ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Ανάπτυξη Περιβάλλοντος Ευφυών Προγραμματιζόμενων Δικτύων για Οπτικά Δίκτυα Μεταφοράς Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Δημητράκη Βασίλειου

Επιβλέπων : Ηρακλής Αβραμόπουλος

Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Ανάπτυξη Περιβάλλοντος Ευφυών Προγραμματιζόμενων Δικτύων για Οπτικά Δίκτυα Μεταφοράς Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Δημητράκη Βασίλειου

Επιβλέπων : Ηρακλής Αβραμόπουλος
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Δεκεμβρίου 2014.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Ηρακλής Αβραμόπουλος	Νικόλαος Ουζούνογλου	Χρήστος Καψάλης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2014

(Υπογραφή)

.....

Δημητράκης Βασίλειος

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

Περίληψη

Τα Ευφυή Προγραμματιζόμενα Δίκτυα τείνουν να αποτελέσουν το κύριο μέσο για την εξυπηρέτηση της τηλεπικοινωνιακής κίνησης που δημιουργείται από τα datacenters αλλά και από τις σύγχρονες εφαρμογές cloud. Η νέα αυτή αρχιτεκτονική δικτύων δίνει τη δυνατότητα για αποδοτική διαχείριση των διαθέσιμων πόρων του δικτύου και ελάχιστη ανθρώπινη παρέμβαση. Το OpenFlow αποτελεί την πρώτη προτυποποιημένη διεπαφή για SDN δίκτυα και μάλιστα ο κώδικάς του είναι open-source, γεγονός που επιτρέπει την υλοποίηση πληθώρας δικτυακών εφαρμογών, οι οποίες διαχωρίζονται από το υφιστάμενο hardware και δε σχετίζονται με τον κατασκευαστή αυτού (vendor agnostic).

Μία νέα τάση είναι η επέκταση του πρωτοκόλλου OpenFlow στο οπτικό επίπεδο μεταφοράς δεδομένων. Τα πλεονεκτήματα αυτού του εγχειρήματος είναι πολλά και μάλιστα αρκετά σημαντικά. Αρχικά μειώνεται σημαντικά η IP κίνηση της πληροφορίας, γίνεται εκμετάλλευση του πολύ μεγάλου εύρους ζώνης των οπτικών δικτύων και μέσω της επίπεδης αρχιτεκτονικής που επιτυγχάνεται, γίνεται πολύ καλύτερη αξιοποίηση των δικτυακών πόρων.

Σκοπός της παρούσας διπλωματικής εργασίας είναι η εξοικείωση με τα διαθέσιμα εργαλεία και η υλοποίηση μίας ενιαίας πλατφόρμας προσομοίωσης δικτύου, η οποία θα υποστηρίζει μια επέκταση του πρωτοκόλλου OpenFlow για την ενσωμάτωση οπτικών παραμέτρων μετάδοσης, όπως είναι το μήκος κύματος, ο τύπος διαμόρφωσης των δεδομένων κ.ά. Οι ιδιότητες αυτές θα δώσουν τη δυνατότητα για τη δημιουργία SDN δικτύων για την εξυπηρέτηση τηλεπικοινωνιακής κίνησης σε οπτικό δίκτυο.

Λέξεις – κλειδιά

Ευφυή Προγραμματιζόμενα Δίκτυα, OpenFlow πρωτόκολλο, μεταγωγέας, ελεγκτής, flow εγγραφή, flow πίνακας δρομολόγησης, Mininet, Wireshark, Oftest, Οπτικά δίκτυα μεταφοράς δεδομένων

Abstract

Software Defined Networks (SDN) tend to be the basic means of serving the telecommunication traffic, which is produced mainly by datacenters and the modern cloud applications. This new architecture of networks gives the possibility for efficient management of network resources and offers minimum human intervention. OpenFlow is the first standardized interface for SDN Networks. Its code is open-sourced and this innovation allows the creation of many network applications and the configuration of hardware equipment that is vendor agnostic.

A new trend is emerging, which is the extension of the OpenFlow protocol in the optical transport networks. The advantages of this effort are numerous and very important. Thanks to this advancement, we can achieve IP traffic offloading, utilization of the huge bandwidth of the optical networks and, through the flat architecture that this model offers, better utilization of network resources.

The main goal of this diploma thesis is the familiarization with the existing software utilities for SDN and the creation of a simulation platform, in which optical parameters will be introduced, such as wavelength. These properties will give the possibility for the extension of the SDN Networks in the optical transport networks.

Key – words

Software Defined Networks, OpenFlow protocol, switch, controller, flow entry, flow table, Mininet, Wireshark, OFTest, Optical Transport Networks

Ευχαριστίες

Καταρχάς, θα ήθελα να ευχαριστήσω θερμά τον Καθ. Ηρακλή Αβραμόπουλο που μου έδωσε τη δυνατότητα να πραγματοποιήσω την παρούσα διπλωματική εργασία υπό την επίβλεψη και καθοδήγησή του.

Στη συνέχεια, θα ήθελα να ευχαριστήσω τη Διδάκτορα Μαρία Σπυροπούλου, τον Υποψήφιο Διδάκτορα Βασίλειο Κατωπόδη και το Διδάκτορα Χρήστο Κουλουμέντα για την αμέριστη βοήθειά τους και τη συνεχή καθοδήγησή τους καθόλη τη διάρκεια της εκπόνησης της παρούσας διπλωματικής εργασίας.

Θα ήθελα ακόμα να ευχαριστήσω την οικόγενειά μου και τους φίλους μου για την αγάπη και τη συμπαράστασή τους αυτά τα πέντε χρόνια φοίτησής μου στο ΕΜΠ. Τέλος θα ήθελα να ευχαριστήσω τη μητέρα μου, Σοφία, για τη φιλολογική επιμέλεια του κειμένου.

Πίνακας Περιεχομένων

Πίνακας σχημάτων

Λίστα πινάκων

1. ΕΥΦΥΗ ΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΑ ΔΙΚΤΥΑ

1.1. Εισαγωγή

Με την εμφάνιση του cloud computing και των υπηρεσιών video υψηλών ρυθμών μετάδοσης, οι εφαρμογές για τα κέντρα δεδομένων (data centres) παρουσιάζουν χαρακτηριστικά υψηλής εκρηκτικότητας και πολύ μεγάλου εύρους ζώνης. Αυτά αφορούν κυρίως εφαρμογές υπερ-υψηλών μηκών κύματος με ταχύτητες που ξεπερνούν τα 100 Gbps. Τα Ευφύη Προγραμματιζόμενα Δίκτυα με τη βοήθεια του πρωτοκόλλου OpenFlow, που λειτουργεί ως μία κεντροποιημένη αρχιτεκτονική ελέγχου, έχουν αποκτήσει ιδιαίτερη δημοτικότητα, δίοτι υποστηρίζουν προγραμματιζόμενες λειτουργίες δικτύου και πρωτοκόλλων. Οι τελευταίες δίνουν τη δυνατότητα για μεγαλύτερη ευελιξία στους χρήστες και δημιουργούν έναν ενοποιημένο έλεγχο πάνω σε διάφορους δικτυακούς πόρους για τη βελτιστοποίηση λειτουργιών και υπηρεσιών [1].

Τα Ευφυή Προγραμματιζόμενα Δίκτυα (Software Defined Networks – SDN) [2] αποτελούν μία νέα προσέγγιση στον τρόπο δικτύωσης, στην οποία ο έλεγχος του δικτύου διαχωρίζεται από τη διαδικασία της προώθησης δεδομένων και γίνεται απευθείας προγραμματιζόμενος. Το γεγονός αυτό έχει ως αποτέλεσμα τη δημιουργία μίας αρχιτεκτονικής εξαιρετικά δυναμικής, διαχειρίσιμης, αποτελεσματικής από άποψη κόστους και εφαρμόσιμης, που δίνει στους διαχειριστές τη δυνατότητα του προγραμματισμού, του αυτοματισμού και του ελέγχου.

Η ιδέα των προγραμματιζόμενων δικτύων έχει προταθεί μάλιστα και ως ένας τρόπος για τη διευκόλυνση της εξέλιξης αυτών. Όπως αναφέρθηκε και προηγουμένως, ο έλεγχος έχει αποσπαστεί πλέον από τη λειτουργία της προώθησης των δεδομένων και έχει μεταφερθεί σε κεντρικούς ελεγκτές (controllers), οι οποίοι δίνουν τη δυνατότητα στην υποδομή διαδικτύωσης να είναι διαφοροποιημένη σε σχέση με την εκάστοτε εφαρμογή. Επιπλέον η αρχιτεκτονική των Ευφυών Προγραμματιζόμενων Δικτύων παρέχει ένα σύνολο από Διεπαφές Προγραμματιζόμενων Εφαρμογών (Application Programming Interfaces – APIs), γεγονός που απλοποιεί σημαντικά την ενσωμάτωση των κοινών διαδικτυακών εφαρμογών (για παράδειγμα, δρομολόγηση, πολυεκπομπή, ασφάλεια, έλεγχος πρόσβασης, διαχείριση φάσματος, traffic engineering, QoS – Quality of Service, απόδοση ενέργειας και διάφοροι τύποι διαχείρισης πολιτικής).

Σε αυτού του είδους τα δίκτυα η ευφυία τους είναι λογικά κεντροποιημένη σε ελεγκτές βασισμένους σε λογισμικό (controllers) (στο επίπεδο ελέγχου) και επιπλέον οι συσκευές διαδικτύωσης μετατρέπονται σε απλές συσκευές προώθησης πακέτων (το επίπεδο δεδομένων), οι οποίες μπορούν να προγραμματιστούν μέσα από μία διεπαφή (open interface).

1.1.1. Cloud computing

Το cloud computing [3] είναι ένας γενικός όρος για οτιδήποτε περιλαμβάνει τη διανομή υπηρεσιών μέσω του διαδικτύου. Αυτές οι υπηρεσίες είναι ευρέως διαμοιρασμένες σε τρεις κατηγορίες. Το όνομα του cloud computing εμπνεύστηκε από το σύμβολο του cloud, το οποίο συχνά χρησιμοποιείται για την αναπαράσταση του Internet στα διαγράμματα ροής.

Μία υπηρεσία cloud έχει τρία ξεχωριστά χαρακτηριστικά, τα οποία το διαφοροποιούν από το παραδοσιακό hosting. Η διανομή του γίνεται με πώληση on-demand, και πιο συχνά με τα λεπτά ή την ώρα. Είναι ελαστική, πράγμα που σημαίνει ότι ο χρήστης μπορεί να λάβει ό,τι χρειάζεται από την υπηρεσία σε οποιαδήποτε στιγμή. Η υπηρεσία μάλιστα διευθύνεται πλήρως από τον πάροχο. Σημαντικές καινοτομίες που έχουν λάβει χώρα, όπως το virtualization και το distributed computing, καθώς επίσης η βελτιωμένη πρόσβαση σε υψηλής ταχύτητας Internet έχουν επιταχύνει το ενδιαφέρον για το cloud computing.

Ένα cloud μπορεί να είναι ιδιωτικό ή δημόσιο. Ένα δημόσιο cloud πουλάει υπηρεσίες σε οποιονδήποτε στο Internet (Την τρέχουσα περίοδο, οι διαδικτυακές υπηρεσίες του Amazon αποτελούν το μεγαλύτερο πάροχο δημοσίου cloud) [3]. Ένα ιδιωτικό cloud είναι ένα κατάλληλο δίκτυο ή κέντρο δεδομένων (data center), το οποίο παρέχει ενσωματωμένες υπηρεσίες σε έναν περιορισμένο αριθμό ανθρώπων. Όταν ένας πάροχος υπηρεσιών χρησιμοποιεί πόρους δημοσίου cloud για τη δημιουργία του δικού του ιδιωτικού, αυτό τότε ονομάζεται εικονικό ιδιωτικό cloud. Ανεξάρτητα αν είναι ιδιωτικό ή δημόσιο, ο στόχος του cloud computing είναι να παρέχει εύκολη και επεκτάσιμη πρόσβαση σε υπολογιστικούς πόρους και IT υπηρεσίες.

Infrastracture-as-a-Service (IaaS): Το IaaS [3] (Διαδικτυακές υπηρεσίες του Amazon) παρέχει στον πελάτη δυνατότητα αποθήκευσης εικονικού διακομιστή, καθώς επίσης και Application Program Interfaces (APIs), οι οποίες δίνουν τη δυνατότητα στον πελάτη να ξεκινήσει, να σταματήσει, να έχει πρόσβαση και να διαμορφώσει τους εικονικούς του διακομιστές και το χώρο αποθήκευσης. Αυτό το μοντέλο επιτρέπει σε μία εταιρεία να πληρώνει μόνο για τη χωρητικότητα, την οποία χρειάζεται, και για την απόκτηση

περισσότερης online, σε περίπτωση που τη χρειαστεί. Επειδή αυτού του είδους το μοντέλο μοιάζει με τον τρόπο με τον οποίο ο ηλεκτρισμός, το νερό και τα καύσιμα καταναλώνονται, ονομάζεται επίσης utility computing.

Platform-as-a-Service (PaaS): Ορίζεται [3] ως ένα σύνολο από εργαλεία ανάπτυξης λογισμικού, τα οποία φιλοξενούνται στην υποδομή του διακομιστή. Οι προγραμματιστές δημιουργούν εφαρμογές στην πλατφόρμα του παρόχου πάνω από το Internet. Οι PaaS πάροχοι μπορεί να χρησιμοποιούν APIs, website portals ή gateway software, που είναι εγκατεστημένα στον υπολογιστή του πελάτη. Η Force.com και οι GoogleApps είναι παραδείγματα υπηρεσιών PaaS. Οι προγραμματιστές χρειάζεται να γνωρίζουν ότι δεν υπάρχουν πρότυπα για διαλειτουργικότητα ή μεταφερσιμότητα δεδομένων στο cloud. Κάποιοι πάροχοι μπορεί να μην επιτρέπουν τη μεταφορά λογισμικού, που δημιουργήθηκε από τους πελάτες, εκτός της πλατφόρμας του παρόχου.

Software-as-a-Service (SaaS): Σε ένα SaaS μοντέλο [3], ο κατασκευαστής παρέχει την υποδομή, το λογισμικό και αλληλεπιδρά με τους χρήστες μέσω ενός front-end portal. Η υπηρεσία SaaS είναι μία πολύ ευρεία αγορά. Οι υπηρεσίες μπορεί να είναι οποιεσδήποτε από Web-based email, inventory control και database processing. Επειδή ο πάροχος υπηρεσιών φιλοξενεί και την εφαρμογή αλλά και τα δεδομένα, ο τελικός χρήστης είναι ελεύθερος να χρησιμοποιήσει την υπηρεσία από οπουδήποτε.

1.1.2. Big Data Centers

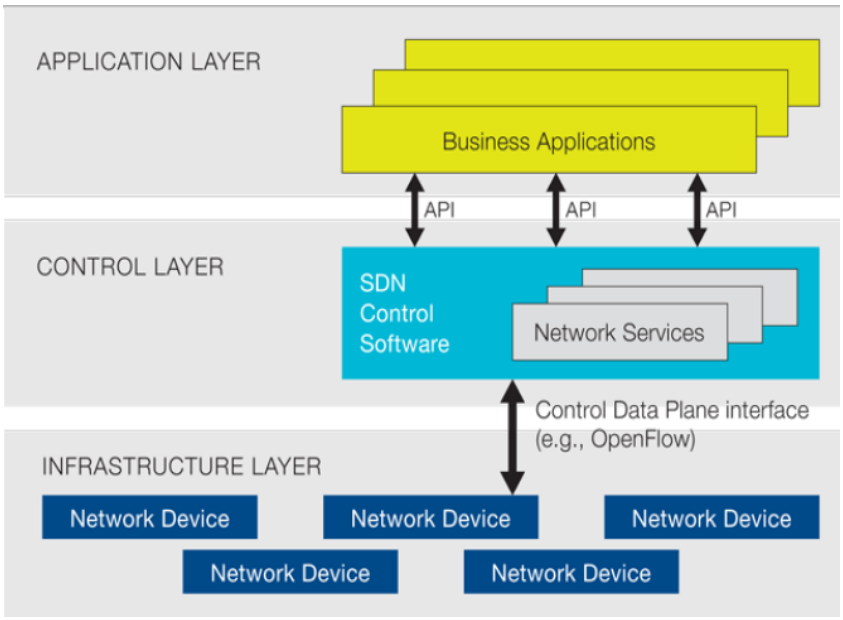
Η εξέλιξη στον τομέα των ψηφιακών αισθητήρων, των τηλεπικοινωνιών, της υπολογισιμότητας και της αποθήκευσης έχει δημιουργήσει τεράστιες συλλογές δεδομένων, προερχόμενες από διάφορους τομείς της καθημερινότητας, όπως είναι οι επιχειρήσεις, η επιστήμη, οι κυβερνήσεις και γενικότερα η κοινωνία. Για παράδειγμα, οι εταιρείες μηχανών αναζήτησης, όπως είναι η Google, η Yahoo! και η Microsoft, έχουν δημιουργήσει μία εξ ολοκλήρου νέα επιχείρηση με το να συλλέγουν ελεύθερα πληροφορία που είναι διαθέσιμη στον Παγκόσμιο Ιστό (WorldWide Web – www) και να την παρέχουν στους χρήστες με διάφορους τρόπους. Αυτές οι εταιρείες συλλέγουν τρισεκατομμύρια από bytes καθημερινά και συνεχώς προσθέτουν νέες υπηρεσίες, όπως είναι οι εικόνες από δορυφόρο και οι οδηγίες πλοήγησης. Τα οφέλη από αυτές τις υπηρεσίες είναι αμέτρητα, έχοντας αλλάξει τον τρόπο με τον οποίο οι άνθρωποι βρίσκουν και χρησιμοποιούν την πληροφορία σε καθημερινή βάση.

Όπως οι μηχανές αναζήτησης έχουν αλλάξει τον τρόπο πρόσβασης στην πληροφορία, άλλες μορφές big data computing μπορούν και θα έχουν στο μέλλον τη δυνατότητα να

διαφοροποιήσουν τις διαστηριότες των εταιρειών, των επιστημόνων, την εθνική άμυνα και τις ευφυείς λειτουργίες [4].

1.2. Αρχιτεκτονική SDN δικτύων

Η αρχιτεκτονική των Ευφυών Προγραμματιζόμενων Δικτύων διαφέρει σε μεγάλο βαθμό σε σχέση με αυτή των συμβατικών. Αποτελείται πλέον από τρία επίπεδα: το επίπεδο υποδομής (Infrastracture layer), το επίπεδο ελέγχου (Control layer) και το επίπεδο εφαρμογής (Application layer). Το πρώτο αποτελεί το επίπεδο στο οποίο επικοινωνούν μεταξύ τους οι εκάστοτε συσκευές δικτύου, όπως είναι οι μεταγωγείς. Την εποπτεία αυτής της επικοινωνίας αναλαμβάνει το αμέσως ανώτερο επίπεδο, αυτό του ελέγχου, το οποίο επικοινωνεί με το επίπεδο εφαρμογής μέσω της διεπαφής ελέγχου-δεδομένων, όπως είναι το OpenFlow. Τέλος, το ανώτερο επίπεδο από τα τρία είναι αυτό της εφαρμογής που αφορά όλες τις χρησιμοποιούμενες εφαρμογές και επικοινωνεί με το επίπεδο ελέγχου μέσω μιας API διεπαφής. Η αρχιτεκτονική αυτή, όπως περιγράφηκε προηγουμένως, παρουσιάζεται στο σχήμα 1.



Σχήμα 1: Αρχιτεκτονική SDN δικτύων

1.2.1. Δομικά στοιχεία των SDN δικτύων

Ο SDN μεταγωγέας [2] [5] (π.χ.OpenFlow μεταγωγέας), ο SDN ελεγκτής (controller), οι διεπαφές που είναι προσαρτημένες στον ελεγκτή για την επικοινωνία με τις συσκευές προώθησης δεδομένων (πιο γενικά η λεγόμενη Southbound Interface (OpenFlow)) και οι διεπαφές εφαρμογών δικτύου (northbound interfaces) αποτελούν τα θεμελιώδη στοιχεία μίας SDN εφαρμογής.

Οι μεταγωγείς στα SDN δίκτυα συχνά αναπαριστώνται σαν η βασική συσκευή προώθησης που είναι προσβάσιμη μέσω της ανοιχτής εφαρμογής (open interface), καθώς οι αλγόριθμοι αλλά και η λογική ελέγχου είναι μετατοπισμένες στον ελεγκτή. Οι OpenFlow μεταγωγείς μπορούν να χωριστούν σε δύο κατηγορίες: τους αμιγώς OpenFlow (pure – OpenFlow only) μεταγωγείς και τους υβριδικούς μεταγωγείς (OpenFlow-enabled). Οι αμιγώς OpenFlow μεταγωγείς βασίζονται πλήρως στον ελεγκτή για τις αποφάσεις προώθησης. Από την άλλη πλευρά, οι υβριδικοί μεταγωγείς υποστηρίζουν το OpenFlow αλλά και την παραδοσιακή λειτουργία προώθησης.

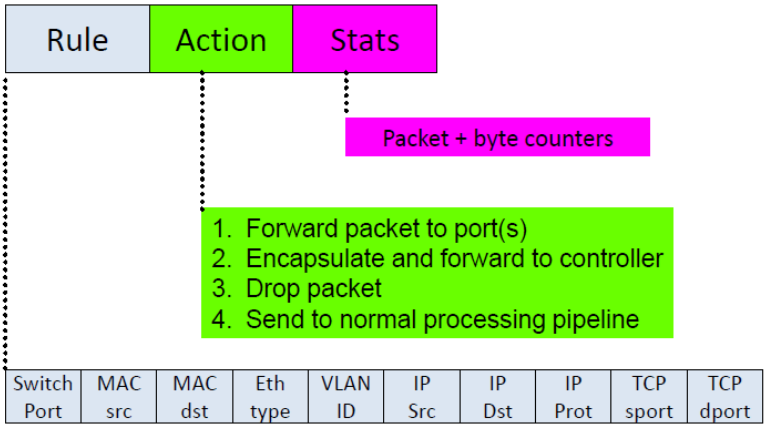
Ένας OpenFlow μεταγωγέας αποτελείται από ένα flow table, ο οποίος εκτελεί την αναζήτηση των πακέτων και την αντίστοιχη προώθηση. Επιπλέον να επισημανθεί ότι ένας flow table σε ένα μεταγωγέα αποτελείται από ένα σύνολο από flow entries.

Στο σημείο αυτό θα γίνει μια αναφορά στο τι εννοείται με τον όρο flow. Ένα flow αποτελεί μία σειρά από πακέτα, η οποία έχει κοινά χαρακτηριστικά (π.χ. IP destination address, Ethernet address, priority κ.ά) και ταιριάζει με μία συγκεκριμένη εγγραφή (flow entry) στον πίνακα δρομολόγησης (flow table) του μεταγωγέα. Έτσι κάθε πακέτο που ανήκει στο flow φτάνει στην είσοδο του μεταγωγέα, ακολουθεί την ίδια διαδικασία για την έξοδό του από αυτόν. Η διαδικασία αυτή περιγράφεται με ένα σετ από κανόνες (rules) και ενέργειες (actions). Τέλος, ο συνδυασμός των flow entries σε πολλαπλούς μεταγωγείς ορίζει ένα μονοπάτι που ακολουθεί το συγκεκριμένο flow.

Τα flow entries μέσα σε ένα flow table αποτελούνται από :

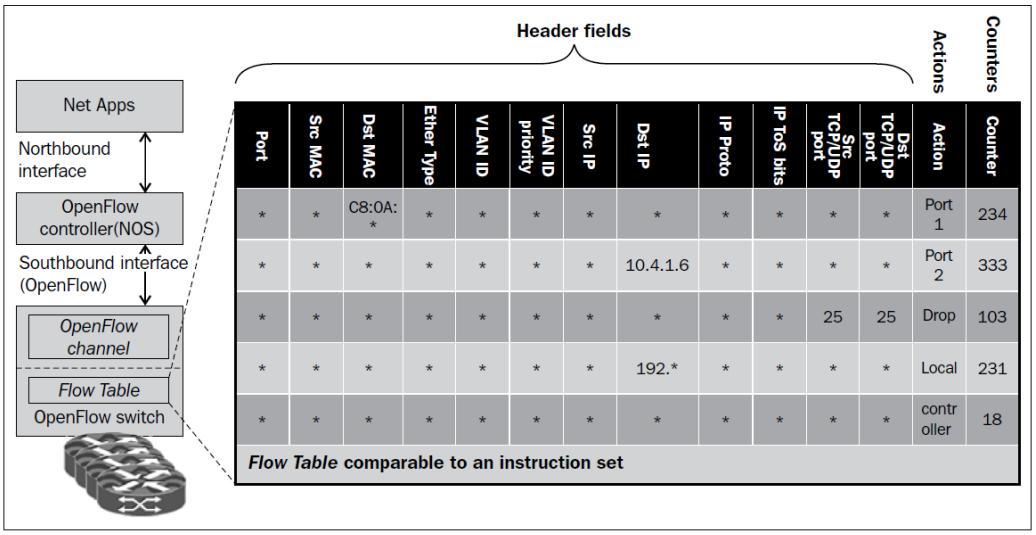
- Τα πεδία επικεφαλίδας ή ταιριάσματος (Header or Match fields) : Με πληροφορίες που βρίσκονται στην επικεφαλίδα των πακέτων, στη θύρα εισόδου του μεταγωγέα, καθώς και στα metadata και συμβάλλουν στο να επιτευχθεί το match με τα εισερχόμενα πακέτα.
- Μετρητές (Counters) : Χρησιμοποιούνται για τη συλλογή στατιστικών για ένα συγκεκριμένο flow, π.χ. τον αριθμό των εισερχομένων πακέτων, τον αριθμό των bytes, τη χρονική διάρκεια του flow.
- Ένα σύνολο εντολών ή ενεργειών (Action) : Εφαρμόζονται μετά από κάθε ταίριασμα και ορίζουν πώς θα γίνει ο χειρισμός των ταιριασμένων πακέτων. Για παράδειγμα, μία ενέργεια μπορεί να είναι η προώθηση ενός πακέτου προς μία συγκεκριμένη θύρα του μεταγωγέα.

Η εικόνα ενός flow entry με τα αντίστοιχα πεδία του παρουσιάζεται στο σχήμα 2:



Σχήμα 2: Παρουσίαση ενός flow entry με τα αντίστοιχα πεδία του

Στη συνέχεια παρουσιάζουμε και την εικόνα της αρχιτεκτονικής των SDN δικτύων και σε μεγέθυνση την εικόνα ενός flow table (σχήμα 3). Παρατηρώντας τον πίνακα δρομολόγησης διαπιστώνεται ότι αρκετά πεδία του έχουν οριστεί με το σύμβολο "*". Το γεγονός αυτό σημαίνει πως τα εν λόγω πεδία έχουν οριστεί ως wildcarded. Ο όρος αυτός δηλώνει πως δεν ενδιαφέρει η τιμή του συγκεκριμένου πεδίου στο flow entry και ένα εισερχόμενο πακέτο θα κάνει match με αυτό το πεδίο οπιαδήποτε τιμή και αν φέρει σε αυτό. Στη συνέχεια εξηγούμε συνοπτικά τη λειτουργία του ακόλουθου flow table. Αν για παράδειγμα καταφθάσει ένα πακέτο με ορισμένη τη διεύθυνση IP προορισμού στην τιμή 10.4.1.6, τότε αυτό θα οδηγηθεί στη θύρα 2. Σε περίπτωση όμως που έρθει πακέτο με όλα τα πεδία του να είναι wildcards ή καμία τιμή από τα πεδία του πακέτου να μην ταιριάζει με κάποια από τις υπάρχουσες εγγραφές, τότε ο μεταγωγέας στέλνει μήνυμα στον ελεγκτή να αποφασίσει αυτός για τη μεταγωγή του πακέτου.



