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

Σύστημα Υλοποίησης Ασφάλειας σε Περιβάλλον OpenFlow με χρήση Pyretic Policy Function

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χρήστος Λ. Κωνσταντινίδης

Επιβλέπων : Βασίλειος Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2015

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

Σύστημα Υλοποίησης Ασφάλειας σε Περιβάλλον OpenFlow με χρήση Pyretic Policy Function

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χρήστος Λ. Κωνσταντινίδης

Επιβλέπων : Βασίλειος Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 23η Οκτωβρίου 2015.

.....	.....	.....
Β. Μάγκλαρης Καθηγητής Ε.Μ.Π.	Δ. Καλογεράς Ερευνητής ΕΠΙΣΕΥ	Ε. Συκάς Καθηγητής Ε.Μ.Π

Αθήνα, Οκτώβριος 2015

.....
Χρήστος Λ. Κωνσταντινίδης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Βρισκόμαστε σε μια μεταβατική περίοδο όπου οι εταιρείες, οι πάροχοι υπηρεσιών διαδικτύου και το διαδίκτυο γενικότερα προσαρμόζεται στις απαιτήσεις της έκδοσης 6 του πρωτοκόλλου του διαδικτύου, του IPv6.

Στο κομμάτι της ασφάλειας, υπάρχουν σοβαρά προβλήματα στο IPv6 καθώς δεν υπάρχει ακόμα αρκετή εμπειρία από τους προγραμματιστές δικτύου, παρότι οι επιθέσεις στα δίκτυα IPv6 μοιάζουν με αυτές των δικτύων IPv4. Κύριες διαφορές των δυο εκδόσεων είναι η αντικατάσταση του πρωτοκόλλου Address Resolution Protocol (ARP) με το πρωτόκολλο Neighbor Discovery Protocol (NDP) και το μεγάλο εύρος διαθέσιμων IP διευθύνσεων στα δίκτυα IPv6. Συγκεκριμένα, το μικρότερο δυνατό υποδίκτυο IPv6 θα έχει 18,446,744,073,709,551,616 (18.4 Quintillion) διαθέσιμες διευθύνσεις.

Παρ' όλα αυτά, η ασφάλεια ενός δικτύου δεν μπορεί να στηριχθεί στην δυσκολία εύρεσης των ενεργών διευθύνσεων λόγω του μεγάλου εύρους διαθέσιμων διευθύνσεων IP. Αυτή η διπλωματική εργασία ασχολείται με την δημιουργία ενός συστήματος ασφάλειας για την αντιμετώπιση των επιθέσεων στα δίκτυα IPv6, εστιάζοντας κυρίως στον εντοπισμό και την αντιμετώπιση ενός επιτιθέμενου που πραγματοποιεί σκανάρισμα ενός δικτύου για να μάθει τις ενεργές του διευθύνσεις. Η δημιουργία ενός τέτοιου συστήματος θα ήταν αδύνατη χωρίς τα πλεονεκτήματα των Ευφυών Προγραμματιζόμενων Δικτύων (Software Defined Networking - SDN) και εξαιρετικά δύσκολη χωρίς την χρήση της υψηλού επιπέδου SDN γλώσσας Pyretic.

Στον παραδοσιακό προγραμματισμό με OpenFlow είναι σχεδόν αδύνατος ο συνδυασμός ανεξάρτητων πολιτικών δικτύου χωρίς την ανησυχία ότι θα παρεμβάλει η μια πολιτική στην άλλη. Με το Pyretic μπορούμε να συνδυάσουμε πολλαπλές πολιτικές χρησιμοποιώντας χειριστές σύνθεσης πολιτικών. Έτσι, φτιάξαμε μια Pyretic εφαρμογή η οποία έχει ένα σύστημα για να γνωρίζει μόνιμα τις ενεργές διευθύνσεις του δικτύου και συνδυάζει τρεις πολιτικές, μια πολιτική παρακολούθησης της κίνησης του δικτύου, μια πολιτική που αντιλαμβάνεται πότε κάποιος προσπαθεί να σκανάρει το δίκτυο IPv6 και μια πολιτική που αναλαμβάνει να του απαγορεύει να στείλει ξανά.

Λέξεις Κλειδιά: Ευφυή Προγραμματιζόμενα Δίκτυα, σκανάρισμα δικτύου, σκανάρισμα διευθύνσεων IP, OpenFlow, POX, Pyretic, IPv6, δυναμικές πολιτικές δικτύων, Πρωτόκολλο Ανακάλυψης Γείτονα, ICMPv6

Abstract

We are in a transitional period where companies, internet service providers and the internet in general, are adjusting to the requirements of the Internet Protocol version 6, IPv6.

There are major security issues in IPv6 as network developers do not have enough experience yet, although the attacks on the IPv6 networks are similar to those on the IPv4 networks. Main differences of these two versions are the replacement of the Address Resolution Protocol (ARP) with the Neighbor Discovery Protocol (NDP), and the large range of the available IP addresses on the IPv6 networks. Specifically, the smallest possible IPv6 subnets will have 18,446,744,073,709,551,616 (18.4 Quintillion) available addresses.

However, the security of a network cannot rely on the assumption of how difficult it may be to find active addresses due to the large range of available IP addresses. This thesis deals with creating a security system for addressing the attacks on the IPv6 networks, focusing mainly on tracking and confronting the attacker who is scanning a network in order to find out its active addresses. Creating such a system would be impossible without the advantages of the Software-Defined Networks (SDN) and highly difficult without using the high-level SDN language, Pyretic.

In conventional OpenFlow programming, the combination of independent network policies without intervening with each other, is almost impossible. With Pyretic, we can combine multiple policies using policy composition operators. Thus, we have created a Pyretic application that combines a system for the continuous acknowledge of a network's active addresses with three network policies, one monitoring policy of the network traffic, one policy that detects when someone is trying to scan the IPv6 network and one policy that prohibits them from sending again.

Keywords: Software Defined Networks, scanning network, scanning IP addresses, OpenFlow, POX, Pyretic, IPv6, dynamic network policies, Neighbor Discovery Protocol, NDP, ICMPv6

Ευχαριστίες

Η διπλωματική αυτή εργασία αποτελεί το τελευταίο στάδιο των προπτυχιακών μου σπουδών στο Εθνικό Μετσόβιο Πολυτεχνείο. Αρχικά, θα ήθελα να ευχαριστήσω τον Ερευνητή ΕΠΙΣΕΥ, Δημήτρη Καλογερά, για την καθοδήγησή του κατά την διάρκεια αυτής. Ακόμη, θα ήθελα να ευχαριστήσω επιβλέποντα καθηγητή μου, κ. Βασίλειο Μάγκλαρη, για την δυνατότητα που μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα καθώς και για την έμπνευση που προσδίδει στους φοιτητές του να ασχοληθούν ενεργά με τα δίκτυα υπολογιστών.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για την αμέριστη υποστήριξη που μου έχουν προσφέρει όλα αυτά τα χρόνια σε όλα τα στάδια της ζωής μου, αλλά και τους φίλους για την κατανόηση που έχουν δείξει καθ' όλη τη διάρκεια των σπουδών μου.

Χρήστος Λ. Κωνσταντινίδης
Οκτώβριος 2015

Πίνακας περιεχομένων

Περιεχόμενα Εικόνων

1 ΕΙΣΑΓΩΓΗ

1.1 Ερευνητικό Πρόβλημα – Προσέγγιση

Η ασφάλεια των δικτύων είναι μια μεγάλη πρόκληση τόσο για τους οργανισμούς που προστατεύουν την διαφάνεια του διαδικτύου όσο και για τους μηχανικούς δικτύου, οι οποίοι προσπαθούν να προστατέψουν τα τοπικά τους δίκτυα από ποικίλες επιθέσεις. Η ανάγκη για ασφαλή δίκτυα γίνεται ακόμα μεγαλύτερη όσο ενσωματώνεται το Internet of Things στην καθημερινή ζωή των ανθρώπων, όπου πλέον ο κίνδυνος δεν σταματάει σε έναν υπολογιστή αλλά σε οτιδήποτε είναι συνδεδεμένο στο διαδίκτυο.

Σε αντιστοιχία με τους κινδύνους, οι τεχνολογίες των δικτύων εξελίσσονται ραγδαία, με σταθμό ορόσημο την δημιουργία των Ευφυών Προγραμματιζόμενων Δικτύων (Software-Defined Networks - SDN). Τα SDN έδωσαν την δυνατότητα στους μηχανικούς δικτύων να καινοτομήσουν σε κάθε παράμετρο των δικτύων όπως η ταχύτητα, η εξισορρόπηση φορτίου και η ασφάλεια. Ειδικότερα στην ασφάλεια, το SDN δίνει απεριόριστες επιλογές στους προγραμματιστές καθώς μπορούν πλέον να δημιουργούν εφαρμογές οι οποίες έχουν επίγνωση της κατάστασης ολόκληρου του δικτύου. Έτσι, γίνεται πιο δύσκολο για κάθε επιτιθέμενο να βρει κενά ασφαλείας καθώς υπάρχει εποπτεία ολόκληρου του δικτύου.

Για το σπάσιμο της ασφάλειας των δικτύων, ο επιτιθέμενος χρειάζεται να συλλέξει δεδομένα από το δίκτυο και ένα από τα πιο σημαντικά είναι οι ενεργές IP διευθύνσεις του. Για τον λόγο αυτό πραγματοποιεί ένα σκανάρισμα δικτύου για να καταλάβει ποιες IP διευθύνσεις μπορούν να προσπελαστούν μέσα από το διαδίκτυο και πιθανώς την αρχιτεκτονική του δικτύου. Στο πρωτόκολλο IPv6, μία λύση στα προβλήματα ασφαλείας δίνει το μεγάλο εύρος διαθέσιμων IP διευθύνσεων. Συγκεκριμένα, ένα απλό οικιακό δίκτυο θα έχει 18,446,744,073,709,551,616 (18,4 Quintillion) διαθέσιμες διευθύνσεις IP. Επομένως, ο επιτιθέμενος προκειμένου να σκανάρει όλο το δίκτυο και να βρει τις ενεργές διευθύνσεις IP θα χρειαστεί χρόνια.

Παρόλα αυτά η ασφάλεια ενός δικτύου δεν μπορεί να στηρίζεται στην δυσκολία εύρεσης των ενεργών διευθύνσεων λόγω του μεγάλου εύρους διαθέσιμων IP διευθύνσεων. Η εργασία αυτή ασχολείται με την δημιουργία ενός συστήματος για τον εντοπισμό του επιτιθέμενου που κάνει σκανάρισμα ενός δικτύου IPv6 και την δημιουργία δυναμικών πολιτικών δικτύου (network policies) έτσι ώστε να μην επιτραπεί ξανά η πρόσβαση του στο δίκτυο. Η δημιουργία ενός τέτοιου συστήματος θα ήταν αδύνατη χωρίς τα πλεονεκτήματα του SDN και εξαιρετικά δύσκολη χωρίς την χρήση της υψηλού επιπέδου SDN γλώσσας Pyretic. Στον παραδοσιακό προγραμματισμό με OpenFlow είναι αδύνατος ο συνδυασμός ανεξάρτητων πολιτικών δικτύου χωρίς την ανησυχία ότι θα παρεμβάλει η μία πολιτική στην

άλλη. Με το Pyretic, μπορούμε να συνδυάσουμε πολλαπλές πολιτικές χρησιμοποιώντας χειριστές σύνθεσης πολιτικών. Έτσι, μπορούμε να συνδυάσουμε μια πολιτική η οποία θα παρακολουθεί το δίκτυο, με μια δεύτερη πολιτική η οποία θα καταλαβαίνει αν κάποιος προσπαθεί να σκανάρει το δίκτυο και μια τρίτη πολιτική η οποία θα αναλαμβάνει να μην τον αφήνει να εισέλθει ξανά.

1.2 Συνεισφορά Εργασίας

Η εργασία αυτή έχει ως στόχο τον εντοπισμό και την αντιμετώπιση του σκαναρίσματος ενός IPv6 δικτύου (IPv6 address scanning). Στα πλαίσια της εργασίας, λύνονται προβλήματα τα οποία δίνουν την ευχέρεια στην κοινότητα του SDN να ασχοληθεί πιο ενεργά με τα θέματα που υπάρχουν στα δίκτυα IPv6. Συγκεκριμένα:

- Η επέκταση του έργου ανοιχτού λογισμικού (open-source project) Pyretic, για να μπορεί πλέον να διαχειρίζεται δίκτυα IPv6, δίνει την δυνατότητα στους SDN προγραμματιστές να αντιμετωπίσουν τα προβλήματα των IPv6 δικτύων με την χρήση πολιτικών.
- Η τροποποίηση του Pyretic για την ευκολότερη δημιουργία πολιτικών ασφάλειας στο IPv6
- Ο εντοπισμός του επιτιθέμενου που προκαλεί σκανάρισμα στο IPv6 δίκτυο.
- Η αντιμετώπιση του σκαναρίσματος δικτύου IPv6 με την χρήση δυναμικών πολιτικών.

1.3 Δομή Εργασίας

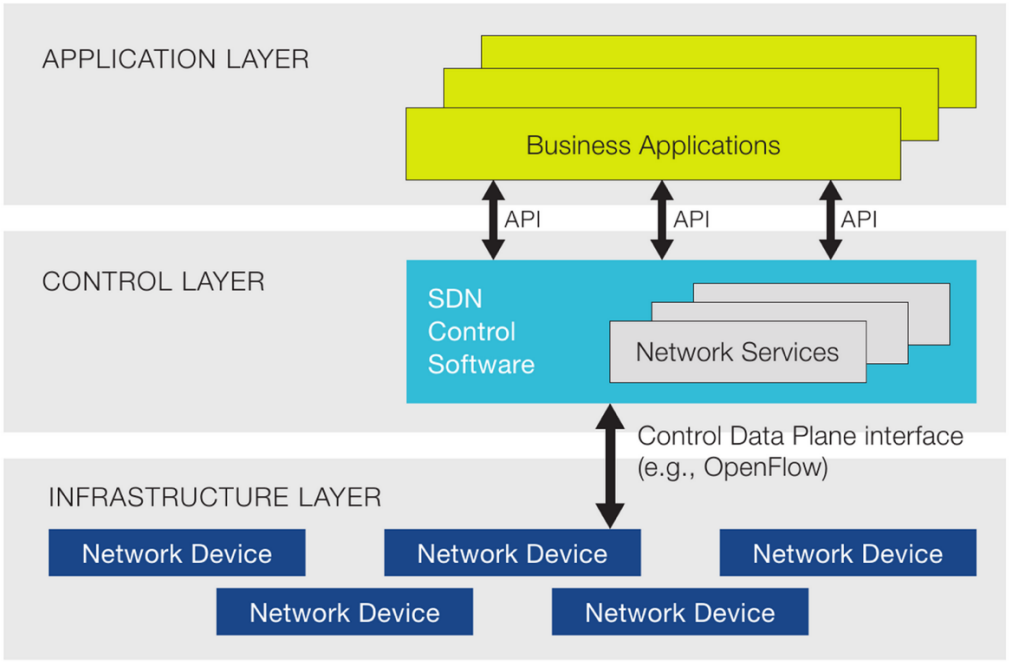
Η διπλωματική εργασία αποτελείται από επτά κεφάλαια. Στο πρώτο κεφάλαιο, αναπτύχθηκαν οι ανάγκες που οδήγησαν στην δημιουργία της διπλωματικής εργασίας και η οργάνωση του κειμένου. Στο δεύτερο κεφάλαιο παρουσιάζεται η αρχιτεκτονική δικτύου Software-Defined-Networking (SDN). Στο τρίτο κεφάλαιο παρουσιάζεται η SDN υψηλού επιπέδου γλώσσα, Pyretic μαζί με το Runtime σύστημά της. Στο τέταρτο κεφάλαιο αναλύονται κυριότερες επιθέσεις σε δίκτυα IPv6 ενώ στο πέμπτο κεφάλαιο παρουσιάζεται αναλυτικά ο τρόπος δημιουργίας του συστήματός μας. Στο έκτο παρουσιάζονται τα αποτελέσματα της εργασίας μας και στο έβδομο κεφάλαιο οι επεκτάσεις της. Τέλος, παρατίθενται η βιβλιογραφία και το παράρτημα με τον κώδικα.

2 ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

2.1 Ευφυή Προγραμματιζόμενα Δίκτυα (Software-Defined Networks - SDN)

Το SDN παρουσιάζεται ως μια δυναμική, διαχειρίσιμη, οικονομικά αποδοτική και εύκολα προσαρμόσιμη αρχιτεκτονική, κατάλληλη για τις υψηλών απαιτήσεων δυναμικές εφαρμογές της σημερινής εποχής.

Η διαφορά της σε σχέση με τα υπάρχοντα δίκτυα είναι η αποσύνδεση των λειτουργιών ελέγχου του δικτύου με τις λειτουργίες προώθησης. Πλέον, δεν είναι αναγκαίος ο έλεγχος και έπειτα η προώθηση των πακέτων του δικτύου από τον κάθε μεταγωγέα ή δρομολογητή με τα δικά του πρωτόκολλα, αλλά αρκείται στο να τα προωθεί. Το κομμάτι του ελέγχου μπορεί να γίνεται απομακρυσμένα δίνοντας στους υπεύθυνους του δικτύου την δυνατότητα να δημιουργήσουν ισχυρές και καινοτόμες εφαρμογές. [1] Η αρχιτεκτονική του SDN διαχωρίζει το δίκτυο σε 3 επίπεδα: εφαρμογών, ελέγχου και δεδομένων ή υποδομών, όπως φαίνεται παρακάτω:



Εικόνα 2.1: Αρχιτεκτονική SDN τριών επιπέδων
[https://www.sdxcentral.com/resources/sdn/inside-sdn-architecture]

Το επίπεδο εφαρμογών περιέχει εφαρμογές SDN οι οποίες επικοινωνούν μέσω διεπαφής προγραμματισμού εφαρμογών (Application Programming Interface - API) με τον ελεγκτή (controller) και του ανακοινώνουν τις επιθυμητή συμπεριφορά του δικτύου.

Το επίπεδο ελέγχου, είναι ένα λειτουργικό σύστημα δικτύου, το οποίο λαμβάνει οδηγίες ή απαιτήσεις από το στρώμα εφαρμογών SDN και τις αναμεταδίδει στο επίπεδο δεδομένων. Οι αποφάσεις λαμβάνονται με μια γενική άποψη ολοκλήρου του δικτυού και όχι με την περιορισμένη ορατότητα των γειτονικών δικτυακών συσκευών, όπως κάνουν οι δρομολογητές σήμερα.

Στο επίπεδο δεδομένων έχουμε τις συσκευές υλικού στις οποίες πραγματοποιείται η προώθηση των πακέτων. [2]

Τα βασικά στοιχεία ενός δικτύου αρχιτεκτονικής SDN είναι οι ελεγκτές στο επίπεδο ελέγχου, οι μεταγωγείς στο επίπεδο δεδομένων και το πρωτόκολλο επικοινωνίας τους με πιο διαδεδομένο το OpenFlow.

2.1.1 Το πρωτόκολλο OpenFlow

Το προγραμματιζόμενο πρωτόκολλο δικτύου OpenFlow είναι η πρώτη πρότυπη διεπαφή επικοινωνιών μεταξύ του επιπέδου ελέγχου και προώθησης σε μια αρχιτεκτονική SDN.

Επιτρέπει την άμεση πρόσβαση και διαχείριση της κίνησης των δεδομένων του επιπέδου προώθησης των συσκευών δικτύου (δρομολογητών, μεταγωγέων, επαναληπτών), εικονικά́ και φυσικά́. Εφαρμόζεται στις δυο πλευρές της διεπαφής μεταξύ́ των συσκευών υποδομής δικτύου και του λογισμικού́ ελέγχου SDN.
Χρησιμοποιεί́ τους πίνακες ροής (flow tables) για την αναγνώριση της κίνησης δικτύου που βασίζεται σε «κανόνες» που έχουν προγραμματιστεί́ δυναμικά́ ή στατικά́ από το λογισμικό́ ελέγχου SDN. Επίσης, επιτρέπει την κατεύθυνση της κίνησης ορίζοντας παραμέτρους όπως μοτίβα χρήσης και εφαρμογές. Εφόσον στο πρωτόκολλο OpenFlow το δίκτυο προγραμματίζεται με βάση τις ροές, μια αρχιτεκτονική́ SDN – OpenFlow παρέχει εξαιρετικά́ λεπτομερή́ έλεγχο, επιτρέποντας στο δίκτυο να απαντήσει σε αλλαγές σε πραγματικό χρόνο στα επίπεδα της εφαρμογής, χρήστη και συνοδού. [3]

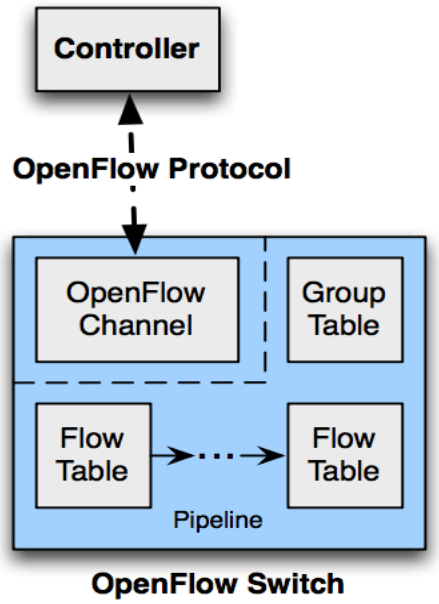
2.1.2 Ελεγκτής OpenFlow

Σε ένα δίκτυο, μπορεί να υπάρχουν ένας ή περισσότεροι ελεγκτές. Ο ελεγκτής εκτελεί τις διεργασίες ελέγχου του δικτύου OpenFlow και παρέχει διεπαφή για την διαχείριση και κατεύθυνση των πινάκων ροών των συσκευών που ελέγχει. Ακόμα, στέλνει στις συσκευές προώθησης καταχωρήσεις ροής (flow entries), με βάση τις οποίες γίνεται η δρομολόγηση και η προώθηση δεδομένων. Οι ροές δεδομένων δημιουργούνται δηλαδή ανάλογα με την ζήτηση την κάθε χρονική στιγμή, ενώ ο ελεγκτής προσφέρει δυναμική ανάθεση πόρων.

Υπάρχουν τρεις κατηγορίες επικοινωνίας στο πρωτόκολλο OpenFlow: (α) η ελεγκτής - μεταγωγέας, (β) η ασύγχρονη και (γ) η συμμετρική επικοινωνία. Όλες υλοποιούνται μέσω ενός ασφαλούς καναλιού ελέγχου. Η επικοινωνία ελεγκτής-μεταγωγέας είναι υπεύθυνη για την ανίχνευση χαρακτηριστικών, την παραμετροποίηση, τον προγραμματισμό του μεταγωγέα και την ανάκτηση πληροφοριών. Μια ασύγχρονη επικοινωνία ενεργοποιείται από τον μεταγωγέα χωρίς καμία πρόσκληση από τον ελεγκτή. Χρησιμοποιείται για να ενημερώσει τον ελεγκτή για τη άφιξη πακέτων, την αλλαγή κατάστασης του και τυχόν λάθη που προέκυψαν. Τέλος, μία συμμετρική επικοινωνία υλοποιείται όταν αποστέλλονται μηνύματα χωρίς πρόσκληση από καμία από τις δύο πλευρές, δηλαδή, τόσο ο μεταγωγέας όσο και ο ελεγκτής είναι ελεύθεροι να εκκινήσουν την επικοινωνία χωρίς πρόσκληση από την άλλη πλευρά. Παραδείγματα για συμμετρική επικοινωνία είναι “hello” ή “echo” μηνύματα που μπορούν να χρησιμοποιηθούν για να προσδιοριστεί εάν το κανάλι ελέγχου εξακολουθεί να είναι διαθέσιμο.

