ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)

στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

"Αξιολόγηση επιδόσεων ροής δεδομένων διαδικτυακής τηλεόρασης με κεντρική δρομολόγηση πολυδιανομής"

Κουτρούπης Μιχαήλ

ΜΜ4120005

ΑΘΗΝΑ, ΔΕΚΕΜΒΡΙΟΣ 2014

Περίληψη

Το διαδίκτυο είναι ένα παγκόσμιο σύστημα διασυνδεδεμένων δικτύων υπολογιστών που χρησιμοποιούν την στοίβα πρωτοκόλλων Internet (TCP / IP) για την σύνδεση πολλών δισεκατομμυρίων συσκευών ανά τον κόσμο. Πρόκειται για ένα διεθνές δίκτυο δικτύων που αποτελείται από εκατομμύρια ιδιωτικά, δημόσια, ακαδημαϊκά, επιχειρηματικά και κυβερνητικά δίκτυα μεταγωγής πακέτων, που συνδέονται χρησιμοποιώντας ένα ευρύ φάσμα ηλεκτρονικών, ασύρματων και οπτικών τεχνολογιών δικτύωσης.

Το διαδίκτυο δημιουργήθηκε στις μέσα της δεκαετίας του 80 με σκοπό την απομακρυσμένη σύνδεση σε υπολογιστές και υπηρεσίες που για εκείνη την εποχή σπάνιζαν. Από τότε μέχρι σήμερα η χρήση του διαδικτύου έχει αλλάξει. Ο λόγος για αυτήν την αλλαγή είναι ο αυξανόμενος αριθμός συσκευών που επιθυμούν να συνδεθούν, όπως κινητά τηλέφωνα και κινητές συσκευές, αλλά και οι αυξανόμενος αριθμός υπηρεσιών που προσφέρουν όλο και μεγαλύτερο όγκο πληροφοριών. Αξίζει να σημειωθεί ότι πολλές από τις υπηρεσίες αυτές είναι πραγματικού χρόνου. Αυτή η στροφή στην χρήση του διαδικτύου έχει επιφέρει μεγάλες αλλαγές και στα πρωτόκολλα που χρησιμοποιούνται, αλλά έχει αποκαλύψει και πολλές αδυναμίες των ήδη υπαρχόντων πρωτοκόλλων στην προσπάθεια τους να καλύψουν τις νέες ανάγκες. Συνέπεια της στροφής αυτής είναι να οδηγηθεί η ερευνητική και ακαδημαϊκή κοινότητα στην μελέτη εναλλακτικών αρχιτεκτονικών δικτύων και εναλλακτικών μορφών πρωτοκόλλων.

Μια εναλλακτική αρχιτεκτονική δικτύων είναι και η αρχιτεκτονική ICN (Information Centric Networking). Η αρχιτεκτονική αυτή βασίζεται στο γεγονός ότι η χρήση του διαδικτύου πλέον περιστρέφεται γύρω από την παροχή της αναζητούμενης πληροφορίας, η οποία δεν εξαρτάται από τον εξυπηρετητή που την παρέχει αλλά το που βρίσκεται η πληροφορία αυτή καθαυτή. Η αρχιτεκτονική αυτή έχει μερικά σημαντικές διαφοροποιήσεις σε σχέση με την σημερινή αρχιτεκτονική. Αρχικά το ICN διαχωρίζει την πληροφορία από τις πηγές της θεωρώντας την πληροφορία και την πηγή ξεχωριστές έννοιες. Μια βασική προϋπόθεση είναι ότι η πληροφορία είναι αναγνωρίσιμη, μπορεί να βρεθεί και να ταυτοποιηθεί ανεξαρτήτως της τοποθεσίας της. Αυτό έχει σαν αποτέλεσμα η πληροφορία να μπορεί να αναγνωριστεί αμφιμονοσήμαντα από την ονομασία της. Άλλη μια θεώρηση είναι ότι η ανάκτηση της πληροφορίας γίνεται μόνο κατόπιν αιτήματος του χρήστη με αποτέλεσμα το δίκτυο να μην μεταδίδει δεδομένα αν δεν τα ζητήσει ρητά ο χρήστης. Εφόσον ο χρήστης το αιτηθεί, το δίκτυο είναι υπεύθυνο να μεταφέρει την επιθυμητή πληροφορία από την βέλτιστη πηγή στον χρήστη. Όσον αφορά την φορητότητα στο ICN το πρόβλημα των φορητών κόμβων λύνεται με την υλοποίηση του μοντέλου επικοινωνίας δημοσίευσης – εγγραφής. Στο μοντέλο αυτό υπάρχει η δυνατότητα ένας κόμβος να δημοσιεύσει μια πληροφορία πριν κάποιος την ζητήσει καθώς και η δυνατότητα εγγραφής σε μια πληροφορία μετά την δημοσίευσή της. Αυτό το χαρακτηριστικό σε συνδυασμό με την απουσία της πληροφόρησης σχετικά με το πόσοι παρέχουν την πληροφορία και πόσοι την λαμβάνουν, οδηγεί τους φορητούς κόμβους να επανεκπέμπουν την διάθεση τους για εγγραφή σε μια πληροφορία σε κάθε αλλαγή δικτύου και το δίκτυο με την σειρά του να προωθεί τις αιτήσεις αυτές στο κοντινότερο κόμβο που δημοσιεύει την συγκεκριμένη πληροφορία. Μία ακόμη θεώρηση είναι ότι στο θέμα της ασφάλειας, η αρχιτεκτονική ICN βασίζεται στο γεγονός ότι η έναρξη της διαδικασίας μετάδοσης πραγματοποιείται από τον χρήστη ο οποίος θα ζητήσει μια συγκεκριμένη πληροφορία. Αυτό το γεγονός σε συνδυασμό με ότι τα ονόματα των πληροφοριών είναι αυτό-πιστοποιημένα και υπάρχει απουσία πληροφόρησης σχετικά με ποιοι χρήστες παρέχουν πληροφορίες και ποιες, βοηθάει στην καλύτερη ανάπτυξη μηχανισμών ανίχνευσης και προσδιορισμού λανθάνουσας συμπεριφοράς, καθώς και στην μείωση επιθέσεων DoS, μιας και κανένας χρήστης δεν δύναται να ξέρει ποιος του παρέχει την πληροφορία που αιτήθηκε.

Ένα από τα μοντέλα που βασίζονται στην αρχιτεκτονική ICN είναι και το μοντέλο PURSUIT. Στο μοντέλο αυτό υπάρχουν δυο ειδών κόμβοι, οι κόμβοι που αιτούνται πληροφορίες και στέλνουν μηνύματα εγγραφής στο δίκτυο για την πληροφορία που επιθυμούν, και οι κόμβοι που παρέχουν τις πληροφορίες και στέλνουν μήνυμα δημοσίευσης στο δίκτυο ότι διαθέτουν την συγκεκριμένη πληροφορία. Μια λειτουργία του δικτύου είναι η

διαδικασία της προώθησης, που την αναλαμβάνουν οι κόμβοι προώθησης του δικτύου και έχουν ως σκοπό την προώθηση αιτημάτων. Η διαδικασία της συνάντησης είναι άλλη μια λειτουργία του δικτύου που την αναλαμβάνουν οι κόμβοι συνάντησης του δικτύου και έχουν στόχο την διατήρηση της πληροφορίας σχετικά με αντιστοιχίσεις μεταξύ δημοσίευσης και εγγραφής. Τέλος υπάρχει και η διαδικασίας της δημοσίευσης, που την αναλαμβάνουν οι κόμβοι που είναι υπεύθυνοι για την διαχείριση του δικτύου και έχουν ως στόχο την δημιουργία της διαδρομής που θα ακολουθήσουν τα δεδομένα για την μετάδοση από τον εξυπηρετητή στον αιτούντα. Αυτές οι λειτουργίες χρησιμοποιούνται για την υλοποίηση του μοντέλου δημοσίευσης – εγγραφής, ενός μοντέλου που μπορεί να εφαρμοστεί πάνω στην αρχιτεκτονική ICN.

Στο διαδίκτυο την σημερινή εποχή υπάρχει μια αυξανόμενη τάση για χρήση εφαρμογών πολυδιανομής, όπως η διαδικτυακή τηλεόραση, η διαδικτυακή συνδιάσκεψη κ.α. Η δυνατότητα για πολυδιανομή είναι εφικτή στο μοντέλο PURSUIT, καθώς κάθε προώθηση είναι ανεξάρτητη και δεν απαιτείται από τους κόμβους προώθησης να αποθηκεύουν πληροφορίες σχετικά με αυτήν. Όμως για να πραγματοποιηθεί η εύρεση της πληροφορίας και η διαχείριση των εγγεγραμμένων χρηστών για τέτοιου είδους μεταδόσεις απαιτείται παρακολούθηση και εξυπηρέτηση των δημοσιεύσεων με κατανεμημένο τρόπο. Αυτό το επιπλέον φόρτο το αναλαμβάνουν οι κόμβοι συνάντησης με χρήση κατανεμημένου πίνακα καταγραφής συσχετίσεων μεταξύ δημοσιεύσεων και εγγραφών, καθώς και οι κόμβοι διαχείρισης του δικτύου οι οποίοι πρέπει να υπολογίσουν τα μονοπάτια για το δέντρο πολυδιανομής. Για να παραχθεί το δέντρο πολυδιανομής πρέπει να υπολογιστεί το μονοπάτι ανάμεσα στον κάθε αιτών και σε αυτόν που παρέχει την πληροφορία, ώστε η ένωσή τους να δώσει το δέντρο που θα χρησιμοποιηθεί για πολυδιανομή.

Στα δίκτυα πολυδιανομής λοιπόν τα κυριότερα πρόβλημα είναι η διαχείριση του κατανεμημένου πίνακα και ο υπολογισμός του δέντρου πολυδιανομής με βάση τα συντομότερα μονοπάτια. Πάνω σε αυτό το πρόβλημα, δημιουργήθηκε ένα πρόγραμμα σε γλώσσα C++ και χρησιμοποιώντας τον μεταγλωττιστή GCC για να προσομοιώσει ένα δίκτυο ICN το οποίο θα παρέχει υπηρεσίες πολυδιανομής. Πιο συγκεκριμένα, αναπτύχθηκε ένας εξυπηρετητής που παρέχει διαδικτυακή τηλεόραση, και αναλαμβάνει την εξυπηρέτηση αιτημάτων για εγγραφή στην υπηρεσία και για απεγγραφή από την υπηρεσία. Ο εξυπηρετητής επιπρόσθετα αναλαμβάνει την παρακολούθηση των εγγεγραμμένων χρηστών ανά κανάλι, υπολογίζει συντομότερα μονοπάτια και διαχειρίζεται την τοπολογία του δικτύου. Στα πλαίσια αυτής της προσομοίωσης δημιουργήθηκε και ένας πελάτης που σκοπός του είναι να στέλνει μηνύματα στο εξυπηρετητή ζητώντας την εγγραφή του σε κάποιο κανάλι ή την διαγραφή από κάποιο κανάλι. Με την ολοκλήρωση της ανάπτυξης του προγράμματος αυτού, διεξήχθησαν διάφορα πειράματα για την παραγωγή στατιστικών στοιχείων έτσι ώστε να αξιολογηθεί η επίδοση του εξυπηρετητή σε μια τέτοιου είδους αρχιτεκτονική.

Περιεχόμενα

Εισαγωγή

Στην σημερινή εποχή η πρόσβαση στο διαδίκτυο έχει αυξηθεί σημαντικά καθώς η διάδοση των φορητών συσκευών είτε smartphones, είτε tablets έχει δώσει αυξημένες δυνατότητες χρήσης του. Αυτό σε συνδυασμό με το γεγονός ότι το social working είναι πλέον ο πιο διαδομένος τρόπος εργασίας, η ανάγκη για ανάκτηση και ανταλλαγή πληροφοριών είναι μεγαλύτερη από ποτέ. Οι πληροφορίες λοιπόν παίζουν σημαντικό ρόλο σε αυτό το περιβάλλον καθώς και το πόσο γρήγορα μπορούν αυτές να ανακτηθούν.

Από την φύση της η πληροφορία έχει τα εξής βασικά χαρακτηριστικά, εγκυρότητα, ακεραιότητα, επικαιρότητα, πληρότητα, πολυχρησιμότητα, εξατομίκευση και αναδυτικότητα [1]. Όλα αυτά τα χαρακτηριστικά είναι που κάνουν την πληροφορία μοναδική και σημαντική για τον σύγχρονο κόσμο. Το διαδίκτυο αποτελεί πλέον την μεγαλύτερη πηγή πληροφοριών και αυτό έχει σαν αποτέλεσμα να έχει φτιαχτεί ολόκληρο οικοσύστημα γύρω από αυτό που να εκμεταλλεύεται αυτή του την ιδιότητα.

Όμως αυτή η αυξανόμενη πρόσβαση και αυξανόμενη πληθώρα πληροφοριών και διακίνησή τους έχει βασιστεί πάνω σε ένα σύστημα το οποίο δεν σχεδιάστηκε για τον σκοπό αυτό, ούτε έχει τις δυνατότητες πλέον για αποτελεσματική επεκτασιμότητα και αναζήτηση. Για τον λόγο αυτό τα τωρινά προβλήματα που αντιμετωπίζει το διαδίκτυο είναι προβλήματα λόγω της αρχιτεκτονικής του. Το διαδίκτυο δημιουργήθηκε με την προδιαγραφή για απομακρυσμένη σύνδεση σε σπάνιους και ακριβούς πόρους της εποχής εκείνης, όπως περιφερειακά και διακομιστές. Έτσι το διαδίκτυο χτίστηκε πάνω από ένα τηλεφωνικό δίκτυο αλλαγής κυκλωμάτων μεταφέροντας πακέτα από μια συγκεκριμένη αφετηρία προς ένα συγκεκριμένο προορισμό. Η μετάδοση αυτή προϋποθέτει ότι η επικοινωνία γίνεται μεταξύ δύο σταθερών άκρα, κόμβων, και ότι το δίκτυο έχει ως μοναδικό σκοπό την προώθηση των πακέτων από την μία πλευρά στην άλλη. Η διόρθωση, επαναποστολή, η σειρά των πακέτων και ο έλεγχος ροής είναι αρμοδιότητες που οι κόμβοι έπρεπε να λύσουν από άκρο σε άκρο. Η αρχιτεκτονική αυτή οδήγησε σε ένα αξιόπιστο και αποδοτικό δίκτυο δισεκατομμυρίων κόμβων. Με δεδομένο ότι οι ανάγκες πλέον είναι διαφορετικές, και έχουν αλλάξει από την απομακρυσμένη σύνδεση σε διαμοιρασμό πληροφορίας, και με δεδομένο ότι οι κόμβοι, λόγω των φορητών συσκευών, δεν είναι πλέον σταθεροί, δημιουργούνται δυσκολίες στην κάλυψη των αναγκών με βάση την υπάρχουσα υποδομή.

Διάφορες τεχνικές έχουν αναπτυχθεί για την κάλυψη των αναγκών αυτών όπως τα Δίκτυα Διανομής Περιεχομένου (CDN) [2], τα Δίκτυα Peer-to-Peer (P2P) [3] και οι Mobile IP [4] διευθύνσεις. Κάθε μια από αυτές τις τεχνικές έχει εφαρμοστεί πάνω στην υπάρχουσα υποδομή, και έχει χρησιμοποιηθεί ως «μπάλωμα» κάνοντας χρήση άλλων τεχνικών, συχνά με λάθος τρόπο, προκαλώντας ακόμα μεγαλύτερα προβλήματα. Μερικά τέτοια προβλήματα είναι η έλλειψη επεκτασιμότητας και η παραβίαση κανόνων ή πολιτικών που εφαρμόζονται από ορισμένα πρωτόκολλα. Η τεχνική των CDN βασίζεται στην κεντρικοποιημένη προώθηση περιεχόμενου εκ των προτέρων σε επιλεγμένους διακομιστές που είναι διασκορπισμένοι σε διάφορες γεωγραφικές θέσεις για να καλύψουν αιτήματα χρηστών με βάση τη γεωγραφική τους θέση. Αυτό έχει σαν αποτέλεσμα στην προσπάθειά τους να γεφυρώσουν το χάσμα μεταξύ του κεντρικού σημείου εξυπηρέτησης και της αυξημένης διάσπαρτης ζήτησης, να κάνουν κακή χρήση του πρωτοκόλλου DNS [5] και να ανακατευθύνουν αιτήματα των χρηστών στους κοντινότερους διακομιστές για την μετάδοση του επιθυμητού περιεχομένου. Τα δίκτυα P2P επιτρέπουν στους χρήστες την ανταλλαγή δεδομένων δημιουργώντας δίκτυα επικάλυψης σε επίπεδο εφαρμογής πάνω από υπάρχοντα με δική τους δρομολόγηση και λογική προώθησης. Αυτό έχει ως συνέπεια να δημιουργούνται ασυμμετρίες στην μεταφορά των δεδομένων και παραβιάσεις στην πολιτική δρομολόγησης. Η τεχνική του Mobile IP από την άλλη προσπαθεί να καλύψει την ανάγκη των φορητών συσκευών (κινητών κόμβων) στο να λειτουργούν σαν τερματικές συσκευές όταν είναι συνδεδεμένες στο διαδίκτυο. Στην τεχνική αυτή κάθε κινητός κόμβος προσδιορίζεται με βάση τη διεύθυνση του σπιτιού του αγνοώντας την τρέχουσα θέση του στο διαδίκτυο. Έτσι όταν ο κόμβος κινείται σε δίκτυο μακριά από το δικτύου του σπιτιού του, ο κινητός κόμβος λαμβάνει και μια δεύτερη νέα διεύθυνση που προσδιορίζει την τρέχουσα θέση του. Έχοντας δυο διευθύνσεις δημιουργείται ένα tunnel μεταξύ του τρέχοντος σημείου και της

διεύθυνσης του σπιτιού του ώστε να ανακατευθύνονται τα πακέτα από και προς τον κόμβο μέσω του tunnel. Αυτή η λειτουργικότητα μπορεί να καλύπτει την ανάγκη των φορητών συσκευών για σύνδεση στο διαδίκτυο αλλά προκαλεί τριγωνική δρομολόγηση, που έχει σαν αποτέλεσμα να δημιουργείται επιπρόσθετη κίνηση στο διαδίκτυο.

Αρχιτεκτονική ICN

Όλες οι αναφερθείσες αδυναμίες είναι γνωστές εδώ και μερικές δεκαετίες με αποτέλεσμα η επιστημονική κοινότητα να προσπαθεί να βρει τρόπους διαφοροποίησης και αλλαγής του τρόπου διαχείρισης του διαδικτύου. Πολλές από τις εναλλακτικές λύσεις που έχουν εξεταστεί στοχεύουν στην αλλαγή της αρχιτεκτονικής φέρνοντας την πληροφορία ως κεντρικό ρόλο στην σχεδίαση και στη λειτουργία του διαδικτύου. Μία τέτοια αρχιτεκτονική είναι και η αρχιτεκτονική Information-Centric Networking (ICN) και εστιάζει σε τέσσερις βασικές έννοιες και αρχές:

- Την ονοματολογία
- Την μετάδοση της πληροφορίας
- Την φορητότητα και
- Την ασφάλεια

Ονοματολογία

Το διαδίκτυο έχει μεταμορφωθεί από ένα ακαδημαϊκό δίκτυο σε μια παγκόσμια υποδομή για μαζικό διαμοιρασμό πληροφοριών. Η χρήση του γίνεται όλο και μεγαλύτερη και αφορά κυρίως συγκέντρωση πληροφοριών, δεδομένων, εγγράφων, κ.α., οπουδήποτε και αν βρίσκονται. Παρόλα αυτά το διαδίκτυο βασίζεται σε ένα κομβοκεντρικό μοντέλο επικοινωνίας που απαιτεί από τον χρήστη όχι μόνο τον προσδιορισμό της επιθυμητής πληροφορίας αλλά και τον προσδιορισμό του επιθυμητού εξυπηρετητή για το αίτημά του. Λόγω σχεδιασμού το διαδίκτυο δεν έχει τρόπους για εύρεση της επιθυμητής πληροφορίας από την βέλτιστη τοποθεσία για τον χρήστη, εκτός και αν ο χρήστης ξέρει εκ των προτέρων τι και που πρέπει να ζητήσει.

Το ICN διαχωρίζει την πληροφορία από τις πηγές του θεωρώντας την πληροφορία και την πηγή ξεχωριστές έννοιες. Η βασική θεώρηση είναι ότι η πληροφορία είναι αναγνωρίσιμη, μπορεί να βρεθεί και να ταυτοποιηθεί ανεξαρτήτως της τοποθεσίας της. Για τον λόγο αυτό μπορεί να βρίσκεται παντού στο δίκτυο. Στο ICN για να γίνει ένα αίτημα το μόνο που χρειάζεται είναι να προσδιοριστεί η πληροφορία, και όχι το ζεύγος πληροφορία – εξυπηρετητής. Αυτό έχει σαν αποτέλεσμα η ανάκτηση της πληροφορίας να γίνεται μόνο κατόπιν αιτήματος του χρήστη και όχι όπως μέχρι στιγμής ισχύει όπου ο χρήστης είχε απόλυτο έλεγχο πάνω στα δεδομένα που ανταλλάσσονταν. Στο ICN το δίκτυο δεν μεταδίδει δεδομένα αν δεν τα ζητήσει ρητά ο χρήστης και μόλις γίνει αυτό, το δίκτυο είναι υπεύθυνο να φέρει την επιθυμητή πληροφορία από την βέλτιστη πηγή για τον χρήστη.

Μετάδοση της πληροφορίας

Το διαδίκτυο αυτή την στιγμή δεν έχει τους μηχανισμούς για να διαχειριστεί μεγάλες ριπές κυκλοφορίας, γνωστά και ως flash crowds [6], και να μεταδώσει αξιόπιστα την πληροφορία που απαιτείται, καθώς για το δίκτυο όλα τα αντικείμενα του δικτύου είναι μια σειρά από bits που έχουν μια προέλευση και ένα προορισμό. Με βάση το γεγονός αυτό, δεν υπάρχει δυνατότητα για το δίκτυο να μπορεί να κάνει τις απαραίτητες βελτιστοποιήσεις, όπως προσωρινή αποθήκευση εντός του δικτύου, αντιγραφή της πληροφορίας σε διάφορα σημεία του δικτύου κ.α., για να μεταφέρει αποτελεσματικά μεγάλες ποσότητες πληροφορίας. Το διαδίκτυο για να μπορεί να το επιτύχει αυτό πρέπει να αποκτήσει μηχανισμούς που να αναγνωρίζουν την πληροφορία μέσα στο δίκτυο και να επιτρέπουν την ταυτοποίησή της και την ανάκτησή της. Τέτοιοι μηχανισμοί υπάρχουν και είναι εγκατεστημένοι αλλά λειτουργούν σε επίπεδο εφαρμογής, πράγμα που αποδεικνύει ότι δεν είχε προβλεφθεί κάτι τέτοιο κατά την σχεδίαση του.

Τα δίκτυα CDN είναι ένας τέτοιος μηχανισμός που λειτουργεί σε επίπεδο εφαρμογής και επικαλύπτει υπάρχοντα δίκτυα. Παρόλα αυτά στις περιπτώσεις που έχουμε flash crowds, τα δίκτυα αυτά δεν είναι σίγουρο ότι μπορούν να καλύψουν την ζήτηση καθώς δεν μπορούν να προβλέψουν το μέγεθός της, το είδος της πληροφορίας που ζητείται καθώς και το από πού θα ζητηθεί. Επιπρόσθετα τα δίκτυα αυτά χρησιμοποιούν μηχανισμούς που δεν λαμβάνουν υπόψη τους το υποκείμενο δίκτυο με αποτέλεσμα να είναι ανεπαρκής η χρήση των πόρων του δικτύου.

Στο ICN το δίκτυο μπορεί να υποστηρίξει τέτοιου είδους ζήτηση χρησιμοποιώντας τους μηχανισμούς που διαθέτει για εντοπισμό της βέλτιστης πηγής που περιέχει την πληροφορία, καθώς και τους μηχανισμούς για προσωρινή αποθήκευση της πληροφορίας εντός του προσκείμενου δικτύου. Αυτό μπορεί να επιτευχθεί χωρίς επιπλέον μηχανισμούς και δαπανηρές λύσης καθώς το επίπεδο δικτύου λειτουργεί ακριβώς πάνω στην ονομαστική πληροφορία. Οι αρχιτεκτονικές βασισμένες στο ICN γνωρίζουν επακριβώς την πληροφορία που κινείται μέσα στο δίκτυο με αποτέλεσμα να μπορεί να εφαρμόσει βελτιστοποιήσεις είτε σε όλη την πληροφορία είτε σε μέρη της, σε αντίθεση με την τρέχουσα υποδομή του διαδικτύου όπου πρέπει να εφαρμοστούν τεχνικές ανίχνευσης των δεδομένων του πακέτου στους δρομολογητές του δικτύου. Επίσης με την πληροφορία να είναι ονομαστική υπάρχει η δυνατότητα στα δίκτυα ICN της άθροισης όλων των αιτημάτων για την ίδια πληροφορία έτσι ώστε να χρησιμοποιηθεί πολυεκπομπή ακόμα και σε επίπεδο δικτύου.

Φορητότητα

Το σύστημα διευθυνσιοδότησης του διαδικτύου σχεδιάστηκε για σταθερούς κόμβους καθώς ένας κόμβος για να επικοινωνήσει με κάποιον πρέπει να ανήκει στο δίκτυο. Παρόλα αυτά με βάση στατιστικά χρήσης [7] υπάρχει η πρόβλεψη ότι από το 2015 οι φορητοί κόμβοι θα είναι περισσότεροι από τους σταθερούς. Αυτό δημιουργεί ένα επιπρόσθετο πρόβλημα καθώς οι κόμβοι ενώ κινούνται πρέπει να αλλάζουν IP διευθύνσεις επειδή συνδέονται κάθε φορά σε διαφορετικό δίκτυο. Αυτό δημιουργεί επιπλέον τρόπους επικοινωνίας επιτρέποντας την διαλείπουσα και ευκαιριακή συνδεσιμότητα, χωρίς όμως να διασφαλίζει συνεχή συνδεσιμότητα που είναι μια σημαντική απαίτηση, για την οποία υπάρχει συνεχής ζήτηση από τις φορητές συσκευές.

Το πρωτόκολλο Mobile IP, που είναι μια προσωρινή λύση στο πρόβλημα των κινητών κόμβων, απαιτεί τριγωνική δρομολόγηση. Αυτό σημαίνει ότι πρώτα τα πακέτα δρομολογούνται στο κόμβο που βρίσκεται στο δίκτυο του σπιτιού, και από εκεί στην τρέχουσα θέση της φορητής συσκευής μέσω tunnel. Αυτή η τεχνική είναι άκρως αναποτελεσματική καθώς η μετάδοση πρέπει να πραγματοποιηθεί σε μεγαλύτερη απόσταση από την ιδανική. Αυτό κλιμακώνεται ακόμα περισσότερο όταν ο κινητός κόμβος, ο κόμβος που βρίσκεται στο δίκτυο του σπιτιού και ο εξυπηρετητής βρίσκονται σε διαφορετικές ζώνες AS (Autonomous Systems). Το ίδιο ισχύει και στην περίπτωση που η αφετηρία της κίνησης είναι από την φορητή συσκευή, όπου η κυκλοφορία θα δρομολογηθεί αρχικά στο κόμβο που βρίσκεται στο δίκτυο του σπιτιού της, και από εκεί στον επιθυμητό προορισμό. Παρόλα αυτά στην περίπτωση αυτή υπάρχει η πιθανότητα η κίνηση να απορριφθεί καθώς πολλοί δρομολογητές εφαρμόζουν για λόγους ασφάλειας ingress filtering [8], που σημαίνει ότι απαγορεύουν κίνηση από κάποιο δίκτυο προς αυτούς αν δεν είναι από το ίδιο δίκτυο που το παρήγαγε. Το Mobile IP, όπως και όλα τα δίκτυα που επικαλύπτουν υπάρχοντα δίκτυα, παραβιάζουν τις πολιτικές δρομολόγησης του πρωτοκόλλου BGP καθώς τα πακέτα δρομολογούνται πρώτα προς το δίκτυο του σπιτιού του φορητού κόμβου και μετά προς το τρέχον δίκτυο του κόμβου. Αυτό οδηγεί αρχικά σε «valley routing» [9] όταν ένας κόμβος του AS δρομολογεί κίνηση για έναν πάροχο AS, και σε «exit policy violation» [9] όταν η κίνηση φεύγει από ένα σημείο το οποίο είναι διαφορετικό από αυτό που αναμένεται.

Στο ICN το πρόβλημα των φορητών κόμβων λύνεται με την υλοποίηση του μοντέλου επικοινωνίας δημοσίευσης – εγγραφής. Σε αυτό το μοντέλο οι χρήστες εκφράζουν το ενδιαφέρον τους για μια πληροφορία

με την εγγραφή τους σε αυτή. Στην ουσία εκφράζουν το ενδιαφέρον τους στο δίκτυο μέσω μηνύματος, οι χρήστες εκφράζουν την διάθεση της πληροφορίας στο δίκτυο μέσω μηνύματος και οι κόμβοι «brokers» αναλαμβάνουν την ευθύνη να ταυτίσουν εγγραφές με δημοσιεύσεις. Αξίζει να σημειωθεί ότι η ορολογία δημοσίευσης – εγγραφής που χρησιμοποιείται στα πλαίσια του ICN είναι διαφορετική από τα παραδοσιακά συστήματα δημοσίευσης – εγγραφής [10]. Στα παραδοσιακά συστήματα δημοσίευσης – εγγραφής η διάθεση της πληροφορίας σημαίνει και μετάδοσή της ενώ η εγγραφή σηματοδοτεί την άμεση διαθεσιμότητα για λήψη δεδομένων που θα είναι διαθέσιμα στο μέλλον, και προαιρετική διαθεσιμότητα για λήψη δεδομένων που είναι διαθέσιμα στο παρόν. Αντίθετα, στην αρχιτεκτονική ICN η δημοσίευση σηματοδοτεί μόνο την ανακοίνωση για διαθεσιμότητα στο δίκτυο και η εγγραφή αναφέρεται σε δεδομένα που είναι εκείνη την στιγμή διαθέσιμα στο δίκτυο, αφήνοντας προαιρετική την δυνατότητα για μόνιμες εγγραφές.

Η πραγματική δύναμη του μοντέλου δημοσίευσης – εγγραφής είναι στην απουσία υποχρεωτικού συγχρονισμού σε χρόνο και σε χώρο ανάμεσα στις δύο λειτουργίες. Δηλαδή υπάρχει η δυνατότητα ένας κόμβος να δημοσιεύσει μια πληροφορία πριν κάποιος την ζητήσει, καθώς υπάρχει και η δυνατότητα εγγραφής σε μια πληροφορία μετά την δημοσίευσή της. Άλλο ένα σημαντικό στοιχείο του μοντέλου αυτού είναι ότι υπάρχει η άγνοια σχετικά με το πόσοι παρέχουν την πληροφορία αυτή, και από την πλευρά των κόμβων που δημοσιεύουν την πληροφορία αυτή και από την πλευρά των κόμβων που ζητάνε την πληροφορία. Το ίδιο ισχύει και για το πόσοι λαμβάνουν την πληροφορία. Οι κόμβοι που την δημοσιεύουν δεν γνωρίζουν πόσοι την λαμβάνουν και οι κόμβοι που λαμβάνουν δεν γνωρίζουν αν υπάρχουν άλλοι κόμβοι που έχουν ζητήσει την ίδια πληροφορία. Αυτή η ιδιότητα ωφελεί την φορητότητα ιδιαίτερα καθώς οι φορητοί κόμβοι επανεκπέμπουν την διάθεση τους για εγγραφή σε μια πληροφορία σε κάθε αλλαγή δικτύου και έτσι το δίκτυο προωθεί τις εγγραφές αυτές στο κοντινότερο κόμβο που δημοσιεύει την συγκεκριμένη πληροφορία.

Ασφάλεια

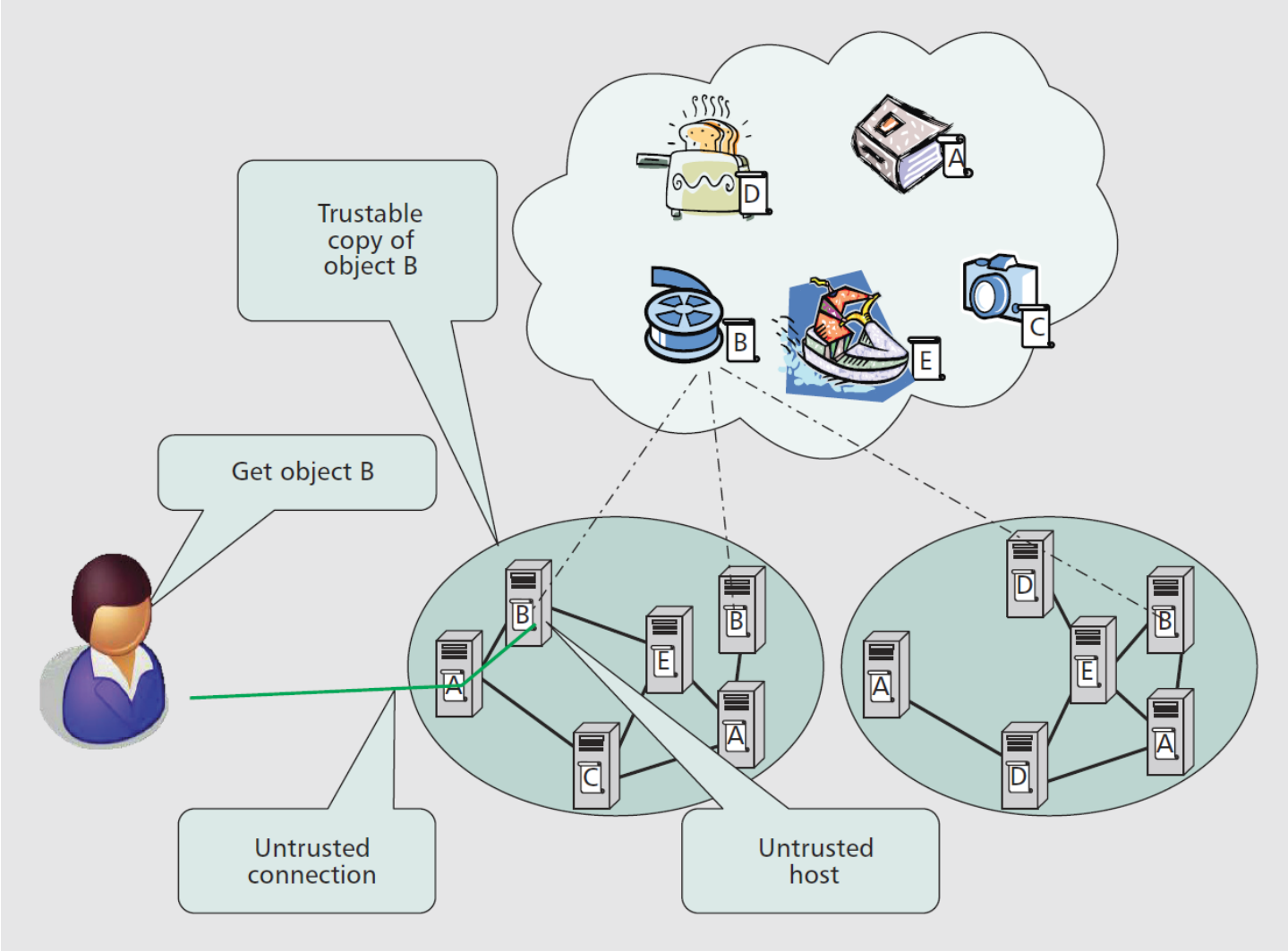
Η αρχική προδιαγραφή του διαδικτύου είναι η λειτουργία του κάτω από ένα απόλυτα αξιόπιστο περιβάλλον, με την αυθεντικοποίηση χρήστη, δεδομένων, την ακεραιότητα των δεδομένων και την ιδιωτικότητα του χρήστη να είναι προαιρετικές προϋποθέσεις, καθώς στόχος της λειτουργίας του ήταν η δυνατότητα προσθήκης καινούργιων κόμβων στο δίκτυο και η ευέλικτη διαχείρισή τους. Αυτό σε συνδυασμό με το ότι το δίκτυο προωθεί οποιαδήποτε κίνηση εισέλθει στο δίκτυο, έδωσε την δυνατότητα σε κακόβουλους χρήστες να πραγματοποιούν επιθέσεις τύπου DoS [11] στην υποδομή του διαδικτύου αλλά και σε μεμονωμένους κόμβους και υπηρεσίες. Η επίθεση αυτού του τύπου προκαλεί κατάρρευση των επιτιθέμενων κόμβων καθώς δέχονται υπερβολικά πολλά αιτήματα προς διαχείριση, εκμεταλλευόμενη την δυνατότητα που της παρέχεται για μαζική προώθηση ή μαζική παραγωγή αιτημάτων. Για την αντιμετώπιση αυτών των φαινομένων έχουν δημιουργηθεί διάφορες τεχνικές για παροχή ασφάλειας όπως τείχη προστασίας, φίλτρα ανεπιθύμητων, αλλά και πρωτόκολλα που συμπληρώνουν και διασφαλίζουν την υφιστάμενη ασφάλεια, όπως IPSec [12], DNSSec [13]. Παρόλα αυτά οι λύσεις αυτές δεν διεισδύουν σε βάθος μέσα στο δίκτυο, με αποτέλεσμα λανθασμένα δεδομένα να εξακολουθούν να υπάρχουν. Ο κύριος λόγος που συμβαίνει αυτό είναι γιατί οι λύσεις αυτές προσφέρουν ασφάλεια από άκρο ως άκρο και όχι από κόμβο σε κόμβο, όπου η ασφάλεια θα ήταν πιο αποτελεσματική.

Πολλά από τα προβλήματα ασφάλειας οφείλονται στο κενό που υπάρχει ανάμεσα στο τι μεταφέρεται και στο τι είναι αυτό που μεταφέρεται. Αυτό οφείλεται στην αρχιτεκτονική του διαδικτύου που δεν παρέχει αυτή την δυνατότητα καθώς το επίπεδο εφαρμογής, που έχει την δυνατότητα προσδιορισμού της πληροφορίας, είναι διαφορετικό από το επίπεδο δικτύου, το οποίο μεταφέρει τα IP πακέτα. Αυτό είναι και σημαντικό τροχοπέδη για λύσεις που προσπαθούν να αναγνωρίσουν την πληροφορία που αποστέλλεται. Οι λύσεις αυτές προσπαθούν να καλύψουν το κενό αυτό με την στατιστική συλλογή δεδομένων και προσομοίωση συμπεριφορών. Τα μειονεκτήματα των λύσεων αυτών είναι ότι έχουν σχετικά μεγάλο κόστος και μπορούν να εφαρμοστούν μόνο σε σημαντικά περιπτώσεις όπως η επιβολή του νόμου. Έτσι ενώ η ασφάλεια από άκρο σε άκρο μπορεί να

επιτευχθεί, δεν υπάρχει η πρόβλεψη για προστασία του ίδιου του διαδικτύου απέναντι σε κακόβουλες επιθέσεις και δεδομένα. Το γεγονός αυτό σε συνδυασμό με την έλλειψη μιας σταθερής πλατφόρμας που να παρέχει την δυνατότητα για ανίχνευση και προσδιορισμό λανθάνουσας συμπεριφοράς αλλά και εκτίμησης των συνεπειών της, χωρίς αυτό ταυτόχρονα να επηρεάζει την απόδοση, δημιουργούν έναν ακόμα πολύ σημαντικό περιορισμό στην προσπάθεια για επίτευξη ασφάλειας στο διαδίκτυο.

Οι αρχιτεκτονικές ICN βασίζονται στην έναρξη της διαδικασίας από τον χρήστη που θα ζητήσει την συγκεκριμένη πληροφορία. Αυτό από μόνο του μειώνει την πιθανότητα να υπάρχουν spam δεδομένα στο δίκτυο αλλά βοηθάει και στην καλύτερη ανάπτυξη μηχανισμών για την ανίχνευσης και προσδιορισμού της λανθάνουσας συμπεριφοράς. Επίσης αυτές οι αρχιτεκτονικές χρησιμοποιούν αυτο-πιστοποιημένα ονόματα για την ονοματολογία των πληροφοριών με αποτέλεσμα να δίνεται η δυνατότητα για ευκολότερο διαχωρισμό μεταξύ των δεδομένων που μεταφέρονται στο δίκτυο, σε επιθυμητά και ανεπιθύμητα. Τέλος, η αναντιστοιχία που υπάρχει στα συστήματα αυτά ανάμεσα στους χρήστες που ζητούν τις πληροφορίες και στους χρήστες που τις παρέχουν, βοηθάει στην μείωση των επιθέσεων DoS καθώς κανένας χρήστης δεν δύναται να ξέρει ποιος του παρέχει την πληροφορία. Έτσι το να μπορεί να προσδιοριστεί κάποιος κόμβος και να πραγματοποιηθεί μια επίθεση DoS προς αυτόν είναι αρκετά δύσκολο, με την δυνατότητα να εξεταστεί το αίτημά του μέσα στο δίκτυο να το κάνει ακόμα δυσκολότερο.

Στο σχήμα 1 παρουσιάζεται η επισκόπηση της αρχιτεκτονικής ICN όπως αναφέρθηκε στις προηγούμενες ενότητες.



Σχήμα 1, Επισκόπηση της αρχιτεκτονικής ICN

Προσεγγίσεις

Οι διάφορες υπάρχουσες προσεγγίσεις σε ICN συστήματα επικεντρώνονται στο να δημιουργήσουν μια αρχιτεκτονική που θα αντικαταστήσει το τωρινό μοντέλο η οποία θα λύσει τα προβλήματα και τους περιορισμούς που έχει το σημερινό διαδίκτυο. Διάφορα μοντέλα έχουν παρουσιαστεί βασισμένα πάνω στην ιδέα του ICN, όπως το DONA του πανεπιστημίου Berkeley, το PURSUIT, ο απόγονος του PSIRP, το SAIL, απόγονος του 4WARD, το COMET, το NDN, απόγονος του CCN κ.α.

Παρότι όλα αυτά τα μοντέλα είναι ακόμα σε φάση σχεδιασμού, έχουν προσεγγίσει τις διάφορες λειτουργίες με διαφορετικό τρόπο. Μερικές από αυτές είναι:

Ονοματολογία

Στην ονοματολογία για την ταυτοποίηση της πληροφορίας όλες οι προσεγγίσεις έχουν χρησιμοποιήσει διαφορετικό είδους ονόματα, με μερικά να είναι ευανάγνωστα για τους ανθρώπους, μερικά άλλα να υποδηλώνουν κάποια ιεραρχία και κάποια άλλα όχι, κ.ο.κ. Παρόλα αυτά σε καμία προσέγγιση το όνομα της πληροφορίας δεν σχετίζεται με την τοποθεσία της.

Ανάλυση ονόματος και δρομολόγηση δεδομένων

Το ICN βασίζεται κυρίως στο να βρεθεί η ζητούμενη πληροφορία και κατ' επέκταση να δρομολογηθεί στον αιτούντα. Αυτό σημαίνει λοιπόν ότι πρέπει να αναλυθεί το όνομα της ζητούμενης πληροφορίας να ταυτοποιηθούν οι πάροχοι που την δημοσιεύουν και μετά να φτιαχτεί ένα μονοπάτι από τον πάροχο στον αιτούντα για να μεταφερθεί η πληροφορία. Ένα σημαντικό σημείο σε αυτό είναι το κατά πόσο αυτές οι δυο λειτουργίες είναι στενά συνδεδεμένες και κατά πόσο όχι. Στην περίπτωση που είναι στενά συνδεδεμένες, όταν το αίτημα για εγγραφή δρομολογηθεί στον πάροχο, τα δεδομένα αποστέλλονται στον χρήστη μέσω του ίδιου μονοπατιού που ακολούθησε και το αίτημα. Στην αντίθετη περίπτωση η δρομολόγηση για το αίτημα προς εγγραφή θα είναι διαφορετική από την δρομολόγηση των δεδομένων που θα αποσταλθούν στο χρήστη.

Προσωρινή αποθήκευση

Στο ICN υπάρχουν δύο δυνατότητες προσωρινής αποθήκευσης, η εντός μονοπατιού αποθήκευση και η εκτός μονοπατιού. Οι διαφορές ανάμεσα σε αυτές είναι, ότι στην εντός μονοπατιού το δίκτυο χρησιμοποιεί την προσωρινή αποθήκευση της πληροφορίας που υπάρχει στο δίκτυο, κατά το μονοπάτι της επίλυσης του ονόματος, ενώ η εκτός μονοπατιού χρησιμοποιεί την προσωρινή αποθήκευση της πληροφορίας που υπάρχει εκτός μονοπατιού. Στα μοντέλα ICN που η ανάλυση του ονόματος και η δρομολόγηση δεδομένων δεν είναι στενά συνδεδεμένες, η εκτός μονοπατιού προσωρινή αποθήκευση θα πρέπει να υποστηριχθεί από το σύστημα ανάλυσης ονόματος, ενώ στην εντός μονοπατιού δρομολόγηση από το σύστημα δρομολόγησης των αιτημάτων για πληροφορίες.

Φορητότητα

Η δυνατότητα ο αιτών να αλλάζει τοποθεσία υποστηρίζεται πλήρως στην αρχιτεκτονική ICN, καθώς το μόνο που χρειάζεται είναι να επανυποβληθούν οι αιτήσεις των χρηστών σε κάθε αλλαγή δικτύου. Ωστόσο η φορητότητα των εξυπηρετητών είναι δυσκολότερη καθώς απαιτείται είτε η ενημέρωση του συστήματος ανάλυσης του ονόματος είτε του πινάκων δρομολόγησης.

Ασφάλεια

Η ασφάλεια είναι περισσότερο συνδεδεμένη με το πώς έχει υλοποιηθεί η λειτουργία της ονοματολογίας. Τα ονόματα που είναι κατανοητά από ανθρώπους απαιτούν ένα αξιόπιστο χρήστη ή μια αξιόπιστη σχέση με το σύστημα ανάλυσης του ονόματος για να επιβεβαιώσει ότι η πληροφορία που λήφθηκε είναι όντως αυτή που αιτήθηκε. Αντίθετα στα ονόματα που δεν είναι κατανοητά από ανθρώπους αλλά υποστηρίζουν αυτοεπιβεβαίωση απαιτούν ένα άλλο αξιόπιστο σύστημα που να αντιστοιχεί ονόματα σε ονόματα κατανοητά σε ανθρώπους.