Σχήμα 3: Αρχιτεκτονική SDN δικτύων και παρουσίαση ενός flow table

1.2.2. Μηνύματα στο πρωτόκολλο OpenFlow

Το πρωτόκολλο OpenFlow [2] περιγράφει τις ανταλλαγές μηνυμάτων, οι οποίες λαμβάνουν χώρα ανάμεσα στον OpenFlow ελεγκτή και στον OpenFlow μεταγωγέα. Τυπικά, το πρωτόκολλο είναι ενσωματωμένο στην κορυφή του Επιπέδου Ασφάλειας Μεταφοράς (Transport Layer Security – TLS) ή SSL, παρέχοντας με αυτόν τον τρόπο ένα ασφαλές OpenFlow κανάλι.

Το OpenFlow πρωτόκολλο δίνει τη δυνατότητα στον ελεγκτή να εκτελέσει ενέργειες, όπως προσθήκη, ανανέωση και διαγραφή των flow entries στους flow tables. Υποστηρίζει μάλιστα μηνύματα τριών ειδών:

1. **Controller – to – Switch μηνύματα**
2. **Ασύγχρονα μηνύματα (Asychronous messages)**
3. **Συμμετρικά μηνύματα (Symmetric messages)**

1.2.2.1. Controller – to – Switch μηνύματα

Τα εν λόγω μηνύματα αρχικοποιούνται από τον ελεγκτή και χρησιμοποιούνται με σκοπό να διευθύνουν ή να διερευνήσουν απευθείας την κατάσταση του μεταγωγέα. Αυτός ο τύπος μηνυμάτων μπορεί να απαιτεί, αλλά ίσως και όχι, μία απάντηση από το μεταγωγέα. Αυτού του είδους τα μηνύματα κατηγοριοποιούνται στις ακόλουθες υποκατηγορίες :

- **Features:** Μετά από την εγκατάσταση της TLS συνεδρίας, ο controller στέλνει ένα μήνυμα αίτησης (request) στο switch. Ο switch πρέπει να απαντήσει με ένα μήνυμα features reply, το οποίο προσδιορίζει τα χαρακτηριστικά και τις ικανότητες που υποστηρίζονται από το switch.
- **Configuration:** Ο controller είναι ικανός να ορίσει και να ζητήσει παραμέτρους διαμόρφωσης στο switch. Ο τελευταίος μόνο απαντά στην ερώτηση από τον controller.
- **Modify – state: (add, modify, delete, modify_strict, delete_strict):** Αυτά τα μηνύματα στέλνονται από τον controller, για να διαχειριστεί την κατάσταση των switches. Χρησιμοποιούνται για την προσθήκη/διαγραφή ή τροποποίηση των εισόδων στους flow tables ή για τον ορισμό των ιδιοτήτων των θυρών του switch.
- **Read – state:** Αυτά τα μηνύματα συλλέγουν στατιστικά από τους flow tables του switch, τις θύρες και τις ξεχωριστές flow entries.
- **Send – packet:** Χρησιμοποιούνται από τον controller για την αποστολή πακέτων προς μία θύρα του switch.

- **Barrier:** Τα barrier request/reply μηνύματα χρησιμοποιούνται από τον controller, για να επιβεβαιώσουν εξαρτήσεις μηνυμάτων ή για να λάβουν ειδοποιήσεις για ολοκληρωμένες λειτουργίες.

1.2.2.2. Ασύγχρονα μηνύματα (Asynchronous messages)

Τα ασύγχρονα μηνύματα αρχικοποιούνται από τον μεταγωγέα και χρησιμοποιούνται, για να ενημερώσουν τον ελεγκτή του δικτύου σχετικά με γεγονότα και αλλαγές στην κατάσταση του μεταγωγέα. Οι μεταγωγείς στέλνουν ασύγχρονα μηνύματα στον ελεγκτή με σκοπό να ενημερώσουν για μία άφιξη πακέτου, για αλλαγή της κατάστασης του μεταγωγέα ή για κάποιο ενδεχόμενο λάθος. Υπάρχουν τέσσερα κύρια ασύγχρονα μηνύματα, τα οποία περιγράφονται κάτοθι :

- **Packet – in:** Για όλα τα πακέτα, τα οποία δεν έχουν μία flow entry μέσα στο flow table που να μπορούν να κάνουν match ή για ένα πακέτο που ταιριάζει με ένα flow entry, το οποίο στέλνει το πακέτο στον controller, ένα packet – in μήνυμα στέλνεται στον ελεγκτή. Αν ο μεταγωγέας έχει επαρκή μνήμη για την ενταμίευση των πακέτων που στέλνονται στον ελεγκτή, το packet-in μήνυμα περιέχει κομμάτια της επικεφαλίδας του πακέτου (συνήθως, 128 bytes) και στη συνέχεια ένα αναγνωριστικό χρησιμοποιείται από τον ελεγκτή για τη στιγμή εκείνη που ο μεταγωγέας θα είναι έτοιμος να στείλει τα πακέτα. Τέλος οι μεταγωγείς, που δεν υποστηρίζουν εσωτερική ενταμίευση, θα πρέπει να στείλουν ολόκληρο το πακέτο ως κομμάτι του μηνύματος.
- **Flow – removal:** Όταν μία flow entry προστίθεται στο μεταγωγέα από ένα μήνυμα αλλαγής του flow, η τιμή του idle timeout δείχνει πότε η flow entry θα πρέπει να αφαιρεθεί εξαιτίας της έλλειψης δραστηριότητας. Από την άλλη μεριά η τιμή του hard timeout δείχνει το πότε μία flow entry θα πρέπει να αφαιρεθεί ανεξάρτητα από τη δραστηριότητα. Τέλος το flow modify μήνυμα καθορίζει επίσης το κατά πόσο ο μεταγωγέας θα έπρεπε να στείλει ένα μήνυμα απομάκρυνσης ενός flow στον ελεγκτή, όταν το flow entry λήξει.
- **Port – status:** Ο μεταγωγέας είναι αναμενόμενο να στείλει μηνύματα σχετικά με την κατάσταση των θυρών στον ελεγκτή, καθώς η κατάσταση ρυθμίσεων των θυρών αλλάζει.
- **Error:** Ο μεταγωγέας είναι ικανός να ειδοποιήσει τον ελεγκτή για προβλήματα χρησιμοποιώντας κατάλληλα μηνύματα λάθους. Αυτά τα γεγονότα περιλαμβάνουν

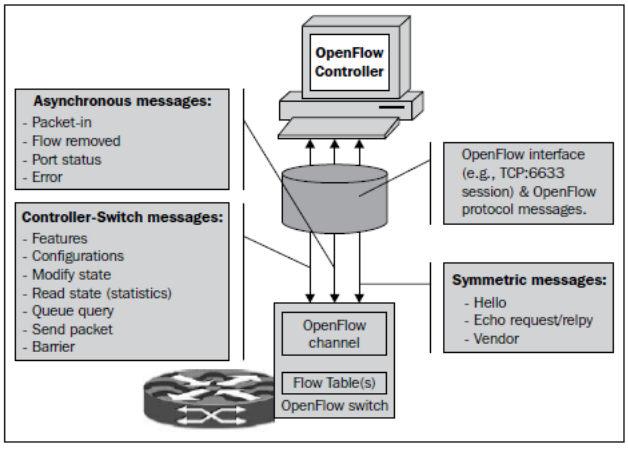
αλλαγές στην κατάσταση των θυρών ή αλλαγές στην κατάσταση της θύρας, όπως ορίζεται από το 802.1D (Spanning Tree).

1.2.2.3. Συμμετρικά μηνύματα (Symmetric messages)

Τα συμμετρικά μηνύματα αρχικοποιούνται είτε από το switch είτε από τον controller και στέλνονται χωρίς παράκληση.

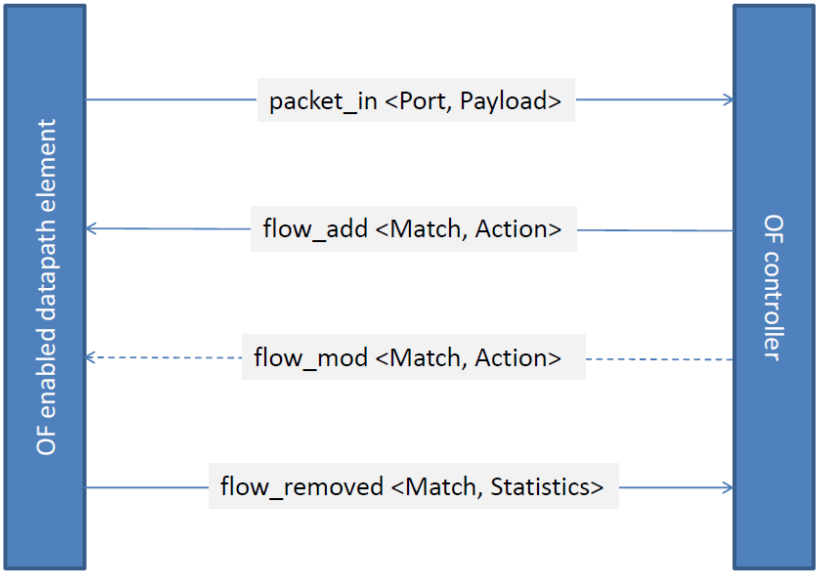
- **Hello:** Τα Hello μηνύματα ανταλλάσσονται ανάμεσα στο switch και στον controller μετά την εγκατάσταση σύνδεσης.
- **Echo:** Τα μηνύματα Echo request/reply μπορούν να σταλούν είτε από το switch είτε από τον controller, και πρέπει να επιστρέψει ένα echo reply. Αυτά τα μηνύματα μπορούν να χρησιμοποιηθούν, για να δείξουν την καθυστέρηση, το εύρος ζώνης και/ή τη ζωτικότητα της σύνδεσης μεταξύ controller και switch.
- **Vendor:** Αυτά τα μηνυμάτα παρέχουν έναν πρότυπο τρόπο για τους OpenFlow switches να προσφέρουν επιπρόσθετη λειτουργικότητα για τα OpenFlow μηνύματα για μελλοντικές προσθήκες στο πρωτόκολλο.

Μετά την παρουσίαση των μηνυμάτων του προτοκόλλου OpenFlow, γίνεται παράθεση της εικόνας που δείχνει την ανταλλαγή αυτών μεταξύ controller και switch.



Σχήμα 4: Ανταλλαγή μηνυμάτων μεταξύ controller και switch

Στη συνέχεια παρουσιάζουμε μία τυπική ροή μηνυμάτων OpenFlow.



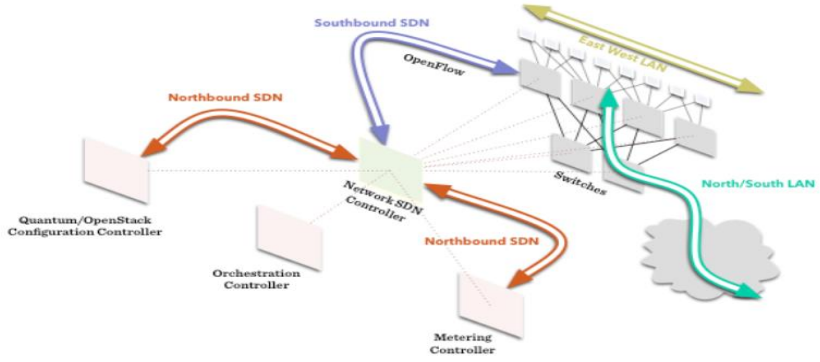
Σχήμα 5: Τυπική ροή ανταλλαγής μηνυμάτων

1.2.3. Southbound και Northbound Διεπαφές

Σε γενικές γραμμές, ο OpenFlow ελεγκτής και οι OpenFlow συσκευές θεωρούνται πως συνδέονται μέσω μιας Southbound API. Στα SDN δίκτυα το ρόλο της Southbound διεπαφής διαδραματίζει το OpenFlow πρωτόκολλο ή κάποιο παρεμφερές [6]. Οι διεπαφές αυτές διευκολύνουν τον αποτελεσματικό έλεγχο στο δίκτυο και καθιστούν τον SDN ελεγκτή ικανό να κάνει αλλαγές σύμφωνα με τις ανάγκες και απαιτήσεις σε πραγματικό χρόνο. Το OpenFlow, που αποτελεί την πρώτη και πιο γνωστή southbound διεπαφή, διαθέτει την κατάλληλη προτυποποίηση, έτσι ώστε ο SDN ελεγκτής να είναι σε θέση να κάνει διάφορες προσαρμογές στο δίκτυο για την καλύτερη εξυπηρέτηση των συνεχώς μεταβαλλόμενων απαιτήσεων των σημερινών εφαρμογών [7].

Επιπλέον οι Northbound APIs χρησιμοποιούνται για την επικοινωνία του SDN ελεγκτή με τις υπηρεσίες και τις εφαρμογές που τρέχουν στο δίκτυο. Οι διεπαφές αυτές μπορούν να διευκολύνουν την αυτοματοποίηση του δικτύου με το να εξυπηρετούν τις ποικίλες ανάγκες διαφορετικών δικτύων [8] .

Στο σχήμα 6 παρουσιάζονται η αρχιτεκτονική SDN δικτύων, καθώς επίσης και οι Northbound και Southbound εφαρμογές αυτής.



Σχήμα 6: Southbound και Northbound διεπαφές στην Αρχιτεκτονική των SDN δικτύων

1.3. Πρωτόκολλο OpenFlow

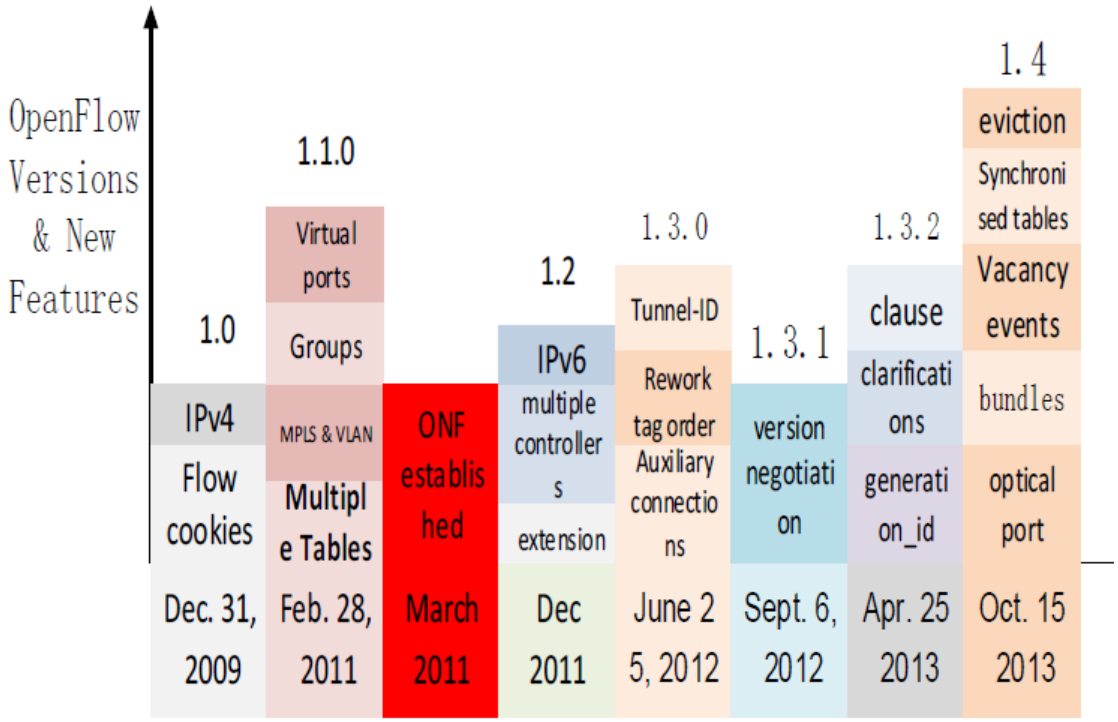
1.3.1. Εισαγωγή στο OpenFlow

Το OpenFlow αποτελεί ένα ανοιχτό πρότυπο, το οποίο επιτρέπει στους ερευνητές να τρέξουν πειραματικά πρωτόκολλα σε δίκτυα. Το OpenFlow προστίθεται σαν ένα νέο χαρακτηριστικό στους εμπορικούς Ethernet μεταγωγείς (switches), δρομολογητές (routers) και ασύρματα δίκτυα πρόσβασης (wireless access points) και παρέχει στους ερευνητές έναν προτυποποιημένο τρόπο να εκτελούν πειράματα, χωρίς προηγουμένως οι κατασκευαστές να χρειάζεται να κάνουν γνωστή την εσωτερική λειτουργία των συσκευών δικτύου. Επιπρόσθετα, το OpenFlow έχει πλέον ενσωματωθεί από τους κύριους κατασκευαστές και στην αγορά είναι διαθέσιμοι μεταγωγείς που υποστηριζουν το OpenFlow πρωτόκολλο. Τέλος πρέπει να επισημανθεί πως η επικοινωνία μεταξύ των μεταγωγέων και του ελεγκτή (controller) πραγματοποιείται διαμέσου του πρωτοκόλλου OpenFlow, το οποίο ορίζει διαφόρου είδους μηνύματα για αυτό το σκοπό, τα οποία περιγράφησαν στην προηγούμενη παράγραφο.

Το OpenFlow πρωτόκολλο [9] είναι η πρώτη διεπαφή για προτυποποίηση της επικοινωνίας οριζόμενη ως μία southbound διεπαφή. Αυτό αποτελεί το πρώτο πρότυπο για SDN και είναι ένα ζωτικής σημασίας στοιχείο της αρχιτεκτονικής των ελεύθερων για τροποποιήσεις ευφυών προγραμματιζόμενων δικτύων. Το OpenFlow προτάθηκε για πρώτη φορά από το Nick McKeown τον Απρίλιο του 2008. Μετά την ίδρυση του Open Network Foundation (ONF) (ένας μη-κερδοσκοπικός οργανισμός αφιερωμένος στην προαγωγή και στην υιοθέτηση των SDN δικτύων μέσα από την ανάπτυξη ανοικτών προτύπων) το Μάρτιο του 2011, το OpenFlow έχει παρουσιάσει τεράστια ανάπτυξη. Τα τελευταία αυτά χρόνια ο ONF έχει δημοσιεύσει διάφορες εκδόσεις του OpenFlow, οι οποίες είναι οι 1.0.0, 1.1.0, 1.2, 1.3.0, 1.3.1, 1.3.2 με τελευταία την 1.4, που δημοσιεύτηκε στις 15 Οκτωβρίου 2013. Η

έκδοση 1.4 βελτίωσε την επεκτασιμότητα του πρωτοκόλλου και πρόσθεσε κάποια νέα χαρακτηριστικά.

Στη συνέχεια παρουσιάζεται ένα σχήμα με την ιστορία και την εξέλιξη του πρωτοκόλλου OpenFlow.



Σχήμα 7: Εκδόσεις του OpenFlow και νέα χαρακτηριστικά

1.3.2. Τα νέα χαρακτηριστικά του OpenFlow 1.4.0

Όπως φαίνεται και από τον Πίνακα 1, τα νέα χαρακτηριστικά του OpenFlow 1.4.0 μπορούν να κατηγοριοποιηθούν σε τρεις διαφορετικούς τύπους: τις προσθήκες, τις βελτιώσεις και τις αλλαγές. Οι προσθήκες είναι τελείως νέα χαρακτηριστικά στο OpenFlow πρωτόκολλο. Οι βελτιώσεις είναι αναθεωρήσεις που εισάγονται στις ήδη υπάρχουσες ιδιότητες για τη βελτίωσή του. Ο τρίτος και τελευταίος τύπος είναι οι αλλαγές που έχουν γίνει στην προεπιλεγμένη TCP θύρα. Στη συνέχεια παρουσιάζονται με λεπτομέρεια τα νέα αυτά στοιχεία των τριών κατηγοριών.

Type	Features
Additions	Bundles Optical ports Synchronized tables
Improvements	Flow-removed reason for meter delete New mechanisms support for multi-controller Eviction and vacancy events More descriptive reasons for packet-in PBB UCA header field New error codes
Change	Change default TCP port to 6653

Πίνακας 1: Νέα χαρακτηριστικά στο OpenFlow 1.4

Α) Προσθήκες στο OpenFlow (Additions)

1. **Bundles**
2. **Συγχρονισμένοι Πίνακες (Synchronized Tables)**
3. **Ιδιότητες οπτικών θυρών (Optical Port Properties)**

1. **Bundles**

Ο συγκεκριμένος μηχανισμός ενσωματώθηκε στην 1.4.0 έκδοση του OpenFlow, με σκοπό να ομαδοποιήσει ένα σύνολο μηνυμάτων OpenFlow σε μία μοναδική λειτουργία, η οποία επιτρέπει την οιονεί ατομική εφαρμογή των σχετικών αλλαγών και τον καλύτερο συγχρονισμό των αλλαγών μέσα από ένα σύνολο μεταγωγέων.

Τα bundles μπορούν να αναγνωρίσουν δύο διαφορετικές λειτουργίες. Η πρώτη ομαδοποιεί τις τροποποιήσεις σε ένα μεταγωγέα, έτσι ώστε όλες να εφαρμοστούν μαζί ή καμία από αυτές να μην εφαρμοσθεί. Αυτό σημαίνει ότι, αν μία αποτύχει, τότε θα αποτύχει το σύνολο των τροποποιήσεων. Ωστόσο μόνο αν όλες εφαρμοσθούν επιτυχώς, τότε οι τροποποιήσεις θα είναι επιτυχώς εφαρμοσμένες. Η δεύτερη λειτουργία έχει ως στόχο να συγχρονίζει καλύτερα τις αλλαγές σε ένα σύνολο OpenFlow μεταγωγέων.

Όταν ένας ελεγκτής θέλει να προσθέσει bundles σε ένα μεταγωγέα, θα στείλει ένα OFPBCT_OPEN_REQUEST μήνυμα, για να εκκινήσει ένα bundle σε αυτό το συγκεκριμένο μεταγωγέα, έπειτα προσθέτει μηνύματα σε κάθε bundle ένα προς ένα. Αν το Νστο μήνυμα αποτύχει, ο μεταγωγέας θα ενημερώσει τον ελεγκτή με ένα μήνυμα απάντησης λάθους. Έπειτα ο τελευταίος θα ελέγξει τον μεταγωγέα, με σκοπό ο τελευταίος να απορρίψει το bundle συμπεριλαμβανομένων όλων των μηνυμάτων μέσα σε αυτό. Κάτω από αυτό το μηχανισμό είναι πιο βολικό για τα μηνύματα να αποσυρθούν, ενώ, εάν εμφανιστεί κάποιο πρόβλημα, το οποίο δεν μπορεί να αμεληθεί, είναι εκείνο στο οποίο εμφανίζεται λάθος και συνεπώς όλα τα μεταδιδόμενα μηνύματα γίνονται μη έγκυρα. Αν ο αριθμός είναι υπερβολικά μεγάλος, τότε η χρησιμοποίηση του καναλιού μειώνεται αισθητά. Βασιζόμενοι σε αυτήν την περίπτωση, θεωρώντας ότι τα bundles μπορούν να εφαρμοστούν σε ποικίλα περιβάλλοντα, είναι πιο λογική η πρόταση για υιοθέτηση της μέγιστης ποσότητας μηνυμάτων. Επιπλέον, αν εφαρμοστεί σε ένα bundle ένας μηχανισμός με ανοχή σε σφάλματα, τότε η αποδοτικότητά του θα αυξηθεί σημαντικά. Στην 1.4 έκδοση του OpenFlow, ο τύπος των μηνυμάτων που μπορούν να προστεθούν σε ένα bundle είναι περιορισμένος.

2. Synchronized Tables

Τα Flow Tables μπορούν να συγχρονιστούν και προς τις δύο κατευθύνσεις ή μόνο προς τη μία. Για οποιοδήποτε συγχρονισμένο table, μία προτεραιότητα αυτού περιγράφει την πηγή από την οποία είναι συγχρονισμένος ο flow table. Αν ο συγχρονισμός είναι αμφίδρομος, οι αλλαγές που γίνονται από τον ελεγκτή σε ένα συγχρονισμένο flow entry χρειάζεται να αντανακλώνται στην πηγή του flow entry.

Ο συγχρονισμένος table εκφράζεται χρησιμοποιώντας μία νέα ιδιότητα που εισήχθη στα χαρακτηριστικά του και είναι η OFPTFPT_TABLE_SYNC_FROM. Καθορίζει το συγχρονισμό μεταξύ δύο flow tables. Ωστόσο δεν ορίζει και δεν εκφράζει την αλλαγή του flow entry μεταξύ δύο flow tables. Η μετάφραση ανάμεσα σε συγχρονισμένες flow entries δεν καθορίζεται από το OpenFlow πρωτόκολλο, αλλά εξαρτάται από τη διαμόρφωση και τις ρυθμίσεις του μεταγωγέα.

Πολλοί μεταγωγείς μπορούν να εκτελέσουν πολλαπλές αναζητήσεις στα ίδια δεδομένα. Για παράδειγμα, ένας standard Ethernet table εκτελεί ένα learning και ένα forwarding lookup στο ίδιο σύνολο των MAC διευθύνσεων. Το συγχρονισμένο table feature καθιστά εφικτή την

αναπαράσταση αυτών των δομών σαν ένα σύνολο από δύο tables, των οποίων τα δεδομένα είναι συγχρονισμένα.

3. Optical Port Properties

Στην έκδοση 1.4 του OpenFlow [4], [5], πολλά μέρη του πρωτοκόλλου έχουν αλλάξει με την προσθήκη των TLV (Type-Length Value) δομών για αυξημένη επεκτασιμότητα. Αυτή η κίνηση επηρέασε πολλές περιοχές του πρωτοκόλλου. Νέα TLVs έχουν προστεθεί σε προηγούμενες δομές με τη μορφή ιδιοτήτων στο τέλος της εν λόγω δομής. Για παράδειγμα, στις δομές θυρών, η προσθήκη ιδιοτήτων περιγραφής, στατιστικών και μετατροπής θύρας. Σε μερικές περιπτώσεις τα ήδη υπάρχοντα TLVs έχουν αλλάξει, με σκοπό να χρησιμοποιήσουν τη δομή της κοινής ιδιότητας TLV. Με την αυξημένη επεκτασιμότητα το πρωτόκολλο έχει τη δυνατότητα να παρέχει με τον ευκολότερο τρόπο την προσθήκη νέων χαρακτηριστικών στο μέλλον και να επεκτείνει επαρκώς το Experimenter Extension API.

Ανάμεσα στις δομές θυρών ένα νέο σύνολο ιδιοτήτων θυρών προστέθηκε, με σκοπό την υποστήριξη οπτικών θυρών. Αυτές περιλαμβάνουν πεδία για τη διαμόρφωση και απεικόνιση της μετάδοσης και λήψης της συχνότητας ενός laser, καθώς επίσης και της οπτικής ισχύος. Αυτές οι νέες ιδιότητες μπορούν να χρησιμοποιηθούν για τη διαμόρφωση και απεικόνιση είτε των οπτικών θυρών Ethernet είτε των οπτικών θυρών σε μεταγωγείς κυκλώματος. Με την οπτική θύρα μπορεί να εφαρμοστεί Fiber Channel over Ethernet (FCoE) στο SDN, το οποίο είναι μία τεχνολογία δικτύων υπολογιστών που ενθυλακώνει πλαίσια Fiber Channel πάνω από Ethernet δίκτυα. Αυτό επιτρέπει στα Fiber Channels να χρησιμοποιούν 10 Gigabit Ethernet δίκτυα (ή υψηλότερες ταχύτητες), εφόσον διατηρούν το Fiber Channel πρωτόκολλο.

