ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

NETMODE - NETWORK MANAGEMENT & OPTIMAL DESIGN LABORATORY

Αντιμετώπιση Κατανεμημένων Επιθέσεων Μεγάλης Κλίμακας στο Σύστημα Ονοματοδοσίας Τομέων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Γ. Κωστόπουλος

Επιβλέπων : Βασίλειος Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

NETMODE - NETWORK MANAGEMENT & OPTIMAL DESIGN LABORATORY

Αντιμετώπιση Κατανεμημένων Επιθέσεων Μεγάλης Κλίμακας στο Σύστημα Ονοματοδοσίας Τομέων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Γ. Κωστόπουλος

Επιβλέπων : Βασίλειος Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 27η Σεπτεμβρίου 2017

…….	……..	..……
Βασίλειος Μάγκλαρης	Συμεών Παπαβασιλείου	Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2017

……...

Νικόλαος Κωστόπουλος

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

Αφιερώνεται στην Ίριδα – Μαρία

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Βασίλειο Μάγκλαρη για την πολύτιμη καθοδήγησή του και την ευκαιρία που μου έδωσε να ασχοληθώ με το συγκεκριμένο πολύ ενδιαφέρον θέμα. Επίσης, θα ήθελα να ευχαριστήσω ιδιαίτερα τον υποψήφιο διδάκτορα Αδάμ Παυλίδη για τις εξαιρετικές συμβουλές του και το χρόνο που αφιέρωσε σε αυτήν την εργασία. Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου για τη στήριξή τους.

Περίληψη

Το Σύστημα Ονοματοδοσίας Τομέων (Domain Name System, DNS) παρέχει αντιστοίχιση ονομάτων υπολογιστών σε διευθύνσεις IP και αντίστροφα. Η σωστή λειτουργία του είναι ζωτική για την ομαλή λειτουργία του διαδικτύου. Είναι φυσικό, λοιπόν, να αποτελεί υποψήφιο στόχο των Κατανεμημένων Επιθέσεων Άρνησης Παροχής Υπηρεσιών (Distributed Denial of Service Attacks, DDoS attacks).

Μία διαδεδομένη και πολύ αποτελεσματική επίθεση DDoS κατά του DNS είναι η επίθεση water torture, που έχει ως στόχο τον αρμόδιο (authoritative) εξυπηρετητή μιας ζώνης DNS. Ο επιτιθέμενος πλημμυρίζει τον εξυπηρετητή με μεγάλο όγκο ερωτημάτων για να εξαντλήσει την υπολογιστική ισχύ του και να τον καταστήσει ανίκανο να απαντά σε ερωτήματα νομίμων πελατών. Τα ερωτήματα αυτά περιλαμβάνουν ονόματα τυχαίας μορφής, τα οποία είναι βέβαιο πως δεν περιέχονται στα αρχεία ζώνης του εξυπηρετητή. Έτσι, επιτυγχάνεται η παράκαμψη της προσωρινής μνήμης (DNS cache) των αναδρομικών (recursive) εξυπηρετητών που προωθούν την κίνηση, εξασφαλίζοντας ότι όλα τα κακόβουλα μηνύματα θα φτάσουν στο θύμα.

Σε αυτήν την εργασία αναπτύχθηκε μηχανισμός ανίχνευσης και αντιμετώπισης μιας τέτοιας επίθεσης σε περιβάλλον δικτύου οριζομένου από λογισμικό (Software Defined Network, SDN). Πληροφορίες σχετικές με τον εξυπηρετητή και το δίκτυο συλλέγονται από ένα μηχανισμό παρακολούθησης. Έπειτα, ένας μηχανισμός ανίχνευσης μη ομαλής κίνησης αποφαίνεται εάν ο εξυπηρετητής κινδυνεύει ή όχι. Στην πρώτη περίπτωση, ο ελεγκτής SDN εγκαθιστά κανόνες στο μεταγωγέα (switch) του δικτύου, ώστε ερωτήματα από IPs που έχουν χαρακτηριστεί ως κακόβουλες να οδηγούνται σε μία μονάδα αντιμετώπισης. Τα τυχαίας μορφής ερωτήματα απορρίπτονται και τα έγκυρα προωθούνται στον εξυπηρετητή.

Ειδικότερα, οι πληροφορίες συγκεντρώνονται χρησιμοποιώντας δειγματοληψία πακέτων και λογισμικό συλλογής στατιστικών δεδομένων. Η μονάδα ανίχνευσης κινδύνου βασίζεται σε έναν αλγόριθμο μηχανικής μάθησης (machine learning) και η μονάδα που πραγματοποιεί το φιλτράρισμα των ερωτημάτων βασίζεται σε ένα σύνολο από φίλτρα bloom (bloom filters). Το bloom filter είναι μία χωρικά αποδοτική αποθηκευτική δομή, που επιτρέπει να ελεγχθεί ταχύτατα εάν κάτι βρίσκεται αποθηκευμένο σε αυτό ή όχι με μηδενικά false negatives και μικρή πιθανότητα από false positives.

Κύριος στόχος της εργασίας είναι να εξετάσει εάν τα bloom filters μπορούν να προστατεύσουν αποτελεσματικά authoritative εξυπηρετητές από επιθέσεις DNS water torture. Ο μηχανισμός αυτός έχει τη δυνατότητα να προσαρμοστεί και σε recursive εξυπηρετητές ως μέρος ενός συνεργατικού σχήματος άμυνας κοντά στις πηγές της επίθεσης.

Λέξεις Κλειδιά: << Σύστημα Ονοματοδοσίας Τομέων, επίθεση DNS water torture, κατανεμημένες επιθέσεις άρνησης παροχής υπηρεσιών, Δίκτυα Οριζόμενα από Λογισμικό, μηχανική μάθηση, φίλτρο bloom >>

Abstract

Domain Name System (DNS) provides associations between hostnames and IP addresses. Therefore, its proper function is vital for the normal operation of the internet. Thus, it is a potential target of Distributed Denial of Service attacks (DDoS attacks).

A well-known and highly effective DDoS attack against DNS is water torture which targets the authoritative server of a DNS zone. The attacker floods the server with great volume of requests in order to exhaust its computation power and make it incapable of responding to the requests of legitimate clients. These requests contain random names which are not included in the zone files of the server. In this way, the DNS cache of recursive servers can be bypassed and the malicious messages will surely reach the victim.

In this thesis, a mechanism capable of detecting and mitigating such an attack was constructed in a Software Defined Networking (SDN) environment. Metrics regarding the server and the network are collected by a monitoring mechanism. Afterwards, an anomaly detection mechanism decides whether the network is in danger or not. In the former case, the SDN controller installs rules to the switch of the network, so that requests from IPs considered malicious are driven to a mitigation mechanism. Random requests are dropped and valid requests are forwarded to the server.

In particular, information is gathered by utilizing sampling and statistics collection software. Danger detection unit is based on a machine learning algorithm and the unit which filters the requests is based on a bloom filter cluster. A bloom filter is a space-efficient hashtable, that allows to quickly examine if an element is stored in it with zero false negatives and a small probability of false positives.

The main purpose of this thesis is to examine whether bloom filters can efficiently protect authoritative servers from DNS water torture attacks. This mechanism can also be adapted to recursive servers as part of a collaborative defence scheme near the source of the attack.

Keywords: << DNS, DNS water torture attack, Distributed Denial of Service attacks (DDoS), Software Defined Networks (SDN), machine learning, bloom filter >>

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

1 Εισαγωγή

1.1: Περιγραφή του προβλήματος

Η επικοινωνία ανάμεσα στους υπολογιστές του διαδικτύου προϋποθέτει εκείνοι να μπορούν να αναγνωριστούν μεταξύ τους. Για το σκοπό αυτό, οι υπολογιστές χρησιμοποιούν τις **διευθύνσεις IP**, οι οποίες αξιοποιούνται από τους δρομολογητές (routers) για την προώθηση των πακέτων στον προορισμό τους.

Ωστόσο, είναι πρακτικά αδύνατο για τους ανθρώπους να απομνημονεύουν μεγάλο πλήθος από διευθύνσεις IP. Είναι πολύ πιο εύκολο να συγκρατούν και να χρησιμοποιούν **ονόματα υπολογιστών (hostnames)**. Έτσι, οι υπολογιστές χρησιμοποιούν ονόματα, τα οποία παρέχουν πληροφορίες για τη λειτουργία που επιτελούν.

Κατά συνέπεια, απαιτείται ένας μηχανισμός, ο οποίος θα προσφέρει αντιστοίχιση διευθύνσεων IP σε hostnames και αντίστροφα. Ο μηχανισμός που αναλαμβάνει να πραγματοποιήσει αυτή τη λειτουργία είναι το **Σύστημα Ονοματοδοσίας Τομέων (Domain Name System, DNS)**.

Είναι εμφανές ότι η εξασφάλιση της ομαλής λειτουργίας του DNS είναι απαραίτητη για τη σωστή λειτουργία του διαδικτύου. Είναι φυσικό, λοιπόν, η υποδομή του DNS να αποτελεί, μεταξύ άλλων, υποψήφιο στόχο των **κατανεμημένων επιθέσεων άρνησης παροχής υπηρεσιών (Distributed Denial of Service attacks, DDoS)**. Τέτοιες επιθέσεις αποσκοπούν στο να καταστήσουν μια υπηρεσία μη προσβάσιμη στους πελάτες της, εξαντλώντας την με μεγάλο όγκο κίνησης από πολλαπλές πηγές. Πρόσφατη έρευνα της Akamai κατέδειξε ότι στους πρώτους τρεις μήνες του 2017, **το 20% των DDoS επιθέσεων στόχευαν το DNS** [1].

Χαρακτηριστικό παράδειγμα τέτοιας επίθεσης είναι η πρόσφατη **επίθεση στον πάροχο DNS Dyn** που πραγματοποιήθηκε στις 21 Οκτωβρίου 2016 και είχε ως επακόλουθο την αποκοπή πολύ μεγάλου αριθμού πελατών από σημαντικές διαδικτυακές υπηρεσίες. Ο όγκος της κίνησης, ο οποίος οδηγήθηκε στους εξυπηρετητές της Dyn έφτασε το πολύ μεγάλο μέγεθος των **1.2 Tbps** [2].

Ο κυριότερος τύπος επίθεσης που χρησιμοποιήθηκε [3, 4] ήταν η λεγόμενη **DNS water torture.** Η επίθεση αυτή πλημμυρίζει τον αρμόδιο (authoritative) εξυπηρετητή μιας ζώνης DNS με πολύ μεγάλο όγκο ερωτημάτων για να εξαντλήσει την υπολογιστική ισχύ του και να καταστεί ανίκανος να απαντά σε ερωτήματα νομίμων πελατών. Τα ερωτήματα αυτά περιλαμβάνουν ονόματα, τα οποία δημιουργούνται με τυχαίο τρόπο, επιτρέποντας, έτσι, την παράκαμψη της προσωρινής μνήμης (DNS cache) των αναδρομικών (recursive) DNS εξυπηρετητών. Η τεχνική αυτή εξασφαλίζει ότι το ερώτημα θα φτάσει σίγουρα στο θύμα.

1.2: Συνεισφορά εργασίας

Στα πλαίσια της παρούσης διπλωματικής εργασίας προτείνεται ένας μηχανισμός ανίχνευσης και αντιμετώπισης μιας επίθεσης DNS water torture σε περιβάλλον **Δικτύου Οριζόμενου από Λογισμικό (Software Defined Network, SDN)**. Ένα τέτοιο περιβάλλον διευκολύνει την καταπολέμηση επιθέσεων DDoS, καθώς προσφέρει ευελιξία στη διαχείριση των κακόβουλων ροών που εντοπίζονται. Ο μηχανισμός αυτός εξετάζει εάν ο εξυπηρετητής βρίσκεται σε κίνδυνο ή όχι και, στην πρώτη περίπτωση, λαμβάνει μέτρα για την προστασία του.

Αρχικά, πληροφορίες σχετικές με το δίκτυο και τον εξυπηρετητή συλλέγονται σε ένα κεντρικό σημείο:

- Το ποσοστό χρήσης του επεξεργαστή του authoritative εξυπηρετητή.

- Το ποσοστό των απαντήσεων DNS του εξυπηρετητή με κωδικό NXDOMAIN ως προς το πλήθος των ερωτήσεων DNS που καταφθάνουν σε αυτόν για ένα χρονικό διάστημα.

- Μέσω δειγματοληψίας με το πρότυπο **sFlow**, εντοπίζονται διευθύνσεις IP που αποστέλλουν ερωτήματα DNS για ανύπαρκτα ονόματα, τα οποία έχουν παραχθεί με τυχαίες διαδικασίες. Οι διευθύνσεις αυτές χαρακτηρίζονται ως επικίνδυνες.

Ένας αλγόριθμος **μηχανικής μάθησης (machine learning)** αξιοποιεί τις πληροφορίες αυτές και αποφασίζει εάν ο DNS server κινδυνεύει ή όχι. Η επιλογή ενός αλγορίθμου μηχανικής μάθησης επιτρέπει έναν πιο ευέλικτο καθορισμό των ορίων απόφασης του μηχανισμού, ενσωματώνοντας την εμπειρία του διαχειριστή, αλλά και δεδομένα από προηγούμενες περιπτώσεις παρατήρησης της επίθεσης.

Στην περίπτωση που θεωρηθεί ότι ο εξυπηρετητής βρίσκεται σε κίνδυνο, ο **ελεγκτής (controller)** SDN εγκαθιστά δυναμικά κανόνες στο μεταγωγέα (switch) του δικτύου, ώστε τα ερωτήματα από τις κακόβουλες IPs να οδηγούνται πρώτα σε ένα σύνολο από **bloom filters**. Το bloom filter είναι ένα αποδοτικό ως προς το χώρο αποθήκευσης hashtable, που επιτρέπει να ελεγχθεί ταχύτατα εάν κάτι βρίσκεται αποθηκευμένο σε αυτό ή όχι με μηδενικά false negatives και μικρή πιθανότητα από false positives. Αν το όνομα της ερώτησης βρίσκεται στο bloom filter, εκείνη προωθείται στον εξυπηρετητή DNS, αλλιώς απορρίπτεται.

Στόχος της εργασίας είναι να δείξει ότι τα bloom filters επιτρέπουν την αποτελεσματική αντιμετώπιση επιθέσεων DNS water torture και να πείσει για την υιοθέτηση μηχανισμών αντιμετώπισης βασισμένων σε bloom filters από authoritative DNS servers, αλλά και recursive DNS servers.

1.3: Οργάνωση εργασίας

Στην εργασία περιλαμβάνονται τα παρακάτω κεφάλαια:

- Στο **κεφάλαιο 2** παρουσιάζονται οι θεωρητικές γνώσεις που είναι απαραίτητες για την κατανόηση του μηχανισμού που υλοποιήθηκε.
- Στο **κεφάλαιο 3** περιγράφεται ο μηχανισμός που υλοποιήθηκε και η λειτουργία του.
- Στο **κεφάλαιο 4** αναλύονται λεπτομέρειες που αφορούν τις επιμέρους μονάδες που συνιστούν το μηχανισμό και επεξηγείται ο κώδικας της υλοποίησης.
- Στο **κεφάλαιο 5** παρουσιάζονται μετρήσεις και αποτελέσματα που σχετίζονται με την αξιολόγηση του μηχανισμού που κατασκευάστηκε.
- Στο **κεφάλαιο 6** περιλαμβάνονται τα συμπεράσματα της εργασίας και προτείνονται μελλοντικές επεκτάσεις του μηχανισμού.

2 Θεωρητικό Υπόβαθρο

Στην ενότητα αυτή, παρουσιάζεται το θεωρητικό υπόβαθρο που είναι απαραίτητο για την κατανόηση του μηχανισμού που υλοποιήθηκε στα πλαίσια της παρούσης διπλωματικής εργασίας.

2.1. Σύστημα Ονοματοδοσίας Τομέων (Domain Name System, DNS)

Το **DNS** αποτελεί, ουσιαστικά, την υπηρεσία καταλόγου του δημοσίου διαδικτύου [5]. Όπως ακριβώς, ένας τηλεφωνικός κατάλογος αντιστοιχίζει ονόματα ανθρώπων και τηλεφωνικούς αριθμούς, έτσι και το DNS είναι υπεύθυνο για την αντιστοίχιση **ονομάτων υπολογιστών (hostnames)** σε **διευθύνσεις IP (IP addresses)** και αντίστροφα.

Διαφαίνεται, λοιπόν, η ζωτική του σημασία για την απρόσκοπτη λειτουργία του διαδικτύου. Οποιαδήποτε βλάβη ή απρόσεχτη ρύθμιση στο DNS είναι ικανή να παρεμποδίσει την πρόσβαση μεγάλου αριθμού χρηστών στο διαδίκτυο, επιφέροντας σημαντικές ζημιές είτε σε οικονομικό επίπεδο είτε στην αξιοπιστία των διαχειριστών των δικτύων.

Κρίνεται, λοιπόν, απαραίτητο να διασφαλίζεται κάθε στιγμή η εύρυθμη λειτουργία του. Προκειμένου να κατανοήσουμε τους κινδύνους που απειλούν το DNS, πρέπει πρώτα να μελετηθεί ο τρόπος με τον οποίο λειτουργεί και η βασική ορολογία που σχετίζεται με αυτό.

Το σύστημα DNS περιλαμβάνει δύο βασικές συνιστώσες:

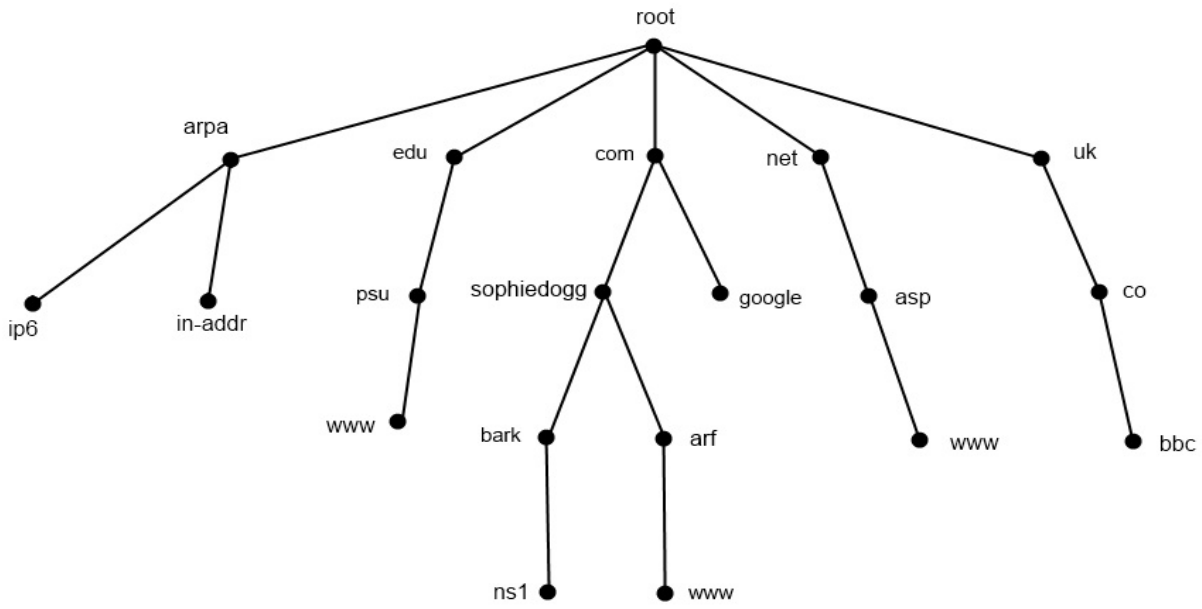
- μία κατανεμημένη, ιεραρχική **βάση δεδομένων**.
- το ομώνυμο **πρωτόκολλο** επιπέδου εφαρμογής για την επικοινωνία των υπολογιστών της ιεραρχίας μεταξύ τους.

2.1.1 Η ιεραρχία DNS

Το σύστημα DNS αποτελείται από ένα σύνολο **εξυπηρετητών DNS (DNS servers)**, οι οποίοι είναι λογικά κατανεμημένοι σε μια ιεραρχική δομή, όπως φαίνεται στο παρακάτω σχήμα (2.1), που ονομάζεται **ιεραρχία DNS**.

Η κατασκευή του DNS με τέτοιο τρόπο εξασφαλίζει γρήγορη αντιστοίχιση ονομάτων υπολογιστών και διευθύνσεων IP **(επίλυση ονόματος ή name resolution)** και εύκολη κλιμάκωση του DNS, καθώς το σύνολο της πληροφορίας διαμοιράζεται στους servers που συνιστούν το σύστημα. Το DNS, επίσης, είναι ένα εύρωστο (robust) σύστημα, καθώς δεν εξαρτάται από ένα μοναδικό σημείο αποτυχίας. Οι πληροφορίες που περιέχει ένας

εξυπηρετητής βρίσκονται κάθε στιγμή αποθηκευμένες και σε άλλους servers, οι οποίοι βρίσκονται κατανεμημένοι στην ιεραρχία DNS.

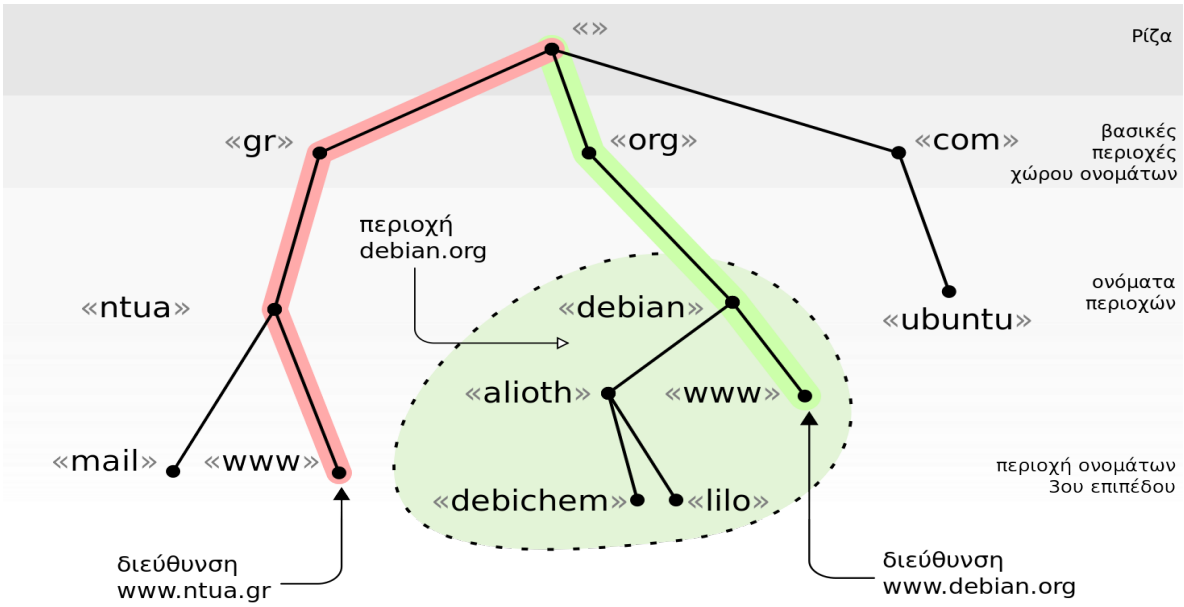


Σχήμα 2.1 - Η ιεραρχία DNS. Κάθε κόμβος της χαρακτηρίζεται από μία ετικέτα (label).

Κάθε κόμβος του δέντρου DNS χαρακτηρίζεται από μία **ετικέτα (label)**, δηλαδή ένα αλφαριθμητικό, όπως φαίνεται στο παραπάνω σχήμα (2.1). Το διαδίκτυο είναι νοητά χωρισμένο σε **περιοχές (domains)**, οι οποίες παριστάνονται ως ένα υποδέντρο του δέντρου DNS. Μία περιοχή είναι δυνατόν να περιέχει hosts ή να χωρίζεται σε περισσότερες **υποπεριοχές (subdomains)**.

Η ακριβής θέση ενός πόρου στην ιεραρχία DNS προσδιορίζεται από το **πλήρως πιστοποιημένο όνομα περιοχής (Fully Qualified Domain Name, FQDN)**. Το FQDN προκύπτει από την ένωση όλων των labels, ξεκινώντας από εκείνο του πόρου μέχρι τη ρίζα του δέντρου DNS με τελείες ανάμεσά τους. Η ρίζα του δέντρου DNS έχει επικρατήσει να συμβολίζεται είτε με κενό χαρακτήρα είτε με μία τελεία, η οποία προσκολλάται στο τέλος του FQDN.

Για παράδειγμα, στο παρακάτω σχήμα φαίνεται ο υπολογιστής με FQDN www.ntua.gr., το οποίο, όπως δείχνει η ροζ γραμμή, δημιουργείται από την ένωση των labels www, ntua, gr και . (ρίζα δέντρου DNS).



Σχήμα 2.2 – Διαχωρισμός της ιεραρχίας DNS σε περιοχές: η περιοχή debian.org. περιλαμβάνει τους υπολογιστές που βρίσκονται μέσα στο κυκλικό σχήμα. Η περιοχή alioth.debian.org. αποτελεί subdomain του domain debian.org.

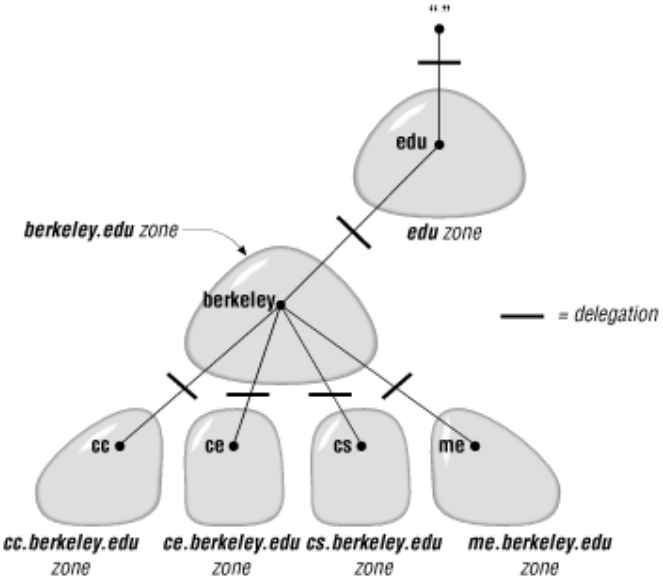
Ωστόσο, πρέπει να επισημανθεί ότι η διαχείριση των **εγγραφών DNS (DNS records)**, δηλαδή των αντιστοιχίσεων μεταξύ ονομάτων υπολογιστικών πόρων και διευθύνσεων IP (ή άλλων πληροφοριών σε ορισμένες περιπτώσεις), στους DNS servers, δε γίνεται με βάση τα domains, αλλά τις **ζώνες (zones)**. Μία ζώνη είναι ένα υποσύνολο του δέντρου DNS για την οποία ένας DNS server είναι υπεύθυνος να απαντάει με βεβαιότητα για τις εγγραφές που περιλαμβάνει. Ένας τέτοιος server ονομάζεται **authoritative DNS server**.

Μία ζώνη περιλαμβάνει ένα σύνολο από authoritative εξυπηρετητές DNS που διατηρούν την ίδια πληροφορία. Ο εξυπηρετητής ο οποίος αναλαμβάνει να απαντήσει στα ερωτήματα DNS ονομάζεται **πρωταρχικός εξυπηρετητής (primary server)**. Οι υπόλοιποι ονομάζονται **δευτερεύοντες εξυπηρετητές (secondary servers)** και εξυπηρετούν ερωτήματα στην περίπτωση που συμβεί κάποια βλάβη στον πρωταρχικό εξυπηρετητή. Τα περιεχόμενα του αρχείου ζώνης του πρωταρχικού εξυπηρετητή μεταφέρονται στους δευτερεύοντες εξυπηρετητές με ενέργειες που ονομάζονται **μεταφορές ζώνης (zone transfers)**.

Σκοπός της διαίρεσης μιας περιοχής σε πολλές ζώνες είναι η αποτελεσματικότερη διαχείριση της περιοχής με την απόδοση της διαχείρισης των ζωνών σε διαφορετικά τμήματα μιας οργάνωσης (zone delegation), ο διαμοιρασμός της κίνησης σε περισσότερους DNS servers και η ταχύτερη επίλυση ονομάτων.

Για παράδειγμα, στην παρακάτω εικόνα (2.3) φαίνονται 6 ζώνες διαχείρισης εγγραφών DNS, καθεμία από τις οποίες προσδιορίζεται με ένα κλειστό σχήμα. Καθεμία από τις ζώνες

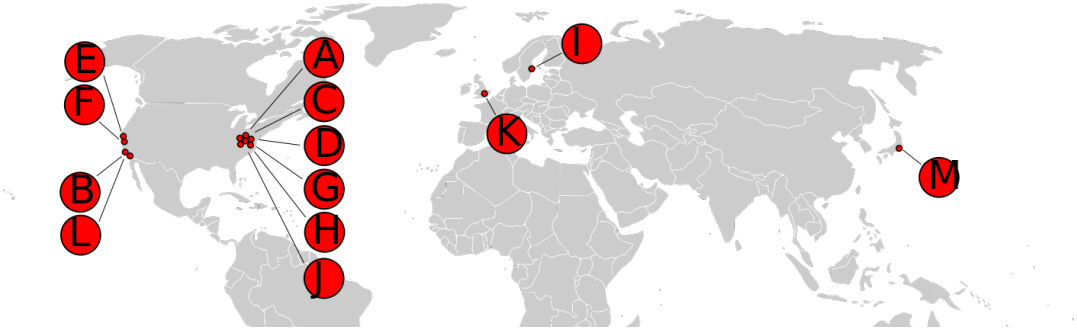
αυτές έχει αποκλειστική κυριότητα και ευθύνη για τη διαχείριση των εγγραφών που περιλαμβάνει.



Σχήμα 2.3 – Διαίρεση της ιεραρχίας DNS σε ζώνες για τη διαχείριση των εγγραφών DNS

Συνεπώς, μπορούμε να διακρίνουμε τα παρακάτω είδη περιοχών DNS, ανάλογα με τη θέση τους στην ιεραρχία, καθώς και τους εξυπηρετητές DNS που σχετίζονται με αυτές:

- **Root domain**: βρίσκεται στην κορυφή του δέντρου DNS. Η περιοχή αυτή περιλαμβάνει τους root DNS servers, οι οποίοι είναι υπεύθυνοι να γνωρίζουν τις διευθύνσεις των αρμοδίων εξυπηρετητών για τα labels του πρώτου επιπέδου της ιεραρχίας DNS (TLD εξυπηρετητές). Αυτά είναι τα labels που βρίσκονται στη δεξιότερη θέση των FQDNs. Οι root DNS servers είναι 13 στον αριθμό και είναι παγκόσμια κατανεμημένοι, με τους περισσότερους να βρίσκονται στη Βόρεια Αμερική. Οι εξυπηρετητές αυτοί φαίνονται στο παρακάτω σχήμα (2.4):



Σχήμα 2.4 – Οι 13 root DNS servers. Καθένας συμβολίζεται με ένα γράμμα του λατινικού αλφαβήτου από το a μέχρι το m.

Είναι απαραίτητο να επισημανθεί ότι, στην πραγματικότητα, οι εξυπηρετητές

αυτοί δεν είναι μόνο 13, αλλά κάθε εξυπηρετητής αποτελείται από ένα σύμπλεγμα εξυπηρετητών με κοινή διεύθυνση IP. Ο καταλληλότερος εξυπηρετητής του συμπλέγματος επιλέγεται με γεωγραφικά κριτήρια, δηλαδή ποιος εξυπηρετητής βρίσκεται κοντύτερα στον υπολογιστή που διατυπώνει το ερώτημα DNS (πρωτόκολλο **anycast**).

- **Top-Level Domains (TLDs)**: περιλαμβάνει τα domains του πρώτου επιπέδου του δέντρου DNS, όπως είναι τα com, net, gr, fr, org. Κάθε label δίνει πληροφορία για τη χώρα ή την οργάνωση που χρησιμοποιεί το όνομα αυτό. Οι εξυπηρετητές TLD είναι υπεύθυνοι να γνωρίζουν τις διευθύνσεις των authoritative servers του αμέσως κατώτερου στην ιεραρχία επιπέδου.

- **Second-level domains**: τα domains του δευτέρου επιπέδου της ιεραρχίας DNS.

2.1.2: Παράδειγμα επίλυσης ονόματος

Ο τρόπος λειτουργίας του DNS θα γίνει ξεκάθαρος μέσα από ένα παράδειγμα επίλυσης ονόματος. Έστω ότι ένας χρήστης θέλει να μάθει την διεύθυνση IP του υπολογιστή www.example.com. Θεωρούμε ότι ο authoritative server που είναι υπεύθυνος για τη ζώνη example.com γνωρίζει την IP του υπολογιστή αυτού. Όπως φαίνεται στο παρακάτω σχήμα (2.5), θα ακολουθηθεί η παρακάτω διαδικασία:

1) Ο υπολογιστής θα στείλει το αίτημά του στον server RDNS, ο οποίος είναι ένας **αναδρομικός (recursive)** DNS server και ονομάζεται **recursor** (ή **resolver**). Ο εξυπηρετητής αυτός δεν είναι αρμόδιος να διατηρεί εγγραφές DNS για κάποια ζώνη διαχείρισης και να απαντάει σε ερωτήσεις για αυτές. Ένας recursor αναλαμβάνει τη διαδικασία αναζήτησης της εγγραφής DNS που του ζητήθηκε μέχρι να την πάρει από τον authoritative εξυπηρετητή που είναι υπεύθυνος για αυτήν και να την επιστρέψει στον υπολογιστή που διατύπωσε το ερώτημα.

2) Έτσι, ο recursor στέλνει αίτημα στον root DNS server που βρίσκεται πλησιέστερα, ρωτώντας τον ποια είναι η διεύθυνση IP του υπολογιστή με FQDN www.example.com. Ο root DNS server είναι υπεύθυνος να απαντά έως ένα label βάθος στην ιεραρχία DNS, δηλαδή να γνωρίζει πληροφορίες μόνο για το Top-Level Domain της ερώτησης. Έτσι, θα αγνοήσει το πρώτο τμήμα του FQDN και θα αναζητήσει στα αρχεία του ποιος είναι ο TLD DNS server που είναι υπεύθυνος για τη ζώνη com και τη διεύθυνση IP του.

3) Ο root DNS server επιστρέφει στον recursor ποιος είναι ο TLD DNS server που είναι αρμόδιος για τη ζώνη com και τη διεύθυνση IP του.

4) Στη συνέχεια, ο recursor στέλνει αίτημα στον TLD DNS server που είναι υπεύθυνος για τη ζώνη com, ρωτώντας τον ποια είναι η IP του υπολογιστή με FQDN

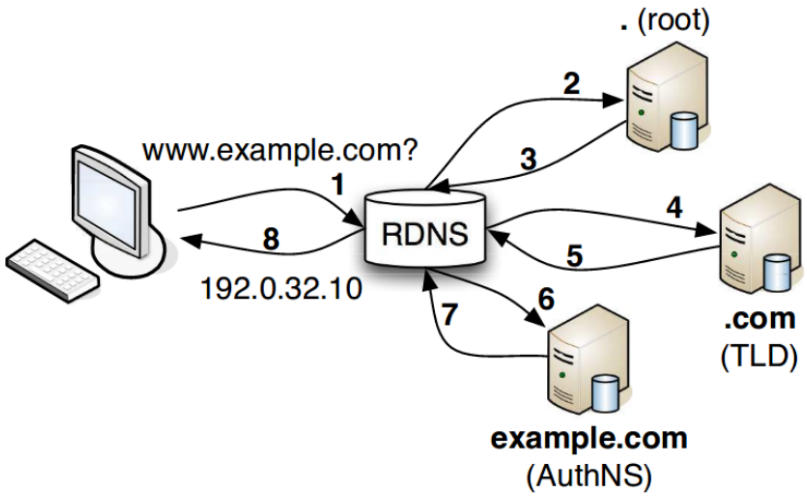
www.example.com. Ο TLD server, όμως, είναι υπεύθυνος να απαντάει μέχρι δύο labels βάθος στην ιεραρχία DNS. Συνεπώς, θα αγνοήσει το πρώτο τμήμα του FQDN και θα αναζητήσει στα αρχεία του ποιος είναι ο authoritative DNS server που είναι υπεύθυνος για τη ζώνη example.com και τη διεύθυνση IP του.

5) Ο TLD DNS server επιστρέφει στον recursor ποιος είναι ο authoritative DNS server που περιέχει τις εγγραφές για τη ζώνη example.com και τη διεύθυνση IP του.

6) Ο recursor στέλνει αίτημα στον authoritative DNS server της ζώνης example.com, ρωτώντας ποια είναι η διεύθυνση IP του υπολογιστή με FQDN www.example.com.

7) Ο authoritative DNS server ξέρει ότι είναι υπεύθυνος για τη ζώνη example.com, θα αναζητήσει στα αρχεία του την αντιστοίχιση του FQDN και της διεύθυνσης IP και θα επιστρέψει στον recursor τη διεύθυνση IP του υπολογιστή για τον οποίο ρώτησε.

8) Τέλος, ο recursor επιστρέφει τη διεύθυνση IP στον υπολογιστή από τον οποίο ξεκίνησε η διαδικασία.



Σχήμα 2.5 – Παράδειγμα επαναληπτικού τρόπου επίλυσης μιας ερώτησης DNS.

Επισημαίνεται ότι η παραπάνω διαδικασία ονομάζεται **επαναληπτική** (iterative) επίλυση ερωτήματος. Η αναζήτηση ενός ονόματος μπορεί να γίνει και με **αναδρομικό** (recursive) τρόπο. Σε αυτήν την περίπτωση, ο κάθε server ζητάει από τον αμέσως κατώτερο στην ιεραρχία server να αναλάβει εκείνος την αναζήτηση της εγγραφής εκ μέρους του, όπως ακριβώς λειτουργεί και ο recursor για τον υπολογιστή που διατυπώνει το ερώτημα DNS. Ωστόσο, ο τρόπος που εφαρμόζεται, συνήθως, στην πράξη είναι ο επαναληπτικός τρόπος. Σε κάθε περίπτωση, όμως, ο αρχικός υπολογιστής διατυπώνει ένα αναδρομικό ερώτημα στον recursor.

Ένα δίκτυο είναι ελεύθερο να χρησιμοποιήσει είτε το recursor ενός παρόχου διαδικτύου είτε να χρησιμοποιήσει recursors που τους διαχειρίζονται άλλοι οργανισμοί και είναι ρυθμισμένοι να εξυπηρετούν τα αναδρομικά ερωτήματα DNS οποιουδήποτε υπολογιστή.

Στην τελευταία περίπτωση, οι εξυπηρετητές αυτοί ονομάζονται **open resolvers**.

Ένας open resolver μπορεί να αποτελέσει εναλλακτική επιλογή ενός χρήστη, όταν ο recursor που του παρέχεται από τον DNS πάροχό του έχει υποστεί κάποια βλάβη ή δεν τον ικανοποιεί. Ωστόσο, το γεγονός ότι ένας open resolver εξυπηρετεί αναδρομικά DNS ερωτήματα από **κάθε** χρήστη γεννά σημαντικούς κινδύνους, οι οποίοι θα παρουσιαστούν σε επόμενα κεφάλαια.

2.1.3: Προσωρινή αποθήκευση αντιστοιχίσεων DNS (DNS caching)

Όπως είναι φυσικό, η διαδικασία αναζήτησης της αντιστοίχισης ενός ονόματος υπολογιστή σε μία διεύθυνση IP προκαλεί σημαντική καθυστέρηση και είναι σημαντικό να επαναλαμβάνεται όσο το δυνατόν λιγότερες φορές. Έτσι, είναι επιθυμητό, εάν χρειαστεί ξανά η εγγραφή αυτή σε σύντομο χρονικό διάστημα, νη μην επαναληφθεί η χρονοβόρα διαδικασία αναζήτησής της.

Για να καταστεί δυνατό αυτό, ο recursive εξυπηρετητής αποθηκεύει τις απαντήσεις στις αναζητήσεις που πραγματοποιεί σε μια προσωρινή μνήμη (DNS cache) για χρονικό διάστημα που καθορίζεται από μία παράμετρο, η οποία ονομάζεται **TTL (Time To Live)**. Η τεχνική αυτή ονομάζεται **DNS caching**. Όταν λήξει το TTL, η αποθηκευμένη εγγραφή αφαιρείται από την προσωρινή μνήμη, γιατί, πλέον, θεωρείται αναξιόπιστη, αφού είναι πιθανό η αντιστοίχιση ονόματος και διεύθυνσης να έχει αλλάξει στη διάρκεια του χρονικού διαστήματος μη αναζήτησης της αποθηκευμένης εγγραφής που υπαγορεύτηκε από το TTL.

Επομένως, το προηγούμενο παράδειγμα εμπλουτίζεται με ένα ακόμη βήμα. Πριν ο recursive εξυπηρετητής απευθυνθεί σε έναν root DNS server, θα ελέγξει την DNS cache του για να διαπιστώσει αν έχει αποθηκευμένη ήδη την επιθυμητή αντιστοίχιση. Εάν η αντιστοίχιση βρίσκεται ήδη στη μνήμη του, θα την επιστρέψει αμέσως στον υπολογιστή που διατύπωσε το ερώτημα. Εάν όχι, θα συνεχίσει κανονικά τη διαδικασία αναζήτησης, όπως εκείνη περιγράφτηκε στην προηγούμενη ενότητα. Όταν η απάντηση επιστραφεί στον recursor, εκείνος θα την αποθηκεύσει στην προσωρινή του μνήμη για να την επιστρέψει αμέσως αν αναζητηθεί ξανά σε σύντομο χρονικό διάστημα.

2.1.4: Τύποι εγγραφών DNS

Εγγραφή DNS (Resource Record ή RR) είναι μια απεικόνιση μεταξύ ενός ονόματος υπολογιστικού πόρου (FQDN) και αγαθών (resources) διαφόρων τύπων. Οι εγγραφές DNS αποθηκεύονται στα αρχεία ζώνης των authoritative εξυπηρετητών DNS.

Υπάρχουν πολλοί διαφορετικοί τύποι εγγραφών DNS. Ενδεικτικά, κάποιοι από αυτούς παρουσιάζονται παρακάτω:

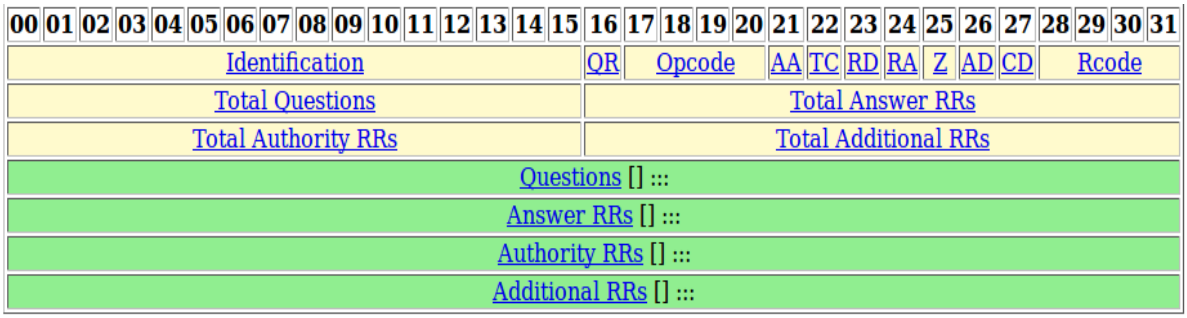
- ***A*:** χρησιμοποιείται για την αντιστοίχιση ενός FQDN σε μια διεύθυνση IP των 32

bits.
- ***AAAA:*** χρησιμοποιείται για την αντιστοίχιση ενός FQDN σε μια διεύθυνση IPv6 των 128 bits.
- ***ANY:*** επιστρέφει όλες τις εγγραφές που γνωρίζει ο nameserver που ερωτάται.
- ***NS:*** οι εγγραφές αυτές προσδιορίζουν τους εξυπηρετητές DNS που είναι authoritative για μία ζώνη.
- ***MX:*** προσδιορίζει τους mail servers μιας ζώνης DNS.
- ***PTR:*** χρησιμοποιούνται για την αντιστοίχιση μιας διεύθυνσης IP σε ένα FQDN, δηλαδή την αντίστροφη διαδικασία της επίλυσης ενός ονόματος.
- ***SOA*** (Start Of Authority): περιλαμβάνει πληροφορίες σχετικές με το ποιος είναι ο primary εξυπηρετητής της ζώνης, ποιος είναι ο διαχειριστής της ζώνης, πώς μπορεί κάποιος να επικοινωνήσει μαζί του, ποιο είναι το προκαθορισμένο TTL των εγγραφών και άλλες παραμέτρους που σχετίζονται με τις εγγραφές της ζώνης.

