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

Εφαρμογή SDN για δυναμική δρομολόγηση σε Software Defined Networks

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παληού Ε. Δέσποινα

Επιβλέπων: Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2016

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

Εφαρμογή SDN για δυναμική δρομολόγηση σε Software Defined Networks

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παληού Ε. Δέσποινα

Επιβλέπων: Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 25η Οκτωβρίου 2016

.....	.....	.....
Ευστάθιος Συκάς	Μιχαήλ Θεολόγου	Γεώργιος Στασινόπουλος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2016

.....
Παληού Ε. Δέσποινα

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

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη μιας εφαρμογής SDN που ελέγχει εξωτερικά τον ελεγκτή ενός SDN δικτύου και αποσκοπεί στον επαναπροσδιορισμό της δρομολόγησης στο δίκτυο προκειμένου να επιτευχθεί δρομολόγηση με ενεργειακά βέλτιστο τρόπο. Για την επιλογή της δρομολόγησης λαμβάνονται υπόψιν ενεργειακά μοντέλα των συσκευών καθώς και στατιστικά της κίνησης στις ροές του δικτύου, που προέκυψαν μετά από κατάλληλη παραμετροποίησή του μέσω του ελεγκτή.

Τα Ευφυή Προγραμματιζόμενα Δίκτυα μέσω του βασικού πρωτοκόλλου OpenFlow, διαχωρίζουν το επίπεδο ελέγχου από το επίπεδο προώθησης δεδομένων παρέχοντας έναν κεντρικοποιημένο έλεγχο του δικτύου μέσω του ελεγκτή. Ο ελεγκτής που επιλέχθηκε για τη διπλωματική είναι ο OpenDaylight controller ενώ για την τοπολογία του δικτύου χρησιμοποιήθηκε το Mininet που είναι εξομοιωτής δικτύων. Η τοπολογία που επιλέχθηκε προσομοιάζει τις τοπολογίες που επιλέγονται για τα Data Centers λόγω του μεγάλου ποσοστού ενέργειας που καταναλώνεται σ’ αυτά. Για την εκμάθηση της τοπολογίας του δικτύου καθώς και για την παραμετροποίησή του χρησιμοποιήθηκε το REST api του OpenDaylight, ενώ για την εύρεση της βέλτιστης δρομολόγησης χρησιμοποιήθηκε το λογισμικό CPLEX.

Λέξεις Κλειδιά : OpenFlow, OpenDaylight Controller, REST api, ροή, στατιστικά ροής, Mininet, ενεργειακή δρομολόγηση

Abstract

The purpose of this diploma thesis is to develop an SDN app that externally manages the Controller of an SDN network and which aims to reformulate the network routing, with a view to achieve routing at an optimal consumption of energy. For the process of selection of routing we consider the energy models of the devices as well as network traffic flow statistics that have been derived through the proper configuration achieved via the Controller.

The Software Defined Networks, through the essential protocol OpenFlow, decouple the control plane from the data plane, providing centralized control of the network via the Controller. The Controller chosen for this thesis is the OpenDaylight Controller, while for the network's topology the network emulator Mininet was used. The topology chosen emulates Data Centers topologies owing to their vast consumption of energy. For the network topology learning and configuration we used Rest api, while for finding optimal routes we used CPLEX.

Key Words: OpenFlow, OpenDaylight Controller, REST api, flow, flow statistics, Mininet, Energy Aware Routing

Ευχαριστίες

Η διπλωματική αυτή εργασία αποτελεί το επιστέγασμα της φοιτητικής μου πορείας στο Εθνικό Μετσόβιο Πολυτεχνείο και θα ήθελα να εκφράσω τις ευχαριστίες μου στους ανθρώπους που στάθηκαν δίπλα μου και συνετέλεσαν σε αυτήν.

Αρχικά, ιδιαίτερες ευχαριστίες οφείλω στον επιβλέποντα καθηγητή μου, κ. Ευστάθιο Συκά για την καθοδήγηση που μου παρείχε στην επιλογή αυτού του ενδιαφέροντος θέματος καθώς και για την εμπιστοσύνη που μου έδειξε για την ανάθεση του. Επίσης θα ήθελα να ευχαριστήσω θερμά τον υποψήφιο διδάκτορα, κ. Πάρη Χαραλάμπου για την υποστήριξη και την πολύτιμη βοήθεια που μου παρείχε καθώς και για την άριστη επικοινωνία μας.

Είναι ευκαιρία να ευχαριστήσω εδώ και τους φίλους μου που στάθηκαν δίπλα μου αυτά τα χρόνια, πολλοί εκ των οποίων υπήρξαν και συνεργάτες μου στη σχολή. Τους ευχαριστώ πολύ για τις δημιουργικές συνεργασίες που είχαμε, τις όμορφες αλλά και δύσκολες στιγμές που περάσαμε παρέα προκειμένου να ανταπεξέλθουμε στις απαιτήσεις της σχολής.

Τέλος, θα ήθελα να εκφράσω την ευγνωμοσύνη μου στην οικογένεια μου, στα δύο μου αδέλφια και ιδιαίτερα στους γονείς μου. Με την διακριτική παρουσία τους πάντα στη ζωή μου, την εμπιστοσύνη και υποστήριξη που έδειχναν πάντα στις επιλογές μου, με βοηθήσαν να βρίσκομαι εδώ σήμερα, ως φοιτήτρια και ως άνθρωπος.

Δέσποινα Ε. Παληού,

Αθήνα, Οκτώβριος 2016

Πίνακας περιεχομένων

Ευρετήριο εικόνων

1 Εισαγωγή

1.1 Σκοπός

Σε αυτήν τη διπλωματική εργασία αυτοματοποιείται η διαδικασία συλλογής στατιστικών στοιχείων σε Ευφυή Προγραμματιζόμενα Δίκτυα (Software Defined Networks) μικρής ή και μεγάλης κλίμακας προκειμένου να χρησιμοποιηθούν για την εύρεση αποδοτικότερης δρομολόγησης εντός του δικτύου σε πραγματικό χρόνο, με σκοπό να εξοικονομηθεί ενέργεια από τη βέλτιστη χρησιμοποίηση των πόρων του. Η ιδέα της διερεύνησης αυτής προέκυψε από την παρατήρηση πως η ολοένα κι αυξανόμενη χρησιμοποίηση του διαδικτύου και η ανάγκη ύπαρξης μεγάλων υποδομών σε κέντρα δεδομένων (data centers) επιβαρύνει το περιβάλλον λόγω της απαίτησης για κατανάλωση ενέργειας.

Στα πλαίσια της εργασίας χρησιμοποιήθηκε ο OpenDaylight Controller (ODL). Μέσω του ODL, ο διαχειριστής δύναται απομακρυσμένα και από οποιοδήποτε μηχάνημα εκτός του δικτύου να συλλέξει πληροφορίες για το δίκτυο, αλλά και να του στείλει επίσης πίσω κατάλληλες παραμετροποιήσεις για τη συνέχιση και διαφοροποίηση της λειτουργίας του. Δημιουργήθηκε εφαρμογή που εξωτερικά ελέγχει τον ODL, ο οποίος με τη σειρά του εποπτεύει το δίκτυο. Η τοπολογία που επιλέχθηκε τρέχει στο εργαλείο εξομοίωσης δικτύων Mininet, με την εφαρμογή να μπορεί να χρησιμοποιηθεί χωρίς αλλαγή σε οποιαδήποτε τοπολογία και να κληθεί από οποιοδήποτε μηχάνημα.

Σε πρώτη φάση συλλέγονται πληροφορίες για την τοπολογία του εκάστοτε δικτύου μέσω του ODL. Ανάλογα τις πληροφορίες αυτές στέλνονται κατάλληλες παραμετροποιήσεις για την τροποποίηση στοιχείων και στη συνέχεια γίνεται συλλογή, επεξεργασία και εξαγωγή των στατιστικών. Στη συνέχεια με τη χρήση αυτών, γίνεται επιλογή για το ποιες διεπαφές θα παραμείνουν ανοιχτές και ποιες θα κλείσουν. Η εφαρμογή τρέχει διαρκώς και σε πραγματικό χρόνο, προκειμένου να εποπτεύει την ενεργειακά αποδοτική χρησιμοποίηση του εκάστοτε δικτύου.

2 Ευφυή Προγραμματιζόμενα Δίκτυα

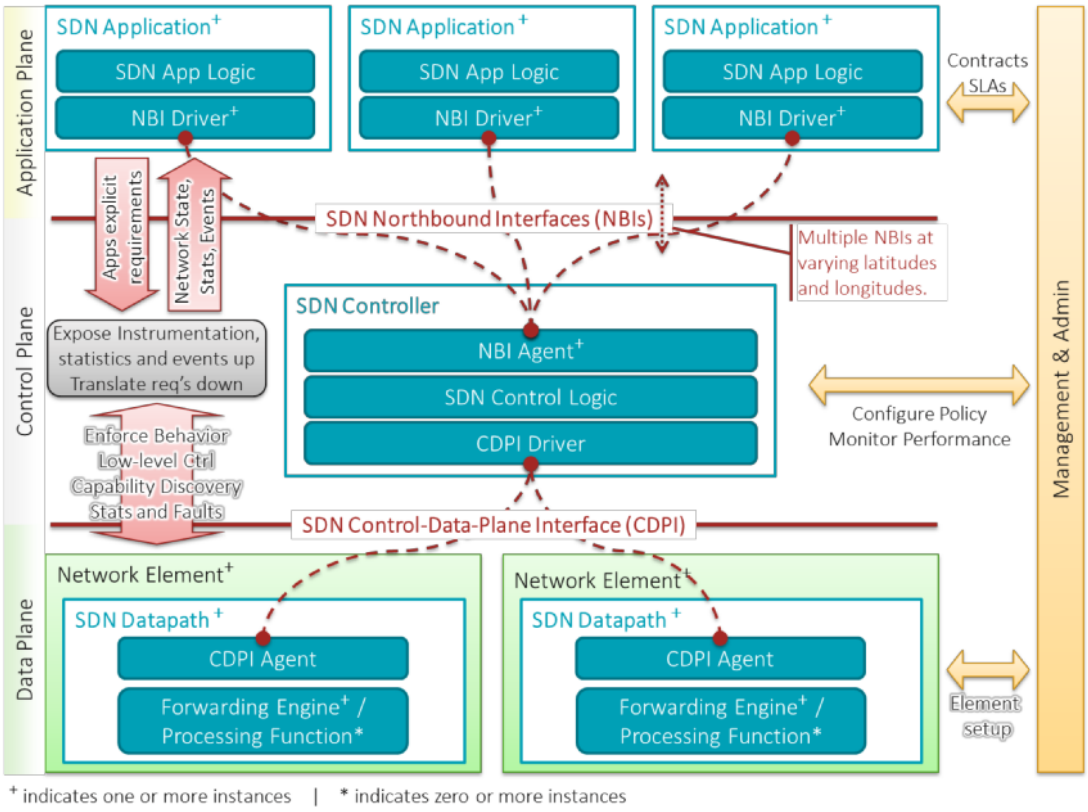
Η εξάπλωση των κινητών ευφυών συσκευών, η ανάγκη εικονικοποίησης των μηχανημάτων καθώς και οι νέες υπηρεσίες Cloud επιτάσσουν τη δημιουργία μιας νέας αρχιτεκτονικής στα δίκτυα προκειμένου να ικανοποιηθούν οι νέες ανάγκες στις οποίες δεν μπορούν να ανταπεξέλθουν τα παραδοσιακά δίκτυα. Έτσι με τη βοήθεια του Ανοικτού Λογισμικού, η κοινότητα Open Networking Foundation (ONF) εισήγαγε τη νέα γενιά αρχιτεκτονικής δικτύων, τα Λογισμικώς Καθοριζόμενα Δίκτυα (Software Defined Networks).

Το Software Defined Networking (SDN) στοχεύει στο να δοθεί η δυνατότητα στους διαχειριστές δικτύων να μπορούν άμεσα και απομακρυσμένα να ελέγξουν το δίκτυο μέσω ενός κεντρικοποιημένου ελέγχου που προσφέρουν. Το SDN μπορεί να χρησιμοποιήσει υποδομή διαφορετικών κατασκευαστών προκειμένου να σχεδιάσει, ελέγξει και διαχειριστεί τα δίκτυα που γίνονται πιο ευέλικτα μέσω της κεντρικής ιδέας του SDN , τον διαχωρισμό του επιπέδου ελέγχου από το επίπεδο δεδομένων.

Το επίπεδο ελέγχου είναι υπεύθυνο για τη δρομολόγηση των πακέτων ενώ το επίπεδο δεδομένων για τη μεταφορά τους. Σε αντίθεση με την παραδοσιακή TCP/IP αρχιτεκτονική που η δρομολόγηση γίνεται βάσει στατικών κανόνων του πρωτοκόλλου IP, στο SDN παρέχεται η δυνατότητα να χρησιμοποιηθούν πολιτικές δρομολόγησης βάσει των απαιτήσεων του κάθε πελάτη και των αναγκών του δικτύου. Τα δίκτυα γίνονται προγραμματίσιμα και διαχειρίσιμα κάνοντας την υποδομή διαφανή για τις εφαρμογές και υπηρεσίες που παρέχουν, ενώ διευκολύνεται και η κατανομή πόρων, η κλιμακωσιμότητα στα κατανεμημένα κέντρα δεδομένων και η εικονικοποίηση των συσκευών που είναι απαραίτητη για περιβάλλοντα Cloud. [1] [2]

2.1 Αρχιτεκτονική SDN

Υπάρχουν τρεις βασικές δομές στο SDN, το επίπεδο εφαρμογών, το επίπεδο ελέγχου και το επίπεδο δεδομένων. Στο επίπεδο εφαρμογών ανήκουν οι εφαρμογές που μέσω της NorthBound SDN διεπαφής εκφράζουν στο επίπεδο ελέγχου , δηλαδή στον SDN ελεγκτή, τις ανάγκες του δικτύου. Στο επίπεδο δεδομένων ανήκουν οι δικτυακές συσκευές κι εκεί λαμβάνεται χώρα η προώθηση των πακέτων με τρόπο που έχει ορίσει το επίπεδο ελέγχου. Στην παρακάτω εικόνα φαίνεται αναλυτικά η αρχιτεκτονική του SDN. [3]



Εικόνα 1: Επισκόπηση Αρχιτεκτονικής των Software Defined Networks

2.1.1 **Επίπεδο Εφαρμογών** (SDN Application)

Πρόκειται για προγράμματα μέσω των οποίων ο διαχειριστής του δικτύου μπορεί άμεσα να επικοινωνήσει με τον ελεγκτή ώστε να μάθει τις απαιτήσεις του δικτύου και να ορίσει την απαιτούμενη συμπεριφορά που πρέπει να εφαρμοστεί στο δίκτυο.

2.1.2 **Επίπεδο Ελέγχου** (SDN Controller)

Η πλατφόρμα ελέγχου περιλαμβάνει στην ουσία τον SDN ελεγκτή που αποτελεί μια λογικά κεντρικοποιημένη ενότητα. Σε φυσικό επίπεδο ο ελεγκτής μπορεί να είναι κατανεμημένος σε διαφορετικά φυσικά μηχανήματα προκειμένου να υπάρχει κλιμακωσιμότητα και βελτιστοποίηση των διεργασιών. Ωστόσο, ο ελεγκτής συμπεριφέρεται ως μοναδική λογική μονάδα, με δικό της λειτουργικό σύστημα λαμβάνοντας ενιαίες αποφάσεις. Αρχικά λαμβάνει τις απαιτήσεις του συστήματος -δηλαδή του επιπέδου ελέγχου- και τις μεταφέρει στο επίπεδο δεδομένων κι αντίστοιχα αντιλαμβάνεται τυχόν αλλαγές στη λειτουργία του δικτύου -επίπεδο δεδομένων- και ενημερώνει το επίπεδο εφαρμογών.

Αποτελείται στην ουσία από τους NorthBound Agents που συντελούν στην επικοινωνία του ελεγκτή με το επίπεδο εφαρμογών, την κεντρική μονάδα ελέγχου που είναι υπεύθυνη για τις διεργασίες που λαμβάνουν χώρα για τη λήψη αποφάσεων, καθώς και από CDPI (Control to Data-Plane Interface) Drivers που είναι υπεύθυνοι για τη σωστή επικοινωνία με το επίπεδο δεδομένων.

2.1.3 **Επίπεδο Δεδομένων** (SDN Datapath)

Αποτελεί μια δικτυακή λογική οντότητα που παρέχει εποπτεία κι έλεγχο των διαδικασιών προώθησης και επεξεργασίας δεδομένων που λαμβάνουν χώρα στο δίκτυο. Αποτελείται από έναν CDPI Agent προκειμένου να επιτυγχάνεται η επικοινωνία με τον ελεγκτή και από ένα σύνολο συσκευών υπεύθυνων για την προώθηση της κίνησης καθώς και τις αντίστοιχες μεθόδους.

Το επίπεδο δεδομένων -αν και συμπεριφέρεται ως ενιαία ενότητα- δύναται να αποτελείται από κατανεμημένες φυσικές συσκευές με διαμοιραζόμενους πόρους, ενώ μπορεί να λειτουργήσει και με συσκευές που δεν είναι SDN.

2.1.4 **NorthBound SDN Διεπαφή** (SDN NorthBound Interfaces – NBI)

Πρόκειται για διεπαφές που είναι υπεύθυνες για την επικοινωνία του επιπέδου εφαρμογών με το επίπεδο ελέγχου. Παρέχουν αφενός άποψη του δικτύου στο επίπεδο εφαρμογών και αφετέρου τη δυνατότητα άμεσης παρέμβασης στις απαιτήσεις του δικτύου.

2.1.5 **Διεπαφή διασύνδεσης επιπέδου ελέγχου-δεδομένων** (SDN Control to Data-Plane Interface – CDPI)

Αποτελεί τη διεπαφή που είναι υπεύθυνη για την επικοινωνία του ελεγκτή με το επίπεδο δεδομένων. Παρέχει προγραμματιστικό έλεγχο όλων των λειτουργιών προώθησης των δεδομένων, αναφορά για τα στατιστικά της κίνησης στο δίκτυο καθώς και ενημέρωση του ελεγκτή σε περίπτωση κάποιας αλλαγής ή προβλήματος στο δίκτυο.

2.1.6 **Διαχείριση** (Management & Admin)

Δεν αποτελεί μέρος της Αρχιτεκτονικής του SDN αλλά αποτελεί ουσιαστικά το λόγο ανάπτυξης των SDN, δηλαδή τον έλεγχο που μπορεί πλέον να εφαρμοστεί στα δίκτυα που υποστηρίζουν SDN. Βρίσκεται έξω από τις εφαρμογές, το επίπεδο ελέγχου και δεδομένων των SDN. Μέσω του επιπέδου εφαρμογών ο διαχειριστής μπορεί να ενημερωθεί για το δίκτυο ορίζοντας κατάλληλες συμπεριφορές. Όλη αυτή η επικοινωνία των εφαρμογών και του διαχειριστή, γίνεται με τη βοήθεια του ελεγκτή.

2.2 Το πρωτόκολλο OpenFlow

Το OpenFlow είναι το πρώτο προτυποποιημένο πρωτόκολλο για την επικοινωνία του επιπέδου ελέγχου με το επίπεδο δεδομένων σε SDN αρχιτεκτονικές. Επιτρέπει την άμεση πρόσβαση και διαχείριση της πλατφόρμας προώθησης των δεδομένων των δικτυακών συσκευών, δηλαδή των εικονικών και φυσικών μεταγωγέων και δρομολογητών. Το OpenFlow είναι απαραίτητο για τη μετακίνηση του ελέγχου του δικτύου από τις δικτυακές συσκευές σε ένα λογικά κατανεμημένο λογισμικό ελέγχου και εφαρμόζεται και στις δυο πλευρές της διεπαφής μεταξύ του λογισμικού ελέγχου του SDN και της δικτυακής υποδομής.

