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

Προσθήκη διεπαφής τεχνολογίας 802.3ad σε σύνθετο περιβάλλον SDX OpenFlow

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλέξανδρος Ι. Δούκας

Επιβλέπων : Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2015

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

Προσθήκη διεπαφής τεχνολογίας 802.3ad σε σύνθετο περιβάλλον SDX OpenFlow

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλέξανδρος Ι. Δούκας

Επιβλέπων : Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ..... Δεκεμβρίου 2015.

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

Αθήνα, Δεκέμβριος 2015

.....

Αλέξανδρος Ι. Δούκας

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

Περίληψη

Στην περίοδο που διανύουμε έχει αρχίσει να διαδίδεται όλο και περισσότερο η χρήση των Προγραμματιζόμενων Δικτύων (Software Defined Networks) καθώς προσφέρουν πλεονεκτήματα σε αρκετές εφαρμογές σε σχέση με τα συμβατικά δίκτυα. Η χρήση τους πρόσφατα έχει επεκταθεί στο πεδίο των Παρόχων Υπηρεσιών Διαδικτύου (Internet Service Providers) καθώς και σε νέες υλοποιήσεις του πρωτοκόλλου Border Gateway Protocol (BGP). Ως εκ τούτου, έχουν αρχίσει να υλοποιούνται εφαρμογές SDN και για Internet Exchange Points (IXPs) με στόχο την υιοθέτησή τους σε πραγματικά περιβάλλοντα IXP.

Η παρούσα εργασία πραγματεύεται την προσθήκη διεπαφών τεχνολογίας 802.3ad (πρωτόκολλο Link Aggregation Control Protocol - LACP) σε μία υπάρχουσα υλοποίηση Software Defined Internet Exchange Point (SDX) η οποία βασίζεται στο πρωτόκολλο OpenFlow για Software Defined Networks (SDN). Αρχικά γίνεται μία βιβλιογραφική επισκόπηση του πρωτοκόλλου OpenFlow, του πρωτοκόλλου BGP καθώς και της τεχνολογίας 802.3ad. Στη συνέχεια γίνεται μία παρουσίαση των εργαλείων Ryu SDN Framework και Mininet network emulator που χρησιμοποιήθηκαν για την υλοποίηση. Τέλος αναλύεται εκτενώς η συνολική λειτουργία της υλοποίησης που έγινε σε γλώσσα python με παράθεση παραδειγμάτων καθώς και μερών από τον πηγαίο κώδικα.

Λέξεις Κλειδιά: Προγραμματιζόμενα Δίκτυα, OpenFlow, Ryu, IXP, BGP, πολιτικές προώθησης πακέτων, LACP, SDX, SDN, 802.3ad

Abstract

In the present time period the use of Software Defined Networks (SDN) is increasing rapidly. This happens because the SDN have some advantages over conventional networks in several applications. The use of the SDN philosophy is expanding in the field of Service Providers (ISPs) and also in new Border Gateway Protocol (BGP) implementations. Therefore, several implementations about Software Defined Internet Exchanged Points have been developed in order to be adopted soon in working environments.

This thesis is about the addition of Link Aggregation Control Protocol (LACP) capabilities (IEEE 802.3ad) in one Software Defined Internet Exchange (SDX) implementation based in the OpenFlow protocol.

In the beginning, the technologies that have been used are being introduced. The OpenFlow protocol, the BGP protocol, the function of an IXP and also the LACP protocol. Later we present the tools that have been used, the Ryu SDN Framework and the Mininet network emulator. Finally, we dive into details of the solution of an SDX controller that has been adopted and of the addition of the LACP capabilities by using examples and parts from the source code.

Keywords: Software Defined Networks, SDN, SDX, OpenFlow, Ryu, Internet Exchange Point, IXP, BGP, forwarding policies, LACP, 802.3ad

Ευχαριστίες

Με την ολοκλήρωση της παρούσας διπλωματικής εργασίας, θα ήθελα να εκφράσω τις ιδιαίτερες ευχαριστίες μου σε όσους βοήθησαν για την εκπόνησή της.

Αρχικά θα ήθελα να ευχαριστήσω θερμά τον Ερευνητή του ΕΠΙΣΕΥ, κ. Δημήτρη Καλογερά, για την καθοδήγησή του και την άψογη συνεργασία σε κάθε βήμα αυτής της εργασίας. Επίσης θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή, κ. Ευστάθιο Συκά για το ιδιαίτερα ενδιαφέρον θέμα που μου εμπιστεύτηκε.

Τέλος, ιδιαίτερες ευχαριστίες θα ήθελα να αποδώσω στην οικογένειά μου για τη στήριξή τους και στους φίλους μου για την κατανόηση που έχουν δείξει σε όλη τη διάρκεια των σπουδών μου.

Περιεχόμενα

Πίνακας Εικόνων

1. Εισαγωγή

1.1 Πρόβλημα – Προσέγγιση

Οι τεχνολογίες των δικτύων εξελίσσονται ραγδαία, με σταθμό ορόσημο την δημιουργία των Προγραμματιζόμενων Δικτύων (Software Defined Networks - SDN). Τα SDN έδωσαν την δυνατότητα στους μηχανικούς δικτύων να καινοτομήσουν σε κάθε παράμετρο των δικτύων όπως η ταχύτητα, η εξισορρόπηση φορτίου, η ασφάλεια και ο πλήρης έλεγχος της διαδρομής των πακέτων στο επίπεδο των εφαρμογών (application layer). Ειδικότερα στο τελευταίο, το SDN δίνει απεριόριστες επιλογές στους προγραμματιστές καθώς μπορούν πλέον να δημιουργούν εφαρμογές οι οποίες ελέγχουν τη ροή της κίνησης κατά μήκος ολόκληρης της διαδρομής χωρίς να περιορίζονται μόνο στο ταίριασμα των διευθύνσεων και των πορτών πηγής και προορισμού.

Η υιοθέτηση της φιλοσοφίας των SDN σε τομείς όπως η διασύνδεση Παρόχων Υπηρεσιών Διαδικτύου (Internet Service Provider – ISP) και των μεταξύ τους δικτύων, όπου θα μπορούσαν να υπάρξουν πολλά οφέλη, είναι ακόμα σε αρχικά στάδια. Υπάρχουν κάποιες πειραματικές υλοποιήσεις για προγραμματιζόμενα σημεία ανταλλαγής κίνησης διαδικτύου (software defined internet exchange points) αλλά και πάλι έχουν ελλείψεις σε θέματα εξισορρόπησης φορτίου και υποστήριξης πολλαπλών διασυνδέσεων μεταξύ των κόμβων που συμμετέχουν. Η υποστήριξη τέτοιων λειτουργιών είναι ζωτικής σημασίας σε επίπεδο ISP.

1.2 Συνεισφορά Εργασίας

Η εργασία αυτή έχει ως στόχο την προσθήκη της υποστήριξης της τεχνολογίας της IEEE, 802.3ad, για τη δημιουργία διασυνδέσεων που αποτελούνται από ομάδες πολλαπλών φυσικών διεπαφών, σε SDN υλοποιήσεις σημείων ανταλλαγής κίνησης διαδικτύου. Στα πλαίσια της εργασίας, λύνεται το πρόβλημα της προώθησης πακέτων προς έναν δρομολογητή σε περίπτωση σφάλματος μίας εκ των φυσικών διεπαφών του και ωθεί την κοινότητα του SDN να ασχοληθεί πιο ενεργά με θέματα υποστήριξης εφεδρικών διασυνδέσεων στις νέες εφαρμογές. Συγκεκριμένα:

- Επεκτείνεται η λειτουργία του ανοιχτού λογισμικού εξομοιωτή τοπολογιών δικτύων Mininet με κλάσεις που αυτοματοποιούν τη δημιουργία τοπολογιών με πολλαπλές διασυνδέσεις (multiple links) μεταξύ των κόμβων τους.
- Τροποποιείται η λειτουργία της υλοποίησης SDX ώστε να υποστηρίζει πλέον το πρωτόκολλο LACP και να αποτρέπει σφάλματα απώλειας της σύνδεσης για τους συμμετέχοντες του.

1.3 Δομή Εργασίας

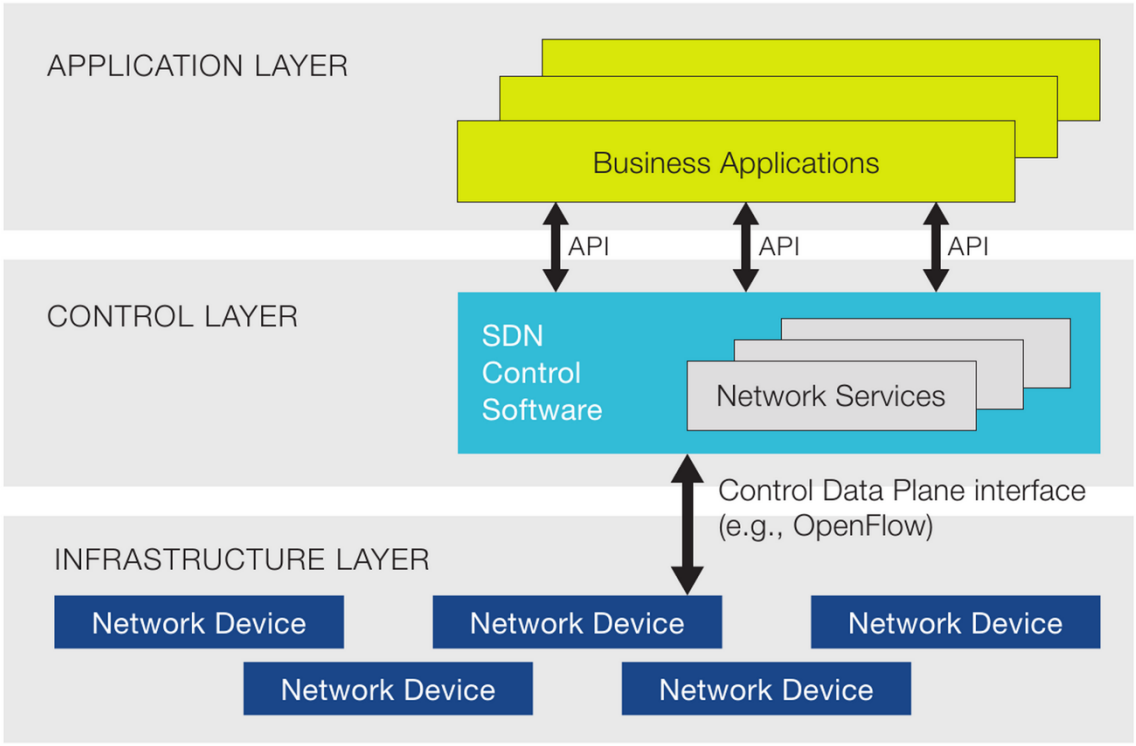
Η διπλωματική εργασία αυτή αποτελείται από πέντε κεφάλαια. Στο πρώτο κεφάλαιο (**Εισαγωγή**), παρουσιάζονται οι ανάγκες που οδήγησαν στην δημιουργία της συγκεκριμένης εργασίας καθώς και η δομή της. Στο επόμενο κεφάλαιο (**Θεωρητικό Υπόβαθρο**) παρουσιάζεται η αρχιτεκτονική δικτύων Software Defined Networks (SDN) και τα πρωτόκολλα που χρησιμοποιούνται. Στο τρίτο κεφάλαιο (**Ryu-Mininet**) αναλύεται το Ryu SDN Framework καθώς και ο εξομοιωτής δικτύων mininet. Στο τέταρτο κεφάλαιο (**Επισκόπηση SDX**) αναλύουμε την αρχιτεκτονική και τη λειτουργία του Software Defined Internet Exchange Point (SDX) που χρησιμοποιήθηκε σαν βάση για την ανάπτυξη του συστήματός μας. Στο πέμπτο κεφάλαιο (**Σύστημα Υλοποίησης SDX με διεπαφές LACP**) παρουσιάζεται αναλυτικά ο τρόπος υλοποίησης του συστήματός μας. Τέλος στο έκτο και τελευταίο κεφάλαιο (**Επεκτάσεις**) αναλύονται τα συμπεράσματα της υλοποίησης αυτής.

2. Θεωρητικό Υπόβαθρο

2.1 Software Defined Networks (SDN)

Το SDN παρουσιάζεται ως μια δυναμική, διαχειρίσιμη, οικονομικά αποδοτική και εύκολα προσαρμόσιμη αρχιτεκτονική, κατάλληλη για τις υψηλών απαιτήσεων δυναμικές εφαρμογές της σημερινής εποχής.

Η διαφορά της σε σχέση με τα υπάρχοντα δίκτυα είναι η αποσύνδεση των λειτουργιών ελέγχου του δικτύου με τις λειτουργίες προώθησης. Πλέον, δεν είναι αναγκαίος ο έλεγχος και έπειτα η προώθηση των πακέτων του δικτύου από τον κάθε μεταγωγέα ή δρομολογητή με τα δικά του πρωτόκολλα, αλλά αρκείται στο να τα προωθεί. Το κομμάτι του ελέγχου μπορεί να γίνεται απομακρυσμένα δίνοντας στους υπεύθυνους του δικτύου την δυνατότητα να δημιουργήσουν ισχυρές και καινοτόμες εφαρμογές.



Εικόνα 1: Η αρχιτεκτονική SDN

Η αρχιτεκτονική του SDN διαχωρίζει το δίκτυο σε 3 επίπεδα: εφαρμογών, ελέγχου και δεδομένων ή υποδομών, όπως φαίνεται παρακάτω :

- **Επίπεδο εφαρμογών:** Περιέχει εφαρμογές SDN οι οποίες επικοινωνούν μέσω διεπαφής ανάπτυξης εφαρμογών (Application Programming Interface - API) με τον ελεγκτή (controller) και του ανακοινώνουν τις επιθυμητή συμπεριφορά του δικτύου.
- **Επίπεδο ελέγχου:** Είναι ένα λειτουργικό σύστημα δικτύου, το οποίο λαμβάνει οδηγίες ή απαιτήσεις από το στρώμα εφαρμογών SDN και τις αναμεταδίδει στο επίπεδο δεδομένων. Οι αποφάσεις λαμβάνονται με μια γενική άποψη ολοκλήρου του δικτύου και όχι με την περιορισμένη ορατότητα των γειτονικών δικτυακών συσκευών, όπως κάνουν οι δρομολογητές σήμερα.
- **Επίπεδο δεδομένων:** Εδώ έχουμε τις συσκευές υλικού στις οποίες πραγματοποιείται η προώθηση των πακέτων.

Τα βασικά στοιχεία ενός δικτύου αρχιτεκτονικής SDN είναι οι ελεγκτές στο επίπεδο ελέγχου, οι μεταγωγείς στο επίπεδο δεδομένων και το πρωτόκολλο επικοινωνίας τους με πιο διαδεδομένο το OpenFlow.

2.1.1 Το πρωτόκολλο OpenFlow

Το πρωτόκολλο δικτύων OpenFlow είναι η διεπαφή επικοινωνιών μεταξύ του επιπέδου ελέγχου και προώθησης σε μια SDN αρχιτεκτονική.

