ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΞΙΣΟΡΡΟΠΗΣΗ ΦΟΡΤΙΟΥ ΔΙΑΔΙΚΤΥΑΚΩΝ ΥΠΗΡΕΣΙΩΝ ΣΤΑ ΕΥΦΥΗ ΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΑ ΔΙΚΤΥΑ

Διπλωματική Εργασία

του

Καλαφατίδη Σαράντη

Θεσσαλονίκη, Φεβρουάριος 2019

ΕΞΙΣΟΡΡΟΠΗΣΗ ΦΟΡΤΙΟΥ ΔΙΑΔΙΚΤΥΑΚΩΝ ΥΠΗΡΕΣΙΩΝ ΣΤΑ ΕΥΦΥΗ ΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΑ ΔΙΚΤΥΑ

Καλαφατίδης Σαράντης

Μηχανικός Γεωπληροφορικής και Τοπογραφίας, Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κεντρικής Μακεδονίας, 2015

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Μαμάτας Ελευθέριος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 26/02/2019

Μαμάτας Ελευθέριος (Επίκουρος Καθηγητής)	Σακελλαρίου Ηλίας (Επίκουρος Καθηγητής)	Παπαδημητρίου Παναγιώτης (Επίκουρος Καθηγητής)
.....	.....	.....

Καλαφατίδης Σαράντης

.....

Περίληψη

Η εξισορρόπηση φορτίου στα δίκτυα υπολογιστών είναι μια τεχνική που χρησιμοποιείται για το διαμοιρασμό της κυκλοφορίας δεδομένων ή του φορτίου εργασίας σε πολλαπλούς δικτυακούς συνδέσμους (links) ή διακομιστές. Στα ευφυή προγραμματιζόμενα δίκτυα (ΕΠΔ), με την αποσύνδεση του επιπέδου ελέγχου από το επίπεδο δεδομένων επιτυγχάνεται μεγαλύτερος έλεγχος ενός δικτύου μέσω του προγραμματισμού, έτσι δύσκολα προβλήματα που αφορούν την βελτιστοποίηση της απόδοσης των δικτύων (όπως η εξισορρόπηση φορτίου) γίνονται εύκολα διαχειρίσιμα, με σωστά σχεδιασμένους κεντροποιημένους (centralized) μηχανισμούς.

Η διπλωματική εργασία στοχεύει στη βελτίωση του τρόπου διαμοιρασμού της κυκλοφορίας δεδομένων με τη χρήση τεχνικών εξισορρόπησης φορτίου, αξιοποιώντας την τεχνολογία των ευφυών προγραμματιζόμενων δικτύων. Το υπόβαθρο της εργασίας περιλαμβάνει τη θεωρητική μελέτη των ευφυών προγραμματιζόμενων δικτύων και των βασικότερων μεθόδων εξισορρόπησης φορτίου. Στη συνέχεια, προτείνουμε μια δική μας μέθοδο εξισορρόπησης φορτίου, η οποία βασίζεται στη δικτυακή κίνηση αλλά και στους διαθέσιμους υπολογιστικούς πόρους των διακομιστών, λαμβάνοντας υπόψη το είδος των εφαρμογών που εκτελούνται.

Στην πειραματική προσέγγιση της εργασίας προσομοιώσαμε περιβάλλον ΕΠΔ με τη βοήθεια του εξομοιωτή Mininet και το δικτυακό ελεγκτή Floodlight, με σκοπό τη μελέτη και την αξιολόγηση μηχανισμών εξισορρόπησης φορτίου. Τέλος, πραγματοποιήσαμε εκτεταμένη συγκριτική αξιολόγηση της απόδοσης δύο αλγορίθμων (του αλγόριθμου κυκλικής επιλογής και του αλγόριθμου που βασίζεται σε στατιστικά της δικτυακής κίνησης) με τη δική μας πρόταση, καταλήγοντας σε επιπρόσθετες προτάσεις βελτίωσης.

Λέξεις Κλειδιά: Ευφυή προγραμματιζόμενα δίκτυα, εξισορρόπηση φορτίου,Mininet, Floodlight

Abstract

Load balancing on computer networks is a technique used to share data traffic or task load on multiple network links or servers. In Software Defined Networks (SDNs), by separating the control plane from the data plane, more control over a network is achieved through programming, so difficult network optimization problems (such as load balancing) can be easily managed, by using properly designed centralized mechanisms.

This thesis aims to optimize the way of sharing data traffic using load balancing techniques, utilizing SDN technology. The background of the thesis includes the theoretical study of SDN and the basic methods of load balancing. We present our proposal, a load balancing scheme that is based on network traffic and available server computing resources, taking into account the type of applications that are running.

In the experimental approach we simulate an SDN environment using the Mininet simulator and the Floodlight SDN controller in order to study and evaluate load balancing algorithms. Finally, we performed an extensive comparative evaluation of the performance of two algorithms (simple round robin and an algorithm based on network traffic statistics) with our proposal, resulting in additional improvement suggestions.

Keywords: Software Defined Networks (SDN), Load Balancing, Mininet, Floodlight

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κατάλογος Γραφημάτων

Συμβολισμοί

Συμβολισμοί	Αγγλική ορολογία	Ελληνική ορολογία
SDN	Software Defined Networking	Ευφυή προγραμματιζόμενα δίκτυα
REST	Representational State Transfer	Μεταφορά κατάστασης
SSH	Secure Shell	Ασφαλές δικτυακό πρωτόκολλο
HTTP	Hypertext Transfer Protocol	Πρωτόκολλο μεταφοράς υπερκειμένου
QoS	Quality of service	Ποιότητα των υπηρεσιών
TCP	Transmission Control Protocol	Πρωτόκολλο Ελέγχου Μετάδοσης
API	Application Programming Interface	Διεπαφή προγραμματισμού εφαρμογών
WAN	Wide Area Network	Δίκτυο ευρείας περιοχής
IP	Internet Protocol	Διαδικτυακό πρωτόκολλο
JSON	JavaScript Object Notation	Μορφότυπο αντικειμένου JavaScript

1 Εισαγωγή

1.1 Πρόβλημα - Σημαντικότητα του θέματος

Τα τελευταία χρόνια όλο και περισσότερες εφαρμογές μεταφέρονται σε περιβάλλον υπολογιστικού νέφους (cloud computing) και συχνά απαιτείται η μεταφορά μεγάλου όγκου δεδομένων μέσω πολλαπλών δικτυακών διαδρομών. Επίσης, η χρήση των συστοιχιών διακομιστών (server clusters) έχει γίνει ένα βασικό μοντέλο του υπολογιστικού νέφους. Δηλαδή, στα κέντρα δεδομένων συγκεντρώνονται πολλοί διακομιστές που παρέχουν τις ίδιες υπηρεσίες, οι οποίες μπορούν να αυξήσουν σημαντικά τη διαθέσιμη υπολογιστική ισχύ καθώς και να μειώσουν τα σημεία αποτυχίας, παρέχοντας έτσι μεγαλύτερη διαθεσιμότητα και ανοχή στα σφάλματα.

Η αποδοτική ρύθμιση της κυκλοφορίας δεδομένων (data traffic) ανάμεσα στους κόμβους των δικτύων είναι πολύ σημαντική για όλα τα δικτυακά περιβάλλοντα και κυρίως στα δίκτυα των κέντρων δεδομένων. Η χρήση μηχανών με ισχυρές δυνατότητες δικτύωσης και επεξεργασίας συχνά δεν αρκεί για τη μέγιστη απόδοση των εφαρμογών. Πολλές φορές, η κίνηση δεν είναι σταθερή και παρουσιάζει σημαντικές διακυμάνσεις στις παρεχόμενες υπηρεσίες και τους διαθέσιμους υπολογιστικούς πόρους, με αποτέλεσμα να υπάρξει η πιθανότητα συμφόρησης στο δίκτυο ή υπερφόρτωσης των διακομιστών. Ως εκ τούτου, είναι σημαντικό να εξισορροπείται ομοιόμορφα το κυκλοφοριακό φορτίο (traffic load) ανάμεσα σε πολλαπλές διαδρομές.

Η εξισορρόπηση φορτίου στα δίκτυα υπολογιστών είναι μια τεχνική που χρησιμοποιείται για το διαμοιρασμό του φορτίου εργασίας (task load) μέσω πολλαπλών δικτυακών συνδέσμων (links) σε εναλλακτικούς διακομιστές [39]. Για να εξυπηρετήσουν περισσότερους πελάτες με ελάχιστη καθυστέρηση και μέγιστη απόδοση, οι μηχανισμοί εξισορρόπησης φορτίου διανέμουν το εισερχόμενο φορτίο σε μια σειρά αντιγράφων διακομιστών (server replicas).

Συχνά στα παραδοσιακά δίκτυα, η εξισορρόπηση φορτίου χρησιμοποιεί ειδικές συσκευές (hardware) για το διαμοιρασμό της δικτυακής κυκλοφορίας. Παρόλο που αυτές οι συσκευές είναι γρήγορες, είναι ακριβές και έχουν έλλειψη ευελιξίας στη διαμόρφωση τους, δηλαδή η ρύθμιση τους δεν μπορεί να γίνει

δυναμικά, βάσει της παρακολούθησης της κατάσταση του δικτύου σε πραγματικό χρόνο.

Επίσης λόγω της ιδιαίτερης τοπολογίας και των χαρακτηριστικών της δικτυακής κίνησης στα κέντρα δεδομένων, οι στατικοί αλγόριθμοι εξισορρόπησης φορτίου (π.χ. ο αλγόριθμος κυκλικής επιλογής) δεν είναι κατάλληλοι, επειδή δεν λαμβάνουν υπόψη τους πληροφορίες που έχουν να κάνουν με τη δικτυακή συμφόρηση.

Μια πολύ ενδιαφέρουσα εναλλακτική λύση, είναι η χρήση δυναμικών αλγορίθμων εξισορρόπησης φορτίου μέσω των ευφυών προγραμματιζόμενων δικτύων (ΕΠΔ) - Software-Defined Networks (SDNs) [26], στα οποία οι διαχειριστές των δικτύων έχουν τη δυνατότητα να υλοποιήσουν αλγορίθμους συμφώνα με τις ανάγκες των διαδικτυακών υπηρεσιών και τις δυνατότητες των διαδικτυακών και επεξεργαστικών τους πόρων. Τα ΕΠΔ προσφέρουν την απαραίτητη ευελιξία στο δίκτυο να προσαρμοστεί σε δυναμικές συνθήκες, χρησιμοποιώντας μια παραδοσιακή γλώσσα προγραμματισμού (π.χ. Javaή Python). Οι αντίστοιχοι αλγόριθμοι μπορούν να υλοποιούν στρατηγικές εξισορρόπησης φορτίου, λαμβάνοντας υπόψη τους τη μεγάλη εικόνα για το δίκτυο, δηλαδή την τοπολογία συνολικά.

1.2 Σκοπός - Στόχοι

Σκοπός της εργασίας είναι η μελέτη της βελτίωσης του τρόπου διαμοιρασμού της δικτυακής κυκλοφορίας με τη χρήση τεχνικών εξισορρόπησης φορτίου, αξιοποιώντας την τεχνολογία των ευφυών προγραμματιζόμενων δικτύων. Στόχος αυτής της παρέμβασης είναι η μείωση του φορτίου εργασίας των διακομιστών σε ένα δικτυακό περιβάλλον για την αποφυγή συμφόρησης και τη βελτίωση της απόδοσης των εφαρμογών που παρέχονται.

Το υπόβαθρο της εργασίας περιλαμβάνει αρχικά τη θεωρητική μελέτη των ευφυών προγραμματιζόμενων δικτύων και των βασικότερων μεθόδων εξισορρόπησης φορτίου. Εντοπίσαμε ότι οι σχετικές προτάσεις δεν έχουν τη δυνατότητα δυναμικής προσαρμογής της εξισορρόπησης φορτίου στο είδος της εφαρμογής. Στη συνέχεια προτείνουμε μια δική μας μέθοδο εξισορρόπησης φορτίου, που βασίζεται στη δικτυακή κίνηση αλλά και στους διαθέσιμους υπολογιστικούς πόρους των διακομιστών, λαμβάνοντας υπόψη της και το είδος των εφαρμογών που εκτελούνται.

Στη συνέχεια, παρουσιάζουμε την πειραματική μας μεθοδολογία. Πιο συγκεκριμένα, αρχικά δημιουργούμε ένα εικονικό δικτυακό περιβάλλον. Έπειτα

εφαρμόζουμε και αξιολογούμε ένα στατικό και έναν δυναμικό (που βασίζεται στη δικτυακή κίνηση), αλγόριθμο εξισορρόπησης φορτίου σε ένα ΕΠΔ. Τέλος, πραγματοποιούμε συγκριτική αξιολόγηση της απόδοσης των δύο αλγορίθμων (του αλγόριθμου κυκλικής επιλογής και του αλγόριθμου που βασίζεται σε στατιστικά της δικτυακής κίνησης) με τη δική μας πρόταση και καταγράφουμε περαιτέρω λύσεις του προβλήματος.

1.3 Ερωτήματα Υποθέσεις

Η παρούσα μελέτη καλείται να απαντήσει στα ακόλουθα ερευνητικά ερωτήματα:

1. Ποιες είναι οι βασικές τεχνικές εξισορρόπησης φορτίου στα ευφυή προγραμματιζόμενα δίκτυα;
2. Ποια πλεονεκτήματα προσφέρει η αξιοποίηση της τεχνολογίας ευφυών προγραμματιζόμενων δικτύων στις τεχνικές εξισορρόπησης φορτίου;
3. Επαρκεί η παρακολούθηση της δικτυακής κίνησης ανάμεσα στους διακομιστές για μια αποδοτική τεχνική εξισορρόπησης φορτίου;
4. Πρέπει στις τεχνικές εξισορρόπησης φορτίου να λαμβάνεται υπόψη η κατανάλωση των επεξεργαστικών πόρων των διακομιστών;
5. Πρέπει να λαμβάνεται υπόψη το είδος της Διαδικτυακής εφαρμογής που παρέχεται από τους διακομιστές σε μια απόφαση εξισορρόπησης φορτίου;
6. Ποια χαρακτηριστικά των εφαρμογών επηρεάζουν την απόδοση των διακομιστών; Πως επηρεάζει ο αλγόριθμος εξισορρόπησης φορτίου την απόδοση των Διαδικτυακών υπηρεσιών;
7. Θα ωφελούσε η χρήση ιδιαίτερων αλγορίθμων εξισορρόπησης φορτίου ανά τύπο εφαρμογής;
8. Σε ποιες συνθήκες η μία τεχνική εξισορρόπησης φορτίου είναι καλύτερη από κάποια άλλη;

1.4 Συνεισφορά

Η συνεισφορά της εργασίας αφορά στις παρακάτω πλευρές:

1) Το βασικό θεωρητικό πλαίσιο που αναφέρεται στην αρχιτεκτονική των ευφυών προγραμματιζόμενων δικτύων και η ανάδειξη των πλεονεκτημάτων τους σε σύγκριση τους με τα παραδοσιακά δίκτυα, καθώς και η παρουσίαση των βασικών δομικών στοιχείων τους.

2) Την παρουσίαση του προβλήματος που καλούνται να λύσουν οι βασικοί μηχανισμοί εξισορρόπησης φορτίου, όπως και τα κυρίαρχα χαρακτηριστικά τους. Ιδιαίτερα σημαντική είναι η παρουσίαση βασικών αδυναμιών τους.
3) Την πρόταση ενός νέου μηχανισμού εξισορρόπησης φορτίου, ο οποίος έχει σαν στοιχείο πρωτοτυπίας ότι διαμοιράζει το φορτίο λαμβάνοντας υπόψη του το είδος της εφαρμογής.
4) Την εισαγωγή του αναγνώστη στην πειραματική διαδικασία που ακολουθήθηκε και την εξοικείωση του με τα εργαλεία λογισμικού που χρησιμοποιήθηκαν.
5) Την υλοποίηση πειραματικών σεναρίων με σκοπό την αξιολόγηση της απόδοσης του νέου αλγορίθμου, σε σχέση με δύο αντιπροσωπευτικές προτάσεις της βιβλιογραφίας: του αλγόριθμου κυκλικής επιλογής και του αλγόριθμου που βασίζεται σε στατιστικά της δικτυακής κίνησης.

Η εργασία καταλήγει σε συμπεράσματα που μπορούν να αποτελέσουν βάση για περεταίρω έρευνα στις τεχνολογίες εξισορρόπησης φορτίου.

1.5 Διάρθρωση της μελέτης

Στην ενότητα που ακολουθεί παρουσιάζουμε τη σημερινή κατάσταση των ευφυών προγραμματιζόμενων δικτύων και τις βασικότερες μεθόδους εξισορρόπησης φορτίου. Η ενότητα καταλήγει στη βιβλιογραφική επισκόπηση της εργασίας και την πρόταση ενός νέου μηχανισμού εξισορρόπησης φορτίου από εμάς, ο οποίος βασίζεται στη δικτυακή κίνηση και τους διαθέσιμους υπολογιστικούς πόρους των διακομιστών, λαμβάνοντας υπόψη του το είδος των εφαρμογών που εκτελούνται. Στην τρίτη ενότητα περιγράφουμε την πειραματική προσέγγιση της εργασίας, πιο συγκεκριμένα: αναφέρουμε τους στόχους της, τις διάφορες τεχνικές επιλογές που υιοθετήσαμε, τα εργαλεία που χρησιμοποιήσαμε καθώς επίσης αναλύουμε και το περιβάλλον δοκιμών μας. Στην τέταρτη ενότητα παρουσιάζουμε τα αποτελέσματα των πειραματικών σεναρίων μας, με βασικούς στόχους: (α) την επικύρωση των τεχνικών επιλογών της διατριβής, και (β) την αξιολόγηση της απόδοσης του νέου αλγορίθμου, σε αντιπαραβολή με δύο αντιπροσωπευτικούς αλγορίθμους της βιβλιογραφίας. Τέλος, καταγράφουμε τα συμπεράσματα και τους περιορισμούς της εργασίας.

2 Θεωρητικό Υπόβαθρο-Ανάλυση Τεχνολογιών

2.1 Τα Ευφυή Προγραμματιζόμενα Δίκτυα (ΕΠΔ)

Τα πρωτόκολλα δρομολόγησης των παραδοσιακών δικτύων, τα οποία εκτελούνται μέσα στους δικτυακούς κόμβους (δρομολογητές, μεταγωγείς), είναι οι βασικές τεχνολογίες που επιτρέπουν τη διακίνηση πληροφοριών με τη μορφή ψηφιακών πακέτων. Παρά την ευρεία υιοθέτησή τους, τα παραδοσιακά δίκτυα IP είναι πολύπλοκα στη διαχείριση και την επεκτασιμότητα τους [3]. Για να εφαρμόσουν δικτυακές πολιτικές σε υψηλό επίπεδο, οι διαχειριστές των παραδοσιακών δικτύων πρέπει να ρυθμίσουν κάθε μεμονωμένη δικτυακή συσκευή ξεχωριστά, χρησιμοποιώντας εντολές χαμηλού επιπέδου οι οποίες συχνά είναι εξειδικευμένες και περιορισμένες από τον κατασκευαστή των δικτυακών συσκευών. Επιπρόσθετα, ένα δικτυακό περιβάλλον πρέπει να είναι ανθεκτικό στα σφάλματα και να προσαρμόζεται στις αλλαγές του δικτυακού φορτίου. Τέτοιοι προχωρημένοι μηχανισμοί προσαρμογής είναι σχεδόν ανύπαρκτοι στα παραδοσιακά δίκτυα IP. Η ρύθμιση και οι δυναμικές αλλαγές σε τέτοια δικτυακά περιβάλλοντα είναι συνεπώς εξαιρετικά δύσκολες. είναι μια αναδυόμενη τεχνολογία που έρχεται να μειώσει τους παραπάνω περιορισμούς των υφιστάμενων δικτυακών υποδομών.

Τα ευφυή προγραμματιζόμενα δίκτυα (ΕΠΔ) αποτελούν μια αρχιτεκτονική που στοχεύει σε δίκτυα εύκαμπτα και ευέλικτα, καθώς προχωρά στον φυσικό διαχωρισμό του επιπέδου ελέγχου (control plane) από το επίπεδο δεδομένων (data plane). Η διαχείριση των αποφάσεων για τη δρομολόγηση λαμβάνεται από ένα κεντρικό σημείο διαχείρισης, τον ελεγκτή (controller). Ο σχεδιασμός και ο τρόπος λειτουργίας των δικτύων αναπτύσσεται με διαφορετικό τρόπο από τα παραδοσιακά δίκτυα: η τεχνολογία ΕΠΔ επιδιώκει να αντιμετωπίσει τη στατική αρχιτεκτονική των παραδοσιακών δικτύων με έναν αυξημένο επίπεδο προσαρμογής που βασίζεται σε κεντροποιημένο έλεγχο, υλοποιώντας μηχανισμούς δικτυακών ελεγκτών (network controllers) με παραδοσιακές γλώσσες προγραμματισμού.

2.1.1 Ιστορική εξέλιξη της τεχνολογίας ΕΠΔ

Η αρχή των ΕΠΔ μπορεί να εντοπισθεί στην πρώτη υλοποίηση του διαχωρισμού των επιπέδων ελέγχου και δεδομένων στο δημόσιο τηλεφωνικό δίκτυο. Ο φορέας Διαδικτυακών προτύπων Internet Engineering Task Force (IETF) [41] άρχισε να εξετάζει τρόπους αποσύνδεσης των λειτουργιών ελέγχου από αυτές της

προώθησης, σε ένα προτεινόμενο πρότυπο διασύνδεσης (interface) που δημοσιεύτηκε το 2004 με την ονομασία Forwarding and Control Element Separation (ForCES) [40].

Αυτές οι πρώτες προσπάθειες απέτυχαν να κερδίσουν την ευρεία αποδοχή για δύο λόγους. Ο ένας είναι ότι πολλοί στην κοινότητα του Διαδικτύου θεώρησαν ότι ο διαχωρισμός του επιπέδου ελέγχου από την προώθηση δεδομένων είναι επικίνδυνος, ιδίως λόγω της πιθανότητας μιας αποτυχίας στον έλεγχο του πρωτοκόλλου. Το δεύτερο είναι ότι οι κατασκευαστές συσκευών ανησυχούσαν ότι η δημιουργία πρότυπων διεπαφών προγραμματισμού εφαρμογών (API) μεταξύ των επιπέδων ελέγχου και δεδομένων θα δημιουργούσε ζητήματα ανταγωνισμού, με αποτέλεσμα χαμηλότερο μερίδιο στην αντίστοιχη αγορά.

Η χρήση του λογισμικού ανοιχτού κώδικα σε αρχιτεκτονικές που διαχωρίζουν τον έλεγχο με τα δεδομένα έχει τις ρίζες του στο έργο Ethane [38] του τμήματος πληροφορικής του Stanford. Ο σχεδιασμός ενός μεταγωγέα (switch) Ethane οδήγησε στη δημιουργία του OpenFlow [24], του βασικότερου πρωτοκόλλου για τα ΕΠΔ. Ο Nick McKeown τον Απρίλιο του 2008 πρότεινε την πρώτη διεπαφή στο OpenFlow. Με την ίδρυση του Open Network Foundation (ONF) [24] το Μάρτιο του 2011 το OpenFlow παρουσίασε τεράστια ανάπτυξη. Ο φορέας ONF είναι ο σημαντικότερος μη κερδοσκοπικός οργανισμός για τα ΕΠΔ.

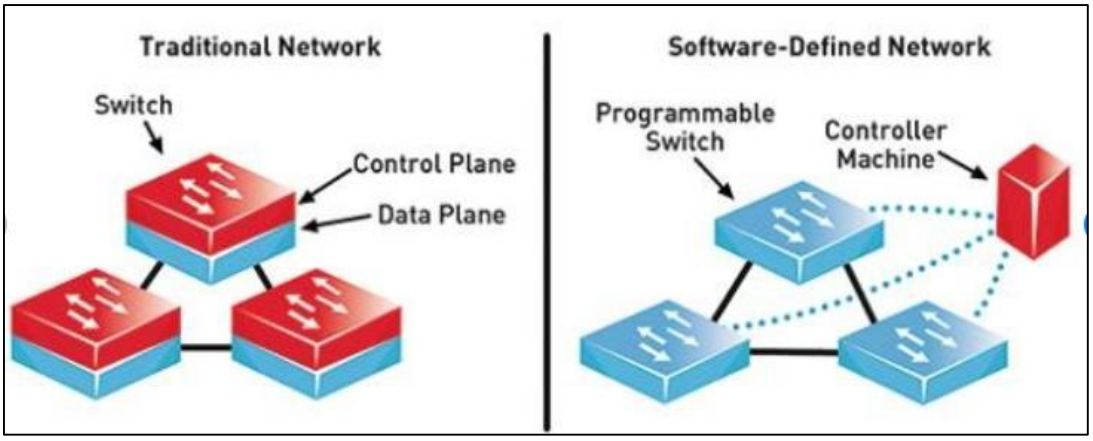
2.1.2 Από τα παραδοσιακά δίκτυα στα ΕΠΔ

Τα παραδοσιακά δίκτυα αποτελούνται συνήθως από τις συσκευές τελικού χρήστη ή κεντρικούς υπολογιστές διασυνδεδεμένους από την δικτυακή υποδομή. Αυτή η υποδομή χρησιμοποιεί συσκευές προώθησης όπως δρομολογητές και μεταγωγείς καθώς και κανάλια επικοινωνίας για τη μεταφορά δεδομένων μεταξύ αυτών. Οι δρομολογητές και οι μεταγωγείς είναι συνήθως «κλειστά» συστήματα δηλαδή ο διαχειριστής δεν μπορεί να παρέμβει στην λειτουργία της συσκευής, και οι δυνατότητες τους είναι συχνά περιορισμένες από τον κατασκευαστή. Λόγο αυτού του περιορισμού τα δικτυακά περιβάλλοντα των παραδοσιακών δικτύων παρουσιάζουν δυσκολίες στην επεκτασιμότητα. Για παράδειγμα αν οι δικτυακές συσκευές ενός δικτύου δεν υποστηρίζουν ένα πρωτόκολλο όπως το IPv6, τότε είναι πολύ δύσκολο και σχεδόν ακατόρθωτο να το εφαρμόσουν.

Στα παραδοσιακά δίκτυα το επίπεδο ελέγχου και δεδομένων δε διαχωρίζονται. Κάθε κόμβος είναι υπεύθυνος και για τις δύο λειτουργίες. Το επίπεδο ελέγχου είναι υπεύθυνο για την επιλογή των διαδρομών και το επίπεδο δεδομένων για

την προώθηση των πακέτων (δεδομένων). Σε περίπτωση αλλαγών στην διάταξη του δικτύου, για την ορθή λειτουργία του πρέπει να ενημερωθούν όλες οι υπάρχουσες δικτυακές συσκευές. Στα παραδοσιακά δίκτυα όμως κάθε κόμβος του δικτύου αδυνατεί να συλλέξει πληροφορίες για όλο το δίκτυο και η γνώση τους συνήθως περιορίζεται στις πληροφορίες που συλλέγει από και προς τα κανάλια των γειτονικών τους κόμβων. Συνεπώς οι χειριστές των παραδοσιακών δικτύων πρέπει να ρυθμίσουν κάθε μεμονωμένη συσκευή του δικτύου ξεχωριστά. Η ρύθμιση και οι αλλαγές με δυναμικό τρόπο σε ένα τέτοιο δικτυακό περιβάλλον είναι εξαιρετικά δύσκολες.

Από την άλλη, τα ΕΠΔ εκφράζουν ένα δικτυακό μοντέλο που βασίζεται στην ιδέα της μετάβασης από το παραδοσιακό πλήρως κατανεμημένο μοντέλο σε μια περισσότερο κεντροποιημένη προσέγγιση (Εικόνα 2.1). Στα ΕΠΔ οι συσκευές προώθησης έχουν περιορισμένες δυνατότητες λήψης αποφάσεων και εφαρμόζουν αποφάσεις οι οποίες έχουν ληφθεί από τους ελεγκτές.



Εικόνα 2.1 - Παραδοσιακά δίκτυα και ΕΠΔ [42]

Στα ΕΠΔ, οι μεταγωγείς είναι αποσυνδεμένοι από το επίπεδο ελέγχου και εξυπηρετούν μόνο το επίπεδο δεδομένων, ενώ οι ελεγκτές (controllers) είναι υπεύθυνοι για το χειρισμό τους. Οι αποφάσεις ελέγχου, στην περίπτωση αυτή, γίνονται έχοντας μια συνολική άποψη της κατάστασης του δικτύου. Στα SDN, το επίπεδο ελέγχου λειτουργεί ως μοναδικό, κεντροποιημένο λειτουργικό σύστημα δικτύου. Δεδομένου του ότι ο ελεγκτής γνωρίζει ολόκληρη την τοπολογία του δικτύου, μπορεί εύκολα να προσαρμοστεί στις απαιτήσεις όσον αφορά στην επεκτασιμότητα και στην ευελιξία.

2.1.3 Πλεονεκτήματα των ΕΠΔ

Με την αποσύνδεση του επιπέδου ελέγχου από το επίπεδο δεδομένων, επιτυγχάνεται μεγαλύτερος έλεγχος ενός δικτύου μέσω του προγραμματισμού. Αυτό το χαρακτηριστικό φέρνει πλεονεκτήματα στην βελτίωση της ρύθμισης και την απόδοσης και ενθαρρύνει την καινοτομία στην αρχιτεκτονική και την λειτουργία των δικτύων. Επιπλέον, με την ικανότητα να μπορούν να αποκτήσουν στιγμιαία πληροφορίες για την κατάσταση του δικτύου, τα ΕΠΔ επιτρέπουν τον κεντρικό έλεγχο ενός δικτυού σε πραγματικό χρόνο ο οποίος βασίζεται τόσο στην στιγμιαία κατάσταση δικτύου όσο και σε πολιτικές προκαθορισμένες από τον διαχειριστή. Αυτό περαιτέρω οδηγεί σε οφέλη στη βελτιστοποίηση της διαμόρφωσης και της απόδοσης του δικτύου. Το δυνητικό όφελος της τεχνολογίας ΕΠΔ αποδεικνύεται στη συνέχεια, από το γεγονός ότι προσφέρει μια βολική πλατφόρμα για πειραματισμούς σε νέες τεχνικές και ενθαρρύνει νέες αρχιτεκτονικές, που αποδίδονται στο προγραμματισμό του δικτύου. Μερικά από τα οφέλη των ΕΠΔ δικτύων δίνονται παρακάτω:

Κεντρικοποιημένη αυτόματοποιημένη διαχείριση: Όταν προστίθεται νέος εξοπλισμός σε ένα υπάρχον δίκτυο, απαιτούνται σωστές ρυθμίσεις για την επίτευξη συνεκτικής λειτουργίας του δικτύου στο σύνολό του. Ωστόσο, λόγω της ετερογένειας μεταξύ των δικτυακών συσκευών και των διεπαφών, η τρέχουσα ρύθμιση του δικτύου συνήθως περιλαμβάνει ένα ορισμένο επίπεδο χειρωνακτικής επεξεργασίας. Αυτή η διαδικασία χειροκίνητης ρύθμισης είναι κουραστική και επιρρεπής σε σφάλματα. Ταυτόχρονα, σημαντική προσπάθεια επίσης απαιτείται από τους διαχειριστές για την αντιμετώπιση προβλημάτων του δικτύου που προκύπτουν από εσφαλμένη ρύθμιση του. Είναι γενικά αποδεκτό ότι, με τον τρέχοντα σχεδιασμό των παραδοσιακών δικτύων, η δυναμική αναδιάταξη ενός δικτύου παραμένει μια μεγάλη πρόκληση. Τα SDN βοηθάνε στην αντιμετώπιση μιας τέτοιας κατάστασης, γιατί καθιστούν δυνατή τη ρύθμιση των δικτυακών συσκευών από ένα μόνο σημείο, αυτόματα, μέσω λογισμικού. Έτσι, ένα ολόκληρο δίκτυο μπορεί να ρυθμιστεί προγραμματιστικά και δυναμικά βάση της κατάστασης του.

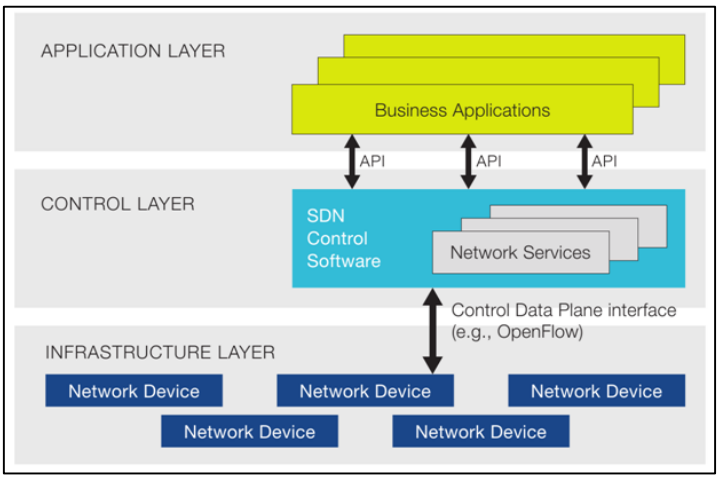
Βελτίωση της απόδοσης: Ένας από τους βασικούς στόχους της ορθής λειτουργίας ενός δικτύου είναι να μεγιστοποιηθεί η αξιοποίηση της υποδομής του. Ωστόσο, λόγω της συνύπαρξης διαφόρων τεχνολογιών σε ένα ενιαίο δίκτυο, βελτιστοποίηση της απόδοσης του δικτύου στο σύνολό του είναι αρκετά δύσκολη. Οι τρέχουσες

προσεγγίσεις συχνά επικεντρώνονται στη βελτιστοποίηση της απόδοσης ενός υποσυνόλου δικτύων ή στην ποιότητα της εμπειρίας των χρηστών για ορισμένες υπηρεσίες που παρέχονται σε αυτό. Προφανώς, αυτές οι προσεγγίσεις θα μπορούσαν να οδηγήσουν σε υποβέλτιστες επιδόσεις,. Τα ΕΠΔ προσφέρουν μια ευκαιρία για τη βελτίωση της απόδοσης του δικτύου στο σύνολο του. Συγκεκριμένα, στα ΕΠΔ επιτρέπουν τον κεντρικό έλεγχο, έχοντας εικόνα ολόκληρου του δικτύου και έναν έλεγχο ανάδρασης με πληροφορίες που ανταλλάσσονται μεταξύ διαφορετικών επιπέδων στην αρχιτεκτονική του δικτύου. Ως εκ τούτου, πολλά δύσκολα προβλήματα της βελτιστοποίησης της απόδοσης γίνονται διαχειρίσιμα, με σωστά σχεδιασμένους κεντρικούς αλγόριθμους. Το να ακολουθούν νέες λύσεις για κλασικά προβλήματα, όπως ο έλεγχος συμφόρησης από άκρο σε άκρο, εξισορρόπηση φορτίου , ενεργειακά αποδοτική λειτουργία και η υποστήριξη της ποιότητας των υπηρεσιών μπορούν να αναπτυχθούν πιο εύκολα για να συμβάλουν στην μεγιστοποίηση της απόδοσης των δικτύων.