Το OpenFlow χρησιμοποιεί την έννοια της ροής (flow) για να αναγνωρίσει τη δικτυακή κίνηση και η οποία βασίζεται σε προκαθορισμένους κανόνες αντιστοίχισης που μπορούν στατικά ή δυναμικά να οριστούν από τον ελεγκτή. Επίσης επιτρέπει στον διαχειριστή να ορίσει πως θα κατανεμηθεί η κίνηση στις δικτυακές συσκευές ανάλογα τους πόρους του δικτύου, τις ανάγκες των εφαρμογών και τα πρότυπα της συνηθισμένης κίνησης στο δίκτυο. Ενώ στα παραδοσιακά IP δίκτυα η δρομολόγηση στο διαδίκτυο γίνεται βάσει των διευθύνσεων IP και δύο ροές με τα ίδια εναρκτήρια και τερματικά σημεία θα ακολουθήσουν την ίδια διαδρομή λόγω των ίδιων διευθύνσεων IP που έχουν, στην αρχιτεκτονική SDN δεν συμβαίνει το ίδιο. Πλέον δύο ροές μπορεί να έχουν ίδια σημεία πηγής και προορισμού αλλά να ακολουθούν άλλη διαδρομή και με διαφορετική προτεραιότητα διότι έχουν διαφορετικές απαιτήσεις, οπότε ο πάροχος να επιθυμεί διαφορετική πολιτική δρομολόγησης για την καθεμία.

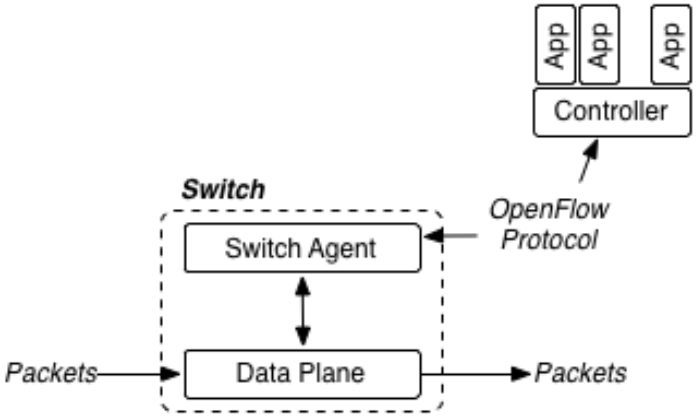
Δίκτυα αρχιτεκτονικής SDN που υποστηρίζουν OpenFlow μπορούν να αναπτυχθούν σε ήδη υπάρχοντα δίκτυα. Οι συσκευές δικτύωσης μπορούν να υποστηρίξουν προώθηση βάσει του OpenFlow όσο και βάσει της συμβατικής δρομολόγησης TCP/IP. Υπάρχει η δυνατότητα υβριδικών μεταγωγέων που λειτουργούν σε κάθε περιβάλλον (SDN ή μη) ή εφαρμόζουν κεντρικοποιημένο έλεγχο στις εφαρμογές (παρακολούθηση και διαχείριση του δικτύου), οπότε δεν απαιτείται να αντικαταστηθεί κάθε δικτυακή συσκευή με SDN-μεταγωγείς. [4] Αυτό είναι μεγάλο πλεονέκτημα για τους παρόχους που μπορούν να οδηγηθούν στην αρχιτεκτονική SDN σταδιακά, βασιζόμενοι στα παραδοσιακά δίκτυα τους μετατρέποντάς τα σε SDN.

2.3 OpenFlow Μεταγωγέας (OpenFlow Switch)

Ο OpenFlow μεταγωγέας αποτελείται από δύο μέρη: το switch-agent και την πλατφόρμα δεδομένων.

Το switch-agent επικοινωνεί μέσω του OpenFlow πρωτοκόλλου με τον ελεγκτή καθώς και με την πλατφόρμα δεδομένων μέσω εσωτερικού κατάλληλου πρωτοκόλλου. Είναι υπεύθυνο στο να μεταφράζει τις εντολές του ελεγκτή σε κατάλληλη γλώσσα μηχανής και να τις στέλνει στην πλατφόρμα δεδομένων προκειμένου να της είναι κατανοητές. Επίσης αντίστοιχα λαμβάνει από την πλατφόρμα δεδομένων τυχόν ειδοποιήσεις και τις μετατρέπει σε OpenFlow μηνύματα κατανοητά από τον ελεγκτή και του τα προωθεί.

Η πλατφόρμα δεδομένων αναλαμβάνει τη διαχείριση και προώθηση των πακέτων, ενώ ανάλογα την παραμετροποίηση ενδέχεται να στείλει πακέτα στο switch-agent για επεξεργασία. Περιλαμβάνει πίνακες ροής και ανάλογες δράσεις για κάθε ροή, οπότε τα πακέτα προωθούνται ανάλογα σε ποια ροή ανήκουν και σε ποιον κανόνα του πίνακα ροής θα αντιστοιχιστούν. [2]



Εικόνα 2: Δομή OpenFlow Μεταγωγέα

3 OpenDaylight Controller

Ο SDN ελεγκτής αποτελεί τη μονάδα ελέγχου του ευφυούς δικτύου, αντιπροσωπεύοντας το επίπεδο ελέγχου μετά το διαχωρισμό του από το επίπεδο δεδομένων. Είναι η πλατφόρμα που εποπτεύει και διαχειρίζεται τις ροές κίνησης μέσω του ελέγχου των δικτυακών συσκευών με τα southbound APIs και παράλληλα ενημερώνει και διαχειρίζεται τις εφαρμογές με τα northbound APIs. [5]

Οι SDN ελεγκτές χρησιμοποιούν συνήθως το OpenFlow ή και το OVSDB πρωτόκολλο για την επικοινωνία τους με τις δικτυακές συσκευές. Επίσης υπάρχουν και άλλα πρωτόκολλα γι αυτήν την επικοινωνία, το NetConf και το Yang. Στην παρούσα διπλωματική επιλέχθηκε ο OpenDaylight SDN ελεγκτής.

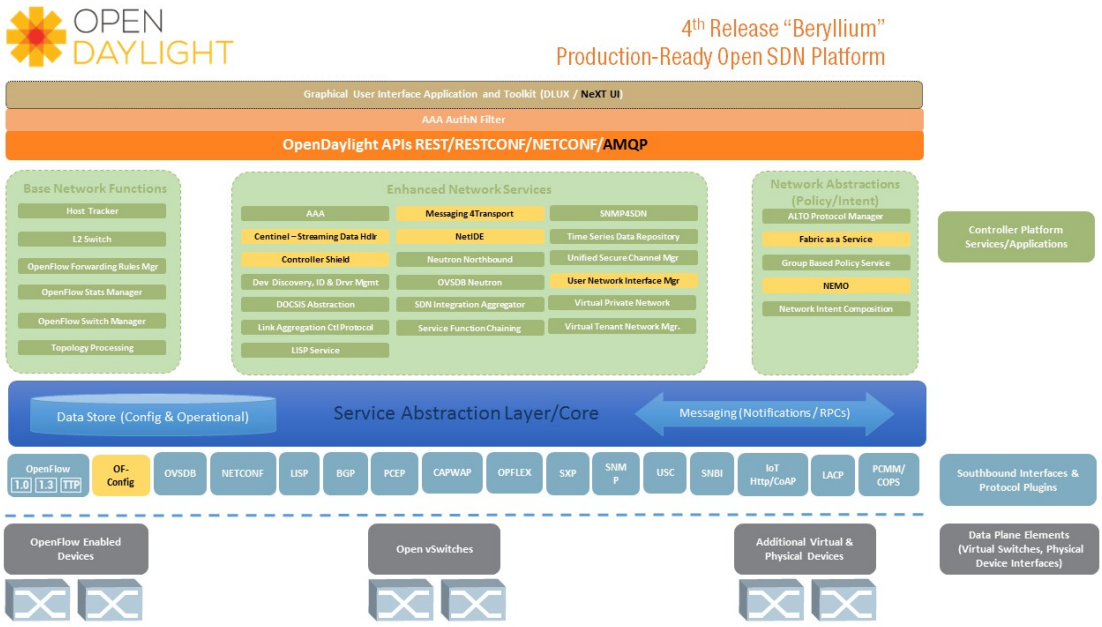
Ο OpenDaylight ελεγκτής αναπτύχθηκε από την κοινότητα Linux Foundation και αποτελεί το μεγάλο έργο Ανοικτού Λογισμικού σχετικά με το Software Defined Networking. Ο κώδικάς του είναι γραμμένος σε γλώσσα JAVA και δύναται να επεκταθεί ως προς τις λειτουργίες του. Μεγάλες εταιρίες του κλάδου όπως οι Cisco, Brocade , Juniper Networks, IBM, Vmware, Ericsson, Microsoft, Big Switch Networks καθώς και ανεξάρτητοι μηχανικοί στα πλαίσια του Ανοικτού Λογισμικού δουλεύουν επίσης για να συμβάλουν στην ανάπτυξη του OpenDaylight project. Ο OpenDaylight υποστηρίζει το OpenFlow πρωτόκολλο καθώς και άλλα πρωτόκολλα και τεχνολογίες SDN αρχιτεκτονικής εφόσον μπορεί να παραμετροποιηθεί καταλλήλως από τον διαχειριστή. [6] [7]

3.1 Αρχιτεκτονική OpenDaylight

Ο OpenDaylight ελεγκτής αποτελείται από τρία μέρη:

- Την πλατφόρμα του ελεγκτή
- Τις Northbound εφαρμογές και υπηρεσίες
- Τις Southbound επεκτάσεις και πρωτόκολλα

Στην παρακάτω εικόνα φαίνονται τα στοιχεία του OpenDaylight στη νεότερη έκδοση του (Αύγουστος 2016), την Beryllium.



Εικόνα 3: Επισκόπηση Αρχιτεκτονικής OpenDaylight Controller (Έκδοση Beryllium) [8]

3.2 Πλατφόρμα Ελεγκτή

Η πλατφόρμα του ελεγκτή εμπεριέχει τόσο Northbound όσο και Southbound διεπαφή.

Η Northbound διεπαφή παρέχει στον ελεγκτή υπηρεσίες καθώς και ένα σύνολο REST API εφαρμογών που μπορούν να χρησιμοποιηθούν για τη διαχείριση και την παραμετροποίηση της υποδομής του δικτύου. Η πρόσβαση στη Northbound διεπαφή του ελεγκτή είναι εφικτή μετά την επαλήθευση ταυτότητας και αδειοδότησης του χρήστη, όπως φαίνεται στο υψηλότερο επίπεδο της παραπάνω εικόνας.

Η Southbound διεπαφή υλοποιεί πρωτόκολλα για τη διαχείριση και τον έλεγχο της δικτυακής υποδομής που βρίσκεται στο κατώτερο επίπεδο. Εδώ υλοποιούνται αρκετές επεκτάσεις είτε για την υποστήριξη δικτυακών πρωτοκόλλων είτε για την απευθείας επικοινωνία με το υλικό. Τα πιο γνωστά πρωτόκολλα που υλοποιούνται είναι τα OpenFlow, NetConf, SNMP, OVSDB.

Η πλατφόρμα του ελεγκτή επικοινωνεί με το δίκτυο χρησιμοποιώντας τη Southbound διεπαφή και παρέχει βασικές δικτυακές υπηρεσίες μέσω ενός συνόλου λειτουργιών: Host Tracker, L2 Switch, OpenFlow Forwarding Rules Mgr, OpenFlow Stats Manager, OpenFlow Switch Manager, Topology Processing.

3.2.1 **Βασικές Λειτουργίες Υπηρεσιών Δικτύωσης** (Base Network Service Functions)

- Host Tracker

Αποθηκεύει πληροφορίες για τα τερματικά του συστήματος (διευθύνσεις MAC, IP, τύπος μεταγωγέα, αριθμός θύρας) και παρέχει API για την ανάκτηση πληροφοριών γι αυτά. Δύναται να δουλέψει με στατικό ή δυναμικό τρόπο. Στατικώς, η βάση δεδομένων του Host Tracker μεταχειρίζεται χειροκίνητα μέσω Northbound APIs ενώ δυναμικώς ο Host Tracker χρησιμοποιεί το ARP πρωτόκολλο για να ανιχνεύσει τις πληροφορίες για τη βάση δεδομένων του.

- L2 Switch

Με την άφιξη ενός πακέτου, η λειτουργία αυτή μαθαίνει την MAC διεύθυνση της πηγής. Εάν γνωρίζει τον προορισμό, μεταφέρει το πακέτο στον προορισμό, διαφορετικά στέλνει broadcast μήνυμα (δηλαδή μήνυμα προς όλες τις συσκευές) σε όλες τις εξωτερικές πόρτες του δικτύου.

- OpenFlow Forwarding Rules Manager

Διαχειρίζεται βασικούς κανόνες προώθησης , επιλύει τυχόν συγκρούσεις μεταξύ των κανόνων αυτών και τους επικυρώνει. Επικοινωνεί με Southbound επεκτάσεις φορτώνει OpenFlow κανόνες στους διαχειριζόμενους μεταγωγείς.

- OpenFlow Statistics Manager

Υλοποιεί τη συλλογή στατιστικών αποστέλλοντας αιτήσεις για στατιστικά σε όλους τους ενεργούς κόμβους (δηλαδή τους διαχειριζόμενους μεταγωγείς) του ευφυούς δικτύου και αποθηκεύει τις απαντήσεις τους στο operational πεδίο. Επίσης επιστρέφει πληροφορίες στο Northbound API για τα εξής:

- node-connector (την πόρτα του συνδεόμενου μεταγωγέα)
- flow
- meter
- table
- group statistics

- OpenFlow Switch Manager

Παρέχει πληροφορίες για τους κόμβους (μεταγωγείς) τους ευφυούς δικτύου καθώς και τις πόρτες στις οποίες συνδέονται. Όσο ο ελεγκτής ανακαλύπτει νέα δικτυακά στοιχεία, οι

πληροφορίες γι αυτά αποθηκεύονται στο δέντρο δεδομένων του Switch Manager. Μπορεί να χρησιμοποιηθεί το Northbound API για την ανάκτηση των πληροφοριών αυτών.

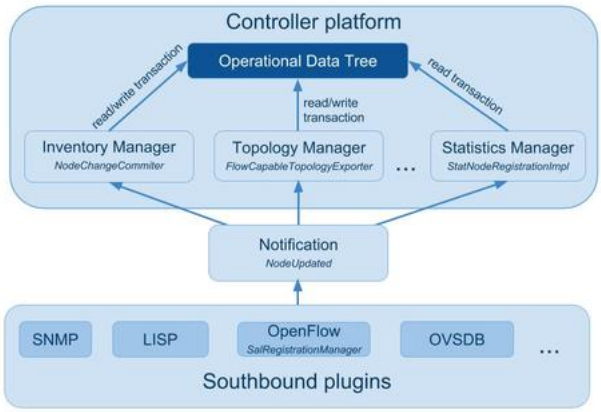
- Topology Processing

Αποθηκεύει και μεταχειρίζεται πληροφορίες σχετικά με τις διαχειριζόμενες δικτυακές συσκευές. Όταν ο ελεγκτής εκκινείται, ο Topology Manager δημιουργεί τον κεντρικό κόμβο της τοπολογίας. Έκτοτε ενημερώνεται για την τοπολογία, για τυχόν ειδοποιήσεις και αλλαγές σ’ αυτό το δέντρο, συμπεραλαμβανομένων όλων των μεταγωγέων που έχουν ανακαλυφθεί.

3.3 MD-SAL (Model Driven-Service Abstraction Layer)

Το MD-SAL παρέχει μια προσέγγιση που επιτρέπει την ενοποίηση των Southbound και Northbound APIs με τις δομές δεδομένων που χρησιμοποιούνται σε διάφορα στοιχεία του ελεγκτή. Στο MD-SAL, κάθε δεδομένο σχετιζόμενο με την κατάσταση λειτουργίας αποθηκεύεται σε μορφή DOM (Document Object Model), γνωστό ως δέντρο δεδομένων. Υπάρχουν δύο είδη δέντρου δεδομένων στον OpenDaylight ελεγκτή:

- Configuration : Χρησιμοποιείται για να αποθηκευτούν πληροφορίες που δεν θα διαγραφτούν μετά και την επανεκκίνηση του ελεγκτή. Επίσης αποθηκεύονται παραμετροποιήσεις που στέλνονται στον ελεγκτή μέσω του REST API
- Operational : Εδώ ο ελεγκτής αποθηκεύει προσωρινές πληροφορίες που προκύπτουν κατά την εκτέλεση των διεργασιών στο δίκτυο, καθώς και πληροφορίες που ο ελεγκτής ανακάλυψε από το Configuration πεδίο δεδομένων



Εικόνα 4: Αλληλεπίδραση μεταξύ του ελεγκτή και της Southbound επέκτασης

Όταν ο μεταγωγέας προσπαθεί να εγγραφτεί στα στοιχεία του ελεγκτή, πρώτα εγγράφεται στην Southbound επέκταση η οποία προκαλεί μια ειδοποίηση της μεθόδου NodeUpdated. Μετά τη λήψη της ειδοποίησης:

- Ο Inventory Manager προσθέτει το νέο κόμβο (μεταγωγέα) στο operational δέντρο δεδομένων.
- Ο Statistics Manager παίρνει τις ανανεωμένες πληροφορίες από τη βάση δεδομένων inventory μέσα από συναλλαγές ανάγνωσης στο operational δέντρο δεδομένων.
- Ο Topology Manager ανανεώνει την αντίστοιχη τοπολογία στο DOM υπόδεντρο.

3.4 REST API

Το REST (Representational State Transfer) αποτελεί αρχιτεκτονική που χρησιμοποιείται σε διαδικτυακές εφαρμογές (WEB development). Τα συστήματα που σχεδιάζονται σε REST αρχιτεκτονική έχουν καλή επίδοση, αξιοπιστία και δυνατότητα κλιμακωσιμότητας ως προς το πλήθος των χρηστών που μπορούν σταδιακά να υποστηρίξουν. Τα RESTful συστήματα επικοινωνούν συνήθως μέσω του HTTP πρωτοκόλλου μέσω των μεθόδων GET, PUT, POST, DELETE τις οποίες χρησιμοποιεί συνήθως το πρόγραμμα περιήγησης για να ανακτήσει και να στείλει δεδομένα σε απομακρυσμένους εξυπηρετητές. [9]

Επίσης, εκτός του προγράμματος περιήγησης, δύναται να ανακτηθούν πληροφορίες και αποσταλούν παραμετροποιήσεις σε RESTful συστήματα μέσω ενός τερματικού και συγκεκριμένα της bash εντολής curl, συνοδευόμενη με τα κατάλληλα ορίσματα (username, κωδικό, URI). Στην περίπτωση που αποστέλλονται παραμετροποιήσεις, μέσω του curl μπορεί κάποιος να στείλει το κατάλληλο κείμενο σε μορφή json ή xml. Οι πληροφορίες που επιστρέφονται από το σύστημα είναι επίσης σε json ή xml μορφή.

