Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ελεγκτής OpenFlow για Υπηρεσίες με Επίγνωση Περιβάλλοντος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ουρανία Στέλλα Σ. Βουκελάτου

Επιβλέπων : Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2015

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ελεγκτής OpenFlow για Υπηρεσίες με Επίγνωση Περιβάλλοντος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ουρανία Στέλλα Σ. Βουκελάτου

Επιβλέπων : Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την …….

.....	.....	.....
Ευστάθιος Συκάς	Μιλτιάδης Αναγνώστου	Γεώργιος Στασινόπουλος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2015

.....
Ουρανία Στέλλα Σ. Βουκελάτου

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη και η παρουσίαση ενός μηχανισμού που εφαρμόζεται σε έναν κεντρικό ελεγκτή δικτύου υπολογιστών, προκειμένου να επιτευχθεί η βέλτιστη δρομολόγηση των δεδομένων στο δίκτυο με την συνεχή εναλλαγή της λειτουργίας των συσκευών του δικτύου και την επίγνωση των στοιχείων περιβάλλοντος. Η επιλογή της κατάλληλης διαδρομής γίνεται με βάση την τοπολογία και τις αποφάσεις του διαχειριστή του δικτύου.

Για την ανάπτυξη του μηχανισμού αυτού, χρησιμοποιήθηκε ένα πρωτόκολλο επικοινωνιών που ονομάζεται OpenFlow. Το πρωτόκολλο αυτό αντιπροσωπεύει μία νέα αρχιτεκτονική δικτύου που διαχωρίζει τον μηχανισμό προώθησης των δεδομένων από τον μηχανισμό ελέγχου για να υλοποιηθούν από διαφορετικά στοιχεία δικτύου. Βασικό συστατικό είναι ο κεντρικοποιημένος ελεγκτής που συντηρεί και δημιουργεί τον πίνακα δρομολόγησης των ροών κάθε μεταγωγέα του δικτύου. Συγκεκριμένα, η υλοποίηση του μηχανισμού γίνεται με την χρήση ενός βασικού ελεγκτή πρωτοκόλλου OpenFlow, του ελεγκτή Beacon.

Επιπλέον, χρησιμοποιήθηκε το λογισμικό Mininet. Το Mininet αποτελεί έναν εξομοιωτή δικτύου. Έχει την δυνατότητα δημιουργίας μιας εικονικής τοπολογίας δικτύου, όπου εκτελούνται ταυτόχρονα τερματικά, μεταγωγείς, δρομολογητές, ελεγκτές μέσα σε έναν ενιαίο πυρήνα (kernel) Linux.

Τέλος, χρησιμοποιήθηκε η μέθοδος επικοινωνίας Anycast. Πρόκειται για μία μέθοδο δρομολόγησης πακέτων βασισμένη στην εύρεση της συντομότερης διαδρομής όταν πρόκειται για δρομολόγηση προς μία διεύθυνση IP, που την μοιράζεται μία συγκεκριμένη ομάδα συσκευών δικτύου.

Λέξεις Κλειδιά : πρωτόκολλο OpenFlow, Beacon controller, Anycast, IPv4, πίνακας ροών, context-aware δρομολόγηση, Mininet, Linux.

Abstract

The objective of the present thesis is the development and presentation of a mechanism that is applied to a central computer network controller, in order to establish the best data routing in the network, by constantly switching the operation of the network devices and being aware of the context elements. Choosing the appropriate route is based on the topology and the decisions made by the network administrator.

For the development of this mechanism, a communications protocol was used, called OpenFlow. This protocol represents a new network architecture which separates the data routing mechanism from the control mechanism, for them to be implemented by different network elements. The basic component is the centralized controller that sustains and creates the flow routing table of every switch in the network. Specifically, the implementation of this mechanism is completed using a basic OpenFlow protocol controller, Beacon controller.

Furthermore, the Mininet software was used. Mininet is a network emulator. It has the ability of creating a virtual network topology, where hosts, switches, routers, controllers are executed simultaneously on a single Linux kernel.

Finally, the Anycast method of communication was used. It is a data routing method, based on finding the shortest route when it concerns routing to an IP address, shared by a particular group of network devices.

Key Words: OpenFlow protocol, Beacon controller, Anycast, IPv4, flow table, context – aware routing, Mininet, Linux.

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε κατά το ακαδημαϊκό έτος 2014-2015 στον τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου.

Υπεύθυνος κατά την εκπόνηση της διπλωματικής ήταν ο καθηγητής κ. Ευστάθιος Συκάς στον οποίο οφείλω ιδιαίτερες ευχαριστίες για την ανάθεση αυτής και την δυνατότητα που μου δόθηκε να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα. Θα ήθελα επίσης να ευχαριστήσω θερμά τον υποψήφιο διδάκτορα κ. Πάρη Χαραλάμπου για την υποστήριξη και την καθοδήγηση που μου παρείχε κατά την συγγραφή της εργασίας.

Κυρίως όμως θέλω να εκφράσω την ευγνωμοσύνη μου στην οικογένεια μου για την αμέριστη υποστήριξη τους όλα αυτά τα χρόνια. Τέλος, θα ήταν παράλειψη να μην ευχαριστήσω όλους τους δικούς μου ανθρώπους για την διάθεση και την κατανόηση που επέδειξαν σε όλη μου την πορεία.

Ουρανία Στέλλα Σ. Βουκελάτου

Φεβρουάριος 2015

Πίνακας περιεχομένων

Περιεχόμενα Εικόνων

1 ΕΙΣΑΓΩΓΗ

1.1Σκοπός διπλωματικής εργασίας

Η έκρηξη των κινητών συσκευών, το εικονικό περιβάλλον των διακομιστών και η εμφάνιση των υπηρεσιών συννέφου (cloud services) έχουν οδηγήσει την βιομηχανία δικτύωσης να επανεξετάσει τις αρχιτεκτονικές δικτύου. Οι παραδοσιακές αρχιτεκτονικές δικτύου θεωρούνται ακατάλληλες για να αντιμετωπίσουν τις σημερινές ανάγκες των υπηρεσιών, των διακομιστών και των χρηστών.

Ανανεώνοντας τους μεταγωγείς ώστε να διαχωριστούν τα επίπεδα ελέγχου και δεδομένων, ένας κεντρικός ελεγκτής δημιουργεί βελτιστοποιημένες διαδρομές για την προώθηση της κίνησης. Έτσι, αναπτύχθηκε μία νέα αρχιτεκτονική, η Software Defined Networking (SDN) η οποία επιτρέπει σε κέντρα δεδομένων και σε ερευνητές να καινοτομήσουν στα δίκτυα και να χειριστούν την δυναμική φύση των δικτύων υπολογιστών.

Από την άλλη, οι υπηρεσίες Anycast είναι ένας νέος, μοναδικός τρόπος διευθυνσιοδότησης των αυτόνομων συστημάτων στο πρωτόκολλο IP. Η ανάγκη που οδήγησε στην ανάπτυξη της μεθόδου επικοινωνίας Anycast βρισκόταν στην αύξηση της λειτουργικότητας που ήταν δύσκολα υλοποιήσιμη στο TCP/IP. Στην δρομολόγηση, η Anycast επιτρέπει στα πακέτα την προώθηση στον κοντινότερο δρομολογητή μιας ομάδας ισοδύναμων δρομολογητών, και έτσι επιτρέπεται η καλύτερη κατανομή του φορτίου μεταξύ των δρομολογητών και η δυναμική ευελιξία στην περίπτωση που κάποιοι δρομολογητές παραμείνουν αδρανείς.

Η παρούσα διπλωματική εργασία έχει ως σκοπό την ανάπτυξη ενός μηχανισμού που θα συνδυάζει την αρχιτεκτονική SDN με την μέθοδο επικοινωνίας Anycast και θα ελέγχει δυναμικά την δρομολόγηση των πακέτων. Έτσι, θα εξασφαλίζει την βέλτιστη

δρομολόγηση μέσα σε ένα αυτόνομο σύστημα. Για την ανάπτυξη του μηχανισμού χρησιμοποιήθηκε το πρωτόκολλο OpenFlow που βασίζεται στην αρχιτεκτονική SDN και το λογισμικό Mininet, παρέχοντας την δυνατότητα δημιουργίας ενός εικονικού δικτύου, στο οποίο θα δρομολογηθούν πακέτα από τους μεταγωγείς OpenFlow με τον βέλτιστο τρόπο.

1.2Δομή Διπλωματικής Εργασίας

Η διπλωματική εργασία αποτελείται από έξι κεφάλαια. Στο πρώτο κεφάλαιο, αναπτύχθηκαν οι ανάγκες που οδήγησαν στην δημιουργία της διπλωματικής εργασίας και η οργάνωση του κειμένου.

Στο δεύτερο κεφάλαιο, παρουσιάζεται η αρχιτεκτονική δικτύου Software Defined Networking (SDN) και η δομή της. Στην συνέχεια, αναλύεται το πρωτόκολλο OpenFlow που υποστηρίζει την συγκεκριμένη αρχιτεκτονική, παρουσιάζονται εν συντομία οι πιο γνωστοί ελεγκτές που έχουν αναπτυχθεί στο πλαίσιο του πρωτοκόλλου, ενώ στο τέλος αναπτύσσεται η δομή του μεταγωγέα OpenFlow.

Στο τρίτο κεφάλαιο, αναλύεται ο ελεγκτής Beacon που χρησιμοποιείται για την παρούσα διπλωματική εργασία. Πιο συγκεκριμένα, αναφέρονται οι ανάγκες που οδήγησαν στην ανάπτυξη του, τα πλεονεκτήματα του συγκριτικά με άλλους γνωστούς ελεγκτές πρωτοκόλλου OpenFlow, όπως επίσης αναλύεται η δομή και η αρχιτεκτονική του.

Στο τέταρτο κεφάλαιο, αναλύεται το λογισμικό Mininet που χρησιμοποιήθηκε για την δημιουργία της εικονικής τοπολογίας του δικτύου μας. Παρουσιάζονται τα πλεονεκτήματα , οι δυνατότητες και η δομή του λογισμικού.

Στο πέμπτο κεφάλαιο, περιγράφονται εν συντομία οι συνήθεις μέθοδοι επικοινωνίας του πρωτοκόλλου IPv4 και παρουσιάζεται περιεκτικά η μέθοδος επικοινωνίας Anycast. Αναφέρονται τα πλεονεκτήματα της τεχνικής και στην συνέχεια αναλύεται η υλοποίηση της υπηρεσίας Anycast στο πρωτόκολλο OpenFlow.

Στο έκτο κεφάλαιο, αναλύεται ο μηχανισμός που αναπτύχθηκε για την δρομολόγηση των πακέτων με στόχο την δημιουργία καταχωρήσεων στους πίνακες ροής των μεταγωγέων χρησιμοποιώντας την τεχνική Anycast και την δρομολόγηση μέσω της συντομότερης διαδρομής. Αναλύεται η διαδικασία που ακολουθήθηκε, παρουσιάζεται η τοπολογία που χρησιμοποιήθηκε και υλοποιήθηκε από το λογισμικό Mininet και εξηγείται η διαδικασία της βέλτιστης δρομολόγησης Context-Aware. Τέλος, αναφέρονται τα στοιχεία που συμβάλλουν στην σημαντικότητα της αρχιτεκτονικής SDN στα δίκτυα υπολογιστών.

2 OpenFlow

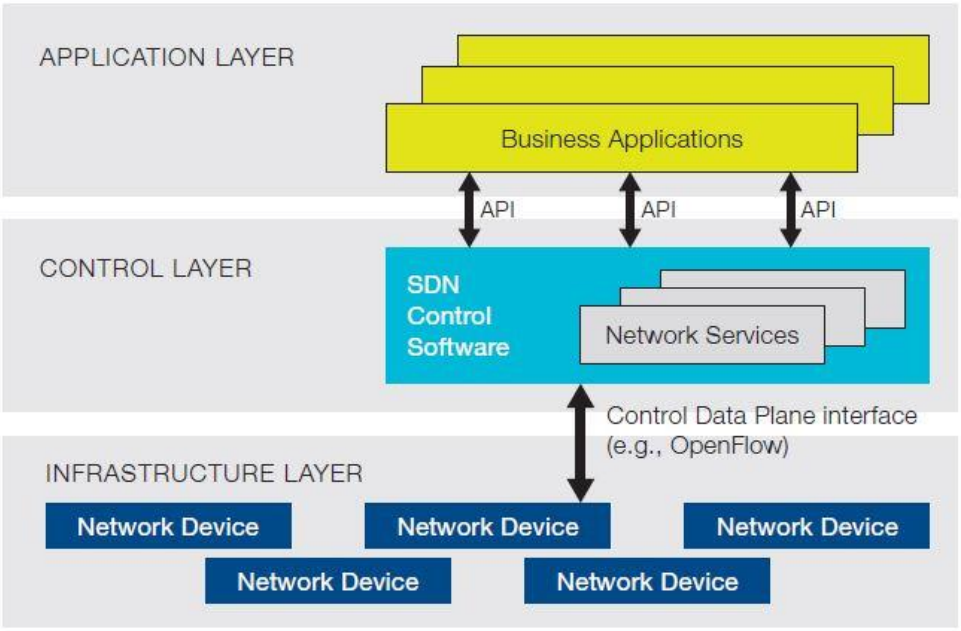
2.1 Software Defined Networking (SDN)

Στις μέρες μας, το διαδίκτυο είναι ένα σημαντικό κομμάτι της καθημερινής ζωής. Πίσω από όλες τις εταιρείες πολλών δισεκατομμυρίων δολαρίων, όπως η Google, Amazon, Facebook, βρίσκονται δίκτυα που πρέπει να προσαρμοστούν στις συνεχώς μεταβαλλόμενες ανάγκες, χωρίς να απαιτείται μεγάλη αλλαγή σε λογισμικό και υλικό. Αυτό έχει ως αποτέλεσμα μια πιο ευέλικτη και αυτόματη δικτύωση. Αυτό «υπόσχεται» να το επιτρέψει το SDN προκειμένου να καλυφθούν οι ανάγκες των εφαρμογών και των επιχειρήσεων.

Το SDN επιτρέπει τις εφαρμογές να έχουν γνώση του δικτύου προσεγγίζοντας με έναν καινούριο τρόπο την αρχιτεκτονική των δικτύων. Σε ένα παραδοσιακό δίκτυο, ο δικτυακός εξοπλισμός, όπως ένας μεταγωγέας ή ένας δρομολογητής, περιέχουν το επίπεδο ελέγχου και το επίπεδο δεδομένων. Το επίπεδο ελέγχου διευκρινίζει την διαδρομή την οποία θα ακολουθήσουν τα πακέτα μέσα στο δίκτυο, ενώ το επίπεδο δεδομένων είναι το κομμάτι του δικτύου που φέρει τα πακέτα. [1]

Η νοημοσύνη του δικτύου βρίσκεται κεντρικά σε βασιζόμενους σε λογισμικό ελεγκτές SDN, που διατηρούν μια γενική άποψη του δικτύου. Αυτό έχει ως αποτέλεσμα το δίκτυο να εμφανίζεται σε εφαρμογές ως ένας λογικός δρομολογητής. Με το SDN, οι επιχειρήσεις και οι μεταφορείς αποκτούν έλεγχο σε ολόκληρο το δίκτυο από ένα λογικό σημείο, το οποίο απλουστεύει σημαντικά τον σχεδιασμό δικτύου και την λειτουργία του. Επίσης, το SDN απλουστεύει σημαντικά τις συσκευές δικτύου, καθώς δεν χρειάζεται πλέον να κατανοήσουν και να επεξεργαστούν χιλιάδες πρότυπα πρωτοκόλλου, αλλά να δεχτούν εντολές από τους ελεγκτές SDN. [3]

Η τεχνολογία του SDN διαχωρίζει το δίκτυο σε 3 επίπεδα: εφαρμογών, ελέγχου και δεδομένων ή υποδομών, όπως φαίνεται και από την εικόνα.



Εικόνα 2.1 Η αρχιτεκτονική SDN 3 επιπέδων [3]

Πιο συγκεκριμένα, το επίπεδο εφαρμογών περιέχει εφαρμογές SDN και επικοινωνούν μέσω διεπαφής προγραμματισμού εφαρμογών (Application Programming Interface, API). Οι εφαρμογές SDN είναι λογισμικά τα οποία απευθείας επικοινωνούν τις ανάγκες δικτύου τους και την επιθυμητή δικτυακή συμπεριφορά με τον ελεγκτή SDN. Το επίπεδο ελέγχου, ή ελεγκτής SDN, είναι ουσιαστικά ένα λειτουργικό σύστημα δικτύου, το οποίο μεταβιβάζει τις ανάγκες από το επίπεδο εφαρμογών στο επίπεδο δεδομένων και παρουσιάζει έναν λογικό χάρτη του δικτύου στις εφαρμογές SDN που υλοποιούνται πάνω σε αυτό. Οι αποφάσεις λαμβάνονται με μια γενική άποψη ολόκληρου του δικτύου και όχι με την περιορισμένη ορατότητα των γειτονικών δικτυακών συσκευών, όπως κάνουν οι δρομολογητές σήμερα. Τέλος, στο επίπεδο δεδομένων ουσιαστικά πραγματοποιείται η κίνηση και η προώθηση των πακέτων. [4]

2.2 Πρωτόκολλο OpenFlow

Το προγραμματιζόμενο πρωτόκολλο δικτύου OpenFlow, αποτέλεσμα ακαδημαϊκής έρευνας είναι η πρώτη πρότυπη διεπαφή επικοινωνιών μεταξύ του επιπέδου ελέγχου και του επιπέδου προώθησης σε μια αρχιτεκτονική SDN.

Επιτρέπει την άμεση πρόσβαση και διαχείριση της κίνησης των δεδομένων του επιπέδου προώθησης των συσκευών δικτύου (δρομολογητών, μεταγωγέων, επαναληπτών), εικονικά και φυσικά. Εφαρμόζεται στις δύο πλευρές της διεπαφής μεταξύ των συσκευών υποδομής δικτύου και του λογισμικού ελέγχου SDN.

Χρησιμοποιεί τους πίνακες ροής (flow tables) για την αναγνώριση της κίνησης δικτύου που βασίζεται σε «κανόνες» που έχουν προγραμματιστεί δυναμικά ή στατικά από το λογισμικό ελέγχου SDN. Επίσης, επιτρέπει την κατεύθυνση της κίνησης ορίζοντας παραμέτρους όπως μοτίβα χρήσης και εφαρμογές. Εφόσον στο πρωτόκολλο OpenFlow το δίκτυο προγραμματίζεται με βάση τις ροές, μια αρχιτεκτονική SDN – OpenFlow παρέχει εξαιρετικά λεπτομερή έλεγχο, επιτρέποντας στο δίκτυο να απαντήσει σε αλλαγές σε πραγματικό χρόνο στα επίπεδα της εφαρμογής, χρήστη και συνόδου. [3]

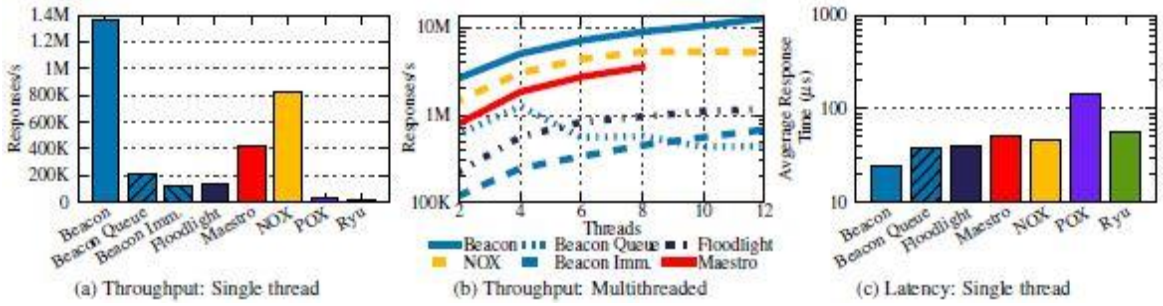
2.3 Το Δίκτυο

Ένα OpenFlow δίκτυο αποτελείται από κάποιες βασικές συνιστώσες σε επίπεδο δεδομένων και ελέγχου. Ένας μεταγωγέας είναι το βασικό δομικό στοιχείο ενός δικτύου OpenFlow: μπορεί να είναι ένας εμπορικός υλικός μεταγωγέας ή μια εικονική υλοποίηση. Από την άλλη, το επίπεδο ελέγχου αποτελείται από τους Ελεγκτές OpenFlow (OpenFlow Controllers).

Ο μεταγωγέας OpenFlow αποτελείται από πίνακες ροής, οι οποίοι χρησιμοποιούνται για την αναζήτηση και προώθηση πακέτων από ένα ασφαλές κανάλι που συνδέεται με τον ελεγκτή OpenFlow. Μια εικονική υλοποίηση δρομολογητή είναι ο OpenVSwitch (OVS).

Σε ένα δίκτυο, μπορεί να υπάρχουν ένας ή περισσότεροι ελεγκτές. Ο ελεγκτής εκτελεί τις διεργασίες ελέγχου του δικτύου OpenFlow, παρέχει διεπαφή για την διαχείριση και κατεύθυνση των πινάκων ροών των συσκευών που ελέγχει. Ακόμα, στέλνει στις συσκευές προώθησης καταχωρήσεις ροής (flow entries), με βάση τις οποίες γίνεται η δρομολόγηση και η προώθηση δεδομένων. Οι ροές δεδομένων δημιουργούνται δηλαδή ανάλογα με την ζήτηση την κάθε χρονική στιγμή, ενώ ο ελεγκτής προσφέρει δυναμική ανάθεση πόρων.

Υπάρχουν αρκετοί ελεγκτές OpenFlow ανοιχτού λογισμικού, σχεδόν για κάθε αναπτυξιακό περιβάλλον, όπως ο NOX σε C, οι POX, RYU σε Python, οι Floodlight, Beacon σε Java και πολλοί ακόμα. Σε αυτήν την διπλωματική εργασία χρησιμοποιήθηκε ο ελεγκτής Beacon, όπως θα αναλυθεί παρακάτω.



Εικόνα 2.2 Σύγκριση διαφόρων ελεγκτών όσον αφορά την απόδοση και την καθυστέρηση[7]

2.3.1 Ο ελεγκτής NOX

Ο NOX ήταν η πρώτη αναφορική εφαρμογή ενός ελεγκτή OpenFlow. Η συγκεκριμένη υλοποίηση, αναφέρεται ως NOX Classic, ήταν προγραμματισμένος σε Python και C++. Η σημερινή εφαρμογή είναι μόνο σε C++ και προσφέρει μια υλοποίηση ενός API σε C++ του OpenFlow 1.0, το οποίο προσφέρει γρήγορες και ασύγχρονες εισόδους – εξόδους (I/O). Είναι προσανατολισμένος για λειτουργία σε σύγχρονα συστήματα Linux και περιλαμβάνει έτοιμα υποπρογράμματα(components) για λειτουργίες όπως η δρομολόγηση πακέτων (routing) και η ανίχνευση τοπολογίας δικτύου (topology discovery).

2.3.2 Ο ελεγκτής POX

Ο ελεγκτής POX είναι η βελτιωμένη εξέλιξη της διεπαφής OpenFlow σε Python του NOX. Εκτός από την υλοποίηση του OpenFlow API σε Python, ο POX προσφέρει έτοιμα components όπως και ο NOX: λειτουργεί σε πολλές πλατφόρμες όπως Linux, Mac OS, Windows και υποστηρίζει το ίδιο γραφικό περιβάλλον χρήστη και ίδια εργαλεία οπτικοποίησης όπως ο NOX.

2.3.3 Ο ελεγκτής Floodlight

Ο ελεγκτής Floodlight είναι μια υψηλής επίδοσης υλοποίηση OpenFlow σε Java, η οποία δουλεύει σε πλαίσια πρωτοβουλίας πύλης ανοιχτής υπηρεσίας (Open Service Gateway initiative – OSGi) όπως τα Equinox, SpringSource. Ο συγκεκριμένος ελεγκτής προκύπτει από τον ελεγκτή Beacon, αποτελείται από συλλογή λειτουργικών μονάδων με έτοιμα υποπρογράμματα διαφορετικών λειτουργιών δικτύου, το οποίο τον καθιστά εύκολο στην κατανόηση για έναν νέο προγραμματιστή.

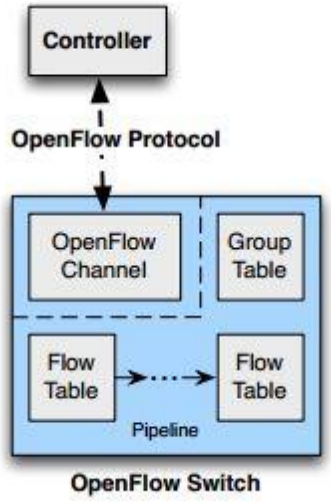
2.3.4 Ο ελεγκτής RYU

Ο ελεγκτής RYU είναι μια υλοποίηση OpenFlow σε Python, θεωρείται ευέλικτος και ευκίνητος για τις συνήθεις λειτουργίες δικτύου. Αποτελείται από συλλογή υποπρογραμμάτων και βιβλιοθηκών όπως Netconf, OF-conf και υποστηρίζει τα πρωτόκολλα OpenFlow 1.0, 1.2, 1.3, 1.4 [8]

2.3.5 Ο ελεγκτής Beacon

Ο ελεγκτής Beacon είναι ένας ελεγκτής OpenFlow σε Java, ο οποίος δημιουργήθηκε το 2010. Είναι ευρέως γνωστός για διδασκαλία, έρευνα, καθώς και η βάση για τον ελεγκτή Floodlight. Μπορεί να λειτουργήσει σε πολλές πλατφόρμες, όπως Windows, Linux, Android OS, υποστηρίζει πολυνηματική (multithreaded) λειτουργία και βασίζεται σε τμηματική λογική (modular), οπότε είναι εύκολα επεκτάσιμος.

