypes;

int main(int argc, char ** argv)
{

                                        if (argc != 2)
                                        {
                                            printf("USAGE: %s <RNC_NUMBER> from 1..7\n", argv[0]);
                                            exit(0);
                                        }

                                        int rnc_number = atoi(argv[1]), rasn, local_port;

                                        switch(rnc_number)
                                        {
                                            case 1:
                                                        rasn = 0;
                                                        local_port = 5501;
                                                        printf("RNC1");
                                            break;
                                            case 2:
                                                        rasn = 0;
                                                        local_port = 5502;
                                                        printf("RNC2");
                                            break;
                                            case 3:
                                                        rasn = 0;
                                                        local_port = 5503;
                                                        printf("RNC3");
                                            break;
                                            case 4:
                                                        rasn = 1;
                                                        local_port = 5504;
                                                        printf("RNC4");
                                            break;
                                            case 5:
                                                        rasn = 1;
                                                        local_port = 5505;
                                                        printf("RNC5");
                                            break;
                                            case 6:
                                                        rasn = 1;
                                                        local_port = 5506;
                                                        printf("RNC6");
                                            break;
                                            case 7:
                                                        rasn = 1;
                                                        local_port = 5507;
                                                        printf("RNC7");
                                            break;
                                            default:
                                                        printf("USAGE: %s <RNC_NUMBER> from 1..7\n", argv[0]);
                                                        exit(0);
                                        }

                                        int sd, nbytes;
                                        char buf[BUFSIZE];
                                        struct sockaddr_in server;
                                        struct sockaddr_in sin;

                                        if ( (sd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
                                        {
                                            perror("SOCKET");
                                            exit(1);
                                        }

                                        int enable = 1;
                                        setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int));

                                        sin.sin_family = AF_INET;
                                        sin.sin_addr.s_addr = inet_addr(LOCAL_ADDR);
```

```
sin.sin_port = htons((u_short)local_port);
memset(&(sin.sin_zero), '\0', 8);
if ( bind(sd, (struct sockaddr *)&sin, sizeof(sin)) < 0 )
{
    perror("BIND");
    exit(1);
}

server.sin_family = AF_INET;
if (rasn == 0)
    server.sin_addr.s_addr = inet_addr(RASN0_REMOTE_ADDR);
if (rasn == 1)
    server.sin_addr.s_addr = inet_addr(RASN1_REMOTE_ADDR);
server.sin_port = htons((u_short)REMOTE_PORT);
memset(&(server.sin_zero), '\0', 8);
if ( connect(sd, (struct sockaddr *)&server, sizeof(server)) < 0 )
{
    perror("CONNECT");
    exit(1);
}

printf(" CONNECTED TO <%s:%d>\n\n", inet_ntoa(server.sin_addr), REMOTE_PORT);

int msg_type, index;
while (1)
{
    index = 0;

    if ( (nbytes =read(sd, buf, BUFSIZE)) <=0 )
    {
            if (nbytes == 0)
                    puts("CONNECTION LOST");
            else
                    perror("READ");

            exit(0);
    }

    memcpy((char *)&msg_type, buf, sizeof(msg_type));

    switch(msg_type)
    {
            case INITIAL_TE_ADDRESS:
                    printf("RECEIVED INITIAL_TE_ADDRESS %d BYTES\n", nbytes);
                    continue;

            break;
            case RELOCATION_REQUEST:
                    printf("RECEIVED RELOCATION_REQUEST %d BYTES\n", nbytes);

                    struct Reloc_Req reloc_req;
                    memcpy((char *)&reloc_req, buf, sizeof(reloc_req));
                    index += sizeof(reloc_req);

                    struct Reloc_Req_Ack reloc_req_ack;
                    reloc_req_ack.msgID = RELOCATION_REQUEST_ACK;
                    reloc_req_ack.mtID = reloc_req.mtID;
                    reloc_req_ack.setup_rabs[0].RABID = reloc_req.setup_rabs[0].RABID;
                    reloc_req_ack.setup_rabs[0].result = OK;

