ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μελέτη των Δικτύων που καθορίζονται από Λογισμικό (Software Defined Networking - SDN) και υλοποίηση εφαρμογής παραμετροποίησης δικτυακών υπηρεσιών με βάση το Ryu SDN framework

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΤΑΤΟ ΙΑΝΤΙ

(ΑΕΜ: 2601)

Επιβλέπων: **Νικολάου Σπυρίδων**

Λέκτορας

Καστοριά Μήνας - Έτος (παρουσίασης της εργασίας)

Η παρούσα σελίδα σκοπίμως παραμένει λευκή

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΉΜΑ ΠΛΗΡΟΦΟΡΙΚΉΣ

Μελέτη των Δικτύων που καθορίζονται από Λογισμικό (Software Defined Networking - SDN) και υλοποίηση εφαρμογής παραμετροποίησης δικτυακών υπηρεσιών με βάση το Ryu SDN framework

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

του

ΤΑΤΟ ΙΑΝΤΙ

(ΑΕΜ: 2601)

Επιβλέπων: **Νικολάου Σπυρίδων**

Λέκτορας

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ημερομηνία εξέτασης

.....	.....	.....
Ον/μο Μέλους	Ον/μο Μέλους	Ον/μο Μέλους
Ιδίοτητα Μέλους	Ιδίοτητα Μέλους	Ιδίοτητα Μέλους

Καστοριά Μήνας - Έτος (παρουσίασης της εργασίας)

Ευχαριστίες

Για την εκπόνηση της παρούσας πτυχιακής εργασίας, θα ήθελα να ευχαριστήσω θερμά τον καθηγητή μου κ. Νικολάου, για την πολύτιμη καθοδήγηση και τη συνεργασία του, καθ' όλη τη διάρκεια.

Θα ήθελα, επίσης, να εκφράσω τις ευχαριστίες μου στην οικογένεια μου, για την αμέριστη συμπαράσταση, την υπομονή τους και την κατανόηση που επέδειξαν.

Περίληψη

Η ραγδαία άνοδος της δημοτικότητας του διαδικτύου την τελευταία δεκαετία κατέστησε σαφές ότι η κάλυψη των μελλοντικών απαιτήσεων της αγοράς είναι σχεδόν αδύνατη με τη χρήση καθιερωμένων αρχιτεκτονικών δικτύωσης. Ως εκ τούτου, υπάρχει ανάγκη ανάπτυξης νέων αρχιτεκτονικών δικτύωσης με αυξημένες δυνατότητες που θα μπορούν να εξυπηρετήσουν αποτελεσματικά τις ανάγκες των μελλοντικών χρηστών, ευνοώντας παράλληλα τη δημιουργία νέων αποδοτικών υπηρεσιών.

Η τεχνολογία Software Defined Networking (SDN) είναι το κλειδί στην επίλυση του άνωθεν προβλήματος. Η συγκεκριμένη τεχνολογία είναι μια αναδυόμενη αρχιτεκτονική δικτύωσης στην οποία ο έλεγχος (control plane) έχει διαχωριστεί από την προώθηση (data plane) και είναι άμεσα προγραμματιζόμενος. Αυτή η μεταγωγή του ελέγχου από τις συσκευές δικτύου σε υπολογιστικές μηχανές προσφέρει τη δυνατότητα στην υποκείμενη υποδομή να αποσπαστεί από την υλοποίηση των εφαρμογών και των δικτυακών υπηρεσιών. Το δίκτυο θα αντιμετωπίζεται πλέον ως μια λογική ή εικονική οντότητα, ενώ ο ελεγκτής θα είναι υπεύθυνος για τον έλεγχο και την ενορχήστρωση των λειτουργιών του.

Η παρούσα πτυχιακή εργασία έχει ως στόχο την εξερεύνηση της τεχνολογίας SDN, την παρουσίαση των βασικών αρχών της αρχιτεκτονικής και στην συνέχεια την ανάπτυξη μίας εφαρμογής SDN με την χρήση του Ryu SDN framework. Θα παρουσιαστούν επίσης το πρωτόκολλο OpenFlow και οι SDN ελεγκτές (controllers) και μεταγωγείς (switches), ενώ θα ακολουθήσει μια ανάλυση του πλαισίου λειτουργίας της πλατφόρμας Ryu παρουσιάζοντας πολλά από τα χαρακτηριστικά της, καθώς επίσης και σειρά δοκιμών σε περιβάλλον εργαστηρίου που αποδεικνύουν τις αναφερθείσες λειτουργίες του ελεγκτή.

Συνοπτικά η τεχνολογία SDN προσφέρει μια πλειάδα νέων υπηρεσιών και εφαρμογών δικτύου οι οποίες εύκολα αναπτύσσονται, εφαρμόζονται και διαλειτουργούν. Θα είναι μια πρόκληση με την τεχνολογία SDN, όσο παράλληλα οι πάροχοι και επιχειρήσεις θα επωφελούνται, καθώς οι νέες αυτές υπηρεσίες θα αυξήσουν τα έσοδα και θα μειώσουν τα κόστη τους.

Λέξεις Κλειδιά: Δικτύωση καθοριζόμενη από λογισμικό, Ryu, OpenFlow, Mininet, SDN Controller, Multipath Load Balancer, Flow, Network Virtualization

Abstract

The rapid rise in popularity of the internet over the last decade has made it clear that meeting the market demands of the future is almost impossible with the use of established networking architectures. Therefore, there is a need to develop new networking architectures with increased capabilities that can effectively serve the needs of the future users, while favoring the creation of new efficient services. Software Defined Networking (SDN) technology is the key to solving the above problem. This technology is an emerging networking architecture in which the control plane is separated from the data plane and is directly programmable. This transfer of control from the network devices to the computing machines enables the underlying infrastructure to be detached from the implementation of applications and network services. The network will now be treated as a logical or virtual entity, while the controller will be responsible for controlling and orchestrating its functions.

The present dissertation aims to explore SDN technology, introduce the basic principles of the architecture and then develop an SDN application using the Ryu SDN framework. The protocol and OpenFlow switches will also be presented, followed by an analysis of the operating framework of the Ryu platform presenting many of its features as well as a series of tests in a laboratory environment that demonstrate the mentioned functions of the controller.

In short, SDN technology offers a host of new services and applications that are easy to develop, implement and operate. Applications and features that seem impossible today will simply be a challenge with SDN technology, while providers and businesses will benefit as these new services will increase revenue and reduce costs.

Key Words: Software Defined Networking, Ryu, OpenFlow, Mininet, SDN Controller, Multipath Load Balancer, Flow, Network Virtualization.

Πίνακας Περιεχομένων

Λίστα Εικόνων

Λίστα Πινάκων

Εισαγωγή

Οι σύγχρονες τεχνολογίες και ιδιαίτερα το διαδίκτυο (Internet), η εξάπλωση του οποίου χαρακτήρισε τις τελευταίες δεκαετίες, έχουν αποτελέσει τη βάση της οικονομικής ανάπτυξης τα τελευταία χρονιά σε παγκόσμιο επίπεδο. Μάλιστα, τόσο η δυναμική τους όσο και οι δυνατότητες αξιοποίησης που προσφέρουν έχουν οδηγήσει σε τέτοιες σημαντικές αλλαγές, ώστε πολλοί έγκυροι αναλυτές να παρομοιάζουν σήμερα τη κατάσταση με την βιομηχανική επανάσταση του 19^{ου} αιώνα. Η ραγδαία εξέλιξη των νέων τεχνολογιών (π.χ. 5G, cloud computing, big data), η ευρεία τους διάχυση σε όλο το φάσμα της οικονομίας και η ενσωμάτωση τους σε όλες σχεδόν τις διαστάσεις της καθημερινής ζωής χτίζουν μια Κοινωνία της Πληροφορίας με παγκόσμια διάσταση που προβάλει νέα δεδομένα και προσφέρει νέες ευκαιρίες για ανάπτυξη, ανταγωνιστικότητα, απασχόληση, ευημερία και ποιότητα ζωής.

Στο σημείο αυτό αντιλαμβανόμαστε πως η αλλαγή αυτή επιβάλλει νέες προκλήσεις στο μελλοντικό Διαδίκτυο, για τις οποίες η πανταχού παρούσα προσβασιμότητα, το υψηλό εύρος ζώνης και η δυναμική διαχείριση είναι ζωτικής σημασίας. Ωστόσο οι παραδοσιακές προσεγγίσεις που βασίζονται στη μη αυτόματη διαμόρφωση των συσκευών είναι δυσμεταχείριστες και επιρρεπείς σε σφάλματα, ενώ δεν μπορούν να αξιοποιήσουν πλήρως τις δυνατότητες της φυσικής υποδομής δικτύου. Οι σημερινές τεχνολογίες δικτύωσης υπολογιστών αντιμετωπίζουν τις απαιτήσεις διαφορετικών ενδιαφερομένων, συμπεριλαμβανομένων των διαχειριστών δικτύου, των παροχών υπηρεσιών, των προγραμματιστών και των τελικών χρηστών. Οι απαιτήσεις αυτές είναι συχνά συσχετισμένες, αλλά διακριτές και ενδέχεται να έρχονται σε σύγκρουση μεταξύ τους.

Εν ολίγοις ο παραδοσιακός σχεδιασμός δικτύου δεν διαθέτει την ικανότητα να ικανοποιήσει το ευρύ φάσμα των απαιτήσεων λόγω της οστεοποίησης της τρέχουσας αρχιτεκτονικής δικτύου, η οποία προέρχεται κυρίως από: (α) την ενοποίηση των λειτουργιών ελέγχου και προώθησης που δημιουργεί ένα σύνθετο και άκαμπτο περιβάλλον ελέγχου και διαχείρισης δικτύου, και (β) τη στενή σύνδεση μεταξύ των υπηρεσιών και της υποδομής του δικτύου που περιορίζει την ικανότητα του δικτύου όσον αναφορά την εξέλιξη. Δύο σημαντικές πρόσφατες καινοτομίες στις τεχνολογίες δικτύωσης για την αντιμετώπιση των προαναφερθεισών προκλήσεων είναι η δικτύωση που καθορίζεται από λογισμικό (**Software Defined Networking, SDN**) και η εικονικοποίηση λειτουργιών δικτύου (**Network Function Virtualization, NFV**).

Η βασική ιδέα του SDN έγκειται στην αποσύνδεση των λειτουργιών ελέγχου και διαχείρισης δικτύου από τις λειτουργίες προώθησης δεδομένων για την δημιουργία μιας κεντρικής πλατφόρμας ελέγχου η οποία θα υποστηρίζει τον ευέλικτο προγραμματισμό του δικτύου. Τα βασικά στοιχεία της αρχιτεκτονικής SDN περιλαμβάνουν ένα επίπεδο δεδομένων που αποτελείται από τις συσκευές δικτύου που εκτελούν την προώθηση δεδομένων, ένα επίπεδο ελέγχου που παρέχει έναν λογικά συγκεντρωτικό έλεγχο των πόρων του δικτύου και ένα επίπεδο εφαρμογών, όπου μέσω ενός ελεγκτή γίνεται ο προγραμματισμός της συμπεριφοράς του δικτύου.

Διάρθρωση της Εργασίας

Στο πρώτο κεφάλαιο αναλύεται η έννοια της Εικονικοποίησης καθώς και τα οφέλη της, ενώ ύστερα γίνεται αναφορά στην Εικονικοποίηση Λειτουργιών Δικτύου και των πεδίων εφαρμογής της.

Στο δεύτερο κεφάλαιο γίνεται μια εκτενής παρουσίαση της αρχιτεκτονικής SDN όπου και εξετάστηκαν οι θεμελιώδεις ιδέες, τα διαφορετικά επίπεδα, το πρωτόκολλο OpenFlow, το επίπεδο ασφάλειας και οι διάφορες υλοποιήσεις ελεγκτών.

Το τρίτο και τελευταίο κεφάλαιο περιέχει το πειραματικό τμήμα της εργασίας. Αρχικά γίνεται μια σύντομη εισαγωγή στον ελεγκτή Ryu και το πειραματικό περιβάλλον, ενώ στη συνέχεια περιγράφεται η διαδικασία υλοποίησης των διάφορων εφαρμογών που αναπτύχθηκαν.

Τέλος, η εργασία κλείνει με μια ανάλυση της απόδοσης του ελεγκτή υπό διάφορα προσομοιωμένα σενάρια.

1. Εικονικοποίηση

Η υποστήριξη ενός ευρέος φάσματος υπηρεσιών με διαφορετικές απαιτήσεις που βασίζονται σε δίκτυα με ετερογενείς τεχνολογίες έχει καταστεί μια σημαντική πρόκληση για την έρευνα και την ανάπτυξη νέων τεχνολογιών δικτύωσης στην τρέχουσα αρχιτεκτονική δικτύου IP [1].

Η αρχιτεκτονική αυτή πιθανόν να μην μπορεί να ικανοποιήσει τις απαιτήσεις του δικτύου του μέλλοντος. Για να αντιμετωπίσουν το δύσκολο αυτό πρόβλημα, οι ερευνητές έχουν προτείνει ένα πλαίσιο οπού ο σχεδιασμός υιοθετεί την **εικονικοποίηση** ως βασικό χαρακτηριστικό για τις μελλοντικές αρχιτεκτονικές δικτύου.

Η εικονικοποίηση ως έννοια αναφέρεται στην πράξη διαχωρισμού του λογισμικού από το υποκείμενο υλικό για τη δημιουργία εικονικών αναφορών στους πόρους του συστήματος ή πιο γενικά μπορούμε να πούμε πως είναι η δυνατότητα παράλληλης λειτουργίας πολλαπλών επιπέδων λογισμικού/υλικού στο ίδιο σύστημα ως προσομοιώσεις πραγματικών συστημάτων. Οι τεχνολογίες εικονικοποίησης έχουν χρησιμοποιηθεί ευρέως σε διάφορους τομείς , όπως το cloud computing & storage virtualization , όπου και η επιτυχία τους έχει εμπνεύσει την υιοθέτησή τους και στον τομέα της δικτύωσης. Η εφαρμογή της εικονικοποίησης σε δίκτυα οδηγεί στις έννοιες της εικονικοποίησης δικτύου (**Network Virtualization, NV**) και της εικονικοποίησης λειτουργιών δικτύου (**Network Function Virtualization, NFV**).

1.1. Ορισμός της Εικονικοποίησης Δικτύου

Παρόλο που η ερευνητική κοινότητα προσπάθησε να ακολουθήσει μια νέα προσέγγιση ως προς την ανάπτυξη νέων αρχιτεκτονικών και πρωτοκόλλων για την αντιμετώπιση των προκλήσεων της μελλοντικής δικτύωσης, η ανάπτυξη καινοτόμων τεχνολογιών περιορίζεται στην ουσία σε προσεγγίσεις οι οποίες απλώς επικαλύπτουν το πρόβλημα με μικρή ικανότητα εισαγωγής θεμελιωδών αλλαγών στην βασική αρχιτεκτονική. Όπως επεσήμανε ο Άντερσον [2], οι υπάρχουσες προσεγγίσεις «επικάλυψης» δεν μπορούν να παρέχουν μια αποτελεσματική πορεία ανάπτυξης για ριζοσπαστικές τεχνολογίες δικτύωσης για δύο κύριους λόγους: πρώτον, χρησιμοποιούνται ως επί το πλείστων για την ανάπτυξη σταδιακών λύσεων σε συγκεκριμένα προβλήματα χωρίς μια ολιστική άποψη των αλληλεπιδράσεων μεταξύ των συνυπαρχόντων τεχνολογιών. Δεύτερον, οι λύσεις αυτές συχνά σχεδιάζονται και αναπτύσσονται στο επίπεδο εφαρμογής στο πάνω μέρος του IP και επομένως δεν μπορούν να υποστηρίξουν μια ριζικά διαφορετική αρχιτεκτονική δικτύωσης.

Οι ερευνητές έχουν συνειδητοποιήσει ότι το κλειδί για την υπέρβαση της οστεοποίησης της τρέχουσας αρχιτεκτονικής και του τέλους της αδιεξόδου στις καινοτομίες δικτύου έγκειται στην αποσύνδεση των λειτουργιών δικτύου για την παροχή υπηρεσιών και των υποδομών μεταφοράς και επεξεργασίας. Μια τέτοια αποσύνδεση επιτρέπει την ανάπτυξη και εφαρμογή εναλλακτικών αρχιτεκτονικών και πρωτοκόλλων για την κάλυψη των διάφορων απαιτήσεων δίχως όμως να υπάρχουν περιορισμοί από τα χαρακτηριστικά της φυσικής υποδομής.

Οι τεχνολογίες εικονικοποίησης έχουν ήδη χρησιμοποιηθεί στη δικτύωση σε διάφορα σενάρια. Για παράδειγμα, ένα εικονικό τοπικό δίκτυο (**VLAN**) σχηματίζει έναν μοναδικό τομέα

μετάδοσης, διασυνδέοντας λογικά μια ομάδα υπολογιστών, και ένα εικονικό ιδιωτικό δίκτυο (**VPN**) επιτρέπει ιδιωτικές συνδέσεις μεταξύ πολλών τοποθεσιών χρησιμοποιώντας ασφαλείς "*σήραγγες*" μέσω κοινών δημόσιων δικτύων, ωστόσο, καμία από τις προαναφερθείσες προσεγγίσεις εικονικοποίησης δεν είναι ικανή να αντιμετωπίσει πλήρως τις ανεπάρκειες της αρχιτεκτονικής. Αυτό οφείλεται κυρίως στο ότι αυτές οι ερευνητικές προσπάθειες βασίζονται σε μια συμβατική άποψη ως προς την αρχιτεκτονική του δικτύου που ονομάζεται «**purist view**» [3]. Η άποψη αυτή πρεσβεύει μια αρχιτεκτονική γενικής χρήσης που παρέχει μια κατάλληλη πλατφόρμα για το σύνολο όλων των υπαρχόντων και αναδυόμενων εφαρμογών. Μια τέτοια πλατφόρμα (π.χ. το πρωτόκολλο IP) σχηματίζει ένα «***σημείο συμφόρησης***» στην αρχιτεκτονική που βρίσκεται ανάμεσα σε όλες τις διάφορες εφαρμογές και τεχνολογίες μεταφοράς. Εάν το σημείο είναι μια πλατφόρμα παράδοσης πακέτων από άκρο σε άκρο, τότε οποιαδήποτε σημαντική τροποποίηση αυτής της πλατφόρμας απαιτεί καθολική συμφωνία και συντονισμό μεταξύ όλων των εμπλεκόμενων φορέων, κάτι που είναι πολύ δύσκολο - αν όχι αδύνατο - στο σημερινό Διαδίκτυο.

Αντιθέτως, η «**pluralist**» άποψη υποστηρίζει τη συνύπαρξη εναλλακτικών αρχιτεκτονικών δικτύωσης πάνω σε μια κοινόχρηστη υποδομή, η οποία παρέχει ένα μέσο για να ξεπεραστεί η οστεοποίηση του σημερινού διαδικτύου και εισάγει μια πρόσθετη ελευθερία σχεδιασμού για τις μελλοντικές αρχιτεκτονικές. Η πλουραλιστική άποψη επιτρέπει στους σχεδιαστές δικτύου να εκμεταλλευτούν πλήρως τη δύναμη της εικονικοποίησης για να αντιμετωπίσουν ορισμένες από τις θεμελιώδεις προκλήσεις που αναφερθήκαμε προηγουμένως. Στα σχέδια δικτύωσης που ακολουθούν την πλουραλιστική άποψη, η εικονικοποίηση είναι ένα βασικό αρχιτεκτονικό χαρακτηριστικό που υποστηρίζει την συνύπαρξη ανεξάρτητων εικονικών δικτύων με διαφορετικές αρχιτεκτονικές και πρωτόκολλα σε ένα κοινό υπόστρωμα φυσικής υποδομής.

Επομένως, ο βασικός στόχος του NV είναι να δημιουργήσει ένα περιβάλλον δικτύωσης το οποίο επιτρέπει σε πολλά ανεξάρτητα εικονικά δίκτυα (**Virtual Networks, VN**) να μοιράζονται μια κοινή υποδομή δικτύου. Κάθε VN μπορεί να έχει τη δική του αρχιτεκτονική δικτύου, συμπεριλαμβανομένης της μορφής πακέτου, του σχήματος διευθύνσεων, του μηχανισμού προώθησης, των πρωτοκόλλων δρομολόγησης και ούτω καθεξής, σχεδιασμένο να παρέχει κάθε είδους υπηρεσίας για την κάλυψη των διαφορετικών απαιτήσεων της εκάστοτε εφαρμογής.

1.1.1. Οφέλη της εικονικοποίησης

Συνήθως, ένα εικονικό **network link** είναι ένας σύνδεσμος που δεν αποτελείται από μια φυσική (ενσύρματη ή ασύρματη) σύνδεση μεταξύ δύο υπολογιστικών συσκευών αλλά υλοποιείται χρησιμοποιώντας μεθόδους εικονικοποίησης [4]. Έτσι, η διεπαφή και οι πόροι του εικονικού συστήματος χαρτογραφούνται στη διεπαφή και στους πόρους ενός άλλου «πραγματικού» συστήματος, ενώ η εικονικοποίηση παρέχει τους απαιτούμενους πόρους έτσι ώστε να μην χάνει σημαντικές λειτουργίες το εικονικοποιημένο σύστημα.

Εστιάζοντας στην εικονικοποίηση δικτύου, τα οφέλη συνοψίζονται ως εξής:

- ***Βελτιστοποίηση πόρων:*** Για την ικανοποίηση πραγματικών αναγκών, το διαθέσιμο εύρος ζώνης δικτύου μπορεί να χρησιμοποιηθεί πολύ πιο αποτελεσματικά. Οι σήραγγες και τα εικονικά δίκτυα μπορούν να απομονώσουν ή να περιορίσουν την

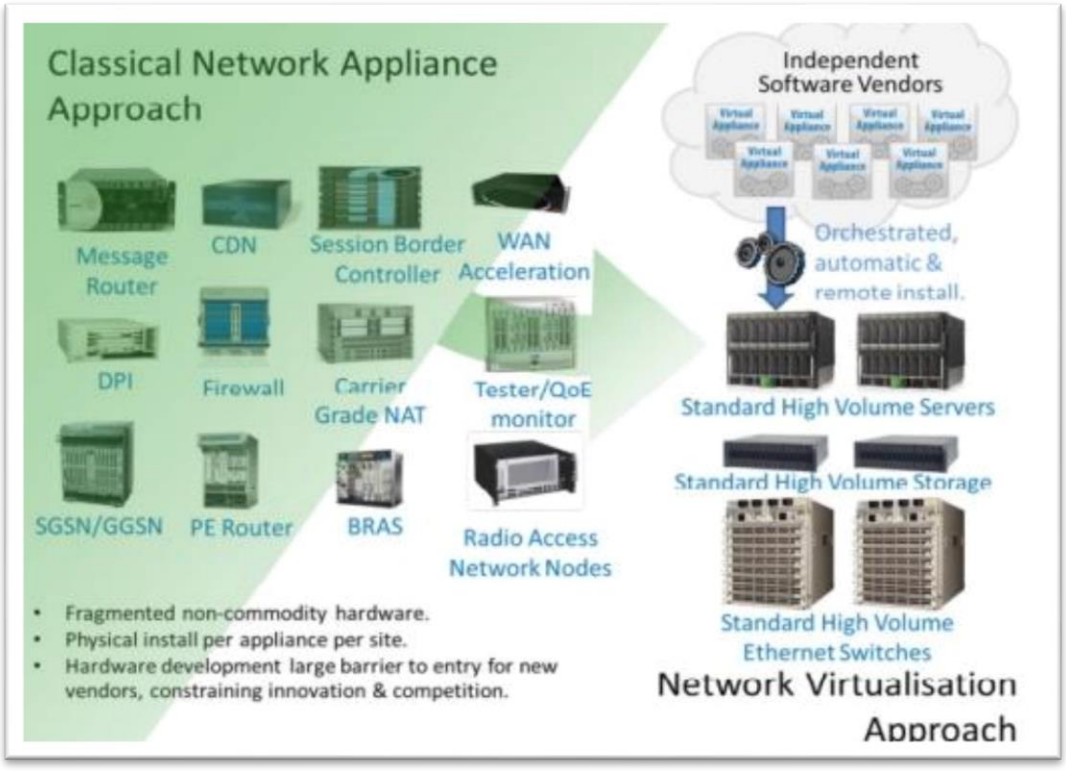
κυκλοφορία του δικτύου βάσει των αναγκών της κάθε εφαρμογής, παρέχοντας έτσι μια αυξημένη δυνατότητα ελέγχου εντός του δικτύου σε σύγκριση με άλλες παραδοσιακές λύσεις.

- ***Πολλαπλά περιβάλλοντα εκτέλεσης:*** Τα εικονικά δίκτυα επιτρέπουν ένα απομονωμένο περιβάλλον λειτουργίας πολλαπλών εφαρμογών που μπορεί να αποφέρει πολύπλευρα οφέλη όπως, απομόνωση, περιορισμό της κακόβουλης κυκλοφορίας εντός του δικτύου, παροχή πλεονάζοντος (*redundancy*) για εφεδρικά αντίγραφα, δυνατότητες κοινής χρήσης φορτίου κ.λπ.
- ***Εντοπισμός σφαλμάτων και εισβολής:*** Χρησιμοποιώντας εικονικά δίκτυα, μπορούμε να δημιουργήσουμε τεμάχια (*slices*), μια προσέγγιση απομόνωσης όπου γίνεται κράτηση ενός συγκεκριμένου ποσού πόρων δικτύωσης για μια συγκεκριμένη εφαρμογή. Σε περίπτωση υποεπιθέσεων, ένα *τεμάχιο* μπορεί να αντιμετωπιστεί εύκολα, για παράδειγμα, με την εκτέλεση εντοπισμού εισβολής, παρακολούθησης και Deep Packet Inspection (DPI) .
- ***Κινητικότητα:*** Η εικονικοποίηση όχι μόνο παρέχει μια προγραμματιζόμενη προσέγγιση ως προς την παραμετροποίηση και διαχείριση νέων υπηρεσιών, αλλά μπορεί επίσης να βοηθήσει άψογα στην μετεγκατάσταση λογισμικού. Για παράδειγμα, η *ζωντανή* μετεγκατάσταση περιλαμβάνει τη μετεγκατάσταση του περιεχομένου μνήμης ενός VM (Virtual Machine), τη διατήρηση της πρόσβασης στην υπάρχουσα υποδομή αποθήκευσης που περιέχει το αποθηκευμένο περιεχόμενο του VM, ενώ παρέχει συνεχή συνδεσιμότητα δικτύου στον υπάρχοντα τομέα δικτύου. Οι υφιστάμενες συναλλαγές του VM μπορούν να διατηρηθούν και οποιαδήποτε νέα συναλλαγή θα επιτρέπεται σε οποιοδήποτε στάδιο της ζωντανής μετεγκατάστασης, παρέχοντας έτσι συνεχή διαθεσιμότητα δεδομένων.
- ***Ασφάλεια Συσκευής:*** Δυνατότητα στιγμιαίας ενεργοποίησης / απενεργοποίησης των plug-and-play λειτουργιών ασφαλείας που μπορούν εύκολα να εφαρμοστούν σε οποιοδήποτε τμήμα του δικτύου ή να συνδεθούν με υπάρχουσες υπηρεσίες στο cloud.
- ***Δοκιμή / Διασφάλιση ποιότητας:*** Τα εικονικά δίκτυα προσφέρουν απομονωμένους, περιορισμένους και δοκιμαστικούς χώρους στους προγραμματιστές. Αντί να χρειάζεται η αγορά και η διατήρηση αποκλειστικού φυσικού υλικού, τα εικονικά δίκτυα μπορούν να δημιουργήσουν πολλαπλά απομονωμένα περιβάλλοντα δικτύωσης. Μαζί με τα VMs και τους εικονικούς χώρους αποθήκευσης , μπορούμε γρηγορά να δημιουργήσουμε περιβάλλοντα δοκίμων, όπου ο κάθε προγραμματιστής θα μπορεί να δημιουργεί έναν απεριόριστο αριθμό διαμορφώσεων πάνω στα φυσικά μηχανήματα και να επιλεγεί την καταλληλότερη διαμόρφωση σε κάθε στάδιο. Αυτό δίνει τη δυνατότητα πειραματισμού με δυνητικά ασύμβατες εφαρμογές και εκτέλεση δοκιμών με διαφορετικά προφίλ χρηστών. Επιπλέον, οι δοκιμές θα είναι ανεξάρτητες από το υλικό, φορητές και η δημιουργία αντίγραφων ασφαλείας και θα καταστεί εύκολη.

1.2. Εικονικοποίηση Λειτουργιών Δικτύου

Τα δίκτυα των φορέων εκμετάλλευσης δικτύου (Internet Service Provider) αποτελούνται από μια μεγάλη και αυξανόμενη ποικιλία ιδιόκτητων φυσικών συσκευών [5]. Η εγκατάσταση μιας νέας υπηρεσίας απαιτεί συχνά την ανάπτυξη/εγκατάσταση πρόσθετων συσκευών, μια διαδικασία που γίνεται ολοένα και πιο δύσκολη σε συνδυασμό με το αυξανόμενο κόστος ενέργειας, τις προκλήσεις για επενδυτικό κεφάλαιο και τη δυσεύρετες δεξιότητες που απαιτούνται για το σχεδιασμό, την ενσωμάτωση και τη λειτουργία ολοένα και πιο περίπλοκων συσκευών. Επιπλέον, οι συσκευές που βασίζονται σε υλικό φτάνουν γρήγορα στο τέλος της ζωής τους, απαιτώντας μεγάλο μέρος του κύκλου σχεδίασης & ανάπτυξης να επαναληφθεί με ελάχιστο ή μηδενικό επενδυτικό όφελος.

Η εικονικοποίηση των λειτουργιών του δικτύου στοχεύει στην αντιμετώπιση αυτών των προβλημάτων εξελίσσοντας τις υπάρχουσες τεχνολογίες εικονικοποίησης με τέτοιο τρόπο ώστε να γίνει ενοποίηση των διάφορων τύπων εξοπλισμού σε τυποποιημένους διακομιστές υψηλής απόδοσης , μεταγωγείς και χώρους αποθήκευσης, οι οποίοι θα μπορούν να βρίσκονται σε data centers, κόμβους και στις εγκαταστάσεις του τελικού χρήστη, όπως φαίνεται στην Εικόνα 1. Το NFV περιλαμβάνει την υλοποίηση των λειτουργιών δικτύου σε λογισμικό που μπορεί να εκτελεστεί σε μια σειρά από industry-standard server hardware που μπορεί να μετακινηθεί ή να δημιουργηθεί σε διάφορες τοποθεσίες στο δίκτυο, όπως απαιτείται, χωρίς την ανάγκη εγκατάστασης νέου εξοπλισμού.



Εικόνα 1: Network Virtualization Approach
Πηγή: [5] p.5

1.2.1. Πεδία εφαρμογής, περιπτώσεις χρήσης και οφέλη NFV

Το NFV μπορεί να εφαρμοστεί για οποιαδήποτε επεξεργασία πακέτου δεδομένων και λειτουργία του επιπέδου ελέγχου σε δίκτυα κινητής και σταθερής [5]. Πιθανά παραδείγματα χρήσης είναι τα:

- **Switching elements:** BNG, CG-NAT, routers.
- **Mobile network nodes:** HLR/HSS, MME, SGSN, GGSN/PDN-GW, RNC, Node B, eNode B.
- Λειτουργίες που περιέχονται σε οικιακούς δρομολογητές και αποκωδικοποιητές για τη δημιουργία εικονικοποιημένων οικιακών περιβαλλόντων.
- **Tunnelling gateway elements:** IPSec/SSL VPN gateways.
- **Traffic analysis:** DPI, QoE measurement.
- **Service Assurance:** SLA monitoring, Test and Diagnostics.
- **NGN signaling:** SBCs, IMS.
- **Converged and network-wide functions:** AAA servers, policy control and charging platforms.
- **Application-level optimization**: CDNs, Cache Servers, Load Balancers, Application Accelerators.
- **Security functions:** Firewalls, virus scanners, intrusion detection systems, spam protection.

Ενώ συνοπτικά το NFV θα μπορούσε ενδεχομένως να προσφέρει τα ακόλουθα οφέλη:

- Μειωμένο κόστος εξοπλισμού και κατανάλωσης ενέργειας μέσω της ενοποίησης του εξοπλισμού και της εκμετάλλευσης των οικονομιών κλίμακας της βιομηχανίας.
- Μείωση του χρόνου "*time to market*" ελαχιστοποιώντας τον απαιτούμενο χρόνο έρευνας για τις νέες καινοτομίες.
- Είναι δυνατή η στοχευμένη εισαγωγή υπηρεσιών με κριτήριο την γεωγραφική περιοχή ή τα υποσύνολα των πελατών. Οι υπηρεσίες μπορούν να κλιμακωθούν, ή το αντίθετο, γρήγορα και ανάλογα με τις απαιτήσεις.
- *Άνοιγμα* της αγοράς εικονικών συσκευών σε απλούς προγραμματιστές λογισμικού, μικρές εταιρίες και ακαδημαϊκούς, ενθαρρύνοντας έτσι περισσότερες καινοτομίες.
- Η δυνατότητα εκτέλεσης των production, test και reference versions πάνω στην ιδιά υποδομή παρέχει μια πιο αποτελεσματική οδό για την υλοποίηση νέων τμημάτων δικτύου.
- Βελτιστοποίηση της διαμόρφωσης ή/και τοπολογίας του δικτύου σε σχεδόν πραγματικό χρόνο με βάση τα πραγματικά μοτίβα επισκεψιμότητας / κινητικότητας και χρήσης των υπηρεσιών. Για παράδειγμα, η βελτιστοποίηση και η ανάθεση πόρων στις λειτουργίες του δικτύου αυτόματα και σε σχεδόν πραγματικό χρόνο θα μπορούσε να παρέχει προστασία από αστοχίες χωρίς την υλοποίηση πραγματικής 1+1 ασφάλειας.

2. Software Defined Networking

Η ***Δικτύωση Καθοριζομένη από Λογισμικό***, σε συντομογραφία ***SDN***, έχει πρόσφατα γίνει ένα από τα πιο δημοφιλή θέματα στο τον τομέα της πληροφορικής. Σε αυτήν την ενότητα, θα παρουσιάσουμε πρώτα έναν γενικά αποδεκτό ορισμό του SDN και, στη συνέχεια, θα περιγράψουμε ένα σύνολο βασικών πλεονεκτημάτων και προκλήσεων.

2.1. Ορισμός της Δικτύωσης Καθοριζομένης από Λογισμικό

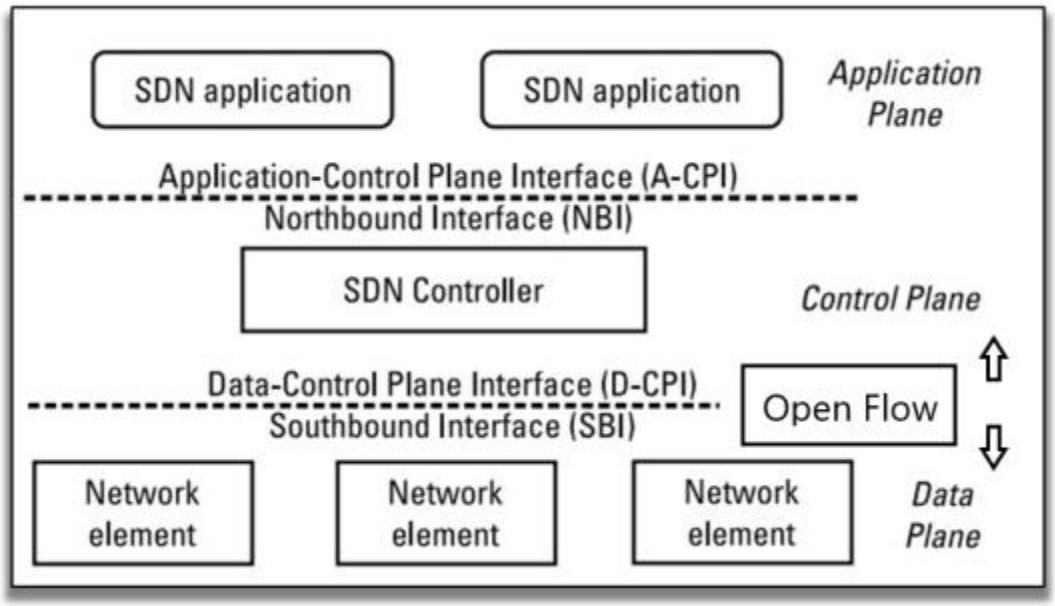
Σύμφωνα με τον **Open Networking Foundation** (μια μη κερδοσκοπική κοινοπραξία αφιερωμένη στην ανάπτυξη, την τυποποίηση και την εμπορευματοποίηση του SDN) το SDN ορίζεται ως: μια αναδυόμενη αρχιτεκτονική δικτύου όπου ο έλεγχος του δικτύου αποσυνδέεται από την προώθηση και είναι άμεσα προγραμματιζόμενος. Σύμφωνα με αυτόν τον ορισμό, το SDN ορίζεται από δύο χαρακτηριστικά, δηλαδή την *αποσύνδεση* των επιπέδων **ελέγχου και δεδομένων** και το *προγραμματιζόμενο* **επίπεδο ελέγχου** [6].

Συγκεκριμένα, το SDN προσφέρει απλές προγραμματιζόμενες συσκευές δικτύου αντί των πιο περίπλοκων υπαρχόντων συσκευών. Επιπλέον, το SDN αγορεύει τον διαχωρισμό του επιπέδων ελέγχου και δεδομένων στον αρχιτεκτονικό σχεδιασμό του δικτύου έτσι ώστε ο έλεγχος δικτύου να γίνεται ξεχωριστά στο επίπεδο ελέγχου χωρίς να επηρεάζεται η ροή δεδομένων. Ο διαχωρισμός αυτός έχει ως αποτέλεσμα τα switches να γίνουν *ανόητες* συσκευές προώθησης, με τη λογική ελέγχου να εφαρμόζεται από έναν κεντρικό ελεγκτή. Αυτό όχι μόνο επιτρέπει στους διαχειριστές δικτύου πολύ πιο λεπτομερή έλεγχο της ροής της κυκλοφορίας, αλλά τους δίνει επίσης τη δυνατότητα να ανταποκρίνονται με πιο αποτελεσματικό τρόπο στις μεταβαλλόμενες απαιτήσεις ενός δυναμικού περιβάλλοντος [7].

2.2. Θεμελιώδεις ιδέες του SDN

Προκειμένου να επιτευχθεί ο στόχος του, το SDN διαχωρίζει τις λειτουργίες ελέγχου και τις λειτουργίες προώθησης δεδομένων και μεταφέρει τον έλεγχο δικτύου σε ένα ειδικό στοιχείο που ονομάζεται **ελεγκτής SDN** (Controller) [1] . Ο ελεγκτής SDN παρέχει μια προσέγγιση για τον έλεγχο και τη διαχείριση των πόρων δικτύου μέσω λογισμικού, ή αλλιώς τις **εφαρμογές SDN** (Applications). Επομένως, τα βασικά στοιχεία του SDN μπορούν να οργανωθούν σε τρεις ομάδες: το **επίπεδο δεδομένων** (*Data Plane*), το **επίπεδο ελέγχου** (*Control Plane*) και το **επίπεδο εφαρμογής** (*Application Plane*) όπως φαίνεται στην Εικόνα 2.

Συνοπτικά, το **επίπεδο δεδομένων** περιλαμβάνει τους κατανεμημένους πόρους δικτύου που εκτελούν τις λειτουργίες επεξεργασίας και μεταφοράς δεδομένων. Τα *network elements* του επιπέδου δεδομένων εκθέτουν τις δυνατότητές τους και τις καταστάσεις των πόρων τους στο **επίπεδο ελέγχου** μέσω μιας τυπικής διεπαφής, από οπού και ελέγχονται άμεσα οι συμπεριφορές τους. Οι εφαρμογές SDN καθορίζουν τις απαιτήσεις της δικτύωσής τους στον ελεγκτή και τις λειτουργίες των αφηρημένων πόρων δικτύου μέσω της διεπαφής αυτής. Ο ελεγκτής *SDN* μεταφράζει τις **απαιτήσεις** των εφαρμογών σε οδηγίες ελέγχου **χαμηλού επιπέδου** που ενδέχεται να εκτελούνται από τα στοιχεία δικτύου στο επίπεδο δεδομένων.



Εικόνα 2: Μια γενική εικόνα της αρχιτεκτονικής SDN.

Πηγή : [1] p.33

2.2.1. Διαχωρισμός των Επίπεδων

Το πρώτο θεμελιώδες χαρακτηριστικό του SDN είναι ο **διαχωρισμός των επιπέδων προώθησης (δεδομένων) και ελέγχου** [8]. Η λειτουργία προώθησης, συμπεριλαμβανομένης της λογικής και των πινάκων για την επιλογή του τρόπου αντιμετώπισης των εισερχόμενων πακέτων, με βάση χαρακτηριστικά όπως διεύθυνση MAC, διεύθυνση IP και αναγνωριστικό VLAN, βρίσκονται στο επίπεδο προώθησης. Οι θεμελιώδεις ενέργειες που εκτελούνται από το επίπεδο προώθησης μπορούν να περιγραφούν από τον τρόπο με τον οποίο απαλλάσσεται από τα πακέτα που φθάνουν.