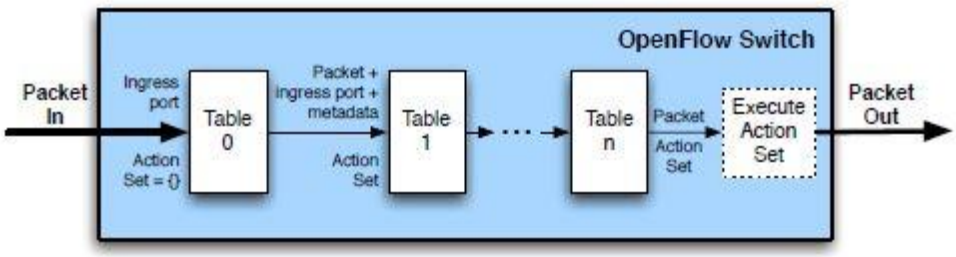
2.4 Μεταγωγέας OpenFlow (OpenFlow Switch)



Εικόνα 2.3 Ένας OpenFlow μεταγωγέας επικοινωνεί με τον ελεγκτή μέσω ασφαλούς σύνδεσης χρησιμοποιώντας το πρωτόκολλο OpenFlow[6]

Ένας μεταγωγέας OpenFlow αποτελείται από έναν ή περισσότερους πίνακες ροής(flow tables) και έναν πίνακα ομάδας, οι οποίοι εκτελούν ανίχνευση πακέτων και προώθηση, και ένα κανάλι OpenFlow προς έναν εξωτερικό ελεγκτή. Ο μεταγωγέας επικοινωνεί με τον ελεγκτή και ο ελεγκτής διαχειρίζεται τον μεταγωγέα μέσα από το πρωτόκολλο OpenFlow.

Χρησιμοποιώντας το πρωτόκολλο OpenFlow, ο ελεγκτής μπορεί να προσθέσει, να ανανεώσει ή να διαγράψει καταχωρήσεις ροής (flow entries) στους πίνακες ροής, διαδραστικά (σαν απάντηση σε αιτήσεις πακέτων) ή προληπτικά. Κάθε πίνακας ροής περιέχει μια συλλογή καταχωρήσεων ροής, οι οποίες αποτελούνται από πεδία αντιστοιχίας (match fields), μετρητές (counters), και μια συλλογή οδηγιών, για να εφαρμόσουν στα αντιστοιχισμένα πακέτα. Η αντιστοίχιση ξεκινά από τον πρώτο πίνακα ροής και μπορεί να συνεχίσει και στους υπόλοιπους αν υπάρχουν, αφού οι πίνακες ροής βρίσκονται σε διασωληνωμένη μορφή (pipelined). Εάν βρεθεί καταχώρηση που να ταιριάζει με το πακέτο, τότε εκτελούνται οι οδηγίες που συνοδεύουν την συγκεκριμένη καταχώρηση, αλλιώς το πακέτο προωθείται στον ελεγκτή OpenFlow, χάνεται ή συνεχίζει την αντιστοίχιση στον επόμενο πίνακα ροής.



Εικόνα 2.4 Διαδρομή πακέτου σε έναν OpenFlow μεταγωγέα[6]

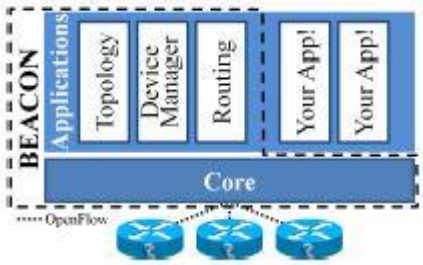
Καταχωρήσεις ροής μπορούν να προωθηθούν σε μια θύρα. Η θύρα μπορεί να είναι φυσική, αλλά μπορεί να είναι και εικονική που ορίζεται από τον μεταγωγέα. Οι εικονικές θύρες μπορούν να προσδιορίσουν διαδικασίες προώθησης, όπως αποστολή στον ελεγκτή, υπερχείλιση ή προώθηση χρησιμοποιώντας μεθόδους χωρίς την χρήση OpenFlow, όπως συμβατική λειτουργία μεταγωγέα.

Εκτός από την ξεχωριστή επεξεργασία των πακέτων, μπορούν να επεξεργαστούν μαζικά με την χρήση πινάκων ομάδας (group tables). Μια ομάδα περιέχει συλλογή διαδικασιών για υπερχείλιση, αλλά και πιο πολύπλοκες λειτουργίες, όπως πολυδιόδευση πακέτων (multipath), γρήγορη επαναδρομολόγηση (fast reroute) και συσσωμάτωση ζεύξεων (link aggregation).

Οι σχεδιαστές των μεταγωγέων μπορούν να υλοποιήσουν τα εσωτερικά κατασκευαστικά στοιχεία με όποιον τρόπο επιθυμούν, με τον όρο οι συσκευές να πληρούν την βασική αρχιτεκτονική που απαιτείται για το πρωτόκολλο OpenFlow.

3 BEACON CONTROLLER

3.1 Ο ελεγκτής Beacon



Εικόνα 3.1 Ο ελεγκτής Beacon [7]

Ο ελεγκτής Beacon είναι ένας ελεγκτής OpenFlow ανοιχτού κώδικα βασισμένος στη γλώσσα προγραμματισμού Java, δημιουργήθηκε το 2010 στο πανεπιστήμιο του Στάνφορντ και χρησιμοποιείται ευρέως για διδασκαλία, έρευνα και σαν βάση για τον ελεγκτή Floodlight.

Η πρώτη πλατφόρμα ανοιχτού κώδικα για τις πρώτες εφαρμογές της τεχνολογίας OpenFlow ήταν ο NOX. Ο NOX επέτρεπε στους προγραμματιστές να επιλέξουν μεταξύ δικτυακών εφαρμογών με μια γλώσσα φιλική στον προγραμματιστή, όπως η Python, ή εφαρμογές υψηλής απόδοσης με την γλώσσα C++. Με τον καιρό, δημιουργήθηκε η ανάγκη δημιουργίας μιας πλατφόρμας που θα συνδύαζε την ευκολία στην ανάπτυξη εφαρμογών και υψηλές αποδόσεις.

Η γλώσσα προγραμματισμού και το αναπτυξιακό περιβάλλον κατέχουν σημαντικό ρόλο στην παραγωγικότητα των προγραμματιστών, αλλά αποτελούν και έναν περιοριστικό παράγοντα στην απόδοση των εφαρμογών. Υπάρχουν πολλές γλώσσες φιλικές στον προγραμματιστή, αλλά η απόδοση τους όταν χρησιμοποιούνται σε έναν ελεγκτή OpenFlow ήταν άγνωστη. [7]

3.2 Παραγωγικότητα προγραμματιστή

Αυτή η ενότητα αναφέρεται στις επιλογές σχεδιασμού του ελεγκτή Beacon, προκειμένου να βελτιωθεί η παραγωγικότητα του έργου του προγραμματιστή. Οι σχεδιαστικές επιλογές αφορούν: την γλώσσα προγραμματισμού, τις βιβλιοθήκες και την διεπαφή προγραμματισμού εφαρμογών (API).

3.2.1 Γλώσσα προγραμματισμού

Οι γλώσσες C και C++ ήταν οι πρώτες γλώσσες προγραμματισμού για τους ελεγκτές OpenFlow πριν την ανάπτυξη του Beacon. Με αυτές τις γλώσσες, μπορούν να υλοποιηθούν εφαρμογές πολύ υψηλής απόδοσης, αλλά έχουν σημαντικά μειονεκτήματα όπως μεγάλους χρόνους μεταγλώττισης (>10 λεπτών), σφάλματα μεταγλώττισης χωρίς σαφή διευκρίνιση και σφάλματα διαχείρισης μνήμης που οδηγούν σε κατακερματισμό σφαλμάτων, διαρροές μνήμης κτλ.

Έχουν γίνει προσπάθειες για να ελαχιστοποιηθούν τα προβλήματα που αναφέρθηκαν παραπάνω, όμως αποδεικνύονται ατελείς, γιατί απαιτούνται τεχνικές δύσχρηστες και περιφερειακές συσκευές. Έτσι, δημιουργήθηκε η ανάγκη ανάπτυξης ενός ελεγκτή OpenFlow υψηλής απόδοσης σε μια περισσότερο φιλική γλώσσα προγραμματισμού, ο οποίος θα μπορεί να λειτουργεί σε βασικές υλικολογισμικές συσκευές, όπου η CPU και η RAM θα μπορούν να επεκταθούν εύκολα και με το χαμηλότερο δυνατό κόστος.

Τα χαρακτηριστικά της επιλεγόμενης γλώσσας προγραμματισμού για την ανάπτυξη του ελεγκτή ήταν η διαχείριση μνήμης, η πολλαπλή πλατφόρμα και η υψηλή απόδοση.

Η αυτόματη διαχείριση μνήμης μπορεί να εξαλείψει τα περισσότερα σφάλματα διαχείρισης μνήμης. Οι γλώσσες με αυτές την ικανότητα συνήθως έχουν καθόλου ή ελάχιστη μεταγλώττιση, εξαλείφοντας το χρόνο αναμονής για την μεταγλώττιση του προγράμματος. Επίσης, αυτές οι γλώσσες παρέχουν αναφορά σφάλματος υποδεικνύοντας το ακριβές σημείο όπου σημειώθηκε το σφάλμα. Οι γλώσσες που έχουν αυτές τις δυνατότητες είναι οι C#, Java και Python.

Αρχικά, ο ελεγκτής Beacon είχε σχεδιαστεί να λειτουργεί στο λειτουργικό σύστημα Linux. Οι δημιουργοί του όμως ήθελαν να μπορεί να λειτουργεί και σε άλλες πλατφόρμες, όπως Mac OSX, Windows χωρίς να σημειώνει πρόβλημα φορητότητας. Το θέμα της φορητότητας είχε αποτρέψει και άλλους δημιουργούς από την ανάπτυξη ελεγκτών σε πλατφόρμες εκτός Linux. Παρόλο που οι C#, Java, Python μπορούν να εκτελεστούν σε όλες τις πλατφόρμες, ο CLR (Common Language Runtime), ο επίσημος διερμηνέας της C#, υποστήριζε μόνο τα Windows. Έτσι, η γλώσσα C# αποκλείστηκε από επιλογή.

Ο όρος της υψηλής απόδοσης είναι υποκειμενικός. Ένα χαρακτηριστικό είναι η ικανότητα να αυξάνεται η αποδοτικότητα ανάλογα με την χρήση των επεξεργαστών. Η έλλειψη της πραγματικής πολυνημάτωσης (multi-threading) στον επίσημο διερμηνέα της Python εξέλειψε αυτήν την γλώσσα από επιλογή. Για την άλλη επιλογή, την Java, δεν είχαν γνώση της απόδοσης της σε έναν ελεγκτή OpenFlow. Με βάση άλλα προγράμματα στα οποία είχε χρησιμοποιηθεί η Java (Hadoop, Tomcat) όπου εμφάνιζε υψηλή απόδοση, αποφασίστηκε να χρησιμοποιηθεί αυτή σαν γλώσσα προγραμματισμού και όπως αποδείχθηκε, η απόδοση της ήταν εκπληκτικά υψηλή.

3.2.2 Βιβλιοθήκες

Ο ελεγκτής Beacon χρησιμοποιεί πολλαπλές βιβλιοθήκες προκειμένου να μεγιστοποιήσει την επαναχρησιμοποίηση του κώδικα και να διευκολύνει το φόρτο του ελεγκτή και των εφαρμογών που τον χρησιμοποιούν. Η πιο σημαντική βιβλιοθήκη είναι η Spring. Η Spring είναι ένα πλαίσιο εφαρμογών ανοιχτού κώδικα και αναστροφής ελέγχου για την πλατφόρμα Java. Χρησιμοποιείται αρκετά και θεωρείται η αντικατάσταση, αν όχι προσθήκη του μοντέλου Enterprise JavaBean(EJB). Η πρώτη έκδοση αναπτύχθηκε από τον Rod Johnson, που δημοσίευε το πλαίσιο στο βιβλίο του "Expert One-on-One J2EE Development without EJB" τον Οκτώβριο του 2002. [9]

Τα δύο κύρια στοιχεία της βιβλιοθήκης Spring που χρησιμοποιούνται στον Beacon είναι το πλαίσιο διαδικτύου (Web framework) και το πλαίσιο βασισμένο στην αναστροφή ελέγχου (Inversion of Control - IoC). Μια κοινή εργασία στην Java είναι η δημιουργία αντικειμένων και η σύνδεση μεταξύ τους καθιστώντας το ένα ιδιότητα κάποιου άλλου. Η χρήση του πλαισίου IoC επιτρέπει στους προγραμματιστές να διατηρούν σε λίστα σε ένα

XML αρχείο ή σαν σημειώσεις Java, ποια αντικείμενα δημιουργήθηκαν, πώς είναι συνδεδεμένα μεταξύ τους, καθώς και πώς να τα ανακτούν. Χρησιμοποιώντας ένα πλαίσιο IoC, εξοικονομείται σημαντικός χρόνος, σε αντίθεση με την κοινή εναλλακτική της δημιουργίας πολλών κλάσεων ταυτόχρονα. Ο ελεγκτής Beacon χρησιμοποιεί το πλαίσιο IoC της Spring για την σύνδεση εντός και μεταξύ εφαρμογών. Το Web Framework της Spring χρησιμοποιείται για να καθορίζει τις αιτήσεις Web και REST σε απλές κλήσεις μεθόδων, εκτελεί αυτόματη μετατροπή του αιτήματος και των δεδομένων απόκρισης σε αντικείμενα Java.

3.2.3 Διεπαφή Προγραμματισμού Εφαρμογών (API) Beacon

Η Διεπαφή Προγραμματισμού Εφαρμογών (API) για τον ελεγκτή Beacon είναι σχεδιασμένη να είναι απλή και να μην επιβάλλει περιορισμούς, έτσι ώστε οι προγραμματιστές να μπορούν να χρησιμοποιούν οποιαδήποτε διαθέσιμη δομή Java, όπως νήματα (threads), χρονόμετρα (timers) , υποδοχές (sockets) κτλ. Το API περιλαμβάνει πληροφορίες σχετικά με την αλληλεπίδραση με τους μεταγωγείς OpenFlow. Ο ελεγκτής Beacon χρησιμοποιεί το μοντέλο του παρατηρητή, όπου οι ακροατές (listeners) καταγράφονται για να ανταποκρίνονται σε συγκεκριμένα γεγονότα.

Ο Beacon περιλαμβάνει την βιβλιοθήκη OpenFlowJ για την επεξεργασία μηνυμάτων τύπου OpenFlow. Η OpenFlowJ είναι μια αντικειμενοστραφής υλοποίηση Java του πρωτοκόλλου OpenFlow 1.0. Περιέχει κώδικα για την αποσειριοποίηση μηνυμάτων που προέρχονται από το δίκτυο, σε αντικείμενα, και για την σειριοποίηση μηνυμάτων για την αποστολή στο δίκτυο.

Η αλληλεπίδραση με τους μεταγωγείς OpenFlow γίνονται μέσω της διεπαφής IBeaconProvider. Οι ακροατές συσχετίζονται για να ανταποκρίνονται σε γεγονότα όπως η προσθήκη ή αφαίρεση ενός μεταγωγέα (IOFSwitchListener), η αρχικοποίηση ενός μεταγωγέα (IOFInitializerListener) και η λήψη συγκεκριμένων μηνυμάτων τύπου OpenFlow (IOFMessageListener).

Επίσης, ο Beacon περιλαμβάνει εφαρμογές που λειτουργούν σχετικά με τον πυρήνα, προσθέτοντας επιπλέον API:

Device Manager. Παρακολουθεί συσκευές που είναι συνδεδεμένες στο δίκτυο, συμπεριλαμβανομένου των διευθύνσεων τους (Ethernet, IP), την τελευταία ημερομηνία σύνδεσης, τον μεταγωγέα και την πύλη στην οποία κάθε συσκευή συνδέθηκε την τελευταία φορά. Παρέχει μια διεπαφή (IDeviceManager) για την αναζήτηση γνωστών συσκευών, και την ικανότητα ενημέρωσης για συγκεκριμένα γεγονότα, όπως σύνδεση νέων συσκευών, ενημέρωση ή αφαίρεση συσκευών.

Topology. Ανακαλύπτει συνδέσμους μεταξύ συνδεδεμένων μεταγωγέων OpenFlow. Η διεπαφή της (ITopology) επιτρέπει την ανάκτηση μιας λίστας τέτοιων συνδέσμων και ενημέρωση για την σύνδεση ή αφαίρεση ενός συνδέσμου.

Routing. Παρέχει την συντομότερη διαδρομή δρομολόγησης (επιπέδου δύο) μεταξύ συσκευών στο δίκτυο. Η εφαρμογή εξάγει την διεπαφή IRoutingEngine, επιτρέποντας εναλλαγές στις ρυθμίσεις που ελέγχουν την δρομολόγηση. Χρησιμοποιεί την μέθοδο υπολογισμού συντομότερης διαδρομής μεταξύ όλων των συσκευών και εξαρτάται από τις εφαρμογές Topology, Device Manager.

Web. Παρέχει μια διεπαφή χρήστη (User Interface – UI) για τον Beacon. Μέσω της διεπαφής IWebManageable, επιτρέπει την προσθήκη νέων στοιχείων στο UI.

3.3 Δομοστοιχείωση Εκτέλεσης (Runtime Modularity)

Οι περισσότεροι ελεγκτές OpenFlow έχουν την ικανότητα να επιλέγουν ποιες εφαρμογές θα δημιουργήσουν (δομοστοιχείωση χρόνου μεταγλώττισης) και ποιες θα εκτελεστούν με την εκκίνηση του ελεγκτή (δομοστοιχείωση χρόνου εκκίνησης). Ο Beacon έχει την δυνατότητα όχι μόνο την εκκίνηση και διακοπή εφαρμογών κατά την διάρκεια της λειτουργίας του, αλλά και την πρόσθεση, αφαίρεση τους (δομοστοιχείωση χρόνου εκτέλεσης), χωρίς να διακοπεί η λειτουργία του.

Αυτό προσφέρει στους προγραμματιστές νέους τρόπους αλληλεπίδρασης με τον ελεγκτή. Πιθανές χρήσεις είναι η δημιουργία και εγκατάσταση εφαρμογής για την προσωρινή βελτίωση της συλλογής πληροφοριών σχετικά με την αποσφαλμάτωση, η διόρθωση

σφαλμάτων ή βελτίωση των εφαρμογών που ήδη υπάρχουν, η εγκατάσταση νέων εφαρμογών κατά την διάρκεια της λειτουργίας του, η απομόνωση και απενεργοποίηση των εφαρμογών που παρουσιάζουν εσφαλμένη συμπεριφορά.

Για την ενεργοποίηση αυτής της λειτουργίας, ο Beacon χρησιμοποιεί μια υλοποίηση του OSGi, τον Equinox. Το OSGi καθορίζει τα bundles (οι εφαρμογές ή τα εξαρτήματα) που είναι αρχεία JAR και περιέχουν κλάσεις και /ή άλλες πηγές αρχείων μαζί με τα απαραίτητα δεδομένα. Τα δεδομένα ενός bundle προσδιορίζουν την ταυτότητα, την έκδοση, τις εξαρτήσεις από άλλα bundles ή πακέτα κώδικα και πακέτα κώδικα που χρησιμοποιούνται από άλλα bundles. Ακόμη, κατά την διάρκεια της εκτέλεσης, τα bundles είναι αυτά τα οποία μπορούν να εκκινήσουν, να σταματήσουν, να προστεθούν και να αφαιρεθούν.

Είναι στην ευχέρεια του προγραμματιστή να καθορίσει πώς θα δομοστοιχειωθεί η εφαρμογή. Για παράδειγμα, ένα bundle μπορεί να περιέχει πολλαπλές εφαρμογές, μόνο μία εφαρμογή ή μια εφαρμογή μπορεί να χρησιμοποιείται από πολλαπλά bundles. Αυτές οι αποφάσεις γίνονται βασισμένες στον βαθμό δομοστοιχείωσης μια εφαρμογής. Για παράδειγμα, εάν ένα μέρος μιας εφαρμογής μπορεί να αντικατασταθεί κατά την εκκίνηση ή εκτέλεση, όπως η μηχανή παραγωγής διαδρομών που χρησιμοποιείται από την εφαρμογή δρομολόγησης του Beacon.

Ένα στοιχείο των προδιαγραφών OSGi είναι το μητρώο υπηρεσίας. Ενεργεί ως μεσάζων για τις υπηρεσίες και τους καταναλωτές, έτσι ώστε οι υπηρεσίες να καταχωρούνται, και οι καταναλωτές να ανακτούν την συγκεκριμένη υπηρεσία που χρειάζονται. Ο Beacon χρησιμοποιεί αρκετά αυτό το μοντέλο, παρέχοντας στους καταναλωτές την δυνατότητα να κάνουν αιτήσεις και να λαμβάνουν εφαρμογές των υπηρεσιών. Η διάρκεια του κύκλου ζωής μιας υπηρεσίας είναι δυναμική. Ανάλογα με το ποια bundles είναι εγκατεστημένα και εκτελούνται, καθορίζεται ποιες υπηρεσίες λειτουργούν.

3.4 Απόδοση

Η απόδοση σε έναν ελεγκτή OpenFlow τυπικά μετράται με τον αριθμό των εισερχόμενων μηνυμάτων εισερχόμενων πακέτων (PacketIn) που μπορεί να επεξεργαστεί και

να απαντήσει ανά δευτερόλεπτο ένας ελεγκτής, αλλά και με τον μέσο χρόνο επεξεργασίας κάθε γεγονότος. Αυτή η ενότητα περιγράφει την αρχιτεκτονική του Beacon όσον αφορά την επεξεργασία μηνυμάτων τύπου OpenFlow.

3.4.1 Χειρισμός γεγονότων

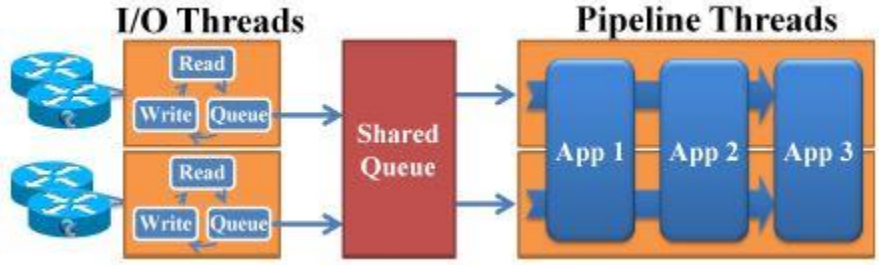
Οι εφαρμογές που υλοποιούν την διεπαφή IOFMessageListener καταχωρούνται με την υπηρεσία IBeaconProvider, για να λαμβάνουν συγκεκριμένα μηνύματα τύπου OpenFlow από τους μεταγωγείς OpenFlow. Οι καταχωρημένοι ακροατές σχηματίζουν έναν σειριακό αγωγό επεξεργασίας για κάθε μήνυμα τύπου OpenFlow. Η σειρά των ακροατών μέσα σε κάθε αγωγό είναι ρυθμιζόμενη και ο ακροατής μπορεί να επιλέξει εάν θα διαδώσει ένα γεγονός ή όχι. Ένα παράδειγμα αγωγού φαίνεται στην εικόνα 3.2. Αυτός ο αγωγός έχει τρεις καταχωρημένες εφαρμογές για την λήψη μηνυμάτων PacketIn: Device Manager, Topology, Routing.

Εικόνα 3.2 Ροή διαδικασιών του IOFMessageListener[7]

3.4.2 Ανάγνωση Μηνυμάτων τύπου OpenFlow

Για την επίτευξη υψηλής απόδοσης, όλες οι εφαρμογές του Beacon είναι πολυνηματικές. Δύο από τις πολυνηματικές εφαρμογές που χρησιμοποιούνται για την ανάγνωση και επεξεργασία μηνυμάτων OpenFlow παρουσιάζονται στην συνέχεια.

3.4.2.1 Κοινόχρηστη Ουρά Αναμονής (Shared Queue)



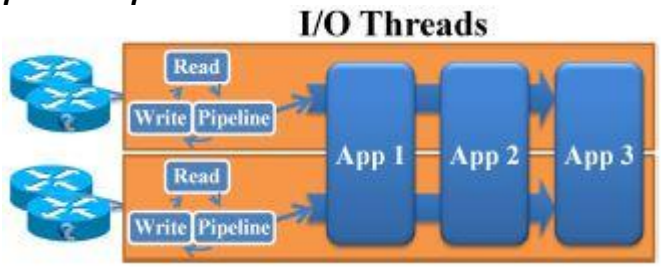
Εικόνα 3.3 Κοινόχρηστη Ουρά Αναμονής [7]

Η παραπάνω εικόνα δείχνει μια κοινόχρηστη ουρά αναμονής, η οποία περιέχει δύο σύνολα νημάτων. Το πρώτο σύνολο, τα λεγόμενα νήματα I/O (I/O threads), είναι υπεύθυνο για την ανάγνωση και αποσειριοποίηση των μηνυμάτων OpenFlow από τους μεταγωγείς. Έπειτα, μεταφέρει τα αναγνωσμένα σε μια ουρά αναμονής. Κάθε μεταγωγέας αντιστοιχίζεται σε ένα μόνο νήμα I/O και πολλαπλοί μεταγωγείς μπορούν να αντιστοιχηθούν στο ίδιο νήμα. Επίσης, το νήμα I/O είναι υπεύθυνο και για την καταγραφή εξερχόμενων μηνυμάτων OpenFlow προς τους μεταγωγείς.

Το δεύτερο σύνολο, τα νήματα αγωγού (pipeline threads), λαμβάνει τα μηνύματα από την κοινόχρηστη ουρά αναμονής και χρησιμοποιεί τον αγωγό επεξεργασίας της διεπαφής IOFMessageListener, όπου αντιστοιχίζονται ανάλογα με τον τύπο του κάθε μηνύματος. Όλα τα νήματα αγωγού μπορούν να απασχολούνται επεξεργάζοντας μηνύματα OpenFlow που βρίσκονται σε ουρά αναμονής. Ωστόσο, αυτό το μοντέλο απαιτεί ασφάλεια στην κοινόχρηστη ουρά αναμονής, καθώς και μεταξύ των δύο συνόλων νημάτων.