                    reloc_req_ack.setup_rabs[1].RABID = 0;
                    reloc_req_ack.setup_rabs[1].result = OK;


                    memcpy( (char *)buf, (char *)&reloc_req_ack, sizeof(reloc_req_ack) );

            break;
            case RELOCATION_COMMAND:
                    printf("RECEIVED RELOCATION_COMMAND %d BYTES\n", nbytes);

                    struct Reloc_Command reloc_command;
                    memcpy((char *)&reloc_command, buf, sizeof(reloc_command));
                    index += sizeof(reloc_command);

                    struct Reloc_Complete reloc_complete;
                    reloc_complete.msgID = RELOCATION_COMPLETE;
                    reloc_complete.mtID = reloc_command.mtID;

                    memcpy((char *)buf, (char *)&reloc_complete, sizeof(reloc_complete));

                    break;
            case Iu_RELEASE_COMMAND:
                    printf("RECEIVED Iu_RELEASE_COMMAND %d BYTES\n", nbytes);

                    struct Iu_Release_Command iu_release_command;
                    memcpy((char *)&iu_release_command, buf, sizeof(iu_release_command));
                    index += sizeof(iu_release_command);

                    struct Iu_Release_Complete iu_rel_complete;
                    iu_rel_complete.msgID = Iu_RELEASE_COMPLETE;
                    iu_rel_complete.mtID = iu_release_command.mtID;
                    iu_rel_complete.result = OK;

                    memcpy((char *)buf, (char *)&iu_rel_complete, sizeof(iu_rel_complete));

                    break;
            case RAB_ASSIGNMENT_REQUEST:
                    printf("RECEIVED RAB_ASSIGNMENT_REQUEST %d BYTES\n", nbytes);

                    struct RAB_Assign_Req rab_assign_req;
```

```
                                                memcpy((char *)&rab_assign_req, buf, sizeof(rab_assign_req));
                                                index += sizeof(rab_assign_req);

                                                struct RAB_Assign_Resp rab_assign_resp;

                                                rab_assign_resp.msgID = RAB_ASSIGNMENT_RESPONSE;

                                                rab_assign_resp.mtID = rab_assign_req.mtID;
                                                rab_assign_resp.add_rabs[0].RABID = rab_assign_req.add_rabs[0].RABID;
                                                rab_assign_resp.add_rabs[0].result = OK;
                                                rab_assign_resp.add_rabs[1].RABID = 0;
                                                rab_assign_resp.add_rabs[1].result = OK;
                                                rab_assign_resp.del_rabs[1].RABID = 0;
                                                rab_assign_resp.del_rabs[1].result = OK;

                                                memcpy((char *)buf, (char *)&rab_assign_resp, sizeof(rab_assign_resp));

                                                break;
                                        default:
                                                assert(false);
                                }

                                //sleep(1);
                                if ( (nbytes=write(sd, buf, index)) != index )
                                {
                                        perror("SEND");
                                }

                                printf("SENT REPLY %d BYTES\n\n", nbytes);
                        }
                        return 0;
}
```

Αρχείο: Makefile