3) **Ενθάρρυνση της καινοτομίας**: Τα ΕΠΔ ενθαρρύνουν την καινοτομία παρέχοντας μία προγραμματιζόμενη πλατφόρμα δικτύου για υλοποίηση, πειραματισμό και ανάπτυξη νέων ιδεών.

2.1.4 Αρχιτεκτονική των ΕΠΔ-Μοντέλο αναφοράς

Το μοντέλο αναφοράς όπως προτείνεται από τον Open Networking foundation (ONF) απεικονίζεται στην εικόνα 2.2. Το μοντέλο αυτό αποτελείται από τρία στρώματα το στρώμα υποδομής (Infrastructure layer), το στρώμα ελέγχου (Control layer) και το στρώμα εφαρμογής (Application layer),



Εικόνα 2.2 - Αρχιτεκτονική SDN

Το επίπεδο υποδομής αποτελείται από συσκευές προώθησης (π.χ. μεταγωγείς, δρομολογητές κ.λπ.) που ανήκουν στο επίπεδο δεδομένων. Οι λειτουργίες αυτών των συσκευών προώθησης είναι ως επί το πλείστον δύο. Πρώτον, είναι υπεύθυνες για τη συλλογή πληροφοριών για την κατάσταση του δικτύου (τοπολογία δικτύου, μετρητές στατιστικών στοιχείων της δικτυακής κυκλοφορίας κ.α..) και την αποστολή τους στους ελεγκτές. Δεύτερον, είναι υπεύθυνες για την επεξεργασία και την προώθηση των πακέτων βάσει των κανόνων που παρέχονται από τον ελεγκτή.

Το επίπεδο ελέγχου γεφυρώνει το επίπεδο εφαρμογής και το επίπεδο υποδομής μέσω των δύο διεπαφών του (νότια και βόρεια διεπαφή).

Το επίπεδο εφαρμογής περιέχει εφαρμογές ΕΠΔ σχεδιασμένες να πληρούν τις απαιτήσεις των χρηστών του δικτύου. Μέσα από την προγραμματιζόμενη πλατφόρμα που παρέχεται από το επίπεδο ελέγχου, οι εφαρμογές ΕΠΔ είναι σε θέση να έχουν πρόσβαση και τον έλεγχο των συσκευών μεταγωγής στο επίπεδο υποδομής. Παράδειγμα εφαρμογών ΕΠΔ μπορεί να είναι ο δυναμικός έλεγχος πρόσβασης στο δίκτυο (Access contol), η εξισορρόπηση φορτίου των διακομιστών και η εικονοποίηση του δικτύου (virtual network).

2.1.5 Επίπεδο υποδομής

Στο επίπεδο υποδομής ανήκουν οι δικτυακές συσκευές ΕΠΔ. Γενικά, μια συσκευή δικτύου είναι μια οντότητα που λαμβάνει πακέτα στις θύρες της και εκτελεί μία ή περισσότερες δικτυακές λειτουργίες. Παραδείγματα δικτυακών λειτουργιών είναι: προώθηση ενός ληφθέντος πακέτου, η απόρριψή του, η αλλαγή της κεφαλίδας του πακέτου κ.λπ. Μια δικτυακή συσκευή αποτελείται από πολλαπλούς πόρους.. Παράδειγμα τέτοιων δικτυακών συσκευών είναι οι μεταγωγείς και οι δρομολογητές. Οι συσκευές δικτύου μπορούν να είναι υλοποιημένες σε υλικό ή λογισμικό και μπορούν να είναι είτε φυσικές είτε εικονικές.

Στα ΕΠΔ, οι δικτυακές συσκευές είναι υπεύθυνες για τη προώθηση και την επεξεργασία των δεδομένων (πακέτων). Οι μεταγωγείς σε ένα SDN δίκτυο συχνά χρησιμοποιούνται ως βασικό προσβάσιμο υλικό προώθησης μέσω μιας διεπαφής (south-bound interface), καθώς η λογική του ελέγχου και οι αλγόριθμοι διατηρούνται σε έναν ελεγκτή. Τέτοιες συσκευές προώθησης αναφέρονται συνήθως ως "SDN switches" στην ορολογία των SDN. Πολλοί SDN μεταγωγείς συμπεριφέρονται σαν ένα τυπικό Ethernet μεταγωγέα και υποστηρίζουν πρωτόκολλα όπως το address resolution protocol (ARP). Ωστόσο, είναι δυνατό ένας

SDN μεταγωγέας να ρυθμιστεί, σε μια αποκλειστική λειτουργία προώθησης, όπου μόνο οι ροές που επιτρέπονται ή διαμορφώνονται από τον ελεγκτή λειτουργούν.

Η απόδοση των SDN μεταγωγέων εξαρτάται από την αποτελεσματική χρήση της μνήμης. Όσο πιο μεγάλο είναι το δίκτυο τόσο περισσότερες είναι και οι καταχωρίσεις μέσα στις συσκευές για αυτό η μνήμη πρέπει να είναι επαρκής

2.1.6 Ελεγκτές ΕΠΔ (Επίπεδο ελέγχου)

Ο ελεγκτής ανήκει στο επίπεδο ελέγχου και είναι ο πυρήνας των SDN δικτύων, καθώς είναι το κύριο μέρος της λειτουργίας τους. Βρίσκεται μεταξύ των δικτυακών συσκευών στο ένα άκρο και των SDN εφαρμογών στο άλλο άκρο. Ένας ελεγκτής SDN ευθύνεται για τη δημιουργία κάθε ροής στο δίκτυο και την εγκατάσταση της καταχώρησης κάθε ροής στις συσκευές προώθησης. Οι βασικές λειτουργίες των SDN ελεγκτών είναι δύο. Η πρώτη έχει σκοπό τον έλεγχο του δικτύου, συμπεριλαμβανομένων των πολιτικών που επιβάλλονται από το επίπεδο εφαρμογής και την εγκατάσταση των κανόνων προώθησης στην υποδομή του δικτύου. Η δεύτερη λειτουργία του είναι παρακολούθηση του δικτύου στο σύνολο του. Ο ελεγκτής μέσω της νότιος διεπαφής μεταφράζει τις καθορισμένες πολιτικές εφαρμογή σε κανόνες προώθησης πακέτων. Το κύριο μέλημα αυτής της διαδικασίας είναι η εξασφάλιση της εγκυρότητας και της συνέπειας των κανόνων προώθησης. Ο ελεγκτής μέσω της βόρειας διεπαφής αξιοποιεί τις πληροφορίες που συλλέγονται από την υποδομή για την κατάσταση του δικτύου, με την λήψη αποφάσεων από τις εφαρμογές. Όπως προαναφέρθηκε στην προηγούμενη ενότητα ελεγκτής ο SDN έχει δύο διεπαφές την νότια διαπαφή, η οποία χαρακτηρίζεται ως η υποδομή του ελεγκτή και ασχολείται με τις συναλλαγές με το επίπεδο υποδομής, δηλ. τη συλλογή της κατάστασης δικτύου και ενημερώνει τους κανόνες προώθησης πακέτων στις συσκευές μεταγωγής. Και την βόρεια διεπαφή, η οποία χαρακτηρίζεται ως ελεγκτής εφαρμογής, χειρίζεται τις συναλλαγές με το επίπεδο εφαρμογή.

Οι λειτουργίες ρύθμισης των ροών διακρίνονται σε δύο κατηγορίες: στις ανταποκριτικές (reactive) και στις προληπτικές (proactive). Στις proactive ρυθμίσεις, οι κανόνες ροής είναι προεγκατεστημένοι στους πίνακες ροής. Έτσι, η ρύθμιση ροής πραγματοποιείται πριν φτάσει το πρώτο πακέτο μιας ροής στον SDN μεταγωγέα και έτσι δεν χρειάζεται αυτός να ρωτήσει τον ελεγκτή για το τι θα κάνει με την διαχείριση της . Τα κυριότερα πλεονεκτήματα της proactive ρύθμισης της ροής είναι η μείωση της καθυστέρησης εγκατάστασης των ροών και η μείωση της συχνής επαφής των

SDN μεταγωγέων με τον ελεγκτή. Σε σχέση με μια ενεργό ροή ρυθμίσεων, ο κανόνας ροής ρυθμίζεται από τον ελεγκτή μόνο εάν δεν υπάρχει εγγραφή τους πίνακες ροής και αυτό γίνεται μόλις το πρώτο πακέτο μιας ροής φτάνει στον SDN μεταγωγέα. Έτσι, μόνο το πρώτο πακέτο ενεργοποιεί μια επικοινωνία μεταξύ του μεταγωγέα και του ελεγκτή. Αυτές οι καταχωρίσεις ροής λήγουν μετά από ένα προκαθορισμένο χρονικό όριο αδράνειας και θα πρέπει να επαναληφθούν. Η reactive ρύθμιση των ροών πάσχει από χρόνο ανταπόκρισης (round trip time), αλλά παρέχει ένα συγκεκριμένο βαθμό ευελιξίας για τη λήψη αποφάσεων της διαχείρισης των ροών όταν εφαρμόζονται δικτυακές πολιτικές όπως QoS γιατί η ρύθμιση των ροών γίνεται με βάση τις συνθήκες του δικτύου σε πραγματικό χρόνο.

Λόγω του ότι υπάρχει μεγάλη πληθώρα από ελεγκτές οι οποίοι έχουν διαφορετικά χαρακτηριστικά, η επιλογή ενός ως του καλύτερου εξαρτάται από τις ανάγκες των δικτυακών εφαρμογών που θέλουμε να υλοποιήσουμε. Οι ελεγκτές μπορούν να χωριστούν σε 2, κατηγορίες στους εμπορικούς «κλειστού κώδικα» και στους ελεγκτές ανοιχτού λογισμικού. Στην παρούσα εργασία ερευνήσαμε μόνο ελεγκτές ανοιχτού κώδικα και τους αναφέρουμε παρακάτω.

2.1.6.1 NOX

Ο NOX [22] είναι ο πρώτος, διαθέσιμος στο κοινό, ελεγκτής που δημιουργήθηκε για το OpenFlow ενός νήματος (single-thread) και είναι γραμμένος σε C++. Αναπτύχθηκε από την Nicira, αλλά στην εξέλιξή του συμμετείχαν τα πανεπιστήμια του Στάνφορντ και του Μπέρκλεϊ. Υποστηρίζει μόνο την πρώτη έκδοση του OpenFlow και η εξέλιξη του έχει σταματήσει από το 2012.

2.1.6.2 Maestro

Ο Maestro [21] είναι ένα λειτουργικό σύστημα δικτύου γραμμένο σε Java. Παρέχει διεπαφές για την υλοποίηση εφαρμογών για την πρόσβαση την τροποποίηση της κατάστασης του δικτύου. Ο Maestro επωφελείται από το τεχνολογία πολλών πυρήνων επεξεργαστών (multicore) για την εκτέλεση παραλληλισμού σε χαμηλό επίπεδο, διατηρώντας παράλληλα ένα απλό μοντέλο προγραμματισμού για τους προγραμματιστές των εφαρμογών. Επιτυγχάνει την απόδοση μέσω της διανομής διεργασιών ισομερώς πάνω από διαθέσιμα νήματα. Αυτή η τεχνολογία του επιτρέπει να επεξεργάζεται μία παρτίδα αιτήσεων ροής, αυξάνοντας ταυτόχρονα έτσι την απόδοση του.

2.1.6.3 Beacon

Ο Beacon [35] δημιουργήθηκε στο Πανεπιστήμιο του Στάνφορντ. Είναι ένα πολυκομματικός ελεγκτής γραμμένος σε Java. Το βασικό χαρακτηριστικό του, είναι η δυνατότητα του να μπορούμε να επιτύχουμε εγκατάσταση και ανανέωση εφαρμογών κατά την λειτουργία του χωρίς να χρειάζεται επανεκκίνηση.

2.1.6.4 Ryu

Ο Ryu [30] είναι ένας ελεγκτής γραμμένος εξολοκλήρου στη γλώσσα προγραμματισμού Python, διαθέτει κάποια έτοιμα πρόσθετα και υποστηρίζει όλες τις εκδόσεις του πρωτοκόλλου OpenFlow.

2.1.6.5 ONOS

Ο Open Network Operating System (ONOS) [23] είναι γραμμένος στη γλώσσα προγραμματισμούJava και υποστηρίζει και αυτός πρόσθετες εφαρμογές. Το βασικό του χαρακτηριστικό, είναι ότι υποστηρίζει κατανεμημένο έλεγχο. Δηλαδή πολλοί ελεγκτές με τα ίδια χαρακτηρίστηκα συνεργάζονται για τον έλεγχο του δικτύου. Αυτή η δυνατότητα κάνει το ΕΠΔ ανεκτικό στα σφάλματα, δηλαδή αν υπάρχει αστοχία του συστήματος σε έναν από τους ελεγκτές το δίκτυο δεν καταρρέει.

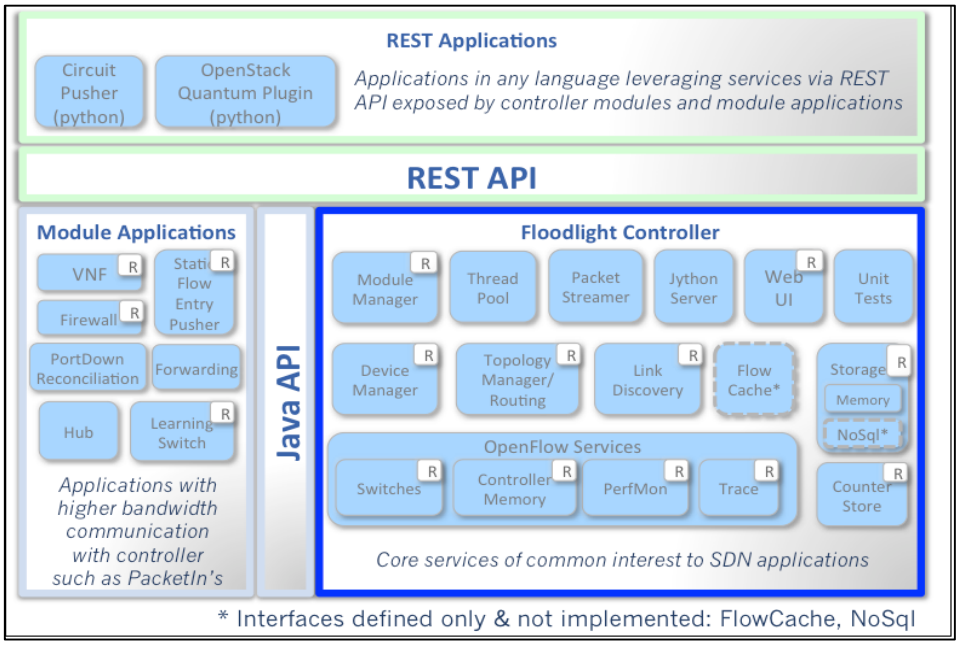
2.1.6.6 OpenDaylight

Ο OpenDaylight [24][25] είναι και αυτός γραμμένος στη γλώσσα προγραμματισμού Java και η δημιουργία του βασίζεται στην ελεγκτή Beacon. Το βασικό του χαρακτηριστικό είναι ότι πέρα από το OpenFlow υποστηρίζει και άλλα νότια (south-bound) APIs.

2.1.6.7 Floodlight

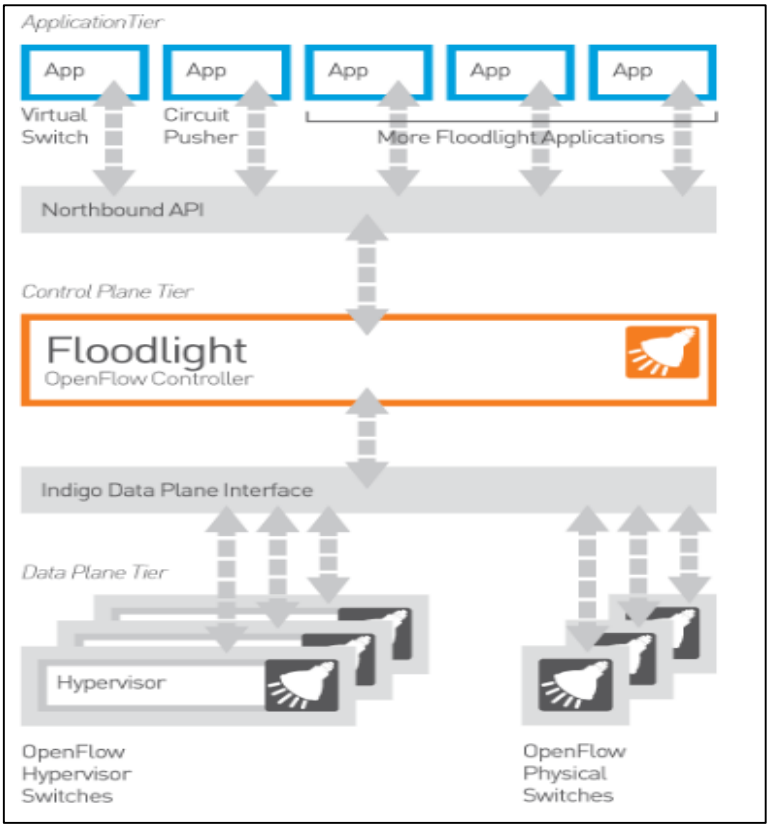
Ο Floodlight [15] είναι ένας ελεγκτής ΕΠΔ που αναπτύχθηκε από μια ανοιχτή κοινότητα προγραμματιστών, από τους οποίους πολλοί προέρχονταν από την Big Switch Networks. Χρησιμοποιεί το πρωτόκολλο OpenFlow για να ενορχηστρώσει τις ροές κυκλοφορίας σε ένα περιβάλλον ΕΠΔ. Είναι ένας απλός ελεγκτής γραμμένος σε Java και προέρχεται από τον Beacon. Αυτή τη στιγμή υποστηρίζεται και βελτιώνεται από μια μεγάλη κοινότητα, συμπεριλαμβανομένων μεγάλων εταιριών της πληροφορικής όπως την Intel, την Cisco, της HP και την IBMΟ ελεγκτής Floodlight είχε αρχικά προσφερθεί από το Big Switch ως μέρος του Project OpenDaylight, αλλά τον Ιούνιο του 2013, ο Big Switch αποχώρησε από το project.

Μερικά από τα Χαρακτηριστικά του Floodlight είναι ότι προσφέρει αρκετά πρόσθετα τα οποία μπορούν εύκολα να επεκταθούν (Εικόνα 2.3). Η ρύθμιση του μέσα σε ένα δίκτυο είναι πολύ εύκολη και δεν απαιτεί εξαρτήσεις. Υποστηρίζει ένα ευρύ φάσμα από εικονικούς και φυσικούς μεταγωγείς OpenFlow. Μπορεί να διαχειρίζεται μικτά OpenFlow και μη-OpenFlow δίκτυα. Σχεδιάστηκε για να έχει υψηλή απόδοση γιατί είναι πολυνηματικός (multi-threaded) και υποστηρίζει την πλατφόρμα ενορχήστρωσης «cloud» OpenStack.



Εικόνα 2.3 - Floodlight Controller [15]

Ο Floodlight δεν είναι απλά ένας ελεγκτής ΕΠΔ, εμπεριέχει μια συλλογή εφαρμογών οι οποίες είναι ενσωματωμένες στην «κορυφή» (Εικόνα 2.4). Πέρα από το ότι υποστηρίζει ένα σύνολο κοινών λειτουργιών, για τον έλεγχο ενός OpenFlow δικτύου, οι εφαρμογές πάνω από αυτόν πραγματοποιούν διάφορες λειτουργίες για την επίλυση διαφορετικών αναγκών των χρηστών του δικτύου.



Εικόνα 2.4 - Αρχιτεκτονική Floodlightελεγκτή [15]

Ο Floodlight είναι ο ελεγκτής τον οποίο αποφασίσαμε να χρησιμοποιήσουμε για την πειραματική προσέγγιση αυτής της εργασίας. Θεωρήσαμε ότι ο Floodlight είναι καλύτερος ελεγκτής για να εξυπηρετήσει τις ανάγκες της εργασίας μας λόγω των βασικών χαρακτηριστικών του και επειδή στις προεγκατεστημένες εφαρμογές τις όποιες υποστηρίζει, υπάρχουν εφαρμογές με μηχανισμούς εξισορρόπησης φορτίου. Αυτοί οι μηχανισμοί, καθώς και η εφαρμογή του Floodlight στο πείραμα μας, περιγράφονται στο κεφάλαιο της ανάλυσης της πειραματικής προσέγγισης.

2.1.7 Εφαρμογές ΕΠΔ(Επίπεδο εφαρμογής)

Οι εφαρμογές δικτύου είναι προγράμματα που επικοινωνούν με τον ΕΠΔ ελεγκτή μέσω APIs. Αυτές οι εφαρμογές μπορούν να δημιουργήσουν μια εικόνα του δικτύου συλλέγοντας πληροφορίες από τον ελεγκτή με σκοπό τη λήψη αποφάσεων. Αυτές οι εφαρμογές θα μπορούσαν να συμπεριλαμβάνουν διαχείριση του δικτύου, αναλύσεις ή επιχειρηματικές εφαρμογές. Για παράδειγμα, μπορεί να δημιουργηθεί μια εφαρμογή ανάλυσης για να αναγνωριστεί η δραστηριότητα του δικτύου για σκοπούς ασφαλείας. Σε ένα ΕΠΔ, οι πάροχοι υπηρεσιών μπορούν να δημιουργήσουν

πολλές εφαρμογές, με στόχο τη μείωση του κόστους, τη βελτίωση της εμπειρίας του πελάτη κ.ά. Είναι αναμενόμενο ότι δεν θα είναι εντελώς νέες όλες οι εφαρμογές ΕΠΔ. Πολλές από αυτές αναπαράγουν ή βελτιώνουν εφαρμογές που εκτελούνται σε δρομολογητές και μεταγωγείς (επίπεδο ελέγχου και δεδομένων). Για παράδειγμα, οι εφαρμογές δρομολόγησης επιτρέπουν τη λήψη αποφάσεων δρομολόγησης με βάση την γνώση και τα χαρακτηριστικά του επιπέδου εφαρμογής. Επιπλέον, χρησιμοποιώντας εφαρμογές ΕΠΔ, η δρομολόγηση περιεχομένου μπορεί να σχεδιαστεί για να διεξάγει ελέγχους διαθεσιμότητας υπηρεσιών πριν από την μεταβίβαση των ροών στους μεταγωγείς ΕΠΔ. Συμπεραίνεται ότι αν και το ΕΠΔ επικεντρώνεται κυρίως στο επίπεδο ελέγχου και δεδομένων, οι εφαρμογές δικτύου έχουν επίσης μερίδιο για τη βελτίωση των λειτουργιών τόσο των διαχειριστών του δικτύου όσο και των χρηστών.

2.1.8 Νότια διεπαφή (Southbound Interface)

Μια σημαντική πτυχή των ΕΠΔ είναι η σύνδεση μεταξύ του επίπεδου δεδομένων και του επιπέδου ελέγχου. Αυτό πετυχαίνεται μέσω της νότιας διεπαφής η οποία επιτρέπει τον ελεγκτή να αλληλεπιδρά με τις συσκευές προώθησης που βρίσκονται στο επίπεδο υποδομής. Τα Southbound APIs διευκολύνουν τον αποτελεσματικό έλεγχο του δικτύου και ενεργοποιούν τον ελεγκτή για να κάνει δυναμικά αλλαγές σύμφωνα με τις απαιτήσεις και τις πολιτικές που έχουν οριστεί από τις εφαρμογές, σε πραγματικό χρόνο. Το OpenFlow, το οποίο αναπτύχθηκε από το ONF, είναι το πρώτο και ίσως το πιο σημαντικό southbound API. Ενώ το OpenFlow είναι το πιο γνωστό southbound API, δεν είναι το μόνο διαθέσιμο ή το μόνο σε ανάπτυξη.

2.1.9 Βόρεια διεπαφή (Northbound Interface)

Η επικοινωνία μέσω της βόρειας διεπαφής (Northbound) αναφέρεται στην επικοινωνία μεταξύ του επιπέδου εφαρμογής και του επίπεδου ελέγχου. Ένα northbound API χρησιμοποιείται για την υλοποίηση και ανάπτυξη ανεξάρτητων εφαρμογών όπως διαχείρισης και παρακολούθησης του δικτύου, εξισορρόπησης φορτίου κ.λπ.

Ένα άλλο πλεονέκτημα των northbound APIs είναι ότι είναι εύκολο να τροποποιηθούν χρησιμοποιώντας υψηλού επιπέδου γλώσσες προγραμματισμού όπως Python, Java, C ++ κτλ. Προς το παρόν δεν υπάρχει αποδεκτό πρότυπο. Η βόρεια

διεπαφή καθορίζεται εξ ολοκλήρου από λογισμικό, ενώ οι αλληλεπιδράσεις του ελεγκτή με τους μεταγωγείς υλοποιούνται σε υλικό (hardware). Ενώ υπάρχουν πολλοί ελεγκτές σε λειτουργία, οι διεπαφές των εφαρμογών τους βρίσκονται ακόμη σε αρχικά στάδια και είναι ανεξάρτητες η μία από την άλλη. Μέχρι να προκύψει ένα σαφές πρότυπο βόρειας διεπαφής, οι εφαρμογές ΕΠΔ θα εξακολουθήσουν να αναπτύσσονται με «ad hoc» τρόπο και η έννοια των ευέλικτων και φορητών εφαρμογών δικτύου ίσως χρειαστεί να περιμένει για κάποιο χρονικό διάστημα

2.1.10 Τεχνολογία OpenFlow

Με το πρωτόκολλο OpenFlow πραγματοποιείται η σύνδεση του επιπέδου έλεγχου και του επιπέδου προώθησης σε ένα ΕΠΔ. Όπως προαναφέρθηκε είναι ένα south-bound API. Είναι ένα έργο ανοικτού κώδικα το οποίο αποσκοπεί στο να προσφέρει ένα εύκολα προγραμματιζόμενο και ανοικτό περιβάλλον δικτύωσης, με σκοπό την εφαρμογή νέων τεχνολογιών, μεθόδων, αλγορίθμων δρομολόγησης και ασφάλειας δικτύων.

Το OpenFlow είναι ένα πρωτόκολλο που παρέχει πρόσβαση στο επίπεδο δεδομένων σε ένα ΕΠΔ. Είναι ένα ανοιχτό πρότυπο και παρέχεται ως επιπλέον χαρακτηριστικό στους εμπορικούς μεταγωγείς, δρομολογητές και ασύρματα σημεία πρόσβασης. Το OpenFlow αυτή τη στιγμή χρησιμοποιείται από μεγάλους προμηθευτές και πλέον υπάρχουν εμπορικά διαθέσιμοι δικτυακοί μεταγωγείς που το υποστηρίζουν.

Το πρωτόκολλο αυτό, επιτρέπει σε απομακρυσμένους ελεγκτές να καθορίσουν τη διαδρομή των πακέτων μέσω των OpenFlow μεταγωγέων σε ένα ΕΠΔ. Ο χωρισμός του επιπέδου ελέγχου απ’ το επίπεδο δρομολόγησης των πακέτων, επιτρέπει μια πιο εξελιγμένη διαχείριση της κίνησης από εκείνη που γίνεται χρησιμοποιώντας πρωτόκολλα δρομολόγησης και λίστες ελέγχου πρόσβασης (Access control lists). Επίσης, το πρωτόκολλο OpenFlow επιτρέπει στους μεταγωγείς από διαφορετικούς προμηθευτές (συχνά ο καθένας με τα δικά του χαρακτηριστικά) να είναι διαχειρίσιμοι εξ’ αποστάσεως, χρησιμοποιώντας αυτό το μοναδικό ανοιχτό πρωτόκολλο.

Το OpenFlow είναι ένα ασφαλές πρωτόκολλο γιατί χρησιμοποίει κρυπτογράφηση μέσω του Transport Layer Security (TLS). Οι ελεγκτές θα πρέπει να «ακούν» στην TCP πόρτα 6553 για να καθιερώσουν μια νέα σύνδεση με τους

μεταγωγείς. Νεότερες εκδόσεις του OpenFlow πρωτοκόλλου χρησιμοποιούσαν ανεπίσημα την πόρτα 6633. Η έκδοση 1.1 του OpenFlow κυκλοφόρησε τον Φεβρουάριο του 2011 και η ανάπτυξη του προτύπου διαχειρίζεται απ' την ανοιχτή κοινότητα δικτύωσης Open Networking Foundation (ONF). Τον Δεκέμβριο του 2011 ο ONF ενέκρινε την έκδοση 1.2 η οποία δημοσιεύτηκε τον Φεβρουάριο του 2012. Η τωρινή έκδοση του OpenFlow είναι η 1.5.

Το OpenFlow χρησιμοποιεί πίνακες-ροής (flow-tables) που είναι παρόμοιοι με τους πίνακες προώθησης (forwarding tables), στους παραδοσιακούςEthernet δρομολογητές και μεταγωγείς.

Η αρχιτεκτονική του OpenFlow αποτελείται από τρία βασικά υποσυστήματα:

1. OpenFlow-συμβατούς μεταγωγείς που συνθέτουν το επίπεδο δεδομένων.
2. Το επίπεδο ελέγχου που αποτελείται από έναν ή περισσότερους OpenFlow ελεγκτές.
3. Ένα ασφαλές κανάλι ελέγχου το οποίο συνδέει τους μεταγωγείς με το επίπεδο ελέγχου.

Οι μεταγωγείς έχουν δύο ειδών κανάλια επικοινωνίας. Μέσω του πρώτου καναλιού επικοινωνούν μεταξύ τους αλλά και με τους τελικούς χρήστες (διαδρομή δεδομένων - data path). Επιπρόσθετα, χρησιμοποιούν το δεύτερο κανάλι για την επικοινωνία τους με τον ελεγκτή μέσω του πρωτοκόλλου OpenFlow (διαδρομή ελέγχου - control path).

2.2 Κατηγοριοποίηση τεχνολογιών εξισορρόπησης φορτίου.

Η εξισορρόπηση φορτίου είναι ένα σημαντικό ζήτημα για το σχεδιασμό πολλών κατανεμημένων συστημάτων. Ως εξισορρόπηση φορτίου ορίζουμε την τεχνική διαμοιρασμού του φορτίου εργασίας σε πολλαπλούς πόρους (διακομιστές ή δικτυακές συσκευές)με σκοπό τη βελτίωση της απόδοσης των εφαρμογών και της βέλτιστης αξιοποίησης των πόρων των διακομιστών [19].

Οι μηχανισμοί εξισορρόπησης φορτίου έχουν μελετηθεί ευρέως από θεωρητική άποψη και εφαρμόζονται συχνά σε πραγματικά συστήματα. Η αποτελεσματικότητα τους εξαρτάται σε μεγάλο βαθμό από το υπό εξέταση σενάριο. Για παράδειγμα, εξισορρόπηση φορτίου σε ένα κέντρο δεδομένων είναι πολύ διαφορετική από την εξισορρόπηση φορτίου σε έναν ISP (Internet Service Provider). Υποκείμενες υποθέσεις σχετικά με τις διαθέσιμες πληροφορίες, όπως η κατάσταση

των επεξεργαστικών πόρων των διακομιστών (π.χ. τρέχον φορτίο CPU, ποσοστό χρήσης μνήμης) και τη ζήτηση (π.χ. χρόνος επεξεργασίας ενός αιτήματος) επηρεάζουν σε μεγάλο βαθμό το σχεδιασμό ενός αποτελεσματικού μηχανισμού εξισορρόπησης φορτίου. Παρά τη μακρά ιστορία της, η έρευνα στους μηχανισμούς εξισορρόπησης φορτίου συνεχίζεται, καθώς διαφορετικά σενάρια και παραδοχές προκύπτουν από τις νέες τεχνολογικές εξελίξεις και εφαρμογές. Στην ερευνά μας θα επικεντρωθούμε ειδικά στην εξισορρόπηση φορτίου στα κέντρα δεδομένων.

Λόγω της ιδιαίτερης τοπολογίας και των χαρακτηριστικών της Διαδικτυακής κυκλοφορίας, οι παραδοσιακοί αλγόριθμοι εξισορρόπησης φορτίου δεν είναι κατάλληλοι για το δικτυακό περιβάλλον των κέντρων δεδομένων. Από την άλλη πλευρά, στη συμβατική στατική εξισορρόπηση φορτίου οι αλγόριθμοι είναι αγνωστικιστικοί σε πληροφορίες, όπως π.χ. το ποσοστό χρήσης των επεξεργαστικών πόρων, και συχνά μπορεί να οδηγήσουν σε συμφόρηση των διακομιστών [19].

Τα τελευταία χρόνια, οι ερευνητές και οι μηχανικοί πρότειναν ένα μεγάλο αριθμό μηχανισμών για να αντιμετωπιστεί το πρόβλημα εξισορρόπησης φορτίου στα κέντρα δεδομένων. Η εστίαση αυτών των μηχανισμών ποικίλλει πολύ. Συνοψίζοντας, οι στόχοι ενός μηχανισμού εξισορρόπησης φορτίου περιλαμβάνουν κυρίως: την απόδοση, την καθυστέρηση, την ευρωστία, την επεκτασιμότητα και την ενεργειακή αποδοτικότητα.

Πιο συγκεκριμένα:

- Η απόδοση και η καθυστέρηση εστιάζουν στη βελτίωση της χρήσης του δικτύου και τη μείωση του χρόνου ολοκλήρωσης των δικτυακών ροών.
- Η ευρωστία αναφέρεται στο πως αντιδρούν οι μηχανισμοί σε μεταβολές της τοπολογίας, όπως τις αποτυχίες συνδέσεων και την διακοπή λειτουργίας συστημάτων (π.χ. διακομιστές).
- Η κλιμάκωση (scalability) απαιτεί την ανάπτυξη αντίστοιχων μηχανισμών σε μεγάλης κλίμακας δίκτυα, όμως με αποδεκτό κόστος.
- Οι μηχανισμοί ενεργειακής απόδοσης προσπαθούν να ολοκληρώσουν τις εργασίες με έναν μέτριο αριθμό συσκευών, για τη μείωση της κατανάλωσης ενέργειας.