Παραλλαγές αυτού του μοντέλου μπορούν να έχουν πολλαπλές ουρές, ίσως ανά μεταγωγέα ή περισσότερες με διαφορετικές προτεραιότητες η κάθε μία. Αυτό θα μπορούσε να βελτιώσει την αμεροληψία στην επεξεργασία μηνυμάτων OpenFlow μεταξύ των μεταγωγέων, μέσα από έναν χρονοπρογραμματισμό εξυπηρέτησης των ουρών εκ περιτροπής (round-robin).

3.4.2.2 Ολοκληρωμένη εκτέλεση



Εικόνα 3.4 Run-to-completion [7]

Η εικόνα 3.4 δείχνει το μοντέλο ολοκληρωμένης εκτέλεσης, η οποία είναι μια απλοποιημένη εκδοχή της κοινόχρηστης ουράς αναμονής με ένα μόνο σύνολο νημάτων I/O. Κάθε νήμα σημειώνει ομοιότητες με τα νήματα I/O στο μοντέλο της κοινόχρηστης ουράς. Ωστόσο, αντί να υπάρχει ουρά αναμονής για κάθε μήνυμα ώστε να επεξεργαστεί από τη νήμα διασωλήνωσης, το αναγνωσμένο μήνυμα οδηγείται άμεσα στον αγωγό διεπαφής IOFMessageListener.

Αυτό το μοντέλο δεν απαιτεί ασφάλεια κατά την διάρκεια ανάγνωσης του μηνύματος, γιατί το νήμα που αποσειριοποιεί το μήνυμα το διοχετεύει στον αγωγό. Επίσης, προσφέρει την ακόλουθη εγγύηση για τις διεπαφές IOFMessageListener: για κάθε μεταγωγέα, μόνο ένα νήμα επεξεργάζεται μηνύματα από αυτόν κάθε φορά.

Ένας περιορισμός αυτού του μοντέλου είναι ότι οι απασχολημένοι μεταγωγείς θα είναι στατικά αντιστοιχισμένοι σε ένα υποσύνολο νημάτων, αφήνοντας άλλα νήματα ανενεργά. Το πρόβλημα αυτό θα μπορούσε να αντιμετωπιστεί με περιοδική αναπροσαρμογή των μεταγωγέων στα νήματα.

Ο Beacon χρησιμοποιεί ην ολοκληρωμένη εκτέλεση με έναν ρυθμιζόμενο αριθμό νημάτων I/O. Οι μεταγωγείς OpenFlow, κατά την σύνδεση τους, αντιστοιχίζονται μέσω χρονοπρογραμματισμού επεξεργασίας εκ περιτροπής σε νήματα I/O και παραμένουν στατικά συνδεδεμένοι μέχρι να αποσυνδεθούν από το δίκτυο. Κάθε νήμα I/O επεξεργάζεται όλα τα δεδομένα από κάθε μεταγωγέα που είναι συνδεδεμένος μαζί του.

3.4.3 Αποστολή Μηνυμάτων OpenFlow

Ένας άλλος παράγοντας που επηρεάζει την απόδοση του ελεγκτή είναι ο τρόπος με τον οποίο στέλνονται τα μηνύματα στους μεταγωγείς. Επειδή ο Beacon είναι πολυνηματικός, πολλαπλά νήματα στέλνουν ταυτόχρονα μηνύματα και οι μέθοδοι πρέπει να είναι συγχρονισμένες για να αποφευχθούν συγκρούσεις.

Στο σχήμα 3.5, φαίνεται ο αλγόριθμος εγγραφής και αποστολής μηνυμάτων σε μεταγωγείς OpenFlow. Οι εφαρμογές καλούν την μέθοδο "Write", η οποία σειριοποιεί και καταχωρεί το μήνυμα σε μια προσωρινή μνήμη του μεταγωγέα, και στην συνέχεια το προωθεί στην υποδοχή.

```
1  function WRITE(OFMessage msg)
2      buf.append(msg)
3      flush()
4  end function
5  function FLUSH
6      socket.write(buf)
7  end function
```

Εικόνα 3.5 Μέθοδοι αρχικοποίησης μεταγωγέα

Με αυτό το μοντέλο, η απόδοση ήταν χαμηλή. Η μέθοδος ανίχνευσης έδειξε ότι το εικονικό μηχάνημα Java (Java Virtual Machine – JVM) αντιστοιχούσε κάθε εγγραφή στην υποδοχή με εγγραφή στον πυρήνα του συστήματος, χωρίς ενδιάμεση επεξεργασία. Σε ένα απασχολημένο σύστημα, ο χρόνος που δαπανάται σε ανταλλαγή δεδομένων μεταξύ χρήστη και πυρήνα να είναι αρκετά μεγάλος.

Για την αντιμετώπιση αυτού του προβλήματος, αναπτύχθηκε ένα νέο μοντέλο. Το πρώτο κομμάτι του μοντέλου φαίνεται στην εικόνα 3.6, είναι μια τροποποιημένη μέθοδος για άδειασμα της προσωρινής μνήμης "flush", που περιέχει δύο μεταβλητές boolean, τις written και needSelect. Η μεταβλητή written θεωρείται αληθής όταν εμφανίζεται μια εγγραφή υποδοχής, αποτρέποντας ακόλουθες εγγραφές μέχρι η μεταβλητή να τεθεί ψευδής. Η μεταβλητή needSelect ορίζεται όταν υπάρχουν δεδομένα που δεν έχουν γραφτεί, μετά από μια εγγραφή υποδοχής, υποδεικνύοντας ότι η προσωρινή μνήμη εξερχόμενου TCP ήταν γεμάτη, και τα υπόλοιπα δεδομένα πρέπει να εγγραφούν μόλις υπάρξει διαθέσιμος χώρος στην προσωρινή μνήμη.

```
1  function FLUSH
2      if !written && !needSelect then
3          socket.write(buf)
4          written <-- true
5          if buf.remaining() > 0 then
6              needSelect <-- true
7          end if
8      end if
9  end function
```

Εικόνα 3.6 Τροποποιημένη μέθοδος Flush

Το δεύτερο κομμάτι του μοντέλου αφορά τροποποιήσεις στον βρόχο I/O όπως φαίνεται στην εικόνα 3.7. Οι γραμμές 3-9 και 15-18 προστέθηκαν για την υποστήριξη του μοντέλου. Οι γραμμές 3-9 εξασφαλίζουν ότι κάθε μεταγωγέας θα γράψει όποια εξερχόμενα δεδομένα μία φορά ανά βρόχο I/O, όταν η ουρά TCP δεν είναι πλήρης. Οι γραμμές 15-18 εξασφαλίζουν ότι οι εγγραφές πραγματοποιούνται μόνο όταν υπάρχει διαθέσιμος χώρος στην προσωρινή μνήμη.

Το αποτέλεσμα είναι ότι θα μηνύματα εγγράφονται είτε άμεσα είτε μια φορά ανά βρόχο I/O. Για τα συστήματα που βρίσκονται υπό μεγάλο φόρτο εργασίας, δημιουργείται ένας μηχανισμός μεταξύ βρόχων I/O, μειώνοντας τις κλήσεις εγγραφής και την αλληλεπίδραση μεταξύ χρήστη και πυρήνα.

```
1  function IOLOOP
2      while true do
3          for all Switch sw : switches do
4              sw.written <-- false
5              sw.flush()
6              if sw.needSelect then
7                  sw.selectKey.addOp(WRITE)
8              end if
9          end for
10         readySwitches = select(switches)
11         for all Switch sw : readySwitches do
12             if sw.selectKey.readable then
13                 readAndProcessMessages(sw)
14             end if
15             if sw.selectKey.writeable then
16                 sw.needSelect <-- false
17                 sw.flush()
18             end if
19         end for
20     end while
21 end function
```

Εικόνα 3.7 Βρόχος I/O

4 MININET

4.1 Το λογισμικό Mininet

Για το πειραματικό μέρος της διπλωματικής εργασίας, χρειάζεται η δημιουργία ενός δικτύου. Το δίκτυο αυτό πρέπει να αποτελείται από συσκευές προώθησης και δρομολόγησης πακέτων, συσκευές που θα αντιπροσωπεύουν τους τελικούς χρήστες και συσκευές πάνω στις οποίες θα λειτουργεί ο ελεγκτής OpenFlow. Γίνεται κατανοητό ότι για καλύτερη εκτίμηση των αποτελεσμάτων, το δίκτυο θα πρέπει να είναι εύκολα τροποποιήσιμο. Κατά την εκτέλεση των πειραμάτων, είναι απαραίτητη η ευελιξία της τοπολογίας του δικτύου, για να δοκιμαστεί σε διαφορετικές συνθήκες. Αυτό καθιστά δύσκολη έως και αδύνατη την προσομοίωση του συστήματος με φυσικές συσκευές, διότι απαιτούνται οικονομικοί πόροι, χρόνος, αλλά και βεβαιότητα ότι οι πραγματικές δικτυακές συσκευές θα συνεχίσουν να λειτουργούν σε περίπτωση δυσκολιών.

Αρχικά, το δίκτυο περιλάμβανε φυσικούς μεταγωγείς, εγκατεστημένους σε εργαστήριο της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών. Όμως αποδείχθηκε δύσκολη η διεξαγωγή πειραμάτων, καθώς η σύνδεση με την συσκευή του ελεγκτή OpenFlow έπρεπε να βρίσκεται στον χώρο του Πολυτεχνείου και η όποια διακοπή ρεύματος προκαλούσε απώλεια των δεδομένων.

Έτσι, έγινε χρήση του προγράμματος Mininet. Το Mininet είναι ένας εξομοιωτής δικτύου. Εκτελεί ταυτόχρονα ένα σύνολο από τερματικά, μεταγωγείς, δρομολογητές και συνδέσμους σε έναν μοναδικό πυρήνα Linux. Χρησιμοποιεί ένα ελαφρύ εικονικό περιβάλλον για να μετατρέψει ένα σύστημα σε ένα πλήρες δίκτυο, με τον ίδιο πυρήνα, σύστημα και κώδικα χρήστη. Ένα τερματικό στο Mininet συμπεριφέρεται όπως ένα πραγματικό μηχάνημα. Παρέχεται η δυνατότητα ασφαλούς σύνδεσης (όπως ssh) και η εκτέλεση οποιοδήποτε προγράμματος, αρκεί να είναι εγκατεστημένο στο σύστημα Linux. Τα προγράμματα που εκτελούνται στέλνουν πακέτα μέσω μιας φαινομενικής σύνδεσης-

διεπαφής τύπου Ethernet. Τα πακέτα επεξεργάζονται από συσκευές που φαίνεται να λειτουργούν σαν μεταγωγείς, δρομολογητές τύπου Ethernet, σε ουρές αναμονής. Όταν δύο προγράμματα, όπως το iperf (μετρητής χωρητικότητας γραμμής δύο σημείων) μεταξύ πελάτη και διακομιστή επικοινωνούν μέσω Mininet, η απόδοση θα μπορούσε να ταιριάξει με αυτή των δύο φυσικών μηχανών.

Εν συντομία, τα εικονικά τερματικά, οι μεταγωγείς, οι σύνδεσμοι και οι ελεγκτές δημιουργούνται με την χρήση λογισμικού (software) και όχι υλικού (hardware). Είναι δυνατή η δημιουργία ενός δικτύου Mininet που μοιάζει με ένα δίκτυο που βασίζεται σε hardware ή η δημιουργία ενός δικτύου hardware παρόμοιο με του Mininet, τα οποία θα εκτελούν τον ίδιο δυαδικό κώδικα και τις ίδιες εφαρμογές σε κάθε μία πλατφόρμα.

4.2 Πλεονεκτήματα του Mininet

Το Mininet αποτελεί ένα εύχρηστο και εύκολο εργαλείο στην προσομοίωση δικτύων, με πολλά σημαντικά πλεονεκτήματα. Αυτή η ενότητα περιέχει τα πιο αξιοπρόσεκτα από αυτά.

- Η δημιουργία ενός απλού δικτύου πραγματοποιείται σε ελάχιστο χρονικό διάστημα. Αυτό έχει ως συνέπεια την γρήγορη εκτέλεση, τροποποίηση και αποσφαλμάτωση.
- Η δημιουργία πολλαπλών τοπολογιών. Από ένα απλουστευμένο δίκτυο, μέχρι μεγαλύτερες τοπολογίες (όπως το Internet), κέντρο δεδομένων και πολλά άλλα.
- Οποιοδήποτε πρόγραμμα, λογισμικό είναι εγκατεστημένο στο λειτουργικό σύστημα Linux μπορεί να εκτελεστεί.
- Μπορεί να τροποποιηθεί η προώθηση πακέτων, καθώς οι μεταγωγείς του Mininet είναι προγραμματιζόμενοι χρησιμοποιώντας το πρωτόκολλο OpenFlow.
- Η ευέλικτη εκτέλεση του Mininet σε πολλαπλές πλατφόρμες, υπολογιστές, εικονικά μηχανήματα, ακόμα και σε τεχνολογία cloud (νεφοϋπολογιστική).
- Η αναπαραγωγή των αποτελεσμάτων μπορεί να γίνει σε οποιοδήποτε υπολογιστή, αρκεί να εκτελεστεί ο ίδιος κώδικας.

- Το Mininet είναι εύχρηστο λογισμικό χρησιμοποιώντας την φιλική γλώσσα προγραμματισμού Python.
- Είναι λογισμικό ανοιχτού κώδικα και υπό ενεργή ανάπτυξη. Υπάρχει μια ενεργή κοινότητα του Mininet αποτελούμενη από χρήστες και προγραμματιστές, η οποία μπορεί να συμβάλλει στην αντιμετώπιση οποιουδήποτε προβλήματος και στην προσθήκη επεξηγήσεων.

4.3 Δημιουργία τοπολογιών

Το Mininet υποστηρίζει παραμετροποιήσιμες τοπολογίες. Με την χρήση της γλώσσας προγραμματισμού Python, μια ευέλικτη τοπολογία μπορεί να δημιουργηθεί, βασισμένη στις παραμέτρους που επιθυμεί ο χρήστης, και να χρησιμοποιηθεί ξανά για πολλαπλά πειράματα.

Για παράδειγμα, μια απλουστευμένη τοπολογία δικτύου αποτελούμενη από έναν συγκεκριμένο αριθμό τερματικών συνδεδεμένα σε έναν μεταγωγέα φαίνεται στο παρακάτω σχήμα.

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def build(self, n=2):
        switch = self.addSwitch('s1')
        # Python's range(N) generates 0..N-1
        for h in range(n):
            host = self.addHost('h%s' % (h + 1))
            self.addLink(host, switch)
```

```
def simpleTest():
    "Create and test a simple network"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo)
    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    net.stop()

if __name__ == '__main__':
    # Tell mininet to print useful information
    setLogLevel('info')
    simpleTest()
```

Σημαντικές κλάσεις, μέθοδοι, συναρτήσεις και μεταβλητές του παραπάνω κώδικα περιλαμβάνουν:

Topo : Η βασική κλάση για τις τοπολογίες στο Mininet.

addSwitch() : Προσθήκη ενός μεταγωγέα στην τοπολογία και επιστρέφει την ονομασία του.

addHost() : Προσθήκη ενός τερματικού στην τοπολογία και επιστρέφει την ονομασία του.

addLink() : Προσθήκη ενός αμφίδρομου συνδέσμου στην τοπολογία. Οι σύνδεσμοι στο λογισμικό Mininet είναι αμφίδρομοι, εκτός και αν αναφέρεται διαφορετικά.

Mininet : Η βασική κλάση, υπεύθυνη για την δημιουργία και διαχείριση του δικτύου.

start() : Ενεργοποίηση λειτουργίας του δικτύου.

pingAll() : Έλεγχος συνδεσιμότητας των τερματικών εκτελώντας διαδοχικά ping μεταξύ των κόμβων.

stop() : Διακοπή λειτουργίας του δικτύου.

net.hosts : Εμφάνιση των ονομασιών όλων των τερματικών στο δίκτυο.

dumpNodeConnections() : Απόρριψη συνδέσεων από και προς ένα σύνολο κόμβων.

4.4 Ρύθμιση Παραμέτρων Απόδοσης

Εκτός από την βασική δικτυακή συμπεριφορά, το Mininet παρέχει περιορισμό απόδοσης και χαρακτηριστικά απομόνωσης μέσω των κλάσεων CPULimitedHost και TCLink.

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def build(self, n=2):
        switch = self.addSwitch('s1')
        for h in range(n):
            # Each host gets 50%/n of system CPU
            host = self.addHost('h%s' % (h + 1),
               cpu=.5/n)
            # 10 Mbps, 5ms delay, 10% loss, 1000 packet queue
            self.addLink(host, switch,
               bw=10, delay='5ms', loss=10, max_queue_size=1000, use_htb=True)

def perfTest():
```

```
    "Create network and run simple performance test"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo=topo,
                  host=CPULimitedHost, link=TCLink)
    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    print "Testing bandwidth between h1 and h4"
    h1, h4 = net.get('h1', 'h4')
    net.iperf((h1, h4))
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    perfTest()
```

Μερικές αξιοσημείωτες μέθοδοι και παράμετροι:

self.addHost(name, cpu=f) : Επιτρέπει την παραμετροποίηση των πόρων του επεξεργαστή του συστήματος που θα κατανεμηθεί στο εικονικό τερματικό.

self.addLink(node1, node2, bw=10, delay=’5ms’, max_queue_size=1000, loss=10, use_htb=True) : Προσθήκη ενός αμφίδρομοι συνδέσμου με χαρακτηριστικά όπως ανεκτικότητα απώλειας, καθυστέρηση και χωρητικότητα, μέγιστος αριθμός πακέτων για ουρά αναμονής τα χίλια πακέτα Η παράμετρος bw εκφράζεται σε Mb/s, η καθυστέρηση σε συμβολοσειρά, η ανεκτικότητα απώλειας σε ποσοστό και το μέγιστο μέγεθος ουράς αναμονής σε πακέτα.

4.5 Εκτέλεση Προγραμμάτων στα Τερματικά

Η εκτέλεση προγραμμάτων στα εικονικά τερματικά είναι ένα από τα σημαντικότερα πράγματα που πρέπει να διεξαχθούν κατά την διάρκεια των πειραμάτων, έτσι ώστε να υποστηρίζονται περισσότερες εντολές όπως η pingAll() και η iperf(), οι οποίες παρέχονται από το λογισμικό Mininet.

Κάθε τερματικό στο Mininet είναι ουσιαστικά ένα κέλυφος τύπου bash συνδεδεμένο με μία ή πολλές διεπαφές δικτύου, οπότε ο ευκολότερος τρόπος αλληλεπίδρασης γίνεται μέσω της μεθόδου cmd().

Για την εκτέλεση μιας εντολής σε ένα τερματικό και την αποτύπωση του αποτελέσματος της μέσω cmd() , χρησιμοποιείται ο παρακάτω κώδικας.

```
h1 = net.get('h1')
result = h1.cmd('ifconfig')
print result
```

Σε πολλές περιπτώσεις είναι εφικτή η εκτέλεση μιας εντολής στο παρασκήνιο, η διακοπή της και η αποθήκευση του αποτελέσματος της σε ένα αρχείο, όπως φαίνεται από τον παρακάτω κώδικα.

```
from time import sleep
...
print "Starting test..."
h1.cmd('while true; do date; sleep 1; done > /tmp/date.out &')
sleep(10)
print "Stopping test"
h1.cmd('kill %while')
print "Reading output"
f = open('/tmp/date.out')
lineno = 1
for line in f.readlines():
    print "%d: %s" % ( lineno, line.strip() )
    lineno += 1
f.close()
```

4.6 Σύστημα Αρχείων

Από προεπιλογή, τα τερματικά στο Mininet διαμοιράζονται το σύστημα αρχείων root του συστήματος του υποκειμένου διακομιστή. Συνήθως θεωρείται θετικό, καθώς η δημιουργία ξεχωριστού συστήματος φακέλων για κάθε εικονικό τερματικό του Mininet είναι αρκετά χρονοβόρα και δύσκολη διαδικασία.

Επίσης, δεν υφίσταται σχεδόν ποτέ ανάγκη αντιγραφής δεδομένων μεταξύ τερματικών του Mininet, διότι έχουν ήδη δημιουργηθεί.

Όμως, η κοινή χρήση των φακέλων έχει το μειονέκτημα ότι εάν χρειαστεί συγκεκριμένη ρύθμιση παραμέτρων (httpd) τότε πρέπει να δημιουργηθούν νέα αρχεία παραμετροποίησης για κάθε τερματικό του Mininet. Επίσης, μπορεί να δημιουργηθούν συγκρούσεις αρχείων, έχοντας δημιουργήσει το ίδιο αρχείο στον ίδιο φάκελο σε πολλαπλά τερματικά.

4.7 Μέθοδοι Παραμετροποίησης των Τερματικών

Τα τερματικά του Mininet παρέχουν έναν αριθμό μεθόδων για την παραμετροποίηση του δικτύου.

IP() : Επιστρέφει την διεύθυνση IP ενός τερματικού ή συγκεκριμένης διεπαφής.

MAC() : Επιστρέφει την διεύθυνση MAC ενός τερματικού ή συγκεκριμένης διεπαφής.

setARP() : Προσθέτει μια στατική εγγραφή ARP στην κρυφή μνήμη ARP ενός τερματικού.

setIP() : Ορίζει την διεύθυνση IP για ένα τερματικό ή μία συγκεκριμένη διεπαφή.

setMAC() : Ορίζει την διεύθυνση MAC για ένα τερματικό ή μία συγκεκριμένη διεπαφή.

Για παράδειγμα,

```
print "Host", h1.name, "has IP address", h1.IP(), "and MAC address", h1.MAC()
```

4.8 Διεπαφή Γραμμής Εντολών Mininet (Command-Line Interface - CLI)

Το Mininet περιλαμβάνει μία CLI το οποίο μπορεί να επικληθεί σε ένα δίκτυο, παρέχει μια ποικιλία χρήσιμων εντολών, παρουσιάζει την ικανότητα δημιουργίας πολλαπλών παραθύρων, το καθένα με ανεξάρτητη είσοδο-έξοδο (μέσω του συστήματος X Window System), και την εκτέλεση εντολών σε ξεχωριστούς κόμβους στο δίκτυο. Η διεπαφή γραμμής εντολών μπορεί να κληθεί με το CLI(), όπως φαίνεται παρακάτω.

```
from mininet.topo import SingleSwitchTopo
from mininet.net import Mininet
from mininet.cli import CLI

net = Mininet(SingleSwitchTopo(2))
net.start()
CLI(net)
net.stop()
```

Η εκκίνηση της CLI είναι χρήσιμη για την αποσφαλμάτωση του δικτύου, καθώς επιτρέπει την προβολή της τοπολογίας του δικτύου, τον έλεγχο της συνδεσιμότητας και την αποστολή εντολών σε μεμονωμένα τερματικά.

```
*** Starting CLI:
mininet> net
c0
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (0/2 lost)
mininet> h1 ip link show
746: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
749: h1-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
    link/ether d6:13:2d:6f:98:95 brd ff:ff:ff:ff:ff:ff
```

4.9 Διεπαφή Προγραμματισμού Εφαρμογών Mininet (Application Program Interface - API)

Υπάρχουν αρκετές κλάσεις γλώσσας Python που αποτελούν την API του Mininet, όπως οι Topo, Mininet, Host, Switch, Link και οι κλάσεις αυτών. Αυτές οι κλάσεις μπορούν να διαχωριστούν σε επίπεδα, καθώς οι API υψηλού επιπέδου διαμορφώνονται χρησιμοποιώντας τις API των χαμηλότερων επιπέδων.

Η API του Mininet διαμορφώνεται σε τρία κύρια επίπεδα:

API Χαμηλού Επιπέδου (Low-level API). Αποτελείται από τις κύριες κλάσεις των κόμβων και συνδέσμων, όπως οι Host, Switch, Link και τις κλάσεις αυτών, οι οποίες μπορούν να αρχικοποιηθούν ανεξάρτητα και να χρησιμοποιηθούν για την δημιουργία ενός δικτύου. Βέβαια, η δημιουργία ενός δικτύου μόνο με τις κλάσεις του επιπέδου αυτού θεωρείται μια δύσχρηστη διαδικασία.

API Μεσαίου Επιπέδου (Mid-level API). Η API μεσαίου επιπέδου προσθέτει αντικείμενο τύπου Mininet το οποίο λειτουργεί επιπρόσθετα στους κόμβους και στους συνδέσμους. Παρέχει έναν αριθμό μεθόδων, όπως addHost(), addSwitch(), addLink(), που εξυπηρετούν

την προσθήκη κόμβων και συνδέσμων στο δίκτυο, αλλά και την παραμετροποίηση του δικτύου, την εκκίνηση, τον τερματισμό (συγκεκριμένα οι start(), stop()).

API Υψηλού Επιπέδου (High-level API). Προσθέτει ρυθμίσεις διαμόρφωσης της τοπολογίας. Συγκεκριμένα η κλάση Topo παρέχει την δυνατότητα δημιουργίας προτύπου τοπολογίας που μπορεί να επαναχρησιμοποιηθεί και να παραμετροποιηθεί. Αυτά τα πρότυπα εκτελούνται από την γραμμή εντολών μέσω της εντολής mn.

Γενικά, ο απευθείας έλεγχος των κόμβων και των συνδέσμων γίνεται μέσω του χαμηλού επιπέδου, ενώ η εκκίνηση ή διακοπή ενός δικτύου γίνεται μέσω του μεσαίου επιπέδου. Παρακάτω φαίνονται παραδείγματα δημιουργίας δικτύων χρησιμοποιώντας κάθε ένα από τα επίπεδα API.

