Ατομική Διπλωματική Εργασία

SDN Network for Smart Home

Σωτήρης Μιχαηλίδης

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Δεκέμβριος 2020

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

SDN Network for Smart Home

Σωτήρης Μιχαηλίδης

Επιβλέπων Καθηγητής
Αντρέας Πιτσιλλίδης

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Δεκέμβριος 2020

Ευχαριστίες

Πρωτίστως θα ήθελα να εκφράσω τις ευχαριστίες και την ευγνωμοσύνη μου στον επιβλέποντα καθηγητή μου Δρ. Αντρέα Πιτσιλλίδη ο οποίος με εμπιστευτικέ και παράλληλα μου έδωσε την ευκαιρία να συνεργαστώ μαζί του. Τον ευχαριστώ για όλη την πολύτιμη βοήθεια και τις χρήσιμες συμβουλές που μου πρόσφερε σε όλη την διάρκεια της διπλωματικής μου εργασίας.

Επιπρόσθετα, εκφράζω εγκάρδιες ευχαριστίες στον κο. Ιάκωβο Ιωάννου, ο οποίος, με βοήθησε στην διεκπεραίωση της διπλωματικής μου εργασίας, αφού ο ίδιος κατέχει άριστες γνώσεις στον τομέα των δικτύων, και όχι μόνο. Εκτός από την διπλωματική μου εργασία, ο κος. Ιάκωβος μου πρόσφερε πολύτιμες συμβουλές για την μελλοντική μου ακαδημαϊκή πορεία. Εύχομαι στον κο. Ιάκωβο, κάθε επιτυχία στην διδακτορική του διατριβή και στην μετέπειτα πορεία του.

Επίσης ευχαριστώ θερμά τον Δρ. Γιάννο Μυλωνά , πέραν των όσων μου δίδαξε στον τομέα των δικτύων , για όλες τις συμβουλές του σχετικά με την μετέπειτα πορεία μου.

Επιπλέων θα ήθελα να ευχαριστήσω όλους τους καθηγητές , από τους οποίους είχα την τιμή να διδαχθώ και να πάρω πολλές γνώσεις στον τομέα της πληροφορικής αλλά και γενικά της ζωής.

Τέλος νιώθω την ανάγκη να εκφράσω την βαθιά μου ευγνωμοσύνη προς την οικογένεια μου και τους φίλους μου που ήταν δίπλα μου καθόλη την διάρκεια των σπουδών μου.

Περίληψη

Τα τελευταία 30 χρόνια , παρατηρείται ένα χάσμα μεταξύ της εξέλιξης της τεχνολογίας γενικώς και της εξέλιξης του τρόπου λειτουργίας των δικτύων . Ενώ η τεχνολογία σχεδόν σε όλους τους τομής της έχει σημειώσει σημαντική πρόοδο, η αρχιτεκτονική των δικτύων μας είναι ακριβώς η ίδια , με μηχανήματα κλειστού λογισμικού από εξειδικευμένες εταιρίες κολοσσούς (π.χ Cisco), τα οποία δεν μπορεί να εκμεταλλευτεί ο χρήστης στο 100% των δυνατοτήτων τους , αλλά ούτε και να τα ρυθμίσει στις ανάγκες του. Αυτή η έλλειψη ελέγχου στα μηχανήματα, είναι και ο λόγος που δεν υπήρξε κάποια σημαντική καινοτομία στον τομέα αυτό. Με τον ερχομό όμως , καινούριων τεχνολογιών όπως το cloud computing , IoT , NFV, συνειδητοποιήσαμε ότι δεν μπορεί να συνεχιστεί αυτή η στασιμότητα στον τομέα των δικτύων, καθώς αυτά δεν θα μπορούν να μας εξυπηρετούν για πάντα.

Η διπλωματική μου εργασία ασχολείται με τα SDN δίκτυα τα οποία προσπαθούν να καλύψουν αυτό το χάσμα, ακολουθώντας πορεία διαχωρισμού software και hardware. Αν και τα αποτελέσματα αυτών των δικτύων είναι πολύ υποσχόμενα , αφού ήδη χρησιμοποιούνται από μεγάλες εταιρίες και αποτελούν βασικό πυλώνα των ασύρματων δικτύων 5ης γενεάς , δεν παύουν να υπάρχουν δυσκολίες και αμφιβολίες κατά πόσο μπορούν να χρησιμοποιηθούν εκτός κέντρων δεδομένων (Data centers).

Στόχος λοιπόν της διπλωματικής μου εργασίας η δημιουργία ενός SDN δικτιού , αρχικά με simulator και στην συνέχεια με πραγματικό hardware , για να εξεταστεί κατά πόσο τα SDN δίκτυα μπορούν να αντικαταστήσουν τα τρέχοντα δίκτυα στα σπίτια μας /μικρές επιχρίσεις και κατά πόσο μπορούν να βελτιωθούν . Επίσης βασικές λειτουργίες απαραίτητες σε ένα σπιτικό δίκτυο όπως firewall κτλπ. θα εξεταστούν κατά πόσο είναι εφικτό να συνυπάρξουν με ένα SDN δίκτυο. Το δίκτυο αυτό θα χρησιμοποιεί μόνο προγράμματα ανοικτού κώδικα , και μηχανήματα χαμηλού κόστους για να μπορεί να απευθυνθεί στο μέσο χρήστη. Τέλος θα διερευνηθεί κατά πόσο τα SDN, δίκτυα μπορούν να συνυπάρξουν με τα ήδη υπάρχοντα δίκτυα.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Γενική Εισαγωγή

Η εποχή που οι Η/Υ ήταν κάποια μαύρα κουτιά που λειτουργικό σύστημα και εφαρμογές ήταν αλληλένδετα και αδιαίρετα συνδεδεμένα με το hardware , έχει τελειώσει. Πλέον hardware , εφαρμογές και λειτουργικό σύστημα είναι ανεξάρτητα και επικοινωνούν μεταξύ τους βάση ανοικτών πρωτοκόλλων . Με άλλα λόγια κάποιος μπορεί να αγοράσει από διαφορετικές εταιρίες , λειτουργικό σύστημα και hardware , ή ακόμη να δημιουργήσει δικά του συστατικά στοιχεία και να τα συνδυάσει όλα μαζί. Με αυτή την νέα τάξη πραγμάτων, δόθηκε η ευκαιρία στους ανθρώπους της τεχνολογίας να πειραματιστούν, να καινοτομήσουν και να σκεφτούν καινούρια πράγματα και τεχνολογίες χωρίς περιορισμούς.

Από την άλλη, αν κάποιος συγκρίνει την αρχιτεκτονική των δικτύων σήμερα με αυτήν πριν από 30-40 χρόνια θα δει ότι έχει μείνει σχεδόν στάσιμη . Υπηρεσίες δικτύων παρέχονται από μερικούς εξειδικευμένους πάροχους , με μηχανήματα δικτύου στα οποία δεν έχει πρόσβαση ο χρήστης αφού προστατεύονται από δικαιώματα πνευματικής ιδιοκτησίας και κλειστού λογισμικού που ορίζονται από τους κατασκευαστές και τους πάροχους.

Τα SDN δίκτυα (Software Define Networks) αποτελούν μια αρχιτεκτονική δικτύου παρόμοια με αυτήν των Η/Υ, διαχωρίζουν δηλαδή software και hardware. Πλέον, αντί για εξειδικευμένους routers , υπάρχουν απλά μηχανήματα προώθησης πακέτων τα οποία επικοινωνούν με καθορισμένο τρόπο (με προκαθορισμένα ανοικτά πρωτόκολλα) με κάποιο εξειδικευμένο λογισμικό (ελεκτής) το οποίο είναι υπεύθυνο για την δρομολόγηση , εύρεση λύσης σε περίπτωση κάποιας βλάβης, επίτευξη QoS και όλες τις άλλες λειτουργίες τις οποίες μέχρι τώρα εκτελούσε ο δρομολογητής . Αυτή η αφαίρεση όλων των σύνθετων λειτουργιών από το hardware, επιτρέπει στον προγραμματιστή να προγραμματίσει την λειτουργεία του δικτύου. Κύριος στόχος αυτών των δικτύων είναι να επιτρέψουν τον έλεγχο τους από τους χρήστες και τους διαχειριστές τους χωρίς να βασίζονται στις εταιρίες κατασκευής του hardware , φέρνοντας την εξέλιξη και την καινοτομία που απουσιάζει τα τελευταία χρόνια.

1.2 Κίνητρο και Στόχος

Η ιδέα των SDN δικτιών δεν είναι καινούρια, αφού "υπάρχει στο τραπέζι" για πάνω από 10 χρόνια. Παρόλ' αύτα δεν χρησιμοποιούνται πουθενά πλην στα data centers κάποιων μεγάλων εταιριών όπως η Google , η Amazon και η Microsoft. Στα πλαίσια του μαθήματος ΕΠΛ450 , «Network Management», σε μια ατομική εργασία , ερεύνησα αυτήν την καινούργια αρχιτεκτονική των δικτύων αφού ήταν αρκετά υποσχόμενη. Συγκεκριμένα , μελέτησα κάποιες αναφορές της Google σχετικά με τις SDN υποδομές της, και τα πράγματα που πέτυχε με αυτές. Κατάλαβα ότι αυτή η αρχιτεκτονική ίσως είναι το μέλλον των δικτύων .

Στον τομέα των δικτύων όμως , ευτυχώς ή δυστυχώς , κυριαρχεί η σκέψη του ότι «Δεν αλλάζεις κάτι το οποίο λειτουργεί» , και όντως τα δίκτυα μας , μέχρι τώρα μας εξυπηρετούν αρκετά καλά. Με την άνοδο όμως του Cloud computing, την ανάγκη για περισσότερο Bandwidth (με την μεταφορά / επεξεργασία των Big Data), την είσοδο μας στην εποχή του IoT, καθώς επίσης και την ανάγκη για καλύτερη διαχείριση των δικτύων μας, πιστεύω ότι τα SDN δίκτυα ίσως να είναι ο μόνος δρόμος.

Έτσι , αποφάσισα να δημιουργήσω ένα SDN δίκτυο με φτηνά συστατικά στοιχεία, για να βοηθήσω στην εδραίωση τους. Αποσκοπεί σε μικρές εταιρίες και σπίτια, που θέλουν φτηνά αποδοτικά δίκτυα με περισσότερο έλεγχο, θα τους δοθεί η βάση να εγκαταστήσουνε οι ίδιοι το δικό τους SDN δίκτυο.

Κεφάλαιο 2

SDN και OPENFLOW

2.1 Αρχιτεκτονική SDN και Παραδοσιακών δικτύων

2.1.1. Παραδοσιακά δίκτυα (traditional networks)

Ως παραδοσιακά δίκτυα , ονομάζουμε τα δίκτυα της εποχής μας. Αποτελούνται από ένα σύνολο μηχανημάτων δικτύου (κυρίως routers/switches) , τα οποία έχουν προκαθορισμένο και αμετάλλακτο σύνολο λειτουργιών. Οι λειτουργίες αυτές υποστηρίζουν / λειτουργούν καλά μαζί και υποστηρίζουν το δίκτυο . Αρκετές λειτουργίες ενσωματώνονται στο επίπεδο hardware. Το μεγάλο πρόβλημα των δικτύων αυτών είναι η έλλειψη ευελιξίας λόγο του κλειστού λογισμικού το οποίο τρέχουν. Υπάρχουν πολύ λίγα APIs τα οποία δεν δίνουν την δυνατότητα στους τελικούς χρήστες για πλήρη έλεγχο. Με άλλα λόγια , το λογισμικό κάποιας εταιρίας μπορεί να λειτουργεί άψογα με το μηχάνημα της , όμως το λογισμικό αυτό δεν μπορεί να αλλάξει (ή να παραμετροποιηθεί) ευκολά και γρήγορα , ανάλογα με τις ανάγκες του εκάστοτε χρηστή .

2.1.2 Control/Data plane

Control plane: Είναι αφαιρετική έννοια και αναφέρεται στο σύνολο των λειτουργιών , οι οποίες αποφασίζουν πως θα κινηθούν τα πακέτα , για να φτάσουν στον τελικό τους προορισμό. Είναι σημαντικό να αναφερθεί ότι ενσωματώνει και πολλές λειτουργίες του management plane αφού αυτό βρίσκεται στα αρχικά του στάδια και δεν έχει εδραιωθεί . Μερικές λειτουργίες του είναι:

1) Να τρέξει αλγόριθμους δρομολόγησης και να δημιουργήσει πίνακες δρομολόγησης .

2) Παροχή υπηρεσιών διαχείρισης (system configuration/management etc.)

3) Υποστήριξη διαφόρων πρωτοκόλλων όπως ARP / DHCP / STP

Data/forwarding plane : Είναι επίσης αφαιρετική έννοια και αναφέρεται σε όλες τις λειτουργίες και διαδικασίες που λαμβάνουν μέρος για να μεταφερθεί κάποιο πακέτο από ένα interface σε

κάποιο άλλο (στον ίδιο δρομολογητή), με απώτερο σκοπό να φτάσει το πακέτο στο τελικό του προορισμό. Οι αποφάσεις που λαμβάνονται στο Data plane είναι εξαρτώμενες από τις αποφάσεις του Control plane. Η σχέση του με το Control plane μπορεί να χαρακτηριστεί σαν Master-Slave αφού το Control plane είναι ο εγκέφαλος και το Forward plane αυτό που υλοποιεί ότι απόφαση παρθεί από αυτόν.

2.1.3 SDN δίκτυα

Στα υφιστάμενα δίκτυα, τα Control Plane και Data plane παρόλο που είναι διαχωρισμένα σε λογικό επίπεδο , παραμένουν ενωμένα σε επίπεδο υλικού. Δηλαδή κάθε μηχάνημα δικτύου περιλαμβάνει και τα 2 planes. Στα SDN δίκτυα , τα 2 planes διαχωρίζονται και σε φυσικό επίπεδο σε διαφορετικέ μηχανές. Αυτός ο διαχωρισμός επιτρέπει στο δίκτυο να είναι απευθείας συνδεδεμένο με τις εφαρμογές μέσω διαφόρων APIs. Έτσι δίνεται η δυνατότητα για ένα δίκτυο με μεγαλύτερη απόδοση , ποιο ευέλικτο , ποιο ασφαλές , και μια αρχιτεκτονική δικτύου η οποία μπορεί να αλλάζει δυναμικά ανάλογα με την περίσταση .

2.1.4 SDN Components

1) (OpenFlow) Switches : Απλό forwarding hardware το οποίο είναι υπεύθυνο για όλες τις λειτουργίες του Data plane (sdn switches) . Αποτελούνται από 1 ή περισσότερους forwarding tables βάση των οποίων προωθούν πακέτα . Κάθε switch επικοινωνεί με κάποιον controller μέσω ενός (ή περισσότερων) καναλιού OpenFlow.

Model	Virtualize	Notes	
HP Procurve 5400zl or 6600	1 OF instance per VLAN	-LACP, VLAN and STP processing before OpenFlow -Wildcard rules or non-IP pkts processed in s/w -Header rewriting in s/w -CPU protects mgmt during loop	
NEC IP8800	1 OF instance per VLAN	-OpenFlow takes precedence -Most actions processed in hardware -MAC header rewriting in h/w	
Pronto 3290 or 3780 with Pica8 or Indigo firmware	1 OF instance per switch	-No legacy protocols (like VLAN and STP) -Most actions processed in hardware -MAC header rewriting in h/w	

Εικόνα 1 : OF Switches

2) Controller : Σύνθετο software το οποίο τρέχει συνήθως απομακρυσμένα (αλλά μπορεί και τοπικά στην ίδια μηχανή) το οποίο είναι υπεύθυνο για όλες τις λειτουργίες του Control plane. Επικοινωνεί με τα switches μέσω του πρωτοκόλλου OpenFlow και εγκαθιστά εγγραφές στα forwarding tables (των switches) με σκοπό τα πακέτα να φτάσουν στο προορισμό τους. Αλγόριθμοί δρομολόγησης και εύρεσης καλύτερου μονοπατιού τρέχουν στον controller.

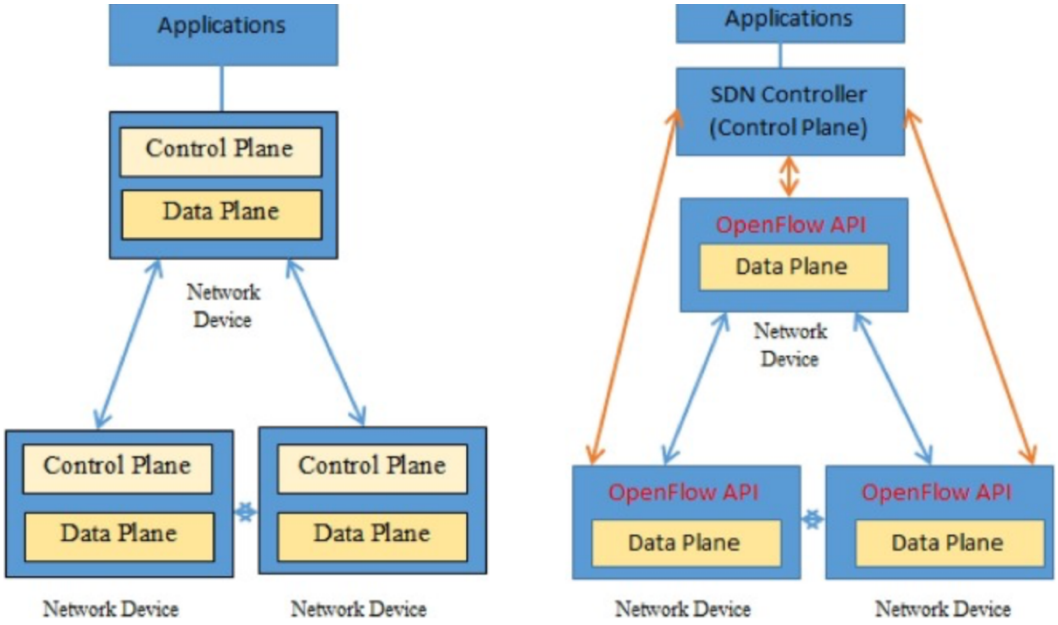
Vendor	Notes
Nicira's NOX	•GPL •C++ and Python
SNAC	•GPL •Code based on NOX0.4 •Enterprise network •C++, Python and Javascript •Currently used by campuses

Vendor	Notes
Stanford's Beacon	•BSD-like license •Java-based
Maestro (from Rice Univ)	•GPL •Based on Java
NEC's Trema	•Open-source •Written in C and Ruby •Included test harness

Εικόνα 2 : SDN Controllers

3) OpenFlow: Η επικοινωνία μεταξύ Controller – Switch (γνωστό και σαν 'southbound interface') γίνεται βάση προκαθορισμένων πρωτοκόλλων , συνήθως του πρωτοκόλλου OpenFlow (όπως προαναφέρθηκε) το οποίο θα αναλυθεί αργότερα.

4) APIs: Ο controller επίσης επικοινωνεί με τις διάφορες εφαρμογές μέσω διαφόρων APIs (γνωστό και σαν 'northbound interface'). Μέσω αυτής της επικοινωνίας οι developers μπορούν να προγραμματίσουν απευθείας το δίκτυο.



Εικόνα 3 : αρχιτεκτονική παραδοσιακών (αριστερά) και sdn δικτυών δεξιά

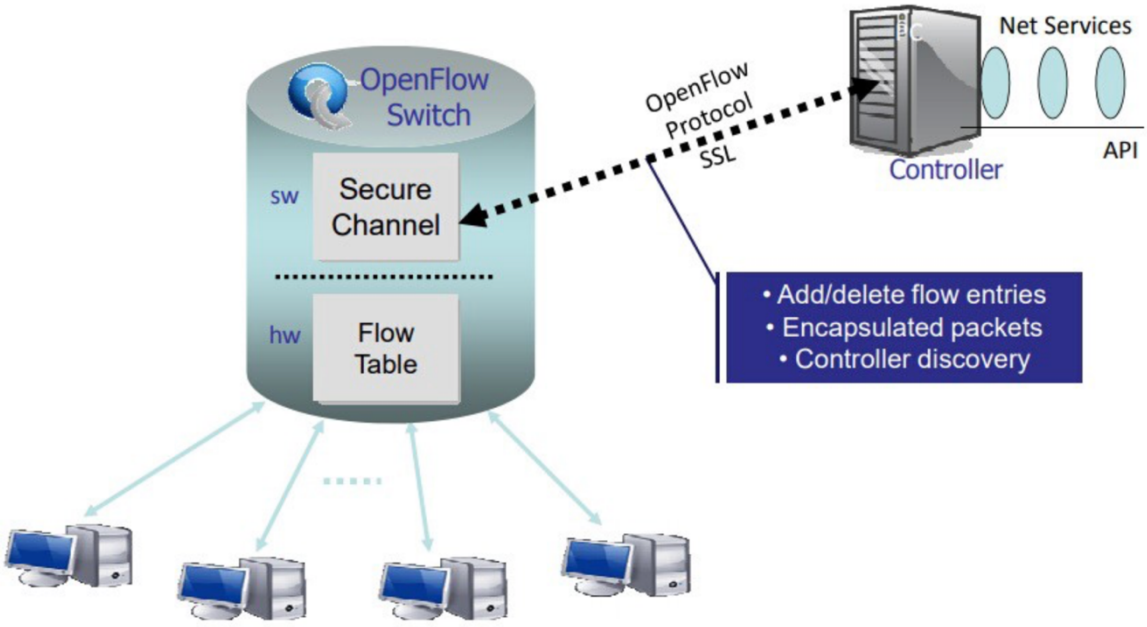
2.2 Πρωτόκολλο OpenFlow

2.2.1 OpenFlow

Το OpenFlow είναι το πρωτόκολλο επικοινωνίας μεταξύ του Controller και των OF-Switches. Είναι ουσιαστικά το southbound interface στην αρχιτεκτονική SDN. Ορίζει το περιεχόμενο και την μορφή των μηνυμάτων που ανατάσσουν το Control και Data plane. Είναι ένα ανοικτό πρωτόκολλο, του οποίου η χρήση επιτρέπει στο δίκτυο να λειτουργεί με switches από διαφορετικούς κατασκευαστές.
OpenFlow δεν σημαίνει SDN , αλλά λόγο κοινής κατεύθυνσης και φιλοσοφίας μπορεί να βοηθήσει στην ανάπτυξη και την λειτουργία του SDN , γι' αυτό τείνει να γίνει το de-facto πρωτόκολλο των SDN δικτύων.

Το OpenFlow , τρέχει πάνω από TCP (Transmission Control Protocol) , ποιο συγκεκριμένα σε TLS(Transport Layer Security) και ακούει το port 6633. Επιτρέπει την απομακρυσμένη

διαχείριση layer-3 switches από τον Controller, με την δυνατότητα για εισαγωγή , επεξεργασία και διαγραφή εγγραφών στα Forwarding/Flow tables των OpenFlow switches.



Εικόνα 4 : αρχιτεκτονική OF

2.2.2 Τύποι μηνυμάτων OpenFlow

Υπάρχουν 3 τύποι μηνυμάτων με τα οποία μπορούν να επικοινωνήσουν ο controller και κάποιο switch:

1. Συμμετρικά: Μπορούν να σταλθούν από και προς τον Controller / Switch χωρίς να απαιτείται κάποια προ-υπάρχουσα σχέση ή επικοινωνία . Παραδείγματα συμμετρικών μηνυμάτων είναι : •Hello message
   •Echo request / echo reply message
   •Error message
   •Experimenter message

   Μόνη διαφορά , το γεγονός που παρότι και ένα switch ενώ μπορεί να στείλει κάποιο Error μήνυμα , δεν έχει την δυνατότητα να επεξεργαστεί κάποιο .

2. Ασύγχρονα : Μπορούν να σταλθούν από και προς τον Controller / Switch μόνο όταν υπάρχει κάποια αλλαγή στην κατάσταση του συστήματος. Παραδείγματα ασύγχρονων μηνυμάτων είναι : Packet-in message
   - Flow-removed
   - Table/Port-status
   - Controller-role status

3. Controller-to-Switch : Μηνύματα που στέλνει ο Controller στο Switch όταν θέλει να πάρει συγκεκριμένες πληροφορίες από αυτό , ή να επεξεργαστεί εγγραφές στο Flow table .

   Παραδείγματα Controller-to-Switch μηνυμάτων είναι:
   - Flow-mod message
   - Port-mod message
   - Stats-request message

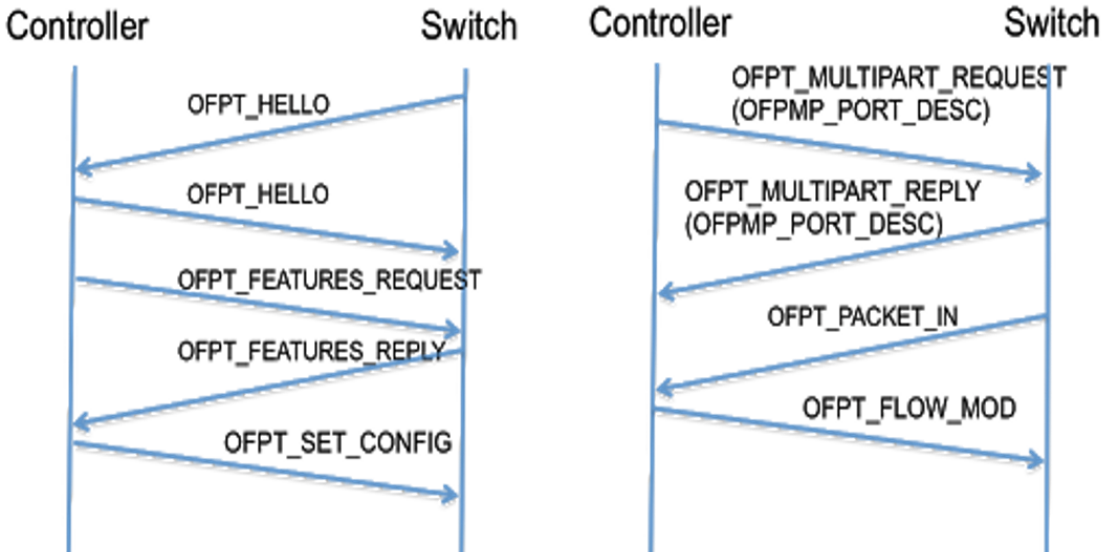
2.2.3 OpenFlow Μηνύματα

Βασικά μηνύματα OpenFlow

Message	Type	Description
Hello	Controller->Switch	Following the Tcp Handshake Controller sends its version of OF.
Hello	Switch->Controller	Switch replies with its supported version of OF
Features Request	Controller->Switch	Controller asks for the switch's Available ports
Set Config	Controller->Switch	Controller sets the fragmentation handling properties of the packet
Features Reply	Switch->Controller	Switch replies with its available ports to controller
Port Status	Switch->Controller	Switch informs controller with the status of some ports
Echo Request	Controller->Switch	Controller sends periodically echo request packets to gather various information of the

		switches. If a switch does not reply , it is consider out of order.
Echo Reply	Switch->Controller	Switch replies to the Controller echo request message, with the asking information
Packet In	Switch->Controller	This message contains a packet which was received at a switch , bit there were no information about the appropriate output interface
Packet Out	Controller->Switch	Controller, finds the appropriate interface of a packet in message, and forwards it back to the switch with an output interface
Flow Mod	Controller->Switch	Instructs a switch to add/delete/modify a flow
Flow Expired	Switch->Controller	Informs the controller that a flow has expired after a period of time.