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

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

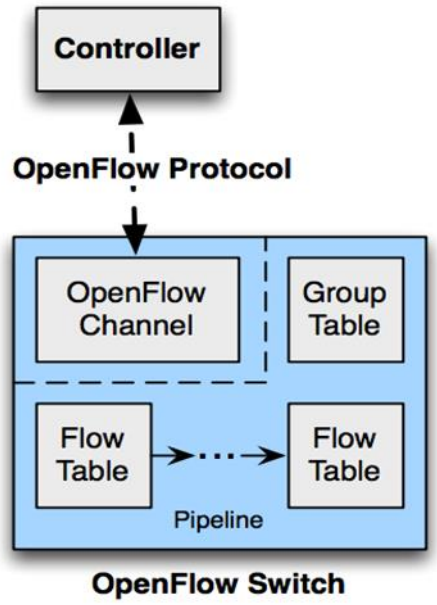
2.1.2 Ελεγκτής OpenFlow

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

Υπάρχουν τρεις κατηγορίες επικοινωνίας στο πρωτόκολλο OpenFlow: (α) η ελεγκτής - μεταγωγέας, (β) η ασύγχρονη και (γ) η συμμετρική επικοινωνία. Όλες υλοποιούνται μέσω ενός ασφαλούς καναλιού ελέγχου. Η επικοινωνία ελεγκτής-μεταγωγέας είναι υπεύθυνη για την ανίχνευση χαρακτηριστικών, την παραμετροποίηση, τον προγραμματισμό του μεταγωγέα και την ανάκτηση πληροφοριών. Μια ασύγχρονη επικοινωνία ενεργοποιείται από τον μεταγωγέα χωρίς καμία πρόσκληση από τον ελεγκτή. Χρησιμοποιείται για να ενημερώσει τον ελεγκτή για τη άφιξη πακέτων, την αλλαγή κατάστασης του και τυχόν λάθη που προέκυψαν. Τέλος, μία συμμετρική επικοινωνία υλοποιείται όταν αποστέλλονται μηνύματα χωρίς πρόσκληση από καμία από τις δύο πλευρές, δηλαδή, τόσο ο μεταγωγέας όσο και ο ελεγκτής είναι ελεύθεροι να εκκινήσουν την επικοινωνία χωρίς πρόσκληση από την άλλη πλευρά. Παραδείγματα για συμμετρική επικοινωνία είναι “hello” ή “echo” μηνύματα που μπορούν να χρησιμοποιηθούν για να προσδιοριστεί εάν το κανάλι ελέγχου εξακολουθεί να είναι διαθέσιμο.

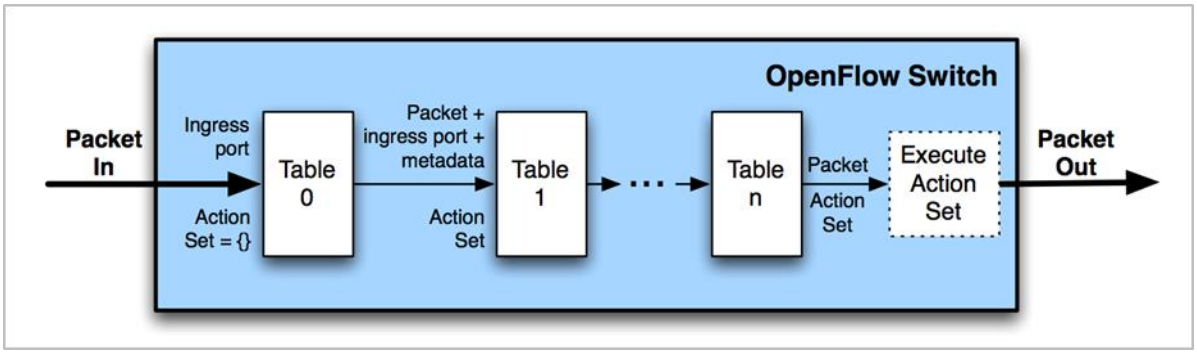
2.1.3 Μεταγωγέας OpenFlow

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



Εικόνα 2: Ο μεταγωγέας OpenFlow

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



Εικόνα 3: Οι πίνακες ροής ενός μεταγωγέα OpenFlow

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

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

2.1.4 Πίνακας Ροών

Κάθε εγγραφή του πίνακα ροών περιέχει όπως φαίνεται και στην Εικόνα 4 τα παρακάτω :

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Main components of a flow entry in a flow table.

Εικόνα 4: Βασική δομή ενός πίνακα ροής

- **Πεδία ταιριάσματος:** Γίνεται ταίριασμα με κάθε πακέτο. Τα πεδία αυτά είναι η πόρτα (port) εισόδου, οι κεφαλίδες του πακέτου καθώς και προαιρετικά δεδομένα που μπορεί να έχουν εισαχθεί από προηγούμενους πίνακες ροών.
- **Προτεραιότητα:** Προτεραιότητα ταιριάσματος της εγγραφής σε σχέση με άλλες εγγραφές
- **Μετρητές:** Αυξάνονται κατά το ταίριασμα πακέτων (παραδείγματα : μετρητής πακέτων που ταίριαξαν ή μετρητής συνολικών πακέτων εισόδου στη πόρτα εισόδου κλπ.)
- **Οδηγίες:** Λένε στο μεταγωγέα τί πρέπει να κάνει με το πακέτο που μόλις έχει ταιριάξει στην εγγραφή.
- **Χρονικά όρια:** Η μέγιστη χρονική τιμή που μπορεί να μείνει η εγγραφή στον πίνακα
- **Cookies:** Μπορεί να χρησιμοποιηθεί από τον ελεγκτή για να φιλτράρει στατιστικά για τις ροές ή για αλλαγή-διαγραφή ροών, όμως δε χρησιμοποιείται κατά την επεξεργασία των πακέτων.

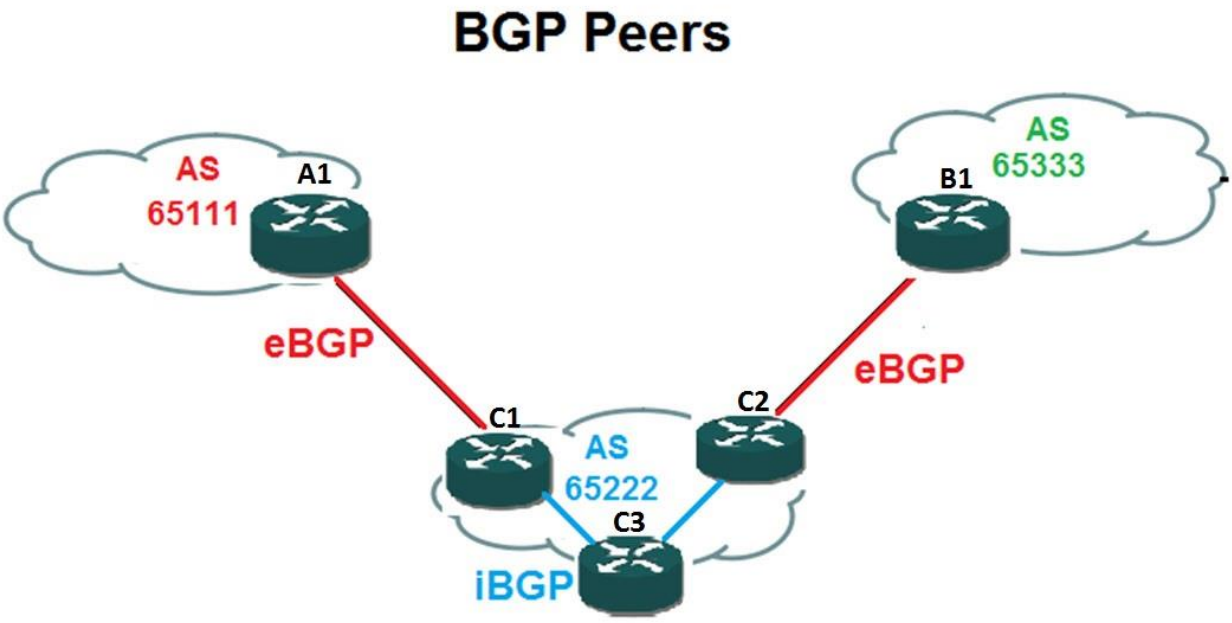
Γενικότερα, μία εγγραφή του πίνακα ροών ταυτοποιείται από τα παιδία ταιριάσματος και την προτεραιότητα της. Αυτά κάνουν την εγγραφή μοναδική στον πίνακα.

2.2 Border Gateway Protocol (BGP)

Το BGP είναι ένα πρωτόκολλο εξωτερικής πύλης (exterior gateway protocol) που έχει σχεδιαστεί για να ανταλλάσσει πληροφορίες προσβασιμότητας και δρομολόγησης στο Διαδίκτυο ανάμεσα σε ξένα μεταξύ τους δίκτυα ή αλλιώς αυτόνομα συστήματα (autonomous systems). Οι αποφάσεις δρομολόγησής του βασίζονται στις διαδρομές (paths), στις πολιτικές του δικτύου καθώς και σε σύνολα κανόνων που έχουν δημιουργηθεί από τον διαχειριστή του δικτύου. Το BGP χρησιμοποιείται κυρίως από Παρόχους Υπηρεσιών Διαδικτύου (ISPs) και μεγάλα εταιρικά δίκτυα.

2.2.1 BGP Peers

Οι συνδέσεις BGP μεταξύ δύο δρομολογητών (BGP peers) εγκαθίστανται με διαμόρφωση του καθενός ώστε να δημιουργήσει μία συνεδρία TCP στην πόρτα 179. Για να διατηρηθεί η διασύνδεση μεταξύ τους, στέλνεται ένα μήνυμα “keep-alive” κάθε 60 δευτερόλεπτα. Μεταξύ των πρωτοκόλλων δρομολόγησης, το BGP είναι το μοναδικό που χρησιμοποιεί το TCP σαν το πρωτόκολλο μεταφοράς του. Έτσι οι δύο δρομολογητές (που έχουν δημιουργήσει επικοινωνία BGP μεταξύ τους) δεν είναι απαραίτητο να έχουν μία φυσική διασύνδεση μεταξύ τους, δηλαδή να είναι άμεσα συνδεδεμένοι.



Εικόνα 5: Διαφορές μεταξύ internal και external BGP

Όταν το BGP τρέχει μεταξύ δύο δρομολογητών στο ίδιο αυτόνομο σύστημα τότε συνήθως αναφέρεται ώς εσωτερικό BGP (internal BGP, iBGP) ενώ αντίθετα όταν οι δρομολογητές δεν ανήκουν στο ίδιο αυτόνομο σύστημα, τότε καλείται εξωτερικό BGP(external BGP, eBGP). Οι δρομολογητές που τρέχουν μεταξύ τους eBGP ονομάζονται συνοριακοί δρομολογητές και συνήθως είναι συνδεδεμένοι άμεσα μεταξύ τους. Στην Εικόνα 5 φαίνεται η διάκριση μεταξύ eBGP και iBGP συνδέσεων (οι συνδέσεις A1↔C1 και B1↔C2 είναι eBGP ενώ οι C1↔C3 και C2↔C3 είναι iBGP).

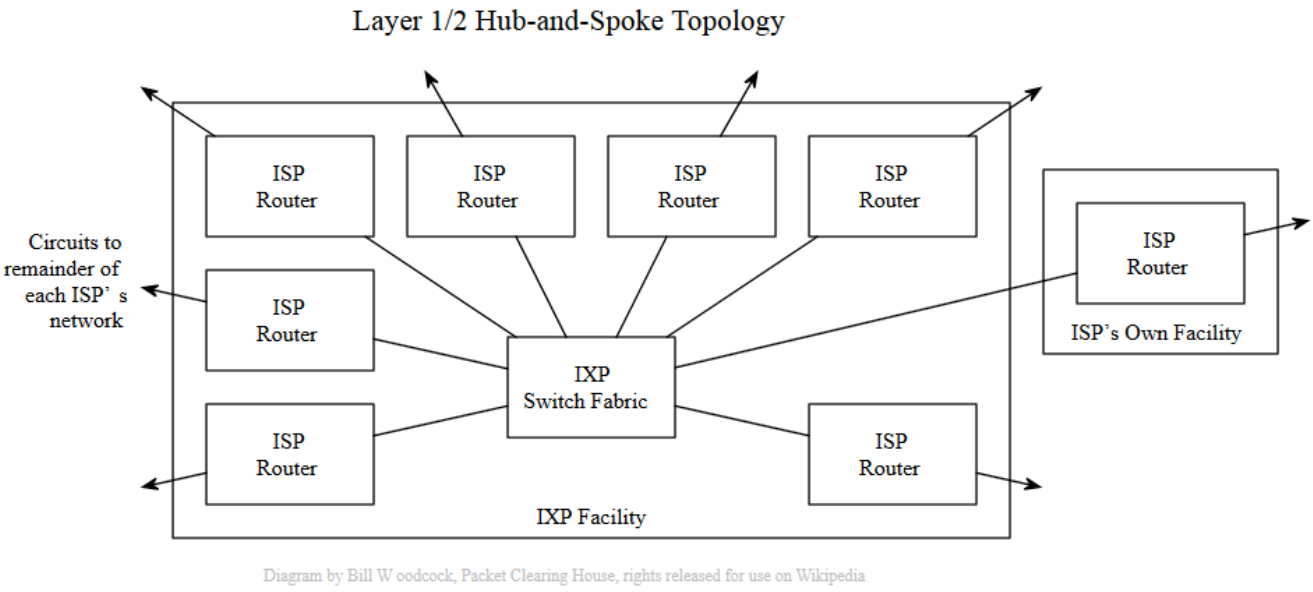
2.2.2 BGP Path Attributes

Για να επιτραπούν οι συνδέσεις μεταξύ δύο συμμετεχόντων στο BGP θα πρέπει να υπάρξει μία συμφωνία μεταξύ τους που συνήθως έχει περίπλοκες πολιτικές δρομολόγησης. Για αυτό το λόγο, το BGP παρέχει ένα μεγάλο αριθμό από “γνωρίσματα” (attributes).Τα γνωρίσματα αυτά χωρίζονται σε δύο κατηγορίες. Σε αυτά που είναι υποχρεωτικό από τον προσδιορισμό του BGP να υπάρχουν σε κάθε δρομολογητή, ανεξάρτητα τον κατασκευαστή του, τα λεγόμενα δημοφιλή (well-known attributes) και στα προαιρετικά (optional attributes). Τα πιο ευρέως χρησιμοποιούμενα είναι :

- **AS path:** Η συνολική διαδρομή που δείχνει από ποιά Αυτόνομα Συστήματα πρέπει να περάσει ένα πακέτο για να φτάσει στο προορισμό του.
- **Local preference:** Το εσωτερικό κόστος (cost) ενός προορισμού (χρησιμοποιείται για να εξασφαλίζει συνοχή σε όλο το αυτόνομο σύστημα).
- **Multi-exit discriminator:** Δίνει τη δυνατότητα σε άλλα αυτόνομα συστήματα να προτιμούν ένα σημείο γειτνίασης με ένα άλλο σύστημα (δεδομένου ότι έχουν πολλαπλά τέτοια σημεία) από ένα άλλο.
- **Community:** Ένα σύνολο από γενικές ετικέτες (tags) που χρησιμοποιείται για να ορίσει διάφορες διαχειριστικές πολιτικές μεταξύ των δρομολογητών.

2.3 Internet Exchange Point (IXP)

Ένα Σημείο Ανταλλαγής πληροφορίας Διαδικτύου (IXP) είναι μία υποδομή μέσω της οποίας Πάροχοι Υπηρεσιών Διαδικτύου καθώς και Δίκτυα Διανομής Πληροφοριών (Content Delivery Networks) ανταλλάσσουν διαδικτυακή κίνηση μεταξύ των δικτύων τους. Ο κύριος σκοπός ενός IXP είναι να επιτρέψει στα δίκτυα αυτά να συνδεθούν μεταξύ τους απευθείας, χωρίς να υπάρχουν ενδιάμεσα τρίτα δίκτυα. Τα κυριότερα πλεονεκτήματα αυτής της διασύνδεσης είναι καλύτερο κόστος, latency και εύρος ζώνης.

