# Ανοικτό Πανεπιστήμιο Κύπρου

## Σχολή Θετικών και Εφαρμοσμένων Επιστημών


## Μεταπτυχιακή Διατριβή
## Στην Ασφάλεια Υπολογιστών και Δικτύων

# Εντοπισμός και Αντιμετώπιση DoS/DDoS Επιθέσεων σε SDN


**Γεώργιος Αντωνιάδης**

**Επιβλέπων Καθηγήτρια**
**Αδαμαντίνη Περατικού**



**Μάιος 2022**

# Ανοικτό Πανεπιστήμιο Κύπρου

## Σχολή Θετικών και Εφαρμοσμένων Επιστημών


# Εντοπισμός και Αντιμετώπιση DoS/DDoS Επιθέσεων σε SDN


**Γεώργιος Αντωνιάδης**

**Επιβλέπων Καθηγήτρια**
**Αδαμαντίνη Περατικού**



Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε
προς μερική εκπλήρωση των απαιτήσεων για απόκτηση

μεταπτυχιακού τίτλου σπουδών
στην Ασφάλεια Υπολογιστών και Δικτύων

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

**Μάιος 2022**

# Περίληψη

Η αρχιτεκτονική του SDN κεντρίζει πλέον το ενδιαφέρον αρκετών επιχειρήσεων για την ένταξη του στην υποδομή τους. Οργανισμοί οι οποίοι προμηθεύουν σε επιχειρήσεις τεχνολογικές λύσεις, αναπτύσσουν προϊόντα βασιζόμενα στο SDN καθώς τα χαρακτηριστικά που διαθέτει διευκολύνουν την διαχείριση του δικτύου. Η ευελιξία που παρέχει το SDN να χρησιμοποιείται τόσο σε cloud όσο και σε on-premises ή υβριδικό περιβάλλον, το καθιστά ως επιλογή για κάθε είδος υποδομής ενός οργανισμού. Η κεντρικοποιημένη διαχείριση του δικτύου που προσφέρεται από το SDN, διαχωρίζοντας το control plane από το data plane, δημιουργεί προοπτικές ανάπτυξης επιπλέον τεχνολογικών λύσεων με στόχο την απλοποίηση, την ασφάλεια και το κόστος της υποδομής.

Στην παρούσα εργασία αναλύονται τα κυριότερα χαρακτηριστικά του SDN επισημαίνοντας ταυτόχρονα και τα τρωτά του σημεία τα οποία χρήζουν περισσότερης προσοχής ως προς την ασφάλεια του. Ειδικότερα, πραγματοποιείται η υλοποίηση ενός SDN έχοντας ως στόχο την ανίχνευση και αντιμετώπιση DoS/DDos επιθέσεων μέσα στο δίκτυο. Γίνεται μια αναλυτική καταγραφή της υλοποίησης και του σκεπτικού αντιμετώπισης των επιθέσεων ενώ στο τέλος παρουσιάζεται ο αυτοματισμός της διαδικασίας με την δημιουργία ενός script σε python.

# Summary

SDN's architecture is now attracting the interest of several companies for its integration into their infrastructure. Vendors that provide technology solutions to companies, develop SDN-based products as its features facilitate network management. The flexibility provided by SDN to be used in both cloud and on-premises or hybrid environments, makes it an option for any type of infrastructure of an organization. The centralized management of the network offered by SDN, separating the control plane from the data plane, creates prospects for the development of additional technological solutions aimed at simplification, security and cost of infrastructure.

In the present dissertation, the main features of SDN are analyzed, at the same time pointing out its vulnerabilities which need more attention in terms of its security. In particular, an SDN is implemented with the aim of detecting and mitigating DoS/DDos attacks within the network. A detailed record of the implementation and the rationale for dealing with the attacks is made, while at the end, the automation of the process is presented by creating a script in python.

# Ευχαριστίες

Θα ήθελα να ευχαριστήσω την οικογένεια μου καθώς και του εργοδότες μου για την στήριξη τους σε αυτή μου την προσπάθεια για διεκπεραίωση των σπουδών μου. Ευχαριστώ επίσης την Δρ. Περατικού για την στήριξη και καθοδήγηση της για την διεκπεραίωση της παρούσας διπλωματικής διατριβής.

# Περιεχόμενα

# Κεφάλαιο 1
# Εισαγωγή

Οι ολοένα αυξανόμενες ανάγκες των επιχειρήσεων σε θέματα τεχνολογίας, οδηγεί συνεχώς στην βελτίωση αλλά και την ανάπτυξη νέων υπηρεσιών όπως και λύσεων οι οποίες ικανοποιούν τις ανάγκες τους έτσι ώστε να παραμένουν ανταγωνιστικοί μέσα σε ένα απαιτητικό περιβάλλον. Πρώιμες τεχνολογίες όπως το virtualization αλλά και το cloud είναι γνωστές εδώ και τουλάχιστον τέσσερεις δεκαετίες ωστόσο στα μέσα του 2000 με την εμφάνιση των Google Cloud Platform, Microsoft Azure και Amazon Web Services, άρχισε να γίνεται η καθιέρωση του cloud computing καθώς επίσης και του μοντέλου "as-a-service" ξεκινώντας βασικά από το "server-as-a-service" [1]. Παράλληλα αναπτύχθηκαν τα smartphones, επεκτάθηκε και βελτιώθηκε το διαδίκτυο κάτι το οποίο έδωσε την δυνατότητα στις επιχειρήσεις να έχουν ουσιαστικά τα δεδομένα τους στην τσέπη τους ανά πάσα στιγμή.

Από τότε μέχρι σήμερα έγιναν μεγάλες αλλαγές και βελτιώσεις ενώ πλέον έχει επικρατήσει ο όρος "Anything-as-Service" (XaaS) καθώς όλο και περισσότεροι vendors παρέχουν τις υπηρεσίες τους σε περιβάλλον cloud κάτι το οποίο έχει πολλαπλά οφέλη για τις εταιρίες.

Ασφαλώς, αυτή μετάβαση θα ήταν αδύνατο να πραγματοποιηθεί με το παραδοσιακό μοντέλο δικτύου. Παράλληλα, έπρεπε να αντικατασταθούν οι φυσικές συσκευές δικτύου (όπως για

παράδειγμα switches, routers, firewalls) με λογισμικά, πλήρως προγραμματιζόμενά, κάτι που τελικά μας οδήγησε στην αρχιτεκτονική Network Functions Virtualization (NFV) το οποίο αποτελείται από εικονικές συσκευές δικτύου – Virtual Network Functions (VNFs) [2].

Νωρίτερα άρχισε να αναπτύσσεται και το Software Defined Networking (SDN) όπου στόχος ήταν να γίνει ο διαχωρισμός του ποιος αποφασίζει για την δρομολόγηση των πακέτων (control plane) στο δίκτυο και ποιος θα στέλνει τα πακέτα στον προορισμό τους (data plane) καθιστώντας έτσι την διαχείριση του δικτύου προγραμματιζόμενη. Αυτός ο διαχωρισμός που γίνεται, αποτελεί και την κύρια διαφορά μεταξύ NFV όπου η λειτουργία του δικτύου σε μια φυσική συσκευή μιμείται από έναν server ενώ στο SDN διαχωρίζεται το control από το data plane καθιστώντας και τα δύο προγραμματιζόμενα.

Και στις δύο περιπτώσεις, η διαχείριση του δικτύου καθίσταται πολύ ευκολότερη καθώς γίνεται από κεντρικά σημεία του δικτύου κάτι ωστόσο που δημιουργεί και τον κίνδυνο απώλειας όλων των υπηρεσιών σε περίπτωση που για οποιοδήποτε λόγο ο κεντρικός server (controller στην περίπτωση του SDN) τεθεί εκτός λειτουργίας. Γίνεται έτσι αντιληπτό, ότι όπως ακριβώς συμβαίνει στο παραδοσιακό δίκτυο με τις redundant συσκευές, τα load balancers κτλ. θα πρέπει να γίνει και στην αρχιτεκτονική του SDN. Ασφαλώς το κόστος του redundancy σε ένα SDN μειώνεται δραματικά σε σχέση με το παραδοσιακό φυσικό δίκτυο.

## 1.1 Σκοπός της Έρευνας

Αντικείμενο της έρευνας αποτελεί η αντιμετώπιση DoS/DDoS επιθέσεων στην αρχιτεκτονική SDN τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο. Ποιο συγκεκριμένα, επιχειρείται η ανίχνευση και αντιμετώπιση των επιθέσεων εστιάζοντας στην χρησιμοποίηση εργαλείων ανοικτού κώδικα τα οποία θα βοηθήσουν στην αρχικά στην υλοποίηση ενός SDN, στην πραγματοποίηση επιθέσεων και στην ανίχνευση τους έχοντας ως στόχο την αυτοματοποίηση της αντιμετώπισης τους χρησιμοποιώντας ιδιότητες που προσφέρει η τεχνολογία του SDN.

## 1.2 Βασικά Ερευνητικά Ερωτήματα

Για να εξυπηρετηθεί ο σκοπός της έρευνας, θα πρέπει να απαντηθούν τα πιο κάτω ερωτήματα:

i. Ποιες δυνατότητες, τις οποίες δεν παρέχει ένα παραδοσιακό δίκτυο, μας παρέχει το SDN για ανίχνευση και αντιμετώπιση επιθέσεων.

ii. Πόσο πιο εύκολη ή δύσκολη είναι η αντιμετώπιση των επιθέσεων στο SDN το οποίο παρέχει μια κεντρική μονάδα διαχείρισης των συσκευών δικτύου σε σχέση με το παραδοσιακό δίκτυο.

iii. Ποια εργαλεία, ανοικτού κώδικα, μπορούν να χρησιμοποιηθούν στην αρχιτεκτονική του SDN για ανίχνευση και αντιμετώπιση επιθέσεων.

iv. Ποιες ενέργειες μπορούν να γίνουν μετά από τον εντοπισμό και κατά την αντιμετώπιση των επιθέσεων.

## 1.3 Μεθοδολογία

Η έρευνα αρχικά πρέπει να μελετήσει την βιβλιογραφία για τις διάφορες υπάρχουσες ή προτεινόμενες λύσεις που απαντούν στα πιο πάνω ερωτήματα. Ακολούθως, θα πρέπει να γίνει ανάλυση της αρχιτεκτονικής του SDN παραθέτοντας και εξηγώντας τα κυριότερα συστατικά του, έχοντας σαν στόχο να κατανοηθεί ο τρόπος λειτουργίας του δίνοντας μας έτσι την δυνατότητα να προτείνουμε και να υλοποιήσουμε λύσεις αξιοποιώντας αυτά τα συστατικά του. Το επόμενο βήμα της έρευνας είναι να δημιουργηθεί το περιβάλλον SDN χρησιμοποιώντας τα πιο διαδεδομένα εργαλεία ανοικτού κώδικα όπως για παράδειγμα το OpenStack, το Onos και το Mininet. Ακολούθως, θα πρέπει να πραγματοποιηθούν οι επιλεγμένες επιθέσεις DoS/DDos για να εντοπιστεί και να καταγραφεί ο αντίκτυπος που μπορεί να έχουν στο δίκτυο δίνοντας μας ταυτόχρονα και την ευκαιρία για καλύτερη κατανόηση των επιθέσεων σε περιβάλλον SDN. Κατά την πραγματοποίηση των επιθέσεων, γίνεται ανάλυση και καταγραφή των πληροφοριών που παίρνουμε από τον controller όπως για παράδειγμα τα διάφορα πεδία που χρησιμοποιούνται στην επικεφαλίδα των πακέτων έτσι ώστε να γίνει εφικτή η οδηγία προς τον controller για την απόρριψη τους. Στη συνέχεια, θα μελετηθούν οι δυνατότητες αυτοματισμού που παρέχει ο controller που θα χρησιμοποιηθεί και ειδικότερα οι δυνατότητες του Application Programming Interface (API) ως προς την δημιουργία κανόνων διαχείρισης μιας ροής πακέτων. Αφού εξεταστεί η δομή και τα απαιτούμενα πεδία που χρειάζονται για την καταχώρηση των νέων κανόνων στον controller, επόμενο βήμα είναι ο έγκαιρος εντοπισμός των επιθέσεων έτσι ώστε να καταχωρούνται οι κανόνες. Για αυτό τον σκοπό επιλέχθηκε ένα IDS/IPS ανοιχτού κώδικα, το

οποίο θα λειτουργεί απλά σαν IDS και θα παράγει ειδοποιήσεις ανάλογα με τους κανόνες που θα θέσουμε. Σημαντικό ρόλο για την επιλογή του IDS, θα διαδραματίσει και η ικανότητα του να παράγει ειδοποιήσεις που θα έχουν όσο το δυνατό περισσότερη πληροφορία την οποία χρειάζεται ο controller καθώς επίσης και να είναι σε μορφή την οποία εύκολα μπορεί να την επεξεργαστεί. Το τελικό στάδιο της έρευνας είναι να συνδυαστούν οι πιο πάνω ενέργειες έτσι ώστε να γίνει ο αυτοματισμός ξεκινώντας από τον εντοπισμό μιας επίθεσης, την δημιουργία της κατάλληλης οδηγίας και καταχώρηση της στον controller καθώς επίσης και την παρακολούθηση της εξέλιξης της επίθεσης με σκοπό να επανέλθει η επικοινωνία με τον τερματισμό της επίθεσης.

## 1.4 Δομή της Μεταπτυχιακής Διατριβής

Η παρούσα μεταπτυχιακή διατριβή έχει την ακόλουθη δομή:

i. Το Κεφάλαιο 2 περιλαμβάνει μια σύντομη επισκόπηση της βιβλιογραφίας σχετικά με διάφορες προτεινόμενες λύσεις για τον μετριασμό επιθέσεων σε SDN.

ii. Το Κεφάλαιο 3 περιλαμβάνει πληροφορίες ως προς την αρχιτεκτονική του SDN και τα σημαντικότερα συστατικά που το αποτελούν δίνοντας παράλληλα πληροφορίες ως προς τον τρόπο λειτουργίας του.

iii. Το Κεφάλαιο 4 περιλαμβάνει την υλοποίηση του συστήματος και των επιθέσεων καθώς επίσης και τα βήματα που ακολουθήθηκαν για τον αυτοματισμό αντιμετώπισης συγκεκριμένων επιθέσεων.

iv. Το Κεφάλαιο 5 περιλαμβάνει τα Συμπεράσματα καθώς και σκέψεις για μελλοντική ανάπτυξη και βελτίωση του προτεινόμενου τρόπου εντοπισμού κυρίως αλλά και αντιμετώπισης διαφόρων επιθέσεων σε SDN.

# Κεφάλαιο 2
# Βιβλιογραφική Ανασκόπηση

Στη βιβλιογραφία εκπονήθηκαν αρκετές μελέτες οι οποίες εστιάζουν στην ασφάλεια των SDN κάτι που αποδεικνύει ότι πλέον οι υπό ανάπτυξη τεχνολογίες εξελίσσονται με βάση την ασφάλεια. Για να κατανοηθούν οι πτυχές της ασφάλειας του SDN, χρειάζεται πρώτα να κατανοηθεί η αρχιτεκτονική του και τα επιμέρους στοιχεία που το αποτελούν.

Οι Blial et al. [3] κάνουν μια επισκόπηση της αρχιτεκτονικής του SDN και το διαχωρίζουν σε έξι διαφορετικά στοιχεία:

i. Το management plane το οποίο είναι ένα σύνολο εφαρμογών δικτύου που διαχειρίζονται την λογική του δικτύου,

ii. Το control plane το οποίο αποτελεί τον εγκέφαλο και το κατ' επέκταση το πιο σημαντικό στοιχείο του SDN,

iii. Το data plane γνωστό και ως infrastructure layer το οποίο αποτελείται από συσκευές δικτύου όπως routers, switches και load balancers. Επικοινωνεί με τον controller χρησιμοποιώντας τα Southbound interfaces (SBI),

iv. Τα Northbound interfaces (NBI) τα οποία είναι ένα σύνολο από APIs, συνήθως ανοιχτού κώδικα, τα οποία επιτρέπουν την επικοινωνία του control plane με το management plane,

v. Τα East-West interfaces (EWI) τα οποία επιτρέπουν την επικοινωνία μεταξύ των controllers χρησιμοποιώντας ένα σύστημα ειδοποιήσεων και μηνυμάτων ή ακόμη και παραδοσιακά πρωτόκολλά δρομολόγησης όπως το BGP και το OSPF και

vi. Τα SBI τα οποία επιτρέπουν την επικοινωνία μεταξύ control plane και data plane. Το πιο διαδεδομένο southbound API που χρησιμοποιείται είναι το πρωτόκολλο OpenFlow.

Οι συγγραφείς εστιάζουν στην αναγκαιότητα της δημιουργίας multicontroller αρχιτεκτονικών SDN για τρείς, κυρίως, λόγους: Την αποτελεσματικότητα (efficiency), την επεκτασιμότητα (scalability) και την διαθεσιμότητα (availability) του δικτύου η οποία εμπεριέχει το redundancy και το security. Σε περίπτωση αποτυχίας του controller σημαίνει και αποτυχία εξυπηρέτησης ολόκληρου του δικτύου έτσι η ύπαρξη περισσότερων controllers αποτελεί μονόδρομο στην αρχιτεκτονική του SDN. Επίσης, στην έρευνα τους γίνεται διαχωρισμός των διάφορων αρχιτεκτονικών του δικτύου αποσαφηνίζοντας έννοιες όπως: Φυσικά κεντροποιημένο έναντι φυσικά κατανεμημένου, λογικά κατανεμημένου έναντι λογικά κεντρικοποιημένου, επίπεδη αρχιτεκτονική έναντι ιεραρχικής αρχιτεκτονικής και δυναμικής αρχιτεκτονικής έναντι στατικής αρχιτεκτονικής. Στο τέλος παρουσιάζουν διάφορα παραδείγματα (Onix, HyperFlow, Onos, Disco, Elasticon) λογικά κεντρικοποιημένης αρχιτεκτονικής πολλαπλών controllers (logically centralized multi-controller architectures) καθώς και παραδείγματα (Kandoo και Orion) λογικά κατανεμημένης αρχιτεκτονικής (Logically distributed architectures).

Οι Liu et al. [4] στο άρθρο τους αναλύουν την αρχιτεκτονική και την λειτουργία του SDN και στη συνέχεια συνοψίζουν τα τυπικά ζητήματα ασφαλείας που αφορούν στα διαφορετικά στοιχεία που αποτελούν το SDN (application layer, northbound interface, control layer, southbound interface και data layer). Επίσης κάνουν μια επισκόπηση διαφόρων λύσεων που προτάθηκαν έτσι ώστε να βελτιώσουν τα επίπεδα ασφάλειας κάθε επιπέδου και περιλαμβάνει την εξουσιοδοτημένη μονάδα ελέγχου ταυτότητας, απομόνωσης εφαρμογών, άμυνας από επιθέσεις DoS/DDoS, ανάπτυξης πολλαπλών ελεγκτών και ανίχνευση flow rule consistency.

Οι Melkov και Paulikas [5] αναφέρονται στην προοπτική χρήσης νοημοσύνης (Intelligence) στα δίκτυα και στην προοπτική που παρέχει στις συσκευές ο διαχωρισμός του data plane να λειτουργούν σε headless mode κατά τη διάρκεια κρίσιμων καταστάσεων όπου το control plane δεν είναι σε λειτουργία. Εστιάζουν επίσης στην απλοποίηση δημιουργίας πολιτικών ασφαλείας καθώς διασφαλίζουν και την ελαχιστοποίηση της πιθανότητας λάθους. Τέλος αναφέρονται στο ζήτημα της αρχιτεκτονικής όπου υπάρχει ένα κύριο σημείο κυβερνοεπιθέσεων (main point of cyber-attack).

Οι Cox et al. [6], χρησιμοποιούν τις δυνατότητες που προσφέρονται από το SDN για να εντοπίσουν και να εξουδετερώσουν κακόβουλους DHCP servers σε ένα δίκτυο. Αρχικά, αναλύουν τις δυσκολίες που υπάρχουν στον εντοπισμό ενός κακόβουλου DHCP server σε ένα παραδοσιακό δίκτυο, εστιάζοντας περισσότερο στην χρονοβόρα διαδικασία. Επίσης, αναφέρονται σε παραδοσιακές μεθόδους πρόληψης τέτοιων επιθέσεων και τις δυσκολίες που υπάρχουν ως προς την εφαρμογή τους στο δίκτυο. Για παράδειγμα, το DHCP snooping, το οποίο υποστηρίζεται από τις πλείστες συσκευές δικτύου, η εφαρμογή του αποτελεί μια επίπονη εργασία καθώς πρέπει να ρυθμιστεί μέσω CLI για κάθε port ξεχωριστά. Εάν αναλογιστούμε ότι σε ένα δίκτυο μπορεί να υπάρχουν και συσκευές από διαφορετικούς κατασκευαστές, γίνεται αντιληπτό ότι η διαδικασία είναι και επιρρεπής σε λανθασμένες ρυθμίσεις. Χρησιμοποιώντας τον POX controller και το πρωτόκολλο OpenFlow, προσομοιώνουν ένα ιδεατό δίκτυο (Mininet), τον controller, έναν ISC-DHCP server, έξι hosts (εκ των οποίων ο ένας αποτελεί τον κακόβουλο DHCP server) και ένα NAT. Παράλληλα, δημιουργούν ένα application σε γλώσσα Pyretic το οποίο αρχικά εξετάζει όλα τα πακέτα με DHCP headers (πχ Ethernet type = 2048, πρωτόκολλο UDP και source port = 67) και ακολούθως εξετάζεται η εγκυρότητα του *DHCPOFFER* μέσω ενός whitelist που καθορίζεται από τον network operator. Εάν είναι έγκυρο τότε γίνεται η προώθηση του πακέτου στον client διαφορετικά το πακέτο γίνεται drop.

Οι Zaballa et al. [7] προχωρούν ένα βήμα παραπέρα και παρουσιάζουν την ένταξη του SDN στο Fog και Edge Computing. Και οι δυο αυτές έννοιες αφορούν στην αποκεντροποίηση των δεδομένων (από το Cloud συνήθως) και την μεταφορά των δεδομένων όσο πιο κοντά γίνεται στον τελικό χρήστη. Με παρόμοιο τρόπο, στο Next-generation SDN αναπτύσσονται μηχανισμοί οι οποίοι δημιουργούν SDN control planes locally αλλά και remotely. Αυτό γίνεται εφικτό με την χρησιμοποίηση της γλώσσας P4 (data plane programming language) και του πρωτόκολλου P4Runtime το οποίο πλέον αντικαθιστά το OpenFlow και διαχειρίζεται P4 switches. Η εξέλιξη αυτή παρέχει νέες δυνατότητες στο data plane όπως για παράδειγμα να πραγματοποιούνται υπολογιστικές λειτουργίες σε line-speed. Όσον αφορά στους local και remote controllers, γίνεται

διαχωρισμός των ευθυνών τους. Για παράδειγμα, οι local controllers, μπορούν να προγραμματιστούν να απαντούν σε requests που χρήζουν άμεσης απάντησης (όπως συμβαίνει στο Edge Computing) ενώ ένας remote controller μπορεί να προγραμματιστεί να παίρνει αποφάσεις που αφορούν αποκλειστικά την συνολική εικόνα του δικτύου, όπως για παράδειγμα, τον υπολογισμό των δρομολογήσεων (path computation). Η δημιουργία αυτής της πολύ-επίπεδης (multi-layer) αρχιτεκτονικής των SDN control planes, εστιάζεται στην ενσωμάτωση του control plane στο data plane το οποίο είναι πιο κοντά στο network Edge ενώ οι συγγραφείς δίνουν παραδείγματα για το πώς μπορεί αυτή η αρχιτεκτονική να βελτιώσει τα Edge/Fog δίκτυα.

Οι Fan et al. [8], εστιάζουν στα τρωτά σημεία του SDN τόσο στο control plane όσο και στο data plane. Επίσης, αναφέρονται στην ασφάλεια των πρωτοκόλλων όπως για παράδειγμα το OpenFlow στο οποίο η κρυπτογράφηση δεν είναι υποχρεωτική. Επιπλέον αναφέρονται σε επιθέσεις DoS που μπορεί να υπάρξουν στον controller ο οποίος, ως μοναδικό σημείο αποτυχίας, αποτελεί τον κύριο στόχο των επιτιθέμενων. Στην έρευνα τους προσομοιάζουν τέσσερεις διαφορετικές επιθέσεις (Network Scanning, OpenFlow flooding, Switch compromised attack, ARP attack) και παρατηρούν την συμπεριφορά του δικτύου προτείνοντας ταυτόχρονα τρόπους ανίχνευσης των επιθέσεων.

Οι Shao et al. [9], προτείνουν την χρήση του SPBFT αλγορίθμου για να διασφαλίσουν την επικοινωνία μεταξύ των controllers. Χρησιμοποιούν την τεχνολογία του blockchain για να δημιουργήσουν μια κατανεμημένη βάση δεδομένων η οποία θα είναι αναγνώσιμη (readable), επεκτάσιμη (addable) και μη τροποποιήσιμη (unmodifiable) η οποία περιέχει μια λίστα με όλες τις διεργασίες που έγιναν στον κάθε controller. Δημιουργείται μια προσομοίωση από την οποία εξάγουν το συμπέρασμα της βελτίωσης της αποδοτικότητας καθώς και της ασφάλειας που προσδίδει ο SPBFT σε σχέση με τον PBFT.

Οι Xu et al. [10], εστιάζουν σε επιθέσεις DoS στον controller οι οποίες χαρακτηρίζονται και ως New-Flow Attacks. Οι επιθέσεις αυτές εκμεταλλεύονται συγκεκριμένα χαρακτηριστικά του SDN: Ο controller ως υπεύθυνος της δρομολόγησης, δημιουργεί νέα flows για κάθε πακέτο με καινούρια επικεφαλίδα. Κατά την υλοποίηση ενός New-Flow attack, ο επιτιθέμενος στέλνει κακόβουλα πακέτα με διαφορετικές επικεφαλίδες αναγκάζοντας τον controller να υπολογίζει και να δημιουργεί νέα flows για κάθε ένα από τα πακέτα. Αυτό έχει σαν αποτέλεσμα την εξάντληση των πόρων τόσο του controller όσο και των συσκευών δικτύου οδηγώντας έτσι σε DoS. Χρησιμοποιώντας στατιστικά ενός δικτύου, καθορίζουν το επιτρεπόμενο όριο των *Controller-to-Switch* και *Asynchronous* (*Packet-In* και *Flow Removed*) μηνυμάτων. Αξιοποιώντας αυτές τις

μετρήσεις ο εντοπισμός της επίθεσης γίνεται σε δύο επίπεδα. Αρχικά, σε περίπτωση που παρατηρηθεί σημαντική αύξηση των νέων flows τότε γίνεται και η εξέταση του αριθμού των πακέτων που εμπίπτουν σε κάθε νέο flow entry. Η σημαντική διαφορά που παρατηρείται σε κακόβουλα flows είναι ο αριθμός των πακέτων που το αποτελούν έτσι, σε περίπτωση που το flow αποτελείται από λιγότερα πακέτα από το αναμενόμενο μέσο όρο, τότε χαρακτηρίζεται ως ύποπτο και διαχωρίζεται από τα υπόλοιπα flows ενώ τα πακέτα συνεχίζουν να εξετάζονται από το security middleware που χρησιμοποιούν.

Οι Ursman et al. [11], εξετάζουν τον μετριασμό HTTP DoS επιθέσεων σε endpoints ενός SDN. Η λύση που προτείνουν έχει το σκεπτικό χρησιμοποίησης IDS για εντοπισμό της επίθεσης, κάτι το οποίο προτείνεται και στην παρούσα διατριβή έχοντας ωστόσο μεγάλες διαφορές ως προς την υλοποίηση του. Στην έρευνα τους, χρησιμοποιούν τον Ryu SDN Controller ο οποίος τους δίνει την δυνατότητα να χρησιμοποιήσουν απευθείας τα alerts του Snort μέσω του simple_switch_snort.py. Η επίθεση πραγματοποιείται με το εργαλείο slowloris-ng και οι πληροφορίες του alert στέλνονται στον controller μέσω network socket χρησιμοποιώντας το Pigrelay. Το IDS λειτουργεί στο ίδιο σύστημα μαζί με τον controller κάτι που σημαίνει ότι υπάρχει σημαντική επιβάρυνση του σε περίπτωση ενός Flooding attack. Επίσης, η απόρριψη των πακέτων γίνεται εξολοκλήρου από το source IP Address χωρίς να λαμβάνεται υπόψιν το πρωτόκολλο επικοινωνίας και δεν εξετάζεται ούτε ο τερματισμός της επίθεσης έτσι ώστε να επανέλθει η επικοινωνία. Ερωτηματικό επίσης αποτελεί και το τι συμβαίνει σε περίπτωση DDoS επίθεσης: Το IDS δημιουργεί alerts για κάθε source IP Address; Εάν η απάντηση είναι ναι τότε ο controller είναι σε θέση να δημιουργεί flows για κάθε alert χωρίς από μόνος του να δημιουργεί DoS τουλάχιστον του data plane;

# Κεφάλαιο 3
# Software Defined Networking

## 3.1 Περιγραφή και συστατικά του SDN

Όπως αναφέρθηκε και στην εισαγωγή, το κύριο χαρακτηριστικό του μοντέλου SDN, είναι ο διαχωρισμός της λήψης αποφάσεων για την δρομολόγηση των πακέτων και της ίδιας την δρομολόγησης. Σε αντίθεση με τα παραδοσιακά δίκτυα, όπου οι αποφάσεις και η δρομολόγηση-προώθηση των πακέτων γίνεται αποκλειστικά από τις διάφορες συσκευές δικτύου, πλέον στο SDN οι αποφάσεις λαμβάνονται αποκλειστικά από λογισμικό ανεξάρτητα από τις συσκευές δικτύου. Το λογισμικό αυτό αποτελεί τον controller του SDN ο οποίος επικοινωνεί συνεχώς με τις συσκευές του δικτύου - φυσικές ή/και εικονικές - χρησιμοποιώντας συγκεκριμένα πρωτόκολλα ενώ από τον διαχωρισμό που γίνεται, προκύπτει ο όρος control plane (καθορισμός δρομολόγησης) καθώς και το data plane (δρομολόγηση των δεδομένων).