2.2.4 Επικοινωνία Controller-Switch



Εικόνα 5 : Επικοινωνία Controller-Switch

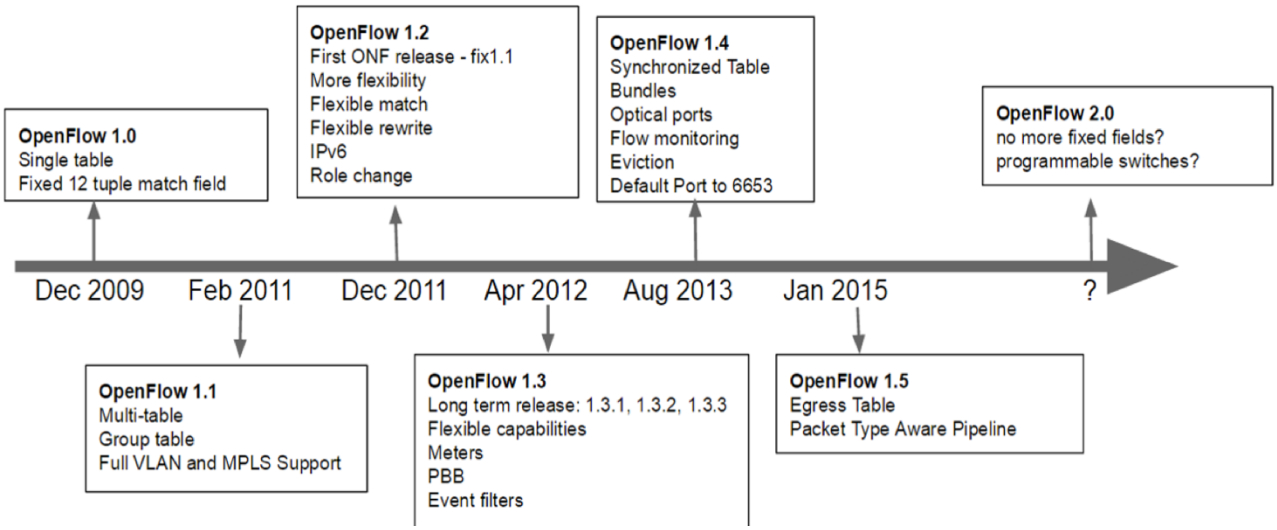
Επεξήγηση της ποιο πάνω φωτογραφίας :

Ακολουθώντας το TCP handshake, ο controller και το switch συμφωνούν στην έκδοση OpenFlow η οποία θα χρησιμοποιηθεί κατά την διάρκεια της επικοινωνίας, μέσω ton hello μηνυμάτων. Στην συνέχεια ο Controller ζητά από το switch τα διαθέσιμα ports του , αλλά και άλλα χαρακτηριστικά του. Το switch απαντά , και στη συνεχεία ο Controller μετά από επεξεργασία των πληροφοριών ρυθμίζει στο switch ορισμένες λειτουργίες όπως το packet fragmentation / communication with controller properties . Από αυτή την στιγμή , η επικοινωνία έχει ξεκινήσει . Στην πορεία ο Controller μπορεί μέσω Multipart request,να ζητήσει πληροφορίες για υφιστάμενες ροές , ή να τις επεξεργαστεί ακόμη και να εγκαταστήσει καινούργιες με την χρίση του Flow Mod. Το switch, είναι πλέον σε θέση να προωθεί πακέτα τα οποία δεν ξέρει πως να χειριστεί , στον Controller για επεξεργασία μέσω του packet _in μηνύματος

2.2.5 Versions of OpenFlow

Version	Major Features	Major Achievements
1.0	Only 1 Flow table , Fixed 12-tuple match field	1) Switch to single Controller communication
1.1	Multiple Flow tables , Group Table , Vlan/MPLS support	1) Avoid flow entry Explosion. 2) Opportunity to create sets of actions and group of flows
1.2	OXM Match , Support for Multiple controllers	1) A more scalable architecture
1.3	Meter Table , Table Miss Entry	1) Support of Qos / DiffServ 2) A more flexible system
1.4	Synchronized Table , Bundle	1) More scalable tables 2) Better synchronization with the switches
1.5	Egress Table , Schedule bundle	1) Processing functions to the output ports

		2) Better synchronization with the switches
2.0 **(In development)**	(Probably) No more Fixed fields	1) (Probably) Programmable switches



Εικόνα 6 : OF timeline

Κεφάλαιο 3

Εξέλιξη , Πλεονεκτήματα και Μειονεκτήματα SDN.

3.1 Εξέλιξη SDN και Ενσωμάτωση με Cloud Computing & NFVs

3.1.1 Ιστορική αναδρομή

Η ιστορία του SDN ξεκινά 20 χρόνια πριν , όταν το διαδίκτυο έκανε δειλά-δειλά τα πρώτα του βήματα . Με αυτή την ανάπτυξη του διαδικτύου , δημιουργήθηκε η ιδέα για ένα programmable network infrastructure το οποίο θα μπορούσε να επεξεργαστεί πακέτα σε υψηλές ταχύτητες . Μπορούμε να χωρίσουμε την ιστορία του SDN σε 3 κύριες φάσεις :

1990-2000: Κατά την περίοδο αυτή , ήρθαν για πρώτη φορά οι πρώτες προγραμματιζόμενες λειτουργίες δικτύων (programmable functions in networks), φέρνοντας έτσι μεγαλύτερη ανάπτυξη .

2001-2007: Κατά την περίοδο αυτή διαχωρίζεται το Control από το Data plane. Δημιουργούνται επίσης τα πρώτα ανοικτά Πρωτόκολλα για την επικοινωνία των 2 planes, με ποιο σημαντικό από όλα το Ethane , το οποίο αναπτύχθηκε από τον Martin Casado στα πλαίσια του διδακτορικού του στο Stanford University.

2007-2011 : Κατά την περίοδο αυτή , ο διαχωρισμός Control – Data plane καθώς και πρωτοκόλλα επικοινωνίας μεταξύ τους εδραιώθηκαν στον κόσμο της πληροφορικής. Πλέον γίνονται περισσότερες έρευνες και αναπτύσσεται λογισμικό , βάση αυτής της καινούργιας φιλοσοφίας. Με βάση το Ethane, το 2011 έρχεται η πρώτη έκδοση του OpenFlow (v1.0).

3.1.2 Cloud Computing , SDN , NFVs

Κανείς δεν μπορεί να μιλήσει για την εξέλιξη του SDN , χωρίς να μιλήσει για την ανάπτυξη του Cloud Computing και των NFVs, το κάθε ένα επηρέασε σημαντικά την ανάπτυξη των άλλων. Δηλαδή αν δεν υπήρχε σημαντική άνοδος στο Cloud Computing πιθανός να μην υπήρχε τόση ανάπτυξη στα SDN δίκτυα ή το αντίστροφο (Το ίδιο ισχύει και για τα NFVs).

Cloud Computing :

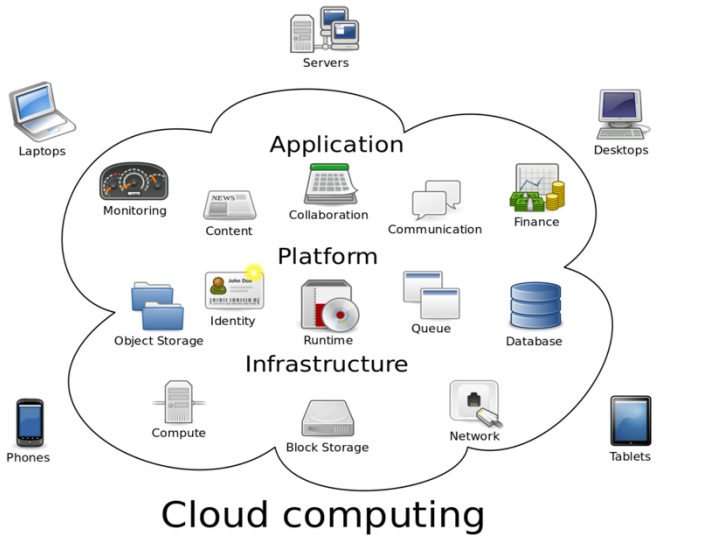
Μπορούμε να ορίσουμε cloud computing , ως την παροχή διαφόρων υπηρεσιών/πόρων μέσω του διαδικτύου . Μεταξύ άλλων , κάποιες υπηρεσίες είναι : Servers , Storage , Networks , Software , Intelligence .

Το Cloud Computing παρόλο που υπάρχει από την δεκαετία του 90 (αναπτίχθηκε για εκπαιδευτικούς σκοπούς), γνώρισε τεράστια ανάπτυξη τα τελευταία 10 χρόνια και μπορεί να διαχωριστεί σε 3 κύριες κατηγορίες:

1. Infrastructure as a Service: Είναι η ποιο βασική κατηγορία , και περιλαμβάνει την ενοικίαση απομακρυσμένου IT infrastructure ,όπως Virtual machines , Servers , Operating Systems , Networks etc.

2. Platform as a Service: Περιλαμβάνει την παροχή κάποιου περιβάλλοντος για την ανάπτυξη , την εξέλιξη , τον έλεγχο και γενικότερα την διαχείριση εφαρμογών λογισμικού.

3. Software as a Service: Περιλαμβάνει την παροχή διαφόρων εφαρμογών λογισμικού μέσω του διαδικτύου .

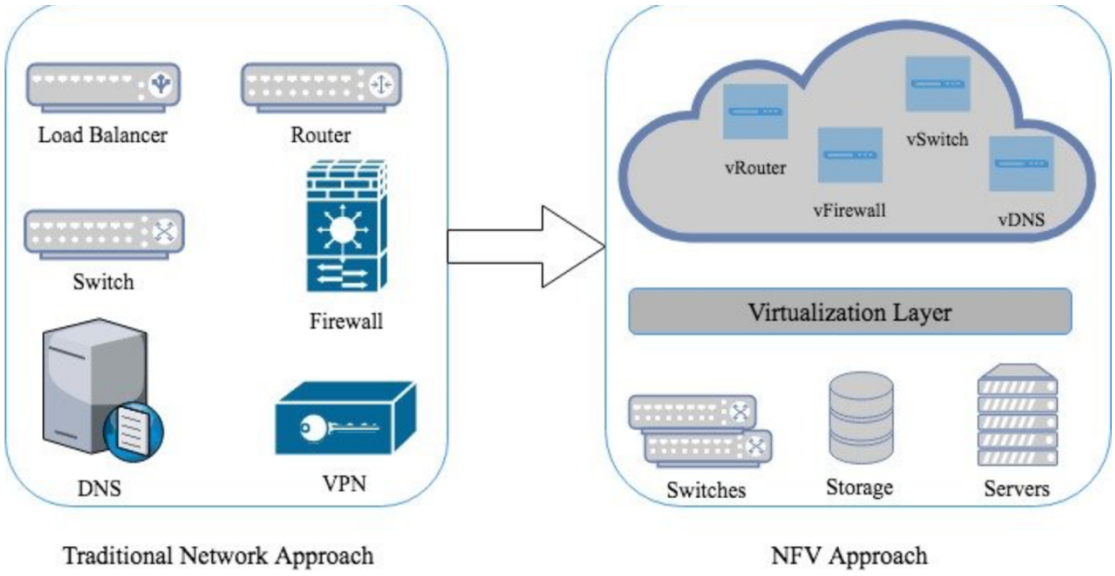


Εικόνα 7 : Cloud Computing

Network Functions Virtualization (NFVs) :

Μπορούμε να ορίσουμε ως NFV , την μετατροπή των hardware-based λειτουργιών δικτύου , όπως routing/LB/Firewall κτλ, σε λογισμικό το οποίο λειτουργεί ανεξάρτητα από το hardware στο οποίο τρέχει. Τα NFVs σαν ιδέα ήρθαν το 2012 και έκτοτε αναπτύσσονται με

γρήγορους ρυθμούς. Επιτρέπουν στους πάροχους , να φέρνουν καινούργιες υπηρεσίες και εφαρμογές στα δίκτυα τους χωρίς να χρειάζονται πλέον ακριβό εξειδικευμένο εξοπλισμό (specialized hardware equipment).



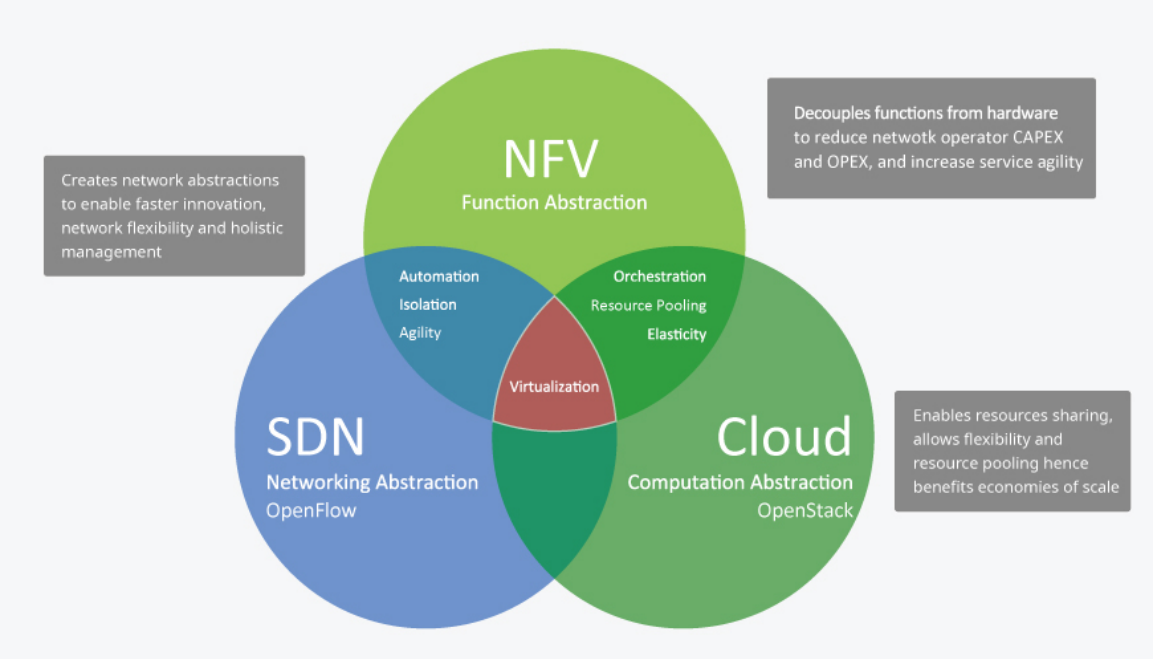
Εικόνα 8 : Προσέγγιση παραδοσιακών δικτύων (αριστερά), προσέγγιση NFV (δεξιά).

3.1.3 Συνδυασμός SDV / NFVs / Cloud Computing

Είναι σημαντικό να κατανοηθεί ότι αυτές οι 3 τεχνολογίες δεν είναι εξαρτημένες η μια από την άλλη, όμως λόγο των πολλών δυνατοτήτων που προσφέρουν όταν συνδυάζονται , τείνουν να χρησιμοποιούνται σχεδόν πάντα μαζί. Ξεκινώντας με τα NFVs και το SDN , μπορούμε να δούμε ότι βασίζονται σε παρόμοια αρχιτεκτονική. Το SDN διαχωρίζει το Control Plane από το υποκείμενο hardware (underlying hardware), όπως παρομοίως και τα NFVs διαχωρίζουν διάφορες λειτουργίες δικτύου από το υποκείμενο υλικό. Μπορούμε με ασφάλεια να πούμε ότι και ο ίδιος ο controller των SDN δικτιών είναι ένα NFV αφού εκτελεί λειτουργίες routing / firewall κλπ. σε επίπεδο λογισμικού. Επίσης λόγο της παρόμοιας αρχιτεκτονικής διάφορα NFVs μπορούν να εγκατασταθούν με ευκολία στα SDN δίκτυα , σε όλα τα μηχανήματα αφού είναι ανεξάρτητες του υποκείμενου υλικού , προσφέροντας τους μεγαλύτερες δυνατότητες , περισσότερες λειτουργίες αλλά επίσης και την δυνατότητα εύκολης επέκτασης (scalability). Τέλος , όλα τα NFVs (συμπεριλαμβανομένου του controller) μπορούν να διαχειρίζονται κεντρικά από διάφορα management-frameworks (ex. NFV MANO) , προσφέροντας έτσι πολύ καλύτερη διαχείριση στους φυσικούς πόρους που υπάρχουν.

Τα ίδια πλεονεκτήματα τα NFVs προσφέρουν και στο Cloud Computing. Οι πάροχοι Cloud Computing , μπορούν να παρέχουν περισσότερες λειτουργίες , σε φτηνότερες τιμές με ποιο εύκολη διαχείριση πόρων. Δηλαδή σε κάποιον server μπορεί να τρέχουν πολλά NFVs , με την διαθεσιμότητα πόρων για κάθε NFV να αλλάζει δυναμικά ανάλογα με τις ανάγκες του χριστή. Παρόμοια , τα Cloud Computing δίκτυα , γίνονται ποιο ευκολά επεκτάσιμα με την απλή εγκατάσταση καινούργιων λειτουργιών σαν NFVs (new switches , new functionalities , new routers etc.).

Τέλος τα SDN δίκτυα , μπορούν να βοηθήσουν το Cloud Computing σε πολύ μεγάλο βαθμό. Ας αναλογιστούμε την φράση του David Linthicum (chief cloud strategy consultant of Cisco) : «Όποιος ελέγχει το SDN δίκτυο του, μπορεί να ελέγχει το Cloud δίκτυο του, από μια μηχανή και μια όψη», την οποία έγραψε σε 1 άρθρο του μιλώντας για τα SDN δίκτυα. Όπως προαναφέραμε, τα SDN δίκτυα μπορούν να διαχειριστούν με μεγαλύτερη ευκολία από τον ίδιο τον χριστή, μέσω των διαφόρων APIs που προσφέρουν. Άρα οι πάροχοι μπορούν να διαχειριστούν τα ίδια τα δίκτυα ανάλογα με τις ανάγκες τους. Επιπρόσθετα, πολλά άλλα χαρακτηριστικά των SDN δικτύων όπως η ευελιξία τους, ο κεντρικός έλεγχος , η εύκολη προσαρμωγή και η ταχύτητα τους , τα κάνει ακόμη ιδανικότερα για ενσωμάτωση στο Cloud Computing.



Εικόνα 9 : Cloud Computing , NFV , SDN Overlapping

3.2 Πλεονεκτήματα και Μειονεκτήματα SDN

3.2.1 Πλεονεκτήματα SDN

Τα SDN δίκτυα έχουν πολλά πλεονεκτήματα έναντι των παραδοσιακών. Μερικά εξ 'αυτών είναι:

1. Κεντρικό σημείο ελέγχου (Centralized Control) : Η διαχείριση του δικτυού βασίζεται σε ένα κεντρικό σημείο , τον controller, ο οποίος διατηρεί σφαιρική άποψη της τοπολογίας .

2. Προγραμματιζόμενά δίκτυα (Programmable Networks) : Μέσω των διαφόρων APIs και των ανοικτών πρωτοκόλλων , νέες εφαρμογές και λειτουργίες μπορούν να προστεθούν στο δίκτυο εύκολα και γρήγορα, χωρίς κάποια ιδιαίτερη απαίτηση στο υλικό.

3. Επιτρέπουν την Καινοτομία : Αφού ο καθένας μπορεί να προγραμματίσει τα δίκτυα , υπάρχει μεγαλύτερη δυνατότητα για ανάπτυξη και καινοτομία.

4. Κόστος Υλικού : Το κόστος υλικού μειώνεται αφού δεν απαιτείται πλέον κάποιο εξειδικευμένο υλικό.

5. Κόστος Ανθρώπινου δυναμικού : Με τα πιο απλά μηχανήματα, είναι λογικό να μειώνεται και ο αριθμός ανθρώπων που είναι αναγκαίος για την συντήρηση/ρύθμιση τους.

6. Μείωση ανθρώπινου λάθους : Αποτέλεσμα του προηγούμενου, αφού μειώνεται το ανθρώπινο δυναμικό.

7. Καλύτερη διαχείριση δικτύων : Με τον κεντρικό έλεγχο η διαχείριση δικτύων μπορεί να επωφεληθεί σε μεγάλο βαθμό. Λάθη και σφάλματα προβλέπονται / εντοπίζονται ποιο έγκαιρα ενώ το OpenFlow στηρίζει υποστηρίζει απομακρυσμένη ρύθμιση / επιδιόρθωση .

8. Υψηλή διαθεσιμότητα (high availability) : Αποτέλεσμα του κεντρικού ελέγχου, και τις καλύτερης διαχείρισης .

9. Μειωμένη Πολυπλοκότητα : Με το απλό υλικό κι τον αφαιρετικό σχεδιασμό στα SDN δίκτυα η πολυπλοκότητα τους μειώνεται κατακόρυφα.

3.2.2 Μειονεκτήματα SDN

Εκτός από τα πολλά πλεονεκτήματα που προσφέρουν τα SDN, όπως είναι φυσικό και αυτά με την σειρά τους έχουν να αντιμετωπίσουν αρκετές προκλήσεις και αδυναμίες . Μερικές από τις προκλήσεις και τις αδυναμίες είναι οι εξής:

1. Ασφάλεια : Μπορεί πλέον η ασφάλεια στην επικοινωνία Switch-Controller να είναι πλέον δεδομένη, με την χρήση του OF over TLS ωστόσο η ασφάλεια του Controller δεν είναι. Μπορεί ο κεντρικός έλεγχος να προσφέρει πολλά πλεονεκτήματα , όμως την ίδια στιγμή είναι "single point of failure' δηλαδή, στην περίπτωση όπου ο Controller κτυπηθεί από κάποια κακόβουλη επίθεση, τότε ολόκληρο το δίκτυο απειλείται αφού ο ίδιος είναι το σημείο ελέγχου ολόκληρου του δικτύου. Γι' αυτό το λόγο, είναι αναγκαία η ανάπτυξη συστημάτων attack-detection και attack-migration έτσι ώστε η ασφάλεια του Controller να είναι εγγυημένη. Επίσης υπάρχει η ανησυχία για τις distributed DoS attacks, η οποίες μπορούν φορτώσουν τον Controller με τόσα πολλά πακέτα, σε βαθμό που δεν θα είναι διαχειρισίμα, με αποτέλεσμα την κατάρρευση ολόκληρου του δικτύου

2. Αλλαγή ολόκληρης της υποδομής: Για να εδραιωθούν τα SDN δίκτυα απαιτείται ολόκληρη η αλλαγή της υφιστάμενης υποδομής για υποστήριξη των διαφόρων πρωτοκόλλων του SDN. Αυτό αναμένεται να έχει σχετικά μεγάλο κόστος.

3. Ανάγκη υποστήριξης από Developers: Τα εργαλεία που υπάρχουν για την διαχείριση SDN δικτύων είναι περιορισμένα. Υπάρχει η ανάγκη για την δημιουργία περισσότερων, εξειδικευμένων, ανοικτού κώδικα εργαλείων για την υποστήριξη τους.

4. Εξειδίκευση Ανθρώπινου δυναμικού: Το ανθρώπινο δυναμικό πρέπει να εξοικειωθεί με την αυτήν την καινούργια αρχιτεκτονική , καθώς επίσης και με τα διάφορα εργαλεία που προσφέρονται για τον προγραμματισμό και την διαχείριση της.

5. Αρκετά προβλήματα τα οποία δεν εντοπίστηκαν: Λόγο του ότι, τα SDN δίκτυα σαν ιδέα, είναι σχετικά καινούργια, δεν έχει γίνει ακόμη επαρκής έρευνα και πειραματισμός έτσι ώστε να εντοπιστούν άλλα πιθανά προβλήματα, τα οποία αρχικά δεν είναι εμφανής σε εμάς.

Κεφάλαιο 4

Προσέγγιση και Επίλυση Προβλήματος

4.1 Πρόβλημα

Το πρόβλημα που μελετήθηκε στα πλαίσια αυτής της διπλωματικής, ήταν η κατασκευή ενός SDN δικτύου για εσωτερική χρήση (σπίτια και μικρές επιχειρήσεις). Στα πλαίσια αυτά έπρεπε να διερευνηθεί κατά πόσο αυτή η κατασκευή είναι εύκολη , επικερδής και αποδοτική . Επίσης μελετήθηκαν τρόποι με τους οποίους πιθανόν τα SDN δίκτυα να γίνουν ακόμη καλύτερα.

