ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΑΠΤΥΞΗ APPLICATION AWARE CONTROLLER ΣΕ ΠΕΡΙΒΑΛΛΟΝ OPENFLOW

Αλέξανδρος Ν. Ιεροδιακόνου

Επιβλέπων : Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2013

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΑΠΤΥΞΗ APPLICATION AWARE CONTROLLER ΣΕ ΠΕΡΙΒΑΛΛΟΝ OPENFLOW

Αλέξανδρος Ν. Ιεροδιακόνου

Επιβλέπων : Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 6η Ιουνίου 2013.

.....	.....	.....
Ε. Συκάς	Μ. Θεολόγου	Γ. Στασινόπουλος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2013

.....

Αλέξανδρος Ν. Ιεροδιακόνου

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

ΠΕΡΙΛΗΨΗ

Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη και παρουσίαση ενός προγράμματος λογισμικού (εφαρμογής) που εφαρμόζεται σε ένα ελεγκτή δικτύου υπολογιστών, με στόχο την δημιουργία διαδρόμων δεδομένων στο δίκτυο με μεταβλητή προτεραιότητα κίνησης, αναλόγως των εφαρμογών και συσκευών που δραστηριοποιούνται στο δίκτυο και των αποφάσεων του διαχειριστή του δικτύου.

Για την ανάπτυξη της εφαρμογής αυτής χρησιμοποιείται ένα ερευνητικό πρωτόκολλο επικοινωνιών που ακούει στην ονομασία OpenFlow.Η χρήση του OpenFlow είναι σημαντική, διότι προσφέρει την δυνατότητα απόζευξης του πεδίου δεδομένων από το πεδίο ελέγχου του δικτύου. Έτσι υπάρχει ένα κεντροποιημένο δίκτυο που βασίζει την λειτουργία του σε ένα κεντρικό ελεγκτή, αντί σε επιμέρους συσκευές.

Η ανάπτυξη της εφαρμογής προτεραιότητας με γνώση και αναγνώριση των εφαρμογών και συσκευών που χρησιμοποιούν το δίκτυο γίνεται με την χρήση του πλέον βασικού ελεγκτή του πρωτοκόλλου OpenFlow, του ελεγκτή NOX. Βάση των εργαλείων και προγραμματιστικών διεπαφών του NOX, γίνεται η κωδικοποίηση και κατασκευή της εφαρμογής αναγνώρισης των εφαρμογών και συσκευών και δημιουργίας καταλλήλων ροών δεδομένων στο δίκτυο.

Η εφαρμογή αυτή αναπτύσσεται στο δίκτυο OpenFlow του εργαστηρίου Δικτύων Υπολογιστών της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του ΕΜΠ, όπου και γίνεται επίδειξη της επιτυχούς κατασκευής ροών δεδομένων αναλόγως των εφαρμογών και συσκευών που δραστηριοποιούνται στο δίκτυο.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Πρωτόκολλο OpenFlow, NOX controller, ροές, προτεραιότητα δρομολόγησης, εφαρμογή pyswitch, πίνακας ροών, NetFPGA, ρουτίνα dpctl

ABSTRACT

The primary objective of this thesis is the development and presentation of a controller subprogram (application) that is deployed on a computer network's controller, aiming to create specific priority datapaths, accordingly to the applications and the devices active in the network, as well to the decisions of the network's administrator.

For the development of this application, a research communication protocol is used, which is known as OpenFlow. The use of the OpenFlow protocol is important because it offers the decoupling of data fields and control fields in a computer network. This way a centralized network is created, that operates based on central controller software, rather than having to micromanage every device in the network, from routers to end hosts.

The development of this application aware program is based on the tools and application programmatic interfaces (API's) of OpenFlow's most basic controller, the NOX controller. By using the components, ready basic applications and API's of NOX, the application aware program is created on a firm basis, which is capable of routing and creating appropriate data flows, according to the application operating in the network, or to the priority level of a device that is active in the network.

This application is then applied to the OpenFlow network of NTUA's Computer Networks Laboratory, housed in the School of Electrical and Computer Engineering. Then a small demonstration is held, where we show the successful build and deployment of data flows in the network, always accordingly to the application using the network or the priority level of a device present in the network.

KEY WORDS

OpenFlow protocol, NOX controller, flows, routing priority, pyswitch application, flow table, NetFPGA, dpctl script

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε κατά το ακαδημαϊκό έτος 2012-2013, υπό την επίβλεψη του καθηγητή Ευστάθιου Συκά, του τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής, στον οποίο οφείλω ιδιαίτερες ευχαριστίες για την ανάθεσή της. Ιδιαίτερες ευχαριστίες οφείλω επίσης και στον Πάρη Χαραλάμπους, υποψήφιο Διδάκτορα του εργαστηρίου, για τις πολύτιμες συμβουλές και σημαντική προσφορά του στην υλοποίηση και ολοκλήρωση της παρούσας εργασίας. Θα ήθελα να ευχαριστήσω, τέλος, τους γονείς μου, την αδερφή μου και τους φίλους και συναδέλφους μου για την στήριξη και βοήθειά τους, καθ' όλη την διάρκεια των σπουδών μου στο Εθνικό Μετσόβιο Πολυτεχνείο.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΚΟΝΕΣ – ΣΧΗΜΑΤΑ – ΔΙΑΓΡΑΜΜΑΤΑ

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

1.1 Σκοπός της Διπλωματικής Εργασίας

Στην σύγχρονη εποχή η χρήση και αξιοποίηση του διαδικτύου αυξάνεται συνεχώς, πράγμα που οδηγεί στην ανάγκη εξεύρεσης μεθόδων και λύσεων που να ανταποκρίνονται στην καλύτερη χρησιμοποίηση των διαθέσιμων πόρων δικτύωσης και βελτίωσης της ποιότητας και ταχύτητας διασύνδεσης μεταξύ των χρηστών.

Αυτές οι μέθοδοι μπορούν να αναζητηθούν σε πολλαπλά επίπεδα στην δομή του παγκόσμιου ιστού, με το κυριότερο από αυτά να είναι η ίδια η υποδομή του διαδικτύου. Η υποδομή αυτή είναι το σύνολο των δρομολογητών (routers), μεταγωγέων (switches) , εξυπηρετητών (servers) και πελατών (clients) που είναι διασυνδεδεμένοι μεταξύ τους. Βάση αυτών των διασυνδέσεων δημιουργούνται ροές δεδομένων και κίνηση δεδομένων στο δίκτυο.

Στην παραδοσιακή μορφή ενός δικτύου, οι δρομολογητές και οι μεταγωγείς δημιουργούν διαδρομές στο δίκτυο με χρήση διάφορων αλγορίθμων (πχ ελάχιστης απόστασης – Dijkstra) για επικοινωνία των διαφόρων συσκευών μεταξύ τους. Παρόλο που συνήθως επιλέγεται η δρομολόγηση ελάχιστης απόστασης για την επικοινωνία των κόμβων υπάρχουν περιπτώσεις που η επιλογή εναλλακτικών διαδρομών να είναι προτιμότερη λόγω χρήσης συνδέσεων με υψηλότερες ταχύτητες , συνδέσεων με λιγότερο φορτίο κίνησης και συνεπώς μικρότερο χρόνο αναμονής και ούτω καθεξής.

Φυσικά οι απαιτήσεις σε κάθε διασύνδεση μεταξύ των κόμβων του δικτύου εξαρτώνται άμεσα από την λειτουργία που επιτελείται από κάθε εφαρμογή. Σε μια κλήση και επικοινωνία φωνής με χρήση τεχνολογιών όπως VoIP (Voice over IP) είναι αναγκαία η γρήγορη και άμεση μεταφορά των δεδομένων στο δίκτυο, με ανοχές σε περίπτωση απώλειας κάποιων πακέτων δεδομένων. Σε άλλη περίπτωση όπως είναι η μεταφορά δεδομένων πληροφορίας όπως κειμένου ή πολυμεσικών δεδομένων (audio-video) υπάρχουν διαφορικές απαιτήσεις , όπως υψηλότερο εύρος ζώνης,

ανοχή σε περίπτωση χρονικής καθυστέρησης αλλά μηδενική ανοχή σε περίπτωση απώλειας πακέτου δεδομένων.

Έτσι σε αυτή τη διπλωματική εργασία γίνεται μια προσπάθεια δημιουργίας ενός μηχανισμού ελέγχου (application-aware controller) που να δημιουργεί την κατάλληλη δρομολόγηση σε μια γνωστή τοπολογία δικτύου με μεταγωγείς, με το να λαμβάνει υπόψη την εφαρμογή που χρησιμοποιεί το δίκτυο καθώς και τις απαιτήσεις και την τιθέμενη από τον διαχειριστή προτεραιότητα της συσκευής ή εφαρμογής που επίσης δραστηριοποιείται στο δίκτυο. Με αυτό τον τρόπο μπορεί να διασφαλίζεται καλύτερα η ποιότητα υπηρεσίας (QoS – Quality of Service) ανάλογα με την προτεραιότητα της εφαρμογής ή συσκευής που χρησιμοποιεί το δίκτυο.

Για την ανάπτυξη του μηχανισμού γίνεται η χρήση μιας νέας αρχιτεκτονικής, του Openflow, που παρέχει την δυνατότητα δημιουργίας εικονικών δικτυακών υποδομών σε δρομολογητές και δημιουργίας πολλαπλών διαδρομών σε ένα φυσικό δίκτυο χωρίς να είναι αναγκαία η επανεκτέλεση αλγορίθμου δημιουργίας συνδέσεων για κάθε διαδρομή από τον δρομολογητή.

1.2 Διάρθρωση της Διπλωματικής Εργασίας

Η εργασία απαρτίζεται από έξι (6) συνολικά κεφάλαια. Στο παρόν κεφάλαιο γίνεται η πρώτη εισαγωγή και μια επεξήγηση της διάρθρωσης της εργασίας.

Ακολούθως, στο δεύτερο κεφάλαιο γίνεται μια περιγραφή του Openflow ως πρωτόκολλο, οι λόγοι για τους οποίους αναπτύχθηκε και πως εφαρμόζεται σε υπάρχουσες δομές δικτύων. Επίσης γίνεται και μια επεξήγηση για το τι είναι γενικότερα τα Software Defined Networks (SDN’s) και διάφορες υλοποιήσεις αυτών που ήδη υπάρχουν και εφαρμόζονται.

Στο τρίτο κεφάλαιο γίνεται μια περιγραφή του NOX controller, ο οποίος είναι ένας ελεγκτής που βασίζεται στο Openflow για την δημιουργία ροών δρομολόγησης στο δίκτυο. Γίνεται μια ανάπτυξη των δυνατοτήτων που έχει ως εργαλείο δημιουργίας δικτύων καθώς και των συσκευών που υποστηρίζει. Πέραν του NOX όμως υπάρχουν και άλλοι controllers που κυκλοφορούν, είτε στο εμπόριο, είτε για ακαδημαϊκή χρήση και γίνεται μια παρουσίαση αυτών και των συσκευών που υποστηρίζουν.

Μια πιο εις βάθος ανάλυση των προγραμματιστικών εργαλείων και κάποιων βασικών εφαρμογών που παρέχει η προγραμματιστική έκδοση του NOX

(developer's release) γίνεται στο τέταρτο κεφάλαιο. Η δημιουργία του μηχανισμού για κατασκευή ροών με ανάθεση κατάλληλης προτεραιότητας γίνεται εφαρμόζοντας μια σειρά από συνοδευτικές εφαρμογές και εργαλεία (components) του NOX για την δημιουργία και ανάθεση διαφόρων παραμέτρων των ροών. Εξηγείται η δημιουργία ενός flow, όπως αναφέρεται στην ορολογία και συνοδευτικό υλικό του NOX και του Openflow, για την βέλτιστη εξυπηρέτηση των απαιτήσεων της συσκευής που δημιουργεί κίνηση στο δίκτυο.

Στο πέμπτο κεφάλαιο, γίνεται μια περιγραφή των βασικών μηχανισμών και των προγραμματιστικών διεπαφών (API's) που χρησιμοποιούνται για την ανάπτυξη της εφαρμογής της διπλωματικής. Επίσης γίνεται μια περιγραφή του βασικού αλγορίθμου δρομολόγησης του ελεγκτή, της παρεχόμενης εφαρμογής pyswitch.py και παρέχεται η τοπολογία του δικτύου που χρησιμοποιήθηκε στο εργαστήριο, της διάρθρωσης των Openflow switches και του NOX controller. Στο τέλος του κεφαλαίου δίνεται η application aware εφαρμογή ctrlswl3.py, καθώς και η συμπληρωματική device aware εφαρμογή ctrlsw.py και αναλύονται τα βασικά τους χαρακτηριστικά, καθώς και ο τρόπος λειτουργίας τους.

Τέλος στο έκτο κεφάλαιο γίνεται μια ανακεφαλαίωση της όλης διαδικασίας, παρέχονται στιγμιότυπα λειτουργίας (screenshots) του ελεγκτή και των υποκείμενων μεταγωγέων. Αναλύονται τα συμπεράσματα που προκύπτουν από την χρήση της εφαρμογής ctrlswl3.py που αναπτύσσεται στο πέμπτο κεφάλαιο, όσο αφορά την χρήση application-device aware μηχανισμών για την εγκατάσταση ροών στο δίκτυο ανάλογα με την εφαρμογή και με κατάλληλες παραμέτρους προτεραιότητας. Αναλύονται επίσης τα οφέλη και τα μειονεκτήματα που μπορεί να έχει μια αρχιτεκτονική SDN για την δρομολόγηση δεδομένων και την ασφάλεια των δικτύων υπολογιστών στα οποία αναπτύσσεται.

ΚΕΦΑΛΑΙΟ 2

2.1 Το πρωτόκολλο OpenFlow

Το πρωτόκολλο OpenFlow αναπτύχθηκε από το πανεπιστήμιο του Stanford στις Ηνωμένες Πολιτείες το 2008. Είναι ένα έργο ανοικτού κώδικα (open source), το οποίο αποσκοπεί στο να προσφέρει ένα εύκολα προγραμματιζόμενο και ανοικτό περιβάλλον δικτύωσης για δοκιμή και εφαρμογή νέων τεχνολογιών, μεθόδων και αλγορίθμων δρομολόγησης και ασφάλειας δικτύου.

Σαν πρωτόκολλο, το OpenFlow επιλύει ένα βασικό πρόβλημα: την δημιουργία και έλεγχο νέων μεθόδων δικτύωσης σε ήδη υπάρχοντα και λειτουργικά δίκτυα χωρίς να παρεμβαίνει στην λειτουργία των ήδη ενεργών πρωτοκόλλων δρομολόγησης και ασφάλειας. Αυτό λύνει τα χέρια των ερευνητών και μηχανικών στην προσπάθεια ανάπτυξης και εφαρμογής νέων τεχνολογιών δικτύωσης χωρίς να παρεμβαίνουν στην υπάρχουσα υποδομή του δικτύου.

Η ταυτόχρονη λειτουργία του OpenFlow και των παραδοσιακών πρωτοκόλλων σε ένα μηχάνημα επιτυγχάνεται με την χρήση του virtualization (εικονικοποίησης), δηλαδή την ανάπτυξη μιας εικονικής μηχανής σε ένα δρομολογητή (router) ή μεταγωγέα (switch), η οποία τρέχει ταυτόχρονα με το λειτουργικό σύστημα της συσκευής OpenFlow και αναλαμβάνει την διαχείριση των σχετικών πακέτων που λαμβάνονται από την συσκευή (και προωθούνται στην εικονική μηχανή αν ανήκουν στο OpenFlow).

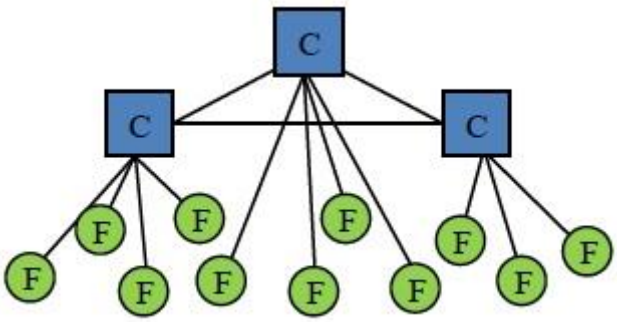
Γενικά το OpenFlow είναι μια πρωτοπόρα τεχνολογία όσο αφορά το software defined networking (SDN) , χρησιμοποιούμενη από αρκετούς ερευνητές λόγω των δυνατοτήτων που αναφέρθηκαν ανωτέρω, καθώς τυγχάνει και ευρείας αποδοχής από κατασκευαστές συσκευών δικτύου. Αυτό συμβαίνει διότι με την χρήση του OpenFlow δεν είναι απαραίτητη οποιαδήποτε τροποποίηση στις εσωτερικές λειτουργίες και αρχιτεκτονική της συσκευής οπότε αποφεύγεται ο κίνδυνος αστοχίας και λανθασμένης λειτουργίας της συσκευής όπως αυτή παρέχεται από τον κατασκευαστή και θα μπορούσε πιθανά να αποτελέσει κίνδυνο για την ακεραιότητα (integrity) και ανθεκτικότητα (robustness) του δικτύου στο οποίο εγκαθίσταται η εν λόγω συσκευή.

2.2 Αρχιτεκτονική του OpenFlow

Η προώθηση των πακέτων μέσω μιας συσκευής δικτύου μπορεί να γίνεται μέσω δρομολόγησης ή μεταγωγής. Οι μεταγωγείς (switches) είναι συσκευές με βασικές δυνατότητες κατεύθυνσης πακέτων από την θύρα εισόδου στην θύρα εξόδου που ορίζεται από τις οδηγίες δρομολόγησης στην επικεφαλίδα του πακέτου και δουλεύουν στο επίπεδο 2 (ζεύξης δεδομένων – data link layer) του πρωτοκόλλου OSI. Για την μεταγωγή οι συσκευές χρησιμοποιούν πίνακες προώθησης βάση των οποίων λαμβάνεται η απόφαση για την θύρα εξόδου. Οι δρομολογητές (routers) είναι πιο πολύπλοκες συσκευές οι οποίες έχουν την δυνατότητα απόφασης δρομολόγησης του πακέτου και που μπορούν να ελέγχουν την επικεφαλίδα IP για τον τελικό προορισμό του πακέτου. Λειτουργούν στο επίπεδο 3 του OSI (επίπεδο δικτύου – network layer) και έχουν την δυνατότητα δημιουργίας και διαχείρισης πολλαπλών δικτύων καθώς και λειτουργίας σαν πύλης εξόδου (gateway) και διαθέτουν πίνακες προώθησης, όπως οι μεταγωγείς. Σημειωτέον ότι ο όρος μεταγωγέας (switch) μπορεί να χρησιμοποιηθεί και σαν ευρύτερη έννοια περιλαμβάνοντας τις δυνατότητες και των δύο ειδών συσκευών που περιγράφονται ανωτέρω, σύμβαση η οποία ακολουθείται και στο παρόν κείμενο.

Η αρχιτεκτονική του πρωτοκόλλου OpenFlow βασίζεται στις αρχές λειτουργίας του software defined networking (SDN). Σύμφωνα με το SDN το κανάλι ελέγχου του δικτύου (control plane, πεδίο ελέγχου – CP) διαχωρίζεται από τον τρόπο που προωθούνται τα πακέτα (data plane, πεδίο δεδομένων – DP). Αυτό μπορεί να επιτευχθεί εφαρμόζοντας το πεδίο ελέγχου σε κατάλληλο λογισμικό σε εξυπηρετητές και το πεδίο δεδομένων σε συσκευές του δικτύου όπως είναι οι μεταγωγείς και οι δρομολογητές.

Αυτό ακριβώς το μοντέλο υιοθετείται και από το OpenFlow, που είναι και ο κυριότερος εκπρόσωπος του SDN, αναθέτοντας τον έλεγχο της δρομολόγησης σε ένα αυτόνομο ελεγκτή (stand-alone controller) ο οποίος τρέχει στον εξυπηρετητή και επικοινωνεί άμεσα με τις υπόλοιπες συσκευές του δικτύου (προώθησης δεδομένων – data forwarding devices) που έχουν την δυνατότητα προώθησης πακέτων συμβατή με την αρχιτεκτονική του OpenFlow. Σχηματικά αυτό παρουσιάζεται πιο κάτω:



Σχήμα 2.2-1 Controller and Forwarding Layers

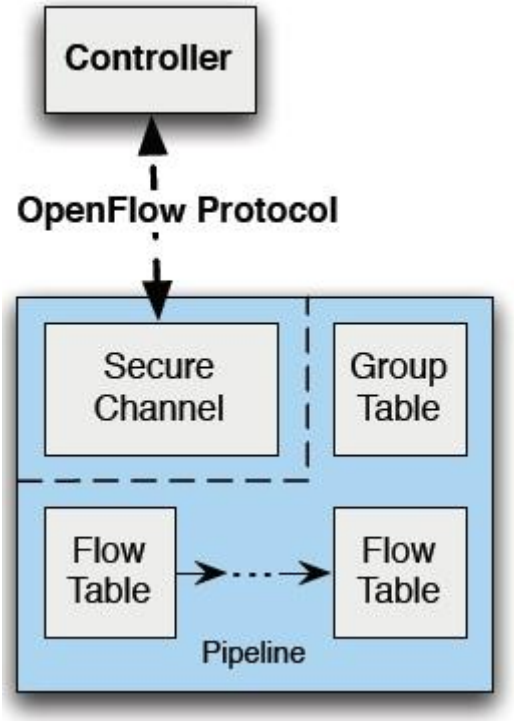
Όπως φαίνεται ανωτέρω, είναι δυνατό να υπάρχουν πολλαπλοί OpenFlow ελεγκτές, οι οποίοι να επικοινωνούν αμφιμονοσήμαντα μεταξύ τους και ο κάθε controller να επικοινωνεί με τις δικές του συσκευές προώθησης πακέτων.

Οι διαδικασίες προώθησης και δρομολόγησης δεδομένων επιτελούνται από τις συσκευές προώθησης βάση των καταχωρίσεων ροής (flow entries) όπως αυτές αποστέλλονται στην συσκευή από τον ελεγκτή. Αυτό προσφέρει σημαντική ευελιξία ως προς την δυναμική ανάθεση πόρων από τον ελεγκτή καθώς και την δυνατότητα δημιουργίας ροών δεδομένων στο δίκτυο ανάλογα με την ζήτηση και την εφαρμογή που εκτελείται ανά δεδομένη χρονική στιγμή.

Αυτές οι καταχωρίσεις ροής, τα entries, εγγράφονται στους πίνακες προώθησης των συσκευών (δρομολογητών και μεταγωγέων) και οι οποίοι πλέον ονομάζονται πίνακες ροής (flow tables). Αναλυτικότερα τα flow entries περιγράφονται στο κεφάλαιο 2.4.

2.3 Το OpenFlow Switch

Το OF switch μπορεί να είναι οποιαδήποτε συσκευή προώθησης πακέτων (δρομολογητής – router και μεταγωγέας – switch) που να διαθέτει ένα η περισσότερους πίνακες ροής (flow tables) , ένα πίνακα ομαδοποίησης (group table) και δυνατότητα χρήσης ενός ξεχωριστού καναλιού για άμεση επικοινωνία με τον OF ελεγκτή. Μέσω αυτού του καναλιού γίνεται η διαχείριση του μεταγωγέα από τον ελεγκτή και μπορούν να προστεθούν, ενημερωθούν ή να αφαιρεθούν καταχωρίσεις ροής (flow entries) από τα flow tables.



Σχήμα 2.3-1 Δομή του OpenFlow switch

Κάθε flow entry σε ένα flow table απαρτίζεται από τρία πεδία, το πεδίο αντιστοίχισης (match field), το πεδίο μετρητών (counters field) και το πεδίο εντολών (instructions field). Βάση αυτών των πεδίων γίνεται η αντιστοίχιση και δρομολόγηση των λαμβανομένων από το μεταγωγέα πακέτων.

Ο μεταγωγέας με την χρήση του εκάστοτε flow entry προωθεί τα πακέτα σε μια θύρα εξόδου (out port) από μια θύρα εισόδου (in port). Η θύρα αυτή μπορεί να είναι φυσική θύρα του μεταγωγέα ή και εικονική θύρα, όπως αυτή τίθεται από την εικονική μηχανή (virtual machine) που τρέχει στον μεταγωγέα και ορίζεται από την συσκευή για αυτό το σκοπό, την χρήση της για εφαρμογή του πρωτοκόλλου OF. Επίσης μπορεί να υπάρχουν δεσμευμένες θύρες στον μεταγωγέα, οι οποίες να

χρησιμοποιούνται για συγκεκριμένες λειτουργίες προώθησης , όπως αποστολή πακέτων στον ελεγκτή, μαζική προώθηση σε όλες τις συσκευές (flooding) ή και χρήση non-OpenFlow μεθόδων προώθησης όπως λειτουργεί εξ' ορισμού (by default) ο δρομολογητής.

Πέρα της διαχείρισης κάθε πακέτου ξεχωριστά, ο μεταγωγέας μπορεί να χρησιμοποιήσει το group table για πιο μαζική επεξεργασία της κίνησης. Ένα flow entry μπορεί να αντιστοιχεί σε ένα group table action (ενέργεια που ορίζεται στο group table) για πακέτα που αντιστοιχούν στο εν λόγω flow entry. Τέτοιες ενέργειες που επιτελούνται βάση και με χρήση του group table και των group entries που αυτό περιλαμβάνει μπορούν να είναι πιο πολύπλοκες, όπως η πολυδιόδευση πακέτων (multipath forwarding), γρήγορη επαναδρομολόγηση (fast reroute) ή η συσσωμάτωση ζεύξεων (link aggregation). Αναλυτικότερα το group table περιγράφεται στο κεφάλαιο 2.5.

Τα flow tables σε ένα switch βρίσκονται σε διασωληνωμένη μορφή (pipelined) και ανάλογα με την μορφή της διασωλήνωσης οι μεταγωγείς διακρίνονται σε δύο είδη:

Α. Οι OpenFlow-only μεταγωγείς, οι οποίοι υποστηρίζουν μόνο λειτουργίες OpenFlow και όλα τα πακέτα επεξεργάζονται με την OF διασωλήνωση του μεταγωγέα.

Β. Οι OpenFlow-hybrid μεταγωγείς, οι οποίοι υποστηρίζουν και λειτουργία OpenFlow καθώς και λειτουργία Ethernet (κλασσική λειτουργία OSI layer 2 και 3 δρομολόγησης). Αυτοί οι μεταγωγείς περιλαμβάνουν ένα μηχανισμό κατηγοριοποίησης που ανακατευθύνει τα ληφθέντα πακέτα είτε στην διασωλήνωση OF είτε στη κανονική διασωλήνωση Ethernet. Ο μηχανισμός αυτός μπορεί να βασίζεται στην θύρα εισόδου των πακέτων ή πιθανά σε κάποια ετικέτα (tag) στην επικεφαλίδα (header) των πακέτων.

Οι σχεδιαστές των μεταγωγέων μπορούν να σχεδιάζουν την εσωτερική δομή των συσκευών κατά βούληση, με τον περιορισμό ότι για να πληρούν τις προϋποθέσεις για το πρωτόκολλο OpenFlow οι συσκευές πρέπει να διαθέτουν την βασική αρχιτεκτονική που περιγράφηκε ανωτέρω. Φυσικά μπορούν να γίνουν διαφοροποιήσεις που να είναι συμβατές με το OF, για παράδειγμα η χρήση μιας μάσκας bit (bitmask) για προώθηση πακέτων σε πολλαπλές θύρες σε ένα flow entry, αντί να γίνει χρήση του group table.

2.4 Flow table και Flow entries

Η δομή ενός Flow entry είναι η ακόλουθη:

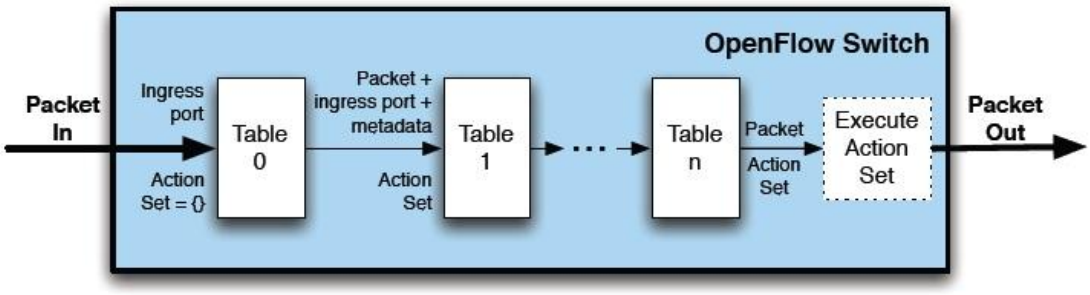
Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Σχήμα 2.4-1 Δομή Flow entry

Πολλά τέτοια entries είναι καταχωρημένα σε ένα πίνακα που λέγεται flow table. Βάση αυτών των καταχωρίσεων γίνεται η δρομολόγηση των πακέτων από το πρωτόκολλο OF. Πιο αναλυτικά τα πεδία ενός flow entry είναι:

Τα ¨match fields¨ είναι τα πεδία αντιστοίχισης για τα ληφθέντα πακέτα. Αυτά περιέχουν την θύρα εισόδου (ingress port), τις επικεφαλίδες και θύρες πακέτων (headers,ports) και προαιρετικά κάποια μεταδεδομένα (metadata). Οι ¨counters¨ είναι διάφοροι μετρητές που αφορούν τα ληφθέντα πακέτα, για παράδειγμα πόσα αντιστοιχήθηκαν επιτυχώς ή πόσα απορρίφθηκαν. Τα ¨Timeouts" περιλαμβάνουν τον χρόνο αναμονής και χρόνο λήξης της ροής και το ¨cookie¨ είναι διάφορα δεδομένα που αποθηκεύει ο ελεγκτής στην ροή και μπορούν να χρησιμοποιηθούν για το φιλτράρισμα στατιστικών της ροής ή των αλλαγών της.Οι εντολές ¨instructions¨ αφορούν οδηγίες για την επιτέλεση ή τροποποίηση κάποιων ενεργειών που γίνονται κατά την διάρκεια της επεξεργασίας του πακέτου. Η προτεραιότητα, ¨Priority¨ είναι η τιμή με την οποία διαμορφώνεται η διαδικασία αντιστοίχισης (matching) από ένα μεταγωγέα, με ροές που έχουν υψηλότερη τιμή προτεραιότητας να προηγούνται κατά την αναζήτηση από άλλες που έχουν χαμηλότερη τιμή.

Για την διαδοχική σύγκριση στα flow tables το OF ακολουθεί διασωληνωμένη λογική (pipeline) δημιουργώντας έτσι μια σταθερή ροή πακέτων μέσω των flow tables, όπως παρουσιάζεται ακολούθως:



Σχήμα 2.4-2 Διαδρομή πακέτου μέσω των flow tables

Όλα τα flow tables αριθμούνται ακολουθιακά, αρχίζοντας από το 0. Η επεξεργασία αρχίζει ακολουθιακά :

Για να γίνει η αντιστοίχιση κατά την λήψη ενός πακέτου γίνεται σύγκριση των δεδομένων της επικεφαλίδας του πακέτου, της θύρας εισόδου του πακέτου και τυχόν μεταδεδομένων διαδοχικά με το πρώτο flow entry του πρώτου flow table και σταδιακά όλων των επόμενων entries στα υπόλοιπα flow tables. Η αντιστοίχιση γίνεται κατά σειρά προτεραιότητας, δηλαδή η πρώτη επιτυχής αντιστοίχιση flow entry είναι αυτή που ακολουθείται, με τα επόμενα entries στα υπόλοιπα flow tables να χρησιμοποιούνται ανάλογα με τις οδηγίες του matched entry .

Όταν επιτύχει η αντιστοίχιση σε ένα flow entry εκτελείται το αντίστοιχο σετ εντολών (instruction set) που περιλαμβάνει το entry. Οι εντολές αυτές μπορεί να ανακατευθύνουν το πακέτο σε άλλο flow entry σε επόμενο flow table όπου η διαδικασία προώθησης επαναλαμβάνεται (εντολές Goto). Σημειωτέον ότι ένα flow entry μπορεί να κατευθύνει ένα πακέτο σε flow table με αύξοντα αριθμό μεγαλύτερο από το τρέχον flow table (στο οποίο βρίσκεται το entry). Έτσι η αρχιτεκτονική διασωλήνωσης μπορεί να λειτουργεί μόνο με προώθηση προς τα εμπρός και όχι πίσω.

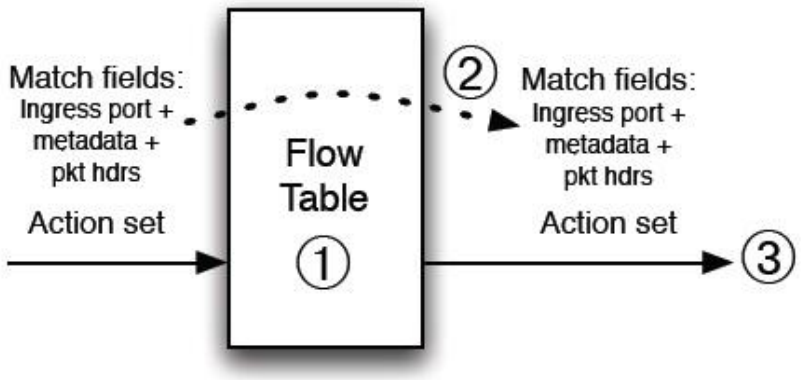
Όταν το πακέτο φτάσει στο τέλος της διασωλήνωσης και δεν υπάρχει άλλη εντολή προώθησης τότε το πακέτο επεξεργάζεται με τις εντολές του flow entry και προωθείται στην ανάλογη θύρα εξόδου.

Γενικά κατά την εκτέλεση οποιασδήποτε εντολής σε ένα flow table η επεξεργασία γίνεται όπως στο σχήμα 2.2.2-3:

i. Με την τροποποίηση του πακέτου ανάλογα με την εντολή που εκτελείται και ενημέρωση των πεδίων αντιστοίχισης (επικεφαλίδα, θύρα εισόδου, μεταδεδομένα)

ii. Ενημέρωση του πεδίου εντολών του πακέτου (εκκαθάριση εντολών ή εισαγωγή νέων εντολών για μετέπειτα επεξεργασία)

iii. Ενημέρωση μεταδεδομένων πακέτου



Σχήμα 2.4-3 Επεξεργασία πακέτου σε flow table

Στην περίπτωση που ένα πακέτο δεν αντιστοιχηθεί τότε έχουμε αστοχία πίνακα – table miss. Η διαδικασία που ακολουθείται για τέτοια πακέτα εξαρτάται από την πολιτική του εκάστοτε δικτύου OpenFlow. Εξ' ορισμού τα πακέτα στέλνονται στον ελεγκτή μέσω του καναλιού ελέγχου (control channel). Επίσης υπάρχει και η επιλογή απόρριψης του πακέτου (drop).

2.5 Το Group table

Ένας ακόμη είδος πινάκων που ορίζεται από την αρχιτεκτονική του OpenFlow είναι τα group tables, τα οποία εμπεριέχουν καταχωρίσεις που ονομάζονται group entries. Με την χρήση αυτών των tables είναι δυνατή η προώθηση ομάδων ή και όλων των πακέτων που βρίσκονται σε αντιστοιχία με ένα group entry. Η δομή ενός group entry είναι:

Group Identifier	Group Type	Counters	Action Buckets

Σχήμα 2.5-1 Group Entry

Αναλυτικά τα πεδία του group entry είναι:

i. Το group identifier, αναγνωριστικό ομάδας, που είναι ένας μη προσημασμένος ακέραιος αριθμός των 32bit και χρησιμεύει για την ταυτοποίηση της ομάδας πακέτων που αφορούν το συγκεκριμένο entry.

ii. Το group type, ο τύπος της ομάδας, που δείχνει ποιες εντολές από την ομάδα εντολών (action bucket) θα εκτελεστούν για τα πακέτα που ανήκουν στο entry.

iii. Οι διάφοροι μετρητές – counters, που χρησιμοποιούνται για διάφορες πληροφορίες, όπως αριθμός πακέτων που ανήκουν στο group entry.

iv. Οι ομάδες εντολών, action buckets, μια διατεταγμένη λίστα εντολών στην οποία κάθε εντολή περιέχει ένα σύνολο ενεργειών και τις σχετικές παραμέτρους για να εφαρμοστούν στο προς επεξεργασία πακέτο.

Όσο αφορά τα group types, υπάρχουν τέσσερις κατηγορίες:

1. Κατηγορία all: Εκτελούνται όλες οι εντολές στις ομάδες εντολών, με το πακέτο να κλωνοποιείται σε αντίγραφα και να επεξεργάζεται από κάθε εντολή στο action bucket. Αυτή η κατηγορία χρησιμοποιείται συνήθως για ευρυεκπομπή (broadcasting) και πολυεκπομπή (multicasting).
2. Κατηγορία select: Εκτελείται μόνο μια εντολή από το σύνολο της ομάδας εντολών, βάση των παραμέτρων του πακέτου και των αλγορίθμων επιλογής της εκάστοτε συσκευής.
3. Κατηγορία indirect: Εκτελείται μια μόνο προκαθορισμένη εντολή στο group table, πράγμα που επιτρέπει πολλαπλές ροές πακέτων να δείχνουν σε ένα

αναγνωριστικό ομάδας (group identifier) και να υποστηρίζεται γρηγορότερη και αποτελεσματικότερη προώθηση ομάδων πακέτων σε συγκεκριμένο προορισμό.

4. Κατηγορία fast failover: Εκτελείται η πρώτη εν ενεργεία ομάδα εντολών, δηλαδή το σύνολο εντολών που είναι συσχετισμένο με μια ενεργή θύρα εξόδου. Με αυτή την μέθοδο κάθε δέσμη ενεργειών στο action bucket συσχετίζεται με μια θύρα εξόδου για να μπορεί η συσκευή δρομολόγησης σε περίπτωση απώλειας σύνδεσης από μια θύρα εξόδου να επαναδρομολογήσει πακέτα σε εφεδρικές θύρες χωρίς να χρειαστεί ξανά επικοινωνία με τον controller.

2.6 Διαδικασία Αντιστοίχισης – Matching

Κατά την λήψη ενός πακέτου από μια συσκευή OpenFlow εκτελείται μια διαδικασία αναζήτησης και ταύτισης των flow entries στα flow tables της συσκευής με τα δεδομένα που φέρει το πακέτο στην επικεφαλίδα του (header).

Η αντιστοίχιση ενός πακέτου με ένα flow entry θεωρείται επιτυχής όταν τα πεδία αντιστοίχισης (match fields) ταυτίζονται ακριβώς με τα πεδία που ορίζονται στο flow entry. Μπορεί όμως ένα πεδίο στο flow entry να έχει την τιμή ANY, οπότε και αντιστοιχίζονται όλες οι πιθανές τιμές του εισερχόμενου πακέτου για το πεδίο αυτό.

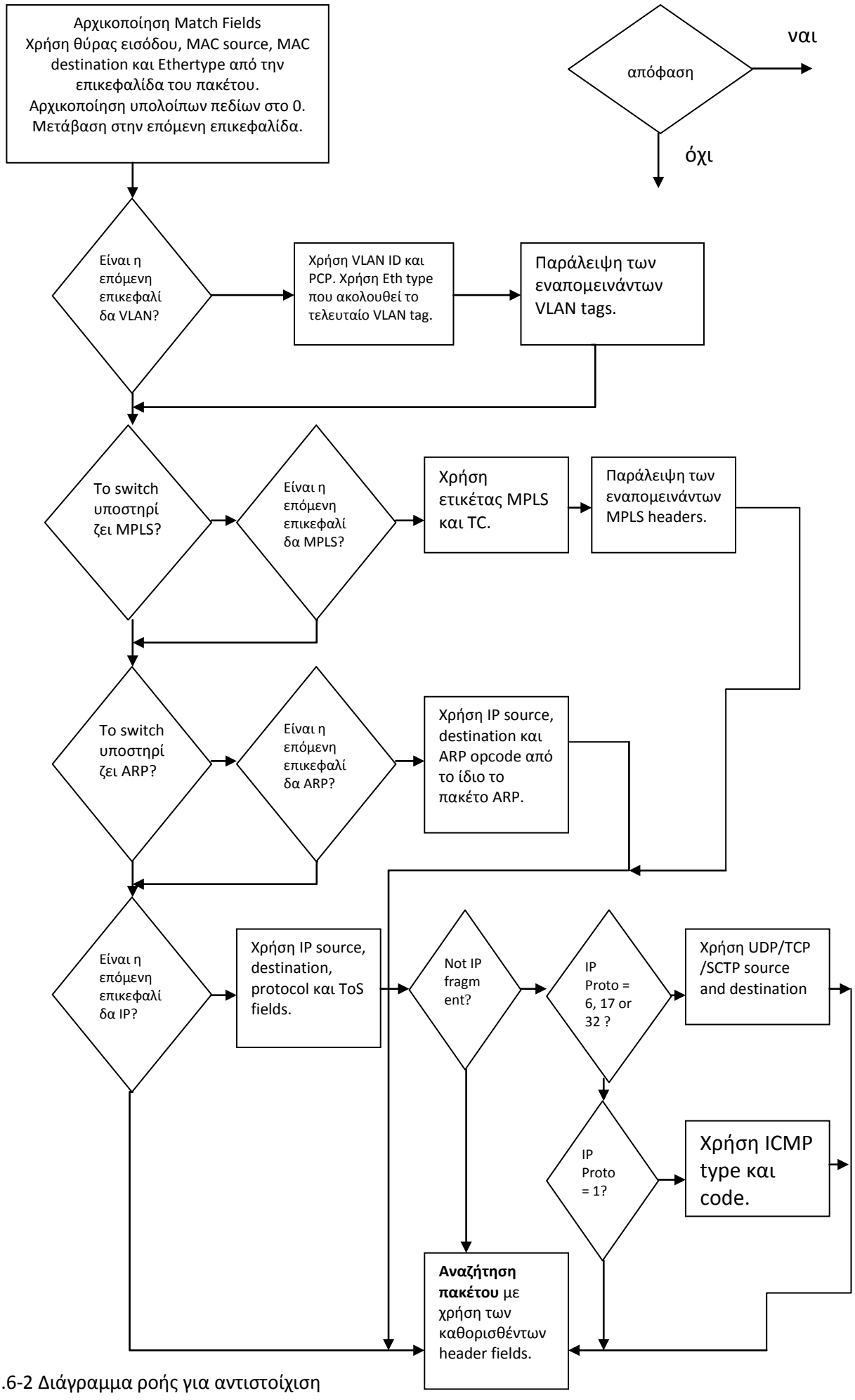
Η σάρωση των πεδίων των πακέτων στο OpenFlow βασίζεται στην δομή των πλαισίων όπως ορίζονται από το πρωτόκολλο Ethernet. Λόγω όμως του ότι υπάρχουν πολλά είδη πλαισίωσης στο Ethernet η αντιστοίχιση από το OpenFlow γίνεται βάση του τι θεωρείται από το OpenFlow ως το ωφέλιμο φορτίο (payload) του πακέτου. Έτσι μπορεί να εντοπιστεί το πεδίο Ethertype στο πλαίσιο Ethernet και να καθοριστεί ποιό πρωτόκολλο είναι ενθυλακωμένο στο πλαίσιο (συνήθως MPLS, Internet Protocol (IP), ARP) όπως στο πιο κάτω σχήμα.

<table>
<tr><td>Preamble</td><td>Start of frame delimiter</td><td>Προορισμός MAC</td><td>Πηγή MAC</td><td>Ετικέτα VLAN</td><td>Ethertype (Ethernet II) or length (IEEE 802.3)</td><td>Payload</td><td>32‑bit CRC</td><td>Interframe gap</td></tr>
<tr><td>7 octets</td><td>1 octet</td><td>6 octets</td><td>6 octets</td><td>4 octets</td><td>2 octets</td><td>42–1500 octets</td><td>4 octets</td><td>12 octets</td></tr>
<tr><td colspan="2"></td><td colspan="6">64-1522 octets</td><td></td></tr>
<tr><td colspan="8">72-1530 octets</td><td></td></tr>
<tr><td colspan="9">84-1542 octets</td></tr>
</table>

Σχήμα 2.6-1 Το Ethernet frame

Για παράδειγμα κατά την λήψη πακέτου με ετικέτα VLAN (virtual local area network) το Ethertype βρίσκεται αμέσως μετά το πεδίο ετικέτα VLAN στο πιο πάνω σχήμα. Εξαίρεση αποτελεί το MPLS (multiprotocol layer switching) που λόγω της δομής του, το OpenFlow δεν μπορεί να προσδιορίσει το πρωτόκολλο που ενθυλακώνεται στο πλαίσιο, οπότε και χρησιμοποιείται η ετικέτα MPLS για την αντιστοίχιση.

Πιο παραστατικά η πλήρης διαδικασία για την αντιστοίχιση φαίνεται στο ακόλουθο διάγραμμα ροής:



Σχήμα 2.6-2 Διάγραμμα ροής για αντιστοίχιση

2.7 Οδηγίες και Δέσμες Ενεργειών – Instructions and Action Sets

Οι οδηγίες (instructions) αποτελούν το τρίτο πεδίο σε ένα flow entry και εκτελούνται όταν ένα πακέτο ταυτιστεί με το entry. Αυτές οι οδηγίες μπορεί να μεταβάλουν το πακέτο, τη δέσμη ενεργειών (action set) που είναι συσχετισμένη με το πακέτο (που εξ' ορισμού είναι άδεια) και την πορεία του πακέτου μέσα στην διασωλήνωση των flow tables.

Τέτοιες οδηγίες μπορεί να είναι:

- Apply-Actions: Εφαρμόζονται άμεσα οι ενέργειες που είναι συσχετισμένες με το πακέτο χωρίς καμιά μεταβολή στην δέσμη ενεργειών.
- Clear-Actions: Διαγράφονται όλες οι ενέργειες στην δέσμη ενεργειών.
- Write-Actions: Ενσωματώνει τις καθορισμένες από το flow entry ενέργειες στην δέσμη ενεργειών του πακέτου.
- Write-Metadata: Γράφει τα μεταδεδομένα στο αντίστοιχο πεδίο του match field.
- Goto-Table: Υποδεικνύει το επόμενο table για μετάβαση μέσα στην διασωλήνωση.

Οι δέσμες ενεργειών (action sets) είναι το σύνολο εντολών που συσχετίζονται με ένα πακέτο. Πιο συγκεκριμένα, είναι οι εντολές που εφαρμόζονται στο πακέτο κατά την επεξεργασία του μέσω της διασωλήνωσης της συσκευής και ορίζονται από το πεδίο¨ instructions¨ μέσω του οποίου εγγράφονται, μεταβάλλονται ή διαγράφονται οι εν λόγω εντολές. Έτσι κάθε δέσμη εντολών που αφορά ένα πακέτο μεταφέρεται μεταξύ των flow tables κατά την επεξεργασία. Όταν η οδηγία δεν είναι η ¨Goto-Table¨ τότε σταματά η μετάβαση στην διασωλήνωση και εκτελούνται οι εντολές στην δέσμη ενεργειών.

Κάθε δέσμη ενεργειών μπορεί να έχει το μέγιστο μια εντολή για κάθε τύπο πακέτου (για IP,MPLS,ARP,VLAN). Αν χρειάζονται πολλαπλές εντολές για τον ίδιο τύπο, μπορεί να χρησιμοποιηθεί η οδηγία Apply-Actions.

Οι εντολές που υποστηρίζονται από το OpenFlow είναι οι εξής:

1. Output: Προώθηση πακέτου σε μια προκαθορισμένη θύρα
2. Output ALL: Προώθηση του πακέτου σε όλες τις θύρες εξόδου αλλά όχι στην θύρα εισόδου του πακέτου και σε θύρες που τέθηκαν OFPPC_NO_FWD.
3. Output CONTROLLER: Ενθυλάκωση και προώθηση του πακέτου στον ελεγκτή.

