Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων
Τομεας Επικοινωνιων, Ηλεκτρονικης και Συστηματων
Πληροφορικης

Υλοποίηση Πιθανοτικών Δομών Δεδομένων στο Επίπεδο Δεδομένων για την Αποδοτική Αντιμετώπιση Επιθέσεων DDoS στο Σύστημα DNS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αναστασία Γ. Δημακοπούλου

Επιβλέπων : Βασίλειος Μάγκλαρης
Ομότιμος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2022

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων
Τομεας Επικοινωνιων, Ηλεκτρονικης και Συστηματων
Πληροφορικης

Υλοποίηση Πιθανοτικών Δομών Δεδομένων στο Επίπεδο Δεδομένων για την Αποδοτική Αντιμετώπιση Επιθέσεων DDoS στο Σύστημα DNS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αναστασία Γ. Δημακοπούλου

Επιβλέπων : Βασίλειος Μάγκλαρης
Ομότιμος Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Ιουλίου 2022.

……	……	……
Βασίλειος Μάγκλαρης	Συμεών Παπαβασιλείου	Νεκτάριος Κοζύρης
Ομότιμος Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2022

……
Αναστασία Δημακοπούλου

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

Ευχαριστίες
Θα ήθελα να ευχαριστήσω τον Ομότιμο Καθηγητή Ε.Μ.Π. Κ. Βασίλη Μάγκλαρη για την ευκαιρία που μου έδωσε να ασχοληθώ με το συγκεκριμένο θέμα. Ακόμα, θα ήθελα να δώσω θερμές ευχαριστίες στον Υποψήφιο Διδάκτορα Νίκο Κωστόπουλο για το χρόνο που αφιέρωσε σε αυτή την εργασία, τις χρήσιμες συμβουλές και την πολύτιμη καθοδήγηση του σε όλα τα στάδια εκπόνησής της. Τέλος, ευχαριστώ την οικογένεια μου και τους φίλους μου για την υποστήριξή τους κατά τη διάρκεια των σπουδών μου.

Κατάλογος σχημάτων:

Κατάλογος πινάκων:

Περίληψη

Το Σύστημα Ονοματοδοσίας Τομέων (DNS) είναι αναπόσπαστο κομμάτι της ομαλής λειτουργίας του διαδικτύου, καθώς μέσω αυτού αντιστοιχίζονται ονόματα υπολογιστών με διευθύνσεις IP και το αντίστροφο. Για αυτό γίνεται στόχος Κατανεμημένων Επιθέσεων Άρνησης Υπηρεσίας (DDoS), οι οποίες αποσκοπούν στη διακοπή της υπηρεσίας που παρέχει.

Μία από αυτές τις επιθέσεις είναι η DNS Water Torture επίθεση, κατά την οποία ο επιτιθέμενος πλημμυρίζει τον αρμόδιο (authoritative) για μια ζώνη DNS εξυπηρετητή με ερωτήματα για ονόματα που δεν υπάρχουν στη ζώνη. Σκοπός της επίθεσης είναι να εξαντληθεί η επεξεργαστική ισχύς τους εξυπηρετητή - θύμα, με αποτέλεσμα να μην μπορεί να απαντήσει σε ερωτήματα καλόβουλων χρηστών.

Ο μηχανισμός που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας στοχεύει στην αντιμετώπιση μιας τέτοιας επίθεσης, με χρήση πιθανοτικών δομών δεδομένων, και προγραμματισμό στο επίπεδο δεδομένων.

Συγκεκριμένα, γίνεται επεξεργασία των ερωτημάτων DNS στο επίπεδο XDP, το οποίο βρίσκεται στο χαμηλότερο επίπεδο της στοίβας δικτύου του Linux. Σε αυτό το επίπεδο δεν έχει γίνει κατανομή μνήμης από τον πυρήνα, και αυτό το καθιστά αποτελεσματικό στην απόρριψη μεγάλου όγκου πακέτων, όπως σε μία επίθεση Water Torture. Κατά την επεξεργασία των πακέτων ελέγχεται αν το όνομα προς επίλυση περιέχεται στα αρχεία ζώνης του εξυπηρετητή, αναζητώντας το όνομα στην εκάστοτε δομή δεδομένων που χρησιμοποιείται (Bloom Filter, Cuckoo Filter, Morton Filter), η οποία έχει δημιουργηθεί και φορτωθεί στο επίπεδο ελέγχου (control plane). Αν το όνομα βρεθεί, θεωρείται πως υπάρχει στα αρχεία ζώνης και συνεχίζει την πορεία του, αλλιώς απορρίπτεται. Ως αποτέλεσμα, ο μεγαλύτερος όγκος της κακόβουλης κίνησης απορρίπτεται, και οι καλόβουλοι χρήστες εξυπηρετούνται κανονικά.

Σκοπός της εργασίας είναι η αξιολόγηση της απόδοσης αυτού του μηχανισμού.

Λέξεις κλειδιά: Σύστημα Ονοματοδοσίας Τομέων, Κατανεμημένη Επίθεση Άρνησης Υπηρεσίας, επίθεση DNS Water Torture, πιθανοτικές δομές δεδομένων, φίλτρο Bloom, φίλτρο Cuckoo, φίλτρο Morton, extended Berkeley Packet Filter, eXpress Data Path, Προγραμματισμός στο Επίπεδο Δεδομένων

Abstract

The Domain Name System (DNS) is an integral part of the proper operation of the Internet, as, through it, host names are matched to IP addresses, and vice versa. This is why it is the target of Distributed Denial of Service attacks (DDoS), that aim to halt its service provision.

DNS Water Torture is such an attack, during which the attacker floods the authoritative server of a DNS zone with queries concerning names that do not exist in the zone. Their goal is to exhaust the processing power of the victim server, resulting in its inability to respond to queries of legitimate users.

The mechanism developed in the context of this thesis aims to mitigate such an attack, using probabilistic data structures, and data plane programming.

Specifically, DNS queries are processed at XDP level, which is located at the lowest level of the Linux network stack. At this point no memory allocation by the kernel has taken place, making it efficient to drop a large volume of packets, such as in a Water Torture attack. During the packet processing, the name to be resolved is checked against the data structure that is used (Bloom Filter, Cuckoo Filter, Morton Filter), which has been created and loaded at the control plane. Should the lookup be successful, the packet passes, otherwise it is dropped. As a result, most of the malicious traffic is dropped, and legitimate users are served as normal.

The purpose of this work is to evaluate the performance of this mechanism.

Keywords: Domain Name System (DNS), Distributed Denial of Service (DDoS), DNS Water Torture Attack, probabilistic data structures, Bloom Filter, Cuckoo Filter, Morton Filter, extended Berkeley Packet Filter (eBPF), eXpress Data Path (XDP), Data Plane Programming

Περιεχόμενα

Κεφάλαιο 1: Εισαγωγή

1.1 Περιγραφή του προβλήματος

Το διαδίκτυο είναι, πλέον, βασικός πυλώνας της καθημερινότητας των ανθρώπων. Χρησιμοποιείται για εργασία, ψυχαγωγία, δημόσιες υπηρεσίες και πολλές άλλες εφαρμογές.

Οι υπολογιστές στο διαδίκτυο επικοινωνούν με διευθύνσεις IP. Αυτές οι διευθύνσεις μπορεί να είναι IPv4, άρα της μορφής x.x.x.x, όπου x ένας αριθμός 8 bit, ή μπορεί να είναι IPv6, δηλαδή της μορφής x.x.x.x.x.x.x.x., όπου x ένας αριθμός 16 bit. Είναι, λοιπόν, δύσχρηστο οι χρήστες να τις θυμούνται και να τις χρησιμοποιούν. Για αυτό το λόγο οι χρήστες, αλλά και υπολογιστικά συστήματα, χρησιμοποιούν ονόματα για να προσπελάσουν πόρους στο διαδίκτυο. Το σύστημα που αντιστοιχίζει τα ονόματα των πόρων στο διαδίκτυο με τις διευθύνσεις IP είναι το Σύστημα Ονοματοδοσίας Τομέων (Domain Name System - DNS). Πρόκειται ουσιαστικά για ένα κατανεμημένο σύστημα το οποίο αντιστοιχεί ονόματα με υπολογιστικούς πόρους, όπως διευθύνσεις IP. Είναι σημαντικό οι εξυπηρετητές DNS να είναι διαθέσιμοι να απαντούν ερωτήματα συνεχώς, για την εύρυθμη λειτουργία του διαδικτύου.

Το DNS είναι συχνά στόχος επιθέσεων, με πολλές από αυτές να είναι επιθέσεις άρνησης υπηρεσίας (Denial of Service - DoS). Στόχος αυτών των επιθέσεων είναι η υπηρεσία που παρέχει το DNS να σταματήσει να είναι προσβάσιμη από καλόβουλους χρήστες.

Μία από αυτές της επιθέσεις είναι η επίθεση DNS Water Torture. Ο επιτιθέμενος πλημμυρίζει τον, αρμόδιο(authoritative) για μια ζώνη DNS, εξυπηρετητή - θύμα με ερωτήματα για ονόματα που δε βρίσκονται στα αρχεία του. Τα ονόματα αυτά είναι τυχαία και μη επαναλαμβανόμενα για να παρακάμπτουν την προσωρινή μνήμη ενδιάμεσων εξυπηρετητών και να φτάνουν στο θύμα. Το αποτέλεσμα της επίθεσης είναι η εξάντληση των υπολογιστικών πόρων του θύματος, και η τελική αδυναμία του εξυπηρετητή να απαντά σε έγκυρα ερωτήματα.

1.2 Σκοπός της εργασίας

Στόχος της εργασίας είναι η μελέτη και αντιμετώπιση (mitigation) μιας τέτοιας επίθεσης, της επίθεσης Water Torture. Κατά τη διάρκεια αυτής της επίθεσης, φτάνει μεγάλος όγκος ερωτημάτων για ονόματα που δεν υπάρχουν,με σκοπό να εξαντλήσουν την υπολογιστική ισχύ του εξυπηρετητή DNS. Ο μηχανισμός αντιμετώπισης της επίθεσης θα πρέπει να ξεχωρίζει αποδοτικά τα καλόβουλα από τα κακόβουλα ερωτήματα, και να απορρίπτει τα τελευταία.

Ο μηχανισμός που αναπτύχθηκε σε αυτή τη διπλωματική εργασία χρησιμοποιεί πιθανοτικές δομές δεδομένων, υλοποιημένες στο περιβάλλον εκτέλεσης eBPF. Αυτές οι δομές αποθηκεύουν τα στοιχεία τους σε hashed μορφή, και αυτό τις κάνεις πιο αποδοτικές στην κατανάλωση μνήμης, αλλά και στο χρόνο αναζήτησης ενός στοιχείου σε αυτές, σε σχέση με κάποια ντετερμινιστική δομή δεδομένων, όπως ένας πίνακας ή ένα δέντρο. Σε αυτές τις δομές βρίσκονται τα ονόματα για τα οποία ο εξυπηρετητής είναι αρμόδιος να απαντά σε

ερωτήματα. Για κάθε εισερχόμενο πακέτο, γίνεται έλεγχος του ονόματος του ερωτήματος που περιέχει, και αν το όνομα βρεθεί στη δομή, τότε το πακέτο προχωρά προς επίλυση, αλλιώς απορρίπτεται.

Οι δομές υλοποιήθηκαν στο περιβάλλον εκτέλεσης eBPF, και ο αμυντικός μηχανισμός προσαρτάται στο eXpress Data Path (XDP) hook. Το XDP είναι μια τεχνολογία Data Plane η οποία βρίσκεται στο χαμηλότερο στρώμα της στοίβας δικτύου του Linux, και επιτρέπει την ταχύτερη ανάλυση και απόρριψη πακέτων, καθώς δεν έχει γίνει εκχώρηση μνήμης από τον πυρήνα. Για αυτό το λόγο παρέχει πολύ υψηλές αποδόσεις και είναι κατάλληλη για την αντιμετώπιση επιθέσεων DDoS.

1.3 Δομή εργασίας

Στο κεφάλαιο 1 γίνεται μια εισαγωγή στη θεωρία και το σκοπό της εργασίας. Στο κεφάλαιο 2 έχουμε το θεωρητικό υπόβαθρο που είναι απαραίτητο για την κατανόηση της λειτουργίας του μηχανισμού που υλοποιήθηκε. Στο κεφάλαιο 3 περιγράφεται η πειραματική διάταξη και τα εργαλεία, ενώ στο κεφάλαιο 4 ακολουθούν τα αποτελέσματα και η αξιολόγησή τους. Τέλος στο κεφάλαιο 5 περιλαμβάνει τα συμπεράσματα της εργασίας καθώς και μελλοντικές επεκτάσεις.

Κεφάλαιο 2: Θεωρητικό Υπόβαθρο

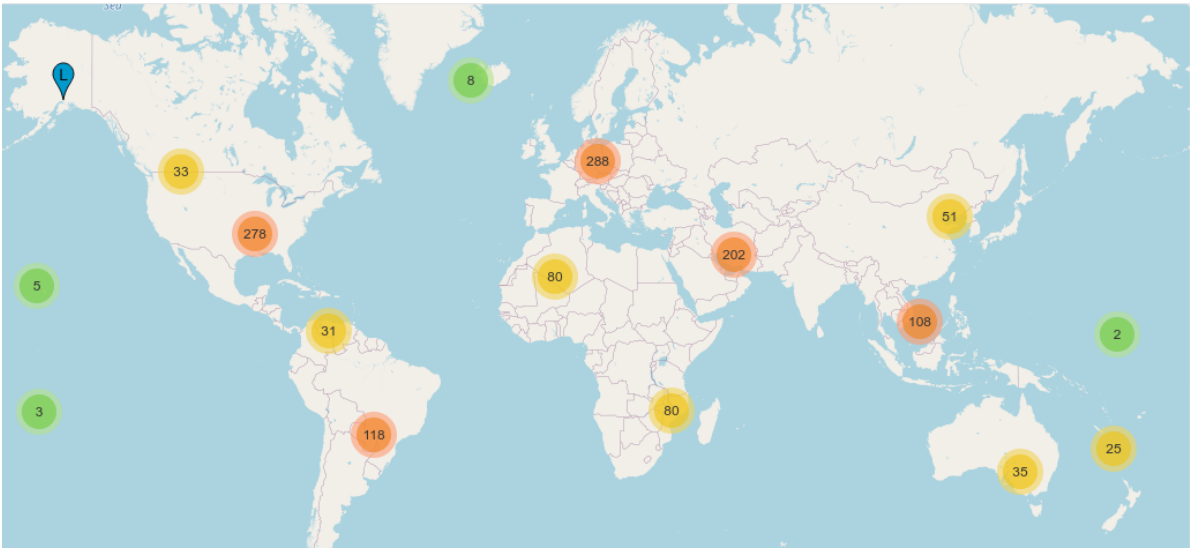
Σε αυτό το κεφάλαιο αναλύεται το θεωρητικό υπόβαθρο πάνω στο οποίο βασίζεται η παρούσα διπλωματική εργασία.

2.1 Domain Name System

Το Σύστημα Ονοματοδοσίας Τομέων (Domain Name System - DNS) αντιστοιχεί πλήρη ονόματα τομέων(Fully Qualified Domain Names - FQDNs) με υπολογιστικούς πόρους.. Είναι ιεραρχικό και αποκεντροποιημένο στη δομή του. Με τον όρο DNS ορίζουμε αρχικά την κατανεμημένη βάση δεδομένων, που έχει υλοποιηθεί σε μια ιεραρχία εξυπηρετητών DNS, και έπειτα το πρωτόκολλο επιπέδου εφαρμογής που επιτρέπει στους χρήστες να υποβάλουν ερωτήματα σε αυτή τη βάση[1, p. 131].

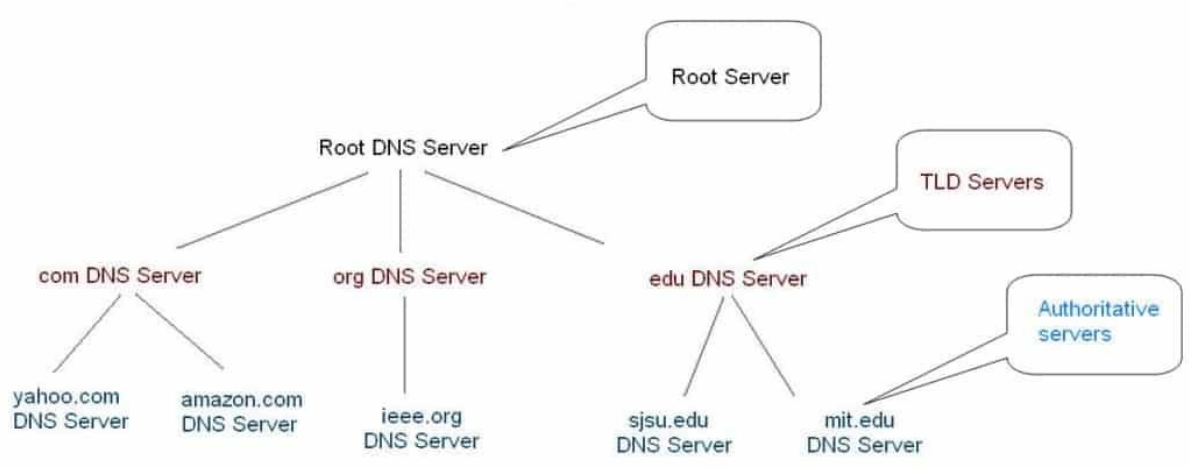
2.1.1 Δομή

Πιο ψηλά στην ιεραρχία βρίσκονται οι ριζικοί εξυπηρετητές (root nameservers) και είναι υπεύθυνοι για το ριζικό τομέα, ο οποίος συμβολίζεται με μια τελεία. Έχουν ονόματα της μορφής letter.root-servers.net, όπου “letter” είναι κάποιο από τα γράμματα a-m. Αν και στα αρχικά στάδια του DNS, κάθε root nameserver βρισκόταν σε ένα σημείο, στο σημερινό DNS κάθε root nameserver αντιστοιχεί σε ένα σύνολο εξυπηρετητών που απαντά στην ίδια διεύθυνση IP, για καταμερισμό του φόρτου αλλά και ανοχή σε σφάλματα και επιθέσεις.



Σχήμα 2.1: Οι τοποθεσίες των root nameservers, 2022[2]

Στις εγγραφές τους περιέχονται οι διευθύνσεις των εξυπηρετητών τομέων ανώτατου επιπέδου (Top Level Domains - TLDs). Μερικοί από τους πιο γνωστούς TLDs είναι οι .com, .net, .org, .edu καθώς και οι τομείς κάθε χώρας, για παράδειγμα .gr (Ελλάδα), .uk (Ηνωμένο Βασίλειο), .fr (Γαλλία), .ru (Ρωσία).
Οι TLD εξυπηρετητές, με τη σειρά τους, έχουν στις εγγραφές τους τις διευθύνσεις των τομέων δεύτερου επιπέδου (Second Level Domains).



Σχήμα 2.2: Ιεραρχία των εξυπηρετητών DNS [3]

Η περιοχή για την οποία ένας εξυπηρετητής είναι αρμόδιος να απαντά σε ερωτήματα ονομάζεται ζώνη, και ο εξυπηρετητής ονομάζεται authoritative nameserver.

Οι DNS resolvers είναι εξυπηρετητές που κάνουν ερωτήματα DNS στους Authoritative nameservers, και συνήθως αποθηκεύουν προσωρινά τις απαντήσεις (caching), ώστε ονόματα που ζητούνται συχνά να είναι άμεσα διαθέσιμα.

Οι απαντήσεις που αποθηκεύονται στην προσωρινή μνήμη των Resolvers βρίσκονται εκεί για ορισμένο χρονικό διάστημα (Time To Live - TTL), μετά την πάροδο του οποίου διαγράφονται από τη μνήμη, και η αντίστοιχη πληροφορία πρέπει να αναζητηθεί εκ νέου από τους αρμόδιους εξυπηρετητές. Αυτό συμβαίνει ώστε οι πληροφορίες που αποθηκεύονται στην προσωρινή μνήμη να είναι έγκυρες, για παράδειγμα, στην περίπτωση που αλλάξει η διεύθυνση IP ή το όνομα ενός υπολογιστικού πόρου.

2.1.2 Fully Qualified Domain Name (FQDN)

Το Πλήρες Όνομα Τομέα (FQDN) εντοπίζει ακριβώς τη θέση ενός διαδικτυακού πόρου στην ιεραρχία DNS. Διαβάζεται από τα δεξιά προς τα αριστερά, προχωρώντας από το πιο ψηλό επίπεδο της ιεραρχίας στο χαμηλότερο. Κάθε επίπεδο διαχωρίζεται με μία τελεία (.), και η τελεία στη πιο δεξιά θέση συμβολίζει τη ρίζα του δέντρου DNS.

Σχήμα 2.3: Σχηματισμός του FQDN του www.example.com

2.1.3 Εγγραφές Πόρων

Κάθε authoritative εξυπηρετητής διατηρεί εγγραφές πόρων (Resource Records - RRs) για τους οποίους είναι υπεύθυνος. Η εγγραφή πόρου έχει την εξής μορφή[4]:

- NAME: Το όνομα του πόρου στον οποίο αναφέρεται η εγγραφή
- TYPE: Ο τύπος της εγγραφής. Μερικοί από τους τύπους είναι:
    - A: Η εγγραφή αντιστοιχίζει ένα όνομα σε μια διεύθυνση IPv4 (32 bits)
    - AAAA: Η εγγραφή αντιστοιχίζει ένα όνομα σε μια διεύθυνση IPv6 (128 bits)
    - NS: Η εγγραφή προσδιορίζει τον αρμόδιο (authoritative) εξυπηρετητή για μία ζώνη
    - MX: Η εγγραφή προσδιορίζει τους εξυπηρετητές ηλεκτρονικού ταχυδρομείου (mail servers) για μία ζώνη
    - PTR: Η εγγραφή αντιστοιχίζει μια διεύθυνση (IPv4/IPv6) σε ένα όνομα
    - CNAME: Η εγγραφή ορίζει το κανονικό (canonical) όνομα για ένα ψευδώνυμο (alias)
    - SOA: Η εγγραφή περιλαμβάνει πληροφορίες για τη ζώνη, όπως ποιος είναι ο διαχειριστής, ποιος είναι ο πρωτεύων εξυπηρετητής, το προκαθορισμένο TTL των εγγραφών κ.ά.
- CLASS: Η κλάση της εγγραφής. Συνήθως έχει την τιμή IN (Internet).
- TTL: Το TTL (Time-to-Live) είναι ο χρόνος ζωής της εγγραφής, δηλαδή ορίζει για πόσο χρόνο μια εγγραφή παραμένει στην προσωρινή μνήμη ενός Resolver.
- RDLENGTH: Προσδιορίζει το μέγεθος του πεδίου RDATA σε bytes.
- RDATA: Περιέχει την τιμή της εγγραφής, και έχει διαφορετική μορφή ανάλογα με τον τύπο. Για παράδειγμα, για τον τύπο NS, περιέχει το όνομα του αρμόδιου εξυπηρετητή για τη ζώνη (η οποία ορίζεται στο πεδίο NAME).

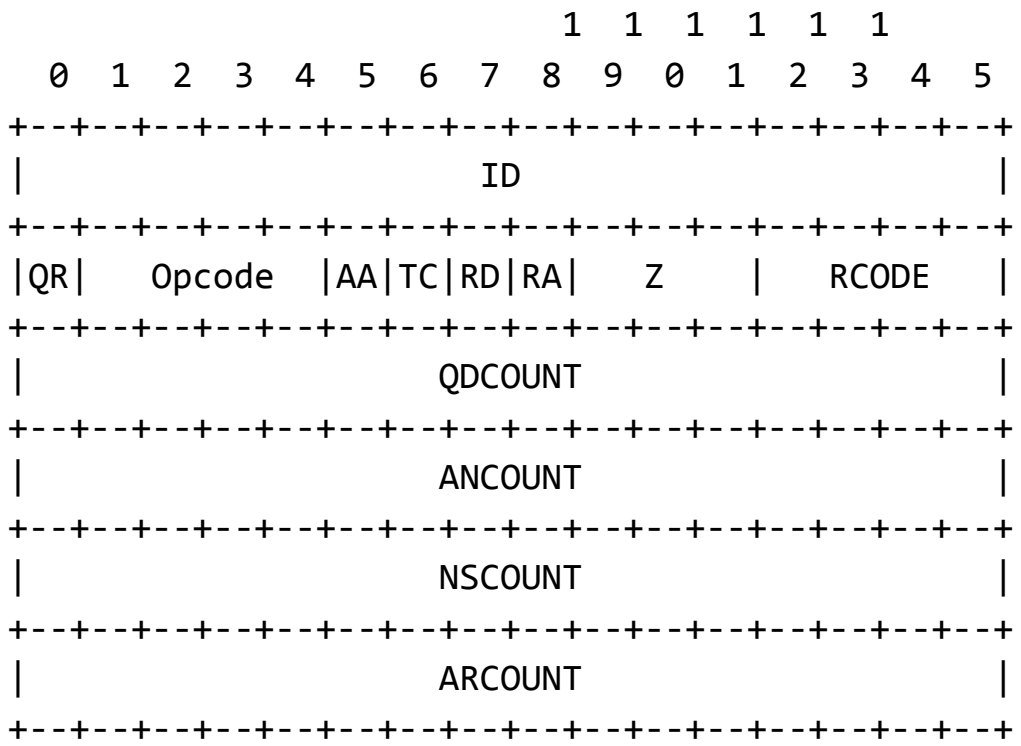
2.1.4 Πακέτα DNS

Τα πακέτα στο πρωτόκολλο DNS μπορεί να είναι είτε ερωτήσεις (queries) είτε απαντήσεις (replies). Χρησιμοποιούν την ίδια δομή[4], με τα πεδία Question, Answer, Authority και Additional να έχουν μεταβλητό μήκος:

Πεδίο	**Περιγραφή**
Header	Επικεφαλίδα
Question	Η ερώτηση για τον εξυπηρετητή
Answer	Εγγραφές που απαντούν στην ερώτηση
Authority	Εγγραφές που δείχνουν προς authoritative nameservers
Additional	Εγγραφές με πρόσθετες πληροφορίες

Πίνακας 2.1: Δομή ενός πακέτου DNS

Η επικεφαλίδα DNS και στις δύο περιπτώσεις είναι κοινή:

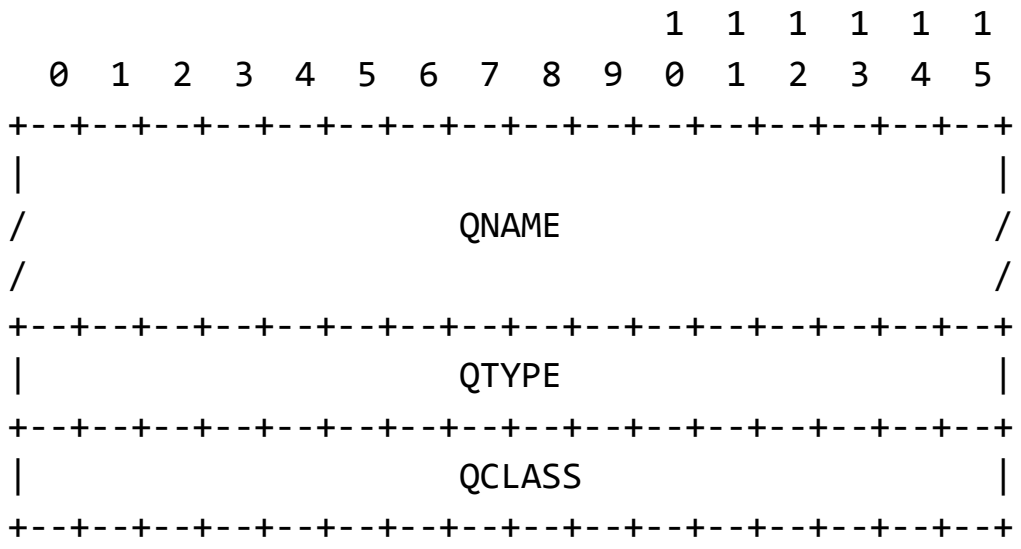


όπου:

- ID: ένα αναγνωριστικό 16 bit ώστε να αντιστοιχίζονται τα ερωτήματα με τις απαντήσεις τους
- QR (Query/ Response): 1 bit που προσδιορίζει αν πρόκειται για ερώτηση(0) ή απάντηση(1).
- OPCODE: 4 bit που προσδιορίζουν το είδος της ερώτησης. Για τυπικές ερωτήσεις έχει την τιμή 0.
- AA (Authoritative Answer): 1 bit που προσδιορίζει αν ο εξυπηρετητής που απαντά είναι αρμόδιος για τον τομέα ονόματος που ζητείται.
- TC (TrunCation): 1 bit που προσδιορίζει αν το μήνυμα έχει κοπεί λόγω μεγάλου μεγέθους.
- RD (Recursion Desired): 1 bit που προσδιορίζει αν ζητείται το ερώτημα να επιλυθεί αναδρομικά.
- RA (Recursion Available): 1 bit που προσδιορίζει αν η αναδρομική επίλυση υποστηρίζεται από τον εξυπηρετητή.
- Z: Δεσμευμένο bit για μελλοντική χρήση. Έχει την τιμή 0.
- RCODE (Response Code): 4 bit που παίρνουν τις εξής τιμές:
    - 0: Κανένα σφάλμα
    - 1: Σφάλμα μορφής ερωτήματος
    - 2: Σφάλμα εξυπηρετητή
    - 3: Σφάλμα ονόματος
    - 4: Σφάλμα υλοποίησης: Ο εξυπηρετητής δεν υποστηρίζει το συγκεκριμένο

τύπο ερωτήματος.
- 5: Άρνηση απάντησης: Ο εξυπηρετητής δεν εκτελεί την συγκεκριμένη λειτουργία για λόγους πολιτικής (policy).
- 6-15: Δεσμευμένα για μελλοντική χρήση
- QDCOUNT: Ο αριθμός των ερωτημάτων στο τμήμα ερωτήσεων.
- ANCOUNT: Ο αριθμός των εγγραφών στο τμήμα απαντήσεων.
- NSCOUNT: Ο αριθμός των εγγραφών NS στο τμήμα αρχών (authority).
- ARCOUNT: Ο αριθμός των εγγραφών στο τμήμα πρόσθετων στοιχείων.