4.2 Πλάνο Επίλυσης

Για την επίλυση αυτής της εργασίας , από την αρχή δημιουργήθηκε ένα πλάνο το οποίο ακολουθήθηκε πιστά. Μπορούμε να χωρίσουμε το πλάνο αυτό σε 4 κύριες φάσεις.

Πρώτη φάση : Μελέτη προβλήματος, εξοικείωση με Mininet και την Python.

Κατά την διάρκεια της πρώτης φάσης, μελετήθηκαν διάφορα projects που ασχολήθηκαν με το θέμα αυτό. Επίσης , στην φάση αυτή έγινε και η εξοικείωση μου με το εργαλείο Mininet , και την γλώσσα Python, ακολουθώντας κάποια έτοιμα tutorials και ασκήσεις οι οποίες προσφέρονται από το ίδιο το Mininet. Στην πρώτη φάση επίσης έγινε η επιλογή Controller και του OF switch που θα χρησιμοποιούνταν στην κατασκευή του δικτύου.
Σκοπός της φάσεις αυτής ήταν η καλύτερη κατανόηση του προβλήματος και να εξακριβωθούν τα εργαλεία που θα χρησιμοποιούσα για την κατασκευή του SDN δικτύου

Δεύτερη φάση : Κατασκευή δικτύου στο εργαλείο Mininet.

Κατά την διάρκεια της δεύτερης φάσης, έγινε η κατασκευή διαφόρων δικτύων στο Mininet. Η κατασκευή συμπεριλάμβανε , την ανάπτυξη ενός προγράμματος για την δημιουργία μιας τοπολογίας , καθώς και την ενσωμάτωση των εξής λειτουργιών : Mac Blocker , Load balancer, καθώς και 2 προγράμματα τα οποία αλλοιώνουν την συμπεριφορά του Controller με σκοπό την βελτίωση επίδοσης του δικτύου. Κύριος σκοπός αυτής της φάσης , η καλύτερη κατανόηση της λειτουργίας των SDN δικτύων και κατά πόσο είναι εύκολο να ενσωματωθούν σε αυτά απαραίτητές λειτουργίες για ένα σπίτι/μικρή επιχείρηση.

Τρίτη Φάση : Κατασκευή πραγματικού δικτύου.

Κατά την διάρκεια της τρίτης φάσης , κατασκευαστικέ το πραγματικό SDN δίκτυο, ενσωματώνοντας όλες τις λειτουργίες που κατασκευάστηκαν στην φάση 2. Σκοπός αυτής της φάσης να εξακριβωθεί κατά πόσο είναι εύκολο και εφικτό να κατασκευαστεί ένα SDN δίκτυο.

Τέταρτη Φάση : Διεξαγωγή Πειραμάτων και Συμπεράσματα .

Κατά την διάρκεια της τέταρτης φάσης , έγιναν έλεγχοι για την απόδοση του δικτύου μας , καθώς και για εντοπισμό πιθανών bottlenecks τα οποία να υπάρχουν και τρόποι επίλυσης τους. Σκοπός αυτής της φάσης ήταν να εξακριβωθεί η αποτελεσματικότητα του δικτύου , καθώς και των προγραμμάτων που γράφτηκαν για βελτίωση.

4.3 Περιορισμοί και Προκλήσεις

Στα πλαίσια αυτής της διπλωματικής, ήρθα αντιμέτωπος με αρκετές προκλήσεις και περιορισμούς. Μερικοί εξ ‘αυτών είναι οι παρακάτω :

1. Έλλειψη πληροφοριών και πηγών : Αν και τα SDN δεν είναι τόσο καινούργια ιδέα , πάσχουν από έλλειψη πειραματισμού. Δεν υπάρχουν αρκετές έρευνες ή οδηγοί , ιδικά για κατασκευή SDN δικτύου. Επίσης οι πλείστες έρευνες που μελετήθηκαν , δεν βοήθησαν ουσιαστικά γιατί απαιτούσαν φυσικό εξοπλισμό SDN.

2. Έλλειψη εξοπλισμού : Η έλλειψη εξοπλισμού, και ειδικότερα η έλλειψη OF Switches ήταν σημαντική πρόκληση που έπρεπε να αντιμετωπίσω .

3. Ανεπάρκεια ισχυρών εργαλείων ανοικτού κώδικα : Όλα τα εργαλεία που χρησιμοποιήθηκαν ήταν ανοικτού κώδικα , χωρίς κάποιο δυνατό Community πίσω τους για την ανάπτυξη και την βελτίωση τους , με αποτέλεσμα να ήταν σε πειραματικό στάδιο ή να μην έχουν καλή επίδοση .

4. Mininet : Το εργαλείο Mininet , όπως είναι γνωστό πάσχει από ασυνέπεια μεταξύ των ίδιων του των κλάσεων με αποτέλεσμα τον περιορισμό των δυνατοτήτων του και την αδυναμία αξιοποίησης διαφόρων έτοιμων κλάσεων.

5. Φυσικός εξοπλισμός : Ο φυσικός εξοπλισμός που είχα στη διάθεση μου ήταν περιορισμένων δυνατοτήτων .

6. Πανδημία Covid-19: Λόγο της πανδημίας του Covid-19, και του lock down το οποίο επήλθε σαν αποτέλεσμα, δεν μπορούσα να μεταβώ στο πανεπιστήμιο για την δημιουργία του SDN δικτύου και την διεξαγωγή πειραμάτων καθώς τα μηχανήματα βρίσκονταν εκεί. Έτσι, υπήρξε μια καθυστέρηση 2-3 μηνών την οποία θα μπορούσα να αξιοποιήσω στην δημιουργία περισσότερων προγραμμάτων και πειραματιστών στο SDN δίκτυο

Κεφάλαιο 5

Κατασκευή Δικτύων και Προγραμμάτων στο Mininet

5.1 Εργαλείο Mininet

5.1.1 Mininet

Το Mininet είναι ένα εργαλείο για προσομοιώσεις δικτύων . Μπορεί να προσομοιώσει μεγάλο αριθμό end-hosts , switches , routers , και links σε ένα μόνο linux kernel. Κάνει χρήση ελαφριάς μορφής εικονικοποίησης και κάνει το σύστημα να δείχνει σαν ολοκληρωμένο δίκτυο. Οι μηχανές τις οποίες προσομοιώνει , συμπεριφέρονται σαν τις κανονικές. Η συμπεριφορά ολόκληρου του δικτύου , είναι αντιπροσωπευτική ενός αντίστοιχου πραγματικού δικτύου.
Από την επίσημη σελίδα του Mininet: "*In short, Mininet's virtual hosts, switches, links, and controllers are the real thing –they are just created using software rather than hardware –and for the most part their behavior is similar to discrete hardware elements. It is usually possible to create a Mininet network that resembles a hardware network, or a hardware network that resembles a Mininet network, and to run the same binary code and applications on either platform."*

Όπως προαναφέρθηκε, η υποστήριξη του Mininet σταμάτησε το 2018, αφήνοντας το με μη λειτουργικό κώδικα, και ασυνέπειες μεταξύ των κλάσεων του, πράγμα που περιορίζει τις δυνατότητες του προγραμματιστή. Αυτό είναι ένα δυσάρεστο γεγονός, αφού υπάρχει ανάγκη για ολοκληρωμένα εργαλεία στην προσομοίωση SDN δικτύων και το Mininet είναι όλες τις προοπτικές έτσι ώστε να γίνει το καλύτερο εργαλείο στον τομέα του. Αυτό με ανησυχεί, διότι μας δείχνει ότι δεν επενδύονται οι κατάλληλοι πόροι στην ανάπτυξη και την υιοθέτηση αυτής της αρχιτεκτονικής

5.1.2 Πλεονεκτήματα Mininet

Το Mininet επιλέχτηκε για την ανάπτυξη του εικονικού δικτύου αφού είναι από τα ποιο ολοκληρωμένα , δωρεάν, εργαλεία που χρησιμοποιούνται στην ανάπτυξη SDN δικτύων, προσφέροντας πολλές επιλογές για Controllers και Switches. Μερικά από τα πλεονεκτήματα του είναι :

1) Δημιουργία δικτύων σε ελάχιστο χρονικό διάστημα. Αυτό έχει ως αποτέλεσμα την άμεση εκτέλεση, τροποποίηση και αποσφαλμάτωση.

2) Δημιουργία πολύπλοκων τοπολογιών.

3) Οι end-hosts μπορούν να τρέξουν οποιοδήποτε πρόγραμμα το οποίο υποστηρίζεται από το Linux.

4) Δυνατότητα τροποποίησης του τρόπου προωθήσεις πακέτων με την χρήση POX και OpenFlow.

5) Εύκολο ευχάριστο λογισμικό με εξαιρετικό API.

6) Χρήση απλής γλώσσας προγραμματισμού , Python .

5.1.3 Δημιουργία τοπολογιών στο Mininet

Στο Mininet υπάρχουν 3 τρόποι για να δημιουργηθεί κάποια τοπολογία δικτύου. Ο ποιο εύκολος και γρήγορος είναι να χρησιμοποιήσουμε την εντολή sudo mn -topo και κάποια παράμετρο της όπως tree , single ,linear κτλ. η οποία ορίζει την τοπολογία. Η εντολή αυτή δημιουργεί αυτοματοποιημένα ένα δίκτυο , σε ελάχιστο χρόνο.

Όπως προαναφέρθηκε το Mininet υποστηρίζει παραμετροποιήσιμες τοπολογίες με την χρήση της γλώσσας προγραμματισμού Python. Με την χρήση της κλάσης Topo, η οποία είναι η βασική κλάση που επιτρέπει την δημιουργία τοπολογιών ο χρήστης μπορεί να δημιουργήσει μια ευέλικτη τοπολογία την οποία θα μπορεί να ξαναχρησιμοποιήσει στο μέλλον όσες φορές θέλει.
Βασικές μέθοδοι για την δημιουργία τοπολογίας είναι:

1) addSwitch() : Προσθήκη switch.

2) addHost() : Προσθήκη end-host.

3) addLink() : Προσθήκη ενός συνδέσμου μεταξύ 2 συστατικών στοιχείων .

4) Mininet() : Η βασική μέθοδος που δημιουργεί το δίκτυο

5) Topo._init_(): Αρχικοποίηση τοπολογίας

6) start() : Εκκίνηση λειτουργίας του δικτύου.

7) pingAll() : Έλεγχος συνδεσιμότητας μεταξύ όλων των end-host. Χρησιμοποιεί το πρωτόκολλο ICMP.

8) stop() : Διακοπή́ λειτουργιάς του δικτυού.

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

Η προσθήκη Controller γίνεται ξεχωριστά , στην γραμμή εντολών την ώρα που δημιουργούμε την τοπολογία μας , βάζουμε την παράμετρο –controller=remote .

```
"""Custom topology example

Two directly connected switches plus a host for each switch:

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

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

from mininet.topo import Topo

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

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

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )


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

Εικόνα 10 : Παράδειγμα Custom τοπολογίας σε Python

5.2 NOX/POX Controller and its API

5.2.1 NOX/POX Controller

Ο Nox δημιουργήθηκε από την Nicira το 2008 στην γλώσσα προγραμματισμού C++, και θεωρείται ο κλασσικός Controller SDN δικτύων. Υποστηρίζει λειτουργιές ελέγχου όπως

routing , έλεγχος κίνησης πακέτων , spanning tree algorithm κλπ ,οι οποίες προσφέρονται ως εφαρμογές από το API . Ο Nox έχει την δυνατότητα να εγκαταστήσει ροές στα switches κατ' εντολή. Αυτό γίνεται ανάλογα με τα πακέτα που λαμβάνει και τα γεγονότα (events) που συμβαίνουν στο δίκτυο τα οποία συλλέγει περιοδικά . Επίσης έχει την δυνατότητα να εγκαταστήσει ροές αυτόνομα αν αυτό κριθεί αναγκαίο για μείωση πιθανής μελλοντικής καθυστέρησής.

Ο POX είναι μια εξελιγμένη έκδοση του NOX , σε γλώσσα Python. Προσφέρει ένα μεγάλο API με πολλές κλάσεις αντικειμένων και συναρτήσεων ενώ παράλληλα η συμπεριφορά του μπορεί να καθοριστεί εξολοκλήρου από τον προγραμματιστή.. Έχει κερδίσει , μεγάλο μερίδιο στον τομέα της εκπαίδευσης και της ερευνάς , για την ανάπτυξη εφαρμογών και τεχνικών πάνω στην σχεδίαση SDN τεχνολογιών .

Αποτελεί τον Controller τον οποίο θα χρησιμοποιήσουμε για τα πειράματα μας.

5.2.2 POX API

Το API του είναι τεράστιο και μας δίνει πολλές δυνατότητες σαν προγραμματιστές. Ακολουθούν 2 φωτογραφίες που δείχνουν την έκταση του POX API. Θα αναλυθούν μόνο οι συναρτήσεις οι οποίες θα χρησιμοποιηθούν στα προγράμματα για βελτίωση επίδοσης του δικτύου μας. Ποιο κάτω ακολουθούν φωτογραφίες η οποία δείχνουν το μέγεθος του POX API.

Εικόνα 11&12 : POX API

Χρήσιμες συναρτήσεις :

1. Connection.send() : στέλνει ένα OpenFlow μήνυμα από τον Controller στο δίκτυο.

2. of.ofp_action_output() : Καθορίζει ένα port στο οποίο το switch πρέπει να προωθήσει το πακέτο .Χρησιμοποιείται με μηνύματα packet_out & flow_mod. Ακολουθεί παράδειγμα χρήσης.

   Δημιουργία ενός output action το οποίο θα προωθεί το πακέτο στο port 5
   action = of_action_output(port=5)

3. of.ofp_match() : Καθορίζει μια αντιστοιχία για να μπορέσει ο Controller να εφαρμόσει κανόνες και Actions σε συγκεκριμένα πακέτα . Σημαντικά πεδία του αντικειμένου αυτού είναι dl_src (source MAC) , dl_dst (destination MAC) , in_port . Ακολουθεί παράδειγμα χρήσης.

   Δημιουργία ενός match για τα πακέτα που ήρθαν από το port 1.
   match = of.ofp_match()
   match.in_port=1

4. of_packet_out() : Ο Controller δίνει εντολή στο switch να προωθήσει ένα πακέτο. Συνδυάζεται πάντα με κάποιο action. Το πακέτο αυτό μπορεί να το δημιούργησε ο controller , ή να είναι κάποιο πακέτο που του προώθησε ένα switch καθώς δεν γνώριζε που να το προωθήσει . Χρήσιμα πεδία του αντικειμένου αυτού είναι :

   • idle_timeout : χρόνος πρώτού αφαιρεθεί η ροή αν δεν χρησιμοποιείται (προαπαιτεί εγκατάσταση ροής).
   • hard_timeout : χρόνος προτού αφαιρεθεί η ροή.
   • actions : Μια λίστα με actions που θα εφαρμοστούν στο πακέτο. (e.g. ofp_action_output).
   • priority : Προτεραιότητα του πακέτου.
   • buffer_id : Αριθμός buffer εφόσον θέλει να εφαρμόσει τα actions σε ολόκληρο buffer.

• in_port : Χρήση μόνο αν χρησιμοποιείται το buffer_id , για επιλογή κάποιου buffer σε συγκεκριμένο port.
• data : Τα περιεχόμενα του πακέτου

Ακολουθεί παράδειγμα χρήσης.

Παραλαβή ενός πακέτου packet (τύπος packet_in) στον controller , και προώθηση του στο port 5 από το switch.

```
msg=of.ofp_packet_out()
msg.data=packet_in
action = of.ofp_action_output (port = 5)
msg.actions.append(action)
self.connection.send(msg)
```

5. of_flow_mod() : Ο Controller δίνει εντολή στο switch να εγκαταστήσει , να επεξεργαστεί ή να διαγράψει κάποια ροή. Χρήσιμα πεδία του αντικειμένου αυτού είναι :

• idle_timeout : χρόνος πρώτού αφαιρεθεί η ροή αν δεν χρησιμοποιείται .
• hard_timeout : χρόνος προτού αφαιρεθεί η ροή.
• actions : Μια λίστα με actions που ια εφαρμοστούν στα πακέτα της ροής. (e.g. ofp_action_output).
• priority : Προτεραιότητα των πακέτων της ροής.
• buffer_id : Αριθμός buffer εφόσον θέλει να εφαρμόσει τα actions σε ολόκληρο buffer.
• in_port : Χρήση μόνο αν χρησιμοποιείται το buffer_id , για επιλογή κάποιου buffer σε συγκεκριμένο port.
• match: κάποιο match για καθορισμό των πακέτων που θα εφαρμοστούν τα actions.

Ακολουθεί παράδειγμα χρήσης.

Δημιουργία ροής , για προώθηση των πακέτων από το port 4 στο 10.

```
msg=of.ofp_flow_mod()
match = of.ofp_match()
match.in_port = 4
msg.match = match
action = of.ofp_action_output(port=10)
msg.actions.append(action)
self.connection.send(msg)
```

5.3 Κατασκευή δικτύων και προγραμμάτων στο Mininet

Στο Mininet δεν κατασκευαστικέ ένα μόνο δίκτυο, κατασκευάστηκαν αρκετά δίκτυα με σκοπό την διεξαγωγή πειραμάτων , την ενσωμάτωση λειτουργιών, και την εύρεση βελτιστοποιήσεων. Ποιο Συγκεκριμένα κατασκευάστηκαν:

1. Δίκτυο το οποίο ενσωματώνει Load Balancer.
2. Δίκτυο το οποίο ενσωματώνει MAC Blocker.
3. 2 προγράμματα με σκοπό την βελτίωση του προγράμματος.
4. 2 παρόμοια δίκτυα , ένα SDN και ένα παραδοσιακό με σκοπό την σύγκριση τους .

Καθώς η κατασκευή δικτύων στο Mininet είναι αυτόματη, για το υπόλοιπο του κεφαλαίου θα ασχοληθούμε κυρίως με την λειτουργικότητα και την αποτελεσματικότητα τους ενώ με την κατασκευή θα ασχοληθούμε στο κεφάλαιο 6. Η ιδέα για την κατασκευή MAC Blocker, Load Balancer και Smart Switch, προέκυψε μετά από την μελέτη του επίσημου οδηγού για το OpenFlow ο οποίος δημιουργήθηκε από την ομάδα ανάπτυξης του Mininet.

5.3.1 Πρώτο δίκτυο με Load Balancer

Στο πρώτο δίκτυο σκοπός ήταν η υποστήριξη Load Balancer . Έγινε χρήση μιας ελαφρός αλλαγμένης έκδοση (για υποστήριξη round robin) του προγράμματος ip_loadbancer.py που προσφέρεται από τον POX controller. Βασική ιδέα είναι ότι ο Controller με τις δυνατότητες που έχει και την εξουσία του πάνω στα switch μπορεί εύκολα να λειτουργήσει και σαν load balancer με το εγκαθιστά και διαγράφει ροές συνεχώς.

Αλγόριθμος load balancer (Δεδομένα: IP διευθύνσεις των servers και μια διεύθυνσης IP στην οποία θα γίνεται το load balancing) :

Ο Controller λαμβάνει κάποιο πακέτο από κάποιο switch καθώς αυτό δεν γνωρίζει που να προωθήσει. Ο Controller ελέγχει αν το πακέτο προορίζεται για την διεύθυνση την οποία εξυπηρετεί σαν load balancer. Αν όχι τότε θα συνεχίσει κανονικά την δουλειά του. Αν ναι , τότε θα εγκαταστήσει μια ροή μεταξύ του host και του πρώτου server στην λίστα του η οποία είναι ρυθμισμένη να διαγράφεται αυτόματα από το switch σε σύντομο χρονικό διάστημα. Αν λοιπόν κάποιος άλλος end-host ή ακόμη και ίδιος με την πρώτη φορά , θελήσει να επικοινωνήσει με κάποιο server , δεν θα υπάρχει ροή στο switch και έτσι θα ξανασταλεί πακέτο στο Controller . Ο Controller με την σειρά του μπορεί δημιουργήσει καινούργια ροή , μεταξύ του host και του επόμενου server στη λίστα.

Βήματα:

1. Κατασκευή δικτύου : Για την κατασκευή δικτύου χρησιμοποιούμε την εντολή
   >sudo mn -topo single , 6 –mac –arp –controller = remote
   Η εντολή αυτή κατασκευάζει ένα δίκτυο με ένα switch , και 6 hosts.

   Παράμετροι :
   --mac: αυτόματη ανάθεση MAC διευθύνσεων.
   -- arp: Γεμίζει με στατικά ARP entries του κάθε host σε κάθε άλλο host.
   --controller=remote : Σύνδεση σε κάποιο απομακρυσμένο Controller. Από την στιγμή που δεν διευκρινίσαμε το ip του controller , εξυπακούεται ότι ο Controller τρέχει στον ίδιο υπολογιστή.
   --single: Εντολή για δημιουργία τοπολογίας με 1 switch .

```
mininet@mininet-vm:~$ sudo mn --topo single,6 --mac --arp --controller=remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> xterm h1 h2 h3
```

Εικόνα 11 : Δημιουργία δικτύου

Με την κατασκευή του δικτύου οι hosts παίρνουν τα ανάλογα ip, δηλαδή

Host 1 : ip 10.0.0.1

Host 2 : ip 10.0.0.2

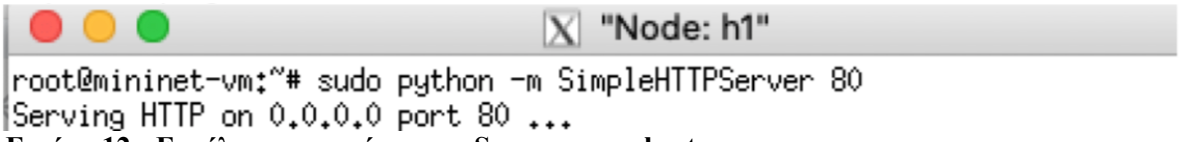
.

.

Host 6: ip 10.0.0.6

2. Κάνοντας χρήση του εργαλείου xterm μπορούμε να διαχειριστούμε τους host μας (παράθυρα terminal ανοίγουν για κάθε ένα από τους host), να τους βάλουμε να εκτελέσουν κάποιο πρόγραμμα ή να στείλουν πακέτα.
   Εκτελούμε λοιπόν την εντολή
   >xterm h1 h2 h3 h4

   Στα νέα παράθυρα του host 1& host 2 εκτελούμε το πρόγραμμα των server , το οποίο έρχεται προεγκατεστιμένο στην python .



Εικόνα 12 : Εκτέλεση προγράμματος Server στους host

3. Σε εντελώς διαφορετικό παράθυρο ξεκινούμε τον Controller μας με την εντολή

   >./pox.py log.level --DEBUG misc.ip_loadbalancer --ip=10.0.1.1

--servers=10.0.0.1,10.0.0.2

Παράμετροι :

-DEBUG: εκτύπωση μηνυμάτων στο πρόγραμμα για αποσφαλμάτωση.

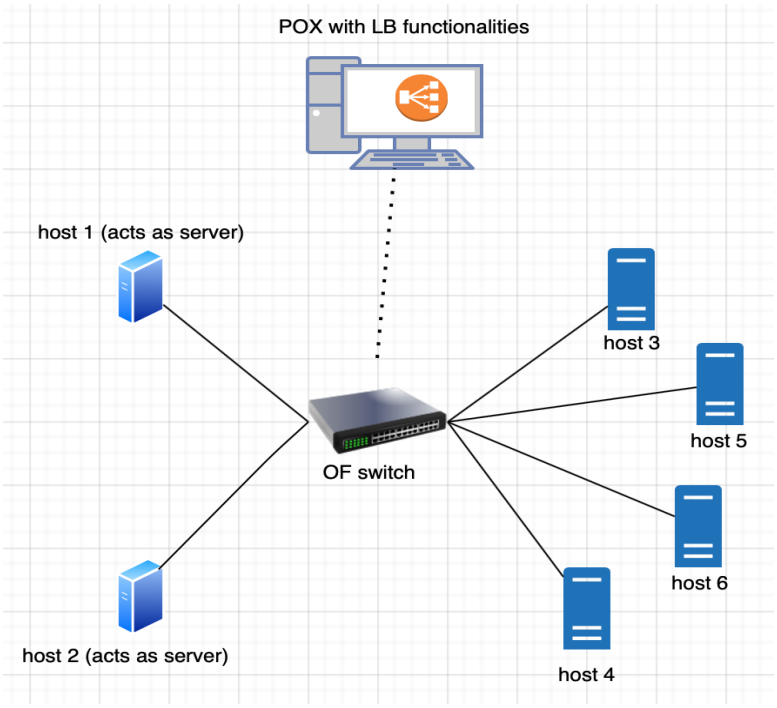
-misc.ip_loadbalancer : Το πρόγραμμα του Load Balancer.

1) ip: Η διεύθυνση που θα εφαρμόζεται Load Balancing.
2) servers : οι διευθύνσεις των servers

