Ατομική Διπλωματική Εργασία

ΠΟΛΙΤΙΚΕΣ ΔΙΑΧΕΙΡΙΣΗΣ ΔΕΔΟΜΕΝΩΝ ΣΕ ΔΙΚΤΥΑ ΠΑΡΑΔΟΣΗΣ ΠΕΡΙΕΧΟΜΕΝΟΥ

Θεόδωρος Δημητρίου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2011

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΟΛΙΤΙΚΕΣ ΔΙΑΧΕΙΡΙΣΗΣ ΔΕΔΟΜΕΝΩΝ ΣΕ ΔΙΚΤΥΑ ΠΑΡΑΔΟΣΗΣ ΠΕΡΙΕΧΟΜΕΝΟΥ

Θεόδωρος Δημητρίου

Επιβλέπων Καθηγητής
Δρ. Γεώργιος Πάλλης

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

Μάιος 2011

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή αυτής της διπλωματικής εργασίας, τον Λέκτορα Δρ. Γεώργιο Πάλλη, ο οποίος μου έδωσε την ευκαιρία να ασχοληθώ με την παρούσα εργασία και που με την καθοδήγηση και τη βοήθεια του κατάφερα να τη φέρω εις πέρας.

Θα ήθελα να ευχαριστήσω τον Δρ. Κωνσταντίνο Στάμο για την παραχώρηση του προσομοιωτή CDNSim στον οποίο έτρεξα τα πειράματα μου καθώς επίσης και για τη βοήθεια που μου έδινε σε τυχόν απορίες που είχα για το συγκεκριμένο εργαλείο.

Θα ήθελα να ευχαριστήσω το φοιτητή Δημητρίου Νικόλα για την παραχώρηση των datasets που χρησιμοποίησα στα πειράματα μου.

Περίληψη

Αυτή η διπλωματική ασχολείται ερευνητικά στο πεδίο των Δικτύων Παράδοσης Περιεχομένου (Content Delivery Networks). Ένα Δίκτυο Παράδοσης Περιεχομένου είναι ένα δίκτυο από εξυπηρετητές που περιέχουν ενδιάμεσες μνήμες (cache) οι οποίοι ονομάζονται εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου και ανήκουν στον ίδιο Παροχέα Διαδικτύου. Το δίκτυο παραδίδει περιεχόμενο στους χρήστες και έχει ως στόχο να μειώσει το χρόνο καθυστέρησης που βιώνουν οι χρήστες καθώς και τη χρησιμοποίηση του δικτύου. Για να το πετύχει αυτό χρειάζεται να βρεθούν αλγόριθμοι οι οποίοι κάνουν τοποθέτηση του περιεχομένου στους εξυπηρετητές.

Σε αυτή τη διπλωματική προτείνεται ένας αλγόριθμος τοποθέτησης αντικειμένων ο οποίος βασίζεται στη μετρική utility (Κεφάλαιο 2) και γίνεται πειραματική μελέτη του αλγορίθμου και σύγκριση του με ένα υπάρχον αλγόριθμο.

Τα πειράματα τρέχουν στο CDNSim ο οποίος προσομοιώνει ένα Δίκτυο Παράδοσης Περιεχομένου.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο Διπλωματικής

Σήμερα στο χώρο του παγκόσμιου ιστού λόγω του μεγάλου όγκου δεδομένων που ζητούνται καθημερινά σχεδόν από κάθε περιοχή του κόσμου δημιουργήθηκε η ανάγκη δικτύων ευρείας έκτασης τα οποία φυλάσσουν αντίγραφα αντικειμένων που αιτήθηκαν οι χρήστες με σκοπό την μεγιστοποίηση του εύρους ζώνης που χρησιμοποιούν. Αυτά ονομάζονται Δίκτυα Παράδοσης Περιεχομένου (Content Delivery Networks: CDN). Ένα από τα προβλήματα που αντιμετωπίζουν τα Δίκτυα Παράδοσης Περιεχομένου είναι αυτό της τοποθέτησης των αντικειμένων. Έχουν ήδη χρησιμοποιηθεί κάποιοι ευριστικοί μέθοδοι που κάνουν τοποθέτηση αντικειμένων με βάση την τυχαιότητα και τη δημοτικότητα. Η μέθοδος της τυχαίας τοποθέτησης, όπου τα αντικείμενα τοποθετούνται σε τυχαίους εξυπηρετητές Δικτύων Παράδοσης Περιεχομένου είναι απλοϊκή, μη επεκτάσιμη και δε στηρίζεται σε καμία μετρική για να παράξει τα αποτελέσματα της. Η μέθοδος που βασίζεται στην δημοτικότητα των αντικειμένων απαιτεί τη συλλογή στατιστικών στοιχείων. Αυτή η μέθοδος έχει τα εξής μειονεκτήματα: Η συλλογή αξιόπιστων στατιστικών για κάθε αντικείμενο απαιτεί μεγάλο χρονικό διάστημα. Δε θα είναι λειτουργήσιμη για νεοεισερχόμενους ιστιακούς χώρους εκτός και αν περάσει το χρονικό διάστημα. Επίσης όπως αναφέρεται στο [18] η δημοτικότητα του κάθε αντικειμένου μεταβάλλεται συνεχώς . Το [18] φέρει σαν παράδειγμα τις αιτήσεις των χρηστών στο Παγκόσμιο κύπελλο του 98, όπου μόνο 40% των αντικειμένων με ψηλή δημοτικότητα παρέμειναν και την επόμενη μέρα δημοφιλή.

Αυτή η διπλωματική έχει σαν κίνητρο να βοηθήσει στην έρευνα που ασχολείται με την τοποθέτηση αντικειμένων με την υλοποίηση ενός αλγορίθμου τοποθέτησης που είναι βασισμένος στη μετρική Utility[3]. Η μετρική Utility είναι η τιμή που εκφράζει τη σχέση μεταξύ του αριθμού των bytes που ανεβάζει (upload) ένας εξυπηρετητής έναντι του αριθμού των bytes που κατεβάζει (download). Αυτή η μετρική μας δίνει μια αίσθηση της χρησιμότητας του κάθε εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου και μπορεί επίσης να χρησιμοποιηθεί σαν παράμετρος για να καθορίσει τιμολόγηση μιας πολιτικής που θα χρησιμοποιήσει ένα Δίκτυο Παράδοσης Περιεχομένου. Όπως αναφέρει το άρθρο [3] η βελτίωση της μετρικής Utility θα βελτιώσει λειτουργίες που παρέχουν τα Δίκτυα Παράδοσης Περιεχομένου, όπως application acceleration, e-commerce, παράδοση δυναμικού περιεχομένου και εφαρμογές Web 2.0.

Άλλες προκλήσεις που αντιμετωπίζουν τα Δίκτυα Παράδοσης Περιεχομένου

Τα ποιο σημαντικά προβλήματα των Δικτύων Παράδοσης Περιεχομένου αφορούν τη διαχείριση περιεχομένου και είναι τα εξής:

***Πρόβλημα τοποθέτησης των εξυπηρετητών* Δικτύων Παράδοσης Περιεχομένου**

Για να μπορεί να μεταφερθεί περιεχόμενο στον τελικό χρήστη με κάποια ποιότητα υπηρεσίας (Quality of service) οι διαχειριστές πρέπει να εξασφαλίσουν ότι οι εξυπηρετητές Δικτύων Παράδοσης Περιεχομένου είναι τοποθετημένοι σε στρατηγικές θέσεις του διαδικτύου. Γενικά το πρόβλημα είναι να τοποθετηθούν N εξυπηρετητές Δικτύων Παράδοσης Περιεχομένου μεταξύ M διαφορετικών σημείων με τέτοιο τρόπο έτσι ώστε να έχουμε το μικρότερο κόστος. (Γνωστό και ως το πρόβλημα Minimum K-Median). Προηγούμενες εργασίες έχουν γίνει στο θέμα και έχουν προταθεί κάποιοι αλγόριθμοι όπως Greedy – τοποθέτηση αντικειμένων αυξητικά, Hotspot – τοποθέτηση κοντά στους χρήστες που δημιουργούν τον περισσότερο φόρτο [14], Tree-based – βασίζεται στην υπόθεση ότι οι τοπολογίες μπορούν να αναπαρασταθούν σαν δέντρα [16], HotZone – που βασίζεται στο latency [17]. Αυτοί οι αλγόριθμοι καθορίζουν που πρέπει να τοποθετηθούν οι εξυπηρετητές Δικτύων Παράδοσης Περιεχομένου για να πετύχουμε καλύτερη απόδοση.

Η διπλωματική ασχολείται με το πρόβλημα της τοποθέτησης εξυπηρετητών στα Δίκτυα Παράδοσης Περιεχομένου με τη δημιουργία ενός αλγορίθμου που περιγράφεται στο Κεφάλαιο 5 και έχει σαν κίνητρο να παράξει πιο ρεαλιστικά πειραματικά

αποτελέσματα. Αυτό επιτυγχάνεται με την τοποθέτηση των εξυπηρετητών Δικτύων Παράδοσης Περιεχομένου σε περιοχές της τοπολογίας του διαδικτύου που βασίζεται στο άρθρο [5] το οποίο περιέχει στοιχεία του πλήθους και της τοποθεσίας των εξυπηρετητών πραγματικών παρόχων των Δικτύων Παράδοσης Περιεχομένου.

Πρόβλημα επιλογής περιεχομένου

Πρέπει να καθορίσουμε ποιο περιεχόμενο πρέπει να φύγει από την πηγή, δηλαδή ποια αντικείμενα του Πηγαίου Εξυπηρετητή (Origin Server) θα ανατεθούν στους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου.

1.2 Συνεισφορά

Η διπλωματική συνεισφέρει ερευνητικά στον τομέα των Δικτύων Παράδοσης Περιεχομένου προσφέροντας τα εξής:

1) Ανάπτυξη και υλοποίηση αλγορίθμου που κάνει εναποθήκευση (caching) του περιεχομένου ενός διαδικτυακού τόπου σε εξυπηρετητές ενός Δικτύου Παράδοσης Περιεχομένου. Με δεδομένα τα περιεχόμενα του διαδικτυακού τόπου, τις αιτήσεις των χρηστών, και την συνδεσμολογία του δικτύου CDN ο αλγόριθμος βρίσκει μια τοποθέτηση (Si , Oj) όπου Si είναι ο i εξυπηρετητής Δικτύου Παράδοσης Περιεχομένου και Oj το αντικείμενο j του διαδικτυακού χώρου. Η τοποθέτηση βασίζεται στην μετρική Utility [3], η οποία μας δίνει τη χρησιμότητα του κάθε εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου.

2) Υλοποίηση εργαλείου που δημιουργεί ένα ρεαλιστικό Δίκτυο Παράδοσης Περιεχομένου. Το εργαλείο λαμβάνει ένα αρχείο που περιγράφει μια φυσική τοπολογία του διαδικτύου υπό μορφή γράφου και μια παράμετρο που ορίζει τον αριθμό των εξυπηρετητών του Δικτύου Παράδοσης Περιεχομένου. Ομαδοποιεί τους κόμβους της τοπολογίας και σε κάθε ομάδα συνδέει ένα συγκεκριμένο αριθμό από εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου. Ο αριθμός των εξυπηρετητών που θα τοποθετηθούν σε κάθε ομάδα περιγράφεται σαν το ποσοστό στο άρθρο [5].

3) Πειραματική μελέτη της απόδοσης του αλγορίθμου που περιγράφεται στο Κεφάλαιο 4 και σύγκριση του με τους 2 αλγορίθμους που αναφέρονται στο Κεφάλαιο 2. Για συγκεκριμένες τιμές ο αλγόριθμος του Κεφαλαίου 4 επιτυγχάνει καλύτερα αποτελέσματα από αυτόν του Κεφαλαίου 2 ως προς το Mean Surrogate Server Utility και Hit Ratio Percentage (περιγράφονται στο Κεφάλαιο 6)

Κεφάλαιο 2

Συναφείς Εργασίες

2.1 Αλγόριθμοι για τοποθέτηση αντικειμένων σε εξυπηρετητές Δικτύων Παράδοσης Περιεχομένου

2.2 Αλγόριθμος Lat-cdn [4]

Προτείνει ένα αλγόριθμο ο οποίος προσπαθεί να λύσει το πρόβλημα της τοποθέτησης αντικειμένων στους εξυπηρετητές Δικτύων Παράδοσης Περιεχομένου.

Η κύρια ιδέα του αλγορίθμου είναι να τοποθετηθούν τα αντικείμενα στους εξυπηρετητές του Δικτύου Παράδοσης Περιεχομένου λαμβάνοντας υπόψη το συνολικό χρόνο καθυστέρησης (latency) που προκαλούν στο δίκτυο. Ο κάθε εξυπηρετητής Δικτύου Παράδοσης Περιεχομένου διατηρεί μια ενδιάμεση μνήμη (cache memory) που αποθηκεύεται στο δίσκο. Όταν ένας εξυπηρετητής Δικτύου Παράδοσης Περιεχομένου λάβει ένα αίτημα για κάποιο αντικείμενο, εάν το έχει (cache hit) το εξυπηρετεί τοπικά ενώ σε αντίθετη περίπτωση (cache miss) παραπέμπει την αίτηση σε άλλο εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου ή στον Πηγαίο εξυπηρετητή. Ο αλγόριθμος κάνει την υπόθεση ότι οι εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου συνεργάζονται μεταξύ τους, δηλαδή ότι ο καθένας γνωρίζει από πριν το περιεχόμενο που βρίσκεται

στους υπόλοιπους εξυπηρετητές που ανήκουν στο ίδιο Δίκτυο Παράδοσης Περιεχομένου.

Αρχικά θεωρεί ότι όλα τα αντικείμενα είναι τοποθετημένα στον Πηγαίο Εξυπηρετητή και όλοι οι εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου είναι άδειοι. Για κάθε ένα αντικείμενο που πρόκειται να φύγει από τον Πηγαίο Εξυπηρετητή βρίσκει ποιος είναι ο ιδανικός εξυπηρετητής για να το τοποθετήσει. Αυτός ο αλγόριθμος θεωρεί ότι ο ιδανικός εξυπηρετητής είναι αυτός που ελαχιστοποιεί το χρόνο καθυστέρησης του δικτύου. Σε πρώτη φάση βρίσκει για κάθε αντικείμενο που πρόκειται να φύγει από τον Πηγαίο εξυπηρετητή το κόστος (το κόστος μετριέται σε μορφή latency) που θα δημιουργήσει αν αντιγραφεί στον εξυπηρετητή 1,2,3..,N. Μετά επιλέγει από τα ζευγάρια αντικείμενο – εξυπηρετητής που βρέθηκαν στο προηγούμενο βήμα αυτά που παράγουν το μεγαλύτερο χρόνο καθυστέρησης και τα τοποθετεί στους συγκεκριμένους εξυπηρετητές. Επαναλαμβάνει τη διαδικασία μέχρι όλοι οι εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου να γεμίσουν. Αυτό μπορεί να έχει σαν αποτέλεσμα ένα αντικείμενο να βρίσκεται σε πολλούς εξυπηρετητές, αλλά ένας εξυπηρετητής θα έχει το πολύ ένα αντίγραφο του αντικειμένου.

2.3 Αλγόριθμος il2p [1]

Το [1] προτείνει τον αλγόριθμο il2p ο οποίος προσπαθεί να λύσει το πρόβλημα της τοποθέτησης αντικειμένων στους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου. Θεωρεί ότι η πολιτική που χρησιμοποιεί το Δίκτυο Παράδοσης Περιεχομένου είναι cooperative-push based. Στο cooperative-push based το περιεχόμενο κινείται από τον Πηγαίο εξυπηρετητή προς τους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου και μετά αυτοί συνεργάζονται μεταξύ τους για να μειώσουν το κόστος που επιφέρει η αντιγραφή περιττού περιεχομένου στις ενδιάμεσες μνήμες τους. Αυτή η πολιτική αποδείχτηκε στο [7] ότι έχει τα καλύτερα αποτελέσματα. Η ίδια πολιτική επικοινωνίας των εξυπηρετητών χρησιμοποιείται και σε αυτή τη διπλωματική στον αλγόριθμό που περιγράφεται στο κεφάλαιο 4.

Το άρθρο θεωρεί ένα δίκτυο με N εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου, και 1 Πηγαίο εξυπηρετητή στον οποίο φυλάσσεται το περιεχόμενο ενός διαδικτυακού

τόπου. Αρχικά όλα τα αντικείμενα είναι τοποθετημένα στον Πηγαίο εξυπηρετητή. Χωρίζει το πρόβλημα της τοποθέτησης αντικειμένων σε 2 φάσεις:

Στην πρώτη φάση βρίσκει για κάθε αντικείμενο που πρόκειται να φύγει από τον Πηγαίο εξυπηρετητή το κόστος (latency) που θα δημιουργήσει αν αντιγραφεί στον εξυπηρετητή 1,2,3..,N.

Στη δεύτερη φάση επιλέγει από τα ζευγάρια αντικείμενο – εξυπηρετητής που βρέθηκαν στο προηγούμενο βήμα αυτά που παράγουν το μεγαλύτερο Utility και τα τοποθετεί στους συγκεκριμένους εξυπηρετητές. Επαναλαμβάνει τη διαδικασία μέχρι όλοι οι εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου να γεμίσουν.

Η μετρική Utility ορίζεται ως εξής:

$$\textbf{Utility Value}_k = \textbf{load}_k \times \textbf{latency}_k \textbf{, όπου}$$

$$\textbf{load}_k = \textbf{access rate}_k \times \textbf{s}_k$$

Το latency_k είναι ο χρόνος καθυστέρησης που δημιουργεί το αντικείμενο k αν αντιγραφεί στον εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου που βρέθηκε από την πρώτη φάση, το load_k είναι το συνολικό φορτίο που προκαλεί το αντικείμενο k, το s_k είναι το μέγεθος του αντικειμένου k και το access rate_k είναι ο αριθμός των φορών που ζητείται το αντικείμενο.

Αυτός ο αλγόριθμος χρησιμοποιήθηκε σε αυτή τη διπλωματική στην πειραματική αξιολόγηση (Κεφάλαιο 6) σαν μέτρο σύγκρισης με τον αλγόριθμο που υλοποιήθηκε στο Κεφάλαιο 4 που επίσης αφορά την τοποθέτηση αντικειμένων στους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου. **Να σημειωθεί ότι η μετρική Utility που χρησιμοποιείται σε αυτό το άρθρο και η μετρική Utility που χρησιμοποιείται στον αλγόριθμο του Κεφαλαίου 4 και περιγράφεται στο 2.4 υπολογίζονται με διαφορετικό τρόπο**.

2.4 Μετρική Utility [3]

Το άρθρο [3] αναφέρει ότι τα Δίκτυα Παράδοσης Περιεχομένου εξισορροπούν το κόστος και την ποιότητα σε υπηρεσίες που αφορούν την παράδοση περιεχομένου. Αυτό οδήγησε πολλούς επιχειρηματίες να κάνουν συμβόλαια με τους παροχείς Δικτύου Παράδοσης Περιεχομένου. Αναφέρει ότι έχουν προταθεί πολλές τεχνικές μέχρι τώρα

για να βελτιώσουν την απόδοση των Δικτύων Παράδοσης Περιεχομένου και πως όλες είχαν σαν στόχο να βελτιώσουν τη χρησιμότητα των εξυπηρετητών των Δικτύων Παράδοσης Περιεχομένου. Το άρθρο ορίζει μια τέτοια μετρική η οποία ονομάζεται Utility. Αυτή η μετρική καταγράφει τη δραστηριότητα της κυκλοφορίας σε ένα Δίκτυο Παράδοσης Περιεχομένου, εκφράζοντας τη χρησιμότητα των εξυπηρετητών των Δικτύων Παράδοσης Περιεχομένου βάση του ποσοστού των δεδομένων που ανεβάζουν και κατεβάζουν στο δίκτυο.

Η μετρική Utility είναι η τιμή που εκφράζει τη σχέση μεταξύ του αριθμού των bytes που ανεβάζει (upload) ένας εξυπηρετητής έναντι του αριθμού των bytes που κατεβάζει (download). Η μετρική είναι κανονικοποιημένη στο εύρος [0..1] και μας δίνει πληροφορία για το πόσο ενεργός είναι ένας εξυπηρετητής CDN. Το Utlity U_i ενός εξυπηρετητή CDN i δίνεται από την ακόλουθη εξίσωση:

$$\boldsymbol{Ui = \frac{2}{\pi} \times \tan^{-1}(\xi)}$$

Αυτή η μετρική μας δίνει μια αίσθηση της χρησιμότητας του κάθε εξυπηρετητή CDN. Η παράμετρος ξ είναι το άθροισμα των bytes τα οποία ανέβασε ένας εξυπηρετητής προς το άθροισμα των bytes τα οποία κατέβασε. Για να πάρουμε τιμή Ui = 1 πρέπει ο εξυπηρετητής Δικτύου Παράδοσης Περιεχομένου να ανεβάζει μόνο περιεχόμενο (το ξ είναι άπειρο). Αν η τιμή του Utility βρίσκεται στην τιμή 0 τότε ο εξυπηρετητής Δικτύου Παράδοσης Περιεχομένου κατεβάζει μόνο περιεχόμενο. Σε ένα πραγματικό δίκτυο με N εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου η μετρική utility μπορεί να εκφραστεί ως εξής:

$$u = \frac{\sum_{i=1}^{N} \boldsymbol{Ui}}{\boldsymbol{N}}$$

Η μετρική utlity μπορεί επίσης να χρησιμοποιηθεί σαν παράμετρος για να καθορίσει τιμολόγηση μιας πολιτικής Δικτύου Παράδοσης Περιεχομένου. Ένα Δίκτυο παράδοσης περιεχομένου μεταφέρει περιεχόμενο για κάποιο παροχέα και χρεώνει βάση του traffic που χρησιμοποιεί. Ο στόχος είναι να βρεθεί το τελικό κόστος για τον παροχέα. Αφού η μετρική utility δείχνει τη χρήση του Δικτύου Παράδοσης Περιεχομένου μπορεί εύκολα

να μετατραπεί σε κόστος. Όπως βρέθηκε στο [19] το κόστος ενός Δικτύου Παράδοσης Περιεχομένου με βάση τη μετρική utility μπορεί να οριστεί με τον εξής τύπο:

$$\boldsymbol{UCDN = V(X) + \tau(N) \times X - Co - P(u)}$$

Όπου το UCDN είναι το τελικό κόστος του παροχέα, V(X) είναι το πλεονέκτημα του παροχέα όταν ανταποκρίνεται σε ένα πλήθος από αιτήσεις (X) , το τ(N) είναι το πλεονέκτημα ανά αίτηση, όπως παραδίδεται γεωγραφικά από N εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου, Co είναι το κόστος μετακίνησης αντικειμένων από την πηγή, P(u) είναι η συνάρτηση κόστους που βασίζεται στην χρησιμοποίηση και το u είναι η μετρική utility.