4. Output TABLE: Υποβολή του πακέτου στο πρώτο flow table για να αρχίσει η επεξεργασία του στην διασωλήνωση.
5. Output IN_PORT: Αποστολή του πακέτου πίσω και μέσω της θύρας εισόδου του.
6. Output LOCAL: Αποστολή του πακέτου στην θύρα τοπικού δικτύου της συσκευής για να μπορούν να έχουν πρόσβαση άμεσα στο πακέτο μέσω του OpenFlow οι συσκευές τοπικού δικτύου.
7. Output NORMAL: Επεξεργασία του πακέτου με το εξ' ορισμού πρωτόκολλο της συσκευής (IP για παράδειγμα) και όχι μέσω της διασωλήνωσης OpenFlow.
8. Output FLOOD: Αποστολή του πακέτου σε όλες τις θύρες εξόδου της συσκευής, εκτός της θύρας εισόδου του πακέτου και θύρες που τέθηκαν OFPPS_BLOCKED.
9. Set-Queue: Ορίζεται αριθμός ουράς του πακέτου, που όταν το πακέτο εξέρχεται από μια θύρα της συσκευής, ο αριθμός αυτός καθορίζει ποια ουρά συνδεδεμένη στην εν λόγω θύρα χρησιμοποιείται για την προώθηση του πακέτου.
10. Drop: Χωρίς να ορίζεται κάποια συγκεκριμένη ενέργεια για την εντολή αυτή, το πακέτο αυτό εξέρχεται από την διασωλήνωση και παραλείπεται από την συσκευή.
11. Group: Το πακέτο επεξεργάζεται μέσω του καθορισμένου group.
12. Push-Tag/Pop-Tag: Όταν η συσκευή έχει την δυνατότητα, μπορούν να εισαχθούν ή να εξαχθούν ετικέτες από την επικεφαλίδα του πακέτου (Ethernet, VLAN, MPLS, ARP/IP, TCP/UDP/SCTP).
13. Set-Field: Οι διάφορες εντολές set-field μπορούν να τροποποιήσουν τις τιμές των αντίστοιχων πεδίων της επικεφαλίδας του πακέτου.

2.8 Το κανάλι OpenFlow – OpenFlow Channel

Το κανάλι OpenFlow είναι η διεπαφή που συνδέει ένα OpenFlow μεταγωγέα με τον ελεγκτή. Μέσω αυτής της διεπαφής ο ελεγκτής ρυθμίζει και ελέγχει τον μεταγωγέα, λαμβάνοντας συμβάντα και στέλνοντας πακέτα στον μεταγωγέα. Τα μηνύματα που ανταλλάσσονται μεταξύ ελεγκτή – μεταγωγέα πρέπει να είναι διαμορφωμένα σύμφωνα με το πρωτόκολλο OpenFlow και για λόγους ασφαλείας το κανάλι είναι κρυπτογραφημένο.

Η επικοινωνία μεταξύ ελεγκτή – μεταγωγέα στο κανάλι OpenFlow γίνεται με χρήση συνδέσεων TCP ή TLS για αυξημένη ασφάλεια. Το TLS (Transport Layer Security) είναι πρωτόκολλο κρυπτογράφησης ασφαλούς μεταφοράς, εξέλιξη από το γνωστό SSL (Secure Sockets Layer). Ο μεταγωγέας μπορεί να επικοινωνήσει με τον ελεγκτή μέσω αυτών των συνδέσεων χρησιμοποιώντας την IP διεύθυνση του ελεγκτή και θύρα εισόδου που ορίζεται από τον χρήστη. Όταν γίνεται χρήση του TLS μπορεί να χρησιμοποιηθεί η εξ' ορισμού θύρα TCP που είναι η 6633.

Το πρωτόκολλο OpenFlow υποστηρίζει τρεις τύπους μηνυμάτων, controller-to-switch, asynchronous και symmetric, καθένα με πολλαπλούς υποτύπους.

Τα μηνύματα controller-to-switch στέλνονται από τον ελεγκτή και χρησιμοποιούνται για τον απευθείας έλεγχο του switch ή την ανασκόπηση της κατάστασης του μεταγωγέα. Αυτά τα μηνύματα είναι:

- Features: Ο ελεγκτής ζητά να μάθει τις δυνατότητες και τα χαρακτηριστικά του μεταγωγέα με μήνυμα features request και ο μεταγωγέας απαντάει με μήνυμα features response.
- Configuration: Ο ελεγκτής θέτει και ρυθμίζει τις παραμέτρους λειτουργίας του μεταγωγέα.
- Modify-State: Αυτά τα μηνύματα μεταβάλλουν την κατάσταση λειτουργίας των μεταγωγέων στο δίκτυο και χρησιμοποιούνται για την προσθαφαίρεση και αλλαγή των flow entries ή group entries στα αντίστοιχα tables των συσκευών καθώς και την ρύθμιση λειτουργίας τω θυρών των συσκευών.
- Read-State: Ο ελεγκτής συλλέγει στατιστικά στοιχεία για την λειτουργία και τρέχουσα κατάσταση των μεταγωγέων στο δίκτυο.
- Packet-Out: Αυτά τα μηνύματα χρησιμοποιούνται από τον ελεγκτή για την αποστολή πακέτων μέσω μιας προκαθορισμένης θύρας σε ένα μεταγωγέα καθώς και για την προώθηση πακέτων που λήφθησαν με μήνυμα Packet-In.
- Barrier: Barrier request/reply μηνύματα χρησιμοποιούνται από τον ελεγκτή για να διασφαλίζεται ότι οι διάφορες παράμετροι και εξαρτήσεις των

πακέτων τηρούνται κατά την επεξεργασία τους, καθώς και για λήψη ειδοποιήσεων όταν περατώνεται μια διεργασία.

Τα asynchronous μηνύματα στέλνονται από τον μεταγωγέα και χρησιμοποιούνται για την ενημέρωση του ελεγκτή σε ότι αφορά συμβάντα δικτύου και αλλαγές στην κατάσταση του μεταγωγέα:

- Packet-In: Για όλα τα πακέτα που δεν υπάρχει επιτυχής αντιστοίχιση με ένα flow entry στέλνεται ένα Packet-In συμβάν από τον μεταγωγέα στον ελεγκτή. Επίσης για πακέτα που προωθούνται στην εικονική θύρα του ελεγκτή πάλι αποστέλλεται συμβάν Packet-In στον ελεγκτή. Αν ο μεταγωγέας έχει αρκετή μνήμη στην προσωρινή του μνήμη (buffer) μπορεί να αποθηκεύσει το μήνυμα και να στείλει στον ελεγκτή το Packet-In συμβάν με ένα μέρος της επικεφαλίδας του πακέτου και ένα αύξων αριθμό buffer για να γνωρίζει ο ελεγκτής ποιο είναι το πακέτο. Αν ο μεταγωγέας δεν έχει αρκετή μνήμη ή εσωτερικό buffer τότε αποστέλλει ολόκληρο το πακέτο στον ελεγκτή με το Packet-In συμβάν.
- Flow-Removed: Όταν ένα flow entry προστίθεται στον μεταγωγέα από μήνυμα Modify-State τίθεται και μια τιμή αφαίρεσης του entry σε περίπτωση που δεν χρησιμοποιηθεί για κάποιο διάστημα (idle timeout) καθώς επίσης και μια τιμή αφαίρεσης του entry ανεξαρτήτως δραστηριότητας. Έτσι κατά την αφαίρεση του flow entry στέλνεται από τον μεταγωγέα το μήνυμα Flow-Removed για ενημέρωση του ελεγκτή. Flow-Removed μήνυμα παράγεται επίσης και όταν ζητηθεί από η διαγραφή ενός entry από τον ελεγκτή.
- Port-Status: Ο μεταγωγέας αναμένεται να αποστείλει Port-Status μηνύματα για κάθε αλλαγή στην κατάσταση των θυρών του, αν μια θύρα για παράδειγμα απενεργοποιηθεί από τον χρήστη.
- Error: Μηνύματα error αποστέλλονται κάθε φορά που ο μεταγωγέας αδυνατεί να αντιμετωπίσει ένα πρόβλημα σύνδεσης, δρομολόγησης ή χειρισμού πακέτου.

Τέλος, τα μηνύματα symmetric μπορούν να σταλούν και από τις δύο μεριές (ελεγκτής ή μεταγωγέα) και αφορούν λειτουργίες γενικού σκοπού :

- Hello: Μηνύματα Hello ανταλλάσσονται μεταξύ ελεγκτή και μεταγωγέα κατά την αρχικοποίηση μιας σύνδεσης.
- Echo: Echo request/reply μηνύματα μπορεί να σταλούν είτε από τον ελεγκτή είτε από τον μεταγωγέα και πάντα πρέπει να επιστρέφεται ένα reply. Αυτά τα μηνύματα χρησιμοποιούνται για την μέτρηση της καθυστέρησης ή του

εύρους ζώνης σε μια σύνδεση καθώς και για τον έλεγχο αν είναι ακόμα ενεργή η σύνδεση.

- Experimenter: Αυτά τα μηνύματα παρέχουν ένα πρότυπο (standard) τρόπο για χρήση πρόσθετων ή πειραματικών δυνατοτήτων του OpenFlow από τον μεταγωγέα με χρήση του χώρου των μηνυμάτων OpenFlow.

2.9 Μηνύματα τροποποίησης του Flow table και μηνύματα λάθους

Τα μηνύματα που χρησιμοποιούνται για την τροποποίηση των flow entries στα flow tables έχουν τους ακόλουθους τύπους:

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

Για αιτήσεις προσθήκης flow entry, τύπου OFPFC_ADD, ο μεταγωγέας πρέπει να πρώτα να ελέγξει αν υπάρχουν επικαλυπτόμενα entries στο flow table. Επικάλυψη καταχωρίσεων έχουμε όταν ένα πακέτο μπορεί να αντιστοιχιστεί σε δύο ή περισσότερα entries με αυτά να έχουν την ίδια προτεραιότητα εκτέλεσης. Αν ανιχνευτεί μια τέτοια επικάλυψη, ο μεταγωγέας πρέπει να αρνηθεί την προσθήκη με μήνυμα ofp_error_msg τύπου OFPET_FLOW_MOD_FAILED.

Για έγκυρες, μη επικαλυπτόμενες καταχωρίσεις ο μεταγωγέας εγγράφει την νέα καταχώριση στο ζητούμενο flow table. Αν μια καταχώριση με τα ίδια πεδία ταύτισης (match fields) και προτεραιότητα εκτέλεσης ήδη υπάρχει τότε διαγράφεται πλήρως από το flow table προτού εγγραφεί η νέα καταχώριση.

Για αιτήσεις τροποποίησης τύπου OFPFC_MODIFY ή OFPFC_MODIFY_STRICT αν μια καταχώριση υπάρχει στο flow table που ζητείται από την αίτηση και αντιστοιχηθεί επιτυχώς με την εντολή OFPFC_MODIFY τότε το πεδίο instructions της καταχώρισης ενημερώνεται και τροποποιείται σύμφωνα με τις παραμέτρους των OFPFC_MODIFY ή OFPFC_MODIFY_STRICT ενώ τα υπόλοιπα πεδία της καταχώρισης, cookie, idle_timeout, hard_timeout, σημαίες, μετρητές και duration παραμένουν αμετάβλητα. Σε περίπτωση που δεν προϋπάρχει κάποια καταχώριση στο flow table

με τις παραμέτρους της OFPFC_MODIFY τότε η αίτηση δρα σαν OFPFC_ADD και η νέα καταχώριση εγγράφεται στο flow table.

Για αιτήσεις διαγραφής τύπου OFPFC_DELETE ή OFPFC_DELETE_STRICT αν υπάρχει η αντίστοιχη καταχώριση στο flow table τότε αυτή διαγράφεται και ο μεταγωγέας παράγει ένα μήνυμα επιτυχούς διαγραφής (flow removed). Αν δεν υπάρχει αντιστοίχιση σε καμιά καταχώριση τότε δεν επιτελείται καμιά ενέργεια και ούτε παράγεται μήνυμα λάθους.

Οι αιτήσεις τύπου OFPFC_MODIFY και OFPFC_DELETE έχουν δύο μορφές, strict (OFPFC_MODIFY_STRICT, OFPFC_DELETE_STRICT) και non-strict. Στις non-strict μπορεί να γίνει και μερική αντιστοίχιση χωρίς να συμπεριλαμβάνονται αναγκαστικά όλα τα πεδία για την ζητούμενη καταχώριση (χρήση wildcards). Αντίθετα, στις strict είναι απαραίτητο όλα τα πεδία να είναι πλήρως ορισμένα για να επιτευχθεί πλήρης (1-1) αντιστοίχιση αυτών με τη ζητούμενη καταχώριση. Για παράδειγμα αν σταλεί αίτηση OFPFC_DELETE με όλες τις σημαίες ενεργές (wildcards) θα διαγραφούν όλες οι καταχωρίσεις σε όλα τα flow tables ενώ με αίτηση OFPFC_DELETE_STRICT θα διαγραφούν μόνο καταχωρίσεις που αφορούν ένα συγκεκριμένο κανόνα για τα εισερχόμενα πακέτα από καθορισμένη θύρα. Οι αιτήσεις διαγραφής μπορούν, προαιρετικά, να εξειδικευθούν βάση της θύρας εξόδου ή το group προορισμού. Αν το πεδίο out_port στην περιγραφή του flow entry έχει οποιαδήποτε τιμή πέραν της OFP_ANY εισάγεται περιορισμός ως προς την εμβέλεια της αίτησης διαγραφής.

Μηνύματα λάθους, ofp_error_msg, παράγονται στις εξής περιπτώσεις:

- Αν η αίτηση τροποποίησης υποδεικνύει μη έγκυρο flow table ή τιμή 0xFF παράγεται μήνυμα λάθους τύπου OFPET_FLOW_MOD_FAILED και κωδικού OFPFMFC_BAD_TABLE_ID.
- Αν δεν βρεθεί χώρος για εγγραφή νέας καταχώρισης στο flow table παράγεται μήνυμα λάθους τύπου OFPET_FLOW_MOD_FAILED και κωδικού OFPFMFC_TABLE_FULL.
- Αν η οδηγία που ζητείται να εκτελέσει ο μεταγωγέας είναι άγνωστη στην συσκευή παράγεται μήνυμα λάθους τύπου OFPET_BAD_INSTRUCTION και κωδικού OFPBIC_UNKNOWN_INST.
- Αν η οδηγία που ζητείται να εκτελέσει ο μεταγωγέας δεν υποστηρίζεται από την συσκευή παράγεται μήνυμα λάθους τύπου OFPET_BAD_INSTRUCTION και κωδικού OFPBIC_UNSUP_INST.
- Αν η οδηγία περιλαμβάνει εντολή Goto με μη έγκυρη τιμή προορισμού παράγεται μήνυμα λάθους τύπου OFPET_BAD_INSTRUCTION και κωδικού OFPBIC_BAD_TABLE_ID.
- Αν η οδηγία περιλαμβάνει εντολή Write-Metadata και η τιμή των μεταδεδομένων ή η μάσκα μεταδεδομένων δεν υποστηρίζεται από την

συσκευή παράγεται μήνυμα λάθους τύπου OFPET_BAD_INSTRUCTION και κωδικού OFPBIC_UNSUP_METADATA ή OFPBIC_UNSUP_METADATA_ MASK.

- Αν η οδηγία περιλαμβάνει Experimenter εντολή η οποία δεν υποστηρίζεται από την συσκευή παράγεται μήνυμα λάθους τύπου OFPET_BAD_INSTRUCTION και κωδικού OFPBIC_UNSUP_EXP_INST.
- Αν η αντιστοίχιση σε μια καταχώριση κατά την αίτηση τροποποίησης περιέχει πεδία που δεν υποστηρίζονται από το flow table παράγεται μήνυμα λάθους τύπου OFPET_BAD_MATCH και κωδικού OFPBMC_BAD_FIELD.
- Αν η αντιστοίχιση σε μια καταχώριση κατά την αίτηση τροποποίησης ορίζει πεδίο που δεν υποστηρίζεται από το flow table ως wildcard παράγεται μήνυμα λάθους τύπου OFPET_BAD_MATCH και κωδικού OFPBMC_BAD_WILDCARDS.
- Αν η αντιστοίχιση σε μια καταχώριση κατά την αίτηση τροποποίησης ορίζει μια αυθαίρετη μάσκα δικτύου (bitmask) που δεν υποστηρίζεται παράγεται μήνυμα λάθους τύπου OFPET_BAD_MATCH και κωδικού OFPBMC_BAD_DL_ADDR_MASK ή OFPBMC_BAD_NW_ADDR_MASK.
- Αν η αντιστοίχιση σε μια καταχώριση κατά την αίτηση τροποποίησης περιέχει τιμές που δεν μπορούν να αντιστοιχηθούν (πχ ετικέτα VLAN με τιμή μεγαλύτερη του 4095) παράγεται μήνυμα λάθους τύπου OFPET_BAD_MATCH και κωδικού OFPBMC_BAD_VALUE.
- Αν μια εντολή υποδεικνύει σε θύρα που δεν είναι έγκυρη στην συσκευή παράγεται μήνυμα λάθους τύπου OFPET_BAD_ACTION και κωδικού OFPBMC OFPBAC_BAD_OUT_PORT.
- Αν μια εντολή σε αίτηση τροποποίησης υποδεικνύει σε group που δεν ορίζεται στον μεταγωγέα παράγεται μήνυμα λάθους τύπου OFPET_BAD_ACTION και κωδικού OFPBAC_BAD_OUT_GROUP.
- Αν μια εντολή σε αίτηση τροποποίησης περιέχει μη έγκυρη τιμή σε ένα πεδίο ή υπάρχει εντολή push με μη έγκυρο τύπο Ethernet παράγεται μήνυμα λάθους τύπου OFPET_BAD_ACTION και κωδικού OFPBAC_BAD_ARGUMENT.
- Αν μια εντολή σε αίτηση τροποποίησης προσπαθεί να εκτελέσει μια διεργασία που δεν είναι συμβατή με το καταχωρηθέν entry (πχ εκτέλεση ανάκλησης ετικέτας VLAN σε πακέτο που δεν εμπεριέχει VLAN) παράγεται μήνυμα λάθους τύπου OFPET_BAD_ACTION και κωδικού OFPBAC_MATCH_INCONSISTENT.

Σε οποιαδήποτε άλλα σφάλματα που δεν είναι δυνατός ο προσδιορισμός της φύσης τους παράγεται μήνυμα λάθους τύπου OFPET_FLOW_MOD_FAILED και κωδικού OFPFMC_UNKNOWN.

2.10 Μηνύματα τροποποίησης του Group table

Οι αιτήσεις που χρησιμοποιούνται για την τροποποίηση των group entries στα group tables έχουν τους ακόλουθους τύπους:

```
                                        /* Group commands */
enum ofp_group_mod_command {
        OFPGC_ADD,                      /* New group. */
        OFPGC_MODIFY,           /* Modify all matching groups. */
        OFPGC_DELETE,           /* Delete all matching groups. */
};
```

Η δέσμη εντολών για κάθε ομάδα εντολών (action bucket) τροποποιείται σύμφωνα με τους κανόνες περί επικάλυψης, προσθήκης και διαγραφής που περιγράφονται για flow entries ανωτέρω, με κάποιους επιπρόσθετους ελέγχους και εντολές που αφορούν το group table. Αν μια εντολή σε μια ή περισσότερες ομάδες εντολών δεν υποστηρίζεται η συσκευή μπορεί να παράξει ένα μήνυμα λάθους ofp_error_msg, τύπου OFPET_BAD_ACTION.

Κάθε group απαρτίζεται από μηδέν ή περισσότερες ομάδες εντολών (buckets). Ένα group χωρίς ομάδα εντολών δεν επηρεάζει καθόλου το σύνολο εντολών ενός πακέτου που έχει οριστεί πριν κατά την επεξεργασία του από την συσκευή. Μπορούν επίσης να υπάρχουν ομάδες εντολών οι οποίες να προωθούν σε άλλα group entries.

Για αιτήσεις add, OFPGC_ADD, αν μια καταχώριση (group entry) με τον ίδιο αναγνωριστικό αριθμό ήδη υπάρχει στο group table τότε η συσκευή αρνείται την καταχώριση και παράγει μήνυμα λάθους τύπου OFPET_GROUP_MOD_FAILED και κώδικα OFPGMFC_GROUP_EXISTS.

Για αιτήσεις τροποποίησης , OFPGC_MODIFY, αν μια καταχώριση με τον ίδιο αναγνωριστικό αριθμό ήδη υπάρχει πάλι, τότε αυτή διαγράφεται και η νέα τροποποιημένη εντολή με τις νέες ομάδες ενεργειών εγγράφεται στο table. Αν ο αναγνωριστικός αριθμός δεν είναι έγκυρος τότε η συσκευή αρνείται την καταχώριση και παράγει μήνυμα λάθους τύπου OFPET_GROUP_MOD_FAILED και κώδικα OFPGMFC_UNKNOWN_GROUP.

Για αιτήσεις διαγραφής OFPGC_DELETE, αν το αναγνωριστικό είναι έγκυρο διαγράφεται η καταχώριση και μαζί όλες οι καταχωρίσεις ροών (flow entries) που προωθούν σε αυτήν. Σε περίπτωση μη έγκυρου αναγνωριστικού δεν παράγεται κανένα μήνυμα λάθους και δεν γίνεται καμιά τροποποίηση στο group table. Η εντολή delete διαφέρει από τις άλλες (add και modify) στο ότι η χρήση της εντολής

χωρίς να οριστεί ομάδα ενεργειών δεν παράγει μήνυμα λάθους σε μελλοντικές προσπάθειες τροποποίησης της καταχώρισης από άλλη αίτηση με το ίδιο αναγνωριστικό. Επίσης για την διαγραφή όλων των group entries με ένα μήνυμα μπορεί να σταλεί η αίτηση με ενεργοποιημένη την σημαία OFPG_ALL.

Μηνύματα λάθους (επιπρόσθετα των μηνυμάτων για flow entries που αναφέρονται ανωτέρω) είναι:

- Αν ο τύπος group που ορίζεται είναι μη έγκυρος η συσκευή αρνείται να εγγράψει την καταχώριση και παράγεται μήνυμα λάθους τύπου OFPET_GROUP_MOD_FAILED και κώδικα OFPGMFC_INVALID_GROUP.
- Αν η συσκευή δεν υποστηρίζει καταχωρίσεις με διαφορετικό φορτίο και εμβέλεια εντολών τότε αρνείται να εγγράψει την καταχώριση και παράγεται μήνυμα λάθους τύπου OFPET_GROUP_MOD_FAILED και κώδικα OFPGMFC_WEIGHT_UNSUPPORTED.
- Αν η συσκευή δεν μπορεί να εγγράψει την καταχώριση λόγω έλλειψης αποθηκευτικού χώρου τότε παράγεται μήνυμα λάθους τύπου OFPET_GROUP_MOD_FAILED και κώδικα OFPGMFC_OUT_OF_GROUPS.
- Αν η συσκευή δεν μπορεί να εγγράψει την καταχώριση λόγω άλλων περιορισμών τότε παράγεται μήνυμα λάθους τύπου OFPET_GROUP_MOD_FAILED και κώδικα OFPGMFC_OUT_OF_BUCKETS.
- Αν η συσκευή δεν μπορεί να εγγράψει την καταχώριση επειδή δεν μπορεί να υποστηρίξει την εν ενεργεία διάρκειά της (liveliness) τότε παράγεται μήνυμα λάθους τύπου OFPET_GROUP_MOD_FAILED και κώδικα OFPGMFC_WATCH_UNSUPPORTED.

2.11 State of the art υλοποιήσεις του OpenFlow – Το FlowVisor

Το OpenFlow χρησιμοποιείται σε αρκετές εφαρμογές που αποσκοπούν στην χρήση του ως πρωτόκολλο ταυτόχρονα με την παρούσα υποδομή του δικτύου και του τρέχων πρωτοκόλλου (συνήθως IP) σε λειτουργία. Αυτό επιτυγχάνεται με την χρήση εικονικών μηχανών (virtual machines) στις οποίες τρέχει η εφαρμογή OpenFlow.

Μια τέτοια υλοποίηση μέσω εικονικών μηχανών είναι το FlowVisor, ένα εργαλείο το οποίο αναπτύχθηκε από το πανεπιστήμιο του Stanford. Το FlowVisor αποσκοπεί στον διαμοιρασμό των πόρων του δικτύου για την δημιουργία και αξιοποίηση νέων τεχνολογιών, τεχνικών προώθησης και πρωτοκόλλων, στο ήδη υπάρχον δίκτυο. Είναι ένα εργαλείο network virtualization, ¨εικονικοποίησης δικτύου¨, το οποίο χρησιμοποιεί το OpenFlow για την προώθηση και δρομολόγηση των πακέτων από και προς OpenFlow ελεγκτών μέσω εικονικής μηχανής.

Άλλοι ελεγκτές που αναπτύχθηκαν βάση του OpenFlow είναι οι Beacon και Floodlight. Οι Beacon και Floodlight δεν είναι εργαλεία με κύριο προσανατολισμό τον διαμοιρασμό πόρων δικτύων όπως το FlowVisor, αλλά είναι ελεγκτές με στόχο την πιο γρήγορη ανάπτυξη και ενεργοποίηση εφαρμογών που αφορούν το δίκτυο σε λειτουργία και διαθέτουν αρχιτεκτονική προσανατολισμένη σε αυτό.

2.11 .1 Σχεδιαστικοί Στόχοι του FlowVisor

Το FlowVisor σχεδιάστηκε έχοντας υπόψη τρείς βασικούς σχεδιαστικούς στόχους:

1. Διαφάνεια – Transparency. Το εικονικό στρώμα όπως προκύπτει από την εικονική μηχανή είναι διαφανές και στις συσκευές του αρχικού δικτύου και στους ελεγκτές που διαχειρίζονται από το FlowVisor. Αυτό δίνει την δυνατότητα για ανάπτυξη και έλεγχο καινούριων πρωτοκόλλων μέσω των εικονικών μηχανών σε ένα πραγματικό περιβάλλον δικτύου. Επίσης με την διαφάνεια του στρώματος επιτυγχάνεται η αποσύζευξη της διαμόρφωσης και τεχνολογίας του εικονικού δικτύου από την σχεδίαση του υλικού των ελεγκτών (controllers) του δικτύου και έτσι το εικονικό δίκτυο μπορεί να ενημερώνεται και να εξελίσσεται ανεξάρτητα από την σχεδίαση και κατασκευή του υλικού των συσκευών.
2. Απομόνωση – Isolation. Το στρώμα δικτύου που δημιουργείται από την εικονική μηχανή απομονώνει πλήρως τα επιμέρους δίκτυα OpenFlow (network slices) που αναπτύσσονται ούτως ώστε ένα δίκτυο να μη μπορεί να

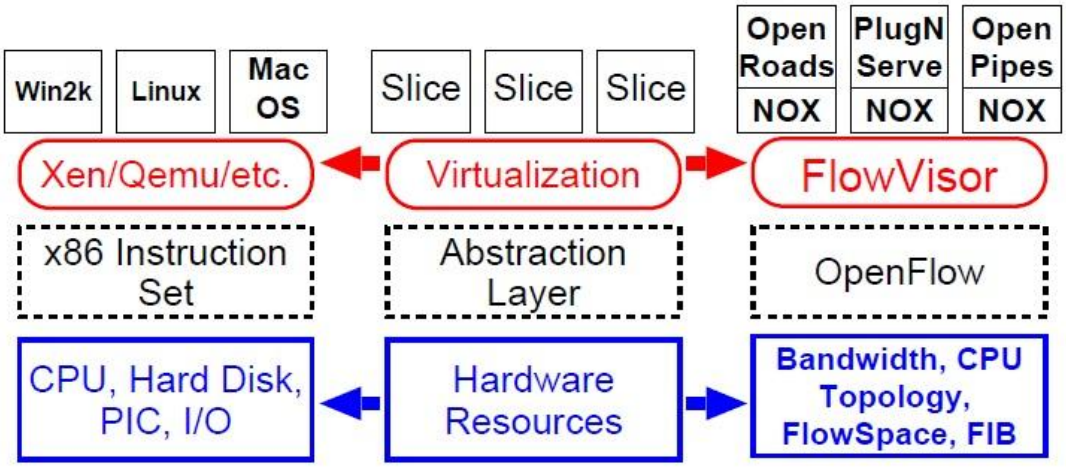
χρησιμοποιήσει πόρους άλλου υποδικτύου σε περίπτωση που εξαντλήσει τους δικούς του.

3. Επεκτάσιμη Δυνατότητα ορισμού δικτύων - Extensible Slice Definition. Λόγω του ότι τα εικονικά δίκτυα πρέπει να έχουν την δυνατότητα τροποποίησης ή προσθήκης ή αφαίρεσης δικτύων, η πολιτική ανάθεσης και δημιουργίας τους από την εικονική μηχανή είναι ευέλικτη, εύκολα επεκτάσιμη και τμηματική.

2.11 .2 Αρχιτεκτονική του FlowVisor

Το FlowVisor τοποθετείται μεταξύ του υποκείμενου υλικού της συσκευής δικτύου στην οποία εγκαθίσταται και του λογισμικού το οποίο ελέγχει, ακριβώς όπως η εικονική μηχανή σε ένα υπολογιστή. Με την χρήση του πρωτοκόλλου OpenFlow το FlowVisor ελέγχει αυτό το υλικό για την δημιουργία του εικονικού δικτύου. Το FlowVisor υποστηρίζει πολλαπλά δίκτυα OpenFlow, δεχόμενο ξεχωριστό ελεγκτή γα το κάθε δίκτυο. Αυτά ορίζονται ως τεμάχια δικτύου (network slices) και είναι πλήρως απομονωμένα.

Η φιλοσοφία του FlowVisor ως εργαλείο παρουσιάζεται στο πιο κάτω Σχήμα 2.11.2-1 που δείχνει την δημιουργία του αφαιρετικού στρώματος δικτύου κατά αντιστοιχία με τις εικονικές μηχανές που αναπτύσσονται σε υπολογιστές:

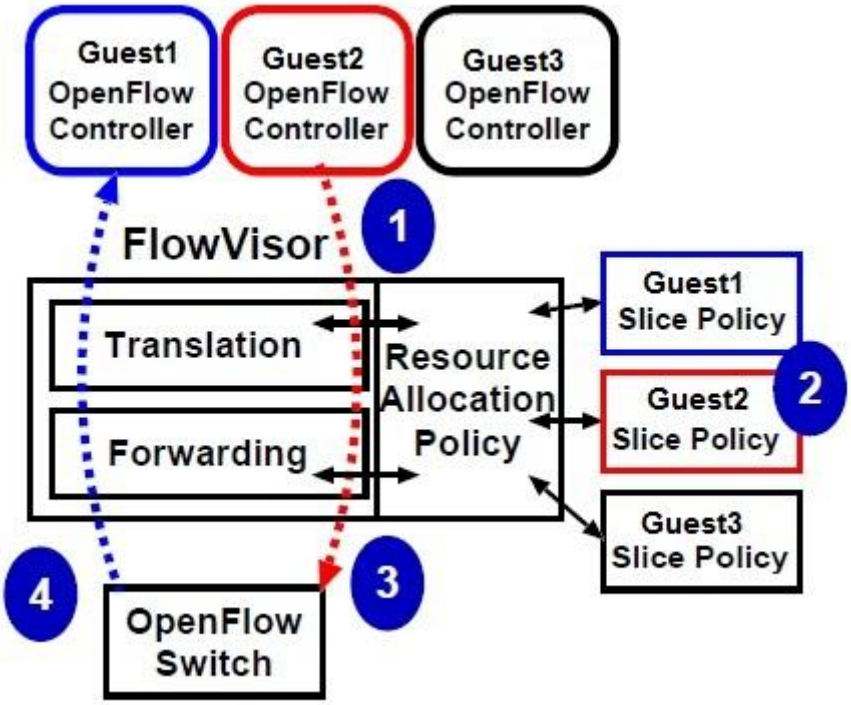


Σχήμα 2.11.2-1 Αρχιτεκτονική του FlowVisor

Το FlowVisor θεωρεί κάθε slice ως σύνολο από flow entries οι οποίες συνολικά δημιουργούν ένα ενιαίο πεδίο κίνησης (traffic) που ονομάζεται flowspace. Δοθέντος μιας επικεφαλίδας ενός πακέτου το FlowVisor μπορεί να αποφασίσει σε ποια slices (δίκτυα) ανήκει και δημιουργεί τα ανάλογα flowspaces για κάθε δίκτυο. Είναι δυνατό ένα πακέτο να ανήκει σε περισσότερα του ενός slices, με μόνο

περιορισμό τα flowspaces για κάθε slice να μην επικαλύπτονται σε κανένα σημείο της τοπολογίας.

Πιο παραστατικά το FlowVisor λειτουργεί σαν ενδιάμεσος (proxy) μεταξύ OpenFlow συσκευών δικτύου και πολλαπλών φιλοξενούμενων (guest) ελεγκτών. Όλα τα μηνύματα OpenFlow περνούνε από το FlowVisor, από και προς τους guest ελεγκτές και μέσω μηνυμάτων OpenFlow το FlowVisor επικοινωνεί και με τους guests και με τις υπόλοιπες συσκευές του δικτύου. Οι guest ελεγκτές θεωρούν ότι επικοινωνούν κατευθείαν με τις συσκευές δικτύου χωρίς να ¨βλέπουν¨ το FlowVisor, όπως φαίνεται στο ακόλουθο σχήμα 2.11.2-1:



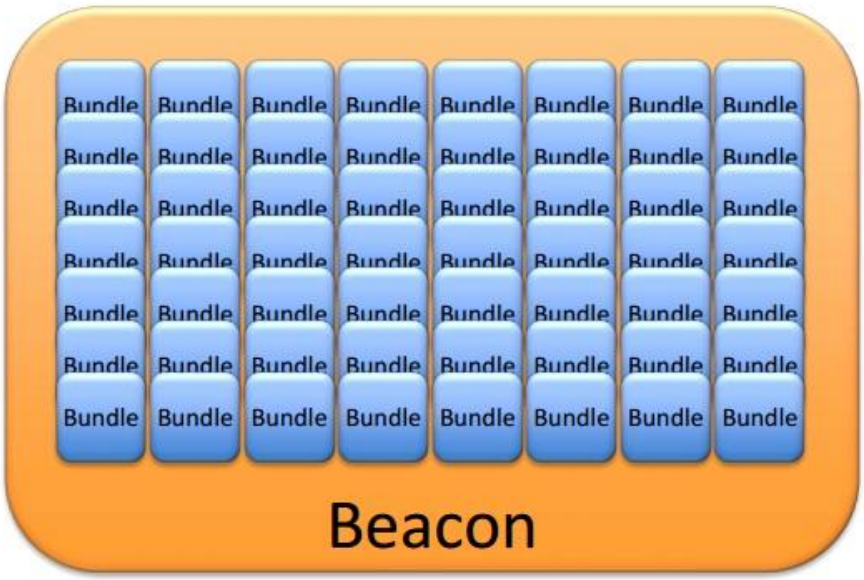
Σχήμα 2.11.2-2 FlowVisor controller με guest controllers

Εδώ βλέπουμε ένα FlowVisor με τρεις guest ελεγκτές (σημείο 1) οι οποίοι επικοινωνούν με το OpenFlow switch μέσω του FlowVisor. Ο κάθε guest έχει το δικό του slice δίκτυο όπως του ανατίθεται και στο FlowVisor διατηρούνται αρχεία πολιτικών (policies) για κάθε guest (σημείο 2) και βάση αυτών δημιουργούνται τα flowspaces. Αν ένας από τους guests επιχειρήσει να αναθέσει ένα flow entry στο switch το FlowVisor ελέγχει σύμφωνα με την πολιτική του guest αν τέτοια ανάθεση είναι επιτρεπτή και έγκυρη, οπότε προωθεί το flow entry στο flow table του μεταγωγέα (σημείο 3). Επίσης οποιαδήποτε κίνηση από τον μεταγωγέα προς τους guest ελεγκτές ελέγχεται από το FlowVisor (σημείο 4) και ανατίθεται στο ανάλογο flowspace.

2.11 .3 Ο ελεγκτής Beacon

Ο Beacon είναι ένας ελεγκτής OpenFlow που αναπτύχθηκε σε περιβάλλον Java και διατέθηκε για χρήση το 2011. Μπορεί να τρέξει σε αρκετές πλατφόρμες (Windows, Linux, Android OS) και υποστηρίζει πολυνηματική (multithreaded) λειτουργία. Βασίζεται σε τμηματική λογική (modular) οπότε είναι εύκολα επεκτάσιμος με χρήση νέων τμημάτων (modules) για υποστήριξη επιπροσθέτων λειτουργιών.

Με την χρήση δεσμών κώδικα (code bundles) ο Beacon μπορεί να εκτελεί με δυναμικό τρόπο εφαρμογές που δεν είναι αλληλοεξαρτώμενες ξεκινώντας, σταματώντας ή θέτοντας σε παύση δέσμες κώδικα σύμφωνα με τις επιλογές του χρήστη, χωρίς να είναι αναγκαία η επανεκκίνηση του ελεγκτή για κάθε αλλαγή. Η αρχιτεκτονική αυτή του Beacon παρουσιάζεται στο πιο κάτω σχήμα 2.11.3-1.



Σχήμα 2.11.3-1 Beacon controller με δέσμες κώδικα (Bundles)

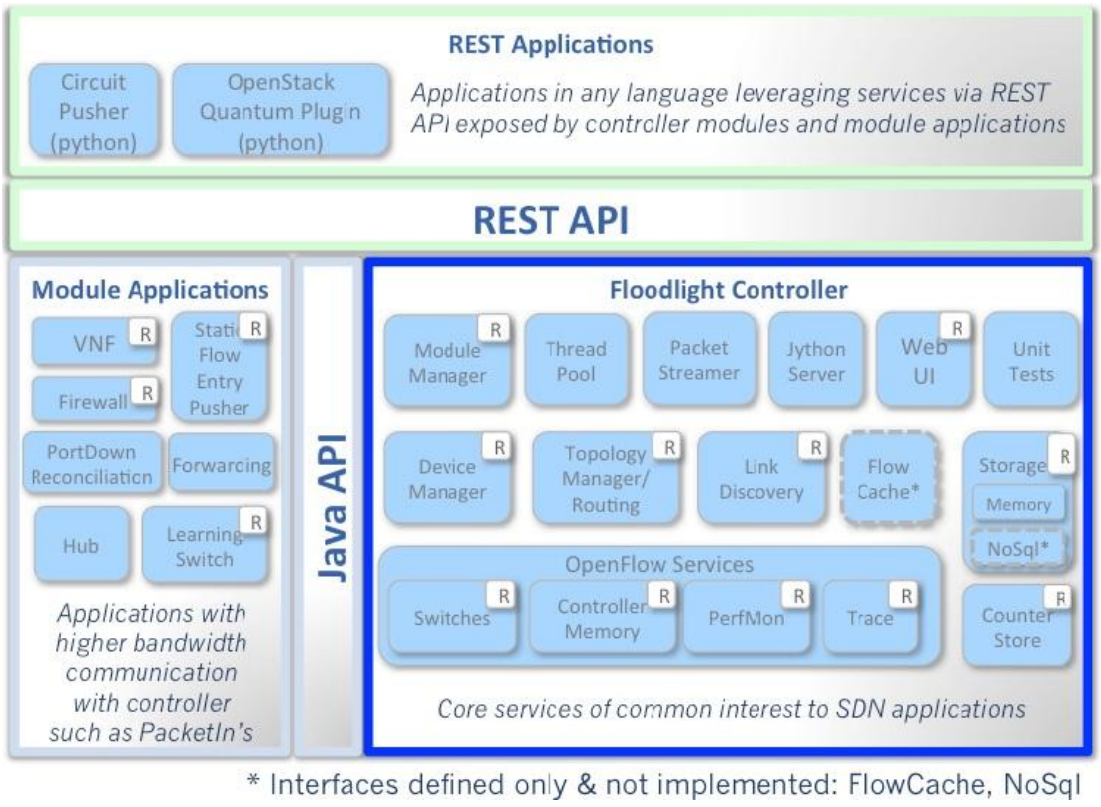
Οι δέσμες κώδικα ανωτέρω μπορούν να δουλεύουν ταυτόχρονα, να διαμοιράζονται τα πακέτα κώδικα και εντολών τους (Java packages) με άλλες δέσμες, να επεκτείνουν το ρεπερτόριο λειτουργιών τους με άλλο κώδικα (extend) ή και να έχουν πολλαπλές εκδόσεις σε λειτουργία ταυτόχρονα (versioning).

Ο Beacon παρέχεται με όλες τις βασικές δέσμες κώδικα, όπως είναι η OpenFlowJ (OF 1.0 Protocol) για το πρωτόκολλο OpenFlow, οι δέσμες για κωδικοποίηση και αποκωδικοποίηση πακέτων (Ethernet, ARP, IPv4, LLDP, TCP, UDP), δέσμες για βασικές λειτουργίες μεταγωγέα (Core, Learning Switch, Hub,

Device Manager) καθώς και δέσμες αλγορίθμων και εύρεσης τοπολογίας (Topology, Layer 2 Shortest Path Routing).

2.11 .4 Ο ελεγκτής Floodlight

Ο Floodlight ελεγκτής είναι και αυτός βασισμένος στην Java όπως ο Beacon, αλλά υιοθετεί μια διαφορετική αρχιτεκτονική και τρόπο λειτουργίας. Ο ελεγκτής αποτελείται από μια αυτόνομη συλλογή λειτουργικών μονάδων που επιτελούν τις κύριες λειτουργίες του Floodlight ως ελεγκτής OpenFlow καθώς και εφαρμογές που αναπτύσσονται με τρόπο ώστε να υπερτίθενται (on-top) της βασικής μονάδας του ελεγκτή (REST applications) ή να λειτουργούν μαζί με τον ελεγκτή (Module applications) , όπως παρουσιάζεται στο ακόλουθο σχήμα:



Σχήμα 2.11.4-1 Floodlight controller και REST applications

Όπως παρουσιάζεται πιο πάνω, ο Floodlight αποτελείται από τις λειτουργικές μονάδες τοπολογίας και ζεύξης (Topology Manager, Link Discovery), ελέγχου συσκευών και μονάδων (Device Manager, Module Manager), υπηρεσιών OpenFlow (OpenFlow Services), αποθήκευσης και χειρισμού μονάδας (Storage, Counter Store, Flow Cache, Packet Streamer, Thread Pool).

Οι εφαρμογές που χρησιμοποιούν τον Floodlight ως υποκείμενο στρώμα αναπτύσσονται ως ανεξάρτητες μονάδες Java και χρησιμοποιούν την

προγραμματιστική διεπαφή REST του ελεγκτή (REST API – application programming interface). Μέσω του REST API οι εφαρμογές που αναπτύσσονται επικοινωνούν με τον Floodlight ελεγκτή και μπορούν να χρησιμοποιήσουν το δίκτυο για την όποια λειτουργία τους.

Επίσης οι εφαρμογές μπορούν να αναπτυχθούν και να ρυθμιστούν στο επίπεδο του Floodlight ελεγκτή, φτάνει να αναπτυχθούν και να εφαρμοστούν στον ελεγκτή ως υπομονάδες Java (Java modules). Αυτός ο τρόπος, παρότι πιο απαιτητικός και δύσκολος από ότι η χρήση των REST API's, έχει σημαντικά οφέλη όσο αφορά την ταχύτητα εκτέλεσης της εφαρμογής καθώς και προσφορά μεγαλύτερου εύρους ζώνης επικοινωνίας με τον Floodlight, μιας και η σύζευξη εφαρμογής – ελεγκτή γίνεται πιο άμεσα με την χρήση των Java API's.

ΚΕΦΑΛΑΙΟ 3

3.1 Ο ελεγκτής NOX – NOX controller

Ο NOX ελεγκτής είναι μια πλατφόρμα λογισμικού η οποία προσφέρει την δυνατότητα κατασκευής, διαχείρισης και ελέγχου υπολογιστικών δικτύων με χρήση του πρωτοκόλλου OpenFlow. Ο NOX μάλιστα ήταν ο πρώτος ελεγκτής που αναπτύχθηκε αποκλειστικά για το πρωτόκολλο OpenFlow και συνεπώς είναι από τους πιο διαδεδομένους ελεγκτές για συσκευές OpenFlow.

Ο NOX αρχικά αναπτύχθηκε από την εταιρεία Nicira Networks, ταυτόχρονα με την ανάπτυξη του πρωτοκόλλου OpenFlow . Δόθηκε στην ερευνητική κοινότητα το 2008 και από τότε συνεχίστηκε η εξέλιξη του και υπήρξε η βάση για διάφορα ερευνητικά έργα που αφορούν το SDN (software defined networking), έννοια η οποία ξεκίνησε από ακαδημαϊκές εργασίες για δημιουργία αρχιτεκτονικών για εταιρικά δίκτυα όπως είναι το SANE (Secure Architecture for the Networked Enterprise) και το Ethane.

Γενικά ο ελεγκτής NOX προσφέρει μια πλήρη προγραμματιστική διεπαφή (API – application programming interface) του OpenFlow 1.0 με χρήση των γλωσσών C++ καθώς και Python, χρήση ασύγχρονων εισόδων – εξόδων (I/O) και είναι προσανατολισμένος για λειτουργία σε συστήματα Linux, συγκεκριμένα για τις εκδόσεις Ubuntu 11.10 και 12.04, καθώς και για Debian.

Για τον έλεγχο του δικτύου στο οποίο αναπτύσσεται, ο NOX χρησιμοποιείται ως πλατφόρμα στην οποία αναπτύσσονται και τρέχουν εφαρμογές οι οποίες εξειδικεύονται για να επιτελούν τις επιθυμητές ενέργειες στο δίκτυο από τον χρήστη. Οι εφαρμογές αυτές χρησιμοποιούν το OpenFlow API του NOX και έτοιμα υποπρογράμματα (components) από τον ελεγκτή για άμεση χρήση σε συνήθεις λειτουργίες όπως η δρομολόγηση των πακέτων (routing component) ή η ανίχνευση τοπολογίας του δικτύου (topology component).

3.2 Ιστορικό Ανάπτυξης του NOX – Η έννοια του Λειτουργικού Συστήματος Δικτύων

Όπως αναφέρθηκε ανωτέρω, ο NOX είναι ο κυριότερος εκπρόσωπος του SDN, όπως προέκυψε από ακαδημαϊκές εργασίες όπως το SANE και το Ethane. Αυτές οι εργασίες προέκυψαν ως αποτέλεσμα του προβλήματος διαχείρισης μεγάλων εταιρικών δικτύων και το SANE, όπως και το Ethane, είναι προσπάθειες εξεύρεσης λύσεων για πιο αποτελεσματική διαχείριση και παραμετροποίηση μεγάλης κλίμακας δικτύων υπολογιστών όπως είναι τα εταιρικά.

Σε αυτά τα εκτεταμένα δίκτυα παρατηρείται συστηματικά το πρόβλημα της αποτελεσματικής διαχείρισης τους, λόγω του μεγέθους και της πολυπλοκότητας τους. Αυτό το πρόβλημα μπορεί να παραλληλιστεί με τα θέματα που αντιμετώπιζαν οι ηλεκτρονικοί υπολογιστές σε προγράμματα και λειτουργικά συστήματα στην απαρχή της εξέλιξης της πληροφορικής, όταν τα προγράμματα γράφονταν σε γλώσσα μηχανής (assembly language) και δεν υπήρχε κανένα αφαιρετικό επίπεδο μεταξύ αυτών και των φυσικών πόρων των ηλεκτρονικών συστημάτων.

Έτσι δημιουργείτο το ίδιο πρόβλημα αποτελεσματικής διαχείρισης των διαφόρων προγραμμάτων λόγω των δυσκολιών που προέκυπταν στο γράψιμο, στη μεταφορά σε άλλα συστήματα (port), στην απασφαλμάτωση (debug) και στην εύρεση συμβατότητας των προγραμμάτων με άλλες πλατφόρμες. Αυτό επιλύθηκε με την εισαγωγή προγραμματιστικών διεπαφών υψηλού επιπέδου (high level API's), οι οποίες αναλαμβάνουν την πρόσβαση στους πόρους (μνήμη, αποθήκευση, επεξεργαστής, επικοινωνία) και στις πληροφορίες (αρχεία, καταλόγους αρχείων) του συστήματος. Με αυτό τον τρόπο τα προγράμματα μπορούν να επιτελούν σύνθετες εργασίες με ασφάλεια και αποδοτικότητα και να μπορούν να δουλεύουν σε μια πληθώρα υπολογιστικών υλικών (computing hardware). Το σύνολο αυτών των προγραμματιστικών διεπαφών είναι το λειτουργικό σύστημα.