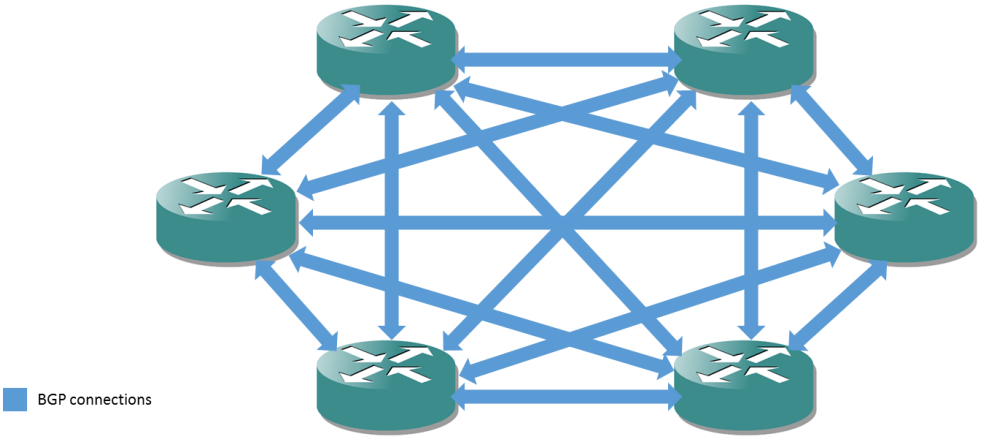


Εικόνα 6: Γραφική αναπαράσταση ενός IXP

Η Εικόνα 6 δείχνει ένα συνηθισμένο Internet Exchange Point. Αποτελείται από έναν ή περισσότερους μεταγωγείς δικτύου (switch fabric) πάνω στους οποίους συνδέεται το κάθε συμμετέχον δίκτυο. Η ανταλλαγή κίνησης μεταξύ των δικτύων διέπεται από συμφωνίες που έχουν κάνει οι εμπλεκόμενοι πάροχοι. Η λειτουργία του IXP διευκολύνεται από το BGP, καθώς δημιουργούνται γειτνιάσεις (BGP peerings) μεταξύ των δικτύων που συμμετέχουν.

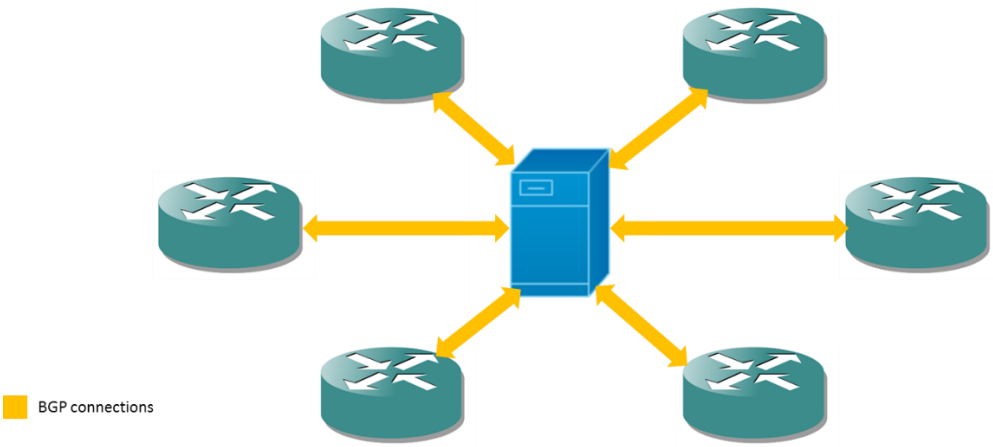
2.3.1 BGP Route Server

Κανονικά, ένα IXP για να επικοινωνήσουν όλα τα διαφορετικά αυτόνομα συστήματα μεταξύ τους, θα πρέπει να εγκαταστήσουν αμφίπλευρες συνδέσεις με όλους τους άλλους συμμετέχοντες. Καθώς όμως ο αριθμός των συμμετεχόντων αυξάνεται, γίνεται όλο και πιο δύσκολη και πολύπλοκη η διαχείριση των αυξανόμενων νέων συνδέσεων. Η λύση στο πρόβλημα δίνεται με τον εξυπηρετητή διαδρομών (route server).



Εικόνα 7: Τοπολογία με συνδέσεις μεταξύ όλων των μετόχων του IXP (full-mesh topology)

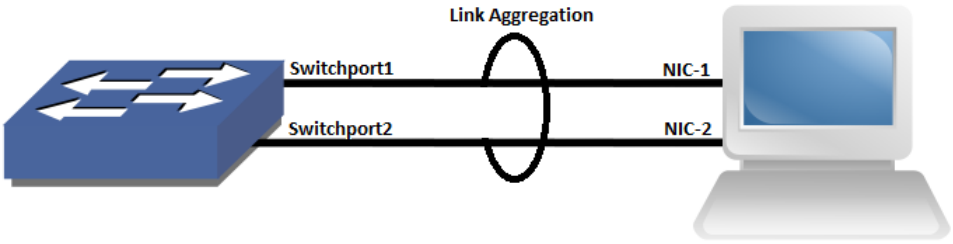
Ο εξυπηρετητής διαδρομών παρέχει μία εναλλακτική στη πλήρη διασύνδεση όλων των δρομολογητών που συμμετέχουν στο IXP. Με τη λειτουργία του, μειώνει την πολυπλοκότητα στη διαμόρφωση ενός δρομολογητή καθώς επίσης και την κατανάλωση πόρων του συστήματος του (μνήμη και επεξεργαστή). Προσφέρει επίσης υποστήριξη για πολιτικές και έτσι μπορεί να παρακάμψει την κλασική καλύτερη διαδρομή που παρέχει το BGP με μία διαφορετική που βασίζεται σε συγκεκριμένη πολιτική ή ακόμα και να μη διαφημίζει ένα συγκεκριμένο δίκτυο αν είναι επιθυμητό.



Εικόνα 8: Τοπολογία IXP με χρήση Route Server

2.4 Link Aggregation

Η συσσωμάτωση συνδέσμων (link aggregation) επιτρέπει έναν ή περισσότερους συνδέσμους να ενωθούν μεταξύ τους και να δημιουργήσουν μία ομάδα συνδέσεων τέτοια ώστε μία συσκευή επιπέδου 2 (Layer 2) να μεταχειρίζεται την ομάδα σαν να είναι ένας και μοναδικός σύνδεσμος. Χρησιμοποιείται για τη μεγέθυνση της διακίνησης δεδομένων που μπορεί να έχει μία σύνδεση αλλά και για λόγους πλεονασμού (redundancy) για την αποφυγή αποκοπής της σύνδεσης σε περίπτωση σφάλματος.



Εικόνα 9: Αναπαράσταση συσσωμάτωσης συνδέσμων

2.4.1 Link Aggregation Control Protocol (LACP)

Το πρωτόκολλο ελέγχου συσσωμάτωσης συνδέσμων (Link Aggregation Control Protocol, LACP) είναι μέρος του προσδιορισμού (specification) **802.3ad** της IEEE που επιτρέπει την δημιουργία δέσμης φυσικών θυρών (ports) σε ένα λογικό δίαυλο (logical channel). Επιτρέπει στη συσκευή δικτύου να διαπραγματευθεί αυτόματα τη δημιουργία της δέσμης, στέλνοντας κατάλληλα πλαίσια (frames) στη γειτονική συσκευή. Τα πλαίσια αυτά (LACPDUs) στέλνονται σε όλους τους συνδέσμους που έχει ενεργοποιηθεί το πρωτόκολλο. Αν στο άλλο άκρο υπάρχει συσκευή με το LACP επίσης ενεργοποιημένο τότε και αυτή θα στέλνει ανεξάρτητα από την πρώτη πλαίσια στους ίδιους συνδέσμους, επιτρέποντας έτσι τις δύο συσκευές να εντοπίσουν πολλαπλούς συνδέσμους μεταξύ τους και έτσι να δημιουργήσουν έναν μονό ενωμένο λογικό σύνδεσμο. Με το που θα λάβει η συσκευή σε έναν σύνδεσμό της ένα πλαίσιο LACP τότε αμέσως θα παράγει ένα πλαίσιο απάντησης. Το πρωτόκολλο μπορεί να διαμορφωθεί με δύο λειτουργίες, την ενεργή και την παθητική. Στην πρώτη, στέλνει πάντα πλαίσια στους ενεργοποιημένους συνδέσμους. Ωστόσο στη δεύτερη λειτουργία απλά περιμένει να λάβει στον σύνδεσμο ένα πλαίσιο από την απέναντι πλευρά. Είναι προφανές ότι σε μία διασύνδεση πρέπει πάντα να υπάρχει τουλάχιστον ένας ενεργός σύνδεσμος.

Τα πλεονεκτήματα του LACP σε σχέση με απλή στατική διαμόρφωση του κάθε συνδέσμου ώστε να δημιουργηθεί χειροκίνητα η συσσωμάτωση τους είναι:

- Η αυτόματη αποκατάσταση της επικοινωνίας στον λογικό σύνδεσμο σε περίπτωση σφάλματος σε έναν από τους φυσικούς συνδέσμους, καθώς με στατική συσσωμάτωση η γειτονική συσκευή θα συνέχιζε να στέλνει δεδομένα από την πλευρά της με αποτέλεσμα την απώλεια της επικοινωνίας.
- Οι συσκευές επικοινωνούν μεταξύ τους για το αν είναι και οι δύο σε θέση να υποστηρίξουν την συσσωμάτωση και αν όχι δεν πραγματοποιείται. Στη περίπτωση της στατικής συσσωμάτωσης θα είχαμε ανεπιθύμητες παρενέργειες στο δίκτυο αν για παράδειγμα είχε γίνει κάποιο λάθος από αμέλεια στην καλωδίωση ή στην διαμόρφωση των συσκευών (για παράδειγμα, ενεργοποίηση της συσσωμάτωσης μόνο σε έναν από τους δύο συνδέσμους σε μία πλευρά).

2.4.2 Linux Bonding Driver

Ο οδηγός δέσμευσης του Linux (Linux Bonding driver) παρέχει τις απαραίτητες μεθόδους για τη δημιουργία συσσωματώσεων από πολλαπλές δικτυακές διεπαφές. Η δικτυακή συμπεριφορά της συσσωματωμένης διεπαφής εξαρτάται από τη λειτουργία (mode) που έχει διαμορφωθεί. Οι διαθέσιμες λειτουργίες είναι :

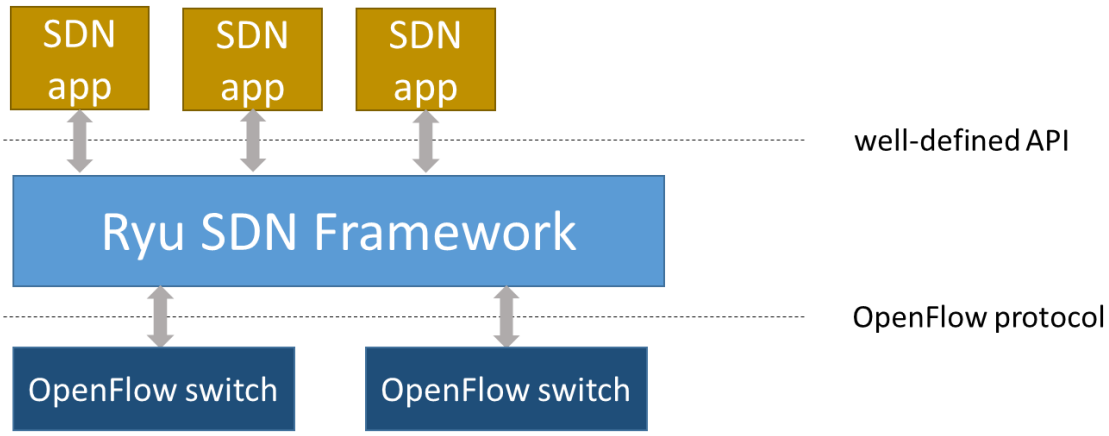
- balance-rr (mode 0): Πολιτική round-robin, δηλαδή μετάδωση των πακέτων με διαδοχική σειρά από τον πρώτο φυσικό σύνδεσμο μέχρι τον τελευταίο. Αυτή η λειτουργία προσφέρει εξισορρόπηση φορτίου καθώς και ανοχή σε σφάλματα.
- active-backup (mode 1): Μόνο ένας σύνδεσμος είναι ενεργός. Σε περίπτωση απώλειάς του τότε ενεργοποιείται ένας άλλος κ.ο.κ. Η λειτουργία αυτή παρέχει μόνο ανοχή στα σφάλματα.
- balance-xor (mode 2): Πολιτική μετάδοσης βασισμένη σε κατακερματισμό (hashing). Η προεπιλεγμένη πολιτική είναι ενδεικτικά “(source MAC address XOR destination MAC addres XOR packet type ID) modulo (slave interface count)”. Αυτή η λειτουργία παρέχει και εξισορρόπηση φορτίου και ανοχή σε σφάλματα.
- broadcast (mode 3): Μεταδίδει όλα τα πακέτα από όλους τους φυσικούς συνδέσμους ταυτόχρονα. Παρέχει μόνο ανοχή σε σφάλματα.
- 802.3ad (mode 4): Υλοποίηση του πρωτοκόλλου LACP
- balance-tlb (mode 5): Προσαρμοζόμενη εξισορρόπηση φορτίου μετάδοσης.
- balance-alb (mode 6): Προσαρμοζόμενη εξισορρόπηση φορτίου και λήψης και μετάδοσης.

3. Ryu και Mininet

Σε αυτό το κεφάλαιο παρουσιάζονται με λεπτομέρεια τα δύο βασικά περιβάλλοντα-εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση της εργασίας.

3.1 Ryu SDN Framework

Το Ryu είναι ένα προσδιοριζόμενο μέσω λογισμικού (software-defined) δικτυακό framework ανοιχτού κώδικα (διαθέσιμου υπό την άδεια Apache 2.0) βασισμένο σε συνιστώσες (component-based). Είναι γραμμένο πλήρως στη γλώσσα Python. Παρέχει συνιστώσες (components) λογισμικού με καλώς ορισμένες διεπαφές ανάπτυξης εφαρμογών (APIs) που διευκολύνουν τους προγραμματιστές στη δημιουργία εφαρμογών ελέγχου και διαχείρισης δικτύων. Υποστηρίζει διάφορα πρωτόκολλα για τη διαχείριση δικτυακών συσκευών όπως τα OpenFlow, Netconf κλπ. Στην Εικόνα 10 φαίνεται συνοπτικά η αρχιτεκτονική του Ryu.



Εικόνα 10: Αρχιτεκτονική του Ryu SDN Framework

3.1.1 Λειτουργία σαν Ελεγκτής OpenFlow

Το Ryu μπορεί να λειτουργήσει σαν ελεγκτής OpenFlow και έτσι μπορούν εύκολα να υλοποιηθούν εφαρμογές για τον έλεγχο μεταγωγέων OpenFlow. Υποστηρίζει πλήρως τις εκδόσεις 1.0, 1.2, 1.3, 1.4 και 1.5 (καθώς και τις επεκτάσεις της εταιρείας Nicira).