Στη μετρική Utility που περιγράφεται σε αυτό το άρθρο βασίζεται ο αλγόριθμος που περιγράφεται στο Κεφάλαιο 4 ο οποίος αφορά την τοποθέτηση αντικειμένων στους εξυπηρετητές του δικτύου Παράδοσης Περιεχομένου.

2.5 Γεωγραφική κατανομή των εξυπηρετητών Δικτύων Παράδοσης Περιεχομένου

Στο άρθρο [5] γίνονται μετρήσεις για δύο Δίκτυα Παράδοσης Περιεχόμενου μεγάλης έκτασης, του Akamai και του Limelight. Χρησιμοποιώντας μια εμπορική βάση που αντιστοιχεί το IP με την τοποθεσία το άρθρο βρίσκει ότι το Akamai έχει συνολικά 27000 εξυπηρετητές που αντιστοιχούν σε 65 διαφορετικές χώρες. Περισσότερο από το 60% των εξυπηρετητών βρίσκονται στην Αμερική και το 40% σε άλλες 10 χώρες. Η βάση δεδομένων που αντιστοιχεί το IP είναι ακριβής σε επίπεδο χώρας μόνο, άρα δεν υπάρχει η πληροφορία για την ακριβή γεωγραφική θέση των εξυπηρετητών.

Η γεωγραφική κατανομή των εξυπηρετητών της Akamai φαίνεται στο σχήμα 2.1

Country	# of IP	Percentage(%)
United States	16,843	61.09
United Kingdom	1,690	6.13
Japan	1,622	5.88
Germany	1,103	4.00
Netherlands	857	3.11
France	722	2.62
Australia	514	1.86
Canada	438	1.59
Sweden	396	1.44
Hong Kong SAR	370	1.34
Others	3018	10.95
Total	27,573	100.00

Σχήμα 2.1: *Δείχνει το πλήθος των μοναδικών IP που αντιστοιχούν σε εξυπηρετητές CDN ανά χώρα και το ποσοστό που αντιστοιχεί στο πλήθος για το δίκτυο της Akamai [5].*

Τα στοιχεία αυτού του άρθρου που αφορούν την κατανομή των εξυπηρετητών σε 11 χώρες χρησιμοποιούνται σε αυτή τη διπλωματική στο Κεφάλαιο 5 στον Αλγόριθμο για τοποθέτηση των Εξυπηρετητών Δικτύου Παράδοσης Περιεχομένου στην τοπολογία του διαδικτύου.

2.6 Πρόβλημα ιδανικής τοποθέτησης αντικειμένων στους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου: NP Complete

Το [7] Μελετά το πρόβλημα της ιδανικής τοποθέτησης αντικειμένων στους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου. Στο μοντέλο που περιγράφουν θεωρούν ότι κάθε Αυτόνομο Σύστημα (Autonomous System : AS) είναι ένας κόμβος που έχει πεπερασμένο χώρο για αντικείμενα. Το πρόβλημα βελτιστοποίησης είναι να αντιγράψουμε αντικείμενα έτσι ώστε όταν οι χρήστες ζητήσουν και λάβουν το αιτούμενο αντικείμενο ο μέσος αριθμός των αυτόνομων συστημάτων (κόμβων) που διασχίζονται από τον εξυπηρετητή προς τον χρήστη να ελαχιστοποιηθεί. Διατυπώνουν

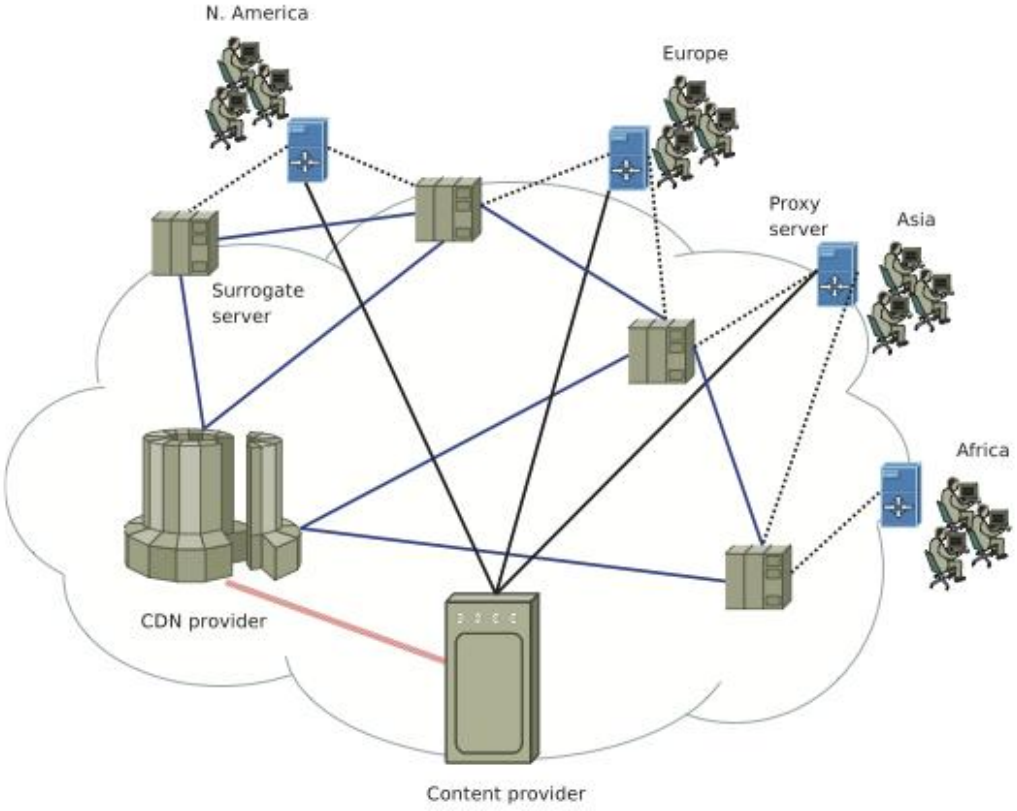
το πρόβλημα σαν ένα συνδυαστικό πρόβλημα βελτιστοποίησης και αποδεικνύουν ότι είναι NP-Complete.

Η τοποθέτηση αντικειμένων στους εξυπηρετητές του Δικτύου Παράδοσης Περιεχομένου είναι ένα από τα θέματα που ασχολείται αυτή η διπλωματική. Λόγω του ότι το πρόβλημα που περιγράφηκε είναι NP-Complete αυτό συνεπάγεται ότι δε μπορεί να λυθεί σε πολυωνιμικό χρόνο.

Κεφάλαιο 3

Θεωρητικό Υπόβαθρο

3.1 Δίκτυο Παράδοσης Περιεχομένου (CDN: Content Delivery Network)



Σχήμα 3.1: *Δίκτυο Παράδοσης Περιεχομένου (CDN) [20]*

Ένα Δίκτυο Παράδοσης Περιεχομένου είναι ένα δίκτυο από εξυπηρετητές που περιέχουν ενδιάμεσες μνήμες (cache) οι οποίοι ονομάζονται εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου και ανήκουν στον ίδιο Παροχέα Διαδικτύου. Το δίκτυο παραδίδει περιεχόμενο στους χρήστες. Μια τοπολογία Δικτύου Παράδοσης Περιεχομένου περιλαμβάνει 1) Σύνολο από εξυπηρετητές που φυλάσσουν στην ενδιάμεση μνήμη τους το περιεχόμενο ενός Πηγαίου Εξυπηρετητή 2) Δρομολογητές και άλλα συστατικά του δικτύου που παραδίδουν αιτήσεις για περιεχόμενο στη βέλτιστη τοποθεσία και στον βέλτιστο εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου 3) Μηχανισμό καταγραφής που παρέχει αρχεία log και δίνει πληροφορίες στον Πηγαίο εξυπηρετητή.

Στην υποδομή του Δικτύου Παράδοσης Περιεχομένου η επικοινωνία μεταξύ πελάτη-εξυπηρετητή αντικαθιστάται με 2 ροές: Μια μεταξύ του πελάτη και του εξυπηρετητή και μια άλλη μεταξύ του εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου και του

Πηγαίου εξυπηρετητή. Αυτή η διαφοροποίηση σε 2 ροές μειώνει τη συμφόρηση και αυξάνει τη διαθεσιμότητα των αντικειμένων.

Ένας χρήστης στέλνει αιτήσεις για αντικείμενα στον πλησιέστερο εξυπηρετητή. Ο τρόπος που διαχειρίζεται ένα cache miss και οι μεταπληροφορίες που χρειάζονται στον εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου εξαρτώνται από την πολιτική που αποφασίζει να χρησιμοποιήσει ο παροχέας του Δικτύου Παράδοσης Περιεχομένου. Παρόμοια θέματα όπως η οργάνωση του δικτύου σε ιεραρχία, ομάδες από εξυπηρετητές, τρόπος συνεργασίας μεταξύ τους, πολιτικές προσδιορισμού του κατάλληλου εξυπηρετητή για ένα χρήστη καθορίζονται επίσης από το παροχέα του Δικτύου Παράδοσης Περιεχομένου [1].

3.2 **Τοποθέτηση εξυπηρετητών σε ένα Δίκτυο Παράδοσης Περιεχομένου**

Η τοποθέτηση αντικειμένων σε συγκεκριμένες θέσεις στο δίκτυο μπορεί να βελτιώσει την επίδοση του δικτύου [14]. Μια ιδανική τοποθέτηση τους σε μια τοπολογία δικτύου μπορεί να ελαχιστοποιήσει τον αριθμό τους και αυτό θα έχει σαν αποτέλεσμα τη μείωση του κόστους και από πλευράς του παροχέα αλλά και από πλευράς χρήστη. Το πρόβλημα της τοποθέτησης διατυπώνεται ως εξής: Υπάρχουν N θέσεις σε ένα δίκτυο όπου μπορούν να τοποθετηθούν οι εξυπηρετητές ενός Δικτύου Παράδοσης Περιεχομένου. Από αυτές τις N θέσεις πρέπει να επιλεγούν K θέσεις έτσι ώστε το K να είναι κατά πολύ μικρότερο από το N ($K << N$). Με τον τρόπο αυτό θα ελαχιστοποιηθεί το συνολικό κόστος του δικτύου. Το πρόβλημα που περιγράφηκε είναι NP-Complete [7] και άρα δεν έχουν βρεθεί αποτελεσματικοί πολυωνυμικοί αλγόριθμοι με αποτέλεσμα να χρησιμοποιούνται εμπειρικές ευριστικές προσεγγίσεις.

3.3 **Διαχείριση Περιεχομένου**

Για τη διαχείριση περιεχομένου πρέπει α) Να καθοριστεί ποιο περιεχόμενο από τον Πηγαίο Εξυπηρετητή θα διατεθεί για να αντιγραφεί στους εξυπηρετητές ενός Δικτύου Παράδοσης Περιεχομένου. β) Σε ποιους εξυπηρετητές θα αντιγραφεί αυτό το περιεχόμενο. Εξ αιτίας του μικρού κόστους των αποθηκευτικών χώρων μια λύση θα ήταν να αντιγραφεί όλο το περιεχόμενο σε όλους τους εξυπηρετητές. Μια τέτοια λύση όμως εξ' αιτίας της μεγάλης πληροφορίας που θα διακινούταν θα δημιουργούσε μεγάλη συμφόρηση στο δίκτυο με κάθε αλλαγή του περιεχομένου του Πηγαίου εξυπηρετητή.

3.4 **Πολιτικές επικοινωνίας εξυπηρετητών σε ένα Δίκτυο Παράδοσης Περιεχομένου.**

Πολιτικές που υπάρχουν σήμερα στα Δίκτυα Παράδοσης Περιεχομένου είναι: Uncooperative pull-based, cooperative pull-based , cooperative push-based and uncooperative push-based.

Uncooperative pull-based

Οι αιτήσεις των χρηστών κατευθύνονται στον πιο κοντινό (από πλευράς γεωγραφικής απόστασης ή ποσότητας φορτίου) εξυπηρετητή. Τα μειονεκτήματα αυτής της μεθόδου είναι ότι δεν επιλέγεται πάντα ο βέλτιστος εξυπηρετητής στον οποίο θα αποσταλεί το περιεχόμενο όπως επίσης και το γεγονός ότι ένα αντικείμενο αντιγράφεται πολλές φορές σε διαφορετικούς εξυπηρετητές. Αυτή η μέθοδος χρησιμοποιείται από δημοφιλείς παροχείς όπως η Akamai και το LimeLight Network [11].

Cooperative pull-based

Οι αιτήσεις των χρηστών κατευθύνονται (μέσω ανακατεύθυνσης DNS) στον πιο κοντινό εξυπηρετητή. Το χαρακτηριστικό της πολιτικής cooperative pull-based είναι ότι οι εξυπηρετητές συνεργάζονται μεταξύ τους σε περίπτωση που υπάρχει cache miss. Με τη χρήση κατανεμημένων ευρετηρίων οι εξυπηρετητές βρίσκουν κοντινά αντίγραφα των ζητούμενων αντικειμένων και τα αποθηκεύουν στις κρύπτες τους [11].

Uncooperative push-based

Το περιεχόμενο κινείται από τον Πηγαίο εξυπηρετητή στους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου. Οι αιτήσεις μπορούν να ικανοποιηθούν είτε στον τοπικό εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου είτε στον Πηγαίο εξυπηρετητή, αλλά όχι σε κοντινό εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου εξ' αιτίας του ότι ο τοπικός εξυπηρετητής Δικτύου Παράδοσης Περιεχομένου δε γνωρίζει το περιεχόμενο των υπόλοιπων εξυπηρετητών. Σαν αποτέλεσμα αυτή η πολιτική δεν έχει μεγάλη ευελιξία στη μεταβολή του περιεχομένου που αντιγράφεται και στη διαχείριση του κόστους [11].

Cooperative push-based

Το περιεχόμενο κινείται από τον Πηγαίο εξυπηρετητή στους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου. Όταν γίνει ένα αίτημα για κάποιο αντικείμενο εάν ο εξυπηρετητής το έχει αποθηκευμένο στην ενδιάμεση μνήμη του το αποστέλλει, αλλιώς το διαβιβάζει στον αμέσως πιο κοντινό εξυπηρετητή που το έχει. Σε περίπτωση που το ζητούμενο αντικείμενο δεν έχει αντιγραφεί σε κάποιο εξυπηρετητή το αίτημα εξυπηρετείται από τον Πηγαίο Εξυπηρετητή. Αυτή η πολιτική τοποθέτησης ονομάζεται μακροπρόθεσμη προανάκτηση (long-term prefetching) και προσπαθεί να βρει μια συλλογή από αντικείμενα τα οποία είναι χρήσιμα για να αντιγραφούν. Παρόλο που αυτή η πολιτική ορίζει πως οι εξυπηρετητές έχουν συνεργασία μεταξύ τους και αυτό απαιτεί επιπλέον κόστος σε εύρος ζώνης (bandwidth) και σε επεξεργασία, το κόστος αυτό αποσβήνεται από το γεγονός ότι οι εξυπηρετητές μοιράζονται αποδοτικά το εύρος ζώνης μεταξύ τους και επίσης μειώνουν τα περιττά αντίγραφα στις ενδιάμεσες μνήμες τους [11].

3.5 **Πηγαίος εξυπηρετητής (Origin Server)**

Είναι ένας εξυπηρετητής διαδικτύου που περιέχει την αρχική σελίδα. Ο όρος χρησιμοποιείται για να το διαχωρίζει από τον εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου. Επειδή δεν υπάρχει φυσική διαφορά μεταξύ ενός αρχικού ψηφιακού (original file) και ενός ψηφιακού αντιγράφου (copy) ο όρος υποδηλώνει ότι ο Πηγαίος Εξυπηρετητής είναι αυτός που διαχειρίζεται και αλλάζεται από την επιχείρηση [3].

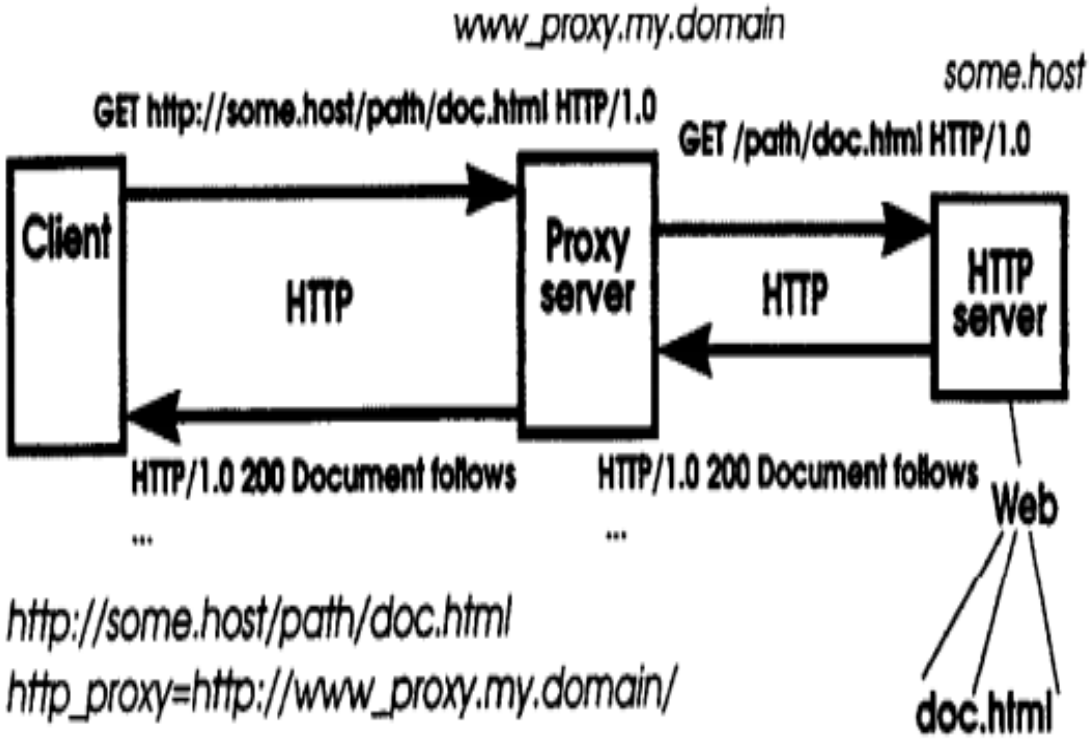
3.6 **Εξυπηρετητής Δικτύου Παράδοσης Περιεχομένου**

Σε ένα Δίκτυο Παράδοσης Περιεχομένου οι εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου προσπαθούν να μειώσουν το φόρτο του Πηγαίου Εξυπηρετητή με το να κάνουν παράδοση περιεχομένου αντί αυτού. Ο τρόπος λειτουργίας τους περιγράφεται ως εξής: Είναι τοποθετημένοι σε στρατηγικές θέσεις του διαδικτύου. Φυλάσσουν αντίγραφα των αντικειμένων που περιέχονται στον πηγαίο εξυπηρετητή στην ενδιάμεση μνήμη τους. Όταν ένας χρήστης κάνει αίτηση για ένα αντικείμενο μεταφέρεται αρχικά σε ένα εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου. Αν ο εξυπηρετητής έχει το αντικείμενο το αποστέλλει στον χρήστη. Με αυτό τον τρόπο έχουμε τη μείωση φόρτου στον Πηγαίο εξυπηρετητή, μικρότερους χρόνους απόκρισης για τους χρήστες και λιγότερη χρησιμοποίηση του δικτυού. [15]

3.7 **Εναποθήκευση Δεδομένων (Caching)**

Η εναποθήκευση έχει σαν στόχο να πετύχει καλύτερη επίδοση στο διαδίκτυο με το να αποθηκεύει στην ενδιάμεση μνήμη των εξυπηρετητών αντικείμενα που πρόκειται να ζητηθούν μελλοντικά από τους χρήστες. Με τον τρόπο αυτό μειώνεται η χρησιμοποίηση του Πηγαίου Εξυπηρετητή όπως επίσης και το traffic του συνολικού δικτύου εξ’ αιτίας της συντομότερης διαδρομής που ακολουθούν τα πακέτα για να φτάσουν στον χρήστη. Βελτιώνει επίσης την αξιοπιστία των Πηγαίων Εξυπηρετητών γιατί οι πελάτες μπορούν να ανακτήσουν τα αντικείμενα από τους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου ακόμα και αν οι Πηγαίοι Εξυπηρετητές δεν είναι διαθέσιμοι.

Η εναποθήκευση μπορεί να γίνει με την τοποθέτηση ενδιάμεσων μνημών σε συγκεκριμένες τοποθεσίες του δικτύου. Οι ενδιάμεσες μνήμες μπορούν να τοποθετηθούν είτε στην πλευρά του πελάτη είτε στην πλευρά του εξυπηρετητή. Αν η τοποθέτηση γίνει στην πλευρά του χρήστη τότε αυτή υλοποιείται μέσα στο λογισμικό πλοήγησης όμως η μνήμη είναι πολύ περιορισμένη αλλά η καθυστέρηση από πλευράς χρήστη είναι μειωμένη. Αν γίνει στην πλευρά του πληρεξούσιου εξυπηρετητή υπάρχει το πλεονέκτημα της μνήμης που έχουν οι εξυπηρετητές έναντι της μνήμης του υπολογιστή του χρήστη.



Σχήμα 3.2: Διαδικασία Εναποθήκευσης [13]

Το πιο πάνω σχήμα παρουσιάζει τη διαδικασία της εναποθήκευσης. Ο χρήστης (Client) κάνει ένα αίτημα http για το αντικείμενο doc.html μέσα από τον πλοηγό του. Η αίτηση του χρήστη περνά πρώτα από τον πληρεξούσιου εξυπηρετητή (proxy server). Στην περίπτωση του Δικτύου Παράδοσης Περιεχομένου ο πληρεξούσιος εξυπηρετητής ονομάζεται εξυπηρετητής Δικτύου Παράδοσης Περιεχομένου. Αυτός ελέγχει αν το αντικείμενο το οποίο αιτήθηκε ο χρήστης υπάρχει στην ενδιάμεση μνήμη του. Στην περίπτωση μας το αντικείμενο δεν υπάρχει στον πληρεξούσιο εξυπηρετητή και έτσι αυτός κάνει ένα αίτημα http για το ίδιο αντικείμενο στον πηγαίο εξυπηρετητή. Ο τελευταίος επιστρέφει το αντικείμενο στον πληρεξούσιο εξυπηρετητή ο οποίος το αποθηκεύει στην ενδιάμεση μνήμη του. Τέλος ο πληρεξούσιος εξυπηρετητής/ εξυπηρετητής Δικτύου Παράδοσης Περιεχομένου αποστέλλει το αρχείο στον πελάτη [13].