Κατά αντιστοιχία, τα δίκτυα υπολογιστών διαχειρίζονται μέσω παραμετροποίησης χαμηλού επιπέδου των διαφόρων και ανεξαρτήτων συσκευών δικτύου που τα απαρτίζουν. Πολύ συχνά κιόλας η παραμετροποίηση των συσκευών εξαρτάται άμεσα και από το πρωτόκολλο δικτύου που χρησιμοποιείται (IP, ARP, ICMP) ή ακόμα και από την τοπολογία του δικτύου. Έτσι αυτά τα δίκτυα μοιάζουν με υπολογιστή χωρίς λειτουργικό σύστημα, με την εξαρτώμενη από τον χρήστη παραμετροποίηση των συσκευών του δικτύου να έχει το ρόλο του εξαρτώμενου από το υλικό προγραμματισμού σε γλώσσα μηχανής.

Άρα είναι φανερό ότι χρειαζόταν ένα λειτουργικό σύστημα για δίκτυα, το οποίο να προσφέρει μια κοινή και κεντροποιημένη προγραμματική διεπαφή για

ολόκληρο το δίκτυο. Γενικά αυτό το λειτουργικό σύστημα δικτύου πρέπει να παρέχει την δυνατότητα παρατήρησης και ελέγχου του δικτύου με τρόπο ανάλογο του λειτουργικού συστήματος σε ένα υπολογιστή.

Εντούτοις, ένα λειτουργικό σύστημα δικτύου δεν διαχειρίζεται το δίκτυο αυτό καθ' αυτό, παρά μόνο προσφέρει ένα προγραμματιστικό περιβάλλον για τον σκοπό αυτό. Οι εργασίες διαχείρισης και ελέγχου του δικτύου επιτελούνται από εφαρμογές που αναπτύσσονται και εφαρμόζονται επί του λειτουργικού συστήματος δικτύου (on top), οπότε και είναι αναγκαία μια προγραμματιστική διεπαφή γενικού σκοπού, η οποία να μπορεί να υποστηρίξει ένα όσο το δυνατό πιο ευρύ φάσμα εφαρμογών διαχείρισης και παρακολούθησης του δικτύου.

Η ανάπτυξη ενός τέτοιου λειτουργικού συστήματος δικτύου σηματοδοτεί δύο πολύ σημαντικές διαφοροποιήσεις από την μέχρι τώρα ανάπτυξη δικτύων υπολογιστών. Καταρχάς εισάγεται η έννοια ενός κεντρικού προγραμματιστικού μοντέλου, βάση του οποίου οι εφαρμογές για το δίκτυο γράφονται θεωρώντας ότι ολόκληρο το δίκτυο είναι παρών και σε λειτουργία. Κατά δεύτερο, οι εφαρμογές γράφονται σε λογική υψηλού αφαιρετικού επιπέδου, με χρήση για παράδειγμα ονομάτων χρήστη και εξυπηρετητή αντί παραμέτρων χαμηλού επιπέδου όπως είναι οι διευθύνσεις IP και MAC.

Με αυτές τις συμβάσεις είναι δυνατή η εφαρμογή και εκτέλεση οδηγιών διαχείρισης από τις εφαρμογές προς το δίκτυο ανεξαρτήτως των υποκείμενων συσκευών δικτύου ή της τοπολογίας και πρωτοκόλλου διαχείρισης του δικτύου. Η ανάγκη για παρακολούθηση της κατάστασης του δικτύου, της τοπολογίας, των ζεύξεων και των συσχετίσεων μεταξύ των συσκευών του δικτύου μετατίθεται και αναλαμβάνεται αποκλειστικά από το λειτουργικό σύστημα δικτύου, που συν τοις άλλοις διατηρεί και μεταβάλλει ανάλογα τους συσχετισμούς μεταξύ των αφαιρετικών επιπέδων προγραμματισμού και των παραμέτρων χαμηλού επιπέδου των συσκευών.

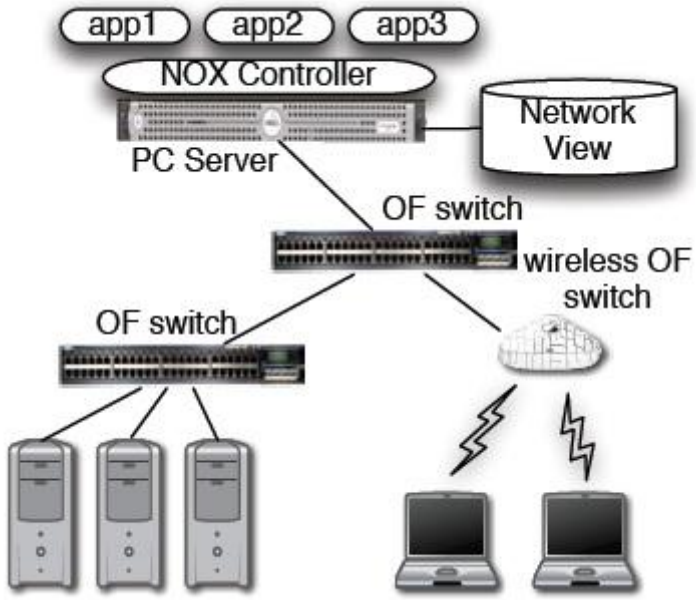
Συνοψίζοντας, ένα λειτουργικό σύστημα δικτύου επιτρέπει την ανάπτυξη εφαρμογών διαχείρισης και ελέγχου δικτύου σε μια κεντροποιημένη μορφή με χρήση αφαιρετικής λογικής υψηλού επιπέδου εν αντιθέσει με την χρήση διανεμημένων αλγορίθμων με πρωτόκολλα χαμηλού επιπέδου στις συσκευές του δικτύου για την επιτέλεση μιας λειτουργίας. Φυσικά η ανάπτυξη ενός τέτοιου λειτουργικού συστήματος δικτύου εμπεριέχει σημαντικές τεχνολογικές προκλήσεις και με την ανάπτυξη του πρωτοκόλλου OpenFlow δόθηκε η δυνατότητα για την δημιουργία μιας πλήρης πλατφόρμας ελέγχου δικτύου, ο ελεγκτής NOX, που διαθέτει τις δυνατότητες και την δομή ενός πλήρους λειτουργικού συστήματος δικτύου.

3.3 Επισκόπηση του NOX

Ο NOX ελεγκτής αναπτύχθηκε με βάση την φιλοσοφία και λειτουργικότητα ενός ενιαίου λειτουργικού συστήματος δικτύων υπολογιστών, χρησιμοποιώντας ως βάση για την ανάπτυξη του το πρωτόκολλο OpenFlow. Τα συστατικά μέρη ενός NOX δικτύου, οι βασικές λειτουργίες και χαρακτηριστικά του είναι ως ακολούθως:

3.3.1 Τα συστατικά μέρη του NOX (components)

Ένα δίκτυο βασισμένο στην χρήση του ελεγκτή NOX απαρτίζεται από ορισμένα συστατικά μέρη τα οποία διαχειρίζεται ή εγκαθίσταται ο NOX. Παραστατικά τα μέρη αυτά παρουσιάζονται στο σχήμα 3.3-1:



Σχήμα 3.3.1-1 Τα συστατικά μέρη ενός δικτύου NOX

Όπως παρουσιάζεται και ανωτέρω ένα δίκτυο NOX αποτελείται από τον εξυπηρετητή στον οποίο εγκαθίσταται και τρέχει ο NOX (PC Server στο σχήμα), τους OpenFlow μεταγωγείς (OF switches, ενσύρματους ή ασύρματους) και τα τερματικά του δικτύου, τους σταθερούς και φορητούς υπολογιστές στο σχήμα.

Στον εξυπηρετητή τρέχει το λογισμικό του NOX και σε αυτό εισάγονται οι διάφορες εφαρμογές διαχείρισης του δικτύου (app1,app2,app3) που όπως προαναφέρθηκε λειτουργούν υπερκείμενα του ελεγκτή (on top). Το λογισμικό του NOX αυτό καθ' αυτό μπορεί να θεωρηθεί ως μια σειρά διεργασιών (processes) για

την διαχείριση των OpenFlow μεταγωγέων και μιας βάσης δεδομένων για την ολική κατάσταση του δικτύου (Network View στο σχήμα).

Η βάση δεδομένων για την ολική κατάσταση του δικτύου περιέχει τα αποτελέσματα από τις παρατηρήσεις του NOX μέσω των διεργασιών του για την κατάσταση, τοπολογία και γενικά χαρακτηριστικά του δικτύου για δεδομένη χρονική στιγμή. Με χρήση αυτής της βάσης δεδομένων οι εφαρμογές του NOX μπορούν να λαμβάνουν αποφάσεις και να επιτελούν ενέργειες για την διαχείριση και παρακολούθηση του δικτύου.

Ο έλεγχος της κίνησης του δικτύου από τον NOX γίνεται με την έκδοση ενεργειών OpenFlow που αποστέλλονται στους μεταγωγείς, οι οποίοι και έχουν εγγενή υποστήριξη του πρωτοκόλλου OpenFlow.

3.3.2 Διακριτότητα στον NOX

Ένα σημαντικό σχεδιαστικό χαρακτηριστικό του NOX είναι η διακριτότητα που παρέχει σε θέματα παρατήρησης και ελέγχου των λειτουργιών του. Με τον όρο διακριτότητα εννοούμε την δημιουργία διακριτών τμημάτων στον NOX, τα οποία μπορούν να παρατηρηθούν και να διαχειριστούν ξεχωριστά από άλλα τμήματα του ελεγκτή καθώς και την διακριτή ποσότητα των πληροφοριών που μπορεί να δοθεί αναφορικά με την κατάσταση και τις παραμέτρους του δικτύου.

Με την αύξηση των διακριτών τμημάτων κώδικα του NOX είναι φανερό ότι αυξάνεται η ευελιξία (flexibility) στην διαχείριση του ελεγκτή με την προσφορά περισσότερων δυνατοτήτων παραμετροποίησης και ελέγχου, με ανάλογο κόστος όμως στην δυνατότητα επεκτασιμότητας (scalability) του NOX, η οποία επιβαρύνεται με την προσθήκη πρόσθετων διακριτών μερών στον ελεγκτή. Από την στιγμή που και τα δύο χαρακτηριστικά είναι σημαντικά για την διαχείριση και λειτουργία μεγάλης κλίμακας δικτύων γίνεται μια αντιστάθμιση (trade-off) μεταξύ βάθους διακριτότητας και πρόνοιας για επεκτασιμότητα του δικτύου.

Για θέματα διακριτότητας παρακολούθησης του δικτύου, η βάση δεδομένων για την κατάσταση δικτύου (Network View) περιλαμβάνει πληροφορίες που αφορούν την τοπολογία των μεταγωγέων, τις διευθύνσεις και χώρους των χρηστών, ξενιστών και ενδιάμεσων κόμβων καθώς και πληροφορίες για τις υπηρεσίες που παρέχονται από το δίκτυο (τύπου HTTP ή NFS για παράδειγμα). Η βάση δεδομένων περιλαμβάνει και όλους τους συσχετισμούς μεταξύ ονομάτων και διευθύνσεων, χωρίς όμως να περιλαμβάνει την τρέχουσα κατάσταση της κίνησης του δικτύου. Με αυτό τον τρόπο η πληροφορία που παρέχεται από τον ελεγκτή είναι αρκετή για πληθώρα εργασιών διαχείρισης του δικτύου και μεταβάλλεται αρκετά αργά ούτως

ώστε να μπορεί να επεκταθεί σε περίπτωση αύξησης του δικτύου (προσθήκη μεταγωγέων, τερματικών) σε μεγαλύτερη κλίμακα.

Όσο αφορά την διακριτότητα ελέγχου του NOX, τα πράγματα είναι κάπως πιο πολύπλοκα. Μια κεντρική διεπαφή ελέγχου για κάθε πακέτο στο δίκτυο θα ήταν εξαιρετικά δύσκολο να αναπτυχθεί και να εφαρμοστεί σε οποιοδήποτε δίκτυο μεγάλης κλίμακας, μιας και η διαχείριση κάθε πακέτου ξεχωριστά θα ήταν πολύ χρονοβόρα, πολύπλοκη και πρακτικά μη αποδοτική. Από την άλλη, θέτοντας ως βάθος διακριτότητας την δρομολόγηση που να βασίζεται σε πίνακες με βάση το πρόθεμα του πακέτου (prefix-based) θα απέκλειε τον αποτελεσματικό έλεγχο της κίνησης του δικτύου, αφού όλα τα πακέτα που θα ανταλλάζονταν μεταξύ δύο κόμβων θα έπρεπε να ακολουθούν πάντα το ίδιο μονοπάτι στο δίκτυο. Έτσι για τον NOX επιλέχθηκε ένα ενδιάμεσο επίπεδο διακριτότητας με την χρήση των ροών (flows) του πρωτοκόλλου OpenFlow. Με αυτό τον τρόπο και σύμφωνα με την περιγραφή του OpenFlow στο κεφάλαιο 2 είναι δυνατή η εφαρμογή πολιτικών δρομολόγησης σε πακέτα κατευθυνόμενα σε κοινό προορισμό με την εγκαθίδρυση και παραμετροποίηση καταλλήλων ροών. Επίσης με την χρήση των ροών είναι δυνατή η διατήρηση ενός αποδεκτού επιπέδου επεκτασιμότητας του δικτύου σε περίπτωση προσθήκης νέων μονάδων, διατηρώντας ταυτόχρονα την ευελιξία του δικτύου σε ζητήματα διαχείρισης και δρομολόγησης.

3.3.3 Οι μεταγωγείς στον NOX με χρήση του OpenFlow

Οι εφαρμογές διαχείρισης που τρέχουν στον ελεγκτή NOX ελέγχουν και ρυθμίζουν την κίνηση του δικτύου στέλνοντας οδηγίες και εντολές στους μεταγωγείς. Αυτές οι οδηγίες πρέπει να είναι ανεξάρτητες από το υλικό κάθε μεταγωγέα και πρέπει να υποστηρίζουν διακριτότητα στο επίπεδο ροών (flows) όπως περιγράφηκε ανωτέρω.

Για την ικανοποίηση των παραπάνω προϋποθέσεων αποφασίστηκε από την αρχή η χρήση του πρωτοκόλλου OpenFlow για την δημιουργία ενός αφαιρετικού επιπέδου στους μεταγωγείς που να αυξάνει την ευελιξία και την επεκτασιμότητα του δικτύου. Σύμφωνα λοιπόν και με την περιγραφή της δομής του OpenFlow στο 2^{ο} κεφάλαιο, οι μεταγωγείς στον NOX αντιπροσωπεύονται με πίνακες ροής (flow tables) με καταχωρήσεις της μορφής:

< header: counters, actions>

Για κάθε πακέτο που αντιστοιχίζεται με την ανάλογη επικεφαλίδα (header), οι μετρητές ενημερώνονται και επιτελούνται οι κατάλληλες ενέργειες. Αν ένα πακέτο αντιστοιχιστεί με περισσότερες από μια ροές τότε επιλέγεται η ροή με την μεγαλύτερη προτεραιότητα.

Για καλύτερη κατανόηση του ρόλου των NOX και OpenFlow μπορούμε να παρατηρήσουμε τα αφαιρετικά επίπεδα που προκύπτουν, όπου ο NOX προσφέρει μια κοινή διεπαφή σε όλο το δίκτυο για χρήση από διάφορες εφαρμογές, ανάλογα με ένα λειτουργικό σύστημα σε ένα υπολογιστή. Η χρήση του πρωτοκόλλου OpenFlow δημιουργεί ένα αφαιρετικό επίπεδο για την χρήση και λειτουργίας μιας συσκευής δικτύου (όπως οι μεταγωγείς), ανάλογα με τα προγράμματα οδήγησης συσκευών (drivers) στο παράδειγμα με τον ηλεκτρονικό υπολογιστή.

Αφού βασίζεται στο πρωτόκολλο OpenFlow, ο NOX υποστηρίζει όλο το βασικό ρεπερτόριο ενεργειών του πρωτοκόλλου, όπως προώθηση εξ ορισμού, προώθηση σε μια συγκεκριμένη έξοδο, απόρριψη πακέτου, άρνηση προώθησης, προώθηση σε μια διεργασία του ελεγκτή καθώς και τροποποίηση διαφόρων πεδίων της επικεφαλίδας του πακέτου (ετικέτες VLAN, IP διευθύνσεις αποστολέα και προορισμού και άλλων).

3.3.4 Λειτουργία του NOX

Ο NOX βασίζει την λειτουργία του στην αποστολή και εγκατάσταση ροών στους μεταγωγείς που επικοινωνούν άμεσα με αυτόν. Γενικά ο NOX είναι το κεντρικό σημείο παρακολούθησης και ελέγχου του δικτύου και λειτουργεί σαν ενδιάμεσος μεταξύ των εφαρμογών διαχείρισης και των συσκευών του δικτύου.

Όταν ένα εισερχόμενο πακέτο αντιστοιχίζεται επιτυχώς με μια καταχώριση ροής σε ένα μεταγωγέα, τότε ο μεταγωγέας ενημερώνει κατάλληλα το πεδίο μετρητών και εφαρμόζει τις προκαθορισμένες, από την ροή, ενέργειες. Με τις ενέργειες αυτές το πακέτο μπορεί να προωθηθεί σε επόμενη συσκευή μεταγωγέα, να εξέλθει από μια συγκεκριμένη θύρα εξόδου του μεταγωγέα, να μεταβληθούν κατάλληλα τα πεδία του πακέτου ή να προωθηθεί το πακέτο στον NOX ελεγκτή.

Αν το πακέτο δεν αντιστοιχίζεται με καμιά καταχώριση ροής, τότε προωθείται στον NOX για επεξεργασία και τελική απόφαση. Τυπικά αποστέλλονται τα πρώτα 200 bytes του πακέτου στον ελεγκτή αντί όλο το πακέτο αλλά αυτό μπορεί να αλλάξει στον ελεγκτή ανάλογα με τις αποφάσεις του διαχειριστή δικτύου. Τέτοια μη αντιστοιχίσιμα πακέτα μπορεί να είναι πακέτα από συγκεκριμένα πρωτοκόλλα (DNS για παράδειγμα) στα οποία ο NOX ρυθμίζεται να τα λάβει και να τα χειριστεί ο ίδιος, παρά να γίνει προώθηση στους μεταγωγείς του δικτύου, οπότε και δεν εγκαθίσταται καμιά καταχώριση ροής.

Εκτός αυτού, αποτυχία αντιστοίχισης μπορεί συχνά να παρατηρηθεί κατά τα πρώτα πακέτα μιας ροής, όταν γίνεται προσπάθεια να εγκατασταθεί η καταχώριση ροής σε όλους τους μεταγωγείς του δικτύου, η λεγόμενη αρχικοποίηση ροής (flow

initiation). Αν σε ένα μεταγωγέα δεν έχει προλάβει να γίνει η αρχικοποίηση της ροής, τότε η αντιστοίχιση αποτυγχάνει και τα πακέτα αποστέλλονται στον NOX.

Αυτή η αρχικοποίηση ροής είναι αρκετά χρήσιμη για τις εφαρμογές του NOX, διότι μπορεί να χρησιμοποιηθεί ,μαζί με άλλες πληροφορίες για την κίνηση στο δίκτυο, στην κατασκευή της βάσης δεδομένων για την κίνηση στο δίκτυο (Network View), καθώς και για τον καθορισμό επιθυμητών μονοπατιών δρομολόγησης κίνησης στο δίκτυο.

3.3.5 Απαιτήσεις για επεκτασιμότητα στον NOX

Για την εξασφάλιση της επεκτασιμότητας στον NOX πρέπει να ληφθούν υπόψη εκ των προτέρων κάποιες προϋποθέσεις και απαιτήσεις που να διασφαλίζουν την τήρηση των σωστών χρονικών κλιμάκων και της συνοχής του ελεγκτή. Σε ότι αφορά την τήρηση της σωστής κλίμακας χρόνου κατά την λειτουργία του NOX γίνεται μια διάκριση τριών διαφορετικών ρυθμών επεξεργασίας :

- Άφιξη πακέτων. Η άφιξη πακέτων γίνεται στην κλίμακα εκατομμυρίων αφίξεων ανά δευτερόλεπτο για μια ζεύξη των 10 Gbps.
- Αρχικοποίηση ροών. Βάση της λειτουργίας του NOX κατά την εγκατάσταση ροών, ο ρυθμός αρχικοποίησης ροών είναι τυπικά μιας ή μεγαλύτερης τάξης μεγέθους μικρότερος από τον ρυθμό άφιξης πακέτων.
- Αλλαγές στην βάση δεδομένων της κατάστασης δικτύου, που είναι συνήθως της τάξης των δεκάδων συμβάντων ανά δευτερόλεπτο για δίκτυα χιλιάδων συσκευών.

Σε ότι αφορά την συνοχή του ελεγκτή, η μόνη κατάσταση δικτύου που είναι καθολικά αποδεκτή είναι αυτή που περιγράφεται στην βάση δεδομένων κατάστασης δικτύου (Network View). Λέγοντας καθολικά αποδεκτή, εννοούμε την κατάσταση που χρησιμοποιείται από τις διεργασίες του NOX για να αντλήσουν δεδομένα για την τρέχουσα τοπολογία και χαρακτηριστικά του δικτύου και βάση αυτών να δημιουργήσουν τις κατάλληλες ροές για εγγραφή στους μεταγωγείς ανάλογα με τις απαιτήσεις των εφαρμογών. Η βάση δεδομένων δικτύου μεταβάλλεται αρκετά αργά ώστε να παρέχει αξιόπιστη πληροφόρηση σε όλες τις τρέχουσες διεργασίες του ελεγκτή.

Η απαίτηση της συνοχής ανακύπτει λόγω του ότι οι εφαρμογές που τρέχουν στον ελεγκτή επιτελούνται από τις διεργασίες του NOX που όπως αναφέρθηκε χρησιμοποιούν την βάση δεδομένων δικτύου για τις αποφάσεις ελέγχου. Έτσι αυτές οι αποφάσεις πρέπει να είναι οι ίδιες ανεξάρτητα από το ποια διεργασία του

NOX είναι υπεύθυνη για την όποια εφαρμογή και οι ροές που κατασκευάζονται και αποστέλλονται στους μεταγωγείς να είναι οι ίδιες, ανεξαρτήτως πάλι των διεργασιών του NOX που τις κατασκευάζουν. Από την στιγμή λοιπόν που ούτε η κατάσταση πακέτων (packet state) ή η κατάσταση ροών (flow state) είναι μέρη της ολικής κατάστασης δικτύου (Network View), οι καταστάσεις αυτές μπορούν να αποθηκευθούν και να μεταβληθούν τοπικά, χωρίς επιπτώσεις στην συνοχή του δικτύου.

Για κατηγορίες συμβάντων που επέρχονται σε ταχεία χρονική κλίμακα ο NOX μπορεί να χρησιμοποιήσει παραλληλισμό για εξασφάλιση της επεκτασιμότητας του δικτύου. Οι αφίξεις των πακέτων χειρίζονται από ανεξάρτητους μεταγωγείς χωρίς καθολικό συντονισμό για κάθε πακέτο στο δίκτυο, ενώ οι αρχικοποιήσεις ροών χειρίζονται από ανεξάρτητες διεργασίες του ελεγκτή, χωρίς πάλι να είναι αναγκαίος ο συντονισμός των ροών σε όλο το δίκτυο. Οι ροές μπορούν να επεξεργαστούν από οποιαδήποτε διεργασία του NOX, έτσι η χωρητικότητα του συστήματος μπορεί να αυξηθεί με την προσθήκη περισσότερων εξυπηρετητών για λειτουργία περισσότερων διεργασιών του ελεγκτή.

Όπως αναφέρθηκε η βάση δεδομένων της συνολικής κατάστασης δικτύου μεταβάλλεται αρκετά αργά και για αυτό μπορεί να διατηρείται και να αποθηκεύεται κεντρικά στον ίδιο τον NOX, ενώ μπορούν να κρατούνται και αντίγραφα ανά τακτά χρονικά διαστήματα για αύξηση της ανθεκτικότητας του δικτύου.

3.3.6 Εκτέλεση Εφαρμογών στον NOX

Οι εφαρμογές που αναπτύσσονται στον NOX μπορούν να γραφούν σε Python και C++ και χρησιμοποιούν τις προγραμματιστικές διεπαφές του NOX για την επιτέλεση των λειτουργιών τους. Τρέχουν σε διεργασίες του ελεγκτή και μπορούν να φορτωθούν σε αυτό με δυναμικό τρόπο.

Η ίδια αρχική υποδομή του NOX καθώς και οι κρίσιμες λειτουργίες του είναι γραμμένες και εφαρμοσμένες σε C++. Ενδεικτικά ο ίδιος ο κώδικας του ελεγκτή με όλα τα βασικά του τμήματα είναι περίπου 32000 γραμμές.

3.4 Το μοντέλο προγραμματιστικών διεπαφών του NOX

Ο NOX υιοθετεί ένα απλό μοντέλο προγραμματιστικών διεπαφών (programmatic interfaces) που περιστρέφεται γύρω από τρείς πυλώνες: τα συμβάντα (events), τον χώρο ονομάτων (namespace) και τη βάση δεδομένων κατάστασης δικτύου (Network View). Προγραμματιστικές διεπαφές επίσης υπάρχουν που αφορούν τον έλεγχο στο δίκτυο (control) και υπηρεσίες υψηλού επιπέδου (High Level Services).

3.4.1 Τα συμβάντα (Events)

Τα μεγάλης κλίμακας δίκτυα έχουν ένα πολύ ευμετάβλητο χαρακτήρα και δεν είναι στατικά με λίγες μεταβολές, όπως τα οικιακά δίκτυα. Ροές μπορεί να εγγράφονται και να διαγράφονται από τους μεταγωγείς, χρήστες μπορεί να εισέρχονται ή να αναχωρούν από το δίκτυο, οι διάφορες ζεύξεις μπορεί να πέφτουν ή να επανασυνδέονται και συσκευές μπορεί να προστίθενται ή να αφαιρούνται από το δίκτυο.

Για να ανταπεξέλθει σε όλες αυτά τα συμβάντα, οι εφαρμογές που τρέχουν στον NOX χρησιμοποιούν ένα σύνολο ρουτινών χειρισμού (handlers) που είναι καταχωρημένες στον NOX για να εκτελεστούν όταν ένα συγκεκριμένο συμβάν προκύψει στο δίκτυο. Οι χειριστές αυτών των συμβάντων εκτελούνται με σειρά προτεραιότητας, που ορίζεται κατά την καταχώριση και εγγραφή των χειριστών στον ελεγκτή. Η τιμή που επιστρέφει μια ρουτίνα χειρισμού υποδεικνύει στον NOX τις επόμενες ενέργειες που θα ακολουθήσει, αν θα σταματήσει την επεξεργασία για το προκύπτον συμβάν ή αν θα συνεχίσει, περνώντας την επεξεργασία του συμβάντος στην επόμενη ρουτίνα χειρισμού μέχρι την ολοκλήρωση του συμβάντος.

Εκτός από αυτά που αναφέρθηκαν πιο πάνω, κάποια συμβάντα δημιουργούνται απευθείας από μηνύματα OpenFlow στο δίκτυο, όπως switch join, switch leave, packet received και switch statistics received. Άλλα συμβάντα δημιουργούνται από τις εφαρμογές που τρέχουν στον NOX ως αποτέλεσμα επεξεργασίας χαμηλού επιπέδου συμβάντων ή/και συμβάντα που δημιουργούνται από άλλες εφαρμογές.

Για παράδειγμα, ο NOX περιλαμβάνει εφαρμογές που για να επικυρώσουν την αυθεντικότητα ενός χρήστη (authenticate) χρειάζεται η επαναδρομολόγηση της HTTP κίνησης του χρήστη σε μια εξωτερική ενεργή διαδικτυακή πύλη (web portal). Μετά την επικύρωση του χρήστη η εφαρμογή που αρχικά ζήτησε την αυθεντικοποίηση μπορεί να παράξει ένα συμβάν επιτυχούς αυθεντικοποίησης του χρήστη που να μπορεί να χρησιμοποιηθεί και από άλλες εφαρμογές.

Άλλες εφαρμογές που βασίζονται στην παραγωγή συμβάντων για την λειτουργία τους μπορεί να είναι εφαρμογές που ανακατασκευάζουν τις παραμέτρους ενός μεταγωγέα ή την τοπολογία σε επίπεδο τερματικών, εφαρμογές ανακάλυψης (discovery) υπηρεσιών δικτύου, αυθεντικοποίησης χρηστών και εξυπηρετητών και επιβολής πολιτικής δικτύου.

3.4.2 Βάση δεδομένων κατάστασης δικτύου (Network View) και χώρος ονομάτων (Namespace)

Ο NOX περιλαμβάνει αρκετές βασικές εφαρμογές που μπορούν να κατασκευάσουν την βάση δεδομένων δικτύου και να δημιουργήσουν ένα υψηλού επιπέδου χώρο ονομάτων που μπορούν να χρησιμοποιηθούν άμεσα από άλλες εφαρμογές.

Αυτού του είδους οι εφαρμογές χειρίζονται και την αυθεντικοποίηση χρηστών και εξυπηρετητών και μπορούν να προσδιορίσουν τα ονόματα τους στο δίκτυο με παρακολούθηση του DNS (domain name service) σε αυτό. Η συμπερίληψη υψηλού επιπέδου ονοματολογίας των συσκευών και των συσχετισμών τους στην βάση δεδομένων του δικτύου επιτρέπει σε οποιαδήποτε εφαρμογή του NOX να μετατρέπει ένα υψηλού επιπέδου όνομα συσκευής σε χαμηλού επιπέδου διεύθυνση (ή και αντίθετα) μέσω της βάσης δεδομένων. Αυτό δίνει την δυνατότητα γραφής εφαρμογών στον NOX ανεξαρτήτως της τοπολογίας και μορφής του υποκείμενου δικτύου.

Για την επίτευξη τέτοιων γρήγορων μετατροπών ονομάτων και διευθύνσεων, οι υψηλού επιπέδου δηλώσεις (declarations) στις εφαρμογές του NOX μπορούν να συνταχθούν (compiled) κατά αντιστοιχία με τις καταχωρίσεις στην βάση δεδομένων δικτύου για να παραχθούν λειτουργίες αναζήτησης χαμηλού επιπέδου που να μπορούν να εφαρμόζονται και ανά πακέτο. Φυσικά αυτές οι λειτουργίες πρέπει να ανασυντάσσονται (recompile) κάθε φορά που οι ονοματολογίες και οι συσχετισμοί τους αλλάζουν στην βάση δεδομένων δικτύου.

Λόγω του ότι η βάση δεδομένων δικτύου πρέπει να είναι συνεπής και άμεσα διαθέσιμη σε όλες τις διεργασίες του ελεγκτή, η εγγραφές και οι διαγραφές καταχωρίσεων από αυτή επισύρουν κάποιο κόστος στην ταχύτητα και απόδοση του ελεγκτή. Έτσι, οι εφαρμογές στον NOX πρέπει να εγγράφουν στην βάση δεδομένων δικτύου μόνο όταν εντοπιστεί μια αλλαγή στο δίκτυο και όχι για κάθε ληφθέν πακέτο από την εφαρμογή.

Αυτή η αντιμετώπιση είναι παρόμοια με το μοντέλο πρόσβασης μνήμης NUMA (non-uniform memory access) στην αρχιτεκτονική υπολογιστή για διατήρηση συνέπειας και όσο των δυνατών λιγότερο προσβάσεων στην μνήμη. Στην περίπτωση του NOX, όπως και των υπολογιστικών συστημάτων, μια κακογραμμένη εφαρμογή, όσο αφορά την πρόσβαση στην βάση δεδομένων, μπορεί να οδηγήσει σε σοβαρή υποβάθμιση της συνολικής απόδοσης του δικτύου.

3.4.3 Προγραμματιστικές Διεπαφές για τον έλεγχο του δικτύου

Για τον έλεγχο του δικτύου προσφέρονται διεπαφές διαχείρισης από τον NOX, οι οποίες ασκούν λειτουργίες ελέγχου με χρήση των εντολών του πρωτοκόλλου OpenFlow. Το ψηλό αφαιρετικό επίπεδο των μεταγωγέων του δικτύου επιτρέπει στις εφαρμογές του ελεγκτή να προσθέτουν, να μεταβάλλουν ή να διαγράφουν καταχωρίσεις ροής από τους πίνακες ροής των μεταγωγέων καθώς και να αντλούν διάφορες πληροφορίες για την διαμόρφωση της κίνησης στο δίκτυο διαβάζοντας τους μετρητές που υπάρχουν στις καταχωρήσεις ροής κάθε μεταγωγέα.

Μέσω αυτής της δυνατότητας τροποποίησης ροών μια εφαρμογή διαχείρισης έχει πλήρη έλεγχο της δρομολόγησης σε επίπεδο ζεύξης δεδομένων (Level 2 του OSI) και σε θέματα χειρισμού της επικεφαλίδας των πακέτων. Με τον ορισμό περισσότερων εντολών διαχείρισης ενεργειών στους μεταγωγείς οι εφαρμογές διαχείρισης θα μπορούσαν να ελέγχουν ακόμα και την βασική επεξεργασία δεδομένων ανά πακέτο σε θέματα όπως κρυπτογράφηση και περιορισμού ταχύτητας.

Φυσικά το αφαιρετικό επίπεδο του πρωτόκολλου OpenFlow είναι φτιαγμένο για να προσφέρει ένα πιο γενικό περιβάλλον προγραμματισμού, οπότε τέτοιες εξειδικευμένες λειτουργίες είναι δύσκολο να υποστηριχθούν από OpenFlow ελεγκτές όπως ο NOX. Αντί αυτού ο NOX προσφέρει την δυνατότητα δρομολόγησης μέσω ενδιάμεσων συσκευών (middle boxes) οι οποίες να διαθέτουν δυνατότητες επεξεργασίας και επιθεώρησης πακέτων – DPI (Deep Packet Inspection).

Ο NOX προσφέρει και ένα σύνολο υπηρεσιών υψηλού επιπέδου, εν συντομία HLS (High Level Services). Αυτές οι υπηρεσίες παρέχονται με την χρήση ενός συνόλου βιβλιοθηκών συστήματος που περιλαμβάνει ο ελεγκτής και περιέχουν αποτελεσματικές υλοποιήσεις συναρτήσεων λειτουργιών (functions) που χρησιμοποιούνται συχνά από διάφορες εφαρμογές που τρέχουν στον NOX. Αυτές οι συναρτήσεις περιλαμβάνουν λειτουργίες όπως δρομολόγηση (routing module), γρήγορη κατηγοριοποίηση πακέτων (fast packet classification), κοινές υπηρεσίες

δικτύου (όπως DHCP και DNS) καθώς και λειτουργία φιλτραρίσματος της κίνησης του δικτύου βάση πολιτικής (policy-based filtering module).

3.4.4 Στόχοι για το περιβάλλον Διεπαφής του NOX και Περιορισμοί

Ο NOX με το προγραμματιστικό περιβάλλον που παρέχει αποσκοπεί στην ανάπτυξη μιας πρακτικής και κατανοητής πλατφόρμας για την δημιουργία των εφαρμογών διαχείρισης δικτύου. Οι εφαρμογές αυτές μπορούν εύκολα να επεκταθούν ανάλογα με την αύξηση του δικτύου, όπως φαίνεται και από την περιγραφή της λειτουργίας των προγραμματιστικών διεπαφών του NOX ανωτέρω.

Εντούτοις, από την στιγμή που το προγραμματιστικό περιβάλλον του NOX είναι προσανατολισμένο στην ευελιξία και επεκτασιμότητα στην ανάπτυξη εφαρμογών, υπάρχουν θέματα ασφαλείας που πρέπει να ληφθούν υπόψη κατά την ανάπτυξη του κώδικα των εφαρμογών. Παρόλο που ο NOX προσφέρει ένα επίπεδο ασφάλειας κατά την ανάπτυξη και λειτουργία εφαρμογών, υπάρχουν περιορισμοί όσο αφορά την λειτουργία και απομόνωση των εφαρμογών του NOX μεταξύ τους.

Για παράδειγμα, υποθέτουμε ότι υπάρχει κάποιος συντονισμός μεταξύ προγραμματιστών διαφορετικών εφαρμογών και έτσι δεν γίνεται προσπάθεια για ανάπτυξη προστασίας σε κακόβουλες ή κακογραμμένες εφαρμογές. Μια ελαττωματική εφαρμογή μπορεί λανθασμένα να απορρίψει πακέτα ή συμβάντα, να εγγράψει σε τυχαία τμήματα μνήμης ή να οδηγήσει το σύστημα σε ατέρμονα βρόγχο (infinite loop). Οπότε, είναι καλό κατά την ανάπτυξη των εφαρμογών να λαμβάνονται υπόψη οι περιορισμοί σε θέματα ασφαλείας του NOX, για αποφυγή προβληματικών καταστάσεων κατά την λειτουργία του ελεγκτή.

3.5 Παραδείγματα Απλών Εφαρμογών του NOX

Για καλύτερη κατανόηση του προγραμματιστικού περιβάλλον του NOX και των δυνατοτήτων του, παρουσιάζονται πιο κάτω κάποια απλά παραδείγματα εφαρμογών με χρήση των API's του NOX.

3.5.1 Δημιουργία ετικετών VLAN

Πιο κάτω παρουσιάζεται ο κώδικας για μια εφαρμογή του NOX που ορίζει κανόνες για ετικέτες VLAN όσο αφορά την αυθεντικοποίηση χρήστη, βάση προκαθορισμένων αντιστοιχίσεων χρήστη – VLAN.

```
# On user authentication, statically setup VLAN tagging
# rules at the user's first hop switch
def setup_user_vlan(dp, user, port, host):
      vlanid = user_to_vlan_function(user)
# For packets from the user, add a VLAN tag
      attr_out[IN_PORT] = port
      attr_out[DL_SRC] = nox.reverse_resolve(host).mac
      action_out = [(nox.OUTPUT, (0, nox.FLOOD)),(nox.ADD_VLAN,
(vlanid))]
      install_datapath_flow(dp, attr_out, action_out)
# For packets to the user with the VLAN tag, remove it
      attr_in[DL_DST] = nox.reverse_resolve(host).mac
      attr_in [DL_VLAN] = vlanid
      action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
      install_datapath_flow(dp, attr_in, action_in)
nox.register_for_user_authentication(setup_user_vlan)
```

Βάσει του πιο πάνω κώδικα η εφαρμογή θέτει με στατικό τρόπο ετικέτες VLAN κατά την αυθεντικοποίηση ενός χρήστη. Η εντολή ανάθεσης της ετικέτας VLAN στην ροή για πακέτα από τον χρήστη είναι :

```
action_out = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.ADD_VLAN, (vlanid))]
install_datapath_flow (dp, attr_out, action_out)
```

Για πακέτα κατευθυνόμενα στον χρήστη αφαιρείται η ετικέτα VLAN με τις εντολές:

```
action_in = [(nox.OUTPUT, (0, nox.FLOOD)), (nox.DEL_VLAN)]
install_datapath_flow (dp, attr_in, action_in)
```

Κατά το τρέξιμο της εφαρμογής ο NOX είναι υπεύθυνος για τον εντοπισμό όλων των αρχικοποιήσεων ροών, την ανάθεση της ροής στον σωστό χρήστη και εξυπηρετητή με τα αντίστοιχα σημεία πρόσβασης και την αποστολή του συμβάντος στην εφαρμογή.

3.5.2 Απλή διαδικασία σάρωσης για ξενιστές (hosts)

```
scans = defaultdict (dict)
def check_for_scans(dp, inport, packet):
       dstid = nox.resolve_host_dest(packet)
       if dstid == None:
              scans [packet.l2.srcaddr][packet.l2.dstaddr] = 1
                if packet.l3 != None:
                scans [packet.l2.srcaddr] [packet.l3.dstaddr] = 1
              if len(scans[packet.l2.srcaddr].keys()) > THRESHOLD:
                print nox.resolve_user_source_name(packet)
                print nox.resolve_host_source_name(packet)
# To be called on all packet-in events
nox.register_for_packet_in(check_for_scans)
```

Αυτή η εφαρμογή προσπαθεί να εντοπίσει διαθέσιμους ξενιστές μετρώντας τον αριθμό μοναδικών L2 και L3 προορισμών που ο ξενιστής προσπαθεί να επικοινωνήσει χωρίς να έχουν αυθεντικοποιηθεί. Ο NOX έχει πρόσβαση σε όλη την κίνηση του δικτύου και μπορεί να χρησιμοποιήσει την βάση δεδομένων δικτύου η οποία παρακολουθεί όλους τους αυθεντικοποιημένους ξενιστές του δικτύου για τον εντοπισμό των μη αυθεντικοποιημένων.

3.5.3 Το Ethane

Το Ethane είναι μια εφαρμογή δικτύου που προσφέρει πρόσβαση και έλεγχο σε ολόκληρο το δίκτυο χρησιμοποιώντας μια κεντροποιημένη πολιτική δηλώσεων με χρήση οντοτήτων υψηλού αφαιρετικού επιπέδου. Μιας και το Ethane αναπτύχθηκε και στον NOX και σε άλλους ελεγκτές είναι ένα ενδιαφέρον παράδειγμα που δείχνει πως ο NOX απλοποιεί την όλη διαδικασία ανάπτυξης. Συγκεκριμένα, ενώ για την ανάπτυξη του Ethane σε αυτόνομο κώδικα C++ χρειάστηκαν περίπου 45 000 γραμμές κώδικα, κατά την ανάπτυξη του σε Python για NOX χρειάστηκαν μόνο μερικές χιλιάδες γραμμές κώδικα.

Το Ethane έχει δύο απαιτήσεις οι οποίες κάνουν δύσκολη την ανάπτυξη και εφαρμογή του με την χρήση κλασσικών μεθόδων διαχείρισης δικτύου. Κατά πρώτον, απαιτείται η εκ των προτέρων γνώση των οντοτήτων του δικτύου, όπως είναι οι χρήστες και οι κόμβοι (μεταγωγείς, δρομολογητές). Κατά δεύτερο απαιτείται να υπάρχει έλεγχος όσο αφορά την δρομολόγηση στο επίπεδο που να περιλαμβάνει τα εξής στοιχεία: πηγή χρήστη, πηγή ξενιστή, κόμβος πρώτης μεταπήδησης, πρωτόκολλο δικτύου, προορισμός χρήστη, προορισμός ξενιστή, κόμβος τελευταίας μεταπήδησης. Αυτές οι δύο απαιτήσεις ικανοποιούνται εγγενώς από τον NOX, αφού η εφαρμογή έχει απευθείας πρόσβαση στις οντότητες αποστολέα και προορισμού, υπάρχει συσχέτιση μεταξύ των συμβάντων στο δίκτυο και η μονάδα δρομολόγησης του NOX υποστηρίζει καθορισμό συγκεκριμένης διαδρομής στο δίκτυο.

Για να εφαρμοστεί η βασική λειτουργικότητα του Ethane στον NOX πρέπει να γίνεται έλεγχος κάθε ροής με την πολιτική του δικτύου που ορίζεται στο Ethane και περνώντας τους εκάστοτε περιορισμούς στην μονάδα δρομολόγησης του NOX. Λόγω όμως αυτού του ελέγχου για κάθε ροή προκύπτουν κάποια θέματα αποδοτικότητας του ελεγκτή καθώς η συνεχής σάρωση του αρχείου πολιτικής δικτύου για κάθε ροή που διαμορφώνεται από τον NOX εισάγει καθυστέρηση καθώς η πολυπλοκότητα της πολιτικής που ακολουθείται αυξάνεται.

3.6 Θέματα Επίδοσης του NOX και η εισαγωγή παραλληλισμού – ο NOX-MT

Με την δημιουργία του NOX και την γενικότερη εξέλιξη του SDN τέθηκαν κάποια ερωτήματα σχετικά με την απόδοση σε ένα δίκτυο του ελεγκτή NOX και γενικότερα της προσέγγισης της αρχιτεκτονικής SDN, ως προς τους χρόνους καθυστέρησης (latency), αποκρισιμότητας (responsiveness) και της ρυθμαπόδοσης του ελεγκτή (throughput).

Σε γενικές γραμμές ενώ η SDN αρχιτεκτονική είναι αποτελεσματική για χρήση σε περιβάλλοντα δικτύων όπως είναι τα οικιακά, κέντρα δεδομένων και εταιρικά δίκτυα εντούτοις η μετάθεση του συστήματος ελέγχου σε ένα κεντρικό απομακρυσμένο μηχάνημα δημιουργεί ερωτήματα σχετικά με την όλη προσέγγιση. Τα κυριότερα είναι, κατά πρώτον, πόσο γρήγορα μπορεί ένας ελεγκτής όπως ο NOX να ανταποκριθεί σε αιτήσεις δημιουργίας μονοπατιών στο δίκτυο (data-paths) και δεύτερον, πόσες τέτοιες αιτήσεις μπορεί να χειριστεί ανά δευτερόλεπτο.

Από διάφορες μελέτες προέκυψε ότι ο NOX μπορεί να χειριστεί αποτελεσματικά γύρω στις 30 000 αρχικοποιήσεις ροών ανά δευτερόλεπτο ενώ διατηρεί ένα χρόνο εγκατάστασης ροής σε μεταγωγέα κάτω από 10ms. Δυστυχώς σε πιο μεγάλα ή/και πολύπλοκα δίκτυα αυτές οι επιδόσεις δεν είναι ικανοποιητικές αφού σε ένα δίκτυο με 1500 εξυπηρετητές ο μέσος ρυθμός άφιξης ροών είναι γύρω στις 100 000 ροές ανά δευτερόλεπτο. Επίσης σε δίκτυα με 100 περίπου μεταγωγείς μπορούν να υπάρξουν –στο χειρότερο σενάριο- αιχμές άφιξης ροών που να αγγίζουν το ένα εκατομμύριο ροές το δευτερόλεπτο. Λαμβάνοντας υπόψη και την καθυστέρηση των 10ms για την εγκατάσταση ροής πρέπει να προστεθεί και ένα ποσοστό 10% επιπλέον καθυστέρησης για την πλειοψηφία των ροών που αρχικοποιούνται σε ένα τέτοιο δίκτυο μεταγωγέων.

Οι περιορισμοί στην επίδοση κατά την λειτουργία του ελεγκτή NOX δεν οφείλονται στην αρχιτεκτονική SDN, αλλά είναι αποτέλεσμα της ανάπτυξης του

ίδιου του ελεγκτή, καθώς ο NOX ήταν ο πρώτος ελεγκτής που εισήγαγε επίσημα την SDN λειτουργικότητα σε δίκτυα υπολογιστών. Έτσι ο NOX δεν βελτιστοποιήθηκε ποτέ για αύξηση της απόδοσης του και βασίστηκε σε μονονηματική λειτουργία (single threaded).

Αυτοί οι περιορισμοί στην επίδοση του NOX έδωσαν την αφορμή για διάφορες προσπάθειες βελτίωσης της επίδοσης των ελεγκτών που βασίζονται στην SDN αρχιτεκτονική. Στο πλαίσιο αυτών των προσπαθειών έγινε και η ανάπτυξη του NOX-MT από μια ερευνητική ομάδα του πανεπιστημίου του Τορόντο.

Ο NOX-MT είναι ένας ελεγκτής OpenFlow που μπορεί να υποστηρίξει πολυνηματική επεξεργασία (multi-threaded) και αναπτύχθηκε ειδικά για την βελτίωση της επίδοσης του NOX ελεγκτή.

3.6.1 Τα χαρακτηριστικά του NOX-MT

Ο NOX-MT είναι ελαφρώς τροποποιημένη έκδοση του ελεγκτή NOX που χρησιμοποιεί τεχνικές βελτιστοποίησης για την εισαγωγή της πολυνηματικής επεξεργασίας και για βελτίωση της ρυθμαπόδοσης και χρόνου απόκρισης του NOX.