Τα πλεονεκτήματα της REST αρχιτεκτονικής σε σχέση με παρόμοιες (όπως η SOAP) είναι :

- Μπορεί να αξιοποιηθεί η cache μνήμη
- Πρόκειται για stateless πρωτόκολλο, δηλαδή υπάρχει ανεξαρτησία μεταξύ της κάθε αίτησης πελάτη και αντίστοιχης απάντησης
- Αξιοποιείται ένα πολυεπίπεδο σύστημα
- Αξιοποιείται μια ενιαία διεπαφή

[10]

Το πρωτόκολλο της REST αρχιτεκτονικής είναι το RESTCONF. Το RESTCONF επιτρέπει πρόσβαση σε δομές δεδομένων στον ελεγκτή. Όπως έχει περιγραφτεί και στο MD-SAL, υπάρχουν δύο ειδών δομές δεδομένων:

- Config: Περιέχει τις πληροφορίες που προστίθενται μέσω του ελεγκτή

- Operational: Περιέχει τις πληροφορίες που προστίθενται μέσω του δικτύου

Ο OpenDaylight έχει REST αρχιτεκτονική οπότε μπορεί να προσφέρει τα παραπάνω. Κάθε αίτηση που του γίνεται μέσω HTTP μεθόδου, πρέπει να γίνεται στο αντίστοιχο URI, που έχει τη μορφή :

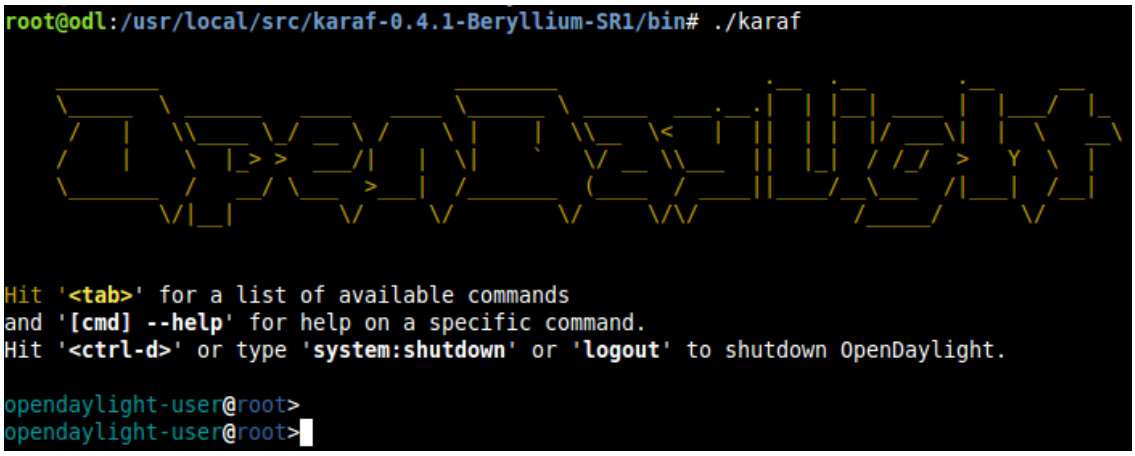
http://<odl-IP>:8181/restconf/operational/<path>/

http://<odl-IP>:8181/restconf/config/<path>/

3.5 Εκκίνηση OpenDaylight

Για την εγκατάσταση του ελεγκτή είναι απαραίτητη η λήψη της αντίστοιχης έκδοσης καθώς και η ύπαρξη των κατάλληλων βιβλιοθηκών JAVA στο μηχάνημα ως:

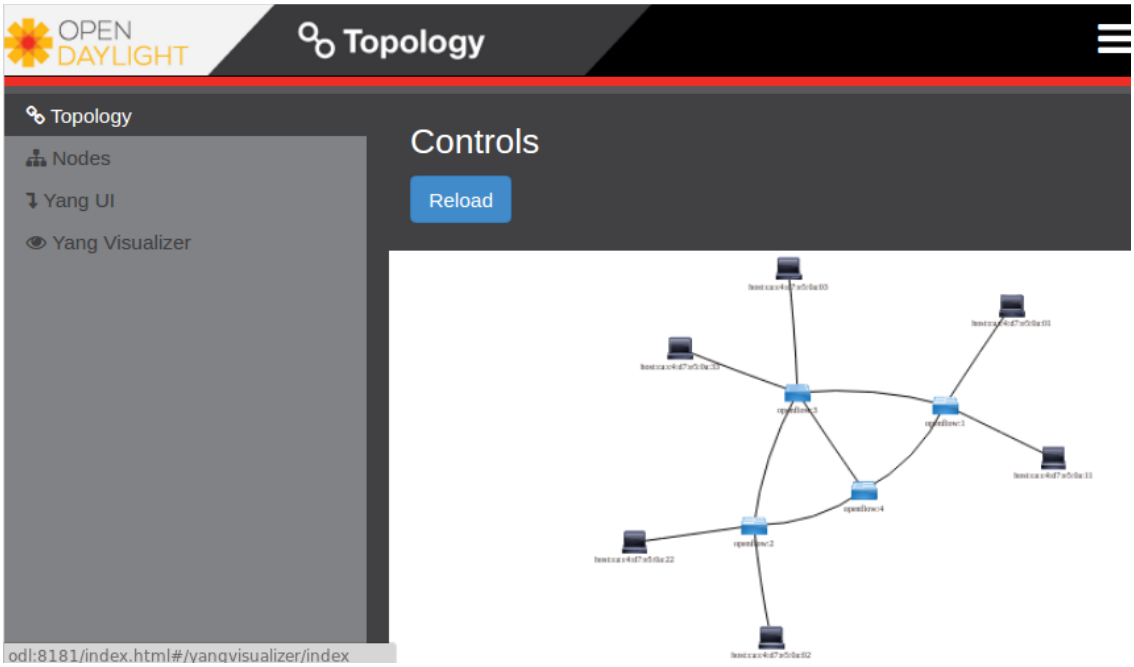
- root@odl: ~# sudo apt-get install maven openjdk-7-jdk openjdk-7-jre
- root@odl: ~# export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-amd64



Εικόνα 5: Εκκίνηση του OpenDaylight

3.6 DLUX

Ο OpenDaylight ελεγκτής δίνει τη δυνατότητα γραφικής αναπαράστασης του δικτύου καθώς και παραμετροποίησής του με gui, εκτός από προγραμματιστικά μέσω του REST API. Μέσω του περιηγητή στο : http://<odl-IP>:8181/index.html#/topology προκύπτει η εξής απεικόνιση ευφυούς δικτύου



Εικόνα 6: Γραφική απεικόνιση τοπολογίας του OpenDaylight μέσω του DLUX

4 Mininet

Το Mininet είναι ένας εξομοιωτής δικτύων που σε έναν Linux πυρήνα μπορεί να εκτελεί και να προσομοιάζει τη λειτουργία εικονικών τερματικών, μεταγωγέων, ελεγκτών και γραμμών. Τα τερματικά στο Mininet τρέχουν Linux λειτουργικό σύστημα και συμπεριφέρονται σαν πραγματικές συσκευές στις οποίες μπορεί να γίνει σύνδεση με ssh (δικτυακό πρωτόκολλο για την ασφαλή μεταφορά δεδομένων) και να γίνει εκτέλεση προγραμμάτων. Τα πακέτα που μεταφέρονται για την εκτέλεση των προγραμμάτων περνούν από διεπαφές Ethernet με χωρητικότητα γραμμής που μπορεί να παραμετροποιηθεί [11]. Οι μεταγωγείς υποστηρίζουν το πρωτόκολλο OpenFlow για υψηλή ευελιξία σε παραμετροποίηση της δρομολόγησης σε Software Defined Networking αρχιτεκτονικές. [12]

Το Mininet έχει ευρεία χρήση από τους μηχανικούς δικτύων για την ανάπτυξη προγραμμάτων και μελέτη των αποτελεσμάτων τους για τους εξής λόγους:

- Δύναται να δημιουργήσει όποια τοπολογία επιθυμεί με ένα απλό Python script.
- Είναι γρήγορο και η έναρξη της λειτουργίας του δικτύου είναι άμεση.
- Είναι έργο Ανοικτού Λογισμικού, που σημαίνει πως μπορεί ο καθένας να συνεισφέρει σε αυτό αλλά και να το παραμετροποιήσει όπως θέλει.
- Υποστηρίζει το OpenFlow πρωτόκολλο κι αποτελεί απλό και οικονομικό τρόπο για πειράματα σε δίκτυα SDN. Μπορεί επομένως κανείς στο Mininet να παραμετροποιήσει την προώθηση πακέτων εφόσον οι μεταγωγείς είναι προγραμματίσιμοι με το OpenFlow
- Μπορεί να τρέξει πραγματικά προγράμματα εφόσον μπορεί να εκτελέσει ό,τι εκτελείται σε Linux και να μεταφέρει αυτά τα προγράμματα σε πραγματικές συσκευές στη συνέχεια
- Δίνει τη δυνατότητα CLI (Command Line Interface) για debugging και εντολές για πειραματισμό στο δίκτυο

Το βασικότερο πλεονέκτημα κατά την προσομοίωση δικτύων στο Mininet είναι πως ο κώδικας που αναπτύσσεται για τους OpenFlow μεταγωγείς και τις υπόλοιπες δικτυακές συσκευές μπορεί να εκτελεστεί σε πραγματικό σύστημα με ελάχιστες αλλαγές. Επομένως ο διαχειριστής μπορεί να χρησιμοποιήσει το Mininet για τον γρήγορο έλεγχο της εφαρμογής του, κι εφόσον είναι έτοιμη, να την εκτελέσει σε πραγματικό δίκτυο.

4.1 Έναρξη Τοπολογίας στο Mininet

Στη συνέχεια θα παρουσιαστεί ένα απλό παράδειγμα τοπολογίας στο Mininet.

