ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αντιμετώπιση δικτυακών επιθέσεων με χρήση τεχνολογιών αιχμής, εφαρμογή του OpenFlow για περιπτώσεις DDoS detection και hybrid honeypots.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Χαρτσιά Παντελεήμονα Κωνσταντίνου

Επιβλέπων : Ευστάθιος Συκάς

Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αντιμετώπιση δικτυακών επιθέσεων με χρήση τεχνολογιών αιχμής, εφαρμογή του OpenFlow για περιπτώσεις DDoS detection και hybrid honeypots.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Χαρτσιά Παντελεήμονα Κωνσταντίνου

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

Αθήνα, Δεκέμβριος 2015

.....

Χαρτσιάς Παντελεήμων Κωνσταντίνος

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

Περίληψη

Ο σκοπός της διπλωματικής εργασίας είναι η αντιμετώπιση δικτυακών επιθέσεων χρησιμοποιώντας τεχνολογίες αιχμής. Καθώς το διαδίκτυο εξελίσσεται και τα δίκτυα υπολογιστών γίνονται όλο και μεγαλύτερα, η ασφάλεια δικτύων είναι ένας από τους σημαντικότερους παράγοντες που πρέπει να ληφθούν υπόψη. Υπάρχουν διάφορα είδη δικτυακών επιθέσεων όπως και τρόποι αντιμετώπισης. Εμείς θα επικεντρωθούμε σε δύο τρόπους αντιμετώπισης δικτυακών επιθέσεων, στον εντοπισμό κατανεμημένων επιθέσεων άρνησης υπηρεσίας με χρήση εντροπίας και στην χρήση υβριδικού honeypot με σκοπό την διαχείριση honeynets.

Η εργασία αυτή ασχολείται με την χρήση των ευφυών προγραμματιζόμενων δικτύων και πιο συγκεκριμένα την τεχνολογία OpenFlow προκειμένου να υλοποιηθούν οι προηγούμενοι τρόποι αντιμετώπισης δικτυακών επιθέσεων. Η τεχνολογία OpenFlow διαχωρίζει το επίπεδο προώθησης και ελέγχου στις δικτυακές συσκευές και πλέον χρησιμοποιείται ένας εξωτερικός ελεγκτής που αποτελεί κατά κάποιο τρόπο το λειτουργικό σύστημα του δικτύου. Έγινε εκτενής ανάλυση των διαφόρων εκδόσεων του OpenFlow και των κυριότερων προσθηκών αλλαγών σε κάθε μία. Επίσης έγινε περιληπτική ανάλυση βασικών OpenFlow ελεγκτών ανοιχτού κώδικα με κυριότερη ανάλυση του POX και των βασικών εξαρτημάτων του.

Υλοποιήθηκαν 2 προσομοιώσεις με τον ελεγκτή POX στο περιβάλλον εξομοίωσης δικτύων, mininet. Το πρώτο ήταν υλοποίηση υβριδικού honeypot το οποίο ανακατευθύνει την δικτυακή κίνηση μετά από έναν συγκεκριμένο αριθμό πακέτων που φτάνουν στον ελεγκτή. Πραγματοποιήθηκε τόσο για ICMP κίνηση όσο και για UDP με την χρήση DNS honeypot και ενός απλοϊκού DNS server. Το δεύτερο ήταν ένας τρόπος εντοπισμού κατανεμημένων επιθέσεων άρνησης υπηρεσίας μέσω της εντροπίας από τη θεωρία πληροφορίας. Αρχικά έγινε σύγκριση των τιμών εντροπίας του Shannon και της εντροπίας του Rényi με παραμέτρους $\alpha = 2$ και $\alpha = 0.5$ για συγκεκριμένα μεγέθη παραθύρου πακέτων. Επίσης με κατάλληλη τροποποίηση του κώδικα υπολογίστηκε η μέγιστη εντροπία του δικτύου. Τέλος με βάση τη μέγιστη εντροπία ως όριο, τροποποιήθηκε ο κώδικας έτσι ώστε κάθε φορά που εντοπίζεται κατανεμημένη επίθεση άρνησης υπηρεσίας, να στέλνονται στο email του διαχειριστή δικτύου αρχεία κειμένου με τις πιθανές IP διευθύνσεις της επίθεσης.

Λέξεις Κλειδιά: Ευφυή προγραμματιζόμενα δίκτυα, OpenFlow, Pox, honeypots, υβριδικό honeypot, κατανεμημένες επιθέσεις άρνησης υπηρεσίας, εντροπία

Abstract

The scope of this diploma thesis is dealing with network attacks using high end technology. As the internet evolves and computer networks become bigger and bigger, network security has become one of the most important factors to consider. There are several types of network attacks as well as ways to defend against them. We are going to focus on two ways of defence against network attacks, first by detecting DDoS attacks using entropy and secondary using a hybrid honeypot which is used to administrate honeynets.

This paper addresses the usage of Software Defined Networks (SDN) and more specifically the usage of OpenFlow technology in order to implement the above types of network defence. OpenFlow separates the data and control plane on each network device and there is an external controller which acts as if it is the operating system of the network. Different OpenFlow versions were analysed extensively and their main additions and changes were mentioned as well. Moreover different open source OpenFlow controllers were analysed briefly and more emphasis was put on POX controller and its main modules.

Two simulations were implemented with POX controller using mininet, which is a network emulation software. First simulation is about a hybrid honeypot which redirects network traffic after a specific number of packets is received by the controller. Redirected traffic was ICMP packets as well as UDP, using DNS honeypot and a minimal DNS server. Second simulation is about detecting DDoS attacks using information entropy. Firstly, a comparison was implemented between Shannon's and Rényi's entropy estimation methods for specific windows of packets. Moreover, with appropriate modification of the code, the maximum entropy of the network was calculated. Finally, using the maximum entropy as a threshold, the code was modified again in order to detect DDoS attacks and every time an attack was detected, a text with possible ip addresses of the attack was sent by email to the network administrator.

Keywords: Software Defined Networks, OpenFlow, DDoS, Pox controller, honeypots, hybrid honeypot, entropy

Ευχαριστίες

Η παρούσα διπλωματική εργασία είναι το επιστέγασμα των προπτυχιακών μου σπουδών στο Εθνικό Μετσόβιο Πολυτεχνείο και θα ήθελα να ευχαριστήσω όσους συνέβαλαν άμεσα ή έμμεσα στην προσπάθειά μου να τις ολοκληρώσω.

Θα ήθελα αρχικά να ευχαριστήσω τον κ. Συκά Ευστάθιο καθηγητή ΕΜΠ για την δυνατότητα την οποία μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον και σημαντικό θέμα των δικτύων υπολογιστών. Επίσης θα ήθελα να ευχαριστήσω τον κ. Καναβίδη Κωνσταντίνο υποψήφιο διδάκτορα και τον κ. Κοροβέση Ιωάννη υπεύθυνο ερευνητή και ιδρυτή του εργαστηρίου Islab του Ινστιτούτου Πληροφορικής και Τηλεπικοινωνιών του Ε.Κ.Ε.Φ.Ε. «Δημόκριτος» για τη βοήθειά τους και της χρήσιμες παρατηρήσεις, προτάσεις και συμβουλές τους καθ' όλη τη διάρκεια συγγραφής της εργασίας.

Ένα ευχαριστώ επίσης στον φίλο και συνάδελφο, Αδάμ για την υποστήριξή του στις απορίες μου για τον τομέα των Software Defined Networks.

Τέλος θα ήθελα να ευχαριστήσω όλα τα άτομα που ήταν κοντά μου μέσα στις δυσκολίες που προέκυψαν κατά καιρούς και ιδίως τους γονείς μου.

Χαρτσιάς Παντελεήμων Κωνσταντίνος

Νοέμβριος 2015

Πίνακας περιεχομένων

1

Εισαγωγή – Αντικείμενο

Στην παρούσα διπλωματική μελετάμε τομείς που αφορούν την ασφάλεια δικτύων υπολογιστών και πιο συγκεκριμένα την αντιμετώπιση δικτυακών επιθέσεων. Στο παρελθόν οι χάκερς ήταν πολύ καλοί γνώστες προγραμματισμού καθώς επίσης και δικτύων επικοινωνιών. Πλέον ο οποιοσδήποτε μπορεί να γίνει χάκερ, κατεβάζοντας έτοιμα προγράμματα από το διαδίκτυο. Όλα αυτά τα πολύπλοκα εργαλεία επιθέσεων έχουν καταστήσει τις ανάγκες για ασφάλεια δικτύων και πολιτικών ασφαλείας επιτακτικές όσο ποτέ άλλοτε. Σκοπός των επιτιθέμενων είναι είτε να διακόψουν την λειτουργία, είτε να αποκτήσουν παράνομη πρόσβαση σε υπολογιστές προκειμένου να υποκλέψουν στοιχεία, να τα παραποιήσουν ή ακόμη και να τα καταστρέψουν. Υπάρχουν ήδη αρκετές μέθοδοι αντιμετώπισης δικτυακών επιθέσεων, αλλά πλέον ανοίγονται και νέοι τρόποι υλοποίησής τους με νέες τεχνολογίες, οι οποίες μπορούν να προσφέρουν ακόμα περισσότερη ευελιξία και περισσότερες δυνατότητες σε έναν διαχειριστή δικτύου.

Η νέα τεχνολογία η οποία δίνει αυτή τη δυνατότητα είναι τα ευφυή προγραμματιζόμενα δίκτυα (Software Defined Networks - SDN). Στην παρούσα διπλωματική θα αναλυθεί ένα από τα επικρατέστερα παραδείγματα ευφυών προγραμματιζόμενων δικτύων, το OpenFlow. Πλέον το κομμάτι προώθησης και ελέγχου που πραγματοποιούνταν μέχρι τώρα μαζί στις δικτυακές συσκευές έχει διαχωριστεί και οι OpenFlow μεταγωγείς αναλαμβάνουν μόνο την προώθηση πακέτων ενώ εισάγονται πλέον ελεγκτές που αποτελούν κατά μία έννοια το νέο λειτουργικό σύστημα των δικτυακών συσκευών και αναλαμβάνουν να προγραμματίσουν τη λειτουργία τους. Αυτός ο προγραμματισμός δίνει νέες δυνατότητες στις δικτυακές συσκευές και πλέον μπορούν να επιτελούν πολύπλοκες λειτουργίες όπως το να κάνουν εξισορρόπηση δικτυακής κίνησης (load balancing), προσαρμοστικό έλεγχο της δικτυακής κίνησης, να εντοπίζουν επιθέσεις άρνησης υπηρεσίας (DoS) ή ακόμη και να ελέγχουν τους μεταγωγείς κατά τέτοιο τρόπο ώστε να υπάρχει η ελάχιστη κατανάλωση ενέργειας.

Στην παρούσα εργασία θα ασχοληθούμε με τον POX ελεγκτή, ο οποίος είναι OpenFlow ελεγκτής ανοιχτού κώδικα και ο οποίος διαθέτει πλήθος από έτοιμα εξαρτήματα (components), εκ των οποίων ένα από αυτά τροποποιήθηκε και υλοποιήθηκε ο ανιχνευτής κατανεμημένων επιθέσεων άρνησης υπηρεσίας (DDoS) και το υβριδικό honeypot. Το περιβάλλον στο οποίο πραγματοποιήθηκαν τα πειράματα, είναι το

mininet το οποίο αποτελεί ένα λογισμικό εξομοίωσης για να υλοποιήσει κάποιος γρήγορα εικονικά δίκτυα και υποστηρίζει OpenFlow μεταγωγείς.

Η παρούσα διπλωματική εργασία μπορεί να διαχωριστεί σε δύο τμήματα όπου το καθένα απαρτίζεται από 2 επιμέρους κεφάλαια.

1ο τμήμα - θεωρητικό

- Ανάλυση του πρωτοκόλλου OpenFlow καθώς επίσης και των διαφορετικών εκδόσεών του και των κυριότερων προσθηκών - αλλαγών. Βασική παρουσίαση OpenFlow ελεγκτών ανοιχτού κώδικα με εκτενή ανάλυση των χαρακτηριστικών του ελεγκτή POX. Έγινε ανάλυση 3 βασικών εξαρτημάτων του και της λειτουργίας που επιτελούν.

2ο τμήμα - πρακτικό

- Σύντομη περιγραφή του περιβάλλοντος εξομοίωσης mininet. Ανάλυση των honeypots και της χρηστικότητάς τους με επικέντρωση στα υβριδικά honeypots και ιδιαίτερα στο honeybrid. Επέκταση του l2_learning switch προκειμένου να επιτελεί λειτουργίες υβριδικού honeypot και παρουσίαση των αποτελεσμάτων. Πραγματοποιείται προσομοίωση τόσο για κίνηση ICMP όσο και UDP με την εγκατάσταση DNS honeypot και στοιχειώδους DNS server.

- Ανάλυση των κατανεμημένων επιθέσεων άρνησης υπηρεσίας (DDoS) και της μεθόδου αντιμετώπισης τους με την χρήση εντροπίας. Αναλύονται και συγκρίνονται τα αποτελέσματα δικτυακής κίνησης στο mininet με τη χρήση της εντροπίας Shannon και του Rényi με παραμέτρους $\alpha = 2$ και $\alpha = 0.5$ για να προκύψουν χρήσιμα συμπεράσματα. Υπολογισμός μέγιστης εντροπίας του δικτύου. Ορισμός της μέγιστης εντροπίας ως ορίου για επιθέσεις DDoS και αποστολή με email των πιθανών IP διευθύνσεων της επίθεσης στο διαχειριστή δικτύου.

2

Ευφυή Προγραμματιζόμενα Δίκτυα - OpenFlow

Η σύγχρονη εποχή έχει αναδείξει νέες ανάγκες αναφορικά με τη δικτύωση των υπολογιστικών συστημάτων και αυτό οφείλεται στις ραγδαίες τεχνολογικές εξελίξεις, όπως την χρήση υπολογιστικών νεφών (Cloud Services), την ανάλυση μεγάλου συνόλου δεδομένων (Big Data), τις πολυμεσικές εφαρμογές υψηλής ανάλυσης που χρησιμοποιούν μεγάλο εύρος ζώνης καθώς επίσης και στο μεγάλο πλήθος φορητών συσκευών και ενσωματωμένων συστημάτων. Όλες αυτές οι νέες απαιτήσεις οδήγησαν στη δημιουργία νέων δικτυακών τεχνολογιών, των ευφυών προγραμματιζόμενων δικτύων (Software Defined Networks - SDN). Η τεχνολογία των από ευφυών προγραμματιζόμενων δικτύων αν και παρουσιάζει ιδιαίτερη άνθιση τα τελευταία χρόνια εντούτοις η λογική πάνω στην οποία βασίζονται στηρίχθηκε σε αρκετές πρώιμες τεχνολογίες οι οποίες είτε εμφανίζονταν σε άλλο τομέα επικοινωνιών είτε λόγω συγκυριών είχαν αποτύχει εμπορικά.

2.1 Εξέλιξη και υποστηρικτικές τεχνολογίες

Κεντρικοποιημένος έλεγχος του δικτύου

Η AT&T το 1981 ξεχώρισε το κανάλι δεδομένων (data plane) και το κομμάτι ελέγχου (signaling - control plane) που αναπτύχθηκε για το τηλεφωνικό δίκτυο. Έτσι έχουμε πλέον ένα κεντρικό σημείο ελέγχου του δικτύου (NCP - network control point) το οποίο μπορεί να επικοινωνεί με μια βάση δεδομένων προκειμένου να προσφέρει καινούριες υπηρεσίες αλλά και υπηρεσίες ανάλογα με τις ανάγκες των πελατών (on demand). Το πλεονέκτημα της σηματοδοσίας εκτός καναλιού (out of band signaling) σε σχέση με τη σηματοδοσία εντός καναλιού (in band signaling) που χρησιμοποιούνταν μέχρι τότε είναι ότι μπόρεσε να περιοριστεί ο χρόνος τον οποίον χρειάζονται συγκεκριμένα κυκλώματα να είναι ενεργά (network switching) και την ικανότητα να ξέρουμε αν ένα κύκλωμα είναι σε χρήση από κάποιον άλλον πριν το χρησιμοποιήσουμε για τη ζεύξη μας. Επίσης ένα άλλο μεγάλο πλεονέκτημα της σηματοδοσίας εκτός καναλιού είναι το πλήθος των υπηρεσιών οι οποίες μπορούν να αναπτυχθούν και να προσφέρουν νέες δυνατότητες στην υπάρχουσα δικτυακή υποδομή.

Ενεργά δίκτυα (Active Networks)

Είναι τα δίκτυα στα οποία οι κόμβοι είναι προγραμματισμένοι να εκτελούν κάποιες συγκεκριμένες ενέργειες στα πακέτα που διέρχονται μέσω αυτών. Δηλαδή ένας κόμβος μπορεί να προγραμματιστεί ώστε να διαχειρίζεται διαφορετικά τα πακέτα του εκάστοτε

χρήστη ή να διαχειρίζεται τα πακέτα πολυεκπομπής (multicasting) διαφορετικά από άλλα πακέτα. Στην ίδια κατηγορία ανήκει και η υλοποίηση "ενδιάμεσων κουτιών" (middle-boxes) όπως firewalls, proxies και γενικώς διαφόρων υπηρεσιών και εφαρμογών. Δεν εδραιώθηκαν σαν τεχνολογία λόγω του ότι την εποχή που αναπτύσσονταν δεν θα μπορούσαν να είχαν κάποια σχετική πρακτική εφαρμογή όπως σε data centers ή στο υπολογιστικό νέφος (cloud) τα οποία ήταν μεταγενέστερες τεχνολογίες. Επίσης το υλικό (hardware) ήταν πολύ ακριβό μιας και μέχρι τότε χρησιμοποιούσαν ASICs ενώ πλέον υπάρχουν TCAMs και FPGAs ευρέως. Ακόμα υπήρχαν θέματα αναφορικά με την ασφάλεια για την οποία χρειαζόταν ειδική γλώσσα προγραμματισμού όπως επίσης και το ότι δημιουργούσε πρόβλημα το γεγονός ότι ο προγραμματιστής μπορούσε να ήταν ο οποιοσδήποτε τελικός χρήστης, σε αντίθεση με τα SDN στα οποία ο προγραμματιστής είναι ο διαχειριστής δικτύου.

Εικονικοποίηση δικτύων (Network Virtualization)

Είναι η ύπαρξη πολλαπλών λογικών δικτύων στο ίδιο φυσικό δίκτυο. Σκοπός της εικονικοποίησης δικτύων είναι ο διαμοιρασμός δικτυακών πόρων σε συστήματα και χρήστες με αποδοτικό, ελεγχόμενο και ασφαλή τρόπο. Τα εικονικά δίκτυα χωρίζονται σε δύο μεγάλες κατηγορίες, εσωτερικά και εξωτερικά.

Εξωτερικά εικονικά δίκτυα.

- **Virtual Local Area Networks (VLANS).** Ένα δίκτυο σε επίπεδο 2 του OSI (data link layer) μπορεί να χωριστεί σε πολλά διαφορετικά λογικά broadcast domains, ενώ από φυσική άποψη είναι το ίδιο broadcast domain. Τα VLANS ξεχωρίζουν μεταξύ τους γιατί μέσα στο πακέτο προστίθεται το VLAN ID και τα switches προωθούν τα πακέτα με βάση και τη mac διεύθυνση και το VLAN ID. Αφού τα VLANS βασίζονται σε διαφορετικά λογικά δίκτυα και όχι σε διαφορετικά φυσικά δίκτυα, είναι μια πρώτη μορφή εικονικοποίησης δικτύου.

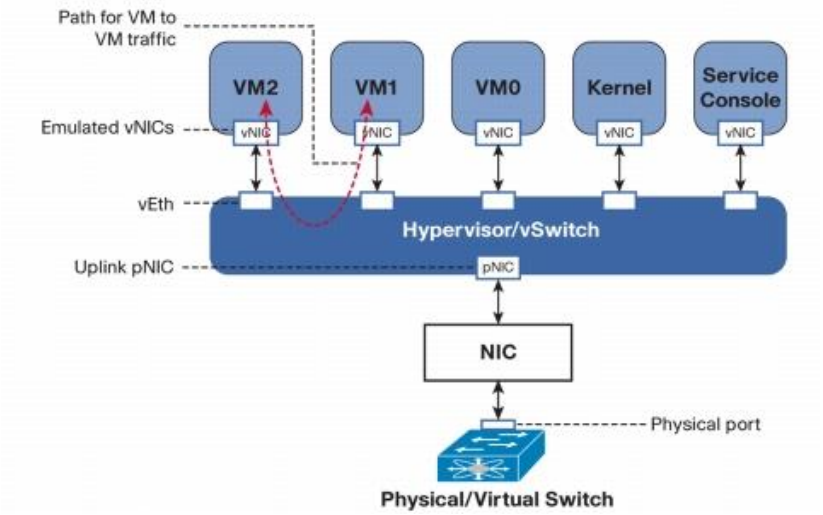
- **Virtual Private Networks (VPNS)**. Είναι ένα δεσμευμένο (dedicated) δίκτυο που συνδέει πολλαπλές τοποθεσίες πάνω από δημόσια δίκτυα επικοινωνίας όπως το ίντερνετ. Τα VPN μπορεί να ενώνουν απομακρυσμένες γεωγραφικά τοποθεσίες μια επιχείρησης (site-to-site VPN) ή ένας χρήστης να θέλει να συνδεθεί σε ένα ιδιωτικό δίκτυο (remote access VPN). Τα πρωτόκολλα tunneling που χρησιμοποιούνται συνήθως είναι τα IPsec, PPTP και L2TP.

- **Overlay Networks**. Αποτελούν λογικά δίκτυα τοποθετημένα πάνω σε ένα ή περισσότερα φυσικά δίκτυα. Το ίντερνετ ξεκίνησε σαν overlay δίκτυο πάνω από το τηλεφωνικό δίκτυο. Για παράδειγμα, κατανεμημένα συστήματα όπως τα δίκτυα peer-to-peer και οι client-server εφαρμογές είναι overlay networks γιατί χρησιμοποιούν ως βασικό δίκτυο το διαδίκτυο. Επίσης πλέον το τηλεφωνικό δίκτυο (VoIP) τείνει να γίνει overlay network πάνω από το διαδίκτυο. Πολλές overlay αρχιτεκτονικές έχουν προταθεί τα τελευταία χρόνια με σκοπό την επίλυση ποικίλων προβλημάτων όπως τη δυνατότητα multicasting, παροχής υπηρεσιών (Quality of Service - QoS), προστασία από επιθέσεις άρνησης υπηρεσίας (Denial of Service - DoS), δίκτυα διαμοιρασμού πολυμεσικού

περιεχομένου (Content Delivery Networks - CDNS) ακόμα και ως δοκιμαστικές πλατφόρμες (testbeds) με σκοπό τον σχεδιασμό και αξιολόγηση νέων αρχιτεκτονικών.

Εσωτερικά εικονικά δίκτυα.

Σε αυτή την περίπτωση σε ένα σύστημα πλέον έχουμε έναν hypervisor με εικονικές μηχανές (virtual machines) ή παρεμφερείς τεχνολογίες στις οποίες έχουμε ως βασικά στοιχεία εικονικές δικτυακές διεπαφές (virtual network interfaces) και εικονικούς μεταγωγείς (virtual switches) προκειμένου να επικοινωνήσουν όλα τα εικονικά μηχανήματα (virtual machines) μεταξύ τους.



Virtual switch, virtual NICs και virtual links για επικοινωνία μεταξύ των VMs, Πηγή[1]

Δεν πρέπει να συγχέεται η έννοια της εικονικοποίησης δικτύων με του SDN καθώς το SDN βασίζεται στο διαχωρισμό data plane και control plane. Το SDN μπορεί να χρησιμοποιήσει την εικονικοποίηση δικτύων για να έχει μεγαλύτερες δυνατότητες και περισσότερες υπηρεσίες. Η εικονικοποίηση δικτύων από μόνη της, δεν περιέχει την έννοια του SDN.

2.2 Εισαγωγή στο πρωτόκολλο OpenFlow

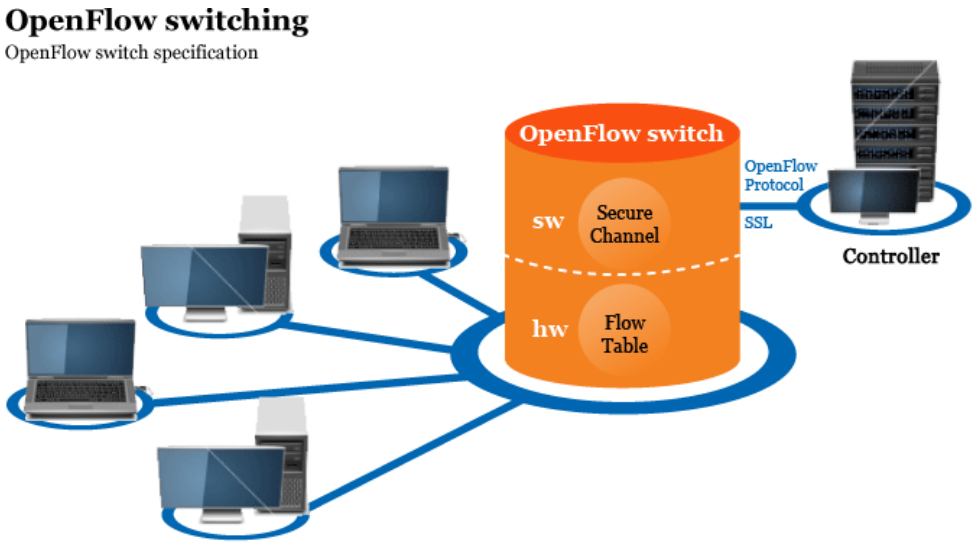
Το OpenFlow αποτελεί μία από τις πιο διαδεδομένες υλοποιήσεις και ένα πολύ χαρακτηριστικό παράδειγμα Software Defined Networking. Η ιδέα πίσω από αυτό ήταν να πάρουμε τις δυνατότητες του υπάρχοντος υλικού (hardware) των μεταγωγέων και να δώσουμε τη δυνατότητα σε ένα πρωτόκολλο ελέγχου να ελέγξει την συμπεριφορά τους. Στο OpenFlow ένας ελεγκτής επικοινωνεί με τον πίνακα ροής (Flow Table) του μεταγωγέα και εισάγει εγγραφές (Flow Table entries) που επηρεάζουν την προώθηση πακέτων από τον μεταγωγέα. Επειδή πολλοί μεταγωγείς ήδη χρησιμοποιούσαν πίνακες

ροής το μόνο που χρειαζόταν ήταν να πειστούν οι εταιρείες παραγωγής των μεταγωγέων να ανοίξουν τη διεπαφή (interface) των πινάκων ροής.

Ένα δίκτυο OpenFlow βασίζεται για το επίπεδο δεδομένων σε μεταγωγείς OpenFlow οι οποίοι μπορεί να είναι είτε σε επίπεδο υλικού είτε μπορεί να ναι εικονικοί (παράδειγμα Open vSwitch). Για το επίπεδο ελέγχου υπάρχουν ένας οι περισσότεροι ελεγκτές που επίσης υποστηρίζουν OpenFlow. Τέλος έχουμε μια ασφαλή σύνδεση ανάμεσα στους μεταγωγείς και τον ελεγκτή, στα πλαίσια του κεντρικοποιημένου ελέγχου του δικτύου που επιτάσσει το OpenFlow.

Τα σημαντικότερα πλεονεκτήματα του OpenFlow είναι ο κεντρικοποιημένος έλεγχος, η ευκολότερη διαχείριση δικτύου και οι νέες δυνατότητες στις δικτυακές συσκευές. Ειδικά σε μεγάλα δίκτυα όπως τα δίκτυα των datacenters, η κατανεμημένη διαχείριση του κάθε μεταγωγέα, θα αύξανε δραματικά τη δυσκολία και το κόστος διαχείρισής τους. Με το OpenFlow δίνεται η δυνατότητα στον διαχειριστή να παραμετροποιεί τους μεταγωγείς πολύ εύκολα και να έχει γενική εποπτεία του δικτύου. Επίσης στον κάθε ελεγκτή μπορούν να γραφτούν modules έτσι ώστε οι δικτυακές συσκευές να πραγματοποιούν παραπάνω ενέργειες από μια απλή αποδοχή και προώθηση δικτυακής κίνησης, όπως το να μπορούν να κάνουν εξισορρόπηση δικτυακής κίνησης (load balancing), προσαρμοστικό έλεγχο της δικτυακής κίνησης, να εντοπίζουν επιθέσεις άρνησης υπηρεσίας (DoS) ή ακόμη και να ελέγχουν τους μεταγωγείς κατά τέτοιο τρόπο ώστε να υπάρχει η ελάχιστη κατανάλωση ενέργειας. Δηλαδή επιτελούν παραπλήσια λειτουργία με τα middle-boxes των active networks.

OPENFLOW PROTOCOL



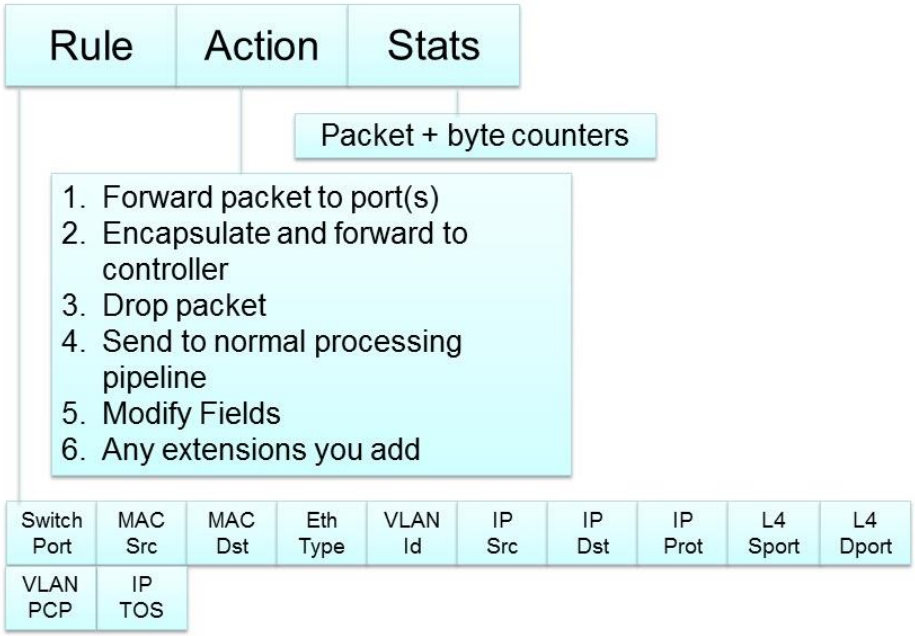
Switch - FlowTable, πηγή [2]

2.3 OpenFlow 1.0

Θα περιγράψουμε τα χαρακτηριστικά του πρωτοκόλλου OpenFlow. Θα αναφερθούμε αρχικά στην έκδοση 1.0 [4] και μετά στις πιο βασικές προσθήκες που υπήρξαν σε επόμενες εκδόσεις.
Ένας μεταγωγέας OpenFlow προωθεί τα πακέτα, αποτελείται από έναν πίνακα ροής και εγκαθιστά ένα ασφαλές κανάλι επικοινωνίας με τον ελεγκτή. Ο ελεγκτής διαχειρίζεται τον μεταγωγέα μέσω του ασφαλούς καναλιού επικοινωνίας.

Εγγραφές (Flow Entries)

Ένας πίνακας ροής περιέχει εγγραφές (Flow Entries) οι οποίες αποτελούνται από τα εξής βασικά: κανόνες, μετρητές και ένα σύνολο ενεργειών που θα εφαρμοστούν στην κάθε περίπτωση.



Συνοπτική παρουσίαση Flow Entry.

Κανόνες (Rules)

Το συγκεκριμένο πεδίο δείχνει τους κανόνες με τους οποίους συγκρίνεται κάθε εισερχόμενο πακέτο. Κάθε πεδίο περιέχει μια συγκεκριμένη τιμή (exact rule) ή την τιμή ANY (wildcard rule) που αντιστοιχίζεται με οποιαδήποτε τιμή πακέτου. Για τους κανόνες που περιέχουν όλα τα πεδία καθορισμένα, μπορούμε να έχουμε το πολύ έναν κανόνα για κάθε ενεργή ροή (active flow). Στους wildcard κανόνες, πολλοί κανόνες μπορούν να αντιστοιχίζουν σε ένα πακέτο, άρα πρέπει να βάλουμε προτεραιότητες (priorities) για να δούμε ποιος θα εφαρμοστεί κάθε φορά. Παρακάτω αναλύονται πιο συγκεκριμένα τα header fields.

Field	Bits	Applicable	Notes
Ingress Port	(εξαρτάται από την υλοποίηση)	Όλα τα πακέτα	Αριθμητική αναπαράσταση της εισερχόμενης θύρας ξεκινώντας από το 1
Ethernet Source Address	48	Όλα τα πακέτα στις ενεργοποιημένες πόρτες.	
Ethernet Destination Address	48	Όλα τα πακέτα στις ενεργοποιημένες πόρτες.	
Ethernet type	16	Όλα τα πακέτα στις ενεργοποιημένες πόρτες.	
VLAN id	12	Όλα τα πακέτα με Ethernet type 0x8100	
VLAN priority	3	Όλα τα πακέτα με Ethernet type 0x8100	VLAN PCP field
IP source address	32	Όλα τα πακέτα IP και ARP	Υποστηρίζονται υποδίκτυα
IP destination address	32	Όλα τα πακέτα IP και ARP	Υποστηρίζονται υποδίκτυα
IP protocol	8	IP and ARP	Χρήση των 8 χαμηλότερων bit του ARP opcode
IP ToS bits	6	IP and ARP	Χαρακτηρισμός ως 8μπιτη ποσότητα και τοποθετώντας το ToS στα υψηλότερα 6 bit.
Transport source port/ICMP Type	16	TCP,UDP,ICMP	Μόνο τα 8 χαμηλότερα bit χρησιμοποιούνται για ICMP Type
Transport destination port/ICMP code	16	TCP,UDP,ICMP	Μόνο τα 8 χαμηλότερα bit χρησιμοποιούνται για ICMP Code

Μετρητές - Counters

Οι μετρητές διατηρούνται και ανανεώνονται για κάθε πίνακα ροής, εγγραφή, θύρα του μεταγωγέα και ουρά και οι οποίοι παρατίθενται παρακάτω.