Από τις παραπάνω εγγραφές, σοβαρό κίνδυνο μπορεί να δημιουργήσει η εγγραφή **ANY**, αφού το μέγεθος της απάντησης που επιστρέφει είναι, συνήθως, πολύ μεγάλο. Έτσι, όπως θα δούμε στη συνέχεια, χρησιμοποιείται συχνά σε επιθέσεις amplification.

2.1.5: Το πρωτόκολλο DNS

Η επικοινωνία των υπολογιστών της ιεραρχίας DNS επιτυγχάνεται με το **πρωτόκολλο DNS**. Όταν ένας υπολογιστής ζητάει μια εγγραφή DNS, διατυπώνει ένα ερώτημα DNS (**DNS request**), ενώ όταν δέχεται απάντηση σε αυτό, λαμβάνει μία απάντηση DNS (**DNS response**). Η επικεφαλίδα του πρωτοκόλλου DNS παρουσιάζεται παρακάτω:



Σχήμα 2.6 – Η επικεφαλίδα του μηνύματος DNS

Τα σημαντικότερα πεδία της επικεφαλίδας ενός μηνύματος DNS επεξηγούνται στη συνέχεια [6]:

- **Identification** (16 bits): αποτελεί την ταυτότητα του μηνύματος. Το πεδίο αυτό του ερωτήματος DNS αντιγράφεται στην απόκριση DNS, ώστε να μπορεί να γίνει η αντιστοίχιση ανάμεσα στην ερώτηση και στην απάντηση που της αντιστοιχεί.
- **QR** (1 bit) – Query/Response: δηλώνει εάν το μήνυμα DNS είναι ερώτημα (τίθεται

ίσο με 0) ή απάντηση (τίθεται ίσο με 1).
- **Opcode** (4 bits): περιγράφει τον τύπο της ερώτησης του μηνύματος.
- **AA** (1 bit) – Authoritative Answer: δηλώνει εάν ο εξυπηρετητής που απάντησε στο ερώτημα DNS ήταν authoritative για το ερώτημα αυτό. Τίθεται ίσο με 0 εάν ο εξυπηρετητής αυτός δεν ήταν authoritative, αλλιώς τίθεται ίσο με 1.
- **TC** (1 bit) – Truncated: δηλώνει ότι μόνο τα πρώτα 512 bytes της απάντησης επιστράφηκαν στον υπολογιστή που διατύπωσε το ερώτημα. Τίθεται ίσο με 0 αν επιστράφηκε όλο το μήνυμα, αλλιώς τίθεται ίσο με 1.
- **RD** (1 bit) – Recursion Desired: τίθεται από τον υπολογιστή που διατυπώνει το ερώτημα DNS και δηλώνει ότι επιθυμεί την αναδρομική επίλυση του ερωτήματος. Τίθεται ίσο με 0 αν η αναδρομή δεν είναι επιθυμητή, αλλιώς τίθεται ίσο με 1.
- **RA** (1 bit) – Recursion Available: τίθεται από τον εξυπηρετητή που απαντάει στο DNS ερώτημα και δηλώνει εάν υποστηρίζεται η αναδρομική επίλυση του ερωτήματος από τον εξυπηρετητή. Τίθεται ίσο με 0 εάν δεν υποστηρίζεται η αναδρομή, αλλιώς τίθεται ίσο με 1 αν υποστηρίζεται.
- **Rcode** (4 bits): στην περίπτωση του ερωτήματος DNS τίθεται ίσο με 0. Στην περίπτωση της απάντησης DNS τίθεται ίσο με έναν αριθμό από το 0 μέχρι το 15. Ενδεικτικά, κάποιες τιμές του έχουν τις εξής ερμηνείες [7]:

Rcode value	**Rcode**	**Rcode περιγραφή**
0	No Error	Κανένα λάθος
1	Format Error	Ο server δεν μπόρεσε να απαντήσει στο ερώτημα γιατί η δομή του ήταν ακατάλληλη
2	Server Failure (SRVFAIL)	Ο server δεν μπόρεσε να απαντήσει στο ερώτημα λόγω προβλημάτων που αντιμετώπιζε ο ίδιος
3	Name Error (NXDOMAIN)	Ο server δεν μπόρεσε να απαντήσει στο ερώτημα γιατί δε βρήκε καμία αντιστοίχιση στα αρχεία του
4	Not implemented	Ο server δεν μπόρεσε να απαντήσει στο ερώτημα γιατί δεν υποστήριζε τον τύπο του ερωτήματος
5	Refused	Ο server δε θέλησε να απαντήσει στο ερώτημα, όχι λόγω βλάβης ή αδυναμίας, αλλά λόγω κάποιας πολιτικής

Πίνακας 2.1 – Ενδεικτικά rcodes μίας απάντησης DNS

- **Total Questions** (16 bits): προσδιορίζει τον αριθμό των ερωτήσεων που τέθηκαν.
- **Total Answer RRs** (16 bits): προσδιορίζει τον αριθμό των απαντήσεων που επιστράφηκαν από τον server.
- **Total Authority RRs** (16 bits): προσδιορίζει τον αριθμό των Resource Records που επιστράφηκαν από authoritative nameserver.
- **Total Additional RRs** (16 bits): προσδιορίζει τον αριθμό των επιπρόσθετων Resource Records που επιστράφηκαν στην απάντηση.
- **Questions** (μεταβλητό μήκος): οι ερωτήσεις
- **Answer RRs** (μεταβλητό μήκος): οι απαντήσεις
- **Authority RRs** (μεταβλητό μήκος): οι authoritative απαντήσεις
- **Additional RRs** (μεταβλητό μήκος): τα επιπρόσθετα Resource Records

2.2: Κατανεμημένες επιθέσεις άρνησης παροχής υπηρεσιών

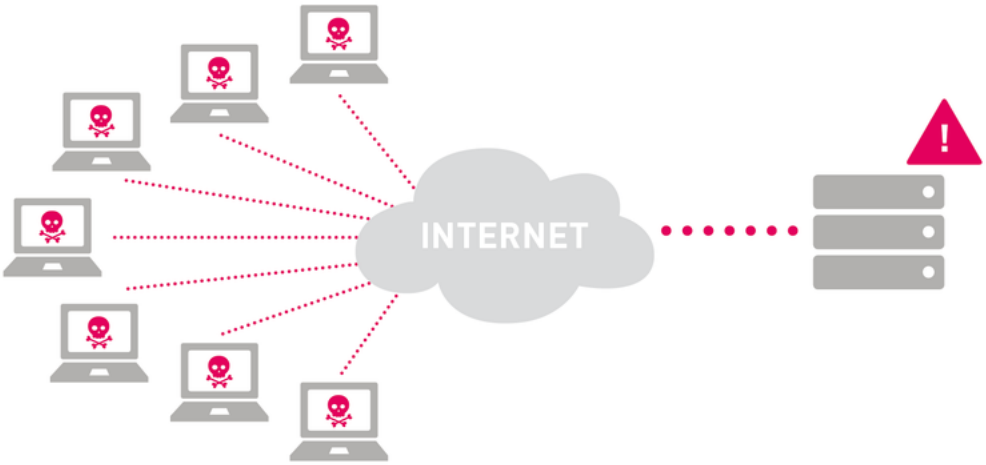
Η μεγαλύτερη απειλή για το σύγχρονο διαδίκτυο εντοπίζεται στις DoS και στις DDoS επιθέσεις, τα κυριότερα σημεία των οποίων αναλύονται στις επόμενες παραγράφους.

2.2.1: Επιθέσεις DoS και DDoS – Ορισμός και σκοπός

Μία επίθεση άρνησης παροχής υπηρεσιών (**Denial of Service, DoS**) είναι μια επίθεση που αποσκοπεί στο να αποκόψει νόμιμους χρήστες του διαδικτύου από ένα δικτυακό πόρο είτε προσωρινά είτε για μεγάλο χρονικό διάστημα. Μία επίθεση DoS έχει αφετηρία **έναν** υπολογιστή και πλημμυρίζει το θύμα με πολύ μεγάλο όγκο κίνησης, στοχεύοντας στην κατασπατάληση των πόρων του (επεξεργαστής, φυσική μνήμη, κτλ.). Το θύμα απασχολείται με ανούσιες εργασίες, αφού καλείται να επεξεργαστεί όλα τα μηνύματα που δέχεται από τον επιτιθέμενο και αδυνατεί να εξυπηρετήσει τους πελάτες του, απαντώντας στα μηνύματά τους.

Σχήμα 2.7 – Denial of Service (DoS) attack

Συνεπώς, οι επιθέσεις DoS δε συνίστανται στην εγκατάσταση κακόβουλου κώδικα στο θύμα, αλλά στην εκμετάλλευση τρωτών σημείων των πρωτοκόλλων επικοινωνίας που χρησιμοποιούνται και στην αδύναμη υποδομή του θύματος (περιορισμένη φυσική μνήμη, επεξεργαστική ισχύς, κλπ).



Σχήμα 2.8 – Distributed Denial of Service (DDoS) attack

Όταν η επίθεση δεν έχει σημείο εκκίνησης έναν υπολογιστή, αλλά πολλούς υπολογιστές, που βρίσκονται κατανεμημένοι σε διαφορετικά σημεία του διαδικτύου και προχωρούν σε συντονισμένη επίθεση, ονομάζεται **κατανεμημένη επίθεση DoS (distributed DoS, DDoS)**. Αξιοποιώντας τη δύναμη μεγάλου αριθμού υπολογιστών, μια επίθεση DDoS είναι πάρα πολύ δυνατότερη από μία απλή επίθεση DoS και μπορεί να στοχεύει στην κατασπατάληση του εύρους ζώνης (bandwidth) των ζεύξεων του δικτύου του θύματος, αλλά και στην ταχύτερη εξάντληση των πόρων του.

2.2.2: IP Spoofing

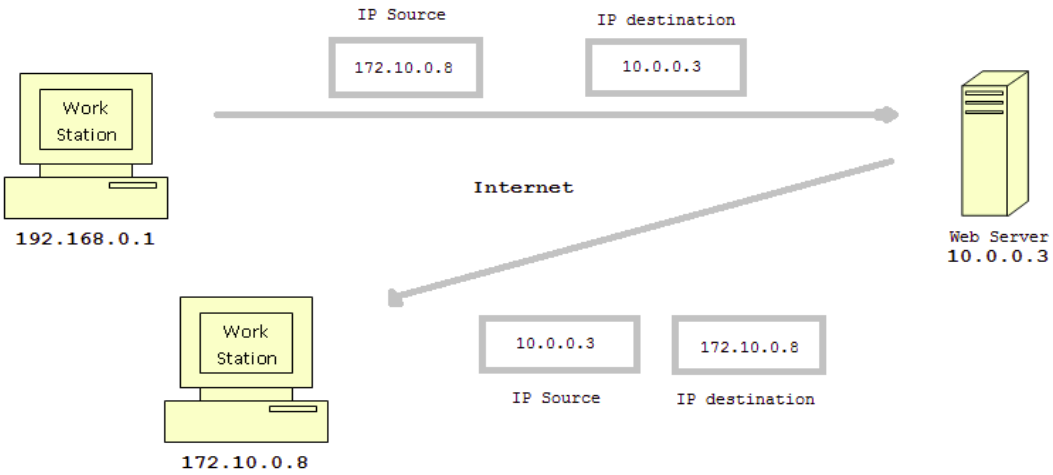
Στις περισσότερες επιθέσεις DDoS, ο επιτιθέμενος κρίνει σκόπιμο να παραλλάξει τη διεύθυνση IP του και να παραστήσει ότι είναι κάποιος άλλος υπολογιστής, χρησιμοποιώντας μια διαφορετική διεύθυνση IP αντί για τη δική του. Η μέθοδος αυτή ονομάζεται **IP spoofing** [8] και το κίνητρο του επιτιθέμενου είναι συνήθως ένα από τα παρακάτω ή και όλα μαζί:

- ***Απόκρυψη ταυτότητας:*** Ο επιτιθέμενος θέλει να αποκρύψει όσο καλύτερα γίνεται την ταυτότητά του από το θύμα για να μην εντοπιστεί. Σε περίπτωση που ο επιτιθέμενος κρατήσει την πραγματική του IP, τότε το θύμα σύντομα θα αναγνωρίσει από πού προέρχεται η επίθεση. Έτσι, θα την εξουδετερώσει εύκολα και θα ενημερώσει τις αρχές για την κακόβουλη δραστηριότητα του επιτιθέμενου.

- ***Reflection:*** Ο επιτιθέμενος δε θέλει να επιτεθεί άμεσα στο θύμα, αλλά έμμεσα. Για να το πετύχει αυτό, εκμεταλλεύεται ακατάλληλα ρυθμισμένους υπολογιστές που γνωρίζει με απόλυτη σιγουριά ότι θα απαντήσουν στα μηνύματά του. Ο επιτιθέμενος, θέτει ως διεύθυνση IP του την IP του θύματος. Έτσι, καταφέρνει να κάνει τον υπολογιστή στον οποίο θα στείλει το μήνυμά του να στείλει, ξεγελασμένος, την απάντησή του στο θύμα. Η τεχνική αυτή ονομάζεται

κατοπτρισμός (reflection).

- ***Αποφυγή απάντησης***: Ο επιτιθέμενος δε θέλει να δεχτεί απάντηση από το θύμα. Εάν ο επιτιθέμενος δέχεται απάντηση για κάθε μήνυμα που συνιστά την επίθεσή του, τότε στην πραγματικότητα θα δεχτεί και εκείνος τις συνέπειες της επίθεσης που εκτέλεσε.



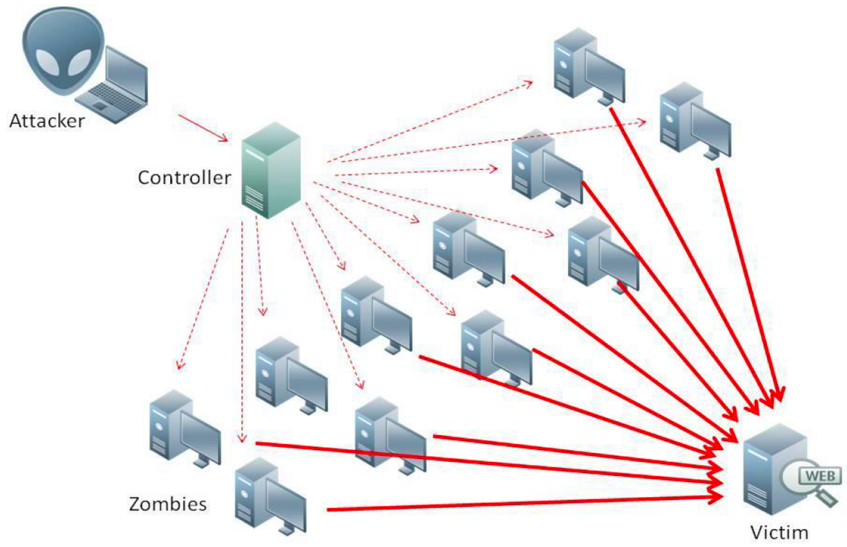
Σχήμα 2.9 - IP spoofing και reflection: ο υπολογιστής 192.168.0.1 υποδύεται τον υπολογιστή 172.10.0.8 και στέλνει μήνυμα στον server 10.0.0.3. Ως αποτέλεσμα, ο 10.0.0.3 θα στείλει την απάντηση στο μήνυμα στον πραγματικό υπολογιστή με διεύθυνση IP 172.10.0.8.

2.2.3: Botnet

Μία πολύ σημαντική παράμετρος που καθορίζει τη δύναμη μιας επίθεσης DDoS είναι ο αριθμός των υπολογιστών που θα χρησιμοποιήσει ο επιτιθέμενος εναντίον του θύματος. Μία πολύ ισχυρή επίθεση DDoS χρησιμοποιεί εκατοντάδες χιλιάδες υπολογιστές για να επιτύχει το στόχο της.

Είναι, φυσικά, αδύνατο ο επιτιθέμενος να έχει στην κατοχή του έναν τόσο μεγάλο αριθμό υπολογιστών. Έτσι, στρατολογεί ένα δίκτυο από κατανεμημένους σε όλο το διαδίκτυο υπολογιστές, τον έλεγχο των οποίων μπορεί και διατηρεί παρά τη θέλησή τους από μία απομακρυσμένη τοποθεσία. Ο επιτιθέμενος αποκτά τον έλεγχο των υπολογιστών αυτών, χρησιμοποιώντας κακόβουλο λογισμικό, όπως σκουλήκια (worms) και Trojan horses, εκμεταλλευόμενος κενά στην ασφάλειά τους. Οι υπολογιστές αυτοί αποκαλούνται υπολογιστές **zombie** ή **bots** και το δίκτυο που σχηματίζουν ονομάζεται **botnet** [9, 10, 11].

Την κατάλληλη στιγμή, ο επιτιθέμενος στέλνει μήνυμα στον **controller** μέσω του οποίου διατηρεί τον έλεγχο του botnet και εκείνος δίνει εντολές στους υπολογιστές zombie για να ξεκινήσει μία συγχρονισμένη επίθεση DDoS. Το botnet δεν πολλαπλασιάζει μόνο τη δύναμη του του επιτιθέμενου, αλλά αποκρύπτει ακόμα περισσότερο την ταυτότητά του, καθώς οι υπολογιστές zombie χρησιμοποιούν IP spoofing στις διευθύνσεις τους.



Σχήμα 2.10 – Επίθεση DDoS με χρήση botnet

2.2.4: Είδη επιθέσεων DDoS

Οι επιθέσεις DDoS μπορούν να χωριστούν σε κατηγορίες [11, 12, 13] ανάλογα με τον τύπο και την ποσότητα της κίνησης που χρησιμοποιεί η επίθεση, καθώς και το ποια αδυναμία του θύματος προσπαθούν να εκμεταλλευτούν για να πετύχουν το στόχο τους. Οι κατηγορίες επιθέσεων DDoS που μπορούμε να διακρίνουμε είναι **τρεις** και αναλύονται παρακάτω:

- ***Επίθεση εναντίον του εύρους ζώνης (volumetric/bandwidth attack):*** τέτοιες επιθέσεις έχουν στόχο να κατασπαταλήσουν το εύρος ζώνης του δικτύου στο οποίο βρίσκεται το θύμα, δηλαδή να προκαλέσουν συμφόρηση (congestion). Έτσι, θα εμποδίσουν τη μεταφορά μηνυμάτων από και προς τον εξυπηρετητή, καθώς θα απορρίπτονται λόγω των πλημμυρισμένων ζεύξεων. Οι επιθέσεις αυτές μετριούνται, συνήθως, σε bits/sec.

- ***Επίθεση στο πρωτόκολλο (protocol attacks):*** τέτοιες επιθέσεις έχουν ως στόχο την κατασπατάληση υπολογιστικών πόρων, όπως είναι η χρήση του επεξεργαστή, η διαθέσιμη μνήμη, ο αριθμός των διαθέσιμων sockets, ο αριθμός προσβάσεων στο δίσκο, κ.λ.π. Η επίθεση αυτή μπορεί να έχει στόχο να πλήξει είτε τον ίδιο τον server είτε ενδιαμέσες συσκευές, όπως είναι τα τείχη προστασίας (firewalls) και οι εξισορροπητές φόρτου (load balancers). Για παράδειγμα, αυτό μπορεί να επιτευχθεί εξαντλώντας τον αριθμό των συνδέσεων TCP που μπορεί να ανοίξει ταυτόχρονα μια συσκευή. Οι επιθέσεις αυτές μετριούνται, συνήθως, σε packets/sec.

- ***Επίθεση κατά της υπηρεσίας (application attacks):*** τέτοιες επιθέσεις αξιοποιούν αδυναμίες στα πρωτόκολλα επιπέδου εφαρμογής και έχουν ως στόχο να πλήξουν τους πόρους του εξυπηρετητή. Ιδιαίτερο χαρακτηριστικό των επιθέσεων αυτών είναι

ότι χρησιμοποιούν νόμιμα και φαινομενικά ακίνδυνα ερωτήματα. Οι επιθέσεις αυτές μετριούνται σε requests/sec.

2.2.5: Συνέπειες και Κίνητρα επιθέσεων DoS/DDoS

Οι συνέπειες μιας επιτυχημένης επίθεσης DoS/DDoS είναι πολύ μεγάλες για τον οργανισμό που διαχειρίζεται το θύμα και αφορούν στους παρακάτω τομείς [14]:

- **απώλεια εσόδων:** πολλές επιχειρήσεις σήμερα λειτουργούν ηλεκτρονικά. Τα έσοδά τους στηρίζονται αποκλειστικά στις αγορές που πραγματοποιούν online οι χρήστες. Οι επιχειρήσεις αυτές έχουν ολοκληρωτική απώλεια εσόδων σε μια επίθεση DDoS.

- **απώλεια παραγωγικότητας:** η εργασία στις σύγχρονες επιχειρήσεις απαιτεί την πρόσβαση των εργαζομένων στο διαδίκτυο, σε απομακρυσμένους εξυπηρετητές, σε υπηρεσίες συννέφου (cloud) και άλλες πολύτιμες δικτυακές υπηρεσίες. Έτσι, σε μια DDoS, οι υπάλληλοι μιας επιχείρισης αδυνατούν να εργαστούν σωστά και πλήττεται η παραγωγικότητα της επιχείρισης.

- **οικονομική επιβάρυνση:** μια επιχείριση επιβαρύνεται οικονομικά από την αμοιβή των υπαλλήλων που καλούνται να αντιμετωπίσουν την επίθεση DDoS και να επαναφέρουν τις υπηρεσίες της επιχείρισης στη σωστή τους λειτουργία. Παράλληλα, πολλοί οργανισμοί επενδύουν μεγάλα χρηματικά ποσά στην πρόληψη τέτοιων επιθέσεων (ειδικό λογισμικό, σύμβουλοι).

- **αποζημιώσεις σε πελάτες:** οι επιχειρήσεις υπογράφουν με τους πελάτες τους συμφωνίες εγγύησης επιπέδου υπηρεσιών (Service Level Agreements, SLAs) που οφείλουν να τηρούν. Σε μια επίθεση DDoS παραβιάζουν τις εγγυήσεις τους και οι πελάτες τους απαιτούν αποζημιώσεις.

- **πλήγμα στην αξιοπιστία και στην εικόνα της επιχείρισης:** η αδυναμία της επιχείρισης να εγγυηθεί τις υπηρεσίες της οδηγεί στην απώλεια της εμπιστοσύνης των πελατών τους. Παράλληλα, μετά από μια DDoS, μια επιχείριση μπορεί να χρειαστεί να επενδύσει χρήματα για να αποκαταστήσει την εικόνα της (διαφημίσεις, δημόσιες σχέσεις).

Τα κίνητρα [11, 15] τέτοιων επιθέσεων μπορεί να αναζητηθούν στους παρακάτω σκοπούς:

- ***ακτιβισμός (hacktivism)*:** Ακτιβιστές του διαδικτύου μπορούν να χρησιμοποιήσουν τις επιθέσεις DDoS ως μέσο για να διαμαρτυρηθούν για κάποιο κοινωνικό ή πολιτικό ζήτημα.

- ***πνευματική πρόκληση (intellectual challenge):*** Ο επιτιθέμενος εκτελεί επιθέσεις για να πειραματιστεί και να αποκτήσει εμπειρία ή για να γίνει το επίκεντρο της

προσοχής, κερδίζοντας έτσι το σεβασμό της κοινότητας των ανθρώπων που επιδίδονται σε επιθέσεις DDoS.

- ***εκβιασμός*:** Ο επιτιθέμενος εκβιάζει το θύμα πως αν δεν του δώσει ένα συγκεκριμένο ποσό χρημάτων, θα εκτελέσει επίθεση DDoS.

- ***εκδίκηση:*** Ο επιτιθέμενος θέλει να εκδικηθεί το θύμα για κάποιο προσωπικό τους ζήτημα και εξαπολύει επίθεση DDoS εναντίον του.

- ***ανταγωνισμός:*** Μία επιχείριση εξαπολύει επίθεση DDoS εναντίον ενός ανταγωνιστή της για να του προκαλέσει μεγάλες οικονομικές ζημιές και πλήγμα στην αξιοπιστία του.

- ***ηλεκτρονικός πόλεμος (cyberwarfare):*** Μία χώρα μπορεί να εξαπολύσει επίθεση DDoS εναντίον μιας αντίπαλης χώρας για να παραλύσει βασικές υπηρεσίες της.

- ***αντιπερισπασμός:*** μία επίθεση DDoS μπορεί να χρησιμοποιηθεί για να τραβήξει την προσοχή μακριά από κάποια άλλη κακόβουλη δραστηριότητα.

2.3: Επιθέσεις στο DNS

Στην παράγραφο αυτή αναλύονται οι σημαντικότερες επιθέσεις εναντίον του DNS.

2.3.1: DNS flood

Πρόκειται για μία DDoS επίθεση, με θύματα έναν ή περισσότερους authoritative εξυπηρετητές μιας ζώνης DNS και αποτελεί μια παραλλαγή της πλημμύρας UDP (UDP flood) κατά της υπηρεσίας.

Χρησιμοποιώντας IP spoofing και botnet, ο επιτιθέμενος βομβαρδίζει άμεσα το θύμα με μεγάλο όγκο κίνησης DNS, που μπορεί να αποτελείται είτε από έγκυρα είτε από άκυρα ερωτήματα DNS. Ως αποτέλεσμα, ο επιτιθέμενος κατορθώνει να εξαντλήσει το εύρος ζώνης του τοπικού δικτύου και να μονοπωλήσει τους πόρους του εξυπηρετητή, επιδραδύνοντας, έτσι, την ικανότητά του να απαντά στα ερωτήματα των πελατών του ή, ακόμα και να τον καταστήσει ανίκανο να απαντά σε αυτά [16].

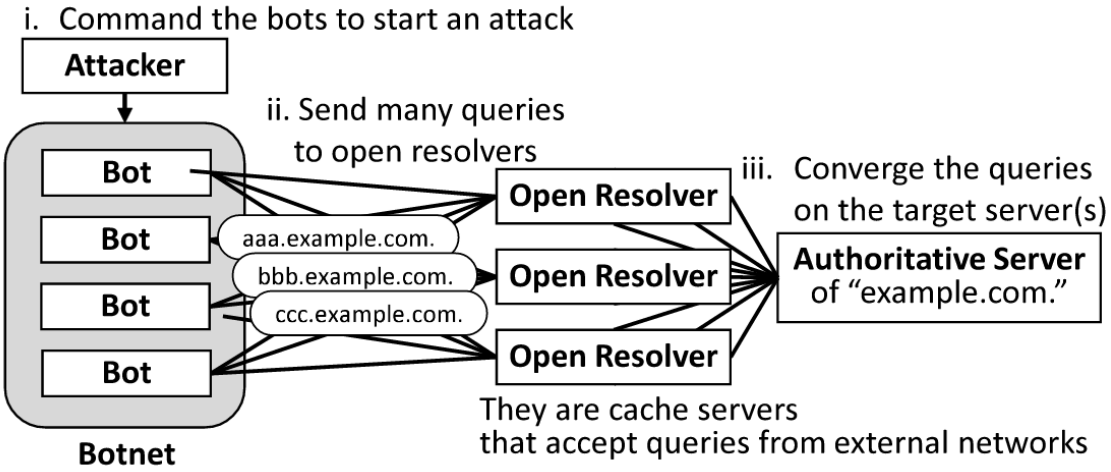
2.3.2: DNS Water Torture

Η **DNS water torture attack**, γνωστή, επίσης, και με τις ονομασίες pseudo random subdomain attack ή DNS slow drip attack, στοχεύει στην εξάντληση της επεξεργαστικής ισχύος ενός **authoritative** DNS server.

Η συγκεκριμένη επίθεση αποτελεί, ουσιαστικά, μία βελτιωμένη έκδοση της DNS flood. Ο

επιτιθέμενος προετοιμάζεται για την επίθεση, δημιουργώντας το botnet του και συγκεντρώνει μια λίστα με διαθέσιμους open resolvers του διαδικτύου. Στη συνέχεια, διατυπώνει έναν πολύ μεγάλο αριθμό από IP spoofed ερωτήματα DNS **τυχαίας μορφής** προς τους open resolvers που έχει καταγράψει. Οι resolvers προωθούν τα ερωτήματα αυτά στον authoritative server.

Η δύναμη της επίθεσης εντοπίζεται στο γεγονός ότι όλη η κίνηση του επιτιθέμενου θα καταλήξει στο θύμα, καθώς η τυχαία μορφή των ερωτημάτων εξασφαλίζει ότι δε θα βρίσκονται αποθηκευμένα στην προσωρινή μνήμη των resolvers. Έτσι, ο επιτιθέμενος μπορεί να στείλει πολύ μεγάλο όγκο κίνησης στο θύμα.



Σχήμα 2.11 – DNS water torture attack.

Η επίθεση αυτή, αν και στοχεύει τον authoritative DNS server, έχει ως παράπλευρη απώλεια και την επιβάρυνση της λειτουργίας των open resolvers.

Ενδείξεις μιας τέτοιας επίθεσης αποτελούν ο μεγάλος αριθμός ερωτημάτων DNS, ο μεγάλος αριθμός απαντήσεων με τον κωδικό **NXDOMAIN** (ανύπαρκτο όνομα) που επιστρέφονται από τον εξυπηρετητή και η αύξηση του ποσοστού χρήσης του επεξεργαστή του εξυπηρετητή.

Ο διαχωρισμός των ουσιαστικών ερωτημάτων από τα ερωτήματα της επίθεσης σε πραγματικό χρόνο είναι πολύ δύσκολος. Η μέθοδος αντιμετώπισης που χρησιμοποιείται, συνήθως, είναι ο περιορισμός του αριθμού των ερωτημάτων που μπορεί να δεχτεί ένας εξυπηρετητής σε ένα χρονικό διάστημα. Η μέθοδος αυτή, όμως, οδηγεί σε απόρριψη πολύ μεγάλου μέρους της καλόβουλης κίνησης, ικανοποιώντας τα κίνητρα της επίθεσης (Πηγή [17]).

2.3.3: DNS cache poisoning

Όπως έχει ήδη αναφερθεί, όταν ένας authoritative DNS server αποστείλει την απάντηση για

ένα ερώτημα DNS σε έναν resolver, ο resolver θα αποθηκεύσει την απάντηση αυτή στην προσωρινή του μνήμη (DNS cache) για δεδομένο χρονικό διάστημα, που ορίζεται από την παράμετρο TTL. Αν ερωτηθεί ξανά από έναν πελάτη για την ίδια εγγραφή μέσα στο διάστημα που ορίζει το TTL, θα του επιστρέψει αμέσως την απάντηση, χωρίς να διατρέξει την ιεραρχία DNS.

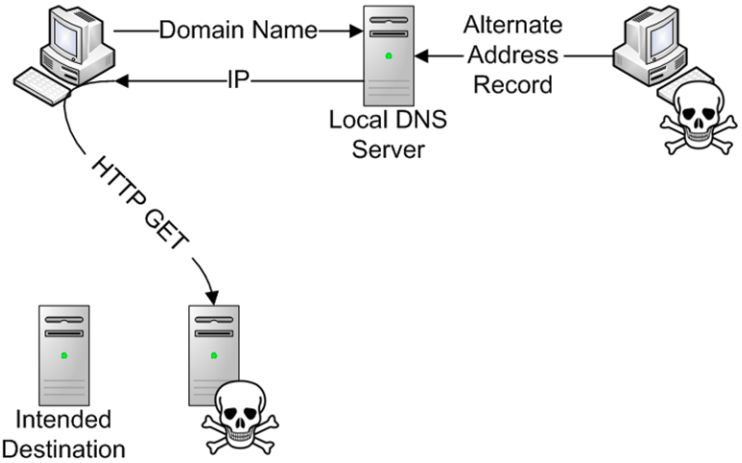
Στην επίθεση αυτή [18, 19], ο επιτιθέμενος επιδιώκει να **"δηλητηριάσει"** το περιεχόμενο της προσωρινής μνήμης (DNS cache) του **recursor** ενός δικτύου, εγκαθιστώντας μία δική του εγγραφή DNS. Η διεύθυνση IP της εγγραφής αυτής θα οδηγεί σε ένα δικό του κακόβουλο μηχάνημα και όχι στον υπολογιστή που υπαγορεύει ο authoritative server που είναι υπεύθυνος για την εγγραφή αυτή. Όταν τα θύματα αναζητήσουν την εγγραφή DNS για το όνομα που έχει παραβιαστεί, αντί να συνδεθούν στο σωστό υπολογιστή, θα συνδεθούν στο μηχάνημα του επιτιθέμενου. Έτσι, εκείνος θα είναι σε θέση να υποκλέψει ευαίσθητες προσωπικές πληροφορίες τους, όπως για παράδειγμα προσωπικά στοιχεία και κωδικούς. Για να το πετύχει, ο επιτιθέμενος υποδύεται τον authoritative server που είναι υπεύθυνος για την εγγραφή και προσπαθεί να αποθηκεύσει στον recursor του δικτύου τη δική του απάντηση, προτού ο πραγματικός authoritative server αποστείλει τη διεύθυνση του σωστού υπολογιστή.

Επισημαίνεται ότι η επίθεση αυτή δεν είναι επίθεση DoS. Ωστόσο, όπως θα δούμε παρακάτω, στην προσπάθειά του να μιμηθεί την απάντηση ενός authoritative server, ο επιτιθέμενος είναι πιθανό να αποστείλει πολύ μεγάλο όγκο δεδομένων στο recursor, εκδηλώνοντας μια επίθεση DoS ως αναγκαίο κακό.

Για να υποδυθεί ο επιτιθέμενος τον authoritative DNS server, πρέπει να μιμηθεί με απόλυτη ακρίβεια την απάντησή του. Συγκεκριμένα, τα πεδία της απόκρισης DNS τα οποία θα πρέπει να μιμηθεί ο επιτιθέμενος είναι τα παρακάτω:

- ***Διεύθυνση προέλευσης της απάντησης (source IP):*** θα πρέπει να είναι η διεύθυνση IP του authoritative DNS server.
- ***Διεύθυνση προορισμού της απάντησης (destination IP):*** θα πρέπει να είναι η διεύθυνση IP του recursor που ανέλαβε να πραγματοποιήσει την αναδρομική αναζήτηση του DNS ερωτήματος.
- Η ***θύρα προέλευσης UDP (source port)*** θα πρέπει να είναι ίδια με τη θύρα στην οποία ακούει ο authoritative server για μηνύματα DNS.
- Η ***θύρα προορισμού UDP*** **(destination port)** θα πρέπει να είναι ίδια με τη θύρα προέλευσης που χρησιμοποίησε ο recursor.
- Το ***UDP checksum*** πρέπει να είναι υπολογισμένο σωστά.
- Η ταυτότητα της δοσοληψίας **(*transaction ID*)** που χρησιμοποίησε ο υπολογιστής που διατύπωσε το ερώτημα DNS πρέπει να είναι η ίδια.
- Το ***FQDN*** που θα υπάρχει στο πεδίο ερώτησης και στο πεδίο απάντησης του πακέτου πρέπει να είναι ίδιο με το FQDN στο πεδίο ερώτησης του ερωτήματος DNS που διατυπώθηκε.

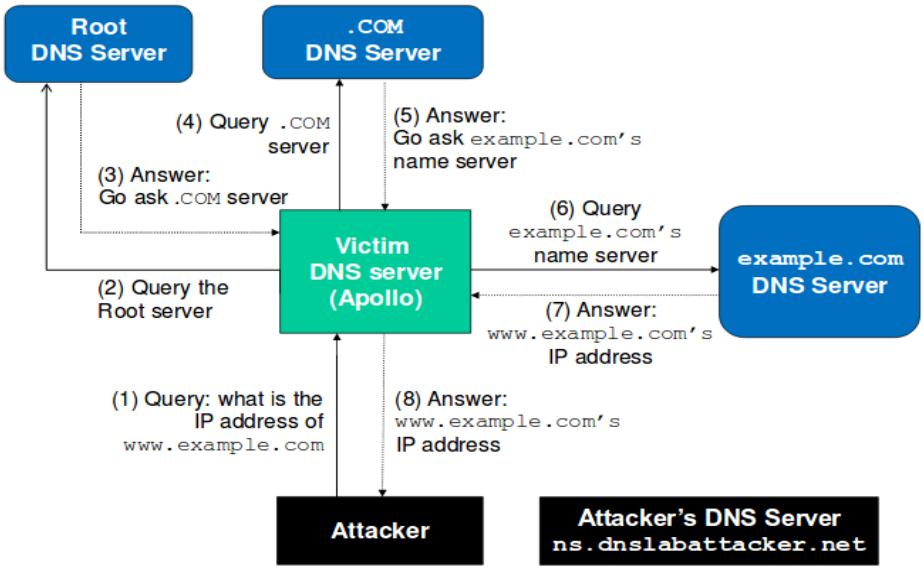
Φυσικά, για να πετύχει η επίθεση, θα πρέπει ο επιτιθέμενος να στείλει τη δική του απάντηση πριν φτάσει η απάντηση του authoritative server. Από τα παραπάνω, είναι εμφανές ότι τα πιο **δύσκολα πεδία** για να μιμηθεί ο επιτιθέμενος είναι το transaction ID και η θύρα προορισμού, καθώς είναι αριθμοί που επιλέγονται με τυχαίο τρόπο κατά τη δημιουργία του ερωτήματος DNS. Η θύρα προέλευσης δεν αποτελεί, συνήθως, πρόβλημα, καθώς οι DNS servers είναι ρυθμισμένοι να ακούν τα μηνύματα DNS στην θύρα UDP 53.



Σχήμα 2.12 - DNS cache poisoning attack: ο επιτιθέμενος δηλητηριάζει την προσωρινή μνήμη του local DNS server και ανακατευθύνει το θύμα σε ένα δικό του μηχάνημα για να του υποκλέψει τα προσωπικά του δεδομένα, χωρίς εκείνο να συνειδητοποιεί τη διαφορά.

Για να γίνει καλύτερα κατανοητή η επίθεση αυτή, θα δοθεί ένα παράδειγμα, το οποίο βασίζεται στο παρακάτω σχήμα (2.14). Ο επιτιθέμενος (Attacker) θέλει να δηλητηριάσει την DNS cache του recursor Apollo. Αρχικά, διατυπώνει ένα ερώτημα DNS για να μάθει τη διεύθυνση IP που αντιστοιχεί στον υπολογιστή με όνομα www.example.com. Ο recursor μην έχοντας την εγγραφή αυτή στην cache του, αναλαμβάνει να βρει την απάντηση. Έπομένως, θα διατυπώσει τα κατάλληλα ερωτήματα μέχρι να βρει τον authoritative server για το domain example.com, ο οποίος φαίνεται δεξιά στο σχήμα. Τέλος, ο authoritative θα στείλει την απάντηση στον recursor και εκείνος θα την αποθηκεύσει στην cache του.

Από τη στιγμή που φτάνει η ερώτηση στον recursor από τον επιτιθέμενο μέχρι τη στιγμή που φτάνει η απάντηση από τον authoritative server, ο επιτιθέμενος στέλνει συνεχώς απαντήσεις DNS στον recursor, υποδυόμενος τον authoritative server της ζώνης example.com. Για να τα καταφέρει και να εγκαταστήσει τη δική του εγγραφή στην cache του recursor, θα πρέπει να πετύχει όλα τα πεδία του μηνύματος που αναφέρθηκαν παραπάνω. Για να πετύχει το transaction ID και το source port, θα χρησιμοποιήσει **τυχαίους** αριθμούς, ελπίζοντας ότι κάποιος συνδυασμός θα είναι ο σωστός. Και τα δύο αυτά πεδία είναι αριθμοί των 16 bits. Κατά συνέπεια, όλοι οι δυνατοί συνδυασμοί είναι $2^{16}*2^{16}=2^{32}$, αριθμός ο οποίος δεν είναι υπερβολικά μεγάλος. Επομένως, αν ο επιτιθέμενος έχει τη δυνατότητα να στείλει K πακέτα σε αυτό το διάστημα, έχει πιθανότητα K στις 2^{32} να πετύχει το σωστό συνδυασμό και να εγκαταστήσει τη δική του εγγραφή.



Σχήμα 2.13 - Παράδειγμα DNS cache poisoning attack

Ωστόσο, παρουσιάζεται ένας πολύ σοβαρός **περιορισμός**. Αν ο επιτιθέμενος δεν καταφέρει να βρει το σωστό συνδυασμό πριν φτάσει η απάντηση του authoritative server στον recursor, ο recursor θα αποθηκεύσει την κανονική εγγραφή στην προσωρινή του μνήμη. Η εγγραφή αυτή θα διατηρηθεί αποθηκευμένη για τόσο χρονικό διάστημα όσο διαρκεί το TTL της, διάστημα που μπορεί να είναι αρκετά μεγάλο (μερικά λεπτά, ώρες ή ακόμα και μέρες). Επομένως, ο επιτιθέμενος θα πρέπει να περιμένει να λήξει το TTL για να προσπαθήσει ξανά. Αν διατυπώσει ξανά το ερώτημα για το www.example.com, ο recursor θα του επιστρέψει αμέσως την απάντηση που κρατάει στην cache του και δε θα επικοινωνήσει με τον authoritative DNS server.

Απάντηση σε αυτό το πρόβλημα έδωσε ο ερευνητής Dan Kaminsky στην προσπάθειά του να καταδείξει την επικινδυνότητα της επίθεσης. Ο Kaminsky πρότεινε και παρουσίασε μια βελτιωμένη μορφή της επίθεσης, με αποτέλεσμα η επίθεση DNS cache poisoning να είναι πλέον γνωστή και ως **Kaminsky Attack**.

Σε αυτήν, ο επιτιθέμενος διατυπώνει ένα ερώτημα DNS για τη διεύθυνση IP ενός ονόματος για το οποίο είναι βέβαιος πως δεν υπάρχει στη ζώνη example.com. Ο recursor θα αναλάβει να βρει την απάντηση στο ερώτημα του επιτιθέμενου, διατρέχοντας την ιεραρχία DNS. Τελικά, ο authoritative server θα απαντήσει στο ερώτημα με NXDOMAIN και, αν αποτύχει η επίθεση, ο recursor θα αποθηκεύσει την απάντηση στην cache του.

Στο μεταξύ, ο επιτιθέμενος βομβαρδίζει τον recursor με απαντήσεις DNS, προσπαθώντας να μαντέψει το source port και το transaction ID. Τώρα, όμως, επιδιώκει να εγκαταστήσει στην cache του recursor όχι μόνο μία εγγραφή τύπου A, δηλαδή μια διεύθυνση IP για το όνομα που ρωτήθηκε, αλλά και μία εγγραφή τύπου NS. Η εγγραφή αυτή θα δηλώνει ότι ο authoritative server για τη ζώνη example.com είναι ο ns.dnslabattacker.net, δηλαδή το μηχάνημα του επιτιθέμενου. Αν τα καταφέρει, λοιπόν, να δηλητηριάσει την προσωρινή

μνήμη του recursor, ο recursor θα αναζητά τις απαντήσεις για τους υπολογιστές της ζώνης example.com στο μηχάνημα του επιτιθέμενου.