Το τμήμα απαντήσεων έχει την εξής δομή:



όπου:

QNAME: Όνομα που απεικονίζεται ως μια ακολουθία από ετικέτες (labels). Κάθε ετικέτα αποτελείται από έναν αριθμό και μία λέξη με μήκος σε χαρακτήρες όσο αυτός ο αριθμός. Το όνομα έχει σαν τελικό χαρακτήρα το μηδενικό χαρακτήρα.

QTYPE: Ο τύπος (π.χ. A, MX) του ερωτήματος. Κάποιοι τύποι ερωτημάτων μπορούν να απαντηθούν με μία εγγραφή, όπως ο τύπος A, όπου ζητείται η διεύθυνση IPv4 ενός υπολογιστικού πόρου. Σε άλλους τύπους ερωτημάτων, όπως ο τύπος AXFR, ζητούνται περισσότερες εγγραφές. Συγκεκριμένα, οι authoritative εξυπηρετητές μιας ζώνης DNS κάνουν ερωτήματα AXFR μεταξύ τους για να επικαιροποιήσουν τα αρχεία ζώνης.

QCLASS: Η κλάση (π.χ. IN) του ερωτήματος.

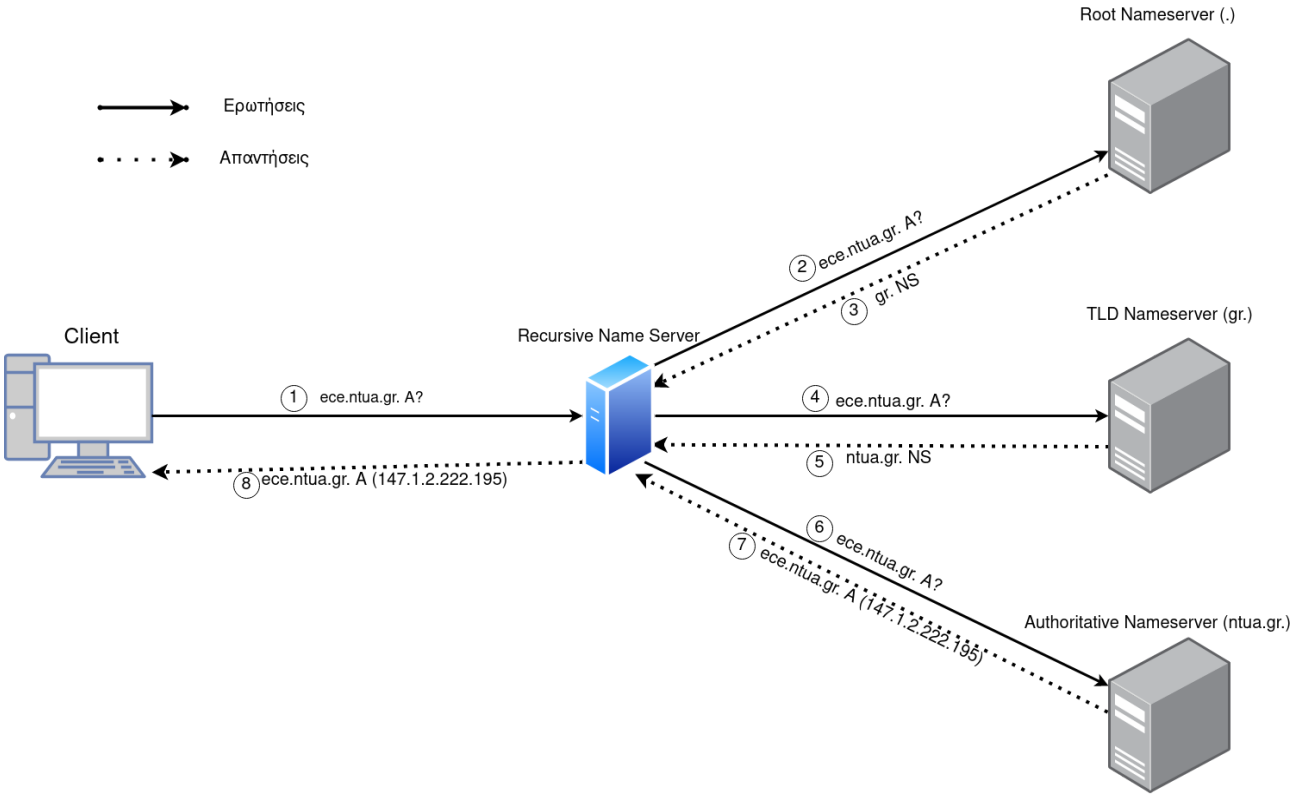
2.1.5 Λειτουργία

Κάθε φορά που ένας χρήστης ζητά τη διεύθυνση ενός ονόματος, ακολουθείται η παρακάτω διαδικασία:

1. Ο χρήστης επικοινωνεί με τον προκαθορισμένο αναδρομικό DNS εξυπηρετητή και ζητά τη διεύθυνση του ονόματος.
2. Αν ο εξυπηρετητής έχει την αντίστοιχη εγγραφή στην προσωρινή μνήμη, απαντά στο χρήστη με τη διεύθυνση που ζήτησε. Αν δεν την έχει, στέλνει ερώτημα στον (κοντινότερο γεωγραφικά) ριζικό εξυπηρετητή.
3. Ο ριζικός εξυπηρετητής απαντά με τη διεύθυνση του κατάλληλου TLD εξυπηρετητή. Ο αρχικός εξυπηρετητής στέλνει ερώτημα σε αυτόν.
4. Ο TLD εξυπηρετητής απαντά με τη διεύθυνση του κατάλληλου αρμόδιου (authoritative) για τον τομέα εξυπηρετητή. Ο αρχικός εξυπηρετητής απευθύνεται σε αυτόν για τη διεύθυνση του ονόματος.
5. Ο authoritative εξυπηρετητής απαντά στον αρχικό με τη διεύθυνση του ονόματος που ζητήθηκε.
6. Ο αρχικός εξυπηρετητής δίνει την τελική απάντηση στο χρήστη, και, προαιρετικά, αποθηκεύει την απάντηση στην προσωρινή του μνήμη.

Παράδειγμα ενός ερωτήματος DNS για το όνομα ece.ntua.gr:

1. Ο χρήστης ζητά τη διεύθυνση IP για το όνομα ece.ntua.gr από τον προκαθορισμένο (default) DNS Resolver (συνήθως καθορίζεται από τον Internet Service Provider - ISP).
2. Αν ο Resolver δεν έχει τη διεύθυνση του ζητούμενου ονόματος στην cache, απευθύνεται στον (συνήθως πιο κοντινό γεωγραφικά) ριζικό εξυπηρετητή.
3. Ο ριζικός εξυπηρετητής απαντά με τη διεύθυνση του TLD εξυπηρετητή.
4. Ο Resolver ρωτά τον TLD εξυπηρετητή.
5. Ο TLD εξυπηρετητής απαντά με τη διεύθυνση του αρμόδιου(authoritative) εξυπηρετητή για το όνομα ece.ntua.gr.
6. Στη συνέχεια ο Resolver στέλνει ερώτηση για τον αρμόδιο εξυπηρετητή για τον τομέα ntua.gr.
7. Ο τελευταίος απαντά με τη διεύθυνση του ονόματος ece.ntua.gr.
8. Ο Resolver προωθεί την απάντηση στο χρήστη.



Σχήμα 2.4: Παράδειγμα επίλυσης του ονόματος ece.ntua.gr

2.1.6 Ασφάλεια στο DNS

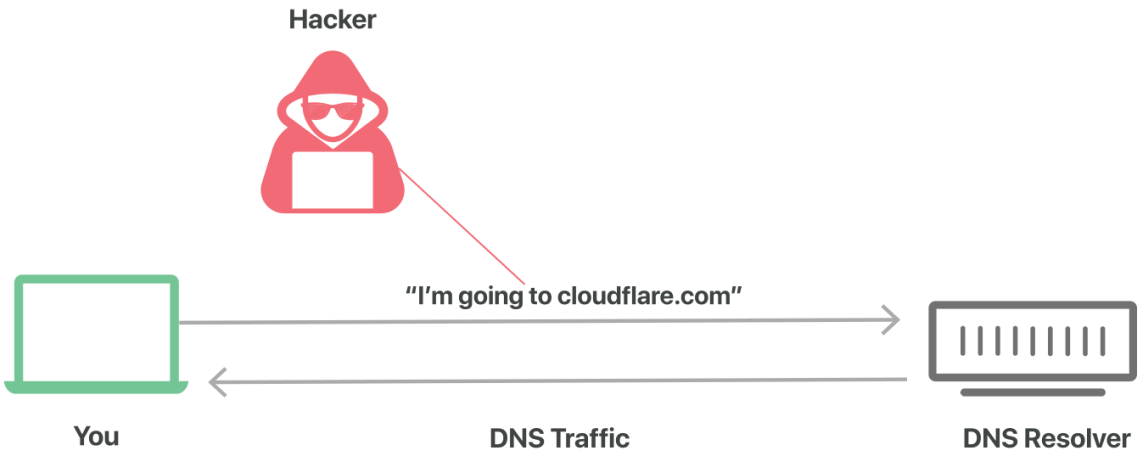
Είναι φανερό ότι η εύρυθμη λειτουργία του DNS είναι απαραίτητη για το σύνολο των υπηρεσιών στο Διαδίκτυο. Οι περισσότερες εφαρμογές που χρησιμοποιούν το διαδίκτυο, χρησιμοποιούν και το DNS. Το πρωτόκολλο DNS, όμως, δε σχεδιάστηκε με γνώμονα την ασφάλεια και την ιδιωτικότητα, αλλά την ευρωστία και την ανοχή σε σφάλματα. Οι πληροφορίες που ανταλλάσσονται όταν γίνεται ένα ερώτημα DNS δεν είναι κρυπτογραφημένες , και όσοι έχουν πρόσβαση στα ερωτήματα, έχουν πρόσβαση και σε επιπλέον πληροφορίες όπως: πότε έγινε το ερώτημα, ποιος το έκανε, ποιο όνομα ζητήθηκε, κλπ[5]. Αυτή την αδυναμία μπορούν να την εκμεταλλευτούν προγράμματα που επεξεργάζονται ερωτήματα DNS σε DNS Resolvers, τα οποία συλλέγουν αυτά τα δεδομένα για π.χ. διαφημιστικούς ή εμπορικούς σκοπούς.

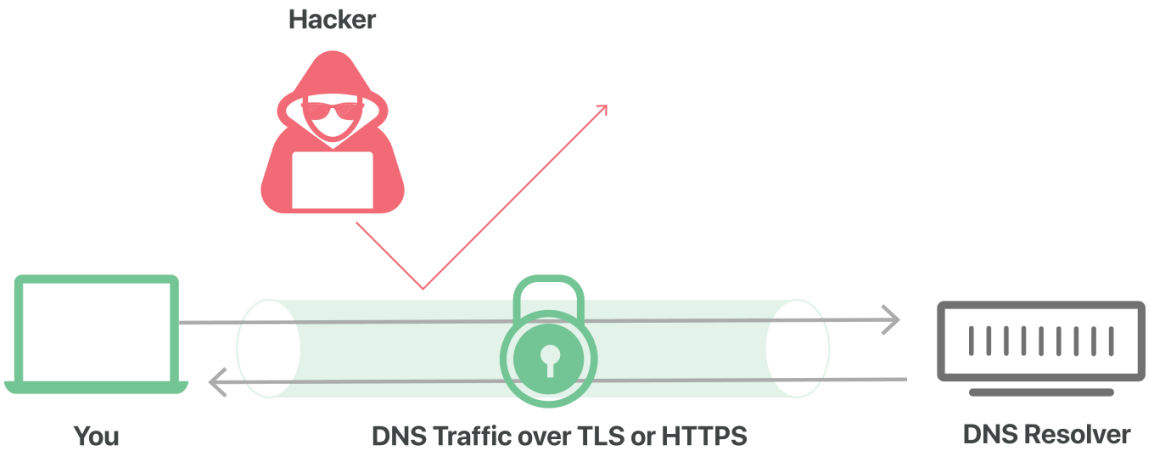
Ένα άλλο προβληματικό σημείο είναι το ότι, επειδή και τα ερωτήματα και οι απαντήσεις σε αυτά δεν κρυπτογραφούνται, δεν μπορεί να ελεγχθεί η αυθεντικότητα αυτών, δηλαδή να πιστοποιηθεί ποιος τα έστειλε. Στην περίπτωση των ερωτημάτων αυτό δεν είναι απαραίτητο, αλλά στην περίπτωση των απαντήσεων, αυτός που έκανε το ερώτημα μπορεί να λάβει απάντηση, όχι από τον εξυπηρετητή στον οποίο έκανε το ερώτημα, αλλά από κακόβουλο χρήστη που παριστάνει τον εξυπηρετητή (επίθεση Man-in-the-Middle)[6, p. 301].

Αυτές οι αδυναμίες αντιμετωπίζονται από ορισμένους μηχανισμούς στο DNS. Ένας από αυτούς τους μηχανισμούς είναι το DNSSEC (DNS SECurity extensions)[7]. Είναι μία επέκταση στο DNS, η οποία χρησιμοποιεί ψηφιακές υπογραφές βασισμένες στην κρυπτογραφία δημοσίου κλειδιού. Κάθε ζώνη DNS έχει ένα ζευγάρι ιδιωτικού/δημόσιου κλειδιού. Κατά την επίλυση ενός ονόματος, η απάντηση υπογράφεται ψηφιακά με το

ιδιωτικό κλειδί της ζώνης, το οποίο είναι γνωστό μόνο στον αρμόδιο εξυπηρετητή, και η υπογραφή ελέγχεται από τον Resolver με το δημόσιο κλειδί, το οποίο είναι διαθέσιμο σε όλους. Αυτό εγγυάται ότι η απάντηση προέρχεται από τον σωστό αρμόδιο εξυπηρετητή, και ότι τα δεδομένα της απάντησης δεν έχουν τροποποιηθεί κατά τη μεταφορά τους.

Ένας άλλος μηχανισμός που αντιμετωπίζει την ελεύθερη πρόσβαση στα δεδομένα των ερωτήσεων DNS είναι το DNS over TLS[8] (DoT) και το DNS over HTTPS[9] (DoH). Χρησιμοποιούν το πρωτόκολλο TLS για να κρυπτογραφήσουν το ερώτημα DNS, και, στην περίπτωση του DoH, το ερώτημα αποστέλλεται με το πρωτόκολλο HTTPS αντί απευθείας πάνω από UDP, και δεν μπορεί να διαχωριστεί από την υπόλοιπη HTTPS κίνηση, καθώς χρησιμοποιεί τη θύρα 443. Στο DoT χρησιμοποιείται η θύρα 853.





Σχήμα 2.5: Ένας κακόβουλος χρήστης προσπαθεί να υποκλέψει πληροφορίες από την κίνηση DNS. Στο DoT/DoH η κίνηση είναι κρυπτογραφημένη.

2.1.7 Επιθέσεις στο DNS

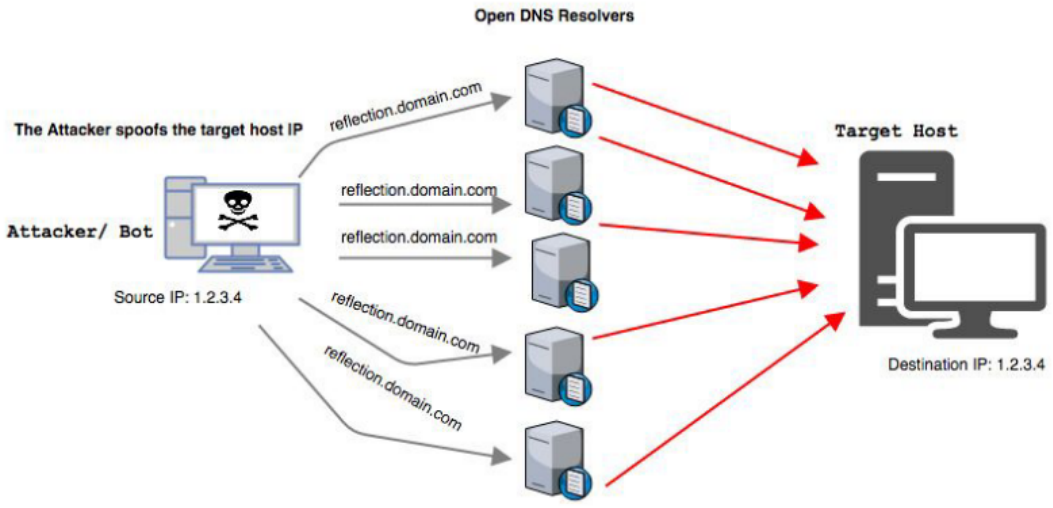
Σε αυτή την ενότητα παρουσιάζονται οι πιο σημαντικές επιθέσεις που στοχεύουν το DNS, ή το χρησιμοποιούν ως μέσο για επιθέσεις σε άλλους δικτυακούς πόρους.

Reflection / Amplification

Κατά τη διάρκεια μιας επίθεσης Reflection[10], ο κακόβουλος χρήστης στέλνει πολύ μεγάλο όγκο μηνυμάτων στο θύμα εκμεταλλευόμενος τους Open Resolvers. Το θύμα δεν είναι αναγκαστικά κάποιος DNS εξυπηρετητής, αλλά οποιοσδήποτε υπολογιστής στο διαδίκτυο.

Οι Open Resolvers χρησιμοποιούνται για να κατευθύνουν την κίνηση DNS προς το θύμα. Ο κακόβουλος χρήστης στέλνει ερωτήματα DNS στους Resolvers, οι οποίοι χρησιμοποιούνται ως ανακλαστήρες (reflectors), εξ ου και reflection (ανάκλαση), αντικαθιστώντας τη διεύθυνση IP από την οποία προέρχονται τα ερωτήματα με τη διεύθυνση του θύματος. Η τεχνική της παραποίησης της διεύθυνσης IP ονομάζεται IP spoofing[11], και χρησιμοποιείται συχνά σε επιθέσεις. Έτσι, οι απαντήσεις DNS θα φτάσουν στο θύμα και όχι στον αρχικό αποστολέα.

Αυτό που κάνει την επίθεση αποτελεσματική, είναι το γεγονός ότι μικρά ερωτήματα DNS μπορούν να προκαλέσουν μεγάλες απαντήσεις (amplification). Τέτοιο ερώτημα είναι, για παράδειγμα, το αίτημα ANY, το οποίο έχει σαν απάντηση όλους τους τύπους των εγγραφών.



Σχήμα 2.6: Παράδειγμα επίθεσης DNS reflection[12]

Cache poisoning

Σε αυτή την επίθεση, ο επιτιθέμενος εκμεταλλεύεται τη διαδικασία επίλυσης ενός ονόματος που δεν υπάρχει στη προσωρινή μνήμη ενός Resolver[13]. Ζητά από τον Resolver τη διεύθυνση αυτού του ονόματος, και ο Resolver ακολουθεί τη διαδικασία που περιγράφηκε παραπάνω, μέχρι να πάρει την απάντηση του authoritative εξυπηρετητή. Στο μεταξύ, ο

επιτιθέμενος στέλνει κατάλληλα τροποποιημένες και μορφοποιημένες απαντήσεις DNS για αυτό το όνομα στον Resolver, υποδυόμενος τον authoritative εξυπηρετητή, ώστε στη μνήμη να αποθηκευτεί η διεύθυνση που θέλει ο επιτιθέμενος, και όχι η πραγματική. Έτσι, σε μελλοντικά ερωτήματα για αυτό το όνομα, ο Resolver θα απαντά με τη διεύθυνση που του έδωσε ο επιτιθέμενος, ανακατευθύνοντας, ενδεχομένως, τους χρήστες σε κακόβουλες ιστοσελίδες, μέχρι να λήξει η συγκεκριμένη εγγραφή στην προσωρινή μνήμη.

Αυτή η επίθεση αντιμετωπίζεται με την επέκταση DNSSEC, καθώς έτσι πιστοποιείται η ταυτότητα του εξυπηρετητή που απαντά στα ερωτήματα.

Παρά την αυθεντικοποίηση των απαντήσεων DNS που παρέχει η επέκταση DNS, ακόμα δεν έχει υιοθετηθεί ευρέως[14].

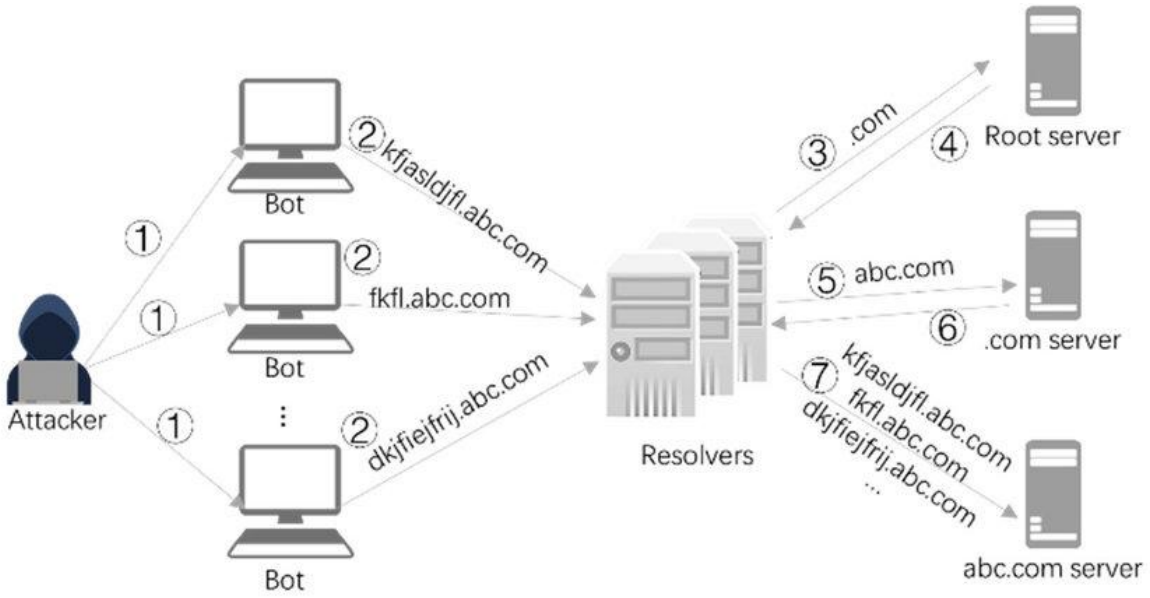
DNS Flood

Σε αυτή την επίθεση, ο επιτιθέμενος στέλνει πολύ μεγάλο όγκο αιτημάτων DNS, όχι απαραίτητα έγκυρων, προς τον authoritative εξυπηρετητή - θύμα, με σκοπό να εξαντλήσει το εύρος ζώνης του δικτύου του εξυπηρετητή και να μονοπωλήσει τους υπολογιστικούς πόρους του. Συνέπεια αυτού είναι ο εξυπηρετητής να μην μπορεί να απαντήσει σε ερωτήματα νόμιμων χρηστών.

Water Torture

Είναι μια παραλλαγή της επίθεσης DNS Flood[15]. Διαφοροποιείται στο ότι τα ερωτήματα που στέλνονται αφορούν τη ζώνη για την οποία είναι αρμόδιος ο εξυπηρετητής θύμα, αλλά τα ονόματα των ερωτημάτων δεν υπάρχουν στα αρχεία της ζώνης. Ο εξυπηρετητής σπαταλά πόρους για να απαντήσει σε αυτά τα ερωτήματα, και, τελικά, αδυνατεί να απαντήσει σε νόμιμα ερωτήματα.

Ακόμα, ο επιτιθέμενος δε χρειάζεται να ξέρει τη διεύθυνση του εξυπηρετητή - θύμα, καθώς μπορεί να αποστείλει τα άκυρα ερωτήματα προς αυτόν μέσω των Open Resolvers. Κατασκευάζοντας τα άκυρα ονόματα με τρόπο που δε θα υπάρχουν στην προσωρινή μνήμη των Resolvers, αυτοί θα ακολουθήσουν τη διαδικασία επίλυσης για κάθε όνομα. Απόρροια αυτών των ενεργειών είναι τα ονόματα που προϋπήρχαν στις προσωρινές μνήμες των Resolvers να αντικαθίστανται από τα άκυρα ονόματα, δεσμεύοντας χώρο από τις προσωρινές μνήμες.

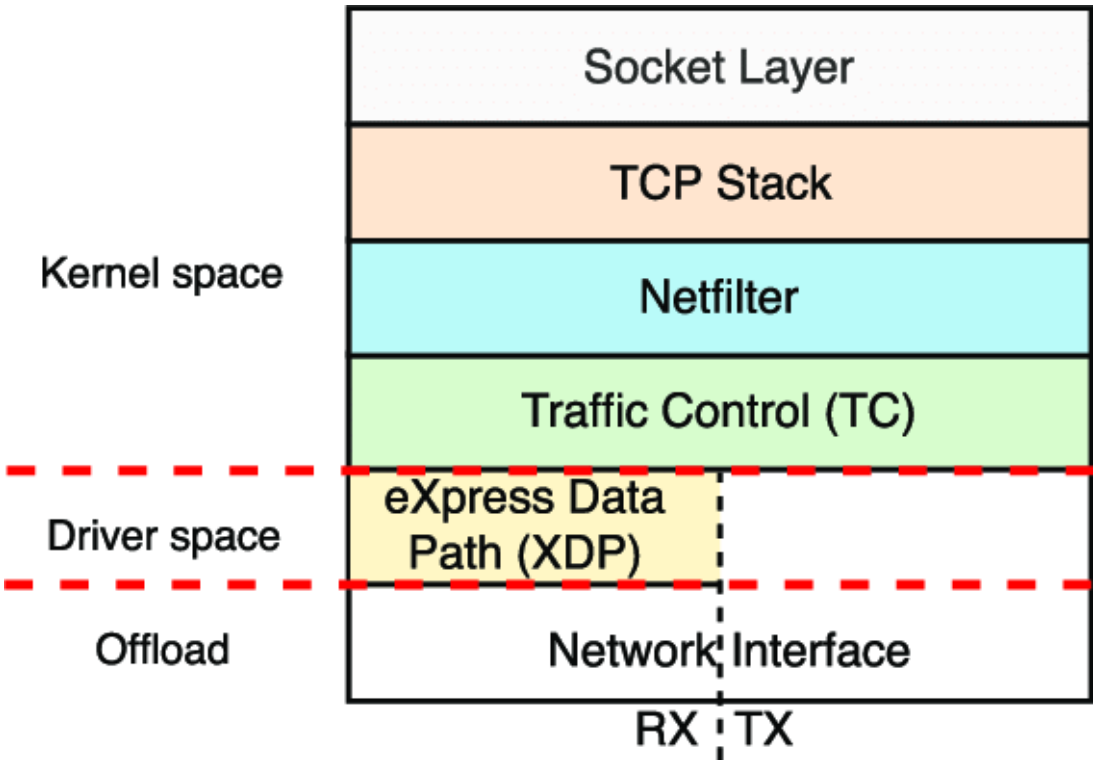


Σχήμα 2.7: Παράδειγμα επίθεσης Water Torture στον αρμόδιο εξυπηρετητή για τον τομέα abc.com [16, Fig. 4]

Παρενέργεια αυτής της επίθεσης είναι η υποβάθμιση ή/και άρνηση υπηρεσίας και στον ενδιάμεσο DNS Resolver.