Counter	Bits
Per Table	
Active Entries	32
Packet Lookups	64
Packet Matches	64
Per Flow	
Received Packets	64
Received Bytes	64
Duration(seconds)	32
Duration(nanoseconds)	32
Per Port	
Received Packets	64
Transmitted Packets	64
Received Bytes	64
Transmitted Bytes	64
Receive Drops	64
Transmit Drops	64
Receive Errors	64
Transmit Errors	64
Receive Frame Alignment Errors	64
Receive Overrun Errors	64
Receive CRC Errors	64
Collisions	64
Per Queue	
Transmit Packets	64
Transmit Bytes	64
Transmit Overrun Errors	64

Ενέργειες (Actions)

Κάθε εγγραφή σχετίζεται με καμία, μία ή περισσότερες ενέργειες που καθορίζουν πώς ο μεταγωγέας θα χειριστεί τα πακέτα που γίνονται αντιστοιχία. Οι λίστες ενεργειών για τις εγγραφές πρέπει να επεξεργαστούν στη σειρά που έχουν καθοριστεί. Ένας μεταγωγέας μπορεί να απορρίψει μια εγγραφή εάν δεν μπορεί να επεξεργαστεί την λίστα ενεργειών με την σειρά που έχει καθοριστεί και σε αυτή την περίπτωση θα πρέπει να επιστρέψει **OFPET_FLOW_MOD_FAILED**. Ένας μεταγωγέας δεν είναι υποχρεωτικό να υποστηρίζει όλα τα ήδη ενεργειών αλλά μόνο αυτές που είναι υποχρεωτικές (required actions) και περιγράφονται πιο κάτω. Όταν συνδέεται με έναν ελεγκτή, ο μεταγωγέας δείχνει ποιες από τις προαιρετικές ενέργειες (optional actions) υποστηρίζει.

Υπάρχουν δύο ήδη OpenFlow μεταγωγέων. Οι OpenFlow-only και οι OpenFlow-enabled. Οι OpenFlow-only μεταγωγείς υποστηρίζουν μόνο τις ενέργειες παρακάτω, ενώ οι OpenFlow-enabled μεταγωγείς, δρομολογητές και access points μπορούν να υποστηρίζουν την ενέργεια NORMAL. Οποιοδήποτε από τα δύο είδη μεταγωγέων μπορεί να υποστηρίζει την ενέργεια FLOOD.

- **Υποχρεωτική ενέργεια (Required Action): Forward.**

Οι OpenFlow μεταγωγείς πρέπει να υποστηρίζουν την προώθηση των πακέτων σε φυσικές πόρτες και στις επόμενες εικονικές:
ALL: Στέλνει το πακέτο έξω από όλες τις διεπαφές του μεταγωγέα, μην περιλαμβάνοντας την εισερχόμενη.
CONTROLLER: Ενθυλακώνει το πακέτο και το προωθεί στον ελεγκτή.
LOCAL: Στέλνει το πακέτο στην τοπική δικτυακή στοίβα του μεταγωγέα.
TABLE: Πραγματοποιεί κάποια ενέργεια στον πίνακα ροής. Μόνο για packet-out μηνύματα.
IN_PORT: Στέλνει το πακέτο από την πόρτα που εισήχθη.

- **Προαιρετική ενέργεια (Optional Action): Forward**

Προαιρετικά ο μεταγωγέας μπορεί να υποστηρίζει την προώθηση πακέτων και στις ακόλουθες εικονικές πόρτες.
NORMAL: Επεξεργάζεται το πακέτο χρησιμοποιώντας τη κλασική προώθηση που υποστηρίζεται από τον μεταγωγέα.
FLOOD: Προώθηση του πακέτου από όλες τις πόρτες του μεταγωγέα, εκτός από αυτήν που εισήχθη, με βάση το ελάχιστο συνεκτικό δέντρο.

- **Υποχρεωτική ενέργεια (Required Action): Drop**

Μία εγγραφή χωρίς ενέργειες υποδηλώνει ότι όλα τα αντιστοιχιζόμενα πακέτα θα απορριφθούν.

- **Προαιρετική ενέργεια (Optional Action): Enqueue**.

Αυτή η ενέργεια προωθεί το πακέτο μέσω μιας ουράς αναμονής που λειτουργεί σε μία θύρα. Το είδος της προώθησης εξαρτάται από την παραμετροποίηση της ουράς και χρησιμοποιείται ως πολύ βασική τεχνική ποιότητας υπηρεσίας (Quality of Service-QoS)

- **Προαιρετική ενέργεια (Optional Action): Modify-Field.**

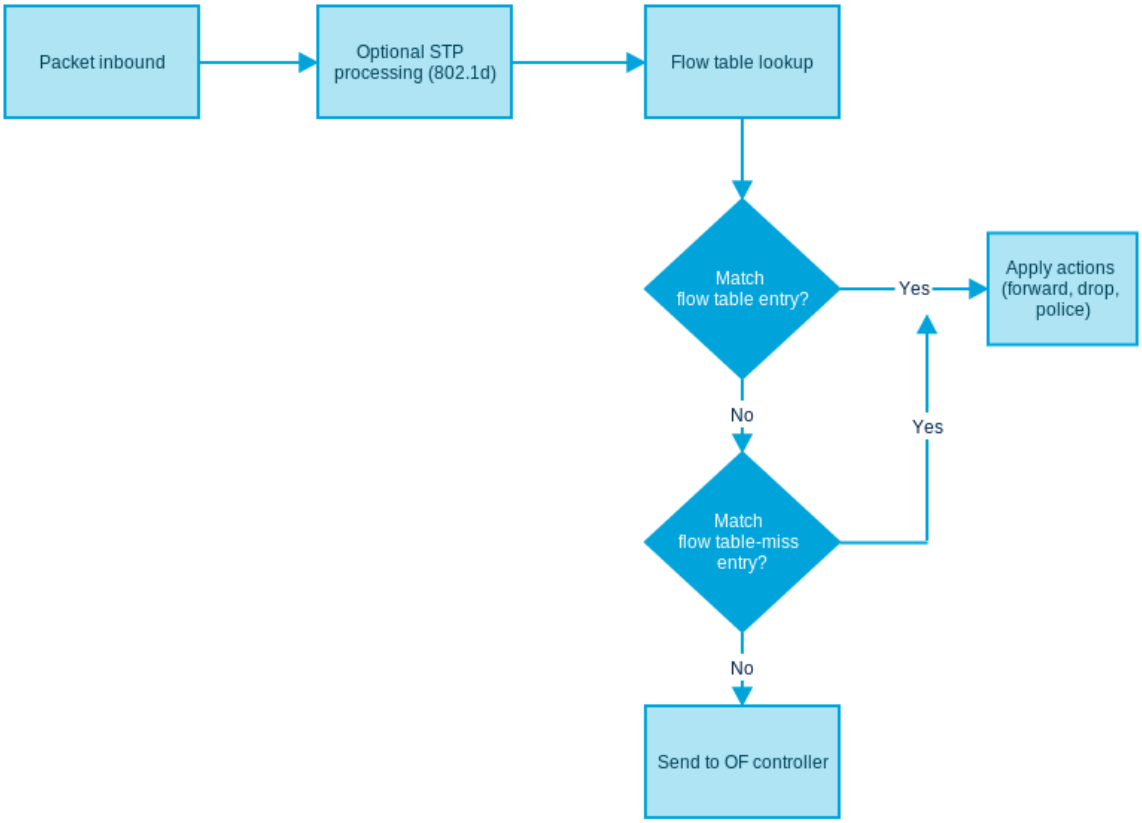
Παρότι δεν είναι υποχρεωτικό, το συγκεκριμένο σύνολο ενεργειών δίνει αυξημένες δυνατότητες και αυξάνει τη χρησιμότητα του μεταγωγέα OpenFlow. Για να διευκολυνθεί η σύνδεση με τα υπάρχοντα δίκτυα, προτείνεται οι μεταγωγείς να υποστηρίζουν τροποποίηση των επικεφαλίδων VLAN.

Action	Associated Data	Description
Set VLAN ID	12 bits	Εάν δεν υπάρχει VLAN, προστίθεται μια νέα κεφαλίδα με το VLAN ID του ορίσματος και

		προτεραιότητα 0. Εάν υπάρχει επικεφαλίδα VLAN, αντικαθίσταται το VLAN ID με αυτό του ορίσματος.
Set VLAN priority	3 bits	Εάν δεν υπάρχει VLAN, προστίθεται μια νέα κεφαλίδα με το VLAN ID του ορίσματος και προτεραιότητα 0. Εάν υπάρχει επικεφαλίδα VLAN, αντικαθίσταται το VLAN ID με αυτό του ορίσματος.
Strip VLAN header	-	Αφαίρεση της επικεφαλίδας VLAN
Modify Ethernet Source MAC address	48 bits για την αντικατάσταση της MAC διεύθυνσης πηγής	Αντικατάσταση της MAC διεύθυνσης πηγής με νέα.
Modify Ethernet Destination MAC address	48 bits για την αντικατάσταση της MAC διεύθυνσης προορισμού	Αντικατάσταση της MAC διεύθυνσης προορισμού με νέα.
Modify IPv4 source address	32 bits για την αντικατάσταση της ipv4 διεύθυνσης πηγής	Αντικατάσταση της υπάρχουσας IP πηγής και ανανέωσης του IP checksum (και του TCP, UDP αν γίνεται).
Modify IPV4 destination address	32 bits για την αντικατάσταση της ipv4 διεύθυνσης προορισμού	Αντικατάσταση της υπάρχουσας IP προορισμού και ανανέωσης του IP checksum (και του tcp, udp αν γίνεται).
Modify IPv4 ToS bits	6 bits για την αντικατάσταση του πεδίου ToS	Αντικατάσταση της υπάρχουσας τιμής του IP ToS. Αυτή η ενέργεια μπορεί να γίνει μόνο σε IPv4 πακέτα.
Modify transport source port	16 bits για την αντικατάσταση της υπάρχουσας πόρτας πηγής TCP/UDP	Αντικατάσταση της υπάρχουσας TCP/UDP πόρτας πηγής με νέα τιμή και ανανέωση του tcp/udp checksum.
Modify transport destination port	16 bits για την αντικατάσταση της υπάρχουσας πόρτας προορισμού TCP/UDP	Αντικατάσταση της υπάρχουσας TCP/UDP πόρτας προορισμού με νέα τιμή και ανανέωση του tcp/udp checksum.

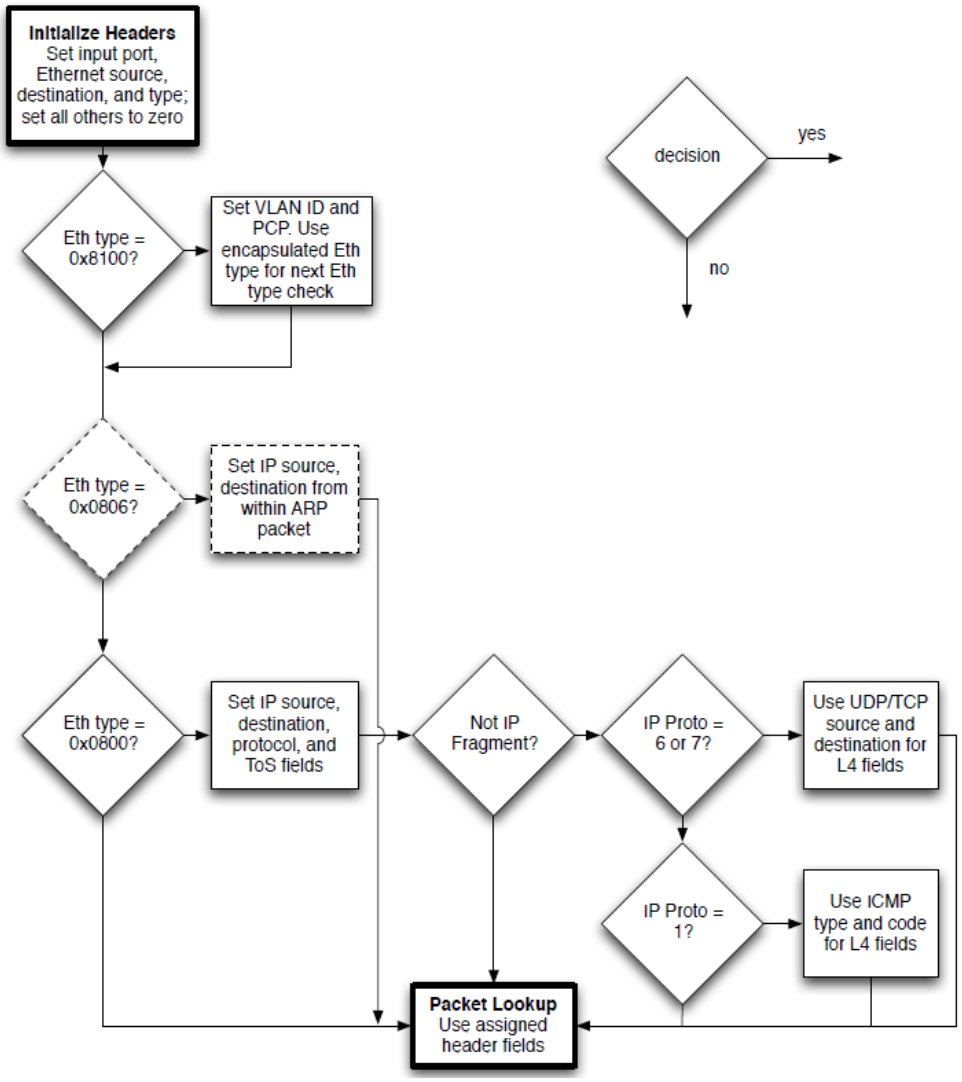
Αντιστοίχιση (Matching)

Ο μηχανισμός προώθησης πακέτων με OpenFlow απεικονίζεται στο επόμενο σχήμα. Αρχικά πραγματοποιείται προαιρετική επεξεργασία για το πρωτόκολλο 802.1d Spanning Tree Protocol. Έπειτα γίνεται σάρωση των επικεφαλίδων του πακέτου και συγκρίνονται με τους κανόνες στον πίνακα ροής. Αν βρεθεί ένας κανόνας του οποίου τα πεδία ταιριάζουν με του πακέτου, εφαρμόζεται αυτός ενώ αν βρεθούν περισσότεροι από ένας, εφαρμόζεται αυτός με την μεγαλύτερη προτεραιότητα. Ο μεταγωγέας επίσης ενημερώνει τους μετρητές. Αν δεν υπάρχει καταχώρηση η οποία να ταιριάζει στο πακέτο τότε ο μεταγωγέας στέλνει ένα packet-in μήνυμα στον ελεγκτή με ενσωματωμένο είτε ολόκληρο το πακέτο, είτε τα πρώτα bytes του αν υπάρχει η δυνατότητα αποθήκευσης του πακέτου στον μεταγωγέα. Ο ελεγκτής λαμβάνοντας το packet-in μήνυμα εγκαθιστά κανόνες στον μεταγωγέα είτε αποστέλλοντας τον κωδικό θέσης όπου αποθηκεύτηκε το πακέτο μαζί με τις ενέργειες που πρέπει να εφαρμοστούν, είτε με απλό μήνυμα Packet-Out.



Αντιστοίχιση στο OpenFlow 1.0, πηγή [3]

Η διαδικασία με την οποία τα πεδία της επικεφαλίδας χρησιμοποιούνται προκειμένου να βρεθεί κάποια αντιστοίχιση στον πίνακα ροής περιγράφεται στο επόμενο σχήμα, πηγή[4].



- Οι κανόνες για το ingress port, συγκρίνονται με την φυσική πόρτα από την οποία εισήλθε το πακέτο.
- Οι επικεφαλίδες Ethernet χρησιμοποιούνται για όλα τα πακέτα.
- Εάν το πακέτο είναι VLAN (Ethernet Type 0x8100) το VLAN ID και PCP χρησιμοποιούνται στην αναζήτηση.
- (προαιρετικό)Για τα ARP πακέτα (Ethernet Type 0x806) η αναζήτηση μπορεί να περιλαμβάνει τις διευθύνσεις IP πηγής και προορισμού.
- Για τα πακέτα IP (Ethernet Type 0x0800), τα πεδία αναζήτησης περιλαμβάνουν και αυτά της επικεφαλίδας IP.
- Για τα πακέτα IP τα οποία είναι TCP ή UDP (IP πρωτόκολλο 6 ή 17) η αναζήτηση περιλαμβάνει και τις πόρτες του επιπέδου μεταφοράς.
- Για τα IP πακέτα που είναι ICMP (IP protocol 1) η αναζήτηση περιλαμβάνει και τα πεδία Type και Code.
- Για τα πακέτα που έχουν υποστεί θρυμματισμό οι πόρτες του επιπέδου μεταφοράς τίθενται στην τιμή 0 για την αναζήτηση.

Ανταλλαγή μηνυμάτων στο OpenFlow

Όπως αναφέρθηκε, ο ασφαλής δίαυλος επικοινωνίας είναι η διεπαφή που συνδέει κάθε OpenFlow μεταγωγέα με τον ελεγκτή. Μέσα από αυτή τη διεπαφή, ο ελεγκτής ρυθμίζει και διαχειρίζεται τον μεταγωγέα, δέχεται γεγονότα (events) από αυτόν και στέλνει πακέτα μέσω αυτού. Ο μεταγωγέας πρέπει να αρχικοποιήσει την επικοινωνία μέσω του ασφαλούς καναλιού (Secure Channel), ξεκινώντας μια σύνοδο (session) προς μια IP διεύθυνση και port. Η επικοινωνία αυτή δεν υπόκειται σε διαδικασία της αντιστοίχισης με τις εγγραφές του μεταγωγέα. Έτσι, ένας μεταγωγέας πρέπει να είναι σε θέση να ξεχωρίζει την υπόλοιπη δικτυακή κίνηση προκειμένου να ξεκινήσει την αντιπαραβολή με τον πίνακα ροής. Ο δίαυλος υλοποιείται μέσω μιας συνόδου Transport Layer Security, προσφέροντας κρυπτογραφημένη επικοινωνία. Όπως αναφέρθηκε, η σύνοδος εγκαθιδρύεται από τον μεταγωγέα και πραγματοποιείται η σύνδεση στην TCP port που αναμένει ο ελεγκτής (συνήθως στην 6633). Εν συνεχεία, η κάθε πλευρά μπαίνει στη διαδικασία πιστοποίησης του εταίρου συμβαλλόμενου, μέσω της ανταλλαγής πιστοποιητικών υπογεγραμμένων από μια αρχή πιστοποίησης.

Όσον αφορά τα μηνύματα που ανταλλάσσονται, το πρωτόκολλο υποστηρίζει τρεις διαφορετικούς τύπους μηνυμάτων, όπου κάθε τύπος έχει υποκατηγορίες που θα αναλυθούν περαιτέρω:

- **Controller – to – switch**: Στέλνονται από τον ελεγκτή και χρησιμοποιούνται για να διαχειριστούν ή επιθεωρήσουν τον μεταγωγέα κατά άμεσο τρόπο.
- **Asynchronous**: Ξεκινούν από τον μεταγωγέα με στόχο την ενημέρωση του ελεγκτή είτε για γεγονότα του δικτύου, είτε για αλλαγές στην κατάσταση του δικτύου.
- **Symmetric**: Προέρχονται είτε από τον ελεγκτή είτε από τον μεταγωγέα, χωρίς να τα ζητήσει συγκεκριμένα η άλλη πλευρά.

Controller-to-switch

Τα μηνύματα αποστέλλονται από τον ελεγκτή και μπορεί να περιμένουν ή όχι απάντηση από τον μεταγωγέα:

- **Features:** Κατά την εγκαθίδρυση της συνόδου, ο ελεγκτής στέλνει στον μεταγωγέα ένα μήνυμα που ζητεί τα χαρακτηριστικά που υποστηρίζει (features request), στο οποίο απαντάει εν συνεχεία ο μεταγωγέας παρέχοντας την λίστα με αυτά (features reply).
- **Configuration:** Ο ελεγκτής έχει την δυνατότητα να προβεί σε ρύθμιση των παραμέτρων του μεταγωγέα, ή να ζητήσει πληροφορίες σχετικά με τις ήδη υπάρχουσες ρυθμίσεις.

- **Modify – State:** Χρησιμοποιείται για την διαχείριση της κατάστασης των μεταγωγέων. Κύριος στόχος του είναι η προσθήκη, διαγραφή ή τροποποίηση των εγγραφών του πίνακα ροής, καθώς και η ρύθμιση ιδιοτήτων των θυρών.
- **Read – State:** Χρησιμοποιείται για την συλλογή στατιστικών δεδομένων από τους πίνακες ροής, τις θύρες και συγκεκριμένες εγγραφές.
- **Send – Packet:** Χρησιμοποιείται για την αποστολή πακέτων έξω από μια συγκεκριμένη θύρα του μεταγωγέα, χωρίς αναγκαστικά να εγκαθίσταται κάποιος κανόνας στον πίνακα του μεταγωγέα.
- **Barrier:** Χρησιμοποιείται είτε για να βεβαιωθεί πως οι εξαρτήσεις προηγούμενων μηνυμάτων έχουν ολοκληρωθεί πριν από την επεξεργασία επόμενων, είτε για την λήψη ειδοποιήσεων για ολοκληρωμένες ενέργειες.

Asynchronous

Τα μηνύματα αποστέλλονται από τους μεταγωγείς χωρίς να ζητά ρητά ο ελεγκτής. Κυρίως χωρίζονται στους ακόλουθους τύπους:

- **Packet – In:** Όλα τα πακέτα τα οποία δεν μπορούν να αντιστοιχηθούν σε κάποια εγγραφή, προκαλούν την αποστολή ενός γεγονότος Packet-in. Εάν ο μεταγωγέας διαθέτει επαρκή διαθέσιμη μνήμη στον buffer ώστε να κρατήσει προσωρινά το πακέτο, τότε το μήνυμα περιλαμβάνει ένα τμήμα του πακέτου (από προεπιλογή τα πρώτα 128 bytes) καθώς και μια αναγνωριστική τιμή στον buffer. Σε περίπτωση μη υποστήριξης της προσωρινής αποθήκευσης ή μη επαρκούς μνήμης, στέλνεται ολόκληρο το πακέτο.
- **Flow – Removed:** Κατά την προσθήκη μιας εγγραφής στον πίνακα ροής του μεταγωγέα από τον ελεγκτή, υπάρχουν δυο πεδία σχετικά με την αυτόματη αφαίρεση της. Όσον αφορά το πρώτο, μετά από ποιο χρονικό διάστημα αδράνειας (soft timeout), δηλαδή καμίας αντιστοίχιση πακέτων με αυτή, η εγγραφή αφαιρείται. Το δεύτερο προσδιορίζει μετά από ποιο χρονικό διάστημα θα αφαιρεθεί χωρίς να χρησιμοποιηθεί κάποιο άλλο κριτήριο (hard timeout). Στην εγγραφή επίσης υπάρχει μια σημαία (flag) η οποία ορίζει εάν η αφαίρεση πρέπει να προκαλέσει την αποστολή μηνύματος Flow–Removed στον controller.
- **Port – status:** Είναι αναμενόμενο από τον μεταγωγέα να στέλνει τέτοια μηνύματα κατά την αλλαγή της κατάστασης σε μία θύρα. Παραδείγματος χάριν λόγω ενεργοποίησης/απενεργοποίησης από έναν χρήστη, ή αλλαγών όπως αυτές ορίζονται στο spanning tree protocol.
- **Error:** Χρησιμοποιούνται για την ενημέρωση του ελεγκτή για τυχόν σφάλματα που αντιμετωπίσθηκαν.

Symmetric

Μηνύματα που στέλνονται και από τον μεταγωγέα και από τον ελεγκτή.

- **Hello:** Ανταλλάσσονται ανάμεσα στον μεταγωγέα και τον ελεγκτή κατά την εκκίνηση της σύνδεσης τους.
- **Echo:** Μηνύματα echo request που στέλνονται είτε από τον μεταγωγέα είτε από τον ελεγκτή και περιμένουν ένα echo reply σε απάντηση. Χρησιμοποιούνται σαν δείκτες για latency/bandwidth, όπως επίσης και για επιβεβαίωση της ακεραιότητας της σύνδεσης.
- **Vendor:** Αποτελούν ένα τρόπο υλοποίησης περαιτέρω χρηστικότητας (αναλόγως τον vendor) εντός του OpenFlow πρωτοκόλλου.

Επικοινωνία μεταγωγέα – ελεγκτή

Όταν δημιουργείται μία σύνδεση OpenFlow, κάθε άκρο της επικοινωνίας πρέπει να στείλει ένα μήνυμα OFPT_HELLO με το είδος της έκδοσης OpenFlow καθορισμένο στο υψηλότερο δυνατό που μπορεί να υποστηρίξει ο αποστολέας. Κατά την παραλαβή ο παραλήπτης μπορεί να υπολογίσει την έκδοση πρωτοκόλλου OpenFlow έτσι ώστε να είναι η μικρότερη δυνατή ανάμεσα σε αυτή που στάλθηκε και παραδόθηκε. Εάν η διαπραγματευόμενη έκδοση υποστηρίζεται από τον παραλήπτη, τότε η σύνδεση δημιουργείται. Αλλιώς ο παραλήπτης πρέπει να απαντήσει με μήνυμα OFTP_ERROR με το type field να είναι OFPT_HELLO_FAILED, το code field να είναι OFPHFC_COMPATIBLE, προαιρετικά μια συμβολοσειρά σε ASCII που να εξηγεί τη συνέβη και τέλος πρέπει να τερματιστεί η σύνδεση.

Διακοπή επικοινωνίας με τον ελεγκτή

Όταν ένας μεταγωγέας χάσει την επικοινωνία με τον ελεγκτή, είτε λόγω echo request time out, είτε λόγω TLS session timeout, είτε για κάποιον άλλο λόγο, πρέπει να προσπαθήσει επικοινωνία με έναν ή περισσότερους εφεδρικούς ελεγκτές. Εάν μετά από έναν συγκεκριμένο αριθμό προσπαθειών του μεταγωγέα να επικοινωνήσει με τον ελεγκτή αποτύχουν, ο μεταγωγέας μπαίνει σε «emergency mode» και κάνει reset σε όλες τις συνδέσεις TCP. Σε αυτή την κατάσταση λειτουργίας η διαδικασία ταιριάσματος πακέτων γίνεται με βάση τα emergency entries του πίνακα ροής (αυτά τα οποία έχουν σημειωθεί με emergency bit όταν προστίθενται στον μεταγωγέα). Οι άλλες εγγραφές διαγράφονται όταν ο μεταγωγέας μπαίνει σε emergency mode. Όταν ξανασυνδέεται ο μεταγωγέας στον ελεγκτή, τα emergency flow entries παραμένουν και ο ελεγκτής έχει την επιλογή να τα διαγράψει αν δεν τα χρειάζεται. Την πρώτη φορά που ο μεταγωγέας ενεργοποιείται είναι σε emergency mode και η παραμετροποίηση των προεπιλεγμένων εγγραφών είναι έξω από το πεδίο του πρωτοκόλλου OpenFlow.

Κρυπτογράφηση

Ο μεταγωγέας και ο ελεγκτής επικοινωνούν μέσω μιας σύνδεσης TLS. Η TLS σύνδεση δημιουργείται από τον μεταγωγέα κατά την εκκίνηση προς τον ελεγκτή, ο οποίος ακούει προεπιλεγμένα στην TCP πόρτα 6633.

2.4 OpenFlow 1.1.0

Κυριότερες αλλαγές-προσθήκες στο OpenFlow 1.1.0

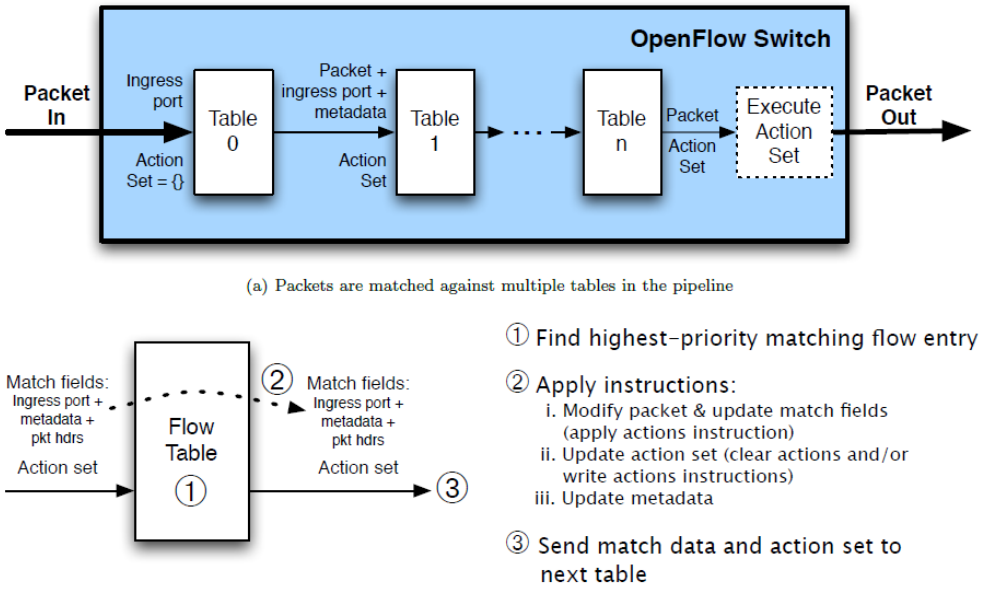
Πίνακας ομάδας (Group Table)

Πλέον στο OpenFlow 1.1.0 [5] ο μεταγωγέας αποτελείται από έναν πίνακα ομάδας (Group Table) και έναν ή περισσότερους πίνακες ροής (Flow Tables) και ένα ασφαλές κανάλι επικοινωνίας. Η αντιστοίχιση πακέτου ξεκινά με τον πρώτο πίνακα ροής και συνεχίζει στους επόμενους. Αν βρεθεί αντιστοίχιση σε κάποια εγγραφή του πίνακα ροής τότε εφαρμόζονται οι ενέργειές του. Αν δεν βρεθεί τότε το τι θα γίνει εξαρτάται από την παραμετροποίηση του μεταγωγέα: μπορεί το πακέτο να προωθηθεί στον ελεγκτή μέσω του καναλιού OpenFlow, να απορριφθεί ή να γίνει προσπάθεια να αντιστοιχηθεί με κάποια εγγραφή του επόμενου πίνακα ροής με την εντολή Goto. Οι πίνακες ροής του μεταγωγέα είναι αριθμημένοι σειριακά ξεκινώντας από το 0 και ένας πίνακας ροής μπορεί να στείλει ένα πακέτο σε πίνακα με νούμερο μεγαλύτερο από το δικό του. Προφανώς οι εγγραφές στον τελευταίο πίνακα δεν έχουν την εντολή Goto. Η σωλήνωση (pipeline) είναι η τεχνική η οποία χρησιμοποιείται προκειμένου τα πακέτα να περάσουν μεταξύ διαφορετικών πινάκων προκειμένου να αντιστοιχηθούν και επίσης δίνουν τη δυνατότητα να μεταφέρεται πληροφορία μεταξύ των διαφορετικών πινάκων με τη μορφή metadata.

Ο πίνακας ομάδας (Group Table) περιλαμβάνει τα εξής:

Group Identifier	Group Type	Counters	Action Buckets

- **Group Identifier:** Ένας μη προσημασμένος ακέραιος αριθμός μήκους 32 bits, ο οποίος προσδιορίζει το κάθε group με μοναδικό τρόπο.
- **Group Type:** Προσδιορίζει τον τύπο του group. Θα αναλυθούν αναλυτικότερα περαιτέρω**.**
- **Counters:** Μετρητές, οι οποίοι ανανεώνονται κάθε φορά που ένα πακέτο επεξεργάζεται από το αντίστοιχο group.
- **Action Buckets:** Μια διατεταγμένη λίστα, από δοχεία με ενέργειες (action buckets). Το κάθε δοχείο περιέχει ένα σύνολο ενεργειών που εκτελούνται κατά την επεξεργασία, καθώς και τις σχετικές με τις ενέργειες αυτές παραμέτρους.



Packet flow through the processing pipeline, πηγή [5]

Στη συνέχεια ορίζονται οι διαφορετικοί τύποι Groups:

- **all:** Εκτέλεση όλων των Action Buckets που ορίζονται στο αντίστοιχο πεδίο. Αυτό το group χρησιμοποιείται για προώθηση πακέτων, multicast ή broadcast. Το πακέτο πρακτικά αντιγράφεται για κάθε bucket και τα αντίγραφα επεξεργάζονται από το αντίστοιχο bucket.
- **select:** Εκτέλεση ενός Action Bucket. Τα πακέτα κατευθύνονται σε ένα μοναδικό bucket του Group, βάση ενός αλγορίθμου επιλογής που υπολογίζεται από τον ίδιο τον μεταγωγέα. Ενδεικτικά παραδείγματα είναι η χρήση συνάρτησης κατακερματισμού σε στοιχεία του OpenFlow match που προσδιορίζεται από τον χρήστη, ή απλή εφαρμογή του αλγορίθμου round robin. Σημειώνεται πως όλες οι ρυθμίσεις είναι εξωτερικές ως προς το πρωτόκολλο OpenFlow.
- **indirect:** Εκτέλεση του μοναδικού Action Bucket που είναι ορισμένο σε αυτό το Group. Το παραπάνω επιτρέπει σε πολλαπλές εγγραφές να χρησιμοποιούν ένα κοινό αναγνωριστικό (Group Identifier), για την εκτέλεση ενός Action Set (παράδειγμα επόμενο βήμα σε προώθηση IP). Αυτός ο τύπος είναι στην πράξη πανομοιότυπος με έναν all, στον οποίο είναι ορισμένο μόνο ένα Action bucket.
- **fast failover:** Εκτέλεση του πρώτου ενεργού Bucket. Κάθε action bucket είναι συνδεδεμένο με ένα συγκεκριμένο port ή και group που ελέγχει την κατάσταση του bucket. Αυτός ο τύπος επιτρέπει στον μεταγωγέα να αλλάζει τον τρόπο προώθησης χωρίς να απαιτείται κάποια ενέργεια από τον ελεγκτή.