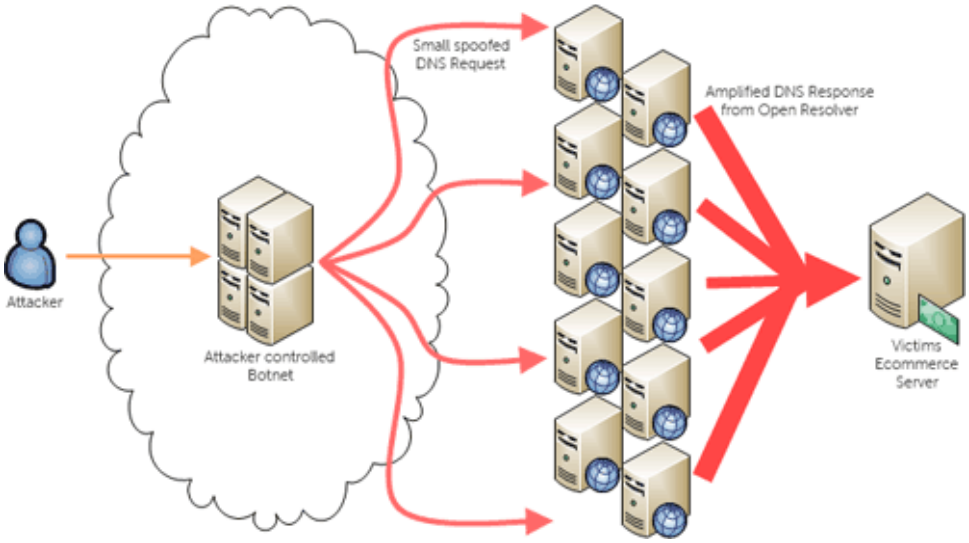
Η σπουδαία βελτίωση της επίθεσης έχει να κάνει με το γεγονός ότι ο επιτιθέμενος αδιαφορεί πια για το αν θα αποθηκευτεί η κανονική απάντηση στην προσωρινή μνήμη του recursor. Αν δεν τα καταφέρει στην πρώτη του προσπάθεια, τότε, άμεσα, θα διατυπώσει ένα ερώτημα DNS για ένα άλλο ανύπαρκτο όνομα υπολογιστή, διαφορετικό από το προηγούμενο και το οποίο γνωρίζει ότι δε βρίσκεται αποθηκευμένο στη μνήμη του recursor.

Στις μέρες μας, η επίθεση Kaminsky δεν αποτελεί πλέον κίνδυνο για την πλειοψηφία των υπολογιστών, αφού έχει υιοθετηθεί το **DNSSec** [20]. Το DNSSec, το οποίο είναι μια επέκταση του DNS, χρησιμοποιεί κρυπτογραφικές τεχνικές και παρέχει αυθεντικοποίηση του server που απαντά σε ένα ερώτημα DNS. Είναι αδύνατο, λοιπόν, κάποιος να υποδυθεί έναν authoritative DNS server που χρησιμοποιεί DNSSec. Βέβαια, υπάρχουν ακόμα πολλοί εξυπηρετητές που δεν έχουν υιοθετήσει το DNSSec.

Ωστόσο, η επίλυση του προβλήματος της επίθεσης DNS cache poisoning, δημιούργησε έναν εξίσου μεγάλο κίνδυνο. Η χρήση DNSSec απαιτεί τη χρήση πολύ μεγαλύτερων σε μέγεθος μηνυμάτων DNS. Το γεγονός αυτό εκμεταλλεύεται η επίθεση που θα παρουσιαστεί στην αμέσως επόμενη ενότητα [61].

2.3.4: DNS Reflection-Amplification

Η επίθεση αυτή [21] είναι μια επίθεση DDoS που σκοπό έχει να πλήξει το **εύρος ζώνης** του δικτύου του θύματος και αποτελεί μια ιδιαίτερα σύνθετη επίθεση.



Σχήμα 2.14 – DNS reflection-amplification attack

Ο επιτιθέμενος, χρησιμοποιώντας το botnet που έχει υπό τον έλεγχό του, στέλνει ένα

μεγάλο αριθμό ερωτημάτων DNS προς μια λίστα από open resolvers. Χρησιμοποιώντας IP spoofing, ο επιτιθέμενος θέτει ως IP προέλευσης των πακέτων αυτών τη διεύθυνση IP του θύματος (κάποιος εξυπηρετητής). Με αυτόν τον τρόπο, οι απαντήσεις των ερωτημάτων DNS θα κατευθυνθούν όλες μαζί στο θύμα (τεχνική reflection).

Το βασικότερο και πιο επικίνδυνο σημείο της επίθεσης είναι η **μεγέθυνση (amplification)** των ερωτημάτων DNS. Ο επιτιθέμενος, κατά το στάδιο προετοιμασίας της επίθεσης, έχει αναζητήσει εγγραφές DNS, οι οποίες για μικρό μέγεθος ερώτησης επιστρέφουν πολύ μεγάλο μέγεθος απάντησης. Τέτοιες εγγραφές είναι, κυρίως, εγγραφές που χρησιμοποιούν DNSSec ή εγγραφές DNS τύπου ANY. Με αυτήν την τεχνική, ο επιτιθέμενος καταφέρνει να στέλνει αιτήματα με πολύ μεγάλη ταχύτητα, χωρίς να κινεί υποψίες, καθώς αυτά έχουν μικρό μέγεθος και να καταφέρνει να επιτίθεται στο θύμα με πολύ μεγάλες απαντήσεις που κατασπαταλούν το εύρος ζώνης του δικτύου του. Ο λόγος του μεγέθους της απάντησης προς το μέγεθος της ερώτησης ονομάζεται **παράγοντας μεγέθυνσης (amplification factor)** και, υπάρχουν και εγγραφές στις οποίες μπορεί να ξεπεράσει το 50.

Η προσφορά της τεχνικής του **reflection** στην επίθεση εντοπίζεται στο ότι ο επιτιθέμενος αποκρύπτει την ταυτότητά του ακόμα περισσότερο, αφού δε χρησιμοποιεί μόνο το botnet ως κάλυψη, αλλά και τους open resolvers. Στην περίπτωση που ο επιτιθέμενος δε χρησιμοποιούσε reflection, το θύμα θα μπορούσε να χρησιμοποιήσει, στα άκρα του δικτύου του, τεχνικές φιλτραρίσματος διευθύνσεων IP. Αυτό σημαίνει ότι θα απέρριπτε πακέτα με κριτήριο τη διεύθυνση IP, καθώς θα γνώριζε ποιες είναι εκείνες που του στέλνουν κακόβουλη κίνηση και θα αντιμετώπιζε την επίθεση αποτελεσματικά. Ωστόσο, όταν χρησιμοποιείται reflection, το θύμα βλέπει ως αποστολείς των μηνυμάτων DNS τους open resolvers. Οι open resolvers αυτοί δε χρησιμοποιούνται μόνο από τον επιτιθέμενο, αλλά και από καλόβουλους χρήστες. Είναι αδύνατον, λοιπόν, για εκείνο να διαχωρίσει την καλόβουλη και την κακόβουλη κίνηση με κριτήριο τις διευθύνσεις IP προέλευσης.

2.4: Δίκτυα νέας γενιάς

2.4.1: Διαστασιολόγηση σύγχρονων δικτύων υπολογιστών

Ο μεθοδικός σχεδιασμός ενός πολύπλοκου συστήματος, όπως είναι ένα δίκτυο υπολογιστών, προϋποθέτει την ανάλυσή του σε απλούστερες συνιστώσες. Η σχεδίαση ενός δικτύου υπολογιστών γίνεται σε τρία **επίπεδα (planes)** λειτουργίας [22]: το data plane, το control plane και το management plane. Καθένα από αυτά τα επίπεδα είναι ανεξάρτητο από τα άλλα, αποτελεί, ουσιαστικά, ένα δίκτυο με ξεχωριστό ρόλο και είδος κίνησης και όλα μαζί, συνιστούν ένα δίκτυο υπολογιστών.

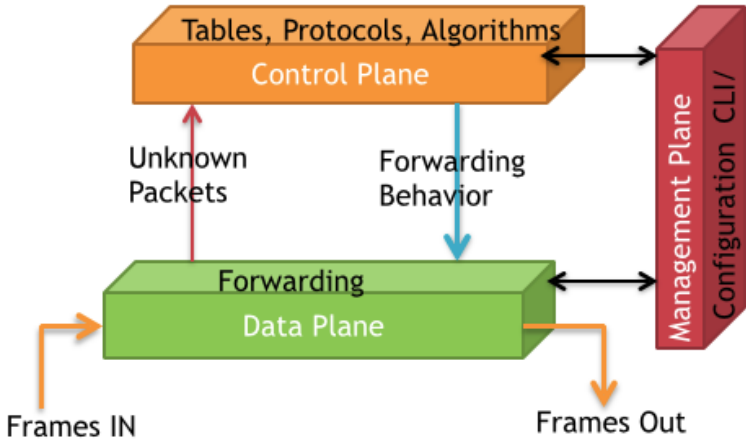
Αναλυτικά, κάθε επίπεδο περιλαμβάνει:

- **data plane:** πρόκειται για τη διάσταση μετάδοσης δεδομένων, δηλαδή περιλαμβάνει την προώθηση δεδομένων σε μεταγωγείς και δρομολογητές, σύμφωνα με

προκαθορισμένους κανόνες, καθώς και την πολυπλεξία των δεδομένων σε φυσικό επίπεδο.

- **control plane:** πρόκειται για το επίπεδο στο οποίο λαμβάνονται οι αποφάσεις για το πώς θα γίνεται η προώθηση των δεδομένων του data plane. Το control plane μεταφέρει τη σηματοδοσία (signalling) των πακέτων (VLAN tags, MPLS labels, IP headers), η οποία είναι απαραίτητη για τη λήψη των αποφάσεων.

- **management plane:** περιλαμβάνει τους κανόνες και τις πολιτικές που καθορίζουν το πώς λαμβάνονται οι αποφάσεις για την προώθηση των πακέτων στο control plane, δηλαδή αφορά τον τρόπο διαχείρισης του δικτύου.



Σχήμα 2.15 – αλληλεπίδραση data plane, control plane, management plane

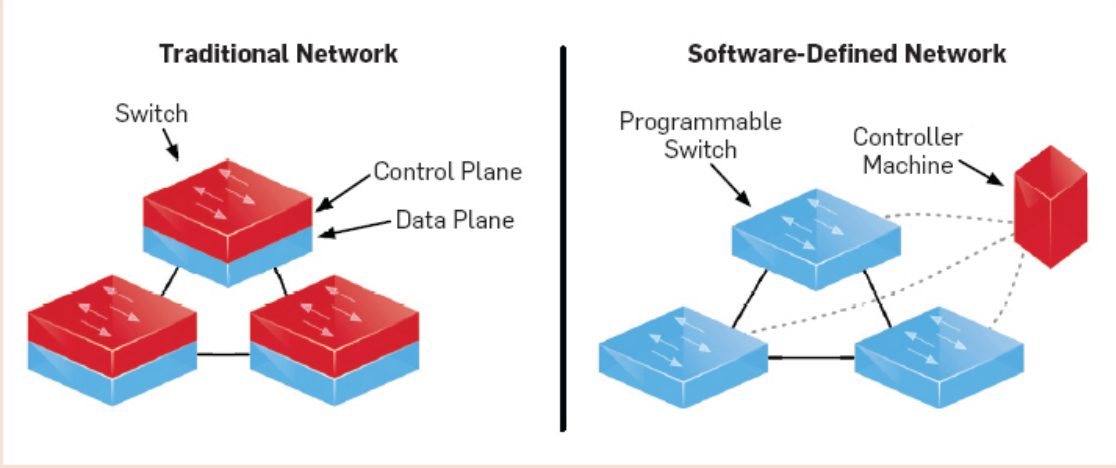
2.4.2: Παραδοσιακά δίκτυα υπολογιστών

Στα παραδοσιακά (legacy) δίκτυα υπολογιστών, η υλοποίηση των τριών επιπέδων σχεδίασης γίνεται πάνω στο firmware των δικτυακών συσκευών, δηλαδή στο ενσωματωμένο από τον κατασκευαστή λογισμικό. Έτσι, κάθε κόμβος διαθέτει το δικό του λειτουργικό σύστημα και κάθε συσκευή έχει το δικό της control plane, που είναι ενωποιημένο με το data plane.

Αυτή η προσέγγιση, αν και εξασφαλίζει υψηλή ανθεκτικότητα για το δίκτυο, δημιουργεί πολύ σοβαρούς περιορισμούς στους διαχειριστές των δικτύων, με αποτέλεσμα πολύ υψηλό διαχειριστικό κόστος. Αρχικά, υπάρχει εξάρτηση από τους κατασκευαστές των δικτυακών συσκευών, καθώς κάθε κατασκευαστής μπορεί να χρησιμοποιεί διαφορετικό σύνολο εντολών για την παραμετροποίηση των συσκευών του. Παράλληλα, εμποδίζεται η κλιμάκωση του δικτύου, αφού η εφαρμογή μιας κοινής πολιτικής σε ολόκληρο το δίκτυο καθίσταται μια πολύ δύσκολη και χρονοβόρα διαδικασία, που απαιτεί τη χειροκίνητη ρύθμιση κάθε συσκευής ξεχωριστά και, μάλιστα, όχι από ένα κεντρικό σημείο. Τέλος,

οποιαδήποτε αλλαγή σε ένα δίκτυο, εκτός από δύσκολη διαδικασία, είναι και επικίνδυνη, γιατί είναι δυνατό να οδηγήσει σε προσωρινή διακοπή της λειτουργίας του. Ως αποτέλεσμα, οι περιορισμοί αυτοί αποθαρρύνουν τους διαχειριστές να κάνουν αλλαγές στα δίκτυά τους, οδηγώντας τα σε στασιμότητα.

Οι σημερινές ανάγκες δικτύωσης (cloud services, Big Data, κατανεμημένες εφαρμογές) καθιστούν την αρχιτεκτονική των δικτύων αυτών ακατάλληλη για τις σύγχρονες εφαρμογές, οι οποίες χαρακτηρίζονται από τεράστιες απαιτήσεις σε υπολογιστική ισχύ και αποθηκευτικό χώρο. Έτσι, προτάθηκε μια νέα προσέγγιση, που ονομάζεται **δικτύωση οριζόμενη από λογισμικό** (**Software-Defined Networking ή SDN**) και δίνει λύση στα παραπάνω προβλήματα, προσφέροντας στους διαχειριστές ευελιξία στη σχεδίαση των δικτύων τους.



Σχήμα 2.16 – Σύγκριση παραδοσιακών δικτύων και SDN

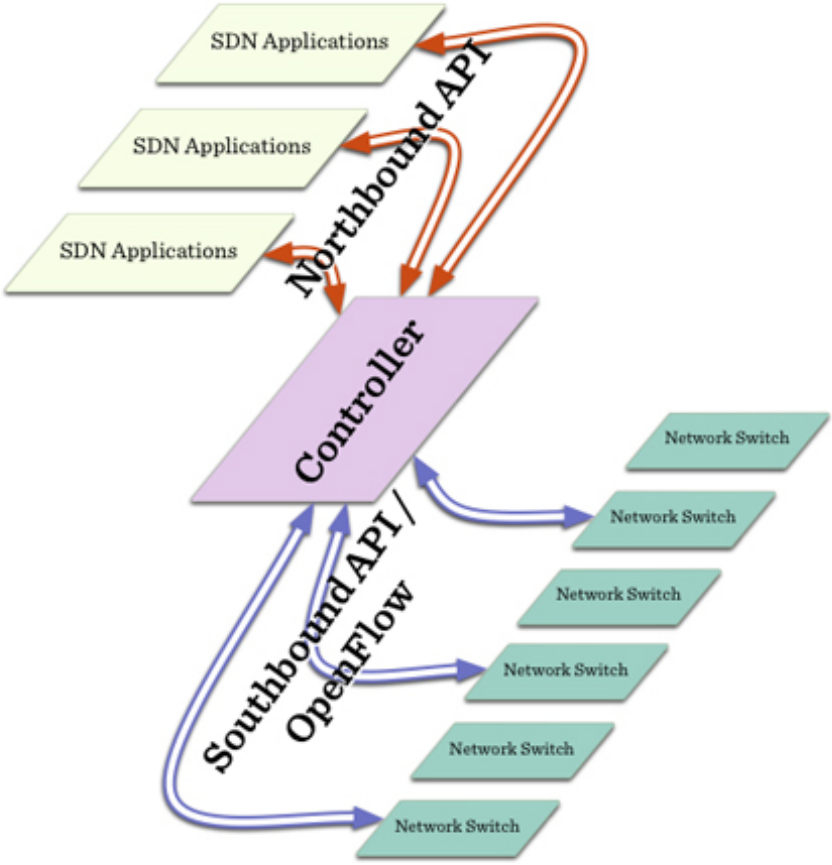
2.4.3: Δίκτυα Οριζόμενα από Λογισμικό (Software Defined Networks, SDN)

Στα δίκτυα οριζόμενα από λογισμικό ή SDNs [23], το data plane των δικτυακών συσκευών διαχωρίζεται από το control plane τους, με αποτέλεσμα τη μετατροπή τους σε απλές συσκευές προώθησης δεδομένων, οι οποίες αποκαλούνται **switches**. Για τους υπολογιστές των χρηστών του δικτύου χρησιμοποιείται ο όρος **hosts**.

Τα control και management planes του δικτύου μεταφέρονται στον **ελεγκτή (controller)**, δηλαδή έναν επόπτη, ο οποίος έχει την ευθύνη για την εξασφάλιση της σωστής λειτουργίας και ομαλής συνύρπαξης των τριών planes. Ο controller μπορεί να είναι είτε ένας μόνο server, είτε μία ομάδα από κατανεμημένους servers, οι οποίοι, όμως, λειτουργούν ως μία, λογικά συγκεντρωμένη σε ένα σημείο, οντότητα.

Ανάλογα με το ποιες οντότητες του SDN θέλουν να επικοινωνήσουν μεταξύ τους, αξιοποιούνται διαφορετικά **APIs** των συσκευών. Ο controller επικοινωνεί με τα switches, χρησιμοποιώντας προτυποποιημένα APIs, τα οποία ονομάζονται southbound interfaces. Οι

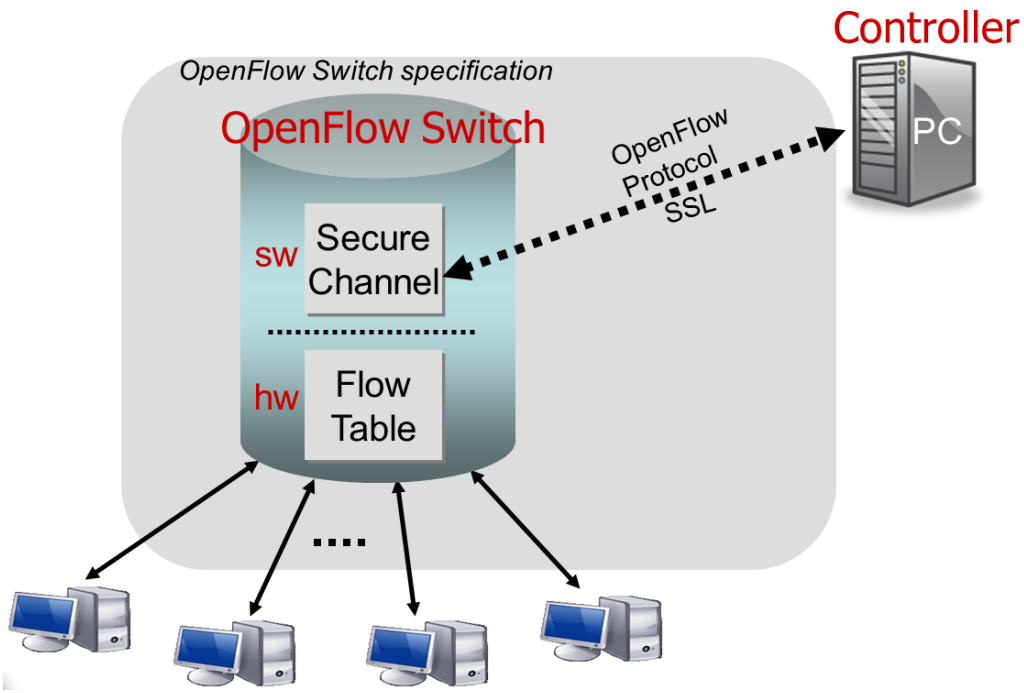
εξυπηρετητές που συνιστούν τον controller επικοινωνούν μεταξύ τους, χρησιμοποιώντας τις east-westbound interfaces, ενώ οι SDN εφαρμογές επικοινωνούν με τον controller, χρησιμοποιώντας τις northbound interfaces.



Σχήμα 2.17 – Επικοινωνία ανάμεσα στις οντότητες του SDN

Μία δημοφιλής μέθοδος επικοινωνίας ανάμεσα στον controller και στα switches είναι το πρότυπο **OpenFlow** [24, 60]. Τα πακέτα του OpenFlow μεταφέρονται είτε μέσω ενός ανοιχτού καναλιού, είτε πάνω από ένα κρυπτογραφημένο με TLS/SSL κανάλι επικοινωνίας (σχήμα 2.24).

Επιπλέον, μία μεγάλη καινοτομία των SDNs είναι η εισαγωγή της έννοιας της **ροής (flow)**. Ως flow ορίζεται ένα σύνολο πακέτων τα οποία έχουν κοινές τιμές σε συγκεκριμένα **πεδία (fields)** της επικεφαλίδας (header) τους. Κάθε switch διαθέτει έναν **πίνακα ροών (flow table)**, σύμφωνα τον οποίο γίνεται η προώθηση των πακέτων που καταφθάνουν στο switch. Σε αυτόν, ο controller μπορεί να εγκαταστήσει **κανόνες ροών (flow rules)**, οι οποίοι υπαγορεύονται από τις πολιτικές διαχείρισης του δικτύου.



Σχήμα 2.18 – Επικοινωνία controller και switch μέσω του OpenFlow.

Κάθε flow rule περιλαμβάνει τρία επιμέρους στοιχεία:

- **πεδία αντιστοίχισης (match fields):** κάθε φορά που ένα πακέτο καταφθάνει στο switch, τα πεδία της επικεφαλίδας του αντιπαραβάλλονται με τα πεδία των εγκατεστημένων κανόνων (match fields). Αν ταιριάξουν με κάποιον, τότε έχουμε **table-hit**, ενημερώνονται οι counters του κανόνα και εφαρμόζεται στο πακέτο το action του κανόνα. Στην περίπτωση ενός **table-miss**, δηλαδή εάν το πακέτο δεν ταιριάξει με κανένα κανόνα του flow table, τότε το πακέτο προωθείται στον controller. Εκείνος, θα εγκαταστήσει ένα νέο flow rule στα flow tables των switches για τα επόμενα πακέτα που θα έχουν ίδια πεδία επικεφαλίδας.

- **μετρητές (counters):** κάθε flow table διατηρεί ένα σύνολο από μετρητές που ανανεώνονται κάθε φορά που ένα εισερχόμενο πακέτο αντιστοιχίζεται σε κάποιο κανόνα. Ο διαχειριστής του δικτύου μπορεί να χρησιμοποιήσει τις τιμές αυτές για να εξαγάγει συμπεράσματα για την κίνηση στο δίκτυό του και, επιπλέον, να προγραμματίσει τον controller να προβαίνει σε συγκεκριμένες ενέργειες ανάλογα με τις τιμές τους.

- **ενέργειες (actions):** στην περίπτωση ενός table-hit, κάποια ενέργεια εφαρμόζεται στο πακέτο. Ένα πακέτο μπορεί να προωθηθεί προς συγκεκριμένη κατεύθυνση, να απορριφθεί, να υποστεί αλλαγές στην επικεφαλίδα του ή να προωθηθεί στον controller.

동작모드	Switch Port	MAC src	MAC dst	Ether type	VLAN ID	Src IP	Dst IP	Proto No.	TCP S_port	TCP D_port	Action	Counter
Switching	*	*	00:1f..	*	*	*	*	*	*	*	Port1	243
Flow Switching	Port3	00:20..	00:2f..	0800	vlan1	1.2.3.4	1.2.3.9	4	4666	80	Port7	123
Routing	*	*	*	*	*	*	1.2.3.4	*	*	*	Port6	452
VLAN Switching	*	*	00:3f..	*	vlan2	*	*	*	*	*	Port6 Port7 Port8	2341
Firewall	*	*	*	*	*	*	*	*	*	22	Drop	544
Default Route	*	*	*	*	*	*	*	*	*	*	Port1	1364

Σχήμα 2.19 – Παράδειγμα flow table.

Επομένως, η αρχιτεκτονική των SDNs προσφέρει μεγάλη ευελιξία στους διαχειριστές των δικτύων. Συγκεκριμένα, τα πλεονεκτήματα των SDNs είναι τα παρακάτω [25]:

- **Άμεσα προγραμματίσιμο control plane**: Το data plane των δικτυακών συσκευών διαχωρίζεται από το control plane τους. Έτσι, ο διαχειριστής του δικτύου έχει τη δυνατότητα να προγραμματίσει άμεσα το control plane του κάθε switch.

- **Κεντρικός έλεγχος**: Η διαχείριση του δικτύου μπορεί να γίνει από ένα κεντρικό σημείο, τον controller, ο οποίος έχει γνώση ολόκληρης της τοπολογίας του δικτύου.

- **Ανεξαρτησία από τον κατασκευαστή**: Η διαχείριση του δικτύου δεν εξαρτάται πλέον από τον κατασκευαστή των δικτυακών συσκευών, καθώς ο προγραμματισμός του control plane των switches γίνεται στον controller.

- **Μείωση πολυπλοκότητας και κόστους**: η δυνατότητα προγραμματισμού του control plane των switches μέσω του controller επιτρέπει την αυτοματοποίηση της διαδικασίας παραμετροποίησής τους, μια διαδικασία που στα παραδοσιακά δίκτυα απαιτεί χειροκίνητες ρυθμίσεις στη συσκευή, έπειτα από άμεση σύνδεση σε αυτή. Έτσι, η πολυπλοκότητα και το κόστος ρύθμισης μιας συσκευής μειώνονται δραματικά.

- **Ευελιξία στη διαχείριση της κίνησης**: τα SDNs επιτρέπουν την προσαρμογή στις διαρκείς αλλαγές που χαρακτηρίζει τη σύγχρονη δικτυακή κίνηση, καθώς ο controller μπορεί να εγκαθιστά δυναμικά κανόνες ροών και να ρυθμίζει καταλλήλως την κίνηση. Έτσι, τα SDNs προσφέρουν μεγάλη ευελιξία στην παρακολούθηση της δικτυακής κίνησης (monitoring) και στην ανίχνευση και αντιμετώπιση δικτυακών

επιθέσεων.

- **Διευκόλυνση του πειραματισμού και καινοτομία**: Με την εγκατάσταση καταλλήλων κανόνων ροών, οι ερευνητές μπορούν να διαχωρίσουν την κανονική κίνηση ενός δικτύου από την κίνηση που δημιουργούν οι δραστηριότητές τους. Έτσι, είναι σε θέση να πειραματιστούν ελεύθερα, χωρίς να δημιουργήσουν προβλήματα στους χρήστες του δικτύου.

2.4.4: SDN controllers

Υπάρχουν πολλοί διαθέσιμοι **controllers** ανοιχτού λογισμικού, από τους οποίους είναι ελεύθερος να επιλέξει ο διαχειριστής ενός δικτύου SDN, ανάλογα με τις δυνατότητες που προσφέρουν και τις ιδιαίτερες προτιμήσεις του. Ο χρήστης μπορεί να αναπτύξει δικό του λογισμικό ή να εκμεταλλευτεί τα εγκατεστημένα **components** του controller, δηλαδή έτοιμο λογισμικό, το οποίο μπορεί να χρησιμοποιήσει χωρίς καμία αλλαγή ή να εμπλουτίσει για να ικανοποιήσει τις ανάγκες του.

Οι πιο διάσημοι SDN controllers και η γλώσσα προγραμματισμού στην οποία είναι γραμμένοι παρουσιάζονται στην παρακάτω λίστα:

- NOX (python)
- POX (python)
- Ryu (python)
- Beacon (java)
- Floodlight (java)
- Trema (Ruby)

2.4.5: OpenFlow switch

Στην ενότητα αυτή θα περιγραφούν τα βασικά σημεία των OpenFlow switches [26].

2.4.5.1: πεδία αντιστοίχισης (match fields)

Είναι τα πεδία που συνιστούν έναν κανόνα και καθορίζουν ποια πακέτα αντιστοιχούν σε αυτόν. Συγκρίνονται με τα αντίστοιχα πεδία της επικεφαλίδας κάθε εισερχόμενου στο switch πακέτου και αποτελούν μία συστάδα (tuple) **12** πεδίων επικεφαλίδας.

Αναλυτικά, πληροφορίες για κάθε match field παρουσιάζονται στον επόμενο πίνακα:

πεδίο	**μήκος σε bits**	**εφαρμόσιμο σε**	**παρατηρήσεις**
Ingress port	Εξαρτάται από την υλοποίηση	Όλα τα πακέτα	Αριθμητική αναπαράσταση της θύρας του switch από την οποία εισέρχεται το πακέτο. Ξεκινάει από το 1.
Ethernet source address	48	Όλα τα πακέτα για enabled θύρες	
Ethernet destination address	48	Όλα τα πακέτα για enabled θύρες	
Ethernet type	16	Όλα τα πακέτα για enabled θύρες	
VLAN id	12	Όλα τα πακέτα με Ethernet type 0x8100	
VLAN priority	3	Όλα τα πακέτα με Ethernet type 0x8100	Πεδίο VLAN PCP
IP source address	32	Όλα τα πακέτα IP και ARP	Υποστηρίζονται υποδίκτυα
IP destination address	32	Όλα τα πακέτα IP και ARP	Υποστηρίζονται υποδίκτυα
IP protocol	8	Όλα τα πακέτα IP, IP over ethernet και ARP	Χρήση μόνο των 8 χαμηλότερων bits του ARP opcode στην περίπτωση πακέτων ARP
IP ToS bits	6	Όλα τα πακέτα IP	Ορίζεται ως τιμή των 8 bits και γίνεται τοποθέτηση του TOS στα 6 ανώτερα bits
Transport source port / ICMP type	16	Όλα τα πακέτα TCP, UDP και ICMP	Χρήση μόνο των 8 χαμηλότερων bits για πακέτα ICMP
Transport destination port / ICMP code	16	Όλα τα πακέτα TCP, UDP και ICMP	Χρήση μόνο των 8 χαμηλότερων bits για πακέτα ICMP

Πίνακας 2.2 – Αναλυτική περιγραφή των match fields ενός flow rule

Κάθε ένα από τα παραπάνω πεδία μπορεί να έχει είτε μία καθορισμένη τιμή ή μπορεί να έχει την τιμή **ANY** (συχνά συμβολίζεται με αστερίσκο), η οποία υποδηλώνει ότι το πεδίο αυτό είναι αδιάφορο και θα ταιριάξει με οποιοδήποτε εισερχόμενο πακέτο, ανεξάρτητα από την τιμή του πεδίου αυτού στην επικεφαλίδα του πακέτου.

2.4.5.2: μετρητές (counters)

Για κάθε flow rule διατηρείται ένα σύνολο από **μετρητές (counters)**, οι οποίοι μπορούν να ληφθούν από τον controller ώστε να εξαχθούν συμπεράσματα για τη δραστηριότητα στο δίκτυο. Κάθε φορά που έχουμε ταίριασμα ενός flow rule με ένα εισερχόμενο πακέτο, οι μετρητές που σχετίζονται με τον κανόνα ανανεώνουν τις τιμές τους. Οι μετρητές αυτοί μπορούν να διατηρούν στατιστικές πληροφορίες είτε ολόκληρου του flow table, είτε για συγκεκριμένες ροές (per flow), είτε για συγκεκριμένες θύρες (per port) του switch ή για συγκεκριμένες ουρές αναμονής (per queue) στο switch.

2.4.5.3: ενέργειες (actions)

Κάθε εγγραφή (flow entry) στον πίνακα περιλαμβάνει ένα πεδίο action, το οποίο ορίζει καμία ή περισσότερες ενέργειες που πρέπει να γίνουν όταν ένα εισερχόμενο πακέτο ταιριάξει με τα matching fields του entry αυτού. Το σύνολο των κανόνων ονομάζεται **λίστα ενεργειών (action list)**. Αν κανένας κανόνας δεν αφορά την προώθηση (forwarding) του πακέτου προς κάποιο προορισμό, το πακέτο απορρίπτεται (drop).

Οι ενέργειες ενός action list πρέπει να εκτελούνται με τη σειρά με την οποία είναι καταγεγραμμένες. Ωστόσο, δεν υπάρχει καμία εγγύηση ότι η σειρά προώθησης των πακέτων, που αφορούν ενέργειες προώθησης, θα είναι σύμφωνη με τη σειρά εκτέλεσης των ενεργειών αυτών. Στην περίπτωση που ένα switch δεν μπορεί να εκτελέσει τις ενέργειες ενός action list με τη σειρά που ορίζονται, απορρίπτει το flow entry και ενημερώνει τον controller με κατάλληλο μήνυμα OpenFlow.

Ένα switch είναι δυνατόν να μην υποστηρίζει όλους τους δυνατούς τύπους ενεργειών. Είναι υποχρεωμένο να υποστηρίζει **συγκεκριμένες ενέργειες (required actions)** και, εάν υποστηρίζει **επιπλέον ενέργειες (optional actions)**, θα ενημερώσει τον controller για αυτές μόλις συνδεθεί σε αυτόν.

Κάθε switch πρέπει **υποχρεωτικά** να υποστηρίζει την απόρριψη πακέτων (ενέργεια drop, περίπτωση κενής λίστας) και την προώθηση (forward) πακέτων σε όλες τις φυσικές θύρες (physical ports), αλλά και στις εικονικές θύρες (virtual ports) που παρουσιάζονται στον παρακάτω πίνακα:

ονομασία εικονικής θύρας	**περιγραφή**
ALL	Προώθηση του πακέτου προς όλες τις διεπαφές του switch, εκτός από εκείνη από την οποία εισήλθε στο switch
CONTROLLER	Αποστολή του πακέτου στον controller ενθυλακωμένο μέσα σε ένα πακέτο openflow
LOCAL	Προώθηση του πακέτου στην τοπική στοίβα δικτύωσης (local networking stack) του switch

TABLE	Εκτέλεση ενεργειών στο flow table. Αναφέρεται μόνο σε μηνύματα packet-out
IN_PORT	Προώθηση του πακέτου στη διεπαφή από την οποία εισήλθε στο switch

Πίνακας 2.3 – Required actions ενός switch

Ενέργειες που, **προαιρετικά**, υποστηρίζει ένα OpenFlow switch παρουσιάζονται στον επόμενο πίνακα:

ονομασίας εικονικής θύρας	**είδος ενέργειας**	**περιγραφή**
NORMAL	forward	Επεξεργασία του πακέτου σύμφωνα με τον παραδοσιακό τρόπο προώθησης (traditional forwarding path), δηλαδή παραδοσιακή layer 2, VLAN, layer 3 επεξεργασία
FLOOD	forward	Πλημμύρα του πακέτου σε όλο το ελάχιστο συνδετικό δέντρο (minimum spanning tree), εξαιρώντας τη διεπαφή προέλευσης του πακέτου
Enqueue	enqueue	Προώθηση του πακέτου σε μία ουρά αναμονής που συνδέεται σε μια θύρα (port) για παροχή επιθυμητής ποιότητας υπηρεσίας (quality of service- QoS)
Modify-field	Field modification	Αλλαγές σε πεδία της επικεφαλίδας του πακέτου, π.χ. στις διευθύνσεις MAC, στις διευθύνσεις IP, στις θύρες προορισμού και προέλευσης, στα VLAN IDs

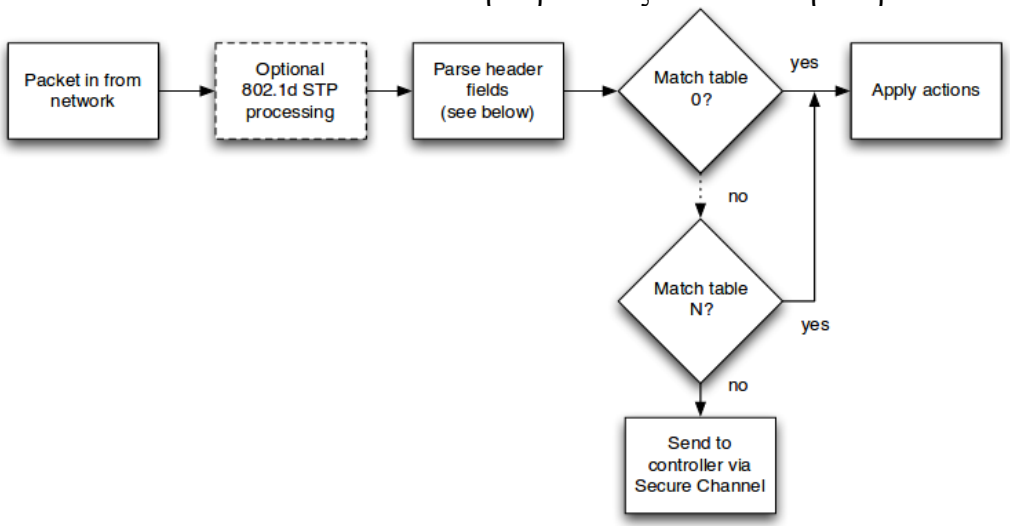
Πίνακας 2.4 – Optional actions ενός switch

Οι συσκευές OpenFlow που υποστηρίζουν μόνο τους υποχρεωτικούς τύπους ενεργειών ονομάζονται openflow-only switches, ενώ οι υπόλοιπες ονομάζονται openflow-enabled συσκευές. Εξαίρεση αποτελεί ο τύπος FLOOD, που είναι δυνατόν να υποστηρίζεται και από τους δύο τύπους συσκευών.

2.4.5.4: Διαδικασία ταιριάσματος (matching)

Όταν ένα switch λάβει ένα πακέτο από το δίκτυο (μήνυμα **packet in**), τότε τα πεδία της

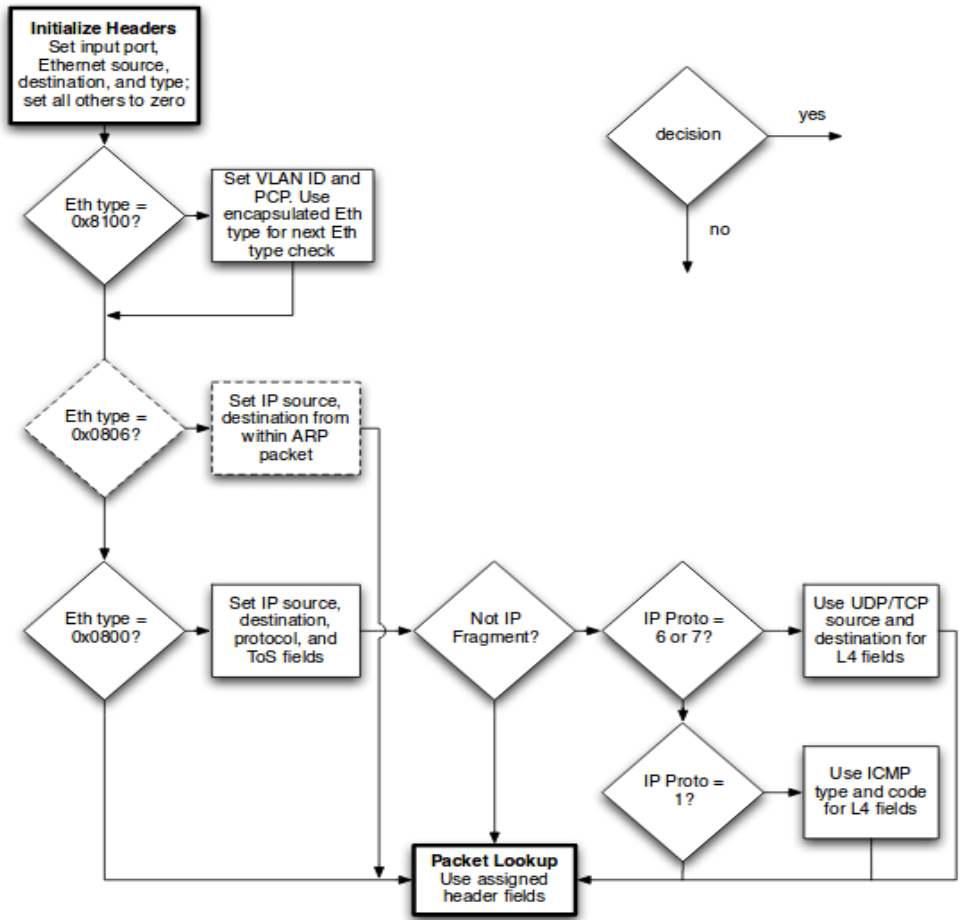
επικεφαλίδας του συγκρίνονται με όλους τους κανόνες στο flow table του switch. Αν βρεθεί ταίριασμα με κάποιον κανόνα, τότε εφαρμόζονται τα actions του κανόνα αυτού. Αν δε βρεθεί κάποιο ταίριασμα, τότε το πακέτο στέλνεται στον controller μέσω ενός κρυπτογραφημένου καναλιού. Προαιρετικά, μπορεί να γίνει επεξεργασία για το πρωτόκολλο 802.1d STP. Η διαδικασία αυτή παρουσιάζεται και στην παρακάτω εικόνα:



Σχήμα 2.20 – Διαδικασία επεξεργασίας εισερχόμενου πακέτου στο switch

Ιδιαίτερη σημασία έχει, φυσικά, η διαδικασία με την οποία γίνεται ο έλεγχος για το αν τα πεδία της επικεφαλίδας του πακέτου ταιριάζουν με τα πεδία ενός flow rule. Ο έλεγχος εκτελείται, ακολουθώντας την παρακάτω σειρά:

- Συγκρίνεται η θύρα από την οποία εισήλθε το πακέτο με κάθε κανόνα που ορίζει μία θύρα εισόδου (ingress port).
- Οι επικεφαλίδες ethernet χρησιμοποιούνται για όλα τα πακέτα.
- Εάν το πακέτο έχει επικεφαλίδες VLAN (τύπος ethernet 0x8100), τότε στην αναζήτηση ταιριάσματος χρησιμοποιούνται τα πεδία VLAN ID και VLAN PCP.
- Προαιρετικά, για τα πακέτα ARP (τύπος ethernet 0x0806), στη διαδικασία αναζήτησης είναι δυνατόν να χρησιμοποιηθούν και τα πεδία IP προέλευσης και IP προορισμού.
- Για πακέτα IP (τύπος ethernet 0x0800), στη διαδικασία αναζήτησης χρησιμοποιούνται και τα πεδία της επικεφαλίδας IP.
- Για πακέτα IP τα οποία είναι TCP ή UDP (το πεδίο IP protocol είναι 6 ή 17 αντίστοιχα), η διαδικασία αναζήτησης χρησιμοποιεί και τις θύρες προέλευσης και προορισμού.
- Για πακέτα IP τα οποία είναι ICMP (το πεδίο IP protocol είναι 1), η διαδικασία αναζήτησης χρησιμοποιεί τα πεδία ICMP type και ICMP code.
- Πακέτα που έχουν υποστεί θρυμματισμό αναγνωρίζονται από τις θύρες του επιπέδου μεταφοράς που έχουν την τιμή 0.



Σχήμα 2.21 – Διαδικασία ελέγχου ταιριάσματος της επικεφαλίδας ενός πακέτου με κάθε flow rule.