3.1.2 Λειτουργίες για συνεργασία με υπάρχουσες δικτυακές συσκευές

Είναι πρακτικά πολύ δύσκολο να αντικατασταθούν όλες οι υπάρχουσες συσκευές ενός δικτύου με μεταγωγείς OpenFlow, οπότε θα πρέπει να συνυπάρχουν οι δύο κατηγορίες συσκευών. Έτσι το Ryu παρέχει λειτουργίες και για τις ήδη υπάρχουσες συσκευές, με σκοπό να γίνει πιο ομαλή η εισαγωγή των μεταγωγέων OpenFlow σε ένα δίκτυο. Αυτές οι λειτουργίες είναι :

- **Λήψη πληροφοριών διαχείρισης:** Είναι κοινή πρακτική η συλλογή δεδομένων όπως τα ποσοστά της ροής της κίνησης του δικτύου ή η κατάσταση των συνδέσμων, μέσω συστημάτων παρακολούθησης και διαχείρισης των δικτύων. Το ίδιο είδος πληροφορίας που αφορά τους μεταγωγείς OpenFlow μπορεί να συλλεχθεί μέσω της λειτουργίας ελεγκτή OpenFlow. Το Ryu παρέχει λειτουργία για συλλογή τέτοιων δεδομένων χωρίς να χρειάζεται να χωριστεί το σύστημα διαχείρισης του δικτύου στα δύο (σε περίπτωση που έχουμε και συμβατικές συσκευές αλλά και μεταγωγείς OpenFlow).
- **Διαμόρφωση (Configuration):** Μερικοί διαχειριστές δικτύων χρησιμοποιούν εργαλεία προβλέψεων (provisioning tools) για να υλοποιούν αυτόματες αλλαγές στις δικτυακές συσκευές. Η προδιαγραφή NETCONF (Network Configuration Protocol) έχει σχεδιαστεί για να παρέχει μία κοινή διαδικασία για τέτοιες εργασίες σε συμβατικές συσκευές. Το Ryu περιλαμβάνει λειτουργίες που υλοποιούν το NETCONF.
- **Ανταλλαγή δεδομένων δρομολόγησης:** Το Ryu συμμορφώνεται με τις προδιαγραφές του BGP, οπότε είναι δυνατό για παράδειγμα να γίνει λήψη πληροφοριών δρομολόγησης από συμβατικές συσκευές και να προωθηθεί η πληροφορία αυτή στους μεταγωγείς OpenFlow.

3.2 Mininet

Το Mininet είναι ένας εξομοιωτής δικτύων που δημιουργεί ένα δίκτυο από εικονικούς κόμβους (hosts), μεταγωγείς, ελεγκτές και συνδέσμους. Οι κόμβοι του mininet τρέχουν το πρότυπο λογισμικό δικτύων του Linux, και οι μεταγωγείς του υποστηρίζουν OpenFlow. Πιό συγκεκριμένα, το mininet:

- Παρέχει μία απλή και οικονομική πλατφόρμα δοκιμών για την ανάπτυξη OpenFlow εφαρμογών.
- Καθιστά ικανό το να δουλεύουν πολλοί προγραμματιστές ταυτόχρονα και ανεξάρτητα στην ίδια τοπολογία.
- Καθιστά δυνατή τη δοκιμή περίπλοκων τοπολογιών χωρίς την ανάγκη ενός φυσικού δικτύου.
- Συμπεριλαμβάνει μία γραμμή εντολών για αναζήτηση και εντοπισμό σφαλμάτων ή για εκτέλεση διάφορων δοκιμών.

- Υποστηρίζει τη κατασκευή αυθαίρετων τοπολογιών και διαθέτει επίσης και ένα σύνολο από βασικές παραμετροποιήσιμες τοπολογίες.
- Παρέχει μία απλή και επεκτάσιμη διεπαφή ανάπτυξης εφαρμογών (API) σε γλώσσα Python για δημιουργία δικτύων και πειραματισμό σε αυτά.

Το Mininet χρησιμοποιεί εικονικοποίηση βασισμένη σε διεργασίες (process-based virtualization) ώστε να μπορεί να ξεκινήσει ταυτόχρονα πολλούς κόμβους και μεταγωγείς σε μοναδικό πυρήνα λειτουργικού συστήματος.

3.2.1 Mininet API

Η διεπαφή ανάπτυξης εφαρμογών του mininet διακρίνεται σε τρία επίπεδα :

- **Low-level API:** Αποτελείται από τις βασικές κλάσεις για δημιουργία κόμβων και συνδέσμων. Παραδείγματα των κλάσεων αυτών φαίνονται στην Εικόνα 11.

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

Εικόνα 11: Παράδειγμα χρήσης του low-level API

- **Mid-level API:** Παρέχει το αντικείμενο `Mininet`(Εικόνα 12) που λειτουργεί ως container για τις κλάσεις των κόμβων και των συνδέσμων. Περιέχει επίσης μεθόδους για την προσθήκη νέων κόμβων και συνδέσμων καθώς και για τη διαμόρφωση του δικτύου.

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

Εικόνα 12: Παράδειγμα χρήσης του αντικειμένου Mininet()

- **High-level API:** Προσθέτει την κλάση Topo (Εικόνα 13) που είναι μία γενίκευση προτύπου (template abstraction) που παρέχει τη δυνατότητα δημιουργίας παραμετροποιήσιμων προτύπων τοπολογιών. Αυτά τα πρότυπα μπορούν να περαστούν σαν παράμετροι στην εντολή έναρξης του mininet και έτσι να χρησιμοποιούνται απευθείας από τη γραμμή εντολών.

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

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

Εικόνα 13: Παράδειγμα χρήσης της κλάσης Topo()

4. Επισκόπηση SDX

Το SDX ή αλλιώς «καθορισμένο από λογισμικό» Σημείο Ανταλλαγής πληροφορίας Διαδικτύου (Software Defined internet eXchange point) έχει σαν στόχο τη δημιουργία σημείων ανταλλαγής διαδικτύου (IXPs) που έχουν ένα μεγαλύτερο εύρος πολιτικών δρομολόγησης (όπως για παράδειγμα δρομολόγηση ανάλογα με την εφαρμογή που χρησιμοποιούμε) απ' ότι τα συμβατικά IXPs (τα οποία βασίζονται μόνο σε προώθηση κίνησης ανάλογα με τη διεύθυνση προορισμού). Σχεδιάστηκε από το Εργαστήριο Λειτουργίας Δικτύων και Ασφάλειας Διαδικτύου του πανεπιστημίου του Princeton (http://noise-lab.net/projects/software-defined-networking/sdx/). Η υλοποίησή του, λύνει τα προβλήματα δρομολόγησης που προκύπτουν από τα τρία χαρακτηριστικά του BGP:

- **Δρομολόγηση βασισμένη μόνο στο πρόθεμα διεύθυνσης (IP prefix) προορισμού:** Το BGP δε μπορεί να πάρει αποφάσεις δρομολόγησης βασισμένο στον τύπο της εφαρμογής ή στον αποστολέα.
- **Επιρροή μόνο στους απευθείας γείτονες BGP:** Ένα δίκτυο θα επιλέξει πως να δρομολογήσει το κάθε πακέτο βασισμένο σε διαδρομές που έχει μάθει από τους άμεσους γείτονές του. Οπότε τα δίκτυα έχουν ελάχιστο έλεγχο για ολόκληρη τη διαδρομή (από την πηγή έως τον προορισμό) που θα ακολουθηθεί.
- **Έμμεση έκφραση πολιτικών δρομολόγησης:** Τα δίκτυα βασίζονται σε δυσνόητους μηχανισμούς για να επηρεάσουν την επιλογή της διαδρομής και δε μπορούν να εκφράσουν άμεσα τις προτιμώμενες εισερχόμενες και εξερχόμενες διαδρομές.

4.1 Εφαρμογές του SDX

Το SDX μπορεί να βοηθήσει σε διάφορες εφαρμογές που με τα μέχρι τώρα μέσα ήταν δύσκολες έως αδύνατες. Αυτές είναι :

- **Ανταλλαγή κίνησης για συγκεκριμένες εφαρμογές:** Η ζήτηση για ανταλλαγή συγκεκριμένης κίνησης (application-specific traffic) αυξάνεται συνεχώς από τους παρόχους υπηρεσιών διαδικτύου (ISPs) όπου δύο γειτονικά δίκτυα ανταλλάσσουν πληροφορία μόνο για κάποιες εφαρμογές που αποφασίζουν από κοινού. Αυτή η λειτουργία γίνεται δυνατή με τη χρήση του SDX, καθώς το BGP, από μόνο του, δεν την υποστηρίζει.
- **Διαχείριση εισερχόμενης κίνησης:** Η δρομολόγηση του BGP βασίζεται μόνο στη διεύθυνση προορισμού. Έτσι τα αυτόνομα συστήματα (ASes) έχουν ελάχιστο έλεγχο για το πώς η κίνηση εισέρχεται στο δίκτυό τους και έτσι χρησιμοποιούν έμμεσους και

δυσνόητους μηχανισμούς για να αποκτήσουν περισσότερο έλεγχο. Με την εισαγωγή κανόνων σε έναν μεταγωγέα SDX, ένα δίκτυο μπορεί να ελέγξει άμεσα την εισερχόμενη κίνηση, σύμφωνα με την διεύθυνση IP αποστολέα ή τον αριθμό της πόρτας.

- **Κατανομή φορτίου σε εξυπηρετητές:** Οι πάροχοι πληροφορίας (content providers) κατανέμουν τα αιτήματα των πελατών τους σε συμπλέγματα από εξυπηρετητές παραποιώντας το σύστημα ονομάτων τομέα (domain name system – DNS). Κάθε υπηρεσία έχει ένα και μόνο όνομα τομέα το οποίο αναλύεται σε πολλαπλές διευθύνσεις IP για διάφορους εξυπηρετητές. Όμως η χρήση του DNS για επιλογή των εξυπηρετητών έχει κάποιους περιορισμούς. Αντί αυτού, ο πάροχος μπορεί να θέσει μία μοναδική διεύθυνση IP για μία υπηρεσία και να αλλάζει την διεύθυνση προορισμού των αιτημάτων από πελάτες στο ενδιάμεσο του δικτύου. Το SDX μπορεί να προσφέρει αυτή τη λειτουργία ανακοινώνοντας αυτή τη μοναδική διεύθυνση και αλλάζοντας τη διεύθυνση προορισμού, βασισμένο σε οποιοδήποτε από τα πεδία της επικεφαλίδας του πακέτου (για παράδειγμα στις διευθύνσεις προορισμού ή αποστολέα ή στη πόρτα κλπ.).

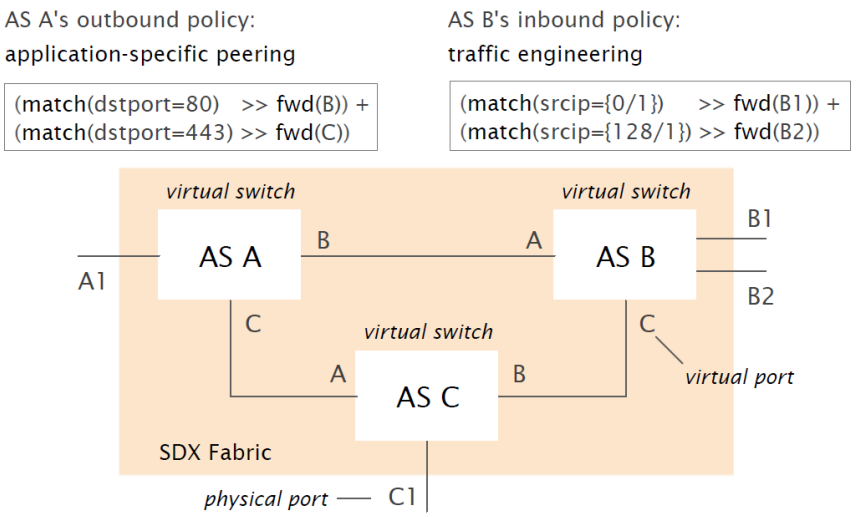
4.2 Λειτουργία του SDX

Το SDX επιτρέπει στους διαχειριστές των συμμετεχόντων αυτόνομων συστημάτων να τρέχουν εφαρμογές που ελέγχουν τη ροή της κίνησης που εισέρχεται και εξέρχεται από τους συνοριακούς δρομολογητές τους ή, στη περίπτωση απομακρυσμένων συμμετεχόντων, τη ροή της κίνησης που προορίζεται για αυτούς. Δίνει σε κάθε συμμετέχοντα τη ψευδαίσθηση ότι έχει το δικό του απομονωμένο εικονικό μεταγωγέα ώστε οι πολιτικές του κάθε αυτόνομου συστήματος να μην επηρεάζουν τις πολιτικές των άλλων και επίσης επιτρέπει τη λήψη αποφάσεων προώθησης βασισμένων στις διαδρομές που μαθαίνονται από το BGP.

4.2.1 Εικονικός SDN μεταγωγέας

Σε ένα παραδοσιακό Σημείο Ανταλλαγής πληροφορίας Διαδικτύου (IXP), το κάθε αυτόνομο σύστημα (AS) που συμμετέχει συνδέει έναν από τους συνοριακούς δρομολογητές του (border router) σε ένα κοινό (για όλα τα AS) δίκτυο επιπέδου δύο (layer-two) καθώς και σε έναν εξυπηρετητή διαδρομών. Σε ένα σημείο SDX, το κάθε αυτόνομο σύστημα μπορεί να τρέξει εφαρμογές SDN που προσδιορίζουν ευέλικτες πολιτικές για απόρριψη, τροποποίηση και προώθηση της κίνησης. Το SDX τότε θα πρέπει να συνδυάσει τις πολιτικές από το κάθε αυτόνομο σύστημα σε μία συνολική πολιτική δρομολόγησης. Σε κάθε αυτόνομο σύστημα δίνεται η ψευδαίσθηση ότι έχει το δικό του εικονικό SDN μεταγωγέα ο οποίος συνδέεται με τους αντίστοιχους εικονικούς μεταγωγείς των γειτονικών συστημάτων, όπως φαίνεται και στην Εικόνα 14. Το αυτόνομο σύστημα A έχει έναν εικονικό μεταγωγέα που συνδέεται με τους εικονικούς μεταγωγείς των B και C. Έτσι το κάθε AS μπορεί να φτιάξει πολιτικές προώθησης

δεδομένων σαν να είναι ο μοναδικός συμμετέχων στο SDX και επίσης δε μπορεί να επηρεάσει το πως τα άλλα AS θα προωθούν τα πακέτα στο δικό τους εικονικό μεταγωγέα.



Εικόνα 14: Αναπαράσταση της έννοιας του εικονικού μεταγωγέα

4.2.2 Πολιτικές προώθησης

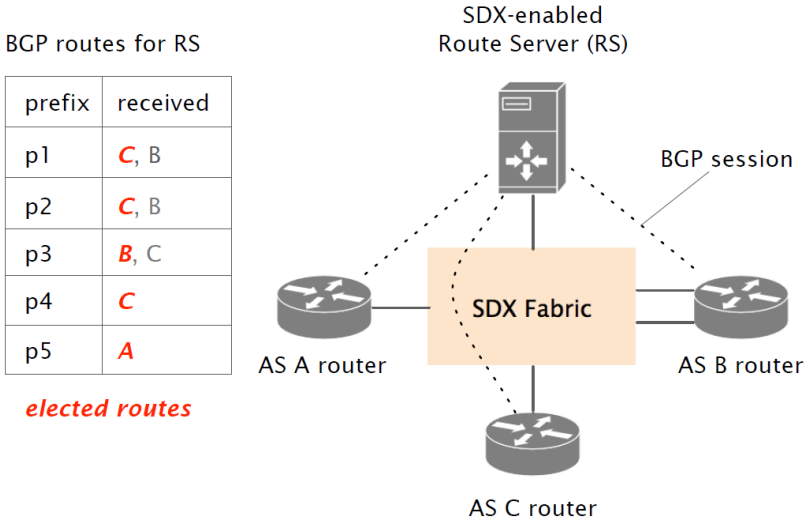
Για τη συγγραφή των πολιτικών προώθησης χρησιμοποιείται η γλώσσα υψηλού επιπέδου Pyretic. Το Pyretic είναι μια γλώσσα συγκεκριμένου τομέα (Domain-Specific Language) για τον προγραμματισμό OpenFlow δικτύων και έχεις ως βάση την γλώσσα προγραμματισμού Python. Πρόκειται για μια υψηλού επιπέδου SDN γλώσσα που σχεδιάστηκε για να λύσει σημαντικά προγραμματιστικά προβλήματα, απλοποιώντας τον τρόπο δημιουργίας και τον τρόπο συνδυασμού διαφορετικών προγραμμάτων για την διαχείριση ενός δικτύου. Δίνει έτσι την δυνατότητα στους προγραμματιστές να εστιάσουν περισσότερο στις πολιτικές που θέλουν να πετύχουν και λιγότερο στον τρόπο υλοποίησής τους.

