Ατομική Διπλωματική Εργασία

ΥΛΟΠΟΙΗΣΗ ΕΞΥΠΗΡΕΤΗΤΗ VEHICULAR INFORMATION TRANSFER PROTOCOL

Αναστασίου Κώστας

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιούνιος 2008

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση Εξυπηρετητή Vehicular Information Transfer Protocol

Αναστασίου Κώστας

Επιβλέπων Καθηγητής
Δικαιάκος Μάριος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Ιούνιος 2008

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου κ. Μάριο Δικαιάκο για την καθοδήγηση που μου πρόσφερε στην υλοποίηση της εργασίας, καθώς και για τις γνώσεις που μου πρόσφερε.

Επίσης, ευχαριστώ το Νικόλα Στυλιανίδη για την ανεκτίμητη σε μέγεθος και σημασία βοήθεια του καθώς και τους Αστέριο Κατσιφοδήμο και Λουλούδη Νικόλα για τη μικρή αλλά σημαντική βοήθεια που μου πρόσφεραν.

Περίληψη

Τα ασύρματα ad-hoc δίκτυα είναι μια ραγδαία εξελισσόμενη και πολλά υποσχόμενη τεχνολογία. Οι πιθανές χρήσεις τους είναι απεριόριστες, μια εκ των οποίων είναι τα VANETs (Vehicular Ad-hoc Networks), επέκταση των MANETS (Mobile Ad-hoc Networks). Τα VANETs παρέχουν επικοινωνία μεταξύ κοντινών οχημάτων και μεταξύ οχημάτων και υπηρεσιών. Ύπαρξη τέτοιου τύπου δικτύου απαιτεί τη δημιουργία κάποιου πρωτοκόλλου επικοινωνίας.

Το πρωτόκολλο VITP (Vehicular Information Transfer Protocol) είναι ένα επιπέδου εφαρμογής (application layer) πρωτόκολλο το οποίο παρέχει την υποδομή για επικοινωνία μεταξύ οχημάτων σε ad-hoc δικτύωση. Το πρωτόκολλο μπορεί να χρησιμοποιηθεί για παροχή υπηρεσιών σχετικές με κυκλοφοριακή κίνηση για διευκόλυνση του οδηγού.

Σκοπός της παρούσας εργασίας είναι η δημιουργία ενός εξυπηρετητή για το πρωτόκολλο VITP. Στο άρθρο θα αναλύσουμε το πρωτόκολλο VITP, δείχνοντας παράλληλα τις απαιτήσεις ενός εξυπηρετητή για το πρωτόκολλο αυτό. Ακολούθως θα παρουσιαστεί ο σχεδιασμός του εξυπηρετητή και θα αναλυθεί η υλοποίηση του. Τέλος, θα παρουσιαστούν κάποια πειράματα και κάποιες μετρήσεις, που δείχνουν την αποδοτικότητα του εξυπηρετητή που υλοποιήθηκε.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Στο παρών κεφάλαιο ορίζουμε το πρόβλημα που επιθυμούμε να επιλύσουμε στην παρούσα εργασία, καθώς και το σκοπό της εργασίας.

1.1 Ορισμός του προβλήματος

Οι εξελίξεις στα ασύρματα δίκτυα είναι σήμερα ραγδαίες. Εκτός από τους προσωπικούς και φορητούς υπολογιστές, πολλές συσκευές κυκλοφορούν με δυνατότητα ασύρματης δικτύωσης. Αυτό οφείλεται σε πολλούς παράγοντες, κυρίως στο γεγονός ότι η ασύρματη δικτύωση παρέχει πολλές διευκολύνσεις στο χρήστη και σε συνάρτηση με τις εξελίξεις στους άλλους τομείς της πληροφορικής κάνει τη ζωή ενός κατόχου υπολογιστή πολύ πιο εύκολη. Η διάδοση πολλών φορητών συσκευών όπως φορητοί υπολογιστές, PDAs και συσκευές αναπαραγωγής mp3 είναι επίσης ένα γεγονός που έδωσε ώθηση στην εξέλιξη των ασύρματων δικτύων.

Μια σημαντική σχετικά πρόσφατη τεχνολογία στα ασύρματα δίκτυα είναι τα ad-hoc ασύρματα δίκτυα. Τα παραδοσιακά ασύρματα δίκτυα έχουν σαν κεντρικό συστατικό τα access point το οποίο διαχειρίζεται την επικοινωνία μεταξύ των κόμβων. Αντιθέτως, την ευθύνη της διαχείρισης της επικοινωνίας στα ad-hoc δίκτυα έχουν οι ίδιοι οι κόμβοι. Κάθε κόμβος είναι πρόθυμος να προωθήσει δεδομένα που λαμβάνει σε κάποιο άλλο κόμβο. Συνεπώς, κάθε κόμβος λειτουργεί εκτός από end point και σαν δρομολογητής. Αυτό το χαρακτηριστικό εκμηδενίζει κάθε ανάγκη κεντρικής υποδομής ως απαραίτητο

συστατικό για ύπαρξη δικτύου. Η κεντρική υποδομή σε ένα ad-hoc δίκτυο είναι οι κόμβοι που το αποτελούν.

Μια πολλά υποσχόμενη εφαρμογή των ad-hoc δικτύων είναι τα VANETs (Vehicular Ad-hoc Networks). Τα VANETs είναι ασύρματα ad-hoc δίκτυα τα οποία παρέχουν επικοινωνία μεταξύ κοντινών οχημάτων. Σκοπός τους είναι η ασφαλέστερη και πιο άνετη οδηγική εμπειρία, χάρις στις υπηρεσίες που μπορούν να προσφέρουν εφαρμογές που χρησιμοποιούν VANET. Με ενσωμάτωση υπολογιστών και ασύρματων συσκευών σε κάποιο όχημα, ανά πάσα στιγμή ο οδηγός μπορεί να γνωρίζει πληροφορίες σχετικές με κυκλοφοριακή κίνηση, ατυχήματα και διάφορες σχετικές υπηρεσίες όπως σταθμούς καυσίμων. Για να είναι αυτό εφικτό, αναγκαία είναι η ύπαρξη ενός πρωτοκόλλου επικοινωνίας.

Το πρωτόκολλο VITP (Vehicular Information Transfer Protocol) είναι ένα τέτοιο πρωτόκολλο. Είναι σε επίπεδο εφαρμογής και προσδιορίζει τη δομή των μηνυμάτων που ανταλλάσσουν τα οχήματα σε VANET, καθώς και τον τύπο των ερωτήσεων και απαντήσεων που ανταλλάσσουν μεταξύ τους. Ένα τέτοιο πρωτόκολλο, πριν μπει σε λειτουργία, πρέπει να αποδειχθεί ότι είναι λειτουργικό και αποτελεσματικό. Στο παρών στάδιο δεν είναι ακόμα δυνατή η δοκιμή του σε πραγματικές συνθήκες. Πρώτα είναι αναγκαία η κατασκευή κάποιων συστατικών όπως ενός εξυπηρετητή και η εργαστηριακή τους δοκιμή, πριν προχωρήσουμε σε δοκιμή πάνω σε αυτοκίνητα.

1.2 Σκοπός της μελέτης

Όπως αναφέρθηκε, για να προχωρήσουμε στην εφαρμογή του VITP πρωτοκόλλου σε πραγματικές συνθήκες, πρέπει πρώτα να κατασκευαστούν κάποιες εφαρμογές που το χρησιμοποιούν και κατόπιν να τύχουν πειραματικής μελέτης. Σκοπός της παρούσας μελέτης είναι η κατασκευή ενός εξυπηρετητή για το πρωτόκολλο.

Ο εξυπηρετητής δέχεται μηνύματα από το VANET, τα επεξεργάζεται και ενεργεί αναλόγως. Οι ενέργειες είναι είτε σύνθεση και αποστολή απάντησης στον αποστολέα του

μηνύματος, αποθήκευση του μηνύματος στην cache μνήμη ή προώθηση του μηνύματος σε άλλο κόμβο του δικτύου, αν το μήνυμα που λήφθηκε δεν αφορά τον παρών κόμβο. Για να γίνει αυτό, πρέπει αρχικά να μελετηθεί το πρωτόκολλο VITP, κατόπιν να γίνει έρευνα πάνω στις υπάρχουσες τεχνολογίες εξυπηρετητών και τέλος να γίνει σχεδιασμός και υλοποίηση του εξυπηρετητή.

Πρέπει όμως να λάβουμε υπόψιν τις ιδιαιτερότητες που διέπουν τη χρήση του πρωτοκόλλου. Το πρωτόκολλο θα εφαρμόζεται πάνω σε κινούμενα οχήματα. Συνεπώς ο εξυπηρετητής δε θα λειτουργεί σε κάποιο ισχυρό υπολογιστή όπως οι παραδοσιακοί HTTP και FTP εξυπηρετητές, αλλά σε ένα πιο περιορισμένης απόδοσης και μνήμης μηχάνημα. Η συνεχής κίνηση του οχήματος είναι επίσης κάτι που πρέπει να ληφθεί υπόψιν, αφού όχι μόνο ο οικοδεσπότης του οχήματος θα κινείται, αλλά και οι υπόλοιποι κόμβοι του VANET θα κινούνται, συνεπώς η τοπολογία του δικτύου θα μεταβάλλεται συνεχώς. Κάθε παράγοντας που αναφέρθηκε επηρεάζει σε κάποιο βαθμό το σχεδιασμό του εξυπηρετητή.

Κεφάλαιο 2

Θεωρία

Στο παρών κεφάλαιο παραθέτουμε το θεωρητικό υπόβαθρο πίσω από το σχεδιασμό και την υλοποίηση του εξυπηρετητή. Η θεωρία που παρατίθεται αποτελεί ουσιαστικά την έρευνα που διεξάγαμε πριν ξεκινήσουμε το σχεδιασμό του εξυπηρετητή.

2.1 Vehicular ad-hoc networks (VANETS)

Η επικοινωνία μεταξύ οχημάτων είναι ένα ενδιαφέρον πεδίο μελέτης. Αρχικός στόχος της είναι η παροχή άνετης και αφαλούς οδηγικής εμπειρίας, η οποία επιτυγχάνεται με παροχή συγκεκριμένων υπηρεσιών στον οδηγό. Υπηρεσίες που περιλαμβάνουν αναφορά τροχαίας κίνησης, ατυχημάτων καθώς και ενημέρωση για υπάρχουσες υπηρεσίες όπως σταθμούς καυσίμων.

Δημιουργία ενός VANET [3] το οποίο λειτουργεί αποτελεσματικά και με αποδεκτό αριθμό σφαλμάτων είναι μια μεγάλη πρόκληση. Υπάρχουν πολλοί παράγοντες που πρέπει να ληφθούν υπόψιν. Κατ' αρχή, η τοπολογία του δικτύου μεταβάλλεται, με συνεχή μετακίνηση των κόμβων του δικτύου καθώς και με είσοδο νέων κόμβων και έξοδο υπαρχόντων κόμβων στο δίκτυο. Αυτό επηρεάζει τη δρομολόγηση των μηνυμάτων και είναι σημαντικό να ληφθεί υπόψιν στην κατασκευή αλγόριθμου δρομολόγησης.

Τίθενται επίσης θέματα ασφάλειας και προστασίας προσωπικών δεδομένων, αφού οι πλείστοι οδηγοί επιθυμούν να μείνουν ανώνυμοι, δηλαδή η ταυτότητα του οχήματος τους να μη γνωστοποιείται στους υπόλοιπους. Τέλος, πρέπει να ληφθεί υπόψιν το γεγονός ότι το bandwidth του ασύρματου μέσου είναι σχετικά μικρό, συνεπώς τα μηνύματα που ανταλλάσσονται πρέπει να είναι μικρού μεγέθους αλλά να έχουν πληρότητα δεδομένων.

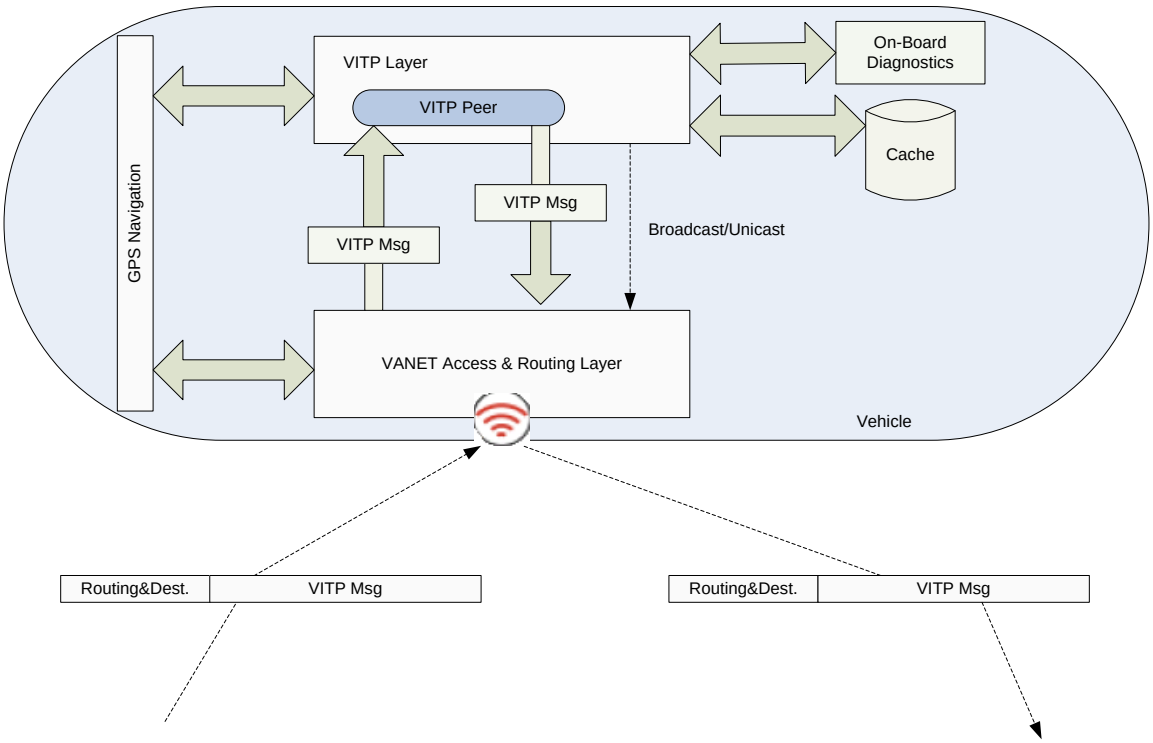
Τα VANETs έχουν τη δυνατότητα προσφοράς ενός εύρους υπηρεσιών στον οδηγό. Ανά πάσα στιγμή ο οδηγός μπορεί να γνωρίζει πληροφορίες για τροχαία κίνηση σε κάποια κοντινή οδό, για τυχόν ατυχήματα που προκλήθηκαν σε κάποιο σημείο πλησίον του οχήματος του καθώς και για κινδύνους που τυχόν να υπάρχουν όπως ολισθηρό οδόστρωμα ή κατολίσθηση. Επίσης, μπορούν να παρέχονται πληροφορίες αναφορικά με οδηγικές υπηρεσίες, όπως σταθμούς καυσίμων, χώρους στάθμευσης και ξενοδοχεία. Οι πληροφορίες αυτές μπορούν να είναι αρκετά λεπτομερείς, όπως τοποθεσία, εύρος υπηρεσιών και τιμές. Οι υπηρεσίες που αναφέρθηκαν έχουν χρησιμότητα όχι μόνο σε οδηγούς, αλλά και σε επιχειρήσεις που χρησιμοποιούν οχήματα εκτεταμένα (για παράδειγμα υπηρεσίες παράδοσης πακέτων), για συνεχή παρακολούθηση των οχημάτων τους.

Τα VANETs είναι μια πολλά υποσχόμενη τεχνολογία. Απαιτεί όμως προσεκτική και εκτεταμένη έρευνα, με κεντρικό άξονα την ασφάλεια του οδηγού και την ποιότητα εξυπηρέτησης, ούτως ώστε να πειστούν οι μεγάλες αυτοκινητοβιομηχανίες ότι αξίζει να επενδύσουν στη συγκεκριμένη τεχνολογία.

2.2 Vehicular Information Transfer Protocol (VITP)

Το VITP [11] είναι ένα application layer πρωτόκολλο επικοινωνίας μεταξύ οχημάτων. Σχεδιάστηκε για χρήση σε VANETs και καθορίζει τη μορφή των μηνυμάτων που ανταλλάσσονται μέσα στο δίκτυο και πως οι κόμβοι του δικτύου ενεργούν όταν λάβουν κάποιο μήνυμα.

Για καλύτερη κατανόηση του πρωτοκόλλου, η στρωματοποίηση των πρωτοκόλλων που αλληλεπιδρούν σε κάποιο όχημα του δικτύου θα ήταν χρήσιμη. Η στρωματοποίηση φαίνεται στο Σχήμα 2.1:



Σχήμα 2.1 Στρωματοποίηση πρωτοκόλλων οχήματος

Το στρώμα του VITP επικοινωνεί απευθείας με τα χαμηλότερα στρώματα του VANET (access και routing layers), με την υπηρεσία GPS του οχήματος καθώς και με αισθητήρες εγκατεστημένους στο όχημα. Απαραίτητη είναι επίσης κάποια ποσότητα μνήμης (cache) για προσωρινή αποθήκευση κάποιων μηνυμάτων τα οποία ίσως χρησιμοποιηθούν στο εγγύς μέλλον.

Επικοινωνία με το ad-hoc δίκτυο επιτυγχάνεται μέσω των στρωμάτων του VANET. Τα στρώματα αυτά, με χρήση ασύρματων συσκευών καθώς και πρωτοκόλλων δρομολόγησης επικοινωνούν με τα υπόλοιπα οχήματα, στέλνοντας και λαμβάνοντας μηνύματα του πρωτοκόλλου VITP (VITP messages). Όποτε κάποιο μήνυμα ληφθεί από το VANET, αυτό στέλνεται στο VITP peer. Αν ο VITP peer για κάποιο λόγο δεν μπορεί να λάβει το μήνυμα, το VANET θα το στείλει σε κάποιο γειτονικό όχημα. Αλλιώς, το

μήνυμα λαμβάνεται και τυγχάνει επεξεργασίας από το VITP peer. Όταν ο VITP peer επιθυμεί να στείλει κάποιο μήνυμα, αυτό προωθείται στο VANET στρώμα και αυτό με χρήση κάποιου πρωτόκολλου δρομολόγησης θα προωθήσει το μήνυμα στο ad-hoc δίκτυο.

Η πληροφορία της τοποθεσίας του οχήματος στο χώρο προέρχεται από το σύστημα GPS. Αυτό στέλνει στο VITP peer δυάδες συντεταγμένων GPS (longitude, latitude). Αυτές, με χρήση κάποιου πρωτοκόλλου μετατρέπονται σε δυάδες συντεταγμένων VITP (road_id, segment_id) οι οποίες αποτελούν την κατανόηση της τοποθεσίας από το VITP.

Η μνήμη cache περιέχει πρόσφατες πληροφορίες οι οποίες μπορεί να ζητηθούν είτε από τον οδηγό είτε από άλλα οχήματα, εκτός εάν η αίτηση προδιαγράφει συγκεκριμένα μη επιθυμία χρήσης της cache.

Τέλος, πληροφορίες για το όχημα και την κατάσταση του (θέση, ταχύτητα, καύσιμα, θερμοκρασία, αριθμοί εγγραφής) στέλνονται από τα on-board diagnostics του οχήματος, δηλαδή διάφορους αισθητήρες οι οποίοι είναι τοποθετημένοι στο όχημα. Όλα τα σύγχρονα οχήματα διαθέτουν τέτοιους αισθητήρες, οι πιο συνήθεις από τους οποίους είναι αισθητήρες στροφών μηχανής, ταχύτητας, θερμοκρασίας μηχανής και ποσοστού διαθέσιμου καύσιμου. Τα πλείστα σύγχρονα οχήματα έχουν επίσης διάφορους άλλους αισθητήρες, όπως αισθητήρες εξωτερικής θερμοκρασίας, διάφορων βλαβών της μηχανής, μέσου όρου κατανάλωσης καύσιμου, μέσου όρου ταχύτητας και διάφορους άλλους, ανάλογα με την εταιρεία κατασκευής του οχήματος.

2.3 Μηνύματα VITP

Η μορφή των μηνυμάτων του VITP φαίνεται στον Πίνακα 2.1:

VITP Message Syntax	
Line 1:	METHOD <uri> VITP/<version_number>
Line 2:	Target: [rd_id-dest , seg_id-dest]
Line 3:	From: [rd_id-src , seg_id-src] with <speed>
Line 4:	Time: <current_time>
Line 5:	Expires: <expiration_time>
Line 6:	Cache-Control: <directive>
Line 7:	TTL: <time_to_live>
Line 8:	msgID : <unique_key>
Line 9:	Content-Length: <number of bytes>
Line 10:	CRLF
Line 11:	<message body>
URI Syntax	
/<type>/<tag>?[<rc_expr>,…]&<param_expr>&…	

Πίνακας 2.1 Μήνυμα VITP

Υπάρχουν δύο τύποι μηνυμάτων (**METHOD**) στο πρωτόκολλο, οι GET και POST. Τα μηνύματα τύπου GET είναι αιτήσεις ενώ τα POST απαντήσεις σε ερωτήματα. Το ερώτημα εμπεριέχεται στο tag <uri>. Υπάρχουν δύο τύποι ερωτημάτων, το vehicle και το service. Το vehicle αφορά ερωτήματα για οχήματα, όπως ποσότητα οχημάτων σε κάποια περιοχή, ενώ το service αφορά υπηρεσίες, όπως σταθμούς καυσίμων.

Στις γραμμές 2 και 3 έχουμε τα Target και From, τα οποία χαρακτηρίζουν την περιοχή που πρέπει να πάει το μήνυμα και από που προέρχεται αντίστοιχα. Στη γραμμή 4, το Time αντιπροσωπεύει τη χρονική στιγμή την οποία στάλθηκε το μήνυμα. Στη γραμμή 5, το Expires καθορίζει τη χρονική στιγμή της λήξης του μηνύματος η οποία όταν περάσει το μήνυμα σταματά να τυγχάνει επεξεργασίας. Στη γραμμή 6 το Cache-Control καθορίζει κάποιες οδηγίες για χρήση της cache σε ερωτήματα και απαντήσεις ερωτημάτων. Το TTL στη γραμμή 7 αφορά το χρόνο για τον οποίο κάποιο μήνυμα θα αποθηκευτεί στη cache. Το msgID αποτελεί μοναδικό string αναγνώρισης του κάθε μηνύματος. Υπάρχουν διάφορες τεχνικές παραγωγής μοναδικού string αναγνώρισης. Τέλος, το Content-Length περιέχει τον αριθμό σε bytes στο message body και το CRLF διαχωρίζει το header του

μηνύματος με τα υπόλοιπα δεδομένα. Τα δεδομένα αυτά υπάρχουν μόνο σε μηνύματα που περιέχουν ενδιάμεσα δεδομένα, όπου δηλαδή για επίλυση ενός ερωτήματος χρειάζεται συνεισφορά πολλών οχημάτων και κάθε όχημα προσθέτει τα δικά του δεδομένα και ακολούθως προωθεί το μήνυμα σε άλλο όχημα.

Συνοπτικά, οι τιμές που μπορεί να πάρει κάθε κομμάτι του μηνύματος έχουν ως εξής:

VITP Value Types	
METHOD	GET, POST
<uri> - <type>	vehicle, service
<uri> - <tag>	traffic, alert, gas, index
Target/From	Formatted position values: [rd_id, seg_id]
Time	Timestamp
Expires	Timestamp
Cache-control	**GET:** yes, no, fwd **POST:** yes, no
TTL	mseconds (integer)
msgID	Globally unique string
Content-length	size in bytes (integer)
CRLF	Special character CRLF
Message body	Intermediate results (string)

Πίνακας 2.2 Τιμές κομματιών μηνύματος VITP

2.4 Ερωτήματα VITP

Στην πρώτη γραμμή του μηνύματος καθορίζονται οι ενέργειες του παραλήπτη ενός μηνύματος. Συγκεκριμένα το κομμάτι METHOD <uri> αποτελεί ένα ερώτημα (query) στον παραλήπτη του μηνύματος, με το METHOD να αποτελεί τον τύπο και το <uri> το περιεχόμενο του ερωτήματος.