Τα εισερχόμενα στο switch πακέτα αντιστοιχίζονται στα flow entries με βάση κάποια **προτεραιτότητα (priority)**, καθώς είναι δυνατό ένα πακέτο να αντιστοιχεί σε πολλά flow entries. Κάθε flow entry που περιλαμβάνει τιμές ANY στα πεδία του, συνοδεύεται και από μία προτεραιότητα. Ένα flow entry που ταιριάζει με το πακέτο και δεν περιλαμβάνει κανένα πεδίο με την τιμή ANY έχει τη μεγαλύτερη δυνατή προτεραιτότητα. Από τα flow entries που ταίριαξαν με το πακέτο, επιλέγεται κάθε φορά εκείνο με τη μεγαλύτερη προτεραιότητα. Ανάμεσα σε flow entries με την ίδια προτεραιότητα, το switch επιλέγει ελεύθερα κάποιο από αυτά.

2.4.5.5: Αφαίρεση flow rule

Ένα flow rule είναι δυνατόν να αφαιρεθεί με δύο τρόπους από ένα flow table:

- ***χειροκίνητα***: ένας controller αφαιρεί ένα flow rule στέλνοντας ένα κατάλληλο μήνυμα OpenFlow στο switch.

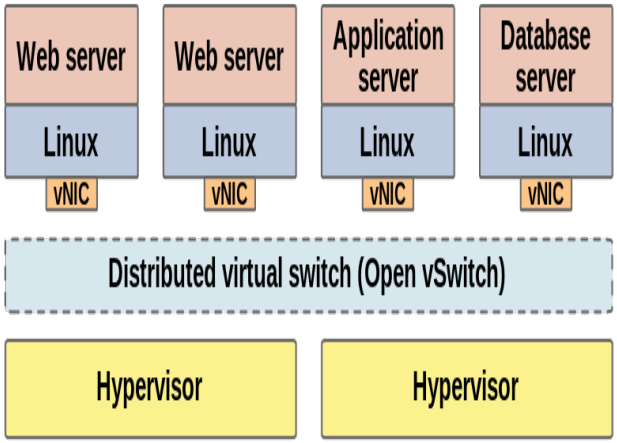
- ***αυτόματα***: όταν ο controller εγκαθιστά ένα flow rule στο switch, προσδιορίζει και

δύο χρονικές τιμές για τον κανόνα, το idle_timeout και το hard_timeout. Εάν κανένα πακέτο δεν έχει ταιριάξει με τον κανόνα αυτό για idle_timeout δευτερόλεπτα ή έχουν περάσει hard_timeout δευτερόλεπτα από την εγκατάσταση του κανόνα, τότε ο κανόνας αφαιρείται αυτόματα από το flow table του switch.

2.4.6 Open vSwitch

Στο παρελθόν, οι εξυπηρετητές των data centers ήταν φυσικά μηχανήματα. Έτσι, για να συνδεθούν με το εξωτερικό δίκτυο και μεταξύ τους έπρεπε να συνδεθούν πρώτα σε ένα hardware-based L2 switch. Με την επικράτηση της εικονικοποίησης υπολογιστικών συστημάτων, οι servers λειτουργούν, πλέον, σε εικονικά μηχανήματα. Επομένως, το L2 switch μπορούσε να υλοποιηθεί πάνω σε λογισμικό.

Το **Open vSwitch (OVS)** [27, 28] είναι μια υλοποίηση ανοιχτού κώδικα (open-source) ενός εικονικού κατανεμημένου πολυεπίπεδου μεταγωγέα (distributed virtual multilayer switch). Το OVS είναι κατάλληλα σχεδιασμένο έτσι ώστε να επιτρέπει την αποτελεσματική υλοποίηση αυτοματισμών στο δίκτυο μέσω προγραμματιστικών επεκτάσεων, ενώ, παράλληλα, υποστηρίζει πρότυπες διαχειριστικές διεπαφές και πρωτόκολλα. Παραδείγματα είναι το NetFlow, το sFlow, το OpenFlow, port-mirroring και εικονικά δίκτυα (VLANs). Το OVS μπορεί να λειτουργεί σε ένα μόνο server ή η λειτουργία του να κατανέμεται σε περισσότερους από έναν servers.



Σχήμα 2.22 – Σύνδεση εικονικών μηχανών σε έναν εικονικό μεταγωγέα OVS κατανεμημένο σε δύο μηχανήματα.

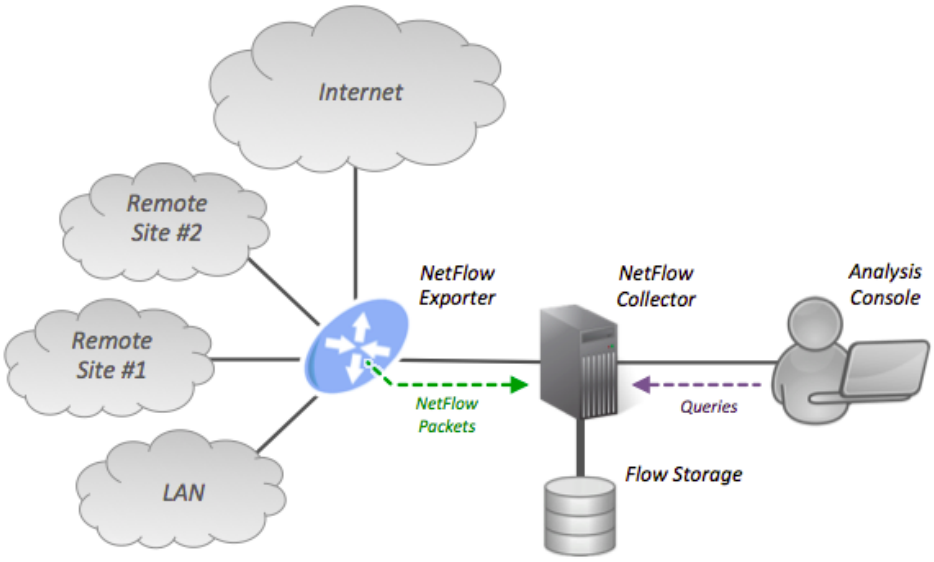
2.5: Παρακολούθηση δικτυακών και υπολογιστικών υποδομών

Βασικό μέρος της διαχείρισης ενός δικτύου είναι η παρακολούθηση (**monitoring**) των συσκευών και της κίνησής του με στόχο την εξαγωγή στατιστικών στοιχείων που θα βοηθήσουν στη βελτιστοποίησή του και στην εξασφάλιση της ασφαλούς λειτουργίας του.

2.5.1: NetFlow

Το **NetFlow** [29, 30] αποτελεί ένα πρότυπο παρακολούθησης δικτυακής κίνησης που επιτρέπει τη συλλογή κίνησης IP, καθώς διέρχεται από ένα μεταγωγέα ή ένα δρομολογητή που το υποστηρίζει. Στατιστικά συλλέγονται από **όλη** την κίνηση IP που φτάνει στις NetFlow – enabled διεπαφές της συσκευής και οργανώνονται σε **ροές (flows)**, δηλαδή ακολουθίες πακέτων που έχουν ίδια τα ακόλουθα **7** πεδία (5η έκδοση) της επικεφαλίδας τους:

- **διεπαφή εισόδου στη συσκευή** (ingress port)
- **διεύθυνση IP προέλευσης** (source IP address)
- **διεύθυνση IP προορισμού** (destination IP address)
- **πρωτόκολλο επιπέδου 3** (πεδίο IP protocol)
- **τύπος υπηρεσίας IP** (IP Type of Service)
- **θύρα προέλευσης** (source port) για UDP και TCP, 0 για τα υπόλοιπα πρωτόκολλα.
- **θύρα προορισμού** (destination port) για UDP και TCP, τύπος και κωδικός για ICMP, 0 για τα υπόλοιπα πρωτόκολλα.



Σχήμα 2.23 – η αρχιτεκτονική λειτουργίας του NetFlow.

Μία υποδομή που πραγματοποιεί monitoring με NetFlow περιλαμβάνει, συνήθως, τρία βασικά συσταστικά στοιχεία:

- **εξαγωγέας ροών (flow extractor):** συγκεντρώνει τα πακέτα σε ροές και στέλνει εγγραφές ροών (flow records) σε έναν ή περισσότερους συλλέκτες ροών.

- **συλλέκτης ροών (flow collector):** λαμβάνει, αποθηκεύει και μορφοποιεί τις

εγγραφές ροών που λαμβάνει από τον flow extractor.

- **εφαρμογή ανάλυσης (analysis application):** εξετάζει τα δεδομένα που λήφθηκαν.

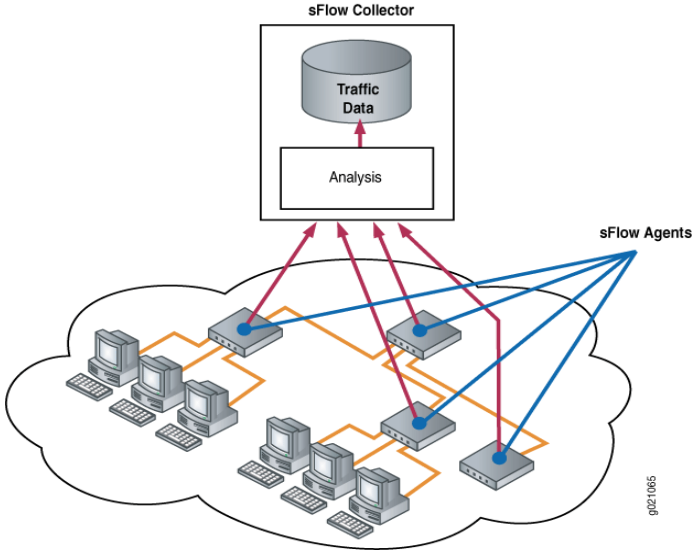
Η αρχιτεκτονική λειτουργίας του NetFlow παρουσιάζεται στην παραπάνω εικόνα.

2.5.2: sFlow

Το **sFlow (sampled flow)** [31, 32] αποτελεί ένα πρότυπο παρακολούθησης κίνησης που απευθύνεται σε δίκτυα πολύ υψηλών ταχυτήτων, στις οποίες καταφέρνει να ανταποκριθεί με επιτυχία, πραγματοποιώντας δειγματοληψία πακέτων.

Το sFlow δειγματοληπτεί **τυχαία** ένα πακέτο από τα n που διέρχονται μέσα από μία δικτυακή συσκευή, για την οποία χρησιμοποιείται ο όρος **sFlow agent**. Στη συνέχεια, τα ενθυλακώνει σε sFlow datagrams και τα στέλνει σε μία συσκευή συλλογής, η οποία ονομάζεται **sFlow collector** για ανάλυση και αποθήκευση. Ένα δεδομενόγραμμα sFlow είναι δυνατό να περιέχει ένα ή περισσότερα δείγματα πακέτων.

Οι κύριες διαφορές του sFlow και του NetFlow είναι ότι το NetFlow δε συλλέγει ολόκληρο το πακέτο, αλλά συγκεκριμένα πεδία της επικεφαλίδας του, ενώ δεν κάνει δειγματοληψία, γεγονός που το καθιστά ακατάλληλο για δίκτυα υψηλών ταχυτήτων.



Σχήμα 2.24 – Η αρχιτεκτονική του sFlow.

2.5.3: Collectd

Το **Collectd** [33] είναι ένας **daemon** ελεύθερου λογισμικού σε περιβάλλον UNIX, ο οποίος επιτρέπει τη συλλογή στατιστικών στοιχείων από δικτυακές συσκευές. Αυτά μπορεί να αφορούν το λειτουργικό τους σύστημα, το υλικό τους, τις εφαρμογές τους, καθώς και τη

δικτυακή κίνηση που σχετίζεται με αυτές.

Η λειτουργία του Collectd βασίζεται σε ένα μεγάλο αριθμό από **plug-ins**. Ο διαχειριστής είναι ελεύθερος να επιλέξει ποια plug-ins θέλει να ενεργοποιήσει από αυτά που υπάρχουν ήδη διαθέσιμα ή να φτιάξει δικά του plug-ins, που εξυπηρετούν τις ανάγκες του καλύτερα.

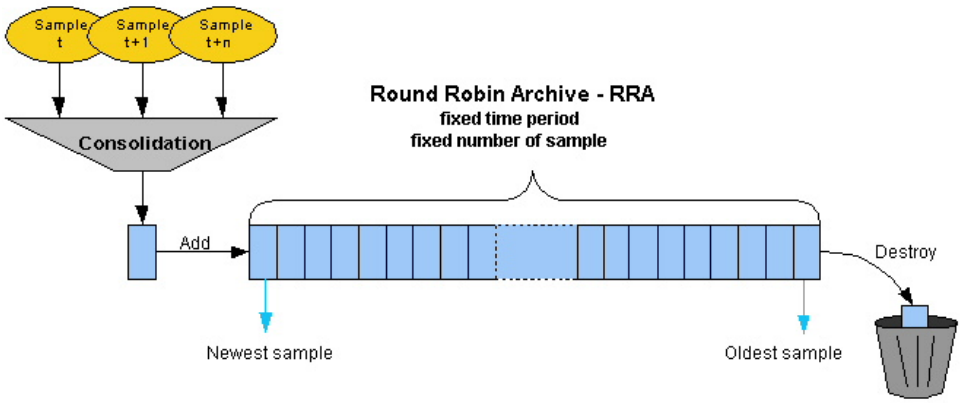
Τα στοιχεία που συλλέγονται από τον Collectd daemon για μια δικτυακή συσκευή (Collectd client) μπορούν είτε να συγκεντρώνονται στη συσκευή αυτή είτε να αποστέλλονται σε έναν κεντρικό συλλέκτη (Collectd server) μέσω του δικτύου. Η ρύθμιση αυτής της αρχιτεκτονικής γίνεται με το **plug-in network**. Ένας Collectd server είναι δυνατόν να συγκεντρώνει δεδομένα από περισσότερους από έναν collectd clients.

Ενδεικτικά plug-ins του Collectd είναι:

- ***cpu***: πληροφορίες για το ποσοστό χρήσης του επεξεργαστή της συσκευής.
- ***dns***: πληροφορίες για τη DNS κίνηση που αφορά τη συσκευή, όπως αριθμός ερωτημάτων-απαντήσεων, πλήθος απαντήσεων με κωδικό NXDOMAIN.
- ***memory***: πληροφορίες για τη φυσική μνήμη της συσκευής.
- ***RRDtool***: αποθήκευση των πληροφοριών που συλλέγονται σε RRDfiles.

2.5.4: Round-Robin Database tool (RRDtool)

Το **Round-Robin Database tool (RRDtool)** [34, 35] είναι ένα εργαλείο κατάλληλα σχεδιασμένο για την επεξεργασία και αποθήκευση χρονοσειρών (time series), που έχουν σταθερή περίοδο δειγματοληψίας. Μία **Round-Robin Database** είναι μία κυκλική δομή δεδομένων με **σταθερό** μέγεθος, στην οποία διαδοχικές τιμές αποθηκεύονται σε διαδοχικές θέσεις. Όταν εξαντληθούν οι θέσεις της δομής δεδομένων, λόγω της κυκλικής φύσης της δομής, η εισαγωγή μίας νέας τιμής στην πρώτη θέση της δομής απαιτεί την αφαίρεση της τιμής που βρίσκεται στην τελευταία της θέση.



Σχήμα 2.25 – Μία Round-Robin δομή δεδομένων.

2.6: Μέθοδοι ελέγχου και κατηγοριοποίησης πληροφορίας

Στην ενότητα που ακολουθεί περιγράφονται αλγόριθμοι και δομές δεδομένων που χρησιμοποιήθηκαν στο μηχανισμό που υλοποιήθηκε ώστε να ελεγχθεί και να κατηγοριοποιηθεί η πληροφορία που συλλέχθηκε.

2.6.1: Μηχανική μάθηση (Machine Learning)

Machine learning [36] είναι ο κλάδος της επιστήμης των υπολογιστών, στον οποίο μελετούνται μέθοδοι, σύμφωνα με τις οποίες ένας υπολογιστής μπορεί να μάθει να εκτελεί μία δραστηριότητα, χωρίς να του έχουν δοθεί σαφώς καθορισμένες οδηγίες για το πώς θα την εκτελέσει. Ο προγραμματιστής δεν παρέχει στον υπολογιστή κάποιον αλγόριθμο που να περιγράφει πώς πρέπει να εκτελέσει μια ενέργεια, αλλά του παρέχει δεδομένα στην είσοδο και, εκείνος, αφού τα μελετήσει και ανακαλύψει κάποιο μοτίβο σε αυτά, θα είναι σε θέση να προβλέψει την επιθυμητή έξοδο με κάποιο αποδεκτό χαμηλό ποσοστό αποτυχίας.

2.6.1.1: Βασικές έννοιες

Στο machine learning συναντώνται τρεις διαφορετικές ομάδες αλγορίθμων:

- ***supervised learning: π***αρέχονται οι είσοδοι και οι επιθυμητές έξοδοι για αυτές (training set, dataset). Τα δεδομένα αυτά μελετώνται από τον αλγόριθμο και προσδιορίζεται ένας γενικός κανόνας αντιστοίχισης εισόδων και εξόδων με βάση κάποιο κριτήριο ελαχιστοποίησης του κόστους λειτουργίας.

- ***unsupervised learning:*** παρέχονται δεδομένα χωρίς να προσδιορίζονται οι επιθυμητές έξοδοι και στόχος είναι να ανακαλυφθεί κάποιο μοτίβο σε αυτά. Με αυτόν τον τρόπο, θα γίνουν προβλέψεις για τις επόμενες εισόδους του χρήστη.

- ***reinforcement learning:*** ο υπολογιστής αφήνεται ελεύθερος να λειτουργήσει μέσα σε ένα δυναμικό περιβάλλον. Μέσω τροφοδότησης, μαθαίνει να εκτελεί σωστά την ενέργεια που πρέπει, καθώς στην περίπτωση που επιλέγει τη σωστή έξοδο θα του παρέχεται "επιβράβευση", ενώ όταν επιλέγει τη λανθασμένη θα του παρέχεται "ποινή".

Επισημαίνεται ότι η είσοδος ενός αλγορίθμου μηχανικής μάθησης είναι ένα διάνυσμα διάστασης Ν και καθεμία από τις συντεταγμένες του ονομάζεται **feature**. Αν η έξοδος του αλγορίθμου είναι μια πραγματική τυχαία μεταβλητή μιλάμε για **regression**, ενώ όταν είναι διακριτή τυχαία μεταβλητή μιλάμε για **classification**.

2.6.1.2: Απήχηση του machine learning και πλεονεκτήματα - μειονεκτήματα

Η δημοφιλία του machine learning στις σύγχρονες εφαρμογές είναι τεράστια. Παραδείγματα σύγχρονων εφαρμογών στις οποίες χρησιμοποιείται machine learning είναι οι παρακάτω:

- Ανίχνευση spam μηνυμάτων ηλεκτρονικού ταχυδρομείου.
- Όραση υπολογιστών, επεξεργασία εικόνας και αναγνώριση φωνής.
- Προβολή διαφημίσεων σύμφωνα με τις προτιμήσεις του χρήστη.
- Διάγνωση ασθενειών στην ιατρική επιστήμη.
- Επεξεργασία φυσικής γλώσσας.
- Μηχανές αναζήτησης (browsers) για την προβολή συνδέσμων σύμφωνα με τις παλαιότερες αναζητήσεις του χρήστη.
- Προβλέψεις για το μέλλον της οικονομίας.
- Αναγνώριση προτύπων σε μεγάλους όγκους δεδομένων.

Το machine learning έχει καθιερωθεί για τα **πλεονεκτήματα** που προσφέρει στους προγραμματιστές, αλλά και από την απήχηση της τεχνητής νοημοσύνης στους χρήστες των εφαρμογών:

- Κάποια προβλήματα είναι τόσο σύνθετα που η συγγραφή ενός αλγορίθμου για την επίλυσή τους είναι είτε πολύ δύσκολη είτε αδύνατη. Σε αυτές τις περιπτώσεις, η εκπαίδευση του συστήματος με τεχνικές μηχανικής μάθησης αποτελεί την καλύτερη επιλογή.

- Οι χρήστες εντυπωσιάζονται ιδιαίτερα από εφαρμογές τεχνητής νοημοσύνης, οι οποίες μπορούν και προσαρμόζονται στις ιδιαίτερες ανάγκες και προτιμήσεις τους.

- Η διακίνηση πολύ μεγάλου όγκου δεδομένων στο διαδίκτυο και η πορεία προς το διαδίκτυο των αντικειμένων (internet of things) καθιστά την επιλογή αλγορίθμων μηχανικής μάθησης την καλύτερη λύση.

Ωστόσο, το machine learning έχει και κάποια **μειονεκτήματα**:

- Η ακρίβεια των προβλέψεων ενός αλγορίθμου μηχανικής μάθησης εξαρτάται από την επιλογή σωστών δεδομένων για την εκπαίδευσή του. Η διαδικασία συγκέντρωσης των δεδομένων αυτών ενδέχεται να είναι δύσκολη και χρονοβόρα και πρέπει να γίνει με πολύ μεγάλη προσοχή.

- Τα αποτελέσματα ενός αλγορίθμου μηχανικής μάθησης αποτελούν προβλέψεις, δηλαδή πρέπει πάντα να εξετάζεται το μέγεθος των συνεπειών από μία λάθος έξοδο του αλγορίθμου.

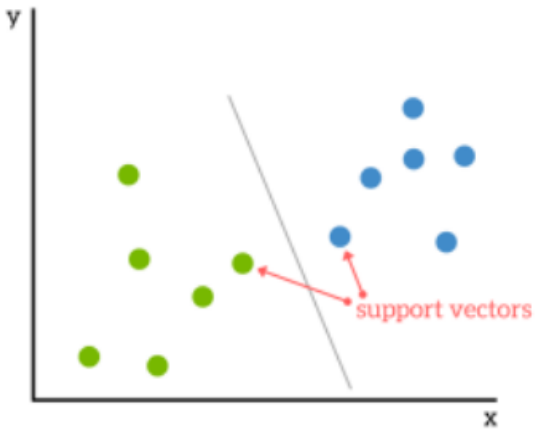
2.6.1.3: Ο αλγόριθμος Support Vector Machines (SVM)

Ο **SVM** [37] είναι ένας αλγόριθμος supervised μηχανικής μάθησης που μπορεί να χρησιμοποιηθεί για **δυαδικά** (binary) classification προβλήματα, δηλαδή για την ταξινόμηση μίας εισόδου σε μία από δύο δυνατές κατηγορίες (**classes**).

Ένα **υπερεπίπεδο (hyperplane)** είναι το ν-διάστατο όριο σύμφωνα με το οποίο λαμβάνεται η απόφαση για την κλάση στην οποία ανήκει ένα σημείο. Στις δύο διαστάσεις, ένα υπερεπίπεδο είναι μια ευθεία γραμμή, ενώ στις τρεις διαστάσεις, ένα υπερεπίπεδο είναι ένα επίπεδο. Ένα υπερεπίπεδο παριστάνεται από την παρακάτω εξίσωση:

$$w^T x+b=0$$

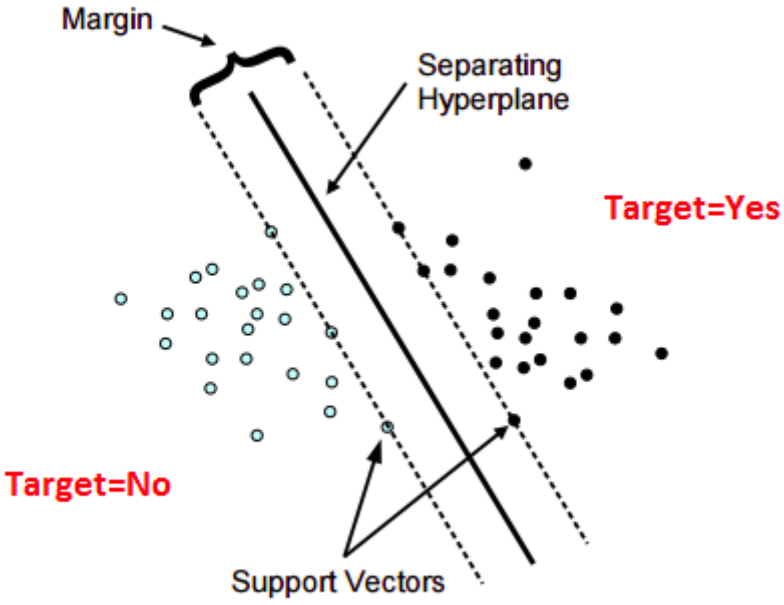
όπου **w** είναι το διάνυσμα με τις παραμέτρους του hyperplane και b μία πραγματική σταθερά (offset παράμετρος).



Σχήμα 2.26 - Support vectors και βέλτιστο hyperplane

Κάθε σημείο του training set παριστάνεται με ένα διάνυσμα. Οι συντεταγμένες των σημείων του training set και των δύο κατηγοριών, που βρίσκονται πλησιέστερα στο υπερεπίπεδο και, κατά συνέπεια, καθορίζουν τη θέση του, ονομάζονται **support vectors**. Ως κριτήριο της απόστασης χρησιμοποιείται η ευκλείδια νόρμα. Τα support vectors είναι, ουσιαστικά, εκείνα τα σημεία του training set, των οποίων η ταξινόμηση είναι και η πιο δύσκολη.

Η απόσταση ανάμεσα στα support vectors των δύο κατηγοριών ονομάζεται διάκενο (**margin**). Όσο πιο μακριά βρίσκεται ένα σημείο από το υπερεπίπεδο, τόσο πιο βέβαιοι μπορούμε να είμαστε ότι ταξινομήθηκε σωστά.



Σχήμα 2.27 – ο αλγόριθμος Support Vector Machines

Στόχος, λοιπόν, του αλγορίθμου SVM είναι να προσδιορίσει το βέλτιστο δυνατό υπερεπίπεδο, δηλαδή εκείνο που εξασφαλίζει το μεγαλύτερο δυνατό margin ανάμεσα στα support vectors, φροντίζοντας, παράλληλα, για τη σωστή ταξινόμησή τους, δηλαδή τα σημεία προς ταξινόμηση να παραμένουν στη σωστή κατηγορία.

Στη συνέχεια, ακολουθεί η **μαθηματική περιγραφή** [62,63] του αλγορίθμου. Έστω ότι τα labels των δύο κατηγοριών του training set είναι -1 και +1, δηλαδή για την έξοδο του αλγορίθμου ισχύει:

$$y=\{-1,+1\}$$

Για τα σημεία του training set με label -1 ισχύει η εξίσωση:

$$\boldsymbol{w}^T \boldsymbol{x}+b\leq -1$$

Αντίστοιχα, για τα σημεία του training set με label +1 ισχύει η εξίσωση:

$$\boldsymbol{w}^T \boldsymbol{x}+b\geq +1$$

όπου η ισότητα ισχύει για τα support vectors κάθε κατηγορίας.

Έπειτα από σύντομες πράξεις και, κανονικοποιώντας με το μέτρο του διανύσματος **w** για να

αποφευχθεί η επίδρασή του, προκύπτει ότι το margin ανάμεσα στα support vectors είναι ίσο με:

$$\frac{2}{\|\boldsymbol{w}\|}=\frac{2}{\sqrt{\boldsymbol{w}^T\boldsymbol{w}}}$$

Στόχος του αλγορίθμου SVM είναι να μεγιστοποιήσει την απόσταση αυτή όσο δυνατόν περισσότερο υπό την προϋπόθεση ότι τα δεδομένα του training set εξακολουθούν να ταξινομούνται στη σωστή κατηγορία. Η απαίτηση αυτή εκφράζεται από το παρακάτω:

$$max\left(\frac{2}{\|\boldsymbol{w}\|}=\frac{2}{\sqrt{\boldsymbol{w}^T\boldsymbol{w}}}\right)$$

$$y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\geq 1$$

Η επίλυση του παραπάνω προβλήματος είναι δύσκολη. Το πρόβλημα αυτό αποδεικνύεται ότι είναι ισοδύναμο με το παρακάτω πρόβλημα τετραγωνικού προγραμματισμού (quadratic programming):

$$min\left(\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}\right)$$

$$y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\geq 1$$

Η ολοκληρωμένη περιγραφή του αλγορίθμου δίνει, επίσης, τη δυνατότητα στο χρήστη του αλγορίθμου να προσδιορίσει ένα trade-off ανάμεσα στην πολυπλοκότητα του hyperplane και την αυστηρή ταξινόμηση των σημείων του training set (παράμετρος C). Τέλος, είναι δυνατό να μετρηθεί το μέγεθος της παραβίασης των απαιτήσεων του αλγορίθμου με τις παραμέτρους ξ_i. Η ολοκληρωμένη έκφραση του προβλήματος που καλείται να επιλύσει ο αλγόριθμος Support Vector Machines είναι η παρακάτω:

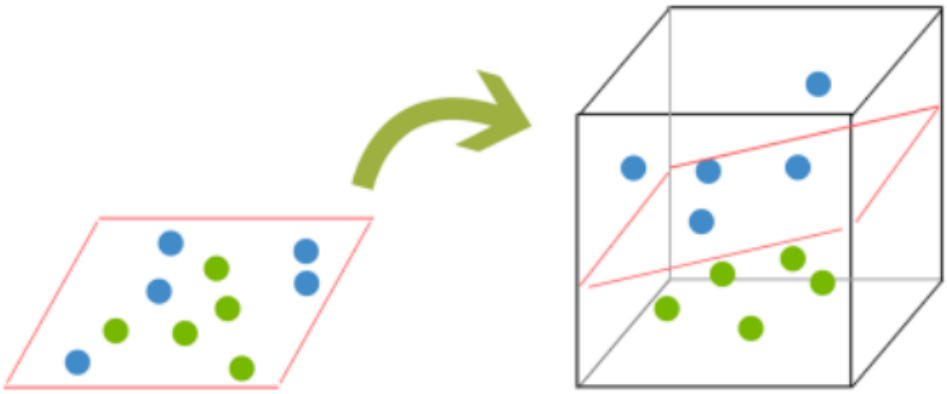
$$min\left(\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}+C\sum_{i=1}^{n}\xi_i\right)$$

$$y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\geq 1-\xi_i,\xi_i\geq 0$$

όπου n ο αριθμός των features.

Στις περισσότερες εφαρμογές, είναι δυνατό ένα dataset να μην είναι διαχωρίσιμο από ένα υπερεπίπεδο. Σε αυτήν την περίπτωση χρησιμοποιείται μία τεχνική, η οποία ονομάζεται

kernelling. Η τεχνική αυτή χρησιμοποιεί τον κατάλληλο πυρήνα και απεικονίζει το dataset σε ένα χώρο με διάσταση μεγαλύτερη κατά 1 από εκείνον στον οποίο βρισκόταν. Η διαδικασία επαναλαμβάνεται μέχρι το dataset να γίνει διαχωρίσιμο από ένα υπερεπίπεδο. Ένα μη διαχωρίσιμο dataset στο χώρο των δύο διαστάσεων παρουσιάζεται στην παρακάτω εικόνα. Σε αυτό, δεν υπάρχει καμία ευθεία που να μπορεί να διαχωρίσει σωστά τις δύο κλάσεις. Έτσι, το dataset απεικονίζεται στο χώρο των τριών διαστάσεων. Τώρα πια, μπορεί να βρεθεί ένα επίπεδο που διαχωρίζει τις δύο κλάσεις με επιτυχία.



Σχήμα 2.28 – Μη διαχωρίσιμο dataset και kernelling

Σε αυτήν την περίπτωση, το κριτήριο βελτιστοποίησης του αλγορίθμου Support Vector Machines εκφράζεται ως εξής:

$$min\left(\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}+C\sum_{i=1}^{n}\xi_i\right)$$

$$y_i\left(\boldsymbol{w}^T\boldsymbol{\varphi}(x_i)+b\right)\geq 1-\xi_i,\xi_i\geq 0$$

όπου φ είναι η συναρτήση που απεικονίζει το training set σε χώρο μεγαλύτερης διάστασης (kernelling).

Τα πλεονεκτήματα του αλγορίθμου SVM σε σχέση με άλλους αλγορίθμους machine learning ή νευρωνικά δίκτυα είναι τα παρακάτω:

- είναι περισσότερο ακριβής.
- η πολύ καλή δυνατότητα γενίκευσης του αλγορίθμου σε μη γραμμικώς διαχωρίσιμα training sets.
- η ανθεκτικότητα του αλγορίθμου σε training sets που περιέχουν θόρυβο.

2.6.2: Bloom filters

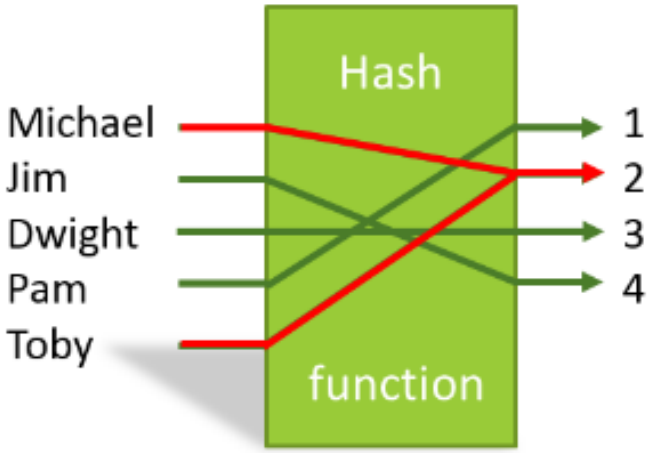
Το **Bloom filter** [38] είναι μία αποδοτική ως προς το χώρο αποθήκευσης **πιθανοτική** δομή δεδομένων (probabilistic data structure), η οποία χρησιμοποιείται για να εξεταστεί εάν ένα στοιχείο ανήκει σε ένα σύνολο ή όχι.

Το bloom filter είναι κατασκευασμένο με τέτοιο τρόπο, ώστε να έχει false positives, αλλά **ποτέ** false negatives. Με άλλα λόγια, εάν η έξοδος ενός bloom filter είναι ότι το στοιχείο ανήκει σε ένα σύνολο, το στοιχείο είναι δυνατόν να ανήκει στο σύνολο αυτό ή και όχι. Ωστόσο, εάν η έξοδος ενός bloom filter είναι ότι το στοιχείο δεν ανήκει σε ένα σύνολο, τότε μπορούμε να είμαστε απόλυτα βέβαιοι ότι το στοιχείο αυτό δεν ανήκει στο σύνολο. Ένα στοιχείο δεν μπορεί να αφαιρεθεί από ένα απλό bloom filter, ενώ, γενικά, όσο μεγαλώνει ο αριθμός των στοιχείων που αποθηκεύονται στο φίλτρο, τόσο μεγαλύτερη είναι η πιθανότητα να εμφανιστεί false positive.

2.6.2.1: Τρόπος λειτουργίας bloom filter

Οι παράμετροι που χαρακτηρίζουν ένα bloom filter και καθορίζουν την απόδοσή του είναι τρεις:

- ***το μέγεθος του bloom filter m*:** το bloom filter είναι ένας πίνακας από bits (bit vector) μεγέθους m, τα οποία αριθμούνται από το 0 μέχρι το m-1.
- ***ο αριθμόν n των στοιχείων*** που είναι αποθηκευμένα στο bloom filter.
- ***ο αριθμός k των hash functions*:** οι συναρτήσεις κατακερματισμού (hash functions) χρησιμοποιούνται για τον προσδιορισμό των θέσεων του φίλτρου στις οποίες θα αποθηκευτεί ένα στοιχείο. Οι hash functions είναι συναρτήσεις που απεικονίζουν δεδομένα αυθαίρετου μεγέθους σε δεδομένα σταθερού μεγέθους. Επισημαίνεται ότι μία hash function με δύο διαφορετικά ορίσματα, μπορεί να επιστρέψει την ίδια έξοδο.

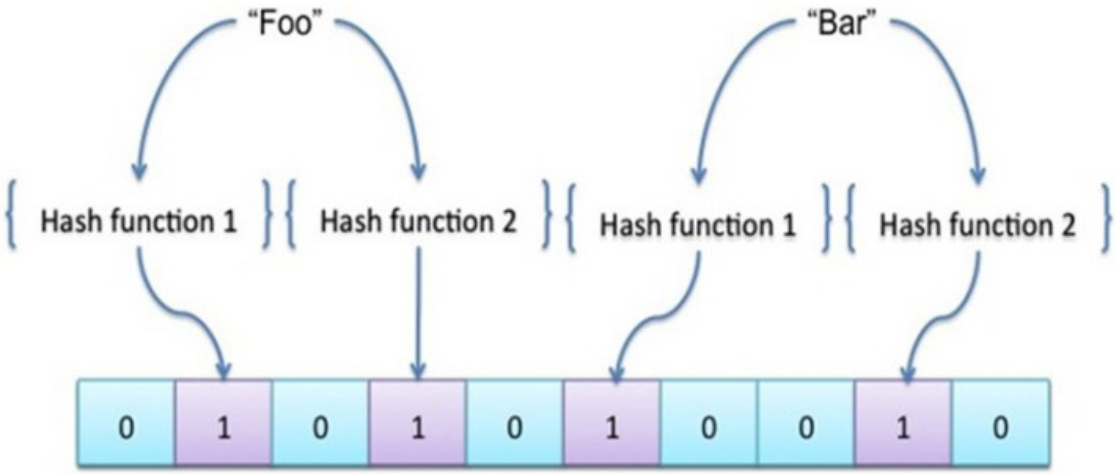


Σχήμα 2.29 – Παράδειγμα λειτουργίας μιας hash function

Αρχικά, όλα τα bits ενός bloom filter τίθενται ίσα με 0. Οι **ενέργειες** που μπορούμε να εφαρμόσουμε πάνω σε ένα bloom filter είναι είτε να προσθέσουμε ένα νέο στοιχείο ή να εξετάσουμε αν ένα στοιχείο είναι αποθηκευμένο στο φίλτρο:

- **Προσθήκη στοιχείου στο φίλτρο**: το στοιχείο που θέλουμε να αποθηκεύσουμε στο φίλτρο δίνεται ως είσοδος στις k συναρτήσεις κατακερματισμού που έχουν επιλεχθεί. Αυτές επιστρέφουν k τιμές, οι οποίες αντιστοιχούν σε κάποια από τα m bits του φίλτρου. Οι θέσεις αυτές αποκτούν την τιμή 1.

- **Εξέταση αν ένα στοιχείο ανήκει στο φίλτρο**: το στοιχείο που μας ενδιαφέρει δίνεται ως είσοδος στις k συναρτήσεις κατακερματισμού που έχουν επιλεχθεί και επιστρέφονται k τιμές που αντιστοιχούν σε κάποιες από τις m θέσεις του φίλτρου. Γίνεται έλεγχος των τιμών των θέσεων αυτών. Εάν όλες οι k θέσεις έχουν την τιμή 1, τότε το στοιχείο **μάλλον** ανήκει στο φίλτρο. Αλλιώς, εάν έστω και μία από τις θέσεις αυτές έχει την τιμή 0, τότε το στοιχείο **σίγουρα** δεν ανήκει στο φίλτρο, καθώς αν είχε αποθηκευτεί σε αυτό, σε όλες τις θέσεις θα είχε αποθηκευτεί η τιμή 1.



Σχήμα 2.30 – προσθήκη στοιχείων σε ένα bloom filter

Η επιλογή των συναρτήσεων κατακερματισμού γίνεται με κριτήριο να είναι ανεξάρτητες μεταξύ τους και ομοιόμορφα κατανεμημένες. Ενδεικτικές συναρτήσεις κατακερματισμού είναι οι murmur, fnv, sha1 και HashMix.

2.6.2.2: Απόδοση bloom filter

Η **πιθανότητα false positive** του bloom filter δίνεται από τον παρακάτω τύπο:

$$(1-e^{-\frac{kn}{m}})^k$$

Επομένως, η πιθανότητα false positive του φίλτρου:

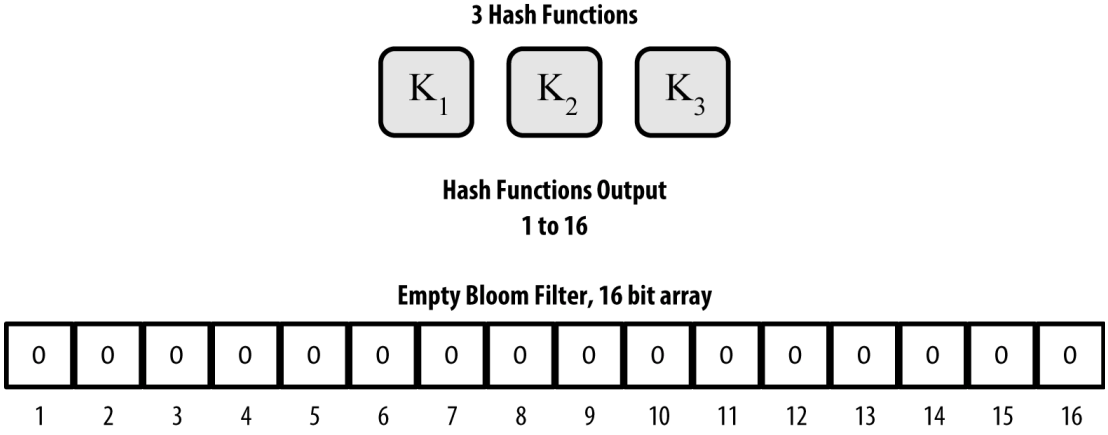
- αυξάνεται όσο αυξάνεται ο αριθμός των στοιχείων που αποθηκεύονται στο φίλτρο.
- μειώνεται όσο αυξάνεται το μέγεθος του φίλτρου.
- μειώνεται μέχρι ένα συγκεκριμένο αριθμό συναρτήσεων κατακερματισμού. Μετά από αυτό το όριο, η πιθανότητα false positive αυξάνεται.

Ο χρόνος αναζήτησης για το αν ένα στοιχείο είναι αποθηκευμένο στο φίλτρο είναι Θ(1), δηλαδή σταθερός.

2.6.2.3: Παράδειγμα λειτουργίας bloom filter

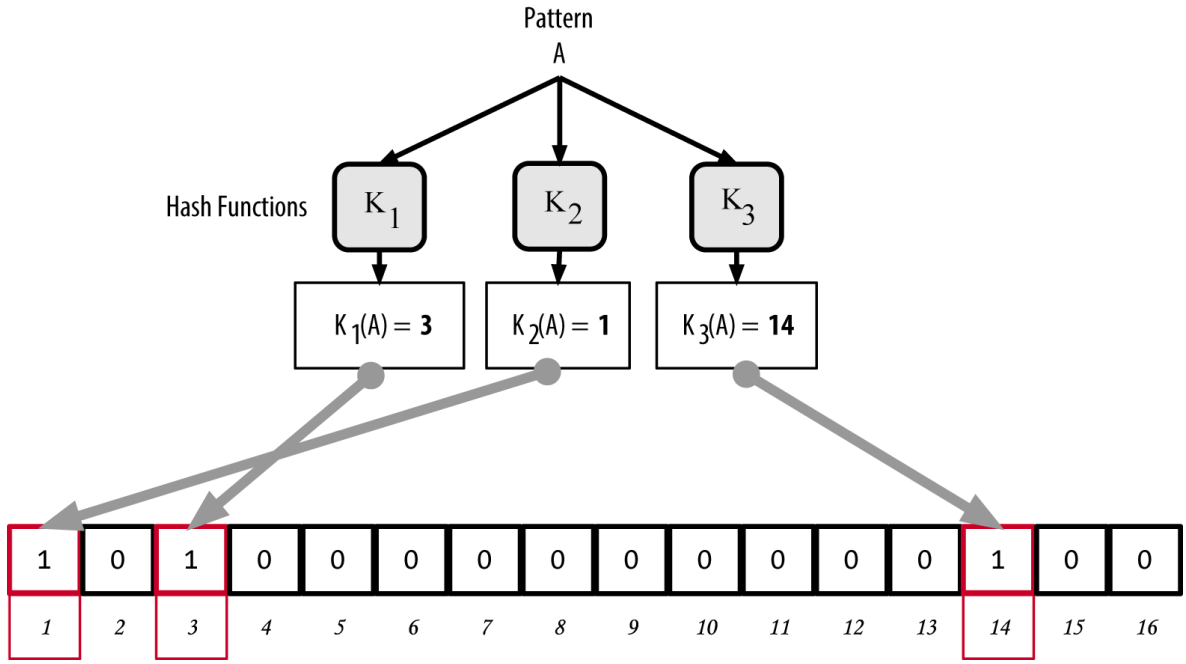
Έστω ένα bloom filter με τις ακόλουθες παραμέτρους [39, 64]:

- μέγεθος φίλτρου m=16 bits. Η αρίθμησή τους στο παράδειγμα θα είναι από το 1 έως το 16.
- αριθμός συναρτήσεων κατακερματισμού k=3. Η έξοδός τους θεωρούμε για απλότητα ότι είναι ένας δεκαδικός αριθμός από το 1 έως το 16.