### 3.1.1 Data Plane

Το Data Plane συχνά χαρακτηρίζεται και ως forwarding plane. Περιλαμβάνει όλες τις λειτουργίες που πρέπει να γίνουν για την επιτυχή επικοινωνία των συστημάτων. Οι λειτουργίες περιλαμβάνουν την λήψη των δεδομένων, την επεξεργασία και την προώθηση τους. Ασφαλώς, για να γίνει η αντιστοίχιση της διεύθυνσης προορισμού της πληροφορίας μαζί με κάποια καταχώρηση στο κατάλληλο table, θα πρέπει να γίνουν και όλες οι ενέργειες που γίνονται και σε ένα παραδοσιακό δίκτυο. Οι κυριότερες ενέργειες είναι το de-encapsulation και re-encapsulation του πακέτου σε frame (συσκευές Layer 3), προσθήκη ή αφαίρεση του 802.1Q trunking header εάν υπάρχουν VLANS (Layer 2 και Layer 3), αντιστοίχιση του destination MAC address ενός frame στο MAC Address table (Layer 2), την αντιστοίχιση του destination IP address στο IP routing table (Layer 3), την κρυπτογράφηση και την δημιουργία ενός IP header σε περίπτωση χρησιμοποίησης VPN, την αλλαγή του source ή destination IP address σε περιπτώσεις NAT, καθώς επίσης και την απόρριψη ενός πακέτου ή frame σύμφωνα με τα ACLs ή το port security [12].

Όλες οι πιο πάνω ενέργειες αποτελούν το Data Plane καθώς για κάθε ένα από τα πακέτα ή frames που εισέρχονται σε κάθε συσκευή, πρέπει να γίνουν οι απαιτούμενες ενέργειες έτσι ώστε να πάρει την τελική μορφή και να σταλεί στο κατάλληλο egress interface. Σε αντίθεση με το Control Plane, όπου για την επεξεργασία των ενεργειών που πρέπει να γίνουν χρησιμοποιείται το CPU, στο Data Plane υπάρχει ένας ειδικός επεξεργαστής που επεξεργάζεται την πληροφορία χωρίς να χρησιμοποιείται το CPU [12].

### 3.1.2 Control Plane

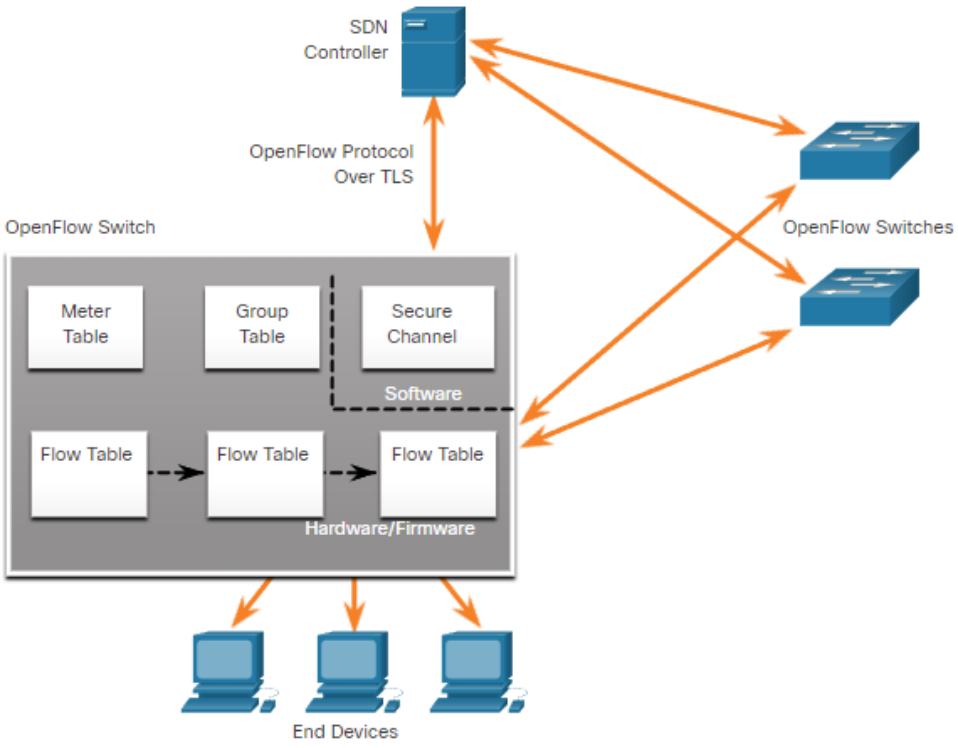
Το Control Plane μπορεί να χαρακτηριστεί ως ο εγκέφαλος του συστήματος. Χρησιμοποιείται για να παίρνει τις αποφάσεις δρομολόγησης καθώς επικοινωνεί με τα SDN applications καθώς και με τις συσκευές δικτύου. Περιλαμβάνει μηχανισμούς και πρωτόκολλα προώθησης σε Layer 2 αλλά και Layer 3 όπως για παράδειγμα routing protocol neighbour tables και topology tables, IPv4 και IPv6 routing tables, STP και ARP tables.

Όπως συμβαίνει και σε ένα παραδοσιακό δίκτυο, όπου οι routers χρησιμοποιούν τα IP routes του routing table για να προωθήσουν τα πακέτα, τα switches χρειάζονται τις πληροφορίες του MAC address table για να προωθήσουν τα frames έτσι συμβαίνει και στην αρχιτεκτονική του SDN. Το Control Plane θα δημιουργήσει αυτά τα tables τα οποία θα χρησιμοποιηθούν από το data plane για την δρομολόγηση της πληροφορίας (πακέτα ή frames) [13].

### 3.1.3 SDN Controller

Ο SDN controller καθορίζει τις ροές δεδομένων (data flows) μεταξύ του κεντρικού control plane και του data plane σε κάθε συσκευή δικτύου. Η κάθε ροή που υπάρχει στο δίκτυο χρειάζεται πρώτα να πάρει την άδεια από τον SDN controller ο οποίος είναι υπεύθυνος να επαληθεύσει ότι η επικοινωνία είναι επιτρεπτή σύμφωνα με την καθορισμένη πολιτική δικτύου. Εάν ο ελεγκτής επιτρέπει μια ροή, υπολογίζει την διαδρομή που θα ακολουθηθεί και προσθέτει την κατάλληλη καταχώρηση σε κάθε μια από τις συσκευές δικτύου κατά μήκος της διαδρομής [12].

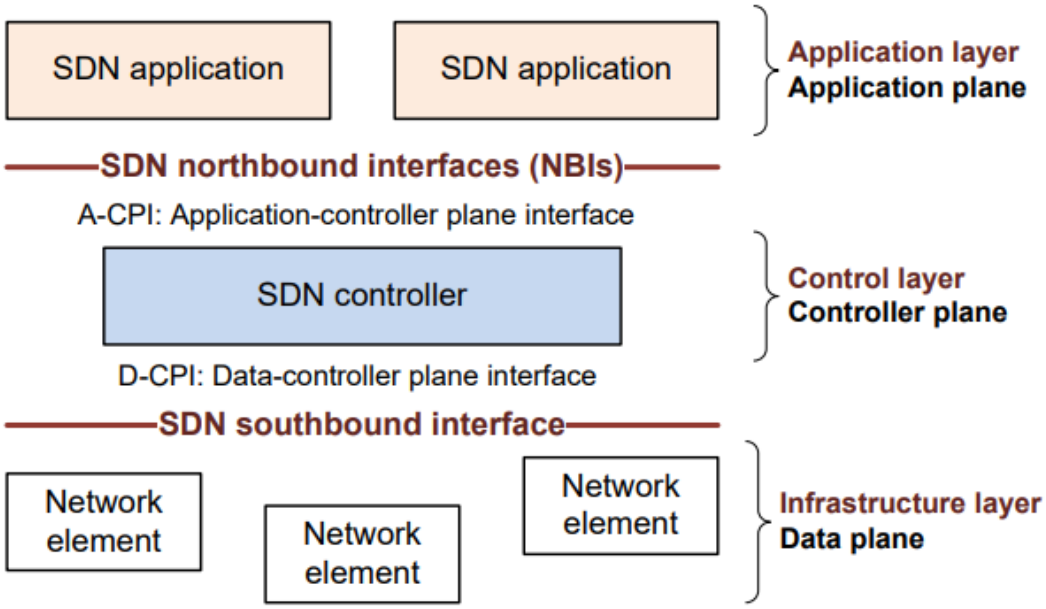
Όλες οι πολύπλοκες λειτουργίες εκτελούνται από τον SDN controller ο οποίος διαμορφώνει τους πίνακες ροής (flow tables) ενώ η διαχείριση τους γίνεται από τις συσκευές δικτύου. Όπως φαίνεται στο πιο κάτω σχήμα, ο SDN controller επικοινωνεί με switches τα οποία είναι συμβατά με το πρωτόκολλο OpenFlow το οποίο χρησιμοποιεί TLS για την ασφαλή αποστολή της επικοινωνίας του control plane στο υπόλοιπο δίκτυο. Τα συμβατά switches μπορούν να ενωθούν μεταξύ τους καθώς επίσης και με τους τελικούς χρήστες οι οποίοι αποτελούν μέρος μιας ροής πακέτων.




**Εικόνα 3.1:** Επικοινωνία SDN controller με επιμέρους συσκευές [12].

Η πιο πάνω επικοινωνία, στην αρχιτεκτονική του SDN συναντάται με τον όρο D-CPI όπου τα Southbound APIs επιτρέπουν στο controller να επικοινωνεί με τα routers και switches

και να πραγματοποιεί δυναμικά αλλαγές σύμφωνα με τις απαιτήσεις και τις ανάγκες σε πραγματικό χρόνο. Από την άλλη, υπάρχει και ο όρος A-CPI όπου τα Northbound APIs επιτρέπουν την επικοινωνία των SDN applications με τον controller. Τα SDN Applications ζητούν από το δίκτυο τους πόρους που χρειάζονται και ο controller θα αναλάβει να παραδώσει τους πόρους ή να επικοινωνήσει στο SDN Application τους διαθέσιμους πόρους του δικτύου [14].




**Εικόνα 3.2:** NBIs και SBIs στην αρχιτεκτονική του SDN [11].

### 3.1.4 OpenFlow Protocol

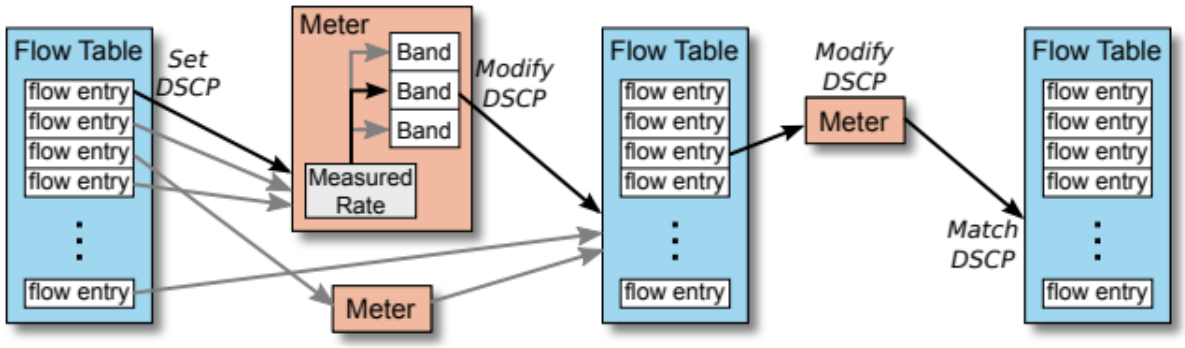
Το πρωτόκολλο OpenFlow είναι το πιο διαδεδομένο, ανοικτού κώδικα, πρωτόκολλο επικοινωνίας που χρησιμοποιείται σε τεχνολογίες SDN. Τα OpenFlow switches αποτελούνται από τα Flow και Group tables (υπεύθυνα για την δρομολόγηση), τα Meter tables καθώς και από τουλάχιστον ένα OpenFlow channel το οποίο επικοινωνεί με τον controller. Το switch επικοινωνεί με τον controller και ο controller διαχειρίζεται το switch μέσω του πρωτοκόλλου OpenFlow επιτρέποντας έτσι στον controller να προσθέτει, να ενημερώνει και να διαγράφει τις καταχωρήσεις ροής στα flow tables. Το κάθε flow table περιέχει τις καταχωρήσεις ροής (flow entries) οι οποίες αποτελούνται από τα πεδία match fields, counters και ένα σύνολο κανόνων (instructions) που εφαρμόζονται για κάθε ένα από τα πακέτα που αντιστοιχίζονται με κάθε flow [15].

Η ενέργεια που καθορίζεται σε ένα flow entry μπορεί να δρομολογήσει απευθείας τα πακέτα ή να τα παραπέμψει στο group table εάν χρειάζεται να γίνει κάποια πιο συγκεκριμένη επεξεργασία. Το κάθε group table, περιέχει με την σειρά του τα group entries τα οποία αποτελούνται από action buckets με συγκεκριμένες λειτουργίες (semantics) αναλόγως του group type. Το κάθε group entry μπορεί να αποτελείται από μηδέν ή περισσότερα buckets εκτός από το *indirect* group το οποίο αποτελείται πάντα από ένα action bucket. Το κάθε OpenFlow switch πρέπει να υποστηρίζει τουλάχιστον τα δύο group types *indirect* και *all* ενώ τα group types *select* και *fast failover* είναι προαιρετικά. Ο controller ενημερώνεται από το κάθε OpenFlow switch για τα προαιρετικά group types τα οποία υποστηρίζει.

Σύμφωνα με το OpenFlow Switch Specification [15], συνοπτικά να αναφέρουμε ότι το *indirect* group type αποτελείται από ένα bucket το οποίο επιτρέπει σε διάφορα flow entries ή groups να αντιστοιχούν σε ένα group identifier επιτυγχάνοντας έτσι μια ταχύτερη και πιο αποτελεσματική σύγκλιση. Η λειτουργία του είναι ακριβώς η ίδια με ένα group type *all* το οποίο περιέχει μόνο ένα bucket το οποίο για παράδειγμα μπορεί να δίνει το next hop σε περίπτωση IP forwarding. Το group type *all* εκτελεί όλα τα buckets που περιέχονται σε αυτό. Χρησιμοποιείται για Multicast ή Broadcast καθώς γίνεται η κλωνοποίηση του αρχικού πακέτου σε όλα τα buckets του group. Σε περίπτωση που ένα bucket προωθεί το πακέτο στο ingress port, τότε γίνεται αυτόματα dropped έτσι εάν πρέπει να γίνει η προώθηση του πακέτου στη θύρα εισόδου θα πρέπει να δημιουργηθεί από τον controller ένα επιπλέον bucket το οποίο θα περιέχει την ενέργεια προώθησης στο reserved port (OFPP_IN_PORT). Το group type *select* εκτελεί ένα bucket το οποίο επιλέγεται βάση αλγορίθμου στο switch. Ο προγραμματισμός του αλγορίθμου γίνεται εκτός του πρωτοκόλλου OpenFlow και εφαρμόζει ένα καταμερισμό του φορτίου ενώ προαιρετικά μπορεί να χρησιμοποιηθεί και η σημαντικότητα του κάθε bucket έτσι ώστε να γίνει η επιλογή αναλόγως της χρήσης του. Όταν για παράδειγμα ένα destination port κάποιου bucket βρεθεί εκτός λειτουργίας, το switch μπορεί να αποκλείσει το συγκεκριμένο bucket, επιλέγοντας κάποιο άλλο το οποίο θα κάνει την δρομολόγηση μέσω κάποιου live port. Τέλος, το group type *fast failover*, εκτελεί το πρώτο στη σειρά live bucket. Η σειρά των buckets, τα οποία σχετίζονται με ένα live port ή ένα άλλο group, αξιολογείται από το κάθε group ξεχωριστά. Αυτό έχει σαν αποτέλεσμα, το OpenFlow switch να αλλάζει δυναμικά τον τρόπο δρομολόγησης χωρίς να απαιτείται το round trip στον controller για να καθορίσει την δρομολόγηση. Σε περίπτωση που δεν υπάρχει κάποιο live bucket, τότε τα πακέτα γίνονται dropped.

Επιπλέον, ένα OpenFlow switch περιέχει και το Meter Table. Αυτός ο πίνακας, ενεργοποιεί μια ποικιλία ενεργειών οι οποίες σχετίζονται με την αποδοτικότητα ενός flow συμπεριλαμβανομένης

και της δυνατότητας περιορισμού του ρυθμού κυκλοφορίας (rate-limiting) [12]. Το Meter Table αποτελείται από τα meter entries τα οποία καθορίζουν την ροή των flows. Δημιουργούνται meters ξεχωριστά για κάθε flow που υπάρχει δίνοντας έτσι την ικανότητα στο OpenFlow να διαχειρίζεται τον ρυθμό κυκλοφορίας καθιστώντας έτσι εφικτό το QoS το οποίο μπορεί να έχει μια απλή λειτουργία, όπως για παράδειγμα το περιορισμό του bandwidth για ένα σύνολο flows, ή ακόμη και περισσότερο σύνθετες λειτουργίες όπως την χρησιμοποίηση των τιμών του DSCP (Differentiated Services Code Point) που μπορεί να ταξινομήσει ένα σύνολο πακέτων σε διαφορετικές κατηγορίες με βάση το rate του, όπως συμβαίνει δηλαδή και κατά την ρύθμιση του QoS σε μια φυσική συσκευή δικτύου [15].




**Εικόνα 3.3:** Meters και Ιεραρχία DSCP metering [12].

Όπως αναφέραμε πιο πάνω, η επικοινωνία του OpenFlow controller με το OpenFlow switch, επιτυγχάνεται μέσω του OpenFlow channel. Μέσω αυτού του interface, ο controller διαμορφώνει και διαχειρίζεται το switch, λαμβάνει διάφορα events από το switch καθώς επίσης στέλνει και τα πακέτα από το switch. Το Control Channel του κάθε switch μπορεί να υποστηρίζει και περισσότερα από ένα OpenFlow channel, δίνοντας έτσι την δυνατότητα της διαχείρισης του switch από περισσότερους controllers. Η επικοινωνία του controller με το OpenFlow channel μπορεί να γίνει απευθείας μέσω TCP, ωστόσο παρέχεται και η δυνατότητα κρυπτογράφησης της επικοινωνίας, χρησιμοποιώντας το πρωτόκολλο TLS, κάτι το οποίο θα πρέπει να εφαρμόζεται σε production environments.

Το πρωτόκολλο OpenFlow, υποστηρίζει τρείς διαφορετικούς τύπους μηνυμάτων (message types) ενώ ο κάθε τύπος αποτελείται από διαφορετικούς υποτύπους (sub-types). Σύμφωνα λοιπόν με OpenFlow Switch Specification [Ibid.], το πρωτόκολλο υποστηρίζει τους εξής τύπους μηνυμάτων: *controller-to-switch, asynchronous* και *symmetric*. Οι ονομασίες από μόνες τους, φανερώνουν κάποιες πληροφορίες ως προς την λειτουργία των μηνυμάτων ωστόσο πιο κάτω παραθέτουμε συνοπτικά πληροφορίες για το κάθε message type.

Ξεκινώντας λοιπόν από το *controller-to-switch*, τα μηνύματα αυτού του τύπου ξεκινούν από τον controller και χρησιμοποιούνται για την άμεση διαχείριση ή την επιθεώρηση της κατάστασης του switch. Αναλόγως του μηνύματος μπορεί να χρειαστεί και η απάντηση από το switch. Τα sub-types των μηνυμάτων είναι:

i. Features: Χρησιμοποιείται κυρίως κατά την δημιουργία του OpenFlow channel. Ο controller ζητά από το switch την ταυτότητα του καθώς επίσης και τις δυνατότητες του στέλνοντας το features request. Με το features reply, το switch πρέπει να στείλει τις πληροφορίες στον controller.

ii. Configuration: Χρησιμοποιείται από τον controller για να ορίσει αλλά και να ρωτήσει για παραμέτρους διαμόρφωσης στο switch. Το switch απαντά μόνο στο ερώτημα για τις παραμέτρους και όχι κατά τον ορισμό τους.

iii. Modify-State: Χρησιμοποιείται από τον controller για την διαχείριση των tables στο switch καθώς επίσης για να ορίσει τις ιδιότητες του κάθε port. Έτσι, ο κύριος σκοπός των μηνυμάτων αυτών είναι να προσθέσουν, να διαγράψουν ή/και να τροποποιήσουν τα flow και group entries καθώς επίσης και να προσθέσουν ή να διαγράψουν τα action buckets από κάποιο group.

iv. Read-State: Χρησιμοποιούνται από τον controller για να πάρει πληροφορίες από το switch όπως τα configurations που υπάρχουν, διάφορα στατιστικά στοιχεία καθώς επίσης για τις δυνατότητες του switch. Συνήθως σε αυτή την επικοινωνία δημιουργείται μια αλληλουχία μηνυμάτων μεταξύ του controller και του switch.

v. Packet-out: Χρησιμοποιείται από τον controller για να προωθήσει τα πακέτα μέσω ενός συγκεκριμένου port στο switch. Τα μηνύματα αυτά μπορεί να περιέχουν ολόκληρο το πακέτο ή απλά ένα buffer ID που αντιστοιχεί σε ένα πακέτο που βρίσκεται στο switch. Το μήνυμα πρέπει να περιλαμβάνει και τις ενέργειες που πρέπει να γίνουν από το switch οι οποίες εκτελούνται ιεραρχικά όπως ορίστηκαν από τον controller. Σε περίπτωση που ο controller δεν δώσει κάποια ενέργεια που πρέπει να γίνει, τότε το πακέτο απορρίπτεται.

vi. Barrier: Η ανταλλαγή αυτών των μηνυμάτων μεταξύ του controller και του switch (request και reply αντίστοιχα) χρησιμοποιούνται από τον controller για την επιβεβαίωση των σωστών και ολοκληρωμένων λειτουργιών που πραγματοποιήθηκαν.

vii. Role-Request: Χρησιμοποιείται κυρίως σε περιπτώσεις που ένα switch επικοινωνεί με περισσότερους από ένα controller ο οποίος με αυτά τα μηνύματα μπορεί να ορίσει ή να ρωτήσει για τον δικό του ρόλο στο OpenFlow channel καθώς επίσης και για το δικό του ID.

viii. Asynchronous-Configuration: Χρησιμοποιείται, επίσης, κυρίως σε περιπτώσεις που υπάρχουν περισσότεροι από ένα controller. Με αυτά τα μηνύματα ο controller μπορεί να ορίσει ή και να ρωτήσει για επιπλέον φίλτρα ασύγχρονων μηνυμάτων που θέλει να λαμβάνει στο δικό του OpenFlow channel. Συνήθως αυτό συμβαίνει κατά την δημιουργία του OpenFlow channel.

Ο δεύτερος τύπος μηνυμάτων είναι τα asynchronous τα οποία στέλνονται από το switch στον controller χωρίς πρώτα να ζητηθούν. Τα μηνύματα αυτά πληροφορούν τον controller για την άφιξη νέων πακέτων ή για την αλλαγή στο state του switch. Οι κύριοι τύποι μηνυμάτων αυτής της κατηγορίας είναι οι ακόλουθοι:

i. Packet-in: Αποτελεί τα πιο σημαντικά μηνύματα που στέλνονται από το switch προς τον controller. Χρησιμοποιούνται κυρίως για να ενημερώσουν τον controller για την άφιξη ενός πακέτου και να μεταφέρουν τον έλεγχο του στον controller. Για όλα τα πακέτα τα οποία περνούν από τα tables του switch δημιουργείται ένα packet-in event το οποίο στέλνεται στον controller. Επίσης packet-in events μπορεί να αποσταλούν στον controller για περαιτέρω επεξεργασία όπως για παράδειγμα για τον έλεγχο του TTL. Τα packet-in events μπορεί να ρυθμιστούν έτσι ώστε να κάνουν buffer των πακέτων χωρίς να χρειάζεται να στέλνεται όλο το πακέτο στον controller. Για πακέτα των οποίων η επεξεργασία γίνεται από κάποιο flow entry ή από κάποιο group bucket η ρύθμιση αυτή μπορεί να γίνει απευθείας από το output αυτών των entries ενώ για τα υπόλοιπα πακέτα η ρύθμιση πραγματοποιείται στις ρυθμίσεις του switch. Σε περίπτωση που γίνει αυτή η ρύθμιση, και το switch έχει αρκετή μνήμη έτσι ώστε να δημιουργήσει το buffer, τότε στέλνεται στον controller ένα μικρό ποσοστό της επικεφαλίδας του πακέτου μαζί με ένα buffer ID. Σε κάθε άλλη περίπτωση το πακέτο στέλνεται ολόκληρο στον controller ως μέρος του packet-in event.

ii. Flow-Removed: Χρησιμοποιούνται από το switch για να ενημερώσει τον controller για την διαγραφή ενός flow entry τα οποία έχουν ενεργοποιημένο το OFFF_SEND_FLOW_REM flag κατά την δημιουργία τους. Αυτά τα μηνύματα συνήθως δημιουργούνται όταν ο controller

ζητήσει την διαγραφή ενός flow entry ή όταν υπερβεί ο προκαθορισμένος χρόνος ύπαρξης του.

iii. Port-status: Πραγματοποιούν την ενημέρωση προς τον controller για αλλαγές που μπορεί να συμβούν σε κάποιο port του switch όπως για παράδειγμα εάν κάποιος χρήστης κλίσει κάποιο port ή ακόμη και αν παρουσιαστεί κάποιο σφάλμα στο link. Επίσης μπορεί να γίνει ρύθμιση στο switch έτσι ώστε να στέλνει στον controller ανά τακτά διαστήματα πληροφορίες για την κατάσταση του κάθε port.

iv. Role-status: Τα μηνύματα αυτά δημιουργούνται κυρίως στην ύπαρξη περισσότερων από ένα controller. Σε περίπτωση που κάποιος από τους controller επιλεχθεί ως master, τότε στέλνονται τα μηνύματα αυτά στον προηγούμενο master controller για να τον ενημερώσει για τον νέο του ρόλο.

v. Controller-Status: Κάθε φορά που υπάρχει κάποια αλλαγή στο OpenFlow channel, τότε το switch στέλνει απευθείας σε όλους τους controllers τα controller-status μηνύματα έτσι ώστε να τους ενημερώσει για αυτές τις αλλαγές. Αυτό βοηθά και στην περίπτωση που για οποιοδήποτε λόγο οι controllers χάσουν την μεταξύ τους επικοινωνία.

vi. Flow-monitor: Χρησιμοποιούνται από το switch για να ενημερώσει τον controller σε κάποια αλλαγή σε ένα flow table. Ο controller μπορεί να καθορίσει στο switch το επίπεδο παρακολούθησης των αλλαγών σε κάθε flow table και να ενημερώνεται αναλόγως.

Τέλος, υπάρχουν τα symmetric messages τα οποία μπορεί να δημιουργηθούν χωρίς να ζητηθούν πρώτα από τον controller ή από το switch. Αυτή η κατηγορία περιλαμβάνει τους ακόλουθους τέσσερεις τύπους μηνυμάτων:

i. Hello: Ανταλλάσσονται μεταξύ του controller και του switch κατά την έναρξη της επικοινωνίας.

ii. Echo: Όπως τα παραδοσιακά ICMP μηνύματα που ανταλλάσσονται μεταξύ controller και switch. Το κάθε echo request απαιτεί και ένα echo reply για να επιβεβαιωθεί η επικοινωνία. Τα μηνύματα αυτά μπορούν επίσης να χρησιμοποιηθούν για την μέτρηση του bandwidth ή του latency.

iii. Error: Χρησιμοποιούνται από το switch ή από τον controller για να ενημερώσουν για προβλήματα από την μεριά τους. Συνήθως δημιουργούνται όταν το switch αποτύχει να εκπληρώσει κάποιο request του controller.

iv. Experimenter: Αποτελεί ένα τυπικό τρόπο για τα OpenFlow switches ο οποίος προσφέρει πειραματικό χώρο για την βελτίωση ή και την ανάπτυξη μελλοντικών μηνυμάτων.

# Κεφάλαιο 4
# Υλοποίηση Συστήματος

## 4.1 Testbed Deployment

Η προσομοίωση έγινε σε περιβάλλον private cloud (OpenStack) μέσα στο οποίο δημιουργήθηκε ένα private network όπου αναπτύχθηκαν δύο συστήματα σε λειτουργικό Ubuntu (cloud) 20.04. Το πρώτο σύστημα χρησιμοποιήθηκε για την δημιουργία του Controller (Onos) και το δεύτερο για τη δημιουργία προσομοίωσης δικτύου (Mininet) όπου και έγιναν οι διάφορες δοκιμές.

### 4.1.1 Εγκατάσταση OpenStack

Το OpenStack είναι ένα σύνολο λογισμικών ανοικτού κώδικα τα οποία στο σύνολο τους δίνουν την δυνατότητα δημιουργίας υποδομής computing cloud. Υπάρχουν διάφοροι τρόποι εγκατάστασης των βασικών αλλά και προαιρετικών υπηρεσιών του OpenStack αναλόγως της χρήσης του αλλά και του διαθέσιμου hardware. Για την παρούσα εργασία προτιμήθηκε ως

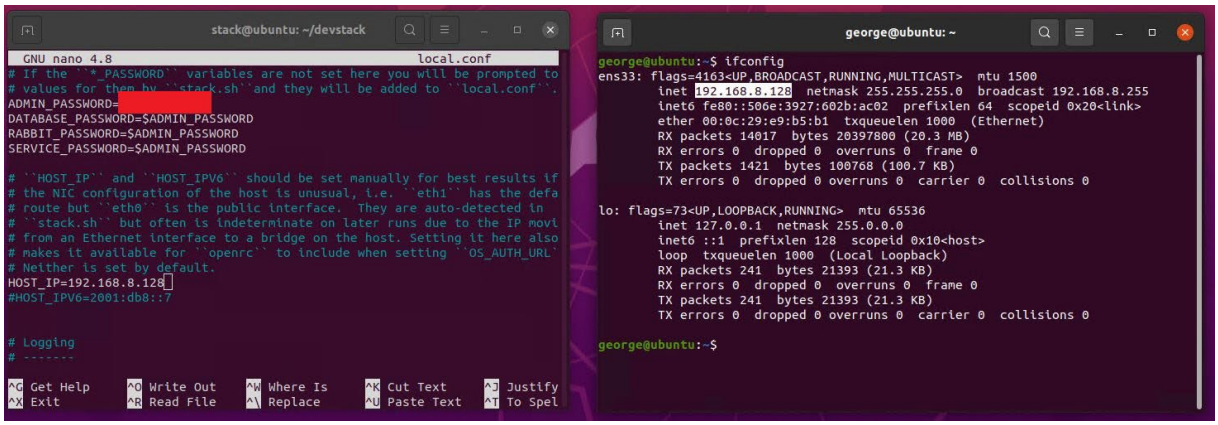
deployment method το DevStack το οποίο αποτελεί μια σειρά από scripts τα οποία εγκαθιστούν όλα τα απαιτούμενα λογισμικά βασιζόμενα στις τελευταίες τους εκδόσεις στο GitHub.