Κάποια παραδείγματα ερωτημάτων φαίνονται στον πιο κάτω πίνακα:

1.	GET	/vehicle/traffic?	[cnt=10&tout=3000msec]	&tframe=3min
2.	GET	/vehicle/traffic?	[cnt=*&tout=1800msec]	&tframe=0min
3.	GET	/service/gas?	[cnt=4&tout=1800msec]	&price<2USD
4.	GET	/vehicle/alert?	[cnt=20&tout=500msec]	&type=accident
5.	POST	/vehicle/alert?	[cnt=*&tout=*]	&type=slippery_road

Πίνακας 2.3 Ερωτήματα VITP

Το πρώτο ερώτημα αναζητά την κυκλοφοριακή κίνηση σε κάποια γεωγραφική περιοχή. Η κυκλοφοριακή κίνηση στο VITP μεταφράζεται ως μέση ταχύτητα κάθε οχήματος που βρίσκεται στη συγκεκριμένη περιοχή. Πιο συγκεκριμένα, το ερώτημα ζητά τη μέση ταχύτητα 10 οχημάτων σε κάποια συγκεκριμένη περιοχή (cnt=10). Το ερώτημα πρέπει να επιλυθεί μέσα σε 3000 milliseconds (tout=3000msec). Τα cnt και tout υπάρχουν σε οποιοδήποτε τύπο και υποκατηγορία ερωτήματος. Μετά τη δυάδα cnt και tout ακολουθούν κάποιες παράμετροι, ανάλογα με το ερώτημα. Στο πρώτο ερώτημα, η παράμετρος tframe έχει νόημα πως η μέση ταχύτητα που θα επιστραφεί θα είναι για τα τελευταία 3 λεπτά (tframe=3min).

Η διαφορά του δεύτερου ερωτήματος με το πρώτο είναι ότι το δεύτερο ερώτημα ουσιαστικά ζητά την ποσότητα των οχημάτων σε κάποια περιοχή. Αυτό επιτυγχάνεται θέτοντας ως τιμή του cnt τον αστερίσκο (*), ότι δηλαδή στο ερώτημα θα συμμετέχουν όλα τα οχήματα στην περιοχή. Η τιμή του tout εξασφαλίζει ότι το μήνυμα δε θα κυκλοφορεί επ’άπειρο μέσα στο δίκτυο. Επίσης, η μηδενική τιμή του tframe συμβολίζει ότι το αυτοκίνητο θα δώσει την τρέχουσα ταχύτητα του, και όχι κάποια μέση ταχύτητα.

Το τρίτο ερώτημα ζητά πληροφορίες για σταθμούς καυσίμων σε κάποια περιοχή. Συγκεκριμένα, ζητά πληροφορίες για μέχρι τέσσερις (4) σταθμούς καυσίμων (cnt=4) όπου η τιμή της βενζίνης δεν ξεπερνά τα δύο δολάρια (price<2USD).

Το τέταρτο ερώτημα αναζητά κατά πόσον υπάρχει δυστύχημα σε κάποια περιοχή. Αυτό επιτυγχάνεται στέλνοντας ερώτημα τύπου /vehicle/alert με παράμετρο type=accident. Παρατηρούμε παράμετρο cnt=20, δηλαδή το ερώτημα πρέπει να περάσει από 20

οχήματα μέχρι να ολοκληρωθεί. Με αυτό τον τρόπο, αυξάνονται οι πιθανότητες να βρεθεί κάποιο όχημα που είχε δυστύχημα, αφού 20 οχήματα είναι ικανοποιητικός αριθμός.

Τέλος το πέμπτο μήνυμα στέλνει ειδοποίηση σε όποια οχήματα μπορούν να το λάβουν ότι υπάρχει ολισθηρός δρόμος στην περιοχή που βρίσκεται το όχημα. Αυτό επιτυγχάνεται στέλνοντας μήνυμα τύπου POST /vehicle/alert με παράμετρο type=slippery_road. Το μήνυμα θα καταφθάσει σε όλα τα οχήματα εντός του δικτύου αφού σαν παραμέτρους έχουμε cnt=* και tout=*.

2.4 Virtual Ad-hoc Server (VAHS)

Ένα ερώτημα που απασχόλησε τους δημιουργούς του VITP είναι κατά πόσον μια συγκεντρωτική προσέγγιση για επικοινωνία μεταξύ οχημάτων αποτελεί μια καλή λύση. Για να απαντηθεί το συγκεκριμένο ερώτημα, πρέπει να μελετηθεί τι απαιτεί μια τέτοια προσέγγιση.

Μια συγκεντρωτική προσέγγιση απαιτεί αρχικά ένα πολύ μεγάλο αριθμό αισθητήρων, οι οποίοι είναι σε συνεχή λειτουργία και συγκεντρώνουν πληροφορίες για την τροχαία κίνηση, ακόμη και αν αυτή είναι ανύπαρκτη ή αν κανένα αυτοκίνητο δε θέτει ερωτήματα. Στη συνέχεια, απαιτείται ένας μεγάλος αριθμός ισχυρών εξυπηρετητών, οι οποίοι ανανεώνουν συνεχώς την εικόνα της τροχαίας κίνησης σε ένα πολύ μεγάλο γεωγραφικό χώρο. Οι συγκεκριμένοι εξυπηρετητές όχι μόνο πρέπει να ανανεώνουν συνεχώς τα δεδομένα τους, πρέπει επίσης να είναι σε θέση να απαντούν με ακρίβεια ένα μεγάλο αριθμό ερωτημάτων σε ελάχιστο χρονικό διάστημα.

Κατά συνέπεια, υπάρχει και ανάγκη για μια ισχυρή δικτυακή υποδομή, η οποία είναι σε θέση να μεταφέρει συνεχώς ένα μεγάλο αριθμό δεδομένων μεταξύ οχημάτων και εξυπηρετητών. Τέλος, απαιτείται μεγάλος αριθμός ανθρώπινου δυναμικού, το οποίο επιτηρεί τους εξυπηρετητές καθ’ όλη τη διάρκεια της ημέρας και είναι σε θέση να ενεργήσει ταχύτατα σε περίπτωση βλάβης κάποιου κομματιού του συστήματος. Συνεπώς,

χρειάζονται έμπειρα και ικανά άτομα για το συγκεκριμένο σκοπό, κάτι που ανεβάζει κατακόρυφα το κόστος του συστήματος.

Έχοντας κατά νου τις απαιτήσεις μιας συγκεντρωτικής προσέγγισης, καταλήγουμε ότι μια τέτοια προσέγγιση είναι πάρα πολύ ακριβή και ακόμη και αν δημιουργηθεί ένα τέτοιο σύστημα, είναι πολύ δύσκολο να επιτευχθεί ορθή λειτουργία του.

Η λύση που προτάθηκε δεν απαιτεί καμία επιπλέον υποδομή. Αντί αυτού, τα οχήματα και οι υπηρεσίες τα οποία αποτελούν τους κόμβους του δικτύου, συνθέτουν ταυτόχρονα και ένα εικονικό εξυπηρετητή. Κάθε κόμβος ο οποίος βρίσκεται μέσα σε κάποια συγκεκριμένη περιοχή και είναι διατεθειμένος να συμμετέχει στην απάντηση ερωτημάτων παρέχοντας πληροφορίες, επικοινωνεί με τους υπόλοιπους με ad-hoc τρόπο. Έτσι, οι κόμβοι σε κάποια περιοχή δημιουργούν τον εικονικό εξυπηρετητή, ο οποίος ονομάζεται Virtual Ad-hoc Server (**VAHS**).

Σημαντικό χαρακτηριστικό του VAHS είναι ότι ακολουθεί προσέγγιση μέγιστης προσπάθειας (best effort approach). Δεν παρέχονται δηλαδή εγγυήσεις ότι κάποιο μήνυμα θα καταλήξει στο στόχο του ή ότι κάποιο ερώτημα θα απαντηθεί. Επίσης, είναι πιθανό κάποιος κόμβος να εισέλθει στο δίκτυο, να συμμετάσχει προσωρινά στην απάντηση κάποιου ερωτήματος και να εξέλθει του δικτύου πριν το ερώτημα απαντηθεί, κάτι που πιθανόν να εγείρει προβλήματα. Ένα άλλο σημαντικό χαρακτηριστικό του VAHS είναι ότι ένας κόμβος δε διαθέτει πληροφορίες για τους υπόλοιπους, αλλά ούτε το ίδιο το VAHS διαθέτει ρητές πληροφορίες για τους κόμβους που το αποτελούν.

Αυτά τα χαρακτηριστικά, παρόλα τα αρνητικά τους, σχηματίζουν ένα σύστημα απλό και αποδοτικό. Συνεπώς, ένας κόμβος μπορεί να λειτουργήσει σε κάποιο μηχάνημα σχετικά περιορισμένης δυναμικότητας, όπως είναι για παράδειγμα ο επεξεργαστής ενός οχήματος.

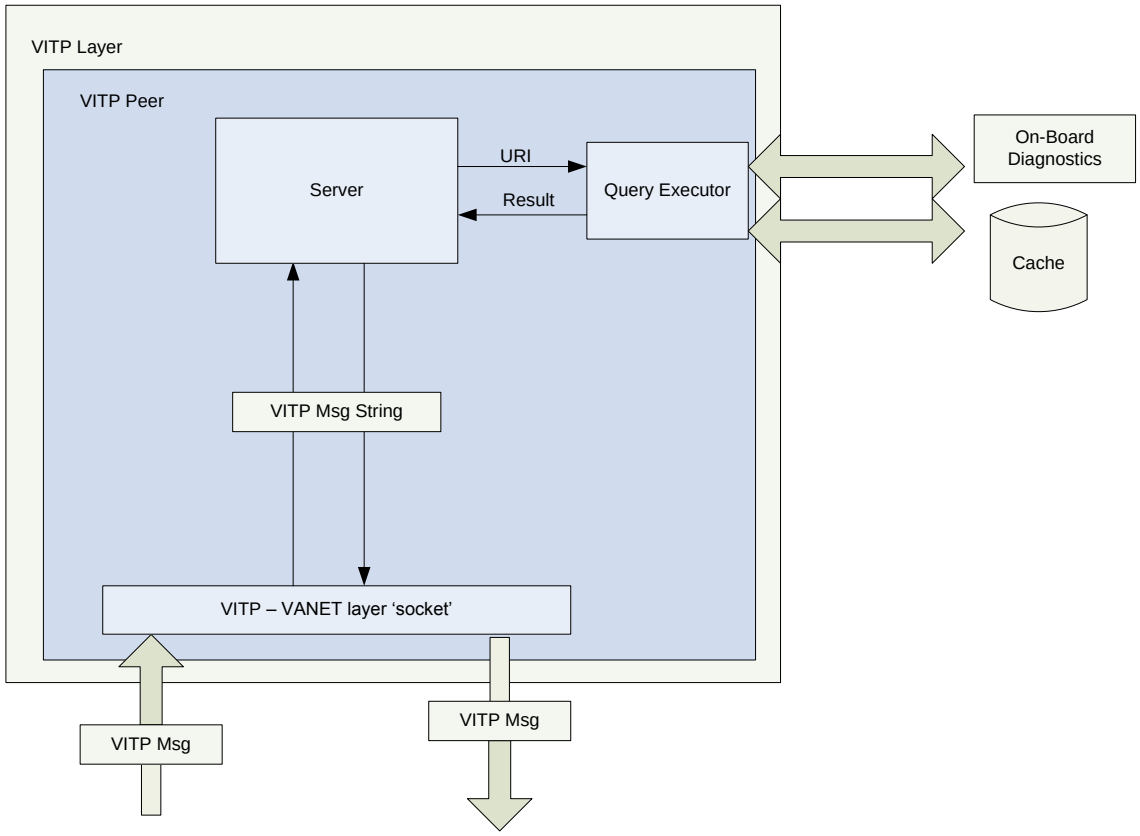
Κεφάλαιο 3

Σχεδιασμός VITP Server

Στο παρών κεφάλαιο περιγράφουμε τη διαδικασία σχεδίασης του εξυπηρετητή, αναφέροντας το λογισμό που ακολουθήσαμε και τους παράγοντες που λάβαμε υπόψιν για να πάρουμε διάφορες αποφάσεις.

3.1 Συστατικά επικοινωνίας με άλλα στρώματα

Ο εξυπηρετητής επικοινωνεί εντός του οχήματος με το στρώμα VANET Access και Routing, με τα on board diagnostics και με τη συσκευή GPS. Τα on board diagnostics και η συσκευή GPS συνεχώς δίνουν πληροφορίες στον εξυπηρετητή, οι οποίες ανά πάσα στιγμή είναι αναγκαίες για δυνατότητα απάντησης στα μηνύματα που λαμβάνονται. Όσον αφορά την επικοινωνία με το στρώμα VANET Access και Routing, ο εξυπηρετητής απλά στέλνει το μήνυμα στο στρώμα για επεξεργασία και αποστολή. Η εικόνα της ενδοεπικοινωνίας εντός του οχήματος φαίνεται στο Σχήμα 3.1:



Σχήμα 3.1 Εικόνα ενδοεπικοινωνίας των στρωμάτων του οχήματος

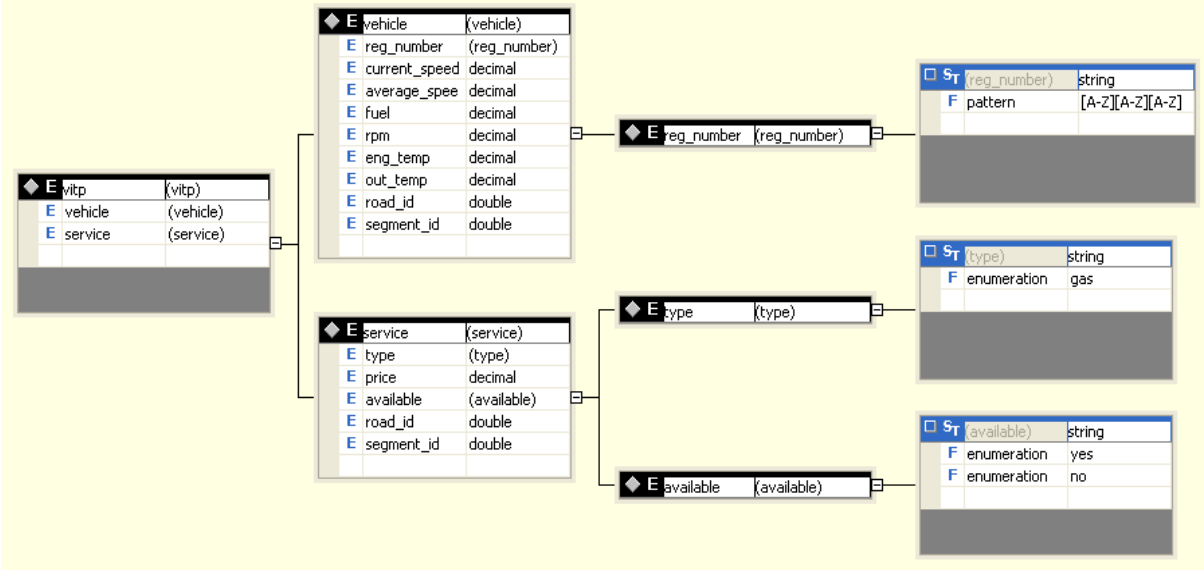
Δημιουργία αλγόριθμου δρομολόγησης για αποστολή μηνυμάτων εντός του ad-hoc δικτύου καθώς και κατασκευή του συστατικού το οποίο μεταφράζει δυάδες longitude, latitude σε δυάδες road_id, segment_id είναι εκτός των στόχων της παρούσας εργασίες. Στην παρούσα εργασία όμως θα υποθέσουμε ότι τα συγκεκριμένα συστατικά θα κατασκευαστούν στο μέλλον, συνεπώς ο σχεδιασμός θα δίνει τη δυνατότητα εύκολης προσθήκες τους στον εξυπηρετητή.

Οι πληροφορίες που στέλνονται από τα on board diagnostics θα αποθηκεύονται σε αρχείο XML. Συνεπώς, θα κατασκευάσουμε μια XML γραμματική για προσδιορισμό της δομής των πληροφοριών που θα αποθηκεύονται στο αρχείο. Είναι αρχικά σημαντικό να διαχωρίζουμε τις πληροφορίες για όχημα και για υπηρεσία, αφού επιθυμούμε να αποθηκεύουμε διαφορετικές πληροφορίες για τον κάθε τύπο κόμβου.

Κάθε κατασκευάστρια εταιρεία αυτοκινήτων έχει διάφορους αισθητήρες για παροχή πληροφοριών στον οδηγό. Οι πιο κοινές πληροφορίες που παρέχονται από κάποιο όχημα και επιθυμούμε να αποθηκεύονται στο αρχείο για τα οχήματα είναι αριθμός εγγραφής, τρέχουσα ταχύτητα, μέση ταχύτητα, ποσοστό διαθέσιμου καύσιμου, στροφές μηχανής, θερμοκρασία μηχανής, εξωτερική θερμοκρασία και συντεταγμένες (δυάδα road_id, segment_id).

Οι πληροφορίες της υπηρεσίας που θέλουμε να αποθηκεύσουμε, σύμφωνα με την παρούσα έκδοση του πρωτοκόλλου VITP, είναι τύπος, τιμή, διαθεσιμότητα και συντεταγμένες (δυάδα road_id, segment_id).

Διάγραμμα της γραμματικής φαίνεται στο Σχήμα 3.2 (αρχείο VITPScema.xsd όπως παρουσιάζεται στο Visual Studio 2005):



Σχήμα 3.2 Απεικόνιση της XML γραμματικής που αναπτύχθηκε

Στη γραμματική δύναται να προστεθούν περισσότερα στοιχεία στο μέλλον, κατόπιν δημιουργίας νέων εκδόσεων του πρωτοκόλλου. Ειδικότερα για τις υπηρεσίες, οι οδηγοί πιθανών να επιθυμούν γνώση περισσότερων πληροφοριών, ανάλογα με τον τύπο της υπηρεσίας. Για παράδειγμα στην περίπτωση του σταθμού καυσίμων, ο οδηγός ίσως

επιθυμεί να γνωρίζει κατά πόσο εκτός από καύσιμο παρέχεται και υπηρεσία πλυσίματος του αυτοκινήτου. Επειδή στην παρούσα έκδοση δεν είναι ξεκάθαρο πόσα και ποια ερωτήματα υποστηρίζονται, τέτοιες πληροφορίες δε συμπεριλήφθηκαν στη γραμματική αλλά μπορούν πολύ εύκολα να προστεθούν στο μέλλον.

Επικοινωνία με το στρώμα VANET access και routing δεν υποστηρίζεται στην παρούσα εργασία. Ο λόγος είναι ότι είναι απαραίτητη περαιτέρω μελέτη, η οποία είναι εκτός των στόχων της εργασίας. Όμως, για σκοπούς πειραμάτων και επίδειξης της εργασίας, ο εξυπηρετητής επικοινωνεί με τον έξω κόσμο μέσω των sockets. Τα sockets είναι οι μηχανισμοί επικοινωνίας μεταξύ του στρώματος εφαρμογής (application layer) και του στρώματος μεταφοράς (transport layer). Συγκεκριμένα θα χρησιμοποιηθούν UDP sockets, τα οποία λόγω του ότι είναι connectionless (δεν υπάρχει handshaking για έναρξη μεταφοράς δεδομένων) προσφέρουν ταχύτερη μεταφορά δεδομένων. Τα πειράματα που θα κάνουμε στη συνέχεια θα είναι ελεγχόμενα, συνεπώς δεν μας ανησυχεί το γεγονός ότι το UDP προσφέρει αναξιόπιστη μεταφορά δεδομένων.

Τέλος, σημαντικό συστατικό είναι ο Query Executor. Ευθύνη του είναι να πάρει το URI κομμάτι του μηνύματος, να το μεταφράσει και είτε να επιστρέψει κάποιο αποτέλεσμα το οποίο θα χρησιμοποιηθεί για τη σύσταση της απάντησης, ή θα προσθέσει το μήνυμα στη μνήμη cache. Για δημιουργία του αποτελέσματος, το συγκεκριμένο συστατικό θέτει ερωτήματα XPath στο XML αρχείο. Ακολούθως επεξεργάζεται το αποτέλεσμα του ερωτήματος και συνθέτει την απάντηση. Εάν χρειαστεί ανατρέχει στην cache για εύρεση απάντησης.

3.2 Εξυπηρετητής VITP

Βασική εργασία του server είναι η παραλαβή μηνυμάτων, απόφαση κατά πόσον και πως τον αφορούν και ακολούθως επεξεργασία και αποστολή μηνύματος. Ουσιαστικά λειτουργεί πολύ παρόμοια με ένα Web Server, δηλαδή εκτελεί ανάγνωση μηνύματος, μετάφραση του URI (όπως ο Web Server κάνει μετάφραση του URL) και τέλος δημιουργία και αποστολή απάντησης. Κάποια μηνύματα δυνατόν να αποθηκευτούν στην

cache. Στον εξυπηρετητή λειτουργούν τρία threads. Το thread παραλαβής, το thread επεξεργασίας και το thread αποστολής. Υπάρχουν επίσης κάποιες επιπλέον κλάσεις, ο Query Executor, ο Message Parser, η κλάση του VITP Message και το VITP Socker.

3.2.1 Query Executor

Η εργασία που εκτελεί ο Query Executor περιγράφηκε στο υποκεφάλαιο 3.1. Αυτός δέχεται σαν είσοδο το μήνυμα, επεξεργάζεται το URI κομμάτι του, εκτελεί ένα XPath query και επιστρέφει κάποια απάντηση.

Πριν εκτελέσει οποιαδήποτε εργασία, ο Query Executor ελέγχει το rc_expression κομμάτι του μηνύματος. Μόνο αν αυτό είναι έγκυρο, αν δηλαδή δεν ικανοποιείται κάποιο return condition θα εκτελέσει το XPath query [8] για εύρεση απάντησης. Ακόμη και αν το μήνυμα είναι έγκυρο, ο Query Executor θα τροποποιήσει αν είναι αναγκαίο το rc_expression. Αν δηλαδή το cnt του rc_expression είναι 10, αυτό θα γίνει 9, δηλαδή το μήνυμα πέρασε από ένα ακόμη κόμβο του δικτύου.

Τα XPath queries που εκτελούνται είναι πολύ απλά queries. Εάν για παράδειγμα το URI του μηνύματος είναι *GET /vehicle/traffic*, το XPath query που θα εκτελεστεί είναι *vitp/vehicle/average_speed*. Με αυτό το query θα πάρει την μέση ταχύτητα του κινούμενου οχήματος και αυτή στη συνέχεια θα προστεθεί από το νήμα (thread) επεξεργασίας στο μήνυμα της απάντησης.

Μια επιπλέον λειτουργία που εκτελεί ο Query Executor είναι η τροποποίηση και αποθήκευση των δεδομένων του XML αρχείου. Αυτό γίνεται και πάλι με XPath queries.

Σε όλες τις λειτουργίες πάνω στο XML αρχείο υπάρχει έλεγχος ταυτοχρονίας. Αυτό συμβαίνει όχι επειδή ο εξυπηρετητής που θα υλοποιηθεί θα έχει παραπάνω από ένα νήμα (thread) με πρόσβαση στο αρχείο, αλλά επειδή όταν μπει σε λειτουργία υπό πραγματικές συνθήκες, οι αισθητήρες του αυτοκινήτου θα έχουν παράλληλη πρόσβαση σε αυτό. Συνεπώς ο έλεγχος ταυτοχρονίας κρίνεται απολύτως απαραίτητος.

3.2.2 Κλάση VITP Message

Η κλάση VITP Message δεν εμπεριέχει οποιαδήποτε λειτουργικότητα. Σκοπός της είναι απλά η αποθήκευση των δεδομένων ενός μηνύματος σε κλάση για ευκολότερη επεξεργασία του μηνύματος από τα υπόλοιπα συστατικά του εξυπηρετητή.

3.2.3 VITP Message Parser

Η κλάση VITP Message Parser μετατρέπει ένα string σε κλάση VITP Message και μια κλάση VITP Message σε string.