2.2 Linux Network Stack

Η στοίβα δικτύου Linux είναι το σύνολο των εργαλείων και των προγραμμάτων του πυρήνα που διαχειρίζεται τα εισερχόμενα και εξερχόμενα πακέτα, από το φυσικό επίπεδο (physical layer) μέχρι και το επίπεδο εφαρμογής (layer application).



Σχήμα 2.8: Η στοίβα δικτύου Linux και η θέση του XDP[17]

2.3 eBPF

Το eBPF (extended Berkeley Packet Filter) είναι μια τεχνολογία που επιτρέπει να εκτελούνται προγράμματα σε ασφαλές, απομονωμένο περιβάλλον, εντός του πυρήνα, χωρίς να χρειάζεται να φορτωθεί κάποιο module. Είναι επέκταση του Berkeley Packet Filter (BPF)[18], το οποίο αρχικά σχεδιάστηκε για φιλτράρισμα πακέτων. Αργότερα βρήκε εφαρμογή και σε άλλες λειτουργίες, όπως παρακολουθήση γεγονότων (events) διαφορετικών από την άφιξη πακέτων, και η αρχιτεκτονική του βελτιώθηκε.

Τα προγράμματα eBPF μπορούν να προσαρτηθούν σε διάφορα σημεία εκτέλεσης στον πυρήνα, και να εκτελούνται κάθε φορά που συμβαίνει κάποιο γεγονός, όπως άφιξη πακέτου, άνοιγμα αρχείου, κάποιο system call κλπ. Αυτά τα σημεία είναι τα hooks, και το XDP hook είναι το σημείο που προσαρτώνται τα XDP προγράμματα, το οποίο βρίσκεται πιο κοντά στην κάρτα δικτύου.

2.3.1 Χάρτες eBPF (eBPF maps)

Οι χάρτες eBPF (eBPF maps) είναι μια γενική(generic) δομή δεδομένων που επιτρέπει την επικοινωνία μεταξύ eBPF (επίπεδο δεδομένων - data plane) και χώρου χρήστη (User Space), καθώς και μεταξύ προγραμμάτων eBPF. Έχει τη δομή Κλειδί/Τιμή (key/value), και μπορεί να είναι ένας από τους παρακάτω τύπους[19, p. 2]:

```
enum bpf_map_type {
    BPF_MAP_TYPE_UNSPEC,
    BPF_MAP_TYPE_HASH,
    BPF_MAP_TYPE_ARRAY,
```

```
        BPF_MAP_TYPE_PROG_ARRAY,
        BPF_MAP_TYPE_PERF_EVENT_ARRAY,
        BPF_MAP_TYPE_PERCPU_HASH,
        BPF_MAP_TYPE_PERCPU_ARRAY,
        BPF_MAP_TYPE_STACK_TRACE,
        BPF_MAP_TYPE_CGROUP_ARRAY,
        BPF_MAP_TYPE_LRU_HASH,
        BPF_MAP_TYPE_LRU_PERCPU_HASH,
    };
```

Για παράδειγμα, ο τύπος BPF_MAP_TYPE_ARRAY είναι ένας χάρτης eBPF στη μορφή πίνακα, όπως χρησιμοποιείται π.χ. στη γλώσσα C. Τα Κλειδιά είναι ακέραιοι αριθμοί, και ο τύπος της Τιμής ορίζεται από το χρήστη κατά τη δημιουργία του χάρτη στο χώρο χρήστη.

Ο τύπος του eBPF χάρτη έχει να κάνει με το πώς αποθηκεύεται εσωτερικά στη μνήμη. Το μέγεθος του κλειδιού και της τιμής ορίζονται από το χρήστη κατά τη δημιουργία του χάρτη, δεν αλλάζουν δυναμικά κατά την εκτέλεση του προγράμματος, και μπορεί να είναι αυθαίρετες δομές, για παράδειγμα η τιμή να είναι κάποιο C struct.

2.3.2 Περιορισμοί στο eBPF

Τα προγράμματα eBPF γράφονται σε ένα υποσύνολο της γλώσσας C, το οποίο μεταγλωττίζεται με τον compiler clang. Ο bytecode που προκύπτει πρέπει να περάσει από τον eBPF verifier[20], ένα πρόγραμμα που ελέγχει αν το eBPF πρόγραμμα πληροί κάποιες προδιαγραφές, για να εξασφαλίσει ότι ο πυρήνας θα συνεχίσει την απρόσκοπτη και ασφαλή λειτουργία του. Κάποιες από αυτές τις προδιαγραφές:

- Το eBPF πρόγραμμα μπορεί να χρησιμοποιήσει συγκεκριμένες βοηθητικές συναρτήσεις (bpf-helpers)[21] και κάποιες ενσωματωμένες συναρτήσεις του compiler. Η εισαγωγή άλλων βιβλιοθηκών δεν επιτρέπεται.
- Από την έκδοση πυρήνα 5.3, τα προγράμματα eBPF έχουν μέχρι 1 εκατομμύριο εντολές assembly[22].
- Απαγορεύονται οι βρόχοι των οποίων το μέγεθος δεν είναι γνωστό κατά τη μεταγλώττιση.
- Η στοίβα έχει μέγεθος 512 bytes.
- Απαγορεύονται οι καθολικές (global) μεταβλητές.
- Απαγορεύονται οι αριθμοί κινητής υποδιαστολής (floating point numbers)
- Όλες οι προσβάσεις στη μνήμη πρέπει να ελέγχονται αν είναι επιτυχημένες πριν τη χρήση τους σε πράξεις.

Η βιβλιοθήκη libbpf[23] είναι η κύρια βιβλιοθήκη για την αλληλεπίδραση του χώρου χρήστη με το eBPF.

2.4 eXpress Data Path (XDP)

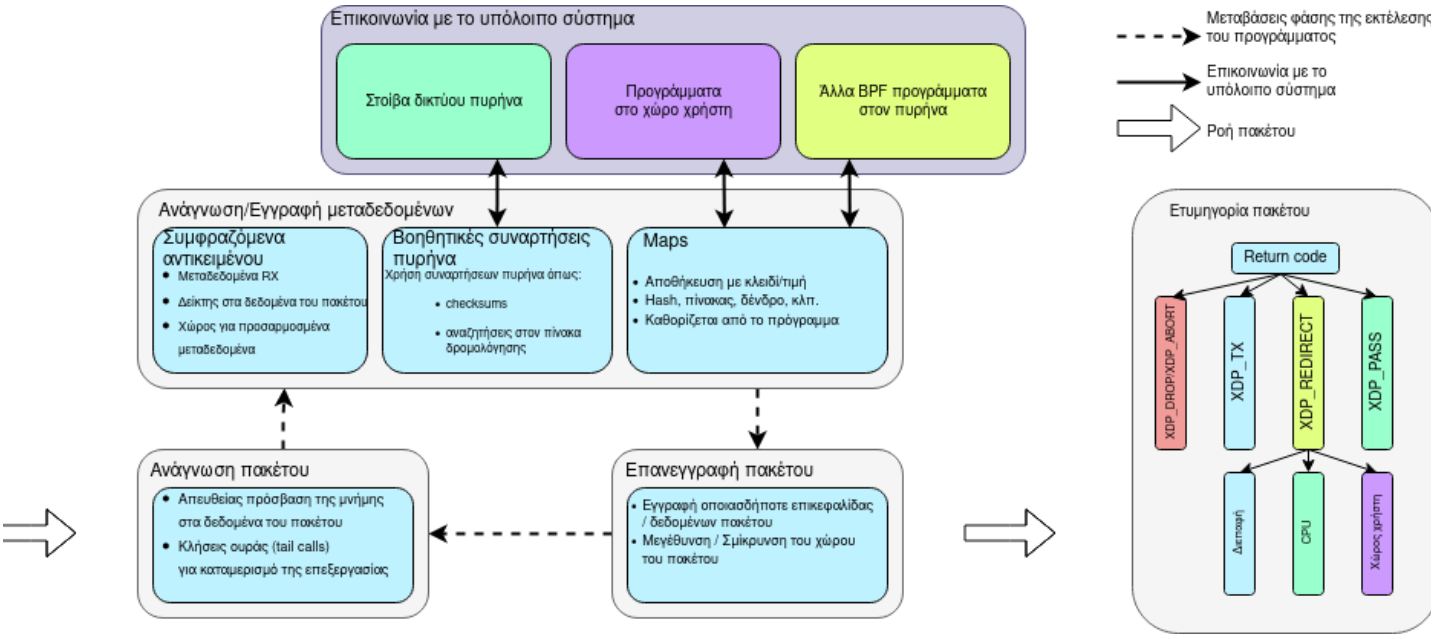
Το eXpress Data Path (XDP) είναι κομμάτι του πυρήνα Linux. Ορίζει ένα περιορισμένο περιβάλλον εκτέλεσης με τη μορφή μιας εικονικής μηχανής (Virtual Machine - VM) που εκτελεί κώδικα eBPF. Αυτό το περιβάλλον εκτελεί προσαρμοσμένα (custom) προγράμματα που επεξεργάζονται ή/και ανακατευθύνουν τα πακέτα που φτάνουν, στο αρχικό στάδιο της

στοίβας δικτύου. Σε αυτό το σημείο δεν έχουν πραγματοποιηθεί ενέργειες όπως η κατανομή μνήμης από τον πυρήνα. Αυτό επιτρέπει τη γρήγορη επεξεργασία μεγάλου όγκου πακέτων, σε υλικό ευρέως διαθέσιμο (commodity hardware)[24].

Το πρόγραμμα που επεξεργάζεται τα πακέτα στο XDP εκτελείται κάθε φορά που εισέρχεται καινούργιο πακέτο (event-driven εκτέλεση), και επιστρέφει ένα από τα πέντε XDP ACTIONS που καθορίζει πώς/αν θα προχωρήσει το πακέτο στη στοίβα δικτύου. Αυτά είναι:

Τιμή	**Ενέργεια**	**Περιγραφή**
0	XDP_ABORTED	Απόρριψη πακέτου λόγω σφάλματος
1	XDP_DROP	Απόρριψη πακέτου
2	XDP_PASS	Αποδοχή πακέτου και συνέχεια στη στοίβα δικτύου
3	XDP_TX	Ανακατεύθυνση του πακέτου στη διεπαφή από την οποία έφτασε
4	XDP_REDIRECT	Ανακατεύθυνση του πακέτου σε διεπαφή διαφορετική από αυτή που έφτασε

Πίνακας 2.2: Ετυμηγορία για κάθε πακέτο στο XDP



Σχήμα 2.9: Ροή εκτέλεσης ενός προγράμματος στο XDP[24]

Σημειώνεται ότι στους ενδιάμεσους ελέγχους που απεικονίζονται στο διάγραμμα, το πρόγραμμα έχει σαν είσοδο το πακέτο ως δείκτη στη δομή xdp_md[25] και πριν τον οποιοδήποτε έλεγχο αυτών πρέπει να ελεγχθεί ρητά κατά πόσο η πρόσβαση στη μνήμη είναι ασφαλής (γραμμή 7).

```
0  int xdp_func(struct xdp_md *ctx)
1  {
2     void *data_end = (void *)(long)ctx->data_end;
3     void *data = (void *)(long)ctx->data;
4     struct ethhdr *eth = data;
5        /* check packet size < ethernet header size
6           Boundary check */
7     if (eth + 1 > data_end){
8           return XDP_DROP; }
9     /* check if packet is ipv4 */
```

```
10    if (bpf_ntohs(eth->h_proto) != ETH_P_IP){
11        return XDP_DROP; }
12    /* get ip header */
13    struct iphdr *iph = data + sizeof(struct ethhdr);
```

Σχήμα 2.10: Έλεγχος για το αν το πακέτο είναι IPv4

```
struct xdp_md {
      __u32 data;
      __u32 data_end;
      __u32 data_meta;
      /* Below access go through struct xdp_rxq_info */
      __u32 ingress_ifindex; /* rxq->dev->ifindex */
      __u32 rx_queue_index;  /* rxq->queue_index  */

      __u32 egress_ifindex;  /* txq->dev->ifindex */
};
```

Σχήμα 2.11: Η δομή xdp_md. Τα πεδία data(αρχή του πακέτου) και data_end(τέλος του πακέτου) χρησιμοποιούνται για να ελεγχθεί κατά πόσο οι προσβάσεις στα δεδομένα του πακέτου είναι νόμιμες, με τρόπο που μπορεί να επαληθεύσει με βεβαιότητα ο eBPF verifier.

Το XDP έχει τρία είδη λειτουργίας:

- **Generic XDP**: Όταν η κάρτα δικτύου δεν υποστηρίζει το XDP, ο πυρήνας προσομοιάζει τη λειτουργία. Αυτό σημαίνει ότι το XDP hook καλείται αργότερα από ότι φτάνει το πακέτο και έχουν ήδη γίνει κάποιες κατανομές μνήμης, συνεπώς η επεξεργασία των πακέτων σε αυτόν τον τρόπο λειτουργίας είναι αρκετά πιο αργή από τους άλλους δύο τρόπους λειτουργίας. Ενδείκνυται για πειραματικά περιβάλλοντα και εφαρμογές καθώς δεν απαιτεί την υποστήριξη από την κάρτα δικτύου.
- **Native XDP**: Σε αυτόν τον τρόπο λειτουργίας το XDP hook είναι διαθέσιμο αμέσως μόλις γίνει διαθέσιμος ο δείκτης προς τα δεδομένα του πακέτου[26, p. 5], δηλαδή πριν οποιαδήποτε εκχώρηση μνήμης, στο επίπεδο των drivers. Είναι ο προκαθορισμένος τρόπος λειτουργίας του XDP.
- **Offloaded XDP**: Το πρόγραμμα XDP εκτελείται από τον επεξεργαστή της κάρτας δικτύου αντί από τον επεξεργαστή του συστήματος, προσφέροντας ακόμα μεγαλύτερη απόδοση, για απλή επεξεργασία πακέτων.

Άλλα συστήματα προγραμματιζόμενης επεξεργασίας πακέτων είναι το Data Plane Development Kit (DPDK)[27] και η γλώσσα P4
(Programming Protocol-independent Packet Processors)[28].

- Σε αντίθεση με το XDP, το οποίο χρησιμοποιεί ένα υποσύνολο της γλώσσας C για τον προγραμματισμό του (λόγω των περιορισμών που εισάγει ο eBPF verifier), το DPDK και η γλώσσα P4 απαιτούν την εκμάθηση και εξοικείωση με καινούριες γλώσσες ειδικού σκοπού (domain-specific languages).

- Ακόμα, η γλώσσα P4 απαιτεί εξειδικευμένο hardware, ενώ το XDP είναι μία software λύση και η χρήση του εξαρτάται από τους drivers της κάρτας δικτύου.
- Το XDP, επειδή δεν παρακάμπτει τον πυρήνα αλλά είναι κομμάτι του, δεν έχει το επιπλέον κομμάτι της διαχείρισης και συντήρησης του hardware, το οποίο συνεχίζει να το διαχειρίζεται ο πυρήνας. Αυτό δεν συμβαίνει με λύσεις που παρακάμπτουν εντελώς τον πυρήνα, όπως το DPDK, καθώς αυτές χρειάζεται να έχουν απευθείας πρόσβαση στο hardware, και χάνουν τις λειτουργικότητες που προσφέρει ο πυρήνας.

Το XDP υποστηρίζεται στον πυρήνα Linux από την έκδοση 4.8.[29]

2.5 Πιθανοτικές δομές δεδομένων

Οι πιθανοτικές δομές δεδομένων, σε αντίθεση με τις ντετερμινιστικές, δε δίνουν πάντα σωστή απάντηση στο ερώτημα “Υπάρχει το στοιχείο Χ στη δομή;”. Χρησιμοποιούνται όταν έχουμε πολλά στοιχεία και περιορισμένο χώρο αποθήκευσης, καθώς τα στοιχεία δεν αποθηκεύονται αυτούσια, αλλά αποθηκεύεται το αποτέλεσμα κάποιου είδους επεξεργασίας τους, όπως μια συνάρτηση κατακερματισμού. Για παράδειγμα, όταν έχουμε να αποθηκεύσουμε εκατομμύρια στοιχεία, το να χρησιμοποιήσουμε πίνακα ο οποίος θα βρίσκεται στη RAM δε θα είναι καθόλου αποδοτικό.

Η συνάρτηση κατακερματισμού είναι μία συνάρτηση η οποία αντιστοιχεί αντικείμενα οποιουδήποτε μεγέθους σε αντικείμενα ορισμένου μεγέθους. Για παράδειγμα, μια συνάρτηση κατακερματισμού μπορεί να αντιστοιχίζει συμβολοσειρές οποιουδήποτε μεγέθους, σε συμβολοσειρές μεγέθους 128 bits.

Το τίμημα για το λιγότερο χώρο που χρησιμοποιούμε είναι ότι οι αναζητήσεις στοιχείων δίνουν σωστή απάντηση με πιθανότητα p<1. Οι δομές που εξετάζουμε έχουν την ιδιότητα να απαντούν πάντα σωστά όταν το στοιχείο υπάρχει στη δομή. Αν το στοιχείο δεν υπάρχει στη δομή, τότε η απάντηση μπορεί να είναι αρνητική, αλλά μπορεί να είναι και θετική (το στοιχείο να βρεθεί εσφαλμένα στη δομή).
Χρησιμοποιούμε την ακόλουθη παραδοχή:

	Το στοιχείο έχει εισαχθεί στη δομή	
Απάντηση της δομής	**ΟΧΙ**	**ΝΑΙ**
ΟΧΙ	TRUE NEGATIVE (TN)	FALSE NEGATIVE(FN)
ΝΑΙ	FALSE POSITIVE (FP)	TRUE POSITIVE (TP)

Πίνακας 2.3: Ορισμός των Εσφαλμένων/Πραγματικών Θετικών/Αρνητικών αποτελεσμάτων

Συνεπώς οι παρακάτω δομές επιτρέπουν εσφαλμένα θετικές απαντήσεις (False Positives) αλλά όχι εσφαλμένα αρνητικές απαντήσεις (False Negatives).

2.5.1 Bloom Filter

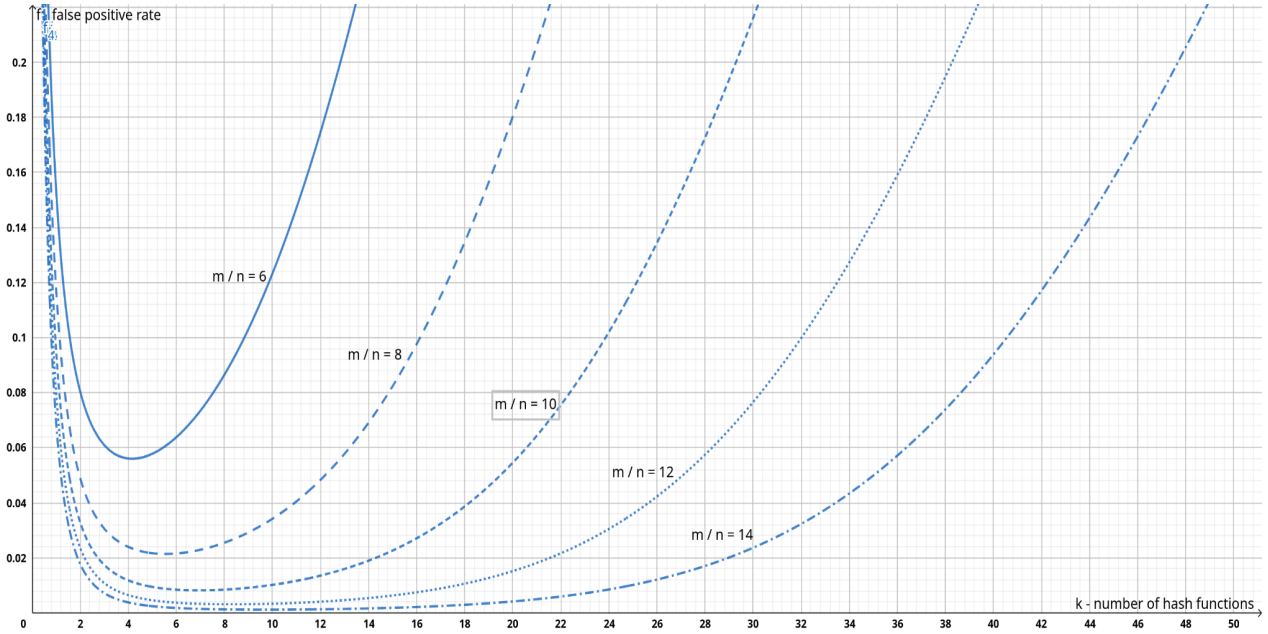
Το Bloom Filter[30] είναι ένας μία πιθανοτική δομή δεδομένων που υποστηρίζει Εισαγωγές (Insertions) και Αναζητήσεις (Queries).

2.5.1.1 Λειτουργία

Χρησιμοποιείται πίνακας m bits. Αρχικά όλα τα bits του πίνακα είναι 0.

- **Εισαγωγή στοιχείου**: Για να εισαχθεί ένα στοιχείο στο Bloom Filter, περνά από k ανεξάρτητες μεταξύ τους συναρτήσεις κατακερματισμού (hash functions), οι οποίες αντιστοιχίζουν το στοιχείο με κάποιο από τα m bits. Οι k θέσεις που προκύπτουν στον πίνακα τίθενται ίσες με 1.
- **Αναζήτηση στοιχείου**: Για να απαντήσουμε αν ένα στοιχείο έχει εισαχθεί στη δομή, βρίσκουμε τις k θέσεις του πίνακα χρησιμοποιώντας τις συναρτήσεις κατακερματισμού. Αν έστω ένα από τα bits είναι 0, το στοιχείο σίγουρα δεν υπάρχει στη δομή. Σε αντίθετη περίπτωση, το στοιχείο υπάρχει στη δομή με κάποια πιθανότητα σφάλματος.

Για ένα Bloom Filter μεγέθους m bits, με k συναρτήσεις κατακερματισμού και με n αποθηκευμένα στοιχεία, η πιθανότητα σφάλματος είναι $f \simeq (1 - e^{(-kn/m)})^k$.



Σχήμα 2.12: Η πιθανότητα σφάλματος f συναρτήσει του λόγου m/n

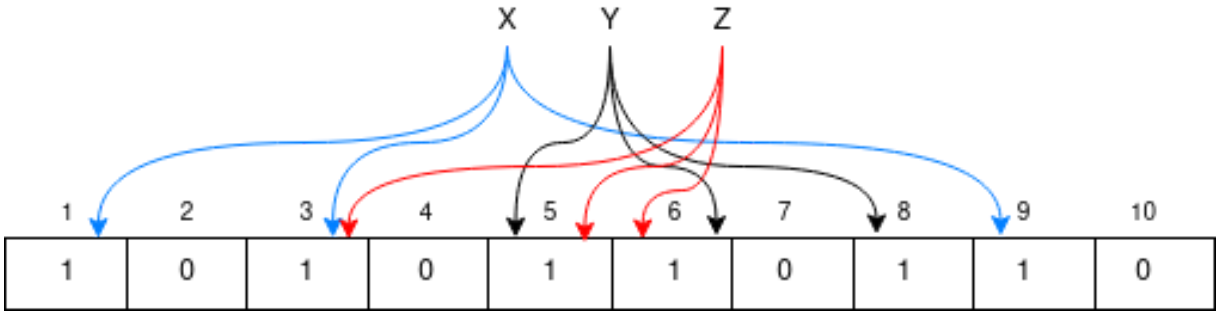
Με σταθερό το λόγο m/n, μπορούμε να υπολογίσουμε πόσες συναρτήσεις κατακερματισμού χρειαζόμαστε για να έχουμε το μικρότερο σφάλμα[31]:

$$k_{opt} = (m/n) ln2$$

Στο παρακάτω παράδειγμα εξηγείται πώς μπορούν να προκύψουν ψευδώς θετικές απαντήσεις σε ερωτήματα.

1. Έστω ότι έχουμε ένα Bloom Filter με m = 10, k = 3.
2. Αρχικά όλα τα bits είναι 0.

3. Προσθέτουμε το στοιχείο X. Έστω ότι οι 3 θέσεις που προκύπτουν από τις συναρτήσεις κατακερματισμού είναι οι 1, 3 και 9. Αυτά τα bits τίθενται ίσα με 1.
4. Στη συνέχεια προσθέτουμε το στοιχείο Y ≠ X. Έστω ότι οι 3 θέσεις που προκύπτουν από τις συναρτήσεις κατακερματισμού είναι οι 5, 6 και 8. Αυτά τα bits τίθενται ίσα με 1.
5. Κάνουμε ερώτημα για το στοιχείο Z, με Z ≠ X και Z ≠ Y. Για να απαντήσουμε αν το στοιχείο υπάρχει στη δομή, βρίσκουμε τις 3 θέσεις που προκύπτουν από τις συναρτήσεις κατακερματισμού. Έστω ότι αυτές είναι οι 3, 5 και 6.
6. Τα bits σε αυτές τις θέσεις του πίνακα είναι ίσα με 1. Συνεπώς η δομή θα απαντήσει, λανθασμένα, ότι το στοιχείο Z υπάρχει.



Σχήμα 2.13: Παράδειγμα ψευδώς θετικής απάντησης για το στοιχείο Z στο Bloom Filter, με m = 10 και k = 3.

2.5.1.2 Δυνατότητες

Το Bloom Filter δεν αποθηκεύει τα στοιχεία σε ένα σύνολο, παρά μόνο την πληροφορία του αν ανήκουν στο σύνολο αυτό. Για αυτό το μέγεθός του δεν εξαρτάται από το πόσο μεγάλο είναι το μέγεθος του στοιχείου, αν είναι μερικά bytes ή Megabytes, αλλά μόνο από το πλήθος των στοιχείων και την επιθυμητή πιθανότητα σφάλματος. Αυτή η ιδιότητα το καθιστά πολύ χρήσιμο σε περιπτώσεις που η διαθέσιμη μνήμη είναι περιορισμένη.

Η πολυπλοκότητα χρόνου της αναζήτησης στο Bloom Filter είναι σταθερή (Θ(k)), και το καθιστά πολύ πιο αποδοτικό από δέντρα (O(logn)) και πίνακες/λίστες (O(n)).

Το Bloom Filter υποστηρίζει εισαγωγές και αναζητήσεις, αλλά δεν υποστηρίζει διαγραφές και αλλαγή μεγέθους. Οι διαγραφές δεν υποστηρίζονται, γιατί έτσι θα είχαμε την πιθανότητα εσφαλμένης αρνητικής απάντησης (False Negatives). Για την αλλαγή μεγέθους, όταν το πλήθος των στοιχείων αρχίζει να επηρεάζει σημαντικά την πιθανότητα σφάλματος, τότε πρέπει να κατασκευαστεί καινούριο Bloom Filter με περισσότερα bits για να υποστηρίξει το μεγαλύτερο πλήθος στοιχείων.