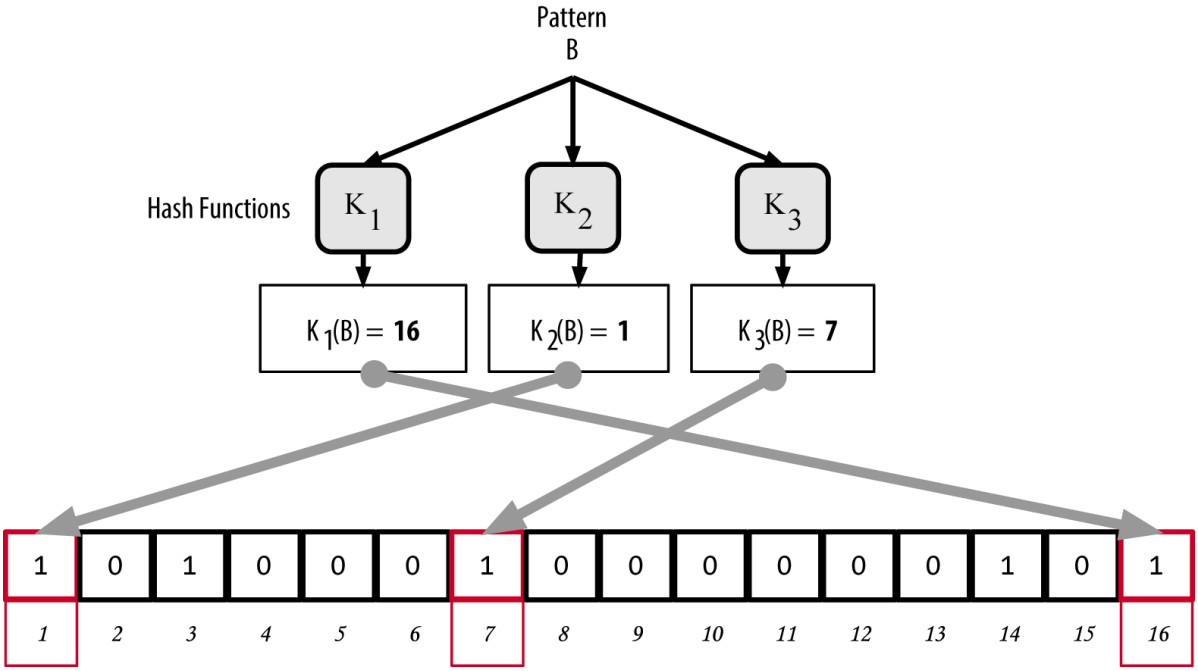
Σχήμα 2.31(α) – Παράδειγμα λειτουργίας bloom filter. Άδειο bloom filter.

Αποφασίζεται να προστεθεί στο φίλτρο ο χαρακτήρας “A”. Οι τρεις συναρτήσεις κατακερματισμού επιστρέφουν τις τιμές 3, 1 και 14 αντίστοιχα. Επομένως, στα bits 1, 3 και 14 του φίλτρου θα αποθηκευτεί η τιμή 1, όπως φαίνεται στην παρακάτω εικόνα σε κόκκινο πλαίσιο.



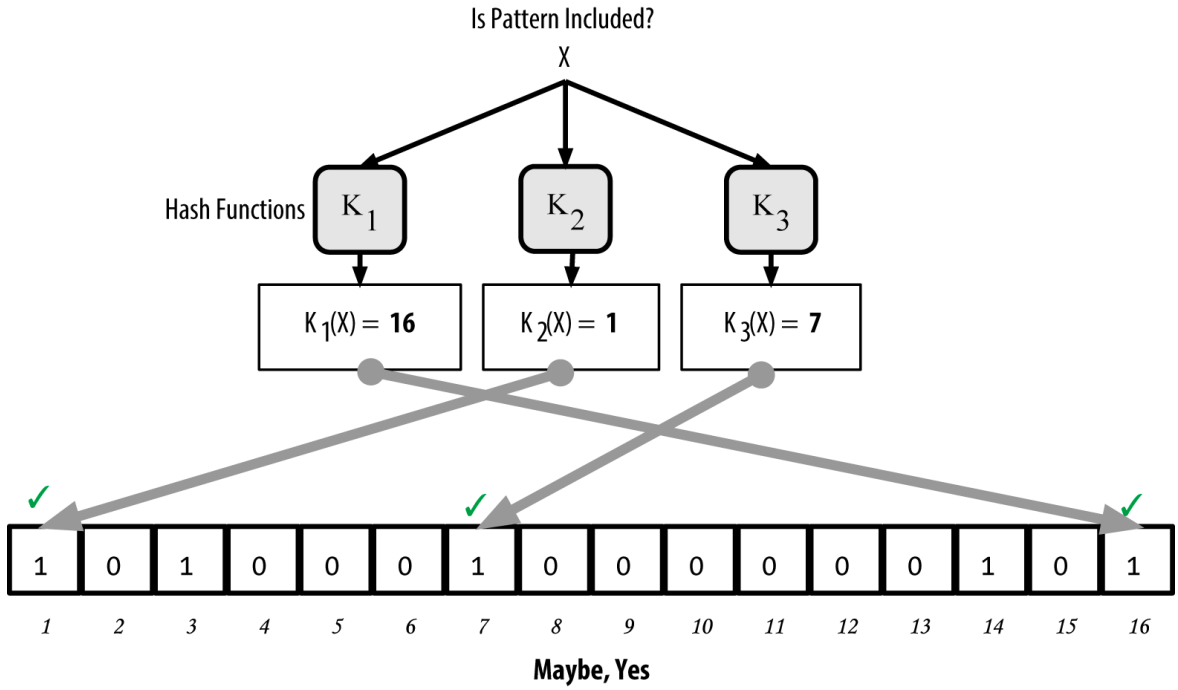
Σχήμα 2.31(β) – παράδειγμα λειτουργίας bloom filter. Προσθήκη στο φίλτρο.

Στη συνέχεια, αποφασίζεται να προστεθεί στο φίλτρο και ο χαρακτήρας “B”. Οι τρεις συναρτήσεις κατακερματισμού επιστρέφουν τις τιμές 1, 7 και 16 αντίστοιχα. Επομένως, στα bits 1, 7 και 16 του φίλτρου θα αποθηκευτεί η τιμή 1. Επισημαίνεται ότι το bit της θέσης 1 είχε ήδη τεθεί ίσο με 1 από την προσθήκη του προηγούμενου στοιχείου.



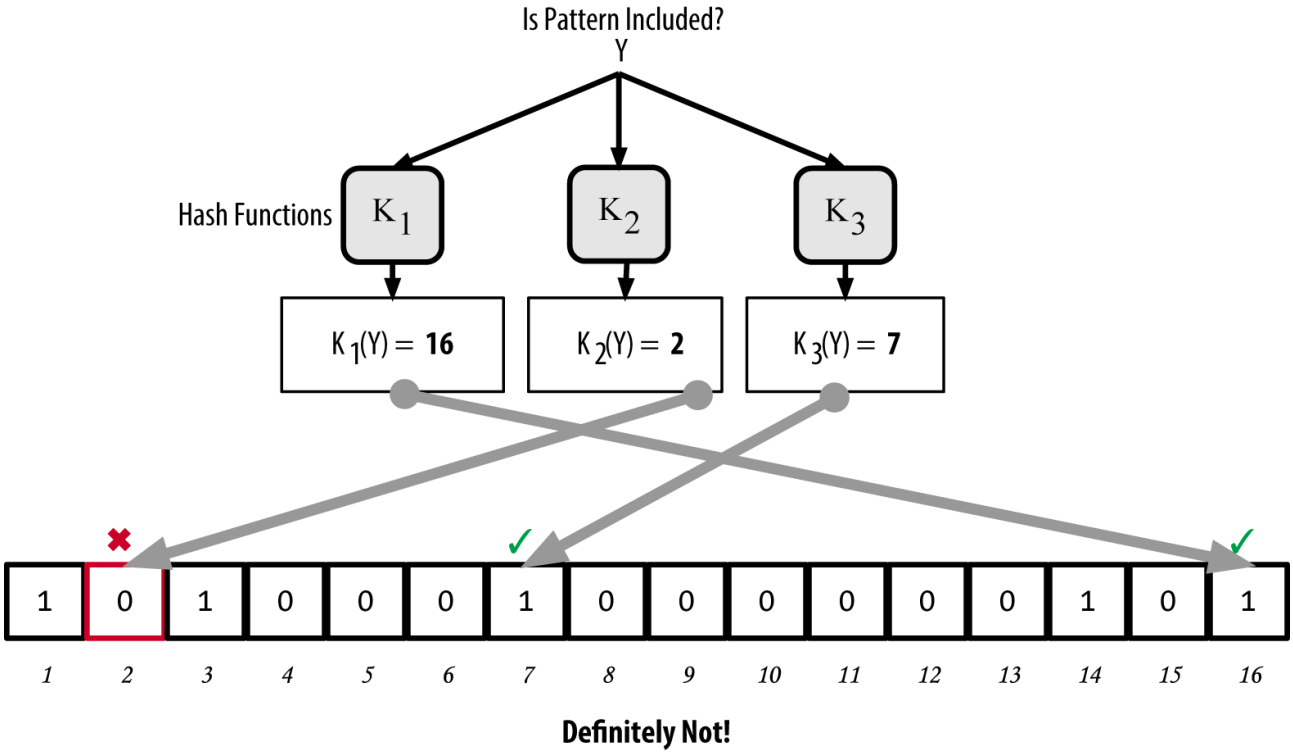
Σχήμα 2.31(γ) – παράδειγμα λειτουργίας bloom filter. Προσθήκη μίας ακόμα λέξης.

Στη συνέχεια, θα εξεταστεί αν ο χαρακτήρας “X” είναι αποθηκευμένος στο φίλτρο. Οι τρεις συναρτήσεις κατακερματισμού επιστρέφουν τις τιμές 1, 7 και 16. Αυτό σημαίνει ότι εάν ο χαρακτήρας “X” είναι αποθηκευμένος στο φίλτρο, θα πρέπει τα bits στις θέσεις 1, 7 και 16 να περιέχουν την τιμή 1. Επειδή όλα τα bits έχουν αποθηκευμένη την τιμή 1, το φίλτρο επιστρέφει ότι ο χαρακτήρας “X” είναι αποθηκευμένος στο φίλτρο. Αυτό είναι φυσικά λάθος. Το συγκεκριμένο false positive οφείλεται στο γεγονός ότι όταν προστέθηκαν στο φίλτρο οι χαρακτήρες “A” και “B”, οι θέσεις 1, 7 και 16 του φίλτρου απέκτησαν την τιμή 1.



Σχήμα 2.31(δ) – παράδειγμα λειτουργίας bloom filter. Παράδειγμα false positive.

Στη συνέχεια, θα εξεταστεί εαν ο χαρακτήρας “Y” είναι αποθηκευμένος στο φίλτρο. Οι τρεις συναρτήσεις κατακερματισμού επιστρέφουν αντίστοιχα τις τιμές 2, 7, 16. Αυτό σημαίνει ότι εάν ο χαρακτήρας “Y” βρίσκεται αποθηκευμένος στο φίλτρο, στις θέσεις 2, 7 και 16 θα βρίσκεται αποθηκευμένη η τιμή 1. Επειδή στη θέση 2 βρίσκεται αποθηκευμένη η τιμή 0, το φίλτρο αποφασίζει με βεβαιότητα ότι ο χαρακτήρας “Y” δε βρίσκεται αποθηκευμένος στο bloom filter.



Σχήμα 2.31(ε) – παράδειγμα λειτουργίας bloom filter. Τα false negatives είναι αδύνατα.

2.6.2.4: Εφαρμογές των bloom filters

Bloom filters έχουν εφαρμοστεί [36] στις παρακάτω περιπτώσεις:

- Η Akamai χρησιμοποιεί bloom filters για να αποτρέψει την αποθήκευση "one-hit wonders" στις προσωρινές μνήμες των δίσκων της, δηλαδή αντικειμένων που ζητούνται μόνο μία φορά από τους χρήστες. Έτσι, αυξάνει τον αριθμό των cache-hits στις προσωρινές μνήμες.
- Τα Google BigTable, Apache Hbase, Apache Cassandra και Postgresql χρησιμοποιούν bloom filters για να μειώσουν τις αναζητήσεις στο δίσκο για ανύπαρκτες γραμμές ή στήλες. Έτσι, οι ερωτήσεις στις βάσεις δεδομένων είναι πολύ πιο γρήγορες.
- Το Google Chrome (web browser) χρησιμοποιεί bloom filters για να εντοπίσει κακόβουλα URLs.
- Το Bitcoin χρησιμοποιεί bloom filters για να επιταχύνει το συγχρονισμό πορτοφολιού (wallet synchronization).
- Το Venti (network storage system) χρησιμοποιεί bloom filters για να εντοπίσει ήδη αποθηκευμένα δεδομένα.
- Το Exim (mail transfer agent) χρησιμοποιεί bloom filters στο μηχανισμό rate-limiting που χρησιμοποιεί.
- Το Medium (online publishing platform) χρησιμοποιεί bloom filters για να αποφεύγει

να προτείνει στο χρήστη άρθρα που έχει ήδη διαβάσει.

2.6.2.5: Επεκτάσεις των bloom filters

Μερικές **επεκτάσεις** της λειτουργίας του απλού bloom filter είναι οι εξής [36]:

- **counting bloom filters:** επέκταση που επιτρέπει την αφαίρεση στοιχείων από το bloom filter, χωρίς αυτό να δημιουργηθεί ξανά από το μηδέν.
- **bloomier bloom filters:** επέκταση που επιτρέπει την αντιστοίχιση μιας τιμής σε κάθε στοιχείο που προστίθεται στο φίλτρο.
- **scalable bloom filters:** επέκταση που επιτρέπει την αποθήκευση απεριόριστων στοιχείων στο bloom filter, ενώ εξασφαλίζει ένα ανώτερο όριο για την πιθανότητα false positive.
- **layered bloom filters:** επέκταση που περιλαμβάνει πολλαπλά στρώματα (layers) από bloom filters και επιτρέπει τη μέτρηση των φορών που ένα στοιχείο έχει προστεθεί στο φίλτρο, ανάλογα με το πόσα layers περιέχουν το στοιχείο.

2.6.3: Ο αλγόριθμος edit distance

Το πρόβλημα που εξετάζει ο αλγόριθμος edit distance [40] είναι η μετατροπή ενός αλφαριθμητικού σε ένα άλλο με όσο το δυνατόν λιγότερες ενέργειες πάνω στα αλφαριθμητικά. Οι επιτρεπόμενες ενέργειες είναι η προσθήκη χαρακτήρων, η διαγραφή χαρακτήρων και η αντικατάσταση χαρακτήρων.

Για παράδειγμα, η edit distance ανάμεσα στις λέξεις FOOD και MONEY ισούται με 4 χαρακτήρες καθώς η συντομότερη μετατροπή της λέξης FOOD στη λέξη MONEY απαιτεί 4 από τις παραπάνω ενέργειες:

- αντικατάσταση του χαρακτήρα F με το χαρακτήρα M.
- αντικατάσταση του χαρακτήρα O με το χαρακτήρα N.
- προσθήκη του χαρακτήρα E.
- αντικατάσταση του D με το χαρακτήρα Y.

Σχήμα 2.32 – παράδειγμα εκτέλεσης του αλγορίθμου edit distance

Στη συνέχεια, θα περιγραφτεί ο αλγόριθμος edit distance που βασίζεται στην τεχνική του δυναμικού προγραμματισμού. Ο **δυναμικός προγραμματισμός (dynamic programming)** [41] είναι μία μέθοδος επίλυσης πολύπλοκων προβλημάτων, που διασπάει ένα πρόβλημα σε απλούστερα υποπροβλήματα, επιλύει το καθένα μόνο μία φορά και αποθηκεύει το αποτέλεσμά του σε έναν πίνακα. Την επόμενη φορά που θα εμφανιστεί το ίδιο

υποπρόβλημα, δε θα χρειαστεί να επιλυθεί ξανά, καθώς η λύση του έχει υπολογιστεί ήδη και έχει αποθηκευτεί στον πίνακα. Έτσι, μειώνεται ο υπολογιστικός χρόνος σε βάρος του αποθηκευτικού χώρου.

Συμβολίζοντας με A[1…..n] το πρώτο αλφαριθμητικό μήκους n και με B[1…..m] το δεύτερο αλφαριθμητικό μήκους m, ο αναδρομικός τύπος υπολογισμού της edit distance ανάμεσα στα δύο αλφαριθμητικά είναι:

$$Edit(A[1..m], B[1..n]) = \min \left\{ \begin{array}{l} Edit(A[1..m-1], B[1..n]) + 1 \\ Edit(A[1..m], B[1..n-1]) + 1 \\ Edit(A[1..m-1], B[1..n-1]) + [A[m] \neq B[n]] \end{array} \right\}$$

Σχήμα 2.33 – Αλγόριθμος edit distance

δηλαδή ο μικρότερος αριθμός από τους τρεις κλάδους. Έστω ότι βάζουμε τις δύο λέξεις τη μία κάτω από την άλλη, χωρίς κενά ανάμεσα στους χαρακτήρες. Υπάρχουν τρεις δυνατότητες για τις τελευταίες στήλες των αλφαριθμητικών:

- Στην πρώτη περίπτωση, αν το A είναι μεγαλύτερο από το B, η edit distance των A και B είναι 1 συν την edit distance ανάμεσα στο B και στο A, χωρίς τον τελευταίο του χαρακτήρα, καθώς έχουμε φροντίσει ήδη για αυτόν.
- Η δεύτερη περίπτωση είναι η συμμετρική της πρώτης.
- Στην τρίτη περίπτωση, και τα δύο αλφαριθμητικά έχουν χαρακτήρες στην τελευταία στήλη. Τότε, αν οι χαρακτήρες είναι ίδιοι, η edit distance των A και B έχει ίδια τιμή με την edit distance των αλφαριθμητικών, χωρίς τους τελευταίους χαρακτήρες τους. Αλλιώς, η edit distance των A και B είναι ίση με την edit distance των αλφαριθμητικών, χωρίς τους τελευταίους χαρακτήρες τους συν μία μονάδα, λόγω της μετατροπής του ενός χαρακτήρα στον άλλον.

Οι οριακές συνθήκες του αλγορίθμου είναι οι εξής:

- η edit distance δύο κενών αλφαριθμητικών είναι μηδενική.
- η edit distance ενός αλφαριθμητικού με ένα κενό αλφαριθμητικό είναι ίση με το μήκος του μη κενού αλφαριθμητικού.

Η χρονική πολυπλοκότητα του παραπάνω αλγορίθμου είναι $\Theta(n^2)$ και η χωρική πολυπλοκότητα είναι $\Theta(n^2)$.

Χαρακτηριστικό παράδειγμα εφαρμογής του αλγορίθμου αυτού είναι η εξακρίβωση των **κινήτρων** πίσω από ένα ερώτημα DNS για έναν ανύπαρκτο στη ζώνη host [59]. Εάν η edit distance του ονόματος της ερώτησης σε σύγκριση με τα ονόματα των hosts που περιλαμβάνονται στη ζώνη είναι μεγάλη, τότε συνάγεται ότι υπάρχει δόλος πίσω από το ερώτημα αυτό. Αλλιώς, μπορεί να θεωρηθεί ότι η ερώτηση αυτή είναι αποτέλεσμα κάποιου λάθους πληκτρολόγησης.

3 Παρουσίαση μηχανισμού

Η μεγάλη σημασία που κατέχει το DNS για την εύρυθμη λειτουργία του σύγχρονου διαδικτύου καθιστά αναγκαία τη λήψη μέτρων για την προστασία του. Η υποδομή του DNS αποτελεί συχνό στόχο των επιτιθέμενων, καθώς τα τρωτά σημεία του πρωτοκόλλου παρέχουν τη δυνατότητα εκτέλεσης πολύ αποτελεσματικών επιθέσεων.

Ο μηχανισμός που υλοποιήθηκε στα πλαίσια της παρούσης διπλωματικής εργασίας έχει ως στόχο την έγκαιρη ανίχνευση και αποτελεσματική αντιμετώπιση της επίθεσης **DNS water torture** σε περιβάλλον Δικτύου Οριζομένου από Λογισμικό (SDN). Ωστόσο, επισημαίνεται ότι ο μηχανισμός μπορεί να προσαρμοστεί και σε παραδοσιακά δίκτυα.

3.1: Ορισμός Προβλήματος

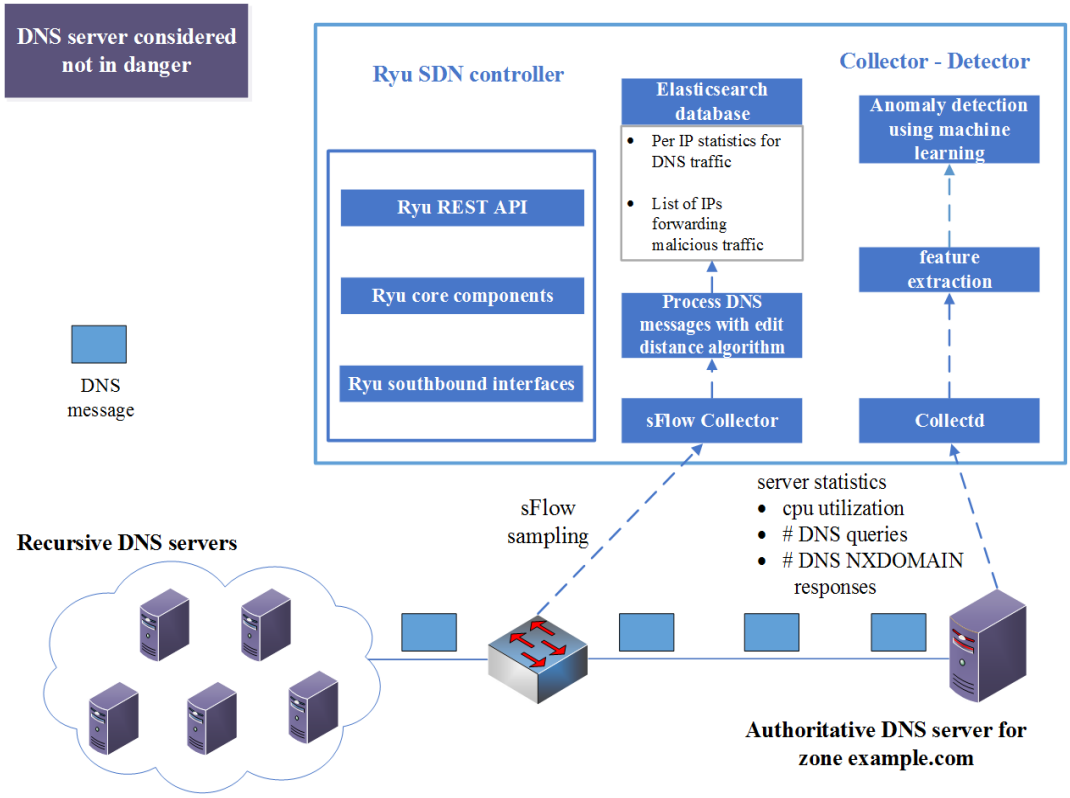
Ως θύμα της επίθεσης επιλέχθηκε ο **authoritative** DNS server της ζώνης example.com, καθώς τελικό στόχο των επιθέσεων DNS water torture αποτελούν authoritative και όχι recursive servers.

Ο server αυτός, λοιπόν, πρόκειται να δεχτεί επίθεση DNS water torture. Αν αφεθεί να λειτουργήσει απροστάτευτος, σύντομα, η συνεχής ροή μεγάλου όγκου ερωτημάτων DNS προς εκείνον, θα οδηγήσει στον κορεσμό την ισχύ του επεξεργαστή του.

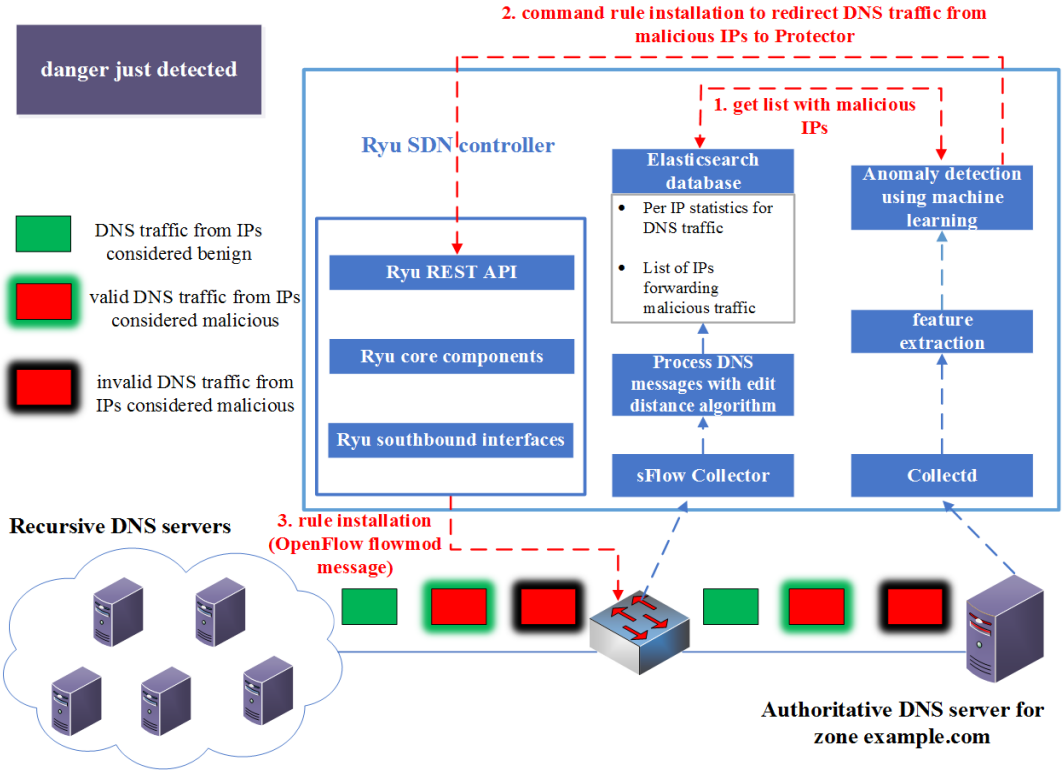
Ως αποτέλεσμα, ο server θα καταστεί ανίκανος να απαντά στα ουσιαστικά ερωτήματα των πελατών του, εμποδίζοντας την πρόσβασή τους σε σημαντικούς δικτυακούς πόρους και θα κινδυνεύσει να χαρακτηριστεί ως μη προσβάσιμος, γεγονός που θα επιφέρει προβλήματα, ακόμα και αφού επανέλθει στην ομαλή λειτουργία του.

3.2: Παρουσίαση αρχιτεκτονικής

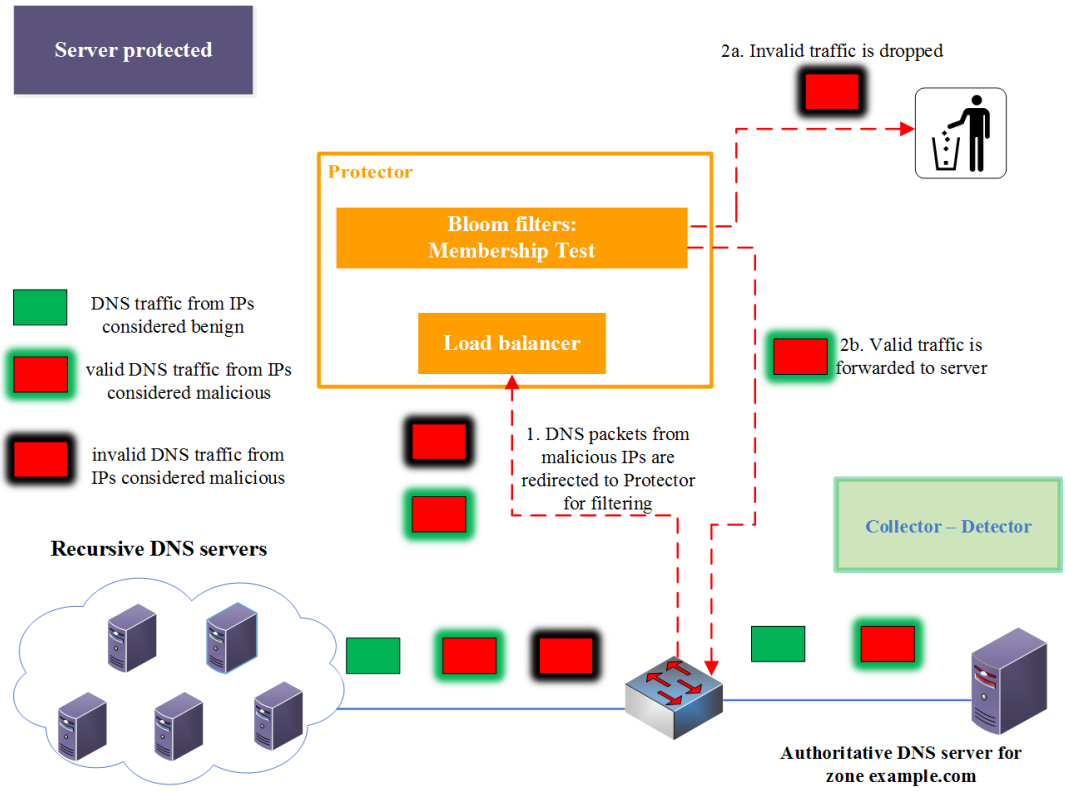
Η αρχιτεκτονική διάταξη της υλοποίησης παρουσιάζεται στα παρακάτω σχήματα:



Σχήμα 3.1 – Η αρχιτεκτονική του μηχανισμού που υλοποιήθηκε (περίπτωση Α: ο εξυπηρετητής θεωρείται ότι δεν κινδυνεύει).



Σχήμα 3.2 – Η αρχιτεκτονική του μηχανισμού που υλοποιήθηκε (περίπτωση Β: μόλις εντοπίστηκε ότι ο εξυπηρετητής κινδυνεύει και λαμβάνονται μέτρα για την προστασία του).



Σχήμα 3.3 – Η αρχιτεκτονική του μηχανισμού που υλοποιήθηκε (Περίπτωση Γ: ο εξυπηρετητής είναι πλέον προστατευμένος).

Ο μηχανισμός που υλοποιήθηκε περιλαμβάνει 4 βασικές οντότητες:

- τον **authoritative DNS server** της ζώνης example.com, ο οποίος είναι το θύμα της επίθεσης DNS water torture.
- τον **Collector-Detector**, ο οποίος αποτελεί το μηχανισμό που υλοποιεί τις απαραίτητες διαδικασίες για τον εντοπισμό μη φυσιολογικής κίνησης DNS που κατευθύνεται στον εξυπηρετητή και τη λήψη μέτρων προστασίας εάν χρειαστούν. Παράλληλα, στη μονάδα αυτή περιλαμβάνεται και ο controller του δικτύου SDN.
- τον **Protector**, δηλαδή τη μονάδα στην οποία γίνεται το φιλτράρισμα των ερωτημάτων DNS όταν θεωρηθεί ότι ο εξυπηρετητής βρίσκεται σε κίνδυνο.
- το **OpenFlow switch**, το οποίο πραγματοποιεί τη μεταγωγή των πακέτων που διακινούνται εντός του τοπικού δικτύου.

Κάτω από συνθήκες φυσιολογικής λειτουργίας, ο **authoritative DNS server** της ζώνης example.com απαντάει στα ερωτήματα DNS που καταφθάνουν σε αυτόν από πελάτες στο εξωτερικό του τοπικού δικτύου. Εάν το όνομα που περιλαμβάνεται στο ερώτημα DNS περιέχεται στα αρχεία ζώνης του, τότε θα επιστρέψει στον υπολογιστή που διατύπωσε την

ερώτηση τη διεύθυνση IP που αντιστοιχεί στο όνομα αυτό (ο μηχανισμός θα περιοριστεί μόνο σε ερωτήσεις για εγγραφές τύπου A). Αλλιώς, θα του επιστρέψει μία απάντηση DNS με τον κωδικό NXDOMAIN, δηλαδή ανύπαρκτο domain για να δείξει ότι δε γνωρίζει την απάντηση σε αυτό το ερώτημα. Επισημαίνεται ότι ο μηχανισμός που υλοποιήθηκε δε λαμβάνει υπόψη του και τις περιπτώσεις άλλων κωδικών DNS, όπως είναι για παράδειγμα ο κωδικός SRVFAIL.

Στο μεταξύ, η μονάδα **Collector – Detector** συγκεντρώνει από το δίκτυο και τον εξυπηρετητή τις απαραίτητες πληροφορίες για την αναγνώριση μη φυσιολογικής κίνησης και την ενεργοποίηση του μηχανισμού προστασίας σε περίπτωση που χρειαστεί. Για να το πετύχει, επιστρατεύει δύο μηχανισμούς:

- Ο DNS server αποστέλλει συνεχώς στον Collector – Detector πληροφορίες που σχετίζονται με τη λειτουργία του και την κίνηση που εξυπηρετεί, χρησιμοποιώντας το λογισμικό **Collectd**. Ο DNS server έχει το ρόλο του Collectd client, καθώς στέλνει δεδομένα, ενώ ο Collector – Detector, στον οποίο συλλέγονται τα στατιστικά δεδομένα, έχει το ρόλο του Collectd server.

  Η πληροφορία που συλλέγεται με αυτόν τον τρόπο είναι το ποσοστό χρήσης του επεξεργαστή του DNS server, ο αριθμός των ερωτημάτων DNS που δέχεται, καθώς και το πλήθος των αποκρίσεων DNS στις οποίες ο DNS server απάντησε με τον κωδικό NXDOMAIN. Οι πληροφορίες αυτές συγκεντρώνονται στον Collector – Detector και αποθηκεύονται σε **rrdfiles**. Αυτά αποτελούν μία αποδοτική ως προς τη μνήμη επιλογή, καθώς ο Collector – Detector εξετάζει τα δεδομένα που συλλέγει για ένα συγκεκριμένο χρονικό παράθυρο στο παρελθόν και όχι για όλο το διάστημα λειτουργίας του εξυπηρετητή DNS.

- Ταυτόχρονα, με το πρωτόκολλο **sFlow** δειγματοληπτείται η κίνηση που διέρχεται μέσα από το OpenFlow switch με ρυθμό δειγματοληψίας 1 προς 128, δηλαδή έναν ικανοποιητικά γρήγορο ρυθμό. Τα πακέτα, που δειγματοληπτούνται με τυχαίο τρόπο, καταφθάνουν στον Collector-Detector ενθυλακωμένα μέσα σε UDP δεδομενογράμματα και συγκεντρώνονται από τον **συλλέκτη sFlow**, ώστε να εξαχθούν πληροφορίες από αυτά.

  Αρχικά, εξετάζεται εάν το δείγμα πρόκειται για μήνυμα DNS (είτε ερώτημα είτε απάντηση). Στη συνέχεια, το όνομα που περιλαμβάνεται στο question section της επικεφαλίδας του πακέτου συγκρίνεται με όλα τα ονόματα που περιλαμβάνονται στο αρχείο ζώνης του DNS server, χρησιμοποιώντας τον αλγόριθμο σύγκρισης αλφαριθμητικών **edit distance**. Ο αλγόριθμος αυτός προτιμήθηκε έναντι άλλων παρόμοιων, διότι μπορεί να λειτουργήσει χωρίς να προηγηθεί κάποια εκπαίδευσή του. Με αυτόν τον τρόπο, θα διαπιστωθεί, σε πρώτο στάδιο, αν το όνομα αυτό περιέχεται στο αρχείο. Σε δεύτερο στάδιο, εάν το όνομα είναι ανύπαρκτο, θα εξεταστεί εάν πρόκειται για ένα λάθος απροσεξίας του χρήστη που διατύπωσε το

ερώτημα ή για ένα σκόπιμο λάθος, δηλαδή παράχθηκε με τυχαίες διαδικασίες και αποτελεί τμήμα μίας κακόβουλης δραστηριότητας.

Οι πληροφορίες αυτές αποθηκεύονται σε μία **NoSQL βάση δεδομένων**, ώστε να εξασφαλίζεται ότι η ανάκτηση μεγάλου όγκου δεδομένων σε περίπτωση που διαγνωστεί κίνδυνος θα είναι γρήγορη. Με δείκτη (index) την IP που διατυπώνει το ερώτημα ή στην οποία αποστέλεται η απάντηση (εξετάζονται ως κοινή περίπτωση), διατηρείται μία εγγραφή στη βάση δεδομένων. Σε αυτήν αποθηκεύονται ο αριθμός των πακέτων που δειγματοληπτήθηκαν, πόσα από αυτά περιελάμβαναν ανύπαρκτα ονόματα (NXDOMAIN), η μέση edit distance των NXDOMAIN ονομάτων και ένας χαρακτηρισμός για τον αν η IP αυτή θεωρείται επικίνδυνη ή όχι.

Δεδομένου ότι ο αριθμός των απαντήσεων με κωδικό NXDOMAIN κυμαίνεται, συνήθως, από 10% έως 15% των συνολικών απαντήσεων DNS, μία IP θεωρείται επικίνδυνη εάν ο μέσος όρος NXDOMAIN ονομάτων για την IP αυτή υπερβαίνει το 15%. Παράλληλα, τα λάθη απροσεξίας που μπορεί να συμβούν κατά την πληκτρολόγηση ενός ονόματος οφείλονται, συνήθως, στην προσθήκη ενός επιπλέον γράμματος, στην παράλειψη ενός ή στη λανθασμένη σειρά ανάμεσα σε δύο γειτονικά γράμματα. Θεωρούμε, λοιπόν, ότι εάν το αποτέλεσμα του αλγορίθμου edit distance είναι μεγαλύτερο του 2, τότε το λάθος κρίνεται σκόπιμο. Έτσι, διευθύνσεις IP με μέσο ποσοστό NXDOMAIN απαντήσεων μεγαλύτερο του 15% ή με μέση edit distance μεγαλύτερη του 2 χαρακτηρίζονται ως **επικίνδυνες**.

Είναι απαραίτητο να επισημανθεί ότι αφού ο μηχανισμός δειγματοληψίας δεν κάνει διάκριση ανάμεσα σε DNS ερωτήσεις και απαντήσεις, είναι δυνατό να δειγματοληπτηθούν και η ερώτηση και η απάντηση της ίδιας δοσοληψίας. Ωστόσο, η πιθανότητα να συμβεί κάτι τέτοιο είναι πολύ μικρή και, ακόμα και αν συμβεί, κρίνεται ότι η επίδρασή της στο μηχανισμό αναγνώρισης ανώμαλης κίνησης θα είναι αμελητέα.

Συγκεντρώνοντας, λοιπόν, τις πληροφορίες αυτές, ο Collector – Detector χρησιμοποιεί έναν **αλγόριθμο μηχανικής μάθησης**, συγκεκριμένα τον Support Vector Machines (SVM), για να αποφανθεί εάν ο εξυπηρετητής DNS βρίσκεται σε κίνδυνο ή εάν η κίνηση που διακινείται στο δίκτυο είναι φυσιολογική. Τα features, τα οποία δίνονται ως είσοδοι στον αλγόριθμο είναι τα παρακάτω:

- Το μέσο ποσοστό χρήσης του επεξεργαστή του server στα τέσσερα τελευταία δευτερόλεπτα λειτουργίας του, δεδομένα που συλλέχθηκαν από το Collectd (feature 1). Το χρονικό αυτό διάστημα καθορίστηκε έπειτα από μετρήσεις πάνω στο μηχανισμό. Μικρές τιμές του οδηγούν σε βιαστικές αποφάσεις, λόγω στιγμιαίων αυξήσεων του ποσοστού χρήσης του επεξεργαστή, ενώ μεγάλες τιμές του έχουν ως αποτέλεσμα την καθυστέρηση της ενεργοποίησης του μηχανισμού.

- Ο λόγος των συνολικών NXDOMAIN απαντήσεων του server προς το λόγο των συνολικών ερωτήσεων DNS, όπως συλλέχθηκαν από το Collectd χωρίς δειγματοληψία για το χρονικό διάστημα των τελευταίων 15 δευτερολέπτων. Θεωρούμε ότι αυτό το χρονικό διάστημα είναι αρκετό για να αναγνωριστεί έγκαιρα μη ομαλή κίνηση και να αποφευχθούν βιαστικές αποφάσεις (feature 2).

Η επιλογή ενός αλγορίθμου μηχανικής μάθησης επιστρέπει να καθοριστούν με ευέλικτο τρόπο τα όρια απόφασης του μηχανισμού, ενσωματώνοντας την εμπειρία του διαχειριστή του δικτύου, αλλά και δεδομένα από προηγούμενες περιπτώσεις παρατήρησης της επίθεσης.

Ο αλγόριθμος machine learning εξετάζει, συνεχώς, τις παραπάνω τιμές. Εάν λάβει την απόφαση ότι ο εξυπηρετητής κινδυνεύει, ανακτά από τη βάση δεδομένων τις διευθύνσεις IP που έχουν χαρακτηριστεί ως επικίνδυνες. Έπειτα, δίνει εντολή στον SDN controller να εγκαταστήσει στο OpenFlow switch κανόνες για τα εισερχόμενα στο δίκτυο μηνύματα DNS που προέρχονται από τις IP αυτές. Στο εξής, τα πακέτα αυτά θα οδηγούνται πρώτα στη μονάδα αντιμετώπισης της επίθεσης (**Protector**) για χρόνους idle_timeout = 120 sec και hard_timeout = 240 sec. Οι διευθύνσεις IP που δεν έχουν χαρακτηριστεί ως επικίνδυνες, θα συνεχίσουν να οδηγούνται απ’ευθείας στο DNS server. Στο μεταξύ, ο αλγόριθμος machine learning θα συνεχίσει να εξετάζει εάν ο εξυπηρετητής εξακολουθεί να βρίσκεται σε κίνδυνο. Θα συνεχίσει, λοιπόν, να εγκαθιστά δυναμικά κανόνες για όσες IP παραμένουν επικίνδυνες ή για όσες νέες IP χαρακτηριστούν ως επικίνδυνες μέχρι να αποκατασταθεί η φυσιολογική λειτουργία στο δίκτυο.

Έτσι, οι επικίνδυνες ροές πακέτων οδηγούνται πρώτα στον Protector. Η μονάδα αυτή περιλαμβάνει έναν **εξισορροπητή φόρτου (load balancer)**, ο οποίος κατανέμει τα εισερχόμενα μηνύματα DNS ισότιμα και εναλλάξ (μέθοδος round robin) σε ένα σύνολο από **bloom filters.** Σε αυτά έχουν αποθηκευτεί από την αρχή λειτουργίας του δικτύου τα περιεχόμενα του αρχείου ζώνης του DNS server.

Για κάθε ερώτημα που καταφθάνει εξετάζεται εάν το όνομα που περιλαμβάνει βρίσκεται αποθηκευμένο στο φίλτρο. Εάν η απάντηση είναι αρνητική, τότε το πακέτο **απορρίπτεται**, χωρίς να σταλθεί κάτι ως απάντηση. Εάν είναι θετική, τότε ο μηχανιμός **προωθεί** το ερώτημα στον DNS server. Έτσι, ο DNS server θα πάψει να υφίσταται καταιγισμό μηνυμάτων και το ποσοστό χρήσης του επεξεργαστή του θα μειωθεί.