Αρχιτεκτονική Publish-Subscribe Internet

Μια από τις προτάσεις αυτές είναι και η αρχιτεκτονική Publish-Subscribe Internet (PSI) η οποία επικεντρώνεται στην πληροφορία αυτή καθαυτή και στο γεγονός ότι στο διαδίκτυο υπάρχουν 2 βασικές λειτουργίες, ότι κάποιος δημοσιεύει την πληροφορία (publish) και ότι κάποιος ζητά να ανακτήσει την πληροφορία. Η PSI αρχιτεκτονική επικεντρώνεται στην ανάκτηση πληροφοριών από οπουδήποτε είναι διαθέσιμη, αποφεύγοντας διευθύνσεις και παραδοχές δρομολόγησης που θα μπορούσαν να αναστείλουν την κλιμάκωση και αδιάλειπτη πρόσβαση στις πληροφορίες. Επιπλέον, διευκολύνει την πολλαπλή διανομή, η οποία μπορεί να βελτιώσει την παροχή πληροφοριών αποδοτικότερα και με δυνατότητα επεκτασιμότητας.

Η πληροφορία είναι η κύρια οντότητα στο PSI, με τους χρήστες αναγγέλλοντας την διαθεσιμότητα ή το ενδιαφέρον τους για αυτή. Τα πληροφοριακά αντικείμενα είναι αναγνωρίσιμα και έτσι υπάρχει η δυνατότητα για προσωρινή αποθήκευση και αναπαραγωγή τους με ομοιόμορφο τρόπο.

Αναλυτικότερα στις αρχιτεκτονικές δημοσίευσης – εγγραφής υπάρχουν τρεις κύριες οντότητες, ο χρήστης που δημοσιεύει, ο χρήστης που εγγράφεται και η οντότητα που καθορίζει το σημείο συνάντησης. Οι χρήστες που δημοσιεύουν πληροφορίες έχουν το ρόλο του πληροφοριακού πάροχου, στέλνοντας μηνύματα παροχής πληροφορίας στο δίκτυο και οι χρήστες που εγγράφονται, τον ρόλο των καταναλωτών πληροφοριών, που στέλνουν μηνύματα για εγγραφή στις πληροφορίες. Η οντότητα αρχικά καθορίζει το σημείο συνάντησης αναγνωρίζει τους παρόχους της αιτούμενης πληροφορίας με βάση τα μηνύματα των πελατών για την περιγραφή της πληροφορίας. Έπειτα η οντότητα ξεκινάει την διαδικασία προώθησης της πληροφορίας από τους παρόχους στους πελάτες. Στην όλη διαδικασία η δημοσίευση των πληροφοριών και η εγγραφή σε πληροφορίες μπορούν να γίνουν ασύγχρονα η μία από την άλλη. Η οντότητα διαμορφώνεται από το δίκτυο συνάντησης (RENE) που είναι κυρίως υπεύθυνο για την αντιστοίχιση των εγγραφών με τις κατάλληλες δημοσιεύσεις. Επίσης είναι υπεύθυνο για την αρχικοποίηση της διαδικασίας προώθησης. Ένα δίκτυο συνάντησης αποτελείται από διάφορους κόμβους συνάντησης (RNs), ο καθένας από τους οποίους είναι υπεύθυνος για ένα σύνολο δημοσιεύσεων. Στην αρχιτεκτονική αυτή η κάθε πληροφορία έχει και ένα αναγνωριστικό που δεν είναι κατανοητό από τους χρήστες και προσδιορίζει αμφιμονοσήμαντα την πληροφορία. Η πληροφορία μπορεί να έχει και πεδίο στο οποίο ανήκει. Έτσι οι κόμβοι συνάντησης πρέπει να

ταυτοποιήσουν την πληροφορία με βάση το αναγνωριστικό της και το πεδίο στο οποίο ανήκει. Οι μηχανισμοί πεδίου χρησιμοποιούνται για τον έλεγχο πρόσβασης και δικαιωμάτων σε ένα συγκεκριμένο πεδίο. Αξίζει να σημειωθεί ότι τα πεδία μπορούν να έχουν ιεραρχική δομή. Επίσης μπορούν να υπάρχουν πεδία με φυσική υπόσταση όπως γειτονικά δίκτυα ή λογική υπόσταση όπως κοινωνικά δίκτυα.

Πάνω στην αρχιτεκτονική αυτή έχει στηριχθεί και το σύστημα PURSUIT, το οποίο προσδιορίζει όλες τις λειτουργίες της αρχιτεκτονικής PSI με μεγαλύτερη λεπτομέρεια.

PURSUIT

Το σύστημα Publish Subscribe Internet Technology (PURSUIT) είναι απόγονος του συστήματος Publish Subscribe Internet Routing Paradigm (PRISP) το οποίο χρηματοδοτήθηκε από το πρόγραμμα της Ευρωπαϊκής Ένωσης και αντικαθιστά την στοίβα πρωτοκόλλων IP με την στοίβα πρωτοκόλλων δημοσίευσης – εγγραφής. Η αρχιτεκτονική PURSUIT αποτελείται από τρεις διαφορετικές λειτουργίες: τη συνάντηση, τη διαχείριση τοπολογίας και την προώθηση. Όταν η διαδικασία της συνάντησης ταιριάξει μια εγγραφή με μια δημοσίευση, καθοδηγεί την λειτουργία της διαχείρισης τοπολογίας στο να δημιουργήσει μια διαδρομή ανάμεσα στον αιτούντα και τον εξυπηρετητή. Αυτή η διαδρομή είναι τελικά αυτή που θα χρησιμοποιηθεί από την διαδικασία της προώθησης για την μεταφορά των δεδομένων.

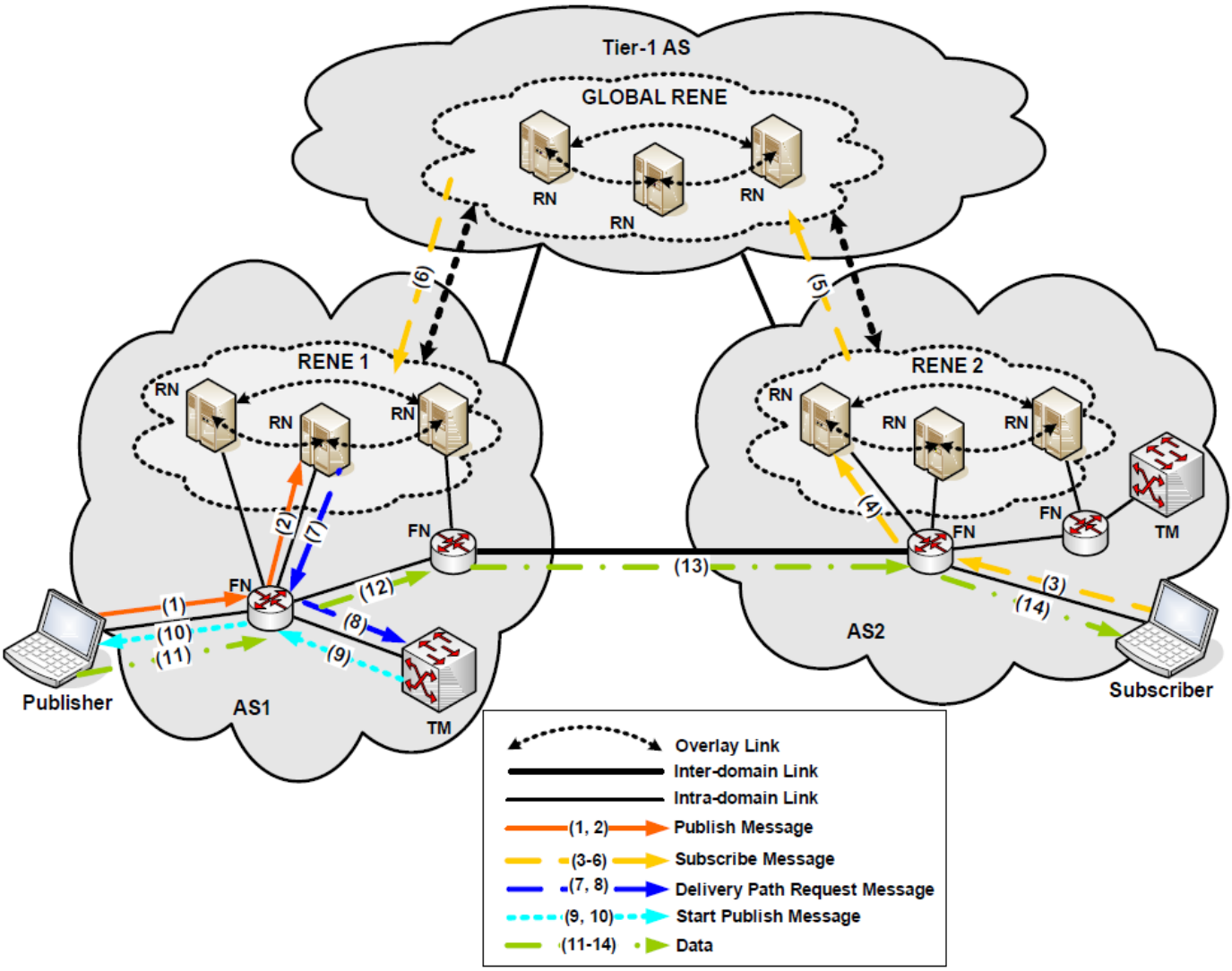
Ονοματολογία

Τα πληροφοριακά αντικείμενα στο PURSUIT αναγνωρίζονται από ένα μοναδικό συνδυασμό από αναγνωριστικά (ID), το αναγνωριστικό πεδίου (scope ID) και το αναγνωριστικό συνάντησης (rendezvous ID). Το αναγνωριστικό πεδίου αναφέρεται στην πληροφορία που σχετίζεται με άλλα πληροφορικά αντικείμενα ενώ το αναγνωριστικό συνάντησης αναφέρεται στην πληροφορία αυτή κάθε αυτή και είναι η πραγματική ταυτότητα του αντικειμένου [14]. Τα πληροφοριακά αντικείμενα μπορούν να ανήκουν σε πολλαπλά πεδία και πιθανότατα με πολλαπλά αναγνωριστικά συνάντησης, αλλά υποχρεωτικά θα ανήκουν σε τουλάχιστον ένα πεδίο. Ο ορισμός του πεδίου βοηθάει στον προσδιορισμό ενός συνόλου από πληροφοριακά αντικείμενα μέσα σε ένα συγκεκριμένο πλαίσιο – περιβάλλον και στην διαμόρφωση ορίων σε μια στρατηγική διάδοσης πληροφορίων σε ένα συγκεκριμένο πλαίσιο – περιβάλλον. Για παράδειγμα ένας εξυπηρετητής μπορεί να δημοσιεύσει μια φωτογραφία κάτω από ένα πεδίο «φίλων» και από ένα πεδίο «οικογένειας», με κάθε πεδίο να ορίζει διαφορετικά δικαιώματα. Στο PURSUIT τα ονόματα των πληροφοριακών αντικειμένων δεν είναι κατανοητά από ανθρώπους και δεν δημιουργούν ιεραρχίες, όμως τα πεδία μπορούν να είναι ιεραρχικά ή οργανωμένα σε οποιαδήποτε μορφή. Έτσι το πλήρες όνομα ενός αντικειμένου αποτελείται από μια ακολουθία αναγνωριστικών πεδίου και ένα μοναδικό αναγνωριστικό συνάντησης.

Ανάλυση ονόματος και δρομολόγηση δεδομένων

Η ανάλυση ονόματος στο PURSUIT πραγματοποιείται από την λειτουργία της συνάντηση, η οποία υλοποιείται από ένα σύνολο από κόμβους συνάντησης (Rendezvous Nodes, RNs) και από ένα δίκτυο συνάντησης (Rendezvous Network, RENE), υλοποιημένο με ένα κατανεμημένο πίνακα κατακερματισμού (DHT), όπως φαίνεται στο σχήμα 2. Όταν κάποιος εξυπηρετητής θέλει να δημοσιεύσει ένα πληροφοριακό αντικείμενο, υποβάλλει ένα μήνυμα *Δημοσίευσης* (Publish) στον τοπικό κόμβο συνάντησης, το οποίο δρομολογείται από τον κατανεμημένο πίνακα κατακερματισμού στον κόμβο συνάντησης που είναι υπεύθυνος για το συγκεκριμένο

πεδίο (βέλη 1 και 2). Όταν ένας κόμβος θέλει να εγγραφεί για το ίδιο πληροφοριακό αντικείμενο, υποβάλλει ένα μήνυμα *Εγγραφής* (Subscribe) στον τοπικό του κόμβο συνάντησης και δρομολογείται από τον κατανεμημένο πίνακα κατακερματισμού στον κόμβο συνάντησης του εξυπηρετητή (βέλη 3 έως 6). Ο κόμβος συνάντησης με την σειρά του καθοδηγεί τον κόμβο που είναι υπεύθυνος για την τοπολογία (Topology Manager) για να δημιουργήσει την διαδρομή που θα ακολουθήσουν τα δεδομένα και θα έχει σαν αφετηρία τον εξυπηρετητή και ως προορισμό τον αιτούντα (βέλη 7 και 8). Ο κόμβος υπεύθυνος για την τοπολογία στέλνει την διαδρομή στον εξυπηρετητή με ένα μήνυμα *Έναρξη Δημοσίευσης* (βέλη 9 και 10), ώστε να την χρησιμοποιήσει για να στείλει το πληροφοριακό αντικείμενο μέσω ενός συνόλου από κόμβους προώθησης (Forwarding Nodes, FNs). Έτσι τελικά ο εξυπηρετητής αρχίζει την μετάδοση της πληροφορίας (βέλη 11 έως 14).



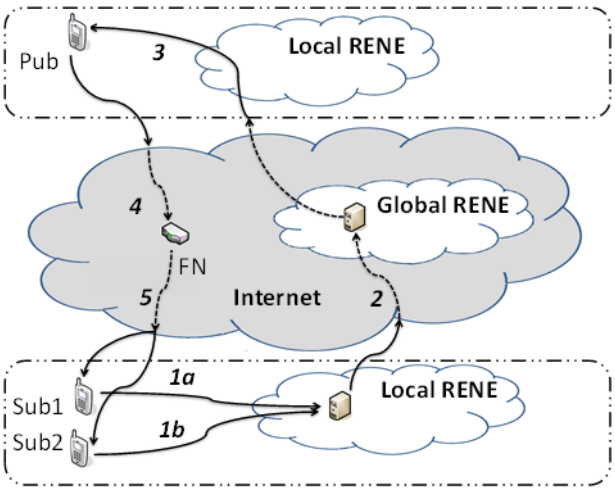
Σχήμα 2, Αρχιτεκτονική PURSUIT

Οι κόμβοι στο PURSUIT που είναι υπεύθυνοι για την τοπολογία υλοποιούν την διαδικασία της διαχείρισης της τοπολογίας με το να εκτελούν ένα κατανεμημένο πρωτόκολλο δρομολόγησης για την εύρεση της τοπολογίας του δικτύου όπως το OSPF. Τα πραγματικά μονοπάτια διανομής υπολογίζονται με την υποβολή του αιτήματος από την διαδικασία της συνάντησης σαν μια σειρά συνδέσμων μεταξύ των κόμβων προώθησης, και κωδικοποιημένα με βάση τις διαδρομές από την πηγή με μια τεχνική βασισμένη στα φίλτρα Bloom [15]. Πιο συγκεκριμένα, κάθε κόμβος του δικτύου δίνει μια ετικέτα σε κάθε εξωτερικό του σύνδεσμο, δηλαδή ένα μεγάλο αριθμό από bit που παράγεται από μια σειρά συναρτήσεων κατακερματισμού, και ανακοινώνει στο δίκτυο τις ετικέτες αυτές μέσω του πρωτοκόλλου δρομολόγησης. Το μονοπάτι παράγεται κάνοντας την πράξη της λογικής διάζευξης (OR) στις ετικέτες για κάθε διαδοχικό σύνδεσμο και ύστερα το τελικό αποτέλεσμα μέσω

φίλτρου Bloom περιλαμβάνεται σε κάθε πακέτο δεδομένων. Όταν το πακέτο δεδομένων φτάνει στον κόμβο προώθησης, αυτός εκτελεί την πράξης της λογικής σύζευξης (AND) στην ετικέτα του κάθε εξωτερικού του συνδέσμου. Αν βρεθεί κάποια ετικέτα που να ταιριάζει τότε το πακέτο προωθείτε στο κατάλληλο σύνδεσμο. Με αυτό τον τρόπο η μόνη κατάσταση που διατηρείται στους κόμβους προώθησης είναι οι ετικέτες των συνδέσμων. Στην περίπτωση που απαιτείται πολυεκπομπή, ολόκληρο το δέντρο πολυεκπομπής κωδικοποιείται με ένα μόνο φίλτρο Bloom.

Η σειρά των πακέτων που αντιστοιχούν στο ίδιο πληροφοριακό αντικείμενο μπορεί να ζητηθεί από τον αιτούντα χρησιμοποιώντας αλγοριθμικά αναγνωριστικά, δηλαδή ονόματα πακέτων παραγμένα από κάποιον αλγόριθμο για τον οποίο όλες οι οντότητες έχουν συμφωνήσει. Αυτές οι αιτήσεις προωθούνται, όπως και με τα πακέτα δεδομένων, χρησιμοποιώντας αντίστροφα φίλτρα Bloom υπολογισμένα από τον κόμβο που είναι υπεύθυνος για την τοπολογία ώστε να παρακάμψουν το δίκτυο συνάντησης, RENE. Αυτό δίνει την δυνατότητα για την ύπαρξη πρωτοκόλλων επιπέδου μεταφοράς για τις εκκρεμείς αιτήσεις, όπως το πρωτόκολλο κυλιόμενου παράθυρου.

Η ανάλυση ονόματος και η δρομολόγηση δεδομένων στο PURSUIT δεν είναι στενά συνδεδεμένες διαδικασίες μιας και η δρομολόγηση οργανώνεται από τον κόμβο που είναι υπεύθυνος για την τοπολογία και εκτελείται από τους κόμβους προώθησης, ενώ η ανάλυση ονόματος γίνεται από το δίκτυο συνάντησης, RENE. Η ανάλυση ονόματος μπορεί να διαρκέσει ένα αρκετά μεγάλο διάστημα καθώς οι κόμβοι που συμμετέχουν στον κατανεμημένο πίνακα κατακερματισμού δεν ακολουθούν τα συντομότερα μονοπάτια για την εύρεση του κατάλληλου κόμβου. Η προώθηση των δεδομένων όμως μπορεί να πραγματοποιηθεί αρκετά γρήγορα μιας και οι κόμβοι προώθησης δεν χρειάζεται να διατηρούν καταστάσεις. Επίσης, ο διαχωρισμός μεταξύ της δρομολόγησης και της προώθησης επιτρέπει στους κόμβους που είναι υπεύθυνοι για την τοπολογία, να υπολογίζουν μονοπάτια χρησιμοποιώντας πολύπλοκα κριτήρια χωρίς να απαιτούν την βοήθεια των κόμβων προώθησης. Παρόλα αυτά, η διαχείριση της τοπολογίας και η λειτουργία της προώθησης, σύμφωνα με την παραπάνω περιγραφή, είναι επαρκείς για περιπτώσεις αιτήσεων και εξυπηρέτησης όταν είναι στο ίδιο δίκτυο. Στις περιπτώσεις που αναφέρονται σε διαφορετικά δίκτυα πρέπει να επεκταθούν, όπως πχ με την μεταγωγή ετικέτας (label switching).

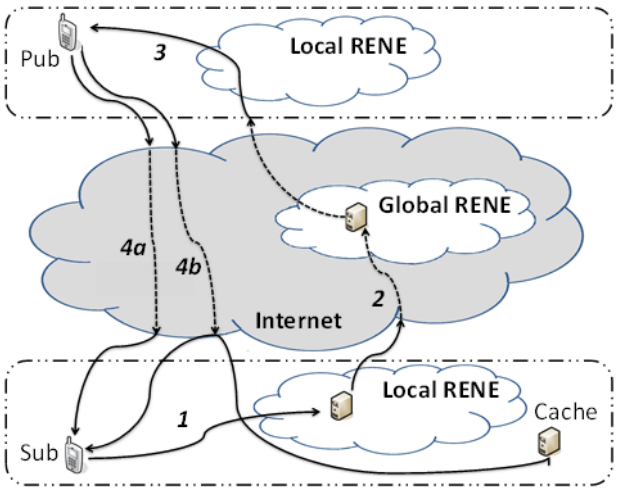


Προσωρινή αποθήκευση

Το Pursuit μπορεί να υποστηρίξει και εντός – μονοπατιού και εκτός – μονοπατιού προσωρινή αποθήκευση. Στην περίπτωση της εντός – μονοπατιού προσωρινής αποθήκευσης τα πακέτα που προωθούνται αποθηκεύονται στους κόμβους προώθησης ώστε να εξυπηρετήσουν μεταγενέστερα αιτήματα. Παρόλα αυτά αυτή η περίπτωση μπορεί να μην είναι πολύ αποτελεσματική λόγω τις ασύνδετης φύσης της ανάλυσης

Σχήμα 3, Εντός - μονοπατιού προσωρινή αποθήκευση

ονόματος και της δρομολόγησης δεδομένων, μιας και τα αιτήματα για το ίδιο πληροφοριακό αντικείμενο μπορούν να φτάσουν στον ίδιο κόμβο συνάντησης και τα δεδομένα να μεταφερθούν από ένα άλλο τελείως διαφορετικό μονοπάτι. Παράδειγμα της εντός – μονοπατιού προσωρινής αποθήκευσης φαίνεται στο σχήμα 3.



Στην περίπτωση της εκτός – μονοπατιού προσωρινής αποθήκευσης, το κάθε δίκτυο αποθηκεύει δημοφιλή πληροφοριακά αντικείμενα ώστε να αποφύγει την χρήση των εξυπηρετητών και κατ' επέκταση να μειώσει το κόστος σύνδεσης και να βελτιώσει την εμπειρία του χρήστη. Στο PSI όλοι οι κόμβοι είναι εν δυνάμει εξυπηρετητές. Έτσι υπάρχει η δυνατότητα να αποθηκευτούν τα δεδομένα με ομοιόμορφο τρόπο είτε σε κόμβους του δικτύου είτε σε ειδικούς κόμβους για αυτήν την λειτουργία, με την επιβάρυνση της διαχείρισης πληροφοριακών αντιγράφων. Το μόνο που χρειάζεται να κάνουν είναι να δημοσιεύσουν το συγκεκριμένο αντίγραφο της πληροφορίας. Παράδειγμα της εκτός – μονοπατιού προσωρινής αποθήκευσης παρουσιάζεται στο σχήμα 4.

Σχήμα 4, Εκτός - μονοπατιού προσωρινή αποθήκευση

Φορητότητα

Η φορητότητα στο PURSUIT διευκολύνεται σε μεγάλο βαθμό από την χρήση της πολυεκπομπής και της προσωρινής αποθήκευσης [16]. Για παράδειγμα ένας τοπικός κινητός κόμβος μπορεί να λαμβάνει πληροφοριακά αντικείμενα μέσω πολυεκπομπής μετά από κάθε αλλαγή δικτύου από γειτονικές τοποθεσίες προσωρινής αποθήκευσης. Ένας κόμβος που βρίσκεται σε ευρύτερης κλίμακας δίκτυο και αλλάζει δίκτυα μπορεί να λαμβάνει πληροφοριακά αντικείμενα με κάποια τροποποίηση στην διαδικασία προώθησης [17]. Επίσης η πρόβλεψη κινητικότητας μπορεί να χρησιμοποιηθεί για να μειώσει την καθυστέρηση μεταξύ των μεταβάσεων του φορητού κόμβου σε διαφορετικά δίκτυα, προωθώντας τα δεδομένα στα δίκτυα που ο κόμβος δύναται να κινηθεί. Έτσι το PURSUIT καλύπτει τύπους φορητότητας είτε σε τοπικό είτε σε ευρύτερο περιβάλλον και είτε σε εναλλαγή τεχνολογιών μεταξύ αλλαγής δικτύων. Παρόλα αυτά η φορητότητα του εξυπηρετητή είναι πιο δύσκολη περίπτωση μιας και η διαδικασία της διαχείρισης της τοπολογίας πρέπει να ενημερώνεται για κάθε νέα θέση του εξυπηρετητή.

Ασφάλεια

Το PURSUIT υποστηρίζει την τεχνική αυθεντικοποίησης πακέτου (Packet Level Authentication, PLA) για την κρυπτογράφηση και την σήμανση των πακέτων. Αυτή η τεχνική εξασφαλίζει την ακεραιότητα δεδομένων, την εμπιστευτικότητα καθώς και τον προσδιορισμό ενός κακόβουλου εξυπηρετητή. Η χρήση της μπορεί να εφαρμοστεί κυρίως στον έλεγχο των πακέτων είτε από τους κόμβους προώθησης είτε από τον τελικό προορισμό. Η χρήση μη ιεραρχικών ονομάτων επιτρέπει την χρήστη αυτο-πιστοποιούμενων ονομάτων για δεδομένα που δεν αλλάζουν, χρησιμοποιώντας την τεχνική του κατακερματισμού του αντικειμένου σαν αναγνωριστικό συνάντησης. Επίσης τα μονοπάτια που είναι κωδικοποιημένα με φίλτρα Bloom μπορούν να χρησιμοποιήσουν δυναμικά αναγνωριστικά συνδέσμου, κάνοντας ανέφικτο για ένα επιτιθέμενο να αλλοιώσει ή να χρησιμοποιήσει παλιά φίλτρα Bloom για την πραγματοποίηση επιθέσεων DoS. Τέλος, έχουν αναπτυχθεί λύσεις για το PURSUIT που στοχεύουν στον περιορισμό των ανεπιθύμητων δεδομένων και στην προστασία της ιδιωτικότητας.

Αρχιτεκτονική Συστήματος

Για την πρακτική εφαρμογή της αρχιτεκτονικής αυτής δημιουργήθηκε ένας πειραματικός εξυπηρετητής που έχει ως στόχο την αξιολόγηση των επιδόσεων του σε περιπτώσεις ροής δεδομένων διαδικτυακής τηλεόρασης σε ένα δίκτυο με αρχιτεκτονική ICN. Δηλαδή ο εξυπηρετητής που δημιουργήθηκε, είναι ένας κόμβος που δημοσιεύει την παροχή διαδικτυακών καναλιών και επιτρέπει σε κόμβους να εγγραφούν στα κανάλια που δημοσιεύει για την λήψη του διαδικτυακού περιεχομένου του. Λόγω της φύσης του εξυπηρετητή, πολύ σημαντικό ρόλο στην αρχιτεκτονική αυτή έχει η πολυδιανομή του περιεχομένου κάθε καναλιού προς όλους τους εγγεγραμμένους κόμβους. Αυτό έχει σαν αποτέλεσμα να απαιτείται ο υπολογισμός των μονοπατιών από τον εξυπηρετητή προς όλους τους κόμβους του για κάθε νέο αίτημα εγγραφής, καθώς πρέπει να υπολογιστεί νέο δέντρο δρομολόγησης που θα συμπεριλαμβάνει το νέο κόμβο και τους ήδη εγγεγραμμένους κόμβους.

Στην αρχιτεκτονική PSI και κατ' επέκταση στο μοντέλο PURSUIT υπάρχουν 3 διαφορετικοί ρόλοι για την ομαλή λειτουργία του δικτύου:

- Ο κόμβος που είναι υπεύθυνος για την τοπολογία του δικτύου, Topology Manager.
- Ο κόμβος που αποτελεί το σημείο συνάντησης, ο Rendezvous Node
- Ο κόμβος που δημοσιεύει το πληροφοριακό περιεχόμενο, ο Publisher

Στην συγκεκριμένη πειραματική αποτίμηση ο εξυπηρετητής περιλαμβάνει και τους 3 αυτούς ρόλους. Έτσι είναι πολύ σημαντικό για την μέτρηση της απόδοσης να μετρηθεί ο χρόνος που απαιτείται για την εύρεση του δέντρου δρομολόγησης των πακέτων, καθώς το σημείο συνάντησης και ο κόμβος που δημοσιεύει το πληροφοριακό περιεχόμενο είναι ο ίδιος ο εξυπηρετητής. Αυτό έχει σαν αποτέλεσμα, ο χρόνος που χρειάζεται για την εγγραφή του νέου κόμβου που ζητά το πληροφοριακό περιεχόμενο και ο χρόνος που απαιτείται για την εύρεση του κόμβου που το δημοσιεύει να είναι ο ίδιος είτε η λειτουργικότητα αυτή υλοποιείται στον ίδιο κόμβο είτε σε άλλο. Το μόνο που διαφέρει στην όλη διαδικασία είναι οι καθυστερήσεις κατά την προώθηση των μηνυμάτων στο δίκτυο που στην περίπτωσή αυτή δεν παίζουν σημαντικό ρόλο. Επίσης ο χρόνος που απαιτείται για την δημοσίευση του περιεχομένου είναι ο ίδιος ανεξαρτήτως αν το υλοποιεί ο ίδιος κόμβος ή κάποιος άλλος, καθώς ο χρόνος για την εκπομπή των πακέτων είναι αμελητέος από την στιγμή που ο κόμβος είναι έτοιμος για την μετάδοση. Έτσι το πιο σημαντικό στοιχείο που χρειάζεται αποτίμηση είναι ο χρόνος υπολογισμού του δέντρου δρομολόγησης που εξαρτάται από την τοπολογία και το πλήθος των κόμβων που είναι εγγεγραμμένοι.

Ο εξυπηρετητής για την πειραματική αποτίμηση δημιουργήθηκε με γνώμονα την γρήγορη εξυπηρέτηση των αιτημάτων καθώς και την αδιάλειπτη λειτουργία, ανεξαρτήτως πλατφόρμας και λειτουργικού συστήματος. Για τον λόγο αυτό επιλέχθηκε η προγραμματιστική γλώσσα C++, ο μεταγλωττιστής GCC και η βιβλιοθήκη Boost.

Αυτές οι τρείς προγραμματιστικές επιλογές ήταν πολύ σημαντικές για τον σχεδιασμό, την ανάλυση και την ολοκλήρωση του προγράμματος, καθώς επιτρέπουν την λειτουργία του προγράμματος και σε λειτουργικό σύστημα Windows, και σε λειτουργικό σύστημα βασισμένο σε Linux – Unix. Έτσι οποιοδήποτε μηχάνημα / εξυπηρετητής μπορεί να λειτουργήσει το πρόγραμμα χωρίς αλλαγή σε αρχεία παρά μόνο με επανάληψη της μεταγλώττισης του με τον συγκεκριμένο μεταγλωττιστή που διαθέτει η πλατφόρμα.

Προγραμματιστική γλώσσα C++

Η προγραμματιστική γλώσσα C++ κατατάσσεται στις προγραμματιστικές γλώσσες γενικού σκοπού και διαθέτει χαρακτηριστικά διαδικασιοστρεφούς, αντικειμενοστρεφούς προγραμματισμού και γενικού προγραμματισμού, ενώ παρέχει και δυνατότητες διαχείρισης της μνήμης.

Η σχεδίασή της έχει βασιστεί στον προγραμματισμό συστημάτων (όπως πυρήνες λειτουργικών συστημάτων, ενσωματωμένα συστήματα) με γνώμονα την αποδοτικότητα, την αποτελεσματικότητα και ευελιξία. Η C++ μπορεί να φανεί πολύ χρήσιμη και στην κάλυψη άλλων αναγκών όπως εφαρμογές υπολογιστών, εξυπηρετητών, σε εφαρμογές κρίσιμης απόδοσης (π.χ. τηλεφωνικοί διακόπτες, ανιχνευτές χώρου κα), καθώς και σε λογισμικό ψυχαγωγίας. Η γλώσσα αυτή ανήκει στα είδη των προγραμματιστικών γλωσσών που απαιτείται μεταγλώττιση. Η εφαρμογή της είναι διαδεδομένη σε διάφορες πλατφόρμες όπως αυτές που παρέχουν οι FSF, LLVM, Microsoft και η Intel.

Η ιστορία της C++

Η C++ έγινε πρότυπο Διεθνή Οργανισμό Τυποποίησης (ISO), με την πιο πρόσφατη και τελευταία έκδοση, που επικυρώθηκε και δημοσιεύτηκε από τον ISO τον Σεπτέμβριο του 2011 σαν ISO/IEC 14882:2011 (ευρέως γνωστή ως C++11). Σαν προγραμματιστική γλώσσα η C++ αρχικά έγινε πρότυπο το 1998 σαν ISO/IEC 14882:1998, και τροποποιήθηκε από την έκδοση και πρότυπο C++03, ISO/IEC 14882:2003. Το τρέχον πρότυπο της C++ 11 αντικαθιστά τις προηγούμενες εκδόσεις, με νέα χαρακτηριστικά και με μια διευρυμένη πρότυπη βιβλιοθήκη.[1]

Η προκαθορισμένη βιβλιοθήκη της C++

Το πρότυπο της C ++ αποτελείται από δύο μέρη: τον πυρήνα της γλώσσας και την προκαθορισμένη βιβλιοθήκη, C++ Standard Library. Σε κάθε έκδοση της C ++ η βιβλιοθήκη παρέχει πολλές χρήσιμες λειτουργίες ώστε να διευκολύνει τους προγραμματιστές. Στην τελευταία έκδοση η βιβλιοθήκη περιλαμβάνει vectors, lists, maps, αλγόριθμους για εύρεση όπως find, for_each, binary_search, random_shuffle, σύνολα, ουρές, στοίβες, πίνακες, πλειάδες, διαδικασίες εισόδου / εξόδου (iostream, για ανάγνωση και εγγραφή από και σε κονσόλα ή αρχεία), έξυπνους δείκτες για την αυτόματη διαχείριση μνήμης, υποστήριξη για κανονικές εκφράσεις, βιβλιοθήκη για παράλληλο προγραμματισμό, υποστήριξη atomics (επιτρέποντας σε ένα το πολύ νήμα να γράφει ή να διαβάζει από μια μεταβλητή σε κάθε στιγμή, χωρίς την ύπαρξη εξωτερικού συγχρονισμού), μεθόδους για διαχείριση χρόνου, γεννήτρια τυχαίων αριθμών καθώς και μετατροπή εξαιρέσεων σε C++ εξαιρέσεις.

Ένα μεγάλο μέρος της βιβλιοθήκης της C++ βασίζεται στην STL (προκαθορισμένη βιβλιοθήκη προτύπων). Αυτό δίνει την δυνατότητα σε αντικείμενα και δομές αποθήκευσης (για παράδειγμα, vectors, lists), να μπορούν να προσπελαστούν με κοινό τρόπο αλλά και να εφαρμοστούν λειτουργίες πάνω τους με ομοιόμορφο τρόπο, όπως η αναζήτηση και η ταξινόμηση. Επιπλέον, πολυδιάστατες δομές όπως πίνακες και σύνολα πολλαπλών διαστάσεων, παρέχουν λειτουργίες για μετατροπή τους σε άλλες συμβατές δομές. Έτσι υπάρχει η δυνατότητα όσο είναι δυνατόν, με τη χρήση προτύπων, να γραφτούν γενικοί αλγόριθμοι που θα μπορούν να λειτουργήσουν ασχέτως με την δομή που χρησιμοποιούν ή τον τρόπο που την προσπελαύνουν. Όπως και στη C, οι λειτουργίες που παρέχει η βιβλιοθήκη γίνονται διαθέσεις με την χρήστη της εντολής #include ακολουθημένη από το όνομα

[1] Περισσότερες πληροφορίες σχετικά με την ιστορία της C++ περιέχονται στο παράρτημα στην σελίδα 78.

της επιθυμητή κεφαλίδας - πρότυπο. Η C++ παρέχει 105 κεφαλίδες - πρότυπο, εκ των οποίων 27 είναι παρωχημένες.

Η ενσωμάτωση της STL βιβλιοθήκης στο πρότυπο αρχικά σχεδιάστηκε από τον Alexander Stepanov [19], ο οποίος πειραματιζόταν με γενικούς αλγορίθμους και πρότυπες δομές για πολλά χρόνια. Όταν ξεκίνησε την ενασχόλησή του με την C ++, συνειδητοποίησε ότι με αυτή τη γλώσσα θα μπορούσε να δημιουργήσει γενικούς αλγορίθμους (π.χ., όπως η ταξινόμηση STL), οι οποίοι αποδίδουν καλύτερα από ό, τι, για παράδειγμα, η βιβλιοθήκη – πρότυπο της C qsort, χάρη στα χαρακτηριστικά της C ++, όπως το επιγραμματικό (inline προσδιορισμό του τύπου και τον προσδιορισμό του τύπου κατά την μεταγλώττιση αντί της χρήσης δεικτών σε συναρτήσεις, όπως και έγινε. Η βιβλιοθήκη της C++ δεν περιέχει αναφορά στην βιβλιοθήκη των προτύπων ως «STL», καθώς είναι πλέον μέρος της, αλλά ο όρος εξακολουθεί να χρησιμοποιείται ευρέως για να ξεχωρίζει από τις υπόλοιπες λειτουργίες της βιβλιοθήκης, όπως ρεύματα εισόδου / εξόδου, διαγνωστικά κ.α.

Οι περισσότεροι μεταγλωττιστές C ++, και όλοι όσοι χρησιμοποιούνται ευρέως, παρέχουν μια υλοποίηση της προκαθορισμένης βιβλιοθήκης που να συμμορφώνεται με το πρότυπο.

Μεταγλωττιστής GCC

Η σειρά μεταγλωττιστών γνωστή και ως GCC είναι ένα σύστημα μεταγλωττιστών για διάφορες γλώσσες προγραμματισμού και υποστηρίζεται από το GNU Project [20]. Οι μεταγλωττιστές αυτοί είναι σημαντικό κομμάτι της αλυσίδας GNU, μιας αλυσίδας προγραμματιστικών εργαλείων για τον προγραμματισμό εφαρμογών και λειτουργικών συστημάτων, οι οποίοι διανέμονται κάτω από την άδεια δημόσιας χρήσης της GNU (GNU GPL) από την Ομοσπονδία Ελεύθερου Λογισμικού (FSF). Αυτό έχει σαν αποτέλεσμα οι GCC να έχουν βοηθήσει πολύ στην ανάπτυξη του ελεύθερου λογισμικού με τη χρήση τους σαν εργαλείο αλλά και σαν παράδειγμα.

Η πρώτη έκδοση του μεταγλωττιστή της GNU εμφανίστηκε το 1987 και ονομαζόταν GNU C Compiler, και μπορούσε να χειριστεί μόνο την γλώσσα C. Τον Δεκέμβριο του ίδιου χρόνου επεκτάθηκε και στην γλώσσα C++ και μετέπειτα και για άλλες γλώσσες προγραμματισμού όπως Objective-C, Objective-C++, Fortran, Java, Ada, και Go.

Ο μεταγλωττιστής GCC [2]έχει μεταφερθεί και σε πολλές αρχιτεκτονικές επεξεργαστών με αποτέλεσμα να θεωρείται ως εργαλείο προγραμματισμού για δωρεάν και ιδιόκτητο λογισμικό. Επίσης το GCC είναι διαθέσιμο και στις ενσωματωμένες πλατφόρμες όπως Symbian, AMCC, και επεξεργαστές τύπου Freescale Power Architecture, αλλά και παιχνιδομηχανές όπως PlayStation 2 and Dreamcast.

Εκτός από το να είναι ο επίσημος μεταγλωττιστής για τα λειτουργικά συστήματα, ο GCC έχει υιοθετηθεί και ως ο προκαθορισμένος μεταγλωττιστής για τα σύγχρονα λειτουργικά συστήματα τύπου Unix, συμπεριλαμβανομένου των εκδόσεων Linux και BSD. Εκδόσεις του συγκεκριμένου μεταγλωττιστής υπάρχουν και για λειτουργικά συστήματα Windows αλλά και για πολλά άλλα.

Σχεδίαση

Η εξωτερική διεπαφή του GCC ακολουθεί τις συμβάσεις του Unix. Οι χρήστες εκτελούν το πρόγραμμα ανάλογα με την γλώσσα που επιθυμούν, gcc για την C, g++ για C++, κτλ, το οποίο διερμηνεύει τα ορίσματα

[2] Η ιστορία του μεταγλωττιστή GCC παρατίθεται στο παράρτημα στην σελίδα 78.

της εντολής καλεί τον πραγματικό μεταγλωττιστή και τρέχει τον assembler στην έξοδο και προαιρετικά μετά τον linker για να παράγει ένα πλήρες εκτελέσιμο αρχείο.

Κάθε μεταγλωττιστής για την κάθε γλώσσα είναι ένα ξεχωριστό πρόγραμμα το οποίο διαβάζει πηγαίο κώδικα και παράγει κώδικα μηχανής. Όλοι έχουν μια κοινή εσωτερική δομή. Το κάθε front end για την κάθε γλώσσα αναλύει τον πηγαίο κώδικα και δημιουργεί ένα γενικό συντακτικό δέντρο.

Αυτό με την σειρά του μετατρέπεται, αν χρειαστεί, σε μια ενδιάμεση μορφή αναπαράστασης, γνωστή ως GENERIC form. Αυτή η μορφή σταδιακά μετατρέπει το πρόγραμμα στην τελική του μορφή. Βελτιώσεις και τεχνικές ανάλυσης στατικού κώδικα, εφαρμόζονται επίσης στο στάδιο αυτό. Αυτές λειτουργούν σε διάφορες αναπαραστάσεις, κυρίως στην ανεξάρτητη αρχιτεκτονικής αναπαράσταση GIMPLE και την εξαρτώμενη από αρχιτεκτονική αναπαράσταση RTL. Τέλος ο κώδικας μηχανής παράγεται χρησιμοποιώντας προσδιορισμό της αρχιτεκτονικής με βάση τον αλγόριθμο του Jack Davidson και Chris Fraser.

Ο GCC αρχικά ήταν γραμμένος σε C εκτός από κομμάτια front end για Ada. Η κύρια διανομή περιλαμβάνει τις προκαθορισμένες βιβλιοθήκες για Ada, C++, και Java, οι οποίες είναι γραμμένες κυρίως στην σχετική γλώσσα. Σε μερικές πλατφόρμες η διανομή περιλαμβάνει και την χαμηλού επιπέδου βιβλιοθήκη κατά τον χρόνο εκτέλεσης, libgcc, η οποία είναι γραμμένη σε ένα συνδυασμό από ανεξάρτητη συστήματος C και κώδικα μηχανής συγκεκριμένο για τον επεξεργαστή, και έχει σχεδιαστεί κυρίως για να χειρίζεται αριθμητικές πράξεις τις οποίες δεν μπορεί ο συγκεκριμένος επεξεργαστής να εκτελέσει απευθείας.

Τον Μάιο του 2010 η οργανωτική επιτροπή του GCC αποφάσισε να χρησιμοποιήσει ένα μεταγλωττιστή C++ για την μεταγλώττιση του GCC χρησιμοποιώντας τις δυνατότητες της C++ για γενικού σκοπού προγραμματισμό.

Τον Αύγουστο του 2012 η οργανωτική επιτροπή του GCC ανακοίνωσε πως η C++ είναι η γλώσσα υλοποίησης του μεταγλωττιστή και έτσι για την κατασκευή του GCC από αρχεία χρειάζεται ένας μεταγλωττιστής που να είναι συμβατός με το πρότυπο ISO/IEC C++03.

Front ends

Κάθε front end χρησιμοποιεί έναν αναλυτή για να δημιουργεί το γενικού συντακτικού δέντρου σύμφωνα με το δοθέν πηγαίο κώδικα. Εξαιτίας του γενικού συντακτικού δέντρου, τα πηγαία αρχεία οποιασδήποτε γλώσσας μπορούν να χρησιμοποιήσουν το ίδιο back end. Ο GCC ξεκίνησε με την χρήση αναλυτών LALR αλλά σταδιακά αντικαταστάθηκαν από χειρόγραφους αναδρομικούς αναλυτές.

Το νόημα του δέντρου ήταν διαφορετικό για κάθε διαφορετικό front end και το καθένα μπορούσε να έχει το δικό του δέντρο κώδικα. Αυτό απλοποιήθηκε με την εισαγωγή του GENERIC και του GIMPLE, δύο νέων μορφών δέντρων ανεξαρτήτου γλώσσας, που ενσωματώθηκαν στην έκδοση 4.0 του GCC. Το GENERIC είναι πιο περίπλοκο και βασίζεται στην ενδιάμεση αναπαράσταση της Java του GCC 3.x. Το GIMPLE είναι μια πιο απλοποιημένη μορφή του GENERIC, το οποίο μετατρέπει δομές σε πολλαπλές εντολές GIMPLE. Τα frond end της C, C++ και Java παράγουν απευθείας GENERIC. Άλλα front end αντίθετα χρησιμοποιούν διάφορες ενδιάμεσες αναπαραστάσεις μετά την ανάλυση και έπειτα τις μετατρέπουν σε GENERIC.

Σε κάθε περίπτωση, η ενδιάμεση μορφή αναπαράστασης μετατρέπεται σε μια απλούστερη μορφή GIMPLE SSA εντολών, έτσι ώστε ανεξαρτήτου γλώσσας και αρχιτεκτονικής να μπορούν να πραγματοποιηθούν βελτιστοποιήσεις.

GENERIC και GIMPLE

Το GENERIC είναι μια ενδιάμεση γλώσσα αναπαράστασης που χρησιμοποιείται ως ενδιάμεση δομή όταν μετατρέπεται ένας πηγαίος κώδικας σε εκτελέσιμο αρχείο. Το GIMPLE είναι ένα υποσύνολο του GENERIC και χρησιμοποιείται από όλα τα front ends του GCC.

Στο ενδιάμεσο στάδιο του GCC, τα βήματα της ανάλυσης κώδικα και οι βελτιστοποιήσεις πραγματοποιούνται ταυτόχρονα πάνω στην μεταγλωττισμένη γλώσσα και στην υποκείμενη αρχιτεκτονική, ξεκινώντας από την αναπαράσταση GENERIC και καταλήγοντας στην RTL.

Βελτιστοποίηση

Η φάση της βελτιστοποίησης μπορεί να συμβεί σε οποιαδήποτε φάση της μεταγλώττισης, αλλά οι περισσότερες βελτιστοποιήσεις γίνονται μετά την συντακτική και σημασιολογική ανάλυση και πριν την δημιουργία του κώδικα για το back end. Για τον λόγο αυτό, αν και είναι λίγο αντιφατικό, το όνομα αυτής της φάσης είναι «middle end».