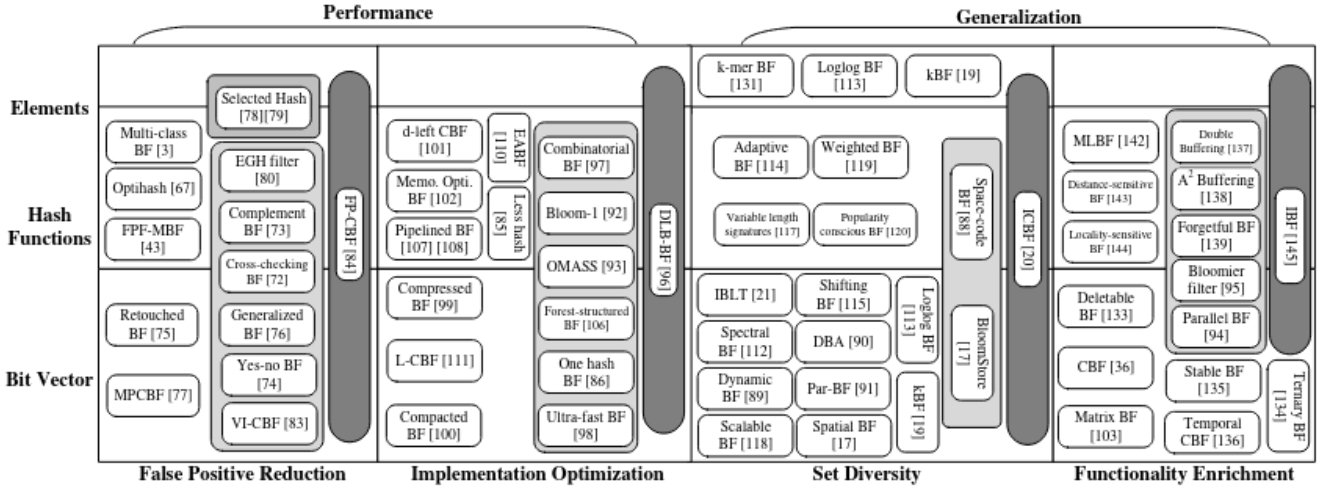
2.5.1.3 Παραλλαγές

Μερικές από τις παραλλαγές / επεκτάσεις του Bloom Filter είναι οι εξής (δεν αναφέρονται όλες):

- Counting Bloom Filter[32]: σε κάθε θέση του πίνακα δεν έχουμε ένα bit, αλλά ένα μετρητή που αυξάνεται όταν προστίθεται ένα στοιχείο και μειώνεται όταν

διαγράφεται ένα στοιχείο. Υποστηρίζει τη διαγραφή στοιχείων από τη δομή, με τίμημα τη χρήση περισσότερων bits ανά στοιχείο. Επίσης, αν γίνει διαγραφή ενός στοιχείου που δεν έχει προηγουμένως εισαχθεί στη δομή, εισάγεται η πιθανότητα false negatives.

- Compressed Bloom Filter[33]: Χρησιμοποιείται στις περιπτώσεις που το Bloom Filter χρειάζεται να μεταδοθεί σαν μήνυμα, οπότε και είναι χρήσιμη η συμπίεσή του.
- Bloomier Filter[34]: Σε κάθε στοιχείο ανατίθεται η τιμή μιας συνάρτησης. Αποθηκεύει μία συμπιεσμένη απεικόνιση μιας συσχέτισης από κλειδιά σε τιμές.



Σχήμα 2.14: Κατηγοριοποίηση επεκτάσεων του Bloom Filter[35]

2.5.1.4 Παραδείγματα χρήσης

Τα Bloom Filters χρησιμοποιούνται στο Bigtable[36] της Google και στο Apache Cassandra[37] τα οποία είναι από τα πιο δημοφιλή κατανεμημένα συστήματα αποθήκευσης μεγάλου όγκου δεδομένων. Συγκεκριμένα, αυτά τα συστήματα διατηρούν τα δεδομένα σε πίνακες Ταξινομημένων Συμβολοσειρών (Sorted String Tables) που βρίσκονται στο δίσκο και είναι δομημένοι ως χάρτες κλειδιού-τιμής. Όταν ο χρήστης αναζητά μια πληροφορία, το σύστημα δεν ξέρει σε ποιον πίνακα βρίσκεται το ζητούμενο. Για αυτό διατηρεί ένα Bloom Filter για κάθε πίνακα στη RAM, και τα χρησιμοποιεί για να βρει σε ποιον πίνακα βρίσκεται η ζητούμενη πληροφορία. Έτσι αποφεύγονται άσκοπες χρονοβόρες προσπελάσεις στο δίσκο.

Άλλες χρήσεις του Bloom Filter:

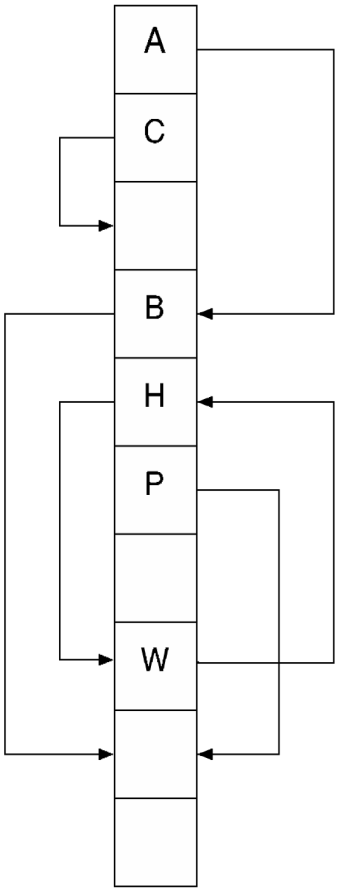
- Η πλατφόρμα Medium χρησιμοποιεί Bloom Filters για να αποφεύγει να προτείνει στους αναγνώστες άρθρα που έχουν ήδη διαβάσει[38].
- Οι εξυπηρετητές του παρόχου περιεχομένου Akamai[39] χρησιμοποιούν Bloom Filters για να μην αποθηκεύουν στην προσωρινή μνήμη περιεχόμενο που οι χρήστες ζητούν μόνο μία φορά. Αποθηκεύουν σε Bloom Filter τις αιτήσεις που κάνουν οι χρήστες για περιεχόμενο, και αποθηκεύουν το περιεχόμενο στην cache κατά τη δεύτερη αίτηση. Χρησιμοποιείται η παραλλαγή Counting Bloom Filter, η οποία υποστηρίζει τη διαγραφή στοιχείων, απαραίτητη όταν ένα στοιχείο αφαιρείται από την cache του εξυπηρετητή.

2.5.2 Cuckoo Filter

Το Cuckoo Filter[40] είναι μία συμπαγής παραλλαγή ενός πίνακα cuckoo κατακερματισμού[41] που αποθηκεύει μόνο αποτυπώματα (fingerprints) αντί για ζευγάρια κλειδιού-τιμής. Το αποτύπωμα ενός αντικειμένου είναι ένα bit string που προκύπτει από μια συνάρτηση κατακερματισμού.

2.5.2.1 Cuckoo Hashing

Το Cuckoo hashing είναι ένας τρόπος επίλυσης των συγκρούσεων που προκύπτουν όταν σε ένα πίνακα κατακερματισμού (hashing table) δύο στοιχεία καταλήγουν στην ίδια θέση. Έχουμε δύο, αντί για μία, συναρτήσεις κατακερματισμού, έστω h_1 και h_2. Κατά την εισαγωγή ενός στοιχείου x, αν στη θέση του πίνακα $h_1(x)$ υπάρχει άλλο στοιχείο, έστω x', τότε το x αποθηκεύεται στη θέση $h_1(x)$ και το x' "εκδιώχνεται" στη θέση $h_2(x')$. Αν σε αυτή την τελευταία υπάρχει άλλο στοιχείο, το x' παίρνει τη θέση του και η διαδικασία συνεχίζει μέχρι να βρεθεί κάποια άδεια θέση ή μέχρι ένα μέγιστο αριθμό επανατοποθετήσεων (relocations) για αποφυγή κύκλων.



Σχήμα 2.15: Παράδειγμα Cuckoo Hashing[42]. Τα βέλη δείχνουν την εναλλακτική θέση κάθε κλειδιού. Ένα νέο κλειδί που εισάγεται στη θέση του A, μετακινεί το A στη θέση του B, και το B μετακινείται στην εναλλακτική θέση του, η οποία είναι κενή. Η εισαγωγή καινούργιου στοιχείου στη θέση του H δεν θα είναι επιτυχημένη: Το H είναι μέρος ενός κύκλου (μαζί με το W), και το καινούργιο στοιχείο θα ξανά έφευγε από τη θέση του.

2.5.2.2 Λειτουργία

Οι τροποποιήσεις που εφαρμόζει το Cuckoo Filter σε σχέση με το Cuckoo Hashing είναι οι εξής:

- Αποθηκεύονται τα αποτυπώματα των στοιχείων και όχι τα ίδια τα στοιχεία
- Ο πίνακας που αποθηκεύονται τα αποτυπώματα είναι ένας πίνακας από buckets, με χωρητικότητα ενός ή περισσότερων στοιχείων. Όταν η χωρητικότητα είναι ίση με 1, έχουμε ένα Cuckoo Hashing Table.
- Χρησιμοποιείται partial-key cuckoo hashing για να βρεθεί η εναλλακτική θέση ενός αποτυπώματος. Αυτή η μέθοδος επιτρέπει, δεδομένου του αποτυπώματος και της θέσης στην οποία βρίσκεται, να βρεθεί η εναλλακτική θέση, χωρίς να γνωρίζουμε το αρχικό στοιχείο.

  Για ένα στοιχείο x, θα έχουμε το εξής:

  $h_1(x) = hash(x)$

  $h_2(x) = h_1(x) \oplus hash(fingerprint(x))$

Η πράξη XOR έχει μια σημαντική ιδιότητα: το $h_1(x)$ μπορεί να υπολογιστεί ξέροντας το $h_2(x)$ και το αποτύπωμα και χρησιμοποιώντας τον ίδιο τύπο, δηλαδή

$h_1(x) = h_2(x) \oplus hash(fingerprint(x))$

Χρησιμοποιείται πίνακας n buckets, χωρητικότητας m αποτυπωμάτων ο καθένας. Αρχικά όλα τα buckets είναι άδεια

- **Εισαγωγή στοιχείου**:
  - Υπολογίζουμε το αποτύπωμα και τα buckets που αντιστοιχούν στο στοιχείο.
  - Αν κάποιο bucket έχει χώρο, το αποτύπωμα αποθηκεύεται σε αυτό το bucket.
  - Αν και τα δύο buckets είναι πλήρη, διαλέγουμε τυχαία ένα από τα δύο.
  - Στη συνέχεια, διαλέγουμε τυχαία κάποιο αποτύπωμα από το bucket που επιλέξαμε, διαφορετικό από αυτό που θέλουμε να εισάγουμε, και επιχειρούμε να το μετακινήσουμε στο εναλλακτικό bucket του.
  - Αν το εναλλακτικό bucket είναι και αυτό πλήρες, επαναλαμβάνουμε τη διαδικασία μέχρι να βρεθεί χώρος σε κάποιο bucket, ή μέχρι ένα μέγιστο αριθμό επαναλήψεων. Αν φτάσουμε στο μέγιστο αριθμό επαναλήψεων, η εισαγωγή του στοιχείου αποτυγχάνει.

Algorithm 1: `Insert(x)`

```
f = fingerprint(x);
i1 = hash(x);
i2 = i1 ⊕ hash(f);
if bucket[i1] or bucket[i2] has an empty entry then
    add f to that bucket;
    return Done;
// must relocate existing items;
i = randomly pick i1 or i2;
for n = 0; n < MaxNumKicks; n++ do
    randomly select an entry e from bucket[i];
    swap f and the fingerprint stored in entry e;
    i = i ⊕ hash(f);
    if bucket[i] has an empty entry then
        add f to bucket[i];
        return Done;
// Hashtable is considered full;
return Failure;
```

Σχήμα 2.16: Εισαγωγή στοιχείου στο Cuckoo Filter[40]

- **Αναζήτηση στοιχείου**: Για να ελέγξουμε αν ένα στοιχείο υπάρχει στη δομή, αρκεί να αναζητήσουμε τα δύο buckets που αντιστοιχούν σε αυτό το στοιχείο. Αν το αποτύπωμα του στοιχείου δε βρίσκεται σε κανένα από τα δύο buckets, το στοιχείο δεν υπάρχει στη δομή.
- **Διαγραφή στοιχείου**: Για να διαγράψουμε ένα στοιχείο από τη δομή, αναζητούμε το αποτύπωμα στα δύο buckets που αντιστοιχούν στο στοιχείο. Πρέπει να είμαστε σίγουροι ότι αυτό το στοιχείο έχει εισαχθεί στο φίλτρο, αλλιώς δημιουργούνται false negatives. Αν το αποτύπωμα βρεθεί σε κάποιο από τα δύο buckets, αφαιρείται μία φορά από αυτό το bucket. Αυτό εγγυάται ότι αν δύο στοιχεία έχουν το ίδιο αποτύπωμα και αντιστοιχίζονται στα ίδια buckets, θα διαγραφεί το ένα από αυτά και δε θα διαγραφούν και τα δύο.

2.5.3 Morton Filter

Το Morton Filter[43] είναι μια παραλλαγή του Cuckoo Filter, που εκμεταλλεύεται τη δομή της μνήμης για την οποία προορίζεται.

Το Morton Filter είναι παρόμοιο με το Cuckoo Filter, όσον αφορά τα εξής:

- Είναι λογικά οργανωμένο ως ένας γραμμικός πίνακας από buckets, και σε κάθε bucket μπορεί να αποθηκευτεί συγκεκριμένος αριθμός αποτυπωμάτων.
- Κάθε αποτύπωμα μπορεί να αποθηκευτεί σε δύο υποψήφια buckets, τα οποία καθορίζονται από δύο ανεξάρτητες συναρτήσεις κατακερματισμού.
- Αν κάποιο από τα δύο buckets δεν είναι πλήρες, τετριμμένα το αποτύπωμα αποθηκεύεται σε αυτό, και οι συγκρούσεις επιλύονται με cuckoo hashing.

Το Morton Filter διαφέρει από το Cuckoo Filter στα παρακάτω:

- Τα buckets δεν αποθηκεύουν αποτυπώματα μέχρι να είναι πλήρη, αλλά τα περισσότερα έχουν κενές θέσεις.
- Τα buckets δεν αποθηκεύονται με τη λογική μορφή τους στη μνήμη, αλλά συμπιεσμένα σε blocks. Αυτό έχει το πλεονέκτημα ότι σε μια γραμμή cache βρίσκονται περισσότερα buckets.
- Οι δύο συναρτήσεις κατακερματισμού διαφοροποιούνται. Η συνάρτηση h_1 ονομάζεται πρωτεύουσα συνάρτηση κατακερματισμού και η συνάρτηση h_2 δευτερεύουσα συνάρτηση κατακερματισμού, και τα αντίστοιχα buckets που αντιστοιχούν σε ένα στοιχείο είναι το πρωτεύον και το δευτερεύον bucket. Κατά την εισαγωγή ενός στοιχείου, πάντα επιχειρείται η εισαγωγή του στο πρωτεύον bucket, και μόνο όταν αυτό αποτύχει επιχειρείται η εισαγωγή του στο δευτερεύον bucket. Στο Cuckoo Filter επιλέγεται τυχαία κάποιο από τα 2 buckets, χωρίς διάκριση.
- Για την εύρεση του εναλλακτικού bucket χρησιμοποιείται η τεχνική even-odd partial key cuckoo hashing, η οποία έχει σαν αποτέλεσμα τα υποψήφια buckets να βρίσκονται σε κοντινές θέσεις στη μνήμη.
    - Even-off partial key cuckoo hashing: Για ένα αυθαίρετο στοιχείο K, με αποτύπωμα $H_F(K)$, β το bucket στο οποίο αντιστοιχίζεται, B το πλήθος των buckets ανά Block, n ο συνολικός αριθμός των buckets, H μια συνάρτηση κατακερματισμού όπως η MurmurHash[44], και map(x,n) μια συνάρτηση που αντιστοιχίζει την τιμή x στο διάστημα [0,n-1], θα έχουμε

$$
\begin{aligned}
H_1(K) &= map(H(K), n) \\
H_2(K) &= map(\ H_1(K) + (-1)^{H_1(K)\&1} * offset(H_F(K)), n) \\
H'(\beta, H_F(K)) &= map(\ \beta + (-1)^{\beta\&1} * offset(H_F(K)), n) \\
offset(F_x) &= [B + (F_x \% OFF\_RANGE)] \mid 1
\end{aligned}
$$

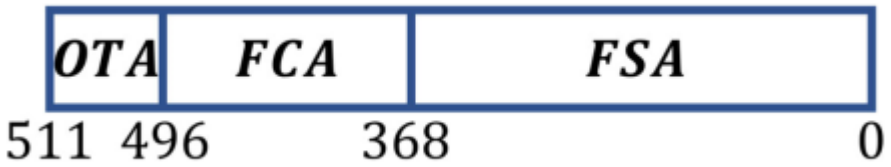
      όπου $H_1(K)$, $H_2(K)$ μας δίνουν το πρωτεύον και δευτερεύον bucket του K αντίστοιχα, και με την H' υπολογίζουμε το εναλλακτικό bucket του στοιχείου, έχοντας διαθέσιμα το αποτύπωμα και το άλλο bucket.
      Με αυτές τις συναρτήσεις, το Morton Filter χωρίζεται λογικά σε 2 μέρη, τα άρτια και τα περιττά buckets. Αν το bucket είναι περιττό, προσθέτουμε το offset στο πρωτεύον bucket, ενώ αν είναι ζυγό, το αφαιρούμε. Τα offsets είναι πάντα περιττά, λόγω του όρου (| 1), και έτσι έχουμε αυτή την εναλλαγή. Χωρίζοντας τα buckets με αυτό τον τρόπο, μπορούμε να υπολογίσουμε το εναλλακτικό bucket του K χωρίς να ξέρουμε το ίδιο το K, αλλά το αποτύπωμά του.
- Το Morton Filter υποστηρίζει την αυτόματη αλλαγή μεγέθους (self-resizing), άρα και δυναμικά φορτία (workloads) το μέγεθος των οποίων δεν είναι γνωστό εκ των προτέρων.

2.5.3.1 Οργάνωση

Το Morton Filter είναι οργανωμένο σε Blocks, τα οποία περιέχουν λογικά buckets και πληροφορίες για αυτά και το περιεχόμενό τους. Το μέγεθος του Block διαφοροποιείται ανάλογα με τη δομή της μνήμης για την οποία προορίζεται, ώστε σε μία γραμμή cache να υπάρχει ακέραιος αριθμός Block. Για παράδειγμα, για μία γραμμή μεγέθους 512 bits, το Block θα είχε μέγεθος 256 ή 512 bits.

Κάθε Block έχει τρία κύρια μέρη:

- τον **Πίνακα Αποθήκευσης Αποτυπωμάτων** (Fingerprint Storage Array - FSA): Είναι ο πίνακας στον οποίο αποθηκεύονται τα αποτυπώματα ενός Block. Τα αποτυπώματα από διαδοχικά buckets αποθηκεύονται ως ακολουθία, χωρίς κενά, και τα κενά αποτυπώματα βρίσκονται στο τέλος του πίνακα. Ο FSA τυπικά έχει λιγότερες θέσεις από τις συνολικές λογικές θέσεις που έχουν όλα τα buckets στη λογική αναπαράσταση.
- τον **Πίνακα Μετρητών Πληρότητας** (Fullness Counter Array - FCA): κωδικοποιεί τη λογική δομή του Block, συσχετίζοντας ένα μετρητή πληρότητας με κάθε bucket στο Block, ο οποίος παρακολουθεί πόσες θέσεις είναι κατειλημμένες. Επιτρέπει επιτόπου αναγνώσεις και εγγραφές στα buckets στο FSA χωρίς να χρειάζεται να υλοποιηθεί η πλήρης λογική δομή του Block. Για να διαβάσουμε ένα bucket στο FSA, αρκεί να προσθέσουμε όλους τους προηγούμενους μετρητές στο FCA και να βρούμε το offset, από το οποίο και έπειτα θα ξεκινήσει η ανάγνωση στο FSA.
- τον **Πίνακα Παρακολούθησης Υπερχείλισης** (Overflow Tracking Array - OTA): είναι ένα διάνυσμα από bits το οποίο θέτει ένα bit ίσο με 1 κάθε φορά που ένα αποτύπωμα υπερχειλίζει, δηλαδή μετακινείται στο εναλλακτικό bucket του και δεν αποθηκεύεται στο πρωτεύον bucket του. Κατά την αναζήτηση ενός στοιχείου, αυτός ο πίνακας μας επιτρέπει να γνωρίζουμε αν αρκεί η πρόσβαση στο πρωτεύον bucket, για να αποφανθούμε αν το στοιχείο υπάρχει στη δομή, ή χρειάζεται να γίνει πρόσβαση και στο δευτερεύον bucket.



Σχήμα 2.17: Παράδειγμα ενός Block μεγέθους 512 bits. Το Block έχει FSA με 46 θέσεις για αποτυπώματα 8-bit, FCA με μετρητές 2-bit για 64 buckets (κάθε bucket έχει 3 θέσεις για αποτυπώματα), και OTA μεγέθους 16-bit.

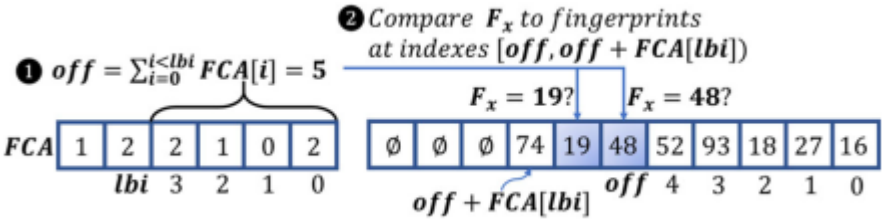
2.5.3.2 Λειτουργία

- **Αναζήτηση στοιχείου**: Αναζητούμε το αποτύπωμα F_x του στοιχείου K_x στη δομή. Αρχικά υπολογίζεται η πρωτεύουσα συνάρτηση κατακερματισμού H_1, για να υπολογιστεί στη συνέχεια ο καθολικός δείκτης του πρωτεύοντος bucket (global bucket index - glbi), έστω glbi1. Η ακέραια διαίρεση του glbi1 με τον αριθμό buckets ανά Block μας δίνει τον αριθμό του πρωτεύοντος Block, και το υπόλοιπο της διαίρεσης μας δίνει τον τοπικό (για το Block) δείκτη του πρωτεύοντος bucket, έστω lbi1. Έχοντας αυτά γνωστά, η συνάρτηση table_read_and_compare αναζητά το στοιχείο στο Block επί τόπου, χωρίς να χρειαστεί να υλοποιηθεί η λογική μορφή του Block.
    - Περιγραφή της συνάρτηση table_read_and_compare:
        i. Υπολογίζουμε τη διαφορά θέσης (offset) του K_x σε αποτυπώματα, από την αρχή του Block. Στο παράδειγμα, το lbi του K_x είναι ίσο με 4, οπότε αθροίζουμε τους μετρητές όλων των buckets πριν από το bucket 4 (0 μέχρι και 3), το οποίο μας δίνει μια διαφορά θέσης ίση με

5.

ii. Επειδή η αρίθμηση ξεκινά από το 0, το πρώτο αποτύπωμα του bucket 4 βρίσκεται στη θέση 5. Εφόσον FCA[lbi] = 2, το bucket 4 έχει 2 αποτυπώματα.

iii. Έτσι, η ανάγνωση του FSA ξεκινά από τη θέση 5 και σταματά πριν τη θέση offset + FCA[lbi] = 5 + 2 = 7. Αν κάποιο από τα αποτυπώματα είναι ίδιο με το F_x, τότε η συνάρτηση επιστρέφει True, αλλιώς επιστρέφει False.



Σχήμα 2.18: Παράδειγμα αναζήτησης αποτυπώματος στο Block

Έχοντας το αποτέλεσμα της table_read_and_compare για το πρωτεύον bucket του στοιχείου, και το αν το bit στον πίνακα OTA που αντιστοιχεί στο στοιχείο είναι ίσο με 0, υπολογίζουμε τη λογική διάζευξη αυτών (λογικό OR). Αν αυτή είναι ψευδής, προχωρούμε στην αναζήτηση στο δευτερεύον bucket του στοιχείου, αλλιώς επιστρέφουμε το αποτέλεσμα της πρώτης κλήσης στην table_read_and_compare.

Algorithm 1 Algorithm for LIKELY_CONTAINS function

```
1: function LIKELY_CONTAINS(MF, K_x)
2:     F_x = H_F(K_x)
3:     glbi1 = H_1(K_x)
4:     block1 = MF.BlockStore[glbi1/B]
5:     lbi1 = mod(glbi1, B)
6:     match = table_read_and_cmp(block1, lbi1, F_x)
7:     if (match or OTA_bit_is_unset(block1, lbi1)) then
8:         return match
9:     else
10:        glbi2 = H_2(K_x)
11:        block2 = MF.BlockStore[glbi2/B]
12:        lbi2 = mod(glbi2, B)
13:        return table_read_and_cmp(block2, lbi2, F_x)
```

Σχήμα 2.19: Αλγόριθμος αναζήτησης του στοιχείου K_x στο Morton Filter

- **Εισαγωγή στοιχείου**: Πρώτα επιχειρείται η εισαγωγή του στοιχείου στο πρωτεύον bucket, glbi1 (γραμμές 2-6). Η συνάρτηση table_simple_store επιτυγχάνει αν και το υποψήφιο bucket και ο πίνακας FSA του Block έχουν κενές θέσεις, δηλαδή δεν έχουμε υπερχείλιση ούτε στο bucket ούτε στο Block. Αν η συνάρτηση table_simple_store αποτύχει, τότε ο αλγόριθμος προχωρά στο δευτερεύον bucket (γραμμές 10-14). Αν η εισαγωγή και στο δευτερεύον bucket αποτύχει, τότε ο αλγόριθμος προχωρά στην επίλυση της σύγκρουσης (γραμμή 17). Σε αυτό το στάδιο, γίνεται μια σειρά από cuckoo επανατοποθετήσεις.

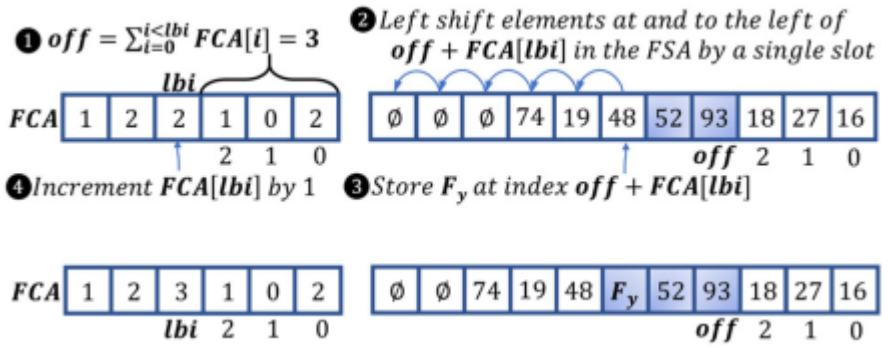
Algorithm 2 Algorithm for INSERT function

```
1: function INSERT(MF, K_x)
2:     F_x = H_F(K_x)
3:     glbi1 = H_1(K_x)
4:     block1 = MF.BlockStore[glbi1/B]
5:     lbi1 = mod(glbi1, B)
6:     success = table_simple_store(block1, lbi1, F_x)
7:     if (success) then
8:         return success
9:     else
10:        set_OTA(block1, lbi1)
11:        glbi2 = H_2(K_x)
12:        block2 = MF.BlockStore[glbi2/B]
13:        lbi2 = mod(glbi2, B)
14:        success = table_simple_store(block2, lbi2, F_x)
15:    if (success) then
16:        return success
17:    return res_conflict(MF, block1, block2, lbi1, lbi2, F_x)
```

Σχήμα 2.20: Ο αλγόριθμος εισαγωγής στοιχείου στο Morton Filter

Παρακάτω έχουμε ένα παράδειγμα του τι συμβαίνει στο Block σε μια επιτυχημένη εισαγωγή ενός στοιχείου, έστω K_x, με αποτύπωμα F_x.

1. Πρώτα υπολογίζεται η διαφορά θέσης (σε αποτυπώματα) του bucket μέσα στο FSA. Σε αυτή την περίπτωση, το τοπικό, στο Block, bucket είναι lbi = 3, άρα προσθέτουμε τους μετρητές πληρότητας όλων των προηγούμενων buckets (buckets 0, 1 και 2).
2. Έπειτα, μετακινούνται όλα τα αποτυπώματα στο FSA, από τη θέση off + FCA[lbi] και μετά, μία θέση αριστερά, για να αφήσουν μία θέση κενή για το καινούριο αποτύπωμα. Οι μετακινήσεις αυτές δεν κοστίζουν πολύ, γιατί όλο το Block βρίσκεται σε μία γραμμή της cache, άρα δε χρειάζονται επιπλέον προσβάσεις στη μνήμη κατά την ανάγνωση ή εγγραφή ενός Block.
3. Το αποτύπωμα F_x αποθηκεύεται στη θέση FSA[off + FCA[lbi]].
4. Το τελικό βήμα είναι να αυξηθεί ο μετρητής πληρότητας του bucket lbi.