Αυτές οι τεχνικές βελτιστοποίησης συμπεριλαμβάνουν ομαδοποίηση εισόδων/εξόδων (I/O batching) για ελαχιστοποίηση της επιβάρυνσης του συστήματος κατά την διάρκεια μαζικών αιτήσεων εισόδων/εξόδων (E/E), την επανακωδικοποίηση του συστήματος χειρισμού E/E για την υποστήριξη και ενίσχυση της βιβλιοθήκης ασύγχρονου χειρισμού E/E (ASIO – Asynchronous I/O). Με την χρήση της ASIO απλοποιείται αρκετά και επιταχύνεται η πολυνηματική επεξεργασία στον ελεγκτή. Τέλος έγινε χρήση μιας υλοποίησης του υποσυστήματος απελευθέρωσης και παραχώρησης μνήμης malloc, η οποία λαμβάνει υπόψη το πολυνηματικό περιβάλλον και βελτιώνει τους χρόνους απόκρισης της μνήμης.

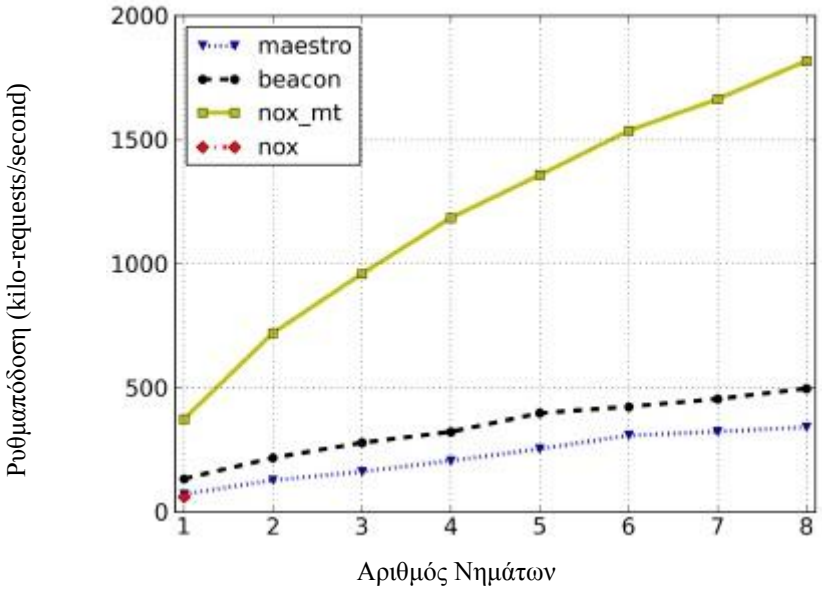
Πέραν αυτών των βελτιστοποιήσεων υπάρχουν και άλλοι τομείς στους οποίους υπάρχουν ανεπάρκειες στον NOX αλλά και στον NOX-MT, όπως είναι η βαριά χρήση δυναμικής παραχώρησης μνήμης και πλεονάζων αντίγραφα μνήμης για κάθε αίτηση καθώς και η χρήση τεχνικών κλειδώματος κατά την λειτουργία των ελεγκτών τη στιγμή που υπάρχουν εναλλακτικές μέθοδοι που εισάγουν σημαντικά λιγότερη καθυστέρηση.

Όσο αφορά την λειτουργία του NOX-MT, ο ελεγκτής λειτουργεί σαν τυπικός ελεγκτής OpenFlow, ο οποίος ρυθμίζει και αποστέλλει καταχωρίσεις ροών σε OpenFlow μεταγωγείς. Η ρύθμιση και εγκατάσταση ροής μπορεί να γίνει στατικά πριν την άφιξη πακέτων που να χρησιμοποιούν την ροή (προληπτική πολιτική – proactive) ή δυναμικά, σαν μέρος της αναζήτησης κατά την άφιξη πακέτου για

αντιστοίχιση ροής στον μεταγωγέα (αντιδραστική πολιτική – reactive). Οι αντιδραστικές ροές έχουν μεγαλύτερη καθυστέρηση, κυρίως όταν πρόκειται για τα πρώτα πακέτα μιας ροής.

3.6.2 Η ρυθμαπόδοση του NOX-MT

Η ρυθμαπόδοση ενός ελεγκτή είναι ένας σημαντικός παράγοντας για την απόφαση του αριθμού των ελεγκτών που θα αναπτυχθούν σε ένα δίκτυο. Για την καλύτερη αξιολόγηση της απόδοσης του NOX-MT γίνεται σύγκριση με άλλους τρεις ελεγκτές, τους Beacon, Maestro και τον απλό NOX.



Σχήμα 3.6.2-1 Ρυθμαπόδοση του NOX-MT

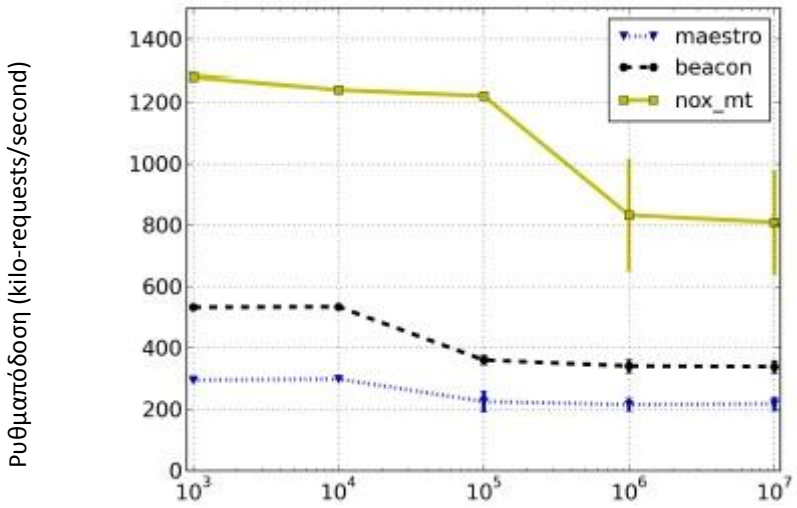
Το ανωτέρω σχήμα δείχνει την μέγιστη ρυθμαπόδοση των ελεγκτών OpenFlow, με βάση των αριθμών νημάτων. Ο NOX έχει μόνο ένα σημείο, μιας και είναι μονονηματικός. Προφανώς ο NOX-MT έχει την υψηλότερη ρυθμαπόδοση, με 2000 kilo-requests/second όταν έχουμε 8 παράλληλα νήματα. Τα οφέλη από την εισαγωγή της πολυνηματικής επεξεργασίας είναι προφανή καθώς η απόδοση όλων των ελεγκτών αυξάνεται ανάλογα με την αύξηση των νημάτων.

Υπό ιδανικές περιστάσεις η ρυθμαπόδοση των ελεγκτών σε ένα δίκτυο δεν επηρεάζεται από τον αριθμό των μεταγωγέων του δικτύου. Εντούτοις, ο αυξανόμενος ανταγωνισμός μεταξύ των νημάτων για υπολογιστικό χρόνο, οι ιδιομορφίες του πρωτοκόλλου TCP καθώς και ο χρονοπρογραμματισμός εργασιών εντός του ελεγκτή επηρεάζουν εν τέλει την ρυθμαπόδοση σε δίκτυα με υψηλό αριθμό ενεργών μεταγωγέων. Επιπρόσθετα, η ολική ρυθμαπόδοση μειώνεται όταν αναπτυχθεί στο δίκτυο ένας μεγάλος αριθμός μεταγωγέων που να ξεπερνά ένα

συγκεκριμένο κατώφλι. Το κατώφλι αυτό στον αριθμό μεταγωγέων οφείλεται σε φαινόμενα όπως η υπερύψωση εντολών χειρισμού E/E (I/O overhead), ανταγωνισμό στην ουρά αναμονής εντολών και άλλους κοινόχρηστους πόρους του συστήματος και στην μείωση της αποδοτικότητας κατά την τμηματοποίηση εργασιών (job batching).

Όσο αφορά τον φόρτο εργασίας του δικτύου για όλους τους ελεγκτές, εφόσον το δίκτυο δουλεύει με μια συγκεκριμένη και σταθερή ρυθμαπόδοση, η αύξηση των αιτήσεων (requests) αυξάνει γραμμικά και την καθυστέρηση στο δίκτυο ούτως ώστε να διατηρείται σταθερή η ρυθμαπόδοση. Φυσικά είναι δυνατό να δοθεί προτεραιότητα στην εκπλήρωση των αιτήσεων, αλλά αυτό θα έχει μια αντίστοιχη μείωση στην ολική ρυθμαπόδοση.

Τέλος γίνεται μια σύγκριση των ελεγκτών σε φορτίο με εντατικές εντολές γραψίματος (write-intensive), πράγμα που αυξάνει τον ανταγωνισμό στις εφαρμογές ελέγχου του δικτύου. Τα αποτελέσματα σε τέτοιου είδους διεργασίες φαίνονται πιο κάτω:



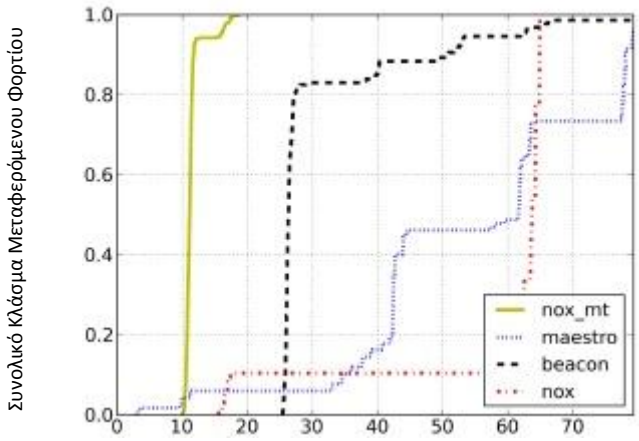
Σχήμα 3.6.2-2 Ρυθμαπόδοση του NOX-MT σε write-intensive διεργασίες

Παρόλο που ελεγκτές όπως ο Beacon και ο Maestro φαίνεται να επηρεάζονται σημαντικά από τέτοιου είδους φόρτο εργασίας, ο NOX-MT αποδίδει καλύτερα σε εργασίες εντατικού γραψίματος. Αυτό συμβαίνει διότι η εφαρμογή μεταγωγής του NOX-MT (switch application) ελαχιστοποιεί τον ανταγωνισμό μεταξύ των διεργασιών, χωρίζοντας τον πίνακα διευθύνσεων MAC του δικτύου σε ένα σύνολο επιμέρους πινάκων (hash tables) που επιλέγονται βάση της εκάστοτε MAC διεύθυνσης.

3.6.3 Ο χρόνος απόκρισης του NOX-MT

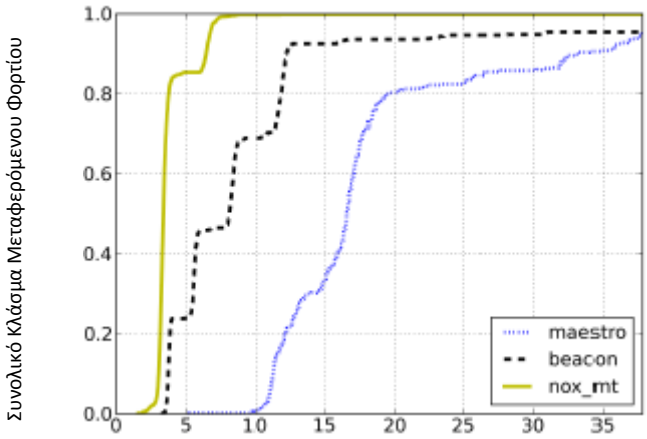
Γενικά, τα δίκτυα που βασίζονται σε SDN αρχιτεκτονική εγκαθιστούν τις καταχωρίσεις ροών στους μεταγωγείς με αντιδραστικό τρόπο, δηλαδή μετά την δημιουργία ανάγκης για νέα ροή. Με αυτό τον τρόπο επηρεάζεται άμεσα ο χρόνος απόκρισης των ελεγκτών κατά την δρομολόγηση των διαφόρων πακέτων στο δίκτυο.

Για την εκτίμηση του χρόνου απόκρισης των ελεγκτών πρέπει να ληφθούν υπόψη δύο μετρικές, ο ελάχιστος χρόνος απόκρισης και ο μέγιστος χρόνος απόκρισης. Ο ελάχιστος χρόνος απόκρισης είναι ο χρόνος που απαιτείται για ένα ακριβώς πακέτο στο δίκτυο να διανύσει την διαδρομή από τον μεταγωγέα στον εκάστοτε ελεγκτή. Ο μέγιστος χρόνος απόκρισης είναι ο χρόνος που απαιτείται για την μεταφορά του μέγιστου φορτίου γραμμής, δηλαδή για τον μέγιστο αριθμό πακέτων που μπορεί να σταλούν από τον μεταγωγέα στον ελεγκτή με πλήρη γέμιση της γραμμής και των καταχωρητών (buffer) των συσκευών.

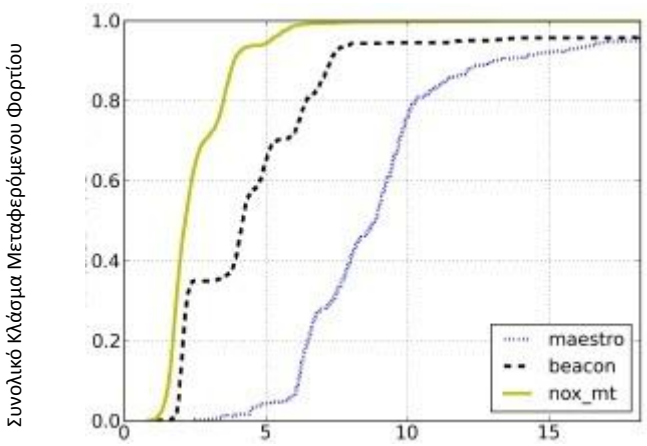


Σχήμα 3.6.3-1 Χρόνος Απόκρισης σε μονονηματική επεξεργασία

Ο NOX-MT έχει τον μικρότερο χρόνο απόκρισης, όσο αφορά την μονονηματική επεξεργασία, πράγμα λογικό μιας και ο NOX-MT έχει την μεγαλύτερη ρυθμαπόδοση όπως φάνηκε και προηγουμένως, άρα από την στιγμή που τα δύο αυτά μεγέθη είναι αντιστρόφως ανάλογα έχουμε και τα αντίστοιχα αποτελέσματα.



Σχήμα 3.6.3-2 Χρόνος Απόκρισης σε επεξεργασία 4 νημάτων



Σχήμα 3.6.3-2 Χρόνος Απόκρισης σε επεξεργασία 8 νημάτων

Με την αύξηση των νημάτων κατά την διακίνηση του ίδιου φορτίου στην γραμμή παρατηρείται μεγάλη βελτίωση στον χρόνο απόκρισης όλων των ελεγκτών, με τον NOX-MT να έχει συστηματικά καλύτερη επίδοση από τους υπόλοιπους ελεγκτές. Τέλος όσο αφορά τον αριθμό των ενεργών μεταγωγέων στο δίκτυο, παρατηρείται μια αύξηση στην καθυστέρηση και στον χρόνο απόκρισης για όλους τους ελεγκτές όσο αυξάνονται οι συσκευές στο δίκτυο, πράγμα αναμενόμενο άλλωστε αφού απαιτείται περισσότερος χρόνος επεξεργασίας και διάθεσης πόρων από τον εκάστοτε ελεγκτή για κάθε συσκευή στο δίκτυο.

ΚΕΦΑΛΑΙΟ 4

4.1 Ο ελεγκτής NOX και τα εργαλεία ανάπτυξης εφαρμογών

Για την δημιουργία του αλγορίθμου δρομολόγησης και εφαρμογής καταλλήλων ροών στην τοπολογία δικτύου που υποστηρίζει το πρωτόκολλο OpenFlow χρησιμοποιούνται τα εργαλεία που παρέχονται από το προγραμματιστικό περιβάλλον του ελεγκτή NOX.

Τα εργαλεία αυτά ανήκουν σε δύο κατηγορίες. Η πρώτη κατηγορία είναι οι έτοιμες συστατικές εφαρμογές (applications) που παρέχει ο NOX. Αυτές οι εφαρμογές αποτελούν ένα σύνολο έτοιμων ρουτινών που μπορεί να εκτελέσει ο NOX κατευθείαν με την κλήση τους από τον ελεγκτή. Τέτοιες εφαρμογές είναι για παράδειγμα το pyloop.py, το countdown.py, το dnsspy.py και αρκετές άλλες που παρέχονται στην έκδοση προγραμματιστών (developer) του NOX.

Η δεύτερη κατηγορία είναι το σύνολο των προγραμματιστικών διεπαφών (API – application programming interfaces) που δίνονται από τον NOX. Οι προγραμματιστικές αυτές διεπαφές είναι οι δομικοί λίθοι με τους οποίους κατασκευάζονται και αναπτύσσονται οι εφαρμογές του NOX. Αυτές οι διεπαφές παρέχονται σε δύο κυρίως αρχεία του NOX, το core.py και το util.py .

Το core.py περιλαμβάνει την πλειονότητα των βασικών διεργασιών που καλείται να εκτελέσει ο NOX. Αυτές οι διεργασίες ποικίλουν και αφορούν θέματα όπως την διαδικασία καταχώρησης νέου μεταγωγέα στον ελεγκτή, την απόδοση flow id σε αυτόν και την εγγραφή στατιστικών στοιχείων για την συσκευή. Άλλη διεργασία μπορεί να αφορά την συμπεριφορά του ελεγκτή κατά την παραλαβή πακέτου OpenFlow από τον NOX (packet_in_event σύμφωνα με το API) ή και την διαδικασία εγκατάστασης ροής σε μια συσκευή σύμφωνα με το API install_datapath_flow. Τέλος, το util.py περιλαμβάνει κάποιες συμπληρωματικές διεργασίες του NOX που μπορούν να κληθούν άμεσα από μια εφαρμογή, χωρίς να παραστεί ανάγκη δημιουργίας επιπλέον κώδικα από τον προγραμματιστή της εφαρμογής.

4.2 Ανάλυση των βασικών προγραμματιστικών διεπαφών του NOX

Για να γίνει κατανοητός ο τρόπος λειτουργίας του ελεγκτή NOX και των αρχών ανάπτυξης της application – aware εφαρμογής είναι απαραίτητο να γίνει μια πιο εις βάθος ανάλυση των βασικών προγραμματιστικών διεπαφών του ελεγκτή. Η γλώσσα προγραμματισμού που χρησιμοποιείται για την έκδοση προγραμματιστών (developer's release) του ελεγκτή είναι η Python, έκδοση 2.6 . Παρέχεται επίσης και η επιλογή για ανάπτυξη σε C++, αλλά προτιμάται η Python λόγω αμεσότητας και μεγαλύτερης ευκολίας για την παραμετροποίηση του NOX.

Να προστεθεί ότι υπάρχει μια ποικιλία διακλαδώσεων (εκδόσεων) της προγραμματιστικής έκδοσης του NOX, καθώς και της νεότερης έκδοσης ελεγκτή OpenFlow που βασίστηκε στον NOX, του POX. Οι εκδόσεις αυτές κατασκευάστηκαν (forked) από την αρχική έκδοση του NOX (ή POX αντίστοιχα) υπάρχουν ως διανομές κώδικα από κατάλληλες υπηρεσίες (πχ github). Για την παρούσα ανάπτυξη χρησιμοποιείται η έκδοση zaku του ελεγκτή NOX, κυρίως λόγω της σταθερότητας της κατά την περίοδο της ανάπτυξης της εφαρμογής. Άλλες εκδόσεις του NOX είναι η impulse, η destiny και η νεότερη verity. Για τον POX υπάρχουν οι εκδόσεις upgrade_switch_2, upgrade_switch ,master, fix_license, debugger_legacy_2012-06-27, debugger_legacy,debugger, carp και betta, με την betta να είναι η νεότερη.

Αξίζει να σημειωθεί ότι οι έτοιμες εφαρμογές που παρέχονται με την έκδοση προγραμματιστών του NOX είναι γραμμένες σχεδόν εξ' ολοκλήρου σε Python. Αναλυτικά κάποιες από αυτές τις εφαρμογές θα εξηγηθούν στην συνέχεια και είναι πλήρως βασισμένες στις βασικές προγραμματιστικές διεπαφές του NOX.

Οι βασικές προγραμματιστικές διεπαφές του ελεγκτή NOX βρίσκονται στο αρχείο βιβλιοθήκης (lib) του πηγαίου (source) κώδικα του NOX. Οι προγραμματιστικές αυτές διεπαφές επισυνάπτονται στο παράρτημα Α. Για την ανάπτυξη και υλοποίηση του μηχανισμού δρομολόγησης που απαιτείται από την διπλωματική εργασία χρησιμοποιούνται οι διεπαφές που περιγράφονται εκατέρωθεν. Όλες αυτές οι διεπαφές βρίσκονται στο αρχείο core.py του NOX που διανέμεται μαζί με την έκδοση προγραμματιστών του ελεγκτή.

4.2.1 Τα βασικά τμήματα της κλάσης μιας εφαρμογής

Κατά την ανάπτυξη εφαρμογών στον NOX ελεγκτή πρέπει να ακολουθείται ένα συγκεκριμένο μοντέλο κατά την γραφή του κώδικα. Η δομή που πρέπει να ακολουθηθεί κατά την γραφή σε Python είναι η ακόλουθη:

```
class foo(Component):


        def (self, ctxt):

            Component.(self, ctxt)


        def install(self):

            # register for event here

            pass


        def getInterface(self):

            return str(foo)


    def getFactory():

        class Factory:

            def instance(self, ctxt):

                return foo(ctxt)


        return Factory()
```

Διάγραμμα 4.2.1 – Η βασική δομή μιας εφαρμογής στον NOX

Κατά την ανάπτυξη των εφαρμογών στον NOX ακολουθείται το μοντέλο του προγραμματισμού ανεξάρτητων τμημάτων (components) με την χρήση της Python. Αυτές οι εφαρμογές μπορούν να κληθούν και να εκτελεστούν κατευθείαν μέσω του εκτελέσιμου αρχείου του NOX – το nox_core.

Γενικά δηλώνεται μια αρχική κλάση (class foo) στον κώδικα που μπορεί να χρησιμοποιήσει τις διεπαφές που παρέχονται από τον NOX για την επιτέλεση της λειτουργίας της εκάστοτε εφαρμογής.

Το τμήμα `def __init__(self, ctxt)` χρησιμοποιείται για την αρχικοποίηση της εφαρμογής κατά την εκκίνηση της και την φόρτωση στον ελεγκτή. Ακολούθως, στο `def configure(self, config)` η εφαρμογή ρυθμίζεται και περνά τις ρυθμίσεις της (παραμέτρους, μεταβλητές, δεδομένα) στον ελεγκτή και μπορεί να καλέσει και άλλες εφαρμογές για επιτέλεση διαφόρων λειτουργιών,

δεδομένου ότι έχουν οριστεί σωστά στην αρχική εφαρμογή. Στο `def install(self)` η εφαρμογή εγκαθιστά στον ελεγκτή τα απαραίτητα τμήματα κώδικα που θα κληθούν κατά την εκτέλεση της και μπορεί πλέον να λειτουργήσει κανονικά, καθώς και να κληθεί από άλλες, εξώτερες, εφαρμογές που μπορεί να χρειαστούν να καλέσουν την εφαρμογή αυτή για κάποιες λειτουργίες.

Τέλος για να τρέξει η εφαρμογή δημιουργείται η ρουτίνα εκτέλεσης factory (`def getFactory():` στο ανωτέρω παράδειγμα) που αναλαμβάνει την κατασκευή και το τρέξιμο της εφαρμογής σύμφωνα με την λειτουργία της Python.

4.2.2 Χειρισμός συμβάντων

Ο παρεχόμενος κώδικας (API) από το core.py είναι:

```
def register_event(self, event_name):
    return self.ctxt.register_event(event_name)

def register_python_event(self, event_name):
    return self.ctxt.register_python_event(event_name)

def register_handler(self, event_name, handler):
    return self.ctxt.register_handler(event_name, handler)

def post_timer(self, event):
    return self.ctxt.post_timer(event)
```

Οι πιο πάνω διεπαφές χρησιμοποιούνται όταν ο ελεγκτής καλείται να αντιμετωπίσει κάποιο συμβάν, όπως για παράδειγμα την άφιξη ενός πακέτου OpenFlow από κάποιο μεταγωγέα του υποκείμενου δικτύου. Συγκεκριμένα χρησιμοποιούνται οι διεπαφές `def register_event(self, event_name)` και `def register_python_event(self, event_name)` ανάλογα με το είδος του συμβάντος που προκύπτει.

Για τον χειρισμό των πιο πάνω συμβάντων είναι αναγκαίο να οριστεί και μια ρουτίνα χειριστή (handler) που σε συνεργασία με τις διεπαφές συμβάντων προωθεί το ληφθέν συμβάν για επεξεργασία από την εφαρμογή του ελεγκτή. Η ρουτίνα χειρισμού είναι η `register_handler(self, event_name, handler)` που δημιουργεί ξεχωριστό νήμα ελέγχου (control thread) για κάθε συμβάν που χρειάζεται να αντιμετωπίσει ή/και να επεξεργαστεί ο NOX ελεγκτής. Οι ρουτίνες χειρισμού μπορούν να εξειδικευτούν ανάλογα με το συμβάν που προκύπτει, για αυτό υπάρχουν και εξειδικευμένοι χειριστές, όπως ο χειριστής για παραλαβή πακέτου που αναφέρεται πιο κάτω, στο 4.1.7.

Η ρουτίνα του `def post_timer(self, event)` μπορεί να χρησιμοποιηθεί για τον χρονισμό της εφαρμογής καθώς και για λειτουργίες που απαιτούν κάποια μορφή μέτρησης χρόνου.

4.2.3 Αποστολή πακέτου OpenFlow

Ο παρεχόμενος κώδικας (API) από το core.py είναι:

```
    def send_openflow_packet(self, dp_id, packet, actions,
                             inport=openflow.OFPP_CONTROLLER):

        if type(packet) == type(array.array('B')):
            packet = packet.tostring()

        if type(actions) == types.IntType:
            self.ctxt.send_openflow_packet_port(dp_id,packet,
actions, inport)
        elif type(actions) == types.ListType:
            oactions = self.make_action_array(actions)
            if oactions == None:
                raise Exception('Bad action')
            self.ctxt.send_openflow_packet_acts(dp_id,packet,
oactions, inport)
        else:
            raise Exception('Bad argument')
```

Η προγραμματιστική αυτή διεπαφή χρησιμοποιείται για την αποστολή ενός πακέτου OpenFlow από τον ελεγκτή σε ένα συνδεδεμένο με αυτόν μεταγωγέα. Για να μπορεί να γίνει αυτή η εργασία είναι απαραίτητα να δοθούν στον ελεγκτή κάποια δεδομένα. Αυτά είναι το `dp_id`, δηλαδή το αναγνωριστικό της διόδου δεδομένων (datapath) της ροής για την οποία προορίζεται το πακέτο, το `packet`, που είναι το ίδιο το σύνολο των δεδομένων προς αποστολή, τα `actions`, η δέσμη ενεργειών που καλείται να εκτελέσει ο μεταγωγέας που θα παραλάβει το πακέτο και το `inport`, η θύρα εισόδου του πακέτου για πληροφορίες πηγής (source), που για αυτή την διεπαφή είναι η θύρα εισόδου του ελεγκτή (`openflow.OFPP_CONTROLLER`).

Με την παραλαβή του πακέτου ελέγχεται ο τύπος του πακέτου [`type(packet)`], και ακολούθως ο τύπος των ενεργειών που παρέχονται για μεταβίβαση στον μεταγωγέα για ενθυλάκωση και αποστολή του πακέτου με την εντολή `self.ctxt.send_openflow_packet_acts(dp_id,packet,actions, inport)`.

4.2.4 Ρουτίνα για τροποποίηση ροής

Ο παρεχόμενος κώδικας (API) από το core.py είναι:

```
    def send_flow_command(self, dp_id, command, attrs,
                          priority=openflow.OFP_DEFAULT_PRIORITY,
                          add_args=None,
                          hard_timeout=openflow.OFP_FLOW_PERMANENT):
        m = set_match(attrs)
        if m == None:
            return False

        if command == openflow.OFPFC_ADD:
            (idle_timeout, actions, buffer_id) = add_args
            oactions = self.make_action_array(actions)
            if oactions == None:
                return False
        else:
            idle_timeout = 0
            oactions = ""
            buffer_id = UINT32_MAX

        self.ctxt.send_flow_command(dp_id, command, m, idle_timeout,
                                    hard_timeout,oactions, buffer_id,
priority)

        return True
```

Συχνά, κατά την λειτουργία του, ο ελεγκτής μπορεί να χρειαστεί να τροποποιήσει τις καταχωρήσεις ροών στους πίνακες ροών (flow tables) των μεταγωγέων στο δίκτυο. Για να γίνει αυτό χρησιμοποιείται μια εντολή ροής (flow command) που εκδίδεται από τον ελεγκτή και αποστέλλεται στους κατάλληλους μεταγωγείς.

Η εντολή ροής απαρτίζεται από τα πεδία `dp_id`, `command`, `attrs`, `priority`, `add_args` και `hard_timeout`. Dp_id είναι το αναγνωριστικό πεδίο της διόδου δεδομένων της ροής,command είναι το είδος της εντολής τροποποίησης, αν πρόκειται για παράδειγμα για προσθήκη ενεργειών, τύπου `OFPFC_ADD`. Το πεδίο attrs αφορά το σύνολο ιδιοτήτων και χαρακτηριστικών που θα διαθέτει η νέα ροή και χρησιμοποιείται για την ανίχνευση και εύρεση της ορθότητας όλων των στοιχείων της με την εντολή `m = set_match(attrs)`.Το priority υποδεικνύει την προτεραιότητα που θα δοθεί στην εντολή τροποποίησης, που εδώ καθορίζεται ως η εξ' ορισμού (`OFP_DEFAULT_PRIORITY`). Το `add_args` χρησιμεύει για χαρακτηρισμό None κάποιων στοιχειών της ροής αν αυτό παραστεί αναγκαίο, σύμφωνα με το

```
        if command == openflow.OFPFC_ADD:
            (idle_timeout, actions, buffer_id) = add_args
```

Τέλος, το `hard_timeout` αναφέρεται στον χρόνο κατά τον οποίο θα παραμείνει ενεργή η νέα καταχώριση ροής στον μεταγωγέα, προτού λήξει και αφαιρεθεί από τον πίνακα ροών.

4.2.5 Ρουτίνες για διαγραφή καταχωρίσεων ροών

Ο παρεχόμενος κώδικας (API) από το core.py είναι:

```
    def delete_datapath_flow(self, dp_id, attrs):

        return self.send_flow_command(dp_id, openflow.OFPFC_DELETE,
attrs)

    def delete_strict_datapath_flow(self, dp_id, attrs,
                        priority=openflow.OFP_DEFAULT_PRIORITY):

        return self.send_flow_command(dp_id,
openflow.OFPFC_DELETE_STRICT, attrs, priority)
```

Για την διαγραφή καταχωρίσεων ροών από τους πίνακες ροών των μεταγωγέων του δικτύου χρησιμοποιούνται αυτές οι διεπαφές από τον ελεγκτή NOX. Για την επιτυχή διαγραφή καταχώρισης ροής είναι απαραίτητο να είναι γνωστό το αναγνωριστικό της διόδου δεδομένων (datapath) της ροής `dp_id` καθώς και τα χαρακτηριστικά της ροής (`attrs`).

Υπάρχουν δύο τύποι εντολών διαγραφής καταχώρισης ροής, η `delete_datapath_flow` και η `delete_strict_datapath_flow`. Η `delete_datapath_flow` διαγράφει όλες τις καταχωρίσεις ροής που έχουν το δοθέν αναγνωριστικό διόδου δεδομένων `dp_id` ανεξαρτήτως λοιπών πεδίων και ιδιοτήτων ροών. Η `delete_strict_datapath_flow` διαγράφει τις καταχωρίσεις ροής που έχουν ακριβή αντιστοιχία με τα πεδία ιδιοτήτων (`attrs`) που εμπεριέχονται στην εντολή διαγραφής.

4.2.6 Ρουτίνα εγγραφής καταχώρισης ροής για μια δίοδο δεδομένων σε πίνακα ροής

Ο παρεχόμενος κώδικας (API) από το core.py είναι:

```
    def install_datapath_flow(self, dp_id, attrs, idle_timeout,
hard_timeout, actions, buffer_id=None,
priority=openflow.OFP_DEFAULT_PRIORITY, inport=None, packet=None):

        if buffer_id == None:
            buffer_id = UINT32_MAX

        self.send_flow_command(dp_id, openflow.OFPFC_ADD, attrs,
priority, (idle_timeout, actions, buffer_id), hard_timeout)

        if buffer_id == UINT32_MAX and packet != None:
            for action in actions:
                if action[0] == openflow.OFPAT_OUTPUT:
                    self.send_openflow_packet(dp_id, packet,
                                              action[1][1], inport)
```

```
        else:
            raise NotImplementedError
```

Για την εγγραφή μιας καταχώρισης ροής σε ένα πίνακα ροών ενός μεταγωγέα χρησιμοποιείται η διεπαφή `install_datapath_flow`. Αυτό απαιτεί ένα σύνολο δεδομένων που πρέπει να δοθούν στην διεπαφή για να έχουμε μια επιτυχή εγκατάσταση ροής. Αυτά τα δεδομένα είναι:

`dp_id`: Η δίοδος δεδομένων (datapath) στην οποία θα εφαρμοστεί η ροή, προωθώντας και λαμβάνοντας πακέτα δεδομένων μέσω αυτής.

`attrs`: Τα χαρακτηριστικά και οι ιδιότητες της ροής που ορίζονται με αυτό το πεδίο και εφαρμόζονται κατάλληλα στον πίνακα ροής του μεταγωγέα. Αυτά τα χαρακτηριστικά είναι:

DL_SRC = "dl_src": Η διεύθυνση MAC πηγής (Ethernet) των πακέτων που αφορούν την ροή.
DL_DST = "dl_dst": Η διεύθυνση MAC προορισμού (Ethernet) των πακέτων που αφορούν την ροή.
DL_VLAN = "dl_vlan": Ετικέτα VLAN για πακέτα που βρίσκονται σε VLAN δίκτυο.
DL_VLAN_PCP = "dl_vlan_pcp": Προτεραιότητα για VLAN πακέτα που βρίσκονται σε VLAN δίκτυο.
DL_TYPE = "dl_type": Τύπος Ethernet συσκευής.
NW_SRC = "nw_src": Η διεύθυνση IPv4 πηγής των πακέτων που αφορούν την ροή.
NW_DST = "nw_dst": Η διεύθυνση IPv4 προορισμού των πακέτων που αφορούν την ροή.
NW_PROTO = "nw_proto": Λεπτομέρειες για το πρωτόκολλο IPv4 και το opcode για το ARP(address resolution protocol).
TP_SRC = "tp_src": Θύρα TCP πηγής των πακέτων που αφορούν την ροή.
TP_DST = "tp_dst": Θύρα TCP προορισμού των πακέτων που αφορούν την ροή.

`idle_timeout`: Ο χρόνος αναμονής χωρίς δραστηριότητα (idle) της ροής προτού διαγραφεί αυτόματα η ροή από τον πίνακα του μεταγωγέα.

`hard_timeout`: Ο χρόνος ολικής παραμονής της ροής προτού διαγραφεί αυτόματα η ροή από τον πίνακα του μεταγωγέα.

`actions`: Είναι μια λίστα όπου κάθε καταχώρηση της είναι επίσης λίστα δύο στοιχείων που αντιπροσωπεύουν μια ενέργεια. Το πρώτο στοιχείο (Elem 0) από μια λίστα ενεργειών θα πρέπει να είναι τύπου ofp_action_type και το επόμενο στοιχείο

(elem 1) θα πρέπει να είναι το επιχείρημα ενέργειας (αν χρειάζεται). Για ενέργεια OFPAT_OUTPUT, το πεδίο actions πρέπει να είναι μια άλλη λίστα δύο στοιχείων με μέγιστο μήκος το πρώτο στοιχείο και αριθμό θύρας (port_no) το δεύτερο.

`buffer_id`: Το αναγνωριστικό του καταχωρητή μνήμης (buffer) στον οποίο θα πρέπει να εφαρμοστούν κάποιες από τις ενέργειες (actions) που περιγράφονται ανωτέρω. Παίρνει την τιμή None αν δεν υπάρχουν κάποιες ενέργειες που να αφορούν τον καταχωρητή.

`priority`: Αυτή η τιμή καθορίζει την σειρά με την οποία εφαρμόζονται οι ροές σε ένα μεταγωγέα, δίνοντας σε αυτές με υψηλότερη τιμή αντίστοιχα ψηλότερη θέση στον πίνακα ροών του μεταγωγέα.

`packet`: Σε περίπτωση που το είναι buffer_id έχει τιμή None τότε με το packet παρέχεται το πακέτο δεδομένων στο οποίο θα εφαρμοστούν οι ενέργειες που περιγράφονται στο actions.

`inport`: Κατά την αποστολή ενός πακέτου από τον ελεγκτή γίνεται έλεγχος για την θύρα εισόδου, οπότε όταν γίνεται flood ένα πακέτο να μην αποστέλλεται πίσω στην θύρα εισόδου του.

4.2.7 Ρουτίνα χειρισμού κατά την παραλαβή πακέτου

Ο παρεχόμενος κώδικας (API) από το core.py είναι:

```
    def register_for_packet_in(self, handler):

        self.register_handler(Packet_in_event.static_get_name(),
                              gen_packet_in_cb(handler))
```

Κατά την παραλαβή ενός πακέτου OpenFlow από τον NOX μπορεί να κληθεί αυτή η ρουτίνα χειρισμού συμβάντος. Με αυτό τον τρόπο δημιουργείται ένας χειριστής (handler) πακέτων για την εισερχόμενη κίνηση στον ελεγκτή. Αυτός ο χειριστής καταχωρεί για κάθε πακέτο τα στοιχεία του και έχει την δομή:

```
        handler (dp_id, inport, ofp_reason, total_frame_len,
buffer_id, captured_data)
```

Dp_id, inport και buffer_id αντιστοιχούν στα γνωστά πεδία που αναλύονται πιο πάνω. Το ofp_reason περιέχει δεδομένα του πρωτοκόλλου OpenFlow, το total_frame_len περιέχει την τιμή μεγέθους του πακέτου και τέλος το captured_data τα ληφθέντα δεδομένα του πακέτου.

4.2.8 Διάφορες ρουτίνες χειρισμού

Αντίστοιχοι χειριστές μπορούν να δημιουργηθούν για ένα αριθμό συμβάντων και λειτουργιών στον ελεγκτή.

```
    def register_for_flow_removed(self, handler):
        self.register_handler(Flow_removed_event.static_get_name(),
                              handler)

    def register_for_flow_mod(self, handler):
        self.register_handler(Flow_mod_event.static_get_name(),
                              handler)
```

- Αυτοί οι χειριστές δημιουργούνται σε περιπτώσεις διαγραφής και τροποποίησης ροών από τον ελεγκτή με ενέργειες όπως `send_flow_command` και `delete_datapath_flow`.

```
    def register_for_bootstrap_complete(self, handler):

self.register_handler(Bootstrap_complete_event.static_get_name(),
                              handler)

    def register_for_datapath_join(self, handler):
        self.register_handler(Datapath_join_event.static_get_name(),
                              gen_dp_join_cb(handler))

    def register_for_datapath_leave(self, handler):

        self.register_handler(Datapath_leave_event.static_get_name(),
                              gen_dp_leave_cb(handler))
```

- Αυτοί οι χειριστές χρησιμεύουν για την παρακολούθηση και διαχείριση συμβάντων προσθήκης ροών σε διόδους δεδομένων (datapath) ή αφαίρεσης ροών από διόδους δεδομένων.

```
    def register_for_switch_mgr_join(self, handler):

self.register_handler(Switch_mgr_join_event.static_get_name(),
                              gen_switch_mgr_join_cb(handler))



    def register_for_switch_mgr_leave(self, handler):

self.register_handler(Switch_mgr_leave_event.static_get_name(),
                              gen_switch_mgr_leave_cb(handler))
```

- Αυτοί οι χειριστές παρακολουθούν και ενημερώνουν για την κατάσταση των μεταγωγέων στον ελεγκτή, ποιοι ενώνονται στο δίκτυο ή ποιοι βγαίνουν εκτός σύνδεσης.

4.3 Ανάλυση βοηθητικών βιβλιοθηκών του NOX

Πέραν των βασικών προγραμματιστικών διεπαφών που διαθέτει ο NOX, παρέχονται και κάποιες βοηθητικές βιβλιοθήκες κώδικα για κάποιες ενέργειες που είναι απαραίτητο να επιτελέσει ο ελεγκτής. Αυτές οι βοηθητικές βιβλιοθήκες βρίσκονται στο αρχείο util.py που διανέμεται μαζί με την έκδοση για προγραμματιστές (developer's release) του ελεγκτή NOX.

4.3.1 Η ρουτίνα αντιστοίχισης πεδίων πακέτων και ροών set_match

Ο παρεχόμενος κώδικας (API) από το util.py είναι:

```
def set_match(attrs):
    m = openflow.ofp_match()
    wildcards = 0
    num_entries = 0

    if attrs.has_key(core.IN_PORT):
        m.in_port = htons(attrs[core.IN_PORT])
        num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_IN_PORT

    if attrs.has_key(core.DL_VLAN):
        m.dl_vlan = htons(attrs[core.DL_VLAN])
        num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_DL_VLAN

    if attrs.has_key(core.DL_VLAN_PCP):
        m.dl_vlan_pcp = attrs[core.DL_VLAN_PCP]
        num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_DL_VLAN_PCP

    if attrs.has_key(core.DL_SRC):
        v = convert_to_eaddr(attrs[core.DL_SRC])
        if v == None:
            print 'invalid ethernet addr'
            return None
        m.set_dl_src(v.octet)
        num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_DL_SRC

    if attrs.has_key(core.DL_DST):
        v = convert_to_eaddr(attrs[core.DL_DST])
        if v == None:
            print 'invalid ethernet addr'
            return None
        m.set_dl_dst(v.octet)
        num_entries += 1
    else:
```

```
        wildcards = wildcards | openflow.OFPFW_DL_DST

    if attrs.has_key(core.DL_TYPE):
        m.dl_type = htons(attrs[core.DL_TYPE])
        num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_DL_TYPE

    if attrs.has_key(core.NW_SRC):
        v = convert_to_ipaddr(attrs[core.NW_SRC])
        if v == None:
            print 'invalid ip addr'
            return None
        m.nw_src = v
        num_entries += 1

        if attrs.has_key(core.NW_SRC_N_WILD):
            n_wild = attrs[core.NW_SRC_N_WILD]
            if n_wild > 31:
                wildcards |= openflow.OFPFW_NW_SRC_MASK
            elif n_wild >= 0:
                wildcards |= n_wild << openflow.OFPFW_NW_SRC_SHIFT
            else:
                print 'invalid nw_src wildcard bit count', n_wild
                return None
            num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_NW_SRC_MASK

    if attrs.has_key(core.NW_DST):
        v = convert_to_ipaddr(attrs[core.NW_DST])
        if v == None:
            print 'invalid ip addr'
            return None
        m.nw_dst = v
        num_entries += 1

        if attrs.has_key(core.NW_DST_N_WILD):
            n_wild = attrs[core.NW_DST_N_WILD]
            if n_wild > 31:
                wildcards |= openflow.OFPFW_NW_DST_MASK
            elif n_wild >= 0:
                wildcards |= n_wild << openflow.OFPFW_NW_DST_SHIFT
            else:
                print 'invalid nw_dst wildcard bit count', n_wild
                return None
            num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_NW_DST_MASK

    if attrs.has_key(core.NW_PROTO):
        m.nw_proto = attrs[core.NW_PROTO]
        num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_NW_PROTO

    if attrs.has_key(core.NW_TOS):
        m.nw_tos = attrs[core.NW_TOS]
        num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_NW_TOS
```

```
    if attrs.has_key(core.TP_SRC):
        m.tp_src = htons(attrs[core.TP_SRC])
        num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_TP_SRC

    if attrs.has_key(core.TP_DST):
        m.tp_dst = htons(attrs[core.TP_DST])
        num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_TP_DST

    if num_entries != len(attrs.keys()):
        print 'undefined flow attribute type in attrs', attrs
        return None

    m.wildcards = c_htonl(wildcards)
    return m
```

Η ρουτίνα αυτή παρέχεται από την βιβλιοθήκη του NOX για να μπορούν να πραγματοποιηθούν λειτουργίες αναζήτησης και ταύτισης των πεδίων αντιστοίχισης που περιλαμβάνουν τα πακέτα OpenFlow με τις ροές που είναι εγκαταστημένες στους μεταγωγείς ή κατασκευάζονται από τον ελεγκτή για να διαχειριστούν την κίνηση των πακέτων στο δίκτυο (και οι οποίες ροές αποστέλλονται στους μεταγωγείς για εγκατάσταση).

Η ρουτίνα set_match διαθέτει μηχανισμό αντιστοίχισης για όλα τα πεδία που γίνεται η σύγκριση πακέτων και ροών. Τα αποτελέσματα των δοκιμών επιτυχούς ή μη επιτυχούς ταύτισης επιστρέφονται μέσω της παραμέτρου m, που είναι ένας συγκεντρωτικός πίνακας αποτελεσμάτων βάση του Openflow πρωτοκόλλου, το `openflow.ofp_match()`.

Συνοπτικά τα πεδία αυτά είναι:

	Πεδίο Δεδομένων	Πεδίο set_match
1.	Ingress Port	m.in_port
2.	Metadata	--
3.	Ether src	m.set_dl_src
4.	Ether dst	m.set_dl_dst
5.	Ether type	m.dl_type
6.	VLAN id	m.dl_vlan
7.	VLAN priority	m.dl_vlan_pcp
8.	MPLS label	--
9.	MPLS tra_c class	--
10.	IPv4 src	m.nw_src
11.	IPv4 dst	m.nw_dst
12.	IPv4 proto / ARP opcode	m.nw_proto
13.	IPv4 ToS bits	m.nw_tos

14.	TCP/ UDP / SCTP src port ICMP Type	m.tp_src
15.	TCP/ UDP / SCTP dst port ICMP Code	m.tp_dst

Πίνακας 4.3.1 – Πεδία παραμέτρων δικτύου στο OpenFlow

4.3.2 Η ρουτίνα εξαγωγής πεδίων ροής από πακέτο extract_flow

Ο παρεχόμενος κώδικας (API) από το util.py είναι:

```
def extract_flow(ethernet):

    attrs = {}
    attrs[core.DL_SRC] = ethernet.src
    attrs[core.DL_DST] = ethernet.dst
    attrs[core.DL_TYPE] = ethernet.type
    p = ethernet.next

    if isinstance(p, vlan):
        attrs[core.DL_VLAN] = p.id
        attrs[core.DL_VLAN_PCP] = p.pcp
        p = p.next
    else:
        attrs[core.DL_VLAN]  =  0xffff  #  XXX  should  be  written
OFP_VLAN_NONE
        attrs[core.DL_VLAN_PCP] = 0

    if isinstance(p, ipv4):
        attrs[core.NW_SRC] = p.srcip
        attrs[core.NW_DST] = p.dstip
        attrs[core.NW_PROTO] = p.protocol
        p = p.next

        if isinstance(p, udp) or isinstance(p, tcp):
            attrs[core.TP_SRC] = p.srcport
            attrs[core.TP_DST] = p.dstport
        else:
            if isinstance(p, icmp):
                attrs[core.TP_SRC] = p.type
                attrs[core.TP_DST] = p.code
            else:
                attrs[core.TP_SRC] = 0
                attrs[core.TP_DST] = 0
    else:
        attrs[core.NW_SRC] = 0
        attrs[core.NW_DST] = 0
        attrs[core.NW_PROTO] = 0
        attrs[core.TP_SRC] = 0
        attrs[core.TP_DST] = 0
    return attrs
```

Η ρουτίνα αυτή μπορεί να εξάγει τα πεδία ενδιαφέροντος από ένα ληφθέν πακέτο OpenFlow και να τα θέτει ως πεδία σε μια νέα ροή υπό κατασκευή από τον ελεγκτή. Ταυτόχρονα με την εξαγωγή, η ρουτίνα αυτή θέτει τα χαρακτηριστικά

πεδία της νέας ροής (Ethernet source, Ethernet destination, Ethernet type, VLAN ID, VLAN priority, IPv4 source, IPv4 destination, IPv4 protocol, TCP source και TCP destination) άμεσα με τις εντολές:

```
attrs[core.DL_SRC] = ethernet.src
attrs[core.DL_DST] = ethernet.dst
attrs[core.DL_TYPE] = ethernet.type
attrs[core.DL_VLAN] = p.id
attrs[core.DL_VLAN_PCP] = p.pcp
attrs[core.NW_SRC] = p.srcip
attrs[core.NW_DST] = p.dstip
attrs[core.NW_PROTO] = p.protocol
attrs[core.TP_SRC] = p.srcport
attrs[core.TP_DST] = p.dstport
```

4.4 Εφαρμογές του NOX που παρέχονται με την έκδοση προγραμματιστών

Ο ελεγκτής NOX παρέχει μια γκάμα εφαρμογών στην έκδοση για προγραμματιστές, οι οποίες παρέχουν διάφορες λειτουργίες και αποτελούν υπόβαθρο για ανάπτυξη άλλων εξειδικευμένων εφαρμογών. Οι εφαρμογές αυτές στον NOX χωρίζονται σε τρεις κατηγορίες:

1. Webapps – Εφαρμογές που αποσκοπούν στην ανάπτυξη λειτουργιών διαδικτυακής μορφής, όπως εξυπηρετητής δικτύου (webserver.py) ή λειτουργίες αυθεντικοποίησης χρηστών στο διαδίκτυο (webauth.py).
2. Netapps – Εφαρμογές που υλοποιούν διάφορες λειτουργίες που αφορούν το δίκτυο, όπως διαδικασίες δρομολόγησης (normal_routing.cc, samplerouting.py) ή εύρεσης τοπολογίας (pytopology.cc, discovery.py).
3. Coreapps – Βασικές εφαρμογές του NOX που υλοποιούν άμεσα λειτουργίες που παρέχει το πρωτόκολλο OpenFlow. Αυτές οι λειτουργίες αφορούν την δημιουργία, εγκατάσταση, τροποποίηση ή αφαίρεση ροών σε μεταγωγείς (switch.cc), ανάλυση στατιστικών στοιχείων των μεταγωγέων του δικτύου (packetdump.py) και βασικών (barebone) εφαρμογών (pyloop.py) που μπορούν να χρησιμοποιηθούν ως βάση για ανάπτυξη άλλων λειτουργιών.