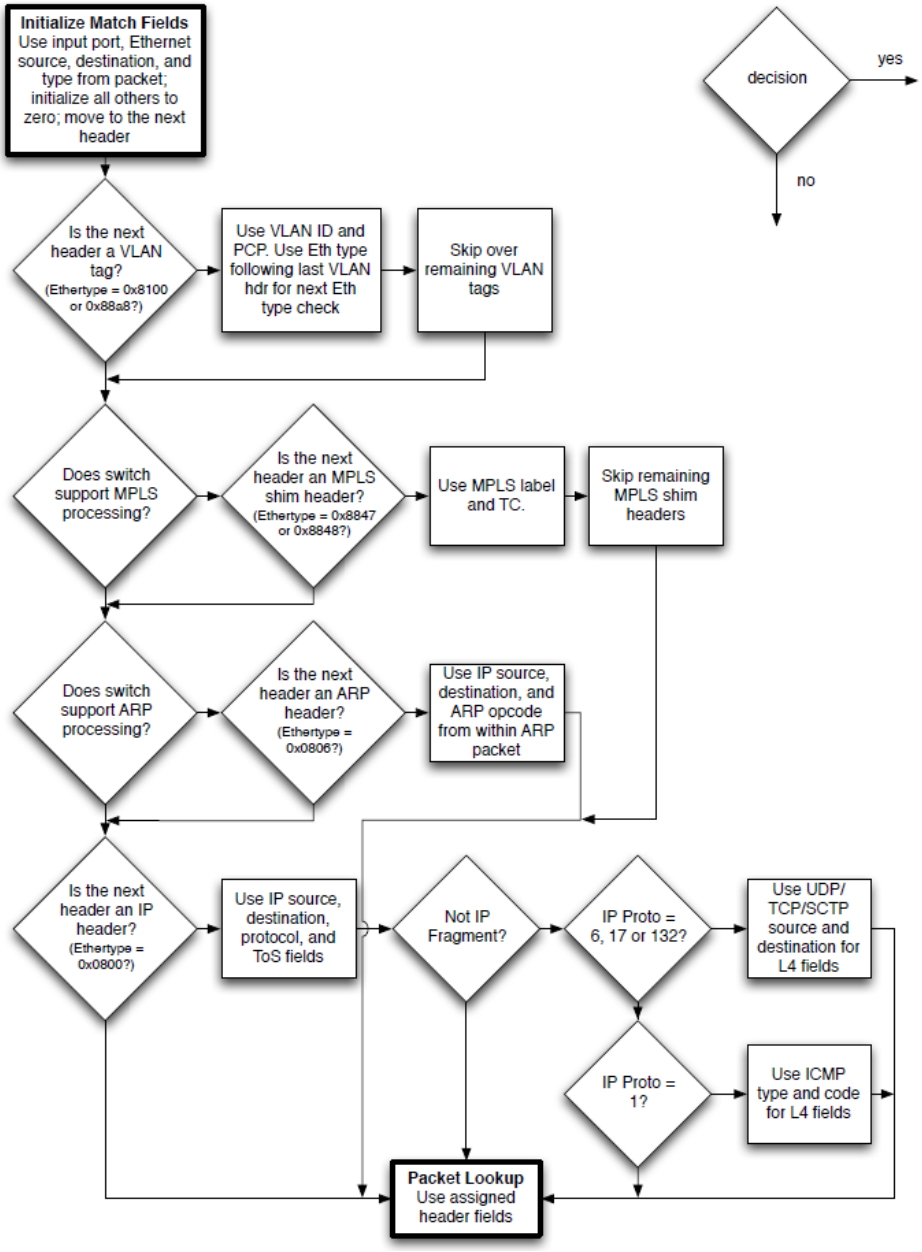
Εγγραφές (Flow Entries)

Match fields	Counters	Instructions

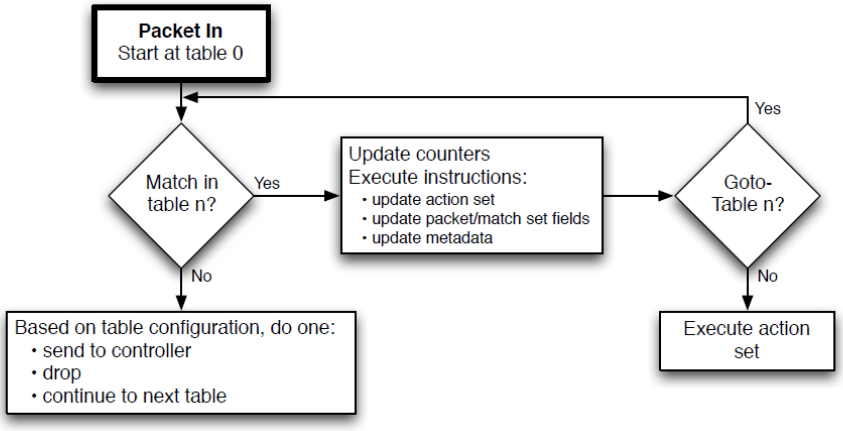
Πλέον με τις νέες εντολές μπορούν να γίνουν αλλαγές στο action set ή στη διαδικασία σωλήνωσης.

Αντιστοίχιση (Matching)

Πλέον τα πεδία στα οποία αντιστοιχίζεται ένα πακέτο είναι τα γνωστά 12 από την προηγούμενη έκδοση, αλλά προστίθενται πλέον και για metadata, MPLS label και MPLS traffic class και έτσι από 12 γίνονται 15.



Διάγραμμα ροής για αντιστοίχιση σε ένα Flow Table, πηγή [5].



Αντιστοίχιση με πολλαπλά Flow Tables, πηγή [5].

Μετρητές - Counters

Ίδιοι με του OpenFlow 1.0.0, με επιπρόσθετους τους επόμενους:

Counter	Bits
Per Group	
Reference Count (flow entries)	32
Packet Count	64
Byte Count	64
Per Bucket	
Packet Count	64
Byte Count	64

Ενέργειες (Actions)

- **Υποχρεωτική ενέργεια (required action) – Output**: Ίδια με την αντίστοιχη Forward του OpenFlow 1.0.0
- **Προαιρετική ενέργεια (optional action) – Output**: Ίδια με την αντίστοιχη προαιρετική Forward του OpenFlow 1.0.0.
- **Προαιρετική ενέργεια – Set-Queue**: Ίδια με την Enqueue του OpenFlow 1.0.0
- **Υποχρεωτική ενέργεια – Drop**: Ίδια με την Drop του OpenFlow 1.0.0
- **Υποχρεωτική ενέργεια – Group**: Επεξεργασία του πακέτου μέσω του συγκεκριμένου Group. Το πώς θα ερμηνευτεί αυτό, εξαρτάται από τον τύπο του Group.

- **Προαιρετική ενέργεια – Push-Tag/Pop-Tag**: Οι μεταγωγείς μπορεί να υποστηρίζουν τη δυνατότητα να βάζουν και να βγάζουν ετικέτες VLAN και MPLS. (Push/Pop MPLS headers και Push/Pop VLAN headers.)

Η σειρά των επικεφαλίδων/ετικετών είναι η εξής:

Ethernet	VLAN	MPLS	ARP/IP	TCP/UDP/SCTP(IP-only)

Προαιρετικές ενέργειες: Set-Filed. Ίδια με τη Modify Field του 1.0.0 με την προσθήκη και πρόσθετων ενεργειών.

- Για MPLS. Set MPLS label. Set MPLS traffic class. Set MPLS TTL. Decrement MPLS TTL.
- Set IPV4 ECN bits.
- Για TTL (εκτός MPLS). Set IPv4 TTL. Decrement IPv4 TTL. Copy TTL outwards. Copy TTL inwards.

Διακοπή επικοινωνίας με τον ελεγκτή

Πλέον εδώ δίνεται η δυνατότητα όταν χαθεί η επικοινωνία με τον ελεγκτή ο μεταγωγέας να μπορεί να μπει σε δύο διαφορετικές καταστάσεις λειτουργίας οι οποίες καθορίζονται από την εκάστοτε παραμετροποίησή του, την «fail secure mode» ή την «fail standalone mode». Στην «fail secure» λειτουργία η μόνη αλλαγή στη λειτουργία του μεταγωγέα είναι ότι τα πακέτα και τα μηνύματα που προορίζονται προς τον ελεγκτή από τον οποίο χάθηκε η επικοινωνία, απορρίπτονται. Στην «fail standalone» λειτουργία ο μεταγωγέας επεξεργάζεται τα πακέτα σαν να είναι κλασικός, μη OpenFlow enabled μεταγωγέας ή δρομολογητής.

2.5 OpenFlow 1.2.0

Προσθέτει την υποστήριξη για IPv6. Πλέον το OpenFlow 1.2.0 [6] κάνει αντιστοίχιση για IPv6 διεύθυνση αποστολέα, διεύθυνση προορισμού, αριθμό πρωτοκόλλου, traffic class, ICMPv6 type, ICMPv6 code, ICMPv6 neighbor discovery header fields και έχουμε IPv6 πίνακες ροής. Επιπλέον μπορεί κάποιος να προεκτείνει τις δυνατότητες αντιστοίχισης, προσθέτοντας και άλλα πεδία για ταίριασμα, της δομής type-length-value (TLV) και αυτή η δυνατότητα ονομάζεται επεκτάσιμο ταίριασμα του OpenFlow (OpenFlow Extensible Match - OXM). Επίσης δίνεται η δυνατότητα σε έναν μεταγωγέα να είναι συνδεδεμένος σε παραπάνω από έναν ελεγκτή. Ο μεταγωγέας ξεκινά τη σύνδεση και οι ελεγκτές αποδέχονται τις συνδέσεις. Ένας ελεγκτής θεωρείται master και είναι αυτός που προγραμματίζει τον μεταγωγέα και οι άλλοι slaves. Έτσι σε περίπτωση βλάβης, που ο master βγει εκτός σύνδεσης, μπορεί κάποιος άλλος ελεγκτής από τους slaves να γίνει master και έτσι να έχουμε αυξημένη αξιοπιστία. Επίσης στις ενέργειες, υπάρχει πλέον ξεχωριστή ενέργεια για να κάνεις κάποιος αλλαγή στο TTL των πακέτων (Change-TTL).

2.6 OpenFlow 1.3.0

Κυριότερες αλλαγές-προσθήκες στο OpenFlow 1.3.0

Εγγραφές αστοχίας πίνακα (Table-miss Flow Entries)

Στο OpenFlow 1.3.0 [7] ο πίνακας ροής πρέπει να υποστηρίζει μια εγγραφή αστοχίας πίνακα (Table-miss Flow Entry) για να επεξεργάζεται τις αστοχίες πακέτων στον πίνακα ροής. Οι εγγραφές αστοχίας πίνακα ροής καθορίζουν πώς θα επεξεργαστούν τα πακέτα σε περίπτωση που δεν υπάρξει καμιά αντιστοιχία στον συγκεκριμένο πίνακα και μπορούν για παράδειγμα να στείλουν τα πακέτα στον ελεγκτή, να τα απορρίψουν ή να τα στείλουν σε κάποιον άλλο πίνακα. Οι εγγραφές αυτές έχουν όλα τα πεδία ασυμπλήρωτα (wildcards) και έχουν την ελάχιστη προτεραιότητα (0). Επίσης συμπεριφέρονται σαν κλασικές εγγραφές: δεν υφίστανται εκ των προτέρων σε έναν πίνακα ροής και ο ελεγκτής μπορεί να τις προσθέσει ή να τις αφαιρέσει οποιαδήποτε στιγμή.

Μετρητές ροής (Meters)

Πλέον εισάγεται και ένας πίνακας με μετρητές ροής (Meter Table). Ένα τέτοιος πίνακας, απαρτίζεται από αντίστοιχες εγγραφές (Meter Entries) οι οποίες ορίζουν μετρητές για κάθε ροή, δίνοντας την δυνατότητα υλοποίησης απλών ή συνθετότερων ενεργειών διασφάλισης ποιότητας (Quality of Service), όπως για παράδειγμα ο περιορισμός της ροής κίνησης (rate limiting). Κάθε τέτοιος δείκτης μετρά το ρυθμό μετάδοσης πακέτων που έχει ανατεθεί σε αυτόν, επιτρέποντας τον έλεγχο της ταχύτητας. Επισημαίνεται πως οι μετρητές ροής, είναι συνδεδεμένοι απευθείας με εγγραφές, σε αντίθεση με τις ουρές (Queues) που είναι συνδεδεμένες με θύρες. Οποιαδήποτε εγγραφή μπορεί να ορίσει σύνδεση με κάποιο meter, μέσω του instruction set της. Επίσης αξίζει να σημειωθεί πως ένας μετρητής ροής ελέγχει την συνολική ροή, από όλες τις εγγραφές που είναι συνδεδεμένες με αυτόν. Όσον αφορά την χρήση πολλαπλών μετρητών στον ίδιο πίνακα ροής, κάτι τέτοιο είναι εφικτό αλλά όχι για διαφορετικούς μετρητές στις ίδιες εγγραφές. Ωστόσο μπορούν να εφαρμοσθούν πολλαπλοί μετρητές στις ίδιες ροές πακέτων σε διαφορετικούς διαδοχικούς πίνακες ροής.
Κάθε εγγραφή του πίνακα μετρητών απαρτίζεται από:

Meter Identifier	Meter Bands	Counters

- **Meter identifier:** Ένας μη προσημασμένος ακέραιος αριθμός μήκους 32 bits, ο οποίος προσδιορίζει μοναδικά την κάθε εγγραφή.

- **Meter bands:** Μία μη διατεταγμένη λίστα από μπάντες μετρητών (meter bands), όπου η καθεμία ορίζει το όριο της μπάντας και τον τρόπο επεξεργασίας των πακέτων που εμπίπτουν σε αυτές.
- **Counters:** Μετρητές (counters όχι meters), που ανανεώνονται όταν επεξεργάζονται πακέτα από τον αντίστοιχο μετρητή.

Μπάντες μετρητών ροής (Meter Bands)

Κάθε μετρητής μπορεί να έχει μια ή περισσότερες μπάντες. Σημειώνεται πως τα πακέτα επεξεργάζονται από μια μόνο μπάντα, βάση της τρέχουσας τιμής της ροής. Ο μετρητής εφαρμόζει την μπάντα με την υψηλότερη τιμή ροής πακέτων η οποία τιμή όμως είναι χαμηλότερη από την τρέχουσα ροή. Σε περίπτωση που δεν υπάρχει μπάντα με αυτές τις προϋποθέσεις, δεν εφαρμόζεται καμία.
Κάθε μπάντα έχει ως αναγνωριστικό την ροή στην οποία εντάσσεται και απαρτίζεται από:

Band Type	Rate	Counters	Type specific arguments

- **Band type**: Ορίζει τον τρόπο επεξεργασίας του πακέτου από αυτή τη μπάντα (drop, dscp remark).
- **Rate:** Χρησιμοποιείται από τον μετρητή για να επιλέξει την αντίστοιχη μπάντα, ορίζει το χαμηλότερο τρέχων ρυθμός μετάδοσης πάνω από τον οποίο μπορεί να εφαρμόζεται η αντίστοιχη μπάντα.
- **Counters:** Ανανεώνονται κατά την επεξεργασία ενός πακέτου από την μπάντα.
- **Type specific arguments**: Πρόσθετοι παράμετροι του band type.

Μετρητές (Counters)

Πλέον με βάση τα προηγούμενα, προστίθενται και μετρητές που αφορούν τους μετρητές και τις μπάντες, επιπρόσθετα από τις προηγούμενες εκδόσεις:

Counter	Bits
Per Meter	
Flow Count	32
Input Packet Count	64
Input Byte Count	64
Duration (seconds)	32
Duration (nanoseconds)	32
Per Meter Band	
In Band Packet Count	64
In Band Byte Count	64

Ενέργειες (Actions)

Πλέον έχουμε και τις επιπρόσθετες ενέργειες: Push-Tag/Pop-Tag οι οποίες μέχρι τώρα ήταν για VLAN και MPLS headers και πλέον προστίθενται και για PBB headers.

2.7 OpenFlow 1.4.0

Κυριότερες αλλαγές-προσθήκες στο OpenFlow 1.4.0

Δέσμες (bundles)

Στο OpenFlow 1.4.0 [8] οι δέσμες είναι ένας μηχανισμός ο οποίος ομαδοποιεί ένα σύνολο μηνυμάτων OpenFlow σε μία οντότητα. Οι δέσμες μπορούν να επιτελούν δύο λειτουργίες. Η πρώτη είναι να ομαδοποιούν συσχετιζόμενες τροποποιήσεις κατάστασης σε έναν μεταγωγέα, έτσι ώστε να εφαρμόζονται είτε όλες οι τροποποιήσεις είτε καμία. Έτσι αν κάποια τροποποίηση αποτύχει να εφαρμοστεί να αποτυγχάνουν και όλες οι υπόλοιπες. Η δεύτερη λειτουργία είναι να συγχρονίζονται καλύτερα οι αλλαγές σε ένα σύνολο από OpenFlow μεταγωγείς. Οι δέσμες μπορούν να προετοιμαστούν και να επικυρωθούν σε κάθε μεταγωγέα και να εφαρμοστούν όλες την ίδια στιγμή. Όταν ένας ελεγκτής θέλει να προσθέσει μια δέσμη σε έναν μεταγωγέα, στέλνει ένα OFPBCT_OPEN_REQUEST μήνυμα για να ανοίξει μια δέσμη σε έναν μεταγωγέα και μετά προσθέτει τα μηνύματα στη δέσμη ένα-ένα. Εάν το ν-ιοστό μήνυμα αποτύχει, ο μεταγωγέας θα ενημερώσει τον ελεγκτή με ένα μήνυμα λάθους και τότε ο ελεγκτής θα αναλάβει τον ρόλο να δώσει εντολή στον μεταγωγέα να απορρίψει την δέσμη μαζί με τα μηνύματα που περιέχει.

Συγχρονισμένοι πίνακες ροής (Synchronized Tables)

Οι πίνακες ροής μπορούν να συγχρονιστούν και προς τις δύο κατευθύνσεις ή προς μία κατεύθυνση. Για κάθε συγχρονιζόμενο πίνακα, μία ιδιότητα του πίνακα περιγράφει τον πίνακα από τον οποίο συγχρονίζεται ο εκάστοτε πίνακας ροής. Εάν ο συγχρονισμός είναι και προς τις δύο κατευθύνσεις τότε οι αλλαγές οι οποίες γίνονται από τον ελεγκτή στην υπό συγχρονισμό εγγραφή πρέπει να αντικατοπτρίζονται και στην αρχική εγγραφή. Οι συγχρονιζόμενοι πίνακες ροής εκφράζονται χρησιμοποιώντας μια συγκεκριμένη ιδιότητα του πίνακα, την OFPTFPT_TABLE_SYNC_FROM, η οποία όμως δεν καθορίζει την αντίστοιχη μετατροπή στους πίνακες ροής.

Ιδιότητες Οπτικών θυρών (Optical Port Properties)

Πλέον το TLV έχει εκσυγχρονιστεί ακόμα περισσότερο από την έκδοση του OpenFlow 1.2.0 και είναι πολύ πιο επεκτάσιμο. Καινούρια TLV έχουν προστεθεί στις προηγούμενες δομές με τη μορφή ιδιοτήτων στο τέλος της δομής. Για παράδειγμα στη δομή για τις θύρες, έχουμε προσθήκη ιδιότητας περιγραφής, ιδιότητας στατιστικών και μετατροπής της θύρας. Ανάμεσα στις δομές θυρών ένα νέο σύνολο ιδιοτήτων θυρών προστέθηκε, με σκοπό την υποστήριξη οπτικών θυρών. Αυτές οι νέες ιδιότητες μπορεί να χρησιμοποιηθούν για τη διαμόρφωση είτε οπτικών θυρών Ethernet είτε

οπτικών θυρών μεταγωγέων κυκλώματος. Με την οπτική θύρα μπορεί να εφαρμοστεί στο SDN Fiber Channel over Ethernet (FCoE), η οποία τεχνολογία ενθυλακώνει πλαίσια Fiber Channel πάνω από Ethernet δίκτυα.

Αφαίρεση των εγγραφών μετά τη διαγραφή του μετρητή ροής (Flow-removed reason for meter delete)

Μέχρι τώρα οι εγγραφές διαγράφονταν από το Flow Table είτε με εντολή του ελεγκτή, είτε επειδή έληγε ο χρόνος διατήρησης της εγγραφής στον μεταγωγέα. Πλέον όταν ένας μετρητής ροής (meter) διαγράφεται από τον μεταγωγέα, όλες οι εγγραφές που χρησιμοποιούν τον μετρητή, διαγράφονται και αυτές.

Συνοπτικά επιπρόσθετες αλλαγές

Νέοι μηχανισμοί για την υποστήριξη πολλών ελεγκτών. Πιο περιγραφικοί λόγοι για τα packet-in events. PBB UCA πεδίο επικεφαλίδας. Αλλαγή προεπιλεγμένης TCP θύρας από 6633 σε 6653. Νέοι κώδικες για σφάλματα.

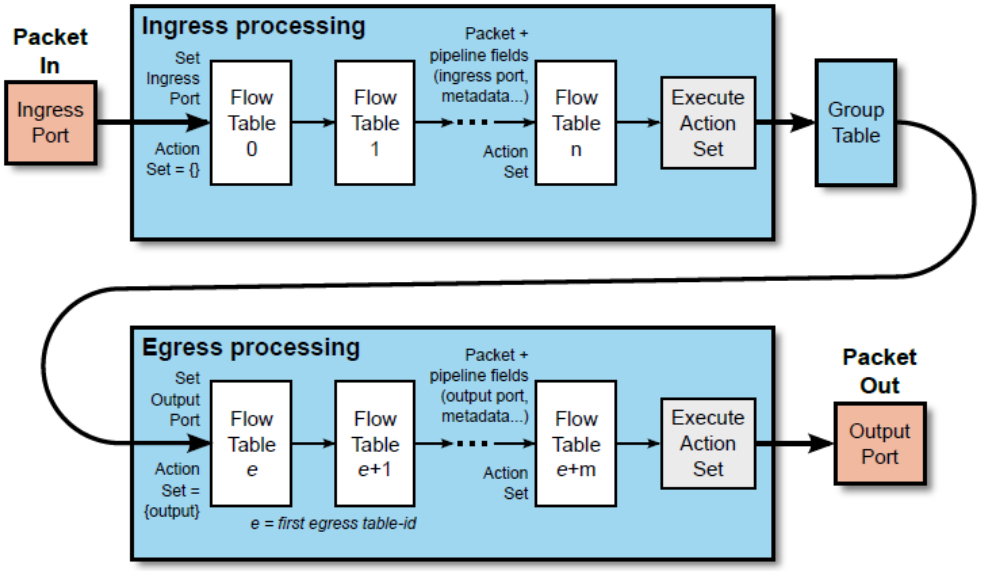
2.8 OpenFlow 1.5.0

Κυριότερες αλλαγές - προσθήκες στο OpenFlow 1.5.0

Πίνακες εξόδου (Egress Tables)

Στις προηγούμενες εκδόσεις, όλη η επεξεργασία γινόταν στα πλαίσια της εισερχόμενης θύρας. Η έκδοση 1.5.0 [9] εισάγει τους πίνακες εξόδου, δίνοντας την δυνατότητα η επεξεργασία να γίνει στα πλαίσια της εξερχόμενης θύρας. Η διαδικασία σωλήνωσης (pipeline) γίνεται πλέον σε δύο στάδια, την επεξεργασία εισόδου και εξόδου (ingress processing - egress processing). Ο διαχωρισμός των δύο σταδίων υποδεικνύεται από τον πρώτο πίνακα εξόδου. Όλοι οι πίνακες με αριθμό μικρότερο από τον πρώτο πίνακα εξόδου χρησιμοποιούνται ως πίνακες εισόδου και κανένας πίνακας με αριθμό μεγαλύτερο ή ίσο από τον πρώτο πίνακα εξόδου δεν μπορεί να χρησιμοποιηθεί ως εισόδου. Η διαδικασία της σωλήνωσης ξεκινά με την εισερχόμενη επεξεργασία του πρώτου πίνακα ροής. Το πακέτο θα πρέπει να αντιστοιχηθεί σε κάποια από τις εγγραφές του πίνακα 0. Άλλοι πίνακες εισόδου μπορεί να χρησιμοποιηθούν ανάλογα με το αποτέλεσμα της αντιστοίχισης του πρώτου πίνακα. Εάν το αποτέλεσμα της επεξεργασίας εισόδου είναι να προωθηθεί το πακέτο σε κάποια από τις θύρες εξόδου, ο μεταγωγέας OpenFlow μπορεί να πραγματοποιήσει επεξεργασία εξόδου στα πλαίσια της θύρας εξόδου. Η επεξεργασία εξόδου είναι προαιρετική, ένας μεταγωγέας μπορεί να μην υποστηρίζει καθόλου πίνακες εξόδου. Εάν δεν υπάρχει πίνακας εξόδου τότε το πακέτο προωθείται έξω από τον μεταγωγέα. Εάν υπάρχει πίνακας εξόδου, το πακέτο

πρέπει να αντιστοιχηθεί με τις εγγραφές αυτού του πίνακα και μπορεί να χρειαστούν και άλλοι πίνακες εξόδου, ανάλογα με το αποτέλεσμα της αντιστοίχισης.



Η ροή του πακέτου μέσα από τη διαδικασία σωλήνωσης, πηγή [9].

Διαδικασία σωλήνωσης με βάση το είδος πακέτου (Packet Type aware pipeline)

Σε όλες τις προηγούμενες εκδόσεις, όλα τα πακέτα έπρεπε να ήταν Ethernet. Πλέον η έκδοση 1.5 δίνει τη δυνατότητα και σε άλλα ήδη πακέτων, όπως PPP πακέτα. Ένα νέο πεδίο OXM σωλήνωσης χαρακτηρίζει το είδος του πακέτου. Επίσης το πεδίο για το είδος του πακέτου μπορεί να χρησιμοποιηθεί σε packet-in και packet-out μηνύματα, για να χαρακτηρίσει το είδος του «φορτίου» που μεταφέρεται από αυτά τα μηνύματα.

Επεκτάσιμα στατιστικά εγγραφών (Extensible Flow entry Statistics)

Εισάγονται πλέον τα OpenFlow eXtensible Statistics (OXS) για να κωδικοποιήσουν τα στατιστικά των εγγραφών. Περιλαμβάνουν πεδία όπως: διάρκεια εγγραφής, πλήθος εγγραφών, πλήθος πακέτων, πλήθος από bytes. Εισάγεται νέο στατιστικό εγγραφών, το flow idle time. Επίσης πλέον χρησιμοποιείται το OXS σε κάθε είδους μήνυμα που περιλαμβάνει στατιστικά εγγραφών.

Έναυσμα αποστολής στατιστικών εγγραφών (Flow Entry Statistics Trigger)

Πλέον τα στατιστικά δεν στέλνονται συνέχεια στον ελεγκτή αλλά όταν υπάρχουν κάποια όρια αριθμού στατιστικών τότε στέλνονται όλα μαζί. Έχουμε νέα εντολή OFPIT_STAT_TRIGGER για να καθορίσει ένα σύνολο στατιστικών ορίων χρησιμοποιώντας τα OXS.

Αντιγραφή πεδίου ενεργειών (Copy Field action)

Η υπάρχουσα Set-Field ενέργεια επιτρέπει να θέσεις το πεδίο της επικεφαλίδας ή το πεδίο σωλήνωσης με μια στατική τιμή. Πλέον το Copy-Field δίνει τη δυνατότητα να αντιγράψει κάποιος την τιμή από ένα πεδίο επικεφαλίδας ή πεδίο σωλήνωσης σε μια άλλη επικεφαλίδα ή πεδίο σωλήνωσης. Νέα εντολή OFPAT_COPY_FIELD δίνει τη δυνατότητα να αντιγραφούν τιμές ανάμεσα σε πεδία.

Αντιστοίχιση «σημαιών» TCP (TCP flags Matching)

Πλέον μπορούμε να έχουμε αντιστοίχιση στα TCP flags SYN, ACK και FIN και μπορεί να χρησιμοποιηθεί για να εντοπιστεί η αρχή και το τέλος μιας TCP σύνδεσης.

Συνοπτικά επιπρόσθετες αλλαγές

Εντολή Group για επιλεκτικές ενέργειες σε δοχεία (selective bucket operation). Πλέον η ενέργεια set field μπορεί να καθορίσει το πεδίο metadata όπως επίσης και κανόνες wildcard. Κατάσταση της σύνδεσης του ελεγκτή. Ενέργειες στους μετρητές ροής. Ιδιότητες θυρών για τα πεδία της σωλήνωσης. Όλα τα OXM-IDs είναι 64 bits.

2.9 OpenFlow Ελεγκτές (OpenFlow Controllers)

Όπως γνωρίζουμε στο OpenFlow υπάρχει διαχωρισμός ανάμεσα στο επίπεδο ελέγχου και δεδομένων. Εδώ θα ασχοληθούμε με το επίπεδο ελέγχου. Ο ελεγκτής είναι κατά κάποιο τρόπο το λειτουργικό σύστημα του δικτύου και έχει την πλήρη εποπτεία του.

2.9.1 Ελεγκτής Ryu

Ο Ryu [10] είναι ελεγκτής ανοιχτού κώδικα γραμμένος σε Python . Υποστηρίζει τις εκδόσεις OpenFlow 1.0, 1.1, 1.2, 1.3, 1.4, 1.5 και τις επεκτάσεις Nicira. Επίσης δουλεύει μαζί με το OpenStack. Διαθέτει πολύ καλό documentation ειδικά για την έκδοση 1.3 του OpenFlow, στην οποία περιγράφεται το τι λειτουργίες μπορεί να επιτελέσει σε έναν OpenFlow μεταγωγέα και είναι για παράδειγμα hub, traffic monitor, spanning tree, link aggregation, firewall, δρομολογητής, QoS per-flow ή DiffServ ή Meter Table και όλα αυτά χρησιμοποιώντας Rest API, έτσι ώστε με μηνύματα σε format JSON να μπορεί κάποιος να δώσει εντολές στον ελεγκτή. Επίσης έχει υλοποιηθεί και module στο οποίο οι ειδοποιήσεις από το snort πηγαίνουν και στον ελεγκτή με μελλοντική προοπτική να υλοποιηθούν ενέργειες για συγκεκριμένες πολιτικές ασφαλείας από τον ελεγκτή. Μειονέκτημα ως προς την επίδοση είναι η χρήση python ως γλώσσας προγραμματισμού.

2.9.2 Ελεγκτής Floodlight

Ο ελεγκτής Floodlight [11] είναι γραμμένος σε java και υποστηρίζει το OpenFlow 1.0. Διαθέτει Rest API και πολύ καλό documentation και προήλθε από τον ελεγκτή Beacon. Ο Floodlight υποστηρίζει δύο εφαρμογές reactive packet forwarding και έχουν διαφορετικές συμπεριφορές.

Forwarding: Είναι ενεργοποιημένη από προεπιλογή. Δίνει τη δυνατότητα να προωθούνται πακέτα από άκρο σε άκρο στις εξής 2 τοπολογίες:

- Μέσα σε ένα "OpenFlow Island"(Η έννοια OpenFlow island χαρακτηρίζει ένα σύνολο από OpenFlow μεταγωγείς με συσκευές συνδεδεμένες σε κάθε ένα από αυτά. Ανάλογα, ένα non OpenFlow island είναι το σύνολο από μη OpenFlow μεταγωγείς). Όταν μία συσκευή A θέλει να στείλει ένα πακέτο σε ίδιο OpenFlow island σε μια συσκευή B, η λειτουργία forwarding υπολογίζει το ελάχιστο μονοπάτι από το A στο B.
- OpenFlow islands με non-OpenFlow islands ανάμεσά τους. Κάθε OpenFlow island μπορεί να έχει ακριβώς μία σύνδεση που συνδέει τα non-OpenFlow islands. Επίσης τα OpenFlow και non-OpenFlow islands δεν πρέπει να δημιουργούν κύκλους. Το Forwarding υπολογίζει το ελάχιστο μονοπάτι σε κάθε OpenFlow island και περιμένει τα πακέτα να προωθηθούν στα non-OpenFlow islands (θεωρώντας ότι κάθε non-OF island είναι στο ίδιο L2 broadcast domain).

  Τα μονοπάτια εγκαθίστανται και μετά από 5 sec by default λήγουν.

Learning switch: Ένας κλασικός L2 μεταγωγέας.
Δεν δουλεύει όταν οι μεταγωγείς σε ένα island δημιουργούν κύκλο.
Επίσης υποστηρίζει Static Flow Entry Pusher εφαρμογή και Circuit Pusher έτσι ώστε οι χρήστες να μπορούν εκ των προτέρων να εγκαθιστούν μονοπάτια προώθησης στο δίκτυο.

2.9.3 Ελεγκτής OpenDaylight

Το project OpenDaylight [12] έχει ως σκοπό την δημιουργία ενός SDN ελεγκτή, κατάλληλου για πολλά διαφορετικά δικτυακά περιβάλλοντα. Ο OpenDaylight χρησιμοποιεί την java ως γλώσσα προγραμματισμού. Η αρχιτεκτονική του βασίζεται σε 3 επίπεδα. Τις υποκείμενες διεπαφές (southbound API), το στρώμα αφαίρεσης (Service Abstraction Layer – SAL) και τις υπερκείμενες διεπαφές (northbound API). Οι υποκείμενες διεπαφές μπορούν να υποστηρίξουν πολλά πρωτόκολλα ως ξεχωριστά plugins, όπως το OpenFlow 1.0, OpenFlow 1.3, BGP-LS, LISP, SNMP και άλλα. Το επίπεδο αφαίρεσης είναι μια κοινή γλώσσα κατανοητή και από τις υπερκείμενες και τις υποκείμενες διεπαφές Οι υπερκείμενες διεπαφές που χρησιμοποιούνται από τις εφαρμογές υποστηρίζουν το OSGi (Open Services Gateway Initiative) το οποίο είναι ένα δυναμικό σύστημα για modules στη java, όσο και bidirectional REST APIs. Το OSGi framework χρησιμοποιείται κυρίως από εφαρμογές που θα τρέξουν τοπικά στον

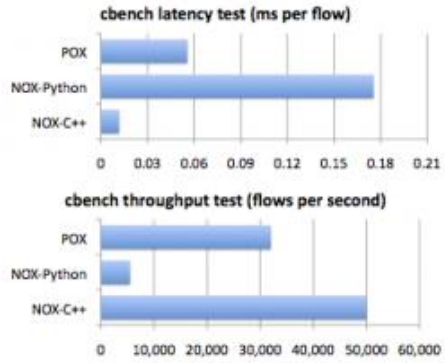
ίδιο χώρο διευθύνσεων με τον ελεγκτή, ενώ το REST API από εφαρμογές που μπορεί να τρέξουν στο ίδιο μηχάνημα με τον ελεγκτή ή σε διαφορετικό μηχάνημα.