Το σύνολο των βελτιστοποιήσεων που παρέχεται από το GCC ποικίλει από έκδοση σε έκδοση αλλά συνήθως περιλαμβάνει αλγορίθμους για βελτιστοποίηση επανάληψης, εναλλαγής νημάτων, εξάλειψη υπο-εκφράσεων, σειράς εντολών κ.α. Μερικές βελτιστοποιήσεις πραγματοποιούνται μετά από κάποιες άλλες όπως η εξάλειψη κώδικα που δεν χρησιμοποιείται, η εξάλειψη μερικού πλεονασμού, η αρίθμηση καθολικών τιμών, η αραιή διάδοση μεταβλητής που χρησιμοποιείται για τον έλεγχο συνθηκών κ.α. Οι βελτιστοποιήσεις που αφορούν πίνακες, όπως αυτόματος παραλληλισμός, πραγματοποιούνται επίσης στη φάση αυτή.

Back end

Η συμπεριφορά του back end ενός GCC εξαρτάται κυρίως από τις προ επεξεργαστικές μακροεντολές και λειτουργίες που αντιστοιχούν σε μια συγκεκριμένη αρχιτεκτονική όπως το μέγεθος λέξης, το είδος endian κ.α. Το μπροστινό κομμάτι του back end χρησιμοποιείται για την δημιουργία του RTL. Έτσι παρόλο που το RTL του GCC είναι ανεξάρτητο από τον επεξεργαστή, η αρχική ακολουθία από γενικές εντολές έχουν ήδη χρησιμοποιηθεί για τον επεξεργαστή που θα τις εκτελέσει. Σε κάθε περίπτωση οι πραγματικές εντολές RTL που αντιστοιχούν στην αναπαράσταση του προγράμματος διαμορφώνονται από την αρχιτεκτονική που στοχεύουν.

Το αρχείο που περιγράφει την σύνθεση του μηχανήματος περιλαμβάνει μοτίβα RTL μαζί με περιορισμούς τελεστών και κομμάτια κώδικα για την εξαγωγή του τελικού αποτελέσματος. Οι περιορισμοί δείχνουν ότι ένα συγκεκριμένο μοτίβο μπορεί να εφαρμοστεί μόνο σε συγκεκριμένους καταχωρητές ή να επιτρέπουν άμεσες μετατοπίσεις τελεστών μόνο συγκεκριμένου μεγέθους, πχ 12, 16, 24 bit κοκ. Κατά την δημιουργία του RTL ελέγχονται οι περιορισμοί για την συγκεκριμένη αρχιτεκτονική. Για να χρησιμοποιηθεί ένα κομμάτι κώδικα RTL πρέπει να ταιριάζει με ένα ή περισσότερα πρότυπα RTL του αρχείου περιγραφής του μηχανήματος και να ικανοποιεί τους περιορισμούς του πρότυπου, αλλιώς είναι αδύνατη η μετατροπή του τελικού RTL σε γλώσσα μηχανής.

Για την ολοκλήρωση της μεταγλώττισης, απαιτείται ένα έγκυρο RTL το οποίο να μπορεί να μετατραπεί σε μια πιο αυστηρή μορφή στην οποία η κάθε εντολή θα απευθύνεται στους καταχωρητές της πραγματικής μηχανής που αναφέρονται στο αρχείο περιγραφής της μηχανής. Η μετατροπή αυτή είναι ένα περίπλοκο και σημαντικό

βήμα, γιατί οι πραγματικοί καταχωρητές επιλέγονται για να αντικαταστήσουν τους ψευτο – καταχωρητές. Μετά από αυτό το βήμα ακολουθεί η φάση της «επαναφόρτωσης» στην οποία οι ψευτο – καταχωρητές που δεν ανατέθηκαν αλλάζουν με ένα καταχωρητή που είναι υπεύθυνος για την στοίβα και το RTL μετατρέπει τα δεδομένα του για χρήση της στοίβας. Ομοίως για μετατοπίσεις που είναι πολύ μεγάλες για να εκτελεστούν σε μία μόνο εντολή, διασπώνται και αντικαθίστανται από ακολουθίες RTL που θα πληρούν τους περιορισμούς των μετατοπίσεων.

Στην τελική φάση, ο κώδικας μηχανής δημιουργείται από την εκτέλεση ενός μικρού κομματιού κώδικα που σχετίζεται με κάθε πρότυπο, με αποτέλεσμα να δημιουργήσει τις πραγματικές εντολές με την χρήση των τελικών καταχωρητών, των τελικών μετατοπίσεων και των διευθύνσεων μνήμης, που επιλέχθηκαν από το στάδιο της «επαναφόρτωσης», για την σειρά εντολών που απαιτείται. Το κομμάτι κώδικα για την δημιουργία του τελικού αρχείου κώδικα μηχανής μπορεί να ποικίλει, από ένα αλφαριθμητικό μέχρι ένα κομμάτι κώδικα C, ώστε να παραχθεί τελικά ένας έγκυρος κώδικας μηχανής.

Αρχιτεκτονικές που υποστηρίζονται

Η σειρές επεξεργαστών που υποστηρίζει ο μεταγλωττιστής GCC από την έκδοση 4.3 και μετά, είναι:

- Alpha
- ARM
- AVR
- Blackfin
- Epiphany (GCC 4.8)
- H8/300
- HC12
- IA-32 (x86)
- IA-64 (Intel Itanium)
- MIPS
- Motorola 68000
- PA-RISC
- PDP-11
- PowerPC
- R8C / M16C / M32C
- SPARC
- SPU
- SuperH
- System/390 / zSeries
- VAX
- x86-64

Λιγότερο γνωστοί επεξεργαστές που υποστηρίζονται στην κανονική έκδοση είναι οι:

- 68HC11
- A29K
- CR16
- C6x
- D30V
- DSP16xx
- ETRAX CRIS
- FR-30
- FR-V
- Intel i960
- IP2000
- M32R
- MCORE
- MIL-STD-1750A
- MMIX
- MN10200
- MN10300
- Motorola 88000
- NS32K
- ROMP

- RL78
- Stormy16
- V850
- Xtensa

Σειρά επεξεργαστών που υποστηρίζονται από τις εκδόσεις του GCC, ξεχωριστά από την έκδοση FSF:

- Cortus APS3
- ARC
- AVR32
- C166 and C167
- D10V
- EISC
- eSi-RISC
- Hexagon[44]
- LatticeMico32
- LatticeMico8
- MeP
- MicroBlaze
- Motorola 6809
- MSP430
- NEC SX architecture[45]
- Nios II and Nios
- OpenRISC
- PDP-10
- PIC24/dsPIC
- PIC32
- Propeller
- Saturn (HP48XGCC)
- System/370
- TIGCC (m68k variant)
- TriCore
- Z8000

Βιβλιοθήκη Boost

Η βιβλιοθήκη Boost [21] είναι ένα σύνολο από βιβλιοθήκες για την προγραμματιστική γλώσσα C++ και παρέχει δυνατότητες για την κάλυψη σχεδόν όλων των αναγκών και λειτουργιών για ένα προγραμματιστή, όπως κανονικές εκφράσεις, δομές δεδομένων, επεξεργασία εικόνων κ.α. Περιέχει περίπου στις ογδόντα διαφορετικές βιβλιοθήκες, οι οποίες διατίθενται κάτω από την γενική άδεια λογισμικού της Boost, δηλαδή για την χρήση τους σε ιδιόκτητα και δωρεάν προγράμματα.

Πολλοί από τους ιδρυτές της Boost είναι στην επιτροπή προτύπων της C++ [22] και πολλές βιβλιοθήκες της Boost έχουν ενσωματωθεί και στην τεκμηρίωση και στην χρήση των εκδόσεων της C++, ειδικά στην έκδοση 11. Για τον λόγο αυτό η Boost στοχεύει στην ενσωμάτωση βιβλιοθηκών που λειτουργούν καλά με την καθιερωμένη βιβλιοθήκη της C++, STL, αλλά και σε αυτές που είναι φορητές για να λειτουργούν σε όλα τα συστήματα. Για την ακρίβεια οι βιβλιοθήκες αυτές περιέχουν συνήθεις πρακτικές και αναφορές ώστε να είναι κατάλληλες για την χρήση τους ως πρότυπο. Δέκα βιβλιοθήκες είναι ενσωματωμένες στην καθιερωμένη βιβλιοθήκη της C++ 11 και έχουν προταθεί ακόμα περισσότερες για την συμπερίληψή τους στην C++17.

Η έκταση και η ποιότητα των βιβλιοθηκών είναι τέτοια ώστε να μπορούν να χρησιμοποιηθούν σε όλα τα είδη προγραμμάτων και από όλα τα επίπεδα προγραμματιστών. Έτσι περιέχουν από βιβλιοθήκες γενικού σκοπού όπως έξυπνους δείκτες, μέχρι χρήση λειτουργιών του λειτουργικού συστήματος ανεξαρτήτου πλατφόρμας όπως το σύστημα αρχείων της Boost. Τα επίπεδα των προγραμματιστών που μπορούν να την χρησιμοποιήσουν είναι από αρχάριους μέχρι εξειδικευμένους προγραμματιστές C++, που να είναι εξοικειωμένοι με τον προγραμματισμό προτύπων, MPL, και γλωσσών συγκεκριμένου πεδίου, DSL.

Για την διασφάλιση της αποτελεσματικότητας και της ευελιξίας, η Boost κάνει εκτεταμένη χρήση των προτύπων, και για τον λόγο αυτό πολλές έρευνες έχουν βασιστεί σε στοιχεία της Boost για γενικό προγραμματισμό και μετα – προγραμματισμό. Οι περισσότερες βιβλιοθήκες είναι βασισμένες σε κεφαλίδες, και περιέχουν inline μεθόδους και πρότυπα. Για τον λόγο αυτό δεν χρειάζεται την εκ των προτέρων μεταγλώττιση τους για την χρήση τους. Επίσης κάποιες βιβλιοθήκες υπάρχουν ως ανεξάρτητες από τις υπόλοιπες, οπότε μπορούν να χρησιμοποιηθούν αυτούσιες χωρίς να υπάρχουν εξαρτήσεις.

Η κοινότητα της Boost είναι αρκετά μεγάλη και είναι ανοιχτή για νέα μέλη καθώς αποτελείται από άτομα και επιχειρήσεις όλων των ειδών, και αυτό είναι ένας από τους λόγους που έχει γίνει τόσο πετυχημένη και διαδεδομένη.

Συμβατότητα με μεταγλωττιστές

Οι κύριοι μεταγλωττιστές στους οποίους έχει δοκιμαστεί η Boost είναι:

Για λειτουργικά συστήματα Linux:

- Clang: 3.0, 3.1, 3.2, 3.3, 3.4
- Clang, C++14: 3.5
- GCC: 4.4.7, 4.5.3, 4.6.4, 4.7.3, 4.8.1, 4.8.2
- GCC, C++98: 4.9.1
- GCC, C++11: 4.4.7, 4.8.2, 4.8.3, 4.9.1
- GCC, C++14: 4.9.1
- Intel: 13.1, 14.0
- Intel, C++11: 13.1, 14.0
- QCC: 4.4.2

Για OS X:

- Apple Clang: 6.0
- Apple Clang, C++11: 6.0
- Apple Clang, C++14: 6.0
- GCC: 4.2.1, 4.9.1
- Intel: 12.0

Για Windows:

- GCC, mingw: 4.4.0, 4.4.7. 4.5.4, 4.6.3, 4.7.2, 4.7.3, 4.8.0, 4.8.2, 4.9.0
- Visual C++: 8.0, 9.0, 10.0, 11.0, 12.0
- FreeBSD:
- GCC: 4.2.1
- QNX:
- QCC: 4.4.2

Επιπρόσθετοι μεταγλωττιστές στους οποίους έχει δοκιμαστεί η Boost είναι:

Για Linux:

- Clang: 3.0, 3.1, 3.2, 3.3, 3.4.2
- Clang, C++14: 3.5.0, trunk
- GCC: 4.4.7, 4.6.4, 4.7.3, 4.8.1, 4.8.2, 5.0 (experimental)
- GCC, C++11: 4.4.7, 4.8.2, 4.8.3, 4.9.1
- GCC, C++14: 4.9.1
- Intel: 11.1, 12.1, 13.0, 13.1, 14.0
- Intel, C++11: 13.1, 14.0

Για OS X:

- Apple Clang: 6.0
- Apple Clang, C++11: 6.0
- Apple Clang, C++14: 6.0
- Clang: trunk
- Clang, C++11: trunk
- GCC: 4.2.1, 4.9.1
- Intel: 12.0

Για Windows:

- GCC, mingw: 4.4.0, 4.4.7, 4.5.4, 4.6.3, 4.7.3, 4.8.0, 4.8.2, 4.9.0
- Visual C++: 8.0, 9.0, 10.0, 11.0, 12.0

Για FreeBSD:

- GCC: 4.2.1

Και για QNX:

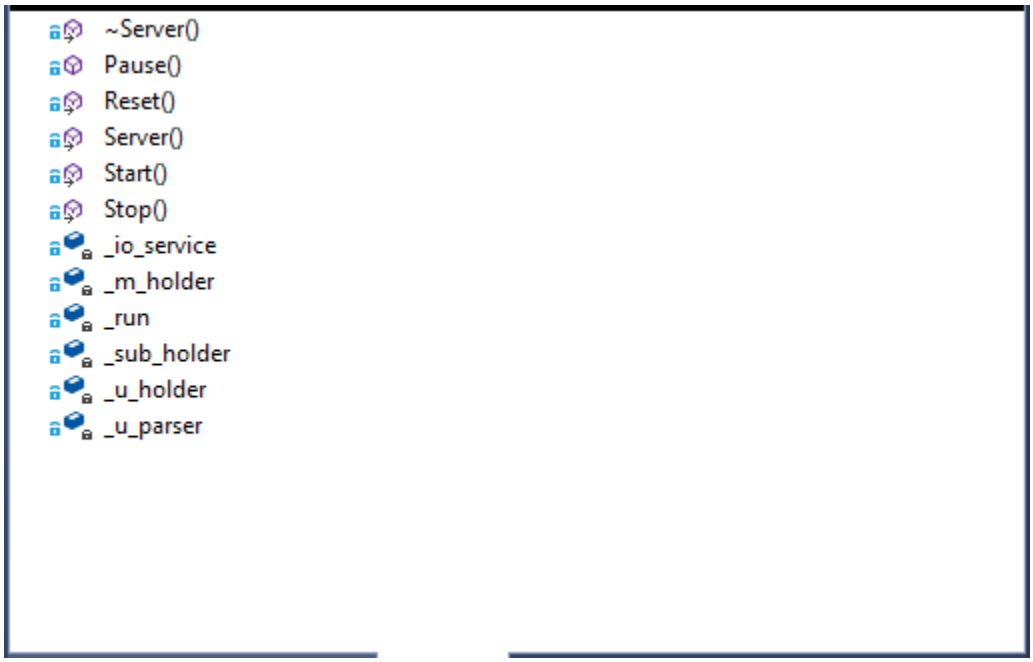
- QCC: 4.4.2

Δομή εξυπηρετητή

Ο εξυπηρετητής λοιπόν, που δημιουργήθηκε για την πειραματική αποτίμηση, προγραμματίστηκε σε γλώσσα C++ έκδοσης 11, χρησιμοποιώντας για μεταγλώττιση τον GCC, έκδοσης 4.8 και την βιβλιοθήκη Boost, έκδοσης 1.57.0[3]. Ειδικότερα το περιβάλλον ανάπτυξης ήταν σε περιβάλλον Windows 8.1 x64 και με εργαλείο προγραμματισμού, IDE, το Visual Studio 2013 Update 4. Το περιβάλλον εξομοίωσης για το οποίο έγινε και η μεταγλώττιση, ήταν Windows x86 ώστε να μπορέσει να λειτουργήσει ο εξυπηρετητής σε οποιοδήποτε μηχάνημα με λειτουργικό σύστημα Windows, ανεξαρτήτως έκδοσης. Αυτό έχει όμως σαν μειονέκτημα ότι ο εξυπηρετητής δεν εκμεταλλεύεται τους καταχωρητές των 64bit, που έχουν τα νεότερα συστήματα, με αποτέλεσμα να έχει μια πιθανή καθυστέρηση στην απόδοση σε σχέση με εκείνα. Αυτή η επιλογή ήταν μια συνειδητή επιλογή και επιλέχθηκε έναντι της επιλογής για λειτουργία του εξυπηρετητή μόνο σε x64 συστήματα.

Server

Η τάξη που υλοποιεί τον εξυπηρετητή είναι η Server και είναι υπεύθυνη στο να παρέχει ένα ομοιογενές και αδιαφανή τρόπο για την διαχείριση του εξυπηρετητή. Δηλαδή να παρέχει τις βασικές λειτουργίες εκκίνησης, παύσης, προσωρινής παύσης και επαναφοράς, χωρίς να αποκαλύπτει εσωτερικές λειτουργίες και τρόπους που αυτή το πραγματοποιεί. Έτσι λοιπόν παρέχει τις αντίστοιχες μεθόδους που το πραγματοποιούν και είναι η Start, Stop, και Reset. Αυτές οι μέθοδοι στην ουσία διαχειρίζονται ένα εσωτερικό αντικείμενο της τάξης, το αντικείμενο τύπου *boost::asio::io_service*, της βιβλιοθήκης *boost/asio.hpp*, το οποίο παρέχει έναν ομοιόμορφο τρόπο για χρήση υπηρεσιών εισόδου – εξόδου του λειτουργικού συστήματος. Επιπλέον η τάξη αυτή διαχειρίζεται και αρχικοποιεί τα υπόλοιπα αντικείμενα που είναι αναγκαία για την λειτουργία της μέσω του κατασκευαστή της. Αυτό έχει σαν αποτέλεσμα αν κάποιο από τα εξαρτώμενα αντικείμενα της δεν μπορεί να αρχικοποιηθεί να πετάξει εξαίρεση και να μην δημιουργηθεί το αντικείμενο. Η δομή του εξυπηρετητή παρουσιάζεται στο σχήμα 5.



Σχήμα 5, Η τάξη Server

[3] Στο παράρτημα Εκδόσεις βιβλιοθήκης Boost παρουσιάζονται μερικές από τις πιο πρόσφατες εκδόσεις της μαζί με αλλαγές στις επιμέρους βιβλιοθήκες.

Ο εξυπηρετητής αποτελείται από λογικής άποψης από τέσσερα διαφορετικά συστατικά:

- Τον UniverseFileParser
- Τον UniverseHolder
- Τον SubscriptionHolder
- Τον IMessageHolder

UniverseFileParser

Το συστατικό με το όνομα *UniverseFileParser* είναι υπεύθυνο για την ανάγνωση του αρχείου που περιέχει την τοπολογία του δικτύου και την μετατροπή του σε μια μορφή ευκολότερα διαχειρίσιμη.

Ειδικότερα, η τάξη *UniverseFileParser* κληρονομεί μία διεπαφή με όνομα *IUniverseParser* και υλοποιεί τις δύο σημαντικές λειτουργίες της, την Validate και την Parse. Η Validate είναι μια μέθοδος που στην ουσία εξετάζει το αρχείο πριν το διατρέξει και πιστοποιεί αν η μορφή του είναι αυτή που αναμένεται ή όχι. Η Parse δε, είναι η πραγματική μέθοδος που διατρέχει το αρχείο και το μετατρέπει στην μορφή ενός ευρετηρίου, map, που έχει για κλειδί τον κωδικό του κόμβου και για τιμές ένα σύνολο, vector, από κωδικούς κόμβων που συνορεύει. Η δομή αυτή έχει την συγκεκριμένη δομή ώστε να αντικατοπτρίζει την τοπολογία του δικτύου. Την δομή μπορούμε να την προσπελάσουμε από την δημόσια μεταβλητή m_universe.

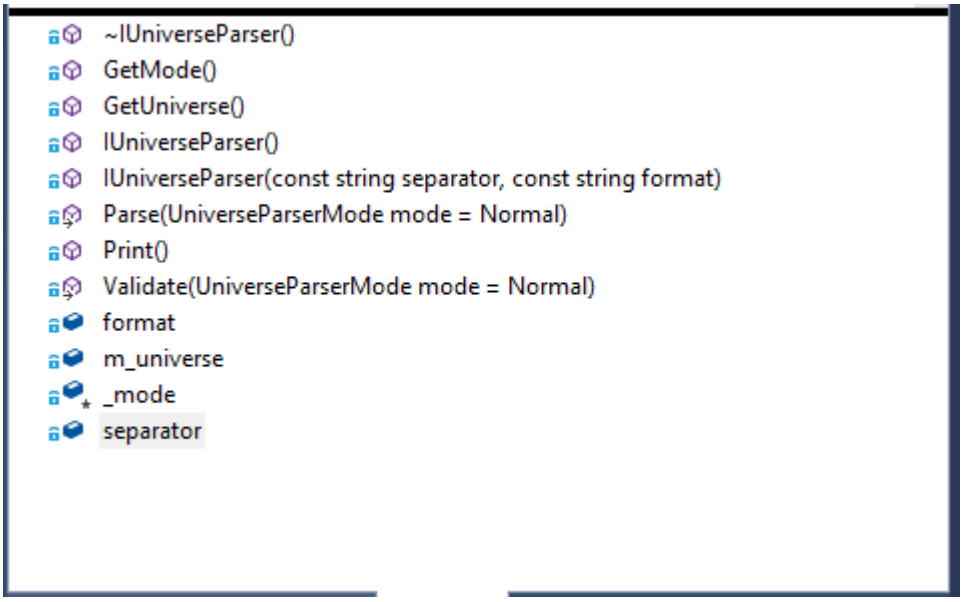
Η διεπαφή *IUniverserParser* είναι στην ουσία μια αφηρημένη τάξη που παρέχει τις βασικές μεθόδους και μεταβλητές για την ανάγνωση και την μετατροπή αρχείων. Ο λόγος που η τάξη αυτή είναι αφηρημένη, είναι για να υπάρχει ομοιόμορφος τρόπος ανάμεσα σε διαφορετικές υλοποιήσεις που αφορούν την ίδια λειτουργικότητα. Δηλαδή σε περίπτωση που υπάρξει η ανάγκη για αλλαγή του τρόπου με τον οποίο «φορτώνεται» το αρχείο, όπως πχ ανάγνωση από την μνήμη, ή από xml αρχείο ή βάση δεδομένων, να μην χρειάζεται μετατροπή και αλλαγή σε όλο το πρόγραμμα. Σε αυτές τις περιπτώσεις το μόνο που απαιτείται είναι η υλοποίηση των διαφορετικών τάξεων κληρονομώντας την αφηρημένη αυτή τάξη, *IUniverserParser*, και υλοποιώντας τις δύο μεθόδους που υλοποιεί και ο *UniverseFileParser*, την Validate και Parse.

Στην διεπαφή αυτή υπάρχει ένα όρισμα για τις δυο μεθόδους που πρέπει να υλοποιήσουν όλες οι τάξεις που την κληρονομούν, το *UniverseParserMode*, που ορίζει τον τρόπο με τον οποίο θα λειτουργήσουν αυτές οι μέθοδοι. Το *UniverseParserMode* είναι μια απαρίθμηση, enum, και έχει δυο τιμές, το Test και το Normal. Η πρώτη τιμή χρησιμοποιείται για να τρέξουν οι μέθοδοι και γενικότερα οι λειτουργίες της IUniverserParser και όσες την κληρονομούν, με διαγνωστικό τρόπο δηλαδή χωρίς να έχουν αποθηκεύσει στην ουσία τα αποτελέσματα της κάθε λειτουργίας. Με αυτό τον τρόπο μπορεί να γίνει εύκολη η διάγνωση και ο εντοπισμός σφαλμάτων που οφείλονται σε λανθασμένη δομή της δοθείσας τοπολογίας. Με την δεύτερη τιμή η τάξη λειτουργεί κανονικά και αποθηκεύει στην μεταβλητή m_universe την τοπολογία που διάβασε. Η τάξη δίνει την δυνατότητα στον προγραμματιστή να δει την τρέχουσα κατάσταση ή αλλιώς την τρέχουσα τιμή της *UniverseParserMode* που έχει δοθεί σαν όρισμα καλώντας την GetMode μέθοδο.

Επίσης διατίθεται και η μέθοδος Print που εκτυπώνει στην κονσόλα τα περιεχόμενα της δομής που κρατάει την τοπολογία, δηλαδή τα περιεχόμενα της μεταβλητή m_universe. Η τάξη έχει για προκαθορισμένο κατασκευαστή τον default που δεν χρειάζεται παραμέτρους αλλά διαθέτει και έναν επιπλέον που παίρνει για ορίσματα το αλφαριθμητικό που χρειάζεται για να ξεχωρίζει τους αριθμούς των κόμβων στο αρχείο καθώς και τον πρότυπο που αντιστοιχεί στην αναπαράσταση του κάθε αριθμού του κόμβου. Αν δεν οριστούν οι δύο αυτές παράμετροι

μπορούν να οριστούν δημόσια από τις μεταβλητές format και separator. Τέλος η τάξη περιλαμβάνει και την μέθοδο GetUniverse που επιστρέφει όλους τους κόμβους που υπάρχουν στην τοπολογία.

Διάγραμμα με το αποτύπωμα της τάξης *IUniverserParser* φαίνεται στο σχήμα 6.



Σχήμα 6, Η τάξη IUniverseParser

Σχήμα 7, Οι τιμές της απαρίθμησης UniverseParserMode

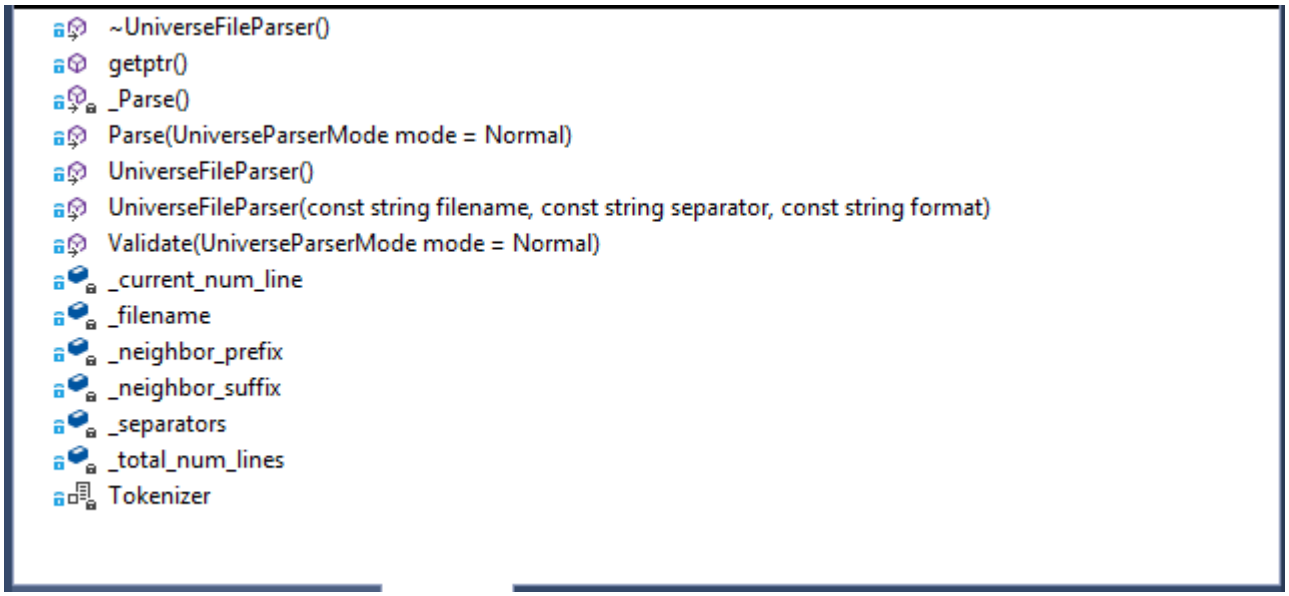
Η υλοποίηση λοιπόν της διεπαφής αυτής είναι η τάξη *UniverseFileParser*. Ουσιαστικά ο *UniverseFileParser* επεκτείνει την τάξη *IUniverserParser* ώστε να λειτουργεί με αρχεία. Και για να γίνει αυτό χρειάζεται ένα ακόμα όρισμα στον κατασκευαστή, το όνομα του αρχείου. Έτσι λοιπόν ο κατασκευαστής αυτής της τάξης, ο προκαθορισμένος, παίρνει τρεις παραμέτρους, το αλφαριθμητικό για το μονοπάτι που βρίσκεται το αρχείο μαζί με το όνομά του, καθώς και το αλφαριθμητικό για το διαχωριστικό με το αλφαριθμητικό του πρότυπου για το όνομα του κόμβου.

Εσωτερικά η τάξη αυτή περιέχει και διάφορες μεταβλητές όπως το σύνολο το γραμμών που βρέθηκαν, την τελευταία γραμμή που διαβάστηκε, κ.α., ως βοηθήματα στις λειτουργίες της. Συγκεκριμένα η τάξη αυτή

χρησιμοποιεί της βιβλιοθήκη *boost/filesystem.hpp*, *boost/iostreams/filtering_stream.hpp*, *boost/tokenizer.hpp* και *boost/lexical_cast.hpp* ώστε να διαχειριστεί αρχεία και να διασπάσει τα περιεχόμενα που διαβάζει.

Στην μέθοδο Parse λοιπόν ανοίγει το αρχείο με βάση το αλφαριθμητικό που έχει δοθεί στον κατασκευαστή της τάξης και εφόσον καταφέρνει να το ανοίξει για ανάγνωση, διαβάζει μια προς μια τις γραμμές που βρίσκει, σπάει τα περιεχόμενα της γραμμής με βάση το αλφαριθμητικό του διαχωριστικού και σε κάθε διάστημα που έχει τεμαχίσει εξάγει την τιμή που βρίσκει με βάση το πρότυπο που είχε οριστεί στον κατασκευαστή και αποθηκεύει στην δομή, m_universe, τις τιμές που βρίσκει ως κόμβους. Η πρώτη τιμή που θα συναντήσει χρησιμοποιείται σαν κλειδί στην δομή και οι υπόλοιπες σαν γειτονικοί κόμβοι. Στην συγκεκριμένη υλοποίηση θεωρούμε ότι όλοι οι κόμβοι στην τοπολογία μας αναγνωρίζονται μοναδικά από ένα θετικό ακέραιο οπότε διπλοεγγραφές στο αρχείο αγνοούνται. Μετά το τέλος της μεθόδου αυτής αν η τιμή που έχει δοθεί σαν *UniverseParserMode* είναι Normal αποθηκεύει τα περιεχόμενα στην δομή αλλιώς τα διαγράφει.

Η μέθοδος Validate στην συγκεκριμένη τάξη ελέγχει αν υπάρχει το αρχείο και αν έχει κανονική δομή και μπορεί να ανοιχτεί για ανάγνωση, αλλιώς πετάει εξαίρεση με το αντίστοιχο σφάλμα.



Σχήμα 8, Η τάξη UniverseFileParser

UniverseHolder

Η τάξη *UniverseHolder* είναι η τάξη που είναι υπεύθυνη για την διαχείριση του γράφου που αντιστοιχεί στην τοπολογία του δικτύου. Η τάξη είναι έχει λοιπόν την δυνατότητα να προσθέσει και να αφαιρέσει κόμβους από το δίκτυο και αντίστοιχα να προσθέσει και να αφαιρέσει γειτονικούς κόμβους από κάθε κόμβο.

Πιο συγκεκριμένα, η τοπολογία του δικτύου μετατρέπεται σε ένα γράφο. Ο γράφος αυτός είναι μη – κατευθυνόμενος, δηλαδή κάθε κορυφή, αν ενώνεται με κάποια ακμή με μια άλλη κορυφή, μπορεί να επικοινωνήσει αμφίδρομα. Έτσι κάθε κορυφή μπορεί να στείλει σε αυτή που συνδέεται, και αντίστοιχα να λάβει. Αυτό έχει σαν αποτέλεσμα για χάρη ευκολίας αντί να έχουμε δυο κατευθυνόμενες ακμές για κάθε ζευγάρι κορυφών να έχουμε μια και να χρησιμοποιείται αυτή για οποιαδήποτε επικοινωνία αφορά τα δύο άκρα.

Ο γράφος στην περίπτωσή μας είναι όπως όλοι οι μη – κατευθυνόμενοι γράφοι με την διαφορά ότι για κορυφές έχουμε τους κόμβους του δικτύου, για ακμές τις επικοινωνιακές γραμμές μεταξύ των γειτονικών κόμβων και

για ρίζα του έχουμε τον κόμβο του εξυπηρετητή. Έτσι λοιπόν ο γράφος είναι στην ουσία ένα δέντρο που ξεκινάει από τον εξυπηρετητή και έχει για φύλλα / παιδιά του τους υπόλοιπους κόμβους του δικτύου.

Η *UniverserHolder* με την βοήθεια των βιβλιοθηκών της Boost, *boost/graph/graph_traits.hpp, boost/graph/adjacency_list.hpp* και *boost/graph/dijkstra_shortest_paths.hpp* διαχειρίζεται τις ακμές και τις κορυφές και εκτυπώνει την τοπολογία που έχει εκείνη τη στιγμή στην κονσόλα. Ειδικότερα, η τάξη έχει ένα κατασκευαστή, τον προκαθορισμένο, με όρισμα ένα αντικείμενο τύπου *UniverseFileParser*. Το όρισμα αυτό το χρησιμοποιεί για να εκτελέσει αρχικά το Validate και το Parse του αντικειμένου πριν ακόμα αρχικοποιηθεί το αντικείμενο. Αυτό συμβαίνει καθώς δεν υπάρχει νόημα να δημιουργηθεί αντικείμενο τύπου *UniverserHolder* αν δεν έχει κάποια δομή, ή έστω έναν κόμβο για να διαχειριστεί. Πριν το τέλος της δημιουργίας του αντικειμένου της τάξης, καλείται μια ιδιωτική μέθοδος η _InitializeGraph, η οποία χρησιμοποιεί το αποτέλεσμα του Parse για να το μετατρέψει σε γράφο και να είναι ευκολότερη η διαχείριση. Τα αποτελέσματα του Parse πρέπει πάντα να περιέχουν για την τιμή που θα έχει ο εξυπηρετητής ώστε να μπορέσει να λειτουργήσει το σύστημα. Αυτή τη στιγμή το σύστημα έχει για αναγνωριστικό του εξυπηρετητή έχει την τιμή 0 η οποία όμως μπορεί να τροποποιηθεί εύκολα, μιας και είναι αποθηκευμένη σε μια καθολική σταθερά.

Επιπλέον η τάξη παρέχει τις μεθόδους AddNode, RemoveNode, RemoveAllNodes και GetNode που διαχειρίζονται τους κόμβους του συστήματος. Οι κόμβοι του συστήματος αναπαρίστανται στο πρόγραμμα με αντικείμενα της τάξης *UniverseNode*. Η τάξη αυτή είναι πολύ απλή και ενθυλακώνει μια ακέραια τιμή τύπου int σαν αναγνωριστικό για να δημιουργούνται και να διαχειρίζονται εντός του προγράμματος οι κόμβοι σαν αντικείμενα ανεξαρτήτου τύπου αναγνωριστικού. Αυτό δίνει την ευελιξία στο πρόγραμμα σε μελλοντική τροποποίηση στον τύπο του αναγνωριστικού να αλλάξει με αρκετή ευκολία. Έτσι η μέθοδος GetNode επιστρέφει ένα αντικείμενο τύπου *UniverseNode* ενώ η AddNode και η RemoveNode να δέχονται για όρισμα αντικείμενα τέτοιου τύπου. Οι δυο αυτές μέθοδοι πραγματοποιούν και ελέγχους κατά την λειτουργία τους ώστε να εμποδίσουν τυχόν σφάλματα όπως να προστεθεί αντικείμενο χωρίς αναγνωριστικό, να προστεθεί αντικείμενο που ήδη υπάρχει, να αφαιρεθεί αντικείμενο που δεν υπάρχει κοκ. Η μέθοδος RemoveAllNodes διαγράφει τα περιεχόμενα του γράφου.

Αξίζει να σημειωθεί ότι η τάξη περιέχει και τις μεθόδους AddEdge και RemoveEdge για την διαχείριση των ακμών μεταξύ των κόμβων. Έτσι η ύπαρξη ακμής μεταξύ των κόμβων υποδεικνύει ότι αυτοί οι κόμβοι μπορούν να επικοινωνήσουν και ότι είναι γειτονικοί. Οι μέθοδοι αυτοί παίρνουν ορίσματα το ζεύγος των κόμβων, δηλαδή δυο αντικείμενα τύπου *UniverseNode*, ώστε να προσδιοριστεί μοναδικά η ακμή μιας και δεν έχει από μόνη της αναγνωριστικό για λόγους απλότητας και έπειτα πραγματοποιούν ελέγχους κατά την λειτουργία τους για αποφυγή σφαλμάτων όπως αν υπάρχουν οι κόμβοι, αν υπάρχει η ακμή κοκ.

Επίσης η *UniverserHolder* διαθέτει την μέθοδο Print για την εκτύπωση στην κονσόλα της δομής του γράφου. Αρχικά εκτυπώνονται οι κόμβοι που περιέχονται και στην συνέχεια για κάθε κόμβο με ποιον άλλο κόμβο υπάρχει ακμή που να τους συνδέει.

Η πιο σημαντική μέθοδος αυτής της τάξης είναι η GetShortestPath που όπως υποδεικνύει και το όνομά της προσπαθεί να βρει το βέλτιστο μονοπάτι ανάμεσα σε δυο κόμβους. Έτσι η μέθοδος αυτή έχει σαν ορίσματα δυο αντικείμενα τύπου *UniverseNode* και πραγματοποιεί αναζήτηση βέλτιστου μονοπατιού με βάση τις ακμές τους. Η αναζήτηση γίνεται φυσικά με την προϋπόθεση ότι οι κόμβοι αυτοί ανήκουν ήδη στον γράφο και ότι η ακμή που δεν έχει κάποιο αναγνωριστικό έχει μόνο σαν γνώρισμα το βάρος, που είναι το κόστος για την χρήση της, και είναι σταθερό και ίσο για όλες τις ακμές και ορίζεται και αυτό παραμετρικά μέσω μιας καθολικής σταθεράς. Για την αναζήτηση λοιπόν χρησιμοποιείται ο αλγόριθμος του Dijkstra [23].

Αλγόριθμος του Dijkstra

Ο αλγόριθμος του Ντάικστρα (Dijkstra) πήρε το όνομά του από τον Ολλανδό Έντσγκερ Ντάικστρα, ο οποίος τον επινόησε το 1956 και τον δημοσίευσε το 1959. Πρόκειται για έναν αλγόριθμο εύρεσης συντομότερων διαδρομών (single-source shortest path problem) από κοινή αφετηρία σε έναν (κατευθυνόμενο ή μη) γράφο με μη αρνητικά βάρη στις ακμές. Ο αλγόριθμος του Dijkstra είναι άπληστος. Δηλαδή, σε κάθε βήμα επιλέγει την τοπικά βέλτιστη λύση, ώσπου στο τελευταίο βήμα συνθέτει μια συνολικά βέλτιστη λύση. Αν ο γράφος περιέχει αρνητικά βάρη, ο αλγόριθμος του Dijkstra δεν δίνει σωστό αποτέλεσμα. Για γράφους που μπορεί να έχουν αρνητικά βάρη στις ακμές, χρησιμοποιούνται πιο περίπλοκοι αλγόριθμοι, όπως αυτός των Bellman και Ford ή των Floyd-Warshall.

Ο αλγόριθμος του Dijkstra είναι πλέον ευρέως διαδεδομένος και χρησιμοποιείται σε πολλές εφαρμογές. Χρήση του αλγόριθμου αυτού κάνει και το πρωτόκολλο OSPF, το οποίο είναι το εσωτερικό πρωτόκολλο πύλης δικτύου του Διαδικτύου.

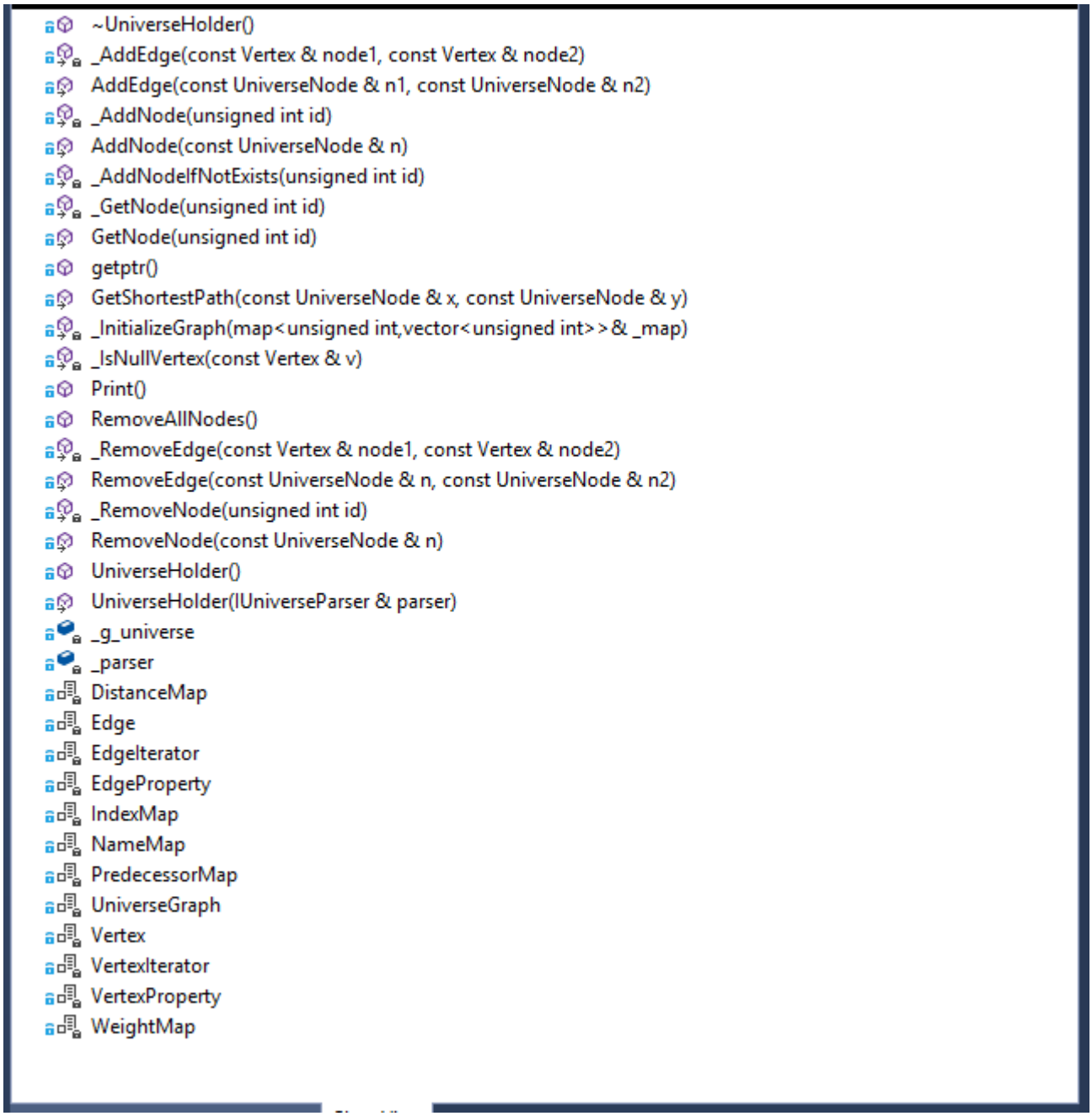
Η λειτουργία του αλγορίθμου είναι η εξής: Εάν έχουμε έναν γράφημα G(V,E), όπου V το σύνολο των κόμβων του και E το σύνολο των ακμών του και μια συνάρτηση βάρους w: E \to \mathbb{R}^+ \cup\{0\} ορισμένη στις ακμές του γράφου. Αυτό σημαίνει ότι για να πάμε από έναν κόμβο του γράφου σε έναν άλλο, θα έχουμε κάποιο κόστος. Ο αλγόριθμος του Dijkstra βρίσκει τα μονοπάτια που πρέπει να ακολουθήσουμε από έναν κόμβο-αφετηρία προς τους υπόλοιπους, ώστε να έχουμε το λιγότερο δυνατό κόστος.

Για τη λειτουργία του αλγόριθμου, σε ένα διάνυσμα d[] μεγέθους |V|=n αποθηκεύουμε την έως τώρα υπολογισμένη απόσταση των κόμβων από την αφετηρία. Κατά την αρχικοποίηση, οι αποστάσεις σημειώνονται d[s]=0 και d[v]=+\infty για κάθε v \not= s, όπου s είναι ο κόμβος-αφετηρία. Επιπλέον ο αλγόριθμος διατηρεί μια ουρά προτεραιότητας Q, για την επεξεργασία των κόμβων του γραφήματος στη σωστή σειρά, και ένα σύνολο S, το σύνολο των κόμβων για τους οποίους ο αλγόριθμος έχει βρει την ελάχιστη διαδρομή. Στην Q εισάγονται όλοι οι κόμβοι του γραφήματος με κλειδί την τιμή d[*], ενώ το σύνολο S είναι αρχικά κενό. Τέλος, ο αλγόριθμος χρησιμοποιεί ένα ακόμη διάνυσμα, το prev[], μεγέθους n, στο οποίο για κάθε κόμβο u αποθηκεύεται ο αμέσως προηγούμενος κόμβος στο ελάχιστο μονοπάτι προς τον u. Για παράδειγμα, έστω ότι το ελάχιστο μονοπάτι από τον s στον b περνά πρώτα από τον a και αμέσως μετά φτάνει στον b. Τότε, prev[b]=a. Αρχικά, κάθε θέση του διανύσματος prev[] λαμβάνει την τιμή null (κενό).

Μετά την αρχικοποίηση των δομών που χρησιμοποιεί, ο αλγόριθμος εξάγει από την Q τον κόμβο x με το ελάχιστο d[*] και τον εισάγει στο σύνολο S. Στο πρώτο βήμα, για παράδειγμα, θα εξάγει τον κόμβο-αφετηρία s, αφού d[s]=0 ενώ όλοι οι υπόλοιποι κόμβοι έχουν άπειρο d[*]. Για κάθε γείτονα y (του x) που δεν ανήκει στο σύνολο S, αν d[y] > d[x] + w(x,y) τότε ενημερώνει το d[y] καταχωρώντας του την τιμή d[x] + w(x,y) και θέτει prev[y]=x. Δηλαδή, αν ο αλγόριθμος υπολογίσει ένα ελαφρύτερο (από το ήδη υπολογισμένο) μονοπάτι για τον κόμβο y, τότε σημειώνει το κόστος του (d[y]) και τον αμέσως προηγούμενο κόμβο του νέου υπολογισμένου μονοπατιού (prev[y]). Με την αλλαγή του d[y] αλλάζει και η θέση του κόμβου y στην ουρά προτεραιότητας Q. Για την ακρίβεια, μεγαλώνει η προτεραιότητα του y, αφού κάθε νέα τιμή του d[y] είναι πάντα μικρότερη από την προηγούμενη. Αφού ο αλγόριθμος εξετάσει όλους τους γείτονες του x που δεν ανήκουν στο σύνολο S, εισάγει στο S τον κόμβο με το ελάχιστο d[*] από όλους όσους δεν ανήκουν στο S. Έπειτα, ο αλγόριθμος επιλέγει πάλι τον πρώτο σε προτεραιότητα κόμβο από την ουρά Q και επαναλαμβάνει τα βήματα αυτά μέχρι να αδειάσει η Q. Όταν πλέον η Q θα έχει αδειάσει, ο αλγόριθμος θα έχει βρει τα ελάχιστα μονοπάτια από τον κόμβο s προς τους όλους τους υπόλοιπους και τα κόστη τους.