Σχήμα 2.21: Παραδειγμα εισαγωγής στοιχείου στο Morton Filter. Ο πίνακας OTA παραλείπεται για ευκολία.

- Παράδειγμα επίλυσης συγκρούσεων κατά την εισαγωγή στοιχείων:

i. Έστω ότι εισάγουμε το στοιχείο K_x, του οποίου το πρωτεύον bucket είναι πλήρες, αλλά το Block B_i στο οποίο περιέχεται το bucket έχει κενές θέσεις. Τότε έχουμε υπερχείλιση του bucket (bucket overflow). Σε αυτή την περίπτωση επιλέγεται ένα αποτύπωμα από το bucket που B_i, έστω το αποτύπωμα F_0, και στη θέση του μετακινείται το αποτύπωμα του K_x(1).

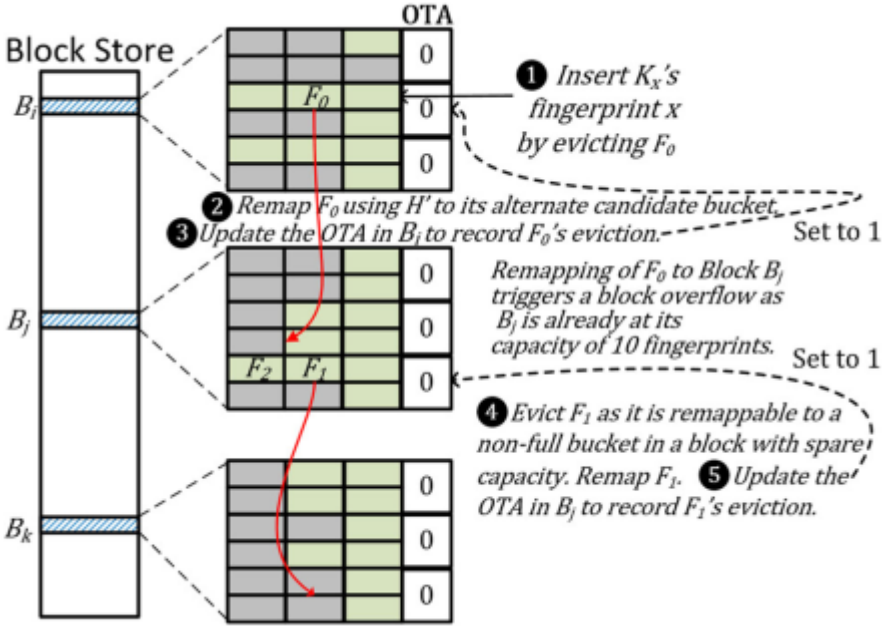
ii. Το αποτύπωμα F_0 μετακινείται στο εναλλακτικό bucket του, το οποίο έστω ότι βρίσκεται στο Block B_j(2).

iii. Το OTA bit στο Block B_i που αντιστοιχεί στο F_0 τίθεται ίσο με 1(3).

iv. Σε αυτό το παράδειγμα, το Block B_j στο οποίο μετακινείται το αποτύπωμα F_0 είναι πλήρες, παρότι το bucket στο οποίο αντιστοιχεί έχει κενές θέσεις. Τότε έχουμε υπερχείλιση του Block (Block overflow). Σε αυτή την περίπτωση, και όταν δεν έχουμε ταυτόχρονα bucket overflow, οποιοδήποτε από τα αποτυπώματα στο Block μπορεί να μετακινηθεί στο εναλλακτικό bucket του.

v. Εν προκειμένω, έστω ότι το F_0 τοποθετείται στη θέση του F_1(4). Το F_1 θα μετακινηθεί στο εναλλακτικό bucket του, στο Block B_k, και επειδή δεν συμβαίνει κάποια υπερχείλιση, η επανατοποθέτηση ολοκληρώνεται.

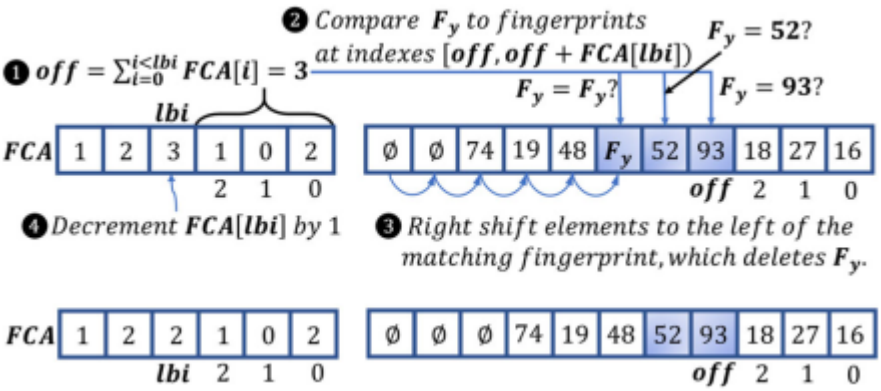
vi. Ο πίνακας OTA στο Block B_j ενημερώνεται για να καταγραφεί η επανατοποθέτηση του αποτυπώματος F_1.



Σχήμα 2.22: Παράδειγμα επίλυσης συγκρούσεων στο Morton Filter.

Αξίζει να σημειωθεί ότι τέτοιου είδους συγκρούσεις είναι σπάνιες, και τα περισσότερα στοιχεία αποθηκεύονται στο πρωτεύον ή δευτερεύον bucket τους.

- **Διαγραφή στοιχείου**: Η διαγραφή ενός στοιχείου είναι παρόμοια με την αναζήτηση. Αφού βρεθεί το στοιχείο με τη διαδικασία που περιγράφηκε πιο πριν, όλα τα αποτυπώματα στο FSA αριστερά του στοιχείου μετακινούνται μία θέση δεξιά. Αν υπάρχουν περισσότερα από ένα ίδια αποτυπώματα στο bucket, διαγράφεται το ένα από αυτά. Αφού γίνει η μετακίνηση των στοιχείων προς τα δεξιά, μειώνεται κατά ένα ο μετρητής πληρότητας του bucket στο οποίο βρέθηκε το στοιχείο.



Σχήμα 2.23: Παράδειγμα διαγραφής στοιχείου στο Morton Filter.

Κεφάλαιο 3: Περιγραφή Μηχανισμού

Η καίρια σημασία του DNS για την ομαλή λειτουργία των εφαρμογών στο διαδίκτυο καθιστά επιτακτική την ανάγκη λήψης μέτρων για την προστασία του. Η υποδομή του DNS αποτελεί συχνά στόχο επιθέσεων, καθώς τα τρωτά σημεία του πρωτοκόλλου παρέχουν τη δυνατότητα εκτέλεσης πολύ αποτελεσματικών επιθέσεων.

Ο μηχανισμός που υλοποιήθηκε στα πλαίσια αυτής της διπλωματικής εργασίας έχει ως στόχο την αντιμετώπιση της επίθεσης DNS Water Torture σε authoritative DNS εξυπηρετητές, στο επίπεδο δεδομένων (data plane).

Τα ονόματα στα αρχεία ζώνης του authoritative εξυπηρετητή αποτελούν, κατά μία έννοια, τα επιτρεπόμενα ονόματα (whitelist) τα οποία θα επιλύσει ο εξυπηρετητής, ενώ θα απορρίψει όλα τα άλλα ονόματα. Τα ονόματα αυτά εισάγονται στις πιθανοτικές δομές δεδομένων Bloom Filter, Cuckoo Filter και Morton Filter, στις οποίες ο εξυπηρετητής ανατρέχει, μέσω XDP, κατά την άφιξη ενός DNS ερωτήματος. Αν βρεθεί το όνομα προς επίλυση στη δομή, το ερώτημα DNS προωθείται στο στρώμα εφαρμογής (application layer), σε αντίθετη περίπτωση το πακέτο με το ερώτημα απορρίπτεται, πριν αποδοθεί μνήμη για αυτό στη στοίβα δικτύου.

Ο μηχανισμός προσαρτάται στο XDP hook, άρα στο επίπεδο του driver της κάρτας δικτύου, και είναι ο λόγος για τον οποίο η επεξεργασία και η απόρριψη μεγάλου όγκου πακέτων είναι αποδοτική, καθώς σε αυτό το επίπεδο δε γίνεται εκχώρηση μνήμης για το πακέτο στον πυρήνα.

Οι δομές επιτρέπουν εσφαλμένες θετικές απαντήσεις (False Positives), αλλά όχι εσφαλμένες αρνητικές απαντήσεις (False Negatives), συνεπώς δεν απορρίπτεται κανένα καλόβουλο ερώτημα DNS προς τον εξυπηρετητή.

Για την αξιολόγηση του μηχανισμού έγινε προσομοίωση επίθεσης Water Torture από έναν επιτιθέμενο (attacker) προς τον authoritative DNS εξυπηρετητή - θύμα. Ο εξυπηρετητής - θύμα είναι authoritative για τρεις DNS ζώνες: τη ζώνη example.com, με 8303 ονόματα, τη ζώνη .nu, με 243293 ονόματα και τη ζώνη .se, με 1406399 ονόματα.

Θα μελετηθούν δύο σενάρια επίθεσης στον εξυπηρετητή:

1. Ονόματα της ζώνης example.com : Σε αυτό το σενάριο ο αμυντικός μηχανισμός έχει φορτωθεί στο XDP hook, οι δομές περιέχουν μόνο τα ονόματα της ζώνης example.com και τα καλόβουλα ερωτήματα αφορούν μόνο αυτά τα ονόματα.
2. Ονόματα όλων των ζωνών : Σε αυτό το σενάριο ο αμυντικός μηχανισμός έχει φορτωθεί στο XDP hook, οι δομές περιέχουν τα ονόματα και των τριών ζωνών (.se, .nu & example.com) και τα καλόβουλα ερωτήματα αφορούν αυτές τις τρεις ζώνες.

Σχετικά με τα ονόματα που χρησιμοποιήθηκαν ως whitelist:

- Τα ονόματα της ζώνης example.com αποτελούνται από subdomains από FQDNs της ζώνης του Ε.Μ.Π. (ntua.gr) και το domain “example.com”

- Τα ονόματα των ζωνών .se και .nu είναι ελεύθερα διαθέσιμα[45]. Δε χρησιμοποιήθηκαν όπως ακριβώς παρέχονται από AXFR requests στις ζώνες αυτές, αλλά κάθε μοναδικό όνομα αντιστοιχήθηκε με μία διεύθυνση IP, άρα και με μία εγγραφή πόρων τύπου A στο αρχείο ζώνης, προς απλοποίηση των απαντήσεων DNS.

Τα ονόματα των ζωνών χρησιμοποιήθηκαν για να εξεταστεί κατά πόσο επηρεάζεται η λειτουργία του μηχανισμού όταν η whitelist περιέχει λιγότερα ή περισσότερα ονόματα.

3.1 Επίπεδο ελέγχου

Το επίπεδο ελέγχου (control plane) περιλαμβάνει τη λογική πίσω από την επεξεργασία των πακέτων. Στη συγκεκριμένη περίπτωση, ορίζει ποια δομή χρησιμοποιείται και ποια ονόματα εισάγονται σε αυτή.

Στο επίπεδο ελέγχου εισάγονται τα ονόματα στις πιθανοτικές δομές, αυτές μεταφέρονται με κατάλληλο τρόπο σε χάρτες eBPF και, τέλος, φορτώνεται το πρόγραμμα eBPF στο XDP hook.

3.1.1 Πιθανοτικές δομές δεδομένων

Στα πειράματα χρησιμοποιήθηκαν οι εξής δομές δεδομένων, των οποίων οι διαστάσεις διαφοροποιούνται σύμφωνα με το πλήθος των ονομάτων που εισάγονται σε αυτές:

3.1.1.1 Bloom Filter

Χρησιμοποιήθηκε ένα Bloom Filter με k = 5 συναρτήσεις κατακερματισμού, με πιθανότητα σφάλματος f = 0.001, με το μέγεθος του πίνακα bit να καθορίζεται από τον τύπο $n = ceil(m / (-k / log(1 - exp(log(f) / k))))$[46]. Επιλέχθηκε η τεχνική double hashing[47], επειδή καταλήγει σε λιγότερους υπολογισμούς σε σχέση με το αν υπολογιζόταν ξεχωριστά κάθε μία από τις πέντε συναρτήσεις κατακερματισμού, και η εξοικονόμηση εντολών είναι σημαντική σε ένα πρόγραμμα eBPF λόγω του ορίου εντολών. Με την τεχνική double hashing, αντί να υπολογίζουμε 5 διαφορετικά hashes, υπολογίζουμε τα δύο πρώτα, και τα υπόλοιπα προκύπτουν ως γραμμικός συνδυασμός αυτών, δηλαδή αν έχουμε τα h_1 και h_2, τα πέντε hashes διαμορφώνονται ως εξής:

- ➢ $hash_1 = h_1$
- ➢ $hash_2 = h_1 + h_2$
- ➢ $hash_3 = h_1 + 2*h_2$
- ➢ $hash_4 = h_1 + 3*h_2$
- ➢ $hash_5 = h_1 + 4*h_2$

3.1.1.2 Cuckoo Filter

Χρησιμοποιήθηκε ένα Cuckoo Filter με 4 θέσεις αποτυπωμάτων για κάθε bucket, σφάλμα ίσο με 0.0003 και μέγιστο αριθμό επανατοποθετήσεων ίσο με 1000. Το αποτύπωμα κάθε στοιχείου έχει μέγεθος 15-bit. Οι κενές θέσεις στα buckets δηλώνονται με την τιμή 0xffff, η οποία είναι 16-bit. Συνεπώς, για να ελέγξουμε αν μία θέση σε κάποιο bucket είναι κενή, αρκεί να ελέγξουμε αν το πιο αριστερό bit (Most Significant Bit - MSB) είναι ίσο με 1. Επίσης χρησιμοποιήθηκε η τεχνική semi sorting, στην οποία ταξινομούνται τα στοιχεία κάθε

bucket, και όταν έχουμε 4 αποτυπώματα ανά bucket, εξοικονομούμε 1 bit ανά στοιχείο στην αποθήκευσή του[40].

3.1.1.3 Morton Filter

Χρησιμοποιήθηκαν δύο Morton Filters, με μέγεθος Block ίσο με 512 bits, γιατί η γραμμή της cache είχε μέγεθος ίσο με 64 bytes = 512 bits, με τις ακόλουθες διαστάσεις[48]:
Morton8:

- μέγεθος αποτυπώματος: 8 bits
- μέγεθος OTA: 16 bits
- buckets/Block: 64
- θέσεις/bucket: 3
- αποτυπώματα/Block: 46

Morton16:

- μέγεθος αποτυπώματος: 16 bits
- μέγεθος OTA: 16 bits
- buckets/Block: 32
- θέσεις/bucket: 3
- αποτυπώματα/Block: 27

3.1.2 Μεταφορά των πιθανοτικών δομών σε χάρτες eBPF

Όπως αναφέρθηκε στο κεφάλαιο 2, οι χάρτες eBPF είναι ζευγάρια κλειδιού/τιμής που μπορεί να είναι διαφόρων μορφών (Hashtable, πίνακας, δέντρο, κλπ). Ανάλογα με τη πιθανοτική δομή που χρησιμοποιείται, αλλάζει και ο τύπος της τιμής στο χάρτη. Σε κάθε περίπτωση, ο χάρτης eBPF είναι τύπου BPF_MAP_TYPE_ARRAY.

- ❖ Για το Bloom Filter, ο πίνακας bit κωδικοποιήθηκε ως ένας πίνακας από 8-bit μη-αρνητικούς ακεραίους.
- ❖ Για το Cuckoo Filter, από τη στιγμή που τα αποτυπώματα χρειάζονται 16 bit (15-bit αποτύπωμα και 1 bit για το αν η θέση είναι κενή), και έχουμε 4 αποτυπώματα ανά bucket, ο τύπος της τιμής στον eBPF χάρτη θα είναι 64-bit μη αρνητικός ακέραιος, και το πλήθος των τιμών θα είναι ίσο με το πλήθος των buckets.
- ❖ Για τα Morton Filter, επιλέχθηκε ο τύπος της τιμής να αντιστοιχεί στο μέγεθος του αποτυπώματος, για πιο εύκολη αναζήτηση. Στο Morton8 η τιμή είναι 8-bit μη αρνητικός ακέραιος, ενώ στο Morton16 η τιμή είναι 16-bit μη αρνητικός ακέραιος.

3.1.3 Φόρτωση του προγράμματος eBPF στο XDP

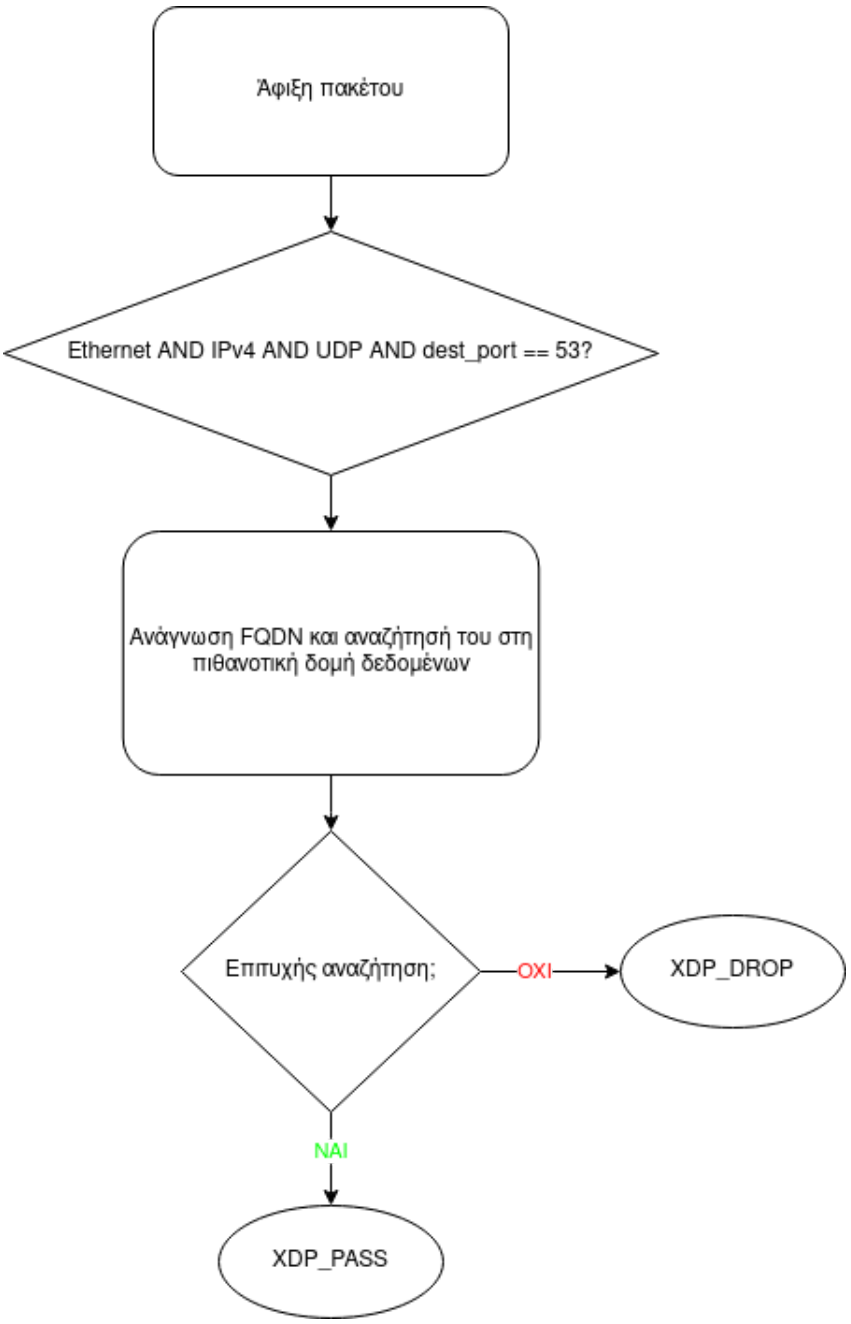
Για τη φόρτωση του προγράμματος eBPF στο XDP hook χρησιμοποιήθηκε η βιβλιοθήκη libbpf και τμήμα του κώδικα[49]. Το τμήμα του προγράμματος που θα φορτωθεί, καθώς και η διεπαφή στην οποία θα φορτωθεί, προσδιορίζονται από το χρήστη.

3.2 Επίπεδο δεδομένων

Στο επίπεδο δεδομένων (data plane) πραγματοποιείται η επεξεργασία των πακέτων, η εξαγωγή του ονόματος και η αναζήτησή του στην εκάστοτε δομή δεδομένων.

Κατά την άφιξη του πακέτου, το πρόγραμμα eBPF ελέγχει αν το πακέτο χρησιμοποιεί τα πρωτόκολλα Ethernet, IPv4, αν έχει πρωτόκολλο μεταφοράς το UDP, και η θύρα προορισμού είναι η 53. Αν το πακέτο που φτάνει δεν είναι ερώτημα DNS, απορρίπτεται.

Εφόσον πρόκειται για DNS ερώτημα, γίνεται ανάγνωση του πλήρους ονόματος του DNS ερωτήματος, με έλεγχο ορίων (boundary checks) για κάθε byte του ονόματος, ώστε ο verifier να μην απορρίψει το πρόγραμμα λόγω επικίνδυνης πρόσβασης στη μνήμη. Στη συνέχεια γίνεται αναζήτηση του ονόματος στη πιθανοτική δομή δεδομένων, η οποία παρέχεται με eBPF χάρτη, μέσω του επιπέδου ελέγχου.
Αν το όνομα βρεθεί στη δομή δεδομένων, το πακέτο προχωρά στη στοίβα δικτύου, ειδάλλως απορρίπτεται.



Σχήμα 3.1: Διάγραμμα ροής του eBPF προγράμματος που υλοποιήθηκε

3.3 Περιορισμοί

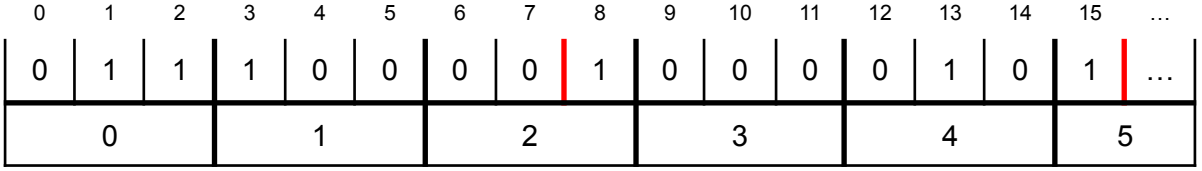
Παρακάτω παρουσιάζονται κάποιοι περιορισμοί που παρουσιάστηκαν κατά την υλοποίηση του μηχανισμού και σχετίζονται με το eBPF.

3.3.1 Έλεγχος των προσβάσεων στη μνήμη

Όπως αναφέρθηκε και στο κεφάλαιο 2, τα προγράμματα eBPF πρέπει να εγκριθούν από τον eBPF verifier, μετά από στατική ανάλυση του bytecode που παράγει ο compiler. Ένα από τα πράγματατα που ελέγχει ο verifier, είναι ότι δε γίνεται πρόσβαση σε μνήμη που δεν ανήκει στο πρόγραμμα, και ότι οι προσβάσεις στη μνήμη είναι καλά ορισμένες.
Κατά την αναζήτηση των στοιχείων στις πιθανοτικές δομές δεδομένων η πρόσβαση στα στοιχεία του eBPF χάρτη που τις περιέχει είναι η εξής:

- στο Bloom Filter, γίνεται μία πρόσβαση για κάθε συνάρτηση κατακερματισμού, δηλαδή το πολύ πέντε προσβάσεις, σε μη-αρνητικούς 8-bit ακεραίους, αν το στοιχείο υπάρχει στη δομή.
- στο Cuckoo Filter, γίνεται μία πρόσβαση για κάθε bucket που ελέγχεται, δηλαδή το πολύ δύο προσβάσεις, σε μη-αρνητικούς 64-bit ακεραίους.
- στο Morton Filter, γίνεται μία πρόσβαση για κάθε Block που ελέγχεται, δηλαδή το πολύ δύο προσβάσεις, σε ένα πίνακα μη-αρνητικών 8-bit/16-bit ακεραίων.

Η τελευταία περίπτωση αποδείχθηκε προβληματική, καθώς στην αναζήτηση ενός στοιχείου στο Morton Filter, χρειάζεται να προσπελάσουμε τους μετρητές των buckets, και η ανάγνωση κάθε bit γίνεται ξεχωριστά. Όταν τα buckets έχουν χωρητικότητα 3 στοιχείων, χρειαζονται 2 bit για την αναπαράσταση του μετρητή στο FCA, και το μέγεθος του ακέραιου στον οποίο αποθηκεύονται είναι ακέραιο πολλαπλάσιο του 2 (σε έναν 8-bit unsigned ακέραιο απεικονίζονται 4 μετρητές, και σε έναν 16-bit unsigned ακέραιο απεικονίζονται 8 μετρητές).
Κατά τον πειραματισμό με άλλες διαστάσεις στο Morton Filter, συγκεκριμένα στην υλοποίηση του Morton Filter με 7 (αντί για 3) θέσεις σε κάθε bucket, και μέγεθος αποτυπώματος ίσο με 8 bit, ο eBPF verifier απορρίπτει το πρόγραμμα eBPF, παρά όλους τους ελέγχους ορίων, για τον εξής λόγο: κατά την ανάγνωση των μετρητών, οι οποίοι τώρα χρειάζονται 3 bits για την αναπαράστασή τους, ο verifier δεν μπορεί να αποφανθεί, κατά τη στατική ανάλυση του κώδικα, πόσες προσβάσεις στη μνήμη θα γίνουν, γιατί αναλύει όλα τα πιθανά μονοπάτια εκτέλεσης του προγράμματος και το βάθος μέχρι το οποίο αναλύει δεν επαρκεί, για να υπάρχει βεβαιότητα για τον αριθμό προσβάσεων στη μνήμη. Επειδή κάθε μετρητής χρειάζεται 3 bits, κάποιοι μετρητές βρίσκονται εξ ολοκλήρου σε έναν 8-bit ακέραιο, ενώ άλλοι απαιτούν την ανάγνωση δύο ακεραίων για τον υπολογισμό τους.



Σχήμα 3.2: Αναπαράσταση των πρώτων 16 bit του πίνακα FCA. Με έντονη μαύρη γραμμή φαίνονται τα όρια κάθε μετρητή, ενώ με κόκκινη γραμμή τα όρια των 8-bit ακεραίων στους οποίους αποθηκεύονται οι μετρητές. Για την ανάγνωση του μετρητή 1 θα χρειαστεί μία πρόσβαση στη μνήμη, ενώ για το μετρητή 2 θα χρειαστούν δύο προσβάσεις στη μνήμη.

3.3.2 Όριο μεγέθους στη στοίβα

Όπως αναφέρθηκε, το όριο της στοίβας κάθε προγράμματος eBPF είναι 512 bytes.
Κατά την αναζήτηση ενός στοιχείου στο Morton Filter, όταν πρέπει να βρεθεί το εναλλακτικό bucket του στοιχείου, κάθε στοιχείο αντιστοιχεί σε ένα OFF_RANGE, το οποίο επιτρέπει τα δύο buckets που αντιστοιχούν στο στοιχείο να βρίσκονται σε κοντινά Blocks. Το σύνολο των τιμών OFF_RANGE, για όλα τα στοιχεία, αρχικά βρισκόταν σε ένα πίνακα μη-αρνητικών 16-bit ακεραίων στο eBPF πρόγραμμα. Έτσι, όμως, η στοίβα είχε μέγεθος που ξεπερνούσε τα 512 bytes, και για αυτό το λόγο αυτός ο πίνακας αποθηκεύεται σε ένα eBPF χάρτη ο οποίος προσπελαύνεται όποτε πρέπει να υπολογιστεί το δευτερεύον bucket ενός στοιχείου, με επιπτώσεις στην απόδοση του μηχανισμού με το Morton Filter.

Κεφάλαιο 4: Περιγραφή και αξιολόγηση πειραμάτων

Σε αυτό το κεφάλαιο παρουσιάζεται η υποδομή που χρησιμοποιήθηκε για την εκτέλεση των πειραμάτων, περιγράφεται ο τρόπος διεξαγωγής τους και αναλύονται τα αποτελέσματα αυτών.