4 Ανάλυση υλοποίησης

Στο κεφάλαιο αυτό θα παρουσιαστούν οι λεπτομέρειες υλοποίησης των επιμέρους τμημάτων του μηχανισμού και θα επεξηγηθεί αναλυτικά ο κώδικας που υλοποιήθηκε.

Στις ενότητες που ακολουθούν θα παρουσιαστεί:

- η υλοποίηση του DNS server.
- η επιλογή του controller και τα components του που χρησιμοποιήθηκαν.
- η υλοποίηση του Collector – Detector που αναλύεται στο μηχανισμό συλλογής δεδομένων από την κίνηση του δικτύου και στο μηχανισμό εντοπισμού κινδύνου.
- η υλοποίηση του Protector που αναλύεται στον load balancer και στην αντιμετώπιση της επίθεσης με bloom filters.

Αρχικά, όμως, θα αποσαφηνιστούν δύο σημαντικές λεπτομέρειες που σχετίζονται με τα παραπάνω:

- τι είναι ένα REST API.
- τι είναι το πρότυπο JSON.

4.1: REST APIs και JSON

Ένα **REST (Representational State Transfer) API** [42] ορίζει ένα σύνολο από συναρτήσεις που μπορούν να χρησιμοποιηθούν για την πρόσβαση και διαχείριση δικτυακών πόρων (web resources) σε μορφή κειμένου, χρησιμοποιώντας το πρωτόκολλο HTTP και τις μεθόδους του, δηλαδή:

- μέθοδος **GET** για τη λήψη δεδομένων από τον πόρο που ορίζει το URI.
- μέθοδος **POST** για την αποστολή δεδομένων προς τον πόρο που ορίζει το URI.
- μέθοδος **PUT** για την αποστολή δεδομένων προς τον πόρο που ορίζει το URI με αντικατάσταση όσων υπάρχουν ήδη.
- μέθοδος **DELETE** για τη διαγραφή του πόρου που ορίζει το URI.

Η αποστολή αιτημάτων HTTP σε περιβάλλον UNIX πραγματοποιείται με την εντολή **curl**.

Το πρότυπο **JSON (JavaScript Object Notation)** [43, 44] αποτελεί έναν τρόπο μετάδοσης πληροφορίας σε μορφή ευανάγνωστη και εύκολα προσβάσιμη από τον άνθρωπο, μεταφέροντας ζεύγη ιδιοτήτων και τιμών (attribute - value pairs), πίνακες με δεδομένα (array data types) ή οποιαδήποτε άλλη serializable τιμή. Το πρότυπο JSON είναι **ανεξάρτητο** από τη γλώσσα προγραμματισμού που χρησιμοποιεί είτε ο αποστολέας είτε ο παραλήπτης της πληροφορίας.

Στο παρακάτω σχήμα παρουσιάζεται ένα παράδειγμα εγκατάστασης ενός flow rule σε ένα OpenFlow switch μέσω του REST API που παρέχει ο Ryu SDN controller. Οι πληροφορίες που συνιστούν τον κανόνα μεταφέρονται ακολουθώντας το πρότυπο JSON.

```
curl -X POST -d '{
  "dpid": 1,
  "cookie": 1,
  "cookie_mask": 1,
  "table_id": 0,
  "idle_timeout": 30,
  "hard_timeout": 30,
  "priority": 11111,
  "flags": 1,
  "match":{
    "in_port":1
  },
  "actions":[
    {
      "type":"OUTPUT",
      "port": 2
    }
  ]
}' http://localhost:8080/stats/flowentry/add
```

Σχήμα 4.1 – παράδειγμα εγκατάστασης flow rule μέσω REST API με μεταφορά δεδομένων σε μορφή JSON

4.2: DNS server

Ο DNS server υλοποιήθηκε χρησιμοποιώντας το λογισμικό **BIND** [45], που αποτελεί δημοφιλές λογισμικό κατασκευής DNS εξυπηρετητών σε περιβάλλον UNIX. Η έκδοση του λογισμικού που επιλέχθηκε είναι η ένατη (BIND9).

Επιπρόσθετα, για την αποστολή δεδομένων σχετικών με τη λειτουργία του DNS server εγκαταστάθηκε ο UNIX daemon **Collectd** και ενεργοποιήθηκαν τα παρακάτω plugins:

- cpu
- dns
- network, με την επιλογή client, ώστε τα δεδομένα να αποστέλλονται πάνω από το δίκτυο στον Collectd server.

Η αποστολή των πληροφοριών από τον εξυπηρετητή στον Collector - Detector επιλέχθηκε να πραγματοποιείται κάθε **ένα δευτερόλεπτο** (παράμετρος interval). Η αιτία επιλογής του συγκεκριμένου χρονικού διαστήματος είναι ότι το Collectd αποστέλλει μέσους όρους για τις τιμές που συλλέγει ανά δευτερόλεπτο. Επιλέγοντας, λοιπόν, διάστημα του ενός δευτερολέπτου, το Collectd θα αποστέλλει ακριβείς τιμές και όχι μέσους όρους. Η επιλογή

αυτή δεν επιβαρύνει τη λειτουργία του εξυπηρετητή, καθώς τα μηνύματα του Collectd ενθυλακωμένα σε δεδομενογράμματα UDP είναι μικρά σε μέγεθος.

4.3: SDN Controller

Ως controller του δικτύου επιλέχθηκε ο **Ryu SDN controller** [46, 47], ο οποίος είναι υλοποιημένος στη γλώσσα Python [48]. Ο Ryu προτιμήθηκε έναντι άλλων SDN controllers, κυρίως, λόγω του REST API που προσφέρει και με το οποίο, η εγκατάσταση κανόνων στο OpenFlow switch με δυναμικό τρόπο επιτυγχάνεται με μεγάλη ευκολία. Η περιγραφή των κανόνων αυτών γίνεται σε πρότυπο JSON.

Τα components που τρέχει ο Ryu controller του μηχανισμού είναι τα παρακάτω:

- ryu-manager
- app.simple_switch.py
- app.ofctl_rest.py

4.3.1: component ryu-manager

Είναι το βασικό component του Ryu SDN controller και αποτελεί προαπαιτούμενο για την εκτέλεση οποιουδήποτε άλλου component.

4.3.2: component app.simple_switch.py

Είναι το component εκείνο του Ryu controller που υλοποιεί τη λειτουργία ενός layer 2 learning switch. Δεν πραγματοποιήθηκε καμία αλλαγή στον κώδικα του component αυτού.

Βασική μέθοδος του component είναι η **_packet_in_handler**, η οποία είναι ένας event handler που ενεργοποιείται όταν καταφθάνει στον controller ένα πακέτο, το οποίο το OpenFlow switch δε γνωρίζει πώς να διαχειριστεί (μήνυμα packet-in).

Ο controller διατηρεί ένα **dictionary** με κλειδιά την ταυτότητα του switch και τη διεύθυνση MAC των συνδεδεμένων στο switch υπολογιστών, στο οποίο καταγράφει τη θύρα του switch στην οποία βρίσκεται ο υπολογιστής με τη συγκεκριμένη διεύθυνση MAC. Όταν καταφθάνει ένα πακέτο packet-in στον controller από ένα switch, εκείνος συμβουλεύεται το dictionary για το συγκεκριμένο switch και ελέγχει εάν γνωρίζει τη θύρα στην οποία βρίσκεται ο υπολογιστής που έχει τη διεύθυνση MAC προορισμού του πακέτου. Εάν ναι, τότε προωθεί το πακέτο στη θύρα αυτή και εγκαθιστά ένα νέο κανόνα στο switch για να αποφύγει μελλοντικά μηνύματα packet-in για αυτόν τον προορισμό. Εάν όχι, πλημμυρίζει το πακέτο σε όλες τις θύρες (εκτός από εκείνη από την οποία προήλθε το πακέτο), μαθαίνοντας παράλληλα την αντιστοιχία θύρας προέλευσης – MAC προέλευσης, ώστε να είναι σε θέση να εγκαταστήσει κανόνες για μελλοντικά πακέτα packet-in.

4.3.3: component app.ofctl_rest.py

Είναι το component εκείνο του Ryu controller που παρέχει REST APIs για τη λήψη ή την ενημέρωση της κατάστασης και των στατιστικών ενός switch. Δεν πραγματοποιήθηκε καμία αλλαγή στον κώδικα του component αυτού.

Ο μηχανισμός που υλοποιήθηκε αξιοποιεί το REST API του Ryu controller για την προσθήκη flow rules στο switch. Το URI που προσδιορίζει αυτήν την περίπτωση είναι το **/stats/flowentry/add**. Ο πίνακας που ακολουθεί επεξηγεί μερικές από τις ιδιότητες που είναι δυνατόν να συμπεριληφθούν στο σώμα ενός αιτήματος HTTP για την εγκατάσταση ενός flow rule:

Attribute	Description	Value type
dpid	datapath ID	ακέραιος
idle_timeout	το idle timeout του κανόνα	ακέραιος
hard_timeout	το hard timeout του κανόνα	ακέραιος
priority	προτεραιότητα του κανόνα	ακέραιος
match	match fields του κανόνα	dictionary
actions	οι εντολές του κανόνα	λίστα από dictionaries

Πίνακας 4.1 – ιδιότητες αιτήματος REST που μπορεί να περιλαμβάνει ένα flow rule

Οι συμβολισμοί που χρησιμοποιεί το REST API του Ryu controller για τα **match fields** παρουσιάζονται στον παρακάτω πίνακα:

Match field	Description
in_port	ingress port
dl_src	ethernet source address
dl_dst	ethernet destination address
dl_vlan	VLAN id
dl_vlan_pcp	VLAN priority
dl_type	ethernet type
nw_tos	IP ToS bits
nw_proto	IP protocol
nw_src	IP source address
nw_dst	IP destination address
tp_src	transport source port / ICMP type

tp_dst	transport destination port / ICMP code

Πίνακας 4.2 – συμβολισμός που χρησιμοποιείται από το REST API του Ryu controller για τα match fields ενός flow rule

4.4: Collector – Detector

4.4.1: Elasticsearch NoSQL database

Το **elasticsearch** [49, 50] αποτελεί μία κατανεμημένη, πολυσυνεργατική full-text μηχανή αναζήτησης που λειτουργεί χρησιμοποιώντας REST APIs και το πρότυπο JSON για τη μεταφορά των δεδομένων.

Στη συγκεκριμένη υλοποίηση θα χρησιμοποιηθεί ως μία **NoSQL** (Not only SQL) βάση δεδομένων [51], δηλαδή μία βάση δεδομένων που επιτυγχάνει μεγαλύτερη ευελιξία από τις κλασικές, χρησιμοποιώντας μεθόδους που δε βασίζονται αποκλειστικά στο παραδοσιακό σχεσιακό μοντέλο (relational databases) βάσεων δεδομένων. Κάθε εγγραφή της Elasticsearch βάσης προσδιορίζεται από ένα **index**, ένα **doc-type** και ένα **id**.

4.4.2: sflowtool

Είναι ο **sFlow collector** που επιλέχθηκε ως καταλληλότερος για τη συλλογή και ανάλυση των πακέτων που δειγματοληπτούνται στο OpenFlow switch, χρησιμοποιώντας το πρωτόκολλο sFlow. Το **sflowtool** [52] είναι γραμμένο στη γλώσσα προγραμματισμού C.

Τα δεδομένα που συλλέγει το sflowtool επιστρέφονται στο χρήστη **ανά γραμμές** και η πρώτη λέξη κάθε γραμμής επεξηγεί τη σημασία των τιμών που περιλαμβάνονται στη γραμμή αυτή. Στον παρακάτω πίνακα, δίνονται οι λέξεις που έχουν ιδιαίτερη σημασία για το αρχείο sflow.py και η σημασία τους. Η σειρά με την οποία παρουσιάζονται στις γραμμές του πίνακα είναι και η σειρά με την οποία επιστρέφονται από το sflowtool.

Πρώτη λέξη γραμμής	**Σημασία**
startSample	δηλώνει την αρχή του δείγματος
headerBytes	η επικεφαλίδα του δείγματος ανά byte σε δεκαεξαδική μορφή
srcIP	IP προέλευσης του πακέτου
dstIP	IP προορισμού του πακέτου
IPProtocol	δηλώνει το πρωτόκολλο στρώματος μεταφοράς
UDPSrcPort	UDP θύρα προέλευσης του πακέτου
UDPDstPort	UDP θύρα προορισμού του πακέτου

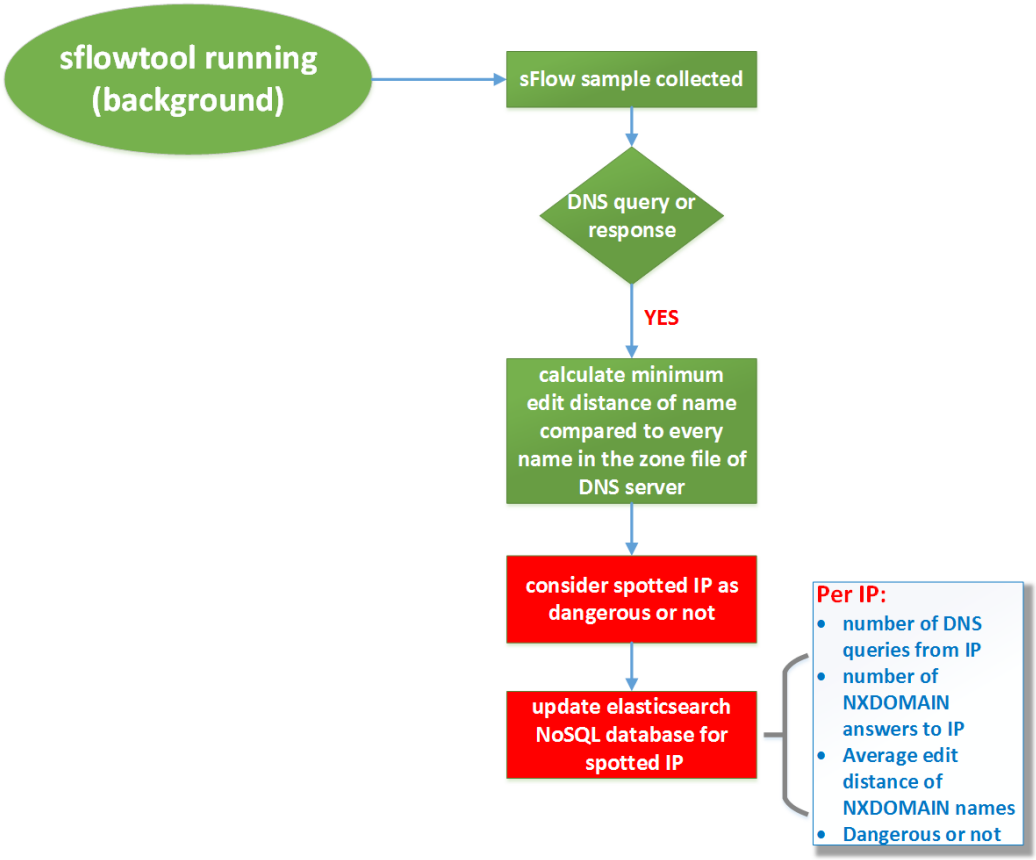
endSample	δηλώνει το τέλος του δείγματος

Πίνακας 4.3 – εναρκτήριες λέξεις των γραμμών στην έξοδο του sflowtool και η σημασία τους

4.4.3: Μηχανισμός δειγματοληψίας – ανάλυση του αρχείου sflow.py

Στο παρακάτω σχήμα, παρουσιάζεται σε διάγραμμα ροής η λειτουργία που επιτελεί ο κώδικας του αρχείου sflow.py (Python).

Ο sFlow collector sflowtool τρέχει στο παρασκήνιο και επεξεργάζεται τα δείγματα που συλλέγει. Αρχικά, ελέγχεται εάν το δείγμα πρόκειται για ερώτημα DNS ή απόκριση DNS. Στην περίπτωση ενός τέτοιου μηνύματος, προσδιορίζεται η ελάχιστη edit distance του ονόματος που βρίσκεται στο question section του μηνύματος ως προς όλα τα ονόματα που υπάρχουν στο αρχείο ζώνης του DNS server του υποδικτύου. Στη συνέχεια, γίνεται έλεγχος για να διαπιστωθεί εάν η IP που σχετίζεται με το μήνυμα DNS πρέπει να θεωρηθεί επικίνδυνη για τη λειτουργία του δικτύου ή όχι. Τέλος, ενημερώνονται οι τιμές που διατηρεί η βάση δεδομένων elasticsearch για την IP αυτή.



Σχήμα 4.2 – διάγραμμα ροής του αρχείου sflow.py

Στη συνέχεια, ακολουθεί αναλυτική παρουσίαση του κώδικα του αρχείου sflow.py:

- **συνάρτηση fill_list:**
  Η συνάρτηση αυτή δε δέχεται κάποιο όρισμα. Δημιουργεί μία λίστα με τα ονόματα που περιλαμβάνονται στο αρχείο ζώνης του DNS server του υποδικτύου. Η λίστα αποθηκεύεται στην καθολικής εμβελείας (global) μεταβλητή listing. Η συνάρτηση δεν επιστρέφει πίσω κάποιο αποτέλεσμα.

- **συνάρτηση edit_distance:**
  Η συνάρτηση αυτή δέχεται ως ορίσματα δύο αλφαριθμητικά και υπολογίζει πόσο διαφέρουν μεταξύ τους, υλοποιώντας τον αλγόριθμο edit distance. Το αποτέλεσμα επιστρέφεται από τη συνάρτηση. Η συνάρτηση χρησιμοποιεί την τεχνική του δυναμικού προγραμματισμού και υλοποιείται, όπως ακριβώς περιγράφεται στην αντίστοιχη ενότητα του δευτέρου κεφαλαίου.

- **συνάρτηση call_and_peek_output:**
  Η συνάρτηση αυτή δέχεται μια εντολή ως όρισμα (μεταβλητή command), την οποία θα εκτελούσε ένας χρήστης σε περιβάλλον φλοιού (shell) και μια λογική τιμή που καθορίζει το αν θα δημιουργηθεί καινούριο shell για αυτή (μεταβλητή shell με προκαθορισμένη τιμή False). Αρχικά, η συνάρτηση δημιουργεί ένα ζεύγος master-slave, χρησιμοποιώντας τη βιβλιοθήκη pty, η οποία υλοποιεί την ιδέα του pseudo-terminal: εκκίνηση μιας νέας διεργασίας και δυνατότητα εγγραφής και ανάγνωσης από το terminal που την ελέγχει μέσω προγραμματισμού. Στη συνέχεια, χρησιμοποιώντας την εντολή Popen() της βιβλιοθήκης subprocess, ο master εκτελεί την εντολή και τα αποτελέσματα που προορίζονται για το stdout κατευθύνονται στον slave. Τέλος, ο master διαβάζει τα αποτελέσματα εκτέλεσης της εντολής, όπως αυτά καταφθάνουν στο slave και επιστρέφει τα αποτελέσματα ανά γραμμές (αναζητείται ο χαρακτήρας ‘\n’).

- **συνάρτηση find_minimum_edit_distance:**
  Η συνάρτηση αυτή δέχεται ως όρισμα το FQDN (μεταβλητή name) που βρίσκεται στο question section της ερώτησης DNS ή απάντησης DNS που δειγματοληπτήθηκε από το sFlow. Κάθε όνομα που περιλαμβάνεται στο αρχείο ζώνης του DNS server (περιλαμβάνονται στη λίστα listing), συγκρίνεται με το περιεχόμενο της μεταβλητής name, σύμφωνα με τον αλγόριθμο edit distance, ώστε να διαπιστωθεί πόσο διαφέρουν. Η μικρότερη τιμή που θα προσδιοριστεί αποθηκεύεται στη μεταβλητή minimum και επιστρέφεται από τη συνάρτηση.

- **συνάρτηση find_query_name:**
  Η συνάρτηση αυτή είναι ακριβώς ίδια με εκείνη που περιλαμβάνεται και στο αρχείο bloom.py. Η περιγραφή της παρουσιάζεται σε επόμενη ενότητα.

- **συνάρτηση set_IP:**
  Η συνάρτηση αυτή δέχεται ως ορίσματα τις IP προέλευσης και IP προορισμού (srcIP

και dstIP αντίστοιχα) του μηνύματος DNS που δειγματοληπτήθηκε με το sFlow. Σκοπός της είναι να ξεχωρίσει την IP από την οποία προήλθε το αρχικό ερώτημα DNS από την IP "10.0.0.1", η οποία είναι η IP του DNS server του υποδικτύου. Η IP αυτή επιστρέφεται από τη συνάρτηση.

- **συνάρτηση update_database:**
  Η συνάρτηση αυτή δέχεται ως ορίσματα την ελάχιστη edit distance που προσδιορίστηκε (μεταβλητή minimum) προηγουμένως από τη συνάρτηση find_minimum_edit_distance και την IP για την οποία θα γίνει ενημέρωση της βάσης δεδομένων (μεταβλητή IP). Δεν επιστρέφει κάποιο αποτέλεσμα πίσω στο κύριο πρόγραμμα.

  Οι πληροφορίες που συγκεντρώνει ο μηχανισμός αποθηκεύονται στην elasticsearch βάση δεδομένων στην εγγραφή που χρησιμοποιεί ως index τη λέξη "dns", ως doc-type τη λέξη "ip3" και ως id την εκάστοτε διεύθυνση IP που σχετίζεται με το δείγμα που λήφθηκε από το μηχανισμό sFlow. Για κάθε IP, διατηρείται στη βάση δεδομένων ο συνολικός αριθμός ερωτημάτων DNS και απαντήσεων που δειγματοληπτήθηκαν (attribute queries), ο συνολικός αριθμός NXDOMAIN μηνυμάτων (attribute nxdomain – αν η τιμή της μεταβλητής minimum είναι διαφορετική του 0, τότε το ερώτημα ή η απάντηση θεωρούνται NXDOMAIN μηνύματα), η μέση τιμή της μεταβλητής minimum για την IP αυτή (attribute malicious_average_length – το attribute malicious_total_length χρησιμοποιείται μόνο για τον υπολογισμό της μέσης τιμής) και ο χαρακτηρισμός για το αν η IP αυτή θεωρείται επικίνδυνη για τη λειτουργία του δικτύου (attribute dangerous – τιμή "no" για όχι επικίνδυνη και "yes" για επικίνδυνη).

  Αρχικά, λαμβάνονται τα περιεχόμενα της βάσης δεδομένων που αφορούν τη διεύθυνση IP του δείγματος με τη συνάρτηση get της βιβλιοθήκης elasticsearch. Το μπλοκ try-except χρησιμοποιείται για την ειδική περίπτωση που η IP αυτή εμφανίζεται για πρώτη φορά. Σε αυτήν την περίπτωση, προκαλείται ένα exception και αρχικοποιούνται τα περιεχόμενα της βάσης δεδομένων για την IP αυτή. Σε κάθε περίπτωση, οι πληροφορίες για την IP επιστρέφονται στο αντικείμενο json_object.

  Στη συνέχεια, λαμβάνονται από το αντικείμενο json_object οι τιμές των attibutes της βάσης δεδομένων για την IP που έχει εντοπιστεί και ενημερώνονται κατάλληλα. Σχετικά με το attribute dangerous, μία IP θεωρείται επικίνδυνη όταν ο λόγος των NXDOMAIN μηνυμάτων προς το συνολικό αριθμό των μηνυμάτων DNS ξεπερνάει το 15% ή όταν η μέση τιμή της μεταβλητής minimum ξεπερνάει την τιμή 2, για λόγους που επεξηγήθηκαν στο τρίτο κεφάλαιο της εργασίας.

  Τέλος, ενημερώνονται οι πληροφορίες που διατηρούνται στη βάση δεδομένων για την IP που προσδιορίστηκε, χρησιμοποιώντας τη συνάρτηση index της βιβλιοθήκης elasticsearch.

- **συνάρτηση main_function:**
  Η συνάρτηση αυτή δε δέχεται κάποιο όρισμα. Αρχικά, καλεί τη συνάρτηση fill_list και δημιουργεί το αντικείμενο es, μέσω του οποίου θα γίνεται η διαχείριση της βάσης δεδομένων elasticsearch.

  Στη συνέχεια, καλείται η συνάρτηση call_and_peek_output με όρισμα την εντολή "sflowtool", δηλαδή για την εκτέλεση του sFlow collector sflowtool στο παρασκήνιο (background). Έπειτα, κάθε γραμμή της εξόδου της εντολής sflowtool συγκρίνεται με μια σειρά από συνθήκες για να συγκεντρωθούν από το δείγμα οι πληροφορίες του που έχουν ενδιαφέρον:

➔ headerBytes: δημιουργείται μια λίστα (μεταβλητή headerBytes) που περιέχει τα bytes της επικεφαλίδας του μηνύματος και συγκρατείται στη μεταβλητή etherType το πρωτόκολλο στρώματος δικτύου.
➔ srcIP: IP προέλευσης
➔ dstIP: IP προορισμού
➔ IPProtocol: πρωτόκολλο στρώματος μεταφοράς
➔ UDPSrcPort: UDP θύρα προέλευσης
➔ UDPDstPort: UDP θύρα προορισμού
➔ endSample: όταν εντοπιστεί η γραμμή ολοκλήρωσης του δείγματος, γίνεται επιτρεπτή η ανάλυση των πληροφοριών που συλλέχθηκαν.

  Όταν ολοκληρωθεί η συλλογή πληροφοριών από ένα δείγμα sFlow, το δείγμα εξετάζεται για να διαπιστωθεί αν πρόκειται για ένα ερώτημα DNS ή μια απόκριση DNS (πρωτόκολλο στρώματος δικτύου IP, δηλαδή τιμή 0x0800, πρωτόκολλο στρώματος μεταφοράς UDP, δηλαδή τιμή 17, θύρα προέλευσης ή προορισμού η 53). Αν είναι μήνυμα DNS, τότε προσδιορίζεται το όνομα που περιλαμβάνεται στο question section του μηνύματος (συνάρτηση find_query_name), η IP που σχετίζεται με αυτό (συνάρτηση set_IP), η ελάχιστη τιμή για την edit distance του ονόματος (συνάρτηση find_minimum_edit_distance) και, τέλος, ενημερώνεται η βάση δεδομένων (συνάρτηση update_database).

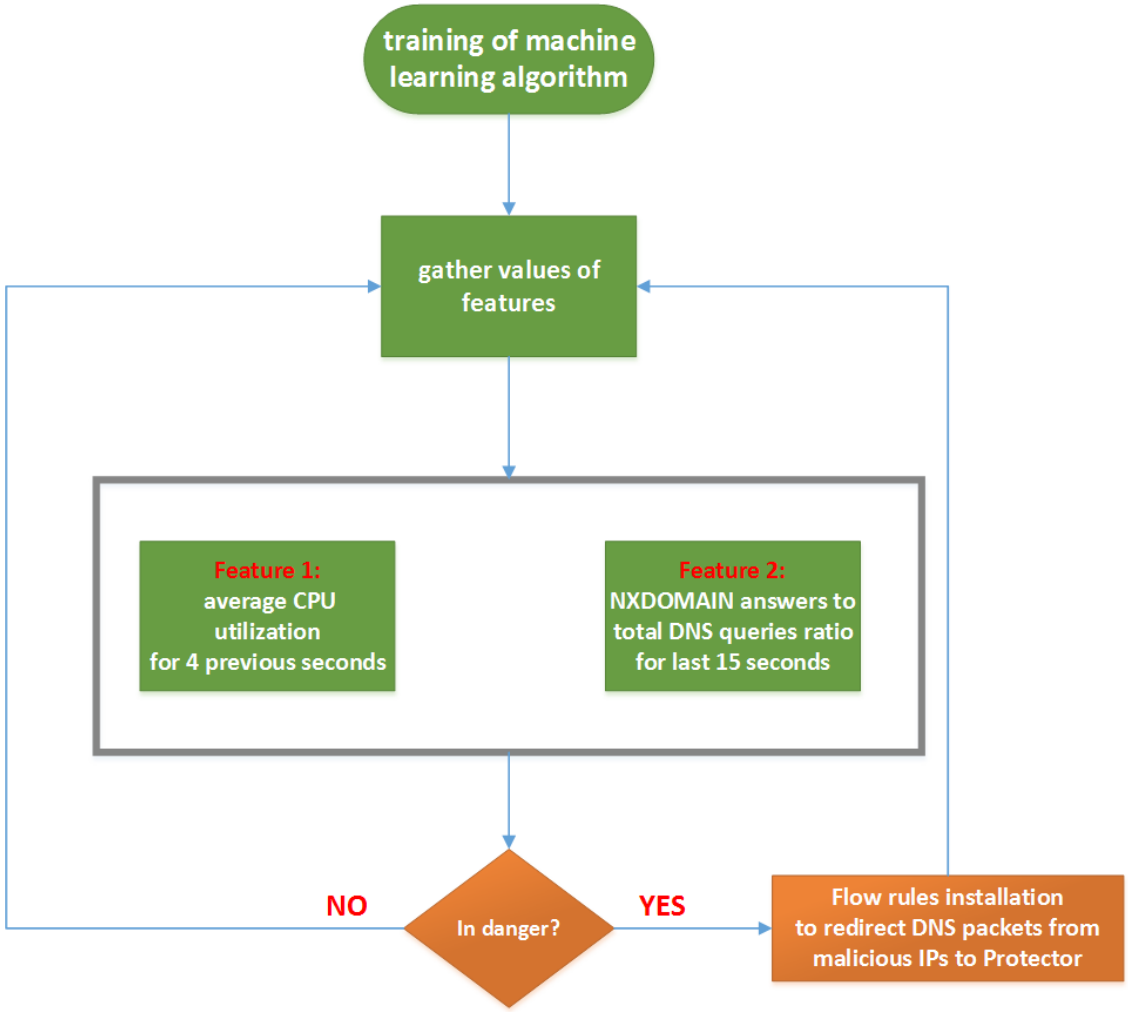
4.4.4: μηχανισμός εντοπισμού κινδύνου – ανάλυση του αρχείου detect.py

Για το Collectd, στον Collector – Detector ενεργοποιήθηκαν τα plug-ins rrdtool, cpu, dns και network με την επιλογή server.

Στο παρακάτω σχήμα παρουσιάζεται σε διάγραμμα ροής η λειτουργία που επιτελεί ο κώδικας του αρχείου detect.py (Python).

Αρχικά, γίνεται η εκπαίδευση του αλγορίθμου machine learning. Έπειτα, συγκεντρώνονται

συνεχώς οι τιμές των εισόδων του αλγορίθμου (features) και ελέγχεται η έξοδός του για αυτές. Αν ο εξυπηρετητής βρίσκεται σε κίνδυνο, τότε πραγματοποιείται εγκατάσταση κανόνων στο OpenFlow switch του δικτύου μέσω του REST API του Ryu controller για την ανακατεύθυνση των πακέτων DNS που προέρχονται από κακόβουλες IPs στον Protector.



Σχήμα 4.3 – διάγραμμα ροής του αρχείου detect.py

Ακολουθεί αναλυτική παρουσίαση του κώδικα του αρχείου detect.py:

- **συνάρτηση sum:**
  Η συνάρτηση δέχεται ως όρισμα μία λίστα (μεταβλητή item) και επιστρέφει πίσω το άθροισμα των στοιχείων της. Η built-in συνάρτηση float χρησιμοποιείται για τη μετατροπή των string στοιχείων σε float αριθμούς.

- **συνάρτηση get_values:**
  Η συνάρτηση δέχεται ως όρισμα τα αποτελέσματα της εντολής rrdtool fetch πάνω σε ένα rrdfile, η οποία επιστρέφει τα περιεχόμενα ενός rrdfile ανάμεσα σε δύο χρονικές

στιγμές οργανωμένα ανά γραμμές με τη μορφή <χρονική στιγμή καταγραφής> **:** <τιμή>. Η συνάρτηση απομονώνει τις τιμές, τις συγκεντρώνει στη λίστα values και επιστρέφει τη λίστα αυτή ως αποτέλεσμα.

- **συνάρτηση save_output:**
  Η συνάρτηση δέχεται ως όρισμα μία εντολή (μεταβλητή command), την οποία θα εκτελούσε ένας χρήστης σε περιβάλλον φλοιού (shell). Σκοπός της είναι να εκτελέσει την εντολή αυτή και να συγκρατήσει την έξοδό της για να μπορέσει ο χρήστης να τη χρησιμοποιήσει στο πρόγραμμά του. Γι' αυτό το λόγο, εκτελείται η εντολή Popen της βιβλιοθήκης subprocess με όρισμα την εντολή αυτή, επιλογή για standard output το ίδιο το πρόγραμμα (παράμετρος subprocess.PIPE) και επιλογή του shell (shell=True) ως του προγράμματος που θα εκτελέσει την εντολή. Το αποτέλεσμα της εντολής διαβάζεται από το stdout με τη συνάρτηση communicate και αποθηκεύεται στη μεταβλητή out. Τέλος, περικόπτονται οι χαρακτήρες αλλαγής γραμμής (μέθοδος rstrip) και το αποτέλεσμα επιστρέφεται στη συνάρτηση.

- **συνάρτηση do_it_for_file:**
  Η συνάρτηση αυτή δέχεται τρία ορίσματα: το όνομα ενός rrdfile (μεταβλητή file_of_interest), μια αρχική χρονική στιγμή (μεταβλητή starting) και μια τελική χρονική στιγμή (μεταβλητή ending). Η συνάρτηση εκτελεί την εντολή rrdtool fetch πάνω στο αρχείο file_of_interest για το χρονικό διάστημα ανάμεσα στην αρχική και την τελική χρονική στιγμή. Οι παράμετροι που χρησιμοποιούνται στην εντολή είναι:

  - *AVERAGE:* εξετάζονται τα τμήματα του rrdfile που αναφέρονται σε μέσες τιμές.
  - *-r:* καθορίζει το resolution, δηλαδή το interval ανάμεσα στις τιμές που επιλέγονται από το αρχείο. Επιλέγεται 1 για να συμπίπτει με την παράμετρο interval του Collectd.
  - *-s:* καθορίζει από ποια χρονική στιγμή θα επιστραφούν τιμές του αρχείου.
  - *-e:* καθορίζει μέχρι ποια χρονική στιγμή θα επιστραφούν τιμές του αρχείου.
  - *-a:* αυτόματη ευθυγράμμιση της αρχικής χρονικής στιγμής με το resolution.

  Στη συνέχεια, εφαρμόζεται στο αποτέλεσμα η συνάρτηση get_values για να επιστραφεί μια λίστα με τις αποθηκευμένες τιμές του αρχείου για το χρονικό διάστημα ενδιαφέροντος, αθροίζονται με τη συνάρτηση sum και το άθροισμα επιστρέφεται από τη συνάρτηση.

- **συνάρτηση get_training_data:**
  Στη συνάρτηση αυτή, η οποία δε δέχεται κάποιο όρισμα, παρέχονται τα δεδομένα που θα χρησιμοποιηθούν για την εκπαίδευση του αλγορίθμου SVM. Οι είσοδοι του αλγορίθμου αποθηκεύονται στη μεταβλητή X, η οποία είναι ένας numpy πίνακας (η βιβλιοθήκη NumPy επιτρέπει το χειρισμό πολύ μεγάλων μονοδιάστατων ή πολυδιάστατων πινάκων) και οι επιθυμητές έξοδοι για αυτά ορίζονται στη λίστα Y. Το στοιχείο στη θέση i της μεταβλητής X αντιστοιχεί στην τιμή του στοιχείου i της

μεταβλητής Y. Η συνάρτηση επιστρέφει πίσω τις μεταβλητές X και Y.

- **συνάρτηση svm_train:**
  Η συνάρτηση αυτή δε δέχεται κάποιο όρισμα και πραγματοποιεί την εκπαίδευση του αλγορίθμου SVM. Αρχικά, ανακτάται το training set με τη συνάρτηση get_training_data και καθορίζονται κύριες παράμετροι του αλγορίθμου με τη μέθοδο SVC της κλάσης svm. Εδώ, η μοναδική παράμετρος kernel καθορίζει ότι ο πυρήνας του αλγορίθμου θα είναι γραμμική συνάρτηση (kernel = 'linear'). Τέλος, με τη μέθοδο fit πραγματοποιείται η εκπαίδευση και το αποτέλεσμα φυλάσσεται στο καθολικής εμβελείας (global) αντικείμενο clf.

- **συνάρτηση svm_predict:**
  Η συνάρτηση δέχεται ως όρισμα μία είσοδο που προορίζεται για τον αλγόριθμο SVM (μεταβλητή input) και με τη μέθοδο predict της κλάσης sklearn.svm υπολογίζει την έξοδο του αλγορίθμου σε αυτή. Η έξοδος επιστρέφεται από τη συνάρτηση.

- **συνάρτηση return_prediction:**
  Η συνάρτηση αυτή δε δέχεται κάποιο όρισμα και σκοπός της είναι να εξετάσει εάν ο DNS server βρίσκεται σε κίνδυνο ή όχι. Αρχικά, υπολογίζονται ο αριθμός των ερωτημάτων DNS στο server (περιέχονται στο rrdfile dns_qtype_A.rrd), ο αριθμός των DNS NXDOMAIN απαντήσεων (περιέχονται στο rrdfile dns_rcode_NXDOMAIN.rrd) και το ποσοστό της CPU του server που παραμένει ανενεργό (περιέχεται στο rrdfile percent_idle.rrd, διαίρεση με 4 γιατί θέλουμε μέσο ποσοστό για τη CPU και όχι άθροισμα, όπως στα υπόλοιπα). Οι χρονικές στιγμές για τις οποίες λαμβάνονται οι τιμές έχουν μία μικρή απόσταση από το παρόν για να εξασφαλιστεί ότι θα έχει πραγματοποιηθεί η αποθήκευσή τους στο rrdfile. Στη συνέχεια, προσδιορίζονται οι είσοδοι του αλγορίθμου SVM, δίνονται σε αυτόν και η έξοδός του επιστρέφεται από τη συνάρτηση svm_predict.

- **συνάρτηση redirect_ips:**
  Η συνάρτηση δέχεται ως όρισμα τη λίστα με τις διευθύνσεις IP (μεταβλητή malicious_ips) που θεωρούνται επικίνδυνες και, μέσω του REST API του controller, εγκαθιστά κανόνες ώστε τα πακέτα DNS από τις IP αυτές να κατευθύνονται πρώτα στον Protector.

- **συνάρτηση main_function:**
  Η συνάρτηση δε δέχεται κάποιο όρισμα. Αρχικά, δημιουργεί το αντικείμενο es για τη διαχείριση της βάσης δεδομένων elasticsearch και εκπαιδεύει τον αλγόριθμο μηχανικής μάθησης, καλώντας τη συνάρτηση svm_train. Στη συνέχεια, το πρόγραμμα μπαίνει σε έναν ατέρμονο βρόχο, όπου συνεχώς εξετάζεται εάν ο εξυπηρετητής DNS βρίσκεται σε κίνδυνο με τη συνάρτηση return_prediction (η ένδειξη 1 δηλώνει κίνδυνο). Σε περίπτωση που κινδυνεύει ο server αναζητούνται με τη μέθοδο search της βιβλιοθήκης elasticsearch οι επικίνδυνες διευθύνσεις IP

(εγγραφές με dangerous:"yes", το κριτήριο αναζήτησης καθορίζεται με την παράμετρο q της συνάρτησης search) και εγκαθιστώνται κανόνες για την ανακατεύθυνση των μηνυμάτων DNS από εκείνες προς τον Protector με τη συνάρτηση redirect_ips. Ο μηχανισμός εξακολουθεί να επαναλαμβάνει τη διαδικασία ελέγχου και εγκατάστασης κανόνων ανά χρονικά διαστήματα που ορίζονται από τη συνάρτηση time.sleep μέχρι το δίκτυο να επιστρέψει στην ομαλή λειτουργία του (ένδειξη 0).

4.5: Protector

Ο Protector αποτελείται από το load balancer και από τα bloom filters.

4.5.1: Load Balancer – Ανάλυση του αρχείου iptables.txt

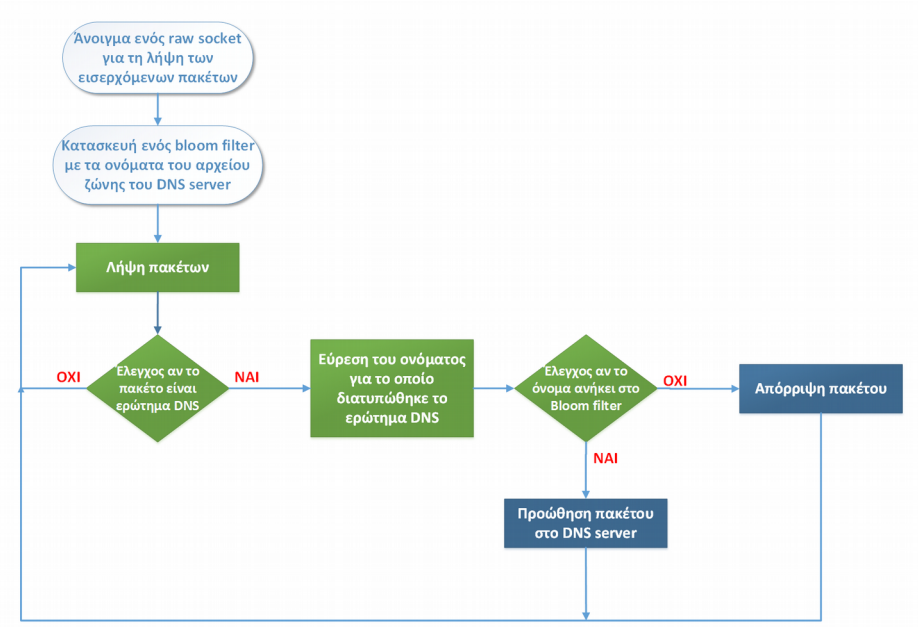
Για την κατασκευή του load balancer χρησιμοποιήθηκε το λογισμικό **iptables** [53], δηλαδή το user-space πρόγραμμα του Linux που επιτρέπει την παραμετροποίηση του firewall που λειτουργεί στον πυρήνα του Linux.

Οι εντολές που είναι απαραίτητες για τη δημιουργία ενός load balancer για **πέντε** bloom filters βρίσκονται στο αρχείο iptables.txt που βρίσκεται στο παράρτημα. Η πρώτη εντολή είναι απαραίτητη ώστε να ενεργοποιηθεί η λειτουργία δρομολόγησης στο kernel του load balancer. Οι επόμενες εντολές τροποποιούν την αλυσίδα (chain) PREROUTING του iptables, δηλαδή την αλυσίδα εκείνη που περιλαμβάνει τους κανόνες που εξετάζονται πριν ληφθεί η απόφαση για τη δρομολόγηση του πακέτου. Εδώ, χρησιμοποιείται το mode **nth** που καθορίζει ότι κάθε ν-οστό (ορίζεται με την παράμετρο --every) ερώτημα DNS (ορίζεται από τη θύρα προορισμού 53, --dport) που καταφθάνει στον load balancer θα δρομολογείται προς τον υπολογιστή, του οποίου η διεύθυνση IP ορίζεται με την παράμετρο --to-destination.

4.5.2: Bloom Filters – Ανάλυση του αρχείου bloom.py

Στο παρακάτω σχήμα παρουσιάζεται σε διάγραμμα ροής η λειτουργία που επιτελεί ο κώδικας που περιλαμβάνεται στο αρχείο bloom.py (Python).

Αρχικά, ανοίγεται ένα raw socket για την παραλαβή των πακέτων που καταφθάνουν και κατασκευάζεται ένα bloom filter με τα ονόματα που περιλαμβάνονται στο αρχείο ζώνης του DNS server. Όταν λαμβάνεται ένα πακέτο, γίνεται έλεγχος αν είναι ερώτημα DNS. Αν είναι, τότε προσδιορίζεται το όνομα που περιλαμβάνεται στο question section του μηνύματος. Αν το όνομα αυτό βρίσκεται αποθηκευμένο στο bloom filter, τότε προωθείται στον DNS server, αλλιώς απορρίπτεται.