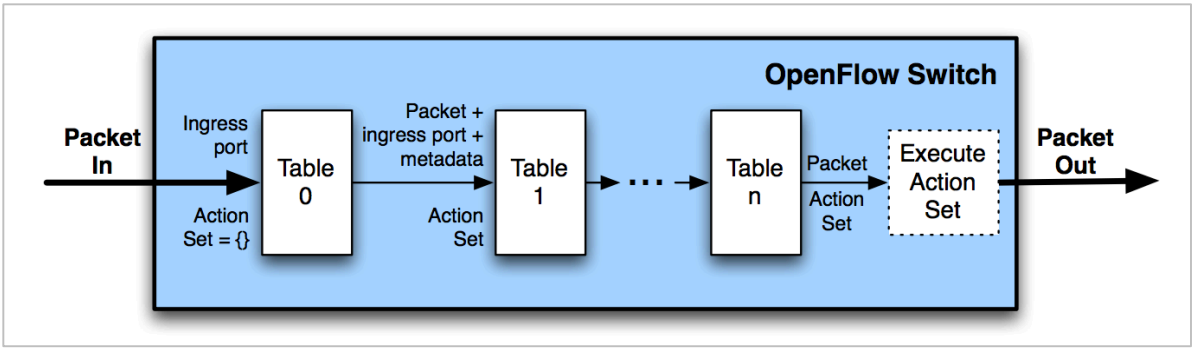
2.1.3 Μεταγωγέας OpenFlow

Ένας μεταγωγέας OpenFlow αποτελείται από έναν ή περισσότερους πίνακες ροής (flow tables) και έναν πίνακα ομάδας, οι οποίοι εκτελούν ανίχνευση πακέτων και προώθηση, και ένα κανάλι OpenFlow προς έναν εξωτερικό ελεγκτή. Ο μεταγωγέας επικοινωνεί με τον ελεγκτή και ο ελεγκτής διαχειρίζεται τον μεταγωγέα μέσα από το πρωτόκολλο OpenFlow.



Εικόνα 2.2: Ένας OpenFlow μεταγωγέας επικοινωνεί με τον ελεγκτή μέσω ασφαλούς σύνδεσης χρησιμοποιώντας το πρωτόκολλο [OpenFlow Switch Specification, Version 1.4.0]

Χρησιμοποιώντας το πρωτόκολλο OpenFlow, ο ελεγκτής μπορεί να προσθέσει, να ανανεώσει ή να διαγράψει καταχωρήσεις ροής (flow entries) στους πίνακες ροής, διαδραστικά (σαν απάντηση σε αιτήσεις πακέτων) ή προληπτικά. Κάθε πίνακας ροής περιέχει μια συλλογή καταχωρήσεων ροής, οι οποίες αποτελούνται από πεδία αντιστοιχίας (match fields), μετρητές (counters), και μια συλλογή οδηγιών, για να εφαρμόσουν στα αντιστοιχισμένα πακέτα. Η αντιστοίχιση ξεκινά από τον πρώτο πίνακα ροής και μπορεί να συνεχίσει και στους υπολοίπους αν υπάρχουν, αφού οι πίνακες ροής βρίσκονται σε διασωληνωμένη μορφή (pipelined). Εάν βρεθεί καταχώρηση που να ταιριάζει με το πακέτο, τότε εκτελούνται οι οδηγίες που συνοδεύουν την συγκεκριμένη καταχώρηση, αλλιώς το πακέτο προωθείται στον ελεγκτή OpenFlow, χάνεται ή συνεχίζει την αντιστοίχιση στον επόμενο πίνακα ροής.



Εικόνα 2.3: Διαδρομή πακέτου σε ένα OpenFlow μεταγωγέα [OpenFlow Switch Specification, Version 1.4.0 (Wire Protocol 0x05), October 2013]

Καταχωρήσεις ροής μπορούν να προωθηθούν σε μια θύρα. Η θύρα μπορεί να είναι φυσική, αλλά μπορεί να είναι και εικονική που ορίζεται από τον μεταγωγέα. Οι εικονικές θύρες μπορούν να προσδιορίσουν διαδικασίες προώθησης, όπως αποστολή στον ελεγκτή, υπερχείλιση ή προώθηση χρησιμοποιώντας μεθόδους χωρίς την χρήση OpenFlow, όπως συμβατική λειτουργία μεταγωγέα.

Εκτός από την ξεχωριστή επεξεργασία των πακέτων, μπορούν να επεξεργαστούν μαζικά με την χρήση πινάκων ομάδας (group tables). Μια ομάδα περιέχει συλλογή διαδικασιών για υπερχείλιση, αλλά και πιο πολύπλοκες λειτουργίες, όπως πολυδιόδευση πακέτων (multipath), γρήγορη επαναδρομολόγηση (fast reroute) και συσσωμάτωση ζεύξεων (link aggregation).

2.2 Εκδόσεις OpenFlow

Το OpenFlow θεωρείται ένα από τα πρώτα Software-Defined Network (SDN) standards. Αυτό όρισε πρώτο το πρωτόκολλο επικοινωνίας στα SDN δίκτυα όπου επέτρεπε στον OpenFlow ελεγκτή να αλληλοεπιδρά με το επίπεδο προώθησης (forwarding plane) και να κάνει τροποποιήσεις στο δίκτυο. [4]

Η τελευταία επίσημη έκδοση OpenFlow είναι η 1.5, ενώ η τελευταία έκδοση η οποία υποστηρίζεται από τους προμηθευτές μεταγωγέων είναι η 1.4. Παρακάτω αναφέρονται επιγραμματικά τα σημαντικότερα χαρακτηριστικά κάθε νέας έκδοσης από το OpenFlow 1.0 και έπειτα.

1.1: Υποστήριξη για MPLS, Q-in-Q, VLANs, multipath, multiple tales, logical ports
1.2: Υποστήριξη για επεκτάσιμες επικεφαλίδες (in match, packet_in, set_field), IPv6
1.3: Υποστήριξη για tunneling, per-flow traffic meters, Provider Backbone Bridging
1.4: Υποστήριξη για synchronized tables, flow monitoring, vacancy events [5]

2.3 Ελεγκτής POX

Ο POX είναι μια ανοιχτού κώδικα πλατφόρμα προγραμματισμού για την δημιουργία SDN εφαρμογών ελέγχου. Αποτελεί την Python μορφή του NOX. Οι προγραμματιστές μπορούν να χρησιμοποιήσουν τον POX για να δημιουργήσουν έναν SDN ελεγκτή χρησιμοποιώντας την γλώσσα προγραμματισμού Python.

Επιπλέον, ο POX δίνει την δυνατότητα στους προγραμματιστές να δημιουργήσουν πιο περίπλοκους SDN ελεγκτές δημιουργώντας καινούρια POX προγράμματα (components) ή γράφοντας εφαρμογές δικτύου οι οποίες απευθύνονται στο POX API.

2.3.1 POX Προγράμματα (Components)

Τα POX components είναι επιπρόσθετα προγράμματα σε Python, τα οποία μπορούν να ενεργοποιηθούν από την γραμμή εντολών μαζί με την εκκίνηση του POX. Ο POX μπορεί άμεσα να χρησιμοποιηθεί ως κύριος SDN ελεγκτής χρησιμοποιώντας τα διαθέσιμα components με τα οποία συνοδεύεται. [6]

2.3.2 POX Core object

Ο POX έχει ένα αντικείμενο το οποίο ονομάζεται "core" και το οποίο λειτουργεί ως ένα κεντρικό σημείο για τα πολλά APIs του POX. Ένας από τους βασικούς λόγους ύπαρξης του core αντικειμένου είναι η παροχή ενός σημείου αναφοράς μεταξύ των διαφορετικών components. Έτσι, αντί να εισάγουμε ένα component μέσα σε άλλο component με σκοπό την αλληλεπίδρασή τους, όλα τα components εγγράφουν τον εαυτό τους στο core αντικείμενο έτσι ώστε να μπορεί να τα ζητήσει οποιοσδήποτε από εκεί. Βασικό πλεονέκτημα αυτής της προσέγγισης είναι ότι οι εξαρτήσεις μεταξύ των components δεν είναι σταθερές και έτσι μπορούν εύκολα διαφορετικά components που εκθέτουν την ίδια διεπαφή να συνεργαστούν.

2.3.3 POX features

Το POX παρέχει αρκετές σημαντικές λειτουργίες οι οποίες βοηθάνε στην γρήγορη ανάπτυξη εφαρμογών, με τις πιο σημαντικές:

- OpenFlow διεπαφή σε Python
- Επαναχρησιμοποιήσιμα components για επιλογή μονοπατιού, εύρεση τοπολογίας κ.α.
- Τρέχει και μπορεί να εγκατασταθεί σε οποιοδήποτε μηχάνημα
- Συγκεκριμένα στοχεύει σε Linux, Mac OS και Windows
- Υποστηρίζει το ίδιο γραφικό περιβάλλον χρήσης και οπτικά εργαλεία με το NOX
- Αποδίδει καλά σε σύγκριση με NOX εφαρμογές γραμμένες σε Python

2.4 IPv6

Το IPv6 είναι η πιο πρόσφατη αναθεώρηση του πρωτοκόλλου Internet (IP), του βασικού πρωτοκόλλου επικοινωνίας πάνω στο οποίο έχει χτιστεί ολόκληρο το διαδίκτυο. Πρόκειται να αντικαταστήσει την υπάρχουσα έκδοση IPv4, η οποία χρησιμοποιείται μέχρι σήμερα (2015). Το IPv6 αναπτύχθηκε από την Τακτική Δύναμη Μηχανικών του Internet (Internet Engineering Task Force), για να ασχοληθεί με το χρόνιο πρόβλημα της εξάντλησης των διευθύνσεων του IPv4.

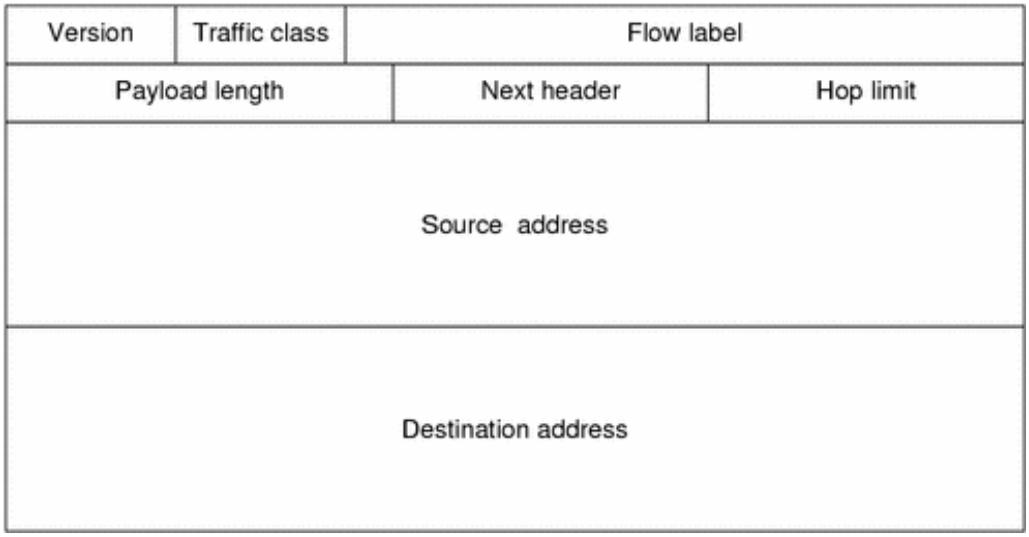
Το IPv6 είναι η νέα έκδοση του πρωτοκόλλου του διαδικτύου και έχει σχεδιαστεί περισσότερο ως βήμα εξέλιξης παρά ως επαναστατικό βήμα από το IPv4. Οι λειτουργίες οι οποίες δούλευαν στο IPv4 παρέμειναν και στο IPv6. Οι λειτουργίες οι οποίες δεν χρησιμοποιούνταν συχνά είτε αφαιρέθηκαν είτε έγιναν προαιρετικές. [7]

Τα πιο σημαντικά χαρακτηριστικά του IPv6 περιλαμβάνουν:

- Διευρυμένη διευθυνσιοδότηση και δυνατότητες δρομολόγησης. Το μέγεθος της διεύθυνσης IP αυξήθηκε από 32 bits σε 128 bits έχοντας πλέον την δυνατότητα να υποστηρίξει μεγαλύτερο αριθμό διευθυνσιοδοτημένων κόμβων, περισσότερα ιεραρχικά επίπεδα διευθυνσιοδότησης και απλούστερη αυτόματη διαμόρφωση των διευθύνσεων.
- Απλοποιημένη διάταξη επικεφαλίδας. Μερικά πεδία της επικεφαλίδας IPv4 διαγράφηκαν ή έγιναν προαιρετικά, προκειμένου να μειωθεί ο χρόνος επεξεργασίας του κάθε πακέτου στην πιο κοινή περίπτωση και να παραμείνει το εύρος ζώνης του overhead της επικεφαλίδας όσο το δυνατόν πιο χαμηλό.
- Υποστήριξη επικεφαλίδας επέκτασης και επιλογών. Οι IPv6 επιλογές τοποθετούνται σε ξεχωριστή επικεφαλίδα, η οποία τοποθετείται μεταξύ της επικεφαλίδας IPv6 και αυτής του επιπέδου μεταφοράς.
- Υποστήριξη για ταυτοποίηση και ιδιωτικότητα. Το IPv6 περιλαμβάνει την δυνατότητα μιας επέκτασης η οποία υποστηρίζει την ταυτοποίηση και την ακεραιότητα των δεδομένων.
- Υποστήριξη για αυτόματη διαμόρφωση. Το IPv6 παρέχει πολλαπλούς τρόπους αυτόματης διαμόρφωσης, από την “plug and play” διαμόρφωση της διεύθυνσης ενός κόμβου σε ένα απομονωμένο δίκτυο έως τις ολοκληρωμένες δυνατότητες που προσφέρει το DHCP.
- Απλή και ευέλικτη μετάβαση από το το IPv4.
- Δυνατότητες παροχής ποιότητας υπηρεσιών (quality of service). Προστέθηκε η δυνατότητα ενεργοποίησης της ετικέτας των πακέτων μια συγκεκριμένης ροής πακέτων για την οποία ο αποστολέας ζητάει ειδική μεταχείριση.

2.4.1 Διάταξη επικεφαλίδας

Η επικεφαλίδα IPv6, παρότι μεγαλύτερη από την επικεφαλίδα IPv4, είναι αισθητά πιο απλοποιημένη.



Εικόνα 2.4: Η διάταξη επικεφαλίδας ενός IPv6 πακέτου [https://docs.oracle.com/cd/E19683-01/817-0573/images/HeaderFormat.epsi.gif]

* Έκδοση (Version) - Ο αριθμός έκδοσης του Πρωτοκόλλου Internet. Για τα IPv6 είναι ο αριθμός 6. (4-bit)

* Ετικέτα Ροής (Flow label) - Το πεδίο αυτό μπορεί να χρησιμοποιηθεί από τον αποστολέα έτσι ώστε να μαρκάρει τα πακέτα για τα οποία θα ήθελε ειδική μεταχείριση από τους δρομολογητές. (28-bit)

* Μήκος Ωφέλιμου Φορτίου (Payload length) - Το μήκος του φορτίου που ακολουθεί την IPv6 επικεφαλίδα σε οκτάβες. (16-bit)

* Επόμενη Επικεφαλίδα (Next header) - Ταυτοποιεί τον τύπο της επόμενης επικεφαλίδας μετά την IPv6. Χρησιμοποιεί τις ίδιες τιμές με το πεδίο “protocol” του IPv4. (8-bit)

* Όριο αλμάτων (Hop limit) - Χρησιμοποιείται για να περιορίσει την επιρροή των ατέρμονων βρόχων δρομολόγησης. Το πεδίο όριο αλμάτων μειώνεται κατά ένας σε κάθε κόμβο που προωθεί το πακέτο. Το πακέτο απορρίπτεται όταν η τιμή του πεδίου γίνει 0. (8-bit)

* Διεύθυνση Αποστολέα (Source Address) - Η διεύθυνση του αρχικού αποστολέα του πακέτου. (128 bit)

* Διεύθυνση Παραλήπτη (Destination Address) - Η διεύθυνση του επιθυμητού παραλήπτη. (128 bit)

2.4.2 IPv6 διευθύνσεις

Μια διεύθυνση IPv6 αποτελείται από 128 bits. Οι διευθύνσεις ομαδοποιούνται σε ποικίλα είδη ανάλογα με την εφαρμογή τους στις βασικές μεθόδους διευθυνσιοδότησης και δρομολόγησης: unicast, multicast και anycast.

Μία διεύθυνση unicast ταυτοποιεί μια διεπαφή δικτύου και στέλνει το πακέτο μόνο σε αυτή. Μία διεύθυνση anycast συνδέεται με μια ομάδα διεπαφών που ανήκουν σε διαφορετικούς κόμβους αλλά το πακέτο στέλνεται σε έναν από τους κόμβους αυτούς, συνήθως τον κοντινότερο. Μία διεύθυνση multicast ταυτοποιεί όσες διεπαφές έχουν συμμετάσχει στην αντίστοιχη multicast ομάδα και στέλνει το πακέτο σε όλες τις συμμετέχουσες επαφές.

Το IPv6 δεν υποστηρίζει την διευθυνσιοδότηση broadcast αλλά την θέση του έχει πάρει η διευθυνσιοδότηση multicast σε όλους τους τοπικούς κόμβους με την χρήση της multicast ομάδας ff02::1.

Η τυπική διάταξη μιας διεύθυνσης unicast:

bits	48 (or more)	16 (or fewer)	64
field	*routing prefix*	*subnet id*	*interface identifier*

Εικόνα 2.5: Διάταξη μιας διεύθυνσης unicast [https://en.wikipedia.org/wiki/IPv6_address]

Όπως για παράδειγμα η διεύθυνση fd76:fce7:1e68:dcc1::1/64.

Η τυπική διάταξη μιας τοπική διεύθυνση:

bits	10	54	64
field	*prefix*	*zeroes*	*interface identifier*

Εικόνα 2.6: Διάταξη μιας τοπικής διεύθυνσης [https://en.wikipedia.org/wiki/IPv6_address]

Το πεδίο prefix (πρόθεμα) περιέχει την τιμή 1111111010. Τα μηδενικά που ακολουθούν κάνουν το πρόθεμα δικτύου ίδιο για όλες τις τοπικές διευθύνσεις (fe80::64 πρόθεμα τοπικής διεύθυνσης), κάνοντάς τες μη δρομολογήσιμες εκτός δικτύου. Ένα παράδειγμα τοπικής διεύθυνσης: fe80::200:ff:fe00:3/64[8]

Η διάταξη μιας Solicited-Node multicast διεύθυνσης:

bits	8	4	4	79	9	24
field	*prefix*	*flg*	*sc*	*zeroes*	*ones*	*unicast address*

Εικόνα 2.7: Διάταξη μιας Solicited-Node multicast διεύθυνσης [https://en.wikipedia.org/wiki/IPv6_address]

Στην μέση περίπτωση τα πρώτα 104 bits της διεύθυνσης αυτής είναι προκαθορισμένα. Συγκεκριμένα, το prefix έχει την τιμή 11111111, το flag έχει την τιμή 0000, και το sc έχει την τιμή 0010. Στη συνέχεια έχουμε 79 μηδενικά και 9 άσσους. Τα τελευταία 24 bits της συμπληρώνονται από τα τελευταία 24 bits της διεύθυνσης για την οποία στάλθηκε το μήνυμα αυτό. Ένα παράδειγμα solicited-node multicast διεύθυνσης: ff02::1:ff00:2/64

Άλλες σημαντικές διευθύνσεις: [9]

Διεύθυνση	**Επεξήγηση**
::/0	Η προεπιλεγμένη unicast διεύθυνση δρομολόγησης. Αντιστοιχεί στην 0.0.0.0/0 του IPv4.
::1/128	Η διεύθυνση βρόχου επιστροφής. Όταν μια εφαρμογή στέλνει πακέτα στην διεύθυνση αυτή τότε τα πακέτα επιστρέφουν στο ίδια διεπαφή απ' όπου στάλθηκαν. Αντιστοιχεί στην 127.0.0.1/8 του IPv4.
ff02::1	Η διεύθυνση πολλαπλής διανομής για όλους τους κόμβους που βρίσκονται στην τοπική ζεύξη. Αντιστοιχεί προσεγγιστικά στο broadcast του IPv4

2.4.3 Πρωτόκολλο Ανακάλυψης Γείτονα (Neighbor Discovery Protocol - NDP)

Το πρωτόκολλο εύρεσης γείτονα (NDP) είναι ένα πρωτόκολλο το οποίο χρησιμοποιείται μόνο μαζί με την έκδοση έξι του Internet Protocol. Λειτουργεί στο επίπεδο δύο του OSI μοντέλου και είναι υπεύθυνο για τις διευθύνσεις αυτόματης διαμόρφωσης των κόμβων, την εύρεση άλλων κόμβων στο τοπικό δίκτυο, την εύρεση των επιπέδου δύο διευθύνσεων των άλλων κόμβων, την εύρεση διπλών διευθύνσεων στο ίδιο τοπικό δίκτυο, την εύρεση των διαθέσιμων δρομολογητών και των εξυπηρετητών Συστήματος Ονομάτων Τομέων (Domain Name System – DNS), την εύρεση των προθεμάτων των δικτύων και την διατήρηση προσβάσιμων πληροφοριών σχετικά με τα μονοπάτια προς άλλους ενεργούς γειτονικούς κόμβους. [10]

Το πρωτόκολλο ορίζει πέντε διαφορετικούς τύπους ICMPv6 πακέτων για την υλοποίηση στο IPv6 λειτουργιών όπως το πρωτόκολλο επίλυσης διεύθυνσης (ARP) και τα πρωτόκολλα Εύρεσης Δρομολογητή (Router Discovery) και Ανακατεύθυνσης Δρομολογητή

(Router Redirect) του πρωτοκόλλου μηνυμάτων ελέγχου του διαδικτύου (ICMP). Επιπλέον, παρέχει διάφορες βελτιώσεις σε σχέση με το IPv4 όπως για παράδειγμα την Εύρεση Απροσιτότητας Γείτονα (Neighbor Unreachability Detection) το οποίο βελτιώνει αξιοπιστία των παραδόσεων των πακέτων στην παρουσίαση των αποτυχημένων δρομολογητών, συνδέσμων ή κινούμενων κόμβων.

Το πρωτόκολλο εύρεσης γείτονα ορίζει πέντε τύπους ICMPv6 πακέτων. Αυτοί είναι:

Παράκληση δρομολογητή (Router Solicitation), τύπος 133.

Οι ξενιστές ρωτούν με μηνύματα παράκλησης δρομολογητή για να εντοπίσουν τους δρομολογητές σε έναν προσαρτημένο σύνδεσμο. Οι κόμβοι που διαβιβάζουν πακέτα που δεν απευθύνονται σε αυτούς, παράγουν μηνύματα διαφήμισης αμέσως μετά την παραλαβή αυτού του πακέτου και όχι κατά την επόμενη προγραμματισμένη χρονική στιγμή.

Διαφήμιση δρομολογητή (Router Advertisement), τύπος 134.

Οι δρομολογητές διαφημίζουν την παρουσία τους μαζί με διάφορους συνδέσμους και παραμέτρους του διαδικτύου περιοδικά ή ως απάντηση σε ένα μήνυμα παράκλησης δρομολογητή.