Θεωρήθηκε ορθό οι μέθοδοι που εκτελούν την μετατροπή να είναι στατικές. Όντας στατικές, μπορούν να κληθούν ανά πάσα στιγμή από οποιοδήποτε αντικείμενο που υπάρχει στον εξυπηρετητή, χωρίς να έχει δημιουργηθεί αντικείμενο της κλάσης VITP Message Parser. Αυτό όχι μόνο απλοποιεί την όλη διαδικασία της μετατροπής, αλλά και αποτρέπει οποιαδήποτε σύγκρουση μεταξύ των threads, αφού κλήση μιας στατικής μεθόδου από κάποιο thread δημιουργεί ξεχωριστό χώρο στη μνήμη για εκτέλεση της.

3.2.4 VITP Cache

Η κλάση VITP Cache αποτελεί wrapper κλάση του Hashtable που προσφέρει η .NET. Δεν περιέχει οποιαδήποτε επιπλέον λειτουργικότητα. Δημιουργήθηκε για καλύτερη κατανόηση του κώδικα του εξυπηρετητή από κάποιο δεύτερο προγραμματιστή.

3.2.5 Thread παραλαβής μηνύματος

Το thread παραλαβής του εξυπηρετητή αποτελεί και το main loop του. Κάνει συνεχώς ακρόαση πάνω σε ένα UDP socket λαμβάνοντας VITP μηνύματα. Τα μηνύματα καταφθάνουν σε μορφή string. Μόλις κάποιο μήνυμα καταφθάσει, αυτό περνά από τον VITP Message Parser, ο οποίος μετατρέπει το ληφθέν string σε κλάση VITP Message.

Ακολούθως, το μήνυμα αποθηκεύεται ασύγχρονα σε μια ουρά για επεξεργασία από το thread επεξεργασίας. Σε αυτό το σημείο υπάρχει έλεγχος ταυτοχρονίας, επειδή το thread επεξεργασίας μπορεί να επιχειρήσει πρόσβαση στην ουρά τη στιγμή που το thread παραλαβής μηνύματος επιχειρεί να αποθηκεύσει κάποιο μήνυμα. Η ουρά στην οποία αποθηκεύονται τα μηνύματα είναι first-com first-served. Συνεπώς, όποιο μήνυμα καταφθάσει πρώτο θα εξυπηρετηθεί πρώτο.

Εδώ τίθεται το ερώτημα κατά πόσων κάποια μηνύματα θα έπρεπε να έχουν προτεραιότητα με βάση κάποιο άλλο κριτήριο πέραν του χρόνου παραλαβής. Συγκεκριμένα τα μηνύματα τύπου POST /vehicle/alert είναι σημαντικό να επεξεργάζονται γρήγορα για μεγαλύτερη ασφάλεια του οδηγού. Θα δούμε στη συνέχεια στα πειράματα, ότι ο χρόνος εξυπηρέτησης ενός μηνύματος, ακόμη και σε φάση αιχμής είναι της τάξης milliseconds. Αν αναλογιστούμε το σενάριο ότι κάποιος οδηγός βρίσκεται στον αυτοκινητόδρομο και οδηγεί με μεγάλη ταχύτητα. Αν συμβεί κάποιο δυστύχημα θα θέλαμε να ενημερωθεί εγκαίρως. Έστω στον αυτοκινητόδρομο, ο οδηγός έχει ορατότητα 500 μέτρων και κινείται με ταχύτητα $100^{km}/_{h}$. Έστω το ατύχημα έγινε ακριβώς στο σημείο που ο οδηγός δεν μπορεί να δει. Μέχρι να καταφθάσει στο σημείο του ατυχήματος, χρειάζεται 18 δευτερόλεπτα. Το μήνυμα από το αυτοκίνητο που έπαθε κάποιο ατύχημα ή εντόπισε ολισθηρό οδόστρωμα θα εξυπηρετηθεί μέσα σε λιγότερο από τρία δευτερόλεπτα σε φάση αιχμής και ο οδηγός θα ενημερωθεί για το συμβάν εγκαίρως, έχοντας το χρόνο να προετοιμαστεί. Συνεπώς, καταλήγουμε ότι δεν υπάρχουν μηνύματα τα οποία θα έπρεπε να έχουν μεγαλύτερη προτεραιότητα.

Αφού κάποιο μήνυμα αποθηκευτεί στην ουρά, γίνεται μια εγγραφή σε log αρχείο, για σκοπούς παρακολούθησης του εξυπηρετητή. Το συγκεκριμένο αρχείο θα χρησιμοποιηθεί αργότερα στα πειράματα μας.

3.2.6 Thread επεξεργασίας μηνύματος

Τα μηνύματα που καταφθάνουν στον εξυπηρετητή, αποθηκεύονται σε μια ουρά first-com first-served. Στη συνέχεια, το thread επεξεργασίας μηνύματος αφαιρεί ένα ένα τα

μηνύματα από την ουρά και τα επεξεργάζεται. Στο σημείο αυτό υπάρχει έλεγχος ταυτοχρονίας πάνω στην ουρά, αφού τόσο το thread παραλαβής όσο και το thread επεξεργασίας μηνύματος έχουν πρόσβαση στην ουρά.

Αφού κάποιο μήνυμα βγει από την ουρά, στέλνεται στον Query Executor. Αυτός δίνει πίσω το αποτέλεσμα της εκτέλεσης του ερωτήματος. Ακολούθως, το thread επεξεργασίας ελέγχει την απάντηση και ενεργεί αναλόγως. Αν ο κύκλος ζωής του μηνύματος ολοκληρώθηκε, θα δημιουργηθεί μήνυμα απάντησης και θα σταλεί στον αρχικό του αποστολέα. Αν δεν ολοκληρώθηκε, απλά θα προστεθούν δεδομένα στο μήνυμα και αυτό θα αποσταλεί στον επόμενο παραλήπτη. Το thread επεξεργασίας δεν είναι υπεύθυνο για αποστολή μηνύματος. Απλά προσθέτει το προς αποστολή μήνυμα σε μια ουρά first-come first-server και ακολούθως αυτό θα αποσταλεί από το thread αποστολής μηνύματος. Στην ουρά υπάρχει πάλι έλεγχος ταυτοχρονίας, αφού τόσο το thread επεξεργασίας όσο και το thread αποστολής μηνύματος έχουν πρόσβαση στην ουρά.

Όταν ολοκληρωθεί η επεξεργασία του μηνύματος, όποια και αν είναι αυτή, γίνεται εγγραφή σε log αρχείο, για λόγους παρακολούθησης του εξυπηρετητή. Αυτό το αρχείο θα χρησιμοποιηθεί στα πειράματα μας, όπως συμβαίνει και με το log αρχείο του thread παραλαβής μηνύματος.

3.2.7 Thread αποστολής μηνύματος

Τα μηνύματα τα οποία έτυχαν επεξεργασίας από το thread επεξεργασίας μηνυμάτων, προστέθηκαν σε μια ουρά first-come first-served για αποστολή. Υπεύθυνο για την αποστολή τους είναι το thread αποστολής μηνύματος. Στο σημείο αυτό υπάρχει έλεγχος ταυτοχρονίας πάνω στην ουρά, αφού τόσο το thread επεξεργασίας όσο και το thread αποστολής μηνύματος έχουν πρόσβαση στην ουρά.

Αφού κάποιο μήνυμα αφαιρεθεί από την ουρά, ακολούθως ελέγχονται οι συντεταγμένες του αποστολέα. Υπάρχει κάποιο αρχείο log το οποίο περιέχει εγγραφές road_id και segment_id με την αντίστοιχη ip διεύθυνση η οποία αντιστοιχεί στις συγκεκριμένες

συντεταγμένες. Όταν η ip διεύθυνση στην οποία πρέπει να σταλεί το μήνυμα ανευρεθεί, το μήνυμα στέλνεται μέσω του VITPSocket.

Η διαδικασία που εκτελεί το thread αποστολής μηνύματος δεν είναι αυτή που πρέπει να εκτελεί ο εξυπηρετητής. Κανονικά το μήνυμα προς αποστολή πρέπει να σταλεί το VANET στρώμα, το οποίο θα αποφασίσει ποιος είναι ο επόμενος παραλήπτης του μηνύματος και ακολούθως, μέσω ενός αλγόριθμου δρομολόγησης να αποστείλει το μήνυμα. Η διεργασία στην παρούσα εργασία υλοποιήθηκε με αυτό τον τρόπο για σκοπούς δοκιμής και επίδειξης του εξυπηρετητή. Περαιτέρω εργασία πρέπει να γίνει για υλοποίηση του VANET στρώματος το οποίο λειτουργεί κάτω από το VITP στρώμα εφαρμογής. Όταν το πρώτο προδιαγραφεί, τότε εργασία πρέπει να γίνει στον εξυπηρετητή για αλλαγή του τρόπου λειτουργίας του thread αποστολής μηνύματος.

3.2.8 VITP Socket

Το VITP Socket είναι μια βοηθητική κλάση, χρήσιμη μόνο για σκοπούς δοκιμής και επίδειξης του εξυπηρετητή. Περιέχει μια κλάση socket που παρέχεται από τη .NET. Ανά πάσα στιγμή, μπορεί να χρησιμοποιηθεί είτε για παραλαβή μηνύματος, είτε για αποστολή μηνύματος.

Στην παραλαβή κάνει ακρόαση πάνω σε κάποια υποδοχή (port) και όταν κάποιο μήνυμα καταφθάσει, επιστρέφει σε string το μήνυμα. Στην αποστολή, δέχεται σε string το μήνυμα που πρέπει να αποσταλεί και παραμέτρους την IP διεύθυνση αποστολής και το port. Ακολούθως αποστέλλει το μήνυμα.

Κεφάλαιο 4

Υλοποίηση VITP Server

Στο παρών κεφάλαιο περιγράφουμε τη διαδικασία υλοποίησης του εξυπηρετητή, αναφέροντας τη γλώσσα προγραμματισμού και τα εργαλεία που χρησιμοποιήσαμε.

4.1 Γλώσσα Προγραμματισμού

4.1.1 Microsoft .NET Framework

Το .NET Framework της Microsoft [6] είναι ένα λογισμικό συστατικό, μέρος του λειτουργικού συστήματος Microsoft Windows. Αποτελεί συστατικό όλων των λειτουργικών Microsoft Windows από τις εκδόσεις NT 4.0 και 98 και μετά. Προσφέρεται δωρεάν από την εταιρεία Microsoft για προγραμματισμό εφαρμογών οι οποίες τρέχουν στα λειτουργικά της. Περιέχει πολλές χρήσιμες βιβλιοθήκες με έτοιμο κώδικα για επίλυση πολλών συνηθισμένων προβλημάτων και αποτελεί ισχυρό εργαλείο για κάθε προγραμματιστή ο οποίος επιθυμεί να δημιουργήσει εφαρμογές για τα λειτουργικά συστήματα της εταιρείας.

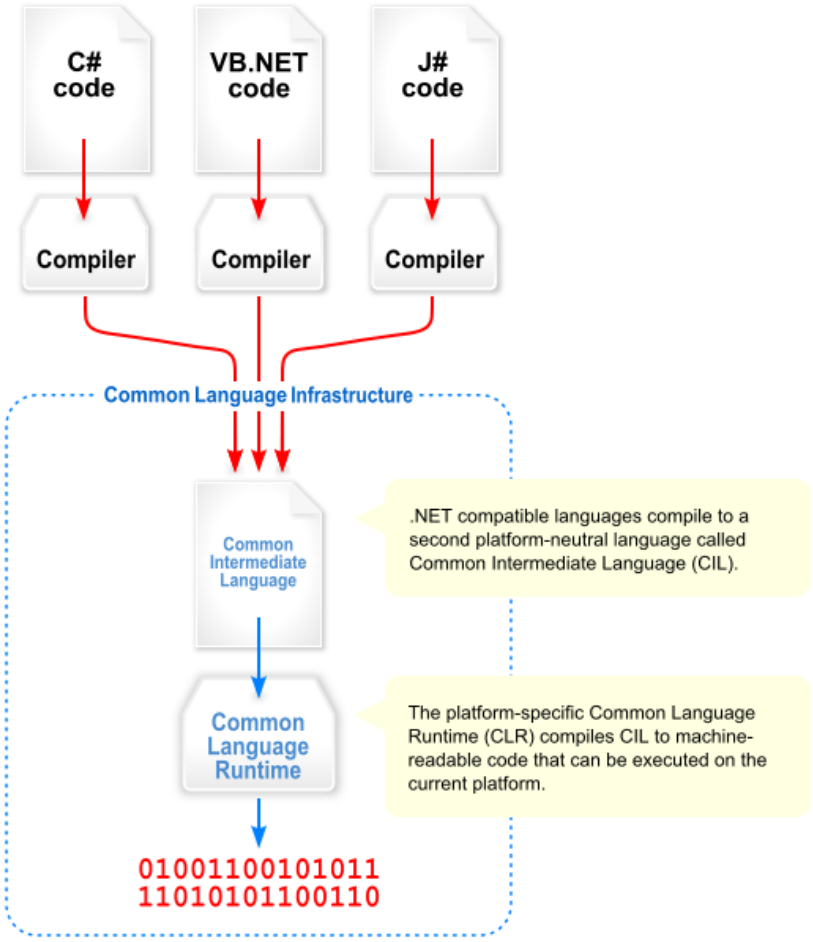
Το .NET Framework δεν περιλαμβάνεται στα Windows XP τα οποία χρησιμοποιήθηκαν για την εκπόνηση της παρούσας εργασίας. Περιλαμβάνεται όμως στο Microsoft Visual Studio 2005, το οποίο είναι ένα IDE για προγραμματισμό πάνω στο .NET Framework. Προσφέρεται επίσης δωρεάν για κατέβασμα από τη Microsoft. Η πιο πρόσφατη του

έκδοση περιλαμβάνεται στην τελευταία έκδοση του λειτουργικού Windows της Microsoft, τα Windows Vista.

Το Framework περιλαμβάνει τις γλώσσες προγραμματισμού Visual Basic, Visual C#, Visual J# και Visual C++. Ο προγραμματιστής μπορεί να δημιουργήσει εφαρμογές σε οποιαδήποτε από αυτές τις γλώσσες και αυτές να τρέξουν χωρίς πρόβλημα συμβατότητας πάνω στο Framework. Αυτό επιτυγχάνεται με χρήση της αρχιτεκτονικής Common Language Infrastructure (CLI).

Η αρχιτεκτονική CLI, προσφέρει μια πλατφόρμα ανεξάρτητη γλώσσας, η οποίο παρέχει λειτουργίες όπως διαχείριση exceptions, garbage collection και ασφάλεια και πάνω στην οποία λειτουργούν οι γλώσσες του .NET Framework. Ο μεταγλωττιστής κάθε γλώσσας, μεταγλωττίζει τον κώδικα σε ένα ενδιάμεσο κώδικα, τον Common Intermediate Language. Ακολούθως, μέσω του Common Language Runtime ο κώδικας μετατρέπεται σε γλώσσα μηχανής και εκτελείται.

Στο Σχήμα 4.1 βλέπουμε καλύτερα τη σημασία του CLI:



Σχήμα 4.1 Γραφική απεικόνιση του CLI

Η έκδοση του .NET Framework που χρησιμοποιήθηκε, είναι η έκδοση 2.0. Οποιαδήποτε εφαρμογή που αναπτύσσεται σε αυτή την έκδοση, είναι συμβατή με τις μεταγενέστερες εκδόσεις του Framework.

Κατόπιν της ανάλυσης του .NET Framework, η επιλογή του είναι προφανής. Αποτελεί μια ώριμη και λειτουργική πλατφόρμα, η οποία παρέχει πολλές βοηθητικές βιβλιοθήκες οι οποίες, όπως θα φανεί παρακάτω, λύνουν τα χέρια του προγραμματιστή σε πολλά προβλήματα που προκύπτουν κατά τη φάση της υλοποίησης.

4.1.2 C++/CLI

Από τις γλώσσες που προσφέρει το .NET Framework, επιλέγηκε σαν γλώσσα υλοποίησης αποκλειστικά η Visual C++ και πιο συγκεκριμένα η Visual C++ 2005.

Ουσιαστικά, η Visual C++ 2005 αποτελεί ένα Integrated Development Environment. Η ονομασία της γλώσσας που χρησιμοποιείται είναι C++/CLI.

Η C++/CLI δεν είναι ακριβώς μια νεότερη έκδοση της παλαιότερης C++. Αποτελεί μια νέα έκδοση της Microsoft η οποία έχει σαν στόχο τη διαδοχή της C++. Παρουσιάζει αλλαγές από τη C++ στο κομμάτι της γραμματικής καθώς και αλλαγές στον τρόπο διαχείρισης της μνήμης και στους δείκτες.

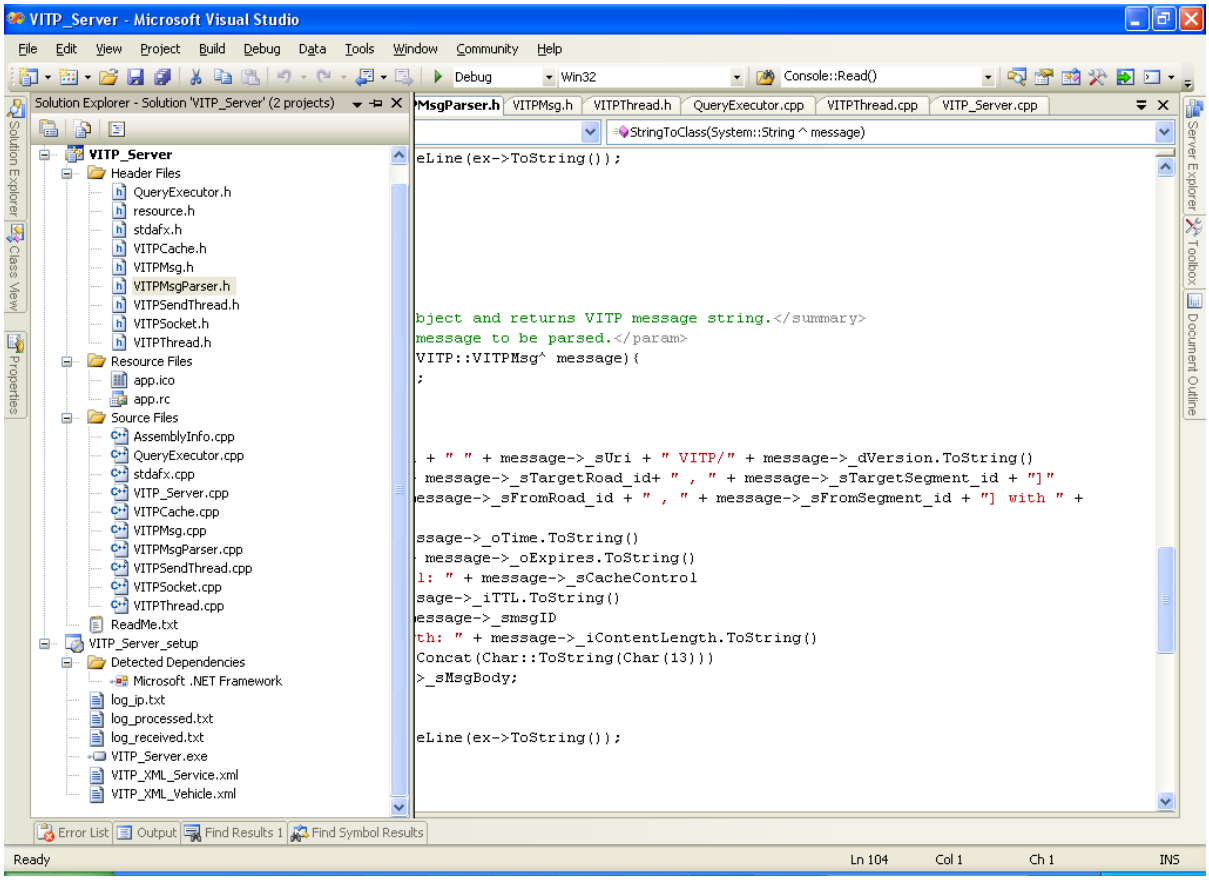
Η συγκεκριμένη γλώσσα επιλέγηκε επειδή αποτελεί τη γλώσσα χαμηλότερου επιπέδου που προσφέρει το .NET Framework. Επειδή στον εξυπηρετητή που υλοποιείται, η ταχύτητα και η μνήμη αποτελούν κρίσιμους παράγοντες, επιλέγηκε υλοποίηση σε γλώσσα του χαμηλότερου επιπέδου η οποία εξακολουθεί να παρέχει τις ευκολίες που παρέχει το .NET Framework. Ένας δεύτερος παράγοντας για επιλογή της συγκεκριμένης γλώσσας, είναι η καλή γνώση και αντίληψη του συγγραφέα σε αυτή.

4.2 Εργαλεία υλοποίησης

Στο παρών υποκεφάλαιο περιγράφονται διάφορα εργαλεία που χρησιμοποιήθηκαν κατά τη φάση της υλοποίησης.

4.2.1 Microsoft Visual Studio 2005

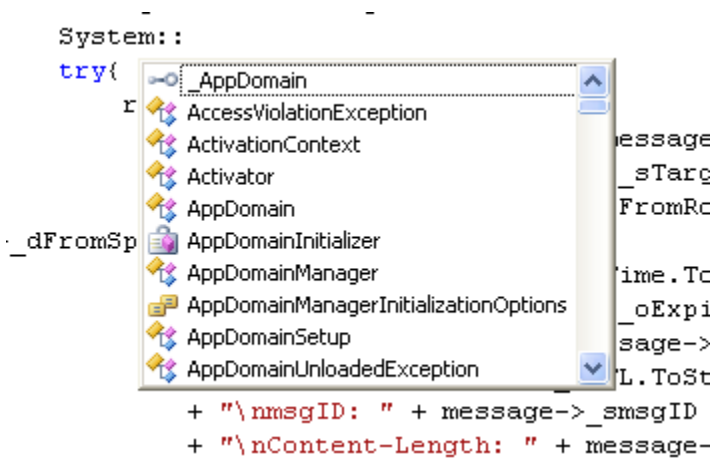
Το Visual Studio 2005 είναι ένα Integrated Development Environment για το .NET Framework. Είναι μια εμπορική εφαρμογή της εταιρείας Microsoft η οποία παρέχεται δωρεάν στους φοιτητές του Πανεπιστημίου Κύπρου. Στο Σχήμα 4.2 βλέπουμε το εργαλείο σε λειτουργία:



Σχήμα 4.2 Η εφαρμογή Visual Studio 2005

Το Visual Studio 2005 παρέχει πολλές ευκολίες στον προγραμματιστή. Κάποιες από αυτές είναι ο δυναμικός συντάκτης κώδικα (code editor), ο οποίος παρέχει syntax highlighting και code completion με χρήση του IntelliSence, debugger για βήμα προς βήμα εκτέλεση του κώδικα, watches και exception reports.