3.8 **Προανάκτηση Δεδομένων (Prefetching)**

Η προανάκτηση δεδομένων χρησιμοποιείται για να δώσει καλύτερα αποτελέσματα επίδοσης στα λειτουργικά συστήματα, στα συστήματα αρχείων και στα συστήματα του διαδικτύου. Η προανάκτηση πρέπει να λαμβάνει υπόψη το πλήθος των αντικειμένων που θα τύχουν προανάκτησης αλλά και το κόστος που θα φέρει στο δίκτυο. Η αυξημένη χρήση του δικτύου για τους σκοπούς της προανάκτησης θα επιβαρύνει το εύρος ζώνης του δικτύου και θα έχει σαν αποτέλεσμα μεγάλες καθυστερήσεις για τους χρήστες. Ο στόχος της προανάκτησης είναι να μειώσει το χρόνο καθυστέρησης. Αυτή η καθυστέρηση είναι το χρονικό διάστημα που περνά από τη στιγμή που ο χρήστης αιτείται μια ιστοσελίδα μέχρι τη στιγμή που αρχίζει να βλέπει την ιστοσελίδα. Τα θέματα που πρέπει να εξεταστούν για να μπορεί να εφαρμοστεί η προανάκτηση σε ένα εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου είναι:

Εφικτότητα: Πρέπει να ωφελούνται όλα τα συστήματα παροχής πληροφοριών από την προανάκτηση.

Σχεδιασμός: Ποιά από τα συστατικά στοιχεία του παγκόσμιου ιστού (χρήστες, εξυπηρετητές) θα λάβουν μέρος στην προανάκτηση.

Περιεχόμενο: Ποίο περιεχόμενο πρέπει να προανακτηθεί; Πρέπει να καθοριστούν οι τύποι των αρχείων που θα λάβουν μέρος στην διαδικασία της προανάκτησης.

Εφαρμογή: Πρέπει να σχεδιαστεί αλγόριθμος που λαμβάνοντας ένα σύνολο από αντικείμενα να μπορεί να προβλέπει ποια από αυτά είναι κατάλληλα να τύχουν προανάκτησης [12].

Κεφάλαιο 4

Αλγόριθμος για τοποθέτηση αντικειμένων στους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου που βασίζεται στη μετρική Utility.

4.1 Διατύπωση του προβλήματος

Έστω W ένας διαδικτυακός τόπος που περιέχει K αντικείμενα και χρησιμοποιεί το δίκτυο ενός παρόχου Δικτύου Παράδοσης Περιεχομένου. Το κάθε αντικείμενο k έχει μέγεθος k_s , όπου $k \in \{1..K\}$. Το δίκτυο αποτελείται από N εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου όπου ο καθένας χρησιμοποιείται σαν ενδιάμεσος μεταξύ του Πηγαίου Εξυπηρετητή και των πελατών.

Ο κάθε εξυπηρετητής i, όπου $i \in \{1..N\}$ ανεβάζει Xi bytes και κατεβάζει Yi bytes. Κάθε φορά που στέλνεται ένα αίτημα για ένα αντικείμενο k στον εξυπηρετητή i έχουμε τις εξής περιπτώσεις:

$$fik \begin{cases} 1\ Εάν\ το\ αντικείμενο\ k\ είναι\ τοποθετημένο\ στον\ εξυπηρετητή\ i \\ 0\ Εάν\ το\ k\ δεν\ είναι\ τοποθετημένο\ στον\ i \end{cases}$$

$$\text{Τότε το } Xi = \begin{cases} Xi + Sk\ εάν\ το\ fik = 1\ και\ ο\ εξυπηρετητής\ i\ ανεβάσει\ το\ k \\ Xi\ εάν\ το\ fik = 0 \end{cases}$$

$$\text{και το } Yi = \begin{cases} Yi + Sk\ εάν\ το\ fik = 0\ και\ ο\ εξυπηρετητής\ i\ κατεβάσει\ το\ k \\ Yi\ εάν\ το\ fik = 1 \end{cases}$$

Αρχικά θεωρώ ότι όλα τα αντικείμενα είναι αποθηκευμένα στον Πηγαίο Εξυπηρετητή και ότι όλοι οι εξυπηρετητές του Δικτύου Παράδοσης Περιεχομένου έχουν 0 αντικείμενα στην ενδιάμεση μνήμη τους.

Σκοπός είναι να βρεθεί μια τοποθέτηση [k , s], όπου k : αντικείμενο Є W και i: εξυπηρετητής, έτσι ώστε να μεγιστοποιηθεί η τιμή

$$\boldsymbol{Ui = \frac{2}{\pi} \times \tan^{-1}(\xi)}$$

Ui : Τιμή Utility σε κάθε εξυπηρετητή i και $\xi = \frac{Xi}{Yi}$

Το πρόβλημα της τοποθέτησης αντικειμένων ανήκει στην κατηγορία των NP-Complete προβλημάτων όπως αποδεικνύεται στο [7]. Άρα πρέπει να βρεθούν ευριστικοί αλγόριθμοι για την επίλυση του. Πιο κάτω προτείνεται ένας αλγόριθμος για τοποθέτηση αντικειμένων σε εξυπηρετητές ενός Δικτύου Παράδοσης Περιεχομένου που στηρίζεται στη μετρική Utility (Κεφάλαιο 2)

4.2 Ο αλγόριθμος utility

Πρώτη Φάση

Ο αλγόριθμος προσπαθεί να εξυπηρετήσει κάθε αντικείμενο που πρόκειται να φύγει από τον Πηγαίο Εξυπηρετητή (outsource). Βρίσκει για κάθε εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου το κόστος (χρόνο καθυστέρησης) που χρειάζεται για να σταλεί το αντικείμενο k από τον εξυπηρετητή i προς όλους τους χρήστες που το ζητούν.

Στη συνέχεια από το κάθε ζευγάρι αντικείμενο k, εξυπηρετητής i βρίσκει αυτά που έχουν το μέγιστο κόστος και το αντικείμενο k τοποθετείται στον εξυπηρετητή i.

Η πρώτη φάση έχει σαν στόχο να πετύχει 2 πράγματα:

1) Να ορίσει τον συνολικό αριθμό των bytes που θα κατεβάσει ο κάθε εξυπηρετητής Δικτύου Παράδοσης Περιεχομένου. Αυτό είναι απαραίτητο να γίνει γιατί χρειάζεται στη δεύτερη φάση του αλγορίθμου στον υπολογισμό της μετρικής Utility. Η μετρική Utility υπολογίζεται με βάση το σύνολο των bytes που ανεβάζει και κατεβάζει ο κάθε εξυπηρετητής Δικτύου Παράδοσης Περιεχομένου.

2) Να ελαχιστοποιήσει το συνολικό χρόνο καθυστέρησης του δικτύου Παράδοσης Περιεχομένου. Όπως περιγράφηκε προηγουμένως στην πρώτη φάση του αλγορίθμου τα αντικείμενα που προκαλούν το μεγαλύτερο χρόνο καθυστέρησης στους συγκεκριμένους εξυπηρετητές κατεβαίνουν σε αυτούς. Επομένως ο συνολικός χρόνος καθυστέρησης του δικτύου μειώνεται.

Δεύτερη Φάση

Για κάθε αντικείμενο k και για κάθε εξυπηρετητή i ελέγχουμε εάν ο i έχει στην ενδιάμεση μνήμη του το k. Αν ναι τότε για κάθε εξυπηρετητή j, όπου $i \neq j$ και j έχει στη μνήμη του το k υπολογίζουμε:

1) Το κόστος (χρόνο καθυστέρησης) για να σταλεί το αντικείμενο k από τον εξυπηρετητή i στο εξυπηρετητή j.

2) Το κόστος (χρόνο καθυστέρησης) για να σταλεί το k από τον Πηγαίο εξυπηρετητή στον εξυπηρετητή j.

Εάν το κόστος από το 1) είναι μεγαλύτερο του κόστους στο 2) τότε ο i κάνει upload το αντικείμενο k. Για κάθε εξυπηρετητή i υπολογίζεται η τιμή της μετρικής Utility που παράγει. Σε αυτό τον αλγόριθμο η πολιτική επικοινωνίας των εξυπηρετητών Δικτύου Παράδοσης Περιεχομένου είναι η cooperative-push based. Αυτή η πολιτική ορίζει ότι το περιεχόμενο κινείται από τον Πηγαίο εξυπηρετητή προς τους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου και μετά αυτοί συνεργάζονται μεταξύ τους για να μειώσουν το κόστος που επιφέρει η αντιγραφή περιττού περιεχομένου στις ενδιάμεσες μνήμες τους. Αυτή η πολιτική αποδείχτηκε στο [7] ότι έχει τα καλύτερα αποτελέσματα. Για το λόγο αυτό στο 1) υπολογίζουμε κόστος (χρόνος καθυστέρησης) που επιφέρει το αντικείμενο k όταν μεταφερθεί από εξυπηρετητή σε εξυπηρετητή και μετά το συγκρίνουμε με το κόστος που υπολογίζεται στο 2) που αφορά κόστος από τον Πηγαίο εξυπηρετητή προς τους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου. Ο λόγος που υπολογίζονται τα 2 κόστη είναι για να γίνεται σύγκριση μεταξύ τους και αν το κόστος 1) είναι μικρότερο από το κόστος 2) τότε να γίνεται ανέβασμα αντικειμένων από ένα εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου προς ένα άλλο εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου παρά από τον Πηγαίο εξυπηρετητή προς ένα εξυπηρετητή Δικτύου Παράδοσης Περιεχομένου.

Στη συνέχεια αφού έχουμε υπολογίσει το Utility όλων των εξυπηρετητών για το αντικείμενο k βρίσκουμε τη μέγιστη τιμή του Utility. Για κάθε εξυπηρετητή i συγκρίνουμε την τιμή του Utility του με την μέγιστη τιμή Utility. Εάν δεν είναι ίση με τη μέγιστη τιμή του Utility τότε αφαιρούμε το αντικείμενο k από την ενδιάμεση μνήμη του i εφ' όσων ισχύει ότι η πιθανότητα να λήξει το k είναι μεγαλύτερη από ένα κατώφλι P. Όπως περιγράφεται στο 4.4 σε κάθε αντικείμενο που υπάρχει στον Πηγαίο εξυπηρετητή ανατίθεται μια πιθανότητα λήξης. Πριν την εκτέλεση των πειραμάτων αφαιρούνται κάποια αντικείμενα με βάση την πιθανότητα λήξης τους. Αυτό γίνεται για να προσομοιωθεί μια πιο πραγματική συμπεριφορά του διαδικτύου όπου τα αντικείμενα έχουν ημερομηνία λήξης. Έτσι ο αλγόριθμος utility λαμβάνει υπόψη τα αντικείμενα που πρόκειται να λήξουν και αφαιρεί μόνο όσα η πιθανότητα τους να λήξουν είναι κάτω από ένα κατώφλι.

Αυτό το βήμα έχει σαν στόχο να μεγιστοποιήσει το συνολικό Utility του δικτύου. Το επιτυγχάνει με το να κρατά στις ενδιάμεσες μνήμες των εξυπηρετητών μόνο τα

αντικείμενα που μεγιστοποιούν το Utility του κάθε εξυπηρετητή. Αφού ο κάθε εξυπηρετητής θα έχει μέγιστο Utility τότε σύμφωνα με τον τύπο:

$$u = \frac{\sum_{i=1}^{N} Ui}{N}$$

θα έχουμε μεγιστοποίηση του συνολικού utility ολόκληρου του δικτύου.

Ui: Το utility του εξυπηρετητή CDN i.

N: Συνολικός αριθμός των εξυπηρετητών CDN.

Η πολυπλοκότητα του αλγορίθμου είναι $O(k * i^2)$ όπου k είναι ο αριθμός των αντικειμένων και i ο αριθμός των εξυπηρετητών.

4.3 Ψευδοκώδικας αλγορίθμου utility

```
Inputs:
Obj[1..K]
Sur[1..N]
Cl[1..M]
Origin Server : o
Output:
A placement x of outsourced objects to surrogate servers
//Phase 1
for each Object k {
      for each Surrogate n {
            for each Client m {
                  if  m. requestsObject(k) {
                        // Cost is the latency
                        Cost_Surrogate[n] += Cost to sent object k
from surrogate n to client m
                  }
            }
}
Find the Surrogates (s) with the maximum Cost_Surrogate;
if  !s.hasObject(k) && s.hasMaximumCost(){
s.downloadObject(k);
}
}
//Phase 2
for each Object k {
      for each Surrogate n1 {
```

```
            if n1.hasObject(k) {
                  for each Surrogate n2 {
                        if (s1.IsNotTheSameWith(s1) &&
n2.hasObject(k)){
                              costS1_S2 = Calculate cost to sent (k)
from s1 to s2;
                              costO_S2 = Calculate cost to sent (k)
from o to s2;
                              if (costS1_S2 <= costO_S2) {
                                    sur1.uploadObject(k);
                              }
                        }
                  }
                  sur1.calculateUtility();
            }
      }
      for each Surrogate n {
            max_Utility = find the Maximum Utility Value;
      }
      for each Surrogate n {
            if n.hasObject(k) {
                  cur_Utility = n.get_Utility();
                  if (cur_Utility < max_Utility) {
                        if (k.getExpireProbability() > P) {
                              n.removeObject(k);
                        }
                  }
            }
      }
}
```

4.4 Διαχείριση Δυναμικών Αντικειμένων

Οι δυο αλγόριθμοι που περιγράφηκαν στο Κεφάλαιο 2 δεν έκαναν καμία υπόθεση, ούτε κάποια διαχείριση των δυναμικών αντικειμένων. Σήμερα λόγω των ραγδαίων αλλαγών που συμβαίνουν στον κόσμο (συνεχείς εξελίξεις στην οικονομία, τεχνολογία, πολιτική) οι διαχειριστές των ιστιακών χώρων κάνουν συνεχώς αλλαγές στο περιεχόμενο των

σελίδων τους με αποτέλεσμα τα αντικείμενα να αλλάζουν συνεχώς. Μπορεί να δημιουργηθεί το εξής ενδεχόμενο: Ένας εξυπηρετητής Δικτύου Παράδοσης Περιεχομένου Α ζητά από τον πηγαίο εξυπηρετητή Β το αντικείμενο k το οποίο αποθηκεύει στην ενδιάμεση μνήμη του στο χρόνο τ. Στο χρόνο τ + Δτ ο διαχειριστής του ιστακού χώρου αποφασίζει να αλλάζει το αντικείμενο k. Έτσι το αντικείμενο k που βρίσκεται στον εξυπηρετητή Α δεν έχει πλέον καμία αξία χρήσης. Ο αλγόριθμος του Utility λαμβάνει υπόψη το γεγονός ότι υπάρχει ημερομηνία λήξης στα αντικείμενα ενός ιστοχώρου. Για το λόγο αυτό υλοποίησα μια μέθοδο η οποία δίνει τυχαία πιθανότητα λήξης στα αντικείμενα του ιστοχώρου στον πηγαίο εξυπηρετητή και μια μέθοδο η οποία αφαιρεί αντικείμενα από τους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου ανάλογα με την πιθανότητα λήξης που έχουν. Τα αντικείμενα αφαιρούνται πριν από την εκτέλεση της προσομοίωσης.

4.4.1 Ανάθεση τυχαίας πιθανότητας λήξης στο κάθε αντικείμενο.

Το αρχείο που περιγράφει τον ιστιακό χώρο είναι της μορφής: «αριθμός αντικειμένου», «μέγεθος αντικειμένου», όπου ο «αριθμός αντικειμένου» είναι ένας αύξον και μοναδικός αριθμός που χαρακτηρίζει ένα αντικείμενο και το «μέγεθος αντικειμένου» είναι το μέγεθος που έχει το αντικείμενο σε bytes.

Αριθμός αντικειμένου	Μέγεθος αντικειμένου
0	388637
1	71125
2	58512
3	101924
4	106333
5	368734

Σχήμα 4.1: *Παράδειγμα αρχείου που περιγράφει τα αντικείμενα μιας ιστοσελίδας*

Δημιουργώ με τυχαίο τρόπο ένα αριθμό που εκφράζει πιθανότητα η οποία είναι ένας πραγματικό αριθμός ακρίβειας 2 δεκαδικών ψηφίων και την αναθέτω σε κάθε αντικείμενο. Η πιθανότητα αυτή ορίζει το πόσο πιθανό είναι να λήξει το συγκεκριμένο αντικείμενο με το 1 να σημαίνει ότι το αντικείμενο θα λήξει οπωσδήποτε και το 0 ότι δεν πρόκειται να λήξει. Αφού γίνει η ανάθεση σε όλα τα αντικείμενα το αρχείο θα έχει

την εξής μορφή: «αριθμός αντικειμένου», «μέγεθος αντικειμένου», «πιθανότητα λήξης».

Αριθμός αντικειμένου	Μέγεθος αντικειμένου	Πιθανότητα Λήξης
0	388637	0.26
1	71125	0.48
2	58512	0.84
3	101924	0.95
4	106333	0.87
5	368734	0.89

Σχήμα 4.2: *Παράδειγμα αρχείου μετά την ανάθεση τυχαίας πιθανότητας σε κάθε αντικείμενο.*

4.4.2 Αφαίρεση αντικειμένων από τους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου ανάλογα με την πιθανότητα λήξης του κάθε αντικειμένου.

Η μέθοδος λαμβάνει το αρχείο που πάραξε το 4.4.1 και τα αρχεία που δημιούργησε ο αλγόριθμος που περιγράφεται στο 4.2. Στη συνέχεια δημιουργεί ένα δυσδιάστατο πίνακα A[K][3], όπου K είναι ο αριθμός των αντικειμένων. Η ανάθεση στον πίνακα γίνεται ως εξής:

A[1][1]: Αριθμός αντικειμένου 1,

A[1][2]: Μέγεθος αντικειμένου 1,

A[1][3]: Πιθανότητα λήξης αντικειμένου 1

, … ,

A[K][1]: Αριθμός αντικειμένου K,

A[K][2]: Μέγεθος αντικειμένου K,

A[K][3]: Πιθανότητα λήξης αντικειμένου K

Στη συνέχεια τα αντικείμενα χωρίζονται σε 10 ομάδες με βάση την πιθανότητα λήξης τους ως εξής; Η ομάδα 1 περιέχει αντικείμενα που η πιθανότητα τους είναι <= 1.0 και >0.9, η ομάδα 2 αντικείμενα με πιθανότητα <=0.9 και >0.8 κ.ο.κ.

Στο επόμενο βήμα ορίζω το πλήθους των αντικειμένων που έχουν λήξει ως εξής:

N1 = Πλήθος αντικειμένων που έχουν λήξει από την ομάδα 1 = 0 % * πλήθος των αντικειμένων που ανήκουν στην ομάδα 1.

N2 = Πλήθος αντικειμένων που έχουν λήξει από την ομάδα 2 = 10 % * πλήθος των αντικειμένων που ανήκουν στην ομάδα 2.

…

N10 = Πλήθος αντικειμένων που έχουν λήξει από την ομάδα 10 = 90 % * πλήθος των αντικειμένων που ανήκουν στην ομάδα 10.

Στη συνέχεια ορίζω ποια αντικείμενα θα αφαιρεθούν από κάθε ομάδα με τον εξής τρόπο:

Από την ομάδα 1 κάνω τυχαία επιλογή N1 αντικειμένων, από την ομάδα 2 τυχαία επιλογή N2 αντικειμένων, … , από την ομάδα 10 επιλογή N10 τυχαίων αντικειμένων.

Στο τελικό βήμα αφαιρώ από τα αρχεία που δημιουργεί ο αλγόριθμος στο 4.2 όλα τα αντικείμενα που έχω επιλέξει από προηγούμενο βήμα.

Κεφάλαιο 5

Αλγόριθμος για τοποθέτηση των Εξυπηρετητών Δικτύου Παράδοσης Περιεχομένου στην τοπολογία του διαδικτύου.

5.1 Περιγραφή του προβλήματος

Ο CDNSim λαμβάνει ένα αρχείο που περιγράφει την τοπολογία του διαδικτύου και παραμέτρους που ορίζουν τον αριθμό των εξυπηρετητών και των πελατών. Στη συνέχεια δημιουργεί το δίκτυο Παράδοσης Περιεχομένου στη μορφή ενός γράφου συνδέοντας με τυχαίο τρόπο τους εξυπηρετητές και τους χρήστες στην δεδομένη τοπολογία. Αυτό θα επηρέαζε την εγκυρότητα των αποτελεσμάτων των πειραμάτων αφού τα αποτελέσματα που θα παρήγαγαν οι 2 αλγόριθμοι (utility και il2p) θα βασίζονταν σε δίκτυο που έχει δημιουργηθεί με τυχαία τοποθέτηση και όχι σε ένα δίκτυο που ανταποκρίνεται στην πραγματικότητα.

Για το λόγο αυτό παρουσιάστηκε η ανάγκη για την τοποθέτηση των εξυπηρετητών και των πελατών σε συγκεκριμένους κόμβους της τοπολογίας με σκοπό τα πειράματα που θα γίνουν να δώσουν πιο ρεαλιστικά αποτελέσματα.

5.2 Περιγραφή Αλγορίθμου

Η τοποθέτηση βασίστηκε στο άρθρο [5]. Το άρθρο περιέχει στοιχεία για πειράματα που έχουν γίνει σε 2 Δίκτυα Παράδοσης Περιεχομένου (Akamai και Limelight).

Το άρθρο βρίσκει το πλήθος των μοναδικών IP τα οποία αντιστοιχούν σε διαφορετικούς εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου του δικτύου της Akamai. Βάσει του IP βρίσκει σε ποια χώρα αντιστοιχούν οι εξυπηρετητές και τους χωρίζει σε τμήματα.

Συγκεκριμένα δημιουργούνται 11 τμήματα όπου στο κάθε ένα αντιστοιχεί ένα ποσοστό το οποίο υπολογίζεται διαιρώντας το πλήθος του τμήματος με το συνολικό αριθμό των εξυπηρετητών.

Country	# of IP	Percentage(%)
United States	16,843	61.09
United Kingdom	1,690	6.13
Japan	1,622	5.88
Germany	1,103	4.00
Netherlands	857	3.11
France	722	2.62
Australia	514	1.86
Canada	438	1.59
Sweden	396	1.44
Hong Kong SAR	370	1.34
Others	3018	10.95
Total	27,573	100.00

Σχήμα 5.1: Πίνακας που δείχνει την κατανομή των εξυπηρετητών στο δίκτυο της Akamai [5].