Μπορεί να προωθήσει, να ρίξει, να καταναλώσει ή να αντιγράψει ένα εισερχόμενο πακέτο (**forward, drop, consume, replicate**), ενώ μπορεί επίσης να μετασχηματίσει το πακέτο με κάποιο τρόπο πριν αναλάβει περαιτέρω δράση. Για βασική προώθηση, η συσκευή καθορίζει τη σωστή θύρα εξόδου πραγματοποιώντας μια αναζήτηση στον πίνακα διευθύνσεων στο ASIC (application-specific integrated circuit) . Για παράδειγμα ένα πακέτο μπορεί να πέσει λόγω των συνθηκών υπερχείλισης του buffer ή λόγω ειδικού φιλτραρίσματος που προκύπτει από μια λειτουργία περιορισμού ρυθμού QoS. Τα πακέτα ειδικής περίπτωσης που απαιτούν επεξεργασία από τα επίπεδα ελέγχου ή διαχείρισης απορροφούνται και μεταφέρονται στο κατάλληλο επίπεδο. Τέλος, μια ειδική περίπτωση προώθησης αφορά το multicast, όπου το εισερχόμενο πακέτο πρέπει να αναπαραχθεί πριν προωθήσει τα διάφορα αντίγραφα σε διαφορετικές θύρες εξόδου. Η λογική και οι αλγόριθμοι που χρησιμοποιούνται για τον προγραμματισμό του επιπέδου προώθησης βρίσκονται στο επίπεδο ελέγχου. Πολλά από αυτά τα πρωτόκολλα και αλγόριθμοι απαιτούν ολική γνώση του δικτύου. Το επίπεδο ελέγχου καθορίζει τον τρόπο προγραμματισμού ή διαμόρφωσης των πινάκων προώθησης και της λογικής του επιπέδου δεδομένων. Δεδομένου

ότι σε ένα παραδοσιακό δίκτυο κάθε συσκευή έχει το δικό της επίπεδο ελέγχου, το πρωταρχικό καθήκον αυτού του επιπέδου ελέγχου SDN είναι να εκτελεί πρωτόκολλα δρομολόγησης ή εναλλαγής έτσι ώστε όλοι οι κατανεμημένοι πίνακες προώθησης των συσκευών του δικτύου να παραμένουν συγχρονισμένοι. Το κύριο αποτέλεσμα αυτού του συγχρονισμού είναι η πρόληψη των βρόχων. Ενώ παραδοσιακά τα προαναφερθέντα επίπεδα θεωρούνται λογικά ξεχωριστά, συνυπάρχουν στους υπάρχουν μεταγωγείς. Στο SDN, το επίπεδο ελέγχου μετακινείται από τους μεταγωγείς σε έναν **κεντρικό ελεγκτή**, όπως φαίνεται στην Εικόνα 2.

2.2.2. Κεντρικός έλεγχος & απλοποιημένες συσκευές

Με βάση την ιδέα του διαχωρισμού των επιπέδων προώθησης και ελέγχου, το επόμενο χαρακτηριστικό είναι η απλοποίηση των συσκευών, οι οποίες στη συνέχεια ελέγχονται από ένα κεντρικό σύστημα που εκτελεί το **λογισμικό διαχείρισης και ελέγχου** [8]. Αντί για εκατοντάδες χιλιάδες γραμμές περίπλοκου κώδικα για το επιπέδου ελέγχου σε κάθε συσκευή, δημιουργείται ένας κεντρικός ελεγκτής οπού υπάρχει όλο το λογισμικό. Αυτός ο ελεγκτής που βασίζεται σε λογισμικό μπορεί στη συνέχεια να διαχειριστεί το δίκτυο βάσει πολιτικών υψηλότερου επιπέδου. Ο ελεγκτής παρέχει απλοϊκές οδηγίες στις απλουστευμένες συσκευές, όταν χρειάζεται, ώστε να τους επιτρέπει να λαμβάνουν γρήγορες αποφάσεις σχετικά με τον τρόπο αντιμετώπισης των εισερχόμενων πακέτων.

2.2.3. Προγραμματισμός υπηρεσιών δικτύου:

Ο κεντρικοποιημένος software-based ελεγκτής SDN παρέχει μια ανοιχτή διεπαφή που κάνει εφικτό τον αυτοματοποιημένο έλεγχο του δικτύου. Στο πλαίσιο του **Open SDN**, οι όροι **northbound** και **southbound** χρησιμοποιούνται συχνά για να διευκρινιστεί εάν η διεπαφή προορίζεται για τις εφαρμογές ή τις συσκευές. Αυτοί οι όροι προέρχονται από το γεγονός ότι στα περισσότερα διαγράμματα οι εφαρμογές απεικονίζονται από πάνω (δηλαδή, προς τα βόρεια) του ελεγκτή, ενώ οι συσκευές απεικονίζονται παρακάτω (δηλ. προς τα νότια) του ελεγκτή. Ένα παράδειγμα southbound API είναι η διεπαφή **OpenFlow** που χρησιμοποιεί ο ελεγκτής για τον προγραμματισμό των συσκευών δικτύου [8]. Το northbound API επιτρέπει στις εφαρμογές λογισμικού να συνδέονται με τον ελεγκτή έτσι ώστε να παρέχουν ανεξάρτητα τους αλγόριθμους και τα πρωτόκολλα που μπορούν να χρησιμοποιηθούν για την αποτελεσματική λειτουργία του δικτύου. Αυτές οι εφαρμογές μπορούν γρήγορα και δυναμικά να πραγματοποιήσουν αλλαγές στο δίκτυο, ανάλογα με τις εκάστοτε ανάγκες. Το northbound API του ελεγκτή προορίζεται να παρέχει *αφαίρεση* των συσκευών δικτύου και της τοπολογίας. Υπάρχουν τρία βασικά οφέλη που μπορεί να αντλήσει ο προγραμματιστής εφαρμογών από το northbound API:

- Χρησιμοποιεί σύνταξη που είναι πιο οικεία στους προγραμματιστές (π.χ., REST ή JSON που είναι πιο βολικές συντακτικά από ό,τι τα **Type Length Value** (TLVs).
- Παρέχει μια αφαίρεση της τοπολογίας του δικτύου και του επιπέδου δικτύου επιτρέποντας στον προγραμματιστή εφαρμογών να χειρίζεται το δίκτυο στο σύνολό του και όχι μεμονωμένα τους κόμβους.
- Παρέχει αφαίρεση των ίδιων των πρωτοκόλλων δικτύου, αποκρύπτοντας τον προγραμματιστή της εφαρμογής από τις λεπτομέρειες του OpenFlow ή του BGP.

Με αυτόν τον τρόπο, μπορούν να αναπτυχθούν εφαρμογές που λειτουργούν σε μια μεγάλη γκάμα εξοπλισμού κατασκευαστών που ενδέχεται να διαφέρουν σημαντικά στις λεπτομέρειες εφαρμογής τους.

Ένα από τα αποτελέσματα αυτού του επιπέδου αφαίρεσης είναι ότι παρέχει τη δυνατότητα εικονικοποίησης του δικτύου, αποσυνδέοντας την υπηρεσία δικτύου από το υποκείμενο φυσικό δίκτυο. Αυτές οι υπηρεσίες εξακολουθούν να παρουσιάζονται σε συσκευές φιλοξενίας με τέτοιο τρόπο ώστε να μην γνωρίζουν ότι οι πόροι δικτύου που χρησιμοποιούν είναι εικονικοί και όχι οι φυσικοί για τους οποίους είχαν αρχικά σχεδιαστεί.

2.2.4. Προσιτότητα

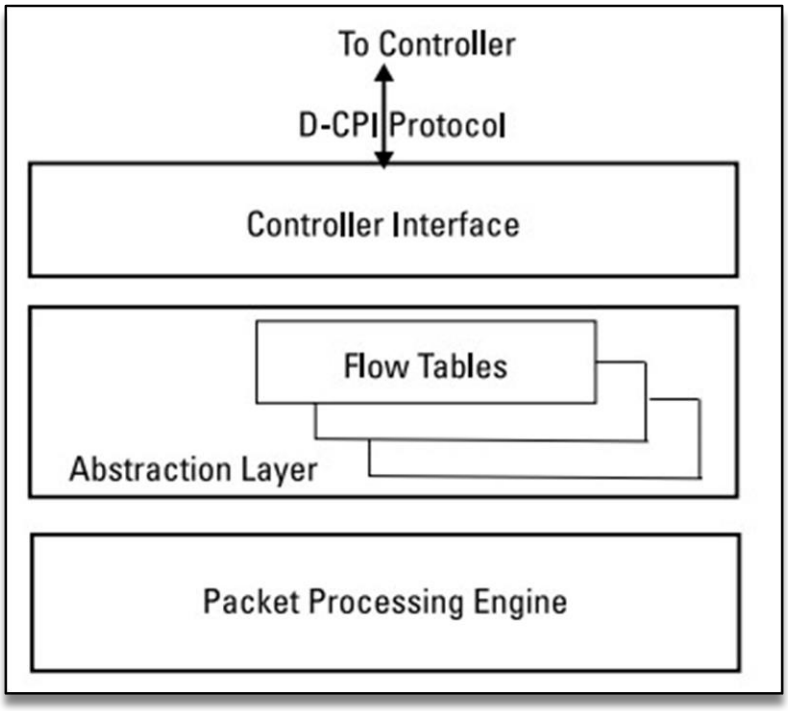
Ένα χαρακτηριστικό του SDN είναι ότι οι διεπαφές του θα πρέπει να παραμείνουν **τυποποιημένες, καλά τεκμηριωμένες** και **όχι ιδιόκτητες** [8]. Τα οριζόμενα API θα πρέπει να παρέχουν επαρκές έλεγχο στο λογισμικό για τον έλεγχο των διαφόρων παραμέτρων του επιπέδου ελέγχου και για πειραματισμό. Η παραδοχή είναι ότι διατηρώντας ανοιχτές τόσο τις northbound όσο και τις southbound συνδέσεις με τον ελεγκτή SDN, θα επιτραπεί η έρευνα για νέες και καινοτόμες μεθόδους λειτουργίας των δικτυών. Τα ερευνητικά ιδρύματα καθώς και οι επιχειρηματίες μπορούν να επωφεληθούν από αυτήν την ικανότητα προκειμένου να πειραματιστούν και να δοκιμάσουν εύκολα νέες ιδέες. Ως εκ τούτου, η ταχύτητα με την οποία αναπτύσσονται και παρατάσσονται οι νέες τεχνολογίες αυξάνεται σημαντικά, καθώς μια πολύ μεγαλύτερη ομάδα ατόμων και οργανισμών είναι σε θέση να βοηθήσει στην επίλυση των προβλημάτων του σήμερα. Αποτέλεσμα είναι μια καλύτερη και ταχύτερη τεχνολογική πρόοδος στη δομή και τη λειτουργία των δικτύων. Η παρουσία αυτών των ανοιχτών διεπαφών ενθαρρύνει επίσης τα έργα ανοιχτού κώδικα που σχετίζονται με το SDN, ενώ εκτός από τη διευκόλυνση της έρευνας και του πειραματισμού, οι ανοιχτές διεπαφές επιτρέπουν την διαλειτουργικότητα του εξοπλισμού διαφορετικών προμηθευτών. Αυτό συνήθως παράγει ένα ανταγωνιστικό περιβάλλον που μειώνει το κόστος για τους καταναλωτές, ένας στόχος που έχει υπάρξει κομμάτι της ατζέντας του SDN από την ίδρυση του.

2.3. Επίπεδο Δεδομένων

Το **επίπεδο δεδομένων** περιλαμβάνει τους **μεταγωγείς SDN**, δηλαδή τα στοιχεία δικτύου που εκτελούν την προώθηση και την επεξεργασία πακέτων, αλλά και τις λειτουργίες του τείχους προστασίας, ελέγχου πρόσβασης και ισορροπίας φορτίου. Το OpenFlow είναι σήμερα το de facto πρότυπο της διεπαφής southbound για τον έλεγχο των μεταγωγέων SDN [1].

2.3.1. Δομικά στοιχεία ενός μεταγωγέα SDN

Η δομή υψηλού επιπέδου ενός μεταγωγέα SDN, όπως φαίνεται στην Εικόνα 3, αποτελείται από τρία βασικά στοιχεία - τη **διεπαφή του ελεγκτή**, ένα **στρώμα αφαίρεσης** (abstraction layer) και τον **μηχανισμό επεξεργασίας πακέτων** (packet processing engine).



Εικόνα 3: Δομή μεταγωγέα SDN

Πηγή: [1] p.42

Η διεπαφή του ελεγκτή διατηρεί ένα ασφαλές κανάλι επικοινωνίας και εφαρμόζει ένα πρωτόκολλο D-CPI (Data–Control Plane Interface π.χ. **OpenFlow**) για την υποστήριξη της επικοινωνίας μεταξύ του μεταγωγέα και του ελεγκτή SDN [1].Ουσιαστικά, η διεπαφή παρέχει ένα API που επιτρέπει στον ελεγκτή να ελέγχει τις λειτουργίες που εκτελούνται στον μεταγωγέα εγκαθιστώντας και ενημερώνοντας τους κανόνες match-action του μεταγωγέα, ενώ επιτρέπει επίσης στον ελεγκτή να συλλεγεί πληροφορίες για την κατάσταση της συσκευής.

Το **επίπεδο αφαίρεσης** βρίσκεται μεταξύ της διεπαφής ελεγκτή και της μηχανής επεξεργασίας πακέτων. Αυτό το επίπεδο παρέχει μια αφηρημένη όψη του μηχανισμού επεξεργασίας πακέτων, πάνω στην οποία ο ελεγκτής μπορεί να προγραμματίσει τις λειτουργίες του μεταγωγέα δίχως να γνωρίζει τις λεπτομέρειες εκτελέσεως του μηχανισμού επεξεργασίας. Το στρώμα αφαίρεσης διατηρεί έναν ή περισσότερους πίνακες ροής για να αποθηκεύσει τους κανόνες αντιστοίχισης πακέτων που λαμβάνονται από τον ελεγκτή.

Ο **μηχανισμός επεξεργασίας πακέτων** ενός μεταγωγέα SDN εκτελεί τις ενέργειες προώθησης και επεξεργασίας πακέτων [1]. Αυτό το στοιχείο περιλαμβάνει ένα σύνολο θυρών εισόδου, ένα σύνολο θυρών εξόδου και μια δομή εναλλαγής που διασυνδέει τις θύρες εισόδου με τις θύρες εξόδου. Για κάθε εισερχόμενο πακέτο που λαμβάνεται σε μια θύρα εισόδου, ο μηχανισμός επεξεργασίας προσπαθεί να προσδιορίσει την καταχώρηση ροής για το πακέτο αυτό αναζητώντας στον πίνακα ροής. Οι οδηγίες που αποθηκεύονται στην αντιστοιχισμένη καταχώριση ροής καθορίζουν τις ενέργειες που πρέπει να εκτελέσει ο μηχανισμός επεξεργασίας στο πακέτο. Οι τυπικές ενέργειες περιλαμβάνουν την προώθηση του πακέτου σε μια θύρα

εξόδου (ή μια συγκεκριμένη ουρά σε μια θύρα εξόδου), προώθηση του πακέτου στον ελεγκτή, τροποποίηση μέρους του πακέτου, ή και drop του πακέτου. Εάν δεν βρεθεί αντιστοιχία στον πίνακα ροής για ένα πακέτο, τότε το πακέτο μπορεί να προωθηθεί στον ελεγκτή για περαιτέρω επεξεργασία ή να απορριφθεί από το μεταγωγέα.

2.3.2. Πίνακες Ροής

Οι πίνακες ροής είναι οι βασικές δομές δεδομένων σε μια συσκευή SDN. Αυτοί οι πίνακες ροής επιτρέπουν στη συσκευή να αξιολογήσει τα εισερχόμενα πακέτα και να προβεί στην κατάλληλη ενέργεια βάσει των περιεχομένων του πακέτου που μόλις ελήφθη [9]. Τα πακέτα παραδοσιακά έχουν παραληφθεί από τις συσκευές δικτύωσης και έχουν αξιολογηθεί με βάση ορισμένα πεδία. Ανάλογα με αυτήν την αξιολόγηση, λαμβάνονται οι αποφάσεις για τις μετέπειτα ενέργειες. Αυτές οι ενέργειες μπορεί να περιλαμβάνουν προώθηση του πακέτου σε μια συγκεκριμένη θύρα, drop του πακέτου και flood του πακέτου σε όλες τις θύρες, μεταξύ άλλων.

Μια συσκευή SDN δεν διαφέρει ουσιαστικά, εκτός από το ότι αυτή η βασική λειτουργία έχει καταστεί πιο γενική και πιο προγραμματιζόμενη μέσω των πινάκων ροής και της σχετικής λογικής τους. Οι πίνακες ροής αποτελούνται από έναν αριθμό **καταχωρήσεων ροής** (Flow Entries) με προτεραιότητα, καθεμία από τις οποίες αποτελείται συνήθως από δύο στοιχεία:

- τα **πεδία αντιστοίχισης** (Match Fields)
- τις **ενέργειες** (Actions).

Τα πεδία αντιστοίχισης χρησιμοποιούνται για τη σύγκριση των εισερχόμενων πακέτων. Ένα εισερχόμενο πακέτο συγκρίνεται με τα πεδία αντιστοίχισης με σειρά προτεραιότητας και επιλέγεται το πρώτο πλήρες ταίριασμα. Οι ενέργειες είναι οι οδηγίες που πρέπει να εκτελέσει η συσκευή δικτύου εάν ένα εισερχόμενο πακέτο ταιριάζει με τα πεδία αντιστοίχισης που καθορίζονται για αυτήν την καταχώριση ροής.

Τα πεδία αντιστοίχισης μπορούν να έχουν "*μπαλαντέρ*" (Wild Card) για πεδία που δεν σχετίζονται με τη συγκεκριμένη αντιστοίχιση. Για παράδειγμα, όταν ταιριάζουν πακέτα που βασίζονται μόνο στη διεύθυνση IP ή το υποδίκτυο, όλα τα άλλα πεδία γίνονται μπαλαντέρ. Παρομοίως, εάν ταιριάζει μόνο σε διεύθυνση MAC ή θύρα UDP / TCP τα άλλα πεδία δεν έχουν σχέση και, κατά συνέπεια, αυτά τα πεδία γίνονται μπαλαντέρ. Ανάλογα με τις ανάγκες της εφαρμογής, όλα τα πεδία μπορεί να είναι σημαντικά, οπότε σε τέτοιες περιπτώσεις δεν υπάρχουν μπαλαντέρ. Ο πίνακας ροής και οι κατασκευές εισόδου ροής επιτρέπουν στον προγραμματιστή των εφαρμογών SDN να έχει ένα ευρύ φάσμα δυνατοτήτων για την αντιστοίχιση των πακέτων και για την λήψη των κατάλληλων ενεργειών.

2.3.3. Υλοποιήσεις σε Λογισμικό και Υλικό

Οι μεταγωγείς SDN μπορούν να κατασκευαστούν σε δυο μορφές, τις βασιζόμενες σε λογισμικό (software-based) και τις βασιζόμενες σε υλικό (hardware-based), με την κάθε μια να έχεις τα δικά της πλεονεκτήματα και περιορισμούς [9]. Οι software-based υλοποιήσεις χρησιμοποιούν λογισμικό που εκτελείται σε διακομιστές εμπορικής με γενικής χρήσης CPU και λειτουργικό σύστημα (π.χ. Linux). Οι hardware-based υλοποιήσεις χρησιμοποιούν εξειδικευμένο

υλικό, όπως Content-Addressable Memories (CAMs), Ternary Content-Addressable Memories (TCAMs) και επεξεργαστές δικτύου.

Η υλοποίηση μεταγωγέων σε λογισμικό προσφέρει έναν απλό τρόπο δημιουργίας συσκευών SDN, επειδή οι πίνακες ροής, οι καταχωρήσεις ροής και τα πεδία αντιστοίχισης αντιστοιχίζονται εύκολα σε γενικές δομές δεδομένων. Οι δυο ευρέως αναγνωρισμένες υλοποιήσεις είναι οι: **Open vSwitch** (OVS) της *Nicira* και το **Indigo Virtual Switch** (IVS) της *Big Switch*. Ωστόσο, οι υλοποιήσεις λογισμικού είναι πιθανό να είναι πιο αργές και λιγότερο αποδοτικές από τις αντίστοιχες που βασίζονται σε υλικό, καθώς δεν επωφελούνται από οποιαδήποτε επιτάχυνση υλικού. Κατά συνέπεια , οι υλοποιήσεις λογισμικού ενδέχεται να μην είναι κατάλληλες για καταστάσεις που απαιτούν πολύ υψηλές ταχύτητες.

Οι hardware-based υλοποιήσεις ενδέχεται να αποδίδουν πολύ πιο γρήγορα από τις αντίστοιχες λογισμικού και επομένως να είναι πιο εφαρμόσιμες σε περιβάλλοντα ευαίσθητα ως προς την απόδοση, όπως τα δίκτυα πυρήνα (Core Network) σε κέντρα δεδομένων και το backbone των φορέων του δικτύου. Από την άλλη πλευρά όμως, η υλοποίηση ευέλικτων λειτουργιών αναζήτησης των flow tables και της αντιστοίχισης κανόνων δημιουργούν ορισμένες προκλήσεις στον σχεδιασμό. Για παράδειγμα, αν και το υλικό θα χειριστεί την αναζήτηση του πίνακα ροής πολύ πιο γρήγορα, οι πίνακες υλικού (hardware tables) έχουν περιορισμούς στον αριθμό των καταχωρήσεων ροής που μπορούν να κρατήσουν ανά πάσα στιγμή. Επιπλέον, ορισμένες ενέργειες όπως η τροποποίηση πακέτων μπορεί να είναι περιορισμένες ή ακόμη και να μην είναι εφικτές εάν αντιμετωπίζονται από την συσκευή. Επομένως, πολλοί μεταγωγείς SDN σχεδιάζονται συνδυάζοντας τεχνολογίες βασισμένες σε λογισμικό και υλικό για να εκμεταλλευτούν πλήρως τα πλεονεκτήματά τους και να ξεπεράσουν τους αντίστοιχους περιορισμούς τους.

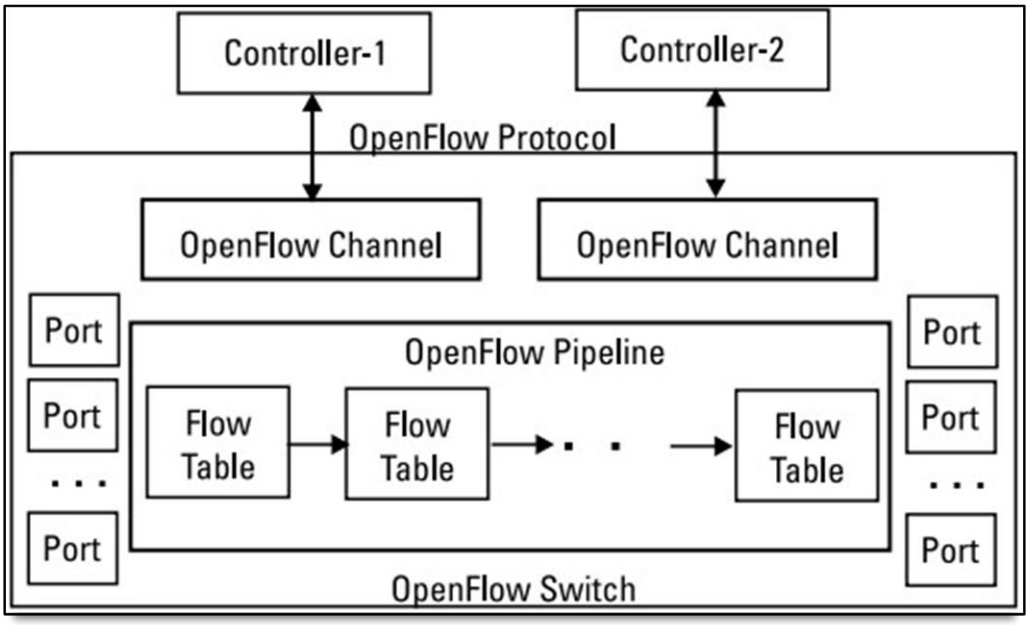
2.4. Πρωτόκολλο OpenFlow

Το **OpenFlow**, όπως ορίζεται από τον ONF [10], είναι το de facto πρότυπο για το D-CPI (**Data-Control Plane Interface,** δρα μεταξύ των επιπέδων ελέγχου και προώθησης) στην αρχιτεκτονική SDN. Η προδιαγραφή OpenFlow ορίζει τόσο το πρωτόκολλο επικοινωνίας μεταξύ του ελεγκτή SDN και των μεταγωγέων όσο και τη διαδικασία διαχείρισης των πινάκων ροής στους μεταγωγείς SDN. Όταν ένας ελεγκτής SDN και οι ελεγχόμενοι μεταγωγείς ακολουθούν την προδιαγραφή OpenFlow, αναφέρονται ως ελεγκτές OpenFlow και μεταγωγείς OpenFlow, αντίστοιχα. Αν και υπάρχουν εναλλακτικά πρωτόκολλα D-CPI, το OpenFlow είναι το μόνο μη ιδιόκτητο πρωτόκολλο γενικής χρήσης για τον προγραμματισμό μεταγωγέων SDN.

2.4.1. Αρχιτεκτονική

Η αρχιτεκτονική ενός μεταγωγέα OpenFlow φαίνεται στην Εικόνα 4, ενώ τα κυριά στοιχεία ενός τέτοιου μεταγωγέα είναι ένα σύνολο από θύρες εισόδου/εξόδου, ένα OpenFlow pipeline που περιχέει έναν η περισσοτέρους πίνακες ροής, και ένα ασφαλές κανάλι επικοινωνίας για επικοινωνία με έναν η περισσοτέρους ελεγκτές OpenFlow [7]. Όπως σε κάθε μεταγωγέα, η βασική λειτουργία ενός μεταγωγέα OpenFlow είναι να παίρνει τα πακέτα που φτάνουν στις

θύρες εισόδου και να τα προωθεί στις προορισμένες θύρες εξόδου. Μια μοναδική πτυχή του OpenFlow βρίσκεται στην επεξεργασία "αγωγού" (**Pipeline Processing**) της λειτουργίας **αντιστοίχισης πακέτων.**



Εικόνα 4: Αρχιτεκτονική ενός μεταγωγέα OpenFlow

***Πηγή:** [1] p.45*

Για κάθε ληφθέν πακέτο, ο μεταγωγέας OpenFlow θα προσδιορίσει πρώτα τη ροή στην οποία ανήκει το πακέτο αυτό και στη συνέχεια θα εκτελέσει τις οδηγίες επεξεργασίας που καθορίζονται για τη ροή αυτήν. Η αναζήτηση της αντίστοιχης ροής κάθε λαμβανόμενου πακέτου και ο καθορισμός των ενεργειών που πρέπει να γίνουν για το πακέτο είναι η βασική ευθύνη του αγωγού OpenFlow. Ο αγωγός περιέχει έναν ή περισσότερους πίνακες ροής. Κάθε καταχώριση σε έναν πίνακα ροής περιέχει πεδία αντιστοίχισης και ένα σύνολο οδηγιών. Συνοπτικά, όταν ένα πακέτο φτάνει σε ένα μεταγωγέα OpenFlow υποβάλλεται σε επεξεργασία με τον εξής τρόπο:

- Ολοκληρώνεται η αναζήτηση πίνακα ροής που προσπαθεί να αντιστοιχίσει τα πεδία της κεφαλίδας του εν λόγω πακέτου με τον τοπικό πίνακα ροής. Εάν δεν υπάρχει αντίστοιχη καταχώριση, τότε το πακέτο αποστέλλεται στον ελεγκτή για επεξεργασία. Όταν υπάρχουν πολλές καταχωρήσεις που ταιριάζουν με το εισερχόμενο πακέτο στον πίνακα ροής, επιλέγεται αυτή με την υψηλότερη προτεραιότητα.
- Τα byte και packet counters ενημερώνονται
- Οι ενέργειες που αντιστοιχούν στον κανόνα ροής προσαρτώνται στο σύνολο ενεργειών. Εάν ένας διαφορετικός πίνακας ροής είναι μέρος της αλυσίδας εκτέλεσης, τότε η επεξεργασία συνεχίζεται.
- Μετά την επεξεργασία όλων των πινάκων ροής, το σύνολο ενεργειών εκτελείται

Η προδιαγραφή OpenFlow ορίζει τις θύρες ενός μεταγωγέα OpenFlow ως τμήμα των διασυνδέσεων δικτύου για τη διέλευση των πακέτων μεταξύ του μεταγωγέα και του υπόλοιπου δικτύου. Το σύνολο των θυρών OpenFlow σε έναν μεταγωγέα ενδέχεται να μην είναι ίδιο με το σύνολο των διεπαφών που παρέχονται από το υλικό. Ορισμένες διεπαφές ενδέχεται να είναι απενεργοποιημένες για το OpenFlow, ενώ ο μεταγωγέας OpenFlow μπορεί να ορίσει πρόσθετες θύρες. Ένας μεταγωγέας OpenFlow πρέπει να υποστηρίζει τρεις τύπους θυρών: **φυσικές θύρες, λογικές θύρες** και **δεσμευμένες θύρες**. Οι φυσικές θύρες είναι θύρες καθορισμένες από τον μεταγωγέα που αντιστοιχούν στις διεπαφές του υλικού. Οι λογικές θύρες είναι "αφαιρέσεις" υψηλότερου επιπέδου που δεν αντιστοιχούν άμεσα σε κάποια διεπαφή αλλά μπορεί να αντιστοιχιστούν σε διάφορες φυσικές θύρες. Οι δεσμευμένες θύρες είναι για γενικές ενέργειες προώθησης, όπως αποστολή πακέτων στον ελεγκτή, broadcasting ή προώθηση πακέτων χρησιμοποιώντας συμβατικές μη OpenFlow μεθόδους.

2.4.2. Ασφαλές Κανάλι Επικοινωνίας

Το πρωτόκολλο OpenFlow καθορίζει τη διαδικασία ανταλλαγής μηνυμάτων μεταξύ ενός ελεγκτή και ενός μεταγωγέα μέσω του καναλιού OpenFlow [11] . Γενικά, αυτή η επικοινωνία προστατεύεται με ασύμμετρη κρυπτογράφηση που βασίζεται σε TLS, αν και επιτρέπονται μη κρυπτογραφημένες συνδέσεις TCP. Το OpenFlow υποστηρίζει τρεις τύπους μηνυμάτων: μηνύματα **ελεγκτή-προς-μεταγωγέα, ασύγχρονα** μηνύματα και **συμμετρικά** μηνύματα.

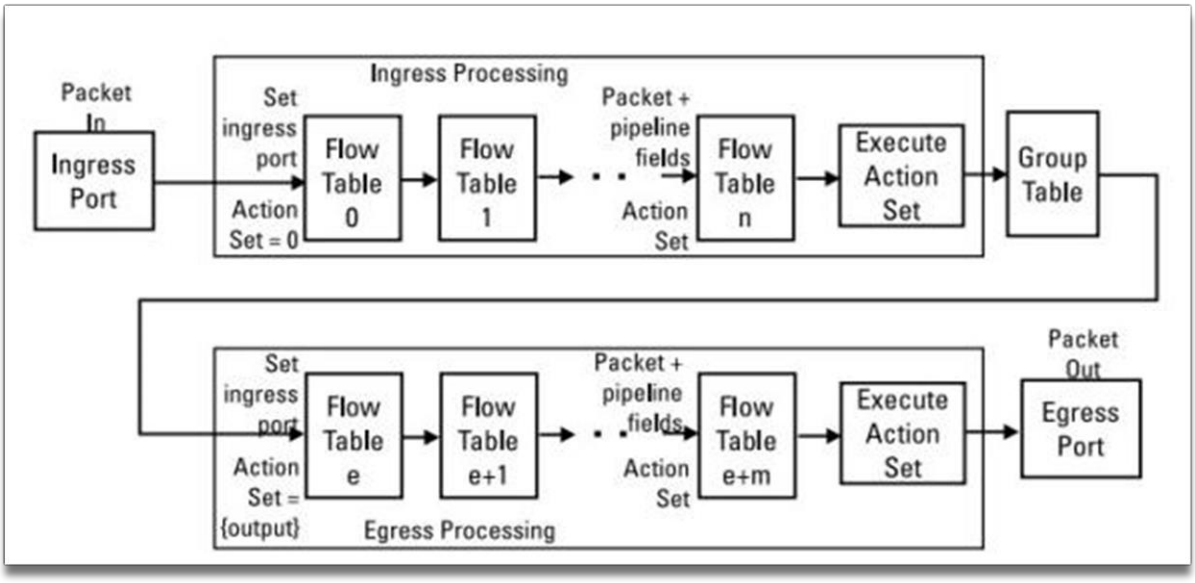
Τα μηνύματα **ελεγκτή-προς-μεταγωγέα** ξεκινούν από τον ελεγκτή και ενδέχεται να απαιτούν απόκριση από τον μεταγωγέα. Αυτός ο τύπος μηνυμάτων χρησιμοποιείται από τον ελεγκτή για την απευθείας παρακολούθηση και διαχείριση της κατάστασης του μεταγωγέα.

Τα ασύγχρονα μηνύματα ξεκινούν από τον μεταγωγέα δίχως να ζητείται από τον ελεγκτή. Αυτός ο τύπος μηνυμάτων χρησιμοποιείται από τον μεταγωγέα για να ειδοποιεί τον ελεγκτή σχετικά με πιθανά συμβάντα και για αλλαγές στις καταστάσεις των μεταγωγέων. Τα ληφθέντα πακέτα του μεταγωγέα μπορούν να προωθηθούν στον ελεγκτή χρησιμοποιώντας ασύγχρονα μηνύματα για περαιτέρω επεξεργασία.

Συμμετρικά μηνύματα μπορούν να αποστέλλονται είτε από το μεταγωγέα είτε από τον ελεγκτή χωρίς να έχει ζητηθεί από τον άλλο. Αυτός ο τύπος μηνυμάτων χρησιμοποιείται κυρίως για αρχικοποίηση μετά την εγκατάσταση του καναλιού OpenFlow ή για τον τακτικό έλεγχο της κατάστασης του καναλιού.

2.4.3. OpenFlow Pipeline Processing

Το **OpenFlow Pipeline Processing**, εν συντομία **OPP**, καθορίζει τον τρόπο αλληλεπίδρασης των πακέτων ενός αγωγού OpenFlow με τους πίνακες ροής του αγωγού αυτού. Στην Εικόνα 5 εμφανίζεται ένα διάγραμμα που απεικονίζει την ακολουθία ροής ενός πακέτου. Όπως φαίνεται στο σχήμα, το OPP αποτελείται από δύο στάδια: την **επεξεργασία εισόδου** και την **επεξεργασία εξόδου** [11]. Ένας μεταγωγέας OpenFlow απαιτείται να έχει τουλάχιστον έναν πίνακα ροής εισόδου και μπορεί προαιρετικά να έχει επιπλέον πίνακες εισόδου και εξόδου. Στην περίπτωση που ο μεταγωγέας έχει μόνο έναν πίνακα ροής , το OPP απλοποιείται ως διαδικασία αναζήτησης ενός πίνακα.



Εικόνα 5: Ροή επεξεργασίας πακέτων μέσω αγωγού OpenFlow

Πηγή: *[11]* ***p.19***

Κάθε πίνακας ροής περιλαμβάνει μια λίστα καταχωρίσεων ροής. Τα κύρια στοιχεία μιας καταχώρησης του πίνακα ροής που ορίζονται στην προδιαγραφή OpenFlow παρατίθενται στη συνέχεια :

- **Match fields:** Χρησιμοποιείται ως κριτήριο για να προσδιοριστεί αν ένα εισερχόμενο πακέτο ταιριάζει με αυτήν την καταχώριση. Τα λεπτομερή περιεχόμενα των πεδίων αντιστοίχισης που ορίζονται από την τρέχουσα προδιαγραφή OpenFlow παρατίθενται στην Εικόνα 8.
- **Priority:** Καθορίζει την αντίστοιχη προτεραιότητα της καταχώρησης ροής, μια καταχώριση υψηλότερης προτεραιότητας στον πίνακα θα αντιστοιχιστεί πριν από μια καταχώριση χαμηλότερης προτεραιότητας.
- **Counters:** Παρακολουθεί τα στατιστικά στοιχεία σχετικά με τη ροή που ταιριάζει με αυτήν την καταχώριση.
- **Instructions:** Εκτελείται όταν ένα πακέτο ταιριάζει με αυτήν την καταχώριση, η εκτέλεση των οδηγιών μπορεί να οδηγήσει σε τροποποίηση του περιεχομένου του πακέτου, ενημέρωση του συνόλου ενεργειών που σχετίζεται με το πακέτο και προσαρμογή του Pipeline Process Sequence του πακέτου.
- **Timeouts:** Καθορίζει το μέγιστο χρονικό διάστημα ή το χρόνο αδράνειας πριν από τη εκπνοή της ροής από το μεταγωγέα.
- **Cookie:** Αδιαφανή τιμή δεδομένων που επιλέγεται από τον ελεγκτή, η οποία μπορεί να χρησιμοποιηθεί από τον ελεγκτή για να φιλτράρει τις καταχωρήσεις ροής που επηρεάζονται από τα στατιστικά στοιχεία ροής, τις τροποποίηση ροής και τα αιτήματα διαγραφής ροής.
- **Flags:** Χρησιμοποιείται για να αλλάξει τον τρόπο διαχείρισης καταχωρήσεων ροής.

Στην Εικόνα 6 παρατίθενται τα πεδία της ροής αντιστοίχισης που ορίζονται στην έκδοση OpenFlow 1.5.1 [11].

Flow Match Fields Defined in OpenFlow Specification Version 1.5.1	
Switch input port	ARP source IPv4 address
Switch physical input port	ARP target IPv4 address
Metadata passed between tables	ARP source hardware address
Ethernet destination address	ARP target hardware address
Ethernet source address	IPv6 source address
Ethernet frame type	IPv6 destination address
VLAN id	IPv6 flow label
VLAN priority	ICMPv6 type
IP DSCP (6 bits in ToS field)	ICMPv6 code
IP ECN (2 bits in ToS field	Target address for ND
IP protocol	Source link-layer for ND
IPv4 source address	Target link-layer for ND
IPv4 destination address	MPLS label
TCP source port	MPLS TC
TCP destination port	MPLS BoS bit
UDP source port	PBB I-SID
UDP destination port	Logical port metadata
SCTP source port	IPv6 extension header pseudo-field
SCTP destination port	PBB UCA header field
ICMP type	TCP flags
ICMP code	Output port from action set metadata
ARP opcode	Packet type value

Εικόνα 6: OpenFlow Flow Match Fields

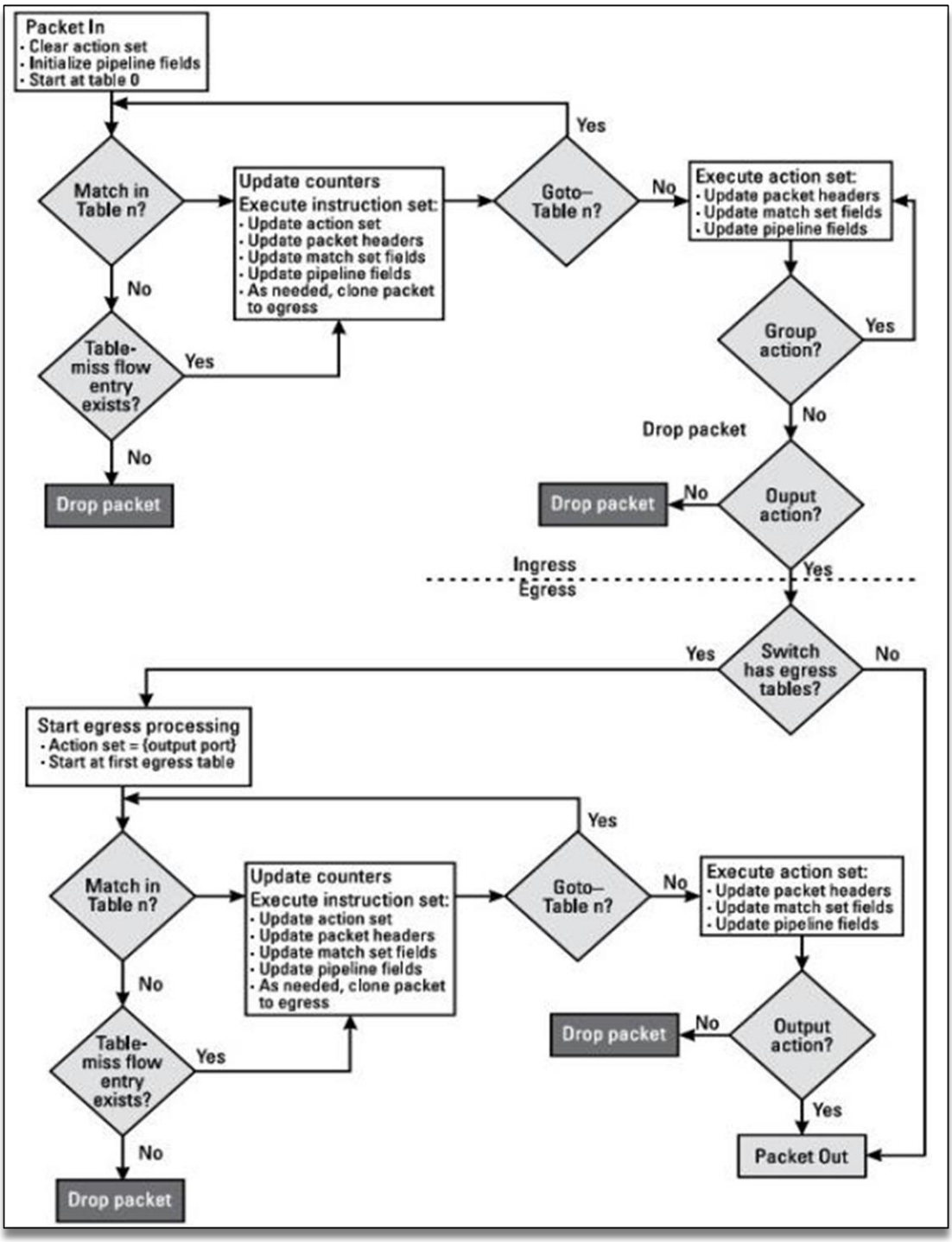
Πηγή: [1] p.63

Ένα πακέτο ταιριάζει με μια καταχώριση ροής εάν όλα τα πεδία αντιστοίχισης της καταχώρησης ταιριάζουν με τα αντίστοιχα πεδία κεφαλίδας και τα πεδία αγωγού του πακέτου. Τα πεδία αντιστοίχισης της ροής εισόδου μπορεί να είναι wildcards χρησιμοποιώντας ένα bitmask, πράγμα που σημαίνει ότι οποιαδήποτε τιμή ταιριάζει με τα unmasked bits θα αντιστοιχίζεται. Το **OpenFlow Extensible Match** (OXM) προσφέρει μια γενική και επεκτάσιμη δυνατότητα αντιστοίχισης πακέτων. Το OXM καθορίζει ένα σύνολο ζευγών type-length-value (TLV) που μπορούν να περιγράψουν σχεδόν οποιοδήποτε από τα πεδία κεφαλίδας πακέτου που θα χρειαστεί να χρησιμοποιήσει ένας μεταγωγέας OpenFlow για αντιστοίχιση.