API Χαμηλού Επιπέδου:

```
h1 = Host( 'h1' )
h2 = Host( 'h2' )
s1 = OVSSwitch( 's1', inNamespace=False )
c0 = Controller( 'c0', inNamespace=False )
Link( h1, s1 )
Link( h2, s1 )
h1.setIP( '10.1/8' )
h2.setIP( '10.2/8' )
c0.start()
s1.start( [ c0 ] )
print h1.cmd( 'ping -c1', h2.IP() )
s1.stop()
c0.stop()
```

API Μεσαίου Επιπέδου:

```
net = Mininet()
h1 = net.addHost( 'h1' )
h2 = net.addHost( 'h2' )
s1 = net.addSwitch( 's1' )
c0 = net.addController( 'c0' )
net.addLink( h1, s1 )
net.addLink( h2, s1 )
net.start()
print h1.cmd( 'ping -c1', h2.IP() )
CLI( net )
net.stop()
```

API Υψηλού Επιπέδου:

```
class SingleSwitchTopo( Topo ):
    "Single Switch Topology"
    def build( self, count=1 ):
        hosts = [ self.addHost( 'h%d' % i )
                  for i in range( 1, count + 1 ) ]
        s1 = self.addSwitch( 's1' )
        for h in hosts:
            self.addLink( h, s1 )

net = Mininet( topo=SingleSwitchTopo( 3 ) )
net.start()
CLI( net )
net.stop()
```

Η API μεσαίου επιπέδου είναι πιο απλή διότι δεν απαιτείται δημιουργία κλάσης που να αφορά την τοπολογία. Το χαμηλό και το μεσαίο επίπεδο είναι ευέλικτα, αλλά η επαναχρησιμοποίηση τους είναι περισσότερο δύσχρηστη, σε αντίθεση με το υψηλό επίπεδο.

4.10 Μέτρηση Απόδοσης

Σε αυτήν την ενότητα παρουσιάζονται τα σημαντικότερα εργαλεία μέτρησης της απόδοσης του λογισμικού Mininet με τις αντίστοιχες εντολές τους.

- Εύρος ζώνης (bwm-ng, ethstats)
- Καθυστέρηση (μέσω της χρήσης της εντολής ping)
- Ουρές Αναμονής (χρήση της εντολής tc που περιλαμβάνεται στο monitor.py)
- Στατιστικά TCP (tcp_probe)
- Χρήση CPU (top, cpuacct)

4.11 Πρωτόκολλο OpenFlow και Προσαρμοσμένη Δρομολόγηση

Ένα από τα πιο σημαντικά χαρακτηριστικά του Mininet είναι ότι χρησιμοποιεί το SDN. Χρησιμοποιώντας το πρωτόκολλο OpenFlow και σχετικά εργαλεία, είναι δυνατός ο προγραμματισμός των μεταγωγέων έτσι ώστε να λαμβάνουν αποφάσεις με την λήψη εισερχόμενων πακέτων. Το OpenFlow καθιστά εξομοιωτές όπως το Mininet πολύ χρήσιμους, διότι τα μοντέλα συστήματος δικτύου, συμπεριλαμβανομένου της προσαρμοζόμενης προώθησης πακέτων με την χρήση του OpenFlow, μπορούν εύκολα να μεταφερθούν σε υλικούς μεταγωγείς OpenFlow για λειτουργίες σχετικά με τον ρυθμό γραμμής.

4.11.1 Ελεγκτές OpenFlow

Εάν εκκινηθεί το Mininet χωρίς να διευκρινίζεται ο ελεγκτής, χρησιμοποιείται από προεπιλογή ο ελεγκτής τύπου ovsc. Η ισοδύναμη εντολή είναι

```
$ sudo mn --controller ovsc
```

Ο συγκεκριμένος τύπος ελεγκτή υλοποιεί έναν απλό μεταγωγέα μάθησης Ethernet, και μπορεί να υποστηρίξει μέχρι 16 μεταγωγείς.

Όταν εκτελεστεί η κλάση Mininet() χωρίς διευκρίνιση κλάσης ελεγκτή, χρησιμοποιείται η προεπιλεγμένη κλάση Controller() για την δημιουργία ενός ελεγκτή τύπου Stanford/OpenFlow. Αντιθέτως, παρέχεται η δυνατότητα χρήσης ενός διαφορετικού τύπου ελεγκτή, δημιουργώντας μια υποκατηγορία του Controller() και μεταφέροντας την στα

αρχεία συστήματος του Mininet. Παρακάτω φαίνεται ένα παράδειγμα δημιουργίας ενός ελεγκτή POX.

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller
from mininet.topo import SingleSwitchTopo
from mininet.log import setLogLevel

import os

class POXBridge( Controller ):
    "Custom Controller class to invoke POX forwarding.l2_learning"
    def start( self ):
        "Start POX learning switch"
        self.pox = '%s/pox/pox.py' % os.environ[ 'HOME' ]
        self.cmd( self.pox, 'forwarding.l2_learning &' )
    def stop( self ):
        "Stop POX"
        self.cmd( 'kill %' + self.pox )

controllers = { 'poxbridge': POXBridge }

if __name__ == '__main__':
    setLogLevel( 'info' )
    net = Mininet( topo=SingleSwitchTopo( 2 ), controller=POXBridge )
    net.start()
    net.pingAll()
    net.stop()
```

4.11.2 Εξωτερικοί Ελεγκτές OpenFlow

Στην παρούσα διπλωματική εργασία γίνεται χρήση του εξωτερικού ελεγκτή Beacon βασισμένος στην γλώσσα προγραμματισμού Java όπως έχει περιγραφεί σε άλλο κεφάλαιο. Συνεπώς, καθίσταται αδύνατη η δημιουργία υπο-κλάσης της Controller(), διότι το Mininet και οι ελεγκτές του χρησιμοποιούν κλάσεις που είναι βασισμένες στην γλώσσα προγραμματισμού Python. Το Mininet παρέχει την δυνατότητα σύνδεσης του εικονικού δικτύου με έναν ήδη υπάρχον ελεγκτή, στην περίπτωση μας τον Beacon, ο οποίος μπορεί να βρίσκεται στο ίδιο τοπικό δίκτυο, σε άλλη εικονική μηχανή ή ακόμα και στον υπολογιστή του χρήστη.

Η κλάση RemoteController λειτουργεί σαν μεσολαβητής για έναν ελεγκτή που λειτουργεί σε οποιοδήποτε σημείο του δικτύου ελέγχου. Όμως, η εκκίνηση και ο τερματισμός

του πρέπει να γίνουν μηχανικά ή μέσω άλλου μηχανισμού εκτός ελέγχου του Mininet. Μερικά παραδείγματα της χρήσης της κλάσης RemoteController() απεικονίζονται παρακάτω.

```
from functools import partial
net = Mininet( topo=topo, controller=partial( RemoteController,
ip='127.0.0.1', port=6633



net = Mininet( topo=topo, controller=lambda name: RemoteController( name,
ip='127.0.0.1' ) )



net = Mininet( topo=topo, controller=None)
net.addController( 'c0', controller=RemoteController, ip='127.0.0.1',
port=6633 )
```

Σημειώνεται ότι το controller αποτελεί μια συνάρτηση δομής (constructor) και όχι ένα αντικείμενο. Μπορεί να δημιουργηθεί μία προσαρμοζόμενη συνάρτηση δομής χρησιμοποιώντας τα ορίσματα, partial, lambda, δημιουργώντας μία συνάρτηση που θα επιστρέφει τον ελεγκτή ως αντικείμενο ή εισάγοντας τον ελεγκτή ως υποκατηγορία της κλάσης RemoteController().

Επίσης, είναι δυνατή η δημιουργία πολλαπλών ελεγκτών και μιας υποκατηγορίας της κλάσης Switch(), η οποία θα είναι υπεύθυνη για την σύνδεση των διαφορετικών ελεγκτών.

```
c0 = Controller( 'c0' )  # local controller
c1 = RemoteController( 'c1', ip='127.0.0.2' )  # external controller
cmap = { 's1': c0, 's2': c1, 's3': c1 }

class MultiSwitch( OVSSwitch ):
    "Custom Switch() subclass that connects to different controllers"
    def start( self, controllers ):
        return OVSSwitch.start( self, [ cmap[ self.name ] ] )
```

Επιπλέον, είναι δυνατός ο καθορισμός του εξωτερικού ελεγκτή από την γραμμή εντολών. [10]