Εισόδοι αλγορίθμου:

1. Αρχείο που περιγράφει τη συνδεσμολογία των δρομολογητών του διαδικτύου υπό μορφή ενός γράφου
2. Αριθμός των εξυπηρετητών που θα έχει το δίκτυο παράδοσης περιεχομένου
3. Αριθμός των χρηστών που θα έχει το δίκτυο παράδοσης περιεχομένου

Έξοδος αλγορίθμου:

1. Αρχείο που περιγράφει τη συνδεσμολογία του δικτύου παράδοσης περιεχομένου υπό μορφή γράφου. Οι κόμβοι της εισόδου 1 χωρίζονται σε τμήματα και στο κάθε τμήμα τοποθετείται συγκεκριμένος αριθμός από εξυπηρετητές και χρήστες όπως περιγράφεται στο άρθρο [5].

5.3 Ορισμός Παραμέτρων Αλγορίθμου

Τα πειράματα του άρθρου [5] γίνονται σε πραγματική τοπολογία διαδικτύου ενώ τα πειράματα αυτής της διπλωματικής βασίζονται σε μια τοπολογία που δίνεται σε μορφή που είναι κατανοητή από το CDNSim. Η τοπολογία αποτελείται από 3037 κόμβους και δημιουργήθηκε με το εργαλείο GT-ITM internetwork topology generator όπως περιγράφεται στο άρθρο [6]. Ο αλγόριθμος λαμβάνει σαν εισόδους την τοπολογία των 3037 κόμβων που αντιπροσωπεύουν δρομολογητές, τον αριθμό των χρηστών και τον αριθμό των εξυπηρετητών του Δικτύου Παράδοσης Περιεχομένου. Δημιουργεί 11 ομάδες που αποτελούνται από κόμβους της τοπολογίας και αναθέτει σε κάθε ομάδα ένα αριθμό από εξυπηρετητές.

5.4 Δημιουργία τμημάτων

Έστω V ένα σύνολο από κόμβους {1..n} ενός γράφου G. Επιλέγουμε τυχαία ένα κόμβο Vi από το σύνολο V. Για κάθε κόμβο Vj Є V υπολογίζουμε το κόστος D(Vi,Vj) όπου το D είναι η διαδρομή ελαχίστου κόστους (βάση του hop count) από το Vi στο Vj. Στη συνέχεια ταξινομούμε το κάθε κόστος σε αύξουσα σειρά έτσι ώστε D(Vi,Vi) < D(Vi,Vj) … < D(Vi,Vj') < D(Vi,Vj''). Έτσι έχουμε μια ακολουθία όπου όσο πιο δεξιά βρίσκεται κάποιο κόστος από το D(Vi,Vi) τόσο πιο μεγάλη θα είναι η απόσταση του κόμβου Vj από τον κόμβο Vi.

Στη συνέχεια χωρίζουμε την ακολουθία σε τμήματα. Έστω A[1..K] πίνακας που κάθε θέση του k αντιπροσωπεύει το ποσοστό που αντιστοιχεί σε ένα από τα K τμήματα. Για να βρούμε πόσοι κόμβοι αντιστοιχούν σε κάθε τμήμα πολλαπλασιάζουμε το Ak * n έτσι ώστε κάθε τμήμα να έχει αριθμό κόμβων:

N(Tk): Όπου T:τμήμα,

k:αριθμός τμήματος

N(T1) = A1 * n,

N(T2) = A2 * n,

, … ,

N(T11) = A11 * n

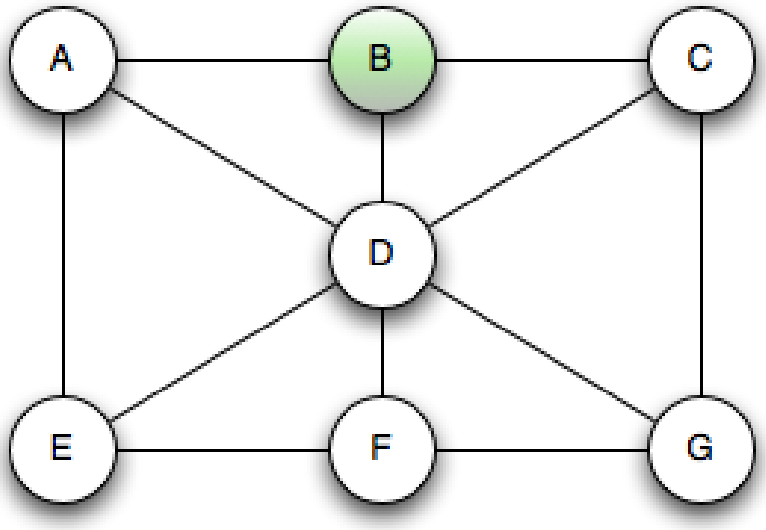
5.5 Ανάθεση κόμβων στα τμήματα

Αφού υπολογίσουμε πόσοι κόμβοι θα αντιστοιχούν σε κάθε τμήμα πρέπει να αναθέσουμε ποιοι κόμβοι θα ανήκουν στο κάθε τμήμα. Από την ακολουθία που βρήκαμε προηγουμένως D(Vi,Vi) < D(Vi,Vj) … < D(Vi,Vj') < D(Vi,Vj'') θέτω πως στη θέση 0 βρίσκεται το κόστος D(Vi,Vi) στη θέση 1 το κόστος D(Vi,Vj), … στη θέση n το κόστος D(Vi,Vj'').

Στο τμήμα 1 αναθέτω τους κόμβους 0 μέχρι το N(T1) στο τμήμα 2 τους κόμβους N(T1) + 1 μέχρι N(T2), … στο τμήμα 11 τους κόμβους N(T10) + 1 μέχρι N(T11). Με

τη μέθοδο αυτή δημιουργούνται 11 τμήματα από κόμβους (εξυπηρετητές). Οι κόμβοι που ανήκουν στο ίδιο τμήμα θα έχουν ίση ή κοντινή απόσταση από τον κόμβο Vi.

Παράδειγμα ανάθεσης κόμβων σε 2 τμήματα



Σχήμα 5.2: *Παράδειγμα γράφου με 7 κόμβους*

Τυχαίος Κόμβος Vi	Κόμβος Vj	Κόστος (Vi,Vj)
B	B	0
B	E	2
B	F	2
B	G	2
B	A	1
B	C	1
B	D	1

Σχήμα 5.3: *Απόσταση κάθε κόμβου από τον τυχαία επιλεγόμενο κόμβο B.*

Γίνεται ταξινόμηση του πίνακα (Σχήμα 5.3) στο πεδίο «Κόμβος Vj» βάσει του πεδίου «Κόστος (Vi,Vj)». Ο πίνακας που θα προκύψει από την ταξινόμηση φαίνεται στο Σχήμα 5.4

Τυχαίος Κόμβος Vi	**Κόμβος Vj**	**Κόστος (Vi,Vj)**
B	B	0
B	A	1
B	C	1
B	D	1
B	E	2
B	F	2
B	G	2

Σχήμα 5.4: *Ταξινομημένος πίνακας*

Επειδή οι κόμβοι A,C,D βρίσκονται σε μικρή απόσταση από το θα ανατεθούν στο τμήμα 1 και οι κόμβοι E,F,G που έχουν μεγαλύτερη απόσταση από το B στο τμήμα 2.

5.6 Ανάθεση Εξυπηρετητών Δικτύου Παράδοσης Περιεχομένου στα τμήματα

Έστω S ένα σύνολο {1..s} από εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου. Χρησιμοποιώντας τον πίνακα A (Σχήμα 5.1) βρίσκουμε τον αριθμό των εξυπηρετητών που αντιστοιχούν σε κάθε τμήμα.
Για να βρούμε πόσοι εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου αντιστοιχούν σε κάθε τμήμα πολλαπλασιάζουμε το Ak *s έτσι ώστε κάθε τμήμα να έχει αριθμό εξυπηρετητών N(Tk): Όπου T:τμήμα και k:αριθμός τμήματος

N(T1) = A1 * s,
N(T2) = A2 * s,

,…,

$N(T11) = A11 * s$

Αφού έχουμε βρει από την τοπολογία που δόθηκε στο 5.3 ποιοι κόμβοι (δρομολογητές) αντιστοιχούν στα 11 τμήματα και έχουμε υπολογίσει πόσοι εξυπηρετητές αντιστοιχούν σε κάθε τμήμα τώρα πρέπει να αναθέσουμε τους εξυπηρετητές στα τμήματα. Αυτό γίνεται ως εξής:

Στους κόμβους (δρομολογητές) που αντιστοιχούν στο τμήμα 1 αναθέτουμε με τυχαίο τρόπο N(T1) πλήθος από εξυπηρετητές στο τμήμα 2 N(T2) πλήθος, κ.ο.κ.

Κεφάλαιο 6

Πειραματική Αξιολόγηση

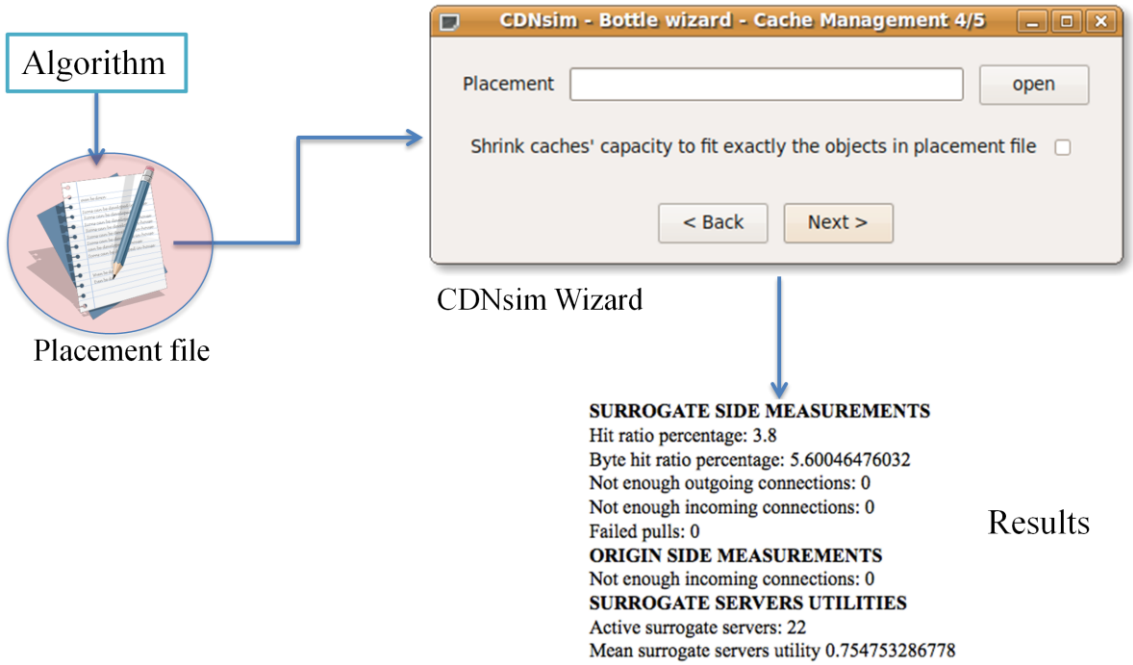
6.1 Εισαγωγή

Σε αυτό το κεφάλαιο περιγράφεται η πειραματική αξιολόγηση του αλγορίθμου utility ο οποίος περιγράφηκε στο κεφάλαιο 4. Γίνεται σύγκριση του με τον αλγόριθμο il2p που περιγράφεται αναλυτικά στο [1] και στο κεφάλαιο 2.
Τα πειράματα τρέχουν στον προσομοιωτή CDNSim [10]

6.2 Μεθοδολογία Εκτέλεσης Πειραμάτων

Ο αλγόριθμος utility που περιγράφηκε στο κεφάλαιο 4 και ο αλγόριθμος il2p που περιγράφηκε στο κεφάλαιο 2 και στο [1] λαμβάνουν σαν εισόδους ένα αρχείο που περιγράφει μια τοπολογία ενός δικτύου παράδοσης περιεχομένου (η έξοδος που παράγει ο αλγόριθμος του κεφαλαίου 5), ένα αρχείο που περιγράφει ένα ιστιακό χώρο και ένα αρχείο που περιγράφει τις αιτήσεις των χρηστών. Οι αλγόριθμοι παράγουν σαν έξοδο ένα αρχείο τοποθέτησης (Placement file) το οποίο ορίζει τα αντικείμενα του Πηγαίου εξυπηρετητή που έχουν αντιγραφεί στον κάθε εξυπηρετητή. Η δομή του κάθε αρχείου εισόδου/εξόδου περιγράφεται λεπτομερώς στο 6.3.

Τα αρχεία εισόδου καθώς και το αρχείο εξόδου εισάγονται στη συνέχεια στο εργαλείο CDNSim το οποίο τρέχει μια προσομοίωση ενός Δικτύου παράδοσης περιεχομένου (χρήση πρωτοκόλλου TCP/IP, congestion control, διαχωρισμός αντικειμένων σε πακέτα) και στο τέλος της προσομοίωσης παράγει αποτελέσματα όπως Hit Ratio Percentage, Mean surrogate server Utility, Mean Response Time κ.α.



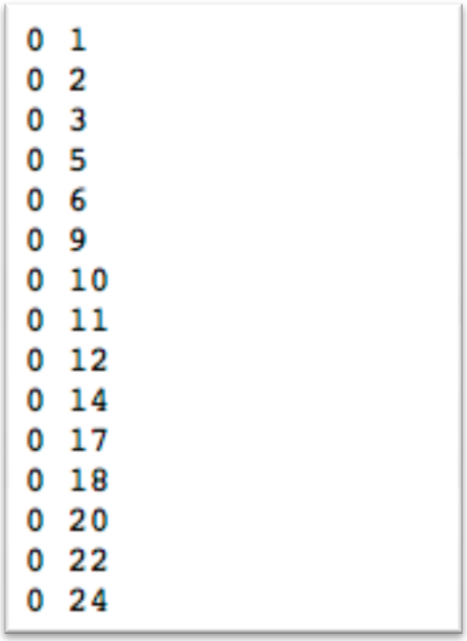
Σχήμα 6. 1: *Μεθοδολογία εκτέλεσης πειραμάτων*

6.3 Περιγραφή των Datasets

Αυτά τα Datasets χρησιμοποιούνται σαν εισόδοι στον CDNSim στον αλγόριθμο του Κεφαλαίου 4 και στον αλγόριθμο il2p [1].

Τοπολογία δικτύου: Το αρχείο περιγράφει το backbone της τοπολογίας του διαδικτύου που αναπαριστάται σαν γράφος από δρομολογητές. Η μορφή του αρχείου είναι “int int \n”. Κάθε ακέραιος αναπαριστά ένα id ενός κόμβου και κάθε γραμμή αναπαριστά ένα σύνδεσμο μεταξύ των κόμβων. Μια γραμμή θεωρείται πως αναπαριστά ακμή διπλής κατεύθυνσης. Ο CDNSim μέσα από την εφαρμογή wizard που προσφέρει τοποθετεί σε τυχαίες θέσεις της τοπολογίας δικτύου τους εξυπηρετητές και τους πελάτες που του ορίζει ο χρήστης. Στο Κεφάλαιο 5 περιγράφεται ένας αλγόριθμος που τοποθετεί με πιο

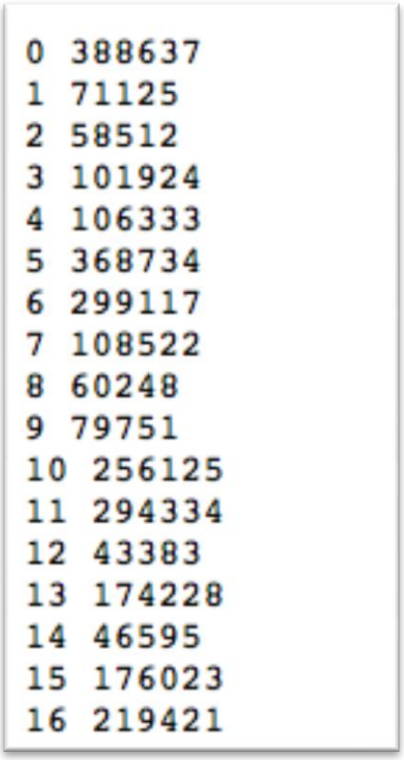
ρεαλιστικό τρόπο τους εξυπηρετητές και τους πελάτες. Οι ταυτότητες των κόμβων πρέπει να είναι στο εύρος [0, αριθμό των δρομολογητών]



Σχήμα 6. 2 : *Τοπολογία AS*

Στα Σενάρια αυτού του κεφαλαίου χρησιμοποιείται η τοπολογία “as3037” Η τοπολογία αποτελείται από 3037 κόμβους και δημιουργήθηκε με το εργαλείο GT-ITM internetwork topology generator όπως περιγράφεται στο άρθρο [6].

Ιστιακός Χώρος: Το αρχείο περιέχει όλα τα αντικείμενα που βρίσκονται στον εξυπηρετητή Origin και πρόκειται να αντιγραφούν σε άλλους εξυπηρετητές του CDN. Η μορφή του αρχείου είναι “int int\n”, όπου ο πρώτος ακέραιος είναι μια μοναδική ταυτότητα αντικειμένου και ο δεύτερος ακέραιος είναι το μέγεθος του αντικειμένου σε bytes. Το εύρος τιμών των ταυτοτήτων των αντικειμένων πρέπει να είναι από [0, αριθμό των αντικειμένων]



Σχήμα 6. 3 *Παράδειγμα αρχείου που περιγράφει ένα ιστιακό χώρο*

Τα αρχεία για τους ιστιακούς χώρους που χρησιμοποιούνται σε αυτή τη διπλωματική τα έλαβα από το φοιτητή Δημητρίου Νικόλα. Τα αρχεία αφορούν 3 ιστακούς τόπους και είναι οι εξής: msnbc, bbc, sporting news. Ο πίνακας με το πλήθος των αντικειμένων που περιέχει ο κάθε ιστιακός τόπος καθώς και το μέγεθος του ιστιακού τόπου φαίνονται στον πιο κάτω πίνακα:

Όνομα ιστιακού χώρου	Πλήθος αντικειμένων	Μέγεθος (Mbytes)
Msnbc	7451	4944
BBC	17030	568
Sporting News	35961	500

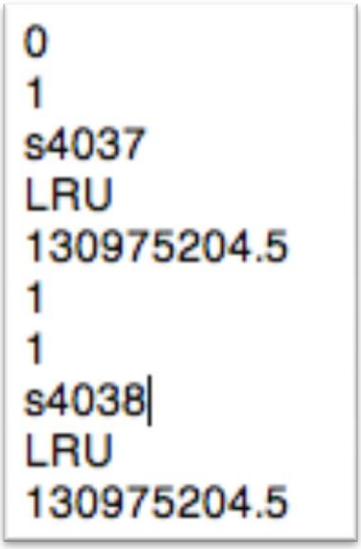
Σχήμα 6.4 : *Datasets των ιστιακών χώρων που χρησιμοποιήθηκαν στα πειράματα.*

Αρχείο Traffic: Αυτό το αρχείο περιγράφει ποιοι χρήστες θα αναζητήσουν ποιο περιεχόμενο και ποια ώρα θα το ζητήσουν. Η μορφή του αρχείου είναι "double in tint\n". Ο πρώτος αριθμός είναι η χρονοσφραγίδα της αίτησης ο δεύτερος αριθμός είναι αριθμός που αντιπροσωπεύει το χρήστη και ο τρίτος αριθμός είναι το αντικείμενο που αιτείται ο χρήστης. Η χρονοσφραγίδα παίρνει τιμές >=0 και το αρχείο του traffic πρέπει να είναι ταξινομημένο σε αύξουσα σειρά με βάση τη χρονοσφραγίδα. Το πεδίο των χρηστών παίρνει τιμές από το [0, αριθμό των πελατών].

```
2 25097 10250
9 58093 11159
15 21390 3229
15 65730 1157
21 77012 269
23 21390 3110
28 58093 11238
28 77854 10350
29 58093 11234
32 58093 11150
34 97938 484
38 25097 10188
41 33884 11470
42 21390 3182
42 65730 1290
44 25097 10158
```

Σχήμα 6. 5*: Παράδειγμα αρχείου που περιγράφει το traffic των χρηστών*

Αρχείο Placement: Αυτό το αρχείο περιγράφει τη ρύθμιση της μνήμης cache σε όλους τους εξυπηρετητές ενός CDN. Κάθε 5 συνεχόμενες γραμμές του αρχείου αφορούν ένα εξυπηρετητή CDN.



Σχήμα 6. 6*: Παράδειγμα αρχείου Placement για 2 εξυπηρετητές CDN*

Περιγραφή ρυθμίσεων πρώτου εξυπηρετητή από το σχήμα 6.4

"0" : Ο αριθμός της ταυτότητας του εξυπηρετητή. Το αρχείο πρέπει να ταξινομείται σε αύξουσα σειρά με βάση αυτό τον αριθμό.

"1" : Ο CDNSim ορίζει ότι αυτό το πεδίο πρέπει να έχει πάντα την τιμή 1 χωρίς να περιγράφει τη χρήση του

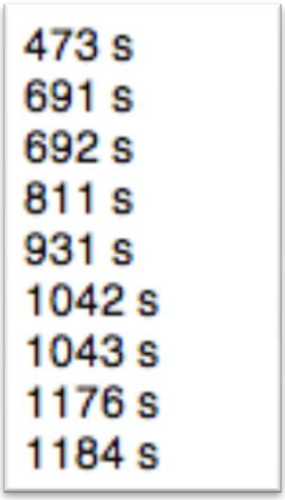
"s4037": Το filename του αρχείου που περιέχει τα αντικείμενα τα οποία φυλάσσονται στη μνήμη cache του εξυπηρετητή στην αρχή της προσωμοίωσης.

"LRU" : Η πολιτική αντικατάστασης των περιεχομένων της μνήμης cache

"1309752204.5": Το μέγεθος της μνήμης cache σε bytes.

Αρχείο s4037: Το αρχείο περιέχει γραμμές της μορφής "int char\n". Ο ακέραιος είναι η ταυτότητα του αντικειμένου που αποθηκεύεται στην αρχή της προσομοίωσης. Ο χαρακτήρας μπορεί να είναι είτε το 's' είτε το 'v'. Αν υπάρχει η τιμή 'v' αυτό σημαίνει ότι το αντικείμενο θα αποθηκευτεί στην αρχή της προσομοίωσης αλλά είναι πιθανόν να

αφαιρεθεί σε οποιαδήποτε στιγμή ανάλογα με την πολιτική της μνήμης cache, ενώ αν υπάρχει η τιμή ‘s’ αυτό σημαίνει ότι το αντικείμενο θα παραμείνει στη μνήμη cache από την αρχή μέχρι το τέλος της προσομοίωσης.