Ο αλγόριθμος είναι άπληστος (greedy): σε κάθε βήμα, εξετάζει μόνο τους γειτονικούς ενός κόμβου (τοπικότητα). Βρίσκει τον κόμβο για τον οποίο έχει υπολογίσει την ελάχιστη διαδρομή και τον εισάγει στο σύνολο S, χρησιμοποιώντας πληροφορίες από προηγούμενα βήματα (σύνθεση). Στο τέλος δίνει ένα αποτέλεσμα για όλους τους κόμβους.

Η τάξη *UniverseHolder* λοιπόν χρησιμοποιεί τον αλγόριθμο Dijkstra για να υπολογίσει το βέλτιστο μονοπάτι από τους δύο δοθέντες κόμβους και επιστρέφει το βέλτιστο μονοπάτι σε μορφή πίνακα από αναγνωριστικά. Έτσι η σειρά που επιστρέφεται, είναι και η σειρά των αναγνωριστικών που πρέπει να ακολουθήσει ένα μήνυμα για να φτάσει από τον πρώτο κόμβο που είναι ο κόμβος αφετηρία στον δεύτερο κόμβο που είναι ο κόμβος προορισμού. Τέλος η τάξη περιέχει εσωτερικά δομές για καλύτερη διαχείριση του γράφου όπως ευρετήριο με βάση την αρίθμηση, το αναγνωριστικό των κόμβων κ.α. καθώς και επαναλήπτες για κόμβους και ακμές.



Σχήμα 9, Η τάξη UniverseHolder

SubscriptionHolder

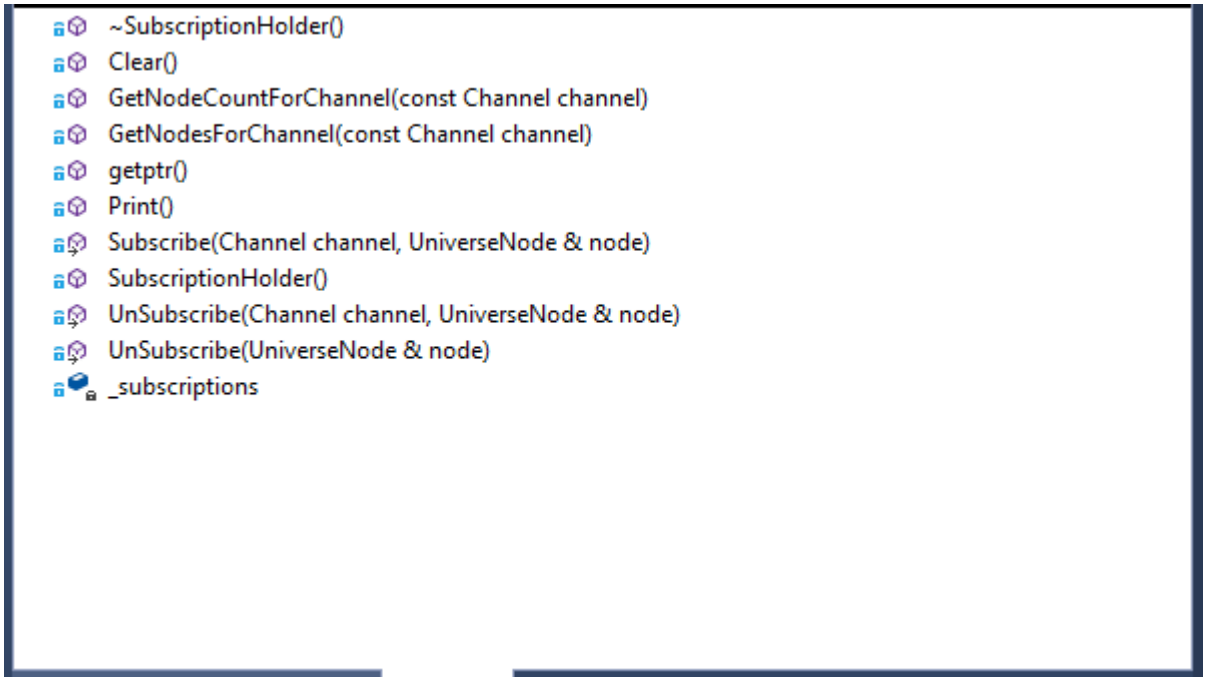
Η τάξη *SubscriptionHolder* είναι η τάξη που είναι υπεύθυνη για την διαχείριση των αιτήσεων από τους κόμβους του δικτύου. Όταν κάποιος κόμβος στείλει ένα αίτημα στον εξυπηρετητή η τάξη *SubscriptionHolder* είναι υπεύθυνη να πραγματοποιήσει την εγγραφή και την διαγραφή του κόμβου σε κάποιο κανάλι. Αυτό έχει ως αποτέλεσμα, εσωτερικά να υπάρχει ένα ευρετήριο που να περιέχει για κλειδί το κανάλι που είναι διαθέσιμο και σαν τιμές ένα σύνολο από κόμβους που αντιστοιχούν στο κανάλι αυτό. Δηλαδή να υπάρχει μια αντιστοίχιση του καναλιού με το σύνολο των κόμβων που είναι εγγεγραμμένοι στο κανάλι.

Η απαρίθμηση *Channel* είναι μια δομή που αντιπροσωπεύει στο σύστημα τα διαθέσιμα κανάλια. Το αναγνωριστικό τους είναι αριθμητικό και υπάρχουν διαθέσιμες 16 διαφορετικές τιμές, καθώς και η τιμή που αντιστοιχεί σε κανένα κανάλι. Οι τιμές της απαρίθμησης φαίνονται στο σχήμα 10.

Σχήμα 10, Οι τιμές της απαρίθμησης Channel

Η *SubscriptionHolder* διαθέτει τις μεθόδους Subscribe και UnSubscribe για την εγγραφή και την διαγραφή ενός κόμβου από ένα κανάλι, για τον λόγο αυτό και τα ορίσματά τους είναι το επιθυμητό κανάλι (η επιθυμητή τιμή της απαρίθμησης) και ο επιθυμητός κόμβος, ο οποίος είναι αντικείμενο τύπου *UniverseNode,* που επιθυμεί την εγγραφή του και απεγγραφή του αντίστοιχα. Φυσικά και στις δυο λειτουργίες πραγματοποιούνται οι λογικοί έλεγχοι για το αν είναι διαθέσιμο το κανάλι ή αν ο κόμβος είναι διαθέσιμος ή αν υπάρχει ήδη εγγραφή του στο κανάλι. Επίσης υπάρχει υπερφόρτωση της μεθόδου UnSubscribe που παίρνει σαν όρισμα το κόμβο και κάνει την διαγραφή του κόμβου από όλα τα κανάλια, έχοντας εξασφαλίσει ότι ο κόμβος υπάρχει στη δομή. Η μέθοδος Clear δίνει την δυνατότητα να διαγραφούν όλες οι τιμές για όλα τα κανάλια ώστε η δομή να είναι αρχικοποιημένη, και η μέθοδος Print τυπώνει στην κονσόλα αρχικά τα διαθέσιμα κανάλια και μετά για κάθε κανάλι τα αναγνωριστικά κάθε κόμβου που είναι εγγεγραμμένος στο αντίστοιχο κανάλι. Τέλος παρέχονται και δυο μέθοδοι για στατιστικούς κυρίους λόγους, όπως πόσους κόμβους εγγεγραμμένους έχει η δομή και πόσους κόμβους έχει ένα κανάλι, καθώς και ποιοι κόμβοι είναι αυτοί, περνώντας σαν όρισμα την τιμή του καναλιού.

Κατά την δημιουργία ενός αντικειμένου της τάξης αυτής, η δομή είναι άδεια και γεμίζει με επιμέρους κλήση των μεθόδων της. Αυτό σημαίνει ότι δεν είναι απαραίτητο όλα τα κανάλια να υπάρχουν σαν κλειδιά στην δομή και ο λόγος είναι για εξοικονόμηση στην μνήμη που χρησιμοποιείται καθώς η χρήση της μπορεί να μεγαλώσει ενδεχομένως αρκετά.



Σχήμα 11, Η τάξη SubscriptionHolder

IMessageHolder

Η τάξη IMessageHolder είναι μια αφηρημένη τάξη που έχει ως στόχο της την λήψη μηνυμάτων από τους κόμβους του δικτύου και την αποστολή μηνυμάτων – απαντήσεων στους κόμβους αυτούς. Έτσι λοιπόν η λειτουργία αυτής της τάξης είναι πολύ σημαντική καθώς αποτελεί όλη την «επιχειρησιακή λογική» του συστήματος. Η λειτουργία της είναι αρκετά απλή αλλά χρειάζεται και τα αντίστοιχα αντικείμενα των προαναφερθέντων τάξεων ώστε να τα «καθοδηγήσει» στο τι πρέπει να κάνουν. Για τον λόγο αυτό η συνεκτικότητα της τάξης αυτής με τις υπόλοιπες είναι αρκετά μεγάλη πράγμα που σημαίνει ότι το API που παρέχουν τα αντικείμενα αυτά, οι μέθοδοι και οι δημόσιες μεταβλητές, πρέπει να είναι σταθερό και να μην αλλάζει, καθώς μια αλλαγή σε κάποιο από αυτά θα επιφέρει και αλλαγή στην τάξη αυτή. Επίσης η τάξη αυτή χρησιμοποιεί τις βιβλιοθήκες *boost/array.hpp*, *boost/bind.hpp* και *boost/asio.hpp* που παρέχει η Boost για την λειτουργικότητα που αφορά το να λάβει μηνύματα από το λειτουργικό σύστημα.

Η τάξη αυτή λοιπόν αποτελείται από ένα κατασκευαστή που παίρνει σαν παραμέτρους ένα αντικείμενο τύπου *io_service*, την τιμή για την πόρτα που θα ακούει από το λειτουργικό σύστημα, ένα αντικείμενο τύπου *UniverseFileParser*, ένα αντικείμενο τύπου *UniverseHolder* και ένα αντικείμενο τύπου *SubscriptionHolder*. Το αντικείμενο τύπου *io_service* ανήκει στην βιβλιοθήκη *boost/asio.hpp* και είναι υπεύθυνο για τη πραγματοποίηση ασύγχρονων λειτουργειών. Όταν μια ασύγχρονη λειτουργία είναι ολοκληρωθεί, χρησιμοποιεί ένα από τα νήματα του *io_service* που τρέχουν για να επιστρέψει το αποτέλεσμα. Αν δεν υπάρχει κάποιο τέτοιο νήμα που χρησιμοποιεί το δικό του εσωτερικό νήμα για την επιστροφή.

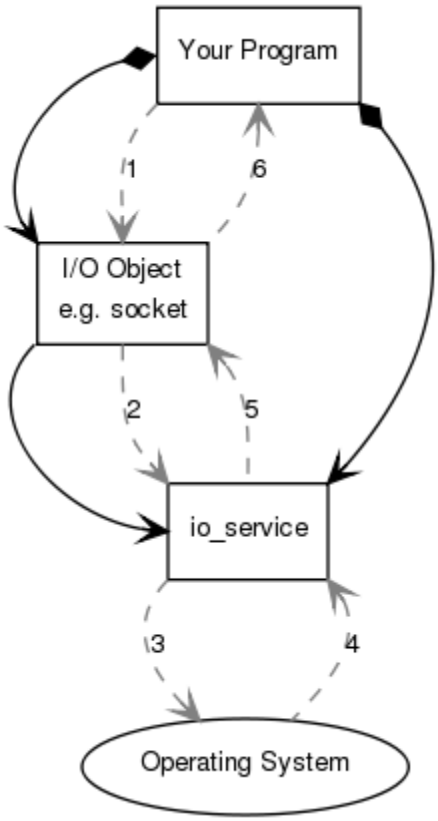
Βασική ανατομία του Boost.Asio

Η βιβλιοθήκη *Boost.Asio* είναι μια φορητή βιβλιοθήκη για προγραμματισμό δικτυακών και άλλων χαμηλού επιπέδου εισόδου – εξόδου λειτουργιών και περιλαμβάνει μετρητές, υποδοχείς (socket), ανάλυση ονομάτων υπολογιστών, σειριακές θύρες κ.α. Η βιβλιοθήκη αυτή λοιπόν μπορεί να χρησιμοποιηθεί και για την πραγματοποίηση τόσο για σύγχρονες όσο και για ασύγχρονες λειτουργίες πάνω σε αντικείμενα εισόδου – εξόδου (IO), όπως οι υποδοχείς. Παρακάτω παρέχεται μια εννοιολογική εικόνα πως λειτουργούν διάφορα αντικείμενα της βιβλιοθήκης μαζί με κάποιο πρόγραμμα.

Σαν αρχικό παράδειγμα είναι το τι ακριβώς γίνεται όταν ένα πρόγραμμα πραγματοποιήσει μια λειτουργία σύνδεσης σε έναν υποδοχέα.

Στην περίπτωση που χρησιμοποιούνται σύγχρονες λειτουργίες, πρέπει το πρόγραμμα (Your program, στο σχήμα) να έχει τουλάχιστον μια αναφορά για ένα αντικείμενο τύπου *io_service*. Το io_service αναπαριστά την σύνδεση του προγράμματος με τις υπηρεσίες εισόδου - εξόδου του λειτουργικού συστήματος.

Σχήμα 12, Διάγραμμα ροής σύγχρονων λειτουργιών της Boost.Asio



Για την πραγματοποίηση λειτουργιών εισόδου – εξόδου το πρόγραμμα θα χρειαστεί ένα αντικείμενο εισόδου – εξόδου (I/O object) όπως έναν υποδοχέα TCP:

```
boost::asio::ip::tcp::socket socket(io_service);
```

Όταν μια σύγχρονη λειτουργία σύνδεσης πραγματοποιείται η ακόλουθη σειρά ενεργειών πραγματοποιείται :

1. Το πρόγραμμα ξεκινά την λειτουργία της σύνδεσης καλώντας το αντικείμενο εισόδου - εξόδου:

```
socket.connect(server_endpoint);
```

2. Το αντικείμενο εισόδου - εξόδου προωθεί το αίτημα στο αντικείμενο io_service.
3. Το αντικείμενο io_service κάνει κλήση στο λειτουργικό σύστημα (operating system στο σχήμα) για την πραγματοποίηση της λειτουργίας σύνδεσης.
4. Το λειτουργικό σύστημα επιστρέφει το αποτέλεσμα της διαδικασίας στο αντικείμενο io_service.
5. Το αντικείμενο io_service μετατρέπει τυχόν λάθη που μπορεί να έχουν προκύψει κατά την διαδικασία σε ένα αντικείμενο τύπου `boost::system::error_code`. Το αντικείμενο `error_code` μπορεί να συγκριθεί με συγκεκριμένες τιμές για να διευκρινιστεί το λάθος που προέκυψε καθώς και με την τιμή `false` αν δεν έχει συμβεί κάποιο λάθος. Έπειτα το αποτέλεσμα προωθείτε πίσω στο αντικείμενο εισόδου – εξόδου.
6. Το αντικείμενο εισόδου - εξόδου πετάει μια εξαίρεση τύπου `boost::system::system_error` αν η διαδικασία απέτυχε, αλλιώς επιστρέφει απλά το αποτέλεσμα.

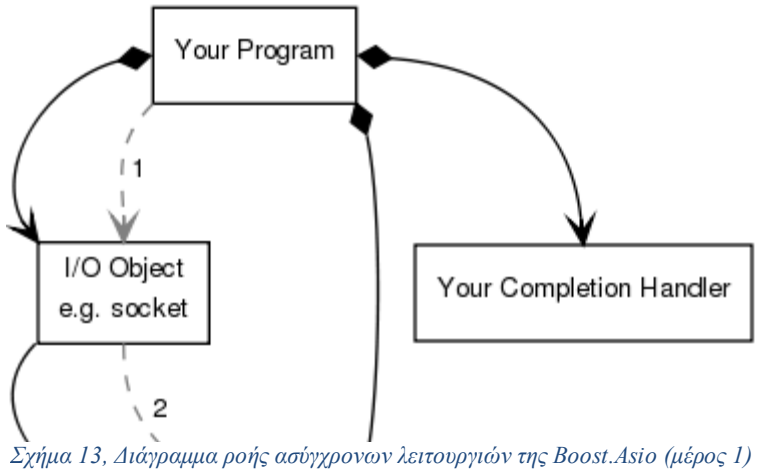
Στην περίπτωση που χρησιμοποιούνται ασύγχρονες λειτουργίες, η ακολουθία γεγονότων είναι διαφορετική.

1. Το πρόγραμμα ξεκινά την λειτουργία της σύνδεσης καλώντας το αντικείμενο εισόδου – εξόδου:

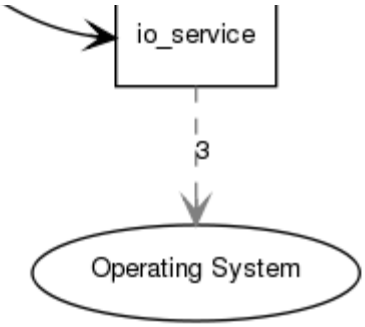
```
socket.async_connect(server_endpoint, your_completion_handler);
```

όπου το `your_completion_handler` είναι μια μέθοδος ή αντικείμενο αναπαράστασης μεθόδου με την παρακάτω υπογραφή:

```
void your_completion_handler(const boost::system::error_code& ec);
```



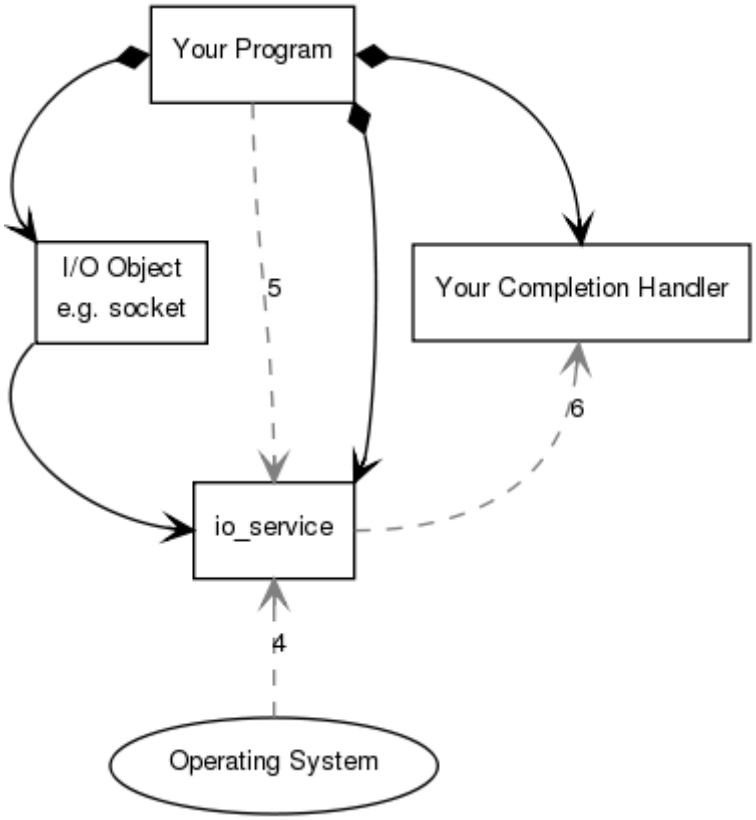
Σχήμα 13, Διάγραμμα ροής ασύγχρονων λειτουργιών της Boost.Asio (μέρος 1)



Η ακριβής υπογραφή της μεθόδου εξαρτάται από την ασύγχρονη λειτουργία που πραγματοποιείται.

2. Το αντικείμενο εισόδου - εξόδου προωθεί το αίτημα στο αντικείμενο io_service.
3. Το αντικείμενο io_service κάνει κλήση στο λειτουργικό σύστημα για να ξεκινήσει την ασύγχρονη λειτουργία σύνδεσης.

Ο χρόνος κυλάει, στην περίπτωση της σύγχρονης λειτουργίας η αναμονή θα ήταν η συνολική διάρκεια για την ολοκλήρωση της λειτουργίας σύνδεσης, και μόλις ολοκληρώνεται η λειτουργία της σύνδεσης συνεχίζεται η διαδικασία ως εξής:

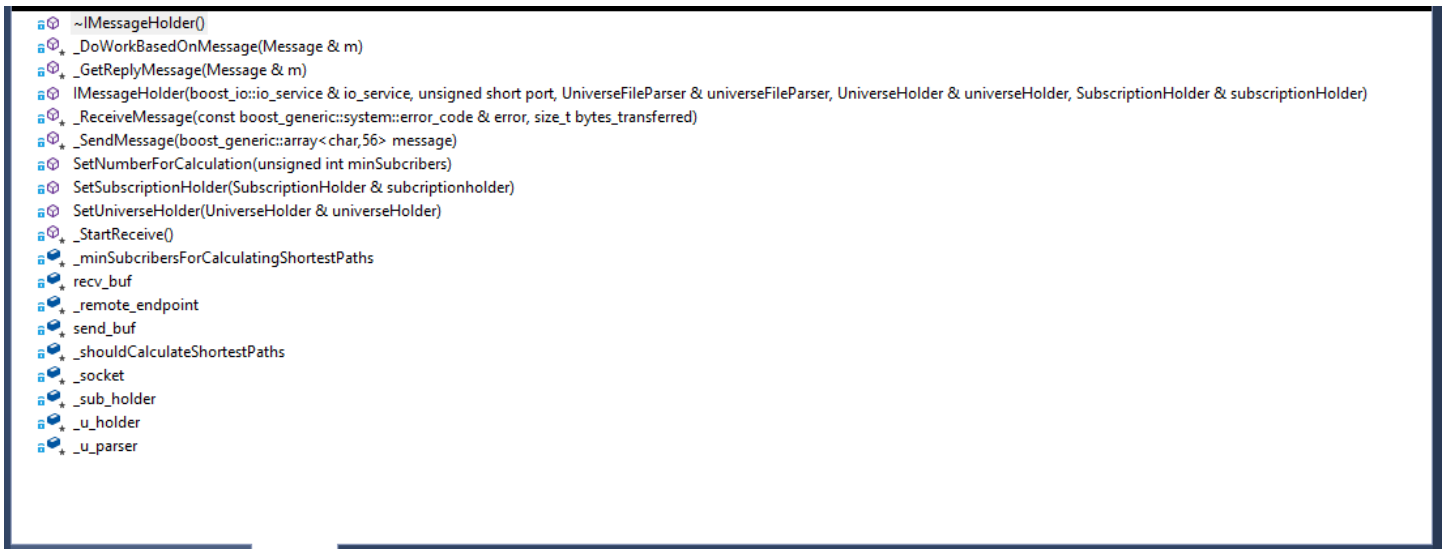


Σχήμα 14, Διάγραμμα ροής ασύγχρονων λειτουργιών της Boost.Asio (μέρος 2)

4. Το λειτουργικό σύστημα με την ολοκλήρωση της διαδικασίας τοποθετεί το αποτέλεσμα σε μια ουρά [24]για να το παραλάβει το αντικείμενο io_service.
5. Το πρόγραμμα πρέπει να κάνει κλήση στη μέθοδο `io_service::run()` (ή σε κάποια παρόμοια μέθοδο) για να παραλάβει το αντικείμενο io_service το αποτέλεσμα. Η κλήση της μεθόδου `io_service::run()` προκαλεί αποκλεισμό όταν υπάρχουν εν λειτουργία ασύγχρονες διαδικασίες γι' αυτό και συνίσταται η κλήση να γίνεται μόλις έχει αρχίσει η πρώτη ασύγχρονη διαδικασία.
6. Κατά την διάρκεια της κλήσης της `io_service::run()`, το αντικείμενο io_service παραλαμβάνει το αποτέλεσμα της διαδικασίας από την ουρά και το μετατρέπει σε ένα αντικείμενο τύπου `error_code`, και το προωθεί στην συνάρτηση επιστροφής που δηλώθηκε στην αρχή.

Η τάξη IMessageHolder εκτός από το αντικείμενο io_service χρησιμοποιεί και το αντικείμενο *UniverseFileParser* για την αρχικοποίηση του αντικειμένου *UniverseHolder*, το οποίο το χρησιμοποιεί για να πραγματοποιήσει εισαγωγή και διαγραφή κόμβων, καθώς και το αντικείμενο *SubscriptionHolder* για την διαχείριση των εγγραφών και απεγγραφών των κόμβων. Τα αντικείμενα αυτά τα αποκτά η τάξη αυτή είτε μέσω του κατασκευαστή ως ορίσματα, είτε μέσω των μεθόδων SetUniverseHolder και SetSubscriptionHolder. Επίσης η τάξη αυτή διαθέτει μερικές μεταβλητές που χρησιμοποιούνται για τις λειτουργίες εισόδου – εξόδου όπως η recv_buf, send_buf, _remote_endpoint κτλ.

Τέλος η IMessageHolder διαθέτει κάποιες εικονικές εσωτερικές μεθόδους που πραγματοποιούν την «ουσιαστική» λειτουργία της τάξης όπως η _DoWorkBasedOnMessage. Η μέθοδος αυτή έχει σαν όρισμα ένα αντικείμενο τύπου *Message* και ανάλογα με το τι τύπου μήνυμα είναι εκτελεί ενέργειες. Η _GetReplyMessage είναι μια μέθοδος σαν την _DoWorkBasedOnMessage που έχει ως όρισμα ένα αντικείμενο τύπου *Message* και επιστρέφει ένα άλλο αντικείμενο τύπου *Message* το οποίο είναι το μήνυμα που θα σταλεί ως απάντηση με βάση το μήνυμα που εστάλη. Επίσης διατίθεται και η μέθοδος _StartReceive που πραγματοποιεί την αρχικοποίηση της λειτουργίας αποδοχής μηνυμάτων. Η _ReceiveMessage είναι μια μέθοδος που πραγματοποιεί την ασύγχρονη αποδοχή των μηνυμάτων, και η _SendMessage που πραγματοποιεί την ασύγχρονη αποστολή των μηνυμάτων. Ο λόγος που είναι εικονικές οι μέθοδοι αυτοί είναι για να μπορεί η κάθε τάξη που κληρονομεί την IMessageHolder να υλοποιήσει δικιές της μεθόδους που να ανταποκρίνονται περισσότερο στις ανάγκες της.



Σχήμα 15, Η τάξη IMessageHolder

Message

Η τάξη Message είναι η τάξη που αντιπροσωπεύει τα μηνύματα στο πρόγραμμα και είναι η δομή δεδομένων που ενσωματώνεται στα μηνύματα UDP που ανταλλάσσονται στο δίκτυο.

Η τάξη αποτελείται λοιπόν από έναν θετικό ακέραιο που υποδηλώνει το αναγνωριστικό του κόμβου, από την απαρίθμηση *MessageAction* που αντιστοιχεί στην ενέργεια που επιθυμεί ο κόμβος, από την απαρίθμηση *Channel* που υποδεικνύει η ενέργεια ποιο κανάλι αφορά, από ένα σταθερό πεδίο 20 χαρακτήρων που αφορά την χρονική σήμανση του μηνύματος και ένα σταθερό πεδίο 24 χαρακτήρων που αφορά επιπλέον επιλογές ή πληροφορίες που αφορούν το μήνυμα αυτό. Οι διαθέσιμες τιμές της απαρίθμησης *MessageAction* παρουσιάζονται στο σχήμα 16.

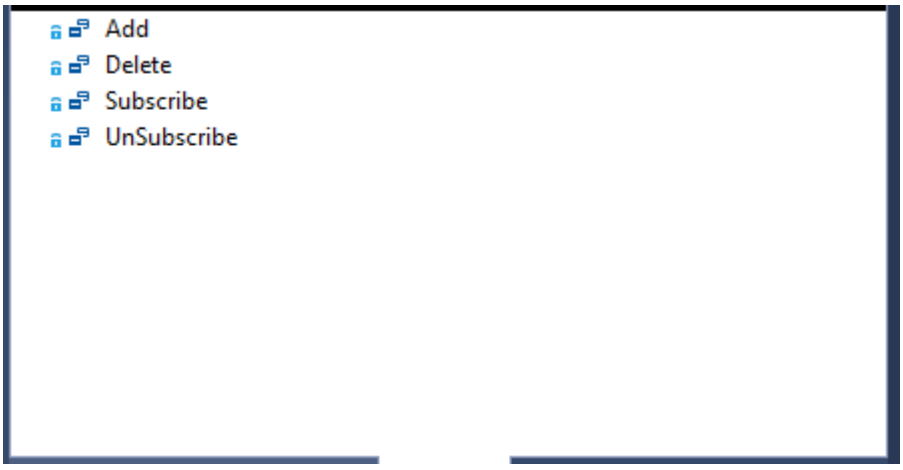
Κάθε αντικείμενο της τάξης αυτής για να θεωρείται έγκυρο πρέπει υποχρεωτικά να περιέχει τιμές στο πεδίο για το αναγνωριστικό του κόμβου, NodeId, και στο πεδίο για την χρονική σήμανση του μηνύματος. Ο λόγος που

πρέπει να είναι αυτά τα δύο πεδία συμπληρωμένα, είναι αρχικά για να είναι δυνατή η αναγνώριση του κόμβου που αποστέλλει το αίτημα και κατά δεύτερον να αναγνωρίζεται μοναδικά το κάθε μήνυμα. Αυτό έχει σαν αποτέλεσμα να υπάρχει η δυνατότητα να αναγνωρίζονται τυχόν διπλότυπα μηνύματα που μπορεί να ληφθούν λόγω λάθους του δικτύου και να απορρίπτεται το όμοιο μήνυμα εφόσον έχει εξυπηρετηθεί το πρώτο μήνυμα. Επίσης υπάρχει η δυνατότητα με βάση το αναγνωριστικό του κόμβου να μαζεύονται στατιστικά και πληροφορίες σχετικά με το ποια μηνύματα έχει αποστείλει ο κάθε κόμβος και κατά συνέπεια τις επιθυμητές κινήσεις που θέλει να πραγματοποιήσει ο κόμβος.

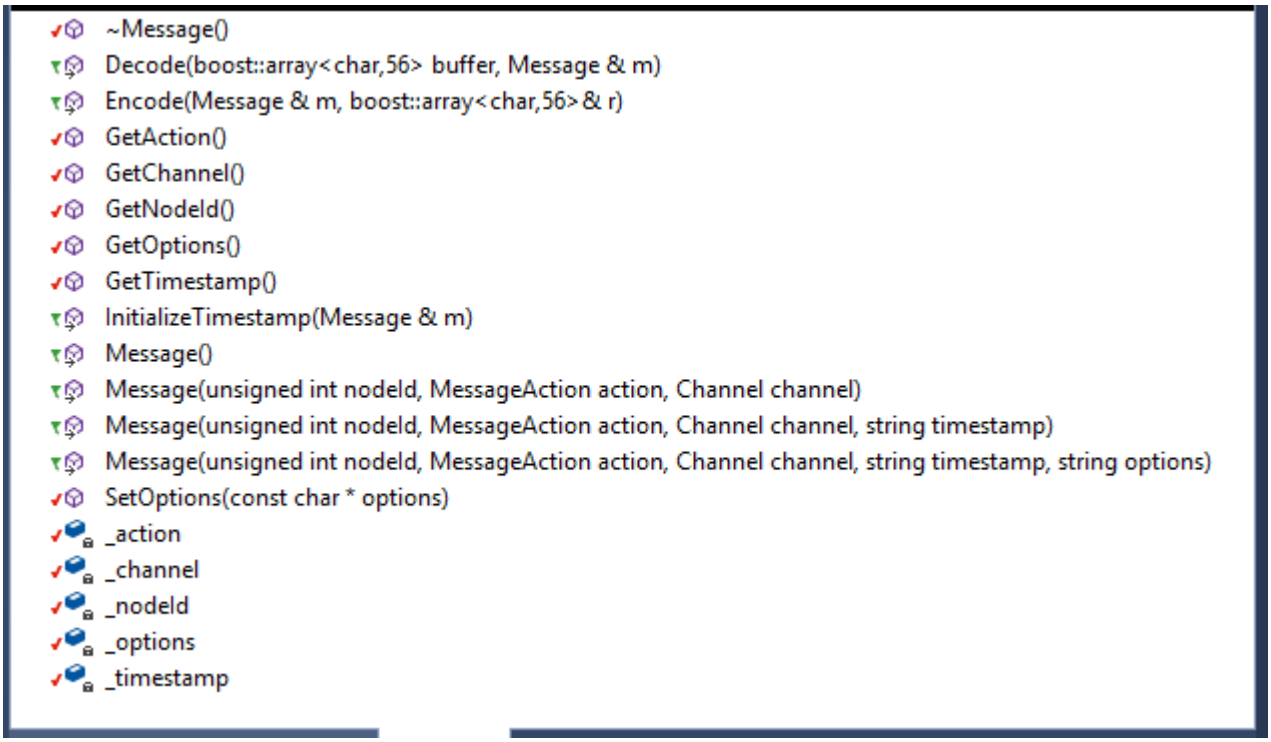
Η τάξη παρέχει μεθόδους για την προσπέλαση των πεδίων του μηνύματος όπως GetNodeId, που επιστρέφει το αναγνωριστικό του κόμβου, το GetAction, που επιστρέφει την τιμή της απαρίθμησης MessageAction που έχει το αντικείμενο, το GetChannel, που επιστρέφει την τιμή της απαρίθμησης Channel που έχει το αντικείμενο, το GetTimestamp που επιστρέφει το αλφαριθμητικό που αντιστοιχεί στην ώρα δημιουργίας του αντικειμένου και το GetOptions, που επιστρέφει το αλφαριθμητικό με τις επιπλέον επιλογές / πληροφορίες που περιέχει το αντικείμενο. Διαθέτει και την μέθοδο SetOptions για την ξεχωριστή εκχώρηση του πεδίου των επιλογών με όρισμα το αλφαριθμητικό που περιέχει τις επιλογές αυτές.

Επίσης η τάξη παρέχει διάφορους κατασκευαστές για την ευκολότερη δημιουργία αντικειμένων. Αρχικά παρέχει ένα προκαθορισμένο κατασκευαστή για την αρχικοποίηση του αντικειμένου με όλες τις τιμές να είναι κενές. Έπειτα παρέχει ένα κατασκευαστή που θέτει τα τρία πιο βασικά πεδία του αντικειμένου με τιμές που παρέχονται από τα ορίσματα και είναι ο ακέραιος αριθμός που αντιστοιχεί στο αναγνωριστικό του κόμβου, η τιμή της απαρίθμησης MessageAction για το προσδιορισμό του είδους του μηνύματος και η τιμή της απαρίθμησης Channel που προσδιορίζει το επιθυμητό κανάλι. Οι υπόλοιποι δυο κατασκευαστές είναι υπερφορτώσεις του κατασκευαστή με τα τρία ορίσματα που αναφέρθηκε προηγουμένως και θέτουν τιμές στα υπόλοιπα πεδία, timestamp και επιλογές που είναι το αλφαριθμητικό της δημιουργίας του μηνύματος και ταυτόχρονα αναγνωριστικό του μηνύματος και το αλφαριθμητικό των επιλογών αντίστοιχα.

Τέλος παρέχονται και οι στατικές μέθοδοι Encode και Decode που μετατρέπουν το αντικείμενο σε μια σειρά από bytes και αντίστροφα και χρησιμοποιούνται για την αποστολή και λήψη μηνυμάτων μέσω του δικτύου καθώς τα πακέτα που λαμβάνονται είναι σειρές από bytes και όχι αντικείμενα κωδικοποιημένα. Η τάξη παρέχει και την στατική μέθοδο InitializeTimestamp που στο αντικείμενο που περνιέται για όρισμα αρχικοποιεί το πεδίο timestamp με την αλφαριθμητική αναπαράσταση της καθολικής ώρας σε μορφή συνολικών tick. Η δομή της τάξης Message απεικονίζεται στο σχήμα 16.



Σχήμα 16, Η απαρίθμηση MessageAction



Σχήμα 17, Η τάξη Message

Ένα αντικείμενο Message μπορεί να έχει τόσες πιθανές μορφές όσες και οι τιμές της απαρίθμησης MessageAction. Δηλαδή οι τύποι των μηνυμάτων είναι οι εξής:

1. Μήνυμα προσθήκης στο δίκτυο

Το μήνυμα προσθήκης στο δίκτυο χρησιμοποιείται από τους κόμβους για την προσθήκη του κόμβου που το αποστέλλει, στην υπάρχουσα τοπολογία. Έτσι είναι δυνατή η αλλαγή της τοπολογίας δυναμικά μέσω μιας τέτοιας μορφής μηνυμάτων. Όταν χρησιμοποιείται αυτό το είδος μηνύματος, είναι υποχρεωτική και η συμπλήρωση του πεδίου NodeId με την τιμή του κόμβου που το αποστέλλει, το πεδίο Action πρέπει να έχει τιμή απαρίθμησης MessageAction, Add και στο πεδίο Timestamp την ώρα που δημιουργήθηκε το μήνυμα. Το πεδίο επιλογές, Options, μπορεί να περιέχει τα αναγνωριστικά των γειτονικών κόμβων του κόμβου που αποστέλλει το μήνυμα, ακολουθούμενα το ένα το άλλο με κόμμα, ώστε να προστεθεί ο κόμβος κατάλληλα στην τοπολογία. Στην περίπτωση που δεν συμπληρωθούν τιμές στο πεδίο επιλογές, υπονοείται ότι ο κόμβος είναι άμεσα συνδεδεμένος με τον εξυπηρετητή. Το πεδίο Channel στην περίπτωση αυτής της μορφής μηνύματος αγνοείται και γι' αυτό μπορεί να πάρει οποιεσδήποτε τιμή ή την προκαθορισμένη τιμή.

Αναγνωριστικό κόμβου	**Add**	None	Χρονική σήμανση	(Γειτονικοί κόμβοι)

Σχήμα 18, Μήνυμα προσθήκης στο δίκτυο

2. Μήνυμα αποχώρησης από το δίκτυο

Το μήνυμα αποχώρησης από το δίκτυο χρησιμοποιείται από τους κόμβους για την αποχώρηση του κόμβου που το αποστέλλει, από την υπάρχουσα τοπολογία. Δηλαδή ο κόμβος που το αποστέλλει επιθυμεί την διαγραφή του από την τοπολογία και από τα διαθέσιμα κανάλια στα οποία ήταν εγγεγραμμένος. Όταν χρησιμοποιείται αυτό το είδος μηνύματος, είναι υποχρεωτική και η συμπλήρωση του πεδίου NodeId με την τιμή του κόμβου που το αποστέλλει, το πεδίο Action πρέπει να έχει τιμή απαρίθμησης MessageAction, Delete και στο πεδίο Timestamp την ώρα που δημιουργήθηκε το μήνυμα. Τα πεδία Channel και Options αγνοούνται και γι' αυτό μπορούν να έχουν οποιεσδήποτε τιμές ή τις προκαθορισμένες τιμές.

Αναγνωριστικό κόμβου	**Delete**	None	Χρονική σήμανση	

Σχήμα 19, Μήνυμα αποχώρησης από το δίκτυο

3. Μήνυμα εγγραφής σε κανάλι

Το μήνυμα εγγραφής σε κανάλι χρησιμοποιείται από τους κόμβους για την εγγραφή του κόμβου που το αποστέλλει, σε κάποιο από τα διαθέσιμα κανάλια. Μιας και τα κανάλια είναι προκαθορισμένα εξαρχής, μιας και ο εξυπηρετητής μπορεί να υποστηρίξει μόνο εγγραφές στα κανάλια που διαθέτει, τα αναγνωριστικά τους είναι γνωστά εκ των προτέρων και ανήκουν στις τιμές της απαρίθμησης Channel. Στην παρούσα έκδοση του εξυπηρετητή υποστηρίζεται μόνο μια εγγραφή ανά μήνυμα, πράγμα που σημαίνει ότι εγγραφή για επιπλέον κανάλι πραγματοποιείται με την λήψη και δεύτερου μηνύματος με τιμή στο πεδίο Channel το επιπλέον κανάλι. Έτσι λοιπόν όταν χρησιμοποιείται αυτό το είδος μηνύματος, είναι υποχρεωτική και η συμπλήρωση του πεδίου NodeId με την τιμή του κόμβου που το αποστέλλει, το πεδίο Action πρέπει να έχει τιμή απαρίθμησης MessageAction, Subscribe, το πεδίο Channel να έχει την τιμή της απαρίθμησης Channel που αντιστοιχεί στο επιθυμητό κανάλι και το πεδίο Timestamp να περιέχει την ώρα που δημιουργήθηκε το μήνυμα. Το πεδίο Options αγνοείται και γι' αυτό μπορεί να έχει οποιεσδήποτε τιμή ή την προκαθορισμένη τιμή. Αξίζει να σημειωθεί ότι εγγραφή ενός κόμβου σε ένα κανάλι δεν προϋποθέτει την απεγγραφή του στα υπόλοιπα. Ο λόγος είναι ότι ένας κόμβος μπορεί να λαμβάνει από περισσότερα από ένα κανάλια ταυτόχρονα και απεγγραφή του από κάποιο από τα κανάλια που είναι εγγεγραμμένος απαιτεί ρητή δήλωση μέσω μηνύματος απεγγραφής από το επιθυμητό κανάλι.

Αναγνωριστικό κόμβου	**Subscribe**	Κανάλι	Χρονική σήμανση	

Σχήμα 20, Μήνυμα εγγραφής σε κανάλι

4. Μήνυμα διαγραφής από κανάλι

Το μήνυμα διαγραφής από κανάλι χρησιμοποιείται από τους κόμβους για την διαγραφή του κόμβου που το αποστέλλει, σε κάποιο από τα κανάλια που είναι ήδη εγγεγραμμένος. Σε αυτή τη μορφή μηνύματος ισχύει ότι και στο μήνυμα εγγραφής, τα κανάλια είναι προκαθορισμένα εξαρχής, αφού ο εξυπηρετητής μπορεί να υποστηρίξει μόνο εγγραφές στα κανάλια που διαθέτει, και έτσι τα αναγνωριστικά τους είναι γνωστά εκ των προτέρων και ανήκουν στις τιμές της απαρίθμησης Channel. Όταν χρησιμοποιείται λοιπόν αυτό το είδος μηνύματος, είναι υποχρεωτική η συμπλήρωση του πεδίου NodeId με την τιμή του κόμβου που το αποστέλλει, το πεδίο Action πρέπει να έχει τιμή απαρίθμησης MessageAction, Unsubscribe, το πεδίο Channel να έχει την τιμή της απαρίθμησης Channel που αντιστοιχεί στο επιθυμητό κανάλι για αποχώρηση και το πεδίο Timestamp να περιέχει την ώρα που δημιουργήθηκε το μήνυμα. Το πεδίο Options αγνοείται και γι' αυτό μπορεί να έχει

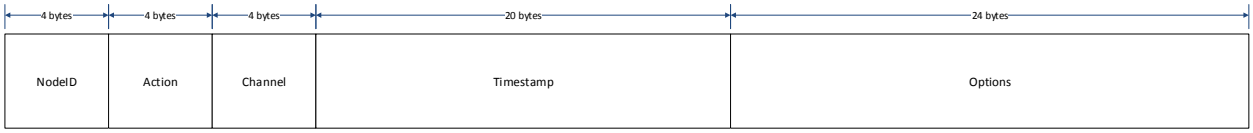
οποιεσδήποτε τιμή ή την προκαθορισμένη τιμή. Αξίζει να σημειωθεί ότι στην περίπτωση που δεν οριστεί κάποιο κανάλι στο πεδίο Channel ο εξυπηρετητής υποθέτει ότι ο κόμβος επιθυμεί την διαγραφή του από όλα τα κανάλια στα οποία είναι εγγεγραμμένος και έτσι θα πραγματοποιηθεί διαγραφή του από όλα τα κανάλια.

Αναγνωριστικό κόμβου	**UnSubscribe**	Κανάλι	Χρονική σήμανση	

Σχήμα 21, Μήνυμα διαγραφής από κανάλι

Πολύ σημαντικό κομμάτι στην επικοινωνία μεταξύ των κόμβων και του εξυπηρετητή αποτελεί και το μέγεθος των δεδομένων που ανταλλάσσονται στα περιεχόμενα των πακέτων καθώς και το συνολικό μέγεθος των πακέτων καθώς στα δίκτυα λόγω ποικιλομορφίας τεχνολογιών και πρωτοκόλλων υπάρχουν διάφοροι περιορισμοί στο πακέτα που ανταλλάσσονται.

Σε ένα τυπικό μηχάνημα x86 η C++ ορίζει ότι ένας ακέραιος θα έχει μέγεθος τουλάχιστον 4 bytes, μια απαρίθμηση μεταφράζεται και αυτή σε ένα ακέραιο μεγέθους τουλάχιστον 4 bytes και ότι κάθε χαρακτήρας αντιστοιχεί σε μέγεθος 1 byte. Άρα σύμφωνα με τα παραπάνω το καθαρό μέγεθος των μηνυμάτων που ανταλλάσσονται στο δίκτυο είναι 56 bytes.



Σχήμα 22, Μέγεθος δεδομένων πακέτου

Ο εξυπηρετητής στην παρούσα έκδοση μεταδίδει μηνύματα πρωτοκόλλου UDP[4] οπότε πρέπει να υπολογιστεί και η επιβάρυνση των κεφαλίδων που επιφέρει το συγκεκριμένο πρωτόκολλο. Προσθέτοντας το μήκος της κεφαλίδας UDP σε ένα σύστημα που ανταλλάσσει δεδομένα μέσω IPv4, με το συνολικό μήκος δεδομένων που μεταφέρεται, υπολογίζεται ότι το σύνολο του πακέτου είναι 76 bytes, αφού είναι 20 bytes για την κεφαλίδα IPv4 του UDP συν τα 56 bytes δεδομένων. Αξίζει να σημειωθεί ότι το UDP επιτρέπει μέχρι 65535 bytes δεδομένων. Σε ένα δίκτυο Ethernet το μέγιστο μέγεθος μετάδοσης, MTU [25], που επιτρέπεται είναι 1500 bytes ενώ μερικοί δρομολογητές δεν μεταφέρουν πακέτα μεγαλύτερα από 512 bytes, με αποτέλεσμα να τα τεμαχίζουν. Στην περίπτωση του προγράμματος δεν υπάρχει κάποια τέτοια περίπτωση μιας και τα δεδομένα που αποστέλλονται είναι πολύ μικρά.