```
CC=g++
CFLAGS=-Wall -g

all:
        $(CC) $(CFLAGS) rasn.cpp rawsockutils.cpp parser1.cpp patterns.cpp -o rasn -lpthread
clean:
        rm -f *.o rasn
```

Παράρτημα Β – Πρωτόκολλο Έναρξης Συνόδου (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) καθώς 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).

Στην τρέχουσα υποενότητα θα γίνει μια αναλυτική αναφορά στις οντότητες του SIP, στα μηνύματα που ανταλάσσονται μεταξύ αυτών των οντοτήτων και στα πεδία επικεφαλίδας αυτών των μηνυμάτων. Σκοπός μας είναι η εξοικείωση του αναγνώστη με αυτές τις βασικές έννοιες του Πρωτοκόλλου Έναρξης Συνόδου.

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 έναρξη 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 κτλ.

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 το 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.

Μηνύματα του Πρωτοκόλλου Έναρξης Συνόδου (SIP Messages)

Υλοποιούνται δύο είδη μηνυμάτων στο Πρωτόκολλο Έναρξης Συνόδου:

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

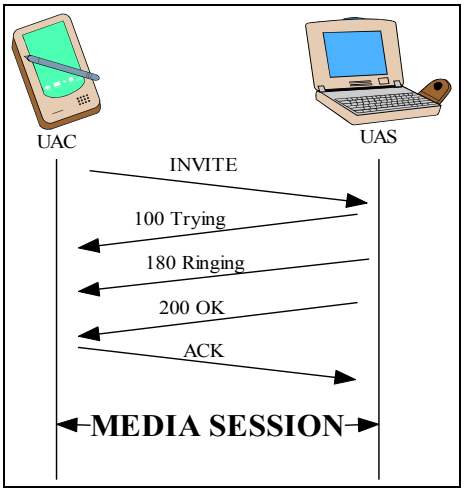
Μηνύματα Αίτησης

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. Για τους σκοπούς της παρούσας διπλωματικής εργασίας θα αναφέρουμε μόνο 5 από τα 13 μηνύματα αίτησης, αυτά που εμπλέκονται στην κινητικότητα 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 28.