Σχήμα 6.7*: Τμήμα από το αρχείο s4037*

6.4 Σενάριο 1 (Μεταβολή του μεγέθους του διαδικτυακού χώρου)

Σε αυτό το σενάριο γίνεται σύγκριση του αλγορίθμου που περιγράφεται στο Κεφάλαιο 4 (utility) με τον il2p. Σε κάθε πείραμα μεταβάλλεται το μέγεθος της σελίδας ενώ οι υπόλοιπες παράμετροι παραμένουν σταθεροί. Από το σενάριο 1 θα παραχθούν 6 πειράματα: 2 (utility & il2p) για τις σελίδες των MSNBC, BBC και Sporting News.

Τοπολογία (AS Internet Topology)	3037 κόμβοι (δρομολογητές)
Εύρος ζώνης μεταξύ των κόμβων	1Gbit/sec
Πολιτική Δικτύου Παράδοσης Περιεχομένου	Cooperative Environment (closest surrogate)
Πλήθος εξυπηρετητών Βάση του [5]	Τμήμα 1 (United States): 61
	Τμήμα 2 (United Kingtom): 6
	Τμήμα 3 (Japan): 5
	Τμήμα 4 (Germany): 4
	Τμήμα 5 (Netherlands): 3
	Τμήμα 6 (France): 3
	Τμήμα 7 (Australia): 2
	Τμήμα 8 (Canada): 2
	Τμήμα 9 (Sweden): 2
	Τμήμα 10 (Hong Kong SAR): 1
	Τμήμα 11 (Others): 11
	Σύνολο = 100
Ενδιάμεση μνήμη εξυπηρετητών	Μέγεθος ενδιάμεσης μνήμης: Κάθε εξυπηρετητής θα έχει ενδιάμεση μνήμη ίση με το 25% του μεγέθους του ιστιακού χώρου.
	Cache replacement policy: LRU
Πλήθος Πηγαίων εξυπηρετητών	1
Πλήθος χρηστών	1000
Αριθμός των συνολικών αιτήσεων των χρστών . Το [8] βρίσκει ότι ο μέσος όρος των clicks που κάνει ένας χρήστης σε	10 000

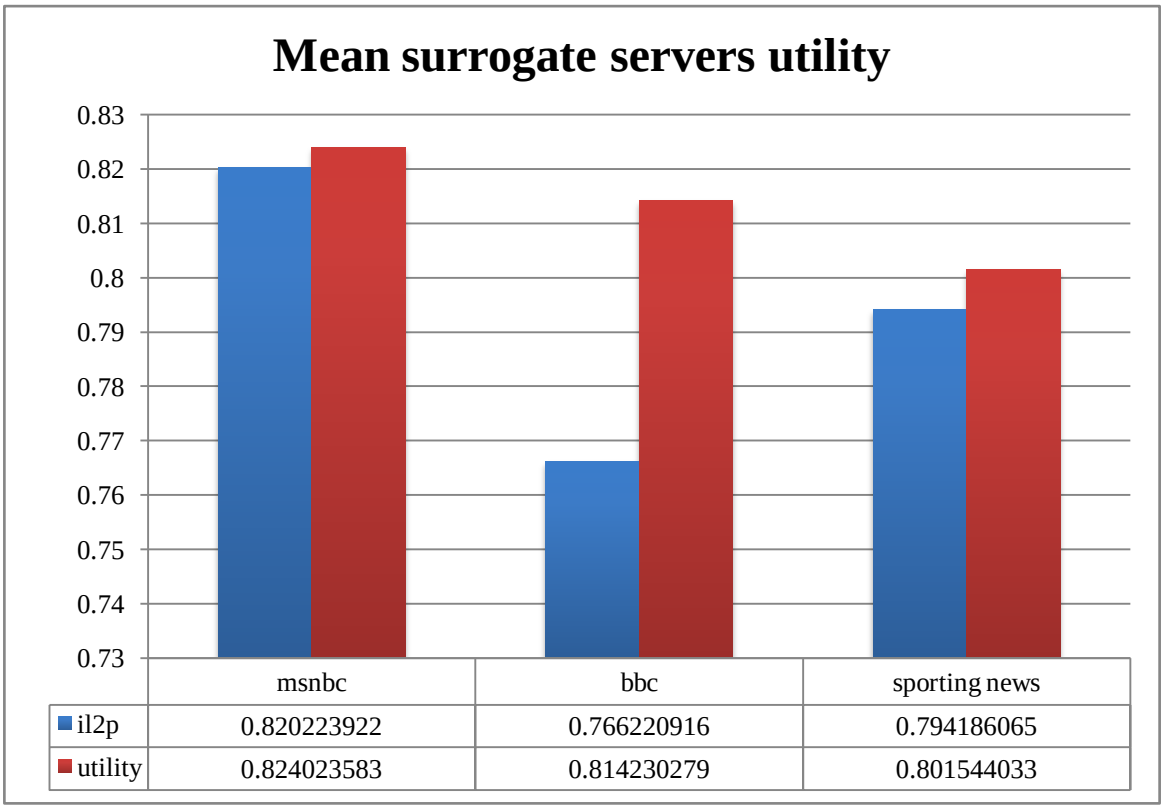
κάθε σελίδα είναι 10.	
Ιστιακός χώρος	Msnbc
	BBC
	Sporting News
	Total = 3
P (Κεφάλαιο 4)	0.5

Scenario 1: Simulation testbed

Γραφική Mean surrogate servers utility

Η γραφική παρουσιάζει τη μέση τιμή της μετρικής Utility των εξυπηρετητών CDN (surrogate servers) συναρτήσει των ιστιακών χώρων. Οι ιστιακοί χώροι είναι ταξινομημένοι σε αύξουσα σειρά από αριστερά προς τα δεξιά με βάση το πλήθος των αντικειμένων που περιέχουν.

Η μετρική Utility περιγράφηκε λεπτομερώς στο Κεφάλαιο 2.



	msnbc	bbc	sporting news
il2p	0.820223922	0.766220916	0.794186065
utility	0.824023583	0.814230279	0.801544033

Από τη γραφική παρατηρούμε ότι ο αλγόριθμος utility παρουσιάζει καλύτερα αποτελέσματα από τον il2p και για τους 3 ιστιακούς χώρους. Οι τιμές του Mean

surrogate server Utility για τους ιστοχώρους του msnbc και του sporting news παρουσιάζουν μικρή βελτίωση της τάξης του 1% ενώ για τον ιστοχώρο του bbc ο αλγόριθμος utility παρουσιάζει βελτίωση 5% έναντι του il2p. Αυτή η μεγάλη βελτίωση του utility στον ιστοχώρο του BBC για τον αλγόριθμο utility είναι πιθανόν να συμβαίνει λόγω της δομής του ιστοχώρου που επέλεξε να έχει το BBC η οποία βοηθά τους εξυπηρετητές να έχουν ψηλό utility.

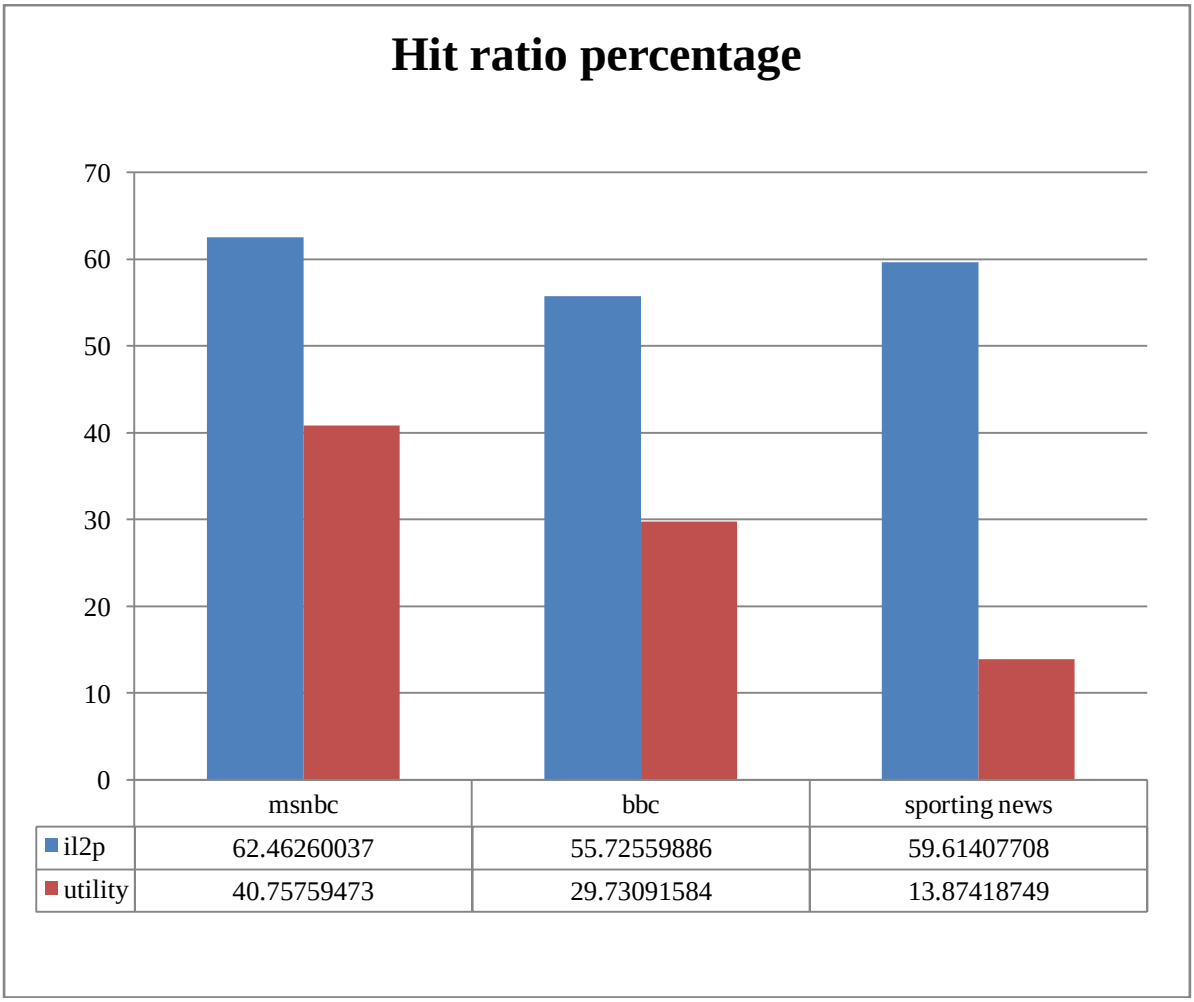
Αυτό σημαίνει ότι στη διάρκεια της προσομοίωσης ο αλγόριθμος utility κατά μέσο όρο έδωσε την ευκαιρία στους εξυπηρετητές Δικτύου Παράδοσης Περιεχομένου να κάνουν περισσότερο upload παρά download από δεδομένα.

Συμπέρασμα: Με βάση το συγκεκριμένο simulation testbed για ιστιακούς χώρους που έχουν διαφορετικό πλήθος αντικειμένων ο αλγόριθμος utility αποδίδει καλύτερα από τον il2p.

Γραφική Hit ratio percentage

Η γραφική παρουσιάζει την τιμή του ποσοστού Hit Ratio (%) των εξυπηρετητών του δικτύου παράδοσης περιεχομένου συναρτήσει των ιστιακών χώρων. Οι ιστιακοί χώροι είναι ταξινομημένοι σε αύξουσα σειρά από αριστερά προς τα δεξιά με βάση το πλήθος των αντικειμένων που περιέχουν.

Το Hit Ratio Percentage υπολογίζεται βάσει του number of hits / (number of hits + number of miss) / 100



	msnbc	bbc	sporting news
il2p	62.46260037	55.72559886	59.61407708
utility	40.75759473	29.73091584	13.87418749

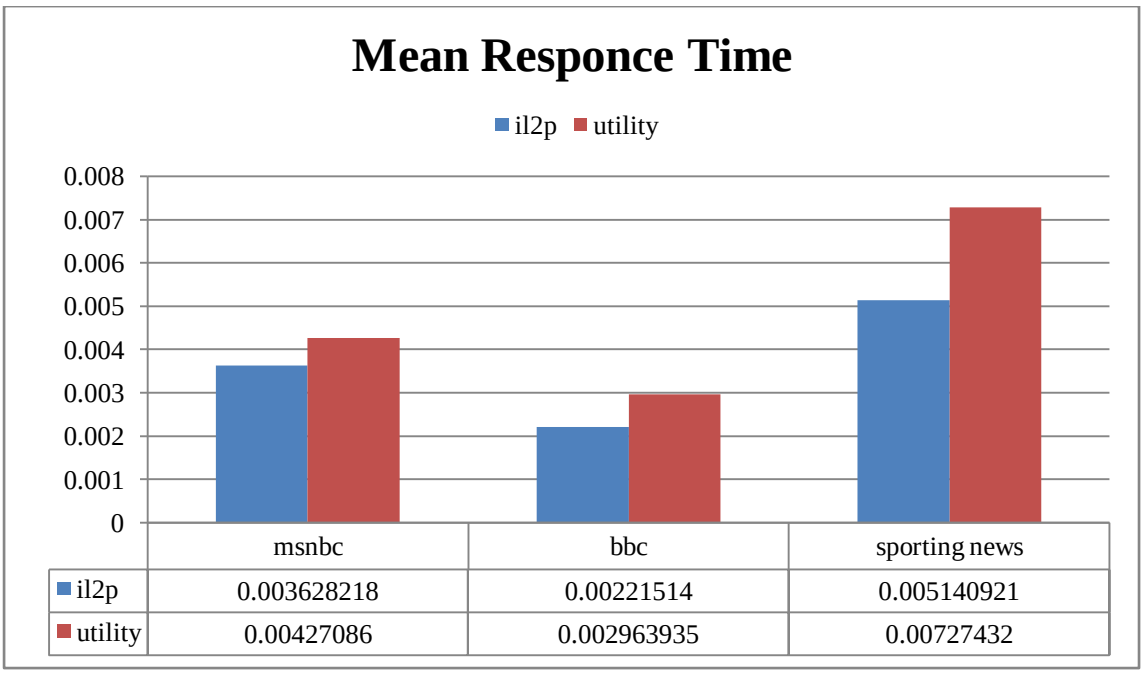
Από τη γραφική παρατηρούμε ότι ο αλγόριθμος il2p παρουσιάζει καλύτερα αποτελέσματα από τον αλγόριθμο utility και για τους 3 ιστιακούς χώρους. Παρατηρούμε επίσης ότι όσο μεγαλώνει το πλήθος των αντικειμένων στον ιστιακό χώρο τόσο μικραίνει το Hit Ratio Percentage.

Το χαμηλό Hit Ratio του αλγορίθμου utility μπορεί να εξηγηθεί από την γραφική **Mean surrogate servers utility.** Επειδή οι εξυπηρετητές όταν λάμβαναν μια αίτηση και δεν

μπορούσαν να την εξυπηρετήσουν (cache miss) παρέπεμπαν την αίτηση στον πιο κοντινό τους εξυπηρετητή (**Scenario 1: Simulation testbed :** CDN policy = Cooperative Environment (closest surrogate)). Ένας μεγάλος αριθμός από εξυπηρετητές είχε το αντικείμενο που ζητούσε η αίτηση και το έκανε upload με αποτέλεσμα να αυξάνονται τα δεδομένα που γίνονταν upload άρα να αυξάνεται και η μετρική Utility η οποία είναι ευθέως ανάλογη του αθροίσματος των δεδομένων (bytes) που κάνει upload ένας εξυπηρετητής.

Γραφική Mean Responce Time

Η γραφική παρουσιάζει το Mean Response Time (sec) των εξυπηρετητών συναρτήσει των ιστιακών χώρων. Οι ιστιακοί χώροι είναι ταξινομημένοι σε αύξουσα σειρά από αριστερά προς τα δεξιά με βάση το πλήθος των αντικειμένων που περιέχουν.
Το Response time είναι ο χρόνος που χρειάζεται για να σταλεί ένα σήμα από τον αποστολέα στον παραλήπτη συν το χρόνο που χρειάζεται για να απαντήσει ο παραλήπτης με ένα ACK ότι παρέλαβε το σήμα.



	msnbc	bbc	sporting news
il2p	0.003628218	0.00221514	0.005140921
utility	0.00427086	0.002963935	0.00727432

Παρατηρούμε ότι ο αλγόριθμος il2p παρουσιάζει καλύτερα αποτελέσματα στο response time και για τους 3 ιστοχώρους, η διαφορά του όμως από τον αλγόριθμο utility είναι μικρή (0.002 sec). Τα ψηλότερα Response times του αλγορίθμου utility πιθανόν να οφείλονται στο χρόνο που χρειάζεται το Δίκτυο παράδοσης περιεχομένου για να

αντιμετωπίσει τα cache misses. Επειδή η διαφορά του response time των 2 αλγορίθμων (il2p , utility) είναι μικρή αυτό είναι ένδειξη ότι τα αντικείμενα βρίσκονται σε πολύ κοντινούς εξυπηρετητές.

6.5 Σενάριο 2 (Μεταβολή του μεγέθους της ενδιάμεσης μνήμης του κάθε εξυπηρετητή)

Σε αυτό το σενάριο γίνεται σύγκριση του αλγορίθμου utility με τον il2p. Σε κάθε πείραμα μεταβάλλεται το μέγεθος της μνήμης cache του κάθε εξυπηρετητή ενώ οι υπόλοιπες παράμετροι παραμένουν σταθεροί. Το μέγεθος της μνήμης cache του κάθε εξυπηρετητή είναι σε μορφή ποσοστού του ιστοχώρου του BBC. Δηλαδή ο κάθε εξυπηρετητής έχει μέγεθος ενδιάμεσης μνήμης σε κάθε πείραμα το οποίο είναι ίσο με το ποσοστό του συνολικού μεγέθους του ιστοχώρου όπως περιγράφεται στο simulation testbed του σεναρίου 2.

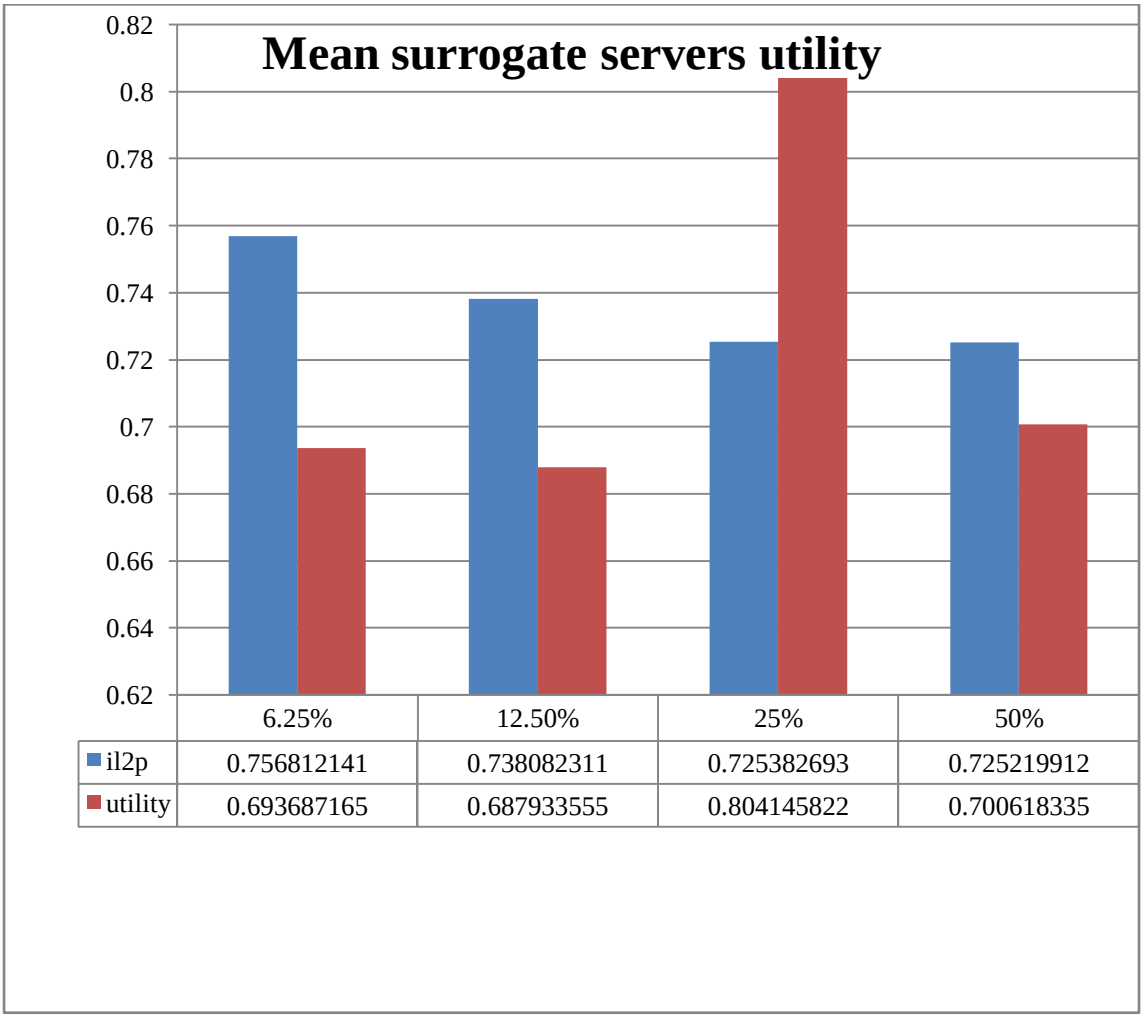
Τοπολογία (AS Internet Topology)	3037 κόμβοι (δρομολογητές)
Εύρος ζώνης μεταξύ των κόμβων	1Gbit/sec
Πολιτική Δικτύου Παράδοσης Περιεχομένου	Cooperative Environment (closest surrogate)
Πλήθος εξυπηρετητών Βάση του [5]	Τμήμα 1 (United States): 61
	Τμήμα 2 (United Kingtom): 6
	Τμήμα 3 (Japan): 5
	Τμήμα 4 (Germany): 4
	Τμήμα 5 (Netherlands): 3
	Τμήμα 6 (France): 3
	Τμήμα 7 (Australia): 2
	Τμήμα 8 (Canada): 2
	Τμήμα 9 (Sweden): 2
	Τμήμα 10 (Hong Kong SAR): 1
	Τμήμα 11 (Others): 11
	Σύνολο = 100
Ενδιάμεση μνήμη εξυπηρετητών	Μέγεθος ενδιάμεσης μνήμης: Κάθε εξυπηρετητής θα έχει ενδιάμεση μνήμη ίση με το 6.25%, 12,5%, 25%, 50% του μεγέθους του ιστιακού χώρου.
	Cache replacement policy: LRU
Πλήθος Πηγαίων εξυπηρετητών	1
Πλήθος χρηστών	1000

Αριθμός των συνολικών αιτήσεων των χρστών . Το [8] βρίσκει ότι ο μέσος όρος των clicks που κάνει ένας χρήστης σε κάθε σελίδα είναι 10.	10 000
Ιστιακός χώρος	BBC
P (Κεφάλαιο 4)	0.5

Scenario 2: Simulation testbed

Γραφική Mean surrogate servers utility

Η γραφική παρουσιάζει τη μέση τιμή της μετρικής Utility των εξυπηρετητών CDN συναρτήσει του μεγέθους της μνήμης cache των εξυπηρετητών. Οι ιστιακοί χώροι είναι ταξινομημένοι σε αύξουσα σειρά από αριστερά προς τα δεξιά με βάση το πλήθος των αντικειμένων που περιέχουν. Η μετρική Utility περιγράφηκε λεπτομερώς στο Κεφάλαιο 2.