Σχήμα 4.4 – διάγραμμα ροής του αρχείου bloom.py

Στη συνέχεια, περιγράφεται αναλυτικά ο κώδικας του αρχείου bloom.py.

- **συνάρτηση main_function:**
  Είναι η κύρια συνάρτηση του προγράμματος. Αρχικά, δημιουργείται μια raw socket (επιλογή SOCK_RAW) για τη λήψη πακέτων πρωτοκόλλου IP (επιλογή 0x0800). Μια raw socket επιτρέπει τη διαχείριση όλων των στρωμάτων πρωτοκόλλων του πακέτου που λαμβάνεται από αυτή. Στη συνέχεια, πραγματοποιείται η αποθήκευση των ονομάτων του αρχείου ζώνης του DNS server στο bloom filter με τη συνάρτηση create_bloom_filter. Τέλος, ένας μη τερματίσιμος βρόχος ακούει συνεχώς για πακέτα που καταφθάνουν στο socket. Ένα πακέτο παραλαμβάνεται με τη συνάρτηση recv της βιβλιοθήκης socket και η επεξεργασία του γίνεται με τη συνάρτηση examine_and_parse_packet.

- **συνάρτηση examine_ethernet_header:**
  Η συνάρτηση δέχεται ως όρισμα το πακέτο που λήφθηκε από το socket σε μορφή bytes object της Python. Τα πρώτα 14 bytes, δηλαδή η επικεφαλίδα ethernet του πακέτου, αποθηκεύονται στη μεταβλητή layer_2_header (επισημαίνεται ότι στις λίστες της Python το ανώτερο όριο είναι ανοιχτό). Στη συνέχεια, χρησιμοποιείται η συνάρτηση unpack της βιβλιοθήκης struct ώστε να επιστραφεί ένα tuple, στις θέσεις του οποίου θα βρίσκονται αποθηκευμένα τα περιεχόμενα της μεταβλητής layer_2_header, διαμορφωμένα σύμφωνα με το πρώτο όρισμα της συνάρτησης unpack. Στη συγκεκριμένη περίπτωση, το όρισμα “!6s6s2s” ορίζει ότι στη θέση 0 του tuple θα αποθηκευτούν τα έξι πρώτα bytes της επικεφαλίδας ethernet, στη θέση 1 τα επόμενα έξι και στη θέση 2 τα τελευταία δύο bytes. Κατά σειρά, οι τιμές αυτές αντιστοιχούν στη διεύθυνση MAC προορισμού, στη διεύθυνση MAC προέλευσης και τον τύπο πρωτοκόλλου του αμέσως ανώτερου πρωτοκόλλου στο πακέτο. Τέλος,

οι πληροφορίες αυτές μετατρέπονται σε δεκαεξαδικές τιμές με τη συνάρτηση hexlify της βιβλιοθήκης binascii και επιστρέφονται πίσω από τη συνάρτηση.

Επισημαίνεται ότι η συνάρτηση αυτή δε χρησιμοποιείται στη συγκεκριμένη υλοποίηση, αλλά καταγράφεται για λόγους πληρότητας για περιπτώσεις που απαιτείται η εξέταση των πληροφοριών της ethernet επικεφαλίδας.

- **συνάρτηση examine_ip_header:**
  Η συνάρτηση δέχεται ως όρισμα το πακέτο που λήφθηκε από το socket σε μορφή bytes object της Python. Απομονώνει το τμήμα του πακέτου που αποτελεί την επικεφαλίδα IP του, δηλαδή τα bytes 14 έως και 33, και επιστρέφει το πρωτόκολλο στρώματος μεταφοράς σε δεκαεξαδική μορφή (μεταβλητή tcp_or_udp) και τις διευθύνσεις IP προέλευσης και προορισμού (μεταβλητές source_ip και destination_ip αντίστοιχα), αφού τις μετατρέψει από δεκαεξαδική μορφή σε μορφή διευθύνσεων IP (μορφή x.x.x.x), χρησιμοποιώντας τη συνάρτηση inet_ntoa της βιβλιοθήκης socket.

- **συνάρτηση examine_udp_header:**
  Η συνάρτηση δέχεται ως όρισμα το πακέτο που λήφθηκε από το socket σε μορφή bytes object της Python, απομονώνει το τμήμα του πακέτου που αντιστοιχεί στην επικεφαλίδα UDP, δηλαδή τα bytes 34 έως και 41 και επιστρέφει τη θύρα προέλευσης (μεταβλητή source_port) και τη θύρα προορισμού (μεταβλητή dst_port) σε δεκαεξαδική μορφή.

- **συνάρτηση examine_dns_header:**
  Η συνάρτηση δέχεται ως όρισμα το πακέτο που λήφθηκε από το socket σε μορφή bytes object της Python. Αρχικά, προσδιορίζεται το συνολικό μέγεθος του πακέτου με την built-in συνάρτηση len και αποθηκεύεται στη μεταβλητή total_packet_length. Έπειτα, συγκρατούνται τα bytes που αντιστοιχούν στην επικεφαλίδα DNS του μηνύματος, δηλαδή τα bytes από το 42ο έως το τέλος του πακέτου, καθώς η επικεφαλίδα DNS περιλαμβάνεται στα δεδομένα του πακέτου. Στη συνέχεια, η επικεφαλίδα DNS μετατρέπεται από μορφή bytes object σε δεκαεξαδική μορφή και το αποτέλεσμα αποθηκεύεται στη μεταβλητή dns_header. Έπειτα, χρησιμοποιείται ένα list comprehension, ώστε να δημιουργηθεί μία λίστα, κάθε στοιχείο της οποίας είναι ένα byte της επικεφαλίδας DNS (το βήμα στο list comprehension είναι 2, αφού 2 δεκαεξαδικοί αριθμοί αντιστοιχούν σε ένα byte). Η λίστα αυτή, αποθηκευμένη στη μεταβλητή dns_header παρέχεται ως όρισμα στη συνάρτηση find_query_name, με την οποία προσδιορίζεται το όνομα της ερώτησης DNS (μεταβλητή query_name) και, τελικά, επιστρέφεται από τη συνάρτηση examine_dns_header.

- **συνάρτηση find_query_name:**
  Η συνάρτηση αυτή δέχεται ως όρισμα την επικεφαλίδα DNS του πακέτου σε μορφή λίστας ανά byte και ο ρόλος της είναι να προσδιορίσει το όνομα που περιλαμβάνεται

στο question section της επικεφαλίδας DNS.

Το τμήμα αυτό της επικεφαλίδας DNS έχει μεταβλητό μήκος. Στο 12ο byte της επικεφαλίδας αναγράφεται το μήκος του πρώτου label του ονόματος. Από το 13ο byte και για όσους χαρακτήρες ορίζει το δωδέκατο byte, δίνεται το πρώτο label. Αμέσως μετά, αναγράφεται το μήκος του δεύτερου label και ακολουθείται η ίδια λογική. Όλα τα labels έχουν προσδιοριστεί πλήρως, όταν στο byte που περιλαμβάνει το μήκος του αμέσως επόμενου label, υπάρχει ο αριθμός μηδέν.

Η συνάρτηση, λοιπόν, υλοποιεί την παραπάνω διαδικασία αναζήτησης. Αρχικά, ορίζεται ένα dictionary labels, στο οποίο θα αποθηκευτούν τα labels, αφού προσδιοριστούν με τη θέση τους στο FQDN ως key του dictionary, ξεκινώντας την αρίθμηση από το πρώτο label. Λαμβάνεται από τη δωδέκατη θέση της επικεφαλίδας DNS το μήκος του πρώτου label σε δεκαεξαδική μορφή και με την built-in συνάρτηση int() μετατρέπεται σε δεκαδική μορφή. Η τιμή αυτή αποθηκεύεται στη μεταβλητή check_length. Στη συνέχεια, λαμβάνεται το περιεχόμενο των επόμενων check_length θέσεων της επικεφαλίδας και αποθηκεύεται στη μεταβλητή string. Επειδή το label βρίσκεται αποθηκευμένο στη μεταβλητή string ανά byte σε δεκαεξαδική μορφή, εφαρμόζεται η συνάρτηση decode με το όρισμα “hex” ώστε να ληφθεί μια αναπαράσταση σε μορφή ASCII. Το label αποθηκεύεται στο dictionary labels. Έπειτα, διαβάζεται το μήκος του επόμενου label. Αν είναι μηδενικό, τότε γίνεται break και η ροή του προγράμματος συνεχίζεται έξω από το βρόχο, αλλιώς η διαδικασία επαναλαμβάνεται για να προσδιοριστεί το επόμενο label.

Τέλος, όλα τα labels ενώνεται με τελείες, δηλαδή προσδιορίζεται το FQDN, αποθηκεύεται στη μεταβλητή query_name και επιστρέφεται από τη συνάρτηση.

- **συνάρτηση create_bloom_filter:**
  Η συνάρτηση δε δέχεται κάποιο όρισμα και ο σκοπός της είναι να αποθηκεύσει τα ονόματα του αρχείου ζώνης του εξυπηρετητή DNS στο bloom filter της μονάδας. Η διαχείριση του φίλτρου γίνεται μέσω της μεταβλητής καθολικής εμβελείας bf. Η συνάρτηση διαβάζει από ένα αρχείο τα ονόματα που είναι υπαρκτά στη ζώνη example.com και τα προσθέτει στο φίλτρο, χρησιμοποιώντας τη μέθοδο add της κλάσης BloomFilter. Η συνάρτηση δεν επιστρέφει κάποιο αποτέλεσμα.

- **κλάση BloomFilter:** αποτελεί την κλάση που ορίζει τις ιδιότητες και τις μεθόδους που σχετίζονται με ένα bloom filter και τη λειτουργία του.

➔ **μέθοδος __init__:**
Είναι ο constructor της κλάσης BloomFilter και δέχεται ως ορίσματα το μέγεθος σε bits (μεταβλητή size) του φίτρου και τον αριθμό των hash functions που αυτό χρησιμοποιεί (μεταβλητή hash_count). Για τα ορίσματα αυτά παρέχονται επιθυμητές default τιμές 4KB και 10 αντίστοιχα. Στη συνέχεια, αρχικοποιούνται οι δύο αυτές

ιδιότητες του φίλτρου, αρχικοποιείται το ίδιο το φίλτρο ως ένα bitarray μεγέθους size και αποθηκεύεται στη μεταβλητή bit_array και, τέλος, όλα τα bits του φίλτρου αρχικοποιούνται με την τιμή 0.

➔ **μέθοδος add:**
Η μέθοδος αυτή δέχεται ως όρισμα μία τιμή και την προσθέτει στο bloom filter. Χρησιμοποιώντας τη συνάρτηση κατακερματισμού mmh3 (murmur hash function) με διαφορετικά seeds, ξεκινώντας από το 0 μέχρι τον αριθμό των hash functions που χρησιμοποιεί το φίλτρο μείον ένα, βρίσκει τις θέσεις του φίλτρου, οι οποίες αντιστοιχούν στο στοιχείο που προστίθεται σε αυτό. Ο τελεστής modulo (%) χρησιμοποιείται για να εξασφαλιστεί ότι η τιμή hash που θα προκύψει θα βρίσκεται εντός των ορίων του φίλτρου. Τέλος, σε όλες τις θέσεις του φίλτρου που προσδιορίστηκαν, τίθεται η τιμή 1.

➔ **μέθοδος query:**
Η μέθοδος δέχεται ως όρισμα μία τιμή και εξετάζει αν η τιμή αυτή ανήκει στο bloom filter. Ακολουθώντας την ίδια διαδικασία με τη μέθοδο add, προσδιορίζει τις θέσεις του φίλτρου στις οποίες θα πρέπει να έχει αποθηκευτεί η τιμή, εάν έχει προστεθεί στο φίλτρο. Αν έστω και μία από αυτές τις θέσεις περιέχει την τιμή μηδέν, τότε επιστρέφει τη λογική τιμή False για να δείξει ότι η τιμή δεν ανήκει στο φίλτρο. Αλλιώς, επιστρέφει τη λογική τιμή True.

- **συνάρτηση examine_and_parse_packet:**
Η συνάρτηση δέχεται ως όρισμα το πακέτο που λήφθηκε από το socket σε μορφή bytes object.

  Αρχικά, ανοίγει ένα socket για την αποστολή του πακέτου προς τον εξυπηρετητή DNS, εάν χρειαστεί. Στη συνέχεια, καλεί τις συναρτήσεις examine_ip_header και examine_udp_header για να εξακριβώσει εάν το πακέτο IP που παραλήφθηκε είναι ερώτημα DNS. Για να συμβεί αυτό θα πρέπει, πρώτα, να διαπιστωθεί αν το πρωτόκολλο στρώματος μεταφοράς είναι UDP. Γι' αυτό το λόγο, ελέγχεται αν η μεταβλητή tcp_or_udp περιέχει τη δεκαεξαδική τιμή 11 (δεκαδική 17) που υποδηλώνει το πρωτόκολλο UDP. Έπειτα, ελέγχεται η θύρα προορισμού (μεταβλητή dst_port) για να διαπιστωθεί εάν στα δεδομένα του πακέτου βρίσκεται ένα ερώτημα DNS. Η θύρα προορισμού που υποδηλώνει το πρωτόκολλο DNS είναι η 35 στο δεκαεξαδικό σύστημα (στο δεκαδικό 53).

  Αν, λοιπόν, το πακέτο είναι ένα ερώτημα DNS, καλείται η συνάρτηση examine_dns_header ώστε να βρεθεί το όνομα για το οποίο διατυπώθηκε η ερώτηση. Στη συνέχεια, γίνεται έλεγχος εάν το όνομα αυτό ανήκει στο bloom filter, εφαρμόζοντας τη μέθοδο query της κλάσης BloomFilter. Αν ναι, το πακέτο αποστέλλεται στον παραλήπτη, δηλαδή στον authoritative DNS server του υποδικτύου, αλλιώς απορρίπτεται.

Στην περίπτωση που χρησιμοποιείται load balancer, η συνάρτηση χρειάζεται να τροποποιηθεί, καθώς η διεύθυνση IP προορισμού του πακέτου δεν είναι πια εκείνη του DNS server, αλλά ο load balancer την έχει αντικαταστήσει με εκείνη του bloom filter που παραλαμβάνει το πακέτο. Πρέπει να επισημανθεί ότι, έχοντας ορίσει τον load balancer να λειτουργεί transparently, η IP προέλευσης και η θύρα προέλευσης του πακέτου δεν αλλάζουν.

Επομένως, η συνάρτηση χρειάζεται να τροποποιηθεί ώστε να τεθεί ως IP προορισμού η διεύθυνση IP του DNS server. Για να γίνει αυτό χρησιμοποιείται η βιβλιοθήκη κατασκευής και τροποποίησης πακέτων της Python, που ονομάζεται Scapy [54]. Αρχικά, το πακέτο μετατρέπεται από bytes object σε μορφή διαχειρίσιμη από το Scapy με τον constructor Ether() και το αποτέλεσμα αποθηκεύεται στη μεταβλητή scapy_packet. Στη συνέχεια, τροποποιείται η διεύθυνση προορισμού του πακέτου (scapy_packet[IP].dst) ώστε να γίνει εκείνη του DNS server. Έπειτα, διαγράφονται τα πεδία checksum (chksum) της επικεφαλίδας IP και UDP, ώστε να προσδιοριστούν σωστά από το Scapy, όταν θα γίνει η αποστολή του πακέτου. Τέλος, ανοίγεται μία layer 2 socket και το πακέτο αποστέλλεται μέσω αυτής.

5 Αξιολόγηση Υλοποίησης

Στο συγκεκριμένο κεφάλαιο, παρουσιάζονται μετρήσεις και αποτελέσματα που σχετίζονται με την **αξιολόγηση** της αποτελεσματικότητας του μηχανισμού άμυνας που υλοποιήθηκε στα πλαίσια της παρούσης διπλωματικής εργασίας.

Στην πρώτη ενότητα, πραγματοποιείται μία γενική μελέτη του τρόπου λειτουργίας των bloom filters, ενώ στις επόμενες ενότητες αξιολογείται ο μηχανισμός που υλοποιήθηκε.

5.1: Μελέτη απόδοσης των bloom filters

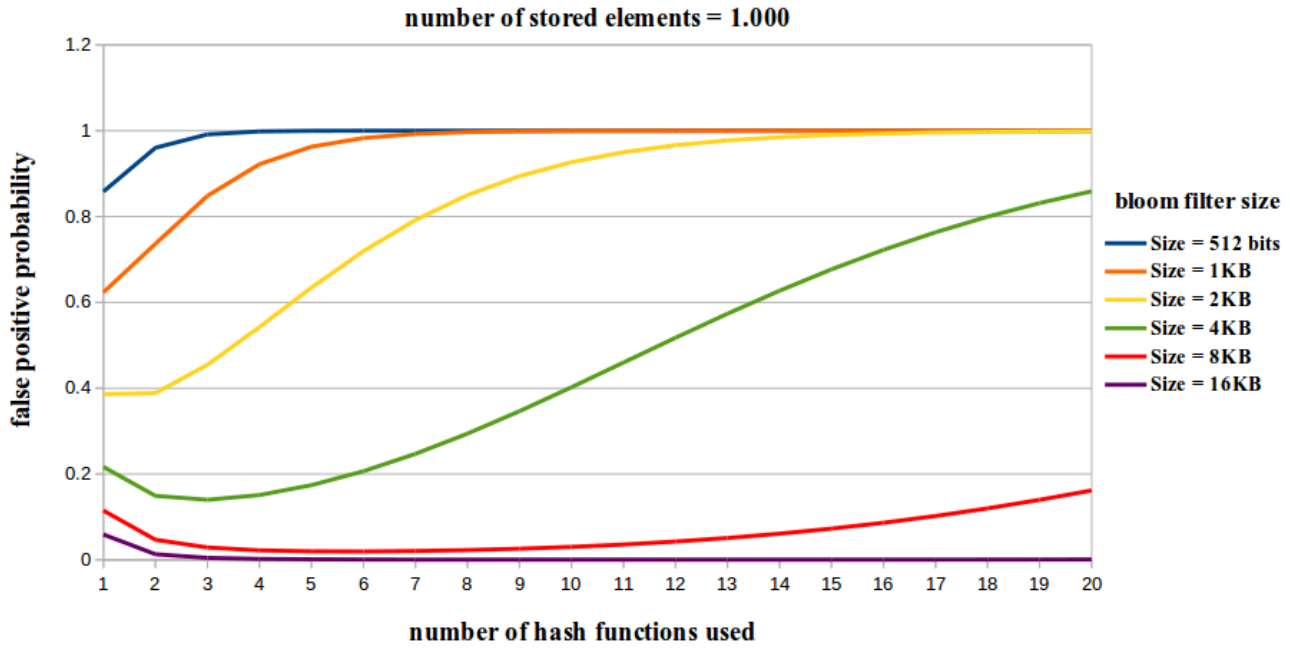
Στην ενότητα αυτή, θα μελετηθούν τα παρακάτω ζητήματα λειτουργίας των bloom filters:

- εξάρτηση της πιθανότητας false positive από τις παραμέτρους του φίλτρου.
- απαιτούμενος χρόνος αποθήκευσης στοιχείων στο bloom filter, καθώς μεταβάλλεται ο αριθμός των hash functions που χρησιμοποιούνται.
- σύγκριση του χρόνου αναζήτησης στοιχείου στο bloom filter με τον αντίστοιχο σε λίστα, χρησιμοποιώντας δυαδική και γραμμική αναζήτηση.

Ο καθοριστικότερος παράγοντας για την αποτελεσματικότητα ενός bloom filter είναι η **πιθανότητα false positive**, δηλαδή η πιθανότητα μία απόφαση ότι ένα στοιχείο ανήκει στο φίλτρο να είναι λανθασμένη. Όπως έχει ήδη αναφερθεί, η πιθανότητα αυτή εξαρτάται από το μέγεθος του φίλτρου, τον αριθμό των στοιχείων που αποθηκεύονται σε αυτό και τον αριθμό των hash functions που χρησιμοποιούνται για την αποθήκευση κάθε στοιχείου του.

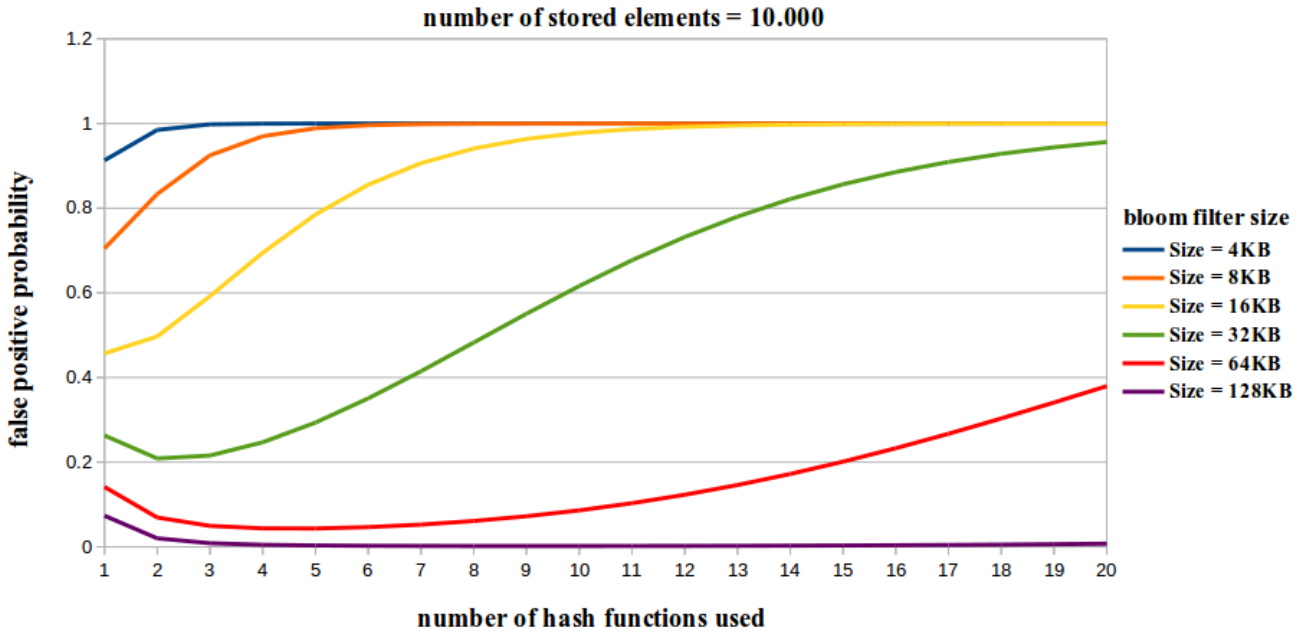
Στα παρακάτω τρία διαγράμματα, μελετάται η εξάρτηση της πιθανότητας false positive ενός bloom filter από τις παραμέτρους του, σύμφωνα με το **θεωρητικό** τύπο υπολογισμού της. Διατηρώντας σταθερό τον αριθμό των στοιχείων που αποθηκεύονται στο φίλτρο, αντίστοιχα **1.000** στοιχεία, **10.000** στοιχεία και **100.000** στοιχεία, τα παρακάτω διαγράμματα παρουσιάζουν την πιθανότητα false positive ενός bloom filter (κατακόρυφος άξονας τιμών) ως συνάρτηση του αριθμού των hash functions που χρησιμοποιεί το φίλτρο (οριζόντιος άξονας τιμών) και του μεγέθους του φίλτρου (χρωματιστές γραμμές).

Το πρώτο διάγραμμα παρουσιάζει την περίπτωση για **1.000 αποθηκευμένα στοιχεία**:



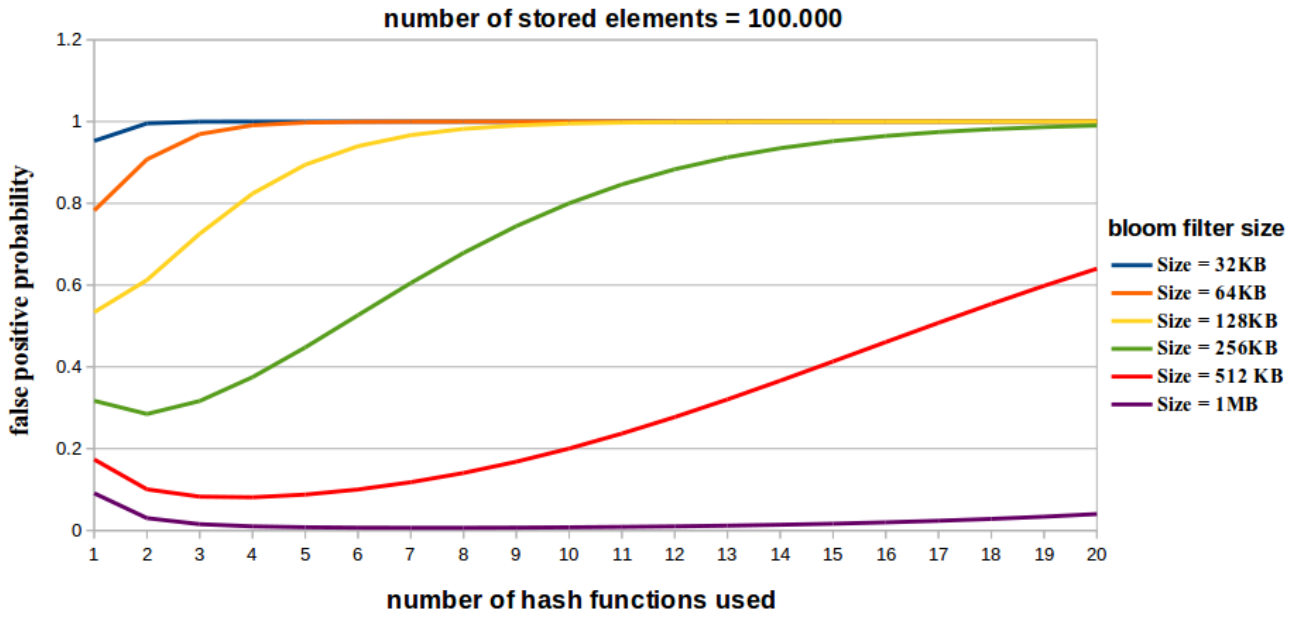
Σχήμα 5.1 – πιθανότητα false positive ενός bloom filter σε συνάρτηση με το μέγεθος του φίλτρου και τον αριθμό των hash functions που χρησιμοποιεί για 1.000 αποθηκευμένα στοιχεία

Το δεύτερο διάγραμμα παρουσιάζει την περίπτωση για **10.000 αποθηκευμένα στοιχεία**:



Σχήμα 5.2 - πιθανότητα false positive ενός bloom filter σε συνάρτηση με το μέγεθος του φίλτρου και τον αριθμό των hash functions που χρησιμοποιεί για 10.000 αποθηκευμένα στοιχεία

Το τρίτο διάγραμμα παρουσιάζει την περίπτωση για **100.000 αποθηκευμένα στοιχεία**:



Σχήμα 5.3 - πιθανότητα false positive ενός bloom filter σε συνάρτηση με το μέγεθος του φίλτρου και τον αριθμό των hash functions που χρησιμοποιεί για 100.000 αποθηκευμένα στοιχεία

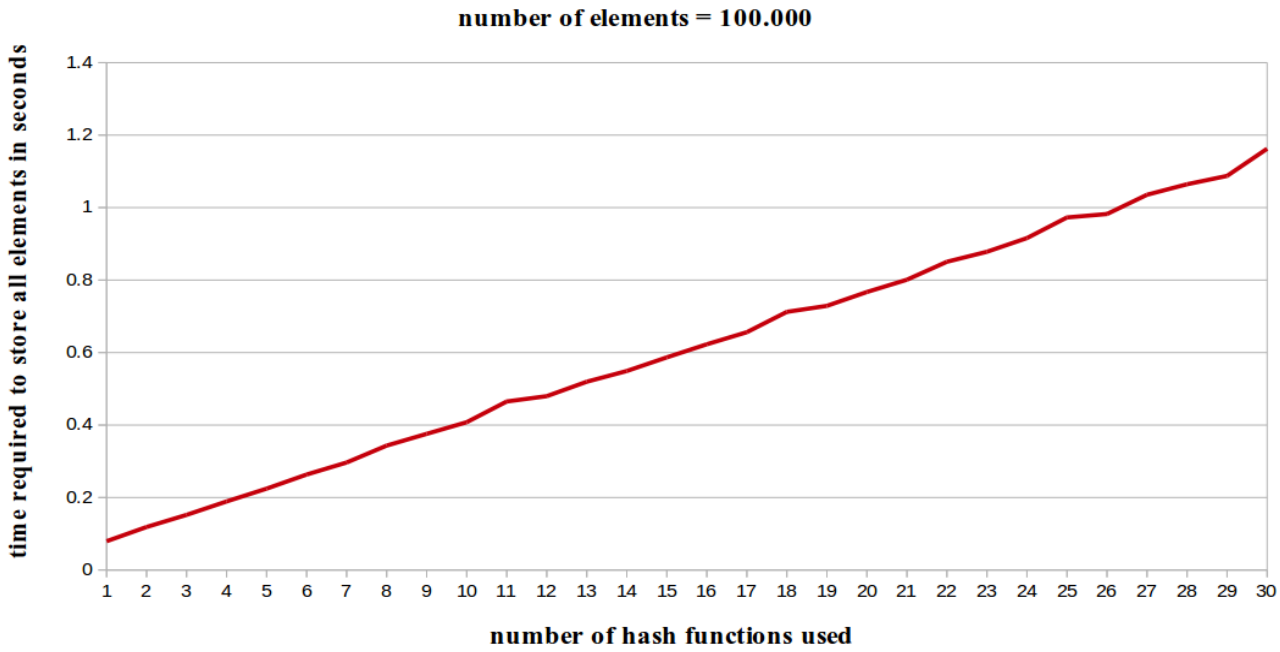
Παρατηρώντας το **τρίτο** διάγραμμα, εξάγονται τα παρακάτω συμπεράσματα:

- Ένα bloom filter μεγέθους **1 MB** μόνο είναι ικανό να αποθηκεύσει **100.000** στοιχεία με πιθανότητα false positive πολύ κοντά στο μηδέν. Αυτό σημαίνει ότι ένα πολύ μικρό φίλτρο είναι ικανό να προστατεύσει **αποτελεσματικά** πολύ μεγάλες ζώνες DNS, δηλαδή ζώνες με μεγάλο πλήθος DNS εγγραφών. Η ελάχιστη τιμή της πιθανότητας false positive στο διάγραμμα είναι **0.65%** για μέγεθος φίλτρου 1 MB και αριθμό hash functions 7.

- Αυξάνοντας το μέγεθος του bloom filter, η πιθανότητα false positive τείνει να μηδενιστεί.

- Συνεχής αύξηση του αριθμού των hash functions που χρησιμοποιεί ένα bloom filter δε συνεπάγεται απαραίτητα μείωση της πιθανότητας false positive του φίλτρου, καθώς πολύ μεγάλος αριθμός από hash functions αυξάνει τις συγκρούσεις στοιχείων στο φίλτρο. Συνεπώς, ο αριθμός των hash functions ενός bloom filter πρέπει να επιλέγεται προσεχτικά, ώστε να μην είναι ούτε πολύ μικρός, αλλά ούτε και πολύ μεγάλος.

Λαμβάνοντας ως δεδομένο τον αριθμό των στοιχείων που θα αποθηκευτούν στο bloom filter, η επιλογή του αριθμού των hash functions που θα χρησιμοποιηθούν καθορίζει εκτός από την τιμή της πιθανότητας false positive και το **χρόνο αποθήκευσης** των στοιχείων στο φίλτρο. Στο διάγραμμα που ακολουθεί στη συνέχεια, παρουσιάζεται η εξάρτηση του χρόνου που απαιτείται για την αποθήκευση **100.000** στοιχείων σε ένα bloom filter από τον αριθμό των hash functions που χρησιμοποιούνται, όταν εκείνος μεταβάλλεται από 1 έως 30. Στον κατακόρυφο άξονα παρουσιάζεται ο απαιτούμενος χρόνος προσθήκης των 100.000 στοιχείων στο bloom filter σε δευτερόλεπτα και στον οριζόντιο άξονα φαίνεται ο αριθμός των hash functions που χρησιμοποιούνται. Οι μετρήσεις πραγματοποιήθηκαν πάνω στον κώδικα που χρησιμοποιήθηκε στο αρχείο bloom.py (class BloomFilter):



Σχήμα 5.4 – εξάρτηση του χρόνου αποθήκευσης 100.000 στοιχείων σε ένα bloom filter από τον αριθμό των hash functions που χρησιμοποιούνται για την αποθήκευση κάθε στοιχείου

Παρατηρούμε ότι ο χρόνος αποθήκευσης στοιχείων σε ένα bloom filter αυξάνεται μονότονα, καθώς μεγαλώνει ο αριθμός των hash functions που χρησιμοποιούνται στο φίλτρο. Ωστόσο, για συνηθισμένες τιμές αριθμού από hash functions, π.χ. 7-12, παρατηρούμε ότι απαιτείται μόλις μισό δευτερόλεπτο για την αποθήκευση όλων των στοιχείων.

Τέλος, αξίζει να εξεταστεί πόσο μικρότερος είναι ο **χρόνος αναζήτησης** για το αν ένα στοιχείο βρίσκεται αποθηκευμένο στο bloom filter (χρονική πολυπλοκότητα $\Theta(1)$) από την αντίστοιχη περίπτωση δυαδικής αναζήτησης (χρονική πολυπλοκότητα $O(\log n)$) και γραμμικής αναζήτησης (χρονική πολυπλοκότητα $O(n)$). Πραγματοποιώντας αναζήτηση σε

ένα bloom filter στο οποίο έχουν αποθηκευτεί 100.000 στοιχεία (αριθμός από hash functions 10) παράλληλα με δυαδική και γραμμική αναζήτηση σε μία λίστα στην οποία έχουν αποθηκευτεί τα ίδια 100.000 στοιχεία για ένα μη αποθηκευμένο στοιχείο, παίρνουμε τα παρακάτω αποτελέσματα:

- χρόνος αναζήτησης στο bloom filter: 1.00135803223e-05 sec
- χρόνος δυαδικής αναζήτησης στη λίστα: 1.3113021850585938e-05 sec
- χρονος γραμμικής αναζήτησης στη λίστα: 0.013561964035 sec

Παρατηρούμε, δηλαδή, ότι ο χρόνος αναζήτησης βελτιώνεται κατά **1.31** φορές από τον αντίστοιχο για δυαδική αναζήτηση και κατά **1354** φορές για τον αντίστοιχο με γραμμική αναζήτηση.

5.2: Αποτελέσματα αλγορίθμου μηχανικής μάθησης

Βασικό τμήμα του μηχανισμού άμυνας αποτελεί ο **αλγόριθμος μηχανικής μάθησης** που αποφασίζει εάν ο authoritative εξυπηρετητής DNS βρίσκεται σε κίνδυνο ή όχι, βασισμένος σε **δύο** features, τα οποία υπενθυμίζονται στη συνέχεια:

- **CPU:** μέσο ποσοστό χρήσης του επεξεργαστή του authoritative DNS server τα 4 τελευταία δευτερόλεπτα λειτουργίας του.

- **DNS_RATIO:** ποσοστό απαντήσεων DNS με κωδικό NXDOMAIN προς τα συνολικά DNS ερωτήματα που δέχτηκε ο server τα τελευταία 15 δευτερόλεπτα λειτουργίας του.

Στη συνέχεια, ακολουθούν ενδεικτικά κάποια ζεύγη εισόδων – εξόδων του αλγορίθμου. Η έξοδος 1 υποδηλώνει ότι ο εξυπηρετητής DNS βρίσκεται σε κίνδυνο, ενώ ή έξοδος 0 υποδηλώνει ότι είναι ασφαλής. Οι τιμές των features δίνονται σε ποσοστό επί τοις εκατό.

CPU	DNS_RATIO	έξοδος
10	10	0
20	20	0
30	30	0
40	40	0
50	50	1
50	30	0
50	40	1
50	60	1
50	70	1
60	60	1
60	10	0
60	25	0
60	30	0
60	35	1
70	20	0
70	25	1
80	15	0
80	20	1
80	25	1
85	5	0
85	10	0
85	15	1
85	20	1
85	25	1
90	5	0
90	10	0
90	15	1
90	20	1
90	25	1
95	5	0
95	10	1
100	5	1
100	20	1

Πίνακας 5.1 – Ενδεικτικά ζεύγη εισόδων – εξόδων του αλγορίθμου μηχανικής μάθησης

Παρατηρώντας τις παραπάνω τιμές, μπορούμε να καταλήξουμε στα παρακάτω συμπεράσματα για τη **συμπεριφορά** του αλγορίθμου μηχανικής μάθησης:

- Ο αλγόριθμος αποφασίζει ότι ο εξυπηρετητής είναι ασφαλής όταν η τιμή του feature CPU είναι μικρότερη του 50% και η τιμή του feature DNS_RATIO είναι μικρότερη του 40%.

- Για τιμές του feature DNS_RATIO μεγαλύτερες του 40%, ο αλγόριθμος αποφασίζει ότι ο εξυπηρετητής κινδυνεύει ανεξάρτητα από την τιμή του feature CPU.

- Για τιμές του feature CPU που αυξάνονται από 50% έως 100%, η απαίτηση για μεγάλη τιμή του feature DNS_RATIO μειώνεται σταδιακά από την τιμή 40% έως την τιμή 5%.

Τα παραπάνω αποδεικνύουν ότι η συμπεριφορά του αλγορίθμου μηχανικής μάθησης είναι η **επιθυμητή**, καθώς είναι σε θέση να ταξινομεί ορθά τις επικίνδυνες εισόδους και είναι πιο ελαστικός στις αποφάσεις του όσο πιο μικρή είναι η τιμή του feature CPU, που είναι εκείνο που καθορίζει την ικανότητα εξυπηρέτησης του server.

5.3: Αξιολόγηση μηχανισμού άμυνας

Η αξιολόγηση της αποτελεσματικότητας του μηχανισμού ανίχνευσης και αντιμετώπισης της επίθεσης DNS water torture που υλοποιήθηκε βασίζεται στα παρακάτω **κριτήρια**:

- την ικανότητα του μηχανισμού δειγματοληψίας μέσω sFlow να εντοπίζει τις διευθύνσεις IP από τις οποίες προέρχεται κακόβουλη κίνηση, έγκαιρα και σωστά, δηλαδή χωρίς να χαρακτηρίζει ως επίκινδυνες, διευθύνσεις που δεν πρέπει.

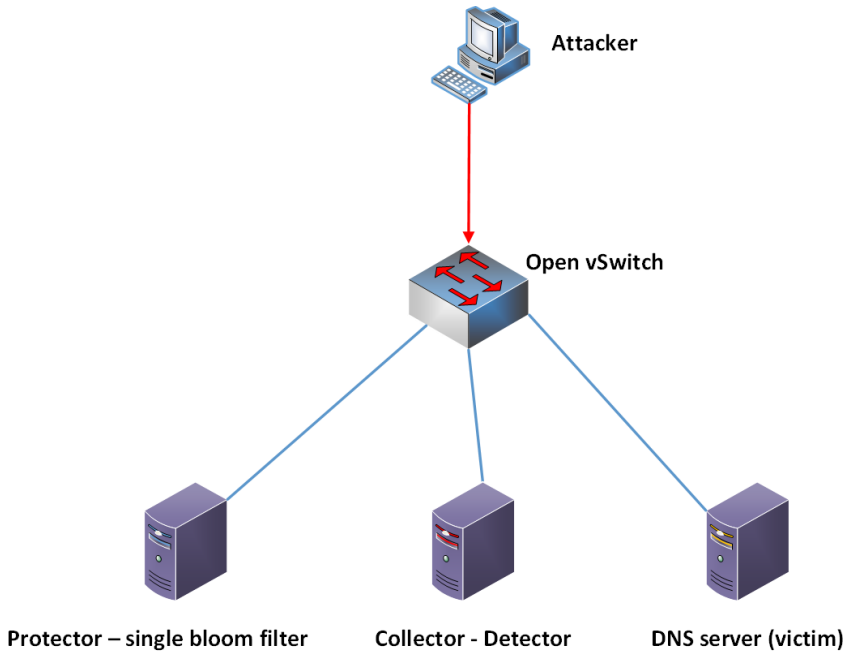
- Τη διαφορά του ποσοστού χρήσης του επεξεργαστή (CPU utilization) του authoritative εξυπηρετητή DNS κατά τη διάρκεια της επίθεσης, όταν χρησιμοποιείται και όταν δε χρησιμοποιείται ο μηχανισμός άμυνας.

- Ο αριθμός των ουσιαστικών ερωτήσεων DNS στις οποίες απαντά ο authoritative DNS server κατά τη διάρκεια της επίθεσης, όταν αφήνεται να λειτουργήσει απροστάτευτος και όταν είναι ενεργοποιημένος ο μηχανισμός άμυνας.

Για το σκοπό αυτό, πραγματοποιήθηκε πείραμα στο πρόγραμμα εικονικοποίησης υπολογιστικών συστημάτων **VirtualBox** [55], χρησιμοποιώντας εικονικά μηχανήματα με λειτουργικό σύστημα Linux Ubuntu 16.04 [56]. Για την εξομοίωση του OpenFlow switch του δικτύου χρησιμοποιήθηκε το λογισμικό **Open vSwitch**, ενώ για τη μονάδα Protector χρησιμοποιήθηκε ένα μοναδικό bloom filter. Η μονάδα Collector – Detector χρησιμοποιήθηκε αυτούσια, χωρίς κάποια απλούστευση.

Στον εξυπηρετητή DNS φορτώθηκε το αρχείο ζώνης του authoritative εξυπηρετητή dolly.netmode.ece.ntua.gr του εργαστηρίου netmode, το οποίο περιλαμβάνει περίπου 250 ονόματα. Αν και πολύ μικρός, ο αριθμός αυτός θεωρούμε πως δεν επηρεάζει το πείραμα για την αξιολόγηση του μηχανισμού, καθώς επιλέχτηκε ένα σχετικά μεγάλο σε μέγεθος φίλτρο. Είδαμε, προηγουμένως, ότι ένα bloom filter μεγέθους 1 MB μπορεί να αποθηκεύσει 100.000 ονόματα με πολύ μικρή πιθανότητα false positive. Το μέγεθος αυτό (ή και παραπάνω) είναι πολύ μικρό, ώστε να είναι διατεθειμένος να το υιοθετήσει οποιοσδήποτε οργανισμός για την προστασία των εξυπηρετητών του.