Η εγκατάσταση[13] συστήνεται να γίνεται σε συστήματα τα οποία θα λειτουργούν αποκλειστικά για αυτό το σκοπό καθώς γίνονται αρκετές αλλαγές στο λειτουργικό κατά την εγκατάσταση. Για την παρούσα εργασία θεωρήθηκε προτιμότερο να εγκατασταθεί σε εικονικό μηχάνημα Ubuntu 20.04, χρησιμοποιώντας το VMware Player το οποίο εγκαταστάθηκε σε ένα Dedicated Root server. Η εγκατάσταση έγινε σε Virtual περιβάλλον και όχι απευθείας στον server για κυρίως δύο λόγους: Κατά την εγκατάσταση του OpenStack γίνονται αρκετές αλλαγές στα network interfaces και απαιτούνται διάφορες DHCP λειτουργίες στο physical network interface. Αυτό όμως δεν είναι εφικτό καθώς ο ενοικιαζόμενος server είναι ρυθμισμένος να χρησιμοποιεί ένα static public IP. Συνεπώς, χρειάζεται ένα ενδιάμεσο vSwitch το οποίο θα επιτρέψει να γίνουν οι απαιτούμενες αλλαγές και να δοθούν τα κατάλληλα private IP addresses. Ο δεύτερος λόγος που επιλέχθηκε το Virtual περιβάλλον, είναι για πρακτικούς κυρίως λόγους καθώς είναι ευκολότερο να δημιουργηθούν backups ή snapshots του συστήματος κατά την υλοποίηση της προσομοίωσης. Το VMware Player μας καλύπτει και τις δύο αυτές ανάγκες.

Για την ομαλή εγκατάσταση σε Linux απαιτείται η δημιουργία νέου χρήστη του οποίου το home directory καθορίζεται στο /opt/stack και η ένταξη του στο group sudo χωρίς να απαιτείται συνθηματικό. Αφού δημιουργηθεί ο χρήστης, γίνονται clone τα scripts στο /opt/stack όπου περιλαμβάνεται και ένα sample configuration (local.conf στον φάκελο samples) το οποίο θα χρησιμοποιηθεί κατά την εγκατάσταση. Αφού αντιγραφεί από το local.conf στο /opt/stack μπορούμε να το επεξεργαστούμε και να καθορίσουμε τα συνθηματικά για τις διάφορες υπηρεσίες που απαιτούνται κατά την διάρκεια της εγκατάστασης καθώς και το host IP address. Όπως φαίνεται και πιο κάτω, επιλέχθηκε ένα admin password το οποίο θα χρησιμοποιηθεί και για τις υπόλοιπες τρεις υπηρεσίες. Στην επιλογή του συνθηματικού θα πρέπει να αποφευχθεί ο χαρακτήρας @ καθώς παρατηρήθηκε ότι δημιουργεί προβλήματα κατά την διάρκεια της δημιουργίας του SQL Database (πιθανόν να μην χρησιμοποιούνται κάποια quotes (“ ”) στο script με αποτέλεσμα να προκύπτει κάποια εντολή σαν: mysql –uroot –pxxxxxxx@ -h127.0.0.1 κτλ)

```
george@ubuntu:~$ sudo -i
root@ubuntu:~#  useradd -s /bin/bash -d /opt/stack -m stack
root@ubuntu:~# echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
stack ALL=(ALL) NOPASSWD: ALL
root@ubuntu:~# su - stack
stack@ubuntu:~$ git clone https://opendev.org/openstack/devstack
Cloning into 'devstack'...
remote: Enumerating objects: 27794, done.
remote: Counting objects: 100% (27794/27794), done.
remote: Compressing objects: 100% (9346/9346), done.
remote: Total 48244 (delta 27130), reused 18448 (delta 18448), pack-reused 20450
Receiving objects: 100% (48244/48244), 10.30 MiB | 11.02 MiB/s, done.
Resolving deltas: 100% (33917/33917), done.
stack@ubuntu:~$ cd devstack/
stack@ubuntu:~/devstack$ cp samples/local.conf .
stack@ubuntu:~/devstack$ nano local.conf
stack@ubuntu:~/devstack$ FORCE=yes ./stack.sh
```

**Εικόνα 4.1:** Συνοπτική διαδικασία εγκατάστασης.




**Εικόνα 4.2:** Επεξεργασία local.conf πριν  από την εγκατάσταση.

Αφού ολοκληρωθεί επιτυχώς η εγκατάσταση (χρειάζονται περίπου 35 λεπτά) , θα εμφανιστεί το ανάλογο μήνυμα το οποίο περιέχει τις διευθύνσεις που τρέχουν οι υπηρεσίες καθώς και η έκδοση του OpenStack η οποία εγκαταστάθηκε (yoga στην προκειμένη περίπτωση):

```
This is your host IP address: 192.168.8.128
This is your host IPv6 address: ::1
Horizon is now available at http://192.168.8.128/dashboard
Keystone is serving at http://192.168.8.128/identity/
The default users are: admin and demo
The password: [redacted]

Services are running under systemd unit files.
For more information see:
https://docs.openstack.org/devstack/latest/systemd.html

DevStack Version: yoga
Change: b6656b7b38db212d6aa471aa01a9cfaf6024d64b Merge "Clean up compile_ovn function's parameters"
OS Version: Ubuntu 20.04 focal
```

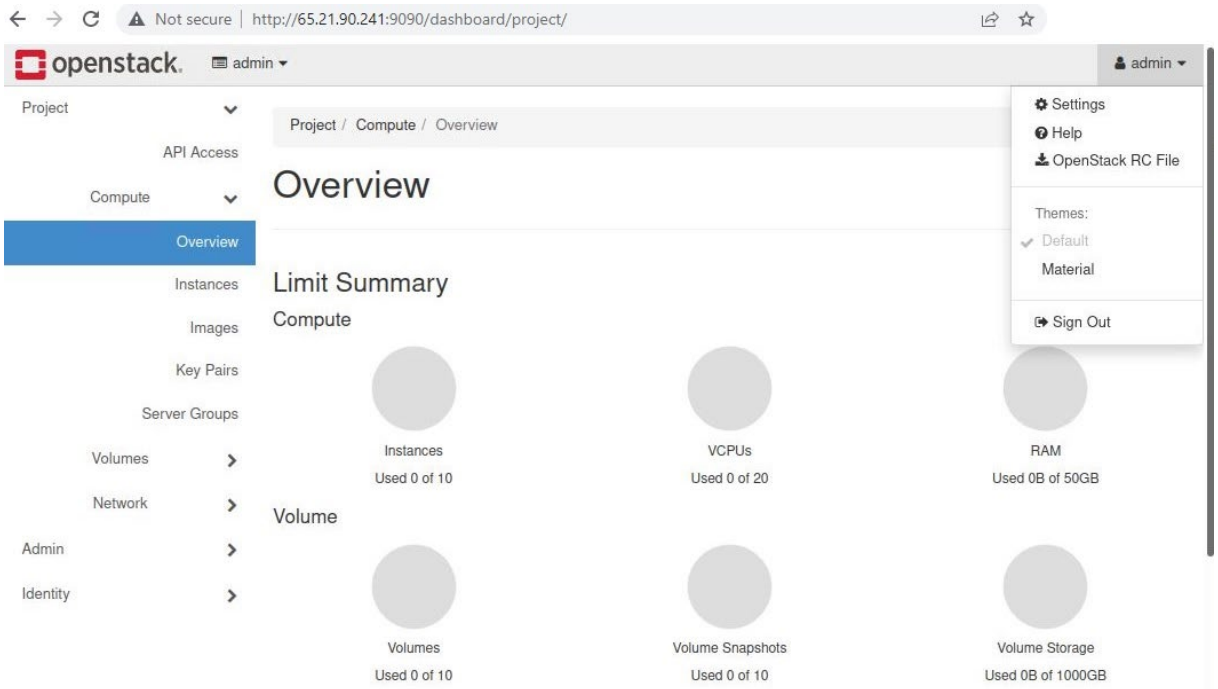
**Εικόνα 4.3:** Επιτυχής εγκατάσταση OpenStack.

Ακολούθως έγιναν και κάποιες επιπλέον ενέργειες (είναι υποχρεωτικές σε περίπτωση που η εγκατάσταση γίνεται σε περιβάλλον χωρίς κάποιο desktop) για την απομακρυσμένη σύνδεση στο dashboard του OpenStack:

```
PS C:\Windows\system32> netsh interface portproxy add v4tov4 listenaddress=65.21.90.241 listenport=9090
 connectaddress=192.168.8.128 connectport 80

PS C:\Windows\system32> netsh interface portproxy show all

Listen on ipv4:             Connect to ipv4:

Address         Port        Address         Port
--------------- ----------  --------------- ----------
65.21.90.241    9090        192.168.8.128   80

PS C:\Windows\system32> netsh advfirewall firewall add rule name="TCP Port 9090 OpenStack" dir=in actio
n=allow protocol=TCP  localport=9090
Ok.
```

**Εικόνα 4.4:** Port forwarding από τον Server:9090 στον VMware Guest:80 και δημιουργία firewall rule για το port 9090.

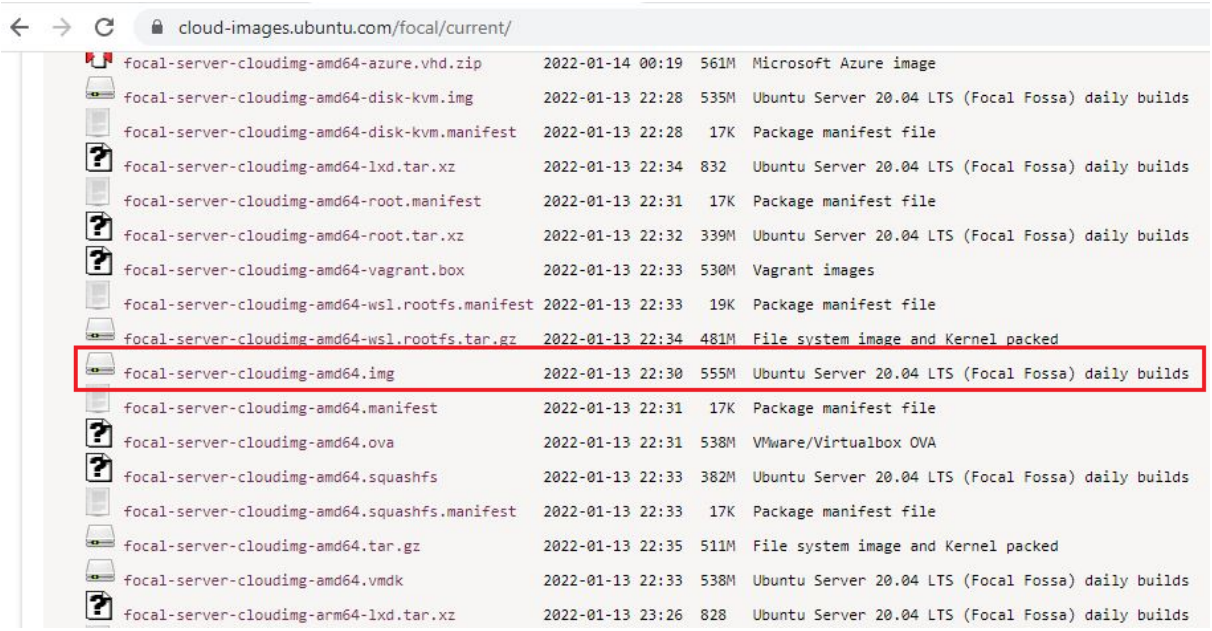
Αφού γίνουν οι πιο πάνω ρυθμίσεις, η πρόσβαση στο dashboard του OpenStack επιτυγχάνεται με την πλοήγηση στο http://*ip*:*port*/dashboard. Όνομα χρήστη είναι το admin και το συνθηματικό το οποίο ορίστηκε στο local.conf πριν από την έναρξη της εγκατάστασης:




**Εικόνα 4.5:** Πρόσβαση στο dashboard του OpenStack.

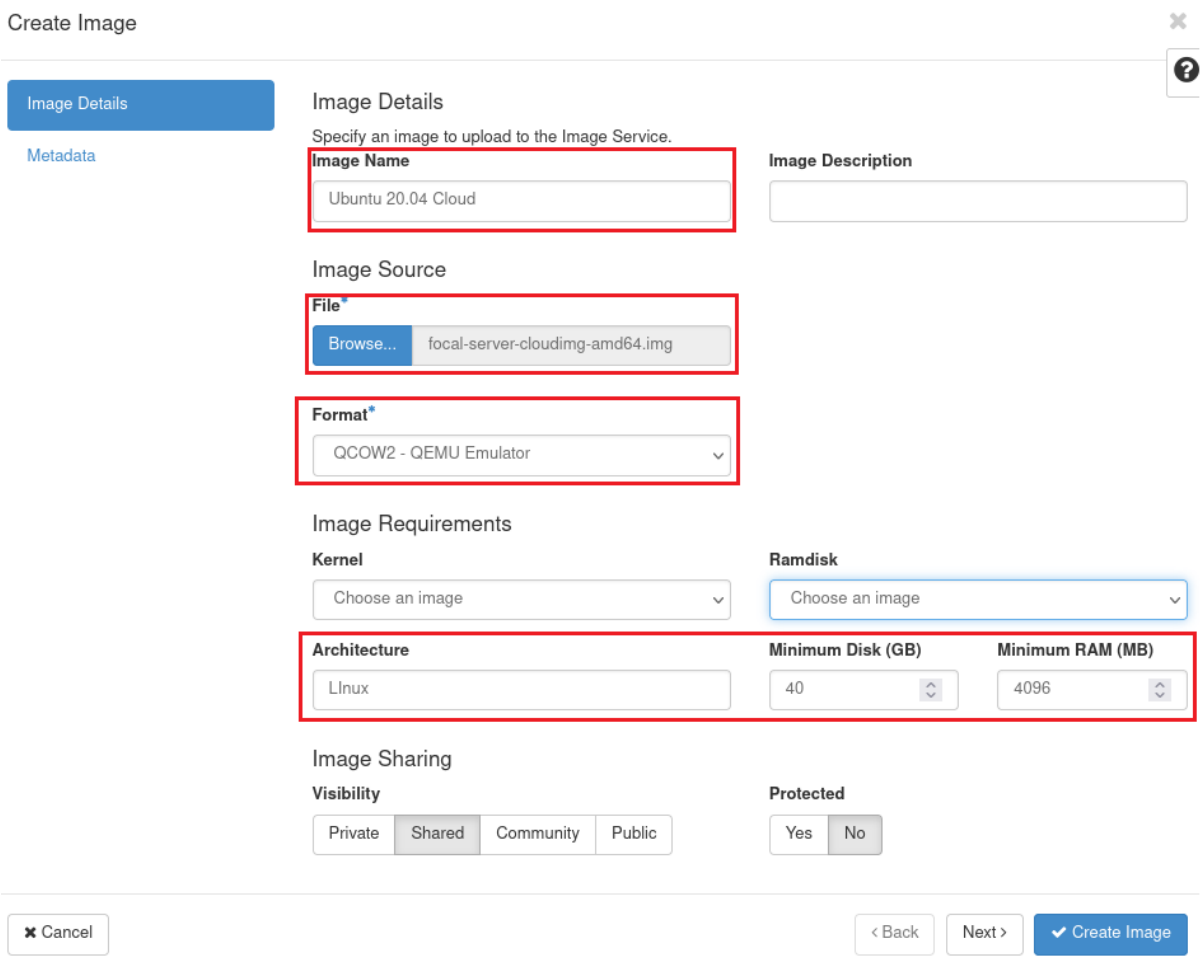
### 4.1.2 Δημιουργία των Instances

Μέσω του OpenStack dashboard, δημιουργήθηκαν δύο Instances χρησιμοποιώντας το cloud image του Ubuntu Server 20.04 [14]. Για QEMU Emulator το οποίο θα χρησιμοποιηθεί αργότερα στο OpenStack, συστήνεται το format .img:



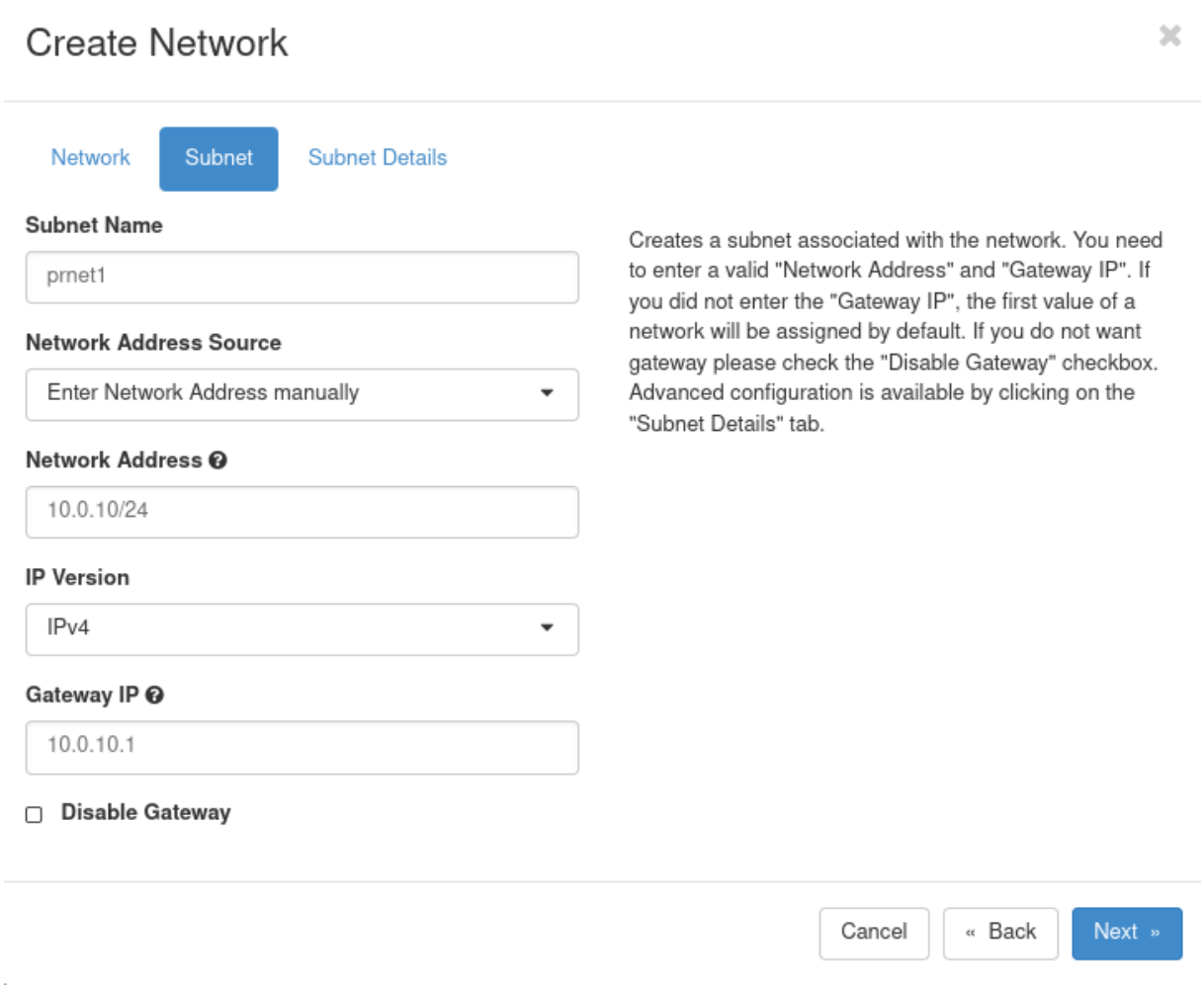

**Εικόνα 4.6:** Επιλογή OS Image

Στη συνέχεια δημιουργούμε το image στο OpenStack (χρησιμοποιώντας το image το οποίο επιλέξαμε πιο πάνω) επιλέγοντας Compute – Images – Create Image. Ακολούθως, στο Image Name χρησιμοποιούμε το όνομα του λειτουργικού συστήματος και στο Image Source επιλέγουμε το image που κατεβάσαμε νωρίτερα, στο Format επιλέγουμε QCOW2 – QEMU Emulator. Προαιρετικά μπορούμε να καθορίσουμε και το ελάχιστο Volume Disk και RAM το οποίο μπορεί να έχει το κάθε instance που θα δημιουργηθεί βασιζόμενο στο συγκεκριμένο image καθώς και την αρχιτεκτονική του λειτουργικού συστήματος. Αφού δώσουμε τις πληροφορίες επιλέγουμε Create Image για να δημιουργηθεί το πρώτο μας image.

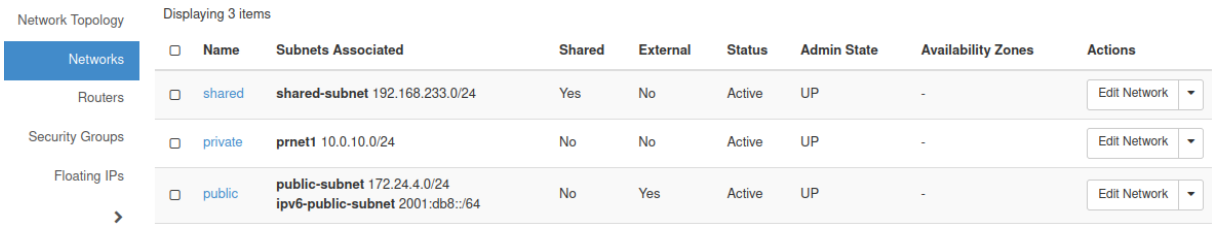



**Εικόνα 4.7:** Δημιουργία Image στο OpenStack

Προτού δημιουργήσουμε το πρώτο instance, μπορούμε να δημιουργήσουμε ένα private subnet (σε περίπτωση που δεν θέλουμε να χρησιμοποιήσουμε το default το οποίο δημιουργείται με την εγκατάσταση του OpenStack) επιλέγοντας το Network – Networks – Create Network στο dashboard. Στο παράθυρο που θα εμφανιστεί επιλέγουμε το όνομα του δικτύου (private στην προκειμένη περίπτωση), στο subnet tab δίνουμε όνομα στο subnet (prnet1), επιλέγουμε το επιθυμητό subnet (10.0.10.0/24) και το Gateway IP (10.0.10.1). Στο Subnet Details tab μπορούμε να καθορίσουμε τυχόν DHCP pools ή DNS Name Servers εάν επιθυμούμε. Αφού συμπληρώσουμε τις απαιτούμενες πληροφορίες επιλέγουμε create και βλέπουμε ότι το Private subnet δημιουργήθηκε:

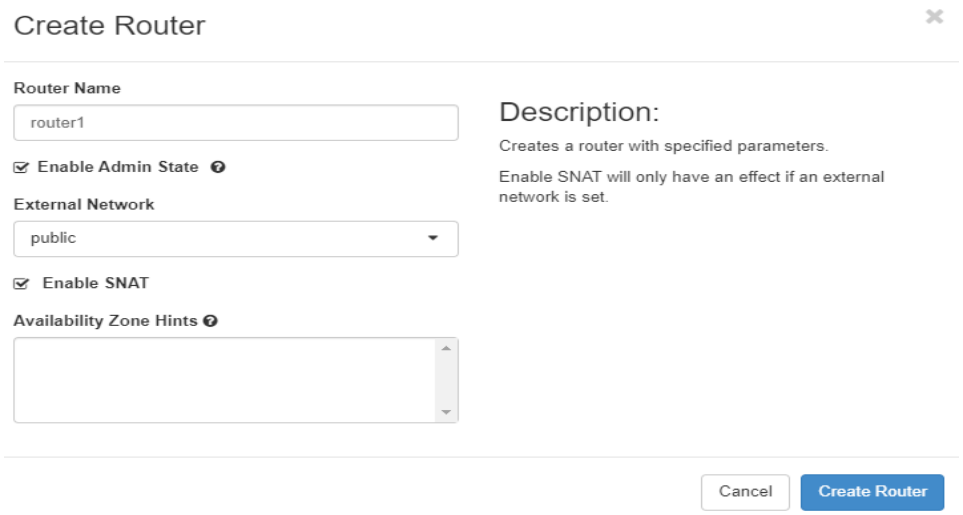



**Εικόνα 4.8:** Δημιουργία subnet στο private network



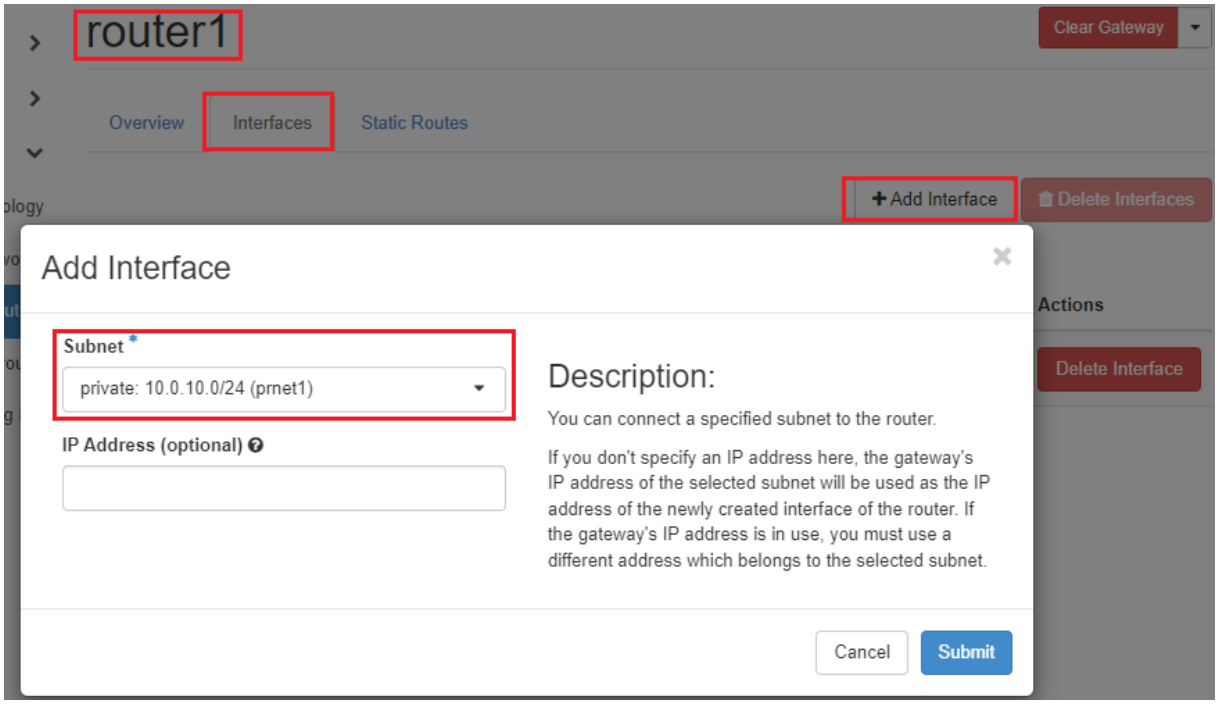

**Εικόνα 4.9:** Διαθέσιμα δίκτυα

Ακολούθως θα δημιουργήσουμε ένα Router ο οποίος θα χρησιμοποιηθεί στη συνέχεια για την δημιουργία Floating IP η οποία επιτρέπει την επικοινωνία του Instance με το internet. Για την δημιουργία του Router, επιλέγουμε στο OpenStack Network – Routers. Στο νέο παράθυρο δίνουμε την ονομασία που επιθυμούμε (router1) και επιλέγουμε public στο External Network:



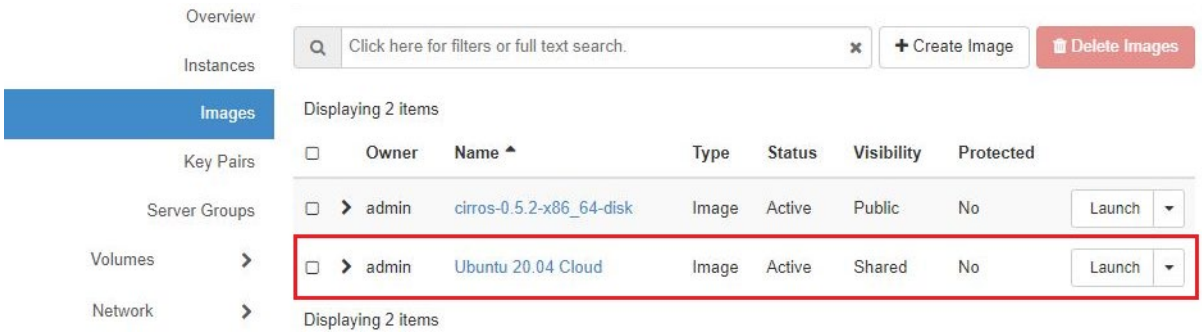

**Εικόνα 4.10:** Δημιουργία Router

Στη συνέχεια επιλέγουμε τον router που δημιουργήθηκε, και επιλέγουμε Interfaces – Add Interface. Στο παράθυρο που θα εμφανιστεί επιλέγουμε το Subnet που δημιουργήσαμε νωρίτερα (prnet1) και το IP Address μπορεί να μείνει κενό.