Παράκληση Γείτονα (Neighbor Solicitation), τύπος 135.

Οι παρακλήσεις γείτονα χρησιμοποιούνται από τους κόμβους για να καθορίσουν την τοπική διεύθυνση ενός γείτονα ή για να επιβεβαιώσουν ότι ένας γείτονας είναι ακόμα προσβάσιμος μέσα από την τοπική διεύθυνση που έχουμε στην κρυφή μνήμη (cache).

Διαφήμιση Γείτονα (Neighbor Advertisement), τύπος 136.

Οι διαφημίσεις γείτονα χρησιμοποιούνται από τους κόμβους ως απάντηση στα μηνύματα παράκλησης γείτονα.

Ανακατεύθυνση (Redirect), τύπος 137.

Οι δρομολογητές ενημερώνουν τους ξενιστές για την ύπαρξη ενός καλύτερου πρώτου δρομολογητή για να φτάσουν σε κάποιον προορισμό. [11] [12]

3 PYRETIC

Η διαχείριση ενός δικτύου απαιτεί την παράλληλη υποστήριξη πολλών διαφορετικών λειτουργιών, από την δρομολόγηση και την παρακολούθηση της κίνησης του δικτύου ως τον έλεγχο πρόσβασης και τον ισόποσο καταμερισμό του φορτίου στους εξυπηρετητές. Το SDN επιτρέπει στις εφαρμογές να υλοποιούν εργασίες απευθείας, εγκαθιστώντας στους μεταγωγείς κανόνες επεξεργασίας πακέτων. Παρόλα αυτά, οι υπάρχουσες πλατφόρμες SDN δεν υποστηρίζουν σε ικανοποιητικό βαθμό την υλοποίηση εφαρμογών χωρισμένες σε μικρότερα ανεξάρτητα κομμάτια (modules), προωθώντας με τον τρόπο αυτό την δημιουργία μεγάλων, μονολιθικών δύσκολα επεξεργάσιμων εφαρμογών.

Το Pyretic ήρθε να λύσει το πρόβλημα αυτό προσφέροντας σημαντικές απλοποιήσεις στην δημιουργία ανεξάρτητων εφαρμογών οι οποίες μπορούν να συνδυαστούν και να διαχειριστούν την κίνηση ενός δικτύου. Το Pyretic είναι μια νέα γλώσσα και ένα νέο σύστημα το οποίο δίνει την δυνατότητα στους προγραμματιστές να καθορίσουν τις πολιτικές ενός δικτύου σε υψηλό επίπεδο, να τις συνδυάσουν και να τις εκτελέσουν σε διαφορετικά μέρη του δικτύου.

3.1 Η γλώσσα Pyretic

Το Pyretic είναι μια γλώσσα συγκεκριμένου τομέα (Domain-Specific Language) για τον προγραμματισμό OpenFlow δικτύων και έχεις ως βάση την γλώσσα προγραμματισμού Python. Πρόκειται για μια υψηλού επιπέδου SDN γλώσσα που σχεδιάστηκε για να λύσει σημαντικά OpenFlow/POX προγραμματιστικά προβλήματα, απλοποιώντας τον τρόπο δημιουργίας και τον τρόπο συνδυασμού διαφορετικών προγραμμάτων για την διαχείριση ενός δικτύου. Δίνει έτσι την δυνατότητα στους προγραμματιστές να εστιάσουν περισσότερο στις πολιτικές που θέλουν να πετύχουν και λιγότερο στον τρόπο υλοποίησής τους.

Σε σχέση με τις υπάρχουσες πλατφόρμες η γλώσσα Pyretic πετυχαίνει τις απλοποιήσεις αυτές στηριζόμενη σε ένα νέο απλοποιημένο μοντέλο πακέτου (3.1.1), τον συνδυασμό πολιτικών υψηλού επιπέδου (3.1.2 - 3.1.6) και τα αντικείμενα δικτύου (network objects).

3.1.1 Απλοποιημένο μοντέλο πακέτου

Ένα από τα σημαντικότερα στοιχεία του προγραμματιστικού μοντέλου του Pyretic είναι το απλοποιημένο και επεκτάσιμο μοντέλο πακέτου (packet model) του. Κάθε πακέτο που μεταφέρεται μέσα στο δίκτυο είναι ένα λεξικό (dictionary) το οποίο συνδέει ονόματα πεδίων με τις τιμές τους. Τα πεδία αυτά περιλαμβάνουν εγγραφές που αφορούν:

1. Την τοποθεσία του πακέτου (packet location). Συγκεκριμένα τα πεδία switch, inport, outport.
2. Τις απαραίτητες OpenFlow επικεφαλίδες (standard OpenFlow headers). Συγκεκριμένα τα πεδία source mac, destination mac, source IP, destination IP, type of service, source port, destination port, ethernet type, protocol, vlan_id και vlan_pcp.
3. Προσαρμόσιμα δεδομένα (custom data). Τα προσαρμοσμένα δεδομένα βρίσκονται μέσα στα εικονικά πεδία (virtual fields) και υπάρχει η δυνατότητα να αναπαραστήσουν όχι μόνο απλά bit strings αλλά και μια αυθαίρετη δομή δεδομένων. Επομένως, η αναπαράσταση αυτή προσφέρει έναν γενικό τρόπο συσχέτισης των υψηλού επιπέδου πληροφοριών με τα πακέτα και να γίνει ο συντονισμός μεταξύ των modules.

Ένα παράδειγμα ενός πακέτου που εισέρχεται στο δίκτυο από το switch A, την φυσική input port 2 και source port 80 θα αναπαρίσταται: [13]

```
{ switch: A, inport: 3, source port: 80, … }
```

3.1.2 Βασικές Πολιτικές (Basic Policies)

Μια Pyretic πολιτική είναι μια συνάρτηση η οποία λαμβάνει ένα πακέτο σαν είσοδο και επιστρέφει ένα σύνολο πακέτων. Η συνάρτηση περιγράφει τις ενέργειες που πρέπει να κάνει ο μεταγωγέας στα εισερχόμενα πακέτα.

POLICY	SYNTAX	EXAMPLE
match	match(f=v)	match(dstmac=EthAddr('00:00:00:00:00:01'))
drop	drop	drop
identity	identity	identity
modify	modify	modify(srcmac=EthAddr('00:00:00:00:00:01'))
fwd	fwd(a)	fwd(1)
flood	flood()	flood()
parallel composition	A + B	fwd(1) + fwd(2)
sequential composition	A >> B	match(switch=1) >> flood()
negation	~A	~match(switch=1)

Εικόνα 3.1: Οι βασικές πολιτικές του Pyretic

Στον πίνακα της εικόνας 3.1 συνοψίζονται οι βασικές πολιτικές του pyretic. Η πολιτική match επιστρέφει το σύνολο των πακέτων στα οποία το πεδίο f έχει την τιμή v, διαφορετικά δεν επιστρέφει τίποτα (το κενό σύνολο). Η πολιτική drop επιστρέφει το κενό σύνολο, ενώ η πολιτική identity επιστρέφει το πακέτο το οποίο εισήλθε. Η πολιτική modify επιστρέφει το εισερχόμενο πακέτο έχοντας αλλάξει την τιμή του πεδίου f στην τιμή v. Η πολιτική forward επιστρέφει το εισερχόμενο πακέτο έχοντας θέσει την τιμή a στο πεδίο outport. Η πολιτική flood επιστρέφει ένα αντίγραφο του εισερχόμενου πακέτου για κάθε port του μεταγωγέα.

Η παράλληλη και η σειριακή σύνθεση (parallel, sequential composition) είναι από τα πιο σημαντικά στοιχεία του Pyretic καθώς είναι αυτά που δίνουν την δυνατότητα συνδυασμού διαφορετικών modules για την δημιουργία ενός συνολικού policy. Το parallel composition επιστρέφει το αποτέλεσμα και της A και της B πολιτικής, ενώ το sequential composition επιστρέφει το αποτέλεσμα της B πολιτικής όπου το αποτέλεσμα της A πολιτικής είναι η είσοδος της B πολιτικής. Το negation αφορά πολιτικές φιλτραρίσματος και επιτρέπει να περάσουν τα αντίστροφα από αυτά που επιτρέπει η πολιτική φιλτραρίσματος A.

Η πολιτική if_ εκφράζει με βολικό τρόπο τον συνδυασμό 2 συμπληρωματικών πολιτικών. Για παράδειγμα η παρακάτω πολιτική ‘split’ η οποία στέλνει στην port 1 τα πακέτα με προορισμό την IP 1fc00:0:0:1::1 και στην port 2 όλα τα υπόλοιπα:

```
split = (match(dstip=IPAddr('fc00:0:0:1::1')) >> fwd(1)) +
      (~match(dstip=IPAddr('fc00:0:0:1::1')) >> fwd(2))
```

μπορεί να γραφεία μέσω της πολιτικής ‘if_’:

```
split = if_(match(dstip=IPAddr('fc00:0:0:1::1')),fwd(1),fwd(2))
```

3.1.3 Πολιτικές Φιλτραρίσματος (Filter Policies)

Τα filter policies ή αλλιώς φίλτρα (filters) είναι πολιτικές που δεν αλλάζουν το πακέτο αλλά επιστρέφουν είτε το αρχικό πακέτο είτε το κενό σύνολο. Επομένως, λειτουργούν σαν φίλτρο για το ποια πακέτα θα συνεχίσουν στην επόμενη ενέργεια και ποια όχι. Από την προηγούμενη ενότητα, οι πολιτικές match, drop και identity είναι φίλτρα.

Σε αντιστοιχία με την σειριακή και την παράλληλη σύνθεση του κεφαλαίου 3.1.2, στα φίλτρα χρησιμοποιείται ο συνδυασμός (conjunction) (&) και η διάζευξη (disjunction) (|).

3.1.4 Πολιτικές Ερωτημάτων (Query Policies)

Στα παραδοσιακά OpenFlow προγράμματα, η συλλογή των στατιστικών της κίνησης του δικτύου απαιτεί την εγκατάσταση κανόνων, την θέσπιση ερωτημάτων (queries) για την παραλαβή των τιμών των μετρητών (counters), την ανάλυση των απαντήσεων κατά την άφιξή τους και τον συνδυασμό των τιμών των μετρητών από διαφορετικούς κανόνες.

Στο Pyretic, η παρακολούθηση (monitoring) του δικτύου είναι ένα ακόμα είδος πολιτικής, η οποία μπορεί συνδυαστεί με κάποια από τις προηγούμενες πολιτικές. Ο παρακάτω πίνακας δείχνει διάφορα είδη πολιτικών παρακολούθησης τα οποία είναι διαθέσιμα στο Pyretic και περιλαμβάνουν πολιτικές για παρακολούθηση ακατέργαστων πακέτων (raw packets), μετρήσεις πακέτων (packet counts) και μετρήσεις byte (byte counts). Οι πολιτικές αυτές δεν προωθούν τα πακέτα περαιτέρω αλλά τα ρίχνουν (drop) έτσι ώστε χρησιμοποιούνται μόνο για monitoring λόγους και να μην δημιουργηθούν διπλά πακέτα.

Syntax	Summary
packets(limit=n, group_by = [f1,f2,…])	Επανάκληση σε κάθε πακέτο που λαμβάνεται, για μέχρι n πακέτα με ίδια τα fields f1, f2, …
Count_packets (interval=t, group_by = [f1, f2, …])	Μετράει κάθε πακέτο που λαμβάνεται και στέλνει κάθε t δευτερόλεπτα τις μετρήσεις για κάθε group
Count_bytes (interval=t, group_by = [f1, f2, …])	Μετράει κάθε byte που λαμβάνεται και στέλνει κάθε t δευτερόλεπτα τις μετρήσεις για κάθε group

Εικόνα 3.2: Πολιτικές ερωτημάτων του Pyretic

Για παράδειγμα, ένας προγραμματιστής δημιουργεί ένα query για το πρώτο πακέτο που φτάνει από μια συγκεκριμένη source IP:

```
Q = packets(limit=1,group_by=['srcip'])
```

και το περιορίζει μόνο για τα http αιτήματα (όσα έρχονται στην port 80):

```
match(dstport=80) >> Q
```

Αν θέλει να εκτυπώσει τα πακέτα που φτάνουν στο ‘Q’ τότε εγγράφει μια ρουτίνα επανάκλησης (callback routine) η οποία διαχειρίζεται τα callbacks του Q:

```
def printer(pkt):

  print pkt

Q.register_callback(printer)
```

3.1.5 Δυναμικές Πολιτικές (Dynamic Policies)

Οι πολιτικές ερωτημάτων (query policies) συχνά χρησιμοποιούνται για να οδηγούν σε αλλαγές στατικές πολιτικές και να τις μετατρέψουν σε δυναμικές. Οι δυναμικές πολιτικές έχουν συμπεριφορά η οποία αλλάζει στον χρόνο ανάλογα με τις προδιαγραφές του προγραμματιστή.

Για παράδειγμα, στο παρακάτω πρόγραμμα η ρουτίνα round_robin δέχεται το πρώτο πακέτο από ένα νέο πελάτη (νέα IP διεύθυνση πηγής) και ανανεώνει τη συμπεριφορά της πολιτικής έτσι ώστε όλα τα νέα πακέτα από αυτόν τον πελάτη να δρομολογούνται στον επόμενο κατά σειρά εξυπηρετητή. Η αλλαγή αυτή στην πολιτική δεν επηρεάζει τους προηγούμενους πελάτες και μάλιστα αφότου εφαρμοστεί, ο δείκτης δείχνει στον επόμενο εξυπηρετητή της λίστας.

```
def round_robin(self,pkt):
  self.policy = if_(match(srcip=pkt['srcip']),
                    modify(dstip=self.server),
                    self.policy)
  self.client += 1
  self.server = self.servers[self.client % m]
```

Ο προγραμματιστής δημιουργεί μια νέα “round-robin load balancer” κλάση δυναμικής πολιτικής με το όνομα ‘rrlb’, με υποκλάση την δυναμική πολιτική και παρέχοντας μια μέθοδο αρχικοποίησης η οποία καλεί την callback routine:

```
class rrlb(DynamicPolicy):
    def __init__(self,s,servers):
      self.switch = s
      self.servers = servers
      ...
      Q.register_callback(self.round_robin)
      self.policy = match(dstport=80) >> Q

    def round_robin(self,pkt):
       ...
```

Πλέον ο προγραμματιστής έχει την δυνατότητα να χρησιμοποιήσει την κλάση αυτή σε οποιοδήποτε πρόγραμμα. Για παράδειγμα, εάν επιθυμεί ο μεταγωγέας 3 να στέλνει εναλλάξ στους εξυπηρετητές με διεύθυνση IP fc00:0:0:1::1 και fc00:0:0:1::2 τότε αρκεί να γράψει: [14]

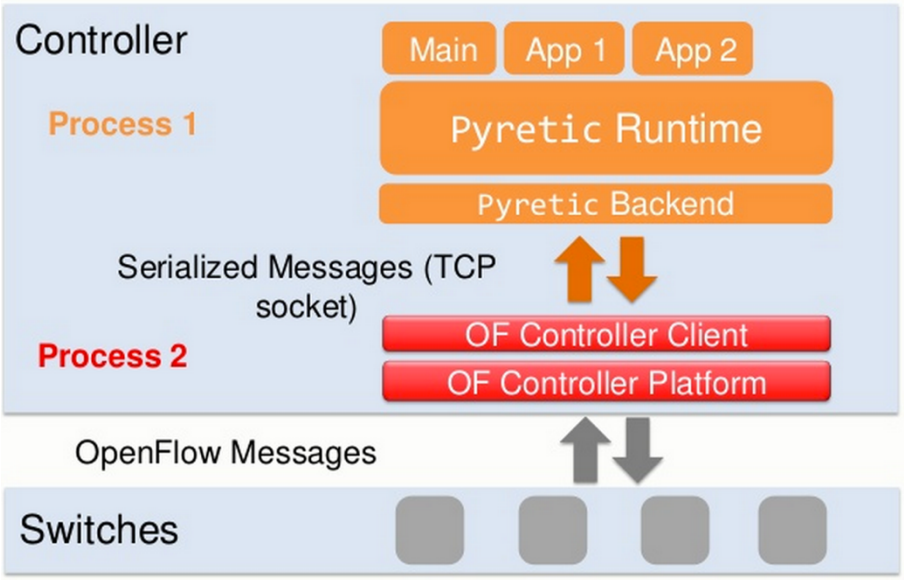
```
servers = [IP('fc00:0:0:1::1'),IP('fc00:0:0:1::2')]
rrlb_on_switch3 = rrlb(3,servers)
```

3.2 Το σύστημα Pyretic runtime

Η υψηλού επιπέδου προγραμματιστική απλοποίηση του Pyretic είναι χρήσιμη μόνο όταν μπορεί να υλοποιηθεί αποδοτικά στους μεταγωγείς. Η ενότητα αυτή παρέχει μια σύντομη επισκόπηση του Pyretic runtime συστήματος, εστιάζοντας στην backend διεπαφή με τους OpenFlow ελεγκτές και την αξιολόγηση των πολιτικών (policy evaluation).

3.2.1 Backend Διεπαφή

Το Pyretic runtime σχεδιάστηκε για να χρησιμοποιηθεί πάνω από ποικίλους διαφορετικούς OpenFlow ελεγκτές. Το runtime συνδέεται μέσω μιας στάνταρ υποδοχής (socket) σε έναν απλό OpenFlow πελάτη (client) ο οποίος θα μπορούσε να γραφτεί πάνω από οποιονδήποτε OpenFlow ελεγκτή. Το runtime διαχειρίζεται το δίκτυο στέλνοντας μηνύματα στον πελάτη όπως για την αλλαγή των κανόνων, το διάβασμα των μετρητών και την διοχέτευση πακέτων. Ομοίως μηνύματα από τον OpenFlow πελάτη κρατούν το Pyretic ενήμερο σχετικά με τα γεγονότα του δικτύου (π.χ. γεγονότα κατάστασης των θυρών, διάβασμα των τιμών των μετρητών). Αυτός ο σχεδιασμός δίνει την δυνατότητα στο Pyretic να εκμεταλλευτεί τον καλύτερα τεχνολογικά διαθέσιμο ελεγκτή και επιτρέπει στο σύστημα να είναι αυτόνομο. Στην παρούσα έκδοση το Pyretic runtime έρχεται σε πακέτο μαζί με έναν OpenFlow πελάτη γραμμένο πάνω στον POX ελεγκτή.



Εικόνα 3.3: Η αρχιτεκτονική της πλατφόρμας Pyretic[http://www.slideshare.net/nvirters/2013-08reichbanv]

3.2.2 Αξιολόγηση πολιτικών

Το Pyretic runtime είναι υπεύθυνο για την λειτουργία ενός διερμηνέα (interpreter) ο οποίος αξιολογεί το πακέτο που εισέρχεται σε έναν μεταγωγέα σε σχέση με την εκάστοτε πολιτική. Στην απλούστερη μορφή του, όλα τα πακέτα αξιολογούνται από τον διερμηνέα. Παράλληλα, το runtime παρακολουθεί τα ενεργά queries, τις ενημερώσεις για τις δυναμικές πολιτικές και τις αλλαγές στην τοπολογία του δικτύου. Στις γενικές του ρυθμίσεις, όταν αυτό είναι ασφαλές να γίνει, το runtime προληπτικά εγκαθιστά κανόνες στους μεταγωγείς προτού αυτά τους χρειαστούν για να αποφύγει περιττή αδράνεια μεταξύ μεταγωγέα - ελεγκτή. [15]

4 ΕΠΙΘΕΣΕΙΣ ΔΙΚΤΥΟΥ ΣΤΟ IPv6

Οι επιθέσεις στα δίκτυα IPv6 δεν έχουν μεγάλες διαφορές με αυτά των δικτύων IPv4. Στο IPv4 εφόσον το Address Resolution Protocol (ARP) είναι ένα πρωτόκολλο χωρίς επίβλεψη της κατάσταση (stateless) λόγω της έλλειψης εξουσιοδότησης στα ARP μηνύματα, πολλές επιθέσεις όπως οι πλαστογραφημένες αιτήσεις/απαντήσεις, Man-in-the-Middle, Denial-of-Service (DoS) είναι πιθανές. Το IPv6 χρησιμοποιεί το Network Discovery Protocol (NDP) για την εύρεση της MAC διεύθυνσης. Το NDP, όπως και το ARP, είναι πρωτόκολλο χωρίς επίβλεψη της κατάστασης και άρα χωρίς εξουσιοδότηση στα μηνύματά του ως προεπιλογή. Έτσι το NDP υποφέρει από πολλές επιθέσεις παρόμοιες με αυτές του ARP. Στο κεφάλαιο αυτό θα περιγράψουμε τις μερικές από τις πιο σημαντικές επιθέσεις στα IPv6 δίκτυα.

4.1 Σκανάρισμα Δικτύου (Network Scanning)

Το σκανάρισμα δικτύου είναι η διαδικασία με την οποία αναγνωρίζονται όλοι οι ενεργοί ξενιστές σε ένα δίκτυο καθώς και οι υπηρεσίες που προσφέρει ο κάθε ένας. Το σκανάρισμα δικτύου χρησιμοποιείται είτε από κάποιον υποψήφιο εισβολέα με σκοπό την διερεύνηση του δικτύου, είτε από τον διαχειριστή του για να αξιολογήσει το επίπεδο ασφάλειας του.

Το σκανάρισμα δικτύου αποτελείται από δύο μέρη, το σκανάρισμα των διευθύνσεων IP και το σκανάρισμα των θυρών. Το πρώτο έχει ως στόχο την εύρεση των ενεργών ξενιστών του δικτύου ενώ το δεύτερο στοχεύει στην εύρεση των υπηρεσιών που προσφέρει ο κάθε ξενιστής. Το σκανάρισμα των διευθύνσεων IP είναι σημαντικό καθώς γνωρίζοντας τις ενεργές διευθύνσεις του δικτύου, ο εισβολέας ουσιαστικά γνωρίζει ποιες συγκεκριμένες διευθύνσεις IP μπορούν να προσπελαστούν μέσα από το διαδίκτυο. [18]

Το σκανάρισμα διευθύνσεων σε ένα δίκτυο IPv6 έχει 2 μεγάλες διαφορές σε σχέση με αυτό του IPv4, την χρήση του NDP έναντι του ARP και την ύπαρξη πολύ μεγαλύτερων δικτύων. Το σκανάρισμα διευθύνσεων έχει διαφορετικές απειλές εάν ο εισβολέας βρίσκεται εντός ή εκτός του δικτύου. Στο 4.1.1 παρουσιάζεται η απειλή όταν ο εισβολέας είναι εντός του δικτύου ενώ στα 4.1.2 - 4.1.3 οι απειλές όταν ο εισβολέας είναι εκτός του δικτύου.

4.1.1 Μέσω της διεύθυνσης πολλαπλής διανομής

Το IPv6 δεν υποστηρίζει το ARP για την εύρεση της διεύθυνσης MAC από την διεύθυνση IP. Στο IPv6, η εύρεση της διεύθυνσης MAC γίνεται μέσα από την διαδικασία του Neighbor Discovery και του Neighbor Solicitation. Το Network Discovery χρησιμοποιεί το ICMPv6 για να καθορίσει ποιες ενεργές τοπικές διευθύνσεις είναι στο τοπικό υποδίκτυο.

Στέλνοντας ένα ICMPv6 πακέτο στην τοπική διεύθυνση πολλαπλής διανομής, το πακέτο αυτό θα φτάσει σε όλες τις ενεργές τοπικές διευθύνσεις του δικτύου. Σύμφωνα με το RFC 3513, η πολλαπλής διανομής διεύθυνση FF02::1 μπορεί να χρησιμοποιηθεί για να σταλεί ένα πακέτο σε όλες τις ενεργές τοπικές διευθύνσεις.