Η προσθήκη των οπτικών ιδιοτήτων επιτρέπει στο SDN να υιοθετήσει τεχνολογίες που περιλαμβάνουν fiber channels, όπως είναι τα FCoE. Το FCoE είναι μία τεχνολογία δικτύων υπολογιστών που ενθυλακώνει Fiber Channel πλαίσια πάνω από δίκτυα Ethernet, το οποίο επιτρέπει στο Fiber Channel να χρησιμοποιεί 10 Gigabit Ethernet δίκτυα (ή υψηλότερες ταχύτητες) διατηρώντας παράλληλα το Fiber Channel πρωτόκολλο. Ωστόσο οι τεχνολογίες FCoE είναι κλειστές, πράγμα που εμποδίζει την ευρεία αποδοχή τους και τη διεπικοινωνία μεταξύ συστημάτων διαφορετικών κατασκευαστών. Χάρη στο SDN υπάρχει μεγάλη πιθανότητα για την ανάπτυξη FCoE συστημάτων που είναι βασισμένα σε συνηθισμένες δικτυακές συσκευές με χαμηλά κόστη και μεγάλη διαλειτουργικότητα.

Β) Βελτιώσεις στο OpenFlow πρωτόκολλο

Επιγραμματικά θα αναφερθούν οι βελτιώσεις που ενσωματώθηκαν στο OpenFlow στην έκδοση 1.4.

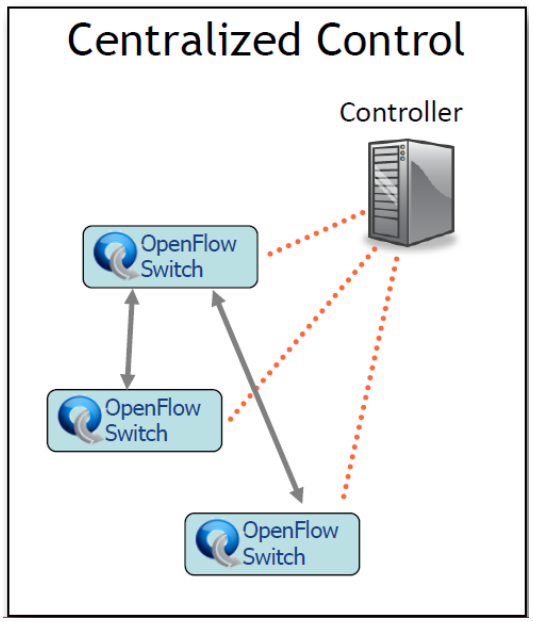
1. ***Αφαίρεση των flow entries με τη διαγραφή του μετρικού***
2. ***Βελτιώσεις για την εισαγωγή πολλαπλών ελεγκτών***
    a. Εισαγωγή του Flow Monitoring
    b. Ειδοποιήσεις σε περίπτωση αλλαγής ενός group ή ενός μετρικού
    c. Καθορισμός από τον ελεγκτή του ρόλου του σε ένα περιβάλλον με πολλαπλούς ελεγκτές
3. ***Γεγονότα απόρριψης ορισμένων flows χαμηλής προτεραιότητας σε περίπτωση πληρότητας του χώρου του flow table***
4. ***Πιο ξεκάθαροι λόγοι για εκτέλεση του Packet-In***
5. ***PBB UCA πεδίο επικεφαλίδας***

C) Αλλαγή της προεπιλεγμένης θύρας από 6633 σε 6653

1.4. Κεντροποιημένος και κατανεμημένος έλεγχος

1.4.1. Κεντροποιημένος έλεγχος

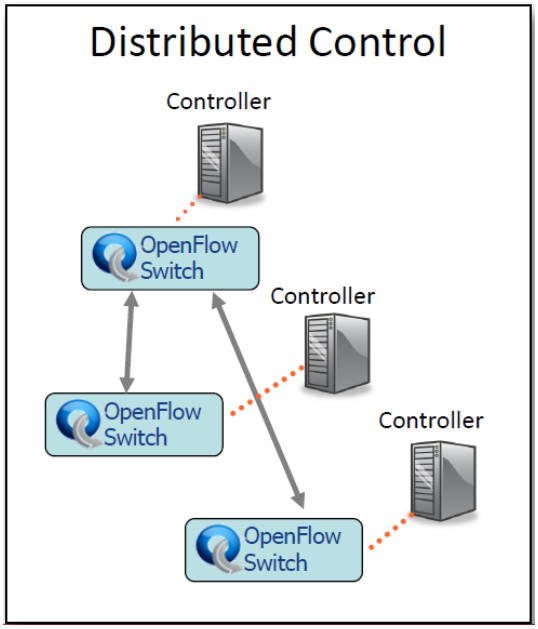
Στον κεντροποιημένο έλεγχο διαθέτουμε έναν ελεγκτή (controller). Ο τελευταίος συνδέεται με κάθε μεταγωγέα του δικτύου και προβαίνει στις απαιτούμενες ενέργειες, όταν χρειαστεί. Οι μεταγωγείς επικοινωνούν μεταξύ τους, αλλά όταν δεν υπάρχει γνώση για το πώς να προωθηθούν τα πακέτα, απευθύνονται στον κοινό ελεγκτή μέσω του πρωτοκόλλου OpenFlow. Η τοπολογία αυτή παρουσιάζεται στην ακόλουθη εικόνα.



Σχήμα 8: Κεντροποιημένος έλεγχος

1.4.2. Κατανεμημένος έλεγχος

Σε αυτού του είδους τον έλεγχο κάθε OpenFlow μεταγωγέας διαθέτει το δικό του ελεγκτή (controller). Σε περίπτωση λοιπόν που δε γνωρίζει κάποιος μεταγωγέας πού να προωθήσει τα εισερχόμενα σε αυτόν πακέτα, έρχεται σε επικοινωνία με το δικό του ελεγκτή, για να αποκτήσει γνώση για την προώθηση αυτών. Η εικόνα που παρουσιάζει την τοπολογία του κατανεμημένου ελέγχου ακολουθεί.



Σχήμα 9: Κατανεμημένος έλεγχος

2. OpenFlow με μεταγωγή κυκλώματος και υποστήριξη οπτικών ιδιοτήτων σε αυτό

2.1. Μεταγωγή κυκλώματος και μεταγωγή πακέτου

2.1.1. Μεταγωγή πακέτου

Σε μία δικτυακή εφαρμογή τα τερματικά συστήματα ανταλλάσσουν μηνύματα μεταξύ τους. Τα μηνύματα μπορεί να περιέχουν οτιδήποτε επιθυμεί ο σχεδιαστής εφαρμογών. Τα μηνύματα είναι δυνατόν να επιτελούν διαφορετικές λειτουργίες, όπως είναι μία λειτουργία ελέγχου, ή να περιέχουν δεδομένα, όπως είναι ένα e-mail, μια JPEG εικόνα ή ένα MP3 audio αρχείο. Για να στείλουμε ένα μήνυμα από ένα τερματικό-πηγή σε ένα τερματικό-προορισμό, η πηγή τεμαχίζει τα μεγάλα μηνύματα σε μικρότερα κομμάτια δεδομένων, γνωστά ως πακέτα [10]. Ανάμεσα στην πηγή και στον προορισμό, κάθε πακέτο ταξιδεύει μέσα από τηλεπικοινωνιακές ζεύξεις και μεταγωγείς πακέτων. Τα πακέτα μεταδίδονται πάνω από μία τηλεπικοινωνιακή ζεύξη με ρυθμό ίσο με τον πλήρη ρυθμό μετάδοσης του καναλιού. Άρα αν το τερματικό πηγή ή μεταγωγέας πακέτου στείλουν ένα πακέτο με μέγεθος *L* bits πάνω από μία ζεύξη με ρυθμό μετάδοσης ίσο με *R* bits/sec, τότε ο χρόνος μετάδοσης του πακέτου ισούται με *L/R* δευτερόλεπτα.

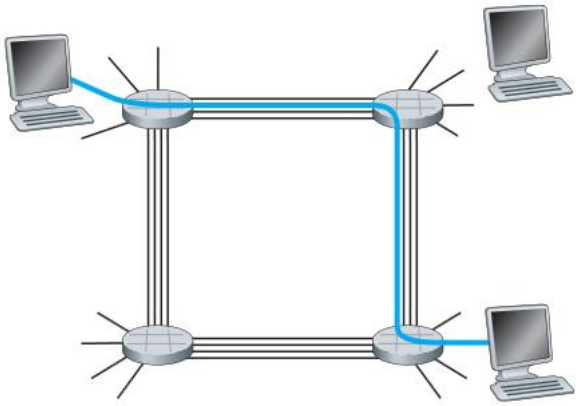
Να επισημανθεί επίσης ότι οι περισσότεροι μεταγωγείς πακέτου χρησιμοποιούν την τεχνική μετάδοσης που τα πακέτα αρχικά αποθηκεύονται και μετά προωθούνται. Πιο συγκεκριμένα, αυτή η τεχνική σημαίνει ότι ο μεταγωγέας πρέπει να λάβει ολόκληρο το πακέτο, πριν ξεκινήσει να μεταδίδει το πρώτο bit του πακέτου στη ζεύξη εξόδου.

2.1.2. Μεταγωγή κυκλώματος

Όπως προαναφέρθηκε στην προηγούμενη υποενότητα, η μεταγωγή πακέτου αποτελεί μία από τις πιο βασικές προσεγγίσεις για μετάδοση δεδομένων μέσα από ένα δίκτυο ζεύξεων και μεταγωγέων. Μία άλλη εξίσου σημαντική μέθοδος είναι η μεταγωγή κυκλώματος που θα παρουσιαστεί στη συνέχεια.

Σε ένα δίκτυο μεταγωγής κυκλώματος [10] οι πόροι που χρειάζονται κατά μήκος ενός μονοπατιού (ενταμιευτές, ρυθμός μετάδοσης ζεύξης) για την επικοινωνία δύο τερματικών δεσμεύονται καθόλη τη διάρκεια της επικοινωνίας μεταξύ των δύο συστημάτων. Στα συστήματα μεταγωγής πακέτου, αυτοί οι πόροι δεν είναι δεσμευμένοι. Τα μηνύματα μιας συνεδρίας χρησιμοποιούν τους πόρους on demand και αυτό ίσως έχει ως αποτέλεσμα την ανάγκη για αναμονή σύνδεσης σε μία τηλεπικοινωνιακή ζεύξη.

Τα παραδοσιακά τηλεφωνικά δίκτυα είναι παραδείγματα δικτύων μεταγωγής κυκλώματος. Όταν κάποιος θέλει να στείλει πληροφορία, π.χ. φωνή σε ένα άλλο τηλεφωνικό δίκτυο, τότε πριν σταλεί οποιαδήποτε πληροφορία, το δίκτυο πρέπει να εγκαταστήσει μία σύνδεση μεταξύ αποστολέα και παραλήπτη. Όταν το δίκτυο εγκαθιστά το κύκλωμα, ταυτόχρονα δεσμεύει ένα σταθερό ρυθμό μετάδοσης στις ζεύξεις δικτύου για τη διάρκεια της κλήσης. Αφού λοιπόν ο ρυθμός μετάδοσης είναι δεσμευμένος εξαρχής για την επικοινωνία, τότε ο αποστολέας μπορεί να μεταδώσει τα δεδομένα του στον παραλήπτη με έναν εγγυημένο ρυθμό μετάδοσης. Στο σχήμα 10 παρουσιάζεται ένα πολύ απλό δίκτυο μεταγωγής κυκλώματος. Οι τέσσερις μεταγωγείς κυκλώματος διασυνδέονται μέσω τεσσάρων ζεύξεων. Κάθε ζεύξη διαθέτει τέσσερα διαφορετικά κυκλώματα, έτσι ώστε κάθε μία να μπορεί να εξυπηρετήσει τέσσερις διαδοχικές συνδέσεις. Τα τερματικά, π.χ. οι προσωπικοί υπολογιστές, είναι απευθείας συνδεδεμένα σε έναν από τους μεταγωγείς. Όταν δύο τερματικά θέλουν να επικοινωνήσουν, το δίκτυο εγκαθιστά μία από άκρο σε άκρο σύνδεση μεταξύ αυτών. Έτσι λοιπόν, αν ο χρήστης Α θέλει να επικοινωνήσει με τον Β, το δίκτυο θα πρέπει πρώτα να να δεσμεύσει ένα κύκλωμα σε κάθε μία από τις δύο ζεύξεις.



Σχήμα 10: Δίκτυο μεταγωγής κυκλώματος

2.1.3. Σύγκριση μεταξύ μεταγωγής πακέτου και μεταγωγής κυκλώματος

Έχοντας περιγράψει και τους δύο τρόπους μεταγωγής, τώρα θα γίνει μία σύντομη σύγκριση μεταξύ των δύο. Είναι γνωστό πως τα δίκτυα μεταγωγής πακέτου αποφεύγονται στις περιπτώσεις υπηρεσιών πραγματικού χρόνου (για παράδειγμα, τηλεφωνικές κλήσεις και βιντεοδιασκέψεις), εξαιτίας των μεταβαλλόμενων και μη προβλέψιμων καθυστερήσεων (κατά κύριο λόγο αυτές οι καθυστερήσεις οφείλονται στις καθυστερήσεις στις ουρές των ενταμιευτών). Από την άλλη πάντως η μεταγωγή πακέτου προσφέρει καλύτερη αξιοποίηση της χωρητικότητας της ζεύξης σε σχέση με τη μεταγωγή κυκλώματος.

2.2. Ενσωμάτωση οπτικών ιδιοτήτων στο OpenFlow πρωτόκολλο

Ένας από τους κύριους στόχους της παρούσας διπλωματικής εργασίας είναι να συμπεριλάβει τις οπτικές ιδιότητες μέσα στο OpenFlow πρωτόκολλο. Αυτό μπορεί να γίνει είτε με την τεχνική της μεταγωγής κυκλώματος είτε μέσω αυτής του πακέτου. Στη συνέχεια περιγράφονται ξεχωριστά οι δύο τρόποι.

2.2.1. Ενσωμάτωση οπτικών ιδιοτήτων με την τεχνική της μεταγωγής κυκλώματος

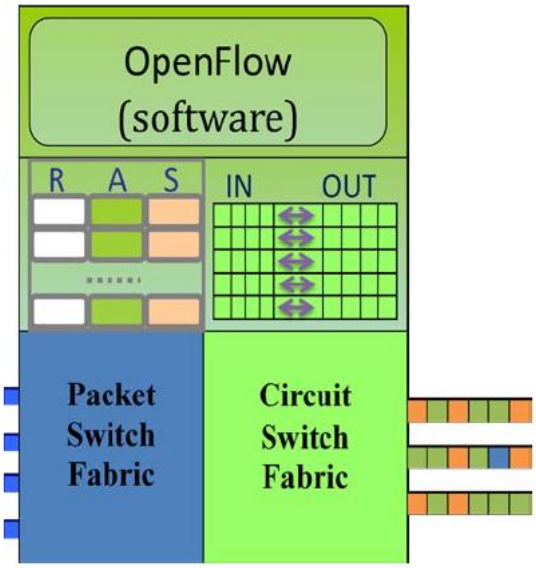
Ένας OpenFlow μεταγωγέας κυκλώματος ή υβριδικός μεταγωγέας [11] αποτελείται από έναν πίνακα διασταυρούμενων συνδέσεων (cross-connect table), ο οποίος αποθηκεύει τις υπάρχουσες ροές με μεταγωγή κυκλώματος και δημιουργεί επίσης ένα ασφαλές κανάλι για την επικοινωνία με έναν εξωτερικό ελεγκτή με τη χρήση του OpenFlow πρωτοκόλλου.

Ο flow table με μεταγωγή κυκλώματος περιλαμβάνει μία σειρά από circuit flow entries, οι οποίες παρουσιάζουν με λεπτομέρεια συνδέσεις ανάμεσα στα κανάλια εισόδου και εξόδου (Σχήμα 11). Επιπρόσθετα διατηρεί μία ή περισσότερες ενέργειες για κάθε circuit flow. Τέλος, περιέχει, όπου κρίνεται απαραίτητο, μία σειρά από στατιστικά για το συγκεκριμένο flow.

Σχήμα 11: Παρουσίαση ενός circuit flow

Σε αντίθεση με τον OpenFlow μεταγωγέα πακέτου, ο cross-connect table δε χρησιμοποιείται για την αναζήτηση των flows, καθώς τυπικά οι θύρες της μεταγωγής κυκλώματος δεν έχουν τη δυνατότητα να δουν στο εσωτερικό των πακέτων, δηλαδή στο payload. Σε έναν circuit switch (οπτικό ή ηλεκτρονικό) δεν υπάρχει ενταμίευση. Αντίστοιχα πακέτα δεν προωθούνται στον controller. Ο controller είναι υπεύθυνος για τον έλεγχο και την απομάκρυνση συνδέσεων σε έναν OpenFlow μεταγωγέα κυκλώματος χάρη στις επεκτάσεις στο πρωτόκολλο OpenFlow για την υποστήριξη των ιδιοτήτων της μεταγωγής κυκλώματος. Μερικοί circuit switches περιλαμβάνουν διεπαφές για πακέτα και μπορούν επιπρόσθετα να κάνουν μεταγωγή πακέτων ηλεκτρονικά, πριν τα στείλουν στις διεπαφές για μεταγωγή κυκλώματος εξόδου. Τέτοιου είδους switches πρέπει να διατηρούν δύο ειδών flow tables, έναν που υποστηρίζει τη μεταγωγή πακέτου και έναν άλλον για την υποστήριξη

της μεταγωγής κυκλώματος. Στο σχήμα 12 παρουσιάζεται ένα παράδειγμα ενός τέτοιου μεταγωγέα.



Σχήμα 12: Ένας OpenFlow μεταγωγέας με ενσωματωμένους flow tables μεταγωγής πακέτου και μεταγωγής κυκλώματος

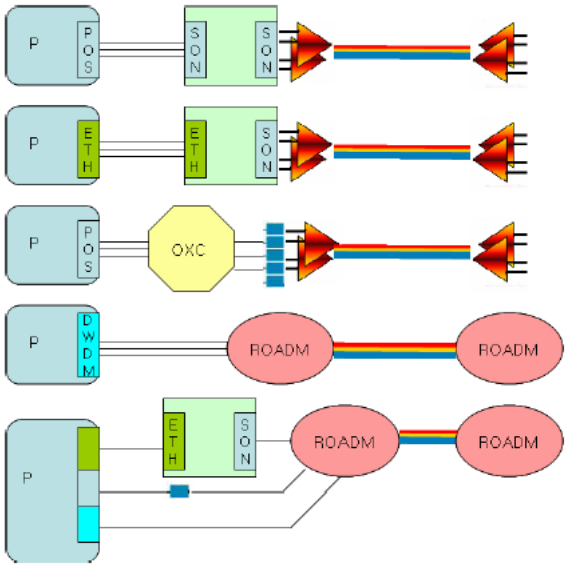
Στη συνέχεια παρουσιάζονται διάφοροι τρόποι διασύνδεσης μεταγωγέων πακέτου και κυκλώματος.

1. Ο μεταγωγέας πακέτου P είναι συνδεδεμένος με τον TDM μεταγωγέα κυκλώματος C μέσω μίας διεπαφής Πακέτου πάνω από Sonet δίκτυο (POS – Packet Over Sonet). Οι μεταγωγείς SONET (DXC) είναι συνδεδεμένοι ο ένας με τον άλλο μέσω στατικών DWDM γραμμών.
2. Στη δεύτερη περίπτωση πάλι ένας μεταγωγέας πακέτου P συνδέεται σε έναν TDM C μέσω μιας διεπαφής Ethernet. Ο μεταγωγέας C έχει τη δυνατότητα να προσαρμόσει τα IP πακέτα από τη διεπαφή Ethernet σε πλαισίωση τύπου SONET.
3. Ο μεταγωγέας P συνδέεται μέσω ενός OXC, όπως μία διασύνδεση ίνας. Η πλαισίωση POS χρησιμοποιείται στη μεταγωγή πακέτου. Αυτή η διεπαφή στο μεταγωγέα πακέτου δε χρησιμοποιεί υψηλής ποιότητας πομπούς-δέκτες, που χρειάζονται για επικοινωνία μεγάλων αποστάσεων, ούτε χρησιμοποιεί τα προτυποποιημένα από την ITU δίκτυα μηκών κύματος. Συνεπώς DWDM transponders χρειάζονται, πριν το σήμα μεταδοθεί μέσω των γραμμών DWDM.
4. Ο μεταγωγέας πακέτου χρησιμοποεί σε αυτήν την περίπτωση DWDM πομπούς-δέκτες με την κατάλληλη χρήση πλαισίων. Το μήκος κύματος στους πομπούς

υπάρχει δυνατότητα να ρυθμιστεί. Το στατικό σύστημα DWDM έχει τώρα αντικατασταθεί από πιο μοντέρνα ROADM/WSS βασιζόμενα σε OXCs.

5. Ο τελευταίος τρόπος διασύνδεσης δείχνει πως αυτή μπορεί να επιτευχθεί με ένα συνδυασμό των ανωτέρω τρόπων. Μερικές διεπαφές στο μεταγωγέα πακέτου θα μπορούσαν να συνδεθούν σε TDM μεταγωγείς κυκλώματος, οι οποίοι με τη σειρά τους συνδέονται μέσω ROADMs. Άλλες διεπαφές μπορούν άμεσα να συνδεθούν στο OXC μέσω transponders ή κατάλληλα ρυθμιζόμενων DWDM διεπαφών.

Για την καλύτερη κατανόηση των παραπάνω μεθόδων παρατίθεται ένα σχήμα που τα παρουσιάζει.



Σχήμα 13: Διάφοροι τρόποι διασύνδεσης των μεταγωγέων πακέτου και κυκλώματος

Ορισμός χρησιμοποιούμενων όρων :

- TDM/DWDM – Time Division Multiplexing/Dense Wavelength Division Multiplexing.
- SONET/SDH – Synchronous Optical NETwork/Synchronous Digital Hierarchy
- DXC/OXC – Digital Cross-connect/Optical Cross-connect
- ROADM/WSS – Reconfigurable Optical Add Drop Multiplexer/Wavelength Selective Switch
- VCAT/VCG/LCAS – Virtual Concatenation/Virtual Concatenation Group/Link Capacity Adjustment Scheme
- POS/GFP – Packet over SONET framing/Generic Framing Procedure
- ITU – International Telecommunications Union

2.2.1.1. Οι επεκτάσεις στο OpenFlow Protocol

Το OpenFlow πρωτόκολλο, όπως προαναφέρθηκε, υποστηρίζει τρεις τύπους μηνυμάτων : controller–to–switch, ασύγχρονα και συμμετρικά, το καθένα μάλιστα με πολλούς υποτύπους.

Αλλαγές στην έκδοση v1.0 του OpenFlow για υποστήριξη της μεταγωγής κυκλώματος

1. Επεκτάσεις στο ofp_switch_features για την υποστήριξη των ιδιοτήτων της μεταγωγής κυκλώματος, οι οποίες δεν υπάρχουν στον κοινό μεταγωγέα πακέτου.
2. Δημιουργία μίας σειράς αριθμών για εικονικές θύρες στο ίδιο namespace με τις φυσικές πόρτες. Δημιουργία της δομής ofp_phy_cport, για να συμπεριλάβει τα χαρακτηριστικά των circuit ports. Η δομή αυτή περιλαμβάνει όλα τα πεδία, τα οποία είναι μέρος των ofp_phy_port. Επίσης υπάρχουν πεδία που αναφέρονται μόνο σε circuit θύρες.
3. Ορισμός της δομής struct ofp_connect για τον καθορισμό της διασύνδεσης στους circuit switches. Αυτή η δομή είναι το λογικό ισοδύναμο της δομής struct ofp_match στην έκδοση για το μεταγωγέα πακέτου.
4. OpenFlow μηνύματα: Η συγκεκριμένη προσέγγιση για μεταγωγή κυκλώματος ορίζει δύο επιπρόσθετα μηνύματα στο enum ofp_type, ένα από αυτά είναι το OFPT_CFLOW_MOD με μία επιπλέον εντολή τροποποίησης του flow, η οποία είναι η OFPFC_DROP. Αυτό το μήνυμα χρησιμοποιείται, για να δημιουργήσει εικονικές πόρτες, καθώς επίσης και για να καθορίσει τις πολλαπλές συνδέσεις. Το άλλο καινούριο μήνυμα είναι το OFPT_CPORT_STATUS με νέα πεδία “reasons” στο ofp_port_reason.
5. Προσθήκη δύο νέων τύπων actions: του OFPAT_CKT_OUTPUT και του OFPAT_CKT_INPUT για την προσαρμογή των flows με μεταγωγή πακέτων σε flows με μεταγωγή κυκλώματος και για την εξαγωγή των flows με μεταγωγή πακέτων από flows με μεταγωγή κυκλώματος.
6. Προσθήκη μηνυμάτων λάθους, για να ενημερώνουν τον ελεγκτή για προβλήματα στις μετατροπές των flows με μεταγωγή κυκλώματος.
7. Τέλος, ορίζεται ένας τρόπος με τον οποίο ο μεταγωγέας θα ενημερώνει τον ελεγκτή ότι μερικά χαρακτηριστικά της θύρας με μεταγωγή κυκλώματος έχουν αλλάξει. Αυτή η λειτουργία θα μπορούσε να βοηθήσει στην επαναφορά της λειτουργίας του δικτύου έπειτα από κάποιο σοβαρό σφάλμα.

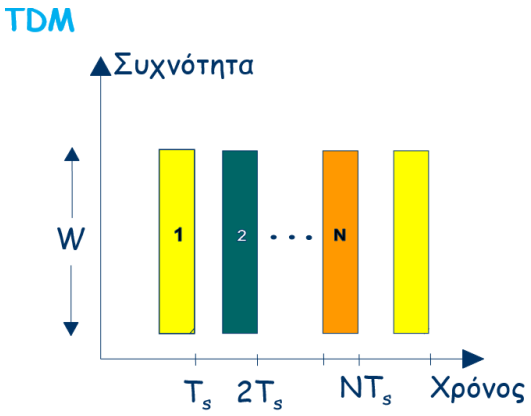
2.2.1.2. Είδη πολυπλεξίας χρησιμοποιούμενα στη μέθοδο μεταγωγής κυκλώματος

Στην περίπτωση των οπτικών ιδιοτήτων με τη μέθοδο της μεταγωγής κυκλώματος τρία είδη πολυπλεξίας μπορούν να χρησιμοποιηθούν. Αυτά είναι : Η Πολυπλεξία Διαίρεσης Χρόνου (Time Division Multiplexing – TDM), η Πολυπλεξία Διαίρεσης Μήκους Κύματος (Wavelength Division Multiplexing – WDM) και η Πυκνή Πολυπλεξία Μήκους Κύματος (Dense Wavelength Division Multiplexing – DWDM).

Στη συνέχεια γίνεται μία σύντομη παρουσίαση των τριών αυτών ειδών πολυπλεξίας.