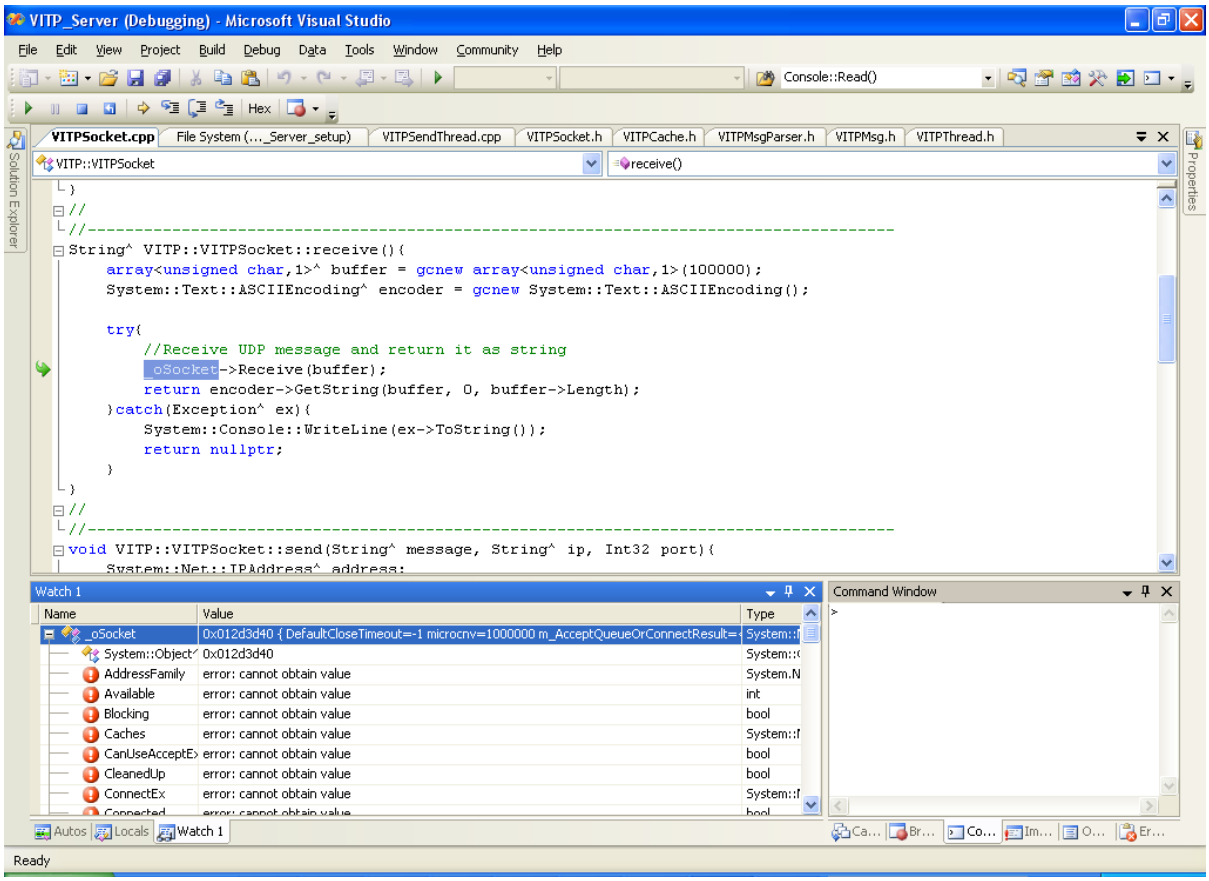
Το IntelliSence (Σχήμα 4.3) είναι μια διευκόλυνση που προσφέρει το Visual Studio. Με πληκτρολόγηση μέρος μιας γνωστής βιβλιοθήκης, κλάσης, μεταβλητής ή μεθόδου, παρουσιάζεται στο χρήστη ένα παράθυρο με τις πιθανές συμπληρώσεις του κώδικα, καθώς και με περιγραφές της λειτουργίας των μεθόδων και των παραμέτρων τους. Αυτό το χαρακτηριστικό επιταχύνει τη συγγραφή κώδικα, ειδικά όταν ο χρήστης αποκτήσει εμπειρία στη χρήση του. Επίσης, προσφέρει μια μέθοδο εξερεύνησης των λειτουργιών κάποιας κλάσης ή βιβλιοθήκης, βοηθώντας το χρήστη να αποφασίσει τον έτοιμο κώδικα που θα χρησιμοποιήσει.



Σχήμα 4.3 Το IntelliSence σε εφαρμογή

Το εργαλείο προσφέρει επίσης δυναμικό Debugger (Σχήμα 4.4), με δυνατότητα παύσης της εκτέλεσης του κώδικα βήμα προς βήμα και παύση της εκτέλεσης του προγράμματος ανά πάσα στιγμή και τοποθέτηση break points για παρακολούθηση συγκεκριμένων κομματιών κώδικα. Επίσης, με χρήση watches παρέχεται παρακολούθηση των τιμών μιας μεταβλητής, μιας κλάσης και γενικά διαφόρων δομών. Τέλος, τα exception reports που παρέχονται κατά την εκτέλεση του debugging βοηθούν στην παρακολούθηση των

exceptions που εγείρονται κατά την εκτέλεση του προγράμματος. Όλα αυτά τα εργαλεία είναι εύκολα στην εξοικείωση και χρήση τους.



Σχήμα 4.4 Ο debugger του Visual Studio 2005

Το Visual Studio 2005 προσφέρει επίσης πολλές άλλες διευκολύνσεις, όπως solution explorer, properties editor, form designer, toolbox και open tabs explorer.

Η επιλογή του εργαλείου, από τη στιγμή που χρησιμοποιήθηκε το .NET Framework, αποτελούσε μονόδρομο. Το IDE που προσφέρει η Microsoft για το Framework της είναι ένα πάρα πολύ καλό εργαλείο, εύκολο στη χρήση ακόμα και από κάποιο προγραμματιστή με μικρή εμπειρία και επιταχύνει κατά πολύ τη συγγραφή του κώδικα. Η εμπειρία του συγγραφέα στο συγκεκριμένο εργαλείο αποτέλεσε επιπλέον παράγοντα στην επιλογή του.

4.2.2 Άλλα βοηθητικά εργαλεία

Για εκπόνηση της εργασίας, χρησιμοποιήθηκαν διάφορες άλλες εφαρμογές, τόσο για σκοπούς άντλησης πληροφοριών και έρευνας, όσο και για της συγγραφή του εγγράφου της.

Ένα μέρος της έρευνας διεξάχθηκε μέσω του διαδικτύου και πολλές πληροφορίες χρήσιμες στη φάση της υλοποίησης αντλήθηκαν από αυτό. Με χρήση του web browser Firefox της εταιρείας Mozilla αντλήθηκαν πολλές πληροφορίες από την ιστοσελίδα Microsoft Development Network [6] και από τη σελίδα Wikipedia [3] [4], οι οποίες χρησίμευσαν στην εμβάθυνση του προγραμματιστή σε διάφορες πτυχές της γλώσσας προγραμματισμού.

Το έγγραφο γράφτηκε με χρήση της Word, η οποία αποτελεί την πιο διαδεδομένη εφαρμογή word processing της εταιρείας Microsoft. Με το συγκεκριμένο εργαλείο, η συγγραφή του εντύπου διευκολύνθηκε σε μεγάλο βαθμό.

Τέλος, οι μετρήσεις που πάρθηκαν έτυχαν επεξεργασίας μέσω της Excel της εταιρείας Microsoft και μέσω του εργαλείου δημιουργήθηκαν οι γραφικές παραστάσεις που θα παρουσιαστούν στη συνέχεια.

4.3 Δομές, βιβλιοθήκες και προγραμματιστικές τεχνικές

Στο παρών υποκεφάλαιο περιγράφονται δομές, βιβλιοθήκες και προγραμματιστικές τεχνικές που χρησιμοποιήθηκαν για την υλοποίηση του εξυπηρετητή.

4.3.1 Δομές και βιβλιοθήκες

Για αποθήκευση τόσο των εισερχόμενων όσο και των εξερχόμενων μηνυμάτων, χρησιμοποιήθηκε η δομή **Queue** της βιβλιοθήκης System.Collections. Η συγκεκριμένη δομή, αποθηκεύει αντικείμενα οποιουδήποτε τύπου, και τα αφαιρεί με τη σειρά που

αποθηκεύτηκαν. Είναι συνεπώς μια δομή first-com first-served. Αυτή είναι η επιθυμητή συμπεριφορά, αφού με αυτό τον τρόπο διαφυλάσσεται η δικαιοσύνη στην επεξεργασία των μηνυμάτων. Όπως είπαμε, δεν ανευρέθηκε κάποιος σημαντικός παράγοντας που να καθορίζει σημαντικότητα ενός τύπου μηνύματος επί άλλων. Για διαφύλαξη του thread safety των δύο αντικειμένων τύπου Queue που χρησιμοποιούνται στην εφαρμογή, χρησιμοποιήθηκε η δομή mutex.

Η δομή **Mutex** είναι κομμάτι της βιβλιοθήκης System.Threading. Αποτελεί ένα εργαλείο διαφύλαξης της ακεραιότητας διαφόρων δομών και προστασία κομματιών κώδικα τα οποία είναι επιρρεπή σε σφάλματα από threads. Το thread το οποίο επιθυμεί να χρησιμοποιήσει κάποιο κώδικα ο οποίος είναι προστατευμένος από κάποιο mutex, καλεί αρχικά τη μέθοδο WaiOne του αντικειμένου. Αν κάποιο άλλο thread έχει δεσμεύσει το mutex, τότε το thread που κάλεσε τη μέθοδο αναμένει (γίνεται suspended), μέχρι το πρώτο να αποδεσμεύσει το mutex. Η αποδέσμευση επιτυγχάνεται με κλήση της μεθόδου ReleaseMutex του αντικειμένου. Στον κώδικα του εξυπηρετητή διασφαλίζεται η μη ύπαρξη deadlock ενός thread. Το σενάριο που μπορεί να οδηγήσει σε deadlock, είναι κάποιο thread να δεσμεύσει το mutex και ακολούθως να εγείρει κάποιο exception πριν αποδεσμεύσει το αντικείμενο. Με χρήση του try-catch-finally, μιας προγραμματιστικής τεχνικής η οποία θα αναλυθεί στη συνέχεια.

Η εκτέλεση των XML Queries γίνεται μέσω της κλάσης **XmlDocument** της βιβλιοθήκης System.XML. Με χρήση των μεθόδων που προσφέρει η κλάση, μπορούμε να εκτελέσουμε XML Queries πάνω σε κάποιο XML αρχείο και να μεταβάλουμε τα περιεχόμενα του αρχείου. Αυτό επιτυγχάνεται με χρήση των μεθόδων SelectSingleNode και ReplaceChild αντίστοιχα. Στην παρούσα έκδοση του εξυπηρετητή, μόνο ένα thread έχει πρόσβαση στο XML αρχείο, η οποία πρόσβαση είναι μόνο ανάγνωση των περιεχομένων. Έχουν όμως υλοποιηθεί συναρτήσεις για γράψιμο στο αρχείο, για μελλοντική εισαγωγή δεδομένων που στέλνουν οι αισθητήρες του οχήματος. Συνεπώς, εγείρεται ανάγκη ελέγχου ταυτοχρονίας τόσο κατά την ανάγνωση όσο και κατά την εγγραφή στο αρχείο. Χρησιμοποιήθηκε και σε αυτή την περίπτωση η κλάση mutex. Συγκεκριμένα, χρησιμοποιείται το ίδιο mutex στα δύο σημεία του κώδικα όπου γίνεται

εγγραφή και ανάγνωση, διασφαλίζοντας ότι δεν μπορούν παραπάνω από ένα thread να γράφουν στο αρχείο και ότι δεν μπορεί να γίνει εγγραφή όταν γίνεται ανάγνωση και το αντίθετο.

Οι κυριότερες βιβλιοθήκες που χρησιμοποιήθηκαν, είναι η **System**, η **System.Threading** και η **System.Xml**. Από τη System χρησιμοποιήθηκαν οι κλάσεις των βασικών τύπων μεταβλητών. Τύποι όπως String, Int32 και array ανήκουν στη βιβλιοθήκη System. Ακόμα και σαν βασικοί τύπο, για χρήση τους απαιτείται δημιουργία αντικειμένου. Παρέχονται πολλές χρήσιμες μέθοδοι στα αντικείμενα των βασικών τύπων για διαχείριση τους.

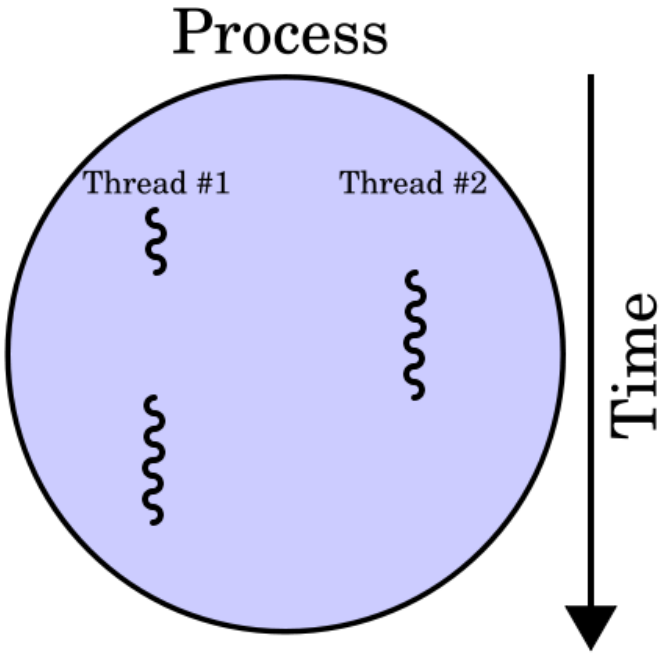
Η βιβλιοθήκη System.Threading χρησιμοποιήθηκε για δημιουργία και διαχείριση των threads τα οποία τρέχουν στον εξυπηρετητή. Επίσης, χρησιμοποιήθηκε η κλάση mutex, η οποία ανήκει στην παρούσα βιβλιοθήκη, για έλεγχο ταυτοχρονίας.

Στη System.Collections περιέχονται δομές για αποθήκευση δεδομένων με διάφορους τρόπους. Από αυτή τη βιβλιοθήκη χρησιμοποιήθηκαν οι δομές Queue και Hashtable για αποθήκευση των μηνυμάτων. Η δομή Hashtable χρησιμοποιήθηκε για κατασκευή της cache μνήμης του εξυπηρετητή.

4.3.2 Προγραμματιστικές τεχνικές

Στην υλοποίηση του εξυπηρετητή δε χρησιμοποιήθηκαν περίπλοκοι αλγόριθμοι ή εξεζητημένες προγραμματιστικές τεχνικές. Η πορεία της υλοποίησης είχε σαν κεντρικό στόχο την απλότητα. Ο λόγος είναι ότι η ταχύτητα είναι σημαντικός παράγοντας στη λειτουργία του εξυπηρετητή και ότι δεν απαιτούνται περίπλοκοι υπολογισμοί για ορθή λειτουργία του εξυπηρετητή. Βεβαίως, δε υλοποιήθηκε μετατροπή συντεταγμένων GPS σε συντεταγμένες VITP ούτε χρησιμοποιείται κάποιος αλγόριθμος δρομολόγησης για αποστολή των μηνυμάτων. Υλοποίηση των πιο πάνω συνεπάγεται σε υλοποίηση σχετικά περίπλοκων αλγορίθμων, οι οποίοι ίσως επιβραδύνουν αρκετά τον εξυπηρετητή. Σε μελλοντικές εργασίες πάνω στο πρωτόκολλο VITP και στην εφαρμογή του οι παράγοντες που αναφέρθηκαν πρέπει να ληφθούν υπόψη.

Βασική πραγματιστική τεχνική που χρησιμοποιήθηκε είναι ο πολυνηματικός προγραμματισμός (Σχήμα 4.5) και κατά συνέπεια ο έλεγχος ταυτοχρονίας. Στον εξυπηρετητή τρέχουν παράλληλα τρία threads, το καθένα από τα οποία εκτελεί διαφορετική εργασία. Υπάρχουν ξεχωριστά threads για παραλαβή μηνυμάτων, επεξεργασία μηνυμάτων και αποστολή μηνυμάτων. Αυτό βοηθά στην αποδοτική λειτουργία του εξυπηρετητή. Αν για παράδειγμα ο εξυπηρετητής λάβει κάποιο μήνυμα, το επεξεργαστεί αλλά δεν μπορεί να το στείλει λόγω κάποιας τεχνικής βλάβης ή λόγω συνωστισμού του δικτύου, δε θα σταματήσει την επεξεργασία των μηνυμάτων αναμένοντας να γίνει αποστολή του μηνύματος. Αντιθέτως, κάθε απάντηση αποθηκεύεται στην ουρά από το thread επεξεργασίας και ξεχωριστό thread είναι υπεύθυνο για την αποστολή του. Αν από την άλλη η επεξεργασία ενός μηνύματος καταναλώνει χρόνο πέραν του ρυθμού παραλαβής μηνυμάτων, το thread παραλαβής μηνυμάτων συνεχίζει να παραλαμβάνει και δεν περιμένει να γίνει επεξεργασία ενός μηνύματος πριν την παραλαβή ενός άλλου. Αυτή είναι η κεντρική ιδέα του πολυνηματικού προγραμματισμού, ο καταμερισμός δηλαδή της εργασίας που πρέπει να γίνει για εκπόνηση ενός έργου.



Σχήμα 4.5 Καταμερισμός χρόνου επεξεργαστή σε πολυνηματική εφαρμογή

Χρήση πολυνηματικού προγραμματισμού συνεπάγεται χρήση ελέγχου ταυτοχρονίας. Μη χρήση αυτής της τεχνικής εγκυμονεί κινδύνους μη εγκυρότητας δεδομένων σε κάποια δομή. Αυτό είναι δυνατό να συμβεί αν για παράδειγμα κάποιο thread επιχειρήσει να διαβάσει κάποια δεδομένα από ένα χώρος μνήμης στον οποίο γράφει κάποιο άλλο. Τα threads μπαίνουν και βγαίνουν από τον επεξεργαστή σε απρόβλεπτους χρόνους. Συνεπώς, αν το thread που διαβάζει τα δεδομένα διαβάσει ένα κομμάτι του δεδομένου, βγει από τον επεξεργαστή και ακολούθως κάποιο άλλο thread γράψει στο συγκεκριμένο χώρο μνήμη, τότε το thread που διαβάζει θα διαβάσει μη έγκυρα δεδομένα. Ο έλεγχος ταυτοχρονίας επιτυγχάνεται με χρήση δομής mutex. Χρήση έλεγχου ταυτοχρονίας όμως εγκυμονεί κυνδίνους αδιεξόδου (deadlock).

Το αδιέξοδο (deadlock) ενός νήματος (thread) μπορεί να συμβεί αν κάποιο thread δεσμεύσει κάποιο mutex και στη συνέχεια εγείρει exception χωρίς να αποδεσμεύσει το mutex. Αυτό θα έχει ως αποτέλεσμα το mutex να μην αποδεσμευτεί ποτέ. Γενικά δεν επιθυμούμαι να εγερθούν exceptions σε κάποιο thread χωρίς να υπάρχει έλεγχος, επειδή με έγερση exception το thread πεθαίνει. Όλα όμως τα thread πρέπει να είναι ζωντανά για ορθή λειτουργία του εξυπηρετητή. Αυτό αποτρέπεται με τη χρήση του try-catch-finally. Υπό την εντολή try τοποθετείται ο κώδικας που θέλουμε να τρέξουμε ο οποίος μπορεί να εγείρει κάποιο exception. Υπό την εντολή catch τοποθετείται ο κώδικας ο οποίος επιθυμούμε να εκτελεστεί εάν κάποιο exception εγερθεί. Συνήθως, υπό την εντολή catch υπάρχει εκτός των άλλων εντολή για αναφορά του exception, επειδή ακόμη και αν το exception δε σκοτώσει το thread, επιθυμούμε να γνωρίζουμε ότι υπήρξε κάποιο σφάλμα στην εκτέλεση του κώδικα. Τέλος, σημαντική εντολή για αποφυγή του deadlock είναι η εντολή finally. Υπό τη συγκεκριμένη εντολή τοποθετείται κώδικας, ο οποίος θα εκτελεστεί ανεξάρτητα ύπαρξης exception. Σε αυτό το σημείο τοποθετήσαμε τον κώδικα για αποδέσμευση του mutex. Συνεπώς, ακόμη και αν υπάρξει exception, το thread θα συνεχίσει τη λειτουργία του και το mutex θα αποδεσμευτεί.

Κεφάλαιο 5

Πειράματα και μετρήσεις

Στο παρών κεφάλαιο, διεξάγουμε κάποια πειράματα και παίρνουμε διάφορες μετρήσεις, οι οποίες θα μας βοηθήσουν να εξάγουμε συμπεράσματα για την υλοποίηση του εξυπηρετητή.

5.1 Μετρικές

Σημαντικό κομμάτι των πειραμάτων μας είναι η επιλογή των σωστών μετρικών. Με τον όρο «σωστές μετρικές» εννοούμε μετρικές οι οποίες να μπορούν να μας βοηθήσουν να φτάσουμε σε κάποια συμπεράσματα αναφορικά με την εργασία μας.

Η ιδιαιτερότητα του εξυπηρετητή δυσκόλεψε την επιλογή σωστών μετρικών. Κεντρικός άξονας στην απόφαση των μετρικών που χρησιμοποιήσαμε ήταν κυρίως οι απαιτήσεις που έχουμε από τον VITP εξυπηρετητή που κατασκευάσαμε. Σκεπτόμενοι ότι ο εξυπηρετητής βρίσκεται πάνω σε κινούμενο όχημα, είναι προφανές ότι απαιτείται ταχύτατη εξυπηρέτηση των μηνυμάτων που καταφθάνουν, αφού κάθε μήνυμα που αποστέλλεται από κάποιο αυτοκίνητο πρέπει να τύχει επεξεργασίας και να αποσταλεί

πριν το όχημα απομακρυνθεί από το σημείο που βρισκόταν όταν έστειλε το μήνυμα. Αν μπορέσουμε να πετύχουμε αυτό το στόχο, η δρομολόγηση του μηνύματος θα διευκολυνθεί. Επίσης, σημαντικός παράγοντας ο οποίος ενισχύει την απαίτηση ο χρόνος εξυπηρέτησης να είναι μικρός, είναι ότι τα alerts πρέπει να τυγχάνουν άμεσης επεξεργασίας, ούτως ώστε να διασφαλίζεται η ασφάλεια του οδηγού.

Σε αυτό το σημείο τίθεται το ερώτημα ποια είναι η έννοια της εξυπηρέτησης μηνυμάτων. Πρώτη σκέψη είναι ότι ο χρόνος εξυπηρέτησης είναι η χρονική διάρκεια που περνά από τη στιγμή που κάποιο μήνυμα παραλαμβάνεται μέχρι τη στιγμή που αυτό αποστέλλεται πίσω στο δίκτυο. Θα ονομάσουμε αυτή τη μετρική **service duration** και θα τη μετρούμε σε milliseconds. Σε μετέπειτα στάδιο, αυτή η μετρική μπορεί να συνδυαστεί με τη διάρκεια αποστολής του μηνύματος απάντησης στον παραλήπτη του, ούτως ώστε να εξαχθούν τελικά συμπεράσματα σχετικά με την αποδοτικότητα του πρωτοκόλλου VITP. Προς το παρών, παίρνουμε την παρούσα μέτρηση αφού απαιτείται περαιτέρω εργασία για κατασκευή ενός ολοκληρωμένου συστήματος το οποίο θα μπορεί να δοκιμαστεί υπό πραγματικές συνθήκες.

Όμως, δεν απαιτούν όλα τα μηνύματα αποστολή απάντησης. Κάποια μηνύματα είναι τύπου POST, και επεξεργασία τους σημαίνει απλά εισαγωγή στη cache. Συνεπώς, απαιτείται μια δεύτερη μετρική, η οποία μετρά το χρόνο από τη στιγμή που το μήνυμα καταφθάνει, μέχρι τη στιγμή που το thread επεξεργασίας ολοκληρώνει την εργασία του πάνω στο συγκεκριμένο μήνυμα. Θα ονομάσουμε αυτή τη μετρική **process duration** και θα τη μετρούμε σε milliseconds. Αυτή η μετρική είναι ίσως πιο βοηθητική από την προηγούμενη στο παρών στάδιο για εξαγωγή συμπερασμάτων. Αναιρώντας τον παράγοντα του δικτύου, μπορούμε να αναλύσουμε ουσιαστικά το χρόνο που χρειάζεται ο εξυπηρετητής μας να εκτελέσει τη βασική αποστολή του, η οποία είναι επεξεργασία μηνυμάτων. Να σημειωθεί και πάλι ότι στο παρών στάδιο, οι μετρήσεις δεν είναι πλήρως ρεαλιστικές. Αυτό συμβαίνει λόγω του γεγονότος ότι δεν υλοποιήθηκαν το συστατικό μετατροπής GPS συντεταγμένων σε συντεταγμένες VITP και το συστατικό παραλαβής μετρήσεων από τους αισθητήρες του αυτοκινήτου και εγγραφής στο XML αρχείο. Προβλέπεται ότι, κατόπιν υλοποίησης των πιο πάνω, οι χρόνοι που θα παρθούν από τα

πειράματα θα είναι πολύ μεγαλύτεροι. Σε τελική ανάλυση, η παρούσα μετρική θα βοηθήσει το συγγραφέα να καταλήξει στο συμπέρασμα κατά πόσον η εργασία που έγινε είχε επιτυχή αποτελέσματα.

Η μετρική της προηγούμενης παραγράφου θα χρησιμοποιηθεί για μέτρηση του **throughput** του εξυπηρετητή. Το throughput θα μετριέται σε μηνύματα ανά δευτερόλεπτο. Η συγκεκριμένη μετρική διαφέρει κάπως από τη συνώνυμη η οποία μετριέται στα δίκτυα. Στα δίκτυα, το throughput έχει την έννοια μέσου ρυθμού επιτυχούς αποστολής μηνυμάτων. Στον εξυπηρετητή που υλοποιήθηκε, έχει την έννοια μέσου ρυθμού επιτυχούς επεξεργασίας μηνυμάτων. Ο συγγραφέας κρίνει την ονομασία της μετρικής ορθή αλλά προσοχή πρέπει να δοθεί για αποφυγή παρεξηγήσεων.