Κατά την επεξεργασία από έναν πίνακα ροής, το πακέτο ταιριάζει με τις καταχωρήσεις ροής του πίνακα. Για να γίνει αναζήτηση μιας αντιστοιχισμένης καταχώρισης, τα πεδία κεφαλίδας των πακέτων εξάγονται και τα πεδία αγωγού των πακέτων ανακτώνται. Ανάλογα με τον τύπο του πακέτου, διάφορα πεδία της κεφαλίδας των πακέτων μπορούν να χρησιμοποιηθούν για αναζήτηση, όπως Ethernet source/destination ή IPv4 source/destination. Εκτός από τις κεφαλίδες των πακέτων, η αντιστοίχιση μπορεί επίσης να πραγματοποιηθεί με τη θύρα εισόδου, το πεδίο metadata και άλλα πεδία του αγωγού.

Οι πίνακες ροής σε έναν αγωγό OpenFlow αριθμούνται ξεκινώντας από το 0 με τη σειρά που μπορούν να περαστούν από τα πακέτα. Η επεξεργασία για κάθε πακέτο ξεκινά με την επεξεργασία εισόδου για να ταιριάξει το πακέτο με τις καταχωρήσεις ροής του flow table 0, ενώ μπορούν να χρησιμοποιηθούν και άλλοι πίνακες ανάλογα με το αποτέλεσμα που αντιστοιχεί στον πρώτο. Εάν υπάρχει οδηγία Goto-Table-N, οπού n ο αριθμός πίνακα, τότε η επεξεργασία του πακέτου αυτού θα μεταφερθεί στο table-N στο επόμενο βήμα.

Στην Εικόνα 7 φαίνεται ένα απλοποιημένο διάγραμμα ροής που απεικονίζει την επεξεργασία OpenFlow κατά τη μεταφορά ενός πακέτου μέσω ενός μεταγωγέα OpenFlow.



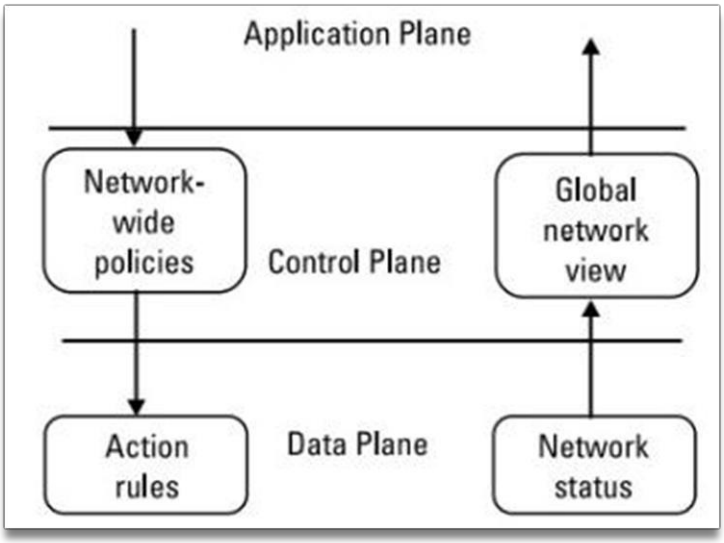
Εικόνα 7: Απλοποιημένο διάγραμμα ροής ενός μεταγωγέα OpenFlow

Πηγή: [11] p.23

2.5. Επίπεδο Ελέγχου

Το επίπεδο ελέγχου **γεφυρώνει το data plane με το application plane**, αποτελώντας έτσι βασικό στοιχείο της αρχιτεκτονική SDN για την επίτευξη του διαχωρισμού μεταξύ των λειτουργιών ελέγχου, διαχείρισης δικτύου και προώθησης δεδομένων [9]. Οι αρμοδιότητες του ελεγκτή περιλαμβάνουν: διατήρηση μιας ολοκληρωτικής άποψης του δικτύου, εφαρμογή των network policies, έλεγχος των συσκευών SDN της υποδομής του δικτύου και παροχή ενός Northbound API για τις εφαρμογές SDN. Ο ελεγκτής εφαρμόζει αποφάσεις πολιτικής σχετικά με τη δρομολόγηση, προώθηση, ανακατεύθυνση, εξισορρόπηση φορτίου κ.λπ., ενώ αυτές οι δηλώσεις αναφέρονται τόσο στον ελεγκτή όσο και στις εφαρμογές που κάνουν χρήση αυτού του ελεγκτή. Οι ελεγκτές συχνά έχουν το δικό τους set κοινών εφαρμογών, όπως μεταγωγέα εκμάθησης, δρομολογητή, τείχος προστασίας και εξισορροπητή φορτιού.

Το επίπεδο ελέγχου παίζει παρόμοιο ρόλο με το λειτουργικό σύστημα ενός υπολογιστή, το οποίο παρέχει ένα επίπεδο αφαίρεσης πάνω από τους πόρους του υλικού και επιτρέπει τον προγραμματισμό των λειτουργιών τους μέσο ενός API. Επομένως, το επίπεδο ελέγχου SDN αναφέρεται συχνά και ως ***λειτουργικό σύστημα δικτύου*** (NOS).



Εικόνα 8: Οι δύο κατευθύνσεις επεξεργασίας στο επίπεδο ελέγχου SDN

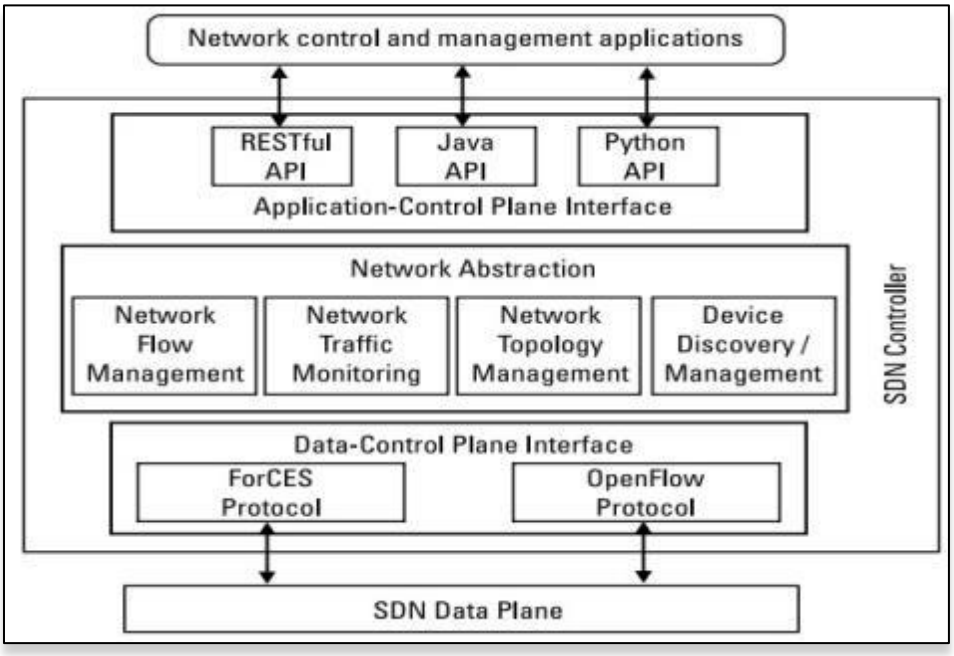
Πηγή: [1] p.68

Όπως φαίνεται στην Εικόνα 8, οι βασικές λειτουργίες του επιπέδου ελέγχου μπορούν να κατηγοριοποιηθούν σε δύο κατευθύνσεις επεξεργασίας. Η ανοδική κατεύθυνση περιλαμβάνει λειτουργίες για τη **συλλογή και τη σύνθεση της κατάστασης δικτύου**, τη **διαχείριση πληροφοριών τοπολογίας δικτύου** και την **παρουσίαση μιας παγκόσμιας εικόνας** του δικτύου στο επίπεδο εφαρμογών. Η καθοδική κατεύθυνση επεξεργασίας μεταφράζει τα αιτήματα των εφαρμογών, τα οποία συνήθως καθορίζουν τις πολιτικές της λειτουργίας δικτύου, σε κανόνες δράσης (action rules) για την επεξεργασία πακέτων στο επίπεδο δεδομένων. Οι κύριες λειτουργίες προς αυτήν την κατεύθυνση περιλαμβάνουν τη δημιουργία, εγκατάσταση και ενημέρωση των action rules σε πίνακες ροής στις συσκευές του επιπέδου δεδομένων, τη

διασφάλιση της εγκυρότητας και της συνέπειας των κανόνων δράσης και τη διατήρηση μιας βάσης δεδομένων για τις ροές που διαχειρίζονται.

Η αρχιτεκτονική ενός ελεγκτή SDN φαίνεται στην Εικόνα 9. Όπως φαίνεται στο σχήμα, ένας ελεγκτής SDN έχει μια μονάδα για την πραγματοποίηση του D-CPI (**Data-Control Plane Interface**), μια μονάδα για την αφαίρεση του επιπέδου δεδομένων και μια μονάδα για την εφαρμογή του A-CPI (**Application-Control Plane Interface**). Ο διαχωρισμός στις λειτουργίες τους έχει ως εξής [9] [1]:

- Το **Southbound API** χρησιμοποιείται για τη διασύνδεση με τις συσκευές SDN επιτρέποντας στον ελεγκτή να ***επικοινωνεί***, να ***αλληλοεπιδρά*** και να ***διαχειρίζεται*** *τις συσκευές αυτές*. Το πιο συνηθισμένο API είναι το **OpenFlow** στην περίπτωση του Open SDN ενώ εναλλακτικές λύσεις είναι οι BGP/ ForCES.
- Το **Northbound API** επιτρέπει στις εφαρμογές του επιπέδου εφαρμογών να ***προγραμματίζουν*** τον ελεγκτή καθιστώντας διαθέσιμες διάφορες λειτουργίες όπως αφηρημένα μοντέλα δεδομένων (abstract data) . Επί του παρόντος, δεν υπάρχει πρότυπο για το A-CPI όπως το OpenFlow για το D-CPI, εν μέρει λόγω της ποικιλομορφίας των εφαρμογών SDN και των απαιτήσεών τους ως προς το A-CPI.
- Η **μονάδα αφαίρεσης** δημιουργεί μια ολική αφηρημένη προβολή του δικτύου του επιπέδου δεδομένων βάσει της οποίας οι εφαρμογές μπορούν να προγραμματίσουν τις λειτουργίες του δικτύου. Οι κυρίες λειτουργίες που εκτελούνται σε αυτή τη μονάδα είναι: user/network device discovery, network device management, network topology management, network traffic monitoring, and flow management.



Εικόνα 9: Αρχιτεκτονική ενός SDN Controller

Πηγή: [1] p.54

2.5.1. Λειτουργίες Ελεγκτών SDN

Ο ελεγκτής SDN αφαιρεί τις λεπτομέρειες του πρωτοκόλλου controller-to-device έτσι ώστε οι εφαρμογές να μπορούν να επικοινωνούν με τις συσκευές SDN δίχως να γνωρίζουν τις λεπτομέρειες τις εκάστοτε συσκευής. Ο ελεγκτής παρέχει επίσης κάποιες βασικές λειτουργίες στις πρωτογενείς διεπαφές, όπως [9]:

- **Network Topology Management:** εντοπισμός των διαθέσιμων συσκευών δικτύου και έλεγχος της κατάστασης των συνδέσεων μεταξύ των συσκευών αυτών.
- **Network Traffic Monitoring:** εκτός από τη διαχείριση της τοπολογίας, η οποία παρουσιάζει μια σχετικά στατική άποψη της υποδομής του επιπέδου δεδομένων, το επίπεδο ελέγχου SDN εκτελεί επίσης παρακολούθηση κίνησης που αντικατοπτρίζει τη δυναμική πτυχή της κατάστασης του δικτύου. Η παρακολούθηση δικτύου συλλεγεί στατιστικά στοιχεία για το φόρτο κίνησης και τις ενέργειες προώθησης πακέτων (π.χ. packet number, data size, duration time).
- **Flow Management:** η διαχείριση ροής ενός ελεγκτή SDN είναι υπεύθυνη για τη δημιουργία και τη συντήρηση όλων των **flow paths** στο δίκτυο. Συγκεκριμένα, ο ελεγκτής δημιουργεί τους κανόνες επεξεργασίας (processing rules) των πακέτων για κάθε ροή έτσι ώστε να πραγματοποιήσει τις πολιτικές που καθορίζονται από τις εφαρμογές SDN και στη συνέχεια να εγκαταστήσει τους κανόνες στους πίνακες ροής. Ο ελεγκτής μπορεί να ενημερώσει τις καταχωρήσεις του πίνακα ροής των συσκευών του δικτύου είτε για να τροποποιήσει τη διαμόρφωση του δικτύου είτε για το δυναμικό έλεγχο του.

2.5.2. Υλοποιήσεις Ελεγκτών

Μερικοί ευρέως αναπτυγμένοι ελεγκτές SDN με βάση το OpenFlow είναι οι [4]:

- **NOX**: Ένας από τους πρώτους διαθέσιμους ελεγκτές OpenFlow. Αρκετές παραλλαγές του NOX υπάρχουν σήμερα, όπως NOX-MT, QNOX, Fort-Nox, POX κ.λπ. η κάθε μια με διαφορετική έμφαση [12].
- **OpenDaylight** (ODL): Ένας ελεγκτής ανοιχτού κώδικα, διαθέσιμος από το 2014. Είναι ένας αρθρωτός ελεγκτής SDN πολλαπλών πρωτοκόλλων που χρησιμοποιείται ευρέως στον κλάδο [13].
- **TungstenFabric:** Η ευελιξία και η λειτουργία κλίμακας του Tungsten Fabric, σε οποιαδήποτε υποδομή δικτύου IP και οποιοδήποτε cloud IaaS (Infrastructure as a Service), το έχει κάνει πολύ δημοφιλές σε πολλές περιπτώσεις χρήσης [14].
- **Beacon:** Είναι ένας γρήγορος, cross-platform, modular, Java-based OpenFlow ελεγκτής, που υποστηρίζει event-based και threaded operations [15].
- **Floodlight**: Ένας community-based ανοιχτού κώδικα OpenFlow ελεγκτής, βασιζόμενος στην Java που υποστηρίζει: τα πρωτόκολλα OpenFlow 1.0 έως 1.5, διάφορους εικονικούς και φυσικούς μεταγωγείς OpenFlow, και το OpenStack [16].
- **Ryu:** Ελεγκτής ανοιχτού κώδικα βασιζόμενος σε Python. Έχει modular χαρακτήρα και υποστηρίζει πολλά πρωτόκολλα όπως OpenFlow, Netconf, OF-config [17].

- **FlowVisor:** Είναι ένας ειδικού σκοπού ελεγκτής OpenFlow που υποστηρίζει έναν αποκεντρωμένο ελεγκτή. Επιτρέπει τη σίγαση των πόρων για να διασφαλιστεί ότι μόνο ένας εκχωρημένος ελεγκτής μπορεί να ελέγξει έναν μεταγωγέα. Έχει επίσης λειτουργίες διαμόρφωσης της κυκλοφορίας [18].

2.5.3. Συγκριτική Αξιολόγηση Απόδοσης Ελεγκτών

Σύμφωνα με το [19] θα αξιολογήσουμε τα χαρακτηριστικά και την ταχύτητα μερικών δημοφιλών ελεγκτών ανοιχτού κώδικα, όπως ONOS, Ryu, Floodlight και OpenDayLight όσον αφορά την καθυστέρηση και την απόδοση.

Βασιζόμενοι στον Πίνακα 1, είναι σαφές ότι οι ελεγκτές OpenDayLight και ONOS είναι οι πιο πλούσιοι σε χαρακτηριστικά ελεγκτές. Αξιοποιώντας το **OSGi** (Open Services Gateway initiative), αυτοί οι ελεγκτές είναι εξαιρετικά αρθρωτοί και έχουν εξαιρετικούς χρόνους εκτέλεσης στη φόρτωση πακέτων. Επιπλέον, και οι δύο διαθέτουν φιλικό προς το χρήστη GUI για του προγραμματιστές, μια ενεργή και αξιόπιστη κοινότητα που συμβάλλει με νέες ιδέες βελτίωσης, ενώ η κατανεμημένη αρχιτεκτονική τους τα καθιστά ιδανικά για ρεαλιστικές εφαρμογές SDN. Τέλος, αυτοί οι ελεγκτές υποστηρίζουν Southbound διεπαφές σχεδιασμένες για υβριδικά δίκτυα SDN και είναι επομένως κατάλληλοι για τέτοια σενάρια. Ωστόσο, το ONOS δεν υποστηρίζει cloud orchestration (π.χ. OpenStack) που είναι επιτακτική για τη διαχείριση εικονικών πόρων.

Πίνακας 1. Συγκριτική Αξιολόγηση SDN Ελεγκτών Ανοικτού Κώδικα *(Πηγή: [19] p.4)*

	Ryu	Floodlight	OpenDayLight	ONOS
Southbound Interfaces	OF 1.0, 1.2, 1.3, 1.4, NETCONF, OFCONFIG, OVSDB	OF1.0,1.1,1.2, 1.3, 1.4,1.5	OF1.0, 1.3,1.4,1.5 NETCONF/YANG,OVSDB, PCEP, BGP/LS, LISP, SNMP, OFCONFIG	OF1.0, 1.3,1.4, 1.5 NETCONF
REST API	Yes (For SB only)	Yes	Yes	Yes
GUI	Yes (Initial phase)	Web/ Java- based	Web-based	Web-based
Modularity	Medium	Medium	High	High
Orchestrator Support	Yes	Yes	Yes	No
OS Support	Most supported on Linux	Linux, Windows , and MAC	Linux, Windows , and MAC	Linux, Windows , and MAC
Partner	Nippo Telegraph And Telephone Corporation (NTT)	Big Switch Networks	Linux Foundation With Memberships Covering Over 40 Companies, like Cisco, IBM,	ON.LAB, Sk Telecom, Cisco, Ericsson, Fujitsu, Huawei, Intel, AT&T, Nec, Ciena, Nsf.Ntt Comunnication
Documentation	Medium	Good	Very good	Good
Programming Language	Python	Java	Java	Java
Multi-threading support	Yes	Yes	Yes	Yes
TLS Support	Yes	Yes	Yes	Yes
Virtualization	Mininet and OVS	Mininet and OVS	Mininet and OVS	Mininet and OVS
Application Domain	Campus	Campus	Data center and Transport-SDN WAN	Data center and Transport-SDN WAN
Distributed/Centralized	Centralized	Centralized	Distributed	Distributed

Το Ryu διαθέτει αρκετά χαρακτηριστικά που το καθιστούν ιδανικό για μικρές εφαρμογές SDN. Η σχετικά μικρή αρθρωτότητά του, η χρήση της Python, η centralized αρχιτεκτονική και η αποκλειστική υποστήριξη για Linux OS περιορίζουν την χρήση του σε δίκτυα μικρής κλίμακας.

Από την σύγκριση των χαρακτηριστικών είναι σαφές ότι το Floodlight έχει τις λιγότερες δυνατότητες. Ένα από τα πιο σημαντικά αλλά περιορισμένα χαρακτηριστικά του είναι ο αριθμός των υποστηριζόμενων Southbound διεπαφών, υποστηρίζοντας μόνο το OpenFlow. Επιπλέον, το Floodlight δεν διαθέτει αρθρωτότητα, δεν υποστηρίζει το distributed scheme και δεν υποστηρίζει cloud orchestration. Όλες αυτές οι ελλείψεις το καθιστούν κατάλληλο μόνο για εφαρμογές μικρής κλίμακας.

Οι μετρήσεις απόδοσης που λαμβάνονται υπόψη είναι: **latency** και **throughput.** Ο κύριος στόχος είναι να διερευνηθεί ποιος ελεγκτής δίνει την υψηλότερη απόδοση και τη χαμηλότερη καθυστέρηση υπό διάφορους φόρτους εργασίας.

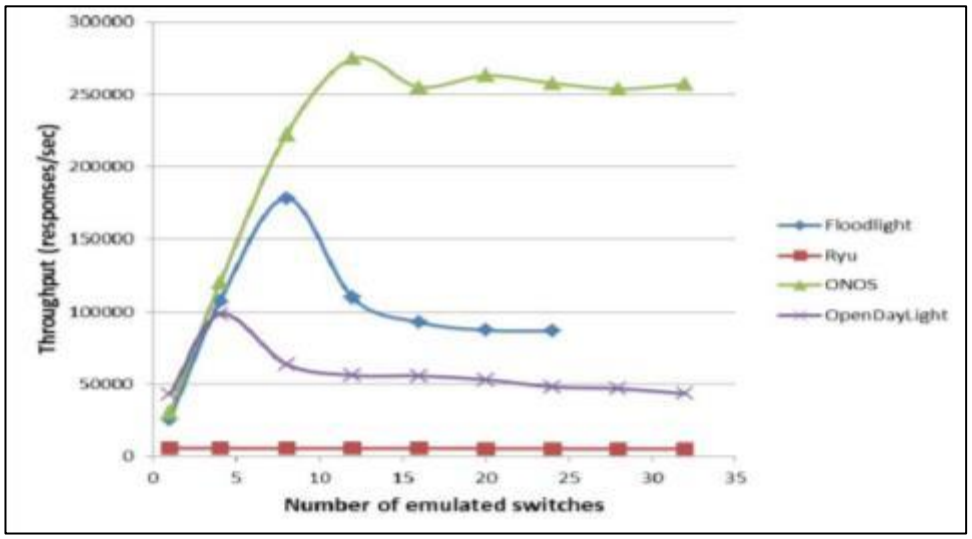
Αποτελέσματα Ανάλυσης Διαμεταγωγής (Throughput): Τα αποτελέσματα της αξιολόγησης της διαμεταγωγήςς που φαίνονται στην Εικόνα 10 δείχνουν ότι το Floodlight και το OpenDayLight επηρεάζονται δραστικά από την αύξηση του αριθμού των ενεργών μεταγωγέων. Αυτό συμβαίνει διότι ο μεγάλος αριθμός μεταγωγέων προκαλεί συμφόρηση στο επίπεδο δεδομένων, απαιτώντας έτσι υψηλή επεξεργαστική ισχύ. Η απόδοση του Ryu είναι η φτωχότερη και παραμένει σταθερή ανεξάρτητα από τον αριθμό των μεταγωγέων. Το ONOS παρουσιάζει την καλύτερη απόδοσης, πιθανά λόγω της έμφυτης υποστήριξής του για δίκτυα πολύ μεγάλης κλίμακας

Αποτελέσματα Ανάλυσης Καθυστέρησης (Latency): Όπως φαίνεται στην Εικόνα 11, όταν οι δοκιμές πραγματοποιήθηκαν σε latency mode, το ONOS παρουσίασε το χειρότερο latency. Η υποβάθμιση της απόδοσης του Ryu είναι αμελητέα μικρή. Η καθυστέρηση του OpenDayLight αυξήθηκε ελαφρώς με τον αυξανόμενο φόρτο εργασίας και το Floodlight εμφάνισε καλύτερη απόδοση latency σε σύγκριση με το OpenDayLight όταν ο αριθμός των MAC ρυθμίστηκε στα 100K.

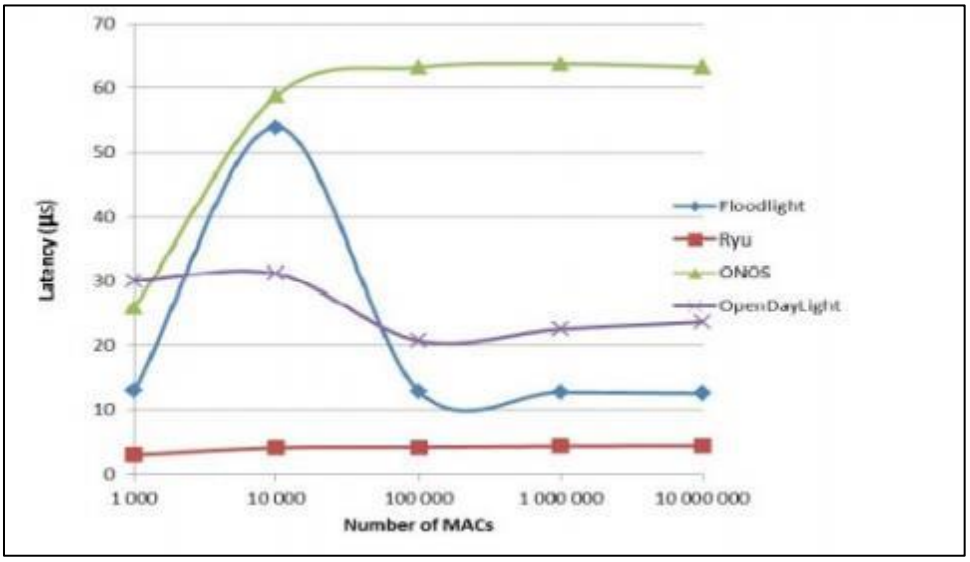
Αποτελέσματα Ανάλυσης Επεκτασιμότητας: Όπως παρουσιάζεται στην Εικόνα 12, τα OpenDayLight, Ryu και Floodlight επηρεάζονται σημαντικά από τον φόρτο εργασίας που προκύπτει από το μεγάλο αριθμό διευθύνσεων MAC. Ωστόσο, το ONOS δεν εμφανίζει παρόμοια συμπεριφορά. Η απόδοση του ONOS είναι σχεδόν σταθερή ξεκινώντας από τους 10K μεταγωγείς.

Αποτελέσματα Συνολικής Απόδοσης: Ως συνολική απόδοση ορίζεται η αναλογία απόδοσης (responses/sec) προς τη καθυστέρηση (sec). Η Εικόνα 13 απεικονίζει τα αποτελέσματα απόδοσης λαμβάνοντας υπόψη τόσο την καθυστέρηση όσο και την απόδοση υπό ποικίλα μεγέθη δικτύου. Αυτά τα αποτελέσματα δείχνουν ότι το ONOS έχει συνολικά την καλύτερη απόδοση καθώς αυξάνεται το μέγεθος του δικτύου.

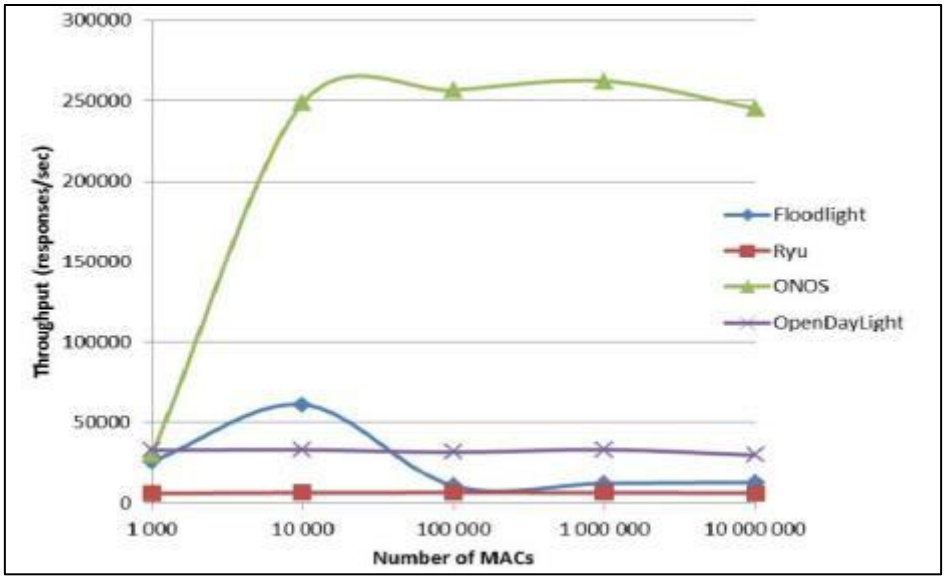
Κάτω από ποικίλο φόρτο εργασίας (MAC) όπως φαίνεται στην Εικόνα 13, το ONOS εξακολουθεί να παρουσιάζει εξαιρετική επεκτασιμότητα, ενώ τα OpenDayLight και Ryu εμφανίζουν συγκριτικά την ίδια απόδοση για 100.000 MAC και παραπάνω. Το Floodlight έχει τη χειρότερη συνολική απόδοση για μεγάλο φόρτο εργασίας.



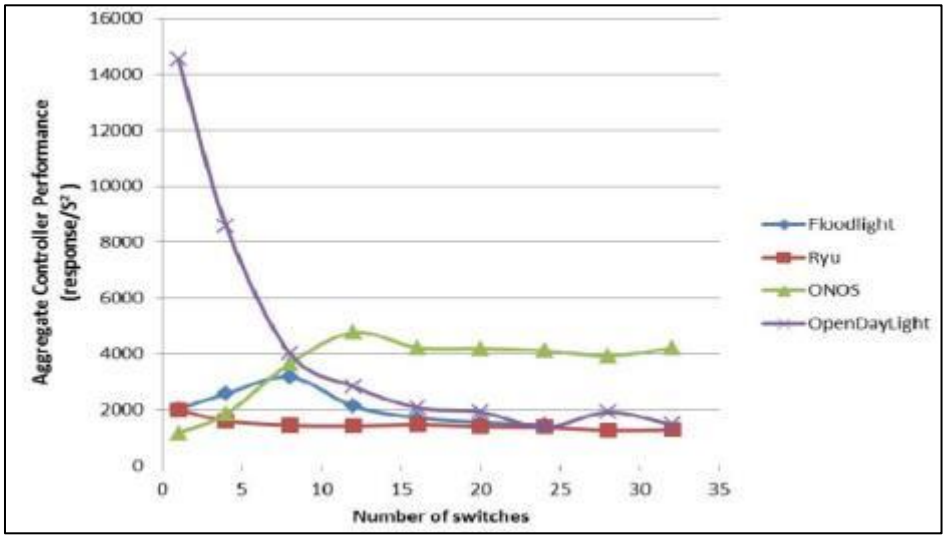
Εικόνα 11: Ανάλυση Throughput

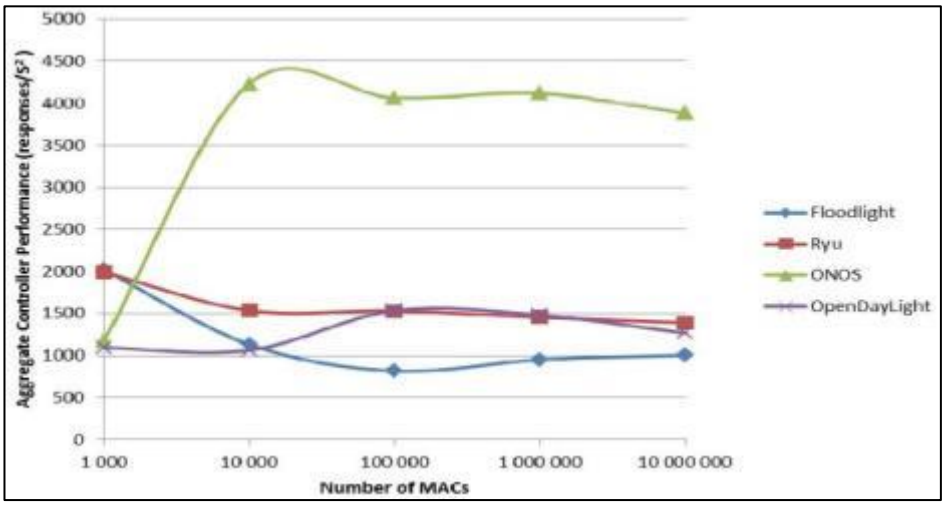


Εικόνα 10: Ανάλυση Latency



Εικόνα 12: Ανάλυση Επεκτασιμότητας





Εικόνα 13: Ανάλυση Συνολικής Απόδοσης

2.5.4. Πιθανά Προβλήματα των Ελεγκτών SDN

Σε γενικές γραμμές, οι ελεγκτές SDN πάσχουν από προβλήματα που είναι κοινά σε οποιαδήποτε νέα τεχνολογία [9]. Ενώ πολλά σημαντικά προβλήματα αντιμετωπίζονται από την αρχιτεκτονική του ελεγκτή, υπήρξαν συγκριτικά λίγες εμπορικές χρήσεις μεγάλης κλίμακας μέχρι στιγμής. Καθώς αυξάνεται η εμπορική χρήση, θα απαιτείται περισσότερη πραγματική εμπειρία σε μεγάλα απαιτητικά δίκτυα. Συγκεκριμένα, θα απαιτείται εμπειρία σε ένα ευρύτερο φάσμα εφαρμογών με έναν πιο ετερογενή συνδυασμό τύπων εξοπλισμού πριν υπάρξει ευρεία εμπιστοσύνη στην αρχιτεκτονική αυτή.

Για παράδειγμα, ενδέχεται να υπάρχουν περισσότερες από μία εφαρμογές SDN που εκτελούνται σε έναν μόνο ελεγκτή. Σε τέτοιες περιπτώσεις, τα ζητήματα που σχετίζονται με την ιεράρχηση της εφαρμογής και τη διαχείριση ροής καθίστανται σημαντικά. Ποια εφαρμογή θα πρέπει να λάβει πρώτη ένα συμβάν? Θα πρέπει να απαιτείται από την εφαρμογή να μεταβιβάσει το συμβάν αυτό στην επόμενη εφαρμογή, ή θα μπορεί να θεωρήσει την διαδικασία ολοκληρωμένη, οπότε καμία άλλη εφαρμογή δεν θα έχει την ευκαιρία να εξετάσει και να δράσει στο ληφθέν συμβάν? Η επίτευξη της επιτυχίας σε αυτές τις ποικίλες εφαρμογές θα απαιτήσει την επαρκή αντιμετώπιση αρκετών πιθανών ζητημάτων του ελεγκτή. Σε ορισμένες περιπτώσεις, αυτές οι λύσεις θα διατίθενται σε πολλές μορφές από διαφορετικούς προμηθευτές. Σε άλλες περιπτώσεις, ένας οργανισμός τυποποίησης όπως ο ONF θα πρέπει να αναθέσει ένα πρότυπο.

Μια κεντρική αρχιτεκτονική ελέγχου είναι απαραίτητη για να αντιμετωπίσει τα ζητήματα του latency, κλίμακας, διαθεσιμότητας και ασφάλειας. Εκτός από αυτά τα γενικότερα ζητήματα, ο κεντρικός ελεγκτής SDN θα πρέπει να αντιμετωπίσει τις προκλήσεις του συντονισμού μεταξύ των εφαρμογών, την έλλειψη ενός τυποποιημένου Northbound API και την ιεράρχηση της ροής.

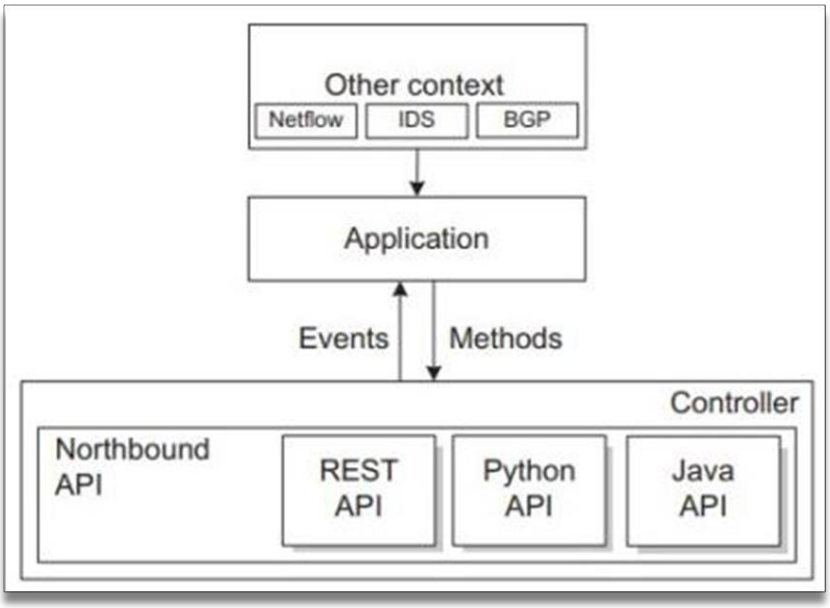
2.6. Επίπεδο Εφαρμογής

Το Επίπεδο Εφαρμογής μπορεί να θεωρηθεί ως ο "**εγκέφαλος**" της αρχιτεκτονική SDN που **λαμβάνει αποφάσεις** σχετικά με τα **network policies** και **ορίζει την συμπεριφορά του δικτύου** για την εκπλήρωση των policies αυτών [9] [1]. Οι εφαρμογές ελέγχου SDN αλληλοεπιδρούν με τους ελεγκτές μέσω του Northbound API, από οπού λαμβάνουν πληροφορίες για την κατάσταση του δικτύου και ζητάνε από τους ελεγκτές να ληφθούν ορισμένες ενέργειες ελέγχου. Μέσω αυτού του API οι εφαρμογές μπορούν:

1. Να διαμορφώσουν τις ροές για να δρομολογήσουν πακέτα μέσω της καλύτερης διαδρομής μεταξύ δύο τελικών σημείων.
2. Εξισορροπήσουν τα φορτία κίνησης σε πολλές διαδρομές ή σε συγκεκριμένα τελικά σημεία.
3. Να αντιδράσουν σε αλλαγές στην τοπολογία του δικτύου, όπως αστοχίες συνδέσμων και προσθήκη νέων συσκευών και διαδρομών
4. Ανακατευθύνουν την κυκλοφορία για επιθεώρηση, authentication και διαχωρισμού παρόμοιων εργασιών που σχετίζονται με την ασφάλεια.

Η γενική ευθύνη μιας εφαρμογής SDN είναι να εκτελεί οποιαδήποτε λειτουργία για την οποία έχει σχεδιαστεί, είτε είναι εξισορρόπηση φορτίου, τείχος προστασίας ή κάποια άλλη λειτουργία. Μόλις ο ελεγκτής τελειώσει με την προετοιμασία των συσκευών και αναφέρει την τοπολογία δικτύου στην εφαρμογή, η εφαρμογή ξοδεύει το μεγαλύτερο μέρος του χρόνου επεξεργασίας στο να ανταποκρίνεται σε γεγονότα. Ενώ η βασική λειτουργικότητα της εφαρμογής θα ποικίλλει από τη μία εφαρμογή στην άλλη, η συμπεριφορά της εφαρμογής καθοδηγείται από συμβάντα που προέρχονται από τον ελεγκτή καθώς και από εξωτερικές εισόδους. Οι εξωτερικές είσοδοι θα μπορούσαν να περιλαμβάνουν συστήματα παρακολούθησης δικτύου, όπως τα Netflow, IDS ή BGP.