```
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG   misc.ip_loadbalancer --ip=10.0.1.1 --servers=10.0.0.1,10.0.0.2
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
DEBUG:core:POX 0.3.0 (dart) going up...
DEBUG:core:Running on CPython (2.7.6/Oct 26 2016 20:32:47)
DEBUG:core:Platform is Linux-4.2.0-27-generic-i686-with-Ubuntu-14.04-trusty
INFO:core:POX 0.3.0 (dart) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:iplb:IP Load Balancer Ready.
INFO:iplb:Load Balancing on [00-00-00-00-00-01 1]
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.1 up
INFO:iplb.00-00-00-00-00-01:Server 10.0.0.2 up
```

Εικόνα 13 : Εκτέλεση Controller

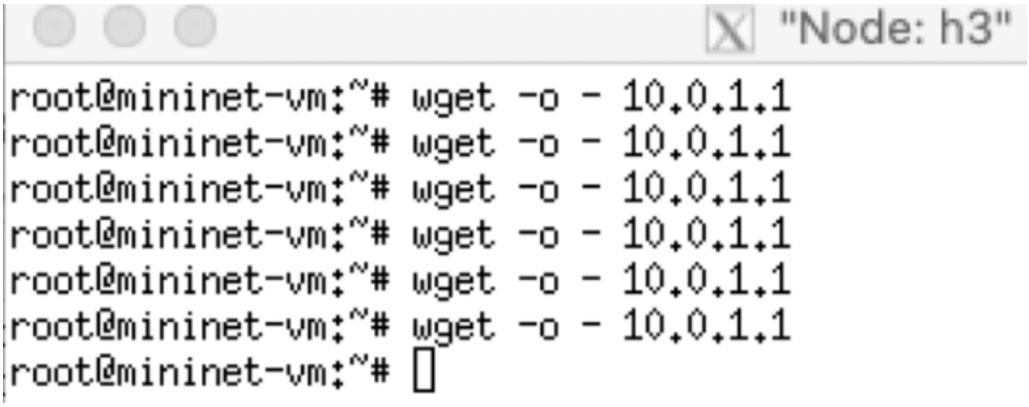
Όπως βλέπουμε στην φωτογραφία πάνω , ο Controller εντοπίζει το switch και τους servers και είναι έτοιμος να εκτελέσει την load balancing λειτουργία .Το δίκτυο αυτή μας είναι ολοκληρωμένο. Το μόνο που απομένει είναι η δημιουργία πακέτων στο δίκτυο.



Εικόνα 14 : Ολοκληρωμένο δίκτυο

4. Στα παράθυρα των host 3 και 4 , δημιουργούμε HTTP πακέτα στο δίκτυο με την εντολή (εκτελούμενη πολλές φορές)

>Wget -o – 10.0.1.1

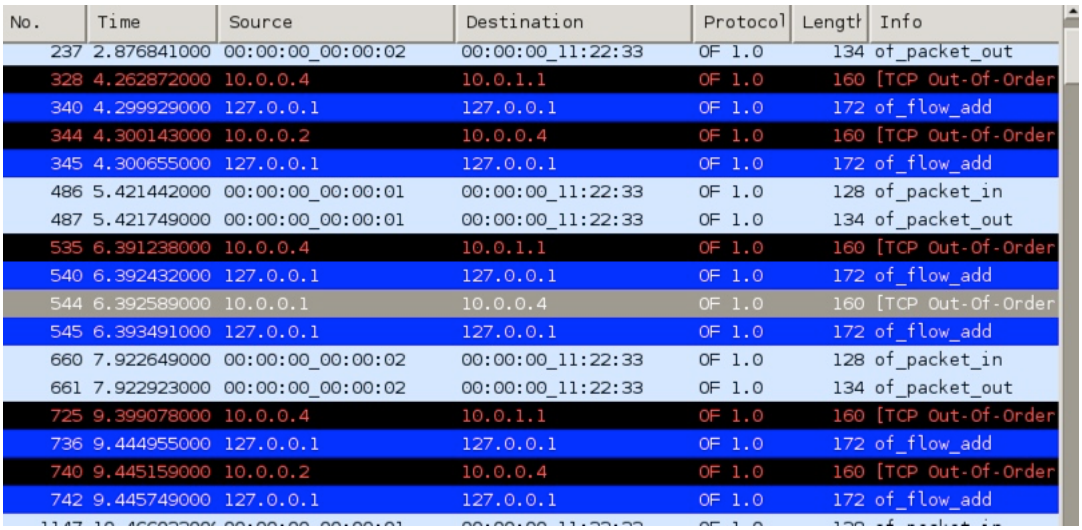


Εικόνα 15 : Εκτέλεση εντολής Wget

5. Στο παράθυρο του Controller μπορούμε να δούμε τα αποτελέσματα, και να ελέγξουμε ότι ο Load balancer μας λειτουργεί κανονικά. Όντως, παρατηρώντας την ποιο κάτω φωτογραφία ο Controller λειτουργεί ορθά και κατανέμει τα HTTP πακέτα ομοιόμορφα στους 2 servers

```
clearDEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.1
DEBUG:iplb.00-00-00-00-00-01:Directing traffic to 10.0.0.2
```

Εικόνα 16 : Αποτελέσματα Load Balancer



No.	Time	Source	Destination	Protocol	Length	Info
237	2.876841000	00:00:00_00:00:02	00:00:00_11:22:33	OF 1.0	134	of_packet_out
328	4.262872000	10.0.0.4	10.0.1.1	OF 1.0	160	[TCP Out-Of-Order
340	4.299929000	127.0.0.1	127.0.0.1	OF 1.0	172	of_flow_add
344	4.300143000	10.0.0.2	10.0.0.4	OF 1.0	160	[TCP Out-Of-Order
345	4.300655000	127.0.0.1	127.0.0.1	OF 1.0	172	of_flow_add
486	5.421442000	00:00:00_00:00:01	00:00:00_11:22:33	OF 1.0	128	of_packet_in
487	5.421749000	00:00:00_00:00:01	00:00:00_11:22:33	OF 1.0	134	of_packet_out
535	6.391238000	10.0.0.4	10.0.1.1	OF 1.0	160	[TCP Out-Of-Order
540	6.392432000	127.0.0.1	127.0.0.1	OF 1.0	172	of_flow_add
544	6.392589000	10.0.0.1	10.0.0.4	OF 1.0	160	[TCP Out-Of-Order
545	6.393491000	127.0.0.1	127.0.0.1	OF 1.0	172	of_flow_add
660	7.922649000	00:00:00_00:00:02	00:00:00_11:22:33	OF 1.0	128	of_packet_in
661	7.922923000	00:00:00_00:00:02	00:00:00_11:22:33	OF 1.0	134	of_packet_out
725	9.399078000	10.0.0.4	10.0.1.1	OF 1.0	160	[TCP Out-Of-Order
736	9.444955000	127.0.0.1	127.0.0.1	OF 1.0	172	of_flow_add
740	9.445159000	10.0.0.2	10.0.0.4	OF 1.0	160	[TCP Out-Of-Order
742	9.445749000	127.0.0.1	127.0.0.1	OF 1.0	172	of_flow_add

Εικόνα 17 : Αποτελέσματα Load Balancer σε Wireshark

Στην εικόνα 17 φαίνεται ξεκάθαρα η εναλλαγή μεταξύ των ροών. Κάθε πακέτο packet_in που λαμβάνει, προωθεί το πακέτο packet_out στο επόμενο server και αμέσως δημιουργεί ροή μεταξύ client-server. Αυτή διαγράφεται σε σύντομο χρονικό διάστημα και μια άλλη εγκαθιστάτε προς τον επόμενο server με την χρήση flow_add πακέτων.

5.3.2 Δεύτερο δίκτυο με MAC Blocker

Στο δεύτερο δίκτυο σκοπός ήταν η υποστήριξη ενός MAC Blocker. Ένας MAC blocker μπορεί να χαρακτηριστεί σαν μια απλουστευμένη μορφή Firewall με την διαφορά ότι ο MAC Blocker δεν μπορεί να εντοπίσει τις επιθέσεις , μπορεί όμως να τις αποκόψει αν αυτές είναι γνωστές , γι' αυτό συχνά δουλεύει σε συνεργασία με κάποιο IDS (intrusion detection system). Για την υλοποίηση του δημιουργήθηκε πρόγραμμα, το οποίο παρομοίως με τον Load Balancer, τρέχει στο Controller . Βασική ιδέα είναι ότι ο Controller με τις δυνατότητες που έχει και την εξουσία του πάνω στα switch μπορεί εύκολα να σταματήσει επιθέσεις από ένα έναν end-host σε κάποιον άλλον.

Αλγόριθμος MAC Blocker (Δεδομένα: γνωστές ροές που αποσκοπούν επίθεση σε κάποιο Host)

Ο Controller , γνωρίζει ποιες ροές είναι επικίνδυνες εξαρχής. Αυτές δίνονται από τον προγραμματιστή στο πρόγραμμα. Ιδανικά αυτές οι ροές εντοπίζονται από κάποιο IDS σύστημα. Έτσι ο Controller δημιουργεί κάποια matches για με σκοπό να μπορεί να εντοπίσει πακέτα που αντιστοιχούν στις τις επικίνδυνες ροές. Μετά δημιουργεί flow_mod μηνύματά και αφήνει την λίστα με τα actions κενή (Στο openflow αν η λίστα με actions κάποιου flow_mod μηνύματος είναι κενή τότε αυτομάτως εφαρμόζεται το Drop action,δηλαδή η ροή θα γίνει delete). Τέλος στέλνει αυτά τα μηνύματα στα switches, έτσι αν σε κάποιο switch , το πεδίο match του flow_mod πακέτου ταιριάξει με κάποια ροή, τότε η ροή γίνεται αμέσως Drop.
Ακόμη ο Controller ελέγχει δυναμικά για malicious ροές, δηλαδή κατά την ώρα που λαμβάνει μηνύματα packet_in έτσι ώστε να μην επιτραπεί η εγκατάσταση τέτοιου είδους ροών. Έτσι αν ο Controller , είναι από την αρχή συνδεδεμένος με όλα τα switches του δικτύου , αποκλείεται να υπάρχει κάποια malicious ροή εγκατεστημένη σε κάποιο switch.

Βήματα:

1. Κατασκευή δικτύου : Για την κατασκευή δικτύου χρησιμοποιούμε την εντολή

   > sudo python myTopo.py

   Η εντολή αυτή κατασκευάζει ένα δίκτυο με 3 switch , και 4 hosts.
   Το αρχείο myTopo.py αποτελεί ένα πρόγραμμα το οποίο κατασκευάζει μια τοπολογία που εγώ δημιούργησα .

```
mininet@mininet-vm:~/mininet/custom$ sudo python myTopo.py
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts
*** Adding switches
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Running CLI
*** Starting CLI:
```

Εικόνα 18 : Δημιουργία δικτύου

2. Σε εντελώς διαφορετικό παράθυρο ξεκινούμε τον Controller μας

>./pox.py log.level –DEBUG misc.of_tutorial misc.Myfirewall.py

```
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
DEBUG:core:POX 0.3.0 (dart) going up...
DEBUG:core:Running on CPython (2.7.6/Oct 26 2016 20:32:47)
DEBUG:core:Platform is Linux-4.2.0-27-generic-i686-with-Ubuntu-14.04-trusty
INFO:core:POX 0.3.0 (dart) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-03 1] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-03 1]
INFO:openflow.of_01:[00-00-00-00-00-02 2] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-02 2]
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 3]
```

Εικόνα 19 : Εκτέλεση Controller

Παράμετροι :

-DEBUG: εκτύπωση μηνυμάτων στο πρόγραμμα για αποσφαλμάτωση.

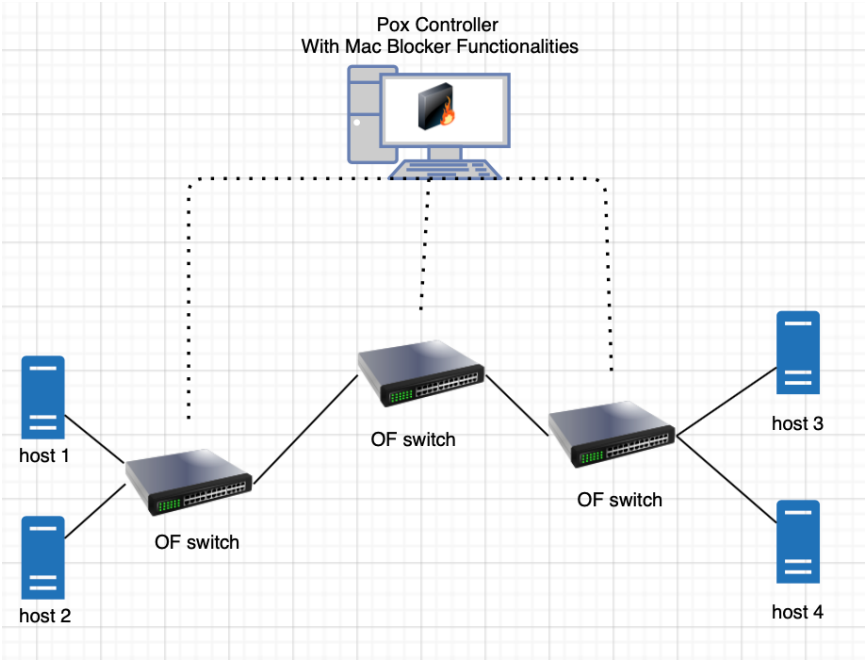
-misc.of_tutorial : Το ποιο απλό πρόγραμμα για εγκατάσταση ροών που έρχεται με τον POX.

-misc.MyFirewall : Το πρόγραμμα που δημιούργησα για προσομοίωση MAC Blocker.

Όπως φαίνεται στην εικόνα 18 , ο Controller ενώνεται με τα 3 switch και είναι έτοιμος να εκτελέσει τις λειτουργείς του σαν MAC Blocker. Εδώ πρέπει να αναφερθεί ότι στο

πρόγραμμα του MAC Blocker , προεγκατέστησα κάποιες ροές σαν malicious με σκοπό να ελεγχθεί η ορθότητα του προγράμματος. Αυτές είναι μεταξύ host1 - host3 καθώς επίσης και host2 - host3. Σε σενάριο πραγματικού κόσμου μπορούμε να πούμε ότι ο host 3 προσπαθεί να επιτεθεί στους host1 και 2. Άρα κάθε προσπάθεια επικοινωνίας μεταξύ αυτών των hosts δεν πρέπει να γίνει αποδεκτή.

Αυτή την στιγμή το δίκτυο έτοιμο για λειτουργία.



Εικόνα 20 : Το δίκτυο που δημιουργεί το myTopo.py

3. Εκτελούμε την εντολή pingall στο παράθυρο του δικτύου μας , για να δούμε την εμβέλειά επικοινωνίας κάθε host (με ποιους μπορεί να επικοινωνήσει).

mininet>pingall

Όπως παρατηρείται στην ποιο κάτω εικόνα έχουμε τα αναμενόμενα αποτελέσματα αφού ο Controller ορθά διαγράφει κάθε προσπάθεια επικοινωνίας μεταξύ host 3 – host 1 & host 3 – host2

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X h4
h2 -> h1 X h4
h3 -> X X h4
h4 -> h1 h2 h3
*** Results: 33% dropped (8/12 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X h4
h2 -> h1 X h4
h3 -> X X h4
h4 -> h1 h2 h3
```

Εικόνα 21 : Αποτελέσματα Mac Blocker

5.3.3 Πρόγραμμα Smart-Switch επιπέδου 2

Σε αυτό το κομμάτι δημιουργήθηκε ένα πρόγραμμα με σκοπό την βελτίωση απόδοσης των δικτύων μας. Το OpenFlow πρωτόκολλο δεν υποστηρίζει προγραμματιζόμενα switch ακόμα. Η ιδέα όμως είναι ότι μπορούμε να προγραμματίσουμε την συμπεριφορά τους μέσω του Controller. Αναπτύχθηκε λοιπόν ένα πρόγραμμα το οποίο κάνει τα switch του δικτύου μας έξυπνα. Τα έξυπνα switch σε αντίθεση με τα κανονικά , κρατούν ένα πίνακα στον οποίο συσχετίζουν MAC διευθύνσεις (εξού και το επίπεδο 2) με ports. Κάθε φορά που λαμβάνουν ένα πακέτο αν υπάρχει κάποια εγγραφή στον πίνακα με το dest_mac του πακέτου τότε αμέσως εγκαθιστάτε ροή, αλλιώς το πακέτο θα γίνει flood σε όλο το δίκτυο.
Χωρίς την λειτουργία αυτή, ο POX κάνει flood πακέτα σε ολόκληρο το δίκτυο σπαταλώντας πόρους και χρόνο. Με την ορθή υλοποίηση smart-switch εφαρμογής , το flooding μειώνεται δραστικά.

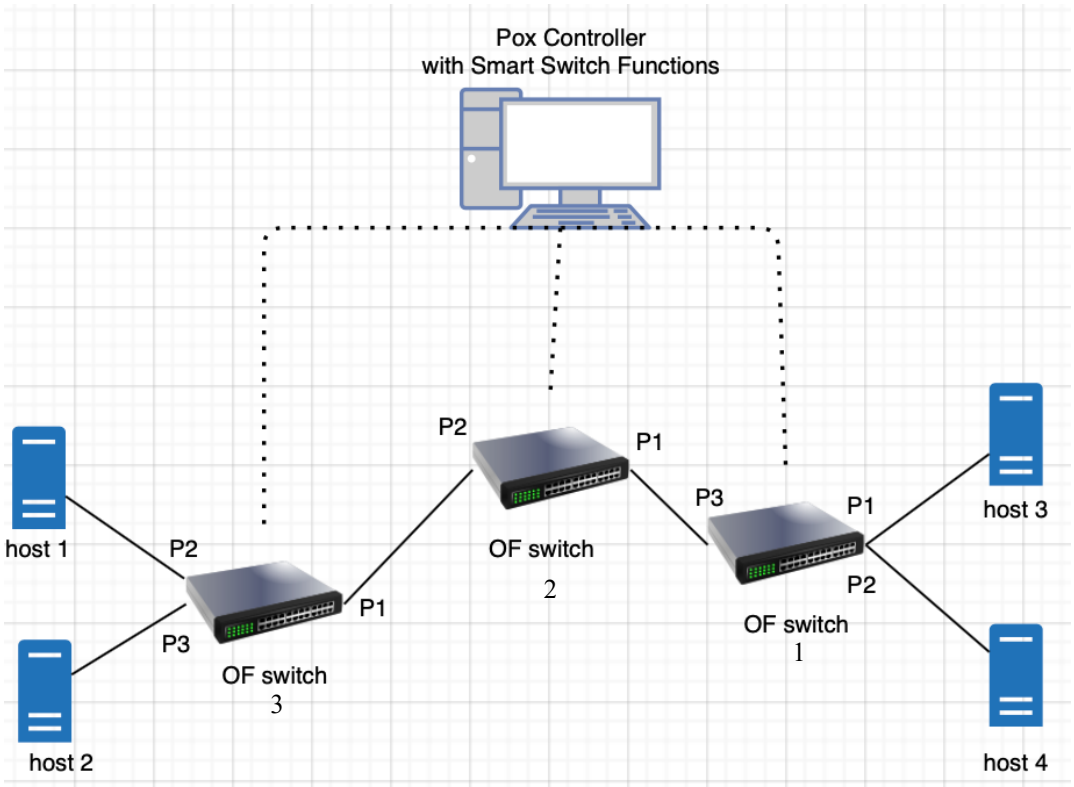
Αλγόριθμος Smart-Switch επιπέδου 2:

Έστω ότι το πακέτο pkt φτάνει σε 1 έξυπνο switch. Αν δεν υπάρχει κάποια εγγραφή στο πίνακα που να συνδέει το src_mac του πακέτου με κάποιο port , τότε δημιουργεί μια καινούρια με την αντιστοιχία src_mac – in_port. Μετέπειτα ελέγχει αν υπάρχει κάποια εγγραφή στον πίνακα που να συσχετίζει το dst_mac με κάποιο port. Αν υπάρχει , τότε δημιουργείται ένα packet_out πακέτο το οποίο ενσωματώνει το pkt και το port στο οποίο πρέπει να πάει για να φτάσει στον προορισμό του , και μετά ένα πακέτο flow_mod για δημιουργία μιας ροή μεταξύ των 2 hosts. Αν δεν υπάρχει το πακέτο γίνεται flood στο σε όλα τα ports του switch.

Θεωρητικό πείραμα για τον έλεγχο των δυνατοτήτων του Smart switch:

Έστω ότι έχουμε τοπολογία δικτύου ίδια με αυτήν της φωτογραφίας 20 , χωρίς όμως το MAC Blocker στον Controller. Στο δίκτυο δεν υπήρξε κάποια προηγούμενη επικοινωνία μεταξύ των hosts . Έτσι αρχικά οι πίνακες αντιστοίχισης των switches δεν έχουν κάποια εγγραφή μέσα. Ποιο κάτω φαίνονται οι πίνακες των switch . Στην στήλη των MAC διευθύνσεων , για ευκολία θα αναγράφεται ο Host.

Switch 1		Switch 2		Switch 3	
Mac (host).	Port	Mac (host).	Port	Mac (host).	Port



Εικόνα 22 : Το Δίκτυο με Smart Switching δυνατότητες. Όπου Px στη φωτογραφία , το port #x στο αντίστοιχο switch.

Έστω ότι την χρονική στιγμή t0 , ο host 4 προσπαθεί να επικοινωνήσει με τον host 1. Έτσι ξεκινά το πρωτόκολλο ARP. Το πρώτο πακέτο φτάνει στο πρώτο switch το οποίο , μη ξέροντας που να το προωθήσει αφού δεν υπάρχει κάποια εγκατεστημένη ροή για αυτό το πακέτο , το στέλνει στον Controller. Ο Controller με την σειρά του βλέπει ότι δεν υπάρχει εγγραφή στον πίνακα αντιστοιχίσεων για το src_mac και έτσι δημιουργεί μια καινούργια με το src_mac και το port_in του πακέτου. Στη συνέχεια , βλέποντας ότι δεν υπάρχει κάποια εγγραφή με το dst_mac του πακέτου στον πίνακα αντιστοίχισης , το στέλνει πίσω στο switch με την εντολή το πακέτο να γίνει flood στο δίκτυο. Η ίδια διαδικασία επαναλαμβάνεται και τα άλλα 2 witch μέχρι το πακέτο να φτάσει στον προορισμό του. Μόλις το πακέτο φτάσει στον προορισμό του, και ο host 4 στείλει πίσω πακέτο στον host 1, επαναλαμβάνεται η αντίστροφη διαδικασία με την διαφορά όμως ότι τώρα ο Controller γνωρίζει σε ποιο port πρέπει να προωθηθεί το πακέτο κάθε φορά. Έτσι μπορεί να εγκαταστήσει την κατάλληλη ροή σε κάθε switch για να επιτευχθεί επικοινωνία. Η όλη διαδικασία αναμένεται να χρειαστεί αρκετό χρόνο και αρκετή σπατάλη πόρων λόγο των 3 flooding που γίνονται στο δίκτυο.

Οι πίνακες αντιστοιχίας όταν έχει ήδη ξεκινήσει η επικοινωνία των host 1 και host 4 .

Switch 1

Mac (host).	Port
Host 4	2
Host 1	3

Switch 2

Mac (host).	Port
Host 4	1
Host 1	2

Switch 3

Mac (host).	Port
Host 4	1
Host 1	2

Έστω ότι την χρονική στιγμή t1 ο host 3 προσπαθεί να επικοινωνήσει με τον host 2. Η διαδικασία που ακολουθείται είναι ακριβός η ίδια με αυτή που έγινε για την επικοινωνία host 4 και host 1 . Ο χρόνος που χρειάζεται πρέπει να κυμαίνεται στα ίδια επίπεδα με τον προηγούμενο.

Οι πίνακες αντιστοιχίας όταν έχει ήδη ξεκινήσει η επικοινωνία των host 2 και host 3 .

Switch 1

Mac (host).	Port
Host 4	2
Host 1	3
Host 3	1
Host 2	3

Switch 2

Mac (host).	Port
Host 4	1
Host 1	2
Host 3	1
Host 2	2

Switch 3

Mac (host).	Port
Host 4	1
Host 1	2
Host 3	1
Host2	3

Έστω ότι την χρονική στιγμή t2 ο host 3 προσπαθεί να επικοινωνήσει με τον host 1 . Το πρώτο πακέτο φτάνει στο πρώτο switch , και αυτό αφού δεν έχει κάποια ροή που να συνδέει τον host 3 με τον host 1 το στέλνει στον. Ο Controller βλέπει στον πίνακα αντιστοιχίας του switch υπάρχει η εγγραφή Host 1<->Port 3 άρα γνωρίζει που πρέπει να προωθηθεί το πακέτο και δεν υπάρχει η ανάγκη για flooding. Έτσι αμέσως μπορεί να δημιουργήσει ένα πακέτο packet_out και να προωθήσει το πακέτο πίσω στο switch έτσι ώστε να σταλεί στο κατάλληλο port και με ένα δεύτερο flow_mod να εγκαταστήσει μια ροή στο switch για τα μελλοντικά πακέτα. Αυτό επαναλαμβάνεται και για τα άλλα 2 switches.

Άρα η επικοινωνία μεταξύ του host 3 με host 1 αναμένουμε ότι θα χρειάζεται πολύ λιγότερο χρόνο για να ξεκινήσει από αυτήν του host 3 και του host 2 ή από αυτήν του host 1 και του host 4.

Το ίδιο ισχύει και σε περίπτωση επικοινωνίας μεταξύ host 2 και host 4 σε κάποια μεταγενέστερη χρονική στιγμή.

Βήματα Για Έλεγχο Πειράματος :

1. Κατασκευή δικτύου : Για την κατασκευή δικτύου χρησιμοποιούμε την εντολή *>sudo python sudo python myTopo.py*

2. Σε εντελώς διαφορετικό παράθυρο ξεκινούμε τον Controller μας. *>./pox.py log.level –DEBUG smartSwitch.py*

   Παράμετροι :

   1) -DEBUG: εκτύπωση μηνυμάτων στο πρόγραμμα για αποσφαλμάτωση.

2) -misc.smartSwitch.py : Το πρόγραμμα για την Smart Switch λειτουργεία.