4.4.1 Η εφαρμογή pyloop.py

```
from nox.lib.core import *

class Pyloop(Component):

    def __init__(self, ctxt):
        Component.__init__(self, ctxt)

    def install(self):
        import code
        code.interact()
```

```
    def getInterface(self):
        return str(Pyloop)


def getFactory():
    class Factory:
        def instance(self, ctxt):
            return Pyloop(ctxt)

    return Factory()
```

- Η εφαρμογή pyloop.py είναι η πιο απλή εφαρμογή που παρέχεται έτοιμη από τον ελεγκτή NOX. Διαθέτει την βασική δομή ανάπτυξης εφαρμογών, δηλαδή το class, εδώ δηλώνεται ως `Pyloop`, τα αναγκαία components, εδώ `__init__(self, ctxt), install(self), getInterface(self)` και το `getFactory()` που τρέχει την εφαρμογή βάση της Python.

4.4.2 Η εφαρμογή countdown.py

```
from nox.lib.core     import *
import logging

logger = logging.getLogger('nox.coreapps.examples.countdown')
numbers = ["one","two","three"]
index = 0

class countdown(Component):

    def __init__(self, ctxt):
        Component.__init__(self, ctxt)

    def install(self):
        # call every second
        self.post_callback(1, lambda : self.count_down())

    def getInterface(self):
        return str(countdown)

    def count_down(self):

        global index
        # No, this isn't mispelled:.  If you're curious, see Farscape
        # episode 1.17
        logger.debug("%s %s" % (numbers[index], 'mippippi'))
        index+=1
        if index < len(numbers):
            self.post_callback(1, lambda : self.count_down())

def getFactory():
    class Factory:
        def instance(self, ctxt):
            return countdown(ctxt)

    return Factory()
```

- Η εφαρμογή countdown απλά μετράει ένα πίνακα αριθμών και επιστρέφει το τυπωμένο αποτέλεσμα στην οθόνη. Είναι και αυτή απλή εφαρμογή και χρησιμεύει στην αξιοποίηση της ρουτίνας `self.post_callback(1, lambda : self.count_down())` που αναλαμβάνει την μέτρηση. Το τυπωμένο αποτέλεσμα είναι το ακόλουθο:

00050|nox|DBG:Datapath 0023205dd18d sent error in response to capability reply, assuming no management support
00051|nox|DBG:No switch auth module registered, auto-approving switch
00052|nox|DBG:Registering switch with DPID = 23205dd18d
00053|nox.coreapps.examples.countdown|DBG:one mippippi
00054|nox.coreapps.examples.countdown|DBG:two mippippi
00055|nox.coreapps.examples.countdown|DBG:three mippippi

4.4.3 Η εφαρμογή packetdump.py

```
import traceback
from nox.lib.packet import *
from nox.lib.core import *

from twisted.python import log

def print_packet(dp, inport, reason, len, bid, packet):
    # packet is already parsed (using code in nox/lib/packet) so just
    # print it and let __str__ do all the work
    print packet

class packetdump(Component):

    def __init__(self, ctxt):
        Component.__init__(self, ctxt)

    def install(self):
        self.register_for_packet_in(print_packet)

    def getInterface(self):
        return str(packetdump)


def getFactory():
    class Factory:
        def instance(self, ctxt):
            return packetdump(ctxt)

    return Factory()
```

- Μια άλλη εφαρμογή είναι η packetdump. Η εφαρμογή αυτή τυπώνει στην οθόνη του τερματικού που τρέχει ο NOX τις παραμέτρους των πακέτων που λαμβάνει ο NOX από τους μεταγωγείς του δικτύου. Αυτό επιτυγχάνεται με τον ορισμό και το τρέξιμο μιας ρουτίνας, της `print_packet`, η οποία δέχεται ως είσοδο τις παραμέτρους που εξηγήθηκαν ανωτέρω (`dp, inport, reason, len, bid, packet`) και τυπώνει τα στοιχεία του πακέτου, όπως φαίνεται πιο κάτω:

00359|nox|DBG:Registering switch with DPID = 232050effd
00360|openflow|DBG:Passive tcp interface received connection
00361|openflow-event|DBG:received packet-in event from 00232050effd (len:60)
[a4:56:30:d1:91:02>01:80:c2:00:00:00:llc]
00362|openflow|DBG:stream: negotiated OpenFlow version 0x01 (we support versions 0x01 to 0x01 inclusive, peer no later than version 0x01)
00363|nox|DBG:Success sending in 'sending switch config'
00364|nox|DBG:Success sending in 'receiving features reply'
00365|nox|DBG:Success receiving in 'receiving features reply'
00366|nox|DBG:Success sending in 'receiving ofmp capability reply'
00367|nox|DBG:Success receiving in 'receiving ofmp capability reply'
00368|nox|DBG:Datapath 00232050effd sent error in response to capability reply, assuming no management support
00369|nox|DBG:No switch auth module registered, auto-approving switch
00370|nox|DBG:Registering switch with DPID = 232050effd
00371|openflow-event|DBG:received packet-in event from 00232050effd (len:60)
[**a4:56:30:d1:91:02>a4:56:30:d1:91:02:9000]**

- Εδώ φαίνεται από τα επισημασμένα σημεία πως έχουμε παραλαβή πακέτων OpenFlow από τον ελεγκτή με MAC προέλευσης το a4:56:30:d1:91:02 και MAC προορισμού τα 01:80:c2:00:00:00:llc και a4:56:30:d1:91:02:9000. Αυτή η εφαρμογή είναι χρήσιμη για να αναλυθούν τα πακέτα που λαμβάνει ο ελεγκτής και να δούμε τις παραμέτρους των πακέτων πριν προχωρήσουμε σε κάποια επεξεργασία.

4.4.4 Η εφαρμογή monitor.py

```
from nox.lib.core     import *
from   nox.coreapps.pyrt.pycomponent   import   Table_stats_in_event,
Aggregate_stats_in_event
from nox.lib.openflow import OFPST_TABLE,   OFPST_PORT, ofp_match,
OFPP_NONE
from nox.lib.packet.packet_utils import longlong_to_octstr

MONITOR_TABLE_PERIOD     = 3
MONITOR_PORT_PERIOD      = 3
MONITOR_AGGREGATE_PERIOD = 3

class Monitor(Component):

    def __init__(self, ctxt):
        Component.__init__(self, ctxt)

    def aggregate_timer(self, dpid):
        flow = ofp_match()
        flow.wildcards = 0xffff
        self.ctxt.send_aggregate_stats_request(dpid, flow,  0xff)
        self.post_callback(MONITOR_TABLE_PERIOD,       lambda       :
self.aggregate_timer(dpid))

    def table_timer(self, dpid):
```

```
        self.ctxt.send_table_stats_request(dpid)
        self.post_callback(MONITOR_TABLE_PERIOD,       lambda       :
self.table_timer(dpid))

    def port_timer(self, dpid):
        self.ctxt.send_port_stats_request(dpid, OFPP_NONE)
        self.post_callback(MONITOR_PORT_PERIOD,       lambda       :
self.port_timer(dpid))

    # For each new datapath that joins, create a timer loop that
monitors
    # the statistics for that switch
    def datapath_join_callback(self, dpid, stats):
        self.post_callback(MONITOR_TABLE_PERIOD,       lambda       :
self.table_timer(dpid))
        self.post_callback(MONITOR_PORT_PERIOD   +   1,   lambda   :
self.port_timer(dpid))
        self.post_callback(MONITOR_AGGREGATE_PERIOD  +  2,  lambda  :
self.aggregate_timer(dpid))

    def aggregate_stats_in_handler(self, dpid, stats):
        print     "Aggregate     stats     in     from     datapath",
longlong_to_octstr(dpid)[6:]
        print '\t',stats

    def table_stats_in_handler(self, dpid, tables):
        print      "Table      stats      in      from      datapath",
longlong_to_octstr(dpid)[6:]
        for item in tables:
            print '\t',item['name'],':',item['active_count']

    def port_stats_in_handler(self, dpid, ports):
        print      "Port      stats      in      from      datapath",
longlong_to_octstr(dpid)[6:]
        for item in ports:
            print '\t',item['port_no'],':',item['tx_packets']

    def install(self):
        self.register_for_datapath_join(lambda    dpid,    stats    :
self.datapath_join_callback(dpid,stats))
        self.register_for_table_stats_in(self.table_stats_in_handler)
        self.register_for_port_stats_in(self.port_stats_in_handler)

self.register_for_aggregate_stats_in(self.aggregate_stats_in_handler)

    def getInterface(self):
        return str(Monitor)


def getFactory():
    class Factory:
        def instance(self, ctxt):
            return Monitor(ctxt)

    return Factory()
```

- Η εφαρμογή monitor χρησιμεύει για την παρακολούθηση των στατιστικών των διαφόρων διόδων δεδομένων, στις οποίες εφαρμόζονται οι ροές και η κίνηση πακέτων στο δίκτυο. Η monitor

εμφανίζει στην οθόνη του ελεγκτή τα στατιστικά στοιχεία τριών κατηγοριών: των πινάκων ροής (Table stats), των θυρών (Port stats) και των συγκεντρωτικών στοιχειών (Aggregate stats) για κάθε δίοδο δεδομένων όπως εμφανίζεται πιο κάτω:

```
00299|openflow-event|DBG:received stats reply from 0023205dd18d
Table stats in from datapath 00:23:20:5d:d1:8d
    nf2 : 140454020513792
    hash2 : 0
    linear : 0
00300|openflow-event|DBG:received stats reply from 00232050effd
Port stats in from datapath 00:23:20:50:ef:fd
    1 : 5187
    2 : 5187
    3 : 5314
    4 : 0
    65534 : 5187
00301|openflow-event|DBG:received stats reply from 00232050effd
Table stats in from datapath 00:23:20:50:ef:fd
    nf2 : 140454020513792
    hash2 : 0
    linear : 0
00302|openflow-event|DBG:received stats reply from 00232050effd
Table stats in from datapath 00:23:20:50:ef:fd
    nf2 : 140454020513792
    hash2 : 0
    linear : 0
00303|openflow-event|DBG:received stats reply from 00232050effd
Aggregate stats in from datapath 00:23:20:50:ef:fd
    {'packet_count': 0L, 'byte_count': 0L, 'flow_count': 0L}
00304|openflow-event|DBG:received stats reply from 00232050effd
Aggregate stats in from datapath 00:23:20:50:ef:fd
    {'packet_count': 0L, 'byte_count': 0L, 'flow_count': 0L}
```

- Η εφαρμογή monitor επιτυγχάνει την παρακολούθηση κάθε διόδου δεδομένων (datapath) που ενώνεται στον ελεγκτή χρησιμοποιώντας την διεπαφή self.post_callback, η οποία καλείται για κάθε ομάδα παραμέτρων. Κατά την κλήση η εκτύπωση των στατιστικών στοιχειών γίνεται απλά καλώντας τις ρουτίνες εκτύπωσης:

```
def aggregate_stats_in_handler(self, dpid, stats):
        print     "Aggregate     stats     in     from     datapath",
longlong_to_octstr(dpid)[6:]
        print '\t',stats

    def table_stats_in_handler(self, dpid, tables):
        print      "Table      stats      in      from      datapath",
longlong_to_octstr(dpid)[6:]
        for item in tables:
            print '\t',item['name'],':',item['active_count']
```

```
    def port_stats_in_handler(self, dpid, ports):
        print      "Port      stats      in      from      datapath",
longlong_to_octstr(dpid)[6:]
        for item in ports:
            print '\t',item['port_no'],':',item['tx_packets']
```

ΚΕΦΑΛΑΙΟ 5

5.1 Η ανάπτυξη του application-aware μηχανισμού δρομολόγησης και η τοπολογία του OpenFlow δικτύου

Η application-aware εφαρμογή που αναπτύχθηκε στα πλαίσια αυτής της διπλωματικής εργασίας είναι βασισμένη στο μοντέλο εφαρμογών και τις προγραμματιστικές διεπαφές (API's) του NOX, όπως αυτές περιγράφονται στο προηγούμενο κεφάλαιο.

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε μια από τις βασικές εφαρμογές του ελεγκτή NOX ως βασικός σκελετός (framework) για την δημιουργία επιπλέον χαρακτηριστικών και λειτουργιών. Η εφαρμογή αυτή είναι η pyswitch.py, η οποία είναι και η προτεινόμενη εφαρμογή από τους κατασκευαστές του NOX, για επέκταση των παρεχόμενων λειτουργιών του ελεγκτή και εξατομίκευση των δυνατοτήτων που προσφέρει.

Ενδεικτικά η παρούσα application-aware εφαρμογή παρέχει την δυνατότητα ελέγχου της κίνησης και της προτεραιότητας που δίνεται σε πακέτα OpenFlow που διακινούνται στο δίκτυο. Η προτεραιότητα αυτή μπορεί να καθορίζεται από την διεύθυνση IP του προορισμού σε ανώτερο επίπεδο δρομολόγησης (επίπεδο 3 του OSI) και από την διεύθυνση MAC αποστολέα και προορισμού (επίπεδο 2 του OSI) σε περίπτωση που δεν υπάρχουν επαρκείς πληροφορίες από τα ληφθέντα πακέτα για IP δρομολόγηση. Η λειτουργία αυτή παρέχεται με χρήση διαφόρων μηχανισμών που παρέχει το πρωτόκολλο OpenFlow και μέσω της εξατομικευμένης εφαρμογής ctrlsw.py που αναπτύχθηκε βασισμένη στην pyswitch.py.

Ο ελεγκτής NOX τρέχει την εφαρμογή αυτή στο OpenFlow δίκτυο που έχει στηθεί στο εργαστήριο Δικτύων Υπολογιστών της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών. Συνοπτικά το δίκτυο απαρτίζεται από ένα σύνολο μεταγωγέων (switches), τον κεντρικό ελεγκτή NOX (NOX controller) και τερματικές συσκευές (clients). Να σημειωθεί ότι οι συσκευές του δικτύου διαθέτουν όλες NetFPGA πλακέτες (boards) με αποτέλεσμα να μπορούν να λειτουργήσουν και ως μεταγωγείς αλλά και ως τερματικά, αντίστοιχα με το λογισμικό που διαθέτουν εγκατεστημένο και εν ενεργεία. Η κίνηση δημιουργείται από τις τερματικές

συσκευές και την προσπάθεια επικοινωνία τους με χρήση του πρωτοκόλλου OpenFlow και του ελεγκτή NOX, αντί των καθιερωμένων πρωτοκόλλων όπως το TCP/IP.

5.2 Η εφαρμογή pyswitch.py του ελεγκτή NOX

Η εφαρμογή pyswitch.py είναι μέρος των βασικών εφαρμογών (core apps) του ελεγκτή NOX στην θέση src/nox/coreapps/examples όπως παρέχεται στην προγραμματιστική έκδοση του NOX. Όπως φαίνεται και από την ονομασία, η εφαρμογή αυτή είναι γραμμένη σε γλώσσα Python και βάση του προγραμματιστικού μοντέλου του ελεγκτή. Οι δύο πιο σημαντικές κλάσεις της pyswitch είναι η `do_l2_learning(dpid, inport, packet)` και η `forward_l2_packet(dpid, inport, packet, buf, bufid)`, που αναλαμβάνουν την εκμάθηση των MAC των συσκευών του δικτύου και την κατασκευή και εγκατάσταση ροών. Και οι δύο αυτές κλάσεις αναλύονται με λεπτομέρεια στα επόμενα κεφάλαια. Ο πλήρης κώδικας της pyswitch, επισυνάπτεται στο παράρτημα Β-1 της παρούσας διπλωματικής εργασίας.

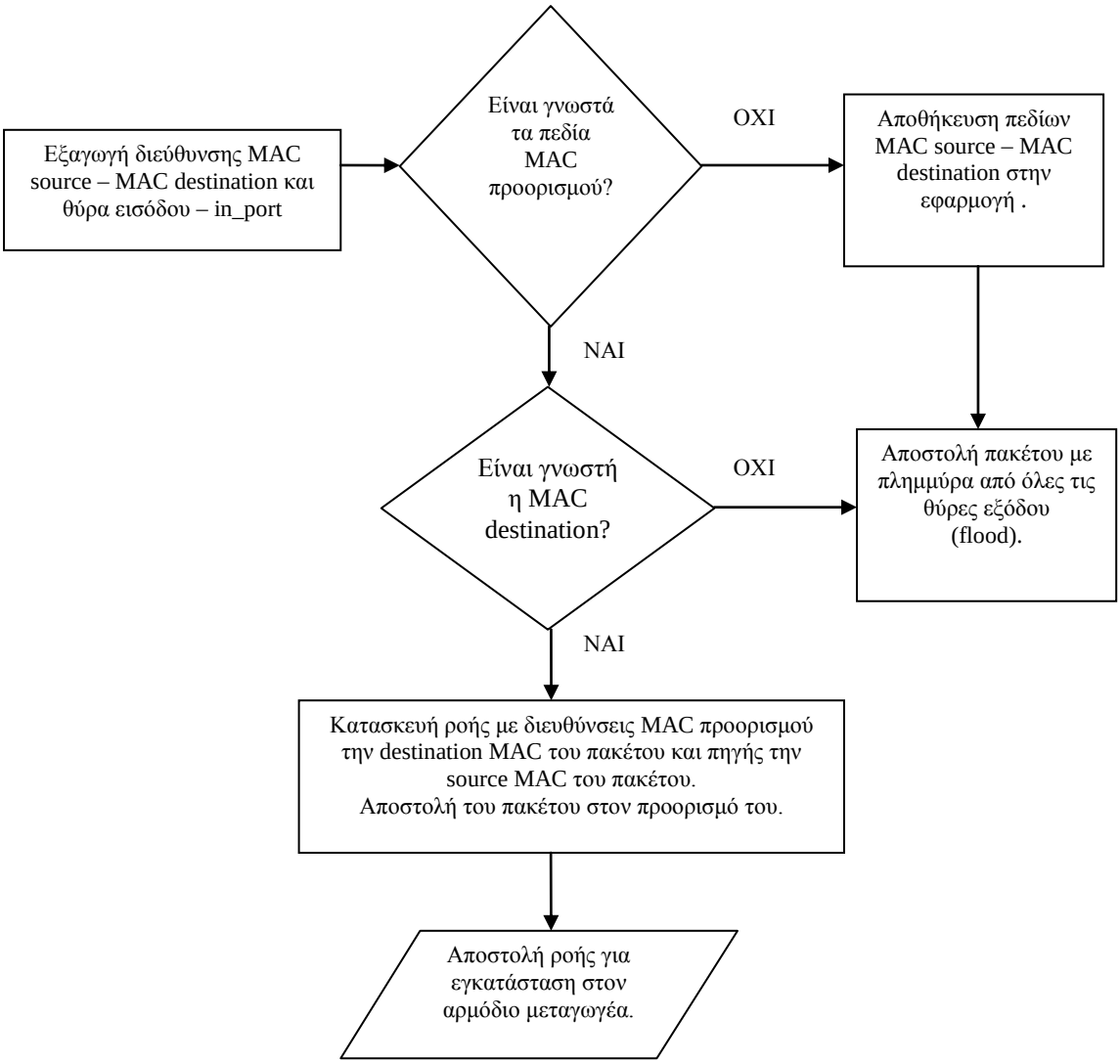
Η pyswitch.py λειτουργεί στην βασική της μορφή ως μονάδα ελέγχου δρομολογητών στο επίπεδο 2 (διευθύνσεις MAC) του μοντέλου OSI (Open Systems Interconnect). Αργότερα, στα πλαίσια της διπλωματικής εργασίας, επεκτείνεται αυτή η λειτουργικότητα και η pyswitch.py μπορεί να παρέχει δυνατότητα δρομολόγησης στο επίπεδο 3, με διευθύνσεις IP. Αυτή η μονάδα ελέγχου θέτει υπό την διαχείριση της όλους τους υποκείμενους μεταγωγείς του OpenFlow δικτύου που είναι ενωμένοι με τον ελεγκτή NOX. Η pyswitch.py διατηρεί ένα αρχείο με όλες τις MAC διευθύνσεις των μεταγωγέων του δικτύου και κάθε φορά που ενώνεται ένας νέος μεταγωγέας με τον ελεγκτή, αποθηκεύει την MAC διεύθυνση του στην μνήμη της.

Επιπρόσθετα με την εκμάθηση και παρακολούθηση των MAC διευθύνσεων των μεταγωγέων, η pyswitch εγκαθιστά ροές στους μεταγωγείς αυτούς ανάλογα με τα πακέτα OpenFlow που διακινούνται στο δίκτυο. Η διαδικασία που ακολουθείται για την εγκατάσταση των ροών συνοψίζεται στα εξής βήματα:

1. Καταγραφή διεύθυνσης MAC και θύρας (MAC address και packet port) για κάθε πακέτο OpenFlow που παραλαμβάνει ο ελεγκτής από τους μεταγωγείς και αποθήκευση των διευθύνσεων αυτών.
2. Έλεγχος αν η διεύθυνση προορισμού (destination MAC) ενός πακέτου OpenFlow που λαμβάνει ο ελεγκτής ταυτίζεται με τις ήδη αποθηκευμένες.

3. Εάν επιτύχει ταύτιση, κατασκευάζεται η ροή με διευθύνσεις MAC προορισμού την destination MAC του πακέτου και πηγής την source MAC του πακέτου.
4. Εγκατάσταση της ροής στον μεταγωγέα που απέστειλε το πακέτο.
5. Αποστολή του εκκρεμούς πακέτου που βρίσκεται στον ελεγκτή στην διεύθυνση προορισμού του.

Το διάγραμμα ροής (flowchart) έχει ως ακολούθως:



Διάγραμμα 5.2 – Διάγραμμα ροής (flowchart) της pyswitch.py

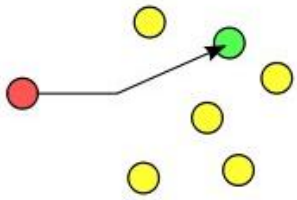
Σε περίπτωση που δεν υπάρχει ταύτιση σε καμία από τις αποθηκευμένες διευθύνσεις MAC στον ελεγκτή, τότε το πακέτο αποστέλλεται με την μέθοδο της πλημμύρας (flood) από όλες τις θύρες του ελεγκτή που είναι συνδεδεμένες με τους μεταγωγείς του δικτύου, πλην της θύρας εισόδου από την οποία προήλθε το πακέτο. Με αυτό τον τρόπο το πακέτο μπορεί να γίνει flood και από τους μεταγωγείς του δικτύου μέχρι να φτάσει στον προορισμό του, ο οποίος για την

δεδομένη στιγμή είναι άγνωστος στον ελεγκτή. Φυσικά οι διευθύνσεις MAC πηγής του πακέτου αποθηκεύονται σύμφωνα με την ανωτέρω διαδικασία, για προσπάθεια ταύτισης και δημιουργίας ροής στο μέλλον.

5.2.1 Διάκριση μεταξύ τύπων αποστολής πακέτων

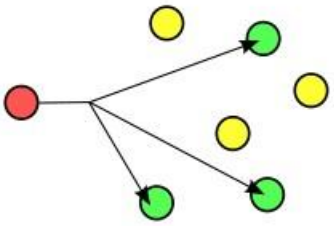
Είναι σημαντικό να γίνεται διάκριση μεταξύ των τύπων αποστολής πακέτου, δηλαδή να λαμβάνουμε υπόψη αν το πακέτο αποστέλλεται σε ένα παραλήπτη, σε πολλαπλούς παραλήπτες ή σε όλο το δίκτυο. Τα πιο κάτω σχεδιαγράμματα απεικονίζουν τους τρόπους αποστολής:

1. Δρομολόγηση από ένα αποστολέα σε ένα παραλήπτη – UNICAST (Μονοεκπομπή)



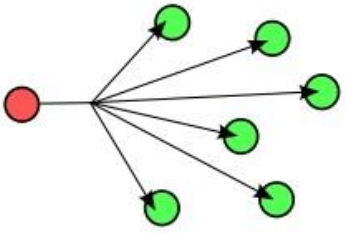
Εικόνα 5.2-1 - Δρομολόγηση Μονοεκπομπής

2. Δρομολόγηση από ένα αποστολέα σε πολλούς παραλήπτες – MULTICAST (Πολυεκπομπή)



Εικόνα 5.2-2 - Δρομολόγηση Πολυεκπομπής

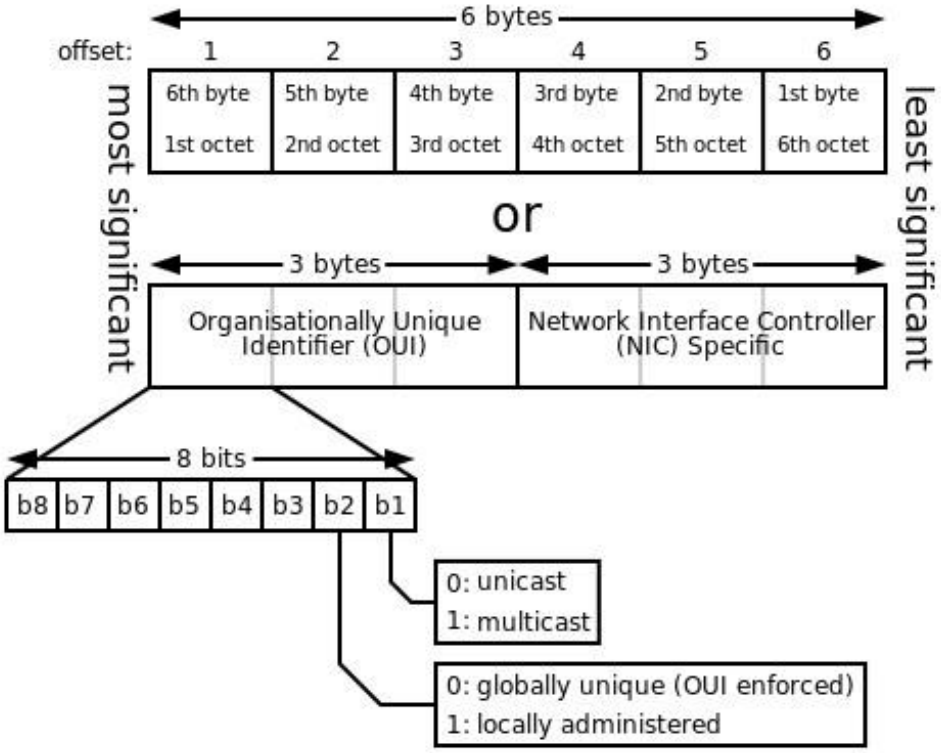
3. Δρομολόγηση από ένα αποστολέα σε όλο το δίκτυο – BROADCAST (Μετάδοση)



Εικόνα 5.2-3 – Δρομολόγηση Μετάδοσης

Η δημιουργία ροών αποδίδει τα μέγιστα σε περιπτώσεις μονής διόδευσης, όπως είναι η Unicast. Για περιπτώσεις διόδευσης πακέτων με πολλούς παραλήπτες όπως είναι η Multicast, μπορούν επίσης να χρησιμοποιηθούν ροές, όμως απαιτείται η δημιουργία πολλαπλών ροών, ίσων με τον αριθμό των διευθύνσεων προορισμού του πακέτου. Τέλος για περιπτώσεις πακέτων Broadcast που απευθύνονται σε όλο το δίκτυο, όπως είναι τα πακέτα ανακάλυψης δικτύου (network discovery) με χρήση μεθόδων όπως η gratuitous ARP, δεν υπάρχει καμιά ουσία στην κατασκευή ροών.

Για να γίνει η διάκριση μεταξύ αυτών των τύπων αποστολής χρησιμοποιείται η διεύθυνση προορισμού MAC, αφού το ίδιο το πρότυπο IEEE 802 προσφέρει αυτές τις πληροφορίες σύμφωνα με το πιο κάτω διάγραμμα:



Εικόνα 5.2-4 - Σχηματικό της MAC διευθυνσιοδότησης

Έτσι, σύμφωνα με το τελευταίο bit της 1^{ης} οκτάδας της MAC διεύθυνσης μπορεί να διαπιστωθεί αν το πακέτο είναι μονής διόδευσης εφόσον το bit αυτό είναι 0. Στην pyswitch ο έλεγχος αυτός πραγματοποιείται στις εξής γραμμές κώδικα :

```
    dstaddr = packet.dst.tostring()
if not ord(dstaddr[0]) & 1 and inst.st[dpid].has_key(dstaddr):
        prt = inst.st[dpid][dstaddr]
```

Η συνάρτηση ord() επιστρέφει το έκτο byte (δηλαδή την πρώτη οκτάδα) της διεύθυνσης προορισμού (`dstaddr[0]`) και ελέγχει αν είναι το 0, με συνδυασμό της Boolean λογικής και της πράξης and με το 1. Ακολούθως γίνεται έλεγχος αν η

διεύθυνση MAC είναι αποθηκευμένη στην μνήμη της εφαρμογής και συνεχίζεται ανάλογα με τα αποτελέσματα η επεξεργασία του πακέτου.

5.2.2 Η κλάση εκμάθησης διευθύνσεων συσκευών

Η κλάση `do_l2_learning(dpid, inport, packet)` είναι υπεύθυνη για την εξαγωγή και αποθήκευση των έγκυρων διευθύνσεων MAC των OpenFlow πακέτων που λαμβάνει ο ελεγκτής NOX :

```
def do_l2_learning(dpid, inport, packet):
    global inst

    # learn MAC on incoming port
    srcaddr = packet.src.tostring()
   if ord(srcaddr[0]) & 1:
        return
   if inst.st[dpid].has_key(srcaddr):
        dst = inst.st[dpid][srcaddr]
        if dst[0] != inport:
            log.msg('MAC has moved from '+ str(dst) + 'to'
+str(inport), system = 'ctrlsw')
        else:
            return
   else:
        log.msg('learned MAC '+mac_to_str(packet.src)+' on %d %d'%
(dpid,inport), system="ctrlsw")

    # learn or update timestamp of entry
   inst.st[dpid][srcaddr] = (inport, time(), packet)

    # Replace any old entry for (switch,mac).
        mac = mac_to_int(packet.src)
```

Κατά την παραλαβή του πακέτου εξάγεται η διεύθυνση MAC από το πεδίο MAC source του πακέτου και μετατρέπεται σε string με την εντολή `srcaddr = packet.src.tostring()` και ακολούθως γίνεται έλεγχος αν πρόκειται για πακέτο μονοεκπομπής ή πολυεκπομπής / μετάδοσης, σύμφωνα με τα όσα εξηγούνται στο κεφάλαιο 5.2.1. Αν πρόκειται για πακέτο μονο-εκπομπής, αποθηκεύεται στην μνήμη της βάσης δεδομένων (`dst = inst.st[dpid][srcaddr]`) ή αν έχει αλλάξει θύρα εισόδου, οπότε ενημερώνεται η βάση δεδομένων με τα νέα στοιχεία. Τέλος, ανανεώνονται τα στοιχεία χρονικής εισόδου του πακέτου και τα στοιχεία διευθύνσεων MAC για την τρέχουσα κατάσταση του αρμόδιου, για το πακέτο, δρομολογητή.

5.2.3 Η κλάση δρομολόγησης γνωστών διευθύνσεων προορισμού

Η κλάση `def forward_l2_packet(dpid, inport, packet, buf, bufid)` αναλαμβάνει την κατασκευή και εγκατάσταση ροών για ληφθέντα πακέτα OpenFlow που έχουν εξαχθεί και επαληθευθεί οι διευθύνσεις προορισμού τους:

```
def forward_l2_packet(dpid, inport, packet, buf, bufid):

    dstaddr = packet.dst.tostring()

    if not ord(dstaddr[0]) & 1 and inst.st[dpid].has_key
            (dstaddr):
        prt = inst.st[dpid][dstaddr]
        if  prt[0] == inport:
            log.err('**warning** learned port = inport',
            system="ctrlsw")
            inst.send_openflow(dpid, bufid, buf,
                        openflow.OFPP_FLOOD, inport)
        else:
            # We know the outport, set up a flow
            log.msg('installing flow for ' + str(packet),
                  system="ctrlsw")

            flow = extract_flow(packet)
            flow[core.IN_PORT] = inport
            actions = [[openflow.OFPAT_OUTPUT, [0, prt[0]]]]

            inst.install_datapath_flow(dpid, flow,
      CACHE_TIMEOUT,openflow.OFP_FLOW_PERMANENT, actions,
            bufid, openflow.OFP_DEFAULT_PRIORITY,inport, buf)

    else:
        # haven't learned destination MAC. Flood
        inst.send_openflow(dpid,  bufid,  buf,  openflow.OFPP_FLOOD,
                              inport)
```

Αρχικά γίνεται εξαγωγή της διεύθυνσης MAC προορισμού του πακέτου με την `dstaddr = packet.dst.tostring()`. Ακολούθως, γίνεται έλεγχος αν πρόκειται για unicast πακέτο ή όχι και αν η MAC destination είναι αποθηκευμένη στην μνήμη των ήδη γνωστών διευθύνσεων του ελεγκτή (`inst.st[dpid].has_key(dstaddr)`). Αν αποτύχει ο έλεγχος τότε το πακέτο προωθείται σε όλες τις θύρες εξόδου (flood), πλην αυτής από την οποία παραλήφθηκε με την εντολή `inst.send_openflow(dpid, bufid, buf, openflow.OFPP_FLOOD,inport)`.

Αν ο έλεγχος είναι επιτυχής και πρόκειται για unicast πακέτο με γνωστή διεύθυνση προορισμού γίνεται ακόμη ένας έλεγχος για να διαπιστωθεί αν η θύρα

εξόδου του (prt[0]) είναι ίδια με την θύρα εισόδου του (inport) με την `if prt[0] == inport`. Αυτό γίνεται για την αποφυγή δημιουργίας βρόχων (loops) στο δίκτυο, αφού ένα πακέτο που θα λαμβάνεται από ένα μεταγωγέα θα επιστρέφει συνεχώς στην θύρα του μεταγωγέα από την οποία προήλθε, χωρίς να γίνεται κάποια δρομολόγηση σε άλλο προορισμό.

Τέλος, με την επιτυχή περάτωση όλων των ελέγχων δημιουργείται και εγκαθίσταται ροή στον μεταγωγέα που απέστειλε το πακέτο, εξάγοντας την διεύθυνση MAC προορισμού και την θύρα εισόδου με τις `flow = extract_flow(packet)` και `flow[core.IN_PORT] = inport`. Με την χρήση των actions δίδεται η εντολή για την εγκατάσταση της ροής για εξερχόμενα πακέτα από την θύρα εξόδου port[0] : `actions = [[openflow.OFPAT_OUTPUT, [0, prt[0]]]]` .Η εγκατάσταση της ροής γίνεται με την εντολή `inst.install_datapath_flow(dpid,flow,CACHE_TIMEOUT,openflow.OFP_FLOW_PERMANENT, actions, bufid, openflow.OFP_DEFAULT_PRIORITY,inport, buf)`.

5.2.4 Η κλάση `packet_in_callback(dpid, inport, reason, len, bufid, packet)`

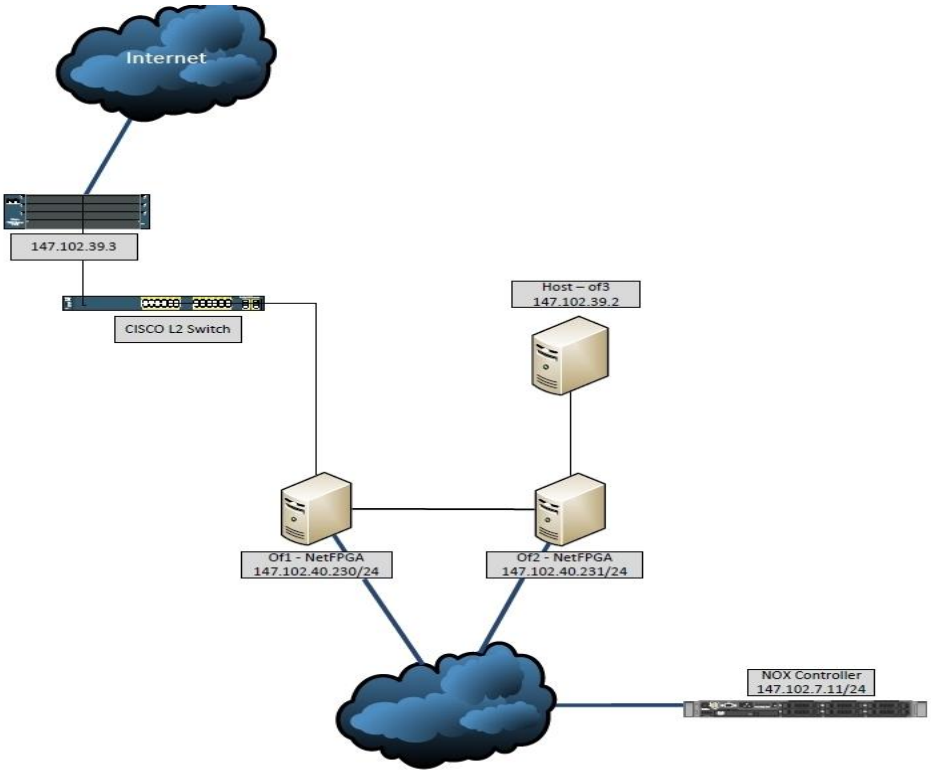
Η κλάση αυτή είναι υπεύθυνη για τον χειρισμό κάθε ληφθέντος πακέτου από τον ελεγκτή NOX. Δεν διαθέτει καμιά εξειδικευμένη λειτουργία, όπως οι προηγούμενες κλάσεις, αλλά αναλαμβάνει την καταχώρηση και εγγραφή των νέων δρομολογητών που ενώνονται στον ελεγκτή με την `inst.st[dpid] = {}`. Ο κώδικας είναι ο ακόλουθος:

```
def packet_in_callback(dpid, inport, reason, len, bufid, packet):

    if not packet.parsed:
        log.msg('Ignoring incomplete packet',system='pyswitch')

    if not inst.st.has_key(dpid):
        log.msg('registering new switch %x' % dpid,system='pyswitch')
        inst.st[dpid] = {}

    # don't forward lldp packets
    if packet.type == ethernet.LLDP_TYPE:
        return CONTINUE

    # learn MAC on incoming port
    do_l2_learning(dpid, inport, packet)

    forward_l2_packet(dpid, inport, packet, packet.arr, bufid)
    return CONTINUE
```

Η `packet_in_callback` καλεί τις άλλες δύο κλάσεις, `do_l2_learning` και `forward_l2_packet` για τον χειρισμό του πακέτου, αφού κάνει τον έλεγχο για νέο δρομολογητή και ανίχνευση αν το πακέτο είναι τύπου LLDP, οπότε και σταματά η επεξεργασία. Αυτό συμβαίνει διότι τα LLDP (Link Layer Discovery Protocol) πακέτα είναι μηχανισμοί ανίχνευσης και ανακάλυψης του δικτύου και δεν αφορούν καμιά λειτουργία μεταφοράς δεδομένων στο δίκτυο μεταξύ τερματικών.

5.3 Η τοπολογία του δικτύου OpenFlow

Το δίκτυο OpenFlow στο εργαστήριο Δικτύων Υπολογιστών αποτελείται από τον ελεγκτή NOX με IP address 147.102.7.11/24, δύο μεταγωγείς OpenFlow, τους of1 (147.102.40.230/24) και of2 (147.102.40.231/24) και δύο hosts, το of3 (147.102.39.2) και ένα Cisco router (147.102.39.3) :



Εικόνα 5.3-1 Το δίκτυο OpenFlow

Ο ελεγκτής NOX (147.102.7.11) είναι συνδεδεμένος μέσω του δικτύου του εργαστηρίου με τους δύο OpenFlow μεταγωγείς (of1 και of2). Ο μεταγωγέας of1 (147.102.40.230) είναι κατευθείαν συνδεδεμένος με τον Cisco δρομολογητή (147.102.40.200/24) που είναι και η πύλη Internet (Internet Gateway) του δικτύου. Επίσης, ο of1 είναι απευθείας συνδεδεμένος και με τον μεταγωγέα of2 (147.102.40.231), ο οποίος με τη σειρά του είναι συνδεδεμένος με το τερματικό of3 (147.102.39.2). Ο ελεγκτής NOX επικοινωνεί μέσω εξωτερικού καναλιού διαχείρισης εκτός ζώνης (out-of-band management channel) με τους μεταγωγείς του δικτύου για την επιτέλεση των διεργασιών του, μεταξύ αυτών και η ανάθεση ή μεταβολή ροών στους μεταγωγείς. Αυτό το κανάλι διαχείρισης μπορεί να ανατίθεται και σε διαφορετικά κανάλια στο δίκτυο, ακόμα και πάνω από SSL σύνδεση. Για πειραματικούς όμως λόγους δεν χρησιμοποιείται το SSL, ούτως ώστε

να μπορεί να γίνει ανίχνευση και ανάλυση των πακέτων επικοινωνίας που ανταλλάζει ο ελεγκτής με τους μεταγωγείς.

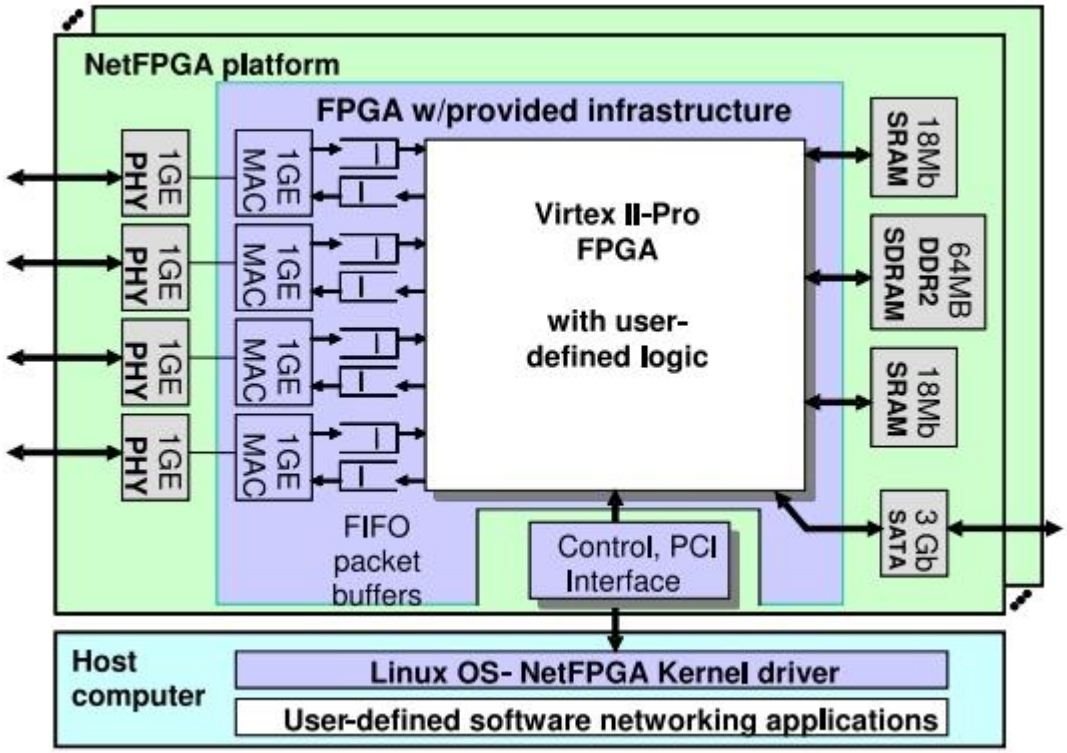
Οι δρομολογητές OpenFlow δημιουργήθηκαν με την χρήση του OpenFlow-NetFPGA (Network Field Programmable Gate Array) σχεδιασμού. Το OpenFlow-NetFPGA είναι ένας υποδειγματικός μεταγωγέας OpenFlow (reference switch) με επιτάχυνση υλικού μέσω του NetFPGA και του συνοδευτικού λογισμικού (ofdatapath με λειτουργίες χειρισμού από το NetFPGA).

Γενικά, το υλικό του NetFPGA χειρίζεται την προώθηση των πακέτων δεδομένων με την μετατροπή των επικεφαλίδων των λαμβανομένων πακέτων, ούτως ώστε να συμβαδίζουν με το πρωτόκολλο OpenFlow. Το NetFPGA διαθέτει ένα πίνακα ακριβούς ταύτισης, μεγέθους 32768 καταχωρίσεων και ένα πίνακα μη ακριβούς ταύτισης (με χρήση τυχαίων πεδίων – wildcards) 24 καταχωρίσεων.

Το συνοδευτικό λογισμικό του NetFPGA έχει τον ρόλο δημιουργίας διαδρομών δεδομένων OpenFlow,του ofdatapath. Οι κανονικοί μεταγωγείς αναφοράς OpenFlow έχουν ένα πίνακα κατακερματισμού (hash table) και ένα γραμμικό πίνακα (linear table) αλλά το λογισμικό διαθέτει επιπλέον και ένα “nf2” πίνακα. Ο nf2 είναι αντιγραφή των πινάκων του μεταγωγέα (κατακερματισμού και γραμμικού) και χρησιμεύει για την ακριβή (exact match) ή μερική (wildcarded) αντιστοίχιση των καταχωρήσεων βάση του πρωτοκόλλου OpenFlow. Κατά την λειτουργία του μεταγωγέα, οι εγγραφές γίνονται στον πίνακα nf2. Αν οι ενέργειες εγγραφής δεν υποστηρίζονται από το NetFPGA ή αν εξαντληθούν οι καταχωρίσεις του nf2, τότε οι καταχωρίσεις εγγράφονται στους πίνακες κατακερματισμού (ή γραμμικό) του υλικού.