Από τους συμμετέχοντες, απαιτείται να καθορίσουν εάν μία πολιτική είναι πολιτική εισερχομένων (inbound) ή εξερχομένων (outbound) πακέτων. Οι πολιτικές εισερχομένων αφορούν σε κίνηση που εισέρχεται στον εικονικό μεταγωγέα από την πλευρά ενός άλλου συμμετέχοντα, ενώ οι πολιτικές εξερχομένων αφορούν τη κίνηση που εισέρχεται στον μεταγωγέα από την πλευρά του ίδιου του συμμετέχοντα.

4.2.3 Ενσωμάτωση των πολιτικών με τη δρομολόγηση μέσω BGP

Τα αυτόνομα συστήματα πρέπει να ορίσουν πολιτικές SDX σε σχέση με τις διαφημιζόμενες διαδρομές του συνολικού συστήματος δρομολόγησης. Για να επιτευχθεί αυτό, το SDX επιτρέπει στους συμμετέχοντες να φτιάξουν πολιτικές προώθησης που σχετίζονται με τις

τρέχουσες διαδρομές του BGP, και έτσι, για να διαδοθούν οι διαδρομές αυτές, ο ελεγκτής SDX ενσωματώνει και έναν εξυπηρετητή διαδρομών (route server) όπως φαίνεται και στην Εικόνα 15. Οι συμμετέχοντες αλληλοεπιδρούν με τον εξυπηρετητή αυτό με τον ίδιο τρόπο που θα το έκαναν με έναν συμβατικό εξυπηρετητή διαδρομών. Ο SDX εξυπηρετητής διαδρομών συλλέγει τις διαδρομές που διαφημίζονται από τον κάθε συμμετέχοντα δρομολογητή του BGP, επιλέγει μία καλύτερη διαδρομή για κάθε πρόθεμα διεύθυνσης (prefix) εκ μέρους του κάθε συμμετέχοντα και ξανά διαφημίζει τη διαδρομή στην αντίστοιχη συνεδρία BGP (BGP session). Σε αντίθεση με τους συμβατικούς εξυπηρετητές διαδρομών όπου ο κάθε συμμετέχων μαθαίνει και χρησιμοποιεί μόνο μία διαδρομή ανά πρόθεμα, ο SDX εξυπηρετητής επιτρέπει στους συμμετέχοντες να προωθούν κίνηση σε όλες τις εφικτές διαδρομές για το κάθε πρόθεμα, ακόμα και αν μαθαίνουν μόνο μία.



Εικόνα 15: Αναπαράσταση της λειτουργίας του Route Server του SDX

Το SDX επιτρέπει σε κάθε AS να βασιστεί σε μία προκαθορισμένη πολιτική προώθησης που καθορίζεται από το BGP αντί για να χρειάζεται να καθορίζει πλήρως πολιτική για όλη την κίνησή του. Στην Εικόνα 14 βλέπουμε ότι η εξερχόμενη πολιτική του A εφαρμόζεται μόνο σε κίνηση διαδικτύου (καθώς αφορά τις πόρτες 80 και 443 μόνο), όλη η υπόλοιπη κίνηση ακολουθεί οποιαδήποτε καλύτερη διαδρομή έχει επιλεχθεί από το BGP. Αυτή η λειτουργία απλοποιεί κατά πολύ τη διαδικασία συγγραφής μίας εφαρμογής SDX καθώς ο προγραμματιστής χρειάζεται μόνο να καθορίσει το χειρισμό για συγκεκριμένη κίνηση. Για παράδειγμα, όπως φαίνεται και στην Εικόνα 15 το AS A θα προωθήσει κίνηση που προορίζεται για το πρόθεμα p1 ή p2 (και δεν είναι κίνηση διαδικτύου) προς το AS C αντί για το AS B.

Η ενσωμάτωση του SDX με την υπάρχουσα υποδομή IXP και τα συμβατικά αυτόνομα συστήματα BGP γίνεται ευθέως. Κάθε συμμετέχων, που είναι φυσικά συνδεδεμένος (physically connected) σε ένα μεταγωγέα SDN και ανταλλάσσει διαδρομές BGP με τον SDX εξυπηρετητή

διαδρομών, μπορεί να γράψει SDX πολιτικές προώθησης. Ένα AS μπορεί να επωφεληθεί από μία υλοποίηση SDX ακόμα και αν όλοι οι υπόλοιποι συμμετέχοντες χρησιμοποιούν μόνο τη συμβατική δρομολόγηση BGP.

4.2.4 Μείωση του μεγέθους της τελικής πολιτικής προώθησης

Η επαύξηση των πολιτικών του κάθε συμμετέχοντα με τα προθέματα που έχουν μαθευτεί από το BGP θα μπορούσε να προκαλέσει μία μη επιθυμητή και ραγδαία αύξηση του μεγέθους της τελικής συνολικής πολιτικής καθώς, σύμφωνα με τη παρούσα κατάσταση, το παγκόσμιο σύστημα δρομολόγησης περιέχει πάνω από πεντακόσιες χιλιάδες προθέματα IP. Έτσι με έναν αφελή αλγόριθμο υλοποίησης για το SDX μπορούμε να οδηγηθούμε σε δημιουργία εκατομμυρίων κανόνων προώθησης, πράγμα το οποίο είναι αδύνατον να λειτουργήσει καθώς οι σύγχρονοι μεταγωγείς SDN μπορούν να διαχειριστούν πολύ μικρότερο αριθμό κανόνων. Τα συμβατικά IXP δεν έχουν τέτοιο θέμα επειδή κάνουν προώθηση πακέτων βασιζόμενα μόνο στην διεύθυνση MAC προορισμού αντί για τις κεφαλίδες IP και TCP/UDP. Για την ελαχιστοποίηση του αριθμού των κανόνων στο μεταγωγέα SDN, το SDX πρώτον ομαδοποιεί τα προθέματα με την ίδια συμπεριφορά προώθησης και δεύτερον βάζει ετικέτες (tags) στα πακέτα που στέλνονται από τον κάθε συνοριακό δρομολογητή του κάθε συμμετέχοντα χρησιμοποιώντας μία εικονική διεύθυνση MAC. Αυτή η τεχνική τελικά μειώνει τον αριθμό των κανόνων προώθησης.

Συνήθως η πολιτική ενός AS χειρίζεται ένα μεγάλο αριθμό από προθέματα με τον ίδιο τρόπο. Για παράδειγμα στην Εικόνα 14 το AS A έχει την ίδια συμπεριφορά προώθησης για τα p1 και p2 (δηλαδή αν είναι κίνηση διαδικτύου τότε τα προωθεί στο AS B, αλλιώς στο AS C). Κανονικά χρειαζόμαστε τέσσερις κανόνες, δύο για το κάθε πρόθεμα. Όμως, ομαδοποιώντας τα προθέματα p1 και p2, μπορούμε να ορίσουμε μία πολιτική με μόνο δύο κανόνες. Τα p1 και p2 ανήκουν στην ίδια κλάση προώθησης (Forwarding Equivalence Class – FEC). Μία κλάση προώθησης είναι ένα σύνολο από προθέματα IP που έχουν την ίδια συμπεριφορά προώθησης μέσα στο SDX.

Κατά την επιλογή μίας BGP διαδρομής για ένα πρόθεμα, ένας δρομολογητής αρχικά εξάγει την διεύθυνση IP του επόμενου κόμβου της διαδρομής (next-hop) από το πακέτο της διαφήμισης BGP της διαδρομής. Έπειτα αναζητά στον πίνακα ARP την αντιστοίχηση διεύθυνσης IP σε διεύθυνση MAC, και τελικά εγκαθιστά μία εγγραφή στον πίνακα προώθησης η οποία θέτει σε κάθε πακέτο τη διεύθυνση MAC προορισμού του πριν το κατευθύνει προς την φυσική πόρτα εξόδου. Συνήθως αυτή η διεύθυνση MAC προορισμού αντιστοιχεί στην φυσική διεύθυνση της διεπαφής του επόμενου κόμβου. Στο SDX όμως η διεύθυνση MAC αντιστοιχεί σε μία εικονική διεύθυνση MAC (virtual MAC – VMAC) η οποία χρησιμοποιείται σαν ετικέτα για να ταυτοποιηθεί η κλάση προώθησης (FEC) για το πρόθεμα. Το σύστημα SDX μπορεί τότε απλά να ταιριάξει την VMAC και να χρησιμοποιήσει τους κανόνες προώθησης που αντιστοιχούν στην κλάση. Στο συνοριακό δρομολογητή του αυτόνομου συστήματος στέλνεται μέσω του BGP μία

διεύθυνση IP σαν διεύθυνση επόμενου κόμβου η οποία ονομάζεται διεύθυνση εικονικού κόμβου (virtual next hop – VNH). Πρακτικά το SDX αφού υπολογίσει, ανάλογα με τις πολιτικές, τις κλάσεις προώθησης, αναθέτει σε κάθε μία από αυτές από ένα μοναδικό ζευγάρι διευθύνσεων VNH και VMAC. Τότε μετατρέπει τις πολιτικές ώστε να ταιριάζουν με τη διεύθυνση VMAC αντί για το πρόθεμα προορισμού και τελικά δίνει την εντολή στον εξυπηρετητή διαδρομών ώστε να θέσει την διεύθυνση VNH στα αντίστοιχα μηνύματα του BGP που στέλνει και επίσης κατευθύνει τον δικό του εξυπηρετητή ARP ώστε να απαντάει στα αιτήματα ARP για μία VNH διεύθυνση IP με την αντίστοιχη VMAC.

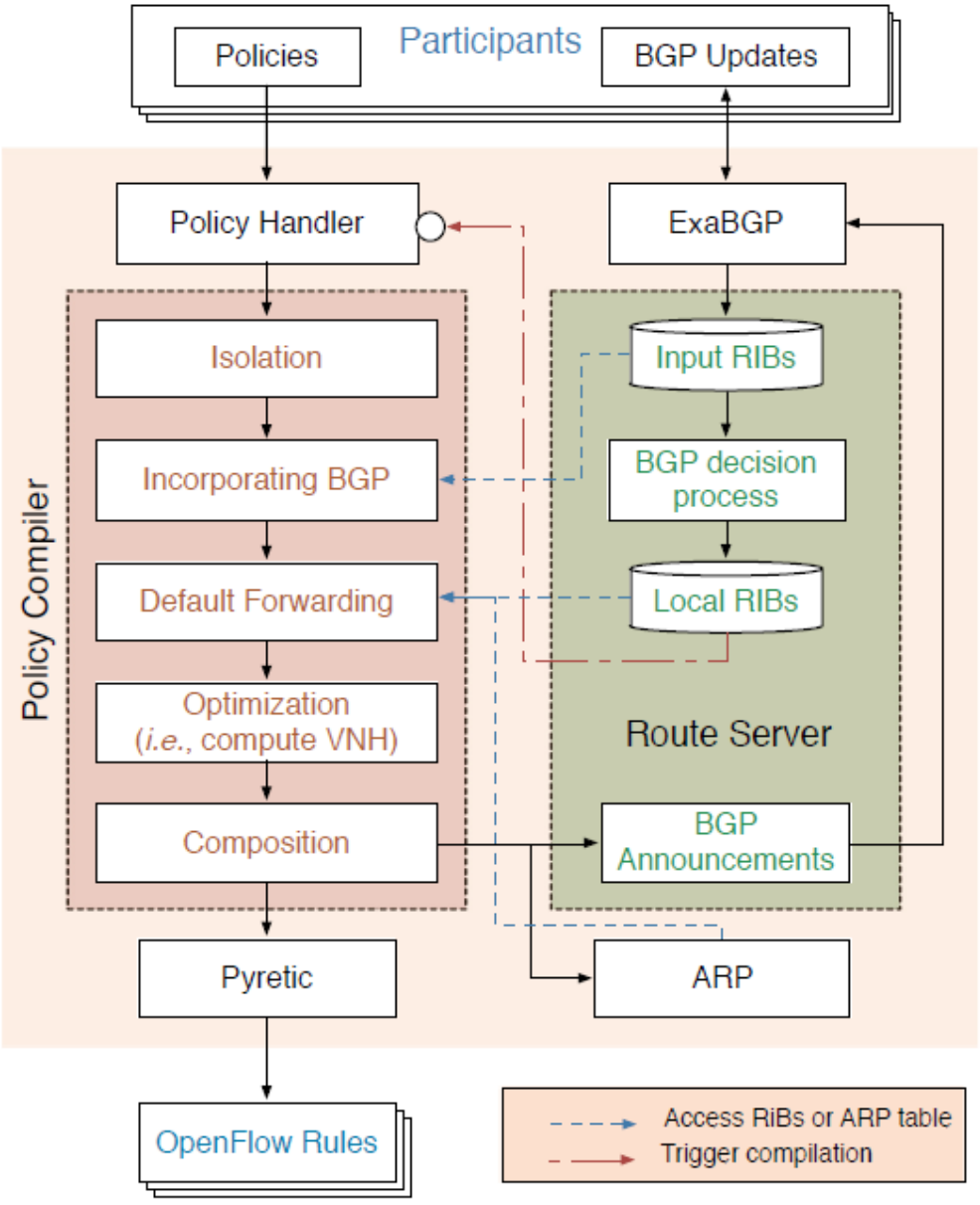
Ο υπολογισμός της διεύθυνσης VNH απαιτεί την αναγνώριση όλων των ομάδων από προθέματα που έχουν την ίδια συμπεριφορά προώθησης, λαμβάνοντας υπόψιν ταυτόχρονα και την λειτουργία προώθησης του BGP και τις πολιτικές SDX του κάθε αυτόνομου συστήματος. Για να εξασφαλιστεί το γεγονός ότι δύο προθέματα που έχουν την ίδια συμπεριφορά θα είναι πάντα στην ίδια κλάση FEC, το SDX υπολογίζει τις κλάσεις σε τρία στάδια:

- Στο **πρώτο** στάδιο δημιουργούνται οι ομάδες προθεμάτων για τα οποία η συμπεριφορά τους επηρεάζεται με τον ίδιο τρόπο έστω και από μία πολιτική εξερχομένων (outbound) πακέτων. Στις Εικόνες 14 και 15 φαίνεται ότι η προκαθορισμένη συμπεριφορά προώθησης της ομάδας προθεμάτων {p1, p2, p3} παρακάμπτεται από την outbound πολιτική του AS A, που προωθεί την κίνηση για το διαδίκτυο στο AS B. Ομοίως, για την ομάδα {p1, p2, p3, p4} έχουμε παράκαμψη λόγω της πολιτικής για τα πακέτα που αφορούν την πόρτα 443 που στέλνονται στο AS C. Οπότε το μόνο πρόθεμα που δεν παρακάμπτεται η προκαθορισμένη συμπεριφορά του είναι το p5.
- Στο **δεύτερο** στάδιο, το SDX ομαδοποιεί τα προθέματα των οποίων η προκαθορισμένη συμπεριφορά τους παρακάμφθηκε σύμφωνα με τον επόμενο κόμβο(next-hop) που έχει επιλεχθεί από τον εξυπηρετητή διαδρομών. Έτσι στο παράδειγμά μας, τα προθέματα p1, p2, p3, p4 θα χωριστούν σε δύο ομάδες, {p1, p2, p4} όπου ο προκαθορισμένος next-hop είναι το AS C και {p3} με next-hop το AS B αντίστοιχα.
- Στο **τρίτο** και τελευταίο στάδιο έχουμε αρχικά την ένωση των ομάδων από τα πρώτα δύο στάδια σε μία ομάδα G = {{p1, p2, p3}, {p1, p2, p3, p4}, {p1, p2, p4}, {p3}}. Η ομάδα G τότε μετατρέπεται στην ομάδα G’ = {{p1, p2}, {p3}, {p4}} που είναι το ελάχιστο ασυνεχές υποσύνολο (minimum disjoint subset) του G. Η ομάδα G’ είναι και ο τελικός υπολογισμός για τις κλάσεις FEC.