Στην εργασία μας θα επικεντρωθούμε σε μηχανισμούς εξισορρόπησης φορτίου που εστιάζουν στην αύξηση της απόδοσης και την μείωση της καθυστέρησης.

Εξισορρόπηση φορτίου διακομιστών

Οι μηχανισμοί εξισορρόπησης φορτίου συνοψίζονται σε δύο κατηγορίες, στους μηχανισμούς εξισορρόπησης φορτίου συνδέσμων (link load balancing) και στους μηχανισμούς εξισορρόπησης φορτίου διακομιστών. Στην παρούσα εργασία επικεντρωνόμαστε στην εξισορρόπηση φορτίου μεταξύ των διακομιστών (server load balancing).

Οι μηχανισμοί εξισορρόπησης φορτίου διακομιστών είναι σχεδιασμένοι για να εξισορροπούν την επισκεψιμότητα μεταξύ των διακομιστών. Μπορούν να ταξινομηθούν περαιτέρω σε δυο κατηγορίες: σε μηχανισμούς εξισορρόπησης φορτίου διακομιστών επίπεδου 4 (μεταφοράς) και επίπεδου 7 (εφαρμογής).

Οι μηχανισμοί εξισορρόπησης φορτίου επιπέδου 4 δεν γνωρίζουν πληροφορίες για το είδος της εφαρμογής και διαχειρίζονται την κίνηση των ροών του δικτύου βασιζόμενοι στη διεύθυνση IP και τις θύρες. Η εξισορρόπηση φορτίου διακομιστών επιπέδου 4 διαδραματίζει σημαντικό ρόλο στην κλιμάκωση των Διαδικτυακών υπηρεσιών, όπως την αναζήτηση στο Web, το ηλεκτρονικό εμπόριο κλπ. Αυτές οι Διαδικτυακές υπηρεσίες συνήθως εκτελούνται σε πολλούς διακομιστές σε κέντρα δεδομένων, και κάθε διακομιστής έχει μια μεμονωμένη διεύθυνση IP (DIP). Μια υπηρεσία εκτίθεται μέσω μίας ή περισσοτέρων εικονικών διευθύνσεων IP (VIP) προς τους χρήστες. Ο μηχανισμός εξισορρόπησης φορτίου λαμβάνει την κίνηση που προορίζεται για αυτήν την VIP και εξισορροπεί το φορτίο ανάμεσα στους διακομιστές, βάσει των αντίστοιχων DIP διευθύνσεων.

Οι μηχανισμοί εξισορρόπησης φορτίου διακομιστών επιπέδου 7 γνωρίζουν πληροφορίες για τις εφαρμογές. Μπορούν να επιθεωρήσουν το περιεχόμενο των αιτημάτων και να τα δρομολογήσουν σύμφωνα με τις πληροφορίες του επιπέδου εφαρμογής (επίπεδο7 κατά OSI).

2.2.1 Στατικοί και δυναμικοί αλγόριθμοι εξισορρόπησης φορτίου

2.2.1.1 Στατικοί αλγόριθμοι εξισορρόπησης φορτίου

Οι μηχανισμοί εξισορρόπησης φορτίου χωρίζονται σε δυο ακόμη κατηγορίες, στους στατικούς και τους δυναμικούς. Οι στατικοί αλγόριθμοι λαμβάνουν υπόψη τους τις πληροφορίες όπως και την κατάσταση των διακομιστών η

την κίνηση του δικτύου. Ένας γνωστός και ευρέως διαδεδομένος στατικός αλγόριθμος είναι ο αλγόριθμος κυκλικής επιλογής (Round-Robin).

Ο αρχικός τρόπος με τον οποίο δουλεύει αυτός ο μηχανισμός είναι να εκχωρεί μία προς μία όλες τις ροές σε όλες τις εφικτές διαδρομές προς τους διακομιστές. Δεδομένου δε ότι λαμβάνει υπόψη του πληροφορίες σχετικές με τη διαδικτυακή κίνηση, είναι πολύ πιθανό να υπάρξει συμφόρηση στο δίκτυο σε σενάρια όταν υπάρχει μεγάλος φόρτος. Στην περίπτωση που έχουμε ετερογένεια στο σύστημα μας (π.χ. διακομιστές με διαφορετικές δυνατότητες) μια εξέλιξη του συγκεκριμένου αλγορίθμου είναι ο αλγόριθμος κυκλικής επιλογής με βάρη (Weighted Round Robin): Ο αλγόριθμος παραμένει ο ίδιος απλώς δίνεται προτεραιότητα στους διακομιστές που έχουν μεγαλύτερη απόδοση.

2.2.1.2 Δυναμικοί αλγόριθμοι εξισορρόπησης φορτίου

Οι δυναμικοί αλγόριθμοι λαμβάνουν υπόψη πληροφορίες όπως οι ενεργές συνδέσεις, την κατανάλωση των επεξεργαστικών πόρων των διακομιστών, τον φόρτο τις διαδικτυακής κίνησης κ.α. σε πραγματικό χρόνο. Αυτοί οι αλγόριθμοι διαχειρίζονται τα αιτήματα προς τους διακομιστές δυναμικά, υπολογίζοντας την καλύτερη διαδρομή σύμφωνα με τις πληροφορίες που τους παρέχονται. Παρακάτω περιγράφονται δύο γνωστοί δυναμικοί αλγόριθμοι εξισορρόπησης φορτίου.

Αλγόριθμος δυναμικής εξισορρόπησης κυκλικής επιλογής (dynamic weighted round robin): Η βασική λειτουργία του αλγορίθμου αυτού είναι ίδια με αυτή της κυκλικής επιλογής με βάρη. Η διαφορά τους εντοπίζεται στην δυναμική απόδοση τιμών για τα βάρη ανάλογα με τις παραμέτρους που έχουν επιλεγεί. **Αλγόριθμος εξισορρόπησης λιγότερης συμφόρησης (least congestion)**: Αυτός ο μηχανισμός συλλέγει πληροφορίες για κάθε διαθέσιμη διαδρομή σε πραγματικό χρόνο και επιλέγει την διαδρομή με τη μικρότερη συμφόρηση. Η αποδοτικότητα αυτού του μηχανισμού συσχετίζεται σε μεγάλο βαθμό με την ακρίβεια και την απόκριση των πληροφοριών για κάθε διαθέσιμη διαδρομή.

2.2.2 Κεντροποιημένοι και Κατανεμημένοι Μηχανισμοί εξισορρόπησης φορτίου.

Οι υπάρχοντες μηχανισμοί εξισορρόπησης φορτίου θα μπορούσαν να ταξινομηθούν ευρέως σε Κεντρικούς Μηχανισμούς και Κατανεμημένους Μηχανισμούς.

Κεντροποιημένοι Μηχανισμοί: Στους κεντροποιημένους μηχανισμούς ένα λογισμικό που εκτελείται σε κεντρικό ελεγκτή συλλέγει πληροφορίες για τη συμφόρηση ανάμεσα στους διακομιστές και εκχωρεί τις ροές ανάμεσα σε αυτούς σύμφωνα τη συνολική άποψη που έχει σχηματίσει για το δίκτυο. Για παράδειγμα, ο ελεγκτής αξιοποιεί τα πρωτόκολλα OpenFlow σε επικοινωνία με τους μεταγωγείς του δικτύου και μετρούν το δίκτυο μέσω μιας εφαρμογής. Με τη βοήθεια των μετρήσεων της εφαρμογής, η μηχανισμός εξισορρόπησης φορτίου μπορεί εύκολα να αποκτήσει πληροφορίες σχετικά με την κυκλοφορία ανάμεσα στα κανάλια του δικτύου και να χρησιμοποιήσει τις πληροφορίες αυτές για την εξισορρόπηση του φόρτου. Ωστόσο, η πρόσθετη επικοινωνία ανάμεσα στις δικτυακές συσκευές, τους διακομιστές και τον ελεγκτή αυξάνει πολύ η επιβάρυνση (overhead) του δικτύου

Κατανεμημένοι Μηχανισμοί Αντίθετα με τους κεντροποιημένους μηχανισμούς οι κατανεμημένοι δεν χρησιμοποιούν κάποιο κεντρικό λογισμικό για την εξισορρόπηση του φορτίου και υπολογίζουν τις διαδρομές στους κόμβους του δικτύου (υπολογιστές ή μεταγωγείς). Αυτοί οι μηχανισμοί είναι αρκετά γρήγοροι στο να επιλέξουν κάποια διαδρομή, για να εξισορροπήσουν τον φόρτο μέσα σε ένα δίκτυο. Από την άλλη πλευρά όμως είναι πολύ δύσκολο κάθε κόμβος του δικτύου να συλλέξει πληροφορίες για όλο το δίκτυο και η γνώση τους συνήθως περιορίζεται στις πληροφορίες που συλλέγουν από και προς τα κανάλια των γειτονικών τους κόμβων. Επιπλέον, οι κατανεμημένοι μηχανισμοί είναι συνήθως σχεδιασμένοι για συμμετρικά δίκτυα και χρησιμοποιούν στατικούς αλγόριθμους που είναι δύσκολο να αντιδράσουν σε αλλαγές τοπολογίας, όπως οι αποτυχίες συνδέσεων.

2.2.3 Εξισορρόπηση φορτίου ανά ροή (per flow) και ανά πακέτο (per packet)

Οι μηχανισμοί εξισορρόπησης φορτίου μπορούν να χωριστούν σε δυο ακόμη κατηγορίες: στους μηχανισμούς εξισορρόπησης φορτίου ανά ροή (per flow) και εξισορρόπηση φορτίου ανά πακέτο (per packet).

Η εξισορρόπηση φορτίου ανά πακέτο συναντάται κυρίως στον διαμοιρασμό της κυκλοφορίας ανάμεσα στις δικτυακές συσκευές και όχι ανάμεσα στους διακομιστές, καθώς αυτοί πρέπει να εξυπηρετήσουν το κάθε αίτημα ξεχωριστά. Η εξισορρόπηση φορτίου ανά πακέτο σημαίνει ότι μία δικτυακή συσκευή (δρομολογητής η μεταγωγές) στέλνει ένα πακέτο για τον προορισμό1 μέσω της

πρώτης διαδρομής, το δεύτερο πακέτο για τον ίδιο προορισμό (1) πάνω από τη δεύτερη διαδρομή κ.ο.κ. Η εξισορρόπηση φορτίου ανά πακέτο εγγυάται ίσο φορτίο σε όλους τους συνδέσμους (links). Ωστόσο, υπάρχει πιθανότητα τα πακέτα να φτάσουν εκτός σειράς στον προορισμό, επειδή μπορεί να υπάρχει διαφορετική καθυστέρηση εντός του δικτύου, αυτό οδηγεί αυτομάτως σε αποτυχία τη σύνδεση και επαναπροώθηση (retransmission) των πακέτων.

Η εξισορρόπηση φορτίου ανά προορισμό σημαίνει ότι ο δρομολογητής διανέμει τα πακέτα με βάση τη διεύθυνση του αποστολέα. Δεδομένου ότι υπάρχουν διαφορετικά αιτήματα πελατών για τον ίδιο προορισμό (π.χ μια εικονική διεύθυνση VIP) όλα τα πακέτα για κάθε TCP σύνδεση από τον αποστολέα 1 σε αυτό το δίκτυο περνούν από την πρώτη διαδρομή και όλα τα πακέτα από τον αποστολέα 2 σε αυτό το δίκτυο περνούν από τη δεύτερη διαδρομή κ.ο.κ. Αυτός ο μηχανισμός διατηρεί την σειρά πακέτων, με πιθανή άνιση χρήση των συνδέσμων. Το μειονεκτήματα αυτής της προσέγγισης είναι δύο. Πρώτον ότι για στις δικτυακές συσκευές που ανήκουν στον πυρήνα(core) του δικτύου θα υπάρχουν πάρα πολλές καταχωρίσεις (μια για κάθε ροή/ TCP σύνδεση), οπότε οι απαιτήσεις μνήμης και επεξεργασίας καθίστανται πολύ απαιτητικές. Δεύτερον η κάθε TCP σύνδεση δεν έχει την ίδια χρονική διάρκεια και συνεπώς τον ίδιο όγκο δεδομένων που πρέπει να μεταφερθεί. Αυτό έχει σαν αποτέλεσμα μεγάλες σε διάρκεια και χρόνο ροές (ονομάζονται elephant flows [36]) να σωρευτούν σε ένα σύνδεσμο προς μία κατεύθυνση.

2.3 Βασικότεροι μηχανισμοί εξισορρόπησης φορτίου

Η βιβλιογραφική μας επισκόπηση περιλαμβάνει τη μελέτη προτάσεων παρόμοιων με το αντικείμενο της εργασίας μας. Η έρευνα μας εστίασε στην αναζήτηση δημοσιεύσεων σχετικών με τους μηχανισμούς εξισορρόπησης φορτίου στα ΕΠΔ. Αφού συγκεντρώσαμε κάποιο υλικό από τις σχετικές δημοσιεύσεις, αναλύσαμε και συγκρίναμε τις υπάρχουσες λύσεις οι σημαντικότερες εκ των οποίων αναφέρονται παρακάτω:

Ο αλγόριθμος Hedera [18] στοχεύει στην εξισορρόπηση μεγάλων ροών στα κέντρα δεδομένων. Ο Hedera ανιχνεύει τις μεγάλες ροές στους μεταγωγείς, επιλέγει ένα δίαυλο του οποίου η χωρητικότητα μπορεί να φιλοξενήσει αυτές τις ροές και εγκαθιστά τις διαδρομές στους αντίστοιχους μεταγωγείς. Μια ροή θεωρείται μεγάλη όταν ο όγκος της είναι μεγαλύτερος από το 10% της χωρητικότητας του διαύλου.

Ο αλγόριθμος FDALB [12] είναι κεντροποιημένος και χρησιμοποίει έναν ελεγκτή για να συλλέγει πληροφορίες σχετικά με την κατάσταση του δικτύου. Διαχωρίζει τις ροές σε μικρές και μεγάλες. Οι μεγάλες ροές ανιχνεύονται και «σημειώνονται» από τους διακομιστές και στέλνονται στον ελεγκτή και έπειτα μεταφέρονται στους διαύλους με την μικρότερη κίνηση, οι μικρές ροές μεταδίδονται από τους μεταγωγείς με στατικό τρόπο.

Στην εργασία [26] ο προτεινόμενος αλγόριθμος εγκαθιστά τους κανόνες στους μεταγωγεις προληπτικά (proactive). Στόχος του είναι να κατευθύνει τα αιτήματα μεγάλων ομάδων πελατών, χωρίς να εμπλέκει τον ελεγκτή για να εξοικονομήσει χώρο στον πίνακα ροών των μεταγωγέων και να μειώσει την καθυστέρηση του δικτύου. Για να επαληθευθεί η εγκυρότητα του αλγορίθμου χρησιμοποιήθηκε ο ελεγκτής Floodlight σε εικονικό δικτυακό περιβάλλον. Η εξομοίωση του δικτύου έγινε με τη χρήση του Mininet. Το σύστημα, χρησιμοποιεί το sFlow για να λαμβάνει στατιστικά στοιχεία από τους μεταγωγείς και το Host sFlow project για να εξάγει στατιστικά χρήσης (utilization) των επεξεργαστών στους διακομιστές, σε πραγματικό χρόνο.

Ο αλγόριθμος [27] επιχειρεί να δώσει λύση στο πρόβλημα του πως η εγκατάσταση ροής για κάθε αίτημα οδηγεί σε τεράστιο αριθμό κανόνων στους πίνακες ροών των μεταγωγέων. Ο αλγόριθμος χρησιμοποιεί κανόνες «μπαλαντέρ» για ένα σύνολο αιτημάτων και εκχωρεί τους κανόνες προληπτικά στους διακομιστές. Με αυτό τον τρόπο μεγάλα μεγέθη επισκεψιμότητας πελατών μεταφέρονται σε μία συστάδα διακομιστών.

Στην εργασία [9], ο προτεινόμενος αλγόριθμος εκχωρεί τα αιτήματα των πελατών στο διακομιστή με τον μικρότερο φόρτο. Για τον υπολογισμό του φόρτου λαμβάνει υπόψη την δικτυακή κίνηση, τον χρόνο απόκρισης των αιτημάτων, καθώς και το ποσοστό χρήσης των πόρων των επεξεργαστών και την υπολογιστική ισχύ των διακομιστών. Ο ελεγκτής που χρησιμοποιήθηκε είναι ο Floodlight.

Στην ερευνητική δουλειά [32] εφαρμόζονται ένας στατικός και ένας δυναμικός αλγόριθμος εξισορρόπησης φορτίου. Ο δυναμικός αλγόριθμος συλλέγει πληροφορίες για τη δικτυακή κίνηση και επιλεγεί τον διακομιστή με τις λιγότερες συνδέσεις για να κατευθύνει τα αιτήματα. Ο ελεγκτής που χρησιμοποιείται είναι ο Floodlight.

Κατά τη βιβλιογραφική έρευνα μας παρατηρήσαμε ότι υπάρχουν αρκετές προτάσεις για την εξισορρόπηση φορτίου. Οι περισσότεροι από τους δυναμικούς

αλγορίθμους που προτείνονται εστιάζουν στη δικτυακή κίνηση και κάποιοι από αυτούς στο μέγεθος των ροών. Επίσης υπάρχουν κάποιοι μηχανισμοί που για τον υπολογισμό του φορτίου πέρα από την δικτυακή κίνηση λαμβάνουν υπόψη τους την επεξεργαστική κατανάλωση των διακομιστών. Εντοπίσαμε ότι οι σχετικές προτάσεις δεν έχουν τη δυνατότητα δυναμικής προσαρμογής της εξισορρόπησης φορτίου στο είδος της εφαρμογής. Στην συνέχεια προτείνουμε μια δική μας μέθοδο εξισορρόπησης φορτίου, η οποία βασίζεται στη δικτυακή κίνηση αλλά και στους διαθέσιμους υπολογιστικούς πόρους των διακομιστών, λαμβάνοντας υπόψη της το είδος των εφαρμογών που εκτελούνται

2.4 Μηχανισμός εξισορρόπησης φορτίου προσαρμοσμένος στην εφαρμογή

Σε αυτό το σημείο, παρουσιάζουμε τη δική μας πρόταση για την εξισορρόπηση φορτίου: έναν μηχανισμό που διαμοιράζει το φορτίο σύμφωνα με το είδος της εφαρμογής και βασίζεται σε πληροφορίες της δικτυακής κίνησης και της επεξεργαστικής κατανάλωσης των διακομιστών. Του δώσαμε την ονομασία εξισορρόπηση φορτίου που προσαρμόζεται στην εφαρμογή (application adaptive load balancing, AALB).

Υπάρχουν πολλοί παράγοντες που θα μπορούσαν να συμβάλουν στο φορτίο ενός διακομιστή. Στην εργασία μας εξετάζουμε το πώς κάθε διακομιστής επηρεάζεται ανάλογα με την παρεχόμενη υπηρεσία. Στο μηχανισμό μας, για τον υπολογισμό του φορτίου των διακομιστών αθροίζουμε το ποσοστό της χρήσης του επεξεργαστή (CPU), τη χρήση της μνήμης και το μέγεθος της δικτυακής κυκλοφορίας.

Διάφοροι παράγοντες, όπως το είδος της υπηρεσίας ή τα τεχνικά χαρακτηριστικά του κάθε διακομιστή, έχουν διαφορετικό βαθμό επίδρασης στο φορτίο. Θέλουμε ο αλγόριθμος μας να προσαρμόζεται στο είδος των εφαρμογών. Άρα, για τον υπολογισμό του φορτίου των διακομιστών θεωρούμε ότι δεν είναι αντιπροσωπευτικό μόνο το παραπάνω άθροισμα μετρικών(χρήση επεξεργαστή, μνήμης και δικτυακή κυκλοφορία) και προσθέσαμε βάρη σε κάθε μια μετρική, η οποία δείχνει τη βαρύτητα της κάθε μιας στο τελικό φορτίο. Για παράδειγμα, ο ίδιος ακριβώς αλγόριθμος (AALB) θα μπορούσε να χρησιμοποιηθεί σε ένα ετερογενές σύμπλεγμα διακομιστών, όπου τα τεχνικά χαρακτηριστικά των διακομιστών διαφέρουν, όπως στην εργασία [26]. Στην περίπτωση αυτή η εξισορρόπηση φορτίου

δεν πρέπει να λαμβάνει υπόψη της μόνο το φορτίο, αλλά επίσης την ικανότητα επεξεργασίας του κάθε διακομιστή.

Στο δικτυακό περιβάλλον της εργασίας μας όλα τα χαρακτηριστικά του δικτύου είναι ομοιογενή. Όποτε ο μοναδικός παράγοντας που επηρεάζει το φορτίο των διακομιστών είναι η παρεχόμενη υπηρεσία. Στόχος αυτού του αλγορίθμου είναι να εξυπηρετεί το φορτίο ανάμεσα σε διακομιστές υπολογίζοντας τις «ανάγκες» της κάθε εφαρμογής. Για παράδειγμα, έχουμε μία υπηρεσία για την οποία η εξυπηρέτηση των αιτημάτων προς αυτήν καταλαμβάνει μεγάλο μέρος της μνήμης, συγκριτικά με τις άλλες υπηρεσίες που παρέχονται σε έναν διακομιστή, τότε τα αιτήματα προς αυτήν πρέπει να κατευθύνονται προς το διακομιστή με την περισσότερη διαθέσιμη μνήμη. Ο τύπος του προτεινόμενου αλγορίθμου περιγράφεται παρακάτω.

$$L = w_1 * R(CPU) + w_2 * R(mem) + w_3 * R(band)$$

Όπου R(CPU) το ποσοστό χρήσης του επεξεργαστή, όπου R(mem) το ποσοστό χρήσης της μνήμης, όπου R(band) το ποσοστό της δικτυακής κίνησης. Τα w_1, w_2, w_3 είναι τα βάρη που έχουμε προσθέσει σε κάθε μετρική. Η τιμή του κάθε βάρους ορίζεται, αφού πρώτα ερευνήσουμε για κάθε υπηρεσία το πως επηρεάζονται οι επεξεργαστικοί πόροι των διακομιστών. Η αντίστοιχη μελέτη αποτελεί ειδικό σενάριο της πειραματικής μας ανάλυσης (Σενάριο 1). Η τιμή του L υποδεικνύει το βαθμό φόρτωσης του διακομιστή και η μέγιστη τιμή του όπως και μέγιστη τιμή των βαρών (w)είναι το 1. Όσο μεγαλύτερο είναι το L, τόσο πιο φορτωμένος είναι ο διακομιστής. όσο μικρότερη είναι η τιμή, τόσο πιο ελαφρύ είναι το φορτίο. Για παράδειγμα, έχουμε μία υπηρεσία για την οποία η εξυπηρέτηση των αιτημάτων προς αυτήν, καταλαμβάνει μεγάλο μέρος της μνήμης, συγκριτικά με τις άλλες υπηρεσίες που παρέχονται σε ένα διακομιστή. Τότε το βάρος της μνήμης για τον υπολογισμό του φόρτου θα είναι μεγαλύτερο για αυτήν, από ότι σε άλλες υπηρεσίες. Ο αλγόριθμος αντιλαμβάνεται το είδος της εφαρμογής, σύμφωνα με την πόρτα που χρησιμοποιεί και τα βάρη της κάθε εφαρμογής ορίζονται από τον διαχειριστή.

2.4.1 Υλοποίηση

Για την υλοποίηση του αλγορίθμου αρχικά δημιουργήσαμε ένα πρόγραμμα το οποίο λαμβάνει τα ποσοστά της κατανάλωσης των επεξεργαστικών πόρων του κάθε

διακομιστή (CPU usage, Memory usage) και στην συνέχεια τα αποθηκεύει σε ένα αρχείο σε μορφή JSON (Εικόνα 3.9). Για τη συγκεκριμένη υλοποίηση χρησιμοποιήσαμε την υπηρεσία Docker Stats γιατί κάθε διακομιστής είναι ένας διαφορετικός υποδοχέας Docker Container.


```
{"container":"e128797e2f3f","name":"mn.d4","memory":{"raw":"37.62MiB /
128MiB","percent":"29.39%"},"cpu":"0.20%"}

{"container":"68539585b0f4","name":"mn.d3","memory":{"raw":"40.98MiB /
128MiB","percent":"32.02%"},"cpu":"0.21%"}

{"container":"81dd43024c6d","name":"mn.d2","memory":{"raw":"48.17MiB /
128MiB","percent":"37.63%"},"cpu":"0.17%"}

{"container":"1d6c2ee856e7","name":"mn.d1","memory":{"raw":"46.97MiB /
```


Εικόνα 2.5 - Αποτελέσματα Docker Stats

Στην συνέχεια δημιουργήσαμε μια κλάση που την ονομάσαμε Monitoring στον Floodlight controller μέσω της οποίας αντλούμε πληροφορίες για την τρέχουσα κατανάλωση επεξεργαστή και μνήμης για κάθε διακομιστή σύμφωνα με το όνομα του. (Παράρτημα Α - Κλάση Monitoring). Η πηγή από την οποία αντλεί τις πληροφορίες της χρήσης του επεξεργαστή (CPU) και της μνήμης (RAM)είναι το αρχείο μορφής JSON που περιγράφτηκε παραπάνω. Οι πληροφορίες της δικτυακής κυκλοφορίας συλλέγονται από το OpenVswitch.

Ο αλγόριθμος διατηρεί μια λίστα με τιμές για κάθε διακομιστή. Η κάθε τιμή είναι το αποτέλεσμα του τύπου που αναφέρθηκε στο προηγούμενο υποκεφάλαιο. Ο διακομιστής που έχει την μικρότερη τιμή θεωρείται καλύτερος και όλα τα αιτήματα των πελατών εκχωρούνται προς αυτόν (Παράρτημα Α - Συλλογή).

3 Πειραματική προσέγγιση

Κατά την πειραματική προσέγγιση της εργασίας προσομοιώσαμε ένα δικτυακό περιβάλλον ΕΠΔ, με σκοπό την μελέτη και την αξιολόγηση μηχανισμών εξισορρόπησης φορτίου, που στόχο έχουν την βελτίωση του τρόπου διαμοιρασμού της δικτυακής κυκλοφορίας ανάμεσα στους διακομιστές. Επιλέξαμε την τεχνολογία των ΕΠΔ γιατί θέλουμε να αναδείξουμε το πώς μπορούν μέσω της χρήσης της, να υλοποιηθούν και να βελτιστοποιηθούν δικτυακές πολιτικές όπως η εξισορρόπηση φορτίου.

Το δίκτυο που αναπτύξαμε είναι ομοιογενές δηλαδή οι διακομιστές, οι δίαυλοι επικοινωνίας και οι δικτυακές συσκευές που εμπεριέχονται σε αυτό είναι ίσων δυνατοτήτων. Ο λόγος για τον οποίο επιλέξαμε να έχουμε ομοιογένεια στο δίκτυο, είναι γιατί θέλουμε εστιάσουμε την εργασία μας στην ετερογένεια των διαδικτυακών υπηρεσιών που παρέχονται σε κάθε διακομιστή. Η εξομοίωση του δικτυακού περιβάλλοντος υλοποιήθηκε με τον εξομοιωτή δικτύων Mininet και την χρήση του OpenFlow πρωτόκολλου και του Open V switch μεταγωγέα [20]. Επιλέξαμε το OpenVswitch γιατί υποστηρίζει το OpenFlow, υλοποιείται εικονικά μόνο με λογισμικό χωρίς την ανάγκη ξεχωριστού υλικού, είναι ο πιο διαδεδομένος εικονικός μεγαγωγέας και επίσης υποστηρίζεται από το Mininet.

Ο ελεγκτής που χρησιμοποιήσαμε για τα πειράματα μας είναι ο Floodlight. Επιλέξαμε τον Floodlight γιατί: υποστηρίζει το OpenFlow, είναι ανοιχτού λογισμικού, είναι πολυνηματικός (multithreaded) άρα έχει πολύ υψηλή απόδοση, η διαχείριση του είναι πολύ εύκολη μέσω του REST API που διαθέτει, εμπεριέχει μια συλλογή προτύπων (εφαρμογών) τα οποία μπορούν εύκολα να επεκταθούν και στις προεγκατεστημένες εφαρμογές τις οποίες υποστηρίζει υπάρχουν εφαρμογές με μηχανισμούς εξισορρόπησης φορτίου.

Οι υπηρεσίες που παρέχουμε μέσω των διακομιστών, είναι μια υπηρεσία παροχής ιστοσελίδων και μια υπηρεσία ζωντανής μετάδοσης βίντεο. Επιλέξαμε αυτές της δύο υπηρεσίες διότι θέλουμε να αναδείξουμε το πρόβλημα του μεγέθους των ροών [36] και να εξετάσουμε το αν και κατά πόσο αυτές επηρεάζουν την εξισορρόπηση του φορτίου ανάμεσα στους διακομιστές. Πιο συγκεκριμένα η υπηρεσία που παρέχει ιστοσελίδες παράγει ροές μικρού μεγέθους και διάρκειας ζωής (mice flows [36]), ενώ στην υπηρεσία βίντεο η ροές είναι πολλαπλάσια μεγαλύτερες (elephant flows [36]).

Οι τρεις μηχανισμοί εξισορρόπησης φορτίου που χρησιμοποιήσαμε είναι κεντροποιημένοι (centralized). Δηλαδή ένα λογισμικό που εκτελείται σε έναν κεντρικό ελεγκτή συλλέγει πληροφορίες για το δίκτυο και εκχωρεί τις ροές ανάμεσα στους διακομιστές, σύμφωνα με το μηχανισμό εξισορρόπησης φορτίου που έχει επιλεγεί να χρησιμοποιηθεί. Επίσης και οι τρεις μηχανισμοί είναι επιπέδου τέσσερα (server load balancing) επειδή διαχειρίζονται τα αιτήματα των πελατών σύμφωνα με την εικονική διεύθυνση και την θύρα που προορισμού του κάθε αιτήματος. Ο πρώτος αλγόριθμος είναι στατικός και εξυπηρετεί τα αιτήματα κυκλικά ένα προς ένα σε κάθε διακομιστή (Simple Round Robin). Ο δεύτερος είναι δυναμικός βάσει του κυκλοφοριακού φορτίου (Statistics). Ο τρίτος μηχανισμός είναι μία δικιά μας πρόταση εξισορρόπησης φορτίου και ο αλγόριθμος του υλοποιήθηκε από εμάς. Είναι ένας δυναμικός αλγόριθμος, ο οποίος βασίζεται στη δικτυακή κίνηση αλλά και στους διαθέσιμους υπολογιστικούς πόρους των διακομιστών λαμβάνοντας υπόψη το είδος των εφαρμογών που παρέχονται από αυτούς (Application Adaptive Load Balancing).

Στο πρώτο υποκεφάλαιο γίνεται μια αναφορά στα εργαλεία που χρησιμοποιήσαμε, στο δεύτερο περιγράφουμε το περιβάλλον δοκιμών, στο τρίτο περιγράφουμε αναλυτικά τον τρόπο με τον οποίο έγινε η ανάπτυξη των εφαρμογών και η ρύθμιση των εργαλείων. Τέλος, στο τέταρτο υποκεφάλαιο περιγράφουμε τα χαρακτηριστικά και τον τρόπο λειτουργίας του δικού μας μηχανισμού εξισορρόπησης φορτίου.

3.1 Επισκόπηση εργαλείων

Η εξομοίωση του δικτύου στην πειραματική προσέγγιση της εργασίας υλοποιήθηκε μέσω του εργαλείου Containernet, το οποίο είναι μια προέκταση του Mininet και οι διακομίστες του δικτυού είναι Docker containers . Για την υπηρεσία της παροχής βίντεο χρησιμοποιήσαμε το VLC και για την υπηρεσία παροχής ιστοσελίδων το Flask. Η προσομοίωση των πελατών, δηλαδή τα αιτήματα προς τις υπηρεσίες και η μέτρηση της απόδοσης τους, έγινε μέσω του εργαλείου JMeter.

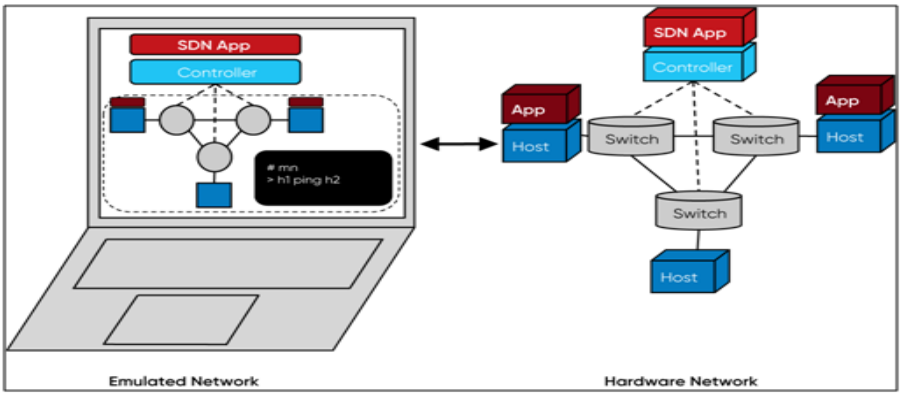
3.1.1 Mininet

Το Mininet [11] είναι ένας ενορχηστρωτής εξομοίωσης δικτύου. Μέσω αυτού μπορούμε να προσομοιώσουμε και να διαχειριστούμε ένα δικτυακό περιβάλλον. Λειτουργεί πάνω σε έναν πυρήνα Linuxκαι χρησιμοποιεί μεθόδους ελαφριάς εικονικοποίησης (lightweight virtualization). Έχει την δυνατότητα να προσομοιώνει

διαφορετικές συσκευές ενός δικτύου όπως τελικούς υπολογιστές (hosts), μεταγωγείς (switches), δρομολογητές (routers) και συνδέσμους (Εικόνα 3.1).

Ένας εξομοιωτής Mininet συμπεριφέρεται ακριβώς όπως μια πραγματική μηχανή και μπορεί να γίνει σύνδεση σε αυτήν με ssh (Secure Shell) για την διαχείριση της. Τα προγράμματα που εκτελούνται μπορούν να στέλνουν πακέτα μέσω μιας διεπαφής που μοιάζει με αληθινή Ethernet διεπαφή, με συγκεκριμένη ταχύτητα σύνδεσης και καθυστέρηση. Τα εικονικά στοιχεία του Mininet όπως οι εξυπηρετητές (hosts), οι μεταγωγείς, οι συνδέσεις και οι ελεγκτές δημιουργούνται χρησιμοποιώντας μόνο λογισμικό και όχι υλικό (hardware) και η συμπεριφορά τους είναι όμοια με αυτή των στοιχείων υλικού.