2.9.4 Ελεγκτής NOX

Υπάρχουν δύο κατηγορίες του ελεγκτή NOX [13]. Η NOX-Classic η οποία ήταν γραμμένη σε C++/Python και πλέον δεν υποστηρίζεται. Η καινούρια έκδοση του NOX «new NOX» είναι γραμμένη μόνο σε C++. Υποστηρίζει OpenFlow 1.0. Το προγραμματιστικό μοντέλο βασίζεται σε γεγονότα (events) και ο προγραμματιστής καλείται να γράψει σχετικές ρουτίνες χειρισμού (event handlers) για να εκτελεστούν όταν ένα συγκεκριμένο συμβάν προκύψει στο δίκτυο. Η τιμή που επιστρέφει ένας event handler υποδεικνύει και το ποιες ενέργειες πρέπει να εφαρμοστούν. Έχει εγκατεστημένες εφαρμογές όπως learning switch, topology discovery και network-wide switch. Μπορεί να έχει προγραμματιστική δυσκολία λόγω του ότι η C++ είναι γλώσσα προγραμματισμού «χαμηλού επιπέδου» αλλά έχει πολύ καλή επίδοση λόγω αυτού.

2.9.5 Ελεγκτής POX

Ο ελεγκτής POX [14][15] είναι ουσιαστικά ο NOX ελεγκτής σε Python [25].Υποστηρίζει μόνο την έκδοση OpenFlow 1.0. Έχει πολύ μεγάλη υποστήριξη από developers και η ανάπτυξη κώδικα είναι σχετικά εύκολη. Λόγω του ότι είναι γραμμένος σε Python η οποία είναι γλώσσα «υψηλού επιπέδου» έχει το μειονέκτημα ότι έχει μειωμένη επίδοση σε σχέση με τον NOX.



Συγκριτικό latency και throughput ανάμεσα σε NOX-python, Nox-C++ και Pox, πηγή [14]

Η κύρια λειτουργικότητα του POX παρέχεται από εξαρτήματα (components). Όταν αναφερόμαστε σε components, εννοούμε αρχεία που μπορούμε να προσθέσουμε στην εντολή με την οποία εκτελούμε τον POX. Ο POX διαθέτει εκ των προτέρων components και κάποια παρέχουν τη βασική λειτουργικότητα, κάποια ορισμένα βολικά χαρακτηριστικά και κάποια είναι απλώς παραδείγματα. Στην συνέχεια παρατίθενται ενδεικτικά κάποια από τα διαθέσιμα components.

2.9.5.1 Εφαρμογές στον POX (POX components)

Py
Εκκίνηση ενός διαδραστικού Python interpreter. Χρήση για αποσφαλμάτωση και πειραματισμό.

Forwarding.hub
Απλή εγκατάσταση FLOOD κανόνων σε έναν OpenFlow μεταγωγέα, μετασχηματίζοντας το σε hub.

Forwarding.l2_learning
Παρόμοια λειτουργία με ένα L2 learning switch. Παρόλο που το εξάρτημα εστιάζει στις διάφορες διευθύνσεις του στρώματος 2, οι κανόνες που εγκαθίστανται είναι όσο το δυνατόν πιο συγκεκριμένοι (full – match).

Forwarding.l2_pairs
Όμοια λειτουργία με το l2_learning ωστόσο οι κανόνες βασίζονται κυρίως στις διευθύνσεις MAC.

forwarding.l3_learning
Το συγκεκριμένο εξάρτημα δεν αποτελεί ακριβώς έναν δρομολογητή, αλλά επίσης δεν είναι ένας L2 μεταγωγέας. Μαθαίνει και καταχωρεί διευθύνσεις IP, χωρίς να δίνει ιδιαίτερη σημασία σε έννοιες όπως τα διαφορετικά υποδίκτυα. Προκειμένου να εναρμονιστεί με τον τρόπο λειτουργίας των host, είναι δυνατό κατά την φόρτωση να ορίσουμε fake gateways.

forwarding.l2_multi
Συμπεριφέρεται σαν ένα learning switch. Η σημαντική του διαφορά με τα αντίστοιχα εξαρτήματα είναι πως χρησιμοποιεί το **openflow.discovery** για να ενημερωθεί για όλη την τοπολογία του δικτύου. Έτσι μόλις ένας μεταγωγέας καταχωρήσει μια διεύθυνση MAC όλα μοιράζονται αυτή την καταχώρηση. Σημειώνεται πως απαιτείται επίσης, η χρήση του εξαρτήματος **openflow.discovery**.

forwarding.topo_proactive
Εγκατάσταση κανόνων προκαταβολικά, με βάση μια διεύθυνση IP που έχει σημασία για την τοπολογία. Η μεταγωγή βασίζεται στο forwarding.l2_multi που αναφέρθηκε προηγουμένως.

openflow.spanning_tree
Χρησιμοποιεί το **openflow.discovery** για την δημιουργία μιας επισκόπησης της τοπολογίας του δικτύου, δημιουργεί ένα spanning tree και στη συνέχεια απενεργοποιεί το flooding για τις πόρτες που δεν είναι στο spanning tree.

openflow.webservice
Παρέχει ένα web service τύπου JSON-RPC για την αλληλεπίδραση με το OpenFlow πρωτόκολλο. Προέρχεται από την υπηρεσία μηνυμάτων of_service και προϋποθέτει την χρήση του webcore εξαρτήματος.

web.webcore
Εκκίνηση ενός web server εντός της διεργασίας του POX.

Messenger
Προσφέρει μια διεπαφή για τον POX ώστε να επικοινωνεί με εξωτερικές διαδικασίες μέσω αμφίδρομων μηνυμάτων βασισμένα σε JSON.

openflow.of_01
Επικοινωνία με OpenFlow 1.0 switches. Στην περίπτωση φόρτωσης διαφορετικών components που χρησιμοποιούν OpenFlow φορτώνεται και αυτό με τις προεπιλεγμένες τιμές. Υπάρχει δυνατότητα αλλαγής των τιμών στην χειροκίνητη εκτέλεση (port, address, private-key, certificate, certificate-authority)

openflow.discovery
Αποστολή ειδικά διαμορφωμένων LLDP μηνυμάτων ώστε ο ελεγκτής να ανακαλύψει την τοπολογία του δικτύου

openflow.debug
Δημιουργία αρχείων pcap με τα OpenFlow μηνύματα που ανταλλάζονται.

misc.full_payload
Η προεπιλεγμένη συμπεριφορά όταν ένα πακέτο δεν μπορεί να αντιστοιχισθεί σε κάποια εγγραφή του Flow Table είναι η αποστολή ενός μέρους του πακέτου (τα πρώτα 128 bytes) στον ελεγκτή. Αυτό το εξάρτημα τροποποιεί κατάλληλα τους μεταγωγείς που συνδέονται στον ελεγκτή ώστε να στέλνουν ολόκληρο το πακέτο.

2.9.5.2 Log

Ο POX χρησιμοποιεί το σύστημα καταγραφής συμβάντων της Python. Το εξάρτημα log επιτρέπει την παραμετροποίηση του σε σημαντικό βαθμό μέσω της γραμμής εντολών.

Ενδεικτικά:

- Απενεργοποίηση της καταγραφής
- Μορφοποίηση της καταγραφής, προσθήκη και μορφοποίηση timestamps
- Εκτύπωση stderr, stdout, αρχεία, αποστολή μέσω TCP Sockets, UDP Datagrams και HTTP GET/POST

log.level

Η καταγραφή ακολουθεί την ίδια δομή που ακολουθεί το σύστημα της Python. Διαφορετικά εξαρτήματα έχουν τους δικούς τους loggers το όνομα του καθενός φαίνεται στα μηνύματα καταγραφής. Στη πραγματικότητα οι καταγραφείς σχηματίζουν την δική τους ιεραρχία και επιπρόσθετα κάθε ένας έχει ένα επίπεδο (level), το οποίο αντιστοιχεί πόσο σημαντικό ή όχι είναι ένα μήνυμα. Το εξάρτημα log.level επιτρέπει την παραμετροποίηση του επιπέδου των πληροφοριών που δείχνει το κάθε σύστημα καταγραφής.

Το πόσο σημαντικό είναι κάθε μήνυμα περιγράφεται από τις ακόλουθες κατηγορίες (από το πλέον στο λιγότερο σημαντικό):

- CRITICAL
- ERROR
- WARNING
- INFO
- DEBUG

Το προεπιλεγμένο επίπεδο καταγραφής για τον POX είναι το INFO.

2.9.5.3 POX APIs

Εκτός από τα εξαρτήματα στα οποία έγινε αναφορά, ο POX διαθέτει ένα σύνολο APIs προκειμένου να διευκολύνει την ανάπτυξη εφαρμογών δικτυακού ελέγχου. Στη συνέχεια παρουσιάζονται ενδεικτικά κάποια εξ αυτών:

POX Core

Ο POX έχει το αντικείμενο «core», το οποίο λειτουργεί σαν κεντρικός κόμβος για μεγάλο μέρος του API του POX. Κάποιες από τις λειτουργίες που προσφέρει είναι απλώς wrappers (ρουτίνες και συναρτήσεις που κύριος στόχος τους είναι η κλήση

δευτερευουσών ρουτινών και συναρτήσεων) ενώ άλλες είναι αυτόνομες. Ωστόσο, από τους βασικότερους λόγους ύπαρξης του αντικειμένου core είναι να χρησιμοποιείται σαν σημείο συνάντησης μεταξύ των components. Έτσι αντί να χρησιμοποιούνται οι εντολές import ώστε ένα εξάρτημα να εισάγει ένα άλλο προκειμένου να αλληλεπιδράσει μαζί του, τα εξαρτήματα εγγράφονται πλέον (register) στο core και επικοινωνούν μεταξύ τους μέσω αυτού.
Πολλά modules στον POX χρειάζεται να έχουν πρόσβαση στο core αντικείμενο. Από σύμβαση αυτό γίνεται γράφοντας from pox.core import core.

Δουλεύοντας με Διευθύνσεις: pox.lib.addresses

IPv4, IPv6 και Ethernet διευθύνσεις αναπαρίστανται ως IPAddr, IPAddr6 και EthAddr κλάσεις του pox.lib.addresses.

Σύστημα διαχείρισης γεγονότων: pox.lib.revent

Ο χειρισμός των γεγονότων στον POX ακολουθεί το μοτίβο «publish – subscribe». Το ζητούμενο είναι όταν συμβαίνει ένα event (γίνεται raise) να ενεργοποιηθεί ένα συγκεκριμένο τμήμα κώδικα για τον χειρισμό του (γνωστό και ως event handler). Σημαντικό μέρος του τρόπου λειτουργίας του POX βασίζεται στη δημοσίευση και στον χειρισμό των events. Κάτι τέτοιο διευκολύνει την ανάπτυξη εφαρμογών απλώς δημιουργώντας εξαρτήματα που περιέχουν handlers για τα διάφορα events (Packet In, Connection Up, Port Status).

Πακέτα (pox.lib.packet)

Πολλές εφαρμογές στον POX βασίζονται στην αλληλεπίδραση με πακέτα (για παράδειγμα κατασκευή πακέτων και αποστολή μέσω του μεταγωγέα ή προσπέλαση ενός πακέτου που ελήφθη από ένα packet in event). Προκείμενου να διευκολύνει την παραπάνω διαδικασία ο POX διαθέτει μια βιβλιοθήκη για την ανάλυση και κατασκευή πακέτων. Κάποιοι από τους τύπους των πακέτων που υποστηρίζει η βιβλιοθήκη του POX είναι Ethernet, ARP, IPv4, ICMP, TCP, UDP, DHCP, DNS, LLDP, VLAN. Κάθε τύπος υποστηρίζει διάφορα attributes.

Νήματα, διεργασίες και χρονόμετρα (pox.lib.recoco)

Η βιβλιοθήκη recoco του POX χρησιμοποιείται για την υλοποίηση απλών συνεργατικών διεργασιών. Ίσως το βασικότερο πλεονέκτημα των διεργασιών αυτών είναι πως συνήθως δεν χρειάζεται να γίνει κάποια ειδική ενέργεια για τον συγχρονισμό τους. Όσον αφορά την δημιουργία διαφορετικών νημάτων μπορεί να χρησιμοποιηθεί ξανά η βιβλιοθήκη recoco ή συνηθισμένη βιβλιοθήκη “threading” της Python. Το πιο βασικό ενός recoco task είναι ότι δεν χρειάζεται να γίνει block. Κάνοντας stall ένα recoco task θα εμποδίσει άλλα tasks από το να εκτελεστούν. Κάποιες διεργασίες όπως sleep, έχουν παραπλήσιες εντολές στη recoco.

Ενδιαφέρον παρουσιάζει και η εκτέλεση διεργασιών μετά από κάποιο χρονικό μέσω χρονομέτρων. Ενδεικτικά παραδείγματα χρήσης η αποσύνδεση του ελεγκτή από τον μεταγωγέα αν δεν έχει ληφθεί για κάποιο διάστημα απάντηση σε μηνύματα echo, ή επίσης το αίτημα στον μεταγωγέα ανά περιόδους ώστε να στέλνει στατιστικά δεδομένα για τους κανόνες που περιέχει.

Παρακάτω θα αναλυθούν βασικά components [16] του POX, προκειμένου να διαφανούν οι νέες δυνατότητες που προσφέρει το OpenFlow στις δικτυακές συσκευές.

2.9.5.4 Hub (hub.py)

```
from pox.core import core

import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr

log = core.getLogger()

def _handle_ConnectionUp (event):
  msg = of.ofp_flow_mod()
  msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
  event.connection.send(msg)
  log.info("Hubifying %s", dpidToStr(event.dpid))

def launch ():
  core.openflow.addListenerByName("ConnectionUp",
_handle_ConnectionUp)

  log.info("Hub running.")
```

Η _handle_ConnectionUp συνάρτηση, δημιουργεί ένα μήνυμα που τροποποιεί τον πίνακα ροής του μεταγωγέα. Στην αρχή δημιουργούμε το μήνυμα και μέσα σε αυτό περιλαμβάνουμε την ενέργεια να στείλει το πακέτο έξω από κάθε θύρα του μεταγωγέα (OFPP_FLOOD). Έτσι ουσιαστικά φτιάχνουμε εύκολα έναν αναγεννητή σήματος που ότι πακέτο παίρνει σε μία θύρα το στέλνει έξω από όλες τις άλλες.

2.9.5.5 Learning_Switch1 (l2_pairs.py)

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
log = core.getLogger()
table = {}
all_ports = of.OFPP_FLOOD

def _handle_PacketIn (event):
  packet = event.parsed
  table[(event.connection,packet.src)] = event.port
  dst_port = table.get((event.connection,packet.dst))

  if dst_port is None:
    msg = of.ofp_packet_out(data = event.ofp)
    msg.actions.append(of.ofp_action_output(port = all_ports))
    event.connection.send(msg)
  else:
    msg = of.ofp_flow_mod()
    msg.match.dl_dst = packet.src
    msg.match.dl_src = packet.dst
    msg.actions.append(of.ofp_action_output(port = event.port))
    event.connection.send(msg)
    msg = of.ofp_flow_mod()
    msg.data = event.ofp
    msg.match.dl_src = packet.src
    msg.match.dl_dst = packet.dst
    msg.actions.append(of.ofp_action_output(port = dst_port))
    event.connection.send(msg)
    log.debug("Installing %s <-> %s" % (packet.src, packet.dst))

def launch (disable_flood = False):
  global all_ports
  if disable_flood:
    all_ports = of.OFPP_ALL
  core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
  log.info("Pair-Learning switch running.")
```

Εδώ έχουμε μια πολύ πιο απλοϊκή υλοποίηση μεταγωγέα. Ο ελεγκτής περιμένει για packet in events. Έχουμε ένα dictionary στο οποίο γίνεται αντιστοίχιση mac address και θύρας μεταγωγέα και εδώ αρχικοποιείται ως table={}. Με το table[(event.connection,packet.src)] = event.port μαθαίνουμε την mac διεύθυνση αποστολέα και την θύρα από την οποία εισήλθε το πακέτο. Εάν δεν υπάρχει η διεύθυνση προορισμού μέσα στον πίνακα θα σταλεί το πακέτο έξω από όλες τις θύρες εκτός από αυτήν από την οποία εισήλθε. Πλέον αν γνωρίζει ο ελεγκτής τη διεύθυνση προορισμού και αποστολής μπορεί να στείλει κανόνες στο μεταγωγέα και για τις δύο κατευθύνσεις.

2.9.5.6 Learning_Switch2 (l2_learning.py)

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
import time

log = core.getLogger()
_flood_delay = 0

class LearningSwitch (object):
  def __init__ (self, connection, transparent):
    self.connection = connection
    self.transparent = transparent
    self.macToPort = {}
    connection.addListeners(self)
    self.hold_down_expired = _flood_delay == 0

  def _handle_PacketIn (self, event):
    packet = event.parsed
    def flood (message = None):
      msg = of.ofp_packet_out()
      if time.time() - self.connection.connect_time >= _flood_delay:

        if self.hold_down_expired is False:
          self.hold_down_expired = True
          log.info("%s: Flood hold-down expired -- flooding",
              dpid_to_str(event.dpid))

        if message is not None: log.debug(message)
        msg.actions.append(of.ofp_action_output(port =
of.OFPP_FLOOD))
      else:
        pass
      msg.data = event.ofp
      msg.in_port = event.port
      self.connection.send(msg)

    def drop (duration = None):
      if duration is not None:
        if not isinstance(duration, tuple):
          duration = (duration,duration)
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet)
        msg.idle_timeout = duration[0]
        msg.hard_timeout = duration[1]
        msg.buffer_id = event.ofp.buffer_id
        self.connection.send(msg)
      elif event.ofp.buffer_id is not None:
        msg = of.ofp_packet_out()
        msg.buffer_id = event.ofp.buffer_id
        msg.in_port = event.port
        self.connection.send(msg)

    self.macToPort[packet.src] = event.port
```

```
if not self.transparent:
      if packet.type == packet.LLDP_TYPE or
packet.dst.isBridgeFiltered():
        drop()
        return

    if packet.dst.is_multicast:
      flood()
    else:
      if packet.dst not in self.macToPort:
        flood("Port for %s unknown -- flooding" % (packet.dst,))
      else:
        port = self.macToPort[packet.dst]
        if port == event.port:
          log.warning("Same port for packet from %s -> %s on %s.%s.
Drop."
              % (packet.src, packet.dst, dpid_to_str(event.dpid),
port))
          drop(10)
          return
        log.debug("installing flow for %s.%i -> %s.%i" %
                  (packet.src, event.port, packet.dst, port))
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet, event.port)
        msg.idle_timeout = 10
        msg.hard_timeout = 30
        msg.actions.append(of.ofp_action_output(port = port))
        msg.data = event.ofp
        self.connection.send(msg)

class l2_learning (object):
  def __init__ (self, transparent):
    core.openflow.addListeners(self)
    self.transparent = transparent

  def _handle_ConnectionUp (self, event):
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)

def launch (transparent=False, hold_down=_flood_delay):
  try:
    global _flood_delay
    _flood_delay = int(str(hold_down), 10)
    assert _flood_delay >= 0
  except:
    raise RuntimeError("Expected hold-down to be a number")

  core.registerNew(l2_learning, str_to_bool(transparent))
```

Περιγραφή λειτουργίας l2_learning switch

- Ο πίνακας του μεταγωγέα είναι αρχικά άδειος.
- Για κάθε εισερχόμενο πλαίσιο αποθηκεύει:
  1) Την εισερχόμενη διεπαφή από την οποία ήρθε το πλαίσιο.
  2) Τον χρόνο στον οποίο το συγκεκριμένο πλαίσιο ήρθε.
  3) Διαγράφει την εγγραφή εάν κανένα πλαίσιο με την συγκεκριμένη διεύθυνση αποστολέα δεν έλθει μέσα σε ένα ορισμένο χρονικό διάστημα.
  (Επομένως κάθε πλαίσιο που φτάνει στον μεταγωγέα δημιουργεί μία καινούρια εγγραφή FlowTable του μεταγωγέα)

Ο αλγόριθμος που χρησιμοποιεί είναι ο εξής:

1) Χρήση της διεύθυνσης mac αποστολέα και της θύρας του μεταγωγέα για να γίνει ενημέρωση του πίνακα που περιέχει αντιστοιχία διεύθυνσης-θύρας. Πρόκειται για ένα λεξικό (dictionary).

2) Εάν το Ethertype είναι LLDP ή η διεύθυνση αποστολής του πακέτου είναι φιλτραρισμένη, τότε το πακέτο απορρίπτεται.(DROP)

3) Αν το πακέτο είναι πολυεκπομπής (multicast) τότε το πακέτο στέλνεται από όλες τις άλλες θύρες, πλην αυτής που προήλθε. (FLOOD)

4) Εάν η θύρα για την διεύθυνση προορισμού δεν είναι μέσα στο λεξικό αντιστοιχίας mac διεύθυνσης-θύρας τότε το πακέτο στέλνεται από όλες τις άλλες θύρες, πλην αυτής που προήλθε. (FLOOD)

5) Εάν η θύρα εξόδου του πακέτου είναι ίδια με την θύρα εισόδου, τότε το πακέτο απορρίπτεται.

6) Αλλιώς εγκαθιστά εγγραφή στον πίνακα ροής του μεταγωγέα (flow modification entry) η οποία προκύπτει από την αντιστοίχιση της διεύθυνσης αποστολέα του πακέτου και της θύρας του μεταγωγέα από την οποία το έλαβε. Έτσι όταν θα δεχτεί το switch πακέτο με προορισμό μια γνωστή διεύθυνση που υπάρχει μέσα στο λεξικό του μεταγωγέα, ο μεταγωγέας θα ξέρει από ποια θύρα να στείλει το πακέτο.

Επεξήγηση του κώδικα

Βλέπουμε ότι έχουμε μια μέθοδο launch η οποία εγγράφει το l2_learning αντικείμενο με το core του ελεγκτή POX. Με την αρχικοποίηση, αυτό το αντικείμενο προσθέτει έναν listener (core.openflow.addListener(self)) για να είναι σίγουρο ότι μπορεί να χειριστεί connection up events από OpenFlow μεταγωγείς οι οποίοι προσπαθούν να συνδεθούν στον συγκεκριμένο ελεγκτή. Μετά από μία προσπάθεια σύνδεσης από ένα OpenFlow μεταγωγέα, αυτό το αντικείμενο αρχικοποιεί το αντικείμενο

LearningSwitch και περνά το connection event σε αυτό. Στη γραμμή self.macToPort = {} αρχικοποιείται το λεξικό στο οποίο αντιστοιχίζονται mac διευθύνσεις με θύρες του μεταγωγέα. Με το connection.addListener(self) προσθέτουμε έναν listener για packet-in μηνύματα. Το βασικό εδώ είναι η λειτουργία του μεταγωγέα η οποία διαφαίνεται με τον κώδικα self.macToPort[packet.src] = event.port στην οποία γίνεται η αντιστοίχιση mac διεύθυνσης αποστολέα με τη θύρα του μεταγωγέα από την οποία εισήλθε. Έπειτα στον υπόλοιπο κώδικα υπάρχουν οι ενέργειες FLOOD και DROP ανάλογα με τις διάφορες περιπτώσεις όπως περιγράφηκαν πιο πάνω. Τέλος από τη γραμμή msg=of.ofp_flowmod() μέχρι self.connection.send(msg) είναι το μήνυμα flow modification που στέλνεται στον μεταγωγέα και καθορίζει πώς θα χειριστούν τα εισερχόμενα πακέτα.

2.9.5.7 Διαφορές Learning_Switch1 - Learning_Switch2

Γενικώς τα forwarding εξαρτήματα του POX είναι απλά παραδείγματα και δεν είναι όλα στο ίδιο επίπεδο πληρότητας. Έτσι έχουν διαφορετικές προσεγγίσεις στην υλοποίηση του γνωστού κλασικού μεταγωγέα, με το l2_pairs να είναι πιο απλό.

- Το l2_learning κάνει drop LLDP πακέτα ενώ το pairs όχι. Το pairs δεν το υλοποιεί αυτό γιατί είναι πιο απλή έκδοση.
- Το l2_learning κάνει drop πακέτα όταν εάν η θύρα εισόδου και εξόδου ενός πακέτου είναι η ίδια. Παράδειγμα είναι να έχουμε ένα hub συνδεδεμένο στο switch και 2 hosts συνδεδεμένους στο hub. Το switch βλέπει και προορισμό και αποστολέα στην ίδια θύρα. Το pairs δεν το υλοποιεί αυτό γιατί είναι πιο απλή έκδοση.
- Το learning περιλαμβάνει exact match εγγραφές και πραγματοποιεί flow-based routing, ενώ το pairs ασχολείται μόνο με mac διευθύνσεις.

2.9.6 Συνοπτική επισκόπηση των ελεγκτών

Ryu: Υποστηρίζει OpenFlow 1.0 - 1.5. Γλώσσα προγραμματισμού η Python. Δουλεύει με το OpenStack. Μειονέκτημά του η χαμηλή απόδοση.

Floodlight: Υποστηρίζει OpenFlow 1.0. Γλώσσα προγραμματισμού η Java. Διαθέτει καλό documentation και υποστηρίζει REST API. Είναι αρκετά δύσκολος στην εκμάθηση.

OpenDaylight: Υποστηρίζει OpenFlow 1.0 και 1.3. Μεγάλο πλεονέκτημα είναι ότι είναι επεκτάσιμος, δουλεύει με το OpenStack και άλλες εφαρμογές του cloud. Το μειονέκτημά του είναι ότι είναι αρκετά πολύπλοκος και δύσκολος με μέτριο documentation.

Nox: Υποστηρίζει OpenFlow 1.0. Μέτρια δυσκολία εκμάθησης. Έχει πολύ υψηλή απόδοση. Χρειάζεται καλή γνώση C++ και ρουτινών χαμηλού επιπέδου.

Pox: Υποστηρίζει OpenFlow 1.0. Γλώσσα προγραμματισμού η Python. Χρησιμοποιείται ευρέως και είναι αρκετά εύκολος στην εκμάθηση. Μειονέκτημα η χαμηλή απόδοση. Η χρήση του ενδείκνυται για έρευνα κα γρήγορη πραγματοποίηση πειραμάτων με χρήση OpenFlow.

3

Υλοποίηση υβριδικού honeypot με τον ελεγκτή POX

3.1 Honeypots

Το honeypot [17],[18],[19] είναι ένα υπολογιστικό σύστημα το οποίο έχει σαν στόχο να εντοπίσει ή να εκτρέψει την μη εξουσιοδοτημένη χρήση πληροφοριακών συστημάτων. Πρόκειται για υπολογιστικούς πόρους οι οποίοι έχουν σαν στόχο να ξεγελάσουν τον επιτιθέμενο δίνοντας την εντύπωση πως πρόκειται για πραγματικό σύστημα που περιέχει χρήσιμες πληροφορίες. Έτσι κατ' αυτόν τον τρόπο με μια παθητική μέθοδο ασφαλείας, συλλέγουμε δεδομένα για τις τεχνικές που χρησιμοποιεί ο επιτιθέμενος, προκειμένου να καταστήσουμε το δίκτυο μας πιο ασφαλές. Γενικώς η λειτουργία του honeypot βασίζεσαι στο ότι κανείς δεν πρέπει να αλληλεπιδρά με αυτά, οπότε η οποιαδήποτε κίνηση δεδομένων σε αυτά μπορεί να θεωρηθεί ως κακόβουλη. Ως honeypot θα μπορούσε να θεωρηθεί οποιοδήποτε είδος server όπως Web ή database ή file του οποίου μάλιστα οι υπηρεσίες δεν διαφημίζονται από άλλες υπηρεσίες, όπως DNS servers μιας και δεν έχουν καμία αξία παραγωγής. Οι αρχιτεκτονικές τοποθέτησης των honeypots είναι δύο, η περιμετρική και η εσωτερική. Στην περιμετρική η οποία επιλέγεται συνήθως για honeypots έρευνας, το honeypot βρίσκεται εξωτερικά του firewall προς το διαδίκτυο. Έτσι δέχονται πολλές επιθέσεις και καθίσταται ευκολότερη η ανάλυση των επιθέσεων. Στην εσωτερική τοποθέτηση το honeypot τοποθετείται εσωτερικά του τείχους προστασίας και είναι συνήθης τοποθέτηση για honeypots παραγωγής. Σε αυτή την περίπτωση στόχος είναι να ανιχνευτεί αν κάποιο εσωτερικό μηχάνημα έχει παραβιαστεί.

Κατηγοριοποίηση των Honeypots

Η κατηγοριοποίηση των honeypots μπορεί να γίνει με βάση διάφορα κριτήρια όπως το επίπεδο αλληλεπίδρασης με τους επιτιθέμενους, το πώς γίνεται η εγκατάστασή τους όπως και ο τομέας στον οποίο χρησιμοποιούνται.

3.1.1 Κατηγοριοποίηση με βάση τον τομέα χρήσης

- **Honeypots παραγωγής**

Τα honeypots παραγωγής χρησιμοποιούνται για να βοηθήσουν έναν οργανισμό ή μια εταιρεία στο να προστατέψει το εσωτερικό της δίκτυο. Δεν μπορούν όμως να αποτρέψουν επιθέσεις για αυτό και χρειάζεται να εφαρμόζονται οι κλασικοί μηχανισμοί ασφαλείας, όπως η απενεργοποίηση υπηρεσιών που δεν χρησιμοποιούνται, η έγκαιρη εγκατάσταση των patch, μηχανισμοί ασφαλείας όπως το firewall, intrusion detection systems, anti-virus και μηχανισμοί πιστοποίησης για να εμποδίσουν τους κακόβουλους χρήστες να αποκτήσουν πρόσβαση στα πληροφοριακά συστήματα της εταιρείας. Επίσης το honeypot παραγωγής πρέπει να έχει στηθεί κατά τέτοιο τρόπο έτσι ώστε ο χάκερ να μπορεί εύκολα να αποκτήσει πρόσβαση σε αυτό. Μπορούν επίσης να φανούν χρήσιμα για επιθέσεις οι οποίες δεν μπορούν να εντοπιστούν από intrusion detection systems, όπως αγνώστου τύπου επιθέσεις που δεν έχουν συγκεκριμένες υπογραφές (signatures) ή επιθέσεις που πραγματοποιούνται από υπαλλήλους της εκάστοτε εταιρίας (insiders) και οι οποίοι έχουν αυξημένα δικαιώματα. Επίσης στα IDS μπορεί να δημιουργηθούν προβλήματα όταν υπάρχει υψηλή κίνηση πακέτων και αρχίζουν να χάνουν πακέτα, πράγμα το οποίο δεν υπάρχει στα honeypots παραγωγής.

- **Honeypots έρευνας**

Τα συγκεκριμένα honeypots χρησιμοποιούνται για την συλλογή πληροφοριών σχετικά με τις τακτικές επιθέσεων κακόβουλων χρηστών. Η έρευνα των επιθέσεων και η εύρεση κάποιου αποτελεσματικού τρόπου προστασίας από αυτές είναι ο στόχος των συγκεκριμένων honeypots. Τα συστήματα αυτά περιέχουν τεράστια ποσότητα δεδομένων και μπορούν να ανακαλυφθούν μέσω αυτών νέα είδη απειλών όπως 0 day exploits ή νέα worms. Τα honeypots έρευνας είναι πολύπλοκα στην εγκατάσταση και συντήρηση και χρησιμοποιούνται από πανεπιστήμια, στρατιωτικούς ή κυβερνητικούς οργανισμούς και ερευνητικά κέντρα που τους ενδιαφέρει ο τομέας της ασφάλειας.

3.1.2 Κατηγοριοποίηση με βάση το πώς έγινε η εγκατάστασή τους.

- **Φυσικά honeypots (Physical honeypots)**

Τα honeypots τρέχουν σε φυσικά μηχανήματα και συνήθως πρόκειται για honeypots υψηλής αλληλεπίδρασης. Είναι ακριβά στην εγκατάστασή τους και συντήρησή τους, ενώ δεν είναι καθόλου πρακτική η εγκατάσταση ενός φυσικού honeypot για κάθε IP διεύθυνση.

- **Εικονικά honeypots (Virtual honeypots)**

Τα εικονικά honeypots αναπτύσσονται σε εικονικά μηχανήματα (virtual machines) τα οποία βρίσκονται στο ίδιο φυσικό μηχάνημα. Με την ιδέα αυτή μπορούμε να έχουμε πολλά honeypots στο ίδιο μηχάνημα, πράγμα που τα καθιστά φθηνά στην εγκατάσταση και στη συντήρηση. Γνωστοί hypervisors πάνω στους οποίους στήνονται τα VMs είναι ο KVM και ο Xen.