Βάση λοιπόν της λειτουργίας του NetFPGA, για να δουλέψουν οι συσκευές of1 και of2 ως μεταγωγείς OpenFlow πρέπει πρώτα να αρχικοποιηθεί και να ενεργοποιηθεί η εφαρμογή του NetFPGA που είναι εγκατεστημένη στους δύο μεταγωγείς. Αυτό επιτυγχάνεται στον φάκελο εγκατάστασης του NetFPGA με τις εντολές που παρέχονται στο παράρτημα Γ. Με τις εντολές εκκίνησης ενεργοποιείται ο μεταγωγέας OpenFlow NetFPGA και συνδέεται στον ελεγκτή NOX που βρίσκεται στην διεύθυνση 147.102.7.11, δημιουργεί τους πίνακες δρομολόγησης nf2c και μπορεί πλέον να διαχειριστεί πακέτα κίνησης σε πρωτόκολλο OpenFlow. Για το κλείσιμο του μεταγωγέα OpenFlow χρησιμοποιείται η ανάλογη εντολή από το παράρτημα Γ of_stop , που κλείνει όλες τις ενεργές διεπαφές (active interfaces) του ελεγκτή και αποσυνδέεται από τον ελεγκτή NOX.

Ενδεικτικά, το σχηματικό ενός Virtex II-Pro 50 OpenFlow NetFPGA έχει την ακόλουθη μορφή:



Εικόνα 5.3-1 Σχηματικό Διάγραμμα OpenFlow NetFPGA

5.4 Η εξειδικευμένη application-aware εφαρμογή ctrlswl3.py και η εφαρμογή προτεραιότητας συσκευών ctrlsw.py

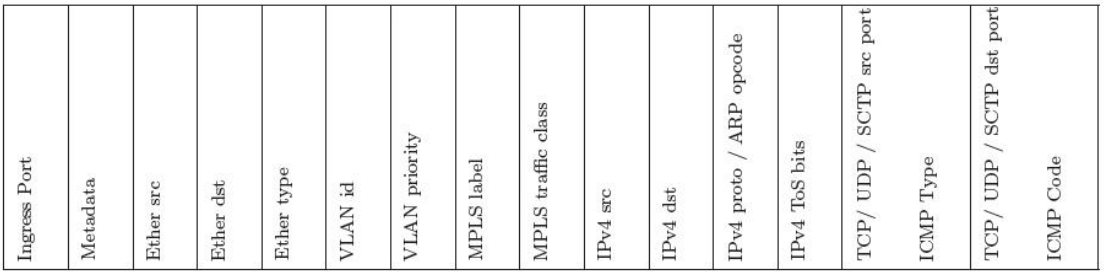
Η application-aware εφαρμογή ctrlswl3.py (**con**t**ro**l **s**oft**w**are) αναπτύχθηκε για την εγκατάσταση καταλλήλων ροών σε OpenFlow μεταγωγείς κατά την αποστολή πακέτων OpenFlow μιας προκαθορισμένης εφαρμογής, βάσει γνωστών θυρών προορισμού. Από την άλλη,η εφαρμογή προτεραιότητας συσκευών ctrlsw.py αναλαμβάνει την δρομολόγηση των πακέτων OpenFlow βάση σειράς προτεραιότητας, ανάλογα με την συσκευή που αποστέλλει τα πακέτα δεδομένων. Οι δύο αυτές εφαρμογές αποτελούν επέκταση και εξειδίκευση των δυνατοτήτων της εφαρμογής pyswitch.py που παρέχεται στην προγραμματιστική έκδοση του ελεγκτή NOX και χρησιμεύει σαν υπόβαθρο (framework) για την ανάπτυξη πιο εξειδικευμένων λειτουργιών, όπως έγινε με την δημιουργία της ctrlswl3.py. Η επέκταση αυτή γίνεται κυρίως με την προσθήκη δυνατοτήτων επεξεργασίας και ανάλυσης πακέτων στο επίπεδο 3 (TCP/IP) της application-aware εφαρμογής.

Η εφαρμογή αυτή είναι πλήρως γραμμένη σε γλώσσα Python και δομημένη σύμφωνα με το μοντέλο προγραμματισμού του NOX, με τις κύριες κλάσεις

λειτουργιών να βρίσκονται στην αρχή του κώδικα και την δημιουργία και τρέξιμο της εφαρμογής στο τέλος του κώδικα.

Στο παράρτημα Β-2 και Β-3 παρέχεται ο πηγαίος κώδικας της τελικής εφαρμογής ctrlswl3.py και ctrlsw.py αντίστοιχα, με όλες τις προσθήκες στα βασικά αρχεία του NOX που απαιτούνται για το επιτυχές τρέξιμο της εφαρμογής στην πλατφόρμα του ελεγκτή.

Η application-aware εφαρμογή ctrlswl3.py βασίζει την λειτουργία της στην εξαγωγή και έλεγχο γνωστών θυρών επικοινωνίας εφαρμογών που δραστηριοποιούνται σε ένα δίκτυο υπολογιστών. Πιο συγκεκριμένα η εφαρμογή βασίζεται στην ανάλυση της TCP ή UDP θύρας προορισμού που διαθέτει ένα πακέτο OpenFlow. Για καλύτερη κατανόηση των πεδίων αυτών παρουσιάζεται η πιο λεπτομερής μορφή των πεδίων ενός πακέτου OpenFlow πιο κάτω:



Εικόνα 5.4-1 – Λεπτομερής απεικόνιση πεδίων ενός πακέτου OpenFlow

Με την αναγνώριση του πεδίου TCP/UDP/SCTP destination port, η ctrlswl3.py μπορεί να αναγνωρίσει την εφαρμογή που αποστέλλει το πακέτο και να εγκαταστήσει την κατάλληλη ροή. Στα πλαίσια της εργασίας αυτής, η ctrlswl3.py ελέγχει για εφαρμογές HTTP (http – hypertext transfer protocol) που επικοινωνούν μέσω της θύρας 80 σε επίπεδο TCP. Έτσι, όποτε η ctrlswl3.py αναγνωρίσει την υπηρεσία http, εγκαθιστά κατάλληλες ροές στους μεταγωγείς που χρησιμοποιούνται από την εφαρμογή http και συνεπώς στέλνουν τα αντίστοιχα πακέτα OpenFlow. Με αυτό τον τρόπο δημιουργείται μια απευθείας δίοδος δεδομένων (datapath) για την εφαρμογή http στο δίκτυο, χωρίς να παρεμβάλλεται ο ελεγκτής, με αποτέλεσμα να επιταχύνεται και να βελτιστοποιείται η επικοινωνία δύο συσκευών μέσω της υπηρεσίας http.

Η application-aware ctrlswl3.py μπορεί να εγκαταστήσει ροές και για άλλες εφαρμογές που χρησιμοποιούν το δίκτυο, είτε πρόκειται για εφαρμογές μεταφοράς αρχείων (FTP – file transfer protocol), επικοινωνίας (IRC – internet relay chat) κ.ο.κ. Πιο παραστατικά, παρέχεται ο κάτωθι πίνακας των πλέον γνωστών θυρών TCP/UDP ανάλογα με την εφαρμογή που χρησιμοποιεί ένα δίκτυο υπολογιστών:

Port Number	Description
1	TCP Port Service Multiplexer (TCPMUX)
5	Remote Job Entry (RJE)
7	ECHO
18	Message Send Protocol (MSP)
20	FTP -- Data
21	FTP -- Control
22	SSH Remote Login Protocol
23	Telnet
25	Simple Mail Transfer Protocol (SMTP)
29	MSG ICP
37	Time
42	Host Name Server (Nameserv)
43	WhoIs
49	Login Host Protocol (Login)
53	Domain Name System (DNS)
69	Trivial File Transfer Protocol (TFTP)
70	Gopher Services
79	Finger
80	HTTP
103	X.400 Standard
108	SNA Gateway Access Server
109	POP2
110	POP3
115	Simple File Transfer Protocol (SFTP)
118	SQL Services
119	Newsgroup (NNTP)
137	NetBIOS Name Service
139	NetBIOS Datagram Service
143	Interim Mail Access Protocol (IMAP)
150	NetBIOS Session Service
156	SQL Server
161	SNMP
179	Border Gateway Protocol (BGP)
190	Gateway Access Control Protocol (GACP)
194	Internet Relay Chat (IRC)

Πίνακας 5.4-2 – Γνωστές θύρες TCP/UDP

Για την δημιουργία ροών διαφορετικής προτεραιότητας, δημιουργήθηκε επίσης και η εφαρμογή ctrlsw.py, που δημιουργεί ροές με διαφορετική προτεραιότητα για κάθε συσκευή που ενώνεται στο δίκτυο, ανάλογα με την διεύθυνση MAC της συσκευής. Με την χρήση αυτης της εφαρμογής μπορεί να ασκηθεί πλήρης έλεγχος σε ένα δίκτυο, ελέγχοντας τις εφαρμογές με την χρήση της application-aware ctrlswl3.py ή εναλλακτικά της device-aware ctrlsw.py.

Η ctrlsw.py παραλαμβάνει και χειρίζεται τα πακέτα OpenFlow για τα οποία δεν υπάρχει καμιά καταχώριση ροής στους μεταγωγείς του δικτύου. Τα πακέτα αυτά ελέγχονται ως προς τον τύπο τους (unicast, multicast και broadcast) και αν κατευθύνονται σε γνωστό προορισμό, δηλαδή διαθέτουν γνωστή διεύθυνση προορισμού MAC στον ελεγκτή, τότε εγκαθίσταται ροή στον μεταγωγέα που απέστειλε το πακέτο, για μελλοντικό χειρισμό παρομοίων πακέτων, ενώ το πακέτο στον ελεγκτή αποστέλλεται στον προορισμό του.

Η διαφοροποίηση της εφαρμογής ctrlsw.py από την ctrlswl3.py έγκειται στην διαχείριση συσκευών αντί εφαρμογών στο δίκτυο. Αυτό επιτυγχάνεται με την δημιουργία τριών επιπέδων προτεραιότητας για τα πακέτα με γνωστούς προορισμούς. Η διάκριση αυτή επιτυγχάνεται με την κατασκευή δύο λιστών αποθήκευσης διευθύνσεων στην εφαρμογή. Οι λίστες αυτές ορίζονται στις γραμμές κώδικα 24-25 του παραρτήματος Β-3 και είναι:

1. Λίστα υψηλής προτεραιότητας : macs_str_hp
2. Λίστα χαμηλής προτεραιότητας : macs_str_lp

Με την χρήση των λιστών αυτών, η ctrlsw.py είναι σε θέση να εντοπίσει τους προορισμούς υψηλής, χαμηλής και κανονικής προτεραιότητας και να εγκαταστήσει τις κατάλληλες ροές στους μεταγωγείς του δικτύου. Είναι σημαντικό να τονιστεί ότι οι προορισμοί υψηλής και χαμηλής προτεραιότητας πρέπει να τίθενται από τον διαχειριστή του δικτύου πριν από την εκκίνηση της εφαρμογής. Οι λίστες macs_str_hp και macs_str_lp δηλώνονται στις γραμμές κώδικα 24 και 25, όπου στον παρών κώδικα δεν δηλώνονται οποιοιδήποτε προορισμοί, απλά παρέχεται ως παράδειγμα το `'xx:xx:xx:xx:xx:xx'` που αντιπροσωπεύει μια αλφαριθμητική ακολουθία MAC (MAC string).

Με την χρήση αυτών των λιστών, ο διαχειριστής του δικτύου μπορεί να αναθέτει προορισμούς με υψηλή προτεραιότητα και απαιτήσεις χαμηλής καθυστέρησης στην λίστα macs_str_hp, γράφοντας την κατάλληλη MAC διεύθυνση της συσκευής που τρέχει την εφαρμογή υψηλών απαιτήσεων. Αντίστοιχα, για συσκευές που τρέχουν μη-κρίσιμες εφαρμογές ο διαχειριστής μπορεί να καταχωρίσει την MAC διεύθυνση τους στην λίστα χαμηλής προτεραιότητας, macs_str_lp. Για οποιεσδήποτε άλλες συσκευές-τερματικά που χρησιμοποιούν το

δίκτυο, η ctrlsw.py αναθέτει μια εξ' ορισμού κανονική προτεραιότητα (default) όπως παρέχεται τυπικά από το πρωτόκολλο OpenFlow.

Οι καταστάσεις υψηλής, χαμηλής και κανονικής προτεραιότητας μπορούν να οριστούν με χρήση δύο συγκεκριμένων λειτουργιών που παρέχει το πρωτόκολλο OpenFlow. Η πρώτη λειτουργία είναι η διάρκεια παραμονής (idle_timeout) μιας καταχώρισης ροής στον πίνακα ενός μεταγωγέα κατά την οποία η ροή αυτή βρίσκεται σε αναμονή, μέχρι κάποιο πακέτο να αντιστοιχηθεί με τις παραμέτρους της ροής. Η διάρκεια αυτή διακρίνεται σε δύο επίπεδα στην ctrlsw.py, την CACHE_TIMEOUT που διαρκεί έξι (6) δευτερόλεπτα και την CACHE_PERMFLOW που διαρκεί εξήντα (60) δευτερόλεπτα. Για ροές με υψηλή προτεραιότητα, κατά την εγκατάσταση ροής με την εντολή `inst.install_datapath_flow`, παρέχεται ως παράμετρος αναμονής η CACHE_PERMFLOW που κρατάει ενεργή την ροή για ένα λεπτό, ενώ για χαμηλή και κανονική προτεραιότητα δίδεται ως χρόνος αναμονής η CACHE_TIMEOUT.

Η δεύτερη λειτουργία που παρέχεται είναι η προτεραιότητα πίνακα που παρέχει το OpenFlow. Η τιμή αυτή αντιπροσωπεύει την προτεραιότητα στην αντιστοίχιση της ροής στον πίνακα ροών του μεταγωγέα με ένα πακέτο OpenFlow κατά την λήψη του από τον μεταγωγέα. Έτσι το πακέτο πρώτα ελέγχεται αν αντιστοιχίζεται επιτυχώς με ροές υψηλής τιμής προτεραιότητας προτού η διαδικασία αντιστοίχισης προχωρήσει σε ροές με χαμηλότερη τιμή προτεραιότητας. Το πρωτόκολλο OpenFlow υποστηρίζει τιμές προτεραιότητας από 0 – 65535. Οπότε, για ροές υψηλής προτεραιότητας δίνεται η μέγιστη τιμή προτεραιότητας (65535), για χαμηλή προτεραιότητα μια μοναδιαία τιμή (1) και για ροές κανονικής προτεραιότητας δίδεται η προεπιλεγμένη (DEFAULT) τιμή (32768) μέσω της χρήσης της εντολής εγκατάστασης ροής `inst.install_datapath_flow` και της παραμέτρου `openflow.OFP_DEFAULT_PRIORITY` . Για την ανάθεση μέγιστης προτεραιότητας προστίθεται η τιμή 32767 στην `OFP_DEFAULT_PRIORITY`,όπως φαίνεται στην γραμμή 131 του αντίστοιχου κώδικα στο παράρτημα Β-3 και αντίστοιχα για χαμηλή προτεραιότητα αφαιρείται η τιμή 32767 (γραμμή 137).

5.4.1 Η κλάση create_l3_out_flow της ctrlswl3.py

Η κλάση create_l3_out_flow είναι υπεύθυνη για την κατασκευή της ροής και πιο συγκεκριμένα την δημιουργία των πεδίων προορισμού και προέλευσης, βάση της εξαγωγής που γίνεται από τα πεδία του πακέτου που παραλαμβάνει ο ελεγκτής από τους μεταγωγείς.

```
032 def create_l3_out_flow(ethernet):
033     attrs = {}
```

```
034     attrs[core.DL_DST] = ethernet.dst
035     attrs[core.DL_SRC] = ethernet.src
036     attrs[core.DL_TYPE] = ethernet.type
037     p = ethernet.next
038 
039     if isinstance(p, vlan):
040          attrs[core.DL_VLAN] = p.id
041          attrs[core.DL_VLAN_PCP] = p.pcp
042          p = p.next
043     else:
044          attrs[core.DL_VLAN] = 0xffff
045          attrs[core.DL_VLAN_PCP] = 0
046 
047     if isinstance(p, ipv4):
048          attrs[core.NW_SRC] = p.srcip
049          attrs[core.NW_DST] = p.dstip
050          attrs[core.NW_PROTO] = p.protocol
051          p = p.next
052 
053         if isinstance(p, udp) or isinstance(p, tcp):
054              attrs[core.TP_SRC] = p.srcport
055              attrs[core.TP_DST] = p.dstport
056         else:
057              if isinstance(p, icmp):
058                  attrs[core.TP_SRC] = p.type
059                  attrs[core.TP_DST] = p.code
060              else:
061                  attrs[core.TP_SRC] = 0
062                  attrs[core.TP_DST] = 0
063     else:
064          attrs[core.NW_SRC] = 0
065          attrs[core.NW_DST] = 0
066          attrs[core.NW_PROTO] = 0
067          attrs[core.TP_SRC] = 0
068          attrs[core.TP_DST] = 0
069 
070     return attrs
```

Η κλάση αποδίδει τις παραμέτρους (attributes) της ροής, που όπως προαναφέρθηκε είναι τα πεδία προορισμού και προέλευσης, εξάγοντας τα από το ληφθέν πακέτο που παίρνει ως είσοδο (Ethernet). Αρχικά αποδίδονται οι διευθύνσεις MAC προορισμού και προέλευσης, σύμφωνα με τις εντολές `attrs[core.DL_DST] = ethernet.dst`, `attrs[core.DL_SRC] = ethernet.src` . Μετά γίνεται αναζήτηση αν υπάρχει ετικέτα VLAN (virtual local area network) και ακολούθως γίνεται έλεγχος αν το πακέτο διαθέτει διευθύνσεις προορισμού και προέλευσης στο επίπεδο 3 βάσει OSI, δηλαδή διευθύνσεις IP που αποδίδονται στην ροή με τις εντολές `attrs[core.NW_SRC] = p.srcip` και `attrs[core.NW_DST] = p.dstip`. Ακολούθως, γίνεται αναζήτηση για πεδία TCP και UDP στο πακέτο, καθώς και ICMP. Σε αυτά τα πεδία βρίσκονται και οι θύρες προορισμού του πακέτου που ενδιαφέρει την εφαρμογή ctrlswl3.py. Αν αποτύχει η αναζήτηση σε όλα τα πεδία ανωτέρου επιπέδου από το επίπεδο 2 (MAC), τότε η κλάση επιστρέφει μόνο τις διευθύνσεις MAC προορισμού και προέλευσης του πακέτου, αφού το πακέτο δεν διαθέτει άλλες πληροφορίες.

Να σημειωθεί ότι η αναζήτηση γίνεται με την χρήση του δείκτη p, ο οποίος εμφανίζεται ανωτέρω ως `p = ethernet.next` και χρησιμεύει ως ένδειξη για τα πεδία που περιλαμβάνει το πακέτο OpenFlow. Με την χρήση του `p.next` επιτυγχάνεται η σάρωση σε όλα τα πεδία διευθυνσιοδότησης που διαθέτει το πακέτο.

5.4.2 Η κλάση forward_l3_packet της ctrlswl3.py

Η κλάση forward_l3_packet είναι επέκταση της αντίστοιχης κλάσης που διαθέτει η pyswitch.py και πλέον αναλαμβάνει ένα πιο εξειδικευμένο ρόλο, την κατασκευή ροών μετά την αναγνώριση της σωστής θύρας για τις εφαρμογές που διαχειρίζεται η ctrlswl3.py. Στην εργασία αυτή γίνεται χειρισμός και ανάλυση της TCP θύρας 80, που είναι αρμόδια για την υπηρεσία HTTP. Έτσι, με την εντοπισμό της θύρας αυτής από την εφαρμογή γίνεται δημιουργία και εγκατάσταση της κατάλληλης ροής, σύμφωνα με το τμήμα κώδικα που παρουσιάζεται πιο κάτω:

```
0121 def forward_l3_packet(dpid, inport, packet, buf, bufid):
0122   dstaddr = packet.dst.tostring()
0123   pt=packet.next
0124
0125   if not ord(dstaddr[0]) & 1 and inst.st[dpid].has_key(dstaddr):
0126         prt = inst.st[dpid][dstaddr]
0127
0128
0129
0130   if isinstance(pt, ipv4):
0131         pt=pt.next
0132    if isinstance(p, udp) or isinstance(p, tcp):
0133           appsrcprt = pt.srcport
0134           appdstprt = pt.dstport
0135    else:
0136      print 'No valid TCP/UDP protocol (http) detected.
Flooding..'
0137           inst.send_openflow(dpid, bufid, buf,
openflow.OFPP_FLOOD, inport)
0138           return
0139         if prt[0] == inport:
0140            log.err('**Warning** learned port = inport',
system="ctrlswl3")
0141             inst.send_openflow(dpid, bufid, buf,
openflow.OFPP_FLOOD, inport)
0142             return
0143         else:
0144            flow = create_l3_out_flow(packet)
0145            flow[core.IN_PORT] = inport
0146            # actions = [[openflow.OFPAT_OUTPUT, [0, port[0]]]]
0147            actions = [[openflow.OFPAT_OUTPUT, [0, 3]]]
0148            print 'Searching for valid appication (http)
port..',str(packet)
0149       if appdstprt==80:
0150         print 'NOTICE - Valid port detected. Installing
Appropriate flow to accommodate the application.'
```

```
0151              inst.install_datapath_flow(dpid,flow,CACHE_PERMFLOW,
0152                                  openflow.OFP_FLOW_PERMANENT,actions,
0153                          bufid, openflow.OFP_DEFAULT_PRIORITY+32767,
0154                                                 inport, buf)
0155               return
0156             else:
0157           print 'No valid application (http) port. Flooding..'
0158               inst.send_openflow(dpid, bufid, buf,
openflow.OFPP_FLOOD, inport)
0159           return
0160     else:
0161       print 'No valid IP address (http) detected. Flooding..'
0162       inst.send_openflow(dpid, bufid, buf, openflow.OFPP_FLOOD,
inport)
```

Αρχικά, γίνεται η ανάθεση του δείκτη της εφαρμογής pt με την εντολή `pt=packet.next` που χρησιμοποιείται για την σάρωση των πεδίων δρομολόγησης του πακέτου OpenFlow που παραλαμβάνει ο ελεγκτής και η εφαρμογή ctrlswl3.py. Ακολούθως γίνεται ένας έλεγχος για την μορφή του πακέτου με την `if not ord(dstaddr[0]) & 1` (που είναι βασισμένη στην MAC διεύθυνση) για να εξακριβωθεί αν πρόκειται για πακέτο μονοεκπομπής ή πολυεκπομπής/μετάδοσης. Αυτός ο αρχικός έλεγχος γίνεται για σκοπούς συμβατότητας με τον ελεγκτή NOX, ούτως ώστε να εξαχθεί η θύρα εξόδου στον μεταγωγέα αν έχουμε πακέτο μονοεκπομπής (`prt = inst.st[dpid][dstaddr]`) και να ελεγχθεί αν υπάρχουν βρόχοι στο δίκτυο (loops) που μπορεί να δημιουργήσουν προβλήματα στην κίνηση.

Στην συνέχεια, προχωρούμε στον κεντρικό μηχανισμό της κλάσης δρομολόγησης. Πρώτα γίνεται έλεγχος αν η διεύθυνση προορισμού του πακέτου στο πεδίο IP, βάση του δείκτη pt, είναι έγκυρη σύμφωνα με την βιβλιοθήκη ipv4 του NOX και με την εντολή isinstance() της Python: `if isinstance(pt, ipv4).` Μετά, γίνεται ανάλογος έλεγχος στα πεδία θυρών TCP/UDP του πακέτου με τις εντολές `if isinstance(p, udp) or isinstance(p, tcp)` και χρησιμοποιώντας τις αντίστοιχες βιβλιοθήκες (tcp και udp). Με την επιτυχή περάτωση του ελέγχου οι θύρες αποστολέα και προορισμού ανατίθενται στις μεταβλητές `appsrcprt` και `appdstport` αντίστοιχα. Σε περιπτώσεις αποτυχίας των πιο πάνω ελέγχων, το πακέτο προωθείται με την μέθοδο της πλημμύρας (flood) από όλες τις θύρες εξόδου του ελεγκτή.

Ένας τελευταίος έλεγχος γίνεται και για την εγκυρότητα της θύρας εξόδου του μεταγωγέα στις γραμμές κώδικα 139-142, στις οποίες αν ανιχνευτεί ότι η θύρα εξόδου στον μεταγωγέα ταυτίζεται με την θύρα εισόδου σε αυτόν (`if prt[0] == inport`) το πακέτο πλημμυρίζεται για να μην εγκατασταθεί ροή που να δημιουργεί αχρείαστους βρόχους κίνησης στο δίκτυο.

Στις γραμμές 144-148 γίνεται η δημιουργία της ροής δρομολόγησης (`create_l3_out_flow(packet)`) και η ενσωμάτωση σε αυτήν των καταλλήλων

ενεργειών που θα γίνονται στον μεταγωγέα (`actions = [[openflow.OFPAT_OUTPUT, [0, prt[0]]]]`), στην περίπτωση μας την άμεση δρομολόγηση του πακέτου από την κατάλληλη θύρα εξόδου του μεταγωγέα. Ο τελευταίος έλεγχος γίνεται στην γραμμή 149 (`if appdstprt==80`) όπου επαληθεύεται ότι η TCP θύρα επικοινωνίας είναι η 80, που ανήκει σε εφαρμογή HTTP, και γίνεται η εγκατάσταση της ροής.

Η εντολή που κάνει αυτή την εγκατάσταση στον αρμόδιο μεταγωγέα είναι η `inst.install_datapath_flow` που δημιουργεί ροή υψηλής προτεραιότητας θέτοντας μέγιστη προτεραιότητα (`openflow.OFP_DEFAULT_PRIORITY+32767`) και υψηλό χρόνο αναμονής των εξήντα δευτερολέπτων. Σε οποιοδήποτε σημείο της κλάσης αποτύχουν οι έλεγχοι για την επικύρωση της εφαρμογής γίνεται πλημμυρίδα του πακέτου, σύμφωνα με όσα περιγράφηκαν ανωτέρω, με τις αντίστοιχες εντολές της κλάσης `inst.send_openflow(dpid, bufid, buf, openflow.OFPP_FLOOD, inport)`.

5.4.3 Η κλάση do_l3_learning της ctrlswl3.py

Η κλάση do_l3_learning(dpid, inport, packet) είναι υπεύθυνη για την αναγνώριση και επιτυχή ταυτοποίηση της διεύθυνσης IP των πακέτων που λαμβάνει ο ελεγκτής NOX. Με την χρήση αυτής της κλάσης μπορεί να διαπιστωθεί αν υπάρχουν έγκυρες ή μη διευθύνσεις IP στα πακέτα OpenFlow που λαμβάνει ο ελεγκτής από μια εφαρμογή ή συσκευή του δικτύου, προτού προχωρήσει η επεξεργασία για την εγκατάσταση ροής στην forward_l3_packet.

```
078 def do_l3_learning(dpid, inport, packet):
079     global inst
080     global iplst_src
081     global iplst_dst
082 
083     print 'L3 resolver. Scanning for valid IP addresses before
processing.'
084 
085     pt=packet.next
086     if isinstance(pt, ipv4):
087         ip_p_src = pt.srcip
088         ip_p_dst = pt.dstip
089         iplst_src.append(ip_p_src)
090         iplst_dst.append(ip_p_dst)
091         print 'IP scanning succesful.'
092         print iplst_src,iplst_dst
093         return
094     else:
095      print 'IP detection failed. Checking current field status..'
096     if isinstance(packet.next, str) or isinstance(packet.next,
int) or isinstance(packet.next, long):
```

```
097         a = create_ipaddr(packet.next)
098         print 'IP status:', a
099         srcaddr = packet.src.tostring()
100         print 'Fallback. Application has not checked out.Level 2
learning..'
101     if ord(srcaddr[0]) & 1:
102         print 'Source MAC address is 0.'
103         return
104     if inst.st[dpid].has_key(srcaddr):
105         dst = inst.st[dpid][srcaddr]
106         if dst[0] != inport:
107             log.msg('MAC has moved from
'+str(dst)+'to'+str(inport), system='ctrlswl3')
108         else:
109             print 'MAC is unchanged:',mac_to_str(packet.src)
110             return
111     else:
112         log.msg('learned MAC '+mac_to_str(packet.src)+' on %d
%d'% (dpid,inport), system="ctrlswl3")
113         print 'learned MAC',mac_to_str(packet.src),' on
dpid',dpid,'and ingress port ',inport
114
115     # learn or update timestamp of entry
116     inst.st[dpid][srcaddr] = (inport, time(), packet)
117     # Replace any old entry for (switch,mac).
118     mac = mac_to_int(packet.src)
```

Σε περιπτώσεις αποτυχίας εύρεσης έγκυρης διεύθυνσης IP από την εφαρμογή, η κλάση αυτή χρησιμεύει στην ανάλυση και παρακολούθηση των πακέτων επιπέδου 3 (IP) που καταφτάνουν στον ελεγκτή. Αυτή η παρακολούθηση γίνεται με την καταγραφή των IP διευθύνσεων αποστολέα και προορισμού στις λίστες `iplst_src` και `iplst_dst`.

Αν αποτύχει η ανίχνευση έγκυρων διευθύνσεων IP, η κλάση προσπαθεί να εντοπίσει την μορφή της αλφαριθμητικής ακολουθίας που υπάρχει στο πεδίο IP του πακέτου με τις εντολές `if isinstance(packet.next, str) or isinstance(packet.next, int) or isinstance(packet.next, long): a = create_ipaddr(packet.next)` και ενημερώνει σχετικά.

Ακολούθως, η εφαρμογή επιστρέφει σε ανίχνευση επιπέδου 2 και αποθήκευσης της διεύθυνσης MAC στις γραμμές κώδικα 101-118, σύμφωνα με τα όσα εξηγήθηκαν σε ανώτερο κεφάλαιο για την λειτουργία της pyswitch.py. Αυτό γίνεται για να είναι σε θέση η εφαρμογή να εφαρμόσει την μέθοδο της πλημμύρας σε πακέτα OpenFlow που δεν διαθέτουν έγκυρα IP πεδία.

ΚΕΦΑΛΑΙΟ 6

6.1 Εκτέλεση της εφαρμογής ctrlswl3.py και ctrlsw.py στον ελεγκτή NOX

Με το πέρας της ανάπτυξης της application – aware εφαρμογής ctrlswl3.py και ctrlsw.py, οι εφαρμογές αναπτύχθηκαν στον ελεγκτή NOX για εκτέλεση και δοκιμή, ούτως ώστε να επαληθευθεί η σωστή και σταθερή λειτουργίας τους. Λόγω κάποιων ιδιμορφιών στο δίκτυο, η εγκαταστάση των ροών δεδομένων γίνεται χρησιμοποιώντας την device-aware εφαρμογή ctrlsw.py, αλλά παρουσιάζεται και η λειτουργία της ctrlswl3.py στο επόμενο κεφάλαιο. Τα αποτελέσματα της εκτέλεσης αυτής παρουσιάζονται πιο κάτω, όπου γίνονται διάφορες αναπτύξεις του κώδικα, ανάλογα με τις συσκευές στις οποίες ανατίθεται προτεραιότητα στο δίκτυο.

Για τον έλεγχο των αποτελεσμάτων και της επιτυχούς ή μη εγκατάστασης ροής στους μεταγωγείς χρησιμοποιούνται εντολές κατευθείαν στους μεταγωγείς OpenFlow. Οι εντολές αυτές είναι μέρος του dpctl, ενός βοηθητικού προγράμματος των μεταγωγέων που τρέχουν το πρωτόκολλο OpenFlow με την χρήση των NetFPGA's. Με την χρήση του dpctl γίνεται άμεσα παρατήρηση και έλεγχος όλων των παραμέτρων του NetFPGA, πέραν των καταχωρίσεων ροών που διαθέτει ο μεταγωγέας.

Πέραν του προγράμματος dpctl παρέχονται και στιγμιότυπα (screenshots) κατά την εκτέλεση των εφαρμογών ctrlswl3.py και ctrlsw.py από τον ελεγκτή NOX, όπως αυτά τυπώνονται στην οθόνη του ελεγκτή. Με τα στιγμιότυπα αυτά δίνεται μια πιο ξεκάθαρη εικόνα της λειτουργίας των εφαρμογών, καθώς παρουσιάζονται τα μηνύματα των συμβάντων και η εγκατάσταση των ροών στους μεταγωγείς σε πραγματικό χρόνο.

Η παρουσίαση των αποτελεσμάτων με αυτό τον τρόπο διαμορφώθηκε για την παρουσίαση επιτυχούς εγκατάστασης ροών στο δίκτυο OpenFlow. Ο μηχανισμός αυτός είναι ο ίδιος για τις δύο εφαρμογές (ctrlswl3.py και ctrlsw.py) και για την παρουσίαση αυτή, γίνεται ένας διαχωρισμός των παραμέτρων της εφαρμογής ctrlsw.py σε προορισμούς υψηλής και χαμηλής προτεραιότητας, ούτως ώστε να καταστεί σαφής η λειτουργία της εφαρμογής κατά την εγκατάσταση ροών στους

υποκείμενους μεταγωγείς. Πρώτα όμως γίνεται η παρουσίαση της λειτουργίας της application-aware εφαρμογής ctrlswl3.py, που είναι και ο κύριος στόχος της διπλωματικής αυτής εργασίας.

6.2 Ανάπτυξη και λειτουργία της ctrlswl3.py στον ελεγκτή NOX

Η application-aware εφαρμογή ctrlswl3.py αναπτύσσεται στον ελεγκτή NOX για λειτουργία στο δίκτυο OpenFlow του εργαστηρίου Δικτύων Υπολογιστών. Βάση της λειτουργίας της, που αναπτύχθηκε ενδελεχώς στο προηγούμενο κεφάλαιο, η εφαρμογή ανακόπτει πακέτα OpenFlow και προσπάθει να εξάγει την διεύθυνση IP προορισμού και την θύρα TCP/UDP του πακέτου. Βάση αυτής της θύρας η εφαρμογή αντιλαμβάνεται ποια εφαρμογή από την τερματική συσκευή που απέστειλε το πακέτο, χρησιμοποιεί το δίκτυο. Αν η εφαρμογή αυτή είναι τύπου http και επικοινωνεί μέσω της TCP θύρας 80, τότε η ctrlswl3.py εγκαθιστά ροή (flow) στον μεταγωγέα που απέστειλε το πακέτο, με αποτέλεσμα να δημιουργείται μια απευθείας ροή δεδομένων της υπηρεσίας http στο δίκτυο.

Ένα παράδειγμα της λειτουργίας και ανάπτυξης της ctrlswl3.py παρουσιάζεται πιο κάτω:

```
00095|openflow-event|DBG:received packet-in event from 0023208e4d70 (len:60)
L3 resolver. Scanning for valid IP addresses before processing.
IP detection failed. Checking current field status..
IP status: None
Fallback. Application has not checked out.Level 2 learning..
MAC is unchanged: a4:56:30:d1:91:02
No valid IP address (http) detected. Flooding..
00096|openflow-event|DBG:received packet-in event from 0023208e4d70 (len:60)
L3 resolver. Scanning for valid IP addresses before processing.
IP detection failed. Checking current field status..
IP status: None
Fallback. Application has not checked out.Level 2 learning..
MAC is unchanged: a4:56:30:d1:91:02
00097|openflow-event|DBG:received packet-in event from 0023208e4d70 (len:60)
L3 resolver. Scanning for valid IP addresses before processing.
IP detection failed. Checking current field status..
IP status: None
Fallback. Application has not checked out.Level 2 learning..
MAC is unchanged: a4:56:30:d1:91:02
00098|openflow|DBG:stream: idle 15 seconds, sending inactivity probe
00099|openflow|DBG:stream: message received, entering CONNECTED
00100|openflow-event|DBG:received echo-request event from 002320985833
(len:0)
00101|openflow-event|DBG:received packet-in event from 0023208e4d70 (len:60)
L3 resolver. Scanning for valid IP addresses before processing.
IP detection failed. Checking current field status..
IP status: None
Fallback. Application has not checked out.Level 2 learning..
```

```
MAC is unchanged: a4:56:30:d1:91:02
00102|openflow-event|DBG:received packet-in event from 0023208e4d70 (len:77)
L3 resolver. Scanning for valid IP addresses before processing.
IP detection failed. Checking current field status..
IP status: None
Fallback. Application has not checked out.Level 2 learning..
learned MAC 00:02:7e:19:e0:1d  on dpid 150870052208 and ingress port  4
00103|openflow-event|DBG:received packet-in event from 0023208e4d70 (len:60)
L3 resolver. Scanning for valid IP addresses before processing.
IP detection failed. Checking current field status..
IP status: None
Fallback. Application has not checked out.Level 2 learning..
learned MAC a4:56:30:d1:91:02  on dpid 150870052208 and ingress port  4
00104|openflow-event|DBG:received packet-in event from 0023208e4d70 (len:60)
L3 resolver. Scanning for valid IP addresses before processing.
IP detection failed. Checking current field status..
IP status: None
Fallback. Application has not checked out.Level 2 learning..
MAC is unchanged: a4:56:30:d1:91:02
00240|openflow-event|DBG:received packet-in event from 0023208e4d70 (len:60)
L3 resolver. Scanning for valid IP addresses before processing.
IP detection failed. Checking current field status..
IP status: None
Fallback. Application has not checked out.Level 2 learning..
MAC is unchanged: a4:56:30:d1:91:02
No valid IP address (http) detected. Flooding..
```

Στιγμιότυπο 6.2-1 – Λειτουργία της ctrlswl3.py στον NOX

Η επεξεργασία ενός πακέτου αρχίζει κατά την λήψη του από τον ελεγκτή, με συμβάν εισόδου πακέτου (π.χ. `00095|openflow-event|DBG:received packet-in event from 0023208e4d70 (len:60)`). Ακολούθως γίνεται η προσπάθεια εξαγωγής των διευθύνσεων IP προορισμού και αποστολέα, η οποία στο υφιστάμενο δίκτυο αποτυγχάνει λόγω του ότι τα NetFPGA's δημιουργούν πακέτα επιπέδου 2, με MAC διευθυνσιοδότηση.

Έτσι, η εφαρμογή προσπαθεί να προσπελάσει το πεδίο IP του πακέτου και να αποκωδικοποιήσει την παρεχομένη τιμή (αν υπάρχει) του πεδίου αυτού και να ενημερώσει τον διαχειριστή σχετικά με τον τύπο του πακέτου. Εδώ βλέπουμε ότι δεν παρέχεται καμιά τιμή στο IP πεδίο, οπότε και η εφαρμογή επιστρέφει το ανάλογο μήνυμα (`IP status: None`).

Τελικά, η εφαρμογή υποχωρεί στο προηγούμενο επίπεδο (MAC – level) με το μήνυμα < `Fallback. Application has not checked out.Level 2 learning..` >και εξάγει τα πεδία προορισμού και αποστολέα MAC (`learned MAC a4:56:30:d1:91:02 on dpid 150870052208 and ingress port  4`). Αυτό γίνεται για να μπορέσει η εφαρμογή να εφαρμόσει την μέθοδο της πλημμύρας στο πακέτο (flood) και να στείλει το πακέτο στις θύρες εξόδου του ελεγκτή μετά την αποτυχία έγκυρης εύρεσης IP εφαρμογής (`No valid IP address (http) detected. Flooding..`).

6.3 Ο ελεγκτής NOX κατά την λειτουργία της ctrlsw.py

Παρατίθεται αρχικώς μια σειρά στιγμιότυπων από την οθόνη του ελεγκτή NOX, κατά το τρέξιμο της εφαρμογής ctrlsw.py από τον ελεγκτή. Το τρέξιμο της εφαρμογής γίνεται και με τους δύο OpenFlow μεταγωγείς του δικτύου σε λειτουργία και ενωμένους με τον NOX. Τα αποτελέσματα έχουν ως εξής:

```
00001|nox|INFO:Starting nox_core (/usr/local/src/nox/build/src/.libs/lt-nox_core)
00002|pyrt|DBG:Loading a component description file
'nox/coreapps/simple_c_py_app/meta.json'.
00003|pyrt|DBG:Loading a component description file 'nox/coreapps/snmp/meta.json'.
00004|pyrt|DBG:Loading a component description file 'nox/coreapps/pyrt/meta.json'.
00005|pyrt|DBG:Loading a component description file 'nox/coreapps/switch/meta.json'.
00006|pyrt|DBG:Loading a component description file
'nox/coreapps/simple_c_app/meta.json'.
00007|pyrt|DBG:Loading a component description file 'nox/coreapps/examples/meta.json'.
00008|pyrt|DBG:Loading a component description file
'nox/coreapps/examples/t/meta.json'.
00009|pyrt|DBG:Loading a component description file
'nox/coreapps/messenger/meta.json'.
00010|pyrt|DBG:Loading a component description file
'nox/coreapps/coretests/meta.json'.
00011|pyrt|DBG:Loading a component description file
'nox/coreapps/testharness/meta.json'.
00012|pyrt|DBG:Loading a component description file 'nox/coreapps/hub/meta.json'.
00013|pyrt|DBG:Loading a component description file
'nox/netapps/networkstate/meta.json'.
00014|pyrt|DBG:Loading a component description file
'nox/netapps/user_event_log/meta.json'.
00015|pyrt|DBG:Loading a component description file 'nox/netapps/data/meta.json'.
00016|pyrt|DBG:Loading a component description file 'nox/netapps/route/meta.json'.
00017|pyrt|DBG:Loading a component description file
'nox/netapps/switch_management/meta.json'.
00018|pyrt|DBG:Loading a component description file 'nox/netapps/topology/meta.json'.
00019|pyrt|DBG:Loading a component description file 'nox/netapps/discovery/meta.json'.
00020|pyrt|DBG:Loading a component description file
'nox/netapps/switchstats/meta.json'.
00021|pyrt|DBG:Loading a component description file
'nox/netapps/authenticator/meta.json'.
00022|pyrt|DBG:Loading a component description file 'nox/netapps/tests/meta.json'.
00023|pyrt|DBG:Loading a component description file 'nox/netapps/tablog/meta.json'.
00024|pyrt|DBG:Loading a component description file 'nox/netapps/routing/meta.json'.
00025|pyrt|DBG:Loading a component description file 'nox/netapps/storage/meta.json'.
00026|pyrt|DBG:Loading a component description file 'nox/netapps/storage/t/meta.json'.
00027|pyrt|DBG:Loading a component description file
'nox/netapps/bindings_storage/meta.json'.
00028|pyrt|DBG:Loading a component description file
'nox/netapps/bindings_storage/t/meta.json'.
00029|pyrt|DBG:Loading a component description file 'nox/netapps/hoststate/meta.json'.
00030|pyrt|DBG:Loading a component description file 'nox/netapps/lavi/meta.json'.
00031|pyrt|DBG:Loading a component description file
'nox/netapps/flow_fetcher/meta.json'.
00032|pyrt|DBG:Loading a component description file
'nox/webapps/webservice/meta.json'.
00033|pyrt|DBG:Loading a component description file 'nox/webapps/webserver/meta.json'.
00034|pyrt|DBG:Loading a component description file 'nox/webapps/miscws/meta.json'.
00035|pycomponent|DBG:Importing Python module nox.coreapps.examples.ctrlsw
00036|nox|DBG:Application installation report:
00037|nox|DBG:python:
        Current state: INSTALLED
        Required state: INSTALLED
        Dependencies:
```

```
00038|nox|DBG:built-in DSO deployer:
        Current state: INSTALLED
        Required state: INSTALLED
        Dependencies:

00039|nox|DBG:ctrlsw:
        Current state: INSTALLED
        Required state: INSTALLED
        Dependencies: 'python' OK, 'python' OK

00040|nox|DBG:built-in event dispatcher:
        Current state: INSTALLED
        Required state: INSTALLED
        Dependencies:

00041|openflow|DBG:Passive tcp interface bound to port 6633
00042|nox|INFO:nox bootstrap complete
```

Στιγμιότυπο 6.3-1 – Λειτουργία του NOX

Αρχικά γίνεται η εκκίνηση της εφαρμογής ctrlsw.py στον ελεγκτή και ελέγχεται από τον μεταφραστή της Python, μετά την φόρτωση των απαραιτήτων συνοδευτικών εφαρμογών, που είναι απαραίτητες για την λειτουργία της ctrlsw.py.

```
00041|openflow|DBG:Passive tcp interface bound to port 6633
00042|nox|INFO:nox bootstrap complete
00043|openflow|DBG:Passive tcp interface received connection
00044|openflow|DBG:stream: negotiated OpenFlow version 0x01 (we support versions 0x01
to 0x01 inclusive, peer no later than version 0x01)
00045|nox|DBG:Success sending in 'sending switch config'
00046|nox|DBG:Success sending in 'receiving features reply'
00047|nox|DBG:Success receiving in 'receiving features reply'
00048|nox|DBG:Success sending in 'receiving ofmp capability reply'
00049|nox|DBG:Success receiving in 'receiving ofmp capability reply'
00050|nox|DBG:Datapath 002320aefd39 sent error in response to capability reply,
assuming no management support
00051|nox|DBG:No switch auth module registered, auto-approving switch
00052|nox|DBG:Registering switch with DPID = 2320aefd39
00053|nox.coreapps.examples.pyswitch|INFO:Switch 2320aefd39 has joined the network
00054|openflow|DBG:Passive tcp interface received connection
00055|openflow|DBG:stream: negotiated OpenFlow version 0x01 (we support versions 0x01
to 0x01 inclusive, peer no later than version 0x01)
00056|nox|DBG:Success sending in 'sending switch config'
00057|nox|DBG:Success sending in 'receiving features reply'
00058|nox|DBG:Success receiving in 'receiving features reply'
00059|nox|DBG:Success sending in 'receiving ofmp capability reply'
00060|nox|DBG:Success receiving in 'receiving ofmp capability reply'
00061|nox|DBG:Datapath 0023204bc4a3 sent error in response to capability reply,
assuming no management support
00062|nox|DBG:No switch auth module registered, auto-approving switch
00063|nox|DBG:Registering switch with DPID = 23204bc4a3
00064|nox.coreapps.examples.pyswitch|INFO:Switch 23204bc4a3 has joined the network
00065|openflow-event|DBG:received packet-in event from 0023204bc4a3 (len:60)
L3 resolver
IP status: None
level 2 learning
learned MAC a4:56:30:d1:91:02  on dpid 150865691811 and ingress port  4
NOTICE - Installing flow for: [a4:56:30:d1:91:02>a4:56:30:d1:91:02:9000]
NOTICE - Installing DEFAULT priority flow for dpid 150865691811
00066|openflow-event|DBG:received packet-in event from 0023204bc4a3 (len:60)
```

```
L3 resolver
IP status: None
level 2 learning
```

Στιγμιότυπο 6.3-2 – Η ctrlsw.py

Σε αυτό το στιγμιότυπο βλέπουμε την αρχικοποίηση του ελεγκτή και την καταγραφή των ροών δεδομένων καθώς οι δύο μεταγωγείς OpenFlow (of1 και of2) συσχετίζονται (associated) κατά την εκκίνηση τους με τον NOX. Η καταχώρηση των μεταγωγέων στον ελεγκτή γίνεται ως ακολούθως:

1. Παραλαβή αίτησης σύνδεσης μέσω θύρας 6633 του TCP:
   `00043|openflow|DBG:Passive tcp interface received connection`
2. Ανάλυση και καταγραφή υποστηριζόμενων δυνατοτήτων και παραμέτρων του μεταγωγέα:
   ```
   00044|openflow|DBG:stream: negotiated OpenFlow version 0x01 (we support
   versions 0x01 to 0x01 inclusive, peer no later than version 0x01)
   00045|nox|DBG:Success sending in 'sending switch config'
   00046|nox|DBG:Success sending in 'receiving features reply'
   00047|nox|DBG:Success receiving in 'receiving features reply'
   00048|nox|DBG:Success sending in 'receiving ofmp capability reply'
   00049|nox|DBG:Success receiving in 'receiving ofmp capability reply'
   00050|nox|DBG:Datapath 002320aefd39 sent error in response to capability reply,
   assuming no management support
   ```
3. Αναγνώριση και καταγραφή ροής δεδομένων (datapath)
   ```
   00050|nox|DBG:Datapath 002320aefd39 sent error in response to capability reply,
   assuming no management support
   ```
4. Τελική έγκριση και καταγραφή μεταγωγέα
   ```
   00051|nox|DBG:No switch auth module registered, auto-approving switch
   00052|nox|DBG:Registering switch with DPID = 2320aefd39
   00053|nox.coreapps.examples.pyswitch|INFO:Switch 2320aefd39 has joined the
   network
   ```

Τέλος, εδώ παρουσιάζεται πιο καθαρά η διαδικασία κατασκευής και εγκατάστασης ροής από τον ελεγκτή. Η εγκατάσταση της ροής γίνεται μετά την παραλαβή ενός συμβάντος παραλαβής πακέτου από τον ελεγκτή (packet-in event) :

```
learned MAC a4:56:30:d1:91:02  on dpid 150865691811 and ingress port  4
NOTICE - Installing flow for: [a4:56:30:d1:91:02>01:00:0c:cc:cc:cc:llc]
NOTICE - Installing DEFAULT priority flow for dpid 150865691811
00185|openflow-event|DBG:received packet-in event from 0023204bc4a3 (len:90)
L3 resolver
IP status: None
level 2 learning
MAC is unchanged: a4:56:30:d1:91:02
NOTICE - Installing flow for: [a4:56:30:d1:91:02>01:00:0c:00:00:00:llc]
NOTICE - Installing DEFAULT priority flow for dpid 150865691811
00186|openflow-event|DBG:received packet-in event from 0023204bc4a3 (len:60)
L3 resolver
IP status: None
level 2 learning
MAC is unchanged: a4:56:30:d1:91:02
NOTICE - Installing flow for: [a4:56:30:d1:91:02>a4:56:30:d1:91:02:9000]
NOTICE - Installing DEFAULT priority flow for dpid 150865691811
00187|openflow|DBG:stream: idle 15 seconds, sending inactivity probe
```

```
00188|openflow|DBG:stream: message received, entering CONNECTED
00189|openflow-event|DBG:received echo-request event from 002320aefd39
(len:0)
00190|openflow-event|DBG:received flow expired event from 0023204bc4a3
00191|openflow-event|DBG:received flow expired event from 0023204bc4a3
00192|openflow-event|DBG:received flow expired event from 0023204bc4a3
00193|openflow-event|DBG:received packet-in event from 0023204bc4a3 (len:60)
```

Στιγμιότυπο 6.3-3 – Διαδικασία κατασκευής και εγκατάστασης ροής

Το συμβάν παραλαβής πακέτου, που εμφανίζεται σε αρκετά μηνύματα στο πιο πάνω στιγμιότυπο, έχει την δομή:

00185|openflow-event|DBG:received packet-in event from 0023204bc4a3 (len:90)			
Αύξων Αριθμός	Ειδοποίηση τύπου συμβάντος OpenFlow, εν προκειμένω παραλαβής (packet-in)	Αρμόδιος Μεταγωγέας πακέτου	Μήκος μηνύματος

Αυτή η ροή ορίζεται στην προκαθορισμένη προτεραιότητα του ελεγκτή, αφού ακόμα δεν έχουν ανατεθεί συσκευές προτεραιότητας στις λίστες προτεραιότητας της εφαρμογής ctrlsw.py. Τα μηνύματα εγκατάστασης ροής έχουν την ακόλουθη δομή:

NOTICE - Installing flow for: [a4:56:30:d1:91:02>01:00:0c:00:00:00:llc]	
Ειδοποίηση Εγκατάστασης	MAC δ/νση αποστολέα > MAC δ/νση προορισμού

NOTICE - Installing DEFAULT priority flow for dpid 150865691811		
Συνέχεια Ειδοποίησης	Προτεραιότητα ροής	Διαδρομή δεδομένων μεταγωγέα

Λόγω της μικρής διάρκειας αναμονής της ροής (idle_timeout), που είναι έξι δευτερόλεπτα, παρατηρούμε και πολλαπλά μηνύματα λήξης ροών που φτάνουν στον ελεγκτή, στα μηνύματα 00190 – 00193, όπου βλέπουμε το συμβάν λήξης και τον αρμόδιο μεταγωγέα, βάση της ροής δεδομένων με την οποία είναι συσχετισμένος, για παράδειγμα:

00190|openflow-event|DBG:received flow expired event from 0023204bc4a3	
Συμβάν Λήξης	Μεταγωγέας

6.4 Ανάθεση λίστας υψηλής προτεραιότητας στην ctrlsw.py

Για την ανάθεση συσκευών με υψηλή προτεραιότητα, χρησιμοποιείται η λίστα macs_str_hp της εφαρμογής ctrlsw.py, όπου γίνεται ανάθεση προτεραιότητας στη διεύθυνση `a4:56:30:d1:91:02`. Η διεύθυνση αυτή παράγεται από τα NetFPGA's που είναι στις τερματικές συσκευές για την δημιουργία κίνησης στο δίκτυο. Να διευκρινιστεί ότι για την δημιουργία πολλαπλών ροών στους μεταγωγείς, αίρεται προσωρινά ο έλεγχος για προορισμούς πολυεκπομπής και μετάδοσης, με αποτέλεσμα να έχουμε την δημιουργία πολλαπλών ροών. Οι ροές αυτές δεν είναι αναγκαίες υπό κανονικές συνθήκες λειτουργίας, αλλά παρουσιάζονται εδώ για σκοπούς επίδειξης της λειτουργίας του ελεγκτή.