Σχήμα 28: 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 τη περίπτωση έναρξης μιας συνόδου, ένα μήνυμα INVITE αποστέλνεται στο μέσο μιας κλήσης (mid-call) μεταξύ δύο πρακτόρων χρήστη από το 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 3.

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

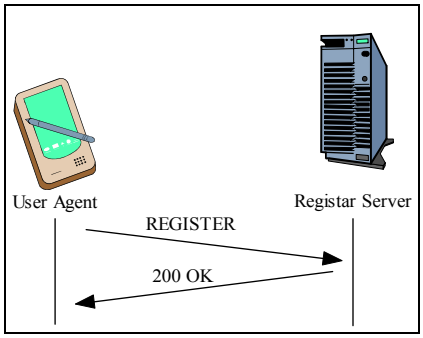
Πίνακας 3: Υποχρεωτικά πεδία του μηνύματος INVITE.

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 ένα 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 29.



Σχήμα 29: Μήνυμα REGISTER.

Τα υποχρεωτικά πεδία του μηνύματος REGISTER φαίνονται στον πιο κάτω πίνακα 4.

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

Πίνακας 4: 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 στον πιο κάτω πίνακα 5.

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

Πίνακας 5: 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 6.

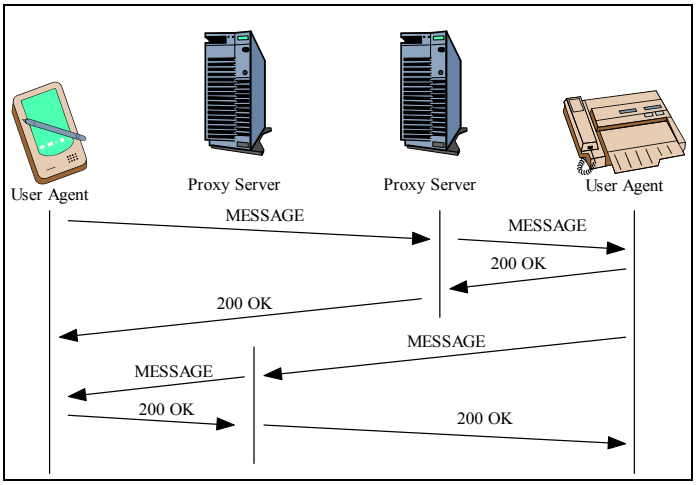
Υποχρεωτικά πεδία του μηνύματος ACK
Call-ID
CSeq
From
To
Via
Max-Forwards

Πίνακας 6: 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 30.



Σχήμα 30: Μήνυμα MESSAGE.

Τα υποχρεωτικά πεδία του μηνύματος MESSAGE φαίνονται στον πίνακα 7.

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

Πίνακας 7: Υποχρεωτικά πεδία του μηνύματος MESSAGE.

Μηνύματα Απόκρισης