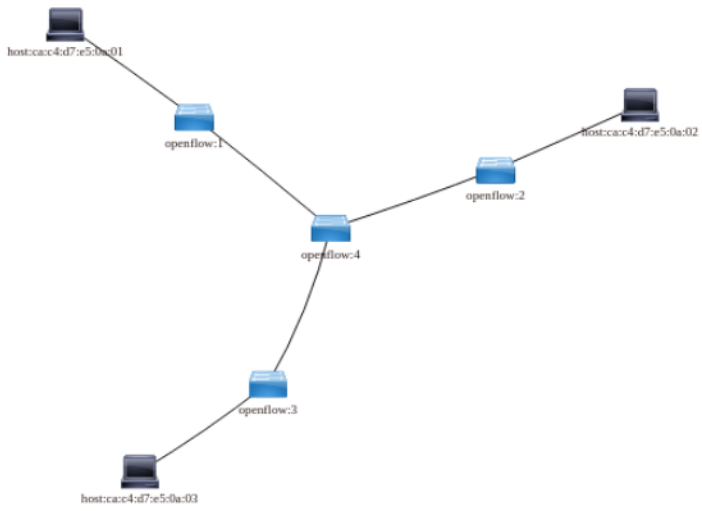
```
from mininet.topo import Topo

class MyTopo( Topo ):

    def __init__( self ):

        # Initialize topology
        Topo.__init__( self )

        host1 = self.addHost( 'h1', ip='192.168.1.101/24' , mac='ca:c4:d7:e5:0a:01' )
        host2 = self.addHost( 'h2', ip='192.168.1.102/24', mac='ca:c4:d7:e5:0a:02' )
        host3 = self.addHost( 'h3', ip='192.168.1.103/24', mac='ca:c4:d7:e5:0a:03' )

        switch1= self.addSwitch( 's1')
        switch2= self.addSwitch( 's2' )
        switch3= self.addSwitch( 's3')
        switch4= self.addSwitch( 's4')

        # Add links
        self.addLink( host1, switch1 )
        self.addLink( host2, switch2 )
        self.addLink( host3, switch3 )
        self.addLink( switch4, switch1 )
        self.addLink( switch4, switch2 )
        self.addLink( switch4, switch3 )

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

- Topo: Η βασική κλάση για τις τοπολογίες στο Mininet
- addHost: προσθέτει ένα τερματικό στην τοπολογία. Η διεύθυνση IP και MAC που τέθηκαν ως ορίσματα, είναι προαιρετικά
- addSwitch: προσθέτει έναν μεταγωγέα στην τοπολογία
- addLink: προσθέτει μια αμφίδρομη γραμμή στην τοπολογία μεταξύ των δύο συσκευών που έχουν τεθεί ως ορίσματα. Μπορούν επίσης να τεθούν κι άλλα ορίσματα όπως η χωρητικότητα της γραμμής, η καθυστέρηση και ο μέγιστος αριθμός πακέτων στην ουρά αναμονής.

Για την εκτέλεση του παραπάνω script στο Mininet, εκτελείται η εντολή:

sudo mn --custom topo.py --topo topology --controller=remote, ip=<odl-IP>

Όπου καθορίζεται η τοπολογία που θα τρέξει, η ονομασία του script στο οποίο βρίσκεται καθώς και η σύνδεση της τοπολογίας με τον ελεγκτή μέσω της διεύθυνσής του.



Εικόνα 7: Παράδειγμα τοπολογίας στο Mininet

5 Data Centers

Τα κέντρα δεδομένων είναι χώροι με υψηλής κλίμακας υπολογιστικά συστήματα που περιέχουν IT υλικοτεχνικό εξοπλισμό όπως εξυπηρετητές (servers) για επεξεργασία δεδομένων, συσκευές αποθήκευσης δεδομένων και συσκευές δικτύωσης για την επικοινωνία εντός και εκτός του κέντρου. Επίσης τα Data Centers περιέχουν εξοπλισμό για τη λειτουργία της υποδομής, όπως συσκευές για την ηλεκτροδότηση των μηχανημάτων καθώς και hardware για την ανάκτηση δεδομένων σε περίπτωση βλάβης. Για τη σωστή και αδιάκοπη λειτουργία όλων αυτών των συσκευών απαιτείται επίσης και εξοπλισμός για τη διατήρηση κατάλληλων συνθηκών περιβάλλοντος (θερμοκρασίας και υγρασίας). [13]

Η λειτουργία των αναφερθέντων μηχανημάτων συνιστούν τα κέντρα δεδομένων ως μια σημαντική υποδομή σε κατανάλωση ηλεκτρικής ενέργειας. Επίσης, αποθηκεύουν δεδομένα που μπορεί ανά πάσα στιγμή να χρειαστούν ανάκτηση λόγω της αίτησης πελάτη, οπότε πρέπει οι συσκευές να βρίσκονται πάντα σε ετοιμότητα, είτε υπάρχουν αιτήσεις από πελάτες είτε όχι. Η λειτουργία τους είναι αδιάκοπη λοιπόν.

Οι περισσότερες και μεγαλύτερες επιχειρήσεις (όπως Facebook, Google, Amazon) κάνουν χρήση cloud computing, το οποίο απαιτεί την ύπαρξη Data Centers για την κατανεμημένη και αδιάλειπτη αποθήκευση, ανάκτηση και επεξεργασία δεδομένων. Ομοίως και η ανάγκη για μεγαλύτερη χρήση του διαδικτύου -λόγω της αύξησης των χρηστών και των ευφυών συσκευών- καθώς και οι ηλεκτρονικές συναλλαγές που τείνουν να αντικαταστήσουν τις παλιότερες μεθόδους , καθιστούν τα Data Centers, το νέο βασικό άξονα για την ανάπτυξη της οικονομίας αλλά και το νέο αναπτυσσόμενο πεδίο μελέτης για του επιστήμονες. Τόσο για τη βελτιστοποίηση των λειτουργιών των Data Centers, όσο και για την εύρεση τρόπων για την εξοικονόμηση ενέργειας για την ηλεκτροδότηση τους, αφού αναμένεται στο μέλλον να παίξουν καθοριστικό ρόλο στην αποφυγή των καταστροφών λόγω της υπερθέρμανσης του πλανήτη.

Για την Αμερική συγκεκριμένα -στην οποία λειτουργούν οι μεγαλύτερες εταιρίες που κάνουν χρήση των Data Centers- υπολογίζεται πως σε ένα έτος καταναλώθηκαν πάνω από 90 δις kWh για τη λειτουργία των Data Centers. Εκτιμάται δε, πως η κατανάλωση αυτή μέχρι το 2020 θα αυξάνεται σε ρυθμούς 7% κάθε χρόνο και θα αγγίξει

τα 140 δις kWh ετησίως ή 13 δις $ κόστος σε κατανάλωση αυτού του ρεύματος, μόνο για την Αμερική. Για τον πλανήτη τα ποσά αυτά σημαίνουν καύση περίπου 150 τόνων άνθρακα, που καθιστά τεράστια οικολογική επιβάρυνση και εντείνουν την ανάγκη εύρεσης τρόπων για τον περιορισμό αυτής της κατανάλωσης.

5.1 Άλλοι μέθοδοι εξοικονόμησης ενέργειας

Έχουν ήδη γίνει μελέτες που να κινούνται σε αυτό το επίπεδο, στην ελάττωση της ηλεκτρικής ενέργειας που απαιτείται για τη λειτουργία των Data Centers, και συνεπώς στην προστασία του πλανήτη. Είναι πραγματικότητα μάλιστα, πως ενώ οι συσκευές συνεχίζουν να κοστίζουν λιγότερο ακολουθώντας το νόμο του Moore, το ηλεκτρικό ρεύμα τείνει να γίνει ακριβότερο και υπολογίζεται ότι θα κοστίζει 140€/ Mwh το 2020 από τα 110€/ Mwh που στοίχιζε το 2010. [14]

Η μεγαλύτερη πρόοδος έχει γίνει πάνω στην επάρκεια και αποτελεσματικότητα του εξοπλισμού προκειμένου να ελαττωθεί ο χρόνος αδρανοποίησης τους. Πρόβλημα αποτελεί ωστόσο η χρησιμοποίηση των εξυπηρετητών και το γεγονός πως βρίσκονται διαρκώς σε ετοιμότητα χωρίς συχνά να παρέχουν κάποιο έργο στην εξυπηρέτηση αιτήσεων πελατών.

Ο μέσος εξυπηρετητής που λειτουργεί, προσφέρει μόνο το 12-18% των δυνατοτήτων του, σε χωρητικότητα και λειτουργίες, παραμένοντας ενεργός διαρκώς και καταναλώνοντας ενέργεια εν αναμονή λήψης δεδομένων ή αίτησης για κάποια συναλλαγή. Φτάνοντας σε υψηλά ποσοστά χρησιμοποίησης των εξυπηρετητών -άνω του 50% - , θα αυξανόταν το κόστος εγκατάστασης αλλά το κόστος αυτό μπορεί να αντισταθμιστεί με το κέρδος από την εξοικονόμηση ενέργειας που θα μπορούσε να επιτευχθεί. Όσο λιγότερο χρησιμοποιείται ο εξυπηρετητής για κάποια λειτουργία, τόσο περισσότερο μειώνεται η αποτελεσματικότητά του, ενώ καταναλώνει ρεύμα. Με την ίδια κατανάλωση ρεύματος, θα μπορούσαν να γίνουν πολλαπλάσιες συναλλαγές αν

αντιμετωπιζόταν το πρόβλημα της χαμηλής χρησιμοποίησης των εξυπηρετητών, που αποτελεί και το μείζον θέμα που ασχολούνται οι ερευνητές για την ελάττωση των εκπομπών στο περιβάλλον λόγω της ηλεκτροδότησης των Data Centers.

Εκτός αυτού, υπάρχουν εξυπηρετητές που δεν παρέχουν καμία απολύτως λειτουργικότητα αλλά παραμένουν στο σύστημα καταναλώνοντας ενέργεια 24 ώρες τη μέρα αδιαλείπτως, οι λεγόμενοι “comatose servers”. Οι “comatose” ή αλλιώς “zombie” servers προκύπτουν συνήθως μετά το πέρας ή την αλλαγή κάποιου έργου ή και λόγω της έλλειψης ενδιαφέροντος των χρηστών για την εφαρμογή για την οποία χρησιμοποιούνταν ο server. Οι εξυπηρετητές αυτοί αποτελούν συνήθως το 20-30% των εξυπηρετητών κάθε Data Center. Επομένως ένας τρόπος εξοικονόμησης ενέργειας είναι η απομάκρυνση και αντικατάσταση αυτών των ανενεργών εξυπηρετητών. Το πρόβλημα είναι πως συχνά οι διαχειριστές των Data Centers δεν γνωρίζουν τους ιδιοκτήτες των εξυπηρετητών, γι αυτό και είναι δύσκολο να δώσουν την έγκριση για την απομάκρυνση των ανενεργών, με επιφύλαξη μήπως δημιουργήσουν πρόβλημα σε κάποιο σημαντικό έργο. Το ποσοστό των εξυπηρετητών που συνήθως οι διαχειριστές δεν γνωρίζουν σε ποιον ανήκουν ανέρχεται σε 15-30%.

Μία ακόμη καινοτομία που μπορεί να βοηθήσει τους διαχειριστές των Data Centers να ελαττώσουν την κατανάλωσή τους σε ρεύμα είναι η επέκταση της τεχνικής της Εικονικοποίησης (Virtualization). Η τεχνική αυτή επιτρέπει τη δημιουργία εικονικών μηχανημάτων, όπως hardware, συσκευών δικτύωσης, μνήμης και λειτουργικών συστημάτων. Ενώ παλιότερα μπορούσε να εγκαταστηθεί μόνο ένα λειτουργικό σύστημα σε έναν υπολογιστή, τώρα στα εικονικά μηχανήματα μπορούν να λειτουργήσουν πολλά λειτουργικά συστήματα στο ίδιο μηχάνημα. Πλέον μπορούν να λειτουργήσουν πολλοί εικονικοί εξυπηρετητές πάνω σε έναν φυσικό εξυπηρετητή, οπότε επενδύοντας σε λιγότερα φυσικά μηχανήματα μπορούν να έχουν οι εταιρίες τις ίδιες υπηρεσίες με παλιότερα. Αυτό αποφέρει μείωση των εξόδων, ελάττωση των μηχανημάτων τόσο για τις διεργασίες των Data Centers όσο και την υποστήριξή του (συστήματα διατήρησης συνθηκών περιβάλλοντος) και επομένως σημαντική ελάττωση στην κατανάλωση ενέργειας. Ωστόσο η Εικονικοποίηση δεν έχει πάρει ακόμα τις διαστάσεις που θα μπορούσε να έχει, αφού δεν έχουν αντικαταστηθεί πλήρως τα μηχανήματα με εικονικά.

5.2 Εξοικονόμηση ενέργειας μέσω των δικτυακών συσκευών

Ακόμα και αν τα μηχανήματα δικτύωσης στα Data Centers δεν καταναλώνουν τόση ενέργεια όσο οι εξυπηρετητές, εξακολουθούν να καταναλώνουν τεράστια ποσά ενέργειας, γι αυτό και οι Μηχανικοί Δικτύων προσανατολίζονται προς αυτήν την κατεύθυνση προκειμένου να βρουν τρόπους να ελαττώσουν τις εκπομπές από τα Data Centers προς το περιβάλλον. Πάνω στην ιδέα αυτή βασίζεται η παρούσα εργασία.

Έχουν ήδη γίνει έρευνες σε ερευνητικά κέντρα του εξωτερικού πάνω στο Energy Aware Routing, δηλαδή στην εύρεση βέλτιστων μεθόδων δρομολόγησης για να εξοικονομηθεί ηλεκτρική ενέργεια στα Data Centers. Οι μέθοδοι αυτοί χρησιμοποιούν την τοπολογία του εκάστοτε δικτύου και την κίνηση του, και με τη βοήθεια ευριστικών μεθόδων καθορίζουν ποιες δικτυακές συσκευές υπολειτουργούν και θα πρέπει να απενεργοποιηθούν [15] . Σε αυτήν την κατεύθυνση κινείται η μέθοδος “ElasticTree” κατά την οποία ανάλογα το που υπάρχει μεγαλύτερο φορτίο στο δίκτυο, με ευριστικές, άπληστες και μεθόδους πρόβλεψης, βρίσκονται ποιες δικτυακές συσκευές (μεταγωγείς, δρομολογητές, γραμμές) πρέπει να απενεργοποιηθούν. Προκύπτει ένα Συνδετικό Δέντρο για τα τερματικά (εξυπηρετητές) μεν, αλλά που δεν περιλαμβάνει όλες τις συσκευές του δικτύου. Αυτή η μέθοδος είναι χρήσιμη σε περίπτωση χαμηλής κινητικότητας στο δίκτυο. Με τη μέθοδο αυτή υπολογίζεται ελάττωση της κατανάλωσης ενέργειας της τάξεως των 25-40%. [16]

Παράλληλα, εκτός από ερευνητικά κέντρα του εξωτερικού, πραγματοποιούνται έρευνες και σε ελληνικά κέντρα. Συγκεκριμένα στο Εθνικό Μετσόβιο Πολυτεχνείο σε συνεργασία με το Εθνικό Δίκτυο Έρευνας & Τεχνολογίας (GRNET) μελετήθηκαν πολιτικές δρομολόγησης για ενεργειακή εξοικονόμηση που εκτιμάται πως μπορούν να εξοικονομήσουν πάνω από το 50% της ενέργειας που καταναλώνεται απενεργοποιώντας τις κατάλληλες δικτυακές συσκευές.

Η διαφορά αυτών των μελετών από αυτές του εξωτερικού είναι πως επικεντρώθηκαν στην απενεργοποίηση γραμμών, δηλαδή μόνο των διεπαφών των

μεταγωγέων και όχι των ίδιων των συσκευών. Οι γραμμές καταναλώνουν συνολικά πάνω από το 70% της ενέργειας που χρειάζεται ο δρομολογητής. Το να απενεργοποιηθεί τελείως η συσκευή (όπως ορίζουν οι υπόλοιπες έρευνες) θα επέφερε μεγαλύτερα κέρδη εν μέρει, είναι πιο δύσκολο να εφαρμοστεί όμως γιατί ακόμα και οι δρομολογητές που ανήκουν στο backbone internet (οι βασικοί δρομολογητές για τη μεταφορά κίνησης στο διαδίκτυο), συχνά έχουν διεπαφές που συνδέονται με τοπικούς πελάτες και δεν μπορούν να απενεργοποιηθούν.

Για τη εξαγωγή των αποτελεσμάτων λαμβάνονται υπόψιν η τοπολογία του δικτύου, η κίνηση πάνω στις γραμμές του -συγκεκριμένα το bandwidth από όλους τους μεταγωγείς-πηγής προς όλους τους μεταγωγείς-προορισμού – καθώς και ενεργειακά στοιχεία για την κατανάλωση ρεύματος των συσκευών και τη χωρητικότητα των γραμμών.

Ακόμα και για συσκευές που καταναλώνουν σχεδόν το μέγιστο της ενέργειας που θα μπορούσαν, σχεδόν το 50% από αυτήν την ενέργεια απαιτείται για τις γραμμές και αυτό θα μπορούσε να ελαττωθεί απενεργοποιώντας με ευφυή τρόπο τις σωστές διεπαφές. Η εξοικονόμηση αυτή κυμαίνεται στο 50% για τις ώρες που δεν υπάρχει πολλή κίνηση και φτάνει στο 70% τις ώρες αιχμής. Το τελευταίο ποσοστό μπορεί να αγγίξει το 95% αν χρησιμοποιηθούν ενεργειακές συσκευές με γραμμική κατανάλωση ως προς την κίνηση στο δίκτυο. Ο αλγόριθμος που μελετήθηκε είναι υπεύθυνος για την εύρεση των διεπαφών που θα πρέπει ευφυώς να απενεργοποιηθούν προκειμένου να υπάρξει βέλτιστη δρομολόγηση στο δίκτυο, και είναι ευθύνη του διαχειριστή συστήματος στη συνέχεια να αυτοματοποιήσει τη διαδικασία και να ενεργοποιήσει/ απενεργοποιήσει τις κατάλληλες διεπαφές. [17]

Ο αλγόριθμος που μελετήθηκε γράφτηκε σε CPLEX με τη βοήθεια του mixed integer programming.

6 Ανάπτυξη εφαρμογής ελέγχου του δικτύου

Η αύξηση των χρηστών του διαδικτύου, η διαπίστωση πως οι περισσότεροι συνδέονται στο διαδίκτυο με περισσότερες από μία συσκευές, καθώς και η ανάγκη των επιχειρήσεων για υποστήριξή τους από κέντρα δεδομένων (data centers) έχουν καταστήσει το διαδίκτυο ως το πλέον απαραίτητο εργαλείο για όλους. Αυτή η έκρηξη της χρήσης του όμως έχει δημιουργήσει πολλά ερωτήματα για το κατά πόσο τα μηχανήματα που είναι απαραίτητα για τη λειτουργία του καταναλώνουν ενέργεια με συνετό και οικολογικό τρόπο. Από έρευνες που διεξήχθησαν το 2011, υπολογίζεται πως για τη λειτουργία του διαδικτύου καταναλώνονται 84 με 143 GigaWatt ηλεκτρικής ενέργειας κάθε χρόνο, ή περίπου το 3.6% - 6.2% της συνολικής ενέργειας παγκοσμίως [16] . Πολύ μεγάλο μέρος αυτού του ποσοστού ενέργειας καταναλώνεται στα data centers. Είναι ευκαιρία λοιπόν τώρα με την καινοτομία και την εξάπλωση του SDN, να εκμεταλλευτούμε τη δυνατότητα του κεντρικού απομακρυσμένου ελέγχου που μας δίνει και να προωθήσουμε πολιτικές δρομολόγησης με σκοπό την εξοικονόμηση ενέργειας στα δίκτυα νέας γενιάς που έρχονται στο προσκήνιο.

Η ιδέα είναι η δημιουργία μιας εφαρμογής -σε ρόλο διαχειριστή δικτύου- που επικοινωνεί εξωτερικά με τον ελεγκτή και υλοποιεί έναν μηχανισμό συλλογής στατιστικών τα οποία αποτελούν τα απαραίτητα στοιχεία για την εύρεση αποδοτικής πολιτικής δρομολόγησης μέσω κατάλληλου αλγορίθμου.

6.1 Απαιτούμενα εργαλεία

6.1.1 Λογισμικό CPLEX

Έχει αναπτυχθεί από την IBM ένα πακέτο λογισμικού για βελτιστοποιήσεις, κατάλληλο για επίλυση προβλημάτων γραμμικού προγραμματισμού, το IBM ILOG CPLEX Optimization Studio ή εν συντομία CPLEX. Το CPLEX ενδείκνυται για προβλήματα απόφασης δίνοντας τη δυνατότητα για διευκόλυνση των πράξεων και συνεπώς για μικρότερους χρόνους εκτέλεσης. Συγκεκριμένα δίνει τη δυνατότητα επεξεργασίας απαιτητικών προβλημάτων σε κατανεμημένα συστήματα μέσω παράλληλων αλγορίθμων. Και στη δική μας περίπτωση γίνεται εκτέλεση του προγράμματος σε παράλληλα νήματα διότι για πολύ μεγάλα δίκτυα η επεξεργασία των στοιχείων ήταν χρονοβόρα και αυτό καθυστερούσε τη λήψη απόφασης για την αλλαγή κατάστασης των γραμμών του δικτύου. [18]

Η εκτέλεση προγραμμάτων στο CPLEX απαιτεί την εγκατάσταση του αντίστοιχου πακέτου της IBM και πραγματοποιείται στο unix shell μέσω της bash εντολής oplrun. Οπότε η ενσωμάτωση του αλγορίθμου βέλτιστης ενεργειακά δρομολόγησης με τις διαχειριστικές εντολές του ελεγκτή στο ευφυές δίκτυο, ήταν εφικτή.

6.1.2 Η βιβλιοθήκη jq

Όλες οι πληροφορίες που χρειάστηκαν από τον ODL, κατέβηκαν στη μορφή json (εναλλακτικά θα μπορούσε να είχε χρησιμοποιηθεί και xml). Για τη μορφοποίηση των πολύ μεγάλων αρχείων, την επεξεργασία και τη μαζική εξαγωγή συγκεκριμένων στοιχείων χρησιμοποιήθηκε η εντολή jq, που αποτελεί ουσιαστικά ένα φίλτρο για json αρχεία. Συνέβαλε τόσο απ’την πρακτική μεριά της εφαρμογής όσο και στη βελτιστοποίηση της από άποψη χρόνου και χώρου.

6.1.3 Απαιτήσεις OpenDaylight

Κατά τη μελέτη του ελεγκτή OpenDaylight για την εκπόνηση αυτής της διπλωματικής εργασίας, ανανεώθηκαν οι εκδόσεις του. Αρχικά έγινε χρήση της έκδοσης Lithium, ενώ στη συνέχεια έγινε εγκατάστασης της τελευταίας έκδοσης Beryllium για τυχόν αναβαθμίσεις.

Για να τρέξει ο ελεγκτής θα πρέπει να γίνει εγκατάσταση κάποιων επεκτάσεων (features) προκειμένου να υποστηριχτεί η μέθοδος l2switch, το mdsal μοντέλο, το restconf πρωτόκολλο, η δυνατότητα σύνδεσης και η υποστήριξη της γραφικής απεικόνισης και παραμετροποίησης του δικτύου με το dlux. Η εγκατάσταση γίνεται ως εξής:

opendaylight-user@root> feature:install <feature>

Τα features που απαιτείται να εγκαταστιθούν είναι τα εξής:

- odl-dlux-all odl-dlux-core odl-dlux-node odl-dlux-yangui odl-dlux-yangvisualizer
- odl-restconf odl-restconf-noauth
- odl-aaa-api odl-aaa-authn
- odl-config-persister odl-config-startup odl-config-netty odl-config-api odl-config-core odl-config-manager
- odl-openflowplugin-southbound odl-openflowplugin-flow-services odl-openflowplugin-nsf-services odl-openflowplugin-nsf-model odl-openflowplugin-flow-services-rest odl-openflowplugin-flow-services-ui odl-openflowplugin-app-config-pusher odl-openflowplugin-app-lldp-speaker
- odl-l2switch-switch odl-l2switch-switch-rest odl-l2switch-switch-ui odl-l2switch-hosttracker odl-l2switch-addresstracker odl-l2switch-arphandler odl-l2switch-loopremover odl-l2switch-packethandler
- odl-mdsal-common odl-mdsal-broker-local odl-mdsal-xsql odl-mdsal-clustering-commons odl-mdsal-distributed-datastore odl-mdsal-remoterpc-connector odl-mdsal-broker odl-mdsal-clustering odl-mdsal-apidocs odl-mdsal-binding-runtime odl-mdsal-binding-base odl-mdsal-models
- ssh war http config

6.2 Τοπολογία

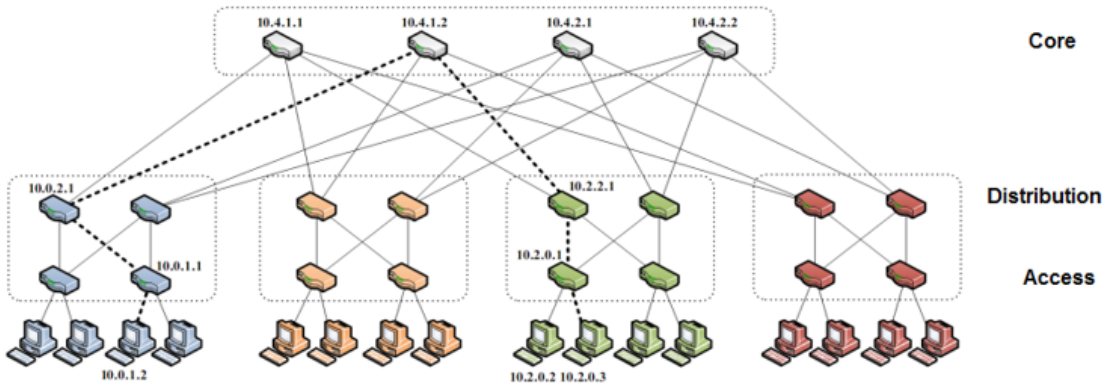
6.2.1 Τοπολογία των Data Centers

Η τοπολογία που χρησιμοποιείται για τα Data Centers είναι της μορφής FatTree κατά την οποία τα τερματικά -ή στην περίπτωση μας- οι εξυπηρετητές βρίσκονται στην κατώτερη στοιβάδα του δέντρου. Το Data Center αποτελείται από:

- Servers (Εξυπηρετητές) : Τοποθετημένοι σε Racks και συνδεόμενοι στο διαδίκτυο μέσω μεταγωγέων
- Access/Edge Switches : Το επίπεδο των μεταγωγέων που συνδέεται άμεσα με τους εξυπηρετητές
- Distribution/Aggregation Switches : Το επίπεδο των μεταγωγέων που συνδέει τους Access μεταγωγείς και τους παρέχει συνδεσιμότητα στο διαδίκτυο
- Core Switches : Το ανώτερο επίπεδο μεταγωγέων που συνδέεται μόνο με Distribution μεταγωγείς και παρέχει συνδεσιμότητα εντός του Data Center αλλά και εκτός αυτού, σε άλλες γεωγραφικές περιοχές για την πρόσβαση του Data Center στο διαδίκτυο [19]

Η μορφή αυτή τριών επιπέδων προτιμάται διότι προσδίδει κλιμακωσιμότητα και εξισορρόπηση της κίνησης στο δίκτυο.

Παρακάτω παρουσιάζεται η μορφή που έχουν οι συσκευές σε ένα Data Center

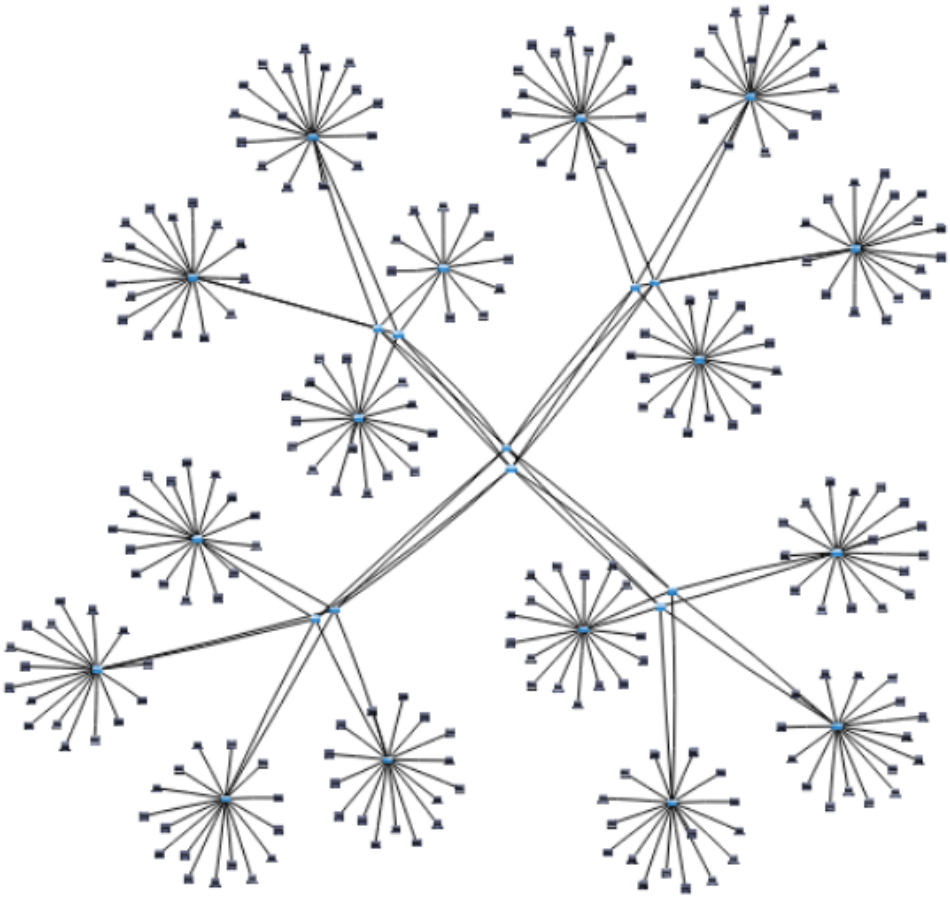


Εικόνα 8: Επισκόπηση Αρχιτεκτονικής Fattree σε Data Center

6.2.2 Τοπολογία Εφαρμογής

Η τοπολογία που επιλέχθηκε να χρησιμοποιηθεί για την εφαρμογή είναι της μορφής FatTree, προσομοιάζοντας έτσι τη μορφή που έχουν οι τοπολογίες στα Data Centers, αφού αποτελούν τα σημεία στα οποία καταναλώνονται τα μεγαλύτερα ποσοστά ενέργειας του διαδικτύου και σε αυτά θα είχε περισσότερη απήχηση η εφαρμογή αυτού του είδους των βελτιστοποιήσεων.

Συγκεκριμένα, η τοπολογία είναι της μορφής:



Εικόνα 9: Τοπολογία δικτύου εφαρμογής (OpenDaylight DLUX)

Η εικόνα ελήφθη μέσω browser από την εφαρμογή DLUX του ODL θέτοντας τα στοιχεία του διαχειριστή στο url: http://<odl-IP>:8181/index.html#/topology

Στην παραπάνω τοπολογία υπάρχουν συνολικά 26 switches και 256 τερματικά. Συγκεκριμένα έχουν τοποθετηθεί: 2 core switches, 8 distribution switches , 16 access switches με το κάθε access switch να συνδέεται άμεσα με 16 τερματικά.

Να επισημανθεί πως η παραπάνω είναι μια τοπολογία που τυχαίως επιλέχθηκε, και δεν εξαρτάται η λειτουργία της εφαρμογής από αυτή καθώς ό,τι πληροφορία απαιτεί, την λαμβάνει από τον ελεγκτή κάθε φορά που εκκινείται νέο δίκτυο.

6.3 Αρχιτεκτονική Εφαρμογής

Για την εφαρμογή αξιοποιείται η REST αρχιτεκτονική του ODL και γίνεται χρήση του REST api στο http://<odl-IP>:8181/restconf/ . Στον παρακάτω πίνακα φαίνονται τα μέρη του REST api που χρειάστηκαν καθώς και οι αντίστοιχες μέθοδοι που χρησιμοποιήθηκαν και τα στοιχεία που εξήχθησαν από τον ελεγκτή ή τα στοιχεία που του στάλθηκαν :

Μέθοδος	URL	Στοιχεία που εξήχθησαν ή στάλθηκαν	Σχόλια
GET	http://<ODL-IP>:8181/restconf/operational/network-topology:network-topology/	*Λήψη:* i) link-id ii) source-node iii) source-tp iv) dest-tp	Τα id για το συγκεκριμένο link από τα οποία αργότερα θα εξαχθούν κατάλληλες πληροφορίες όπως : switch-ID, MAC διευθύνσεις, port-ID
GET	http://<ODL-IP>:8181/restconf/operational/opendaylight-inventory:nodes/openflow:<switchID>/node-connector/openflow:<switchID>:<port-ID>/	*Λήψη:* flow-node-inventory:name	Το όνομα του συγκεκριμένου interface
GET	http://<ODL-IP>:8181/restconf/operational/opendaylight-inventory:nodes/openflow:<switchID>/node-connector/openflow:<switchID>:LOCAL	*Λήψη:* flow-node-inventory:name	Το όνομα του συγκεκριμένου switch
GET	http://<ODL-IP>:8181/restconf/operational/opendaylight-inventory:nodes/openflow:<switchID>/table/0	*Λήψη:* i) flow-id ii) hash	Στοιχεία που θα χρειαστούν για την αναγνώριση της κάθε ροής
GET	http://<ODL-IP>:8181/restconf/operational/opendaylight-inventory:nodes/openflow:<switchID>/table/0/flow/<flow-ID>	*Λήψη:* byte-count	Ο μετρητής για τα bytes που μεταφέρθηκαν με τη συγκεκριμένη ροή
PUT	http://<ODL-IP>:8181/restconf/operational/opendaylight-inventory:nodes/openflow:<switchID>/table/0/flow/<flow-ID>	*Αποστολή:* Το xml με τα στοιχεία της ροής όπως θέλουμε να	Δημιουργία ροής της επιθυμίας μας

		την παραμετροποιήσουμε	
DELETE	http://<ODL-IP>:8181/restconf/operational/opendaylight-inventory:nodes/openflow:<switchID>/table/0/flow/<flow-ID>	*Αποστολή:* Το xml αρχείο που συνιστά τη συγκεκριμένη ροή	Διαγραφή της συγκεκριμένης ροής
DELETE	http://<ODL-IP>:8181/restconf/config/opendaylight-inventory:nodes/	*Αποστολή:* Όλο το περιεχόμενο του config state	Διαγραφή όλου του config state

Πίνακας με τις μεθόδους που χρησιμοποιήθηκαν στο REST api

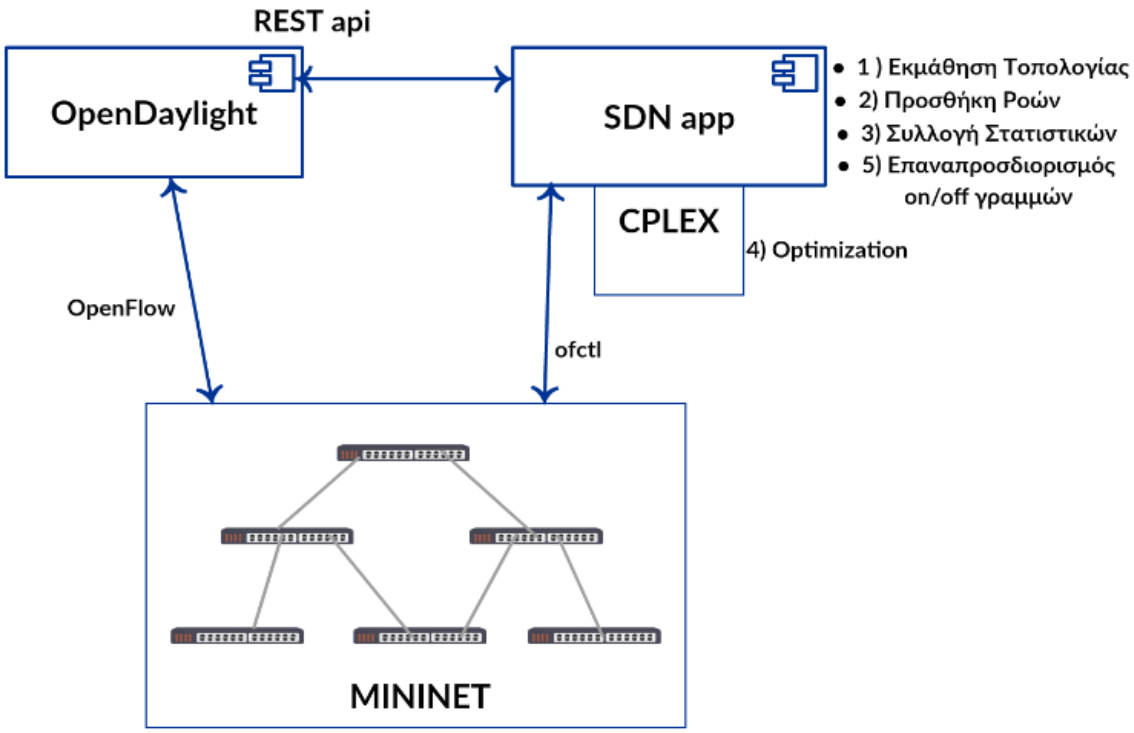
Εν γένει, τροποποίηση γίνεται μόνο στο config state, το οποίο ενημερώνει αυτόματα το operational state του ελεγκτή. Από το operational state, ο διαχειριστής λαμβάνει τις πληροφορίες που επιθυμεί.

Ως γλώσσα προγραμματισμού επιλέχθηκε η python διότι μπορεί εύκολα κανείς μέσω αυτής να εκτελέσει bash εντολές που μας ήταν απαραίτητες. Συνδυάστηκε η python με bash shell scripting προκειμένου να εκμεταλλευτούμε το REST api του ODL μέσω της εντολής curl. Η εφαρμογή δεν απαιτεί την ύπαρξη browser, ούτε καν την ύπαρξη γραφικού περιβάλλοντος στο μηχάνημα που εκτελείται. Μπορεί εύκολα ο διαχειριστής απομακρυσμένα, με ένα εικονικό μηχάνημα (VM) και ένα terminal να εκτελέσει την εφαρμογή ελέγχοντας το δίκτυο του. Απαραίτητη είναι η επαλήθευση ταυτότητας και αδειοδότηση του χρήστη με τα στοιχεία του διαχειριστή (username, password) καθώς και η εγκατάσταση των βιβλιοθηκών που βρίσκονται στις επικεφαλίδες των python αρχείων και του cplex.

Για την προσθήκη καινούριων ροών που θα περιγραφτεί παρακάτω, εκμεταλλευτήκαμε τις δυνατότητες του Openflow πρωτοκόλλου, το οποίο και είναι υπεύθυνο για κάθε επικοινωνία του ελεγκτή με τα switches. Ουσιαστικά το REST api είναι υπεύθυνο για την εξωτερική επικοινωνία που έχει ο διαχειριστής-εφαρμογή με τον

ελεγκτή και το Openflow είναι υπεύθυνο για την επικοινωνία εντός του δικτύου (μεταξύ ελεγκτή - switches) .

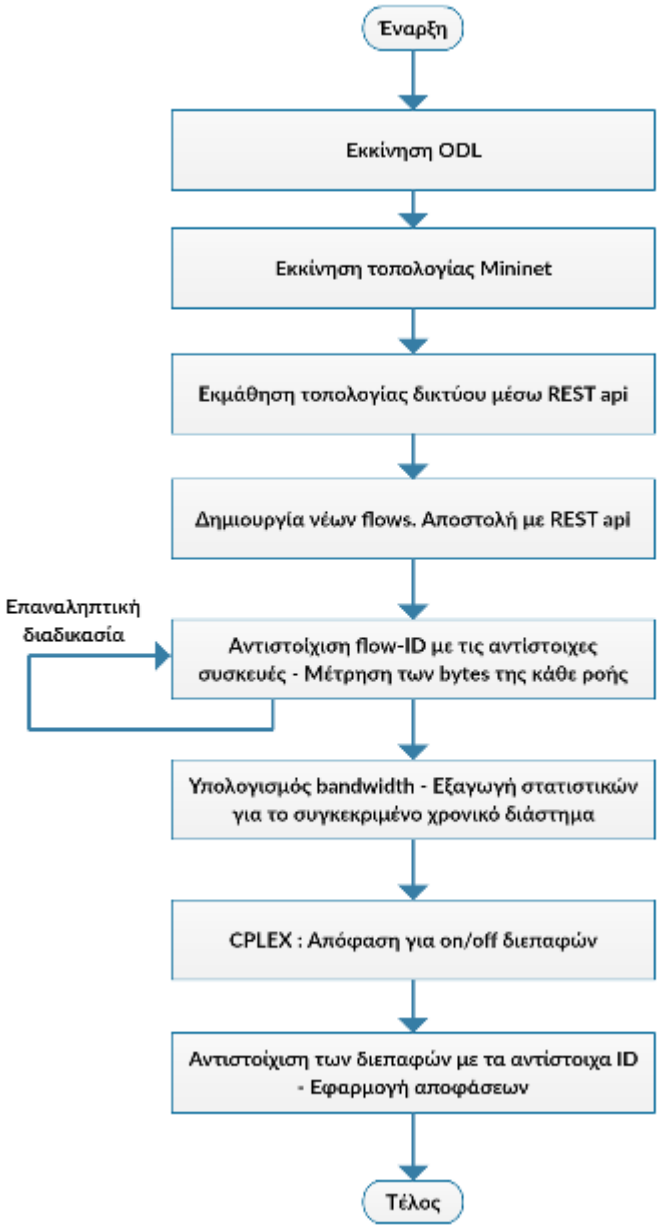
Συνοψίζοντας, παρουσιάζεται η αρχιτεκτονική της εφαρμογής στην παρακάτω εικόνα.



Εικόνα 10: Αρχιτεκτονική Συστήματος

Η εικόνα αφορά σε γενική αρχιτεκτονική του συστήματος, της συνδεσμολογίας και της επικοινωνίας μεταξύ των μηχανημάτων.

6.3.1 Διάγραμμα Ροής – Σύνοψη εφαρμογής



Εικόνα 11: Διάγραμμα ροής της εφαρμογής

6.4 Εκμάθηση Τοπολογίας Συστήματος

Σε πρώτη φάση, μέσω του REST api εκμαιεύουμε την τοπολογία του εκάστοτε δικτύου.

Συγκεκριμένα μέσω της εντολής curl (εντολή bash) , με τις κατάλληλες επικεφαλίδες και μέσω της μεθόδου GET στο εξής URL :
http://<odl-IP>:8181/restconf/operational/network-topology:network-topology/
λαμβάνουμε σε json (ή xml) μορφή πληροφορίες της μορφής:

```
<link>
        <link-id>openflow:3000:1</link-id>
        <destination>
                <dest-tp>openflow:2000:3</dest-tp>
                <dest-node>openflow:2000</dest-node>
        </destination>
        <source>
                <source-tp>openflow:3000:1</source-tp>
                <source-node>openflow:3000</source-node>
        </source>
</link>
```

Το παραπάνω αφορά σε link μεταξύ 2 switches.

Από το πεδίο source-tp μπορεί να εξαχθή το id του switch που προέρχεται η κίνηση, καθώς και η πόρτα που συνδέεται το συγκεκριμένο link. Από το dest-tp μπορούν να εξαχθούν τα αντίστοιχα στοιχεία για το switch στο οποίο καταλήγει το το link.

```
<link>
        <link-id>openflow:3000:9/host:9a:2a:f8:ec:5b:20</link-id>
```

```
        <destination>
                <dest-tp>host:9a:2a:f8:ec:5b:20</dest-tp>
                <dest-node>host:9a:2a:f8:ec:5b:20</dest-node>
        </destination>
        <source>
                <source-tp>openflow:3000:9</source-tp>
                <source-node>openflow:3000</source-node>
        </source>
</link>
```

Το παραπάνω αφορά σε link μεταξύ switch - host.

Εδώ επειδή το link καταλήγει σε κάποιο host του switch, από το dest-tp μπορεί να εξαχθεί η MAC διεύθυνση του συγκεκριμένου τερματικού.

Με τη βοήθεια της βιβλιοθήκης jq και χρήση της με κατάλληλο τρόπο κάθε φορά, γίνεται λήψη του αρχείου json που αφορά στην τοπολογία όλου του συστήματος. Συγκεκριμένα λαμβάνονται τα:

- link-id
- source-node
- source-tp
- dest-tp

Στη συνέχεια από τα παραπάνω εξάγονται τα κατάλληλα στοιχεία για την εποπτεία της τοπολογίας (id, MAC address,port number) τα οποία παραμένουν σταθερά όσο υπάρχει το δίκτυο και χρησιμοποιούνται για τις παραμετροποιήσεις και μετρήσεις που θα περιγραφούν παρακάτω.

6.5 Μηχανισμός Συλλογής Στατιστικών

Για να εκτελεστεί ο αλγόριθμος στο cplex, απαιτεί το εύρος ζώνης (bandwidth) από όλους τους μεταγωγείς (switches), προς όλους τους υπόλοιπους. Το πρόβλημα που υπήρχε και έπρεπε να λυθεί είναι πως ο ODL (όπως και κάθε ελεγκτής) δίνει πληροφορίες ροής που αφορούν σε ροές που βασίζονται σε υπαρκτές γραμμές (links) ή διαφορετικά δίνει πληροφορίες μόνο μεταξύ των άμεσα συνδεδεμένων συσκευών. Είτε πρόκειται για γραμμή μεταξύ switch-switch είτε μεταξύ switch-host. Για να ληφθούν λοιπόν οι πληροφορίες που απαιτεί ο αλγόριθμος για την εύρεση των διεπαφών που πρέπει να κλείσουν, θα έπρεπε να δημιουργηθεί ένας μηχανισμός συλλογής στατιστικών που δεν υποστηρίζει ο ελεγκτής από μόνος του, εφόσον η μόνη πληροφορία που μπορεί να δώσει είναι το πλήθος δεδομένων που εστάλησαν μεταξύ των άμεσα συνδεδεμένων μεταγωγέων.

Ο ODL -λόγω του L2Switch Main feature – κατά την εκκίνηση του δικτύου εγκαθιστά προκαθορισμένα κάποιες ροές για να είναι εφικτή η ευφυής επικοινωνία μεταξύ των μηχανημάτων του. Οι κανόνες αυτοί είναι με βάση τις MAC διευθύνσεις εκατέρωθεν της κίνησης, όμοιοι δηλαδή με τους κανόνες που θα χρειαστεί να προσθέσουμε εμείς.

Για να ωθήσουμε τον ελεγκτή να συλλέξει τις πληροφορίες από όλους τους μεταγωγείς, προς όλους τους υπόλοιπους, θα έπρεπε να προσθέσουμε τις κατάλληλες ροές. Για να επιλέξει μάλιστα τις δικές μας ροές αντί των προκαθορισμένων, κατά τη δημιουργία τους θα θέσουμε τη μέγιστη προτεραιότητα, δηλαδή 65535. Οπότε ο ελεγκτής για την επιλογή διαδρομής για την κίνηση, θα αντιστοιχίσει κάθε φορά την κίνηση με την αντίστοιχη δικιά μας ροή, κι οπότε κάθε κίνηση στο δίκτυο θα καταμετράται και θα συλλέγεται αργότερα.

Για να γίνει προσθήκη των κατάλληλων ροών, πρέπει να έχει προηγηθεί σωστή εξαγωγή και αντιστοίχιση της τοπολογίας που έχει ήδη ληφθή. Σε πρώτη φάση αποθηκεύονται τα στοιχεία κάθε switch (ID, όνομα) και των τερματικών (MAC

διεύθυνση). Έπειτα γίνεται αντιστοίχιση κάθε τερματικού στο switch που συνδέεται, αποθηκεύοντας τον αριθμό της θύρας που συνδέονται με κατάλληλη μέθοδο για να μπορέσει αργότερα να γίνει σωστή αντιστοίχιση. Επίσης με διαφορετικό τρόπο γίνεται η εξαγωγή και αποθήκευση στοιχείων για το ποιοι μεταγωγείς είναι άμεσα συνδεδεμένοι με ποιους μεταγωγείς, οπότε γίνεται καταμέτρηση πληροφοριών που θα χρειαστούν (ID των switches, όνομα διεπαφών, θύρες διεπαφών).

Στη συνέχεια περνάμε στη φάση προσθήκης των κατάλληλων ροών. Η προσθήκη νέων flows (ροών) μπορεί να συμβεί με δύο μεθόδους:

(1) Χρήση του OFCTL στο Mininet

(2) Χρήση του REST api του ODL

Χρησιμοποιώντας το REST api του ODL και συγκεκριμένα μέσω της PUT μεθόδου, στέλνουμε στον ελεγκτή τα στοιχεία των ροών που θέλουμε να προσθέσει στο config state μέσω της PUT μεθόδου στο :

http://<odl-IP>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:<switch-ID>/table/0/flow/<flow-ID>

Από το config state, αυτόματα ο ODL θα προσθέσει αυτά τα στοιχεία στο operational state από το οποίο μπορούμε να λάβουμε πληροφορίες μέσω της GET μεθόδου στο :

http://<odl-IP>:8181/restconf/operational/opendaylight-inventory:nodes/node/openflow:<switch-ID>/table/0/flow/<flow-ID>

Οι ροές δημιουργούνται μόνο από access switches (που έχουν τερματικά συνδεδεμένα πάνω τους) προς access switches. Ουσιαστικά δημιουργούμε ροές όπως η κίνηση που υπάρχει στο δίκτυο, δηλαδή με διεύθυνση πηγής και προορισμού τις MAC διευθύνσεις των τερματικών κι έτσι γίνεται καταμέτρηση των δεδομένων που αποστέλλονται μεταξύ τους.

Για τη δημιουργία της ροής χρησιμοποιείται κανόνας αντίστοιχος με τη σύνταξη της παρακάτω εντολής στο mininet:

ovs-ofctl add-flow <switchName> dl_src=<hostMacI> dl_dst=<hostMacJ>, in_port=* actions=output:<hostPort>

Ο κανόνας που ακολουθείται για τη δημιουργία ροών έχει ως διεύθυνση προορισμού τη διεύθυνση των τερματικών (ξεχωριστά το κάθε τερματικό του σε άλλη ροή) του εκάστοτε switch και ως διεύθυνση πηγής τη διεύθυνση όλων των υπολοίπων τερματικών που συνδέονται με τα εναπομείναντα access switches. Οπότε κατά τον υπολογισμό του bandwidth αργότερα, τα στοιχεία θα ληφθούν με αντίστροφο τρόπο από αυτόν με τον οποίο προστέθηκαν. Αυτό συμβαίνει διότι σε μια επικοινωνία μεταξύ δύο κόμβων δεν γνωρίζουμε εξ αρχής ποια διαδρομή θα επιλεχθεί. Δεν γνωρίζουμε από ποιο ενδιάμεσο link θα κατευθυνθούν τα πακέτα, παραμόνο από ποιον αποστολέα προήλθαν. Οπότε για τον κανόνα δημιουργίας ροών, ως γνωστή, θεωρείται η διεύθυνση προορισμού, η οποία αργότερα θα χρησιμοποιηθεί για να γίνει αντιστοίχιση της ροής και να εξαχθούν οι κατάλληλοι μετρητές δεδομένων.

Στον ODL χρησιμοποιήθηκε η εντολή curl, η μέθοδος PUT και η μορφή xml αντί για json τώρα, με τον εξής τρόπο:

```
curl --noproxy <odl-IP> -u <username>:<password> -H 'Content-Type: application/yang.data+xml' -X PUT -d
'


<flow xmlns="urn:opendaylight:flow:inventory">
        <priority>65535</priority>
        <flow-name> 1</flow-name>
        <match>
                <ethernet-match>
                        <ethernet-type>
                                <type>2048</type>
                        </ethernet-type>
                        <ethernet-destination>
                                <address>  “MACaddress”  </address>
                        </ethernet-destination>
                        <ethernet-source>
```

```
                                <address> “MACaddress”  </address>
                        </ethernet-source>
                </ethernet-match>
        </match>
        <id> ”Flow ID” </id>
        <table_id>0</table_id>
        <instructions>
                <instruction>
                        <order>0</order>
                        <apply-actions>
                                <action>
                                        <order>0</order>
                                        <output-action>
                                        <output-node-connector>“OUTPORT”</output-node-
connector>
                                                <max-length>60</max-length>
                                        </output-action>
                                </action>
                        </apply-actions>
                </instruction>
        </instructions>
</flow>