3. Δημιουργία πακέτων με την σειρά όπως στο θεωρητικό πείραμα

t0 : Host 4 -> Host 1

t1 : Host 3 -> Host 2

t2 : Host 4 -> Host 2 / Host 3 -> Host 1

Για την δημιουργία πακέτων χρησιμοποιήθηκε το εργαλείο Ping το οποίο παράγει ICMP πακέτα. Στην ποιο κάτω εικόνα φαίνονται η εκτέλεση και τα αποτελέσματα

```
*** Running CLI
*** Starting CLI:
mininet> h4 ping -c5 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=110 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.659 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.061 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.060 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.056 ms

--- 10.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4003ms
rtt min/avg/max/mdev = 0.056/22.254/110.437/44.092 ms
mininet> h3 ping -c5 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=90.7 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.208 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.044 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4003ms
rtt min/avg/max/mdev = 0.044/18.226/90.774/36.274 ms
mininet> h4 ping -c5 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2.48 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.042 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.046 ms

--- 10.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4002ms
rtt min/avg/max/mdev = 0.055/5.327/26.409/10.541 ms
mininet> h3 ping -c5 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.693 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.042 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.053 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.050 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.044 ms
```

Εικόνα 23 : αποτελέσματα πειράματος

Όπως παρατηρείται , έχουμε τα αναμενόμενά αποτελέσματα. Ο χρόνος που χρειάζεται να φτάσει το πρώτο πακέτο και να ξεκινήσει επικοινωνία μεταξύ των 2 άκρων είναι :

Host 4 -> Host 1 : 110ms , floods needed: 3

Host 3 -> Host 2 : 90.7ms , floods needed: 3

Host 4 -> Host 1 : 2.48ms , floods needed: 0

Host 3 -> Host 1: 0.69msc, floods needed 0

Το πρόγραμμα μας λειτουργεί ορθά και επιβεβαιώνει το θεωρητικό αφού μειώνει τον χρόνο που χρειάζεται για να εγκατασταθεί μια ροή μέχρι 100 φορές . Επίσης ο αριθμός των floods που χρειάζονται μειώνεται στο μισό σε σχέση με κάποιο όχι έξυπνο switch αφού σε εκείνη την περίπτωση θα χρειάζονταν 3 floods για κάθε ροή.

5.3.4 Πρόγραμμα για βελτίωση TCP Connection set Up Time

Σε αυτό το κομμάτι δημιουργήθηκε ένα πρόγραμμα με σκοπό την βελτίωση απόδοσης των δικτύων μας και ποιο συγκεκριμένα τις TCP συνδέσεις. Το πρόγραμμα αφορά το συγκεκριμένο πρωτόκολλο καθώς είναι το ποιο ευρέως χρησιμοποιημένο πρωτόκολλο στον ιστό. Αποτελεί επέκταση του προηγούμενου προγράμματος δηλαδή του Smart Switch. Η ιδέα είναι η εκμετάλλευση της ιδιότητας του TCP (transmission control protocol) πρωτοκόλλου ως αξιόπιστο , για να βελτιώσουμε τον χρόνο που χρειάζεται για να εγκατασταθεί μια TCP ροή στα switch. Όταν ο Controller λάβει ένα πακέτο packet_in από κάποιο switch , πριν να δημηγορήσει ένα flow_mod πακέτο για να το στείλει , δημιουργεί ένα πακέτο τύπου packet_out και το στέλνει στο switch. Το packet_out πακέτο είναι το πακέτο που του έστειλε πριν το switch συμπεριλαμβανομένου και κάποιου output action έτσι ώστε να προωθηθεί στο τελικό του προορισμό όσο ποιο γρήγορα γίνεται. Αυτό όμως καθυστερεί σημαντικά την εγκατάσταση της ροής. Το πρόγραμμα μας , προσπαθεί να πετύχει βελτίωση στην εγκατάσταση της ροής , με το να μην στείλει ποτέ το πακέτο packet_in πίσω στο δίκτυο. Το πακέτο θα ξανασταλεί από το TCP όταν τελειώσει ένας προκαθορισμένος χρόνος και δεν υπήρξε επιβεβαίωση παραλαβής από τον παραλήπτη .

Αλγόριθμος Smart-Switch επιπέδου 2 με βελτιστοποίηση στις TCP ροές:

Έστω ότι το πακέτο pkt φτάνει σε 1 έξυπνο switch. Αν δεν υπάρχει κάποια εγγραφή στο πίνακα που να συνδέει το src_mac του πακέτου με κάποιο port , τότε δημιουργεί μια καινούρια με την αντιστοιχία src_mac – in_port. Μετέπειτα ελέγχει αν υπάρχει κάποια εγγραφή στον πίνακα

που να συσχετίζει το dst_mac με κάποιο port. Αν υπάρχει , τότε ελέγχεται αν το πακέτο είναι TCP. Αν είναι το packet in γίνεται drop, και αμέσως ένα πακέτο flow_mod για δημιουργία μιας ροή μεταξύ των 2 hosts, ενώ αν το πακέτο δεν είναι TCP τότε το πρόγραμμα συμπεριφέρεται κανονικά , όπως είδαμε στο προηγούμενο πρόγραμμα.

Για τον έλεγχο του προγράμματος μας θα δημιουργήσουμε 2 ίδιες τοπολογίες (η τοπολογία δεν παίζει ρόλο) με μόνη τους διαφορά το πρόγραμμα που θα τρέχει στον Controller. Στο ένα θα τρέχει το απλό πρόγραμμα smart switch και στο άλλο θα τρέχει το πρόγραμμα που βελτιστοποιεί τον χρόνο εγκατάστασης ροής στο switch. Αναμένουμε ότι στο 2ο δίκτυο κάποια από τα πρώτα πακέτα θα χαθούν αφού ο Controller μας δεν θα τα στέλνει πίσω στο δίκτυο. Αυτό όμως δεν μας ενδιαφέρει αφού τα πακέτα θα ξανασταλούν λόγο του TCP, όπως επίσης δεν μας ενδιαφέρει το γεγονός ότι τα πακέτα θα φτάσουν με διαφορετική σειρά στον παραλήπτη αφού το TCP βάζει τα πακέτα σε παράθυρα πριν τα στείλει στο application layer. Όμως ο χρόνος ο οποίος θα χρειάζεται το πρώτο πακέτο να μεταφερθεί θα είναι σαφώς μικρότερος αφού την ώρα που θα γίνει retransmit θα υπάρχει ροή εγκατεστημένη μεταξύ των 2 άκρων. Για την δημιουργία tcp πακέτων γίνεται χρίση του εργαλείου hping3.
Στις φωτογραφίες κάτω φαίνονται τα αποτελέσματα

```
mininet> h1 hping3 -c 100 h3
HPING 10.0.0.3 (h1-eth0 10.0.0.3): NO FLAGS are set, 40 headers + 0 data bytes
len=40 ip=10.0.0.3 ttl=64 DF id=28014 sport=0 flags=RA seq=0 win=0 rtt=29.8 ms
len=40 ip=10.0.0.3 ttl=64 DF id=28150 sport=0 flags=RA seq=1 win=0 rtt=50.4 ms
len=40 ip=10.0.0.3 ttl=64 DF id=28230 sport=0 flags=RA seq=2 win=0 rtt=0.8 ms
len=40 ip=10.0.0.3 ttl=64 DF id=28400 sport=0 flags=RA seq=3 win=0 rtt=0.1 ms
len=40 ip=10.0.0.3 ttl=64 DF id=28592 sport=0 flags=RA seq=4 win=0 rtt=4.2 ms
len=40 ip=10.0.0.3 ttl=64 DF id=28665 sport=0 flags=RA seq=5 win=0 rtt=3.9 ms
len=40 ip=10.0.0.3 ttl=64 DF id=28746 sport=0 flags=RA seq=6 win=0 rtt=3.9 ms
```

Εικόνα 24 : αποτελέσματα πειράματος , Smart Switch

```
root@mininet-vm:~# hping3 -c 100 10.0.0.6
HPING 10.0.0.6 (h5-eth0 10.0.0.6): NO FLAGS are set, 40 headers + 0 data bytes
len=40 ip=10.0.0.6 ttl=64 DF id=18278 sport=0 flags=RA seq=2 win=0 rtt=2.8 ms
len=40 ip=10.0.0.6 ttl=64 DF id=18345 sport=0 flags=RA seq=3 win=0 rtt=2.3 ms
len=40 ip=10.0.0.6 ttl=64 DF id=18413 sport=0 flags=RA seq=4 win=0 rtt=1.7 ms
len=40 ip=10.0.0.6 ttl=64 DF id=18598 sport=0 flags=RA seq=5 win=0 rtt=1.4 ms
len=40 ip=10.0.0.6 ttl=64 DF id=18814 sport=0 flags=RA seq=6 win=0 rtt=1.2 ms
len=40 ip=10.0.0.6 ttl=64 DF id=18927 sport=0 flags=RA seq=7 win=0 rtt=1.0 ms
len=40 ip=10.0.0.6 ttl=64 DF id=19119 sport=0 flags=RA seq=8 win=0 rtt=0.6 ms
len=40 ip=10.0.0.6 ttl=64 DF id=19156 sport=0 flags=RA seq=9 win=0 rtt=4.2 ms
```

Εικόνα 25 : αποτελέσματα πειράματος , Smart Switch with TCP Enhancement

Όπως όλα φαίνονται, το πρόγραμμα μας λειτουργεί όπως αναμενόταν με πολύ σημαντική

μείωση στον χρόνο μεταφοράς του πρώτου πακέτου. Για να εξακριβώσουμε την ακριβής επιτάχυνση θα μελετήσουμε τα πακέτα στο Wireshark. Ορίζουμε ως χρόνο εγκατάστασης ροής , τον χρόνο από την στιγμή που ο Controller παραλαμβάνει το πρώτο packet_in πακέτο μέχρι την στιγμή που αποστέλλει το πρώτο flow_mod πακέτο. Ποιο κάτω φαίνονται τα πακέτα στο Wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
180	0.867745000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_request
181	0.894531000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_reply
231	3.072983000	10.0.0.1	10.0.0.5	OF 1.0	184	of_packet_in
237	3.114613000	127.0.0.1	127.0.0.1	OF 1.0	92	of_packet_out
246	3.114824000	10.0.0.5	10.0.0.1	OF 1.0	184	of_packet_in
248	3.115086000	127.0.0.1	127.0.0.1	OF 1.0	92	of_packet_out
257	3.151410000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
278	4.074324000	10.0.0.1	10.0.0.5	OF 1.0	184	of_packet_in
279	4.094063000	127.0.0.1	127.0.0.1	OF 1.0	92	of_packet_out
286	4.094967000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
954	8.868040000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_request

Εικόνα 26 : Wireshark packets , Smart Switch.

No.	Time	Source	Destination	Protocol	Length	Info
170	1.977431000	10.0.0.1	10.0.0.5	OF 1.0	184	of_packet_in
179	1.980884000	127.0.0.1	127.0.0.1	OF 1.0	92	of_packet_out
188	1.981137000	10.0.0.5	10.0.0.1	OF 1.0	184	of_packet_in
193	1.981839000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
221	2.976569000	10.0.0.1	10.0.0.5	OF 1.0	184	of_packet_in
222	3.013635000	127.0.0.1	127.0.0.1	OF 1.0	148	of_flow_add
513	7.096356000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_request
515	7.132057000	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_reply
1863	12.09572800(	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_request
1866	12.13140800(	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_reply
2072	17.09608900(	127.0.0.1	127.0.0.1	OF 1.0	76	of_echo_request

Εικόνα 27 : Wireshark packets , Smart Switch with TCP Enhancement

•Flow_establishment_time_SmartSwitch = 3.1514-3.0729=0.0785s

•Flow_establishment_time_SmartSwitch_TCP = 1.9818-1.9774=0.0044s

•Flow_establishment_Speedup = 0.0785/0.044=17.8409

Με το πρόγραμμά μας πετύχαμε μια επιτάχυνση 18 φορές στον χρόνο εγκατάστασης ροών.

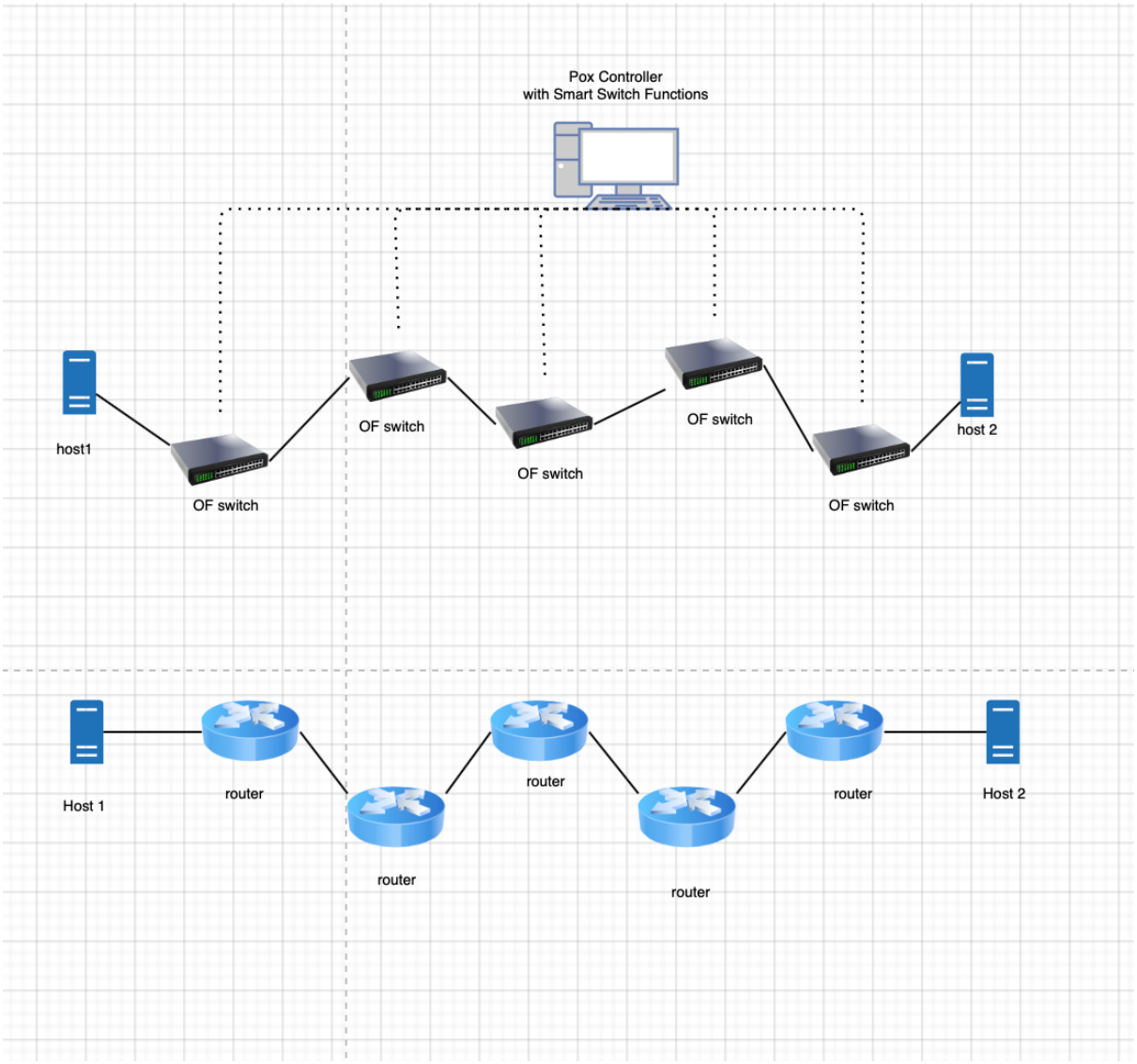
Σημαντική σημείωση :

Το εργαλείο hping3 δεν χρησιμοποιεί TCP handshake παρότι αποστέλλει TCP πακέτα. Αρχικά αυτό μου διέφυγε , όμως το αντιλήφθηκα μετά από επικοινωνία που είχα με τον επιβλέπον καθηγητή μου. Αυτό ήταν πολύ σημαντικό αφού η μη αποστολή του πακέτου packet_out ,το οποίο αποσκοπεί στην γρήγορη παραλαβή των πρώτων πακέτων από τον παραλήπτη , συνεπάγεται καθυστέρηση στην εγκατάσταση του TCP connection. Έτσι παρόλο που η ροή στα switch, δεν υπάρχει TCP επικοινωνία μεταξύ των 2 άκρων μέχρι το πρώτο πακέτο TCP να ξανασταλεί .Έτσι έγινε επανάληψη του πειράματος με HTTP πακέτα αυτή τη φορά μεταξύ ενός server και ενός client. Παρότι υπήρχε η ίδια επιτάχυνση στην εγκατάσταση των ροών , το TCP connection setup time ήταν >100 φορές ποιο αργό. Αυτό δημιούργησε μεγάλη καθυστέρηση στην επικοινωνία των 2 άκρων.

Μετά από έρευνα μου , ανακάλυψα στο RFC 1122 ,ότι το αρχικό retransmission time στα Linux είναι στα 3s. Αυτός ο χρόνος μπορεί να αλλάξει με μια ολική μεταγλώττιση του λειτουργικού συστήματος πράγμα που δεν μπόρεσα να ελέγξω ο ίδιος. Η αλλαγή του σε μικρότερο χρόνο ίσως διορθώσει και το πρόβλημα . Στην παρούσα φάση , θα θεωρήσουμε το πείραμα σαν αποτυχία. Μπορεί ωστόσο να χρησιμοποιηθεί με πρωτόκολλά σαν το UDP που δεν είναι πρόβλημα αν χαθούν λίγα πακέτα.

5.3.4 Σύγκριση χρόνου μεταφοράς πακέτων σε SDN και Traditional δίκτυα

Στο τελευταίο πείραμα στο Mininet κατασκευάστηκε ένα SDN δίκτυο , και ένα παραδοσιακό με την ίδια τοπολογία. Οι ροές στα SDN switches όπως εξηγήθηκε σε προηγούμενο κεφάλαιο, έχουν παράμετρο hard_timeout , δηλαδή σε κάποιο συγκεκριμένο χρόνο , άσχετα με το γεγονός αν χρησιμοποιούνται , θα γίνουν drop από το switch και πρέπει να επαναεγκατασταθούν. Έτσι σκοπός είναι να εξεταστεί αν αυτή η επανεγκατάσταση ροών επηρεάζει σημαντικά την απόδοση του SDN δικτύου κατά την πάροδο του χρόνου. Στο παραδοσιακό δίκτυο οι routers τρέχουν τον RIP v2 αλγόριθμο δρομολόγησης ο οποίος στέλνει τους πίνακες δρομολόγησης στους διπλανούς routers κάθε 30s. Έτσι για ποιο δίκαια σύγκριση τα switches μας ρυθμίστηκαν να κάνουν drop τις ροές κάθε 30s. Επίσης επιλέχθηκε να η δημιουργία 5 routers/switches καθώς πιστεύω είναι ο μέγιστος αριθμός που μπορεί να έχει μια μικρή επίχριση. Σύγκριση μικρότερων δικτύων έδειξε ότι δεν υπάρχει σημαντική διαφορά μεταξύ των 2. Για την κατασκευή των δικτύων , χρησιμοποιήθηκαν έτοιμες τοπολογίες που προσφέρονται στο Mininet και στο online community του. Η χρήση του RIP v2, ήταν μονόδρομος, αφού είναι ο μόνος αλγόριθμος δρομολόγησης, εκτός του static routing, ο οποίος υποστηρίζεται από το Mininet.



Εικόνα 28 : Τα 2 δίκτυα που συγκρίνουμε

Βήματα Για Σύγκριση:

1. Κατασκευή SDN δικτύου : Για την κατασκευή δικτύου χρησιμοποιούμε την εντολή *> sudo mn –topo linear,5 –mac –arp –controller=remote*

2. Σε εντελώς διαφορετικό παράθυρο ξεκινούμε τον Controller μας *>./pox.py log.level –DEBUG smartSwitch.py*

3. Δημιουργία κίνησης μεταξύ των ακρινών hosts , και καταμέτρηση του χρόνου που χρειάστηκε να φτάσουν όλα τα πακέτα στον προορισμό τους. Το εργαλείο ping χρησιμοποιείται για ακόμη μια φορά αφού εκτός από την δημιουργία πακέτων , δείχνει και τον συνολικό χρόνο.

   mininet>h1 ping -i 1 -c60 h2

Παράμετροι ping :

1) i : χρόνος που μεσολαβεί μεταξύ την αποστολή πακέτων (σε δευτερόλεπτα).
2) c : αριθμός πακέτων που θα σταλούν.

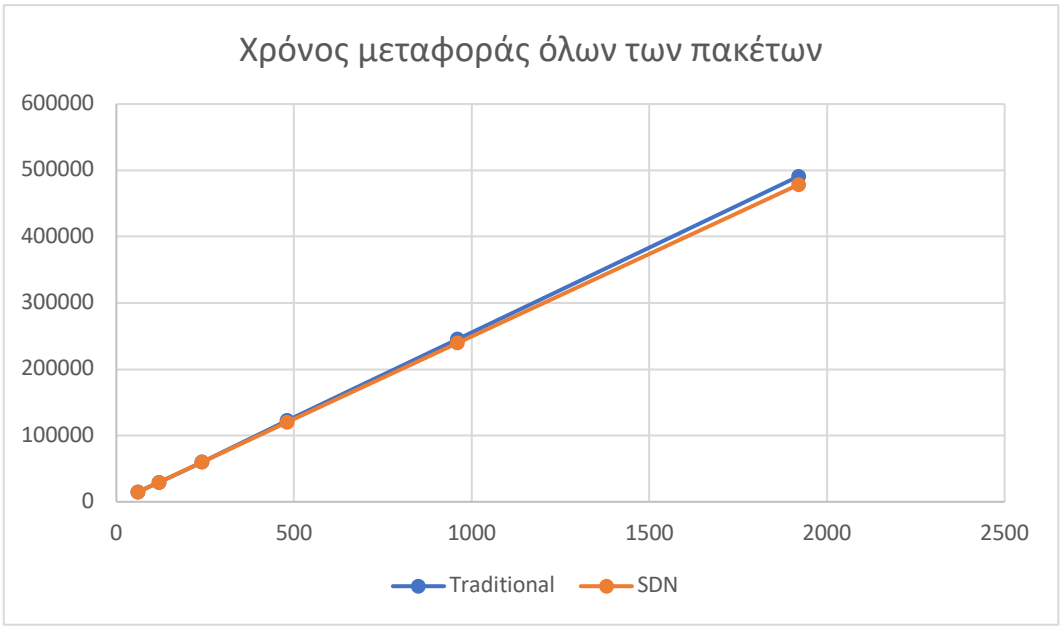
Δηλαδή ο συνδυασμός των ποιο πάνω μας εξασφαλίζει την αποστολή 60 πακέτων σε χρονικό διάστημα 1 λεπτού. Μετρήσεις έγιναν για c = 30 ,60 , 120 , 240 , 480 , 960 , 1920

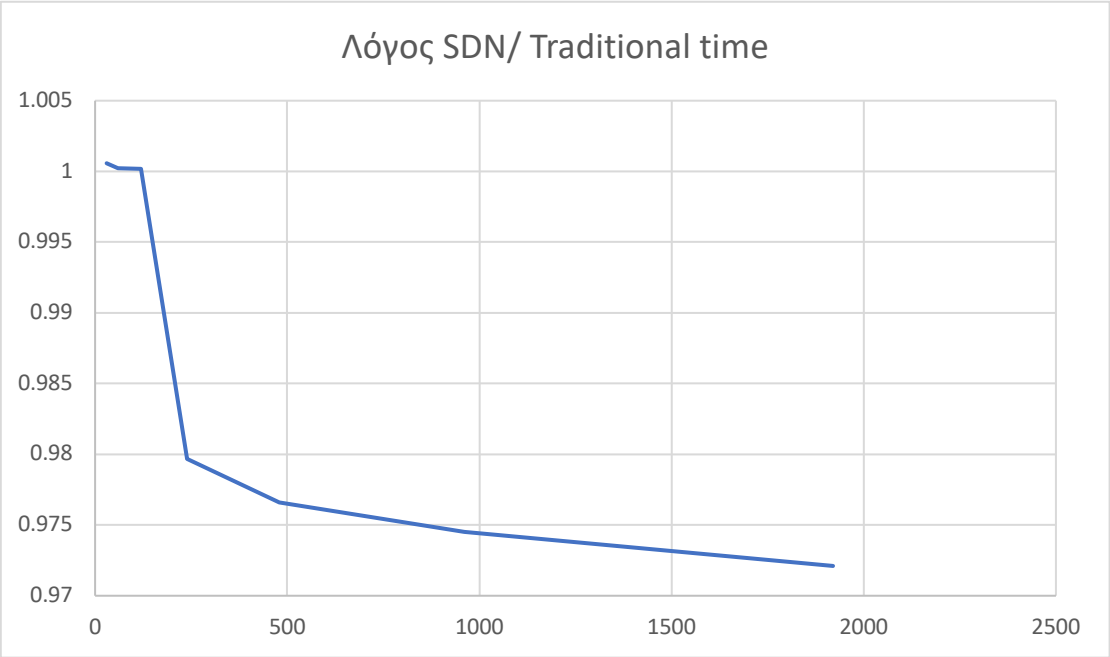
4. Κατασκευή παραδοσιακού δικτύου : Για την κατασκευή δικτύου χρησιμοποιούμε την εντολή.

>sudo python routerline.py

5. Ίδιο και απαράλλακτο με το 3ο βήμα

Στις ποιο κάτω γραφικές φαίνονται τα αποτελέσματα των μετρήσεων.





Στην πρώτη γραφική παρατηρούνται οι χρόνοι που χρειάστηκαν τα 2 δίκτυα, Παραδοσιακό και SDN για να μεταφέρουν τα πακέτα. Η δεύτερη γραφική είναι συμπληρωματική της πρώτης, αφού αυτή δεν μας δίνει ξεκάθαρα αποτελέσματα σε κάποιές περιπτώσεις, και αναπαριστά την συνάρτηση F(t) = Traditional(t) / SDN(t). Από τις γραφικές , βλέπουμε ξεκάθαρα ότι το SDN δίκτυο έχει καλύτερη απόδοση όσο περνά ο χρόνος, πράγμα που δεν περίμενα. Ξεκάθαρα βλέπουμε ότι μέχρι τα 120s το παραδοσιακό δίκτυο έχει ελάχιστα καλύτερες αλλά μετά παρατηρείτε μια σταθερή βελτίωση του SDN δικτύου .