Τα μηνύματα απόκρισης είναι μηνύματα που παράγονται από ένα πράκτορα χρήστη εξυπηρετητή (UAS) ή ένα 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 8.

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 8: 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 (πχ κουδούνισμα) έχει ενεργοποιηθεί για να ενημερώσει τον χρήστη.

Μηνύματα Επιτυχούς Έκβασης – 2xx

- Μήνυμα 200 OK

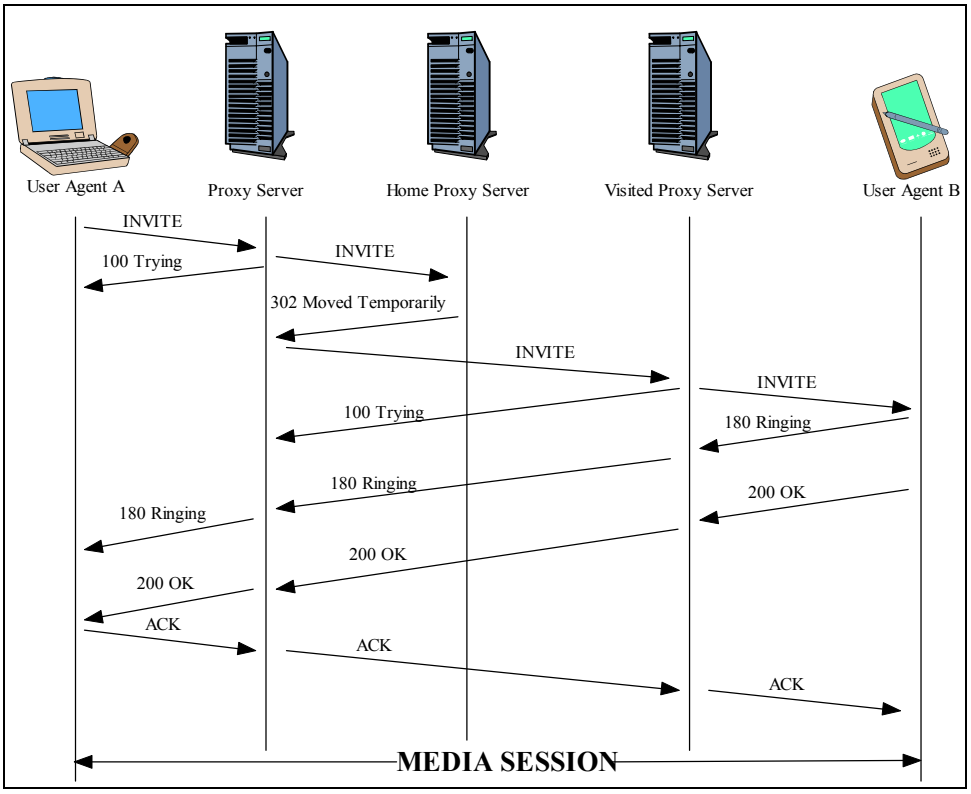
Το μήνυμα αυτό έχει δύο χρήσεις στο Πρωτόκολλο Έναρξης Συνόδου. Πρώτιστα χρησιμοποιείται για την αποδοχή έναρξης συνόδων, οπότε περιέχει στο κυρίως μέρος του σώματός του τα μέσα επικοινωνίας (media properties) του καλούμενου (UAS). Επίσης όταν χρησιμοποιείται σαν απόκριση σε άλλες αιτήσεις δηλώνει επιτυχή 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 το 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 31.



Σχήμα 31: 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).

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 του 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).

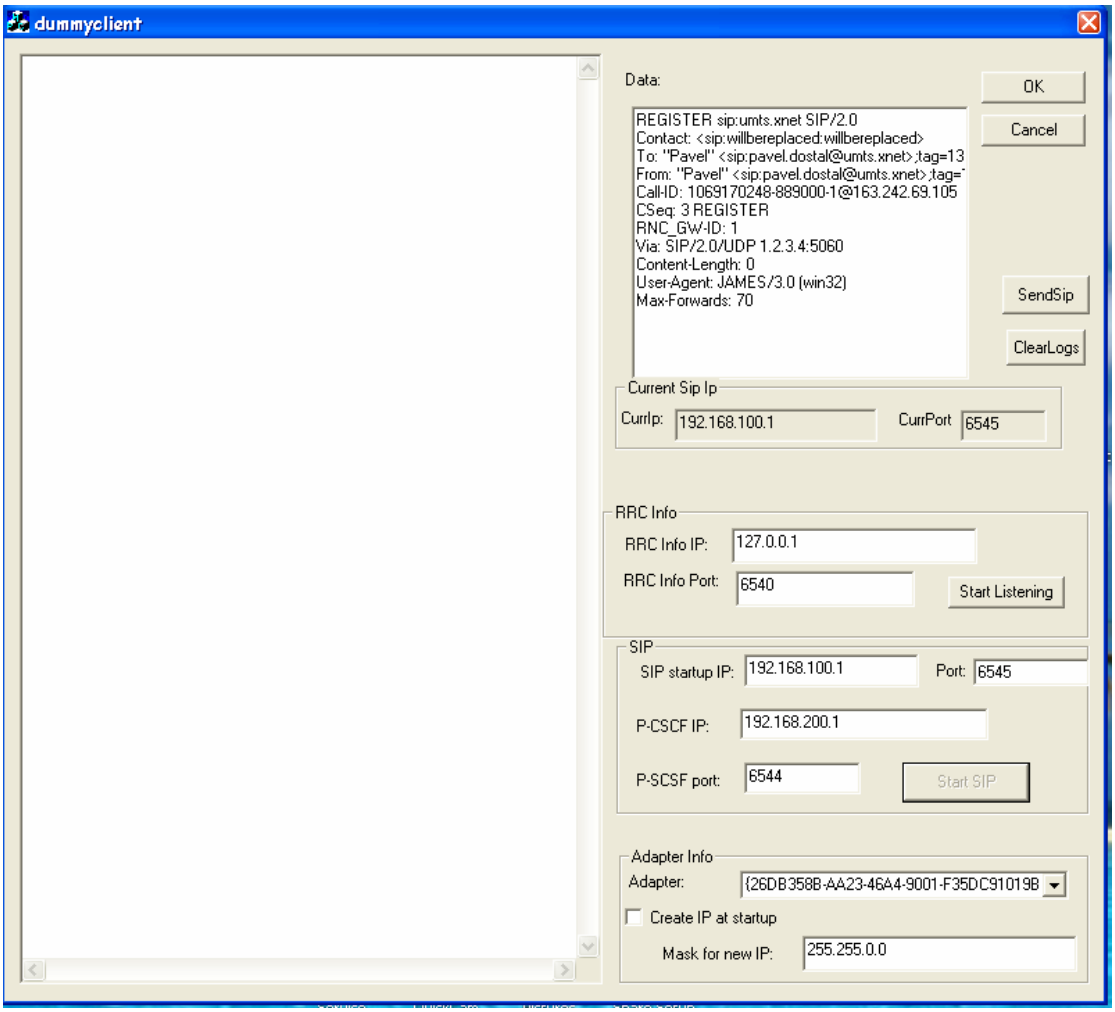
Πεδίο Cseq

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.

Παράρτημα Γ – Προετοιμασία εκτέλεσης των εφαρμογών dummyclient και CSCFDummy

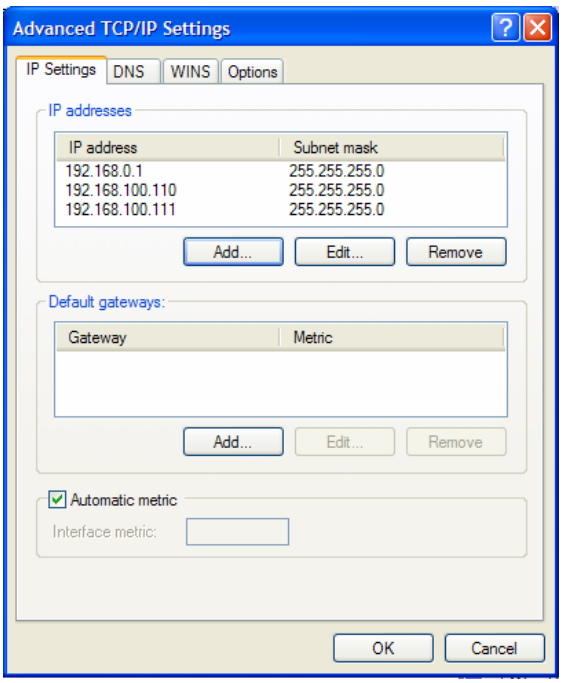