Πιο κάτω παρουσιάζεται η εκτύπωση μηνυμάτων από τον ελεγκτή με την εγκατάσταση ροών και πιο κάτω την ειδοποίηση προτεραιότητας :

```
00065|openflow-event|DBG:received packet-in event from 0023209aaa11 (len:60)
L3 resolver
IP status: None
level 2 learning
learned MAC a4:56:30:d1:91:02  on dpid 150870862353 and ingress port  4
NOTICE - Installing flow for: [a4:56:30:d1:91:02>01:80:c2:00:00:00:llc]
NOTICE - Installing HIGH priority flow for dpid 150870862353
00066|openflow-event|DBG:received packet-in event from 0023209aaa11
(len:128)
L3 resolver
IP status: None
level 2 learning
MAC is unchanged: a4:56:30:d1:91:02
NOTICE - Installing flow for: [a4:56:30:d1:91:02>01:00:0c:cc:cc:cc:llc]
NOTICE - Installing HIGH priority flow for dpid 150870862353
00067|openflow-event|DBG:received packet-in event from 0023209aaa11 (len:60)
L3 resolver
IP status: None
level 2 learning
MAC is unchanged: a4:56:30:d1:91:02
NOTICE - Installing flow for: [a4:56:30:d1:91:02>a4:56:30:d1:91:02:9000]
NOTICE - Installing HIGH priority flow for dpid 150870862353
00068|openflow|DBG:stream: idle 15 seconds, sending inactivity probe
00069|openflow|DBG:stream: message received, entering CONNECTED
00070|openflow-event|DBG:received echo-request event from 00232032c635
(len:0)
00071|openflow|DBG:stream: idle 15 seconds, sending inactivity probe
00072|openflow|DBG:stream: message received, entering CONNECTED
00073|openflow-event|DBG:received echo-request event from 0023209aaa11
(len:0)
00074|openflow|DBG:stream: idle 15 seconds, sending inactivity probe
00075|openflow|DBG:stream: message received, entering CONNECTED
00076|openflow-event|DBG:received echo-request event from 00232032c635
(len:0)
00077|openflow-event|DBG:received packet-in event from 0023209aaa11 (len:60)
L3 resolver
IP status: None
```

```
level 2 learning
learned MAC a4:56:30:d1:91:02  on dpid 150870862353 and ingress port  4
NOTICE - Installing flow for: [a4:56:30:d1:91:02>01:00:0c:cc:cc:cc:llc]
NOTICE - Installing HIGH priority flow for dpid 150870862353
00078|openflow-event|DBG:received packet-in event from 0023209aaa11 (len:90)
L3 resolver
IP status: None
level 2 learning
MAC is unchanged: a4:56:30:d1:91:02
learned MAC 00:02:7e:19:e0:1d  on dpid 150870862353 and ingress port  4
NOTICE - Installing flow for: [(Cisco Systems,
Inc.):19:e0:1d>ab:00:00:02:00:00:6002]
NOTICE - Installing DEFAULT priority flow for dpid 150870862353
```

Παρατηρούμε ότι γίνεται επιτυχής εγκατάστασης όλων των ροών υψηλής προτεραιότητας καθώς εμφανίζεται το μήνυμα: `NOTICE - Installing HIGH priority flow for dpid 150870862353`.

Κατά το τρέξιμο της εφαρμογής τυπώνονται κάποια μηνύματα στην οθόνη που αφορούν την ενημέρωση της μνήμης της εφαρμογής για τους διάφορους προορισμούς των πακέτων στο δίκτυο, όπως είναι το :

learned MAC a4:56:30:d1:91:02 on dpid 150870862353 and ingress port 4		
Διεύθυνση MAC	Διαδρομή δεδομένων (datapath) που αφορά	Θύρα εισόδου στον αρμόδιο μεταγωγέα

Εκτός αυτού, τυπώνονται και μηνύματα ενημέρωσης για ήδη αποθηκευμένες διευθύνσεις (`MAC is unchanged: a4:56:30:d1:91:02`) καθώς και μηνύματα προσπάθειας εύρεσης διευθύνσεων IP στο πακέτο (`L3 resolver,IP status: None`). Τέλος, όταν ένας μεταγωγέας βρίσκεται σε κατάσταση αναμονής (όπως ο of1) ο ελεγκτής στέλνει μηνύματα ανίχνευσης (inactivity probes), για να επαληθεύσει ότι ο μεταγωγέας είναι ακόμα ενωμένος στο δίκτυο. Αυτά τα μηνύματα ανίχνευσης είναι:

l00074|openflow|DBG:stream: idle 15 seconds, sending inactivity probe		
Αύξων Αριθμός	Ενημέρωση αποστολής μηνύματος ανίχνευσης	
00075|openflow|DBG:stream: message received, entering CONNECTED		
Αύξων Αριθμός	Ενημέρωση επιτυχούς παραλαβής απάντησης	
00076|openflow-event|DBG:received echo-request event from 00232032c635 (len:0)		
Αύξων Αριθμός	Ενημέρωση επιτυχούς παραλαβής μηνύματος ηχούς από τον μεταγωγέα	Μεταγωγέας που ανταποκρίθηκε

Για επαλήθευση της επιτυχούς εγκατάστασης των ροών στους μεταγωγείς OpenFlow του δικτύου, τρέχουμε το βοηθητικό πρόγραμμα dpctl σε καθένα από τους δύο μεταγωγείς και προκύπτουν τα πιο κάτω στιγμιότυπα:

1. Από τον of1 μεταγωγέα:

```
sudo dpctl dump-flows unix:/root/test
stats_reply (xid=0xde6a5017): flags=none type=1(flow)
$
$
$ sudo dpctl dump-flows unix:/root/test
stats_reply (xid=0x815bd852): flags=none type=1(flow)
$
$
$
$ sudo dpctl dump-flows unix:/root/test
stats_reply (xid=0x79954ecc): flags=none type=1(flow)
$
$
$ sudo dpctl dump-flows unix:/root/test
stats_reply (xid=0x630147bf): flags=none type=1(flow)
$
$ sudo dpctl dump-flows unix:/root/test
stats_reply (xid=0xdab9d837): flags=none type=1(flow)
$
$
$
$ sudo dpctl dump-flows unix:/root/test
stats_reply (xid=0x45e034c3): flags=none type=1(flow)
$
$
$ sudo dpctl dump-flows unix:/root/test
stats_reply (xid=0x620c59fc): flags=none type=1(flow)
```

Στιγμιότυπο 6.4-1 Κατάσταση ροών υψηλής προτεραιότητας στον μεταγωγέα of1

2. Από τον of2 μεταγωγέα:

```
$ sudo dpctl dump-flows unix:/var/run/test
stats_reply (xid=0x94ff4c65): flags=none type=1(flow)
  cookie=0,    duration_sec=172s,    duration_nsec=72000000s,    table_id=0,
priority=65535,                  n_packets=6,                  n_bytes=540,
idle_timeout=60,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=01:
00:0c:00:00:00,dl_type=0x004c,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_p
roto=0,tp_src=0,tp_dst=0,actions=output:3
  cookie=0,    duration_sec=172s,    duration_nsec=75000000s,    table_id=0,
priority=65535,                  n_packets=6,                  n_bytes=360,
idle_timeout=60,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=01:
00:0c:cc:cc:cc,dl_type=0x0022,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_p
roto=0,tp_src=0,tp_dst=0,actions=output:3
  cookie=0,    duration_sec=199s,    duration_nsec=884000000s,    table_id=0,
priority=65535,                  n_packets=20,                 n_bytes=1200,
idle_timeout=60,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=a4:
56:30:d1:91:02,dl_type=0x9000,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_p
roto=0,tp_src=0,tp_dst=0,actions=output:3
  cookie=0,    duration_sec=200s,    duration_nsec=83000000s,    table_id=0,
priority=65535,                  n_packets=12,                 n_bytes=5472,
idle_timeout=60,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=01:
00:0c:cc:cc:cc,dl_type=0x01ba,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_p
roto=0,tp_src=0,tp_dst=0,actions=output:3
```

```
  cookie=0,    duration_sec=202s,    duration_nsec=65000000s,    table_id=0,
priority=65535,                  n_packets=101,                  n_bytes=6060,
idle_timeout=60,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=01:
80:c2:00:00:00,dl_type=0x0026,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_p
roto=0,tp_src=0,tp_dst=0,actions=output:3
$
$
$ sudo dpctl dump-flows unix:/var/run/test
stats_reply (xid=0xdb488cdf): flags=none type=1(flow)
  cookie=0,    duration_sec=255s,    duration_nsec=412000000s,    table_id=0,
priority=65535,                   n_packets=9,                   n_bytes=810,
idle_timeout=60,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=01:
00:0c:00:00:00,dl_type=0x004c,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_p
roto=0,tp_src=0,tp_dst=0,actions=output:3
  cookie=0,    duration_sec=255s,    duration_nsec=415000000s,    table_id=0,
priority=65535,                   n_packets=9,                   n_bytes=540,
idle_timeout=60,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=01:
00:0c:cc:cc:cc,dl_type=0x0022,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_p
roto=0,tp_src=0,tp_dst=0,actions=output:3
  cookie=0,    duration_sec=283s,    duration_nsec=224000000s,    table_id=0,
priority=65535,                  n_packets=29,                  n_bytes=1740,
idle_timeout=60,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=a4:
56:30:d1:91:02,dl_type=0x9000,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_p
roto=0,tp_src=0,tp_dst=0,actions=output:3
  cookie=0,    duration_sec=283s,    duration_nsec=423000000s,    table_id=0,
priority=65535,                  n_packets=13,                  n_bytes=5928,
idle_timeout=60,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=01:
00:0c:cc:cc:cc,dl_type=0x01ba,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_p
roto=0,tp_src=0,tp_dst=0,actions=output:3
  cookie=0,    duration_sec=285s,    duration_nsec=405000000s,    table_id=0,
priority=65535,                  n_packets=143,                  n_bytes=8580,
idle_timeout=60,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=01:
80:c2:00:00:00,dl_type=0x0026,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_p
roto=0,tp_src=0,tp_dst=0,actions=output:3
$
```

Στιγμιότυπο 6.4-2 Κατάσταση ροών υψηλής προτεραιότητας στον μεταγωγέα of2

Από τα πιο πάνω στιγμιότυπα βλέπουμε τις ροές που έχουν εγκατασταθεί επιτυχώς στον μεταγωγέα of2. Στον μεταγωγέα of1 βλέπουμε ότι δεν υπάρχουν ροές, καθώς δεν έγινε καμιά αίτηση για χειρισμό πακέτου από τον ελεγκτή NOX. Στον μεταγωγέα of2 έχουμε εγκατάσταση ροών μέγιστης προτεραιότητας (priority=65535) για όλες τις ροές με χρόνο αναμονής τα εξήντα δευτερόλεπτα (idle_timeout).

Χαρακτηριστικά, η δομή μιας καταχώρησης ροής όπως φαίνεται και πιο πάνω διαθέτει τα πεδία:

Α/Α	Πεδίο Ροής	Επεξήγηση
1.	cookie	Δεδομένα για αποθήκευση στην ροή
2.	duration_sec	Διάρκεια ροής σε δευτερόλεπτα
3.	duration_nsec	Διάρκεια ροής σε νανοδευτερόλεπτα
4.	table_id	Αύξων αριθμός πίνακα μεταγωγέα

5.	priority	Προτεραιότητα αντιστοίχισης ροής
6.	n_packets	Αριθμός πακέτων που αντιστοιχήθηκαν με την ροή
7.	n_bytes	Συνολικά bytes που διακινηθήκαν από την ροή
8.	idle_timeout	Χρόνος Αναμονής
9.	hard_timeout	Χρόνος Λήξης
10.	in_port	Θύρα εισόδου πακέτων
11.	dl_src	Προορισμός πακέτων στο επίπεδο 2 (MAC)
12.	dl_dst	Πηγή πακέτων στο επίπεδο 2 (MAC)
13.	dl_type	Τύπος πρωτοκόλλου στο επίπεδο 2
14.	nw_src	Προορισμός πακέτων στο επίπεδο 3 (IP)
15.	nw_dst	Πηγή πακέτων στο επίπεδο 3 (IP)
16.	nw_tos	Τύπος υπηρεσίας στο επίπεδο 3 (TCP/IP)
17.	nw_proto	Τύπος πρωτοκόλλου στο επίπεδο 3 (TCP/IP)
18.	tp_src	Προορισμός πακέτων στο επίπεδο 3 (TCP)
19.	tp_dst	Πηγή πακέτων στο επίπεδο 3 (TCP)
20.	actions	Δέσμη ενεργειών που εκτελεί η ροή στο αντιστοιχηθέν πακέτο

Πίνακας 6.4-3 – Δομή καταχώρησης ροής

6.5 Ανάθεση λίστας χαμηλής προτεραιότητας στην ctrlsw.py

Για την ανάθεση συσκευών με χαμηλή προτεραιότητα χρησιμοποιείται αυτή τη φορά η λίστα macs_str_lp της εφαρμογής ctrlsw.py, όπου γίνεται ανάθεση προτεραιότητας στη διεύθυνση `a4:56:30:d1:91:02`. Η διεύθυνση αυτή παράγεται από τα NetFPGA's που είναι στις τερματικές συσκευές για την δημιουργία κίνησης στο δίκτυο. Τώρα παρατηρείται μεγαλύτερη κίνηση στον ελεγκτή καθώς οι ροές χαμηλής προτεραιότητας λήγουν γρήγορα, με αποτέλεσμα ο NOX να πρέπει να επανεγκαταστήσει μια ροή χαμηλής προτεραιότητας, για την οποία παρήλθε ο χρόνος αναμονής της.

Πιο κάτω παρουσιάζεται η εκτύπωση μηνυμάτων από τον ελεγκτή με την εγκατάσταση ροών και πιο κάτω την ειδοποίηση προτεραιότητας :

```
L3 resolver
IP status: None
level 2 learning
learned MAC a4:56:30:d1:91:02  on dpid 150861246508 and ingress port  4
NOTICE - Installing flow for: [a4:56:30:d1:91:02>a4:56:30:d1:91:02:9000]
NOTICE - Installing LOW priority flow for dpid 150861246508
00095|openflow-event|DBG:received flow expired event from 00232007f02c
00096|openflow-event|DBG:received packet-in event from 00232007f02c (len:60)
L3 resolver
IP status: None
level 2 learning
learned MAC a4:56:30:d1:91:02  on dpid 150861246508 and ingress port  4
NOTICE - Installing flow for: [a4:56:30:d1:91:02>a4:56:30:d1:91:02:9000]
NOTICE - Installing LOW priority flow for dpid 150861246508
```

```
00097|openflow-event|DBG:received packet-in event from 00232007f02c (len:60)
L3 resolver
IP status: None
level 2 learning
MAC is unchanged: a4:56:30:d1:91:02
NOTICE - Installing flow for: [a4:56:30:d1:91:02>01:00:0c:cc:cc:cc:llc]
NOTICE - Installing LOW priority flow for dpid 150861246508
00098|openflow-event|DBG:received packet-in event from 00232007f02c (len:90)
L3 resolver
IP status: None
level 2 learning
MAC is unchanged: a4:56:30:d1:91:02
NOTICE - Installing flow for: [a4:56:30:d1:91:02>01:00:0c:00:00:00:llc]
NOTICE - Installing LOW priority flow for dpid 150861246508
00099|openflow|DBG:stream: idle 15 seconds, sending inactivity probe
00100|openflow|DBG:stream: message received, entering CONNECTED
00101|openflow-event|DBG:received echo-request event from 002320f699a8 (len:0)
00102|openflow-event|DBG:received flow expired event from 00232007f02c
00103|openflow-event|DBG:received flow expired event from 00232007f02c
00104|openflow-event|DBG:received flow expired event from 00232007f02c
00105|openflow-event|DBG:received packet-in event from 00232007f02c (len:60)
L3 resolver
IP status: None
level 2 learning
learned MAC a4:56:30:d1:91:02  on dpid 150861246508 and ingress port  4
NOTICE - Installing flow for: [a4:56:30:d1:91:02>a4:56:30:d1:91:02:9000]
NOTICE - Installing LOW priority flow for dpid 150861246508
00106|openflow-event|DBG:received flow expired event from 00232007f02c
00107|openflow-event|DBG:received packet-in event from 00232007f02c (len:128)
L3 resolver
IP status: None
level 2 learning
MAC is unchanged: a4:56:30:d1:91:02
NOTICE - Installing flow for: [a4:56:30:d1:91:02>01:00:0c:cc:cc:cc:llc]
NOTICE - Installing LOW priority flow for dpid 150861246508
00108|openflow|DBG:stream: idle 15 seconds, sending inactivity probe
00109|openflow|DBG:stream: message received, entering CONNECTED
00110|openflow-event|DBG:received echo-request event from 002320f699a8 (len:0)
00111|openflow-event|DBG:received packet-in event from 00232007f02c (len:60)
L3 resolver
IP status: None
level 2 learning
learned MAC a4:56:30:d1:91:02  on dpid 150861246508 and ingress port  4
NOTICE - Installing flow for: [a4:56:30:d1:91:02>a4:56:30:d1:91:02:9000]
NOTICE - Installing LOW priority flow for dpid 150861246508
00112|openflow-event|DBG:received flow expired event from 00232007f02c
00113|openflow-event|DBG:received flow expired event from 00232007f02c
00114|openflow-event|DBG:received packet-in event from 00232007f02c (len:60)
L3 resolver
IP status: None
level 2 learning
learned MAC a4:56:30:d1:91:02  on dpid 150861246508 and ingress port  4
NOTICE - Installing flow for: [a4:56:30:d1:91:02>a4:56:30:d1:91:02:9000]
NOTICE - Installing LOW priority flow for dpid 150861246508
00115|openflow-event|DBG:received packet-in event from 00232007f02c (len:60)
L3 resolver
IP status: None
level 2 learning
MAC is unchanged: a4:56:30:d1:91:02
NOTICE - Installing flow for: [a4:56:30:d1:91:02>01:00:0c:cc:cc:cc:llc]
NOTICE - Installing LOW priority flow for dpid 150861246508
00116|openflow-event|DBG:received packet-in event from 00232007f02c (len:90)
L3 resolver
IP status: None
level 2 learning
MAC is unchanged: a4:56:30:d1:91:02
NOTICE - Installing flow for: [a4:56:30:d1:91:02>01:00:0c:00:00:00:llc]
```

```
NOTICE - Installing LOW priority flow for dpid 150861246508
```

Παρατηρούμε ότι γίνεται επιτυχής εγκατάστασης όλων των ροών χαμηλής προτεραιότητας καθώς εμφανίζεται το μήνυμα: `NOTICE - Installing LOW priority flow for dpid 150861246508.` Καθώς έχουμε ροές με χαμηλό χρόνο αναμονής, τώρα εμφανίζονται και μηνύματα λήξης χρόνου αναμονής ροής στον μεταγωγέα, όπως το: `00112|openflow-event|DBG:received flow expired event from 00232007f02c`

Για επαλήθευση της επιτυχούς εγκατάστασης των ροών στους μεταγωγείς OpenFlow του δικτύου, τρέχουμε το βοηθητικό πρόγραμμα dpctl σε καθένα από τους δύο μεταγωγείς και προκύπτουν τα πιο κάτω στιγμιότυπα:

1. Από τον of1 μεταγωγέα:

```
$ sudo dpctl dump-flows unix:/root/test
stats_reply (xid=0xf11d2f98): flags=none type=1(flow)
$
$
$
$
$ sudo dpctl dump-flows unix:/root/test
stats_reply (xid=0x81f9e15b): flags=none type=1(flow)
$
$
$
$
$ sudo dpctl dump-flows unix:/root/test
stats_reply (xid=0xf80e8415): flags=none type=1(flow)
$
```

Στιγμιότυπο 6.5-1 – Κατάσταση ροών χαμηλής προτεραιότητας στον μεταγωγέα of1

2. Από τον of2 μεταγωγέα:

```
$
$ sudo dpctl dump-flows unix:/var/run/test
stats_reply (xid=0xe41c5df6): flags=none type=1(flow)
  cookie=0,     duration_sec=1s,     duration_nsec=188000000s,     table_id=0,
priority=1,                      n_packets=1,                      n_bytes=60,
idle_timeout=6,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=a4:5
6:30:d1:91:02,dl_type=0x9000,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_pr
oto=0,tp_src=0,tp_dst=0,actions=output:3
  cookie=0,     duration_sec=169s,     duration_nsec=151000000s,     table_id=0,
priority=1,                     n_packets=85,                     n_bytes=5100,
idle_timeout=6,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=01:8
0:c2:00:00:00,dl_type=0x0026,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_pr
oto=0,tp_src=0,tp_dst=0,actions=output:3
$
$
```

```
$
$ sudo dpctl dump-flows unix:/var/run/test
stats_reply (xid=0x4e90f6c): flags=none type=1(flow)
  cookie=0,     duration_sec=5s,     duration_nsec=811000000s,     table_id=0,
priority=1,                      n_packets=1,                      n_bytes=60,
idle_timeout=6,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=a4:5
6:30:d1:91:02,dl_type=0x9000,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_pr
oto=0,tp_src=0,tp_dst=0,actions=output:3
  cookie=0,    duration_sec=173s,    duration_nsec=774000000s,    table_id=0,
priority=1,                    n_packets=87,                    n_bytes=5220,
idle_timeout=6,hard_timeout=0,in_port=4,dl_src=a4:56:30:d1:91:02,dl_dst=01:8
0:c2:00:00:00,dl_type=0x0026,nw_src=0.0.0.0,nw_dst=0.0.0.0,nw_tos=0x00,nw_pr
oto=0,tp_src=0,tp_dst=0,actions=output:3
$
```

Στιγμιότυπο 6.5-2 – Κατάσταση ροών χαμηλής προτεραιότητας στον μεταγωγέα of2

Όπως πριν, έτσι και τώρα δεν υπάρχει καμιά καταχώριση ροής στον of1 μεταγωγέα, αλλά παρατηρούμε μια σειρά επιτυχών εγκαταστάσεων ροών στον of2. Από την εικόνα 6.4-2 παρατηρούμε ότι όλες οι ροές που αφορούν την συσκευή χαμηλής προτεραιότητας `a4:56:30:d1:91:02` έχουν εγκατασταθεί επιτυχώς με τιμή προτεραιότητας ίση με 1 (priority=1) και χρόνο αναμονής (idle_timeout) έξι δευτερόλεπτα.

6.6 Ανακεφαλαίωση και τελικά συμπεράσματα

Η εφαρμογή της application-aware ctrlswl3.py σε δίκτυο OpenFlow μπορεί να δημιουργήσει αποκλειστικές (dedicated) ροές δεδομένων (dataflows) για μια εφαρμογή ή υπηρεσία που δραστηριοποιείται στο δίκτυο, όπως είναι για παράδειγμα η http. Με αυτό τον τρόπο επιτυγχάνεται η καλύτερη και αποδοτικότερη λειτουργία της εφαρμογής, με βέλτιστη χρήση των πόρων του υποκείμενου δικτύου.

Σε μεγαλύτερης έκτασης δίκτυα, όπως είναι τα εταιρικά, η ctrlswl3.py μπορεί να επεκταθεί, προσφέροντας υποστήριξη για πληθώρα εφαρμογών στο δίκτυο, όπως είναι η μεταφορά αρχείων με εφαρμογή FTP, υπηρεσιών ηλεκτρονικού ταχυδρομείου με λειτουργίες POP3 ή SMTP ή ακόμα εφαρμογές βάσεων δεδομένων (databases) με χρήση SQL Services ή θυρών του SQL Server.

Εκτός αυτού μπορεί να δημιουργηθούν και ροές δεδομένων μεταβλητής προτεραιότητας, όπως φάνηκε με την επιτυχή λειτουργία της εφαρμογής ctrlsw.py κατά την ανάπτυξη της στον ελεγκτή NOX. Βλέπουμε ότι η ανάθεση προτεραιότητας σε συσκευές ενωμένες σε ένα δίκτυο OpenFlow μπορεί να επιτευχθεί εύκολα με την χρήση δύο βασικών λειτουργιών του πρωτοκόλλου OpenFlow,του πεδίου προτεραιότητας (priority) και του χρόνου αναμονής (idle_timeout).

Με την χρήση αυτών των δύο πεδίων μπορεί να γίνει μια πιο αποτελεσματική και οικονομική χρήση των πόρων ενός δικτύου OpenFlow. Έτσι, συσκευές και εφαρμογές με υψηλές απαιτήσεις σε κίνηση και κρίσιμες για την λειτουργία που επιτελούν μπορούν να αποκτήσουν μόνιμες ροές στο δίκτυο με υψηλή προτεραιότητα αντιστοίχισης στα πακέτα τους, χωρίς να παρεμποδίζονται από άλλες εφαρμογές και συσκευές που να μην είναι κρίσιμης λειτουργίας.

Αντίστοιχα, για συσκευές που τρέχουν εφαρμογές επουσιώδους σημασίας για τους χρήστες του δικτύου, μπορεί να χρησιμοποιηθεί η λειτουργία χαμηλής προτεραιότητας. Σε αυτή την λειτουργία εξοικονομούνται πόροι του δικτύου, καθώς και ο χρόνος χρήσης αυτού, μια και ο χρόνος αναμονής των ροών χαμηλής προτεραιότητας είναι μόλις έξι δευτερόλεπτα. Για συσκευές που δεν υπάρχει πρόνοια από τον διαχειριστή του δικτύου, εγκαθίστανται αυτόματα από τον ελεγκτή ροές με προεπιλεγμένες ρυθμίσεις, σύμφωνα με την κανονική λειτουργία του πρωτοκόλλου OpenFlow.

Γενικά, μπορούμε να δούμε με την χρήση του ελεγκτή NOX και του πρωτοκόλλου OpenFlow το κύριο όφελος της SDN (Software Defined Networking) αρχιτεκτονικής, συγκριτικά με άλλες αρχιτεκτονικές δικτύου. Με την χρήση ενός

κεντρικού ελεγκτή οι διαχειριστές ενός δικτύου SDN μπορούν να επέμβουν άμεσα και να τροποποιήσουν με όποιο τρόπο επιθυμούν τις υπηρεσίες που παρέχει το δίκτυο, μέσω του ελεγκτή, χωρίς να είναι αναγκαία η χειροκίνητη ρύθμιση των παραμέτρων των συσκευών του δικτύου (μεταγωγέων, δρομολογητών ή τερματικών). Επίσης αναβαθμίζεται και η συνολική ασφάλεια του δικτύου, καθώς με ένα κεντρικό τρόπο ελέγχου μπορεί πιο εύκολα να ανιχνευθούν κακόβουλες ενέργειες ή λειτουργίες στο δίκτυο και να αντιμετωπιστούν άμεσα από τον διαχειριστή του δικτύου.

Φυσικά πρέπει να σημειωθεί και το βασικό μειονέκτημα αυτής της προσέγγισης, που είναι η κεντροποιημένη (centralized) μορφή του δικτύου. Μια οποιαδήποτε αστοχία του ελεγκτή (πρόβλημα στο διακομιστή, απώλεια σύνδεσης με τις υποκείμενες συσκευές του δικτύου) μπορεί να προκαλέσει μεγάλη αστάθεια στην λειτουργία του δικτύου, ειδικότερα για συσκευές που προσπαθούν να συνδεθούν στο δίκτυο με τον ελεγκτή να είναι εκτός λειτουργίας. Εν γένει, μια παρατεταμένη απώλεια λειτουργίας του ελεγκτή μπορεί να θέσει εκτός λειτουργίας όλο το δίκτυο.

Τέτοια όμως καταστροφικά σενάρια είναι σχετικά απίθανο να συμβούν, ειδικότερα σε σύγχρονα δίκτυα με δοκιμασμένο και αξιόπιστο εξοπλισμό. Λύσεις μπορούν να βρεθούν με χρήση μεθόδων πλεονασμού (redundancy), όπως είναι για παράδειγμα η ανάπτυξη ενός δευτερεύον ελεγκτή που να είναι εκτός λειτουργίας και να χρησιμοποιείται σε περιπτώσεις απώλειας του κανονικού ελεγκτή του δικτύου.

Η SDN αρχιτεκτονική είναι ακόμα μια νέα μέθοδος δημιουργίας και διαχείρισης δικτύων υπολογιστών, που ακόμη αναπτύσσεται και δοκιμάζεται. Με την πάροδο όμως του χρόνου τα πρωτόκολλα που την υποστηρίζουν (όπως το OpenFlow) γίνονται πιο ώριμα και αξιόπιστα, ενώ αναπτύσσονται όλο και περισσότεροι ελεγκτές σε διάφορες γλώσσες προγραμματισμού, πέραν του NOX, όπως ο Beacon ή ο Floodlight, που προσφέρουν όλο και περισσότερες δυνατότητες για την δημιουργία κεντροποιημένων, λειτουργικών και αξιόπιστων δικτύων υπολογιστών.

Βιβλιογραφία

[1] OpenFlow Specifications Sheet v1.1.0 2011
[2] Rob Sherwood, Glen Gibby, Kok-Kiong Yapy, Guido Appenzellery, Martin Casado, Nick McKeowny, Guru Parulkary *Flowvisor: A network virtualization layer* 2009
[3] www.noxrepo.org , www.openflow.org
[4] Openflow Switch Consortium: www.openflowswitch.org
[5] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan *Turner Openflow: Enabling Innovation in Campus Networks* 2008
[6] Martin Casado, Tal Garfinkel, Aditya Akella, Michael J. Freedman,Dan Boneh, Nick McKeown, Scott Shenker *SANE: A Protection Architecture for Enterprise Networks* 2008
[7] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, Rob Sherwood *On Controller Perormance in Software Defined Networks*
[8] Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller, Nick McKeown *Implementing an OpenFlow Switch on the NetFPGA platform 2008*
[9] Hilmi E. Egilmez, Burak Gorkemli, A. Murat Tekalp, Seyhan Civanlar *SCALABLE VIDEO STREAMING OVER OPENFLOW NETWORKS: AN OPTIMIZATION FRAMEWORK FOR QOS ROUTING*
[10] Natasha Gude , Teemu Koponen , Justin Pettit , Ben Pfaff , Martín Casado , Nick McKeown , Scott Shenker *NOX: Towards an Operating System for Networks*
[11] Saurav Das, Yiannis Yiakoumis, Guru Parulkar, Nick McKeown, Preeti Singh, Daniel Getachew, Premal Dinesh Desai *Application-Aware Aggregation and Traffic Engineering in a Converged Packet-Circuit Network*

ΠΑΡΑΡΤΗΜΑ Α

ΟΙ ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΔΙΕΠΑΦΕΣ (API's) ΤΟΥ ΕΛΕΓΚΤΗ NOX

- Παρακάτω επισυνάπτεται το περιεχόμενο του αρχείου core.py και του util.py που περιλαμβάνουν το σύνολο των βασικών διεπαφών του NOX.

1. Το αρχείο core.py που βρίσκεται στην θέση /usr/local/src/nox/src/nox/lib/ της προγραμματιστικής έκδοσης του ελεγκτή NOX.

```
# Copyright 2008 (C) Nicira, Inc.
#
# This file is part of NOX.
#
# NOX is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published
by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# NOX is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with NOX.  If not, see <http://www.gnu.org/licenses/>.

import logging
import array
import struct
import types
from socket import htons, htonl
import nox.lib.openflow as openflow

from nox.coreapps.pyrt.pycomponent import *
from util import *
from nox.lib.netinet.netinet import Packet_expr
from nox.lib.netinet.netinet import create_eaddr
from nox.lib.packet import *

lg = logging.getLogger('core')

IN_PORT    = "in_port"
AP_SRC     = "ap_src"
AP_DST     = "ap_dst"
DL_SRC     = "dl_src"
DL_DST     = "dl_dst"
```

```
DL_VLAN    = "dl_vlan"
DL_VLAN_PCP = "dl_vlan_pcp"
DL_TYPE    = "dl_type"
NW_SRC     = "nw_src"
NW_SRC_N_WILD = "nw_src_n_wild"
NW_DST     = "nw_dst"
NW_DST_N_WILD = "nw_dst_n_wild"
NW_PROTO   = "nw_proto"
NW_TOS     = "nw_tos"
TP_SRC     = "tp_src"
TP_DST     = "tp_dst"
GROUP_SRC  = "group_src"
GROUP_DST  = "group_dst"

N_TABLES    = 'n_tables'
N_BUFFERS   = 'n_bufs'
CAPABILITES = 'caps'
ACTIONS     = 'actions'
PORTS       = 'ports'
DL_VLAN_PCP = "dl_vlan_pcp"
DL_TYPE    = "dl_type"
NW_SRC     = "nw_src"
NW_SRC_N_WILD = "nw_src_n_wild"
NW_DST     = "nw_dst"
NW_DST_N_WILD = "nw_dst_n_wild"
NW_PROTO   = "nw_proto"
NW_TOS     = "nw_tos"
TP_SRC     = "tp_src"
TP_DST     = "tp_dst"
GROUP_SRC  = "group_src"
GROUP_DST  = "group_dst"

N_TABLES    = 'n_tables'
N_BUFFERS   = 'n_bufs'
CAPABILITES = 'caps'
ACTIONS     = 'actions'
PORTS       = 'ports'

PORT_NO     = 'port_no'
SPEED       = 'speed'
CONFIG      = 'config'
STATE       = 'state'
CURR        = 'curr'
ADVERTISED  = 'advertised'
SUPPORTED   = 'supported'
PEER        = 'peer'
HW_ADDR     = 'hw_addr'

################################################################################
############
# API NOTES:
#
# Automatically returns CONTINUE for handlers that do not
```

```
# return a value (handlers are supposed to return a Disposition)
#
# All values should be passed in host byte order.  The API changes
# values to network byte order based on knowledge of field.  (NW_SRC
# --> 32 bit val, TP_SRC --> 16 bit value, etc.).  Other than
# DL_SRC/DST and NW_SRC/DST fields, packet header fields should be
# passed as integers.  DL_SRC, DL_DST fields should be passed in as
# either vigil.netinet.ethernetaddr objects.  They can however also
be
# passed in any other type that an ethernetaddr constructor has be
# defined for.  NW_SRC/NW_DST fields meanwhile can be passed either
# ints, ip strings, or vigil.netinet.ipaddr objects.
##########################################################################
######

class Component:
    """Abstract class to inherited by all Python components.
    \ingroup noxapi
    """
    def __init__(self, ctxt):
        self.ctxt = ctxt
        self.component_names = None

    def configure(self, config):
        """Configure the component.
        Once configured, the component has parsed its configuration
and
        resolve any references to other components it may have.
        """
        pass

    def install(self):
        """Install the component.
        Once installed, the component runs and is usable by other
        components.
        """
        pass

    def getInterface(self):
        """Return the interface (class) component provides.  The
default
        implementation returns the class itself."""
        return self.__class__

    def resolve(self, interface):
        return self.ctxt.resolve(str(interface))

    # Interface to allow components to check at runtime without
having
    # to import them (which will cause linking errors)
    def is_component_loaded(self, name):
        if not self.component_names:
            self.component_names = []
```

```
            for component in self.ctxt.get_kernel().get_all():
                self.component_names.append(component.get_name())
        return name in self.component_names

    def register_event(self, event_name):
        return self.ctxt.register_event(event_name)

    def register_python_event(self, event_name):
        return self.ctxt.register_python_event(event_name)

    def register_handler(self, event_name, handler):
        return self.ctxt.register_handler(event_name, handler)

    def post_timer(self, event):
        return self.ctxt.post_timer(event)

    def post(self, event):
        # if event is a swigobject, make sure that it doesn't try
        # to handle memory deletion
        if hasattr(event, 'thisown'):
            event.thisown = 0 # just in case
        return self.ctxt.post(event)

    def make_action_array(self, actions):
        action_str = ""

        for action in actions:
            if action[0] == openflow.OFPAT_OUTPUT \
                    and isinstance(action[1],list) \
                    and len(action[1]) == 2:
                a = struct.pack("!HHHH", action[0], 8,
                                action[1][1], action[1][0])
            elif action[0] == openflow.OFPAT_SET_VLAN_VID:
                a = struct.pack("!HHHH", action[0], 8, action[1], 0)
            elif action[0] == openflow.OFPAT_SET_VLAN_PCP:
                a = struct.pack("!HHBBH", action[0], 8, action[1], 0,
0)
            elif action[0] == openflow.OFPAT_STRIP_VLAN:
                a = struct.pack("!HHI", action[0], 8, 0)
            elif action[0] == openflow.OFPAT_SET_DL_SRC \
                    or action[0] == openflow.OFPAT_SET_DL_DST:
                eaddr = convert_to_eaddr(action[1])
                if eaddr == None:
                    print 'invalid ethernet addr'
                    return None
                a = struct.pack("!HH6sHI", action[0], 16,
eaddr.binary_str(), 0, 0)
            elif action[0] == openflow.OFPAT_SET_NW_SRC \
                    or action[0] == openflow.OFPAT_SET_NW_DST:
                iaddr = convert_to_ipaddr(action[1])
                if iaddr == None:
                    print 'invalid ip addr'
                    return None
```

```
                a = struct.pack("HHI", htons(action[0]), htons(8),
htonl(ipaddr(iaddr).addr))
            elif action[0] == openflow.OFPAT_SET_TP_SRC \
                    or action[0] == openflow.OFPAT_SET_TP_DST:
                a = struct.pack("!HHHH", action[0], 8, action[1], 0)
            else:
                print 'invalid action type', action[0]
                return None

            action_str = action_str + a

        return action_str

    def send_port_mod(self, dpid, portno, hwaddr, mask, config):
        try:
            addr = create_eaddr(str(hwaddr))
            return self.ctxt.send_port_mod(dpid, portno, addr, mask,
config)
        except Exception, e:
            print e
            #lg.error("unable to send port mod"+str(e))

    def send_switch_command(self, dpid, command, arg_list):
        return self.ctxt.send_switch_command(dpid, command,
",".join(arg_list))

    def switch_reset(self, dpid):
        return self.ctxt.switch_reset(dpid)

    def switch_update(self, dpid):
        return self.ctxt.switch_update(dpid)


    def send_openflow_packet(self, dp_id, packet, actions,
                             inport=openflow.OFPP_CONTROLLER):
        """
        sends an openflow packet to a datapath

        dp_id - datapath to send packet to
        packet - data to put in openflow packet
        actions - list of actions or dp port to send out of
        inport - dp port to mark as source (defaults to Controller
port)
        """
        if type(packet) == type(array.array('B')):
            packet = packet.tostring()

        if type(actions) == types.IntType:
            self.ctxt.send_openflow_packet_port(dp_id, packet,
actions, inport)
        elif type(actions) == types.ListType:
            oactions = self.make_action_array(actions)
            if oactions == None:
```

```
                raise Exception('Bad action')
            self.ctxt.send_openflow_packet_acts(dp_id, packet,
oactions, inport)
        else:
            raise Exception('Bad argument')

    def send_openflow_buffer(self, dp_id, buffer_id, actions,
                             inport=openflow.OFPP_CONTROLLER):
        """
        Tells a datapath to send out a buffer

        dp_id - datapath to send packet to
        buffer_id - id of buffer to send out
        actions - list of actions or dp port to send out of
        inport - dp port to mark as source (defaults to Controller
port)
        """
        if type(actions) == types.IntType:
            self.ctxt.send_openflow_buffer_port(dp_id, buffer_id,
actions,
                                                inport)
        elif type(actions) == types.ListType:
            oactions = self.make_action_array(actions)
            if oactions == None:
                raise Exception('Bad action')
            self.ctxt.send_openflow_buffer_acts(dp_id, buffer_id,
oactions,
                                                inport)
        else:
            raise Exception('Bad argument')

    def post_callback(self, t, function):
        from twisted.internet import reactor
        reactor.callLater(t, function)

    def send_flow_command(self, dp_id, command, attrs,
                          priority=openflow.OFP_DEFAULT_PRIORITY,
                          add_args=None,
                          hard_timeout=openflow.OFP_FLOW_PERMANENT):
        m = set_match(attrs)
        if m == None:
            return False

        if command == openflow.OFPFC_ADD:
            (idle_timeout, actions, buffer_id) = add_args
            oactions = self.make_action_array(actions)
            if oactions == None:
                return False
        else:
            idle_timeout = 0
            oactions = ""
            buffer_id = UINT32_MAX
```

```
        self.ctxt.send_flow_command(dp_id, command, m, idle_timeout,
                                    hard_timeout, oactions,
buffer_id, priority)

        return True

    # Former PyAPI methods

    def send_openflow(self, dp_id, buffer_id, packet, actions,
                      inport=openflow.OFPP_CONTROLLER):
        """
        Sends an openflow packet to a datapath.

        This function is a convenient wrapper for
send_openflow_packet
        and send_openflow_buffer for situations where it is unknown
in
        advance whether the packet to be sent is buffered.  If
        'buffer_id' is -1, it sends 'packet'; otherwise, it sends the
        buffer represented by 'buffer_id'.

        dp_id - datapath to send packet to
        buffer_id - id of buffer to send out
        packet - data to put in openflow packet
        actions - list of actions or dp port to send out of
        inport - dp port to mark as source (defaults to Controller
                 port)
        """
        if buffer_id != None:
            self.send_openflow_buffer(dp_id, buffer_id, actions,
inport)
        else:
            self.send_openflow_packet(dp_id, packet, actions, inport)

    def delete_datapath_flow(self, dp_id, attrs):
        """
        Delete all flow entries matching the passed in (potentially
        wildcarded) flow

        dp_id - datapath to delete the entries from
        attrs - the flow as a dictionary (described above)
        """
        return self.send_flow_command(dp_id, openflow.OFPFC_DELETE,
attrs)

    def delete_strict_datapath_flow(self, dp_id, attrs,
                        priority=openflow.OFP_DEFAULT_PRIORITY):
        """
        Strictly delete the flow entry matching the passed in
(potentially
        wildcarded) flow.  i.e. matched flow have exactly the same
        wildcarded fields.
```

```
        dp_id - datapath to delete the entries from
        attrs - the flow as a dictionary (described above)
        priority - the priority of the entry to be deleted (only
meaningful
                   for entries with wildcards)
        """
        return self.send_flow_command(dp_id,
openflow.OFPFC_DELETE_STRICT,
                                      attrs, priority)


######################################################################
######
    # The following methods manipulate a flow entry in a datapath.
    # A flow is defined by a dictionary containing 0 or more of the
    # following keys (commented keys have already been defined
above):
    #

    # DL_SRC     = "dl_src"
    # DL_DST     = "dl_dst"
    # DL_VLAN    = "dl_vlan"
    # DL_VLAN_PCP = "dl_vlan_pcp"
    # DL_TYPE    = "dl_type"
    # NW_SRC     = "nw_src"
    # NW_DST     = "nw_dst"
    # NW_PROTO   = "nw_proto"
    # TP_SRC     = "tp_src"
    # TP_DST     = "tp_dst"
    #
    # Absent keys are interpretted as wildcards

######################################################################
######

    def install_datapath_flow(self, dp_id, attrs, idle_timeout,
hard_timeout,
                              actions, buffer_id=None,
                              priority=openflow.OFP_DEFAULT_PRIORITY,
                              inport=None, packet=None):
        """
        Add a flow entry to datapath

        dp_id - datapath to add the entry to

        attrs - the flow as a dictionary (described above)

        idle_timeout - # idle seconds before flow is removed from dp

        hard_timeout - # of seconds before flow is removed from dp

        actions - a list where each entry is a two-element list
representing
```

```
        an action.  Elem 0 of an action list should be an
ofp_action_type
        and elem 1 should be the action argument (if needed). For
        OFPAT_OUTPUT, this should be another two-element list with
max_len
        as the first elem, and port_no as the second

        buffer_id - the ID of the buffer to apply the action(s) to as
well.
        Defaults to None if the actions should not be applied to a
buffer

        priority - when wildcards are present, this value determines
the
        order in which rules are matched in the switch (higher values
        take precedence over lower ones)

        packet - If buffer_id is None, then a data packet to which
the
        actions should be applied, or None if none.

        inport - When packet is sent, the port on which packet came
in as input,
        so that it can be omitted from any OFPP_FLOOD outputs.
        """
        if buffer_id == None:
            buffer_id = UINT32_MAX

        self.send_flow_command(dp_id, openflow.OFPFC_ADD, attrs,
priority,
                               (idle_timeout, actions, buffer_id),
hard_timeout)

        if buffer_id == UINT32_MAX and packet != None:
            for action in actions:
                if action[0] == openflow.OFPAT_OUTPUT:
                    self.send_openflow_packet(dp_id, packet,
action[1][1], inport)
                else:
                    raise NotImplementedError

    def register_for_packet_in(self, handler):
        """
        register a handler to be called on every packet_in event
        handler will be called with the following args:

        handler(dp_id, inport, ofp_reason, total_frame_len,
buffer_id,
        captured_data)

        'buffer_id' == None if the datapath does not have a buffer
for
        the frame
```

```
        """
        self.register_handler(Packet_in_event.static_get_name(),
                              gen_packet_in_cb(handler))

    def register_for_flow_removed(self, handler):
        self.register_handler(Flow_removed_event.static_get_name(),
                              handler)

    def register_for_flow_mod(self, handler):
        self.register_handler(Flow_mod_event.static_get_name(),
                              handler)

    def register_for_bootstrap_complete(self, handler):

self.register_handler(Bootstrap_complete_event.static_get_name(),
                              handler)


########################################################################
###########
    # register a handler to be called on a every switch_features
event
    # handler will be called with the following args:
    #
    # handler(dp_id, attrs)
    #
    # attrs is a dictionary with the following keys:


    # the PORTS value is a list of port dictionaries where each
dictionary
    # has the keys listed in the register_for_port_status message

########################################################################
###########

    def register_for_datapath_join(self, handler):
        self.register_handler(Datapath_join_event.static_get_name(),
                              gen_dp_join_cb(handler))


########################################################################
###########
    # register a handler to be called whenever table statistics are
    # returned by a switch.
    #
    # handler will be called with the following args:
    #
    # handler(dp_id, stats)
    #
    # Stats is a dictionary of table stats with the following keys:
    #
    #   "table_id"
```

```
    #   "name"
    #   "max_entries"
    #   "active_count"
    #   "lookup_count"
    #   "matched_count"
    #
    # XXX
    #
    # We should get away from using strings here eventually.
    #

##########################################################################
###########

    def register_for_table_stats_in(self, handler):
        self.register_handler(Table_stats_in_event.static_get_name(),
                              gen_ts_in_cb(handler))


##########################################################################
###########
    # register a handler to be called whenever port statistics are
    # returned by a switch.
    #
    # handler will be called with the following args:
    #
    # handler(dp_id, stats)
    #
    # Stats is a dictionary of port stats with the following keys:
    #
    #   "port_no"
    #   "rx_packets"
    #   "tx_packets"
    #   "rx_bytes"
    #   "tx_bytes"
    #   "rx_dropped"
    #   "tx_dropped"
    #   "rx_errors"
    #   "tx_errors"
    #   "rx_frame_err"
    #   "rx_over_err"
    #   "rx_crc_err"
    #   "collisions"
    #

##########################################################################
###########

    def register_for_port_stats_in(self, handler):
        self.register_handler(Port_stats_in_event.static_get_name(),
                              gen_ps_in_cb(handler))
```

```
##########################################################################
###########
    # register a handler to be called whenever table aggregate
    # statistics are returned by a switch.
    #
    # handler will be called with the following args:
    #
    # handler(dp_id, stats)
    #
    # Stats is a dictionary of aggregate stats with the following
keys:
    #
    #   "packet_count"
    #   "byte_count"
    #   "flow_count"
    #

##########################################################################
###########

    def register_for_aggregate_stats_in(self, handler):

self.register_handler(Aggregate_stats_in_event.static_get_name(),
                              gen_as_in_cb(handler))



##########################################################################
###########
    # register a handler to be called whenever description
    # statistics are returned by a switch.
    #
    # handler will be called with the following args:
    #
    # handler(dp_id, stats)
    #
    # Stats is a dictionary of descriptions with the following keys:
    #
    #   "mfr_desc"
    #   "hw_desc"
    #   "sw_desc"
    #   "serial_num"
    #

##########################################################################
###########

    def register_for_desc_stats_in(self, handler):
        self.register_handler(Desc_stats_in_event.static_get_name(),
                              gen_ds_in_cb(handler))

    def register_for_datapath_leave(self, handler):
```

```
        """
        register a handler to be called on a every datapath_leave
        event handler will be called with the following args:

        handler(dp_id)
        """
        self.register_handler(Datapath_leave_event.static_get_name(),
                              gen_dp_leave_cb(handler))


##########################################################################
#####
    # register a handler to be called on a every port_status event
    # handler will be called with the following args:
    #
    # handler(dp_id, ofp_port_reason, attrs)
    #
    # attrs is a dictionary with the following keys:



##########################################################################
######
    def register_for_port_status(self, handler):
        self.register_handler(Port_status_event.static_get_name(),
                              gen_port_status_cb(handler))


##########################################################################
######
    # register a handler to be called on every packet_in event
matching
    # the passed in expression.
    #
    # priority - the priority the installed classifier rule should
have
    # expr - a dictionary containing 0 or more of the following keys.


    # Absent keys will be interpretted as wildcards (i.e. any value
is
    # accepted for those attributes when checking for a potential
match)
    #
    # handler will be called with the following args:
    #
    # handler(dp_id, inport, ofp_reason, total_frame_len, buffer_id,
    # captured_data)
    #
    # 'buffer_id' == None if the datapath does not have a buffer for
    # the frame
```

```
##########################################################################
######

    def register_for_packet_match(self, handler, priority, expr):
        e = Packet_expr()
        for key, val in expr.items():
            if key == AP_SRC:
                field = Packet_expr.AP_SRC
                val = htons(val)
            elif key == AP_DST:
                field = Packet_expr.AP_DST
                val = htons(val)
            elif key == DL_VLAN:
                field = Packet_expr.DL_VLAN
                val = htons(val)
            elif key == DL_VLAN_PCP:
                field = Packet_expr.DL_VLAN_PCP
                val = val
            elif key == DL_TYPE:
                field = Packet_expr.DL_TYPE
                val = htons(val)
            elif key == DL_SRC:
                field = Packet_expr.DL_SRC
                val = convert_to_eaddr(val)
                if val == None:
                    print 'invalid ethernet addr'
                    return False
            elif key == DL_DST:
                field = Packet_expr.DL_DST
                val = convert_to_eaddr(val)
                if val == None:
                    print 'invalid ethernet addr'
                    return False
            elif key == NW_SRC:
                field = Packet_expr.NW_SRC
                val = convert_to_ipaddr(val)
                if val == None:
                    print 'invalid ip addr'
                    return False
            elif key == NW_DST:
                field = Packet_expr.NW_DST
                val = convert_to_ipaddr(val)
                if val == None:
                    print 'invalid ip addr'
                    return False
            elif key == NW_PROTO:
                field = Packet_expr.NW_PROTO
            elif key == TP_SRC:
                field = Packet_expr.TP_SRC
                val = htons(val)
            elif key == TP_DST:
                field = Packet_expr.TP_DST
```

```
                val = htons(val)
            elif key == GROUP_SRC:
                field = Packet_expr.GROUP_SRC
                val = htonl(val)
            elif key == GROUP_DST:
                field = Packet_expr.GROUP_DST
                val = htonl(val)
            else:
                print 'invalid key', key
                return False

            if isinstance(val, ethernetaddr):
                e.set_eth_field(field, val)
            else:
                # check for max?
                if val > UINT32_MAX:
                    print 'value %u exceeds accepted range', val
                    return False
                e.set_uint32_field(field, val)

        return
self.ctxt.register_handler_on_match(gen_packet_in_cb(handler),
priority, e)

    def register_for_switch_mgr_join(self, handler):
        """
        register a handler to be called on every switch_mgr_join
        event handler will be called with the following args:

        handler(mgmt_id)
        """

self.register_handler(Switch_mgr_join_event.static_get_name(),
                              gen_switch_mgr_join_cb(handler))

    def register_for_switch_mgr_leave(self, handler):
        """
        register a handler to be called on every switch_mgr_leave
        event handler will be called with the following args:

        handler(mgmt_id)
        """

self.register_handler(Switch_mgr_leave_event.static_get_name(),
                              gen_switch_mgr_leave_cb(handler))

    def unregister_handler(self, rule_id):
        """
        Unregister a handler for match.
        """
        return self.ctxt.register_handler(event_type, event_name,
handler)
```