Τα πλεονεκτήματα του Mininet είναι πολλά. Ανάμεσα σε αυτά συγκαταλέγονται η ταχύτητα ανάπτυξης των δικτύων (μερικά δευτερόλεπτα είναι αρκετά για να δημιουργηθεί ένα απλό δίκτυο), η δυνατότητα ανάπτυξης προσαρμοσμένων τοπολογιών δικτύου, η δυνατότητα εκτέλεσης προγραμμάτων που είναι διαθέσιμα στο λειτουργικό σύστημα Linux. Έχει τη δυνατότητα προγραμματισμού των μεταγωγέων (switches) μέσω του OpenFlow πρωτοκόλλου. Είναι εύκολο στη χρήση του και έχει μικρές σχετικά απαιτήσεις (μπορεί να «τρέξει» μέχρι και σε έναν απλό φορητό υπολογιστή).Διαθέτει μία διεπαφή προγραμματισμού σε γλώσσα Python που μέσω αυτής μπορεί να γίνει η ανάπτυξη των εικονικών δικτύων. Τέλος το Mininet είναι ένα πρόγραμμα ανοιχτού λογισμικού, ανοιχτό σε όλους για περεταίρω βελτιώσεις-προσθήκες ή διόρθωση προβλημάτων, κάτι που το κάνει να είναι σε διαρκή ανάπτυξη.



Εικόνα 3.1 - Εικονικό και πραγματικό δίκτυο [11]

Με τη βοήθεια της Python διεπαφής προγραμματισμού που διαθέτει το Mininet, μπορούν να δημιουργηθούν παραμετροποιήσιμες δικτυακές τοπολογίες γράφοντας προγράμματα python. Οι σημαντικότερες κλάσεις και μεταβλητές περιλαμβάνουν:

- Topo: Η βασική κλάση που δημιουργεί τοπολογίες.
- Build(): Η μέθοδος για την δημιουργία τοπολογιών.
- addSwitch(): Προσθέτει έναν μεταγωγέα (switch) στην τοπολογία
- addHost(): Προσθέτει έναν υπολογιστή (host) στην τοπολογία
- addLink(): Προσθέτει έναν σύνδεσμο στην τοπολογία ανάμεσα σε δύο οντότητες (π.χ ενώνει έναν μεταγωγέα με έναν υπολογιστή .
- Mininet: Η κύρια κλάση (main class) για δημιουργία και διαχείριση δικτύου.
- start(): Ξεκινάει το δίκτυο.
- stop(): Σταματάει το δίκτυο.

3.1.2 Containernet

Το Containernet [7] είναι μια προέκταση του Mininet η οποία και επιτρέπει τη χρήση Docker containers με τη μορφή κεντρικών υπολογιστών σε εξομοιωμένες τοπολογίες δικτύου.

Τα βασικά χαρακτηριστικά και οι δυνατότητες του επιγραμματικά είναι:

- Δυνατότητα προσθαφαίρεσης Docker Container στις τοπολογίες του Mininet
- Σύνδεση Docker Container στην τοπολογία (π.χ με μεταγωγέα)
- Δυνατότητα εκτέλεσης εντολών των Docker χρησιμοποιώντας την γραμμή εντολών του Mininet
- Δυνατότητα δυναμικών αλλαγών στην τοπολογία
- Περιορισμός πόρων των Docker Container (CPU,Memory)

Στο προηγούμενο κεφάλαιο αναφέραμε ότι μέσω μίας προγραμματιστικής διεπαφής (API) σε Python, μπορούμε να δημιουργήσουμε παραμετροποιημένες τοπολογίες. Στην παρακάτω εικόνα (3.2) περιγράφεται το πώς με την χρήση του Containernet και αυτής της διεπαφής μπορούμε να προσθέσουμε Docker containers ως κεντρικούς υπολογιστές (hosts) σε μία τοπολογία. Ποιο συγκεκριμένα αυτός ο κώδικας δημιουργεί τέσσερις hosts (Εικόνα 3.2) και ό καθένας από αυτούς έχει

128MB μέγεθος μνήμης RAM και χρησιμοποιεί το 10% του επεξεργαστή του μηχανήματος που είναι εγκατεστημένο το containernet.

```
info('*** Adding docker containers\n')
d1 = net.addDocker('d1', dimage="servers:latest",cpuset_cpus="0,1",mem_limit='128m', memswap_limit='0m')
d2 = net.addDocker('d2', dimage="servers:latest",cpuset_cpus="0,1",mem_limit='128m', memswap_limit='0m')
d3 = net.addDocker('d3', dimage="servers:latest",cpuset_cpus="0,1",mem_limit='128m', memswap_limit='0m')
d4 = net.addDocker('d4', dimage="servers:latest",cpuset_cpus="0,1",mem_limit='128m', memswap_limit='0m')
```

Εικόνα 3.2 - Πρόσθεση Docker container στο Mininet

3.1.3 VLC

Το VLC [37] είναι ένα ελεύθερο λογισμικό ανοιχτού κώδικα αναπαραγωγής πολυμέσων (multimedia) που αναπτύσσεται και συντηρείται από το VideoLAN project. Είναι ένα πρόγραμμα αναπαραγωγής πολυμέσων, κωδικοποιητής, και streaming υποστηρίζοντας πολλές μορφές κωδικοποίησης (codec) ήχου και βίντεο και διάφορα πρωτόκολλα ροής δεδομένων(streaming). Είναι ικανό να κάνει streaming πάνω από δίκτυα, καθώς και να επανακωδικοποιεί αρχεία πολυμέσων και να τα αποθηκεύει σε διάφορες μορφές. Τα αρχικά VLC σήμαιναν *Video LAN Client*, αλλά επειδή το VLC δεν είναι πια μόνο ένα λογισμικό πελάτης, η ονομασία του δεν είναι πλέον αντιπροσωπευτική αλλά είναι ένα απλό όνομα. Το VLC διαθέτει εκδόσεις για πολλές πλατφόρμες, με εκδόσεις για τα Microsoft Windows, το Mac OS X, το Linux, κ.α.

Το VLC περιλαμβάνει ένα μεγάλο αριθμό από ελεύθερες βιβλιοθήκες κωδικοποίησης και αποκωδικοποίησης, αποφεύγοντας έτσι την ανάγκη για εύρεση κατάλληλων ιδιόκτητων πρόσθετων.

Οι διακομιστές πολυμέσων (media servers) είναι αυτοί που αναλαμβάνουν τη διαδικασία μετάδοσης του περιεχομένου από την πηγή του αρχείου σε πολλαπλούς χρήστες του διαδικτύου. Η αναπαραγωγή μέσω streaming μπορεί να γίνει είτε σε πραγματικό χρόνο (live) την ίδια στιγμή που πραγματοποιείται μία ηχητική ή βίντεο μετάδοση είτε από αρχείο ήχου η βίντεο που είναι υποθηκευμένο σε κάποιον διακομιστή. Αυτό που απαιτείται από τον τελικό χρήστη (client), είναι ένα λογισμικό (πχ Windows Media Player, RealPlayer κ.α.)το οποίο αποσυμπιέζει την πληροφορία και την αναπαράγει με την μορφή εικόνας και ήχου. Κατά την εργασία μας πραγματοποιήθηκαν διάφορες δοκιμές λογισμικών ανοιχτού κώδικα παροχής βίντεο όπως (Red 5,Mist). Επιλέξαμε το VLCγιατί περιλαμβάνει ένα μεγάλο αριθμό από

ελεύθερες βιβλιοθήκες κωδικοποίησης και αποκωδικοποίησης, αποφεύγοντας έτσι την ανάγκη για εύρεση κατάλληλων πρόσθετων. Επίσης είναι μικρό σε μέγεθος, εύκολο στην χρήση του και επιτρέπει τη ρύθμιση του μέσω της κονσόλας των Linux.
Οι παρακάτω μέθοδοι steaming είναι διαθέσιμες για χρήση με τον VLC streaming server:

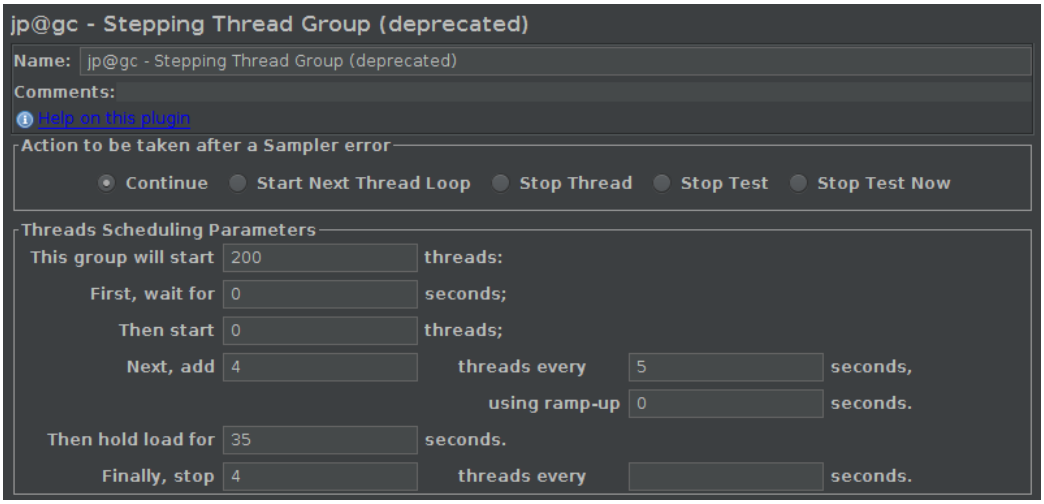
- Τοπικά: εμφάνιση της ροής στην οθόνη
- File: Αποθήκευση της ροής σε ένα αρχείο.
- HTTP: Καθορίζοντας μόνο την TCP πόρτα που θα ακούει ο server (Broadcast)
- MS-WMSP (MMSH): Αυτή η μέθοδος επιτρέπει την ροή στο Microsoft Windows Media Player (Broadcast)
- UDP: Unicast-Multicast
- RTP: Unicast-Multicast

3.1.4 Apache JMeter

Η εφαρμογή Apache JMeter [5] είναι λογισμικό ανοιχτού κώδικα, γραμμένο σε Java που έχει σχεδιαστεί για προσομοίωση πελατών και τη μέτρηση της απόδοσης των εφαρμογών. Αρχικά σχεδιάστηκε για τη δοκιμή εφαρμογών Ιστού, αλλά από τότε έχει επεκταθεί σε άλλες λειτουργίες δοκιμής. Έχει δυνατότητα μεταφόρτωσης και δοκιμής της απόδοσης πολλών διαφορετικών τύπων εφαρμογών, διακομιστών, πρωτοκόλλων, όπως:

- Web - HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, ...)
- SOAP / REST Webservices
- FTP
- Βάση δεδομένων μέσω του JDBC
- LDAP
- Το μεσαίο λογισμικό προσανατολισμού μηνυμάτων (MOM) μέσω του JMS
- Mail - SMTP (S), POP3 (S) και IMAP (S)
- Εγγενείς εντολές ή δέσμες ενεργειών κελύφους
- TCP
- Java Objects

Κατά την διάρκεια της εργασίας χρησιμοποιήσαμε διάφορα εργαλεία παρόμοια με το JMeter όπως το siege και το apache bench. Το apache JMeter ήταν το μοναδικό εργαλείο, μέσω του οποίου μπορούμε να κάνουμε κλιμακωτά αιτήματα στους διακομιστές χωρίς να περιμένουμε να τελειώσει η προηγούμενη σύνδεση (Εικόνα 3.3). Για παράδειγμα η υπηρεσία video streaming παρέχει ένα video 30 δευτερολέπτων μέσω του πρωτοκόλλου HTTP. Αν κάνουμε 2 αιτήματα μέσω του siege το δεύτερο αίτημα θα ξεκινήσει σε 30 δευτερόλεπτα περιμένοντας να τελειώσει το πρώτο.



Εικόνα 3.3 - Περιβάλλον εργασίας JMeter

3.1.5 Docker Container

Ένα Container είναι μια τυποποιημένη μονάδα λογισμικού που πακετάρει τον κώδικα και όλες τις εξαρτήσεις του, έτσι ώστε η εφαρμογή να τρέχει γρήγορα και αξιόπιστα από ένα περιβάλλον υπολογιστών σε άλλο. Ένα Docker image είναι ένα ελαφρύ, αυτόνομο και εκτελέσιμο πακέτο λογισμικού που περιλαμβάνει όλα όσα απαιτούνται για την εκτέλεση μιας εφαρμογής όπως: κώδικας, χρόνος εκτέλεσης, εργαλεία συστήματος, βιβλιοθήκες συστήματος και ρυθμίσεις.

Τα container images γίνονται containers κατά τη διάρκεια εκτέλεσης τους και στην περίπτωση των Docker containers [10] τα images γίνονται containers όταν εκτελούνται στο Docker engine. Τα Docker containers είναι διαθέσιμα τόσο για εφαρμογές που βασίζονται στο Linux όσο και για Windows, το λογισμικό μέσα στον container θα εκτελείται πάντοτε με τον ίδιο τρόπο, ανεξάρτητα από την υποδομή. Τα container απομονώνουν το λογισμικό από το περιβάλλον του και εξασφαλίζουν ότι

λειτουργεί ομοιόμορφα παρά τις διαφορές στο λειτουργικό σύστημα που τα φιλοξενεί.

Στην συγκεκριμένη εργασία χρησιμοποιήσαμε τα Docker containers για να απομονώσουμε τις εφαρμογές που παρέχουν Διαδικτυακές υπηρεσίες. Χρησιμοποιήσαμε τα Docker Containers στο περιβάλλον εργασίας του Mininet προσομοιώνοντας τα σαν ξεχωριστά υπολογιστικά σύστημα εξυπηρετητών.

3.1.6 Flask

Το Flask [13][14] είναι ένα framework για την ανάπτυξη υπηρεσιών ιστού μέσω της γλώσσας προγραμματισμού Python. Είναι μια βιβλιοθήκη ή μια συλλογή βιβλιοθηκών που στοχεύει στην επίλυση μέρους κάποιων προβλημάτων που παρουσιάζονται στην δημιουργία υπηρεσιών ιστού. Κατά την ανάπτυξη εφαρμογών ιστού υπάρχουν ορισμένα προβλήματα που θα πρέπει πάντα να λυθούν, όπως η δρομολόγηση από διευθύνσεις URL στους πόρους του συστήματος, την εισαγωγή δυναμικών δεδομένων σε HTML και την αλληλεπίδραση με έναν τελικό χρήστη. Το Flask είναι ένα μικρό σε μέγεθος framework επειδή υλοποιεί μόνο βασικές λειτουργίες (συμπεριλαμβανομένης και της δρομολόγησης) αλλά αφήνει πιο προηγμένες λειτουργίες (π.χ. έλεγχο ταυτότητας) σε επεκτάσεις (extensions) . Αυτό το κάνει εύχρηστο για τους νέους χρήστες.

3.2 Περιβάλλον δοκιμών

Όλα τα εργαλεία που προαναφέρθηκαν εγκαταστάθηκαν σε λειτουργικό σύστημα Ubuntu 16.04. Οι εκδόσεις των εργαλείων που χρησιμοποιήσαμε και τα τεχνικά χαρακτηριστικά του υπολογιστή στον οποίο εκτελέσαμε τα πειράματα αναφέρονται στους παρακάτω πίνακες (Πίνακας 3.1, 3.2).

CPU	i7-4510U quadcore	2793MHz
Memory	Samsung 8GiB	1600MHz
Storage	Samsung SSD 850pro	250GB

Πίνακας 3.1- Τεχνικά χαρακτηριστικά υπολογιστή

Όνομα	Έκδοση
Docker	18.09.1
Flask	1.0.2
Floodlight	1.2
Jmater	5.0
Mininet	2.3.0
Openvswitch	2.10.0
VLC	2.2.2

Πίνακας 3.2 - Εκδόσεις λογισμικών

3.3 Εξομοίωση Εφαρμογών

Σε αυτό το κεφάλαιο περιγράφονται αναλυτικά τα βήματα για την υλοποίηση της πειραματικής προσέγγισης της εργασίας.

3.3.1 Εφαρμογές Διαδικτυακών υπηρεσιών

Το πρώτο βήμα για την εκτέλεση των πειραμάτων είναι η δημιουργία των υπηρεσιών που παρέχονται από τους διακομιστές.

3.3.1.1 Υπηρεσία παροχής ιστοσελίδων (Web service)

Για την εξυπηρέτηση των ιστοσελίδων χρησιμοποιήσαμε το flask. Δημιουργήσαμε ένα HTML αρχείο (εικόνα 3.4) το οποίο παραπέμπει σε 3 διαφορετικές ιστοσελίδες διαφορετικού μεγέθους: μικρού, μεσαίου και μεγάλου (πίνακας 3.3). Ο πλήρης κώδικας του αλγορίθμου βρίσκεται στο Παράρτημα Α - Flask.

Εικόνα 3.4 - Αρχική Σελίδα

Όνομα	Μέγεθος bytes	href (link's URLs)
Light	26259 (0,26MB)	1
Medium	1021425 (1MB)	43
Heavy	17692570 (17MB)	77

Πίνακας 3.3 - Ιστοσελίδες που παρέχονται μέσω του flask

3.3.1.2 Video Streaming Service

Κατά την εργασία μας πραγματοποιήθηκαν διάφορες δοκιμές video streaming server λογισμικών ανοιχτού κώδικα όπως Red 5 και ο Mist. Για την εργασία μας καταλήξαμε να χρησιμοποιήσουμε το VLC streaming server γιατί περιλαμβάνει ένα μεγάλο αριθμό από ελεύθερες βιβλιοθήκες κωδικοποίησης και αποκωδικοποίησης, αποφεύγοντας έτσι την ανάγκη για εύρεση κατάλληλων πρόσθετων. Επίσης είναι μικρό σε μέγεθος, εύκολο στην χρήση του, επιτρέπει την ρύθμιση του μέσω της κονσόλας των Linux.

Η μετάδοση του video έγινε μέσω του HTTP πρωτοκόλλου καθώς είναι το μοναδικό στο VLC που υποστηρίζει την έναν προς πολλούς (Broadcast) μετάδοση. Η πόρτα που ορίσαμε να ακούει η συγκεκριμένη υπηρεσία είναι η 8080. Το video το οποίο παρέχεται είναι μεγέθους 5mb και χρονικής διάρκειας 31 δευτερολέπτων.

3.3.2 Δημιουργία Docker image

Το επόμενο βήμα για την εκτέλεση των πειραμάτων μας είναι δημιουργία του Docker Image. Κατά την εκτέλεση του ως Docker Container αυτό το Image κατά τη διάρκεια της εξομοίωσης μας θα χρησιμοποιείται ως κεντρικός υπολογιστής (host). Το image δημιουργήθηκε μέσω ενός Dockerfile και περιγράφεται (Εικόνα 3.5) παρακάτω.

```
#Το αρχικό image που θέλουμε να χρησιμοποιήσουμε
FROM ubuntu:14.04

#Αντιγραφή αρχείων που θα χρησιμοποιήσουμε
COPY SampleVideo_5mb.mp4 SampleVideo_5mb.mp4
COPY vlcVideoServer vlcVideoServer
COPY ftp.py ftp.py
COPY http.py http.py
COPY files files
COPY start_services start_services
COPY templates templates
COPY static static

#Update Ubuntu 14.04
RUN apt-get update && \
#install vlcVideoServer
apt-get -y install vlc && \
#install python
apt-get -y install python && \
apt-get install -y python-pip && \
yes | pip install Flask && \
sed -i 's/geteuid/getppid/' /usr/bin/vlc

#Άνοιγμα των θυρών που θα ακούνε οι servers
EXPOSE 8080
EXPOSE 80

#Έναρξη υπηρεσιών
ENTRYPOINT ["./start_services"]
```

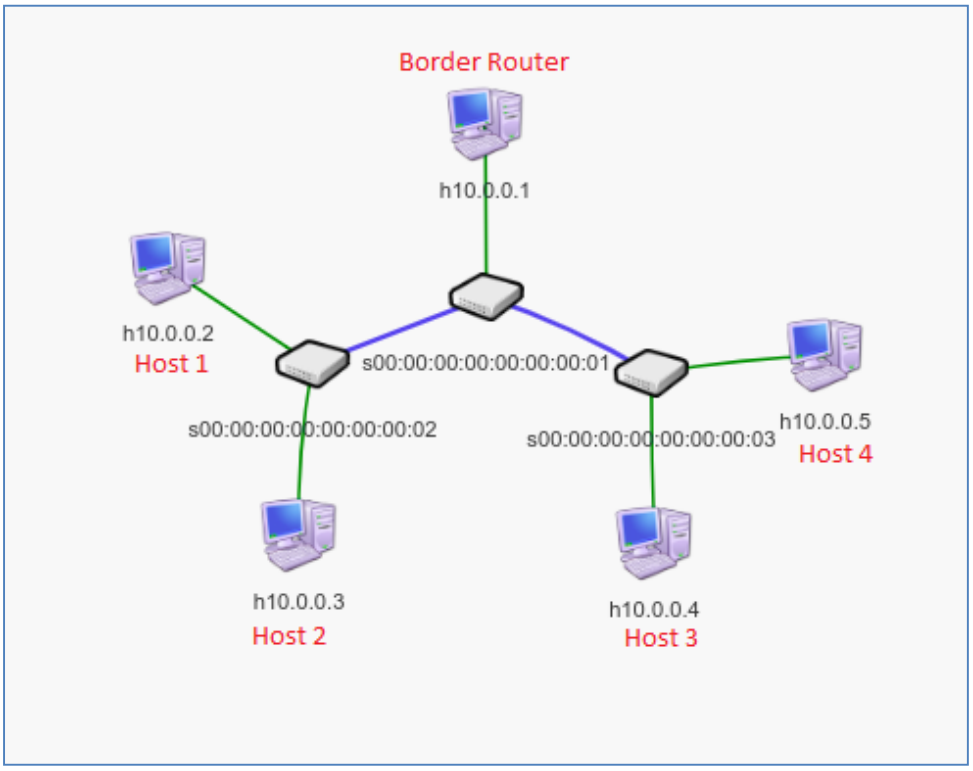
Εικόνα 3.5 - Dockerfile

```
#!/bin/bash
./vlcVideoServer &
python http.py
```

Εικόνα 3.6 - Έναρξη υπηρεσιών

3.3.3 Δημιουργία τοπολογίας

Για την δημιουργία της τοπολογίας μας χρησιμοποιήσαμε Python API του Mininet. Στην τοπολογία εισάγαμε τρείς SDN μεταγωγείς και τους συνδέσαμε με τον Floodlightελεγκτή. Οι μεταγωγείς τρέχουν το OpenVSwitch και το πρωτόκολλο OpenFlow 1.3. Επίσης εισάγαμε τέσσερις υπολογιστές οι όποιοι είναι οι εξυπηρετητές που θα χρησιμοποιούμε στα σενάρια μας. Οι εξυπηρετητές είναι Docker containers των οποίων το image είναι ίδιο και είναι αυτό το οποίο περιγράφηκε παραπάνω. Επίσης μέσα στον τοπολογία εισάγαμε έναν ακόμα κόμβο ο οποίος χρησιμοποιείται ως Border Router για τη σύνδεση του εικονικού δικτύου με τον υπόλοιπο κόσμο και το διαδίκτυο. Επί της ουσίας το interface του border router είναι ένα interface του υπολογιστή στο οποίο τρέχει το πείραμα και είναι συνδεδεμένος μέσω αυτού στην εικονική τοπολογία του Mininet. Ο πλήρης κώδικας του αλγορίθμου βρίσκεται στο Παράρτημα Α -Τοπολογία. Στην παρακάτω εικόνα 3.7, παρουσιάζουμε το δίκτυό μας χρησιμοποιώντας το γραφικό περιβάλλον του Floodlight.



Εικόνα 3.7 - Τοπολογία Mininet

3.3.4 Παρακολούθηση διακομιστών (Server monitoring).

Στην εργασία μας χρειαζόμαστε ένα μηχανισμό για την παρακολούθηση των διακομιστών (που στη συγκεκριμένη πειραματική προσέγγιση είναι Docker containers) για δύο λόγους: πρώτον γιατί θέλουμε να αναλύσουμε τα αποτελέσματα του στα πειραματικά σενάρια της εργασίας και δεύτερον για την χρήση του στον αλγόριθμο εξισορρόπησης φορτίου προσαρμοσμένο στην εφαρμογή. Στην αρχή δοκιμάσαμε με τα εργαλεία influxdb – collectd αλλά αυτή η μέθοδος δεν υποστηρίζει παρακολούθηση container αλλά μόνο υπολογιστών host. Επίσης δοκιμάσαμε μια ακόμα μέθοδο μέσω των εργαλείων Prometheus- CAdvisor τα οποία προτείνονται ως καλύτερα για παρακολούθηση των docker containers. Απορρίψαμε όμως και αυτήν τη μέθοδο γιατί αυξάνει κατά πολύ την πολυπλοκότητα του πειράματος, κατανάλωνε πολλούς από τους επεξεργαστικούς πόρους του υπολογιστή στον οποίο έτρεξαν τα πειράματα και είχε καθυστέρηση στην απόκριση.

Για τις ανάγκες της εργασίας μας λοιπόν θεωρήσαμε καλύτερο να δημιουργήσουμε ένα δικό μας πρόγραμμα το οποίο βασίζεται στην εφαρμογή Docker Stats. Το Docker Stats είναι υπηρεσία η οποία μεταδίδει την κατανάλωση των επεξεργαστικών πόρων των Docker container σε πραγματικό χρόνο. Τρέχει μέσω της γραμμής εντολών των Linux και έχει πολύ υψηλή απόκριση. Χρησιμοποιήσαμε αυτήν την υπηρεσία και υλοποιήσαμε ένα δικό μας πρόγραμμα μέσω του οποίου μπορούμε να αντλήσουμε πληροφορίες όπως μεγίστη και μέση κατανάλωση επεξεργαστή και μνήμης για κάθε container. Ο πλήρης κώδικας του προγράμματος βρίσκεται στο Παράρτημα Α - Monitoring. Στην παρακάτω εικόνα (Εικόνα 3.8) παρουσιάζεται ένα παράδειγμα των αποτελεσμάτων του προγράμματος.

```
mn.d1, Max cpu:15.15%, Max memory:41.76%, Average cpu:0.88%, Average mem: 37.0%
mn.d2, Max cpu:17.27%, Max memory:46.74%, Average cpu:0.81%, Average mem: 35.39%
mn.d3, Max cpu:18.18%, Max memory:39.18%, Average cpu:1.64%, Average mem: 34.32%
mn.d4, Max cpu:16.28%, Max memory:41.89%, Average cpu:1.54%, Average mem: 34.93%
```

Εικόνα 3.8 - Αποτελέσματα DockerMon

4 Πειραματικά αποτελέσματα

Ο στόχος της εργασία μας είναι η μελέτη τεχνικών εξισορρόπησης φορτίου, για την αποφυγή συμφόρησης και τη βελτίωση της απόδοσης Διαδικτυακών υπηρεσιών στα ευφυή προγραμματιζόμενα δίκτυα. Προκειμένου να αναδείξουμε διαφορετικές πλευρές του προβλήματος, ορίσαμε τρία διαφορετικά σενάρια, όπου το καθένα από αυτά στοχεύει στην απάντηση διαφορετικών επιστημονικών ερωτημάτων.

Πιο συγκεκριμένα, στόχος του πρώτου σεναρίου είναι να διερευνήσουμε τον βαθμό της επίδρασης των διαφορετικών υπηρεσιών στους πόρους επεξεργασίας. Για να το πετύχουμε αυτό πραγματοποιήσαμε πειράματα με διαφορετικούς αριθμούς αιτημάτων (100,200 και 400) προς κάθε υπηρεσία, μετρώντας την κατανάλωση των πόρων επεξεργασίας στους διακομιστές. Όπως αναφέραμε στο κεφάλαιο 2.4, τα αποτελέσματα του πρώτου σεναρίου ρυθμίζουν τα βάρη του αλγορίθμου μας AALB.

Στόχος του δεύτερου σεναρίου είναι η αξιολόγηση τριών αλγορίθμων εξισορρόπησης φορτίου. Οι αλγόριθμοι που εξετάζουμε είναι: ο στατικός αλγόριθμος κυκλικής επιλογής (SRR), ο δυναμικός αλγόριθμος που βασίζεται στην δικτυακή κίνηση (Statistics) και ο αλγόριθμος μας (AALB), ο οποίος διαμοιράζει το φορτίο σύμφωνα με το είδος της εφαρμογής, βασιζόμενος σε μετρήσεις της δικτυακής κίνησης και το ποσοστό χρήσης των πόρων επεξεργασίας των διακομιστών. Σε αυτό το σενάριο θέλουμε να ερευνήσουμε πώς επηρεάζονται οι επιδόσεις κάθε εφαρμογής με κάθε αλγόριθμο και κατά πόσο εξισορροπείται η κατανάλωση των πόρων επεξεργασίας ανάμεσα στους διακομιστές.

Στο τρίτο σενάριο επιχειρούμε να αναλύσουμε την απόδοση των αλγορίθμων εξισορροπώντας τα αιτήματα για δύο εφαρμογές ταυτόχρονα. Πιο συγκεκριμένα, σε αυτό το σενάριο θέλουμε να ερευνήσουμε ποιος συνδυασμός αλγορίθμων (και για τις δυο εφαρμογές) έχει την καλύτερη απόδοση ως προς την εξυπηρέτηση των αιτημάτων και κατά πόσο εξισορροπείται η κατανάλωση των πόρων επεξεργασίας, ανάμεσα στους διακομιστές. Δηλαδή, υλοποιήσαμε πειράματα με διαφορετικούς αριθμούς αιτημάτων προς τις δύο εφαρμογές ταυτόχρονα για κάθε πείραμα.

4.1 Σενάριο 1- Υπολογισμός κατανάλωσης πόρων επεξεργασίας για κάθε Διαδικτυακή Υπηρεσία

Στόχος του πρώτου σεναρίου είναι να διερευνήσουμε τον βαθμό της επίδρασης διαφορετικών υπηρεσιών στους πόρους επεξεργασίας των διακομιστών. Για να το επιτύχουμε αυτό, καταστρώσαμε σενάρια με διαφορετικούς αριθμούς αιτημάτων, αξιοποιώντας το εργαλείο Jmeter. Κάθε thread του JMeter εκφράζει 2 αιτήματα (Εικόνα 4.1). Το πρώτο αίτημα κατευθύνεται στον πρώτο εξυπηρετητή (server 1) και το δεύτερο αίτημα στον δεύτερο. Το πρώτο αίτημα είναι προς την υπηρεσία ιστοσελίδων (web), ζητώντας ιστοσελίδα μεσαίου μεγέθους, και το δεύτερο είναι προς τον VLC, κατεβάζοντας τovideo που παρέχεται.

HTTP requests:

1. **http://10.0.0.2:80/medium**
2. **http://10.0.0.3:8080/stream**

Εικόνα 4.1 - Αιτήματα

Πείραμα 1

Για 100 threads στο Jmeter ορίσαμε να ξεκινάνε 2 threads κάθε 5 δευτερόλεπτα. Επίσης ορίσαμε το μέγιστο που θα περιμένει κάθε αίτημα για να ολοκληρωθεί η σύνδεση(session) να είναι 35 δευτερόλεπτα, γιατί το Video έχει χρονική διάρκεια 31 δευτερολέπτων. Άρα κάθε 5 δευτερόλεπτα θα γίνονται συνολικά 4 αιτήματα, 2 στον πρώτο διακομιστή και 2 στον δεύτερο. Το σύνολο των threads και πως αυτά κατανέμονται κατά την χρονική διάρκεια του πειράματος παρουσιάζονται στην εικόνα παρακάτω. Το ίδιο γίνεται αντίστοιχα και για τα επόμενα πειράματα.

4.1.1 Αποτελέσματα

Τα αποτελέσματα που καταγράψαμε για κάθε πείραμα είναι.

- Ο χρόνος ολοκλήρωσης του κάθε αιτήματος σε msec (**Avg Load Time)**
- Η μέγιστη τιμή της χρήσης του επεξεργαστή (**Max CPU**)
- Η μέγιστη τιμή της χρήσης της μνήμης (**Max memory**)
- Η μέση τιμή της χρήσης του επεξεργαστή (**Average cpu**)
- Η μέση τιμή της χρήσης της μνήμης (**Average mem**)

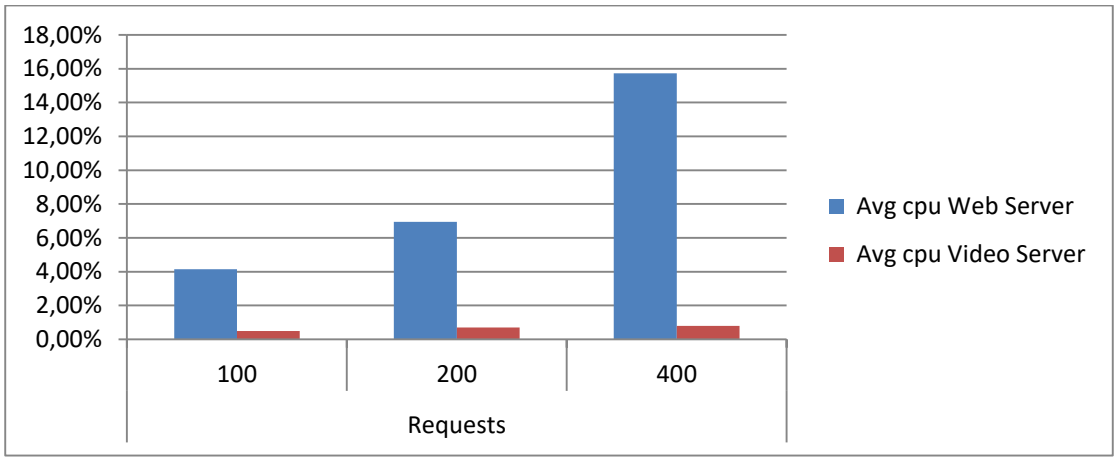
Web Service					
Requests	**Avg Load Time (msec)**	**Max cpu**	**Max memory**	**Average cpu**	**Average mem**
100	1604	20,21%	39,42%	3,80%	35,27%
200	2446	37,05%	40,52%	8,01%	36,00%
400	3820	48,69%	41,67%	15,73%	36,81%

Πίνακας 4.1 - Αποτελέσματα Web υπηρεσίας

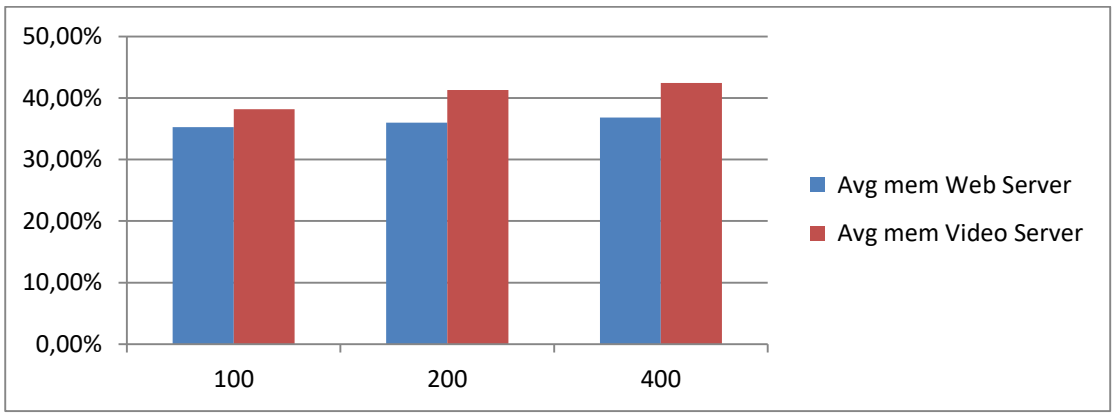
Video Service					
Requests	**Avg Load Time (msec)**	**Max cpu**	**Max memory**	**Average cpu**	**Average mem**
100	14196	0,80%	44,05%	0,49%	38,17%
200	15057	2,67%	48,12%	0,70%	41,28%
400	18667	1,32%	47,54%	0,79%	42,45%

Πίνακας 4.2 - Αποτελέσματα υπηρεσίας Video

Στα παρακάτω γραφήματα (4.2,4.3) παρουσιάζεται η μέση τιμή της χρήσης της μνήμης και του επεξεργαστή για κάθε υπηρεσία.