```
$ sudo mn --controller remote,ip=192.168.51.101
```

5 ANYCAST

5.1 Συνήθεις Μέθοδοι Επικοινωνίας Πρωτοκόλλου IPv4

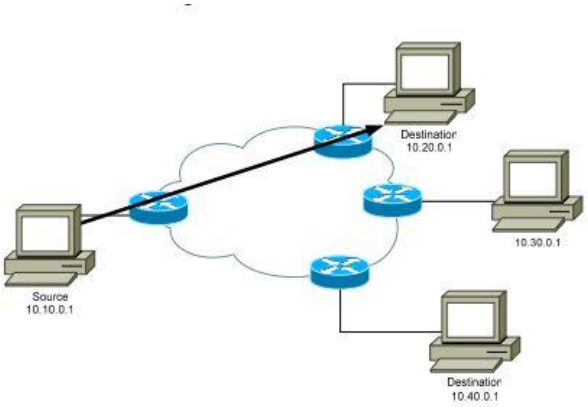
Καθώς ο κόσμος του διαδικτύου μεγαλώνει και τα συστήματα και οι υπηρεσίες δικτύου κυριαρχούν μέσα στον κόσμο των επιχειρήσεων, προκύπτουν συνεχώς πολλές υπηρεσίες με υψηλές απαιτήσεις ζήτησης διαθεσιμότητας. Συνεπώς, έχουν αυξηθεί οι απαιτήσεις όσον αφορά την αξιοπιστία των δικτύων πάνω στις οποίες βασίζονται αυτές οι υπηρεσίες.

Για αντιμετωπιστεί αυτό το ζήτημα αυξημένης ζήτησης διαθεσιμότητας υπηρεσιών του διαδικτύου, έχουν αναπτυχθεί αρκετές τεχνικές δρομολόγησης. Οι τρεις πιο γνωστές μέθοδοι δημιουργίας μιας σύνδεσης στο IPv4 πρωτόκολλο είναι οι μονής διανομής (unicast), πολλαπλής διανομής (multicast) και εκπομπής (broadcast). Το πρωτόκολλο IPv6 σύστησε μία τέταρτη μέθοδο επικοινωνίας, η ένα-προς-το-κοντινότερο (anycast), που αργότερα χρησιμοποιήθηκε από το IPv4.

5.1.1 Unicast

Το unicast ορίζεται ως μία δρομολόγηση πακέτων «ένα-προς-ένα» μεταξύ πηγής και προορισμού. Όπως φαίνεται από την εικόνα 5.1, ο διακομιστής (server) αναγνωρίζεται από μια μοναδική διεύθυνση IP (10.20.0.1), η οποία περιέχεται στην επικεφαλίδα του πακέτου που στέλνεται από τον πελάτη (client) και αντίστοιχα ο πελάτης αναγνωρίζεται από μια μοναδική διεύθυνση IP που περιέχεται στην επικεφαλίδα του πακέτου. Σε αυτήν την περίπτωση, το δίκτυο θα επιχειρήσει να παραδώσει το πακέτο στον προοριζόμενο

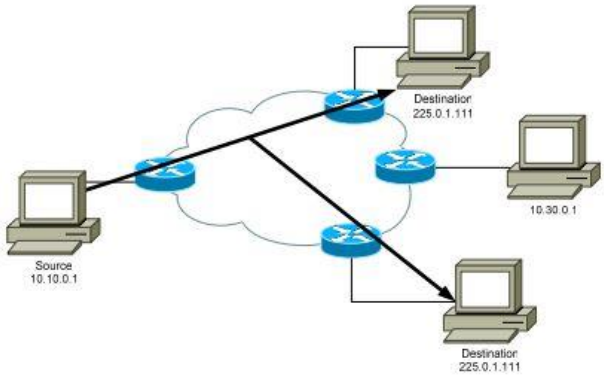
διακομιστή, βασιζόμενο στις πληροφορίες των πινάκων δρομολόγησης των δρομολογητών του δικτύου.



Εικόνα 5.1 Τεχνική Unicast [11]

5.1.2 Multicast

Το Multicast ορίζεται ως μια ροή πακέτων μεταξύ πηγής και ενός ή περισσότερων προορισμών με την αποστολή ενός αντιγράφου του πακέτου σε μια διεύθυνση ομάδας (για παράδειγμα 225.0.1.111), όπως φαίνεται στην εικόνα 5.2. Το δίκτυο των δρομολογητών θα παραδώσει τα πακέτα στους προοριζόμενους πελάτες που προσδιορίζονται από μια διεύθυνση ομάδας multicast. Χρησιμοποιείται ένας πίνακας multicast δρομολόγησης για να προωθηθεί η κίνηση μέσα στο δίκτυο. Οι διευθύνσεις multicast αποτελούνται μόνο από διευθύνσεις IP προορισμού και όχι διεύθυνση πηγής.



Εικόνα 5.2 Τεχνική Multicast [11]

5.1.3 Broadcast

Η τρίτη τεχνική σύνδεσης IPv4 είναι το broadcast. Οι διευθύνσεις broadcast υποστηρίζουν εξεύρεση υπηρεσιών ή διακομιστών. Ένα παράδειγμα πρωτοκόλλου που χρησιμοποιεί το broadcast είναι το Πρωτόκολλο Επίλυσης Διευθύνσεων (Address Resolution Protocol – ARP). Τα πακέτα που αποστέλλονται σε μια διεύθυνση IP broadcast παραδίδονται σε όλα τα τερματικά ενός υλικού δικτύου ή υποδικτύου IP. Οι διευθύνσεις broadcast χρησιμοποιούνται μόνο στο πεδίο διεύθυνσης προορισμού μιας επικεφαλίδας IP καθώς είναι αδύνατο για ένα μοναδικό πακέτο να προέρχεται από περισσότερες από μία διεπαφές. Είναι προφανές ότι η τεχνική broadcast πραγματοποιείται μόνο τοπικά σε ένα υποδίκτυο.

5.2 Μέθοδος Επικοινωνίας Anycast

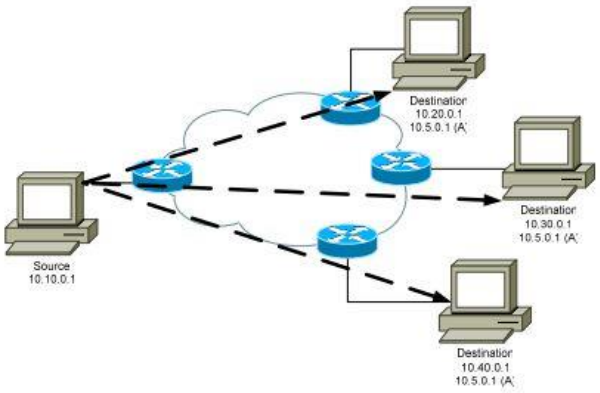
Η τεχνική anycast είναι ροή πακέτων μεταξύ ενός πελάτη και του κοντινότερου προοριζόμενου διακομιστή, ο οποίος ορίζεται από μια διεύθυνση anycast. Συνήθως, υλοποιείται με το Πρωτόκολλο Πύλης Συνόρου (Border Gateway Protocol – BGP). Το BGP είναι ένα πρωτόκολλο δρομολόγησης πακέτων και ανταλλαγής πληροφοριών μεταξύ αυτόνομων συστημάτων στο διαδίκτυο. Ανήκει στην κατηγορία των πρωτοκόλλων διανύσματος αποστάσεων και ως εκ τούτου οι αποστάσεις εκφράζονται σε βήματα (hops). [14]

Σήμερα, οι δύο κύριες ερευνητικές περιοχές της μεθόδου επικοινωνίας Anycast είναι οι επιπέδου εφαρμογής (application-layer) και επιπέδου δικτύου (network-layer) ή IP. Βέβαια, η μη επίγνωση των αλλαγών στην τοπολογία και στις συνθήκες φορτίου εμποδίζει την ανάπτυξη του application-layer Anycast. Επιπρόσθετα, η μετάφραση των διευθύνσεων είναι δαπανηρή όταν υπάρχει βαρύ φορτίο στο δίκτυο. Το IP Anycast ξεπερνά αυτές τις δυσκολίες με μια απλή υλοποίηση ρυθμίζοντας τις παραμέτρους στους διακομιστές και στην υποδομή δρομολόγησης (δρομολογητές, μεταγωγείς) έτσι ώστε να υποστηρίζεται η υπηρεσία Anycast. Παρόλα αυτά, ο υπάρχων δικτυακός εξοπλισμός λειτουργούν ως μαύρα κουτιά, χωρίς την δυνατότητα επέκτασης για την ανάπτυξη της IP Anycast.

Η ιδέα πίσω από το anycast είναι ότι εάν ένας πελάτης επιθυμεί να στείλει πακέτα σε έναν εξυπηρετητή προσφέροντας μια υπηρεσία ή εφαρμογή, δεν είναι σημαντική η επιλογή του εξυπηρετητή. Μια μοναδική διεύθυνση anycast ανατίθεται σε έναν ή περισσότερους εξυπηρετητές. Ένας πελάτης στέλνει πακέτα στον εξυπηρετητή τοποθετώντας την διεύθυνση anycast στην επικεφαλίδα του πακέτου. Τότε, το δίκτυο των δρομολογητών θα παραδώσει το πακέτο στον κοντινότερο εξυπηρετητή που φέρνει την διεύθυνση anycast προορισμού, και αυτός με την σειρά του θα απαντήσει. [11]

Στο υπόλοιπο κεφάλαιο θα αναφερθούν οι ορισμοί διεύθυνση Anycast (Anycast Address) και πελάτης (Client). Ως διεύθυνση Anycast σε αυτήν την εργασία αναφέρεται ως η κοινή διεύθυνση IP που χρησιμοποιείται από μια ομάδα μελών ή εξυπηρετητών. Ως πελάτης αναφέρεται ο ξενιστής που προσπαθεί να επικοινωνήσει στέλνοντας πακέτα προς την διεύθυνση Anycast.

Από όταν προτάθηκε το 1993, θεωρείται μια δυνατή μέθοδος δρομολόγησης και παράδοσης πακέτων IP, καθώς και μια αποτελεσματική, διαφανής, ισχυρή ανακάλυψη υπηρεσιών (service discovery). Υλοποιήθηκε αρχικά από το πρωτόκολλο IPv6 και μετά χρησιμοποιήθηκε από το IPv4.



Εικόνα 5.3 Τεχνική Anycast [11]

Ένα από τα χαρακτηριστικά της μεθόδου Anycast είναι ότι πολλαπλοί κόμβοι διαμορφώνονται έτσι ώστε να λαμβάνουν πακέτα χρησιμοποιώντας την ίδια διεύθυνση IP, αλλά έχουν μια μοναδική διεύθυνση για λόγους διαχείρισης. Βέβαια, τα πακέτα Anycast μπορεί να χαθούν όπως μπορεί να συμβεί σε κάθε τεχνική δρομολόγησης, αφού δεν είναι ειδικά χαρακτηρισμένα.

Συνήθως, ένας μόνο εξυπηρετητής λαμβάνει ένα πακέτο, όμως δεν συμβαίνει πάντα. Είναι πιθανό εάν σταλθούν διαδοχικά πακέτα από έναν πελάτη προς μια διεύθυνση Anycast, να ληφθούν από διαφορετικούς εξυπηρετητές. Επίσης, ο εξυπηρετητής που λαμβάνει ένα συγκεκριμένο πακέτο καθορίζεται αποκλειστικά και μόνο από το πρωτόκολλο δρομολόγησης unicast του δικτύου, διότι δεν υπάρχει πίνακας Anycast δρομολόγησης ισοδύναμος με έναν αντίστοιχο πίνακα δρομολόγησης κίνησης multicast.

5.3 Διευθύνσεις Anycast

Υπάρχουν πολλοί τρόποι υποστήριξης της χρήσης διευθύνσεων Anycast, κάποιοι από τους οποίους χρησιμοποιούν μικρά κομμάτια του υπάρχοντος χώρου διευθύνσεων, ενώ άλλοι απαιτούν την ανάθεση μιας ειδικής κλάσης διευθύνσεων IP.

Το κύριο πλεονέκτημα της χρήσης του υπάρχοντος χώρου διευθύνσεων είναι ότι διευκολύνει την δρομολόγηση. Σαν παράδειγμα αναφέρεται μία περίπτωση όπου μια ιστοσελίδα αναθέτει ένα σύνολο των διευθύνσεων του υποδικτύου της σαν διευθύνσεις Anycast. Εάν, όπως αναμένουν οι ειδικοί, οι διαδρομές Anycast αντιμετωπίζονται σαν διαδρομές ξενιστών από τα πρωτόκολλα δρομολόγησης, οι διευθύνσεις Anycast δε θα απαιτούσαν ειδική διαφήμιση έξω από την ιστοσελίδα, διότι οι διαδρομές των ξενιστών μπορούσαν να ενσωματωθούν με την διαδρομή του δικτύου. Αντίθετα, εάν οι διευθύνσεις Anycast υποστηρίζονταν από ξενιστές έξω από το δίκτυο, τότε θα έπρεπε να διαφημίζονται χρησιμοποιώντας διαδρομές ξενιστών. Ένα βασικό μειονέκτημα αυτής της προσέγγισης είναι ότι δεν υπάρχει εύκολος τρόπος για πρωτόκολλα όπως το Πρωτόκολλο Ελέγχου Μεταφοράς (Transmission Control Protocol – TCP) να ανακαλύψουν ότι μια διεύθυνση είναι στην ουσία διεύθυνση Anycast. Επίσης, είναι περισσότερο δύσκολη η υποστήριξη γνωστών διευθύνσεων Anycast στο διαδίκτυο, διότι το διαδίκτυο μπορεί να καθιερώσει μια συγκεκριμένη διεύθυνση Anycast ως μια λογική διεύθυνση του εξυπηρετητή του Συστήματος Ονομάτων Τομέων (Domain Name System – DNS). Έτσι, το λογισμικό του ξενιστή θα μπορούσε να διαμορφωθεί ώστε να στέλνει αιτήματα DNS στην διεύθυνση Anycast DNS.

Χρησιμοποιώντας μια ξεχωριστή κλάση διευθύνσεων καθιστά εύκολη την διευκρίνιση αν μια διεύθυνση είναι διεύθυνση Anycast, όπως και την υποστήριξη γνωστών διευθύνσεων Anycast. Όμως, η δρομολόγηση είναι δυσκολότερη λόγω της συνεχούς ανίχνευσης των διαδρομών Anycast από τα πρωτόκολλα δρομολόγησης.

Μια ενδιάμεση προσέγγιση είναι η διαμόρφωση ενός μέρους του τρέχοντος χώρου διευθύνσεων(256 διευθύνσεις κλάσης C) και η μετατροπή των διευθύνσεων δικτύου σε διευθύνσεις Anycast (αγνοώντας το μέρος των ξενιστών από τις διευθύνσεις κλάσης C). Αυτό καθιστά τις διαδρομές Anycast να μοιάζουν με διαδρομές δικτύου και έτσι ευκολότερες ως προς την διαχείριση από τα πρωτόκολλα δρομολόγησης. Όμως, ο χώρος διευθύνσεων χρησιμοποιείται αναποτελεσματικά περιορίζοντας ταυτόχρονα τον αριθμό των διευθύνσεων Anycast που μπορεί να υποστηριχθούν.

Από τα παραπάνω προκύπτει ότι προτιμάται η χρήση ξεχωριστής κλάσης διευθύνσεων IP, διότι αποφεύγεται η αποτυχία αναγνώρισης διευθύνσεων Anycast και η ανεπαρκής χρήση του χώρου διευθύνσεων. [12]

5.4 Anycast στο IPv4

Το πρωτόκολλο IPv4 δεν μπορεί να χρησιμοποιήσει την μέθοδο Anycast βέλτιστα, αλλά αξιοποιεί σημαντικά την υποδομή για κάποιες εφαρμογές. Η χρησιμοποίηση του Anycast περιορίζεται σε σύντομες ανταλλαγές πληροφοριών χωρίς σύνδεση. Για παράδειγμα, το Σύστημα Ονομάτων Τομέων (Domain Name System – DNS) θεωρείται πλέον κατάλληλο για την χρήση της μεθόδου Anycast. Χρησιμοποιεί μόνο δύο πακέτα του Πρωτοκόλλου Δεδομένων Χρήστη (User Datagram Protocol – UDP), την αίτηση και την απάντηση. Η επόμενη αίτηση μπορεί να σταλεί σε διαφορετικό εξυπηρετητή χωρίς να σημειωθεί πρόβλημα. Ένα παράδειγμα αποτελούν οι εξυπηρετητές F-root που έχουν αναπτυχθεί από την επιχείρηση μη κερδοσκοπικού χαρακτήρα Internet Systems Consortium (ISC) και χρησιμοποιούν διευθυνσιοδότηση Anycast από τον Νοέμβριο του 2002.

Άλλες χρήσεις του Anycast είναι τα Σημεία Συνάντησης Multicast (Multicast Rendezvous Points – RPs), το Πρωτόκολλο Χρόνου Δικτύου (Network Time Protocol –

NTP), Syslog (χρησιμοποιείται για αποστολή μηνυμάτων καταγραφής από συσκευές για διαχείριση συστήματος), εξαγωγή πληροφοριών ροής (Netflow της Cisco) κτλ.

Αντιθέτως, το Πρωτόκολλο Ελέγχου Μεταφοράς (Transmission Control Protocol – TCP) χρειάζεται πολύπλοκους μηχανισμούς αλλαγής κατάστασης, ώστε να χρησιμοποιήσει το Anycast. Έτσι, θεωρείται λιγότερο κατάλληλο για την χρήση του.

5.5 Πλεονεκτήματα της Μεθόδου Anycast

Η διευθυνσιοδότηση με Anycast είναι μια χρήσιμη τεχνική διανομής μιας υπηρεσίας στο δίκτυο για αρκετούς λόγους. Μερικοί από αυτούς εξηγούνται στην παρούσα ενότητα.

Μειωμένη χρήση πόρων των δρομολογητών και των συνδέσεων. Η δρομολόγηση IP μεταφέρει πακέτα μέσω της συντομότερης διαδρομής στον κοντινότερο διαθέσιμο εξυπηρετητή.

Απλουστευμένη διαμόρφωση. Το μόνο που χρειάζεται ένας πελάτης είναι μια μοναδική διεύθυνση Anycast που προσδιορίζει έναν από πολλούς πιθανούς εξυπηρετητές που προσφέρουν μια συγκεκριμένη υπηρεσία ή εφαρμογή.

Πλεονασμός δικτύου. Εάν ένας εξυπηρετητής μιας ομάδας Anycast απομακρυνθεί, τότε το δίκτυο θα παραδώσει πακέτα στον επόμενο κοντινό εξυπηρετητή Anycast. Υπάρχει έτσι μια αξιόπιστη υπηρεσία.

Εξισορρόπηση φορτίου. Οι εξυπηρετητές Anycast του δικτύου θα μπορούν να εξισορροπήσουν το φορτίο κίνησης που προέρχεται από πολλαπλούς πελάτες.

Μειωμένη καθυστέρηση. Η μέση καθυστέρηση απόκρισης δικτύου μεταξύ πελάτη και εξυπηρετητή θα μειωθεί. Το Anycast αποτελεί μια βελτιστοποίηση για όσες υπηρεσίες έχουν καθυστερήσεις συναλλαγών στενά συνδεδεμένες με καθυστέρησης απόκρισης μεταξύ πελάτη και εξυπηρετητή.

Ασφάλεια. Οι επιθέσεις άρνησης των υπηρεσιών (Denial-of-Service – DoS) ή οποιαδήποτε κακόβουλη κίνηση διανέμεται με κίνηση ερωτήσεων στον πελάτη. Αυτό μπορεί να οδηγήσει στην απομόνωση των επιπτώσεων μιας επίθεσης σε μια υπηρεσία, σε συγκεκριμένα σημεία του διαδικτύου. Επίσης, το πρότυπο μιας κίνησης επιθέσεων που πραγματοποιείται σε διαφορετικούς κόμβους υπηρεσιών μπορεί να οδηγήσει στον εντοπισμό των σημείων από όπου προέρχονται οι επιθέσεις.

5.6 Θέματα Ασφάλειας

Υπάρχουν τουλάχιστον δύο ζητήματα ασφαλείας όταν χρησιμοποιείται η τεχνική Anycast. Πρώτον, ένας κακόβουλος εξυπηρετητής μπορεί να μεταφέρει πακέτα δεδομένων από έναν άλλον εξυπηρετητή στον εαυτό του. Σε μία περίπτωση δυναμικής δρομολόγησης, θα πρέπει να ισχύουν ειδικά μέτρα όπως κατάλληλο φιλτράρισμα ή έλεγχος των ενημερώσεων των διαδρομών, για να αποφευχθεί η διαφήμιση πληροφοριών δρομολόγησης από ανεπιθύμητους εξυπηρετητές.

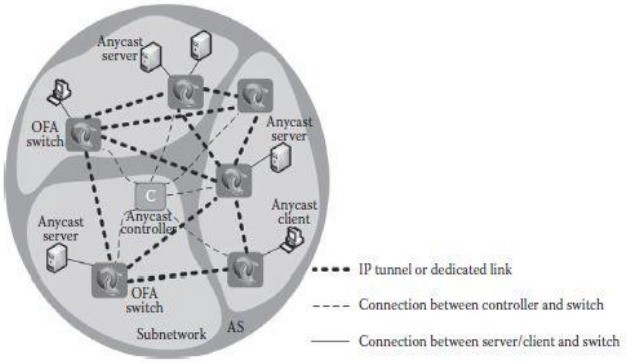
Επίσης, οι ξενιστές μπορούν να υποκλέψουν και να απαντήσουν σε αιτήσεις Anycast με ανακριβείς πληροφορίες. Εφόσον είναι αδύνατη η επαλήθευση της χρησιμοποίησης της διεύθυνσης Anycast, δεν υπάρχει τρόπος να ελεγχθεί εάν ο συγκεκριμένος ξενιστής ανήκει ή όχι στην ομάδα που χρησιμοποιεί την διεύθυνση Anycast.

5.7 Υλοποίηση της υπηρεσίας Anycast στο OpenFlow

Όπως έχει αναλυθεί σε προηγούμενο κεφάλαιο, το SDN έχει την δυνατότητα ελέγχου των δικτύων προγραμματιστικά διαχωρίζοντας τα επίπεδα ελέγχου και δεδομένων, το οποίο αποτελεί μια υποσχόμενη τεχνική για την ανάπτυξη νέων εφαρμογών και υπηρεσιών στα δίκτυα. Συνεπώς, όποια ζητήματα υπήρχαν στην ευρεία ανάπτυξη των application-layer Anycast και IP Anycast μπορούν να λυθούν με αυτήν την αρχιτεκτονική.

5.7.1 Αρχιτεκτονική Συστήματος

Η προτεινόμενη αρχιτεκτονική για ένα σύστημα Anycast βασισμένο στην τεχνολογία OpenFlow αποτελείται από εξυπηρετητές Anycast που παρέχουν υπηρεσίες Anycast, ο πελάτης Anycast, οι μεταγωγείς OpenFlow Anycast (OFA) και ο ελεγκτής Anycast, με τον οποίο είναι άμεσα ή έμμεσα συνδεδεμένοι μέσω των μεταγωγέων OFA οι εξυπηρετητές και οι πελάτες. Σε αυτήν την ενότητα, μελετάται η περίπτωση ενδοτομεακών δικτύων ενός αυτόνομου συστήματος, όπου οι μεταγωγείς OFA υλοποιούνται στα υποδίκτυα του αυτόνομου συστήματος και συνδέονται μέσω μιας σήραγγας IP ή ειδικού συνδέσμου σχηματισμού ενός δικτύου επιπέδου 2. Ο ελεγκτής Anycast έχει συνδέσμους IP με τους μεταγωγείς OFA.

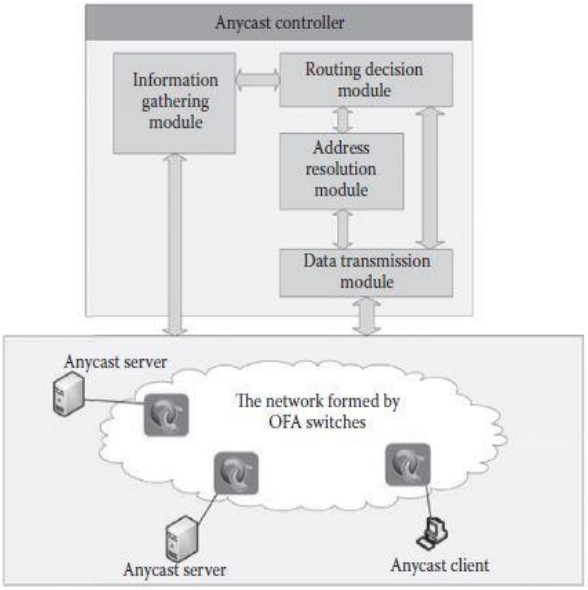


Εικόνα 5.4Αρχιτεκτονική Συστήματος εφαρμογής Anycast σε Αυτόνομο Σύστημα βασισμένο σε OpenFlow[5]

Ένα σύνολο διευθύνσεων ανατίθεται για τις υπηρεσίες Anycast. Ο ελεγκτής συγκεντρώνει τις πληροφορίες των μεταγωγέων OFA και των συνδεδεμένων εξυπηρετητών/ πελατών και ανάλογα με τις μετρήσεις απόδοσης λαμβάνει αποφάσεις δρομολόγησης για αιτήσεις Anycast. Αυτές οι αποφάσεις προστίθενται σαν καταχωρήσεις στον πίνακα ροής κάθε μεταγωγέα OFA. Έπειτα, οι μεταγωγείς OFA προωθούν τα πακέτα ανάλογα με τις καταχωρήσεις αυτές. Ακόμα, οι μεταγωγείς OFA έχουν την δυνατότητα διαφήμισης του προθέματος των διευθύνσεων Anycast. [5]

5.7.2 Σχεδιασμός Ελεγκτή Anycast

Το πρωτόκολλο OpenFlow επιτυγχάνει υψηλή ευελιξία στην δρομολόγηση των ροών ενός δικτύου και την επιλογή αλλαγής της συμπεριφοράς της κίνησης του δικτύου διαχωρίζοντας το επίπεδο ελέγχου στις συσκευές δικτύου από το επίπεδο δεδομένων, οδηγώντας στην σημασία του ελεγκτή στις τεχνολογίες OpenFlow. Σε αυτήν την ενότητα, ο ελεγκτής είναι σχεδιασμένος να υποστηρίζει υπηρεσίες Anycast, όπως φαίνεται στην εικόνα 5.5. [5]



Εικόνα 5.5 Μοντέλο ελεγκτή Anycast [5]

5.7.3 Δομοστοιχείο Συλλογής Πληροφοριών (Information Gathering Module)

Ο ελεγκτής Anycast πρέπει να έχει επίγνωση των αλλαγών της τοπολογίας και των αλλαγών των φορτίων στους εξυπηρετητές/πελάτες και στους συνδέσμους του δικτύου, προκειμένου να παίρνει κατάλληλες αποφάσεις σε στρατηγικές δρομολόγησης. Εάν η επιλογή εξυπηρετητή εξαρτάται από τον αριθμό των βημάτων από τον πελάτη που στέλνει αιτήσεις προς τον εξυπηρετητή Anycast, ο ελεγκτής πρέπει να έχει επίγνωση της τοπολογίας του δικτύου. Από την άλλη, εάν πρέπει να υπολογιστεί η εξισορρόπηση του φορτίου στην λήψη αποφάσεων για την δρομολόγηση, τότε ο ελεγκτής πρέπει να έχει επίγνωση του φορτίου σε κάθε σύνδεσμο δικτύου, σε εξυπηρετητές και σε πελάτες.

Γενικά, ο ελεγκτής μπορεί να ανιχνεύει την τοπολογία του δικτύου και την κατάσταση φορτίου σε εξυπηρετητές, πελάτες και συνδέσμους δικτύου επικοινωνώντας με τους μεταγωγείς OFA που βασίζονται στο πρωτόκολλο OpenFlow. Οι μεταγωγείς αναφέρουν τις αλλαγές των θυρών στον ελεγκτή όταν λαμβάνουν ένα μήνυμα ασύγχρονου τύπου Port-Status. Έτσι, ανιχνεύεται τυχόν αλλαγή στην τοπολογία του δικτύου. Ο ελεγκτής μπορεί να λάβει πληροφορίες κατάστασης των μεταγωγέων OFA στέλνοντας ένα μήνυμα τύπου ελεγκτή-προς-τον-μεταγωγέα Read-State. Με αυτόν τον τρόπο ο ελεγκτής έχει τις απαραίτητες πληροφορίες σχετικά με την κατάσταση φορτίου στις δικτυακές συνδέσεις ολόκληρου του δικτύου. [5]

5.7.4 Δομοστοιχείο Απόφασης Δρομολόγησης (Routing Decision Module)

Αυτό το δομοστοιχείο λαμβάνει αποφάσεις για τα πακέτα που δεν ταιριάζουν με τις καταχωρήσεις στους πίνακες ροής των μεταγωγέων OFA. Ακολουθούν οι δύο στρατηγικές μέτρησης δρομολόγησης: μέτρηση βημάτων και φορτίο σύνδεσης.

5.7.4.1 Στρατηγική Δρομολόγησης με Μέτρηση Βημάτων

Η συγκεκριμένη στρατηγική επιλέγει τον κοντινότερο εξυπηρετητή Anycast για να απαντήσει στην αίτηση, όπου ο όρος κοντινότερος υπολογίζεται από την μέτρηση βημάτων από τον εξυπηρετητή προς τον πελάτη από όπου προέρχεται η αίτηση. Είναι προφανές ότι είναι σημαντικός ο υπολογισμός των βημάτων μεταξύ δύο οποιωνδήποτε μεταγωγέων OFA στο δίκτυο. Εάν ολόκληρη η τοπολογία του δικτύου αποτυπώνεται σε ένα γράφημα, το πρόβλημα μετασχηματίζεται σε υπολογισμό συντομότερης διαδρομής μεταξύ δύο οποιωνδήποτε σημείων του γραφήματος. Ένα παράδειγμα αλγορίθμου που χρησιμοποιεί την παραπάνω στρατηγική είναι ο Floyd-Warshall. [5]

5.7.4.2 Στρατηγική Δρομολόγησης με Φορτία Σύνδεσης

Η στρατηγική δρομολόγησης που βασίζεται στα φορτία σύνδεσης επιλέγει τον καλύτερο εξυπηρετητή για να ανταποκριθεί στην αίτηση, όπου ο όρος καλύτερος αναφέρεται σε όποια σύνδεση μεταξύ εξυπηρετητή και πελάτη φέρει το βέλτιστο φορτίο. Τα φορτία σύνδεσης μπορούν να μετρηθούν ως ο αριθμός των πακέτων που μεταφέρεται από την σύνδεση ανά χρονική μονάδα, ο οποίος μπορεί να ανακτηθεί από τους μετρητές των καταχωρήσεων των πινάκων ροών των μεταγωγέων OFA. [5]

5.7.5 Δομοστοιχείο Επίλυσης Διευθύνσεων (Address Resolution Module)

Η κύρια εργασία του δομοστοιχείου επίλυσης διευθύνσεων είναι η δημιουργία πακέτων ανταπόκρισης Πρωτοκόλλου Επίλυσης Διευθύνσεων (Address Resolution Protocol – ARP) σε αιτήσεις Anycast. Ειδικότερα, ο ελεγκτής εξάγει την IP διεύθυνση Anycast από τα πακέτα αιτήσεων ARP και επικαλείται το δομοστοιχείο απόφασης δρομολόγησης ώστε να επιλεχθεί ένας εξυπηρετητής Anycast προορισμού. Το δομοστοιχείο επιστρέφει την διεύθυνση MAC του επιλεγμένου εξυπηρετητή και δημιουργεί ένα πακέτο απάντησης ARP χρησιμοποιώντας την MAC διεύθυνση προορισμού. Τέλος, το δομοστοιχείο επίλυσης διευθύνσεων επικαλείται το δομοστοιχείο μετάδοσης δεδομένων για την παράδοση της απάντησης τύπου ARP στον πελάτη.

Ένας πελάτης Anycast πρέπει να μετατρέψει την διεύθυνση IP σε διεύθυνση MAC πριν την επικοινωνία με τους εξυπηρετητές του ίδιου αυτόνομου συστήματος. Συγκεκριμένα, σύμφωνα με το πρωτόκολλο ARP, μία διεύθυνση IP μπορεί να αντιστοιχηθεί μόνο σε μία διεύθυνση MAC σε ένα αυτόνομο σύστημα. Υλοποιώντας την υπηρεσία Anycast, ίσως υπάρχουν περισσότερο από ένας εξυπηρετητές με την ίδια διεύθυνση Anycast σε ένα αυτόνομο σύστημα. Συνεπώς, ένας πελάτης που στέλνει μία αίτηση ARP, μπορεί να λάβει πολλαπλές απαντήσεις, οδηγώντας σε μια σύγκρουση επίλυσης ARP.

Η σύγκρουση επίλυσης ARP επιλύεται αλλάζοντας τον τρόπο αποστολής αιτήσεων ARP. Στην υπηρεσία Anycast, μόνο ένας εξυπηρετητής επιλέγεται να απαντήσει σε μια

αίτηση. Με την βοήθεια του ελεγκτή, μπορεί να βρεθεί η διαδρομή από τον πελάτη στον συγκεκριμένο εξυπηρετητή. Τότε, ο ελεγκτής μπορεί να απαντήσει στην αίτηση ARP εκ μέρους του επιλεγμένου εξυπηρετητή. Για να αποφευχθεί η σύγκρουση, αυτή η μέθοδος εξασφαλίζει ότι μόνο μία απάντηση ARP μπορεί να δημιουργηθεί κατά την διαδικασία της επίλυσης ARP.

Η καταχώρηση στον πίνακα ροής όλων των μεταγωγέων OFA για το πακέτο αίτησης ARP πρέπει να έχει προστεθεί προηγουμένως, ώστε να οδηγηθεί αυτό το πακέτο στον ελεγκτή, ο οποίος θα επιλέξει έναν κατάλληλο εξυπηρετητή με την βοήθεια του δομοστοιχείου λήψης απόφασης δρομολόγησης. Μετά, θα δημιουργήσει ένα πακέτο απάντησης τύπου ARP χρησιμοποιώντας στην διεύθυνση MAC του επιλεγμένου εξυπηρετητή ως την διεύθυνση πηγής MAC αυτού του πακέτου για να σταλθεί πίσω στον πελάτη. Υπό αυτές τις συνθήκες, υπάρχει μόνο μία ανταλλαγή πακέτων αίτησης και απόκρισης για να επιτευχθεί η επίλυση πρωτόκολλου ARP. [5]

5.7.6 Δομοστοιχείο Μετάδοσης Δεδομένων (Data Transmission Module)

Το συγκεκριμένο δομοστοιχείο είναι ο μεσάζων μεταξύ των μεταγωγέων, του δομοστοιχείου απόφασης δρομολόγησης και του δομοστοιχείου επίλυσης διευθύνσεων. Λαμβάνει πακέτα από τους μεταγωγείς, τα μεταφέρει σε διαφορετικές μονάδες και αντίστοιχα.

Τα πακέτα δεδομένων που δεν ταιριάζουν με τις καταχωρήσεις του πίνακα ροής ενός μεταγωγέα παραδίδονται στον ελεγκτή. Αναλύει την επικεφαλίδα του πακέτου και ανάλογα με τον τύπο πρωτοκόλλου, παραδίδει τα πακέτα στις αντίστοιχες μονάδες για περαιτέρω επεξεργασία. [5]

6 CONTEXT-AWARE ΔΡΟΜΟΛΟΓΗΣΗ

6.1 Επίγνωση περιβάλλοντος (Context-awareness)

Ο όρος Context χρησιμοποιείται για να διακρίνει και να περιγράψει την κατάσταση μιας οποιαδήποτε οντότητας που βρίσκεται στο περιβάλλον. Είναι η πληροφορία η οποία θεωρείται σημαντική για την επικοινωνία μεταξύ χρηστών και εφαρμογών, όπου η ταυτότητα, η τοποθεσία, η κατάσταση των αντικειμένων λαμβάνονται υπόψιν. Ένα δίκτυο περιβάλλοντος (context network) συμπεριφέρεται σαν υπηρεσία συγκεντρώνοντας αποτελέσματα, και μετατρέπεται σε ένα αποτελεσματικό προσαρμοζόμενο σύστημα. Η διανομή της υπηρεσίας διατηρείται στο δυναμικό περιβάλλον. Η αδράνεια μερικών οντοτήτων δεν επηρεάζει την υποδομή. Η επίγνωση περιβάλλοντος (Context-awareness) φέρει μία μεγάλη προοπτική στην παραγωγή καινοτόμων υπηρεσιών, στην κατανομή των πόρων και στις υπηρεσίες έρευνας και ανάπτυξης της ποιότητας και οι δυναμικές υπηρεσίες μπορούν να προσαρμοστούν αυτόματα ανάλογα με τα δεδομένα περιβάλλοντος αλλάζοντας την συμπεριφορά της υπηρεσίας.

6.2 Μάθηση Μεταγωγέα και Προώθηση (Switch Learning and Forwarding)

Ένας μεταγωγέας είναι μια συσκευή επιπέδου 2, το οποίο σημαίνει ότι επεξεργάζεται πλαίσια και μαθαίνει τις διευθύνσεις MAC. Έπειτα, αποθηκεύει τις πληροφορίες σε έναν εσωτερικό πίνακα. Κάθε φορά που ο μεταγωγέας λαμβάνει ένα καινούριο πλαίσιο, αποθηκεύει την διεύθυνση πηγής MAC του πλαισίου, όπως και την θύρα από όπου προήλθε, στον πίνακα του. Την επόμενη φορά που θα λάβει ένα πλαίσιο με την συγκεκριμένη διεύθυνση MAC σαν προορισμό, θα γνωρίζει μέσω ποιας θύρας θα προωθήσει το πλαίσιο.

Στην περίπτωση που δεν περιλαμβάνεται η διεύθυνση MAC στον εσωτερικό πίνακα, τότε ο μεταγωγέας θα εκπέμψει ή πλημμυρίσει το πλαίσιο με την άγνωστη διεύθυνση προορισμού σε όλες τις θύρες του, εκτός από αυτήν που προήλθε, πραγματοποιώντας τεχνική broadcast και όχι unicast. Η διαδικασία λέγεται πλημμύρα άγνωστης unicast (unknown unicast flooding).

Οι μεταγωγείς μαθαίνουν τις διευθύνσεις MAC δυναμικά, ενώ αποθηκεύονται στον πίνακα για μια συγκεκριμένη περίοδο χρόνου. Επίσης, οι στατικές διευθύνσεις MAC μπορούν να ρυθμιστούν στον πίνακα.

6.3 Δομοστοιχείο Μάθησης Μεταγωγέα (Learning Switch Module)

Το Learning Switch Module υλοποιεί συμπεριφορά παρόμοια με αυτή των τυπικών μεταγωγέων. Ανακαλύπτει και μαθαίνει νέες συσκευές βασίζοντας στις διευθύνσεις MAC αυτών.

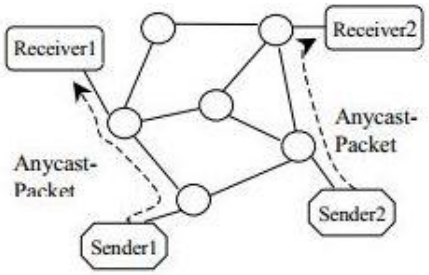
Όταν ανιχνεύεται μία νέα ροή, το Learning Switch Module αναγνωρίζει τους μεταγωγείς εισόδου και εξόδου, όπως και όλους τους μεταγωγείς που ανήκουν στην συντομότερη διαδρομή μεταξύ αποστολέα και δέκτη. Μόλις βρεθεί η διαδρομή, εγκαθίστανται κατάλληλοι κανόνες OpenFlow για την διαχείριση των νέων ροών σε όλους τους συμμετέχοντες μεταγωγείς.

Επιπλέον, η απόδοση του Learning Switch Module είναι υψηλή καθώς μετά την εγκατάσταση των κανόνων OpenFlow, η προώθηση πακέτων πραγματοποιείται μόνο στην διαδρομή προώθησης και δεν χρειάζεται πλέον η επικοινωνία με τον ελεγκτή. Βέβαια, εάν η διεύθυνση MAC προορισμού είναι άγνωστη τα πακέτα διαδίδονται σε όλο το δίκτυο.

6.4 Επέκταση Δομοστοιχείου Μάθησης Μεταγωγέα (Learning Switch Module Extension)

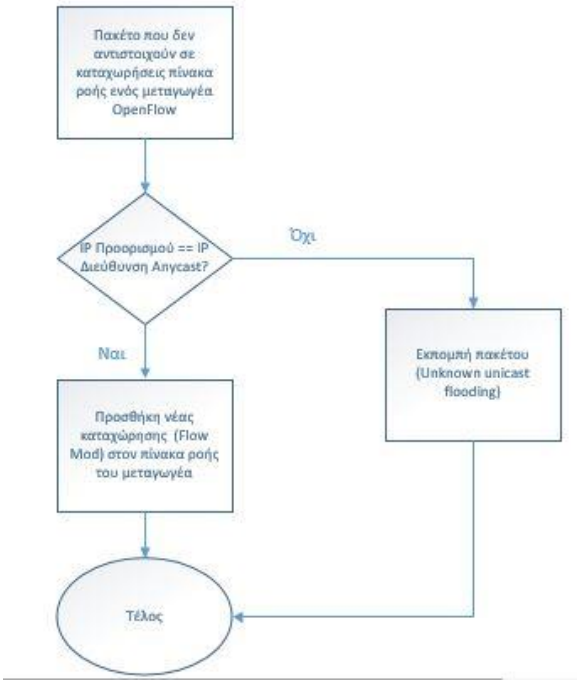
Στην παρούσα διπλωματική εργασία, χρησιμοποιούνται ο ελεγκτής OpenFlow Beacon και το Learning Switch Module. Επεκτείνοντας την κλάση Learning Switch δημιουργώντας μία κλάση Flow Manager, έχουμε δημιουργήσει έναν ελεγκτή ο οποίος υποστηρίζει πολλαπλούς μεταγωγείς, και ο κάθε μεταγωγέας λειτουργεί με βάση την ροή (flow-based switch). Όταν ένας μεταγωγέας λαμβάνει ένα πακέτο με έναν συγκεκριμένο προορισμό, αντί να μπαίνει στην διαδικασία μάθησης, δημιουργείται μια νέα καταχώρηση ροής από τον ελεγκτή ώστε να προωθηθεί το πακέτο υπό τις συνθήκες και τους κανόνες της καταχώρησης αυτής. Με έναν τέτοιο μεταγωγέα, τα υπόλοιπα πακέτα που αντιστοιχούν στην καταχώρηση ροής διαχειρίζονται στο επίπεδο δεδομένων, αποφεύγοντας την χρήσιμη αλλά χρονοβόρα επικοινωνία με τον ελεγκτή.

Όπως θα αναλυθεί παρακάτω, προσθέσαμε μια νέα καταχώρηση ροής στους μεταγωγείς, στην περίπτωση που ληφθεί ένα πακέτο με μία διεύθυνση προορισμού IP Anycast, η οποία έχει οριστεί στο αρχείο topologia.py του λογισμικού Mininet σε δύο από τα τερματικά που λειτουργούν στην παρούσα τοπολογία, να προωθούνται προς την επιθυμητή θύρα.



Εικόνα 6.1 Οι Receiver1, Receiver2 έχουν την ίδια διεύθυνση IP Anycast. Εάν ο Sender1 στείλει ένα πακέτο στην συγκεκριμένη διεύθυνση Anycast, θα παραδοθεί στον Receiver1, ενώ εάν ο Sender2 στείλει ένα πακέτο στην ίδια διεύθυνση, θα παραδοθεί στον Receiver2 [18]

Όπως φαίνεται στην εικόνα 6.2 στο διάγραμμα ροής, ένα πακέτο που δεν αντιστοιχεί σε καταχώρηση στον πίνακα ροής ενός μεταγωγέα OpenFlow, στέλνεται στον ελεγκτή. Εάν η IP προορισμού είναι η IP διεύθυνση Anycast που έχουμε ορίσει, τότε προστίθεται μία νέα καταχώρηση στον πίνακα ροής του μεταγωγέα με τους κανόνες που έχουμε ορίσει. Σε αντίθετη περίπτωση, λειτουργεί όπως ο μεταγωγέας μάθησης και εκπέμπει το πακέτο. Τα επόμενα πακέτα που θα αντιστοιχηθούν με την συγκεκριμένη καταχώρηση, θα δρομολογηθούν χωρίς να χρειαστεί αλληλεπίδραση μεταγωγέα OpenFlow και ελεγκτή.



Εικόνα 6.2 Διάγραμμα Ροής Κλάσης Flow Manager

Βέβαια, έχοντας επεκτείνει την κλάση Learning Switch, εύκολα μπορεί να αναρωτηθεί κανείς πώς τα πακέτα θα δρομολογούνται με βάση την δική μας καταχώρηση και

όχι του μεταγωγέα μάθησης. Αυτό διευκρινίζεται με δύο τρόπους. Πρώτον, με τον ορισμό της δικής μας καταχώρησης ως πρώτη σε προτεραιότητα σε περίπτωση που προωθείται πακέτο προς την IP Διεύθυνση Anycast. Δεύτερον, με τον έλεγχο επικάλυψης, για την αποφυγή δύο ροών προώθησης των ίδιων πακέτων.

6.5 Διαχείριση Πληροφοριών

Για την διευκόλυνση του χρήστη και την επίτευξης της κατανομής κίνησης κατά την διεξαγωγή του πειράματος, θεωρήθηκε απαραίτητη η συλλογή και η διανομή πληροφοριών, κάποιες από τις οποίες θα χρησιμοποιηθούν ως δεδομένα. Ο αλγόριθμος που χρησιμοποιείται λαμβάνει πληροφορίες σχετικά με την κατάσταση των δρομολογητών και την τοπολογία του δικτύου.

6.5.1 Τοπολογία Δικτύου

Για τον καλύτερο σχεδιασμό της κατανομής της κίνησης στο δίκτυο, λαμβάνεται υπόψιν η διασύνδεση του δικτύου και συγκεκριμένα η διασύνδεση των μεταγωγέων OpenFlow. Παρόλο που θα ήταν δυνατή η εισαγωγή στατικής πληροφορίας όσον αφορά την τοπολογία του δικτύου, σε αυτήν την περίπτωση αποφεύγεται. Ο σημαντικότερος λόγος είναι η υποστήριξη της δυναμικότητας και της καθολικότητας του δικτύου, και έτσι το λογισμικό να χρησιμοποιείται γενικευμένα, αντί να αποτελεί μια μεμονωμένη περίπτωση.

Σε αυτήν την ενότητα χρησιμοποιείται ο ελεγκτής Beacon με σκοπό την δημιουργία και αποστολή μηνυμάτων προς όλους τους μεταγωγείς της τοπολογίας, οι οποίοι απαντούν αναφέροντας τις θύρες τους και τις γειτονικές δικτυακές συσκευές με τις οποίες συνδέονται. Πιο συγκεκριμένα, μέσω του thread Topology, ανταλλάσσονται μηνύματα με τις δικτυακές συσκευές, ενώ μέσω της διεπαφής ITopology δημιουργείται ένας κατάλογος των συνδέσμων μεταξύ συνδεδεμένων μεταγωγέων OpenFlow χρησιμοποιώντας την συνάρτηση

Map<LinkTuple,Long> getLinks(). Η συνάρτηση αυτή καλεί την κλάση LinkTuple, η οποία με την σειρά της καλεί την κλάση SwitchPortTuple. Στο τέλος έχουμε μια πλήρη εικόνα των συνδέσεων μεταξύ των μεταγωγέων OpenFlow του δικτύου.

Η κλάση LinkTuple χρησιμοποιείται στο thread topology.web, το οποίο έχει τροποποιηθεί έτσι ώστε να τοποθετούνται τα αποτελέσματα σε ένα αρχείο δεδομένων, topologia.txt, η μορφή του οποίου αναλύεται στην συνέχεια. Το αρχείο αυτό αποθηκεύεται σε έναν συγκεκριμένο φάκελο της επιλογής του προγραμματιστή (στην περίπτωση μας στον φάκελο eclipse). Στο ακόλουθο σχήμα φαίνεται η διαδικασία αποθήκευσης των στοιχείων της τοπολογίας του δικτύου.