[4] Υπάρχει αναλυτικότερη περιγραφή του πρωτοκόλλου UDP στην σελίδα 43.

Στο πρόγραμμα του εξυπηρετητή υπάρχουν δυο υλοποιήσεις της αφηρημένης τάξης *IMessageHolder*, η τάξη *MessageHolder* και η *MessageHolderForStats*. Ο λόγος που υπάρχουν αυτές οι δυο υλοποιήσεις είναι γιατί διαφέρουν στον τρόπο που εσωτερικά λειτουργούν και όχι στον τρόπο που απαντάνε στο κόμβο που έστειλε το αίτημα. Για τον αιτών ο τρόπος που ανταποκρίνονται οι δυο αυτές υλοποιήσεις είναι ακριβώς ίδιος.

Αναλυτικότερα:

MessageHolder

Η τάξη *MessageHolder* κληρονομεί την αφηρημένη τάξη *IMessageHolder* και επομένως είναι υπεύθυνη για την λήψη και απάντηση των αιτημάτων που λαμβάνει από το δίκτυο. Η τάξη αυτή δεν προσθέτει παραπάνω μεθόδους από αυτές που διαθέτει η *IMessageHolder* αλλά αλλάζει την υλοποίηση των εικονικών μεθόδων προσθέτοντας δικά της στοιχεία.

Αρχικά η *MessageHolder* έχει ένα κατασκευαστή που είναι ίδιος με τον κατασκευαστή που παρέχει η αφηρημένη τάξη και για τον λόγο αυτό τις τιμές που λαμβάνει στα ορίσματα που έχει τις περνάει στον κατασκευαστή της *MessageHolder*. Δηλαδή το αντικείμενου τύπου *io_service*, τον ακέραιο που αφορά την πόρτα, το αντικείμενο τύπου *UniverseFileParser*, το αντικείμενο τύπου *UniverseHolder* και το αντικείμενο τύπου *SubscriptionHolder*. Στο κατασκευαστή λοιπόν τα αντικείμενα των *UniverseFileParser*, *UniverseHolder* και *SubscriptionHolder*, αναθέτονται στις αντίστοιχες μεταβλητές για να χρησιμοποιηθούν στην συνέχεια. Έπειτα αρχικοποιείται ο υποδοχέας κάνοντας χρήση του αντικειμένου *io_service*, και δηλώνοντας την χρήση του πρωτοκόλλου IPv4 για UDP [26] στην πόρτα που δηλώθηκε ως όρισμα. Η πόρτα ορίζεται ως παράμετρος και όχι σαν σταθερά ώστε να μπορεί να αλλαχθεί εύκολα και να λειτουργεί η τάξη ανεξαρτήτως πόρτας. Η τιμή που παίρνει το όρισμα ανήκει σε μια καθολική σταθερά και μοιράζεται σε όλο το πρόγραμμα. Τέλος πριν την επιστροφή του αντικειμένου καλείται η μέθοδος _StartReceive, που έχει ξανά οριστεί στην τάξη αυτή, και ανοίγει ένα ασύγχρονο νήμα για την παραλαβή των πακέτων από το δίκτυο, δηλώνοντας ένα buffer μεγέθους 56 byte για κάθε πακέτο που λαμβάνεται από τον υποδοχέα και ορίζοντας ως συνάρτηση που θα καλεστεί όταν ληφθεί το αποτέλεσμα την _ReceiveMessage. Ο τρόπος λειτουργίας της μεθόδου αυτής είναι συνυφασμένος με τον τρόπο λειτουργίας της *Boost.Asio* που αναλύθηκε σε προηγούμενη ενότητα.

Πρωτόκολλο UDP

Πριν την συνέχεια της ανάλυσης της τάξης *MessageHolder*, αξίζει να γίνει μια αναφορά στο πρωτόκολλο UDP.

Το πρωτόκολλο User Datagram Protocol (UDP) είναι ένα από τα βασικά πρωτόκολλα που χρησιμοποιούνται στο διαδίκτυο. Μία εναλλακτική ονομασία του πρωτοκόλλου είναι Universal Datagram Protocol. Διάφορα προγράμματα χρησιμοποιούν το πρωτόκολλο UDP για την αποστολή σύντομων μηνυμάτων (γνωστών και ως datagrams) από τον έναν υπολογιστή στον άλλον μέσα σε ένα δίκτυο υπολογιστών.

Ένα από τα κύρια χαρακτηριστικά του UDP είναι ότι δεν εγγυάται αξιόπιστη επικοινωνία. Τα πακέτα UDP που αποστέλλονται από έναν υπολογιστή μπορεί να φτάσουν στον παραλήπτη με λάθος σειρά, διπλά ή να μην φτάσουν καθόλου εάν το δίκτυο έχει μεγάλο φόρτο. Αντιθέτως, το πρωτόκολλο TCP διαθέτει όλους τους απαραίτητους μηχανισμούς ελέγχου και επιβολής της αξιοπιστίας και συνεπώς μπορεί να εγγυηθεί την αξιόπιστη επικοινωνία μεταξύ των υπολογιστών. Η έλλειψη των μηχανισμών αυτών από το πρωτόκολλο UDP το καθιστά αρκετά πιο γρήγορο και αποτελεσματικό, τουλάχιστον για τις εφαρμογές εκείνες που δεν απαιτούν αξιόπιστη επικοινωνία.

Οι εφαρμογές audio και video streaming χρησιμοποιούν κατά κόρον πακέτα UDP. Για τις εφαρμογές αυτές είναι πολύ σημαντικό τα πακέτα να παραδοθούν στον παραλήπτη σε σύντομο χρονικό διάστημα ούτως ώστε να μην υπάρχει διακοπή στην ροή του ήχου ή της εικόνας. Κατά συνέπεια προτιμάται το πρωτόκολλο UDP διότι είναι αρκετά γρήγορο, παρόλο που υπάρχει η πιθανότητα μερικά πακέτα UDP να χαθούν. Στην περίπτωση που χαθεί κάποιο πακέτο, οι εφαρμογές αυτές διαθέτουν ειδικούς μηχανισμούς διόρθωσης και παρεμβολής ούτως ώστε ο τελικός χρήστης να μην παρατηρεί καμία αλλοίωση ή διακοπή στην ροή του ήχου και της εικόνας λόγω του χαμένου πακέτου. Σε αντίθεση με το πρωτόκολλο TCP, το UDP υποστηρίζει broadcasting, δηλαδή την αποστολή ενός πακέτου σε όλους τους υπολογιστές ενός δικτύου, και multicasting, δηλαδή την αποστολή ενός πακέτου σε κάποιους συγκεκριμένους υπολογιστές ενός δικτύου. Η τελευταία δυνατότητα χρησιμοποιείται πολύ συχνά στις εφαρμογές audio και video streaming ούτως ώστε μία ροή ήχου ή εικόνας να μεταδίδεται ταυτόχρονα σε πολλούς συνδρομητές.

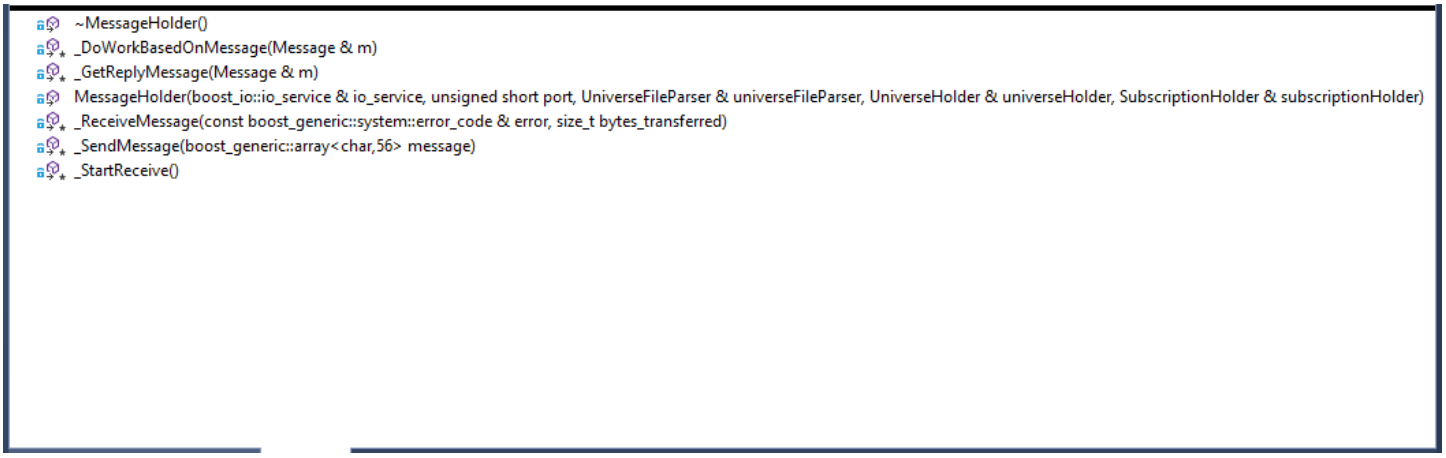
Μερικές σημαντικές εφαρμογές που χρησιμοποιούν πακέτα UDP είναι οι εξής: Domain Name System (DNS), IPTV, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) και τα παιχνίδια που παίζονται ζωντανά μέσω του διαδικτύου.

Με την δημιουργία του αντικείμενου της τάξης MessageHolder ξεκινάει και η διαδικασία αναμονής του νήματος του io_service για λήψη κάποιου πακέτου από το δίκτυο. Μόλις ληφθεί κάποιο πακέτο, το io_service το παραλαμβάνει από την ουρά του λειτουργικού συστήματος, το τοποθετεί στο buffer που ορίσαμε και καλεί την συνάρτηση που ορίσαμε σαν συνάρτηση επιστροφής. Στην συγκεκριμένη υλοποίηση είναι η συνάρτηση _ReceiveMessage που δέχεται για ορίσματα ένα αντικείμενο τύπου *boost::system::error_code*, το οποίο υποδηλώνει αν έχει συμβεί κάποιο σφάλμα στην λήψη του πακέτου ή αν έχει κάποιο σφάλμα το πακέτο, και ένα όρισμα που υποδηλώνει τον αριθμό των bytes που μεταφέρθηκαν. Στην συνάρτηση αυτή λοιπόν, αρχικά ελέγχεται αν υπάρχει κάποιο σφάλμα δικτύου ή κάποιο σφάλμα στα δεδομένα. Στην περίπτωση αυτή απορρίπτεται το πακέτο και συνεχίζει ο εξυπηρετητής να περιμένει για το επόμενο πακέτο που θα λάβει. Στην περίπτωση που το πακέτο είναι σωστό και κανένα σφάλμα δεν έχει συμβεί, η συνάρτηση συνεχίζει με την μετατροπή του μηνύματος από το buffer σε ένα αντικείμενο τύπου *Message*, καλώντας την στατική συνάρτηση Decode. Μετά την μετατροπή εξάγεται το αναγνωριστικό του κόμβου που έστειλε το πακέτο, το timestamp του πακέτου και το messageAction που υποδεικνύει το είδος του αιτήματος. Επόμενο βήμα είναι η κλήση της συνάρτησης _DoWorkBasedOnMessage που ανάλογα το είδος του μηνύματος πραγματοποιεί τις αντίστοιχες ενέργειες. Έπειτα καλείται η _GetReplyMessage που επιστρέφει ένα μήνυμα απάντησης βασισμένο στο μήνυμα που λήφθηκε, το οποίο μήνυμα απάντησης κωδικοποιείται σε bytes και αντιγράφεται στο buffer για αποστολή και καλείται ο υποδοχέας να το στείλει ασύγχρονα στην διεύθυνση IP του αποστολέα. Όταν το πραγματοποιήσει θα καλέσει την συνάρτηση _SendMessage ως συνάρτηση επιστροφής. Τέλος καλείται η συνάρτηση _StartReceive για να επαναληφθεί η ίδια διαδικασία από την αρχή.

Η μέθοδος _DoWorkBasedOnMessage στην συγκεκριμένη υλοποίηση είναι υπεύθυνη για να πραγματοποιήσει τις κατάλληλες ενέργειες ανάλογα με την μορφή του μηνύματος. Αρχικά εξετάζει τι είδους μήνυμα είναι το μήνυμα που περάστηκε σαν όρισμα, ουσιαστικά το μήνυμα που περιείχε το πακέτο. Στην περίπτωση που το μήνυμα είναι μήνυμα εγγραφής, καλείται το αντικείμενο τύπου *SubscriptionHolder*, μέσω της τοπικής μεταβλητής, χρησιμοποιώντας την μέθοδο Subscribe για να πραγματοποιήσει την εγγραφή του κόμβου, σύμφωνα με κανάλι και το αναγνωριστικό του κόμβου που περιέχει το πακέτο. Στην περίπτωση που είναι μήνυμα διαγραφής από κανάλι, καλείται πάλι το αντικείμενο τύπου *SubscriptionHolder*, χρησιμοποιώντας αυτή τη φορά την μέθοδο UnSubscribe για την διαγραφή του αντικειμένου με βάση το κανάλι και το αναγνωριστικό του κόμβου. Αν στο πακέτο δεν έχει οριστεί κάποιο κανάλι τότε καλείται η υπερφορτωμένη μέθοδος UnSubscribe που διαγράφει τον κόμβο από όλα τα κανάλια που έχει εγγραφεί. Στην περίπτωση που το μήνυμα είναι μήνυμα προσθήκης καλείται το αντικείμενο τύπου *UniverseHolder*, μέσω της τοπικής μεταβλητής και καλεί την μέθοδό του, AddNode, για την προσθήκη του κόμβου στην τοπολογία. Επίσης διαβάζονται οι πληροφορίες που περιέχονται στο πεδίο options του μηνύματος για να βρεθούν οι γειτονικοί του

κόμβοι. Έτσι για κάθε γειτονικό κόμβο, εφόσον βρεθεί στην τοπολογία, προστίθεται μια νέα ακμή που τον συνδέει με τον αιτώντα κόμβο, δηλαδή τον κόμβο που έστειλε το πακέτο. Με αυτό τον τρόπο ενημερώνεται και η τοπολογία. Τέλος στην περίπτωση που το μήνυμα είναι τύπου διαγραφής, καλείται αρχικά το αντικείμενο τύπου *SubscriptionHolder*, το οποίο καλεί την μέθοδο του UnSubscribe για να τον διαγράψει από όλα τα κανάλια, και έπειτα το αντικείμενο τύπου *UniverseHolder* για να τον διαγράψει από την τοπολογία, διαγράφοντας όλες τις ακμές των κόμβων που συνδέονται με τον αιτώντα. Έτσι λοιπόν, ανάλογα με το τύπο του μηνύματος πραγματοποιούνται και οι κατάλληλες ενέργειες.

Η μέθοδος _GetReplyMessage χρησιμοποιείται από την τάξη MessageHolder για την δημιουργία μηνύματος απάντησης με βάση το μήνυμα που λήφθηκε. Στην μέθοδο αυτή αντιγράφεται το μήνυμα που λήφθηκε, αλλάζει το αναγνωριστικό του κόμβου που το έστειλε με το αναγνωριστικό του εξυπηρετητή και διατηρείται ίδιο το αναγνωριστικό του μηνύματος, timestamp, ώστε ο αποστολέας να αναγνωρίσει για ποιο αίτημά του πήρε απάντηση. Το αναγνωριστικό του εξυπηρετητή είναι μια καθολική σταθερά που δεν αλλάζει σε όλη την διάρκεια εκτέλεσης του προγράμματος και είναι εκ των προτέρων γνωστό. Μετά την αντιγραφή υπολογίζεται το συντομότερο μονοπάτι από τον κόμβο που έστειλε το μήνυμα στον κόμβο του εξυπηρετητή μέσω της μεθόδου GetShortestPath, σύμφωνα με το αντικείμενο τύπου *UniverseHolder* που γνωρίζει την τοπολογία. Το αποτέλεσμα το οποίο είναι ένα σύνολο αναγνωριστικών μετατρέπεται σε ένα αλφαριθμητικό με τα αναγνωριστικά των κόμβων να ακολουθεί το ένα το άλλο με κόμμα. Το αλφαριθμητικό αυτό αποθηκεύεται στο μήνυμα απάντησης στο πεδίο options και το μήνυμα αυτό αποστέλλεται σαν απάντηση στον αποστολέα.



Σχήμα 23, Η τάξη MessageHolder

MessageHolderForStats

Η τάξη *MessageHolderForStats* κληρονομεί και αυτή την αφηρημένη τάξη *IMessageHolder* και επομένως είναι άλλη μια υλοποίηση της που είναι υπεύθυνη για την λήψη και απάντηση των αιτημάτων που λαμβάνει από το δίκτυο. Ούτε αυτή η τάξη προσθέτει άλλες μεθόδους παραπάνω από αυτές που διαθέτει η *IMessageHolder* αλλά αλλάζει την υλοποίηση των εικονικών μεθόδων σύμφωνα με τις ανάγκες της.

Συγκεκριμένα η τάξη αυτή παρέχει ένα κατασκευαστή όμοιο με τον κατασκευαστή της *IMessageHolder* με την διαφορά ότι έχει ένα επιπλέον όρισμα, έναν ακέραιο που ορίζει τον ελάχιστο αριθμό εγγεγραμμένων σε ένα κανάλι που χρειάζεται για να υπολογιστούν τα συντομότερα μονοπάτια, τον οποίο ακέραιο τον αποθηκεύει σε μια τοπική μεταβλητή. Στην συνέχεια ο κατασκευαστής έχει την ίδια λειτουργικότητα που έχει και ο κατασκευαστής του *MessageHolder,* δηλαδή αρχικοποιούνται οι μεταβλητές για τα αντικείμενα τύπου *UniverseFileParser, UniverseHolder* και *SubscriptionHolder*, αρχικοποιείται ο υποδοχέας για χρήση του πρωτοκόλλου UDP IPv4 στην πόρτα που του δίνεται από το όρισμα και καλεί την μέθοδο _StartReceive.

Η μέθοδος _StartReceive με την σειρά της δημιουργεί ένα ασύγχρονο νήμα για την λήψη με βάση τον υποδοχέα και θέτει ως μέθοδο επιστροφής την συνάρτηση _ReceiveMessage, όπως ακριβώς πραγματοποιείται και στην τάξη *MessageHolder*.

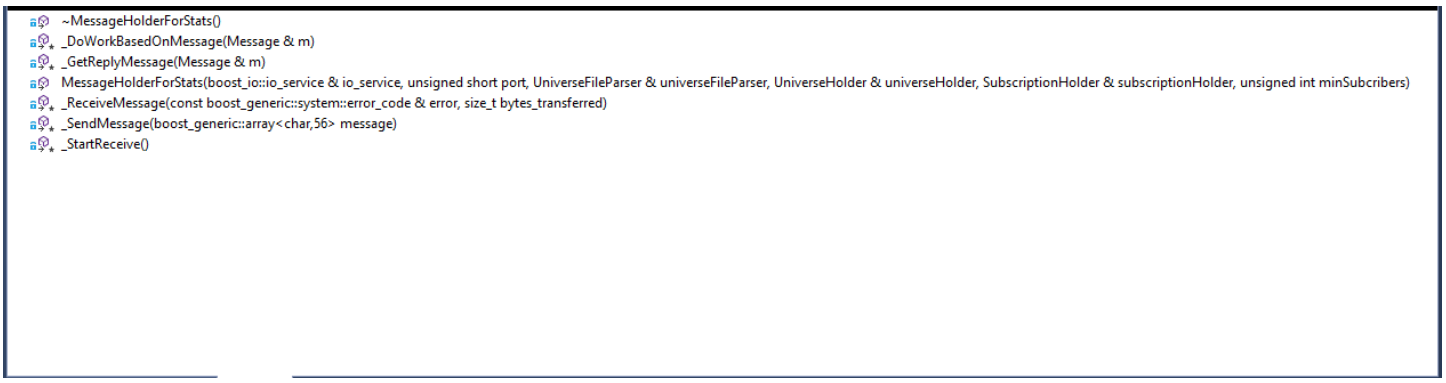
Στην συγκεκριμένη υλοποίηση η συνάρτηση _ReceiveMessage δέχεται για ορίσματα ένα αντικείμενο τύπου *boost::system::error_code* που υποδηλώνει αν έχει συμβεί κάποιο σφάλμα στην λήψη του πακέτου ή αν έχει κάποιο σφάλμα το πακέτο, και ένα όρισμα που υποδηλώνει τον αριθμό των bytes που μεταφέρθηκαν. Όταν εκτελείται η συνάρτηση αυτή γίνεται έλεγχος για το αν υπάρχει κάποιο σφάλμα δικτύου ή κάποιο σφάλμα στα δεδομένα. Αν έχει συμβεί κάτι τέτοιο το πακέτο απορρίπτεται και αναμένει η μέθοδος για κάποιο καινούργιο. Στην περίπτωση που το πακέτο είναι σωστό, τότε το μετατρέπει σε ένα αντικείμενο τύπου *Message* από την ακολουθία των byte του buffer σε ένα αντικείμενο τύπου *Message*, καλώντας την στατική συνάρτηση Decode. Από το αντικείμενο *Message* εξάγεται το αναγνωριστικό του κόμβου που έστειλε το πακέτο, το timestamp του πακέτου και το messageAction που υποδεικνύει το είδος του αιτήματος. Έτσι το αντικείμενο αυτό μεταβιβάζεται στην συνάρτηση _DoWorkBasedOnMessage για την πραγματοποίηση της κατάλληλης ενέργειας. Μετά την εκτέλεση της τελευταίας, καλείται η _GetReplyMessage που επιστρέφει ένα μήνυμα απάντησης βασισμένο στο μήνυμα που λήφθηκε, το οποίο με την σειρά του κωδικοποιείται σε bytes και αντιγράφεται στο buffer για αποστολή. Ο υποδοχέας καλείται να το στείλει ασύγχρονα στην διεύθυνση IP του αποστολέα και να καλέσει την συνάρτηση _SendMessage ως συνάρτηση επιστροφής εφόσον αυτό πραγματοποιηθεί. Τέλος καλείται η συνάρτηση _StartReceive για να επαναλαμβάνει η ίδια διαδικασία από την αρχή.

Μέχρι στιγμής η λειτουργικότητα της *MessageHolderForStats* είναι ίδια με την λειτουργικότητα της *MessageHolder*, σε αυτό που διαφέρουν είναι στις άλλες δυο μεθόδους που κληρονομούν από την *IMessageHolder*, την _DoWorkBasedOnMessage και την _GetReplyMessage. Στην μέθοδο _DoWorkBasedOnMessage της τάξης *MessageHolderForStats*, γίνονται τα εξής: Αρχικά το πακέτο ελέγχεται για το τύπου είναι, αν είναι πακέτο για προσθήκη στην τοπολογία, το αντικείμενο *UniverseHolder* προσθέτει στην τοπολογία τον κόμβο που το απέστειλε καθώς και τις ακμές από τους γειτονικούς του κόμβους προς αυτόν, με βάση το περιεχόμενο του πακέτου στο πεδίο επιλογές. Αν το πακέτο αφορά διαγραφή το αντικείμενο *SubscriptionHolder* καλεί την UnSubscribe του για να διαγράψει τον κόμβο από όλα τα κανάλια, και έπειτα το αντικείμενο *UniverseHolder* διαγράφει όλες τις ακμές των υπόλοιπων κόμβων προς αυτόν ώστε να διαγραφεί και τελικά από την τοπολογία. Στην περίπτωση που το πακέτο είναι για εγγραφή του κόμβου σε κάποιο κανάλι υπολογίζεται ο συνολικός αριθμός των εγγεγραμμένων που υπάρχουν στο επιλεγμένο κανάλι. Αν ο αριθμός αυτός είναι μεγαλύτερος από τον αριθμό που δηλώθηκε στον κατασκευαστή για το ελάχιστο πλήθος των εγγεγραμμένων τότε η τοπική μεταβλητή για το αν πρέπει να υπολογιστούν τα συντομότερα μονοπάτια, _shouldCalculateShortestPaths, παίρνει την τιμή `true`. Αν το πακέτο είναι τύπου διαγραφής από κανάλι,

καλείται πάλι το αντικείμενο τύπου *SubscriptionHolder*, χρησιμοποιώντας αυτή τη φορά την μέθοδο UnSubscribe για την διαγραφή του κόμβου από την λίστα με τους εγγεγραμμένους κόμβους.

Η μέθοδος _GetReplyMessage της τάξης *MessageHolderForStats* επιστρέφει ένα αντικείμενο τύπου *Message* με βάση το αντικείμενο *Message* που λήφθηκε. Αρχικά αντιγράφεται το μήνυμα που λήφθηκε, και αλλάζει το αναγνωριστικό του κόμβου που το έστειλε με το αναγνωριστικό του εξυπηρετητή. Τα υπόλοιπα πεδία του μηνύματος όπως το αναγνωριστικό του, timestamp,το είδος του, MessageAction και το κανάλι, παραμένουν ίδια. Το μόνο που αλλάζει είναι το συντομότερο μονοπάτι ανάμεσα στον αποστολέα και τον εξυπηρετητή καθώς αυτήν την φορά, αν η μεταβλητή _shouldCalculateShortestPaths είναι αληθής, τότε υπολογίζονται όλα τα μονοπάτια από τον εξυπηρετητή για τους κόμβους που είναι εγγεγραμμένοι στο επιθυμητό κανάλι και ο αριθμός τους είναι μεγαλύτερος από το ελάχιστο αριθμό εγγεγραμμένων κόμβων που ορίστηκε στον κατασκευαστή. Δηλαδή αν ο αριθμός που ορίστηκε στον κατασκευαστή είναι 2 και οι εγγεγραμμένοι κόμβοι είναι 4, θα υπολογιστούν 3 μονοπάτια για τους 3 τελευταίους κόμβους. Αν ο αριθμός που ορίστηκε είναι 3 και οι κόμβοι 6 θα υπολογιστούν 4 μονοπάτια κοκ. Έτσι μπορούμε να μαζέψουμε στατιστικά για το πόσο χρόνο χρειάζεται για να υπολογίσουμε συντομότερα μονοπάτια για k κόμβους.

Όπως και στην τάξη *MessageHolder*, το αναγνωριστικό του εξυπηρετητή είναι μια καθολική σταθερά που δεν αλλάζει σε όλη την διάρκεια εκτέλεσης του προγράμματος και είναι εκ των προτέρων γνωστό. Το αποτέλεσμα του συντομότερου μονοπατιού από τον αιτούντα στον εξυπηρετητή αποθηκεύεται στο μήνυμα απάντησης στο



Σχήμα 24, Η τάξη MessageHolderForStats

πεδίο επιλογές και το μήνυμα αυτό αποστέλλεται σαν απάντηση στον αποστολέα.

Έτσι λοιπόν έχουμε δυο διαφορετικές υλοποιήσεις για την ίδια βασική λειτουργικότητα, την λήψη και την απάντηση στα μηνύματα των υπόλοιπων κόμβων. Η μια είναι αυτή που παρέχει η τάξη *MessageHolder* και η άλλη αυτή που παρέχει η *MessageHolderForStats*. Ενώ είναι παρόμοιες λειτουργικότητες, η πρώτη υπολογίζει ένα μονοπάτι την φορά και το αποστέλλει στο μήνυμα απάντησης, ενώ η δεύτερη υπολογίζει μονοπάτια κατά περίπτωση και συνήθως περισσότερα από ένα την φορά για ένα αίτημα. Έτσι η δεύτερη χρησιμοποιείται για την εύρεση της χρονικής επιβάρυνσης που υπάρχει λόγω των πολλαπλών υπολογισμών για συντομότερα μονοπάτια για διαφορετικούς κόμβους, με βάση παραμέτρους. Αν στην τάξη *MessageHolderForStats* το όρισμα στον κατασκευαστή είναι 0, δηλαδή ο ελάχιστος αριθμός εγγεγραμμένων για τον υπολογισμό μονοπατιών τότε η τάξη αυτή θα συμπεριφερόταν όπως και η *MessageHolder*.

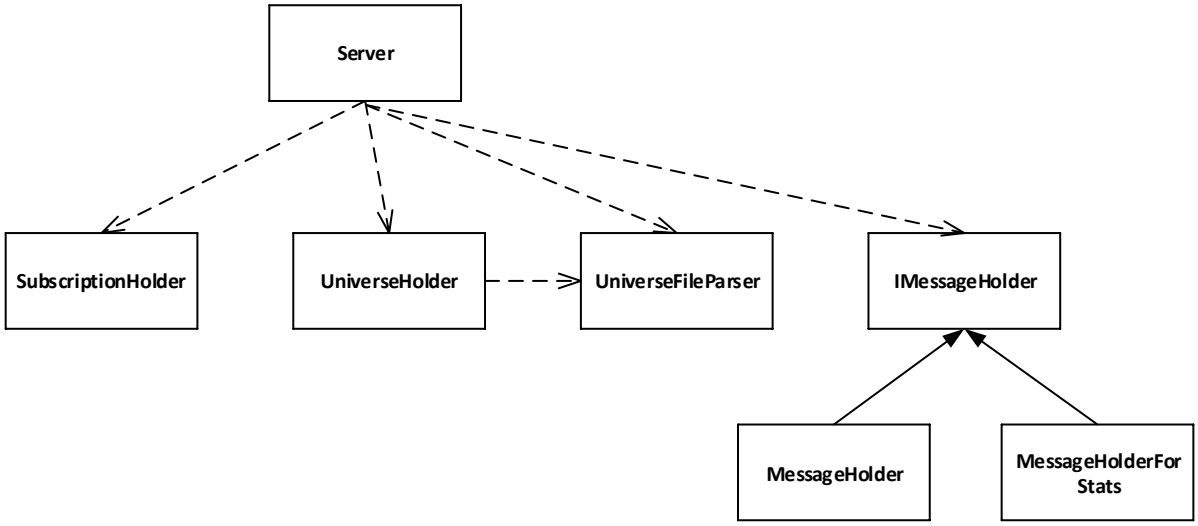
Συνοψίζοντας η δομή του εξυπηρετητή αποτελείται από τις 4 βασικές αυτές τάξεις:

- Την τάξη UniverseFileParser,
- Την τάξη UniverseHolder,
- Την τάξη SubscriptionHolder,

- Και μια υλοποίηση από τις δυο που διατίθενται και κληρονομούν την IMessageHolder, είτε την MessageHolderForStats είτε την MessageHolder.

Η ευκολία που παρέχεται μέσω του αντικειμενοστρεφούς προγραμματισμού να κρύβεται η λειτουργικότητα ενός εξυπηρετητή σε ένα αντικείμενο χωρίς να είναι γνωστή η εσωτερική λειτουργία δίνει την δυνατότητα για εύκολη διαχείριση αλλά και επαναχρησιμοποίηση του καθώς για την εξυπηρέτηση πολλαπλών κόμβων δεν χρειάζεται τίποτα παραπάνω από μερικά όμοια αντικείμενα. Έτσι η επεκτασιμότητα του προγράμματος μπορεί να αυξηθεί σημαντικά.

Διαγραμματικά οι διάφορες συνιστώσες - συστατικά της τάξης server παρουσιάζονται στο σχήμα 25.



Σχήμα 25, Η δομή του εξυπηρετητή

Δομή πελάτη

Στα πλαίσια της πειραματικής αποτίμησης δημιουργήθηκε και μια τάξη για την δημιουργία αντικειμένων που θα πραγματοποιούν αιτήματα προς τον εξυπηρετητή με σκοπό να προσομοιώσουν κόμβους του δικτύου που στέλνουν μηνύματα για προσθήκη στην τοπολογία, διαγραφή, εγγραφή σε κάποιο κανάλι και απεγγραφή από κάποιο άλλο.

Client

Η τάξη *Client* είναι η τάξη υπεύθυνη για την προσομοίωση και αποστολή αιτημάτων στον εξυπηρετητή. Για την πραγματοποίηση της αποστολής και λήψης μηνυμάτων από το δίκτυο, ενσωματώνει τις εξωτερικές βιβλιοθήκες της Boost, *boost/array.hpp*, *boost/asio.hpp*, *boost/thread/thread.hpp*, και *boost/date_time/posix_time/posix_time.hpp*. Αρχικά η τάξη αποτελείται από ένα κατασκευαστή ο οποίος έχει σαν ορίσματα ένα αντικείμενο τύπου *boost::asio::io_service*, για να διαχειρίζεται τις λειτουργίες εισόδου – εξόδου από το λειτουργικό σύστημα, ένα αλφαριθμητικό που δηλώνει την διεύθυνση του εξυπηρετητή και ένα ακέραιο που είναι η πόρτα του λειτουργικού συστήματος από την οποία λαμβάνει ο εξυπηρετητής. Έτσι υπάρχει η δυνατότητα να οριστεί προτεραιότητα στις διαδικασίες εισόδου – εξόδου και να διαχειριστεί η λειτουργία αποστολής και λήψης για όλα τα αντικείμενα ομοιογενώς λόγω του κοινού αντικειμένου io_service. Δηλαδή η έναρξη, η παύση και η διακοπή των λειτουργιών εισόδου – εξόδου, κοινώς η αποστολή και λήψη πακέτων, μπορεί να ρυθμιστεί μέσω αυτού του αντικειμένου και έτσι όλα πακέτα μπορούν να ξεκινήσουν, να σταματήσουν κ.ο.κ ταυτόχρονα. Στον κατασκευαστή λοιπόν, δημιουργείται ένα αντικείμενο τύπου *boost::asio::ip::udp::resolver::query* με όρισμα το πρωτόκολλο, που θα χρησιμοποιηθεί για την αποστολή και λήψη, την διεύθυνση και την πόρτα του εξυπηρετητή. Το αντικείμενο αυτό θα περασθεί σαν όρισμα στη μέθοδο resolve ενός αντικειμένου τύπου *boost::asio::ip::udp::resolver*, που αρχικοποιήθηκε με βάση το αντικείμενο io_service, έτσι ώστε να βρει την πραγματική διεύθυνση IP του εξυπηρετητή. Η λειτουργία αυτή πραγματοποιείται μέσω του πρωτοκόλλου DNS. Η δυνατότητα ανάλυσης της ονομασίας του εξυπηρετητή μπορεί να χρησιμοποιηθεί για την δυναμική εύρεση του εξυπηρετητή πριν την έναρξη της αποστολής καθώς και στο να χρησιμοποιηθεί και η λειτουργία των CDN ώστε να προωθούν τα αιτήματα στους κοντινότερους, από πλευρά τοπολογίας, εξυπηρετητές. Τέλος η λειτουργικότητα του κατασκευαστή ολοκληρώνεται με την ανάθεση της πραγματικής διεύθυνσης IP του εξυπηρετητή για χρήσης της σε όλη την διάρκεια ζωής του αντικειμένου καθώς και με το άνοιγμα του υποδοχέα για αποστολή με χρήση πρωτοκόλλου IPv4. Ο κατασκευαστής αυτός είναι και ο μοναδικός για την τάξη Client.

Όπως και στην τάξη *Server* έτσι και στην *Client* παρέχεται η δυνατότητα για διαχείριση του αντικειμένου μέσω των μεθόδων Run, Stop, Pause και Reset. Οι μέθοδοι αυτές αλλάζουν τις τιμές των εσωτερικών μεταβλητών _run και _counter που χρησιμοποιούνται ως συνθήκες για την αποστολή ή όχι πακέτων. Η μεταβλητή _counter χρησιμοποιείται επίσης και για το μέτρηση των πακέτων που έχουν αποσταλεί από την τελευταία φορά που πραγματοποιήθηκε επαναφορά, και μπορεί να προσπελαστεί η τιμή της μέσω της μεθόδου GetCounter που διαθέτει η τάξη. Επίσης η τάξη Run έχει όρισμα ένα ακέραιο που υποδηλώνει την μέγιστη τιμή που μπορεί να πάρει η μεταβλητή _counter. Έτσι η μέθοδος αυτή έχει ένα βρόχο επανάληψης όπου εφόσον η τιμή της μεταβλητής _run είναι `true` και η τιμή της μεταβλητής _counter είναι μικρότερη από την τιμή που τέθηκε σαν όρισμα πραγματοποιεί την αποστολή και την λήψη μηνύματος καλώντας τις μεθόδους _SendMessage και _ReceiveMessage.

Επίσης η τάξη παρέχει και μερικές μεθόδους για παραμετροποίηση του αντικειμένου. Για την ακρίβεια, παρέχεται η μέθοδος SetNodeId που παίρνει σαν όρισμα τον ακέραιο που αντιστοιχεί στο αναγνωριστικό που θα χρησιμοποιεί το αντικείμενο για την αποστολή των μηνυμάτων του. Έτσι είναι δυνατόν ο ορισμός κάθε αντικειμένου με το αναγνωριστικό που του αντιστοιχεί ξεχωριστά, και η χρήση του ίδιου αναγνωριστικού για πολλά αντικείμενα στην περίπτωση που χρειάζεται να εξεταστεί ή να προσομοιωθεί η ικανότητα του εξυπηρετητή για διαχείριση μεγάλου όγκου αιτημάτων. Η μέθοδος GetNodeId χρησιμοποιείται αντίστοιχα για την ανάκτηση της τιμής του αναγνωριστικού που έχει ανατεθεί στο αντικείμενο. Άλλη μια μέθοδος για παραμετροποίηση είναι SetChannel που δέχεται σαν όρισμα μια τιμής της απαρίθμησης *Channel* και αντιστοιχεί στο κανάλι για το οποίο το αντικείμενο θα αιτείται εγγραφή και απεγγραφή. Η SetMessageType είναι η μέθοδος που ορίζει τον τύπο μηνυμάτων τον οποίο θα στέλνει το αντικείμενο. Δηλαδή ένας ακέραιος

περνιέται σαν όρισμα στην μέθοδο και μπορεί να πάρει τιμές από το 0 έως και το 4. Το 0 αντιστοιχεί στην τυχαία επιλογή του τύπου μηνύματος που θα στείλει τελικά το αντικείμενο, 1 για την αίτηση για προσθήκη στην τοπολογία, 2 για την αίτηση διαγραφής από την τοπολογία, 3 για την αίτηση εγγραφής σε κανάλι και 4 για την αίτηση απεγγραφής από κανάλι.

Επιπρόσθετα μια ακόμα μέθοδος παραμετροποίησης είναι και η μέθοδος SetParameters που παρέχεται από την τάξη κα αφορά την γενικότερη παραμετροποίηση του αντικειμένου ασχέτως με το τι αίτημα θα στείλει. Έτσι η μέθοδος αυτή έχει ως όρισμα έναν ακέραιο αριθμό που αντιστοιχεί στην καθυστέρηση σε ms (milliseconds) που θα έχει ο κόμβος ανάμεσα σε κάθε επανάληψη του βρόχου της Run. Δηλαδή την καθυστέρηση ανάμεσα στην μια αποστολή και λήψη μηνύματος και στην επομένη. Αυτός ο ακέραιος δίνει την δυνατότητα να προσομοιώσουμε διάφορες τυχαίες συμπεριφορές κόμβων που πραγματοποιούν αιτήματα προς τον εξυπηρετητή. Ένα ακόμα όρισμα της συνάρτησης είναι και ο ακέραιος που ορίζει τον μέγιστο αριθμό τύπων μηνυμάτων που μπορεί να έχει το αντικείμενο και χρησιμοποιείται στην περίπτωση τυχαίας επιλογής τύπου μηνύματος, μιας και ορίζει τον μέγιστο επιτρεπτό αριθμό. Το ίδιο ισχύει και για το επόμενο όρισμα το οποίο είναι ο ακέραιος που ορίζει τον μέγιστο αριθμό καναλιών που στην περίπτωση τυχαίας επιλογής καναλιών, μπορεί το αντικείμενο να επιλέξει. Το επόμενο όρισμα είναι ο αριθμός των γειτονικών κόμβων που το αντικείμενο θα έχει. Το όρισμα αυτό δίνει την δυνατότητα για ορισμό των πόσων συνδέσεων θα έχει τελικά το αντικείμενο στην τοπολογία, το οποίο χρησιμοποιείται κατά την αίτηση εισαγωγής στην τοπολογία και παίζει ρόλο στον υπολογισμό των συντομότερων μονοπατιών που υπολογίζει ο εξυπηρετητής. Το τελευταίο όρισμα της μεθόδου αυτής είναι ο μέγιστος αριθμός των γειτονικών κόμβων που αποτελεί τον ακέραιο ο οποίος θα είναι ο μέγιστος αριθμός γειτόνων που θα μπορεί ένα αντικείμενο να έχει στην περίπτωση τυχαίας επιλογής γειτόνων. Η μέθοδος αυτή δίνει την δυνατότητα για παραμετροποίηση της τυχαιότητας των μηνυμάτων, των τύπων των μηνυμάτων, των περιεχομένων των μηνυμάτων καθώς και της συχνότητας αποστολής αιτημάτων.

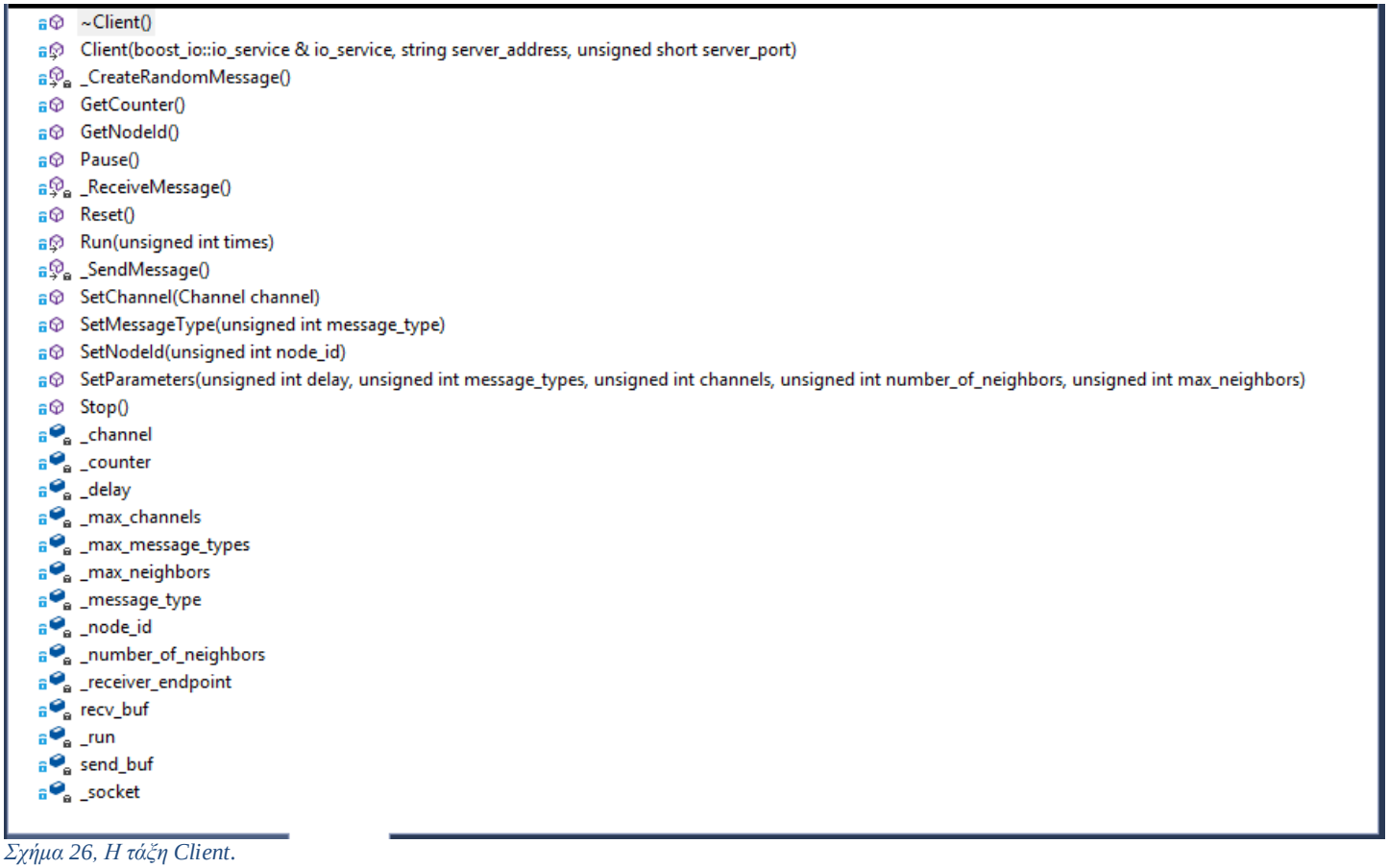
Η τάξη Client παρέχει και τις μεθόδους _SendMessage και _ReceiveMessage. Η _SendMessage είναι η μέθοδος που χρησιμοποιείται για την αποστολή μηνυμάτων προς τον εξυπηρετητή. Αναλυτικότερα η μέθοδος αυτή καλεί την _CreateRandomMessage για την δημιουργία ενός νέου αιτήματος και καλεί τον υποδοχέα που διαθέτει η τάξη για την αποστολή του πακέτου, καλώντας την μέθοδό του send_to και περνώντας σαν όρισμα το πακέτο που έχει μετατραπεί σε μια σειρά από bytes και την διεύθυνση IP του εξυπηρετητή που βρέθηκε στον κατασκευαστή του αντικειμένου. Έπειτα αυξάνεται η τιμή της μεταβλητής _counter για να μετρηθεί ο αριθμός των αιτημάτων που έχουν αποσταλεί. Η _ReceiveMessage είναι η μέθοδος που χρησιμοποιείται για την λήψη των μηνυμάτων από τον εξυπηρετητή ως απάντηση στα μηνύματα που έχουν αποσταλεί. Όπως αναφέρθηκε και πριν, η μέθοδος αυτή καλείται μετά την κλήση της _SendMessage μέσα στην μέθοδο _Run. Η μέθοδος αυτή λοιπόν διαβάζει από τον υποδοχέα της τάξης την σειρά από bytes που εστάλησαν από την IP διεύθυνση του εξυπηρετητή καθώς και το αντικείμενο *boost::system::system_error* που έχει τις πληροφορίες για τυχόν σφάλματος κατά την λήψη του πακέτου ή στα δεδομένα του πακέτου. Στην περίπτωση που το πακέτο περιέχει κάποιο σφάλμα ή κάποιο σφάλμα έχει πραγματοποιηθεί το πακέτο απορρίπτεται και η διαδικασία συνεχίζεται. Στην συνέχεια διαβάζονται τα περιεχόμενα της απάντησης του πακέτου και εκτυπώνονται στην κονσόλα.