'
'http://<odl-IP>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:<switch-
ID>/table/0/flow/<Flow ID>'
```

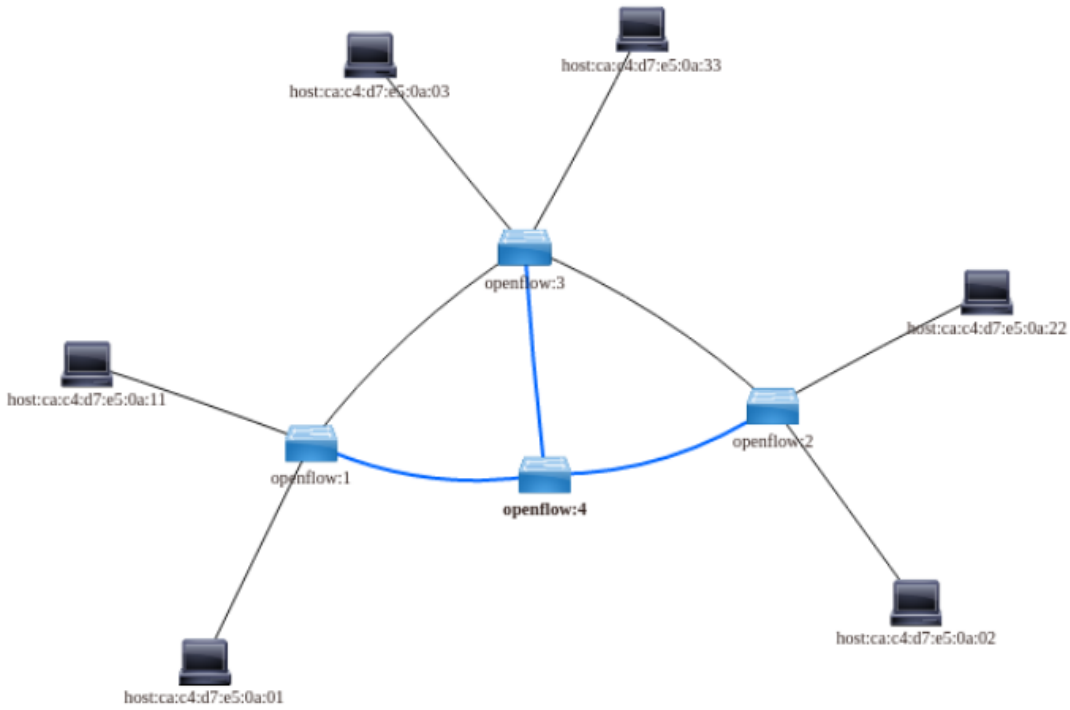
Για κάθε access switch, δημιουργούνται κανόνες με MAC διεύθυνση πηγής, όλες τις MAC διευθύνσεις όλων των άλλων τερματικών εκτός των δικών του. Η διεύθυνση προορισμού είναι κάθε φορά η διεύθυνση του εκάστοτε τερματικού του και η θύρα προορισμού για το συγκεκριμένο action είναι ο αριθμός της θύρας του στην οποία συνδέεται το συγκεκριμένο τερματικό. Το πλήθος των ροών που δημιουργούνται λοιπόν για N access switches στο καθέ από τα οποία συνδέονται M τερματικά είναι : N* M*(N-1)*M . Στην περίπτωση του δικτύου που αναφέρθηκε λοιπόν (16 access switches και 16 τερματικά το καθένα), δημιουργήθηκαν 61.440 ροές.

Μετά τη δημιουργία των ροών στο config state του ODL, αυτόματα οι πληροφορίες αποθηκεύονται στο operational state από το οποίο μπορούμε να λάβουμε τα στατιστικά, δηλαδή τον όγκο των δεδομένων που έχουν μεταφερθεί για τη συγκεκριμένη ροή μέχρι τη στιγμή που θα το ζητήσουμε.

Να επισημανθεί πως τα περιεχόμενα του config state δεν διαγράφονται από τη μνήμη του ελεγκτή ούτε μετά το κλείσιμο του δικτύου για το οποίο προστέθηκαν οι παραμετροποιήσεις ούτε μετά την επανεκκίνηση του ελεγκτή, γι αυτό και δημιουργήθηκαν κατάλληλα scripts που το υλοποιούν. Για τη διαγραφή των δεδομένων αυτών στο config state, γίνεται χρήση της μεθόδου DELETE μέσω του REST api και με τη βοήθεια της εντολής curl στο url:

http://<odl-IP>:8181/restconf/config/opendaylight-inventory:nodes/

Συγκεκριμένα, στο παρακάτω απλό δίκτυο ο μηχανισμός που αρχικά προσθέτει κι έπειτα συλλέγει τα στατιστικά θα λειτουργήσει με τον τρόπο που θα περιγραφτεί παρακάτω.



Εικόνα 12: Μικρή τοπολογία – Παράδειγμα

Οι ροές που προστίθενται είναι:

Flow	Source	Destination
1	Host2a	Host1a
2	Host2b	Host1a
3	Host3a	Host1a
4	Host3c	Host1a
5	Host2a	Host1b
6	Host2b	Host1b
7	Host3a	Host1b
8	Host3c	Host1b
9	Host1a	Host2a
10	Host1b	Host2a
11	Host3a	Host2a
12	Host3b	Host2a
13	Host1a	Host2b
14	Host1b	Host2b
15	Host3a	Host2b
16	Host3b	Host2b
17	Host1a	Host3a
18	Host1b	Host3a
19	Host2a	Host3a
20	Host2b	Host3a
21	Host1a	Host3b
22	Host1b	Host3b
23	Host2a	Host3b
24	Host2b	Host3b

Για τον υπολογισμό των δεδομένων μεταξύ των access switch:

Source Switch	Destination Switch	Flow’s Data
1	2	Flow9 + Flow10 + Flow13 + Flow14
1	3	Flow17 + Flow18 + Flow21 + Flow22
2	1	Flow1 + Flow2 + Flow5 + Flow6
2	3	Flow19 + Flow20 + Flow23 + Flow24
3	1	Flow3 + Flow4 + Flow7 + Flow8
3	2	Flow11 + Flow12 + Flow15 + Flow16

Σε καθεμία από τις άνω προσθέσεις λαμβάνονται υπόψιν το πλήθος των bytes που έχουν μεταφερθεί με το συγκεκριμένο flow. *Το παραπάνω αποτελεί απλό παράδειγμα και όχι την τοπολογία που χρησιμοποιήθηκε και υπάρχει η δυνατότητα να υποστηριχτεί.*

6.6 Υπολογισμός Βέλτιστης Ενεργειακά Δρομολόγησης

Αφού συλλεχθεί το πλήθος των δεδομένων σε κάθε ροή, ακολουθεί ο υπολογισμός του bandwidth. Για να υλοποιηθεί αυτό, λαμβάνονται αποτελέσματα και γίνονται οι αντίστοιχοι υπολογισμοί ανά κάποια χρονικά διαστήματα, αυτόματα και για όσο λειτουργεί το δίκτυο. Μετά από αυτό το στάδιο είναι έτοιμα τα στοιχεία που είναι απαραίτητα για την εκτέλεση του αλγορίθμου στο cplex. Λαμβάνεται υπόψιν η τοπολογία του δικτύου (το πλήθος των switch και η συνδεσμολογία τους) , η χωρητικότητα των γραμμών και το bandwidth μεταξύ όλων των switch προς όλων των υπολοίπων.

Η εκτέλεση αυτή μπορεί να αργήσει από δευτερόλεπτα εώς και αρκετά λεπτά, ανάλογα το μέγεθος και την πολυπλοκότητα του εκάστοτε δικτύου, καθώς και ανάλογα της κίνησης που είχε παραχθεί σ'αυτό. Μετά την εκτέλεσή του, έχουμε σε μορφή 0/1 ποια index των γραμμών θα κλείσουν και ποια θα μείνουν ανοιχτά. Οπότε σε τελικό στάδιο πλέον, πρέπει η εφαρμογή να αντιστοιχίσει τα αποτελέσματα αυτά με τις διεπαφές που αναλογούν.

Υπάρχουν τρεις μέθοδοι για την αλλαγή της κατάστασης κάθε πόρτας.

(1) Χρήση του ovsdb για την αλλαγή της πόρτας από την βάση και την παραμετροποίηση του Open vSwitch

(2) Χρήση του REST-API του ODL

(3) Χρήση του OFCTL για τη διαχείριση του Open vSwitch

Μιας και η τοπολογία εκτελείται στον ίδιο φυσικό εξυπηρετητή επιλέχθηκε η χρήση του (3). Η χρήση του ovsdb από την εφαρμογή δεν είναι εφικτή καθώς οι μεταγωγείς OpenvSwitch εντός του Mininet δεν είναι προσπελάσιμοι από το δίκτυο και το API του ODL δεν υποστηρίζει σε αυτή την έκδοση την επιθυμητή λειτουργία.

Η εντολή χρησιμοποιήθηκε ως εξής:

ovs-ofctl mod-port <switchName> <interfaceName> down/up

6.7 Εξαγωγή Αποτελεσμάτων

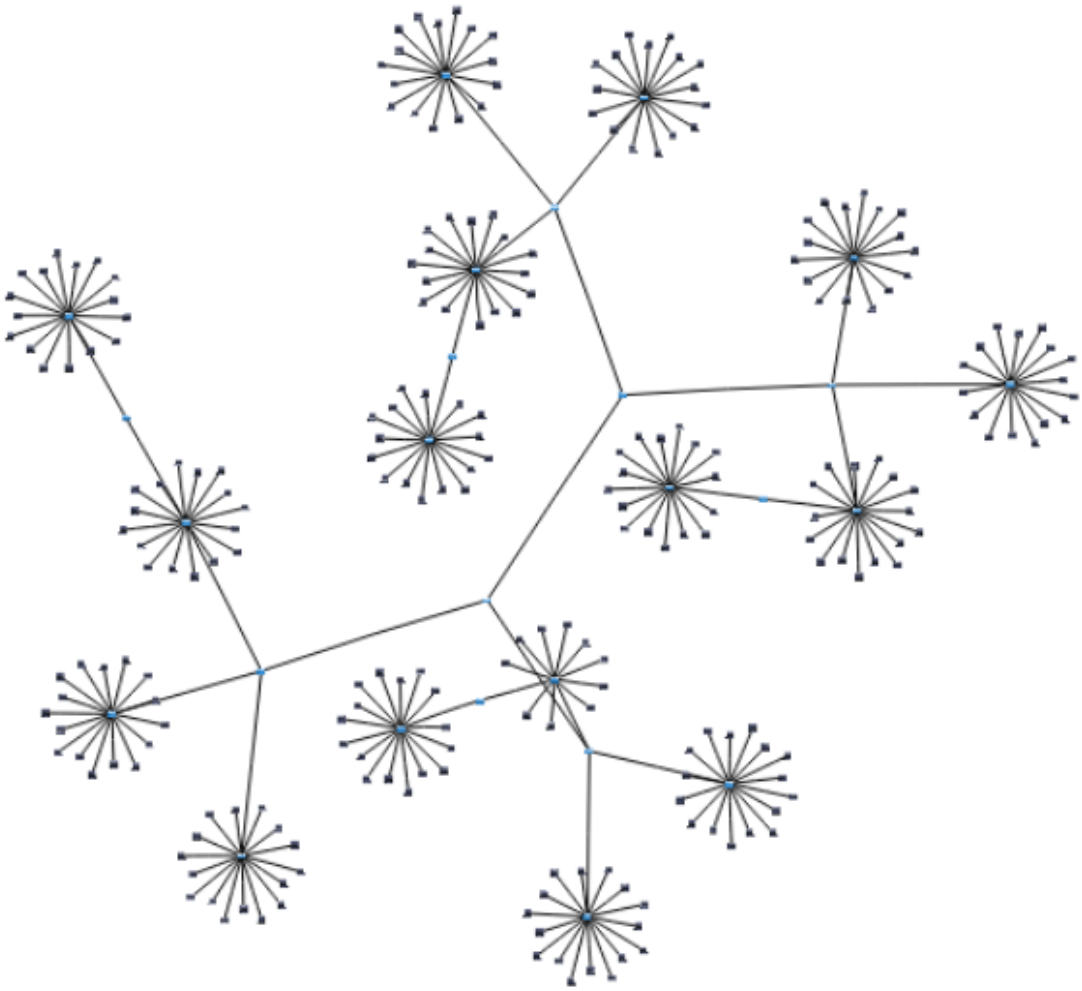
Μετά από την εκτέλεση της εφαρμογής το αρχικό δίκτυο,φέρεται να έχει τη μορφή:

Εικόνα 13: Τοπολογία δικτύου εφαρμογής μετά την βελτιστοποίηση (OpenDaylight DLUX)

Προκύπτει ένα Συνδετικό Δέντρο (Spanning Tree) που περιέχει όλα τα Access switches της αρχικής τοπολογίας καθώς και όλα τα τερματικά που παραμένουν όλα προσπελάσιμα και συνδεδεμένα. Για κάποια Distribution switches και για το ένα εκ των δύο Core, έχουν απενεργοποιηθεί όλες οι διεπαφές οπότε έχουν κλείσει οι συσκευές. Στην παραπάνω τοπολογία παρέμειναν ενεργές 20 από τις 48 γραμμές. Συγκεκριμένα από τις

16 γραμμές των 10Gb (σύνδεση μεταξύ core-distribution switch) παρέμειναν ενεργές οι 4 ενώ από τις 32 γραμμές των 1Gb (σύνδεση μεταξύ distribution-access switch) παρέμειναν ανοιχτές οι 16.

Οι επιλογές που γίνονται για το ποιες διεπαφές θα αλλάξουν κατάσταση λειτουργίας, προσαρμόζονται ανάλογα την κίνηση που που φέρεται να είχε πιο πρόσφατα το δίκτυο, προκειμένου να εκμεταλλευτούμε στο έπακρο τα μηχανήματα και τη χωρητικότητα των γραμμών, καθώς και τα ενεργειακά μοντέλα των μεταγωγέων. Στο παραπάνω παράδειγμα, τα Access switches θεωρήθηκε ότι καταναλώνουν τη μισή ενέργεια από τα Distribution και Core switches. Για μεταγωγείς με ίδια ενεργειακά μοντέλα, δηλαδή ίδια κατανάλωση σε Watt, προέκυψε το παρακάτω παράδειγμα:



Εικόνα 14: Τοπολογία δικτύου εφαρμογής μετά την βελτιστοποίηση (OpenDaylight DLUX) για ίδια ενεργειακά μοντέλα σε όλους τους μεταγωγείς

Στο ανώτερο παράδειγμα φαίνεται πως κάποια Access switches προωθούν πλέον κίνηση, διότι αυτό συμφέρει ενεργειακά εφόσον δεν διαχωρίστηκαν τα είδη των μεταγωγέων ως προς την κατανάλωσή τους.

Ο OpenDaylight αυτόματα δημιουργεί το Minimum Spanning Tree της εκάστοτε τοπολογίας μέσω της μεθόδου loopremover του L2 Switch feature. Χρησιμοποιεί ένα δικό του Spanning Tree Protocol (STP) για την επιλογή του , επομένως διαφορετικό κριτήριο από το δικό μας για την επιλογή του δέντρου που θα καθορίσει τη δρομολόγηση στο ευφυές δίκτυο, με βάση το πλήθος των hops (ενδιάμεσες συσκευές μεταξύ πηγής και προορισμού), χωρίς να λάβει υπόψιν του τα χαρακτηριστικά του κάθε μεταγωγέα ως προς την κατανάλωσή του.

Μέσω της εφαρμογής, καταλήγουμε , εν γένει, σε διαφορετικό δέντρο από αυτό που μόνος του επιλέγει ο ελεγκτής και αλλάζουμε την πολιτική προώθησης με γνώμονα την εξοικονόμηση ενέργειας από την ηλεκτροδότηση των μηχανημάτων και γραμμών και όχι με γνώμονα το πλήθος των hops που ενδεχομένως να οδηγεί σε ίδιο συνδετικό δέντρο κάθε φορά για την ίδια τοπολογία.

Ο μηχανισμός που αναπτύχθηκε για τη συλλογή στατιστικών για το πλήρες δίκτυο και όχι μόνο μεταξύ των άμεσα συνδεδεμένων μηχανημάτων -όπως υποστηρίζει ο ODL και κάθε ελεγκτής- , έχει γενικευμένη λειτουργία και δύναται να χρησιμοποιηθεί σε οποιοδήποτε δίκτυο χωρίς να απαιτεί τη γνώση κανενός στοιχείου γι αυτό, πριν την εκτέλεση της εφαρμογής. Μπορεί να χρησιμοποιηθεί από επόμενες εκδόσεις του ODL καθώς και από άλλους ελεγκτές που υποστηρίζουν REST api όμοια με τον ODL, με ελάχιστες ή καθόλου αλλαγές, ανάλογα τη μορφή των json/xml αρχείων που χρησιμοποιούνται, εφόσον δεν επηρεάζεται από τη γλώσσα προγραμματισμού που είναι βασισμένος ο ελεγκτής. Παράλληλα υπάρχει αυτονομία στο σύστημα λόγω του ότι ο ελεγκτής εποπτεύεται και παραμετροποιείται εξωτερικά χωρίς να γίνονται αλλαγές στον πυρήνα του, κάτι που μπορεί να επηρέαζε τη λειτουργία του δικτύου. Επίσης μπορεί να χρησιμοποιηθεί σε διαφορετική εφαρμογή, για εξαγωγή διαφορετικών συμπερασμάτων από αυτά της παρούσης εργασίας.

Βιβλιογραφία

[1] ONF White Paper, “Software-Defined Networking: The New Norm for Networks” ,April 2012

[2] “OpenFlow” , http://flowgrammable.org/sdn/openflow/

[3] Open Networking Foundation (ONF), “SDN Architecture Overview” ,Version 1.0, December 2013

[4] “Hybrid SDN is the gateway drug to the new network ” http://searchsdn.techtarget.com/feature/Hybrid-SDN-is-the-gateway-drug-to-the-new-network

[5] “OpenDaylight” https://www.opendaylight.org/faq

[6] “OpenDaylight Project”, https://en.wikipedia.org/wiki/OpenDaylight_Project

[7] “What’s in OpenDaylight?”, https://www.mirantis.com/blog/whats-opendaylight/

[8] “ODL Beryllium”, https://www.opendaylight.org/odlbe

[9] “REST (representational state transfer)”, http://searchsoa.techtarget.com/definition/REST

[10] “What Is RESTCONF”, http://sdntutorials.com/what-is-restconf/

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

[12] “Introduction to Mininet“, https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[13] Josh Whitney (Anthesis), Pierre Delforge (NRDC) , “Data Center Efficiency Assessment” , August 2014

[14] Paris Charalampou, Anastasios Zafeiropoulos, Constantinos Vassilakis, Chrysostomos Tziouvaras, Vasiliki Giannikopoulou, Nikolaos Laoutaris, “Empirical evaluation of energy saving margins in backbone”

[15] Yunfei Shang, Dan Li, Mingwei Xu , “Energy-aware Routing in Data Center Network”

[16] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma , Sujata Banerjee , Nick McKeown “ElasticTree: Saving Energy in Data Center Networks Brandon”

[17] “How much energy does the Internet use?”, http://computer.howstuffworks.com/internet/basics/how-much-energy-does-internet-use2.htm

[18] “CPLEX Optimizer”, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[19] “Data Center Reference Architectures”, https://www.ietf.org/proceedings/82/slides/armd-1.pdf

[20] “Editing OpenDaylight OpenFlow Plugin:End to End Flows:Example Flows ”, https://wiki.opendaylight.org/view/Editing_OpenDaylight_OpenFlow_Plugin:End_to_End_Flows:Example_Flows

Παράρτημα

```
#!/usr/bin/python

import os
import sys
import subprocess
import json
import string
import shutil
import time


#αφαίρεση των link μεταξύ host-switch
def clear(list1,  mystring):
        end = len(list1)
        i = 0
        while (i<end):
                if (mystring in list1[i]):
                        if i!=len(list1)-1:
                                list1 = list1[:i] + list1[i+1 :]
                        else:
                                list1 = list1[:i]

                        end-=1
                        i-=1
                i+=1
        return [list1]


#Επιλογή links μόνο μεταξύ switch-host
def onlyHost(listSrc, listDest):
        end = len(listSrc)
        i = 0
        myOpenflow = "openflow"
        myHost = "host"
        while (i<end):
                if ((myOpenflow not in listSrc[i]) or (myHost not in listDest[i]) ):
                        if i!=len(listSrc)-1:
                                listSrc = listSrc[:i] + listSrc[i+1 :]
                                listDest = listDest[:i] + listDest[i+1 :]
                        else:
                                listSrc = listSrc[:i]
                                listDest = listDest[:i]
                        end-=1
                        i-=1
                i+=1
        return [listSrc, listDest]