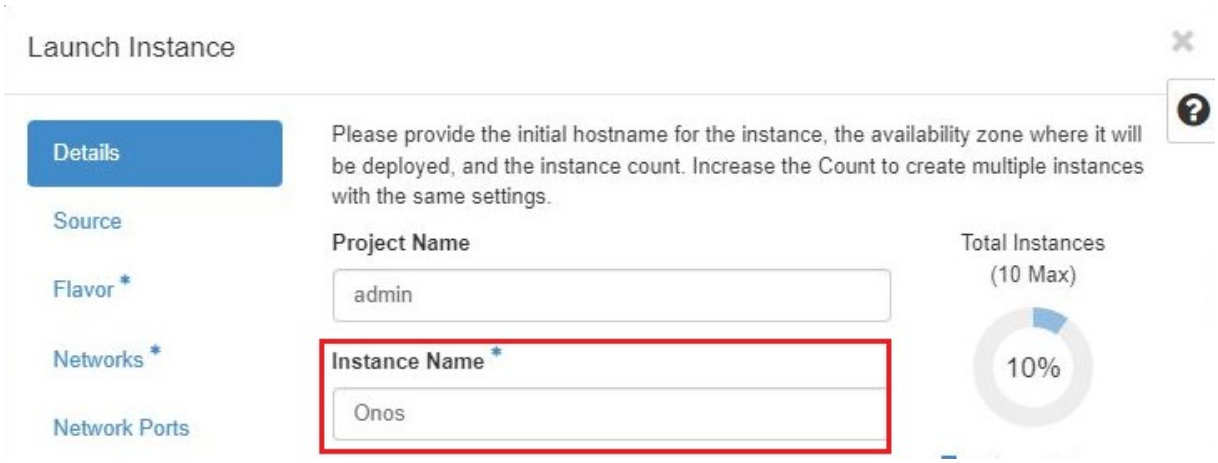

**Εικόνα 4.11:** Δημιουργία Interface στον Router

Αφού δημιουργήθηκαν τα δίκτυα και το image, είμαστε έτοιμοι να δημιουργήσουμε το πρώτο instance (VM) χρησιμοποιώντας το Ubuntu Cloud Image επιλέγοντας Compute – Images – Launch:

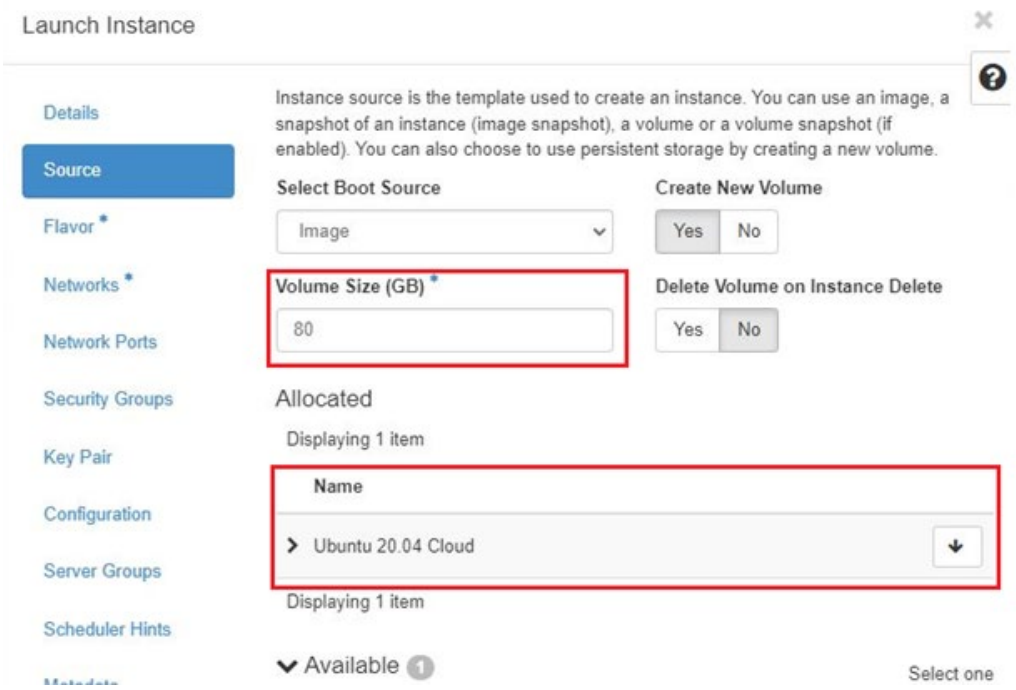



**Εικόνα 4.12:** Εκκίνηση image για την δημιουργία του πρώτου instance

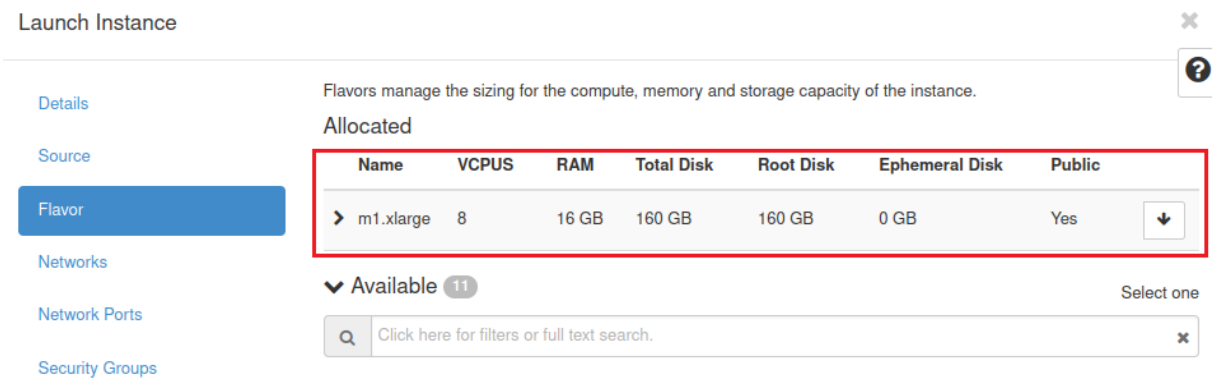
Στο παράθυρο που θα εμφανιστεί συμπληρώνουμε στα Details το όνομα του instance, και στο Source επιλέγουμε το Volume Size. Στο Flavor μπορούμε να επιλέξουμε τον συνδυασμό του compute, memory και storage capacity που επιθυμούμε αναλόγως των αναγκών του συστήματος αλλά και των διαθέσιμων πόρων του συστήματος που είναι εγκαταστημένο το OpenStack (στην προκειμένη εργασία δόθηκαν 40GB RAM, 12CPU και 500GB volume disk στον VMware Guest). Για το συγκεκριμένο instance το οποίο θα χρησιμοποιηθεί για την υλοποίηση του SDN Controller (Onos) επιλέχθηκε το flavor m1.xlarge το οποίο προσφέρει 8VCPUS και 16GB RAM. Ακολούθως στο Networks επιλέγουμε το private network που δημιουργήσαμε νωρίτερα και τέλος στο Key Pair δημιουργούμε τα κρυπτογραφικά κλειδιά που τα οποία θα χρησιμοποιηθούν για SSH authentication (το private key πρέπει να αποθηκευτεί καθώς θα χρησιμοποιηθεί αργότερα για αυθεντικοποίηση). Τα Security Groups θα δημιουργηθούν αργότερα. Πιο κάτω φαίνεται αναλυτικά η διαδικασία δημιουργίας ενός instance:



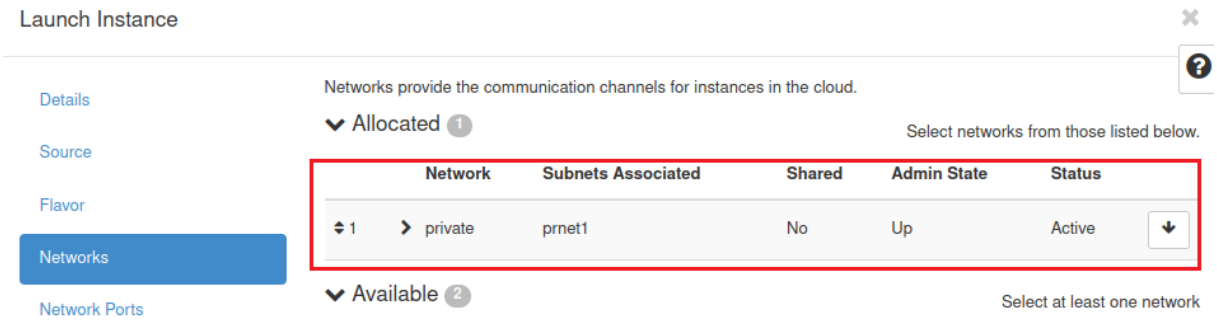

**Εικόνα 4.13:** Δημιουργία instance – Details



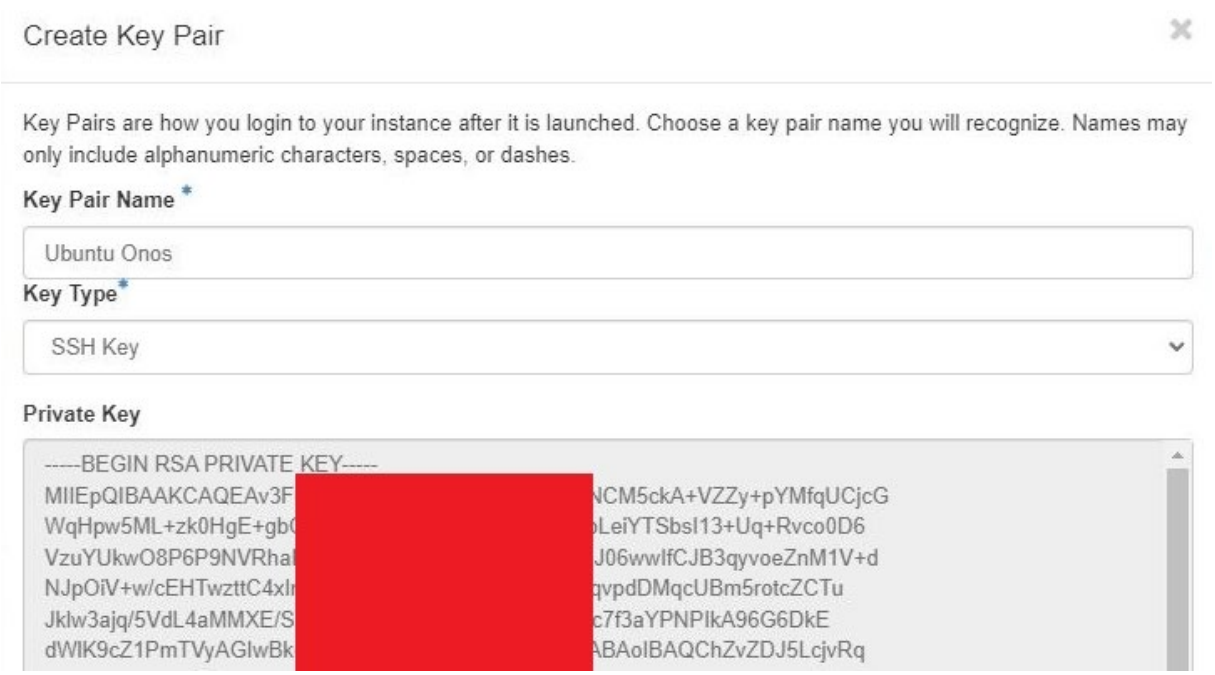

**Εικόνα 4.14:** Δημιουργία instance – Source




**Εικόνα 4.15:** Δημιουργία instance – Flavor



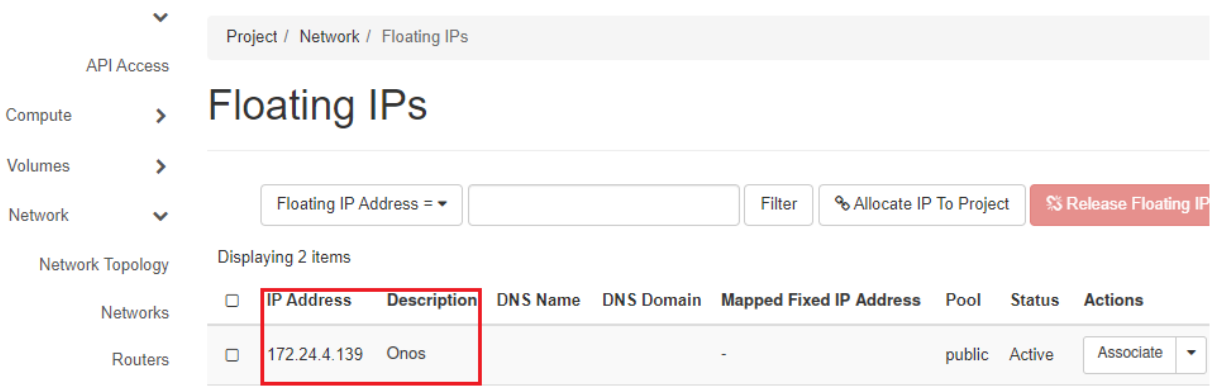

**Εικόνα 4.16:** Δημιουργία instance – Networks




**Εικόνα 4.17:** Δημιουργία instance – SSH Key

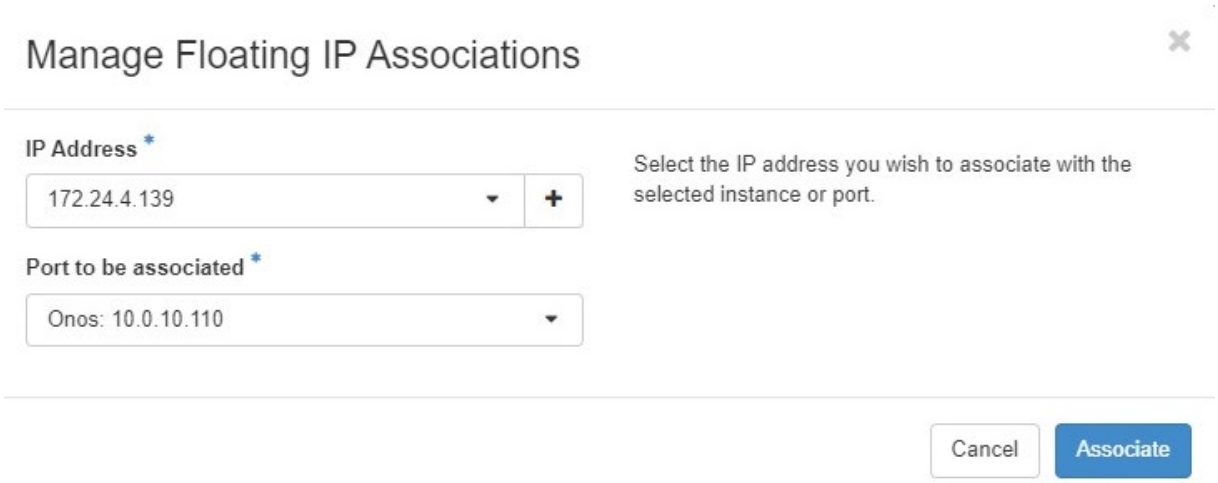
Αφού ολοκληρωθεί η δημιουργία του instance με επιτυχία, θα δημιουργήσουμε και θα ενεργοποιήσουμε το Floating IP το οποίο ουσιαστικά δημιουργεί virtual network interface στο instance και το συνδέει με το network interface του host μέσω του Router που δημιουργήθηκε νωρίτερα το οποίο επιτρέπει στο instance την επικοινωνία με το internet. Η λειτουργία του Floating IP επομένως μπορεί να παρομοιαστεί σαν το Public IP του κάθε instance.

Για την δημιουργία Floating IP στο OpenStack, επιλέγουμε Network – Floating IPs – Allocate IP To Project. Στο Pool είναι προεπιλεγμένο το public, ενώ στο Description μπορούμε να προσθέσουμε το όνομα του instance το οποίο θα χρησιμοποιήσει το Floating IP. Ακολούθως επιλέγουμε Allocate IP και βλέπουμε ότι δημιουργείται ένα IP στο subnet 172.24.4.0/24:




**Εικόνα 4.18:** Δημιουργία Floating IP στο OpenStack

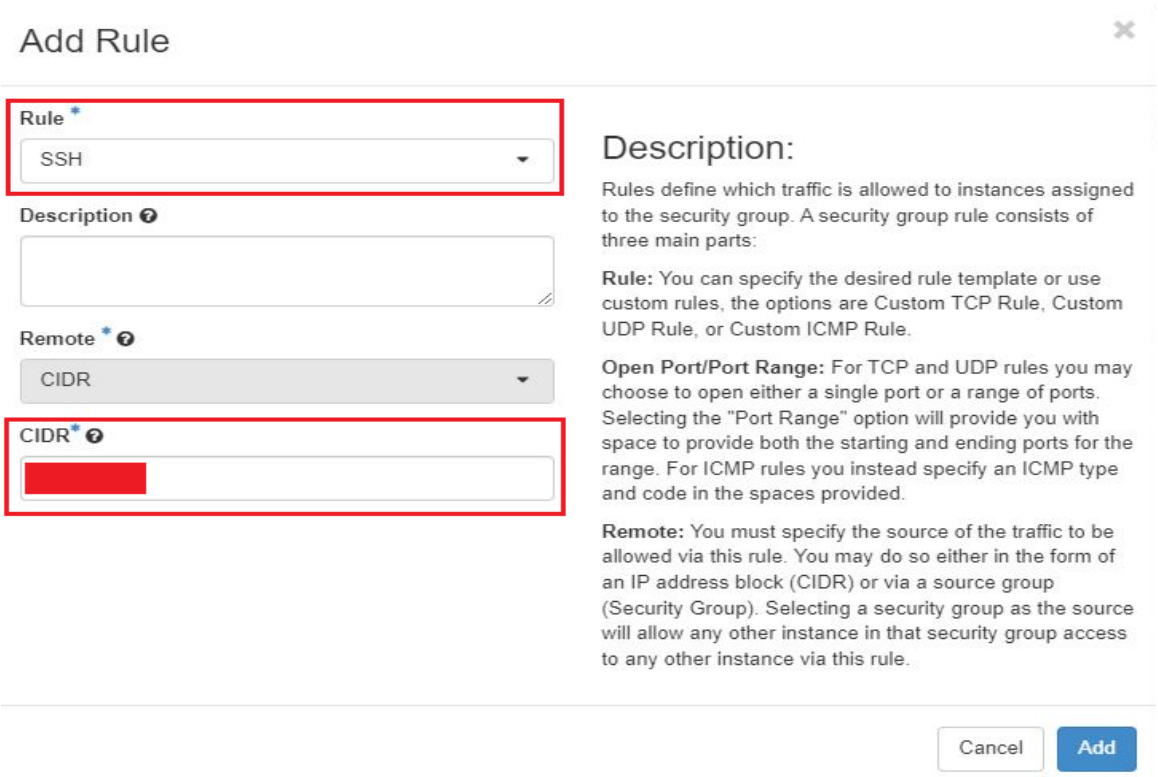
Στη συνέχεια επιστρέφουμε στο Compute – Instances και στο Actions επιλέγουμε Associate Floating IP. Στο παράθυρο που θα εμφανιστεί επιλέγουμε το IP που δημιουργήθηκε ενώ στο Port association επιλέγουμε το private IP του instance.




**Εικόνα 4.19:** Floating IP Association στο OpenStack

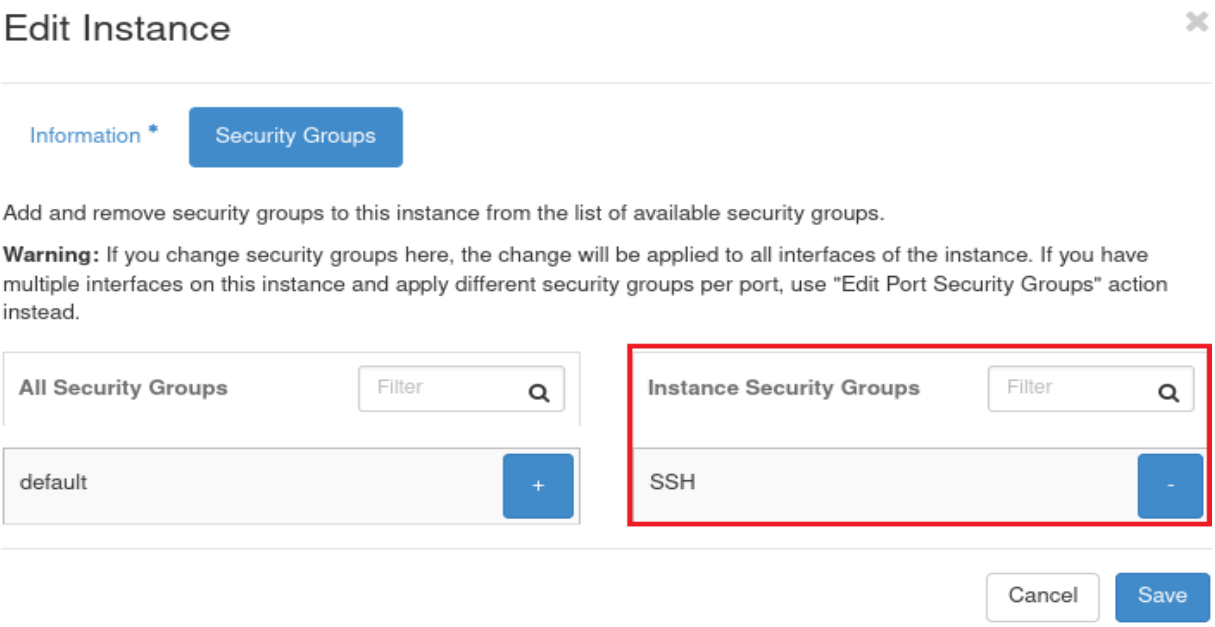
Το τελικό στάδιο που χρειάζεται να πραγματοποιήσουμε είναι η δημιουργία Security Groups τα οποία θα περιέχουν firewall rules τα οποία μπορούν να δοθούν στο κάθε instance. Αρχικά θα δημιουργηθεί ένα rule το οποίο θα επιτρέπει IPv4 TCP συνδέσεις στο port 22 το οποίο θα χρησιμοποιηθεί για SSH. Αργότερα όπως θα δούμε θα πρέπει να δημιουργηθούν τουλάχιστον ακόμη δύο rules.

Για την δημιουργία των Security Groups, επιλέγουμε στο OpenStack Network – Security Groups. (Βλέπουμε ότι υπάρχει το default Security Group το οποίο επιτρέπει όλο το traffic προς όλα τα ports τόσο για IPv4 όσο και για IPv6 το οποίο θα πρέπει να αφαιρεθεί από το Instance που δημιουργήσαμε). Στη συνέχεια επιλέγουμε Create Security Group δίνουμε για όνομα το SSH και επιλέγουμε Create Security Group. Με την δημιουργία του κάθε Security Group δημιουργούνται δύο Egress rules (IPv4 και IPv6) τα οποία επιτρέπουν όλα τα πρωτόκολλα για όλα τα IPs (εάν το επιθυμούμε μπορούμε να περιορίσουμε). Για την δημιουργία νέου rule επιλέγουμε Add Rule και στο Rule μπορούμε να επιλέξουμε SSH (είναι προκαθορισμένο το Ingress Direction στο Port 22) και ακολούθως επιλέγουμε Add (σε πραγματικό περιβάλλον δίνεται και η επιλογή να περιορίσουμε τα remote IPs που επιτρέπονται).



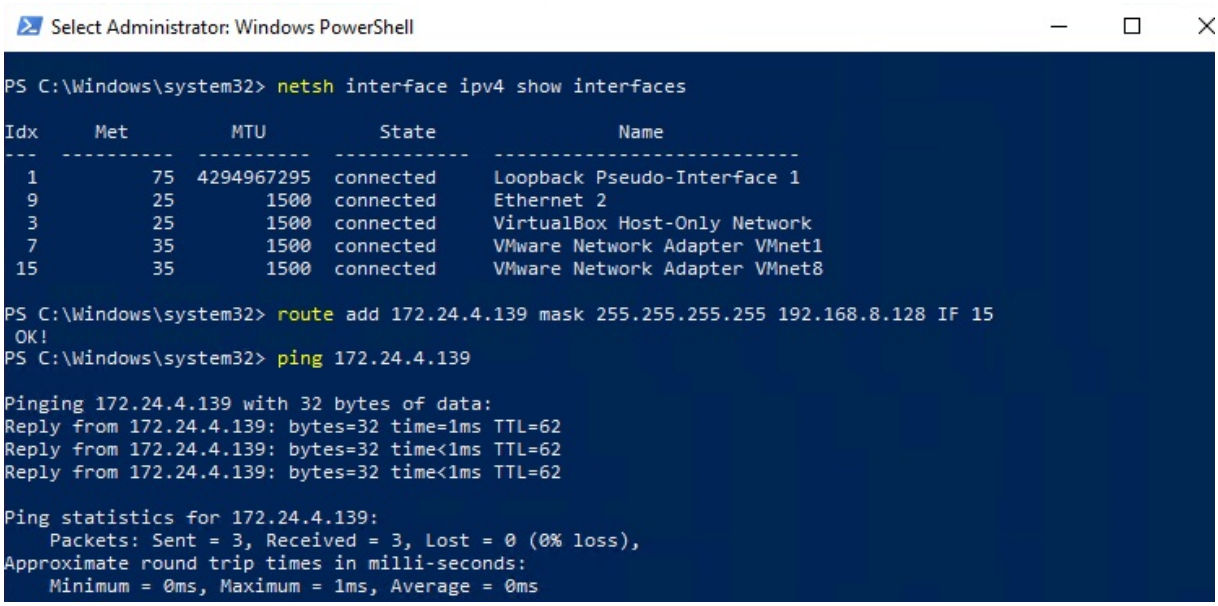

**Εικόνα 4.20:** OpenStack – Δημιουργία Firewall rule

Αφού δημιουργηθεί το Security Group μαζί με τα rules, το προσθέτουμε στο instance επιλέγοντας Compute – Instances και στο Actions του instance επιλέγουμε από το drop down menu το Edit Security Groups. Στο νέο παράθυρο επιλέγουμε το Security Group που δημιουργήσαμε (SSH στην προκειμένη περίπτωση) και αφαιρούμε το default group

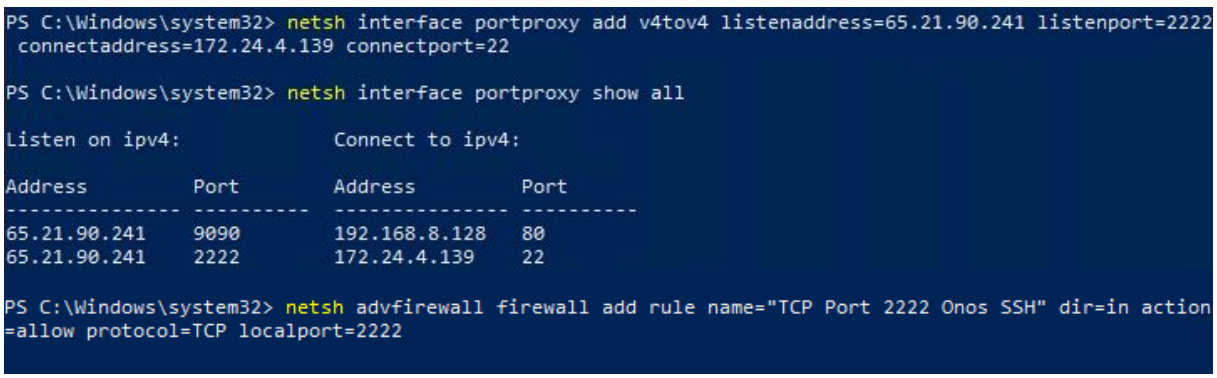



**Εικόνα 4.21:** Πρόσθεση Security Group στο instance

Για την δημιουργία απομακρυσμένης σύνδεσης στο Instance θα χρησιμοποιηθεί το “public” IP address (172.24.4.139). Έτσι θα πρέπει να δημιουργηθούν δύο κανόνες στο Host system. Ο πρώτος κανόνας είναι η δημιουργία route στο interface του Host προς το Instance στο OpenStack μέσω του VMware και ο δεύτερος κανόνας είναι ένα port forwarding από τον Host (θα χρησιμοποιηθεί το port 2222 του οποίου η πρόσβαση θα επιτραπεί μέσω νέου firewall rule) στο port 22 του Instance:




**Εικόνα 4.22:** Δημιουργία Route από τον Host στο Instance




**Εικόνα 4.23:** Δημιουργία Port forwarding από τον Host:2222 στο Instance:22

```
PS C:\Users\DESKTOP PC> Test-NetConnection -Port 2222 65.21.90.241


ComputerName     : 65.21.90.241
RemoteAddress    : 65.21.90.241
RemotePort       : 2222
InterfaceAlias   : Ethernet
SourceAddress    : 192.168.2.20
TcpTestSucceeded : True
```

**Εικόνα 4.24:** Επιβεβαίωση απομακρυσμένης σύνδεσης

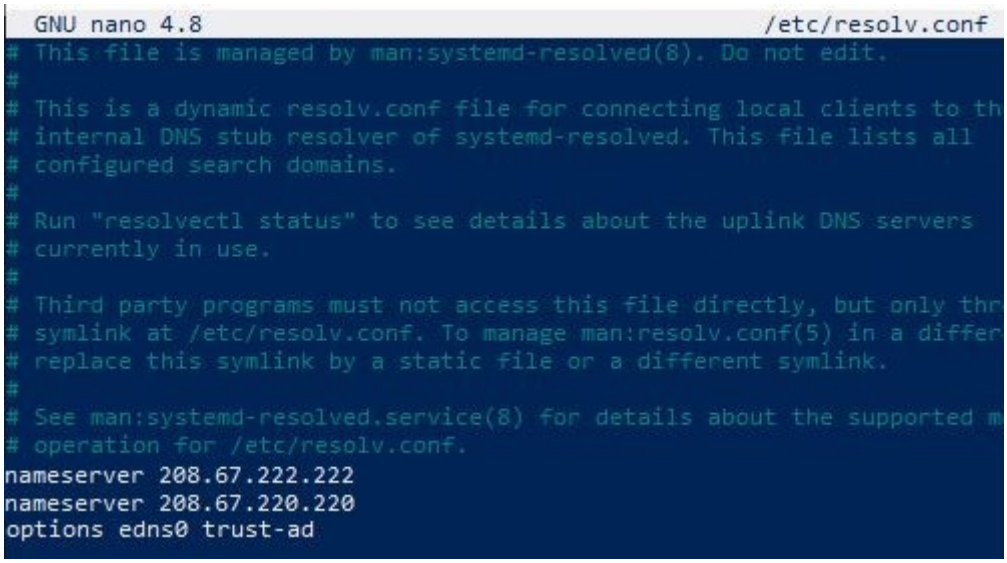
Ποιο κάτω φαίνεται η διαδικασία πρόσβασης στο instance τόσο από τον Ubuntu Host, τόσο και από απομακρυσμένη τοποθεσία. Είναι καλό να σημειωθεί ότι μετά την πρώτη πρόσβαση στο instance μπορεί να αποθηκευτεί το public key του ssh client στο αρχείο /etc/.ssh/known_hosts έτσι ώστε η αυθεντικοποίηση να γίνεται με το public key του client και όχι με το private key του host. Επίσης θα χρειαστεί και η προσθήκη των Nameservers στο Instance (η προσθήκη γίνεται στο αρχείο /etc/resolv.conf προσωρινά ενώ για μόνιμη λύση μπορεί να χρησιμοποιηθεί το εργαλείο resolvconf).

```
ubuntu@onos: ~
george@ubuntu:~$ chmod 0600 onos.pem
george@ubuntu:~$ ssh-add onos.pem
Identity added: onos.pem (onos.pem)
george@ubuntu:~$ ssh ubuntu@172.24.4.139
The authenticity of host '172.24.4.139 (172.24.4.139)' can't be established.
ECDSA key fingerprint is SHA256:uLz4BzcPnu/9yrmPS9cPclSM8tnn8AguMjHdHGECix0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.24.4.139' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-94-generic x86_64)
```

**Εικόνα 4.25:** SSH από τον Ubuntu Host

```
PS C:\Users\DESKTOP PC> ssh -i $key ubuntu@65.21.90.241 -p 2222
The authenticity of host '[65.21.90.241]:2222 ([65.21.90.241]:2222)' can't be established.
ECDSA key fingerprint is SHA256:uLz4BzcPnu/9yrmPS9cPclSM8tnn8AguMjHdHGECix0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[65.21.90.241]:2222' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-94-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sun Jan 16 19:55:57 UTC 2022

  System load:  0.43              Processes:             159
  Usage of /:   0.8% of 154.90GB  Users logged in:       0
  Memory usage: 1%                IPv4 address for ens3: 10.0.10.110
  Swap usage:   0%


1 update can be applied immediately.
To see these additional updates run: apt list --upgradable
```

**Εικόνα 4.26:** SSH από remote client



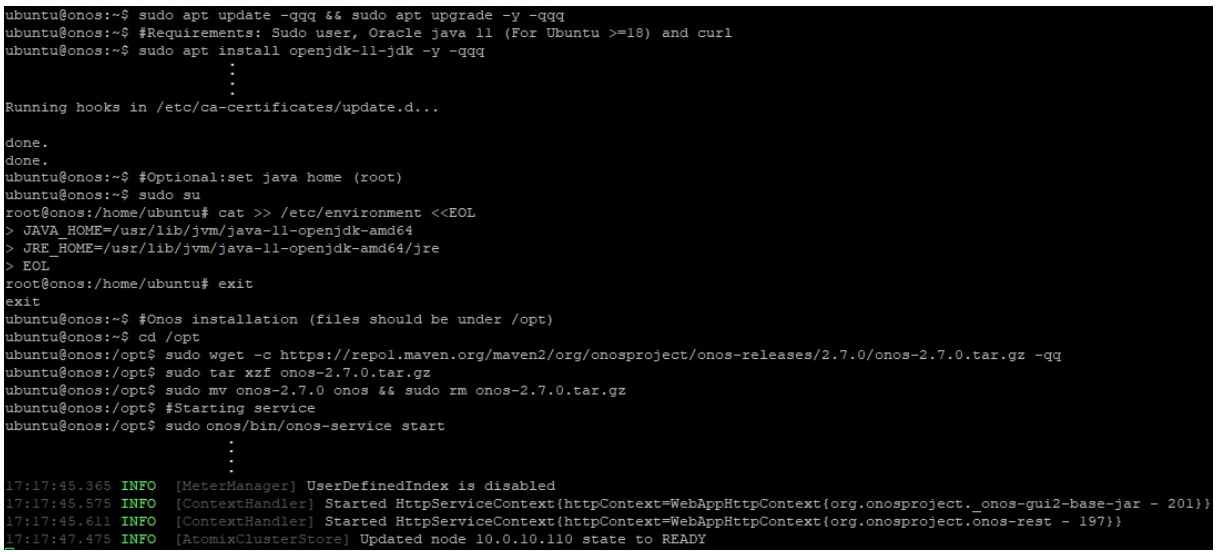

**Εικόνα 4.27:** Προσθήκη προσωρινών Nameservers στο Instance

### 4.1.3 Εγκατάσταση του ONOS

Το Open Network Operating System (ONOS) είναι μια πλατφόρμα ανοιχτού κώδικα η οποία αποτελείται από ένα σύνολο εφαρμογών οι οποίες δημιουργούν ένα SDN Controller που επιτρέπει το κτίσιμο των SDN/NFV (Network Function Virtualization) [15].