Η μέθοδος _CreateRandomMessage είναι η μέθοδος που πραγματοποιεί την πιο σημαντική λειτουργία, από άποψη «επιχειρησιακής λογικής» για την τάξη αυτή, την δημιουργία ενός τυχαίου μηνύματος που θα σταλεί ως αίτημα προς εξυπηρέτηση από το αντικείμενο αυτόν. Για την ακρίβεια αρχικά δημιουργείται ένα αντικείμενο τύπου *Message*, που περιέχει τις προκαθορισμένες τιμές και αρχικοποιείται η συνάρτηση τυχαιότητας, srand[5], με βάση τη δεδομένη χρονική στιγμή, ώστε να μην επαναλαμβάνονται οι αριθμοί που θα προκύπτουν από την τυχαία επιλογή που παρέχει η μέθοδος της rand[6], και να υπάρχει πραγματική τυχαιότητα στις επιλογές, και όχι επαναλαμβανόμενη τυχαιότητα. Έπειτα ελέγχεται ο τύπος της μεταβλητής message_type που έχει πάρει τιμή από την συνάρτηση SetMessageType. Στην περίπτωση που είναι 0, τότε η μεταβλητή παίρνει την τιμή που προκύπτει από την επιλογή της τιμής που δίνει η συνάρτηση rand με βάση τον μέγιστο αριθμό επιτρεπτών τύπων μηνυμάτων που μπορεί να πάρει το αντικείμενο, που είχε δοθεί στο αντικείμενο στην συνάρτηση SetParameters. Αν η νέα τιμή είναι 1 τότε η μέθοδος θα παραμετροποιήσει το αντικείμενο *Message* για να αποτελέσει ένα μήνυμα αίτησης για προσθήκη στην τοπολογία. Έτσι λοιπόν επιλέγεται ένας αριθμός από γειτονικούς κόμβους που ο κόμβος θα έχει, με βάση την επιλογή της συνάρτησης rand πάνω στον μέγιστο αριθμό γειτονικών κόμβων που έχει οριστεί. Με βάση αυτόν τον αριθμό είναι και οι φορές που επιλέγεται

[5] Η συνάρτηση srand αποτελεί μέρος της βιβλιοθήκης STD
[6] Η συνάρτηση rand αποτελεί μέρος της βιβλιοθήκης STD

τυχαία το αναγνωριστικό του κόμβου που θα είναι γειτονικός κόμβος του αντικειμένου. Το αναγνωριστικό του κόμβου που θα αντιστοιχηθεί ως ο νέος γειτονικός κόμβος προκύπτει από την επιλογή της συνάρτησης που rand στον μέγιστο αριθμό αναγνωριστικών που μπορεί να έχει ο κόμβος, ο οποίος αριθμός έχει οριστεί και αυτός στην SetParameters. Έπειτα το αντικείμενο *Messsage* παίρνει τιμές με βάση το αναγνωριστικό του κόμβου που του ανατέθηκε, σαν MessageAction την τιμή της απαρίθμησης για προσθήκη στην τοπολογία και ορίζοντας στο πεδίο επιλογές, την μετατροπή των αναγνωριστικών των γειτονικών κόμβων που επιλέχθηκαν σε αλφαριθμητικό το οποίο θα περιέχει κόμματα ανάμεσα στα αναγνωριστικά. Στην περίπτωση που η τιμή είναι 2 τότε το αντικείμενο *Messsage*, παίρνει τιμές για το αναγνωριστικό του κόμβου που του ανατέθηκε, σαν MessageAction την τιμή της απαρίθμησης για διαγραφή από την τοπολογία και οι υπόλοιπες τιμές κενές. Αν η τιμή της μεταβλητής είναι 3 τότε το μήνυμα που θα σταλεί θα είναι τύπου εγγραφής σε κανάλι. Αν έχει ορισθεί τιμή για το κανάλι για το οποίο θα αιτείται ο κόμβος τότε το αντικείμενο *Messsage*, παίρνει τιμές για το αναγνωριστικό του κόμβου που του ανατέθηκε, σαν MessageAction την τιμή της απαρίθμησης για αίτηση εγγραφής, και την τιμή της απαρίθμησης Channel για την οποία θα γίνει η αίτηση. Αν δεν έχει ορισθεί κάποιο κανάλι ή έχει ορισθεί η τιμή None της απαρίθμησης τότε επιλέγεται το κανάλι που προκύπτει από την επιλογή της rand με βάση τον μέγιστο αριθμό καναλιών που μπορεί να έχει ο κόμβος σύμφωνα με την τιμή που ορίστηκε στο αντικείμενο. Τέλος στην περίπτωση που η τιμή της μεταβλητής είναι 4 τότε το μήνυμα που θα σταλεί θα είναι τύπου απεγγραφής από κανάλι. Εάν έχει ορισθεί τιμή για το κανάλι για το οποίο θα αιτηθεί απεγγραφή ο κόμβος, τότε το αντικείμενο *Messsage* παίρνει τιμές για το αναγνωριστικό του κόμβου που του ανατέθηκε, σαν MessageAction την τιμή της απαρίθμησης για αίτηση απεγγραφής, και την τιμή της απαρίθμησης Channel για την οποία θα γίνει η αίτηση. Αν δεν έχει ορισθεί κάποιο κανάλι ή έχει ορισθεί η τιμή None της απαρίθμησης τότε επιλέγεται το κανάλι που προκύπτει από την επιλογή της rand με βάση τον μέγιστο αριθμό καναλιών που μπορεί να έχει ο κόμβος σύμφωνα με την τιμή που ορίστηκε στο αντικείμενο. Για την ολοκλήρωση της λειτουργίας της μεθόδου καλείται η στατική μέθοδος *InitializeTimestamp* της Message για να δώσει μοναδικό αναγνωριστικό στο μήνυμα και έπειτα μετατρέπεται το μήνυμα σε μια σειρά από bytes για να αποσταλεί στον εξυπηρετητή.

Τέλος θα πρέπει να σημειωθεί ότι η τάξη Client σε αντίθεση με την τάξη Server είναι σύγχρονη ως προς την εκτέλεση των λειτουργιών της καθώς η λογική της λειτουργίας της βασίζεται στην σειριακή αποστολή μηνυμάτων και όχι στην εξυπηρέτηση τους. Αυτό έχει ως αποτέλεσμα το αντικείμενο io_service να εκτελεί όλο το κύκλο διαδικασιών σειριακά και τα αντικείμενα της τάξης Client σε περίπτωση αποστολής επόμενου αιτήματος να περιμένουν την λήψη πρώτα αιτήματος απάντησης από τον εξυπηρετητή για να συνεχίσουν. Η δομή της τάξης παρουσιάζεται στο σχήμα 26.



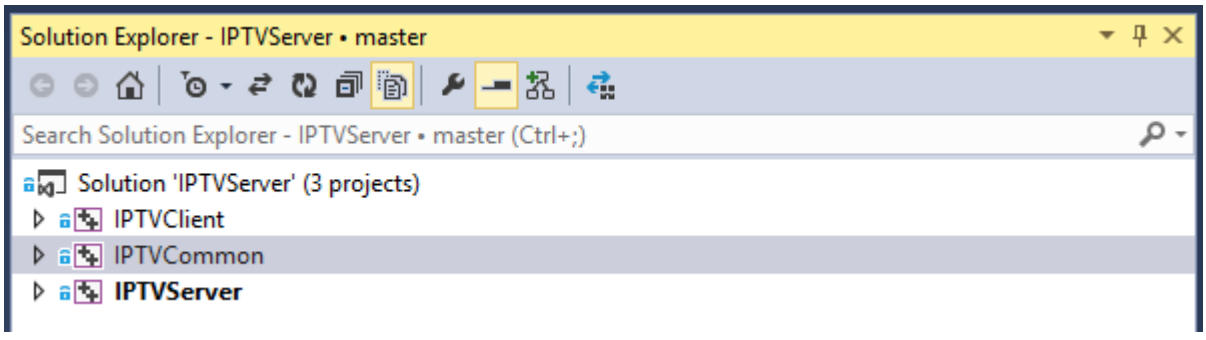
Σχήμα 26, Η τάξη Client.

Δομή προγράμματος

Το πρόγραμμα που υλοποιεί την συγκεκριμένη πειραματική αποτίμηση, ονομάζεται IPTVServer και είναι χωρισμένο σε 3 διαφορετικά υποπρογράμματα με βάση τον λογικό διαχωρισμό των λειτουργιών. Τα 3 αυτά υποπρογράμματα είναι:

- IPTVClient
- IPTVCommon
- IPTVServer

Η δομή του προγράμματος παρουσιάζεται στο σχήμα 27.



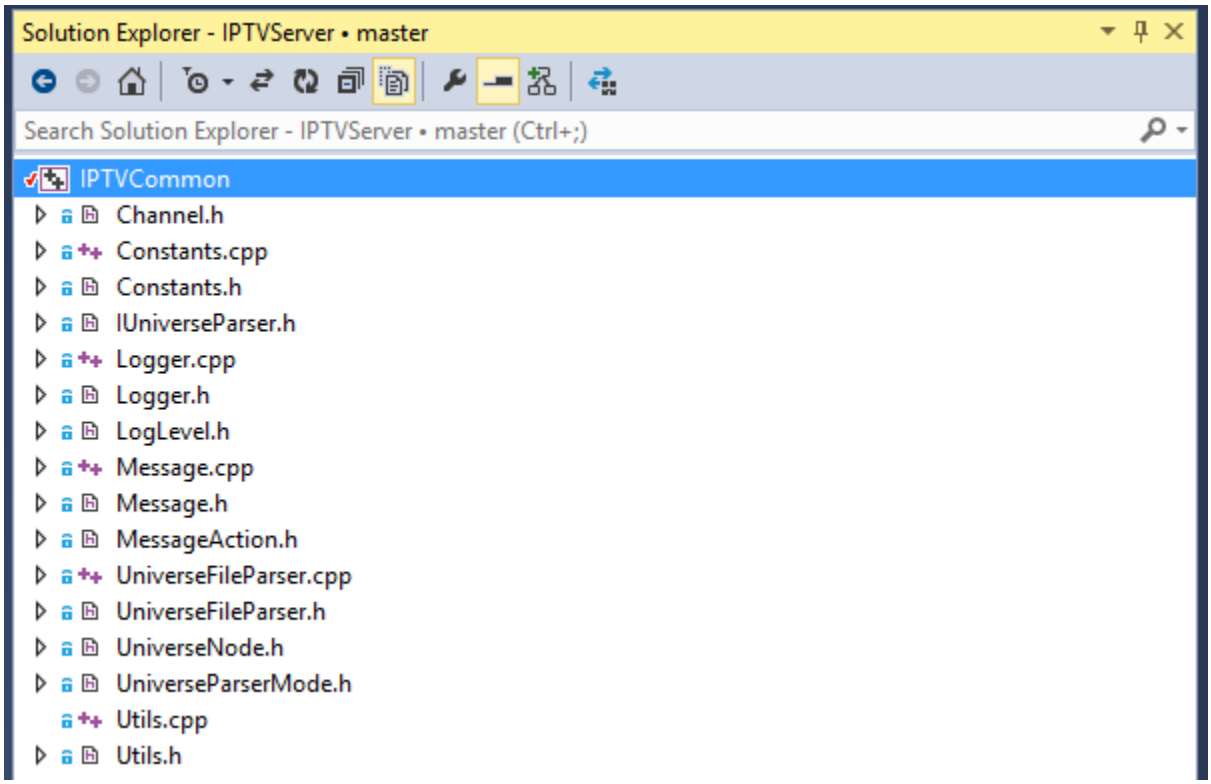
Σχήμα 27, Η δομή του προγράμματος

IPTVCommon

Το υποπρόγραμμα IPTVCommon περιέχει τις κοινές τάξεις που μοιράζονται τα δυο άλλα υποπρογράμματα. Έτσι το υποπρόγραμμα αυτό είναι βιβλιοθήκη για τα άλλα δυο. Μια βιβλιοθήκη που συμπεριλαμβάνεται στις

δηλώσεις των τάξεων των υποπρογραμμάτων που την χρησιμοποιούν για να έχουν πρόσβαση στις μεταβλητές και στις τάξεις που περιέχει. Η χρήση του υποπρογράμματος σαν βιβλιοθήκη δίνει την δυνατότητα τροποποίησης και συντήρησης των λειτουργιών μόνο σε ένα σημείο καθώς και ομοιόμορφη προσπέλαση και χρήση των περιεχομένων της.

Η βιβλιοθήκη IPTVCommon λοιπόν περιέχει τις καθολικές μεταβλητές του συστήματος, τις απαριθμήσεις που χρησιμοποιούνται, τις τάξεις για ανάγνωση της τοπολογίας, για καταγραφή συμβάντων, τις τάξεις για μετατροπές αντικειμένων καθώς και τις τάξεις που αποτελούν δομικά συστατικά του συστήματος όπως η *Message*. Αναλυτικότερα η τάξη *Constants* περιέχει τις σταθερές του συστήματος όπως η διεύθυνση IP του εξυπηρετητή, η πόρτα του εξυπηρετητή, το όνομα του αρχείου που περιέχει την τοπολογία καθώς και διάφορες άλλες παράμετροι που χρησιμοποιούνται από το σύστημα. Επίσης οι απαριθμήσεις *Channel, LogLevel, UniverseParseMode* και *MessageAction* βρίσκονται στην βιβλιοθήκη αυτή μιας και χρησιμοποιούνται και από τα άλλα δυο υποπρογράμματα. Η τάξη *Utils* είναι μια τάξη που μετατρέπει αντικείμενα σε σειρά από bytes και αντίστροφα, συν διάφορες άλλες μεθόδους μετατροπής ανάμεσα σε δομές και χρησιμοποιείται για την παροχή μεθόδων που δεν ανήκουν στην λειτουργικότητα των υπολοίπων τάξεων και χρειάζονται ανά περιπτώσεις. Η βιβλιοθήκη περιέχει και την τάξη *IUniverseParser* και την υλοποίησή της, *UniverseFileParser*, για την φόρτωση της τοπολογίας τόσο από το υποπρόγραμμα του εξυπηρετητή όσο και από το υποπρόγραμμα του πελάτη. Τέλος η βιβλιοθήκη διαθέτει και την τάξη *Logger* που χρησιμοποιείται για την καταγραφή συμβάντων αλλά και στατιστικών σε μορφή αρχείου txt. Τα περιεχόμενα της βιβλιοθήκης IPTVCommon παρουσιάζονται στο σχήμα 28.



Σχήμα 28, Τα περιεχόμενα του υποπρογράμματος IPTVCommon

IPTVClient

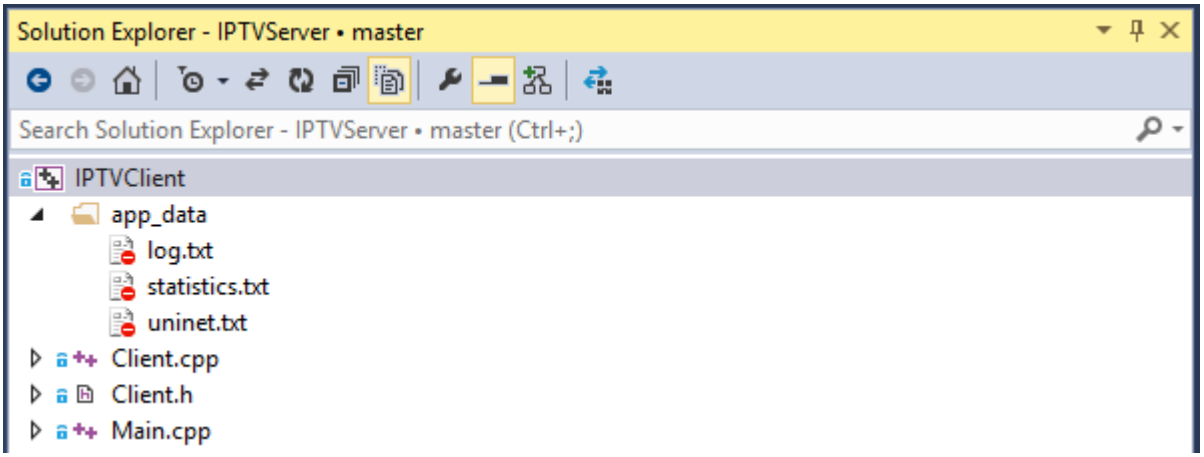
Το υποπρόγραμμα IPTVClient περιέχει τις τάξεις που αφορούν την λειτουργικότητα των κόμβων που δημιουργούν αιτήματα προς τον εξυπηρετητή. Τα αρχεία που περιλαμβάνονται στο υποπρόγραμμα είναι το αρχείο Main.cpp, το αρχείο που περιέχει την κεφαλίδα της τάξης Client και το αρχείο με την υλοποίησή της.

Η τάξη *Client*, είναι η τάξη που περιέχει όλη την λειτουργικότητα και την «επιχειρησιακή λογική» για την δημιουργία αιτημάτων. Όπως αναφέρθηκε και στην ενότητα Δομή πελάτη, η τάξη αυτή είναι μια τάξη που αναλόγως τις παραμέτρους που θα περαστούν, θα δημιουργήσει ένα αντικείμενο που θα αποτελέσει ένα κόμβο του δικτύου ο οποίος θα στέλνει μηνύματα στον εξυπηρετητή. Το είδος των μηνυμάτων αυτών καθώς και οι τιμές που θα περιέχουν, εξαρτώνται από την παραμετροποίηση του αντικειμένου. Επίσης και η συχνότητα αποστολής και το πλήθος των μηνυμάτων εξαρτώνται από τις παραμέτρους του αντικειμένου. Αυτό έχει σαν αποτέλεσμα τα αντικείμενα να είναι πλήρως παραμετροποιήσημα και η συμπεριφορά τους να ποικίλει εξαιτίας του χαρακτηριστικού αυτού. Παρόλα αυτά οι μέθοδοι που παρέχει η τάξη Client επιτρέπουν την κοινή τους διαχείριση μέσω των μεθόδων για έναρξη, διακοπή, επαναφορά κτλ. Έτσι όλα τα αντικείμενα μπορούν να διαχειριστούν ομοιόμορφα.

Το αρχείο Main.cpp είναι το αρχείο που περιέχει την βασική μέθοδο για την εκτέλεση του υποπρογράμματος IPTVClient. Η μέθοδος αυτή δημιουργεί αντικείμενα της τάξης *Client*, τα παραμετροποιεί κατάλληλα και τα διαχειρίζεται ώστε να πραγματοποιηθούν τα κατάλληλα αιτήματα. Τα αντικείμενα αυτά ουσιαστικά είναι η προσομοίωση πολλών κόμβων που αιτούνται και λειτουργούν με σειριακό τρόπο, και ανεξάρτητα το ένα από το άλλο.

Η λειτουργικότητα που παρέχει το υποπρόγραμμα οφείλεται και στην χρήση των τάξεων και των μεταβλητών της βιβλιοθήκης IPTVCommon. Για τον λόγο αυτό η βιβλιοθήκη συμπεριλαμβάνεται στην σύνδεση των αρχείων, linking, κατά την διάρκεια της μεταγλώττισης και της δημιουργίας του εκτελέσιμου κώδικα, καθώς και στις κεφαλίδες των αρχείων για την δήλωση της χρήσης των απαραίτητων αρχείων.

Τέλος, κατά την λειτουργία του υποπρογράμματος καταγράφονται τα συμβάντα που πραγματοποιούνται στον κύκλο ζωής των αντικειμένων και αποτελούν κυρίως προγραμματιστικά ίχνη για αποσφαλμάτωση αλλά και συμβάντα που συνδέονται με την παραγωγή στατιστικών για την εξομοίωση που πραγματοποιείται. Τα συμβάντα αυτά καταγράφονται σε ένα τοπικό φάκελο με όνομα app_data σε αρχεία με την χρήση της τάξης *Logger* της βιβλιοθήκης IPTVCommon. Ο φάκελος αυτός πρέπει να είναι στο ίδιο φάκελο με το εκτελέσιμο του προγράμματος και μέσα στον οποίο θα δημιουργηθούν δυο αρχεία κάθε φορά που εκτελείται το πρόγραμμα. Ένα αρχείο με όνομα log.txt και θα περιέχει τα συμβάντα καταγραφής για αποσφαλμάτωση και ένα αρχείο με όνομα statistics.txt που θα περιέχει τα συμβάντα καταγραφής για στατιστικά στοιχεία.



Τα περιεχόμενα του υποπρογράμματος IPTVClient παρουσιάζονται στο σχήμα 29.

Σχήμα 29, Τα περιεχόμενα του υποπρογράμματος IPTVClient

IPTVServer

Το υποπρόγραμμα IPTVServer περιέχει τις τάξεις που αφορούν την λειτουργικότητα του κόμβου εξυπηρετητή από την αρχικοποίησή του μέχρι και την κυρίως λειτουργία του. Τα αρχεία που περιλαμβάνονται στο υποπρόγραμμα είναι το αρχείο Main.cpp, το αρχείο που περιλαμβάνει την κεφαλίδα της τάξης Server, το αρχείο με την υλοποίησή της, το αρχείο με την κεφαλίδα της *SubscriptionHolder*, το αρχείο της υλοποίησης της, το αρχείο με την κεφαλίδα της *UniverseHolder*, το αρχείο της υλοποίησής της, το αρχείο με την τάξη *IMessageHolder* καθώς και τα αρχεία για τις δυο υλοποιήσεις της, την *MessageHolder* και την *MessageHolderForStats*.

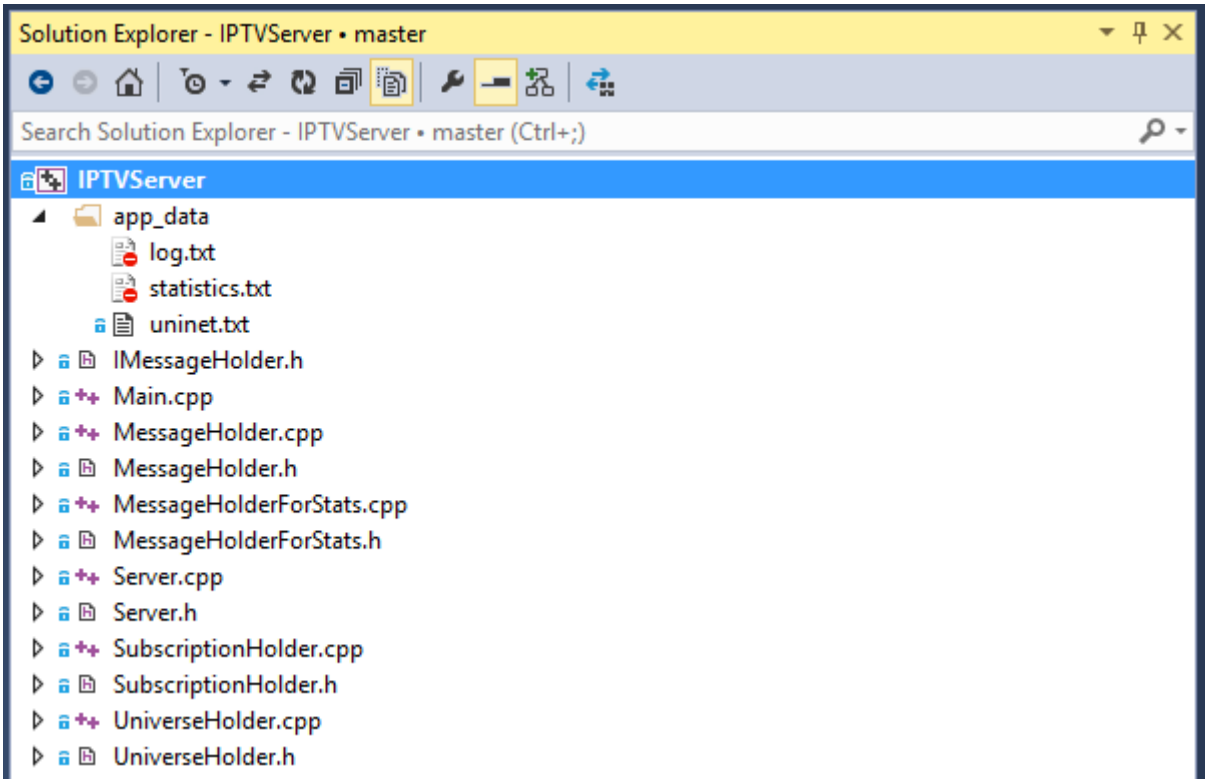
Η τάξη *Server*, είναι η τάξη που περιέχει όλη την λειτουργικότητα και την «επιχειρησιακή λογική» για την εξυπηρέτηση των αιτημάτων. Όπως αναφέρθηκε και στην ενότητα Δομή εξυπηρετητή, η τάξη αυτή είναι μια ενθυλάκωση άλλων αντικειμένων για μια τάξη που θα δημιουργήσει ένα αντικείμενο που θα λαμβάνει αιτήματα από τους κόμβους του δικτύου, θα τα εξυπηρετεί και θα αποστέλλει απαντήσεις προς τους αιτούμενους κόμβους ασύγχρονα. Οι λειτουργίες που θα εκτελέσει εξαρτώνται από το είδος του μηνύματος και αντίστοιχα θα δημιουργήσει ένα μήνυμα απάντησης για να ειδοποιήσει τον κόμβο αποστολής για το αίτημά του. Κατά την αρχικοποίηση ο εξυπηρετητής αρχικοποιεί τα απαραίτητα αντικείμενα που χρειάζεται για να φορτώσει την τοπολογία του δικτύου που πρέπει να γνωρίζει, να αρχικοποιήσει τις δομές για εγγραφές κόμβων στα αντίστοιχα κανάλια καθώς και να ανοίξει την πόρτα του λειτουργικού συστήματος από την οποία θα δέχεται αιτήσεις. Η τάξη *Server* περιλαμβάνει επίσης μεθόδους για την διαχείριση του εξυπηρετητή όπως έναρξη, διακοπή, προσωρινή διακοπή κτλ.

Το αρχείο Main.cpp είναι το αρχείο που περιέχει την βασική μέθοδο για την εκτέλεση του υποπρογράμματος IPTVServer. Στην μέθοδο αυτή ουσιαστικά δημιουργείται ένα αντικείμενο της τάξης Server, και καλείται η μέθοδος για έναρξη, ώστε να εξυπηρετηθούν τα αιτήματα που θα έρθουν. Παρόλα αυτά μέσα στην μέθοδο αυτή μπορούν να δημιουργηθούν παραπάνω από ένα αντικείμενα εξυπηρετητή τα οποία θα λαμβάνουν μηνύματα και θα απαντάνε ανάλογα με τις ανάγκες και την συμπεριφορά που επιδιώκεται.

Η λειτουργικότητα που παρέχει το υποπρόγραμμα οφείλεται και στην χρήση των τάξεων και των μεταβλητών της βιβλιοθήκης IPTVCommon. Για τον λόγο αυτό η βιβλιοθήκη συμπεριλαμβάνεται και σε αυτό το υποπρόγραμμα κατά την σύνδεση των αρχείων, linking, κατά την λειτουργία της μεταγλώττισης και της δημιουργίας του εκτελέσιμου κώδικα. Το ίδιο ισχύει και για τις κεφαλίδες των αρχείων που περιέχουν τις δηλώσεις των αρχείων που χρειάζονται.

Τέλος, κατά την λειτουργία του υποπρογράμματος, όπως και στο αντίστοιχο υποπρόγραμμα IPTVClient, καταγράφονται τα συμβάντα καταγραφής για αποσφαλμάτωση που πραγματοποιούνται στον κύκλο ζωής του εξυπηρετητή καθώς και τα συμβάντα που συνδέονται με την παραγωγή στατιστικών σύμφωνα με τα αιτήματα που δέχεται ο εξυπηρετητής. Τα συμβάντα αυτά καταγράφονται σε ένα τοπικό φάκελο με όνομα app_data σε αρχεία με την χρήση της τάξης *Logger* της βιβλιοθήκης IPTVCommon. Ο φάκελος αυτός είναι ξεχωριστός από τον αντίστοιχο του υποπρογράμματος IPTVClient αλλά έχει όμοια δομή για χάρη ευκολίας. Δηλαδή ο φάκελος αυτός θα περιέχει δυο αρχεία που θα δημιουργούνται κάθε φορά που εκτελείται το πρόγραμμα. Ένα αρχείο με όνομα log.txt και θα περιέχει τα συμβάντα καταγραφής για αποσφαλμάτωση και ένα αρχείο με όνομα statistics.txt που θα περιέχει τα συμβάντα καταγραφής για στατιστικά στοιχεία. Επίσης στον φάκελο αυτό περιέχεται και το αρχείο, uninet.txt, που χρησιμοποιείται για την αρχικοποίηση και την φόρτωση της τοπολογίας από τον εξυπηρετητή με την βοήθεια της τάξης *UniverseFileParser* της βιβλιοθήκης IPTVCommon.

Τα περιεχόμενα του υποπρογράμματος IPTVServer παρουσιάζονται στο σχήμα 30.



Σχήμα 30, Τα περιεχόμενα του υποπρογράμματος IPTVServer

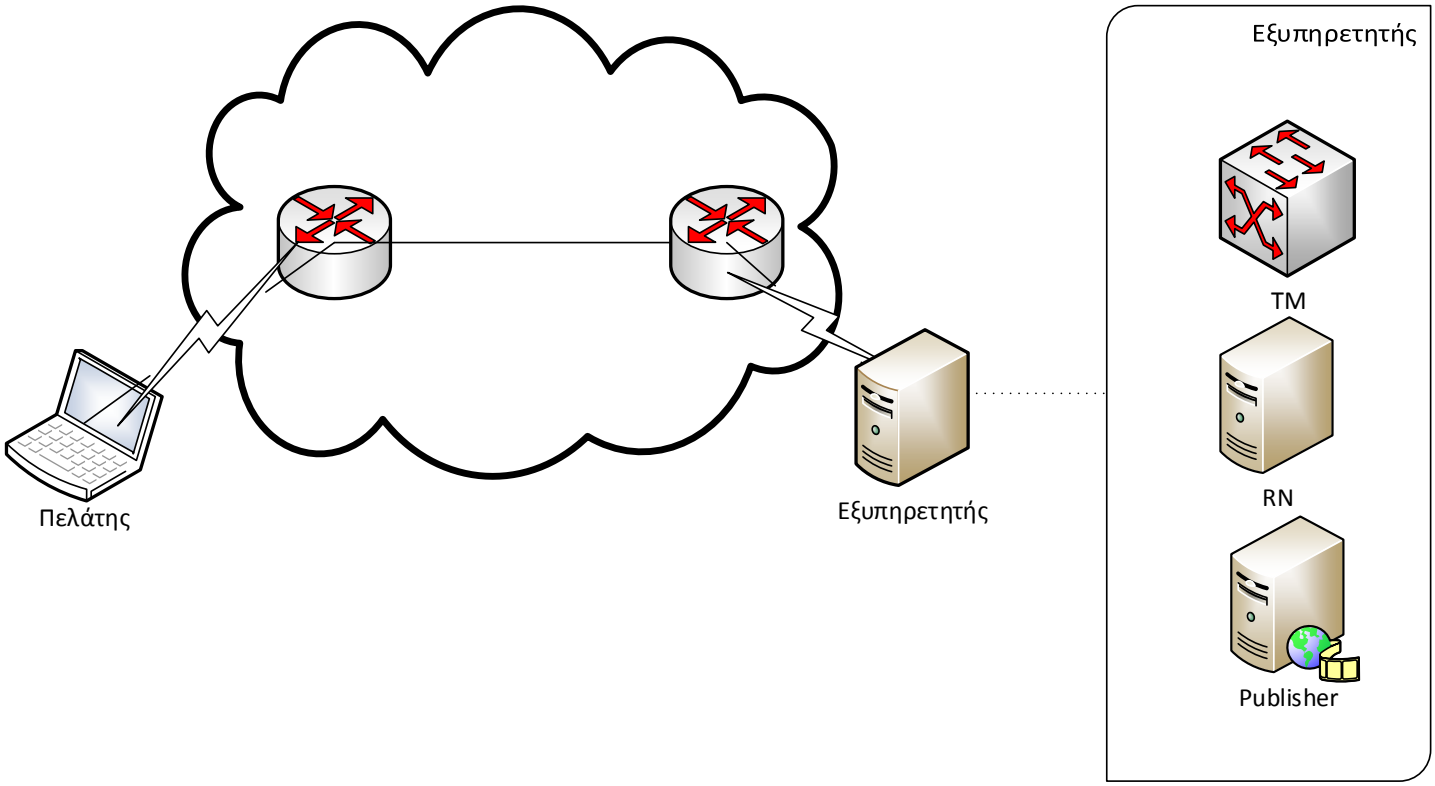
Ροή μηνυμάτων

Στο σύστημα που δημιουργήθηκε για την πειραματική αποτίμηση εξηγήθηκαν οι βασικές δομές λειτουργίας για την κατανόηση τόσο του τι κάνει ο εξυπηρετητής όσο και του τι κάνει ο πελάτης, κατά την αρχικοποίηση και την ανταλλαγή μηνυμάτων. Στην ενότητα αυτή θα παρουσιαστεί η συνολική εικόνα του συστήματος κατά την ανταλλαγή αιτημάτων μεταξύ ενός κόμβου και του εξυπηρετητή.

Στο σύστημα PURSUIT το πρώτο βήμα του συστήματος είναι ο εξυπηρετητής να ξεκινήσει την αρχικοποίησή του και να δημοσιεύσει στο δίκτυο ότι διαθέτει πληροφοριακό περιεχόμενο προς δημοσίευση. Στην συγκεκριμένη προσομοίωση του συστήματος, ο κόμβος που είναι υπεύθυνος για την τοπολογία του δικτύου, Topology Manager, ο κόμβος που αποτελεί το σημείο συνάντησης, ο Rendezvous Node, και ο κόμβος που δημοσιεύει το πληροφοριακό περιεχόμενο, ο Publisher, είναι ο ίδιος ο εξυπηρετητής. Για τον λόγο αυτό δεν χρειάζεται ο εξυπηρετητής να δημοσιεύσει σε κάποιον άλλο εκτός από τον εαυτό του ότι διαθέτει πληροφοριακό περιεχόμενο προς δημοσίευση μιας και όλα τα αιτήματα για ερώτηση στο ποιος κατέχει το αιτούμενο πληροφοριακό περιεχόμενο απαντώνται από τον ίδιο. Επίσης στη συγκεκριμένη προσομοίωση το μόνο πληροφοριακό περιεχόμενο που ζητείται και παρέχεται, είναι η παροχή ροής δεδομένων διαδικτυακής τηλεόρασης. Έτσι δεν προκύπτουν άλλες λειτουργίες για προώθηση μηνυμάτων σε άλλους κόμβους που δημοσιεύουν περιεχόμενα, ή αντιστοίχιση κόμβων με περιεχόμενα.

Κατά την αρχικοποίηση ο εξυπηρετητής επειδή δεν γνωρίζει την τοπολογία, και ούτε περιγράφεται στο σύστημα PURSUIT, η ανακάλυψη των γειτονικών κόμβων, φορτώνει την αρχική τοπολογία μέσω αρχείου η οποία εκ των υστέρων διαμορφώνεται δυναμικά με μηνύματα αίτησης για προσθήκη στην τοπολογία και αντίστοιχα διαγραφής από την τοπολογία. Η φόρτωση πραγματοποιείται μέσω του αντικειμένου *UniverseFileParser* και η διαχείριση της τοπολογίας μέσω του *UniverseHolder*. Μετά την φόρτωση αρχικοποιείται και η δομή διαχείρισης των εγγραφών ανά κανάλι μέσω του αντικειμένου *SubscriptionHolder* και τέλος ξεκινάει ο εξυπηρετητής να περιμένει για αιτήματα σε μια προκαθορισμένη πόρτα και IP διεύθυνση, μέσω του αντικειμένου *MessageHolder*. Από την πλευρά του πελάτη η αρχικοποίηση ξεκινάει με το τι αιτήματα θα κάνει, για ποια κανάλια θα αιτηθεί, πόσους και ποιους γείτονες θα έχει και τι αναγνωριστικό κόμβου θα έχει. Η διεύθυνση του εξυπηρετητή καθώς και η πόρτα του θεωρούνται γνωστά και για τον λόγο αυτό κατά την παραμετροποίηση τα παρέχουμε στον πελάτη. Αυτό συμβαίνει γιατί στο σύστημα PURSUIT η ανακάλυψη του κόμβου συνάντησης είναι εσωτερική λειτουργία του δικτύου και δεν εξαρτάται από το σύστημα αυτό καθ' εαυτό.

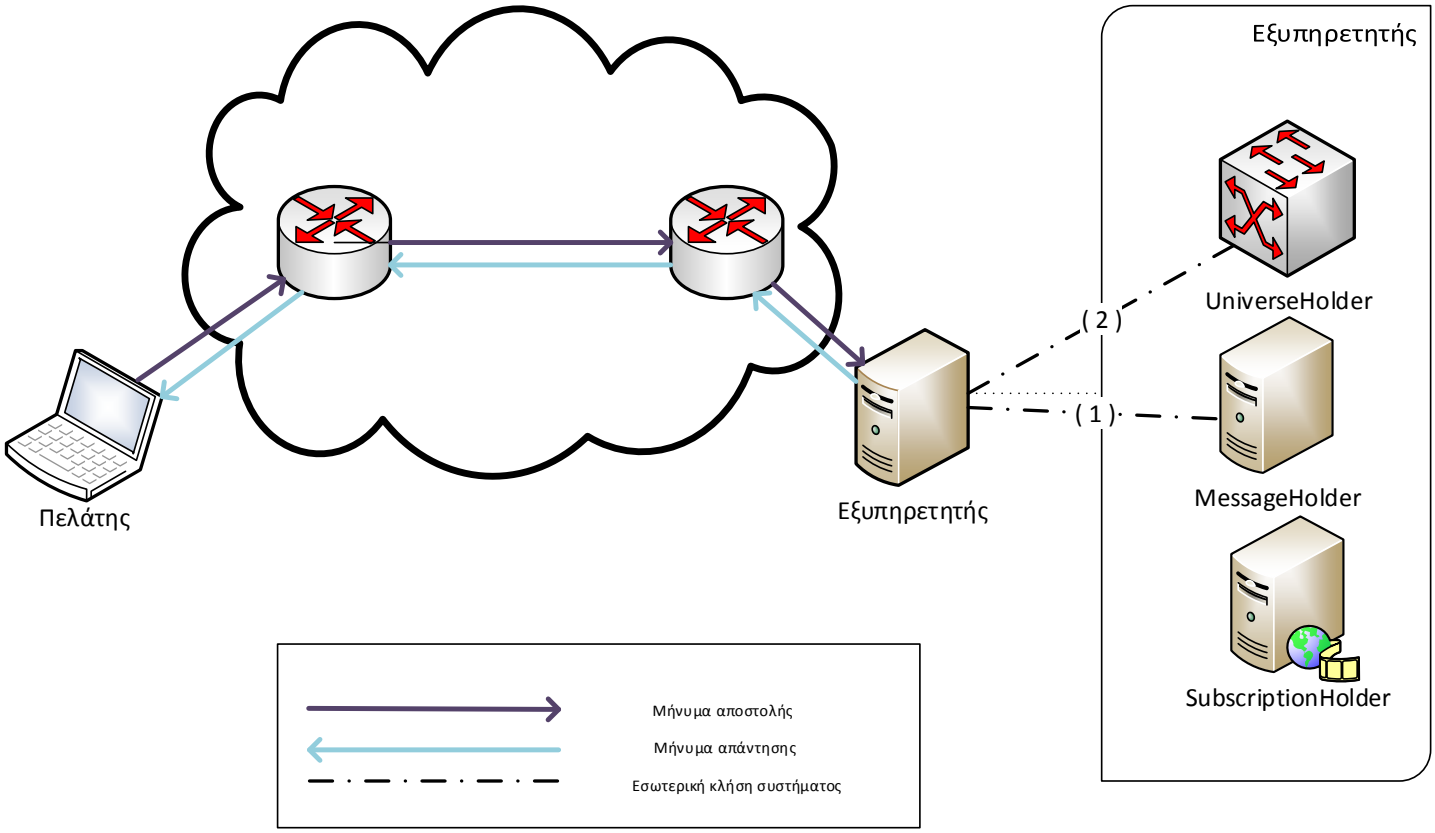
Σαν αποτέλεσμα η αρχιτεκτονική του δικτύου να είναι πολύ απλή σε σχέση με αυτή που απαιτείται σε ένα σύστημα ICN, όπως το PURSUIT. Η αρχιτεκτονική του δικτύου παρουσιάζεται στο σχήμα 31.



Σχήμα 31, Αρχιτεκτονική συστήματος

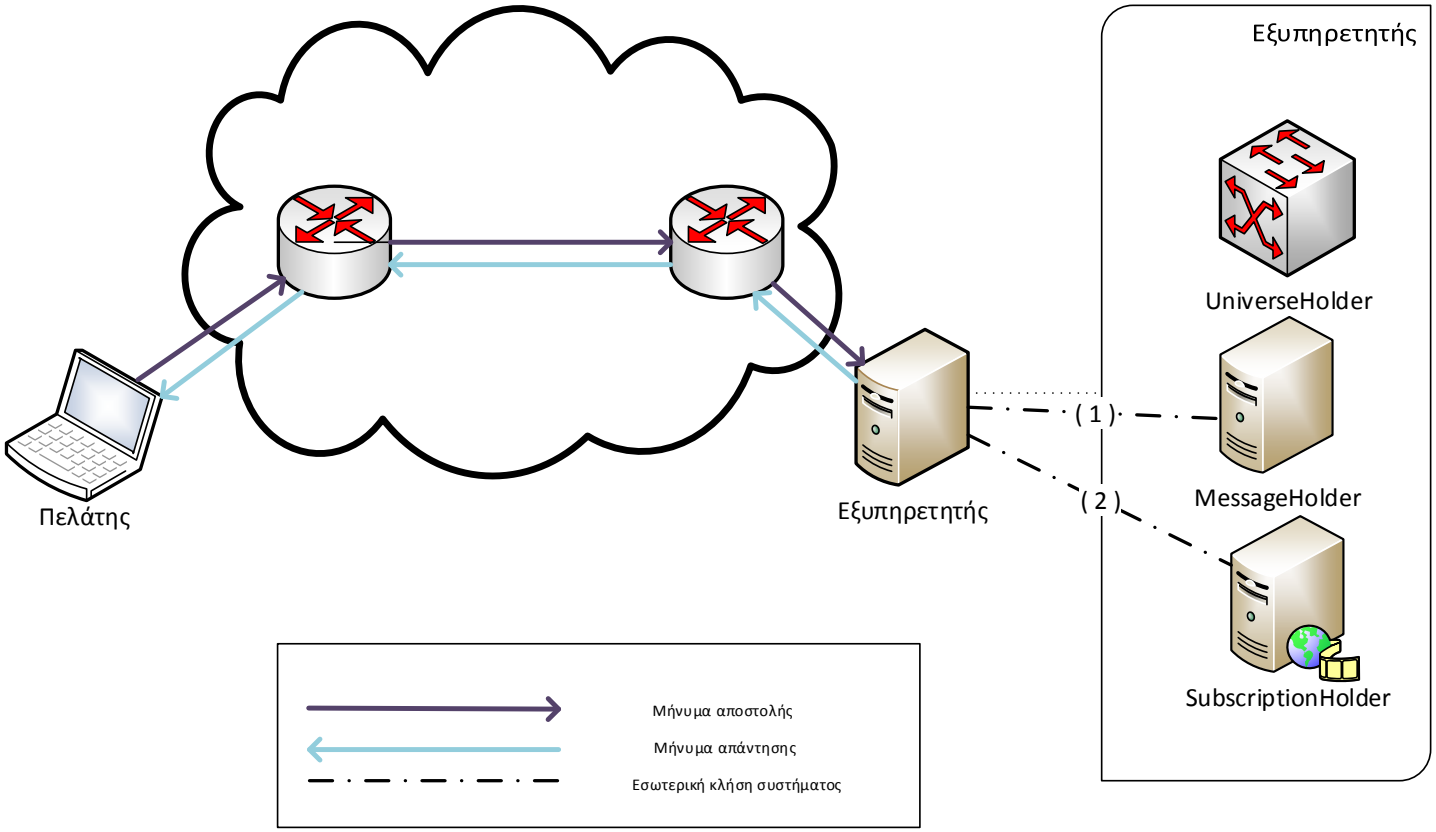
Εφόσον και τα δυο επικοινωνούντα μέρη αρχικοποιήθηκαν η επικοινωνία ουσιαστικά ξεκινάει με την αποστολή μηνυμάτων από την πλευρά του πελάτη.

Έστω ότι ο κόμβος - πελάτης θέλει να εισέλθει στην τοπολογία του εξυπηρετητή. Αρχικά ο κόμβος αυτός θα πρέπει να δημιουργήσει ένα *Μήνυμα προσθήκης στο δίκτυο*. Στο μήνυμα αυτό θα πρέπει να συμπεριλάβει στο πεδίο επιλογές και τα αναγνωριστικά των γειτονικών του κόμβων και στο πεδίο αναγνωριστικό κόμβου, το μοναδικό του αναγνωριστικό. Με την δημιουργία του μηνύματος θα το αποστείλει στο δίκτυο, εφόσον βρει πρώτα την πραγματική διεύθυνση του εξυπηρετητή. Με την αποστολή του μηνύματος, ο κόμβος περιμένει για απάντηση από τον εξυπηρετητή. Ο εξυπηρετητής λαμβάνει το μήνυμα μέσω του μέρους του προγράμματος που είναι υπεύθυνο για την λήψη των μηνημάτων, MessageHolder, το ελέγχει, αναγνωρίζει το είδος του μηνύματος και καλεί το μέρος του προγράμματος που είναι υπεύθυνο για την διαχείριση της τοπολογίας, UniverseHolder, για να προσθέσει τον κόμβο στην τοπολογία. Μόλις αυτό πραγματοποιηθεί, ο εξυπηρετητής αποστέλλει ένα αντίγραφο του μηνύματος που έλαβε για να επιβεβαιώσει την προσθήκη του κόμβου στην τοπολογία. Η ροή αυτή παρουσιάζεται στο σχήμα 32.