Η εφαρμογή επηρεάζει το δίκτυο απαντώντας στα συμβάντα όπως φαίνεται στην Εικόνα 14. Η εφαρμογή γίνεται listener για συγκεκριμένα συμβάντα και ο ελεγκτής επικαλείται την μέθοδο επανάκλησης της εφαρμογής κάθε φορά που συμβαίνει ένα τέτοιο συμβάν. Μερικά παραδείγματα συμβάντων που χειρίζεται μια εφαρμογή SDN είναι τα: End-User Device Discover, Network Device Discovery, και Incoming Packet. Στις δύο πρώτες περιπτώσεις, τα συμβάντα αποστέλλονται στην εφαρμογή SDN κατά την ανακάλυψη μιας νέας συσκευής τελικού χρήστη (δηλ. MAC address) ή μιας νέας συσκευής δικτύου (π.χ. switch, router, wireless access point), αντίστοιχα. Τα συμβάντα εισερχόμενων πακέτων αποστέλλονται στην εφαρμογή SDN όταν λαμβάνεται ένα πακέτο από μια συσκευή SDN, λόγω είτε μιας καταχώρησης ροής που δίνει εντολή στη συσκευή SDN να προωθήσει το πακέτο στον ελεγκτή είτε επειδή δεν υπάρχει αντίστοιχη καταχώριση ροής στη συσκευή SDN.



Εικόνα 14: Northbound API Ελεγκτή

Πηγή: [9] p.74

Υπάρχουν πολλοί τρόποι με τους οποίους μια εφαρμογή SDN μπορεί να ανταποκριθεί σε συμβάντα που έχουν ληφθεί από τον ελεγκτή SDN. Υπάρχουν οι απλές απαντήσεις, όπως η λήψη ενός συνόλου προεπιλεγμένων καταχωρήσεων ροής σε μια συσκευή που ανακαλύφθηκε

πρόσφατα. Αυτές οι προεπιλεγμένες ή στατικές ροές είναι συνήθως ίδιες για κάθε κατηγορία συσκευής δικτύου που ανακαλύπτεται και, ως εκ τούτου, απαιτείται ελάχιστη επεξεργασία από την εφαρμογή. Υπάρχουν επίσης πιο πολύπλοκες απαντήσεις που μπορεί να απαιτούν πληροφορίες για την κατάσταση του δικτύου που συλλέγονται από κάποια άλλη πηγή εκτός από τον ελεγκτή.

2.7. Ασφάλεια δικτύων SDN

Η υιοθέτηση του SDN μπορεί να προσφέρει οφέλη όχι μόνο στη διαχείριση και στην ενορχήστρωση των δικτυών, αλλά και στην ασφάλεια τους. Όπως πάντα όμως, η εισαγωγή μιας νέας τεχνολογίας φέρνει πλεονεκτήματα αλλά και προβληματισμούς. Έτσι, η ασφάλεια του SDN από μόνη της είναι ένας αρκετά σημαντικός τομέας έρευνας. Ο συγκεντρωτικός σχεδιασμός του SDN μπορεί να δημιουργήσει προκλήσεις ασφάλειας, όπως επιθέσεις κατανεμημένης άρνησης υπηρεσίας (DoS) κατά του ελεγκτή SDN. Κάθε επίπεδο της αρχιτεκτονικής SDN μπορεί να έχει πολλά εκτεθειμένα σημεία.

2.7.1. Προκλήσεις Ασφαλείας δικτύων SDN

Οι σχέσεις μεταξύ των στοιχείων ενός δικτύου SDN μπορούν να εισαγάγουν νέες ευπάθειες, οι οποίες απουσιάζουν στα παραδοσιακά δίκτυα. Για παράδειγμα, η χρήση του transport layer security είναι προαιρετική στα δίκτυα OpenFlow, ενώ η φύση του πρωτοκόλλου επικοινωνίας μπορεί να δημιουργήσει προβλήματα ασφάλειας όπως το DoS, την εισαγωγή δόλιου κανόνα ροής και την τροποποίηση υπάρχοντος κανόνα [20].

Τα τρία επίπεδα: εφαρμογών, ελέγχου και δεδομένων μπορούν να υποστούν επιθέσεις. Για παράδειγμα, μπορεί να υπάρχουν ευπάθειες λογισμικού στους ελεγκτές SDN (OpenDayLight, ONOS, Floodlight). Επιπλέον, τα κανάλια επικοινωνίας μεταξύ των επιπέδων, δηλαδή τα Northbound και Southbound APIs μπορούν να αντιμετωπίσουν επιθέσεις ασφαλείας. Πιο συγκεκριμένα [20] [4]:

- **Επίπεδο Εφαρμογών:** Όλα τα θέματα ασφάλειας που μπορεί να υπάρχουν σε μια τυπική εφαρμογή web, όπως Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) ισχύουν και για το SDN. Οι κακόβουλες / παραβιασμένες εφαρμογές μπορούν να επιτρέψουν την εξάπλωση της επίθεσης σε ολόκληρο το δίκτυο.
- **Επίπεδο Ελέγχου:** Ο εισβολέας μπορεί να δημιουργήσει κίνηση από πλαστές διευθύνσεις IP και να στείλει έναν τεράστιο όγκο κίνησης στον ελεγκτή. Χρησιμοποιώντας αυτήν την επίθεση μπορεί να κορεστεί η επικοινωνία μεταξύ του μεταγωγέα και του ελεγκτή, αυξάνοντας έτσι το service latency ή στη χειρότερη περίπτωση μπορεί να βγεί εκτός λειτουργίας ο ελεγκτής.
- **Επίπεδο Δεδομένων:** Οι εισβολείς μπορούν να αλλάξουν την παγκόσμια προβολή του δικτύου, φτιάχνοντας πλαστά πακέτα Link Layer Discovery Protocol (LLDP). Οι επιτιθέμενοι μπορούν επίσης να παρατηρήσουν την καθυστέρηση στην επικοινωνία μεταξύ του επιπέδου ελέγχου και των εφαρμογών του επιπέδου δεδομένων χρησιμοποιώντας ειδικά κατασκευασμένα πακέτα, προσδιορίζοντας έτσι τη λογική

της εφαρμογής του ελεγκτή. Οι επιτιθέμενοι μπορούν επίσης να στοχεύσουν την μνήμη των μεταγωγέων, υπερχειλίζοντας την με την εισαγωγή ενός μεγάλου αριθμού κανόνων.

- **Κανάλι Επικοινωνίας:** Τα κανάλια επικοινωνίας μεταξύ των μεταγωγέων και των ελεγκτών (Southbound API), και ελεγκτών με επίπεδο εφαρμογών (Northbound API) μπορούν να υποβληθούν σε επίθεση Man-in-the Middle (MITM) . Το ARP Poisoning είναι ένα παράδειγμα τέτοιας επίθεσης. Άλλες επιθέσεις που στοχεύουν στο κανάλι επικοινωνίας περιλαμβάνουν την παρακολούθηση της κίνησης μεταξύ των τερματικών υπολογιστών και την κρυφή τροποποίηση της κίνησης μεταξύ των τερματικών αυτών.

Ο Πίνακας 2 συνοψίζει ορισμένα από τα ζητήματα ασφαλείας που σχετίζονται με τα διαφορετικά στοιχεία του SDN που περιγράψαμε παραπάνω.

Πίνακας 2. Θέματα ασφαλείας που σχετίζονται με τα διαφορετικά επίπεδα SDN *(Πηγή: [20])*

Security Issue/Attack	SDN Layer Affected or Targeted				
	Application Layer	App-Ctl Interface	Control Layer	Ctl-Data Interface	Data Layer
Unauthorized Access e.g.					
Unauthorized Controller Access			✓	✓	✓
Unauthenticated Application	✓	✓	✓		
Data Leakage e.g.					
Flow Rule Discovery (Side Channel Attack on Input Buffer)					✓
Forwarding Policy Discovery (Packet Processing Timing Analysis)					✓
Data Modification e.g.					
Flow Rule Modification to Modify Packets			✓	✓	✓
Malicious Applications e.g.					
Fraudulent Rule Insertion	✓	✓	✓		
Controller Hijacking			✓	✓	✓
Denial of Service e.g.					
Controller-Switch Communication Flood			✓	✓	✓
Switch Flow Table Flooding					✓
Configuration Issues e.g.					
Lack of TLS (or other Authentication Technique) Adoption			✓	✓	✓
Policy Enforcement	✓	✓	✓		

2.7.2. Σημαντικές Απειλές Ασφάλειας

Μερικές από τις πιο σημαντικές απειλές ασφάλειας για τα δίκτυα SDN είναι οι [4]:

I. **Fake Traffic Flows:** Οι κακόβουλοι χρήστες μπορούν να χρησιμοποιήσουν επιθέσεις DoS προς τους TCAM (ternary content-addressable memory) μεταγωγείς της υποδομής SDN, με στόχο την εξάντληση της χωρητικότητας τους. Το πρόβλημα

μπορεί να μετριαστεί χρησιμοποιώντας έναν απλό μηχανισμό ελέγχου ταυτότητας, αλλά εάν ο εισβολέας είναι σε θέση να εισχωρήσει στον διακομιστή της εφαρμογής που περιχέει τα στοιχεία των χρηστών, τότε ο εισβολέας θα μπορεί να χρησιμοποιήσει τα στοιχεία αυτά για να εγχύσει πλάστες ροές στο δίκτυο.

II. **Switch Specific Vulnerabilities:** Οι μεταγωγείς που υπάρχουν στο περιβάλλον SDN μπορεί να έχουν ευπάθειες. Για παράδειγμα, μια ευπάθεια στο Juniper OS (CVE-2018-0019) SNMP MIB-II (mib2d) επιτρέπει σε έναν απομακρυσμένο χρήστη να προκαλέσει την κατάρρευση της διεργασίας mib2d με αποτέλεσμα την άρνησης υπηρεσίας (DoS) για το υποσύστημα SNMP. Ένας μεταγωγέας μπορεί να χρησιμοποιηθεί για να επιβραδύνει την κίνηση στο περιβάλλον SDN, να παρεκκλίνει τη κίνηση του δικτύου για να κλέψει πληροφορίες, να χρησιμοποιηθεί για την εισαγωγή πλαστών αιτημάτων κυκλοφορίας με στόχο την υπερφόρτωση του ελεγκτή ή των γειτονικών μεταγωγέων.

III. **Control Plane Communication Attack:** Ο ελεγκτής είναι το πιο σημαντικό κομμάτι σε ένα περιβάλλον SDN. Ένας παραβιασμένος ελεγκτής μπορεί να ρίξει ολόκληρο το δίκτυο. Για παράδειγμα, μια παλιά έκδοση του ελεγκτή ONOS πάσχει από απομακρυσμένη επίθεση DoS (CVE-2015-7516). Ο εισβολέας μπορεί να προκαλέσει την παραπομπή ενός δείκτη NULL και την αποσύνδεση του μεταγωγέα, στέλνοντας δυο πλαίσια Ethernet με ether_type Jumbo Frame (0x8870) στον ελεγκτή ONOS v1.5.0.

IV. **Έλλειψη εμπιστοσύνης μεταξύ των εφαρμογών Controller και Management:** Οι εφαρμογές του ελεγκτή και της διαχείρισης στερούνται ενός ενσωματωμένου μηχανισμού για την εδραίωση της εμπιστοσύνης, ενώ η δημιουργία πιστοποιητικών μπορεί να διαφέρει από συσκευή σε συσκευή SDN.

2.7.3. Σχεδιασμός ασφαλούς και αξιόπιστης πλατφόρμας SDN

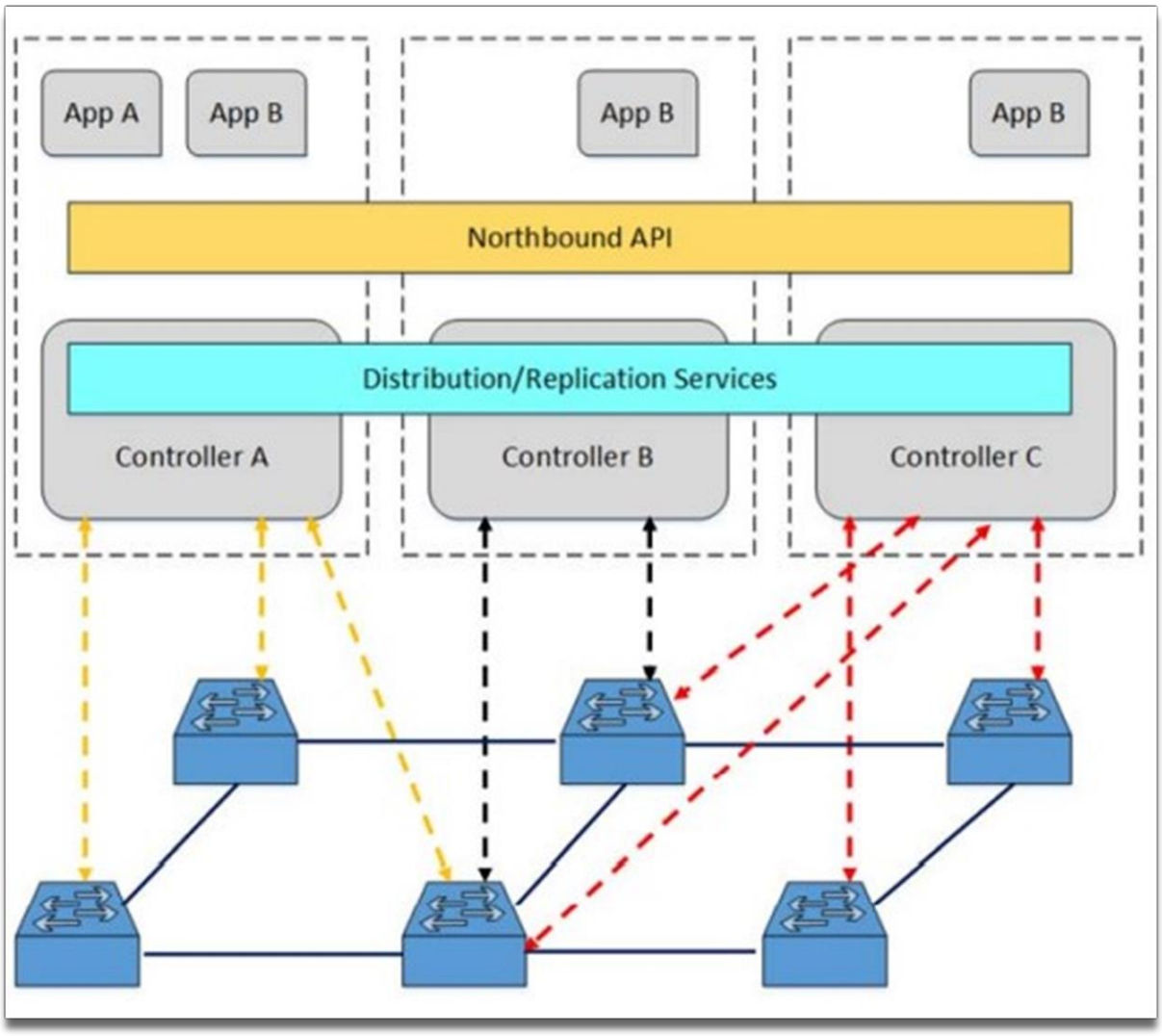
Μια ασφαλής και αξιόπιστη αρχιτεκτονική SDN όπως φαίνεται στην Εικόνα 15, με χαρακτηριστικά όπως ανοχή σε σφάλματα, self-healing, ένα αξιόπιστο framework και δυναμικές δυνατότητες παροχής υπηρεσιών μπορεί να χρησιμοποιηθεί για την αντιμετώπιση των απειλών που συζητήθηκαν στην προηγούμενη υποενότητα.

Σε αυτήν την ενότητα, θα αναλύσουμε καθέναν από τους μηχανισμούς ασφαλείας που μπορούν να ενσωματωθούν στο σχεδιασμό ενός πιο ασφαλούς δικτύου SDN [4]. Οι μηχανισμοί αυτοί είναι οι εξής:

- **Replication (Αντιγραφή):** Η αντιγραφή των εφαρμογών και του ελεγκτή μπορεί να βοηθήσει στην αντιμετώπιση των περιπτώσεων αποτυχίας λόγο μεγάλου όγκου κίνησης ή τρωτών σημείων στο λογισμικό. Για παράδειγμα, θα μπορούσαν να χρησιμοποιηθούν δυο ή και τρεις διαφορετικοί ελεγκτές ταυτόχρονα, με τις εφαρμογές να τρέχουν παράλληλα σε κάθε ελεγκτή. Αυτή η προσέγγιση μπορεί να βοηθήσει στην αντιμετώπιση των αστοχιών υλικού και λογισμικού (τυχαίων ή κακόβουλων). Ένα άλλο πλεονέκτημα της αντιγραφής είναι η απομόνωση των κακόβουλων εφαρμογών διατηρώντας παράλληλα τη συνέπεια των υπηρεσιών.

- **Diversity (Ποικιλία):** Η χρήση ενός μόνο είδους λογισμικού ή λειτουργικού συστήματος διευκολύνει τους επιτιθέμενους στο να εκμεταλλευτούν έναν στόχο [21]. Η ποικιλομορφία βελτιώνει την ανοχή κατά των εισβολών και βοηθά στην αποφυγή των κοινών βλαβών και τρωτών σημείων, καθώς υπάρχουν μόνο μερικές διασταυρούμενες ευπάθειες μεταξύ διαφορετικών λογισμικών ή λειτουργικών συστημάτων. Στο επίπεδο ελέγχου SDN, η χρήση ποικίλων ελεγκτών μπορεί να βοηθήσει στη μείωση της πλευρικής κίνησης ενός εισβολέα και της κατάρρευσης του συστήματος που μπορεί να προκληθεί από κοινά τρωτά σημεία.
- **Automated Recovery (Αυτόματη Ανάκτηση):** Σε περίπτωση επίθεσης που οδηγεί σε διακοπή των υπηρεσιών, οι προληπτικοί και αντιδραστικοί μηχανισμοί αποκατάστασης της ασφάλειας μπορούν να βοηθήσουν στη διατήρηση της βέλτιστης διαθεσιμότητας των υπηρεσιών. Κατά την αντικατάσταση λογισμικού, π.χ. του ελεγκτή SDN, είναι απαραίτητο να πραγματοποιηθεί η αντικατάσταση με νέες και διαφορετικές εκδόσεις του στοιχείου.
- **Dynamic Device Association (Δυναμική Συσχέτιση Συσκευών):** Η συσχέτιση μεταξύ του ελεγκτή και των συσκευών όπως ο μεταγωγέας OpenFlow θα πρέπει να είναι δυναμικής φύσης. Για παράδειγμα, εάν αποτύχει ένας ελεγκτή, ο μεταγωγέας θα πρέπει να μπορεί να συσχετιστεί δυναμικά με τον εφεδρικό ελεγκτή με έναν ασφαλή τρόπο (ύπαρξη κατάλληλου μηχανισμού ελέγχου της ταυτότητας για τον εντοπισμό πιθανού κακόβουλου ελεγκτή).
- **Controller-Switch Trust:** Ένας μηχανισμός εγκαθίδρυσης εμπιστοσύνης μεταξύ του ελεγκτή και του μεταγωγέα είναι σημαντικός για την αντιμετώπιση των περιπτώσεων εισαγωγής ψεύτικων ροών. Σε ένα βασικό σενάριο εμπιστοσύνης, ο ελεγκτής μπορεί να διατηρήσει μια λίστα των μεταγωγέων που επιτρέπεται να στέλνουν συγκεκριμένα μηνύματα στον ελεγκτή. Σε πιο περίπλοκο σενάριο, μπορεί να χρησιμοποιηθεί το Public Key Infrastructure (PKI) μεταξύ του επιπέδου ελέγχου και των συσκευών του επιπέδου δεδομένων. Οι συσκευές που εμφανίζουν ανώμαλη συμπεριφορά μπορούν να τεθούν σε λειτουργία καραντίνας από τον ελεγκτή.
- **Controller-App Plane Trust:** Τα στοιχεία λογισμικού αλλάζουν συμπεριφορά λόγω αλλαγών στο περιβάλλον. Επιπλέον, η γήρανση του λογισμικού μπορεί να δημιουργήσει τρωτά σημεία ασφαλείας. Τα στοιχεία του επιπέδου ελέγχου και εφαρμογής πρέπει να χρησιμοποιούν αυτονόμους μηχανισμούς διαχείρισης της εμπιστοσύνης, όπως τρίτες εφαρμογές (π.χ. Certificate Authority). Οι ποιοτικές μετρήσεις όπως η εμπιστευτικότητα, η ακεραιότητα και η διαθεσιμότητα μπορούν επίσης να αξιοποιηθούν για να καθοριστεί η αξιοπιστία μιας εφαρμογής στο SDN framework [22].
- **Security Domains:** Οι τομείς ασφαλείας βοηθούν στην κατάτμηση του δικτύου σε διαφορετικά επίπεδα εμπιστοσύνης και τον περιορισμό των απειλών μόνο στο επηρεασμένο τμήμα. Μπορεί επίσης να γίνεται απομόνωση των τομέων ασφάλειας για πιο εις βάθος άμυνα. Για παράδειγμα, μια web-server εφαρμογή σε έναν φυσικό διακομιστή θα πρέπει να αλληλοεπιδρά μόνο με τις back-end εφαρμογές της βάσης δεδομένων και όχι με οποιαδήποτε άλλη εφαρμογή που εκτελείται στο ίδιο δίκτυο.

Ένα σημαντικό σημείο κατά την ανάπτυξη της επιθυμητής λύσης ασφαλείας ή συνδυασμού λύσεων είναι η ανάλυση κόστους-οφέλους (το delay που εισάγεται στο δίκτυο, χρήση CPU/πόρων) αυτών των λύσεων μεμονωμένα καθώς και μαζί.



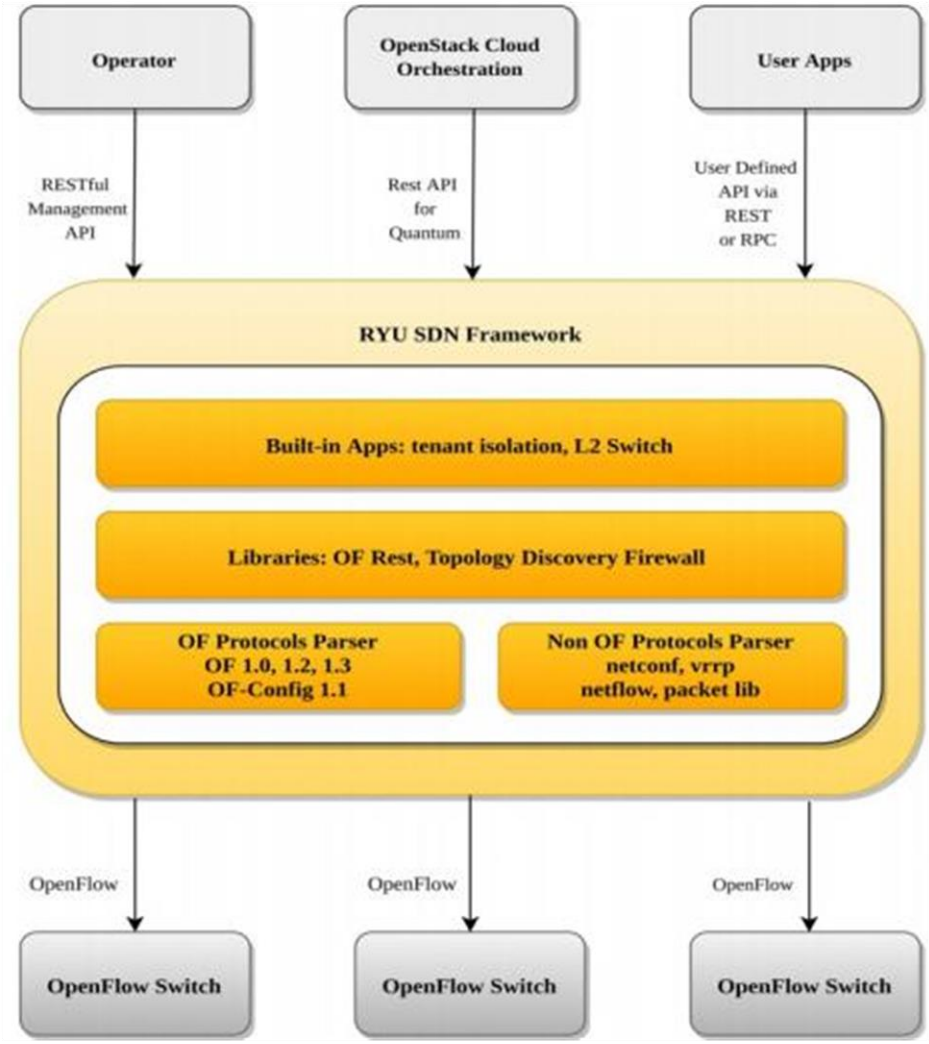
Εικόνα 15: Ασφαλές και αξιόπιστο δίκτυο SDN.

Πηγή: [4] p.141

3. Υλοποίηση Εφαρμογής Ryu SDN Controller

3.1. Ryu SDN Controller

Η υλοποίηση της εφαρμογής SDN για τις ανάγκες της εργασίας αυτής έγινε σε περιβάλλον προσομοίωσης χρησιμοποιώντας το **Ryu Framework** [23]. Το RYU είναι ένα open-source component-based software-defined networking framework που αναπτύχθηκε από την Nippon Telegraph and Telephone (NTT). Το RYU είναι ένας ελεγκτής SDN που εφαρμόζεται πλήρως στην Python. Ο ελεγκτής RYU παρέχει software components με καλά καθορισμένες διεπαφές (APIs) που διευκολύνουν στη δημιουργία νέων εφαρμογών διαχείρισης και ελέγχου δικτύου. Το RYU υποστηρίζει διάφορα πρωτόκολλα για τη διαχείριση συσκευών δικτύου, όπως OpenFlow, Netconf, OF-config κ.λπ., ενώ υποστηρίζει επίσης τις εκδόσεις OpenFlow 1.0,1.2,1.3,1.4,1.5 και τα Nicira Extensions. Η αρχιτεκτονική του RYU SDN Controller φαίνεται στην Εικόνα 16.



Εικόνα 16: Αρχιτεκτονική RYU SDN Controller

Πηγή: [23] p.559

Ο ελεγκτής RYU SDN έχει τρία επίπεδα. Το επάνω επίπεδο, γνωστό ως επίπεδο εφαρμογών, αποτελείται από επιχειρησιακές και δικτυακές λογικές εφαρμογές. Το μεσαίο επίπεδο αποτελείται από υπηρεσίες δικτύου γνωστές ως επίπεδο ελέγχου ή πλαίσιο SDN, οπού και βρίσκονται τα Southbound & Northbound APIs, ενώ το κάτω επίπεδο που αποτελείται από φυσικές και εικονικές συσκευές είναι γνωστό ως επίπεδο δεδομένων. Το RYU παρέχει διαφορά χρήσιμα εξαρτήματα για εφαρμογές SDN όπως: OpenStack Quantum, Firewall, OFREST κ.λπ. Η Southbound διεπαφή είναι ικανή να υποστηρίξει πολλαπλά πρωτόκολλα όπως: OpenFlow, Netconf, OF-config, κ.λπ. Έχει δοκιμαστεί και πιστοποιηθεί ότι λειτουργεί με πολλούς μεταγωγείς OpenFlow, συμπεριλαμβανομένου του Open vSwitch και άλλες λοιπές λύσεις των Centec, Hewlett Packard, IBM και NEC.

3.2. Environment Setup

Η υλοποίηση του Software Defined Network για τις ανάγκες της πτυχιακής εργασίας έγινε σε περιβάλλον προσομοίωσης χρησιμοποιώντας παρόμοια ή ίδια εργαλεία με αυτά που λειτουργούν σε πραγματικές εγκαταστάσεις υλικού. Η κύρια διαφορά είναι πως το επίπεδο δεδομένων προσομοιώθηκε μέσω **Mininet** καθώς η εξεύρεση OpenFlow enabled μεταγωγέα και η δημιουργία μιας φυσικής υποδομής είναι ένα δαπανηρό και δύσκολο έργο.

3.2.1. Απαιτήσεις συστήματος

Για την υλοποίηση χρησιμοποιήθηκε ένας μόνο διακομιστής (Host) με τα ακόλουθα χαρακτηριστικά:

- OS: Ubuntu 20.04.2 LTS
- OS Type: 64-bit
- CPU: Intel Core i5-4460 CPU
- Memory: 16 GB
- Graphics Unit: NVIDIA GeForce GTX 780
- Disk Capacity: VHD 24.0 GB

3.2.2. Εγκατάσταση Mininet & RYU

Το εργαλείο που χρησιμοποιείται για τη δημιουργία και την ανάπτυξη της εφαρμογής SDN είναι το Mininet. Το Mininet είναι ένας εξομοιωτής δικτύου που δημιουργεί ένα δίκτυο εικονικών κεντρικών υπολογιστών, μεταγωγέων, ελεγκτών και τις μεταξύ τους συνδέσεις. Δημιουργεί ένα **ρεαλιστικό εικονικό δίκτυο** που λειτουργεί σε ένα μόνο μηχάνημα, με το οποίο ο χρήστης μπορεί εύκολα να αλληλοεπιδράσει χρησιμοποιώντας το Command Line Interface (CLI). Η συμπεριφορά των εικονικών κεντρικών υπολογιστών, μεταγωγέων, συνδέσμων και ελεγκτών είναι παρόμοια με των πραγματικών συσκευών δικτύου. Σε πολλές περιπτώσεις είναι δυνατό να δημιουργηθεί ένα δίκτυο Mininet που μοιάζει με ένα πραγματικό δίκτυο ή ένα πραγματικό δίκτυο που μοιάζει με δίκτυο Mininet, οπού αμφότερα τρέχουν τον ίδιο κώδικα. Ο

ευκολότερος τρόπος χρήσης του Mininet είναι η εγκατάσταση ενός Virtual Machine (VM). Τα βήματα που ακολουθήθηκαν περιγράφονται παρακάτω:

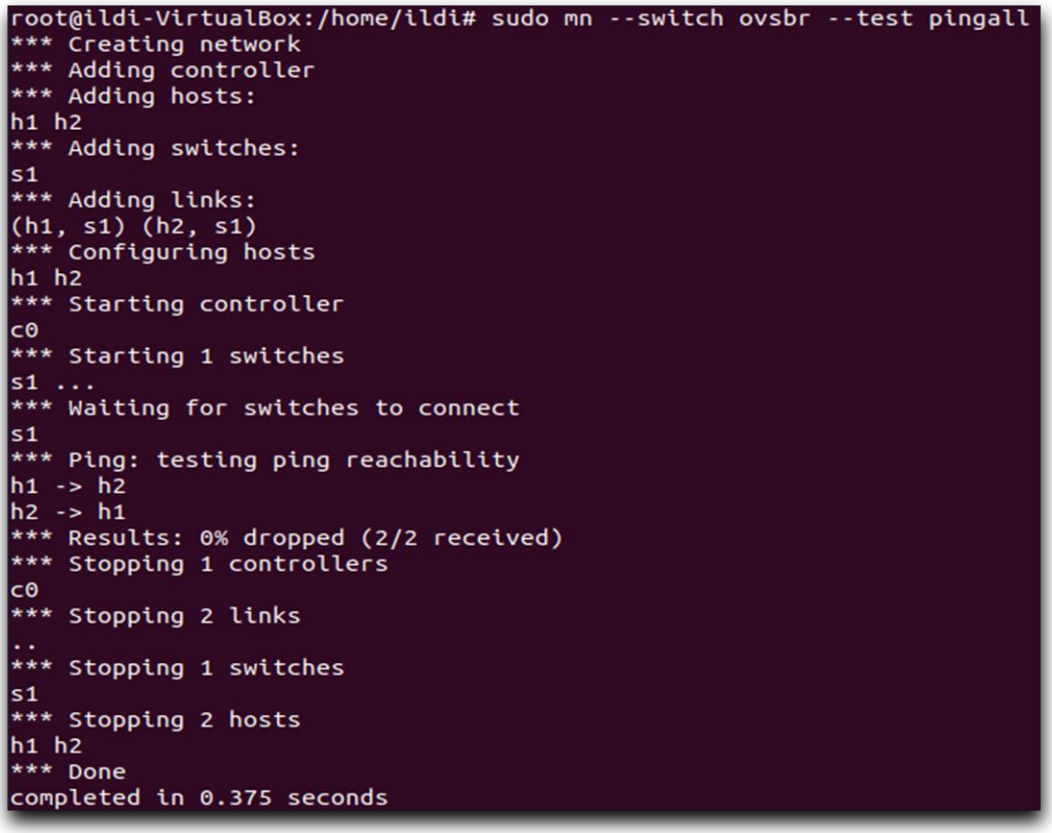
Για την εγκατάσταση του Mininet 2.3.0 ακολουθήθηκαν τα βήματα που περιγράφονται στην ιστοσελίδα: Download/Get Started With Mininet - Mininet.

- Έναρξη του Ubuntu Virtual Box.
- Εγκατάσταση του source code με την χρήση των εντολών :

```
git clone git://github.com/mininet/mininet
mininet/util/install.sh -a
```

- Έλεγχος κυρίας λειτουργικότητας με την χρήση της εντολής:

```
sudo mn --switch ovsbr --test pingall
```



Εικόνα 17: Έλεγχος λειτουργικότητας Mininet

- Εγκατάσταση του RYU σύμφωνα με το: Getting Started — Ryu 4.34 documentation

```
pip install ryu
```

Το Mininet περιέχει μια έκδοση του Open vSwitch (OVS), η οποία χρησιμοποιείται για την υλοποίηση των μεταγωγέων SDN.

3.2.3. Εισαγωγή στο Mininet

Μετά την εγκατάσταση του Mininet, είναι δυνατή η δημιουργία τοπολογίας και η σύνδεση με τον ελεγκτή. Η προεπιλεγμένη τοπολογία στο Mininet είναι η minimal η οποία αποτελείται από έναν μεταγωγέα OpenFlow, δύο κεντρικούς υπολογιστές συνδεδεμένους σε αυτόν και τον ελεγκτή αναφοράς OpenFlow. Για να γίνει εκκίνηση της τοπολογία αυτής, πρέπει να εισαχθεί η ακόλουθη εντολή στο CLI:

```
sudo mn
```

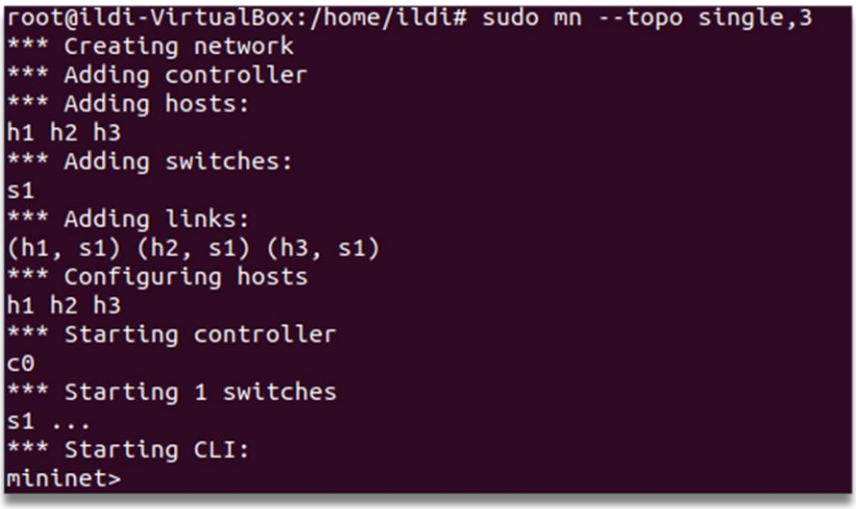
Ενώ μερικές από τις εντολές που μπορούν να χρησιμοποιηθούν είναι οι:

- Εμφάνιση των Mininet CLI commands: `mininet> help`
- Εμφάνιση των Nodes: `mininet> nodes`
- Εμφάνιση των Links: `mininet> net`
- Πληροφορίες σχετικά με όλα τα nodes του δικτύου: `mininet> dump`
- Δοκιμή συνδεσιμότητας μεταξύ όλων των hosts : `mininet> pingall`
- Έξοδος από την τοπολογία: `mininet> exit`
- Αυτόματος καθαρισμός του Mininet Console: `mininet> sudo mn -c`
- Αλληλεπίδραση με τους hosts π.χ. h1: `mininet> xterm h1`

Όπως ήδη αναφέρθηκε, η minimal τοπολογία αποτελείται από έναν μόνο μεταγωγέα συνδεδεμένο σε δύο hosts και μπορεί να καθοριστεί με την εντολή CLI:

```
sudo mn –topo=minimal
```

Το όρισμα `--topo` ακολουθούμενο από μια παράμετρο προστίθεται για να αλλάξει αυτή η τοπολογία σε μια διαφορετική. Οι διαθέσιμες τοπολογίες είναι οι εξής: **Linear, Tree, Torus, Single, Reversed και Minimal**. Για παράδειγμα, η παράμετρος "single" χρησιμοποιείται στο Mininet VM όπως φαίνεται στην Εικόνα 18, η οποία υποδεικνύει μια τοπολογία που έχει έναν μόνο μεταγωγέα και HOST_NUMBER = 3 αριθμό κεντρικών υπολογιστών συνδεδεμένων σε αυτόν.



Εικόνα 18: Single Topology με 3 hosts

Εκτός από τις τοπολογίες που αναφέρθηκαν προηγουμένως, οι προσαρμοσμένες (custom) τοπολογίες μπορούν εύκολα να οριστούν χρησιμοποιώντας ένα απλό Python API. Η εντολή που θα χρησιμοποιηθεί για την χρήση μιας custom τοπολογίας είναι η:

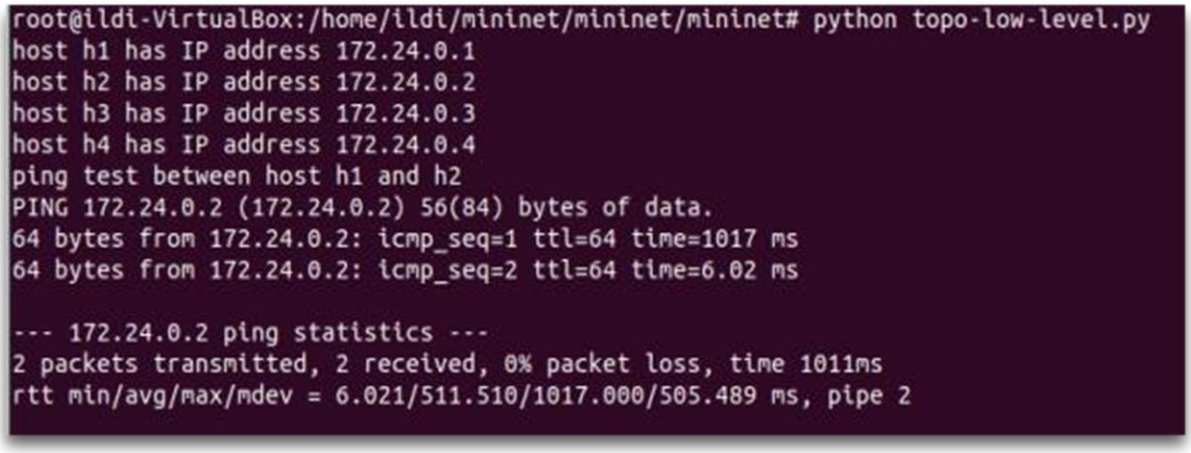
```
sudo mn –custom [PYTHON_SCRIPT_PATH] –topo mytopo
```

Με λίγες γραμμές κώδικα python, μπορούμε να δημιουργήσουμε προσαρμοσμένες τοπολογίες σύμφωνα με τις πειραματικές μας ανάγκες. Στο παρακάτω παράδειγμα δημιουργήσαμε την τοπολογία **topo-low-level.py**:

```
from mininet.node import OVSSwitch,Host,Controller
from mininet.link import Link
h1=Host('h1')                           #δημιουργία hosts
h2=Host('h2')
h3=Host('h3')
h4=Host('h4')
s1=OVSSwitch('s1',inNamespace=False)    #δημιουργία switch s1
c0=Controller('c0',inNamespace=False)   #δημιουργία  controller c0
Link(h1,s1)                             #δημιουργία  links
Link(h2,s1)
Link(h3,s1)
Link(h4,s1)
h1.setIP('172.24.0.1/16')               #ορισμός element ip
h2.setIP('172.24.0.2/16')
h3.setIP('172.24.0.3/16')
h4.setIP('172.24.0.4/16')
c0.start()                              #έναρξη controller
s1.start([c0])                          #σύνδεση s1 --> c0
print("host h1 has IP address",h1.IP())
print("host h2 has IP address",h2.IP())
print("host h3 has IP address",h3.IP())
print("host h4 has IP address",h4.IP())
print("ping test between host h1 and h2")
print h1.cmd('ping -c2',h2.IP())
s1.stop()
c0.stop()
```

Το αποτέλεσμα της εκτέλεσης του παραπάνω custom topology φαίνεται στην Εικόνα 19.
Για το παράδειγμα χρησιμοποιήθηκαν οι εντολές :

- **Host(), Link()** & **Switch():** για την δημιουργία των hosts και των μεταγωγέων, και για την μεταξύ τους σύνδεση.
- **setIP() & setMAC():** για τον ορισμό των διευθύνσεων στους hosts.
- **IP() & MAC():** Get IP/MAC.
- **setARP():** δημιουργία στατικών καταχωρήσεων ARP



Εικόνα 19: Αποτέλεσμα από την εκτέλεση του Low-Level Custom Topology

Εκτός από τη βασικές δυνατότητες δικτύωσης, το Mininet παρέχει δυνατότητες απομόνωσης και περιορισμού της απόδοσης μέσω των CPULimitedHost και TCLink. Η κλάση TCLink αντιπροσωπεύει μια σύνδεση ελεγχόμενης κυκλοφορίας και είναι δυνατό να οριστεί αυτόματα μέσω του command line με την χρήση της παρακάτω εντολής:

```
sudo mn –link tc,bw=10,delay=10ms
```

3.3. Εφαρμογές Ryu

3.3.1. Εισαγωγικά ελεγκτή Ryu

Ο ελεγκτής Ryu διαθέτει ενσωματωμένες εφαρμογές, διαθέσιμες στην ιστοσελίδα: Ryu App. Μερικές από αυτές είναι οι:

- simple_switch
- simple_monitor
- oftcl_rest
- rest_qos
- rest_firewall
- rest_router

Για να γίνει χρήση των εφαρμογών αυτών γράφουμε την εντολή:

```
ryu-manager ryu.app.<app_name>
```

Μερικές από τις εντολές που μπορούν να χρησιμοποιηθούν στο Ryu Command Line είναι οι:

```
ryu-manager –help
ryu-manager –ofp-tcp-listen-port 6634
ryu-manager –observe-links
ryu-manager -verbose
```

Ο ελεγκτής Ryu λειτουργεί με **Reactive & Proactive Flows.**

Reactive Flows:

- Όταν ένα νέο πακέτο που εισέρχεται στο μεταγωγέα δεν ταιριάζει με τις υπάρχουσες ροές, ο μεταγωγέας το στέλνει στον ελεγκτή.
- Ο ελεγκτής επιθεωρεί το πακέτο και δημιουργεί τη "λογική"
- Γίνεται εγκατάσταση της ροής για αυτήν τη συνεδρία στον μεταγωγέα

Proactive Flows:

- Ο ελεγκτής OpenFlow θα εγκαταστήσει τους πίνακες ροής εκ των προτέρων για όλες τις αντιστοιχίσεις.