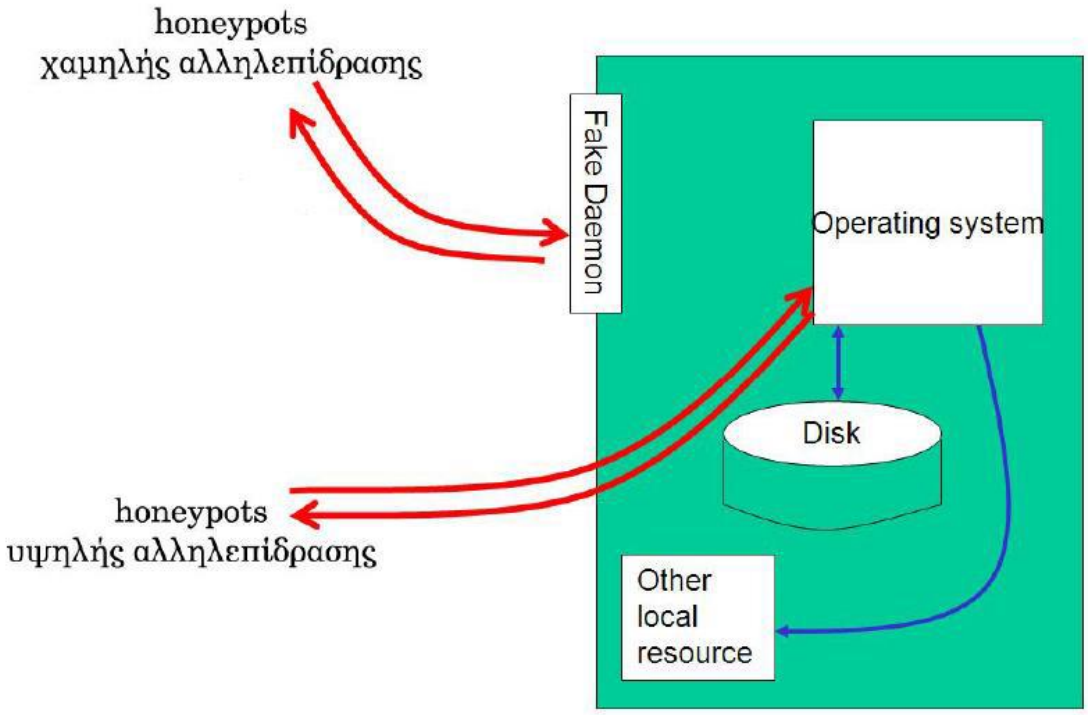
3.1.3 Κατηγοριοποίηση με βάση το βαθμό αλληλεπίδρασης.

3.1.3.1 Honeypots χαμηλής αλληλεπίδρασης (Low Interaction Honeypots)

Τα χαμηλής αλληλεπίδρασης honeypots εξομοιώνουν υπηρεσίες οι οποίες δεν είναι δυνατόν να αξιοποιηθούν από τον επιτιθέμενο έτσι ώστε να αποκτήσει πλήρη έλεγχο του συστήματος. Το ρίσκο της χρήσης τους είναι μικρό αφού ο διαχειριστής του honeypot έχει τον πλήρη έλεγχο του συστήματος στην περίπτωση μιας επίθεσης. Η αλληλεπίδραση πάντως γενικώς μεταξύ επιτιθέμενου και χρήστη εξαρτάται από το επίπεδο ακρίβειας της εξομοίωσης των υπηρεσιών από τον δημιουργό του honeypot. Για παράδειγμα μπορεί να εξομοιώνονται μόνο συγκεκριμένα Services αλλά όχι το λειτουργικό σύστημα. Επίσης η εγκατάστασή τους είναι πολύ εύκολη μιας και έχουν προκαθορισμένες ρυθμίσεις και η διαχείριση τους καθίσταται αρκετά εύκολη. Ένα μεγάλο μειονέκτημά τους είναι ότι ανεξάρτητα το πώς τα έχουν διαμορφώσει οι δημιουργοί τους, οι εξομοιωμένοι πόροι τείνουν να συμπεριφέρονται διαφορετικά από τους πραγματικούς. Και αυτή η διαφοροποίηση θα μπορούσε πιθανόν να υποψιάσει τον επιτιθέμενο ότι πρόκειται για honeypot και αυτός να τερματίσει την επίθεσή του προτού μπορέσουμε να εξάγουμε συμπεράσματα για αυτή. Για παράδειγμα το Conpot που αποτελεί honeypot που εξομοιώνει βιομηχανικά πρωτόκολλα όπως το Modbus για αρκετό καιρό εξομοίωνε μόνο τα services, αλλά δεν έδειχνε κανένα σχετικό λειτουργικό σύστημα. Έτσι ένας επιτιθέμενος κάνοντας os scan με nmap μπορούσε εύκολα να εντοπίσει ότι δεν πρόκειται για πραγματικό σύστημα. Η λύση ήρθε με το oschameleon το οποίο γράφτηκε από τους δημιουργούς, προκειμένου να εξομοιώνεται και το λειτουργικό. Τέλος για τα 0-day vulnerabilities, δηλαδή κενά ασφαλείας «της τελευταίας μέρας», είναι δύσκολο να εξομοιωθούν από τα low interaction honeypots.

3.1.3.2 Honeypots υψηλής Αλληλεπίδρασης (High Interaction Honeypots)

Η κύρια διαφορά ανάμεσα στα χαμηλής και υψηλής αλληλεπίδρασης honeypots είναι ότι τα υψηλής αλληλεπίδρασης παρέχουν πραγματικά λειτουργικά συστήματα και πόρους και δεν τους εξομοιώνουν. Δηλαδή πρόκειται για πραγματικά συστήματα με τη μόνη διαφορά ότι δεν πρόκειται για συστήματα παραγωγής, αλλά σαν σκοπό έχουν να γίνονται στόχος επιθέσεων. Ως εκ τούτου το κύριο πλεονέκτημά τους είναι ότι είναι πολύ δύσκολο να εντοπίσει ο επιτιθέμενος ότι πρόκειται για honeypots και έτσι μπορούμε εύκολα να καταγράψουμε τις κινήσεις του. Επίσης τα honeypots υψηλής αλληλεπίδρασης δεν εντοπίζουν μόνο τις επιθέσεις, αλλά μπορούν να δώσουν και πληροφορίες στον διαχειριστή για τα προγράμματα ή και τις εντολές που δόθηκαν από τον επιτιθέμενο. Επίσης δίνεται η δυνατότητα παρατήρησης 0-day vulnerabilities. Το μεγαλύτερο μειονέκτημά τους είναι ότι σε περίπτωση που επιτυχημένα ο επιτιθέμενος καταφέρει να αποκτήσει υπό τον έλεγχό του το honeypot, μπορεί να χρησιμοποιηθεί για επιθέσεις σε άλλα μηχανήματα.



Low-High interaction honeypot, πηγή [19].

3.1.3.3 Υβριδικά Honeypots (Hybrid Honeypots)

Τα υβριδικά honeypots όπως και το όνομά τους υποδηλώνει συνδυάζουν χαρακτηριστικά από τα honeypots χαμηλής και υψηλής αλληλεπίδρασης και εκμεταλλεύονται ιδιότητες και οφέλη και των δύο κατηγοριών. Χαρακτηριστικό παράδειγμα είναι το honeybrid.

Το honeybrid [20],[21],[22] είναι μια δικτυακή εφαρμογή η οποία έχει σχεδιαστεί για να

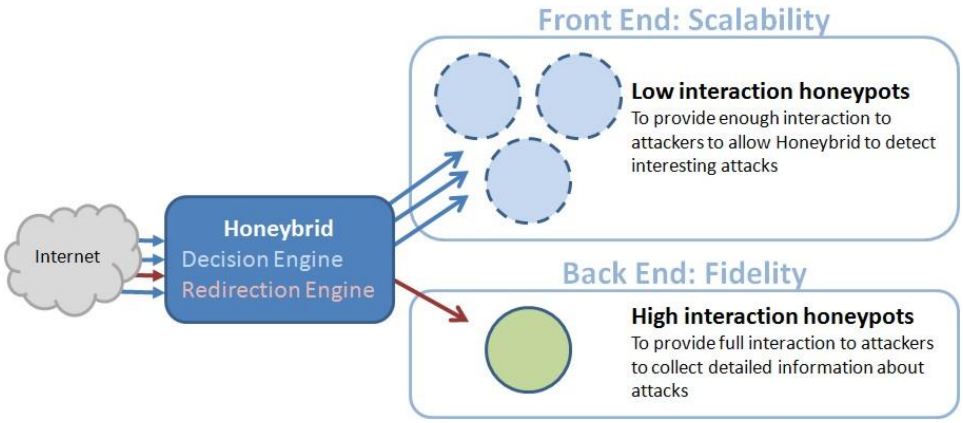
- Διαχειρίζεται honeynets
- Παρέχει την υβριδική λειτουργικότητα του συνδυασμού honeypots χαμηλής και υψηλής αλληλεπίδρασης.

Η πρώτη λειτουργικότητα επιτυγχάνεται μέσω μια μηχανής απόφασης (Decision Engine) η οποία επιτρέπει στους χρήστες να φιλτράρουν όλη την εισερχόμενη κίνηση βασιζόμενοι σε κριτήρια όπως την IP διεύθυνση αποστολέα ή κάποιο περιεχόμενο του πακέτου και μία μηχανή ελέγχου (Control Engine) που αυτόματα περιορίζει την εξερχόμενη κίνηση από τα μηχανήματα που έχουν δεχθεί επίθεση και βρίσκονται υπό τον έλεγχο των επιτιθέμενων.

Η δεύτερη λειτουργικότητα επιτυγχάνεται μέσω μια μηχανής ανακατεύθυνσης (Redirection Engine) η οποία ανακατευθύνει τις συνδέσεις TCP ή UDP από τον πρωτεύον παραλήπτη στον δευτερεύον.

Δύο ενδεικτικοί τρόποι με τους οποίους το honeybrid μπορεί να κάνει διαχείριση σε ένα δίκτυο από honeypots.

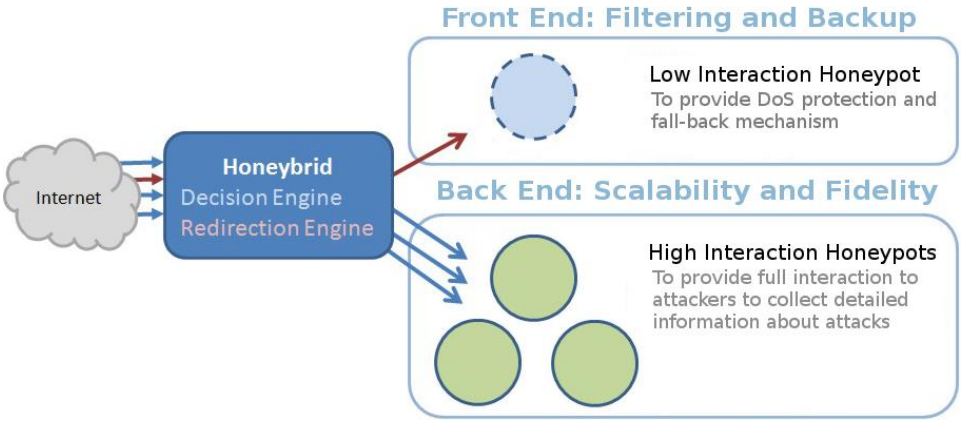
1)



Σχήμα, πηγή [20].

Σε αυτή την περίπτωση έχουμε μια ομάδα από honeypots χαμηλής αλληλεπίδρασης αρκετά έτσι ώστε να προσελκύσουν και να προσφέρουν μια βασική αλληλεπίδραση στους επιτιθέμενους. Μετά με βάση συγκεκριμένα κριτήρια, γίνεται η ανακατεύθυνση της δικτυακής κίνησης προς ένα honeypot υψηλής αλληλεπίδρασης το οποίο συλλέγει αναλυτικά στοιχεία για τον επιτιθέμενο.

2)



Σχήμα, πηγή [20].

Σε αυτή την περίπτωση έχουμε ένα honeypot χαμηλής αλληλεπίδρασης το οποίο προσφέρει προστασία σε επιθέσεις άρνησης υπηρεσίας (DoS). Μετά με βάση συγκεκριμένα κριτήρια γίνεται ανακατεύθυνση της δικτυακής κίνησης προς honeypots υψηλής αλληλεπίδρασης τα οποίο συλλέγουν αναλυτικά στοιχεία για τον επιτιθέμενο.

Το honeybrid αποτελείται από 4 στοιχεία:

- **Decision engine**: Για να αποφασιστεί ποιο είδος κίνησης πηγαίνει σε ποιο honeypot
- **Redirection engine**: Για να αποφασιστεί εάν κάποια δικτυακή κίνηση πρέπει να ανακατευθυνθεί για επιπλέον ανάλυση.
- **Control engine**: Για να περιοριστεί εξερχόμενη από το δίκτυο κίνηση από μηχανήματα που είναι υπό τον έλεγχο των επιτιθέμενων.
- **Log engine**: Για να κρατηθεί εκτενής καταγραφή της κίνησης που επεξεργάστηκε.

Αυτά τα 4 στοιχεία σχετίζονται άμεσα με τις προδιαγραφές ενός πειράματος με honeypots. Κάθε φορά πρέπει να σκεφτόμαστε το είδος της κίνησης την οποία θέλουμε να αναλύσουμε και πώς θα το κάνουμε αυτό.

Υπάρχουν συγκεκριμένοι κανόνες:

- **Filter rule**: Καθορίζει χρησιμοποιώντας σύνταξη του tcpdump το είδος της κίνησης το οποίο θα πρέπει να χειριστεί ο στόχος.
- **Frontend rule**: Καθορίζει ποιο honeypot πρέπει πρώτα να αλληλεπιδράσει με την εισερχόμενη κίνηση και με ποια κριτήρια θα δεχτεί την εισερχόμενη κίνηση.
- **Backend rule**: Καθορίζει σε ποιο honeypot θα γίνει ανακατεύθυνση της κίνησης για πιο εκτενή ανάλυση και με ποια κριτήρια θα γίνει αυτή η ανακατεύθυνση.
- **Control rule**: Καθορίζει πώς να περιορίσει την εξερχόμενη κίνηση που δημιουργείται από τα honeypots.

Τα κριτήρια για την αποδοχή, την ανακατεύθυνση και τον έλεγχο της δικτυακής κίνησης βασίζονται σε ένα module. Τα modules είναι συναρτήσεις οι οποίες μπορούν να αναλύουν τα πακέτα και αποφασίζουν αν εκπληρώνουν συγκεκριμένα κριτήρια. Τα modules που υποστηρίζει μέχρι τώρα το honeybrid είναι:

- **Mod_random()**: Είναι ένα module το οποίο τυχαία αποδέχεται ένα πακέτο. Η πιθανότητα δίνεται από μία παράμετρο, για παράδειγμα το mod_random(0.1) θα αποδέχεται το 10% των πακέτων.
- **Mod_yesno()**: Είναι ένα module το οποίο πάντα αποδέχεται ή απορρίπτει πακέτα.
- **Mod_counter()**: Είναι ένα module το οποίο αποδέχεται πακέτα μετά από έναν ορισμένο αριθμό πακέτων τα οποία θα ληφθούν. Για παράδειγμα το mod_counter(3) θα πραγματοποιηθεί εάν αν η έχουν ληφθεί ήδη 3 πακέτα για την συγκεκριμένη σύνδεση.
- **Mod_source()**: Είναι ένα module το οποίο αποδέχεται πακέτα μόνο από νέες IP διευθύνσεις. Μία IP διεύθυνση η οποία θα προσπαθήσει να συνδεθεί πολλές φορές στο honeynet θα απορριφθεί μετά την δεύτερη προσπάθεια.

- **Mod_hash()**: Είναι ένα module το οποίο αποδέχεται πακέτα που έχουν καινούριο περιεχόμενο, το οποίο σημαίνει που το περιεχόμενό τους δεν έχει ξαναπεράσει από ανάλυση. Αυτό το module δουλεύει υπολογίζοντας μια τιμή hash για κάθε περιεχόμενο πακέτου που αναλύεται και κρατώντας μια βάση δεδομένων με όλα τα γνωστά περιεχόμενα πακέτων.
- **Mod_control()**: Είναι ένα module για να περιορίσει το όριο των πακέτων βασισμένο στην IP αποστολέα. Τα πακέτα απορρίπτονται όταν μια IP αποστολέα έχει στείλει περισσότερα πακέτα από έναν καθορισμένο αριθμό.

3.2 Υλοποίηση hybrid honeypot με OpenFlow

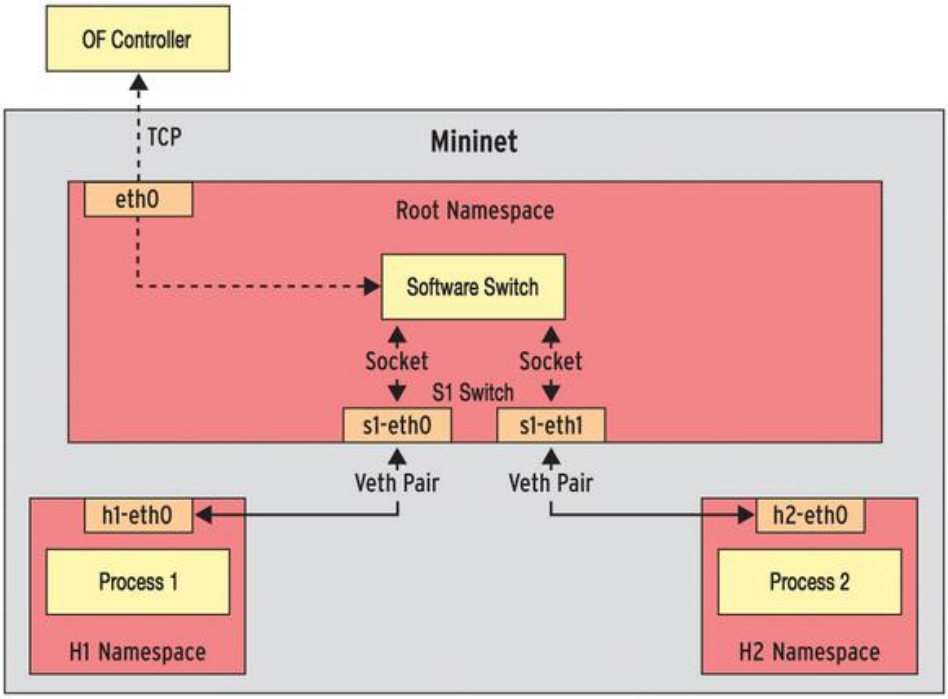
3.2.1 Εισαγωγή στο mininet.

Για λόγους πληρότητας γίνεται μια μικρή εισαγωγή στο περιβάλλον στο οποίο θα πραγματοποιηθούν οι διάφορες υλοποιήσεις.

Το πρόγραμμα το οποίο χρησιμοποιήθηκε για την πραγματοποίηση του πειραματικού μέρους είναι το mininet [23] το οποίο αποτελεί έναν εξομοιωτή δικτύου. Έχει τη δυνατότητα να εκτελεί ταυτόχρονα ένα σύνολο από τερματικά, μεταγωγείς, συνδέσεις και ελεγκτές στον ίδιο πυρήνα linux. Το mininet παράγει εικονικά μηχανήματα τα οποία μοιράζονται μεγάλο μέρος του αρχικού συστήματος όπως το σύστημα αρχείων, βιβλιοθήκες και άλλα προγράμματα. Κάθε μηχάνημα περιορίζεται σε μια απλή διεργασία κελύφους με το δικό της network namespace όπου περιλαμβάνει διεύθυνση IP, πίνακα δρομολόγησης, arp cache και άλλες δικτυακές παραμέτρους.

Θα δείξουμε την λειτουργία του mininet υποθέτοντας 2 hosts, έναν μεταγωγέα και έναν ελεγκτή.

Αρχικά τρέχουμε το πρόγραμμα mn μέσα στο root network namespace και στη συνέχεια δημιουργούνται δύο bash processes, ένα για τον host1 και ένα για τον host2. Μετά οι διεργασίες τοποθετούνται σε ξεχωριστά network namespaces, δημιουργούνται virtual ethernet pairs και παραμετροποιούνται κατάλληλα προκειμένου να επικοινωνήσει το root namespace με τα 2 νέα network namespaces μέσω αυτών. Από προεπιλογή ο μεταγωγέας τοποθετείται στο root namespace. Ο controller μπορεί να είναι είτε εξωτερικός και πρέπει να καθορίσουμε την IP διεύθυνση και την θύρα του ή μπορεί να είναι στο ίδιο μηχάνημα (loopback address), όπως στα παραδείγματα που θα υλοποιήσουμε.



Εσωτερική δομή mininet για 2 hosts, πηγή [24]

Με το mininet μπορεί κάποιος γράφοντας σε γλώσσα python [25] να φτιάξει δικές του δικτυακές τοπολογίες από hosts, switches και συνδέσεων μεταξύ τους, όπως επίσης και να παραμετροποιήσει τις μεταξύ τους συνδέσεις, όπως την χωρητικότητα, καθυστέρηση, ανεκτικότητα απωλειών πακέτων με μέγιστο μέγεθος ουράς τα 100 πακέτα. Επίσης διαθέτει εντολές για τη μέτρηση των παραμέτρων του δικτύου, όπως το εύρος ζώνης, την καθυστέρηση, το μέγεθος της ουράς αναμονής, στατιστικά tcp όπως και τη χρήση της CPU.

Διαθέτει εύκολα δημιουργούμενες τοπολογίες μέσω του CLI.

- sudo mn --topo single,N στην οποία έχουμε ένα switch και N hosts
- sudo mn --topo linear,N στην οποία N switches συνδέονται σε ευθεία γραμμή, κάθε ένα με έναν host
- sudo mn --topo tree,depth=M,fanout=N, όπου έχουμε δενδρική δομή με βάθος M και αριθμό κόμβων που δείχνει ο κάθε κόμβος, N

Βασικές εντολές του mininet είναι οι:

- nodes: δείχνει όλους του κόμβους
- dump: δείχνει πληροφορίες για τους κόμβους
- net: δείχνει της συνδέσεις ανάμεσα στους κόμβους
- xterm: ανοίγει ένα τερματικό για κάθε host
- help: δείχνει όλες τις εντολές
- exit: πραγματοποιείται έξοδος από το περιβάλλον mininet

3.2.2 Υλοποίηση υβριδικού honeypot στον POX.

Τροποποιούμε το l2_learning switch του POX το οποίο χρησιμοποιείται ευρέως σε πειράματα OpenFlow [26], έτσι ώστε μετά από έναν συγκεκριμένο αριθμό packet in events για το honeypot χαμηλής αλληλεπίδρασης, όλη η κίνηση των πακέτων ανακατευθύνεται προς το honeypot υψηλής αλληλεπίδρασης, όπου τα 2 honeypots είναι δύο τυχαίοι hosts του mininet. Αρχικοποιούμε στον constructor της κλάσης LeaningSwitch το **self.counter=0** και έπειτα η μέθοδος _handle_PacketIn έχει ως εξής:

```
def _handle_PacketIn (self, event):

  counter_limit=6
  low_ip="10.0.0.2"
  high_ip="10.0.0.3"
  low_mac="26:62:8c:24:bf:b2"
  high_mac="32:b2:a0:2b:f9:03"
  high_port=3
  idle_t=30
  hard_t=30

  packet = event.parsed
  myport = event.port
  if packet.type == ethernet.IP_TYPE:
    ipv4_packet=event.parsed.find('ipv4')
    src_ip=ipv4_packet.srcip
    dst_ip=ipv4_packet.dstip
    log.info("destination ip")
    if str(dst_ip) == low_ip:
     self.counter+=1
     print "the counter is %d" % self.counter
     if self.counter==counter_limit:
       self.counter=0
       msg = of.ofp_flow_mod()
       msg.priority = 65535
       msg.match.dl_type=0x800
       msg.idle_timeout = idle_t
       msg.hard_timeout = hard_t
       msg.match.nw_dst = IPAddr(low_ip)
       msg.actions.append(of.ofp_action_nw_addr.set_dst(IPAddr(high_ip)))
       msg.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(high_mac)))
       msg.actions.append(of.ofp_action_output(port = high_port))
       self.connection.send(msg)
       msg = of.ofp_flow_mod()
       msg.priority = 65535
       msg.match.dl_type=0x800
       msg.idle_timeout = idle_t
       msg.hard_timeout = hard_t
       msg.match.nw_dst = IPAddr(src_ip)
       msg.actions.append(of.ofp_action_nw_addr.set_src(IPAddr(low_ip)))
       msg.actions.append(of.ofp_action_dl_addr.set_src(EthAddr(low_mac)))
       msg.actions.append(of.ofp_action_output(port = myport))
       self.connection.send(msg)
```

Επιπλέον βιβλιοθήκες που χρειάστηκαν:

```
from pox.lib.packet.ethernet import ethernet
from pox.lib.packet.ipv4 import ipv4
from pox.lib.addresses import IPAddr, EthAddr
```

Χρησιμοποιούνται οι συγκεκριμένες βιβλιοθήκες προκειμένου να μπορέσουμε να επεξεργαστούμε στο εσωτερικό του πακέτου διευθύνσεις mac και IP.

Αρχικοποιήσεις του κώδικα:

- counter_limit: για το πόσα packet in events θα γίνουν στον ελεγκτή από έναν συγκεκριμένο host μέχρι να κάνει ανακατεύθυνση της κίνησης
- low_ip: η IP διεύθυνση του honeypot χαμηλής αλληλεπίδρασης
- high_ip: η IP διεύθυνση του honeypot υψηλής αλληλεπίδρασης
- low_mac: η mac διεύθυνση του honeypot χαμηλής αλληλεπίδρασης
- high_mac: η mac διεύθυνση του honeypot υψηλής αλληλεπίδρασης
- high_port: η θύρα του μεταγωγέα στην οποία είναι συνδεδεμένο το honeypot υψηλής αλληλεπίδρασης
- idle_t: είναι το idle timeout για τους κανόνες, δηλαδή σε πόσο χρόνο θα διαγραφούν από τον μεταγωγέα αν δεν υπάρχει κίνηση που να ικανοποιεί την αντιστοίχιση στον πίνακα ροής
- hard_t: είναι το hard timeout για τους κανόνες, δηλαδή σε πόσο χρόνο θα διαγραφούν από τον μεταγωγέα οι κανόνες, οτιδήποτε και αν συμβεί

Η μέθοδος _handle_PacketIn υλοποιεί τον εξής αλγόριθμο:

- Αποθηκεύει σε μεταβλητές την θύρα του μεταγωγέα από την οποία προήλθε το εισερχόμενο πακέτο και την IP διεύθυνση αποστολέα και παραλήπτη.
- Αν ο παραλήπτης είναι το honeypot χαμηλής αλληλεπίδρασης, τότε ξεκινά να αυξάνεται ο μετρητής.
- Όταν ο μετρητής φτάσει ένα συγκεκριμένο όριο packet-in events από τον επιτιθέμενο, μπαίνει σε λειτουργία ο μηχανισμός ανακατεύθυνσης με την εγκατάσταση συγκεκριμένων κανόνων στον μεταγωγέα (**Flow Modification Message**).

Flow Modification Message

1) **Εάν η διεύθυνση προορισμού είναι του honeypot χαμηλής αλληλεπίδρασης:**

- Αλλάζει την IP διεύθυνση προορισμού από την IP του honeypot χαμηλής αλληλεπίδρασης στου honeypot υψηλής αλληλεπίδρασης.
- Αλλάζει την mac διεύθυνση προορισμού από την mac του honeypot χαμηλής αλληλεπίδρασης στου honeypot υψηλής αλληλεπίδρασης.
- Προωθεί το πακέτο που προοριζόταν για το honeypot χαμηλής αλληλεπίδρασης στην θύρα στην οποία είναι συνδεδεμένη το honeypot υψηλής αλληλεπίδρασης.

2) **Εάν η διεύθυνση προορισμού είναι αυτή του επιτιθέμενου, έτσι όπως έχει αποθηκευτεί στη μεταβλητή src_ip:**

- Αλλάζει την IP διεύθυνση του honeypot υψηλής αλληλεπίδρασης, σε αυτή του χαμηλής.
- Αλλάζει την mac διεύθυνση του honeypot υψηλής αλληλεπίδρασης, σε αυτή του χαμηλής.
- Στέλνει το πακέτο από την θύρα του μεταγωγέα myport, έτσι όπως έχει αποθηκευτεί στην αρχή.

Σημαντικές επισημάνσεις σχετικά με την υλοποίηση:

Τα honeypots δεν υποτίθεται ότι πρέπει να δέχονται κίνηση δεδομένων, οπότε η οποιαδήποτε κίνηση καταγραφεί θεωρείται κακόβουλη, άρα και κάθε packet in event στον POX θεωρείται αποτέλεσμα επίθεσης. Επίσης στόχος μας ήταν να δείξουμε την ανακατεύθυνση της κίνησης και για αυτό δεν εγκαταστήσαμε σχετικά honeypots.

Τα priorities έχουν τεθεί στην τιμή 65535 που αντιπροσωπεύει full match, δηλαδή και τα 12 πεδία της τούπλας της εγγραφής. Αυτό οφείλεται στην λογική του l2_learning που είναι έτσι γραμμένο ώστε να κάνει full match, στην περίπτωση του l2_pairs θα μπορούσαμε να θέσουμε στα priorities σε οποιεσδήποτε μεγαλύτερες τιμές από τα priorities των flows κανονικής λειτουργίας. Επιπλέον η σειρά με την οποία αναλύεται το πακέτο στον κώδικα, γίνεται σύμφωνα με τη σειρά που έχει υποδειχτεί στα διαγράμματα ροής της αντιστοίχισης του OpenFlow.

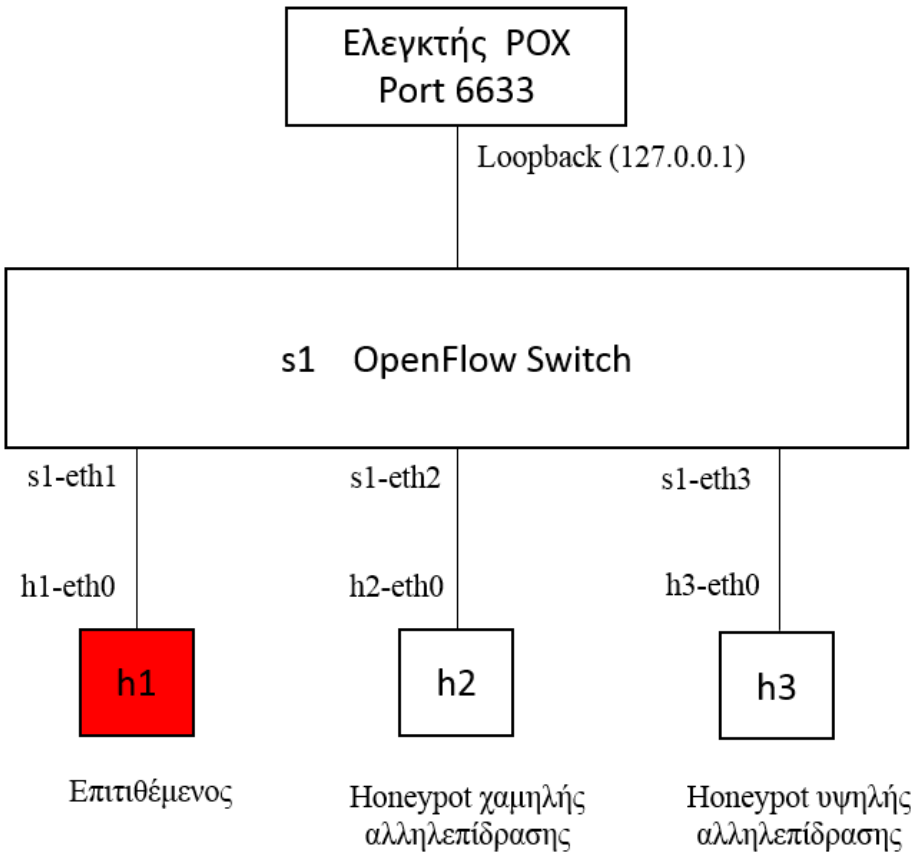
Ανακατεύθυνση ροής μπορεί να γίνει σε ICMP, UDP και TCP traffic με το μόνη παρατήρηση ότι σε TCP traffic δεν μπορούμε να αναπαράγουμε το 3-way handshake και άρα η συγκεκριμένη σύνδεση η οποία υπάρχει τη στιγμή της ανακατεύθυνσης θα διακοπεί.

Εμείς θα υποθέσουμε ότι στο mininet ο h1 είναι ο επιτιθέμενος, ο h2 το honeypot χαμηλής αλληλεπίδρασης και ο h3 το honeypot υψηλής αλληλεπίδρασης στο οποίο θα γίνει η ανακατεύθυνση της κίνησης. Στην αρχή το πραγματοποιούμε για ICMP κίνηση και έπειτα και για UDP με τη βοήθεια ενός UDP honeypot για να έχουμε πιο καλά αποτελέσματα.

Το πόσο γρήγορα θα γίνει η ανακατεύθυνση εξαρτάται από δύο παράγοντες. Ο πρώτος είναι η μεταβλητή counter_limit που περιγράφει τα πόσα packet in events χρειάζονται να πραγματοποιηθούν στον ελεγκτή προκειμένου να γίνει η ανακατεύθυνση και οι μεταβλητές msg.idle_timeout και msg.hard_timeout οι οποίες περιγράφουν για πόσο χρονικό διάστημα εγκαθίστανται εγγραφές στον μεταγωγέα σε κανονική λειτουργία και έτσι δεν στέλνονται πακέτα στον ελεγκτή.