```
BufferedWriter writer = null;
try {
    writer = new BufferedWriter(new FileWriter("topologia.txt"));
    List<String> columnName = new ArrayList<String>();
    columnName = new ArrayList<String>();
    columnName.add("Src Id");
    columnName.add("Src Port");
    columnName.add("Dst Id");
    columnName.add("Dst Port");
    int n = columnName.size();
    for(int i = 0; i < n ; i++){
      writer.write(columnName.get( i ) );
      writer.write("                  ");
    }
    writer.newLine();
    for (LinkTuple lt : topology.getLinks().keySet()) {
        List<String> row = new ArrayList<String>();
        row.add(HexString.toHexString(lt.getSrc().getSw().getId()));
        row.add(lt.getSrc().getPort().toString());
        row.add(HexString.toHexString(lt.getDst().getSw().getId()));
        row.add(lt.getDst().getPort().toString());
        for(int i = 0; i < n ; i++){
            writer.write( row.get( i ) );
            writer.write("          ");
          }
        writer.newLine();
```

Εικόνα 6.3 Αποθήκευση στοιχείων τοπολογίας του δικτύου

Τα στοιχεία που συγκεντρώθηκαν τοποθετούνται σε τέσσερις στήλες πίνακα, ώστε το αρχείο να είναι ευανάγνωστο. Στο αρχείο θα απεικονίζονται τα πεδία:

Src Id. Στο πεδίο αυτό εγγράφεται η διεύθυνση MAC (datapath id) των μεταγωγέων από τους οποίους στάλθηκε μήνυμα τύπου OpenFlow στους γειτονικούς μεταγωγείς.

Src Port. Εγγράφεται η θύρα των μεταγωγέων από την οποία αποστέλλεται το μήνυμα τύπου OpenFlow στους γειτονικούς μεταγωγείς.

Dst Id. Εγγράφεται η διεύθυνση MAC (datapath id) των μεταγωγέων οι οποίοι λαμβάνουν το μήνυμα τύπου OpenFlow που στάλθηκε από τους γειτονικούς μεταγωγείς.

Dst Port. Εγγράφεται η θύρα των μεταγωγέων οι οποίοι λαμβάνουν το μήνυμα τύπου OpenFlow που στάλθηκε από τους γειτονικούς μεταγωγείς.

6.5.2 Στατιστικά Στοιχεία

Προκειμένου να βεβαιωθούμε ότι η δρομολόγηση με βάση την υπηρεσία Anycast λειτουργεί, η συλλογή των στατιστικών στοιχείων κάθε δικτυακής συσκευής αποτελεί σημαντικό βήμα. Μέσω του ελεγκτή Beacon δημιουργούνται συνεχώς αιτήματα τύπου OFStatisticsRequest για να αναζητηθούν οι καταστάσεις κίνησης των μεταγωγέων. Στην συνέχεια, ο ελεγκτής λαμβάνει από τους μεταγωγείς μηνύματα τύπου OFStatisticsReply που περιέχουν πληροφορίες που αφορούν τα μηνύματα που διαχειρίζονται οι μεταγωγείς στο δίκτυο. Στην εικόνα 6.4 φαίνεται ο κώδικας που εκτελείται στον Beacon, με βάση τον οποίο λαμβάνονται οι απαντήσεις από τους μεταγωγείς και οι πληροφορίες αποθηκεύονται σε αρχείο .txt. Επίσης, δημιουργείται άλλο ένα αρχείο v2.txt, από το οποίο διαγράφονται οι όμοιες καταχωρήσεις ροής (χωρίς ροή πακέτων) που μπορεί να δημιουργηθούν κατά την διάρκεια της αποστολής αιτήσεων και κλήσεων ping. Τέλος, τα μηνύματα που λαμβάνονται από κάθε μεταγωγέα αποθηκεύονται σε χωριστό αρχείο με όνομα την διεύθυνση MAC της δικτυακής συσκευής.

```
System.out.println("---->switchId="+switchId);
String sw_id =switchId.replace(":", "_");
System.out.println("---->switchId="+sw_id);
FileWriter writer = new FileWriter(sw_id+".txt");
BufferedWriter bufferWritter = new BufferedWriter(writer);
List<String> columnName = new ArrayList<String>();
columnName = new ArrayList<String>();
columnName.add("SwitchId");
columnName.add("In Port");
columnName.add("DL Source");
columnName.add("NW Source");
columnName.add("NW Destination");
columnName.add("DL Destination");
columnName.add("Out Port");
columnName.add("Byte Count");
columnName.add("Packet Count");
columnName.add("Time(s)");
int n = columnName.size();
for (int i =0; i < n; i++){
    bufferWritter.write(columnName.get(i));
    bufferWritter.write("                ");

}
bufferWritter.newLine();
for(int i=0; i<data.size(); i++){
    List<String> rows = new ArrayList<String>();
    rows.add(switchId.replace(":",""));
    rows.add(U16.f(data.get(i).getMatch().getInputPort())+" ");
```

```
    rows.add(HexString.toHexString(data.get(i).getMatch().getDataLayerSource())+" ");
    rows.add(IPv4.fromIPv4Address(data.get(i).getMatch().getNetworkSource())+" ");
    rows.add(IPv4.fromIPv4Address(data.get(i).getMatch().getNetworkDestination())+ " ");
    StringBuffer outPorts = new StringBuffer();
    for (OFAction action : data.get(i).getActions()) {
        if (action instanceof OFActionOutput) {
            OFActionOutput ao = (OFActionOutput)action;
            if (outPorts.length() > 0)
                outPorts.append(" ");
            outPorts.append(U16.f(ao.getPort()));
        }
    }
    rows.add(HexString.toHexString(data.get(i).getMatch().getDataLayerDestination())+" ");
    rows.add(outPorts.toString()+" ");
    rows.add(U64.f(data.get(i).getByteCount())+" ");
    rows.add(U64.f(data.get(i).getPacketCount())+" ");
    double duration = (double)U32.f(data.get(i).getDurationSeconds())+((double) data.get(i).getDurationNanoseconds()) / 1000000000d;
    rows.add(Double.toString(duration));
    for (int l = 0;  l < n; l++){
        bufferWritter.write(rows.get(l));
        bufferWritter.write("          ");

    }
    bufferWritter.newLine();
}
bufferWritter.close();

// STRIP DUPLICATES FROM FILES
```

```
    BufferedReader rd = new BufferedReader(new FileReader(sw_id+".txt"));
    Set<String> lines = new LinkedHashSet<String>(10000);
    String line;
    while ((line = rd.readLine()) != null){
        lines.add(line);
    }
    rd.close();
    BufferedWriter wr = new BufferedWriter(new FileWriter(sw_id+"v2.txt"));
    for (String unique : lines){
        wr.write(unique);
        wr.newLine();
    }

    wr.close();
```

Εικόνα 6.4 Αποθήκευση στατιστικών στοιχείων σε αρχείο .txt

Όπως φαίνεται, τα στατιστικά στοιχεία τοποθετούνται σε εννέα στήλες πίνακα, ώστε το αρχείο να είναι ευανάγνωστο. Θα απεικονίζονται τα πεδία:

SwitchId. Στο πεδίο αυτό εγγράφεται η διεύθυνση MAC του μεταγωγέα από τον οποίο λαμβάνουμε τα στατιστικά στοιχεία.

In Port. Εγγράφεται η θύρα του μεταγωγέα από την οποία λαμβάνει μηνύματα.

DL Source, NW Source. Εγγράφονται η διεύθυνση MAC και η διεύθυνση IP του τερματικού που στέλνει μηνύματα και πακέτα.

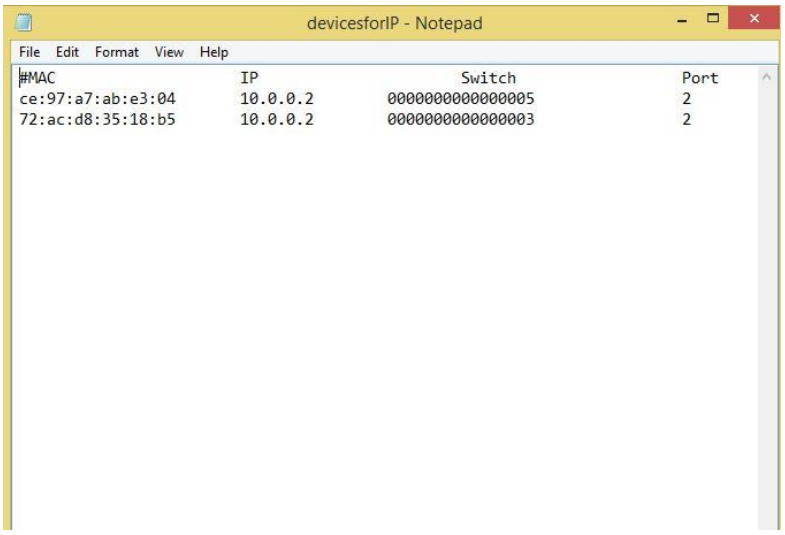
NW Destination, DL Destination. Εγγράφονται η διεύθυνση MAC και η διεύθυνση IP του τερματικού που θα λάβει τα μηνύματα και πακέτα.

Out Port. Η θύρα από την οποία θα στείλει τα μηνύματα και τα πακέτα

Byte Count, Packet Count, Time. Αναγράφονται το μέγεθος των byte, ο αριθμός των πακέτων που έχει σταλεί μέσα από την συγκεκριμένη ροή και η χρονική διάρκεια της καταχώρησης αυτής.

6.5.3 Δημιουργία Αρχείου και Καταχώρησης Ροής

Η καταγραφή των στατιστικών στοιχείων και της τοπολογίας του δικτύου διευκολύνει τον χρήστη να δημιουργήσει ένα αρχείο devicesforIP.txt, στο οποίο έχουμε καταγράψει στοιχεία για τα τερματικά που ανήκουν στην ίδια ομάδα Anycast. Με την χρήση του αρχείου, ο ελεγκτής δημιουργεί και στέλνει στους μεταγωγείς μία καταχώρηση ροής, όπου θα περιλαμβάνεται η θύρα από την οποία θα προωθήσει τα μηνύματα ο κάθε μεταγωγέας. Παρακάτω απεικονίζεται η μορφή του αρχείου devicesforIP.txt.



Εικόνα 6.5 devicesforIP.txt

Συγκεκριμένα, ο ελεγκτής Beacon αναλύει τα δεδομένα του αρχείου και στέλνει μηνύματα τύπου OpenFlow στους μεταγωγείς. Όταν δεν υπάρχει καταχώρηση στον πίνακα ροής του μεταγωγέα, ο ελεγκτής λαμβάνει ένα πακέτο OFPacketIn και το καταχωρεί σε μια

δομή τύπου OFMatch. Έπειτα αποθηκεύονται σε μεταβλητές στοιχεία όπως οι διευθύνσεις MAC πηγής, προορισμού, αλλά και η IP διεύθυνσης προορισμού. Εάν αυτή ισούται με την IP Anycast που έχουμε ορίσει στην τοπολογία του δικτύου στο λογισμικό Mininet, τότε ο ελεγκτής προχωρά στην ανά σειρά ανάγνωση του αρχείου devicesforIP.txt και αποθηκεύει σε έναν πίνακα τα δεδομένα. Στην συνέχεια ελέγχει εάν η datapathid του μεταγωγέα που έστειλε το πακέτο ισούται με μία από τις datapathid του αρχείου και ανάλογα αναθέτει την σωστή θύρα που έχουμε ορίσει στο αρχείο ώστε να προωθηθεί το πακέτο και να δημιουργηθεί η καταχώρηση στον πίνακα ροής του μεταγωγέα.

Η ανάθεση θύρας και η δημιουργία καταχώρησης ροής στον πίνακα ροής του μεταγωγέα πραγματοποιείται με την αποστολή αντίστοιχων μηνυμάτων από τον ελεγκτή. Είναι προφανές ότι τα μηνύματα αυτά έχουν μία συγκεκριμένη δομή για την αναγνώριση των πληροφοριών από τους μεταγωγείς. Στην συγκεκριμένη περίπτωση ο ελεγκτής στέλνει μηνύματα τύπου OFPT_FLOW_MOD για να τροποποιήσει την συμπεριφορά του μεταγωγέα, η δομή του οποίου φαίνεται στην παρακάτω εικόνα.

```
/* Flow setup and teardown (controller -> datapath). */
struct ofp_flow_mod {
    struct ofp_header header;
    uint64_t cookie;            /* Opaque controller-issued identifier. */
    uint64_t cookie_mask;       /* Mask used to restrict the cookie bits
                                   that must match when the command is
                                   OFPFC_MODIFY* or OFPFC_DELETE*. A value
                                   of 0 indicates no restriction. */

    /* Flow actions. */
    uint8_t table_id;           /* ID of the table to put the flow in.
                                   For OFPFC_DELETE_* commands, OFPTT_ALL
                                   can also be used to delete matching
                                   flows from all tables. */
    uint8_t command;            /* One of OFPFC_*. */
    uint16_t idle_timeout;      /* Idle time before discarding (seconds). */
    uint16_t hard_timeout;      /* Max time before discarding (seconds). */
    uint16_t priority;          /* Priority level of flow entry. */
    uint32_t buffer_id;         /* Buffered packet to apply to, or
                                   OFP_NO_BUFFER.
                                   Not meaningful for OFPFC_DELETE*. */
    uint32_t out_port;          /* For OFPFC_DELETE* commands, require
                                   matching entries to include this as an
                                   output port.  A value of OFPP_ANY
                                   indicates no restriction. */
    uint32_t out_group;         /* For OFPFC_DELETE* commands, require
                                   matching entries to include this as an
                                   output group.  A value of OFPG_ANY
                                   indicates no restriction. */
    uint16_t flags;             /* Bitmap of OFPFF_* flags. */
    uint16_t importance;        /* Eviction precedence (optional). */
    struct ofp_match match;     /* Fields to match. Variable size. */
    /* The variable size and padded match is always followed by instructions. */
    //struct ofp_instruction_header instructions[0];
                        /* Instruction set - 0 or more. The length
                           of the instruction set is inferred from
                           the length field in the header. */
};
OFP_ASSERT(sizeof(struct ofp_flow_mod) == 56);
```

Εικόνα 6.6 Δομή μηνύματος OFPT_FLOW_MOD [6]

Η δομή της εντολής για τροποποίηση ροής (προσθήκη, τροποποίηση, αφαίρεση) φαίνεται παρακάτω.

```
enum ofp_flow_mod_command {
    OFPFC_ADD           = 0, /* New flow. */
    OFPFC_MODIFY        = 1, /* Modify all matching flows. */
    OFPFC_MODIFY_STRICT = 2, /* Modify entry strictly matching wildcards and
                                priority. */
    OFPFC_DELETE        = 3, /* Delete all matching flows. */
    OFPFC_DELETE_STRICT = 4, /* Delete entry strictly matching wildcards and
                                priority. */
};
```

Εικόνα 6.7 Δομή FLOW_MOD_COMMAND [6]

Τέλος, η δομή του πεδίου flags απεικονίζεται παρακάτω. Στην παρούσα διπλωματική εργασία, σημειώνεται ότι χρησιμοποιήθηκαν συνολικά η εντολή προσθήκης και η σημαία ελέγχου επικαλυπτόμενων ροών.

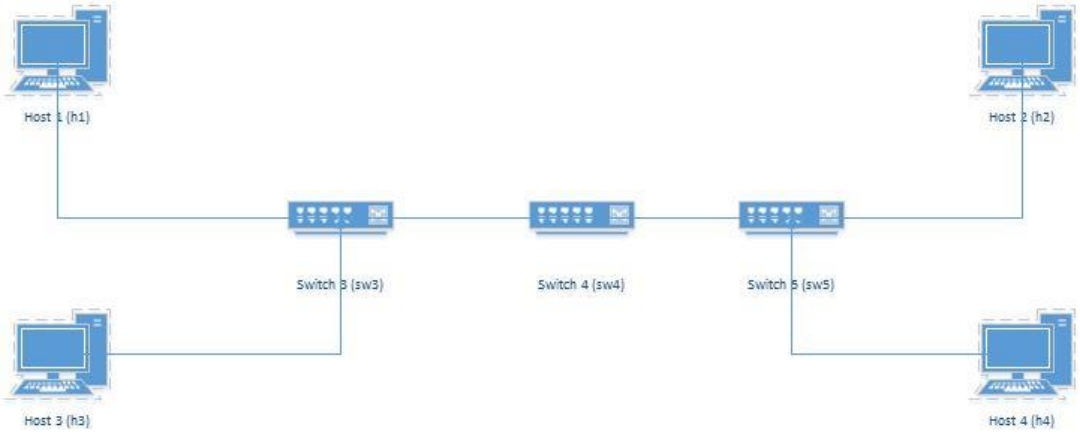
```
enum ofp_flow_mod_flags {
    OFPFF_SEND_FLOW_REM = 1 << 0,  /* Send flow removed message when flow
                                    * expires or is deleted. */
    OFPFF_CHECK_OVERLAP = 1 << 1,  /* Check for overlapping entries first. */
    OFPFF_RESET_COUNTS  = 1 << 2,  /* Reset flow packet and byte counts. */
    OFPFF_NO_PKT_COUNTS = 1 << 3,  /* Don't keep track of packet count. */
    OFPFF_NO_BYT_COUNTS = 1 << 4,  /* Don't keep track of byte count. */
};
```

Εικόνα 6.8 FLOW_MOD_FLAGS [6]

Κατά την διάρκεια της αποστολής των μηνυμάτων, οι μεταγωγείς τροποποιούν κατάλληλα τους πίνακες ροής και η κίνηση του δικτύου δρομολογείται ανάλογα με την επιθυμητή κατάσταση.

6.6 Βέλτιστη Context – Aware Δρομολόγηση

Γενικά, το Mininet επιτρέπει την δημιουργία προσαρμοσμένων τοπολογιών δικτύων ανάλογα με τις ανάγκες του χρήστη. Στην παρούσα διπλωματική εργασία επιλέχθηκε η εξής τοπολογία.



Εικόνα 6.9 Τοπολογία διπλωματικής εργασίας

Η τοπολογία μας αποτελείται από τέσσερα τερματικά και τρείς μεταγωγείς με τις αντίστοιχες συνδέσεις. Βέβαια, για την τοπολογία αυτή χρειάστηκαν λίγες γραμμές κώδικα Python στο Mininet. Παρακάτω παρουσιάζεται ο κώδικας για την τοπολογία του δικτύου.

```
"""Custom topology example

Four hosts, connected to two switches, connected to 1 switch. In total, 3 switches and 4 hosts:

   host-host --- switch --- switch --- switch --- host-host

Adding the 'topos' dict with a key/value pair to generate our newly defined
topology enables one to pass in '--topo=mytopo' from the command line.
"""

from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost1 = self.addHost( 'h1' , ip='10.0.0.1', mac='fa:d0:2f:4f:88:71')
        leftHost2 = self.addHost( 'h3' , ip='10.0.0.2',  mac='72:ac:d8:35:18:b5')
        rightHost1 = self.addHost( 'h2' , ip='10.0.0.2', mac='ce:97:a7:ab:e3:04' )
        rightHost2 = self.addHost( 'h4' , ip='10.0.0.4',  mac='82:0a:a0:64:f7:b1')
        leftSwitch = self.addSwitch( 's3' , dpid='0000000000000003')
        middleSwitch = self.addSwitch( 's4' ,dpid='0000000000000004')
        rightSwitch = self.addSwitch( 's5' ,dpid='0000000000000005')

        # Add links
        self.addLink( leftHost1, leftSwitch )
        self.addLink( leftHost2, leftSwitch )
        self.addLink( leftSwitch, middleSwitch )
        self.addLink( middleSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost1 )
        self.addLink( rightSwitch, rightHost2 )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Εικόνα 6.10 Τοπολογία σε Python

Σημειώνεται ότι για λόγους ευκολίας προσδιορισμού και εύρεσης των τερματικών για την παρούσα διπλωματική εργασία, οι διευθύνσεις ip και mac των τερματικών είναι

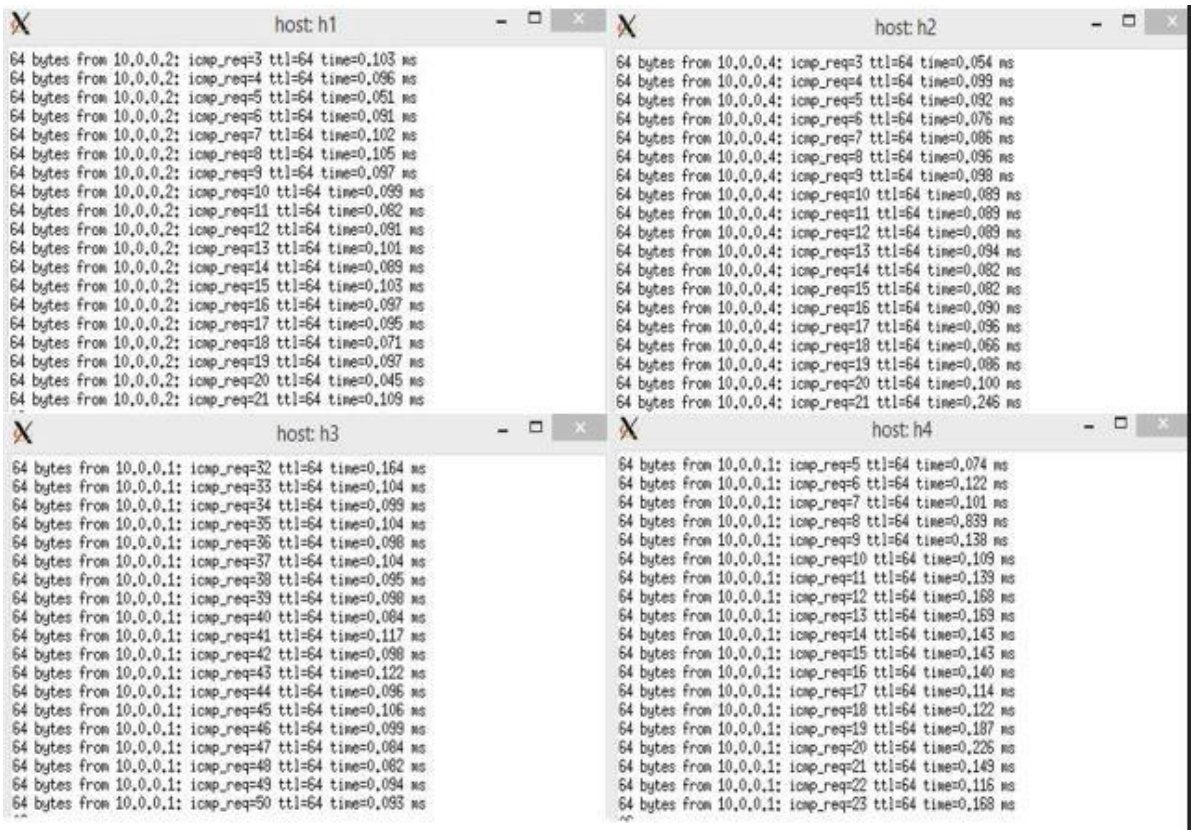
προκαθορισμένες, καθώς και τα datapathids(dpid) των μεταγωγέων είναι επίσης προκαθορισμένα. Παρατηρείται επίσης ότι τα δύο από τα τερματικά έχουν διαφορετικές ονομασίες, όμως μοιράζονται την ίδια ip, για τους λόγους που εξηγούνται σε άλλο κεφάλαιο.