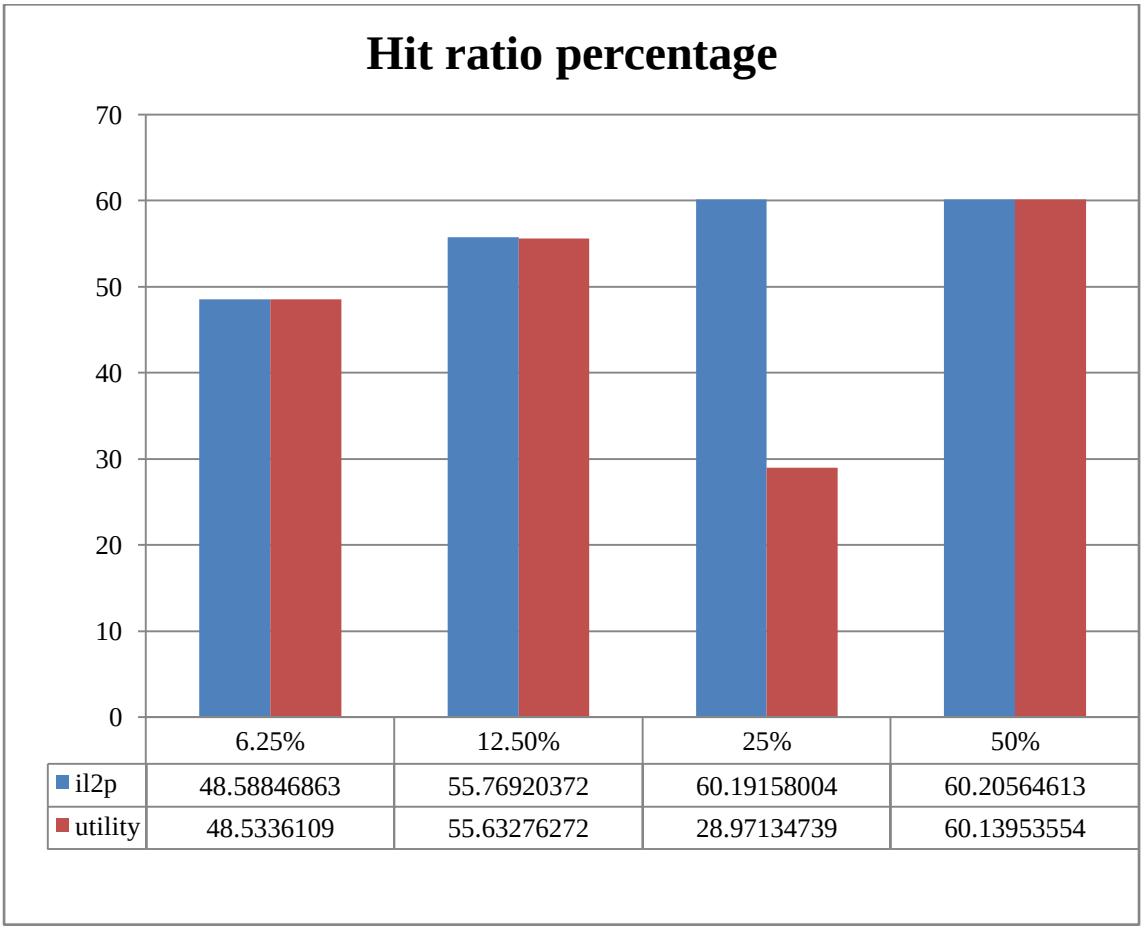
	6.25%	12.50%	25%	50%
il2p	0.756812141	0.738082311	0.725382693	0.725219912
utility	0.693687165	0.687933555	0.804145822	0.700618335

Παρατηρούμε ότι ο αλγόριθμος il2p για μέγεθος cache του κάθε εξυπηρετητή ίσο με 6,25%, 12,5% και 50% δίνει καλύτερα αποτελέσματα από τον αλγόριθμο utility για το mean surrogate servers utility. Βλέπουμε ότι για αυτές τις τιμές του cache size η διαφορά της τιμής utility που έχουν οι 2 αλγόριθμοι είναι μεταξύ 2 - 5%.

Για τιμή του cache size ίσο με 25% ο αλγόριθμος utility παράγει το καλύτερο αποτέλεσμα από όλες τις υπόλοιπες τιμές και επιτυγχάνει μια πολύ ψηλή τιμή που φτάνει το 0.8041 (η μέγιστη τιμή της μετρικής είναι 1).

Γραφική Hit ratio percentage

Η γραφική παρουσιάζει την τιμή του ποσοστού Hit Ratio (%) των εξυπηρετητών συναρτήσει του μεγέθους της ενδιάμεσης μνήμης των εξυπηρετητών.



	6.25%	12.50%	25%	50%
il2p	48.58846863	55.76920372	60.19158004	60.20564613
utility	48.5336109	55.63276272	28.97134739	60.13953554

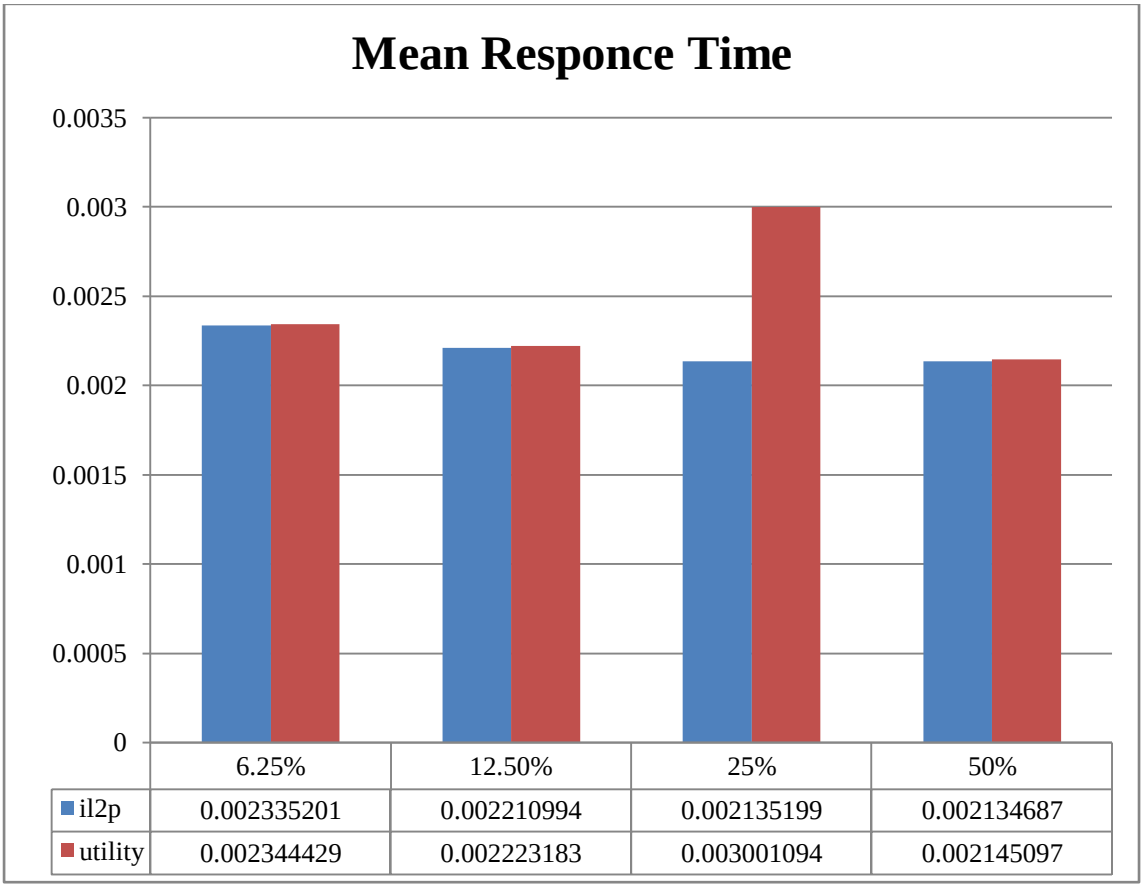
Για όλες τις τιμές του cache size εκτός του 25% οι αλγόριθμοι il2p και utility παρουσιάζουν πολύ κοντινές τιμές του hit ratio percentage που έχουν διαφορά της τάξης του 0.05%.

Για cache size = 25% το hit ratio percantage του αλγόριθμου utility μειώνεται στο 28 % (32% διαφορά από το αντίστοιχο του il2p).

Συμπέρασμα: Το χαμηλό hit ratio percentage για τον αλγόριθμο utility (25%) αναγκάζει τους εξυπηρετητές να κάνουν συχνά upload και έτσι να αυξηθεί το συνολικό Mean surrogate server utility που βρέθηκε στην πιο πάνω γραφική.

Γραφική Mean Responce Time

Η γραφική παρουσιάζει το Mean Response Time (sec) των εξυπηρετητών συναρτήσει του μεγέθους της ενδιάμεσης μνήμης των εξυπηρετητών.



	6.25%	12.50%	25%	50%
il2p	0.002335201	0.002210994	0.002135199	0.002134687
utility	0.002344429	0.002223183	0.003001094	0.002145097

Για όλες τις τιμές του cache size εκτός του 25% οι αλγόριθμοι il2p και utility παρουσιάζουν πολύ κοντινές τιμές του mean response time που έχουν διαφορά της τάξης του 0.0001 sec (Ο αλγόριθμος il2p παρουσιάζει τα καλύτερα αποτελέσματα).

Η μεγαλύτερη διαφορά συμβαίνει στο cache size = 25% στην οποία ο αλγόριθμος utility έχει τιμή 0.003 sec ενώ ο il2p 0.002 sec. Αυτή η διαφορά κάνει τον αλγόριθμο utility 0.001 sec πιο αργό από τον il2p για τον κάθε ένα από τους 1000 χρήστες στους οποίους έτρεξε η προσομοίωση. Πιθανός λόγος είναι το χαμηλό hit ratio percentage (πολλά cache misses) που αναγκάζει το δίκτυο να σπαταλήσει περισσότερο χρόνο για να εξυπηρετήσει ένα αίτημα. Αυτή η διαφορά δεν είναι μεγάλη και αυτό μας δίνει μια ένδειξη ότι τα αντικείμενα που αιτούνταν βρίσκονταν σε εξυπηρετητές που είχαν μικρή απόσταση (με βάση το χρόνο καθυστέρησης) από τους χρήστες που τα ζητούσαν.

6.6 Σενάριο 3 (Μεταβολή του αριθμού των εξυπηρετητών του Δικτύου Παράδοσης Περιεχομένου)

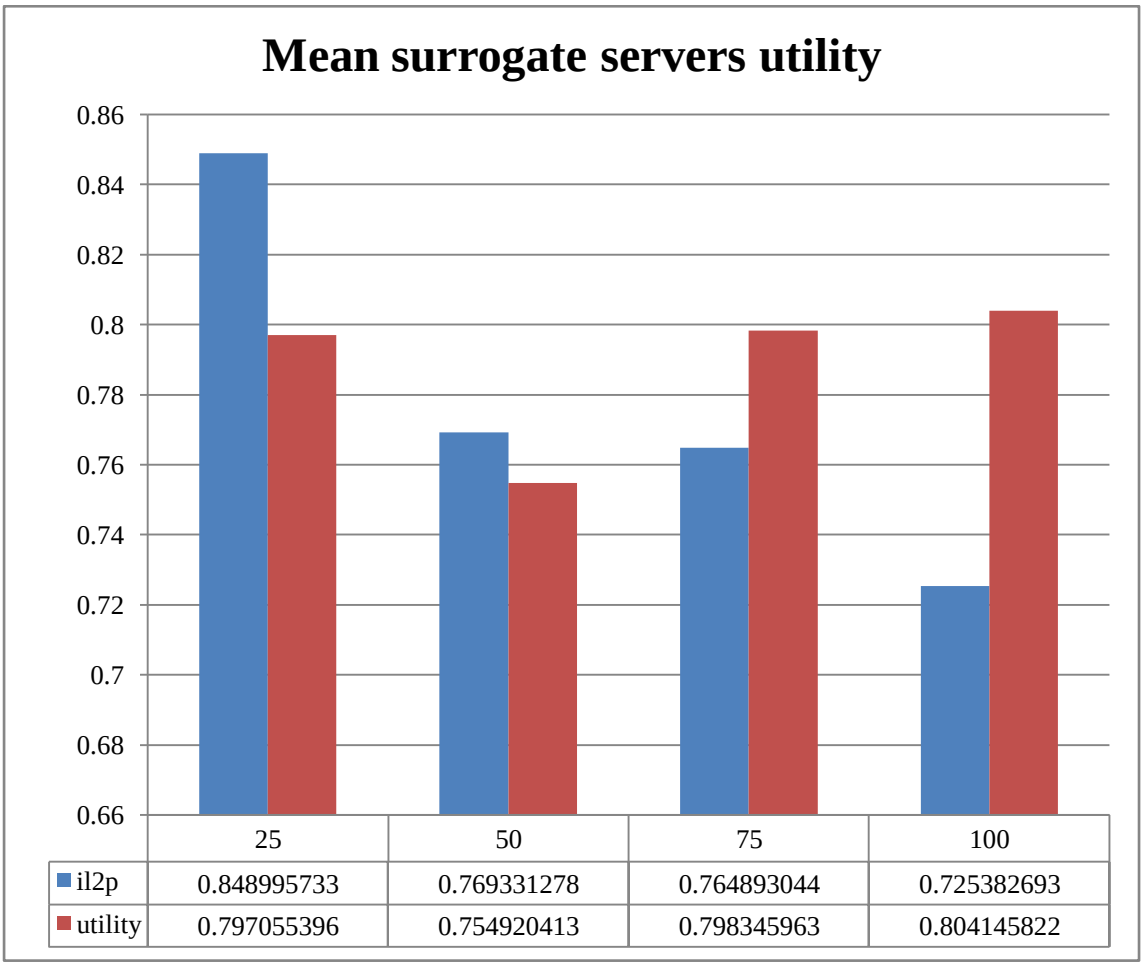
Σε αυτό το σενάριο γίνεται σύγκριση του αλγορίθμου utility με τον il2p. Σε κάθε πείραμα μεταβάλλεται ο αριθμός των εξυπηρετητών ενώ οι υπόλοιπες παράμετροι παραμένουν σταθεροί.

Τοπολογία (AS Internet Topology)	3037 κόμβοι (δρομολογητές)				
Εύρος ζώνης μεταξύ των κόμβων	1Gbit/sec				
Πολιτική Δικτύου Παράδοσης Περιεχομένου	Cooperative Environment (closest surrogate)				
Πλήθος εξυπηρετητών Βάση του [5]	Τμήμα 1 (United States)	15	31	46	61
	Τμήμα 2 (United Kingtom)	2	3	5	6
	Τμήμα 3 (Japan)	1	3	4	5
	Τμήμα 4 (Germany)	1	2	3	4
	Τμήμα 5 (Netherlands)	1	2	2	3
	Τμήμα 6 (France)	1	2	2	3
	Τμήμα 7 (Australia)	1	1	2	2
	Τμήμα 8 (Canada)	1	1	2	2
	Τμήμα 9 (Sweden)	1	1	2	2
	Τμήμα 10 (Hong Kong SAR)	0	1	1	1
	Τμήμα 11 (Others)	2	3	6	11
	Σύνολο	25	50	75	100
Ενδιάμεση μνήμη εξυπηρετητών	Μέγεθος ενδιάμεσης μνήμης: Κάθε εξυπηρετητής θα έχει ενδιάμεση μνήμη ίση με το 25% του μεγέθους του ιστιακού χώρου.				
	Cache replacement policy: LRU				
Πλήθος Πηγαίων	1				

εξυπηρετητών	
Πλήθος χρηστών	1000
Αριθμός των συνολικών αιτήσεων των χρστών . Το [8] βρίσκει ότι ο μέσος όρος των clicks που κάνει ένας χρήστης σε κάθε σελίδα είναι 10.	100 000
Ιστιακός χώρος	Sporting News
P (Κεφάλαιο 4)	0.5

Γραφική Mean surrogate servers utility

Η γραφική παρουσιάζει τη μέση τιμή της μετρικής Utility των εξυπηρετητών συναρτήσει του πλήθους των εξυπηρετητών.

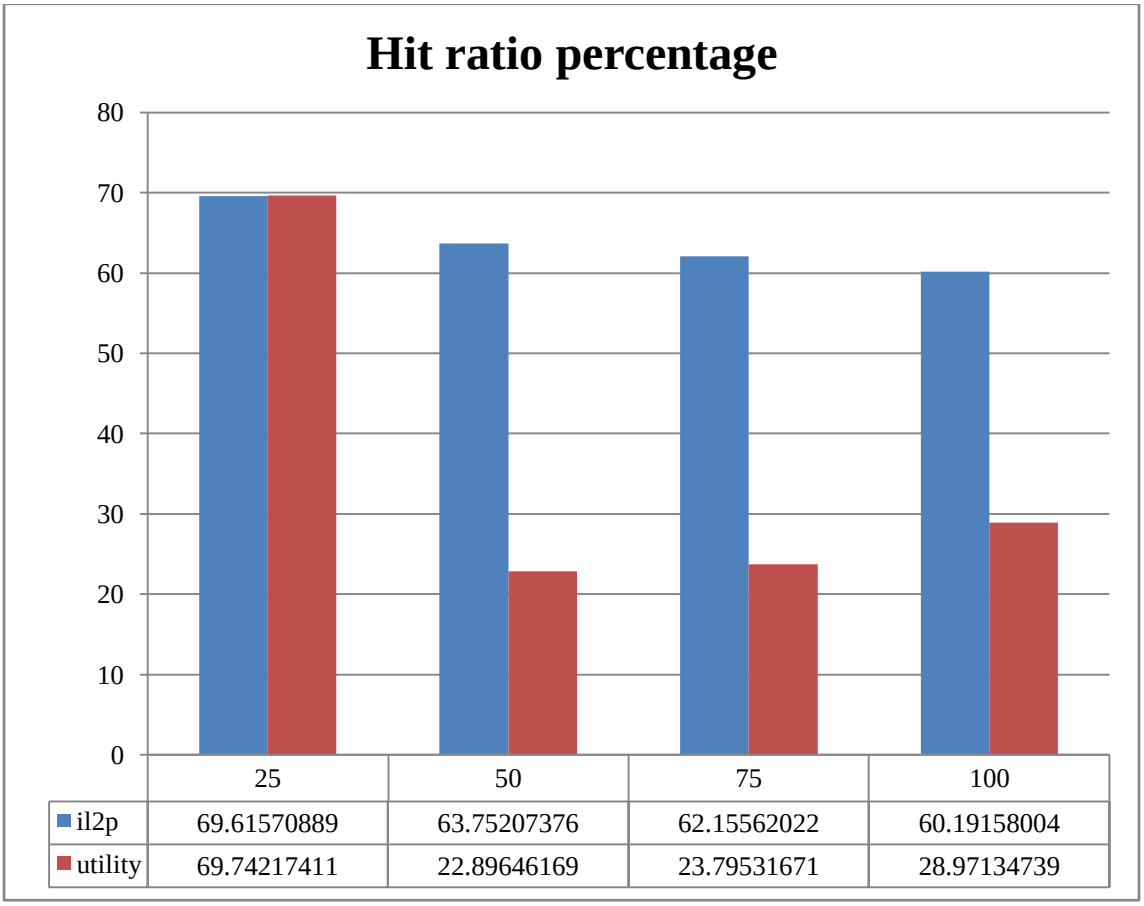


	25	50	75	100
il2p	0.848995733	0.769331278	0.764893044	0.725382693
utility	0.797055396	0.754920413	0.798345963	0.804145822

Παρατηρούμε ότι με 25 εξυπηρετητές ο αλγόριθμος il2p παρουσιάζει καλύτερα αποτελέσματα από τον αλγόριθμο utility. Καθώς όμως αυξάνεται ο αριθμός των εξυπηρετητών η διαφορά των mean surrogate server utility μεταξύ τους μειώνεται και για αριθμό εξυπηρετητών 75 και 100 ο αλγόριθμος utility δίνει καλύτερα αποτελέσματα. Επίσης καθώς ο αριθμός των εξυπηρετητών αυξάνει ο αλγόριθμος utility αυξάνει περισσότερο τη διαφορά του (θετική διαφορά) από τον il2p. Συμπέρασμα: Ο αλγόριθμος utility έχει καλύτερα αποτελέσματα από τον il2p για πραγματικά δεδομένα όπου ο αριθμός των εξυπηρετητών είναι μεγάλος.