Η πειραματική διάταξη παρουσιάζεται στο ακόλουθο σχήμα:



Σχήμα 5.5 – η πειραματική διάταξη για την αξιολόγηση του μηχανισμού άμυνας

Για την πραγματοποίηση του πειράματος, δημιουργήθηκε αρχικά μίγμα καλόβουλης και κακόβουλης κίνησης DNS, αξιοποιώντας τη βιβλιοθήκη **Scapy** της Python. Η κίνηση αυτή αποτελείται από δύο κατηγορίες, καθεμία από τις οποίες έχει τα παρακάτω χαρακτηριστικά:

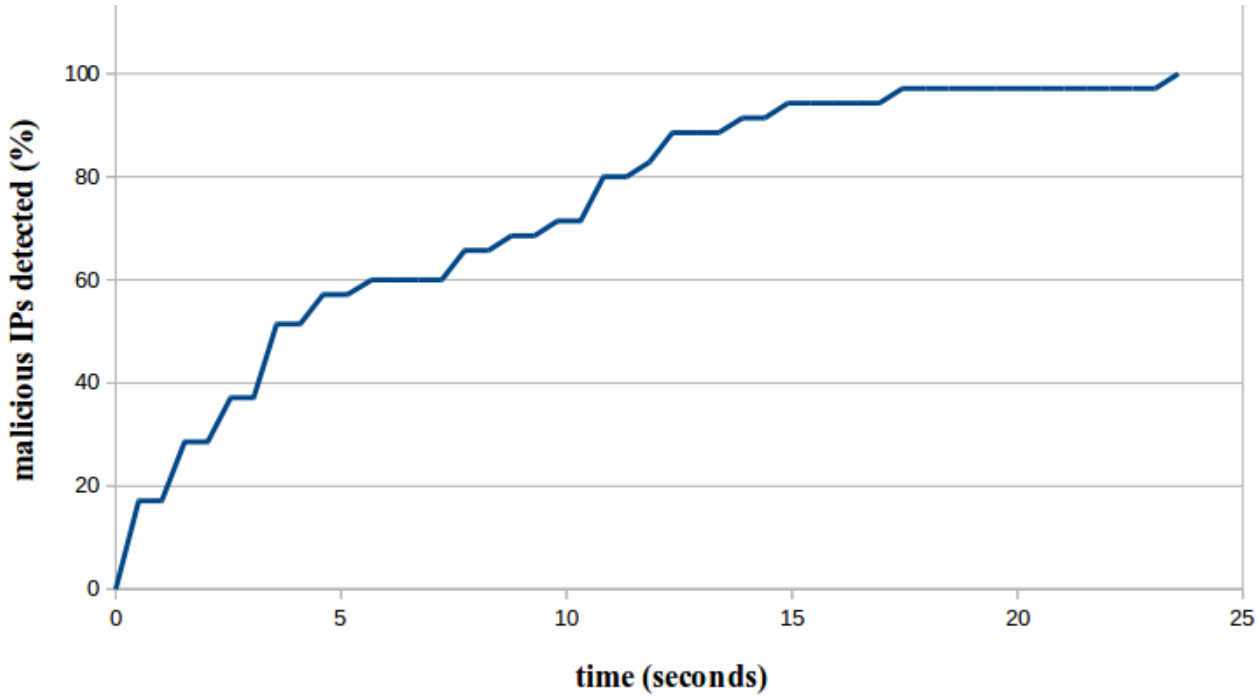
- **Κατηγορία 1:** η πρώτη κατηγορία περιλαμβάνει την κίνηση που προωθείται από τους recursive εξυπηρετητές DNS, οι οποίοι συμμετέχουν στην επίθεση DNS water torture. Κατά συνέπεια, η κίνηση αυτή θα καταφθάνει στον authoritative server με πολύ μεγάλη ταχύτητα με σκοπό να εξαντλήσει τους πόρους του. Η κατηγορία αυτή περιέχει ερωτήματα DNS από **36** διαφορετικές διευθύνσεις IP, καθεμία από τις οποίες περιλαμβάνει 30.000 πακέτα (συνολικά, δηλαδή, 1.080.000 πακέτα). Το ποσοστό των ερωτημάτων για ανύπαρκτα ονόματα δεν είναι το ίδιο για όλες τις IP. Για τις 10 πρώτες είναι 80%, για τις επόμενες 15 είναι 90% και για τις υπόλοιπες 11 είναι 100%, δηλαδή όλες οι ερωτήσεις αυτές θα οδηγήσουν σε απαντήσεις με κωδικό NXDOMAIN.

- **Κατηγορία 2 :** η δεύτερη κατηγορία περιλαμβάνει φυσιολογική κίνηση που προωθείται από recursive εξυπηρετητές DNS, οι οποίοι δε συμμετέχουν στην επίθεση DNS water torture. Αποτελείται από **64** διαφορετικές IP, καθεμία από τις οποίες περιλαμβάνει 150 πακέτα (συνολικά 9600), τα οποία θα καταφθάνουν στον authoritative server σε λογικές ταχύτητες (π.χ. 100 packets per second). Η επιλογή των συγκεκριμένων τιμών έγινε γιατί, σε συνθήκες φυσιολογικής λειτουργίας, καταφθάνουν στον authoritative server λίγα ερωτήματα DNS από πολλές διαφορετικές διευθύνσεις IP.

Συνολικά, λοιπόν, το **dataset** της επίθεσης περιλαμβάνει 1.089.000 ερωτήματα DNS, εκ των οποίων μόλις **114.600** ερωτήματα είναι ουσιαστικά μηνύματα, δηλαδή αφορούν υπαρκτούς πόρους.

Οι δύο κατηγορίες κίνησης της επίθεσης, λοιπόν, παράγονται με τη βιβλιοθήκη Scapy της Python και συλλαμβάνονται με την εντολή **tcpdump** [57] σε αρχεία pcap, τα οποία, στη συνέχεια, επαναλαμβάνονται από τον επιτιθέμενο (υπολογιστής **attacker** του σχήματος) στην επιθυμητή ταχύτητα με την εντολή **tcpreplay** [58].

Αρχικά, εξετάζεται πόσο ικανός είναι ο **μηχανισμός δειγματοληψίας με sFlow** να εντοπίσει τις 36 επικίνδυνες IP έγκαιρα και σωστά. Στο παρακάτω διάγραμμα παρουσιάζεται η αναγνώριση των επικίνδυνων IP από το μηχανισμό για δειγματοληψία ρυθμού **1 προς 128**, όπως εξελίσσεται στο χρόνο:

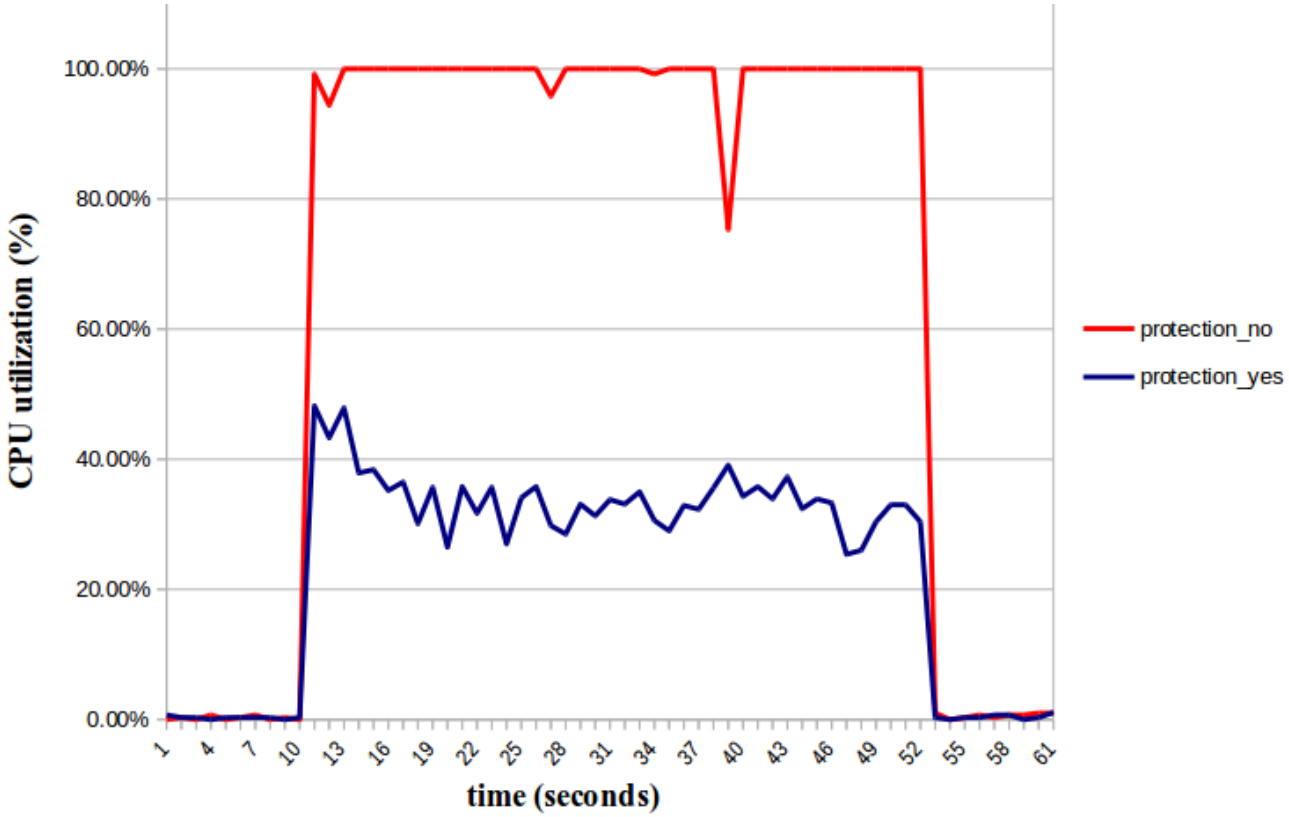


Σχήμα 5.6 – απαραίτητος χρόνος αναγνώρισης των επικίνδυνων IP με ρυθμό δειγματοληψίας 1/128

Παρατηρούμε ότι η αναγνώριση **όλων** των επικίνδυνων IP πραγματοποιείται σε περίπου **24** δευτερόλεπτα, δηλαδή σε πολύ ικανοποιητικό χρόνο. Μάλιστα, η αναγνώριση των μισών από τις επικίνδυνες IP πραγματοποιείται σε χρόνο λιγότερο των **5** δευτερολέπτων. Ο συγκεκριμένος ρυθμός δειγματοληψίας αποδεικνύεται αποτελεσματικός και, μάλιστα, ιδιαίτερα ικανοποιητικός σε σύγκριση με εκείνους που υποδεικνύονται από την εμπειρία [65].

Έπειτα, στο παρακάτω διάγραμμα αντιπαραβάλλονται το **ποσοστό χρήσης του επεξεργαστή** του authoritative DNS server όταν δε χρησιμοποιείται ο μηχανισμός άμυνας (φαίνεται με κόκκινο χρώμα) με εκείνο όταν χρησιμοποιείται ο μηχανισμός άμυνας

(φαίνεται με μπλε χρώμα). Παρατηρώντας το, γίνεται φανερό ότι ο μηχανισμός άμυνας είναι ικανός να προστατεύσει αποτελεσματικά το θύμα από την επίθεση.



Σχήμα 5.7 – ποσοστό χρήσης CPU θύματος όταν δε χρησιμοποιείται μηχανισμός άμυνας (κόκκινη γραμμή) και όταν χρησιμοποιείται (μπλε γραμμή)

Τέλος, στον παρακάτω πίνακα παρουσιάζονται οι **απώλειες** ουσιαστικών μηνυμάτων DNS, δηλαδή τα μηνύματα που αδυνατεί να εξυπηρετήσει το θύμα λόγω εξάντλησης των πόρων του χωρίς και με χρήση του μηχανισμού άμυνας:

Κατάσταση μηχανισμού άμυνας	**Ποσοστό απώλειας ουσιαστικών μηνυμάτων**
απενεργοποιημένος	74.36%
ενεργοποιημένος	2.18%

Πίνακας 5.2 – ποσοστό απώλειας ουσιαστικών μηνυμάτων χωρίς και με μηχανισμό άμυνας

6 Επίλογος

Στο κεφάλαιο αυτό παρατίθενται τα συμπεράσματα της εργασίας και παρουσιάζονται προτάσεις για την επέκταση του μηχανισμού που αφήνονται ως μελλοντική εργασία.

6.1: Σύνοψη και Συμπεράσματα

Στα πλαίσια της παρούσης διπλωματικής εργασίας υλοποιήθηκε και αξιολογήθηκε ένας μηχανισμός ανίχνευσης και αντιμετώπισης μιας επίθεσης DNS water torture που είχε ως στόχο έναν authoritative εξυπηρετητή DNS. Ο μηχανισμός αυτός συλλέγει πληροφορίες από τη λειτουργία του εξυπηρετητή και την κίνηση που διακινείται στο τοπικό του δίκτυο, αποφασίζει με τη βοήθεια ενός αλγορίθμου μηχανικής μάθησης εάν ο εξυπηρετητής βρίσκεται σε κίνδυνο ή όχι και, στην πρώτη περίπτωση, φιλτράρει τα ερωτήματα DNS από διευθύνσεις IP που έχουν χαρακτηριστεί ως κακόβουλες, χρησιμοποιώντας μία μονάδα βασισμένη σε bloom filters.

Ο κύριος σκοπός της εργασίας ήταν να εξετάσει εάν τα **bloom filters** αποτελούν κατάλληλη επιλογή για την αντιμετώπιση των επιθέσεων DNS water torture. Όπως αποδείχτηκε από τα αποτελέσματα της πειραματικής διαδικασίας, τα bloom filters είναι μία **πολύ καλή επιλογή** απέναντι σε μια τέτοια επίθεση. Μπορούν να προστατεύσουν αποτελεσματικά έναν εξυπηρετητή DNS, χωρίς να καθυστερήσουν ιδιαίτερα τις υπηρεσίες που προσφέρει και με ελάχιστες απώλειες ουσιαστικών ερωτημάτων DNS. Η μελέτη αυτή καταδεικνύει ότι μηχανισμοί βασισμένοι σε bloom filters είναι απαραίτητο να υιοθετηθούν από authoritative εξυπηρετητές DNS.

6.2: Μελλοντικές επεκτάσεις

Ο μηχανισμός που υλοποιήθηκε περιορίστηκε στην αντιμετώπιση της επίθεσης DNS water torture στο **τοπικό δίκτυο** του **authoritative** εξυπηρετητή DNS, ο οποίος αποτελεί το θύμα της επίθεσης. Ωστόσο, η αντιμετώπιση μιας επίθεσης DDoS είναι πολύ πιο αποτελεσματική όταν πραγματοποιείται κοντά στις πηγές της. Ως μελλοντική επέκταση, λοιπόν, αφήνεται η υλοποίηση ενός συνεργατικού μηχανισμού (collaboration scheme) βασισμένου σε bloom filters ανάμεσα σε recursive εξυπηρετητές DNS και τους authoritive εξυπηρετητές DNS που βρίσκονται σε κίνδυνο.

Όταν η μονάδα ανίχνευσης κινδύνου ενός authoritative DNS server αποφασίσει ότι ο εξυπηρετητής κινδυνεύει, δε λαμβάνει μέτρα μόνο για το φιλτράρισμα των μηνυμάτων DNS για IPs που έχουν χαρακτηριστεί ως κακόβουλες. Παράλληλα, ενημερώνει τις διευθύνσεις αυτές ότι προωθούν κακόβουλη κίνηση και τους μεταφέρει τα περιεχόμενα του

bloom filter που διαθέτει. Επομένως, οι recursive εξυπηρετητές θα χρησιμοποιήσουν δικά τους bloom filters για να φιλτράρουν τα ερωτήματα που προωθούν πιο κοντά στις πηγές της επίθεσης.

Σε περίπτωση που ένας recursive εξυπηρετητής δεν επιθυμεί ή δεν έχει τη δυνατότητα να συνεργαστεί, η άμυνα απέναντι στην επίθεση για την κακόβουλη κίνηση που προωθεί ο συγκεκριμένος server θα περιορίζεται στο μηχανισμό που αναπτύχθηκε στα πλαίσια της παρούσης διπλωματικής εργασίας.

Βιβλιογραφία

[1] akamai's [state of the internet] / security Q1 2017 report, Available online: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-security-report.pdf

[2] Dyn Analysis Summary Of Friday October 21 Attack, Available online: https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/

[3] A. Sreekanth, P. Sri, T. Vartiainen, "Dyn DDOS Cyberattack – a case study", Aalto University, Finland

[4] Internet of Threats: IoT Botnets and the Economics of DDoS Protection, Available online: https://security.radware.com/ddos-threats-attacks/ddos-attack-types/iot-botnets-and-economics/

[5] J. Kurose and K. Ross, Computer Networking: A Top Down Approach Using the Internet, Addison-Wesley Computer Science, 6th Edition, 2013.

[6] DNS, Domain Name System, Network Sorcery, Available online: http://www.networksorcery.com/enp/protocol/dns.htm

[7] DNS Message Header and Question Section Format, The TCP/IP Guide, Available online: http://www.tcpipguide.com/free/t_DNSMessageHeaderandQuestionSectionFormat.htm

[8] IP Spoofing, Incapsula, Available online: https://www.incapsula.com/ddos/ip-spoofing.html

[9] Botnet DDOS attacks, Incapsula, Available online: https://www.incapsula.com/ddos/botnet-ddos.html

[10] Z. Zhu, G. Lu, Y. Chen, Z. Fu, P. Roberts, K. Han, "Botnet Research Survey", Available online: https://pdfs.semanticscholar.org/70a4/791f7d9f5a42ee09d94f2be5dbf2645a30d9.pdf

[11] S. T. Zargar, J. Joshi, D. Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks", Available online: http://d-scholarship.pitt.edu/19225/1/FinalVersion.pdf

[12] DDOS attacks, Incapsula, Available online: https://www.incapsula.com/ddos/ddos-attacks/

[13] Three Types of DDoS Attacks, A. Kesavan (article), Available online: https://blog.thousandeyes.com/three-types-ddos-attacks/

[14] Distributed Denial-of-Service (DDoS) Attacks: An Economic Perspective, A business whitepaper from nsfocus, Available online: https://www.infosecurityeurope.com/__novadocuments/264689?v=636075474758900000

[15] “9 Reasons for Denial-Of-Service (DoS) Attacks: Why Do They Happen?”, L. Zeltser (article), Available online: https://zeltser.com/reasons-for-denial-of-service-attacks/

[16] DNS flood, Incapsula, Available online: https://www.incapsula.com/ddos/attack-glossary/dns-flood.html

[17] Y. Takeuchi, T. Yoshida, R. Kobayashi, M. Kato, H. Kishimoto, “ Detection of the DNS Water Torture Attack by Analyzing Features of the Subdomain Name”, Available online: https://www.jstage.jst.go.jp/article/ipsjjip/24/5/24_793/_pdf

[18] Local DNS Attack Lab, SEED Labs, Syracuse University, Available online: http://www.cis.syr.edu/~wedu/seed/Labs_12.04/Networking/DNS_Local/DNS_Local.pdf

[19] Remote DNS Cache Poisoning Attack Lab, SEED Labs, Syracuse University, Available online: http://www.cis.syr.edu/~wedu/seed/Labs_12.04/Networking/DNS_Remote/DNS_Remote.pdf

[20] Domain Name System Security Extensions, Wikipedia, Available online: https://en.wikipedia.org/wiki/Domain_Name_System_Security_Extensions

[21] DNS amplification, Incapsula, Available online: https://www.incapsula.com/ddos/attack-glossary/dns-amplification.html

[22] plane (in Networking), Available online: http://whatis.techtarget.com/definition/plane-in-networking

[23] Q. Niyaz, W. Sun, A.Y. Javaid, “A Deep Learning Based DDoS Detection System in Software-Defined Networking (SDN)”, Available online: https://arxiv.org/pdf/1611.07400.pdf

[24] OpenFlow protocol, Available online: http://archive.openflow.org

[25] “Eight Big Benefits of Software-Defined Networking”, N. Sharma (article), Available online: http://www.serverwatch.com/server-tutorials/eight-big-benefits-of-software-defined-networking.html

[26] OpenFlow Switch Specification, vesion 1.0.0, Available online: http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf

[27] Open vSwitch, Available online: http://www.openvswitch.org/

[28] Open vSwitch, Wikipedia, Available online: https://en.wikipedia.org/wiki/Open_vSwitch

[29] NetFlow, Cisco IOS NetFlow, Available online: https://en.wikipedia.org/wiki/NetFlow

[30] NetFlow, Wikipedia, Available online: https://en.wikipedia.org/wiki/NetFlow

[31] sFlow, Available online: http://www.sflow.org/

[32] sFlow, Wikipedia, Available online: https://en.wikipedia.org/wiki/SFlow

[33] collectd – The system statistics collection daemon, Available online: https://collectd.org/

[34] RRDtool, Available online: https://oss.oetiker.ch/rrdtool/

[35] RRDtool, Wikipedia, Available online: https://en.wikipedia.org/wiki/RRDtool

[36] Machine Learning, Wikipedia, Available online: https://en.wikipedia.org/wiki/Machine_learning

[37] "Support Vector Machines: A Simple Explanation", N. Bambrick (article), Available online: http://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html

[38] Bloom filter, Wikipedia, Available online: https://en.wikipedia.org/wiki/Bloom_filter

[39] Bloom filters by Example, Available online: https://llimllib.github.io/bloomfilter-tutorial/

[40] Edit distance algorithm, Jeff Erickson, Lecture 5: Dynamic Programming, Available online: http://jeffe.cs.illinois.edu/teaching/algorithms/notes/05-dynprog.pdf

[41] Dynamic programming, Wikipedia, Available online: https://en.wikipedia.org/wiki/Dynamic_programming

[42] Representational state transfer, Wikipedia, Available online: https://en.wikipedia.org/wiki/Representational_state_transfer

[43] Introducing JSON, Available online: http://www.json.org/

[44] JSON, Wikipedia, Available online: https://en.wikipedia.org/wiki/JSON

[45] BIND open source DNS server, Available online: https://www.isc.org/downloads/bind/

[46] Ryu component-based software defined networking framework, Available online: https://github.com/osrg/ryu

[47] Ryu SDN controller documentation, Available online: http://ryu.readthedocs.io/en/latest/ryu_app_api.html

[48] Python Programming Language, Available online: http://www.python.org

[49] Open Source Search and Analytics – Elasticsearch | Elastic, Available online: https://www.elastic.co/

[50] Elasticsearch, Wikipedia, Available online: https://en.wikipedia.org/wiki/Elasticsearch

[51] NoSQL, Wikipedia, Available online: https://en.wikipedia.org/wiki/NoSQL

[52] sflowtool | sFlow collector, Available online: https://github.com/xchenum/sflowtool-mod

[53] iptables man page, Available online: http://ipset.netfilter.org/iptables.man.html

[54] Scapy, Available online: http://www.secdev.org/projects/scapy/

[55] Oracle VM VirtualBox, Available online: https://www.virtualbox.org/

[56] Ubuntu, Available online: https://www.ubuntu.com/

[57] tcpdump, Available online: http://www.tcpdump.org/

[58] tcpreplay, Available online: http://linux.die.net/man/1/tcpreplay

[59] S. Yadav, A. K. K. Reddy, A. L. Narashimha Reddy, S. Ranjan, "Detecting Algorithmically Generated Malicious Domain Names", Available online: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.221.1167&rep=rep1&type=pdf

[60] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, "OpenFlow: Enabling Innovation in Campus Networks", Available online: http://book.itep.ru/depository/open_flow/openflow-wp-latest.pdf

[61] A. Cowperthwaite, A. Somayaji, "The Futility of DNSSec", Available at: https://www.ccsl.carleton.ca/paper-archive/cowperthwaite-asia-10.pdf

[62] C-J. Lin, Support vector machines: status and challenges, Available at: http://www.csie.ntu.edu.tw/~cjlin/talks/caltech.pdf

[63] A. Osareh, B. Shadgar, "Intrusion Detection in Computer Networks based on Machine Learning Algorithms", Available online: http://rms.scu.ac.ir/Users/9-1357078/Articles/Journals/Osareh_IJCSNS_Final.pdf

[64] A. Antonopoulos, "Mastering Bitcoin", O'Reilly, Chapter 6, related article available online: http://chimera.labs.oreilly.com/books/1234000001802/ch06.html

[65] sFlow sampling rates, Available online: http://blog.sflow.com/2009/06/sampling-rates.html

Παράρτημα

Α. Κώδικας

1. αρχείο sflow.py

```
import subprocess
from elasticsearch import Elasticsearch
import json
import os
import sys
import pty

def fill_list():
        global listing
        dolly_names = open("dolly_contents.txt","r")
        listing = list()

        for line in dolly_names:
                listing.append(line.rstrip() + ".example.com.")

# https://stackoverflow.com/questions/2460177/edit-distance-in-python
# given by user Santosh

def edit_distance(s1, s2):
    m=len(s1)+1
    n=len(s2)+1

    tbl = {}
    for i in range(m): tbl[i,0]=i
    for j in range(n): tbl[0,j]=j
    for i in range(1, m):
        for j in range(1, n):
            cost = 0 if s1[i-1] == s2[j-1] else 1
            tbl[i,j] = min(tbl[i, j-1]+1, tbl[i-1, j]+1, tbl[i-1, j-1]+cost)
    return tbl[i,j]


def call_and_peek_output(cmd, shell=False):
    # Create a new pair master, slave
    master, slave = pty.openpty()
    print cmd
    p = subprocess.Popen(cmd, shell=shell, stdin=None, stdout=slave, close_fds=True)
    os.close(slave)
    line = ""
    while True:
```

```
        try:
            ch = os.read(master, 1)
        except OSError:
            # We get this exception when the spawn process closes all references to the
            # pty descriptor which we passed him to use for stdout
            # (typically when it and its childs exit)
            break
        line += ch
        if ch == '\n':
            yield line
            line = ""
    if line:
        yield line

    ret = p.wait()
    if ret:
        raise subprocess.CalledProcessError(ret, cmd)

def find_minimum_edit_distance(name):
        global listing

        minimum = 100000
        for item in listing:
                comparison = edit_distance(item,name)
                if comparison < minimum:
                        minimum = comparison
        return minimum

def find_query_name(message):
        labels = dict()

        check_length = int(message[54],16)
        current_index = 54
        label_position = 0

        while True:
                string = ""
                for index in range(1,check_length+1):
                        string = string + message[current_index + index]

                labels[label_position] = string.decode("hex")
                label_position = label_position + 1

                current_index = current_index + int(message[current_index],16) + 1
                check_length = int(message[current_index],16)

                if int(message[current_index],16) == 0:
```

```
                break

        query_name = ""
        for label_position in labels:
                query_name = query_name + labels[label_position] + "."

        return query_name

def set_IP(srcIP,dstIP):
        if srcIP == "10.0.0.1":
                IP = dstIP
        else:
                IP = srcIP

        return IP

def update_database(minimum,IP):
        global es

        try:
                json_object = es.get(index='dns',doc_type='ip3',id=IP)
        except:
                es.index(index='dns',doc_type='ip3',id=IP,body={
                        "queries":0,
                        "nxdomain":0,
                        "malicious_total_length":0,
                        "malicious_average_length":0,
                        "dangerous":"no"
                })

                json_object = es.get(index='dns',doc_type='ip3',id=IP)

        queries = json_object['_source']["queries"]
        nxdomain = json_object['_source']["nxdomain"]
        malicious_total_length = json_object['_source']["malicious_total_length"]
        malicious_average_length = json_object['_source']["malicious_average_length"]

        queries = queries + 1
        if minimum != 0:
                nxdomain = nxdomain + 1
                malicious_total_length = malicious_total_length + minimum
                malicious_average_length = malicious_total_length/nxdomain

        dns_ratio = nxdomain/float(queries)
        if dns_ratio > 0.15 or malicious_average_length>2:
                dangerous = u"yes"
        else:
```

```
            dangerous = u"no"

        es.index(index='dns',doc_type='ip3',id=IP,body={
            "queries":queries,
            "nxdomain":nxdomain,
            "malicious_total_length":malicious_total_length,
            "malicious_average_length":malicious_average_length,
            "dangerous":dangerous
            })

def main_function():
        fill_list()
        global es
        es  = Elasticsearch([{'host':'localhost','port':9200}])

        for output in call_and_peek_output(["sflowtool"],shell=True):
            if "headerBytes" in output:
                headerBytes = output
                headerBytes = headerBytes[12:-2]
                headerBytes = headerBytes.split("-")
                etherType = headerBytes[12] + headerBytes[13]
            elif "srcIP" in output:
                srcIP = output
                srcIP = srcIP[6:-2]
            elif "dstIP" in output:
                dstIP = output
                dstIP = dstIP[6:-2]
            elif "IPProtocol" in output:
                proto = output
                proto = proto[11:-2]
            elif "UDPSrcPort" in output:
                srcPort = output
                srcPort = srcPort[11:-2]
            elif "UDPDstPort" in output:
                dstPort = output
                dstPort = dstPort[11:-2]
            elif "endSample" in output:

                if proto == "17" and etherType == "0800" and (dstPort == "53" or srcPort == "53"):
                    name = find_query_name(headerBytes)
                    IP = set_IP(srcIP,dstIP)
                    minimum = find_minimum_edit_distance(name)
                    update_database(minimum,IP)

if __name__ == "__main__":
        main_function()
```

2. αρχείο detect.py

```
import subprocess
from elasticsearch import Elasticsearch
import os
import sys
import pty
import time
import socket
import numpy as np
from sklearn import svm

def sum(item):
        sum = 0
        for number in item:
                sum = sum + float(number)
        return sum

def get_values(arg):
        values = arg.split(":")
        values = [x.split("\n") for x in values]
        values = [x[0] for x in values]
        values = values[1:]
        return values

def save_output(command):
        proc = subprocess.Popen(command,stdout=subprocess.PIPE,shell=True)
        (out,err)=proc.communicate()
        outwithoutreturn=out.rstrip('\n')
        return outwithoutreturn

def do_it_for_file(file_of_interest, starting, ending):
        command = "sudo rrdtool fetch " + file_of_interest + "  AVERAGE -r 1 -s -" + str(starting) + "
-e -" + str(ending) + " -a"
        out = save_output(command)
        values = get_values(out)
        summation = sum(values)
        return summation


def get_training_data():
        X = np.array([[0,45],
                      [25,45],
                      [45,45],
                      [45,40],
                      [45,35],
                      [45,30],
```

```
                [50,30],
                [60,24],
                [70,20],
                [80,14],
                [82,10],
                [82,5],
                [82,0],
                [0,55],
                [25,55],
                [45,55],
                [55,50],
                [55,40],
                [60,35],
                [70,30],
                [80,25],
                [92,10],
                [92,5],
                [92,0]])

        Y=[0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1]
        return X,Y

def svm_train():
        global clf
        X,Y = get_training_data()
        clf = svm.SVC(kernel = 'linear')
        clf.fit(X,Y)

def svm_predict(input):
        global clf
        prediction = clf.predict(input)
        return prediction

def return_prediction():
        global es

        query = do_it_for_file("/var/lib/collectd/rrd/server/dns/dns_qtype-A.rrd", 15, 1)
        nxdomain = do_it_for_file("/var/lib/collectd/rrd/server/dns/dns_rcode-NXDOMAIN.rrd", 15,
1)
        cpu = do_it_for_file("/var/lib/collectd/rrd/server/cpu-0/percent-idle.rrd", 4, 1)/4

        cpu_absolute = abs(cpu-100)
        print("NXDOMAIN",nxdomain)
        print("QUERY",query)

        try:
                dns_ratio = (nxdomain/query)*100
```

```
        except ZeroDivisionError:
            dns_ratio = 0

        print(cpu_absolute,dns_ratio)
        input_to_algorithm = [cpu_absolute,dns_ratio]
        prediction = svm_predict(input_to_algorithm)
        return prediction

def redirect_ips(malicious_ips):
        for ip in malicious_ips:
            print("prohibit",ip)
            rule = "'sudo curl -X POST -d
'{"dpid":"236382277540426","priority":"65535","idle_timeout":300,"hard_timeout":600,"actions":
[{"type":"SET_DL_DST","dl_dst":"08:00:27:07:40:85"},{"type":"OUTPUT","port":"3"}],"match":
{"dl_type":"0x0800","nw_proto":17,"tp_dst":53,"in_port":"4","dl_dst":"08:00:27:a8:6b:8c","nw_sr
c":"" + ip + "","nw_dst":"10.0.0.1"}}' http://127.0.0.1:8080/stats/flowentry/add'"
            os.system(rule)

def main_function():
        global clf
        global es

        es = Elasticsearch([{'host':'localhost','port':9200}])
        # first, train the machine learning algorithm
        svm_train()

        while True:
            malicious_ips = list()
            if return_prediction() == 1:
                matches = es.search(index='dns',doc_type='ip3',q='dangerous:u"yes"',size=100)
                hits = matches['hits']['hits']

                for item in hits:
                    ip = item["_id"]
                    malicious_ips.append(ip)

                redirect_ips(malicious_ips)

                while return_prediction() == 1:

                    matches = es.search(index='dns',doc_type='ip3',q='dangerous:u"yes"',size=100)
                    hits = matches['hits']['hits']

                    malicious_ips = list()

                    for item in hits:
                        ip = item["_id"]
```

```
                malicious_ips.append(ip)

            redirect_ips(malicious_ips)
            time.sleep(10)

main_function()
```

3. αρχείο iptables.txt

```
sudo sysctl -w net.ipv4.ip_forward = 1

sudo iptables -t nat -A PREROUTING -p udp --dport 53 -m state --state NEW -m statistic --mode
nth --every 5 --packet 0 -j DNAT --to-destination IP1:53
sudo iptables -t nat -A PREROUTING -p udp --dport 53 -m state --state NEW -m statistic --mode
nth --every 4 --packet 0 -j DNAT --to-destination IP2:53
sudo iptables -t nat -A PREROUTING -p udp --dport 53 -m state --state NEW -m statistic --mode
nth --every 3 --packet 0 -j DNAT --to-destination IP3:53
sudo iptables -t nat -A PREROUTING -p udp --dport 53 -m state --state NEW -m statistic --mode
nth --every 2 --packet 0 -j DNAT --to-destination IP4:53
sudo iptables -t nat -A PREROUTING -p udp --dport 53 -m state --state NEW -m statistic --mode
nth --every 1 --packet 0 -j DNAT --to-destination IP5:53
```

4. αρχείο bloom.py

```
#https://stackoverflow.com/questions/24666039/simple-raw-packet-sniffer-in-python

import socket
import sys
import binascii
import struct
from struct import *
from bitarray import bitarray
import mmh3
from scapy.all import *

def find_query_name(message):
    labels = dict()
    check_length = int(message[12],16)
    current_index = 12
    label_position = 0

    while True:
        string = ""
        for index in range(1,check_length+1):
            string = string + message[current_index + index]

        labels[label_position] = string.decode("hex")
```

```
            label_position = label_position + 1

            current_index = current_index + int(message[current_index],16) + 1
            check_length = int(message[current_index],16)

            if int(message[current_index],16) == 0:
                break

        query_name = ""
        for label_position in labels:
            query_name = query_name + labels[label_position] + "."

        return query_name

def examine_ethernet_header(packet):
        layer_2_header = packet[0:14]
        ethernet_info = struct.unpack("!6s6s2s",layer_2_header)
        destination_mac = binascii.hexlify(ethernet_info[0])
        source_mac = binascii.hexlify(ethernet_info[1])
        protocol = binascii.hexlify(ethernet_info[2])

        return destination_mac,source_mac,protocol

def examine_ip_header(packet):
        layer_3_header = packet[14:34]
        ip_header = struct.unpack("!9s1s2s4s4s",layer_3_header)
        tcp_or_udp = binascii.hexlify(ip_header[1])
        source_ip = socket.inet_ntoa(ip_header[3])
        destination_ip = socket.inet_ntoa(ip_header[4])
        return tcp_or_udp,source_ip,destination_ip

def examine_udp_header(packet):
        transport_header = packet[34:42]
        udp_header = struct.unpack("!2s2s2s2s",transport_header)
        source_port = binascii.hexlify(udp_header[0])
        dst_port = binascii.hexlify(udp_header[1])
        return source_port,dst_port

def examine_dns_header(packet):
        total_packet_length = len(packet)
        layer_4_header = packet[42:total_packet_length+1]
        dns_header = binascii.hexlify(layer_4_header).decode()
        dns_header = [dns_header[i:i+2] for i in range(0,len(dns_header),2)]
        query_name = find_query_name(dns_header)
        return query_name

# ownership of https://gist.github.com/mburst/4700640 Max Burstein (with minor modifications)
```

```
class BloomFilter:

    def __init__(self, size = 1024*4, hash_count = 10):
        '''
            constructor of class BloomFilter. Arguments:
            - size: size of bloom filter in bits
            - hash_count: number of hash functions the filter uses
        '''
        self.size = size  # the size of the bloom filter in bits
        self.hash_count = hash_count  # the number of hash functions the filter uses
        self.bit_array = bitarray(size)  # the bloom filter itself
        self.bit_array.setall(0)  # initialize bloom filter with zeroes

    def add(self, string):
        for seed in xrange(self.hash_count):
            result = mmh3.hash(string, seed) % self.size # modulo the size of the filter to ensure
            self.bit_array[result] = 1                   # the result is smaller than the size
                                                 # of the filter
    def query(self, string):
        for seed in xrange(self.hash_count):
            result = mmh3.hash(string, seed) % self.size
            if self.bit_array[result] == 0:
                return False
        return True

def create_bloom_filter():
        global bf
        bf  = BloomFilter()
        dolly_names = open("dolly_contents.txt","r")
        for line in dolly_names:
                to_add = line.rstrip() + ".example.com."
                bf.add(to_add)

def examine_and_parse_packet(packet):
        global bf
        sock = socket.socket(socket.PF_PACKET,socket.SOCK_RAW,socket.IPPROTO_IP)
        #s = conf.L2socket(iface='enp0s8')
        #destination_mac,source_mac,protocol = examine_ethernet_header(packet)
        tcp_or_udp,source_ip,destination_ip = examine_ip_header(packet)
        if tcp_or_udp == "11":
                source_port,dst_port = examine_udp_header(packet)
                if dst_port == "0035":
                        query_name = examine_dns_header(packet)
                        if bf.query(query_name) == True:
                                sock.sendto(packet,('enp0s8',53))
                                #scapy_packet = Ether(packet)
```

```
                #scapy_packet[IP].dst = "10.0.0.1"
                #del scapy_packet[IP].chksum
                #del scapy_packet[UDP].chksum
                #s.send(scapy_packet)

def main_function():
    rawSocket = socket.socket(socket.PF_PACKET,socket.SOCK_RAW,socket.htons(0x0800))
    create_bloom_filter()
    while True:
        # receive a packet
        packet = rawSocket.recv(128)
        examine_and_parse_packet(packet)

if __name__ == "__main__":
    main_function()
```

Β. Πηγές σχημάτων και πινάκων

Παρακάτω καταγράφονται οι πηγές των σχημάτων και των πινάκων που χρησιμοποιήθηκαν στην εργασία.

1. Πηγές σχημάτων

- **Σχήμα 2.1:** Available online: http://sophiedogg.com/wp-content/uploads/2013/09/dns_tree.jpg
- **Σχήμα 2.2:** Domain Name System, Wikipedia, Available online: https://el.wikipedia.org/wiki/Domain_Name_System#cite_note-1
- **Σχήμα 2.3:** Available online: https://docstore.mik.ua/orelly/networking_2ndEd/dns/figs/dns4_0209.gif
- **Σχήμα 2.4:** Root-Nameserver, Wikipedia, Available online: https://de.wikipedia.org/wiki/Root-Nameserver
- **Σχήμα 2.5:** M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou II, D. Dagon, "Detecting Malware Domains at the Upper DNS Hierarchy", Available online: https://www.usenix.org/legacy/event/sec11/tech/full_papers/Antonakakis.pdf
- **Σχήμα 2.6:** DNS, Domain Name System, Available online: http://www.networksorcery.com/enp/protocol/dns.htm
- **Σχήμα 2.7:** Available online: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRh6triwGmdBSFZTlEQFiSQwmOxVBXjOoH-3MMa9Yuf9u17FMGr
- **Σχήμα 2.8:** Available online: https://assets.paessler.com/media/scales/lightbox/common/files/blog/2017/ddos-attack-infographic.png
- **Σχήμα 2.9:** Available online: https://www.cheapsslshop.com/wp-content/uploads/2015/01/Spoofed-source-IP-address.png
- **Σχήμα 2.10:** Available online: https://cyberwurx.com/ddos-attack-ex.png
- **Σχήμα 2.11:** Y. Takeuchi, T. Yoshida, R. Kobayashi, M. Kato, H. Kishimoto, " Detection of the DNS Water Torture Attack by Analyzing Features of the Subdomain Name", Available online: https://www.jstage.jst.go.jp/article/ipsjjip/24/5/24_793/_pdf
- **Σχήμα 2.12:** Available online: https://upload.wikimedia.org/wikipedia/commons/e/eb/Dns-cache-poisoning.png
- **Σχήμα 2.13:** Remote DNS Cache Poisoning Attack Lab, SEED Labs, Syracuse University, Available online: http://www.cis.syr.edu/~wedu/seed/Labs_12.04/Networking/DNS_Remote/DNS_Remote.pdf
- **Σχήμα 2.14:** Available online: http://4.bp.blogspot.com/--TD71Sz8T_I/Ul8HTGfhzqI/AAAAAAAABN4/tmnAs7pZhpE/s1600/dns_amplification.png
- **Σχήμα 2.15:** Available online: https://cdn.thenewstack.io/media/2014/11/img1.png
- **Σχήμα 2.16:** Available online: http://deliveryimages.acm.org/10.1145/2670000/2661063/figs/f1.jpg
- **Σχήμα 2.17:** Available online: http://twimgs.com/networkcomputing/news/2012/SDN-Diagram-2.jpg

- **Σχήμα 2.18:** Available online: http://www1.hauman.com.tw/upload/Network/SDN/sdn04.png
- **Σχήμα 2.19:** Available online: https://image.slidesharecdn.com/sdn-140731201253-phpapp01/95/sdn-10-638.jpg?cb=1406837662
- **Σχήμα 2.20:** OpenFlow Switch Specification, vesion 1.0.0, Available online: http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf
- **Σχήμα 2.21:** OpenFlow Switch Specification, vesion 1.0.0, Available online: http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf
- **Σχήμα 2.22:** Availble online: https://upload.wikimedia.org/wikipedia/commons/thumb/c/c5/Distributed_Open_vSwitch_instance.svg/774px-Distributed_Open_vSwitch_instance.svg.png
- **Σχήμα 2.23:** NetFlow, Wikipedia, Available online: https://en.wikipedia.org/wiki/NetFlow
- **Σχήμα 2.24:** Available online: https://www.juniper.net/documentation/images/g021065.gif
- **Σχήμα 2.25:** Available online: http://www.loriotpro.com/Products/On-line_Documentation_V5/images/V22-A1_Introduction_RRD_fichiers/image001.jpg
- **Σχήμα 2.26:** "Support Vector Machines: A Simple Explanation", N. Bambrick (article), Available online: http://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html
- **Σχήμα 2.27:** Available online: http://dni-institute.in/blogs/wp-content/uploads/2015/09/SVM-Planes.png
- **Σχήμα 2.28:** "Support Vector Machines: A Simple Explanation", N. Bambrick (article), Available online: http://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html
- **Σχήμα 2.29:** Available online: http://www.qlikfix.com/wp-content/uploads/2014/03/Collision.png
- **Σχήμα 2.30:** Available online: https://venkateshnarayanan.files.wordpress.com/2013/09/092613_0919_bloomfilter1.jpg?w=714
- **Σχήμα 2.31:** A. Antonopoulos, "Mastering Bitcoin", O'Reilly, Chapter 6, related article available online: http://chimera.labs.oreilly.com/books/1234000001802/ch06.html
- **Σχήμα 2.32:** Edit distance algorithm, Jeff Erickson, Lecture 5: Dynamic Programming, Available online: http://jeffe.cs.illinois.edu/teaching/algorithms/notes/05-dynprog.pdf
- **Σχήμα 2.33:** Edit distance algorithm, Jeff Erickson, Lecture 5: Dynamic Programming, Available online:
- http://jeffe.cs.illinois.edu/teaching/algorithms/notes/05-dynprog.pdf
- **Σχήμα 4.1:** Ryu SDN controller documentation, Available online: http://ryu.readthedocs.io/en/latest/ryu_app_api.html

2.Πηγές Πινάκων

- **Πίνακας 2.1:** DNS, Domain Name System, Network Sorcery, Available online: http://www.networksorcery.com/enp/protocol/dns.htm
- **Πίνακας 2.2:** OpenFlow Switch Specification, vesion 1.0.0, Available online: http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf
- **Πίνακας 2.3:** OpenFlow Switch Specification, vesion 1.0.0, Available online:,

http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf
- **Πίνακας 2.4:** OpenFlow Switch Specification, vesion 1.0.0, Available online:, http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf
- **Πίνακας 4.1:** Ryu SDN controller documentation, Available online: http://ryu.readthedocs.io/en/latest/app/ofctl_rest.html
- **Πίνακας 4.2:** Ryu SDN controller documentation, Available online: http://ryu.readthedocs.io/en/latest/app/ofctl_rest.html