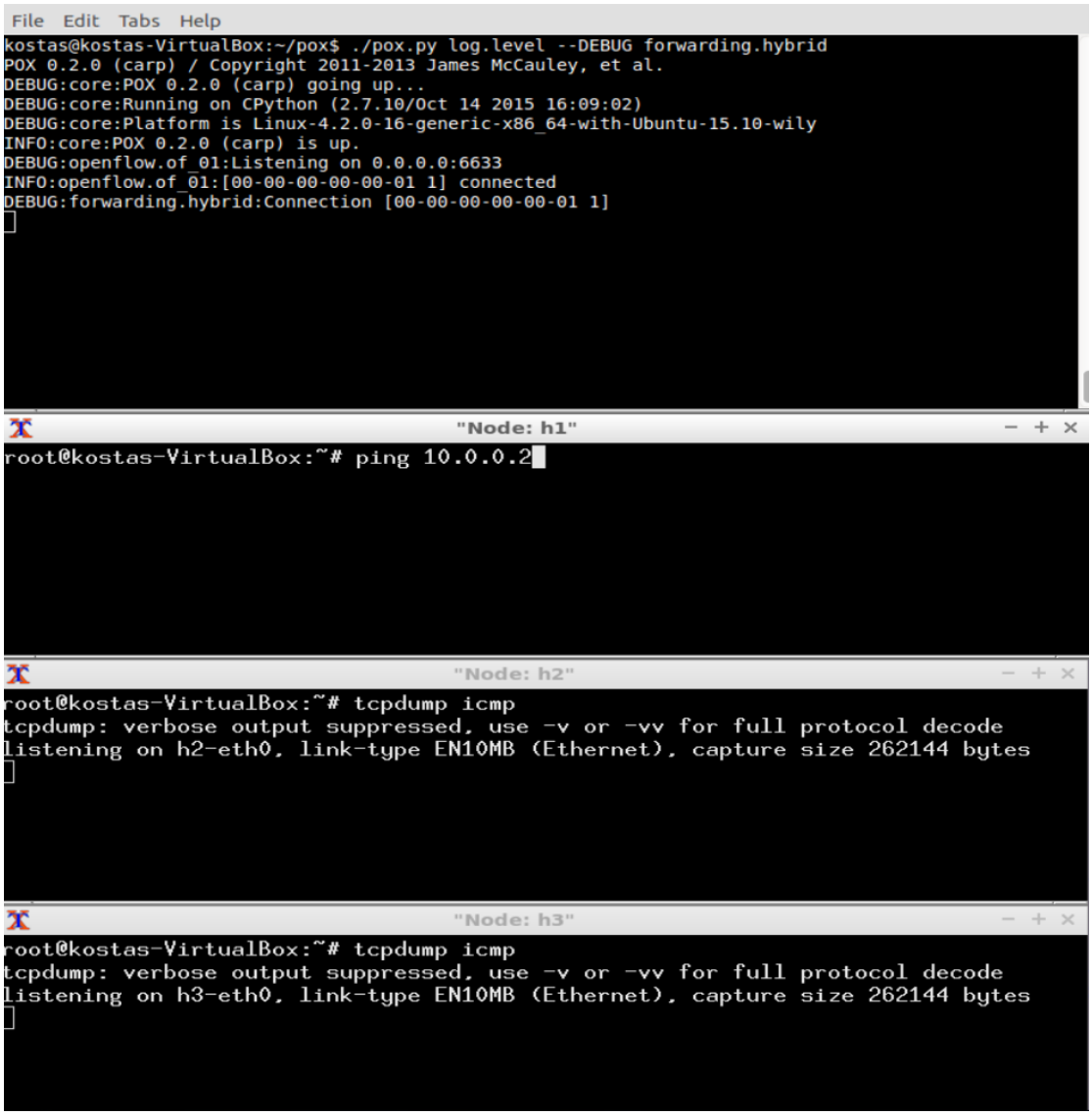
Η τοπολογία και σύνδεση με τον ελεγκτή γίνεται με την εντολή
sudo mn --controller=remote,port=6633 --topo single,3
για ελεγκτή στο τοπικό μηχάνημα και έναν μεταγωγέα με 3 hosts συνδεδεμένους σε αυτό. Παρατίθεται και σχήμα της τοπολογίας.

Οι εγγραφές που είναι αποθηκευμένες στον μεταγωγέα εμφανίζονται γράφοντας
dpctl dump-flow στο mininet.

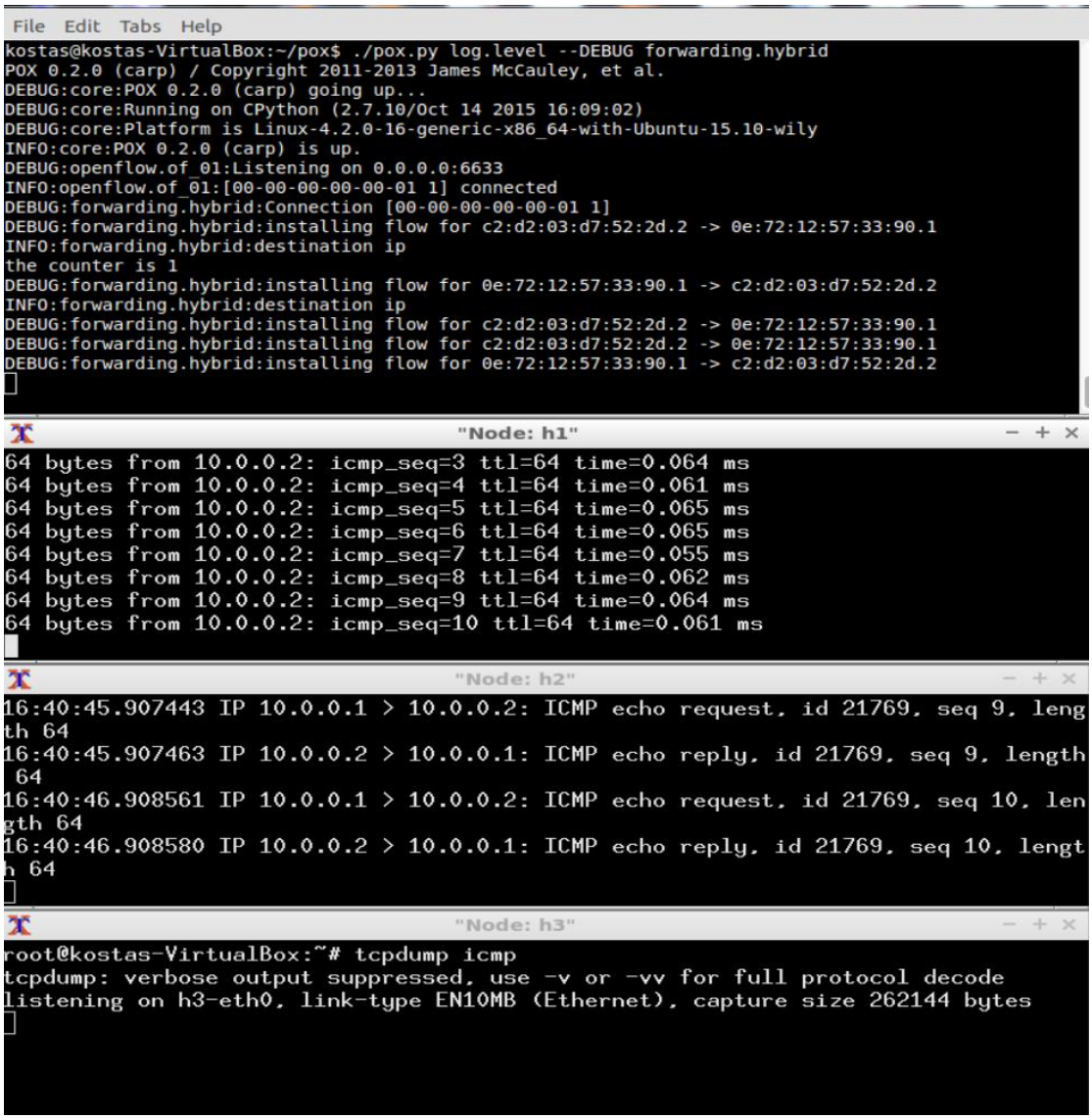


Αρχικοποίηση

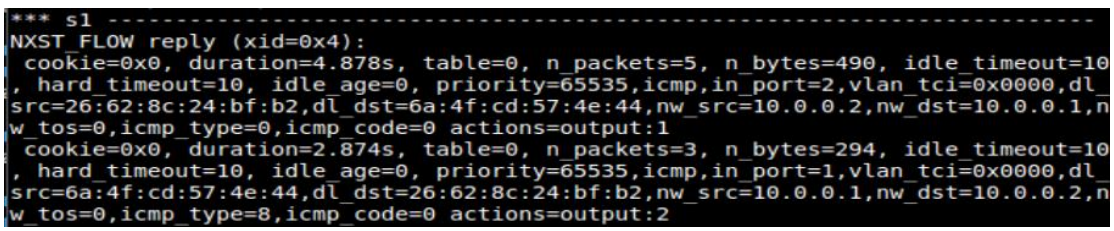
(Αντί για log.debug μήνυμα κάναμε print για να ναι πιο ευδιάκριτος ο αριθμός των packet-in events)



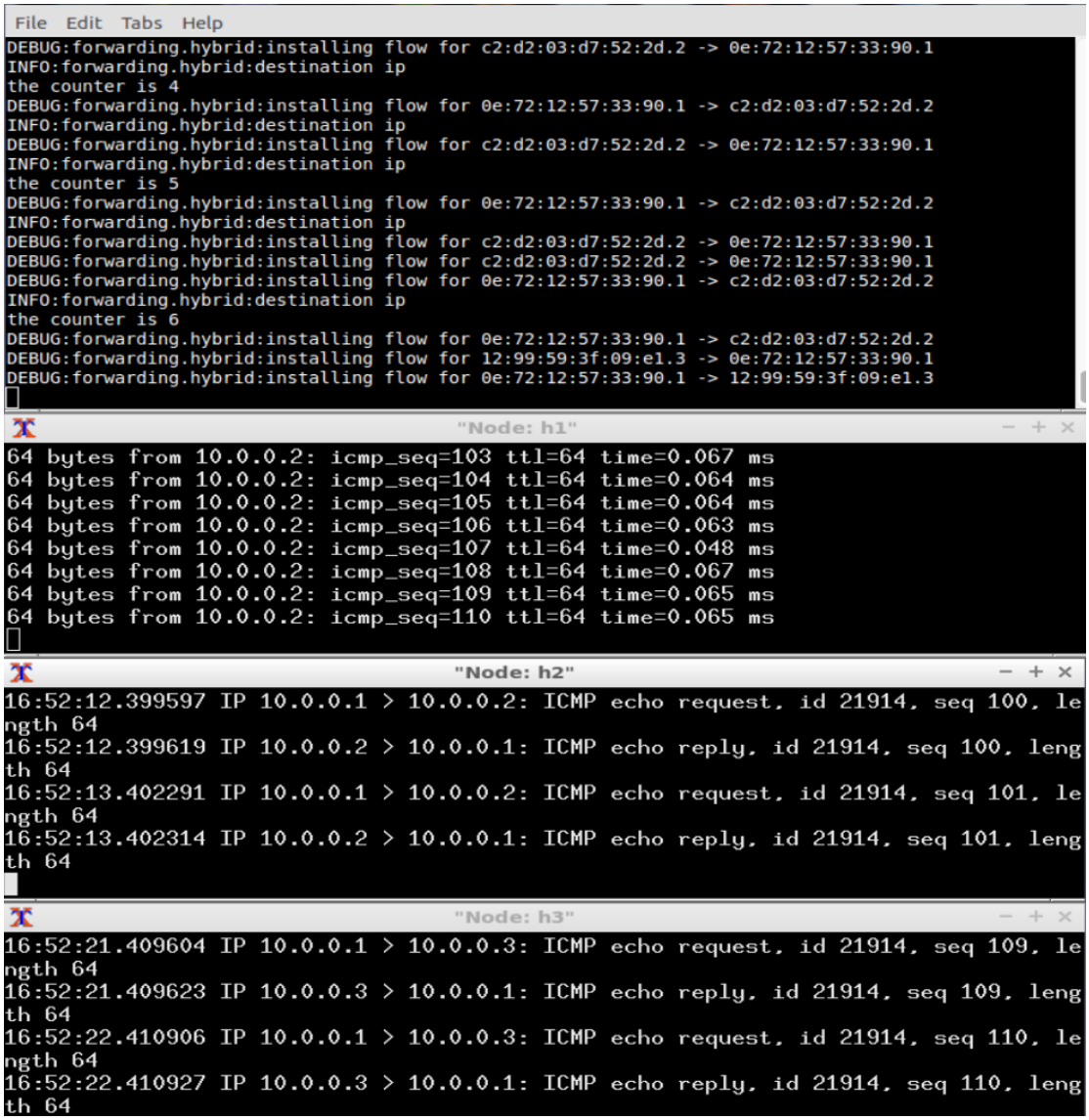
Ping στο honeypot χαμηλής αλληλεπίδρασης (h2)



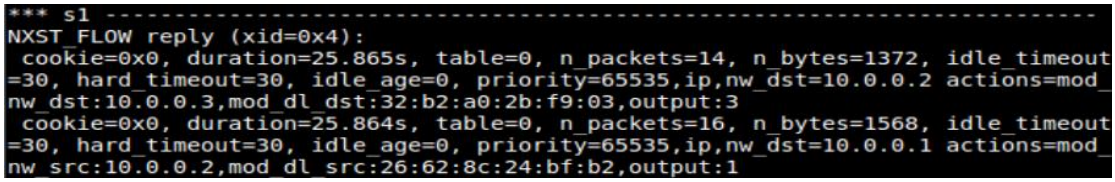
Και οι αντίστοιχες εγγραφές στο switch.



Ανακατεύθυνση της κίνησης στο honeypot υψηλής αλληλεπίδρασης (h3) μετά από 6 packet in events από τον ίδιο επιτιθέμενο (h1).



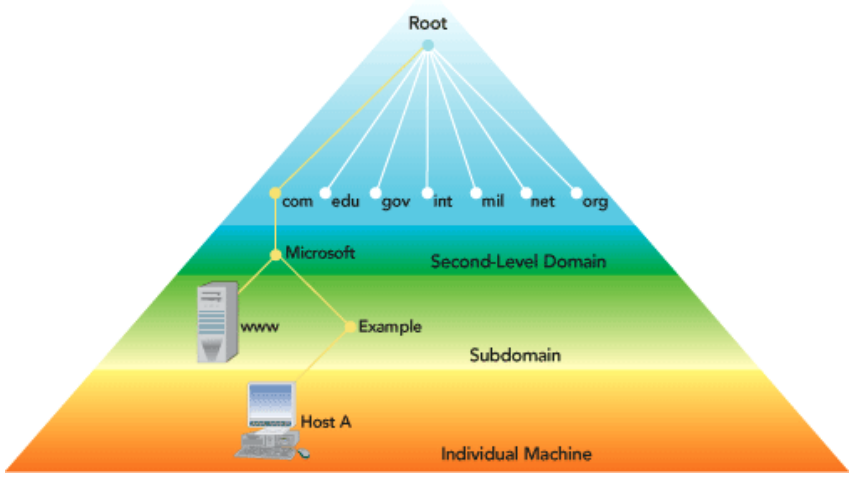
Και οι αντίστοιχες εγγραφές στο switch.
(Ο επιτιθέμενος δεν βλέπει καμία διαφορά, καθ' όλη τη διάρκεια νομίζει ότι κάνει επίθεση στο h2.)



3.2.3 Ανακατεύθυνση DNS κίνησης.

Θα χρησιμοποιήσουμε το περιβάλλον mininet ξανά σε αυτή την προσομοίωση και επίσης γνωρίζουμε ότι οι hosts μοιράζονται το file system. Εγκαθιστούμε το dns.py που είναι DNS honeypot και το minidns 0.3 που είναι απλοϊκός DNS server. Παρακάτω παρατίθεται και σύντομη εισαγωγή στο DNS.

Το DNS (Domain Name Sytem)[27],[28] είναι ένα ιεραρχικό σύστημα ονομάτων το οποίο συσχετίζει διάφορα ονόματα που έχουν αποδοθεί στους χρήστες του διαδικτύου με τις διευθύνσεις IP. Οι DNS servers αναλαμβάνουν να κάνουν αυτή την αντιστοίχιση και η δουλειά τους είναι να απαντάνε σε DNS ερωτήματα (DNS queries) που γίνονται από τους χρήστες. Τα ονόματα χώρου (domain names) αποτελούνται από ακολουθίες χαρακτήρων οι οποίοι διαχωρίζονται μεταξύ τους με τελείες και κάθε τελεία αντιπροσωπεύει αλλαγή ευθύνης στη διαχείριση του χώρου ονομάτων. Οι πληροφορίες σχετικά με την αντιστοίχιση ονομάτων σε IP διευθύνσεις και γενικά οποιαδήποτε άλλη πληροφορία αφορά σε ονόματα χώρου, εμφανίζεται μέσα σε μια ζώνη υπό τη μορφή εγγραφών (Recource Records). Επίσης οι DNS servers ακολουθούν συγκεκριμένη ιεραρχία χωρισμένη όπως για παράδειγμα απεικονίζεται παρακάτω.



Παράδειγμα ιεραρχίας DNS, πηγή [27].

Τύποι DNS servers:

- **Authoritative DNS**

Ένας authoritative DNS server είναι ένας διακομιστής ο οποίος απαντά σε ερωτήματα για τις ζώνες για τις οποίες είναι υπεύθυνος. Υπάρχουν οι master DNS οι οποίοι έχουν την πιο πρόσφατη πληροφορία στις DNS εγγραφές τους και οι οποίοι διαμοιράζουν τα αρχεία στου slaves DNS μέσω της διαδικασίας zone transfer. Σε περίπτωση που πρόκειται να γίνει αλλαγή στις εγγραφές της ζώνης οι master ενημερώνονται πρώτοι και μετά ενημερώνουν και τους slave, οι οποίοι αποτελούν εφεδρικά μηχανήματα των master.

- **Recursive and caching name**

Στη θεωρία οι authoritative servers φτάνουν για τη λειτουργία του internet. Όμως με μόνο τους authoritative servers να λειτουργούν, κάθε DNS ερώτημα πρέπει να ξεκινά με αναδρομικά ερωτήματα από την root ζώνη του DNS και κάθε υπολογιστικό σύστημα θα έπρεπε να έχει τη δυνατότητα να κάνει αναδρομικά ερωτήματα. Για να βελτιωθεί η επίδοση, για να περιοριστεί η DNS κίνηση στο internet και να αυξηθεί η απόδοση των εφαρμογών των χρηστών, υπάρχουν οι DNS cache servers οι οποίοι κρατάνε εγγραφές DNS για ένα συγκεκριμένο χρονικό διάστημα που καθορίζεται από την τιμή της time to live παραμέτρου. Επίσης οι caching DNS servers (DNS caches), εκτελούν αναδρομικό αλγόριθμο αν χρειάζεται ξεκινώντας από τον DNS root, μέχρι τους authoritative servers του κάθε domain. Ο συνδυασμός του DNS caching και των αναδρομικών ερωτημάτων, συνήθως πραγματοποιείται μαζί, αλλά μπορεί και ξεχωριστά αν υπάρχουν συγκεκριμένες απαιτήσεις.

- **Forwarding**

Οι forwarding DNS servers, είναι διακομιστές οι οποίοι προωθούν τα DNS ερωτήματα σε recursive διακομιστές ονομάτων και τις απαντήσεις τις αποθηκεύουν σε cache. Το πλεονέκτημα είναι ότι μπορεί το σύστημα να εκμεταλλευτεί την τοπική cache, χωρίς να χρειαστεί να κάνει τη δουλειά των αναδρομικών ερωτήσεων και να την αναθέσει σε κάποιο άλλο σύστημα, το οποίο θα αναλάβει το συγκεκριμένο φόρτο.

Εγγραφές DNS - Resource Records

Οι εγγραφές πόρων αποθηκεύονται σε αρχεία (αρχεία ζώνης) στους εξυπηρετητές ονομάτων.

Τύπος RR: (name, type, class, ttl, rdata)

SOA (Start of Authority): Καθορίζει την επίσημη πληροφορία για μια ζώνη DNS όπως τον κύριο εξυπηρετητή ονομάτων, τη διεύθυνση email του διαχειριστή της περιοχής και πολλά χρονόμετρα σχετιζόμενα με την ανανέωση της πληροφορίας ζώνης.

NS: Με την εγγραφή Name Server καθορίζονται οι authoritative servers και θα πρέπει να ορίζονται τουλάχιστον δύο, ένας master και ένας slave.

- name είναι η περιοχή (π.χ. foo.com)
- rdata είναι η διεύθυνση ip του επίσημου name server για την περιοχή

MX: Με την εγγραφή Mail Exchange καθορίζονται οι διακομιστές e-mail οι οποίοι εξυπηρετούν την ηλεκτρονική αλληλογραφία για το συγκεκριμένο domain.

- rdata είναι το όνομα του mail server που σχετίζεται με το name

CNAME: Με την εγγραφή Canonical Name καθορίζουμε ψευδώνυμα για τα domain names.

- name είναι το ψευδώνυμο για κάποιο «επίσημο» όνομα
- rdata είναι το επίσημο (canonical) όνομα

A: Οι εγγραφές A πραγματοποιούν την αντιστοίχιση των domain names σε διευθύνσεις IPv4.

- name είναι το όνομα του host
- rdata είναι η διεύθυνση ip

AAAA: Οι εγγραφές αυτές πραγματοποιούν την αντιστοίχιση των domain names σε IPv6 διευθύνσεις.

DNS honeypot – Udpot (dns.py)

Η ιδέα πίσω από το συγκεκριμένο honeypot γραμμένο σε python, είναι ότι λειτουργεί σαν forwarding DNS server, ο οποίος στέλνει τα queries σε έναν άλλον DNS server ο οποίος και μπορεί να εξυπηρετήσει τα ερωτήματα. Στο Udpot [29] υπάρχει η δυνατότητα να προωθηθεί συγκεκριμένος αριθμός DNS μηνυμάτων από συγκεκριμένη IP διεύθυνση και για τα υπόλοιπα να λειτουργεί ως «καταβόθρα» μηνυμάτων, χωρίς να τα προωθεί.

Στις παραμέτρους που παίρνει όταν τρέχει το honeypot, το κυριότερο είναι να οριστεί η IP διεύθυνση του DNS server που θα αποκριθεί στα DNS ερωτήματα. Προαιρετικά είναι η UDP πόρτα στην οποία θα ακούει το DNS honeypot, ο αριθμός των ερωτημάτων από συγκεκριμένη IP διεύθυνση που θα προωθήσει αφού σταματήσει να προωθεί μηνύματα για πρώτη φορά, για πόσο χρόνο θα συνεχίσει να μην προωθεί, όπως επίσης και το να εμφανίζει στην οθόνη τα DNS ερωτήματα.

Minidns 0.3

Πρόκειται για έναν απλό DNS server με REST API, μόνο για να δουλεύει στην τοπική στοίβα του μηχανήματος (localhost). Το Minidns 0.3 [30] προσφέρει δυνατότητα για να κάνει αναδρομικά ερωτήματα και (recursive dns server), αλλά επίσης μπορεί να θέσει νέα authoritative domains και A records τα οποία είναι διαθέσιμα τοπικά.

Παραμετροποίηση προγραμμάτων.

Αναφορικά με την παραμετροποίηση του Minidns server και δημιουργία του A record.

minidns start
minidns add kostaschartsias.com
minidns record kostaschartsias.com a www 192.192.192.192

Για το UDpot

python dns.py –v –p 53 127.0.0.1

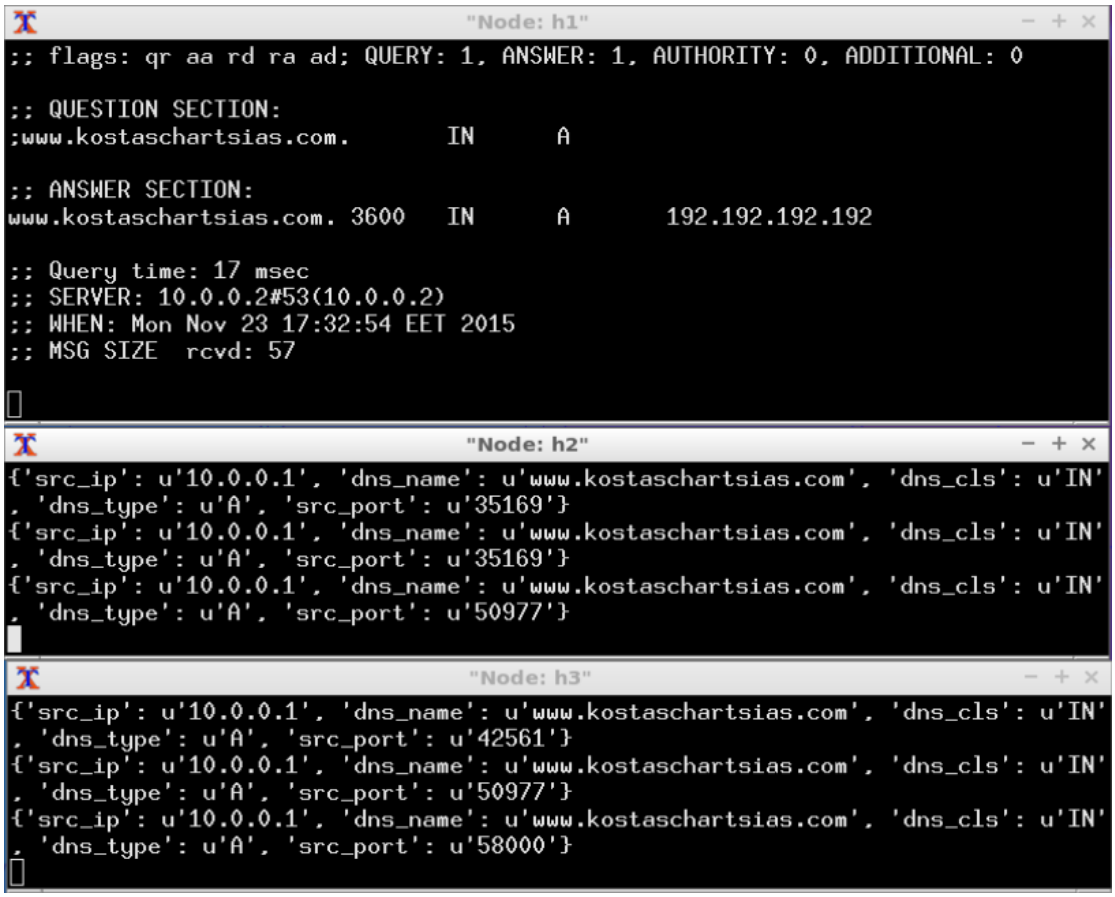
-v είναι για να βλέπουμε στην οθόνη τα dns ερωτήματα, -p 53 για να ακούει στην πόρτα 53 και στην 127.0.0.1 στέλνει τα ερωτήματα που τρέχει το minidns.

Για τα DNS requests έχουμε το εξής bash script, στο οποίο ζητάμε από τον host 2 (10.0.0.2) να μας κάνει resolve το όνομα www.kostaschartsias.com το οποίο είχαμε εισάγει στο minidns.

```
#!/bin/bash
while true
do
dig www.kostaschartsias.com@10.0.0.2
sleep 4
done
```

Αφού αφήσαμε αρκετή ώρα να τρέξει το script, είδαμε ότι όντως τα DNS queries ανακατευθύνονταν σωστά από το h2 στο h3.

Παρακάτω παρατίθεται και σχετική εικόνα έπειτα από ανακατεύθυνση από το h2 στο h3.

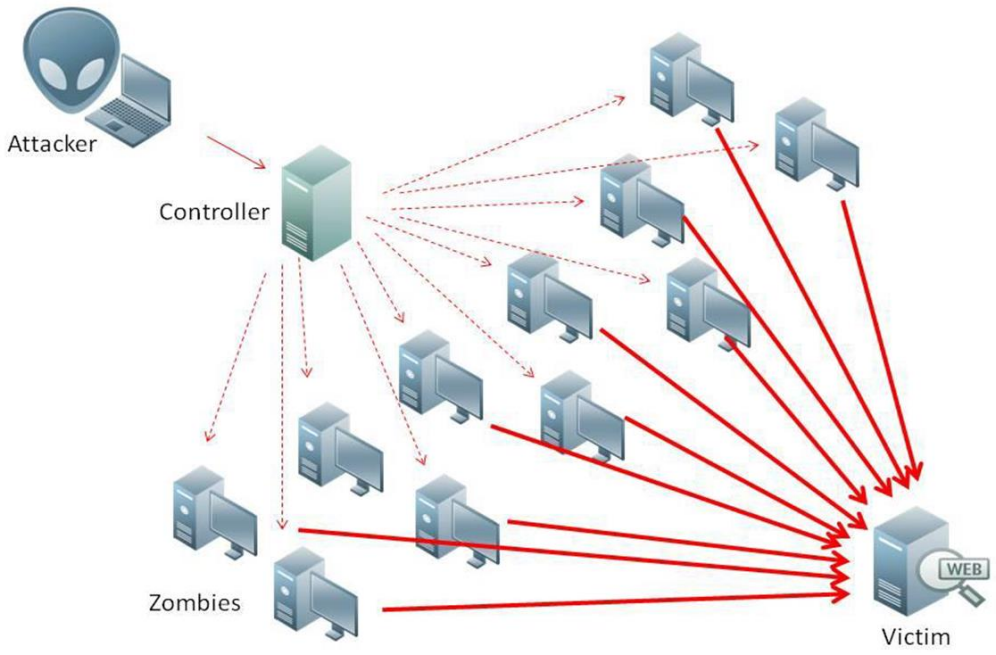


4

Εντοπισμός κατανεμημένων επιθέσεων άρνησης υπηρεσίας

4.1 Κατανεμημένες επιθέσεις άρνησης υπηρεσίας (DDoS)

Στους υπολογιστές μία επίθεση DoS (Denial of Service) είναι ένα είδος επίθεσης η οποία έχει ως σκοπό να καταστήσει έναν υπολογιστή ανίκανο να δεχτεί άλλες συνδέσεις από άλλους υπολογιστές. Μία κατανεμημένη επίθεση DoS (Distributed Denial of Service) είναι μια επίθεση DoS η οποία γίνεται από περισσότερες από μία και αρκετές φορές χιλιάδες μοναδικές διευθύνσεις IP. Στην περίπτωση του DDoS [32] ο επιτιθέμενος έχει υπό τον έλεγχό του έναν αριθμό από υπολογιστές των οποίων έχει αποκτήσει πρόσβαση, οι οποίοι ονομάζονται bots ή zombies. Οι ομάδες υπολογιστών που χρησιμοποιούνται για επιθέσεις DDoS, τα λεγόμενα botnets αποτελούν την μεγαλύτερη απειλή στο διαδίκτυο, ακόμα περισσότερο από το spam, τους ιούς ή τα worms.
Το κοινό γεγονός στις διάφορες επιθέσεις DDoS είναι η ασυνήθιστη κίνηση η οποία στέλνεται στον στόχο. Σε κανονικές συνθήκες υπάρχει ένα μοτίβο στην κίνηση του δικτύου και ένα συγκεκριμένο όριο χρήσης εύρους ζώνης. Εάν υπάρξει κάποια ξαφνική αύξηση στην δικτυακή κίνηση, στην καθυστέρηση άφιξης πακέτων, στην χρήση CPU ή κάποια πτώση στην απόδοση του δικτύου αυτό μπορεί να θεωρηθεί ως ασυνήθιστο γεγονός και το οποίο εντοπίζεται από ένα σύστημα εντοπισμού DDoS επιθέσεων.



DDoS Attack, πηγή [31]

Βασικά ήδη DDoS επιθέσεων:

- HTTP flood: Οι επιθέσεις αυτές πραγματοποιούνται με τον επιτιθέμενο να στέλνει είτε GET είτε POST requests του πρωτοκόλλου HTTP συχνά από τροποποιημένες διευθύνσεις IP και προϋποθέτουν καλή γνώση για το πώς λειτουργεί ένα site από τον επιτιθέμενο. Σε γενικές γραμμές είναι πιο δύσκολο να υλοποιηθούν σε σχέση με άλλες επιθέσεις, αλλά απαιτούν λιγότερο εύρος ζώνης και εντοπίζονται πιο δύσκολα. Λόγω του ότι τα POST requests περιλαμβάνουν παραμέτρους (παραδείγματος χάρη πρόσβαση σε βάση δεδομένων), συνήθως χρειάζονται περισσότερη επεξεργασία από τον server από ότι με το GET, οπότε είναι και πιο αποτελεσματική επίθεση. Από την άλλη μεριά τα GET requests είναι πιο συνήθη, γιατί είναι πολύ πιο απλό να υλοποιηθούν μιας και χρειάζεται παραδείγματος χάρη ένα link μιας εικόνας ή γνώση πώς γίνεται αναζήτηση περιεχομένου σε ένα site (?s για WordPress μέσα στο url) και έτσι οποιοσδήποτε επισκέπτεται το site στέλνει και ένα GET request που προκαλεί συμφόρηση στον server.

- DNS flood: Η συγκεκριμένη επίθεση βασίζεται στέλνοντας πακέτα, συχνά από τροποποιημένες ip διευθύνσεις τα οποία κάνουν συνεχόμενα DNS ερωτήματα σε έναν DNS server τα οποία εξαντλούν τους πόρους του (CPU, memory). Επίσης μια κατηγορία επίθεσης DNS flood είναι η DNS NXDOMAIN flood attack στην οποία ο επιτιθέμενος γεμίζει τον DNS server με ερωτήματα για εγγραφές οι οποίες δεν υπάρχουν ή δεν είναι έγκυρες.

- UDP flood: Είναι ένα είδος επίθεσης στο οποίο ο επιτιθέμενος στέλνει έναν μεγάλο αριθμό πακέτων, συχνά από παραποιημένες ip διευθύνσεις σε τυχαίες πόρτες του μηχανήματος, με αποτέλεσμα το μηχάνημα να χρησιμοποιεί πόρους προκειμένου να εντοπίσει αν υπάρχουν διαθέσιμες υπηρεσίες στις συγκεκριμένες πόρτες. Το μηχάνημα επίσης πρέπει να στείλει Destination Unreachable πακέτα ως απάντηση για κάθε εισερχόμενο πακέτο. Καθώς ο αριθμός των εισερχόμενων πακέτων αυξάνεται, η καθυστέρηση αυξάνεται και στο τέλος το μηχάνημα δεν μπορεί να εξυπηρετήσει άλλους χρήστες.

- SYN flood: Σε αυτή την περίπτωση ο επιτιθέμενος ξεκινά TCP συνδέσεις, στέλνοντας SYN πακέτα στον στόχο αλλά δεν στέλνει πίσω ACK για να πραγματοποιηθεί η τριμερής χειραψία. Έτσι το μηχάνημα που δέχεται επίθεση περιμένει χρησιμοποιεί πόρους περιμένοντας την απάντηση ACK από τον επιτιθέμενο διατηρεί τις συνδέσεις «ημιανοιχτές» και σπαταλά άσκοπα πόρους.

- ICMP flood: Ο επιτιθέμενος στέλνει πολλά ICMP echo requests (εντολή ping) και ο στόχος σπαταλά πόρους στέλνοντας πίσω echo replies.