Επομένως, εάν ο εισβολέας είναι εντός του δικτύου, έχει την δυνατότητα να στείλει ένα ICMPv6 πακέτο στην διεύθυνση FF02::1 και να μάθει ποιες διευθύνσεις είναι ενεργές από τις απαντήσεις. Μάλιστα, εάν στείλει από την παγκόσμια (global) IP διεύθυνση του, θα μπορέσει να μάθει από τις απαντήσεις τις παγκόσμιες διευθύνσεις των ενεργών διευθύνσεων του δικτύου.

4.1.2 Μέσω αιτήματος ICMPv6

Στην μέθοδο σκαναρίσματος του 4.2.1 περιγράφηκε ο τρόπος που ένας εισβολέας μπορεί εντός δικτύου να μάθει όλες τις ενεργές διευθύνσεις του δικτύου. Στο 4.2.2 θα δούμε πως μπορεί ο εισβολέας εκτός δικτύου να μάθει τις ενεργές IPv6 διευθύνσεις ενός δικτύου.

Στέλνοντας ένα ICMPv6 echo request σε μια διεύθυνση IPv6 του δικτύου, εάν η διεύθυνση είναι ενεργή θα απαντήσει με ένα ICMPv6 echo reply ενώ εάν δεν είναι ενεργή, δεν θα απαντήσει καθόλου. Στην περίπτωση αυτή, το μεγάλο εύρος διευθύνσεων IPv6 δυσκολεύει το έργο του εισβολέα ο οποίος θα χρειαστεί να χρησιμοποιήσει κάποιο κείμενο κώδικα (script) με σκοπό να ελέγξει γρήγορα όλες τις διευθύνσεις του δικτύου.

4.1.3 Μέσω inverse mapping

Στην περίπτωση του σκαναρίσματος μέσω inverse mapping ο επιτιθέμενος κάνει ping το οποίο δεν μπορεί να φτάσει μέχρι τον ξενιστή καθώς το ενδιάμεσο τείχος προστασίας ή ο

δρομολογητής είναι υπεύθυνα για να απαντούν στα μηνύματα αυτά. Τυπικά, τα τείχη προστασίας και οι δρομολογητές δεν απαντάνε στα ping requests όταν η διεύθυνση προορισμού υπάρχει στο δίκτυο, αλλά απαντάνε με ένα ICMP host unreachable όταν η διεύθυνση προορισμού δεν είναι διαθέσιμη. Η απουσία απάντησης δίνει την δυνατότητα στον επιτιθέμενο να μαντέψει ποιες διευθύνσεις IP του δικτύου είναι ενεργές και ποιες όχι.[19]

4.2 Ψεύτικες Διαφημίσεις Δρομολογητή (Fake Router Advertisements)

Ένας επιτιθέμενος μπορεί να στείλει σε πολλές διευθύνσεις ένα ψεύτικο Router Advertisement (RA) πακέτο σε μια ζεύξη ή σαν απάντηση σε μηνύματα παράκλησης δρομολογητή (Router Solicitation). Εάν ένα κόμβος επιλέξει έναν ψεύτικο RA τότε ο επιτιθέμενος μπορεί να μεταμφιεστεί σε δρομολογητή και να πραγματοποιεί man-in-the-middle επιθέσεις σε επίπεδο δικτύου. Ο επιτιθέμενος μπορεί επίσης να ξεγελάσει τα RA που έρχονται από τον πραγματικό δρομολογητή αλλάζοντάς τους την διάρκεια ζωής σε μηδέν. Έτσι ο ξενιστής θα νομίζει ότι κανονικός δρομολογητής δεν θα του δρομολογήσει τα πακέτα.

4.3 Πλημμύρα Ανακάλυψης Γείτονα - άρνηση υπηρεσιών (Neighbor Discovery Flood - DoS)

Κάθε ξενιστής στο διαδίκτυο μπορεί να στείλει σε μια αυθαίρετη διεύθυνση ενός δεδομένου προθέματος. Εάν ο τελευταίος δρομολογητής δεν έχει επίγνωση της διεύθυνσης του πακέτου, τότε αναγκαστικά θα στείλει ένα Neighbor Solicitation με στόχο να βρει την διεύθυνση του κόμβου αυτού. Ο επιτιθέμενος μπορεί να πλημμυρίσει τον δρομολογητή με πακέτα τα οποία να απαιτούν κάποια Neighbor Discovery δραστηριότητα με αποτέλεσμα να χρησιμοποιούν πολλούς πόρους του δρομολογητή.

4.4 Πλαστογράφηση Παράκλησης/Διαφήμισης Δρομολογητή (Neighbor Solicitation/Advertisement Spoofing)

Ένας επιτιθέμενος μπορεί να προκαλέσει πακέτα τα οποία στέλνονται σε πραγματικούς κόμβους, είτε ξενιστές είτε δρομολογητές, να σταλούν τελικά σε διαφορετικές επιπέδου ζεύξης (link-layer) διευθύνσεις από τις πραγματικές. Αυτό μπορεί να επιτευχθεί είτε στέλνοντας ένα Neighbor Solicitation μήνυμα με διαφορετική επιπέδου ζεύξης διεύθυνση

πηγής, είτε στέλνοντας ένα Neighbor Advertisement μήνυμα με διαφορετική επιπέδου ζεύξης διεύθυνση στόχου. Η επίθεση πετυχαίνει καθώς η νέα εγγραφή στην κρυφή μνήμη γείτονα (Neighbor Cache) αντικαθιστά την παλιά. Εάν η πλαστογραφημένη διεύθυνση επιπέδου ζεύξης είναι πραγματική τότε όσο ο επιτιθέμενος απαντάει στα Neighbor Solicitation μηνύματα που στέλνονται ως μέρους του μηχανισμού Ανίχνευση μη Προσβασιμότητα Γείτονα (Neighbor Unreachability Detection), τα πακέτα θα συνεχίσουν να ανακατευθύνονται.

4.5 Ρύθμιση Ψεύτικου Προθέματος Διεύθυνσης (Bogus Address Configuration Prefix)

Ο επιτιθέμενος κόμβος μπορεί να στείλει ένα Router Advertisement μήνυμα ορίζοντας ένα άκυρο πρόθεμα υποδικτύου να χρησιμοποιηθεί από έναν ξενιστή για αυτόματη διαμόρφωση διεύθυνσης. Ο ξενιστής εκτελώντας τον αλγόριθμο αυτόματης διαμόρφωσης διεύθυνσης χρησιμοποιεί το διαφημιζόμενο πρόθεμα για την κατασκευή μια διεύθυνσης ακόμα και αν η διεύθυνση δεν είναι σωστή για το υποδίκτυο. Ως αποτέλεσμα, τα πακέτα επιστροφής ποτέ δεν φτάνουν στον ξενιστή επειδή η διεύθυνση πηγής του ξενιστή είναι μη έγκυρη.

Η επίθεση έχει την δυνατότητα να διαδοθεί πέρα από την άμεση επίθεση στον ξενιστή εάν ο επιτιθέμενος ξενιστής πραγματοποιήσει μια δυναμική ενημέρωση του Συστήματος Ονομάτων Τομέων (DNS) βασιζόμενος στις ψεύτικες διευθύνσεις. Σε περίπτωση που συμβεί αυτό, οι εφαρμογές που πραγματοποιούν ανάλυση ονομάτων μέσω του DNS αποκτούν την ψεύτικη διεύθυνση και κάθε προσπάθεια να επικοινωνήσουν με τον ξενιστή αποτυγχάνει.

4.6 Επίθεση Ανίχνευσης Διπλής Διεύθυνσης (Duplicate Address Detection Attack - DAD)

Στα δίκτυα που ένας νέος ξενιστής αποκτάει την διεύθυνσή του χρησιμοποιώντας την χωρίς επίβλεψη κατάστασης αυτόματη ρύθμιση διεύθυνσης, ένας επιτιθέμενος κόμβος μπορεί να ξεκινήσει μια άρνηση υπηρεσιών (DoS) επίθεση απαντώντας σε κάθε προσπάθεια ανίχνευσης διπλής χρήσης κάποιας διεύθυνσης. Εάν ο επιτιθέμενος υποστηρίξει ότι του ανήκει μια διεύθυνση τότε ο ξενιστής δεν θα μπορεί να αποκτήσει την διεύθυνση αυτή. Όσο αυτό συνεχίζει να γίνεται για κάθε διεύθυνση ο ξενιστής δεν θα μπορέσει να αποκτήσει ποτέ κάποια διεύθυνση. Ο επιτιθέμενος μπορεί να υποστηρίξει ότι έχει μια διεύθυνση με δύο τρόπους, είτε να απαντήσει με ένα Neighbor Solicitation, προσομοιώνοντας ότι πραγματοποιεί και αυτός DAD, είτε απατώντας με ένα Neighbor Advertisement, προσομοιώνοντας ότι έχει ήδη καταλάβει την διεύθυνση αυτή. Η απειλή αυτή εντοπίστηκε

στο RFC 2462 και μπορεί να εμφανιστεί και σε άλλου είδους ρυθμίσεις διεύθυνσης. Αποτελεί μια DoS επίθεση. [20]

4.7 Ψεύτικος DHCPv6 εξυπηρετητής (Fake DHCPv6 Server)

Το Πρωτόκολλο Δυναμικής Ρύθμισης Ξενιστή (Dynamic Host Configuration Protocol) είναι ευάλωτο σε αρκετές επιθέσεις, ιδιαίτερα σε ψεύτικες επιθέσεις. Ένας επιτιθέμενος έχει την δυνατότητα να χρησιμοποιήσει μια ψεύτικη διεύθυνση για να ξεγελάσει το πρωτόκολλο ή να ξεκινήσει μια επίθεση. Ένας κακόβουλος ψεύτικος DHCPv6 εξυπηρετητής μπορεί να παρέχει λανθασμένες ρυθμίσεις στους πελάτες έτσι ώστε να επικοινωνούν με κακόβουλες υπηρεσίες όπως το DNS ή το NTP. Μπορεί επιπλέον να εξαπολύσει μια DoS επίθεση μέσω της κακής ρύθμισης του πελάτη ή να μαζέψει κρίσιμες πληροφορίες του πελάτη. [21]

5 ΣΥΣΤΗΜΑ ΥΛΟΠΟΙΗΣΗΣ ΑΣΦΑΛΕΙΑΣ ΜΕ PYRETIC

Στόχος μας είναι να δημιουργήσουμε ένα σύστημα μέσω του Pyretic το οποίο να μπορεί εύκολα να εντοπίζει και να αντιμετωπίζει επιθέσεις σε δίκτυα IPv6. Το Pyretic μας βοηθάει σε αυτό με την εύκολη χρήση των πολιτικών του και την απλοποιημένο τρόπο δομής των εφαρμογών του χάρη στην υψηλού επιπέδου SDN γλώσσα του. Το Pyretic υποστηρίζει μόνο IPv4 και επομένως θα χρειαστεί να του κάνουμε τις απαραίτητες τροποποιήσεις έτσι ώστε να λειτουργεί σε περιβάλλον IPv6. Στο κεφάλαιο αυτό, περιγράφεται το περιβάλλον στο οποίο έγινε η εργασία, η τροποποίηση του Pyretic έτσι ώστε να μπορεί να ανταποκριθεί σε IPv6 δίκτυα, οι βελτιώσεις του Pyretic για την δημιουργία IPv6 πολιτικών και δύο εφαρμογές που δημιουργήθηκαν για την αντιμετώπιση σκαναρίσματος δικτύου σε ένα IPv6 δίκτυο και αποτελούν την βάση για την αντιμετώπιση πολλών άλλων επιθέσεων.

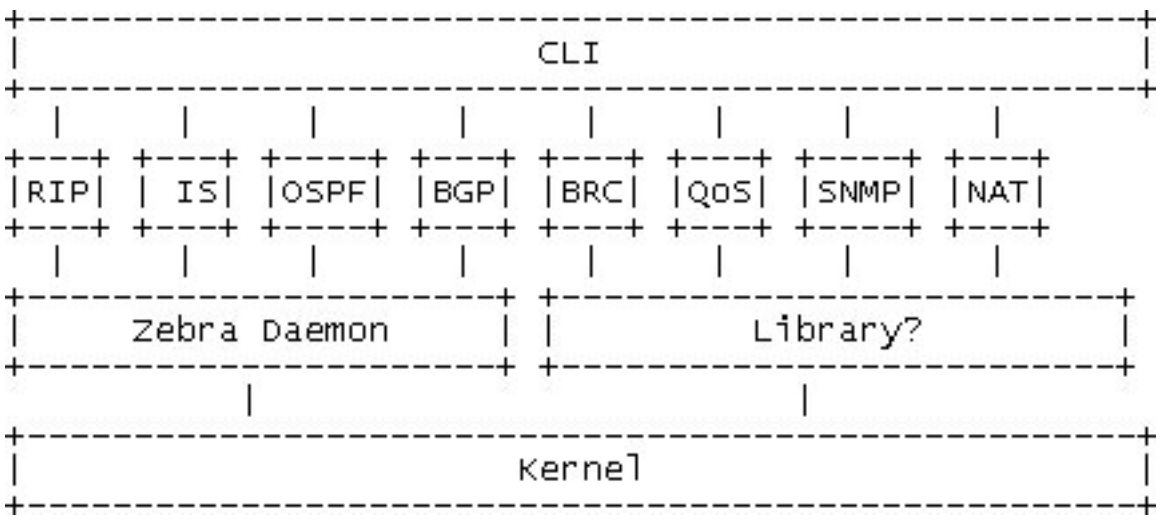
5.1 Προετοιμασία

Στην προετοιμασία περιλαμβάνονται όλες οι αλλαγές και τα εργαλεία που χρειάστηκε να χρησιμοποιήσουμε έτσι ώστε να προσομοιώσουμε ένα σύστημα IPv6 στο οποίο οι μεταγωγείς ακολουθούν τις πολιτικές τις οποίες έχουμε εμείς επιλέξει από την Pyretic εφαρμογή μας. Τα κύρια εργαλεία ήταν το MiniNExT και το Quagga. Όσον αφορά το IPv6 κομμάτι, χρειάστηκε να τροποποιήσουμε αρχικά το Runtime σύστημα και την γλώσσα του Pyretic έτσι ώστε να περιλαμβάνει τις διευθύνσεις IPv6 και στην συνέχεια να χρησιμοποιήσουμε το Nicira Extension με σκοπό την προσομοίωση ενός IPv6 συστήματος σε OpenFlow 1.0.

5.1.1 Quagga

Το Quagga είναι ένα λογισμικό δρομολόγησης ανοιχτού κώδικα το οποίο αναπαριστά την λειτουργία ενός πραγματικού δρομολογητή. Η αρχιτεκτονική του περιλαμβάνει έναν πυρήνα δαίμονα, τον zebra, ο οποίος λειτουργεί ως ένα στρώμα αφαίρεσης στο υποκείμενο πυρήνα Unix (kernel) και παρουσιάζει το Zserv API πάνω από ένα Unix ή TCP ρεύμα

(stream) σε Quagga πελάτες. Μας είναι χρήσιμο καθώς έχει την δυνατότητα να δρομολογεί πακέτα IPv4 και IPv6.



Εικόνα 5.1: Η δομή ενός συστήματος που τρέχει Quagga[http://www.linuxplanet.org/blogs/?cat=5036]

5.1.2 MiniNExT

Το πιο συνηθισμένο εργαλείο για την προσομοίωση SDN δικτύων είναι το Mininet, ένας προσομοιωτής δικτύων ο οποίος χρησιμοποιεί ελαφριά εικονικοποίηση (virtualization) για να κάνει ένα σύστημα να λειτουργεί σαν ένα ολόκληρο δίκτυο. Στην περίπτωσή μας, χρειαζόμασταν τουλάχιστον ένα δρομολογητή στο δίκτυό μας και θέλαμε και να υπάρχει η δυνατότητα προσθήκης και άλλων δρομολογητών σε περίπτωση επέκτασης της εργασίας. Έτσι, επιλέξαμε να χρησιμοποιήσουμε μια επέκταση του Mininet, το MininNExT. Το MininNExT (Mininet ExTended) είναι ένα επίπεδο επέκτασης το οποίο διευκολύνει την δημιουργία σύνθετων δικτύων στο Mininet.

Το γεγονός ότι το MiniNExT είχε ένα πρότυπο τρόπο για την χρήση του μαζί με το Quagga μας βοήθησε στην επιλογή του λόγο του ελάχιστου απαιτούμενου χρόνου εκμάθησης του εργαλείου. Επιπλέον, σημαντικό είναι ότι το MiniNExT χρησιμοποιεί εικονικοποίηση που στηρίζεται στις διεργασίες (process-based virtualization) με αποτέλεσμα να έχουμε την δυνατότητα να χρησιμοποιήσουμε πάνω από έναν δρομολογητές χωρίς τον φόβο ότι θα χτυπάνε όλοι στην ίδια θύρα.

Το MiniNExT περιλαμβάνει πιο περίπλοκους κόμβους οι οποίοι χρησιμοποιούνται σε πολλά πειραματικά δίκτυα όπως:

- Μηχανές δρομολόγησης (Quagga και Bird)
- Εξυπηρετητές (BIND και Apache)

- Εξαρτήματα συνδεσιμότητας (OpenVPN, κ.α.)
- NAT και εξαρτήματα διαχείρισης δικτύου (DHCP, κ.α.)

Στην περίπτωσή μας, παρέχει συγκεκριμένο τρόπο για την δημιουργία ενός δρομολογητή, τρέχοντας το Quagga σε ένα ξενιστή και εισάγοντάς του τα απαραίτητα αρχεία ρυθμίσεων (configuration files) τα οποία μπορείτε να βρείτε στο Παράρτημα Κώδικα. Ένα άλλο πλεονέκτημα του MiniNExT είναι ότι η λειτουργία του Quagga ξεκινούσε αυτόματα κατά το χτίσιμο του δικτύου με αποτέλεσμα να γλυτώνουμε χρόνο από το να στήνουμε το Quagga σε έναν ξενιστή κάθε φορά που χτίζουμε το δίκτυο. [22]

5.1.3 Τροποποίηση Pyretic για υποστήριξη IPv6

Η επίσημη έκδοση του Pyretic είναι χτισμένη πάνω στον ελεγκτή POX και υποστηρίζει μόνο το πρωτόκολλο IPv4. Για να χρησιμοποιήσουμε το Pyretic στην εργασία μας και να καταφέρουμε να δημιουργήσουμε πολιτικές που να αντιμετωπίζουν επιθέσεις στα IPv6 δίκτυα, θα πρέπει το Pyretic να υποστηρίζει IPv6. Η λύση του προβλήματος αυτού ήταν αναπόφευκτη ούτως ώστε να υλοποιήσουμε το σύστημα ασφάλειας που επιθυμούμε στα IPv6 δίκτυα.

Οι πιθανές λύσεις που βρήκαμε ήταν δύο. Η πρώτη αφορούσε την διατήρηση του POX ως ελεγκτή του δικτύου, παρότι υποστηρίζει το OpenFlow 1.0, και την χρήση μιας επέκτασης του πρωτοκόλλου OpenFlow που λέγεται Nicira Extensions. Το Nicira Extensions θα βοηθούσε μέσω του Nicira Extended Match να διαχειρίζονται οι μεταγωγείς, ο POX και ο POX πελάτης τα πακέτα IPv6 με την μορφή ενός ευέλικτου από πεδία πακέτου. Η λύση αυτή φαινόταν να είναι μη χρονοβόρα και θεωρητικά να μην μας αναγκάζει να εμβαθύνουμε στον πυρήνα του Pyretic runtime.

Η άλλη επιλογή που είχαμε ήταν η τροποποίηση του συστήματος Pyretic έτσι ώστε να υποστηρίζει OpenFlow 1.3, το οποίο υποστηρίζει IPv6 πακέτα. Στην πράξη, αυτό σήμαινε ότι θα πρέπει να χρησιμοποιήσουμε κάποιον άλλον ελεγκτή όπως ο RYU ελεγκτής ο οποίος υποστηρίζει OpenFlow 1.3, να δημιουργήσουμε έναν RYU πελάτη που θα επικοινωνεί με το Pyretic runtime και να κάνουμε ό,τι τροποποιήσεις θα χρειαζόταν στο πίσω μέρος του Pyretic.

Τελικά, επιλέξαμε την πρώτη λύση, δηλαδή την διατήρηση του POX ως ελεγκτή και την χρήση των Nicira Extensions. Ο κύριος λόγος ήταν χρονικός καθώς η δημιουργία ενός RYU πελάτη απαιτούσε την δημιουργία από το μηδέν ενός προγράμματος με πολλές λειτουργίες καθώς και εξοικείωση με κάθε κομμάτι (component) του Pyretic.

Τα Nicira Extensions είναι επεκτάσεις του πρωτοκόλλου OpenFlow οι οποίες δίνουν κάποιες περαιτέρω δυνατότητες στους μεταγωγείς και στους ελεγκτές που τα υποστηρίζουν

όπως τα επεκτάσιμα μηνύματα Packet-in (Extended PacketIn Messages) και το Nicira Extensible Match.

Στην δική μας περίπτωση θέλουμε οι μεταγωγείς (Open vSwitch) να αντιλαμβάνονται και να διαχειρίζονται τα IPv6 πακέτα, ο POX ελεγκτής να μπορεί να μεταφέρει σωστές πληροφορίες από και προς τους μεταγωγείς και ο POX πελάτης να μπορεί να δέχεται την μορφή των πακέτων αυτών.

Τα Open vSwitch έχουν μια επεκτάσιμη μορφή του packet-in μηνύματος η οποία περιέχει τον λόγο του packet-in μηνύματος (πχ. εάν στάλθηκε εξαιτίας ενός send-to-controller action ή λόγω κάποιου table miss) και στην μέση περίπτωση το match του σχετικού πίνακα εγγραφών. Για την ενεργοποίηση της extended-packet-in λειτουργίας στους μεταγωγείς αρκεί κατά την αρχική επικοινωνία με τον ελεγκτή, στον ConnectionUp χειριστή (handler), να σταλεί ένα nx_packet_in_format μήνυμα:

```
event.connection.send(nx.nx_packet_in_format())
```

Στην πλευρά του POX controller πρέπει να επαναπρογραμματίσουμε το προεπιλεγμένο OpenFlow packet-in γεγονός και στην θέση του να ορίσουμε το Nicira packet-in. Για να το κάνουμε αυτό πρέπει να προσθέσουμε στην αρχική κλήση του POX ελεγκτή το επιχείρημα (argument) --convert-packet-in από το openflow.nicira εξάρτημα (component). Συγκεκριμένα στο pyretic, επειδή ο POX καλείται κατά την εκκίνηση του pyretic.py, δεν μπορούμε να βάλουμε στην γραμμή εντολών το παραπάνω argument και για τον λόγο αυτό θα το προσθέσουμε στο αντίστοιχο μέρος που καλείται το pox.py μέσα στο pyretic.py.