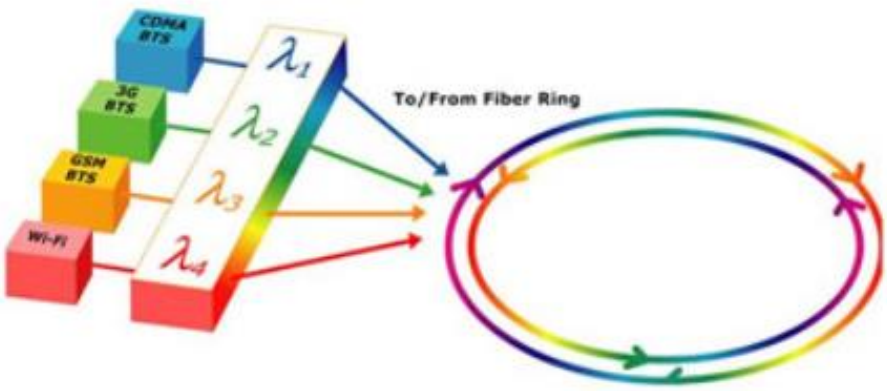
Πολυπλεξία Διαίρεσης Χρόνου (*Time Division Multiplexing-TDM*):

Η Πολυπλεξία Διαίρεσης Χρόνου χωρίζει το χρόνο σε χρονοθυρίδες, έτσι ώστε τα bit από κάθε πηγή εισόδου να μεταφέρονται σε μία συγκεκριμένη χρονοθυρίδα και επομένως τελικά να αυξάνει ο αριθμός αυτών που μεταδίδονται ανά δευτερόλεπτο. Η τεχνική αυτή χρησιμοποιείται από τεχνολογίες, όπως είναι τα Σύγχρονα Οπτικά Δίκτυα (*Synchronous Digital Network-Sonet*) στις ΗΠΑ και η Σύγχρονη Ψηφιακή Ιεραρχία (*Synchronous Digital Hierarchy*) στην Ευρώπη. Οι παραπάνω τεχνολογίες προσφέρουν σημαντικές βελτιώσεις και πλεονεκτήματα στα ευρυζωνικά δίκτυα. Χρησιμοποιούν ηλεκτρο-μεταγωγικά κέντρα, τα οποία συνδέονται με οπτικές ίνες και χρησιμοποιούν *TDM*. Η πολυπλεξία διαίρεσης χρόνου αντιμετωπίζει ορισμένα σημαντικά προβλήματα που δυσκολεύουν τη μετάδοση του σήματος. Αυτά είναι η χρωματική διασπορά και η διασπορά τρόπων πόλωσης.



Σχήμα 14: Πολυπλεξία Διαίρεσης Χρονου

Πολυπλεξία Διαίρεσης Μήκους Κύματος (*Wavelength Division Multiplexing-WDM*):



Σχήμα 15: Πολυπλεξία Διαίρεσης Μήκους Κύματος

Αυτό το είδος πολυπλεξίας χρησιμοποιείται κατά κόρον στις οπτικές επικοινωνίες. Η τεχνολογία αυτή πολυπλέκει έναν αριθμό από οπτικά φέροντα σήματα σε μία οπτική ίνα χρησιμοποιώντας διαφορετικά μήκη κύματος από φως *laser*. Αυτή η τεχνική επιτρέπει τη λειτουργία αμφίδρομων επικοινωνιών πάνω από μία ίνα, πράγμα που οδηγεί σε σημαντική αύξηση της χωρητικότητας. Το σύστημα *WDM* χρησιμοποιεί έναν πολυπλέκτη, για να ενώσει στον εκπομπό τα διαφορετικά σήματα και έναν αποπολυπλέκτη στο δέκτη, ο οποίος με τη σειρά του τα διαχωρίζει. Χάρη σε αυτό και στους οπτικούς ενισχυτές είναι εφικτό να βελτιωθεί σε μεγάλο βαθμό η χωρητικότητα, χωρίς να χρειαστεί να επέμβουμε και να αλλάξουμε κάτι στο υπάρχον δίκτυο. Μπορεί να βελτιωθεί απλά με αναβάθμιση των πολυπλεκτών και αποπολυπλεκτών στο κάθε τερματικό. Η τεχνική που μελετούμε σε αυτήν την παράγραφο αντιμετωπίζει κάθε εισερχόμενο σήμα ανεξάρτητα από τα άλλα. Αυτό σημαίνει ότι κάθε κανάλι έχει το δικό του εύρος ζώνης. Όλα τα σήματα καταφθάνουν την ίδια χρονική στιγμή προκειμένου να αποπολυπλεχθούν και να διαβιβαστούν. Τα *WDM* συστήματα διαχωρίζονται σε δύο διαφορετικές τεχνικές: την Πυκνή Πολύπλεξη Διαίρεσης Μήκους Κύματος (*Dense Wavelength Division Multiplexing-DWDM*), όταν αυξάνεται ο αριθμός των μεταφερόμενων καναλιών, μέσω της διαίρεσης του καθορισμένου εύρους συχνοτήτων σε περισσότερα μήκη κύματος, και της Αραιής Πολύπλεξης Διαίρεσης Μήκους Κύματος (*Coarse Wavelength Division Multiplexing-CWDM*).

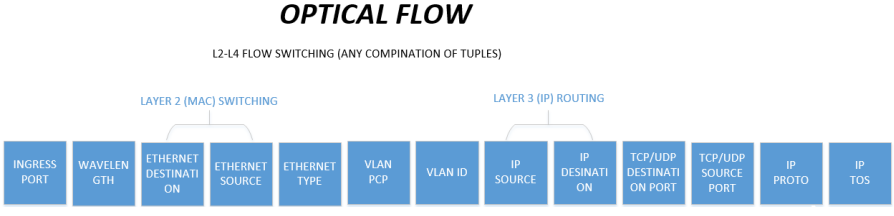
Πυκνή Πολύπλεξη Διαίρεσης Μήκους Κύματος (Dense Wavelength Division Multiplexing – DWDM)

Η μόνη διαφορά αυτής της τεχνικής σε σχέση με την απλή *WDM* έγκειται στο γεγονός ότι η πρώτη χρησιμοποιεί μεγαλύτερο αριθμό μεταφερόμενων καναλιών μέσω της διαίρεσης του διαθέσιμου φάσματος σε περισσότερα μήκη κύματος. Βέβαια αυτός ο μεγαλύτερος αριθμός καναλιών τα οδηγεί σε μικρότερες αποστάσεις μεταξύ τους, γεγονός που μπορεί να προκαλέσει διακαναλική παρεμβολή (*ACI – Adjacent Channel Interference*). Η *DWDM* συναντάται σε υποδιαίρεση 40 καναλιών με απόσταση 100*GHz* ή 80 καναλιών με απόσταση 50*GHz*. Παρόλα αυτά στα σύγχρονα οπτικά δίκτυα καταργείται η σταθερή απόσταση των WDM καναλιών και αντικαθίσταται από μία ευέλικτη κατανομή του συνολικού διαθέσιμου εύρους ζώνης (flexible grid) η οποία μπορεί να μεταβάλλεται ανάλογα με τον τύπο διαμόρφωσης και τον ρυθμό μετάδοσης της πληροφορίας σε ένα κανάλι. Οι ονομαστικές συχνότητες των φέροντων καναλιών απέχουν ακέραια πολλαπλάσια του 6.25GHz με ελάχιστο εύρος 12.5GHz. Ο νέος τρόπος διαχωρισμού του συνολικού εύρους ζώνης έχει σαν αποτέλεσμα την βέλτιστη αξιοποίηση των διαθέσιμων πόρων και την βέλτιση διαχείριση της κίνησης στο οπτικό δίκτυο.

2.2.2. Ενσωμάτωση οπτικών ιδιοτήτων με την τεχνική της μεταγωγής πακέτου

Ένας άλλος τρόπος ενσωμάτωσης των οπτικών ιδιοτήτων είναι με τη μέθοδο της μεταγωγής πακέτου. Με το συγκεκριμένο τρόπο αυτό που πρέπει να γίνει είναι η προσθήκη των οπτικών ιδιοτήτων στο πεδίο της επικεφαλίδας (Header) των flow entries των flow tables. Για χάριν ευκολίας στη συγκεκριμένη περίπτωση θα χρησιμοποιηθεί μόνο μία οπτική ιδιότητα, αυτή του μήκους κύματος.

Η λογική αυτής της τεχνικής έγκειται στο ότι το action του κάθε flow entry καθορίζεται με βάση το μήκος κύματος. Να επισημανθεί μάλιστα πως στην παρούσα διπλωματική εργασία έχει δοθεί έμφαση στα επόμενα κεφάλαια στην ενσωμάτωση οπτικών ιδιοτήτων με χρήση της τεχνικής μεταγωγής πακέτου. Ένα παράδειγμα ενός οπτικού flow δίνεται στο σχήμα που ακολουθεί.



Σχήμα 16: Οπτικό flow με ενσωμάτωση του μήκους κύματος

2.3. Στόχος της παρούσας διπλωτικής εργασίας

Στη σημερινή εποχή με την ραγδαία ανάπτυξη των τηλεπικοινωνιών και των υπολογιστικών συστημάτων η πληροφορία που μεταδίδεται είναι τεράστια και αγγίζει μάλιστα σε τάξη μεγέθους τα exabyte. Η αυξημένη IP κίνηση λόγω των cloud εφαρμοφών (Dropbox, Googledrive), των κοινωνικών δικτύων (Facebook, Twitter, Instagram), καθώς επίσης και ο πολύ μεγάλος αριθμός φορητών αλλά και σταθερών ηλεκτρονικών συσκευών, έχει οδηγήσει στην αναζήτηση νέων μεθόδων για την εξυπηρέτηση της τηλεπικοινωνιακής κίνησης. Επιπλέον τα τελευταία χρόνια υπάρχει μία αυξανόμενη τάση προς τα προγραμματιζόμενα δίκτυα και της εικονοποίησης αυτών λόγω των αρκετών πλεονεκτημάτων που προσφέρουν. Συνεπώς τα εν λόγω δίκτυα σε συνδυασμό με την επέκταση του στο οπτικό επίπεδο μπορούν να οδηγήσουν σε καλύτερη αξιοποίηση των δικτυακών πόρων αλλά σε καλύτερη επίδοση του συστήματος. Τέλος η εμφάνιση ευέλιγκτων συσκευών του δικτύου που είναι και συνάμα προγραμματιζόμενες υποστηρίζουν προσαρμοζόμενους ρυθμούς μετάδοσης και εύρος ζώνης ανάλογα με την εφαρμογή. Συνεπώς όλα τα παραπάνω καθιστούν αναγκαία την χρησιμοποίηση Ευφυών Προγραμματιζόμενων Δικτύων, τα οποία θα μπορούν να λειτουργούν σε πολλαπλά επίπεδα ακόμα και στο οπτικό. Μέρος όλης αυτής της προσπάθειας αποτελεί η επέκταση του πρωτοκόλλου OpenFlow, ώστε να συμπεριλάβει οπτικές παραμέτρους, όπως είναι το μήκος κύματος, το σχήμα διαμόρφωσης, η οπτική ισχύς σε συνδυασμό με την ανάπτυξη μιας διεπαφής για απομακρυσμένο έλεγχο των οπτικών ιδιοτήτων σε οπτικούς μεταγωγείς και πομποδέκτες νέας γενιάς.

Σε αυτό το πλαίσιο η παρούσα διπλωματική εργασία καλείται να συνεισφέρει στα παρακάτω:

1. Ανάπτυξη πλατφόρμας λογισμικού για προσομοίωση μεταγωγής ηλεκτρικών και οπτικών πακέτων με βάση το πρωτόκολλο OpenFlow.
2. Μελέτη των υπάρχοντων μεταγωγέων που υποστηρίζουν Openflow και ανασκόπηση των λειτουργιών/δυνατοτήτων τους.
3. Επέκταση του πρωτοκόλλου OpenFlow για την υποστήριξη οπτικών παραμέτρων μετάδοσης σε ευέλικτα οπτικά δίκτυα.
4. Εκτέλεση απλών τεστ δημιουργίας και μεταγωγής οπτικών πακέτων με χρήση εργαλείων λογισμικού που υποστηρίζουν openflow.

3. Ανάπτυξη πλατφόρμας λογισμικού για δίκτυα που υποστηρίζουν το πρωτόκολλο OpenFlow

Στο παρόν κεφάλαιο θα παρουσιάσουμε τον τρόπο με τον οποίο δημιουργήσαμε την πλατφόρμα εξομοίωσης ενός SDN δικτύου, παρουσιάζοντας αναλυτικά τα βήματα της πορείας μας, καθώς και τα εργαλεία που χρησιμοποιήθηκαν για αυτό το σκοπό.

Για τη δημιουργία της πλατφόρμας χρησιμοποιήθηκαν τα εξής :

1. Το VMware player
2. Ο προσομοιωτής δικτύου Mininet
3. Ο libfluid driver
4. Το Wireshark

3.1. VMware player

Το VMware player [12] είναι ένα εργαλείο, το οποίο επιτρέπει στους χρήστες να δημιουργήσουν και να χρησιμοποιήσουν εικονικές μηχανές στα Windows και σε άλλες VMware πλατφόρμες. Είναι σαν να λειτουργεί κάποιος ένα PC μέσα στο ίδιο του το PC.

Το VMware χρησιμοποιήθηκε στην παρούσα διπλωματική εργασία, διότι ο προσομοιωτής δικτύου Mininet που χρησιμοποιήσαμε λειτουργεί σε περιβάλλον Linux. Επομένως χάρη στο VMware δημιουργήσαμε δύο εικονικές μηχανές με περιβάλλον Linux και δουλέψαμε με την έκδοση 14.04 των Ubuntu.

Επιπλέον το υπολογιστικό σύστημα, που χρησιμοποιήθηκε, αποτελείται από τα ακόλουθα στοιχεία :

- **Επεξεργαστής** : Intel® Core™ i7-4700MQ CPU @ 2.40 GHz
- **Μνήμη RAM** : 8 GB στα 1366 MHz
- **Λειτουργικό σύστημα** : Windows 7 – 64 bit

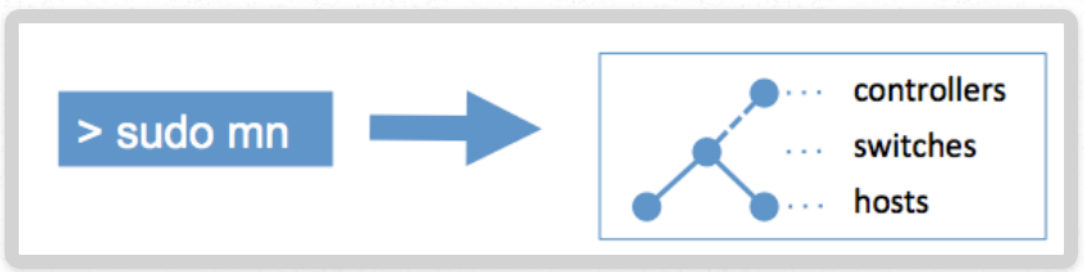
3.2. Mininet

Το Mininet [13] είναι ένας εξομοιωτής δικτύου. Εκτελεί μία συλλογή από τελικά τερματικά, μεταγωγείς, δρομολογητές και ζεύξεις σε ένα μοναδικό Linux kernel. Χρησιμοποιεί χαμηλό από άποψης απαιτήσεων virtualization και κάνει ένα σύστημα να φαίνεται σαν ένα ολοκληρωμένο δίκτυο, εκτελώντας στον ίδιο kernel, σύστημα και user code. Ένα τερματικό του mininet συμπεριφέρεται σα μία πραγματική μηχανή. Μπορεί κάποιος να εκτελέσει ssh

σε αυτό ή να τρέξει μία σειρά από τυχαία προγράμματα. Τα προγράμματα αυτά μπορούν να στέλνουν πακέτα μέσω κάποιας διεπαφής που μοιάζει με πραγματική Ethernet διεπαφή με δοσμένη ταχύτητα ζεύξης και καθυστέρηση. Τα πακέτα δέχονται επεξεργασία από έναν φαινομενικά πραγματικό Ethernet μεταγωγέα, δρομολογητή. Όταν δύο προγράμματα, όπως είναι ένας iperf πελάτης και διακομιστής, επικοινωνούν μέσω του Mininet, η υπολογιζόμενη επίδοση θα έπρεπε να ταιριάζει με εκείνη των δύο (πιο αργών) προυπαρχουσών μηχανών.

Εν συντομία, τα εικονικά τερματικά, μεταγωγείς, ζεύξεις και ελεγκτές του Mininet είναι το πραγματικό γεγονός – δημιουργήθηκαν στην ουσία με τη χρήση λογισμικού παρά υλικού – και για το μεγαλύτερο κομμάτι η συμπεριφορά τους είναι παρόμοια με ξεχωριστά στοιχεία του υλικού. Είναι συχνά πιθανό να δημιουργηθεί ένα δίκτυο Mininet που να μοιάζει με ένα δίκτυο υλικού και το αντίστροφο και να μπορεί κάποιος να τρέξει τον ίδιο δυαδικό κώδικα και εφαρμογές σε οποιαδήποτε πλατφόρμα.

Επιπλέον το Mininet μπορεί, όπως προαναφέρθηκε, να δημιουργήσει ένα ρεαλιστικό εικονικό δίκτυο με τη βοήθεια μίας μόνο εντολής μέσα σε ελάχιστο χρόνο. Αυτή παρουσιάζεται στο σχήμα 17.



Σχήμα 17: Βασική εντολή για τη δημιουργία τοπολογίας στο Mininet

Τα θετικά και τα πλεονεκτήματα [14] του Mininet είναι ποικίλα.

1. Είναι γρήγορο – η εκκίνηση ενός απλού δικτύου παίρνει μόνο λίγα δευτερόλεπτα. Αυτό σημαίνει ότι η επεξεργασία του κώδικα καθώς και η εκτέλεση των πειραμάτων θα είναι αρκετά γρήγορη.
2. Υπάρχει δυνατότητα δημιουργίας διάφορων τοπολογιών: π.χ. με έναν μεταγωγέα, ένα κέντρο δεδομένων κτλ.
3. Δυνατότητα εκτέλεσης πραγματικών προγραμμάτων: οτιδήποτε εκτελείται σε περιβάλλον Linux είναι διαθέσιμο προς εκτέλεση από web servers μέχρι και απεικονιστικά εργαλεία, όπως είναι το Wireshark.

4. Μπορεί ο καθένας να κάνει προσαρμογή της μεταγωγής πακέτων: Οι μεταγωγείς του Mininet είναι προγραμματιζόμενοι χρησιμοποιώντας το OpenFlow πρωτόκολλο. Σχέδια Ευφυών Προγραμματιζόμενων Δικτύων που τρέχουν στο Mininet μπορούν με ευκολία να μεταφερθούν στο υλικό (hardware) των OpenFlow μεταγωγέων για την προώθηση των πακέτων στη γραμμή μεταφοράς.
5. Το Mininet είναι δυνατόν να τρέχει σε ένα laptop, σε έναν server, σε μία εικονική μηχανή, σε ένα ενσωματωμένο Linux box ή στο cloud.
6. Μπορεί ο χρήστης να μοιραστεί και να αντιγράψει αποτελέσματα.
7. Μπορεί να χρησιμοποιηθεί το Mininet πολύ εύκολα: ο χρήστης έχει τη δυνατότητα να δημιουργήσει και να τρέξει πειράματα στο Mininet γράφοντας απλά ή περίπλοκα, ανάλογα με την εφαρμογή, Python κώδικες.
8. Είναι ένα Open Source project, επομένως ο καθένας μπορεί να εξετάσει τους κώδικες που είναι ενσωματωμένοι σε αυτό, να τους τροποποιήσει, να διορθώσει τυχόν λάθη και να υποβάλλει τροποποιήσεις ή και βελτιώσεις.

3.3. Wireshark

Το Wireshark [15] αποτελεί τον αναλυτή πρωτοκόλλων δικτύου που χρησιμοποιείται στο μεγαλύτερο ποσοστό παγκοσμίως. Δίνει τη δυνατότητα να δει ο χρήστης τι συμβαίνει στο δίκτυο σε μικροσκοπική κλίμακα. Χρησιμοποιείται ευρέως σε πολλές επιχειρήσεις αλλά και σε επιστημονικά ινστιτούτα.

Για τη συγκεκριμένη πλατφόρμα χρησιμοποιήθηκε η έκδοση 1.12, διότι περιέχει όλα τα φίλτρα απεικόνισης για πακέτα OpenFlow από τη version 1.0 έως την 1.4.

3.4. Libfluid driver

Ο Libfluid driver [16] είναι ένας σύνδεσμος βιβλιοθηκών που παρέχει τα βασικά χαρακτηριστικά για την εισαγωγή ενός OpenFlow controller.

Έχει δημιουργηθεί από δύο ξεχωριστές βιβλιοθήκες:

- **libfluid_base:** κλάσεις για τη δημιουργία ενός OpenFlow διακομιστή (server), ο οποίος “ακούει” σε συνδέσεις και διαχειρίζεται διάφορα γεγονότα.
- **libfluid_msg:** κλάσεις για τη δημιουργία και το πέρασμα μηνυμάτων OpenFlow.

Το **libfluid_base** ορίζει μία αρχιτεκτονική πελάτη-διακομιστή (client-server), στην οποία ο ελεγκτής (controller) είναι ένας διακομιστής και ο μεταγωγέας είναι ένας πελάτης. Παρέχει

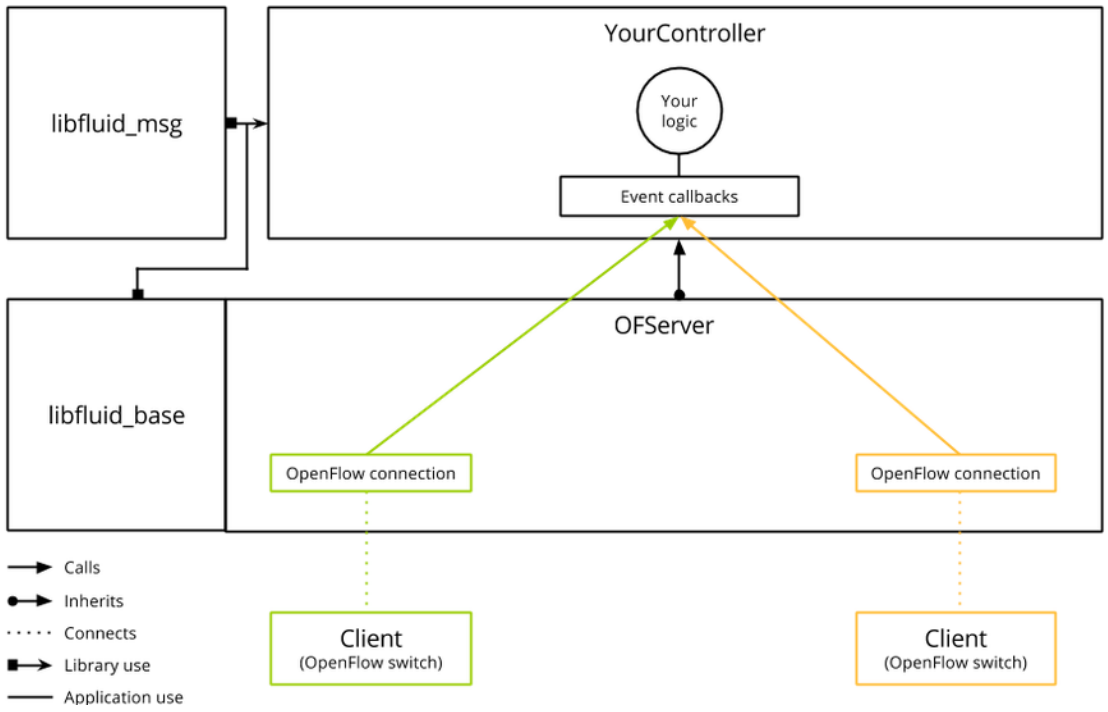
μία βασική κλάση (fluid_base::OFServer) πάνω στην οποία μπορεί ο καθένας να χτίσει τον δικό του controller.

Επιπλέον οι συνδέσεις θα αναπαριστώνται από αντικείμενα της κλάσης fluid_base::OFConnection, τα οποία παρέχουν τις βασικές λειτουργίες μίας OpenFlow σύνδεσης.

Το **libfluid_msg** σε μεγάλο βαθμό απλοποιεί τη δημιουργία και το πέρασμα μηνυμάτων στον controller. Παρέχει κλάσεις για τη δημιουργία OpenFlow μηνυμάτων για τη σειριοποίηση και το αποσειριοποίηση μεθόδων. Σειριοποιώντας τα αποτελέσματα ενός αντικειμένου σε ένα OpenFlow μήνυμα στη δικτυακή ταξινόμηση byte (wire format), τότε αυτά καθίστανται έτοιμα για μετάδοση μέσω μίας OpenFlow σύνδεσης. Για τη σειριοποίηση η βιβλιοθήκη περνάει τα δεδομένα που είναι σε OpenFlow wire format και καθορίζει τα χαρακτηριστικά ενός αντικειμένου σε φορτωμένες τιμές.

Κάθε μήνυμα από το libfluid_msg κληρονομεί στοιχεία από το fluid_msg::OFMsg, το οποίο περιλαμβάνει τις βασικές πληροφορίες της επικεφαλίδας OpenFlow, επιτρέποντας στο χρήστη να δημιουργήσει νέα μηνύματα OpenFlow επεκτείνοντάς το.

Στο σχήμα που ακολουθεί παρουσιάζεται ο τρόπος με τον οποίο η εφαρμογή των κωδίκων του χρήστη μπορεί να χρησιμοποιήσει το libfluid.