Για την εκτέλεση των εφαρμογών dummyclient.exe και CSCFDummy.exe σε περιβάλλον Windows απαιτούνται να προηγηθούν ορισμένες ενέργειες και να ικανοποιηθούν κάποιες προϋποθέσεις.

Οι απαραίτητες προϋποθέσεις που πρέπει να ικανοποιούνται για την ορθή λειτουργία της εφαρμογής dummyclient.exe (βλέπε σχήμα 32) που προσομοιώνει το κινητό τερματικό παρουσιάζονται πιο κάτω.



Σχήμα 32: Εφαρμογή dummyclient.

Ορίζουμε (βλέπε σχήμα 33) μερικές διευθύνσεις IP πάνω στην κάρτα Ethernet του υπολογιστή στον οποίο τρέχει η εν λόγω εφαρμογή. Η πρώτη διεύθυνση π.χ. 192.168.100.1 θα αποτελεί την αρχική διεύθυνση του κινητού τερματικού προτού εγγραφεί στο δίκτυο. Η διεύθυνση αυτή τοποθετείται αρχικά στο πεδίο 'SIP startup IP' και η θύρα που θα ακούει η εφαρμογή στο πεδίο 'Port' όπως φαίνεται στο σχήμα 32. Η επόμενη διεύθυνση IP θα είναι μια από τις διαθέσιμες διευθύνσεις τις οποίες θα δίνει ο κόμβος RASN στο κινητό τερματικό κατά τη διαδικασία της εγγραφής. Αν η διεύθυνση αυτή δεν είναι ορισμένη πάνω στην κάρτα δικτύου, τότε η εφαρμογή δεν θα λειτουργήσει ορθά. Μπορούμε να ορίσουμε και άλλες διευθύνσεις στην κάρτα δικτύου αναλόγως αν θα αρχικοποιήσουμε και άλλα κινητά τερματικά πάνω στον ίδιο υπολόγιστη.

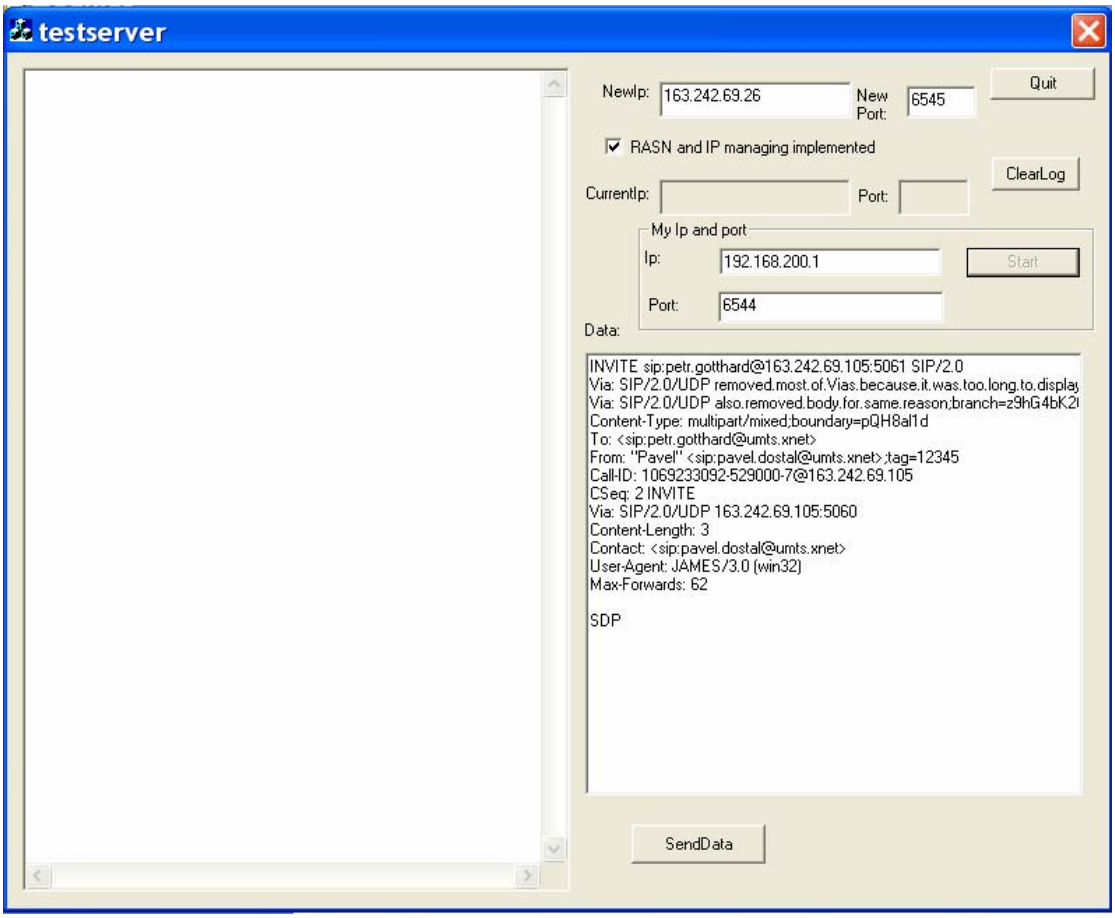


Σχήμα 33: Ορισμός IP διευθύνσεων στην κάρτα δικτύου.