Τέλος, πρέπει να αλλάξουμε τον τρόπο που αντιλαμβάνεται τα στοιχεία ενός IPv6 πακέτου ο POX πελάτης. Αυτό σημαίνει ότι θα πρέπει να ορίσουμε ένα extended match με τα κατάλληλα IPv6 πεδία. Το Nicira Extended Match (NXM), η βάση του OpenFlow Extensible Match (OXM) μας επιτρέπει ακριβώς αυτή την λειτουργία. Ο πυρήνας του NXM είναι η nx_match δομή η οποία αντικαθιστά την αυθεντική ofp_match δομή ως τρόπο καθορισμού των matches στους πίνακες εγγραφών. Αντίθετα με το ofp_match, το οποίο είναι μια σταθερή συλλογή από πεδία, το nx_match είναι ένα ευέλικτο κιβώτιο για ατομικά nxm_entrys. Στον POX, η βασική διεπαφή είναι ίδια με αυτή μια Python λίστας. Έτσι, για να μπορούν να γίνονται match τα IPv6 πακέτα αρκεί να ελέγχουμε το ethernet type του πακέτου. Αν αυτό είναι 0x86dd (ethernet type για τα IPv6), τότε να ορίζονται όλα τα πεδία που αντιστοιχούν σε ένα IPv6 πακέτο. [23]

5.2 Σύστημα υλοποίησης ασφάλειας με Pyretic Policy Functions

Τα είδη των επιθέσεων που μπορούν να γίνουν στα IPv4 δίκτυα είναι πολλά και το ίδιο ισχύει και για τα δίκτυα IPv6. Η μόνη διαφορά είναι ότι στα IPv4 δίκτυα έχουν βρεθεί τρόποι αντιμετώπισης των περισσοτέρων επιθέσεων, εν αντιθέσει με τα IPv6 δίκτυα, τα οποία είναι ακόμη πολύ ευάλωτα. Για τον λόγο αυτό χρειαζόμαστε ένα σύστημα ασφάλειας με το οποίο να μπορούμε εύκολα να εντοπίζουμε και να αντιμετωπίζουμε νέες επιθέσεις στα IPv6 δίκτυα. Το Pyretic είναι ένα σύστημα και μια γλώσσα προγραμματισμού η οποία απλοποιεί τον τρόπο με τον οποίο μπορεί ένας προγραμματιστής να δημιουργήσει πολιτικές, χρησιμοποιώντας τες μάλιστα ως συναρτήσεις.

Στην επίσημη έκδοσή του το Pyretic λειτουργεί πάνω από τον ελεγκτή POX και ικανοποιεί μόνο IPv4 δίκτυα. Για τον λόγο αυτό θα χρειαστεί να του κάνουμε τις απαραίτητες τροποποιήσεις ούτως ώστε να είναι λειτουργικό και σε IPv6 δίκτυα. Πέρα από την βασική του λειτουργικότητα θα ασχοληθούμε και με τον εμπλουτισμό της γλώσσας του Pyretic ώστε να καλύπτει τις νέες απειλές που προκύπτουν στα δίκτυα IPv6.

Στον τομέα της λειτουργικότητας, το Pyretic ήθελε αρκετές τροποποιήσεις ούτως ώστε να λειτουργεί σε ένα IPv6 δίκτυο. Αρχικά, χρειάστηκε η προσθήκη του Nicira Extension που αναφέραμε στο κεφάλαιο 5.1.3 για την διαχείριση μηνυμάτων IPv6 στο επίπεδο του μεταγωγέα αλλά και μέχρι τον POX πελάτη. Επιπλέον, χρειάστηκαν αλλαγές στην διαχείριση των flows ώστε να ακολουθούν το μοντέλο του Nicira Extension και στις συναρτήσεις της Python οι οποίες διαχειρίζονταν μόνο τις IPv4 διευθύνσεις. Τέλος, διορθώσαμε διάφορα σφάλματα που υπήρχαν στο Pyretic ή τις βιβλιοθήκες που χρησιμοποιούσε και δεν είχαν γίνει φανερά λόγω της μη χρήσης του με IPv6. Περισσότερες λεπτομέρειες για τις αλλαγές αυτές μπορείτε να βρείτε στο Παράρτημα Κώδικα της εργασίας.

Έχοντας ετοιμάσει το Pyretic να λειτουργεί κανονικά σε ένα IPv6 δίκτυο, είχαμε προβλήματα στην δημιουργία πολιτικών για την εντοπισμό και την αντιμετώπιση των επιθέσεων σε δίκτυα IPv6. Ο κύριος λόγος ήταν ότι δεν μπορούσαμε με τα υπάρχοντα πεδία των πακέτων του OpenFlow να φτιάξουμε πολιτικές που να σχετίζονται με τον κάθε τύπο του IPv6 μηνύματος και ιδιαίτερα τα μηνύματα του Neighbor Discovery Protocol, το οποίο αποτελεί πολύ σημαντικό μέρος της ασφάλειας ενός δικτύου. Για τον λόγο αυτό δημιουργήσαμε δύο δικά μας πεδία σε κάθε πακέτο IPv6, αναλύοντας ουσιαστικά δεδομένα που υπήρχαν έτσι κι αλλιώς στα πακέτα ως ακατέργαστα δεδομένα (raw data).

Το nxt είναι το πρώτο από τα δύο πεδία που δημιουργήσαμε. Αντιστοιχεί, στο πεδίο “Next Header” της επικεφαλίδας του IPv6 πακέτου και μας δείχνει τον τύπο της επόμενης επικεφαλίδας. Πρόκειται για το έκτο byte της IP επικεφαλίδας και οι τιμές του είναι αντίστοιχες με του πεδίου protocol στο IPv4, το οποίο και αντικατέστησε. Το nxt μας ενδιαφέρει για να μπορούμε να διαχειριστούμε τα πακέτα που περνάνε με βάση τον τύπο

τους. Είναι σημαντικό να γνωρίζουμε εάν ένα πακέτο είναι τύπου ICMPv6, TCP ή UDP ή κάποιο άλλο πρωτόκολλο από το επίπεδο μεταφορά.

```
location:       1       2       None
source:         fc00:0:0:1::1   00:00:00:00:00:01
dest:           ff02::1:ff00:11 33:33:ff:00:00:11
md5:            935fc499de09262e516426345dee1271
ethtype:        34525
header_len:     14
icmpv6_type:    135
nxt:            58
payload_len:    86
```

Εικόνα 5.2: Τα πεδία ενός τροποποιημένου πακέτου Pyretic

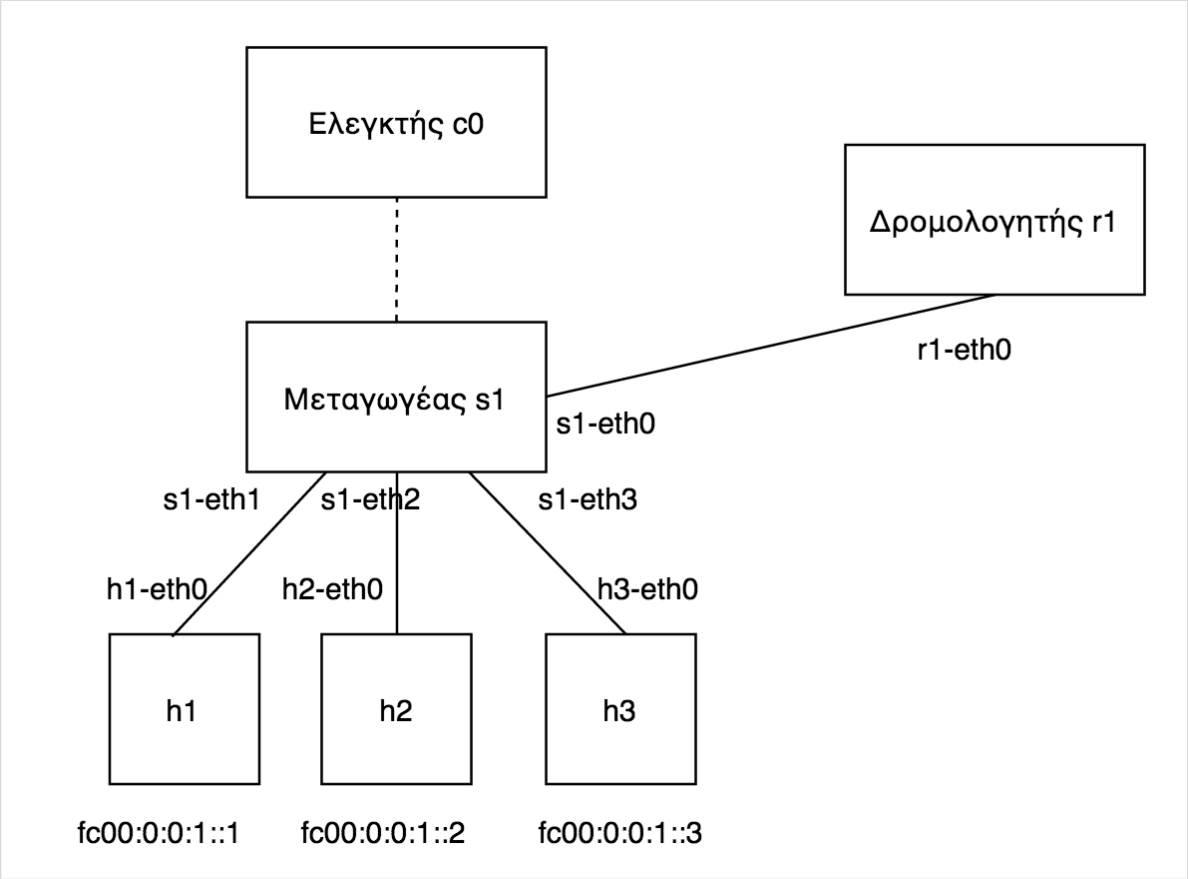
Το icmpv6_type είναι το δεύτερο πεδίο που δημιουργήσαμε. Όπως δηλώνει και το όνομά του αντιστοιχεί στα bits τα οποία δείχνουν τον τύπο ενός icmpv6 μηνύματος, όταν φυσικά αυτό είναι τύπου ICMPv6. Το Neighbor Discovery Protocol είναι από τα σημαντικότερα πρωτόκολλα στα πλαίσια της ασφάλειας ενός δικτύου και τα μηνύματά του είναι τύπου ICMPv6. Επομένως, είναι πολύ σημαντικό να γνωρίζουμε τον τύπο των ICMPv6 μηνυμάτων ώστε να γνωρίζουμε εάν κάποιος επιτιθέμενος προσπαθεί είτε να σκανάρει το δίκτυο είτε να στείλει ψεύτικα μηνύματα για να εξαπατήσει κάποιους ξενιστές.

5.3 Αντιμετώπιση σκαναρίσματος δικτύου από επιτιθέμενο εντός δικτύου

Έχοντας το Pyretic έτοιμο να διαχειριστεί IPv6 μηνύματα, θα δημιουργήσουμε μια Pyretic εφαρμογή η οποία θα εντοπίζει όταν ένας ξενιστής εντός δικτύου προσπαθεί να σκανάρει το δίκτυο στο οποίο βρίσκεται με σκοπό να γνωρίζει ποιες είναι οι ενεργές IP διευθύνσεις. Η εφαρμογή αυτή αποτελεί μια βάση για την δημιουργία Pyretic εφαρμογών οι οποίες αντιμετωπίζουν άλλα είδη επιθέσεων σε IPv6 δίκτυα.

5.3.1 Τοπολογία

Θεωρούμε ότι ένα δίκτυο μπορεί στην απλή μορφή του να αναπαρασταθεί με ένα δρομολογητή, ένα μεταγωγέα και τρεις ξενιστές όπως φαίνεται στην παρακάτω εικόνα.



Εικόνα 5.3: Η τοπολογία για την προσομοίωση του επιτιθέμενου εντός δικτύου

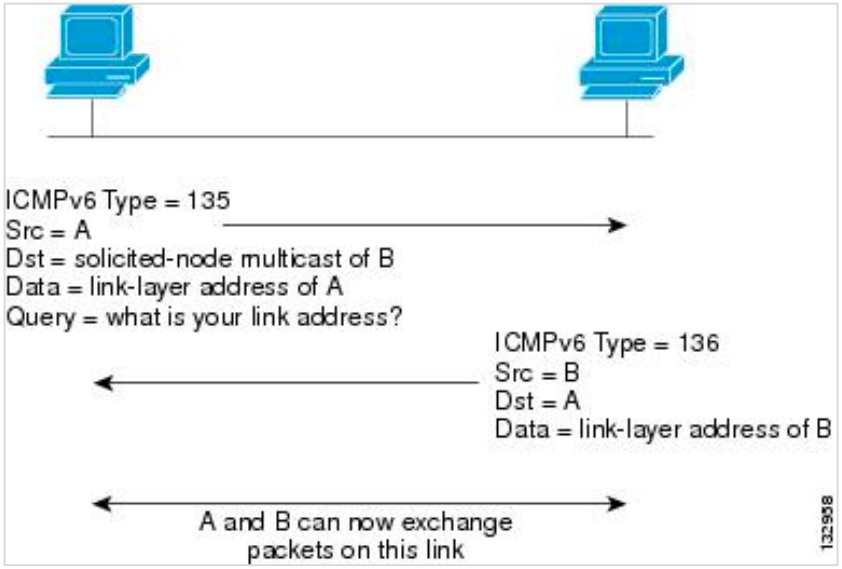
Στην τοπολογία αυτή οι ξενιστές είναι συνδεδεμένοι με τον μεταγωγέα ο οποίος είναι απαραίτητος διαμεσολαβητής, είτε αυτοί θέλουν να επικοινωνήσουν μεταξύ τους, είτε με το υπόλοιπο διαδίκτυο καθώς η προεπιλεγμένη πύλη που τους συνδέει με αυτό είναι η εσωτερική διεπαφή του δρομολογητή (r1-eth0). Ο μεταγωγέας ακολουθεί τους κανόνες που δέχεται από τον POX ελεγκτή, ο οποίος με την σειρά του μεταφέρει τους κανόνες που στέλνει η Pyretic εφαρμογή μας. Αντίστοιχα, ο POX ελεγκτής ενημερώνει το Pyretic σχετικά με τα γεγονότα που συμβαίνουν στον μεταγωγέα.

Οι εξυπηρετητές και η εσωτερική διεπαφή του δρομολογητή έχουν δύο διευθύνσεις IPv6, μια τοπική και μια παγκόσμια. Όταν ένα μήνυμα στέλνεται εντός του δικτύου τότε στέλνεται έχοντας ως διεύθυνση αποστολέα την τοπική διεύθυνση του ξενιστή (πχ. fe80::200:ff:fe00:1/64). Αντίστοιχα, όταν στέλνεται ένα πακέτο εκτός δικτύου δεν χρησιμοποιείται η τοπική διεύθυνση IPv6 αλλά μόνο η παγκόσμια, δηλαδή η τοπική διεύθυνση δεν εμφανίζεται ποτέ εκτός δικτύου.

5.3.2 Εντοπισμός επιτιθέμενου

Για να εντοπίσουμε τον επιτιθέμενο θα πρέπει αρχικά να κατανοήσουμε τι διαφορετικό έχει μια συνηθισμένη συμπεριφορά ξενιστή εντός ενός δικτύου με αυτή του ενός ξενιστή ο οποίος σκανάρει το δίκτυο.

Όταν ένας ξενιστής λειτουργεί φυσιολογικά μέσα στο δίκτυο τότε θα ανταλλάσσει πακέτα με άλλους ξενιστές εντός και εκτός δικτύου. Στην παρούσα τοπολογία δεν μας ενδιαφέρει η κίνηση εκτός δικτύου επομένως θα επικεντρωθούμε στην κίνηση εντός του δικτύου. Αυτό που μας ενδιαφέρει να εξετάσουμε είναι το ξεκίνημα της επικοινωνίας του ξενιστή h1 με τον ξενιστή h2. Έτσι, όταν ο h1 θελήσει να επικοινωνήσει για πρώτη φορά με τον ξενιστή h2, γνωρίζοντας μόνο την διεύθυνση IPv6 του (fc00:0:0:1::2), θα του στείλει αρχικά ένα ICMPv6 μήνυμα Neighbor Solicitation για να μάθει την MAC διεύθυνσή του. Συγκεκριμένα, επειδή στο IPv6 δεν υπάρχει broadcast, θα στείλει το μήνυμα Neighbor Solicitation στην Solicited-Node multicast διεύθυνση, δηλαδή την ff02::1:ff00:2/64 όπως περιγράφεται στο κεφάλαιο 2.4.2. Ο h2, με την σειρά του, θα απαντήσει με ένα ICMPv6 μήνυμα Neighbor Advertisement για να ενημερώσει τον h1 ποια είναι η MAC διεύθυνσή του. Αυτή ήταν η διαδικασία του Neighbor Discovery Protocol για την γνωριμία των δυο ξενιστών και έπειτα αυτής μπορούν να επικοινωνούν κανονικά μεταξύ τους.



Εικόνα 5.4: Γνωριμία δύο ξενιστών μέσω του Neighbor Discovery Protocol[http://www.cisco.com/c/en/us/td/docs/security/asa/asa84/configuration85/guide/asa_cfg_cli_85/route_ipv6_neighbor.html]

Στην περίπτωση του ξενιστή ο οποίος θέλει να σκανάρει το δίκτυο στο οποίο βρίσκεται, για να γνωρίζει ποιες είναι οι ενεργές διευθύνσεις IPv6, υπάρχουν δύο περιπτώσεις. Η πρώτη περίπτωση είναι να στείλει ένα ICMPv6 πακέτο στην διεύθυνση πολλαπλής διανομής ff02::1, η οποία αντιπροσωπεύει όλους τους εγγεγραμμένους ξενιστές στο υπάρχον δίκτυο και άρα από το ποιοι θα απαντήσουν θα μπορέσει να καταλάβει τις

ενεργές διευθύνσεις IPv6 του δικτύου. Η δεύτερη περίπτωση είναι να στείλει σε όλες τις δυνατές διευθύνσεις του δικτύου ένα ICMPv6 μήνυμα Neighbor Solicitation και ανάλογα με το ποιες διευθύνσεις θα απαντήσουν να γνωρίζει τις ενεργές διευθύνσεις IPv6 του δικτύου.

Για τον εντοπισμό του επιτιθέμενου θα συγκρίνουμε τις τρεις καταστάσεις, αυτή του φυσιολογικού χρήστη με αυτές του επιτιθέμενου. Παρατηρούμε ότι οι διαφορές τους είναι ότι ο επιτιθέμενος, είτε θα στείλει ICMPv6 στην διεύθυνση ff02::1 είτε θα στείλει πολλά μηνύματα Neighbor Solicitation σε ανύπαρκτες διευθύνσεις IPv6. Επομένως, για να τον εντοπίσουμε αρκεί να ελέγχουμε όσα μηνύματα έχουν διεύθυνση προορισμού την ff02::1 και να γνωρίζουμε ποιες διευθύνσεις στο δίκτυό μας είναι ενεργές ώστε όταν ένας ξενιστής στέλνει μηνύματα Neighbor Solicitation σε αρκετές ανενεργές διευθύνσεις να γνωρίζουμε ότι σκανάρει το δίκτυό μας.

5.3.3 Προβλήματα

Κάθε περίπτωση έχει τα δικά της προβλήματα τα οποία καλούμαστε να λύσουμε. Αρχικά, στην περίπτωση της διεύθυνσης πολλαπλής διανομής ff02::1, εάν εντοπίσουμε τον επιτιθέμενο με βάση την διεύθυνση, αυτό σημαίνει ότι θα έχει ήδη μάθει τις ενεργές διευθύνσεις του δικτύου αφότου καταλάβουμε ποιος είναι, επομένως δεν τον αποτρέπουν από το να πάρει την πληροφορία που θέλει. Στην δεύτερη περίπτωση, για τον εντοπισμό του επιτιθέμενου θα πρέπει να γνωρίζουμε ποιες είναι οι ενεργές διευθύνσεις του δικτύου, ώστε να γνωρίζουμε αν προσπαθεί να επικοινωνήσει με άλλους ξενιστές ή σκανάρει το δίκτυο και χρειαζόμαστε έναν αυτοματοποιημένο τρόπο να βρίσκουμε την διεύθυνση του παραλήπτη στα Neighbor Solicitation μηνύματα καθώς στην διεύθυνση παραλήπτη θα υπάρχει η Solicited-Node multicast διεύθυνση.

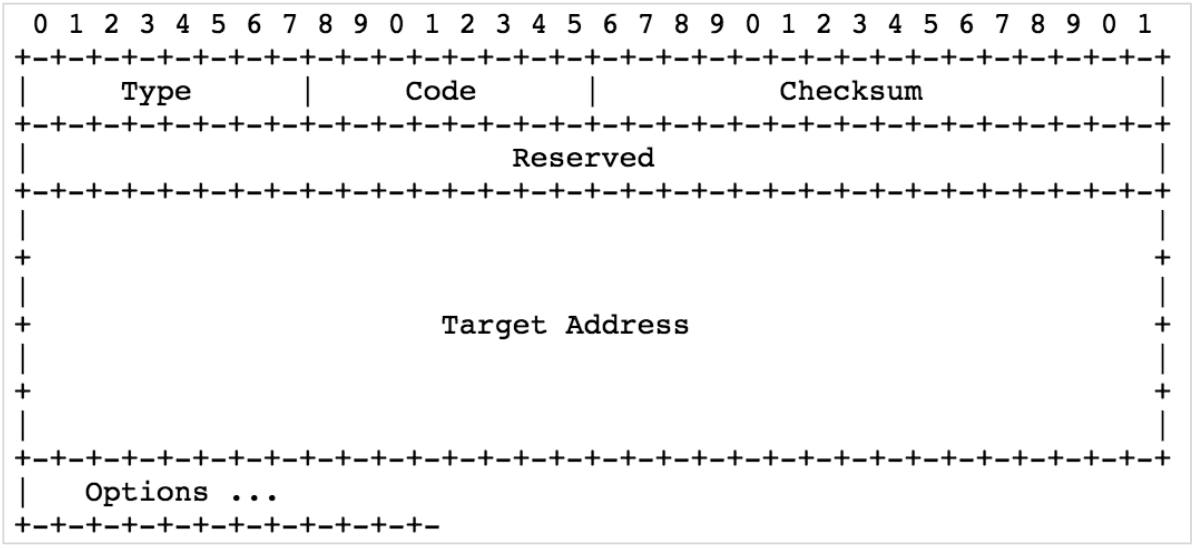
5.3.4 Λύσεις

Η περίπτωση της πολλαπλής διανομής ff02::1 έχει μια πολύ άμεση λύση, η οποία δεν επιτρέπει σε καμία διεύθυνση IPv6, εκτός από κάποιες διαχειριστικές, να στέλνουν στην διεύθυνση ff02::1.

Η δεύτερη περίπτωση έχει δύο ξεχωριστά προβλήματα. Το πρώτο βασικό πρόβλημα είναι το πώς θα υλοποιήσουμε ένα ρεαλιστικό δίκτυο όπου θα γνωρίζουμε τις ενεργές διευθύνσεις IPv6. Στην περίπτωση αυτή, κάνουμε την παραδοχή πως μία διεύθυνση IPv6 θεωρείται ενεργή την πρώτη φορά που ο ξενιστής της θα στείλει ένα μήνυμα προς οποιονδήποτε άλλο ξενιστή και γίνεται ανενεργή είτε όταν ο ξενιστής αποσυνδέεται από το

δίκτυο είτε όταν δεν έχει στείλει κάποιο πακέτο, δηλαδή είναι μη ενεργή, για περισσότερα λεπτά απ' όσο διαρκεί η εγγραφή της στις ροές του μεταγωγέα.

Το πρόβλημα της εύρεσης της διεύθυνσης του παραλήπτη την λύσαμε αναλύοντας το πακέτο ICMPv6 Neighbor Solicitation και τραβώντας από αυτό τα κατάλληλα bytes του φορτίου (payload) τα οποία περιέχουν την διεύθυνση IPv6 της οποία στοχεύει να μάθει την MAC (target address).



Εικόνα 5.5: Η διάταξη του ICMPv6 πακέτου Neighbor Solicitation [https://tools.ietf.org/html/rfc4861#section-4.3]

5.3.5 Υλοποίηση

Για τον εντοπισμό και την αντιμετώπιση του σκαναρίσματος δικτύου από τον επιτιθέμενο, ο οποίος βρίσκεται εντός του δικτύου, φτιάξαμε την Pyretic εφαρμογή internal.py.