Εικόνα 4.2 - Μέσες τιμές χρήσης επεξεργαστή



Εικόνα 4.3 - Μέσες τιμές χρήσης μνήμης

4.1.2 Συμπεράσματα

Με μια προσεκτική μελέτη των παραπάνω αποτελεσμάτων (Πίνακας 4.1, Πίνακας 4.2) καθίσταται φανερό πως: για τον ίδιο αριθμό πελατών κάθε εφαρμογή έχει διαφορετική κατανάλωση επεξεργαστικών πόρων. Πιο συγκεκριμένα παρατηρούμε ότι η Web εφαρμογή κατανάλωσε πολύ μεγαλύτερη ποσότητα επεξεργαστικής ισχύος (η μέση τιμή έφτασε ως 15,73%) από την υπηρεσία Video Streaming (0,79%). Αντιθέτως η εφαρμογή video δέσμευσε περισσότερο χώρο στη μνήμη (47,54%) έναντι της Webυπηρεσίας (41,67%). Επίσης παρατηρούμε ότι όσο αυξάνεται ο αριθμός των πελατών αυξάνεται και η κατανάλωση των επεξεργαστικών πόρων των διακομιστών από τις εφαρμογές.

4.2 Σενάριο2- Αξιολόγηση απόδοσης αλγορίθμων εξισορρόπησης φορτίου για κάθε Διαδικτυακή Υπηρεσία

Στόχος του δεύτερου σεναρίου είναι η αξιολόγηση των αλγορίθμων εξισορρόπησης φορτιού που χρησιμοποιήσαμε. Πιο συγκεκριμένα σε αυτό το σενάριο θέλουμε να ερευνήσουμε για κάθε αλγόριθμο πώς επηρεάζονται οι επιδόσεις των εφαρμογών και κατά πόσο εξισορροπείται η κατανάλωση των επεξεργαστικών πόρων ανάμεσα στους διακομιστές.

Οι τρεις μηχανισμοί που χρησιμοποιήσαμε είναι κεντροποιημένοι (centralized), δηλαδή ένα λογισμικό που εκτελείται σε κεντρικό ελεγκτή συλλέγει πληροφορίες για το δίκτυο και εκχωρεί τις ροές ανάμεσα στους διακομιστές σύμφωνα με την πολιτική δικτύου που έχει οριστεί από τον διαχειριστή.

Η ρύθμιση των ροών στους μεταγωγείς και οι εκχώρηση των κανόνων για την προώθηση τους γίνεται με την reactive λειτουργία. Δηλαδή κάθε φορά που ένα νέο αίτημα εισέρχεται στο δίκτυο ο μεταγωγέας ενημερώνει τον ελεγκτή (μέσω του OpenFlow). Ο ελεγκτής σύμφωνα με την δικτυακή πολιτική που έχει οριστεί υπολογίζει την διαδρομή και εκχωρεί τους κανόνες στους μεταγωγείς.

Πείραμα 1: Αλγόριθμος κυκλικής επιλογής Simple Round Robin (Static Centralized Layer 4 Server Load balancing)

Αυτός ο αλγόριθμος εκχωρεί ένα προς ένα όλα τα αιτήματα των πελατών σε όλους τους διαθέσιμους διακομιστές. Είναι στατικός γιατί δεν λαμβάνει υπόψη του πληροφορίες που έχουν να κάνουν με την δικτυακή κίνηση η την κατάσταση των διακομιστών σε πραγματικό χρόνο. Επίσης είναι Layer 4 Server Load balancing

επειδή διαχειρίζεται τα αιτήματα των πελατών σύμφωνα με τις εικονική διεύθυνση και την θύρα που προορισμού του κάθε αιτήματος.

Πείραμα 2 : Statistics (Dynamic Centralized Layer 4 Server Load balancing)
Αυτός ο αλγόριθμος είναι δυναμικός, γιατί λαμβάνει πληροφορίες για την δικτυακή κίνηση και εκχωρεί τα αιτήματα των πελατών στον διακομιστή, του οποίου το κανάλι έχει την λιγότερη κίνηση (μικρότερο traffic). Επίσης είναι Layer 4 Server Load balancing επειδή διαχειρίζεται τα αιτήματα των πελατών συμφώνα με την εικονική διεύθυνση και την θύρα που προορισμού του κάθε αιτήματος.

Πείραμα 3 : Application Adaptive Load Balancing (Dynamic Centralized Layer 4 Server Load balancing)

Αυτός ο αλγόριθμος λαμβάνει υπόψη του πληροφορίες που έχουν να κάνουν με την δικτυακή κίνηση αλλά και τον επεξεργαστικό φόρτο των διακομιστών σε πραγματικό χρόνο.

Ο αλγόριθμος διατηρεί μια λίστα με τιμές για κάθε διακομιστή. Η κάθε τιμή είναι το άθροισμα του ποσοστού κατανάλωσης του επεξεργαστή, της μνήμης και της δικτυακής κίνησης (για κάθε διακομιστή). Ο διακομιστής που έχει την μικρότερη τιμή θεωρείται καλύτερος και όλα τα αιτήματα των πελατών εκχωρούνται προς αυτόν.

Από το πρώτο σενάριο συμπεραίνουμε ότι: αναλόγως με το είδος της εφαρμογής, για τον ίδιο αριθμό αιτημάτων, έχουμε διαφορετική κατανάλωση επεξεργαστή, μνήμης και δικτυακής κίνησης. Όποτε στον αλγόριθμο μας για να αθροίσουμε την τιμή των ποσοστών κατανάλωσης του κάθε διακομιστή ορίσαμε ένα βάρος για κάθε ποσοστό σύμφωνα με τις ανάγκες της εφαρμογής. Π.χ ο Web server παρατηρήσαμε ότι έχει πολύ μεγαλύτερη κατανάλωση επεξεργαστικής ισχύος από ότι ο video server. Οπότε για τα αιτήματα προς web server δίνεται περισσότερο βάρος στην κατανάλωση της επεξεργαστικής ισχύος κάθε διακομιστή.

4.2.1 Υλοποίηση Σεναρίου

Στην εφαρμογή αυτού του σεναρίου υλοποιούμε εξισορρόπηση φορτίου ανάμεσα σε τέσσερεις διακομιστές οι οποίοι παρέχουν τις δύο Διαδικτυακές υπηρεσίες (Web service , video service) ο καθένας. Κάθε διακομιστής έχει μια μεμονωμένη διεύθυνση IP (DIP). Επίσης για όλα τα πειράματα ορίσαμε μία εικονική

διεύθυνση (VIP) την 10.0.0.200 και όλα τα αιτήματα των πελατών δρομολογήθηκαν προς αυτήν. Οι αλγόριθμοι εξισορρόπησης φορτίου λαμβάνουν την κίνηση που προορίζεται για αυτήν την VIP και το εξισορροπούν το φόρτο ανάμεσα στους διακομιστές βάσει των αντίστοιχων διευθύνσεων τους (DIP). Η τοπολογία του δικτύου περιγράφεται στο κεφάλαιο «δημιουργία τοπολογίας». Η πόρτα που ακούει ο web server είναι η 80 και ο video server η 8080. Για όλα τα πειράματα χρησιμοποιήθηκε ο ελεγκτής Floodlight.

Η προσομοίωση των πελατών για την αξιολόγηση των εφαρμογών έγινε μέσω του JMeter. Σε αυτό το σενάριο για κάθε thread στο JMeter ορίσαμε να γίνεται ένα αίτημα προς την VIP ανάλογα με την υπηρεσία που εξετάζουμε (Εικόνα 4.3).

HTTP requests:

1. Web: http://10.0.0.200:80/medium

2. Video: http://10.0.0.200:8080/stream

Εικόνα 4.4 - Αιτήματα

Για 100 threads στο JMeter ορίσαμε να ξεκινάνε 2 threads κάθε 5 δευτερόλεπτα. Επίσης ορίσαμε το μέγιστο που θα περιμένει κάθε αίτημα για να ολοκληρωθεί η σύνδεση να είναι 35 δευτερόλεπτα. Άρα κάθε 5 δευτερόλεπτα θα γίνονται συνολικά 2 αιτήματα τα οποία θα διαμοιράζονται ανάμεσα στους διακομιστές. Για κάθε αλγόριθμο υλοποιήσαμε 3 πειράματα μέσω του JMeter για 100 threads (2 ανά 5 δευτερόλεπτα ,200 threads (4 ανά 5 δευτερόλεπτα) και 400 threads (7 ανά 5 δευτερόλεπτα). Τα αποτελέσματα της κατανάλωσης των πόρων επεξεργασίας για κάθε διακομιστή σε αυτό το σενάριο περιγράφονται στο Παράρτημα Γ - Αποτελέσματα

4.2.2 Πείραμα 1 Αλγόριθμος κυκλικής επιλογής (Simple Round Robin)

Στόχος του πειράματος είναι η αξιολόγηση της συγκεκριμένης μεθόδου και ο εντοπισμός των πλεονεκτημάτων και των μειονεκτημάτων της.

Ο συγκεκριμένος αλγόριθμος είναι προ εγκατεστημένος στο Floodlight controller. Για την εφαρμογή του στο πείραμα μας αρχικά ορίσαμε με την βοήθεια του REST API του Floodlight την εικονική διεύθυνση VIP με IP 10.0.0.200 και το pool με τις ακριβείς διευθύνσεις του κάθε διακομιστή. Οι εντολές που χρησιμοποιήσαμε παρουσιάζονται στο Παράρτημα Β – Simple Round Robin**.**

4.2.2.1 Αποτελέσματα

Στους παρακάτω πίνακες (4.3, 4.4) παρουσιάζεται ο μέσος (avg), ελάχιστος (min) και μέγιστος (max) χρόνος που έκανε το κάθε αίτημα για να ολοκληρωθεί καθώς και το ποσοστό των αιτημάτων που απέτυχαν να ολοκληρωθούν (error).

Web Server

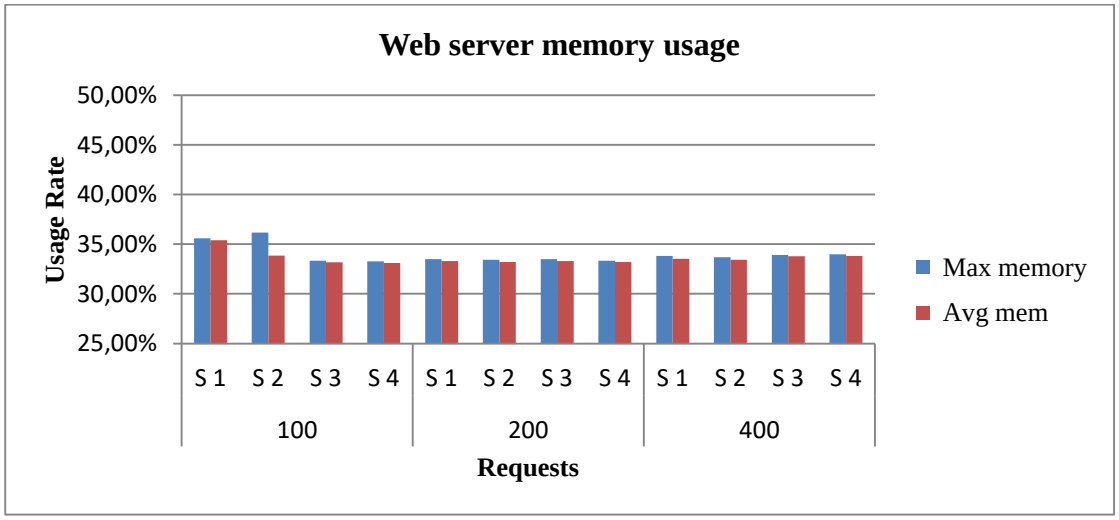
Web HTTP Request (Load test results)				
Requests	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
100	4136	4118	4213	0
200	4155	4122	4268	0
400	4209	4120	4361	0

Πίνακας 4.3 - Χρόνος ολοκλήρωσης αιτημάτων (web requests)

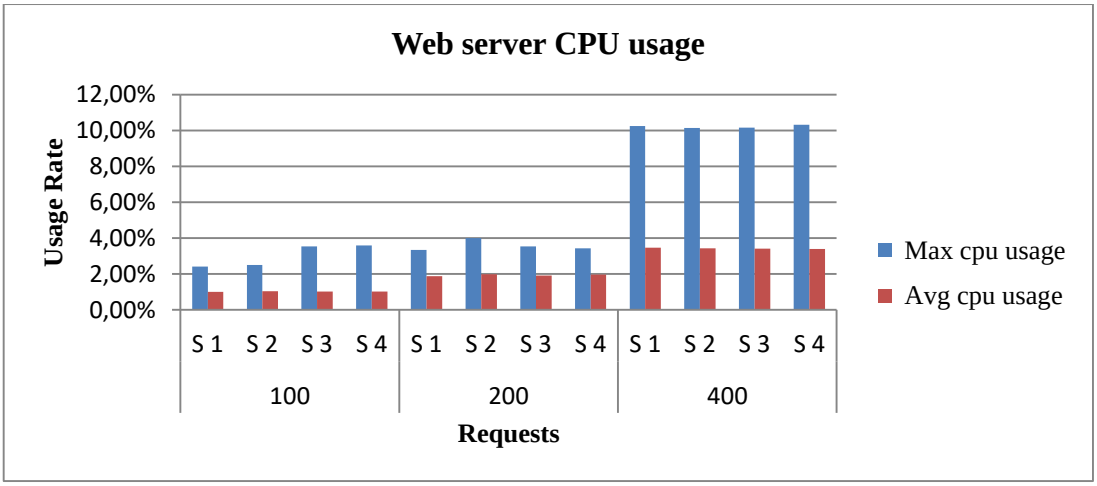
Video HTTP Request (Load test results)				
Requests	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
100	16707	1265	32111	0
200	16881	1181	31863	0
400	17621	2294	32305	0

Πίνακας 4.4 - Χρόνος ολοκλήρωσης αιτημάτων (video requests)

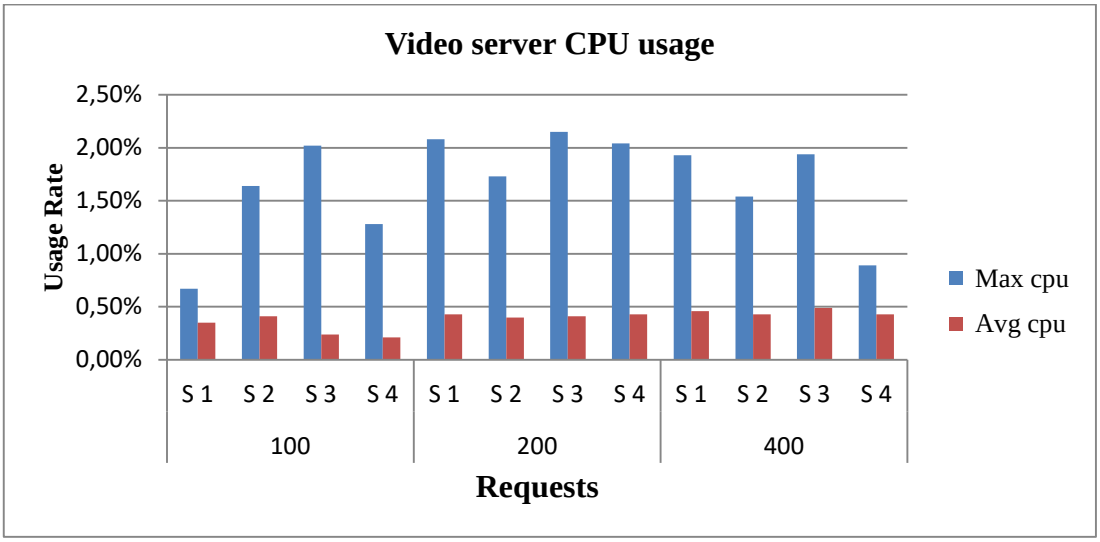
Στα γραφήματα (4.1,4.2,4.3,4.4) παρουσιάζονται τα αποτελέσματα της παρακολούθησης των διακομιστών.



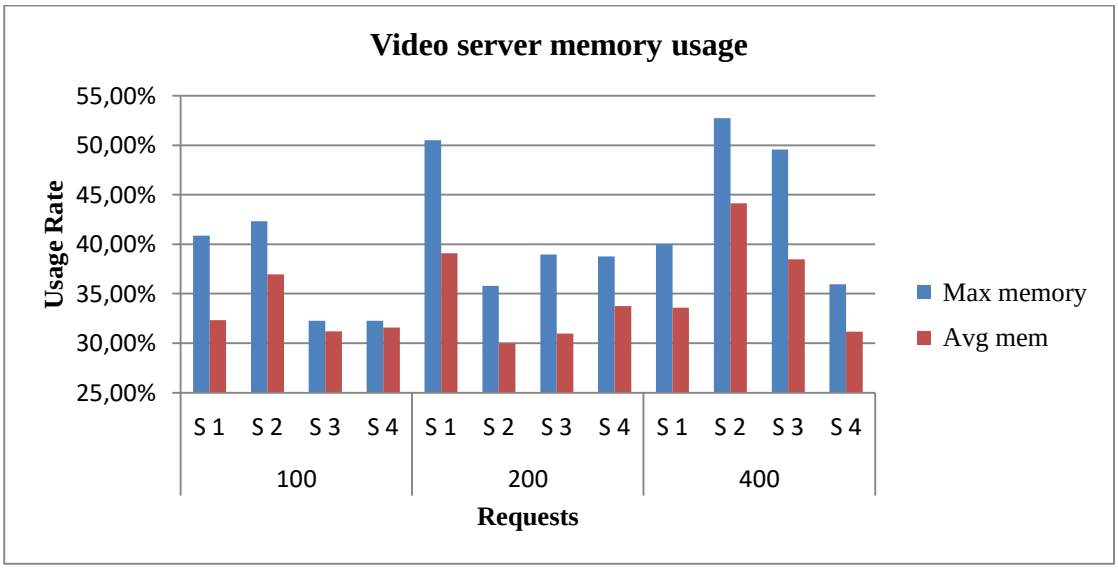
Γράφημα 4.1 - Δέσμευση μνήμης RAM (web service)



Γράφημα 4.2 - Κατανάλωση CPU (web service)



Γράφημα 4.3 - Κατανάλωση CPU (video service)



Γράφημα 4.4 - Δέσμευση μνήμης RAM (video service)

4.2.2.2 Συμπεράσματα

Σε αυτό το σημείο της εργασίας μας πρέπει να αναφέρουμε ότι για κάθε σύνδεσμο που υπάρχει σε μια ιστοσελίδα δημιουργείται διαφορετική TCP σύνδεση. Ο αλγόριθμος που χρησιμοποιούμε αντιλαμβάνεται την κάθε σύνδεση σαν διαφορετικό αίτημα. Π.χ για 43 συνδέσμους που υπάρχουν στην ιστοσελίδα μεσαίου μεγέθους που χρησιμοποιούμε στο πείραμα μας θα γίνουν 43 TCP συνδέσεις. Στην περίπτωση αυτή για τον συγκεκριμένο αλγόριθμο, ο πρώτος σύνδεσμος θα εξυπηρετηθεί από τον πρώτο διακομιστή και ο τελευταίος (43) από τον τρίτο διακομιστή (43mod4) .

Στην περίπτωση του Video όμως η διάρκεια της κάθε σύνδεσης εξαρτάται από την χρονική στιγμή που γίνεται το αίτημα και ξεκινάει η σύνδεση (session). Πιο συγκεκριμένα ο webserver κάνει ζωντανή μετάδοση (live streaming) ενός βίντεο 31 δευτερολέπτων και κάθε φορά που τελειώνει η μετάδοση το επαναμεταδίδει από την αρχή. Αν για παράδειγμα ένα αίτημα γίνει την χρονική στιγμή που ο server μεταδίδει το 10_o δευτερόλεπτο του video η σύνδεση θα κρατήσει 21 δευτερόλεπτα. Προσεγγίσαμε το συγκεκριμένο θέμα με αυτό τον τρόπο γιατί θέλουμε να εξετάσουμε την περίπτωση που τα αιτήματα των πελατών απευθύνονται σε video διαφορετικής διάρκειας.

Παρατηρώντας τα παραπάνω αποτελέσματα συμπεραίνουμε ότι ο συγκεκριμένος αλγόριθμος (SRR) εξισορροπεί πολύ καλύτερα την υπηρεσία παροχής ιστοσελίδων από ότι την υπηρεσία μετάδοσης βίντεο. Αυτό συμβαίνει γιατί το μέγεθος των ροών των ιστοσελίδων είναι πολύ μικρότερο και σταθερού μεγέθους από ότι του βίντεο, οπότε και η κάθε σύνδεση διαρκεί λιγότερο χρόνο. Στην υπηρεσία μετάδοσης βίντεο όμως που τα αιτήματα δεν έχουν σταθερό μέγεθος και διάρκεια ζωής, παρατηρούμε ότι ο συγκεκριμένος αλγόριθμος αδυνατεί να εξισορροπήσει τον φόρτο σε επιθυμητά επίπεδα. Επειδή ο αλγόριθμος είναι στατικός διαμοιράζει όλα τα αιτήματα με τον ίδιο τρόπο, συνεπώς αιτήματα μεγάλου μεγέθους και διάρκειας να συσσωρεύονται σε έναν διακομιστή.

Για την καλύτερη ανάλυση των αποτελεσμάτων υπολογίσαμε το εύρος ανάμεσα στα ποσοστά της επεξεργαστικής κατανάλωσης των διακομιστών (Πίνακας 4.5). Από ότι παρατηρούμε το εύρος ανάμεσα στα ποσοστά της δέσμευσης μνήμης για την βίντεο υπηρεσία έφτασε το 12,99% .

Requests	Web Service		Video Service	
	Range avg cpu	Range avg mem	Range avg cpu	Range avg mem
100	0,04%	2,27%	0,20%	5,73%
200	0,10%	0,11%	0,03%	9,13%
400	0,08%	0,37%	0,06%	12,99%

Πίνακας 4.5 – Εύρος τιμών

4.2.3 Πείραμα 2 Αλγόριθμος βασισμένος σε στατιστικά (Statistics)

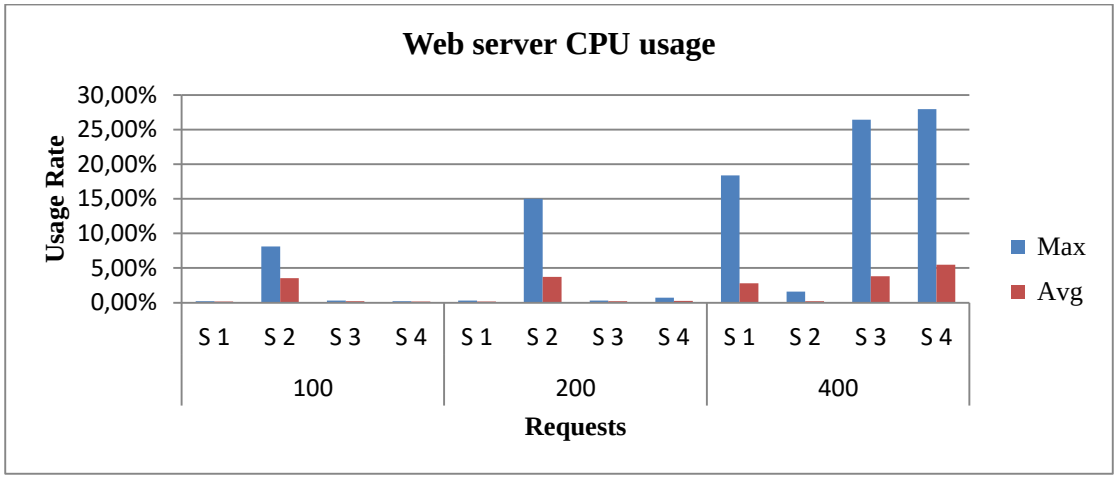
Αυτός ο αλγόριθμος αντίθετα με τον προηγούμενο είναι δυναμικός γιατί λαμβάνει πληροφορίες για την δικτυακή κίνηση και εκχωρεί τα αιτήματα των πελατών στον διακομιστή του οποίου οσύνδεσμος (link)έχει την λιγότερη κίνηση (μικρότερο traffic). Είναι και αυτός επίσης προεγκατεστημένος στον Floodlight controller. Χρησιμοποιώντας το το REST API του Floodlight ορίσαμε την εικονική διεύθυνση VIP με IP 10.0.0.200 και το pool με τις ακριβείς διευθύνσεις του κάθε διακομιστή. Οι εντολές που χρησιμοποιήσαμε παρουσιάζονται στο Παράρτημα Β - Statistics. Ο συγκεκριμένος αλγόριθμος αν και είναι δυναμικός, είναι σχετικά απλός . Δημιουργεί μια λίστα με όλα τα μέλη του pool και στην συνέχεια την ενημερώνει, καταχωρώντας την κίνηση του δικτύου που έχει καταγράψει ο μεταγωγέας στην θύρα με την οποία συνδέεται ο εκάστοτε διακομιστής. Έχουμε ορίσει τον ελεγκτή να ελέγχει την κίνηση του δικτύου κάθε πέντε δευτερόλεπτα (προεπιλογή Floodlight 10 δευτερόλεπτα).

4.2.3.1 Αποτελέσματα

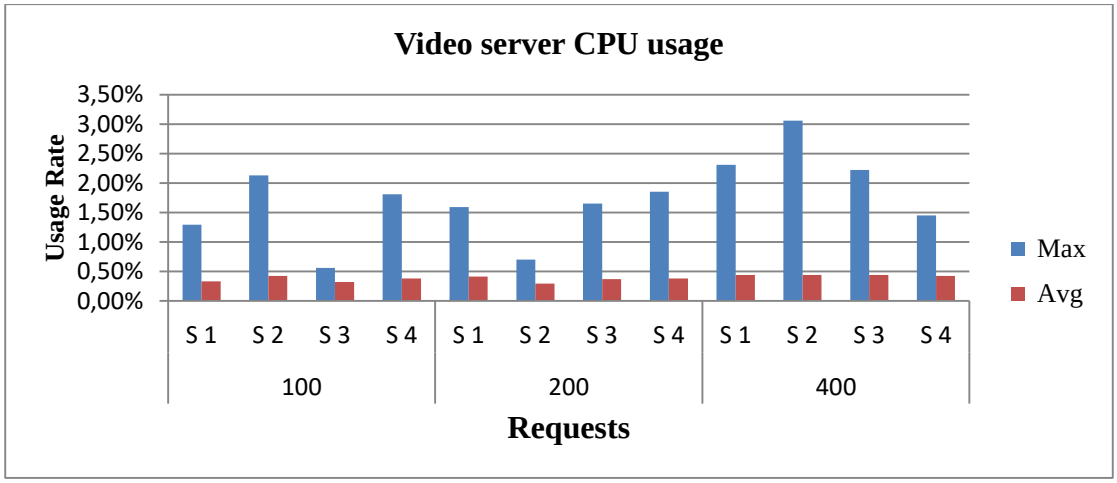
Web HTTP Request				
Requests	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
100	4191	4123	4283	0
200	4321	4123	4500	0
400	5018	4601	5403	0

Video HTTP Request				
Requests	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
100	15243	1262	31273	0
200	16384	1111	31723	0
400	16892	1331	31793	0

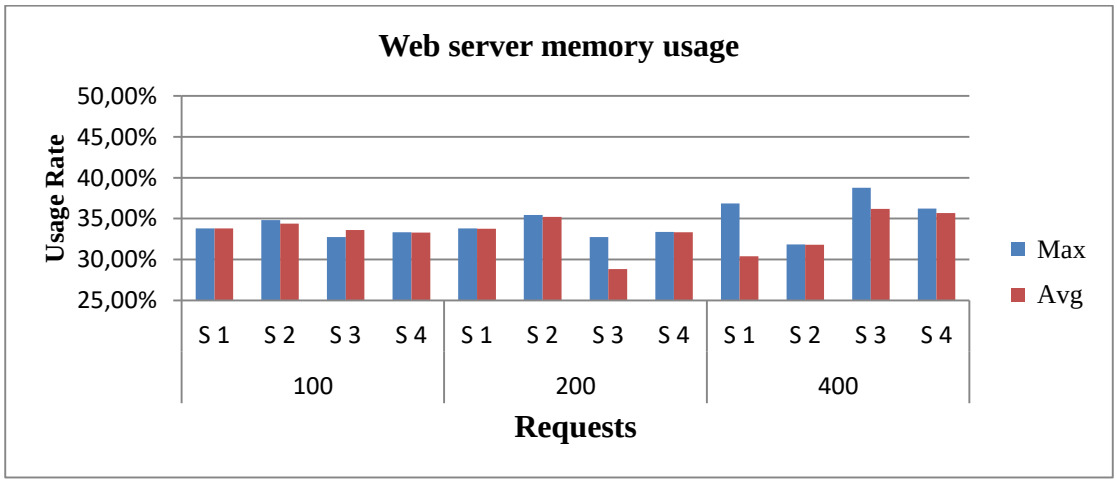
Πίνακας 4.6 - Χρόνος ολοκλήρωσης αιτημάτων



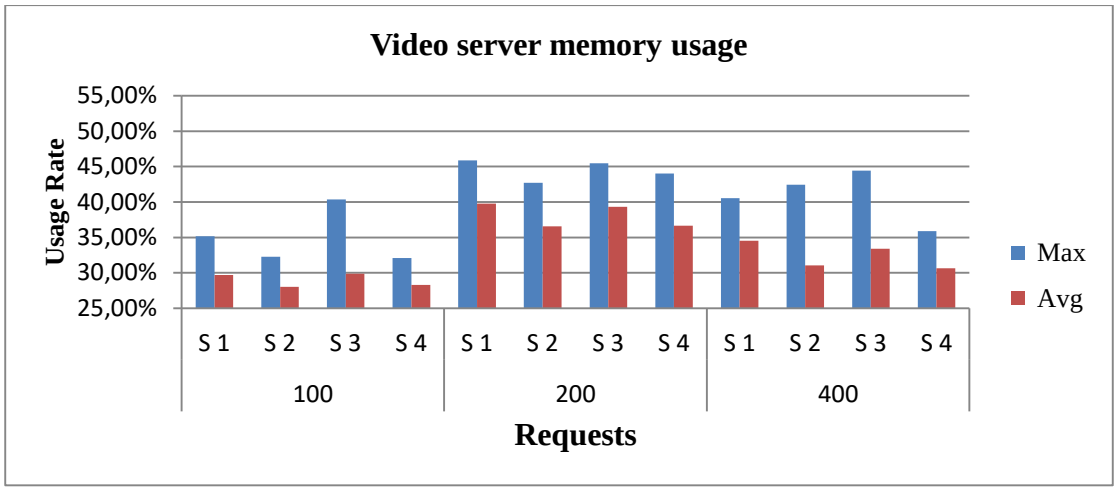
Γράφημα 4.5 - Κατανάλωση CPU (web service)



Γράφημα 4.6 - Κατανάλωση CPU (video service)



Γράφημα 4.7 - Δέσμευση μνήμης RAM(web service)



Γράφημα 4.8 - Δέσμευση μνήμης RAM(videoservice)

Web Service			Video Service		
Threads	**Range avg cpu**	**Range avg mem**	**Threads**	**Range avg cpu**	**Range avg mem**
100	3,37%	1,10%	100	0,06%	1,87%
200	3,56%	6,40%	200	0,10%	3,18%
400	5,24%	5,80%	300	0,08%	3,51%

4.2.3.2 Συμπεράσματα

Σύμφωνα με τα αποτελέσματα αυτού του σεναρίου για την υπηρεσία παροχής ιστοσελίδων παρατηρούμε ότι για τα πειράματα των 100 και 200 αιτημάτων, ο αλγόριθμος δεν εξισορρόπησε το φορτίο (Γράφημα 4.5). Αυτό συμβαίνει γιατί ένα αίτημα για να ολοκληρωθεί κάνει περίπου τέσσερα δευτερόλεπτα ενώ ο αλγόριθμος ενημερώνεται για τη δικτυακή κίνηση του δικτύου κάθε 5 δευτερόλεπτα. Όποτε για κάθε νέο γύρο αιτημάτων ο αλγόριθμος έβλεπε ότι η δικτυακή κίνηση είναι ίδια προς όλους τους διακομιστές με αποτέλεσμα να στείλει όλα τα αιτήματα σε ένα διακομιστή. Επίσης παρατηρούμε ότι ακόμα και στο πείραμα των 400 αιτημάτων που η ολοκλήρωση των αιτημάτων ξεπέρασε τα 5 δευτερόλεπτα είχαμε εξισορρόπηση φορτίου αλλά όχι ανάμεσα σε όλους τους διακομιστές.