Παρατηρήθηκε επίσης κατά την διάκια του πειράματος ότι οι χρόνοι που χρειάζονται τα πακέτα στο παραδοσιακό δίκτυο κυμαίνονται σε λίγο μεγαλύτερα όρια από αυτά του SDN

Χρονικά όρια πακέτων (SDN): 30-60ms με τα περισσότερα να χρειάζονται περίπου 45ms.

Εγκατάσταση ροής : πρώτη φορά > 90ms και μετά περίπου 1ms

Χρονικά όρια πακέτων (Traditional): 45-70ms με τα περισσότερα να χρειάζονται περίπου 55ms.

Τα όρια είναι ενδεικτικά , σε πολλές περιπτώσεις τα πακέτα χρειάζονται περισσότερο ή λιγότερο χρόνο. Παρατίθεται παράδειγμα εκτέλεσης

```
mininet> h1 ping -i  0.5 -c40 h2
*** errRun: ['stty', '-icanon', 'min', '1']
  0PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_seq=1 ttl=61 time=0.091 ms
64 bytes from 10.0.3.10: icmp_seq=2 ttl=61 time=0.041 ms
64 bytes from 10.0.3.10: icmp_seq=3 ttl=61 time=0.049 ms
64 bytes from 10.0.3.10: icmp_seq=4 ttl=61 time=0.058 ms
64 bytes from 10.0.3.10: icmp_seq=5 ttl=61 time=0.057 ms
64 bytes from 10.0.3.10: icmp_seq=6 ttl=61 time=0.054 ms
64 bytes from 10.0.3.10: icmp_seq=7 ttl=61 time=0.055 ms
64 bytes from 10.0.3.10: icmp_seq=8 ttl=61 time=0.054 ms
64 bytes from 10.0.3.10: icmp_seq=9 ttl=61 time=0.046 ms
64 bytes from 10.0.3.10: icmp_seq=10 ttl=61 time=0.055 ms
64 bytes from 10.0.3.10: icmp_seq=11 ttl=61 time=0.056 ms
64 bytes from 10.0.3.10: icmp_seq=12 ttl=61 time=0.056 ms
64 bytes from 10.0.3.10: icmp_seq=13 ttl=61 time=0.045 ms
64 bytes from 10.0.3.10: icmp_seq=14 ttl=61 time=0.055 ms
64 bytes from 10.0.3.10: icmp_seq=15 ttl=61 time=0.055 ms
64 bytes from 10.0.3.10: icmp_seq=16 ttl=61 time=0.052 ms
64 bytes from 10.0.3.10: icmp_seq=17 ttl=61 time=0.055 ms
64 bytes from 10.0.3.10: icmp_seq=18 ttl=61 time=0.072 ms
64 bytes from 10.0.3.10: icmp_seq=19 ttl=61 time=0.060 ms
```

Εικόνα 29 : Δείγμα πακέτων στο Traditional δικτύου

```
mininet> h1 ping -i  0.5 -c40 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=93.3 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.296 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.035 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.040 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.053 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.038 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.039 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.045 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.054 ms
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=0.045 ms
```

Εικόνα 30 : Δείγμα πακέτων του SDN δικτύου

Με βάση αυτά τα αποτελέσματα μπορούμε να συμπεράνουμε , ότι η εγκατάσταση ροών που γίνεται στα switch ανά τακτά χρονικά διαστήματα , δεν είναι αρκετή για να καλύψει το χάσμα που δημιουργεί η εκτέλεση του RIP v2 καθώς και τον ελαφρά χειρότερο χρόνο που χρειάζονται τα πακέτα να φτάσουν στον προορισμό τους. Αρχικά , το παραδοσιακό δίκτυο είναι καλύτερο διότι , η πρώτη εγκατάσταση ροής που χρειάζεται στο SDN δίκτυο , καλύπτει ένα μεγάλο ποσοστό του συνολικού χρόνου μεταφοράς πακέτων. Όσο τα πακέτα αυξάνονται, το αυτό ποσοστό μειώνεται , ώσπου σε κάποια φάση το SDN δίκτυο , ξεπερνά το παραδοσιακό .

Κεφάλαιο 6

Κατασκευή Πραγματικού SDN Δικτύου

6.1 Λογισμικό και Μηχανήματα/Υλικό

6.1.1 Λογισμικό

Εδώ καταγράφεται όλο το λογισμικό που χρησιμοποιήθηκε στην κατασκευή του δικτύου:

1. POX Controller : Χρησιμοποιήσαμε τον POX και στην κατασκευή του πραγματικού SDN δικτύου και όλες τις λειτουργίες τις οποίες ενσωματώσαμε κατά την διάρκεια του πειραματισμού στο Mininet.

2. OPEN vSwitch (OVS) : Το OVS , είναι ένα λογισμικό ανοικτού κώδικα το οποίο προσομοιώνει τις λειτουργίες ενός κατανεμημένου, εικονικού, πολυεπίπεδου (distributed virtual multilayer) switch. Δημιουργήθηκε με σκοπό την παροχή ενός ολοκληρωμένου switch σε εικονικά δίκτυα Η/Υ. Υποστηρίζει πολλά πρωτόκολλα , μεταξύ άλλων και OpenFlow γι' αυτό χρησιμοποιείται αρκετά συχνά στην κατασκευή SDN δικτύων. Το OVS επιλέχτηκε , γιατί όπως έδειξαν υφιστάμενες έρευνες, οι επιδώσεις του είναι εξαιρετικές. Επίσης το OVS υποστηρίζει stand-alone mode,το οποίο του επιτρέπει να λειτουργεί αυτόματα αν για κάποιο λόγο ο Controller βγει εκτός λειτουργίας.

3. CentOs : Ως λειτουργικό σύστημα των μηχανών επιλέχθηκε το CentOS 7 καθώς υποστηρίζει την εγκατάσταση όλων των απαραίτητων συστατικών στοιχείων λογισμικού .

4. VirtualBox : Το VirtualBox είναι ένας Hypervisor για x86 αρχιτεκτονική. Είναι ανοικτού κώδικα και προσφέρεται δωρεάν. Χρησιμοποιήθηκε για την δημιουργία εικονικών Hosts.

5. Mininet : Είναι το λογισμικό το οποίο τρέχει σε κάθε Virtual Machine. Είναι ουσιαστικά ένα λειτουργικό Ubuntu χωρίς γραφικό περιβάλλον . Επιλέχθηκε γιατί έρχεται με πολλά προεγκατεστιμένα χρήσιμα εργαλεία.

Συνολικό κόστος 0$

6.1.2 Μηχανήματα/Υλικό

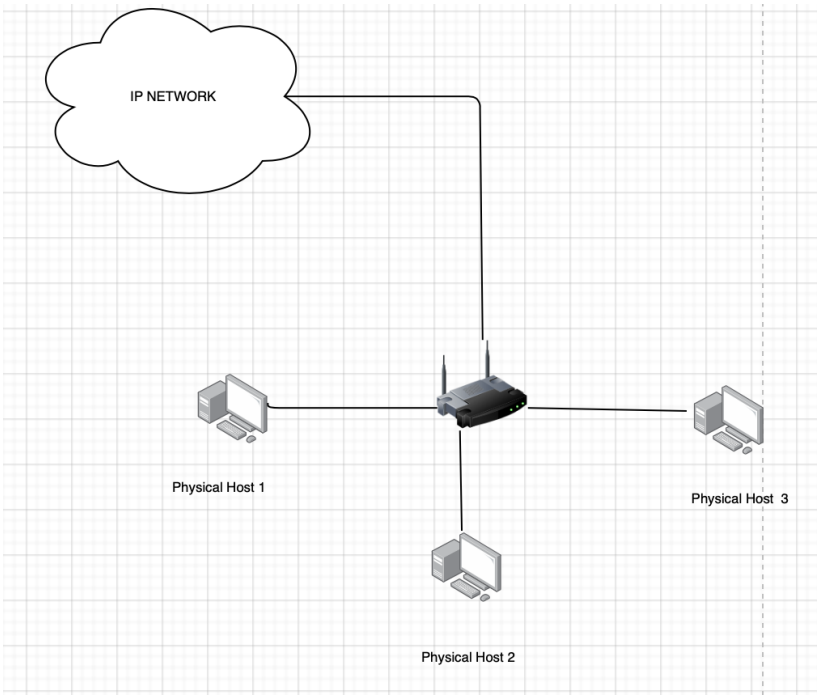
Εδώ καταγράφεται όλος ο υλικός εξοπλισμός που χρησιμοποιήθηκε στην κατασκευή του δικτύου:

1. 5 Ethernet καλώδια , τύπου Cat 5. Max speed 100mbps
2. DHCP Server : TP-LINK TL-WR841N (Wireless) Router, Max speed 300mbps
3. Physical Hosts : HP Compaq 1000 elite. Intel Core 2 Duo E8400 Processor
   3.0Ghz 6M L2 cache , 133Mhz FSB with VPro Technology .
   8GB GDDR3 SDRAM PC3-10600 (1,33-MHz) Non ECC (4 x 2GB)(SFF)(CMT)

Συνολικό κόστος εξοπλισμού : 4 x 7$ (Ethernet) + 3 x 200$ (Physical Hosts) + 20$ (DHCP Server) = 648$

6.2 Διαδικασία Κατασκευής

6.2.1 Φυσική Τοπολογία



Εικόνα 31 : Φυσική Τοπολογία δικτύου

Στην ποιο πάνω εικόνα βλέπουμε την φυσική τοπολογία του δικτύου μας. Είναι αρκετά απλή με 3 υπολογιστές ενωμένους στον Router μας , και αυτός με την σειρά του είναι συνδεδεμένος στο διαδίκτυο. Όλα είναι συνδεδεμένα με Ethernet Cat 5. Ιδανικά θα έπρεπε οι υπολογιστές να είναι συνδεδεμένοι μεταξύ τους αλλά αυτό δεν έγινε λόγο έλλειψης εξοπλισμού.

6.2.2 Ρύθμιση φυσικών Hosts

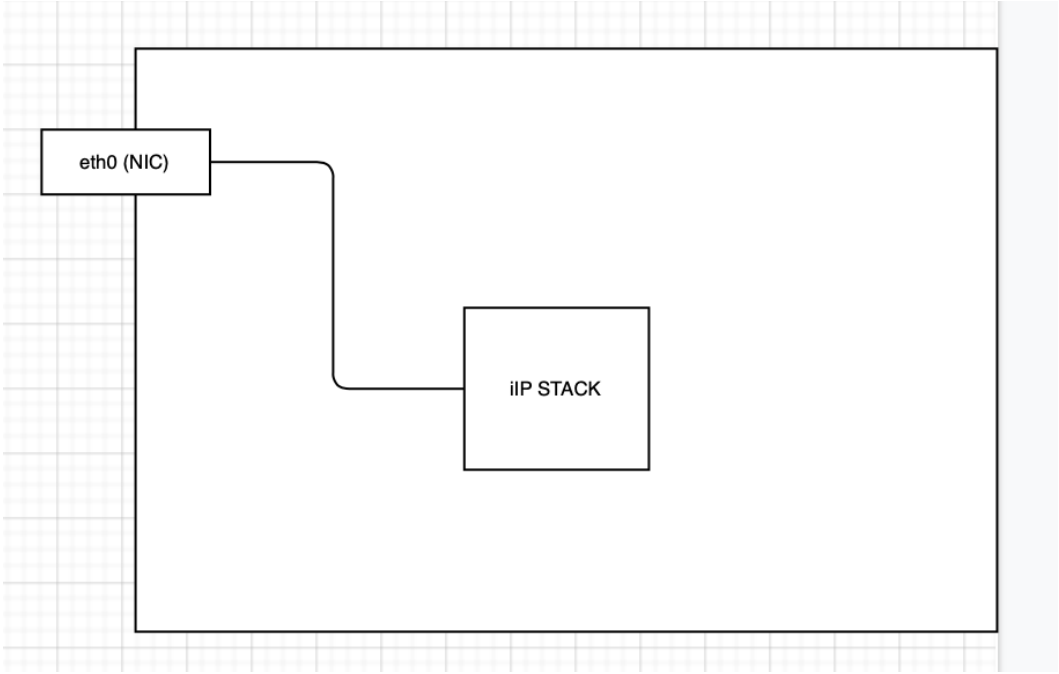
6.2.2.1 Host 3

Στον φυσικό Host 3 , δεν χρειάζεται ιδιαίτερη ρύθμιση . Είναι αυτός ο οποίος θα τρέχει το πρόγραμμα του Controller. Άρα στον εν λόγο host , το μόνο που χρειάζεται είναι η εγκατάσταση της Python και η μεταφορά των αρχείων από το Mininet στον υπολογιστή.
Οδηγίες για εγκατάσταση της python θα υπάρχουν στα παραρτήματα.

6.2.2.2 Host 1 και 2

Στους φυσικούς hosts 1 & 2 , εγκαθιστούμε το πρόγραμμα του OVS καθώς είναι αυτοί οι οποίο θα λειτουργούν σαν Switches. Επίσης σε αυτούς γίνεται η εγκατάσταση και του VirtualBox και σε αυτό δημιουργούνται 2 εικονικές μηχανές για κάθε hosts οι οποίες θα ενώνονται στα Switch μας. Οδηγίες με εγκατάσταση του OVs / VirtualBox / δημιουργία Virtual Machine θα υπάρχουν κάτω στα παραρτήματα.

Πριν ρυθμίσουμε το οτιδήποτε στους host μας ,αυτοί είναι ρυθμισμένοι έτσι ώστε να συνδέονται απευθείας στο δίκτυο μέσω του πρώτου ethernet interface (eth0) όπως και κάθε κανονικός Η/Υ. Μετά την εγκατάσταση των απαραίτητων προγραμμάτων στους υπολογιστές μας , πρέπει να ακολουθηθούν κάποια βήματα έτσι ώστε να ρυθμιστούν τα switches και η δρομολόγηση των πακέτων προς το δίκτυο να γίνεται μέσω αυτών αντί του eth0. Επίσης πρέπει να γίνουν και οι απαραίτητες ρυθμίσεις έτσι ώστε να ενωθούν και τα virtual machines στο switch.



Εικόνα 32 : Κανονική ρύθμιση Η/Υ για σύνδεση στο δίκτυο.

6.2.2.2.1 Ρύθμιση OVS

Βήματα

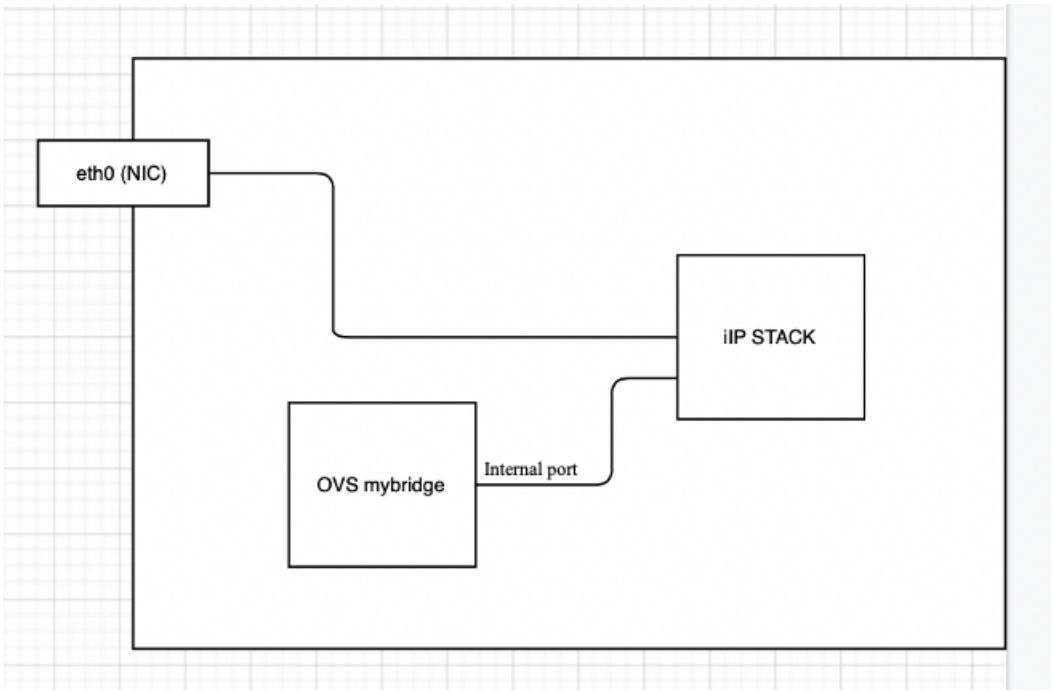
1. Εκκίνηση λειτουργίας του OVS και δημιουργία καινούργιου switch

   >sudo systemctl start openvswitch //εκκίνηση λειτουργιών OVS.

   >ovs-vsctl add-br mybridge. //δημιουργία καινούργιου switch με όνομα mybridge.

   >ifconfig mybrifge up //ενεργοποίηση του switch μας.

   Το τι πετυχαίνουμε με αυτές τις εντολές είναι η δημιουργία ενός switch το οποίο ονομάζεται mybridge και τρέχει στον υπολογιστή . Το OVS είναι συνδεδεμένο με το IP Stack του Η/Υ μέσω ενός εσωτερικού interface . Εξακολουθεί όμως να μην έχει πρόσβαση στο έξω δίκτυο αφού δεν είναι συνδεδεμένο με το eth0.Ακολουθεί φωτογραφία που δείχνει τις αλλαγές.

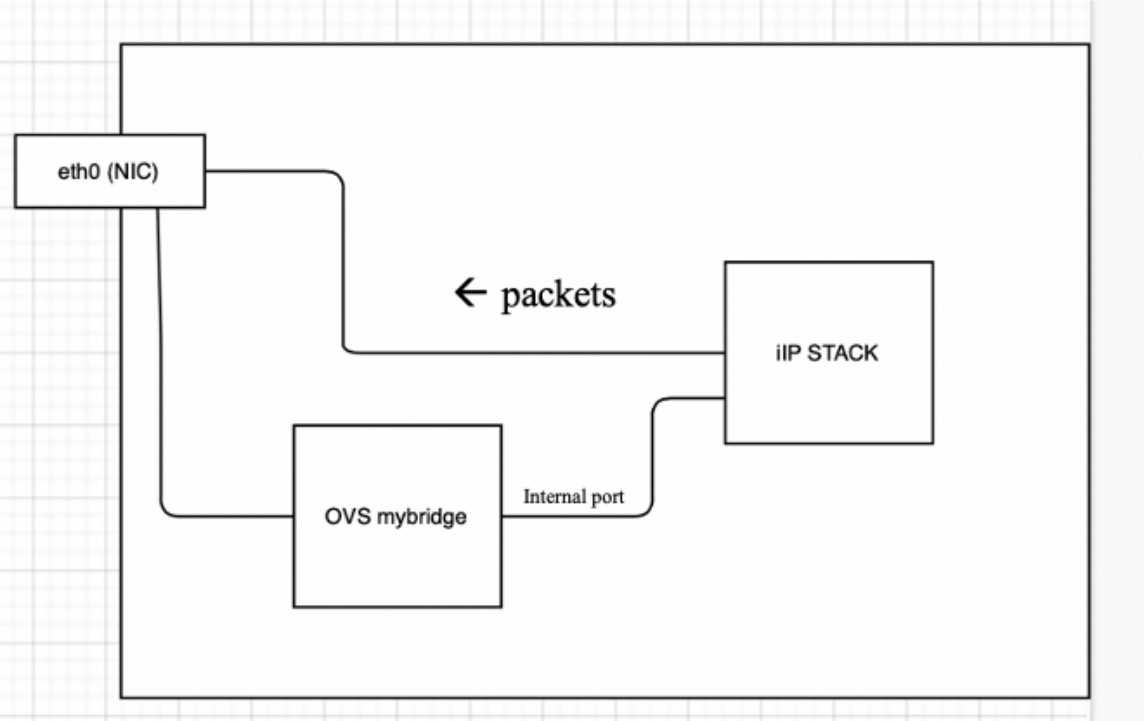


Εικόνα 33 : Αλλαγές του βήματος 1

2. Ένωση του switch μας με το eth0

   >ovs-vsctl add-port mybridge eth0 //δημιουργία ένωσης μεταξύ του ethernet 0 και του switch.

   Με την εντολή αυτή ολοκληρώθηκε το μονοπάτι που επιτρέπει του Η/Υ να συνδεθεί στο δίκτυο μέσω του OVS. Παρόλ' αυτά ο υπολογιστής είναι ακόμη ρυθμισμένος να προσπαθεί να συνδεθεί στο δίκτυο μέσω του eth0. Ακολουθεί φωτογραφία που δείχνει τις αλλαγές.



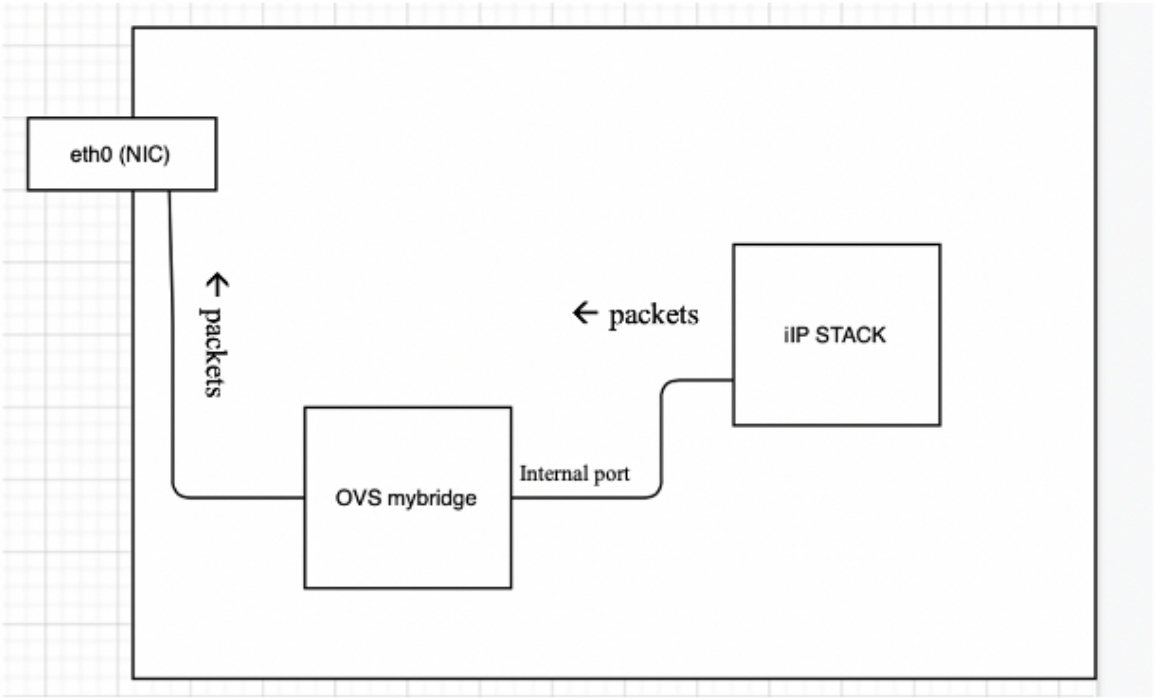
Εικόνα 34 Αλλαγές του βήματος 2

3. Αλλαγή ποριάς σύνδεσης

   >ifconfig eth0 0 *//αφαίρεση IP configuration από το eth0.*

   >sudo dhclient mybridge *//το switch μας γίνεται dhcp client και παίρνει Ip configuration.*

   Με τις 2 αυτές εντολές, οι IP λειτουργίες αφαιρέθηκαν από eth0 και ενσωματώθηκαν στο switch μας. Με αυτό τον τρόπο οποιαδήποτε επικοινωνία προς το δίκτυο, αναγκαστικά δρομολογείτε μέσω του OVs. Αυτό αποτελεί και το τελευταίο βήμα για την δημιουργία και ορθή λειτουργία του OVS στους υπολογιστές μας.
   Ακολουθεί φωτογραφία που δείχνει τι άλλαξε.



Εικόνα 35 : Αλλαγές του βήματος 3

6.2.2.3 Σύνδεση των VM στα Switch

Βήματα για σύνδεση των VM

1. Σύνδεση του switch με tap

   >ip tuntap add mode tap vport1 //δημιουργία tap που ονομάζεται vport 1 για σύνδεση του με το VM.