Σχήμα 18: Αρχιτεκτονική του libfluid driver

Στο σημείο αυτό θα δημιουργήσουμε μία τοπολογία χρησιμοποιώντας τον libfluid switch ως μεταγωγέα και τον msg_controller ως ελεγκτή. Αυτά τα στοιχεία είναι ενσωματωμένα μέσα στο libfluid driver

```
mininet@mininet-vm:~$ sudo mn --topo single,3 --mac --switch fluid --controller=
remote,ip=127.0.0.1,port=6653
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
*** Starting 1 switches
s1 switch -i s1-eth1 s1-eth2 s1-eth3 1> /tmp/s1-ofd.log 2> /tmp/s1-ofd.log &

*** Starting CLI:
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

Σχήμα 19: Δημιουργία βασικής τοπολογίας με τον libfluid μεταγωγέα

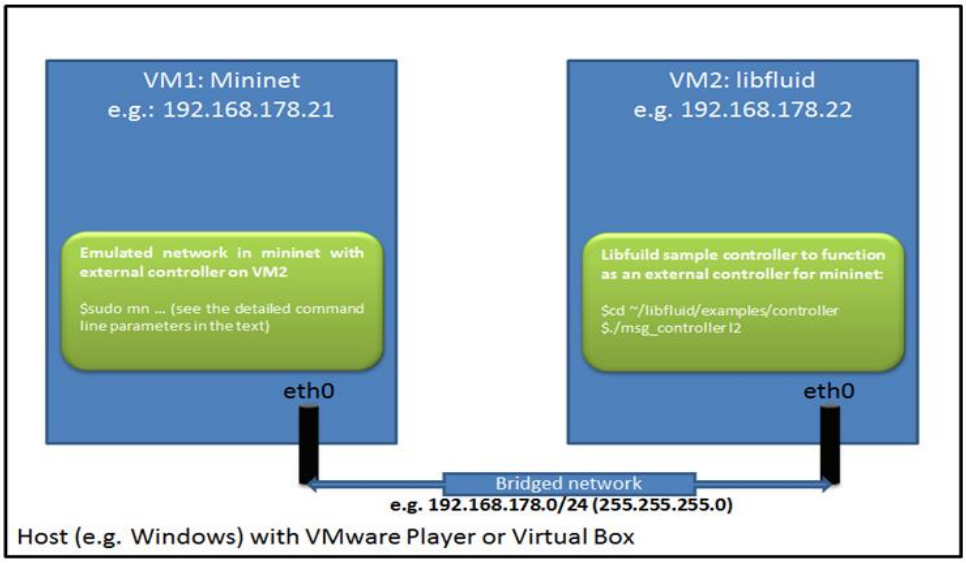
```
 42 8.332452000   127.0.0.1   127.0.0.1   OpenFlow   74 Type: OFPT_HELLO
 44 8.332652000   127.0.0.1   127.0.0.1   OpenFlow   74 Type: OFPT_FEATURES_REQUEST
 45 8.332743000   127.0.0.1   127.0.0.1   OpenFlow   98 Type: OFPT_FEATURES_REPLY
 89 18.830073000  10.0.0.1    10.0.0.2    OpenFlow   184 Type: OFPT_PACKET_IN
 90 18.830457000  127.0.0.1   127.0.0.1   OpenFlow   156 Type: OFPT_FLOW_MOD
114 19.236132000  10.0.0.1    10.0.0.3    OpenFlow   184 Type: OFPT_PACKET_IN
115 19.236356000  127.0.0.1   127.0.0.1   OpenFlow   156 Type: OFPT_FLOW_MOD
167 19.839169000  10.0.0.2    10.0.0.3    OpenFlow   184 Type: OFPT_PACKET_IN
168 19.839392000  127.0.0.1   127.0.0.1   OpenFlow   156 Type: OFPT_FLOW_MOD
```

Σχήμα 20: Μηνύματα OpenFlow που ανταλλάχθηκαν στην εν λόγω τοπολογία

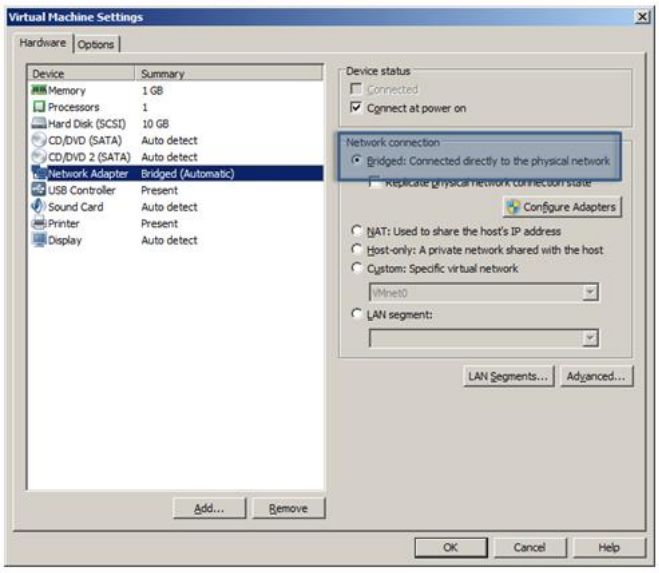
3.5. Δημιουργία βασικής τοπολογίας

Το πειραματικό περιβάλλον, στο οποίο θα λάβει χώρα η ανάπτυξη λογισμικού βασισμένη στον ONF OpenFlow driver (libfluid), περιλαμβάνει δύο εικονικές μηχανές που θα βρίσκονται στο VMware player. Η μία θα τρέχει το Mininet και η άλλη το libfluid controller. Αυτή η διάταξη παρουσιάζεται στο σχήμα 21. Η εικονική μηχανή VM1 τρέχει την εικόνα του Mininet με τη βοήθεια της οποίας δημιουργήθηκε προσομοίωση μιας τοπολογίας δικτύου που συνδέει μεμονομένους χρήστες hosts και openflow switches υπό την εποπτεία ενός controller που επίσης υποστηρίζει OpenFlow. Αν και το Mininet περιέχει κάποιους Openflow controllers, είναι δυνατή η χρήση ενός απομακρυσμένου (remote) controller (όπως είναι ο msg_l2 controller που συμπεριλαμβάνεται στο Libfluid driver), ο οποίος τρέχει σε μια δεύτερη εικονική μηχανή (VM2) μέσω της διάταξης που εικονίζεται στο σχήμα 21. Το σημαντικό σημείο είναι η σωστή λειτουργία αυτής της τοπολογίας έχοντας IP συνδεσιμότητα μεταξύ των δύο VMs. Όπως παρουσιάζεται και από το σχήμα 21, καθένα VM είναι εφοδιασμένο με μία Ethernet διεπαφή (π.χ. eth0). Αυτή η διεπαφή θα πρέπει να

οριστεί κατάλληλα από τις ρυθμίσεις του VMware player. Πιο συγκεκριμένα, πριν την έναρξη λειτουργίας των δύο VMs, η ρύθμιση δικτύου γι' αυτές θα πρέπει να αλλάξει και να έχει οριστεί η Ethernet διεπαφή να λειτουργεί σε "Bridged Mode" (σχήμα 22). Οι προεπιλεγμένες ρυθμίσεις καθορίζουν τη δικτυακή διεπαφή της εικονικής μηχανής να είναι ορισμένη στη ρύθμιση NAT (Network Address Translation). Η ρύθμιση NAT τοποθετεί την Ethernet διεπαφή σε ένα απομονωμένο IP δίκτυο αφαιρώντας της τη δυνατότητα για επικοινωνία με άλλες εικονικές μηχανές. Άρα η σωστή ρύθμιση είναι η "bridged", διότι τοποθετεί τις διεπαφές των δύο εικονικών μηχανών σε κατάσταση γέφυρας. Αυτή η συνδεσμολογία μπορεί να θεωρηθεί ως ένας μεταγωγέας πάνω στον οποίο όλες οι διεπαφές Ethernet είναι συνδεδεμένες.



Σχήμα 21: Επικοινωνία μεταξύ δύο VMs. Στο VM1 τρέχει το Mininet και στο VM2 τρέχει ο libfluid driver



Σχήμα 22 : Ρύθμιση του Network Adaptor για την επικοινωνία των δύο VM

Για την εκτέλεση της προσομοίωσης πρώτα γίνεται εκκίνηση του libfluid controller στο VM2 εκτελώντας στο τερματικό τις ακόλουθες εντολές.

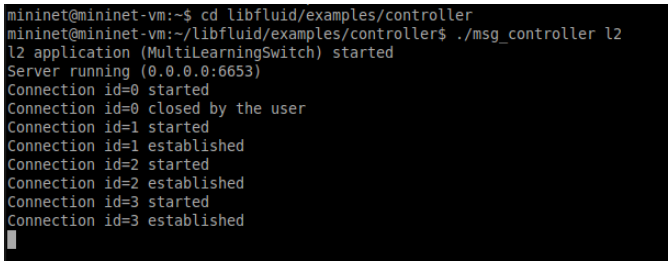
$ cd ~/libfluid/examples/controller

$ ./msg_controller l2

Μετά την εκκίνηση του ελεγκτή, ο οποίος επικοινωνεί μέσω της πόρτας 6653, εκτελούνται οι ακόλουθες εντολές για τη δημιουργία της τοπολογίας. Με την πρώτη εντολή γίνεται η δημιουργία μιας τοπολογίας δένδρου με 3 switches και 4 hosts, οι οποίοι βρίσκονται υπό την εποπτεία ενός controller σε απομακρυσμένη τοποθεσία με την IP 192.168.1.2 και «ακούει» στην θύρα 6653.

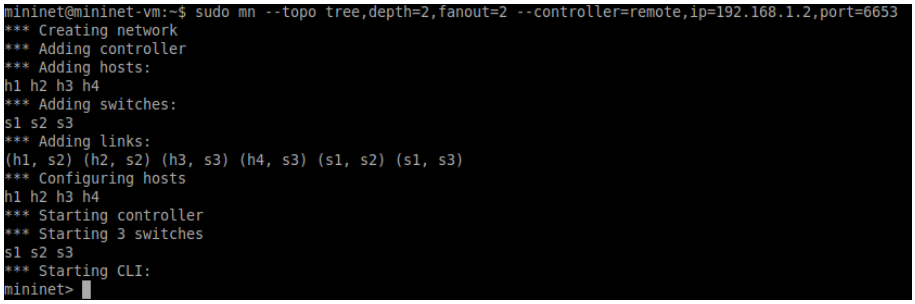
$ sudo mn --topo tree,depth=2,fanout=2 --controller=remote,ip=192.168.1.2,port=6653

Στη συνέχεια παρουσιάζεται η εικόνα από το terminal του controller στο VM2 μετά την εγκατάσταση της τοπολογίας του VM1.



Σχήμα 23: Εκκίνηση του msg controller

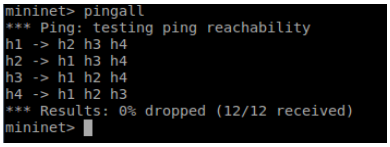
Η εικόνα παρουσιάζει την εγκατάσταση της τοπολογίας.



Σχήμα 24: Δημιουργία τοπολογίας για την επικοινωνία των δύο VMs

Με την εντολή pingall γίνεται έλεγχος της συνδεσιμότητας των δύο χρηστών (host1 και host2) για την ανταλλαγή πακέτων μηνυμάτων μέσω του switch. Στο σχήμα 25 φαίνεται ότι ότι οι χρήστες μπορούν να επικοινωνούν μεταξύ τους.

mininet> pingall

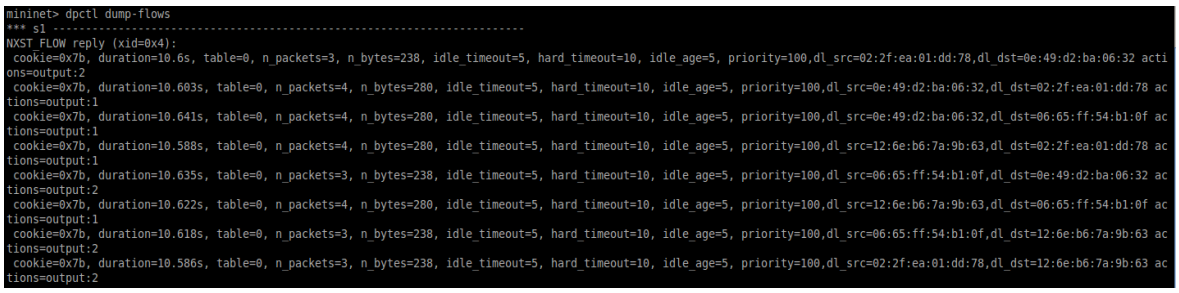


Σχήμα 25: Εκτέλεση της εντολής ping μεταξύ των δύο VMs

Με την εντολή dpctl dump-flows γίνεται εμφάνιση των flows που έχουν εγκατασταθεί στον πίνακα μεταγωγής των switches.

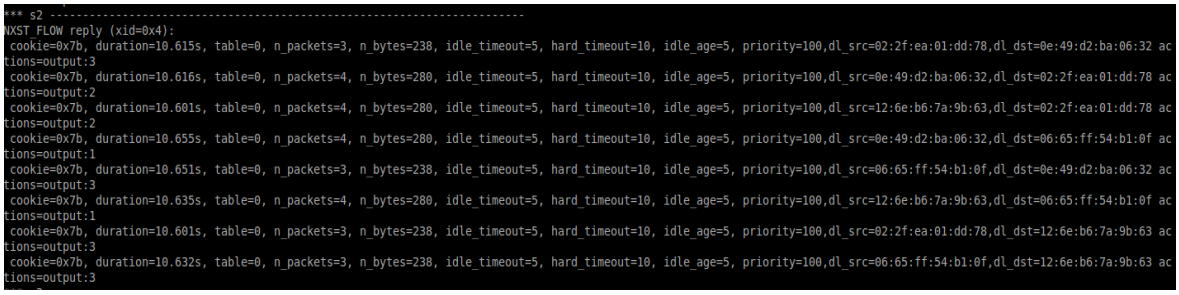
mininet> dpctl dump-flows

Για το switch 1 μας δίνει :



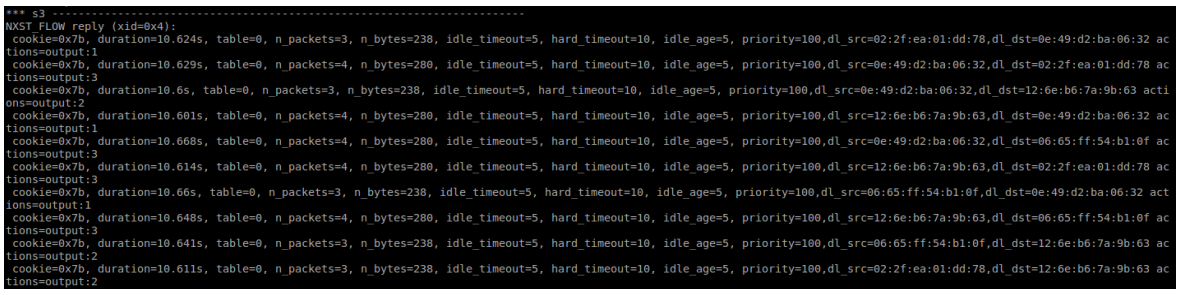
Σχήμα 26: Εκτέλεση της εντολής dpctl dump-flows για το μεταγωγέα 1

Για το switch 2 μας δίνει :



Σχήμα 27: Εκτέλεση της εντολής dpctl dump-flows για το μεταγωγέα 2

Για το switch 3 μας δίνει :



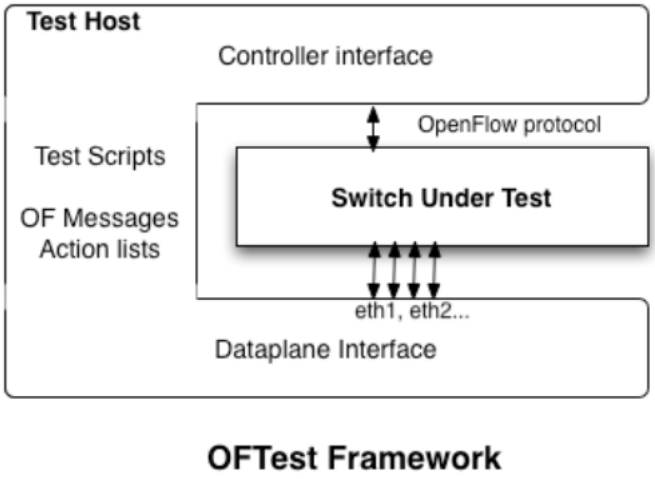
Σχήμα 28: Εκτέλεση της εντολής dpctl dump-flows για το μεταγωγέα 3

4. Εκτέλεση απλών test cases με τη βοήθεια του πρωτοκόλλου OpenFlow

Στο κεφάλαιο αυτό γίνεται παρουσίαση βασικών test cases με τη βοήθεια του εργαλείου OFTest, το οποίο περιλαμβάνεται στην εικονική μηχανή του Mininet. Το OFTest [17] αποτελεί ένα πλαίσιο για την εκτέλεση βασικών tests σε εναν OpenFlow switch, ο οποίος είναι γραμμένος σε γλώσσα Python. Σκοπός του OFTest είναι η εκτίμηση απόδοσης λειτουργίας του εν λόγω switch ο οποίος χρησιμοποιεί το πρωτόκολλο Openflow για την επικοινωνία του με τον controller. Το OFTest βασίζεται στο unittest, το οποίο περιλαμβάνεται στο βασικό πακέτο της Python.

4.1. Παρουσίαση Αρχιτεκτονικής

Στο σχήμα 29 παρουσιάζεται το OFTest Framework και δείχνει τον τρόπο με τον οποίο λειτουργούν αυτά τα βασικά tests.



Σχήμα 29: Πλαίσιο λειτουργίας του OFTEST

Βασικές επισημάνσεις για την προαναφερθείσα αρχιτεκτονική:

- Ο **test host** περιβάλλει το **switch under test (SUT)** (ο μεταγωγέας είναι η συσκευή που γίνονται τα τεστ για τον έλεγχο λειτουργίας του)
- Ο κώδικας της Python για το OFTest τρέχει στο περιβάλλον του test host.
- Ο SUT έχει μία σύνδεση με το επίπεδο ελέγχου και ένα σύνολο από dataplane συνδέσεις.
- Οι διεπαφές του controller και του dataplane του OFTest, που βρίσκονται στον test host, συνδέονται με τις συνδέσεις του SUT.

- Τα στοιχεία του controller και του dataplane έχουν κάποια αντικείμενα Python, που αντιπροσωπεύουν αυτά στο OFTest framework.
- Το OFTest dataplane αντικείμενο παρέχει μεθόδους για την αποστολή και τη λήψη πακέτων από το dataplane.
- Το αντικείμενο του controller παρέχει μεθόδους για την ανταλλαγή OpenFlow μηνυμάτων με το SUT.

Το OFTest framework παρέχει μία πλήρως αντικειμενοστραφή αναπαράσταση των μηνυμάτων OpenFlow και το σχέδιο από το OpenFlow πρωτόκολλο, χάρη στο οποίο θα δημιουργηθούν τα tests.

4.2. Βασικά στοιχεία του unittest

1. Κληρονομικότητα: **SimpleProtocol** κληρονομεί στοιχεία από το **unittest.TestCase.** Αποτελεί τη βάση για όλα τα test cases που παρέχονται στο OFTest.
2. **runTest:** Αποτελεί τη βασική ρουτίνα για ένα test case.
3. **Assertions:** Χρησιμοποίηση των ρουτινών **asserrtTrue, assertEqual** κτλ, για την επαλήθευση συνθηκών, οι οποίες, αν αποτύχουν, συνιστούν ένα test case αποτυχίας.
4. **setUp:** Αυτή η ρουτίνα καλείται αυτόματα από το test framework μετά το τρέξιμο του **runTest.**
5. **tearDown:** Αυτή η ρουτίνα καλείται αυτόματα από το test framework μετά την εκτέλεση του **runTest.**

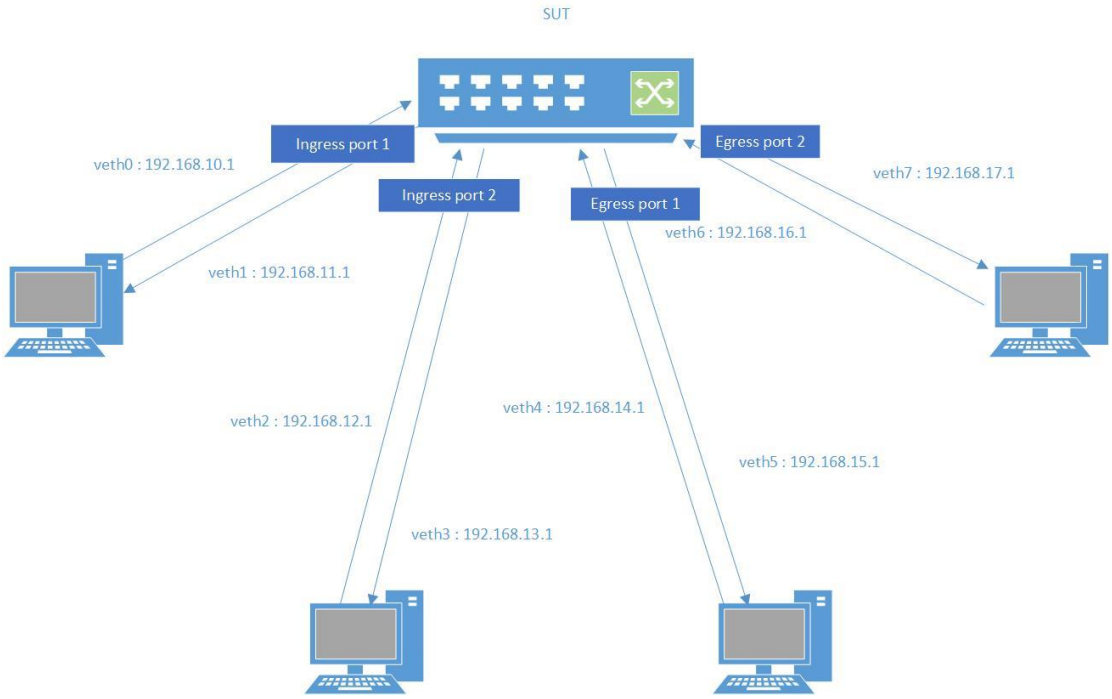
4.3. Βασικές ιεραρχίες κλάσεων και στοιχεία

Τα νέα tests γράφονται ως υποκλάσεις είτε του test του βασικού πρωτοκόλλου, SimpleProtocol (για tests που απαιτούν μόνο επικοινωνία με το μεταγωγέα πάνω από τη διεπαφή του ελεγκτή) είτε του βασικού dataplane test, SimpleDataPlane, το οποίο παρέχει το αντικείμενο του dataplane, για να στείλει και να λάβει πακέτα προς και από τις OpenFlow θύρες.

Να επισημανθεί ότι το **SimpleProtocol** και το **SimpleDataPlane** ορίζονται στο αρχείο tests/basic.py.

4.4. Βασικά παραδείγματα

Στα ήδη υπάρχοντα παραδείγματα έχουν συμπεριληφθεί τέσσερις ακόμα κλάσεις, η my_test, η my_test2, η my_test3 και η my_test4 στο αρχείο flow_stats.py, οι οποίες εκτελούν βασικά και απλά tests. Το ρόλο του μεταγωγέα παίζει ένας πολύ απλός μεταγωγέας που δημιουργήθηκε στα πλαίσια του oftest. Η τοπολογία που δημιουργείται με την εκκίνηση του εν λόγω μεταγωγέα αποτελείται από τον ίδιο και τέσσερις φυσικές διεπαφές. Η κάθε μία φυσική διεπαφή διαθέτει δύο εικονικές με διαφορετική IP διεύθυνση για την εξυπηρέτηση της αμφίδρομης κίνησης δεδομένων για τον host που αντιστοιχίζεται σε αυτή. Η συγκεκριμένη τοπολογία παρουσιάζεται στο σχήμα 30.



Σχήμα 30: Τοπολογία του OFTEST

- ***Class my_test***

Η κλάση αυτή δημιουργεί δύο πακέτα, τα οποία τα ταιριάζει με δύο flows, και στη συνέχεια τα στέλνει σε μία πόρτα εισόδου (ingress port) και από εκεί σε μία εξόδου (egress port). Στην προκειμένη περίπτωση στάλθηκε το πακέτο τύπου 1 από τη θύρα εισόδου 1 (Ingress port 1) στη θύρα εξόδου 2 (Egress port 2) και το πακέτο τύπου 2 από τη θύρα εισόδου 2 (Ingress port 2) στη θύρα εξόδου 1 (Egress port1). Να επισημανθεί επίσης ότι τα πακέτα που στάλθηκαν, το μεν πρώτο ήταν ένα προεπιλεγμένο TCP πακέτο και το δεύτερο ένα TCP πακέτο έχοντας ορίσει αυτή τη φορά τη θύρα πηγής στην τιμή 1281. Επιπλέον με την εντολή match1.wildcards |= ofp.OFPFW_TP_SRC δίνεται η εντολή στο μεταγωγέα να

ταιριάξει τα εισερχόμενα πακέτα στα οποία είναι ορισμένη μόνο η TCP θύρα πηγής και να απορρίψει οποιαδήποτε άλλα πακέτα που έχουν ορίσει κάποιο διαφορετικό πεδίο στην επικεφαλίδα. Στην παρακάτω εικόνα φαίνεται η επιτυχής εκτέλεση του test.

```
mininet@mininet-vm:~/oftest$ sudo ./oft flow_stats.my_test
WARNING: No route found for IPv6 destination :: (no default route?)
flow_stats.my_test ... ok

----------------------------------------------------------------------
Ran 1 test in 1.124s

OK
mininet@mininet-vm:~/oftest$
```

Σχήμα 31: Επιτυχής εκτέλεση του my_test

Πιο αναλυτικά θα γίνει παρουσίαση του τρόπου που πραγματοποιείται το ταίριασμα. Ο πίνακας 2 παρουσιάζει την κατάσταση στην οποία ο μεταγωγέας έχει ορίσει όλα του τα πεδία να είναι wildcarded, δηλαδή να ταιριάζει οποιοδήποτε εισερχόμενο πακέτο.

nw_tos	dl_vlan_pcp	nw_dst_mask	nw_dst_all	nw_src_mask	nw_src_all	tp_dst	tp_src	nw_proto	dl_type	nw_dst_shift	dl_dst	nw_src_shift	nw_src_bits	nw_dst_bits	dl_src	dl_vlan	in_port
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Πίνακας 2: Όλα τα πεδία του flow entry είναι ορισμένα ως wildcarded

Στη συνέχεια στο τεστ καλείται η μέθοδος packet_to_flow_match_v1 και μετά από κατάλληλους υπολογισμούς κατά την επιστροφή του match από την εν λόγω μέθοδο, τα wildcards έχουν διαμορφωθεί ως εξής.

nw_tos	dl_vlan_pcp	nw_dst_mask	nw_dst_all	nw_src_mask	nw_src_all	tp_dst	tp_src	nw_proto	dl_type	nw_dst_shift	dl_dst	nw_src_shift	nw_src_bits	nw_dst_bits	dl_src	dl_vlan	in_port
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Πίνακας 3: Το match γίνεται μόνο με την τιμή της θύρας εισόδου

Εν προκειμένω τα πακέτα έχουν ορισμένη μόνο τη θύρα εισόδου σε αυτό το σημείο του τεστ.

Τέλος ορίζεται και το πεδίο στο οποίο θέλουμε να κάνουμε match και στη συγκεκριμένη περίπτωση είναι η TCP θύρα εισόδου. Επομένως ο πίνακας των wildcards διαμορφώνεται, όπως φαίνεται στη συνέχεια.

nw_tos	dl_vlan_pcp	nw_dst_mask	nw_dst_all	nw_src_mask	nw_src_all	tp_dst	tp_src	nw_proto	dl_type	nw_dst_shift	dl_dst	nw_src_shift	nw_src_bits	nw_dst_bits	dl_src	dl_vlan	in_port
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1

Πίνακας 4: Το match πραγματοποιείται με την τιμή της θύρας εισόδου και με την τιμή της TCP θύρας πηγής

Ο πίνακας 4 δηλώνει πως τα πακέτα, στα οποία έχει οριστεί η TCP θύρα πηγής, καθώς και η θύρα εισόδου του μεταγωγέα, θα γίνονται δεκτά.

- ***Class my_test2***

Η συγκεκριμένη κλάση δημιουργεί τρία διαφορετικά πακέτα και τα ταιριάζει το καθένα με ένα flow entry. Βέβαια μέσα στο τεστ έχει οριστεί να γίνεται επιτυχές ταίριασμα μόνο με τα πακέτα, στα οποία έχουν ορισμένη την Ethernet διεύθυνση προορισμού. Έτσι το τρίτο είδος πακέτου στο οποίο έχει περαστεί ως παράμετρος η Ethernet διεύθυνση πηγής δεν μπορεί να κάνει το ταίριασμα και γι' αυτό το λόγο τα μηνύματα αυτά απορρίπτονται. Να επισημανθεί ακόμα πως το τεστ τρέχει μόνο στην περίπτωση που και τα τρία πακέτα έχουν οριστεί να ταιριάζουν με το eth_dst. Αν στο τρίτο πακέτο έχει οριστεί η διεύθυνση πηγής αντί για αυτή του προορισμού, τότε το τεστ σταματάει, διότι δεν μπορεί να κάνει επαλήθευση σωστής αποστολής του τρίτου είδους πακέτου.

Το σχήμα 32 παρουσιάζει την επιτυχή εκτέλεση του my_test2 μετά από την αποστολή και των τριών πακέτων με ορισμένη την Ethernet διεύθυνση προορισμού.