Όσον αφορά την υπηρεσία βίντεο, παρατηρούμε ότι αυτός ο αλγόριθμος εξισορρόπησε πολύ καλύτερα τα αίτημα από τον προηγούμενο (SRR). Επειδή ο αλγόριθμος είναι δυναμικός δεν διαμοιράζει όλα τα αιτήματα συνεχώς με τον ίδιο τρόπο. Ο αλγόριθμος παρακολουθεί τη δικτυακή κίνηση και προωθεί τα αιτήματα στο

διακομιστή με το μικρότερο φορτίο, συνεπώς δεν υπάρχει ο κίνδυνος αιτήματα μεγάλης διάρκειας ζωής να σωρευτούν σε ένα διακομιστή.

4.2.4 Πείραμα 3 Αλγόριθμος προσαρμοσμένος στο είδος της εφαρμογής (AALB)

Ο τρίτος αλγόριθμος τον οποίο θα αξιολογήσουμε είναι ο ‘’Αλγόριθμος εξισορρόπησης φορτίου προσαρμοσμένος στο είδος της εφαρμογής’’ (Application Adaptive Load Balancing). Ο τρόπος λειτουργίας του και η υλοποίηση του περιγράφεται στο αντίστοιχο υποκεφάλαιο της πειραματικής προσέγγισης (Κεφάλαιο 2).

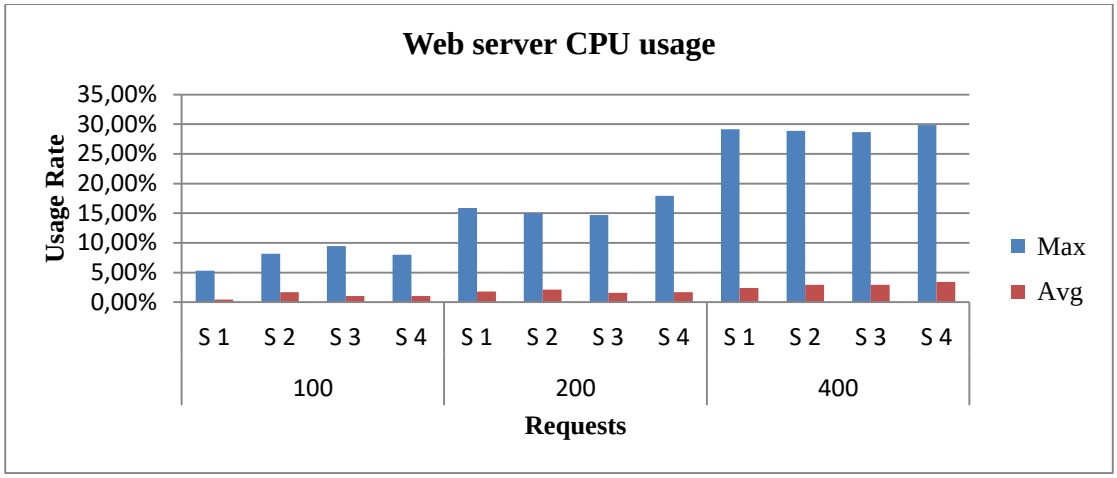
Για τον υπολογισμό των τιμών της μέτρησης του φόρτου στο συγκεκριμένο πείραμα ορίσαμε τα βάρη ως εξής: για την Web εφαρμογή 50% στο bandwidth, 40% στην κατανάλωση του επεξεργαστή και 10% για την μνήμη. Για την Video streaming εφαρμογή 50% στο bandwidth, 10% στον επεξεργαστή και 40% στη μνήμη.. Η τιμή του κάθε βάρους ορίζεται αφού πρώτα ερευνήσουμε για κάθε υπηρεσία το πως επηρεάζονται οι επεξεργαστικοί πόροι των διακομιστών (Σενάριο 1).

4.2.4.1 Αποτελέσματα:

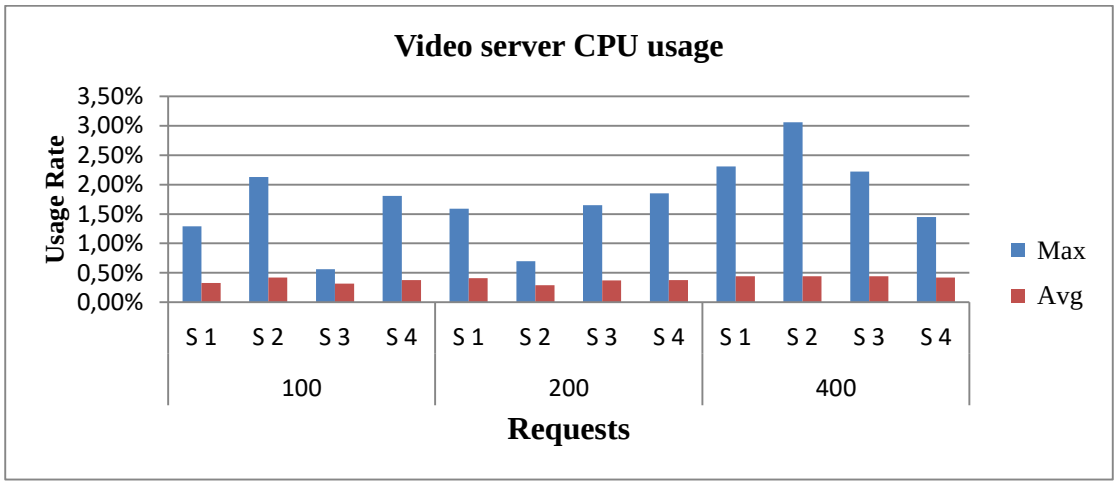
Web HTTP Request				
Threads	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
100	4198	4118	4336	0
200	4331	4140	4541	0
400	4780	4321	5271	0

Video HTTP Request				
Threads	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
100	16060	1062	31374	0
200	17058	1261	31833	0
400	16892	1331	31793	0

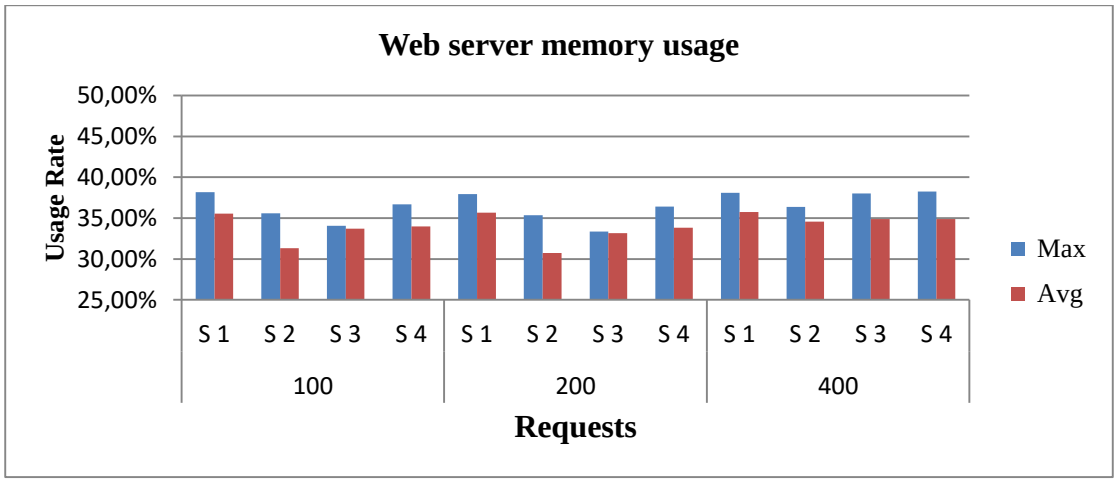
Πίνακας 4.7 - Χρόνος ολοκλήρωσης αιτημάτων



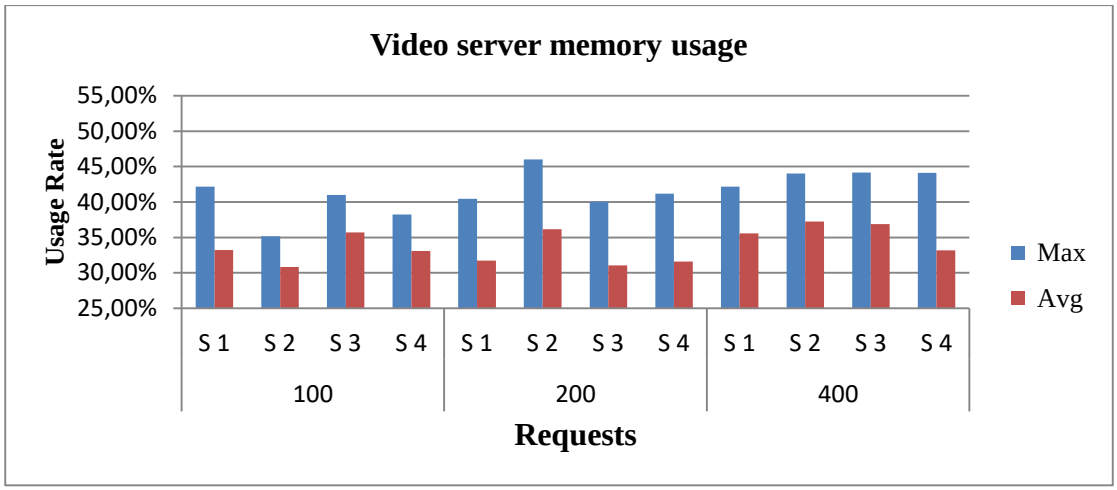
Γράφημα 4.9 - Κατανάλωση CPU (web service)



Γράφημα 4.10 - Κατανάλωση CPU (video service)



Γράφημα 4.11 - Δέσμευση μνήμης RAM (web service)



Γράφημα 4.12 - Δέσμευση μνήμης RAM (video service)

Requests	Web Service		Video Service	
	Range avg cpu	Range avg mem	Range avg cpu	Range avg mem
100	1,23%	4,24%	0,10%	4,86%
200	0,58%	4,92%	0,12%	5,09%
300	1,01%	1,17%	0,02%	4,07%

Πίνακας 4.8 - Εύρος τιμών

4.2.4.2 Συμπεράσματα

Σε αυτό το σενάριο αξιολογήσαμε τους αλγόριθμους εξισορρόπησης φορτιού για κάθε εφαρμογή ξεχωριστά. Δηλαδή σε κάθε πείραμα όλα τα αιτήματα ήταν προς μία υπηρεσία. Σύμφωνα με τα αποτελέσματα, παρατηρούμε ότι αυτός ο αλγόριθμος έχει σχεδόν την ίδια απόδοση με τον προηγούμενο (statistics). Επειδή ο αλγόριθμος είναι προσαρμοσμένος στο είδος των εφαρμογών, η αξιολόγηση του σε σύγκριση με τους άλλους πρέπει να γίνει στην περίπτωση που ο φόρτος των αιτημάτων απευθύνεται προς διαφορετικού είδους υπηρεσίες. Επίσης παρατηρούμε ότι ο αλγόριθμος έχει καλύτερη απόδοση στην εξισορρόπηση των αιτημάτων προς την υπηρεσία παροχής ιστοσελίδων. Αυτό συμβαίνει γιατί ο αλγόριθμος ενημερώνεται για την χρήση των πόρων των διακομιστών πιο σύντομα (κάθε δευτερόλεπτο) από ότι ο statistics για την δικτυακή κυκλοφορία (κάθε 5 δευτερόλεπτα).

4.2.5 Συγκριτική αξιολόγηση αλγορίθμων

Ο στόχος τον αλγόριθμων εξισορρόπησης φορτίου είναι η βελτίωση της απόδοσης των εφαρμογών και η εξισορρόπηση του φόρτου των διακομιστών στο δικτυακό περιβάλλον που δημιουργήσαμε.

4.2.5.1 Σύγκριση των αλγορίθμων ως προς την απόδοση των εφαρμογών

Για να συγκρίνουμε την απόδοση των αλγορίθμων από την πλευρά των πελατών για την webυπηρεσία συγκρίναμε την ταχύτητα φόρτωσης της ιστοσελίδας που ζητήθηκε (Πίνακας 4.9). Από ότι παρατηρούμε και για τα τρία πειράματα (για 100, 200 και 400 αιτήματα) ο αλγόριθμος κυκλικής επιλογής (Simple Round Robin) είχε την καλύτερη απόδοση.

Requests	**Average Page Load Time (msec)**		
	SRR	**STATISTICS**	**AALB**
100	4136	4191	4198
200	4155	4321	4331
400	4209	5018	4780

Πίνακας 4.9 - Σύγκριση Αλγορίθμων

Η υπηρεσία παροχής βίντεο έχει σταθερό ρυθμό μετάδοσης γιατί γίνεται σε πραγματικό χρόνο (live streaming). Επίσης η διάρκεια του κάθε αιτήματος εξαρτάται από την χρονική στιγμή που γίνεται το κάθε αίτημα και ξεκινάει η TCP σύνδεση (session). Πιο συγκεκριμένα, έχουμε ρυθμίσει την βίντεο υπηρεσία να μεταδίδει ένα βίντεο 31 δευτερολέπτων και κάθε φορά που τελειώνει το βίντεο το αναμεταδίδει από την αρχή. Αν ένα αίτημα γίνει τη χρονική στιγμή μου ο διακομιστής μεταδίδει το δέκατο δευτερόλεπτο του video η σύνδεση θα κρατήσει 21 δευτερόλεπτα. Οπότε η σύγκριση των αλγορίθμων ως προς το χρόνο ολοκλήρωσης των αιτημάτων δεν είναι αντιπροσωπευτική. Στόχος της σωστής εξισορρόπησης για τα αιτήματα των βίντεο είναι να μην υπάρχουν απώλειες και επαναμεταδόσεις (retransmission) πακέτων.

Στην περίπτωση που η TCP σύνδεση χαθεί μετά από ένα αίτημα, το Jmeter θεωρεί ότι αυτό απέτυχε και δεν προσπαθεί να επανασυνδεθεί. Το ποσοστό το αποτυχημένων αιτημάτων για όλα τα πειράματα ήταν μηδέν οπότε θεωρούμε ότι σε όλα τα αιτήματα το βίντεο μεταδόθηκε με τον επιθυμητό τρόπο.

4.2.5.2 Σύγκριση της απόδοσης των αλγορίθμων ως προς την εξισορρόπηση του φόρτου των διακομιστών.

Στόχος των αλγορίθμων εξισορρόπησης φορτίου από την πλευρά των διακομιστών είναι η βελτίωση του τρόπου διαμοιρασμού της δικτυακής κυκλοφορίας για την μείωση του φόρτου εργασίας των διακομιστών. Από το πρώτο σενάριο παρατηρούμε ότι στην υπηρεσία παροχής ιστοσελίδων η χρήση του επεξεργαστή ήταν πολύ μεγαλύτερη(η μέση τιμή έφτασε ως 15,73%) από την υπηρεσία παροχής βίντεο (0,79%). Αντιθέτως η υπηρεσία βίντεο δέσμευσε περισσότερο χώρο στην μνήμη. Οπότε για τη σύγκριση της απόδοσης των αλγορίθμων για την υπηρεσία παροχής ιστοσελίδων (web service) συγκρίναμε το εύρος των μέσων τιμών της χρήσης του επεξεργαστή (Πίνακας 4.10) και για την υπηρεσία βίντεο συγκρίναμε το εύρος των μέσων τιμών της δέσμευσης μνήμης (RAM) **(**Πίνακας 4.11).

Web server Avg CPU usage			
Requests	**SRR**	**Statistics**	**AALB**
100	0,04%	3,37%	1,23%
200	0,10%	3,56%	0,58%
400	0,08%	5,24%	1,01%

Web server Avg memory usage			
Requests	**SRR**	**Statistics**	**AALB**
100	5,73%	1,87%	4,86%
200	9,13%	3,18%	5,09%
400	12,99%	3,51%	4,07%

Πίνακας 4.10- Κατανάλωση CPU **Πίνακας 4.11 -Δέσμευση μνήμης RAM**

Παρατηρώντας τα παραπάνω αποτελέσματα συμπεραίνουμε ότι ο αλγόριθμος κυκλικής επιλογής (Simple Round Robin) εξισορροπεί πολύ καλύτερα το φόρτο για την υπηρεσία παροχής ιστοσελίδων. Αυτό συμβαίνει γιατί το μέγεθος των ροών των ιστοσελίδων είναι πολύ μικρό και σταθερό για όλα τα αιτήματα. Στην υπηρεσία των βίντεο όμως που τα αιτήματα δεν έχουν σταθερό μέγεθος και διάρκεια ζωής παρατηρούμε ότι ο συγκεκριμένος αλγόριθμος αδυνατεί να εξισορροπήσει τον φόρτο σε επιθυμητά επίπεδα. Επειδή ο αλγόριθμος είναι στατικός διαμοιράζει όλα τα αιτήματα με τον ίδιο τρόπο, συνεπώς αιτήματα μεγάλου μεγέθους και διάρκειας να σωρεύονται σε έναν διακομιστή.

Οι δυναμικοί αλγόριθμοι (AALB, Statistics)παρατηρούμε ότι εξισορρόπησαν πολύ καλύτερα τον φόρτο των αιτημάτων για τη βίντεο υπηρεσία ανάμεσα στους διακομιστές. Επειδή οι αλγόριθμοι είναι δυναμικοί δεν διαμοιράζουν όλα τα αιτήματα συνεχώς με τον ίδιο τρόπο. Ο αλγόριθμοι παρακολουθούν την δικτυακή κίνηση και προωθούν τα αιτήματα στον διακομιστή με τον μικρότερο φόρτο, συνεπώς δεν υπάρχει ο κίνδυνος αιτήματα μεγάλης διάρκειας ζωής να σωρευτούν σε έναν διακομιστή.

Επίσης παρατηρούμε ότι ο αλγόριθμος AALB έχει καλύτερη απόδοση στην εξισορρόπηση των αιτημάτων προς την υπηρεσία παροχής ιστοσελίδων από τον statistics. Αυτό συμβαίνει γιατί ο αλγόριθμος ενημερώνεται για τη χρήση των επεξεργαστικών πόρων των διακομιστών πιο σύντομα(κάθε δευτερόλεπτο), από ότι ο statisticsγια την δικτυακή κυκλοφορία (κάθε 5 δευτερόλεπτα).

4.3 Σενάριο 3 - Αξιολόγηση των αλγορίθμων εξισορροπώντας τα αιτήματα για δύο εφαρμογές ταυτόχρονα

Στο δεύτερο πειραματικό σενάριο αξιολογήσαμε τους αλγόριθμους εξισορρόπησης φορτιού για κάθε εφαρμογή ξεχωριστά. Στόχος του τρίτου σεναρίου είναι αναλύσουμε τους αλγορίθμους συνδυαστικά εξισορροπώντας τα αιτήματα και για τις δύο εφαρμογές ταυτόχρονα. Πιο συγκεκριμένα σε αυτό το σενάριο θέλουμε να ερευνήσουμε ποιος συνδυασμός αλγορίθμων (και για τις δυο εφαρμογές) έχει την καλύτερη απόδοση ως προς την εξυπηρέτηση των αιτημάτων και κατά πόσο εξισορροπείται η κατανάλωση των επεξεργαστικών πόρων ανάμεσα στους διακομιστές.

4.3.1 Υλοποίηση

Για την υλοποίηση και αυτού του σεναρίου χρησιμοποιήθηκε ο Floodlight controller. Ο Floodlight υποστηρίζει μόνο έναν αλγόριθμο για κάθε ομάδα διακομιστών (pool members). Για την εκτέλεση των πειραμάτων αυτού του σεναρίου χρειάστηκε να παρέμβουμε στον κώδικα του ελεγκτή ώστε να τον προσαρμόσουμε έτσι ώστε για τους ιδίους διακομιστές να υποστηρίζονται διαφορετικοί αλγόριθμοι εξισορρόπησης φορτίου. Η αλλαγές που πραγματοποιήθηκαν στον κώδικα παρουσιάζονται στο Παράρτημα Α - Αποσφαλμάτωση.

Οι αλγόριθμοι που θα χρησιμοποιήσαμε αναλύθηκαν στο δεύτερο σενάριο. Περιγραφικά οι αλγόριθμοι αυτοί είναι:

- Αλγόριθμος κυκλικής κατανομής – **Simple Round Robin**
- Αλγόριθμος που βασίζεται σε στατιστικά (της μέτρησης της δικτυακής κίνησης) - **Statistics**
- Αλγόριθμος που βασίζεται στο είδος της εφαρμογής – **Application Aware Load Balancing**

Τα πειράματα με τους συνδυασμούς των αλγορίθμων εξισορρόπησης φορτίου για την κάθε εφαρμογή παρουσιάζονται στο πίνακα παρακάτω:

Πείραμα	Υπειρεσία	Αλγόριθμος
1	Web Service	SRR
	Video Service	SRR
2	Web Service	SRR
	Video Service	Statistics
3	Web Service	Statistics
	Video Service	Statistics
4	Web Service	AALB
	Video Service	AALB
5	Web Service	AALB
	Video Service	Statistics
6	Web Service	SRR
	Video Service	AALB

Πίνακας 4.12 - Περιγραφή πειραμάτων

Για κάθε thread στο JMeter ορίσαμε να γίνονται 2 αιτήματα. Το πρώτο αίτημα γίνεται προς την υπηρεσία ιστοσελίδων (Web Service) στην VIP 10.0.0.100. Το δεύτερο ερώτημα προς την Video streaming υπηρεσία στην VIP 10.0.0.200..

Σε σύγκριση με το προηγούμενο σενάριο τα αιτήματα είναι διπλάσια γιατί απευθύνονται και στις 2 εφαρμογές ταυτόχρονα. Λόγω της πολυπλοκότητας του πειράματος ο υπολογιστής που χρησιμοποιήθηκε για την εκτέλεση των πειραμάτων δεν κατάφερε να ανταποκριθεί στο πείραμα με 400 threads σε αυτό το σενάριο. Οπότε για κάθε πείραμα υλοποιήσαμε 3 πειράματα μέσω του jmeter για 50 threads (1 ανά 5 δευτερόλεπτα), 100 threads (2 ανά 5 δευτερόλεπτα και 200 threads (4 ανά 5 δευτερόλεπτα).

4.3.2 Πείραμα 1 SRR-SRR

Ο αλγόριθμος εξισορρόπησης κυκλικής επιλογής είναι ο πιο απλός στη λειτουργία του και στην υλοποίηση του. Είναι ένας απλός στατικός αλγόριθμος ο οποίος διαμοιράζει τα αιτήματα ένα προς ένα σε όλους τους διακομιστές. Σε αυτό το πείραμα χρησιμοποιήθηκε ο Simple Round Robin αλγόριθμος και για τις δύο εφαρμογές (web service ,video service).

Αποτελέσματα:

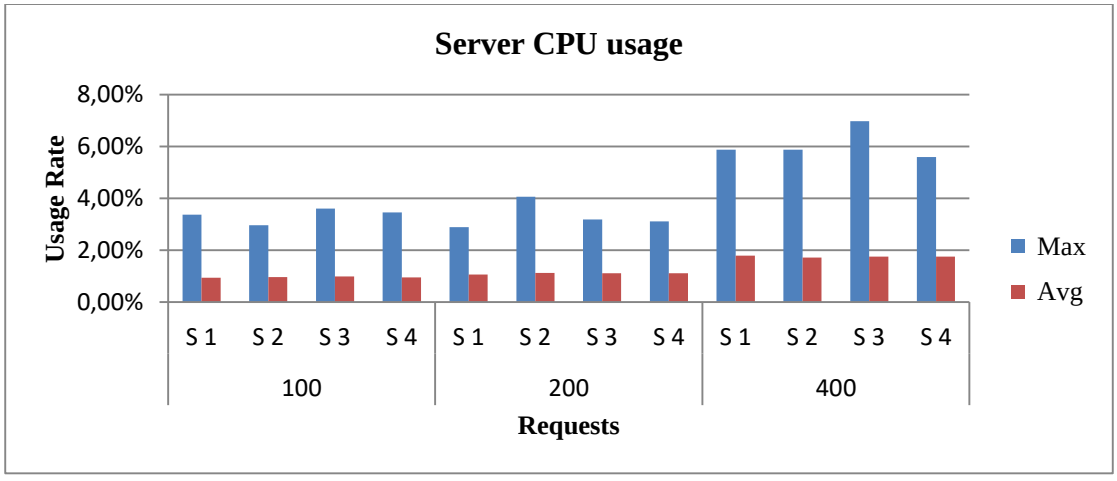
Web HTTP Request (SRR)				
Threads	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
50	4151	4118	4501	0
100	4195	4125	4356	0
200	4202	4118	4517	0

Video HTTP Request (SRR)				
Requests	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
50	16080	1517	31256	0
100	16149	1221	30908	0
200	14929	1187	31442	0

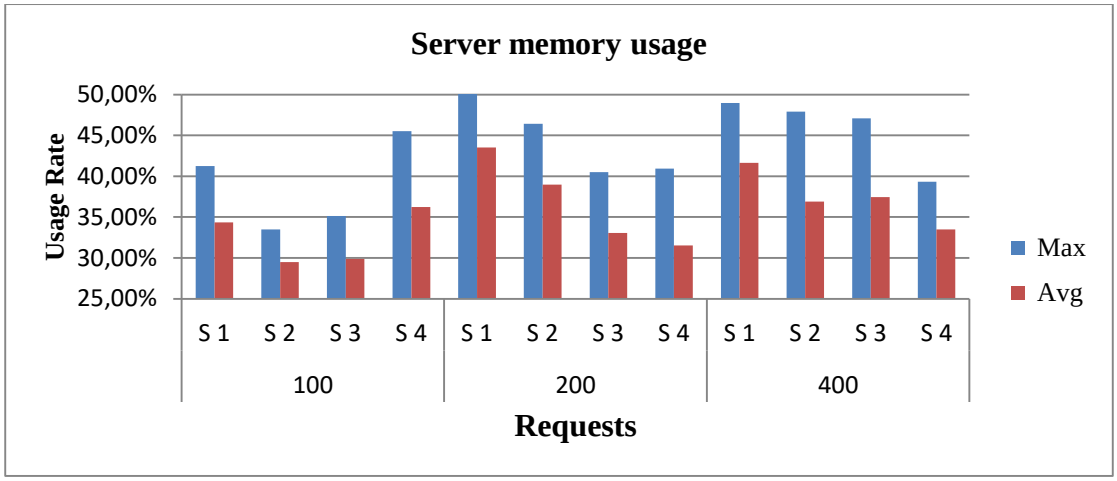
Πίνακας 4.13 - Απόδοση εφαρμογών

Video -SRR / Web -SRR (Server CPU/Mem usage)					
Requests	**Server Name**	**Max cpu**	**Max memory**	**Avg cpu usage**	**Avg mem usage**
50	mn.d1	3,37%	41,24%	0,94%	34,34%
	mn.d2	2,96%	33,51%	0,97%	29,51%
	mn.d3	3,60%	35,12%	0,99%	29,89%
	mn.d4	3,46%	45,51%	0,95%	36,24%
100	mn.d1	2,89%	53,66%	1,06%	43,52%
	mn.d2	4,06%	46,41%	1,13%	38,96%
	mn.d3	3,18%	40,52%	1,12%	33,05%
	mn.d4	3,11%	40,95%	1,11%	31,54%
200	mn.d1	5,87%	48,96%	1,79%	41,64%
	mn.d2	5,88%	47,91%	1,72%	36,91%
	mn.d3	6,97%	47,08%	1,76%	37,44%
	mn.d4	5,59%	39,32%	1,75%	33,48%

Πίνακας 4.14 - Αποτελέσματα παρακολούθησης διακομιστών



Γράφημα 4.13 - Ποσοστά χρήσης επεξεργαστή



Γράφημα 4.14 - Δέσμευση μνήμης RAM

Range		
Threads	Range avg cpu	Range avg mem
50	0,05%	6,73%
100	0,07%	11,98%
200	0,07%	8,16%

Πίνακας 4.15 - Εύρος μέσων τιμών CPU/mem

Συμπεράσματα:

Σύμφωνα με τα αποτελέσματα του εύρους των μέσων τιμών (Πίνακας 4.15) παρατηρούμε ότι η χρήση του επεξεργαστή εξισορροπήθηκε σε καλό βαθμό σε όλους τους διακομιστές, η τιμή δεν πέρασε το 0,07. Το εύρος ανάμεσα στα ποσοστά της δέσμευσης μνήμης όμως έφτασε το 11,98%. Αυτό συμβαίνει γιατί (όπως προείπαμε στην ανάλυση του προηγούμενου σεναρίου) η βίντεο υπηρεσία δεσμεύει μεγάλο μέρος της μνήμης, επίσης ο αλγόριθμος SRR διαμοιράζει όλα τα αιτήματα με τον ίδιο τρόπο, συνεπώς αιτήματα μεγάλου μεγέθους και διάρκειας να σωρεύονται σε ένα διακομιστή.

4.3.3 Πείραμα 2 SRR-Statistics

Στο πρώτο πείραμα παρατηρήσαμε την ανάγκη για δυναμικούς αλγορίθμους εξισορρόπησης φορτίου σε υπηρεσίες που παράγουν μεγάλες ροές. Για την επίλυση αυτού του προβλήματος, σε αυτό το πείραμα χρησιμοποιήθηκε ο αλγόριθμος Simple Round Robin για την υπηρεσία web και ο statistics για την υπηρεσία video streaming. Τα αποτελέσματα ακολουθούν.

Web HTTP Request (SRR)				
Threads	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
50	4158	4119	4432	0
100	4283	4119	5212	0
200	4583	4121	6269	0

Πίνακας 4.16 - Απόδοση υπηρεσίας web

Video HTTP Request (Statistics)				
Threads	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
50	16164	1554	31356	0
100	14005	1364	31354	0
200	15232	2169	31385	0

Πίνακας 4.17 - Απόδοση υπηρεσίας video streaming

Video -SRR / Web -Statistics (Server CPU/Mem usage)					
Threads	**Server Name**	**Max cpu**	**Max memory**	**Avg cpu usage**	**Avg mem usage**
50	mn.d1	3,69%	36,36%	0,88%	31,57%
	mn.d2	4,10%	40,98%	0,81%	33,77%
	mn.d3	2,80%	37,44%	0,79%	31,74%
	mn.d4	3,03%	37,20%	0,87%	31,21%
100	mn.d1	8,38%	40,85%	1,08%	32,86%
	mn.d2	8,01%	36,57%	1,05%	33,01%
	mn.d3	8,30%	46,04%	1,03%	40,88%
	mn.d4	10,56%	38,06%	1,13%	33,30%
200	mn.d1	17,10%	41,18%	1,56%	35,95%
	mn.d2	16,58%	37,80%	1,64%	34,00%
	mn.d3	17,16%	45,85%	1,63%	38,85%
	mn.d4	17,53%	45,79%	1,60%	36,32%

Πίνακας 4.18 - Αποτελέσματα παρακολούθησης διακομιστών

Range		
Threads	Range avg cpu	Range avg mem
50	0,09%	2,56%
100	0,10%	8,02%
200	0,08%	4,85%

Πίνακας 4.19 - Εύρος μέσων τιμών CPU/mem

Συμπεράσματα:

Όπως αναμενόταν συγκριτικά με το προηγούμενο πείραμα (SRR-SRR) η εξισορρόπηση της χρήσης της μνήμης βελτιώθηκε. Η συσσώρευση συνδέσεων (sessions) με μεγάλες ροές σε έναν διακομιστή ελαττώθηκε. Το εύρος των μέσων τιμών της δέσμευσης μνήμης των διακομιστών δεν ξεπέρασε το 8,02%.

4.3.4 Πείραμα 3 Statistics-Statistics

Στο προηγούμενο πείραμα (SRR-Statistics) χρησιμοποιήσαμε τον αλγόριθμο εξισορρόπησης φορτίου που βασίζεται στην δικτυακή κίνηση (statistics) στα αιτήματα που προορίζονται για την υπηρεσία βίντεο. Εφόσον στο προηγούμενο πείραμα παρατηρήσαμε ότι το πρόβλημα εξισορρόπησης των διακομιστών βελτιώνεται με την χρήση δυναμικού αλγορίθμου, σε αυτό το πείραμα χρησιμοποιούμε τον αλγόριθμοstatistics και για τις 2 υπηρεσίες.

Αποτελέσματα

Web HTTP Request (Statistics)				
Requests	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
50	4216	4136	4376	0
100	4895	4172	5750	0
200	6849	4629	8655	0

Πίνακας 4.20 -Απόδοση υπηρεσίας web

Video HTTP Request (Statistics)				
Requests	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
50	4216	4136	4376	0
100	15351	2048	31390	0
200	15118	3679	26049	0

Πίνακας 4.21 -Απόδοση υπηρεσίας video

Video -Statistics / Web -Statistics (Server CPU/Mem usage)					
Threads	**Server Name**	**Max cpu**	**Max memory**	**Avg cpu usage**	**Avg mem usage**
50	mn.d1	8,76%	37,20%	1,24%	32,80%
	mn.d2	8,16%	46,31%	0,63%	36,13%
	mn.d3	9,49%	42,58%	0,74%	34,42%
	mn.d4	7,94%	43,11%	0,96%	32,60%
100	mn.d1	29,95%	49,77%	2,13%	37,86%
	mn.d2	15,97%	41,81%	0,61%	30,33%
	mn.d3	15,85%	43,44%	1,15%	37,27%
	mn.d4	33,78%	46,98%	0,82%	34,93%
200	mn.d1	27,23%	59,30%	1,51%	46,73%
	mn.d2	44,52%	37,93%	1,33%	32,51%
	mn.d3	31,56%	47,87%	1,07%	37,24%
	mn.d4	14,99%	45,72%	0,67%	39,89%

Πίνακας 4.22 - Αποτελέσματα παρακολούθησης διακομιστών

Συμπεράσματα

Σε σύγκριση με τα δύο προηγούμενα πειράματα (SRR-SRR και SRR-Statistics) παρατηρούμε ότι αυτός ο συνδυασμός (Statistics-Statistics) είχε τα χειρότερα αποτελέσματα και στην κατανάλωση μνήμης και στη χρήση του επεξεργαστή (Πίνακας 4.23). Ο αλγόριθμος εξισορρόπησης φορτίου που βασίζεται σε στατιστικά λαμβάνει υπόψη του μόνο τη δικτυακή κυκλοφορία. Λαμβάνει πληροφορίες για τη δικτυακή κίνηση και εκχωρεί τα αιτήματα των πελατών στο διακομιστή του οποίου οσύνδεσμος (link) έχει την λιγότερη κίνηση (μικρότερο traffic), ανεξάρτητα από το είδος της υπηρεσίας για την οποία προορίζεται το κάθε αίτημα. Συνεπώς, το πρόβλημα που αναδεικνύεται σε αυτήν την περίπτωση είναι ότι τα αιτήματα που προορίζονται για μία υπηρεσία που έχει συγκεκριμένη επίδραση στο φόρτο ενός μηχανήματος (π.χ δέσμευσης μνήμης) να σωρεύονται σε ένα διακομιστή.