Παρατηρούμε ότι στο παραπάνω παράδειγμα, το πρόθεμα p5 διατηρεί την προκαθορισμένη συμπεριφορά προώθησης που έχει. Για αυτά τα προθέματα δε χρειάζονται επιπλέον υπολογισμοί από το SDX και έτσι τους συμπεριφέρεται σαν ένα απλό IXP.

4.3 Υλοποίηση του SDX

Στην Εικόνα 16 φαίνεται η υλοποίηση του ελεγκτή SDX η οποία αποτελείται από δύο κύριες διεργασίες. Το μεταγλωττιστή πολιτικών (policy compiler) που είναι βασισμένος στη γλώσσα Pyretic και τον εξυπηρετητή διαδρομών (route server) που βασίζεται στο ExaBGP. Το ExaBGP είναι ένα SDN εργαλείο γραμμένο στη γλώσσα python που χρησιμοποιείται για να δημιουργεί επιθυμητή κίνηση BGP χωρίς όμως να χρειάζεται μεγάλο αριθμό πόρων συστήματος όπως θα έκανε μία πλήρης υλοποίηση BGP.



Εικόνα 16: Διάγραμμα υλοποίησης του SDX

4.3.1 SDX policy compiler

Ο μεταγλωττιστής πολιτικών είναι μία διεργασία του SDX που μετατρέπει τις πολιτικές σε κανόνες προώθησης. Βασισμένος στην ψευδαίσθηση του SDX με τους εικονικούς μεταγωγείς που είδαμε στο 4.2.1, απομονώνει τις πολιτικές που γράφει ο κάθε συμμετέχων. Έπειτα, περιορίζει τις εξερχόμενες πολιτικές σύμφωνα με τις πληροφορίες δρομολόγησης του BGP που παίρνει από τον εξυπηρετητή διαδρομών και έτσι ξαναγράφει τις πολιτικές των συμμετεχόντων ώστε ο κάθε εικονικός μεταγωγέας να μπορεί να προωθήσει την κίνηση ανάλογα με τις προκαθορισμένες πολιτικές του BGP. Στη συνέχεια κάνει τον υπολογισμό των εικονικών επόμενων κόμβων (VNH) για τα προθέματα και αναβαθμίζει τις πολιτικές του κάθε συμμετέχοντα όπου είναι απαραίτητο ακολουθώντας τη διαδικασία που περιεγράφηκε στο 4.2.4. Τέλος ενώνει όλες τις πολιτικές σε μία συνολική και έτσι δημιουργεί τους αντίστοιχους κανόνες προώθησης των πακέτων.

4.3.2 SDX route server

Ο εξυπηρετητής διαδρομών του SDX, όπως και οι παραδοσιακοί εξυπηρετητές διαδρομών, λαμβάνει από το κάθε AS τις BGP διαφημίσεις των διαδρομών και υπολογίζει το βέλτιστο μονοπάτι για κάθε προορισμό για τον κάθε συμμετέχοντα. Καθιστά επίσης δυνατή την ενσωμάτωση των πολιτικών των συμμετεχόντων με τη δρομολόγηση μέσω του BGP παρέχοντας πληροφορίες για διαδρομές από το BGP στην διεργασία του μεταγλωττιστή. Τέλος όταν έχουμε μία αλλαγή στις διαδρομές του BGP ή μία νέα προσθήκη, ο εξυπηρετητής ξανά υπολογίζει τις πολιτικές.

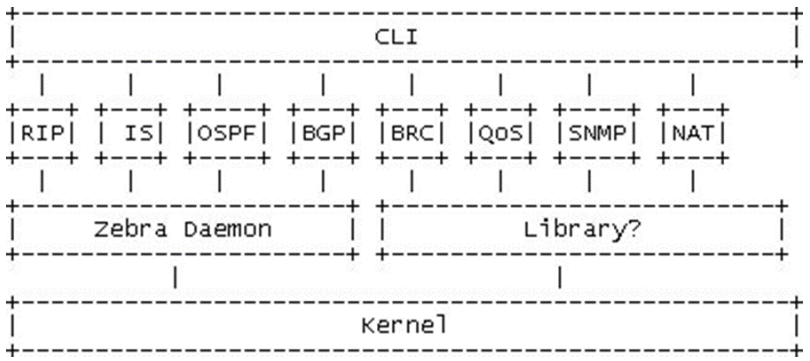
5. Σύστημα Υλοποίησης IXP με διεπαφές LACP

5.1 Εργαλεία για την προσομοίωση

Στην ενότητα αυτή παρουσιάζονται όλα τα εργαλεία που χρησιμοποιήθηκαν για το Σύστημά μας καθώς και οι αλλαγές-προσθήκες που έγιναν στο καθένα από αυτά. Τα εργαλεία που χρησιμοποιήθηκαν για τη δημιουργία της εικονικής τοπολογίας είναι το Quagga και το miniNExT (που βασίζεται στο mininet). Για το SDX, χρησιμοποιήθηκε ο sdx ελεγκτής βασισμένος στο Ryu SDN Framework που μπορεί να βρεθεί στη διεύθυνση https://github.com/sdn-ixp/sdx-ryu. Στον ελεγκτή αυτό, έγιναν προσθήκες για την υποστήριξη διεπαφών LACP οι οποίες παρουσιάζονται στο **παράρτημα κώδικα**.

5.1.1 Quagga

Το Quagga είναι ένα λογισμικό δρομολόγησης ανοιχτού κώδικα το οποίο αναπαριστά την λειτουργία ενός πραγματικού δρομολογητή (router). Η αρχιτεκτονική του περιλαμβάνει έναν πυρήνα δαίμονα (daemon), τον zebra, ο οποίος λειτουργεί ως ένα στρώμα αφαίρεσης στο υποκείμενο πυρήνα Unix (kernel) και παρουσιάζει το Zserv API πάνω από ένα Unix ή TCP ρεύμα (stream) σε Quagga πελάτες (clients). Χρησιμοποιήθηκε καθώς προσφέρει τη δυνατότητα δρομολόγησης πακέτων IP και πιο συγκεκριμένα για τις ανάγκες του συστήματος πακέτα BGP.



Εικόνα 17: Η δομή ενός συστήματος που τρέχει Quagga

5.1.2 miniNExT

Στην περίπτωσή μας, χρειαζόμασταν μόνο δρομολογητές στο δίκτυό μας και έτσι, επιλέξαμε να χρησιμοποιήσουμε μια επέκταση του Mininet, το miniNExT. Το miniNExT (Mininet ExTended) είναι ένα επίπεδο επέκτασης το οποίο διευκολύνει την δημιουργία σύνθετων δικτύων στο Mininet.

Το γεγονός ότι το MiniNExT είχε ένα πρότυπο τρόπο για την χρήση του μαζί με το Quagga συνέβαλλε στην επιλογή του. Επιπλέον, σημαντικό είναι ότι το MiniNExT χρησιμοποιεί εικονικοποίηση που στηρίζεται στις διεργασίες (process-based virtualization) με αποτέλεσμα να έχουμε την δυνατότητα να χρησιμοποιήσουμε πάνω από έναν δρομολογητές χωρίς τον φόβο ότι θα χτυπάνε όλοι στην ίδια θύρα.

Το MiniNExT περιλαμβάνει πιο περίπλοκους κόμβους οι οποίοι χρησιμοποιούνται σε πολλά πειραματικά δίκτυα όπως:

- Μηχανές δρομολόγησης (Quagga και Bird)
- Εξυπηρετητές (BIND και Apache)
- Εξαρτήματα συνδεσιμότητας (OpenVPN, κ.α.)
- NAT και εξαρτήματα διαχείρισης δικτύου (DHCP, κ.α.)

Στην περίπτωσή μας, παρέχει συγκεκριμένο τρόπο για την δημιουργία ενός δρομολογητή, τρέχοντας το Quagga σε έναν κόμβο και εισάγοντάς του τα απαραίτητα αρχεία ρυθμίσεων (configuration files) τα οποία αναφέρθηκαν στη προηγούμενη ενότητα. Ένα άλλο πλεονέκτημα του MiniNExT είναι ότι η λειτουργία του Quagga ξεκινά αυτόματα κατά το χτίσιμο του δικτύου με αποτέλεσμα εξοικονόμηση χρόνου σε σχέση με το στήσιμο του Quagga σε κάθε κόμβο-δρομολογητή κάθε φορά που χτίζεται το δίκτυο.

5.1.2.1 Τροποποίηση miniNExT

Για τις ανάγκες της εργασίας, υλοποιήσαμε δύο συναρτήσεις για τη δημιουργία πολλαπλών διασυνδέσεων μεταξύ των κόμβων και του μεταγωγέα.

```
def addMultiLinks ( net, x, switch):
    portNo = 0

    print "Adding " + x + " Links with the switch for each host\n\n"
    for host in net.hosts:
        for i in range (0,x)
        portNo = portNo + 1
        net.addLink( host, switch, port2 = portNo)
```

Εικόνα 18: Η συνάρτηση προσθήκης πολλαπλών link για το mininet/mininext

Η συνάρτηση addMultiLinks() δέχεται σαν ορίσματα την τοπολογία (net) για να εξάγει από εκεί τους κόμβους που θα συνδεθούν με πολλαπλές διεπαφές, τον αριθμό (x) των διεπαφών που θέλουμε να δημιουργήσουμε καθώς και τον μεταγωγέα (switch) και στη συνέχεια δημιουργεί τις συνδέσεις με αυτόν.

```
def addBondingInterfaces( net ):

    print "Configuring Link Aggregation\n\n"
    for host in net.hosts:
        print "Host name: ", host.name
            host.cmd('modprobe bonding')
            host.cmd('ip link add %s-bond0 type bond' % host.name)
            host.cmd('ip link set bond0 address 00:00:00:00:00:%s' % host.name)
            host.cmd('ip link set %s-eth0 down' % host.name)
            host.cmd('ip link set %s-eth0 address 00:00:00:00:%s:10' % (host.name, host.name))
            host.cmd('ip link set %s-eth0 master bond0' % host.name)
            host.cmd('ip link set %s-eth1 down' % host.name)
            host.cmd('ip link set %s-eth1 address 00:00:00:00:%s:11' % (host.name, host.name))
            host.cmd('ip link set %s-eth1 master bond0' % host.name)
            host.cmd('ip link set bond0 up')
```

Εικόνα 19: Η συνάρτηση ενεργοποίησης Linux Bonding interface σε κάθε κόμβο

Η συνάρτηση addBondingInterfaces() που έχει σαν σκοπό την εισαγωγή των εντολών σε κάθε κόμβο για τη διαμόρφωση και ενεργοποίηση των διεπαφών του Linux Bonding.

5.2 SDX Controller βασισμένος σε Ryu

Το Εργαστήριο Λειτουργίας Δικτύων (που σχεδίασε το SDX όπως αναφέραμε και στο Κεφάλαιο 4) έχει υλοποιήσει δύο εκδόσεις για τον ελεγκτή SDX. Η πρώτη, που είναι και η κύρια έκδοσή του, έχει υλοποιηθεί με τη χρήση της γλώσσας υψηλού επιπέδου Pyretic και λειτουργεί με τον ελεγκτή POX. Η δεύτερη έκδοση χρησιμοποιεί μόνο το Ryu Framework. Η κάθε έκδοση έχει τα δικά της πλεονεκτήματα και μειονεκτήματα.

Η υλοποίηση που βασίζεται στο Ryu είναι πιο εύκολη στη σύνταξη των πολιτικών που προσφέρει (καθώς είναι σε απλή μορφή json), όμως υποστηρίζει ακόμα ένα πολύ μικρό σύνολο από κανόνες ταιριάσματος. Συγκεκριμένα, υποστηρίζει μόνο ταίριασμα διευθύνσεων IPv4 πηγής και προορισμού καθώς και ταίριασμα των αντίστοιχων πορτών (TCP ή UDP). Από την άλλη πλευρά, η έκδοση με το Pyretic υποστηρίζει το σύνολο των κανόνων ταιριάσματος που υποστηρίζει και η γλώσσα Pyretic. Όμως η τελευταία, δε μπορεί να δεχτεί προσθήκη για υποστήριξη της τεχνολογίας 802.3ad, καθώς αυτό δεν το υποστηρίζει και ο ελεγκτής POX.

Οι πιθανές λύσεις που είχαμε ήταν δύο. Η πρώτη αφορούσε στην συγγραφή νέου ελεγκτή βασισμένου σε Ryu (το οποίο υποστηρίζει την 802.3ad) για χρήση με την έκδοση του Pyretic, ενώ η δεύτερη λύση ήταν η προσθήκη υποστήριξης της τεχνολογίας 802.3ad κατευθείαν στην υλοποίηση της έκδοσης που βασίζεται στο Ryu. Η δεύτερη λύση ήταν και αυτή που επιλέχθηκε.

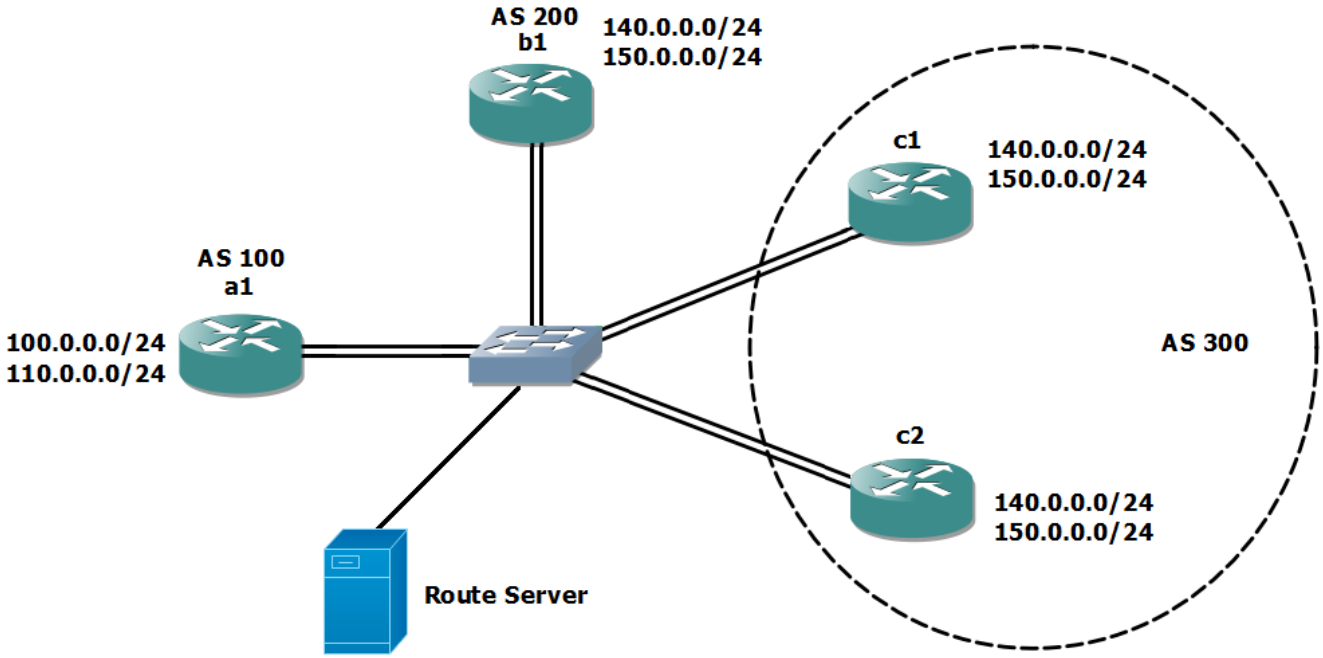
5.2.1 Παραμετροποίηση του Ryu SDX Controller

Για να ορίσουμε την τοπολογία που θέλουμε να βλέπει το SDX στο σημείο IXP πρέπει να φτιάξουμε δύο αρχεία json, ένα που έχει τις πληροφορίες για τον κάθε συμμετέχοντα στο IXP και ένα που περιέχει την τοποθεσία που βρίσκονται τα αρχεία των πολιτικών του κάθε συμμετέχοντα. Το πρώτο αρχείο είναι το *sdx_global.cfg* το οποίο περιέχει πληροφορίες όπως οι διεπαφές απ' όπου συνδέεται στο SDX ο κάθε δρομολογητής του κάθε συμμετέχοντα, το πλήθος των φυσικών διεπαφών που θα ενεργοποιηθεί το LACP, οι διευθύνσεις MAC και IP της σύνδεσης καθώς και το νούμερο του αυτόνομου συστήματος του συμμετέχοντα. Το άλλο αρχείο, *sdx_policies.cfg* , περιέχει το μονοπάτι συστήματος (system path) όπου είναι αποθηκευμένα τα αρχεία των πολιτικών για κάθε συμμετέχοντα. Ένα τρίτο αρχείο, *bgp.cfg*, συμπληρώνει την αρχικοποίηση του SDX το οποίο περιέχει τις οδηγίες για την παραμετροποίηση του route server μέσω του ExaBGP. Τα τρία αυτά αρχεία βρίσκονται στο **παράρτημα κώδικα**.