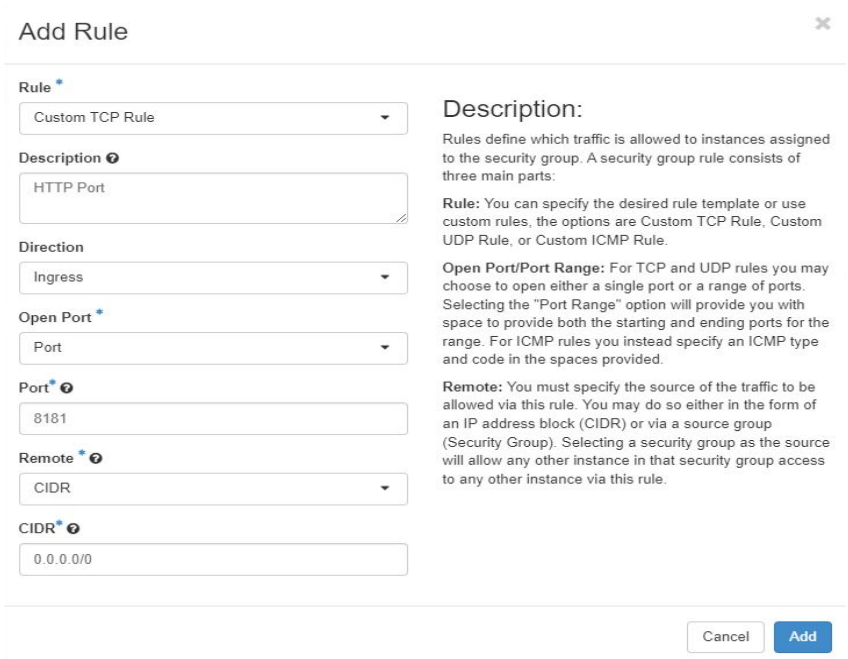
Για την εγκατάσταση του Onos σε Ubuntu 20.04 χρειάζεται ένας user με δικαιώματα sudo, το Oracle java 11 και το curl.

Η εγκατάσταση του Oracle java 11 γίνεται με την εντολή *sudo apt install openjdk-11-jdk* και προαιρετικά καθορίζεται στο environment το Java Home (/usr/lib/jvm/). Ακολούθως γίνεται η λήψη (εντολή wget) του αρχείου onos-2.7.0.tar.gz από τον σύνδεσμο https://repo1.maven.org/maven2/org/onosproject/onos-releases/2.7.0/onos-2.7.0.tar.gz και αφού γίνει extract μετακινείται στο directory /opt/onos/. Η εκκίνηση του γίνεται με την εντολή *sudo /opt/onos/bin/onos-service start*.




**Εικόνα 4.28**: Εγκατάσταση και εκκίνηση του ONOS σε Ubuntu 20.04

Αφού γίνει κανονικά η εκκίνηση του ONOS, το Web GUI λειτουργεί στο port 8181. Για την πρόσβαση στο Web GUI θα χρειαστούν δύο βήματα: Την δημιουργία ενός firewall rule στο OpenStack για την πρόσβαση στο port 8181 καθώς και την δημιουργία port forwarding από τον server στο "public" IP και port 8181 του Instance και ασφαλώς, ένα επιπλέον firewall rule που να επιτρέπει την πρόσβαση στο port 8181 του server. Οι διαδικασίες αυτές είναι οι ίδιες με αυτές που έγιναν για το remote SSH.

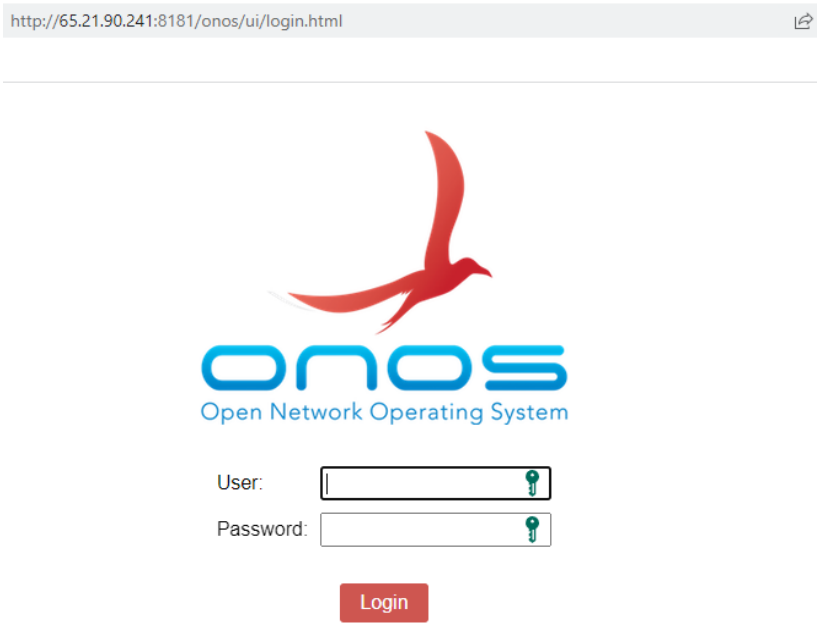



**Εικόνα 4.29**: Δημιουργία Security Group και rule για πρόσβαση στο port 8181

```
PS C:\Windows\system32> netsh interface portproxy add v4tov4 listenaddress=65.21.90.241 listenport=8181
 connectaddress=172.24.4.139 connectport=8181

PS C:\Windows\system32> New-NetFirewallRule -DisplayName 'Allow Onos 8181' -Direction Inbound -Action
Allow -Protocol TCP -LocalPort 8181
```

**Εικόνα 4.30:** Δημιουργία port forwarding και firewall rule στον Server

Αφού γίνουν τα πιο πάνω βήματα, η πρόσβαση στο Web GUI βρίσκεται στο http://*ipaddress*:8181/onos/ui/login.html. onos:rocks είναι ο default user και password.

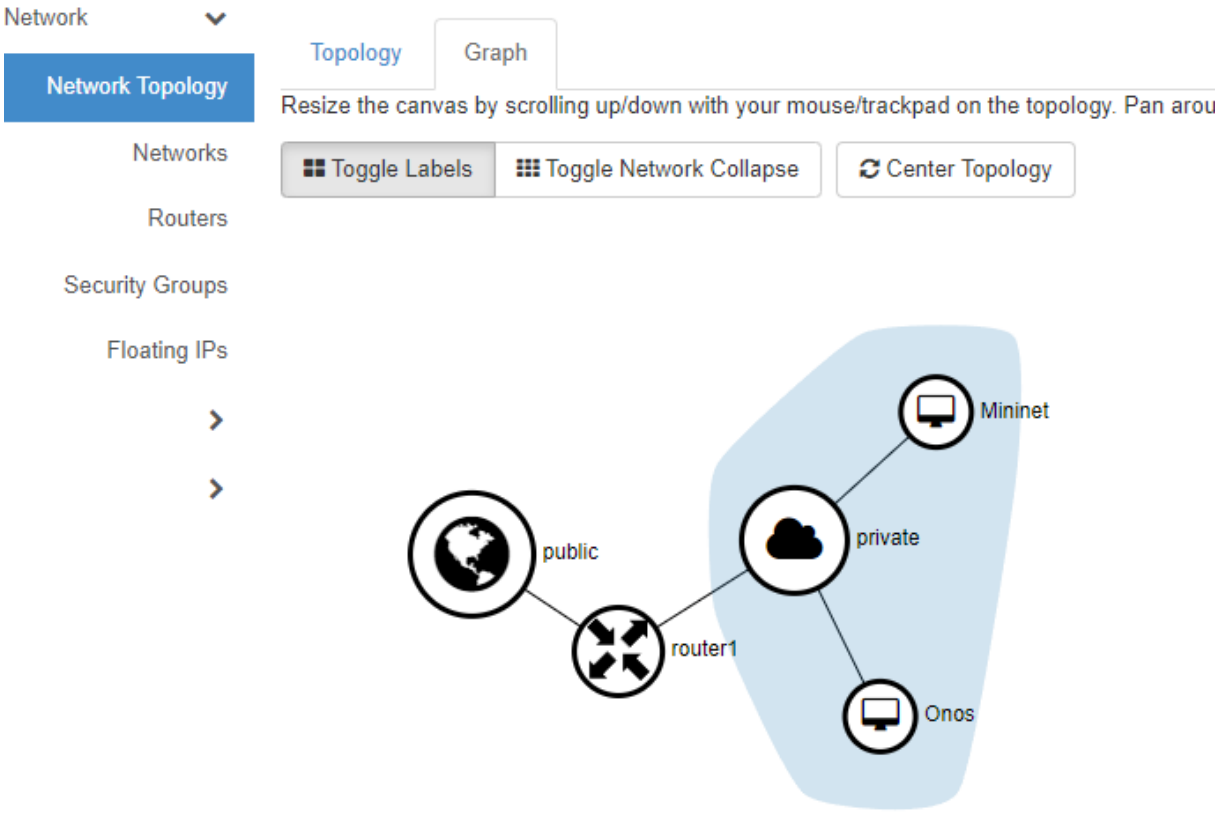



**Εικόνα 4.31**: ONOS Web GUI

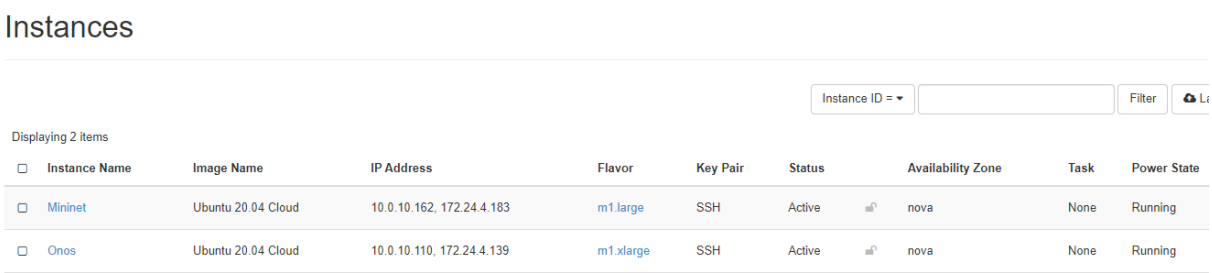
### 4.1.4 Εγκατάσταση του Mininet

Για την δημιουργία του ιδεατού δικτύου επιλέχθηκε το εργαλείο Mininet το οποίο μας δίνει την δυνατότητα να δημιουργήσουμε εύκολα μεγάλα ιδεατά δίκτυα ενώ προσφέρει και την δυνατότητα χρήσης του πρωτοκόλλου OpenFlow το οποίο θα χρησιμοποιηθεί για την επικοινωνία με τον SDN Controller.

Αρχικά δημιουργήθηκε νέο instance στο OpenStack χρησιμοποιώντας το ίδιο image (Ubuntu 20.04 cloud) που χρησιμοποιήθηκε νωρίτερα και ακολουθώντας τα ίδια βήματα που ακολουθήθηκαν για την δημιουργία του πρώτου Instance. Όταν ακολουθηθεί σωστά η διαδικασία, μπορούμε να δούμε την τοπολογία στο OpenStack επιλέγοντας Network – Network Topology – Graph:

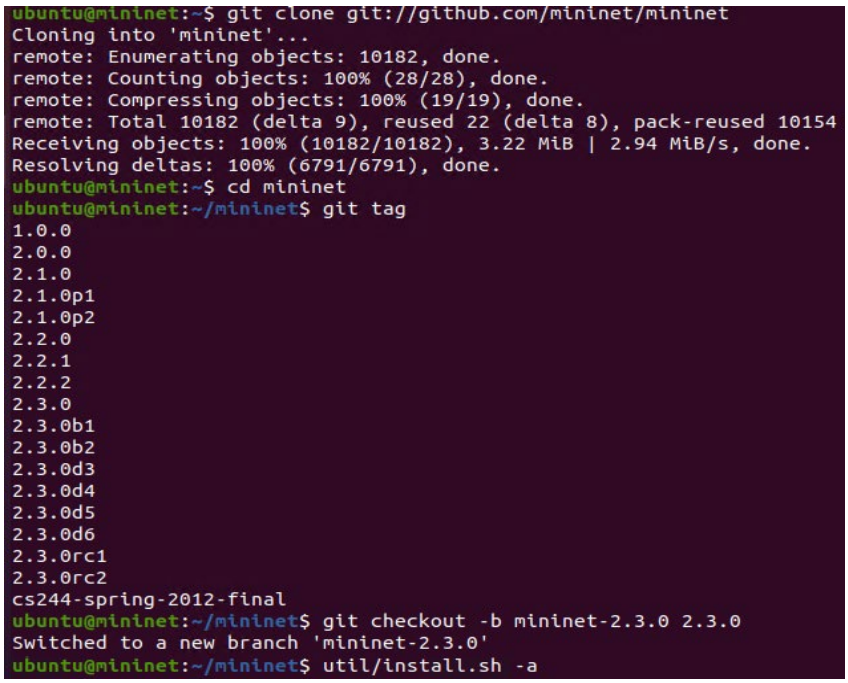



**Εικόνα 4.32:** Τοπολογία δικτύου στο OpenStack




**Εικόνα 4.33**: OpenStack Instances IP Addresses

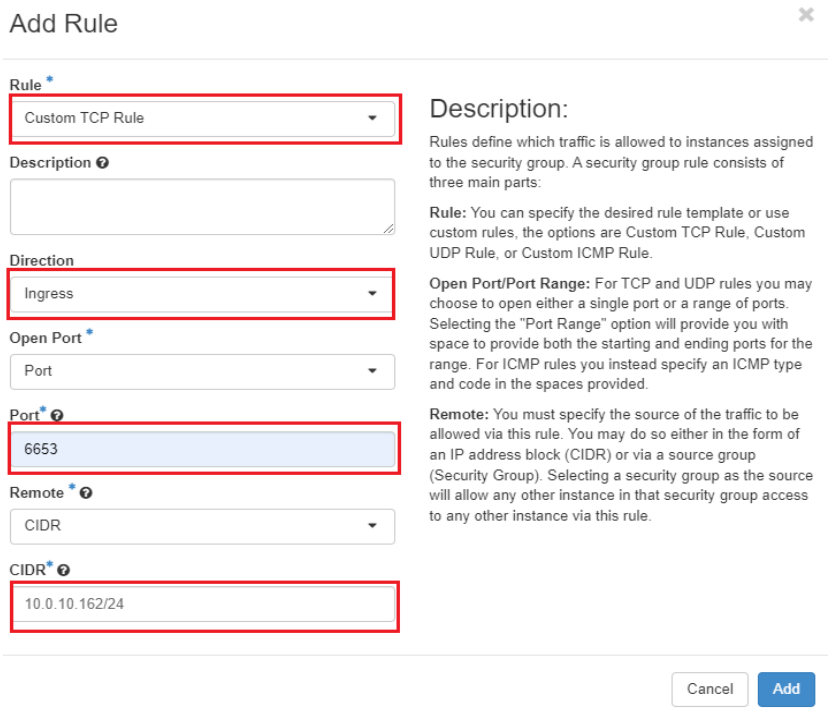
Για την εγκατάσταση του Mininet απαιτείται το εργαλείο git (*sudo apt install git*) με το οποίο γίνεται clone το Mininet. Αφού γίνει clone, με την εντολή *git tag* βλέπουμε τις διαθέσιμες εκδόσεις και με το *git checkout* επιλέγουμε την έκδοση που επιθυμούμε. Ακολούθως τρέχουμε το install.sh που βρίσκεται στο directory mininet/util/ δίνοντας την παράμετρο -a (all).




**Εικόνα 4.34**: Διαδικασία εγκατάστασης Mininet

Για επιβεβαίωση της σωστής εγκατάστασης και λειτουργίας του Mininet μπορεί να δοθεί η εντολή *sudo mn* η οποία δημιουργεί δύο hosts και ένα switch και με την εντολή *pingall* επιβεβαιώνεται η επικοινωνία των δύο hosts.

Ωστόσο για τον σκοπό της παρούσας εργασίας θα χρησιμοποιηθεί το ONOS ως remote controller ο οποίος θα είναι υπεύθυνος για το switching του SDN που θα δημιουργηθεί. Ο controller «ακούει» στο port 6653 (OpenFlow protocol) συνεπώς θα πρέπει να προστεθεί νέο rule στο OpenStack για το συγκεκριμένο port του instance. Σε αυτό το rule συνίσταται να χρησιμοποιηθεί το πεδίο CIDR καθώς περιορίζει την πρόσβαση στο συγκεκριμένο port μόνο από το subnet που θα δώσουμε (στην προκειμένη περίπτωση χρησιμοποιήθηκε αποκλειστικά το internal IP Address του Mininet Instance).