- Reflection attacks: Εδώ ο επιτιθέμενος παραποιεί την ip αποστολής έτσι ώστε να μοιάζει με τον στόχο τον οποίο θέλει να πλήξει. Η λογική εδώ πολλές φορές στηρίζεται και στην λειτουργία του amplification, δηλαδή χρησιμοποιεί πρωτόκολλο στο οποίο η ερώτηση δεν περιέχει πολλά δεδομένα, αλλά η απάντηση μπορεί να περιέχει πολλές φορές μεγαλύτερο πακέτο από αυτό της ερώτησης. Χαρακτηριστικά πρωτόκολλα στα οποία χρησιμοποιείται αυτή η επίθεση είναι το DNS και NTP. Στο DNS reflection, η τεχνική amplification μπορεί να είναι είτε χρησιμοποιώντας την προέκταση πρωτοκόλλου EDNS0 DNS στο DNS request που δίνει τη δυνατότητα για μεγάλα DNS μηνύματα, είτε την προέκταση κρυπτογραφίας του DNS (DNSSEC) για να αυξηθεί το μέγεθος του μηνύματος, είτε τα ερωτήματα να είναι του τύπου "ANY" το οποίο επιστρέφει όλες τις πληροφορίες για την γνωστή ζώνη. Στην περίπτωση του NTP παλιές εκδόσεις του NTP υποστηρίζουν υπηρεσία παρακολούθησης η οποία δίνει τη δυνατότητα στους διαχειριστές να στέλνουν ερωτήματα στον NTP server για μέτρηση της κίνησης. Με την εντολή monlist στέλνει ερώτημα και επιστρέφονται οι τελευταίοι 600 υπολογιστές που έχουν συνδεθεί στον NTP server. Έτσι αλλάζοντας ο επιτιθέμενος την source ip, σε ip του στόχου έχουμε reflection και amplification επίθεση μαζί.

4.2 Τεχνικές εντοπισμού DDoS.

Στα δίκτυα IP υπάρχει συγκεκριμένο εύρος ζώνης και συγκεκριμένη υπολογιστική ισχύς για την εκάστοτε δικτυακή κίνηση. Όταν κάποια χαρακτηριστικά του δικτύου υφίστανται στατιστική ανάλυση τότε για κάθε μία παράμετρο εμφανίζεται συγκεκριμένο μοτίβο. Όσο μεγαλύτερο είναι το χρονικό διάστημα στο οποίο πραγματοποιείται η στατιστική ανάλυση, τόσο πιο αξιόπιστο είναι και το μοτίβο.
Όμως αυτό είναι σωστό μόνο εάν η κίνηση είναι σταθερή ενώ αν υπάρχουν μεταβολές οι οποίες είναι ανεκτές ως κανονική κίνηση, μακροπρόθεσμα τα στατιστικά θα σταθεροποιηθούν και δεν μπορούν να θεωρηθούν ως τελείως αξιόπιστα.

Η συλλογή δεδομένων, το φιλτράρισμα και η επεξεργασία ανώμαλης δικτυακής συμπεριφοράς προσεγγίζεται με διάφορες τεχνικές. Η στατιστική ανάλυση και μηχανική μάθηση είναι δύο από τις κυριότερες μεθόδους.

- Στατιστική ανάλυση όπως η Εντροπία και οι Chi-Square τεχνικές [33],[34], έχουν προταθεί για εντοπισμό της αλλαγής στη δικτυακή κίνηση.

Η εντροπία παρουσιάζει τα πακέτα ως ανεξάρτητα σύμβολα πληροφορίας με ξεχωριστές πιθανότητες εμφάνισης. Είναι ένας πολύ κοινός τρόπος DDoS detection. Επιλέγοντας παράθυρο πακέτων συγκεκριμένου αριθμού, 10000 για παράδειγμα και μετακινώντας το προς τα μπροστά, ένα μοτίβο εμφανίζεται με πιθανότητες για κάθε είδος πακέτου. Μεγάλες αλλαγές στα χαρακτηριστικά των πακέτων σε κάθε παράθυρο πακέτου σημαίνουν ότι κάποια ανωμαλία συμβαίνει στο δίκτυο.

Εάν ένας συγκεκριμένος τύπος επίθεσης αναμένεται και ο τύπος της επικεφαλίδας του πακέτου είναι γνωστός τότε το μοντέλο Chi-Square είναι καλύτερο [33]. Για παράδειγμα, εάν αναμένεται επίθεση TCP SYN flood τότε κάνοντας δειγματοληψία στα πακέτα και μετρώντας τον αριθμό των TCP SYN επικεφαλίδων θα μας δώσει ένα μοτίβο του μέσου αριθμού τέτοιων επικεφαλίδων. Οποιαδήποτε απόκλιση από τα συνήθη όρια είναι ένδειξη επίθεσης.

$$\mathbf{X^2} = \sum_{\mathbf{i=1}}^{\mathbf{B}} \frac{(\mathbf{N_i} - \mathbf{n_i})^2}{\mathbf{n_i}}$$

Η προηγούμενη εξίσωση είναι γνωστή ως Chi-Square. N_i είναι ο αριθμός των πακέτων του δείγματος και n_i είναι ο αναμενόμενος αριθμός πακέτων σε φυσιολογική λειτουργία.

- Η μηχανική μάθηση είναι άλλη μια μέθοδος η οποία χρησιμοποιείται για την προστασία δικτύων ενάντια σε επιθέσεις. Αντί να υπάρχει ένα προκαθορισμένο φίλτρο, ένας αλγόριθμος έχει «εκπαιδευτεί» έτσι ώστε να ανανεώνει συνέχεια τα κριτήρια με τα οποία φιλτράρει τα γεγονότα του δικτύου. Ένα παράδειγμα τέτοιου συστήματος είναι ένα νευρωνικό δίκτυο. Τα νευρωνικά δίκτυα αποτελούνται από αρκετούς κόμβους που δουλεύουν παράλληλα για να επεξεργαστούν δεδομένα. Όταν εκπαιδευτούν ή τους δοθεί ένα μεγάλο μέγεθος πληροφορίας, η συλλεκτική γνώση των νευρώνων ή των κόμβων αναπτύσσουν ένα μοτίβο για να επεξεργαστούν παραπλήσια δεδομένα. Τα τρία κύρια επίπεδα ενός νευρωνικού δικτύου είναι η είσοδος η έξοδος και κρυμμένα επίπεδα στο ενδιάμεσο τα οποία επεξεργάζονται τα δεδομένα. Καθώς ο χρόνος περνά και περισσότερα δεδομένα επεξεργάζονται, οι κόμβοι μαθαίνουν περισσότερα και όλο και πιο καθαρά μοτίβα εμφανίζονται. Ένας αλγόριθμος είναι η καθοδηγούσα δύναμη πίσω από τις αποφάσεις οι οποίες λαμβάνονται από αυτά τα δίκτυα. Μερικοί γνωστοί αλγόριθμοι στην εντοπισμό και αντιμετώπιση ανωμαλιών σε δίκτυα είναι οι Multilayer Perception, Gaussian Classifier, K-means Clustering και τα μαρκοβιανά μοντέλα.

4.3 DDoS detection με χρήση εντροπίας.

Στη θεωρία πληροφορίας μεγαλύτερες τιμές εντροπίας αναμένονται όταν η μεταβλητή της πληροφορίας είναι περισσότερο τυχαία. Άμεσο επακόλουθο αυτού είναι ότι η εντροπία αναμένεται να είναι μικρή όταν η αβεβαιότητα της πληροφορίας είναι μικρή. Για να ποσοτικοποιήσει το μέγεθος της αβεβαιότητας ο Rényi εισήγαγε το μετρικό σύστημα της εντροπίας βαθμού α ως μαθηματική γενίκευση της εντροπίας του Shannon [35],[36],37]. Θεωρούμε διακριτή κατανομή πιθανότητας:

$$\mathbf{P} = \{\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3} \ldots . \mathbf{p_n}\} \quad \textbf{έτσι ώστε} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{p_i} = \mathbf{1} \ , \mathbf{p_i} \geq \mathbf{0}$$

Τότε η εντροπία του Rényi, βαθμού α ορίζεται ως εξής:

$$\boldsymbol{H_a(x)} = \frac{\mathbf{1}}{\mathbf{1-\alpha}} \mathbf{log_2} \left(\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{p_i^{\alpha}} \right)$$

Όπου $\boldsymbol{\alpha \geq 0, \alpha \neq 1, p_i \geq 0.}$

Όταν $\boldsymbol{\alpha = 0}$ ή οι τιμές του p_i είναι οι ίδιες, η μέγιστη εντροπία είναι γνωστή ως Harley entropy και είναι η εξής:

$$\boldsymbol{H_0(X) = \log_2 n}$$

Όταν $\boldsymbol{\alpha \to 1}$ η γενικευμένη εντροπία συγκλίνει στην εντροπία του Shannon [38]:

$$\boldsymbol{H_1(x) = -\sum_{i=1}^{n} p_i \log_2 p_i}$$

Όταν $\boldsymbol{\alpha = 2}$ τότε έχουμε την τετραγωνική εντροπία του Rényi:

$$\boldsymbol{H_2(x) = -\log_2 \sum_{i=1}^{n} p_i^2}$$

Όταν $\boldsymbol{\alpha \to \infty}$, $\boldsymbol{H_{\infty}(x)}$ έχουμε την μικρότερη τιμή εντροπίας της πληροφορίας.

Όταν $\alpha > 1$ η τιμή της γενικευμένης εντροπίας του Rényi, είναι μικρότερη [35] του Shannon και το γεγονός με τη μεγαλύτερη πιθανότητα έχει μεγαλύτερη επίδραση στην τελική εντροπία από ότι του Shannon. Επίσης όταν $0 < \alpha < 1$ η τιμή της γενικευμένης εντροπίας του Rényi είναι μεγαλύτερη από ότι του Shannon και το γεγονός με την μικρότερη πιθανότητα έχει μεγαλύτερη επίδραση στην τελική εντροπία. Για αυτό το

λόγο προκειμένου να έχουμε καλύτερα αποτελέσματα στον εντοπισμό DDoS επιθέσεων χρειάζεται να προσαρμόζουμε της τιμές τις παραμέτρου α, στο εκάστοτε δίκτυο.

Υπάρχουν δύο σημαντικοί παράμετροι στον εντοπισμό DDoS με την χρήση εντροπίας:

1) Το μέγεθος παραθύρου (**window size**).
2) Η τιμή κατωφλίου (**threshold**).

Το μέγεθος παραθύρου μπορεί να βασίζεται είτε σε ένα συγκεκριμένο χρονικό διάστημα, είτε σε συγκεκριμένο αριθμό πακέτων. Η εντροπία υπολογίζεται μέσα σε αυτό το παράθυρο για να εκτιμηθεί η αβεβαιότητα στα εισερχόμενα πακέτα. Εάν η υπολογισμένη εντροπία ξεπεράσει ένα συγκεκριμένο όριο ή είναι μικρότερη από αυτό τότε η επίθεση εντοπίστηκε.

4.4 Εφαρμογή DDoS detection στον POX

Γνωρίζουμε ότι κάθε φορά που ένα πακέτο έχει διαφορετική IP διεύθυνση, στέλνεται στον ελεγκτή. Ξέροντας ότι το πακέτο είναι καινούριο και ο προορισμός είναι στο ίδιο δίκτυο (flat topology), το μέγεθος της τυχαιότητας μπορεί να οριστεί υπολογίζοντας την εντροπία βασισμένη σε ένα συγκεκριμένο μέγεθος παραθύρου. Το μέγεθος παραθύρου είναι ο αριθμός των packet_in events των εισερχόμενων νέων πακέτων τα οποία και χρησιμοποιούνται για να υπολογιστεί η εντροπία. Όσο περισσότερες διαφορετικές IP διευθύνσεις υπάρχουν σε κάθε παράθυρο πακέτων, τόσο μεγαλύτερη είναι και η εντροπία και όσο λιγότερες διαφορετικές IP διευθύνσεις υπάρχουν τόσο μικρότερη είναι η εντροπία (με κατ' ελάχιστο 2 IP διευθύνσεις). Σε μια επίθεση DDoS αναμένουμε μεγάλη αύξηση των διαφορετικών IP διευθύνσεων αποστολέα μέσα στο παράθυρο πακέτων, άρα και αυξημένη τιμή εντροπίας [39].

Στην παρούσα εργασία θα ασχοληθούμε με τον προσδιορισμό της εντροπίας και τις διακυμάνσεις ανάλογα με το μέγεθος παραθύρου, καθώς επίσης και θα κάνουμε συγκριτικό ανάμεσα στην εντροπία του Shannon και στην εντροπία του Rényi με παραμέτρους $\alpha = 2$ και $\alpha = 0.5$ προκειμένου να βγάλουμε χρήσιμα συμπεράσματα. Το πείραμα θα πραγματοποιηθεί για μεγέθη παραθύρου 10, 20, 50 και 100 και η κατανομή των IP διευθύνσεων θα είναι οι δύο ακραίες περιπτώσεις, ένας host προς έναν άλλον και όλοι διαφορετικοί hosts προς έναν άλλον καθώς επίσης και τυχαίες περιπτώσεις (random) που ξεκινάνε από τη μικρότερη κατανομή διαφορετικών IP διευθύνσεων προς μεγαλύτερη κατανομή διαφορετικών IP. Επίσης θα τροποποιήσουμε κατάλληλα τον κώδικα έτσι ώστε να μας υπολογίζει την μέγιστη εντροπία από μία λίστα από εντροπίες της οποίας το μέγεθος καθορίζουμε εμείς. Τέλος θα τροποποιηθεί και πάλι ο κώδικας έτσι ώστε όταν ο υπολογισμός της εντροπίας ξεπεράσει ένα συγκεκριμένο όριο το οποίο υποδεικνύεται από τη μέγιστη εντροπία, θα στέλνεται αρχείο με τις πιθανές IP διευθύνσεις της DDoS επίθεσης, με τρόπο που εξηγείται στα επόμενα κεφάλαια

4.4.1 Υπολογισμός εντροπίας Shannon και Rényi για $\alpha = 2$ και 0.5

Τροποποιούμε τον κώδικα του l2_learning component του ελεγκτή POX. Στον constructor της κλάσης LearningSwitch προσθέτουμε τις εξής αρχικοποιήσεις:

- **self.ip_list=[]**
- **self.ip_dict={}**
- **self.counter=0**

Η μέθοδος _handle_PacketIn διαμορφώνεται ως εξής:

```
def _handle_PacketIn (self, event):

    list1=[]
    list2=[]
    list3=[]
    window_size=10

    packet = event.parsed
    if packet.type == ethernet.IP_TYPE:
      ipv4_packet=event.parsed.find('ipv4')
      src_ip=ipv4_packet.srcip
      log.debug(src_ip)
      self.counter+=1
      log.debug("the counter is %s",self.counter)
      self.ip_list.append(src_ip)
      log.debug(self.ip_list)
      if self.counter==window_size:
         self.counter=0
         for i in self.ip_list:
           if i not in self.ip_dict:
              self.ip_dict[i]=0
           self.ip_dict[i]+=1
         log.debug(self.ip_dict)
         for k in self.ip_dict.values():
           prob=(k/float(window_size))
           list1.append(-prob * math.log(prob,2))
           list2.append(pow(prob,2))
           list3.append(pow(prob,0.5))
         entropy1=sum(list1)
         entropy2=-math.log(sum(list2),2)
         entropy3=2*(math.log(sum(list3),2))
         log.debug("SHANNON ENTROPY IS %s",entropy1)
         log.debug("RENYI a=2 ENTROPY IS %s",entropy2)
         log.debug("RENYI a=0.5 ENTROPY IS %s",entropy3)
         self.ip_list=[]
         self.ip_dict={}
```

Επιπλέον βιβλιοθήκες που χρειάστηκαν:

```
from pox.lib.packet.ethernet import ethernet
from pox.lib.packet.ipv4 import ipv4
from pox.lib.addresses import IPAddr
import math
```

Οι τρεις πρώτες για να μπορούμε να επεξεργαστούμε το πακέτο και να λαμβάνουμε διευθύνσεις IP και η τελευταία προκειμένου να πραγματοποιήσουμε πιο σύνθετες μαθηματικές πράξεις για τον υπολογισμό της εντροπίας.

Η λειτουργία που επιτελεί ο κώδικας είναι η εξής:

- Για κάθε πακέτο που εισέρχεται στον ελεγκτή, εντοπίζεται η IP διεύθυνση αποστολέα.
- Αυξάνεται ο μετρητής κατά ένα και αποθηκεύεται η IP διεύθυνση σε μια λίστα.
- Όταν ο μετρητής φτάσει το επιθυμητό μέγεθος παραθύρου τότε για κάθε ip στη λίστα φτιάχνεται ένα λεξικό, προσπελάζοντας τη προηγούμενη λίστα και το οποίο περιέχει κάθε ip της λίστας και πόσες φορές εμφανίστηκε στο παράθυρο πακέτων.
- Για κάθε τιμή στο λεξικό που δείχνει πόσες φορές εμφανίστηκε η κάθε IP διεύθυνση, υπολογίζεται η πιθανότητα εμφάνισης της IP, η οποία είναι ο λόγος του πόσες φορές εμφανίστηκε η IP διά το μέγεθος του παραθύρου.
- Υπολογίζονται οι εντροπίες του Shannon και του Rényi για παραμέτρους $\alpha = 2$ και $\alpha = 0.5$.
- Επαναρχικοποιείται η λίστα που περιέχει τις διευθύνσεις IP καθώς και το λεξικό.

4.4.2 Πραγματοποίηση πειράματος

Λόγω του ότι πραγματοποιούμε πειράματα σε περιβάλλον mininet, δεν μπορούμε να έχουμε πραγματική κίνηση, αλλά θα μελετήσουμε διάφορες περιπτώσεις για τον υπολογισμό της εντροπίας με βάση διαφορετικά παράθυρα πακέτων. Για την κατανομή IP διευθύνσεων στο παράθυρο θα συμπεριλάβουμε ακραίες περιπτώσεις, δηλαδή όλες οι διευθύνσεις αποστολέα να ναι διαφορετικές και να έχουμε μόνο έναν αποστολέα, όπως και τυχαίες ενδιάμεσες περιπτώσεις. Οι τυχαίες περιπτώσεις ξεκινάνε από κατανομές λιγότερων IP διευθύνσεων προς κατανομές περισσότερων Για λόγους απλότητας, αντί να κάνουμε IP spoofing, θα δοκιμάσουμε να κάνουμε από έναν host ping προς άλλους και θα υποθέσουμε ότι είναι οι άλλοι που επιτίθενται προς αυτόν, χωρίς σφάλμα στα αποτελέσματά μας. Οι IP διευθύνσεις από τα ICMP replies θα είναι αυτά τα οποία θα μας δώσουν τις τιμές της εντροπίας και θα θεωρηθούν ως επίθεση.

Για βοήθεια θα χρησιμοποιήσουμε το εξής bash script:

```
#!/bin/bash
for i in 10.0.0.{1..x}
do
  sleep 1
  ping -c 1 "$i"
done
```

Όπου x ο αριθμός των IP διευθύνσεων που θέλουμε να κάνουμε ping.

Παράθυρο πακέτων μεγέθους 10

1)

IP	Πακέτα
10.0.0.1	5
10.0.0.2	5

Rényi's (a=2) entropy = 1
Shannon's entropy = 1
Rényi's (a=0.5) entropy = 1

2)

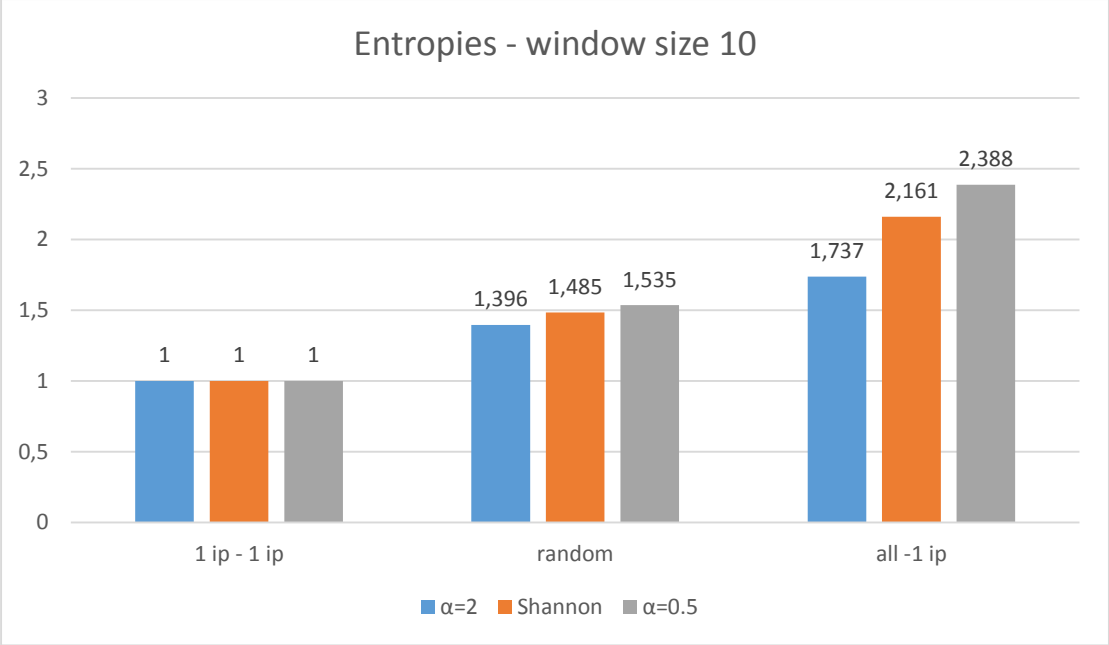
IP	Πακέτα
10.0.0.2	1
10.0.0.6	1
10.0.0.4	1
10.0.0.3	1
10.0.0.5	1
10.0.0.1	5

Rényi's (a=2) entropy = 1.73696559417
Shannon's entropy = 2.16096404744
Rényi's (a=0.5) entropy = 2.38848382726

3)

IP	Πακέτα
10.0.0.3	2
10.0.0.2	3
10.0.0.1	5

Rényi's (a=2) entropy = 1.39592867
Shannon's entropy = 1.48547529723
Rényi's (a=0.5) entropy = 1.5345348592



Παράθυρο πακέτων μεγέθους 20

1)

IP	Πακέτα
10.0.0.1	10
10.0.0.2	10

Rényi's (a=2) entropy = 1
Shannon's entropy = 1
Rényi's (a=0.5) entropy = 1

2)

IP	Πακέτα
10.0.0.9	1
10.0.0.11	1
10.0.0.2	1
10.0.0.6	1
10.0.0.4	1
10.0.0.8	1
10.0.0.10	1
10.0.0.7	1
10.0.0.3	1
10.0.0.5	1
10.0.0.1	10

Rényi's (a=2) entropy = 1.86249647625
Shannon's entropy = 2.66096404744
Rényi's (a=0.5) entropy = 3.11474641721

3)

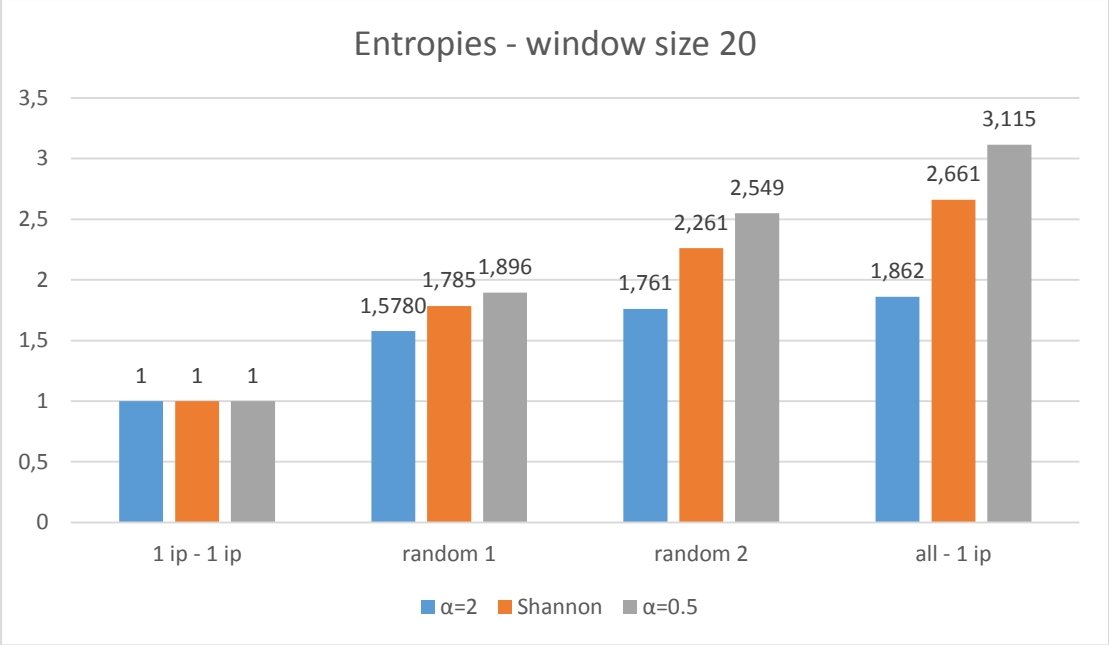
IP	Πακέτα
10.0.0.4	3
10.0.0.3	3
10.0.0.2	4
10.0.0.1	10

Rényi's (a=2) entropy = 1.57776699932
Shannon's entropy = 1.78547529723
Rényi's (a=0.5) entropy = 1.89558220175

4)

IP	Πακέτα
10.0.0.2	2
10.0.0.6	1
10.0.0.4	2
10.0.0.7	1
10.0.0.3	2
10.0.0.5	2
10.0.0.1	10

Rényi's (a=2) entropy = 1.76121314041
Shannon's entropy = 2.26096404744
Rényi's (a=0.5) entropy = 2.54909758862



Παράθυρο πακέτων μεγέθους 50

1)

IP	Πακέτα
10.0.0.1	25
10.0.0.2	25

Rényi's (a=2) entropy = 1
Shannon's entropy = 1
Rényi's (a=0.5) entropy = 1

2)

Παράθυρο με 25 διαφορετικές ip και 25 φορές την ip 10.0.0.1

Rényi's (a=2) entropy = 1.94341647163
Shannon's entropy = 3.32192809489
Rényi's (a=0.5) entropy = 4.16992500144

3)

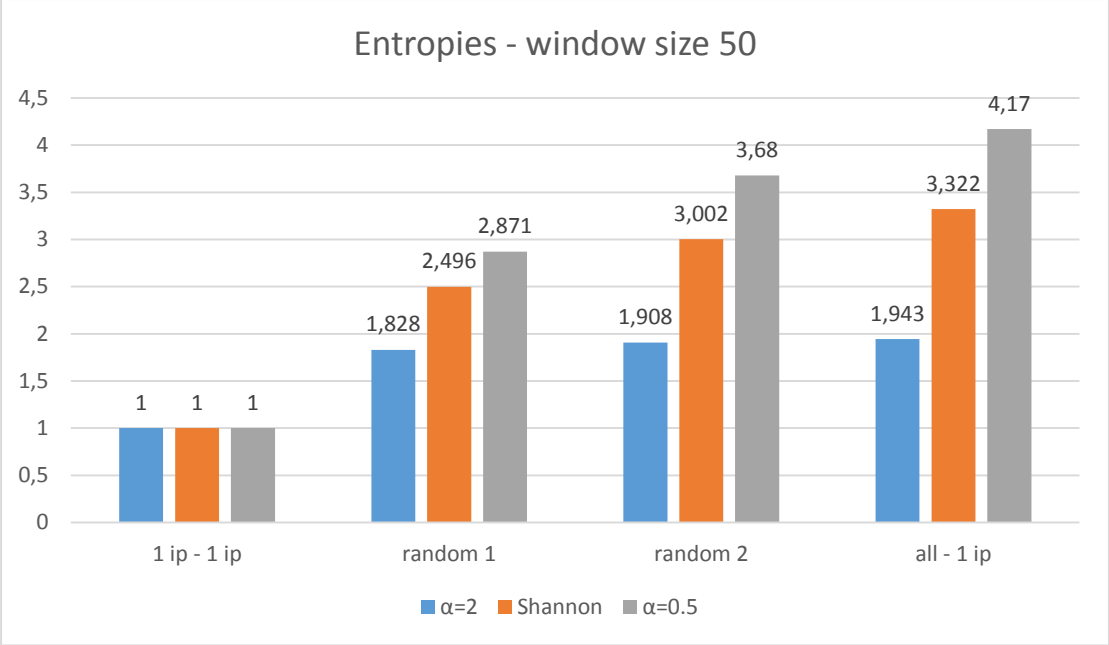
IP	Πακέτα
10.0.0.9	3
10.0.0.2	4
10.0.0.6	3
10.0.0.4	3
10.0.0.8	3
10.0.0.7	3
10.0.0.3	3
10.0.0.5	3
10.0.0.1	25

Rényi's (a=2) entropy = 1.82828076091
Shannon's entropy = 2.49624384458
Rényi's (a=0.5) entropy = 2.87082228129

4)

IP	Πακέτα
10.0.0.9	2
10.0.0.11	1
10.0.0.13	1
10.0.0.15	1
10.0.0.17	1
10.0.0.2	2
10.0.0.6	2
10.0.0.4	2
10.0.0.8	2
10.0.0.10	1
10.0.0.12	1
10.0.0.14	1
10.0.0.16	1
10.0.0.18	1
10.0.0.7	2
10.0.0.3	2
10.0.0.5	2
10.0.0.1	25

Rényi's (a=2) entropy = 1.90833401248
Shannon's entropy = 3.00192809489
Rényi's (a=0.5) entropy = 3.67983776015



Παράθυρο πακέτων μεγέθους 100

1)

IP	Πακέτα
10.0.0.1	50
10.0.0.2	50

Rényi's (a=2) entropy = 1
Shannon's entropy = 1
Rényi's (a=0.5) entropy = 1

2)

Παράθυρο με 50 διαφορετικές ip και 50 φορές την ip 10.0.0.1

Rényi's (a=2) entropy = 1.9714308478
Shannon's entropy = 3.82192809489
Rényi's (a=0.5) entropy = 5.0255191128

3)

IP	Πακέτα
10.0.0.9	3
10.0.0.11	3
10.0.0.13	3
10.0.0.15	3
10.0.0.17	3
10.0.0.2	4
10.0.0.10	3
10.0.0.12	3
10.0.0.4	3
10.0.0.16	3
10.0.0.18	3
10.0.0.7	3
10.0.0.3	3
10.0.0.5	3
10.0.0.20	4
10.0.0.21	3
10.0.0.1	50

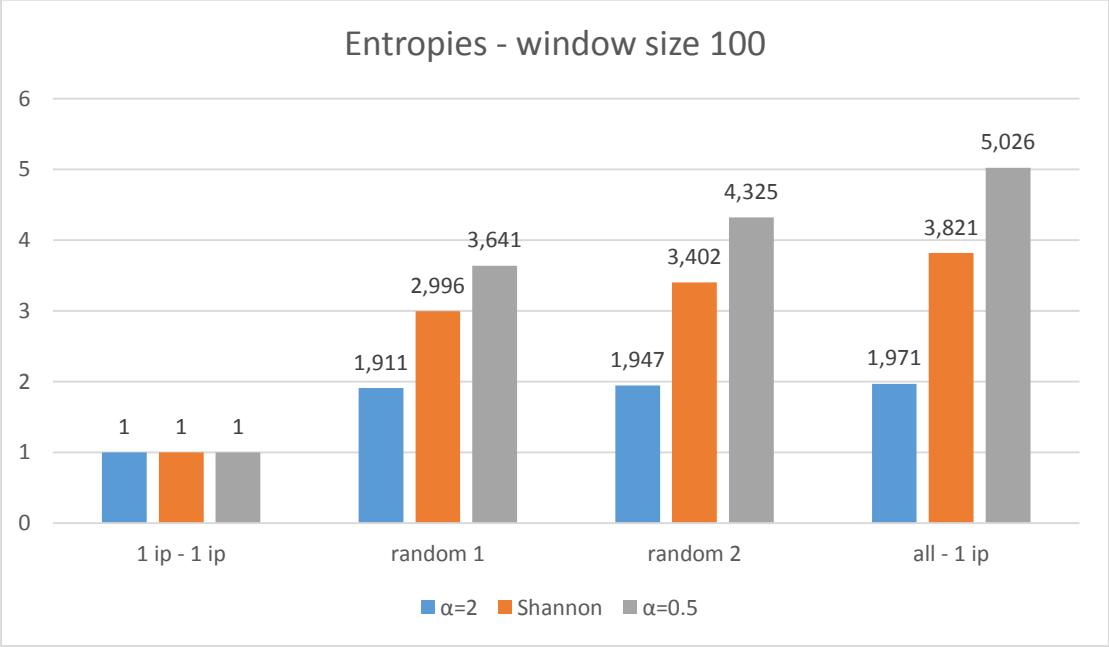
Rényi's (a=2) entropy = 1.91158699027
Shannon's entropy = 2.99624384458
Rényi's (a=0.5) entropy = 3.64095264143

4)

IP	Πακέτα
10.0.0.8	2
10.0.0.21	2
10.0.0.23	1
10.0.0.2	2
10.0.0.17	2
10.0.0.30	1
10.0.0.3	2
10.0.0.2	2
10.0.0.27	1
10.0.0.11	2
10.0.0.25	1
10.0.0.16	2
10.0.0.19	2
10.0.0.24	1
10.0.0.9	2
10.0.0.4	2
10.0.0.29	1
10.0.0.10	2
10.0.0.18	2
10.0.0.5	2
10.0.0.26	1

10.0.0.15	2
10.0.0.6	2
10.0.0.12	2
10.0.0.22	2
10.0.0.20	2
10.0.0.7	2
10.0.0.13	2
10.0.0.28	1
10.0.0.1	50

Rényi's (a=2) entropy = 1.9478623766
Shannon's entropy = 3.40192809489
Rényi's (a=0.5) entropy = 4.32503580317



Συμπεράσματα πειράματος:

1) Η εντροπία αυξάνεται σε ένα δίκτυο λόγω της αύξησης των διαφορετικών IP διευθύνσεων οι οποίες οφείλονται τόσο των πολλών υπολογιστών που συμμετέχουν στην επίθεση, όσο και στο πιθανό IP spoofing που εφαρμόζεται στα πακέτα σε περίπτωση DDoS.