Γραφική Hit ratio percentage

Η γραφική παρουσιάζει το Hit Ratio Percentage συναρτήσει του πλήθους των εξυπηρετητών

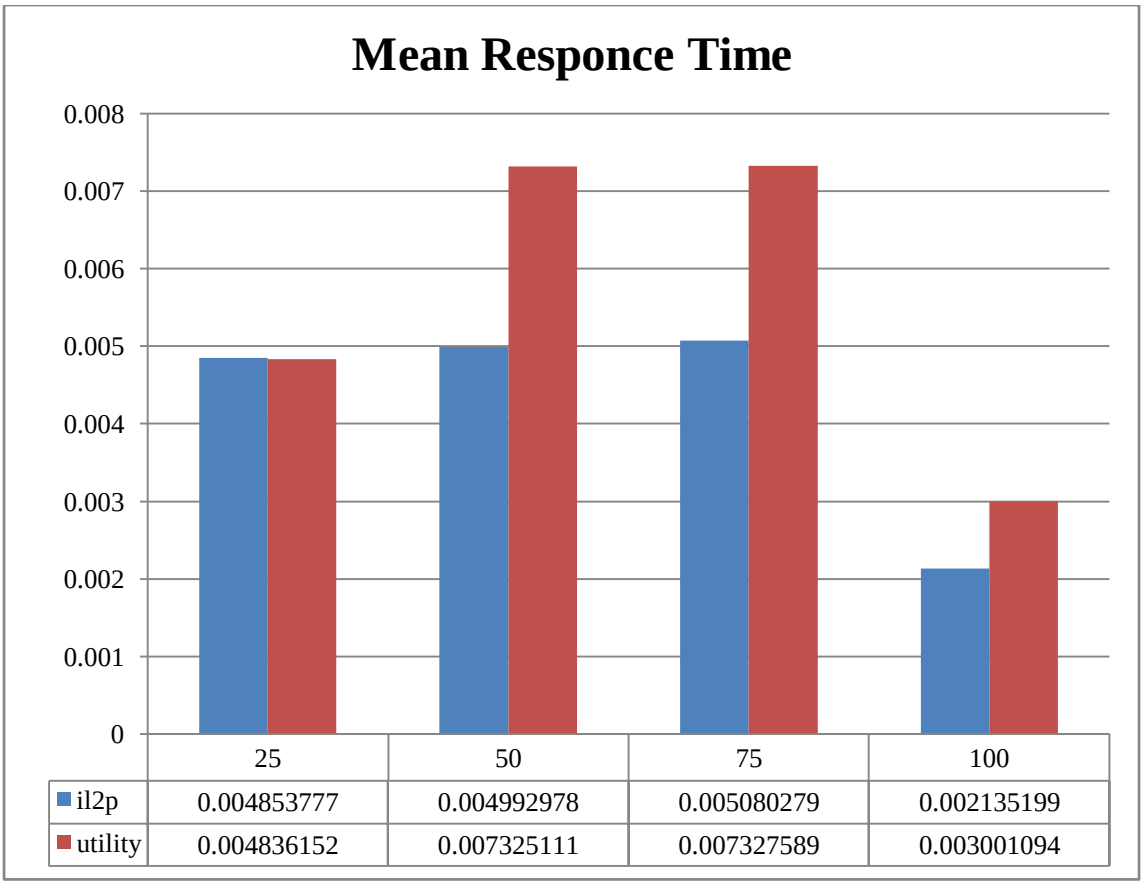


	25	50	75	100
il2p	69.61570889	63.75207376	62.15562022	60.19158004
utility	69.74217411	22.89646169	23.79531671	28.97134739

Παρατηρούμε χαμηλές τιμές του hit ratio percentage για 50, 75 και 100 εξυπηρετητές στον αλγόριθμο utility. Η χαμηλές τιμές δικαιολογούν τις ψηλές τιμές του αλγορίθμου utility στη γραφική mean surrogate servers utility. Επίσης παρατηρούμε μια αύξηση του hit ratio percentage για τον αλγόριθμο utility καθώς αυξάνεται ο αριθμός των εξυπηρετητών (από το 50 και μετά). Συμπέρασμα: Στον αλγόριθμο utility το hit ratio είναι ανάλογο του αριθμού των εξυπηρετητών.

Γραφική Mean Responce Time

Η γραφική παρουσιάζει το μέσο όρο απόκρισης συναρτήσει του πλήθους των εξυπηρετητών



	25	50	75	100
il2p	0.004853777	0.004992978	0.005080279	0.002135199
utility	0.004836152	0.007325111	0.007327589	0.003001094

Για τις τιμές 50, 75 και 100 εξυπηρετητών έχουμε ψηλό mean response time. Αυτό συμβαίνει εξ' αιτίας του χαμηλού hit ratio percentage. Στις τιμές όπου το hit ratio percentage είναι πιο χαμηλό (50, 75) έχουμε ψηλό mean response time για τους λόγους που εξηγήθηκαν και στο σενάριο 2.

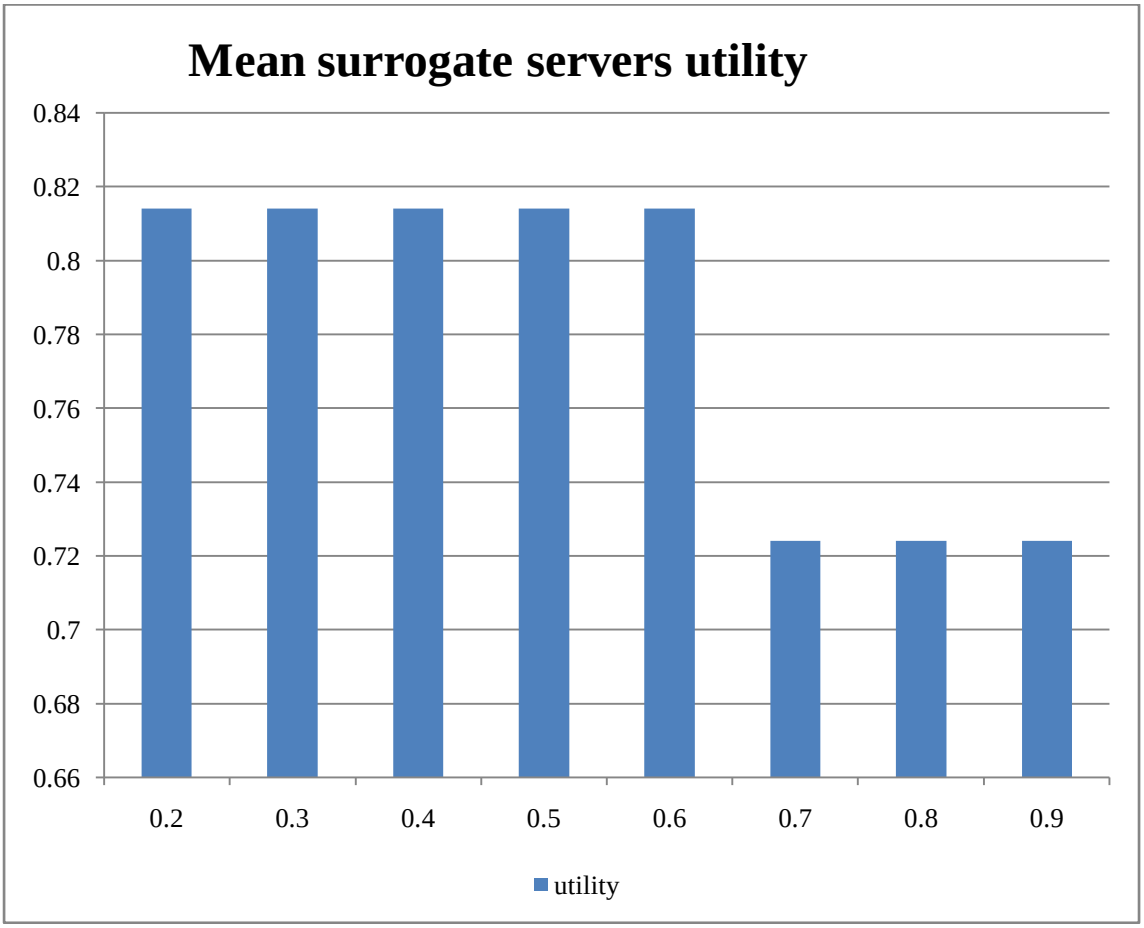
6.7 Σενάριο 4 (Μεταβολή της τιμής P)

Τοπολογία (AS Internet Topology)	3037 κόμβοι (δρομολογητές)
Εύρος ζώνης μεταξύ των κόμβων	1Gbit/sec
Πολιτική Δικτύου Παράδοσης Περιεχομένου	Cooperative Environment (closest surrogate)
Πλήθος εξυπηρετητών Βάση του [5]	Τμήμα 1 (United States): 61
	Τμήμα 2 (United Kingtom): 6
	Τμήμα 3 (Japan): 5
	Τμήμα 4 (Germany): 4
	Τμήμα 5 (Netherlands): 3
	Τμήμα 6 (France): 3
	Τμήμα 7 (Australia): 2
	Τμήμα 8 (Canada): 2
	Τμήμα 9 (Sweden): 2
	Τμήμα 10 (Hong Kong SAR): 1
	Τμήμα 11 (Others): 11
	Σύνολο = 100
Ενδιάμεση μνήμη εξυπηρετητών	Μέγεθος ενδιάμεσης μνήμης: Κάθε εξυπηρετητής θα έχει ενδιάμεση μνήμη ίση με το 25% του μεγέθους του ιστιακού χώρου.
	Cache replacement policy: LRU
Πλήθος Πηγαίων εξυπηρετητών	1
Πλήθος χρηστών	1000
Αριθμός των συνολικών αιτήσεων των χρστών . Το [8] βρίσκει ότι ο μέσος όρος των clicks που κάνει ένας χρήστης σε κάθε σελίδα είναι 10.	10 000

Ιστιακός χώρος	BBC
P (Κεφάλαιο 4)	0.1 - 0.9

Γραφική Mean surrogate servers utility

Η γραφική παρουσιάζει τη μέση τιμή της μετρικής Utility των εξυπηρετητών συναρτήσει των διαφορετικών τιμών της μεταβλητής P (Περιγράφηκε στο Κεφάλαιο 4). Μόνο ο αλγόριθμος utility χειρίζεται δυναμικά αντικείμενα και άρα χρησιμοποιεί τη μετρική P.



Παρατηρούμε ότι για τις τιμές του P από 0.2 μέχρι 0.6 για τον αλγόριθμο utility η τιμή του Mean surrogate servers utility παραμένει σταθερή. Από 0.7 και μετά μειώνεται και παραμένει σταθερή μέχρι το 0.9. Η τιμή 0.6 είναι σημείο καμπής για το Mean surrogate servers utility. Από τη λειτουργία του αλγορίθμου που περιγράφεται στο Κεφάλαιο 4

όσο πιο μικρή είναι η τιμή του P τόσο περισσότερα αντικείμενα θα αποθηκευτούν στις ενδιάμεσες μνήμες των εξυπηρετητών. Συμπέρασμα: Η μεταβλητή P πρέπει να έχει τιμές από 0.6 και κάτω για να αποθηκεύονται περισσότερα αντικείμενα στους εξυπηρετητές και να έχουμε ψηλότερες τιμές του Mean surrogate servers utility.

Κεφάλαιο 7

Γενικά Συμπεράσματα και Μελλοντική Εργασία

Γενικά Συμπεράσματα

- Ο αλγόριθμος utility επιτυγχάνει ψηλά αποτελέσματα του mean surrogate server utility μόνο όταν υπάρχει χαμηλό hit ratio percentage, ενώ ο il2p μπορεί να πετύχει ψηλά αποτελέσματα και των 2 τιμών (του mean surrogate server utility και hit ratio percentage) ταυτόχρονα
- Για συγκεκριμένες τιμές των μεταβλητών των 3 πρώτων σεναρίων ο αλγόριθμος utility πετυχαίνει σημαντικά καλύτερα αποτελέσματα από τον il2p.
- Για τις τιμές του Mean response time οι 2 αλγόριθμοι έχουν παρόμοια αποτελέσματα.
- Η μετρική P για τον αλγόριθμο utility πρέπει να θέτεται σε τιμές κάτω του 0.6 για να έχουμε ψηλότερο utility.

Μελλοντική Εργασία

- Πρέπει να τροποποιηθεί ο αλγόριθμος utility για να έχουμε καλύτερα αποτελέσματα στο mean surrogate server utility αλλά και hit ratio percentage ταυτόχρονα.
- Ο αλγόριθμος του Κεφαλαίου 5 πρέπει να τροποποιηθεί γιατί βρέθηκε ότι οι κόμβοι που δημιουργούνται στο κάθε τμήμα έχουν κοντινή (βάσει hops) απόσταση με τον τυχαίο κόμβο που επιλέγεται αλλά όχι μεταξύ τους. Πρέπει λοιπόν όλοι οι κόμβοι στο κάθε τμήμα να έχουν και κοντινή απόσταση μεταξύ τους.
- Πρέπει να γίνουν πειράματα για μεταβλητό αριθμό από χρήστες

Βιβλιογραφία

[1] G. Pallis, K. Stamos, A. Vakali, D. Katsaros, and A. Sidiropoulos, “Replication Based on Objects Load under a Content Distribution Network,” 22nd International Conference on Data Engineering Workshops (ICDEW’06), Atlanta, GA, USA: , pp. 53-53.

[2] S. Annapureddy, M. J. Freedman, and D. Mazières, “Shark: Scaling File Servers via Cooperative Caching”, Proceedings of the 2nd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), Boston, USA, May 2005.

[3] K. Stamos, G. Pallis, A. Vakali, and M.D. Dikaiakos, “Evaluating the utility of content delivery networks,” Proceedings of the 4th edition of the UPGRADE-CN workshop on Use of P2P, GRID and agents for the development of content networks - UPGRADE-CN ’09, Garching, Germany: 2009, p. 11.

[4] G. Pallis, A. Vakali, K. Stamos, A. Sidiropoulos, D. Katsaros, and Y. Manolopoulos, “A Latency-Based Object Placement Approach in Content Distribution Networks,” Third Latin American Web Congress (LA-WEB’2005), Buenos Aires, Argentina: , pp. 140-147.

[5] C. Huang, A. Wang, J. Li, and K. W. Ross. Measuring and evaluating large-scale cdns (paper withdrawn). In IMC ’08, pages 15–29, New York, NY, USA, 2008. ACM.

[6] E. Zegura, K. Calvert, and S. Bhattacharjee, “How to Model an Internetwork”, Proceedings of the Conference on Computer Communications, 15th Annual Joint Conference of the IEEE Computer and Communications Societies, Networking the Next Generation (IEEE INFOCOM), San Francisco, USA, Mar. 1996, pp. 594-602

[7] J. Kangasharju, “Object replication strategies in content distribution networks,” Computer Communications, vol. 25, 2002, pp. 376-383.

[8] B.A. Huberman, “Strong Regularities in World Wide Web Surfing,” Science, vol. 280, 1998, pp. 95-97.

[9]V. N. Padmanabhan and L. Qiu. The content and access dynamics of a busy web site: findings and implications. In SIGCOMM ’00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pages 111–123, New York, NY, USA, 2000. ACM.

[10] K. Stamos, G. Pallis, A. Vakali, D. Katsaros, A. Sidiropoulos, Y. Manolopoulos: "CDNsim: A Simulation Tool for Content Distribution Networks", ACM Transactions on Modeling and Computer Simulation, accepted, Feb. 2009.

[11] A. Sidiropoulos, G. Pallis, D. Katsaros, K. Stamos, A. Vakali, and Y. Manolopoulos, “Prefetching in Content Distribution Networks via Web Communities Identification and Outsourcing,” World Wide Web, vol. 11, 2007, pp. 39-70.

[12] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos, “A data mining algorithm for generalized web prefetching,” IEEE Transactions on Knowledge and Data Engineering, vol. 15, 2003, pp. 1155-1169.

[13] A. Luotonen and K. Altis, “World-Wide Web proxies,” Computer Networks and ISDN Systems, vol. 27, 1994, pp. 147-154.

[14] V.N. Padmanabhan and G.M. Voelker, with Lili Qiu, “On the placement of Web server replicas,” Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213), Anchorage, AK, USA: , pp. 1587-1596.

[15] A. Vakali and G. Pallis, “Content delivery networks: Status and trends,” IEEE Internet Computing, vol. 7, 2003, pp. 68-74.

[16] K. Sohraby, with Bo Li and Xin Deng, “On the optimal placement of web proxies in the Internet,” IEEE INFOCOM ’99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of th ζe IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320), New York, NY, USA: 1999, pp. 1282-1290 vol.3.

[17] M. Szymaniak, G. Pierre, and M. Van Steen, “Latency- Driven Replica Placement”, Proceedings of the International Symposium on Applications and the Internet (SAINT), Trento, Italy, Feb. 2005, pp. 399-405.

[18] R.H. Katz, with Yan Chen and Lili Qiu and Weiyu Chen and Luan Nguyen, “Efficient and adaptive web replication using content clustering,” IEEE Journal on Selected Areas in Communications, vol. 21, 2003, pp. 979-994.

[19] K. Hosanagar, J. Chuang, R. Krishnan, and M.D. Smith, “Service Adoption and Pricing of Content Delivery Network (CDN) Services,” Management Science, vol. 54, 2008, pp. 1579-1593.

[20] R. Buyya, M. Pathan, and A. Vakali, Eds., Content Delivery Networks, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

Παράρτημα Α

Αλγόριθμος για τοποθέτηση αντικειμένων στους εξυπηρετητές CDN που βασίζεται στη μετρική Utility.

```
public static void algorith_Maximize_Utlility() throws IOException {
    start_timer();

    calculate_cost_surrogateTosurrogate();
    calculate_cost_surrogateToclient();
    out.println("Utility");

    int numOfObjects = objArray.size();
    int numOfSurrogates = s_graph_pos.size();
    int numOfClients = c_graph_pos.size();

    double cost_StartNode_EndNode = 0;
    int clnt = 0;

    double object_size = 0;
    float expire_probability = (float) 0.0;


    for (int i = 0; i < numOfObjects; i++) {


      int object = objArray.get(i).num;// OBJECTS   i
      object_size = objArray.get(i).size;
      expire_probability = objArray.get(i).expireProbability;
      for (int k = 0; k < numOfSurrogates; k++) {// SUROGATES k
        int srgt = s_graph_pos.get(k);
```

```
            G.surrogate_cost[srgt] = 0;
            for (int l = 0; l < numOfClients; l++) { // CLIENTS   l
               cost_StartNode_EndNode = 0;
               clnt = c_graph_pos.get(l);
               object = objArray.get(i).num;
               if (G.requestsObject(clnt, object)) {

                  cost_StartNode_EndNode = costFromEachSurogateToClient[srgt][clnt];

                  G.surrogate_cost[srgt] += cost_StartNode_EndNode;

               }
            }
            //out.println("Obj = " + object + " Sur =  " + hash_num_name.get(srgt) + "
Cost = " + G.surrogate_cost[srgt]);

            //  G.surrogate_cost[srgt] /= object_size; // Cost is the transmission delay
            double sum_Mbits_per_sec = G.surrogate_cost[srgt];
            double bytes_per_second = sum_Mbits_per_sec / 8 * 1024 * 1024;
            G.surrogate_cost[srgt] = object_size / bytes_per_second;


         }

         G.setAllMinimumCostFalse();
         double max_cost = G.surrogate_cost[s_graph_pos.get(0)];
         double sur_m_cost = 0;



         for (int m = 0; m < numOfSurrogates; m++) {
            sur_m_cost = G.surrogate_cost[s_graph_pos.get(m)];
            if (max_cost < sur_m_cost && sur_m_cost != 0) {
```

```
            max_cost = sur_m_cost;

        }
    }


    for (int m = 0; m < numOfSurrogates; m++) {
        int sur = s_graph_pos.get(m);
        sur_m_cost = G.surrogate_cost[sur];

        if (sur_m_cost == max_cost) {
            G.setMinimumCost(sur);
        }
    }


    for (int m = 0; m < numOfSurrogates; m++) {
        int sur = s_graph_pos.get(m);

        if (G.hasMinimumCost(sur) == false) {
            // G.removeObject(sur, obj);
        } else if (G.hasMinimumCost(sur) == true) {
            if (G.hasObject(sur, object) == false) {
                G.downloadObject(sur, object);
            }
        }
    }
}

out.printf("--- End of Phase 1 ---\n");


for (int i = 0; i < numOfObjects; i++) {
```

```
//          out.printf("Object " + i + "\n");

        int object = objArray.get(i).num;// OBJECTS   i
        object_size = objArray.get(i).size;


        for (int m = 0; m < numOfSurrogates; m++) {
           int sur_a = s_graph_pos.get(m);
           if (G.hasObject(sur_a, object)) {
              for (int n = 0; n < numOfSurrogates; n++) {

                 int sur_b = s_graph_pos.get(n);
                 if (G.hasObject(sur_b, object) && sur_a != sur_b) {
                    // int origin = o_graph_pos.get(0); // ONE Origin ------- MAY
CHANGE


                    double                    sum_Mbits_per_sec                    =
costFromEachSurrogateToEachSurrogate[sur_a][sur_b];
                    double bytes_per_second = sum_Mbits_per_sec / 8 * 1024 * 1024;
                    double cost_sura_surb = object_size / bytes_per_second;


                    sum_Mbits_per_sec = costFromOriginToEachSurrogate[sur_b];
                    bytes_per_second = sum_Mbits_per_sec / 8 * 1024 * 1024;
                    double cost_origin_surb = object_size / bytes_per_second;



                    if (cost_sura_surb <= cost_origin_surb) {

                       G.uploadObject(sur_a, object);
```

```
                }
            }

        }
        G.calculateUtility(sur_a);
    }
}

double max_Utility = G.surrogate_utility[s_graph_pos.get(0)];

for (int m = 0; m < numOfSurrogates; m++) {
    double cur_Utility = G.surrogate_utility[s_graph_pos.get(m)];
    if (max_Utility <= cur_Utility) {
        max_Utility = cur_Utility;
    }
}


for (int m = 0; m < numOfSurrogates; m++) {
    int srgt = s_graph_pos.get(m);
    if (G.hasObject(srgt, object)) {
        double cur_Utility = G.surrogate_utility[srgt];
        if (max_Utility > cur_Utility) {
//                  out.print("obj " + object + " removed!\n");
            if (expire_probability > P) {
                G.removeObject(srgt, object);
            }

        }
    }
}
```

```
    }

    end_timer("utility");
    print_ToFile_Or_Screen("utility");


}
```