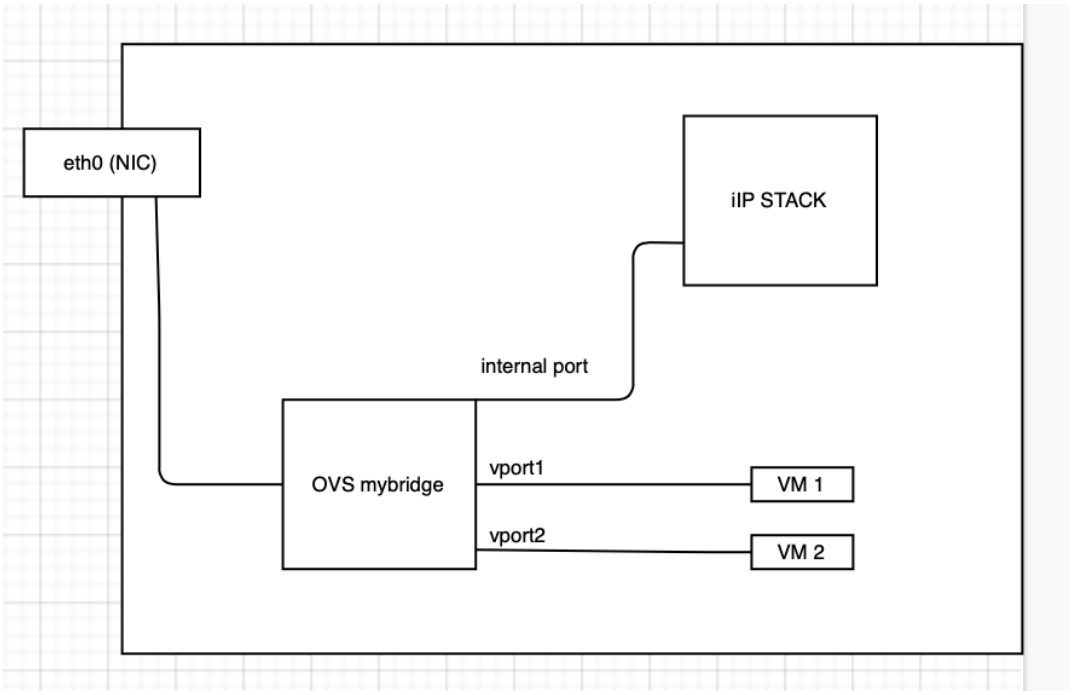
   >ifconfig vport1 //ενεργοποίηση tap

   >ovs-vsctl add-port mybridge vport1 //ένωση switch μαζί με το tap

   Στο πρώτο βήμα δημιουργούμε ένα tap που ονομάζεται vport1. Τα taps είναι interfaces τα οποία ενώνουν το switch μας με εικονικές ethernet συσκευές. Αφού το ενεργοποιήσαμε , το ενώσαμε πάνω στο switch μας

2. Σύνδεση του tap με το Virtual Machine

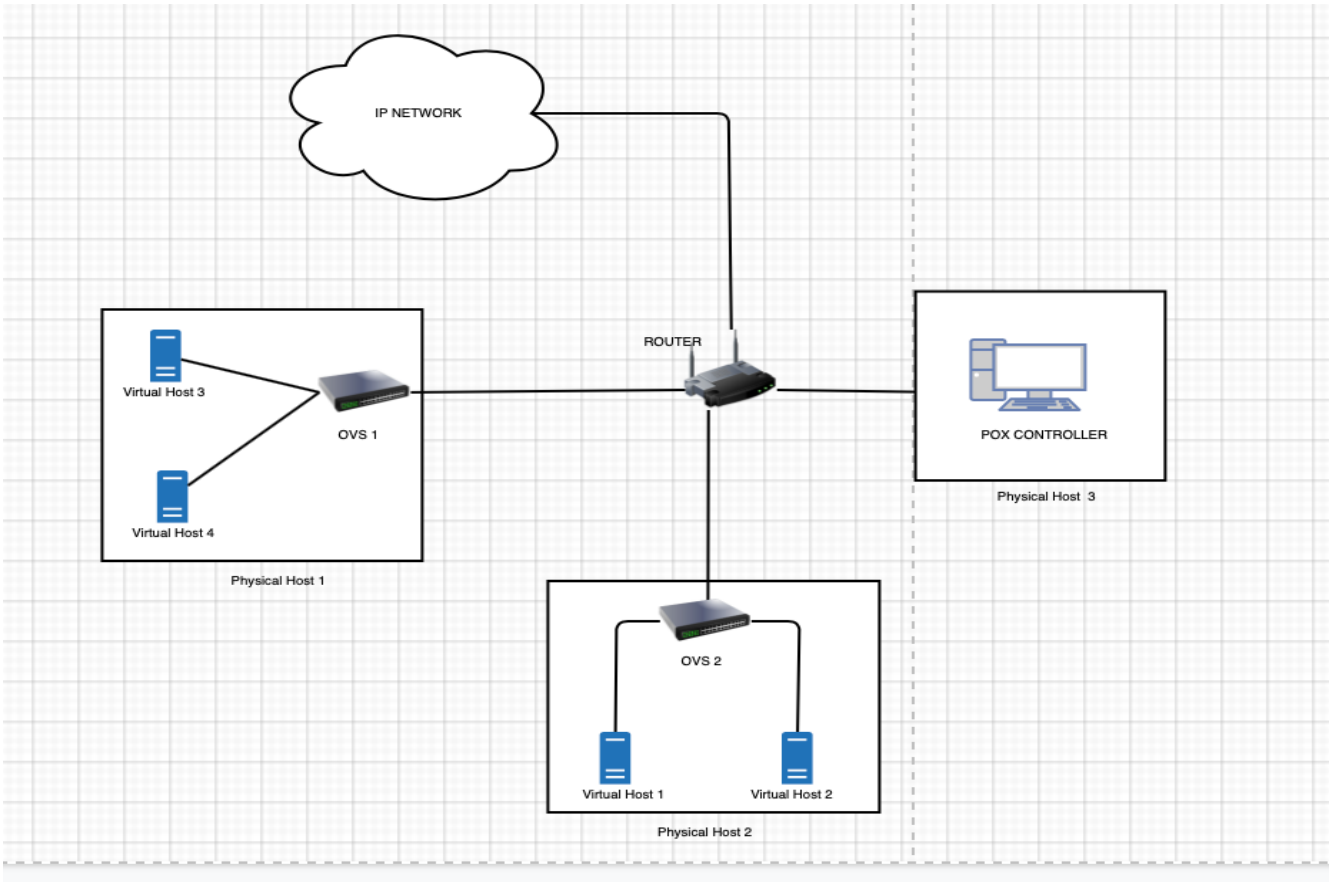
Από το VirtualBox επιλέγουμε την μηχανή που θέλουμε. Επιλέγουμε ρυθμίσεις -> ρυθμίσεις δικτύου -> adapter 1 -> το όνομα του tap που δημιουργήσαμε. Η μηχανή μας έχει πλέον συνδεθεί στο switch. Στο δεύτερο βήμα ενώνουμε το Virtual Machine μας , στο ίδιο tap που δημιουργήσαμε και ενώσαμε στο switch. Αφού κάθε host έχει 2 VM επαναλαμβάνουμε τα προηγούμενα βήματα , με την μόνη διαφορά να είναι η ονομασία του tap.Ακολουθεί φωτογραφία που να δείχνει την τελική ρύθμιση των host 1 και 2.



Εικόνα 36 : Τελική κατάσταση Host 1 και 2.

6.2.3 Ολοκληρωμένη Τοπολογία

Στην ποιο κάτω φωτογραφία μπορούμε να δούμε πως έχει διαμορφωθεί η τελική μας τοπολογία. Ο Router ο οποίος ενώνει τα 2 switch με τον Controller δεν μας επηρεάζει καθώς παραμένει άγνωστος στα SDN συστατικά στοιχεία. Όλες οι μηχανές έχουν ρυθμιστεί να λαμβάνουν στατικό IP μέσω του Router. Ιδανικά το routing θα γινόταν στατικά, όμως δεν υπάρχει η λειτουργία για τ OVS να λαμβάνουν στατικές διευθύνεις IP.



Εικόνα 37 : Ολοκληρωμένη Τοπολογία δικτύου

IP Addresses :

OVS 1 IP : 192.168.0.103

OVS 2 IP : 192.168.0.100

Virtual Host 1 IP : 192.168.0.104

Virtual Host 2 IP : 192.168.0.107

Virtual Host 3 IP : 192.168.0.109

Virtual Host4 IP : 192.168.0.111

Controller IP : 192.168.0.106

Παρά το γεγονός , ότι τα switches μας είναι φαινομενικά συνδεδεμένα με τον Controller , στην πραγματικότητα αυτό δεν ισχύει. Τα switch αγνοούν την παρουσία του Controller στο δίκτυο, αφού δεν έχει ρυθμιστεί η λογική επικοινωνία μεταξύ τους. (Η χρίση του πρωτοκόλλου DHCP στο OVS είναι αυτή που καθιστά δυνατή την λειτουργία των 2 αρχιτεκτονικών μαζί.)
Για την ρύθμιση αυτή χρησιμοποιούμε την εντολή :

>ovs-vsctl set-controller mybridge tcp:192.168.0.109

Με την εντολή αυτή , δίνουμε την IP διεύθυνση του Controller στο switch έτσι ώστε να ξεκινήσει η επικοινωνία μεταξύ τους μέσω των πακέτων OF_Hello.

6.3 Λειτουργεία Δικτιού

Αφού ολοκληρώθηκε η κατασκευή του δικτύου μας, θα ελέγξουμε την λειτουργικότητα του εγκαθιστώντας όλες τις λειτουργίες τις οποίες δημιουργήσαμε στο Mininet. Σκοπός μας να δούμε αν το SDN δίκτυο λειτουργεί όπως αναμένεται. Ακολουθούν τα βήματα που ακολουθήθηκαν. Πολλές από τις φωτογραφίες τραβήχτηκαν με χρήση φωτογραφικής μηχανής καθώς δεν ήταν δυνατή η λήψη screenshots στα VM που έτρεχαν στο CentOs.

Βήματα για έλεγχο λειτουργίας του δικτύου.

1. Έλεγχος για επικοινωνία μεταξύ των hosts και διαχείριση ροών από τον Controller Εκκινούμε τον Controller στον host 3 Και δημιουργούμε κίνηση μεταξύ των hosts με το εργαλείο ping.

```
DEBUG:misc.of_tutorial:ACTION: Out Port =  3
DEBUG:misc.of_tutorial:Learned 00:00:00:00:00:04 from Port 2!
DEBUG:misc.of_tutorial:CAM table hit, sending out packet to Port 1
DEBUG:misc.of_tutorial:Installing flow ...
DEBUG:misc.of_tutorial:MATCH: Destination MAC =  00:00:00:00:00:01
DEBUG:misc.of_tutorial:ACTION: Out Port =  1
DEBUG:misc.of_tutorial:Learned 00:00:00:00:00:04 from Port 3!
DEBUG:misc.of_tutorial:CAM table hit, sending out packet to Port 1
DEBUG:misc.of_tutorial:Installing flow ...
DEBUG:misc.of_tutorial:MATCH: Destination MAC =  00:00:00:00:00:01
DEBUG:misc.of_tutorial:ACTION: Out Port =  1
DEBUG:misc.of_tutorial:CAM table hit, sending out packet to Port 3
DEBUG:misc.of_tutorial:Installing flow ...
DEBUG:misc.of_tutorial:MATCH: Destination MAC =  00:00:00:00:00:04
DEBUG:misc.of_tutorial:ACTION: Out Port =  3
DEBUG:misc.of_tutorial:CAM table hit, sending out packet to Port 2
DEBUG:misc.of_tutorial:Installing flow ...
DEBUG:misc.of_tutorial:MATCH: Destination MAC =  00:00:00:00:00:04
DEBUG:misc.of_tutorial:ACTION: Out Port =  2
DEBUG:misc.of_tutorial:CAM table hit, sending out packet to Port 2
DEBUG:misc.of_tutorial:Installing flow ...
DEBUG:misc.of_tutorial:MATCH: Destination MAC =  00:00:00:00:00:04
DEBUG:misc.of_tutorial:ACTION: Out Port =  2
DEBUG:misc.of_tutorial:Learned 00:00:00:00:00:02 from Port 2!
DEBUG:misc.of_tutorial:CAM table hit, sending out packet to Port 3
DEBUG:misc.of_tutorial:Installing flow ...
DEBUG:misc.of_tutorial:MATCH: Destination MAC =  00:00:00:00:00:04
DEBUG:misc.of_tutorial:ACTION: Out Port =  3
DEBUG:misc.of_tutorial:Learned 00:00:00:00:00:02 from Port 1!
DEBUG:misc.of_tutorial:CAM table hit, sending out packet to Port 2
DEBUG:misc.of_tutorial:Installing flow ...
DEBUG:misc.of_tutorial:MATCH: Destination MAC =  00:00:00:00:00:04
DEBUG:misc.of_tutorial:ACTION: Out Port =  2
DEBUG:misc.of_tutorial:Learned 00:00:00:00:00:02 from Port 3!
DEBUG:misc.of_tutorial:CAM table hit, sending out packet to Port 2
DEBUG:misc.of_tutorial:Installing flow ...
DEBUG:misc.of_tutorial:MATCH: Destination MAC =  00:00:00:00:00:04
DEBUG:misc.of_tutorial:ACTION: Out Port =  2
DEBUG:misc.of_tutorial:CAM table hit, sending out packet to Port 3
```

Εικόνα 39 : Λειτουργία Controller.

```
root@mininet-vm:~/mininet/custom# ping 192.168.0.111
PING 192.168.0.111 (192.168.0.111) 56(84) bytes of data.
64 bytes from 192.168.0.111: icmp_seq=1 ttl=64 time=113 ms
64 bytes from 192.168.0.111: icmp_seq=2 ttl=64 time=0.392 ms
64 bytes from 192.168.0.111: icmp_seq=3 ttl=64 time=0.045 ms
64 bytes from 192.168.0.111: icmp_seq=4 ttl=64 time=0.047 ms
64 bytes from 192.168.0.111: icmp_seq=5 ttl=64 time=0.046 ms
64 bytes from 192.168.0.111: icmp_seq=6 ttl=64 time=0.046 ms
64 bytes from 192.168.0.111: icmp_seq=7 ttl=64 time=0.046 ms
64 bytes from 192.168.0.111: icmp_seq=8 ttl=64 time=0.045 ms
64 bytes from 192.168.0.111: icmp_seq=9 ttl=64 time=0.047 ms
64 bytes from 192.168.0.111: icmp_seq=10 ttl=64 time=0.046 ms
64 bytes from 192.168.0.111: icmp_seq=11 ttl=64 time=0.047 ms
64 bytes from 192.168.0.111: icmp_seq=12 ttl=64 time=0.044 ms
64 bytes from 192.168.0.111: icmp_seq=13 ttl=64 time=0.046 ms
64 bytes from 192.168.0.111: icmp_seq=14 ttl=64 time=0.045 ms
^C
```

Εικόνα 39 : δημιουργία κίνησης μεταξύ Virtual Host 1 και Virtual Host 4.

Ο Controller λειτουργεί κανονικά και εγκαθιστά ροές στα switch έτσι ώστε οι 2 host να μπορούν να επικοινωνήσουν μεταξύ τους.

2. Σύνδεση του προσωπικού μου υπολογιστή στον router μέσω Wi-Fi.
   Το IP που δίνεται στον υπολογιστή μου είναι 192.168.0.101 .Ρυθμίζουμε το MAC Blocker πρόγραμμα έτσι ώστε να αποκόπτει κάθε επικοινωνία από τον υπολογιστή μου και το IP:*192.169.0.102. Η διεύθυνση , είναι αυτή στην οποία ο Controller θα εκτελεί Load balancing.*

3. Στους Virtual Host 1 & 2 εκτελούμε την εντολή που φένεται ποιο κάτω για να λειτουργούν σαν HTTP servers

   >sudo python -m SimpleHTTPServer 80

4. Στον Host 3 εκτελούμε την εντολή που φαίνεται ποιο κάτω για να ξεκινήσουμε τον Controller

   >./pox.py log.level –DEBUG –misc.full_payload.py misc.loadB_Firewall –ip=192.169.0.102 --servers=192.168.0.104 , 192.168.107

   Παράμετροι :
   1) misc.full_payload: εντολή προς τα switch, να έχουν ολοκληρωμένο payload στα

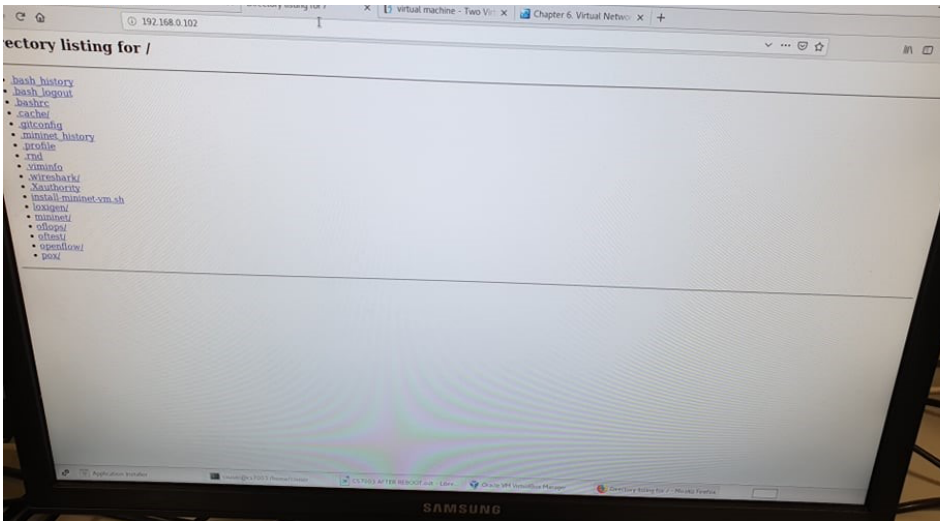
πακέτα τους όταν επικοινωνούν με τον Controller . Τα OVS μπορεί να μην αφήσουν πίσω αχρείαστες πληροφορίες για εξοικονόμηση πόρων.

2) loadB_Firewall: Είναι αρχείο που ενσωματώνει τις 2 λειτουργίες μαζί.

2a) ip/servers : ίδιες παράμετροι με το ip_loadbalancer στο Mininet.

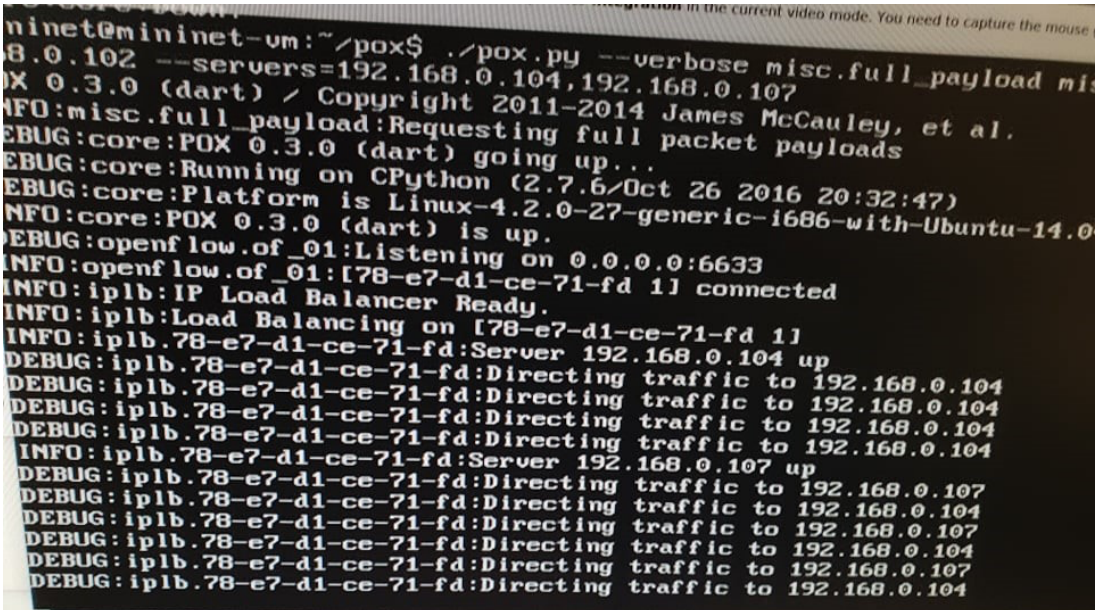
5. Αποστολή πακέτων από το virtual host 3 στους servers.

Στον Host 3 , ανοίγουμε ένα browser και πληκτρολογούμε την διεύθυνση στην οποία γίνεται το load_balancing. Αυτό θα δημιουργήσει κίνηση HTTP πακέτων μεταξύ του Host 3 και κάποιου από του servers.



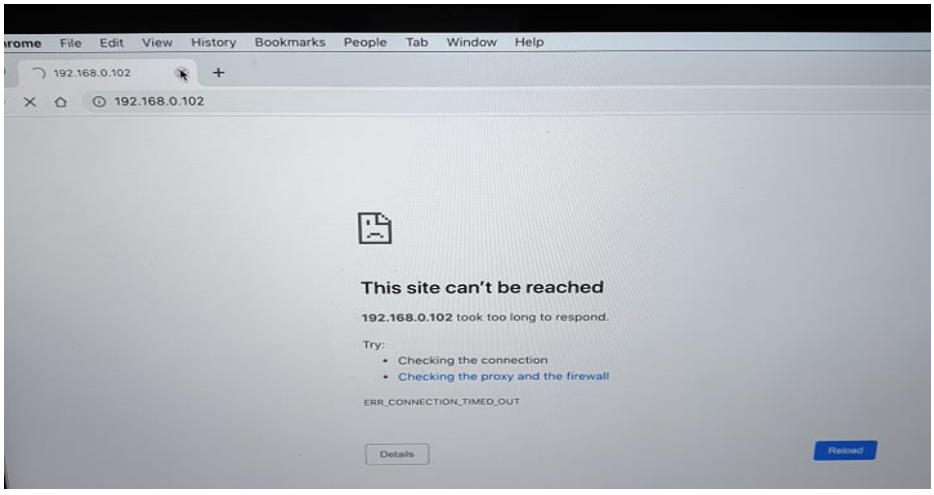
Εικόνα 38 : Η φόρτωση της σελίδας στον Host 3.

Όπως φαίνεται στην εικόνα ποιο πάνω εικόνα , η σελίδα φορτώνεται επιτυχώς. Για έλεγχο του Load Balancing function , εκτελούμε κάποια refresh στην ιστοσελίδα και παρατηρούμε την οθόνη του Controller. Ο Controller αρχικά εντοπίζει μόνο τον 1 server και αποστέλλει τα πακέτα μόνο σε αυτόν. Μόλις εντοπιστεί όμως και ο δεύτερος, ξεκινά να κατανέμει τα πακέτα ομοιόμορφα και στους 2.

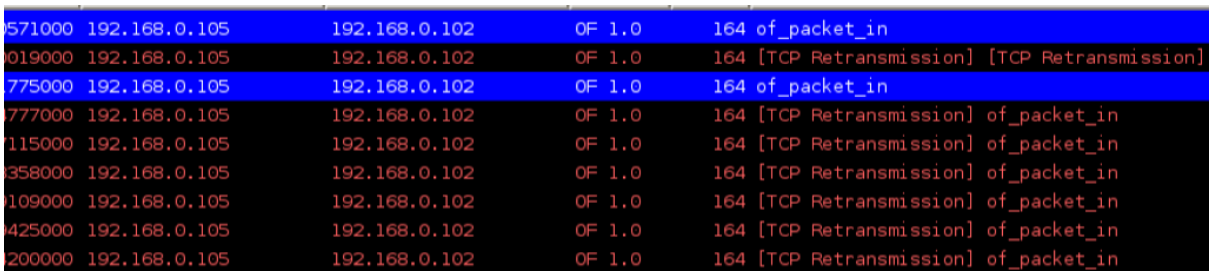


Εικόνα 39 :Αποτελέσματα Load Balancer.

6. Επανάληψη βήματος 4 αλλά αυτή τη φορά , η αποστολή πακέτων (μέσω browser) γίνεται από τον προσωπικό μου υπολογιστή για έλεγχο λειτουργίας Mac Blocker.



Εικόνα 40 :Αποτελέσματα Mac Blocker.

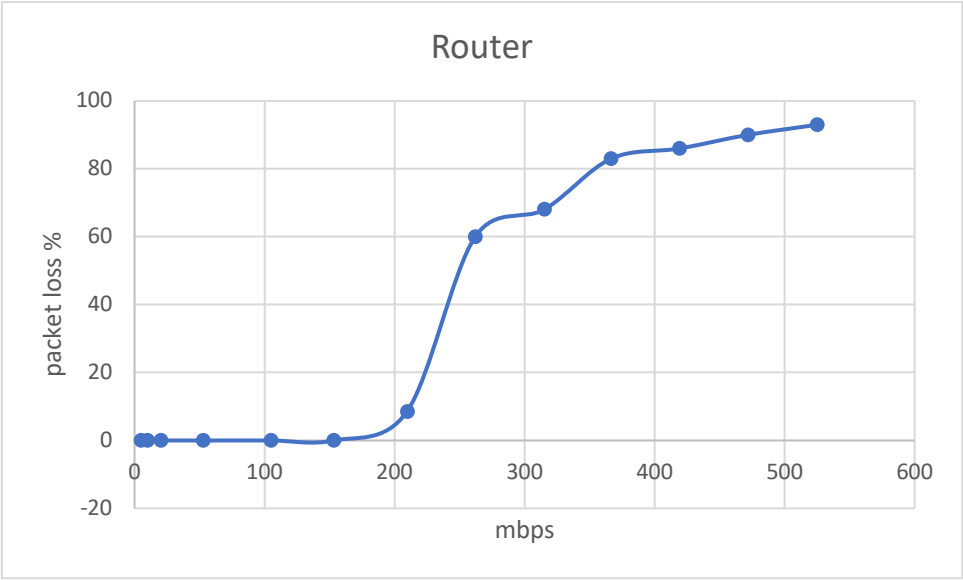


Εικόνα 41 :Αποτελέσματα Mac Blocker στο Wireshark.

Όπως φαίνεται στην εικόνα 41 το switch προωθεί τα πακέτα που έλαβε από τον υπολογιστή μου , στον Controller. Αφού αυτός τα κάνει drop , και γίνεται χρίση του TCP, γίνονται πολλά retransmission στην προσπάθεια να φτάσει το πακέτο στον προορισμό του.