Πέρα από τον κίνδυνο υπερφόρτωσης ενός διακομιστή, παρατηρούμε ότι αυτή η περίπτωση έχει και αρνητική επίδραση και στην απόδοση των εφαρμογών. Ο χρόνος εξυπηρέτησης των αιτημάτων έφτασε στα 6849 msec (πίνακας 4.20) ενώ στα 2 προηγούμενα πειράματα δεν ξεπέρασε τα 4202.

Range		
Requests	**Range avg cpu**	**Range avg mem**
50	0,61%	3,53%
100	1,52%	7,53%
200	0,84%	14,22%

Πίνακας 4.23 - Εύρος μέσων τιμών

4.3.5 Πείραμα4 AALB-AALB

Στο πρώτο πείραμα (SRR-SRR) παρατηρήσαμε την ανάγκη για τη χρήση δυναμικών αλγορίθμων, γιατί όπως είδαμε ο αλγόριθμος κυκλικής επιλογής (SRR) δεν εξισορρόπησε το φορτίο σε επιθυμητά επίπεδα, επειδή τα αιτήματα με μεγάλες ροές σωρεύτηκαν σε ένα διακομιστή. Στο δεύτερο πείραμα (SRR-Statistics) παρατηρήσαμε ότι η χρήση του δυναμικού αλγόριθμου που βασίζεται στη δικτυακή κίνηση για υπηρεσίες που παράγουν μεγάλα flows, (υπηρεσία video streaming) βελτιώνει το πρόβλημα.

Στο τρίτο πείραμα (Statistics-Statistics) παρατηρήσαμε ότι η χρήση ενός δυναμικού αλγορίθμου που εξισορροπείτο φορτίο υπολογίζοντας μόνο μία παράμετρο όπως η δικτυακή κίνηση για όλα τα αιτήματα ανεξάρτητα την υπηρεσία, διογκώνει το πρόβλημα της εξισορρόπησης και αυτό έχει συνέπειες και στην απόδοση των υπηρεσιών.

Σε αυτό το πείραμα θέλουμε να εξετάσουμε την απόδοση των υπηρεσιών και την εξισορρόπηση του φορτίου των διακομιστών, χρησιμοποιώντας έναν αλγόριθμο εξισορρόπησης φορτίου που πέρα από την δικτυακή κίνηση θα ελέγχει και άλλες παραμέτρους όπως την κατανάλωση των επεξεργαστικών πόρων των διακομιστών (επεξεργαστή και μνήμη). Για την υλοποίηση του χρησιμοποιήσαμε τον αλγόριθμο που δημιουργήσαμε εμείς AALB, μόνο που σε αυτήν την περίπτωση ο αλγόριθμος δεν είναι προσαρμοσμένος στην εφαρμογή. Τα βάρη που ορίσαμε είναι 0,5 για την δικτυακή κίνηση 0,25 για την χρήση του επεξεργαστή και 0,25 για την χρήση της μνήμης και στις δύο υπηρεσίες.

Αποτελέσματα

Web HTTP Request (AALB)				
Threads	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
50	4210	4131	4340	0
100	4313	4126	8352	0
200	4456	4178	4764	0

Video HTTP Request (AALB)				
Threads	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
50	15227	1237	31438	0
100	13822	1039	31058	0
200	17956	1990	30004	0

Πίνακας 4.24 - Απόδοση υπηρεσιών

Video -AALB / Web -AALB (Server CPU/Mem usage)					
Threads	**Server Name**	**Max cpu**	**Max memory**	**Avg cpu usage**	**Avg mem usage**
50	mn.d1	9,58%	34,25%	0,98%	30,29%
	mn.d2	5,27%	42,95%	0,49%	34,24%
	mn.d3	8,95%	34,41%	0,82%	29,71%
	mn.d4	7,94%	42,79%	1,27%	36,00%
100	mn.d1	9,55%	39,68%	1,16%	33,18%
	mn.d2	8,77%	44,64%	1,10%	36,55%
	mn.d3	9,06%	37,63%	0,65%	31,80%
	mn.d4	11,65%	40,44%	1,41%	33,46%
200	mn.d1	13,76%	39,16%	0,94%	36,30%
	mn.d2	16,12%	49,02%	1,03%	43,76%
	mn.d3	20,69%	46,09%	1,43%	37,50%
	mn.d4	16,96%	43,79%	1,51%	36,38%

Πίνακας 4.25 - Αποτελέσματα παρακολούθησης διακομιστών

Range		
Threads	Range avg cpu	Range avg mem
50	0,78%	6,29%
100	0,76%	4,75%
200	0,57%	7,46%

Πίνακας 4.26 - Εύρος μέσων τιμών

Συμπεράσματα

Σε αυτό το πείραμα χρησιμοποιήσαμε τον αλγόριθμο AALB με ίδια βάρη και στις δυο εφαρμογές. Παρατηρήσαμε ότι η εξισορρόπηση του φόρτου των διακομιστών

είχε καλύτερα αποτελέσματα από το προηγούμενο (Statistics-Statistics) (Πίνακας 4.23). Παρόλα αυτά όμως σε σύγκριση με το πείραμα 2 (SRR-Statistics) δεν βελτιώνει ούτε την απόδοση των εφαρμογών (Πινακας 4.16) ούτε την εξισορρόπηση του φόρτου των διακομιστών (Πίνακας 4.19). Άρα από αυτό το πείραμα σε σύγκριση με τα προηγούμενα προκύπτει η ανάγκη για χρήση μηχανισμών εξισορρόπησης φορτίου που να είναι προσαρμοσμένοι στο είδος της εφαρμογής.

4.3.6 Πείραμα 5 ALLB-Statistics

Οι αλγόριθμοι που εξετάζουμε σε αυτήν την εργασία είναι οι εξής: ο στατικός αλγόριθμος κυκλικής επιλογής (SRR), ο δυναμικός αλγόριθμος που βασίζεται στην δικτυακή κίνηση (Statistics) και αλγόριθμος που δημιουργήσαμε AALB και βασίζεται στη δικτυακή κίνηση και στην κατανάλωση τον επεξεργαστικών πόρων των διακομιστών και είναι προσαρμοσμένος στο είδος της εφαρμογής. Και για τους τρεις αλγορίθμους υλοποιήσαμε διαφορετικό πείραμα χρησιμοποιώντας τον καθένα και για τις δύο εφαρμογές (Πείραμα 1. SRR-SRR, Πείραμα 3. Statistics-Statistics, Πείραμα 4. ALLB-ALLB).

Στο πείραμα 3 χρησιμοποιήσαμε διαφορετικούς μηχανισμούς για κάθε υπηρεσία, τον SRR για την υπηρεσία παροχής ιστοσελίδων και το Statistics για την υπηρεσία βίντεο. Η επιλογή αυτού του συνδυασμού έγινε αφού πρώτα αξιολογήσαμε ποιος αλγόριθμος εξισορρόπησης φορτίου έχει την καλύτερη απόδοση για κάθε εφαρμογή (στο Σενάριο 2). Το πείραμα 3 είχε τα καλυτέρα αποτελέσματα όσον αφορά στην απόδοση των υπηρεσιών και στην εξισορρόπηση των επεξεργαστικών πόρων των διακομιστών. Αρά, από την παραπάνω ανάλυση προκύπτει πως η χρήση διαφορετικών μηχανισμών εξισορρόπησης φορτίου ανάλογα με το είδος της υπηρεσίας που παρέχεται έχει την καλύτερη απόδοση.

Σε αυτό το πείραμα χρησιμοποιούμε τον αλγόριθμο AALB για την υπηρεσία παροχής ιστοσελίδων και τον Statistics για την υπηρεσία παροχής βίντεο. Από το πρώτο σενάριο της εργασίας παρατηρήσαμε ότι στηνweb εφαρμογή η χρήση του επεξεργαστή ήταν πολύ μεγαλύτερη (η μέση τιμή έφτασε ως 15,73%) από την υπηρεσία Video Streaming (0,79%). Οπότε τα βάρη που ορίσαμε για τον AALB είναι 0,5 για την δικτυακή κίνηση 0,4 για την χρήση του επεξεργαστή και 0,1 για την χρήση της μνήμης.

Αποτελέσματα

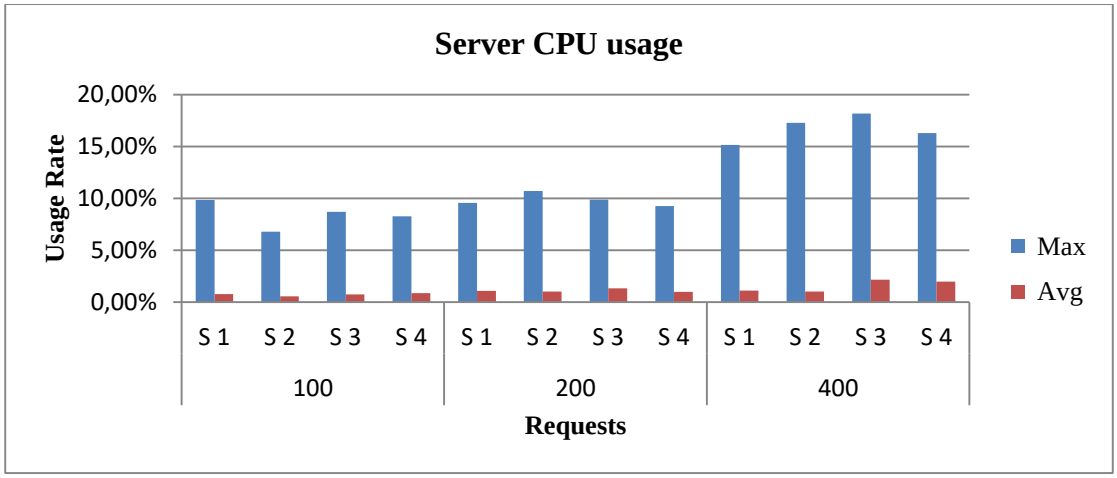
Web HTTP Request (AALB)				
Threads	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
50	4187	4124	4340	0
100	4220	4123	4428	0
200	4458	4157	4876	0

Video HTTP Request (Statistics)				
Threads	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
50	15509	1883	30741	0
100	16212	2021	31100	0
200	14964	1178	30023	0

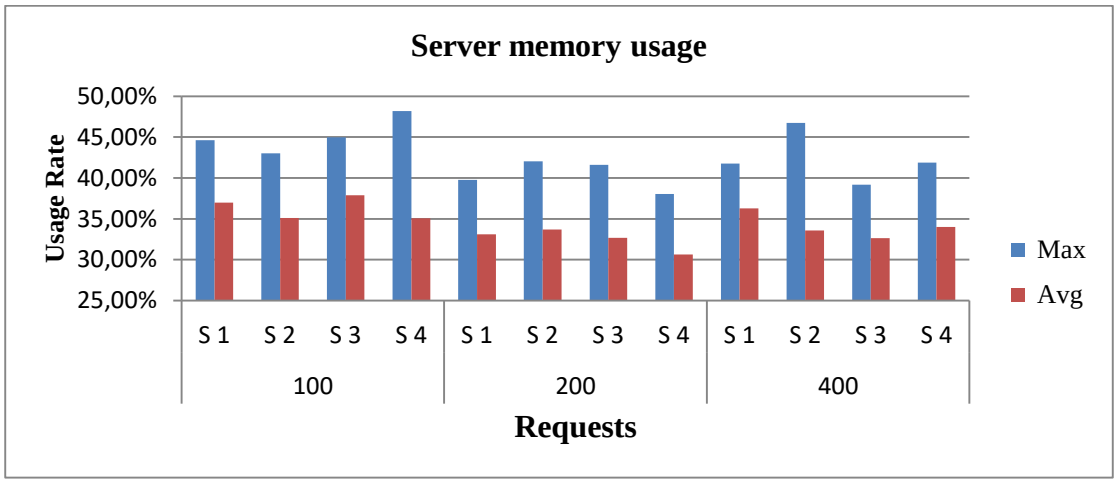
Πίνακας 4.27 - Απόδοση υπηρεσιών

Video -AALB / Web -Statistics (Server CPU/Mem usage)					
Threads	**Server Name**	**Max cpu**	**Max memory**	**Avg cpu usage**	**Avg mem usage**
50	mn.d1	9,84%	44,61%	0,79%	36,98%
	mn.d2	6,79%	43,01%	0,55%	35,10%
	mn.d3	8,70%	44,99%	0,76%	37,90%
	mn.d4	8,26%	48,19%	0,86%	35,08%
100	mn.d1	9,55%	39,78%	1,09%	33,12%
	mn.d2	10,71%	42,03%	1,01%	33,70%
	mn.d3	9,88%	41,62%	1,34%	32,67%
	mn.d4	9,26%	38,03%	1,00%	30,64%
200	mn.d1	15,15%	41,76%	1,13%	36,27%
	mn.d2	17,27%	46,74%	1,02%	33,58%
	mn.d3	18,18%	39,18%	2,17%	32,65%
	mn.d4	16,28%	41,89%	1,98%	34,02%

Πίνακας 4.28 - Αποτελέσματα παρακολούθησης διακομιστών



Γράφημα 4.15 - Κατανάλωση CPU



Γράφημα 4.16 - Δέσμευση μνήμης RAM

Range		
Requests	Range avg cpu	Range avg mem
50	0,31%	2,82%
100	0,34%	3,06%
200	1,15%	3,62%

Πίνακας 4.29 - Εύρος μέσων τιμών

Συμπεράσματα

Από την αξιολόγηση των πειραμάτων μέχρι αυτό το σημείο παρατηρούμε ότι η χρήση του αλγορίθμου AALB που είναι προσαρμοσμένος στην υπηρεσία παροχής ιστοσελίδων και ο αλγόριθμος που βασίζεται σε στατιστικά στην υπηρεσία βίντεο είχε τα καλύτερα αποτελέσματα από όλα τα πειράματα, ως προς την εξυπηρέτηση των επεξεργαστικών πόρων των διακομιστών. Παρόλα αυτά όμως δεν κατάφερε να

ξεπεράσει την απόδοση των εφαρμογών του πειράματος (SRR-SRR). Η φόρτωση της ιστοσελίδας που ζητήθηκε ήταν πιο αργή κατά 200 msec.

Αυτό συμβαίνει γιατί κάθε σύνδεσμος που υπάρχει μέσα σε μία ιστοσελίδα είναι ένα ξεχωριστό αίτημα προς την υπηρεσία παροχής ιστοσελίδων. Στους τρείς αλγορίθμους που χρησιμοποιήσαμε η εκχώρηση των κανόνων των ροών στους μεταγωγείς από τον ελεγκτή γίνεται μέσω της ανταποκριτικής (reactive) μεθόδου. Δηλαδή για κάθε νέο αίτημα που εισέρχεται στο δίκτυο ο μεταγωγέας ενημερώνει τον ελεγκτή, ο ελεγκτής υπολογίζει την κατάλληλη διαδρομή συμφώνα με τον μηχανισμό εξισορρόπησης φορτίου που έχει ρυθμιστεί και εισάγει τον κανόνα για τη ροή στον μεταγωγέα. Λόγω του ότι ο AALB αλγόριθμος λαμβάνει υπόψη τις περισσότερες παραμέτρους είναι ο πιο πολύπλοκος στην εκτέλεση του. Επίσης ο αλγόριθμος AALB ενημερώνεται κάθε 3 δευτερόλεπτα για την δικτυακή κίνηση και κάθε δευτερόλεπτο για την επεξεργαστικό φόρτο των διακομιστών. Όποτε για τουλάχιστον 1 δευτερόλεπτο όλα τα αιτήματα θα έχουν τον ίδιο προορισμό. Από την παραπάνω ανάλυση προκύπτει ότι ο SRR μηχανισμός είναι ο πιο ιδανικός για την υπηρεσία παροχής ιστοσελίδων.

4.3.7 Πείραμα 6 SRR-AALB

Από την ανάλυση του προηγούμενου πειράματος προκύπτει ότι λόγω της απλότητας του, ο αλγόριθμος Simple Round Robin είναι ιδανικός για την υπηρεσία παροχής ιστοσελίδων. Στο τελευταίο πείραμα αυτού του σεναρίου θα χρησιμοποιήσουμε τον SRR για την web υπηρεσία και τον AALB για την υπηρεσία βίντεο. Από το πρώτο σενάριο παρατηρήσαμε ότι η υπηρεσία βίντεο δεσμεύει περισσότερο χώρο στην μνήμη από την webυπηρεσία, οπότε τα βάρη που ορίσαμε είναι 0.5 για την δικτυακή κυκλοφορία, 0.1 για τη χρήση του επεξεργαστή και 0.4 για τη χρήση της μνήμης.

Αποτελέσματα:

Web HTTP Request (SRR)				
Threads	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
50	4149	4120	4213	0
100	4155	4118	4497	0
200	4214	4120	4663	0

Πίνακας 4.30 - Απόδοση web υπηρεσίας

Video HTTP Request (AALB)				
Threads	**Avg Load time (msec)**	**Min Load time (msec)**	**Max load time (msec)**	**Error %**
50	15227	1237	31438	0
100	13822	1039	31058	0
200	17956	1990	30004	0

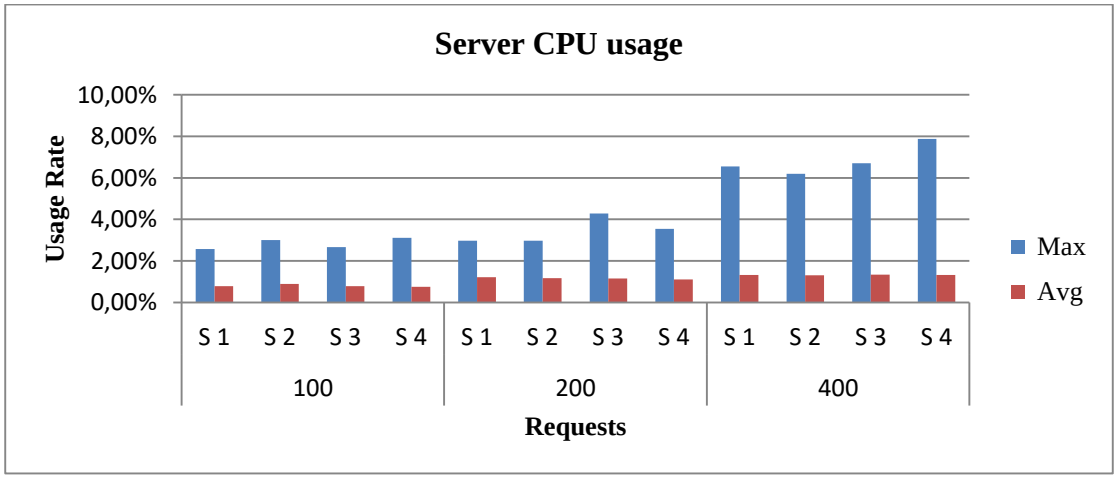
Πίνακας 4.31 - Απόδοση video υπηρεσίας

Video -SRR / Web -AALB (Server CPU/Mem usage)					
Threads	**Server Name**	**Max cpu**	**Max memory**	**Avg cpu usage**	**Avg mem usage**
50	mn.d1	2,57%	34,01%	0,78%	30,03%
	mn.d2	3,01%	42,07%	0,90%	33,21%
	mn.d3	2,66%	33,63%	0,79%	30,35%
	mn.d4	3,11%	40,95%	0,76%	33,73%
100	mn.d1	2,97%	39,92%	1,21%	33,27%
	mn.d2	2,97%	41,08%	1,17%	33,37%
	mn.d3	4,29%	45,82%	1,15%	33,84%
	mn.d4	3,55%	43,62%	1,11%	33,37%
200	mn.d1	6,55%	46,51%	1,33%	36,08%
	mn.d2	6,19%	40,87%	1,31%	34,11%
	mn.d3	6,71%	40,90%	1,34%	34,31%
	mn.d4	7,87%	45,32%	1,32%	35,21%

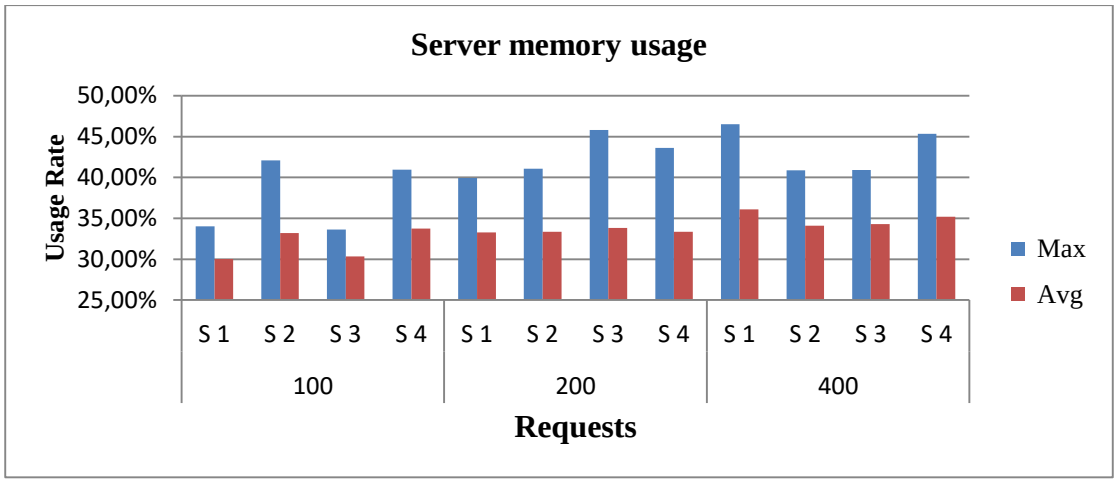
Πίνακας 4.32 - Αποτελέσματα παρακολούθησης διακομιστών

Range		
Threads	Range avg cpu	Range avg mem
50	0,14%	3,60%
100	0,10%	0,57%
200	0,03%	1,97%

Πίνακας 4.33 - Εύρος τιμών



Γράφημα 4.17 - Κατανάλωση CPU



Γράφημα 4.18 - Δέσμευση μνήμης RAM

Συμπεράσματα

Αυτός ο συνδυασμός μηχανισμών εξισορρόπησης φορτίου είχε τα καλυτέρα αποτελέσματα από όλους τους άλλους στην απόδοση των εφαρμογών και στην εξισορρόπηση των διακομιστών.

4.3.8 Συγκριτική ανάλυση

Οι αλγόριθμοι που αξιολογήσαμε σε αυτήν την εργασία είναι: Ο στατικός αλγόριθμος κυκλικής επιλογής (SRR), ο δυναμικός αλγόριθμος που βασίζεται στη δικτυακή κίνηση (Statistics) και αλγόριθμος που δημιουργήσαμε (AALB), ο οποίος βασίζεται στη δικτυακή κίνηση και στην κατανάλωση των πόρων επεξεργασίας των διακομιστών. Δοκιμάσαμε τους τρεις αλγορίθμους ταυτόχρονα και για τις δύο εφαρμογές (πείραμα 1. SRR-SRR, πείραμα 3. Statistics-Statistics, πείραμα 4. ALLB-ALLB). Στο πρώτο πείραμα (SRR-SRR) παρατηρήσαμε την ανάγκη για τη χρήση δυναμικών αλγορίθμων, γιατί όπως είδαμε ο αλγόριθμος κυκλικής επιλογής (SRR) δεν εξισορρόπησε το φορτίο σε επιθυμητά επίπεδα, καθώς αιτήματα που συνδέονται με μεγάλες ροές (flows) σωρεύτηκαν σε έναν διακομιστή. Στο τρίτο πείραμα (Statistics-Statistics) παρατηρήσαμε ότι η χρήση ενός δυναμικού αλγορίθμου που εξισορροπείτο φορτίο υπολογίζοντας μόνο μία παράμετρο, όπως η δικτυακή κίνηση για όλα τα αιτήματα ανεξάρτητα το είδος της υπηρεσίας, διογκώνει το πρόβλημα της εξισορρόπησης και αυτό έχει συνέπειες στην απόδοση των υπηρεσιών. Στο πείραμα 4 (AALB-AALB) παρατηρήσαμε ότι χρήση μηχανισμών που πέρα από τη δικτυακή κίνηση ελέγχουν και άλλες παραμέτρους, όπως την κατανάλωση των πόρων

επεξεργασίας των διακομιστών (επεξεργαστή και μνήμη) βελτιώνει το πρόβλημα της εξισορρόπησης φορτίου.

Αφού εξετάσαμε τη χρήση του κάθε μηχανισμού και για τις δύο υπηρεσίες μαζί, υλοποιήσαμε πειράματα με διαφορικό μηχανισμό για κάθε υπηρεσία. Στο πείραμα (AALB-Statistics) χρησιμοποιήσαμε τον αλγόριθμο AALB (προσαρμόζοντας τον) στην υπηρεσία παροχής ιστοσελίδων και τον αλγόριθμο που βασίζεται σε στατιστικά στην υπηρεσία βίντεο. Παρατηρήσαμε ότι αυτός ο συνδυασμός είχε τα καλύτερα αποτελέσματα από όλα τα πειράματα ως προς την εξυπηρέτηση των πόρων επεξεργασίας των διακομιστών. Παρόλα αυτά όμως η φόρτωση της ιστοσελίδας που ζητήθηκε ήταν πιο αργή κατά 200 msec από τη χρήση του SRR. Στο τελευταίο πείραμα που υλοποιήσαμε (SRR-ALLB) χρησιμοποιήσαμε τον μηχανισμό κυκλικής επιλογής για την υπηρεσία webκαι τον AALB (προσαρμόζοντας τον στο είδος της εφαρμογής) για την υπηρεσία βίντεο. Αυτός ο συνδυασμός μηχανισμών είχε τα καλύτερα αποτελέσματα από όλους τους άλλους, με όρους απόδοσης των εφαρμογών και εξισορρόπησης του φορτίου των διακομιστών. Συνοψίζοντας από την ανάλυση αυτού του σεναρίου καταλήγουμε στα ακόλουθα συμπεράσματα:

1. Στην περίπτωση που έχουμε αιτήματα που προορίζονται για διαφορετικές υπηρεσίες προς ένα ομοιογενές σύμπλεγμα διακομιστών, πρέπει να χρησιμοποιούμε αλγορίθμους εξισορρόπησης φορτίου που είναι προσαρμοσμένοι στο είδος των υπηρεσιών αυτών.
2. Η χρήση ενός απλού αλγορίθμου, όπως ο SRR,για υπηρεσίες που παράγουν ροές μικρού μεγέθους και διάρκειας ζωής έχει την καλύτερη απόδοση.
3. Η χρήση του δυναμικού αλγόριθμου που βασίζεται στη δικτυακή κίνηση (Statistics) για υπηρεσίες που παράγουν μεγάλα flows, (π.χ υπηρεσία video streaming) βελτιώνει το πρόβλημα της εξισορρόπησης φορτίου.
4. Ο αλγόριθμος που δημιουργήθηκε από εμάς (AALB) και λαμβάνει υπόψη του περισσότερους παράγοντες για τον υπολογισμό του φορτίου, όπως είναι τα ποσοστά χρήσης του επεξεργαστή και της μνήμης, έχει καλύτερη απόδοση από τον αλγόριθμο που βασίζεται μόνο στη δικτυακή κίνηση (Statistics).

5 Επίλογος

Η αποδοτική ρύθμιση της δικτυακής κίνησης είναι πολύ σημαντική για την απόδοση των εφαρμογών και την καλύτερη αξιοποίηση των πόρων των διακομιστών. Η εξισορρόπηση φορτίου στα δίκτυα υπολογιστών είναι μια τεχνική που χρησιμοποιείται για το διαμοιρασμό του φόρτου εργασίας σε πολλαπλούς δικτυακούς συνδέσμους (links) ή υπολογιστές. Μια πρόταση για τη βελτίωση της απόδοσης των μηχανισμών εξισορρόπησης φορτίου είναι η χρήση δυναμικών αλγορίθμων εξισορρόπησης φορτίου μέσω των ευφυών προγραμματιζόμενων δικτύων (ΕΠΔ).

Στην παρούσα εργασία παρουσιάσαμε το βασικό θεωρητικό πλαίσιο που αναφέρεται στην αρχιτεκτονική των ευφυών προγραμματιζόμενων δικτύων και τα βασικά χαρακτηριστικά της τεχνολογίας εξισορρόπησης φορτίου. Στην πειραματική προσέγγιση της εργασίας προσομοιώσαμε ένα περιβάλλον ΕΠΔ, με σκοπό τη μελέτη και την αξιολόγηση μηχανισμών εξισορρόπησης φορτίου, που στόχο έχουν τη βελτίωση του τρόπου διαμοιρασμού της δικτυακής κυκλοφορίας ανάμεσα στους διακομιστές. Οι αλγόριθμοι εξισορρόπησης φορτίου που χρησιμοποιήσαμε είναι: ο στατικός αλγόριθμος κυκλικής επιλογής (SRR), ο δυναμικός αλγόριθμος που βασίζεται στη δικτυακή κίνηση (Statistics) και ο αλγόριθμος που είναι μία δικιά μας πρόταση για την εξισορρόπηση φορτίου, ο οποίος διαμοιράζει το φόρτο σύμφωνα με το είδος της εφαρμογής και βασίζεται σε πληροφορίες της δικτυακής κίνησης και της επεξεργαστικής κατανάλωσης των διακομιστών. (Application Adaptive Load Balancing.).

Τέλος, παρουσιάσαμε και αναλύσαμε τις μεθόδους που χρησιμοποιήσαμε σε πρακτικό επίπεδο και υλοποιούμε διάφορα πειραματικά σενάρια, καταγράφοντας τα αποτελέσματα των μετρικών της απόδοσης των αλγορίθμων.

5.1 Σύνοψη και συμπεράσματα

Από το θεωρητικό υπόβαθρο της εργασίας συμπεραίνουμε ότι: τα ευφυή προγραμματιζόμενα δίκτυα διαχωρίζοντας το επίπεδο του ελέγχου από αυτό των δεδομένων παρέχουν μία νέα δυναμική αρχιτεκτονική, μέσω της οποίας, δύσκολα προβλήματα που αφορούν στη βελτιστοποίηση της απόδοσης των δικτύων (όπως η εξισορρόπηση φορτίου) γίνονται διαχειρίσιμα με σωστά σχεδιασμένους κεντρικούς αλγόριθμους.

Στην πειραματική προσέγγιση της εργασίας προσομοιώσαμε ένα δικτυακό περιβάλλον, στο οποίο παρέχουμε μέσω των διακομιστών, μια υπηρεσία παροχής ιστοσελίδων και μια υπηρεσία ζωντανής μετάδοσης βίντεο. Για κάθε εφαρμογή διερευνήσαμε κατά πόσο επηρεάζονται οι επεξεργαστικοί πόροι των διακομιστών και συμπεραίνουμε ότι, για τον ίδιο αριθμό πελατών, κάθε εφαρμογή έχει διαφορετική κατανάλωση επεξεργαστικών πόρων. Πιο συγκεκριμένα παρατηρήσαμε ότι η υπηρεσία παροχής ιστοσελίδων κατανάλωσε πολύ μεγαλύτερη ποσότητα επεξεργαστικής ισχύος από την υπηρεσία μετάδοσης βίντεο. Αντιθέτως η υπηρεσία μετάδοσης βίντεο δέσμευσε περισσότερο χώρο στη μνήμη έναντι της υπηρεσίας παροχής ιστοσελίδων.

Από την σύγκριση των αλγορίθμων για κάθε εφαρμογή ξεχωριστά παρατηρήσαμε την ανάγκη για χρήση δυναμικών αλγορίθμων, γιατί όπως είδαμε ο αλγόριθμος κυκλικής επιλογής (SRR) δεν εξισορρόπησε το φορτίο σε επιθυμητά επίπεδα, καθώς αιτήματα που παράγουν μεγάλες ροές (flows) σωρεύτηκαν σε έναν διακομιστή.

Οι δυναμικοί αλγόριθμοι (AALB, Statistics) παρατηρούμε ότι εξισορρόπησαν πολύ καλύτερα το φορτίο των αιτημάτων για την βίντεο υπηρεσία ανάμεσα στους διακομιστές, επειδή παρακολουθούν τη δικτυακή κίνηση και προωθούν τα αιτήματα στο διακομιστή με το μικρότερο φορτίο, συνεπώς δεν υπάρχει ο κίνδυνος αιτήματα μεγάλης διάρκειας ζωής να σωρευτούν σε ένα διακομιστή.

Στο σενάριο που τα αιτήματα των πελατών απευθύνονται και στις δυο εφαρμογές ταυτόχρονα, παρατηρήσαμε ότι η χρήση ενός δυναμικού αλγορίθμου που εξισορροπεί το φόρτο υπολογίζοντας μόνο μία παράμετρο όπως τη δικτυακή κίνηση για όλα τα αιτήματα ανεξάρτητα το είδος της υπηρεσίας, διογκώνει το πρόβλημα της εξισορρόπησης και αυτό έχει συνέπειες και στην απόδοση των υπηρεσιών. Από την άλλη πλευρά η χρήση μηχανισμών που πέρα από τη δικτυακή κίνηση ελέγχουν και άλλες παραμέτρους όπως την κατανάλωση των επεξεργαστικών πόρων των διακομιστών βελτιώνει το πρόβλημα της εξισορρόπησης.

Αφού εξετάσαμε τη χρήση του κάθε μηχανισμού και στις δύο υπηρεσίες ταυτόχρονα, υλοποιήσαμε πειράματα με διαφορικό μηχανισμό για κάθε υπηρεσία. Σε αυτό το σενάριο παρατηρήσαμε ότι η χρήση διαφορετικών αλγορίθμων ανάλογα με το είδος της υπηρεσίας είναι πιο αποδοτική από το να χρησιμοποιούμε έναν αλγόριθμο για όλες της εφαρμογές. Στο τελικό πείραμα που υλοποιήσαμε χρησιμοποιήσαμε τον μηχανισμό κυκλικής επιλογής για την webυπηρεσία και τον

AALB (προσαρμόζοντας τον στο είδος της εφαρμογής) για την βίντεο υπηρεσία. Αυτός ο συνδυασμός μηχανισμών είχε τα καλύτερα αποτελέσματα από όλους τους άλλους, τόσο στην απόδοση των εφαρμογών όσο και στην εξισορρόπηση των διακομιστών.