Αλγόριθμος Lat-cdn

```
public static void algorith_Lat_Cdn() throws IOException {
        start_timer();
//        calculate_cost_surrogateTosurrogate();
        calculate_cost_surrogateToclient();
        out.println("Lat-Cdn");



        int numOfObjects = objArray.size();
        int numOfSurrogates = s_graph_pos.size();
        int numOfClients = c_graph_pos.size();

        double cost_StartNode_EndNode = 0;
        int clnt = 0;



        ArrayList<Sur_Cost>[] cost_obj = new ArrayList[numOfObjects];
        ArrayList<Integer>[] min_cost_sur = new ArrayList[numOfObjects];

        for (int i = 0; i < numOfObjects; i++) {
           cost_obj[i] = new ArrayList<Sur_Cost>();
           min_cost_sur[i] = new ArrayList<Integer>();
        }



        for (int i = 0; i < numOfObjects; i++) {        // OBJECTS   i

//           out.printf("Object " + i + "\n");



           int object = objArray.get(i).num;
```

```
        double object_size = objArray.get(i).size;


        for (int k = 0; k < numOfSurrogates; k++) {// SUROGATES k

          int srgt = s_graph_pos.get(k);

          if (setDEBUG) {
            out.printf("    srgt " + srgt + "\n");
          }

          G.surrogate_cost[srgt] = 0;
          if (G.hasObject(srgt, object) == false) {

            for (int l = 0; l < numOfClients; l++) { // CLIENTS   l

              cost_StartNode_EndNode = 0;
              clnt = c_graph_pos.get(l);

              if (setDEBUG) {
                out.printf("        clnt =  " + clnt + " req = " + G.requestsObject(clnt,
object) + "\n");
              }
              if (G.requestsObject(clnt, object)) {

                cost_StartNode_EndNode                                    =
costFromEachSurogateToClient[srgt][clnt];

                G.surrogate_cost[srgt] += cost_StartNode_EndNode;
              }
            }

          }
```

```
        //G.surrogate_cost[srgt] /= object_size;
        double sum_Mbits_per_sec = G.surrogate_cost[srgt];
        double bytes_per_second = sum_Mbits_per_sec / 8 * 1024 * 1024;
        G.surrogate_cost[srgt] = object_size / bytes_per_second;


        Sur_Cost sur_cost = new Sur_Cost();
        sur_cost.cost = G.surrogate_cost[srgt];
        sur_cost.sur_num = srgt;

        cost_obj[object].add(sur_cost);

    }//End -> Surogates

}// End -> Objects


double max;
int max_cost_sur_num = 0;
for (int i = 0; i < numOfObjects; i++) {

    max = cost_obj[i].get(0).cost;

    for (int j = 0; j < numOfSurrogates; j++) {
        if (max < cost_obj[i].get(j).cost) {
            max = cost_obj[i].get(j).cost;
            max_cost_sur_num = cost_obj[i].get(j).sur_num;
        }
    }

    min_cost_sur[i].add(max_cost_sur_num);
```

```
            // SOL 2
//            for (int j = 0; j < numOfSurrogates; j++) {
//
//
//                double sur_cost = cost_obj[i].get(j).cost;
//
//                if (sur_cost == max) {
//                    min_cost_sur[i].add(cost_obj[i].get(j).sur_num);
//                }
//
//            }
            // SOL 2


        }


        for (int i = 0; i < numOfObjects; i++) {
            int object = objArray.get(i).num;
            for (int j = 0; j < min_cost_sur[i].size(); j++) {
                int sur = min_cost_sur[object].get(j);
                G.downloadObject(sur, object);
            }
        }

        end_timer("lat_cdn");

        print_ToFile_Or_Screen("lat_cdn");


    }// End -> algorith_Lat_Cdn
```

Αλγόριθμος il2p

```
public static void algorith_Lat_Cdn() throws IOException {
        start_timer();
//        calculate_cost_surrogateTosurrogate();
        calculate_cost_surrogateToclient();
        out.println("Lat-Cdn");



        int numOfObjects = objArray.size();
        int numOfSurrogates = s_graph_pos.size();
        int numOfClients = c_graph_pos.size();

        double cost_StartNode_EndNode = 0;
        int clnt = 0;



        ArrayList<Sur_Cost>[] cost_obj = new ArrayList[numOfObjects];
        ArrayList<Integer>[] min_cost_sur = new ArrayList[numOfObjects];

        for (int i = 0; i < numOfObjects; i++) {
           cost_obj[i] = new ArrayList<Sur_Cost>();
           min_cost_sur[i] = new ArrayList<Integer>();
        }



        for (int i = 0; i < numOfObjects; i++) {        // OBJECTS   i

//           out.printf("Object " + i + "\n");



           int object = objArray.get(i).num;
```

```
        double object_size = objArray.get(i).size;


        for (int k = 0; k < numOfSurrogates; k++) {// SUROGATES k

          int srgt = s_graph_pos.get(k);

          if (setDEBUG) {
            out.printf("    srgt " + srgt + "\n");
          }

          G.surrogate_cost[srgt] = 0;
          if (G.hasObject(srgt, object) == false) {

            for (int l = 0; l < numOfClients; l++) { // CLIENTS   l

              cost_StartNode_EndNode = 0;
              clnt = c_graph_pos.get(l);

              if (setDEBUG) {
                out.printf("        clnt =  " + clnt + " req = " + G.requestsObject(clnt,
object) + "\n");
              }
              if (G.requestsObject(clnt, object)) {

                cost_StartNode_EndNode                                    =
costFromEachSurogateToClient[srgt][clnt];

                G.surrogate_cost[srgt] += cost_StartNode_EndNode;
              }
            }

          }
```

```
        //G.surrogate_cost[srgt] /= object_size;
        double sum_Mbits_per_sec = G.surrogate_cost[srgt];
        double bytes_per_second = sum_Mbits_per_sec / 8 * 1024 * 1024;
        G.surrogate_cost[srgt] = object_size / bytes_per_second;


        Sur_Cost sur_cost = new Sur_Cost();
        sur_cost.cost = G.surrogate_cost[srgt];
        sur_cost.sur_num = srgt;

        cost_obj[object].add(sur_cost);

    }//End -> Surogates

}// End -> Objects


double max;
int max_cost_sur_num = 0;
for (int i = 0; i < numOfObjects; i++) {

    max = cost_obj[i].get(0).cost;

    for (int j = 0; j < numOfSurrogates; j++) {
        if (max < cost_obj[i].get(j).cost) {
            max = cost_obj[i].get(j).cost;
            max_cost_sur_num = cost_obj[i].get(j).sur_num;
        }
    }

    min_cost_sur[i].add(max_cost_sur_num);
```

```
            // SOL 2
//            for (int j = 0; j < numOfSurrogates; j++) {
//
//
//                double sur_cost = cost_obj[i].get(j).cost;
//
//                if (sur_cost == max) {
//                    min_cost_sur[i].add(cost_obj[i].get(j).sur_num);
//                }
//
//            }
            // SOL 2


        }


        for (int i = 0; i < numOfObjects; i++) {
            int object = objArray.get(i).num;
            for (int j = 0; j < min_cost_sur[i].size(); j++) {
                int sur = min_cost_sur[object].get(j);
                G.downloadObject(sur, object);
            }
        }

        end_timer("lat_cdn");

        print_ToFile_Or_Screen("lat_cdn");


    }// End -> algorith_Lat_Cdn
```

Αλγόριθμος για τοποθέτηση των εξυπηρετητών CDN στην τοπολογία του διαδικτύου

```
public class SurrogatePlacement {

   public static WeightedGraph2 G;
   private static String[][] n1n2cost;
   private static boolean COSTS_ARE_DONE = true;
   private static String linkSpeed = "1000000000";


   private static final int NUM_OF_CLIENTS = 1000;

   private static final int NUM_OF_SUROGATES = 75;

   public static void main(String[] args) throws FileNotFoundException, IOException {

      String[] strLineArray;
      String strNode1, strNode2, strCost;

      String strLine;

      if (!COSTS_ARE_DONE) {
         String file_asTopology =
"/Users/theodor/Documents/PLIROFORIKH/7th_Semester/diplomatikh/algorithm/Topo
logies/as3037.txt";

         BufferedReader in = new BufferedReader(new FileReader(file_asTopology));

         HashMap<String, String> routers = new HashMap<String, String>();

         strLine = in.readLine();

         while (strLine != null) {
            strLineArray = strLine.split(" ");
            strNode1 = strLineArray[0];
            strNode2 = strLineArray[1];

            routers.put(strNode1, strNode1);
            routers.put(strNode2, strNode2);

            strLine = in.readLine();

         }


         G = new WeightedGraph2(routers.size());
```

```
        ArrayList arrRouters = new ArrayList(routers.values());

        HashMap<String, Integer> hash_name_num = new HashMap<String,
Integer>();
        HashMap<Integer, String> hash_num_name = new HashMap<Integer,
String>();

        for (int i = 0; i < routers.size(); i++) {
           hash_name_num.put(arrRouters.get(i).toString(), i);
           hash_num_name.put(i, arrRouters.get(i).toString());
        }


        for (int i = 0; i < routers.size(); i++) {
           out.println(i + " " + hash_num_name.get(i));
           G.setLabel(i, hash_num_name.get(i));
        }

        in.close();
        in = new BufferedReader(new FileReader(file_asTopology));
        strLine = in.readLine();
        while (strLine != null) {

           strLineArray = strLine.split(" ");
           strNode1 = strLineArray[0];
           strNode2 = strLineArray[1];


           G.addEdge(hash_name_num.get(strNode1), hash_name_num.get(strNode2),
1);
           G.addEdge(hash_name_num.get(strNode2), hash_name_num.get(strNode1),
1);

           strLine = in.readLine();
        }



        n1n2cost = new String[routers.size()][3];

        String strN1 = hash_num_name.get(hash_name_num.get("5"));
        String strN2;
        int n1, n2;
        n1 = hash_name_num.get("5");

        for (int i = 0; i < routers.size(); i++) {

           n1n2cost[i][0] = strN1;
```

```
            strN2 = hash_num_name.get(i);
            n1n2cost[i][1] = strN2;


            n2 = hash_name_num.get(strN2);
            Double cost = G.calculateCost_Single_Path(G, n1, n2);
            n1n2cost[i][2] = cost.toString();
//            System.out.println(i);
         }

         for (int i = 0; i < routers.size(); i++) {
            System.out.println(n1n2cost[i][0] + " " + n1n2cost[i][1] + " " +
n1n2cost[i][2]);
         }
      }


      String file_N1_N2_Costs =
"/Users/theodor/Documents/PLIROFORIKH/7th_Semester/diplomatikh/algorithm/Nod
e5_Costs/node5costs.txt";
      n1n2cost = new String[3037][3];
      BufferedReader in = new BufferedReader(new FileReader(file_N1_N2_Costs));

      strLine = in.readLine();

      int i = 0;
      while (strLine != null) {
         strLineArray = strLine.split(" ");
         strNode1 = strLineArray[0];
         strNode2 = strLineArray[1];
         strCost = strLineArray[2];

         n1n2cost[i][0] = strNode1;
         n1n2cost[i][1] = strNode2;
         n1n2cost[i][2] = strCost;


         strLine = in.readLine();
         i++;
      }


      for (int j = 0; j < n1n2cost.length; j++) {
         for (int k = 0; k < n1n2cost.length; k++) {
            Double c1 = Double.parseDouble(n1n2cost[j][2]);
            Double c2 = Double.parseDouble(n1n2cost[k][2]);
            if (c1 <= c2) {
               String n1 = n1n2cost[j][0];
               String n2 = n1n2cost[j][1];
```

```
                String cost = n1n2cost[j][2];

                n1n2cost[j][0] = n1n2cost[k][0];
                n1n2cost[j][1] = n1n2cost[k][1];
                n1n2cost[j][2] = n1n2cost[k][2];

                n1n2cost[k][0] = n1;
                n1n2cost[k][1] = n2;
                n1n2cost[k][2] = cost;

            }
        }
    }


//      for (int j = 0; j < n1n2cost.length; j++) {
//          System.out.println(n1n2cost[j][0] + " " + n1n2cost[j][1] + " " + n1n2cost[j][2]);
//      }

    int[] percentGroups = {61, 6, 5, 4, 3, 3, 2, 2, 2, 1, 11};

    int sum = 0;
    for (int j = 0; j < percentGroups.length; j++) {
        sum += percentGroups[j];
    }

    ArrayList<String>[] groups = new ArrayList[percentGroups.length];

    for (int j = 0; j < groups.length; j++) {
        groups[j] = new ArrayList<String>();
    }

    int c2 = 0;
    for (int j = 0; j < percentGroups.length; j++) {
        int groupCount = Math.round((float) 3037 * percentGroups[j] / sum);
        for (int k = 0; k < groupCount; k++) {
            groups[j].add(n1n2cost[c2][1]);
            c2++;
        }
    }


    in.close();
    String file_asTopology =
"/Users/theodor/Documents/PLIROFORIKH/7th_Semester/diplomatikh/algorithm/Topo
logies/as3037.txt";

    in = new BufferedReader(new FileReader(file_asTopology));
```

```
        FileWriter fstream = new
FileWriter("/Users/theodor/Documents/PLIROFORIKH/7th_Semester/diplomatikh/algo
rithm/Topologies/out.txt");
        BufferedWriter out = new BufferedWriter(fstream);

        FileWriter fstream2 = new
FileWriter("/Users/theodor/Documents/PLIROFORIKH/7th_Semester/diplomatikh/algo
rithm/Topologies/outBase.txt");
        BufferedWriter outBase = new BufferedWriter(fstream2);

        strLine = in.readLine();
        int max = -1;
        while (strLine != null) {
           strLineArray = strLine.split(" ");
           strNode1 = "r" + strLineArray[0];
           strNode2 = "r" + strLineArray[1];
           if (max < Integer.parseInt(strLineArray[0])) {
              max = Integer.parseInt(strLineArray[0]);
           }
           if (max < Integer.parseInt(strLineArray[1])) {
              max = Integer.parseInt(strLineArray[1]);
           }

           out.write(strNode1 + " " + strNode2 + " " + linkSpeed + "\n");
           out.write(strNode2 + " " + strNode1 + " " + linkSpeed + "\n");

           // r0.out++ --> fiberline --> r1.in++;

           outBase.write("\t\t" + strNode1 + ".out++ --> fiberline --> " + strNode2 +
".in++;\n");
           outBase.write("\t\t" + strNode2 + ".out++ --> fiberline --> " + strNode1 +
".in++;\n");
           strLine = in.readLine();

        }

        Random rand = new Random();

        int[] percentClients = {61, 6, 5, 4, 3, 3, 2, 2, 2, 1, 11};
        max++;
      int sumCl = 0;

        for (int j = 0; j < percentClients.length; j++) {
           double numOfClients = Math.round((float)percentClients[j] / (float)100 *
(float)NUM_OF_CLIENTS);

           sumCl += numOfClients;
```

```
            for (int l = 0; l < numOfClients; l++) {
                String r = groups[j].get(rand.nextInt(groups[j].size()));
                r = "r" + r;
                String s = "c" + max;
                out.write(r + " " + s + " " + linkSpeed + "\n");
                out.write(s + " " + r + " " + linkSpeed + "\n");

                outBase.write("\t\t" + r + ".out++ --> fiberline --> " + s + ".in++;\n");
                outBase.write("\t\t" + s + ".out++ --> fiberline --> " + r + ".in++;\n");
                max++;

            }
        }

        System.out.println("Total NUM of Clients = " + sumCl);


        int[] percentSurrogates = {61, 6, 5, 4, 3, 3, 2, 2, 2, 1, 11};
        int sumSur =0;
        for (int j = 0; j < percentSurrogates.length; j++) {
            int  numOfSurrogates= Math.round((float)percentSurrogates[j] / (float)100 *
(float)NUM_OF_SUROGATES);

            sumSur +=numOfSurrogates;
            System.out.println("Surrog" +numOfSurrogates);
            ///
            for (int l = 0; l < numOfSurrogates; l++) {
                String r = groups[j].get(rand.nextInt(groups[j].size()));
                r = "r" + r;
                String s = "s" + max;
                out.write(r + " " + s + " " + linkSpeed + "\n");
                out.write(s + " " + r + " " + linkSpeed + "\n");

                outBase.write("\t\t" + r + ".out++ --> fiberline --> " + s + ".in++;\n");
                outBase.write("\t\t" + s + ".out++ --> fiberline --> " + r + ".in++;\n");
                max++;

            }
        }
        System.out.println("Total NUM of Surgts = " + sumSur);

        String r = groups[0].get(0);
        r = "r" + r;
        String s = "o" + max;
        out.write(r + " " + s + " " + linkSpeed + "\n");
        out.write(s + " " + r + " " + linkSpeed);

        outBase.write("\t\t" + r + ".out++ --> fiberline --> " + s + ".in++;\n");
        outBase.write("\t\t" + s + ".out++ --> fiberline --> " + r + ".in++;");
```

```
        in.close();
        out.close();
        outBase.close();



    }


  public static double roundTwoDecimals(double d) {
          DecimalFormat twoDForm = new DecimalFormat("#.##");
                  return Double.valueOf(twoDForm.format(d));
}
}
```

Κώδικας για διαχείριση δυναμικών αντικειμένων

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Random;

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 *
 * @author theodor
 */
class RemoveRandomObjects {

    private static final double P = 0.1;
    private static String FILENAME_OBJECTS =
"/Users/theodor/Desktop/CHARTS/util100c20s100o10000req/web_site100_with_possi
bility.txt";
    private static String PATH_SURROGATES =
"/Users/theodor/Desktop/CHARTS/util100c20s100o10000req/";


     public static void removeLineFromFile(String file, String lineToRemove) {

       try {

          File inFile = new File(file);

          if (!inFile.isFile()) {
             System.out.println("Parameter is not an existing file");
             return;
          }

          //Construct the new file that will later be renamed to the original filename.
          File tempFile = new File(inFile.getAbsolutePath() + ".tmp");

          BufferedReader br = new BufferedReader(new FileReader(file));
          PrintWriter pw = new PrintWriter(new FileWriter(tempFile));
```

```
        String line = null;

        //Read from the original file and write to the new
        //unless content matches data to be removed.
        while ((line = br.readLine()) != null) {

           if (!line.trim().equals(lineToRemove)) {

              pw.println(line);
              pw.flush();
           }
        }
        pw.close();
        br.close();

        //Delete the original file
        if (!inFile.delete()) {
           System.out.println("Could not delete file");
           return;
        }

        //Rename the new file to the filename the original file had.
        if (!tempFile.renameTo(inFile)) {
           System.out.println("Could not rename file");
        }

     } catch (FileNotFoundException ex) {
        ex.printStackTrace();
     } catch (IOException ex) {
        ex.printStackTrace();
     }
}



public static void main(String args[]) throws FileNotFoundException, IOException {

   FileReader input = new FileReader(FILENAME_OBJECTS);

   BufferedReader bufRead = new BufferedReader(input);

   int c = 0;

   String str = bufRead.readLine();
   while (str != null) {
      c++;
      str = bufRead.readLine();
   }
```

```
        System.out.println(c);

        bufRead.close();
        input.close();

        input = new FileReader(FILENAME_OBJECTS);
        bufRead = new BufferedReader(input);

        Float[][] objectProbabillity = new Float[c][2];

        str = bufRead.readLine();
        c = 0;
        while (str != null) {

//          System.out.println(str);
            String[] strArray = str.split(" ");

            objectProbabillity[c][0] = Float.parseFloat(strArray[0]);
            objectProbabillity[c][1] = Float.parseFloat(strArray[2]);

            str = bufRead.readLine();
            c++;
        }

        for (int i = 0; i < objectProbabillity.length; i++) {
            for (int j = i; j < objectProbabillity.length; j++) {
                // System.out.print(objectProbabillity[i][j] + " ");
                if (objectProbabillity[i][1] <= objectProbabillity[j][1]) {
                    float temp = objectProbabillity[i][0];
                    objectProbabillity[i][0] = objectProbabillity[j][0];
                    objectProbabillity[j][0] = temp;

                    float temp2 = objectProbabillity[i][1];
                    objectProbabillity[i][1] = objectProbabillity[j][1];
                    objectProbabillity[j][1] = temp2;
                }
            }
        }

//      System.out.println("------");
//      for (int i = 0; i < objectProbabillity.length; i++) {
//          System.out.println(objectProbabillity[i][0] + " " + objectProbabillity[i][1]);
//      }

        ArrayList<Integer>[] Omades = new ArrayList[10];
        for (int i = 0; i < Omades.length; i++) {
            Omades[i] = new ArrayList<Integer>();
        }
```

```
        for (int i = 0; i < objectProbabillity.length; i++) {

            Float obj_prob = objectProbabillity[i][1];

            if (obj_prob <= 1.0 && obj_prob > 0.9) {
                Omades[9].add(i);
            } else if (obj_prob <= 0.9 && obj_prob > 0.8) {
                Omades[8].add(i);
            } else if (obj_prob <= 0.8 && obj_prob > 0.7) {
                Omades[7].add(i);
            } else if (obj_prob <= 0.7 && obj_prob > 0.6) {
                Omades[6].add(i);
            } else if (obj_prob <= 0.6 && obj_prob > 0.5) {
                Omades[5].add(i);
            } else if (obj_prob <= 0.5 && obj_prob > 0.4) {
                Omades[4].add(i);
            } else if (obj_prob <= 0.4 && obj_prob > 0.3) {
                Omades[3].add(i);
            } else if (obj_prob <= 0.3 && obj_prob > 0.2) {
                Omades[2].add(i);
            } else if (obj_prob <= 0.2 && obj_prob > 0.1) {
                Omades[1].add(i);
            } else if (obj_prob <= 0.1 && obj_prob >= 0.0) {
                Omades[0].add(i);
            }
        }

//      for (int i = 0; i < Omades.length; i++) {
//          for (int j = 0; j < Omades[i].size(); j++) {
//              System.out.print(Omades[i].get(j) + " ");
//          }
//          System.out.println();
//      }

        ArrayList<Integer> expiredItems = new ArrayList<Integer>();


        Integer[] numberOfExpired = new Integer[Omades.length];

        float countf = (float) 0.0;
        for (int i = 0; i < Omades.length; i++) {

            numberOfExpired[i] = Math.round((float) (countf * P * Omades[i].size()));
            countf += 0.1;
        }

        Random r = new Random();
```

```
        for (int i = 0; i < Omades.length; i++) {
            int num = numberOfExpired[i];
//            ArrayList<Integer> temp = new ArrayList<Integer>();
            int[] temp = new int[num];

            int k = 0;
            for (k = 0; k < temp.length; k++) {
                temp[k] = -1;
            }

            int first = Omades[i].get(0);
            int last = Omades[i].get(Omades[i].size() - 1);
            while (true) {

                int n = first + r.nextInt(last - first + 1);

                boolean f = false;
                for (k = 0; k < temp.length; k++) {
                    if (temp[k] == -1) {
                        break;
                    }
                    if (temp[k] == n) {
                        f = true;
                        break;
                    }
                }
                if (f) {
                    continue;
                }
                if (k == temp.length) {
                    break;
                }
                temp[k] = n;

            }
            for (k = 0; k < temp.length; k++) {
                int g = Math.round(objectProbabillity[temp[k]][0]);
                expiredItems.add(g);
            }

        }

        for (int k = 0; k < expiredItems.size(); k++) {
            System.out.print(expiredItems.get(k) + " ");
        }


        FileReader input2;
```

```
        BufferedReader bufRead2 = new BufferedReader(input);


        File directory = new File(PATH_SURROGATES);
        File files[] = directory.listFiles();
        for (File f : files) {
           if (!(f.getName().matches("^s[0-9]+"))) {
              continue;
           }

           input2 = new FileReader(f);
           bufRead2 = new BufferedReader(input2);
           String str2 = bufRead2.readLine();
           while (str2 != null) {
              System.out.println(str2);

              String[] s = str2.split(" ");
              String s0 = s[0];

              for (int i=0; i<expiredItems.size();i++){
                 if (Integer.parseInt(s0) == expiredItems.get(i)){
                removeLineFromFile(f.getAbsolutePath(), str2);
                 }
              }

              str2 = bufRead2.readLine();
           }
           System.out.println(f.getName() + "-------");
        }
     }
  }
```