Για την εκτέλεση της εξομοίωσης αυτού του δικτύου χρησιμοποιούμε την εξής εντολή:

```
mininet@mininet-vm:~/mininet/custom$ sudo mn --custom topologia.py --topo mytopo  --controller=remote,ip=192.168.56.1 -x
```

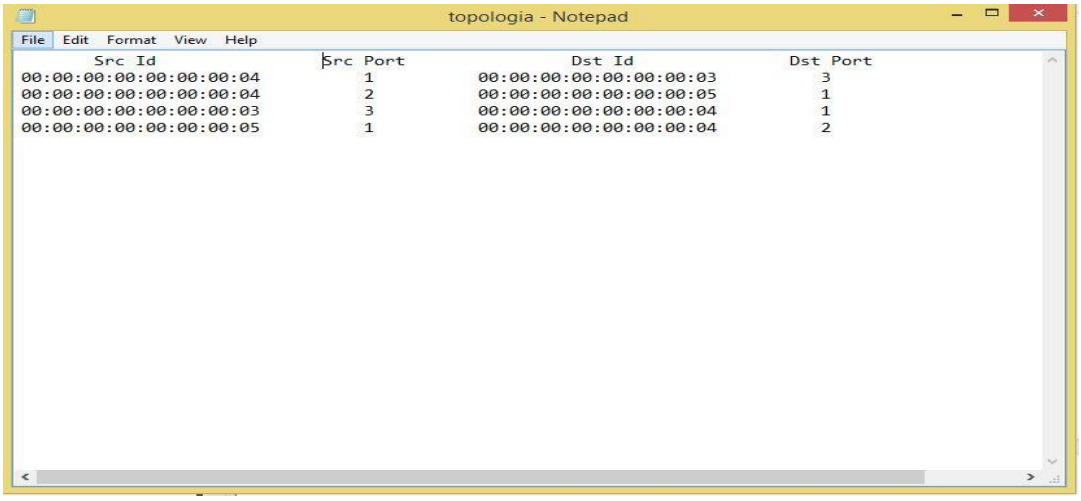
Με όρο sudo mn, ο χρήστης αποκτά δικαιώματα διαχειριστή και δίνει εντολή στο Mininet να εκτελέσει το αρχείο topologia.py που υπάρχει στον φάκελο custom, να δημιουργήσει την τοπολογία mytopo έτσι όπως έχει οριστεί στον κώδικα. Με τους όρους –controller=remote, ip=192.168.56.1, προσδιορίζεται ότι ο εξωτερικός ελεγκτής OpenFlow της τοπολογίας (στην δική μας περίπτωση, ο Beacon), βρίσκεται στην διεύθυνση με ip την 192.168.56.1. Τέλος, με τον όρο –x, ενεργοποιείται το X Window System (xterm) που επιτρέπει την δημιουργία πολλαπλών παραθύρων εκτέλεσης εντολών (command prompts), ένα για κάθε συσκευή, και διευκολύνει την αλληλεπίδραση μεταξύ των τερματικών.

Στις παρακάτω εικόνες φαίνονται παραδείγματα δημιουργίας κίνησης για κάθε τερματικό με την επαναλαμβανόμενη κλήση αιτήσεων τύπου ping. Συγκεκριμένα, βλέπουμε την δημιουργία κίνησης από τον host h1 προς τον host h3, ο host h2 προς τον host h4, ο host h3 προς τον host h1 και ο host h4 προς τον host h1.



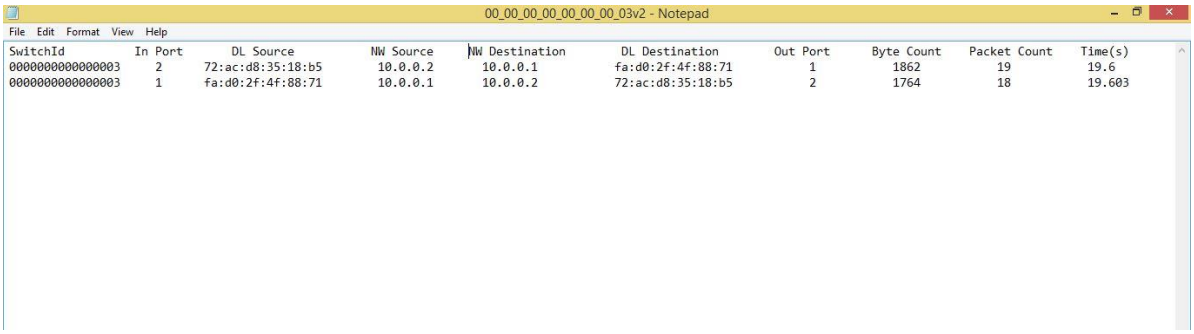
Εικόνα 6.11 Δημιουργία κίνησης μεταξύ των hosts στο λογισμικό Mininet

Στην εικόνα 6.12 απεικονίζεται το περιεχόμενο του αρχείου topologia.txt που αναφέρεται στην τοπολογία που αναπτύχθηκε στο Mininet.



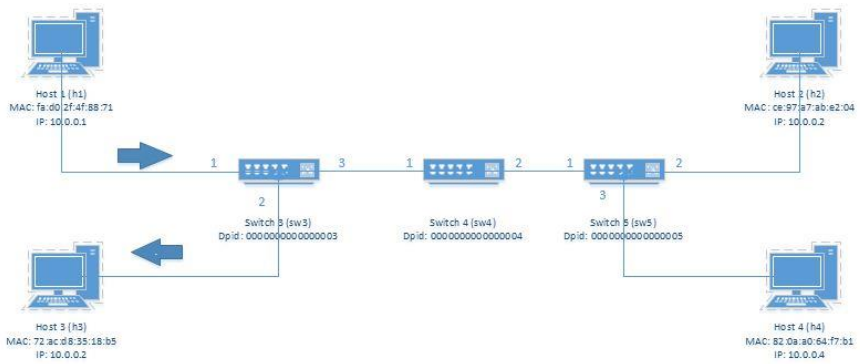
Εικόνα 6.12 topologia.txt

Παρακάτω απεικονίζεται το περιεχόμενο του αρχείου 00_00_00_00_00_00_00_03v2.txt που αναφέρεται στα στατιστικά στοιχεία (flows) που συλλέγονται σε μία χρονική στιγμή από τον μεταγωγέα με την αντίστοιχη διεύθυνση.

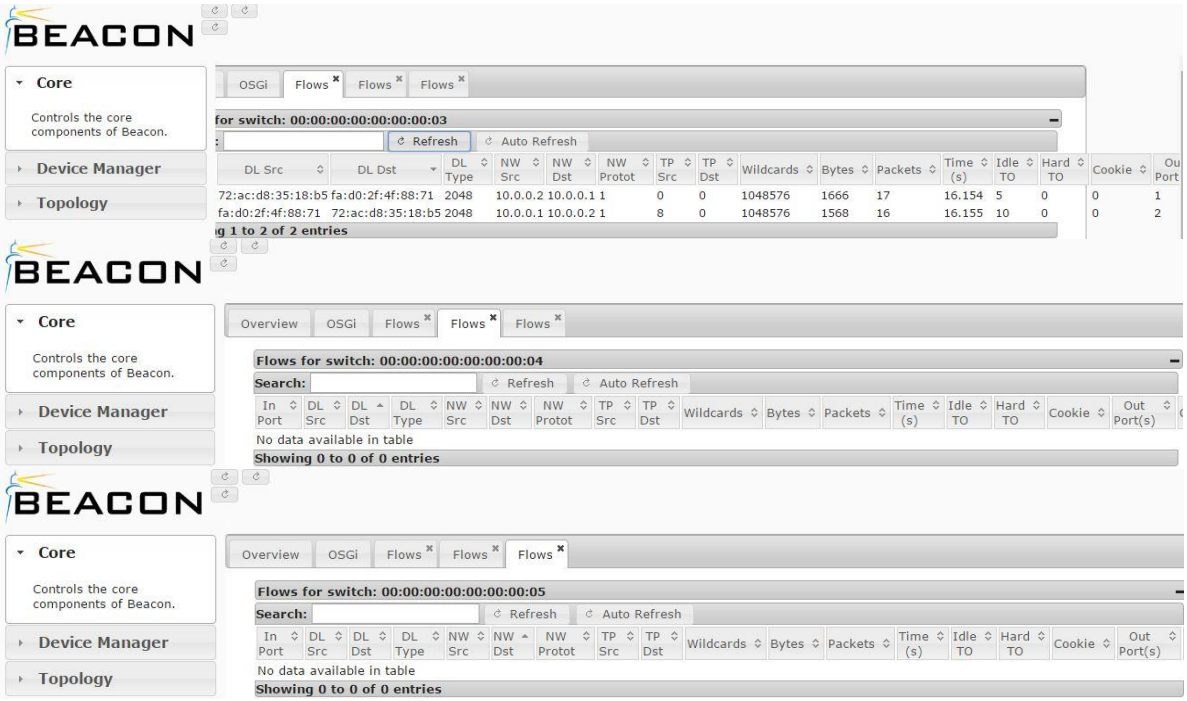


Εικόνα 6.13 00_00_00_00_00_00_00_03v2.txt

Εφαρμόζοντας λοιπόν την παραπάνω τοπολογία και έχοντας ως δεδομένα τα στοιχεία των αρχείων που έχουν αναφερθεί, προχωράμε στην πειραματική επιβεβαίωση της εργασίας. Ένα παράδειγμα της αποδοτικής δρομολόγησης που προωθήσαμε με χρήση την υπηρεσία Anycast και το Learning Switch Module (το οποίο υπενθυμίζεται ότι λειτουργεί βρίσκοντας την συντομότερη διαδρομή) φαίνεται από τις εικόνες παρακάτω. Πιο συγκεκριμένα, εάν εφαρμοστεί η δρομολόγηση που προτείνεται, και σταλεί αίτημα ping από τον Host h1 προς τον Host h2, ο μεταγωγέας sw3 προωθεί το πακέτο στον Host h3. Αυτό συμβαίνει διότι η διεύθυνση IP προορισμού (10.0.0.2) ισούται με την διεύθυνση IP Anycast που έχουμε θέσει και έτσι ο μεταγωγέας προωθεί το πακέτο μέσω της θύρας 2, προς τον Host h3.

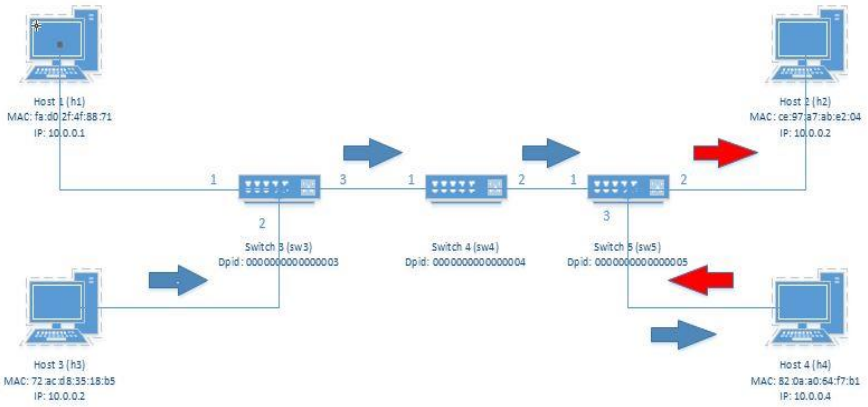


Εικόνα 6.3Κατεύθυνση πακέτων κατά την κλήση εντολής : h1 ping h2

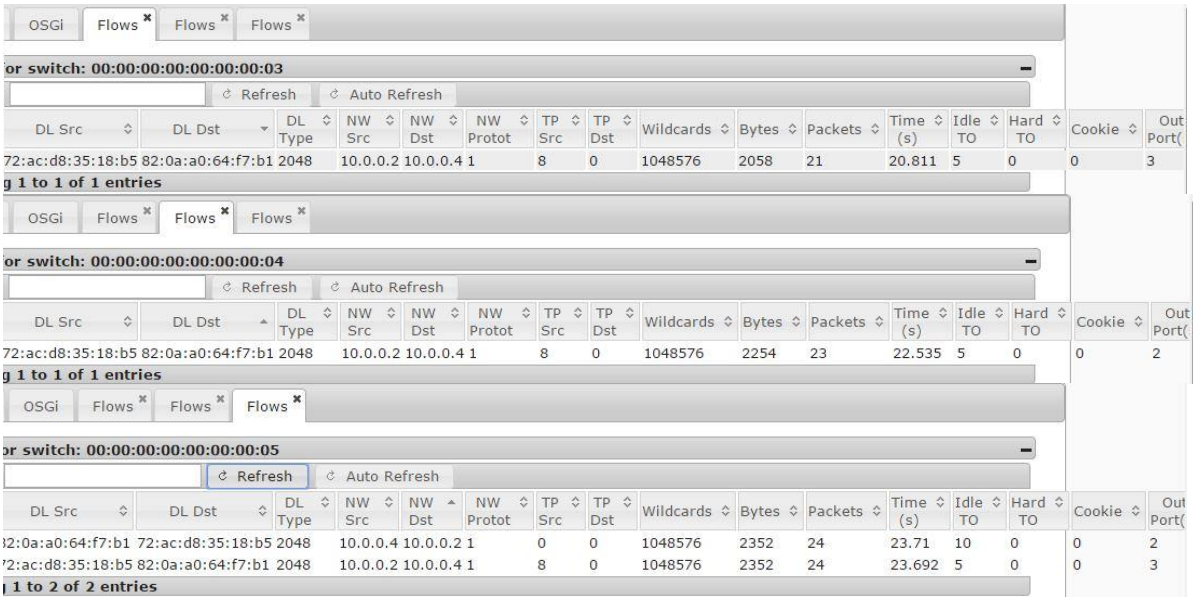


Εικόνα 6.4Κατανομή της κίνησης κατά την αποστολή πακέτων κατά την κλήση της εντολής : h1 ping h2

Ένα ακόμη παράδειγμα της αποδοτικής δρομολόγησης που προωθήσαμε φαίνεται στις εικόνες παρακάτω. Πιο συγκεκριμένα, εάν σταλεί αίτημα ping από τον Host h3 προς τον Host h4, οι μεταγωγείς sw3, sw4, sw5 προωθούν το πακέτο στον Host h4. Αντίθετα, ο Host h3 δεν λαμβάνει απάντηση και το πακέτο χάνεται. Αυτό συμβαίνει διότι η διεύθυνση IP προορισμού (10.0.0.2) ισούται με την διεύθυνση IP Anycast που έχουμε θέσει και έτσι ο μεταγωγέας sw5 προωθεί το πακέτο απάντησης από τον Host h4 μέσω της θύρας 2, προς τον Host h2.



Εικόνα 6.16 Κατεύθυνση πακέτων κατά την κλήση της εντολής : h3 ping h4



Εικόνα 6.17 Κατανομή της κίνησης κατά την αποστολή πακέτων κατά την κλήση της εντολής: h3 ping h4

6.7 Ανακεφαλαίωση και Συμπεράσματα

Η διευθυνσιοδότηση μέσω της μεθόδου Anycast είναι μία λύση για την παροχή εντοπισμού στις υπηρεσίες και τους διακομιστές. Είναι μια χρήσιμη εναλλακτική τεχνική για την γνωστοποίηση ορισμένων υπηρεσιών στο διαδίκτυο. Για τις συγκεκριμένες υπηρεσίες η επεκτασιμότητα, ο πλεονασμός και τα ζητήματα ασφαλείας έχουν βελτιωθεί σημαντικά και αυτό αποτελεί σημαντικό γεγονός διότι πλέον το διαδίκτυο είναι μέρος της καθημερινότητας των ανθρώπων.

Στα πλαίσια της παρούσας διπλωματικής εργασίας, αναπτύχθηκε ένας μηχανισμός μέσω του ελεγκτή Beacon, ο οποίος επικοινωνεί με τους μεταγωγείς και ανταλλάσσουν μηνύματα. Συλλέγονται πληροφορίες σχετικά με την τοπολογία του δικτύου και τα μηνύματα που διαχειρίζονται οι μεταγωγείς του δικτύου. Στην συνέχεια, εισέρχονται πληροφορίες από ένα εξωτερικό αρχείο, με σκοπό την εύρεση της συντομότερης δρομολόγησης αναζητώντας την διεύθυνση IP Anycast. Κατόπιν, ο ελεγκτής Beacon ανταλλάσσει μηνύματα με τους μεταγωγείς, προσθέτοντας μια καταχώρηση στους πίνακες ροής των μεταγωγέων.

Γενικά, μέσω της χρήσης του ελεγκτή Beacon και του πρωτοκόλλου OpenFlow, μπορούμε να συμπεράνουμε την σημαντικότητα της αρχιτεκτονικής SDN, σε σχέση με άλλες αρχιτεκτονικές δικτύου. Διαχωρίζοντας τα επίπεδα ελέγχου και δεδομένων, οι διαχειριστές

ενός δικτύου SDN έχουν την δυνατότητα τροποποίησης και ελέγχου των υπηρεσιών που λειτουργούν στο δίκτυο, μέσω του κεντρικού ή κεντρικών ελεγκτών, χωρίς την χειροκίνητη παραμετροποίηση των δικτυακών συσκευών.

Το SDN είναι μία συνεχώς αναπτυσσόμενη αρχιτεκτονική σχεδιασμού, διατήρησης και λειτουργίας των πληροφοριών ενός δικτύου υπολογιστών. Θεωρείται από τις σημαντικότερες τάσεις για το διαδίκτυο του μέλλοντος και τα περιφερειακά δίκτυα. Έχουν αναπτυχθεί αρκετά πρωτόκολλα που υποστηρίζουν (όπως το OpenFlow) την συγκεκριμένη αρχιτεκτονική, και κεντρικοί ελεγκτές σε ποικίλες γλώσσες προγραμματισμού, όπως οι Beacon, NOX, Floodlight, συμβάλλοντας στην δημιουργία λειτουργικών και αξιόπιστων δικτύων.

Βιβλιογραφία

[1] An introduction to SDN http://www.merunetworks.com/collateral/solution-briefs/an-introduction-to-sdn-sb.pdf

[2] What's software - defined Networking? https://www.sdxcentral.com/resources/sdn/what-the-definition-of-software-defined-networking-sdn/

[3] ONF White Paper, "Software-Defined Networking: The New Norm for Networks", April 2012

[4] Open Networking Foundation (ONF), "SDN Architecture Overview", Version 1.0, December 2013

[5] Fei Hu, "Network Innovation through Openflow and SDN Principles and Design", CRC Press, 2014

[6] OpenFlow Switch Specification, Version 1.4.0 (Wire Protocol 0x05), October 2013

[7] David Erickson, "The Beacon OpenFlow Controller", Stanford University, August 2010

[8] Component- Based Software Defined Networking Framework: Build SDN Agilely, http://osrg.github.io/ryu/

[9] Spring Framework http://en.wikipedia.org/wiki/Spring_Framework

[10] Introduction to Mininet https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[11] Ruud Louwersheimer, "Implementing Anycast in IPv4 Networks", INS, June 2004

[12] C. Partridge, T. Mendez, W. Milliken, "RFC 1546 - Host Anycasting Service", November 1993

[13] Hitesh Ballani, Paul Francis, "Towards a Global IP Anycasting Service", 2005

[14] Border Gateway Protocol http://el.wikipedia.org/wiki/Border_Gateway_Protocol

[15] Anycast http://en.wikipedia.org/wiki/Anycast

[16] Xiaoming Hu, Yun Ding, Nearchos Paspallis, George A. Papadopoulos, Pyrros Bratskas, Paolo Barone, Alessandro Mamelli, Yves Vanrompay, Yolande Berbers, "A Peer-to-Peer Based Infrastructure for Context Distribution in Mobile and Ubiquitous Environments", 2007

[17] Switch Learning and Forwarding https://telconotes.wordpress.com/2013/03/09/how-a-switch-works/

[18] Dina Katabi, "The Use of IP-Anycast for Building Efficient Multicast Trees", MIT

Παράρτημα

Παρακάτω παρουσιάζονται οι κλάσεις του ελεγκτή Beacon που υπέστησαν επεξεργασία και χρησιμοποιήθηκαν κατά την υλοποίηση του κώδικα.