5.3 Χρήση του Συστήματος

5.3.1 Τοπολογία

Για την παρουσίαση του συστήματος καθώς και των διάφορων πολιτικών δρομολόγησης εισερχόμενων και εξερχόμενων πακέτων, υλοποιήθηκε με τη βοήθεια του miniNExT και του Quagga η παρακάτω τοπολογία :



Εικόνα 20: Τοπολογία IXP με πολλαπλά link

Στην Εικόνα 20 φαίνεται ένας μεταγωγέας IXP ο οποίος έχει συνδεδεμένους πάνω του τέσσερις δρομολογητές (a1, b1, c1 και c2) και έναν εξυπηρετητή διαδρομών. Ο δρομολογητής a1 είναι μέλος του αυτόνομου συστήματος 100, ο b1 του αυτόνομου συστήματος 200 και οι c1 και c2 του 300. Όμως, αν και σε διαφορετικά αυτόνομα συστήματα, οι δρομολογητές b1, c1 και c2 διαφημίζουν στο BGP τις δύο ίδιες ομάδες διευθύνσεων IP 140.0.0.0/24 και 150.0.0.0/24. Σκοπός του παραδείγματος αυτού είναι να δείξει ότι με τις κατάλληλες πολιτικές, το SDX μπορεί να ξεχωρίσει ποιά πακέτα πρέπει να δρομολογηθούν προς το b1 και αντίστοιχα ποιά προς το c1 ή το c2 δεδομένου ότι έχουν την ίδια διεύθυνση IP.

5.3.2 Πολιτικές Δρομολόγησης

Ενδεικτικά στους δύο παρακάτω πίνακες φαίνονται παραδείγματα από μία πολιτική προώθησης εξερχομένων πακέτων (outbound) και μία αντίστοιχη για εισερχόμενα πακέτα (inbound). Η δομή συγγραφής των πολιτικών βασίζεται στη μορφή json, που είναι μία μορφή ανταλλαγής δεδομένων (data-interchange format) η οποία είναι ανοιχτού προτύπου (open standard) και εύκολη στη γραφή και στην ανάγνωσή της. Η πρώτη πολιτική (outbound) αφορά στο AS 100 (της Εικόνας 20) και επιβάλλει στα πακέτα που προορίζονται για την πόρτα TCP 80 να προωθούνται προς το AS 200 ενώ στα πακέτα για τις TCP πόρτες 4321 και 4322 προς το AS 300. Η δεύτερη πολιτική (inbound) είναι για το AS 300 και δίνει οδηγία στο SDX τα πακέτα που προορίζονται για την πόρτα TCP 4321 να προωθούνται στον δρομολογητή c1 ενώ τα πακέτα που αφορούν την πόρτα P 4322 να προωθούνται στον δρομολογητή c2 αντίστοιχα.

Outbound Policy:

```
{    "outbound": [
        {
            "match": { "tcp_dst": 80 },
            "action": { "fwd": 2 }
        },
        {
            "match": { "tcp_dst": 4321 },
            "action": { "fwd": 3 }
        },
        {
            "match": { "tcp_dst": 4322 },
            "action": { "fwd": 3 }
        }
    ]
}
```

Inbound Policy:

```
{    "inbound": [
        {
            "match": { "tcp_dst": 4321 },
            "action": { "fwd": 0 }
        },
        {
            "match": { "tcp_dst": 4322 },
            "action": { "fwd": 1 }
        }
    ]
}
```

5.3.3 Εκτέλεση

5.3.3.1 Έναρξη της τοπολογίας

Όπως αναφέραμε στα προηγούμενα κεφάλαια, για την προσομοίωση της λειτουργίας του Συστήματος που υλοποιήθηκε για αυτή την εργασία, χρησιμοποιήθηκε ο εξομοιωτής Mininet. Για την έναρξη της τοπολογίας από το αρχείο *sdx_mininext.py* εκτελούμε την εντολή :

```
sudo ./sdx_mininext.py
```

```
*** Creating network
*** Adding controller
*** Adding hosts:
a1 b1 c1 c2
*** Adding switches:

*** Adding links:

*** Configuring hosts
*** Starting host services
a1: Quagga (OK)
b1: Quagga (OK)
c1: Quagga (OK)
c2: Quagga (OK)
** adding links
** Creating ExaBGP host
** Starting the network
*** Starting controller
*** Starting 1 switches
s1
**Adding Network Interfaces for SDX Setup
Configuring participating ASs


Host name:  a1
Host name:  b1
Host name:  c1
Host name:  c2
Host name:  exabgp
** Running CLI
*** Starting CLI:
mininext>
```

Αφού δημιουργηθεί η τοπολογία και ξεκινήσουν όλοι οι κόμβοι-δρομολογητές, τότε μπορούμε προαιρετικά να επιβεβαιώσουμε την σωστή δημιουργία των διεπαφών με την εντολή του mininet :

```
mininet> net
```

```
mininext> net
a1 a1-eth0:s1-eth1 a1-eth1:s1-eth2
b1 b1-eth0:s1-eth3 b1-eth1:s1-eth4
c1 c1-eth0:s1-eth5 c1-eth1:s1-eth6
c2 c2-eth0:s1-eth7 c2-eth1:s1-eth8
exabgp exabgp-eth0:s1-eth9
s1 lo:  s1-eth1:a1-eth0 s1-eth2:a1-eth1 s1-eth3:b1-eth0 s1-eth4:b1-eth1 s1-eth5:c1-eth0
s1-eth6:c1-eth1 s1-eth7:c2-eth0 s1-eth8:c2-eth1 s1-eth9:exabgp-eth0
c0
mininext>
```

Με την εντολή “`a1 ifconfig`“ μπορούμε να επιβεβαιώσουμε την δημιουργία και τη σωστή λειτουργία της bond διεπαφής στον δρομολογητή a1 (και αντίστοιχα μετατρέπουμε την εντολή για έλεγχο σε όλους τους δρομολογητές).

```
mininext> a1 ifconfig
a1-eth0   Link encap:Ethernet  HWaddr 08:00:27:89:3b:9f
          inet6 addr: fe80::a00:27ff:fe89:3b9f/64 Scope:Link
          UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:56 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:648 (648.0 B)  TX bytes:4276 (4.2 KB)

a1-eth1   Link encap:Ethernet  HWaddr 08:00:27:89:3b:9f
          inet6 addr: fe80::a00:27ff:fe89:3b9f/64 Scope:Link
          UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:6 overruns:0 frame:0
          TX packets:26 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:648 (648.0 B)  TX bytes:2800 (2.8 KB)

bond0     Link encap:Ethernet  HWaddr 08:00:27:89:3b:9f
          inet addr:172.0.0.1  Bcast:172.0.255.255  Mask:255.255.0.0
          inet6 addr: fe80::4c7e:56ff:fea9:3243/64 Scope:Link
          UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
          RX packets:16 errors:0 dropped:6 overruns:0 frame:0
          TX packets:82 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1296 (1.2 KB)  TX bytes:7076 (7.0 KB)
```

5.3.3.2 Έναρξη του SDX Controller

Για την έναρξη του ελεγκτή SDX χρησιμοποιούμε τον διαχειριστή εφαρμογών (application manager) της πλατφόρμας του Ryu. Για να το τρέξουμε πληκτρολογούμε:

```
ryu-manager ~/sdx-ryu/ctrl/asdx.py --asdx-dir diploma
```

```
vagrant@sdx-ryu:~/sdx-ryu/ctrl$ ryu-manager ~/sdx-ryu/ctrl/asdx.py --asdx-dir diploma
loading app /home/vagrant/sdx-ryu/ctrl/asdx.py
loading app ryu.controller.ofp_handler
creating context wsgi
instantiating app None of LacpLib
creating context lacplib
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app /home/vagrant/sdx-ryu/ctrl/asdx.py of aSDX
(6741) wsgi starting up on http://0.0.0.0:8080/
```

Σε ένα άλλο παράθυρο γραμμής εντολών ξεκινάμε τον εξυπηρετητή διαδρομών του SDX και σε ένα τρίτο παράθυρο ξεκινάμε το ExaBGP. Οι εντολές αντίστοιχα είναι :

```
sudo ~/sdx-ryu/xrs/route_server.py diploma
```

και

```
exabgp ~/sdx-ryu/examples/diploma/controller/sdx_config/bgp.conf
```

```
vagrant@sdx-ryu:~/sdx-ryu/xrs$ sudo ./route_server.py diploma
Initialize the Route Server
Start Server
```

5.3.3.3 Δοκιμή λειτουργίας

Αφού έχουμε εκτελέσει τις απαραίτητες ενέργειες για να ξεκινήσει ο εξομοιωτής αλλά και ο ελεγκτής SDX, θα πρέπει να κάνουμε δοκιμές για να επιβεβαιώσουμε την ορθή λειτουργία του IXP.

Αρχικά ελέγχουμε αν όλοι οι δρομολογητές έχουν πάρει τις απαραίτητες διαφημίσεις BGP από τον εξυπηρετητή διαδρομών με την εντολή mininet :

```
<router_name> route –n
```

(Παρατηρούμε στη στήλη Gateway ο κάθε πίνακας έχει τις διευθύνσεις IP του εικονικού next-hop (VNH))

```
mininext> a1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
140.0.0.0       172.0.1.3       255.255.255.0   UG    0      0        0 bond0
150.0.0.0       172.0.1.4       255.255.255.0   UG    0      0        0 bond0
172.0.0.0       0.0.0.0         255.255.0.0     U     0      0        0 bond0
mininext> b1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
100.0.0.0       172.0.1.2       255.255.255.0   UG    0      0        0 bond0
110.0.0.0       172.0.1.1       255.255.255.0   UG    0      0        0 bond0
172.0.0.0       0.0.0.0         255.255.0.0     U     0      0        0 bond0
mininext> c1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
100.0.0.0       172.0.1.2       255.255.255.0   UG    0      0        0 bond0
110.0.0.0       172.0.1.1       255.255.255.0   UG    0      0        0 bond0
172.0.0.0       0.0.0.0         255.255.0.0     U     0      0        0 bond0
mininext>
```

Οι παραπάνω πίνακες επιβεβαιώνουν ότι οι διαφημίσεις BGP έχουν διαδοθεί σωστά. Στη συνέχεια παρατηρούμε ότι και ο ελεγκτής SDX λαμβάνει κανονικά τα πακέτα LACP από την κάθε διεπαφή.

```
. . . .
[LACP][INFO] SW=0000000000000001 PORT=1 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=1 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=3 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=3 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=4 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=4 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=5 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=5 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=6 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=6 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=7 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=7 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=8 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=8 LACP sent.
. . . .
```

Εδώ φαίνεται η λήψη των πακέτων και η αποστολή των απαντήσεων LACP. Για να δοκιμάσουμε ότι έχουν εγκατασταθεί σωστά και οι κανόνες προώθησης που προκύπτουν από όλες τις πολιτικές, χρησιμοποιούμε την εφαρμογή iperf που είναι ένα εργαλείο που μας επιτρέπει να δημιουργήσουμε κίνηση με συγκεκριμένες ιδιότητες. Το iperf αποτελείται από έναν εξυπηρετητή και έναν πελάτη. Ενδεικτικά δοκιμάζουμε την πολιτική που στέλνει τα πακέτα που προορίζονται για την πόρτα TCP 4322 από το AS 100 στο δρομολογητή c2 του AS 300.

Αρχικά ξεκινάμε στον δρομολογητή c2 τον εξυπηρετητή με την εντολή :

```
c2 iperf -s -B 140.0.0.1 -p 4322 &
```

```
mininext> c2 iperf -s -B 140.0.0.1 -p 4322 &
------------------------------------------------------------
Server listening on TCP port 4322
Binding to local address 140.0.0.1
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
mininext>
```

Έπειτα ξεκινάμε τον πελάτη στο δρομολογητή a1 αντίστοιχα με την εντολή :

```
a1 iperf -c 140.0.0.1 -B 100.0.0.1 -p 4322 -t 2
```

```
mininext> a1 iperf -c 140.0.0.1 -B 100.0.0.1 -p 4322 -t 2
------------------------------------------------------------
Client connecting to 140.0.0.1, TCP port 4322
Binding to local address 100.0.0.1
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 100.0.0.1 port 4322 connected with 140.0.0.1 port 4322
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 2.0 sec  1.53 GBytes  6.59 Gbits/sec
```

Δεδομένου ότι ο πελάτης καταφέρνει να επικοινωνήσει με τον εξυπηρετητή, σημαίνει ότι η μεταγλώττιση των πολιτικών ήταν επιτυχής.

5.3.3.4 Ανάκαμψη σε περίπτωση σφάλματος

Για να προσομοιώσουμε την περίπτωση σφάλματος, σε ένα δρομολογητή εντοπίζουμε την φυσική διεπαφή από την οποία στέλνει τα πακέτα για ένα πρόθεμα. Τότε την απενεργοποιούμε ώστε να αναγκάσουμε το LACP να απευθυνθεί στην διεπαφή που έχει μείνει για να στείλει από εκεί τα πακέτα.

Για την απενεργοποίηση της μίας διεπαφής χρησιμοποιούμε την εντολή:

```
<host_name> ip link set <interface_name> nomaster
```

```
mininext> a1 ip link set a1-eth0 nomaster
mininext>
```

Παρατηρούμε ότι μετά την απενεργοποίηση της μίας διεπαφής, ο ελεγκτής απενεργοποιεί και το αντίστοιχο άκρο της σύνδεσης στη πλευρά του μεταγωγέα.

```
. . . .
[LACP][INFO] SW=0000000000000001 PORT=1 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=1 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP received.
[LACP][INFO] SW=0000000000000001 PORT=2 LACP sent.
[LACP][INFO] SW=0000000000000001 PORT=3 LACP received.
. . . .
. . . .
[LACP][INFO] SW=0000000000000001 PORT=1 LACP exchange timeout has occurred.
slave state changed port: 1 enabled: False
. . . .
```

Πλέον ο πελάτης iperf θα αποτύχει να συνδεθεί με τον εξυπηρετητή. Όμως αυτή η συμπεριφορά θα αλλάξει μόλις ο χρονομετρητής (timer) του LACP λήξει. Έτσι η φυσική διεπαφή που έχει απομείνει θα πάρει τη θέση αυτής που δε λειτουργεί και έτσι θα αποκατασταθεί και η σύνδεση (και πλέον ο πελάτης iperf θα μπορεί να ξανά συνδεθεί με τον εξυπηρετητρή).

```
mininext> a1 iperf -c 140.0.0.1 -B 100.0.0.1 -p 4322 -t 2
connect failed: Connection refused.
mininext>
```

6. Συμπεράσματα - Επεκτάσεις

Με την ολοκλήρωσή της, η παρούσα εργασία συμβάλλει στην προσπάθεια που γίνεται από την κοινότητα του SDN για τη διάδοση της φιλοσοφίας του σε εφαρμογές που έχουν σχέση με τους Πάροχους Υπηρεσιών Διαδικτύου αλλά και με εταιρείες μεγάλης κλίμακας. Υπάρχουν αρκετές δυνατότητες επέκτασης της.