```
mininet@mininet-vm:~/oftest$ sudo ./oft flow_stats.my_test2
WARNING: No route found for IPv6 destination :: (no default route?)
flow_stats.my_test2 ... ok

----------------------------------------------------------------------
Ran 1 test in 1.175s

OK
```

Σχήμα 32: Επιτυχής εκτέλεση του my_test2

Στο σημείο αυτό παρουσιάζονται οι πίνακες του wildcards για τα τρία πακέτα, όπου τα δύο πρώτα θα έχουν ορισμένη τη διεύθυνση προορισμού και το τελευταίο τη διεύθυνση πηγής.

nw_tos	dl_vlan_pcp	nw_dst_mask	nw_dst_all	nw_src_mask	nw_src_all	tp_dst	tp_src	nw_proto	dl_type	nw_dst_shift	dl_dst	nw_src_shift	nw_src_bits	nw_dst_bits	dl_src	dl_vlan	in_port
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1

Πίνακας 5: Match με την τιμή της θύρας εισόδου και της Ethernet διεύθυνση προορισμού

nw_tos	dl_vlan_pcp	nw_dst_mask	nw_dst_all	nw_src_mask	nw_src_all	tp_dst	tp_src	nw_proto	dl_type	nw_dst_shift	dl_dst	nw_src_shift	nw_src_bits	nw_dst_bits	dl_src	dl_vlan	in_port
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Πίνακας 6: Ανεπιτυχές match λόγω αποστολής της Ethernet διεύθυνσης πηγής αντί για την ορισμένη να κάνει match Ethernet διεύθυνση προορισμού

Στον πίνακα 6 λόγω του ότι το bit που αντιπροσωπεύει την Ethernet διεύθυνση προορισμού είναι ορισμένο στο 0, δεν μπορεί να γίνει το ταίριασμα με τα πακέτα που έχουν ορισμένη παράμετρο τη διεύθυνση προορισμού και αυτό έχει ως αποτέλεσμα την αστοχία λήψης του πακέτου.

4.5. Έλεγχος του υπό εξέταση μεταγωγέα με βάση το μήκος κύματος.

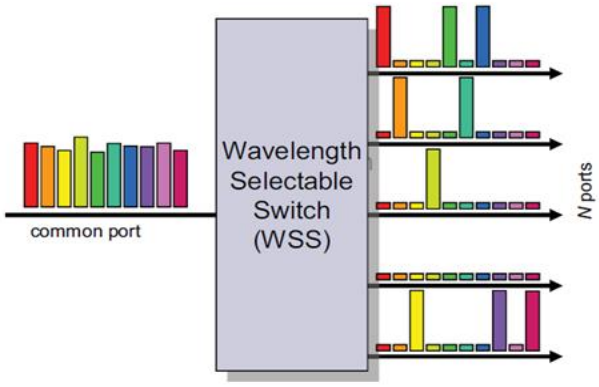
Το τρίτο και βασικό τεστ που πραγματοποιήθηκε είχε ως στόχο την ενσωμάτωση μίας οπτικής ιδιότητας στα flow entries του flow table του μεταγωγέα. Αυτή την ιδιότητα αποτελεί το μήκος κύματος (wavelength). Η διαδικασία για την εισαγωγή αυτού του χαρακτηριστικού παρουσιάζεται λεπτομερώς στη συνέχεια. Στην ουσία το τεστ αυτό που επιδιώκει είναι να εξομοιώσει τη λειτουργία ενός WSS (Wavelength Selective Switch), ο οποίος θα μεταγάγει τα πακέτα με βάση το μήκος κύματος που είναι ενσωματωμένο στα πεδία της επικεφαλίδας του εν λόγω πακέτου.

4.5.1. Wavelength Selective Switch (WSS)

Οι Wavelength Selective Switches (WSS's) [18] αποτελούν την καρδιά των σύγχρονων DWDM οπτικών δικτύων. Λειτουργεί ως ένα οπτικό στοιχείο, το οποίο είναι υπεύθυνο για τη δυναμική δρομολόγηση, το μπλοκάρισμα και την εξασθένηση όλων των DWDM μηκών

κύματος σε έναν κόμβο το δικτύου ανάλογα με την περίσταση και τις ανάγκες της εφαρμογής που θέλει να εξυπηρετήσει. Ως εκ τούτου η λειτουργία αυτής της συσκευής είναι γενικά ένα από τα πιο καθοριστικά χαρακτηριστικά, τα οποία διαμορφώνουν μάλιστα και την επίδοση ολόκληρου του δικτύου. Για αυτό το λόγο η επιλογή ενός WSS με επαρκή και καλή λειτουργία αποτελεί ένα πολύ κρίσιμο ζήτημα.

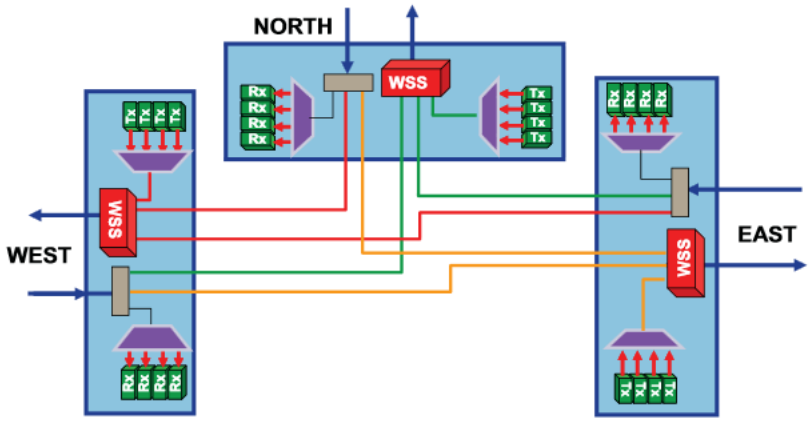
Η λειτουργία του WSS παρουσιάζεται και στο σχήμα 33. Αποτελείται από μία μοναδική κοινή οπτική θύρα και N θύρες που μπορούν να εξυπηρετήσουν πολλαπλά διαφορετικά μήκη κύματος. Αυτό σημαίνει ότι κάθε DWDM μήκος κύματος που εισέρχεται από την κοινή θύρα μπορεί να δρομολογηθεί σε οποιαδήποτε από τις N πολλαπλών μηκών κύματος θύρες, ανεξάρτητα από τον τρόπο που έχουν δρομολογηθεί τα άλλα κανάλια. Η συγκεκριμένη λειτουργία είναι δυναμική και μπορεί να ελεγχθεί από μία ψηφιακή διεπαφή στο WSS. Η λειτουργικότητα του WSS είναι οπτικά αμφικατευθυντική. Αυτό σημαίνει ότι αν πολλαπλά σήματα στο ίδιο DWDM μήκος κύματος φτάσουν σε μία από τις ανεξάρτητες N θύρες, μόνο ένα από αυτά θα μπορέσει να επιλεγχθεί, για να περάσει στην κοινή θύρα. Επιπρόσθετα κάθε δρομολογημένο μήκος κύματος υπάρχει η δυνατότητα να εξασθενηθεί ανεξάρτητα για λόγους ελέγχου της ισχύος του καναλιού.



Σχήμα 33: 1XN WSS

Το σχήμα 34 παρουσιάζει μία τυπική αρχιτεκτονική με τρεις κόμβους, όπου ο καθένας έχει ενσωματωμένο από έναν WSS. Για κάθε μία ίνα μετάδοσης που εισέρχεται στον κόμβο, ένα οπτικό αντίγραφο όλων των DWDM καναλιών σε αυτήν την ίνα κατευθύνεται σε μία εκ των πολλαπλών μηκών κύματος θύρες εισόδου του κάθε WSS. Οι κοινές θύρες καθενός WSS κατευθύνονται σε διαδοχικές ίνες εξόδου. Ως εκ τούτου κάθε κανάλι μήκους κύματος που είναι παρών σε κάθε κόμβο οδηγείται σε κάθε WSS. Για κάθε κανάλι μήκους κύματος η

επιλογή του σήματος που θα δρομολογηθεί στην κοινή θύρα καθορίζει ποια συγκεκριμένα σήματα θα δρομολογηθούν διαμέσου συγκεκριμένων ινών μετάδοσης έξω από τον κόμβο.



Σχήμα 34: Τυπική αρχιτεκτονική οπτικού κόμβου με τη χρήση WSS

Την τρέχουσα εποχή τα στοιχεία των WSS παρέχονται από διάφορους κατασκευαστές και χρησιμοποιούν διάφορες τεχνολογίες. Κάποιες από αυτές είναι οι ακόλουθες: Micro-electromechanical system (MEMS), Liquid Crystal on Silicon (LCoS) και Liquid Crystal (LC).

4.5.2. Τροποποίηση κωδίκων OFTest για την υποστήριξη δημιουργίας και μεταγωγής οπτικών πακέτων

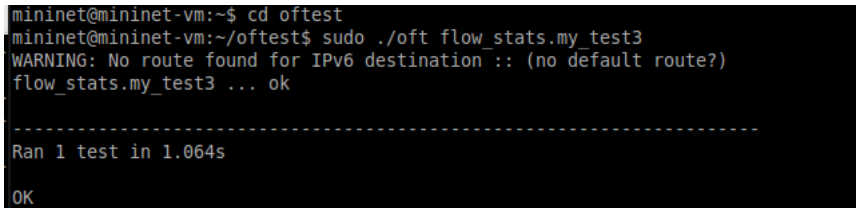
Πρώτα απ' όλα σε πρώτο στάδιο ήταν αναγκαία η δημιουργία ενός καινούριου layer στο φάκελο του scapy. Το scapy είναι ένα εργαλείο διαχείρισης πακέτων για δίκτυα υπολογιστών γραμμένο σε python. Επομένως προστέθηκε ένα αρχείο με όνομα **optical_layer.py** , που περιλαμβάνει τα πεδία του layer αυτού. Για ευκολία δημιουργήθηκε μόνο ένα πεδίο, το wavelength. Στη συνέχεια το οπτικό layer θα έπρεπε να φορτωθεί από το scapy, διαδικασία που διεκπεραιώθηκε μέσα στο αρχείο **fields.py.**

Μετά τις απαραίτητες τροποποιήσεις στο φάκελο του scapy, ήταν αναγκαίες και οι αλλαγές στο φάκελο του oftest, από όπου πραγματοποιούνται τα test. Η εκδοση του πρωτοκόλλου που χρησιμοποιήθηκε είναι η 1.0, άρα οι όποιες τροποποιήσεις έγιναν μέσα σε αυτόν. Πιο συγκεκριμένα, στο αρχείο **const.py** προστέθηκε ένα νέο αναγνωριστικό στα wildacards για το μήκος κύματος. Η προσθήκη του πεδίου για το μήκος κύματος επετεύχθη με διαφοροποίηση των τιμών των wildcards. Για παράδειγμα, όταν όλα τα πεδία είναι wildcards, τότε το match.wildcards έχει τιμή 4194303. Με την προσθήκη ενός επιπλέον bit στην $23^{η}$ θέση για την αναπαράσταση του μήκους κύματος, η τιμή των wildcards_all γίνεται πλέον 8388607.

- ***Class my_test3***

Το πρώτο test που εκτελέστηκε με τις οπτικές παραμέτρους είναι το my_test3. Σκοπός του είναι η δημιουργία ενός αριθμού οπτικών πακέτων και η μεταγωγή του από μία θύρα εισόδου του μεταγωγέα σε μία θύρα εξόδου με κριτήριο το ταίριασμα των wildcards με βάση το μήκος κύματος. Μετά από το επιτυχές ταίριασμα των wildcards ακολουθεί και έλεγχος και της τιμής αυτού, η οποία εμπεριέχεται μέσα στο οπτικό πακέτο.

Στην παρακάτω εικόνα παρουσιάζεται η επιτυχής εκτέλεση του my_test3.



Σχήμα 35: Επιτυχής εκτέλεση του my_test3

Στο σημείο αυτό θα γίνει περιγραφή του τρόπου με τον οποίο λειτουργεί το συγκεκριμένο test. Πλέον ο πίνακας των wildcards έχει αυξηθεί κατά ένα bit για την αναπαράσταση του μήκους κύματος. Επομένως όταν το flag των wildcards για το μήκος κύματος είναι ορισμένο στο 1, αυτό αναπαριστά τον δεκαδικό αριθμό 4194304. Ο έλεγχος που πραγματοποιεί το συγκεκριμένο τεστ είναι διαφοροποιημένος σε σχέση με αυτόν των δύο προηγουμένων. Όταν σταλεί ένα οπτικό πακέτο και έχουμε ορίσει το ταίριασμα να γίνεται με βάση το μήκος κύματος, μόνο τότε το πακέτο θα φτάσει στη θύρα εξόδου του μεταγωγέα επιτυχώς. Αν για παράδειγμα σταλεί οπτικό πακέτο και έχει οριστεί το ταίριασμα να γίνεται με βάση την διεύθυνση Ethernet προορισμού, τότε δε θα υπάρξει επιτυχής ταύτιση πεδίων και το πακέτο θα απορριφθεί. Πιο συγκεκριμένα για την επιτυχή αποστολή πακέτου ο πίνακας των wildcards θα πρέπει να έχει την παρακάτω μορφή.

wave	nw_tos	dl_vlan_pcp	nw_dst_mask	nw_dst_all	nw_src_mask	nw_src_all	tp_dst	tp_src	nw_proto	dl_type	nw_dst_shift	dl_dst	nw_src_shift	nw_src_bits	nw_dst_bits	dl_src	dl_vlan	in_port
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Πίνακας 7: Ο πίνακας των wildcards κατά την επιτυχημένη αποστολή του οπτικού πακέτου

Ο πίνακας 7 μας δείχνει ότι μετά την εκτέλεση της packet_to _flow_match τα πεδία που θα γίνεται το ταίριασμα, θα πρέπει να είναι μηδενικά και στην προκειμένη περίπτωση αυτά είναι το μήκος κύματος και η θύρα εισόδου. Στη συνέχεια εκτελείται η πράξη του δυαδικού AND ανάμεσα στον πίνακα 7 και στα required wildcards (Τα required wildcards ορίζουν 1 τα πεδία στα οποία πρέπει να γίνει το ταίριασμα – Πίνακας 8). Τα τελευταία αποτελούν τον πίνακα που περιέχει άσσους μόνο στα πεδία, όπου θέλουμε να γίνεται το ταίριασμα. Επομένως αν έχει γίνει ακριβές ταίριασμα, τότε τελικά τα πεδία του match θα πρέπει να γίνουν όλα μηδενικά (στην περίπτωση του οπτικού πακέτου). Αν συμβεί αυτό, το τεστ αναθέτει στα πεδία του μήκους κύματος και της θύρας εισόδου την τιμή 1 και σε όλα τα άλλα την τιμή 0. Επομένως τελικά ο πίνακας έχει την ακόλουθη δομή.

wave	nw_tos	dl_vlan_pcp	nw_dst_mask	nw_dst_all	nw_src_mask	nw_src_all	tp_dst	tp_src	nw_proto	dl_type	nw_dst_shift	dl_dst	nw_src_shift	nw_src_bits	nw_dst_bits	dl_src	dl_vlan	in_port
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Πίνακας 8: Required Wildcards για οπτικό πακέτο

Σε περίπτωση αποστολής διαφορετικού τύπου πακέτο εκτός από οπτικό, π.χ. TCP πακέτο, ο πίνακας των wildcards μετά την εκτέλεση του δυαδικού AND θα έχει τη δομή που φαίνεται στον πίνακα 9. Πλέον το match1.wildcards δε θα έχει την τιμή 0, άρα οδηγεί σε απόρριψη αυτού.

wave	nw_tos	dl_vlan_pcp	nw_dst_mask	nw_dst_all	nw_src_mask	nw_src_all	tp_dst	tp_src	nw_proto	dl_type	nw_dst_shift	dl_dst	nw_src_shift	nw_src_bits	nw_dst_bits	dl_src	dl_vlan	in_port
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Πίνακας 9: Ο πίνακας των wildcards κατά το αποτυχημένο ταίριασμα των required wildcards και του flow entry

Σε περίπτωση που γίνει επιτυχές ταίριασμα, στη συνέχεια γίνεται και έλεγχος της τιμής του οπτικού πακέτου που στάλθηκε. Με αυτό τον τρόπο γίνεται επιλεκτική αποδοχή ορισμένων οπτικών πακέτων και όχι οποιουδήποτε. Π.χ στη συγκεκριμένη εκτέλεση στάλθηκε πακέτο

με μήκος κύματος 1570nm και επειδή η τιμή αυτή ήταν η αποδεκτή, το πακέτο έφτασε επιτυχώς στη θύρα προορισμού.

Επιπλέον δίνονται τα αποτελέσματα μέσα από το ofp.log file από την αποστολή ενός οπτικού πακέτου από τη μία θύρα του μεταγωγέα σε μία άλλη, επισημαίνοντας τα βασικά στοιχεία, όπως είναι το μήκος κύματος, τα wildcards με κόκκινο χρώμα.

02:21:42.721 root : INFO : ++++++++ Wed Dec 10 02:21:42 2014 ++++++++
02:21:42.740 scapy.runtime: WARNING : No route found for IPv6 destination :: (no default route?)
02:21:42.776 root : INFO : Importing platform: eth
02:21:42.777 root : DEBUG : Configuration: {'profile': False, 'default_timeout': 2.0, 'platform_args': None, 'xunit_dir': 'xunit', 'disable_ipv6': False, 'random_seed': None, 'interfaces': [], 'port_map': {1: 'veth1', 2: 'veth3', 3: 'veth5', 4: 'veth7'}, 'minsize': 0, 'list_test_names': False, 'openflow_version': '1.0', 'switch_ip': None, 'controller_host': '0.0.0.0', 'default_negative_timeout': 0.01, 'allow_user': False, 'controller_port': 6653, 'test_file': None, 'profile_file': 'profile.out', 'fail_skipped': False, 'test_params': 'None', 'test_dir': '/home/mininet/oftest/tests', 'relax': False, 'list': False, 'test_spec': '', 'platform': 'eth', 'log_dir': None, 'xunit': False, 'debug': 'verbose', 'log_file': 'oft.log', 'platform_dir': '/home/mininet/oftest/platforms'}
02:21:42.777 root : INFO : OF port map: {1: 'veth1', 2: 'veth3', 3: 'veth5', 4: 'veth7'}
02:21:42.777 root : INFO : Autogen random seed: 89579135
02:21:42.777 dataplane : WARNING : Missing pypcap, VLAN tests may fail. See README for installation instructions.
02:21:42.778 root : INFO : *** TEST RUN START: Wed Dec 10 02:21:42 2014
02:21:42.779 root : INFO : ** START TEST CASE flow_stats.my_test3
02:21:42.779 controller: INFO : Create/listen at 0.0.0.0:6653
02:21:43.781 controller: INFO : 0.0.0.0:6653: Incoming connection from ('127.0.0.1', 42368)
02:21:43.782 controller: DEBUG : Msg out: version 1 class hello len 8 xid 2096014349
02:21:43.783 controller: DEBUG : Msg in: version 1 class hello len 8 xid 1382388994
02:21:43.815 root : INFO : Connected ('127.0.0.1', 42368)
02:21:43.816 controller: DEBUG : Running transaction 2182583910
02:21:43.816 controller: DEBUG : Msg out: version 1 class features_request len 8 xid 2182583910
02:21:43.818 controller: DEBUG : Msg in: version 1 class features_reply len 272 xid 2182583910
02:21:43.818 controller: DEBUG : Waiting for transaction 2182583910
02:21:43.819 controller: DEBUG : Matched expected XID 2182583910
02:21:43.820 root : INFO : Supported actions: 0xeff
02:21:43.820 root : INFO : Creation of input and output ports
02:21:43.820 root : INFO : Deleting all flows
02:21:43.821 root : INFO : 72
02:21:43.821 root : INFO : length tou flow delete
02:21:43.821 controller: DEBUG : Msg out: version 1 class flow_delete len 72 xid 2048658745

02:21:43.822 controller: DEBUG : Running transaction 3037675639
02:21:43.822 controller: DEBUG : Msg out: version 1 class barrier_request len 8 xid 3037675639
02:21:43.823 controller: DEBUG : Msg in: version 1 class barrier_reply len 8 xid 3037675639
02:21:43.824 controller: DEBUG : Waiting for transaction 3037675639
02:21:43.824 controller: DEBUG : Matched expected XID 3037675639
02:21:43.824 root : INFO : Creation of the optical packet
02:21:43.826 root : INFO : Creation of the optical flow that will be sent to the controller
02:21:43.826 root : INFO : Ingress 1 to egress 4
02:21:43.827 root : INFO : Inserting optical flow1
02:21:43.827 controller: DEBUG : Msg out: version 1 class flow_add len 80 xid 1660152416
02:21:43.828 controller: DEBUG : Running transaction 3715542277
02:21:43.828 controller: DEBUG : Msg out: version 1 class barrier_request len 8 xid 3715542277
02:21:43.829 controller: DEBUG : Msg in: version 1 class barrier_reply len 8 xid 3715542277
02:21:43.829 controller: DEBUG : Waiting for transaction 3715542277
02:21:43.829 controller: DEBUG : Matched expected XID 3715542277
02:21:43.830 root : INFO : Sending 1 optical pkt1s
02:21:43.830 root : INFO : Sending packet to dp port 1, expecting output on 4
02:21:43.830 dataplane : DEBUG : Sending 60 bytes to port 1
02:21:43.831 dataplane : DEBUG : Pkt len 60 in on port 4
02:21:43.831 dataplane : DEBUG : Grabbing packet
02:21:43.832 dataplane : DEBUG : Checking packet from port 4
02:21:43.832 root : DEBUG : Packet len 60 in on 4
02:21:43.833 root : INFO : Definition of the required wildcards for matching the optical packet
02:21:43.833 root : INFO : Required wildcards = 4194305
02:21:43.833 root : INFO : The test checks if the packet matches the required wildcards
02:21:43.833 root : INFO : Successful match, now the test is checking the value of the matched field
02:21:43.833 root : INFO : The value of wavelength is 1570
02:21:43.834 root : INFO : Verifying 1 optical packets
02:21:43.834 root : INFO : Sending stats request
02:21:43.834 controller: DEBUG : Running transaction 2271272874
02:21:43.834 root : INFO : 56
02:21:43.834 controller: DEBUG : Msg out: version 1 class flow_stats_request len 56 xid 2271272874
02:21:43.835 controller: DEBUG : Msg in: version 1 class flow_stats_reply len 108 xid 2271272874
02:21:43.836 controller: DEBUG : Waiting for transaction 2271272874
02:21:43.836 controller: DEBUG : Matched expected XID 2271272874
02:21:43.836 root : INFO : Received 1 packets
02:21:43.838 controller: INFO : Exiting controller thread
02:21:43.838 controller: INFO : Ignoring switch soc shutdown error
02:21:43.838 controller: INFO : Ignoring listen soc shutdown error

```
02:21:43.839 controller: INFO : Ignoring listen soc shutdown error
02:21:43.839 root : INFO : ** END TEST CASE flow_stats.my_test3
02:21:43.848 root : INFO : *** TEST RUN END : Wed Dec 10 02:21:43 2014
02:21:43.849 dataplane : INFO : Thread exit
```

- ***Class my_test4***

Το επόμενο test που πραγματοποιήθηκε έχει ως στόχο τη δημιουργία οπτικών πακέτων και μεταγωγής αυτών στην κατάλληλη θύρα εξόδου ανάλογα με την τιμή του μήκους κύματος. Πιο συγκεκριμένα, αν η τιμή του μήκους κύματος είναι 1565, τοτε το πακέτο θα θα δρομολογηθεί από τη θύρα εισόδου 1 (φυσική θύρα 1) στη θύρα εξόδου 1 (φυσική θύρα 3) με βάση τα χαρακτηριστικά του flow 1 που έχει εγγραφεί στον πίνακα μεταγωγής του switch. Αντίθετα αν η τιμή του είναι 1560, τότε το πακέτο θα περάσει από τη θύρα εισόδου 1 και θα δρομολογηθεί από την έξοδο 2 (φυσική θύρα 4). Τέλος αν το μήκος κύματος δεν έχει καμία από τις παραπάνω τιμές τότε το πακέτο απορρίπτεται. Και σε αυτό το τεστ παρουσιάζεται το oft.log αρχείο με όλα τα σημαντικά μηνύματα και βήματα της εκτέλεσης του εν λόγω τεστ.