Ένα παράδειγμα reactive εφαρμογής είναι το ενσωματωμένο **simple_switch_13**, μια απλή εφαρμογή μεταγωγέα που λειτουργεί με την εξής λογική:

1. Εγκατάσταση της καταχώρισης Table Miss στον μεταγωγέα .
2. Όταν ένα πακέτο έρχεται στον μεταγωγέα, γίνεται συσχετισμός με την καταχώριση Table Miss και, στη συνέχεια, ο μεταγωγέας το στέλνει στον ελεγκτή (μήνυμα PACKET IN).
3. Ο ελεγκτής βλέπει το src_mac του πακέτου και το ενημερώνει στο data base του (port to mac mapping).
4. Ο ελεγκτής βλέπει το dst_mac του πακέτου και αποφασίζει για τη θύρα εξόδου.
5. Ο ελεγκτής στέλνει το πακέτο στον μεταγωγέα (μήνυμα PACKET OUT).
6. Ο ελεγκτής προσθέτει τη ροή χρησιμοποιώντας το μήνυμα τροποποίησης ροής.

Για την εκτέλεση αυτής της δοκιμής χρειάζεται:

- Να οριστεί το Mininet Topology με την εξής εντολή, όπως φαίνεται στην Εικόνα 20:

```
sudo mn --controller=remote,ip=127.0.0.1 --mac --
switch=ovsk,protocols=OpenFlow13 --topo=single,4
```

Εικόνα 20: Τοπολογία Mininet

- Έναρξη του Ryu Hub App με την εντολή:

  ```
  ryu-manager ruy.app.simple_switch_13
  ```

- Έλεγχος για την ύπαρξη flows, οπού θα πρέπει να υπάρχει το Table Miss Entry, με την παρακάτω εντολή:

  ```
  sudo ovs-ofctl -O OpenFlow13 dump-flows s1
  ```

```
ildi@ildi-VirtualBox:~/mininet/SDN$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=151.749s, table=0, n_packets=4, n_bytes=280, priority=0 actions=CONTROLLER:65535
ildi@ildi-VirtualBox:~/mininet/SDN$
```

Εικόνα 21:Ύπαρξη Table Miss Entry

- Ping h1 προς h2 μέσω mininet prompt:

```
mininet> h1 ping 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=0.416 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.194 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.127 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.128 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3066ms
rtt min/avg/max/mdev = 0.127/0.216/0.416/0.118 ms
mininet>
```

Εικόνα 22: Ping h1 προς h2

- Επανέλεγχος των flows:

```
ildi@ildi-VirtualBox:~/mininet/SDN$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=276.634s, table=0, n_packets=17, n_bytes=1498, priority=1,in_port="s1-eth2",dl_src=00:00:0
0:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=276.632s, table=0, n_packets=16, n_bytes=1400, priority=1,in_port="s1-eth1",dl_src=00:00:0
0:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x0, duration=629.082s, table=0, n_packets=15, n_bytes=1022, priority=0 actions=CONTROLLER:65535
ildi@ildi-VirtualBox:~/mininet/SDN$
```

Εικόνα 23: Ύπαρξη δυο νέων flows

3.3.2. Flow Manager

Το **Flow Manager** είναι μια εφαρμογή ελεγκτή RYU που δίνει στον χρήστη χειροκίνητο έλεγχο των πινάκων ροής σε ένα δίκτυο OpenFlow. Ο χρήστης μπορεί να δημιουργήσει, να τροποποιήσει ή να διαγράψει ροές απευθείας από την εφαρμογή. Ο χρήστης μπορεί επίσης να παρακολουθεί τους μεταγωγείς OpenFlow και να βλέπει στατιστικά στοιχεία.

Για την εγκατάσταση χρειάζεται απλά να γίνει cloning με την εντολή:

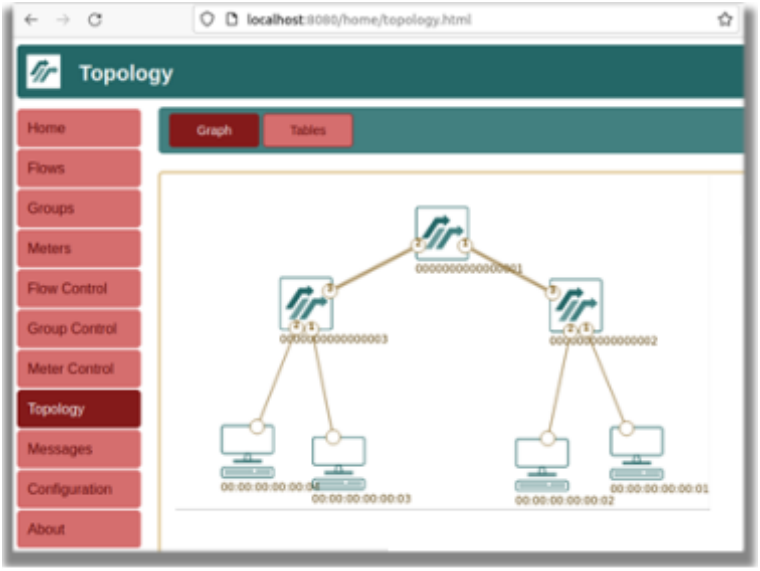
```
git clone https://github.com/martimy/flowmanager
```

Το Flow Manager δεν διαθέτει ενσωματωμένη εφαρμογή ελεγκτή, οπότε για τις λειτουργίες μεταγωγής θα τρέχει παράλληλα μια εφαρμογή ελέγχου.

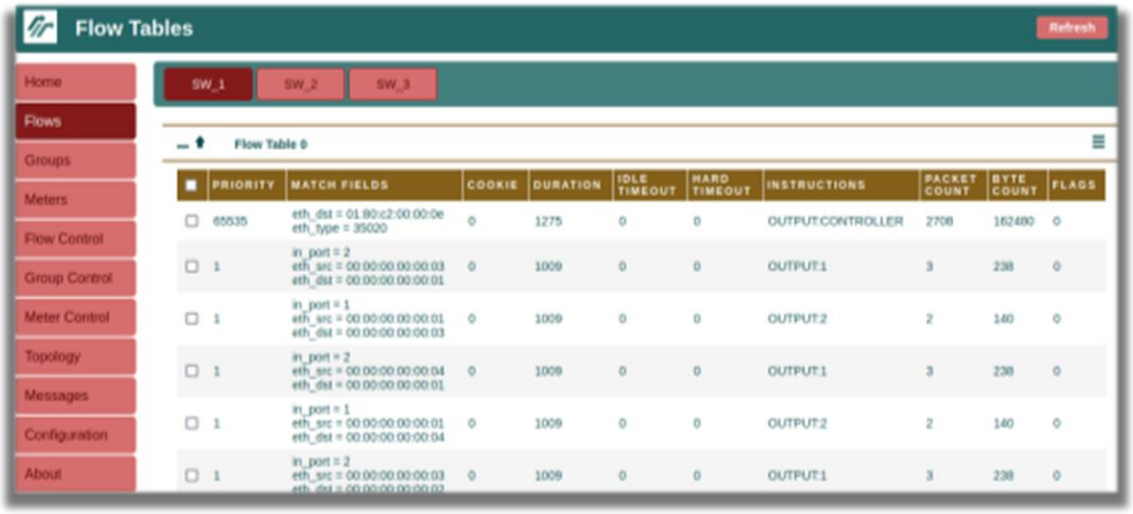
- Για την εκκίνηση του RYU Flow Manager και του Mininet Topology θα χρησιμοποιηθούν οι εντολές:

```
ryu-manager –observe-links ~/flowmanager/flowmanager.py
ryu.app.simple_switch13
sudo mn --controller=remote,ip=127.0.0.1 --mac -i
10.1.1.0/24 --topo=tree,depth=2,fanout=2
mininet>pingall
```

- Επίσκεψη διεύθυνσης: *http://localhost:8080/home/index.html*
- Έλεγχος τουTopology Diagram:

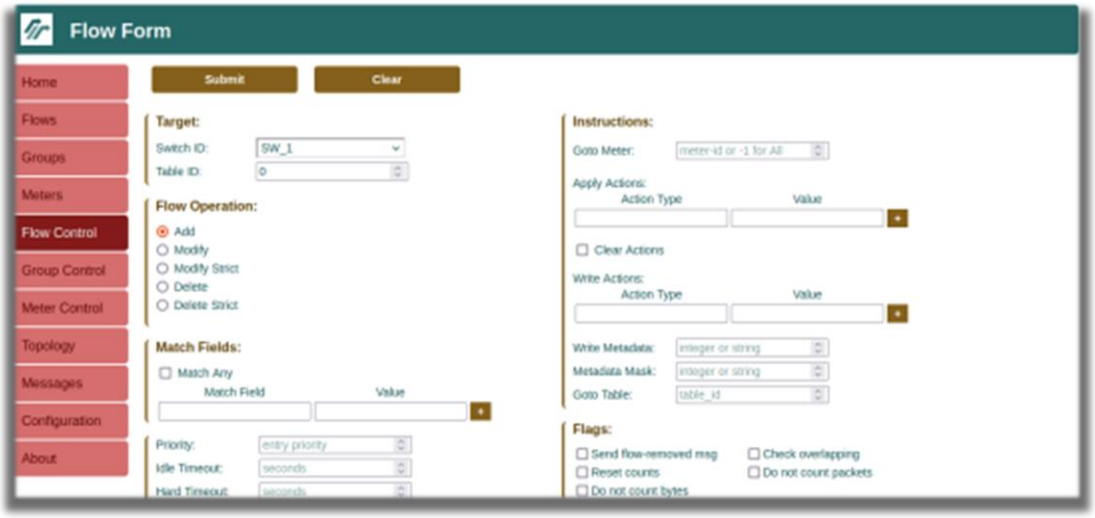


Εικόνα 25: Topology Manager

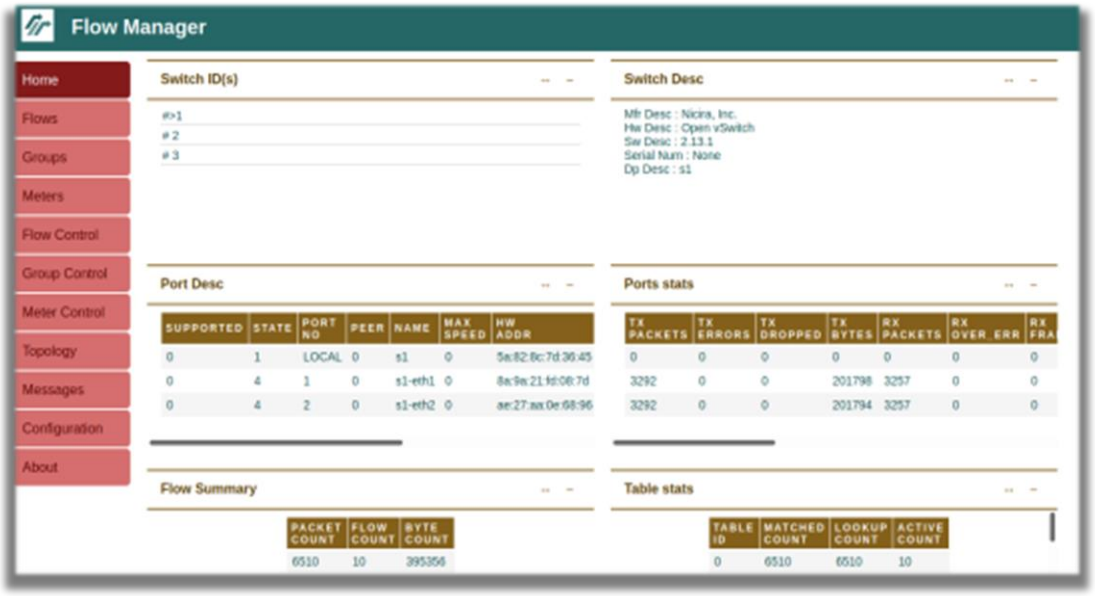


Εικόνα 24: Flow Tables

Το Flow Manager δίνει επίσης τη δυνατότητα ελέγχου και τροποποίησης των ροών μέσο των λειτουργιών: Add, Modify & Delete Flow



Εικόνα 27: Flow Control



Εικόνα 26: Flow Manager

3.3.3. Προγραμματισμός Εφαρμογής Ελεγκτή

Οι εφαρμογές Ryu κατά βάση είναι **Event Based Python Scripts**. Ο ελεγκτής Ryu εκπέμπει events, οπού παραλήπτης το **OpenFlow Message Receiver**, όπως για παράδειγμα το: *ofp_event.EventOFPFlowStatsReply*. Μια εφαρμογή αποτελείται από κάποια βασικά στοιχεία (components) τα οποία είναι [24]:

- ryu-manager: Ο κύριος εκτελέσιμος κώδικας.
- ryu.base.app_manager: Εκτελεί τον κεντρικό έλεγχο των εφαρμογών.
- ryu.ofproto: Protocol encoder & decoder.
- ryu.controller: Υλοποίηση του ελεγκτή OpenFlow, δημιουργία των OpenFlow events.
- ryu.packet: Βιβλιοθήκες ανάλυσης των πακέτων.

Η μέθοδος ανταλλαγής μηνυμάτων OpenFlow μεταξύ Controller και Switch:

1. Hello Message
2. Feature Request/Response Message
3. Flow Modification message για την εγκατάσταση του Table Miss Entry
4. Packet In Message
5. Packet Out Message
6. Flow Modification Message για την εγκατάσταση των Flows

Ανάλυση της εφαρμογής Simple_Switch_13:

Το Simple_Switch_13 όπως αναφέρθηκε και προηγούμενος, είναι μια built-in εφαρμογή μεταγωγέα που μπορεί ευκολά να χωριστεί σε μερικά βασικά τμήματα [24]:

- **Imports:**

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER,
MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
```

- **Application Class:**

```
class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
```

- **OpenFlow Event Handlers:**

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures,
CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        # If you hit this you might want to increase
        # the "miss_send_length" of your switch
        if ev.msg.msg_len < ev.msg.total_len:
          self.logger.debug("packet truncated: only %s
          of %s bytes", ev.msg.msg_len, ev.msg.total_len)
        …
```

Με το ***switch_features_handler*** event, εγκαθιστούμε την καταχώρηση TABLE MISS έτσι ώστε να μπορούμε να λάβουμε το μήνυμα ***packet_in.*** Με το ***packet_in_handler*** event γίνεται επεξεργασία και ανάλυση των διευθύνσεων ***sac_mac & dst_mac,*** δημιουργείται η λογική του μεταγωγέα και γίνεται εγκατάσταση του flow.

Εισαγωγή Ροής:

- Για την εισαγωγή νέας ροής χρειάζεται αρχικά να δημιουργηθούν κριτήρια για το πεδίο Match, στο παράδειγμα μας γίνεται με τη χρήση των εξής εντολών:

```
match = parser.OFPMatch()
match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
```

- Δημιουργία Actions:

```
actions =
[parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML
_NO_BUFFER)
out_port = 1
actions = [parser.OFPActionOutput(out_port)]
```

- Δημιουργία Instruction List με Actions:

```
inst=parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
actions)]
```

- Αποστολή μηνύματος OFP Flow Modification για την δημιουργία νέας ροής:

```
mod = parser.OFPFlowMod(datapath=datapath,
buffer_id=buffer_id,priority=priority,
match=match,instructions=inst)
```

3.3.4. Εφαρμογή Μεταγωγέα με Layer 3 & 4 Matching

Η δυο παρακάτω εφαρμογές χρησιμοποιούν σαν βάση τη λογική του simple_switch_13 με μονή αλλαγή στα πεδία Match, οπού και θα χρησιμοποιηθεί Layer 3 (**src_ip/dst_ip**) & Layer 4 (**src_ip/dst_ip, protocol, port number**) matching αντί για **src_mac/dst_mac.**

Layer 3 Switch:

Αλλαγές στον κώδικα του Simple_Switch_13:

- Import IP Library:

```
from ryu.lib.packet import ipv4
```

- Ορισμός του Match με βάση το IP (έλεγχος αν το πακέτο είναι IP Packet, αποκωδικοποίηση του src_ip/dst_ip από το packet header, ορισμός του Match)

```
# έλεγχος IP Protocol και δημιουργία Match
 if eth.ethertype == ether_types.ETH_TYPE_IP:
 ip = pkt.get_protocol(ipv4.ipv4)
 srcip = ip.src
```

```
    dstip = ip.dst
     match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,ip
v4_src=srcip,ipv4_dst=dstip)
```

Ο έλεγχος λειτουργικότητας γίνεται με τις εξής εντολές:

```
sudo mn --controller=remote,ip=127.0.0.1 --mac --
switch=ovsk,protocols=OpenFlow13 --topo=single,4
ryu-manager l3_switch.py
mininet>pingall
sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

Το αποτέλεσμα της τελευταίας εντολής μας εμφανίζει τα flows:

```
ildi@ildi-VirtualBox:~/mininet/SDN/flowmanager$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
[sudo] password for ildi:
 cookie=0x0, duration=15.756s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:"s1-eth2"
 cookie=0x0, duration=15.755s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=output:"s1-eth1"
 cookie=0x0, duration=15.748s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10.0.0.1,nw_dst=10.0.0.3 actions=output:"s1-eth3"
 cookie=0x0, duration=15.747s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10.0.0.3,nw_dst=10.0.0.1 actions=output:"s1-eth1"
 cookie=0x0, duration=15.738s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10.0.0.1,nw_dst=10.0.0.4 actions=output:"s1-eth4"
 cookie=0x0, duration=15.737s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10.0.0.4,nw_dst=10.0.0.1 actions=output:"s1-eth1"
 cookie=0x0, duration=15.724s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10.0.0.2,nw_dst=10.0.0.3 actions=output:"s1-eth3"
 cookie=0x0, duration=15.722s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10.0.0.3,nw_dst=10.0.0.2 actions=output:"s1-eth2"
 cookie=0x0, duration=15.714s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10.0.0.2,nw_dst=10.0.0.4 actions=output:"s1-eth4"
 cookie=0x0, duration=15.711s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10.0.0.4,nw_dst=10.0.0.2 actions=output:"s1-eth2"
 cookie=0x0, duration=15.696s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10.0.0.3,nw_dst=10.0.0.4 actions=output:"s1-eth4"
 cookie=0x0, duration=15.693s, table=0, n_packets=1, n_bytes=98, priority=1,ip,nw_src=10.0.0.4,nw_dst=10.0.0.3 actions=output:"s1-eth3"
 cookie=0x0, duration=16.386s, table=0, n_packets=39, n_bytes=2394, priority=0 actions=CONTROLLER:65535
ildi@ildi-VirtualBox:~/mininet/SDN/flowmanager$
```

Εικόνα 28: Flows του L3 Switch

Layer 4 Switch Logic:

1. Έλεγχος εάν το ethernet frame type είναι IP
2. Σε περίπτωση που είναι IP Packet, γίνεται εξαγωγή των **src_ip & dst_ip**
3. Έλεγχος IP Protocol (**ICMP / TCP / UDP**)
4. Εάν είναι ICMP γίνεται προετοιμασία του OpenFlow match με βάση **src_ip/dst_ip**
5. Εάν είναι TCP γίνεται εξαγωγή των TCP **src_port/dst_port** και προετοιμασία του OpenFlow match με βάση **src_ip/dst_ip** και **src_port/dst_port**
6. Εάν είναι UDP γίνεται εξαγωγή των UDP **src_port/dst_port** και ακολουθεί παρόμοια διαδικασία με το TCP.

Αλλαγές στον κώδικα του Simple_Switch_13:

- Import τα απαραίτητα libraries:

```
from ryu.lib.packet import in_proto
from ryu.lib.packet import ipv4
from ryu.lib.packet import icmp
from ryu.lib.packet import tcp
from ryu.lib.packet import udp
```

- Ορισμός του Match:

```
# έλεγχος IP Protocol και δημιουργία Match

if eth.ethertype == ether_types.ETH_TYPE_IP:
ip = pkt.get_protocol(ipv4.ipv4)
srcip = ip.src
dstip = ip.dst
protocol = ip.proto
#  ICMP Protocol
if protocol == in_proto.IPPROTO_ICMP:
match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_I,
ipv4_src=srcip, ipv4_dst=dstip, ip_proto=protocol)
#   TCP Protocol
elif protocol == in_proto.IPPROTO_TCP:
t = pkt.get_protocol(tcp.tcp)
match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, ipv4
_src=srcip, ipv4_dst=dstip, ip_proto=protocol, tcp_src=t.src_p
ort, tcp_dst=t.dst_port,)
#   UDP Protocol
elif protocol == in_proto.IPPROTO_UDP:
u = pkt.get_protocol(udp.udp)
match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, ipv4
_src=srcip, ipv4_dst=dstip, ip_proto=protocol, udp_src=u.src_p
ort, udp_dst=u.dst_port,)
```

Ο έλεγχος λειτουργικότητας γίνεται στέλνοντας πακέτα **ICMP/TCP/UDP** μεταξύ των hosts με τις παρακάτω εντολές:

```
mininet>h1 ping h2
mininet>h2 iperf -s & #εκκίνηση iperf server
mininet>h1 iperf -c h2 -b 1m -t 5 #εκκίνηση iperf client
```

Με το πέρας της δοκιμής μπορούμε να δούμε τα αποτελέσματα με την χρήση της εντολής *dump-flows s1.*:

```
ildi@ildi-VirtualBox:~/mininet/SDN/flowmanager$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=378.586s, table=0, n_packets=1, n_bytes=98, priority=1,icmp,nw_src=10.0.0.1,nw_dst=1
 cookie=0x0, duration=378.582s, table=0, n_packets=1, n_bytes=98, priority=1,icmp,nw_src=10.0.0.2,nw_dst=1
 cookie=0x0, duration=349.370s, table=0, n_packets=0, n_bytes=0, priority=1,tcp,nw_src=10.0.0.1,nw_dst=10.
ctions=output:"s1-eth2"
 cookie=0x0, duration=349.368s, table=0, n_packets=0, n_bytes=0, priority=1,tcp,nw_src=10.0.0.2,nw_dst=10.
ctions=output:"s1-eth1"
 cookie=0x0, duration=339.122s, table=0, n_packets=52, n_bytes=658792, priority=1,tcp,nw_src=10.0.0.1,nw_d
5001 actions=output:"s1-eth2"
 cookie=0x0, duration=339.110s, table=0, n_packets=50, n_bytes=3300, priority=1,tcp,nw_src=10.0.0.2,nw_dst
76 actions=output:"s1-eth1"
 cookie=0x0, duration=417.615s, table=0, n_packets=25, n_bytes=1634, priority=0 actions=CONTROLLER:65535
```

Εικόνα 29: Πακέτα ICMP και TCP

3.3.5. Υλοποίηση Flow Timeout

Υπάρχουν δυο είδη **Timeout** για τις ροές OpenFlow:

- **Hard Timeout:** Μια μη μηδενική τιμή προκαλεί την κατάργηση της καταχώρησης ροής μετά τον δεδομένο αριθμό δευτερολέπτων, ανεξάρτητα από το πόσα πακέτα έχουν αντιστοιχιστεί.
- **Idle Timeout:** Μια μη μηδενική τιμή προκαλεί την κατάργηση της καταχώρησης ροής όταν δεν ταιριάζει με κανένα πακέτο στον δεδομένο αριθμό δευτερολέπτων.

Ο ορισμός του timeout είναι εφικτός με μια μικρή αλλαγή στη μεταβλητή mod του υπάρχοντα κώδικα Simple_Switch_13:

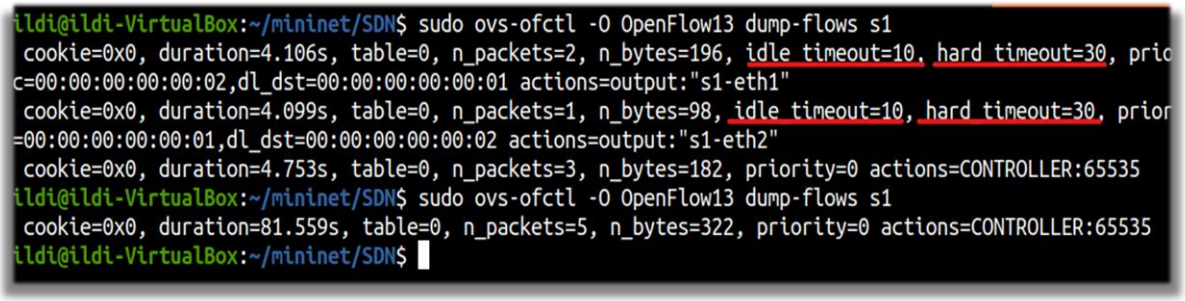
```
def add_flow …
if buffer_id:
mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id
,idle_timeout=10, hard_timeout=30, priority=priority, match=ma
tch,instructions=inst)
else:
mod = parser.OFPFlowMod(datapath=datapath, priority=priority,i
dle_timeout=10, hard_timeout=30, match=match, instructions=ins
t)
datapath.send_msg(mod)
```

Ο έλεγχος λειτουργικότητας γίνεται με τον εξής τρόπο:

1. Εκτέλεση της τοπολογίας Mininet (Single,2) και της εφαρμογής **flow_timeout.py**.
2. Ping των hosts στο Mininet.
3. Έλεγχος των flows με την χρήση της εντολής *dump-flows*.
4. Επανέλεγχος των flows μετά από 10 και 30 δευτερόλεπτα.

Αποτέλεσμα:

Όπως φαίνεται στην Εικόνα 30, τα flows εξαφανίζονται μετά το πέρας ορισμένων δευτερολέπτων.



Εικόνα 30:Αποτελέσματα εντολής *dump-flows* του *flow-timeout.py*

3.3.6. Εφαρμογή Συλλογής Στατιστικών

Το πρωτόκολλο OpenFlow παρέχει εκτενή στατιστικά στοιχεία, όπως:

- Flow Statistics
- Flow Aggregate Statistics
- Port Statistics
- Group Table Statistics
- Meter Statistics

Ο χρήστης μπορεί να συλλέξει αυτά τα στατιστικά στοιχεία στέλνοντας ένα Request Message (π.χ. OpenFlow FlowStatistics Request Message) στο μεταγωγέα. Στο υποκεφάλαιο αυτό θα υλοποιήσουμε τρεις εφαρμογές συλλογής στατιστικών στοιχείων.

Εφαρμογή Flow Statistics:

Ο στόχος αυτής της εφαρμογής είναι να μετράει τα στατιστικά στοιχεία των flows ή το utilization ανά τακτά χρονικά διαστήματα (10 δευτερόλεπτα) και να τα εμφανίζει.

Λογική:

- Χρήση του μηχανισμού **Ryu Thread** (HUB).
- Μέσα στο thread, θα γίνεται αποστολή του OpenFlow Flow Statistics Request Message ανά τακτά χρονικά διαστήματα.
- Η απάντηση των στατιστικών στοιχείων ροής θα λαμβάνεται ως συμβάν
- Θα εμφανίζονται οι τιμές των byte & packet counters.

Κώδικας:

1. Χρήση του Simple_Switch_13 ως βάση.
2. Ζητάμε τα **data paths** των μεταγωγέων για την δημιουργία του Statistics Request Message. Ξεκινάμε δηλώνοντας το ***datapath*** στο ***init***, και υστέρα αποθηκεύουμε το αντικείμενο ***datapath*** στο λεξικό (Python Dictionary) του ***datapaths*** στο πεδίο *switch_features_handler* .

   ```
   self.datapaths = {}
   self.datapaths[datapath.id] = datapath
   ```

3. Import το **Ryu Thread Library.**
   ```
   from ryu.lib import hub
   ```
4. Έναρξη του thread στο *init* με το όνομα ***monitor_thread***
   ```
   self.monitor_thread = hub.spawn(self.monitor)
   ```
5. Δημιουργία του thread function στο *Ryu manager application class*. Στη συνάρτηση αυτή επεξεργαζόμαστε το λεξικό *datapaths* και δημιουργούμε τα μηνύματα **Flow Stats Request** χωρίς κάποιο φίλτρο match. Με αλλά λογία, ζητάμε τα στατιστικά στοιχεία όλων των ροών του μεταγωγέα.

```
def monitor(self):
    self.logger.info("start flow monitor thread")
    while True:
    hub.sleep(10)
    for datapath in self.datapaths.values():
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser
    req = ofp_parser.OFPFlowStatsRequest(datapath)
    datapath.send_msg(req)
```

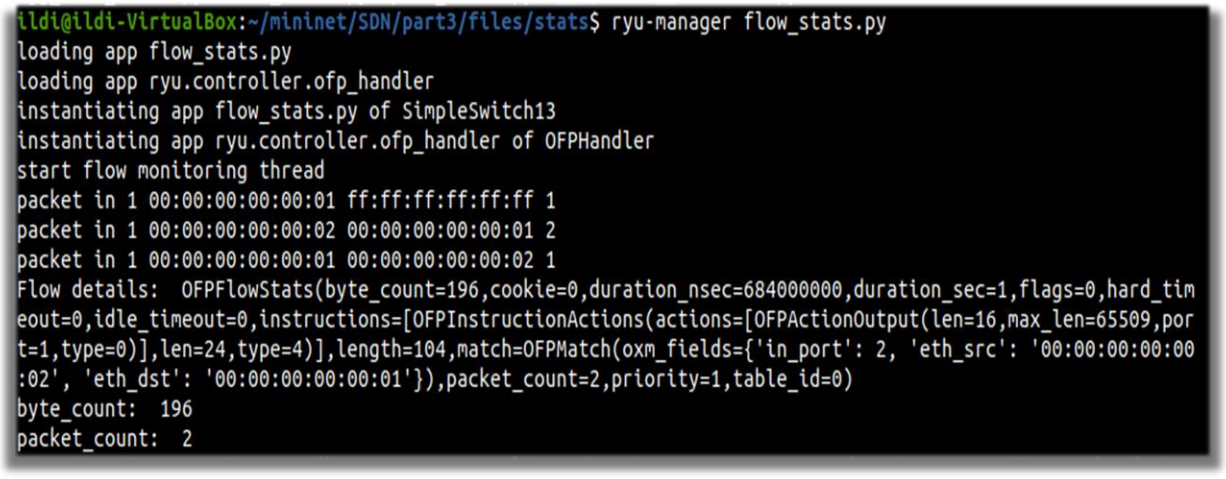
6. Ορισμός του **Flow Stats Response**:

```
@set_ev_cls([ofp_event.EventOFPFlowStatsReply,
             ], MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
for stat in ev.msg.body:
    self.logger.info("Flow details:  %s ", stat)
    self.logger.info("byte_count:  %d ", stat.byte_count)
    self.logger.info("packet_count:  %d ", stat.packet_count)
```

Ο έλεγχος λειτουργικότητας γίνεται με την δημιουργία ενός Mininet Single Topology με 4 Hosts, από όπου θα κάνουμε ping ένα Host. Για την πραγματοποίηση αυτού του σεναρίου χρησιμοποιήθηκαν οι παρακάτω εντολές:

```
sudo mn --controller=remote,ip=127.0.0.1 --mac -i 10.1.1.0/24
--switch=ovsk,protocols=OpenFlow13 --topo=single,4
ryu-manager flow_stats.py
mininet>h1 ping h2
```

Τα αποτελέσματα της εφαρμογής μας φαίνονται στο Ryu Console, Εικόνα 31.



Εικόνα 31: Στατιστικά στοιχεία ροής

Εφαρμογή Aggregate Flow Statistics:

Ο στόχος αυτής της εφαρμογής είναι να μετράει τον συνολικό αριθμό των flows ανά τακτ χρονικά διαστήματα (10 δευτερόλεπτα) και να τον εμφανίζει.

Λογική:

Παρόμοια με την εφαρμογή Flow Statistics, με μονή διαφορά ότι πλέον γίνεται αποστολή του Aggregate Flow Statistics Request

Κώδικας:

I. Παρόμοιος κώδικας, με μόνες διαφορές ότι πλέον ως *req* ορίζεται το ***OFPAggregateStatsRequest*** :

```
def monitor(self):
while True:
    hub.sleep(10)
    for datapath in self.datapaths.values():
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser
    cookie = cookie_mask = 0,match=ofp_parser.OFPMatch()
    req = ofp_parser.OFPAggregateStatsRequest(datapath, 0, ofp
    .OFPTT_ALL, ofp.OFPP_ANY,ofp.OFPG_ANY,cookie, cookie_mask,
     match) datapath.send_msg(req)
```

II. Ως Response ορίζεται το ***EventOFPAggregateStatsReply*** όπως φαίνεται παρακάτω:

```
@set_ev_cls([ofp_event.EventOFPAggregateStatsReply,
            ], MAIN_DISPATCHER)
def stats_reply_handler(self, ev):
result = ev.msg.body
    self.logger.info('AggregateFlowStats  %s', result)
    self.logger.info('FlowCount  %d', result.flow_count)
```

Ο έλεγχος λειτουργικότητας γίνεται με τον ίδιο τρόπο όπως και στο *flow_stats.py* με μονή διαφορά ότι σε αυτή τη δοκιμή χρησιμοποιούμε την εντολή *pingall* στο Mininet.Το αποτέλεσμά φαίνεται στο Ryu Console της Εικόνας 32:

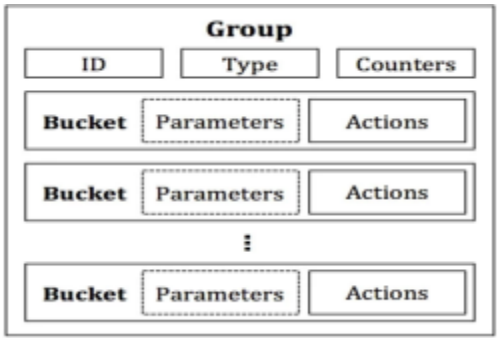
```
AggregateFlowStats  OFPAggregateStats(packet_count=49,byte_count=3514,flow_count=13)
FlowCount  13
```

Εικόνα 32: Αποτέλεσμα εκτέλεσης agg_flow_stats.py

3.3.7. Εφαρμογή Sniffer με χρήση των OpenFlow Group Tables

Τα **OpenFlow Groups** εισήχθησαν στο OpenFlow 1.1 ως ένας τρόπος εκτέλεσης πιο πολύπλοκων λειτουργιών που δεν μπορούν εύκολα να εκτελεστούν μέσω μιας ροής [16]. Κάθε ομάδα λαμβάνει πακέτα στην είσοδο και εκτελεί κάποιες ορισμένες ενέργειες OpenFlow για το εκάστοτε πακέτο. Μια ομάδα δεν είναι σε θέση να εκτελέσει οδηγίες OpenFlow, επομένως δεν μπορεί να στείλει πακέτα σε άλλους πίνακες ροής ή μετρητές.

Επιπλέον, αναμένεται ότι τα πακέτα έχουν αντιστοιχιστεί κατάλληλα πριν από την είσοδο σε μια ομάδα, καθώς οι ομάδες δεν υποστηρίζουν αντιστοίχιση πακέτων. Οι ομάδες είναι απλώς μηχανισμοί για την εκτέλεση προηγμένων ενεργειών ή συνόλων ενεργειών.



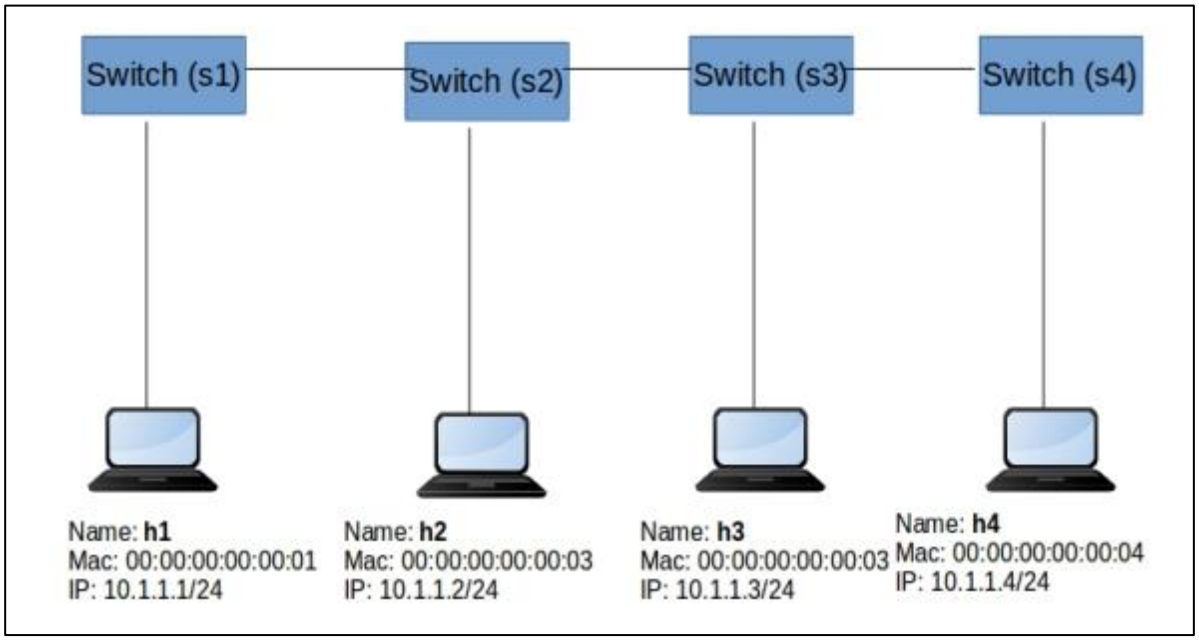
Εικόνα 33: Στοιχεία ενός OpenFlow Group
Πηγή: *[16]*

Όπως φαίνεται στην Εικόνα 33 , η λειτουργικότητας μιας ομάδας πηγάζει από το ότι περιέχει ξεχωριστές λίστες ενεργειών, με κάθε μεμονωμένη λίστα ενεργειών να αναφέρεται ως ένα **OpenFlow Bucket**. Έτσι, μια ομάδα περιέχει μια λίστα από buckets (ή μια λίστα με λίστες ενεργειών). Κάθε bucket ή λίστα από buckets μπορεί να εφαρμοστεί στα εισερχόμενα πακέτα, ενώ η ακριβής συμπεριφορά εξαρτάται από τον τύπο της ομάδας. Υπάρχουν τέσσερις τύποι ομάδων:

- ALL
- SELECT
- INDIRECT
- FAST-FAILOVER

Εφαρμογή Sniffer:

Ο στόχος αυτής της εφαρμογής είναι να υποκλέπτει (***sniff***) όλα τα πακέτα που περνάνε μέσω του μεταγωγέα S2 της παρακάτω τοπολογίας:



Εικόνα 34: Τοπολογία που χρησιμοποιήθηκε για την δοκιμή του Sniffer

Λογική:

Με την χρήση των OpenFlow Groups θα δημιουργήσουμε δυο group tables τύπου *TYPE=ALL*, οπού κάθε πακέτο θα αντιγράφεται στα 2 buckets που θα δημιουργήσουμε και υστέρα θα επεξεργάζεται. Τα groups θα είναι τα:

- Group ***table1*** (Table ID 50) το οποίο θα περιέχει 2 buckets. Το ένα θα προωθεί τα πακέτα στο Port 3, και το άλλο στο Port 1.
- Group ***table2*** (Table ID 51) το οποίο θα περιέχει επίσης 2 buckets με μονή διαφορά ότι το πρώτο bucket θα προωθεί στο Port 2.

Κώδικας:

1. Χρήση του simple_switch_13 ως βάση.
2. Δημιουργία της συνάρτησης group table με την χρήση του **OFPGroupMod API**.