Μια τελευταία μετρική, της οποίας η χρησιμότητα είναι ίσως διφορούμενη, είναι ο αριθμός των εγγραφών στο δίσκο. Θα ονομαστεί **disk writes** και θα μετριέται σε bytes. Να σημειωθεί ότι εγγραφές στο δίσκο γίνονται όταν θα γίνει κάποια εγγραφή σε log αρχείο. Το μέγεθος της εγγραφής είναι πάντα σταθερό. Στο αρχείο log_received το μέγεθος είναι 178 bytes ενώ στο αρχείο log_processed είναι 182 bytes. Η μετρική disk writes χρησιμεύει στην εξαγωγή συμπερασμάτων όσον αφορά τις απαιτήσεις του εξυπηρετητή σε δευτερεύουσα μνήμη.

5.2 Μέθοδος εξαγωγής αποτελεσμάτων (Πρώτη μέθοδος)

Για διεξαγωγή των πειραμάτων χρησιμοποιήθηκαν δύο υπολογιστές συνδεδεμένοι μεταξύ τους μέσω Ethernet πάνω σε router. Οι υπολογιστές είναι και οι δύο Pentium 4 3,2 GHz με 1GB RAM. Επιλέχθηκαν οι συγκεκριμένοι υπολογιστές κυρίως επειδή είναι περιορισμένης ισχύος σε σχέση με τους υπολογιστές που κυκλοφορούν σήμερα και έτσι είναι ίσως πιο κοντά στους υπολογιστές που ίσως εφαρμοστούν πάνω στα οχήματα.

Στόχος μας ήταν η αποστολή μηνυμάτων από τον ένα υπολογιστή ο οποίος λειτουργούσε σαν client στον άλλο υπολογιστή ο οποίος λειτουργούσε σαν host. Για να είναι αυτό εφικτό, χρειάστηκε να υλοποιηθεί εφαρμογή η οποία να στέλνει μηνύματα με κάποιο

ρυθμό μέσω δικτύου. Για τους σκοπούς των πειραμάτων υλοποιήθηκε αυτή η εφαρμογή και ονομάστηκε VITP Message Sender. Η συγκεκριμένη εφαρμογή διαβάζει μηνύματα από κάποιο αρχείο και τα στέλνει με συγκεκριμένο ρυθμό, τον οποίο καθορίζει ο χρήστης. Συγκεκριμένα, ο ρυθμός καθορίζεται με εισαγωγή της χρονικής απόστασης μεταξύ αποστολής ενός μηνύματος από το προηγούμενο. Ο χρήστης γράφει στο αρχείο τα μηνύματα τα οποία θέλει να αποστείλει και προσδιορίζει τη χρονική απόσταση σε milliseconds καθώς και την ip διεύθυνση στην οποία στέλνονται τα μηνύματα.

Για υπολογισμό του process duration, χρησιμοποιήθηκαν οι εγγραφές των log αρχείων log_received και log_processed. Τα αρχεία περιέχουν την ακριβή χρονική στιγμή που κάποιο μήνυμα λήφθηκε (log_received) και την ακριβή χρονική στιγμή που ολοκληρώθηκε η επεξεργασία κάποιου μηνύματος (log_processed). Ο χρόνος καταγράφεται με ακρίβεια δεκάτων του millisecond, ακρίβεια η οποία θεωρείται ικανοποιητική. Εγγραφή στο αρχείο log_received γίνεται αμέσως μετά την παραλαβή ενός μηνύματος από το socket. Εγγραφή στο αρχείο log_processed γίνεται αμέσως μετά την ολοκλήρωση επεξεργασίας ενός μηνύματος από το thread επεξεργασίας μηνυμάτων.

Το throughput υπολογίστηκε αφού ολοκληρώθηκε ο υπολογισμός του process duration. Στέλναμε σε κάθε πείραμα x αριθμό μηνυμάτων με κάποιο ρυθμό r. Υπολογίζαμε για κάθε ρυθμό αποστολής το μέσο όρο διάρκειας επεξεργασίας t και με χρήση του τύπου $throughput = \frac{1000}{t}$. Το throughput μετρήθηκε σε μηνύματα ανά δευτερόλεπτο. Θεωρήθηκε ορθό η μονάδα μέτρησης να μην είναι σε bits ανά δευτερόλεπτο αφού το μέγεθος κάθε μηνύματος δεν είναι σταθερό.

Για μέτρηση του service duration χρησιμοποιήθηκε μα παραλλαγμένη έκδοση του εξυπηρετητή. Κατόπιν παραλαβής ενός μηνύματος, εάν αυτό είναι τύπου GET καταγραφόταν σε αρχείο η χρονική στιγμή που παραλήφθηκε το μήνυμα. Ακολούθως, κατόπιν αποστολής του μηνύματος, καταγραφόταν και πάλι η χρονική στιγμή σε αρχείο. Ο εξυπηρετητής που παραδίδεται δεν εκτελεί αυτή τη λειτουργία, απλά χρησιμοποιήθηκε

η παραλλαγμένη αυτή έκδοση για το πείραμα. Ουσιαστικά αφαιρέθηκε μια I/O λειτουργία και προστέθηκε μια άλλη, για να μην υπάρξει παραπάνω καθυστέρηση.

Για μέτρηση των εγγραφών στο δίσκο, χρησιμοποιήθηκαν τα αρχεία log_received και log_processed. Μετρήθηκαν οι εγγραφές w στο κάθε αρχείο και το μέγεθος s του string που γραφόταν σε κάθε αρχείο σε χαρακτήρες. Ακολούθως, επειδή γράφονταν Unicode χαρακτήρες, το μέγεθος κάθε εγγραφής πολλαπλασιάστηκε επί δύο, όσα bytes είναι κάθε Unicode χαρακτήρας. Ο τύπος που χρησιμοποιήθηκε είναι $disk - writes = w \times s \times 2$.

5.3 Διεξαγωγή πειραμάτων (Πρώτη μέθοδος)

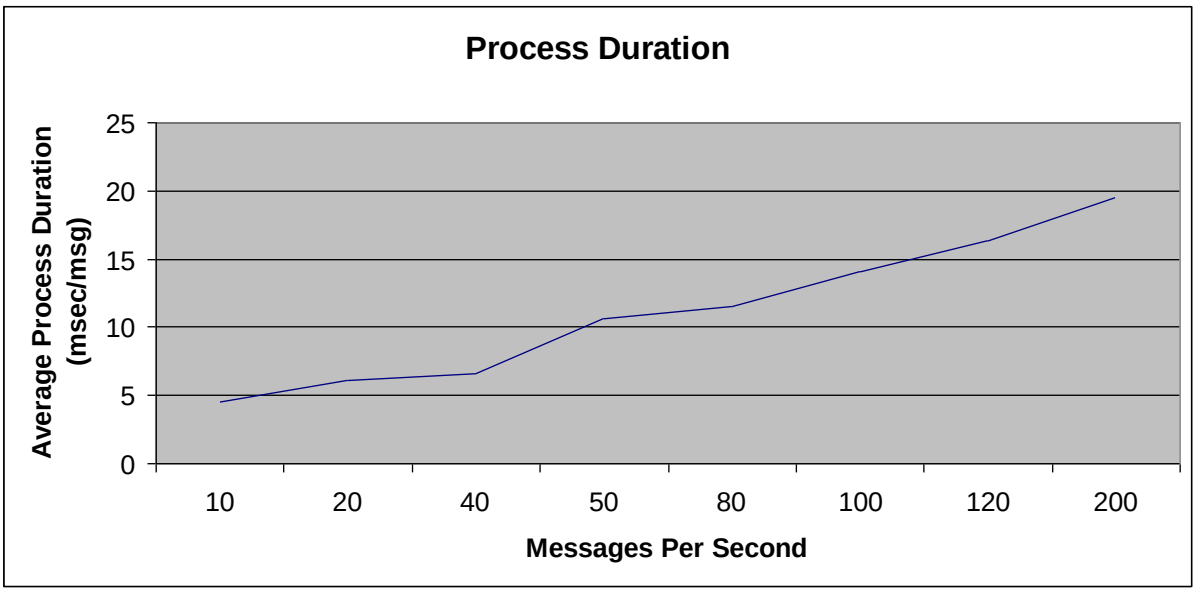
Στο αρχείο μηνυμάτων του VITP Message Sender τοποθετήσαμε 1000 μηνύματα. Το 30% των μηνυμάτων αυτών ήταν μηνύματα τύπου POST και το 70% ήταν μηνύματα τύπου GET. Εκκινούσαμε τον εξυπηρετητή στον ένα υπολογιστή και στέλναμε τα μηνύματα στον άλλο με διάφορους ρυθμούς. Οι ρυθμοί που χρησιμοποιήθηκαν σε μηνύματα ανά δευτερόλεπτο ήταν 10, 20, 40, 50, 80, 100, 120 και 200.

Αφού ολοκληρωνόταν το κάθε πείραμα, αντιγράφαμε τις μετρήσεις στην Excel και τις επεξεργαζόμασταν. Μετά την επεξεργασία δημιουργήσαμε τις γραφικές για κάθε μετρική. Ο τύπος των γραφικών που χρησιμοποιήσαμε ήταν line.

Να σημειωθεί ότι για να μετρήσουμε τα disk writes ακολουθήσαμε άλλη προσέγγιση. Στέλναμε κάθε φορά συγκεκριμένο αριθμό μηνυμάτων και ακολούθως μετρούσαμε τις εγγραφές. Στείλαμε σε ξεχωριστά πειράματα 100, 200, 300, 400, 500, 600, 700, 800, 900 και 1000 μηνύματα.

5.4 Παρουσίαση αποτελεσμάτων (Πρώτη μέθοδος)

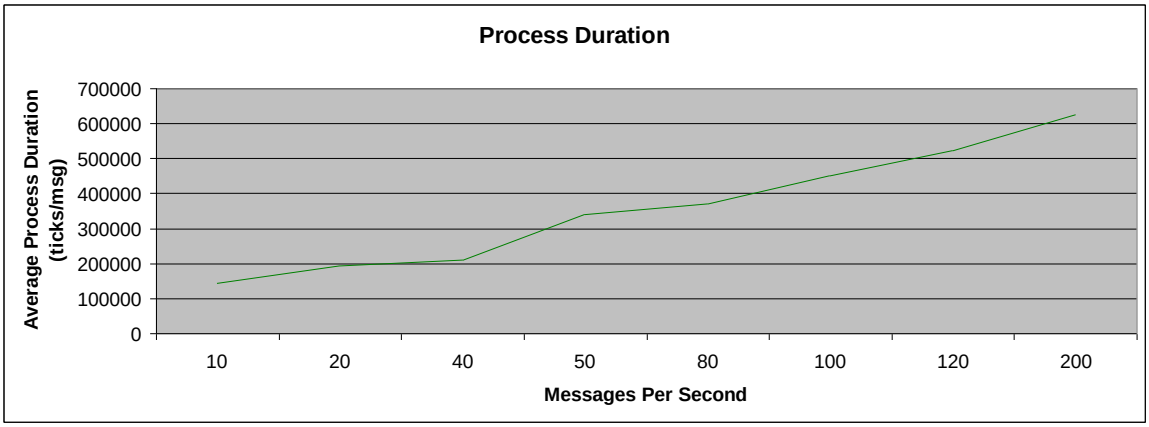
Στο Σχήμα 5.1α παρουσιάζεται η γραφική παράσταση για το **process duration**:



Σχήμα 5.1α Γραφική παράσταση χρόνου επεξεργασίας μηνύματος

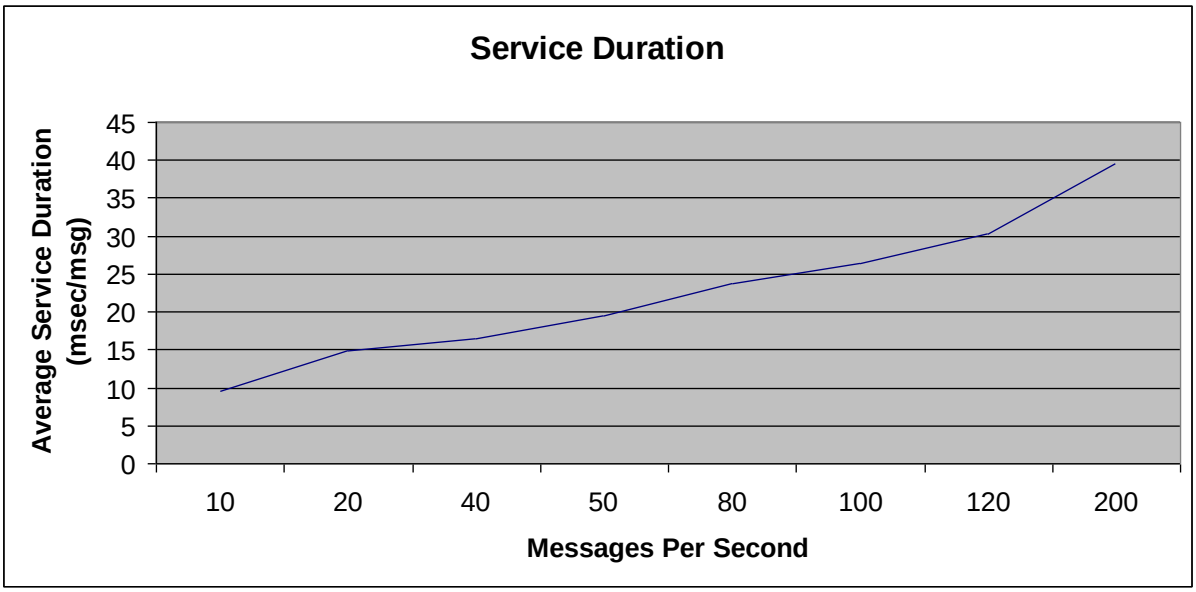
Παρατηρούμε μια σταθερή άνοδο στο χρόνο επεξεργασίας κάθε μηνύματος όσο αυξάνεται ο ρυθμός αποστολής μηνυμάτων στον εξυπηρετητή. Αυτό αρχικά μπορεί να φαίνεται λογικό, όμως δεν παρατηρούμε κάποια σταθερότητα για κάποιο εύρος ρυθμών αποστολής, κάτι που ίσως μαρτυρεί πρόβλημα στη μέθοδο εξαγωγής αποτελεσμάτων.

Στο Σχήμα 5.1β παραθέτουμε την ίδια γραφική, μετρώντας αυτή τη φορά σε κύκλους επεξεργαστή ανά μήνυμα αντί για δευτερόλεπτα ανά μήνυμα. Με αυτό τον τρόπο θα μπορεί ίσως να υπολογιστούν μελλοντικά οι μετρήσεις που πάρθηκαν πάνω σε διαφορετικά μηχανήματα.



Σχήμα 5.1β Γραφική παράσταση χρόνου επεξεργασίας μηνύματος (ticks/msg)

Ακολούθως, στο Σχήμα 5.2α παρουσιάζουμε τη γραφική παράσταση του **service duration**:

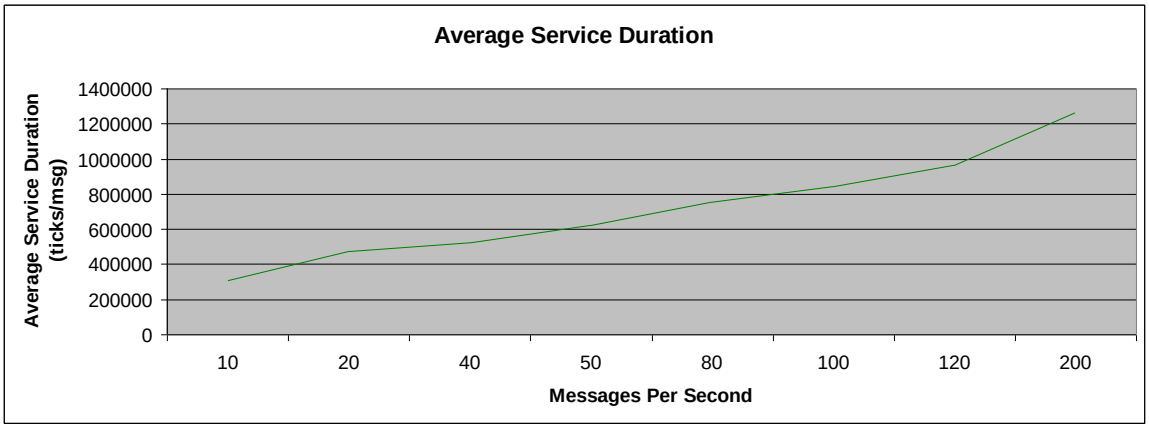


Σχήμα 5.2α Γραφική παράσταση χρόνου εξυπηρέτησης

Παρατηρούμε τιμές σχεδόν διπλάσιες από το process duration. Οι τιμές και πάλι αυξάνονται λιγότερο από αναλογικά σε σχέση με την αύξηση του ρυθμού αποστολής. Παρατηρούμε παρόμοια προβλήματα με την προηγούμενη μετρική, κάτι που μας

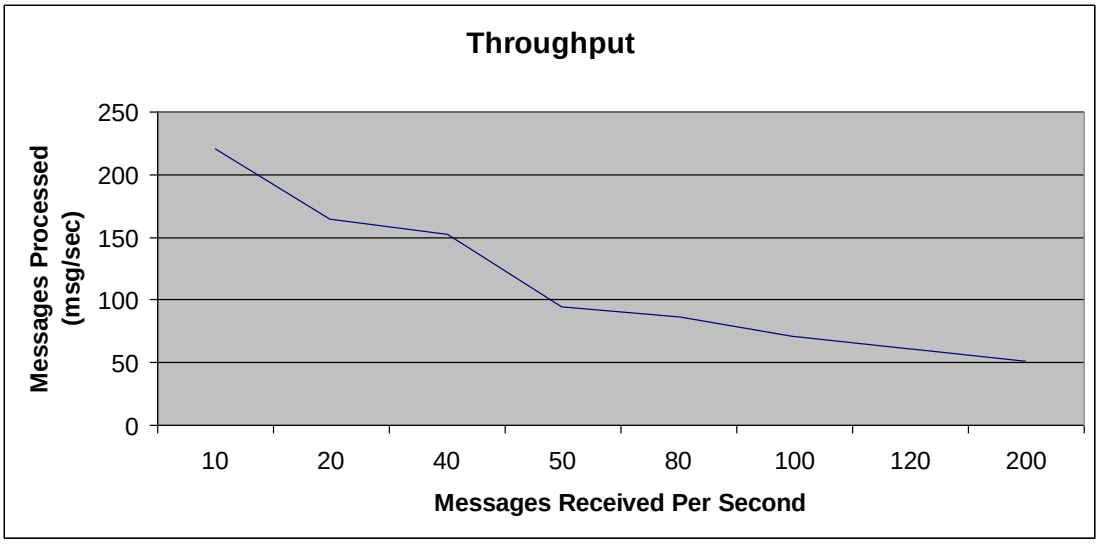
υποχρεώνει να ψάξουμε διαφορετική μέθοδο εξαγωγής αποτελεσμάτων για τις δύο αυτές μετρικές.

Στο Σχήμα 5.2β παραθέτουμε και πάλι την ίδια γραφική παράσταση μετρώντας και πάλι σε κύκλους επεξεργαστή ανά μήνυμα, για τους λόγους που εξηγήσαμε πιο πάνω.



Σχήμα 5.2β Γραφική παράσταση χρόνου εξυπηρέτησης

Ακολούθως, στο Σχήμα 5.3 παρουσιάζουμε τη γραφική του **throughput**:

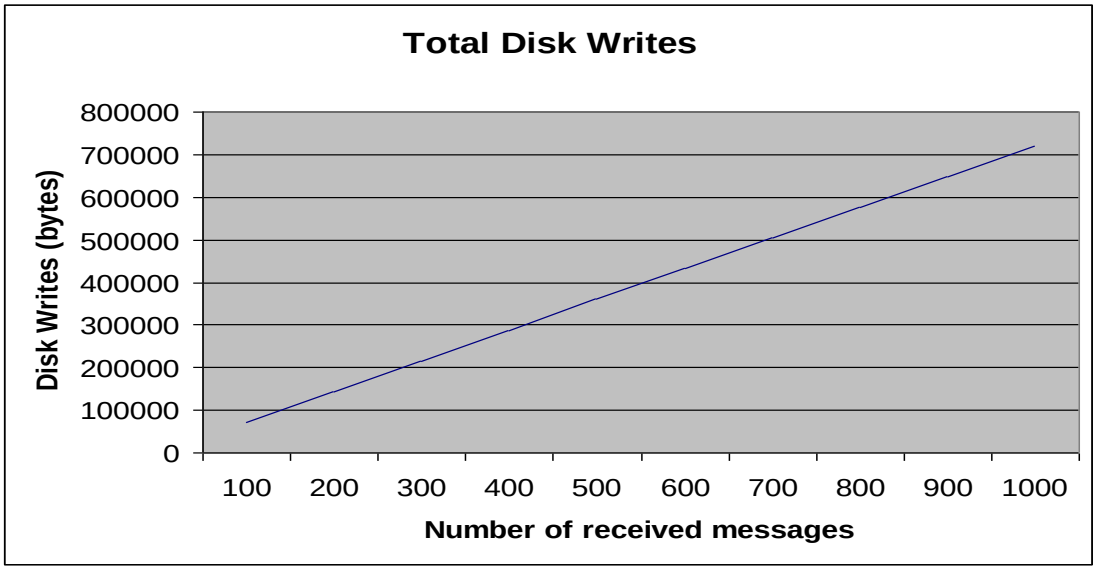


Σχήμα 5.3 Γραφική παράσταση throughput

Το throughput είναι αντιστρόφως ανάλογο του process duration. Παρατηρούμε μείωση του throughput όταν αυξάνεται ο ρυθμός αποστολής μηνυμάτων. Όντας συνάρτηση του

process duration, η μέτρηση του throughput πρέπει να ξανά υπολογιστεί, αφού καταλήξαμε ότι η μέθοδος εξαγωγής αποτελεσμάτων ίσως είναι προβληματική.

Τέλος, στο Σχήμα 5.4 παρουσιάζουμε τη γραφική για τα **disk writes**:



Σχήμα 5.4 Γραφική παράσταση εγγραφών στο δίσκο

Γνωρίζαμε από πριν πως θα φαινόταν αυτή η γραφική. Αφού για κάθε μήνυμα που παραλαμβάνεται, γίνονται δύο εγγραφές στο δίσκο, οι εγγραφές αυξάνονται ανάλογα με την αύξηση του αριθμού των μηνυμάτων που αποστέλλονται. Αυτή η μέτρηση πάρθηκε για να σημειωθεί ότι με συνεχή παραλαβή μηνυμάτων η μνήμη γεμίζει σιγά σιγά.

Ακολούθως, παραθέτουμε τους μέσους όρους κάθε μετρικής:

Μετρική	**Μέσος όρος**	**Μέσος όρος (ticks/msg)**
Process Duration	11,1621 msec/msg	357188
Service Duration	22,5065 msec/msg	720208
Throughput	112,6825864 msg/sec	-
Disk Writes	396 kbytes	-

Πίνακας 5.1 Μέσοι όροι μετρήσεων

5.5 Μέθοδος εξαγωγής αποτελεσμάτων (Δεύτερη μέθοδος)

Παρατηρήσαμε πριν κάποια προβλήματα στις μετρήσεις των process duration και service duration. Φυσιολογικά, αναμέναμε για κάποιο εύρος τιμών ρυθμού αποστολής μηνυμάτων να υπάρχει κάποια σταθερότητα στις μετρήσεις μας. Αυτό όμως δεν παρατηρείται στις μετρήσεις μας.