#Επιλογή links μόνο μεταξύ switch-switch
```

```
def onlySwitch(listSrc, listDest):
        end = len(listSrc)
        i = 0
        myHost = "host"
        while (i<end):
                if ((myHost in listSrc[i]) or (myHost in listDest[i]) ):
                        if i!=len(listSrc)-1:
                                listSrc = listSrc[:i] + listSrc[i+1 :]
                                listDest = listDest[:i] + listDest[i+1 :]
                        else:
                                listSrc = listSrc[:i]
                                listDest = listDest[:i]
                        end-=1
                        i-=1
                i+=1
        return [listSrc, listDest]


#αφαίρεση διπλότυπων
def unique(mylist):
        end = len(mylist)
        i = 0
        while (i<end-1):
                if (mylist[i] == mylist[i+1]):
                        if i!=len(mylist)-1:
                                mylist = mylist[:i] + mylist[i+1 :]
                        else:
                                mylist = mylist[:i]
                        end-=1
                        i-=1
                i+=1
        return mylist


#openflow:1000:10 --> 10
def portNumber(mystring):
        s = mystring
        start = s.find("openflow:") + 9
        end = s.find(":",start)
        s1 =  s[start:end]
        s = "openflow:"+s1+":"
        start = mystring.find(s) +len(s)
        end = mystring.find("\"",start)
        s2 = ( mystring[start:end])
        s1 = int(s1)
        s2 = int(s2)
        return s2


#host:ca:c4:d7:e5:0a:33 --> ca:c4:d7:e5:0a:33
def macAddress(mystring):
        s = mystring
        start = s.find("host:") + 5
```

```
        end = s.find("\"",start)
        s1 = s[start:end]
        return str(s1)

# URL encoding
def urlEncode(mylist):
        i = 0
        while (i<len(mylist)):
                s = mylist[i]
                s = s.replace('#', '%23')
                s = s.replace('$', '%24')
                mylist[i] = s
                i+=1
        return mylist


#Εύρεση της MAC address μέσα από ένα json string
def findSrcMac(myMatch ):
        index = -1
        new = 0
        mac = []
        sMatch = "\"Match"
        sSource = "_ethernetSource=EthernetSource"
        sValue = "[_value="
        for i in range(0,len(myMatch)):
                mbool = sMatch in myMatch[i]
                if (mbool):
                        index +=1
                        new = 1
                        mac.append([])
                if (new and (i < len(myMatch)-2)):
                        sbool = sSource in myMatch[i]
                        vbool = sValue in myMatch[i+2]
                        if (sbool and vbool):
                                s = myMatch[i+2]
                                start = s.find(sValue) + len(sValue)
                                end = s.find("]",start)
                                mac[index] =  s[start:end]
                                new = 0
        return mac


def binarySearch(mylist,mystring,ifInt):
        if (not ifInt):
                mystring = mystring.lower()
                found = 0
                down = 0
                up = len(mylist)
                while (not found):
                        med = (up + down)/2
                        if (mystring in mylist[med]):
                                found = 1
                        elif (mystring < mylist[med]):
```

```
                                        up = med - 1
                                else:
                                        down = med + 1
        else:
                found = 0
                down = 0
                up = len(mylist)
                while (not found):
                        med = (up + down)/2
                        if (mystring == mylist[med]):
                                found = 1
                        elif (mystring < mylist[med]):
                                up = med - 1
                        else:
                                down = med + 1
        return med


#list από : "openflow:1000" -> list από : 1000
def takeID(mylist):
        length = len(mylist)
        i = 0
        while (i<length):
                s = mylist[i]
                start = s.find("openflow:") +9
                end = s.find("\"",start)
                s1 =  s[start:end]
                mylist[i] = int(s1)
                i+=1
        return mylist


def takeID2(mylist):
        length = len(mylist)
        i = 0
        while (i<length):
                s = mylist[i]
                start = s.find("openflow:") +9
                end = s.find(":",start)
                s1 =  s[start:end]
                mylist[i] = int(s1)
                i+=1
        return mylist


# list από: "openflow:1000:10" -> [1000,10]
def takeIDwithPort(mylist):
        length = len(mylist)
        i = 0
        mylist1 = []
        mylist2 = []
        while (i<length):
                mylist1.append([])
```

```
                mylist2.append([])
                mystring = mylist[i]
                s = mylist[i]
                start = s.find("openflow:") +9
                end = s.find(":",start)
                s1 =  s[start:end]
                s = "openflow:"+s1+":"
                start = mystring.find(s) +len(s)
                end = mystring.find("\"",start)
                s2 = ( mystring[start:end])
                mylist1[i] = int(s1)
                mylist2[i] = int(s2)
                i+=1
        return [mylist1, mylist2]


def selectBand(mystring):
        if ("ten-mb" in mystring):
                bandwidth = 10
        elif ("hundred-mb" in mystring):
                bandwidth = 100
        elif ("one-gb" in mystring):
                bandwidth = 1000
        elif ("ten-gb" in mystring):
                bandwidth = 10000
        elif ("forty-gb" in mystring):
                bandwidth = 40000
        elif ("hundred-gb" in mystring):
                bandwidth = 100000
        elif ("one-tb" in mystring):
                bandwidth = 1000000
        else:
                bandwidth = 1000
        return bandwidth


#Απόφαση για το είδος του link
#Ανάλογα το αν τα δύο switch είναι Core,Distibution,Access
def selectLink(ii, jj , listSwitchID, listMappingNumber, listConnectedSw):
        if (listMappingNumber[ii]>0 or listMappingNumber[jj]>0):
                #Link Access-Distribution
                whichLink = 1
        else:
                boolNeighI = 0
                boolNeighJ = 0
                for neighb in range(0,len(listSwitchID)):
                        if (neighb!=ii):
                                if (listConnectedSw[ii][neighb]==1):
                                        if (listMappingNumber[neighb]>0):
                                                boolNeighI = 1  #o i einai distribution
                                                break
                for neighb in range(0,len(listSwitchID)):
                        if (neighb!=jj):
```

```
                                    if (listConnectedSw[jj][neighb]==1):
                                            if (listMappingNumber[neighb]>0):
                                                    boolNeighJ = 1  #o j einai distribution
                                                    break
                        if (boolNeighI and boolNeighJ):
                                #Link Distribution-Distribution
                                whichLink = 2
                        elif (boolNeighI or boolNeighJ):
                                #Link Distribution-Core
                                whichLink = 3
                        else:
                                #Link Core-Core
                                whichLink  = 4
                return whichLink


#1 εάν πρόκειται για access switch
def ifAccess(ii, listMappingNumber):
        if (listMappingNumber[ii]>0):
                access = 1
        else:
                access = 0
        return access


def noquotes(mystring):
        tempStr = mystring.replace('\"', '')
        return tempStr


inventory = "curl -u <USERNAME>:<PASSWORD> -X GET -s http://<ODL-
IP>:8181/restconf/operational/opendaylight-inventory:nodes/"
topology = "curl -u <USERNAME>:<PASSWORD>  -X GET -s http://<ODL-
IP>:8181/restconf/operational/network-topology:network-topology/"
print time.strftime("%H:%M:%S") +"\n"
link =  topology + " | jq '.[\"network-topology\"].topology[0].link[] | select(length >0 )  ' >
text.json "
subprocess.call(link , shell = True)

cat = "cat text.json | jq '.[\"link-id\"] ' "
linkNode = subprocess.check_output(cat , shell = True)
linkID = linkNode.split()

cat = "cat text.json |  jq '.source | .[\"source-node\"] ' "
sourceNode = subprocess.check_output(cat , shell = True)
sourceID = sourceNode.split()

cat = "cat text.json |  jq '.source | .[\"source-tp\"] ' "
linkSourceNode = subprocess.check_output(cat , shell = True)
linkSourceID = linkSourceNode.split()

cat = "cat text.json |  jq '.destination | .[\"dest-tp\"] ' "
linkDestNode = subprocess.check_output(cat , shell = True)
```

```
linkDestID = linkDestNode.split()

hostMac = []
hostPort = []
hostSwitch = []

#Πληροφορίες για host: mac,port του switch στο οποίο συνδέεται, switch στο οποίο συνδέεται
for i in range(0, len(sourceID)):
        if ("host" in sourceID[i]):
                hostMac.append([])
                hostPort.append([])
                hostSwitch.append([])

#επιλογή όλων των switch
[sourceID] = clear(sourceID, "host")
sourceID = takeID(sourceID)
sourceID.sort()

switchID = unique(sourceID)

linkSourceID_sw = []
linkDestID_sw = []
portSource = []

for i in range (0,len(linkSourceID)):
        linkSourceID_sw.append([])
        linkSourceID_sw[i] = linkSourceID[i]
        linkDestID_sw.append([])
        linkDestID_sw[i] = linkDestID[i]
        portSource.append([])

#links μόνο μεταξύ switch-host
[linkSourceID , linkDestID ] = onlyHost(linkSourceID , linkDestID )

#links μόνο μεταξύ switch-switch
[linkSourceID_sw , linkDestID_sw ] = onlySwitch(linkSourceID_sw , linkDestID_sw )

#συλλογή των ports (εκάστοτε interface)
[linkSourceID_sw , portSource] = takeIDwithPort(linkSourceID_sw  )
linkDestID_sw = takeID2(linkDestID_sw )

#αντικατάσταση του ID του switch με το αντίστοιχο index
#με τη βοήθεια binarySearch (n^2logn)
for i in range(0,len(linkSourceID_sw)):
        linkSourceID_sw[i] = binarySearch(switchID, linkSourceID_sw[i] , 1)
        linkDestID_sw[i] = binarySearch(switchID, linkDestID_sw[i] , 1)


#n*n πίνακας με 1 αν τα switches συνδέονται
connectedSw = [[]]
connectedSw = [[0 for i in xrange(len(switchID))] for j in xrange(len(switchID))]

#n*n πίνακας με τον αριθμό της κάθε πόρτας για το εκάστοτε link
```

```
#(0 αν δεν υπάρχει link μεταξύ των switches)
portSw = [[]]
portSw = [[0 for i in xrange(len(switchID))] for j in xrange(len(switchID))]

#n*n πίνακας με το όνομα του interface
interfaceName = [[]]
interfaceName = [[0 for i in xrange(len(switchID))] for j in xrange(len(switchID))]

#συλλογή των ονομάτων των interface
for i in range(0,len(linkSourceID_sw)):
        connectedSw[linkSourceID_sw[i]][linkDestID_sw[i]] = 1
        portSw[linkSourceID_sw[i]][linkDestID_sw[i]] = portSource[i]
        link = inventory +"node/openflow:"+str(switchID[linkSourceID_sw[i]]) +"/node-
connector/openflow:"+str(switchID[linkSourceID_sw[i]]) +":"+str(portSource[i]) +"/|jq '.[] | .[] |
{\"flow-node-inventory:name\" } | .[\"flow-node-inventory:name\"]' "
        interfaceName[linkSourceID_sw[i]][linkDestID_sw[i]] = subprocess.check_output(link ,
shell = True)

switchOne = []
switchTwo = []

for i in range (0, len(linkSourceID_sw)/2+1):
        switchOne.append([])
        switchTwo.append([])


mappingFirst = []
mappingNumber = []
switchName = []

for i in range(0, len(switchID)):
        switchName.append([])                   #όνομα switch
        mappingNumber.append([])                #πλήθος host που συνδέονται πάνω στο
συγκεκριμένο switch
        mappingNumber[i] = 0
        mappingFirst.append([])                         #indexing για εύρεση των host του κάθε
switch
        mappingFirst[i] = 0


swn = open("switchName.txt","w", 0)

#Εύρεση Τοπολογίας
k = 0
for i in range(0,len(switchID)):
        link = inventory +"node/openflow:"+str(switchID[i]) +"/node-
connector/openflow:"+str(switchID[i]) +":LOCAL/|jq '.[] | .[] | {\"flow-node-inventory:name\" }
| .[\"flow-node-inventory:name\"]' "
        switchName[i] = subprocess.check_output(link , shell = True)
        swn.write("%s " %(switchName[i].replace("\n","")))
        first = 0
        for j in range (0,len(linkSourceID)):
                if ("openflow:"+str(switchID[i])+":" in linkSourceID[j]):
```

```
                        if (first == 0):
                                mappingFirst[i] = k
                                first = 1
                        hostPort[k] = portNumber(linkSourceID[j])
                        hostMac[k] = macAddress(linkDestID[j])
                        hostSwitch[k] = i
                        mappingNumber[i] += 1
                        k += 1

swn.close()

#Από τη MAC address θα εξαχθεί το switch στο οποίο συνδέεται το αντίστοιχο τερματικό

hashMac = []
hashSwitch = []

#Δημιουργία διπλότυπων για δημιουργία διαφορετικής μεθόδου αντιστοίχισης
for i in range(0, len(hostMac)):
        hashMac.append([])
        hashMac[i] = hostMac[i]
        hashSwitch.append([])
        hashSwitch[i] = hostSwitch[i]   #το index του switch


zipped = zip(hashMac,hashSwitch)
zipped.sort()
hashMac,hashSwitch = zip(*zipped)


flag = 0

flowBytes = [[]]
flowBytes = [[0 for i in xrange(len(switchID))] for j in xrange(len(switchID))]

flowBps = [[]]
flowBps = [[0.00005 for i in xrange(len(switchID))] for j in xrange(len(switchID))]

inBytesTemp = []
boolTemp = []
for j in range(0,len(switchID)):
        inBytesTemp.append([])
        boolTemp.append([])


topo = open('topology.txt','w', 0)
whichSwitches = open('whichSwitches.txt', 'w' , 0)
con = open('connectedSw.txt','w', 0)
por = open('connectedPort.txt','w', 0)
interf = open ("interface.txt", "w",0)

topo.write("  V = {\n")
for i in range(0,len(switchID)):
        ii = i+1
```

```
        topo.write( "   <  %d,   0 >, \n" %ii)
topo.write( " };\n\n\n")
topo.write( "  E = {\n")
k = 1
for i in range(0,len(switchID)):
        for j in range(i+1,len(switchID)):
                if (connectedSw[i][j]==1):
                        ii = i + 1
                        jj = j + 1
                        switchOne[k] = i
                        switchTwo[k] = j
                        whichLink = selectLink(i, j , switchID, mappingNumber, connectedSw)
                        if (whichLink == 1):     #Link Access-Distribution
                                bandwidth = 1000
                        elif (whichLink == 2): #Link Distribution-Distribution
                                bandwidth = 1000
                        elif (whichLink == 3): #Link Distribution-Core
                                bandwidth = 10000
                        elif (whichLink == 4):  #Link Core-Core
                                bandwidth = 10000
                        else:
                                bandwidth = 10000
                #       link = inventory +"node/openflow:"+str(switchID[linkSourceID_sw[i]])
+"/node-connector/openflow:"+str(switchID[linkSourceID_sw[i]]) +":"+str(portSource[i])+"/|jq
'.[] | .[] | {\"flow-node-inventory:current-feature\" } | .[\"flow-node-inventory:current-feature\"]' "
                #       bandwidth = subprocess.check_output(link , shell = True)
                #       bandwidth = selectBand(bandwidth)
                        topo.write( "   <  %d,   0 >,   " %ii +"<  %d ,   0 >,  " %jj  + "18.75,
22.5,   26.25,   30,  %d" %bandwidth + ",  %d  >,\n" %k )
                        topo.write( "   <  %d,   0 >,   " %jj +"<  %d ,   0 >,  " %ii  + "18.75,
22.5,   26.25,   30,  %d" %bandwidth + ",  %d  >,\n" %k )
                        k += 1
topo.write( " };\n\n\n")

whichSwitches.write("%d \n" %(len(switchOne)-1))
for i in range(1,len(switchOne)):
        whichSwitches.write("%d " %switchOne[i])
whichSwitches.write("\n")
for i in range(1,len(switchTwo)):
        whichSwitches.write("%d " %switchTwo[i])
whichSwitches.close()


con.write("%d \n" %len(switchID))

for i in range(0,len(switchID)):
        for j in range(0,len(switchID)):
                con.write("%d " %connectedSw[i][j])
                por.write("%d " %portSw[i][j])
                if (interfaceName[i][j]):
                        interf.write("%s " %(interfaceName[i][j].replace("\n","")))
                else:
                        interf.write("%s " %interfaceName[i][j])
```

```
        con.write("\n")
        por.write("\n")
        interf.write("\n")

con.close()
por.close()
interf.close()

seconds=3600

while True:
        if (flag):
                print time.strftime("%H:%M:%S") +"      Begin  "

        temp = open('temp.txt','w', 0)
        if (flag):
                #Δημιουργία πίνακα με το πλήθος των switch
                temp.write("  V = {\n")
                for i in range(0,len(switchID)):
                        ii = i+1
                        temp.write( "   <  %d,   0 >, \n" %ii)
                temp.write( " };\n\n\n")
                temp.write( "  E = {\n")
                k = 1
                #Δημιουργία πίνακα με τα στοιχεία της τοπολογίας/συνδεσμολογίας
                #Ποια switch συνδέονται και τι ενεργειακά στοιχεία έχουν
                for i in range(0,len(switchID)):
                        for j in range(i+1,len(switchID)):
                                if (connectedSw[i][j]==1):
                                        ii = i + 1
                                        jj = j + 1
                                        whichLink = selectLink(i, j , switchID,
mappingNumber, connectedSw)
                                        if (whichLink == 1):     #Link Access-Distribution
                                                bandwidth = 1000
                                                energy = "18.75,   22.5,   26.25,   30,"
                                        elif (whichLink == 2): #Link Distribution-Distribution
                                                bandwidth = 1000
                                                energy = "18.75,   22.5,   26.25,   30,"
                                        elif (whichLink == 3): #Link Distribution-Core
                                                bandwidth = 10000
                                                energy = "37.50,   45.0,   52.50,   60,"
                                        elif (whichLink == 4):  #Link Core-Core
                                                bandwidth = 10000
                                                energy = "37.50,   45.0,   52.50,   60,"
                                        else:
                                                bandwidth = 10000
                                                energy = "37.50,   45.0,   52.50,   60,"
                                        temp.write( "   <  <  %d,   0 >,   " %ii +"<  %d ,   0 >,  "
%jj  + energy +"  %d" %bandwidth + ",  %d  >,\n" %k )
                                        temp.write( "   <  <  %d,   0 >,   " %jj +"<  %d ,   0 >,  "
%ii  + energy +"  %d" %bandwidth + ",  %d  >,\n" %k )
                                        k += 1
```

```
                temp.write( " };\n\n\n")

        temp.flush()
        os.fsync(temp)

        for i in range(0,len(switchID)):
                #Στην περίπτωση που το switch είναι Access, θα υπολογίσουμε τα flows του
                if (ifAccess(i,mappingNumber)):
                        link = inventory +"node/openflow:"+str(switchID[i]) +"/table/0/|jq '. ' > flow.json"
                        subprocess.call(link , shell = True)

                        cat = "cat flow.json | jq '.  [\"flow-node-inventory:table\"][0][\"flow-hash-id-map\"][][\"flow-id\"]' "
                        flowID = subprocess.check_output(cat , shell = True)
                        flowID = flowID.split()
                        flowID = urlEncode(flowID)

                        cat = "cat flow.json | jq '.  [\"flow-node-inventory:table\"][0][\"flow-hash-id-map\"][][\"hash\"]'  "
                        match = subprocess.check_output(cat , shell = True)
                        match = match.split()

                        inFlowMac = findSrcMac(match)                   #Πίνακας με τη MAC source του κάθε flow


                        if (flag):
                        #αποθήκευση των μετρητών πριν τη νέα μέτρηση
                        #αποθήκευση στήλης κι όχι γραμμής λόγω του κανόνα δημιουργίας των ροών ως εισερχόμενες
                                for j in range(0,len(switchID)):
                                        inBytesTemp[j] = flowBytes[j][i]
                                        boolTemp[j] = 0

                        for k in range (0,len(flowID)):

                                if (inFlowMac[k]):

                                        #Εύρεση source switch. Εύρεση του index μετά του ID
                                        outSwitchID = hashSwitch[binarySearch(hashMac,inFlowMac[k], 0)]