```
def send_group_mod(self, datapath):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    # Group table1
    # Receiver port2, προωθηση στα port1 & Port3
    actions1 = [parser.OFPActionOutput(1)]
    actions2 = [parser.OFPActionOutput(3)]
    buckets = [parser.OFPBucket(actions=actions1),
       parser.OFPBucket(actions=actions2)]
    req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
            ofproto.OFPGT_ALL, 50, buckets)
    datapath.send_msg(req)
    # Group table2
    # Receive Port3, προωθηση στα port1 & Port2
    actions1 = [parser.OFPActionOutput(1)]
    actions2 = [parser.OFPActionOutput(2)]
    buckets = [parser.OFPBucket(actions=actions1),
       parser.OFPBucket(actions=actions2)]
    req = parser.OFPGroupMod(datapath, ofproto.OFPGC_ADD,
            ofproto.OFPGT_ALL, 51, buckets)
    datapath.send_msg(req)
```

3. Κλήση της συνάρτησης Group Table Creation στο Features Handler Function και προσθήκη των flows στον μεταγωγέα S2.

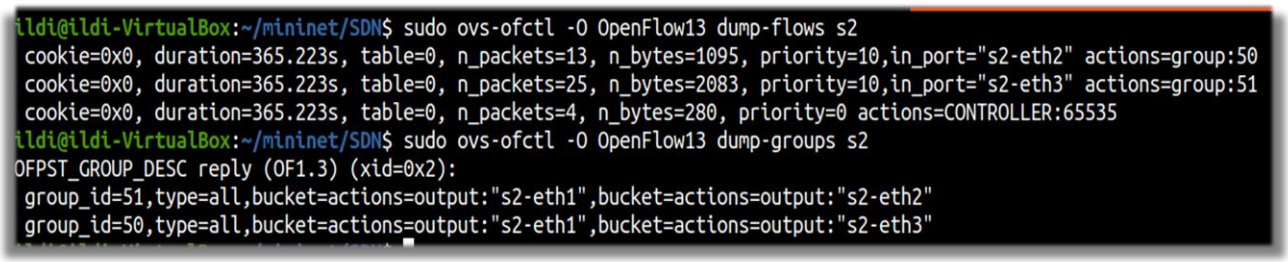
```
# μεταγωγέας s2
if datapath.id == 2:
    self.send_group_mod(datapath)
    actions = [parser.OFPActionGroup(group_id=50)]
    match = parser.OFPMatch(in_port=2)
    self.add_flow(datapath, 10, match, actions)
    actions = [parser.OFPActionGroup(group_id=51)]
    match = parser.OFPMatch(in_port=3)
    self.add_flow(datapath, 10, match, actions)
```

Για τον έλεγχο λειτουργικότητας:

- Αρχικά δημιουργήθηκε το Linear Topology της Εικόνας 35 και έγινε έναρξη του sniffer.py με τις εντολές που βλέπουμε παρακάτω:

```
sudo mn --controller=remote,ip=127.0.0.1 --mac -i
10.1.1.0/24 --switch=ovsk,protocols=OpenFlow13 --
topo=linear,4
ryu-manager sniffer.py
```

- Έλεγχος για την ύπαρξη των group tables & proactive flows του μεταγωγέα S2:



Εικόνα 35: Group Tables & Flows Sniffer Switch

- Έναρξη του sniffer h2 με την χρήση ***tcpdump***:

```
mininet> xterm h2
xterm> tcpdump -i any icmp -v
mininet> pingall
```

- Έλεγχος πακέτων που πέρασαν από τον μεταγωγέα S2:

Στην Εικόνα 36 βλέπουμε ότι ο υπολογιστής h2 λειτουργεί ως sniffer πλέον εφόσον λαμβάνει τα *ICMP echo request/reply* των υπολοίπων hosts.

Εικόνα 36: Πακέτα που υποκλαπήκαν από τον h2

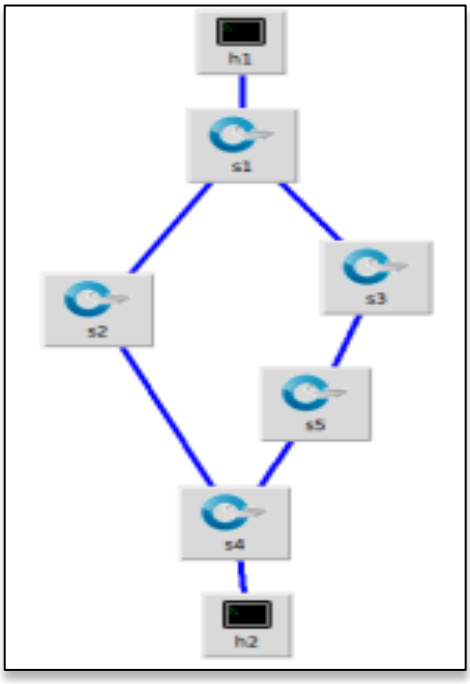
3.4. Εφαρμογή Multipath Load Balancer

Η δρομολόγηση πολλαπλών διαδρομών είναι μια μέθοδος δρομολόγησης που βρίσκει πολλαπλές διαδρομές προς έναν προορισμό σε μια τοπολογία δικτύου. Με την παροχή πολλαπλών διαδρομών για έναν προορισμό γίνεται δυνατή η ίση κατανομή της κυκλοφορίας στο δίκτυο, ένας μηχανισμός γνωστός και ως εξισορροπητή φορτίου, αυξάνοντας έτσι την αποδοτικότητα του δικτύου.

Τα οφέλη μια τέτοιας εφαρμογής είναι η ύπαρξη ενός πιο ισορροπημένου δικτύου, ένα πιθανώς αυξημένο εύρος ζώνης μέσω της παράλληλης μεταφοράς, και αυξημένη ασφάλεια λόγο της παραπάνω δυσκολίας ως προς το sniffing που εισάγουν οι πολλαπλές διαδρομές.

Εύρεση Διαδρομής:

Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκε ο αλγόριθμος **Depth First Search (*DFS)*** [25], ένας αλγόριθμος εύρεσης διαδρομής που χρησιμοποιώντας μια στοίβα διερευνά πιθανές κορυφές σε ένα γράφημα βρίσκοντας πρώτα τη βαθύτερη κορυφή του γραφήματος πριν



Εικόνα 37: Τοπολογία δοκιμής DFS

γυρίσει πίσω για να βρει άλλες πιθανές κορυφές. Το χαρακτηριστικό αυτό (χρήση στοίβας) είναι χρήσιμο στο multipath routing διότι μας δίνει την δυνατότητα να τροποποιήσουμε τον αλγόριθμο και να βρούμε όλες τις πιθανές διαδρομές μεταξύ δυο στοιχείων ενός δικτύου.

Οπότε για την παρακάτω παραδειγματική τοπολογία:

Για την επικοινωνία των h1 και h2 ο αλγόριθμος DFS μπορεί να βρει δυο πιθανά μονοπάτια:

- s1-s2-s4
- s1-s3-s5-s4

Η παραπάνω διαδικασία μπορεί να γραφτεί στον εξής κώδικα:

```
def get_paths(self, src, dst):
    if src == dst:        return [[src]]
    paths = []
    stack = [(src, [src])]
    while stack: (node, path) = stack.pop()
        for next in set(self.adjacency[node].keys()) - set(path):
            if next is dst: paths.append(path + [next])
            else: stack.append((next, path + [next]))
    print "Available paths from ", src, " to ", dst, " : ", paths
    return paths
```

- Τα `src & dst` είναι τα routing targets, δηλαδή οι μεταγωγείς που συνδέουν τους hosts
- Το `self.adjacency` κατέχει το πίνακα γειτνίασης του δικτύου.
- Επιστρέφει μια λίστα `paths`.

Υπολογισμός Path Cost:

Επειδή ο αλγόριθμος DFS επιστρέφει μια μη σταθμισμένη λίστα διαδρομών, πρέπει να είμαστε σε θέση να μετρήσουμε το κόστος μιας διαδρομής. Για την υλοποίηση της λειτουργίας υπολογισμού του path cost χρησιμοποιήθηκε:

$$bw(p) = \left(1 - \frac{pw(p)}{\sum_{i<n}^{i=0} pw(i)}\right) \times 10$$

- Bucket Weight
- Αλγόριθμος τύπου **OSPF** [26]

Ο υπολογισμός του bucket weight γίνεται με την χρήση του παρακάτω τύπου:
Οπού για ένα path **p:**

- ***bw*** είναι το bucket weight, **0≤bw(p)<10**
- ***pw*** είναι το path weight/cost (με τη χρήση του OSPF)
- ***n*** είναι το σύνολο των διαθεσίμων διάδρομων.

Ουσιαστικά βρίσκει την αναλογία του path weight p σε σχέση με το συνολικό path weight των διαθέσιμων διαδρομών. Εφόσον όμως στα OpenFlow Groups η προτεραιότητα ορίζεται με βάση το υψηλότερο bucket weight, ο κώδικας θα δίνει πιο ψηλό bucket weight στις διαδρομές με χαμηλότερο path weight.

Ορίζοντας βαρύτητα 1 σε κάθε link της τοπολογίας της Εικόνας 37 μπορούμε να δοκιμάσουμε τη θεωρία του αλγορίθμου αυτού:

Όπου: pw = path_weight & bw = bucket_weight

pw1=(s4-s2)+(s2-s1)=2

pw2=(s4-s5)+(s5-s3)+(s3-s1)=3

bw1=(1-2/5)*10=6

bw2=(1-3/5)*10=4

Έτσι, όπως ήταν αναμενόμενο, στη μικρότερη διαδρομή (χαμηλότερο pw), αποδίδεται μεγαλύτερο bucket weight.

Ο κώδικας του Path Cost φαίνεται ως έχει:

```
def get_link_cost(self, s1, s2):
    e1 = self.adjacency[s1][s2]
    e2 = self.adjacency[s2][s1]
    bl = min(self.bandwidths[s1][e1], self.bandwidths[s2][e2])
    ew = REFERENCE_BW/bl
    return ew
def get_path_cost(self, path):
    cost = 0
    for i in range(len(path) - 1):
        cost += self.get_link_cost(path[i], path[i+1])
    return cost
def get_optimal_paths(self, src, dst):
    paths = self.get_paths(src, dst)
    paths_count = len(paths) if len(
        paths) &amp;amp;amp;amp;amp;amp;amp;amp;gt; MAX_PATHS
else MAX_PATHS
    return sorted(paths, key=lambda x:
self.get_path_cost(x))[0:(paths_count)]
```

- Το `self.bandwidths` εμπεριέχει το bandwidth του μεταγωγέα
- Το `REFERENCE_BW` είναι το reference bandwidth constant
- Το `MAX_PATHS` ορίζει ένα όριο στον αριθμό των διάδρομων που μπορούν να χρησιμοποιηθούν

Εγκατάσταση Διαδρομών:

Για την εγκατάσταση των διάδρομων που βρέθηκαν στα προηγούμενα βήματα χρησιμοποιήθηκε η εντολή:

```
install_path(self, src, first_port, dst, last_port, ip_src,
ip_dst)
```

Μέσω της εντολής αυτής:

- Παρατίθενται όλες οι διαθέσιμες διαδρομές από τη πηγή προς το προορισμό
- Γίνεται loop σε κάθε μεταγωγέα που περιέχει path/-s
    I. Γίνεται καταχώριση όλων των θυρών του μεταγωγέα που περιέχουν path/-s
    II. Εάν πολλές θύρες στο μεταγωγέα περιέχουν ένα path, δημιουργείται μια ροή group table με τύπου **select (OFPGT_SELECT)** ή αλλιώς γίνεται εγκατάσταση μιας απλής ροής.

Δημιουργία Group Table:

Κατά την δημιουργία των group tables ορίζουμε πολλαπλές ενέργειες εξόδου για κάθε θύρα που περιέχει μια διαδρομή προς το group table όπως φαίνεται και στον παρακάτω κώδικα:

```
buckets = []
   for port, weight in out_ports:
   bucket_weight = int(round((1 - weight/sum_of_pw) * 10))
   bucket_action = [ofp_parser.OFPActionOutput(port)]
   buckets.append(
   ofp_parser.OFPBucket(
   weight=bucket_weight,
   watch_port=port,
   watch_group=ofp.OFPG_ANY,
   actions=bucket_action
   ))
   if group_new:
   req = ofp_parser.OFPGroupMod(
   dp, ofp.OFPGC_ADD, ofp.OFPGT_SELECT, group_id,
   buckets
   )
   dp.send_msg(req)
   else:
   req = ofp_parser.OFPGroupMod(
   dp, ofp.OFPGC_MODIFY, ofp.OFPGT_SELECT,
   group_id, buckets)
   dp.send_msg(req)
```

Packet Processing:

Στα δίκτυα TCP/IP, το πιο σύνηθες είναι το πρώτο πακέτο που αποστέλλεται από έναν κεντρικό υπολογιστή να είναι ένα πακέτο ARP. Στον παρακάτω κώδικα γίνεται η επεξεργασία των πακέτων αυτών:

```
if arp_pkt:
    src_ip = arp_pkt.src_ip
    dst_ip = arp_pkt.dst_ip
    if arp_pkt.opcode == arp.ARP_REPLY:
        self.arp_table[src_ip] = src
        h1 = self.hosts[src]
        h2 = self.hosts[dst]
        out_port = self.install_paths(h1[0], h1[1], h2[0], h2[1], src_ip, dst_ip)
        self.install_paths(h2[0], h2[1], h1[0], h1[1], dst_ip, src_ip) # reverse
    elif arp_pkt.opcode == arp.ARP_REQUEST:
        if dst_ip in self.arp_table:
            self.arp_table[src_ip] = src
            dst_mac = self.arp_table[dst_ip]
            h1 = self.hosts[src]
            h2 = self.hosts[dst_mac]
            out_port = self.install_paths(h1[0], h1[1], h2[0], h2[1], src_ip, dst_ip)
            self.install_paths(h2[0], h2[1], h1[0], h1[1], dst_ip, src_ip)
```

Αρχικά διατηρούμε έναν χάρτη των κεντρικών υπολογιστών στην τοπολογία, με το όνομα `self.hosts`, ο οποίος αντιστοιχίζει τη διεύθυνση MAC κάθε κεντρικού υπολογιστή στον εκάστοτε μεταγωγέα με τον οποίο συνδέεται (προσδιορίζεται με αναγνωριστικό dpid ή datapath στο OpenFlow) και τον αριθμό θύρας που είναι συνδεδεμένος.

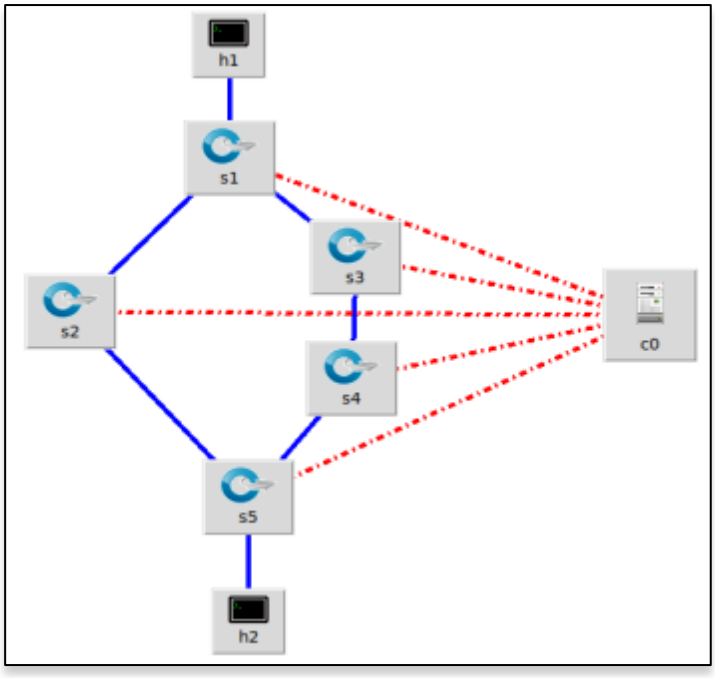
Αρχικοποιούμε το `out_port`, είναι εκεί που καθορίζουμε τη θύρα εξόδου. Ως εφεδρικό ορίσαμε το `ofproto.OFPP_FLOOD`.

Ύστερα με το `arp_pkt` γίνεται εξαγωγή των διευθύνσεων προέλευσης και προορισμού IP του πακέτου και του ARP **opcode**. Μέσο του opcode, μπορούμε να διαφοροποιήσουμε εάν το πακέτο είναι ARP request ή reply. Κάθε φορά που λαμβάνεται ένα ARP reply, αποθηκεύεται η διεύθυνση MAC της πηγή στον πίνακα `self.arp_table`.

3.4.1. Έλεγχος Λειτουργικότητας

Για τον έλεγχο λειτουργικότητας του Multipath:

- Δημιουργήθηκε η παρακάτω τοπολογία με την χρήση του **MiniEdit**.



Εικόνα 38: Τοπολογία Multipath

- Έγινε χρήση *Remote Controller* και *OpenFlow 1.3*
- Εκτέλεση του κώδικα με την χρήση της εντολής:

```
ryu-manager --observe-links multipath.py
```

- Δοκιμή ping ***h1*** → ***h2***.

- Έλεγχος ύπαρξης δύο διαδρομών στον controller. Όπως φαίνεται και στην παρακάτω εικόνα ο ελεγκτής απέδωσε δύο διαδρομές.

```
Available paths from  5  to  1  :  [[5, 4, 3, 1], [5, 2, 1]]
[5, 2, 1] cost =  2.0
[5, 4, 3, 1] cost =  3.0
Path installation finished in  0.0042498111724853516
Available paths from  1  to  5  :  [[1, 3, 4, 5], [1, 2, 5]]
[1, 2, 5] cost =  2.0
[1, 3, 4, 5] cost =  3.0
Path installation finished in  0.003212451934814453
Available paths from  5  to  1  :  [[5, 4, 3, 1], [5, 2, 1]]
[5, 2, 1] cost =  2.0
[5, 4, 3, 1] cost =  3.0
Path installation finished in  0.002660512924194336
Available paths from  1  to  5  :  [[1, 3, 4, 5], [1, 2, 5]]
[1, 2, 5] cost =  2.0
[1, 3, 4, 5] cost =  3.0
Path installation finished in  0.0043179988861083984
```

Εικόνα 39: Διαδρομές που βρέθηκαν

- Έλεγχος ύπαρξης ροών:

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s5
ble=0, n_packets=6887, n_bytes=413220, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTR
ble=0, n_packets=6, n_bytes=588, ip,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=output:"s5-eth3"
ble=0, n_packets=1, n_bytes=42, priority=1,arp,arp_spa=10.0.0.1,arp_tpa=10.0.0.2 actions=output:"s5-eth3"
ble=0, n_packets=6, n_bytes=588, ip,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=group:4224655164
ble=0, n_packets=1, n_bytes=42, priority=1,arp,arp_spa=10.0.0.2,arp_tpa=10.0.0.1 actions=group:4224655164
ble=0, n_packets=29, n_bytes=2690, priority=1,ipv6 actions=drop
ble=0, n_packets=5, n_bytes=238, priority=0 actions=CONTROLLER:65535
```

Εικόνα 40: S5 Flows

Για να φτάσει τον host ***h1*** ο host ***h2*** (διεύθυνση IP 10.0.0.1 και 10.0.0.2) θα πρέπει να γίνει επιλογή μεταξύ των εξερχόμενων θυρών που είναι διαθέσιμες μέσω του μεταγωγέα s5, επόμενος εγκαταστάθηκε ένα group action με το id που βλέπουμε στην Εικόνα 40.

Μπορούμε να δούμε τα actions του group με την εντολή:

```
sudo ovs-ofctl -O OpenFlow dump-groups s5
```

```
ildi@ildi-VirtualBox:~/mininet/SDN$ sudo ovs-ofctl -O OpenFlow13 dump-groups s5
OFPST_GROUP_DESC reply (OF1.3) (xid=0x2):
 group_id=4224655164,type=select,bucket=weight:6,watch_port:"s5-eth2",actions=output:"s5-eth2",
bucket=weight:4,watch_port:"s5-eth1",actions=output:"s5-eth1"
```

Εικόνα 41: Group Actions

Όπως φαίνεται και στην Εικόνα 41, υπάρχουν δυο group actions που προωθούν προς τις θύρες εξόδου:

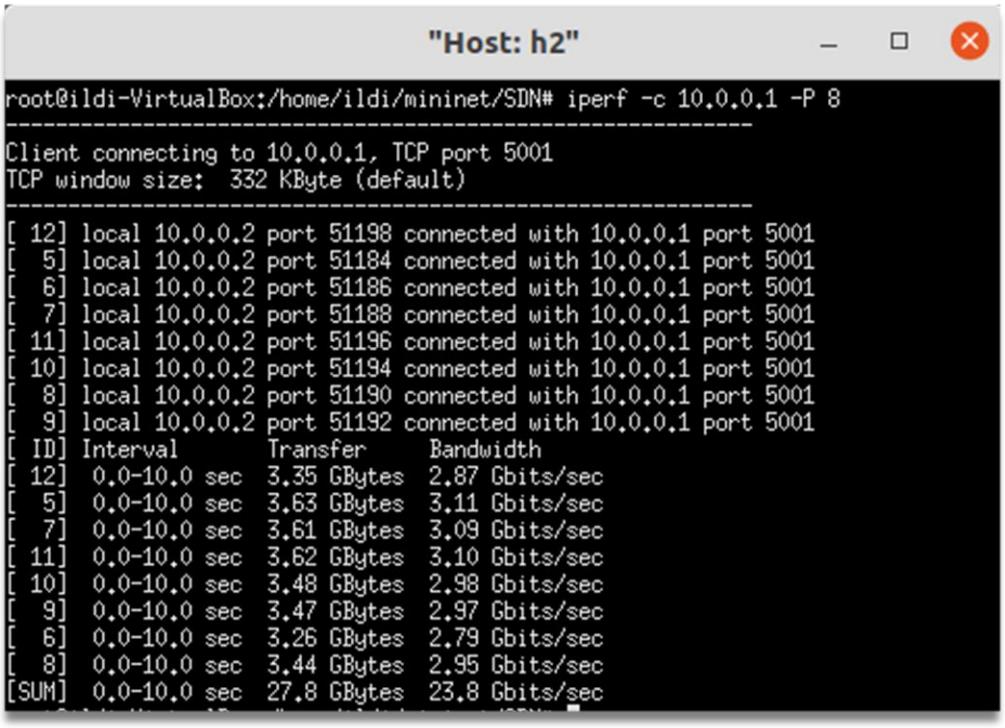
- Το πρώτο action με ***bucket weight 6*** αντιπροσωπεύει την πιο σύντομη διαδρομή **s1→s2→s5**, όπου θύρα εξόδου η ***"s5-eth2"***.
- Το δεύτερο action με ***bucket weight 4*** οδηγεί στην πιο μακρινή διαδρομή **s1→s3→s4→s5** μέσω της θύρας ***"s5-eth1".***

Με την ύπαρξη των actions βεβαιώνεται η σωστή λειτουργία του Multipath Routing.

Έλεγχος λειτουργικότητας Load Balancer:

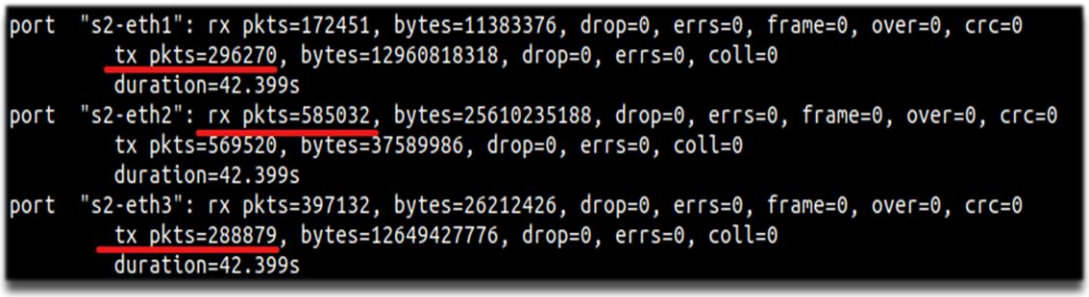
Για τον έλεγχο της ορθής λειτουργίας του load balancer χρειάστηκε να διαπιστωθεί εάν τα πακέτα χωριζόντουσαν ισόνομα και στις δύο διαδρομές. Αυτό έγινε με την χρήση του μηχανισμού `iperf` όπου και ακολουθήθηκαν τα εξής βήματα:

- Ο host ***h1*** ορίστηκε `iperf server` με την χρήση της εντολής:
  `iperf -s`
- Ο host ***h2*** ορίστηκε `iperf client` με 10 parallel clients με την χρήση της εντολής:
  `iperf -c 10.0.0.1 -P 10`



Εικόνα 42: Δοκιμή iperf

- Έλεγχος των πακέτων που στάλθηκαν από τα ports του μεταγωγέα ***s5*** με την χρήση της παρακάτω εντολής:
  `sudo ovs-ofctl -O OpenFlow13 dump-ports s5`



Εικόνα 43: Dump-ports μεταγωγέα S5

Αναλύοντας το output της Εικόνας 43 βλέπουμε τις εξής ροές πακέτων:

- **Port 1 (tx, transmit): 296270**
- **Port 2 (tx): 288879**
- **Port 3 (rx, receive): 585032 =~ Port 1 tx + Port 2 tx**

Η μικρή διαφορά στα tx των ***Port 1 & 2*** προκύπτει από τον τρόπο λειτουργίας του Open vSwitch [27]. Τα δεδομένα μια ροής (το packet header hash) κατακερματίζεται με το bucket ID, με αποτέλεσμα έναν ακέραιο που στη συνέχεια πολλαπλασιάζεται με το βάρος του bucket. Ύστερα επιλέγεται το bucket με την υψηλότερη «βαθμολογία». Αυτό σημαίνει ότι η επιλογή του bucket εξαρτάται σε μεγάλο βαθμό από τον κατακερματισμό των κεφαλίδων και το βάρος του bucket μπορεί να μην επηρεάσει σημαντικά την τελική επιλογή.

Γνωρίζοντας τα παραπάνω και ελέγχοντας την αναλογία του αριθμού πακέτων των δύο θυρών και των bucket weights προκύπτει πως ο μηχανισμός του Load Balancer λειτουργεί επιτυχώς.

3.5. Ανάλυση Απόδοσης

Για την αξιολόγηση της απόδοσης της εφαρμογής έγινε χρήση της τεχνολογίας **sFlow-RT** [28]. Ο μηχανισμός ανάλυσης sFlow-RT λαμβάνει μια συνεχή ροή τηλεμετρίας από τις συσκευές δικτύου μέσω του REST API και παρέχει εργαλεία για την real-time παρακολούθηση του δικτύου. Παράλληλα η δημιουργία κίνησης επιτευχθεί με την χρήση του **D-ITG** (Distributed Internet Traffic Generator) [29].

Συνολικά εξετάστηκαν τρείς διαφορετικές τοπολογίες (**Linear, Tree, Ring**) υπό 2 προσομοιωμένα σενάρια:

I. **Καμία παράμετρος**
II. **7Mbit bandwidth – Max Queue Size 1000 packets – 10 MS delay – 15% loss**

Σε κάθε σενάριο προσομοιώθηκαν 20 ροές μέσω του D-ITG script ***send.sh***. Πιο αναλυτικά:

- `5 x TCP -a 10.0.0.x -c 1024 -t 60000`
- `5 x TCP -a 10.0.0.x -c 1024 -t 60000`
- `5 x UDP -a 10.0.0.x -c 1024 -t 60000`
- `5 x UDP -a 10.0.0.x -c 1024 -t 60000`

Όπου:

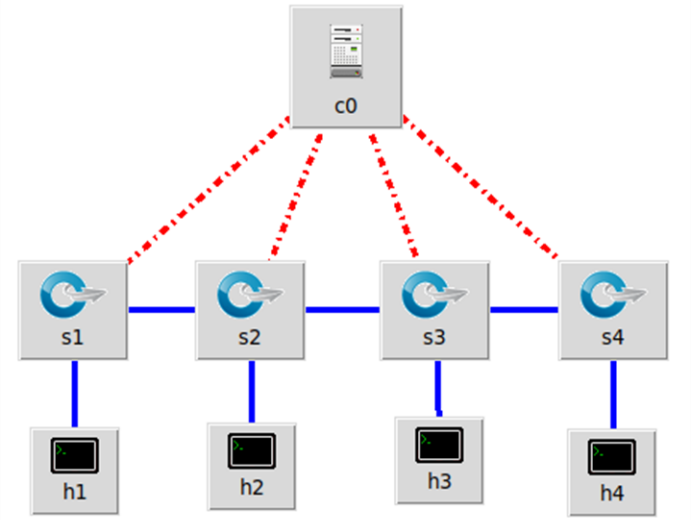
- TCP/UDP = protocol
- -a = ip_destination
- -c = packet size
- -t = time in millisecond
- -x = host

3.5.1. Αξιολόγηση Linear Τοπολογίας

Η τοπολογία που δημιουργήθηκε για το πείραμα αυτό, όπως φαίνεται στην Εικόνα 42, περιέχει 4 μεταγωγείς σε σειριακή σύνδεση με έναν host έκαστος. Για τον ορισμό της τοπολογίας χρησιμοποιήθηκε η παρακάτω εντολή:

```
sudo mn –custom sflow-rt/extras/sflow.py --topo=linear,4
–controller remote,ip=127.0.0.1:6633
```

Η παράμετρος **sflow-rt/extras/sflow.py** χρησιμοποιήθηκε για την ενσωμάτωση του εξομοιωτή με τον μηχανισμό sFlow.



Εικόνα 44: Linear Topology

Για την εκτέλεση της δοκιμής χρειάστηκε να τρέχει παράλληλα ο **sFlow Server** και ο Controller με την χρήση των εντολών:

```
./sflow-rt/starts.sh
```

```
ryu-manager --observe-links
multipath.py,ryu.app.ofctl_rest
```

Σενάριο Ι:

1. Αρχικά έγινε εκκίνηση της τοπολογίας και του controller.
2. Ορίστηκαν οι αποδέκτες με την χρήση των παρακάτω εντολών όπως φαίνεται στην Εικόνα 45 :

```
mininet>xterm h3: ITGRecv -l /tmp/h3NoPar.log
```

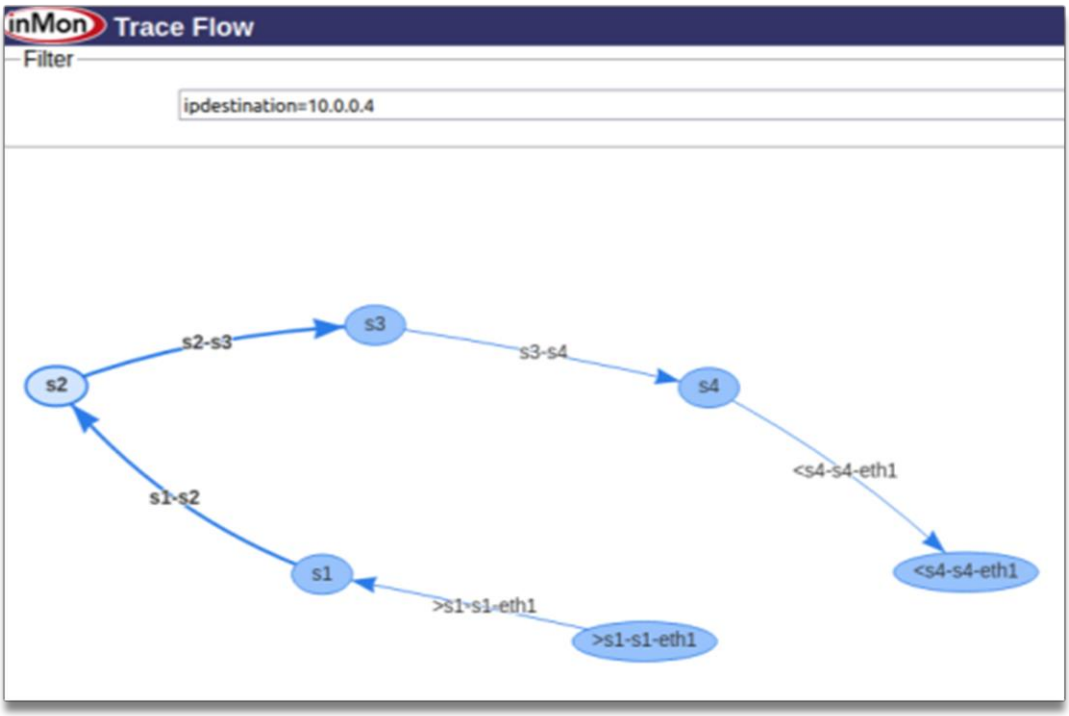
```
mininet>xterm h4: ITGRecv -l /tmp/h4NoPar.log
```

3. Έναρξη ροών μέσω του ***send.sh***

```
mininet>h1 ITGSend /tmp/send.sh
```

Εικόνα 45: D-ITG Listeners

Μπορούμε να ελέγξουμε την λειτουργικότητα της διάταξης μέσο του εργαλείου Trace Flow, από όπου και βλέπουμε πως ο μηχανισμός sFlow έχει επίγνωση της τοπολογίας όπως φαίνεται και στην Εικόνα 46.

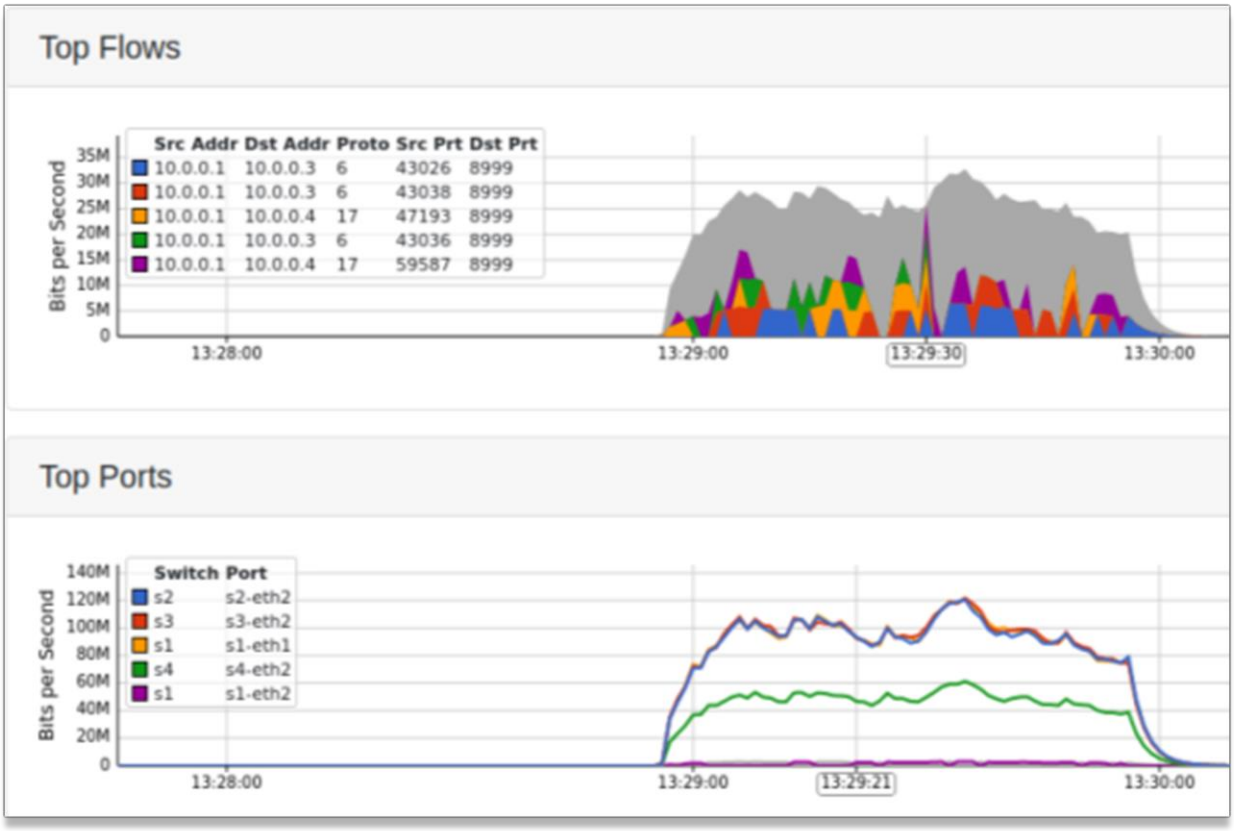


Εικόνα 46: Trace Flow

Αποτελέσματα:

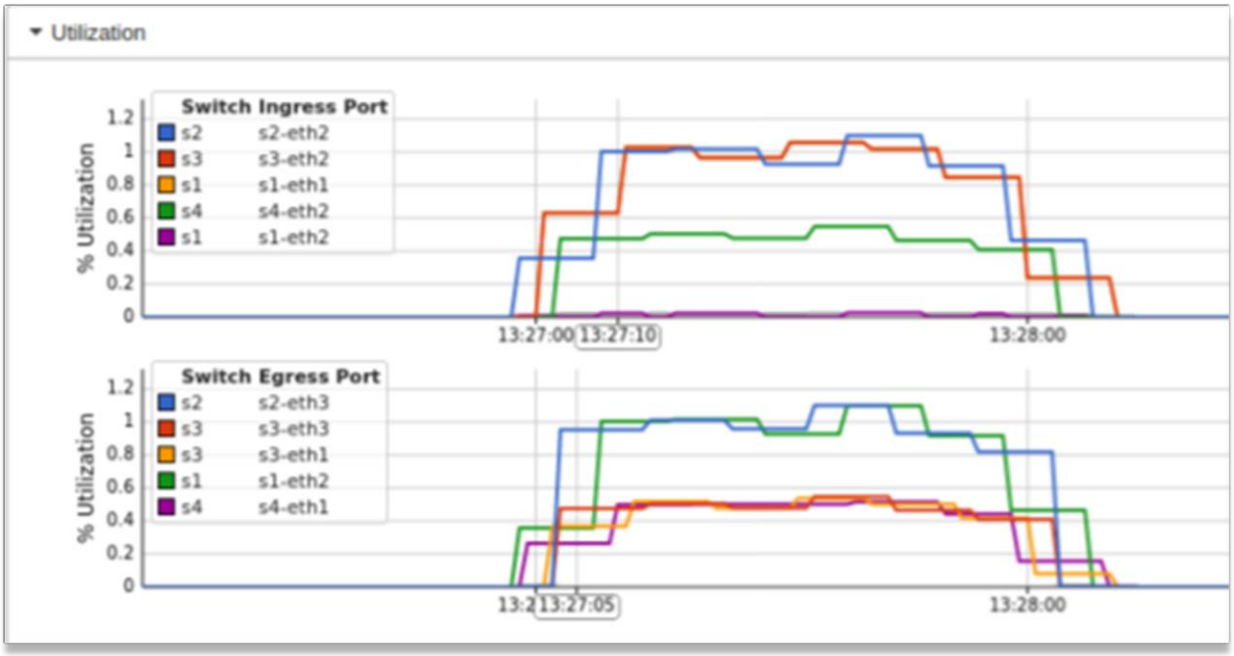
Μέσο του εργαλείου sFlow ήταν δυνατή η ζωντανή παρακολούθηση της κατάστασης του δικτύου. Από την συγκεκριμένη δοκιμή αντλήθηκαν τα γραφήματα:

I. **Top Flows & Top Ports Graph:** Από το γράφημα αυτό αντλούμε πληροφορίες σχετικά με την κίνηση των ροών και των port σε Bits per Second.