Το internal.py βασίζεται στην πολιτική ερωτήσεων (query policy) του Pyretic, η οποία μας δίνει την δυνατότητα να παρακολουθούμε το δίκτυο και να καθορίζουμε την ενεργοποίηση συναρτήσεων με βάση τα χαρακτηριστικά των πακέτων τα οποία περνάνε, χωρίς να επηρεάζουμε την ροή των πακέτων αυτών. Στην περίπτωσή μας, όλα τα πακέτα στέλνονται κανονικά εκτός από όταν σταλθεί ένα πακέτο Neighbor Solicitation όπου τότε επιπρόσθετα ενεργοποιείται η συνάρτηση check_destination. Αντίστοιχα, όταν σταλθεί ένα μήνυμα από νέα MAC διεύθυνση αποστολέα ενεργοποιείται η συνάρτηση mac_learner.

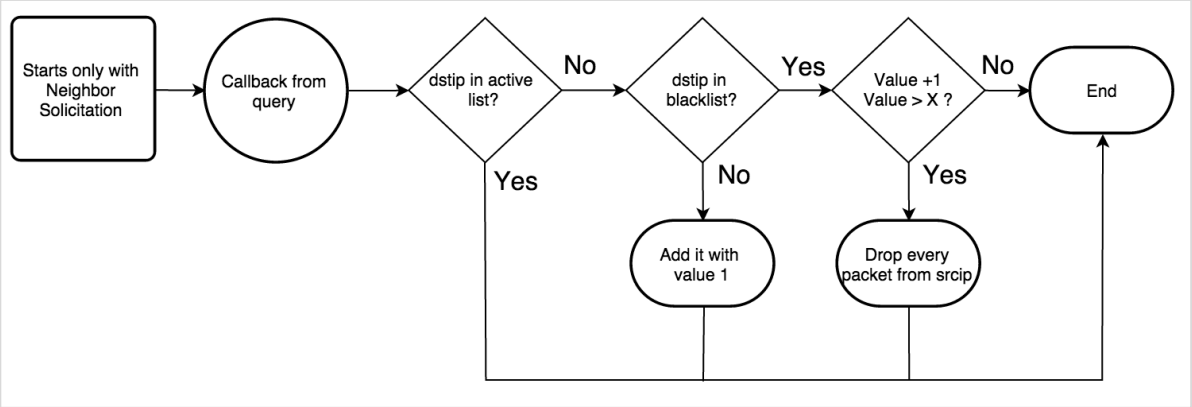
Η συνάρτηση check_destination ελέγχει στο πακέτο Neighbor Solicitation που στάλθηκε εάν η διεύθυνση IP του παραλήπτη είναι ενεργή, δηλαδή εάν υπάρχει στο λεξικό

active. Εάν όχι σημαίνει ότι ο αποστολέας ψάχνει για κάποια διεύθυνση η οποία δεν υπάρχει και για τον λόγο αυτό βάζουμε την διεύθυνση IP του σε ένα Python λεξικό (dictionary) με την τιμή 1. Σε περίπτωση που βρίσκεται ήδη στο λεξικό τότε ανεβάζουμε την αξία (value) που αντιστοιχεί στην διεύθυνση IP του κατά ένα. Τέλος, όταν η τιμή που αντιστοιχεί στην διεύθυνση IP υπερβεί τον αριθμό X τον οποίο ορίζουμε εμείς, τότε αυτόματα ανανεώνεται η πολιτική που ακολουθεί ο μεταγωγέας και πλέον δεν επιτρέπει στην διεύθυνση του αποστολέα να στείλει τίποτα, κάνοντας drop όλα τα πακέτα από την διεύθυνση αυτή.

Η συνάρτηση mac_learner είναι υπεύθυνη για να προσθέσει τις κατάλληλες ροές έτσι ώστε όταν στέλνεται ένα πακέτο σε μια IP διεύθυνση να μην γίνεται flood που είναι ο προεπιλεγμένος τρόπος αποστολής αλλά να στέλνεται μόνο στην διεύθυνση που πρέπει. Έτσι, όταν μια νέα MAC διεύθυνση στέλνει ένα πακέτο τότε αυτόματα η συνάρτηση αυτή ενημερώνει τους κανόνες ούτως ώστε όταν έρθει πακέτο προς την διεύθυνση αυτή να σταλεί στον αντίστοιχο ξενιστή. Επιπλέον, στην συνάρτηση αυτή έχουμε προσθέσει την εισαγωγή των διευθύνσεων στο λεξικό active, οπότε όταν μια νέα διεύθυνση στέλνει ένα μήνυμα αυτόματα την καταχωρούμε στο λεξικό αυτό μαζί με τον μεταγωγέα και την θύρα στην οποία αντιστοιχεί.

Στην εφαρμογή έχουμε και ένα τρίτο query το οποίο μας βοηθάει στο να προσομοιώσουμε έναν ξενιστή ο οποίος είναι ανενεργός και παύει να ανήκει στο δίκτυο καθώς δεν υπάρχει πλέον η εγγραφή του στη neighbor discovery cache του δρομολογητή. Ουσιαστικά, ελέγχει πόσα πακέτα έχει στείλει ο κάθε ξενιστής κάθε 15 λεπτά και όταν βρεθεί ότι κάποιος ξενιστής δεν έχει στείλει κανένα πακέτο τα τελευταία 15 λεπτά τότε τον αφαιρεί από το λεξικό active.

Τέλος, η ενημέρωση του συστήματος και του λεξικού active για το γεγονός ότι κάποιος ξενιστής έφυγε χρειάζεται και αυτή να υλοποιηθεί μέσω της Pyretic εφαρμογής μας. Η υλοποίηση αυτή θα ήταν εξαιρετικά δύσκολη εάν δεν υπήρχαν τα Network Objects του Pyretic Runtime συστήματος. Το Pyretic για την λειτουργία ελέγχει κάθε 0.25 δευτερόλεπτα εάν η τοπολογία του δικτύου έχει αλλάξει έτσι ώστε να μπορεί αυτόματα να μετατρέψει κατάλληλα τους κανόνες των μεταγωγέων και να εξακολουθούν να λειτουργούν οι πολιτικές που έχει προδιαγράψει ο προγραμματιστής. Μέσω της λειτουργίας του το Pyretic ενημερώνεται για τα γεγονότα που επηρεάζουν την τροποποίηση του δικτύου όπως είναι η συνάρτηση port_down το οποίο ενεργοποιείται όταν κάποιος ξενιστής φεύγει από το δίκτυο. Έτσι, εμείς από την συνάρτηση αυτή τραβάμε την πληροφορία που χρειαζόμαστε, δηλαδή την διεπαφή του μεταγωγέα από την οποία αποσυνδέθηκε κάποιος ξενιστής και αφαιρούμε την αντίστοιχη IP διεύθυνση από το λεξικό active.



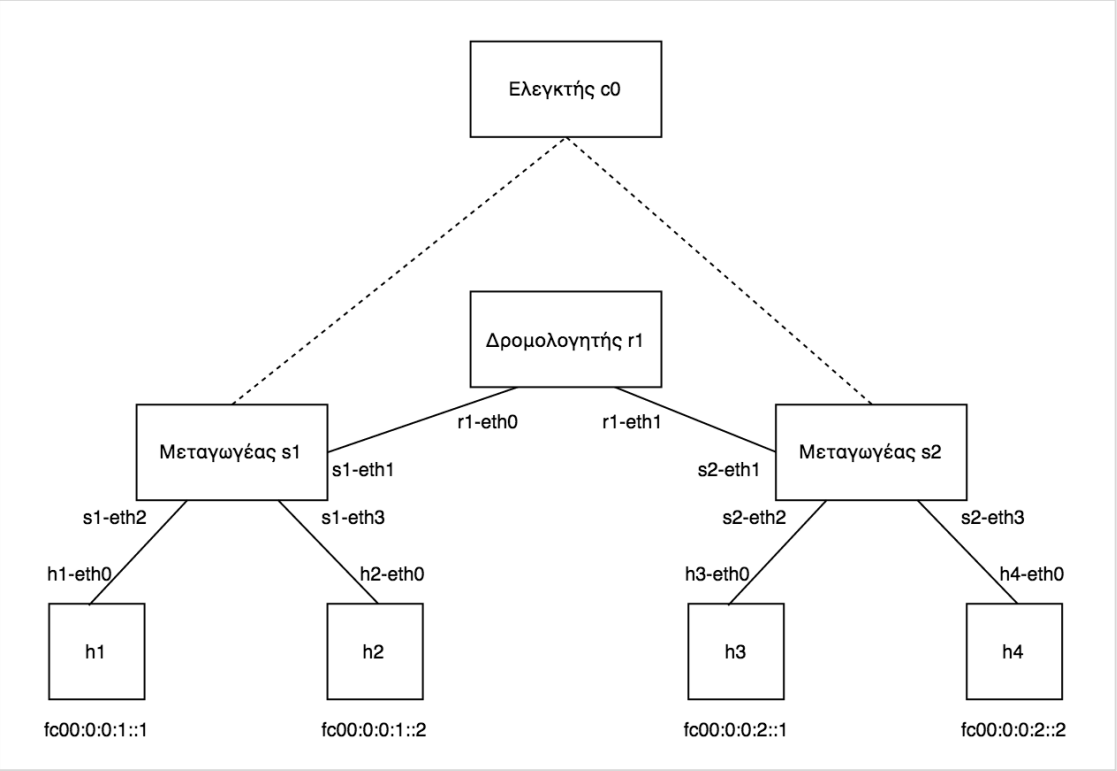
Εικόνα 5.6: Το διάγραμμα ροής μετά το callback

5.4 Αντιμετώπιση σκαναρίσματος δικτύου από επιτιθέμενο σε άλλο ελεγχόμενο δίκτυο

Στο υποκεφάλαιο αυτό θα εξετάσουμε την περίπτωση που ο επιτιθέμενος βρίσκεται σε διαφορετικό δίκτυο από αυτό που σκανάρει αλλά και τα δυο δίκτυα βρίσκονται μέσα στο ίδιο ευρύτερο δίκτυο. Στην πραγματικότητα αυτό μπορεί να αντικατοπτρίζει ένα πανεπιστήμιο το οποίο έχει ένα μεγάλο δίκτυο με μια προεπιλεγμένη πύλη με την οποία συνδέεται με το υπόλοιπο διαδίκτυο όμως εσωτερικά απαρτίζεται από πολλά μικρότερα δίκτυα.

5.4.1 Τοπολογία

Θεωρούμε ότι ένα σύστημα με δυο δίκτυα τα οποία ανήκουν σε ένα κοινό ευρύτερο δίκτυο μπορεί στην απλή μορφή να αναπαρασταθεί με ένα δρομολογητή, δύο μεταγωγείς και 2 ξενιστές σε κάθε μεταγωγέα όπως φαίνεται στην παρακάτω εικόνα.



Εικόνα 5.7: Η τοπολογία του δικτύου για την προσομοίωση του επιτιθέμενου εντός άλλου ελεγχόμενου δικτύου

Στην τοπολογία αυτή έχουμε δυο ξεχωριστά δίκτυα, ένα σε κάθε διεπαφή του δρομολογητή r1. Στην διεπαφή r1-eth0 συνδέεται το δίκτυο fc00:0:0:1::/64 ενώ στην διεπαφή r1-eth1 συνδέεται το δίκτυο fc00:0:0:2::/64. Σε κάθε δίκτυο οι ξενιστές είναι συνδεδεμένοι με ένα μεταγωγέα ο οποίος είναι απαραίτητος διαμεσολαβητής είτε αυτοί θέλουν να επικοινωνήσουν μεταξύ τους , είτε με το υπόλοιπο διαδίκτυο. Οι διεπαφές του δρομολογητή αποτελούν την προεπιλεγμένη πύλη για την επικοινωνία του κάθε δικτύου με το υπόλοιπο διαδίκτυο.

Οι μεταγωγείς ακολουθούν τους κανόνες που δέχονται από τον POX ελεγκτή, ο οποίος με την σειρά του μεταφέρει τους κανόνες που στέλνει η Pyretic εφαρμογή μας. Αντίστοιχα, ο POX ενημερώνει το Pyretic σχετικά με τα γεγονότα που συμβαίνουν στον μεταγωγέα.

Οι ξενιστές και οι διεπαφές του δρομολογητή έχουν από δύο διευθύνσεις IPv6, μια τοπική και μια παγκόσμια. Όταν ένα μήνυμα στέλνεται εντός του δικτύου τότε στέλνεται έχοντας ως διεύθυνση αποστολέα την τοπική διεύθυνση του ξενιστή (πχ. fe80::200:ff:fe00:3/64). Αντίστοιχα, όταν στέλνεται ένα πακέτο εκτός του δικτύου τότε ποτέ δεν χρησιμοποιείται η τοπική διεύθυνση IPv6 αλλά μόνο η παγκόσμια.

5.4.2 Εντοπισμός επιτιθέμενου

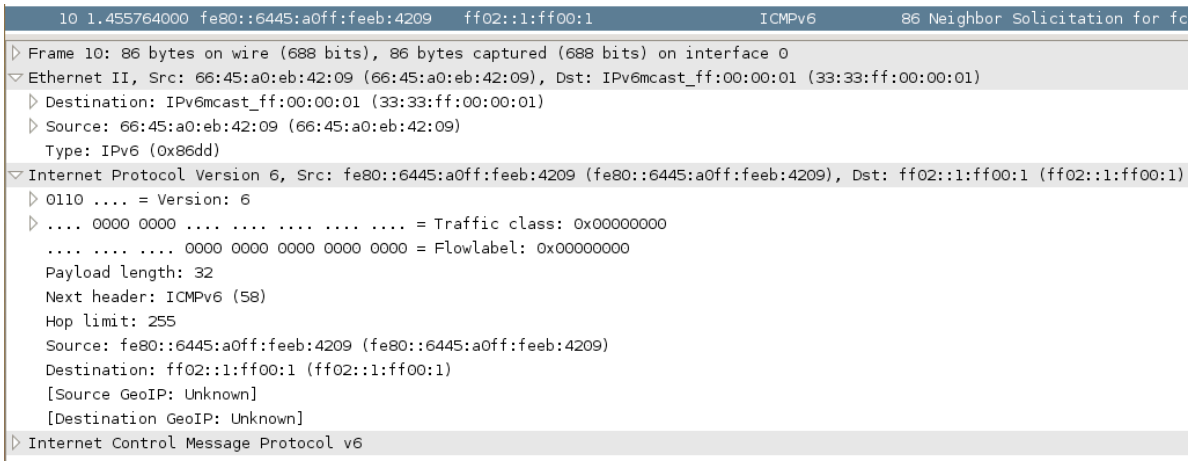
Για να εντοπίσουμε τον επιτιθέμενο θα πρέπει να κατανοήσουμε τις διαφορές στην συμπεριφορά εντός του δικτύου μας από τα πακέτα ενός ξενιστή που λειτουργεί φυσιολογικά και ενός ξενιστή ο οποίος σκανάρει το δίκτυό μας.

Αντίστοιχα με την περιγραφή που έγινε στο 5.3.2, θα ασχοληθούμε με την επικοινωνία μεταξύ δύο ξενιστών που βρίσκονται σε δυο διαφορετικά δίκτυα. Στην περίπτωσή μας θα εξετάσουμε την επικοινωνία μεταξύ του ξενιστή h1 και του ξενιστή h3. Όταν ο ξενιστής h3 θέλει να επικοινωνήσει για πρώτη φορά με τον ξενιστή h1 τότε θα στείλει το πακέτο που θέλει να στείλει στην προεπιλεγμένη πύλη με IP διεύθυνση παραλήπτη την διεύθυνση του h1 και MAC διεύθυνση παραλήπτη αυτή της διεπαφής r1-eth1 του δρομολογητή όπως φαίνεται στην εικόνα 5.8.

```
    9 1.455717000 fc00:0:2::1          fc00:0:1::1          ICMPv6      118 Echo (ping) request
▷ Frame 9: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface 1
▽ Ethernet II, Src: 00:00:00_00:00:03 (00:00:00:00:00:03), Dst: 46:c6:ae:8f:aa:e5 (46:c6:ae:8f:aa:e5)
  ▷ Destination: 46:c6:ae:8f:aa:e5 (46:c6:ae:8f:aa:e5)
  ▷ Source: 00:00:00_00:00:03 (00:00:00:00:00:03)
    Type: IPv6 (0x86dd)
▽ Internet Protocol Version 6, Src: fc00:0:2::1 (fc00:0:2::1), Dst: fc00:0:1::1 (fc00:0:1::1)
  ▷ 0110 .... = Version: 6
  ▷ .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 64
    Next header: ICMPv6 (58)
    Hop limit: 64
    Source: fc00:0:2::1 (fc00:0:2::1)
    Destination: fc00:0:1::1 (fc00:0:1::1)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▷ Internet Control Message Protocol v6
```

Εικόνα 5.8: Το πακέτο (ping6) που στέλνει ο h3 στον h1 όπως φαίνεται στον μεταγωγέα s2

Στην συνέχεια, ο δρομολογητής εάν έχει το ζευγάρι MAC, IP του h1 ως εγγραφή του στον πίνακα ανακάλυψης γείτονα (neighbor discovery table) του, τότε του στέλνει απευθείας το πακέτο του h3 με διεύθυνση. Εάν δεν το έχει στο neighbor discovery table τότε χρειάζεται να στείλει ICMPv6 μήνυμα Neighbor Solicitation για να μάθει την MAC διεύθυνσή του. Συγκεκριμένα, στέλνει το μήνυμα Neighbor Solicitation στην διεύθυνση πολλαπλής διανομής Solicited-Node, δηλαδή την ff02::1:ff00:1 όπως περιγράφεται στο κεφάλαιο 2.4.2. Ο h1 με την σειρά του απαντάει στον δρομολογητή με ένα Neighbor advertisement και ο δρομολογητής του προωθεί το πακέτο του h3. Τέλος, ο h1 απαντάει στο πακέτο του h3 και πλέον μπορούν να επικοινωνούν μεταξύ τους.



Εικόνα 5.9: Το NS πακέτο που στέλνεται από το interface r1-eth0 του δρομολογητή στον h1 όπως φαίνεται στον μεταγωγέα s1

Στην περίπτωση που ο h3 έχει ως στόχο να σκανάρει το δίκτυο fc00:0:0:1::/64, θεωρούμε ως δεδομένο ότι προσπαθεί να επικοινωνήσει και με διευθύνσεις οι οποίες δεν αντιστοιχούν σε κάποιον ξενιστή. Αυτό σημαίνει ότι ο h3 στέλνει στον δρομολογητή αιτήματα για IP διευθύνσεις που δεν υπάρχουν. Ο δρομολογητής με την σειρά του προσπαθεί να βρει τις MAC διευθύνσεις που αντιστοιχούν στις IP διευθύνσεις – καθώς δεν γίνεται να έχει στο neighbor discovery table εγγραφές για IP διευθύνσεις που δεν υπάρχουν – και στέλνει μηνύματα Neighbor Solicitation στις διευθύνσεις πολλαπλής διανομής Solicited-Node που αντιστοιχούν στην κάθε διεύθυνση. Για τα μηνύματα αυτά όπως είναι φυσικό δεν λαμβάνει κάποια απάντηση.

Παρατηρώντας τις δύο καταστάσεις βλέπουμε ότι η αποστολή πολλών μηνυμάτων Neighbor Solicitation σε IP διευθύνσεις οι οποίες δεν υπάρχουν – δεν είναι ενεργές - είναι το χαρακτηριστικό γνώρισμα της κατάστασης που ο επιτιθέμενος σκανάρει. Επομένως, για να εντοπίσουμε ότι γίνεται σκανάρισμα του δικτύου πρέπει να γνωρίζουμε ποιες διευθύνσεις στο δίκτυό μας είναι ενεργές ώστε όταν εμφανίζονται πολλά μηνύματα Neighbor Solicitation σε αρκετές ανενεργές IP διευθύνσεις να γνωρίζουμε γίνεται σκανάρισμα του δικτύου. Για τον εντοπισμό του επιτιθέμενου βλέπουμε ότι το Neighbor Discovery πρωτόκολλο λειτουργεί σε τοπικό επίπεδο και επομένως δεν μπορούμε μέσω αυτού να βρούμε τον επιτιθέμενο καθώς αυτός βρίσκεται σε άλλο δίκτυο. Για τον λόγο αυτό θα χρειαστεί να συνδυάσουμε τα δεδομένα και των δυο μεταγωγέων, s1 και s2. Από τον s1 να αντιλαμβανόμαστε πότε γίνεται σκανάρισμα δικτύου και από τον s2 να βρίσκουμε τον ξενιστή που το προκάλεσε.

5.4.3 Προβλήματα

Όπως και στην προηγούμενη περίπτωση (ενότητα 5.3.3) για τον εντοπισμό του επιτιθέμενου θα πρέπει να γνωρίζουμε ποιες διευθύνσεις του δικτύου είναι ενεργές για να

ξέρουμε εάν ο ξενιστής στέλνει σε υπάρχουσες ή μη διευθύνσεις, οπότε και να καταλάβουμε αν σκανάρει το δίκτυο. Το πρόβλημα των ενεργών διευθύνσεων του δικτύου περιέχει διάφορα μικρότερα προβλήματα όπως το πως θα ενημερώνεται το σύστημα αν κάποιος βγήκε από το δίκτυο ή αν κάποιος είναι ανενεργός οπότε και δεν υπάρχει πλέον η εγγραφή του. Τέλος, στην περίπτωση αυτή έχουμε και το πρόβλημα εντοπισμού της διεύθυνσης του επιτιθέμενου καθώς όντας σε διαφορετικό δίκτυο δεν μπορούμε να τον βρούμε χρησιμοποιώντας τα δεδομένα μόνο του δικού μας δικτύου.

5.4.4 Λύσεις

Τα κύρια προβλήματα που έχουμε να λύσουμε είναι δύο, η συνεχής ενημέρωση για το ποιες IP διευθύνσεις είναι ενεργές και ο εντοπισμός του επιτιθέμενου παρότι βρίσκεται σε διαφορετικό δίκτυο. η λύση των προβλημάτων αυτών είναι δυνατή χάρη στο ευφυές προγραμματιζόμενο δίκτυό μας και του Pyretic.

Αρχικά, κάνουμε την παραδοχή ότι ενεργή θεωρείται μια IP διεύθυνση την πρώτη φορά που θα στείλει ένα μήνυμα προς οποιαδήποτε άλλη IP διεύθυνση. Μια ενεργή διεύθυνση παύει να θεωρείται ενεργή όταν αποσυνδεθεί από το δίκτυο ή όταν είναι ανενεργή για περισσότερο χρόνο απ’ όσο διαρκεί η εγγραφή της στις ροές του μεταγωγέα.