Στήθηκαν δύο εικονικά μηχανήματα (Virtual Machines - VMs) Linux, με τον πυρήνα 5.10, τα οποία επικοινωνούν μεταξύ τους μέσω εσωτερικής δικτύωσης (εικονικό τοπικό δίκτυο). Στην τελευταία μέτρηση, χρησιμοποιήθηκαν κάρτες δικτύου Netronome NFP-4000[50], οι οποίες υποστηρίζουν μεγαλύτερο ρυθμό αποστολής πακέτων. Ο driver αυτής της κάρτας υποστηρίζει τη λειτουργία Native XDP, καθώς και XDP Offloading. Η τελευταία λειτουργία δε δοκιμάστηκε στην παρούσα εργασία, και αφήνεται ως μελλοντική εργασία.

Εικονικό Μηχάνημα 1 - xdp-defense
Αυτό το μηχάνημα χρησιμοποιεί τη διανομή Debian Linux[51], με τον πυρήνα 5.10, διαθέτει έναν επεξεργαστικό πυρήνα και 8 GB φυσικής μνήμης. Εκτελεί την ένατη έκδοση του BIND[52], ως authoritative εξυπηρετητής, υπεύθυνος για τις ζώνες example.com, .se και .nu. Είναι το μηχάνημα στο οποίο εφαρμόζεται ο αμυντικός μηχανισμός που υλοποιήθηκε, σε λειτουργία Native XDP, καθώς ο driver virtio για τις εικονικές διεπαφές που χρησιμοποιούνται μεταφέρει την επεξεργασία στο χώρο του επόπτη (supervisor), και το εικονικό μηχάνημα επιβαρύνεται μόνο με την επεξεργασία των πακέτων που εγκρίνονται προς επεξεργασία από το πρόγραμμα eBPF[53].

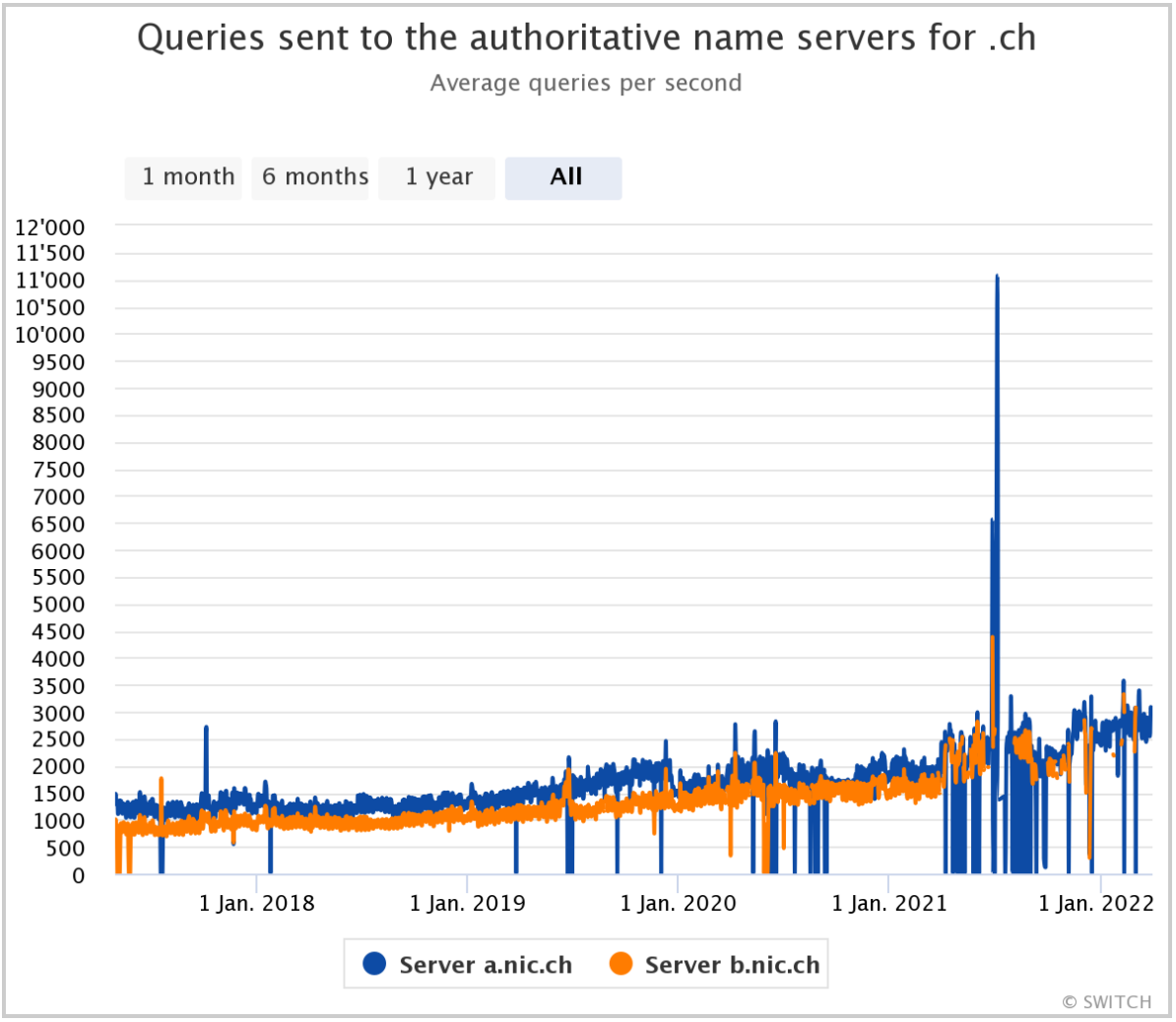
Εικονικό Μηχάνημα 2 - xdp-attack:
Αυτό το μηχάνημα χρησιμοποιεί τη διανομή Debian Linux[51], με τον πυρήνα 5.10, διαθέτει έναν επεξεργαστικό πυρήνα και 2 GB φυσικής μνήμης. Έγινε χρήση του εργαλείου tcprewrite[54] για την προετοιμασία των πακέτων προς αποστολή, με κατάλληλη ρύθμιση των διευθύνσεων IP, MAC, και επαναϋπολογισμό του checksum. Με το εργαλείο tcpreplay[55] στέλνουμε ταυτόχρονα, ρυθμίζοντας το ρυθμό και τη διάρκεια αποστολής, την καλόβουλη και κακόβουλη κίνηση στον authoritative εξυπηρετητή.

Πραγματοποιήθηκαν δύο ειδών επιθέσεις. Στο πρώτο είδος, η καλόβουλη κίνηση περιέχει ερωτήματα μόνο για τα ονόματα της ζώνης example.com (8303 ονόματα), και στις δομές έχουν φορτωθεί μόνο αυτά. Στο δεύτερο, η καλόβουλη κίνηση περιέχει ερωτήματα για ονόματα και από τις τρεις ζώνες για τις οποίες είναι υπεύθυνος ο authoritative εξυπηρετητής, και οι δομές περιέχουν όλα τα ονόματα (1657995 ονόματα). Η διαφοροποίηση στο πλήθος των ονομάτων έγινε για να εξεταστεί κατά πόσο οι δομές, στο χώρο πυρήνα, είναι αποδοτικές με πλήθος στοιχείων διαφορετικής τάξης μεγέθους.

Τέλος, πραγματοποιήθηκε μία επίθεση, χρησιμοποιώντας τις κάρτες δικτύου Netronome που προαναφέρθηκαν, με το πρώτο σύνολο ονομάτων (ονόματα της ζώνης example.com).

Ρυθμοί αποστολής καλόβουλης/κακόβουλης κίνησης
Ο ρυθμός αποστολής της καλόβουλης κίνησης τέθηκε, σε κάθε περίπτωση, ίσος με 2000 πακέτα/δευτερόλεπτο, καθώς, σύμφωνα με στατιστικά[56] δεδομένα από DNS

εξυπηρετητές του οργανισμού SWITCH, οι πραγματικοί ρυθμοί ερωτημάτων/δευτερόλεπτο είναι παρόμοιοι.



Σχήμα 4.1: Μέσος όρος αφίξεων ερωτημάτων DNS στους authoritative DNS εξυπηρετητές για τη ζώνη .ch[56]

Ο ρυθμός της κακόβουλης κίνησης στις εικονικές κάρτες δικτύου είχε εύρος τιμών από 10000 πακέτα/δευτερόλεπτο, μια τιμή ικανή να προκαλέσει μεγάλη απώλεια καλόβουλης κίνησης απουσία κάποιου αμυντικού μηχανισμού, μέχρι 50000 πακέτα/δευτερόλεπτο, με βήμα 10000, και η αποστολή της κακόβουλης κίνησης διήρκεσε 300 δευτερόλεπτα.

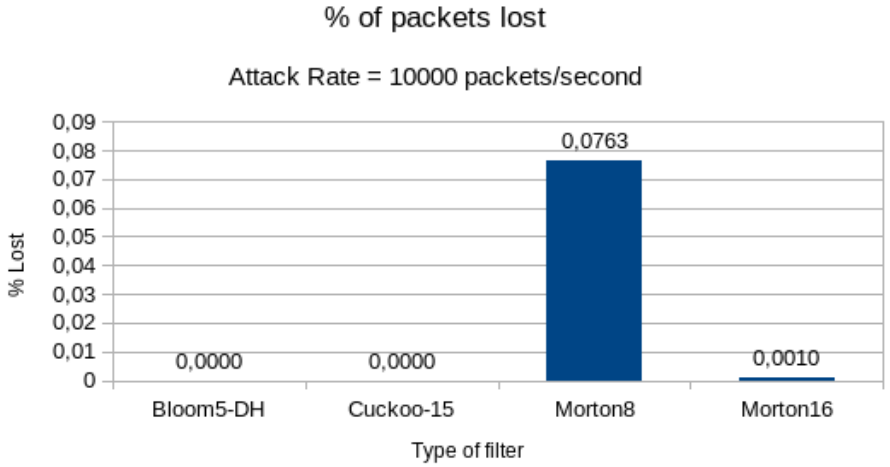
Κατά τη χρήση των καρτών δικτύου Netronome, η διάρκεια αποστολής της κακόβουλης κίνησης ήταν 100 δευτερόλεπτα, και ο ρυθμός αποστολής της κακόβουλης κίνησης ήταν 200000 πακέτα/δευτερόλεπτο. Αυτός ο ρυθμός δεν εξαντλεί το διαθέσιμο εύρος ζώνης των συγκεκριμένων καρτών, ωστόσο δεν ήταν δυνατή η αποστολή πολύ μεγαλύτερου ρυθμού, λόγω περιορισμού από το εργαλείο tcpreplay.

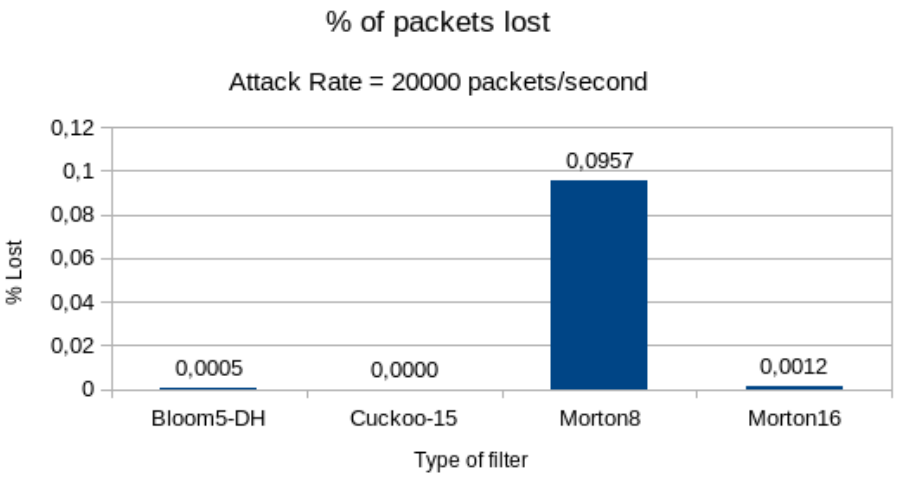
Συλλογή αποτελεσμάτων

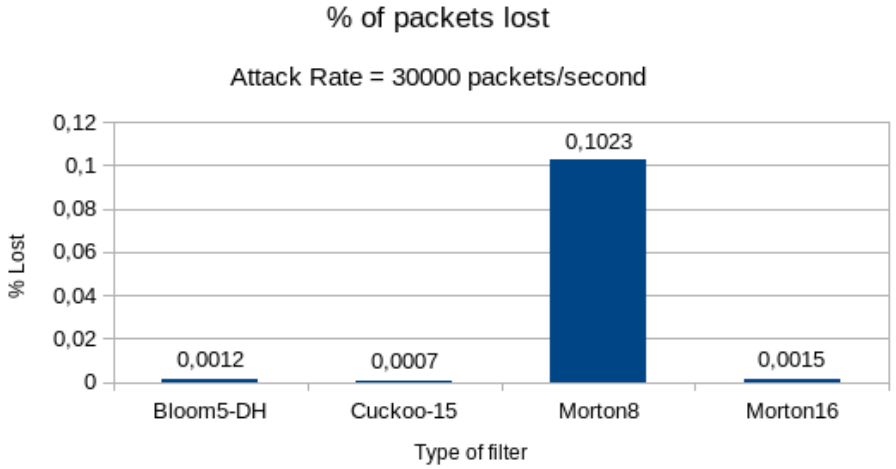
Για την καταγραφή των απαντήσεων του authoritative εξυπηρετητή χρησιμοποιήθηκε το εργαλείο tcpdump[57], και το Wireshark[58] για την ανάλυση και την εξαγωγή γραφημάτων. Η καταγραφή γίνεται με το φίλτρο “port 53 and src host *IP_address*”, όπου IP_address είναι η διεύθυνση IP του authoritative εξυπηρετητή. Καταγράφονται οι απαντήσεις στην καλόβουλη κίνηση, καθώς τα πακέτα της κακόβουλης κίνησης έχουν επεξεργαστεί έτσι ώστε η διεύθυνση IP του αποστολέα να μην αντιστοιχεί σε κάποιο μηχάνημα στο τοπικό δίκτυο των εικονικών μηχανημάτων. Αυτό δε σημαίνει ότι τα ερωτήματα της κακόβουλης κίνησης που, εσφαλμένα, θεωρούνται καλόβουλα (μετά από αναζήτηση του ονόματος στη πιθανοτική δομή, όπως περιγράφηκε παραπάνω), δεν επεξεργάζονται από το BIND, αλλά καταγράφονται οι απαντήσεις του εξυπηρετητή μόνο για τα καλόβουλα ερωτήματα.

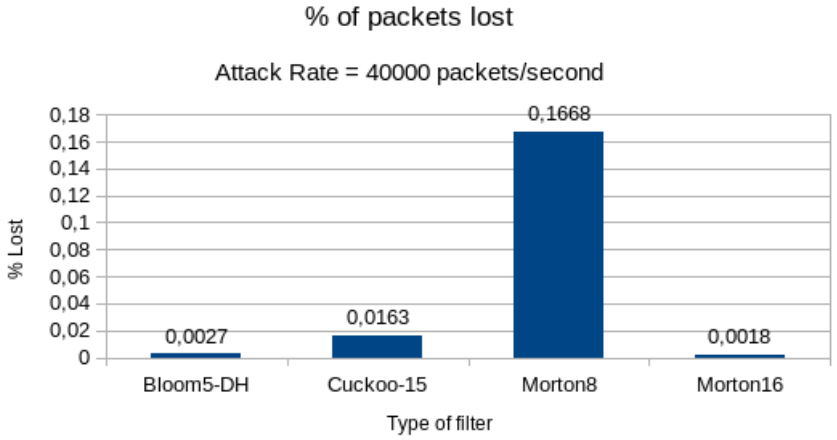
4.1 Αποτελέσματα πειράματος με ονόματα της ζώνης example.com

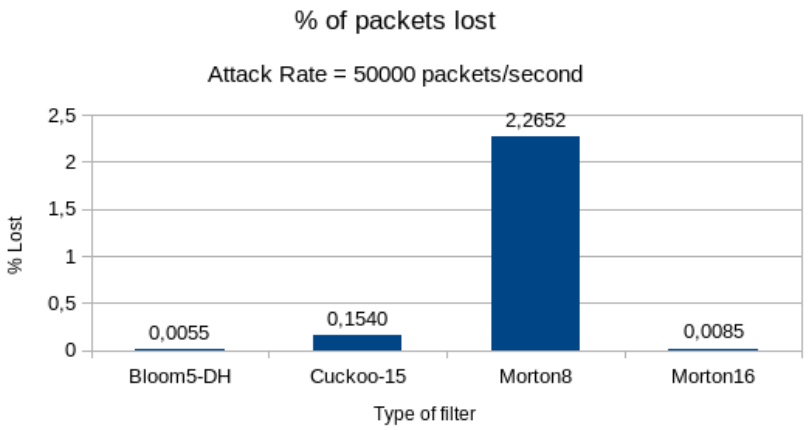
Στο πρώτο πείραμα οι πιθανοτικές δομές και η καλόβουλη κίνηση περιείχαν ονόματα της ζώνης example.com (8303 ονόματα). Στα παρακάτω διαγράμματα παρουσιάζονται τα ποσοστα χαμένων πακέτων, δηλαδή η διαφορά των απαντήσεων στην καλόβουλη κίνηση από τα συνολικά καλόβουλα ερωτήματα που στάλθηκαν ως προς το σύνολο της καλόβουλης κίνησης, στους διάφορους ρυθμούς αποστολής της κακόβουλης κίνησης.




% of packets lost
Attack Rate = 20000 packets/second
0,12
0,1
0,08
0,06
0,04
0,02
0
% Lost
0,0005
0,0000
0,0957
0,0012
Bloom5-DH
Cuckoo-15
Morton8
Morton16
Type of filter

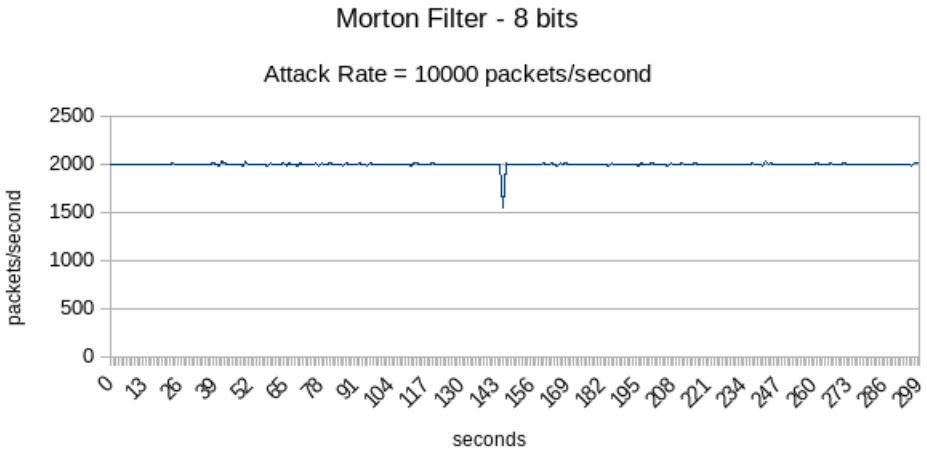

% of packets lost
Attack Rate = 30000 packets/second
0,12
0,1
0,08
0,06
0,04
0,02
0
% Lost
0,0012
0,0007
0,1023
0,0015
Bloom5-DH
Cuckoo-15
Morton8
Morton16
Type of filter

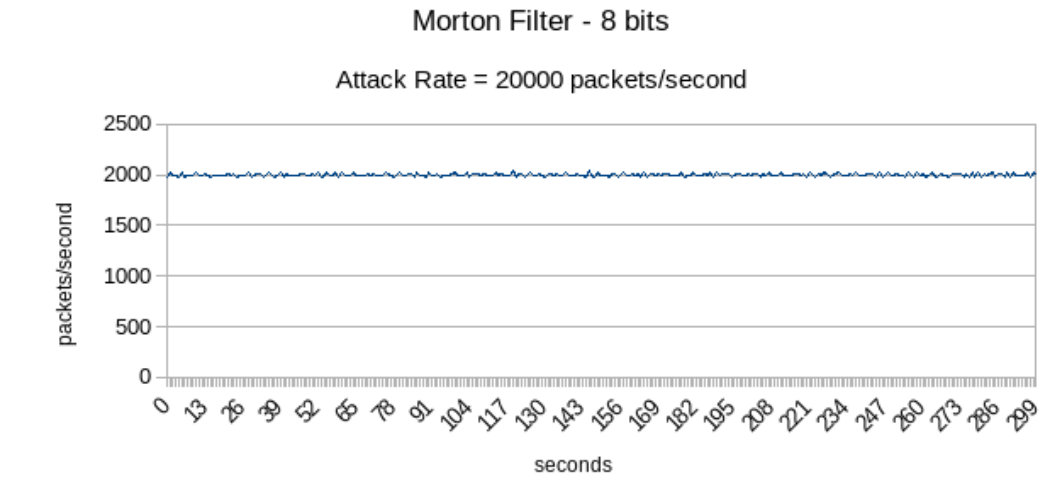

% of packets lost
Attack Rate = 40000 packets/second
0,18
0,16
0,14
0,12
0,1
0,08
0,06
0,04
0,02
0
% Lost
0,0027
0,0163
0,1668
0,0018
Bloom5-DH
Cuckoo-15
Morton8
Morton16
Type of filter

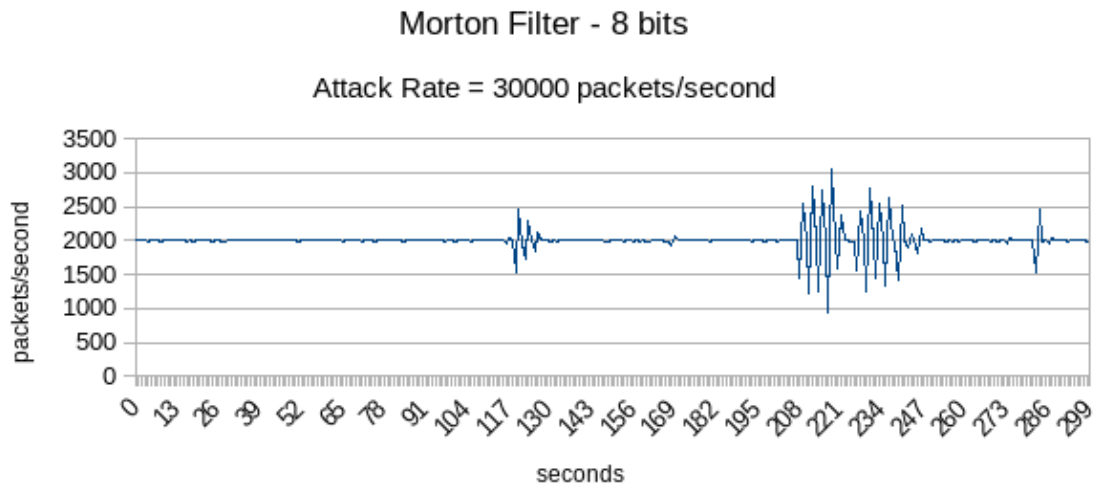


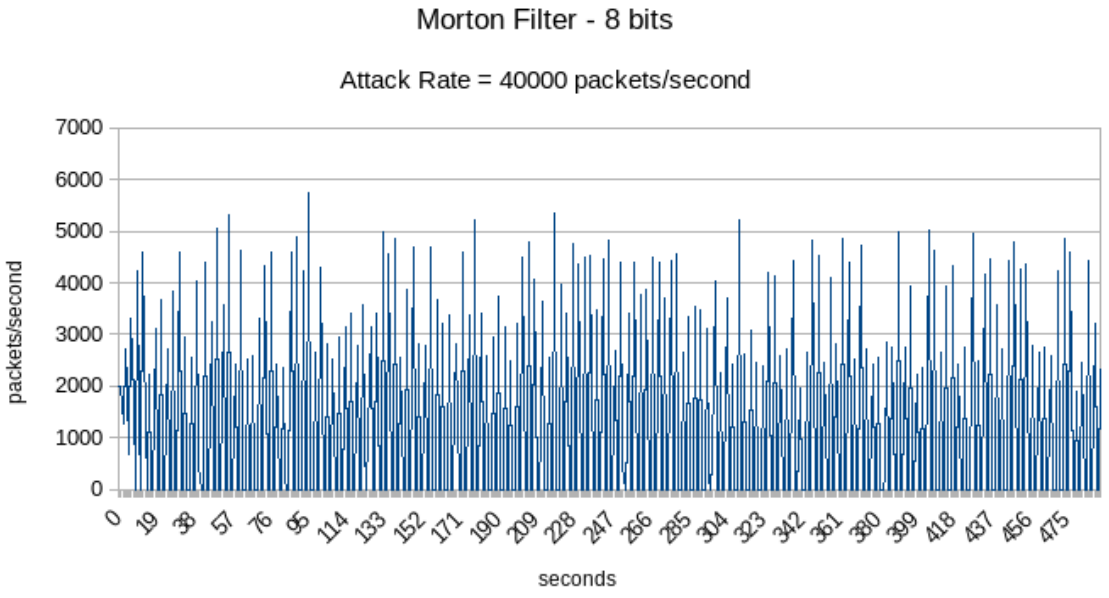
Σχήμα 4.2: Ποσοστά χαμένων καλόβουλων απαντήσεων επί της συνολικής καλόβουλης κίνησης για τους διάφορους ρυθμούς επίθεσης, στο πρώτο πείραμα.

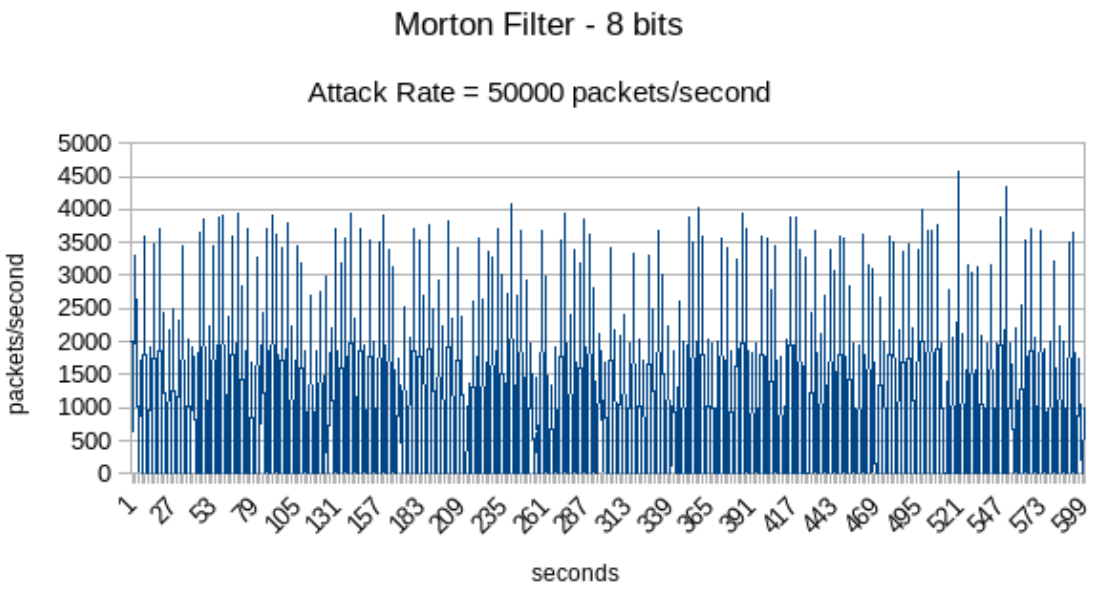
Φαίνεται ότι, καθώς αυξάνεται ο ρυθμός αποστολής της κακόβουλης κίνησης, το Morton Filter με μέγεθος αποτυπώματος 8 bit αδυνατεί να ανταποκριθεί, και αυξάνεται το πλήθος των καλόβουλων πακέτων που δεν εξυπηρετούνται. Αυτό γίνεται πιο σαφές στα επόμενα διαγράμματα, τα οποία δείχνουν το ρυθμό άφιξης πακέτων στον εξυπηρετητή - θύμα.