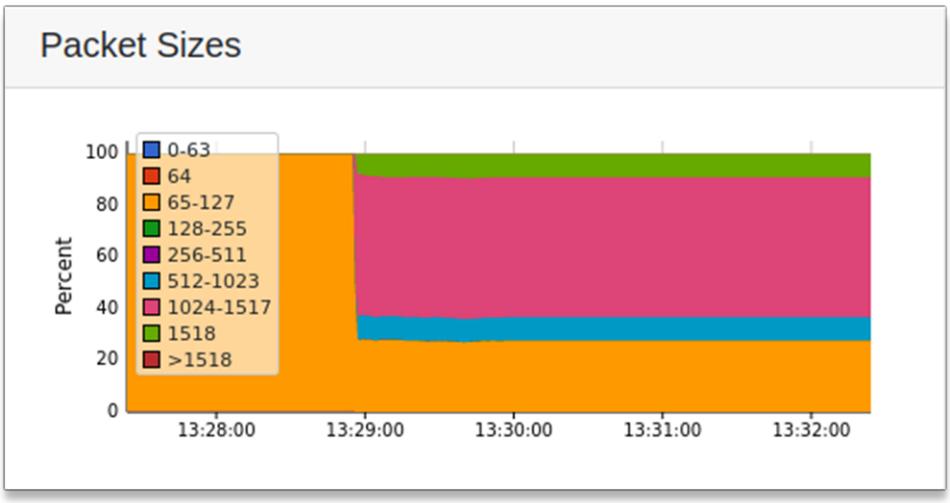
Εικόνα 47: Top Flows & Top Ports

II. **Port Utilization:** Στο συγκεκριμένο γράφημα βλέπουμε το ποσοστό αξιοποίηση κάθε πόρτας.



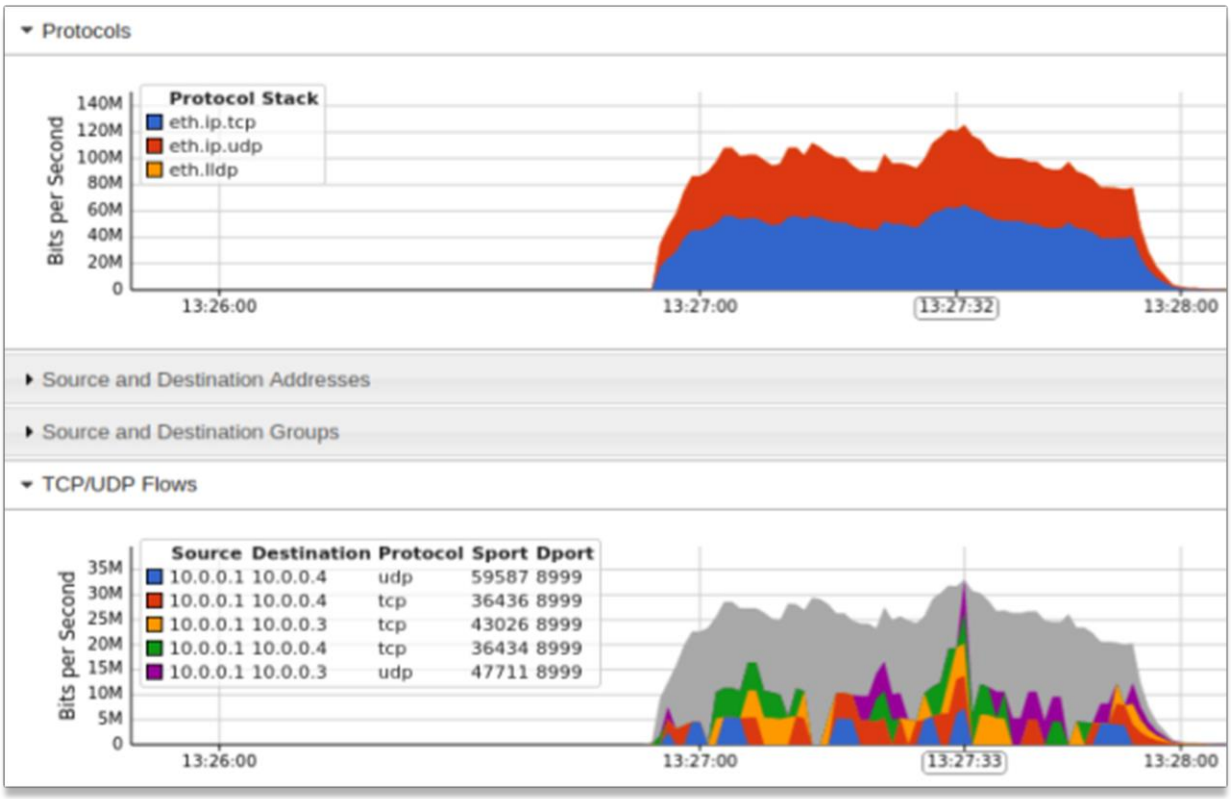
Εικόνα 48: Port Utilization

III. **Packet Sizes:** Ανάλυση του μεγέθους των πακέτων.



Εικόνα 49: Packet Sizes

IV. **Protocols:** Ανάλυση του είδους των ροών (TCP/UDP).



Εικόνα 50: Protocol Stack

Μία επιπλέον πηγή πληροφορίων είναι τα D-ITG output log των ***h3*** & ***h4*** όπως φαίνεται στην Εικόνα 51 & Εικόνα 52.

V. Host H3 Log:

```
****************  TOTAL RESULTS   ******************

Number of flows          =            10
Total time               =     60.127226 s
Total packets            =        335823
Minimum delay            =      0.000006 s
Maximum delay            =      0.028181 s
Average delay            =      0.000405 s
Average jitter           =      0.000534 s
Delay standard deviation =      0.000902 s
Bytes received           =     343882752
Average bitrate          =  45754.015261 Kbit/s
Average packet rate      =   5585.206941 pkt/s
Packets dropped          =            18 (0.01 %)
Average loss-burst size  =      0.000000 pkt
Error lines              =             0
```

Εικόνα 51: Host h3 D-ITG Log

VI. Host H4 Log:

```
****************  TOTAL RESULTS   ******************

Number of flows          =            10
Total time               =     60.088581 s
Total packets            =        333868
Minimum delay            =      0.000006 s
Maximum delay            =      0.037729 s
Average delay            =      0.000412 s
Average jitter           =      0.000541 s
Delay standard deviation =      0.000930 s
Bytes received           =     341880832
Average bitrate          =  45516.912040 Kbit/s
Average packet rate      =   5556.263677 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =             0 pkt
Error lines              =             0
```

Εικόνα 52: Host h4 D-ITG Log

Από τα αποτελέσματα των logs προκύπτει ότι συνολικά στάλθηκαν **669,691 Kbyte** με μέσο ρυθμό μετάδοσης **5,704.4 Kbyte/s.**

Η μέση καθυστέρηση βρίσκεται στα **0.4 ms**, ενώ χάθηκε μόνο το **0,01%** των πακέτων του Host ***h3.***

Σενάριο ΙΙ:

Για την δοκιμή αυτή χρειάστηκε να γίνει προσθήκη των παραμέτρων

- **7Mbit bandwidth**
- **Max Queue Size 1000 packets**
- **10 MS delay**
- **15% loss**

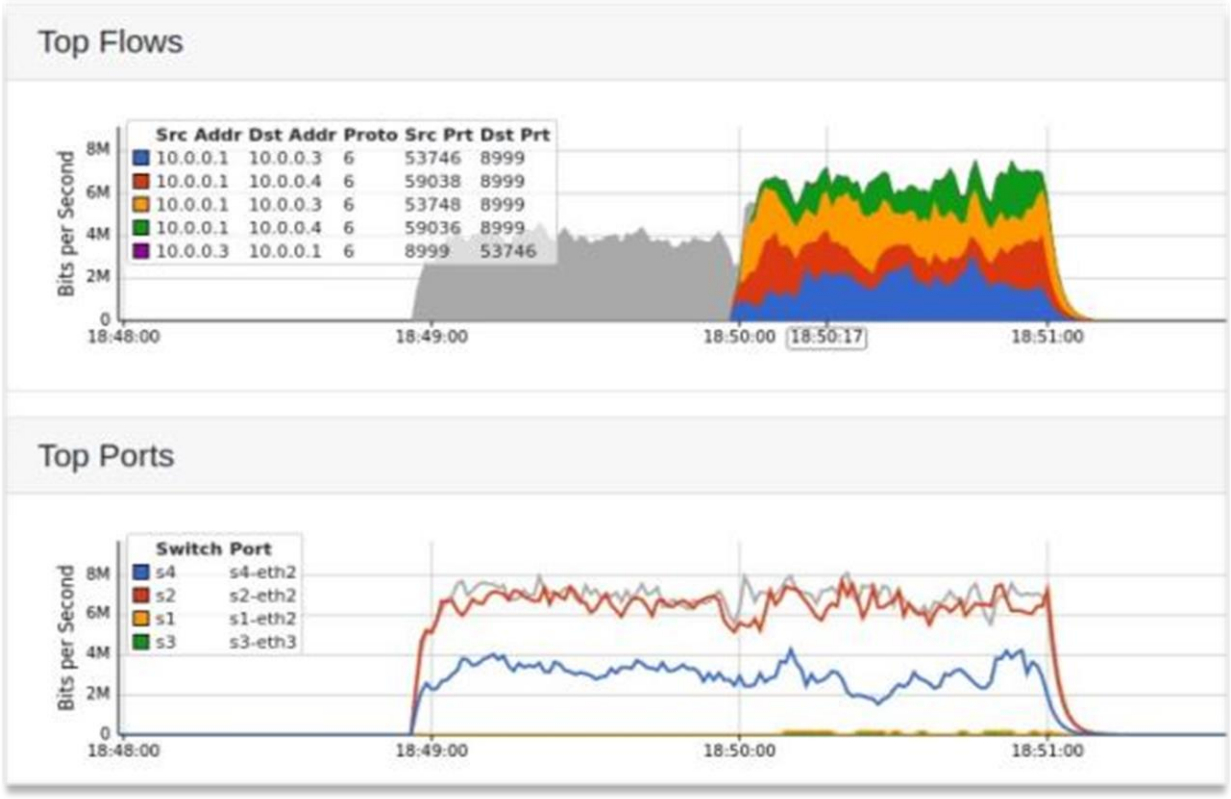
Τα βήματα που ακολουθήθηκαν ήταν παρόμοια με το **Σενάριο Ι**, με μόνη διαφορά την εντολή δημιουργίας της τοπολογίας:

```
sudo mn –custom sflow-rt/extras/sflow.py --link
tc,bw=7,delay=10,loss=0.15,max_queue_size=1000 --
topo=linear,4 –controller remote,ip=127.0.0.1:6633
```

Αποτελέσματα:

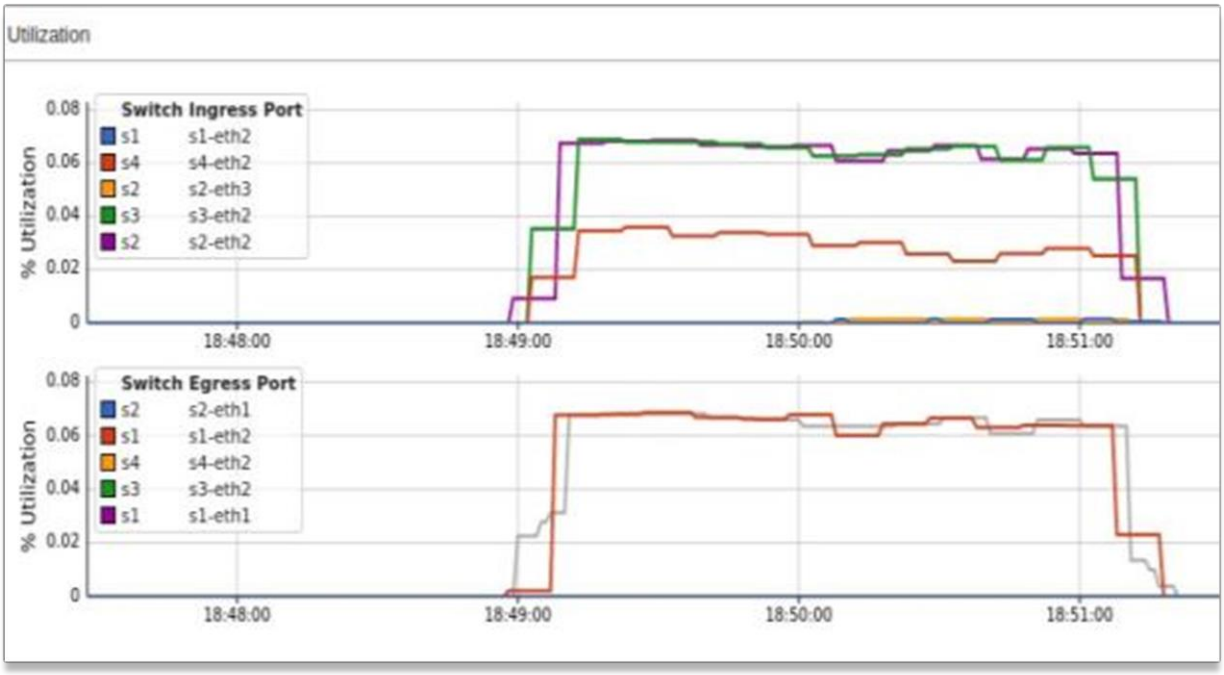
Μέσο του εργαλείου sFlow ήταν δυνατή η ζωντανή παρακολούθηση της κατάστασης του δικτύου. Από την συγκεκριμένη δοκιμή αντλήθηκαν τα γραφήματα:

I. Top Flows & Top Ports Graph:



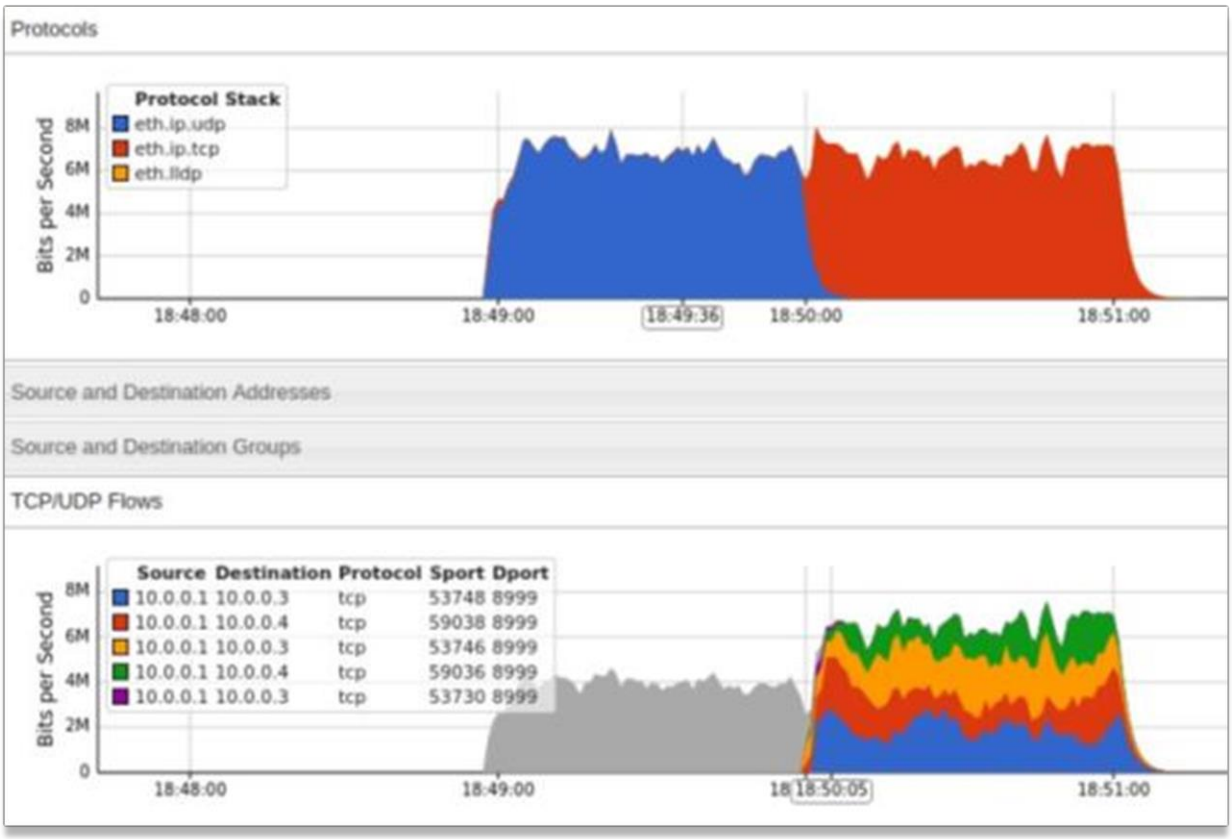
Εικόνα 53: Top Flows & Ports

II. Port Utilization:



Εικόνα 54: Port Utilization

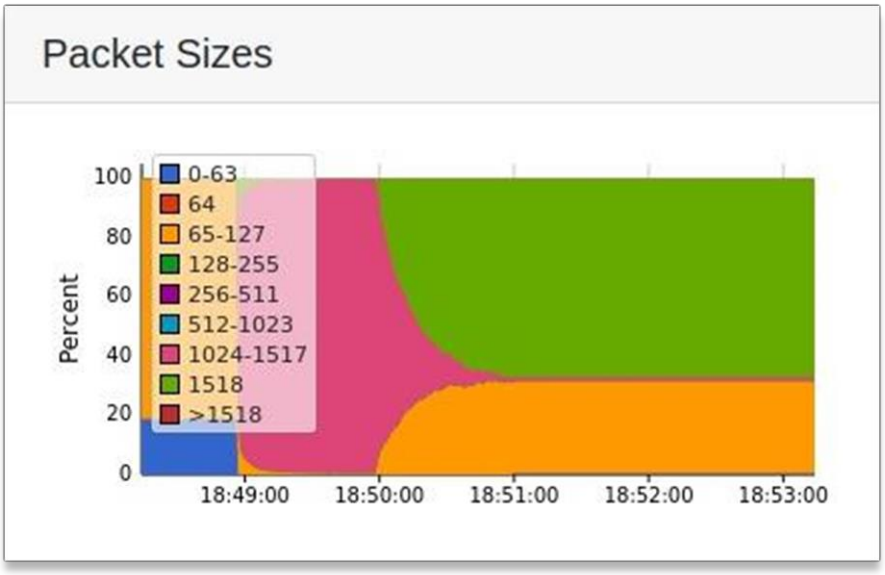
III. Protocols:



Εικόνα 55: Protocol Stack

IV. Packet Sizes:

Αξιοσημείωτη είναι διαφορά στην σειρά αποστολής των πακέτων TCP/UDP. Στο **Σενάριο I**



Εικόνα 56: Packet Sizes

παρατηρήθηκε μια παράλληλη αποστολή των flows ενώ στο **Σενάριο II** βλέπουμε μια διαφοροποίηση με πρώτα να στέλνονται τα πακέτα UDP και ύστερα τα πακέτα TCP.

V. Host H3 Log:

```
************************  TOTAL RESULTS  ************************

Number of flows          =            10
Total time               =    122.839540 s
Total packets            =         44069
Minimum delay            =      0.000325 s
Maximum delay            =     64.338545 s
Average delay            =      1.526452 s
Average jitter           =      0.030861 s
Delay standard deviation =      3.798326 s
Bytes received           =      45126656
Average bitrate          =   2938.901009 Kbit/s
Average packet rate      =    358.752565 pkt/s
Packets dropped          =        161571 (78.57 %)
Average loss-burst size  =      0.000038 pkt
Error lines              =             0
```

Εικόνα 57: Host H3 D-ITG Log

VI. Host H4 Log:

```
****************  TOTAL RESULTS   ******************

Number of flows          =            10
Total time               =    123.701075 s
Total packets            =         51333
Minimum delay            =      0.025704 s
Maximum delay            =     63.690718 s
Average delay            =      1.708741 s
Average jitter           =      0.029987 s
Delay standard deviation =      4.123457 s
Bytes received           =      52564992
Average bitrate          =   3399.484895 Kbit/s
Average packet rate      =    414.976184 pkt/s
Packets dropped          =        161696 (75.90 %)
Average loss-burst size  =      0.000038 pkt
Error lines              =             0
```

Εικόνα 58: Host H4 D-ITG Log

Από τα αποτελέσματα των παραπάνω logs μπορούμε να συμπεράνουμε πως η κίνηση υπερβαίνει τις δυνατότητες του δικτύου. Η μετάδοση των UDP πακέτων ολοκληρώθηκε με σημαντική καθυστέρηση, ενώ χάθηκε το **78.57%** & **75.90%** των απεσταλμένων πακέτων.

Συμπέρασμα:

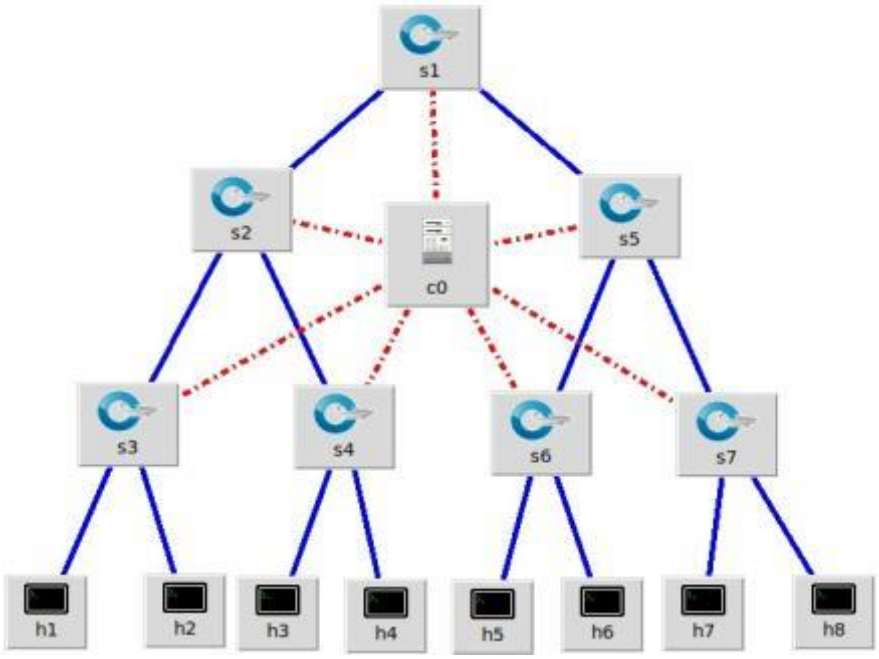
Όπως ήταν αναμενόμενο η τοπολογία στην οποία περιορίσαμε τις δυνατότητες, μέσω των παραμέτρων, απέδωσε πολύ χειρότερα από την κανονική. Συγκεκριμένα παρατηρήθηκαν μεγάλες διαφορές σε:

- Ρυθμός μετάδοσης, μείωση **94%**.
- Συνολικός αριθμός μεταδομένων bytes, μείωση **86%**.
- Μέση καθυστέρηση, αύξηση **4000%**.
- Χαμένα πακέτα, αύξηση **77%**.

3.5.2. Αξιολόγηση Tree Τοπολογίας

Η τοπολογία που δημιουργήθηκε για το πείραμα αυτό, όπως φαίνεται στην Εικόνα 59, περιέχει 7 μεταγωγείς σε τοπολογία τύπου δέντρου και 8 hosts, ενώ πλέον D-ITG Receivers ορίστηκαν οι hosts ***h6*** & ***h7***. Για τον ορισμό της τοπολογίας χρησιμοποιήθηκε η παρακάτω εντολή:

```
sudo mn –custom sflow-rt/extras/sflow.py --topo=tree,3,2
–controller remote,ip=127.0.0.1:6633
```

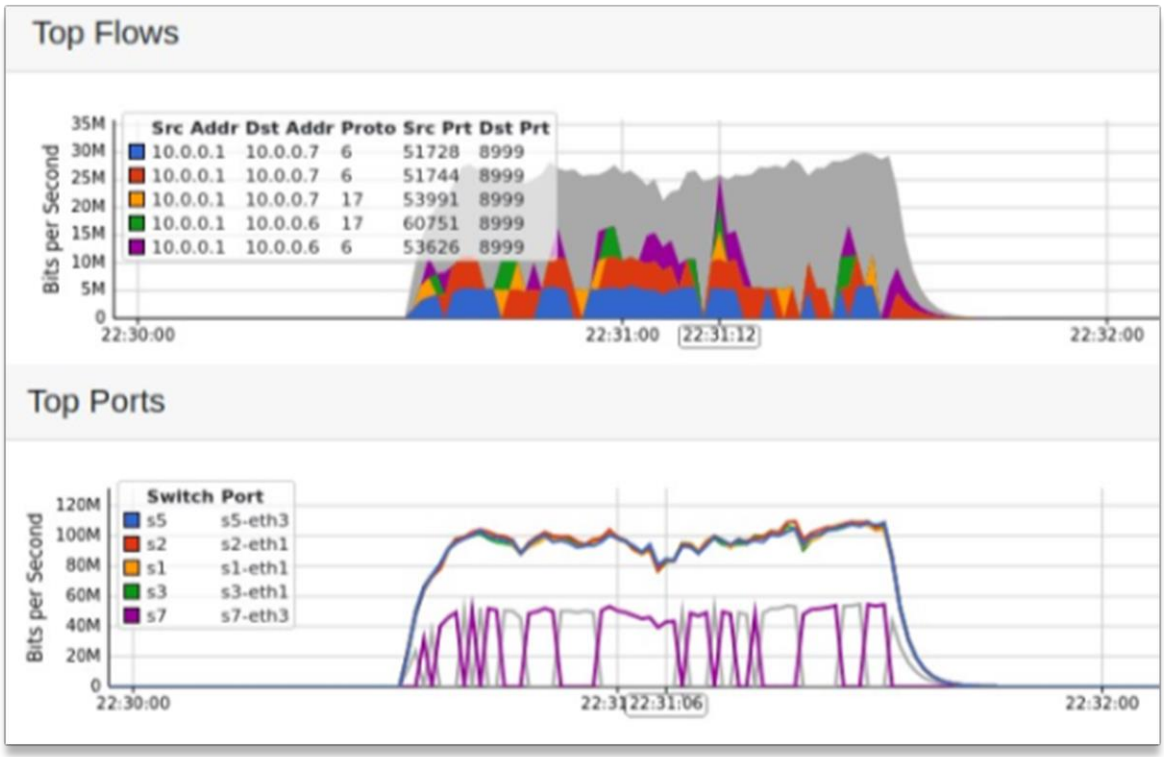


Εικόνα 59: Tree Topology

Αποτελέσματα Σεναρίου I:

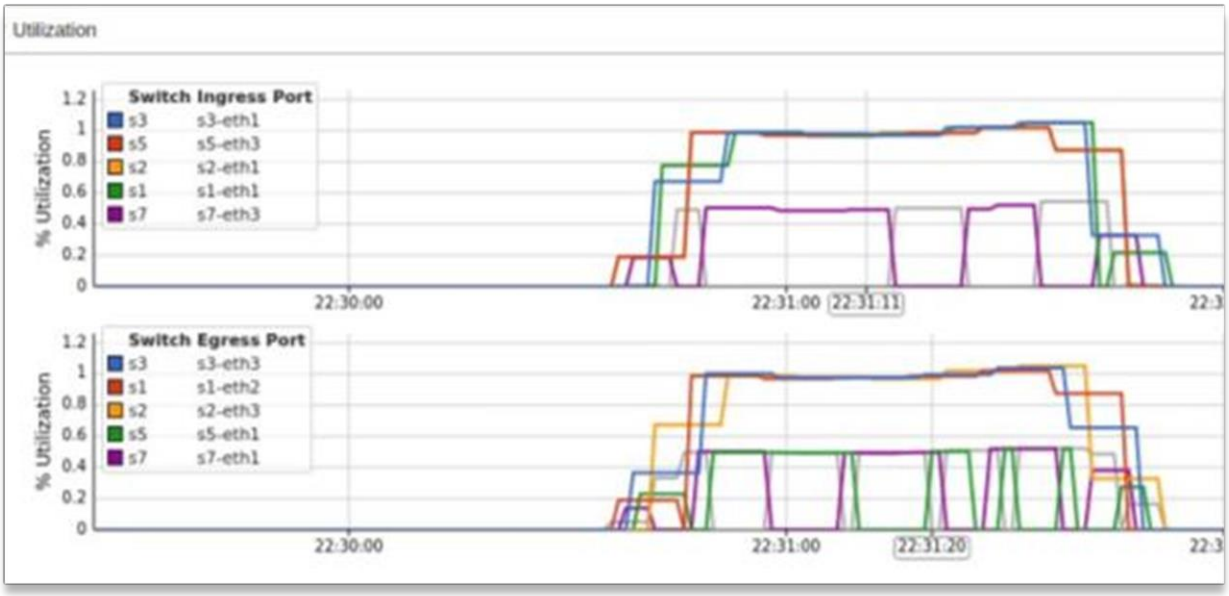
Από την παραπάνω δοκιμή πήραμε τα παρακάτω γραφήματα:

I. Top Flows & Tops Ports Graph:



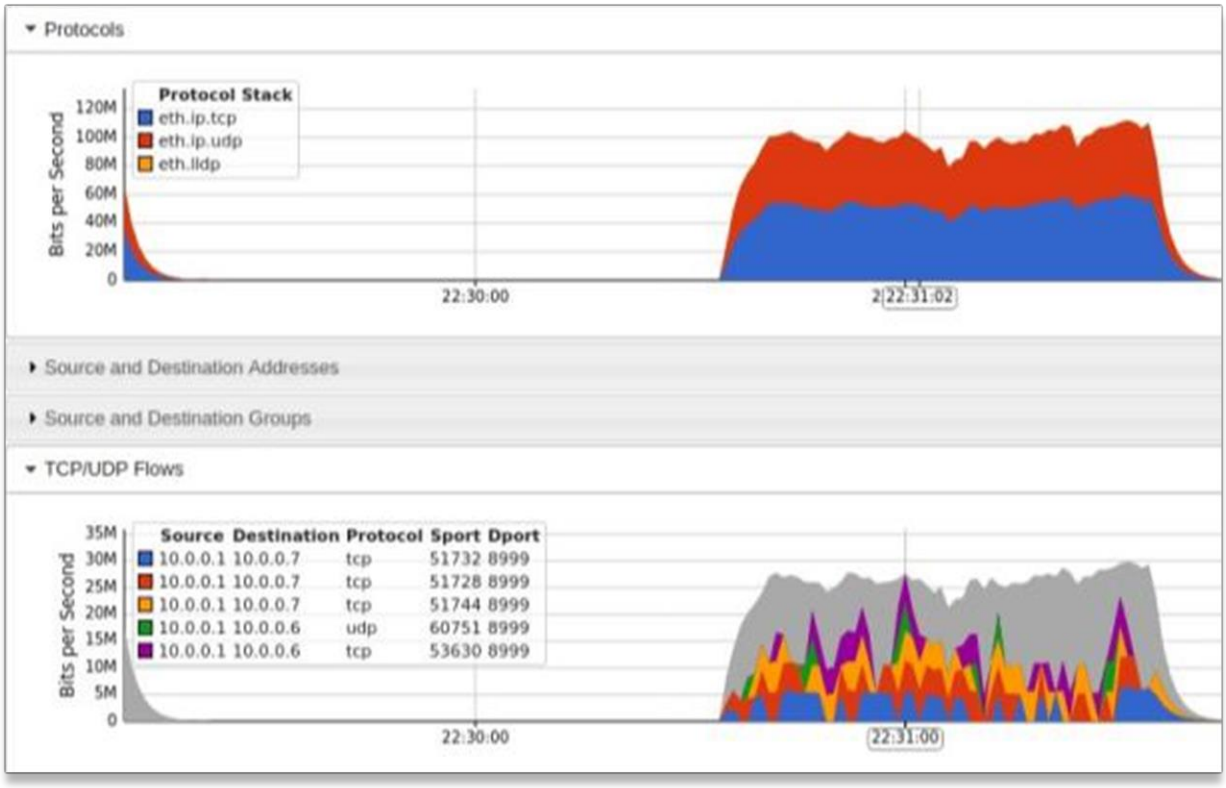
Εικόνα 60: Tree Top Flows & Ports

II. Port Utilization:



Εικόνα 61: Tree Port Utilization Tree

III. Protocols:



Εικόνα 62: Tree Protocol Stack

IV. Host H6 Log:

```
****************  TOTAL RESULTS   ******************

Number of flows          =            10
Total time               =     60.217185 s
Total packets            =        347475
Minimum delay            =      0.000009 s
Maximum delay            =      0.504231 s
Average delay            =      0.000863 s
Average jitter           =      0.000609 s
Delay standard deviation =      0.009258 s
Bytes received           =     355814400
Average bitrate          =  47270.811480 Kbit/s
Average packet rate      =   5770.362730 pkt/s
Packets dropped          =            47 (0.01 %)
Average loss-burst size  =      0.000000 pkt
Error lines              =             0
```

Εικόνα 63: Host h6 D-ITG Log

V. Host H7 Log:

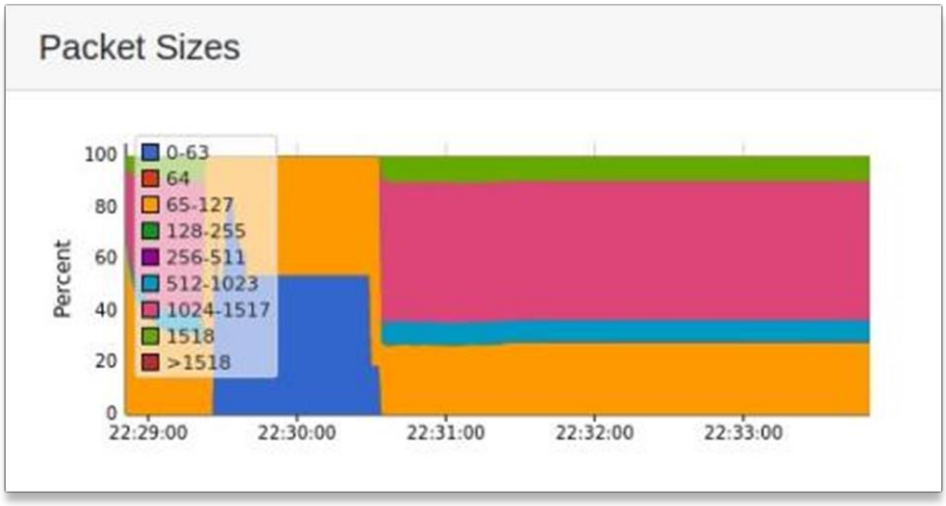
```
****************  TOTAL RESULTS   ******************

Number of flows          =            10
Total time               =     60.164128 s
Total packets            =        349098
Minimum delay            =      0.000010 s
Maximum delay            =      0.504098 s
Average delay            =      0.000974 s
Average jitter           =      0.000611 s
Delay standard deviation =      0.011975 s
Bytes received           =     357476352
Average bitrate          =  47533.487330 Kbit/s
Average packet rate      =   5802.427653 pkt/s
Packets dropped          =            554 (0.16 %)
Average loss-burst size  =      0.000000 pkt
Error lines              =             0
```

Εικόνα 64: Host h7 D-ITG Log

Συγκρίνοντας τα logs των hosts ***h6*** & ***h7*** με τα logs των hosts ***h3*** & ***h4*** της Linear τοπολογίας, Εικόνα 51 και Εικόνα 52, μπορούμε να δούμε πως πλην μιας μικρής διαφοράς στα dropped packets η απόδοση είναι παρόμοια. Παρόμοια είναι και τα μεγέθη των απεσταλμένων πακέτων, όπως φαίνεται στην παρακάτω εικόνα.

VI. Packet Sizes



Εικόνα 65: Tree Packet Sizes

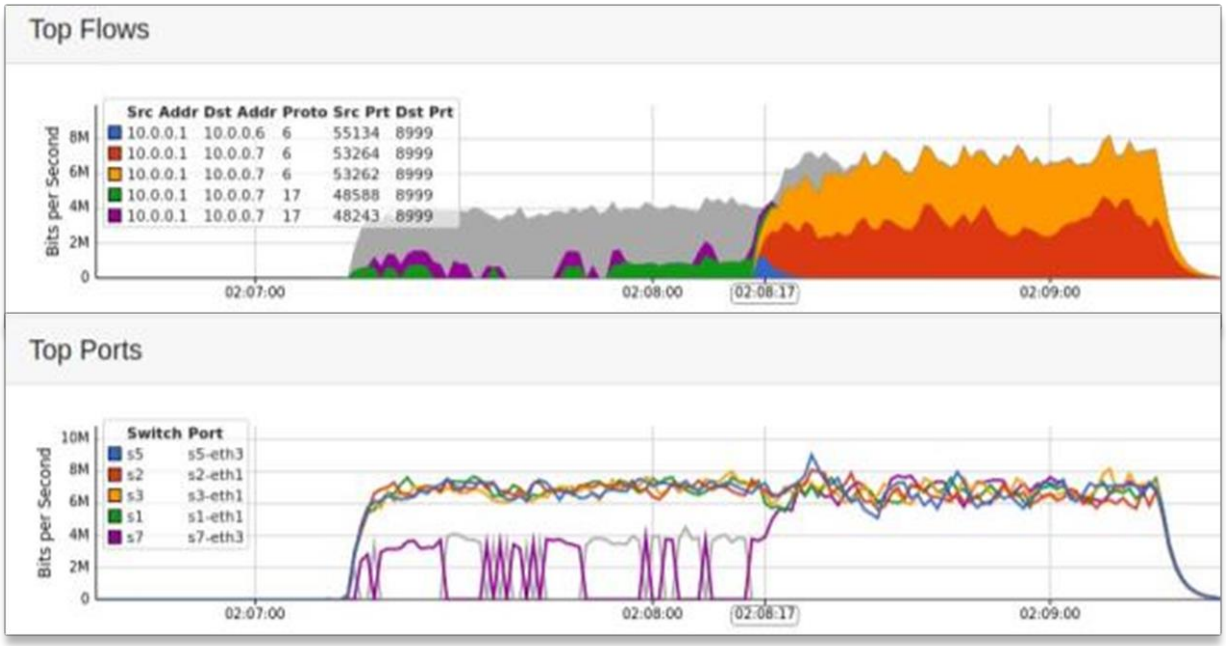
Σενάριο ΙΙ:

Η διαδικασία που ακολουθήθηκε για το σενάριο αυτό ήταν παρόμοια με το **Σενάριο ΙΙ** της Linear τοπολογίας, ενώ για τον ορισμό της τοπολογίας χρησιμοποιήθηκε η παρακάτω εντολή:

```
sudo mn –custom sflow-rt/extras/sflow.py --link
tc,bw=7,delay=10,loss=0.15,max_queue_size=1000 --
topo=tree,3,2 –controller remote,ip=127.0.0.1:6633
```

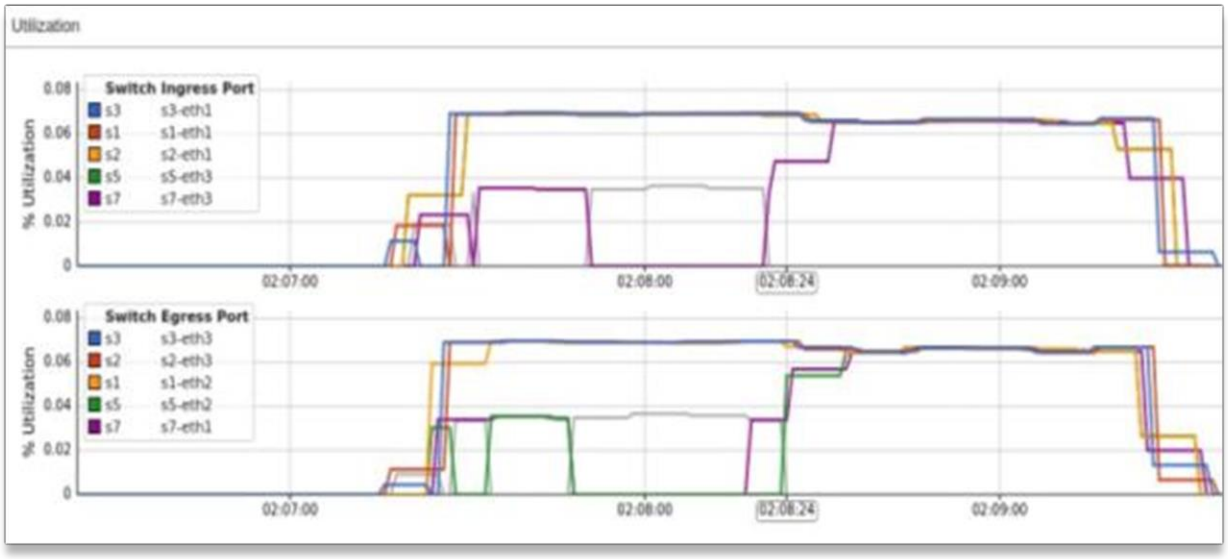
Αποτελέσματα:

I. Top Flows & Top Ports Graph:



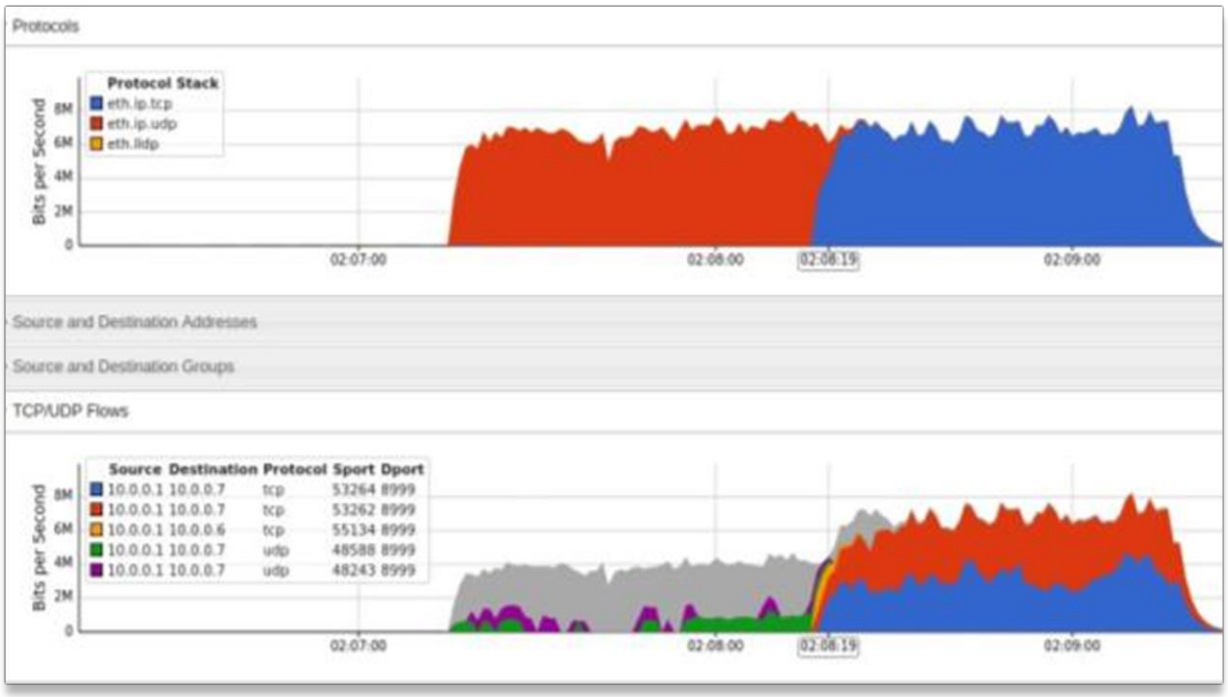
Εικόνα 66: Tree Top Flows & Ports

II. Port Utilization:



Εικόνα 67: Tree Port Utilization

III. Protocols:



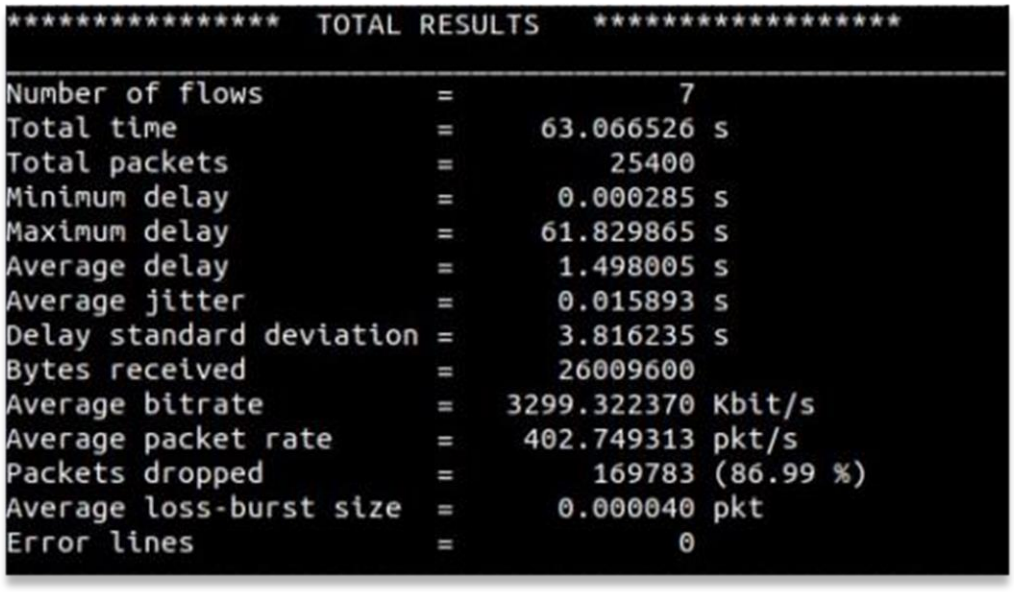
Εικόνα 68: Tree Protocol Stack

Συγκρίνοντας τα γραφήματα των Linear & Tree Σεναρίων I & II συμπεραίνουμε πως και οι δυο τοπολογίες συμπεριφέρονται με τον ίδιο τρόπο ως προς τα διαφορετικά σενάρια. Συγκεκριμένα παρατηρείται μια **αλλαγή στην σειρά αποστολής των flows** μεταξύ Σεναρίου I (**παράλληλη αποστολή**) και Σεναρίου II (**πρώτα τα UDP και ύστερα τα TCP**).