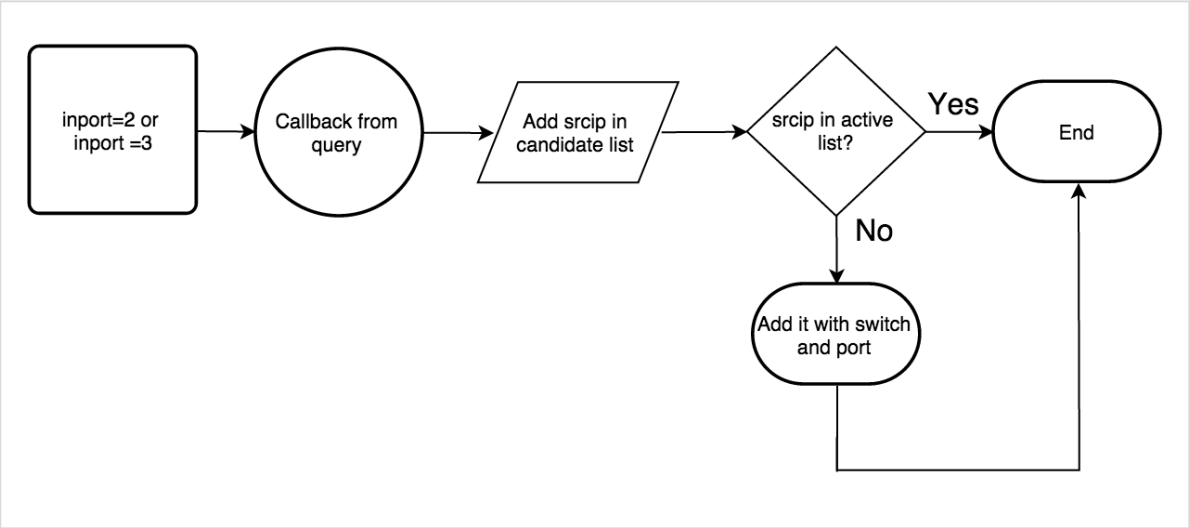
Από το Neighbor Discovery πρωτόκολλο μπορούμε να αντλήσουμε πληροφορίες για την κίνηση εντός του δικτύου και για τον λόγο αυτό θα χρειαστεί να συνδυάσουμε την κίνηση και από τους δυο μεταγωγείς για να εντοπίσουμε τον επιτιθέμενο. Συγκεκριμένα, στους μεταγωγείς θα κρατάμε όποτε ένας ξενιστής στέλνει σε κάποια καινούρια IP διεύθυνση αλλά και όταν δέχεται κάποιο νέο πακέτο από άλλο δίκτυο. Έτσι, στην περίπτωση του σκαναρίσματος του δικτύου έχουμε στον ένα μεταγωγέα την πληροφορία σχετικά με την διεύθυνση αποστολέα και παραλήπτη και στον δεύτερο μεταγωγέα του παραλήπτη που δεν υπάρχει. Ο συνδυασμός των δυο αυτών στοιχείων μας οδηγεί στην εύρεση της διεύθυνσης IP του επιτιθέμενου.

5.4.5 Υλοποίηση

Για τον εντοπισμό και την αντιμετώπιση του σκαναρίσματος δικτύου από τον επιτιθέμενο, ο οποίος βρίσκεται σε άλλο ελεγχόμενο δίκτυο, φτιάξαμε την Pyretic εφαρμογή anti-honeypot.py.

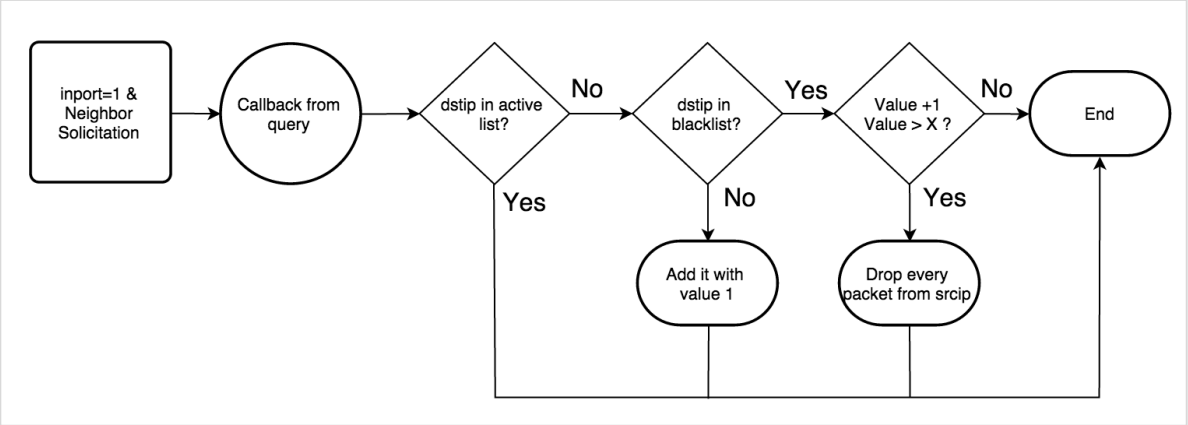
Το anti-honeypot.py, όπως και το internal.py, βασίζεται στην πολιτική ερωτήσεων (query policy) του Pyretic, η οποία μας δίνει την δυνατότητα να παρακολουθούμε το δίκτυο και να καθορίζουμε την ενεργοποίηση συναρτήσεων χωρίς να επηρεάζουμε την ροή των πακέτων. Στην περίπτωσή μας, έχουμε δυο ερωτήσεις (queries) που ενεργοποιούνται με βάση τα χαρακτηριστικά των πακέτων που περνάνε από το τους μεταγωγείς και μια που ενεργοποιείται κάθε 15 λεπτά.

Το πρώτο ερώτημα παρακολουθεί πότε ένα πακέτο στέλνεται από κάποιον ξενιστή εντός τους δικτύου και έχει ένα νέο ζευγάρι IP διεύθυνσης παραλήπτη και αποστολέα. Το ερώτημα αυτό ενεργοποιεί την συνάρτηση lists η οποία αρχικά καταγράφει στην λεξικό (dictionary) candidates την IP διεύθυνση αποστολέα και παραλήπτη και ελέγχει εάν ο αποστολέας βρίσκεται στην λίστα με της ενεργές διευθύνσεις (λεξικό active) και αν δεν είναι τον προσθέτει. Τις πληροφορίες του λεξικού candidates θα τις χρειαστούμε αργότερα ώστε να καταλάβουμε ποια IP διεύθυνση στέλνει μηνύματα σε ανενεργές διευθύνσεις του άλλου δικτύου.



Εικόνα 5.10: Το διάγραμμα ροής στο πρώτο query του anti-honeypot.py

Το δεύτερο ερώτημα παρακολουθεί πότε ένα πακέτο έρχεται από άλλο δίκτυο και έχει ένα νέο ζευγάρι IP διεύθυνσης παραλήπτη και αποστολέα. Αρκεί μια από τις 2 διευθύνσεις να είναι νέες ώστε το ερώτημα αυτό να ενεργοποιήσει την συνάρτηση is_it_active. Η is_it_active ελέγχει εάν η διεύθυνση παραλήπτη είναι ενεργή (ανήκει στο λεξικό active) και αν δεν είναι, τότε μαθαίνει από το λεξικό candidates τον αποστολέα του πακέτου καθώς ο αποστολέας αυτός είναι ύποπτος για σκανάρισμα δικτύου. Έπειτα, καταχωρείται η IP διεύθυνση του αποστολέα στο λεξικό blacklist με την τιμή 1 ή σε περίπτωση που υπάρχει ήδη η διεύθυνση στο λεξικό τότε αυξάνεται η τιμή, η οποία αντιστοιχεί στην διεύθυνση αυτή, κατά ένα. Όταν, μια IP διεύθυνση φτάσει την τιμή 10 στο λεξικό blacklist τότε αυτόματα ενεργοποιείται η συνάρτηση block_this_ip και πλέον η συγκεκριμένη IP διεύθυνση θεωρείται ότι σκανάρει το δίκτυο και δεν τις επιτρέπεται να στείλει κανένα πακέτο.



Εικόνα 5.10: Το διάγραμμα ροής στο δεύτερο query του anti-honeypot.py

Το τρίτο ερώτημα μας βοηθάει στο να προσομοιώσουμε έναν ξενιστή ο οποίος είναι ανενεργός και παύει να ανήκει στο δίκτυο καθώς δεν υπάρχει πλέον η εγγραφή του στην neighbor discovery cache του δρομολογητή. Ουσιαστικά, ελέγχει πόσα πακέτα έχει στείλει ο κάθε ξενιστής κάθε 15 λεπτά και όταν βρεθεί ότι κάποιος ξενιστής δεν έχει στείλει κανένα πακέτο τα τελευταία 15 λεπτά τότε τον αφαιρεί από το λεξικό active.

Τέλος, η ενημέρωση του συστήματος και του λεξικού active για το γεγονός ότι κάποιος ξενιστής έφυγε χρειάζεται και αυτή να υλοποιηθεί μέσω της Pyretic εφαρμογής μας. Η υλοποίηση αυτή θα ήταν εξαιρετικά δύσκολη εάν δεν υπήρχαν τα Network Objects του Pyretic Runtime συστήματος. Το Pyretic για την λειτουργία ελέγχει κάθε 0.25 δευτερόλεπτα εάν η τοπολογία του δικτύου έχει αλλάξει έτσι ώστε να μπορεί αυτόματα να μετατρέψει κατάλληλα τους κανόνες των μεταγωγέων και να εξακολουθούν να λειτουργούν οι πολιτικές που έχει προδιαγράψει ο προγραμματιστής. Μέσω της λειτουργίας του το Pyretic ενημερώνεται για τα γεγονότα που επηρεάζουν την τροποποίηση του δικτύου όπως είναι η συνάρτηση port_down το οποίο ενεργοποιείται όταν κάποιος ξενιστής φεύγει από το δίκτυο. Έτσι, εμείς από την συνάρτηση αυτή τραβάμε την πληροφορία που χρειαζόμαστε, δηλαδή την διεπαφή του μεταγωγέα από την οποία αποσυνδέθηκε κάποιος ξενιστής και αφαιρούμε την αντίστοιχη IP διεύθυνση από το λεξικό active.

6 ΑΠΟΤΕΛΕΣΜΑΤΑ

Στο κεφάλαιο αυτό θα γίνει η παρουσίαση των αποτελεσμάτων των δύο εφαρμογών που περιγράφηκαν στα κεφάλαια 5.3 και 5.4 καθώς και μια σύντομη περιγραφή άλλων επιθέσεων που μπορούν να εντοπιστούν και να αντιμετωπιστούν με βάση τις εφαρμογές αυτές.

6.1 Αποτελέσματα σκαναρίσματος δικτύου από επιτιθέμενο εντός δικτύου

Η εφαρμογή internal.py παρακολουθεί την κίνηση του δικτύου και εντοπίζει πότε κάποιος ξενιστής εντός του δικτύου προσπαθεί να το σκανάρει. Όταν εντοπίσει ότι κάποια IP διεύθυνση έχει στείλει Neighbor Solicitation σε περισσότερες από X ανενεργές IP διευθύνσεις τότε μπλοκάρει την MAC διεύθυνση αυτή από το να ξαναστείλει οποιοδήποτε μήνυμα. Στην περίπτωσή μας ορίζουμε το την μεταβλητή X στον αριθμό δύο, δηλαδή να στείλει αποτυχημένα Neighbor Solicitation μηνύματα σε μέχρι δύο ανενεργές IP διευθύνσεις προτού μπλοκαριστεί η MAC διεύθυνσή του. Συγκεκριμένα, στην συνάρτηση check_destination οι παρακάτω γραμμές:

```
if blacklist[pkt['srcip']] > 2:
    self.block_this_ip(pkt)
```

Φτιάξαμε λοιπόν ένα μικρό κείμενο κώδικα για να στείλουμε ping6 από το h1 αρχικά στην ενεργή διεύθυνση fc00:0:0:1::2, έπειτα σε κάποιες μη υπαρκτές IP διευθύνσεις όπως η fc00:0:0:1::4, η fc00:0:0:1::5 και η fc00:0:0:1::6 και τέλος ξανά στην ενεργή fc00:0:0:1::2. Αναμένουμε ότι αρχικά το echo request θα φτάσει στην διεύθυνση του h2 (fc00:0:0:1::2), δεν θα φτάσει στις μη υπαρκτές διευθύνσεις και τελικά θα αποτύχει στην διεύθυνση του h2 καθώς πλέον θα έχει μπλοκαριστεί η MAC διεύθυνση του h1 από το σύστημα ως επιτιθέμενου για σκανάρισμα δικτύου. Αφότου ξεκινήσαμε την τοπολογία του υποκεφαλαίου 5.3.1 μέσω του MiniNExT, μπήκαμε με την εντολή xterm h1 στον ξενιστή h1 και τρέξαμε τον παρακάτω python κώδικα του αρχείου pings-internal.py:

```
import os

list = [2, 4, 5, 6, 2]

for ping in list:
     address = 'fc00:0:0:1::' + str(ping)
     response = os.system('ping6 -c1 ' + address)
```

Εικόνα 6.1: Ο κώδικας του pings-internal.py

Τα αποτελέσματα του κώδικα αυτού είναι:

```
root@mininet-vm:~# python pings.py
PING fc00:0:0:1::2(fc00:0:0:1::2) 56 data bytes

64 bytes from fc00:0:0:1::2: icmp_seq=1 ttl=64 time=197 ms

--- fc00:0:0:1::2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 197.442/197.442/197.442/0.000 ms
PING fc00:0:0:1::4(fc00:0:0:1::4) 56 data bytes
From fc00:0:0:1::1 icmp_seq=1 Destination unreachable: Address unreachable

--- fc00:0:0:1::4 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

PING fc00:0:0:1::5(fc00:0:0:1::5) 56 data bytes
From fc00:0:0:1::1 icmp_seq=1 Destination unreachable: Address unreachable

--- fc00:0:0:1::5 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

PING fc00:0:0:1::6(fc00:0:0:1::6) 56 data bytes
From fc00:0:0:1::1 icmp_seq=1 Destination unreachable: Address unreachable

--- fc00:0:0:1::6 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

PING fc00:0:0:1::2(fc00:0:0:1::2) 56 data bytes

--- fc00:0:0:1::2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

root@mininet-vm:~# _
```

Εικόνα 6.2: Το αποτέλεσμα του προγράμματος internal.py

Βλέπουμε ότι όπως αναμέναμε το internal.py σταμάτησε τα ping του h1 ακόμα και σε ενεργούς ξενιστές όπως ο h2 (fc00:0:0:1::2) καθώς πλέον θεωρείται ότι κάνει σκανάρισμα δικτύου.

6.2 Αποτελέσματα σκαναρίσματος δικτύου από επιτιθέμενο σε άλλο ελεγχόμενο δίκτυο

Η εφαρμογή anti-honeypot.py παρακολουθεί την κίνηση του δικτύου και εντοπίζει πότε κάποιος ξενιστής από κάποιο άλλο ελεγχόμενο δίκτυο προσπαθεί να το σκανάρει. Ως ελεγχόμενο θεωρούμε ένα δίκτυο στου οποίου τον μεταγωγέα έχουμε πρόσβαση μέσω του ελεγκτή. Όταν εντοπίσει ότι κάποια IP διεύθυνση έχει στείλει Neighbor Solicitation σε περισσότερες από X ανενεργές IP διευθύνσεις τότε μπλοκάρει την IP διεύθυνση που τα έστειλε έτσι ώστε να μην ξαναστείλει οποιοδήποτε άλλο μήνυμα. Στην περίπτωσή μας ορίζουμε την μεταβλητή X στον αριθμό τρία, δηλαδή να στείλει αποτυχημένα Neighbor Solicitation μηνύματα σε μέχρι τρεις ανενεργές IP διευθύνσεις προτού μπλοκαριστεί η IP

διεύθυνσή του αποστολέα. Συγκεκριμένα, στην συνάρτηση is_it_active οι παρακάτω γραμμές:

```
if blacklist[pkt['srcip']] > 3:
    self.block_this_ip(pkt)
```

Φτιάξαμε λοιπόν ένα μικρό κείμενο κώδικα για να στείλουμε ping6 από τον h3 αρχικά στις ενεργές διευθύνσεις του h1 (fc00:0:0:1::1) και του h2 (fc00:0:0:1::2), έπειτα σε κάποιες μη υπαρκτές IP διευθύνσεις όπως η fc00:0:0:1::3, η fc00:0:0:1::4 και η fc00:0:0:1::5 και τέλος ξανά στην ενεργές διευθύνσεις h1 και h2. Αναμένουμε ότι αρχικά το echo request θα φτάσει στις διευθύνσεις h1 και h2, δεν θα φτάσει στις μη υπαρκτές διευθύνσεις και τελικά θα αποτύχει την δεύτερη φορά στις διευθύνσεις των h1 και h2 καθώς πλέον θα έχει μπλοκαριστεί η IP διεύθυνση του h3 από το σύστημα ως επιτιθέμενου για σκανάρισμα δικτύου. Αφότου ξεκινήσαμε την τοπολογία του υποκεφαλαίου 5.4.1 μέσω του MiniNExT, μπήκαμε με την εντολή xterm h3 στον ξενιστή h3 και τρέξαμε τον παρακάτω python κώδικα του αρχείου pings.py:

```
import os

list = [1, 2, 3, 4, 5, 6, 1, 2]

for ping in list:
     address = 'fc00:0:0:1::' + str(ping)
     response = os.system('ping6 -c1 ' + address)
```

Εικόνα 6.3: Ο κώδικας του pings.py

Τα αποτελέσματα του κώδικα αυτού είναι:

```
root@mininet-vm:~# python pings.py
PING fc00:0:0:1::1(fc00:0:0:1::1) 56 data bytes
64 bytes from fc00:0:0:1::1: icmp_seq=1 ttl=63 time=1221 ms

--- fc00:0:0:1::1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 1ms
rtt min/avg/max/mdev = 1221.837/1221.837/1221.837/0.000 ms
PING fc00:0:0:1::2(fc00:0:0:1::2) 56 data bytes
64 bytes from fc00:0:0:1::2: icmp_seq=1 ttl=63 time=706 ms

--- fc00:0:0:1::2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 706.019/706.019/706.019/0.000 ms
PING fc00:0:0:1::3(fc00:0:0:1::3) 56 data bytes
From fc00:0:0:2::8 icmp_seq=1 Destination unreachable: Address unreachable

--- fc00:0:0:1::3 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

PING fc00:0:0:1::4(fc00:0:0:1::4) 56 data bytes
From fc00:0:0:2::8 icmp_seq=1 Destination unreachable: Address unreachable

--- fc00:0:0:1::4 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

PING fc00:0:0:1::5(fc00:0:0:1::5) 56 data bytes
From fc00:0:0:2::8 icmp_seq=1 Destination unreachable: Address unreachable

--- fc00:0:0:1::5 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

PING fc00:0:0:1::6(fc00:0:0:1::6) 56 data bytes

--- fc00:0:0:1::6 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

PING fc00:0:0:1::1(fc00:0:0:1::1) 56 data bytes

--- fc00:0:0:1::1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

PING fc00:0:0:1::2(fc00:0:0:1::2) 56 data bytes

--- fc00:0:0:1::2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

root@mininet-vm:~#
```

Εικόνα 6.4: Το αποτέλεσμα του προγράμματος anti-honeypot.py

Βλέπουμε ότι όπως αναμέναμε το anti-honeypot.py σταμάτησε τα ping του h3 ακόμα και σε ενεργούς ξενιστές όπως το h1 (fc00:0:0:1::1) και το h2 (fc00:0:0:1::2) καθώς πλέον θεωρείται ότι κάνει σκανάρισμα δικτύου.

Για να βεβαιωθούμε ότι η εφαρμογή λειτουργεί σωστά θα κάνουμε άλλη μια δοκιμή αλλάζοντας την μεταβλητή X, η οποία αφορά τον αριθμό των ανενεργών διευθύνσεων που επιτρέπεται να σταλούν Neighbor Solicitation μηνύματα, προτού μπλοκαριστεί η IP διεύθυνση του αποστολέα. Έτσι, θα δοκιμάσουμε να ανεβάσουμε στον αριθμό 5 την μεταβλητή X έτσι ώστε να δούμε αν στο προηγούμενο πείραμα θα σταλούν τα μηνύματα κανονικά στον h1 και στον h2 και την δεύτερη φορά. Τρέχουμε λοιπόν ξανά το προηγούμενο

πείραμα έχοντας αλλάξει μόνο την παρακάτω γραμμή της συνάρτησης is_it_active από τρία σε πέντε:

```
if blacklist[pkt['srcip']] > 5:
    self.block_this_ip(pkt)
```

Το νέο αποτέλεσμα του προγράμματός μας θα είναι:

```
root@mininet-vm:~# python pings.py
PING fc00:0:0:1::1(fc00:0:0:1::1) 56 data bytes
64 bytes from fc00:0:0:1::1: icmp_seq=1 ttl=63 time=1665 ms

--- fc00:0:0:1::1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1665.027/1665.027/1665.027/0.000 ms
PING fc00:0:0:1::2(fc00:0:0:1::2) 56 data bytes
64 bytes from fc00:0:0:1::2: icmp_seq=1 ttl=63 time=1789 ms

--- fc00:0:0:1::2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1789.266/1789.266/1789.266/0.000 ms
PING fc00:0:0:1::3(fc00:0:0:1::3) 56 data bytes
From fc00:0:0:2::8 icmp_seq=1 Destination unreachable: Address unreachable

--- fc00:0:0:1::3 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

PING fc00:0:0:1::4(fc00:0:0:1::4) 56 data bytes
From fc00:0:0:2::8 icmp_seq=1 Destination unreachable: Address unreachable

--- fc00:0:0:1::4 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

PING fc00:0:0:1::5(fc00:0:0:1::5) 56 data bytes
From fc00:0:0:2::8 icmp_seq=1 Destination unreachable: Address unreachable

--- fc00:0:0:1::5 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

PING fc00:0:0:1::1(fc00:0:0:1::1) 56 data bytes
64 bytes from fc00:0:0:1::1: icmp_seq=1 ttl=63 time=215 ms

--- fc00:0:0:1::1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 215.310/215.310/215.310/0.000 ms
PING fc00:0:0:1::2(fc00:0:0:1::2) 56 data bytes
64 bytes from fc00:0:0:1::2: icmp_seq=1 ttl=63 time=161 ms

--- fc00:0:0:1::2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 161.902/161.902/161.902/0.000 ms
root@mininet-vm:~#
```

Εικόνα 6.5: Το αποτέλεσμα του προγράμματος anti-honeypot.py με τιμή περιθώριο πέντε ανενεργών διευθύνσεων

Παρατηρούμε ότι όπως αναμέναμε το anti-honeypot.py επέτρεψε στον h3 να στείλει κανονικά μηνύματα στους ενεργούς ξενιστές h1 (fc00:0:0:1::1) και h2 (fc00:0:0:1::2) καθώς δεν έχει ξεπεράσει το επιτρεπτό όριο ώστε να μπει στην blacklist της εφαρμογής μας.

6.3 Εντοπισμός άλλων επιθέσεων

Με τις παραλλαγές που έχουμε κάνει στο Pyretic και έχοντας ως βάση τα προγράμματα internal.py και anti-honeypot.py, υπάρχει η δυνατότητα εντοπισμού και αντιμετώπισης πολλών άλλων επιθέσεων του IPv6. Ο κύριος λόγος είναι ότι οι περισσότερες επιθέσεις στο IPv6 επίπεδο έχουν ως επίκεντρο το Neighbor Discovery Protocol το οποίο εμείς μπορούμε εύκολα να εντοπίσουμε έχοντας εντάξει στα χαρακτηριστικά του κάθε ICMPv6 πακέτου την αναγνώριση του τύπου του με το πεδίο icmpv6_type.

6.3.1 Εντοπισμός Ψεύτικων Διαφημίσεων Δρομολογητή (Fake Router Advertisements)

Μπορούμε να εντοπίσουμε τα ψεύτικα Router Advertisements παρακολουθώντας την κίνηση του δικτύου και ελέγχοντας εάν τα πακέτα με icmpv6_type με τιμή 134 στέλνονται από την διεπαφή του πραγματικού δρομολογητή. Όσα στέλνονται από οποιαδήποτε άλλη διεπαφή σημαίνει ότι είναι ψεύτικα.