2. Το αρχείο util.py που βρίσκεται στην θέση /usr/local/src/nox/src/nox/lib/ της προγραμματιστικής έκδοσης του ελεγκτή NOX.

```
# Copyright 2008 (C) Nicira, Inc.
#
# This file is part of NOX.
#
# NOX is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published
by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# NOX is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with NOX.  If not, see <http://www.gnu.org/licenses/>.

import core
import array
from socket import htons

import nox.lib.openflow as openflow

from   nox.coreapps.pyrt.pycomponent import UINT32_MAX
from   nox.coreapps.pyrt.pycomponent import CONTINUE

from   nox.lib.packet.packet_exceptions import IncompletePacket

from nox.lib.netinet.netinet import ipaddr, create_ipaddr,
ethernetaddr, create_bin_eaddr, create_eaddr, c_ntohl, c_htonl

from   nox.lib.packet.ethernet import ethernet
from   nox.lib.packet.vlan import vlan
from   nox.lib.packet.ipv4 import ipv4
from   nox.lib.packet.udp import udp
from   nox.lib.packet.tcp import tcp
from   nox.lib.packet.icmp import icmp

from twisted.python import log

##########################################################################
##########
# HELPER FUNCTIONS (NOT REQUIRING ACCESS TO THE CORE)
##########################################################################
##########

def __dladdr_check__(addr):
    if isinstance(addr, basestring) and not (':' in addr):
```

```
        if len(addr) != ethernetaddr.LEN:
            return None
        addr = create_bin_eaddr(addr)
    elif isinstance(addr, basestring) or ((isinstance(addr, long) or
isinstance(addr, int)) and addr >= 0):
        addr = create_eaddr(addr)
    elif not isinstance(addr, ethernetaddr):
        return None

    if addr == None:
        return None

    return addr.hb_long()

def __nwaddr_check__(addr):
    # should check if defined constant here

    if ((isinstance(addr, int) or isinstance(addr, long)) and addr >=
0 and addr <= 0xffffffff) \
            or isinstance(addr, basestring):
        addr = create_ipaddr(addr)
    elif not  isinstance(addr, ipaddr):
        return None

    if addr == None:
        return None

    return c_ntohl(addr.addr)

def convert_to_eaddr(val):
    if isinstance(val, ethernetaddr):
        return val
    if isinstance(val, array.array):
        val = val.tostring()
    if isinstance(val, str) and not (':' in val):
        if len(val) < ethernetaddr.LEN:
            return None
        return create_bin_eaddr(val)
    elif isinstance(val, str) or isinstance(val, int) or
isinstance(val, long):
        return create_eaddr(val)
    return None

def convert_to_ipaddr(val):
    if isinstance(val, ipaddr):
        return val.addr
    if isinstance(val, str) or isinstance(val, int) or
isinstance(val, long):
        a = create_ipaddr(val)
        if a != None:
            return a.addr
    return None
```

```
def gen_packet_in_cb(handler):
    def f(event):
        if event.reason == openflow.OFPR_NO_MATCH:
            reason = openflow.OFPR_NO_MATCH
        elif event.reason == openflow.OFPR_ACTION:
            reason = openflow.OFPR_ACTION
        else:
            print 'packet_in reason type %u unknown...just passing
along' % event.reason
            reason = event.reason

        if event.buffer_id == UINT32_MAX:
            buffer_id = None
        else:
            buffer_id = event.buffer_id

        try:
            packet = ethernet(array.array('B', event.buf))
        except IncompletePacket, e:
            log.err('Incomplete Ethernet header',system='pyapi')

        ret = handler(event.datapath_id, event.in_port, reason,
                      event.total_len, buffer_id, packet)
        if ret == None:
            return CONTINUE
        return ret
    return f

def gen_dp_join_cb(handler):
    def f(event):
        attrs = {}
        attrs[core.N_BUFFERS] = event.n_buffers
        attrs[core.N_TABLES] = event.n_tables
        attrs[core.CAPABILITES] = event.capabilities
        attrs[core.ACTIONS] = event.actions
        attrs[core.PORTS] = event.ports
        for i in range(0, len(attrs[core.PORTS])):
            port = attrs[core.PORTS][i]
            config = port['config']
            state = port['state']
            attrs[core.PORTS][i]['link']    = (state &
openflow.OFPPS_LINK_DOWN) == 0
            attrs[core.PORTS][i]['enabled'] = (config &
openflow.OFPPC_PORT_DOWN) == 0
            attrs[core.PORTS][i]['flood']   = (config &
openflow.OFPPC_NO_FLOOD)  == 0

        ret = f.cb(event.datapath_id, attrs)
        if ret == None:
            return CONTINUE
        return ret
    f.cb = handler
    return f
```

```
def gen_dp_leave_cb(handler):
    def f(event):
        ret = f.cb(event.datapath_id)
        if ret == None:
            return CONTINUE
        return ret
    f.cb = handler
    return f

def gen_ds_in_cb(handler):
    def f(event):
        stats = {}
        stats['mfr_desc'] = event.mfr_desc
        stats['hw_desc'] = event.hw_desc
        stats['sw_desc'] = event.sw_desc
        stats['dp_desc'] = event.dp_desc
        stats['serial_num'] = event.serial_num
        ret = f.cb(event.datapath_id, stats)
        if ret == None:
            return CONTINUE
        return ret
    f.cb = handler
    return f

def gen_as_in_cb(handler):
    def f(event):
        stats = {}
        stats['packet_count'] = event.packet_count
        stats['byte_count'] = event.byte_count
        stats['flow_count'] = event.flow_count
        ret = f.cb(event.datapath_id, stats)
        if ret == None:
            return CONTINUE
        return ret
    f.cb = handler
    return f

def gen_ps_in_cb(handler):
    def f(event):
        ports = event.ports
        ret = f.cb(event.datapath_id, ports)
        if ret == None:
            return CONTINUE
        return ret
    f.cb = handler
    return f

def gen_ts_in_cb(handler):
    def f(event):
        tables = event.tables
        ret = f.cb(event.datapath_id, tables)
        if ret == None:
```

```
            return CONTINUE
        return ret
    f.cb = handler
    return f

def gen_port_status_cb(handler):
    def f(event):
        if event.reason == openflow.OFPPR_ADD:
            reason = openflow.OFPPR_ADD
        elif event.reason == openflow.OFPPR_DELETE:
            reason = openflow.OFPPR_DELETE
        elif event.reason == openflow.OFPPR_MODIFY:
            reason = openflow.OFPPR_MODIFY
        else:
            print 'port_status reason type %u unknown...just passing
along' % event.reason
            reason = event.reason

        config = event.port['config']
        state = event.port['state']
        event.port['link']    = (state & openflow.OFPPS_LINK_DOWN) ==
0
        event.port['enabled'] = (config & openflow.OFPPC_PORT_DOWN)
== 0
        event.port['flood']   = (config & openflow.OFPPC_NO_FLOOD)
== 0

        ret = f.cb(event.datapath_id, reason, event.port)
        if ret == None:
            return CONTINUE
        return ret
    f.cb = handler
    return f

def gen_switch_mgr_join_cb(handler):
    def f(event):
        ret = f.cb(event.mgmt_id)
        if ret == None:
            return CONTINUE
        return ret
    f.cb = handler
    return f

def gen_switch_mgr_leave_cb(handler):
    def f(event):
        ret = f.cb(event.mgmt_id)
        if ret == None:
            return CONTINUE
        return ret
    f.cb = handler
    return f

def set_match(attrs):
```

```
m = openflow.ofp_match()
wildcards = 0
num_entries = 0

if attrs.has_key(core.IN_PORT):
    m.in_port = htons(attrs[core.IN_PORT])
    num_entries += 1
else:
    wildcards = wildcards | openflow.OFPFW_IN_PORT

if attrs.has_key(core.DL_VLAN):
    m.dl_vlan = htons(attrs[core.DL_VLAN])
    num_entries += 1
else:
    wildcards = wildcards | openflow.OFPFW_DL_VLAN

if attrs.has_key(core.DL_VLAN_PCP):
    m.dl_vlan_pcp = attrs[core.DL_VLAN_PCP]
    num_entries += 1
else:
    wildcards = wildcards | openflow.OFPFW_DL_VLAN_PCP

if attrs.has_key(core.DL_SRC):
    v = convert_to_eaddr(attrs[core.DL_SRC])
    if v == None:
        print 'invalid ethernet addr'
        return None
    m.set_dl_src(v.octet)
    num_entries += 1
else:
    wildcards = wildcards | openflow.OFPFW_DL_SRC

if attrs.has_key(core.DL_DST):
    v = convert_to_eaddr(attrs[core.DL_DST])
    if v == None:
        print 'invalid ethernet addr'
        return None
    m.set_dl_dst(v.octet)
    num_entries += 1
else:
    wildcards = wildcards | openflow.OFPFW_DL_DST

if attrs.has_key(core.DL_TYPE):
    m.dl_type = htons(attrs[core.DL_TYPE])
    num_entries += 1
else:
    wildcards = wildcards | openflow.OFPFW_DL_TYPE

if attrs.has_key(core.NW_SRC):
    v = convert_to_ipaddr(attrs[core.NW_SRC])
    if v == None:
        print 'invalid ip addr'
        return None
```

```
        m.nw_src = v
        num_entries += 1

        if attrs.has_key(core.NW_SRC_N_WILD):
            n_wild = attrs[core.NW_SRC_N_WILD]
            if n_wild > 31:
                wildcards |= openflow.OFPFW_NW_SRC_MASK
            elif n_wild >= 0:
                wildcards |= n_wild << openflow.OFPFW_NW_SRC_SHIFT
            else:
                print 'invalid nw_src wildcard bit count', n_wild
                return None
            num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_NW_SRC_MASK

    if attrs.has_key(core.NW_DST):
        v = convert_to_ipaddr(attrs[core.NW_DST])
        if v == None:
            print 'invalid ip addr'
            return None
        m.nw_dst = v
        num_entries += 1

        if attrs.has_key(core.NW_DST_N_WILD):
            n_wild = attrs[core.NW_DST_N_WILD]
            if n_wild > 31:
                wildcards |= openflow.OFPFW_NW_DST_MASK
            elif n_wild >= 0:
                wildcards |= n_wild << openflow.OFPFW_NW_DST_SHIFT
            else:
                print 'invalid nw_dst wildcard bit count', n_wild
                return None
            num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_NW_DST_MASK

    if attrs.has_key(core.NW_PROTO):
        m.nw_proto = attrs[core.NW_PROTO]
        num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_NW_PROTO

    if attrs.has_key(core.NW_TOS):
        m.nw_tos = attrs[core.NW_TOS]
        num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_NW_TOS

    if attrs.has_key(core.TP_SRC):
        m.tp_src = htons(attrs[core.TP_SRC])
        num_entries += 1
    else:
```

```
        wildcards = wildcards | openflow.OFPFW_TP_SRC

    if attrs.has_key(core.TP_DST):
        m.tp_dst = htons(attrs[core.TP_DST])
        num_entries += 1
    else:
        wildcards = wildcards | openflow.OFPFW_TP_DST

    if num_entries != len(attrs.keys()):
        print 'undefined flow attribute type in attrs', attrs
        return None

    m.wildcards = c_htonl(wildcards)
    return m

def extract_flow(ethernet):
    """
    Extracts and returns flow attributes from the given 'ethernet'
packet.
    The caller is responsible for setting IN_PORT itself.
    """
    attrs = {}
    attrs[core.DL_SRC] = ethernet.src
    attrs[core.DL_DST] = ethernet.dst
    attrs[core.DL_TYPE] = ethernet.type
    p = ethernet.next

    if isinstance(p, vlan):
        attrs[core.DL_VLAN] = p.id
        attrs[core.DL_VLAN_PCP] = p.pcp
        p = p.next
    else:
        attrs[core.DL_VLAN] = 0xffff # XXX should be written
OFP_VLAN_NONE
        attrs[core.DL_VLAN_PCP] = 0

    if isinstance(p, ipv4):
        attrs[core.NW_SRC] = p.srcip
        attrs[core.NW_DST] = p.dstip
        attrs[core.NW_PROTO] = p.protocol
        p = p.next

        if isinstance(p, udp) or isinstance(p, tcp):
            attrs[core.TP_SRC] = p.srcport
            attrs[core.TP_DST] = p.dstport
        else:
            if isinstance(p, icmp):
                attrs[core.TP_SRC] = p.type
                attrs[core.TP_DST] = p.code
            else:
                attrs[core.TP_SRC] = 0
                attrs[core.TP_DST] = 0
    else:
```

```
        attrs[core.NW_SRC] = 0
        attrs[core.NW_DST] = 0
        attrs[core.NW_PROTO] = 0
        attrs[core.TP_SRC] = 0
        attrs[core.TP_DST] = 0
    return attrs
```

ΠΑΡΑΡΤΗΜΑ Β

ΟΙ ΒΑΣΙΚΕΣ ΕΦΑΡΜΟΓΕΣ (COREAPPS) ΤΟΥ ΕΛΕΓΚΤΗ NOX ΚΑΙ ΟΙ APPLICATION-DEVICE AWARE ΕΦΑΡΜΟΓΕΣ

- Παρακάτω επισυνάπτεται η εφαρμογή pyswitch.py που υπάρχει στο αρχείο coreapps/examples του ελεγκτή NOX και οι εφαρμογές ctrlswl3.py (application-aware) και ctrlsw.py (device-aware) που αναπτύχθηκαν στα πλαίσια της διπλωματικής εργασίας.

1. Η pyswitch.py

```
# Copyright 2008 (C) Nicira, Inc.
#
# This file is part of NOX.
#
# NOX is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published
# by the Free Software Foundation, either version 3 of the License,
# or (at your option) any later version.
#
# NOX is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with NOX.  If not, see <http://www.gnu.org/licenses/>.
# Python L2 learning switch
#
# ------------------------------------------------------------------
#
# This app functions as the control logic of an L2 learning switch
# for all switches in the network. On each new switch join, it
# creates an L2 MAC cache for that switch.
#
# In addition to learning, flows are set up in the switch for learned
# destination MAC addresses.  Therefore, in the absence of flow-
# timeout, pyswitch should only see one packet per flow (where flows
# are considered to be unidirectional)
#

from nox.lib.core     import *

from nox.lib.packet.ethernet     import ethernet
from nox.lib.packet.packet_utils import mac_to_str, mac_to_int

from twisted.python import log

import logging
from time import time
from socket import htons
from struct import unpack
```

```
logger = logging.getLogger('nox.coreapps.examples.pyswitch')

# Global pyswitch instance
inst = None

# Timeout for cached MAC entries
CACHE_TIMEOUT = 5

# --
# Given a packet, learn the source and peg to a switch/inport
# --
def do_l2_learning(dpid, inport, packet):
    global inst

    # learn MAC on incoming port
    srcaddr = packet.src.tostring()
    if ord(srcaddr[0]) & 1:
        return
    if inst.st[dpid].has_key(srcaddr):
        dst = inst.st[dpid][srcaddr]
        if dst[0] != inport:
            log.msg('MAC has moved from '+str(dst)+'to'+str(inport),
system='pyswitch')
        else:
            return
    else:
        log.msg('learned MAC '+mac_to_str(packet.src)+' on %d %d'%
(dpid,inport), system="pyswitch")

    # learn or update timestamp of entry
    inst.st[dpid][srcaddr] = (inport, time(), packet)

    # Replace any old entry for (switch,mac).
    mac = mac_to_int(packet.src)

# --
# If we've learned the destination MAC set up a flow and
# send only out of its inport.  Else, flood.
# --
def forward_l2_packet(dpid, inport, packet, buf, bufid):
    dstaddr = packet.dst.tostring()
    if not ord(dstaddr[0]) & 1 and inst.st[dpid].has_key(dstaddr):
        prt = inst.st[dpid][dstaddr]
        if  prt[0] == inport:
            log.err('**warning** learned port = inport',
system="pyswitch")
            inst.send_openflow(dpid, bufid, buf, openflow.OFPP_FLOOD,
inport)
        else:
            # We know the outport, set up a flow
            log.msg('installing flow for ' + str(packet),
system="pyswitch")
            flow = extract_flow(packet)
            flow[core.IN_PORT] = inport
            actions = [[openflow.OFPAT_OUTPUT, [0, prt[0]]]]
            inst.install_datapath_flow(dpid, flow, CACHE_TIMEOUT,
                        openflow.OFP_FLOW_PERMANENT, actions,
                        bufid, openflow.OFP_DEFAULT_PRIORITY,
                        inport, buf)
    else:
        # haven't learned destination MAC. Flood
```

```
        inst.send_openflow(dpid, bufid, buf, openflow.OFPP_FLOOD,
inport)

# --
# Responsible for timing out cache entries.
# Is called every 1 second.
# --
def timer_callback():
    global inst

    curtime  = time()
    for dpid in inst.st.keys():
        for entry in inst.st[dpid].keys():
            if (curtime - inst.st[dpid][entry][1]) > CACHE_TIMEOUT:
                log.msg('timing out
entry'+mac_to_str(entry)+str(inst.st[dpid][entry])+' on switch %x' %
dpid, system='pyswitch')
                inst.st[dpid].pop(entry)

    inst.post_callback(1, timer_callback)
    return True

def datapath_leave_callback(dpid):
    logger.info('Switch %x has left the network' % dpid)
    if inst.st.has_key(dpid):
        del inst.st[dpid]

def datapath_join_callback(dpid, stats):
    logger.info('Switch %x has joined the network' % dpid)

# --
# Packet entry method.
# Drop LLDP packets (or we get confused) and attempt learning and
# forwarding
# --
def packet_in_callback(dpid, inport, reason, len, bufid, packet):

    if not packet.parsed:
        log.msg('Ignoring incomplete packet',system='pyswitch')

    if not inst.st.has_key(dpid):
        log.msg('registering new switch %x' % dpid,system='pyswitch')
        inst.st[dpid] = {}

    # don't forward lldp packets
    if packet.type == ethernet.LLDP_TYPE:
        return CONTINUE

    # learn MAC on incoming port
    do_l2_learning(dpid, inport, packet)

    forward_l2_packet(dpid, inport, packet, packet.arr, bufid)

    return CONTINUE

class pyswitch(Component):

    def __init__(self, ctxt):
        global inst
        Component.__init__(self, ctxt)
        self.st = {}
```

```
        inst = self

    def install(self):
        inst.register_for_packet_in(packet_in_callback)
        inst.register_for_datapath_leave(datapath_leave_callback)
        inst.register_for_datapath_join(datapath_join_callback)
        inst.post_callback(1, timer_callback)

    def getInterface(self):
        return str(pyswitch)

def getFactory():
    class Factory:
        def instance(self, ctxt):
            return pyswitch(ctxt)

    return Factory()
```

2. Η application-aware εφαρμογή ctrlswl3.py που αναπτύχθηκε στα πλαίσια της διπλωματικής εργασίας.

```
0001 from nox.lib.core import *
0002
0003 from nox.lib.packet.ethernet import ethernet
0004 from nox.lib.packet.packet_utils import mac_to_str, mac_to_int
0005
0006 from twisted.python import log
0007
0008 import logging
0009 from time import time
0010 from socket import htons
0011 from struct import unpack
0012
0013
0014 from nox.lib.packet.ethernet import ethernet
0015 from nox.lib.packet.vlan import vlan
0016 from nox.lib.packet.ipv4 import ipv4
0017 from nox.lib.packet.udp import udp
0018 from nox.lib.packet.tcp import tcp
0019 from nox.lib.packet.icmp import icmp
0020
0021
0022 from nox.lib.netinet.netinet import ipaddr, create_ipaddr,
ethernetaddr, create_bin_eaddr, create_eaddr, c_ntohl, c_htonl
0023
0024
0025 logger = logging.getLogger('nox.coreapps.examples.ctrlswl3')
0026
0027
0028 # Global pyswitch instance and MAC, IP lists
0029 inst = None
0030 macs_str_hp = [0]
0031 macs_str_lp = [0]
0032 macs_str = [0]
0033 iplst_src = [0]
0034 iplst_dst_hp = [0]
0035 iplst_dst_lp = [0]
0036
```

```
0037
0038 # Timeout for cached MAC entries
0039
0040 CACHE_TIMEOUT = 6 #idle default is 5 sec - time before a flow
expires if there is no activity - alier
0041
0042 CACHE_PERMFLOW= 60 #increased standby for high priority flows
before expiration
0043
0044
0045
0046 def create_l3_out_flow(ethernet):
0047     attrs = {}
0048     attrs[core.DL_DST] = ethernet.dst
0049     attrs[core.DL_SRC] = ethernet.src
0050     attrs[core.DL_TYPE] = ethernet.type
0051     p = ethernet.next
0052
0053
0054     if isinstance(p, ipv4):
0055         attrs[core.NW_SRC] = p.srcip
0056         attrs[core.NW_DST] = p.dstip
0057         attrs[core.NW_PROTO] = p.protocol
0058         p = p.next
0059         if isinstance(p, udp) or isinstance(p, tcp):
0060            attrs[core.TP_SRC] = p.srcport
0061            attrs[core.TP_DST] = p.dstport
0062         else:
0063             if isinstance(p, icmp):
0064                 attrs[core.TP_SRC] = p.type
0065                 attrs[core.TP_DST] = p.code
0066             else:
0067                 attrs[core.TP_SRC] = 0
0068                 attrs[core.TP_DST] = 0
0069     else:
0070         attrs[core.NW_SRC] = 0
0071         attrs[core.NW_DST] = 0
0072         attrs[core.NW_PROTO] = 0
0073         attrs[core.TP_SRC] = 0
0074         attrs[core.TP_DST] = 0
0075     return attrs
0076
0077
0078 def do_l3_learning(dpid, inport, packet):
0079     global inst
0080     global iplst_src
0081     global iplst_dst
0082
0083     print 'L3 resolver. Scanning for valid IP addresses before
processing.'
0084
0085     pt=packet.next
0086     if isinstance(pt, ipv4):
0087         ip_p_src = pt.srcip
0088         ip_p_dst = pt.dstip
0089         iplst_src.append(ip_p_src)
0090         iplst_dst.append(ip_p_dst)
0091         print 'IP scanning succesful.'
0092         print iplst_src,iplst_dst
0093         return
0094     else:
```

```
0095     print 'IP detection failed. Checking current field status..'
0096    if isinstance(packet.next, str) or isinstance(packet.next,
int) or isinstance(packet.next, long):
0097         a = create_ipaddr(packet.next)
0098         print 'IP status:', a
0099         srcaddr = packet.src.tostring()
0100         print 'Fallback. Application has not checked out.Level 2
learning..'
0101    if ord(srcaddr[0]) & 1:
0102         print 'Source MAC address is 0.'
0103         return
0104    if inst.st[dpid].has_key(srcaddr):
0105         dst = inst.st[dpid][srcaddr]
0106         if dst[0] != inport:
0107             log.msg('MAC has moved from
'+str(dst)+'to'+str(inport), system='ctrlswl3')
0108         else:
0109             print 'MAC is unchanged:',mac_to_str(packet.src)
0110             return
0111    else:
0112         log.msg('learned MAC '+mac_to_str(packet.src)+' on %d
%d'% (dpid,inport), system="ctrlswl3")
0113         print 'learned MAC',mac_to_str(packet.src),' on
dpid',dpid,'and ingress port ',inport
0114
0115    # learn or update timestamp of entry
0116    inst.st[dpid][srcaddr] = (inport, time(), packet)
0117    # Replace any old entry for (switch,mac).
0118    mac = mac_to_int(packet.src)
0119
0120
0121 def forward_l3_packet(dpid, inport, packet, buf, bufid):
0122   dstaddr = packet.dst.tostring()
0123   pt=packet.next
0124
0125   if not ord(dstaddr[0]) & 1 and inst.st[dpid].has_key(dstaddr):
0126         prt = inst.st[dpid][dstaddr]
0127   else:
0128    return
0129
0130   if isinstance(pt, ipv4):
0131         pt=pt.next
0132    if isinstance(p, udp) or isinstance(p, tcp):
0133           appsrcprt = pt.srcport
0134           appdstprt = pt.dstport
0135         else:
0136      print 'No valid TCP/UDP protocol (http) detected.
Flooding..'
0137           inst.send_openflow(dpid, bufid, buf,
openflow.OFPP_FLOOD, inport)
0138           return
0139         if prt[0] == inport: #sendport
0140            log.err('**Warning** learned port = inport',
system="ctrlswl3")
0141            inst.send_openflow(dpid, bufid, buf,
openflow.OFPP_FLOOD, inport)
0142            return
0143         else:
0144           flow = create_l3_out_flow(packet)
0145           flow[core.IN_PORT] = inport
```

```
0146            # actions = [[openflow.OFPAT_OUTPUT, [0, port[0]]]]
#sendport - alier
0147            actions = [[openflow.OFPAT_OUTPUT, [0, 3]]] #sendport
- alier
0148            print 'Searching for valid appication (http)
port..',str(packet)
0149        if appdstprt==80:
0150          print 'NOTICE - Valid port detected. Installing
Appropriate flow to accommodate the application.'
0151              inst.install_datapath_flow(dpid,flow,CACHE_PERMFLOW,
0152                                         openflow.OFP_FLOW_PERMANENT,actions,
0153                                  bufid, openflow.OFP_DEFAULT_PRIORITY+32767,
0154                                               inport, buf)
0155              return
0156            else:
0157          print 'No valid application (http) port. Flooding..'
0158              inst.send_openflow(dpid, bufid, buf, openflow.OFPP_FLOOD,
inport)
0159          return
0160    else:
0161      print 'No valid IP address (http) detected. Flooding..'
0162      inst.send_openflow(dpid, bufid, buf, openflow.OFPP_FLOOD,
inport)
0163
0164
0165 # -- Responsible for timing out cache entries. Is called every 1
second. --
0166
0167
0168 def timer_callback():
0169      global inst
0170      curtime = time()
0171
0172      for dpid in inst.st.keys():
0173          for entry in inst.st[dpid].keys():
0174              if (curtime - inst.st[dpid][entry][1]) >
CACHE_TIMEOUT:
0175                  log.msg('timing out
entry'+mac_to_str(entry)+str(inst.st[dpid][entry])+' on switch %x' %
dpid, system='ctrlswl3')
0176                  inst.st[dpid].pop(entry)
0177
0178      inst.post_callback(1, timer_callback)
0179      return True
0180
0181 def datapath_leave_callback(dpid):
0182      logger.info('Switch %x has left the network' % dpid)
0183      if inst.st.has_key(dpid):
0184          del inst.st[dpid]
0185
0186
0187 def datapath_join_callback(dpid, stats):
0188      logger.info('Switch %x has joined the network' % dpid)
0189
0190
0191 # -- Packet entry method. Drop LLDP packets (or we get confused)
and attempt learning and forwarding --
0192
0193 def packet_in_callback(dpid, inport, reason, len, bufid,
packet):
0194      global iplst_dst_hp
0195      global iplst_dst_lp
```

```
0196
0197
0198      if not packet.parsed:
0199          log.msg('Ignoring incomplete packet',system='ctrlswl3')
0200
0201
0202      if not inst.st.has_key(dpid):
0203          log.msg('registering new switch %x' %
dpid,system='ctrlswl3')
0204          inst.st[dpid] = {}
0205
0206      # don't forward lldp packets
0207
0208      if packet.type == ethernet.LLDP_TYPE:
0209          print 'lldp type packet'
0210          return CONTINUE
0211
00212       # learn MAC on incoming port
0213
0214      do_l3_learning(dpid, inport, packet)
0215
0216      forward_l3_packet(dpid, inport, packet, packet.arr, bufid)
0217
0218      return CONTINUE
0219
0220
0221
0222 class ctrlswl3(Component):
0223
0224     def __init__(self, ctxt):
0225          global inst
0226          Component.__init__(self, ctxt)
0227          self.st = {}
0228          inst = self
0229
0230     def install(self):
0231          inst.register_for_packet_in(packet_in_callback)
0232         inst.register_for_datapath_leave(datapath_leave_callback)
0233          inst.register_for_datapath_join(datapath_join_callback)
0234          inst.post_callback(1, timer_callback)
0235
0236     def getInterface(self):
0237          return str(ctrlswl3)
0238
0239 def getFactory():
0240      class Factory:
0241          def instance(self, ctxt):
0242              return ctrlswl3(ctxt)
0243      return Factory()
```

3. Η device –aware ctrlsw.py που αναπτύχθηκε στα πλαίσια της διπλωματικής εργασίας.

```
0001 from nox.lib.core     import *
0002
0003 from nox.lib.packet.ethernet     import ethernet
```

```
0004 from nox.lib.packet.packet_utils import mac_to_str, mac_to_int
0005
0006 from twisted.python import log
0007
0008 import logging
0009 from time import time
0010 from socket import htons
0011 from struct import unpack
0012
0013 from   nox.lib.packet.ethernet import ethernet
0014 from   nox.lib.packet.vlan import vlan
0015 from   nox.lib.packet.ipv4 import ipv4
0016 from   nox.lib.packet.udp import udp
0017 from   nox.lib.packet.tcp import tcp
0018 from   nox.lib.packet.icmp import icmp
0019
0020 logger = logging.getLogger('nox.coreapps.examples.ctrlsw')
0021
0022 # Global pyswitch instance
0023 inst = None
0024 macs_str_hp = [0]
0025 macs_str_lp = [0]
0026 macs_str = [0]
0027
0028 # Timeout for cached MAC entries
0029 CACHE_TIMEOUT = 6 #idle default is 5 sec
0030 CACHE_PERMFLOW= 60 #increased standby
0031
0032 def create_l3_out_flow(ethernet):
0033    attrs = {}
0034    attrs[core.DL_DST] = ethernet.dst
0035    attrs[core.DL_SRC] = ethernet.src
0036    attrs[core.DL_TYPE] = ethernet.type
0037    p = ethernet.next
0038
0039     if isinstance(p, vlan):
0040         attrs[core.DL_VLAN] = p.id
0041         attrs[core.DL_VLAN_PCP] = p.pcp
0042         p = p.next
0043     else:
0044         attrs[core.DL_VLAN] = 0xffff
0045         attrs[core.DL_VLAN_PCP] = 0
0046
0047     if isinstance(p, ipv4):
0048         attrs[core.NW_SRC] = p.srcip
0049         attrs[core.NW_DST] = p.dstip
0050         attrs[core.NW_PROTO] = p.protocol
0051         p = p.next
0052
0053         if isinstance(p, udp) or isinstance(p, tcp):
0054             attrs[core.TP_SRC] = p.srcport
0055             attrs[core.TP_DST] = p.dstport
0056         else:
0057             if isinstance(p, icmp):
0058                 attrs[core.TP_SRC] = p.type
0059                 attrs[core.TP_DST] = p.code
0060             else:
0061                 attrs[core.TP_SRC] = 0
0062                 attrs[core.TP_DST] = 0
0063     else:
0064         attrs[core.NW_SRC] = 0
```

```
0065          attrs[core.NW_DST] = 0
0066          attrs[core.NW_PROTO] = 0
0067          attrs[core.TP_SRC] = 0
0068          attrs[core.TP_DST] = 0
0069
0070      return attrs
0071
0072
0073
0074  def do_l2_learning(dpid, inport, packet):
0075     global inst
0076     global macs_str_hp
0077     global macs_str_lp
0078     global macs_str
0079
0080      # learn MAC on incoming port
0081      srcaddr = packet.src.tostring()
0082      print 'level 2 learning'
0083      if ord(srcaddr[0]) & 1:
0084          print 'Source MAC  address is 0'
0085          return
0086      if inst.st[dpid].has_key(srcaddr):
0087          dst = inst.st[dpid][srcaddr]
0088          if dst[0] != inport:
0089              log.msg('MAC has moved from
                       '+str(dst)+'to'+str(inport), system='ctrlsw')
0090          else:
0091              print 'MAC is unchanged:',mac_to_str(packet.src)
0092              return
0093      else:
0094          log.msg('learned MAC '+mac_to_str(packet.src)+' on %d
                         %d'% (dpid,inport), system="ctrlsw")
0095          print 'learned MAC',mac_to_str(packet.src),' on
                         dpid',dpid,'and ingress port ',inport
0096
0097
0098
0099
0100
0101
0102      # learn or update timestamp of entry
0103      inst.st[dpid][srcaddr] = (inport, time(), packet)
0104
0105      # Replace any old entry for (switch,mac).
0106      mac = mac_to_int(packet.src)
0107
0108  # --
0109  # If we've learned the destination MAC set up a flow and
0110  # send accordingly to device priority.  Else, flood.
0111  # --
0112  def forward_l2_packet(dpid, inport, packet, buf, bufid):
0113      dstaddr = packet.dst.tostring()
0114      if not ord(dstaddr[0]) & 1 and
                              inst.st[dpid].has_key(dstaddr):
0115          prt = inst.st[dpid][dstaddr]
0116          sendport=3
0117          if  prt[0] == inport:
0118              log.err('**warning** learned port = inport',
                              system="ctrlsw")
0119              inst.send_openflow(dpid, bufid, buf,
                         openflow.OFPP_FLOOD, inport)
```

```
0120            print 'Learned port is inport',inport
0121        else:
0122            # We know the outport, set up a flow
0123            flow = create_l3_out_flow(packet)
0124            flow[core.IN_PORT] = inport
0125            actions = [[openflow.OFPAT_OUTPUT, [0,port[0]]]]
0126            print 'NOTICE - Installing flow for:',str(packet)
0127            if mac_to_str(packet.src) in macs_str_hp:
0128               print 'NOTICE - Installing HIGH priority flow for
                          dpid', dpid
0129            inst.install_datapath_flow(dpid,flow,CACHE_PERMFLOW,
0130                           openflow.OFP_FLOW_PERMANENT,actions,
0131                    bufid, openflow.OFP_DEFAULT_PRIORITY+32767,
0132                           inport, buf)
0133            elif mac_to_str(packet.src) in macs_str_lp:
0134               print 'NOTICE - Installing LOW priority flow for
                          dpid', dpid
0135             inst.install_datapath_flow(dpid,flow,CACHE_TIMEOUT,
0136                           openflow.OFP_FLOW_PERMANENT,actions,
0137                    bufid, openflow.OFP_DEFAULT_PRIORITY-32767,
0138                           inport, buf)
0139
0140            else:
0141               print 'NOTICE - Installing DEFAULT priority flow
                          for dpid', dpid
0142             inst.install_datapath_flow(dpid,flow,CACHE_TIMEOUT,
0143                           openflow.OFP_FLOW_PERMANENT,actions,
0144                          bufid, openflow.OFP_DEFAULT_PRIORITY,
0145                           inport, buf)
0146           #else: print  'NO info available
0147    else:
0148        # haven't learned destination MAC. Flood
0149        print 'Flood Action. Non Unicast - Non Stored
     Destination MAC.',      mac_to_str(packet.dst) ,'Source
     MAC:',mac_to_str(packet.src)
0150        inst.send_openflow(dpid, bufid, buf,
                           openflow.OFPP_FLOOD, inport)
0151
0152 # --
0153 # Responsible for timing out cache entries.
0154 # Is called every 1 second.
0155 # --
0156 def timer_callback():
0157     global inst
0158
0159     curtime  = time()
0160     for dpid in inst.st.keys():
0161         for entry in inst.st[dpid].keys():
0162             if (curtime - inst.st[dpid][entry][1]) >
                                          CACHE_TIMEOUT:
0163                 log.msg('timing out
entry'+mac_to_str(entry)+str(inst.st[dpid][entry])+' on switch %x' %
dpid, system='ctrlsw')
0164                 inst.st[dpid].pop(entry)
0165
0166     inst.post_callback(1, timer_callback)
0167     return True
0168
0169 def datapath_leave_callback(dpid):
0170     logger.info('Switch %x has left the network' % dpid)
0171     if inst.st.has_key(dpid):
```

```
0172         del inst.st[dpid]
0173
0174 def datapath_join_callback(dpid, stats):
0175     logger.info('Switch %x has joined the network' % dpid)
0176
0177 # --
0178 # Packet entry method.
0179 # Drop LLDP packets (or we get confused) and attempt learning
0180 # and forwarding
0181 # --
0182 def packet_in_callback(dpid, inport, reason, len, bufid,
                                                          packet):
0183     global macs_str_hp
0184     global macs_str_lp
0185
0186     if not packet.parsed:
0187         log.msg('Ignoring incomplete packet',system='ctrlsw')
0188
0189     if not inst.st.has_key(dpid):
0190         log.msg('registering new switch %x' %
                        dpid,system='ctrlsw')
0191         inst.st[dpid] = {}
0192
0193     # don't forward lldp packets
0194     if packet.type == ethernet.LLDP_TYPE:
0195         print 'lldp type packet'
0196         return CONTINUE
0197
0198     # learn MAC on incoming port
199     do_l2_learning(dpid, inport, packet)
0200
0201     forward_l2_packet(dpid, inport, packet, packet.arr, bufid)
0202
0203     return CONTINUE
0204
0205 class ctrlsw(Component):
0206
0207     def __init__(self, ctxt):
0208         global inst
0209         Component.__init__(self, ctxt)
0210         self.st = {}
0211
0212         inst = self
0213
0214     def install(self):
0215         inst.register_for_packet_in(packet_in_callback)
0216         inst.register_for_datapath_leave(datapath_leave_callback)
0217         inst.register_for_datapath_join(datapath_join_callback)
0218         inst.post_callback(1, timer_callback)
0219
0220     def getInterface(self):
0221         return str(ctrlsw)
0222
0223 def getFactory():
0224     class Factory:
0225         def instance(self, ctxt):
0226             return ctrlsw(ctxt)
0227
0228     return Factory()
```

ΠΑΡΑΡΤΗΜΑ Γ

ΕΤΟΙΜΑ ΥΠΟΠΡΟΓΡΑΜΜΑΤΑ (SCRIPTS) ΤΩΝ NETFPGA'S ΤΟΥ ΕΡΓΑΣΤΗΡΙΟΥ

- Παρακάτω επισυνάπτονται κάποιες εντολές και το βοηθητικό υποπρόγραμμα dpctl που χρησιμοποιηθήκαν στα πλαίσια της διπλωματικής εργασίας.

Εκκίνηση και τερματισμός του NetFPGA:

./of_start.sh 147.102.7.11:6633

./of_stop.sh 147.102.7.11

Η dpctl εφαρμογή:

Εμφάνιση Κατάστασης	dpctl show <socket>
Εμφάνιση Δεδομένων	dpctl status <socket>
Προσθήκη Ροής	dpctl add-flow <socket> <arguments>
Εμφάνιση Ροών	dpctl dump-flows <socket>
Εμφάνιση Θυρών	dpctl dump-ports <socket>
Εμφάνιση Πινάκων Ροών	dpctl dump-tables <socket>
Εμφάνιση Στατιστικών Πρωτοκόλλου	dpctl show-protostat <socket>
Εμφάνιση Ληφθέντων πακέτων από μεταγωγέα	dpctl monitor <socket>

<socket> : Η θέση εγκατάστασης του NetFPGA στο μηχάνημα.
Πχ: unix:/root/test unix:/var/run/test

<arguments> : Οι παράμετροι της ροής για επεξεργασία.
Πχ:
dpctl add-flow unix:/root/test idle_timeout=270, hard_timeout=0, ip, nw_dst=147.102.40.231, in_port=4,actions=output:3