IV. Host H6 Log:

```
**********  TOTAL RESULTS  **********

Number of flows          =          7
Total time               =  63.066526 s
Total packets            =      25400
Minimum delay            =   0.000285 s
Maximum delay            =  61.829865 s
Average delay            =   1.498005 s
Average jitter           =   0.015893 s
Delay standard deviation =   3.816235 s
Bytes received           =   26009600
Average bitrate          = 3299.322370 Kbit/s
Average packet rate      =  402.749313 pkt/s
Packets dropped          =     169783 (86.99 %)
Average loss-burst size  =   0.000040 pkt
Error lines              =          0
```

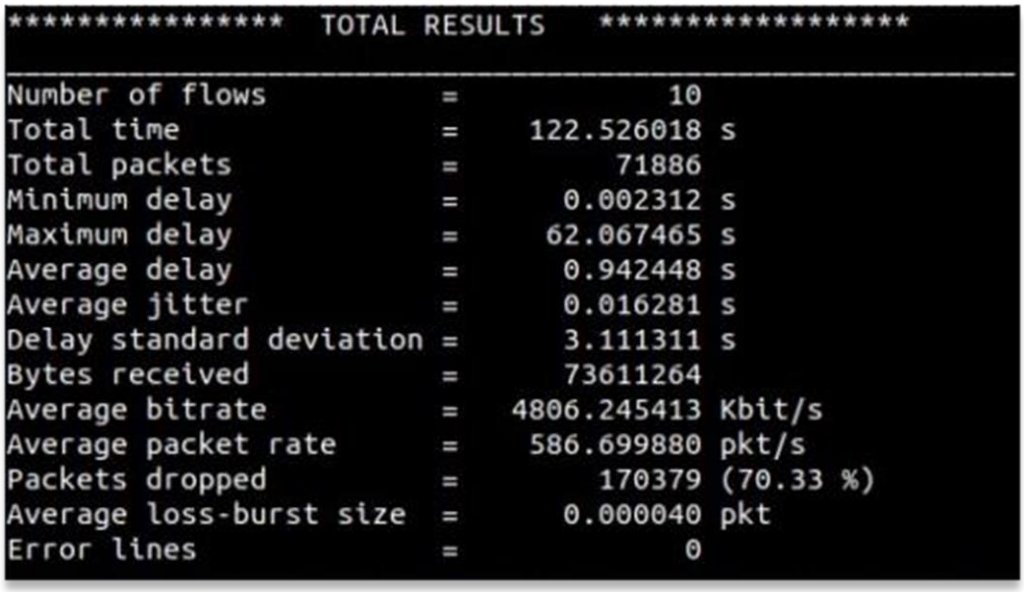


Εικόνα 69: Host h6 D-ITG Log

V. Host H7 Log:

```
**********  TOTAL RESULTS  **********

Number of flows          =         10
Total time               = 122.526018 s
Total packets            =      71886
Minimum delay            =   0.002312 s
Maximum delay            =  62.067465 s
Average delay            =   0.942448 s
Average jitter           =   0.016281 s
Delay standard deviation =   3.111311 s
Bytes received           =   73611264
Average bitrate          = 4806.245413 Kbit/s
Average packet rate      =  586.699880 pkt/s
Packets dropped          =     170379 (70.33 %)
Average loss-burst size  =   0.000040 pkt
Error lines              =          0
```

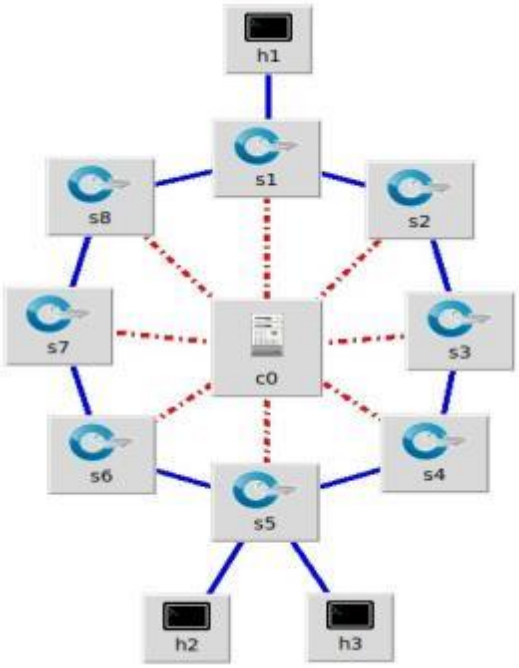


Εικόνα 70: Host h6 D-ITG Log

Μέσα από τα logs του **Σεναρίου ΙΙ** παρατηρήθηκε πως η απόδοση των δυο τοπολογιών ήταν παραπλήσια. Πιο συγκεκριμένα βλέπουμε ίδια ποσοστά χαμένων πακέτων (***Linear* 77.2% / *Tree* 78.66%**) και σχετικά ίδια μέση καθυστέρηση (***Linear* 1.61s / *Tree* 1.17s**). Ωστόσο υπάρχει μια απόκλιση στην απόδοση του host ***h6*** σε σύγκριση με τον host ***h7*** (αναλογία **1:3** Received Bytes). Η σχετικά ίδια απόδοση οφείλεται στο ότι όλη η κίνηση ξεκινάει από έναν host με αποτέλεσμα τον κορεσμό της αρτηρίας όταν εισάγουμε τις παραμέτρους απόδοσης.

3.5.3. Αξιολόγηση Ring Τοπολογίας

Για το πείραμα αυτό δημιουργήθηκε μια τοπολογία δακτυλίου με 8 μεταγωγείς και 3 hosts (1 sender / 2 receiver) όπως φαίνεται στην παρακάτω εικόνα. Οι receivers βρίσκονται και οι δύο στον μεταγωγέα S5 με σκοπό τον έλεγχο της απόδοσης του Multi Path Load Balancer στο σενάριο μειωμένης απόδοσης.



Εικόνα 71: Ring Topology

Αποτελέσματα Σεναρίου I:

Σε γενικές γραμμές η συμπεριφορά του δικτύου στο σενάριο αυτό ήταν παρόμοια με τις προηγούμενες τοπολογίες. Ο έλεγχος των flows μεταξύ των διαδρομών ***S1 – S8 – S7 -S6 -S5*** και ***S1-S2-S3-S4-S5*** έγινε με την χρήση της εντολής `dump-ports` από όπου και προέκυψαν τα:

I. **Switch S7 Dump Ports:**

```
ildi@ildi-VirtualBox:~/mininet/SDN$ sudo ovs-ofctl -O OpenFlow13 dump-ports S7
OFPST_PORT reply (OF1.3) (xid=0x2): 3 ports
  port LOCAL: rx pkts=0, bytes=0, drop=2, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=530.164s
  port  "S7-eth1": rx pkts=106612, bytes=7036276, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=466388, bytes=506012542, drop=0, errs=0, coll=0
           duration=530.201s
  port  "S7-eth2": rx pkts=466387, bytes=506012482, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=106613, bytes=7036336, drop=0, errs=0, coll=0
           duration=530.201s
```

Εικόνα 72: Dump Ports S7

II. **Switch S3 Dump Ports:**

```
ildi@ildi-VirtualBox:~/mininet/SDN$ sudo ovs-ofctl -O OpenFlow13 dump-ports S3
OFPST_PORT reply (OF1.3) (xid=0x2): 3 ports
  port LOCAL: rx pkts=0, bytes=0, drop=2, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=553.653s
  port  "S3-eth1": rx pkts=250240, bytes=268499776, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=158733, bytes=10476476, drop=0, errs=0, coll=0
           duration=553.677s
  port  "S3-eth2": rx pkts=158734, bytes=10476536, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=250240, bytes=268499786, drop=0, errs=0, coll=0
           duration=553.677s
```

Εικόνα 73: Dump Ports S3

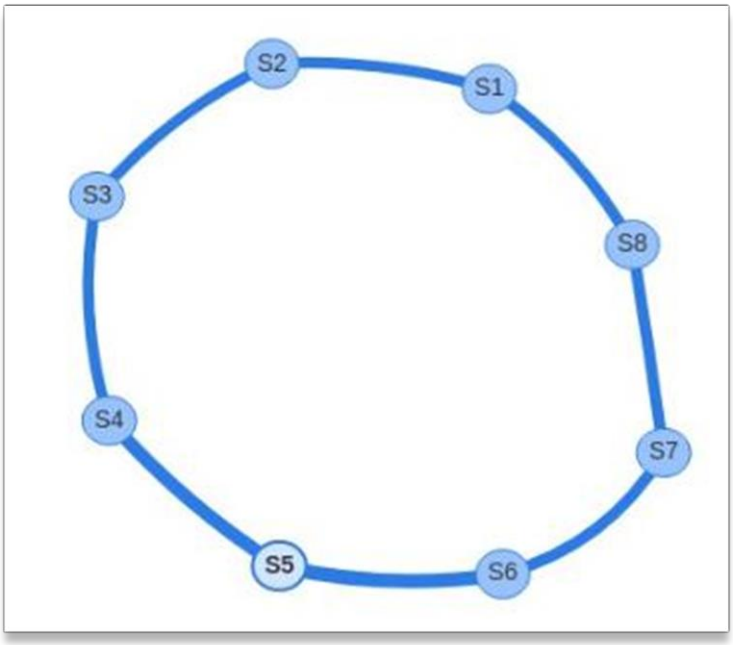
Ο υπολογισμός της κίνησης έγινε με βάση την Egress κίνηση από τους μεταγωγείς που βρίσκονται στο κέντρο της κάθε διαδρομής (***S7 – S3***).

i. **S3-eth2 ←→ S4-eth1** ***tx_pkts:*** 250240
ii. **S7-eth1 ←→ S6-eth2** ***tx_pkts:*** 466388

Η αναλογία **1:2** που προκύπτει είναι μας υποδηλώνει πως το Load Balancer λειτούργησε διαχωρίζοντας τον φόρτο και μη αφήνοντας όλα τα πακέτα να ακολουθήσουν την ίδια διαδρομή.

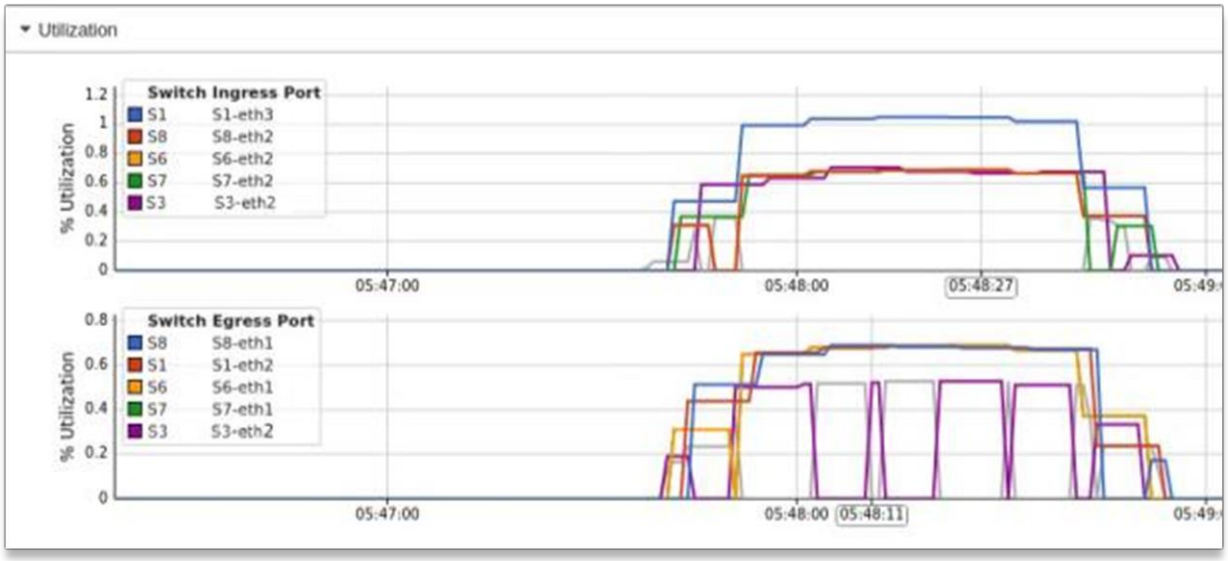
Για να βεβαιώσουμε το παραπάνω συμπέρασμα έγινε έλεγχος των γραφημάτων **I** και **II**. Στο πρώτο μπορούμε να δούμε την κίνηση που κατέγραψε το sFlow προς κάθε κατεύθυνση (μπλε σκούρο χρώμα), ενώ στο δεύτερο, Εικόνα 73, μπορούμε να δούμε την κίνηση που πέρασε από τις πόρτες των ***S3*** και ***S7***.

I. **Topology Flow Visualizer:**



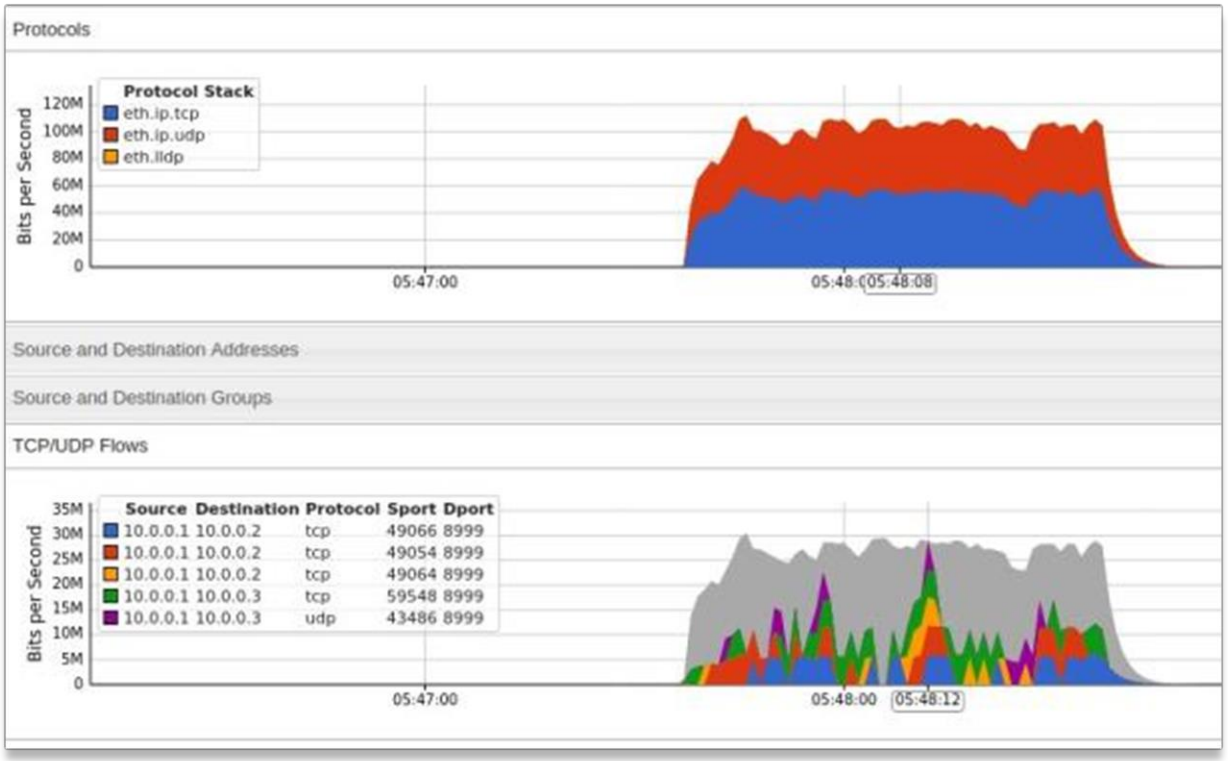
Εικόνα 74: Ring Topology Flow Visualizer

II. Port Utilization:



Εικόνα 75: Ring Port Utilization

III. Protocols:



Εικόνα 76: Ring Protocol Stack

IV. **Host H2 Log:**

```
****************  TOTAL RESULTS   ******************

Number of flows          =            10
Total time               =     60.175325 s
Total packets            =        359280
Minimum delay            =      0.000013 s
Maximum delay            =      0.049658 s
Average delay            =      0.000548 s
Average jitter           =      0.000580 s
Delay standard deviation =      0.001458 s
Bytes received           =     367902720
Average bitrate          =  48910.774641 Kbit/s
Average packet rate      =   5970.553545 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =             0 pkt
Error lines              =             0
```

Εικόνα 77: Host h2 D-ITG Log

V. **Host H3 Log:**

```
****************  TOTAL RESULTS   ******************

Number of flows          =            10
Total time               =     60.180590 s
Total packets            =        359198
Minimum delay            =      0.000013 s
Maximum delay            =      0.050008 s
Average delay            =      0.000558 s
Average jitter           =      0.000579 s
Delay standard deviation =      0.001534 s
Bytes received           =     367818752
Average bitrate          =  48895.333462 Kbit/s
Average packet rate      =   5968.668636 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =             0 pkt
Error lines              =             0
```

Εικόνα 78: Host h3 D-ITG Log

Η απόδοση της τοπολογίας χωρίς παραμέτρους ήταν παρόμοια με τις προηγούμενες δυο όπως προκύπτει από τα logs των **D-ITG Receivers.**

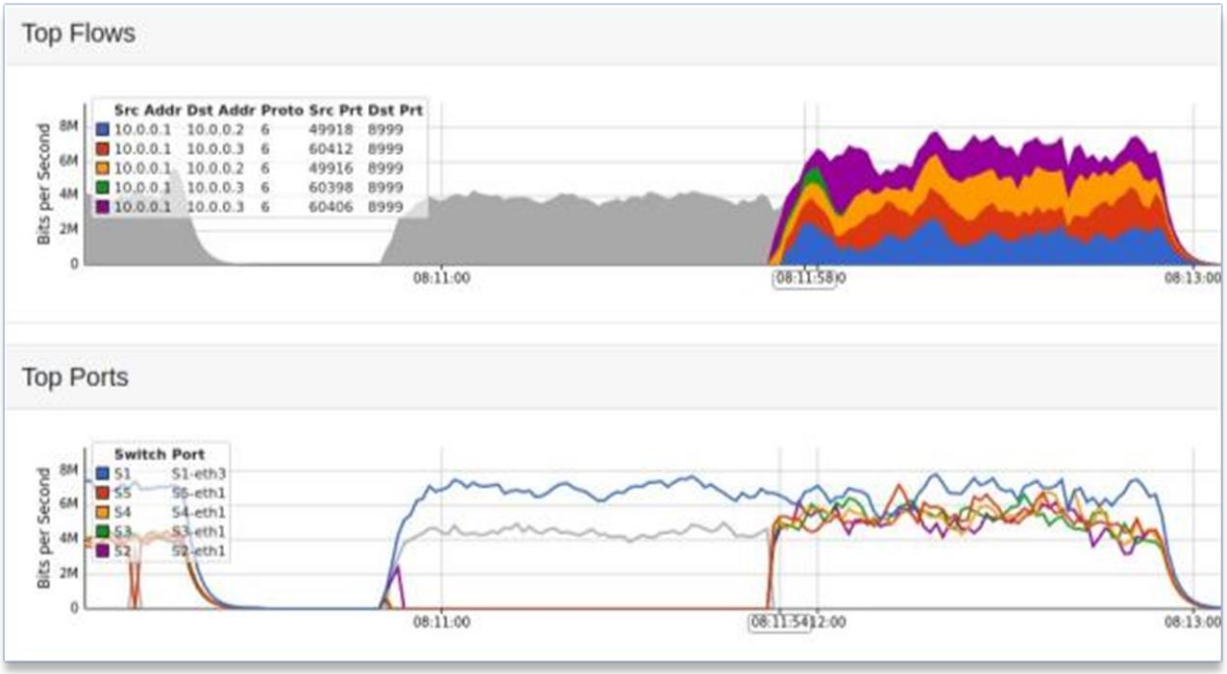
Σενάριο ΙΙ:

Για τον ορισμό των παραμέτρων χρειάστηκε να γίνει κλήση της custom τοπολογίας και εισαγωγή των παραμέτρων με την παρακάτω εντολή:

```
sudo mn –custom top.py,sflow-rt/extras/sflow.py –link
tc,bw=7,delay=10,loss=0.15,max_queue_size=1000 –topo Ring
–controller remote,ip=127.0.0.1:6633
```

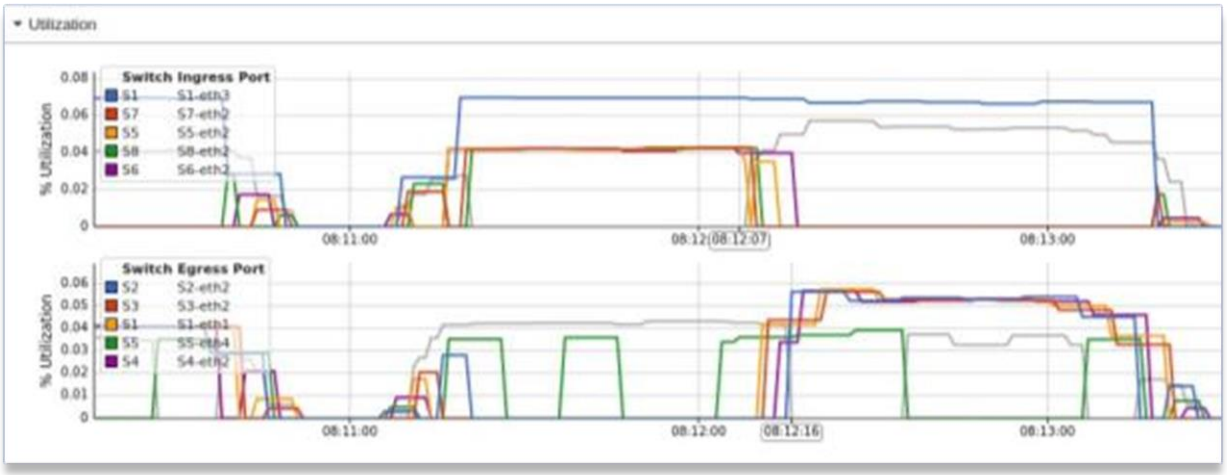
Αποτελέσματα:

I. **Top Flows & Ports:**



Εικόνα 79: Ring Top Flows & Ports

II. **Port Utilization:**



Εικόνα 80: Ring Port Utilization

Αξιοσημείωτη είναι αλλαγή της διαδρομής κατά την αλλαγή των flows από UDP σε TCP στον χρόνο 08:11 όπου βλέπουμε αλλαγή από το path ***S1 – S8 – S7 -S6 -S5*** στο path ***S1-S2-S3-S4-S***.

Τα αποτελέσματα των logs ακολουθούν το ίδιο μοτίβο όπως και στις προηγούμενες δοκιμές.

III. **Host H2 Log:**

```
****************  TOTAL RESULTS   ******************

Number of flows          =           10
Total time               =    123.828427 s
Total packets            =         49117
Minimum delay            =     0.000479 s
Maximum delay            =    66.655461 s
Average delay            =     1.867828 s
Average jitter           =     0.037828 s
Delay standard deviation =     5.547268 s
Bytes received           =     50295808
Average bitrate          =  3249.386863 Kbit/s
Average packet rate      =   396.653670 pkt/s
Packets dropped          =        174735 (78.06 %)
Average loss-burst size  =     0.000041 pkt
Error lines              =            0
```

Εικόνα 81: Host h2 D-ITG Log

IV. **Host H3 Log:**

```
****************  TOTAL RESULTS   ******************

Number of flows          =           10
Total time               =    123.660448 s
Total packets            =         50463
Minimum delay            =     0.026149 s
Maximum delay            =    61.666742 s
Average delay            =     1.906604 s
Average jitter           =     0.029971 s
Delay standard deviation =     3.545037 s
Bytes received           =     51674112
Average bitrate          =  3342.967802 Kbit/s
Average packet rate      =   408.077124 pkt/s
Packets dropped          =        174348 (77.55 %)
Average loss-burst size  =     0.000041 pkt
Error lines              =            0
```

Εικόνα 82: Host h3 D-ITG Log

Συμπεράσματα Δοκιμών:

Με εξαίρεση κάποιες αποκλίσεις στο **Σενάριο ΙΙ** του Tree Topology, η συμπεριφορά και η απόδοση των τοπολογιών στα δυο διαφορετικά σενάρια τείνει να είναι παρόμοια. Τα αποτελέσματα αυτά μπορούμε να τα αποδώσουμε στην φύση των δοκιμών, όπου είχαμε ένα συγκεκριμένο εξερχόμενο φόρτο δικτύου από μόνο έναν transmitter.

Για έναν πιο εκτενή έλεγχο θα χρειαζόταν να γίνει flood του δικτύου με κίνηση προερχόμενη από διαφορετικούς transmitters προς παραπάνω receivers. Το **D-ITG** μας παρέχει κάποια απλά εργαλεία για την δημιουργία κίνησης και την ανάλυση της, ωστόσο ένας ενσωματωμένος μηχανισμός ανάλυσης (π.χ. OpenDaylight Analytics) θα ήταν η βέλτιστη λύση. Έτσι θα ήταν εφικτό να δώσουμε έμφαση στις διαφορές των τοπολογιών.

Παρόλα αυτά, με τις παραπάνω δοκιμές καταφέραμε να αναδείξουμε το Mininet ως ένα ικανό περιβάλλον προσομοίωσης διαφορετικών καταστάσεων δικτύου και το sFlow ως ένα λειτουργικό εργαλείο παρακολούθησης.

Συμπεράσματα

Το Software Defined Networking έχει τραβήξει τα φώτα της δημοσιότητας πάνω του δημιουργώντας μεγάλες προσδοκίες, έναν ενθουσιασμό που έχει οδηγήσει ακόμα και σε χρήση χαρακτηρισμών όπως: η επανάσταση στο χώρο των δικτύων. Η πραγματικότητα είναι πως δεν πρόκειται για κάποια ριζική καινοτομία που θα αλλάξει τα πάντα στον χώρο, αλλά μια συλλογή παλιών και νέων εργαλείων μέσα σε ένα καινούργιο πλαίσιο/αρχιτεκτονική που είναι φορέας μιας νέας αντίληψης ως προς την λειτουργία των δικτύων.

Το πλαίσιο αυτό θα εμπλουτίσει τον τρόπο που οι τεχνικοί σχεδιάζουν, υλοποιούν και διαχειρίζονται τα δίκτυα. Προστίθενται δυνατότητες που δεν υπήρχαν ως τώρα όμως αυτό γίνεται με έναν μη επιθετικό τρόπο καθώς το SDN δεν επιδιώκει την εκδίωξη του "παλιού" και την επικράτηση της δικής του αρχιτεκτονικής, ενώ σε σύγκριση με τα παραδοσιακά δίκτυα, το SDN παρουσιάζει σημαντικά πλεονεκτήματα. Η δυνατότητα αυτοματοποίησης των υπηρεσιών και των λειτουργιών διαχείρισης δικτύου μέσω προγραμματισμού οδηγεί σε μειωμένη πολυπλοκότητα των δικτύων. Διασφαλίζεται η συνεπής εφαρμογή πολιτικών και διευκολύνεται η ανάπτυξη αποτελεσματικότερων μηχανισμών ασφάλειας, λόγω της καθολικής οπτικής του δικτύου και της μεγάλης συλλογής στατιστικών. Ακόμα, οι συχνές ανανεώσεις που πραγματοποιούνται στα δίκτυα SDN περιορίζουν τον κίνδυνο επιθέσεων, όπως της πλαστογράφησης.

Ωστόσο, η διασφάλιση του πεδίου ελέγχου εξακολουθεί να αποτελεί σημείο το οποίο χρήζει ιδιαίτερης προσοχής. Η ύπαρξη ενός κεντρικού ελεγκτή ενέχει μεγάλους κινδύνους ασφάλειας, όπως επιθέσεων DDos και MiM, ενώ η εμφάνιση τέτοιου είδους επιθέσεων στα παραδοσιακά δίκτυα είναι σπανιότερη. Σημαντική είναι και η διασφάλιση της διεπαφής southbound, καθώς μέσω αυτής μεταφέρεται ολόκληρος ο έλεγχος στο επίπεδο δεδομένων.

Συμπεράσματα από την υλοποίηση

Η υλοποίηση μιας προσομοίωσης Software Defined Networking παρουσιάζει μια σειρά μέτριων δυσκολιών, ενώ απαιτεί κάποιες γνώσεις σε Linux Command Line και Python. Συνολικά ήταν μια άκρως διδακτική εμπειρία που μέσα από την υπερπήδηση των δυσκολιών προσφέρει ένα εύρος γνώσεων όσον αφορά την κατανόηση του τρόπου λειτουργίας του Software Defined Networking, των συστημάτων Linux, του Virtualization και της γλώσσας Python.

Το εργαλείο εξομοίωσης Mininet αποτελεί έναν αποδοτικό τρόπο για τη δημιουργία SDN δικτύων, με την ύπαρξη ενός απλού γραφικού περιβάλλοντος και τις επιπλέον δυνατότητες που παρέχονται, μέσω της γραμμής εντολών. Η τοπολογία, που παρουσιάστηκε, δοκιμάστηκε, επιτυχώς, αναφορικά με τη συνδεσιμότητα των κόμβων και την επικοινωνία μεταξύ του ελεγκτή και των μεταγωγέων, όπως αυτή καταγράφεται μέσω των δοκιμών.

Ο ελεγκτής Ryu αποδεικνύεται απαιτητικός σε γνώσεις προγραμματισμού, μιας και απευθύνεται κυρίως σε προγραμματιστές. Παρέχοντας όμως αρκετές δυνατότητες παραμετροποίησης, αποτελεί μία βάση πάνω στην οποία ο εκάστοτε οργανισμός/επιχείρηση θα μπορεί να εξελίξει τις λειτουργίες διαχείρισης δικτύου καθώς και τις προσφερόμενες υπηρεσίες δικτύου.

Το παρόν και το μέλλον του SDN

Στον παρόντα χρόνο το Software Defined Networking ανασυντάσσεται, ωριμάζει, μαθαίνει να τρέχει αντί να περπατά, βρίσκει τα όρια του και τον τρόπο με τον οποίον θα ενταχθεί στην καθημερινή πραγματικότητα των δικτύων παγκοσμίως. Τόσο η βιομηχανία όσο και η επιστημονική και ερευνητική κοινότητα συνεχίζουν να εξελίσσουν, να μετασχηματίζουν και να εμπλουτίζουν το SDN, όμως όχι κάτω από το φως των προβολέων της δημοσιότητας. Αυτό ερμηνεύεται από κάποιους ειδικούς ως σημάδι σοβαρής προετοιμασίας για τη διάθεση ολοκληρωμένων SDN υλοποιήσεων και της συνειδητοποίησης πως, αυτό πρέπει να γίνει με έναν τρόπο που δεν θα δημιουργήσει μεγάλες προσδοκίες οι οποίες στην πορεία θα διαψευστούν, ούτε θα βασιστούν σε μια αρχιτεκτονική που δεν έχει αναπτυχθεί πλήρως.

Παράλληλα αναμένεται να συνεχιστεί η δράση των κοινοτήτων λογισμικού και της ύπαρξης του ανοιχτού κώδικα στο Software Defined Networking με την ανάπτυξη και διεύρυνση των ήδη υπαρχόντων λύσεων, ως παράλληλες στις εταιρικές υλοποιήσεις. Η διαχρονική ανάμειξη μη κερδοσκοπικών δομών στο SDN θα εγγυηθεί την ύπαρξη των ανοιχτών προτύπων καθώς και διάθεση δωρεάν εργαλείων που θα επιτρέπουν σε μικρές εταιρίες αλλά και μεμονωμένα άτομα να έχουν πρόσβαση στην αρχιτεκτονική ώστε να πειραματιστούν και να κάνουν υλοποιήσεις μικρής κλίμακας. Ο κόσμος της πληροφορικής σίγουρα κινείται προς ένα μέλλον καθοριζόμενο από λογισμικό όπου το SDN, με τα πολλά πλεονεκτήματα του και την εκπληκτικά πρόθυμη βιομηχανία, βρίσκεται στο μονοπάτι για να γίνει το νέο πρότυπο για τα δίκτυα.

Μελλοντική Έρευνα

Πιστεύουμε πως υπάρχουν ακόμα αρκετά πράγματα που μπορούν να γίνουν για να θεωρηθεί ολοκληρωμένη η παρούσα εργασία. Πιο συγκεκριμένα:

I. Ενσωμάτωση των εφαρμογών *Flow Timeout, Εφαρμογή Συλλογής Στατιστικών και Sniffer* στην εφαρμογή *Multipath Load Balancer*.
   a. Περαιτέρω ανάπτυξη της εφαρμογής *Multipath Load Balancer* με την ενσωμάτωση μηχανισμών: **Quality of Service, Firewall, Denial of Service Protection, Link Failure Fast Failover & Switchover, Service Function Chaining**
II. Ανάλυση απόδοσης μεγαλύτερων τοπολογιών με παραπάνω Transmitters & Receivers
III. Έλεγχος συμπεριφοράς δικτύου υπό διαφορετικούς τύπους φορτίου (Video Streaming, VOIP)

Βιβλιογραφία

[1] D. Qiang και M. Toy, Virtualized Software-Defined Networks and Services, 2016.

[2] S. Shenker, T. Anderson, L. Peterson, S. Shenker και J. Turner, Overcoming the Internet Impassse, 2005.

[3] J. Turner και D. E. Taylor, «Diversifying the Internet,» σε *Proceedings of the 2015 IEEE*, 2005.

[4] A. Chowdhary, D. Huang και S. Pisharody, Software-Defined Networking and Security: From Theory to Practice, 2018.

[5] ETSI, «Network Functions Virtualisation; an introduction, benefits, enablers, challenges & call for action,» σε *SDN and OpenFlow World Congress*, Darmstadt, Germany, 2012.

[6] S. Asadollahi και B. Goswami, «Revolution in Existing Network under the Influence of SDN,» σε *INDIACom-2017*, New Delhi, 2017.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker και J. Turner, «OpenFlow: Enabling Innovation in Campus Networks,» σε *ACM SIGCOMM*, 2008.

[8] Open Network Foundation, «ONF TR-521: SDN Architecture , Issue 1.1,» 2016.

[9] P. Goransson, C. Black και T. Culver, Software Defined Networks A Comprehensive Approach, 2017.

[10] Open Network Fundation, «ONF SDN Evolution,» 2016.

[11] Open Networking Foundation, «OpenFlow Switch Specification,» 2015.

[12] NOX, «NOX,» [Ηλεκτρονικό]. Available: https://github.com/noxrepo/.

[13] The Linux Foundation, «OpenDaylight,» [Ηλεκτρονικό]. Available: https://www.opendaylight.org/about/platform-overview.

[14] The Linux Foundation, «Tungsten Fabric,» [Ηλεκτρονικό]. Available: https://tungsten.io/.

[15] D. Erickson, «OpenFlow Stanford Edu,» [Ηλεκτρονικό]. Available: https://openflow.stanford.edu/display/Beacon/Home.html.

[16] Project Floodlight , «Project Floodlight,» 13 5 2018. [Ηλεκτρονικό]. Available: https://floodlight.atlassian.net/wiki/spaces/HOME/overview?mode=global. [Πρόσβαση 21 8 21].

[17] R. P. Team, RYU SDN Framework.

[18] O. N. Lab, «Github,» [Ηλεκτρονικό]. Available: https://github.com/OPENNETWORKINGLAB/flowvisor/wiki.

[19] L. Mamushiane, A. Lysko και S. Dlamini, «A Comparative Evaluation of the Performance of Popular SDN Controllers,» σε *10th Wireless Days Conference (WD*, Pretoria, South Africa, 2018.

[20] S. Scott-Hayward, G. O'Callaghan και S. Sezer, «SDN Security: A Survey,» *IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013.

[21] M. Garcia, A. Bessani, I. Gashi, N. Neves και R. Obelheiro, «OS Diversity for Intrusion Tolerance: Myth or Reality?,» σε *IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN), 2011*, 2011.

[22] Z. Yan και C. Prehofer, «Autonomic Trust Management for a Component-Based Software System,» *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING*, τόμ. 8, 2011.

[23] T. Islam, N. Islam και A. Refat, «Node to Node Performance Evaluation through RYU SDN Controller,» *Wireless Personal Communications (2020) ,* 2020.

[24] N. T. a. T. Corporation, «Ryu Read The Docs,» 2021. [Ηλεκτρονικό]. Available: https://ryu.readthedocs.io/en/latest/writing_ryu_app.html. [Πρόσβαση 8 2021].

[25] GeeksforGeeks, «geeksforgeeks.org,» [Ηλεκτρονικό]. Available: https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/. [Πρόσβαση 15 9 2021].

[26] Cisco, «Cisco.com,» Cisco, [Ηλεκτρονικό]. Available: https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/7039-1.html#t6. [Πρόσβαση 15 9 2021].

[27] Linux Foundation, «OpenvSwitch,» [Ηλεκτρονικό]. Available: http://www.openvswitch.org//support/dist-docs/ovs-ofctl.8.txt. [Πρόσβαση 17 9 2021].

[28] Inmon Corp., «inmon.com,» Inmon Corporation, [Ηλεκτρονικό]. Available: https://inmon.com/products/sFlow-RT.php. [Πρόσβαση 10 10 2021].

[29] Universita' degli Studi di Napoli "Federico II, «traffic.comics.unina.it,» [Ηλεκτρονικό]. Available: http://traffic.comics.unina.it/software/ITG/index.php.

[30] W. Xia, W. Yonggang, H. F. Chuan, N. Dusit και X. Haiyong, «A Survey on Software-Defined Networking,» *IEEE COMMUNICATION SURVEYS & TUTORIALS, VOL. 17, NO. 1, FIRST QUARTER 2015*, 2015.

[31] Q. Wang, B. Geddings και I. Ryan, «Project Floodlight,» 13 5 2018. [Ηλεκτρονικό]. Available: https://floodlight.atlassian.net/. [Πρόσβαση 21 8 2021].