6.3.2 Πλημμύρα Ανακάλυψης Γείτονα - άρνησης υπηρεσιών (Neighbor Discovery Flood - DoS)

Η πλημμύρα Ανακάλυψης Γείτονα στηρίζεται στην συνεχή αποστολή Neighbor Solicitation από τον δρομολογητή σύμφωνα με τις απαιτήσεις κάποιου εξωτερικού ξενιστή για μια αυθαίρετη IP διεύθυνση. Μπορούμε να το αντιμετωπίσουμε με τον μηχανισμό του anti-honeypot συνδυάζοντας πληροφορίες του δρομολογητή με τους μεταγωγείς.

6.3.3 Ρύθμιση Ψεύτικου Προθέματος Διεύθυνσης (Bogus Address Configuration Prefix)

Στην ρύθμιση ψεύτικου προθέματος διεύθυνσης όλα ξεκινάνε με την αποστολή ενός ψεύτικου Router Advertisement μηνύματος. Επομένως, μπορούμε να την αντιμετωπίσουμε με τον ίδιο τρόπο που περιγράψαμε στο 6.3.1.

6.3.4 Επίθεση Ανίχνευσης Διπλής Διεύθυνσης (Duplicate Address Detection Attack - DoS)

Η επίθεση ανίχνευσης διπλής διεύθυνσης στηρίζεται στην συνεχή αποστολή Neighbor Solicitation ή Neighbor Advertisement μηνυμάτων ως απάντηση στον έλεγχο ενός ξενιστή για το αν η διεύθυνση που μόλις κατέλαβε από την χωρίς επίβλεψη κατάστασης αυτόματη ρύθμιση διεύθυνσης είναι μοναδική. Με την χρήση του πεδίου icmpv6_type μπορούμε να διακρίνουμε τα Neighbor Solicitation και τα Neighbor Advertisement μηνύματα. Ελέγχοντας τα μηνύματα αυτά, μπορούμε να διακρίνουμε εάν μια MAC διεύθυνση έχει απαντήσει σε περισσότερες από μια IP διευθύνσεις ως κάτοχος.

7 ΕΠΕΚΤΑΣΕΙΣ

Το τροποποιημένο σύστημα Pyretic σε συνδυασμό με τις εφαρμογές internal.py και anti-honeypot.py μπορούν να επιλύσουν ήδη αρκετά προβλήματα αλλά μπορούν και να επεκταθούν περισσότερο λύνοντας τα χέρια πολλών προγραμματιστών δικτύου.

Αρχικά, υπάρχει η δυνατότητα επέκτασης στον εντοπισμό του σκαναρίσματος δικτύου στο λεγόμενο “Internet of Things”. Το Internet of Things στηρίζεται στο IPv6 και για τον λόγο οι εφαρμογές μας μπορούν εύκολα να προσαρμοστούν στις ανάγκες του.

Επιπλέον, η εργασία αυτή θα μπορούσε να επεκταθεί για να προσφέρει μεγαλύτερη πληθώρα λύσεων εάν ενοποιηθεί με το Kinetic. Το Kinetic είναι μια domain specific γλώσσα η οποία έχει χτιστεί πάνω στο Pyretic και επιτρέπει την διαχείριση του δικτύου με βάση γεγονότα (event-based) που συμβαίνουν όπως ο εντοπισμός ενός εισβολέα ή η υπέρβαση του ορίου το εύρους ζώνης. Ο συνδυασμός των δύο αυτών συστημάτων θα μπορούσε να δημιουργήσει ένα event-based σύστημα για την διαχείριση δικτύων στο IPv6.

Με εξαιρετικό τρόπο μπορούν να συνδυαστούν η εργασία αυτή με τον μηχανισμό ασφάλειας δικτύων honeypot. Το honeypot στην μέση λειτουργία του αποτελεί έναν μηχανισμό ο οποίος απαντά εικονικά σε μηνύματα προς ανενεργές διευθύνσεις ενός δικτύου, όπως θα απαντούσαν κανονικά οι ξενιστές. Με τον τρόπο αυτό προσπαθεί στην μετέπειτα επικοινωνία με τον επιτιθέμενο να αντλήσει πληροφορίες για αυτόν και να τον εντοπίσει. Χρησιμοποιώντας το σύστημά μας, μπορούμε εύκολα να γνωρίζουμε τα honeypots και να δημιουργούμε πολιτικές ανάλογα με τα μηνύματα που στέλνει ο επιτιθέμενος στις διευθύνσεις αυτές.

Τέλος, η εύκολη δημιουργία πολιτικών ασφάλειας δικτύου για τα δίκτυα IPv6 μπορεί να χρησιμοποιηθεί για την μεταφορά των λύσεων που υπάρχουν ήδη για τα IPv4 δίκτυα, σε λύσεις των IPv6 δικτύων.

Βιβλιογραφία

[1] Software – Defined Networking https://en.wikipedia.org/wiki/Software-defined_networking
[2] Inside SDN Architecture https://www.sdxcentral.com/resources/sdn/inside-sdn-architecture/
[3] ONF White Paper, "Software-Defined Networking: The New Norm for Networks", April 2012
[4] OpenFlow https://en.wikipedia.org/wiki/OpenFlow
[5] Open Networking Foundation (ONF), “OpenFlow Switch Specification”, Version 1.5, December 2014
[6] Using the POX SDN controller http://www.brianlinkletter.com/using-the-pox-sdn-controller/
[7] IPv6 https://el.wikipedia.org/wiki/IPv6
[8] IPv6 Address https://en.wikipedia.org/wiki/IPv6_address
[9] Solicited-Node Multicast Address https://en.wikipedia.org/wiki/Solicited-node_multicast_address
[10] Neighbor Discovery Protocol https://en.wikipedia.org/wiki/Neighbor_Discovery_Protocol
[11] Multicast IPv6 Addresses https://technet.microsoft.com/en-us/library/cc781068(v=ws.10).aspx
[12] S. Bradner, A. Mankin, “RFC 1752 - The Recommendation for the IP Next Generation Protocol”, January 1995
[13] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, “Composing Software – Defined Networks”, April 2013
[14] Pyretic Github Repository https://github.com/frenetic-lang/pyretic/wiki
[15] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, “Modular SDN Programming with Pyretic”, October 2013
[16] Ryu Shim https://github.com/fp7-netide/Engine/tree/master/ryu-shim
[17] Open Networking Foundation (ONF), “OpenFlow Switch Specification”, Version 1.3, June 2012
[18] Network Scanning http://searchmidmarketsecurity.techtarget.com/definition/network-scanning
[19] A. Pilihanto, “A Complete Guide on IPv6 Attack and Defense”, November 2011
[20] P. Nikander, J. Kempf, E. Nordmark, “RFC 3756 - IPv6 Neighbor Discovery (ND) Trust Models and Threats”, May 2004
[21] S. Jiang, S. Shen, “Internet Draft - DHCPv6 and CGA Interaction: Problem Statement”, January 2009
[22] MiniNExT Github Repository https://github.com/USC-NSL/miniNExT
[23] POX Wiki https://openflow.stanford.edu/display/ONL/POX+Wiki

Παράρτημα Κώδικα

Η τελική έκδοση του κώδικα της εργασίας μαζί με όλες τις τροποποιήσεις βρίσκεται στο:

https://github.com/ChristosKon/pyretic-IPv6

```
interface r1-eth0
  description Local Network (usually br-lan or eth0)
  ! Enable Linux' link detection :
  link-detect
  ! Send router advertisment messages :
  !no ipv6 nd suppress-ra
  ! Max time allowed between sending unsolicited
  !  multicast router advertisements (s) :
  !ipv6 nd ra-interval 10
  ! Set the IPv6 address of the interface :
  ipv6 address fc00:0:0:1::8/64
  ! Set the IPv6 prefix (to advertize) :
  ipv6 nd prefix fc00:0:0:1::/64

interface r1-eth1
  link-detect
  ipv6 address fc00:0:0:2::8/64
  ipv6 nd prefix fc00:0:0:2::/64

interface lo
  link-detect
  ipv6 forwarding

! Log all messages related to the communication with the ker$
! priority level of informational and above to a f$
debug zebra kernel
log file /tmp/zebra.log informational
```

Quagga αρχείο ρυθμίσεων zebra.conf

```
1  import sys
2  import atexit
3
4  import mininet.util
5  import mininext.util
6  mininet.util.isShellBuiltin = mininext.util.isShellBuiltin
7  sys.modules['mininet.util'] = mininet.util
8
9  from mininet.node import OVSController, RemoteController
10 from mininet.log import setLogLevel, info
11
12 from mininext.cli import CLI
13 from mininext.net import MiniNExT
14
15 from topo import QuaggaTopo
16
17 net = None
18
19 def startNetwork():
20
21     info('** Creating Quagga network topology\n')
22     topo = QuaggaTopo()
23
24     info('** Starting the network\n')
25     global net
26     net = MiniNExT(topo, controller=RemoteController)
27     net.start()
28     h1 = net.get('h1')
29     h1.cmd('ifconfig h1-eth0 inet6 add fc00:0:0:1::1/64')
30     h1.cmd('ip -6 route add fc00:0:0:2::/64 via fc00:0:0:1::8')
31     h2 = net.get('h2')
32     h2.cmd('ifconfig h2-eth0 inet6 add fc00:0:0:1::2/64')
33     h2.cmd('ip -6 route add fc00:0:0:2::/64 via fc00:0:0:1::8')
34     h3 = net.get('h3')
35     h3.cmd('ifconfig h3-eth0 inet6 add fc00:0:0:2::1/64')
36     h3.cmd('ip -6 route add fc00:0:0:1::/64 via fc00:0:0:2::8')
37     h4 = net.get('h4')
38     h4.cmd('ifconfig h4-eth0 inet6 add fc00:0:0:2::2/64')
39     h4.cmd('ip -6 route add fc00:0:0:1::/64 via fc00:0:0:2::8')
40     r1 = net.get('r1')
41     r1.cmd('route -A inet6 add fc00:0:0:1::/64 dev a1-eth0')
42     r1.cmd('route -A inet6 add fc00:0:0:2::/64 dev a1-eth1')
43
44     info('** Running CLI\n')
45     CLI(net)
46
47 def stopNetwork():
48     "stops a network (only called on a forced cleanup)"
49     if net is not None:
50         info('** Tearing down Quagga network\n')
51         net.stop()
52
53 if __name__ == '__main__':
54     # Force cleanup on exit by registering a cleanup function
55     atexit.register(stopNetwork)
56
57     # Tell mininet to print useful information
58     setLogLevel('info')
59     startNetwork()
```

MiniNExT αρχείο για την εκκίνηση της τοπολογίας start.py

```
import inspect
import os
from mininext.topo import Topo
from mininext.services.quagga import QuaggaService

from collections import namedtuple

QuaggaHost = namedtuple("QuaggaHost", "name ip loIP")
net = None

class QuaggaTopo(Topo):

    "Creates a topology of Quagga routers"

    def __init__(self):
        """Initialize a Quagga topology with 1 routers, configure their IP
           addresses, loop back interfaces, and paths to their private
           configuration directories."""
        Topo.__init__(self)

        # Directory where this file / script is located"
        selfPath = os.path.dirname(os.path.abspath(
            inspect.getfile(inspect.currentframe())))  # script directory

        # Initialize a service helper for Quagga with default options
        quaggaSvc = QuaggaService(autoStop=False)

        # Path configurations for mounts
        quaggaBaseConfigPath = selfPath + '/configs/'

        # List of Quagga host configs
        quaggaHosts = []
        quaggaHosts.append(QuaggaHost(name='r1',))
        h1 = self.addHost( 'h1', mac='00:00:00:00:00:01' )
        h2 = self.addHost( 'h2', mac='00:00:00:00:00:02' )
        h3 = self.addHost( 'h3', mac='00:00:00:00:00:03' )
        h4 = self.addHost( 'h4', mac='00:00:00:00:00:04' )
        s1 = self.addSwitch( 's1' )
        s2 = self.addSwitch( 's2' )

```

```
41          # Add switch for IXP fabric
42          #ixpfabric = self.addSwitch('fabric-sw1')
43
44          # Setup each Quagga router, add a link between it and the IXP fabric
45          for host in quaggaHosts:
46
47              # Create an instance of a host, called a quaggaContainer
48              r1 = self.addHost(name=host.name,
49                                                ip=host.ip,
50                                                hostname=host.name,
51                                                privateLogDir=True,
52                                                privateRunDir=True,
53                                                inMountNamespace=True,
54                                                inPIDNamespace=True,
55                                                inUTSNamespace=True)
56
57              # Add a loopback interface with an IP in router's announced ran$
58              self.addNodeLoopbackIntf(node=host.name, ip=host.loIP)
59
60              # Configure and setup the Quagga service for this node
61              quaggaSvcConfig = \
62                  {'quaggaConfigPath': quaggaBaseConfigPath + host.name}
63              self.addNodeService(node=host.name, service=quaggaSvc,
64                                  nodeConfig=quaggaSvcConfig)
65
66              #self.addLink()
67              self.addLink(s1, r1)
68              self.addLink(s2, r1)
69              self.addLink(h1, s1)
70              self.addLink(h2, s1)
71              self.addLink(h3, s2)
72              self.addLink(h4, s2)
```

MiniNExT αρχείο για την δημιουργία της τοπολογίας topo.py

Αλλαγές κώδικα σε Pyretic για την υποστήριξη IPv6

```
737         # Drop all IPv6 packets by default.
738         self.install_rule(({'switch':s, 'ethtype':IPV6_TYPE},
739                            TABLE_START_PRIORITY + 1,
740                            [],
741                            self.get_cookie(self.default_cookie, table_id),
742                            False,
743                            table_id))
```

O default κανόνας στο runtime.py που κάνει drop τα IPv6 πακέτα και πρέπει να διαγράψουμε

Στο pyretic/core/packet.py:

```
454        -@of_field("ipv6.srcip", "srcip", ether_validator(IPV6), version="1.0")
       453 +@of_field("ipv6.src", "srcip", ether_validator(IPV6), version="1.0")
455    454  class Ipv6SrcIp(object): pass
456    455
457        -@of_field("ipv6.dstip", "dstip", ether_validator(IPV6), version="1.0")
       456 +@of_field("ipv6.dst", "dstip", ether_validator(IPV6), version="1.0")
```

Στο pyretic/core/network.py:

```
95   95              # string encoding
96   96              else:
97      -                b.frombytes(socket.inet_aton(ip))
     97 +                b.frombytes(socket.inet_pton(socket.AF_INET6, ip))
```

```
113  113     def __repr__(self):
114     -        return socket.inet_ntoa(self.to_bytes())
115     -
     114 +        if socketnt.has_ipv6:
     115 +            return socket.inet_ntop(socket.AF_INET6,self.to_bytes())
     116 +        else:
     117 +            return socket.inet_ntop(socket.AF_INET,self.to_bytes())
```

```
113  113     def __repr__(self):
114     -        if socketnt.has_ipv6:
     114 +        if socket.has_ipv6:
115  115             return socket.inet_ntop(socket.AF_INET6,self.to_bytes())
116  116         else:
117  117             return socket.inet_ntop(socket.AF_INET,self.to_bytes())
```

```
95   95              # string encoding
96   96              else:
97      -                b.frombytes(socket.inet_pton(socket.AF_INET6, ip))
98      -
     97 +                try:
     98 +                    b.frombytes(socket.inet_pton(socket.AF_INET6, ip))
     99 +                except:
    100 +                    b.frombytes(socket.inet_pton(socket.AF_INET, ip))
```

```
113  115     def __repr__(self):
114     -        if socket.has_ipv6:
     116 +        try:
115  117             return socket.inet_ntop(socket.AF_INET6,self.to_bytes())
116     -        else:
     118 +        except:
117  119             return socket.inet_ntop(socket.AF_INET,self.to_bytes())
```

Στο pyretic/core/util.py:

```
38  38  import sys
39      -from ipaddr import IPv4Network, AddressValueError, IPv4Address
    39  +from ipaddr import IPv4Network, AddressValueError, IPv4Address, IPv6Network, IPv6Address
40  40
```

```
42  42  def singleton(f):
        @@ -174,14 +174,21 @@ def string_to_network(ip_str):
174 174     """ Return an IPv4Network object from a dotted quad IP address/subnet. """
175 175     try:
176 176         return IPv4Network(ip_str)
177     -   except AddressValueError:
178     -       raise TypeError('Input not a valid IP address!')
    177 +   except:
    178 +       try:
    179 +           return IPv6Network(ip_str)
    180 +       except AddressValueError:
    181 +           raise TypeError('Input not a valid IP address!')
```

```
180 183 def string_to_IP(ip_str):
181 184     try:
182 185         return IPv4Address(ip_str)
183     -   except AddressValueError:
184     -       raise TypeError('Input not a valid IP address!')
    186 +   except:
    187 +       try:
    188 +           return IPv6Address(ip_str)
    189 +       except AddressValueError:
    190 +           raise TypeError('Input not a valid IP address!')
    191 +
```

Η Pyretic εφαρμογή **internal.py**

```
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *
from pyretic.core.network import IPAddr
from collections import defaultdict

active = defaultdict(list)
candidates = {}
blacklist = {}
innocent = [IPAddr("fc00:0:0:1::8"),IPAddr("fc00:0:0:2::8")]

class dynamic_check(DynamicPolicy):
    """Dynamic policy that detect network scanners according to the
number of neighbor solicitation
       to non active IP addresses."""
    def __init__(self):
```

```
        super(dynamic_check,self).__init__()
        self.flood = flood()            # REUSE A SINGLE FLOOD INSTANCE
        self.set_initial_state()
        self.current = {}

    def set_initial_state(self):
        q = packets()

        self.query = (match(icmpv6_type=135)) >> q
        q.register_callback(self.check_destination)

        self.query2 = count_packets(900, ['srcip'])
        self.query2.register_callback(self.timeout)

        self.query3 = packets(1,['srcmac','switch'])
        self.query3.register_callback(self.learn_new_MAC)

        self.forward = self.flood  # REUSE A SINGLE FLOOD INSTANCE
        self.update_policy()

    def set_network(self,network):
        change = network.something
        for k, v in active.items():
            if change == v:
                active.__delitem__(k)

    def update_policy(self):
        """Update the policy based on current forward and query
policies"""
        self.policy = self.forward + self.query + self.query2 +
self.query3

    def learn_new_MAC(self,pkt):
        """Update forward policy based on newly seen (mac,port) + add
srcip in active {srcip: [switch inport]}"""
        self.forward = if_(match(dstmac=pkt['srcmac'],
                                 switch=pkt['switch']),
                           fwd(pkt['inport']),
                           self.forward)
        self.update_policy()
        if pkt['srcip'] not in active:
            active[pkt['srcip']].append(pkt['switch'])
            active[pkt['srcip']].append(pkt['inport'])

    def check_destination(self,pkt):
        """Update forward policy based on newly seen (mac,port)"""
        address = ""
        raw_bytes = [ord(c) for c in pkt['raw']]
        eth_payload_bytes = raw_bytes[pkt['header_len']:]
        for i in range(48,64):
            address += str(hex(eth_payload_bytes[i])[2:].zfill(2))
            if i%2 == 1:
                address += ":"
        address = address[:-1]
        destination= IPAddr(address)
        if (destination not in active) and (destination not in innocent):
            if blacklist.get(pkt['srcip']) is None:
                blacklist[pkt['srcip']] = 1
            else:
                blacklist[pkt['srcip']] += 1
```

```
                if blacklist[pkt['srcip']] >= 6:
                    self.block_this_ip(pkt)

    def block_this_ip(self,pkt):
        """Update forward policy based on newly blocked srcmac"""
        self.forward = if_(match(srmac=pkt['srcmac']), drop,
self.forward)
        print "blacklist:" + str(blacklist)
        self.update_policy()

    def timeout(self,counts):
        """Represent host's timeout. Remove inactive IPs from 'active'
dictionary"""
        old = self.current.copy()
        for key, value in counts.iteritems():
            self.current[key] = value

        for key in old:
            if key in self.current:
                new_packs = self.current[key] - old[key]
                if str(new_packs) == "0":
                    k = str(str(key).split('\'')[-2]).split('/')[-2]
                    if active[IPAddr(k)]:
                        active.__delitem__(IPAddr(k))
            else:
                pass

def main():
    return dynamic_check()
```

Η Pyretic εφαρμογή **anti-honeypot.py**

```
from pyretic.lib.corelib import *
from pyretic.lib.std import *
from pyretic.lib.query import *
from pyretic.core.network import IPAddr
from collections import defaultdict

active = defaultdict(list)
candidates = {}
blacklist = {}
innocent = [IPAddr("fc00:0:0:1::8"),IPAddr("fc00:0:0:2::8")]

class dynamic_check(DynamicPolicy):
    """Dynamic Policy, after initialization it works with queries and
callbacks"""
    def __init__(self):
        super(dynamic_check,self).__init__()
        self.flood = flood()           # REUSE A SINGLE FLOOD INSTANCE
        self.set_initial_state()
        self.current = {}

    def set_initial_state(self):
        """Initialize query policies for monitoring and forwarding"""
        paket_multi = packets(1,['dstip'])
        paket_int = packets(1,['srcip','dstip'])
        self.query3 = count_packets(900, ['srcip'])
```

```
        self.query = match(inport=1) >> match(icmpv6_type=135) >>
paket_multi
        paket_multi.register_callback(self.is_it_active)

        self.query2 = (match(inport=2) | match(inport=3)) >> paket_int
        paket_int.register_callback(self.lists)

        self.query3.register_callback(self.timeout)

        self.forward = self.flood
        self.update_policy()

    def set_network(self,network):
        """A hacky way to update our system if a host left our network"""
        change = network.something
        for k, v in active.items():
            if change == v:
                active.__delitem__(k)

    def update_policy(self):
        """Update the policy based on current forward and query
policies"""
        self.policy = self.forward + self.query + self.query2 +
self.query3

    def is_it_active(self,pkt):
        """Find from Neighbor Solicitation the destination IP and check
for blacklist"""
        address = ""
        raw_bytes = [ord(c) for c in pkt['raw']]
        eth_payload_bytes = raw_bytes[pkt['header_len']:]
        for i in range(48,64):
            address += str(hex(eth_payload_bytes[i])[2:].zfill(2))
            if i%2 == 1:
                address += ":"
        address = address[:-1]
        destination= IPAddr(address)
        if (destination not in active) and (destination not in innocent):
            try:
                self.intruder = candidates[destination]
                if blacklist.get(self.intruder) is None:
                    blacklist[self.intruder] = 1
                else:
                    blacklist[self.intruder] += 1
                    if blacklist[self.intruder] >= 3:
                        self.block_this_ip(pkt)
            except:
                pass

    def block_this_ip(self,pkt):
        """Update forward policy based on newly blocked srcip"""
        self.forward = if_(match(srcip=self.intruder), drop,
self.forward)
        print "blacklist:" + str(blacklist)
        self.update_policy()

    def lists(self,pkt):
        """ Track {dstip: srcip} as candidate for attack + check if new
srcip in active  """
```

```
        candidates[pkt['dstip']] = pkt['srcip']
        if pkt['srcip'] not in active:
            active[pkt['srcip']].append(pkt['switch'])
            active[pkt['srcip']].append(pkt['inport'])

    def timeout(self,counts):
        """Represent host's timeout. Remove inactive IPs from 'active'
dictionary"""
        old = self.current.copy()
        for key, value in counts.iteritems():
            self.current[key] = value

        for key in old:
            if key in self.current:
                new_packs = self.current[key] - old[key]
                if str(new_packs) == "0":
                    k = str(str(key).split('\'')[-2]).split('/')[-2]
                    if active[IPAddr(k)]:
                        active.__delitem__(IPAddr(k))
            else:
                print "{}"

def main():
    return dynamic_check()
```