```
02:44:51.407 root : INFO : ++++++++ Wed Dec 10 02:44:51 2014 ++++++++
02:44:51.430 scapy.runtime: WARNING : No route found for IPv6 destination :: (no
default route?)
02:44:51.462 root : INFO : Importing platform: eth
02:44:51.462 root : DEBUG : Configuration: {'profile': False, 'default_timeout': 2.0,
'platform_args': None, 'xunit_dir': 'xunit', 'disable_ipv6': False, 'random_seed': None,
'interfaces': [], 'port_map': {1: 'veth1', 2: 'veth3', 3: 'veth5', 4: 'veth7'}, 'minsize': 0,
'list_test_names': False, 'openflow_version': '1.0', 'switch_ip': None, 'controller_host':
'0.0.0.0', 'default_negative_timeout': 0.01, 'allow_user': False, 'controller_port': 6653,
'test_file': None, 'profile_file': 'profile.out', 'fail_skipped': False, 'test_params': 'None',
'test_dir': '/home/mininet/oftest/tests', 'relax': False, 'list': False, 'test_spec': '',
'platform': 'eth', 'log_dir': None, 'xunit': False, 'debug': 'verbose', 'log_file': 'oft.log',
'platform_dir': '/home/mininet/oftest/platforms'}
02:44:51.462 root : INFO : OF port map: {1: 'veth1', 2: 'veth3', 3: 'veth5', 4: 'veth7'}
02:44:51.462 root : INFO : Autogen random seed: 45897107
02:44:51.463 dataplane : WARNING : Missing pypcap, VLAN tests may fail. See
README for installation instructions.
02:44:51.464 root : INFO : *** TEST RUN START: Wed Dec 10 02:44:51 2014
02:44:51.464 root : INFO : ** START TEST CASE flow_stats.my_test4
02:44:51.464 controller: INFO : Create/listen at 0.0.0.0:6653
02:44:52.467 controller: INFO : 0.0.0.0:6653: Incoming connection from ('127.0.0.1',
43832)
02:44:52.468 controller: DEBUG : Msg out: version 1 class hello len 8 xid
3810323583
02:44:52.469 controller: DEBUG : Msg in: version 1 class hello len 8 xid
3009380042
```

02:44:52.501 root : INFO : Connected ('127.0.0.1', 43832)
02:44:52.502 controller: DEBUG : Running transaction 3826937963
02:44:52.502 controller: DEBUG : Msg out: version 1 class features_request len 8 xid 3826937963
02:44:52.503 controller: DEBUG : Msg in: version 1 class features_reply len 272 xid 3826937963
02:44:52.503 controller: DEBUG : Waiting for transaction 3826937963
02:44:52.504 controller: DEBUG : Matched expected XID 3826937963
02:44:52.505 root : INFO : Supported actions: 0xeff
02:44:52.506 root : INFO : Deleting all flows
02:44:52.506 root : INFO : 72
02:44:52.506 root : INFO : length tou flow delete
02:44:52.506 controller: DEBUG : Msg out: version 1 class flow_delete len 72 xid 3340447043
02:44:52.507 controller: DEBUG : Running transaction 3802920036
02:44:52.508 controller: DEBUG : Msg out: version 1 class barrier_request len 8 xid 3802920036
02:44:52.508 controller: DEBUG : Msg in: version 1 class barrier_reply len 8 xid 3802920036
02:44:52.508 controller: DEBUG : Waiting for transaction 3802920036
02:44:52.509 controller: DEBUG : Matched expected XID 3802920036
02:44:52.511 root : INFO : Ingress 1 to egress 4
02:44:52.511 root : INFO : Inserting optical flow2
02:44:52.511 controller: DEBUG : Msg out: version 1 class flow_add len 80 xid 3204511015
02:44:52.512 controller: DEBUG : Running transaction 2467677630
02:44:52.512 controller: DEBUG : Msg out: version 1 class barrier_request len 8 xid 2467677630
02:44:52.513 controller: DEBUG : Msg in: version 1 class barrier_reply len 8 xid 2467677630
02:44:52.513 controller: DEBUG : Waiting for transaction 2467677630
02:44:52.513 controller: DEBUG : Matched expected XID 2467677630
02:44:52.514 root : INFO : Sending 1 optical pkt2s
02:44:52.514 root : INFO : Sending packet to dp port 1, expecting output on 4
02:44:52.515 dataplane : DEBUG : Sending 60 bytes to port 1
02:44:52.515 dataplane : DEBUG : Pkt len 60 in on port 4
02:44:52.515 dataplane : DEBUG : Grabbing packet
02:44:52.516 dataplane : DEBUG : Checking packet from port 4
02:44:52.516 root : DEBUG : Packet len 60 in on 4
02:44:52.517 root : INFO : The required wildcards= 4194305
02:44:52.517 root : INFO : Wildcards of successful match= 4194305
02:44:52.517 root : INFO : Matched value of wavelength= 1560
02:44:52.517 root : INFO : Verifying 1 optical pkt2s
02:44:52.517 root : INFO : Sending stats request
02:44:52.517 controller: DEBUG : Running transaction 581659964
02:44:52.518 root : INFO : 56
02:44:52.518 controller: DEBUG : Msg out: version 1 class flow_stats_request len 56 xid 581659964
02:44:52.519 controller: DEBUG : Msg in: version 1 class flow_stats_reply len 108 xid 581659964

02:44:52.519 controller: DEBUG : Waiting for transaction 581659964
02:44:52.520 controller: DEBUG : Matched expected XID 581659964
02:44:52.520 root : INFO : Received 1 packets
02:44:52.521 controller: INFO : Exiting controller thread
02:44:52.521 controller: INFO : Ignoring switch soc shutdown error
02:44:52.521 controller: INFO : Ignoring listen soc shutdown error
02:44:52.522 controller: INFO : Ignoring listen soc shutdown error
02:44:52.522 root : INFO : ** END TEST CASE flow_stats.my_test4
02:44:52.524 root : INFO : *** TEST RUN END : Wed Dec 10 02:44:52 2014
02:44:52.524 dataplane : INFO : Thread exit

5. Ενσωμάτωση του μήκους κύματος στον OpenFlow 1.3 Software Switch

5.1. Βασικά χαρακτηριστικά του Openflow 1.3 Software Switch

Ο συγκεκριμένος μεταγωγέας που είναι συμβατός με την έκδοση του πρωτοκόλλου 1.3, αποτελεί μία user-space software switch implementation. Ο συγκεκριμένος μεταγωγέας βασίστηκε σε μία προηγούμενη έκδοση, την Ericsson TrafficLab 1.1 softswitch implementation, και δεχόμενος τις κατάλληλες αλλαγές στο επίπεδο προώθησης, κατέστη δυνατή η υποστήριξη από αυτόν του Openflow 1.3.

Το συγκεκριμένο κεφάλαιο έχει ως στόχο την ενσωμάτωση του μήκους κύματος μέσα στο flow table του μεταγωγέα και στα flow entries. Με τη βοήθεια του εργαλείου dpctl πραγματοποιείται εγκατάσταση και απεικόνιση των flows του μεταγωγέα στο περιβάλλον του Mininet.

Το dpctl είναι ένα χρήσιμο εργαλείο για την αλλαγή και την παρουσίαση της κατάστασης του δικτύου με ένα γρήγορο τρόπο, αποφεύγοντας αλλαγές στον controller, όταν στόχος είναι μόνο κάποιες ακριβείς ενέργειες.

5.2. Εκτέλεση βασικών εντολών με τη χρήση του εργαλείου dpctl έχοντας ενσωματώσει το μήκος κύματος

Σε πρώτο στάδιο θα ανοίξουμε την τοπολογία στο Mininet εκτελώντας την ακόλουθη εντολή:

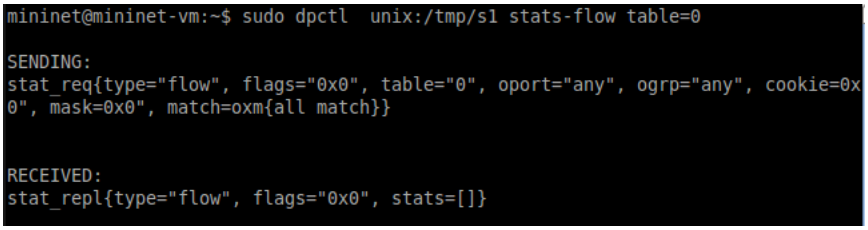
$ sudo mn --topo single,2 --mac --switch user --controller remote

Δημιουργεί μία τοπολογία με ένα switch (s1) και δύο hosts (h1 και h2). Όπως φαίνεται και από την εντολή, ο μεταγωγέας που χρησιμοποποιείται είναι user – space switch και ως ελεγκτής έχει οριστεί κάποιος remote.

Στη συνέχεια γίνεται εκτέλεση της ακόλουθης εντολής χάρη στην οποία επιστρέφονται τα περιεχόμενα του flow table.

$ sudo dpctl unix:/tmp/s1 stats-flow table=0

Πριν από την εισαγωγή κάποιου flow entry στον πίνακα, ο τελευταίος περιέχει τις ακόλουθες εγγραφές.



Σχήμα 36 : Απεικόνιση του flow table του υπό εξέταση μεταγωγέα

Από το σχήμα 36 γίνεται εμφανές πως με την εκτέλεση της εντολής για εμφάνιση του flow table, γίνεται αρχικά αποστολή ενός Stats Request μηνύματος έχοντας όλα τα πεδία στις προεπιλεγμένες τιμές.

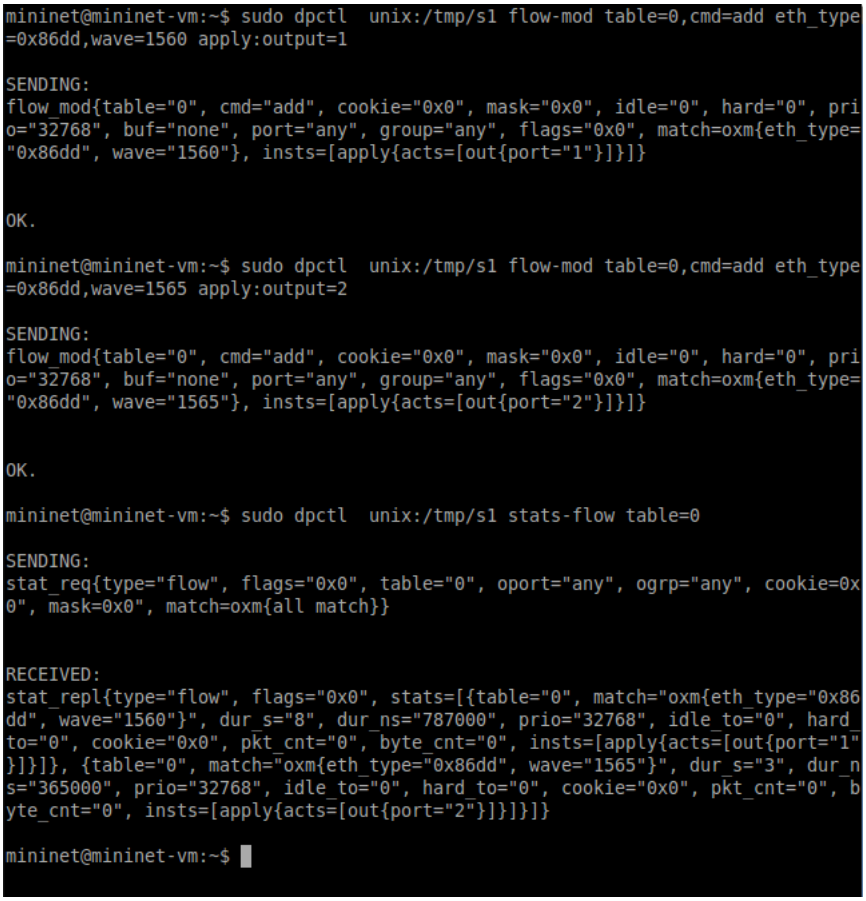
Σε επόμενο στάδιο γίνεται εγκατάσταση δύο flow εγγραφών στον πίνακα 0. Η μεν πρώτη περνάει ως παράμετρο το μήκος κύματος με τιμή 1560nm και έξοδο στην πόρτα 1 με την εκτέλεση της εντολής:

$ sudo dpctl unix:/tmp/s1 flow-mod table=0,cmd=add eth_type=0x86dd, wave=1560 apply:output=1

Η δε δεύτερη περνάει ως παράμετρο το μήκος κύματος με τιμή 1565nm και έξοδο στην πόρτα 2 με εκτέλεση της εντολής:

$ sudo dpctl unix:/tmp/s1 flow-mod table=0,cmd=add eth_type=0x86dd, wave=1565 apply:output=2

Στην εικόνα που ακολουθεί παρουσιάζουμε την εκτέλεση των δύο προαναφερθεισών εντολών, καθώς επίσης και τον πίνακα 0, όπως είναι τώρα διαφοροποιημένος έχοντας ενσωματώσει σε αυτόν τις νέες εγγραφές.



Σχήμα 37 : Εγκατάσταση ενός οπτικού flow στον flow tabletου openflow switch.

Στο σχήμα 37 παρουσιάζεται η εγκατάσταση δύο flows στο flow table. Πλέον οι παράμετροι του εκάστοτε flow δεν έχουν τις προεπιλεγμένες τιμές. Καταρχάς θα γίνει παρουσίαση των παραμέτρων και θα εξηγηθεί η χρησιμότητά τους μέσα στο flow. [19] Η παράμετρος **table** δηλώνει σε ποιον flow table θα εισαχθεί , τροποποιηθεί ή σβηστεί ένα flow entry. Η **cmd** δηλώνει την ενέργεια που θα εκτελεστεί με την εγκατάσταση του flow, που στην προκειμένη περίπτωση είναι add, δηλαδή προσθήκη πεδίων στο match field. Η παράμετρος **flags** παίρνει διάφορες τιμές ανάλογα με το τι ενέργειες θα γίνουν στον υπό εξέταση μεταγωγέα. Όταν ένα flow εγκαθίσταται μέσα στο flow table, τότε το πεδίο **flags** ορίζεται με τις τιμές που δίνονται από το μήνυμα. Επίσης να επισημανθεί ότι, όταν ένα flow κάνει match και τροποποιείται, τότε το πεδίο flags αγνοείται. Στη συνέχεια μέσα στο πεδίο **match** αναγράφονται τα πεδία των επικεφαλίδων στα οποία ένα εισερχόμενο πακέτο μπορεί να κάνει match. Επίσης τα πεδία **dur_s** και **dur_ns** αποτυπώνουν το χρόνο κατά τον οποίο ένα flow είναι εγκατεστημένο στο flow table σε seconds και nanoseconds αντίστοιχα. Το πεδίο **priority** δηλώνει το επίπεδο προτεραιότητας που έχει το εν λόγω flow. Στη συνέχεια δύο άλλες παράμετροι του flow table είναι αυτές που αφορούν το μηχανισμό λήξης των εγκατεστημένων flows. Κάθε flow entry έχει ένα **idle_timeout** και ένα **hard_timeout**. Αν οι

τιμές και των δύο είναι μη-μηδενικές, τότε ο μεταγωγέας πρέπει να γνωστοποιήσει την ώρα άφιξης του συγκεκριμένου flow, καθώς ίσως χρειαστεί να απομακρύνει το flow αργότερα. Το πεδίο **cookie** αποτελεί μία αδιαφανή τιμή δεδομένων που επιλέγεται από τον ελεγκτή. Αυτή η τιμή εμφανίζεται στα μηνύματα αντικατάστασης των flow και στα μηνύματα εμφάνισης των στατιστικών. Επίσης μπορεί να χρησιμοποιηθεί, για να φιλτράρει τα στατιστικά, τη μετατροπή των flows, καθώς και την απομάκρυνσή τους. Να επισημανθεί ότι δε χρησιμοποιείται από διαδικασία επεξεργασίας του πακέτου και για αυτό το λόγο δεν είναι αναγκαίο να βρίσκεται και στο hardware που χρησιμοποιείται. Είναι καλό να αναφερθεί επίσης ότι, όταν ένα flow εισέρχεται σε ένα flow table, τότε στο πεδίο cookie του παρέχεται μία τιμή. Ωστόσο, στην περίπτωση μετατροπής ενός ήδη υπάρχοντος flow, η τιμή του cookie παραμένει αναλλοίωτη. Ένα επιπλέον πεδίο μέσα σε ένα flow είναι οι counters. Αυτοί υφίστανται μέσα σε κάθε flow table, flow entry, θύρα, group, ουρά ή μετρικό που έχουν εγκατασταθεί μέσα στο μεταγωγέα. Ένας μεταγωγέας δεν είναι απαραίτητο να υποστηρίζει όλους τους μετρητές, παρά μόνο όσους ορίζονται ως "*Required".* Στην προκειμένη περίπτωση έχούν οριστεί μόνο οι μετρητές πακέτων και bytes.

Στη συνέχεια εκτελείται ένα διαφορετικό σετ εντολών που αρχικά έχει ως στόχο τη δημιουργία ενός group και μετέπειτα τη δημιουργία ενός flow και αποστολής αυτού στο group που μόλις δημιουργήθηκε.

```
mininet@mininet-vm:~$ sudo dpctl  unix:/tmp/s1 group-mod cmd=add,group=1,type=all

SENDING:
grp_mod{group="1", cmd="add", type="all", buckets=[]}


RECEIVED:
error{type="GROUP_MOD_FAILED", code="GROUP_EXISTS", dlen="16"}

mininet@mininet-vm:~$ sudo dpctl  unix:/tmp/s1 group-mod cmd=add,group=1,type=all

SENDING:
grp_mod{group="1", cmd="add", type="all", buckets=[]}


OK.

mininet@mininet-vm:~$ sudo dpctl unix:/tmp/s1 flow-mod table=0,cmd=add wave=1560 apply:group=1

SENDING:
flow_mod{table="0", cmd="add", cookie="0x0", mask="0x0", idle="0", hard="0", prio="32768", buf="none",
port="any", group="any", flags="0x0", match=oxm{wave="1560"}, insts=[apply{acts=[group{id="1"}]}]}


OK.

mininet@mininet-vm:~$
```

Σχήμα 38 : Δημιουργία ενός group και πέρασμα του μήκους κύματος σε αυτό

Στο συγκεκριμένο σημείο θα παρουσιαστεί η δομή ενός group table. Αποτελείται από group entries. Η δυνατότητα ενός flow entry να δείχνει σε ένα group επιτρέπει στο OpenFlow να αναπαραστήσει πρόσθετες μεθόδους προώθησης.

Group Identifier	Group Type	Counters	Action Buckets

Πίνακας 10: Βασικά στοιχεία ενός group entry μέσα σε ένα group table

Κάθε group entry αναγνωρίζεται από το αναγνωριστικό του και περιέχει:

- Το **group identifier** : Έναν 32-bit μη προσημασμένο ακέραιο μοναδικά οριζόμενο για το group.
- Το **group type** : Για τον καθορισμό των semantics του group.
- Τους **counters** : Διαφοροποιούνται, όταν τα πακέτα επεξεργάζονται από ένα group.
- Τα **action buckets** : Μία ταξινομημένη λίστα από action buckets, όπου κάθε ενέργεια περιέχει ένα σύνολο ενεργειών για εκτέλεση και σχετικές παραμέτρους.

Ένα διαφορετικό σύνολο εντολών που μπορεί να εκτελεστεί είναι η εγκατάσταση ενός flow με προεπιλεγμένες παραμέτρους που να κάνουν match στη θύρα εισόδου in_port και στο eth_type και να ορίζουν μία ενέργεια set_field για επανεγγραφή της διεύθυνσης ip_src.

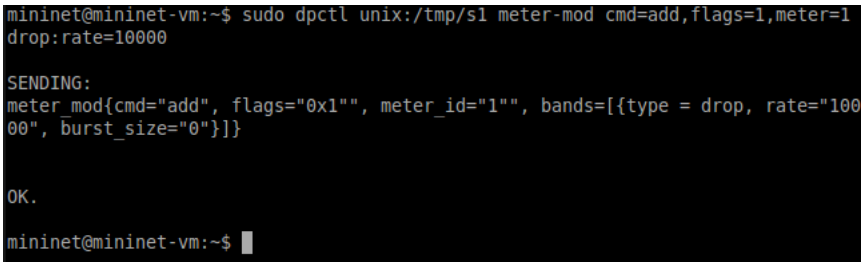
Το field περιέχει ένα περιγραφόμενο πεδίο επικεφαλίδας χρησιμοποιώντας μία μοναδική OXM TLV δομή και μετατροπή του αντίστοιχου πεδίου επικεφαλίδας στο πακέτο με την τιμή του oxm_value, το ωφέλιμο φορτίο του OXM TLV. Το match του flow entry θα πρέπει να περιέχει τα προαπαιτούμενα του OXM, που είναι σχετικά με το πεδίο που έχει οριστεί, αλλιώς πρέπει να πυροδοτηθεί μήνυμα λάθους.

Ο τύπος μίας ενέργειας set_field μπορεί να είναι οποιοσδήποτε έγκυρος τύπος της επικεφαλίδας OXM. Η ενέργεια Set-Field κάνει επανεγγραφή του πεδίου επικεφαλίδας που ορίζεται από το OXM type και εκτελεί τον κατάλληλο CRC επαναϋπολογισμό που βασίζεται στο πεδίο της επικεφαλίδας. Στο σχήμα 39 φαίνεται η τροποποίηση του μήκους κύματος σε μία εγραφή του πίνακα μεταγωγής.

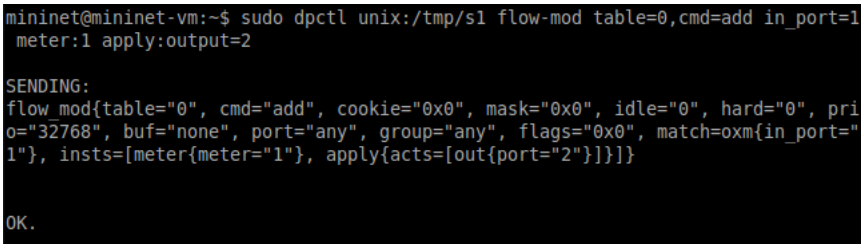
```
mininet@mininet-vm:~$ sudo dpctl  unix:/tmp/s1 flow-mod table=0,cmd=add in_port=
1,eth_type=0x800 apply:set_field=wave:1560

SENDING:
flow_mod{table="0", cmd="add", cookie="0x0", mask="0x0", idle="0", hard="0", pri
o="32768", buf="none", port="any", group="any", flags="0x0", match=oxm{in_port="
1", eth_type="0x800"}, insts=[apply{acts=[set_field{field:wave="1560"}]}]}


OK.

mininet@mininet-vm:~$
```

Σχήμα 39: Επανεγγραφή της τιμής του μήκους κύματος στο flow table

Ένα άλλο παράδειγμα που θα μπορούσε εκτελεστεί είναι η δημιουργία ενός μετρικού (meter) με 10MB όριο στο ρυθμό μετάδοσης. Το μετρικό αποτελεί ένα στοιχείο του μεταγωγέα που μπορεί να υπολογίσει και να ελέγξει το ρυθμό των πακέτων. Πυροδοτεί μία meter band σε περίπτωση που ένα πακέτο ή byte περάσει μέσα από το μετρικό με ρυθμό μεγαλύτερο από ένα προκαθορισμένο κατώφλι. Αν το meter band απορρίπτει το πακέτο, τότε ονομάζεται Rate Limiter (Περιοριστής Ρυθμού Μετάδοσης).



Σχήμα 40: Δημιουργία ενός μετρικού με περιοριστή ρυθμού μετάδοσης στα 10MB



Σχήμα 41: Δημιουργία ενός flow που δείχνει σε ένα μετρικό

Ένα Meter Table αποτελείται από meter entries ορίζοντας μετρικά για κάθε flow. Τα μετρικά σε κάθε flow δίνουν τη δυνατότητα στο OpenFlow να ενσωματώσει ποικίλες απλές λειτουργίες QoS (Quality of Service), όπως είναι το rate-limiting, και μπορεί να συνδυαστεί με ουρές που βρίσκονται σε κάθε θύρα για τηνενσωμάτωση ποικίλων QoS frameworks, όπως είναι το DiffServ.

Ένα μετρικό υπολογίζει το ρυθμό των πακέτων που αντιστοιχούνται σε αυτό και επιτρέπει τον έλεγχο του ρυθμού αυτών των πακέτων. Τα μετρικά προσκολλώνται άμεσα με τα flow entries (σε αντίθεση με τις ουρές που προσκολλώνται στις θύρες). Οποιοδήποτε flow entry μπορεί να καθορίσει ένα μετρικό σε κάθε σύνολο εντολών (instruction set). Το μετρικό υπολογίζει και ελέγχει το ρυθμό του συνόλου όλων των flow entries στα οποία αντιστοιχεί. Πολλαπλά μετρικά μπορούν να χρησιμοποιηθούν στον ίδιο πίνακα αλλά με έναν αποκλειστικό τρόπο. Τα πολλαπλά μετρικά μπορούν να χρησιμοποιηθούν στο ίδιο σύνολο πακέτων χρησιμοποιώντας τα σε διαδοχικούς flow tables.

Meter Identifier	Meter Bands	Counters

Πίνακας 11: Κύρια στοιχεία ενός meter entry στο meter table

Κάθε meter entry αναγνωρίζεται από το αναγνωριστικό του και περιέχει :

- Το **meter identifier** : Ένα 32-bit μη-προσημασμένο ακέραιο, που ορίζει μοναδικά το μετρικό.
- Τις **meter bands** : Μία μη-ταξινομημένη λίστα των meter-bands, όπου κάθε meter band καθορίζει το ρυθμό του band και τον τρόπο που θα πρέπει να γίνει η επεξεργασία του πακέτου.
- Οι **counters** : Ανανεώνονται, όταν τα πακέτα επεξεργάζονται από ένα μετρικό

Να επισημανθεί επίσης ότι κάθε meter μπορεί να έχει ένα ή περισσότερα meter bands. Κάθε band καθορίζει το ρυθμό που καθορίζει το band και τον τρόπο που θα γίνει η επεξεργασία των πακέτων.

5.3. Παρουσίαση αλλαγών για την ενσωμάτωση των οπτικών ιδιοτήτων στο OpenFlow 1.3 Software Switch

Για την ενσωμάτωση των οπτικών ιδιοτήτων, όπως το μήκος κύματος, στον υπο εξέταση switch έλαβαν χώρο ορισμένες αλλαγές στα αρχεία του φακέλου ofsoftswitch13. Αυτές θα παρουσιαστούν με λεπτομέρεια στη συνέχεια.

Φάκελος **include/openflow:**

- **openflow.h :** Στη γραμμή 331 στο enumeration **oxm_ofb_match_fields,** όπου ορίζονται τα πεδία που θα γίνει το match, προστέθηκε η εντολή:

```
OFPXMT_OFB_WAVE = 40          /* Wavelength */
```

Φάκελος **lib:**

- **dhcp-client.c :** Στη μέθοδο **do_send_msg,** στη γραμμή 988, προστέθηκε η εντολή **struct optical_header oh;** για τον ορισμό μιας δομής της οπτικής επικεφαλίδας. Επιπλέον στη γραμμή 1036, ορίζουμε την εντολή **oh.wave = htons(68);,** όπου το htons είναι μία συνάρτηση που επιστρέφει την τιμή σε TCP/IP δικτυακή δυαδική σειρά.

- **flow.c :** Στη σειρά 107, ορίζουμε ένα συγκεκριμένο block, το οποίο χρησιμοποιείται για την τοποθέτηση της οπτικής επικέφαλίδας στο πακέτο.

```
static struct optical_header *
pull_wave(struct ofpbuf *packet)
{
        return ofpbuf_try_pull(packet, OPTICAL_HEADER_LEN);
}
```

Επιπλέον στη σειρά 172, έγινε προσθήκη ενός if-block που ελέγχει, αν έχει οριστεί μέσα στο flow, το μήκος κύματος. Σε περίπτωση που έχει συμβεί αυτό, τότε προστίθεται στο μήκος κύματος στο flow.

```
/* Check for wavelength*/

    if (flow->optic == OPTICAL_TYPE) {
      const struct optical_header *optical = pull_wave(&b);
      if (optical) {
            flow->wave = optical->wave;
            packet->l2 = b.data;
      }
      else  {
        return flow->optic = 0;
      }
    }
    else {
        return retval;
    }
```

Τέλος μέσα στη **flow_print** συνάρτηση προστίθεται και η επιλογή για απεικόνιση του μήκους κύματος.

```
ntohs(flow->wave)
```

- **flow.h :** Μέσα στη δομή που ορίζεται το flow έχουμε ορίσει το μήκος κύματος με μήκος δύο bytes (γραμμή 57).

```
uint16_t wave;          /* Wavelength */
```

Επιπλέον στη γραμμή 63 προστέθηκε η εντολή :

`uint16_t optic;  /* for optical layer */` που στην ουσία ορίζεται, για να δηλώσει το optical layer.

- **packets.h :** Ορισμός της οπτικής επικεφαλίδας στη σειρά 407.

```
#define OPTICAL_TYPE 400


#define OPTICAL_HEADER_LEN 4
struct optical_header {
   uint16_t wave;
   uint16_t optical_csum;
};
BUILD_ASSERT_DECL(OPTICAL_HEADER_LEN == sizeof(struct optical_header));
```

Στη δομή protocols_std, στη σειρά 430, έχει οριστεί ο δείκτης optical, που δείχνει στην οπτική επικεφαλίδα.

```
struct optical_header  * optical;
```

Τέλος στη μέθοδο protocol_reset κάνουμε επαναφορά όλων των πρωτοκόλλων μαζί και του οπτικού.

```
proto->optical     = NULL;
```

Φάκελος **nbee_link:**

- **nbee_link.cpp :** Στη σειρά 566, προσθήση του if – block κώδικα:

```
if (protocol_Name.compare("optical") == 0 && pkt_proto->optical == NULL)
        {
           pkt_proto->optical = (struct optical_header *) ((uint8_t*) pktin->data + proto->Position);
           PDMLReader->GetPDMLField(proto->Name, (char*) "sport", proto->FirstField, &field);
           nblink_extract_proto_fields(pktin, field, pktout, OXM_OF_WAVE);
        }
```

Φάκελος **oflib:**

- **ofl-print.c:** Ορισμός στη γραμμή 271 του μήκους κύματος για εκτύπωση του ονόματος wave σε περίπτωση που επιλεχθεί.

```
case OXM_OF_WAVE:          {fprintf(stream, "wave"); return;}
```

- **ofl-structs-print.c :** Προσθήκη στη συνάρτηση **ofl_structs_oxm_tlv_print** της περίπτωσης εισαγωγής του μήκους κύματος (σειρά 426)

```
case OFPXMT_OFB_WAVE:
        fprintf(stream, "wave=\"%d\"", *((uint16_t*) f->value));
        break;
```

- **oxm-match.c :** Προσθήκη μίας ακόμα περίπτωσης στη συνάρτηση **parse_oxm_entry** για υποστήριξη της οπτικής επικεφαλίδας (γραμμή 395).