Το πρόβλημα οφείλεται στο γεγονός ότι οι μετρήσεις που παίρνουμε είναι αποκλειστικά από εγγραφές σε αρχεία κατά τη διάρκεια της κανονικής λειτουργίας του εξυπηρετητή. Όποτε λαμβάναμε κάποιο μήνυμα, καταγράφαμε τη χρονική στιγμή παραλαβής του σε αρχείο. Το ίδιο κάναμε όταν τελείωνε η εξυπηρέτηση του μηνύματος καθώς και όταν το μήνυμα στελνόταν στο δίκτυο μέσω του socket. Όμως, η συνεχής καταγραφή αποτελεσμάτων σε αρχείο είναι ακριβή διαδικασία, αφού κάθε φορά το αρχείο φορτώνεται στη μνήμη, μεταβάλλεται και ακολούθως αποθηκεύεται στη δευτερεύουσα μνήμη. Επίσης, στον πολυνηματικό προγραμματισμό, όταν κάποιο νήμα επιχειρήσει εργασία εισόδου/εξόδου, το λειτουργικό σύστημα αυτόματα παγώνει το νήμα και το βγάζει αυτόματα από τον επεξεργαστή για να εξυπηρετήσει άλλο νήμα. Αυτό είναι επηρεάζει σε μεγάλο βαθμό τις μετρήσεις μας.

Το πρόβλημα που αναφέραμε μας υποχρέωσε να ψάξουμε για άλλη μέθοδο καταγραφής αποτελεσμάτων. Καταλήξαμε σε μια διαφορετική προσέγγιση, όπου το σύστημα θα καταγράφει συνεχώς τους χρόνους που θέλουμε και θα τους αποθηκεύει στη μνήμη του. Ακολούθως, όταν το πείραμα ολοκληρωθεί, μόνο τότε θα γίνεται αποθήκευση σε αρχείο.

Για να πετύχουμε την επιθυμητή μέθοδο, κάναμε κάποιες αλλαγές στον κώδικα του εξυπηρετητή. Αρχικά προσθέσαμε κάποιες μεταβλητές στην κλάση VITPMsg. Οι μεταβλητές ήταν χρόνος αναμονής στην ουρά παραλαβής (_dWaitQueueTime), χρόνος επεξεργασίας μηνύματος (_dProcessTime), χρόνος αναμονής στην ουρά αποστολής (_dForwardQueueTime) και μια προσωρινή μεταβλητή τύπου DateTime (_oTempTime).

Για να γίνει καλύτερα αντιληπτή η σημασία κάθε μεταβλητής, ας παρατηρήσουμε το Σχήμα 5.5:

Σχήμα 5.5 Αναπαράσταση δομών από τις οποίες περνά το μήνυμα

Ο χρόνος αναμονής στην ουρά παραλαβής μετριέται ως η χρονική απόσταση από το σημείο 1 μέχρι το σημείο 2. Ο χρόνος επεξεργασίας μετριέται ως η χρονική απόσταση από το σημείο 2 μέχρι το σημείο 3. Τέλος, ο χρόνος αναμονής στην ουρά αποστολής μετριέται ως η χρονική απόσταση από το σημείο 3 μέχρι το σημείο 5.

Για μέτρηση κάθε χρόνου, χρησιμοποιούσαμε την προσωρινή μεταβλητή. Για παράδειγμα στο σημείο 1, αποθηκεύαμε στην προσωρινή μεταβλητή την παρούσα ώρα του συστήματος. Ακολούθως, στη σημείο 2 παίρναμε από το σύστημα την ώρα και αφαιρούσαμε από αυτή την ώρα που είχαμε αποθηκεύσει στην προσωρινή μεταβλητή. Ακολουθούσαμε την ίδια μέθοδο για να πάρουμε τους χρόνους όλων των μεταβλητών.

Οι μετρικές που θέλουμε να καταγράψουμε είναι process duration και service duration. Δημιουργήσαμε μια νέα κλάση, τη Statistics. Η κλάση αυτή περιέχει πέντε μεταβλητές. Περιέχει τον αριθμό των μηνυμάτων που έτυχαν επεξεργασίας από τον εξυπηρετητή, το ολικό process duration, το ολικό service duration καθώς και τις χρονικές στιγμές εκκίνησης και ολοκλήρωσης του πειράματος μας. Οι δύο τελευταίες μετρήσεις ανανεώνονταν στην αρχή και στο τέλος του πειράματος. Οι υπόλοιπες ανανεώνονταν όταν κάποιο μήνυμα στελνόταν στο δίκτυο από το socket αποστολής. Τη χρονική στιγμή που έφευγε το μήνυμα, προσθέταμε στο ολικό process duration τις τιμές των μεταβλητών _dWaitQueueTime και _dProcessTime και στο ολικό service duration τις τιμές των μεταβλητών _dWaitQueueTime, _dProcessTime και _dForwardQueueTime. Επίσης, προσθέταμε ένα στην τιμή των μηνυμάτων που επεξεργαστήκαμε. Όταν η τιμή των μηνυμάτων έφτανε τα χίλια, γράφαμε τις τιμές των μεταβλητών της κλάσης statistics σε αρχείο και διακόπταμε τη λειτουργία του εξυπηρετητή.

Έχοντας τις πιο πάνω τιμές σε αρχείο, ο υπολογισμός των μετρικών μας ήταν εύκολος. Το process duration μετριόταν με τον τύπο $\Pr ocess\_duraiton = \frac{Total\_process\_duration}{Number\_of\_messages}$, ενώ το service duration μετριόταν με τον τύπο $Service\_duration = \frac{Total\_service\_duration}{Number\_of\_messages}$. Τέλος, το throughput μετριόταν με τον τύπο $Throughput = \frac{1000}{\Pr ocess\_duration}$.

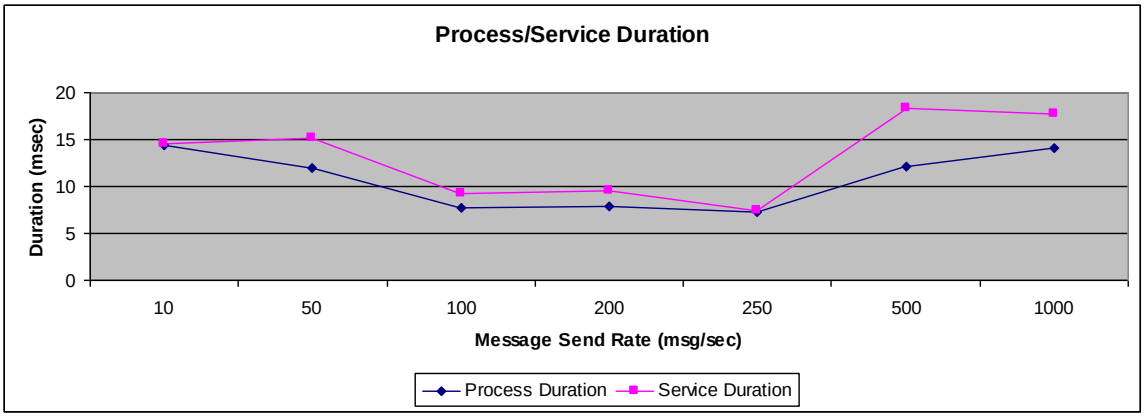
5.6 Διεξαγωγή πειραμάτων (Δεύτερη μέθοδος)

Τα πειράματα διεξήχθησαν με τον ίδιο τρόπο όπως και στην πρώτη μέθοδο. Αυτή τη φορά αποστέλλαμε μόνο GET μηνύματα. Οι ρυθμοί που επιλέξαμε ήταν 10, 50, 100, 200, 250, 500 και 1000 μηνύματα ανά δευτερόλεπτο. Αυτή τη φορά η διεξαγωγή του πειράματος έγινε μεταξύ φορητών υπολογιστών συνδεδεμένων με ασύρματο δίκτυο.

5.7 Παρουσίαση αποτελεσμάτων (Δεύτερη μέθοδος)

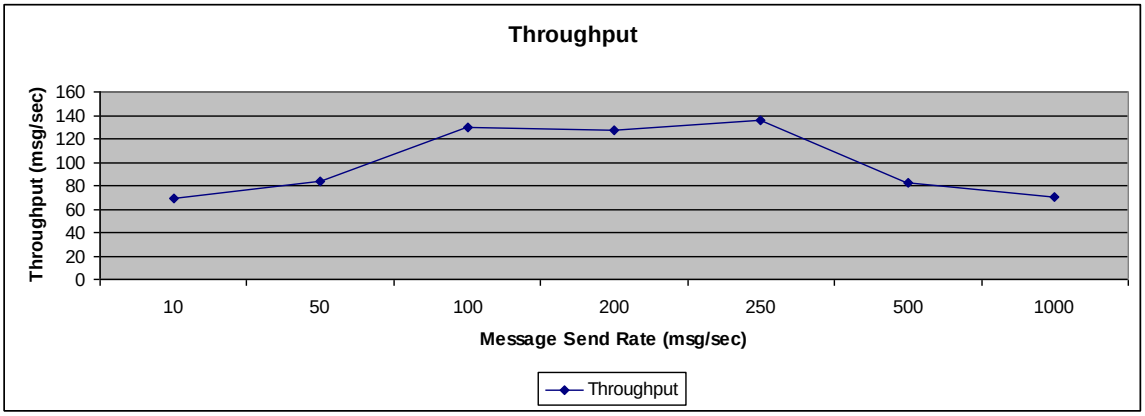
Αφού διεξάγομε τα πειράματα μας και καταγράψαμε τα αποτελέσματα σε αρχεία, περάσαμε τα αποτελέσματα στην Excel και δημιουργήσαμε τις γραφικές μας.

Στο σχήμα 5.6, έχουμε σε κοινό χώρο τις γραφικές του process duration και service duration:



Σχήμα 5.6 Γραφική παράσταση χρόνων επεξεργασίας και εξυπηρέτησης

Παρατηρούμε σταθερές τιμές μεταξύ ρυθμών αποστολής 100 μέχρι 250 μηνύματα ανά δευτερόλεπτο. Για μικρότερους και μεγαλύτερους ρυθμούς, έχουμε μεγαλύτερους χρόνους. Τα αποτελέσματα είναι ικανοποιητικά και αναμενόμενα. Πριν φθάσουμε σε συμπεράσματα, παραθέτουμε τη γραφική του throughput στο Σχήμα 5.7:



Σχήμα 5.7 Γραφική παράσταση του throughput

Παρατηρούμε σταθερά ψηλότερες τιμές μεταξύ ρυθμών αποστολής 100 μέχρι 250 μηνύματα ανά δευτερόλεπτο. Για μικρότερους και μεγαλύτερους ρυθμούς, η απόδοση του εξυπηρετητή είναι σχετικά μειωμένη. Όπως είπαμε και πριν, τα αποτελέσματα αυτά είναι φυσιολογικά. Για μικρούς ρυθμούς παραλαβής μηνυμάτων, ο εξυπηρετητής δεν πιέζεται για να πιάσει κάποια καλή απόδοση. Για μεγάλους ρυθμούς παραλαβής (>250 μηνύματα ανά δευτερόλεπτο) δημιουργείται συνωστισμός στις ουρές, με αποτέλεσμα

κάποια μηνύματα να περιμένουν περισσότερο χρόνο για να τύχουν επεξεργασίας και για να αποσταλούν, κάτι που επηρεάζει τους μέσους όρους των μετρήσεων μας. Σε ρυθμούς παραλαβής μεταξύ 100 μέχρι 250 μηνυμάτων ανά δευτερόλεπτο, ο εξυπηρετητής δουλεύει αποδοτικά.

Ακολούθως, παραθέτουμε τους μέσους όρους κάθε μετρικής:

Μετρική	**Μέσος όρος**
Process Duration	10,77678571 msec/msg
Service Duration	13,11830357 msec/msg
Throughput	99,9993412 msg/sec

Πίνακας 5.2 Μέσοι όροι μετρήσεων

Οι μετρήσεις που θα χρησιμοποιήσουμε για να εξάγουμε συμπεράσματα είναι οι μετρήσεις που πήραμε με τη δεύτερη μέθοδο, αφού η μέθοδος αυτή κρίνεται ως η ορθότερη για διεξαγωγή πειραμάτων στον εξυπηρετητή μας.

Κεφάλαιο 6

Συμπεράσματα

Στο παρών κεφάλαιο εξάγουμε κάποια συμπεράσματα, τόσο για την υλοποίηση του εξυπηρετητή, όσο και για το πρωτόκολλο VITP.

6.1 Γενικά συμπεράσματα

Μετά το πέρας της εκπόνησης της διπλωματικής εργασίας, κατόπιν μελέτης και ανάλυσης του προβλήματος και των παραγόντων που έπρεπε να ληφθούν υπόψιν, αφού διεξήχθησαν διάφορα ελεγχόμενα πειράματα, έφθασε η στιγμή να εξάγουμε τα συμπεράσματα μας.

Αρχικά, είναι αναγκαίο να συμπεράνουμε κατά πόσον η υλοποίηση του εξυπηρετητή κρίνεται ικανοποιητική. Δοκιμάσαμε τον εξυπηρετητή υπό διάφορες συνθήκες συμφόρησης μηνυμάτων και καταλήξαμε σε κάποιους μέσους όρους. Συγκεκριμένα, κατά μέσο όρο, ο μέσος χρόνος επεξεργασίας ενός μηνύματος είναι περίπου 10 mseconds και ο χρόνος εξυπηρέτησης των μηνυμάτων που απαιτούν απάντηση είναι περίπου 13 mseconds.

Όσον αφορά το χρόνο επεξεργασίας, αυτός κρίνεται ικανοποιητικός. Στην περίπτωση που το μήνυμα που επιθυμούμε να τύχει επεξεργασίας είναι κάποιο alert, αυτό θα τύχει έγκαιρης επεξεργασίας και ο χρήστης θα ειδοποιηθεί σε χρόνο που, σε περίπτωση

κινδύνου, θα έχει την ευκαιρία να αντιδράσει και να αποφύγει τον κίνδυνο. Σε σχέση με τις ταχύτητες με τις οποίες κινείται κάποιο όχημα, 10 mseconds είναι ένας πολύ μικρός χρόνος, συνεπώς το alert θα παρουσιαστεί στο χρήστη πολύ πριν το όχημα είναι σε θέση που δεν μπορεί να ελαττώσει ταχύτητα.

Όσον αφορά το χρόνο εξυπηρέτησης, αυτός κρίνεται και πάλι ικανοποιητικός. Για τη συγκεκριμένη μετρική, βασική απαίτηση είναι ο χρήστης να λάβει απάντηση σε χρόνο που να μπορεί να αποφασίσει για παράδειγμα τη διαδρομή του ή πριν το όχημα απομακρυνθεί από την περιοχή στην οποία ήταν όταν έστειλε το μήνυμα. Και πάλι σε σχέση με τις ταχύτητες με τις οποίες κινείται κάποιο όχημα αλλά και σε σχέση με το χρόνο που ο άνθρωπος χρειάζεται για να λάβει κάποια απόφαση, 13 mseconds είναι πολύ μικρό χρονικό διάστημα. Αν υποθέσουμε ότι κάποιο μήνυμα χρειάζεται να περάσει από 10 κόμβους για να ικανοποιηθούν τα return conditions, πάλι 130 mseconds εξακολουθούν να αποτελούν ένα πολύ μικρό χρονικό διάστημα σε σχέση με τις ταχύτητες του μη υπολογιστικού κόσμου.

Το μέσο throughput του εξυπηρετητή είναι συνάρτηση του process duration. Κατά μέσο όρο, το throughput του εξυπηρετητή είναι περίπου 100 msg/sec. Προς το παρών δεν μπορούμε να κρίνουμε τη συγκεκριμένη τιμή, αν δεν γνωρίζουμε σε τι είδος δικτύου θα εφαρμοστεί ο εξυπηρετητής. Γνωρίζοντας τον τύπο δικτύου και το throughput του, θα μπορέσουμε να εξετάσουμε αν η τιμή του throughput του εξυπηρετητή (αφού τη μετατρέψουμε σε bits/sec) είναι κοντά στην τιμή του throughput του δικτύου, ότι δηλαδή ο εξυπηρετητής μας μπορεί να εξυπηρετήσει μηνύματα τα οποία στέλνονται με ρυθμό που αγγίζει τις μέγιστες δυνατότητες του δικτύου. Να αναφέρουμε ότι μεταξύ ρυθμών αποστολής 100 μέχρι 250 μηνυμάτων ανά δευτερόλεπτο, ο εξυπηρετητής δουλεύει αποδοτικότερα, συγκεκριμένα με throughput περίπου 130 msg/sec.

Τέλος, όσον αφορά τις εγγραφές στο δίσκο, μέσος όρος 369 kbytes είναι μια μηδαμινή τιμή σε σχέση με τα σημερινά μεγέθη σκληρών δίσκων που κυκλοφορούν. Μπορούμε να εξετάσουμε επίσης τη μέγιστη τιμή, η οποία είναι 720 kbytes για 1000 μηνύματα. Αν

μέχρι να ολοκληρωθεί η λειτουργία του, ο εξυπηρετητής λάβει μέχρι 100000 μηνύματα, θα χρειάζεται γύρω στα 72 Mbytes για αποθήκευση των log αρχείων.

Όσον αφορά το πρωτόκολλο VITP, ο συγγραφέας κρίνει το πρωτόκολλο πολύ καλό στη σύλληψη, αλλά ίσως δύσκολο στην εφαρμογή. Βεβαίως απαιτείται περαιτέρω μελέτη και εργασία για να κρίνουμε αν το πρωτόκολλο μπορεί να έχει καλή απόδοση. Το βέβαιο είναι πως το VITP είναι πάρα πολύ απλό στην κατανόηση, αφού σε λίγο χρονικό διάστημα έγινε κατανοητό από το συγγραφέα. Επίσης, τα προβλήματα που προσπαθεί να επιλύσει ή κατ' ακρίβεια οι υπηρεσίες που επιχειρεί να προσφέρει είναι πάρα πολύ χρήσιμες, ειδικά στις μεγαλουπόλεις, όπου η κυκλοφοριακή κίνηση αποτελεί καθημερινό πονοκέφαλο των οδηγών. Αν τελικά το πρωτόκολλο εφαρμοστεί, οι οδηγοί θα έχουν τη δυνατότητα να αποφύγουν σημεία με κυκλοφοριακή συμφόρηση, να εντοπίσουν εύκολα οδηγικές υπηρεσίες και να αποφύγουν αρκετούς κινδύνους στο δρόμο.

Η μέχρι τώρα εργασία δείχνει ότι εφαρμογή του πρωτοκόλλου είναι φθηνή. Αν καταστεί δυνατό, κατόπιν περισσότερης μελέτης, τα κόστα να διατηρηθούν χαμηλά, οι αυτοκινητοβιομηχανίες δε θα έχουν επιλογή παρά να το εφαρμόσουν στα οχήματα που κατασκευάζουν. Αυτό το σενάριο βεβαίως είναι πολύ μακροπρόθεσμο.

6.2 Μελλοντική εργασία

Η εργασία που έγινε στην παρούσα διπλωματική αποτελεί ένα πολύ μικρό μέρος του τι απαιτείται για να εφαρμοστεί το πρωτόκολλο υπό κανονικές συνθήκες. Πολλές διαφορετικές εργασίες πρέπει να γίνουν για να μπορεί το VITP να μπει σε εφαρμογή.

Αρχικά, πρέπει να κατασκευαστούν συσκευές για καταγραφή των δεδομένων που στέλνουν οι αισθητήρες του οχήματος. Αυτό αποτελεί τόσο ευθύνη του κλάδου της πληροφορικής όσο και της μηχανολογίας. Ακολούθως, είναι αναγκαίο οι συσκευές να ενωθούν με τον υπολογιστή ο οποίος τρέχει τον εξυπηρετητή και να γίνεται συνεχής ανανέωση του XML αρχείου.

Μια από τις συσκευές που αναφέρθηκαν είναι συσκευή η οποία μετατρέπει τις συντεταγμένες GPS σε συντεταγμένες VITP. Αυτό δεν είναι τόσο απλό όσο φαίνεται. Αρχικά, πρέπει να εισαχθούν στο όχημα χάρτες της περιοχής στην οποία κινείται. Πρέπει στη συνέχεια να προσδιοριστούν οι περιοχές στις οποίες χωρίζεται ο χάρτης, κάτι αρκετά χρονοβόρο αφού ακόμα και η Κύπρος, ένα πολύ μικρό νησί, έχει έκταση περίπου 9000 τετραγωνικά χιλιόμετρα. Αυτή η έκταση πρέπει να διαχωριστεί και κάθε περιοχή να λάβει ένα road_id και να υποδιαιρεθεί σε segment_id. Οι χάρτες πρέπει να ανανεώνονται συνεχώς για αποφυγή λαθών και όλα τα οχήματα πρέπει να έχουν έγκυρα δεδομένα των ονοματολογιών των περιοχών, αλλιώς θα παρουσιαστούν φαινόμενα κρίσιμων σφαλμάτων.

Μελλοντική εργασία είναι επίσης σημαντικό να υπάρξει στο VITP πρωτόκολλο. Η αδυναμία που εντοπίστηκε είναι στους λιγοστούς τύπους μηνυμάτων που υπάρχουν. Συγκεκριμένα, υποστηρίζονται μόνο μηνύματα σχετικά με τροχαία κίνηση, πρατήρια καυσίμων και συναγερμοί για ολισθηρό οδόστρωμα και τροχαίο ατύχημα. Υπάρχει μια πλειάδα τύπων οι οποίοι μπορούν να υποστηριχθούν, όπως φώτα τροχαίας, πλυντήρια αυτοκινήτων, πρατήρια επισκευής αυτοκινήτων, οδικά έργα, χώροι στάθμευσης, κλείσιμο δρόμου λόγω χιονιών ή κατολίσθησης και πολλά άλλα. Είναι επίσης σημαντικό να προσδιοριστούν τα περιεχόμενα του message body κομματιού του VITP μηνύματος, για αποφυγή σφαλμάτων.

Τέλος, μελλοντική εργασία πρέπει να γίνει και στον εξυπηρετητή που υλοποιήθηκε στην παρούσα διπλωματική. Η υλοποίηση του έγινε σε αντικειμενοστρεφή γλώσσα και υπάρχει δυνατότητα εύκολης επαναχρησιμοποίησης κώδικα. Συνεπώς, εύκολα μπορεί η εργασία να ενωθεί με μελλοντικές εργασίες για δημιουργία μιας πλήρους εφαρμογής η οποία θα λειτουργεί πάνω σε κάποιο όχημα. Αν σε μελλοντική έκδοση του πρωτοκόλλου προστεθούν περισσότεροι τύποι μηνυμάτων, είναι αναγκαίο να υπάρξει πρόσβαση στον πηγαίο κώδικα του εξυπηρετητή. Αυτό κρίνεται από το συγγραφέα σαν μια αδυναμία της υλοποίησης. Παρόλα αυτά, επιχειρήθηκε ο κώδικας να είναι όσο πιο ξεκάθαρος και δομημένος γίνεται, για να βοηθηθούν μελλοντικοί προγραμματιστές στην κατανόηση και

τροποποίηση του. Τέλος, ίσως κριθεί αναγκαίο στο μέλλον να γίνουν τροποποιήσεις στην XML γραμματική που κατασκευάστηκε για χάριν της εργασίας. Συγκεκριμένα, μπορούν να προστεθούν περισσότερες πληροφορίες, ανάλογα με το τι είδους αισθητήρες θα επικοινωνούν με τον εξυπηρετητή.

6.3 Επίλογος

Ένα από τις συχνότερες πηγές άγχους, εκνευρισμού, αλλά και κινδύνων του σύγχρονου ανθρώπου είναι η οδήγηση. Συχνά οι ανθρώποι των πόλεων παγιδεύονται σε τροχαία συμφόρηση ή ψάχνουν χωρίς αποτέλεσμα χώρο στάθμευσης. Επίσης, πολλά ατυχήματα θα μπορούσαν να αποφευχθούν, αν οι οδηγοί γνώριζαν τι καραδοκεί στο δρόμο. Το πρωτόκολλο VITP μπορεί να λύσει πολλά από αυτά τα προβλήματα και να αφαιρέσει ένα κομμάτι του άγχους που διακατέχει το σύγχρονο οδηγό.

Όσον αφορά την εμπειρία της εκπόνησης της παρούσας διπλωματικής, αυτή πρόσφερε πολλά εφόδια στο συγγραφέα. Η μελέτη που προηγήθηκε δίδαξε το συγγραφέα να την επιστημονική έρευνα καθώς και τις πηγές επιστημονικών πληροφοριών. Η μελέτη ενός πρωτοκόλλου και κατασκευή εξυπηρετητή για αυτό, με ελάχιστη προηγούμενη εργασία πάνω στο συγκεκριμένο θέμα ήταν μια δύσκολη αλλά παράλληλα διδακτική εμπειρία.