Συνοψίζοντας τα συμπεράσματα αυτής της εργασίας παρατηρούμε ότι: α) Στην περίπτωση που έχουμε αιτήματα που προορίζονται για διαφορετικές υπηρεσίες προς ένα ομοιογενές σύμπλεγμα διακομιστών, η χρήση αλγορίθμων εξισορρόπησης φορτίου που είναι προσαρμοσμένοι στο είδος των υπηρεσιών αυτών έχουν την καλύτερη απόδοση, β) επίσης η χρήση ενός απλού αλγορίθμου όπως ο SRR για υπηρεσίες που παράγουν ροές μικρού μεγέθους και διάρκειας ζωής αποδίδουν καλύτερα, γ) η χρήση δυναμικών αλγορίθμων για υπηρεσίες που παράγουν μεγάλα flows, (π.χ. υπηρεσία video streaming) βελτιώνει το πρόβλημα της εξισορρόπησης φορτίου, δ) ο αλγόριθμος που δημιουργήθηκε από εμάς (AALB) και λαμβάνει υπόψη του περισσότερους παράγοντες για τον υπολογισμό του φορτίου, όπως είναι τα ποσοστά χρήσης του επεξεργαστή και της μνήμης, έχει καλύτερη απόδοση σε σχέση με τους άλλους δύο αλγορίθμους.

5.2 Όρια και περιορισμοί της εργασίας

Ο βασικός περιορισμός της εργασίας μας είναι ότι δεν υπήρχε η δυνατότητα εκτέλεσης των πειραμάτων σε δικτυακό περιβάλλον με πραγματικές συσκευές ΕΠΔ. Για να αντιμετωπίσουμε αυτήν την αδυναμία, χρησιμοποιήσαμε τον εξομοιωτή Mininet, τον οποίο εγκαταστήσαμε σε έναν υπολογιστή. Συνεπώς, πολλές πτυχές των πειραμάτων, όπως ο αριθμός των διαθέσιμων διακομιστών του δικτύου και η προσομοίωση των αιτημάτων των πελατών προς αυτούς, περιορίστηκαν στις δυνατότητες επεξεργασίας του υπολογιστή στον οποίο υλοποιήθηκαν τα πειράματα.

Ένας άλλος σημαντικός περιορισμός που συναντήσαμε κατά την υλοποίηση της εργασίας μας ήταν η αδυναμία στο να βρούμε έτοιμο κώδικα αλγόριθμων εξισορρόπησης φορτίου, ώστε να επεκτείνουμε την πειραματική μας ανάλυση σε περισσότερους αλγορίθμους της βιβλιογραφίας.

5.3 Μελλοντικές Επεκτάσεις

Πιστεύουμε ότι θα είχε ενδιαφέρον η πραγματοποίηση των πειραμάτων μας σε ένα πραγματικό δικτυακό περιβάλλον, με σκοπό να ελεγχθεί αν τα αποτελέσματα είναι αντίστοιχα με αυτά του εικονικού. Επίσης, λόγω του ότι σε ένα πραγματικό

δικτυακό περιβάλλον οι δυνατότητες επεξεργασίας είναι πολλαπλάσιες από ότι σε ένα εικονικό, θα υπάρχει η δυνατότητα πραγματοποίησης πειραμάτων σε μεγαλύτερη κλίμακα (π.χ. με περισσότερα αιτήματα πελατών, μεγαλύτερο αριθμό διακομιστών, κ.α.).

Στα σχέδια μας είναι η σύγκριση του προτεινόμενου αλγορίθμου με περισσότερους αλγορίθμους εξισορρόπησης φορτίου της βιβλιογραφίας (π.χ. ο Hedera).

Ακόμη, ενδιαφέρον θα παρουσίαζε η διερεύνηση της εφαρμογής του αλγορίθμου μας σε ένα περιβάλλον με περισσότερες Διαδικτυακές υπηρεσίες από αυτές που διαθέταμε κατά την εργασία μας, με σκοπό να διαπιστωθεί, αν συνεχίζουν να ισχύουν τα ίδια ποιοτικά συμπεράσματα. Ένα ενδιαφέρον πρόβλημα είναι ο δυναμικός εντοπισμός του είδους της εφαρμογή από τον αλγόριθμο. Αυτό το ζήτημα αξίζει ξεχωριστή διερεύνηση.

Λόγω του ότι οι αλγόριθμοι εξισορρόπησης φορτίου που χρησιμοποιήσαμε εκχωρούν τους κανόνες των ροών στους μεταγωγείς μέσω της ανταποκριτικής (reactive) μεθόδου, ενδιαφέρον θα παρουσίαζε η τροποποίηση τους έτσι ώστε να λειτουργούν βάσει της προληπτικής (proactive) μεθόδου, με στόχο να εντοπιστούν τα πλεονεκτήματα και τα μειονεκτήματα αυτής της τακτικής.

Βιβλιογραφία

[1] A. Aditya, U. Chatterjee, and S. Gupta, “A Comparative Study of Different Static and Dynamic Load Balancing Algorithm in Cloud Computing with Special Emphasis on Time Factor,”*International Journal of Current Engineering and Technology, Vol.5, No.3 (June-2015)*

[2] T. Deepa and D. Cheelu, "A comparative study of static and dynamic load balancing algorithms in cloud computing," *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, Chennai, 2017, pp. 3375-3378.

[3] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, “A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617-1634, 2014..

[4] H. Zhong, Q. Lin, J. Cui, R. Shi, and L. Liu, “An Efficient SDN Load Balancing Scheme Based on Variance Analysis for Massive Mobile Users,” *Mobile Information Systems*, vol. 2015, pp. 1-9, 2015.

.

[5] “Apache JMeter - Apache JMeter™.” [Online]. Available: https://jmeter.apache.org/.

[6] A. Ghaffarinejad, “Comparing a Commercial and an SDN-Based Load Balancer in a Campus Network,”*2015 IEEE 82nd Vehicular Technology Conference*

[7] “Containernet.” [Online]. Available: https://containernet.github.io/.

[8] Z. Shang, W. Chen, Q. Ma, and B. Wu, “Design and implementation of server cluster dynamic load balancing based on OpenFlow,” in *2013 International Joint Conference on Awareness Science and Technology & Ubi-Media Computing (iCAST 2013 & UMEDIA 2013)*, Aizuwakamatsu, Japan, 2013, pp. 691-697.

[9] Z. Shang, W. Chen, Q. Ma and B. Wu, "Design and implementation of server cluster dynamic load balancing based on OpenFlow," *2013 International Joint Conference on Awareness Science and Technology & Ubi-Media Computing (iCAST 2013 & UMEDIA 2013)*, Aizu-Wakamatsu, 2013, pp. 691-697.

[10] “Docker Documentation,” *Docker Documentation*, 21-Feb-2019. [Online]. Available: https://docs.docker.com/. [Accessed: 21-Feb-2019].

[11] “Documentation · mininet/mininet Wiki · GitHub.” [Online]. Available: https://github.com/mininet/mininet/wiki/Documentation.

[12] Shuo Wang, Jiao Zhang, Tao Huang, Tian Pan, Jiang Liu and Yunjie Liu, "FDALB: Flow distribution aware load balancing for datacenter networks," *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, Beijing, 2016, pp. 1-2

[13] “Flask (A Python Microframework).” [Online]. Available: http://flask.pocoo.org/..

[14] G. Dwyer, *Flask by example unleash the full potential of the Flask web framework by creating simple yet powerful web applications*. Birmingham, UK: Packt Publishing, 2016.

[15] “Floodlight OpenFlow Controller -,” *Project Floodlight*. [Online]. Available: http://www.projectfloodlight.org/floodlight/.

[16] “GitHub - containernet/containernet: Mininet fork adding support for container-based (e.g. Docker) emulated hosts.” [Online]. Available: https://github.com/containernet/containernet.

[17] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic Flow Scheduling for Data Center Networks,” p. 15.

[18] “Al-Fares et al. - Hedera Dynamic Flow Scheduling for Data Center N.” .

[19] J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu, and Y. Liu, “Load Balancing in Data Center Networks: A Survey,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2324-2352, 2018.

[20] “Open vSwitch.” [Online]. Available: https://www.openvswitch.org/.

[21] “Maestro Platform.” [Online]. Available: http://zhengcai.github.io/maestro-platform/.

[22] “The NOX Controller.” [Online]. Available: https://github.com/noxrepo/nox.

[23] P. Berde *et al.*, “ONOS: towards an open, distributed SDN OS,” in *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN ’14*, Chicago, Illinois, USA, 2014, pp. 1-6.

[24] “Open Networking Foundation is an operator led consortium leveraging SDN, NFV and Cloud technologies to transform operator networks and business models.” [Online]. Available: https://www.opennetworking.org/.

[25] “OpenDaylight.” [Online]. Available: https://www.opendaylight.org/.

[26] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China, Q. Du, and H. Zhuang, “OpenFlow-Based Dynamic Server Cluster Load Balancing with Measurement Support,” *Journal of Communications*, 2015.

[27] R. Wang, D. Butnariu, and J. R. P. University, “OpenFlow-Based Server Load Balancing Gone Wild,”.

[28] V. Mathapati and D. A. R. Aswatha, “PERFORMANCE ANALYSIS OF

SYSTEM RESOURCES BY SERVER MONITORING," vol. 2, no. 7, p. 5, 2013.

[29] "The NOX Controller." [Online]. Available: https://github.com/noxrepo/nox.

[30] "Ryu SDN Framework." [Online]. Available: http://osrg.github.io/ryu/.

[31] Y. Desmouceaux, P. Pfister, J. Tollet, M. Townsley, and T. Clausen, "Scalable and Application-Aware Load Balancing with Segment Routing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 819-834, Apr. 2018.

[32] Hailong Zhang and Xiao Guo, "SDN-based load balancing strategy for server cluster," *2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems*, Shenzhen, 2014, pp. 662-667.

[33] "Server Load Balancing · OpenState-SDN/ryu Wiki · GitHub." [Online]. Available: https://github.com/OpenState-SDN/ryu/wiki/Server-Load-Balancing.

[34] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *arXiv:1406.0440 [cs]*, Jun. 2014.

[35] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, Hong Kong, China, 2013, p. 13.

[36] Liang Guo and I. Matta, "The war between mice and elephants," in *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, Riverside, CA, USA, 2001, pp. 180-188.

[37] "VLC: Official site - Free multimedia solutions for all OS! - VideoLAN." [Online]. Available: https://www.videolan.org/.

[38] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," *Sigcomm '07*, pp. 1-12, 2007.

[39]R. Godha and S. Prateek, "Load Balancing in a Network," *International Journal of Scientific and Research Publications*, vol. 4, no. 10, p. 3, 2014.

[40]L. Yang, T. Anderson, R. Dantu, and R. Gopal, *Forwarding and Control Element Separation (ForCES) Framework*. 2004.

[41]"Internet Engineering Task Force (IETF)." [Online]. Available: https://www6.ietf.org/.

[42]A. Maleki, M. Hossain, J. Georges, E. Rondeau, and T. Divoux, "An SDN Perspective to Mitigate the Energy Consumption of Core Networks - GÉANT2,"

Παράρτημα Α- Κώδικας

Flask

```
from flask import Flask, render_template, abort , request
fromjinja2 import TemplateNotFound
from flask import send_file

app = Flask(__name__, static_url_path='')
@app.route('/')
def render():
    return render_template('myfiles.html')
@app.route('/')
def root():
    return app.send_static_file('index.html')
@app.route('/<page>')
def html_lookup(page):
    try:
        return render_template('{}.html'.format(page))
    except TemplateNotFound:
        abort(404)
if __name__ == '__main__':
    app.run(host= '0.0.0.0', port=80)
```

Δημιουργία τοπολογίας

```
1.  #!/usr/bin/python
2.  from mininet.net import Containernet
3.  from mininet.node import RemoteController, Docker
4.  from mininet.cli import CLI
5.  from mininet.link import TCLink
6.  from mininet.log import info, setLogLevel
7.  from mininet.node import OVSKernelSwitch
8.  from mininet.util import iRange
9.  from mininet.topo import Topo
10.
11. setLogLevel( 'info' )
```

```
12.
13. REMOTE_CONTROLLER_IP="127.0.0.1"
14.
15. net = Containernet(controller=RemoteController,)
16.
17. info('*** Adding controller\n')
18. net.addController('c0',controller=RemoteController,
19.                     ip=REMOTE_CONTROLLER_IP,
20.                     port=6653)
21.
22. info('*** Adding switches\n')
23. s1 = net.addSwitch('s1',protocols='OpenFlow13')
24. s2 = net.addSwitch('s2',protocols='OpenFlow13')
25. s3 = net.addSwitch('s3',protocols='OpenFlow13')
26.
27. info('*** Adding Nat Host\n')
28. net.addNAT().configDefault()
29.
30. info('*** Adding docker containers\n')
31. d1 = net.addDocker('d1', dimage="servers:latest",cpuset_cpus="0,1",mem_limit='128m', mems
        wap_limit='0m')
32. d2 = net.addDocker('d2', dimage="servers:latest",cpuset_cpus="0,1",mem_limit='128m', mems
        wap_limit='0m')
33. d3 = net.addDocker('d3', dimage="servers:latest",cpuset_cpus="0,1",mem_limit='128m', mems
        wap_limit='0m')
34. d4 = net.addDocker('d4', dimage="servers:latest",cpuset_cpus="0,1",mem_limit='128m', mems
        wap_limit='0m')
35.
36. info('*** Creating links\n')
37. net.addLink(d1, s2, cls=TCLink, delay='0ms', bw=20)
38. net.addLink(d2, s2, cls=TCLink, delay='0ms', bw=20)
39. net.addLink(d3, s3, cls=TCLink, delay='0ms', bw=20)
40. net.addLink(d4, s3, cls=TCLink, delay='0ms', bw=20)
41. net.addLink(s1, s2, cls=TCLink, delay='0ms', bw=40)
42. net.addLink(s1, s3, cls=TCLink, delay='0ms', bw=40)
43.
44. info('*** Starting network\n')
45. net.start()
46.
47. info('*** Testing connectivity\n')
48. net.ping([d2, d1])
49. net.ping([d3, d1])
50. net.ping([d4, d1])
51.
52. info('*** Running CLI\n')
53. CLI(net)
54.
55. info('*** Stopping network')
56. net.stop()
```

Server Monitoring

```
1.  #!/usr/bin/python
2.
3.  import subprocess
4.  import json
5.  import threading, time
6.
7.  #s = sched.scheduler(time.time, time.sleep)
8.  timeSleep=1
9.  names={}
10. cpus={}
11. memorys={}
12. sumCpu={}
13. sumMemory={}
14. avgCpu={}
15. avgMemory={}
16. counter=0
17. results=[]
18. def getDockerStats():
19.     stats = subprocess.check_output('docker stats --no-stream --
    format \"{\\"container\\":\\"{{ .Container }}\\",\\"name\\":\\"{{ .Name }}\\",\\"memory\\":{\\"ra
    w\\":\\"{{ .MemUsage }}\\",\\"percent\\":\\"{{ .MemPerc }}\\"},\\"cpu\\":\\"{{ .CPUPerc }}\\"
    }'",shell=True)
20.     text_file = open("dockerStats.txt", "w")
21.     text_file.write(stats)
22.     text_file.close()
23.     #print(stats)
24.     i=0
25.     global counter
26.     counter+=1
27.     with open('dockerStats.txt') as f:
28.         for line in f:
29.             i+=1
30.             data = json.loads(line)
31.             names["name{0}".format(i)]=data["name"]
32.             if "cpu{0}".format(i) in cpus:
33.                 sumCpu["sumCpu{0}".format(i)]=sumCpu["sumCpu{0}".format(i)]+uToDouble(dat
    a["cpu"])
34.                 avgCpu["avgCpu{0}".format(i)] = round(sumCpu["sumCpu{0}".format(i)]/counter,
    2)
35.                 if(uToDouble(data["cpu"])>cpus["cpu{0}".format(i)]):
36.                     cpus["cpu{0}".format(i)]=uToDouble(data["cpu"])
37.             else:
38.                 cpus["cpu{0}".format(i)]=uToDouble(data["cpu"])
39.                 sumCpu["sumCpu{0}".format(i)]= uToDouble(data["cpu"])
40.                 avgCpu["avgCpu{0}".format(i)]= uToDouble(data["cpu"])
41.
42.             if "memory{0}".format(i) in memorys:
43.                 sumMemory["sumMemory{0}".format(i)]=sumMemory["sumMemory{0}".format(i
    )] + uToDouble(data["memory"]["percent"])
44.                 avgMemory["avgMemory{0}".format(i)]= round(sumMemory["sumMemory{0}".fo
    rmat(i)]/counter,2)
45.                 if(uToDouble(data["memory"]["percent"])>memorys["memory{0}".format(i)]):
46.                     memorys["memory{0}".format(i)]=uToDouble(data["memory"]["percent"])
```

```
47.             else:
48.                 memorys["memory{0}".format(i)]=uToDouble(data["memory"]["percent"])
49.                 sumMemory["sumMemory{0}".format(i)]= uToDouble(data["memory"]["percent"])

50.                 avgMemory["avgMemory{0}".format(i)]= uToDouble(data["memory"]["percent"])
51.     getValues(len(names))
52.
53. def getValues(i):
54.     j=i
55.     text_file = open("DockerStatsMaxValues.txt", "w")
56.     while j > 0:
57.         results = names["name{0}".format(j)]+", Max cpu:"+str(cpus["cpu{0}".format(j)])+"%,
    Max memory:"+str(memorys["memory{0}".format(j)])+"%, Average cpu:"+str(avgCpu["avgC
    pu{0}".format(j)])+"%, Average mem: "+str(avgMemory["avgMemory{0}".format(j)])+"%"
58.         text_file.write(results+"\n")
59.         print(results)
60.         j-=1
61.     text_file.close()
62.
63. def uToDouble(unic):
64.       r=float(unic.encode('ascii').replace("%",""))
65.       return r
66.
67.
68. while True:
69.     time.sleep(timeSleep)
70.     getDockerStats()
```

Κώδικας - Κλάση Monitoring

```
1.  package net.Floodlightcontroller.loadbalancer;
2.
3.  import java.io.BufferedReader;
4.  import java.io.FileNotFoundException;
5.  import java.io.FileReader;
6.  import java.io.IOException;
7.  import java.util.ArrayList;
8.
9.  import org.json.simple.JSONObject;
10. import org.json.simple.parser.JSONParser;
11. import org.json.simple.parser.ParseException;
12.
13. public class Monitoring {
14. private String host;
15. private double cpup;
16. private double memp;
17.
18.    public Monitoring(String name) {
19.        ArrayList<JSONObject> json=new ArrayList<JSONObject>();
20.        JSONObject obj;
21.        String fileName = "dockerStats.txt";
22.        String line = null;
23.        FileReader fileReader;
```

```
24.         try {
25.             fileReader = new FileReader(fileName);
26.             BufferedReader bufferedReader = new BufferedReader(fileReader);
27.             try {
28.                 while((line = bufferedReader.readLine()) != null) {
29.                     String cpu = null;
30.                     String memory = null;
31.                     obj = (JSONObject) new JSONParser().parse(line);
32.                     json.add(obj);
33.                 //Get containers Name
34.                     String containerName = (String)obj.get("name");
35.                 //Get cpu usage percent
36.                     if(containerName.equals(name)) {
37.                     cpu = (String)obj.get("cpu");
38.                 //Get memory usage percent
39.                     JSONObject resultObject = (JSONObject) obj.get("memory");
40.                     memory = (String)resultObject.get("percent");
41.                     //Convert results from string to double
42.                     memp = Double.parseDouble(memory=memory.replaceAll("\\D+",""))/10000;
43.                     cpup = Double.parseDouble(cpu=cpu.replaceAll("\\D+",""))/10000;
44.                     System.out.println(containerName+": cpu:"+
45.                                     cpup+", memory:"+memp);
46.                     }
47.
48.     }
49.             } catch (IOException e) {
50.                 // TODO Auto-generated catch block
51.                 e.printStackTrace();
52.             } catch (ParseException e) {
53.                 // TODO Auto-generated catch block
54.                 e.printStackTrace();
55.             }
56.             // Always close files.
57.         try {
58.             bufferedReader.close();
59.         } catch (IOException e) {
60.             // TODO Auto-generated catch block
61.             e.printStackTrace();
62.         }
63.         } catch (FileNotFoundException e) {
64.             // TODO Auto-generated catch block
65.             e.printStackTrace();
66.         }
67.
68.
69.     }
70.
71.     public double getCpup() {
72.         return cpup;
73.     }
74.
75.     public double getMemp() {
76.         return memp;
77.     }
```

Ενεργοποίηση AALB

```
1.  //Enable Application Aware Load Balancing
2.          }else if(lbMethod == AALB && !membersBandwidth.isEmpty()){
3.              logger.info("Member picked using AALB");
4.              ArrayList<String> poolMembersId = new ArrayList<String>();
5.              for (String memberId: membersMonitoringValues.keySet()){
6.                  for (int i=0;i<members.size();i++){
7.                      if(members.get(i).equals(memberId)){
8.                          poolMembersId.add(memberId);
9.                      }
10.                 }
11.             }
12. //Return the member which has the minimum monitoring usage, out of this pool members
13.             if(!poolMembersId.isEmpty()){
14.                 ArrayList<Double> monitoringMetricValues = new ArrayList<Double>();
15.
16.                 for(int j=0;j<poolMembersId.size();j++){
17.                     monitoringMetricValues.add(membersMonitoringValues.get(poolMembersId.get(j)))
                ;
18.                 }
19.
20.                 logger.info("Member picked using LB : {}", poolMembersId.get(monitoringMetricVal
                ues.indexOf(Collections.min(monitoringMetricValues))));
21.                 logger.info(poolMembersId.get(monitoringMetricValues.indexOf(Collections.min(mon
                itoringMetricValues))));
22.                 return poolMembersId.get(monitoringMetricValues.indexOf(Collections.min(monitori
                ngMetricValues)));
23.             }
24.             return null
```

Συλλογή

```
1.    // Collect member host cpu and memory usage percent
2.
3.      public HashMap<String, Double> collectMonitoringMetrics(String poolid){
4.          double memberHostCpuP, memberHostMemP,memberHostBandwidthP = 0,metricValue;
5.          HashMap<String,U64> memberPortBandwidth = new HashMap<String, U64>();
6.          HashMap<String,Double> memberMonitorMetrics = new HashMap<String, Double>();
7.
8.          //Collect members bandwidth
9.          memberPortBandwidth = collectSwitchPortBandwidth(poolid);
10.         for (LBMember member: members.values()) {
11.             if(member.getPoolId().equals(poolid)) {
12.             Monitoring mon = new Monitoring(member.host);
13.             memberHostCpuP = mon.getCpup();
14.             memberHostMemP = mon.getMemp();
15.             if(memberPortBandwidth != null && !memberPortBandwidth.isEmpty()) {
16.                 if (memberPortBandwidth.get(member.id) == null) {
17.                     memberHostBandwidthP = 0;
18.                 }
19.                 else {
20.                     memberHostBandwidthP = (((memberPortBandwidth.get(member.id).getBigInteger()
    ).doubleValue()))/linkBandwidthLimit;
```

```
21.                 }
22.             }
23.             metricValue = memberHostBandwidthP*0.5+memberHostCpuP*0.1+memberHostMemP*
    0.4;
24.             log.info(memberHostBandwidthP+" "+memberHostCpuP+" "+memberHostMemP);
25.             //log.info(String.valueOf(metricValue));
26.             memberMonitorMetrics.put(member.id, metricValue);
27.             }
28.         }
29.
30.         return memberMonitorMetrics;
31.     }
```

Αποσφαλμάτωση

```
1.  // Debugging Add argument <poolid> to recognize lb_method among many methods
2.  //for the same devices(members,hosts)
3.
4.     public HashMap<String, U64> collectSwitchPortBandwidth(String poolid){
5.         HashMap<String,U64> memberPortBandwidth = new HashMap<String, U64>();
6.         HashMap<IDevice,String> deviceToMemberId = new HashMap<IDevice, String>();
7.
8.         // retrieve all known devices to know which ones are attached to the members
9.         Collection<? extends IDevice> allDevices = deviceManagerService.getAllDevices();
10.
11.        for (IDevice d : allDevices) {
12.            for (int j = 0; j < d.getIPv4Addresses().length; j++) {
13.                if(members != null){
14.                    for(LBMember member: members.values()){
15.                      if(member.getPoolId().equals(poolid)) {
16.                        if (member.address == d.getIPv4Addresses()[j].getInt()) {
17.                            deviceToMemberId.put(d, member.id);
18.                            }
19.                        }
20.                    }
21.                }
22.            }
23.        }
24.        // collect statistics of the switch ports attached to the members
25.        if(deviceToMemberId !=null){
26.            for(IDevice membersDevice: deviceToMemberId.keySet()){
27.                String memberId = deviceToMemberId.get(membersDevice);
28.                for(SwitchPort dstDap: membersDevice.getAttachmentPoints()){
29.                    SwitchPortBandwidth bandwidthOfPort = statisticsService.getBandwidthConsumptio
    n(dstDap.getNodeId(), dstDap.getPortId());
30.                    if(bandwidthOfPort != null) // needs time for 1st collection, this avoids nullPointerEx
    ception
31.                        memberPortBandwidth.put(memberId, bandwidthOfPort.getBitsPerSecondRx());
32.                        String testBandwidthofPort = (bandwidthOfPort.getBitsPerSecondRx()).getBigInte
    ger().toString();
33.                }
34.            }
35.        }
36.    return memberPortBandwidth;
```

Παράρτημα Β - Εντολές

Simple Round Robin

curl -X POST -d
'{"id":"2","name":"vip2","protocol":"tcp","address":"10.0.0.200","port":"8080"}'
http://localhost:8080/quantum/v1.0/vips/
curl -X POST -d '{"id":"2","name":"pool2","protocol":"tcp","vip_id":"2"}'
http://localhost:8080/quantum/v1.0/pools/
curl -X POST -d '{"id":"21","address":"10.0.0.2","port":"8080","pool_id":"2"}'
http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"22","address":"10.0.0.3","port":"8080","pool_id":"2"}'
http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"23","address":"10.0.0.4","port":"8080","pool_id":"2"}'
http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"24","address":"10.0.0.5","port":"8080","pool_id":"2"}'
http://localhost:8080/quantum/v1.0/members/

curl -X POST -d
'{"id":"3","name":"vip2","protocol":"tcp","address":"10.0.0.200","port":"80"}'
http://localhost:8080/quantum/v1.0/vips/
curl -X POST -d '{"id":"3","name":"pool2","protocol":"tcp","vip_id":"3"}'
http://localhost:8080/quantum/v1.0/pools/
curl -X POST -d '{"id":"31","address":"10.0.0.2","port":"80","pool_id":"3"}'
http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"32","address":"10.0.0.3","port":"80","pool_id":"3"}'
http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"33","address":"10.0.0.4","port":"80","pool_id":"3"}'
http://localhost:8080/quantum/v1.0/members/
curl -X POST -d '{"id":"34","address":"10.0.0.5","port":"80","pool_id":"3"}'
http://localhost:8080/quantum/v1.0/members/

Statistics

#----------ENABLE FLOODLIGHT STATISTICS----------

#curl -X POST 127.0.0.1:8080/wm/statistics/config/enable/json

#-------- CREATE VIRTUAL IPs AND HOST IP POOLS------

#VIDEO SERVERS

curl -X POST -d
'{"id":"2","name":"vip2","protocol":"tcp","address":"10.0.0.200","port":"8080"}'
http://localhost:8080/quantum/v1.0/vips/

curl -X POST -d
'{"id":"2","name":"pool2","protocol":"tcp","lb_method":"STATISTICS","vip_id":"2"}'
http://localhost:8080/quantum/v1.0/pools/

curl -X POST -d '{"id":"21","address":"10.0.0.2","port":"8080","pool_id":"2"}'
http://localhost:8080/quantum/v1.0/members/

curl -X POST -d '{"id":"22","address":"10.0.0.3","port":"8080","pool_id":"2"}'
http://localhost:8080/quantum/v1.0/members/

curl -X POST -d '{"id":"23","address":"10.0.0.4","port":"8080","pool_id":"2"}'
http://localhost:8080/quantum/v1.0/members/

curl -X POST -d '{"id":"24","address":"10.0.0.5","port":"8080","pool_id":"2"}'
http://localhost:8080/quantum/v1.0/members/

#WEB SERVERS

curl -X POST -d
'{"id":"3","name":"vip3","protocol":"tcp","address":"10.0.0.200","port":"80"}'
http://localhost:8080/quantum/v1.0/vips/

curl -X POST -d
'{"id":"3","name":"pool3","protocol":"tcp","lb_method":"STATISTICS","vip_id":"3"}'
http://localhost:8080/quantum/v1.0/pools/

curl -X POST -d '{"id":"31","address":"10.0.0.2","port":"80","pool_id":"3"}'
http://localhost:8080/quantum/v1.0/members/

curl -X POST -d '{"id":"32","address":"10.0.0.3","port":"80","pool_id":"3"}'
http://localhost:8080/quantum/v1.0/members/

curl -X POST -d '{"id":"33","address":"10.0.0.4","port":"80","pool_id":"3"}'
http://localhost:8080/quantum/v1.0/members/

curl -X POST -d '{"id":"34","address":"10.0.0.5","port":"80","pool_id":"3"}'
http://localhost:8080/quantum/v1.0/members/

Παράρτημα Γ - Αποτελέσματα

Σενάριο 2
Simple Round Robin

Web Server CPU/Mem usage						Video Server CPU/Mem usage				
Requests	**Server Name**	**Max cpu**	**Max memory**	**Avg cpu**	**Avg mem**	**Server Name**	**Max cpu**	**Max memory**	**Avg cpu**	**Avg mem**
	mn.d1	2,41%	35,60%	1,01%	35,38%	mn.d1	0,67%	40,88%	0,35%	32,32%
	mn.d2	2,50%	36,17%	1,05%	33,86%	mn.d2	1,64%	42,31%	0,41%	36,94%
	mn.d3	3,53%	33,33%	1,03%	33,16%	mn.d3	2,02%	32,28%	0,24%	31,21%
100	mn.d4	3,60%	33,27%	1,02%	33,11%	mn.d4	1,28%	32,27%	0,21%	31,59%
	mn.d1	3,34%	33,48%	1,88%	33,30%	mn.d1	2,08%	50,50%	0,43%	39,09%
	mn.d2	4,01%	33,43%	1,98%	33,19%	mn.d2	1,73%	35,78%	0,40%	29,96%
	mn.d3	3,53%	33,48%	1,92%	33,30%	mn.d3	2,15%	38,96%	0,41%	30,97%
200	mn.d4	3,43%	33,33%	1,96%	33,19%	mn.d4	2,04%	38,76%	0,43%	33,75%
	mn.d1	10,24%	33,80%	3,47%	33,53%	mn.d1	1,93%	40,00%	0,46%	33,60%
	mn.d2	10,13%	33,69%	3,43%	33,44%	mn.d2	1,54%	52,73%	0,43%	44,15%
	mn.d3	10,16%	33,92%	3,41%	33,78%	mn.d3	1,94%	49,58%	0,49%	38,49%
400	mn.d4	10,31%	33,99%	3,39%	33,81%	mn.d4	0,89%	35,95%	0,43%	31,16%

Statistics

Web Server CPU/Mem usage						Video Server CPU/Mem usage				
Requests	**Server Name**	**Max cpu**	**Max memory**	**Avg cpu**	**Avg mem**	**Server Name**	**Max cpu**	**Max memory**	**Avg cpu**	**Avg mem**
	mn.d1	0,22%	33,78%	0,17%	33,78%	mn.d1	2,21%	35,16%	0,40%	29,72%
	mn.d2	8,11%	34,82%	3,54%	34,39%	mn.d2	2,01%	32,29%	0,35%	28,02%
	mn.d3	0,28%	32,76%	0,19%	33,61%	mn.d3	2,05%	40,35%	0,34%	29,89%
100	mn.d4	0,23%	33,31%	0,18%	33,29%	mn.d4	2,20%	32,10%	0,39%	28,32%
	mn.d1	0,28%	33,78%	0,17%	33,75%	mn.d1	1,76%	45,87%	0,36%	39,76%
	mn.d2	15,02%	35,46%	3,73%	35,22%	mn.d2	2,19%	42,73%	0,43%	36,58%
	mn.d3	0,28%	32,73%	0,20%	28,82%	mn.d3	1,72%	45,45%	0,33%	39,33%
200	mn.d4	0,71%	33,35%	0,25%	33,31%	mn.d4	1,56%	44,04%	0,37%	36,66%
	mn.d1	18,40%	36,85%	2,81%	30,40%	mn.d1	1,64%	40,55%	0,41%	34,55%
	mn.d2	1,58%	31,83%	0,23%	31,80%	mn.d2	1,98%	42,45%	0,39%	31,04%
	mn.d3	26,45%	38,76%	3,82%	36,20%	mn.d3	1,81%	44,42%	0,36%	33,40%
400	mn.d4	27,95%	36,22%	5,47%	35,66%	mn.d4	1,74%	35,88%	0,44%	30,66%

AALB

Web Server CPU/Mem usage						Video Server CPU/Mem usage				
Threads	**Server Name**	**Max cpu**	**Max memory**	**Avg cpu**	**Avg mem**	**Server Name**	**Max cpu**	**Max memory**	**Avg cpu**	**Avg mem**
	mn.d1	5,31%	38,19%	0,43%	35,54%	mn.d1	1,29%	42,18%	0,33%	33,22%
	mn.d2	8,16%	35,58%	1,66%	31,30%	mn.d2	2,13%	35,19%	0,42%	30,85%
	mn.d3	9,48%	34,06%	1,04%	33,71%	mn.d3	0,56%	40,99%	0,32%	35,71%
100	mn.d4	8,02%	36,67%	1,03%	33,98%	mn.d4	1,81%	38,25%	0,38%	33,07%
	mn.d1	15,90%	37,92%	1,78%	35,67%	mn.d1	1,59%	40,44%	0,41%	31,75%
	mn.d2	14,99%	35,36%	2,14%	30,75%	mn.d2	0,70%	45,99%	0,29%	36,17%
	mn.d3	14,67%	33,37%	1,56%	33,15%	mn.d3	1,65%	40,06%	0,37%	31,08%
200	mn.d4	17,94%	36,42%	1,71%	33,83%	mn.d4	1,85%	41,20%	0,38%	31,58%
	mn.d1	29,16%	38,10%	2,38%	35,74%	mn.d1	2,31%	42,19%	0,44%	35,57%
	mn.d2	28,87%	36,37%	2,90%	34,57%	mn.d2	3,06%	44,02%	0,44%	37,24%
	mn.d3	28,64%	38,03%	2,91%	34,88%	mn.d3	2,22%	44,17%	0,44%	36,87%
400	mn.d4	29,86%	38,24%	3,39%	34,89%	mn.d4	1,45%	44,12%	0,42%	33,17%