```
/* Optical header */

    case OFI_OXM_OF_WAVE:

        ofl_structs_match_put16(match, f->header, ntohs(*((uint16_t*) value)));

      return 0;
```

- **oxm_match.def:**

  Ορισμός του πεδίου για την υποστήριξη του μήκους κύματος:

```
DEFINE_FIELD  (OF_WAVE,       OXM_DL_NONE,  0,        false)
```

- **oxm_match.h**

  Ορισμός του μήκους κύματος στο **OXM_HEADER**

```
#define OXM_OF_WAVE OXM_HEADER (0x8000,40,2)
```

Φάκελος **udatapath:**

- **dp_actions.c:**

  Ορισμός μέσα στη συνάρτηση **set_field** της case OXM_OF_WAVE:

```
case OXM_OF_WAVE:{
    struct optical_header *optical = pkt->handle_std->proto->optical;
    uint16_t v = htons(*(uint16_t*) act->field->value);
    optical->optical_csum = recalc_csum16(optical->optical_csum, optical->wave,
v);
    memcpy(&optical->wave, &v, OXM_LENGTH(act->field->header));
    break;
       }
```

- **dp_capabilities.h:**

  Στον ορισμό για τα υποστηριζόμενα πεδία, στα οποία γίνεται το match, προστέθηκε και το μήκος κύματος, με την ακόλουθη προσθήκη στη γραμμή 103.

```
| OFPXMT_OFB_WAVE                    )
```

- **flow_table.c:**

  Στα αναγνωριστικά για το match προσθέτουμε αυτό του μήκους κύματος στη λίστα oxm_ids: **OXM_OF_WAVE.**

Φάκελος **utilities:**

- **dpctl.c:**Στη συνάρτηση **parse_match** γίνεται προσθήκη ενός if-block κώδικα για την υποστήριξη στο match της οπτικής ιδιότητας, δηλαδή του μήκους κύματος.

```
/* OPTICAL */
    if (strncmp(token, MATCH_WAVE KEY_VAL, strlen(MATCH_WAVE KEY_VAL))
== 0) {
        uint16_t wave;
        if (parse16(token + strlen(MATCH_WAVE KEY_VAL), NULL, 0, 0xffff, &wave))
{
           ofp_fatal(0, "Error parsing wave %s.", token);
        }
        else ofl_structs_match_put16(m, OXM_OF_WAVE,wave);
        continue;
```

```
}
```

Επιπλέον μέσα στη συνάρτηση **parse_set_field,** η οποία έχει ως στόχο να επαναγράψει μία εντολή σε ένα flow, προστίθεται ένα if-block, που να επιτελεί αυτή τη λειτουργία για το μήκος κύματος.

- **dpctl.h:** Ορισμός του μήκους κύματος

```
#define MATCH_WAVE          "wave"
```

6. Σύνοψη και μελλοντική έρευνα

6.1. Σύνοψη

Η παρούσα διπλωματική εργασία αφορά στην μελέτη των σύγχρονων οπτικών δικτύων τα οποία είναι ευέλικτα και προγραμματιζόμενα ώστε να υποστηρίζουν την μεγάλη αύξηση της διαδικτυακής κίνησης. Η αρχιτεκτονική των SDN (Software Defined Networks) δικτύων έχει τεράστιο όφελος για την διαχείριση ροών δεδομένων μεγάλου όγκου και για την αξιοποίηση του διαθέσιμου εύρους ζώνης με τον βέλτιστο τρόπο και με την μικρότερη δυνατή ανθρώπινη παρέμβαση. Η έννοια του software defined networking έχει ήδη διεισδύσει στα οπτικά δίκτυα δεδομένης της ανάγκης για ευέλικτη μετάδοση των δεδομένων για τα οποία μπορεί να μεταβάλλεται ο τύπος διαμόρφωσης, ο ρυθμός μετάδοσης, το μήκος κύματος και ο αριθμός των φερόντων καναλιών για κάθε ζεύξη και να ελέγχονται από μία κεντρική τοποθεσία. Το πρωτόκολλο Openflow το οποίο λειτουργεί ως διεπαφή μεταξύ των δομικών στοιχείων του δικτύου και του επιπέδου ελέγχου δεν υποστηρίζει μετάδοση οπτικών σημάτων σήμερα. Η διπλωματική εκπονήθηκε στο παραπάνω πλαίσιο και συνεισέφερε με την ανάπτυξη κωδίκων για την επίτευξη των ακόλουθων αντικείμενων:

- Υλοποίηση πλατφόρμας προσομοίωσης δικτύου στο εικονικό περιβάλλον Mininet με μεταγωγείς και ελεγκτές που υποστηρίζουν το Openflow πρωτόκολλο και εκτέλεση απλών τεστ ανταλλαγής πακέτων μηνυμάτων σε μια δικτυακή τοπολογία.
- Τροποποίηση του πρωτοκόλλου Openflow ώστε να υποστηρίζει μεταγωγή οπτικών πακέτων.
- Δημιουργία οπτικών πακέτων και ενσωμάτωση του μήκους κύματος στην επικεφαλίδα.
- Προσθήκη του μήκους κύματος στις εγγραφές του πίνακα μεταγωγής και τροποποίηση πεδίων εγγραφών.
- Εκτέλεση απλών τεστ μεταγωγής οπτικών πακέτων με τη βοήθεια του εργαλείου OFTest που χρησιμοποιείται για την αξιολόγηση της απόδοσης λειτουργίας ενός μεταγωγέα (SUT).

6.2. Μελλοντική έρευνα

Τα αποτελέσματα της διπλωματικής εργασίας αποτελούν μια πρώτη προσέγγιση για την υλοποίηση SDN ευέλικτων οπτικών δικτύων. Το επόμενο βήμα είναι η μεταφορά των απλών τεστ προσομοίωσης που εκτελέστηκαν στο εικονικό περιβάλλον του Mininet με τα

διαθέσιμα εργαλεία όπως το OFTest στο εργαστήριο και η ενσωμάτωση περισσότερων οπτικών ιδιοτήτων, όπως είναι η οπτική ισχύς, το οπτικό πλέγμα. Σε ένα τέτοιο πείραμα θα μπορεί να επιτευχθεί έλεγχος των παραμέτρων λειτουργίας οπτικών στοιχείων, όπως είναι οι πομποδέκτες μεταβλητού μήκους κύματος και τύπου διαμόρφωσης και οι ευέλικτοι WSS μεταγωγείς από ένα κεντρικό λογισμικό. Για μια τέτοια υλοποίηση θα πρέπει να δημιουργηθεί επιπλέον η κατάλληλη διεπαφή του οπτικού στοιχείου με την πλατφόρμα λογισμικού, η οποία θα είναι υπεύθυνη για τη «μετάφραση» των μηνυμάτων OpenFlow στο hardware. Συνήθως αυτή η διεπαφή είναι ένας απλός κώδικας σε labview. Επιπλέον, για την ολοκλήρωση της πλατφόρμας χρειάζεται να γίνει τροποποίηση των OpenFlow μηνυμάτων που αφορούν λειτουργίες του ελεγκτή ώστε να συμπεριληφθούν οι παράμετροι για οπτική μετάδοση. Τέλος, η ολοκλήρωση της υλοποίησης της παραπάνω πλατφόρμας ανοίγει το δρόμο για τη δημιουργία μιας πληθώρας εφαρμογών δικτύου (network APIs), οι οποίες θα αναφέρονται για παράδειγμα στον αλγόριθμο δρομολόγησης, ή στην μεταβολή των παραμέτρων μετάδοσης ανάλογα με τους διαθέσιμους πόρους του δικτύου κ.ά.

Βιβλιογραφία

[1] J. Zhang, Y. Zhao, H. Yang, Y. Ji, H. Li, Y. Lin, G. Li, J. Han, Y. Lee και T. Ma, «First Demonstration of enhanced Software Defined Networking (eSDN) over elastic Grid (eGrid) Optical Networks for Data Center Service Migration,» 2013.

[2] S. Azodolmolky, Software Defined Networking with OpenFlow, Packt Publishing, 2013.

[3] M. Rouse, «http://searchcloudcomputing.techtarget.com/definition/cloud-computing».

[4] «http://en.wikipedia.org/wiki/Data_center».

[5] «OpenFlow Switch Specification, Version 1.4.0 (Wire Protocol 0x05),» The Open Networking Foundation, August 5, 2013.

[6] «http://etherealmind.com/northbound-api-southbound-api-eastnorth-lan-navigation-in-an-openflow-world-and-an-sdn-compass/».

[7] «https://www.sdncentral.com/resources/sdn/southbound-interface-api/».

[8] «https://www.sdncentral.com/resources/sdn/north-bound-interfaces-api/».

[9] T. Ren και Y. Xu, «Analysis of the New Features of OpenFlow 1.4,» Atlantis Press, 2014.

[10] J. Kurose και K. Ross, Computer Networking : A Top-Down Approach, Pearson Education, 2013.

[11] S. Das, «Extensions to the OpenFlow Protocol in support of Circuit Switching,» June, 2010.

[12] «http://vmware-player.joydownload.com,» VMware Inc.

[13] B. Lantz και B. O'Connor, «http://mininet.org/».

[14] «https://github.com/mininet/mininet/wiki/Introduction-to-Mininet».

[15] «https://www.wireshark.org/about.html,» 1998.

[16] A. Vidal, E. L. Fernandes, C. E. Rothenberg και M. R. Salvador, «http://opennetworkingfoundation.github.io/libfluid/index.html,» CPqD, 2013.

[17] «http://archive.openflow.org/wk/index.php/OFTestTutorial».

[18] «A Performance Comparison of WSS Switch Engine Tecnologies,» JDS Uniphase

Corporation, May 2009.

[19] «OpenFlow Switch Specification Version 1.3.0 (Wire Protocol 0x04),» 2012.

Παράρτημα Κωδίκων

<u>Κώδικες των βασικών τεστ που εκτελέστηκαν στο OFTest</u>

- **class my_test**

```
class my_test(base_tests.SimpleDataPlane):
    """
    Verify flow stats are properly retrieved.

    Generate two packets and install two matching flows
    Send some number of packets
    Send a flow stats request to match the flows and retrieve stats
    Verify that the packet counter has incremented

    TODO: add a third flow, and then configure the match to exclude
    that flow?
    """

    def buildFlowModMsg(self, pkt, ingress_port, egress_port):
        match = packet_to_flow_match(self, pkt)
        match.wildcards &= ~ofp.OFPFW_IN_PORT
        self.assertTrue(match is not None,
                        "Could not generate flow match from pkt")
        match.in_port = ingress_port

        flow_mod_msg = ofp.message.flow_add()
        flow_mod_msg.match = match
        flow_mod_msg.cookie = random.randint(0,9007199254740992)
        flow_mod_msg.buffer_id = 0xffffffff
        flow_mod_msg.idle_timeout = 0
        flow_mod_msg.hard_timeout = 0
        act = ofp.action.output()
        act.port = egress_port
        flow_mod_msg.actions.append(act)

        logging.info("Ingress " + str(ingress_port) +
                     " to egress " + str(egress_port))

        return flow_mod_msg

    def sumStatsReplyCounts(self, response):
        total_packets = 0
        for obj in response.entries:
            #self.assertEqual(match, obj.match,


#                   "Matches do not match")
            logging.info("Received " + str(obj.packet_count)
                         + " packets")
```

```
        total_packets += obj.packet_count
    return total_packets

def verifyStats(self, match, out_port, test_timeout, packet_count):
    stat_req = ofp.message.flow_stats_request()
    stat_req.match = match
    stat_req.table_id = 0xff
    stat_req.out_port = out_port

    all_packets_received = 0
    for i in range(0,test_timeout):
        logging.info("Sending stats request")
        # TODO: move REPLY_MORE handling to controller.transact?
        response, pkt = self.controller.transact(stat_req,
                                        timeout=test_timeout)
        self.assertTrue(response is not None,
                    "No response to stats request")
        total_packets = self.sumStatsReplyCounts(response)

        while response.flags == ofp.OFPSF_REPLY_MORE:
          response, pkt = self.controller.poll(exp_msg=
                                    ofp.OFPT_STATS_REPLY,
                                    timeout=test_timeout)
          total_packets += self.sumStatsReplyCounts(response)

        if total_packets == packet_count:
           all_packets_received = 1
           break
        sleep(1)

    self.assertTrue(all_packets_received,
                "Total stats packet count " + str(total_packets) +
                " does not match number sent " + str(packet_count))

def runTest(self):
    # TODO: set from command-line parameter
    test_timeout = 60

    of_ports = config["port_map"].keys()
    of_ports.sort()
    self.assertTrue(len(of_ports) >= 1, "Not enough ports for test")
    ingress_port1 = of_ports[0];
        ingress_port2 = of_ports[1];
    egress_port1 = of_ports[2];
    egress_port2 = of_ports[3];


    delete_all_flows(self.controller)

    pkt1 = simple_tcp_packet()
    flow_mod_msg1 = self.buildFlowModMsg(pkt1, ingress_port1, egress_port2)
```

```
    pkt2 = simple_tcp_packet(eth_src='0:7:7:7:7:7')
    flow_mod_msg2 = self.buildFlowModMsg(pkt2, ingress_port2, egress_port1)

    logging.info("Inserting flow1")
    self.controller.message_send(flow_mod_msg1)
    logging.info("Inserting flow2")
    self.controller.message_send(flow_mod_msg2)
    do_barrier(self.controller)

    num_pkt1s = random.randint(10,30)
    logging.info("Sending " + str(num_pkt1s) + " pkt1s")
    num_pkt2s = random.randint(10,30)
    logging.info("Sending " + str(num_pkt2s) + " pkt2s")
    for i in range(0,num_pkt1s):
       sendPacket(self, pkt1, ingress_port1, egress_port2, test_timeout)
    for i in range(0,num_pkt2s):
       sendPacket(self, pkt2, ingress_port2, egress_port1, test_timeout)

    match1 = packet_to_flow_match(self, pkt1)
    logging.info("Verifying flow1's " + str(num_pkt1s) + " packets")
    self.verifyStats(match1, ofp.OFPP_NONE, test_timeout, num_pkt1s)
    match2 = packet_to_flow_match(self, pkt2)
    logging.info("Verifying flow2's " + str(num_pkt2s) + " packets")
    self.verifyStats(match2, ofp.OFPP_NONE, test_timeout, num_pkt2s)
    match1.wildcards |= ofp.OFPFW_DL_SRC
    logging.info("Verifying combined " + str(num_pkt1s+num_pkt2s) + " packets")
    self.verifyStats(match1, ofp.OFPP_NONE, test_timeout,
                num_pkt1s+num_pkt2s)
    # TODO: sweep through the wildcards to verify matching?
```

- **class my_test2**

```
class my_test2(base_tests.SimpleDataPlane):

…
   def runTest(self):
      # TODO: set from command-line parameter
      test_timeout = 60

      of_ports = config["port_map"].keys()
      of_ports.sort()
      self.assertTrue(len(of_ports) >= 3, "Not enough ports for test")
      ingress_port1 = of_ports[0];
      ingress_port2 = of_ports[1];
      egress_port1 = of_ports[2];
          egress_port2 = of_ports[3];

      delete_all_flows(self.controller)
```

```
    pkt1 = simple_tcp_packet(eth_dst='00:06:07:08:09:0a')
    flow_mod_msg1 = self.buildFlowModMsg(pkt1, ingress_port1, egress_port1)

    pkt2 = simple_tcp_packet(eth_dst='7:7:7:7:7:7')
    flow_mod_msg2 = self.buildFlowModMsg(pkt2, ingress_port2, egress_port2)

        pkt3 = simple_tcp_packet(eth_dst='0:0:0:0:0:1')
    flow_mod_msg3 = self.buildFlowModMsg(pkt3, ingress_port1, egress_port2)

    logging.info("Inserting flow1")
    self.controller.message_send(flow_mod_msg1)
    logging.info("Inserting flow2")
    self.controller.message_send(flow_mod_msg2)
        logging.info("Inserting flow3")
    self.controller.message_send(flow_mod_msg3)
    do_barrier(self.controller)

    num_pkt1s = random.randint(1,10)
    logging.info("Sending " + str(num_pkt1s) + " pkt1s")
    num_pkt2s = random.randint(1,10)
    logging.info("Sending " + str(num_pkt2s) + " pkt2s")
        num_pkt3s = random.randint(1,10)
    logging.info("Sending " + str(num_pkt3s) + " pkt3s")
    for i in range(0,num_pkt1s):
      sendPacket(self, pkt1, ingress_port1, egress_port1, test_timeout)
    for i in range(0,num_pkt2s):
      sendPacket(self, pkt2, ingress_port2, egress_port2, test_timeout)
        for i in range(0,num_pkt3s):
      sendPacket(self, pkt3, ingress_port1, egress_port2, test_timeout)

    match1 = packet_to_flow_match(self, pkt1)
    logging.info("Verifying flow1's " + str(num_pkt1s) + " packets")
    self.verifyStats(match1, ofp.OFPP_NONE, test_timeout, num_pkt1s)
    match2 = packet_to_flow_match(self, pkt2)
    logging.info("Verifying flow2's " + str(num_pkt2s) + " packets")
    self.verifyStats(match2, ofp.OFPP_NONE, test_timeout, num_pkt2s)
        match3 = packet_to_flow_match(self, pkt3)
    logging.info("Verifying flow3's " + str(num_pkt3s) + " packets")
    self.verifyStats(match3, ofp.OFPP_NONE, test_timeout, num_pkt3s)
    logging.info(match1.wildcards)
    match1.wildcards |= ofp.OFPFW_DL_DST
    logging.info(match1.wildcards)
    logging.info("Verifying combined " + str(num_pkt1s+num_pkt2s+num_pkt3s) + "
packets")
    self.verifyStats(match1, ofp.OFPP_NONE, test_timeout,
                num_pkt1s+num_pkt2s+num_pkt3s)
    # TODO: sweep through the wildcards to verify matching?
```

- **class my_test3**

```
class my_test3(base_tests.SimpleDataPlane):
   """
…
   def runTest(self):
      # TODO: set from command-line parameter
      test_timeout = 10

      of_ports = config["port_map"].keys()
      of_ports.sort()
      self.assertTrue(len(of_ports) >= 3, "Not enough ports for test")
      ingress_port1 = of_ports[0];
           ingress_port2 = of_ports[1];
      egress_port1 = of_ports[2];
      egress_port2 = of_ports[3];
      delete_all_flows(self.controller)

      pkt1 = optical_packet(wavelength=1565)
      flow_mod_msg1 = self.buildFlowModMsg(pkt1, ingress_port1, egress_port2)
      logging.info("Inserting optical flow1")
      self.controller.message_send(flow_mod_msg1)
      do_barrier(self.controller)

      num_pkt1s = 1
      logging.info("Sending " + str(num_pkt1s) + " optical pkt1s")
      for i in range(0,num_pkt1s):
         sendPacket(self, pkt1, ingress_port1, egress_port2, test_timeout)
              #self.dataplane.send(ingress_port1, str(pkt1))
      match1 = packet_to_flow_match(self, pkt1)
      logging.info("Verifying optical flow1's " + str(num_pkt1s) + " optical packets")
      self.verifyStats(match1, egress_port2, test_timeout, num_pkt1s)
      match1.wildcards |= ofp.OFPFW_WAVE
      logging.info("Verifying combined " + str(num_pkt1s) + " optical packets")
      #self.verifyStats(match1, ofp.OFPP_NONE, test_timeout,
                 #  num_pkt1s)
      # TODO: sweep through the wildcards to verify matching?
```

- **class my_test4**

```
class my_test4(base_tests.SimpleDataPlane):

…
   def runTest(self):
      # TODO: set from command-line parameter
      test_timeout = 10

      of_ports = config["port_map"].keys()
```

```
of_ports.sort()
self.assertTrue(len(of_ports) >= 3, "Not enough ports for test")
ingress_port1 = of_ports[0];
    ingress_port2 = of_ports[1];
egress_port1 = of_ports[2];
egress_port2 = of_ports[3];
delete_all_flows(self.controller)
    wave = 1560
    logging.info("The wavelength of the incoming packet is " + str(wave) + "nm")
    if wave == 1565:
    pkt1 = optical_packet(wavelength=1565)
    flow_mod_msg1 = self.buildFlowModMsg(pkt1, ingress_port1, egress_port1)
            logging.info("Inserting optical flow1")
            self.controller.message_send(flow_mod_msg1)
    elif wave == 1560:
    pkt2 = optical_packet(wavelength=1560)
    flow_mod_msg2 = self.buildFlowModMsg(pkt2, ingress_port1, egress_port2)
            logging.info("Inserting optical flow2")
    self.controller.message_send(flow_mod_msg2)

    do_barrier(self.controller)
    if wave == 1565:
    num_pkt1s = 1
    logging.info("Sending " + str(num_pkt1s) + " optical pkt1s")
            for i in range(0,num_pkt1s):
            sendPacket(self, pkt1, ingress_port1, egress_port1, test_timeout)
            match1 = packet_to_flow_match(self, pkt1)
elif wave == 1560:
            num_pkt2s = 1
    logging.info("Sending " + str(num_pkt2s) + " optical pkt2s")
    for j in range(0,num_pkt2s):
            sendPacket(self, pkt2, ingress_port1, egress_port2, test_timeout)
            match2 = packet_to_flow_match(self, pkt2)

required_wildcards = ofp.OFPFW_WAVE|ofp.OFPFW_IN_PORT
    logging.info("The required wildcards= "+str(required_wildcards))
if wave == 1565:
            match1.wildcards &=required_wildcards
elif wave == 1560:
            match2.wildcards &=required_wildcards
    if wave == 1565:
    if match1.wildcards == 0:
            match1.wildcards = ofp.OFPFW_WAVE|ofp.OFPFW_IN_PORT
            logging.info("Wildcards of successful match= "+str(match1.wildcards))
            logging.info("Matched value of wavelength= "+ str(match1.wave))
            logging.info("Verifying " + str(num_pkt1s) + " optical pkt1s")
            self.verifyStats(match1, ofp.OFPP_NONE, test_timeout,
                    num_pkt1s)
            else:
                    logging.info("packet is dropped because of miss match")
```

```
    elif wave == 1560:
                if match2.wildcards == 0:
                        match2.wildcards = ofp.OFPFW_WAVE|ofp.OFPFW_IN_PORT
                logging.info("Wildcards of successful match= "+str(match2.wildcards))
                logging.info("Matched value of wavelength= "+ str(match2.wave))
                logging.info("Verifying " + str(num_pkt2s) + " optical pkt2s")
                self.verifyStats(match2, ofp.OFPP_NONE, test_timeout,
                        num_pkt2s)
                else:
                        logging.info("packet is dropped because of miss match")
    else:
        logging.info("packet is dropped because of miss match")
    # TODO: sweep through the wildcards to verify matching?
```

<u>Αλλαγές που έγιναν στο φάκελο του OFTest αλλά και του scapy για την ενσωμάτωση του μήκους κύματος</u>

const.py (αλλαγές για την προσθήκη του μήκους κύματος)

Group ofp_port_features

```
# Identifiers from group ofp_flow_wildcards
OFPFW_IN_PORT = 1
OFPFW_DL_VLAN = 2
OFPFW_DL_SRC = 4
OFPFW_NW_DST_BITS = 6
OFPFW_NW_SRC_BITS = 6
OFPFW_NW_SRC_SHIFT = 8
OFPFW_DL_DST = 8
OFPFW_NW_DST_SHIFT = 14
OFPFW_DL_TYPE = 16
OFPFW_NW_PROTO = 32
OFPFW_TP_SRC = 64
OFPFW_TP_DST = 128
OFPFW_NW_SRC_ALL = 8192
OFPFW_NW_SRC_MASK = 16128
OFPFW_NW_DST_ALL = 524288
OFPFW_NW_DST_MASK = 1032192
OFPFW_DL_VLAN_PCP = 1048576
OFPFW_NW_TOS = 2097152
OFPFW_WAVE = 4194304
OFPFW_ALL = 8388607

ofp_flow_wildcards_map = {
1: 'OFPFW_IN_PORT',
2: 'OFPFW_DL_VLAN',
4: 'OFPFW_DL_SRC',
8: 'OFPFW_DL_DST',
```

```
16: 'OFPFW_DL_TYPE',
32: 'OFPFW_NW_PROTO',
64: 'OFPFW_TP_SRC',
128: 'OFPFW_TP_DST',
1048576: 'OFPFW_DL_VLAN_PCP',
2097152: 'OFPFW_NW_TOS',
4194304: 'OFPFW_WAVE',
}
```

class optical_layer.py

```
import os,struct,time
from scapy.packet import Packet,bind_layers
from scapy.base_classes import Net
from scapy.config import conf
from scapy.packet import *
from scapy.ansmachine import *
from scapy.plist import SndRcvList
from scapy.fields import *
from scapy.sendrecv import srp,srp1
from scapy.fields import ShortField
from scapy.layers.l2 import *
from scapy.layers.inet import *

class Optical(Packet):
        name = "Optical"
        fields_desc = [OpticalField("wave", None, 1550)]

bind_layers(Ether, Optical, type=400)
```

Προσθήκη στο αρχείο fields.py της κλάσης OpticalField

```
class OpticalField(Field):
  """ variable length quantities """

  def __init__(self, name, default, fld):
    Field.__init__(self, name, default)
    self.fld = fld

  def i2m(self, pkt, x):
    if x is None:
      f = pkt.get_field(self.fld)
      x = f.i2len(pkt, pkt.getfieldval(self.fld))
      x = vlenq2str(x)
    return str(x)

  def m2i(self, pkt, x):
    if s is None:
```

```
            return None, 0
        return str2vlenq(x)[1]

    def addfield(self, pkt, s, val):
        return s+self.i2m(pkt, val)

    def getfield(self, pkt, s):
        return s, self.fld
```

packet.py

```
import sys

try:
import scapy.config
import scapy.route
import scapy.route6
import scapy.layers.l2
import scapy.layers.inet
import scapy.layers.inet6
import scapy.layers.optical_layer

except ImportError:
sys.exit("Need to install scapy for packet parsing")

Ether = scapy.layers.l2.Ether
LLC = scapy.layers.l2.LLC
SNAP = scapy.layers.l2.SNAP
Dot1Q = scapy.layers.l2.Dot1Q
IP = scapy.layers.inet.IP
IPOption = scapy.layers.inet.IPOption
IPv6 = scapy.layers.inet6.IPv6
ARP = scapy.layers.inet.ARP
TCP = scapy.layers.inet.TCP
UDP = scapy.layers.inet.UDP
ICMP = scapy.layers.inet.ICMP
ICMPv6Unknown = scapy.layers.inet6.ICMPv6Unknown
ICMPv6EchoRequest = scapy.layers.inet6.ICMPv6EchoRequest
Optical = scapy.layers.optical_layer.Optical
```

testutils.py (αλλαγές για τη δημιουργία του οπτικού πακέτου)

```
def optical_packet(pktlen=60,
wavelength=1550):


if MINSIZE > pktlen:
pktlen = MINSIZE
```

```
pkt = scapy.Optical(wave=wavelength)
logging.info(wavelength)
pkt = pkt/("D" * (pktlen - len(pkt)))
return pkt
```

parse.py (αλλαγές για την υποστήριξη των οπτικών ιδιοτήτων)

```
def packet_to_flow_match_v0(packet):

    import loxi.of10 as ofp


    optic = packet

    match = ofp.match()
    match.wave = optic.wave
    #logging.info(match.wave)
    match.wildcards = ofp.OFPFW_ALL
    match.wildcards &= ~ofp.OFPFW_IN_PORT
    #logging.info(match.wildcards)
    match.wildcards &= ~ofp.OFPFW_WAVE
    #logging.info(match.wildcards)
    try:
        wave = packet_type_classify(optic)
    except:
        raise ValueError("could not classify packet")

    return match
```