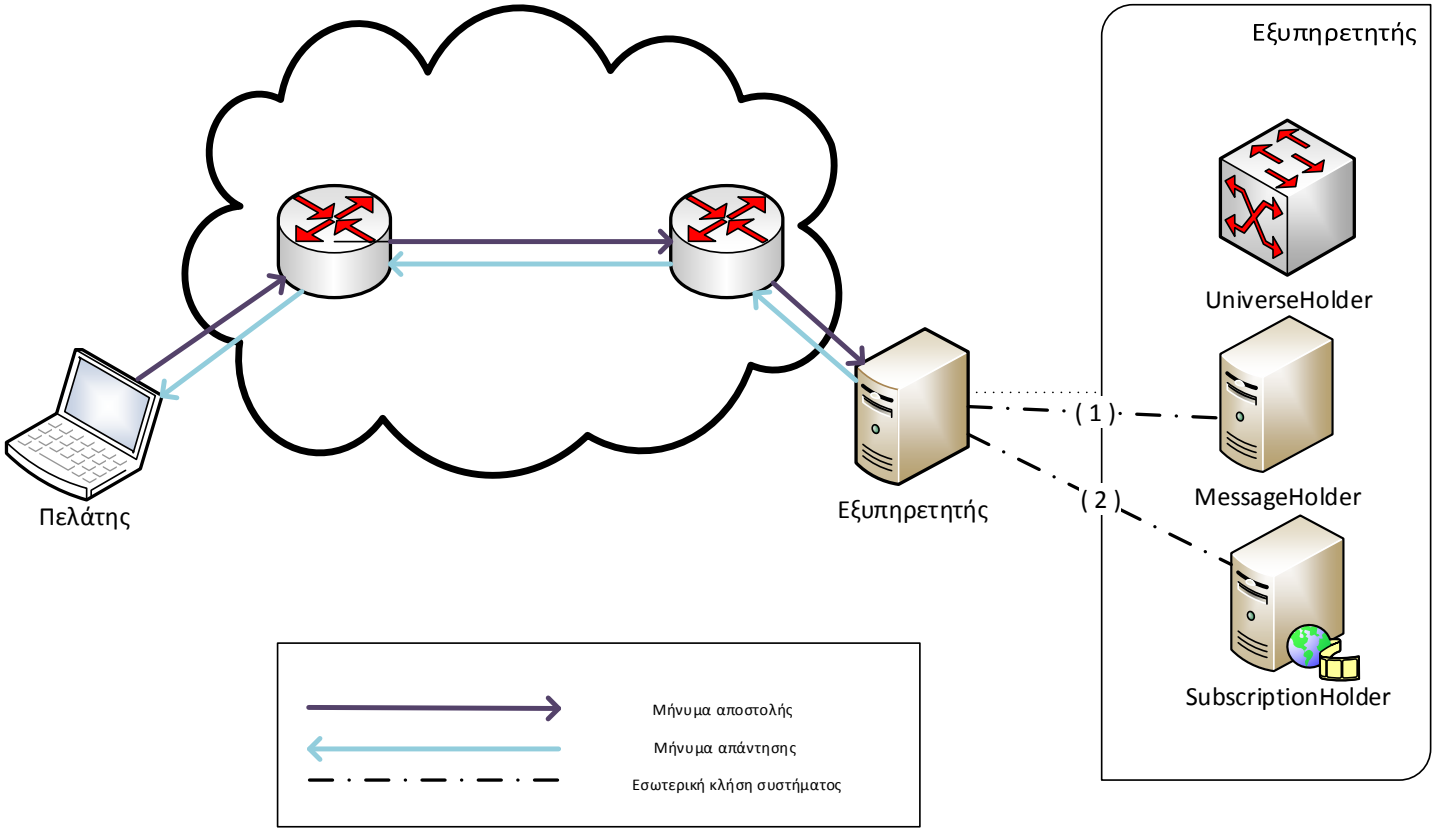
Σχήμα 32, Ροή μηνύματος προσθήκης στο δίκτυο

Εφόσον ο κόμβος - πελάτης έχει εισέλθει στην τοπολογία του εξυπηρετητή, μπορεί να πραγματοποιήσει αιτήματα προς τον εξυπηρετητή για λήψη του πληροφοριακού περιεχομένου. Για να πραγματοποιηθεί αυτό ο κόμβος θα πρέπει πρώτα να εγγραφεί σε κάποιο κανάλι. Έτσι πρέπει να δημιουργήσει ένα *Μήνυμα εγγραφής σε κανάλι*. Στο μήνυμα αυτό θα πρέπει να συμπεριλάβει στο πεδίο κανάλι το επιθυμητό κανάλι. Μόλις το δημιουργήσει θα το αποστείλει στο δίκτυο με παραλήπτη την διεύθυνση του εξυπηρετητή. Με την αποστολή του μηνύματος, ο κόμβος περιμένει για απάντηση από τον εξυπηρετητή. Ο εξυπηρετητής λαμβάνει το μήνυμα μέσω του μέρους του προγράμματος που είναι υπεύθυνο για την λήψη των μηνημάτων, MessageHolder, το ελέγχει, αναγνωρίζει το είδος του μηνύματος και καλεί το μέρος του προγράμματος που είναι υπεύθυνο για την διαχείριση των εγγραφών, τον SubscriptionHolder και εγγράφει τον κόμβο στο κανάλι που ζήτησε. Μόλις αυτό πραγματοποιηθεί, ο εξυπηρετητής αποστέλλει ένα αντίγραφο του μηνύματος που έλαβε για να επιβεβαιώσει την εγγραφή του κόμβου στο αιτούμενο κανάλι. Η ροή αυτή παρουσιάζεται στο σχήμα 33.



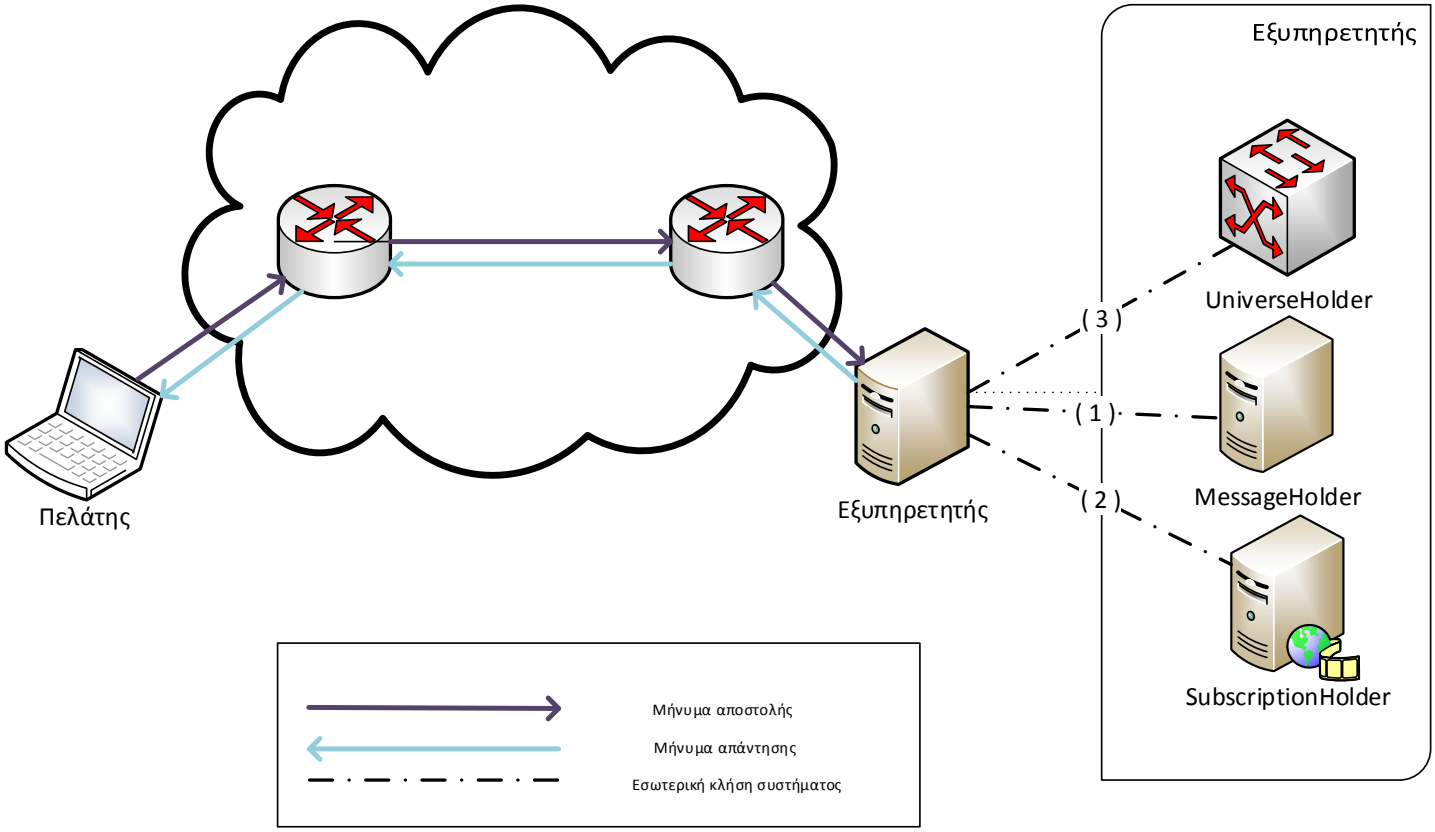
Σχήμα 33, Ροή μηνύματος εγγραφής σε κανάλι

Στην περίπτωση που ο κόμβος - πελάτης έχει εγγραφεί σε ένα κανάλι και θέλει να απεγγραφεί από αυτό θα πρέπει δημιουργήσει ένα *Μήνυμα διαγραφής από κανάλι*. Στο μήνυμα αυτό θα πρέπει να συμπεριλάβει στο πεδίο κανάλι, το κανάλι από το οποίο επιθυμεί να αποχωρήσει. Μόλις το δημιουργήσει θα το αποστείλει στο δίκτυο με παραλήπτη την διεύθυνση του εξυπηρετητή. Με την αποστολή του μηνύματος, ο κόμβος θα περιμένει για απάντηση από τον εξυπηρετητή. Ο εξυπηρετητής λαμβάνει το μήνυμα μέσω του MessageHolder, το ελέγχει, αναγνωρίζει το είδος του μηνύματος και καλεί το μέρος του προγράμματος που είναι υπεύθυνο για την διαχείριση των εγγραφών, τον SubscriptionHolder και διαγράφει τον κόμβο στο κανάλι που ζήτησε, εφόσον είναι εγγεγραμμένος. Είτε είναι είτε δεν είναι, ο εξυπηρετητής αποστέλλει ένα αντίγραφο του μηνύματος που έλαβε και επιβεβαιώνει την απεγγραφή του κόμβου από το αιτούμενο κανάλι. Η ροή αυτή παρουσιάζεται στο σχήμα 34.



Σχήμα 34, Ροή μηνύματος διαγραφής από κανάλι

Τέλος υπάρχει η περίπτωση που ο κόμβος - πελάτης θέλει να εξέλθει από την τοπολογία του εξυπηρετητή. Στην περίπτωση αυτή ο κόμβος θα πρέπει να δημιουργήσει ένα *Μήνυμα αποχώρησης από το δίκτυο*. Στο μήνυμα αυτό θα πρέπει να συμπεριλάβει στο πεδίο αναγνωριστικό κόμβου, το μοναδικό του αναγνωριστικό. Μόλις δημιουργήσει το μήνυμα θα πρέπει να το αποστείλει στο δίκτυο με προορισμό τον εξυπηρετητή. Με την αποστολή του μηνύματος, ο κόμβος μπορεί να κλείσει την επικοινωνία ή να περιμένει για επιβεβαίωση από τον εξυπηρετητή. Ο εξυπηρετητής λαμβάνει το μήνυμα μέσω του MessageHolder, το ελέγχει, αναγνωρίζει το είδος του μηνύματος και καλεί αρχικά τον SubscriptionHolder για να τον διαγράψει από τα κανάλια που είναι εγεγγραμμένος και ύστερα καλεί τον UniverseHolder, για να διαγράψει τον κόμβο και από την τοπολογία. Μόλις αυτό πραγματοποιηθεί, ο εξυπηρετητής αποστέλλει ένα αντίγραφο του μηνύματος που έλαβε για να επιβεβαιώσει την διαγραφή του κόμβου στην τοπολογία. Η ροή αυτή παρουσιάζεται στο σχήμα 35.



Σχήμα 35, Ροή μηνύματος διαγραφής από το δίκτυο

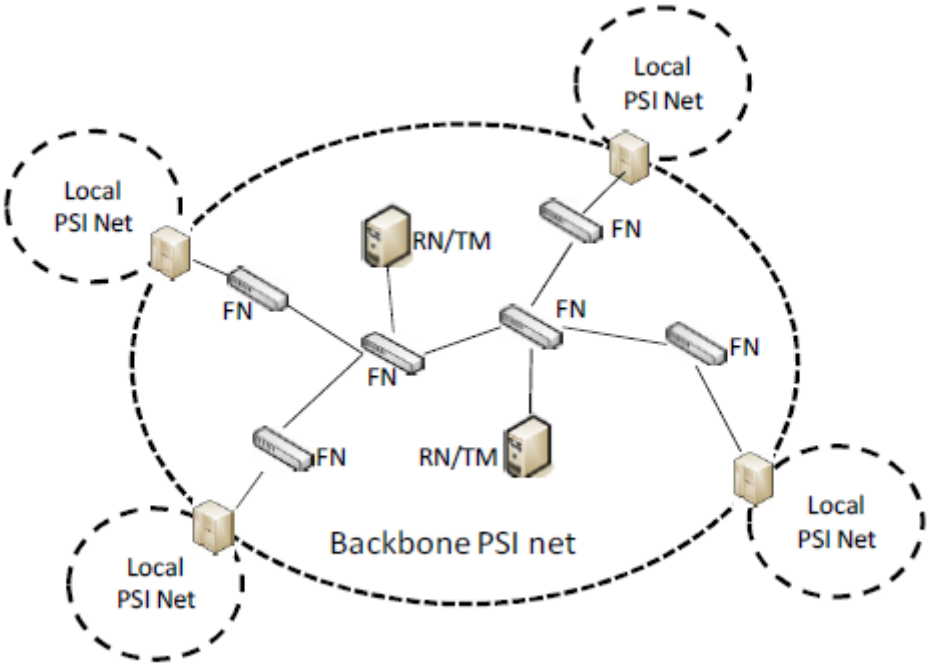
Πειραματική αποτίμηση

Στις προηγούμενες ενότητες αναλύθηκε τόσο η αρχιτεκτονική ICN όσο και η αρχιτεκτονική του συστήματος PURSUIT, στα πλαίσια των οποίων παρουσιάστηκε η λειτουργία τους και τα δομικά χαρακτηριστικά των αρχιτεκτονικών αυτών. Η αρχιτεκτονική ICN προσελκύει όλο και περισσότερο την προσοχή της επιστημονικής κοινότητας σε θέματα δικτύωσης καθώς η αυξανόμενη χρήση του διαδικτύου με σκοπό την πρόσβαση σε πληροφορίες, δίνει την δυνατότητα για σύνδεση πληροφοριών αντί της σύνδεσης μηχανημάτων.

Οι τωρινές προσπάθειες που καταβάλλονται πάνω στην αρχιτεκτονική ICN επικεντρώνονται στην βελτιστοποίηση της παράδοσης της πληροφορίας μέσω του μηχανισμού προσωρινής αποθήκευσης. Προς την κατεύθυνση αυτή κινούνται και ερευνητές που μελετάνε πως η παραμετροποίηση της δρομολόγησης και των πρωτοκόλλων προώθησης μπορεί να βοηθήσει στην περαιτέρω χρήση της μνήμης και του χώρου των δρομολογητών. Τα πλεονεκτήματα της προσωρινής αποθήκευσης εμφανίζονται σε συγκεκριμένου τύπου εφαρμογές όπως οι μεταφορές μεγάλου όγκου δεδομένων. Η χρήση του διαδικτύου παρόλα αυτά βασίζεται στην διάδοση πληροφοριών σε πραγματικό χρόνο, όπως είναι η διαδικτυακή τηλεόραση, το διαδικτυακό ραδιόφωνο, αλλά και στις ειδοποιήσεις σε πραγματικό χρόνο, όπως οι μετρήσεις αισθητήρων, οι μηχανισμοί ειδοποίησης κ.α. Σε αυτού του τύπου εφαρμογές τα πλεονεκτήματα της προσωρινής αποθήκευσης είναι αμφισβητούμενα. Για παράδειγμα στην περίπτωση της συνομιλίας με ήχο, η ύπαρξη αποθήκευσης εντός του δικτύου έχει πολύ μικρό όφελος, γιατί τα πακέτα που μεταφέρουν τον ήχο έχουν πολύ μικρή διάρκεια ζωής. Οι εφαρμογές πραγματικού χρόνου έχουν από την φύση τους την διαδικασία εγγραφής και δημοσίευσης και μπορούν να επωφεληθούν ιδιαιτέρως σε σχήματα με μετάδοση πολυδιανομής. Για τον λόγο αυτό θεωρείται απαραίτητη η διεύρυνση του πεδίου ερευνών και σε συστήματα αποδοτικής μετάδοσης με πολυδιανομή για την κάλυψη και τέτοιου είδους εφαρμογών.

Αρχιτεκτονική PSI σε δίκτυα ευρείας περιοχής

Η αρχιτεκτονική PSI σε δίκτυα ευρείας περιοχής περιλαμβάνει τα τοπικά δίκτυα PSI και ένα δίκτυο – κορμού PSI που τα ενώνει ώστε να παρέχει πρόσβαση στους χρήστες των τοπικών δικτύων PSI και σε πληροφορίες εκτός του τοπικού τους δικτύου. Στην αρχιτεκτονική αυτή χρησιμοποιείται η τεχνική LIPSIN [27] ως λειτουργία προώθησης, η οποία κωδικοποιεί τα μονοπάτια παράδοσης με φίλτρα BLOOM, είτε είναι αποκλειστικής διανομής είτε είναι πολυδιανομής. Το χαρακτηριστικό αυτής της τεχνικής είναι ότι κάθε πολλαπλή προώθηση είναι ξεχωριστή και δεν χρειάζεται οι κόμβοι προώθησης να αποθηκεύουν πληροφορίες για κάθε δέντρο πολυδιανομής. Η λειτουργία της συνάντησης υλοποιείται μέσω πολλαπλών κόμβων Rendezvous που διαμοιράζουν το φόρτο παρακολούθησης και εξυπηρέτησης των δημοσιεύσεων με κατανεμημένο τρόπο. Όταν μια δημοσίευση ή εγγραφή αιτείται, προωθείται στο κοντινότερο κόμβο συνάντησης και από εκεί δρομολογείται μέσω του κατανεμημένου πίνακα καταγραφής συσχετίσεων μεταξύ δημοσιεύσεων και εγγραφών, RN-DHT. Η ύπαρξη των κόμβων συνάντησης και των κόμβων προώθησης είναι συνυφασμένη όπως η ύπαρξη DNS εξυπηρετητή σε ένα κόμβο με IP διεύθυνση. Οι λειτουργίες διαχείρισης της τοπολογίας είναι διασκορπισμένες σε όλους τους κόμβους. Το πρώτο μέρος της λειτουργίας διαχείρισης της τοπολογίας είναι η ανακάλυψη του τοπολογίας και η παρακολούθηση του δικτύου. Αυτό πραγματοποιείται μέσω ενός σχήματος δρομολόγησης με πληροφορίες κατάστασης των συνδέσεων, όπου οι κόμβοι προώθησης και οι δρομολογητές που βρίσκονται ανάμεσα σε διαφορετικά δίκτυα ανταλλάσσουν πληροφορίες σχετικά με τις συνδέσεις τους. Το δεύτερο μέρος της λειτουργίας διαχείρισης είναι ο υπολογισμός των μονοπατιών που πραγματοποιείται από τους κόμβους συνάντησης, RN. Όταν ένας κόμβος συνάντησης εξυπηρετεί μια εγγραφή, εξάγει την πληροφορία για την τοπολογία από το κοντινό του κόμβο προώθησης και υπολογίζει το μονοπάτι ανάμεσα στον αιτών και αυτόν που παρέχει την πληροφορία. Το αποτέλεσμα αυτού του υπολογισμού κωδικοποιείται σε ένα αναγνωριστικό LIPSIN, και παραδίδεται στον κόμβο που δημοσιεύει την πληροφορία. Παράδειγμα της τοπολογίας PSI σε δίκτυα ευρείας περιοχής παρουσιάζεται στο σχήμα 36.



Σχήμα 36, Αρχιτεκτονική PSI σε δίκτυα ευρείας περιοχής

Ο υπολογισμός του δέντρου πολυδιανομής στην αρχιτεκτονική αυτή πραγματοποιείται από τους κόμβους συνάντησης κατά την εξυπηρέτηση ενός αιτήματος για εγγραφή. Η βασική διαδικασία είναι ο υπολογισμός του δέντρου συντομότερων μονοπατιών που προκύπτει από την ένωση των συντομότερων μονοπατιών από τον κόμβου που δημοσιεύει την πληροφορία προς κάθε κόμβου που είναι εγγεγραμμένος για την πληροφορία αυτή. Όμως η κεντρικοποιημένη φύση του υπολογισμού του δέντρου, δίνει την δυνατότητα για εφαρμογή πιο εξελιγμένων αλγορίθμων για την κατασκευή του δέντρου πολυδιανομής. Για παράδειγμα κάθε κόμβος συνάντησης μπορεί να υπολογίσει τα βέλτιστα, σε θέμα κόστους, δέντρα Steiner [28].

Τα δέντρα πολυδιανομής κατασκευάζονται με μηδενικό επιπρόσθετο φόρτο σηματοδότησης για το δίκτυο. Το μόνο που απαιτείται είναι μια η εγγραφή στον κατανεμημένο πίνακα RN-DHT και η αποστολή μιας ειδοποίησης στον κόμβο που δημοσιεύει. Επίσης λόγω της ικανότητας της τεχνικής LIPSIN για ανεξάρτητη πολυδιανομή, δεν χρειάζεται η ανταλλαγή επιπρόσθετων μηνυμάτων ελέγχου. Η κατασκευή δέντρων Steiner δεν απαιτεί επιπλέον σηματοδότηση σε σχέση με την κατασκευή δέντρων συντομότερων μονοπατιών. Σε σύγκριση με το κόστος σηματοδότησης που απαιτείται στο πρωτόκολλο IP για την κατασκευή του δέντρου, στο PSI είναι ανάλογο της ανάλυσης που απαιτείται από το DNS και DHT αντίστοιχα. Από την άλλη πλευρά η λειτουργία αυτή απαιτεί σημαντικά περισσότερο φόρτο υπολογισμού στους κόμβους συνάντησης καθώς ο κάθε κόμβος θα πρέπει διαχειρίζεται δημοσιεύσεις, εγγραφές, να παρακολουθεί τους ενεργούς εγγεγραμμένους για κάθε αντικείμενο και να υπολογίζει δέντρα πολυδιανομής. Για τον λόγο αυτό απαιτείται προσεκτική πρόβλεψη στην επιλογή αριθμού και υλικού δικτύωσης κατά την εγκατάσταση κόμβων συνάντησης.

Η αναφερθείσα αρχιτεκτονική μεταθέτει τις λειτουργίες ελέγχου, όπως η παρακολούθηση των κόμβων που συμμετέχουν σε πολυδιανομή, η δημιουργία του δέντρου κτλ, σε μια λογικά χωρισμένη κεντρική μονάδα τον RN-DHT. Τα κεντρικοποιημένα σχήματα συνήθως αντιμετωπίζονται με σκεπτικισμό λόγω των περιορισμών επεκτασιμότητας, παρόλα αυτά οι κόμβοι συνάντησης λαμβάνουν υπόψιν τους μόνο την τοπολογία της ευρείας περιοχής. Δεν υπολογίζουν τους χρήστες και τα δίκτυα πρόσβασης σαν μέρος του δικτύου κορμού, αλλά σαν γειτονικά δίκτυα PSI. Έτσι αυτή η λογική αφαίρεση μειώνει το φόρτο υπολογισμού για τους κόμβους συνάντησης του δικτύου κορμού. Επίσης τελευταία [29], τα πλεονεκτήματα των μεταθέσεων λειτουργιών σε κεντρικοποιημένες μονάδες αυξάνονται σημαντικά σε σχέση με τα πλεονεκτήματα επεκτασιμότητας των κατανεμημένων συστημάτων.

Προϋποθέσεις

Με βάση αυτή την αρχιτεκτονική δημιουργήθηκε το αναφερθέν πρόγραμμα ώστε να αποτιμηθεί πειραματικά η επίδοση ενός εξυπηρετητή σε δίκτυα πολυδιανομής. Το πρόγραμμα λειτουργεί σαν ένα συνεκτικό δίκτυο. Οι συνδέσεις μεταξύ των κόμβων είναι μέσω καναλιών UDP. Για την ανακάλυψη των κόμβων πραγματοποιείται φόρτωση μέσω αρχείου ώστε να μειωθεί ο φόρτος στους κόμβους συνάντησης και προώθησης και για να αποτιμηθεί πόσο σημαντικό ρόλο παίζει η τοπολογία στην απόδοση του δικτύου. Παρόλα αυτά το πρόγραμμα διαθέτει την ικανότητα για δυναμική διαμόρφωση του δικτύου μέσω μηνυμάτων, δηλαδή προσθήκη και αφαίρεση κόμβων από την τοπολογία. Επίσης για τον υπολογισμό των μονοπατιών είτε για αποκλειστική διανομή είτε για πολυδιανομή χρησιμοποιείται *Αλγόριθμος του Dijkstra* όπου σαν μετρική έχει οριστεί ο αριθμός των κόμβων και όχι των βαρών ανάμεσα στους κόμβους. Υπάρχει η δυνατότητα χρησιμοποίησης διαφορετικού αλγορίθμου για την εύρεση συντομότερων μονοπατιών, όμως επιλέχθηκε ο αλγόριθμος Dijkstra καθώς θεωρείτο ότι τα μονοπάτια έχουν μόνο θετικά βάρη και μπορεί να διαφέρουν από κόμβο σε κόμβο. Για την πειραματική αποτίμηση ωστόσο τα μονοπάτια έχουν βάρος 1 που αντικατοπτρίζει το κόστος για μετάβαση από ένα κόμβο σε έναν άλλο. Έτσι το κόστος που μετράει ο αλγόριθμος Dijkstra είναι ο αριθμός των μεταπηδήσεων (hops) για την μετάβαση από έναν κόμβο σε έναν άλλο.

Ο αλγόριθμος του Dijkstra που χρησιμοποιήθηκε στο πρόγραμμα παρέχεται από την Βιβλιοθήκη Boost. Στην συγκεκριμένη υλοποίηση ο αλγόριθμος αυτός επιλύει το πρόβλημα των συντομότερων μονοπατιών σε ένα κατευθυνόμενο με βάρη γράφο, που έχει μια αφετηρία και πολλούς προορισμούς. Στην έκδοση 1.57.0 η βιβλιοθήκη παρέχει δυο κυρίως επιλογές για την χρήση του αλγορίθμου. Η πρώτη επιλογή είναι ο υπολογισμός του κόμβου αφετηρίας προς κάθε κόμβο του γράφου με βάση την δομή αποστάσεων που παρέχεται. Η δεύτερη επιλογή είναι ο υπολογισμός του κόμβου αφετηρίας προς κάθε κόμβο του γράφου με βάση την δομή προκατόχων (για κάθε κόμβο u του συνόλου των κόμβων, θα υπάρχει προκάτοχός του p[u] αν ο u βρίσκεται στο δέντρο συντομότερων μονοπατιών του). Έτσι δίνεται η δυνατότητα να διατηρούνται τα μονοπάτια σε μια δομή ώστε να μην χρειάζεται ο επαναυπολογισμός τους. Επίσης η βιβλιοθήκη παρέχει τη δυνατότητα αλλαγής του τρόπου «επίσκεψης» των μονοπατιών του δέντρου με χρήση προσαρμοσμένου υποπρογράμματος κατά τις ανάγκες του χρήστη. Στην συγκεκριμένη πειραματική αποτίμηση χρησιμοποιήθηκε η πρώτη επιλογή, υπολογισμός του συντομότερου μονοπατιού με χρήση δομής αποστάσεων ώστε να υπολογίζεται κάθε φορά το μονοπάτι για κάθε κόμβο που χρησιμοποιείται ως αφετηρία. Έτσι υπολογίζεται η χειρότερη περίπτωση, όπου υπολογίζονται πάντα όλα τα μονοπάτια του δέντρου από την αρχή.

Το πληροφοριακό περιεχόμενο του προγράμματος είναι η διαδικτυακή τηλεόραση. Ένα είδος πληροφορίας που βασίζεται για την διανομή της καθαρά σε πολυδιανομή. Η πηγή μετάδοσης της διαδικτυακής τηλεόρασης είναι ο κόμβος που δημοσιεύει την πληροφορία και οι συνδρομητές είναι οι κόμβοι που ζητούν να εγγραφούν στο κανάλι για να τους μεταδοθεί η πληροφορία. Για απλοποίηση το πρόγραμμα δεν στέλνει τελικά πακέτα που να περιέχουν διαδικτυακή ροή τηλεόρασης καθώς το πείραμα εστιάζει στην απόδοση του συστήματος πάνω σε μια τέτοια αρχιτεκτονική και μάλιστα σε σχέση με την πολυδιανομή. Συγκεκριμένα ο εξυπηρετητής μετά την καταχώρηση του αιτήματος ενός κόμβου, δεν αποστέλλει ούτε επιβεβαίωση ούτε και πληροφοριακό περιεχόμενο στον αιτούντα ώστε στο πείραμα ο εξυπηρετητής να απασχολείται κυρίως με την εξυπηρέτηση των αιτημάτων. Ουσιαστικά η πειραματική αποτίμηση περιορίζεται στην μέτρηση και στην αξιολόγηση της απόδοσης του συστήματος για την πιο χρονοβόρα και απαιτητική διαδικασία, τον υπολογισμό του δέντρου πολυδιανομής, σε περιπτώσεις μεγάλου φόρτου όπως όταν οι χρήστες αλλάζουν πολλές φορές κανάλια, zapping.

Οι κόμβοι που επιθυμούν να εγγραφούν σε κάποιο κανάλι αιτούνται την εγγραφή στο επιθυμητό κανάλι κατονομάζοντας το. Τα κανάλια είναι προκαθορισμένα εξαρχής, κατά την έναρξη της λειτουργίας του εξυπηρετητή, και είναι γνωστά στους κόμβους του δικτύου. Έτσι λοιπόν αναφέρονται ονομαστικά στα κανάλια που επιθυμούν. Αξίζει να αναφερθεί ότι δεν υπάρχει περιορισμός στον αριθμό των καναλιών που μπορεί να παρέχει ο εξυπηρετητής, καθώς η δομή που διατηρεί για τους εγγεγραμμένους κόμβους ανά κανάλι είναι πολύ μικρή σε απαιτήσεις μνήμης και ταχύτητας διαχείρισης.

Για την προσθήκη ή την αποχώρηση από το δίκτυο οι κόμβοι θα πρέπει να στείλουν μήνυμα για προσθήκη και διαγραφή αντίστοιχα. Για εγγραφή σε κάποιο κανάλι οι κόμβοι πρέπει να στείλουν μήνυμα για εγγραφή κατονομάζοντας το κανάλι. Το ίδιο ισχύει για την απεγγραφή τους. Δεν υπάρχει κάποιος περιορισμός στον αριθμό των εγγεγραμμένων σε ένα κανάλι ούτε στον αριθμό των κόμβων που μπορούν να υπάρξουν στο

δίκτυο. Κατά την εγγραφή ο εξυπηρετητής ενημερώνει τη δομή που διατηρεί τους χρήστες με την προσθήκη του κόμβου στο επιθυμητό κανάλι και υπολογίζει τα συντομότερα μονοπάτια για κάθε εγγεγραμμένο κόμβο για το κανάλι που ζητήθηκε. Η ένωση των μονοπατιών αυτών αποτελεί και το συντομότερο δέντρο πολυδιανομής. Στην περίπτωση απεγγραφής, ο εξυπηρετητής ενημερώνει τη δομή που διατηρεί τους χρήστες με την αφαίρεση του κόμβου στο επιθυμητό κανάλι υπολογίζει ξανά τα συντομότερα μονοπάτια για κάθε εναπομείναντα εγγεγραμμένο κόμβο για το κανάλι που ζητήθηκε. Η ένωση των μονοπατιών αυτών αποτελεί και πάλι το συντομότερο δέντρο πολυδιανομής.

Η ένωση των μονοπατιών ώστε να προκύψει το δέντρο πολυδιανομής είναι μια λειτουργία που πραγματοποιείται με το τέλος του υπολογισμού όλων των επιμέρους μονοπατιών. Η διαδικασία αυτή πραγματοποιείται με πράξεις δυαδικής αριθμητικής όπως και στην τεχνική LIPSIN, και έχει σχετικά με σταθερό κόστος. Έτσι λοιπόν στην πειραματική αποτίμηση δεν λαμβάνεται υπόψιν ο χρόνος που θα πάρει στον εξυπηρετητή για την ένωση των επιμέρους μονοπατιών έτσι ώστε να προκύψει το δέντρο πολυδιανομής, καθώς έχει μικρή και σταθερή διάρκεια.

Αξίζει να σημειωθεί ότι στην αρχιτεκτονική αυτή γίνεται χρήση των φίλτρων Bloom. Αυτό έχει σαν αποτέλεσμα ότι κατά την λειτουργία της απεγγραφής από το κανάλι πρέπει να επαναυπολογιστούν όλα τα μονοπάτια για τους εναπομείναντες κόμβους καθώς τα φίλτρα Bloom δεν υποστηρίζουν αφαίρεση μονοπατιού. Έτσι δεν είναι δυνατή η αποθήκευση των μονοπατιών που έχουν υπολογιστεί και η διαγραφή του μονοπατιού που πρέπει να αφαιρεθεί. Το πρόβλημα αυτό θα μπορούσε να ξεπεραστεί με counting Bloom filters αλλά απαιτούν πολύ παραπάνω μνήμη και έχουν περιορισμένη χωρητικότητα. Επίσης, όταν το δέντρο πολυδιανομής μεγαλώνει πολύ τα Bloom filters αρχίζουν να δυσκολεύονται και να δυσλειτουργούν. Ο λόγος είναι κυρίως λόγω του συμβιβασμού που γίνεται μεταξύ απαιτήσεων μνήμης και του ποσοστού για λανθασμένα θετικά (false positive). Εναλλακτικοί τρόποι επίλυσης του προβλήματος αυτού θα ήταν η χρήση multi-stage Bloom filters, Bloom filters relays κ.α., κατά την οποία θα διασπούσαν το δέντρο πολυδιανομής σε υποδέντρα. Όμως η χρήση τέτοιων τεχνικών απαιτεί ακριβή πληροφορία για τα κλαδιά που περιέχονται στο δέντρο, επομένως η αποθήκευση των επιμέρους μονοπατιών δεν ωφελεί. Συνοψίζοντας λοιπόν, ο εξυπηρετητής είναι αναγκασμένος να υπολογίζει ξανά και ξανά όλα τα μονοπάτια για να προκύψει το δέντρο πολυδιανομής σε κάθε αίτημα εγγραφής και απεγγραφής του κάθε κόμβου.

Η πειραματική αποτίμηση πραγματοποιήθηκε με βάση την αναφερθείσα ροή μηνυμάτων για τα μηνύματα εγγραφής σε κανάλι και διαγραφής από κανάλι. Δηλαδή στην περίπτωση της εγγραφής σε κανάλι, ο κόμβος αρχικοποιείται με το αναγνωριστικό του κόμβου που επιθυμεί, επιλέγει ένα από τα προκαθορισμένα κανάλια για εγγραφή και αποστέλλει σύγχρονα ένα μήνυμα με την κατάλληλη μορφή για το επιθυμητό κανάλι στην διεύθυνση του εξυπηρετητή (την οποία την γνωρίζει εκ των προτέρων) και περιμένει. Ο εξυπηρετητής λαμβάνει το μήνυμα το μήνυμα μέσω του UDP υποδοχέα που έχει ανοίξει για την λήψη των αιτημάτων και ασύγχρονα τα προωθεί στα εσωτερικά υποπρογράμματα για την επεξεργασία του μηνύματος. Αυτός ο τρόπος λειτουργίας είναι απαραίτητος καθώς προσομοιώνει μια κανονική λειτουργία εξυπηρετητή κατά την οποία έχει ένα νήμα απασχολημένο για την λήψη των μηνυμάτων και ένα νήμα για την εξυπηρέτησή τους. Με αυτόν τον τρόπο διασφαλίζεται η ελαχιστοποίηση της καθυστέρησης και η απρόσκοπτη λειτουργία κατά την λήψη μηνυμάτων. Μετά την λήψη του μηνύματος το άλλο νήμα αρχίζει την επεξεργασία του και αναλόγως πραγματοποιεί ενέργειες για την ικανοποίησή του. Αν το μήνυμα είναι μήνυμα εγγραφής ενημερώνεται η δομή που περιέχει τους εγγεγραμμένους κόμβους ανά κανάλι, εξάγονται τα αναγνωριστικά των κόμβων που είναι εγγεγραμμένοι στο αιτούμενο κανάλι και υπολογίζεται για κάθε ένα κόμβο το συντομότερο μονοπάτι από τον εξυπηρετητή προς το κόμβο. Αν το μήνυμα είναι διαγραφής, ενημερώνεται πάλι η δομή που περιέχει τους εγγεγραμμένους κόμβους ανά κανάλι, εξάγονται τα αναγνωριστικά των εναπομεινάντων κόμβων που είναι εγγεγραμμένοι στο αιτούμενο κανάλι και υπολογίζεται για κάθε ένα κόμβο το συντομότερο μονοπάτι από τον εξυπηρετητή προς αυτόν. Σε κάθε περίπτωση η λειτουργία του προγράμματος είναι εξομοίωση της πραγματικής λειτουργίας ενός εξυπηρετητή διαδικτυακής τηλεόρασης σε ένα δίκτυο ευρείας περιοχής με αρχιτεκτονική PSI.

Ανάλυση

Σκοπός λοιπόν της πειραματικής αποτίμησης είναι η μέτρηση του χρόνου που απαιτείται για τον υπολογισμό του δέντρου πολυδιανομής συντομότερων μονοπατιών. Για τον λόγο αυτό η μέθοδος main του IPTVClient προγράμματος τροποποιήθηκε ώστε να παράγει ένα προκαθορισμένο αριθμό αντικειμένων με βάση την παράμετρο που του δίνεται. Έτσι ο πρόγραμμα IPTVClient, διαβάζει την τοπολογία διαλέγει κάθε φορά 10 σετ n-άδων και εκτελεί αιτήματα εγγραφής για διαφορετικό κανάλι ανά σετ. Η n-άδα καθορίζεται από την παράμετρο n που περνιέται για να ορίσει τον αριθμό των δειγμάτων. Αναλυτικότερα σε κάθε πείραμα επιλέγεται ένας αριθμός n που αντιστοιχεί στο πλήθος των κόμβων που επιλέγονται. Έτσι θα επιλεχθούν κάθε φόρα 10 διαφορετικά σετ δυάδων κόμβων, που αντιστοιχούν στην τοπολογία, 10 σετ τριάδων, τετράδων κ.ο.κ για n ίσο με 2, 3, 4 κ.ο.κ. Αυτό θα έχει σαν αποτέλεσμα για κάθε n-αδα να πραγματοποιούνται αιτήματα μια φορά για κάθε κόμβο για το ίδιο κανάλι έτσι ώστε να υπολογισθούν τα συντομότερα μονοπάτια για κάθε n αιτήματα. Έτσι λοιπόν ο χρόνος που θα μετρηθεί θα είναι ο συνολικός χρόνος για 2 κόμβους επί 10 φορές, για 3 κόμβους επί 10 φορές κοκ για κάθε n ίσο με 2, 3, 4 κτλ.

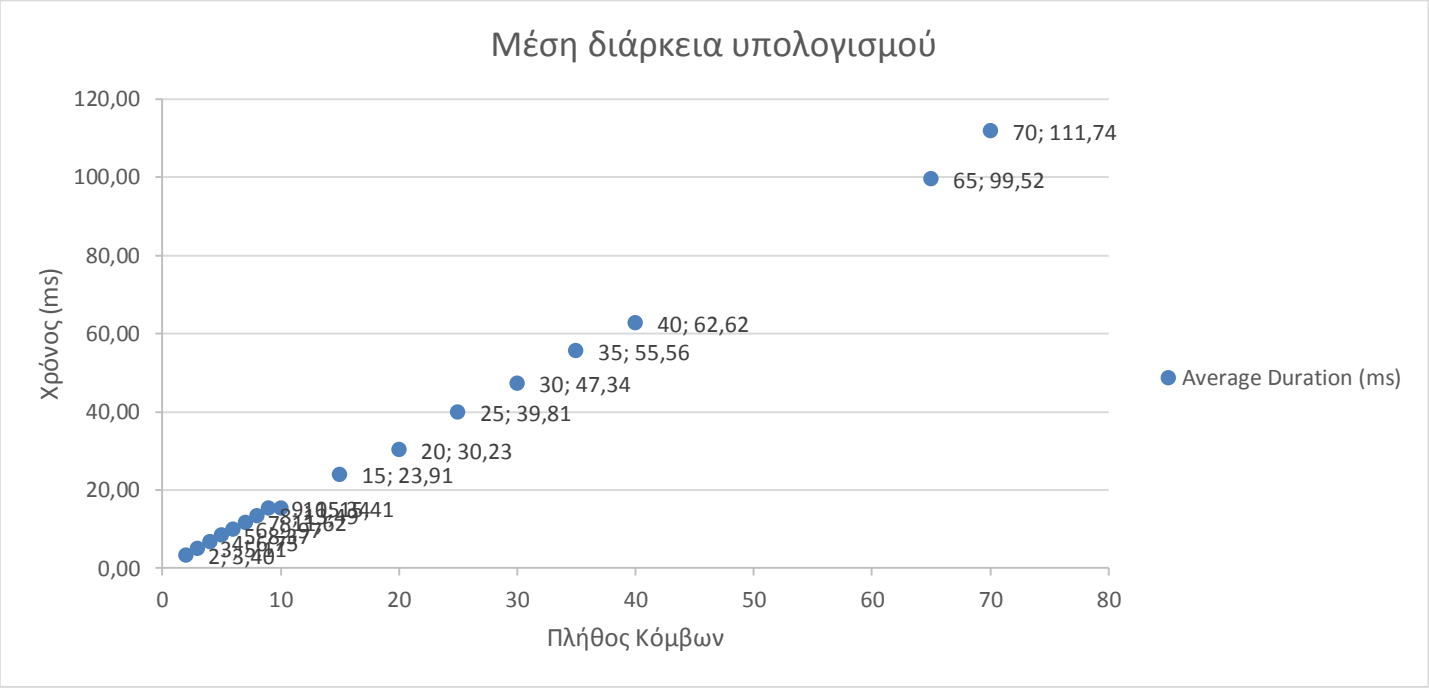
Από την πλευρά του εξυπηρετητή η μόνη αλλαγή ήταν στην τάξη Server του υποπρογράμματος IPTVServer, στην οποία άλλαξε το αντικείμενο που αρχικοποιεί για παραλαβή μηνυμάτων, από τύπου MessageHolder σε MessageHolderForStats. Η αλλαγή αυτή μπορεί να πραγματοποιηθεί μιας και τα αντικείμενα αντιστοιχούν σε τάξεις που κληρονομούν την IMessageHolder την οποία χρησιμοποιεί η τάξη Server. Αυτό έχει σαν αποτέλεσμα ο εξυπηρετητής να καταγράφει τους χρόνους που χρειάζεται για τον υπολογισμό συντομότερων μονοπατιών για κάθε n-αδα και να τους αποθηκεύει σε αρχείο, ώστε να είναι ευκολότερη η ανάλυση των αποτελεσμάτων. Η διάρκεια μέτρησης του χρόνου πραγματοποιήθηκε από την στιγμή πριν αρχίσει ο υπολογισμός του πρώτου στοιχείου της n-αδας μέχρι και την ολοκλήρωση του υπολογισμού για το τελευταίο στοιχείο. Η μέτρηση έγινε σε microseconds ώστε να φανεί η διαφορά στις μετρήσεις.

Στις μετρήσεις που πραγματοποιήθηκαν η επιλογή των κόμβων ήταν τυχαία και δεν επιλέχθηκαν με βάση την θέση τους στην τοπολογία, αλλά με βάση το πλήθος των κόμβων που διέθετε η τοπολογία και το πλήθος που είχε να διαλέξει για το δείγμα. Έτσι έγιναν μετρήσεις πάνω σε διαφορετικά σύνολα ώστε εκτιμηθεί πόσο περίπου χρόνο χρειάζεται ο εξυπηρετητής για τον υπολογισμό ενός επιπλέον κόμβου για την παραγωγή του δέντρου πολυδιανομής συντομότερων μονοπατιών.

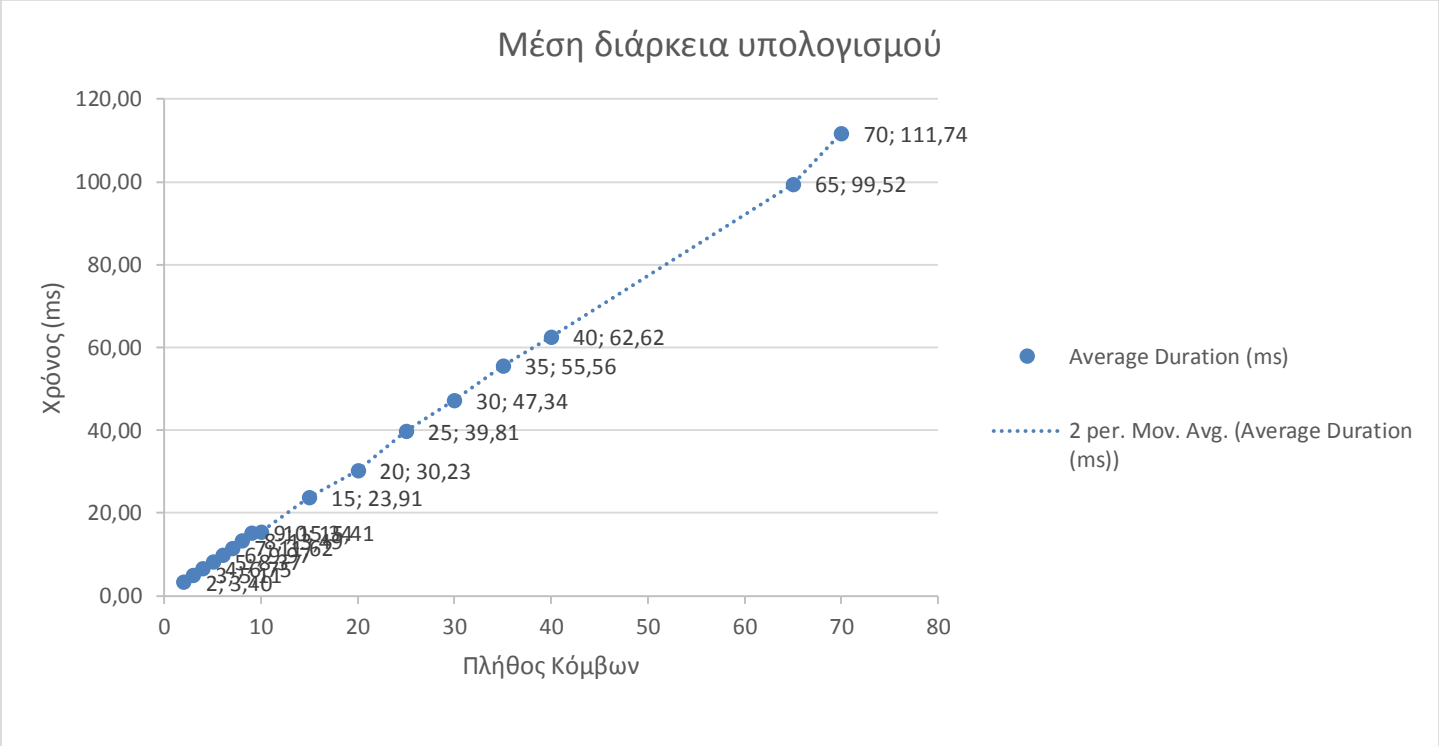
Με σκοπό την εύρεση της διάρκειας υπολογισμού των μονοπατιών χρησιμοποιήθηκαν δύο ειδών τοπολογίες, η τοπολογία Uninet και η τοπολογία 1221. Οι μετρήσεις έγιναν και στις δυο τοπολογίες με ίδιο αριθμό n-άδων ώστε να βγουν συμπεράσματα σχετικά με την συμπεριφορά του εξυπηρετητή στον υπολογισμό μονοπατιών.