- CoreWebManageable.java

```
package net.beaconcontroller.core.web;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;
import net.beaconcontroller.core.IBeaconProvider;
import net.beaconcontroller.core.IOFMessageListener;
import net.beaconcontroller.core.IOFSwitch;
import net.beaconcontroller.packet.IPv4;
import net.beaconcontroller.util.BundleAction;
import net.beaconcontroller.web.IWebManageable;
import net.beaconcontroller.web.view.BeaconJsonView;
import net.beaconcontroller.web.view.BeaconViewResolver;
import net.beaconcontroller.web.view.Tab;
import net.beaconcontroller.web.view.json.DataTableJsonView;
import net.beaconcontroller.web.view.json.OFFlowStatisticsReplyDataTableFormatCallback;
import net.beaconcontroller.web.view.layout.Layout;
import net.beaconcontroller.web.view.layout.OneColumnLayout;
import net.beaconcontroller.web.view.layout.TwoColumnLayout;
import net.beaconcontroller.web.view.section.JspSection;
import net.beaconcontroller.web.view.section.StringSection;
import net.beaconcontroller.web.view.section.TableSection;
import org.openflow.protocol.OFFlowMod;
import org.openflow.protocol.OFMatch;
import org.openflow.protocol.OFPort;
import org.openflow.protocol.OFStatisticsRequest;
import org.openflow.protocol.OFType;
import org.openflow.protocol.action.OFAction;
import org.openflow.protocol.action.OFActionOutput;
import org.openflow.protocol.statistics.OFFlowStatisticsReply;
import org.openflow.protocol.statistics.OFFlowStatisticsRequest;
import org.openflow.protocol.statistics.OFPortStatisticsRequest;
import org.openflow.protocol.statistics.OFStatistics;
import org.openflow.protocol.statistics.OFStatisticsType;
import org.openflow.util.HexString;
import org.openflow.util.U16;
import org.openflow.util.U32;
import org.openflow.util.U64;
import org.osgi.framework.Bundle;
import org.osgi.framework.BundleContext;
import org.osgi.framework.BundleException;
import org.osgi.service.packageadmin.PackageAdmin;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.osgi.context.BundleContextAware;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
```

```
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.View;

@Controller
@RequestMapping("/core")
public class CoreWebManageable implements BundleContextAware, IWebManageable {
    protected interface OFSRCallback {
        OFStatisticsRequest getRequest();    }
    protected static Logger log = LoggerFactory.getLogger(CoreWebManageable.class);
    protected IBeaconProvider beaconProvider;
    protected BundleContext bundleContext;
    protected PackageAdmin packageAdmin;
    protected List<Tab> tabs;
    public CoreWebManageable() {
        tabs = new ArrayList<Tab>();
        tabs.add(new Tab("Overview", "/wm/core/overview.do"));
        tabs.add(new Tab("OSGi", "/wm/core/osgi.do"));    }
    @Autowired
    @Override
    public void setBundleContext(BundleContext bundleContext) {
        this.bundleContext = bundleContext;    }
    @Autowired
    public void setBeaconProvider(IBeaconProvider beaconProvider) {
        this.beaconProvider = beaconProvider;    }
    @Override
    public String getName() {
        return "Core";    }
    @Override
    public String getDescription() {
        return "Controls the core components of Beacon.";    }
    @Override
    public List<Tab> getTabs() {
        return tabs;    }
    @RequestMapping("/overview")
    public String overview(Map<String, Object> model) {
        Layout layout = new TwoColumnLayout();
        model.put("layout", layout);
        // Description
        layout.addSection(
                new StringSection("Welcome",
                        "Thanks for using Beacon! Beacon is listening for connections at " +
                (beaconProvider.getListeningIPAddress().isAnyLocalAddress() ? ""
                        : beaconProvider.getListeningIPAddress().getHostAddress()) +
                " port " + beaconProvider.getListeningPort() + "."),
                TwoColumnLayout.COLUMN1);
        // Switch List Table
        model.put("title", "Switches");
        model.put("switches", beaconProvider.getSwitches().values());
        layout.addSection(new JspSection("switches.jsp", model), TwoColumnLayout.COLUMN2);
        // Listener List Table
        List<String> columnNames = new ArrayList<String>();
        List<List<String>> cells = new ArrayList<List<String>>();
        columnNames = new ArrayList<String>();
        columnNames.add("OpenFlow Packet Type");
        columnNames.add("Listeners");
        cells = new ArrayList<List<String>>();
        for (Entry<OFType, List<IOFMessageListener>> entry :
beaconProvider.getListeners().entrySet()) {
            List<String> row = new ArrayList<String>();
            row.add(entry.getKey().toString());
            StringBuffer sb = new StringBuffer();
            for (IOFMessageListener listener : entry.getValue()) {
                sb.append(listener.getName() + " ");
            }
            row.add(sb.toString());
            cells.add(row);        }
        layout.addSection(new TableSection("OpenFlow Packet Listeners", columnNames, cells,
"table-listeners"), TwoColumnLayout.COLUMN1);
        return BeaconViewResolver.SIMPLE_VIEW;    }
    @RequestMapping("/osgi")
    public String osgi(Map<String, Object> model) {
        Layout layout = new OneColumnLayout();
        model.put("layout", layout);
        // Bundle Form
        model.put("title", "Add Bundle");
        layout.addSection(new JspSection("addBundle.jsp", new HashMap<String,
Object>(model)), TwoColumnLayout.COLUMN1);
```

```
        // Bundle List Table
        model.put("bundles", Arrays.asList(this.bundleContext.getBundles()));
        model.put("title", "OSGi Bundles");
        layout.addSection(new JspSection("bundles.jsp", model), TwoColumnLayout.COLUMN1);
        return BeaconViewResolver.SIMPLE_VIEW;    }
    @RequestMapping("/bundle/{bundleId}/{action}")
    @ResponseBody
    public String osgiAction(@PathVariable Long bundleId, @PathVariable String action) {
        final Bundle bundle = this.bundleContext.getBundle(bundleId);
        if (action != null) {
            try {
                if (BundleAction.START.toString().equals(action)) {
                    bundle.start();
                } else if (BundleAction.STOP.toString().equals(action)) {
                    bundle.stop();
                } else if (BundleAction.UNINSTALL.toString().equals(action)) {
                    bundle.uninstall();
                } else if (BundleAction.REFRESH.toString().equals(action)) {
                    packageAdmin.refreshPackages(new Bundle[] {bundle});                }
            } catch (BundleException e) {
                log.error("Failure performing action " + action + " on bundle " +
bundle.getSymbolicName(), e);
            }        }
        return "";    }
    @RequestMapping(value = "/bundle/add", method = RequestMethod.POST)
    public View osgiBundleAdd(@RequestParam("file") MultipartFile file, Map<String, Object>
model) throws Exception {
        BeaconJsonView view = new BeaconJsonView();
        File tempFile = null;
        Bundle newBundle = null;
        try {
            tempFile = File.createTempFile("beacon", ".jar");
            file.transferTo(tempFile);
            tempFile.deleteOnExit();
            newBundle = bundleContext.installBundle("file:"+tempFile.getCanonicalPath());
            model.put(BeaconJsonView.ROOT_OBJECT_KEY,
                    "Successfully installed: " + newBundle.getSymbolicName()
                            + "_" + newBundle.getVersion());
        } catch (IOException e) {
            log.error("Failure to create temporary file", e);
            model.put(BeaconJsonView.ROOT_OBJECT_KEY, "Failed to install bundle.");
        } catch (BundleException e) {
            log.error("Failure installing bundle", e);
            model.put(BeaconJsonView.ROOT_OBJECT_KEY, "Failed to install bundle.");
        }
        view.setContentType("text/javascript");
        return view;    }
    @RequestMapping("/refreshWeb")
    @ResponseBody
    public String refreshWebBundle() {
        for (Bundle bundle : this.bundleContext.getBundles()) {
            if (bundle.getSymbolicName().equalsIgnoreCase("net.beaconcontroller.web")) {
                packageAdmin.refreshPackages(new Bundle[] {bundle});
                try {
                    Thread.sleep(1000);
                    while (bundle.getState() != Bundle.ACTIVE) {
                        Thread.sleep(100);                    }
                } catch (InterruptedException e) {
                    log.error("Interupted waiting for refresh", e);                }
}        }
        return "";    }

    protected List<OFStatistics> getSwitchStats(OFSRCallback f, String switchId, String
statsType) {
        IOFSwitch sw = beaconProvider.getSwitches().get(HexString.toLong(switchId));
        Future<List<OFStatistics>> future;
        List<OFStatistics> values = null;
        if (sw != null) {
            try {
                future = sw.getStatistics(f.getRequest());
                values = future.get(10, TimeUnit.SECONDS);
            } catch (Exception e) {
                log.error("Failure retrieving " + statsType, e);            }        }
        return values;    }

    protected class makeFlowStatsRequest implements OFSRCallback {
        public OFStatisticsRequest getRequest() {
```

```
            OFStatisticsRequest req = new OFStatisticsRequest();
            OFFlowStatisticsRequest fsr = new OFFlowStatisticsRequest();
            OFMatch match = new OFMatch();
            match.setWildcards(0xffffffff);
            fsr.setMatch(match);
            fsr.setOutPort(OFPort.OFPP_NONE.getValue());
            fsr.setTableId((byte) 0xff);
            req.setStatisticType(OFStatisticsType.FLOW);
            req.setStatistics(fsr);
            req.setLengthU(req.getLengthU() + fsr.getLength());
            return req;
        };    }
    public String addStatsSection(String statsType, @PathVariable String switchId,
Map<String,Object> model, List<OFStatistics> stats) {
        OneColumnLayout layout = new OneColumnLayout();
        model.put("title", statsType + " for switch: " + switchId);
        model.put("layout", layout);
        model.put(statsType, stats);
        layout.addSection(new JspSection(statsType + ".jsp", model), null);
        return BeaconViewResolver.SIMPLE_VIEW;    }
    protected List<OFStatistics> getSwitchFlows(String switchId) {
        return getSwitchStats(new makeFlowStatsRequest(), switchId, "flows");
    }
    // FLOWS added to file .txt
    @RequestMapping("/switch/{switchId}/flows/json")
    public View getSwitchFlowsJson(@PathVariable String switchId, Map<String,Object> model)
{
        BeaconJsonView view = new BeaconJsonView();
        model.put(BeaconJsonView.ROOT_OBJECT_KEY, getSwitchFlows(switchId));
        return view;    }

    @SuppressWarnings("unchecked")
    @RequestMapping("/switch/{switchId}/flows/dataTable")
    public View getSwitchFlowsDataTable(@PathVariable String switchId, Map<String,Object>
model) throws IOException  {
        List<OFFlowStatisticsReply> data = new ArrayList<OFFlowStatisticsReply>();
        List<OFStatistics> stats = getSwitchFlows(switchId);
        if (stats != null) {
            // Ugly cast.. sigh
            data.addAll((Collection<? extends OFFlowStatisticsReply>) stats);        }
        DataTableJsonView<OFFlowStatisticsReply> view = new
DataTableJsonView<OFFlowStatisticsReply>(
                data, new OFFlowStatisticsReplyDataTableFormatCallback(switchId));
        System.out.println("---->switchId="+switchId);
        String sw_id =switchId.replace(":", "_");
        System.out.println("---->switchId="+sw_id);
        FileWriter writer = new FileWriter(sw_id+".txt");
        BufferedWriter bufferWritter = new BufferedWriter(writer);
        List<String> columnName = new ArrayList<String>();
        columnName = new ArrayList<String>();
        columnName.add("SwitchId");
        columnName.add("In Port");
        columnName.add("DL Source");
        columnName.add("NW Source");
        columnName.add("NW Destination");
        columnName.add("DL Destination");
        columnName.add("Out Port");
        columnName.add("Byte Count");
        columnName.add("Packet Count");
        columnName.add("Time(s)");
        int n = columnName.size();
        for (int i =0; i < n; i++){
            bufferWritter.write(columnName.get(i));
            bufferWritter.write("                ");
        }
        bufferWritter.newLine();
        for(int i=0; i<data.size(); i++){
            List<String> rows = new ArrayList<String>();
            rows.add(switchId.replace(":",""));
            rows.add(U16.f(data.get(i).getMatch().getInputPort())+" ");
            rows.add(HexString.toHexString(data.get(i).getMatch().getDataLayerSource())+"
");
            rows.add(IPv4.fromIPv4Address(data.get(i).getMatch().getNetworkSource())+" ");
            rows.add(IPv4.fromIPv4Address(data.get(i).getMatch().getNetworkDestination())+ "
");
            StringBuffer outPorts = new StringBuffer();
            for (OFAction action : data.get(i).getActions()) {
```

```
                if (action instanceof OFActionOutput) {
                    OFActionOutput ao = (OFActionOutput)action;
                    if (outPorts.length() > 0)
                        outPorts.append(" ");
                    outPorts.append(U16.f(ao.getPort()));                }            }
rows.add(HexString.toHexString(data.get(i).getMatch().getDataLayerDestination())+" ");
            rows.add(outPorts.toString()+" ");
            rows.add(U64.f(data.get(i).getByteCount())+" ");
            rows.add(U64.f(data.get(i).getPacketCount())+" ");
            double duration = (double)U32.f(data.get(i).getDurationSeconds())+((double)
data.get(i).getDurationNanoseconds()) / 1000000000d;
            rows.add(Double.toString(duration));

            for (int l = 0;  l < n; l++){
                bufferWritter.write(rows.get(l));
                bufferWritter.write("          ");
            }
            bufferWritter.newLine();        }
        bufferWritter.close();
        // STRIP DUPLICATES FROM FILES
            BufferedReader rd = new BufferedReader(new FileReader(sw_id+".txt"));
            Set<String> lines = new LinkedHashSet<String>(10000);
            String line;
            while ((line = rd.readLine()) != null){
                lines.add(line);
            }
            rd.close();
            BufferedWriter wr = new BufferedWriter(new FileWriter(sw_id+"v2.txt"));
            for (String unique : lines){
                wr.write(unique);
                wr.newLine();
            }
            wr.close();
        return view;    }
    @RequestMapping("/switch/{switchId}/flows")
    public String getSwitchFlows(@PathVariable String switchId, Map<String,Object> model) {
        OneColumnLayout layout = new OneColumnLayout();
        model.put("title", "Flows for switch: " + switchId);
        model.put("layout", layout);
        model.put("switchId", switchId);
        model.put("switchIdEsc", switchId.replaceAll(":", ""));
        layout.addSection(new JspSection("flows.jsp", model), null);
        return BeaconViewResolver.SIMPLE_VIEW;    }
    @RequestMapping("/switch/{switchId}/flow/{flowId}/del")
    public String delFlow(@PathVariable String switchId,
            @PathVariable String flowId, Map<String, Object> model) {
        OFMatch match = new OFMatch();
        match.fromString(flowId);
        OFFlowMod mod = new OFFlowMod();
        mod.setCommand(OFFlowMod.OFPFC_DELETE)
            .setMatch(match)
            .setOutPort(OFPort.OFPP_NONE);
        IOFSwitch sw = beaconProvider.getSwitches().get(
                HexString.toLong(switchId));
        try {
            sw.getOutputStream().write(mod);
        } catch (IOException e) {
            log.error("Failure writing delete flowmod", e);        }
        return BeaconViewResolver.SIMPLE_VIEW;    }
    protected List<OFStatistics> getSwitchTables(String switchId) {
        return getSwitchStats(new OFSRCallback() {
            @Override
            public OFStatisticsRequest getRequest() {
                OFStatisticsRequest req = new OFStatisticsRequest();
                req.setStatisticType(OFStatisticsType.TABLE);
                req.setLengthU(req.getLengthU());
                return req;            }
        }, switchId, "tables");    }
    // TABLES
    @RequestMapping("/switch/{switchId}/tables/json")
    public View getSwitchTablesJson(@PathVariable String switchId, Map<String,Object> model)
{
        BeaconJsonView view = new BeaconJsonView();
        model.put(BeaconJsonView.ROOT_OBJECT_KEY, getSwitchTables(switchId));
        return view;
    }
    @RequestMapping("/switch/{switchId}/tables")
```

```
    public String getSwitchTables(@PathVariable String switchId, Map<String,Object> model) {
        List<OFStatistics> ports = getSwitchTables(switchId);
        return addStatsSection("tables", switchId, model, ports);    }
    protected List<OFStatistics> getSwitchPorts(String switchId) {
        return getSwitchStats(new OFSRCallback() {
            @Override
            public OFStatisticsRequest getRequest() {
                OFStatisticsRequest req = new OFStatisticsRequest();
                OFPortStatisticsRequest psr = new OFPortStatisticsRequest();
                psr.setPortNumber(OFPort.OFPP_NONE.getValue());
                req.setStatisticType(OFStatisticsType.PORT);
                req.setStatistics(psr);
                req.setLengthU(req.getLengthU() + psr.getLength());
                return req;            }
        }, switchId, "ports");    }
    // PORTS
    @RequestMapping("/switch/{switchId}/ports/json")
    public View getSwitchPortsJson(@PathVariable String switchId, Map<String,Object> model)
{
        BeaconJsonView view = new BeaconJsonView();
        model.put(BeaconJsonView.ROOT_OBJECT_KEY, getSwitchPorts(switchId));
        return view;    }
    @RequestMapping("/switch/{switchId}/ports")
    public String getSwitchPorts(@PathVariable String switchId, Map<String,Object> model) {
        List<OFStatistics> ports = getSwitchPorts(switchId);
        return addStatsSection("ports", switchId, model, ports);    }
    @Autowired
    public void setPackageAdmin(PackageAdmin packageAdmin) {
        this.packageAdmin = packageAdmin;    }}
```

- TopologyWebManageable.java

```
package net.beaconcontroller.topology.web;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import net.beaconcontroller.topology.ITopology;
import net.beaconcontroller.topology.LinkTuple;
import net.beaconcontroller.web.IWebManageable;
import net.beaconcontroller.web.view.BeaconViewResolver;
import net.beaconcontroller.web.view.Tab;
import net.beaconcontroller.web.view.layout.Layout;
import net.beaconcontroller.web.view.layout.TwoColumnLayout;
import net.beaconcontroller.web.view.section.TableSection;
import org.openflow.util.HexString;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
@RequestMapping("/topology")
public class TopologyWebManageable implements IWebManageable {
    protected static Logger log = LoggerFactory.getLogger(TopologyWebManageable.class);
    protected List<Tab> tabs;
    protected ITopology topology;
    public TopologyWebManageable() {
        tabs = new ArrayList<Tab>();
        tabs.add(new Tab("Overview", "/wm/topology/overview.do"));    }
    @Override
    public String getName() {
        return "Topology";    }
    @Override
    public String getDescription() {
        return "View the discovered topology.";    }
    @Override
    public List<Tab> getTabs() {
        return tabs;    }
    @RequestMapping("/overview")
    public String overview(Map<String, Object> model) {
        Layout layout = new TwoColumnLayout();
        model.put("layout", layout);
```

```
        // Listener List Table copied to file topologia.txt
        List<String> columnNames = new ArrayList<String>();
        List<List<String>> cells = new ArrayList<List<String>>();
        columnNames = new ArrayList<String>();
        columnNames.add("Src Id");
        columnNames.add("Src Port");
        columnNames.add("Dst Id");
        columnNames.add("Dst Port");
        cells = new ArrayList<List<String>>();
        for (LinkTuple lt : topology.getLinks().keySet()) {
            List<String> row = new ArrayList<String>();
            row.add(HexString.toHexString(lt.getSrc().getSw().getId()));
            row.add(lt.getSrc().getPort().toString());
            row.add(HexString.toHexString(lt.getDst().getSw().getId()));
            row.add(lt.getDst().getPort().toString());
            cells.add(row);        }
        BufferedWriter writer = null;
        try {
            writer = new BufferedWriter(new FileWriter("topologia.txt"));
            List<String> columnName = new ArrayList<String>();
            columnName = new ArrayList<String>();
            columnName.add("Src Id");
            columnName.add("Src Port");
            columnName.add("Dst Id");
            columnName.add("Dst Port");
            int n = columnName.size();
            for(int i = 0; i < n ; i++){
              writer.write(columnName.get( i ) );
              writer.write("                    ");            }
            writer.newLine();
            for (LinkTuple lt : topology.getLinks().keySet()) {
                List<String> row = new ArrayList<String>();
                row.add(HexString.toHexString(lt.getSrc().getSw().getId()));
                row.add(lt.getSrc().getPort().toString());
                row.add(HexString.toHexString(lt.getDst().getSw().getId()));
                row.add(lt.getDst().getPort().toString());
                for(int i = 0; i < n ; i++){
                    writer.write( row.get( i ) );
                    writer.write("          ");                  }
                writer.newLine();            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                // Close the writer regardless of what happens...
                writer.close();
            } catch (Exception e) {
            }
        Map<String,String> tableOptions = new HashMap<String, String>();
        tableOptions.put("\"bFilter\"", "true");
        TableSection tableSection = new TableSection("Discovered Links", columnNames, cells,
"table-links", tableOptions);
        layout.addSection(tableSection, TwoColumnLayout.COLUMN1);

        return BeaconViewResolver.SIMPLE_VIEW;    }}
    @Autowired
    public void setTopology(ITopology topology) {
        this.topology = topology;    }}
```

- LearningSwitch.java

```
package net.beaconcontroller.learningswitch;
import java.io.IOException;
import java.util.Collections;
import net.beaconcontroller.core.IBeaconProvider;
import net.beaconcontroller.core.IOFMessageListener;
import net.beaconcontroller.core.IOFSwitch;
import net.beaconcontroller.core.IOFSwitchListener;
import net.beaconcontroller.packet.Ethernet;
import net.beaconcontroller.util.LongShortHopscotchHashMap;
import org.openflow.protocol.OFFlowMod;
import org.openflow.protocol.OFMatch;
import org.openflow.protocol.OFMessage;
import org.openflow.protocol.OFPacketIn;
import org.openflow.protocol.OFPacketOut;
import org.openflow.protocol.OFPort;
```

```
import org.openflow.protocol.OFType;
import org.openflow.protocol.action.OFAction;
import org.openflow.protocol.action.OFActionOutput;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class LearningSwitch implements IOFMessageListener, IOFSwitchListener {
    protected static Logger log = LoggerFactory.getLogger(LearningSwitch.class);
    protected IBeaconProvider beaconProvider;

    public void startUp() {
        log.trace("Starting");
        beaconProvider.addOFMessageListener(OFType.PACKET_IN, this);
        beaconProvider.addOFSwitchListener(this);    }
    public void shutDown() {
        log.trace("Stopping");
        beaconProvider.removeOFMessageListener(OFType.PACKET_IN, this);
        beaconProvider.removeOFSwitchListener(this);    }
    public Command receive(IOFSwitch sw, OFMessage msg) throws IOException {
        OFPacketIn pi = (OFPacketIn) msg;
        LongShortHopscotchHashMap macTable = (LongShortHopscotchHashMap)
sw.getLocal().get(LearningSwitch.class);
        if (macTable == null) {
            macTable = new LongShortHopscotchHashMap();
            sw.getLocal().put(LearningSwitch.class, macTable);        }
        // Build the Match
        OFMatch match = OFMatch.load(pi.getPacketData(), pi.getInPort());

        byte[] dlDst = match.getDataLayerDestination();
        byte[] dlSrc = match.getDataLayerSource();
        long dlDstLong = Ethernet.toLong(dlDst);
        long dlSrcLong = Ethernet.toLong(dlSrc);
        int bufferId = pi.getBufferId();
        short outPort = -1;
        // if the src is not multicast, learn it
        if ((dlSrc[0] & 0x1) == 0 && dlSrcLong != 0) {
            if (!macTable.contains(dlSrcLong) ||
                    macTable.get(dlSrcLong) != pi.getInPort()) {
                macTable.put(dlSrcLong, pi.getInPort());
            }        }
        // if the destination is not multicast, look it up
        if ((dlDst[0] & 0x1) == 0 && dlDstLong != 0) {
            outPort = macTable.get(dlDstLong);
        }
        // push a flow mod if we know where the destination lives
        if (outPort != -1) {
            // don't send out the port it came in
            if (outPort == pi.getInPort()) {
                return Command.CONTINUE;            }
            OFActionOutput action = new OFActionOutput(outPort);
            OFFlowMod fm = new OFFlowMod()
                .setBufferId(bufferId)
                .setCommand(OFFlowMod.OFPFC_ADD  )
               //check if there is another flow created and change priority
                .setCommand(OFFlowMod.OFPFF_CHECK_OVERLAP)
                .setIdleTimeout((short) 5)
                .setMatch(match)
                .setPriority((short) 1000)
                .setActions(Collections.singletonList((OFAction)action));
            sw.getOutputStream().write(fm);        }
        // If the destination is unknown or the OFPacketIn was not buffered
        // then send an OFPacketOut.
        if (outPort == -1 || bufferId == OFPacketOut.BUFFER_ID_NONE) {
            OFActionOutput action = new OFActionOutput(
                    (short) ((outPort == -1) ? OFPort.OFPP_FLOOD.getValue()
                            : outPort));

            OFPacketOut po = new OFPacketOut()
                .setBufferId(bufferId)
                .setInPort(pi.getInPort())
                .setActions(Collections.singletonList((OFAction)action));
            // Set the packet data if it is included in the Packet In
            if (bufferId == OFPacketOut.BUFFER_ID_NONE) {
                po.setPacketData(pi.getPacketData());            }
            sw.getOutputStream().write(po);        }
        return Command.CONTINUE;    }
    public String getName() {
        return "switch";    }
```

```
    @Override
    public void addedSwitch(IOFSwitch sw) {
    }
    @Override
    public void removedSwitch(IOFSwitch sw) {
        if (sw.getAttributes().remove(LearningSwitch.class) != null)
            log.debug("Removed l2 table for {}", sw);
    }
    public void setBeaconProvider(IBeaconProvider beaconProvider) {
        this.beaconProvider = beaconProvider;    }}
```

- FlowManager.java

```
package net.beaconcontroller.core.internal;
import java.util.*;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import org.openflow.protocol.OFFlowMod;
import org.openflow.protocol.OFMatch;
import org.openflow.protocol.OFMessage;
import org.openflow.protocol.OFPacketIn;
import org.openflow.protocol.OFType;
import org.openflow.protocol.action.OFAction;
import org.openflow.protocol.action.OFActionOutput;
import org.openflow.util.U16;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import net.beaconcontroller.core.IBeaconProvider;
import net.beaconcontroller.core.IOFMessageListener;
import net.beaconcontroller.core.IOFSwitch;
import net.beaconcontroller.learningswitch.LearningSwitch;
import net.beaconcontroller.core.IOFSwitchListener;
import net.beaconcontroller.packet.Ethernet;
import net.beaconcontroller.packet.IPv4;

//class created for project

public class FlowManager extends LearningSwitch implements IOFSwitchListener ,
IOFMessageListener {
    protected static Logger log = LoggerFactory.getLogger(FlowManager.class);
    protected IBeaconProvider beaconProvider;
    protected Map<Long, List<OFFlowMod>> switchToFlowsMap;
    protected Map<IOFSwitch, Map<Long,Short>> macTables =
            new HashMap<IOFSwitch, Map<Long,Short>>();
    public Command receive(IOFSwitch sw, OFMessage msg) throws IOException {
        OFPacketIn pi = (OFPacketIn) msg;
        Map<Long,Short> macTable = macTables.get(sw);
        if (macTable == null) {
            macTable = new HashMap<Long,Short>();
            macTables.put(sw, macTable);        }
        OFMatch ofMatch = OFMatch.load(pi.getPacketData(), pi.getInPort());
        byte[] dlDst = ofMatch.getDataLayerDestination();
        byte[] dlSrc = ofMatch.getDataLayerSource();
        int nwdst = ofMatch.getNetworkDestination();

        if (nwdst == 167772162){
        Long switchid = sw.getId();
        String line;
        String[] fields;
        FileInputStream fi = new FileInputStream("devicesforIP.txt");
        BufferedReader bfreader = new BufferedReader(new InputStreamReader(fi));
        bfreader.readLine();
        while((line = bfreader.readLine()) != null){
            line = line.trim();
            if ((line.length()!=0) && (line.charAt(0)!='#')){
                fields = line.split("\\s+");
                long switchidfromtxt = Long.valueOf(fields[2]);
                int outport = Integer.parseInt(fields[3]);
                if (switchid == switchidfromtxt ){
                    OFActionOutput action = new OFActionOutput();
                    action.setMaxLength((short) 0);
                    ofMatch.setDataLayerDestination(dlDst);
```

```
                ofMatch.setDataLayerSource(dlSrc);
                ofMatch.setDataLayerType(Ethernet.TYPE_IPv4);
                ofMatch.setNetworkDestination(IPv4.toIPv4Address("10.0.0.2"));
                ofMatch.setWildcards(1048576 );
                ofMatch.setNetworkProtocol((byte)1);
                List<OFAction> actions = new ArrayList<OFAction>();
                actions.add(action);
                OFFlowMod flowmod = new OFFlowMod();
                action.setPort((short)outport);
                flowmod.setOutPort((short)outport);
                flowmod.setType(OFType.FLOW_MOD);
                flowmod.setMatch(ofMatch);
                flowmod.setActions(Collections.singletonList((OFAction)action));
                flowmod.setCommand(OFFlowMod.OFPFC_ADD);
                flowmod.setPriority((short)65535);

          flowmod.setLength(U16.t(OFFlowMod.MINIMUM_LENGTH+OFActionOutput.MINIMUM_LENGT
          H));
                flowmod.setBufferId(0xffffffff);
                flowmod.setIdleTimeout((short)10);
                sw.getOutputStream().write(flowmod);      }            }
    }bfreader.close();   }
    return Command.CONTINUE;       }
public void addedSwitch(IOFSwitch sw) {
}
@Override
public void removedSwitch(IOFSwitch sw) {
    macTables.remove(sw);    }
public void startUp() {
    log.trace("Starting");
    beaconProvider.addOFMessageListener(OFType.PACKET_IN, this);
    beaconProvider.addOFSwitchListener(this);    }
public void shutDown() {
    log.trace("Stopping");
    beaconProvider.removeOFMessageListener(OFType.PACKET_IN, this);

    beaconProvider.removeOFSwitchListener(this);    }
public String getName() {
    return "FlowManager";    }
public void setBeaconProvider(IBeaconProvider beaconProvider) {
    this.beaconProvider = beaconProvider;    } }
```