                                        link = inventory +"node/openflow:"+str(switchID[i]) +"/table/0/flow/"+noquotes(flowID[k]) +"| jq '.[\"flow-node-inventory:flow\"][0] | {\"byte-count\": .[\"opendaylight-flow-statistics:flow-statistics\"][\"byte-count\"] } | .[\"byte-count\"] ' "
                                        byteCounter = subprocess.check_output(link , shell = True)

                                        #Πρόσθεση μετρητών από την κίνηση προς όλα τα τερματικά του εκάστοτε switch
                                        #από όλα τα τερματικά του αντίστοιχου εξεταζόμενου
                                        if (flag==0):
```

```
                                        if (byteCounter  == "null\n"):
                                                flowBytes[outSwitchID][i] = 0

                                        else:
                                                flowBytes[outSwitchID][i] +=
int(byteCounter)
                                else:
                                        if (byteCounter =="null\n"):
                                                flowBytes[outSwitchID][i] = 0

                                        else:
                                                if (boolTemp[outSwitchID] == 0):
                                                        flowBytes[outSwitchID][i]  =
int(byteCounter)
                                                        boolTemp[outSwitchID] =1
                                                else:
                                                        flowBytes[outSwitchID][i]  +=
int(byteCounter)


                if (flag):
                        for j in range(0,len(switchID)):
                                if (i!=j):
                                        if (ifAccess(i,mappingNumber) and
ifAccess(j,mappingNumber)):   #kinisi mono meta3y Access switches
                                                flowBps[j][i] = float ((flowBytes[j][i]-
inBytesTemp[j])*8/ float (seconds*1000000))
                                                if (flowBps[j][i]<=0 ):
                                                        flowBps[j][i] = 0.00005
                                        else:
                                                flowBps[j][i] = 0.00000

        if (flag):
                #Πίνακας με τα bandwidth όλων των ροών
                temp.write( "  F = { \n")
                for i in range(0,len(switchID)):
                        for j in range(0,len(switchID)):
                                if (i!=j):
                                        ii = i + 1
                                        jj = j + 1
                                        temp.write("   < < %d,   0 >,   " %ii +"< %d ,   0 >,  "
%jj  + "%.6f >,\n"  %flowBps[i][j])
                temp.write("};")
                temp.write("\n")
                temp.write("\n")
                print time.strftime("%H:%M:%S") +"      Ready\n"

        temp.flush()
        os.fsync(temp)

        f = open('data.txt','w',0)
        subprocess.call("cp temp.txt data.txt" , shell = True)
```

```
        f.flush()
        os.fsync(f)

        flag = 1

        time.sleep(seconds)

f.close()
```

```
#!/usr/bin/python

import subprocess
import json
import string
import shutil
import time


#αφαίρεση των link μεταξύ host-switch
def clear(list1,  mystring):
        end = len(list1)
        i = 0
        while (i<end):
                if (mystring in list1[i]):
                        if i!=len(list1)-1:
                                list1 = list1[:i] + list1[i+1 :]
                        else:
                                list1 = list1[:i]

                        end-=1
                        i-=1
                i+=1
        return [list1]


#Επιλογή links μόνο μεταξύ switch
def onlyHost(listSrc, listDest):
        end = len(listSrc)
        i = 0
        myOpenflow = "openflow"
        myHost = "host"
        while (i<end):
                if ((myOpenflow not in listSrc[i]) or (myHost not in listDest[i]) ):
                        if i!=len(listSrc)-1:
                                listSrc = listSrc[:i] + listSrc[i+1 :]
                                listDest = listDest[:i] + listDest[i+1 :]
                        else:
                                listSrc = listSrc[:i]
                                listDest = listDest[:i]
                        end-=1
                        i-=1
                i+=1
```

```
        return [listSrc, listDest]


#αφαίρεση διπλότυπων
def unique(mylist):
        end = len(mylist)
        i = 0
        while (i<end-1):
                if (mylist[i] == mylist[i+1]):
                        if i!=len(mylist)-1:
                                mylist = mylist[:i] + mylist[i+1 :]
                        else:
                                mylist = mylist[:i]
                        end-=1
                        i-=1
                i+=1
        return mylist


#openflow:1000:10 --> 10
def portNumber(mystring):
        s = mystring
        start = s.find("openflow:") + 9
        end = s.find(":",start)
        s1 =  s[start:end]
        s = "openflow:"+s1+":"
        start = mystring.find(s) +len(s)
        end = mystring.find("\"",start)
        s2 = ( mystring[start:end])
        s1 = int(s1)
        s2 = int(s2)
        return s2


#host:ca:c4:d7:e5:0a:33 --> ca:c4:d7:e5:0a:33
def macAddress(mystring):
        s = mystring
        start = s.find("host:") + 5
        end = s.find("\"",start)
        s1 = s[start:end]
        return str(s1)


#list από : "openflow:1000" -> list από : 1000
def takeID(mylist):
        length = len(mylist)
        i = 0
        while (i<length):
                s = mylist[i]
                start = s.find("openflow:") +9
                end = s.find("\"",start)
                s1 =  s[start:end]
                mylist[i] = int(s1)
```

```
                i+=1
        return mylist


#1 εάν πρόκειται για access switch
def ifAccess(ii, listMappingNumber):
        if (listMappingNumber[ii]>0):
                access = 1
        else:
                access =0
        return access


inventory = "curl -u <USERNAME>:<PASSWORD> -X GET -s http://<ODL-
IP>:8181/restconf/operational/opendaylight-inventory:nodes/"
topology = "curl -u <USERNAME>:<PASSWORD> -X GET -s http://<ODL-
IP>:8181/restconf/operational/network-topology:network-topology/"
addFlow = "curl --noproxy <ODL-IP> -u <USERNAME>:<PASSWORD> -H 'Content-Type:
application/yang.data+xml' -X PUT -d"

link =  topology + "| jq '.[\"network-topology\"].topology[0].link[] | select(length >0 )  ' >
text.json "
subprocess.call(link , shell = True)

cat = "cat text.json | jq '.[\"link-id\"] ' "
linkNode = subprocess.check_output(cat , shell = True)
linkID = linkNode.split()

cat = "cat text.json |  jq '.source | .[\"source-node\"] ' "
sourceNode = subprocess.check_output(cat , shell = True)
sourceID = sourceNode.split()

cat = "cat text.json |  jq '.source | .[\"source-tp\"] ' "
linkSourceNode = subprocess.check_output(cat , shell = True)
linkSourceID = linkSourceNode.split()

cat = "cat text.json |  jq '.destination | .[\"dest-tp\"] ' "
linkDestNode = subprocess.check_output(cat , shell = True)
linkDestID = linkDestNode.split()

hostMac = []
hostPort = []
hostSwitch = []

#Πληροφορίες για host: mac,port του switch στο οποίο συνδέεται, switch στο οποίο συνδέεται
for i in range(0, len(sourceID)):
        if ("host" in sourceID[i]):
                hostMac.append([])
                hostPort.append([])
                hostSwitch.append([])


#επιλογή όλων των switch
```

```
[sourceID] = clear(sourceID, "host")
sourceID = takeID(sourceID)
sourceID.sort()

switchID = unique(sourceID)

#links μόνο μεταξύ switch-host
[linkSourceID , linkDestID ] = onlyHost(linkSourceID , linkDestID )

mappingFirst = []
mappingNumber = []
switchName = []

for i in range(0, len(switchID)):
        switchName.append([])                   #όνομα switch
        mappingNumber.append([])                #πλήθος host που συνδέονται πάνω στο
συγκεκριμένο switch
        mappingNumber[i] = 0
        mappingFirst.append([])                         #indexing για εύρεση των host του κάθε
switch
        mappingFirst[i] = 0


#Εύρεση Τοπολογίας
k = 0
for i in range(0,len(switchID)):
        link = inventory +"node/openflow:"+str(switchID[i]) +"/node-
connector/openflow:"+str(switchID[i]) +":LOCAL/|jq '.[] | .[] | {\"flow-node-inventory:name\" }
| .[\"flow-node-inventory:name\"]' "
        switchName[i] = subprocess.check_output(link , shell = True)
        first = 0
        for j in range (0,len(linkSourceID)):
                if ("openflow:"+str(switchID[i])+":" in linkSourceID[j]):
                        if (first == 0):
                                mappingFirst[i] = k
                                first = 1
                        hostPort[k] = portNumber(linkSourceID[j])
                        hostMac[k] = macAddress(linkDestID[j])
                        hostSwitch[k] = i
                        mappingNumber[i] += 1
                        k += 1

#Προσθήκη των flows
flow_id =1
for i in range(0,len(switchID)):
        if (ifAccess(i,mappingNumber)):
                for j in range(mappingFirst[i],mappingFirst[i] + mappingNumber[i]):
                        for k in range (0,mappingFirst[i]):
                                command = addFlow + " \' <flow
xmlns=\"urn:opendaylight:flow:inventory\"> <priority>65535</priority> <flow-name> " +
str(flow_id) + "</flow-name> <match> <ethernet-match> <ethernet-type> <type>2048</type>
</ethernet-type> <ethernet-destination> <address>"+hostMac[j]
                                command = command + "</address> </ethernet-destination>
```

```
<ethernet-source> <address>" + hostMac[k] + "</address> </ethernet-source> </ethernet-
match> </match> <id>" + str(flow_id) + "</id> <table_id>0</table_id><instructions>
<instruction> <order>0</order> <apply-actions> <action>"
                                    command = command + "<order>0</order> <output-action>
<output-node-connector>" +str(hostPort[j]) + "</output-node-connector> <max-
length>60</max-length> </output-action> </action> </apply-actions> </instruction>
</instructions> </flow>\'"
                                    command = command + " \'http://<ODL-
IP>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:"+str(switchID[i])  +
"/table/0/flow/"+ str(flow_id) + "\' "
                                    command = command.replace('\n', '')
                                    subprocess.call(command , shell = True)
                                    flow_id += 1
                            for k in range (mappingFirst[i] + mappingNumber[i],len(hostMac)):
                                    command = addFlow + " \' <flow
xmlns=\"urn:opendaylight:flow:inventory\"> <priority>65535</priority> <flow-name> " +
str(flow_id) + "</flow-name> <match> <ethernet-match> <ethernet-type> <type>2048</type>
</ethernet-type> <ethernet-destination> <address>"+hostMac[j]
                                    command = command + "</address> </ethernet-destination>
<ethernet-source> <address>" + hostMac[k] + "</address> </ethernet-source> </ethernet-
match> </match> <id>" + str(flow_id) + "</id> <table_id>0</table_id><instructions>
<instruction> <order>0</order> <apply-actions> <action>"
                                    command = command + "<order>0</order> <output-action>
<output-node-connector>" +str(hostPort[j]) + "</output-node-connector> <max-
length>60</max-length> </output-action> </action> </apply-actions> </instruction>
</instructions> </flow>\'"
                                    command = command + " \'http://<ODL-
IP>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:"+str(switchID[i])  +
"/table/0/flow/"+ str(flow_id) + "\' "
                                    command = command.replace('\n', '')
                                    subprocess.call(command , shell = True)
                                    flow_id += 1
```

```
deleteFlow = "curl --noproxy <ODL-IP> -u <USERNAME>:<PASSWORD> -H 'Content-Type:
application/yang.data+xml' -X DELETE -d"

for i in range(0,len(switchID)):
        if (ifAccess(i,mappingNumber)):
                for j in range(mappingFirst[i],mappingFirst[i] + mappingNumber[i]):
                        for k in range (0,mappingFirst[i]):
                                    command = deleteFlow + " \' <flow
xmlns=\"urn:opendaylight:flow:inventory\"> <priority>65535</priority> <flow-name> " +
str(flow_id) + "</flow-name> <match> <ethernet-match> <ethernet-type> <type>2048</type>
</ethernet-type> <ethernet-destination> <address>"+hostMac[j]
                                    command = command + "</address> </ethernet-destination>
<ethernet-source> <address>" + hostMac[k] + "</address> </ethernet-source> </ethernet-
match> </match> <id>" + str(flow_id) + "</id> <table_id>0</table_id><instructions>
<instruction> <order>0</order> <apply-actions> <action>"
                                    command = command + "<order>0</order> <output-action>
<output-node-connector>" +str(hostPort[j]) + "</output-node-connector> <max-
```

```
length>60</max-length> </output-action> </action> </apply-actions> </instruction>
</instructions> </flow>\'"
                                command = command + " \'http://<ODL-
IP>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:"+str(switchID[i])  +
"/table/0/flow/"+ str(flow_id) + "\' "
                            #OFCTL , Mininet
                            #       command = "ovs-ofctl add-flow  " + noquotes(switchName[i]) +
"  \"dl_src=" + hostMac[k] + "  dl_dst=" + hostMac[j]+",  in_port=*
actions=output:"+str(hostPort[j]) +"\'"
                                command = command.replace('\n', '')
                                subprocess.call(command , shell = True)
                                flow_id += 1
                            for k in range (mappingFirst[i] + mappingNumber[i],len(hostMac)):
                                command = deleteFlow + " \' <flow
xmlns=\"urn:opendaylight:flow:inventory\"> <priority>65535</priority> <flow-name> " +
str(flow_id) + "</flow-name> <match> <ethernet-match> <ethernet-type> <type>2048</type>
</ethernet-type> <ethernet-destination> <address>"+hostMac[j]
                                command = command + "</address> </ethernet-destination>
<ethernet-source> <address>" + hostMac[k] + "</address> </ethernet-source> </ethernet-
match> </match> <id>" + str(flow_id) + "</id> <table_id>0</table_id><instructions>
<instruction> <order>0</order> <apply-actions> <action>"
                                command = command + "<order>0</order> <output-action>
<output-node-connector>" +str(hostPort[j]) + "</output-node-connector> <max-
length>60</max-length> </output-action> </action> </apply-actions> </instruction>
</instructions> </flow>\'"
                                command = command + " \'http://<ODL-
IP>:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:"+str(switchID[i])  +
"/table/0/flow/"+ str(flow_id) + "\' "
                            #OFCTL , Mininet
                            #       command = "ovs-ofctl add-flow  " +  noquotes(switchName[i])
+ "  \"dl_src="+ hostMac[k] + "  dl_dst="+hostMac[j]+",  in_port=*
actions=output:"+str(hostPort[j]) + "\'"
                                command = command.replace('\n', '')
                                subprocess.call(command , shell = True)
                                flow_id += 1
```

```
#!/usr/bin/python

import subprocess
import json

cred = 'curl --noproxy <ODL-IP> -u <USERNAME>:<PASSWORD> -H \'Content-Type:
application/yang.data+xml\' -X DELETE -d '
url = 'http://<ODL-IP>:8181/restconf/config/opendaylight-inventory:nodes/'
link = 'curl -u <USERNAME>:<PASSWORD> -H "Accept: application/xml"   -X GET ' + url

config = subprocess.check_output(link , shell = True)

delete_link = cred + "\'" + config + "\' " + "\'" + url + "\' "
```

```
subprocess.call(delete_link , shell = True)
```

```
#!/usr/bin/python

import os
import sys
import subprocess
import string

def noquotes(mystring):
        tempStr = mystring.replace('\'', '')
        return tempStr


subprocess.call("oplrun SDN_flavor.mod data.txt", shell = True)
shut_links = subprocess.check_output("cat shut_links.txt", shell = True)
shut_links = shut_links.replace('\n    ','')
shut_links = shut_links.replace('[','')
shut_links = shut_links.replace(']','')
shut_links = shut_links.split()

for i in range(0,len(shut_links)):
        shut_links[i] = int( shut_links[i] )

fo = open("whichSwitches.txt", "r+")

numberOfLinks = int(fo.readline())

switchOne = fo.readline()
switchOne = switchOne.replace('\n','')
switchOne = switchOne.split()
switchTwo = fo.readline()
switchTwo = switchTwo.replace('\n','')
switchTwo = switchTwo.split()

for i in range(0,len(switchOne)):
        switchOne[i] = int(switchOne[i])
        switchTwo[i] = int(switchTwo[i])

con = open('connectedSw.txt',"r+")
por = open('connectedPort.txt',"r+")
interf = open ("interface.txt","r+")
swn = open("switchName.txt","r+")

numberOfSwitches = int(con.readline())

connectedSw = [[]]
connectedSw = [[0 for i in xrange(numberOfSwitches)] for j in xrange(numberOfSwitches)]
portSw = [[]]
portSw = [[0 for i in xrange(numberOfSwitches)] for j in xrange(numberOfSwitches)]
```

```
interfaceName = [[]]
interfaceName = [[0 for i in xrange(numberOfSwitches)] for j in xrange(numberOfSwitches)]

for i in range(0,numberOfSwitches):
        tempLine = con.readline()
        connectedSw[i] = tempLine.split()

        tempLine = por.readline()
        portSw[i] = tempLine.split()

        tempLine = interf.readline()
        interfaceName[i] = tempLine.split()


switchName = swn.readline()
switchName = switchName.split()

fo.close()
con.close()
por.close()
interf.close()
swn.close()

i=0

ports = open('PORTS.txt','w', 0)

while (i<len(shut_links)):
        if (shut_links[i] == 0 ):
                command = "ovs-ofctl mod-port  " + noquotes(switchName[switchOne[i/2]]) +"
"+ noquotes(interfaceName[switchOne[i/2]][switchTwo[i/2]]) + "  down"
                command = command.replace('\n', ' ')
                ports.write( command )
                ports.write( "\n" )

                command = "ovs-ofctl mod-port  " + noquotes(switchName[switchTwo[i/2]]) +"
"+ noquotes(interfaceName[switchTwo[i/2]][switchOne[i/2]]) + "  down"
                command = command.replace('\n', ' ')
                ports.write( command )
                ports.write( "\n" )

        else:
                command = "ovs-ofctl mod-port  " + noquotes(switchName[switchOne[i/2]]) +"
"+ noquotes(interfaceName[switchOne[i/2]][switchTwo[i/2]]) + "  up"
                command = command.replace('\n', ' ')
                ports.write( command )
                ports.write( "\n" )

                command = "ovs-ofctl mod-port  " + noquotes(switchName[switchTwo[i/2]]) +"
"+ noquotes(interfaceName[switchTwo[i/2]][switchOne[i/2]]) + "  up"
                command = command.replace('\n', ' ')
                ports.write( command )
                ports.write( "\n" )
```

```
        i += 2

ports.write( "\n" )
ports.close()
```

```
"""
   Fattree Topology with 2 Cores, 8 Aggr, 16 TOR/Access
   Created by akaratrantou
   Modified by pchara
         Works for all numbers of switches
         Added Second links
         Added Core link connection
         version 2.0 14/07/2016
"""

from mininet.topo import Topo
from mininet.link import TCLink

class FatTree( Topo ):

   def __init__( self ):
      coreSwitchnumber = 2
      aggregateSwitchnumber = 8
      torSwitchnumber = 16
      racksize = 16

      # Initialize topology
      Topo.__init__( self )

      coreSwitches = []
      aggregateSwitches = []
      torSwitches = []


      # Core Switches
      for x in range(0, coreSwitchnumber):
         coreSwitches.append(self.addSwitch("100" + str(x)))
      # Aggregate Switches
      for x in range(0, aggregateSwitchnumber):
         aggregateSwitches.append(self.addSwitch("200" + str(x)))
      # TOR Switches (Top-Of-Rack)
      for x in range(0, torSwitchnumber):
         torSwitches.append(self.addSwitch("300" + str(x)))

      # Aggregate -> Core Connection
      for y in range(0, len(aggregateSwitches)):
         for x in range (0, len (coreSwitches)):
            self.addLink(coreSwitches[x], aggregateSwitches[y])
```

```
            print 'Core:' + str(x) + ' Ag:' + str(y)

        # TOR -> Aggregate Connection
        for x in range(0, len(aggregateSwitches), 2):
            for y in range(x*len(torSwitches)/len(aggregateSwitches),
(x+2)*len(torSwitches)/len(aggregateSwitches),1):
                self.addLink(aggregateSwitches[x],torSwitches[y])
                self.addLink(aggregateSwitches[x+1],torSwitches[y])

        # Hosts -> TOR switches
        for x in range(0, len(torSwitches) , 2):
            for y in range(0, racksize):
                self.addLink(torSwitches[x], self.addHost("400" + str(x) + str(y)))
                self.addLink(torSwitches[x+1], self.addHost("400" + str(x+1) + str(y)))


            # Connect core switches
            for x in range(0, coreSwitchnumber , 2):
                    self.addLink(coreSwitches[x],coreSwitches[x+1])


topos = { 'fattree': ( lambda: FatTree() ) }
```