Τοπολογία Uninet

Η τοπολογία αυτή αποτελείται από 74 κόμβους, με αναγνωριστικά από το 0 έως και το 73, με τον κόμβο εξυπηρετητή να αποτελεί το νούμερο 0. Από τον πίνακα 1, προκύπτει ότι η μέση διάρκεια υπολογισμού για 2 κόμβους στην τοπολογία αυτή είναι 3,4 ms. Για 5 κόμβους είναι 8,37,για 8 κόμβους 13,49, για 9 κόμβους 15,34 και για 10 15,41 ms. Με βάση αυτά τα νούμερα αναμένεται λοιπόν ότι η μέση διάρκεια υπολογισμού για 1 επιπλέον κόμβο αυξάνει την διάρκεια υπολογισμού κατά περίπου 1,7 ms. Αντίστοιχα, για 15 κόμβους η μέση διάρκεια υπολογισμού είναι 23,91 ms, για 20 κόμβους 39,81, για 30 47,34 και για 70 111, 74. Από τους μέσους όρους αυτούς προκύπτει ότι η μέση διάρκεια υπολογισμού για κάθε 1 επιπλέον κόμβο είναι περίπου 1,9 ms και ανά 5 περίπου 7 ms. Σύμφωνα με τα στοιχεία αυτά αναμένεται η καμπύλη μέσης διάρκειας να είναι περίπου ευθεία, δηλαδή η μέση διάρκεια υπολογισμού είναι ανάλογη του. Πράγμα που αποδεικνύεται και από τον πίνακα 2.

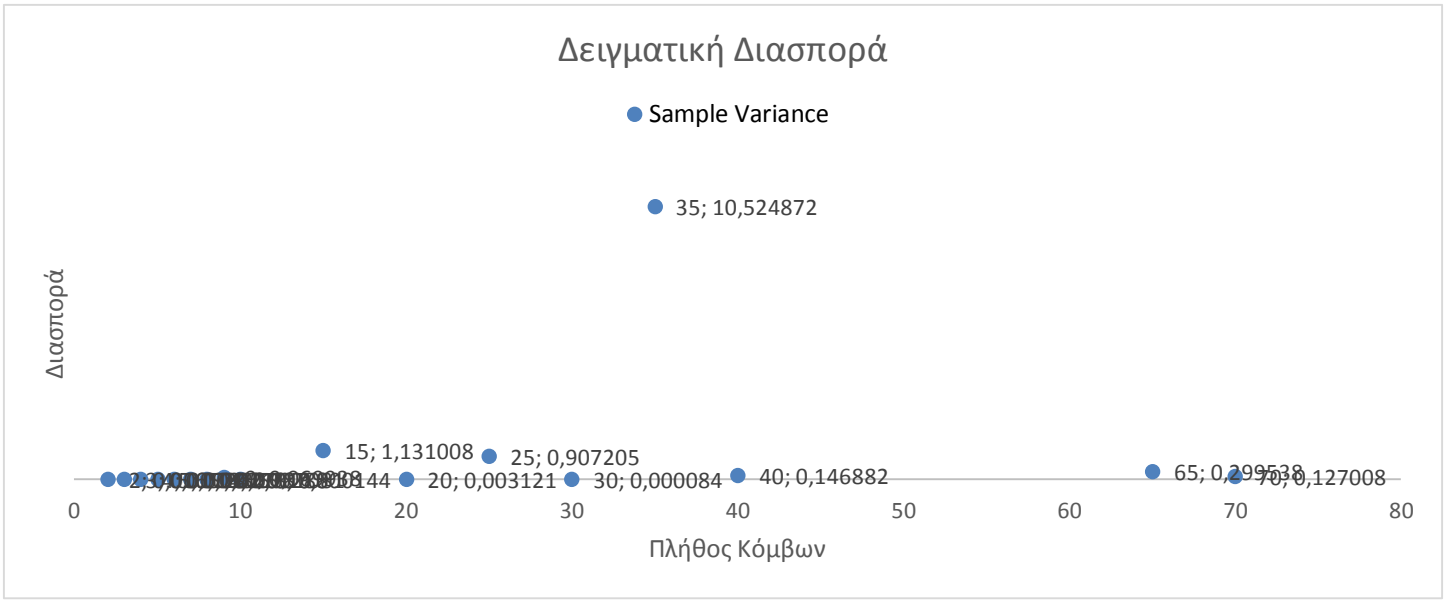


Πίνακας 1, Μέση διάρκεια υπολογισμού μονοπατιών

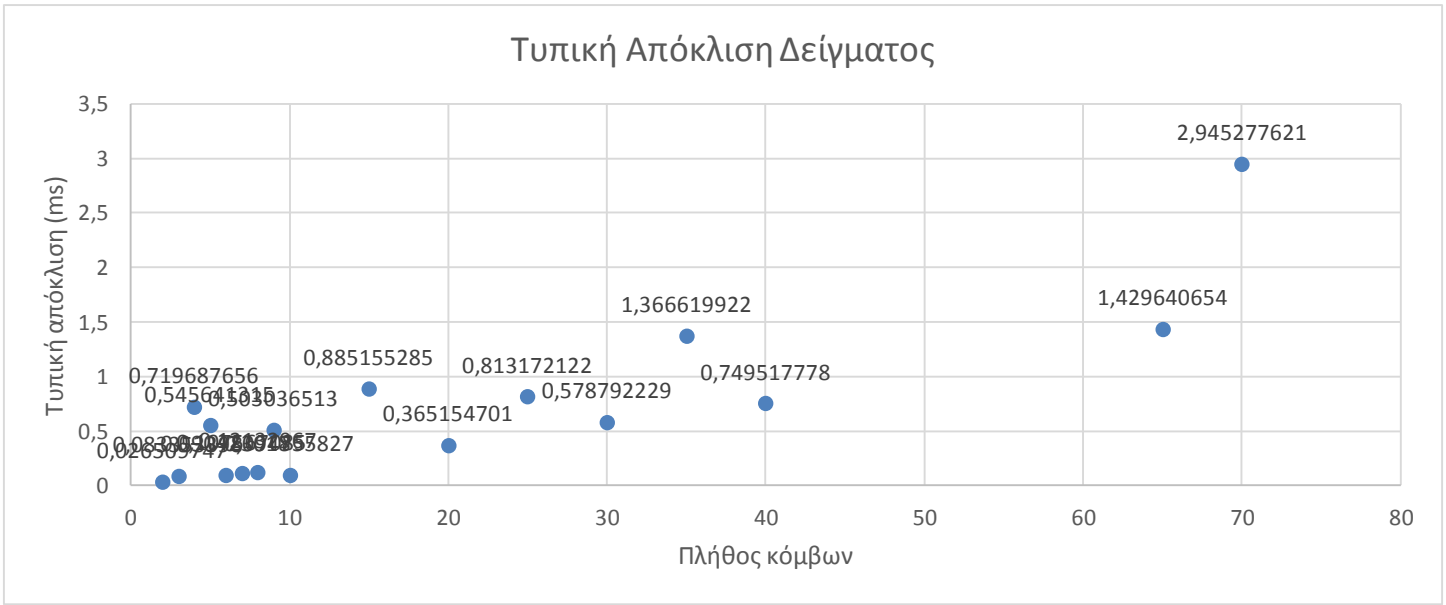


Πίνακας 2, Τάση μέσης διάρκεια υπολογισμού μονοπατιών

Επειδή τα πειράματα είναι δειγματοληπτικά είναι απαραίτητος ο υπολογισμός της διασποράς των δειγμάτων, ώστε να φανεί η απόκλιση που αναμένεται στις πραγματικές τιμές σε σχέση με τον μέσο όρο. Ο υπολογισμός της δειγματικής διασποράς σύμφωνα με τον πίνακα 3, δείχνει ότι οι τιμές που αναμένονται για κάθε επόμενη μέτρηση είναι περίπου ίδιες με απόκλιση από 0,000242 μέχρι 1,131008 ms. Η μόνη περίεργη τιμή της διασποράς είναι στο δείγμα με τους 35 κόμβους όπου η διασπορά είναι μεγαλύτερη, αλλά κρίνοντας από τους κόμβους που επιλέχθηκαν η διακύμανση οφείλεται ότι επιλέχθηκαν πολλοί κόμβοι που ήταν πιο απομακρυσμένα από τον εξυπηρετητή με αποτέλεσμα να επηρεαστεί ο χρόνος όπως ήταν λογικό. Παρόλα αυτά οι χρόνοι θεωρούνται αρκετά κοντά μεταξύ τους.



Πίνακας 3, Δειγματική Διασπορά



Πίνακας 4, Τυπική Απόκλιση Δείγματος

Τέλος ο πίνακας 4 παρουσιάζει την τυπική απόκλιση των δειγμάτων η οποία κυμαίνεται από 0,0265 μέχρι 2,9452 ms. Η τυπική απόκλιση αυξάνει όσο οι διαφορές μεταξύ των δειγμάτων είναι μεγαλύτερες.

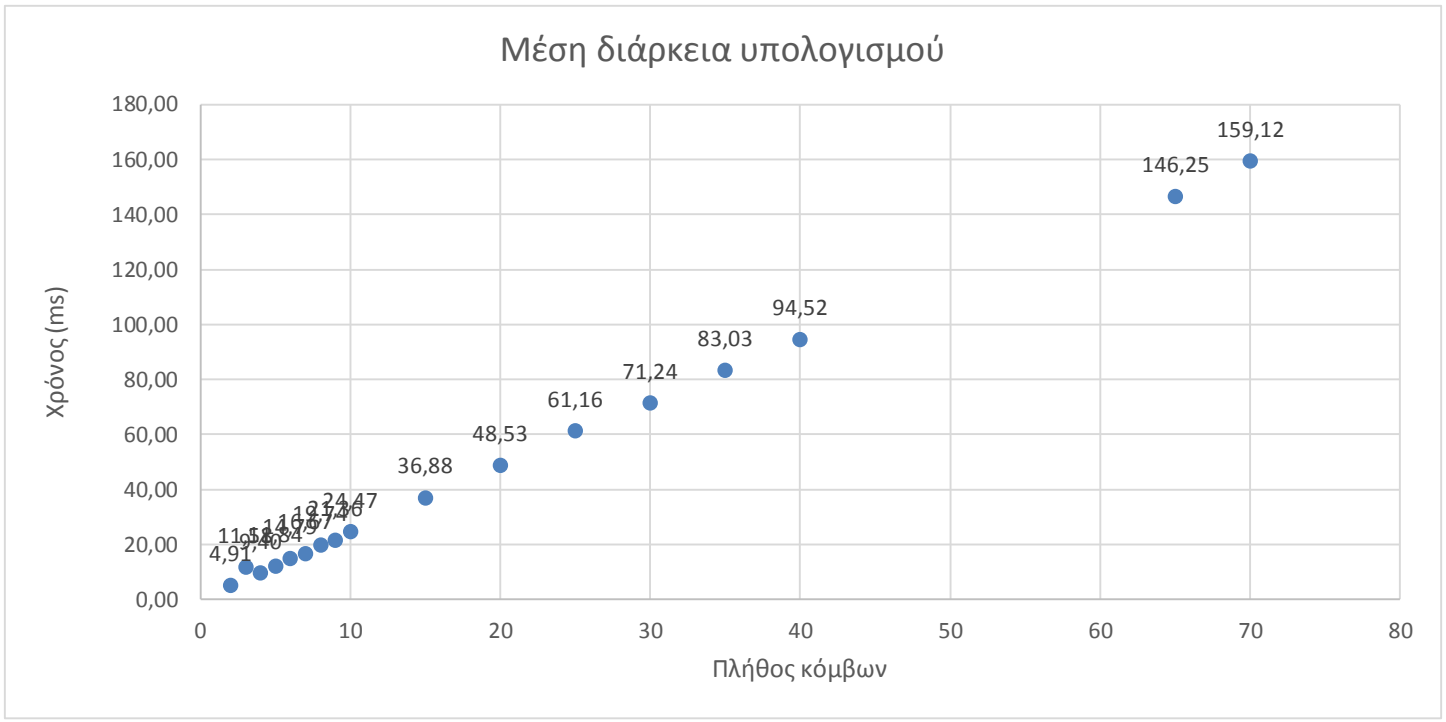
Τοπολογία 1221

Η τοπολογία αυτή αποτελείται από 104 κόμβους, με αναγνωριστικά από το 1 έως και το 108 με μερικούς αριθμούς να λείπουν. Ο κόμβος εξυπηρετητή στην περίπτωση αυτή είναι το νούμερο 1. Στον πίνακα 5 παρουσιάζεται η μέση διάρκεια υπολογισμού για το ίδιο πλήθος κόμβων όπως και στην τοπολογία Uninent. Εδώ προκύπτει ότι για 2 κόμβους είναι 4,91 ms, για 5 κόμβους είναι 11,84, για 8 κόμβους 19,74, για 9 κόμβους 21,36 και για 10 24,47 ms. Με βάση αυτά τα νούμερα αναμένεται λοιπόν ότι η μέση διάρκεια υπολογισμού για 1 επιπλέον κόμβο αυξάνει την διάρκεια υπολογισμού κατά περίπου 2,5 ms. Έτσι σε σχέση με

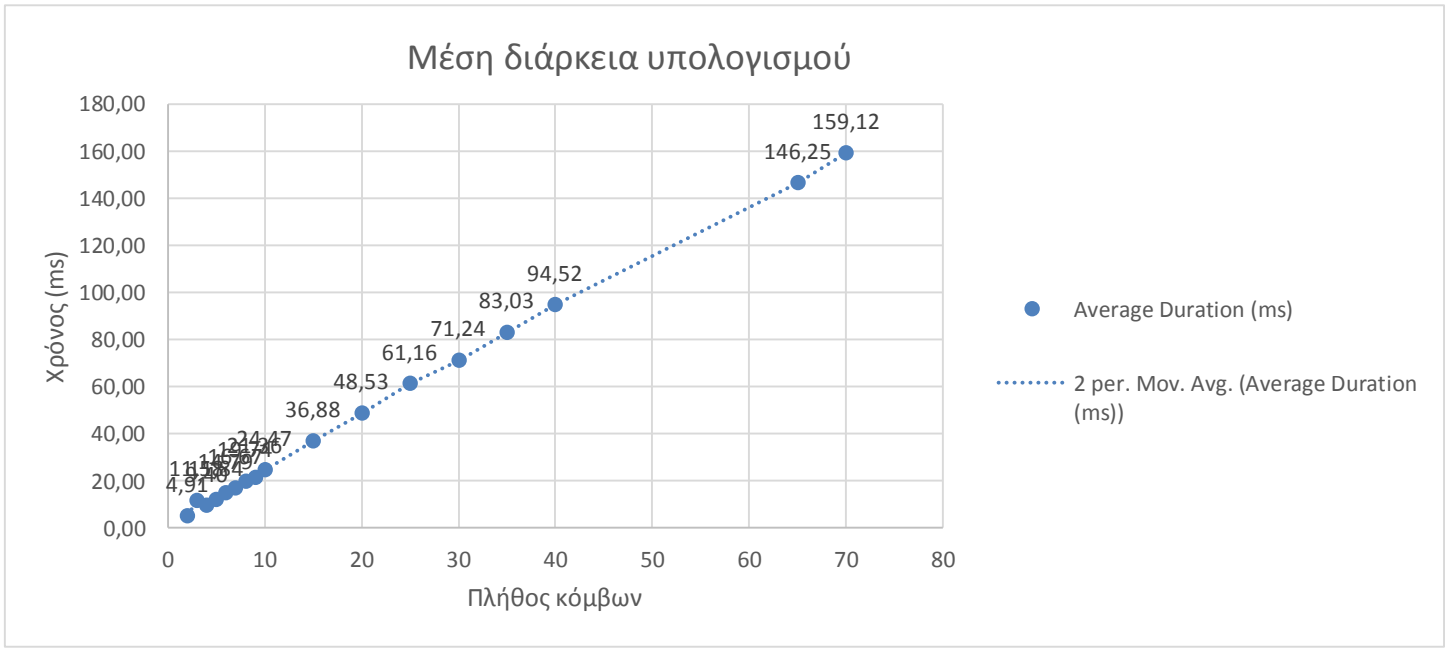
την τοπολογία Uninet παρατηρείται μια αύξηση στην μέση διάρκεια στους 10 κόμβους περίπου 10 ms και μια αύξηση της διάρκειας υπολογισμού για κάθε ένα επιπλέον κόμβο κατά 0,9 ms.

Αυξάνοντας τις 10άδες σε υπολογισμό μονοπατιών για 15 κόμβους, η μέση διάρκεια υπολογισμού ανέρχεται στα 36,88 ms, για 20 κόμβους σε 48,53, για 30 σε 71,24 και για 70 σε 159,12 ms. Από τους μέσους όρους αυτούς προκύπτει ότι η μέση διάρκεια υπολογισμού για κάθε ένα επιπλέον κόμβο είναι περίπου 2,45 ms και ανά 5 περίπου 12,5 ms. Με βάση τα παραπάνω η καμπύλη μέσης διάρκειας αναμένεται να είναι και πάλι περίπου ευθεία, το οποίο αποδεικνύεται και στον πίνακα 6.

Με βάση τα στοιχεία αυτά προκύπτει το συμπέρασμα ότι ακόμα και σε μεγαλύτερη τοπολογία η διάρκεια υπολογισμού μονοπατιών σε σχέση με τον αριθμό των κόμβων είναι αναλογική και κυμαίνεται περίπου στα 2 με 2,5 ms για κάθε επιπλέον κόμβο. Ακόμα και σε διαφορετικού είδους τοπολογίες η μέση διάρκεια υπολογισμού κυμάνθηκε από τα 111,74 στα 159,12 ms για 70 κόμβους, δίνοντας ένα περίπου σταθερό διάστημα διακύμανσης της διάρκειας. Έτσι είναι εφικτός ο υπολογισμός των ορίων που θα διακυμανθεί η μέση διάρκεια υπολογισμού μονοπατιών για δεδομένο αριθμό κόμβων με δεδομένη μια μεγαλύτερη τοπολογία. Όμως είναι εφικτός και ο υπολογισμός των ορίων διακύμανσης της διάρκειας ανεξαρτήτως τοπολογίας, συγκρίνοντας τα στοιχεία των δυο τοπολογιών.



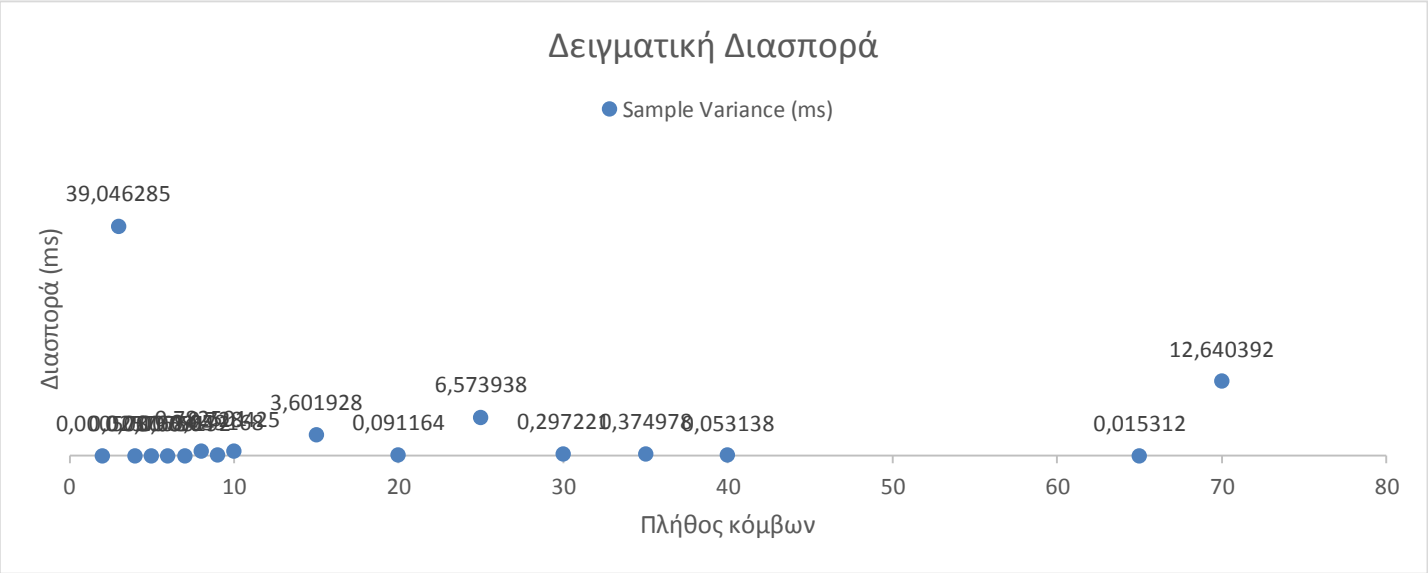
Πίνακας 5,Μέση διάρκεια υπολογισμού μονοπατιών



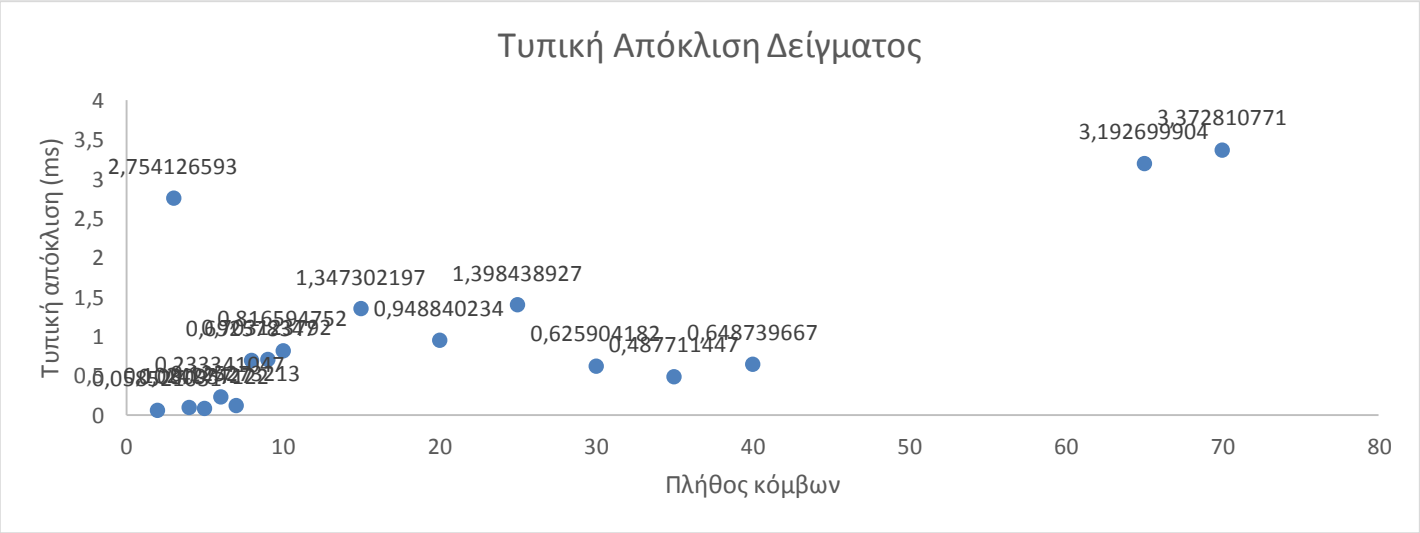
Πίνακας 6, Τάση μέσης διάρκεια υπολογισμού μονοπατιών

Όπως και στην τοπολογία Uninet, και εδώ τα πειράματα είναι δειγματοληπτικά οπότε είναι απαραίτητος ο υπολογισμός της διασποράς των δειγμάτων, ώστε να φανεί η απόκλιση που αναμένεται στις πραγματικές τιμές σε σχέση με τον μέσο όρο. Ο υπολογισμός της δειγματικής διασποράς σύμφωνα με τον πίνακα 7, δείχνει ότι οι τιμές που αναμένονται για κάθε επόμενη μέτρηση είναι περίπου ίδιες με απόκλιση από 0,000578 μέχρι 6,573938 ms. Και στην τοπολογία αυτή έχουμε μερικές τιμές σχετικά υψηλές όπως στην περίπτωση του υπολογισμού για 3 κόμβους με 39,046285 ms και για 70 με 12,640392 που οφείλονται όπως είναι λογικό στην επιλογή των κόμβων είτε σε πολύ κοντινούς ή σε πολύ μακρινούς σε σχέση με τον κόμβο εξυπηρετητή. Τέλος

ο πίνακας 8 παρουσιάζει την τυπική απόκλιση των δειγμάτων η οποία κυμαίνεται από 0,058521031 μέχρι 3,372810771 ms. Οι τιμές αυτές είναι λογικό να διαφέρουν σε σχέση με την προηγούμενη τοπολογία και να είναι αυξημένες, καθώς σε δείγματα μεγάλου αριθμού κόμβων όπως 65 και 70, οι επιλογές κόμβων είναι περισσότερες, ειδικά σε τοπολογίες με περισσότερους κόμβους, με αποτέλεσμα να κυμαίνονται και οι αποστάσεις αλλά και να υπάρχουν και ακραίες τιμές. Παρόλα αυτά οι τιμές αυτές είναι αρκετά μικρές πράγμα που δείχνει ότι το σύνολο είναι περίπου ομοιόμορφα κατανεμημένο.



Πίνακας 7, Δειγματική διασπορά



Πίνακας 8, Τυπική απόκλιση δείγματος

Συμπεράσματα

Με τα πειράματα που διεξήχθησαν αναλύθηκε η συμπεριφορά που έχει ο εξυπηρετητής σε διαφορετικά σύνολα κόμβων. Στα πειράματα αυτά επιλέχθηκε η μέτρηση του χρόνου υπολογισμού των συντομότερων μονοπατιών για όλους τους κόμβους που συμμετέχουν σε ένα κανάλι διότι ο χρόνος αυτός είναι ο σημαντικότερος από άποψη καθυστέρησης για τους κόμβους που αιτούνται εγγραφή σε ένα κανάλι. Στην αρχιτεκτονική PSI ο κόμβος που στέλνει το αίτημα θα έχει καθυστέρηση συνολικά το άθροισμα του χρόνου που χρειάζεται ο κόμβος προώθησης για την προώθηση του μηνύματος στον κόμβο συνάντησης, συν το χρόνο που χρειάζεται ο κόμβος συνάντησης να βρει στον κατανεμημένο πίνακα συσχετίσεων τον κόμβο που δημοσιεύει το κανάλι που ζητήθηκε, να τον εγγράψει και να προωθήσει ένα μήνυμα στον κόμβο που είναι υπεύθυνος για την τοπολογία. Στο άθροισμα αυτό θα προστεθεί και ο χρόνος που θα χρειαστεί ο κόμβος που είναι υπεύθυνος για την τοπολογία να υπολογίσει όλα τα συντομότερα μονοπάτια, και ο χρόνος που θα χρειαστεί ο κόμβος που παρέχει το κανάλι για την συμπερίληψη του αναγνωριστικού του νέου κόμβου στην μετάδοση της ροής δεδομένων. Ο χρόνος που χρειάζεται η μετάδοση δεν λαμβάνεται στα υπόψιν καθώς εξαρτάται αποκλειστικά στην τοπολογία και είναι ο ίδιος είτε χρησιμοποιηθεί αρχιτεκτονική PSI είτε χρησιμοποιηθεί η υπάρχουσα αρχιτεκτονική. Στο άθροισμα αυτό λοιπόν την μεγαλύτερη καθυστέρηση θα την παρουσιάσει το βήμα για τον υπολογισμό των συντομότερων μονοπατιών μιας και τα υπόλοιπα βήματα μπορούν να κυμανθούν σε σταθερά πλαίσια καθυστέρησης.

Ουσιαστικά η καθυστέρηση αυτή είναι η αναμονή ενός χρήστη κατά την διάρκεια αλλαγή καναλιού. Αν λοιπόν συγκεκριμενοποιηθούν τα όρια στα οποία η καθυστέρηση είναι ανεκτή από τον χρήστη μπορεί να υπολογιστεί και η συμπεριφορά του συστήματος. Αυτό μπορεί να επιτευχθεί υπολογίζοντας τις συνολικές καθυστερήσεις που θα συμβούν στο δίκτυο πολυδιανομής σύμφωνα με την αρχιτεκτονική PSI. Ο εξυπηρετητής έδειξε με την γραμμική του συμπεριφορά, γραμμική σε σχέση με το πλήθος των κόμβων για τους οποίους υπολόγιζε συντομότερα μονοπάτια, ότι είναι δυνατή την πρόβλεψη των χρόνων που θα διαρκούσε ο υπολογισμός για ένα δεδομένο αριθμό κόμβων. Υπολογίζοντας λοιπόν και τις υπόλοιπες καθυστερήσεις θα μπορούσε να υπάρξει μια σχετική χρονική εγγύηση της απόδοσης του συστήματος.

Με δεδομένη την δομή του προγράμματος που αναφέρθηκε στις προηγούμενες ενότητες και με τα συμπεράσματα που προέκυψαν από την συμπεριφορά του προγράμματος, σύμφωνα με τα πειράματα, περαιτέρω βελτιώσεις θα βοηθούσαν στην μείωση ακόμα περισσότερο της καθυστέρησης για τον υπολογισμό των συντομότερων μονοπατιών και αύξησης της συνολικής απόδοσης της πολυδιανομής. Αρχικά θα μπορούσε να χωριστεί ένα δίκτυο σε περισσότερους από ένα εξυπηρετητές ώστε να κατανεμηθεί το φόρτο και να εγγυηθεί ο χρόνος απόκρισης του συστήματος, αντί να αυξηθεί η επεξεργαστική ισχύ του εξυπηρετητή. Επίσης μια βελτίωση στον εξυπηρετητή θα ήταν να υπολογίζει τα συντομότερα μονοπάτια από όλους σε όλους εκ των προτέρων, ώστε να μην χρειάζεται να επαναυπολογίζει μονοπάτια. Προς την κατεύθυνση αυτή θα ήταν και η διατήρηση στη μνήμη των συντομότερων μονοπατιών που ήδη έχουν υπολογιστεί από τον εξυπηρετητή ώστε στον επόμενο υπολογισμό του δέντρου πολυδιανομής να χρησιμοποιηθεί ήδη το αποτέλεσμα. Μια ακόμη μικρή βελτίωση θα ήταν να διασπαστούν οι κόμβοι, για τους οποίους χρειάζεται να υπολογιστούν μονοπάτια, σε ξένα σύνολα και να ανατεθούν σε ξεχωριστά νήματα για τον υπολογισμό των μονοπατιών. Αυτή η βελτίωση θα εκμεταλλευόταν το παράλληλο υπολογισμό, μειώνοντας την συνολική καθυστέρηση. Όλες αυτές οι βελτιώσεις είναι αντικείμενο ακαδημαϊκής έρευνας όπως και η ίδια η αρχιτεκτονική PSI. Παρόλα αυτά ο στόχος είναι κοινός και είναι η αλλαγή του τρόπου που σχεδιάζονται τα δίκτυα και λειτουργεί το διαδίκτυο με σκοπό την κάλυψη των αυξανόμενων αναγκών για εύρεση, ανάκτηση και σύνδεση της πληροφορίας.

Βιβλιογραφία

[1] [Ηλεκτρονικό]. Available: http://eprints.rclis.org/7244/1/Tsimpoglou_ICODL3.pdf..

[2] [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Content_delivery_network.

[3] [Ηλεκτρονικό]. Available: http://p2pfoundation.net/.

[4] [Ηλεκτρονικό]. Available: http://dl.acm.org/citation.cfm?id=550430.

[5] [Ηλεκτρονικό]. Available: https://technet.microsoft.com/en-us/library/dd197470%28v=ws.10%29.aspx.

[6] Chaney. [Ηλεκτρονικό]. Available: https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Flash_Crowd.html.

[7] «Cisco. (2011, June) Visual networking index: Forrecast and methodology, 2010-2015. White Paper.».

[8] [Ηλεκτρονικό]. Available: https://www.ietf.org/rfc/rfc2827.txt.

[9] [Ηλεκτρονικό]. Available: https://web.eecs.umich.edu/~zmao/eecs589/notes/lec5_6.pdf.

[10] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Information-centric networking: seeing the forest for the trees," in ACM Workshop on Hot Topics in Networks (HotNets), 2011..

[11] [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Denial-of-service_attack.

[12] [Ηλεκτρονικό]. Available: http://www.islab.demokritos.gr/gr/html/ptixiakes/kostas-aris_ptyxiakh/Phtml/ipsec.htm.

[13] [Ηλεκτρονικό]. Available: http://www.dnssec.net/.

[14] «D. Trossen and G. Parisis, "Designing and realizing an informationcentric Internet," IEEE Communications, vol. 50, no. 7, pp. 60–67, July 2012».

[15] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Communications of the ACM, vol. 13, no. 7, pp. 422–426, July 1970..

[16] «G. Xylomenos, X. Vasilakos, C. Tsilopoulos, V. A. Siris, and G. C.Polyzos, "Caching and mobility support in a publish-subscribe Internet architecture," IEEE Communications, vol. 50, no. 7, pp. 52–58, July 2012.».

[17] «N. Fotiou, K. Katsaros, G. Polyzos, M.Sarela, D. Trossen, and G. Xylomenos, "Handling mobility in future publish-subscribe informationcentric networks," Telecommunication Systems, to appear.».

[18] [Ηλεκτρονικό]. Available: http://el.wikipedia.org/wiki/Simula.

[19] [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Alexander_Stepanov.

[20] [Ηλεκτρονικό]. Available: https://www.gnu.org/gnu/thegnuproject.html.

[21] [Ηλεκτρονικό]. Available: www.boost.org.

[22] [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Boost_%28C%2B%2B_libraries%29#cite_note-1.

[23] [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Edsger_W._Dijkstra.

[24] [Ηλεκτρονικό]. Available: http://arch.icte.uowm.gr/courses/os/OC_lab-OS-10.pdf.

[25] [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Maximum_transmission_unit.

[26] [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/User_Datagram_Protocol.

[27] [Ηλεκτρονικό]. Available: http://www.cs.ucla.edu/classes/winter09/cs217/2009SIGCOMM_LIPSIN.pdf.

[28] [Ηλεκτρονικό]. Available: http://mathworld.wolfram.com/SteinerTree.html.

[29] «N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J.Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," Proc. ACM SIGCOMM, Seattle, USA, Aug. 2008».

Παραρτήματα

Η ιστορία της C++

Το 1979 ο Δανός επιστήμονας υπολογιστών, Bjarne Stroustrup, ξεκίνησε την δουλειά με ένα πρόγονο της C++ με όνομα «C με τάξεις». Το κίνητρο για την δημιουργία μιας νέας γλώσσας προερχόταν από την εμπειρία του στον προγραμματισμό κατά την διδακτορική του διατριβή. Ο Stroustrup ήθελε μια αποτελεσματική ευέλικτη γλώσσα, όπως και η προγραμματιστική γλώσσα C, η οποία όμως να είχε και χαρακτηριστικά προγραμματισμού υψηλού επιπέδου για την οργάνωση του προγράμματος. Όταν λοιπόν άρχισε να εργάζεται στα εργαστήρια της AT & T Bell Labs, είχε αναλάβει την ανάλυση του πυρήνα του UNIX σε σχέση με τα κατανεμημένα υπολογιστικά συστήματα. Έθεσε ως στόχο να ενισχύσει τη γλώσσα C με χαρακτηριστικά όπως στην Simula [18], που βοηθούσαν στην ανάπτυξη λογισμικού. Επέλεξε την C όμως επειδή ήταν γενικής χρήσης, γρήγορη, φορητή και χρησιμοποιούταν ευρέως. Μετά την C και τις επιρροές από την Simula, και άλλες γλώσσες επηρέασαν επίσης την C++, όπως η ALGOL 68, Ada, CLU και η ML.

Ο Stroustrup αρχικά, πρόσθεσε στην C, την έννοια της τάξης (ή κλάσης), την τάξη που κληρονομεί άλλη τάξη, τη ρητή δήλωση μεταβλητών και τις προκαθορισμένες τιμές ορισμάτων ονομάζοντας την νέα γλώσσα «C με τάξεις» και χρησιμοποιώντας τον μεταγλωττιστή που δημιούργησε, τον Cpre, για την μετατροπή σε C.

Το 1983, μετονομάστηκε η γλώσσα αυτή από «C με τάξεις» σε C++, λόγω του ότι το σύμβολο ++ είναι τελεστής επαύξησης στην C. Στη νέα γλώσσα προστέθηκαν και οι έννοιες των εικονικών (virtual) μεθόδων, της ονομασίας των μεθόδων, της υπερφόρτωσης τελεστών, των αναφορών, των σταθερών μεταβλητών, και των τελεστών για δυναμική κατανομή της μνήμης, που μαζί με την ανάπτυξη ενός βελτιωμένου μεταγλωττιστή για C++, τον Cfront, πρόσφεραν μια μεγάλη δυναμική στη γλώσσα.

Το 1985, η πρώτη έκδοση του προγραμματικής γλώσσας C++ έγινε διαθέσιμη, αλλά δεν έγινε ακόμα επίσημο πρότυπο, παρότι έγινε ο οριστικός οδηγός αναφοράς της γλώσσας. Η πρώτη εμπορική εφαρμογή της C++ κυκλοφόρησε τον Οκτώβριο του ίδιου έτους.

Η C++ λόγω των χαρακτηριστικών της έγινε τόσο δημοφιλής που πολλές άλλες σύγχρονες, προγραμματιστικές γλώσσες έχουν επηρεαστεί από αυτήν όπως είναι η C#, η Java, και νεότερες εκδόσεις της C (μετά το 1998).

Η ιστορία του μεταγλωττιστή GCC

Σε μια προσπάθεια για αρχικοποίηση του λειτουργικού συστήματος GNU, ο Richard Stallman ρώτησε τον Andrew S. Tanenbaum, δημιουργού του Amsterdam Compiler Kit (ή αλλιώς Free University Compiler Kit) αν μπορούσε να χρησιμοποιήσει αυτό το λογισμικό στο GNU. Ο Tanenbaum του απάντησε ότι ενώ το Free University Compiler Kit ήταν δωρεάν, ο μεταγλωττιστής του δεν ήταν και έτσι ο Stallman αποφάσισε να γράψει έναν δικό του. Το αρχικό σχέδιο του Stallman ήταν να ξαναγράψει, με την βοήθεια του Len Tower και άλλων, τον ήδη υπάρχον μεταγλωττιστή του Lawrence Livermore Laboratory για μετατροπή της Pastel σε C. Γράφοντας ένα καινούργιο C front end για τον μεταγλωττιστή Livermore compiler συνειδητοποίησε ότι θα χρειαστεί megabytes για μνήμη στοίβας, κάτι που δεν ήταν εφικτό στα συστήματα 68000 Unix που είχαν μόνο 64K. Αυτό τον οδήγησε στο να γράψει ένα καινούργιο μεταγλωττιστή από την αρχή αγνοώντας υπάρχοντα κώδικα μεταγλωττιστή για Pastel.

Το GCC αρχικά δημοσιεύτηκε στις 22 Μαρτίου το 1987 μέσω FTP από το MIT. Ο Stallman θεωρήθηκε σαν συγγραφέας του αλλά ανέφερε άλλους για τη συμβολή τους όπως ο Jack Davidson και ο Christopher Fraser για την ιδέα του για χρήση της RTL σαν ενδιάμεση γλώσσα.

Το 1991, όταν ο GCC 1.x είχε αρχίσει να σταθεροποιείται αλλά με περιορισμούς στην αρχιτεκτονική, το FSF ξεκίνησε την επόμενη έκδοση του GCC 2.x.

Καθώς ο GCC μπορούσε να χρησιμοποιηθεί υπό την άδεια GPL, οι προγραμματιστές μπορούσαν να εργαστούν προς άλλες κατευθύνσεις δημιουργώντας την δικιά τους έκδοση αρκεί να συμμορφώνονται στους όρους της GPL, συμπεριλαμβανομένου και της απαίτησης για διανομή του πηγαίου κώδικα. Πολλές από αυτές τις εκδόσεις βοήθησαν στη διάδοση του μεταγλωττιστή και άλλες χρησιμοποιήθηκαν σαν παραδείγματα.

Από τότε ο GCC συντηρείται από μια πληθώρα προγραμματιστών ανά τον κόσμο κάτω από τις οδηγίες της οργανωτικής επιτροπής και έχει μεταφερθεί σε περισσότερες αρχιτεκτονικές επεξεργαστών από κανένα άλλο μεταγλωττιστή.

Εκδόσεις βιβλιοθήκης Boost

Μερικές από τις πιο πρόσφατες εκδόσεις, με αντίστροφη χρονολογικά σειρά, είναι οι εξής:

*Έκδοση **1.57.0**, 3 Νοεμβρίου 2014 21:55 GMT*

Αναβαθμισμένες βιβλιοθήκες: Any, Asio, Circular Buffer, Config, Container, Coroutine, Flyweight, Geometry, Interprocess, Intrusive, Iterator, Lexical Cast, Math, Move, MultiArray, Multiprecision, Multi-Index Containers, Preprocessor, Thread, TypeIndex, TypeTraits, Units, Unordered, Utility, uBLAS.

*Έκδοση **1.56.0**, 7 Αυγούστου 2014 16:08 GMT*

Νέες βιβλιοθήκες: Align, TypeIndex.

Νέες λειτουργίες σε υπάρχον κώδικα: Assert, Core, Lexical_Cast, Throw_Exception.

Αναβαθμισμένες βιβλιοθήκες: Accumulators, Any, Asio, Assign, Atomic, Circular Buffer, Concept Check, Container, Context, Coroutine, Dynamic Bitset, Chrono, Flyweight, Fusion, Geometry, Hash, Interprocess, Intrusive, Log, Math, Move, MPL, MultiArray, Multi-index Containers, Multiprecision, Odeint, Optional, Predef, Preprocessor, Program Options, Regex, Smart Pointers, Thread, TTI, Unordered, Utility, UUID, Variant.

Παρωχημένες Βιβλιοθήκες: TR1

*Έκδοση **1.55.0**, 11 Νοεμβρίου 2013 19:50 GMT*

Νέες βιβλιοθήκες: Predef.

Αναβαθμισμένες βιβλιοθήκες: Accumulators, Any, Asio, Atomic, Config, Chrono, Circular Buffer, Container, Context, Coroutine, Filesystem, Fusion, Geometry, Graph, Hash, Interprocess, Intrusive, Lexical Cast, Log, Math, Meta State Machine, Move, Multiprecision, Multi-index Containers, MPI, Phoenix, Polygon, PropertyMap, Rational, Thread, Timer, Type Traits, Unordered, Utility, Variant, Wave, xpressive.

*Έκδοση **1.54.0**, 1 Ιουλίου 2013 17:10 GMT*

Αλλαγές και σε υποστηριζόμενους επεξεργαστές.

Νέες βιβλιοθήκες: Log, TTI, Type Erasure.

Αναβαθμισμένες βιβλιοθήκες: Accumulators, Algorithm, Any, Asio, Chrono, Circular Buffer, Container, Context, Coroutine, Geometry, Graph, Interprocess, Intrusive, Iostreams, Lexical Cast, Math, Meta State Machine, Move, Multiprecision, Polygon, Property Map, Range, Thread, Type Traits, uBLAS, Unordered, Utility, Variant, Wave, xpressive

Παρωχημένες Βιβλιοθήκες: Signals.

*Έκδοση **1.53.0**, 4 Φεβρουαρίου 2013 18:29 GMT*
Νέες βιβλιοθήκες: Atomic, Coroutine, Lockfree, Multiprecision, Odeint.

Αναβαθμισμένες βιβλιοθήκες: Algorithm, Array, Asio, Bimap, Chrono, Container, Context, Geometry, GIL, Graph, Hash, Interprocess, Intrusive, Lexical Cast, Locale, Math, MinMax, Move, Polygon, Random, Range, Ratio, Regex, Smart Pointers, StringAlgo, Thread, Utility, Unordered, Variant, Wave, xpressive.

*Έκδοση **1.52.0**, 5 Νοεμβρίου 2012 16:05 GMT*
Αναβαθμισμένες βιβλιοθήκες: Accumulators, Config, Chrono, Container, DateTime, Foreach, Function, Graph, Hash, Interprocess, Iterator, Lexical Cast, Math, Phoenix, Polygon, Proto, Ratio, Result_of, Thread, uBLAS, Unordered, Uuid, Wave, xpressive.

*Έκδοση **1.51.0**, 20 Αυγούστου 2012 23:00 GMT*
Νέες βιβλιοθήκες: Context.

Αναβαθμισμένες βιβλιοθήκες: Algorithm, Asio, Config, Chrono, Geometry, Graph, Hash, Lexical Cast, Math, MSM, Proto, Ratio, Regex, Thread, Unordered, Wave, xpressive

*Έκδοση **1.50.0**, 28 Ιουνίου 2012 12:48 GMT*
Νέες βιβλιοθήκες: Algorithm, Functional/OverloadedFunction, LocalFunction, Utility/IdentityType

Αναβαθμισμένες βιβλιοθήκες: Accumulators, Array, Asio, Bimap, Chrono, Concept Check, Filesystem, Foreach, Graph, Geometry, Hash, Iostreams, Iterator, MultiArray, Lexical cast, Locale, MSM, Program Options, PropertyMap, Proto, Ratio, ScopeExit, Thread, Unordered, Wave, xpressive