Αρχικά προτείνεται η επέκταση και της βασικής υλοποίησης του SDX στη γλώσσα Pyretic ώστε να υποστηρίζει το πρωτόκολλο LACP. Ακόμη μία άλλη πιθανή επέκταση θα ήταν η αναβάθμιση του Ryu SDX ώστε να υποστηρίζει ένα μεγαλύτερο σύνολο πολιτικών προώθησης. Τέλος πολύ χρήσιμη θα είναι πλέον και η προσθήκη υποστήριξης του πρωτοκόλλου διευθυνσιοδότησης IPv6.

Παράρτημα Κώδικα

Η τελική έκδοση του κώδικα της εργασίας μαζί με όλες τις τροποποιήσεις, οδηγίες εγκατάστασης καθώς και οδηγίες εκτέλεσης των παραδειγμάτων βρίσκεται στο GitHub στις διευθύνσεις: https://github.com/alexdouckas/sdx-ryu και https://github.com/alexdouckas/miniNExT.

Αρχεία παραμετροποίησης του Quagga

Παρακάτω φαίνονται (ενδεικτικά μόνο για τον δρομολογητή a1) το αρχείο zebra.conf που περιέχει τις εντολές για τη διευθυνσιοδότηση των διεπαφών του κόμβου και το αρχείο bgpd.conf με τις αντίστοιχες εντολές για τη διαμόρφωση των παραμέτρων που χρειάζεται για να τρέξει το πρωτόκολλο BGP στον κόμβο. (Οι διευθύνσεις και τα δίκτυα που διαφημίζονται στο BGP είναι ενδεικτικά)

zebra.conf

```
!
! Zebra configuration saved from vty
!   2013/10/02 20:47:51
!
hostname Virtual-AS-A
!
!interface eth0
! ipv6 nd suppress-ra
!
!interface eth1
! ipv6 nd suppress-ra
!
interface bond0
 ip address 172.0.0.1/16
 multicast
!
interface a1-eth0
!
interface a1-eth1
!
line vty
!
```

bgpd.conf

```
!
! Zebra configuration saved from vty
!   2013/10/02 20:47:51
!
hostname Virtual-AS-A
password bgpd
log stdout
!
router bgp 100
 bgp router-id 172.0.0.1
 neighbor 172.0.255.254 remote-as 65000
 network 100.0.0.0/24
 network 110.0.0.0/24
 redistribute static
!
line vty
!
```

Αρχείο Mininet/Mininext για τη δημιουργία και την έναρξη της τοπολογίας

```
import inspect, os, sys, atexit
# Import topo from Mininext
from mininext.topo import Topo
# Import quagga service from examples
from mininext.services.quagga import QuaggaService
# Other Mininext specific imports
from mininext.net import MiniNExT as Mininext
from mininext.cli import CLI
import mininext.util
# Imports from Mininet
import mininet.util
mininet.util.isShellBuiltin = mininext.util.isShellBuiltin
sys.modules['mininet.util'] = mininet.util

from mininet.util import dumpNodeConnections
from mininet.node import RemoteController
from mininet.node import Node
from mininet.link import Link
from mininet.log import setLogLevel, info
from collections import namedtuple
#from mininet.term import makeTerm, cleanUpScreens
QuaggaHost = namedtuple("QuaggaHost", "name ")
net = None


class QuaggaTopo( Topo ):
```

```
    "Quagga topology example."

    def __init__( self ):

        "Initialize topology"
        Topo.__init__( self )

        "Directory where this file / script is located"
        scriptdir =
os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe()))) # script
directory
        "Initialize a service helper for Quagga with default options"
        quaggaSvc = QuaggaService(autoStop=False)

        "Path configurations for mounts"
        quaggaBaseConfigPath=scriptdir + '/quaggacfgs/'

        "List of Quagga host configs"
        quaggaHosts = []
        quaggaHosts.append(QuaggaHost(name = 'a1'))
        quaggaHosts.append(QuaggaHost(name = 'b1'))
        quaggaHosts.append(QuaggaHost(name = 'c1'))
        quaggaHosts.append(QuaggaHost(name = 'c2'))

        "Setup each legacy router, add a link between it and the IXP fabric"
        for host in quaggaHosts:
            "Set Quagga service configuration for this node"
            quaggaSvcConfig = { 'quaggaConfigPath' : scriptdir + '/quaggacfgs/' +
host.name }

            quaggaContainer = self.addHost( name=host.name, privateLogDir=True,
privateRunDir=True, inMountNamespace=True, inPIDNamespace=True)
            self.addNodeService(node=host.name, service=quaggaSvc,
nodeConfig=quaggaSvcConfig)


def addBondingInterfaces( net ):

    print "Configuring Link Aggregation\n\n"
    for host in net.hosts:
        print "Host name: ", host.name
            host.cmd('modprobe bonding')
            host.cmd('ip link add %s-bond0 type bond' % host.name)
            host.cmd('ip link set bond0 address 00:00:00:00:00:%s' % host.name)
            host.cmd('ip link set %s-eth0 down' % host.name)
            host.cmd('ip link set %s-eth0 address 00:00:00:00:%s:10' % (host.name,
host.name))
            host.cmd('ip link set %s-eth0 master bond0' % host.name)
            host.cmd('ip link set %s-eth1 down' % host.name)
            host.cmd('ip link set %s-eth1 address 00:00:00:00:%s:11' % (host.name,
host.name))
            host.cmd('ip link set %s-eth1 master bond0' % host.name)
            host.cmd('ip link set bond0 up')
```

```
def addInterfacesForSDXNetwork( net ):
    hosts=net.hosts
    print "Configuring loopbacks\n\n"
    for host in hosts:
        print "Host name: ", host.name
        if host.name=='a1':
            host.cmd('sudo ifconfig lo:1 100.0.0.1 netmask 255.255.255.0 up')
            host.cmd('sudo ifconfig lo:2 100.0.0.2 netmask 255.255.255.0 up')
            host.cmd('sudo ifconfig lo:110 110.0.0.1 netmask 255.255.255.0 up')
        if host.name=='b1':
            host.cmd('sudo ifconfig lo:140 140.0.0.1 netmask 255.255.255.0 up')
            host.cmd('sudo ifconfig lo:150 150.0.0.1 netmask 255.255.255.0 up')
        if host.name=='c1':
            host.cmd('sudo ifconfig lo:140 140.0.0.1 netmask 255.255.255.0 up')
            host.cmd('sudo ifconfig lo:150 150.0.0.1 netmask 255.255.255.0 up')
        if host.name=='c2':
            host.cmd('sudo ifconfig lo:140 140.0.0.1 netmask 255.255.255.0 up')
            host.cmd('sudo ifconfig lo:150 150.0.0.1 netmask 255.255.255.0 up')
        #if host.name == 'exabgp':
            #host.cmd( 'route add -net 172.0.0.0/16 dev exabgp-eth0')
def addMultiLinks ( net, x, switch):
    portNo = 0

    print "Adding " + x + " Links with the switch for each host\n\n"
    for host in net.hosts:
        for i in range (0,x)
        portNo = portNo + 1
        net.addLink( host, switch, port2 = portNo)

def startNetwork():
    info( '** Creating Quagga network topology\n' )
    topo = QuaggaTopo()
    global net
    net = Mininext(topo=topo,
                controller=lambda name: RemoteController( name, ip='127.0.0.1'
),listenPort=6633)

    s1 = net.addSwitch( 's1' )

    info( '** adding links\n' )
       addMultiLinks ( net, 2, s1)

       addBondingInterfaces( net )
    info( '** Creating ExaBGP host\n' )
    exa = net.addHost('exabgp', inNamespace = False)
    net.addLink(exa, s1)
    exa.cmd('ip addr add 172.0.255.254/16 dev exabgp-eth0')
    exa.cmd( 'route add -net 172.0.0.0/16 dev exabgp-eth0')

    info( '** Starting the network\n' )
    net.start()

    info( '** psaux dumps on all hosts\n' )
    for lr in net.hosts:
        if lr.name != 'exabgp':
```

```
            lr.cmdPrint("ps aux")

    info( '**Adding Network Interfaces for SDX Setup\n' )
    addInterfacesForSDXNetwork(net)

    info( '** Running CLI\n' )
    CLI( net )

def stopNetwork():
    if net is not None:
        info( '** Tearing down Quagga network\n' )
        net.stop()

if __name__ == '__main__':
    # Force cleanup on exit by registering a cleanup function
    atexit.register(stopNetwork)

    # Tell mininet to print useful information
    setLogLevel('info')
    startNetwork()
```

Αρχείο JSON για τη δημιουργία της τοπολογίας στο Ryu SDX (sdx_global.cfg)

```
{
"1": {
      "Ports": [
            {
                  "Bond": 2,
                  "MAC": "08:00:27:54:48:ed",
                  "IP": "172.0.0.1"
            }
      ],
      "Peers": [2,3],
      "ASN": 100
},
"2": {
      "Ports": [
            {
                  "Bond": 2,
                  "MAC": "08:00:27:54:56:10",
                  "IP": "172.0.0.11"
            }
      ],
```

```
        "Peers": [1,3],
        "ASN": 200
},
"3": {
        "Ports": [
                {
                        "Bond": 2,
                        "MAC": "08:00:27:54:56:ea",
                        "IP": "172.0.0.21"
                },
                {
                        "Bond": 2,
                        "MAC": "08:00:27:bd:f8:b2",
                        "IP": "172.0.0.22"
                }
        ],
        "Peers": [1,2],
        "ASN": 300
}
}
```

Αρχείο JSON για την ανάγνωση των πολιτικών στο Ryu SDX (sdx_policies.cfg)

```
{
        "1": "/home/vagrant/sdx-ryu/policies/participant_1.py",
        "2": "/home/vagrant/sdx-ryu/policies/participant_2.py",
        "3": "/home/vagrant/sdx-ryu/policies/participant_3.py"
}
```

Αρχείο JSON για την παραμετροποίηση του route server του Ryu SDX (bgp.cfg)

```
group rs {
        process parsed-route-backend {
                run /home/vagrant/sdx-ryu/xrs/client.py;
                encoder json;
                receive-routes;
        }
```

```
        neighbor 172.0.0.1 {
                description "Virtual AS A";
                router-id 172.0.255.254;
                local-address 172.0.255.254;
                local-as 65000;
                peer-as 100;
                hold-time 180;
        }

        neighbor 172.0.0.11 {
                description "Virtual AS B";
                router-id 172.0.255.254;
                local-address 172.0.255.254;
                local-as 65000;
                peer-as 200;
                hold-time 180;
        }

        neighbor 172.0.0.21 {
                description "Virtual AS C Router C1";
                router-id 172.0.255.254;
                local-address 172.0.255.254;
                local-as 65000;
                peer-as 300;
                hold-time 180;
        }
        neighbor 172.0.0.22 {
                description "Virtual AS C Router C2";
                router-id 172.0.255.254;
                local-address 172.0.255.254;
                local-as 65000;
                peer-as 300;
                hold-time 180;
        }
```

Προσθήκες που έγιναν στην αρχική υλοποίηση του Ryu SDX

Οι προσθήκες έγιναν με τη χρήση της βιβλιοθήκης lacplib που παρέχει το Ryu (https://github.com/osrg/ryu/blob/master/ryu/lib/lacplib.py). Παρουσιάζονται παρακάτω οι κύριες διαφορές του αρχικού κώδικα του SDX σε σχέση με το σύστημα που υλοποιήθηκε.

Αρχικά έγινε προσθήκη των διεπαφών οι στις οποίες ανά ζευγάρια θα ενεργοποιείται το LACP.

```
#lacp support for the desired interfaces
self._lacp = kwargs['lacplib']
self._lacp.add(dpid=str_to_dpid('0000000000000001'), ports=[1, 2])
self._lacp.add(dpid=str_to_dpid('0000000000000001'), ports=[3, 4])
self._lacp.add(dpid=str_to_dpid('0000000000000001'), ports=[5, 6])
self._lacp.add(dpid=str_to_dpid('0000000000000001'), ports=[7, 8])

#parsing of sdx_global.cfg
self.sdx = parse_config(base_path, config_file, policy_file)
```

Προσθήκη υποστήριξης LACP πακέτων για τον packet_handler του μεταγωγέα :

```
@set_ev_cls(lacplib.EventPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
```

Προσθήκη συνάρτησης για τη διαγραφή μίας εγγραφής από τον πίνακα ροής του μεταγωγέα σε περίπτωση που έχουμε απώλεια της σύνδεσης:

```
@set_ev_cls(lacplib.EventSlaveStateChanged, MAIN_DISPATCHER)
def _slave_state_changed_handler(self, ev):
    datapath = ev.datapath
    dpid = datapath.id
    port_no = ev.port
    enabled = ev.enabled
    self.logger.info("slave state changed port: %d enabled: %s",
                     port_no, enabled)
    if dpid in self.mac_to_port:
        for mac in self.mac_to_port[dpid]:
            self.delete_flows(self.datapath, NO_COOKIE, MAIN_TABLE, mac)
        del self.mac_to_port[dpid]
    self.mac_to_port.setdefault(dpid, {})
```

Βιβλιογραφία

[1] Software Defined Networking https://en.wikipedia.org/wiki/Software-defined_networking

[2] Inside SDN Architecture https://www.sdxcentral.com/resources/sdn/inside-sdn-architecture/

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

[4] OpenFlow https://en.wikipedia.org/wiki/OpenFlow

[5] Open Networking Foundation (ONF), “OpenFlow Switch Specification”, Version 1.5, December 2014

[6] Border Gateway Protocol https://en.wikipedia.org/wiki/Border_Gateway_Protocol

[7] Introduction to BGP http://searchtelecom.techtarget.com/tip/Introduction-to-Border-Gateway-Protocol-BGP

[8] Internet Exchange Point https://en.wikipedia.org/wiki/Internet_exchange_point

[9] Route Server https://en.wikipedia.org/wiki/Route_server

[10] Link aggregation https://en.wikipedia.org/wiki/Link_aggregation

[11] Linux Bonding Driver Documentation https://www.kernel.org/doc/Documentation/networking/bonding.txt

[12] Mininet Overview http://mininet.org/overview/

[13] Ryu SDN Framework http://osrg.github.io/ryu/

[14] Ryu Introduction http://www.slideshare.net/yamahata/ryu-sdnframeworkupload

[15] Ryu Open-source SDN Platform Software https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201408fa4.html

[16] SDX home page http://noise-lab.net/projects/software-defined-networking/sdx/

[17] SDX: A Software Defined Internet Exchange http://gtnoise.net/papers/2014/gupta-sigcomm2014.pdf

[18] ExaBGP Overview https://github.com/Exa-Networks/exabgp/wiki

[19] Quagga Routing Suite http://www.nongnu.org/quagga/

[20] MiniNExT Github Repository https://github.com/USC-NSL/miniNExT

[21] SDX Github Repository https://github.com/sdn-ixp