Επίσης τοποθετούμε στο πεδίο 'P-CSCF IP' τη διεύθυνση IP του κόμβου P-CSCF δηλαδή τη διεύθυνση της κάρτας δικτύου του υπολογιστή πάνω στον οποίο τρέχει η εφαρμογή CSCFDummy και στο πεδίο 'P-CSCF port' την αντίστοιχη θύρα (βλέπε σχήμα 32).

Η απαραίτητη προϋπόθεση που πρέπει να ικανοποιείται για την ορθή λειτουργία της εφαρμογής CSCFDummy.exe που προσομοιώνει τον κόμβο P-CSCF είναι η

τοποθέτηση της διεύθυνσης IP του εν λόγω κόμβου στο πεδίο ‘Ip’ και της αντίστοιχης θύρας ‘Port’ (βλέπε σχήμα 34).



Σχήμα 34: Εφαρμογή CSCFDummy.

Σημειώσεις

Παραπομπές

[1] H. Schulzrinne, J. Rosenberg, "The Session Initiation Protocol: Internet – Centric Signalling", IEEE Communications Magazine, October 2000.

[2] Alan B. Johnston , SIP: Understanding the Session Initiation Protocol. -2nd ed.- Artech House telecommunication library.

[3] Dave Wisely, Philip Eardley, Louise Burness, IP for 3G NETWORKING TECHNOLOGIES FOR MOBILE COMMUNICATIONS. Wiley.

[4] W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, UNIX Network Programming The Sockets Networking API Volume 1 3rd Edition. Addison Wesley.

[5] 3GPP TS 23.060 V6.0.0 (2003-03)

[6] 3GPP TS 23.101 V4.0.0 (2001-04)

[7] 3GPP TS 23.121 V3.6.0 (2002-06)

[8] 3GPP TS 23.221 V6.0.0 (2003-03)

[9] 3GPP TS 23.002 V6.0.1 (2003-03)

[10] 3GPP TS 22.060 V5.2.0 (2002-06)

[11] SAILOR D03.01

[12] SAILOR D03.02

[13] SAILOR D04.01

[14] SAILOR D04.02

Ευρετήριο

T

U

V

Α

Β

Δ

Ε

Ι

Κ

Μ

Π

Σ