Morton Filter - 8 bits
Attack Rate = 20000 packets/second
packets/second
2500
2000
1500
1000
500
0
0 13 26 39 52 65 78 91 104 117 130 143 156 169 182 195 208 221 234 247 260 273 286 299
seconds


Morton Filter - 8 bits
Attack Rate = 30000 packets/second
packets/second
3500
3000
2500
2000
1500
1000
500
0
0 13 26 39 52 65 78 91 104 117 130 143 156 169 182 195 208 221 234 247 260 273 286 299
seconds


Morton Filter - 8 bits
Attack Rate = 40000 packets/second
packets/second
7000
6000
5000
4000
3000
2000
1000
0
0 19 38 57 76 95 114 133 152 171 190 209 228 247 266 285 304 323 342 361 380 399 418 437 456 475
seconds

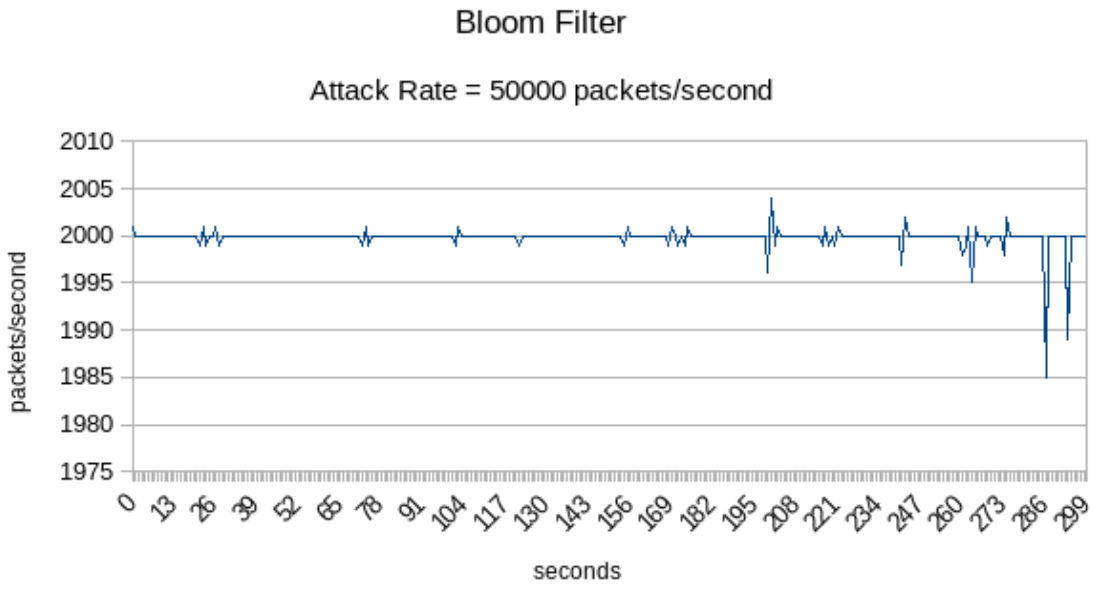


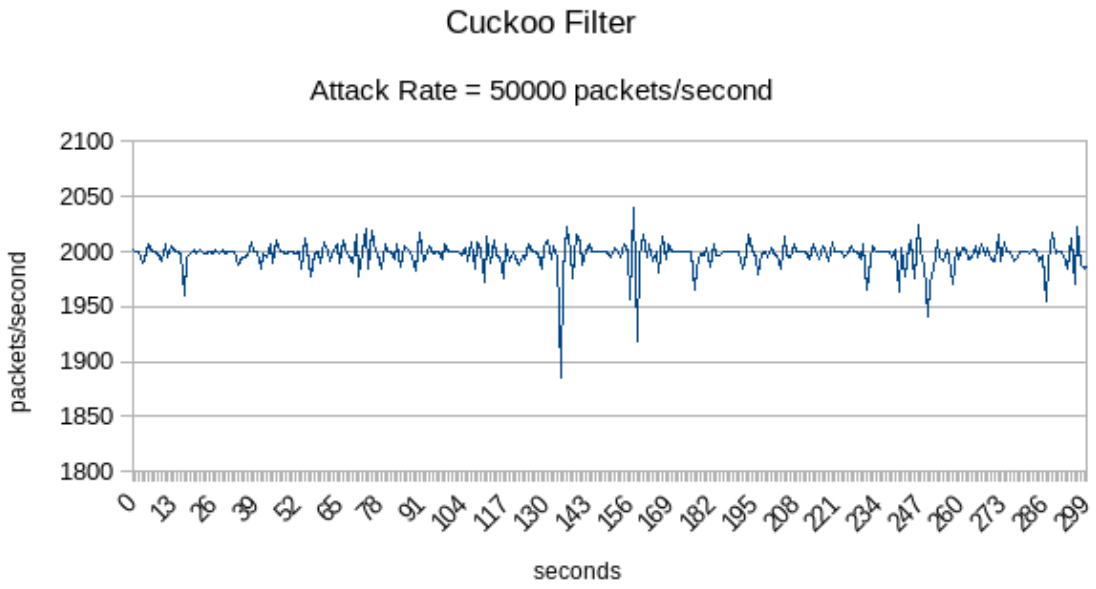
Σχήμα 4.3: Ρυθμοί άφιξης καλόβουλων απαντήσεων στο Morton Filter, με μέγεθος αποτυπώματος ίσο με 8 bit, για τους διάφορους ρυθμούς επίθεσης, στο πρώτο πείραμα.

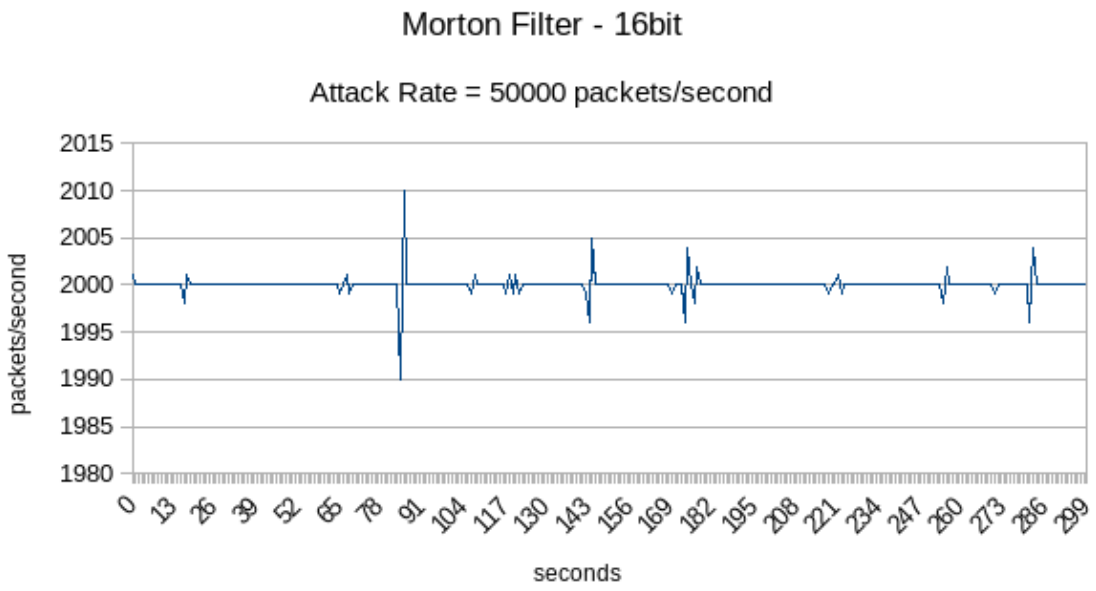
Όταν η επίθεση έχει ρυθμό πάνω από 30 χιλιάδες πακέτα το δευτερόλεπτο, έχουμε τόσες πολλές εσφαλμένως θετικές (false positive) αναζητήσεις στη δομή Morton Filter, ώστε τα πακέτα της καλόβουλης κίνησης αργούν να εξυπηρετηθούν. Στην περίπτωση των 50000 πακέτων/δευτερόλεπτο μάλιστα, κάνουν σχεδόν το διπλάσιο χρόνο, όπως φαίνεται από το σχήμα.

Αυτό συμβαίνει γιατί τα ονόματα που ζητούνται στα κακόβουλα πακέτα βρίσκονται στη δομή, άρα αντιστοιχίζονται στα ίδια buckets και έχουν το ίδιο αποτύπωμα με νόμιμα ονόματα. Συνεπώς, το μέγεθος των 8-bit για το αποτύπωμα ενός ονόματος δεν επαρκεί για να έχουμε αρκετά χαμηλό ρυθμό εύρεσης false positives, και αυτό έχει αντίκτυπο στην απόδοση του αμυντικού μηχανισμού.

Παρότι το ποσοστό 2.2% στις χαμένες καλόβουλες απαντήσεις δεν είναι υψηλό, ο διπλάσιος χρόνος εξυπηρέτησής τους είναι προβληματικός και δείχνει υποβάθμιση(degradation) της υπηρεσίας που προσφέρει ο εξυπηρετητής. Αυτή η συμπεριφορά δεν εμφανίζεται κατά τη χρήση των υπόλοιπων πιθανοτικών δομών, σύμφωνα και με το παρακάτω σχήμα:


Bloom Filter
Attack Rate = 50000 packets/second
packets/second
2010
2005
2000
1995
1990
1985
1980
1975
0
13
26
39
52
65
78
91
104
117
130
143
156
169
182
195
208
221
234
247
260
273
286
299
seconds

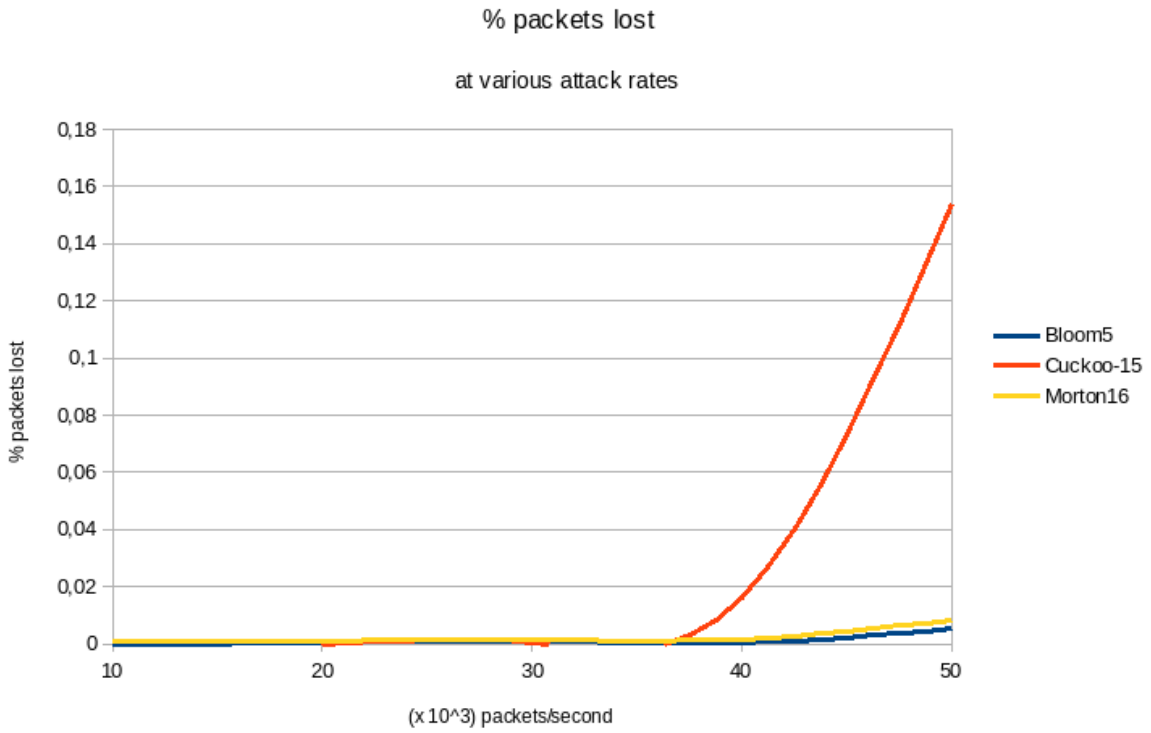

Cuckoo Filter
Attack Rate = 50000 packets/second
packets/second
2100
2050
2000
1950
1900
1850
1800
0
13
26
39
52
65
78
91
104
117
130
143
156
169
182
195
208
221
234
247
260
273
286
299
seconds



Σχήμα 4.4: Ρυθμοί άφιξης καλόβουλων απαντήσεων στις δομές Bloom Filter, Cuckoo Filter και Morton Filter (με μέγεθος αποτυπώματος 16-bit), με ρυθμό επίθεσης 50000 πακέτα/δευτερόλεπτο, στο πρώτο πείραμα

Ο ρυθμός άφιξης των καλόβουλων απαντήσεων δεν παρουσιάζει ιδιαίτερες μεταβολές, και τα ερωτήματα εξυπηρετούνται χωρίς καθυστέρηση. Αυτό δείχνει ότι ο αμυντικός μηχανισμός που υλοποιήθηκε με αυτές τις δομές αντιμετωπίζει επιτυχώς την επίθεση.

Παρατίθεται το διάγραμμα με τα ποσοστά απώλειας καλόβουλων απαντήσεων για τις τρεις τελευταίες δομές:

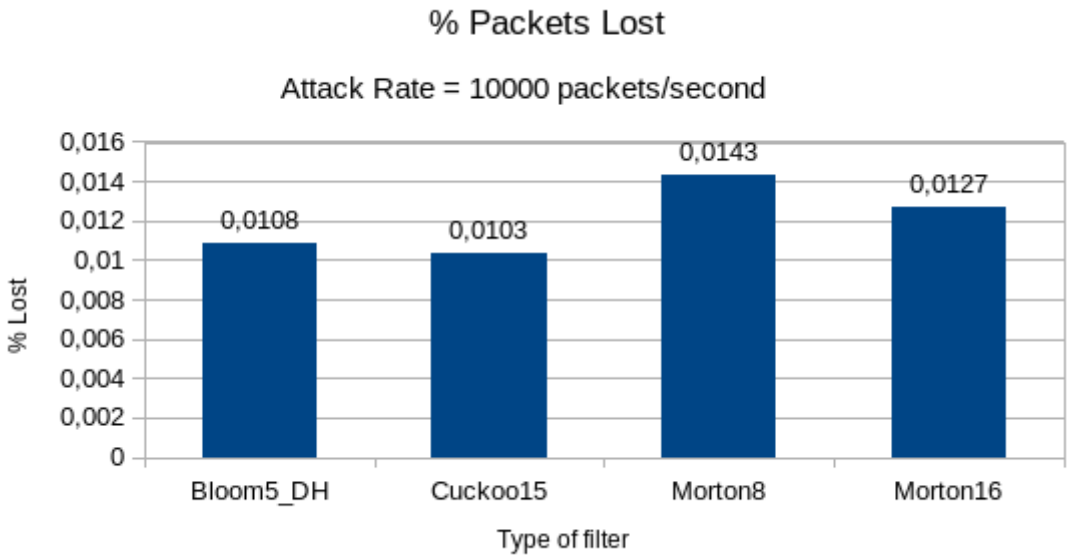


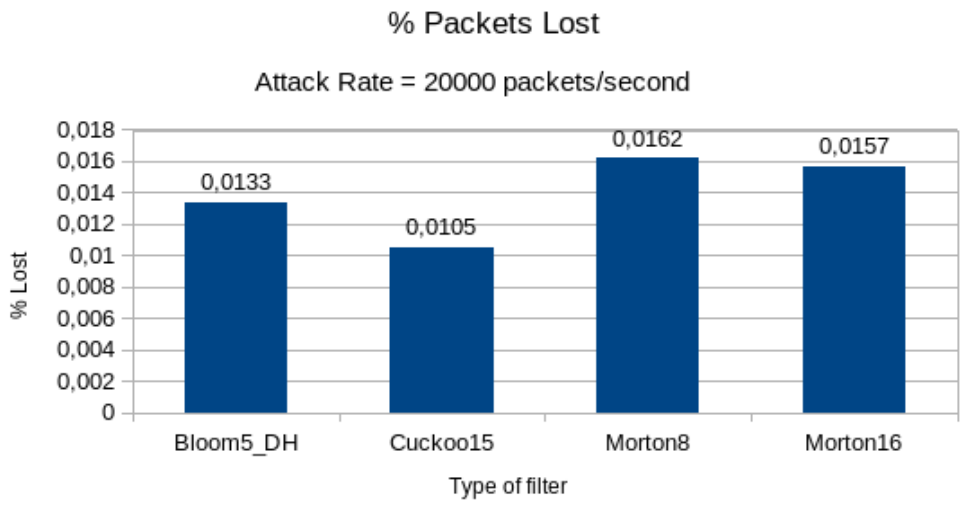
Σχήμα 4.5: Ποσοστά απώλειας καλόβουλων απαντήσεων για τις δομές Bloom Filter, Cuckoo Filter και Morton Filter, με μέγεθος αποτυπώματος 16 bit, για τους διάφορους ρυθμούς επίθεσης, στο πρώτο πείραμα.

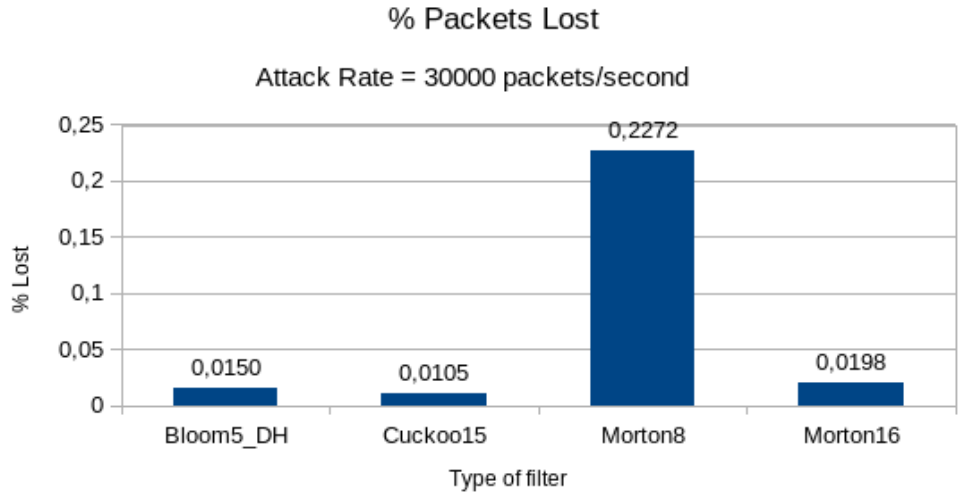
Ενώ τα ποσοστά απώλειας καλόβουλων απαντήσεων δεν είναι υψηλά για καμία δομή, το ποσοστό απώλειας αυξάνεται απότομα για τη δομή Cuckoo Filter, χωρίς προφανή λόγο, ενώ για τις άλλες δύο δομές δεν παρουσιάζει τέτοια μεταβολή. Αυτό μπορεί να συμβαίνει γιατί, κατά τη δημιουργία της δομής και την εισαγωγή των στοιχείων στη δομή Cuckoo Filter, μπορεί να έγιναν μετατοπίσεις στοιχείων. Επειδή το ποιο στοιχείο θα μετατοπιστεί επιλέγεται τυχαία, είναι πιθανό οι μετατοπίσεις να έγιναν με τέτοιο τρόπο που να προκάλεσαν αυτή την αύξηση στο ποσοστό απώλειας, δηλαδή για το συγκεκριμένο σύνολο ονομάτων, να έγιναν πολλές επανατοποθετήσεις και άρα τα περισσότερα στοιχεία να βρίσκονται στο δευτερεύον bucket τους. Άρα η αναζήτηση ονομάτων στη δομή παίρνει, κατά μέσο όρο, περισσότερο χρόνο, καθώς ελέγχονται, κατά μέσο όρο, περισσότερα αποτυπώματα ανά αναζήτηση.

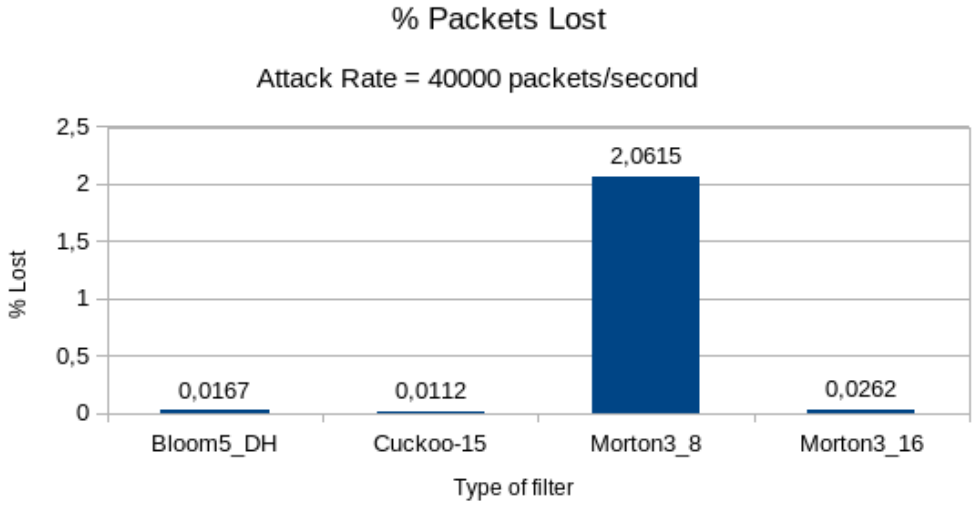
4.2 Αποτελέσματα πειράματος με ονόματα των ζωνών example.com, .se και .nu

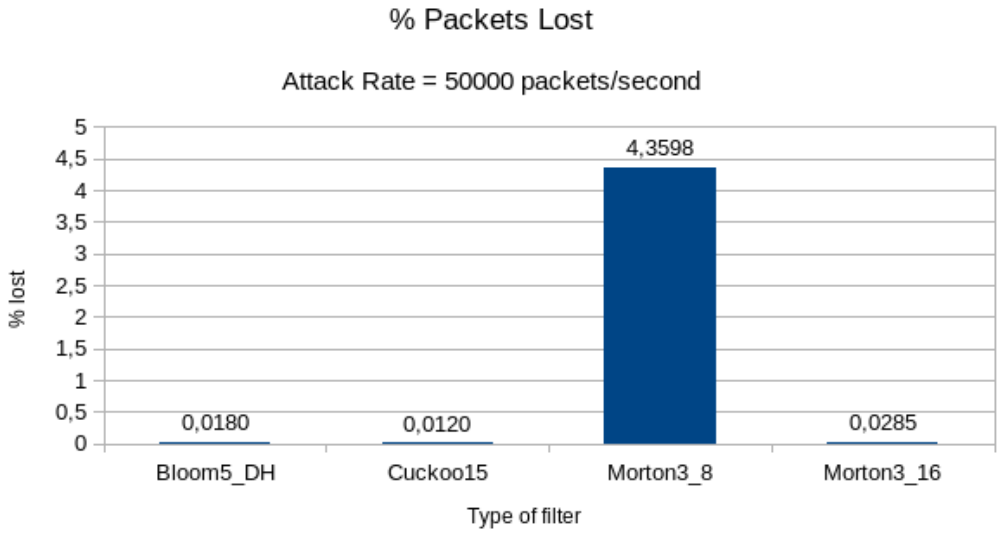
Στο δεύτερο πείραμα οι πιθανοτικές δομές και η καλόβουλη κίνηση περιείχε ονόματα από τις τρεις ζώνες example.com, .se και .nu (1657995 ονόματα). Ακολουθούν τα αντίστοιχα σχήματα για την απώλεια καλόβουλων απαντήσεων:




% Packets Lost
Attack Rate = 20000 packets/second
0,018
0,016
0,014
0,012
0,01
0,008
0,006
0,004
0,002
0
% Lost
0,0133
0,0105
0,0162
0,0157
Bloom5_DH
Cuckoo15
Morton8
Morton16
Type of filter

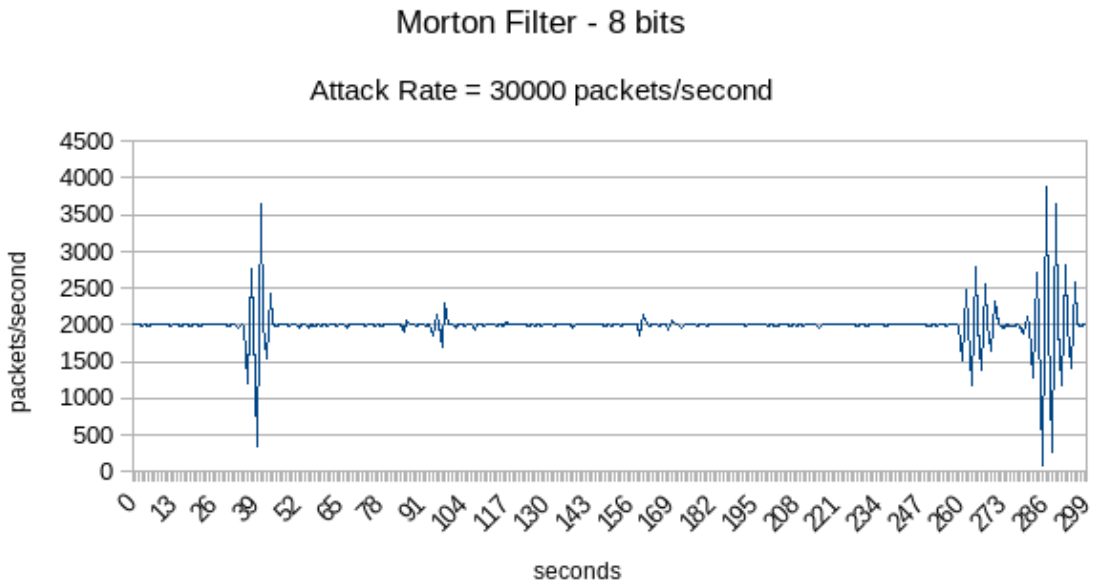

% Packets Lost
Attack Rate = 30000 packets/second
0,25
0,2
0,15
0,1
0,05
0
% Lost
0,0150
0,0105
0,2272
0,0198
Bloom5_DH
Cuckoo15
Morton8
Morton16
Type of filter

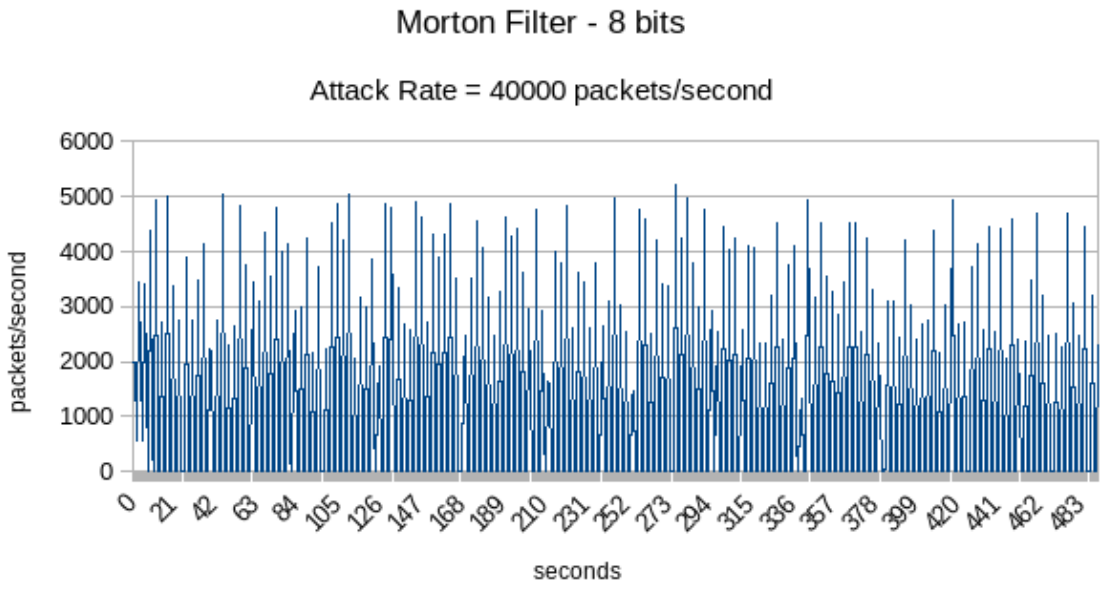

% Packets Lost
Attack Rate = 40000 packets/second
2,5
2
1,5
1
0,5
0
% Lost
0,0167
0,0112
2,0615
0,0262
Bloom5_DH
Cuckoo-15
Morton3_8
Morton3_16
Type of filter



Σχήμα 4.6: Ποσοστά χαμένων καλόβουλων απαντήσεων επί της συνολικής καλόβουλης κίνησης, κατά τους διάφορους ρυθμούς επίθεσης, στο δεύτερο πείραμα

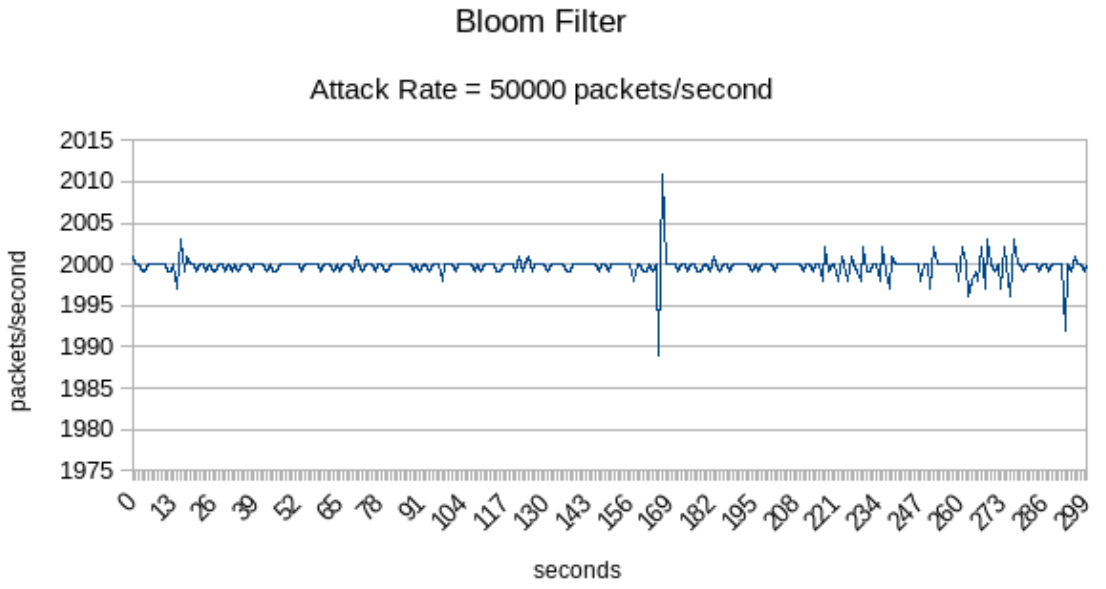
Τα αποτελέσματα είναι παρόμοια με το πρώτο πείραμα, με ελαφρώς αυξημένα ποσοστά χαμένων καλόβουλων απαντήσεων. Στο παρακάτω σχήμα φαίνεται πως, όταν ο αμυντικός μηχανισμός χρησιμοποιεί τη δομή Morton Filter με μέγεθος αποτυπώματος 8 bit, συνεχίζει να αδυνατεί να περιορίσει την επίθεση, όταν ο ρυθμός αποστολής των κακόβουλων πακέτων ξεπεράσει τα 30000 πακέτα το δευτερόλεπτο.