2) Η τιμή της εντροπίας του Rényi για $\alpha > 1$ είναι μεγαλύτερη από την εντροπία του Shannon στις ίδιες δικτυακές συνθήκες. Επίσης η τιμή της εντροπίας του Rényi για $0 \leq \alpha < 1$ είναι μεγαλύτερη από την εντροπία του Shannon στις ίδιες δικτυακές συνθήκες.

3) Ανάμεσα στους 3 αλγορίθμους πιο λεπτομερής καταγραφή της δικτυακής με μεγαλύτερη διασπορά προκύπτει στην εντροπία του Rényi $\alpha = 0.5$.

4.4.3 Υπολογισμός μέγιστης εντροπίας κανονικής λειτουργίας.

Ο κώδικας παραμένει ο ίδιος με του 4.4.1 και από τις εντροπίες που υπολογίζονται μπορούμε να κρατήσουμε οποιαδήποτε θέλουμε. Στην συγκεκριμένη περίπτωση θα χρησιμοποιήσουμε του Shannon. Οι προσθήκες είναι οι εξής:

Στον constructor της κλάσης LearningSwitch προστίθενται οι εξής αρχικοποιήσεις για να δημιουργηθεί μια λίστα με εντροπίες καθώς επίσης και ένας μετρητής έτσι ώστε να δείχνει πόσες τιμές εντροπίας έχουν εισαχθεί στη λίστα αντίστοιχα:

- **self.entropylist=[]**
- **self.countentr=0**

Στη μέθοδο **_handle_PacketIn**:

Μετά τον υπολογισμό της εντροπίας και πριν τις αρχικοποιήσεις της λίστας των IP διευθύνσεων και του λεξικού προστίθεται ο εξής κώδικας:

```
        self.counterentr+=1
        log.debug("ENTROPY COUNTER IS %d",self.counterentr)
        self.entropylist.append(entropy1)
        if self.counterentr==counterentr_num:
         log.debug(self.entropylist)
         log.debug("NO ATTACK ENTROPY IS %s",max(self.entropylist))
         f=open("no_attack_entropy.txt","a")
         f.write("%s\n" % max(self.entropylist))
         f.close()
         self.entropylist=[]
         self.counterentr=0
```

Η λειτουργία που επιτελεί ο κώδικας είναι η εξής:

Κάθε φορά που υπολογίζεται η εντροπία ο μετρητής των τιμών της εντροπίας αυξάνεται κατά ένα και προστίθεται η τιμή της εντροπίας που έχει υπολογιστεί σε μια λίστα. Η εντροπία η οποία προστίθεται μπορεί να είναι οποιαδήποτε από τον γενικευμένο τύπο εντροπίας του Rényi, αλλά στη συγκεκριμένη περίπτωση επιλέγεται του Shannon σύμφωνα με τον κώδικα 4.4.1. Όταν ο μετρητής φτάσει την τιμή που έχει οριστεί (και ο οποίος ορίζει ουσιαστικά το μέγεθος της λίστας), υπολογίζεται η μέγιστη τιμή της εντροπίας από την εν λόγω λίστα. Καταγράφεται επίσης σε αρχείο έτσι ώστε να μπορεί κάποιος να έχει πρόσβαση στις τιμές. Στο τέλος του κώδικα γίνεται επαναρχικοποίηση τόσο του μετρητή όσο και της λίστας με τις εντροπίες.

Ο υπολογισμός της μέγιστης εντροπίας του δικτύου ουσιαστικά μας δίνει το threshold το οποίο χρειαζόμαστε, έτσι ώστε όταν δούμε τιμή μεγαλύτερη από αυτή να ξέρουμε ότι πρόκειται για επίθεση DDoS. Όλα αυτά υπό την υπόθεση ότι η κατανομή των IP διευθύνσεων σε περίπτωση DDoS είναι μεγαλύτερη από ότι σε κανονική λειτουργία.

4.4.4 Εντοπισμός DDoS επίθεσης.

Ο κώδικας παραμένει ο ίδιος με τον κώδικα του 4.4.1 έχοντας τις επόμενες προσθήκες.

Στον constructor της κλάσης **LearningSwitch** προστίθεται η εξής αρχικοποίηση:

- **self.ip_queue=deque(maxlen=n)** Όπου n οποιοσδήποτε αριθμός μεγαλύτερος από το μέγεθος παραθύρου πακέτων και τον οποίο καθορίζει ο χρήστης. Ο αριθμός αυτός δηλώνει πόσες είναι οι πιο πρόσφατες IP διευθύνσεις αποστολέα οι οποίες θα αποθηκεύονται σε μια δομή double – ended queue της Python [25].

Στη μέθοδο **_handle_PacketIn**:

- Αρχικοποιείται μια local μεταβλητή **threshold=num** όπου num είναι το threshold πάνω από τον οποίο θα γίνεται ο εντοπισμός DDoS και το οποίο προέκυψε από την εκτέλεση του κώδικα 4.4.3.

- Προστίθεται η εντολή **self.ip_queue.append(src_ip)** μετά την εντολή για προσθήκη των IP διευθύνσεων στη λίστα έτσι ώστε να προστίθενται οι IP διευθύνσεις και στη νέα δομή double – ended queue.

- Προστίθεται ο εξής κώδικας μετά τον υπολογισμό της εντροπίας και πριν τις επαναρχικοποιήσεις της λίστας των IP διευθύνσεων και του λεξικού:

```
        if entropy1 > threshold:
          print "***** DDoS DETECTED *****"
          ip_buffer=self.ip_queue
          for l in range (0,window_size):
            ip_buffer.pop()
          attack_ip=[ip for ip in self.ip_list if ip not in ip_buffer]
          f=open("DDoS_IPS.txt","w")
          for item in attack_ip:
            f.write("%s\n" % item)
          f.close()
          alert_DDoS()
```

Επιπλέον βιβλιοθήκες που χρειάστηκαν:

```
from collections import deque
import smtplib
from email.MIMEMultipart import MIMEMultipart
from email.MIMEText import MIMEText
from email.MIMEBase import MIMEBase
from email import encoders
```

Η πρώτη βιβλιοθήκη χρειάζεται για να χρησιμοποιηθεί η δομή double – ended queue. Οι επόμενες για να χρησιμοποιηθεί το πρωτόκολλο SMTP και να μπορεί να σταλεί ένα text αρχείο ως attachment στην python.

Η λειτουργία που επιτελεί ο κώδικας είναι η εξής:

Κάθε φορά που ο υπολογισμός της εντροπίας ξεπερνά τη τιμή η οποία έχει τεθεί ως threshold και το οποίο έχει υπολογιστεί από τον κώδικα υπολογισμού εντροπίας κανονικής λειτουργίας τότε συμβαίνουν τα εξής:

- Δημιουργείται ένας buffer, ως αντίγραφο της δομής double ended queue η οποία περιλαμβάνει τις τελευταίες n IP διευθύνσεις αποστολέα των πακέτων που έχουν φτάσει στον ελεγκτή, όπου n το μέγεθος της double ended queue που ορίζεται στις αρχικοποιήσεις.

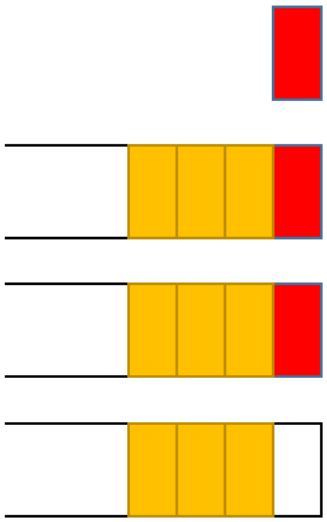
- Από τον buffer αφαιρούμε τα τελευταία m στοιχεία, όπου m είναι το μέγεθος του παραθύρου πακέτων.

- Για κάθε IP διεύθυνση από το παράθυρο πακέτων που έγινε εντοπισμός DDoS ελέγχεται αν στον προηγούμενο buffer υπάρχουν ίδιες IP διευθύνσεις. Αν υπάρχουν τότε δεν λαμβάνονται υπόψη, θεωρώντας ότι είναι IP διευθύνσεις φυσιολογικής λειτουργίας του δικτύου και όλες οι IP που υπάρχουν για πρώτη φορά στο παράθυρο πακέτων καταγράφονται σε ένα αρχείο ως πιθανές της επίθεσης DDoS. Υπάρχει και η περίπτωση να εντοπιστούν IP διευθύνσεις κανονικής λειτουργίας δικτύου οι οποίες να εμφανιστούν για πρώτη φορά στο παράθυρο πακέτων αλλά σε αυτό το γεγονός έγκειται και η δυσκολία εντοπισμού DDoS επιθέσεων.

- Οι καινούριες IP διευθύνσεις που αποθηκεύτηκαν στέλνονται με email σε μορφή text στο διαχειριστή του δικτύου και αυτό πραγματοποιείται με την κλήση της επόμενης συνάρτησης alert_DDoS(). Ο διαχειριστής καλείται να συμπληρώσει το username και password του email στο οποίο θα γίνεται η πρόσβαση, το email στο οποίο θα στέλνεται η ειδοποίηση, το path του αρχείου καταγραφής των IP διευθύνσεων, τον SMTP server και τη θύρα που ακούει ανάλογα με το είδος του email που χρησιμοποιεί.

```
def alert_DDoS():
	content = "DDoS DETECTED"
	fromaddr='email1'
	toaddr='email2'
	msg=MIMEMultipart()
	msg['From']=fromaddr
	msg['To']=toaddr
	msg['Subject']="DDoS DETECTED"
	body="possible malicious ips"
	msg.attach(MIMEText(body,'plain'))
	filename="DDoS_IPS.txt"
	attachment=open("/home/kostas/pox/DDoS_IPS.txt","rb")
	part=MIMEBase('application','octet-stream')
	part.set_payload((attachment).read())
	encoders.encode_base64(part)
	part.add_header('Content-Disposition',"attachment; filename= %s" % filename)
	msg.attach(part)
	mail = smtplib.SMTP('smtp.gmail.com',587)
	mail.starttls()
	mail.login(fromaddr,'password')
	text=msg.as_string()
	mail.sendmail(fromaddr,toaddr,text)
	mail.close()
```

Παρακάτω παρατίθεται και σχηματικό διάγραμμα του εντοπισμού των πιθανών IP διευθύνσεων DDoS. Με κόκκινο χρώμα είναι το παράθυρο πακέτων όταν η εντροπία ξεπέρασε το threshold του δικτύου. Στο συγκεκριμένο παράδειγμα ο buffer είναι τετραπλάσιος του παραθύρου πακέτων. Όταν αφαιρεθούν τα στοιχεία μέσω της μεθόδου pop από τον buffer, γίνεται σύγκριση αν υπάρχουν οι IP διευθύνσεις του παραθύρου πακέτων που συνέβη η DDoS επίθεση στον υπόλοιπο buffer, έτσι ώστε να γίνει μια εκτίμηση των καινούριων IP διευθύνσεων που εισήχθησαν στο δίκτυο σε σχέση με αυτές κανονικής λειτουργίας.



self.ip_list

self.ip_queue

ip_buffer

ip_buffer μετά από την εξαγωγή των τελευταίων m στοιχείων, όπου m το μέγεθος παραθύρου

5

Επίλογος

Σε αυτό το κεφάλαιο παρουσιάζεται μια σύνοψη της παρούσας διπλωματικής εργασίας και συμπεράσματα πάνω στην πειραματική διαδικασία. Επιπροσθέτως, παρατίθενται διάφορες προτάσεις για βελτίωση και περαιτέρω επέκταση στο μέλλον.

5.1 Σύνοψη και συμπεράσματα

Σκοπός της παρούσας διπλωματικής είναι να δείξει τις νέες δυνατότητες τις οποίες προσφέρουν τα ευφυή προγραμματιζόμενα δίκτυα στον τομέα της ασφάλειας και πιο συγκεκριμένα το OpenFlow. Υλοποιήθηκε ένα υβριδικό honeypot το οποίο βασίστηκε στη λογική του honeybrid και το οποίο ανακατευθύνει κίνηση αφού ένας αριθμός πακέτων φτάσουν στον ελεγκτή. Με την συγκεκριμένη υλοποίηση μπορούμε να ανακατευθύνουμε δικτυακή κίνηση μεταξύ ενός χαμηλής και ενός υψηλής αλληλεπίδρασης honeypot και με κατάλληλη αντιγραφή και τροποποίηση κώδικα για ακόμα περισσότερα honeypots. Δίνονται έτσι νέες δυνατότητες στην διαχείριση των honeynets με τη βοήθεια του OpenFlow. Στην δεύτερη περίπτωση είδαμε πώς μπορούμε να εντοπίσουμε DDoS επιθέσεις με την χρήση εντροπίας και παρατηρήσαμε τα διαφορετικά αποτελέσματα που δίνουν τρεις υποπεριπτώσεις της γενικευμένης εντροπίας του Rényi για τιμές της παραμέτρου α 1, 2 και 0.5 αντίστοιχα. Στην τελευταία περίπτωση οι τιμές της εντροπίας ήταν μεγαλύτερες και με μεγαλύτερη απόκλιση για τις ίδιες συνθήκες σε σχέση με τις άλλες και έτσι έχουμε πιο ακριβή περιγραφή της κατάστασης της δικτυακής κίνησης. Επιπλέον υλοποιήσαμε δύο κώδικες ακόμη βασισμένους στον κώδικα υπολογισμού της εντροπίας, έναν για τον υπολογισμό της μέγιστης εντροπίας ενός δικτύου και έναν για την αποστολή των πιθανών IP διευθύνσεων της DDoS επίθεσης στο email του διαχειριστή, σε περίπτωση παραβίασης του ορίου μέγιστης εντροπίας. Ταυτόχρονα πέραν των επιπρόσθετων λειτουργιών ασφαλείας οι OpenFlow μεταγωγείς λειτουργούν σαν κλασικοί μη OpenFlow μεταγωγείς. Βλέπουμε δηλαδή ότι πλέον οι δικτυακές συσκευές πέραν από την κανονική τους λειτουργία μπορούν να λειτουργούν σαν network security middle-boxes, ανοίγοντας νέες δυνατότητες και δίνοντας μεγαλύτερη ευελιξία στον διαχειριστή δικτύου.

5.2 Βελτιώσεις και Μελλοντικές επεκτάσεις.

Στην περίπτωση του υβριδικού honeypot μπορεί να τροποποιηθεί ο κώδικας έτσι ώστε να έχει συμπεριφορά παραπλήσια με τα υπόλοιπα modules του honeybrid. Επίσης μπορεί να δοκιμαστεί να εφαρμοστεί διαφορετική ανακατεύθυνση ανάλογα με το είδος της κίνησης, αν είναι UDP, ICMP ή TCP. Επίσης μπορούν να δοκιμαστούν διαφορετικές τιμές soft και hard timeouts των κανόνων για να ρυθμιστεί ο αριθμός των πακέτων που φτάνουν στον ελεγκτή και επομένως πόσο γρήγορα θα γίνεται η ανακατεύθυνση σε πραγματικές συνθήκες.

Στην περίπτωση του εντοπισμού DDoS, μπορεί να δοκιμαστεί η επανάληψη του υπολογισμού της εντροπίας αρκετές φορές και να υπολογιστεί ο μέσος όρος της, ιδίως σε πραγματικές συνθήκες που υπάρχει μεγάλη κίνηση πακέτων. Τέλος μπορούν να δοκιμαστούν διαφορετικές τιμές soft και hard timeouts των κανόνων για να ρυθμιστεί ο αριθμός πακέτων που φτάνει στον ελεγκτή και έτσι να επηρεαστεί το πόσο γρήγορα υπολογίζεται η εντροπία και αν είναι αποτελεσματικό να εντοπιστεί DDoS επίθεση στις εκάστοτε συνθήκες.

Βιβλιογραφία

[1] Unify Virtual and Physical Networking with Cisco Virtual Interface Card
http://www.cisco.com/c/en/us/products/collateral/interfaces-modules/ucs-m81kr-virtual-interface-card/white_paper_c11-618838.html
[2] Improved hardware limitations – OpenFlow, XINguard
http://www.xinguard.com/en/content.aspx?id=70/
[3] OpenFlow – The Random Security Guy
http://therandomsecurityguy.com/openflow
[4] OpenFlow Switch Specification Version 1.0.0
https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf
[5] OpenFlow Switch Specification Version 1.1.0
http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf
[6] OpenFlow Switch Specification Version 1.2.0
https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf
[7] OpenFlow Switch Specification Version 1.3.0
https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf
[8] OpenFlow Switch Specification Version 1.4.0
https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf
[9] OpenFlow Switch Specification Version 1.5.0
https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf
[10] Ryu SDN Framework
http://osrg.github.io/ryu/
[11] Project Floodlight
http://www.projectfloodlight.org/floodlight/
[12] OpenDaylight Platform
https://www.opendaylight.org/
[13] NOX
http://www.noxrepo.org/
[14] About POX
http://www.noxrepo.org/pox/about-pox/
[15] POX Wiki
https://openflow.stanford.edu/display/ONL/POX+Wiki
[16] POX Components
https://github.com/noxrepo/pox/tree/carp/pox/forwarding
[17] The Honeynet Project
https://www.honeynet.org/
[18] Honeypot (Computing)
https://en.wikipedia.org/wiki/Honeypot_%28computing%29
[19] Κουκουβίνος Ευστάθιος, “Current Trends in Honeypot Technology”, 2015
[20] Honeybrid: Hybrid Honeypot Framework
http://honeybrid.sourceforge.net/

[21] Honeybrid: combining low and high interaction honeypots
https://www.honeynet.org/node/430
[22] Robin Berhier, "Advanced Honeypot Architecture for Network Threats Quantification", University of Maryland, 2009
[23] Mininet, An instant virtual network on your laptop
http://mininet.org/
[24] Testing SDN behavior with Mininet – Linux magazine
http://www.linux-magazine.com
[25] Python
https://www.python.org/
[26] Software Defined Networking, Princeton University.Instrustors: Nick Feamster
https://www.coursera.org/course/sdn1
[27] How IT works – Domain Name System, Microsoft
https://technet.microsoft.com/en-us/magazine/2005.01.howitworksdns.aspx
[28] A Comparison of DNS Server Types: How To Choose the Right DNS Configuration
https://www.digitalocean.com/community/tutorials/a-comparison-of-dns-server-types-how-to-choose-the-right-dns-configuration
[29] UDPot
https://github.com/jekil/UDPot
[30] Minidns 0.3
https://pypi.python.org/pypi/minidns
[31] DDoS Attacks on the Rise and Stronger than Ever, Lifars
https://lifars.com/2015/08/ddos-attacks-are-stronger-more-in-number-than-ever/
[32] Distributed Denial of Service Attack (DDoS) - Imperva
https://www.incapsula.com/ddos/ddos-attacks/
[33] Seyed Mohammad Mousavi "Early Detection of DDoS Attacks in Software Defined Networks Controller" Carleton University, 2014
[34] Seyed Mohammad Mousavi and Marc St-Hilaire "Early Detection of DDoS Attacks against SDN Controllers", International Conference on Computing, Networking and Communications, Communication and Information Security Symposium, 2015
[35] Yang Xiang, Ke Li and Wanlei Zhou "Low-Rate DDoS Attacks Detection and Traceback by Using New Information Metrics", IEEE Transactions on information forensics and security, vol.6, no.2, 2011
[36] Bhuyan M.H, Bhattacharyya D.K, Kalita J.K "Information Metrics for low-rate and high-rate DDoS αttack Detection", Pattern Recognition Letters, Elsevier, 2014
[37] Ke Li, Wanlei Zhou, Shui Yu, Bo Dai "Effective DDoS Detection Using Generalized Entropy Metric" Algorithms and Architectures for Parallel Processing, 9th International Conference, ICA3PP 2009, Taipei, Taiwan, Publisher Springer Berlin Heidelberg
[38] Oleg Gudkov "Calculation Algorithm for Network Flow Parameters, Entropy in Anomaly Detection".Kaspersky Academy IT security for the Next Generation, International Round, Delft University of Technology, 2012
[39] Laura Feinstein, Dan Schnackenberg, Ravinda Balupari, Darrell Kindred "Statistical Approaches to DDoS Attack Detection and Response" Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX' 03) IEEE computer society, 2003

Παράρτημα

Κώδικας entropies.py για τον υπολογισμό της εντροπίας του Shannon και Rényi για $\alpha = 2$ και 0.5.
(Για τον υπολογισμό της μέγιστης εντροπίας και του εντοπισμού DDoS οι προσθήκες – αλλαγές καταγράφονται εντός της διπλωματικής στα ανάλογα κεφάλαια)

```
# Copyright 2011 James McCauley
#
# This file is part of POX.
#
# POX is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# POX is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with POX.  If not, see <http://www.gnu.org/licenses/>.


"""
An L2 learning switch.

It is derived from one written live for an SDN crash course.
It is somwhat similar to NOX's pyswitch in that it installs
exact-match rules for each flow.
"""

from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
import time
from pox.lib.packet.ethernet import ethernet
from pox.lib.packet.ipv4 import ipv4
from pox.lib.addresses import IPAddr
import math


log = core.getLogger()

# We don't want to flood immediately when a switch connects.
# Can be overriden on commandline.
_flood_delay = 0
```

```
class LearningSwitch (object):
  """
  The learning switch "brain" associated with a single OpenFlow
switch.

  When we see a packet, we'd like to output it on a port which will
  eventually lead to the destination.  To accomplish this, we build a
  table that maps addresses to ports.

  We populate the table by observing traffic.  When we see a packet
  from some source coming from some port, we know that source is out
  that port.

  When we want to forward traffic, we look up the desintation in our
  table.  If we don't know the port, we simply send the message out
  all ports except the one it came in on.  (In the presence of loops,
  this is bad!).

  In short, our algorithm looks like this:

  For each packet from the switch:
  1) Use source address and switch port to update address/port table
  2) Is transparent = False and either Ethertype is LLDP or the
packet's
     destination address is a Bridge Filtered address?
     Yes:
        2a) Drop packet -- don't forward link-local traffic (LLDP,
802.1x)
            DONE
  3) Is destination multicast?
     Yes:
        3a) Flood the packet
            DONE
  4) Port for destination address in our address/port table?
     No:
        4a) Flood the packet
            DONE
  5) Is output port the same as input port?
     Yes:
        5a) Drop packet and similar ones for a while
  6) Install flow table entry in the switch so that this
     flow goes out the appopriate port
     6a) Send the packet out appropriate port
  """


  def __init__ (self, connection, transparent):
    # Switch we'll be adding L2 learning switch capabilities to
    self.connection = connection
    self.transparent = transparent

    # Our table
    self.macToPort = {}

    # We want to hear PacketIn messages, so we listen
    # to the connection
    connection.addListeners(self)

    # We just use this to know when to log a helpful message
    self.hold_down_expired = _flood_delay == 0
```

```
    #log.debug("Initializing LearningSwitch, transparent=%s",

    # str(self.transparent))

    self.ip_list=[]
    self.ip_dict={}
    self.counter=0

  def _handle_PacketIn (self, event):

    list1=[]
    list2=[]
    list3=[]
    window_size=10

    packet = event.parsed
    if packet.type == ethernet.IP_TYPE:
      ipv4_packet=event.parsed.find('ipv4')
      src_ip=ipv4_packet.srcip
      log.debug(src_ip)
      self.counter+=1
      log.debug("the counter is %s",self.counter)
      self.ip_list.append(src_ip)
      log.debug(self.ip_list)
      if self.counter==window_size:
         self.counter=0
         for i in self.ip_list:
           if i not in self.ip_dict:
              self.ip_dict[i]=0
           self.ip_dict[i]+=1
         log.debug(self.ip_dict)
         for k in self.ip_dict.values():
           prob=(k/float(window_size))
           list1.append(-prob * math.log(prob,2))
           list2.append(pow(prob,2))
           list3.append(pow(prob,0.5))
         entropy1=sum(list1)
         entropy2=-math.log(sum(list2),2)
         entropy3=2*(math.log(sum(list3),2))
         log.debug("SHANNON ENTROPY IS %s",entropy1)
         log.debug("RENYI a=2 ENTROPY IS %s",entropy2)
         log.debug("RENYI a=0.5 ENTROPY IS %s",entropy3)
         self.ip_list=[]
         self.ip_dict={}


    def flood (message = None):
      """ Floods the packet """
      msg = of.ofp_packet_out()
      if time.time() - self.connection.connect_time >= _flood_delay:
        # Only flood if we've been connected for a little while...

        if self.hold_down_expired is False:
          # Oh yes it is!
          self.hold_down_expired = True
          log.info("%s: Flood hold-down expired -- flooding",
              dpid_to_str(event.dpid))
```

```
      if message is not None: log.debug(message)
      #log.debug("%i: flood %s -> %s", event.dpid,packet.src,packet.dst)
      # OFPP_FLOOD is optional; on some switches you may need to change
      # this to OFPP_ALL.
      msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
    else:
      pass
      #log.info("Holding down flood for %s", dpid_to_str(event.dpid))
    msg.data = event.ofp
    msg.in_port = event.port
    self.connection.send(msg)

  def drop (duration = None):
    """
    Drops this packet and optionally installs a flow to continue
    dropping similar ones for a while
    """
    if duration is not None:
      if not isinstance(duration, tuple):
        duration = (duration,duration)
      msg = of.ofp_flow_mod()
      msg.match = of.ofp_match.from_packet(packet)
      msg.idle_timeout = duration[0]
      msg.hard_timeout = duration[1]
      msg.buffer_id = event.ofp.buffer_id
      self.connection.send(msg)
    elif event.ofp.buffer_id is not None:
      msg = of.ofp_packet_out()
      msg.buffer_id = event.ofp.buffer_id
      msg.in_port = event.port
      self.connection.send(msg)

  self.macToPort[packet.src] = event.port # 1

  if not self.transparent: # 2
    if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
      drop() # 2a
      return

  if packet.dst.is_multicast:
    flood() # 3a
  else:
    if packet.dst not in self.macToPort: # 4
      flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
    else:
      port = self.macToPort[packet.dst]
      if port == event.port: # 5
        # 5a
        log.warning("Same port for packet from %s -> %s on %s.%s.  Drop."
            % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
        drop(10)
        return
      # 6
```

```
        log.debug("installing flow for %s.%i -> %s.%i" %
                  (packet.src, event.port, packet.dst, port))
         msg = of.ofp_flow_mod()
         msg.match = of.ofp_match.from_packet(packet, event.port)
         msg.idle_timeout = 1
         msg.hard_timeout = 1
         msg.actions.append(of.ofp_action_output(port = port))
         msg.data = event.ofp # 6a
         self.connection.send(msg)


class l2_learning (object):
  """
  Waits for OpenFlow switches to connect and makes them learning
switches.
  """
  def __init__ (self, transparent):
    core.openflow.addListeners(self)
    self.transparent = transparent

  def _handle_ConnectionUp (self, event):
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)


def launch (transparent=False, hold_down=_flood_delay):
  """
  Starts an L2 learning switch.
  """
  try:
    global _flood_delay
    _flood_delay = int(str(hold_down), 10)
    assert _flood_delay >= 0
  except:
    raise RuntimeError("Expected hold-down to be a number")

  core.registerNew(l2_learning, str_to_bool(transparent))
```

Κώδικας Hybrid Honeypot

```
# Copyright 2011 James McCauley
#
# This file is part of POX.
#
# POX is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# POX is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with POX.  If not, see <http://www.gnu.org/licenses/>.



"""
An L2 learning switch.

It is derived from one written live for an SDN crash course.
It is somwhat similar to NOX's pyswitch in that it installs
exact-match rules for each flow.
"""

from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
import time
from pox.lib.packet.ethernet import ethernet
from pox.lib.packet.ipv4 import ipv4
from pox.lib.addresses import IPAddr, EthAddr



log = core.getLogger()

# We don't want to flood immediately when a switch connects.
# Can be overriden on commandline.
_flood_delay = 0
```

```
class LearningSwitch (object):
  """
  The learning switch "brain" associated with a single OpenFlow
switch.

  When we see a packet, we'd like to output it on a port which will
  eventually lead to the destination.  To accomplish this, we build a
  table that maps addresses to ports.

  We populate the table by observing traffic.  When we see a packet
  from some source coming from some port, we know that source is out
  that port.

  When we want to forward traffic, we look up the desintation in our
  table.  If we don't know the port, we simply send the message out
  all ports except the one it came in on.  (In the presence of loops,
  this is bad!).

  In short, our algorithm looks like this:

  For each packet from the switch:
  1) Use source address and switch port to update address/port table
  2) Is transparent = False and either Ethertype is LLDP or the
packet's
     destination address is a Bridge Filtered address?
     Yes:
        2a) Drop packet -- don't forward link-local traffic (LLDP,
802.1x)
            DONE
  3) Is destination multicast?
     Yes:
        3a) Flood the packet
            DONE
  4) Port for destination address in our address/port table?
     No:
        4a) Flood the packet
            DONE
  5) Is output port the same as input port?
     Yes:
        5a) Drop packet and similar ones for a while
  6) Install flow table entry in the switch so that this
     flow goes out the appopriate port
     6a) Send the packet out appropriate port
  """
  def __init__ (self, connection, transparent):
    # Switch we'll be adding L2 learning switch capabilities to
    self.connection = connection
    self.transparent = transparent

    # Our table
    self.macToPort = {}

    # We want to hear PacketIn messages, so we listen
    # to the connection
    connection.addListeners(self)

    # We just use this to know when to log a helpful message
    self.hold_down_expired = _flood_delay == 0
    #log.debug("Initializing LearningSwitch, transparent=%s",
    #          str(self.transparent))
    self.counter=0
```

```
def _handle_PacketIn (self, event):

    counter_limit=6
    low_ip="10.0.0.2"
    high_ip="10.0.0.4"
    low_mac="62:d2:6d:2b:c8:8b"
    high_mac="2e:9e:99:13:53:8a"
    high_port=4
    idle_t=30
    hard_t=30

    packet = event.parsed
    myport = event.port
    if packet.type == ethernet.IP_TYPE:
      ipv4_packet=event.parsed.find('ipv4')
      src_ip=ipv4_packet.srcip
      dst_ip=ipv4_packet.dstip
      log.info("destination ip")
      if str(dst_ip) == low_ip:
       self.counter+=1
       print "the counter is %d" % self.counter
       if self.counter==counter_limit:
         self.counter=0
         msg = of.ofp_flow_mod()
         msg.priority = 65535
         msg.match.dl_type=0x800
         msg.idle_timeout = idle_t
         msg.hard_timeout = hard_t
         msg.match.nw_dst = IPAddr(low_ip)
         msg.actions.append(of.ofp_action_nw_addr.set_dst(IPAddr(high_ip)))
         msg.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr(high_mac)))
         msg.actions.append(of.ofp_action_output(port = high_port))
         self.connection.send(msg)
         msg = of.ofp_flow_mod()
         msg.priority = 65535
         msg.match.dl_type=0x800
         msg.idle_timeout = idle_t
         msg.hard_timeout = hard_t
         msg.match.nw_dst = IPAddr(src_ip)
         msg.actions.append(of.ofp_action_nw_addr.set_src(IPAddr(low_ip)))
         msg.actions.append(of.ofp_action_dl_addr.set_src(EthAddr(low_mac)))
         msg.actions.append(of.ofp_action_output(port = myport))
         self.connection.send(msg)
```

```
def flood (message = None):
  """ Floods the packet """
  msg = of.ofp_packet_out()
  if time.time() - self.connection.connect_time >= _flood_delay:
    # Only flood if we've been connected for a little while...

    if self.hold_down_expired is False:
      # Oh yes it is!
      self.hold_down_expired = True
      log.info("%s: Flood hold-down expired -- flooding",
          dpid_to_str(event.dpid))

    if message is not None: log.debug(message)
    #log.debug("%i: flood %s -> %s", event.dpid,packet.src,packet.dst)
    # OFPP_FLOOD is optional; on some switches you may need to change
    # this to OFPP_ALL.
    msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
  else:
    pass
    #log.info("Holding down flood for %s", dpid_to_str(event.dpid))
  msg.data = event.ofp
  msg.in_port = event.port
  self.connection.send(msg)

def drop (duration = None):
  """
  Drops this packet and optionally installs a flow to continue
  dropping similar ones for a while
  """
  if duration is not None:
    if not isinstance(duration, tuple):
      duration = (duration,duration)
    msg = of.ofp_flow_mod()
    msg.match = of.ofp_match.from_packet(packet)
    msg.idle_timeout = duration[0]
    msg.hard_timeout = duration[1]
    msg.buffer_id = event.ofp.buffer_id
    self.connection.send(msg)
  elif event.ofp.buffer_id is not None:
    msg = of.ofp_packet_out()
    msg.buffer_id = event.ofp.buffer_id
    msg.in_port = event.port
    self.connection.send(msg)

self.macToPort[packet.src] = event.port # 1

if not self.transparent: # 2
  if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
    drop() # 2a
    return

if packet.dst.is_multicast:
  flood() # 3a
```

```
    else:
      if packet.dst not in self.macToPort: # 4
        flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
      else:
        port = self.macToPort[packet.dst]
        if port == event.port: # 5
          # 5a
          log.warning("Same port for packet from %s -> %s on %s.%s.  Drop."
              % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
          drop(10)
          return
        # 6
        log.debug("installing flow for %s.%i -> %s.%i" %
                  (packet.src, event.port, packet.dst, port))
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet, event.port)
        msg.idle_timeout = 10
        msg.hard_timeout = 10
        msg.priority = 25
        msg.actions.append(of.ofp_action_output(port = port))
        msg.data = event.ofp # 6a
        self.connection.send(msg)


class l2_learning (object):
  """
  Waits for OpenFlow switches to connect and makes them learning switches.
  """
  def __init__ (self, transparent):
    core.openflow.addListeners(self)
    self.transparent = transparent

  def _handle_ConnectionUp (self, event):
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)


def launch (transparent=False, hold_down=_flood_delay):
  """
  Starts an L2 learning switch.
  """
  try:
    global _flood_delay
    _flood_delay = int(str(hold_down), 10)
    assert _flood_delay >= 0
  except:
    raise RuntimeError("Expected hold-down to be a number")

  core.registerNew(l2_learning, str_to_bool(transparent))
```