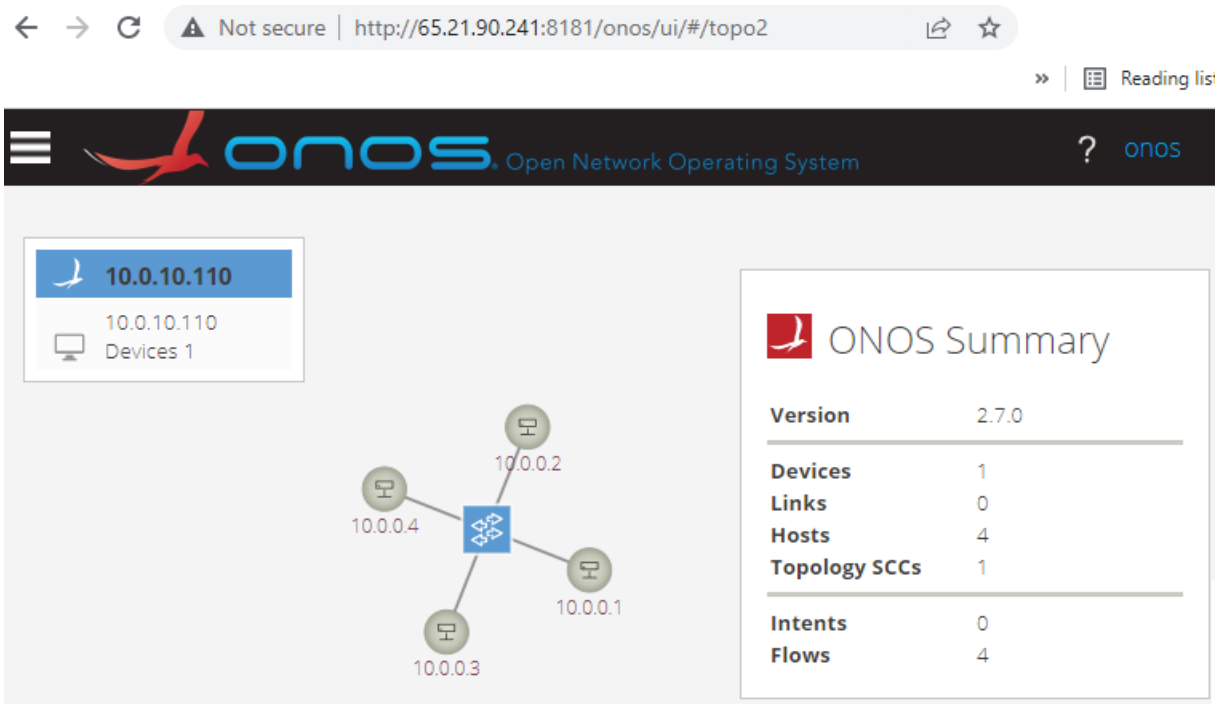

**Εικόνα 4.35**: Δημιουργία rule για πρόσβαση στο listening port του ONOS

Ακολούθως, στο Mininet Instance δίνουμε την εντολή *sudo mn --controller=remote,ip=10.0.10.110,port=6653 --topo tree,depth=1,fanout=4 --switch=ovs,protocols=OpenFlow13*. Με αυτό τον τρόπο καθορίζεται ο SDN Controller δίνοντας το remote internal IP address του ONOS καθώς και το port στο οποίο «ακούει» το πρωτόκολλο OpenFlow. Επίσης επιλέγουμε την τοπολογία tree καθώς χρειαζόμαστε μια σύνδεση για κάθε node, το depth καθορίζει τον αριθμό των switches που θα χρησιμοποιηθούν (ένα στην προκειμένη περίπτωση) και το fanout είναι ο αριθμός των hosts. Με την παράμετρο *switch*, καθορίζουμε το OVS και το πρωτόκολλο OpenFlow13. Αφού δημιουργηθεί η τοπολογία και τρέξουμε την εντολή pingall, αναμένουμε όλοι οι hosts να επικοινωνούν μεταξύ τους.

```
ubuntu@mininet:~$ sudo mn --controller=remote,ip=10.0.10.110,port=6653
 --topo tree,depth=1,fanout=4 --switch=ovs,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

**Εικόνα 4.36**: Δημιουργία SDN δικτύου στο Mininet

Μπορούμε τώρα να συνδεθούμε και στο Web GUI του ONOS και να δούμε την τοπολογία που δημιουργήθηκε.




**Εικόνα 4.37**: Τοπολογία SDN στο ONOS

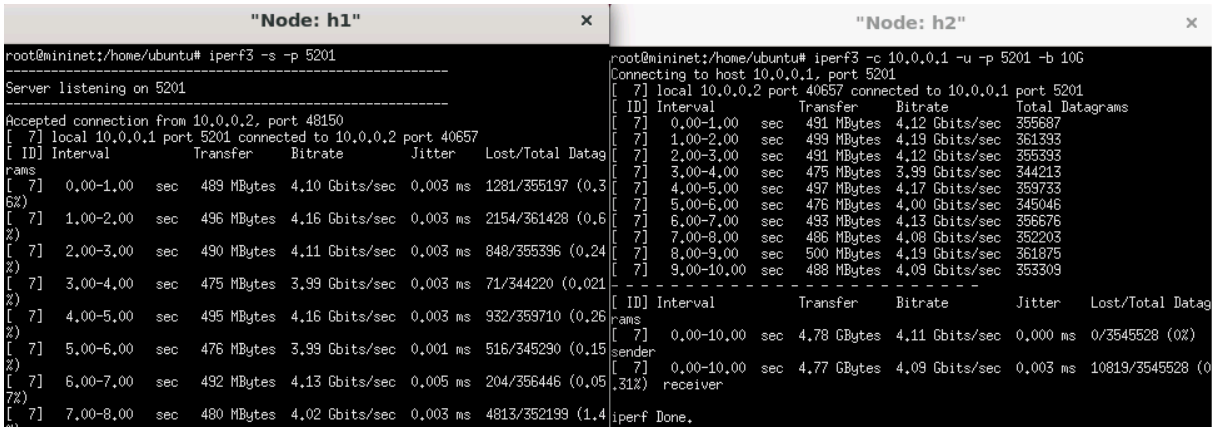
## 4.2 Προσομοίωση Επιθέσεων

Στην παρούσα εργασία μελετώνται οι επιθέσεις flooding σε SDN. Ποιο συγκεκριμένα, θα γίνει προσομοίωση των τριών βασικότερων επιθέσεων αυτής της κατηγορίας: ICMP flooding, UDP flooding και SYN flooding. Κοινό χαρακτηριστικό των τριών αυτών επιθέσεων είναι η δέσμευση όλων των διαθέσιμων πόρων ενός συστήματος στέλνοντας του μεγάλο όγκο δεδομένων τον οποίο δεν μπορεί να επεξεργαστεί με αποτέλεσμα να μην υπάρχει πλέον η δυνατότητα να εξυπηρετήσει τις απαιτούμενες διεργασίες του.

Για την προσομοίωση των επιθέσεων θα χρησιμοποιηθεί το εργαλείο hping3 το οποίο είναι ένα εργαλείο δικτύου το οποίο επιτρέπει να δημιουργούνται και να αποστέλλονται ειδικά διαμορφωμένα packets ή segments/datagrams σε Layer 3 και Layer 4 αντίστοιχα. Μπορούν επίσης να χρησιμοποιηθούν διαφορετικά πρωτόκολλα όπως ICMP, UDP και TCP [16].

Αφού γίνει η εγκατάσταση του hping3 (σε Ubuntu 20.04 γίνεται με την εντολή *sudo apt install hping3*) δημιουργούμε το δίκτυο στο Mininet με την εντολή που δώσαμε νωρίτερα προσθέτοντας

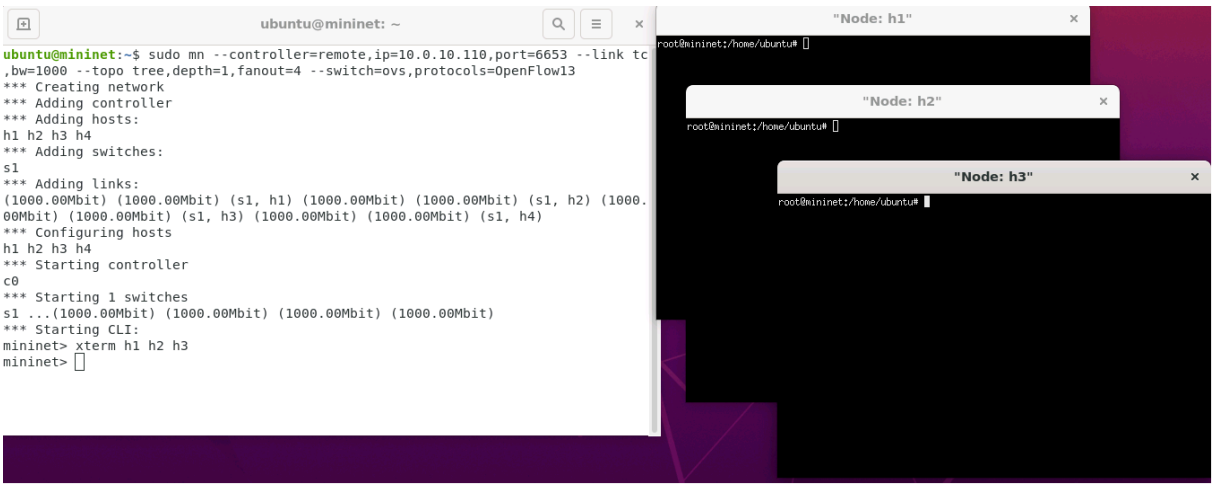
ωστόσο και την παράμετρο *--tc link,bw=1000* η οποία καθορίζει το bandwidth σε 1Gbs για κάθε link. Ο περιορισμός αυτός γίνεται για να μπορέσουμε να δούμε πιο εύκολα πως η επίθεση επηρεάζει την αποδοτικότητα του δικτύου. Χωρίς τον περιορισμό, το bandwidth φτάνει μέχρι τα 10Gbs και ο κάθε host στο Mininet χρησιμοποιεί μόνο ένα thread με αποτέλεσμα, κατά την προσομοίωση ενός flooding attack να μην υπάρχει η δυνατότητα κατάληψης όλων των διαθέσιμων πόρων. Ποιο κάτω φαίνεται το μέγιστο bandwidth που μπορεί να παραχθεί μεταξύ δύο host. Η μέτρηση έγινε με το εργαλείο iperf3 το οποίο χρησιμοποιείται, μεταξύ άλλων, για να μετρηθεί το μέγιστο Bandwidth σε IP δίκτυα [17]. Ο host1 αποτελεί τον iperf server και ο host2, τον Iperf client.




**Εικόνα 4.37**: Μέγιστο Bitrate που μπορεί να παραχθεί από τον κάθε host

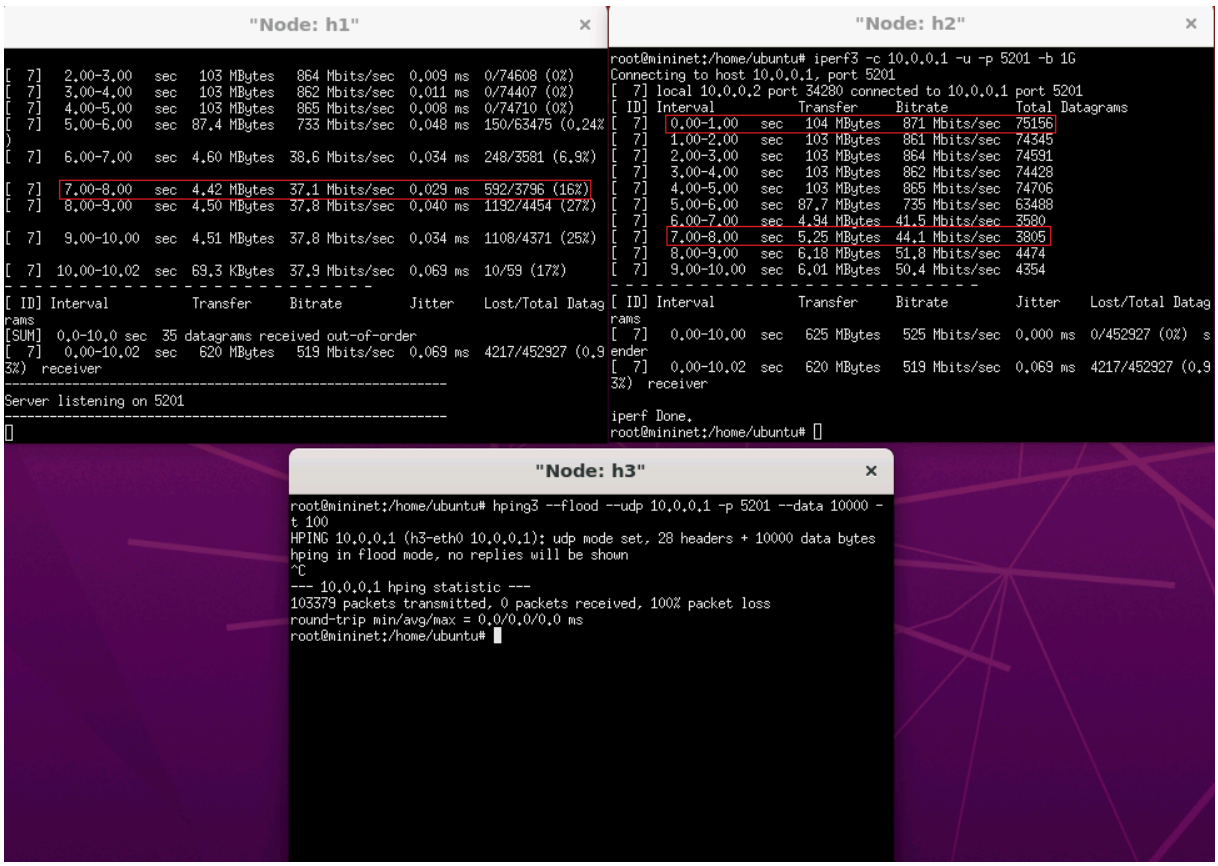
Περιορίζοντας το διαθέσιμο bandwidth του δικτύου σε 1Gbs, θα μπορέσουμε να δούμε τα αποτελέσματα του flooding attack, το οποίο πλέον θα έχει την δυνατότητα να χρησιμοποιήσει ολόκληρο το διαθέσιμο bandwidth.

Αφού δημιουργηθεί το δίκτυο στο Mininet, με την εντολή *xterm h1 h2 h3* ενεργοποιούνται τα τερματικά παράθυρα των hosts (Σε περίπτωση που δεν ενεργοποιούνται τα τερματικά θα πρέπει να δοθεί πρώτα η εντολή *xhost +si:localuser:root* και ακολούθως να δημιουργηθεί το ιδεατό δίκτυο).



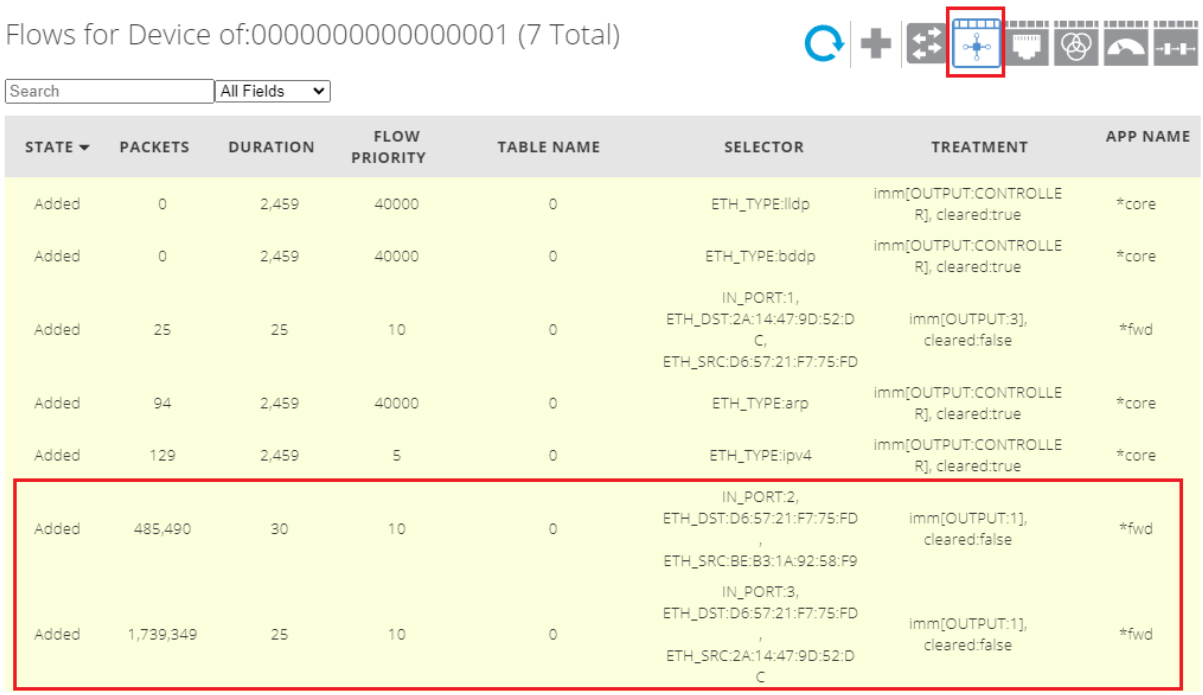

**Εικόνα 4.38:** Πρόσβαση στα τερματικά των Mininet hosts

Έχοντας αυτή την τοπολογία, ο host1 θα πάρει τον ρόλο του iperf server (*iperf3 –s –p 5201*), ο host2 τον ρόλο του iperf client (*iperf3 –c 10.0.0.1 –u –p 5201 –b 1G*) ενώ ο host3 τον ρόλο ενός compromised host στο δίκτυο, ο οποίος θα πραγματοποιήσει αρχικά το udp flooding attack χρησιμοποιώντας το εργαλείο hping3 (*hping3 --flood –udp 10.0.0.1 –p 5201 –data 10000 –t 100*). Παρατηρούμε με αυτό τον τρόπο ότι από το πέμπτο δευτερόλεπτο και έπειτα που ξεκίνησε το UDP flooding attack, το Bitrate από τον host2 στον host1 μειώθηκε από 871 Mbs (μέγιστη τιμή πριν από την επίθεση) σε 44,1 Mbs.



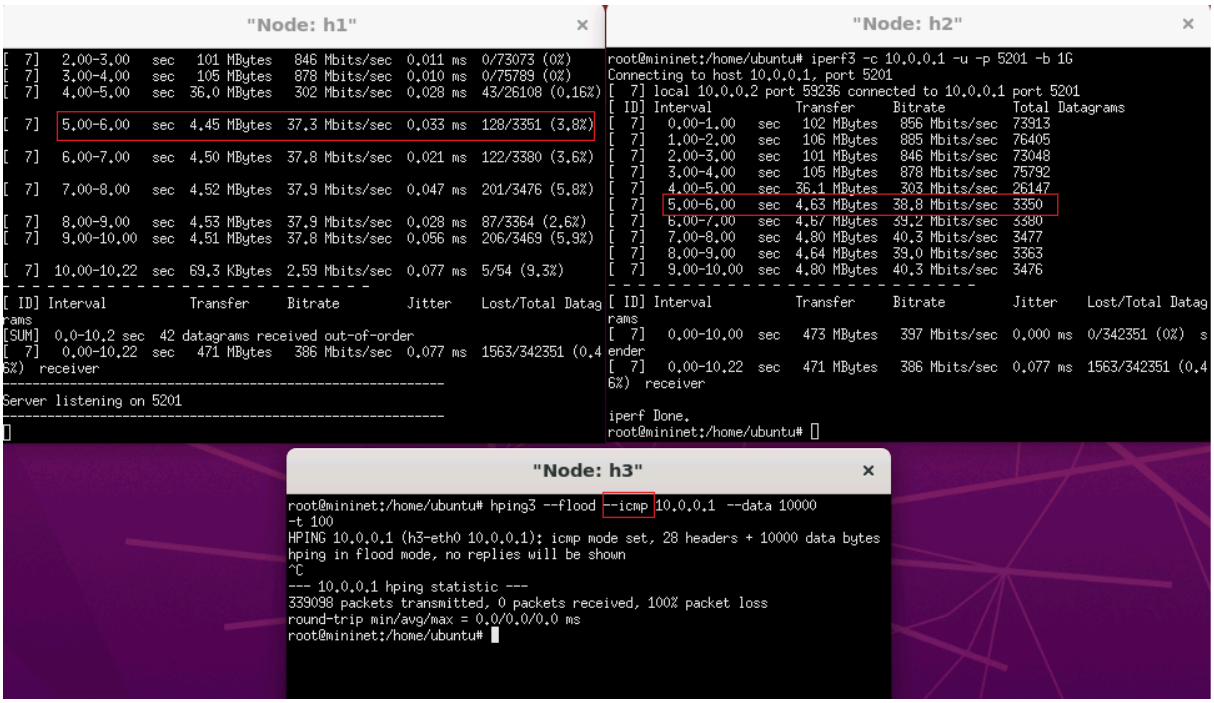

**Εικόνα 4.39:** Αποτελεσματικότητα επίθεσης UDP Flooding

Στο ONOS WebGUI μπορούμε επίσης να δούμε τα flows τα οποία δημιουργήθηκαν καθώς και τον αριθμό των πακέτων για κάθε flow επιλέγοντας το Show flow view for selected device.

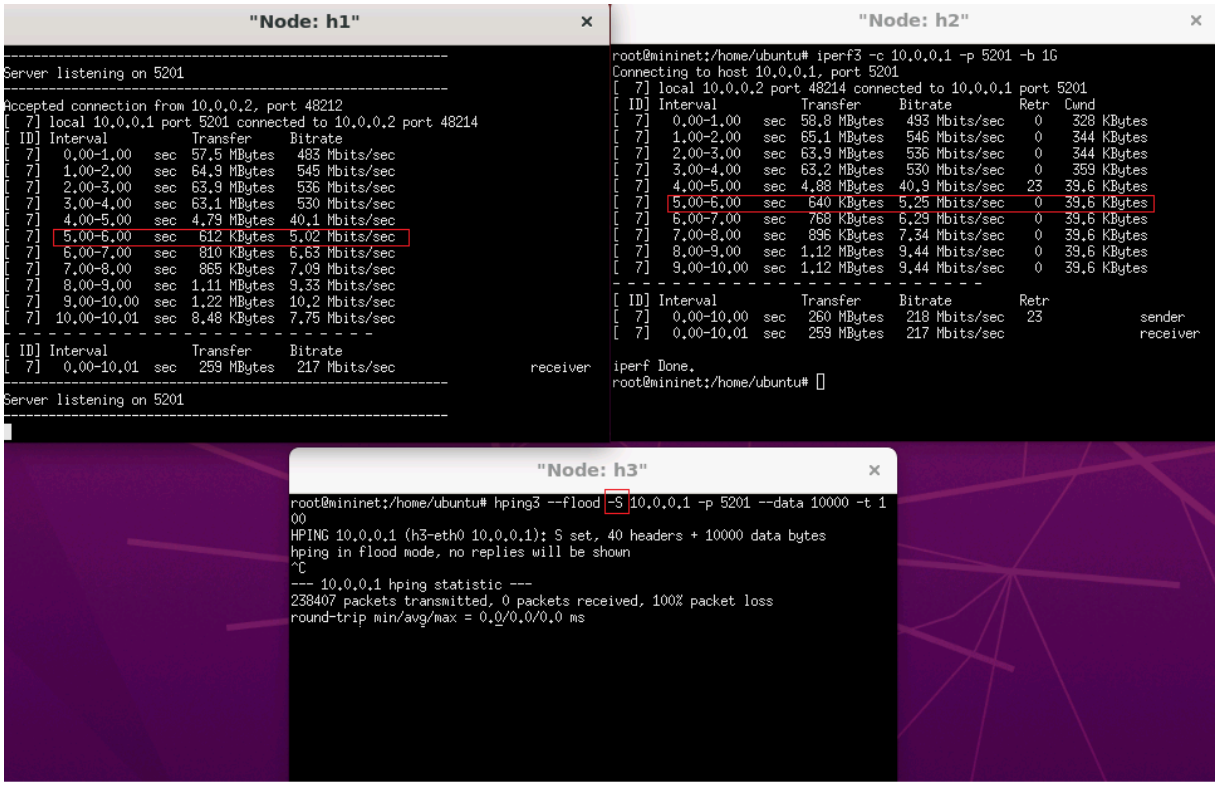



**Εικόνα 4.40:** ONOS WebGUI – Device flows

Παρόμοια αποτελέσματα έχουμε και σε περίπτωση ICMP flooding (αλλάζοντας την παράμετρο στο hping3 σε --icmp) καθώς και σε SYN flooding (αλλάζοντας την παράμετρο σε –S).




**Εικόνα 4.41:** ICMP Flooding Attack




**Εικόνα 4.42:** SYN Flooding Attack

## 4.3 Μετριασμός Επιθέσεων

Ο προτεινόμενος μετριασμός των επιθέσεων μπορεί να διαχωριστεί σε δύο επίπεδα. Το πρώτο επίπεδο είναι η παρακολούθηση και εξέταση όλου του traffic στο δίκτυο και η δημιουργία κανόνων οι οποίοι θα ενεργοποιούν ειδοποιήσεις σε περίπτωση παραβίασης τους. Το δεύτερο επίπεδο αποτελείται από την εξέταση των ειδοποιήσεων και τη δημιουργία νέων κανόνων ροής στον controller οι οποίοι θα αποτρέπουν την προώθηση συγκεκριμένων δεδομένων από τον source προς το destination.

### 4.3.1 Εγκατάσταση Snort3

Για την παρακολούθηση του traffic επιλέχθηκε το Snort3 το οποίο είναι ένα λογισμικό ανοικτού κώδικα το οποίο έχει την δυνατότητα να εκτελεί διεργασίες IDS/IPS. Το Snort3 χρησιμοποιεί μια σειρά προκαθορισμένων κανόνων οι οποίοι βοηθούν στον καθορισμό της κακόβουλης δραστηριότητας στο δίκτυο και χρησιμοποιώντας αυτούς τους κανόνες εντοπίζει τα πακέτα που ταιριάζουν με αυτούς και δημιουργεί τις ανάλογες ειδοποιήσεις. Υπάρχει επίσης η δυνατότητα, το Snort3 να εγκατασταθεί και να ρυθμιστεί με τέτοιο τρόπο που να σταματάει αυτά τα πακέτα. Οι κυριότερες χρήσεις του Snort3 είναι: Ανιχνευτής πακέτων όπως το tcpdump, ως καταγραφέας

(logger) πακέτων το οποίο είναι χρήσιμο για τον εντοπισμό σφαλμάτων της κυκλοφορίας δικτύου ή μπορεί να χρησιμοποιηθεί ως πλήρες σύστημα πρόληψης εισβολών στο δίκτυο [18]. Οι ειδοποιήσεις μπορούν να γίνονται σε json format κάτι το οποίο θα μας βοηθήσει να παίρνουμε την απαιτούμενη πληροφορία για την δημιουργία των κανόνων ροής στον controller.

Η εγκατάσταση του Snort3 έγινε σε Ubuntu 20.04 (στο ίδιο instance που δημιουργείται το ιδεατό δίκτυο με το Mininet) και χρησιμοποιήθηκε το εγχειρίδιο που παρέχει η ομάδα του Snort [19]. Για τον σκοπό της εργασίας δεν χρειάστηκε η εγκατάσταση του PulledPork το οποίο είναι ένα εργαλείο χρησιμοποιείται για την εγκατάσταση των ενημερωμένων rulesets από την ομάδα snort/talos. Οι κύριες ρυθμίσεις λειτουργίας του Snort3 γίνονται στο αρχείο /usr/local/etc/snort/snort.lua. Η πρώτη ρύθμιση που έγινε αφορά στην δημιουργία των ειδοποιήσεων σε json format όπου μπορούν να καθοριστούν και τα fields που θα περιλαμβάνονται ανάλογα με τις απαιτήσεις μας. Επίσης, απενεργοποιήθηκαν τα built-in rules έτσι ώστε να χρησιμοποιηθούν μόνο τα custom rules τα οποία θα δημιουργήσουμε.

```
alert_json =
{
    file = true,
    limit = 100,
    fields = 'seconds action class dir dst_addr dst_ap dst_port eth_dst eth_len \
    eth_src eth_type gid icmp_code icmp_id icmp_seq icmp_type iface ip_id ip_len msg \
    pkt_gen pkt_len pkt_num priority proto rev rule service sid src_addr src_ap src_port \
    target tcp_ack tcp_flags tcp_len tcp_seq tcp_win tos ttl udp_len vlan timestamp',
}
```

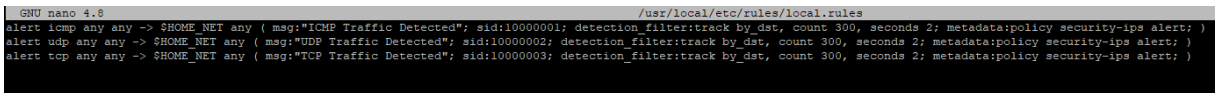
**Εικόνα 4.43:** Snort3 – Καθορισμός ειδοποιήσεων σε json format

```
ips =
{
    -- use this to enable decoder and inspector alerts
    --enable_builtin_rules = true,

    -- use include for rules files; be sure to set your path
    -- note that rules files can include other rules files
    -- (see also related path vars at the top of snort_defaults.lua)
    enable_builtin_rules = false,
    include = RULE_PATH .. "/local.rules",
    variables = default_variables
}
```

**Εικόνα 4.44:** Snort3 – Απενεργοποίηση built-in rules

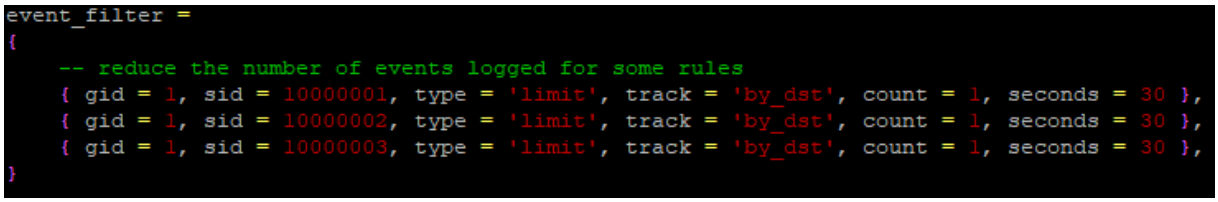
Τα custom rules θα δημιουργηθούν στο αρχείο /usr/local/etc/rules/local.rules (το οποίο περιλαμβάνεται και στην πιο πάνω ρύθμιση) το οποίο θα περιλαμβάνει τρείς κανόνες εντοπισμού αυξημένης δραστηριότητας πακέτων ICMP, UDP και TCP. Η μορφή των κανόνων μπορεί να

χωριστεί σε τρία σημεία. Το πρώτο είναι η οδηγία δημιουργία ειδοποίησης για συγκεκριμένο IP πρωτόκολλο (πχ alert icmp), το δεύτερο είναι το source IP και source Port (any any), το τρίτο αφορά στο destination IP και destination port (για destination IP χρησιμοποιήθηκε η μεταβλητή $HOME_NET η οποία καθορίζεται στο snort.lua, πχ 10.0.0.0/24) και το τέταρτο σημείο καθορίζει τις διάφορες παραμέτρους όπως το μήνυμα που θα εμφανιστεί, το sid το οποίο πρέπει να είναι μοναδικό για το κάθε rule καθώς και το detection filter το οποίο είναι σημαντικό να αποτυπωθεί σωστά για την σωστή λειτουργία του IDS. Για τις ανάγκες της εργασίας, το detection filter καθορίστηκε να γίνεται ο εντοπισμός της επίθεσης από τον όγκο πακέτων προς κάποιο destination, και το threshold σε 300 πακέτα ανά δύο δευτερόλεπτα.




**Εικόνα 4.45:** Snort3 – Δημιουργία custom rules

Επιπλέον, θα πρέπει να ρυθμιστεί και ο αριθμός των ειδοποιήσεων που θα δημιουργούνται ανά χρονικό διάστημα κάτι το οποίο θα αποτρέψει την δημιουργία και επεξεργασία εκατομμύριων ειδοποιήσεων. Στο Snort3 η ρύθμιση γίνεται στο τμήμα event_filter του αρχείου snort.lua. Για τον σκοπό της εργασίας επιλέχθηκε να δημιουργείται μόνο μία ειδοποίηση (type='limit' και count=1) κάθε 30 δευτερόλεπτα. Όπως θα δούμε και πιο κάτω, θα εξετάζεται κάθε 30 δευτερόλεπτα κατά πόσο η επίθεση σταμάτησε έτσι ώστε να επαναφέρεται η επικοινωνία.




**Εικόνα 4.46:** Snort3 – Δημιουργία event filter

Το τελευταίο βήμα είναι να τροποποιηθεί το Startup Script (/lib/system/system/snort3.service) το οποίο δημιουργήθηκε κατά την εγκατάσταση του Snort3. Η εντολή εκκίνησης του Snort3 περιλαμβάνει μεταξύ άλλων το path του snort binary, το path για το αρχείο snort.lua, το network interface από όπου θα παρακολουθείται η κίνηση στο δίκτυο (s1-eth4 όπως θα δούμε πιο κάτω κατά την ρύθμιση του Mininet) καθώς και το path που θα αποθηκεύονται τα logs (/var/log/snort).

```
[Unit]
Description=Snort3 NIDS Daemon
After=syslog.target network.target
[Service]
Type=simple
ExecStart=/usr/local/bin/snort -c /usr/local/etc/snort/snort.lua -s 65535 \
-k none -l /var/log/snort -D -u snort -g snort -i s1-eth4 -m 0x1b --create-pidfile
[Install]
WantedBy=multi-user.target
```

**Εικόνα 4.47:** Snort3 – Startup Script

Πριν από την εκκίνηση του Snort3, θα δημιουργήσουμε το ιδεατό δίκτυο στο Mininet και θα χρησιμοποιήσουμε το interface s1-eth4 ως mirror όλων των υπόλοιπων ports έτσι ώστε να παίρνουμε όλο το traffic από το switch. Αυτό επιτυγχάνεται αφού δημιουργήσουμε το δίκτυο (*sudo mn --controller=remote,ip=10.0.10.110,port=6653 --topo tree,depth=1,fanout=4 --switch=ovs,protocols=OpenFlow13*) στο τερματικό του s1 δίνεται η εντολή *ovs-vsctl del-port s1-eth4* για την διαγραφή του port για τον h4 και αντικατάσταση του σε mirror port όλων των υπόλοιπων ports του switch με την εντολή *ovs-vsctl add-port s1-eth4 -- --id=@p get port s1-eth4 -- --id=@m create mirror name=m0 select-all=true output-port=@p -- set bridge s1 mirrors=@m*


**Εικόνα 4.48**: Mininet – Δημιουργία mirror port

Αφού δημιουργηθεί το mirror port, μπορεί να γίνει η εκκίνηση του Snort3 με την εντολή *sudo service snort3 start* ενώ με την εντολή *sudo service snort3 status* μπορούμε να βεβαιωθούμε ότι το Snort3 και οι ρυθμίσεις του λειτουργούν κανονικά.

Σύμφωνα με τις ρυθμίσεις που έγιναν, αναμένεται να δημιουργούνται ειδοποιήσεις στο αρχείο /var/log/snort/alert_json.txt κάθε 30 δευτερόλεπτα για κάθε επίθεση που θα εντοπιστεί. Δοκιμαστικά, τρέχουμε ένα ICMP flood attack από τον host3 στον h1 και με την εντολή *tail –f /var/log/snort/alert_json.txt* μπορούμε να δούμε την κάθε ειδοποίηση που δημιουργείται (επιπλέον μπορεί να χρησιμοποιηθεί το φίλτρο *grep "\"icmp_type\":8"* για να δούμε μόνο τα echo ICMP).


**Εικόνα 4.49:** Snort3 – Αναμενόμενη δημιουργία ειδοποιήσεων

### 4.3.2 Δημιουργία Flows στον Controller

Αφού ρυθμίστηκαν οι ειδοποιήσεις από το IDS, χρησιμοποιήθηκε το REST API του ONOS για την δημιουργία των απαιτούμενων flows τα οποία θα αποτρέπουν την προώθηση των πακέτων σε περίπτωση κάποιας επίθεσης.

Η δημιουργία των flows στον controller, πραγματοποιείται με POST Requests [20] χρησιμοποιώντας JSON Format το οποίο περιλαμβάνει διάφορα objects μέσα στο criteria array αναλόγως των παραμέτρων που θέλουμε να χρησιμοποιήσουμε [21]. Για το κάθε POST Request, μπορούν να δοθούν διάφορες οδηγίες (instructions) στο treatment array για να καθορίσουν την πορεία των πακέτων. Σε περίπτωση που το treatment array μείνει κενό, τότε τα πακέτα γίνονται drop κάτι το οποίο θα χρησιμοποιηθεί για την υλοποίηση του μετριασμού των επιθέσεων που είδαμε.

Σε ότι αφορά πακέτα ICMP, το POST Request πρέπει να περιλαμβάνει τουλάχιστον τα κριτήρια Ingress Port (IN_PORT), EtherType (ETH_TYPE του οποίου η τιμή είναι 0x800 για IPv4) το source και destination MAC address (ETH_SRC και ETH_DST αντίστοιχα) καθώς και ο αριθμός του IP πρωτοκόλλου (αριθμός 1 για ICMP). Με τα ίδια κριτήρια, μπορεί να δημιουργηθεί και το flow για TCP segments με μόνη διαφορά τον αριθμό πρωτοκόλλου IP ο οποίος είναι 6.

Για UDP datagrams τα κριτήρια θα πρέπει να περιλαμβάνουν τουλάχιστον το Ingress Port, EtherType, το Ethernet Source, το destination IP address (IPV4_DST) καθώς και τον αριθμό του IP πρωτοκόλλου (αριθμός 17 για UDP).

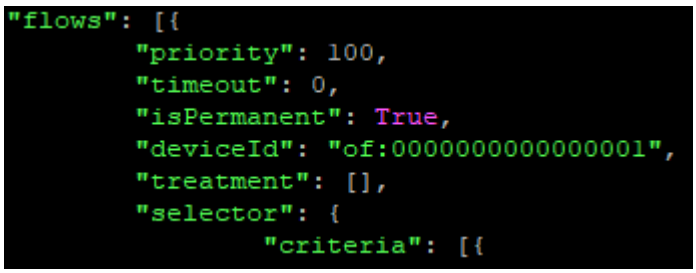
```
"criteria": [{
                "type": "IN_PORT",
                "port": ""
        },
        {
                "type": "ETH_TYPE",
                "ethType": "0x0800"
        },
        {
                "type": "IP_PROTO",
                "protocol": ""
        },
        {
                "type": "ETH_SRC",
                "mac": ""
        },
        {
                "type": "ETH_DST",
                "mac": ""
        }
]
```

**Εικόνα 4.50:** Κριτήρια δημιουργίας flow για ICMP και TCP

```
"criteria": [{
                "type": "IN_PORT",
                "port": ""
        },
        {
                "type": "ETH_TYPE",
                "ethType": "0x800"
        },
        {
                "type": "IPV4_DST",
                "ip": ""
        },
        {
                "type": "IP_PROTO",
                "protocol": "17"
        }
]
```

**Εικόνα 4.51:** Κριτήρια δημιουργίας flow για UDP

Σε κάθε περίπτωση, στο POST Request, περιλαμβάνονται τα properties priority, timeout, isPermanent, deviceId, treatment και selector όπου υπάγονται και τα criteria όπως φαίνεται και πιο κάτω.

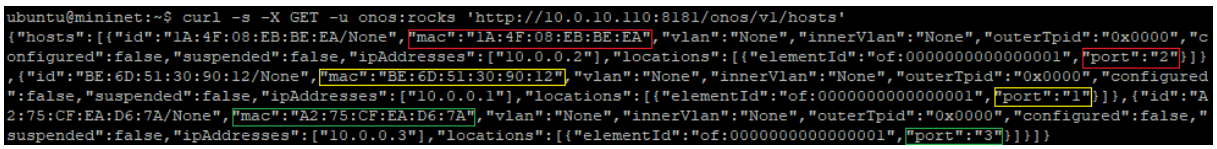



**Εικόνα 4.52**: Flow's properties

Για την αποστολή του Request, χρησιμοποιήθηκε το εργαλείο curl και η εντολή περιλαμβάνει την παράμετρο –X POST, τα headers Content-Type: application/json και Accept: application/json' –d, καθώς και την παράμετρο –u όπου καθορίζονται τα στοιχεία αυθεντικοποίησης. Το request στέλνεται στο core application του ONOS (org.onosproject.core).

Αφού σταλεί επιτυχώς το POST Request, μας δίνεται το flow id (το οποίο θα χρησιμοποιηθεί για να διαγραφεί το flow) ενώ στο Web GUI, μπορούμε να δούμε το νέο flow στο device. Στο πιο κάτω παράδειγμα, οι μεταβλητές ETH_SRC και ETH_DST μπορούν να βρεθούν στην ειδοποίηση που δημιουργεί το IDS, ωστόσο το IN_PORT, το οποίο είναι επίσης μεταβλητή αφού θα είναι διαφορετικό για επιθέσεις από διαφορετικούς hosts, δεν περιέχεται στην ειδοποίηση.

Σύμφωνα με το εγχειρίδιο του ONOS Rest API [23], οι πληροφορίες για τον κάθε host μπορούν να ληφθούν με GET Request στο http://*onos_ip:port*/onos/v1/hosts. Έτσι, μπορεί να γίνει η αντιστοίχιση mac address και ingress port η οποία στη συνέχεια θα συμπεριληφθεί και στον αυτοματισμό.




**Εικόνα 4.53:** Αντιστοίχιση mac address με ingress port

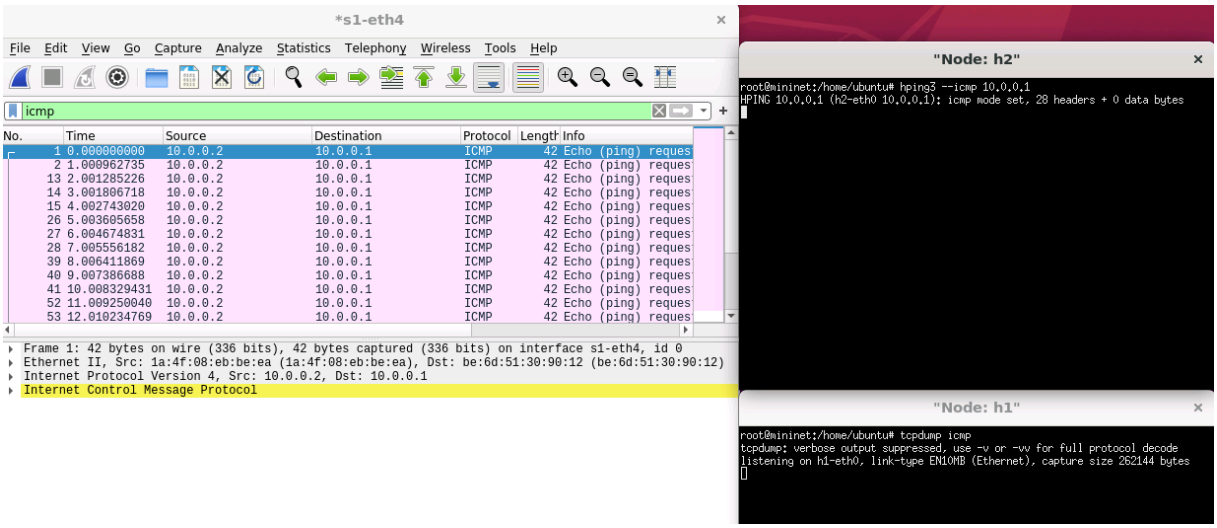
```
ubuntu@mininet:~$ curl -X POST -u onos:rocks --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{
   "flows": [
     {
       "priority": 100,
       "timeout": 0,
       "isPermanent": true,
       "deviceId": "of:0000000000000001",
       "treatment": [],
       "selector": {
         "criteria": [
                  {"type":"IN_PORT","port":"2"},
                  {"type":"ETH_TYPE","ethType":"0x800"},
                  {"type":"ETH_SRC","mac":"1A:4F:08:EB:BE:EA" },
                  {"type":"ETH_DST","mac":"BE:6D:51:30:90:12"},
                  {"type":"IP_PROTO","protocol":"1"}