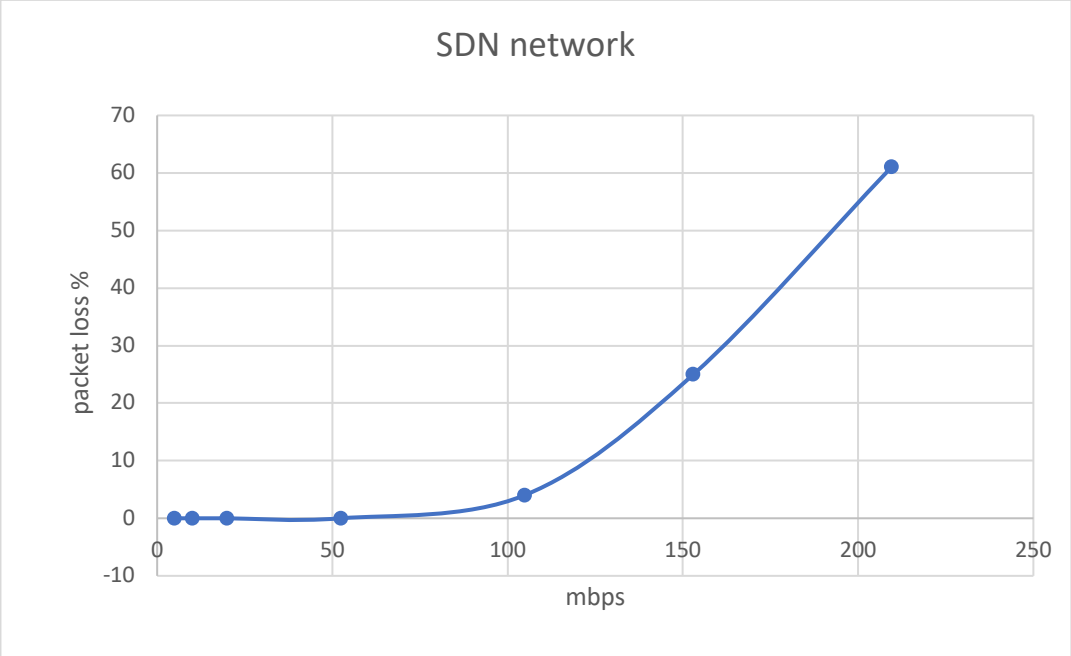
6.4 Αντοχές Δικτύου και Βελτίωση Απόδοσης

Αφού Δημιουργήσαμε το δίκτυο μας , το μόνο που απέμεινε ήταν να ελέγξουμε την αντοχή του. Σκοπός ήταν να δούμε πόσα mb/s μπορεί να διαχειριστεί το δίκτυο μας και με τι ποσοστό χαμένων πακέτων. Επειδή όμως το δίκτυο μας ήταν συνδεδεμένο σε φυσικό router, πρώτα έγιναν μετρήσεις για το πόσα mb/s μπορεί να διαχειριστεί αυτός. Στην καλύτερη περίπτωση ο router θα αποτελούσε ένα άνω φράγμα για την αντοχή του SDN δικτύου. Τα OVS switches απενεργοποιήθηκαν για την μέτρηση αυτή. Η διαδικασία που ακολουθήθηκε ήταν η δημιουργία κίνησης μεταξύ του φυσικού host 1 και 2 με το εργαλείο ping. Το εργαλείο ping έχει την δυνατότητα να στέλνει τεράστιο αριθμό mb/s. Για παράδειγμα με την εκτέλεση
>ping -s 65500 -i 0.00001 192.168.0.102 (-s = size of packet , -i time between each packet)
Στέλνονται στον χρήστη 192.168.0.102 625mbps

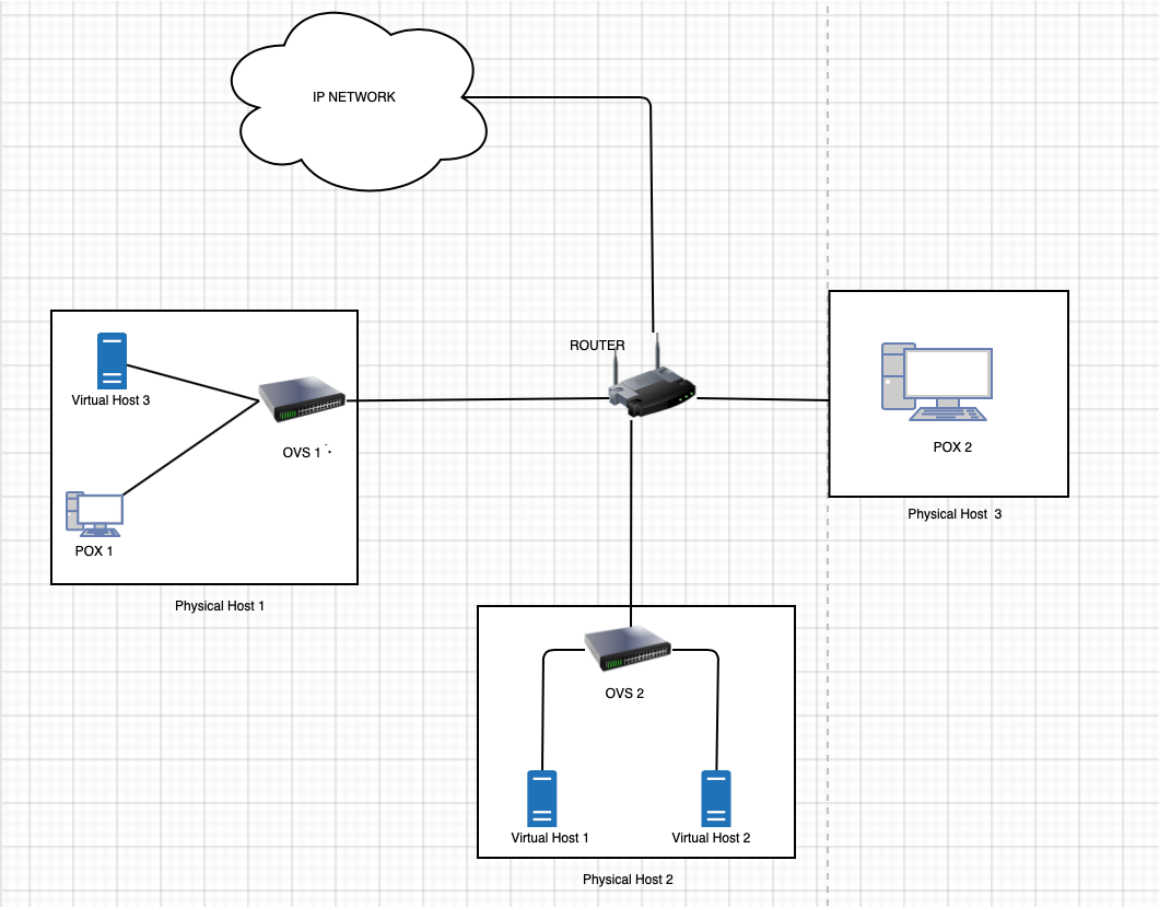


Θεωρητικά ο router μας , σύμφωνα με τα τεχνικά του χαρακτηριστικά , μπορεί να διαχειριστεί 300mbps, αυτό όμως δεν φαίνεται να ισχύει καθώς, παρατηρείται εκθετική αύξηση στα χαμένα πακέτα μόλις ξεπεραστούν τα 210mpbs. Μετά τα 400mbps βλέπουμε ότι σχεδόν κανένα πακέτο δεν φτάνει στον προορισμό του.

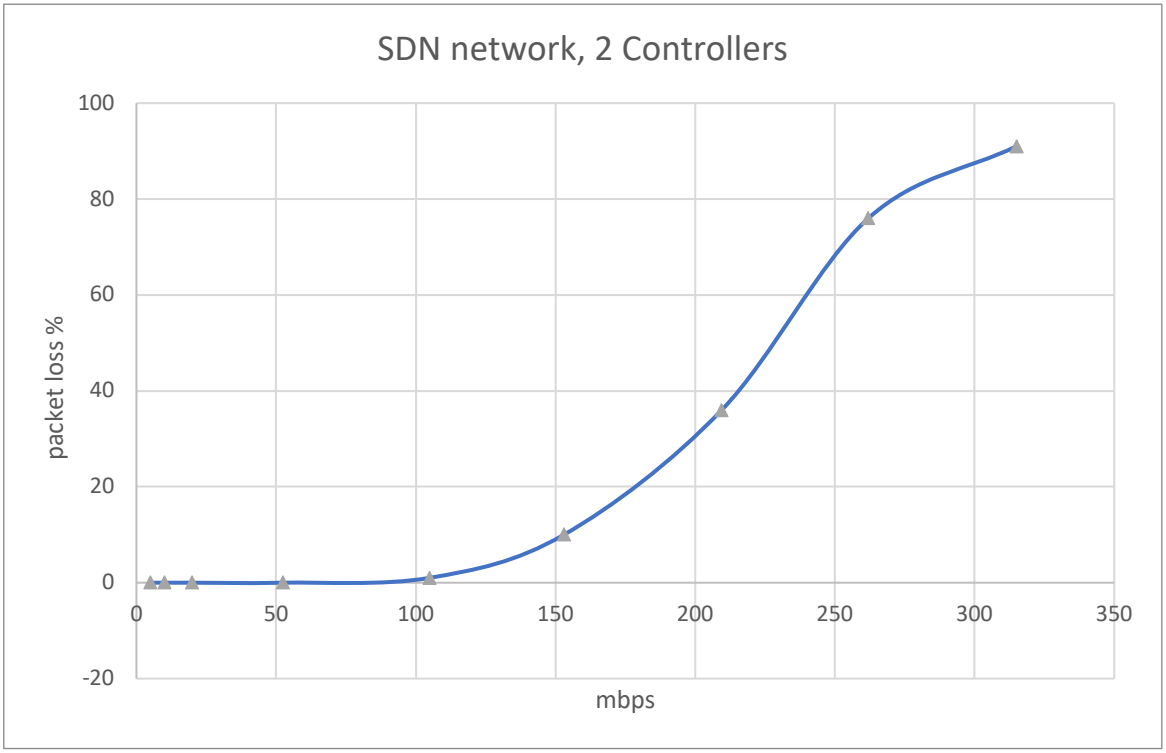
Για τον έλεγχο του SDN δικτύου , ξεκινήσαμε τα OVS switches και δημιουργήσαμε κίνηση μεταξύ του host 1 με host 3, και μεταξύ του host 2 με host 4. Ο Controller λειτουργούσε με το smartSwitch πρόγραμμα έτσι ώστε να ελαχιστοποιείται ο χρόνος επανεγκατάστασης ροών. Ακολουθεί η γραφική παράσταση με τα αποτελέσματα.



Το SDN δίκτυο , σταμάτησε να λειτουργεί εντελώς περίπου στα 200mbps. Το πρόγραμμα του Controller μας , δεν μπορούσε να διαχειριστεί τόση πολλή κίνηση και σταματούσε. Αν δούμε όμως τις μετρήσεις του, πριν να σταματήσει είναι αρκετά καλές αφού μέχρι τα 100mbps οι απώλεια πακέτων είναι σχεδόν μηδενική. Δεν πρέπει να ξεχνάμε το γεγονός ότι οι μηχανή που έτρεχε ο Controller είναι σχετικά αδύναμη μηχανή. Για την βελτίωση του δικτύου είχα 2 επιλογές, αλλαγή controller ή προσθήκη περισσότερων Controllers στο δίκτυο. Μετά από ίδιο πείραμα με διαφορετικό Controller , τον Floodlight , είχα τα ίδια αποτελέσματα πράγμα που επιβεβαίωσε τις υποψίες μου ότι αυτό ήταν θέμα του της περιορισμένης υπολογιστικής δύναμης των μηχανών. Έτσι εγκατέστησα ακόμα ένα Controller στο δίκτυο, στη θέση του virtual host 4 , έτσι ώστε κάθε switch να είναι συνδεδεμένο με 1 διαφορετικό και να καταμερίζεται ο φόρτος πακέτων. Έτσι πλέον δημιουργήσαμε την εξής τοπολογία με τον POX 1 να εξυπηρετεί το switch 1 και τον POX 2 να εξυπηρετεί το switch 2. Αμέσως μετά την τοπολογία ακολουθούν οι αντίστοιχες μετρήσεις .

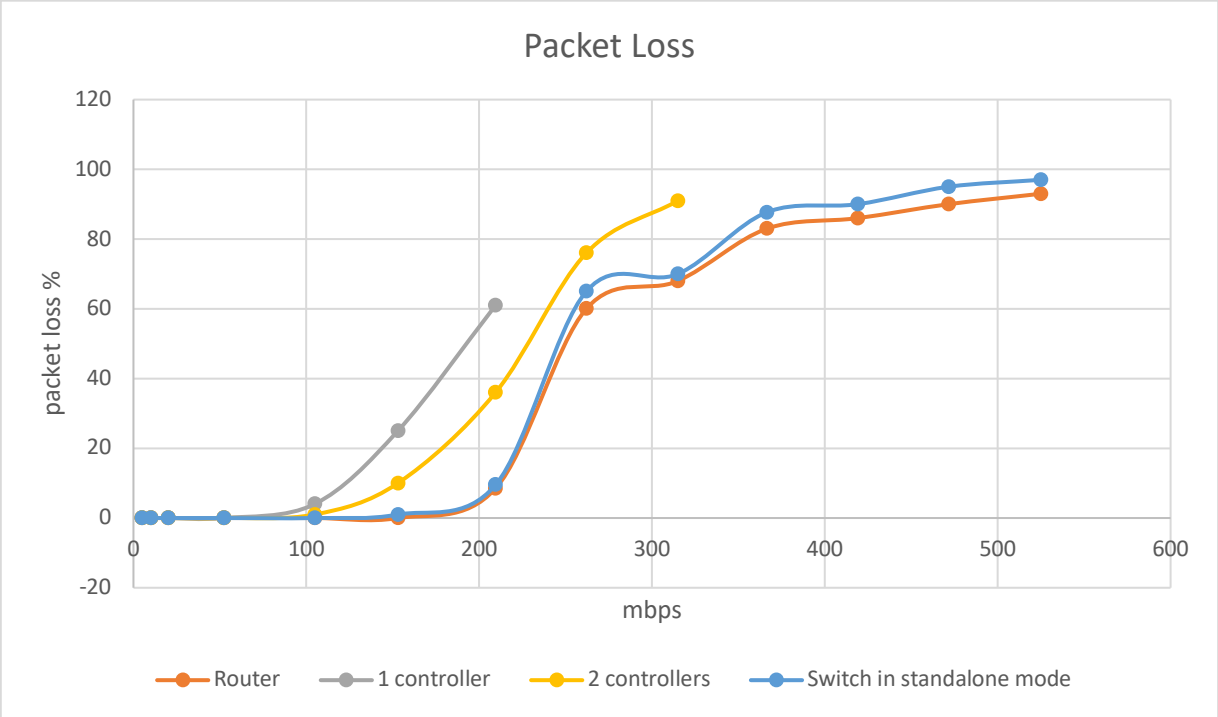


Εικόνα 42 : Τοπολογία με 2 Controllers



Όπως φαίνεται, η αντοχή του δικτύου βελτιώθηκε αρκετά. Από τα 200mpbs που άντεξε πριν τώρα αντέχει γύρο στα 310bps. Το λογικό θα ήταν να διπλασιαστεί αφού εγκαταστάθηκε ακόμα 1 ολόιδιος Controller, όμως δεν πρέπει να παραλείψουμε το γεγονός ότι στην μηχανή που έτρεχε ο καινούριος Controller έτρεχε ακόμη 1 VM και το OVS άρα οι διαθέσιμοι προς τον Controller πόροι ήταν αρκετά λιγότεροι.
Το packet loss , στα 300mbps ήταν αρκετά ψηλό με ποσοστό 87%. Αν το συγκρίνουμε όμως με το προηγούμενο μας δίκτυο , βλέπουμε και εδώ μια βελτίωση της τάξης 40-50 % . Επίσης δεν πρέπει να ξεχνάμε ότι στα 200-300mps ένα μεγάλο μερίδιο πακέτων χάνονται στον ίδιο τον router.

Αφού εξετάσαμε την αντοχή του δικτύου μας, το μόνο που έμενε ήταν να εξετάσουμε την αντοχή των OVS Switches μας. Γι' αυτή την μέτρηση απενεργοποιήσαμε τους Controllers για να μπουν τα switch στο stand-alone mode και επαναλάβαμε την διαδικασία δημιουργίας πακέτων. Ποιο κάτω φαίνεται ο γραφική παράσταση που περιλαμβάνει όλες τις μετρήσεις, μαζί με αυτή του OVS.



Όπως φαίνεται στην γραφική , τα αποτελέσματα του OVS είναι εξαιρετικά. Ακολουθεί παρόμοια πορειά με τον router. Μπορούμε να συμπεράνουμε ότι τα OVS φράσσονται από την απόδοση του router. Είναι εντυπωσιακό το πως ένα λογισμικό μπορεί να ανταγωνιστεί ένα αντίστοιχο hardware.

Κεφάλαιο 7

Σύνοψη, Συμπεράσματα και Μελλοντική εξέλιξη

7.1 Σύνοψη και Συμπεράσματα

Στη διπλωματική εργασία, μελετήθηκε κατά πόσο είναι εφικτή η κατασκευή ενός SDN δικτιού για χρήση σε σπίτια και μικρές επιχειρήσεις. Επίσης εξετάστηκε κατά πόσο είναι δυνατό η ενσωμάτωση απαραίτητων λειτουργιών στα δίκτια αυτά και έγινε ανάλυση της επίδοσης τους. Επιπρόσθετα ερευνήθηκαν τρόποι με τους οποίους μπορεί να γίνει βελτίωση της απόδοσης τους.

Μετά την κατασκευή του δικτιού SDN , κατέληξα στο συμπέρασμα ότι η κατασκευή ενός SDN δικτιού για την περίπτωση χρήσης η οποία μελετήθηκε είναι εφικτή με λίγα χρήματα και ενώ παράλληλα δεν απαιτεί εξειδικευμένο εξοπλισμό. Όπως είδαμε με την ενσωμάτωση , Load Balancer και Mac Blocker , στα SDN δίκτυα είναι αρκετά εύκολο να ενσωματωθούν λειτουργίες δικτύου στην μορφή NFVs, αφού ο κεντρικός έλεγχος και η δυνατότητα προγραμματισμού του τρόπου λειτουργίας των δικτύων αυτών , καθιστά εύκολη την ενσωμάτωση λειτουργιών με κάποιο απλό πρόγραμμα. Ο προγραμματισμός των δικτύων αυτών , δίνει επίσης την δυνατότητα για βελτίωση της επίδοσης και παραμετροποίησης των δικτύων, όπως είδαμε με την κατασκευή προγραμμάτων με OpenFlow εντολές, τα οποία τρέχουν στον Controller. Σύντομα αναμένεται , με το OpenFlow 2.0, ο προγραμματισμός να είναι δυνατός και για τα switches πράγμα που θα δώσει τον απόλυτο έλεγχο των δικτύων από τους ιδιοκτήτες τους. Όσον αφορά την απόδοση των δικτύων , τουλάχιστο με τα δικά μου αποτελέσματα , είναι εξαιρετική και παράλληλα πολλά υποσχόμενη. Το δίκτυο , μετά από τις βελτιστοποιήσεις, μπορεί να διαχειριστεί συνολική ροή πακέτων 150mbps , με ποσοστό απώλειας γύρο στο 10% ενώ μπορεί να αντέξει μέχρι και 300mbps πρωτού σταματήσει να λειτουργεί. Αυτά τα νούμερα μπορεί εκ πρώτης όψης να μην φαίνονται αρκετά καλά, αλλά είναι εξαιρετικά αν λάβουμε υπόψη τους περιορισμούς από τον router και από τους υπολογιστές που είχαμε , οι οποίοι έπρεπε να διαμοιράσουν τους ελάχιστους διαθέσιμους πόρους σε πολλά προγράμματα και VMs. Ο όποιος επιπρόσθετος εξοπλισμός , όπως εξειδικευμένοι Controllers και OF-Switches , αν και όχι απαραίτητος , είναι σίγουρο ότι θα προσφέρει στο δίκτυο εξαιρετικές επιδώσεις.

Τα SDN δίκτυα, στα δικά μου μάτια, είναι μονόδρομος για την πορειά που πρέπει να ακολουθήσουν τα δίκτυα. Η προγραμματιστική τους φύση θα επιτρέψει στην καινοτομία που απουσιάζει από τα δίκτυα να επανέλθει ενώ θα μεταφέρει τον έλεγχο τους από τους πάροχους, στους ιδιοκτήτες. Ήδη οι μεγαλύτερες εταιρίες στον κόσμο , επενδύουν στα δίκτυα αυτά και τα χρησιμοποιούν. Συγκεκριμένα η Google με την SDN υποδομή της κατάφερε να μειώσει

δραματικά το κόστος κατασκευής κατά 75% ενώ παράλληλα επιτεύχθηκε 100% αξιοποίηση των πόρων του δικτύου. Επίσης όπως ερευνήθηκε σε εργασία μου στα πλαίσια του μαθήματος ΕΠΛ450 Διαχείριση Δικτύων , τα δίκτυα αυτά θα συνδράμουν δραματικά στην καλύτερη διαχείριση των δικτύων η οποία σήμερα αντιμετωπίζει σημαντικές δυσκολίες.

Με τα στοιχεία αυτά , και τα δικά μου αποτελέσματα πιστεύω ότι τα SDN δίκτυα είναι ιδανικά για χρήση σε σπίτια και μικρές επιχρίσεις. Υπάρχουν φυσικά αρκετές προκλήσεις που πρέπει να αντιμετωπιστούν , με σημαντικότερη όλων την ασφάλεια, όμως με την επένδυση στα δίκτυα αυτά και την σταδιακή τους ενσωμάτωση , είμαι θετικός ότι η προκλήσεις θα αντιμετωπιστούν.

Ωστόσο η έλλειψη πηγών και πληροφοριών καθώς και η διακοπή υποστήριξης του Mininet, μου δημιουργεί αμφιβολίες και σκέψεις στο γεγονός ότι δεν επενδύονται οι κατάλληλοι πόροι και δεν παρέχονται τα απαραίτητα μέσα για την εδραίωση των SDN. Από την στιγμή, που στα δίκτυα 5G, τα οποία κάνουν τα πρώτα τους βήματα στη ζωή μας, το SDN είναι ένας από τους βασικούς τους πυλώνες, η επιστημονική κοινότητα πρέπει να δώσει μεγαλύτερη βαρύτητα σε αυτά έτσι ώστε να αντιμετωπιστούν τα προβλήματα τους για να εδραιωθούν ποιο σύντομα.

7.2 Μελλοντική εξέλιξη

Η μελλοντική εξέλιξη της διπλωματικής μου εργασίας στοχεύει στην επίλυση και την βελτίωση διαφόρων προβλημάτων και περιορισμών που υπάρχουν στο υφιστάμενό δίκτυο.

Περιορισμός ή Πρόβλημα	**Μελλοντική εξέλιξη**
Ενσωμάτωση Load balancer / Mac Blocker πρόγραμμα στον Controller	Πλήρης ενσωμάτωση σε NFVs Stack για καλύτερη επίδοση και διαχείριση.
Στον Mac Blocker , η εισαγωγή κανόνων γίνεται χειροκίνητα από τον προγραμματιστή.	Δημιουργία ενός πλήρους και λειτουργικού IDS που θα συνεργάζεται με τον Mac Blocker για την εισαγωγή κανόνων.

Χρήση Αδύνατων υπολογιστών	Χρήση μοντέρνων υπολογιστών με περισσότερους υπολογιστικούς πόρους
Χρήση Controller ανοικτού κώδικα.	Χρήση ενός ποιο εξειδικευμένου Controller
Χρήση Virtual switch.	Χρήση ενός εξειδικευμένου OF-enable switch.
Χρήση λίγων Virtual Hosts.	Χρήση περισσότερων πραγματικών μηχανών.
Μικρό δίκτυο.	Επέκταση δικτιού με περισσότερους hosts και switches
Βελτίωση επίδοσης.	Κατασκευή περισσότερων προγραμμάτων και πειραμάτων για βελτίωση επίδοσης του δικτύου.

Βιβλιογραφία

[1] OpenVSwitch feature, (available at source).

[2] Mininet Classes, (available at source)

[3] POX API , (available at source)

[4] Cisco chief cloud strategy consultant David Linthicum , 'SDN is Cloud evolve'. (available at source)

[5] 'Software Define Networks', Cisco (available at source)

[6] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart and Amin Vahdat ,Google Inc. 'B4: Experience with a Globally-Deployed Software Defined WAN'. (available at source)

[7] Michael Dalton, David Schultz, Jacob Adriaens, Ahsan Arefin, Anshuman Gupta, Brian Fahs, Dima Rubinstein, Enrique Cauich Zermeno, Erik Rubow, James Alexander Docauer, Jesse Alpert, Jing Ai, Jon Olson, Kevin DeCabooter, Marc de Kruijf, Nan Hua, Nathan Lewis, Nikhil Kasinadhuni, Riccardo Crepaldi, Srinivas Krishnan, Subbaiah Venkata, Yossi Richter, Uday Naik, and Amin Vahdat, Google, Inc. 'Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization' (available at source).

[7] KK Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, TaeEun Kim, Ashok Narayanan, Ankur Jain, Victor Lin, Colin Rice, Brian Rogan, Arjun Singh, Bert Tanaka, Manish Verma, Puneet Sood, Mukarram Tariq, Matt Tierney, Dzevad Trumic, Vytautas Valancius, Calvin Ying, Mahesh Kallahalla, Bikash Koley, Amin Vahdat , 'Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering' (available at source).

[8] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat, Google, Inc. , 'Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network' (available at source).

[9] Xenofon Foukas, Mahesh K. Marina, Kimon Kontovasilis, 'Software Defined Networking Concepts' (available at source).

[10] Sotiris Michaelides, 'SDN and its Management Aspects' (available at source).

[11] David Mahler, 'Introduction to OpenVswitch' (available at source).

[12] David Mahler, 'Introduction to OpenFlow' (available at source).

[13] David Mahler, 'Introduction to SDN' (available at source).

[14] Mininet team, 'OF-Tutorial'(available at source).

Παράρτημα Α

Το παράρτημα Α περιέχει όλα τα αρχεία κώδικα που χρησιμοποιήθηκαν / ή δημιουργήθηκαν σε αυτήν την διπλωματική εργασία.

Α-1 Κώδικας Load Balancer : Source

Α-2 Κώδικας Mac Blocker : Source

Α-3 Κώδικας Smart Switch : Source

Α-4 Κώδικας Smart Switch (για βελτίωση TCP traffic) : Source

Α-5 Κώδικας Τοπολογίας : Source

Α-6 Κώδικας Router Line source