Βιβλιογραφία

[1] David J. Lilja, “Measuring computer performance: A practitioner’s guide”, Publisher: Cambridge University Press, 2000

[2] Douglas E. Comer, David L. Stevens, “Internetworking with TCP/IP, Vol. III Client-Server Programming and Applications”, Publisher: Prentice Hall/Pearson, 2003

[3] http://en.wikipedia.org/wiki/VANET

[4] http://en.wikipedia.org/wiki/Mobile_ad-hoc_network

[5] http://www.w3schools.com

[6] http://msdn.microsoft.com

[7] James F. Kurose, Keith W. Ross, “Computer Networking: A top-down approach featuring the Internet”, Third edition, Publisher: Pearson Education, 2005

[8] James Clark, Steve DeRose, “XML Path Language (XPath): Version 1.0”, W3C, 1999

[9] Jinyang Li, Charles Blake, Douglas S. J. De Couto, Hu Imm Lee, Robert Morris, “Capacity of Ad hoc wireless networks”, M.I.T. Laboratory for Computer Science, 2001

[10] Magnus Frodigh, Per Johansson and Peter Larsson, “Wireless ad hoc networking—The art of networking without a network”, Ericsson Review No. 4, 2000

[11] Marios D. Dikaiakos, Tamer Nadeem, Saif Iqbal and Liviu Iftode, “VITP: An Information Transfer Protocol for Vehicular Computing”, ACM, New York, NY, USA, 2005

[12] Stephan Eichler, Christoph Schroth and Jörg Eberspächer, “Car-to-car communication”, Proceedings of the VDE Kongress–Innovations for Europe, Oct, 2006

Παράρτημα Α

Πίνακες μηνυμάτων VITP

Πίνακας ενεργειών του Query Executor για τα ανάλογα URI του μηνύματος:

URI μηνύματος	Ενέργεια Query Executor
GET /service/gas	XPath queries: vitp/service/available vitp/service/price vitp/service/road_id vitp/service/segment_id
POST /vehicle/alert	Προσθήκη μηνύματος στην cache
GET /vehicle/alert	Αναζήτηση μηνυμάτων τύπου alert στην cache
GET /vehicle/traffic	XPath query: vitp/vehicle/average_speed

Πίνακας τύπων αποθήκευσης δεδομένων μηνύματος VITP:

Δεδομένο	Τύπος αποθήκευσης
Method	String
URI	String
Version	Double
Sender road_id	String
Sender segment_id	String
Sender speed	Double
Destination road_id	String
Destination segment_id	String
Time sent	DateTime
Expiration time	DateTime
Cache-Control	String
TTL	Int32
Message ID	String
Content Length	Int32
Message Body	String

Κώδικας εξυπηρετητή

VITPThread.h

```
#include "stdafx.h"
#include "QueryExecutor.h"
#include "VITPSendThread.h"

using namespace System;
using namespace System::Threading;

namespace VITP{

	public ref class VITPThread
	{
	public:
		enum class THREADSTATE{
			DEAD = -1,
			ALIVE = 0,
			REQTOEND = 1
		};

		///<summary>
		///The thread object.
		///</summary>
		System::Threading::Thread^ _oThread;
		///<summary>
		///The mutex object.
		///</summary>
		System::Threading::Mutex^ _oMutex;
		///<summary>
		///Stores the received messages.
		///</summary>
		System::Collections::Queue^ _oQueue;
		///<summary>
		///Executes VITP queries.
		///</summary>
		VITP::QueryExecutor^ _oQExecutor;
		///<summary>
		///Diagnostics XML file.
		///</summary>
		System::Xml::XmlDocument^ _fDiagnosticsFile;
		///<summary>
		///Thread to send messages.
		///</summary>
		VITP::VITPSendThread^ _oSendThread;

		///<summary>
		///Current thread state.
		///</summary>
		property THREADSTATE ThreadState{
			THREADSTATE get(){
				return _enState;
			}
		}

		///<summary>
		///Creates a new instance of VITPThread.
		///</summary>
		VITPThread(VITP::QueryExecutor^ qExecutor, String^ dFile);
		!VITPThread(){
			stopProcess();
		}
```

```
            ~VITPThread(){
                  stopProcess();
            }
            ///<summary>
            ///Asynchronously pushes a received message
            ///in the queue for processing.
            ///</summary>
            void Push(VITP::VITPMsg^ message);
            ///<summary>
            ///Asynchronously pops a received message
            ///from the queue for processing.
            ///</summary>
            VITP::VITPMsg^ Pop();
            ///<summary>
            ///Launches the thread process.
            ///</summary>
            void startProcess();

      private:
            THREADSTATE _enState;
            ///<summary>
            ///The main thread process. It pops messages
            ///from the queue and processes them.
            ///</summary>
            void threadProcess();
            ///<summary>
            ///Add a message to the send thread queue.
            ///</summary>
            void sendMessage(String^ message);
            ///<summary>
            ///Creates VITPMsg response to the given message.
            ///</summary>
            String^ createResponse(String^ method, String^ rc_expr, String^
data, VITP::VITPMsg^ receivedMessage);
            ///<summary>
            ///Stops the thread process.
            ///</summary>
            void stopProcess();

      };
}
```

VITPThread.cpp

```
#include "stdafx.h"
#include "VITPThread.h"

//
//-------------------------------------------------------------------------
-------------
VITP::VITPThread::VITPThread(VITP::QueryExecutor^ qExecutor, String^
dFile){
      _oQExecutor = qExecutor;
      _enState = THREADSTATE::DEAD;
      _fDiagnosticsFile = gcnew System::Xml::XmlDocument();
      _fDiagnosticsFile->Load(dFile);
      _oMutex = gcnew System::Threading::Mutex(false);
      _oQueue = gcnew System::Collections::Queue();
      _oSendThread = gcnew VITP::VITPSendThread();
}
//
//-------------------------------------------------------------------------
-------------
```

```
void VITP::VITPThread::Push(VITP::VITPMsg ^message){
	try{
		if(_oMutex->WaitOne(100, false)){
			_oQueue->Enqueue(message);
		}
	}catch(Exception^ ex){
		System::Console::WriteLine(ex->ToString());
	}finally{
		_oMutex->ReleaseMutex();
	}
}
//
//-------------------------------------------------------------------------
-------------
VITP::VITPMsg^ VITP::VITPThread::Pop(){
	VITP::VITPMsg^ message;
	try{
		if(_oMutex->WaitOne(100, false)){
			if(_oQueue->Count > 0)
				message = safe_cast<VITP::VITPMsg^>(_oQueue-
>Dequeue());
			else
				message = nullptr;
		}
	}catch(Exception^ ex){
		System::Console::WriteLine(ex->ToString());
	}finally{
		_oMutex->ReleaseMutex();
	}
	return message;
}
//
//-------------------------------------------------------------------------
-------------
void VITP::VITPThread::startProcess(){
	if(nullptr != _oThread)
		return;
	_enState = THREADSTATE::ALIVE;
	_oThread = gcnew Thread(gcnew ThreadStart(this,
&VITP::VITPThread::threadProcess));
	_oThread->Start();
	_oSendThread->startProcess();
}
//
//-------------------------------------------------------------------------
-------------
void VITP::VITPThread::threadProcess(){
	VITP::VITPMsg^ receivedMsg;
	array<String^>^ queryResult;
	String^ responseMsgString;
	System::IO::StreamWriter^ logWriter = gcnew
System::IO::StreamWriter("Log/sent.txt",true);

	while(_enState == THREADSTATE::ALIVE){
		receivedMsg = Pop();
		if(nullptr == receivedMsg){
			System::Threading::Thread::Sleep(10);
			continue;
		}
		queryResult = _oQExecutor->execute(receivedMsg);

		if(nullptr == queryResult){
			//Send message back to the network
			String^ returnMsg =
VITP::VITPMsgParser::ClassToString(receivedMsg);
			if(nullptr != returnMsg)
```

```
	sendMessage(VITP::VITPMsgParser::ClassToString(receivedMsg));
		}
		//Message is invalid (end of timeout or cnt parameter)
		else if(queryResult[1]->CompareTo("msg_invalid") == 0){
			responseMsgString = createResponse(queryResult[0],
queryResult[1], queryResult[2], receivedMsg);
			//Uncomment for debuging
			//System::Console::WriteLine(responseMsgString);
			sendMessage(responseMsgString);
			logWriter->WriteLine(DateTime::Now.ToString("dd/MM/yyyy
hh:mm:ss:ffff") + ".");
			logWriter->Flush();
		}
		//Message is of type "POST"
		else if(receivedMsg->_sMethod->CompareTo("POST") == 0){
			//Do nothing. Message has already been posted in cache.
		}
		else{
			responseMsgString = createResponse(queryResult[0],
queryResult[1], queryResult[2], receivedMsg);
			//Uncomment for debuging
			//System::Console::WriteLine(responseMsgString);
			sendMessage(responseMsgString);
			logWriter->WriteLine(DateTime::Now.ToString("dd/MM/yyyy
hh:mm:ss:ffff") + ".");
			logWriter->Flush();
		}
	}
	_enState = THREADSTATE::DEAD;
}
//
//-------------------------------------------------------------------------
-------------
String^ VITP::VITPThread::createResponse(String^ method, String^ rc_expr,
String^ data, VITP::VITPMsg^ receivedMessage){
	String^ respMsgString;
	String^ roadNode;
	String^ segmentNode;
	String^ speed;

	array<String^>^ tokens = gcnew array<String^>(2);
	tokens[0] = "[";
	tokens[1] = "]";
	array<String^>^ parts = receivedMessage->_sUri->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);
	parts[1] = rc_expr;

	//Find road_id and segment_id if server is on a vehicle
	roadNode = _oQExecutor->xpathExecute("vehicle/road_id");
	segmentNode = _oQExecutor->xpathExecute("vehicle/segment_id");
	speed = _oQExecutor->xpathExecute("vehicle/current_speed");

	//Find road_id and segment_id if server is on a service
	if(nullptr == roadNode || roadNode->CompareTo("msg_invalid") == 0){
		roadNode = _oQExecutor->xpathExecute("service/road_id");
		segmentNode = _oQExecutor->xpathExecute("service/segment_id");
		speed = _oQExecutor->xpathExecute("service/current_speed");
	}

	//Compose response message as string
	respMsgString =
		method + " " + parts[0] + "["+ parts[1]+ "]" + parts[2] + "
VITP/" + receivedMessage->_dVersion.ToString()
		+ "\nTarget: [" + receivedMessage->_sFromRoad_id + " , " +
receivedMessage->_sFromSegment_id + "]"
```

```
            + "\nFrom: [" + roadNode + " , " + segmentNode + "]" + " with "
+ speed
            + "\nTime: " + System::DateTime::Now.ToString("dd/MM/yyyy
hh:mm:ss")
            + "\nExpires: " +
System::DateTime::Now.AddMilliseconds(100).ToString("dd/MM/yyyy hh:mm:ss")
//Expires after 100msec
            + "\nCache-Control: yes" //Since the message is POST it should
be stored in cache
            + "\nTTL: 10000"
            + "\nmsgID: " + System::Guid::NewGuid().ToString() //Generate
unique random id
            + "\nContent-Length: " + (data->Length*2).ToString() //Unicode
chars are 16-bit
            + String::Concat(Char::ToString(Char(13)),
Char::ToString(Char(10))) //CRLF
            + data;

      return respMsgString;
}
//
//-------------------------------------------------------------------------
-------------
void VITP::VITPThread::sendMessage(String^ message){
      _oSendThread->Push(message);
}
//
//-------------------------------------------------------------------------
-------------
void VITP::VITPThread::stopProcess(){
      _enState = THREADSTATE::REQTOEND;
      while(THREADSTATE::DEAD != _enState){
            System::Threading::Thread::Sleep(10);
      }
      delete _oThread;
      _oThread = nullptr;
}
```

VITPSocket.h

```
#include "stdafx.h"

using namespace System;
using namespace System::Net::Sockets;

namespace VITP{

      public ref class VITPSocket
      {
      public:
            VITPSocket(Int32 port);
            ///<summary>
            ///Receive a string and return it.
            ///</summary>
            String^ receive();
            ///<summary>
            ///Send the specified string to the specified
            ///IP address to the specified port.
            ///</summary>
            void send(String^ message, String^ ip, Int32 port);

      private:
            ///<summary>
```

```
            ///The socket object.
            ///</summary>
            Socket^ _oSocket;
      };

}
```

VITPSocket.cpp

```
#include "stdafx.h"
#include "VITPSocket.h"

//
//-------------------------------------------------------------------------
-------------
VITP::VITPSocket::VITPSocket(int port){
      String^ hostName = System::Net::Dns::GetHostName();
      System::Net::IPHostEntry^ myHost =
System::Net::Dns::GetHostEntry(hostName);
      System::Net::IPAddress^ address =
safe_cast<System::Net::IPAddress^>(myHost->AddressList->GetValue(0));
      System::Net::IPEndPoint^ localPoint = gcnew
System::Net::IPEndPoint(address->Address, port);

      //Create UDP socket and initialize it
      _oSocket = gcnew Socket(AddressFamily::InterNetwork,
SocketType::Dgram, ProtocolType::Udp);
      _oSocket->Bind(localPoint);
}
//
//-------------------------------------------------------------------------
-------------
String^ VITP::VITPSocket::receive(){
      array<unsigned char,1>^ buffer = gcnew array<unsigned
char,1>(100000);
      System::Text::ASCIIEncoding^ encoder = gcnew
System::Text::ASCIIEncoding();

      try{
            //Receive UDP message and return it as string
            _oSocket->Receive(buffer);
            return encoder->GetString(buffer, 0, buffer->Length);
      }catch(Exception^ ex){
            System::Console::WriteLine(ex->ToString());
            return nullptr;
      }
}
//
//-------------------------------------------------------------------------
-------------
void VITP::VITPSocket::send(String^ message, String^ ip, Int32 port){
      System::Net::IPAddress^ address;
      System::Net::IPEndPoint^ endPoint;
      array<wchar_t,1>^ intermidiateMessageBuffer = message->ToCharArray();
      array<unsigned char,1>^ MessageBuffer = gcnew array<unsigned
char,1>(intermidiateMessageBuffer->Length);

      try{
            //Create unsigned char array for message
            for(int i = 0; i < intermidiateMessageBuffer->Length; i++){
                  MessageBuffer[i] = intermidiateMessageBuffer[i];
            }
```

```
            //Parse the IP Address from string
            address = System::Net::IPAddress::Parse(ip);
            endPoint = gcnew System::Net::IPEndPoint(address, port);

            //Send message
            _oSocket->SendTo(MessageBuffer, endPoint);
      }
      catch(Exception^ ex){
            Console::WriteLine(ex->ToString());
      }
}
```

VITPSendThread.h

```
#include "stdafx.h"
#include "VITPMsgParser.h"
#include "VITPSocket.h"

using namespace System;
using namespace System::Threading;

namespace VITP{

      public ref class VITPSendThread
      {
      public:
            enum class THREADSTATE{
                  DEAD = -1,
                  ALIVE = 0,
                  REQTOEND = 1
            };

            ///<summary>
            ///The thread object.
            ///</summary>
            System::Threading::Thread^ _oThread;
            ///<summary>
            ///The mutex object.
            ///</summary>
            System::Threading::Mutex^ _oMutex;
            ///<summary>
            ///Stores the to-send messages.
            ///</summary>
            System::Collections::Queue^ _oSendQueue;
            ///<summary>
            ///Socket object.
            ///</summary>
            VITP::VITPSocket^ _oSocket;
            ///<summary>
            ///StreamReader to read IP log file.
            ///</summary>
            System::IO::StreamReader^ _oReadFile;

            property THREADSTATE ThreadState{
                  THREADSTATE get(){
                        return _enState;
                  }
            }

            ///<summary>
            ///Creates a new instance of VITPSendThread.
            ///</summary>
            VITPSendThread();
```

```
            !VITPSendThread(){
                  stopProcess();
            }
            ~VITPSendThread(){
                  stopProcess();
            }
            ///<summary>
            ///Asynchronously pushes a message
            ///in the queue for sending.
            ///</summary>
            void Push(String^ message);
            ///<summary>
            ///Asynchronously pops a message
            ///from the queue for sending.
            ///</summary>
            String^ Pop();
            ///<summary>
            ///Launches the thread process.
            ///</summary>
            void startProcess();

      private:
            THREADSTATE _enState;
            ///<summary>
            ///The main thread process. It pops messages
            ///from the queue and sends them.
            ///</summary>
            void threadProcess();
            ///<summary>
            ///Sends message according to the road_id and segment_id.
            ///</summary>
            void sendMessage(String^ message, String^ road_id, String^
segment_id);
            ///<summary>
            ///Stops the thread process.
            ///</summary>
            void stopProcess();
      };
}
```

VITPSendThread.cpp

```
#include "stdafx.h"
#include "VITPSendThread.h"

//
//-------------------------------------------------------------------------
-------------
VITP::VITPSendThread::VITPSendThread(){
      _enState = THREADSTATE::DEAD;
      _oMutex = gcnew System::Threading::Mutex(false);
      _oSendQueue = gcnew System::Collections::Queue();
      _oSocket = gcnew VITP::VITPSocket(60000);
}
//
//-------------------------------------------------------------------------
-------------
void VITP::VITPSendThread::Push(String^ message){
      try{
            if(_oMutex->WaitOne(100, false)){
                  _oSendQueue->Enqueue(message);
            }
      }catch(Exception^ ex){
```

```
            System::Console::WriteLine(ex->ToString());
      }finally{
            _oMutex->ReleaseMutex();
      }
}
//
//--------------------------------------------------------------------------
-------------
String^ VITP::VITPSendThread::Pop(){
      String^ message;
      try{
            if(_oMutex->WaitOne(100, false)){
                  if(_oSendQueue->Count > 0)
                        //Pop message from queue
                        message = safe_cast<String^>(_oSendQueue-
>Dequeue());
                  else
                        message = nullptr;
            }
      }catch(Exception^ ex){
            System::Console::WriteLine(ex->ToString());
      }finally{
            _oMutex->ReleaseMutex();
      }
      return message;
}
//
//--------------------------------------------------------------------------
-------------
void VITP::VITPSendThread::startProcess(){
      if(nullptr != _oThread)
            return;
      _enState = THREADSTATE::ALIVE;
      _oThread = gcnew Thread(gcnew ThreadStart(this,
&VITP::VITPSendThread::threadProcess));
      _oThread->Start();
}
//
//--------------------------------------------------------------------------
-------------
void VITP::VITPSendThread::threadProcess(){
      String^ sendMsg;
      VITP::VITPMsg^ sendMsgObject;
      String^ road_id;
      String^ segment_id;

      while(_enState == THREADSTATE::ALIVE){
            //Start popping messages
            sendMsg = Pop();
            if(nullptr == sendMsg){
                  System::Threading::Thread::Sleep(10);
                  continue;
            }
            sendMsgObject = VITP::VITPMsgParser::StringToClass(sendMsg);
            if(nullptr != sendMsgObject){
                  //Find road_id and segment_id and send message
                  road_id = sendMsgObject->_sTargetRoad_id;
                  segment_id = sendMsgObject->_sTargetSegment_id;
                  sendMessage(sendMsg, road_id, segment_id);
            }
      }
      _enState = THREADSTATE::DEAD;
}
//
//--------------------------------------------------------------------------
-------------
```

```
void VITP::VITPSendThread::sendMessage(String^ message, String^ road_id,
String^ segment_id){
	String^ line;
	String^ ipAddress;
	array<String^>^ lineParts;
	array<String^>^ tokens = gcnew array<String^>(4);
	tokens[0] = "road_id=";
	tokens[1] = ",";
	tokens[2] = "segment_id=";
	tokens[3] = "IP=";
	_oReadFile = gcnew System::IO::StreamReader("Log/log_ip.txt");

	try{
		while(_oReadFile->Peek() >= 0 ){
			//Find IP Address from file and send the message
			line = _oReadFile->ReadLine();
			lineParts = line->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);
			if(road_id->CompareTo(lineParts[0]) == 0 && segment_id-
>CompareTo(lineParts[1]) == 0){
				ipAddress = lineParts[2];
				_oSocket->send(message, ipAddress, 65000);
				VITP::VITPMsg^ msg =
VITP::VITPMsgParser::StringToClass(message);
				Console::WriteLine("\nMessage with ID " + msg-
>_smsgID + " sent to (road_id: " + msg->_sTargetRoad_id + ", segment_id: "
+ msg->_sTargetSegment_id + ").");
				break;
			}
		}
	}
	catch(Exception^ ex){
		System::Console::WriteLine(ex->ToString());
	}

	return;
}
//
//-------------------------------------------------------------------------
-------------
void VITP::VITPSendThread::stopProcess(){
	_enState = THREADSTATE::REQTOEND;
	while(THREADSTATE::DEAD != _enState){
		System::Threading::Thread::Sleep(10);
	}
	delete _oThread;
	_oThread = nullptr;
}
```

VITPMsgParser.h

```
#include "stdafx.h"
using namespace System;

namespace VITP{

	///<summary>
	///Class that provides methods for parsing a message
	///string to a VITPMsg object and reversly.
	///</summary>
	public class VITPMsgParser
	{
	public:
```

```
            ///<summary>Takes a VITP message string and returns VITPMsg
object.</summary>
            ///<param name="message">The message string.</param>
            static VITP::VITPMsg^
VITP::VITPMsgParser::StringToClass(System::String ^message){
                  array<String^>^ msgDivide = nullptr;
                  array<String^>^ msgParts = nullptr;
                  array<String^>^ tokens = gcnew array<String^>(1);
                  VITP::VITPMsg^ newMessage = nullptr;
                  System::Threading::Mutex^ _oMutex = gcnew
System::Threading::Mutex(false);

                  try{
                        //Split the message string
                        //Create CRLF character
                        tokens[0] =
String::Concat(Char::ToString(Char(13)), Char::ToString(Char(10)));
                        msgDivide = message->Split(tokens,
System::StringSplitOptions::None);
                        tokens[0] = "\n";
                        msgParts = msgDivide[0]->Split(tokens,
System::StringSplitOptions::None);

                        //Assign each part to the corresponding variable
                        //Line1
                        tokens = gcnew array<String^>(2);
                        tokens[0] = " ";
                        tokens[1] = "VITP/";
                        array<String^>^ Line1 = msgParts[0]->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);
                        String^ Method = Line1[0];
                        String^ Uri = Line1[1];
                        Double Version = Double::Parse(Line1[2]);

                        //Line2
                        tokens = gcnew array<String^>(3);
                        tokens[0] = "Target: [";
                        tokens[1] =  " , ";
                        tokens[2] = "]";
                        array<String^>^ Line2 = msgParts[1]->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);
                        String^ TargetRoad_id = Line2[0];
                        String^ TargetSegment_id = Line2[1];

                        //Line3
                        tokens = gcnew array<String^>(4);
                        tokens[0] = "From: [";
                        tokens[1] =  " , ";
                        tokens[2] = "]";
                        tokens[3] = " with ";
                        array<String^>^ Line3 = msgParts[2]->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);
                        String^ FromRoad_id = Line3[0];
                        String^ FromSegment_id = Line3[1];
                        Double Speed = Double::Parse(Line3[2]);

                        //Line4
                        tokens = gcnew array<String^>(1);
                        tokens[0] = "Time: ";
                        array<String^>^ Line4 = msgParts[3]->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);
                        DateTime FromTime = DateTime::Parse(Line4[0]);

                        //Line5
                        tokens = gcnew array<String^>(1);
                        tokens[0] = "Expires: ";
```

```
                        array<String^>^ Line5 = msgParts[4]->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);
                        DateTime Expires = DateTime::Parse(Line5[0]);

                        //Line6
                        tokens = gcnew array<String^>(1);
                        tokens[0] = "Cache-Control: ";
                        array<String^>^ Line6 = msgParts[5]->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);
                        String^ CacheControl = Line6[0];

                        //Line7
                        tokens = gcnew array<String^>(1);
                        tokens[0] = "TTL: ";
                        array<String^>^ Line7 = msgParts[6]->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);
                        Int32 TTL = Int32::Parse(Line7[0]);

                        //Line8
                        tokens = gcnew array<String^>(1);
                        tokens[0] = "msgID: ";
                        array<String^>^ Line8 = msgParts[7]->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);
                        String^ msgID = Line8[0];

                        //Line9
                        tokens = gcnew array<String^>(1);
                        tokens[0] = "Content-Length: ";
                        array<String^>^ Line9 = msgParts[8]->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);
                        Int32 ContentLength = Int32::Parse(Line9[0]);

                        //Line11
                        String^ MsgBody = msgDivide[1];

                        newMessage =
                              gcnew VITP::VITPMsg(Method, Version, Uri,
FromRoad_id, FromSegment_id, Speed,
                                                                  TargetRoad_id,
TargetSegment_id, FromTime, Expires, CacheControl, TTL,
                                                      msgID, ContentLength,
MsgBody);
                  }
                  catch(Exception^ ex){
                        System::Console::WriteLine(ex->ToString());
                        return nullptr;
                  }

                  return newMessage;

            }
            //
            ///<summary>Takes a VITPMsg object and returns VITP message
string.</summary>
            ///<param name="message">The message to be parsed.</param>
            static String^ ClassToString(VITP::VITPMsg^ message){
                  String^ returnString = "";

                  try{
                        returnString =
                              message->_sMethod + " " + message->_sUri + "
VITP/" + message->_dVersion.ToString()
                              + "\nTarget: [" + message->_sTargetRoad_id+ "
, " + message->_sTargetSegment_id + "]"
                              + "\nFrom: [" + message->_sFromRoad_id + " ,
" + message->_sFromSegment_id + "] with " + message->_dFromSpeed.ToString()
```

```
                                    + "\nTime: " + message->_oTime.ToString()
                                    + "\nExpires: " + message-
>_oExpires.ToString()
                                    + "\nCache-Control: " + message-
>_sCacheControl
                                    + "\nTTL: " + message->_iTTL.ToString()
                                    + "\nmsgID: " + message->_smsgID
                                    + "\nContent-Length: " + message-
>_iContentLength.ToString()
                                    + "\n" +
String::Concat(Char::ToString(Char(13)))
                                    + "\n" + message->_sMsgBody;
                        }
                        catch(Exception^ ex){
                              System::Console::WriteLine(ex->ToString());
                              return nullptr;
                        }

                        return returnString;
                  }

            private:
                  //
                  VITPMsgParser();

            };

}
```