         ]
       }
     }
   ]
 }' 'http://10.0.10.110:8181/onos/v1/flows?appId=org.onosproject.core'
{"flows":[{"deviceId":"of:0000000000000001","flowId":"2814750639673l7"}]}
```

**Εικόνα 4.53:** Δημιουργία Flow μέσω REST API

| STATE ▾ | PACKETS | DURATION | FLOW PRIORITY | TABLE NAME | SELECTOR | TREATMENT | APP NAME |
|---|---|---|---|---|---|---|---|
| Added | 0 | 56 | 100 | 0 | IN_PORT:2, ETH_DST:BE:6D:51:30:90:12 , ETH_SRC:1A:4F:08:EB:BE:EA, ETH_TYPE:ipv4, IP_PROTO:1 | imm[NOACTION], cleared:false | *core |

**Εικόνα 4.54:** Επιβεβαίωση σωστής δημιουργίας του νέου flow

Με το flow ενεργοποιημένο επιτυγχάνεται η διακοπή προώθησης ICMP πακέτων από τον host2 στον host1 και μπορούμε να το επιβεβαιώσουμε χρησιμοποιώντας εργαλεία όπως το tcpdump στον host1 για ανίχνευση πακέτων ICMP αλλά και με το wireshark στο οποίο μπορούμε να παρατηρήσουμε ότι στο interface s1-eth4 (mirror port) φιλτράροντας για ICMP πακέτα βλέπουμε να υπάρχουν μόνο ICMP echo από τον host2 στον host1 χωρίς echo reply από τον host1 στον host2.




**Εικόνα 4.55:** Επιβεβαίωση μη προώθησης ICMP πακέτων

### 4.3.3 Αυτοματισμός σε Python

Αφού έγινε η επιτυχημένη ανίχνευση των επιθέσεων χρησιμοποιώντας το Snort3 και ο τρόπος δημιουργίας των σωστών flows χρησιμοποιώντας το ONOS REST API για διακοπή προώθησης πακέτων/segments από τον επιτιθέμενο στο θύμα, το επόμενο βήμα ήταν να γίνει ο αυτοματισμός της διαδικασίας.

Η λογική του αυτοματισμού μπορεί να χωριστεί σε τέσσερα βασικά σημεία:

1. Αντιστοίχιση mac addresses με Ethernet ports και ανανέωση κάθε πέντε λεπτά,

2. Συνεχής εξέταση για την ύπαρξη νέας ειδοποίησης από το IDS,

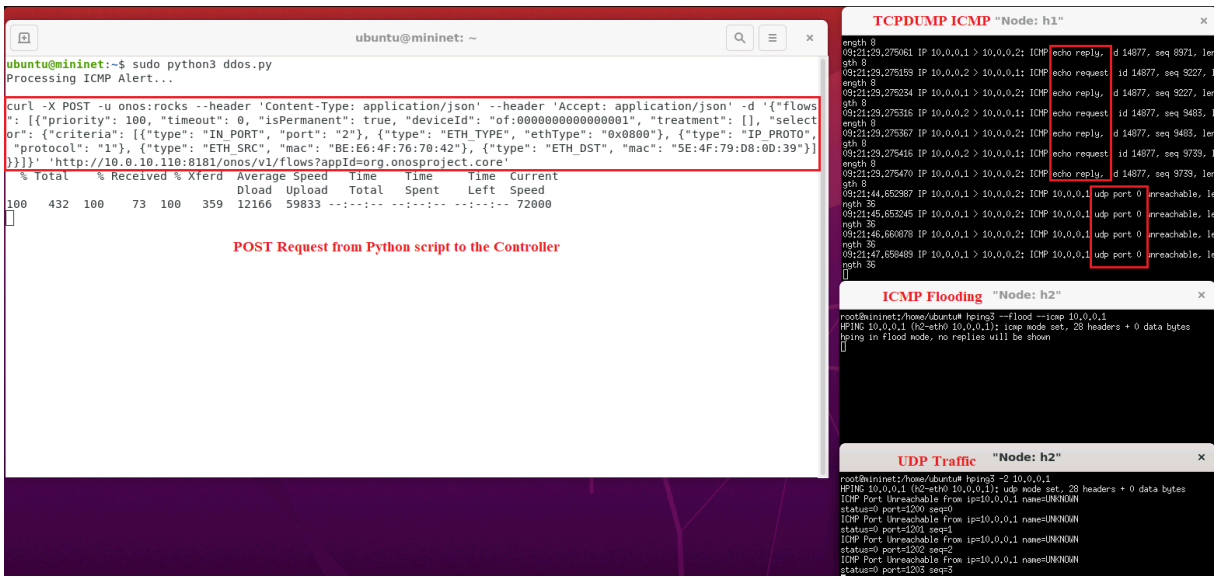
3. Εξέταση ειδοποίησης και δημιουργία του κατάλληλου flow στον controller εάν δεν υπάρχει ήδη,

4. Εξέταση κάθε τριάντα δευτερόλεπτα εάν για κάποιο υφιστάμενο flow υπάρχουν νέες ειδοποιήσεις. Εάν δεν υπάρχουν τότε διαγραφή του flow για συνέχιση της επικοινωνίας.

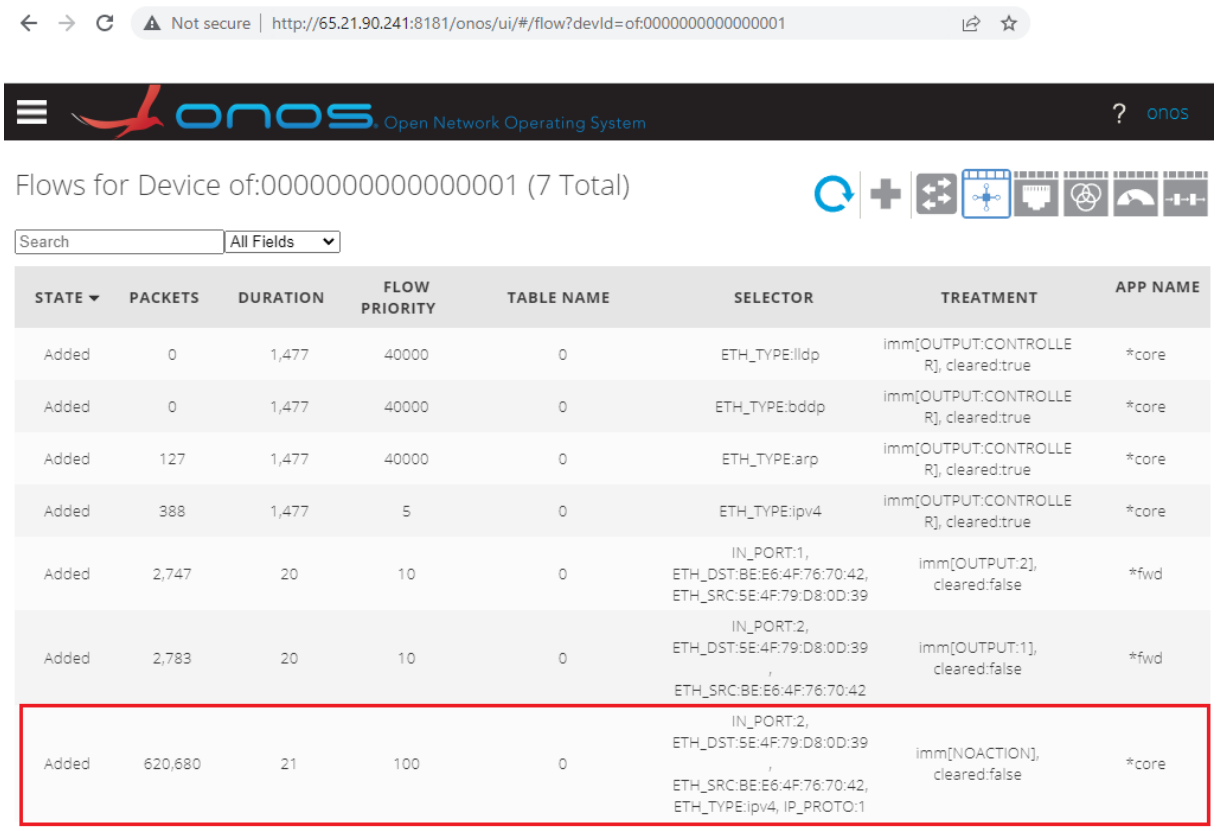
Ο αυτοματισμός έγινε σε Python (Παράρτημα Α.1) και προτείνεται ο owner να καθοριστεί ο root (*chown root:root*) με δικαιώματα 700 (*chmod 700* - μόνο ο owner να μπορεί να διαβάσει, να γράψει και να εκτελέσει το αρχείο). Για την λειτουργία του αυτοματισμού απαιτείται πρώτα να δημιουργηθεί η τοπολογία στο Mininet και καθορισμός του interface το οποίο εξετάζει το IDS. Ακολούθως γίνεται η εκκίνηση του snort3 με την εντολή *sudo service snort3 start* (νοουμένου ότι ακολουθήθηκε η διαδικασία εγκατάστασης του όπως αναφέρθηκε πιο πάνω) και το Python script με την εντολή *sudo python3 <filename.py>*.

Πιο κάτω φαίνονται τα αποτελέσματα αποδοτικότητας για μερικές επιθέσεις:

1. ICMP Flooding από τον host2 στον host1: Ο εντοπισμός και η αντιμετώπιση της επίθεσης γίνεται σε λιγότερο από 3 δευτερόλεπτα (τα πρώτα δύο δευτερόλεπτα χρειάζονται από το IDS το οποίο όπως είδαμε νωρίτερα ρυθμίστηκε να δημιουργεί την ειδοποίηση εάν εντοπιστεί μεγαλύτερος αριθμός πακέτων από τον καθορισμένο σε διάστημα δύο δευτερολέπτων). Όπως φαίνεται και πιο κάτω σταματά η προώθηση πακέτων μόνο του IP πρωτοκόλλου 1 χωρίς να επηρεάζεται η προώθηση πακέτων άλλων πρωτοκόλλων (για παράδειγμα του πρωτοκόλλου 17)



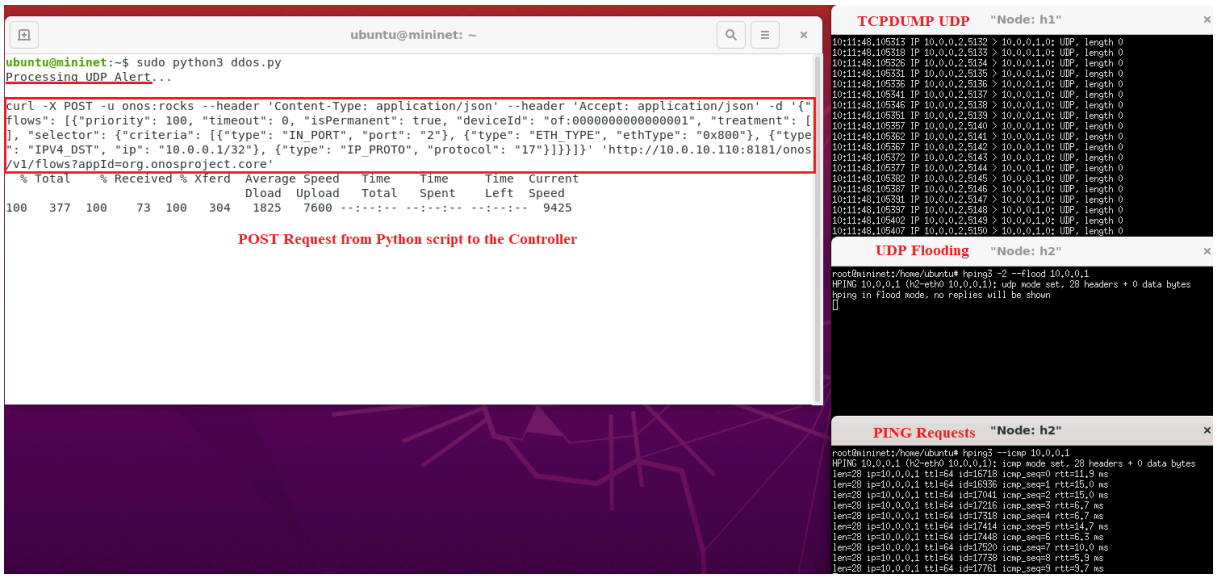

**Εικόνα 4.56:** Αντιμετώπιση ICMP Flooding Attack



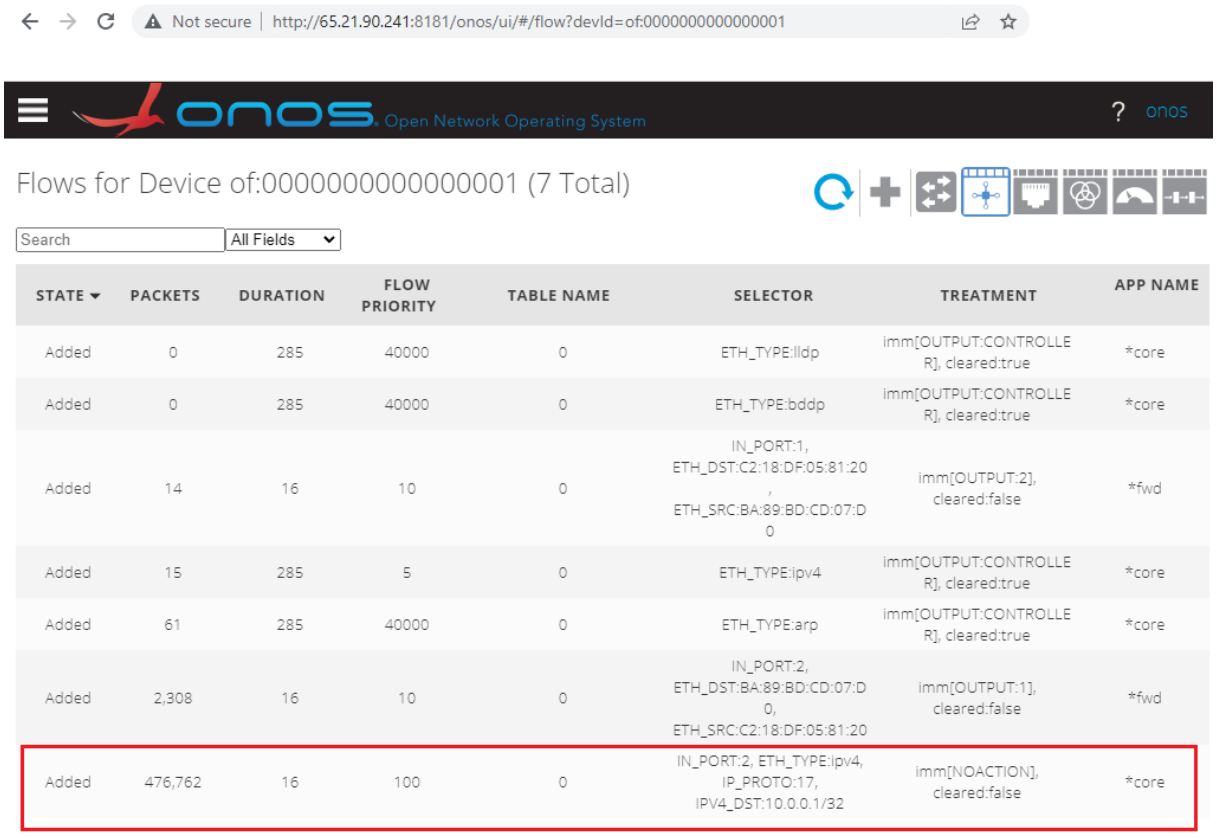

**Εικόνα 4.57:** Καταχώρηση των flows στον Controller

Όπως φαίνεται πιο πάνω, πριν από την δημιουργία του flow, στον host1 έφτασαν μόνο 2783 πακέτα απαντώντας σε 2747 ενώ τα υπόλοιπα 620680 έγιναν drop από τον controller.

2. UDP Flooding Attack από τον host2 στον host1. Παρόμοια αποτελέσματα φαίνεται να έχουμε και σε επιθέσεις UDP Flooding. Ο εντοπισμός και η αντιμετώπιση της επίθεσης γίνεται έγκαιρα χωρίς να επηρεάζονται οι λειτουργίες του host1

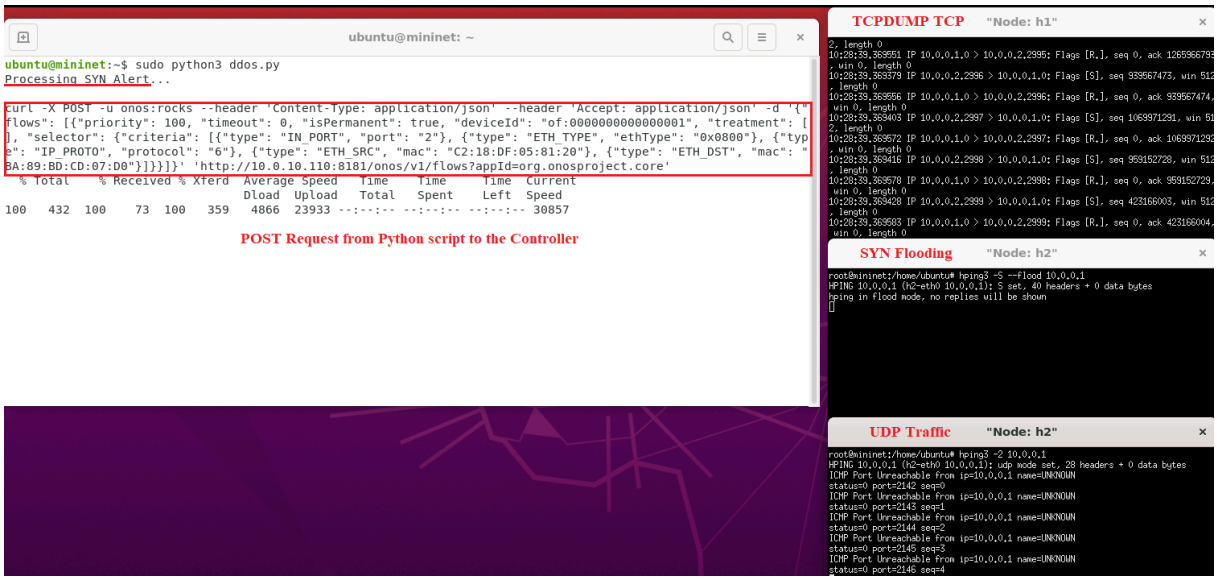



**Εικόνα 4.58:** Αντιμετώπιση UDP Flooding Attack

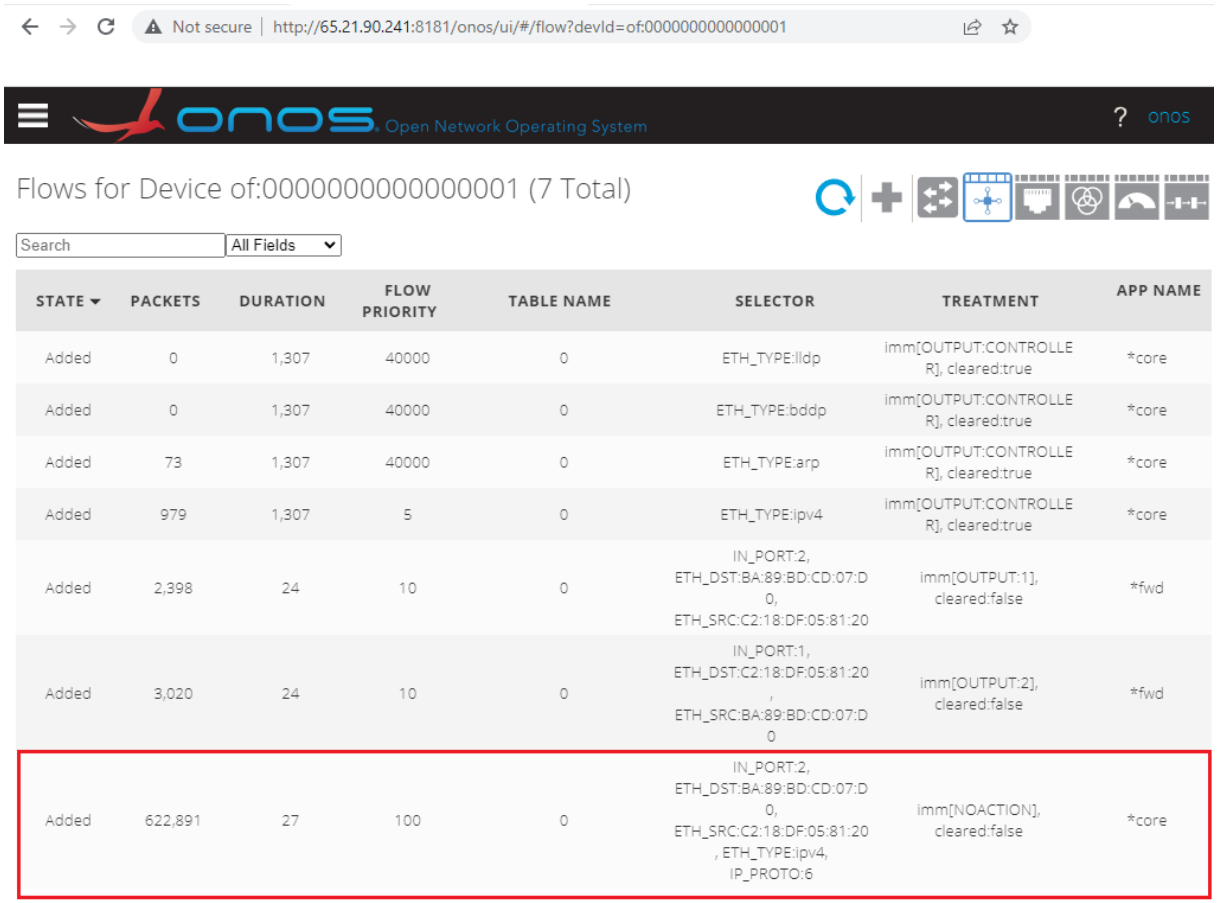



**Εικόνα 4.59:** Καταχώρηση UDP flow στον Controller

3. SYN Flooding Attack από τον host2 προς τον host1. Παρόμοια αποτελέσματα έχουμε και με SYN Flooding attack. Ο εντοπισμός γίνεται έγκαιρα και το flow που δημιουργείται σταματά την προώθηση των TCP segments.

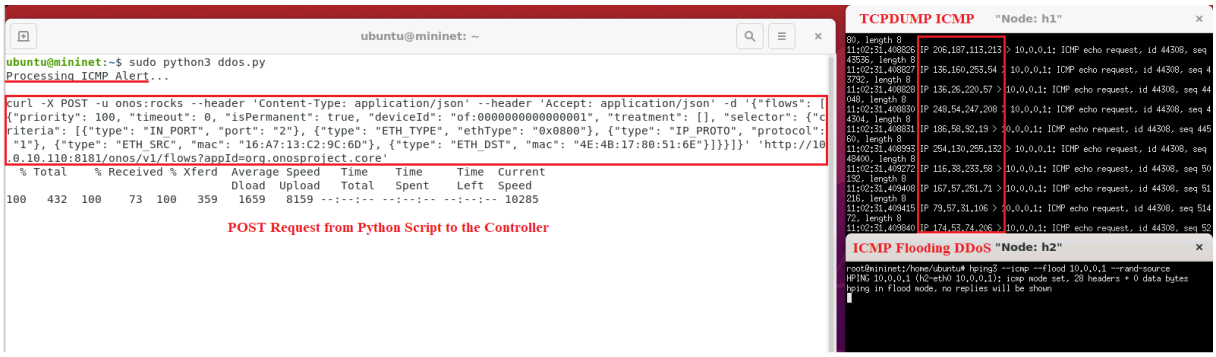



**Εικόνα 4.60:** Αντιμετώπιση TCP Flooding Attack

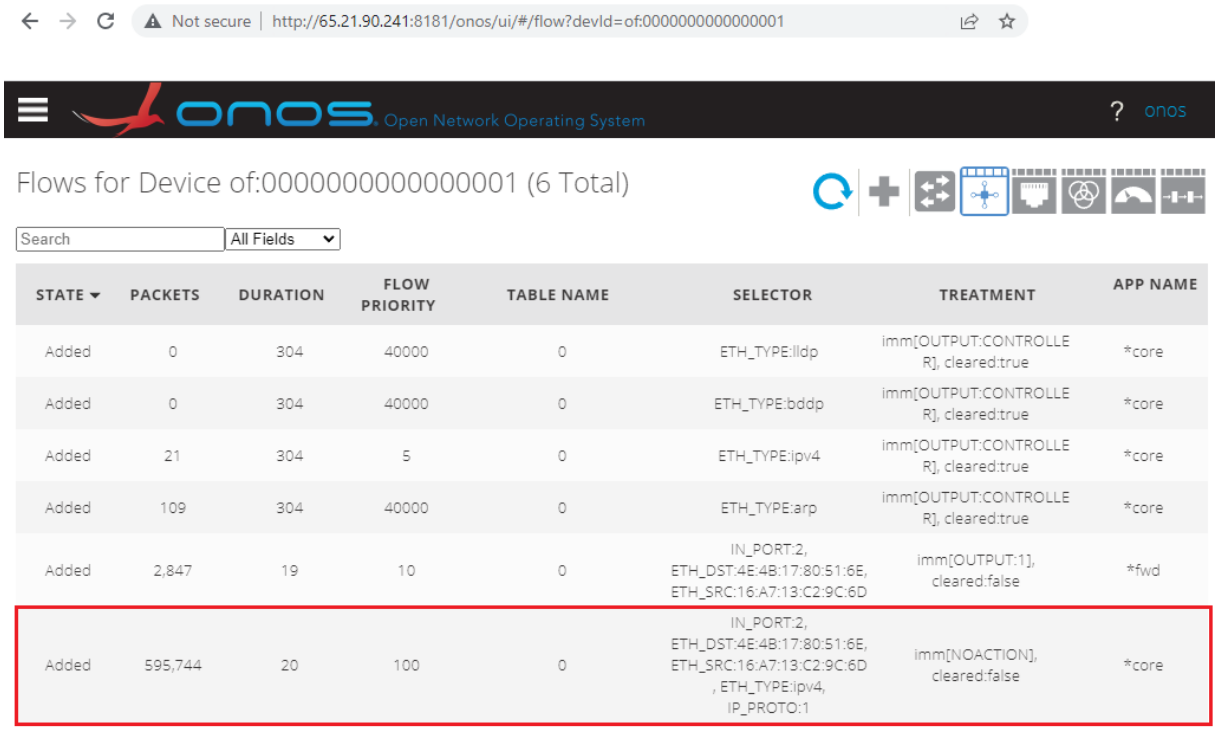



**Εικόνα 4.61:** Καταχώρηση TCP flow στον Controller

4. ICMP Flooding DDoS attack από τον host2 στον host1. Σε αυτή την περίπτωση χρησιμοποιούμε την παράμετρο *--rand-source* στο εργαλείο hping3 η οποία μας δίνει την δυνατότητα να στέλνονται τα πακέτα από διαφορετικά IP addresses προσομοιάζοντας έτσι μια επίθεση DDoS. Η αποτελεσματικότητα του αυτοματισμού είναι ακριβώς η ίδια με αυτή που είδαμε πιο πάνω καθώς εξετάζεται το MAC Address για τον όγκο των δεδομένων που στέλνει. Εδώ είναι καλό να αναφέρουμε ότι σε περιβάλλον SDN, όπως για παράδειγμα το OpenStack που χρησιμοποιήθηκε, παρέχει by default port security κάτι που σημαίνει ότι αποτρέπει την αποστολή και λήψη δεδομένων από κάποιο switchport όταν το MAC Address δεν συμφωνεί με το IP Address που είναι προκαθορισμένα [25].

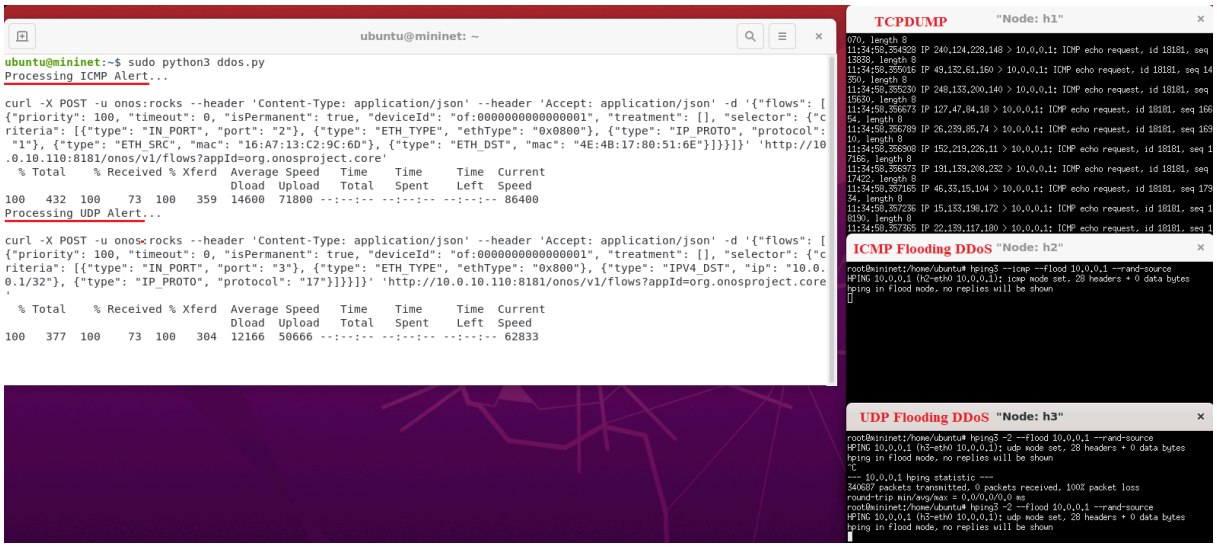



**Εικόνα 4.62:** Αντιμετώπιση ICMP Flooding DDoS Attack

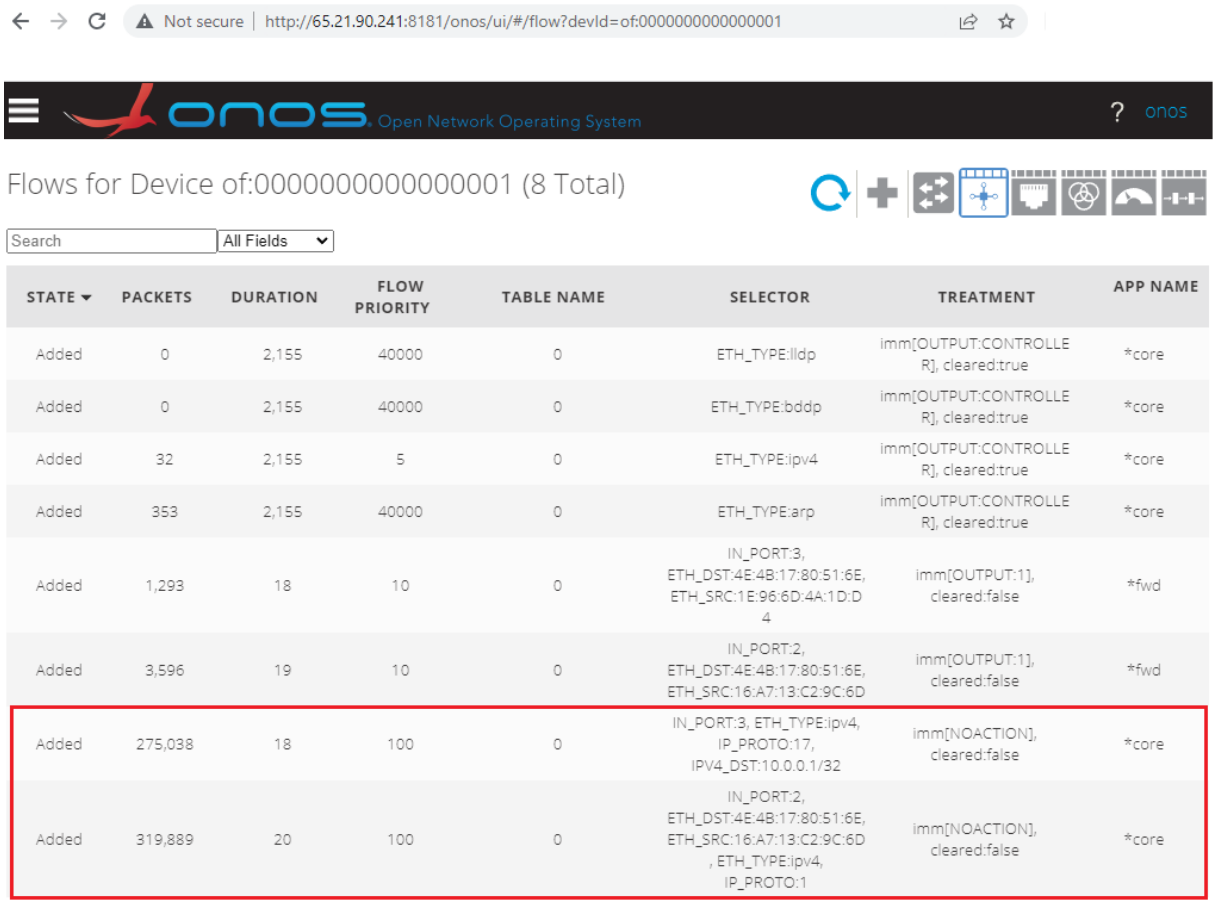



**Εικόνα 4.63:** Καταχώρηση ICMP flow στον Controller

5. Ταυτόχρονες DDoS επιθέσεις ICMP και UDP από τους host2 και host3 στον host1. Παρατηρείται ότι και σε αυτή την περίπτωση, ο αυτοματισμός είναι το ίδιο αποτελεσματικός καθώς δημιουργεί έγκαιρα τα απαιτούμενα flows έτσι ώστε να σταματά αποτελεσματικά η προώθηση των πακέτων.



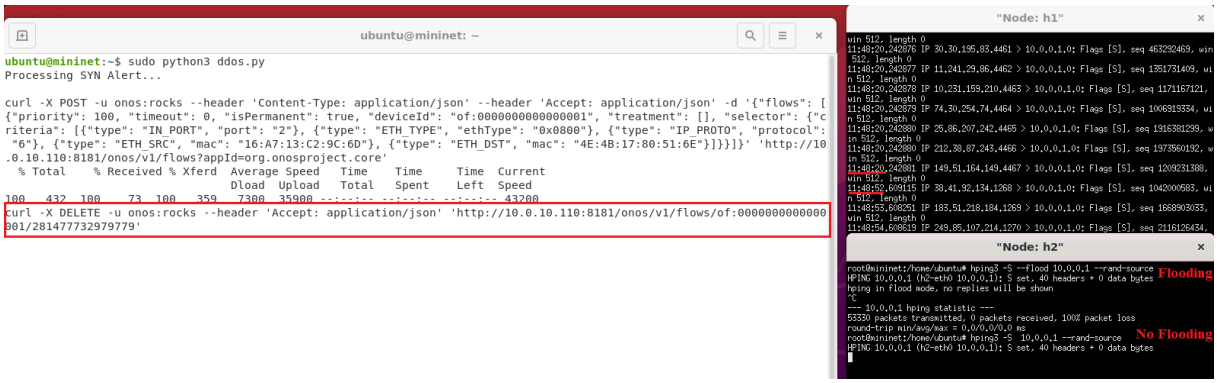

**Εικόνα 4.64:** Αντιμετώπιση ταυτόχρονων επιθέσεων




| STATE | PACKETS | DURATION | FLOW PRIORITY | TABLE NAME | SELECTOR | TREATMENT | APP NAME |
|---|---|---|---|---|---|---|---|
| Added | 0 | 2,155 | 40000 | 0 | ETH_TYPE:lldp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 0 | 2,155 | 40000 | 0 | ETH_TYPE:bddp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 32 | 2,155 | 5 | 0 | ETH_TYPE:ipv4 | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 353 | 2,155 | 40000 | 0 | ETH_TYPE:arp | imm[OUTPUT:CONTROLLER], cleared:true | *core |
| Added | 1,293 | 18 | 10 | 0 | IN_PORT:3, ETH_DST:4E:4B:17:80:51:6E, ETH_SRC:1E:96:6D:4A:1D:D4 | imm[OUTPUT:1], cleared:false | *fwd |
| Added | 3,596 | 19 | 10 | 0 | IN_PORT:2, ETH_DST:4E:4B:17:80:51:6E, ETH_SRC:16:A7:13:C2:9C:6D | imm[OUTPUT:1], cleared:false | *fwd |
| Added | 275,038 | 18 | 100 | 0 | IN_PORT:3, ETH_TYPE:ipv4, IP_PROTO:17, IPV4_DST:10.0.0.1/32 | imm[NOACTION], cleared:false | *core |
| Added | 319,889 | 20 | 100 | 0 | IN_PORT:2, ETH_DST:4E:4B:17:80:51:6E, ETH_SRC:16:A7:13:C2:9C:6D, ETH_TYPE:ipv4, IP_PROTO:1 | imm[NOACTION], cleared:false | *core |

**Εικόνα 4.65:** Καταχώρηση ICMP και UDP flows στον Controller

6. Διαγραφή flow με τον τερματισμό της επίθεσης. Σε αυτό το παράδειγμα προσομοιάζεται η αντιμετώπιση ενός SYN Flooding Attack και ο έλεγχος του τερματισμού της επίθεσης. Όπως φαίνεται πιο κάτω, με τον τερματισμό της επίθεσης, αφού πέρασε το χρονικό περιθώριο που ορίστηκε στο script (32 δευτερόλεπτα) χωρίς κάποια νέα ειδοποίηση, επανέρχεται η επικοινωνία κανονικά.




**Εικόνα 4.66:** Συνέχιση της επικοινωνίας μετά τον τερματισμό της επίθεσης

# Κεφάλαιο 5
# Συμπεράσματα

Ασφαλώς η ανάπτυξη των τεχνολογιών και η μετάβαση τους σε ένα νέο τρόπο λειτουργίας επιφέρει νέες προκλήσεις που αφορούν στην ασφάλεια τους. Όπως είδαμε στα πιο πάνω κεφάλαια, το SDN παρέχει σημαντικές βελτιώσεις στον τρόπο λειτουργίας και περισσότερο στον τρόπο διαχείρισης ενός δικτύου. Η χρησιμοποίηση περισσότερων πρωτοκόλλων σε σχέση με το παραδοσιακό δίκτυο αλλά και οι αρκετές εφαρμογές που το αποτελούν καθιστούν την ασφάλεια του πολυδιάστατη.

Αρχικά είναι σημαντικό να θωρακιστεί όσο το δυνατό καλύτερα το control plane, που αποτελεί τον εγκέφαλο του SDN, καθώς εάν για οποιοδήποτε λόγο αποτύχει η λειτουργία του συνεπάγεται και σε απώλεια της επικοινωνίας. Κατά την δημιουργία ενός SDN θα πρέπει να εφαρμοστούν όλες οι τεχνικές που γνωρίζουμε από τα παραδοσιακά δίκτυα όπως για παράδειγμα redundancy, resiliency, load balancing καθώς και η χρησιμοποίηση boundary firewalls νέας γενιάς έτσι ώστε να μειωθεί σημαντικά η έκθεση του δικτύου σε κακόβουλες ενέργειες.

Ωστόσο, όπως είδαμε στα προηγούμενα κεφάλαια νέα πρωτόκολλα που χρησιμοποιούνται, όπως το OpenFlow, δίνουν την δυνατότητα επιπρόσθετου ελέγχου του δικτύου. Οι δυνατότητες που

προσφέρονται από το OpenFlow, δίνουν την ευκαιρία ανάπτυξης νέων αλγορίθμων εντοπισμού επιθέσεων, χρησιμοποιώντας την πληροφορία που παράγεται σε ένα SDN (μηνύματα όπως Packet_IN, Packet_OUT, Asynchronous, controller-to-switch). Με αυτό τον τρόπο μπορεί να αναπτυχθεί ένα νέο firewall βασιζόμενο σε behaviour/anomaly detection που να ανταποκρίνεται σε κάθε τοπολογία.

Η ύπαρξη του control plane ως κεντρικού στοιχείου προσδίδει καλύτερη εικόνα για την κίνηση και αποδοτικότητα του δικτύου χωρίς να χρειάζεται επιπλέον ρυθμίσεις ή και συσκευές όπως συμβαίνει στα παραδοσιακά δίκτυα ωστόσο χρειάζεται προσοχή κυρίως στον τρόπο επικοινωνίας των επιμέρους στοιχείων. Η ταυτοποίηση και η αυθεντικοποίηση καθώς και η κρυπτογράφηση της επικοινωνίας του control plane με το data plane αλλά και με τα SDN applications πρέπει να ληφθεί υπόψιν καθώς μπορεί να αποτελέσουν τρωτά σημεία του SDN. Οι τεχνικές αυτές ωστόσο δεν διαφέρουν από τις τεχνικές ασφαλείας που ήδη χρησιμοποιούμε στα APIs.

Κατά την υλοποίηση που έγινε στο Κεφάλαιο 4, φαίνεται ότι η δυσκολία είναι στον εντοπισμό της επίθεσης και όχι στην αντιμετώπιση της η οποία γίνεται σχετικά εύκολα. Ιδιαίτερα, κατά τις επιθέσεις DoS το κυριότερο πρόβλημα που αντιμετωπίζουμε είναι ο διαχωρισμός της κακόβουλης ροής με την κανονική ροή στο δίκτυο. Αναλόγως του είδους της DoS επίθεσης πραγματοποιούνται συνεχώς έρευνες και προτείνονται διάφορες λύσεις για αυτόν τον διαχωρισμό. Στην αρχιτεκτονική του SDN είναι εύκολο να δημιουργηθούν isolated honeypots όπου μπορεί να γίνεται περαιτέρω ανάλυση μιας ροής προτού να απορριφθεί από το δίκτυο. Με την αντιμετώπιση της επίθεσης που προτάθηκε στο Κεφάλαιο 4, το request προς τον controller μπορεί να περιλαμβάνει στο πεδίο treatment την δρομολόγηση προς το honeypot σε περίπτωση που επιθυμούμε να αναλύσουμε περισσότερο την ροή.

Επίσης, στην προτεινόμενη λύση δεν λήφθηκαν υπόψη κάποιοι παράμετροι μιας επίθεσης DoS όπως για παράδειγμα, το fragmentation overlap, το bad ή short fragment size τα οποία παρατηρούνται συχνά σε αυτές τις επιθέσεις. Σε μελλοντική εργασία μπορεί να περιληφθούν αυτές οι παράμετροι έτσι το IDS να παράγει τις ανάλογες ειδοποιήσεις. Στο Snort3 που χρησιμοποιήθηκε στην παρούσα εργασία μπορεί να χρησιμοποιηθεί το Inspector Module steam_ip το οποίο αποτελεί ένα επιθεωρητή παρακολούθησης και ανασυγκρότησης (defragmentation) μιας ροής. Για παράδειγμα, χρησιμοποιώντας το gid:123 (stream_ip) σε συνδυασμό με το sid:8 (fragmentation overlap) σε ένα νέο rule στο αρχείο local.rules, δίνεται η δυνατότητα εντοπισμού αυτού του χαρακτηριστικού κατά την επίθεση.

# Βιβλιογραφία


[01] T. Dvorin, 'Cloud Computing History and Evolution | Unboundsecurity', *Unbound Security*, May 11, 2021. https://www.unboundsecurity.com/blog/evolution-and-future-of-cloud/ (accessed Apr. 09, 2022).

[02] L. Xizi, 'In Network Virtualization, How Do SDN and NFV Differ?', *Ormuco*, Jun. 19, 2019. https://ormuco.com/blog/network-virtualization-how-do-sdn-nfv-differ/ (accessed Apr. 09, 2022).

[03] O. Blial, M. Ben Mamoun, and R. Benaini, 'An Overview on SDN Architectures with Multiple Controllers', *Journal of Computer Networks and Communications*, vol. 2016, p. e9396525, Apr. 2016, doi: 10.1155/2016/9396525.

[04] Y. Liu, B. Zhao, P. Zhao, P. Fan, and H. Liu, 'A survey: Typical security issues of software-defined networking', *China Communications*, vol. 16, no. 7, pp. 13–31, Jul. 2019, doi: 10.23919/JCC.2019.07.002.

[05] D. Melkov and S. Paulikas, 'Security Benefits and Drawbacks of Software-Defined Networking', in *2021 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream)*, Apr. 2021, pp. 1–4. doi: 10.1109/eStream53087.2021.9431466.

[06] J. H. Cox, R. J. Clark, and H. L. Owen, 'Leveraging SDN to Improve the Security of DHCP', in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, New Orleans Louisiana USA, Mar. 2016, pp. 35–38. doi: 10.1145/2876019.2876028.

[07] E. O. Zaballa, D. Franco, M. Aguado, and M. S. Berger, 'Next-Generation SDN and Fog Computing: A New Paradigm for SDN-Based Edge Computing', p. 8, 2020.

[08] Z. Fan, Y. Xiao, A. Nayak, and C. Tan, 'An improved network security situation assessment approach in software defined networks', *Peer-to-Peer Netw. Appl.*, vol. 12, no. 2, pp. 295–309, Mar. 2019, doi: 10.1007/s12083-017-0604-2.

[09] Z. Shao, X. Zhu, A. M. M. Chikuvanyanga, and H. Zhu, 'Blockchain-Based SDN Security Guaranteeing Algorithm and Analysis Model', in *Wireless and Satellite Systems*, vol. 281, M. Jia, Q. Guo, and W. Meng, Eds. Cham: Springer International Publishing, 2019, pp. 348–362. doi: 10.1007/978-3-030-19156-6_32.

[10] T. Xu, D. Gao, P. Dong, T. Zheng, and J. Sun, 'SmartSec: A Smart Security Mechanism for the New-Flow Attack in Software-Defined Networking', in *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, Jun. 2017, pp. 1–7. doi: 10.1109/VTCSpring.2017.8108579.

[11] S. Usman, I. Winarno, and A. Sudarsono, 'Implementation of SDN-based IDS to protect Virtualization Server against HTTP DoS attacks', in *2020 International Electronics Symposium (IES)*, Sep. 2020, pp. 195–198. doi: 10.1109/IES50839.2020.9231699.

[12] 'Enterprise Networking, Security, and Automation - Controllers', Cisco Networking Academy. https://contenthub.netacad.com/ensa/13.5.1 (accessed Apr. 12, 2022).

[13] 'SDN and Controller-Based Networks > Introduction to Controller-Based Networking | Cisco Press'. https://www.ciscopress.com/articles/article.asp?p=2995354&seqNum=2 (accessed Apr. 17, 2022).

[14] 'SDN Architecture', Issue 1, *Open Networking Foundation*, ONF TR-502. Accessed: Apr. 16, 2022. [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf

[15] 'OpenFlow Switch Specification - Version 1.5.1', *Open Networking Foundation,* ONF TS-025 . Accessed: Apr. 18, 2022. [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[16] 'DevStack - Openstack documentation', *OpenStack Devstack Team* https://docs.openstack.org/devstack/latest/ (accessed Mar. 13, 2022).

[17] 'Ubuntu 20.04 LTS (Focal Fossa) Daily Build [20220311]'. https://cloud-images.ubuntu.com/focal/current/ (accessed Mar. 13, 2022).

[18] ‘Open Network Operating System (ONOS) SDN Controller for SDN/NFV Solutions’, *Open Networking Foundation*. https://opennetworking.org/onos/ (accessed Mar. 20, 2022).

[19] ‘hping3(8) - Linux man page’. https://linux.die.net/man/8/hping3 (accessed Mar. 25, 2022).

[20] ‘iPerf - The TCP, UDP and SCTP network bandwidth measurement tool’. https://iperf.fr/ (accessed Mar. 27, 2022).

[21] ‘Snort - Network Intrusion Detection & Prevention System’. https://www.snort.org/ (accessed Mar. 29, 2022).

[22] ‘N. Dietrich - Snort 3.1.18.0 on Ubuntu 18 & 20: Configuring a Full NIDS & SIEM’, CC BY-NC-SA 4.0. Accessed: Mar. 29, 2022. [Online]. Available: https://www.snort.org/documents/snort-3-1-18-0-on-ubuntu-18-20

[23] ‘ONOS Guides - Appendix B: REST API - ONOS - Wiki’. https://wiki.onosproject.org/display/ONOS/Appendix+B%3A+REST+API (accessed Apr. 02, 2022).

[24] ‘ONOS Guides - Flow Rules - ONOS - Wiki’. https://wiki.onosproject.org/display/ONOS/Flow+Rules (accessed Apr. 02, 2022).

[25] N. Abbas, ‘Managing port level security in OpenStack’, *Superuser*, Apr. 21, 2017. https://superuser.openstack.org/articles/managing-port-level-security-openstack/ (accessed Apr. 03, 2022).

# Παράρτημα Α
# Αυτοματισμός σε Python

## Α.1 Python Script

```
import os
import json
import time

path_to_alerts = "/var/log/snort/alert_json.txt" #Log file
path_to_latest_alerts = "/home/ubuntu/latest_alerts.json" #Get the latest 10 alerts from log file
path_to_hosts = "/home/ubuntu/hosts.json" #For Network hosts details
path_to_request_response = "/home/ubuntu/requestResponse.json" #For flowId

pre_request_text = "curl -X POST -u onos:rocks --header 'Content-Type: application/json' --header 'Accept: application/json' -d '"
post_request_text = "' 'http://10.0.10.110:8181/onos/v1/flows?appId=org.onosproject.core'"

delete_flow_text = "curl -X DELETE -u onos:rocks --header 'Accept: application/json' 'http://10.0.10.110:8181/onos/v1/flows/"

ICMP_SYN_REQUEST = \
{
    "flows": [{
        "priority": 100,
        "timeout": 0,
        "isPermanent": True,
        "deviceId": "of:0000000000000001", #Would be a variable for topology with more devices
        "treatment": [],
        "selector": {
            "criteria": [{
                    "type": "IN_PORT",
                    "port": "" #Variable
                },
                {
                    "type": "ETH_TYPE",
                    "ethType": "0x0800" #IPv4
```

```
                },
                {
                    "type": "IP_PROTO",
                    "protocol": "" #Variable 1 or 6
                },
                {
                    "type": "ETH_SRC",
                    "mac": "" #Variable
                },
                {
                    "type": "ETH_DST",
                    "mac": "" #Variable
                }
            ]
        }
    }]
}

UDP_REQUEST = \
{
    "flows": [{
        "priority": 100,
        "timeout": 0,
        "isPermanent": True,
        "deviceId": "of:0000000000000001",
        "treatment": [],
        "selector": {
            "criteria": [{
                    "type": "IN_PORT",
                    "port": "" #Variable
                },
                {
                    "type": "ETH_TYPE",
                    "ethType": "0x800" #IPv4
                },
                {
                    "type": "IPV4_DST",
                    "ip": "" #Variable
                },
                {
                    "type": "IP_PROTO",
                    "protocol": "17" #UDP
                }
            ]
        }
    }]
}

# Get Ports
os.system("curl -s -X GET -u onos:rocks 'http://10.0.10.110:8181/onos/v1/hosts' > "
+ path_to_hosts)
with open(path_to_hosts) as h:
    hosts = json.load(h)["hosts"]


# Get Latest Alerts

latestTimestamp = ""
i = 0
# Create the lists
ActiveICMPAlertList = []
ActiveUDPAlertList = []

ICMPAlert = {"sid": "", "port": "", "ETH_SRC": "", "ETH_DST": "", "deviceId": "",
"flowId": "", "seconds": 0}
UDPAlert = {"port": "", "IPV4_DST": "", "deviceId": "", "flowId": "", "seconds": 0}

lastHostsUpdateTime = 0

while True:
    time.sleep(0.1)
    currentTime = int(time.time())

    # Update Hosts
    if currentTime % 300 == 0 and lastHostsUpdateTime < currentTime : # Update hosts
    information every 5 minutes (modulo 300)
        os.system("curl -s -X GET -u onos:rocks
        'http://10.0.10.110:8181/onos/v1/hosts' > " + path_to_hosts)
        with open(path_to_hosts) as h:
            hosts = json.load(h)["hosts"]
        lastHostsUpdateTime = currentTime

    #Format hosts output to JSON Format
    os.system("echo [ > " + path_to_latest_alerts)
    os.system("tail -n 10 " + path_to_alerts + " | sed '$!s/$/,/' >> " +
    path_to_latest_alerts)
```

```
112         os.system("echo ] >> " + path_to_latest_alerts)
113
114         with open(path_to_latest_alerts) as a:
115             alerts = json.load(a)
116
117     # Check if a flow can be removed
118         for activeAlert in ActiveICMPAlertList:
119             found = False
120             for alert in alerts:
121                 if (alert['sid'] == 10000001 and alert['icmp_type'] !=0) or alert['sid']
                    == 10000003: # ICMP (any type except of reply (type 0) or SYN (TCP)
122                     for host in hosts:
123                         if host["mac"] == alert["eth_src"]: #Match mac with port
124                             for location in host["locations"]:
125                                 ICMPAlert["port"] = location["port"]
126                     ICMPAlert["ETH_SRC"] = alert["eth_src"]
127                     ICMPAlert["ETH_DST"] = alert["eth_dst"]
128                     ICMPAlert["sid"] = alert["sid"]
129                     #Time interval to check if a flow can be removed. Alerts are created
                        every 30 seconds, thus we can check every 32 seconds
130                     if alert['seconds'] > currentTime - 32 and activeAlert["port"] ==
                        ICMPAlert["port"] and activeAlert["ETH_SRC"] == ICMPAlert["ETH_SRC"]
                        and activeAlert["ETH DST"] == ICMPAlert["ETH DST"] and
131                         found = True
132             if not found:
133                 removeFlowCommand = delete_flow_text + activeAlert["deviceId"] + "/" +
                    activeAlert["flowId"] + "'"
134                 print(removeFlowCommand)
135                 os.system(removeFlowCommand)
136                 ActiveICMPAlertList.remove(activeAlert)
137
138         for activeAlert in ActiveUDPAlertList:
139             found = False
140             for alert in alerts:
141                 if alert['sid'] == 10000002: # UDP
142                     for host in hosts:
143                         if host["mac"] == alert["eth_src"]: #Match mac with port
144                             for location in host["locations"]:
145                                 UDPAlert["port"] = location["port"]
146                     UDPAlert["IPV4_DST"] = alert["dst_addr"]
147                     if alert['seconds'] > currentTime - 32 and activeAlert["port"] ==
                        UDPAlert["port"] and activeAlert["IPV4_DST"] == UDPAlert["IPV4_DST"]:
148                         found = True
149             if not found:
150                 removeFlowCommand = delete_flow_text + activeAlert["deviceId"] + "/" +
                    activeAlert["flowId"] + "'"
151                 print(removeFlowCommand)
152                 os.system(removeFlowCommand)
153                 ActiveUDPAlertList.remove(activeAlert)
154
155     # Check if a flow needs to be created
156         for alert in alerts:
157             if alert['timestamp'] > latestTimestamp and alert['seconds'] > currentTime -
                59: #Avoid recreating the flow in case of script restart
158                 latestTimestamp = alert['timestamp']
159                 if (alert['sid'] == 10000001 and alert['icmp_type'] !=0) or alert['sid']
                    == 10000003: # ICMP or SYN
160                     if alert['sid'] == 10000001:
161                         print ("Processing ICMP Alert...\n")
162                     elif alert['sid'] == 10000003:
163                         print ("Processing SYN Alert...\n")
164                     for flow in ICMP_SYN_REQUEST["flows"]:
165                         for criteria in flow["selector"]["criteria"]: #Get the variables
                            values for the Request
166                             if criteria["type"] == "IN_PORT":
167                                 for host in hosts:
168                                     if host["mac"] == alert["eth_src"]:
169                                         for location in host["locations"]:
170                                             criteria["port"] = location["port"]
171                                             ICMPAlert["port"] = location["port"]
172                             elif criteria["type"] == "ETH_SRC":
173                                 criteria["mac"] = alert["eth_src"]
174                                 ICMPAlert["ETH_SRC"] = alert["eth_src"]
175                             elif criteria["type"] == "ETH_DST":
176                                 criteria["mac"] = alert["eth_dst"]
177                                 ICMPAlert["ETH_DST"] = alert["eth_dst"]
178                             elif criteria["type"] == "IP_PROTO":
179                                 if alert['sid'] == 10000001:
180                                     criteria["protocol"] = "1"
181                                 elif alert['sid'] == 10000003:
182                                     criteria["protocol"] = "6"
183                     ICMPAlert["sid"] = alert["sid"]
184                     ICMPAlert["seconds"] = alert["seconds"]
185                     found = False
186                     for activeAlert in ActiveICMPAlertList:
187                         if activeAlert["port"] == ICMPAlert["port"] and
```

```
                    print(request)
                    os.system(request + ' > ' + path_to_request_response)
                    with open(path_to_request_response) as r: #Get flowId
                        flows = json.load(r)["flows"]
                    for flow in flows:
                        ICMPAlert["deviceId"] = flow["deviceId"]
                        ICMPAlert["flowId"] = flow["flowId"]
                    ActiveICMPAlertList.append(ICMPAlert)
                else:
                    print("Flow already exists... Nothing to do")
            elif alert['sid'] == 10000002: # UDP
                print ("Processing UDP Alert...\n")
                for flow in UDP_REQUEST["flows"]:
                    for criteria in flow["selector"]["criteria"]:
                        if criteria["type"] == "IN_PORT":
                            for host in hosts:
                                if host["mac"] == alert["eth_src"]:
                                    for location in host["locations"]:
                                        criteria["port"] = location["port"]
                                        UDPAlert["port"] = location["port"]
                        elif criteria["type"] == "IPV4_DST":
                            criteria["ip"] = alert["dst_addr"] + "/32"
                            UDPAlert["IPV4_DST"] = alert["dst_addr"]
                UDPAlert["seconds"] = alert["seconds"]
                found = False
                for activeAlert in ActiveUDPAlertList:
                    if activeAlert["port"] == UDPAlert["port"] and
                    activeAlert["IPV4_DST"] == UDPAlert["IPV4_DST"]: # if flow
                    already exists
                        activeAlert["seconds"] = alert["seconds"]
                        found = True
                if not found:
                    request = pre_request_text + json.dumps(UDP_REQUEST) +
                    post_request_text
                    print(request)
                    os.system(request + ' > ' + path_to_request_response)
                    with open(path_to_request_response) as r:
                        flows = json.load(r)["flows"]
                    for flow in flows:
                        UDPAlert["deviceId"] = flow["deviceId"]
                        UDPAlert["flowId"] = flow["flowId"]
                    ActiveUDPAlertList.append(UDPAlert)
                else:
                    print("Flow already exists... Nothing to do")
```