VITPMsgParser.cpp

```
#include "stdafx.h"
#include "VITPMsgParser.h"
//
//-------------------------------------------------------------------------
-------------
VITP::VITPMsgParser::VITPMsgParser(){
}
```

VITPMsg.h

```
#include "stdafx.h"

using namespace System;

namespace VITP{

      public ref class VITPMsg
      {
      public:
            ///<summary>
            ///The method of the message. Takes values GET and POST
            ///</summary>
            String^ _sMethod;
            //
            ///<summary>
            ///The version of the VITP protocol used.
            ///</summary>
            Double _dVersion;
            //
```

```
            ///<summary>
            ///The uri part of the message. Is of the form
/<type>/<tag>?[<rc_expr>,…]&<param_expr>&…
            ///</summary>
            String^ _sUri;
            //
            ///<summary>
            ///The road_id of the sender.
            ///</summary>
            String^ _sFromRoad_id;
            //
            ///<summary>
            ///The segment_id of the sender.
            ///</summary>
            String^ _sFromSegment_id;
            //
            ///<summary>
            ///The segment_id of the sender.
            ///</summary>
            Double _dFromSpeed;
            //
            ///<summary>
            ///The road_id of the receiver.
            ///</summary>
            String^ _sTargetRoad_id;
            //
            ///<summary>
            ///The segment_id of the receiver.
            ///</summary>
            String^ _sTargetSegment_id;
            //
            ///<summary>
            ///When the message was sent.
            ///</summary>
            DateTime _oTime;
            //
            ///<summary>
            ///When the message is considered obsolete.
            ///</summary>
            DateTime _oExpires;
            //
            ///<summary>
            ///Cache control commands. Takes values yes, no and fwd for GET
            ///messages and values yes and no for POST messages.
            ///</summary>
            String^ _sCacheControl;
            //
            ///<summary>
            ///If them message if to be stored in cache, this variable
            ///defines the period of time in milliseconds.
            ///</summary>
            Int32 _iTTL;
            //
            ///<summary>
            ///Unique message ID
            ///</summary>
            String^ _smsgID;
            //
            ///<summary>
            ///Size of the message body in bytes.
            ///</summary>
            Int32 _iContentLength;
            //
            ///<summary>
            ///The message body of the VITPMsg
            ///</summary>
```

```
            String^ _sMsgBody;

            ///<summary>The VITPMsg constructor.</summary>
            VITPMsg(String^ Method, Double Version, String^ Uri, String^
FromRoad_id,
                        String^ FromSegment_id, Double FromSpeed, String^
TargetRoad_id, String^ TargetSegment_id,
                        DateTime Time, DateTime Expires, String^
CacheControl, Int32 TTL,
                        String^ msgID, Int32 ContentLength, String^
MsgBody);

      };

}
```

VITPMsg.cpp

```
#include "stdafx.h"
#include "VITPMsgParser.h"

//
//------------------------------------------------------------------------
-------------
VITP::VITPMsg::VITPMsg(String^ Method, Double Version, String^ Uri, String^
FromRoad_id,
                        String^ FromSegment_id, Double FromSpeed, String^
TargetRoad_id, String^ TargetSegment_id,
                        DateTime Time, DateTime Expires, String^
CacheControl, Int32 TTL,
                        String^ msgID, Int32 ContentLength, String^
MsgBody){

            _sMethod = Method;
            _dVersion = Version;
            _sUri = Uri;
            _sFromRoad_id = FromRoad_id;
            _sFromSegment_id = FromSegment_id;
            _dFromSpeed = FromSpeed;
            _sTargetRoad_id = TargetRoad_id;
            _sTargetSegment_id = TargetSegment_id;
            _oTime = Time;
            _oExpires = Expires;
            _sCacheControl = CacheControl;
            _iTTL = TTL;
            _smsgID = msgID;
            _iContentLength = ContentLength;
            _sMsgBody = MsgBody;

}
```

VITPCache.h

```
#include "stdafx.h"

using namespace System;

namespace VITP{
      ///<summary>
      ///This class extends the Hashtable class.
```

```
	///No additional functionalities added.
	///</summary>
	public ref class VITPCache : public System::Collections::Hashtable{
	public:
		VITPCache();
	};
}
```

VITPCache.cpp

```
#include "stdafx.h"
#include "VITPCache.h"

//
//-------------------------------------------------------------------------
-------------
VITP::VITPCache::VITPCache(){
}
```

QueryExecutor.h

```
#include "stdafx.h"
#include "VITPCache.h"

//
//-------------------------------------------------------------------------
-------------
VITP::VITPCache::VITPCache(){
}
```

QueryExecutor.cpp

```
#include "stdafx.h"
#include "QueryExecutor.h"

//
//-------------------------------------------------------------------------
-------------
VITP::QueryExecutor::QueryExecutor(VITPCache^ cache, String^ dFile){
	_oCache = cache;
	_fDiagnosticsFile = dFile;
	_fDFile = gcnew System::Xml::XmlDocument();
	_fDFile->Load(dFile);
	_oMutex = gcnew System::Threading::Mutex(false);
}
//
//-------------------------------------------------------------------------
-------------
array<String^>^ VITP::QueryExecutor::execute(VITPMsg^ message){
	/*		The syntax of the URI is                              */
	/*   /<type>/<tag>?[<rc_expr>,…]&<param_expr>&…        */
	/*   it will be firstly divided to three parts.        */
	/* /<type>/<tag>? , [<rc_expr>,…] and &<param_expr>&…*/

	String^ VITPUri = message->_sUri;
```

```
	String^ method = message->_sMethod;

	//Split the URI to 3 parts
	array<String^>^ tokens = gcnew array<String^>(2);
	tokens[0] = "[";
	tokens[1] = "]";
	array<String^>^ parts = VITPUri->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);

	//Get type and tag
	tokens[0] = "/";
	tokens[1] = "?";
	array<String^>^ typeTag = parts[0]->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);

	//Check message validity
	if(!msgValid(parts[1], message->_oTime)){
		array<String^>^ returnStrings = gcnew array<String^>(3);
		returnStrings[0] = "POST";
		returnStrings[1] = "cnt=0&tout=0";
		returnStrings[2] = "";
		return returnStrings;
	}

	//Call function depending on message type
	if(message->_sMethod->CompareTo("GET") == 0){
		if(typeTag[0]->CompareTo("vehicle")==0)
			return vehicleQuery(method, typeTag[1], parts[1],
parts[2]);
		else if(typeTag[0]->CompareTo("service")==0)
			return serviceQuery(method, typeTag[1], parts[1],
parts[2]);
	}
	else if(message->_sMethod->CompareTo("POST") == 0){
		postQuery(message);
	}

	return nullptr;
}
//
//-------------------------------------------------------------------------
-------------
bool VITP::QueryExecutor::modifyDiagnostics(String^ xpathQuery, XmlNode
^newNode){
	System::Xml::XmlNode^ oldNode;
	System::Xml::XmlNode^ tempNode;
	bool returnVal = true;

	try{
		if(_oMutex->WaitOne(100, false)){
			//Get old node through xpath
			oldNode = _fDFile->DocumentElement->SelectSingleNode("//"
+ xpathQuery);
			//Get parent node
			tempNode = _fDFile->DocumentElement-
>SelectSingleNode("//" + xpathQuery + "/..");

			//If newNode is used instead of _fDFile-
>ImportNode(newNode, true)
			//an exception is thrown
			tempNode->ReplaceChild(_fDFile->ImportNode(newNode,
true), oldNode);
			_fDFile->Save(_fDiagnosticsFile);
		}
	}catch(Exception^ ex){
		System::Console::WriteLine(ex->ToString());
```

```
            returnVal = false;
      }finally{
            _oMutex->ReleaseMutex();
      }

      return returnVal;
}
//
//-------------------------------------------------------------------------
-------------
array<String^>^ VITP::QueryExecutor::vehicleQuery(String^ method, String^
tag, String^ rc_expr, String^ parameters){
      String^ xpathDocument = _fDiagnosticsFile;
      array<String^>^ returnStrings = gcnew array<String^>(3);
      String^ rcResponse;

      //Execute XPath query and return the result
      //vehicle/traffic
      if(tag->CompareTo("traffic") == 0 && method->CompareTo("GET") == 0){
            //Check rc_expression and act accortingly
            rcResponse = modifyRc_expr(rc_expr);
            if(rcResponse->CompareTo("msg_complete") == 0)
                  returnStrings[0] = "POST";
            else
                  returnStrings[0] = "GET";
            returnStrings[1] = rcResponse;
            returnStrings[2] = xpathExecute("vitp/vehicle/average_speed");;
            return returnStrings;
      }
      //vehicle/alert
      if(tag->CompareTo("alert") == 0 && method->CompareTo("GET") == 0){
            String^ returnCoords = "";

            array<Object^>^ cache = gcnew array<Object^>(_oCache->Values-
>Count);
            _oCache->Values->CopyTo(cache, 0);

            for each(VITPMsg^ message in cache){
                  //Create return string
                  array<String^>^ tokens = gcnew array<String^>(2);
                  tokens[0] = "&";
                  tokens[1] = "?";
                  array<String^>^ parts = message->_sUri->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);
                  String^ alertType = parts[3];
                  if(parts[0]->CompareTo("/vehicle/alert") == 0 &&
parameters->CompareTo("&"+parts[3]) == 0){
                        returnCoords = returnCoords + alertType + "?" +
message->_sFromRoad_id->ToString() + "&" + message->_sFromSegment_id-
>ToString() + "\n";
                        rcResponse = modifyRc_expr(rc_expr);
                        returnStrings[0] = "POST";
                        returnStrings[1] = rc_expr;
                        returnStrings[2] = returnCoords;
                        return returnStrings;
                  }
            }
      }
      return nullptr;
}
//
//-------------------------------------------------------------------------
-------------
array<String^>^ VITP::QueryExecutor::serviceQuery(String^ method, String^
tag, String^ rc_expr, String^ parameters){
      String^ xpathDocument = _fDiagnosticsFile;
```

```
	array<String^>^ returnStrings = gcnew array<String^>(3);
	String^ serviceType;
	String^ coordinates;
	String^ rcResponse;

	//Get service type
	serviceType = xpathExecute("vitp/service/type");

	//Check rc_expression
	rcResponse = modifyRc_expr(rc_expr);
	if(rcResponse->CompareTo("msg_complete") == 0)
		returnStrings[0] = "POST";
	else
		returnStrings[0] = "GET";
	returnStrings[1] = rcResponse;

	//Execute XPath query and return the result
	//service/gas
	if(tag->CompareTo("gas") == 0 && method->CompareTo("GET") == 0 &&
serviceType->CompareTo("gas") == 0){
		//Check if service is available
		if(xpathExecute("vitp/service/available")->CompareTo("no") ==
0){
			returnStrings[2] = "unavailable";
			return returnStrings;
		}

		array<String^>^ tokens = gcnew array<String^>(2);
		tokens[0] = "&price";
		tokens[1] = "USD";
		array<String^>^ parts = parameters->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);

		//Check price parameter
		Double price =
Double::Parse(xpathExecute("vitp/service/price"));
		Double priceRequired = Double::Parse(parts[0]->Remove(0,1));

		//Price parameter <
		if(parts[0][0] == '<' && price < priceRequired){
			coordinates = xpathExecute("vitp/service/road_id") + "&"
+ xpathExecute("vitp/service/segment_id");
			returnStrings[2] = coordinates;
		}
		//Price parameter >
		if(parts[0][0] == '>' && price > priceRequired){
			coordinates = xpathExecute("vitp/service/road_id") + "&"
+ xpathExecute("vitp/service/segment_id");
			returnStrings[2] = coordinates;
		}
		//Price parameter =
		if(parts[0][0] == '=' && price == priceRequired){
			coordinates = xpathExecute("vitp/service/road_id") + "&"
+ xpathExecute("vitp/service/segment_id");
			returnStrings[2] = coordinates;
		}
		else
			returnStrings[2] = "";
		return returnStrings;
	}
	return nullptr;
}
//
//-------------------------------------------------------------------------
-------------
bool VITP::QueryExecutor::postQuery(VITPMsg^ message){
```

```
	array<String^>^ tokens = gcnew array<String^>(2);
	tokens[0] = "/";
	tokens[1] = "?";
	array<String^>^ parts = message->_sUri->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);

	try{
		_oCache->Add(parts[1] + "&" + message->_smsgID, message);
		Console::WriteLine("\nMessage with ID " + message->_smsgID + "
added to cache");
		return true;
	}
	catch(Exception^ ex){
		System::Console::WriteLine(ex->ToString());
		return false;
	}
}
//
//-------------------------------------------------------------------------
-------------
bool VITP::QueryExecutor::msgValid(String^ rc_expr, DateTime^ timeSent){
	array<String^>^ tokens = gcnew array<String^>(3);
	tokens[0] = "cnt=";
	tokens[1] = "&tout=";
	tokens[2] = "msec";

	try{
		array<String^>^ parts = rc_expr->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);

		//Not valid if cnt is less than zero
		if(parts[0]->CompareTo("*")!= 0 && int::Parse(parts[0]) <= 0)
			return false;

		//If cnt either * or greater than zero and tout=* then message
valid
		if(parts[1]->CompareTo("*")==0){
			return true;
		}

		int tout = int::Parse(parts[1]);

		//Not valid if tout = 0
		if(tout == 0){
			return false;
		}

		DateTime now = DateTime::Now;
		//Not valid if tout has passed
		if(DateTime::Compare(now, timeSent->AddMilliseconds(tout)) > 0)
			return false;

		return true;
	}
	catch(Exception^ ex){
		System::Console::WriteLine(ex->ToString());
		return false;
	}
}
//
//-------------------------------------------------------------------------
-------------
String^ VITP::QueryExecutor::xpathExecute(String^ xpathQuery){
	System::Xml::XmlNode^ result;

	try{
```

```
            if(_oMutex->WaitOne(100, false)){
                  //Execute XPath query and get the result node
                  result = _fDFile->DocumentElement->SelectSingleNode("//"
+ xpathQuery);
            }
      }catch(Exception^ ex){
            System::Console::WriteLine(ex->ToString());
      }finally{
            _oMutex->ReleaseMutex();
      }

      //If no result returned then message is invalid
      if(nullptr == result)
            return "msg_invalid";

      return result->InnerText;
}
//
//-------------------------------------------------------------------------
-------------
bool VITP::QueryExecutor::saveDiagnosticsFile(){
      try{
            _fDFile->Save(_fDiagnosticsFile);
      }catch(Exception^ ex){
            System::Console::WriteLine(ex->ToString());
            return false;
      }
      return true;
}
//
//-------------------------------------------------------------------------
-------------
String^ VITP::QueryExecutor::modifyRc_expr(System::String ^rc_expr){
      array<String^>^ tokens = gcnew array<String^>(3);
      array<String^>^ returnStrings = gcnew array<String^>(2);
      tokens[0] = "cnt=";
      tokens[1] = "&tout=";
      tokens[2] = "msec";
      array<String^>^ parts = rc_expr->Split(tokens,
System::StringSplitOptions::RemoveEmptyEntries);

      if(parts[0]->CompareTo("*") == 0)
            return rc_expr;

      Int32 temp = Int32::Parse(parts[0]);
      temp = temp - 1;

      if(temp <= 0)
            return "msg_complete";
      else
            return "cnt=" + temp.ToString() + "&tout=" + parts[1] + "msec";

}
```

VITPReceiveThread.cpp

```
#include "stdafx.h"
#include "VITPThread.h"

using namespace System;
using namespace VITP;
using namespace System::Xml;

namespace VITP{
```

```
	public ref class VITPReceiveThread
	{
	public:
		enum class THREADSTATE{
			DEAD = -1,
			ALIVE = 0,
			REQTOEND = 1
		};

		///<summary>
		///The thread object.
		///</summary>
		System::Threading::Thread^ _oThread;
		///<summary>
		///Current thread state.
		///</summary>
		property THREADSTATE ThreadState{
			THREADSTATE get(){
				return _enState;
			}
		}

		///<summary>
		///Creates a new instance of VITPSendThread.
		///</summary>
		VITPReceiveThread();
		!VITPReceiveThread(){
			stopProcess();
		}
		~VITPReceiveThread(){
			stopProcess();
		}
		///<summary>
		///Launches the thread process.
		///</summary>
		void startProcess();

	private:
		THREADSTATE _enState;
		///<summary>
		///The main thread process. It pops messages
		///from the queue and processes them.
		///</summary>
		void threadProcess();
		///<summary>
		///Stops the thread process.
		///</summary>
		void stopProcess();

	};
}
```

VITPReceiveThread.cpp

```
#include "stdafx.h"
#include "VITPReceiveThread.h"

//
//------------------------------------------------------------------------
-------------
VITP::VITPReceiveThread::VITPReceiveThread(){
}
//
```

```
//-------------------------------------------------------------------------
-------------
void VITP::VITPReceiveThread::startProcess(){
	if(nullptr != _oThread)
		return;
	_enState = THREADSTATE::ALIVE;
	_oThread = gcnew Thread(gcnew ThreadStart(this,
&VITP::VITPReceiveThread::threadProcess));
	_oThread->Start();
}
//
//-------------------------------------------------------------------------
-------------
void VITP::VITPReceiveThread::threadProcess(){
	VITP::VITPSocket^ ReceiveSocket = gcnew VITP::VITPSocket(65000);
	VITP::VITPCache^ Cache = gcnew VITP::VITPCache();
	System::Collections::Hashtable^ receivedIds = gcnew
System::Collections::Hashtable();

	//Query Executor
	VITP::QueryExecutor^ QExecutor = gcnew VITP::QueryExecutor(Cache,
"XML/VITP_XML.xml");
	//Process Thread
	VITP::VITPThread^ ProcessThread = gcnew VITP::VITPThread(QExecutor,
"XML/VITP_XML.xml");
	//Received messages log writer
	System::IO::StreamWriter^ logWriter = gcnew
System::IO::StreamWriter("Log/log_received.txt",true);
	String^ receivedMessage;
	VITP::VITPMsg^ VITPMessage;

	//Start thread process and wait some time to initialize
	ProcessThread->startProcess();
	System::Threading::Thread::Sleep(100);
	System::Console::WriteLine("Server ready.\n");
	int count = 0;

	while(_enState == THREADSTATE::ALIVE){
		//Receive message
		receivedMessage = ReceiveSocket->receive();
		//Process received message
		if(nullptr != receivedMessage){
			VITPMessage =
VITPMsgParser::StringToClass(receivedMessage);
			if(nullptr == VITPMessage){
				continue;
			}
			logWriter->WriteLine("Received message with ID " +
VITPMessage->_smsgID + "
on " +

DateTime::Now.ToString("dd/MM/yyyy hh:mm:ss:ffff"));
			logWriter->Flush();

			//Check if message with the same id already received
			if(receivedIds->Keys->Count > 0 && receivedIds-
>ContainsKey(VITPMessage->_smsgID)){
				continue;
			}
			receivedIds->Add(VITPMessage->_smsgID, VITPMessage-
>_smsgID);
			ProcessThread->Push(VITPMessage);
			VITPMessage = nullptr;
			receivedMessage = nullptr;
		}
	}
```

```
	//Set thead state as dead
	_enState = THREADSTATE::DEAD;

	//Stop process thread
	delete ProcessThread;
	ProcessThread = nullptr;

	//Save diagnostics file on exit
	QExecutor->saveDiagnosticsFile();

    return;
}
//
//-------------------------------------------------------------------------
-------------
void VITP::VITPReceiveThread::stopProcess(){
	_enState = THREADSTATE::REQTOEND;
	while(THREADSTATE::DEAD != _enState){
		System::Threading::Thread::Sleep(10);
	}
	delete _oThread;
	_oThread = nullptr;
	Console::WriteLine("\nServer shut down.");
}
```

VITP_Server.cpp

```
// VITP_Server.cpp : main project file.
#include "stdafx.h"
#include "VITPReceiveThread.h"

using namespace System;
using namespace VITP;
using namespace System::Xml;

int main(array<System::String ^> ^args)
{
	//Receive Thread
	VITP::VITPReceiveThread^ ReceiveThread = gcnew
VITP::VITPReceiveThread();
	//Console input
	String^ input;

	Console::WriteLine("To start the server type 'start'. To stop the
server type 'stop': ");

	while(1){
		input = Console::ReadLine();

		//Server Start
		if(input->CompareTo("start") == 0){
			Console::WriteLine("\nServer starting.");
			ReceiveThread->startProcess();
		}
		//Server Stop
		else if(input->CompareTo("stop") == 0){
			Console::WriteLine("\nServer shutting down.");
			delete ReceiveThread;
			break;
		}
	}
	Console::ReadKey();
}
```

Κώδικας XML γραμματικής

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:vitp="http://www.cs.ucy.ac.cy/~cs04ka1"
targetNamespace="http://www.cs.ucy.ac.cy/~cs04ka1">

  <xsd:element name="vitp">
    <xsd:complexType>
      <xsd:sequence>

        <!--Start of vehicle definition-->
        <xsd:element name="vehicle">
          <xsd:complexType>
            <xsd:sequence>

              <!--Values of reg_number are of type ABC123-->
              <xsd:element name="reg_number">
                <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                    <xsd:pattern value="[A-Z][A-Z][A-Z][0-9][0-9][0-9]"/>
                  </xsd:restriction>
                </xsd:simpleType>
              </xsd:element>
              <xsd:element name="current_speed" type="xsd:decimal"/>
              <xsd:element name="average_speed" type="xsd:decimal"/>
              <xsd:element name="fuel" type="xsd:decimal"/>
              <xsd:element name="rpm" type="xsd:decimal"/>
              <xsd:element name="eng_temp" type="xsd:decimal"/>
              <xsd:element name="out_temp" type="xsd:decimal"/>

              <xsd:element name="road_id" type="xsd:double"/>
              <xsd:element name="segment_id" type="xsd:double"/>

            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <!--End of vehicle definition-->

        <!--Start of service definition-->
        <xsd:element name="service">
          <xsd:complexType>
            <xsd:sequence>

              <xsd:element name="type">
                <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="gas"/>
                  </xsd:restriction>
                </xsd:simpleType>
              </xsd:element>

              <!--Price is in Euros-->
              <xsd:element name="price" type="xsd:decimal"/>

              <xsd:element name="available">
                <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="yes"/>
                    <xsd:enumeration value="no"/>
                  </xsd:restriction>
                </xsd:simpleType>
```

```
                </xsd:element>

                <xsd:element name="road_id" type="xsd:double"/>
                <xsd:element name="segment_id" type="xsd:double"/>

              </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
          <!--End of service definition-->

        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

</xsd:schema>
```