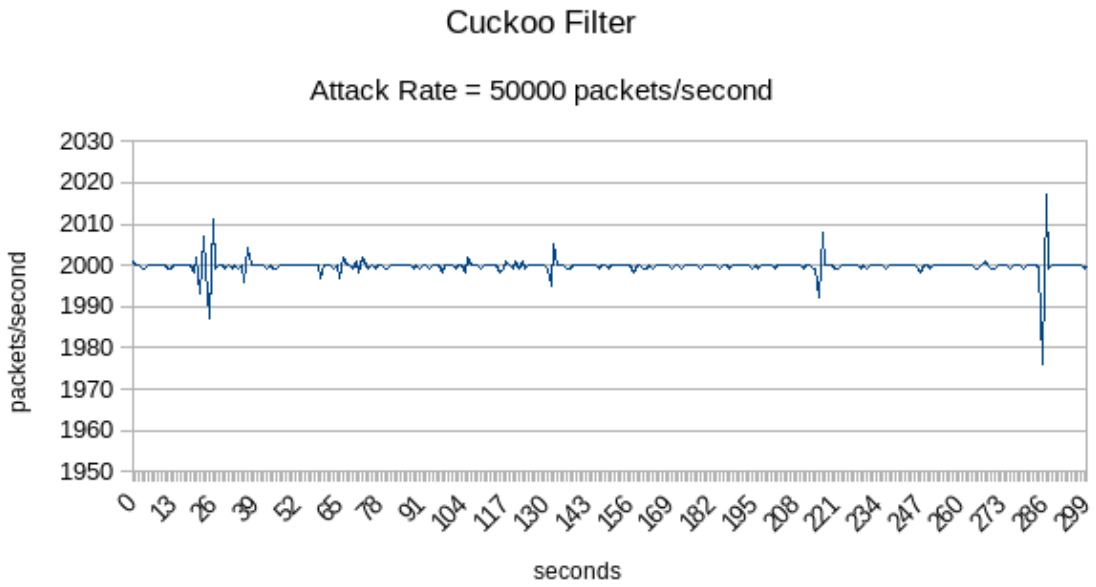


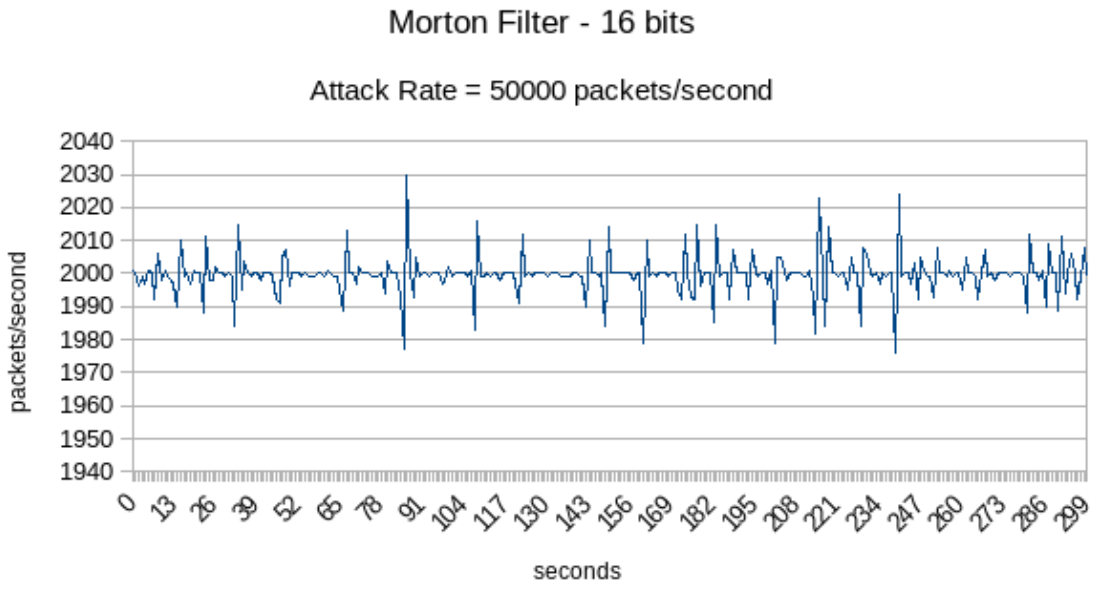


Σχήμα 4.7: Ρυθμοί άφιξης καλόβουλων απαντήσεων στο Morton Filter, με μέγεθος αποτυπώματος ίσο με 8 bit, για ρυθμούς επίθεσης ίσους με 30000 και 40000 πακέτα/δευτερόλεπτο, στο δεύτερο πείραμα.

Αντίθετα, χρησιμοποιώντας τις δομές Bloom Filter, Cuckoo Filter και Morton Filter, με μέγεθος αποτυπώματος 16 bit, ο μηχανισμός αντιμετωπίζει αποτελεσματικά την κίνηση με ρυθμό αποστολής μέχρι και 50000 πακέτα το δευτερόλεπτο.

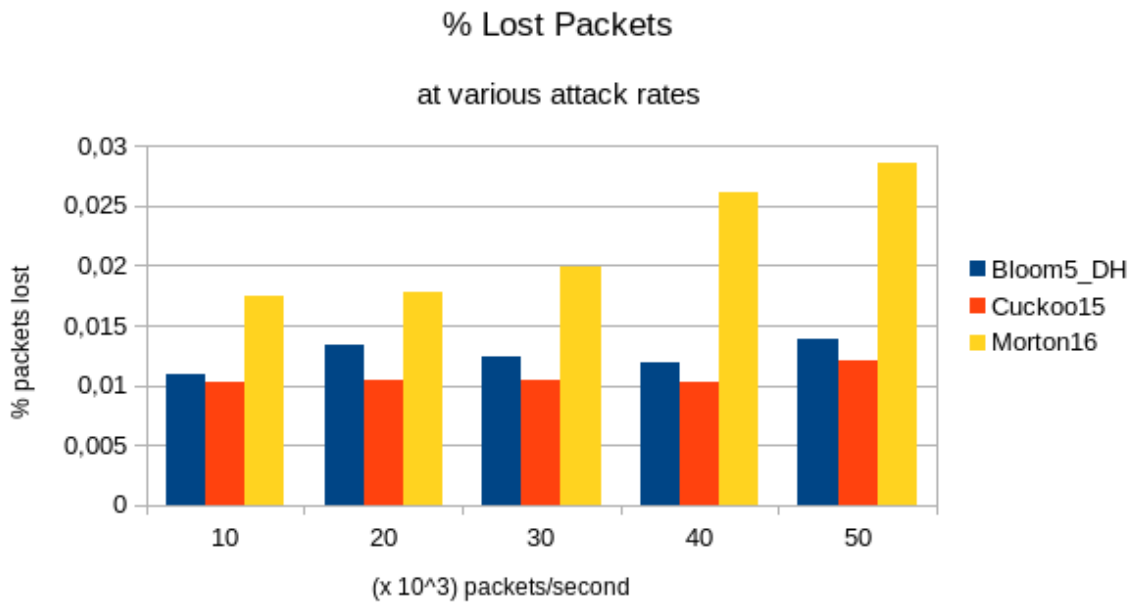






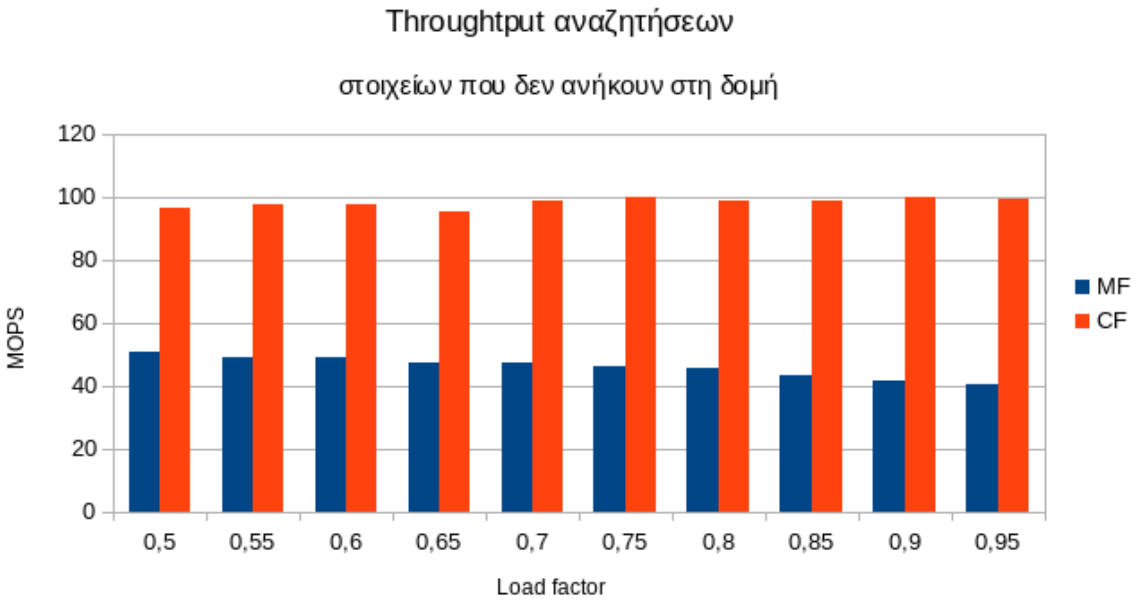
Σχήμα 4.8: Ρυθμοί άφιξης καλόβουλων απαντήσεων στις δομές Bloom Filter, Cuckoo Filter και Morton Filter (με μέγεθος αποτυπώματος 16-bit), με ρυθμό επίθεσης 50000 πακέτα/δευτερόλεπτο, στο δεύτερο πείραμα

Ακολουθεί το διάγραμμα με τα ποσοστά χαμένων καλόβουλων απαντήσεων για τις προαναφερθείσες δομές:



Σχήμα 4.9: Ποσοστά απώλειας καλόβουλων απαντήσεων για τις δομές Bloom Filter, Cuckoo Filter και Morton Filter (με μέγεθος αποτυπώματος 16 bit) για τους διάφορους ρυθμούς επίθεσης, δεύτερο πείραμα.

Τα ποσοστά απώλειας καλόβουλων απαντήσεων δείχνουν ότι η επίθεση αντιμετωπίζεται από τον αμυντικό μηχανισμό. Το γεγονός ότι η δομή Morton Filter έχει ελαφρώς αυξημένα ποσοστά απώλειας πακέτων είναι αναμενόμενο, καθώς, έχοντας εκτελέσει ένα benchmark στο χώρο χρήστη του εξυπηρετητή, το Cuckoo Filter έχει υψηλότερο αριθμό αναζητήσεων ανά δευτερόλεπτο από το Morton Filter, όταν τα στοιχεία που εισάγονται στη δομή είναι λιγότερα από $2^{23} = 8388608$[43, Sec. 7.5], άρα και μικρότερο χρόνο αναζήτησης.

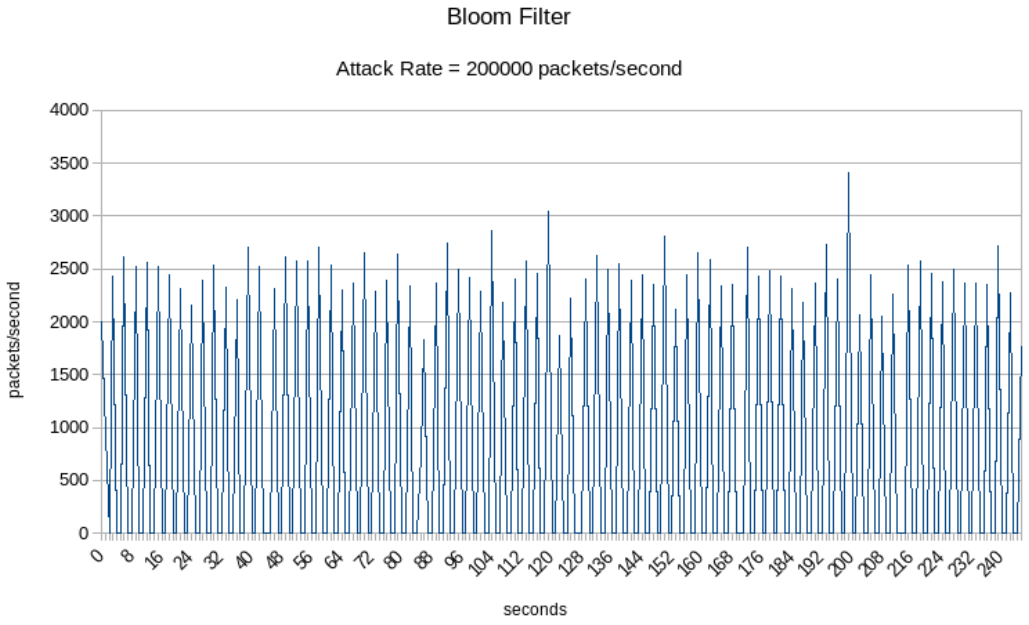


Σχήμα 4.10: Αριθμός αναζητήσεων (x 10^6) το δευτερόλεπτο, για στοιχεία που δεν έχουν εισαχθεί, στις δομές CF (Cuckoo Filter) και MF(Morton Filter), σε userspace benchmark στον εξυπηρετητή. Ο

αριθμός των στοιχείων που έχουν εισαχθεί στις δομές είναι 128x1024x16 = 2097152 = 2^{21}, ίδιας τάξης μεγέθους με το πλήθος των ονομάτων στις ζώνες του εξυπηρετητή.

4.3 Αποτελέσματα πειράματος στις κάρτες Netronome NFP-4000

Ακολουθούν τα διαγράμματα με τους ρυθμούς άφιξης καλόβουλων απαντήσεων, για τις δομές Bloom Filter, Cuckoo Filter και Morton Filter:



Σχήμα 4.11: Ρυθμοί άφιξης καλόβουλων απαντήσεων στις δομές Bloom Filter, Cuckoo Filter και Morton Filter, κατά τη μέτρηση στις κάρτες Netronome

Οι δομές Bloom Filter και Cuckoo Filter αδυνατούν να αντιμετωπίσουν την επίθεση, καθώς ο ρυθμός εξυπηρέτησης των ερωτημάτων δεν είναι σταθερός και υπάρχει μεγάλη καθυστέρηση (η διάρκεια αποστολής της κακόβουλης κίνησης ήταν 100 δευτερόλεπτα). Αντίθετα, η δομή Morton Filter αντιμετωπίζει επιτυχώς την επίθεση.

Αυτό μπορεί να εξηγηθεί με τον τρόπο που γίνονται οι αναζητήσεις στις δομές, και πόσες, κατά μέσο όρο, προσβάσεις γίνονται σε μία αναζήτηση.

Όπως προαναφέρθηκε, η πλειονότητα των αναζητήσεων στις δομές αφορούν στοιχεία που δεν περιμένουμε να βρεθούν στη δομή. Για τις δομές που χρησιμοποιήθηκαν, αυτό μεταφράζεται ως εξής:

- Στο Bloom Filter, γίνονται μέχρι 5 προσβάσεις στη μνήμη, και 5 συγκρίσεις, μέχρι να βρεθεί το πρώτο μηδενικό που δείχνει ότι το στοιχείο δεν υπάρχει στη δομή.
- Στο Cuckoo Filter, γίνονται 2 προσβάσεις στη μνήμη, και 8 συγκρίσεις, μέχρι να διαβαστούν τα 8 αποτυπώματα, και να μη βρεθεί το ζητούμενο στοιχείο.
- Στο Morton Filter, γίνονται το πολύ 2 προσβάσεις στη μνήμη, στα δύο Block που αντιστοιχούν στο στοιχείο. Κατά την αναζήτηση του στοιχείου σε κάθε Block, έχουμε ανοιγμένα (unrolled) for-loops, και πολλά if-blocks, άρα συγκρίσεις, για να ερμηνεύσουμε τα κομμάτια FCA (Fullness Counter Array) και OTA (Overflow Tracking Array) του Block, τα οποία είναι απαραίτητα για την εύρεση του στοιχείου στο Block.

Όπως εξηγήθηκε στην ενότητα 2.5.3, το Morton Filter είναι σχεδιασμένο ώστε τα περισσότερα στοιχεία του να βρίσκονται στα πρωτεύοντα bucket τους. Αυτό σημαίνει ότι τα αντίστοιχα bits στον πίνακα OTA, είναι ίσα με 0. Συνεπώς, κατά την αναζήτηση στοιχείων που δεν υπάρχουν στη δομή, και μετά την ανάγνωση των δικών τους αντίστοιχων bit στον πίνακα OTA, τα οποία είναι 0, δεν είναι απαραίτητη η πρόσβαση στη μνήμη και η ανάγνωση του δευτερεύοντος Block, και η δομή επιστρέφει αρνητική απάντηση, έχοντας εξετάσει μόνο το πρωτεύον Block του κάθε στοιχείου.

Συνεπώς, σε υψηλότερους ρυθμούς αποστολής της κακόβουλης κίνησης, φαίνεται ότι το πλήθος των προσβάσεων στη μνήμη επηρεάζουν περισσότερο την απόδοση του μηχανισμού, από τα εκτεταμένα unrolled for-loops και if-blocks.

Κεφάλαιο 5: Συμπεράσματα και μελλοντική εργασία

Σε αυτό το κεφάλαιο παρουσιάζονται τα συμπεράσματα της εργασίας καθώς και προτάσεις για την επέκταση του αμυντικού μηχανισμού, ως μελλοντική εργασία.

5.1 Συμπεράσματα

Στα πλαίσια της παρούσας διπλωματικής εργασίας μελετήθηκε η επίθεση DNS Water Torture και αναπτύχθηκε ένας μηχανισμός αντιμετώπισής της σε authoritative εξυπηρετητή, στο επίπεδο δεδομένων. Ο μηχανισμός χρησιμοποιεί τις πιθανοτικές δομές δεδομένων Bloom Filter, Cuckoo Filter και Morton Filter για να κρίνει αν το όνομα που ζητείται σε ένα ερώτημα DNS υπάρχει στα αρχεία ζώνης του εξυπηρετητή. Αν υπάρχει, το ερώτημα εξυπηρετείται κανονικά, ειδάλλως απορρίπτεται. Οι δομές υλοποιήθηκαν σε eBPF και ο μηχανισμός προσαρτήθηκε στο XDP hook για να επιτευχθεί υψηλός ρυθμός επεξεργασίας πακέτων.

Σύμφωνα με τα αποτελέσματα, σε χαμηλότερους ρυθμούς αποστολής της κακόβουλης κίνησης, ο μηχανισμός αντιμετωπίζει αποτελεσματικά τέτοιο είδος επίθεσης, εκτός από την περίπτωση που χρησιμοποιεί τη δομή Morton Filter με μέγεθος αποτυπώματος ίσο με 8 bit, καθώς ο ρυθμός με τον οποίο εξυπηρετούνται ερωτήματα που αφορούν έγκυρα ονόματα δεν παρουσιάζει σημαντικές αυξομειώσεις κατά τη διάρκεια της επίθεσης.

Αντίθετα, σε υψηλότερο ρυθμό αποστολής της κακόβουλης κίνησης, ο μηχανισμός αντιμετωπίζει την επίθεση αποτελεσματικά με τη δομή Morton Filter, με μέγεθος αποτυπώματος 16 bit, και όχι με τις δομές Bloom Filter και Cuckoo Filter.

5.2 Μελλοντικές επεκτάσεις

Ο μηχανισμός υλοποιήθηκε σε eBPF και προσαρτήθηκε στο XDP hook. Θα μπορούσε να υλοποιηθεί στη γλώσσα P4[28] ή στο Data Plane Development Kit (DPDK)[27].

Επίσης, ο μηχανισμός θα μπορούσε να υλοποιηθεί κάνοντας χρήση άλλων πιθανοτικών δομών δεδομένων, όπως το Ribbon Filter[59], Vacuum Filter[60] ή Xor Filter[61].

Μπορούν να γίνουν μετρήσεις σε υψηλότερους ρυθμούς αποστολής της κακόβουλης κίνησης, χρησιμοποιώντας εργαλεία διαφορετικά του tcpreplay, όπως το framework PF RING Zero Copy[62].

Ακόμα, ο μηχανισμός μπορεί να προσαρμοστεί για να αντιμετωπίσει άλλου είδους επιθέσεις DDoS.

Τέλος, ο μηχανισμός θα μπορούσε να προσαρτηθεί με τρόπο λειτουργίας Offloaded XDP σε κάρτα δικτύου που το υποστηρίζει, όπως η NVIDIA Mellanox ConnectX-5[63] ή Netronome NICs[64].

Βιβλιογραφία

[1] “Kurose και Ross - 2013 - Computer networking a top-down approach.pdf.” Accessed: Jun. 14, 2022. [Online]. Available: https://eclass.teicrete.gr/modules/document/file.php/TP326/%CE%98%CE%B5%CF%89%CF%81%CE%AF%CE%B1%20(Lectures)/Computer_Networking_A_Top-Down_Approach.pdf

[2] “Root Server Technical Operations Association.” https://root-servers.org/ (accessed Jun. 14, 2022).

[3] “What is DNS Hierarchy Architecture with Examples (Explained),” *Cloud Infrastructure Services*, Nov. 29, 2021. https://cloudinfrastructureservices.co.uk/what-is-dns-hierarchy/ (accessed Jun. 17, 2022).

[4] “RFC 1035 - Domain names - implementation and specification.” Accessed: Jun. 14, 2022. [Online]. Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc1035.txt.pdf

[5] “Introduction to DNS Privacy,” *Internet Society*. https://www.internetsociety.org/resources/deploy360/dns-privacy/intro/ (accessed Jun. 24, 2022).

[6] “Stallings - 2006 - Cryptography and network security principles and .pdf.” Accessed: Jun. 24, 2022. [Online]. Available: http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5680/material-cripto-seg/2014-1/Stallings/Stallings_Cryptography_and_Network_Security.pdf

[7] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, “DNS Security Introduction and Requirements,” Internet Engineering Task Force, Request for Comments RFC 4033, Nov. 2005. doi: 10.17487/RFC4033.

[8] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. E. Hoffman, “Specification for DNS over Transport Layer Security (TLS),” Internet Engineering Task Force, Request for Comments RFC 7858, Feb. 2016. doi: 10.17487/RFC7858.

[9] P. E. Hoffman and P. McManus, “DNS Queries over HTTPS (DoH),” Internet Engineering Task Force, Request for Comments RFC 8484, Jul. 2018. doi: 10.17487/RFC8484.

[10] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis, and S. Gritzalis, “DNS amplification attack revisited,” *Comput. Secur.*, vol. 39, pp. 475–485, Nov. 2013, doi: 10.1016/j.cose.2013.10.001.

[11] “Endpoint Protection - Symantec Enterprise.” Accessed: Jun. 24, 2022. [Online]. Available: https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=9d18fc06-b229-4c4a-8ca5-7386d0870c01&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments

[12] “DNS Reflection, Amplification, & DNS Water-torture,” *https://www.akamai.com/site/en/documents/research-paper/dns-reflection-vs-dns-mirai-technical-publication.pdf*. https://www.akamai.com/site/en/documents/research-paper/dns-reflection-vs-dns-mirai-technical-publication.pdf (accessed Jun. 24, 2022).

[13] “What is DNS cache poisoning? | DNS spoofing,” *Cloudflare*. https://www.cloudflare.com/learning/dns/dns-cache-poisoning/ (accessed Jun. 24, 2022).

[14] “DNSSEC World Map.” https://stats.labs.apnic.net/dnssec (accessed Jun. 28, 2022).

[15] X. Luo, L. Wang, Z. Xu, K. Chen, J. Yang, and T. Tian, “A Large Scale Analysis of DNS Water Torture Attack,” in *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*, New York, NY, USA, Sep. 2018, pp. 168–173. doi: 10.1145/3297156.3297272.

[16] L. Fang, H. Wu, K. Qian, W. Wang, and L. Han, “A Comprehensive Analysis of DDoS

attacks based on DNS," *J. Phys. Conf. Ser.*, vol. 2024, p. 012027, Sep. 2021, doi: 10.1088/1742-6596/2024/1/012027.
[17] M. A. M. Vieira, M. S. Castanho, R. D. G. Pacífico, E. R. S. Santos, E. P. M. Câmara, and L. F. M. Vieira, "Fast packet processing with EBPF and XDP: Concepts, code, challenges, and applications," *ACM Comput. Surv.*, vol. 53, no. 1, Feb. 2020, doi: 10.1145/3371038.
[18] "bpf-usenix93.pdf." Accessed: Jun. 13, 2022. [Online]. Available: https://www.tcpdump.org/papers/bpf-usenix93.pdf
[19] "bpf(2) - Linux manual page." https://man7.org/linux/man-pages/man2/bpf.2.html (accessed Jun. 14, 2022).
[20] "eBPF verifier — The Linux Kernel documentation." https://www.kernel.org/doc/html/latest/bpf/verifier.html (accessed Jun. 20, 2022).
[21] "bpf-helpers(7) - Linux manual page." https://man7.org/linux/man-pages/man7/bpf-helpers.7.html (accessed Jun. 13, 2022).
[22] "Bounded loops in BPF for the 5.3 kernel [LWN.net]." https://lwn.net/Articles/794934/ (accessed Jul. 01, 2022).
[23] *BPF/libbpf usage and questions*. libbpf, 2022. Accessed: Jun. 13, 2022. [Online]. Available: https://github.com/libbpf/libbpf
[24] T. Høiland-Jørgensen, *The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel*. 2022. Accessed: Jun. 13, 2022. [Online]. Available: https://github.com/tohojo/xdp-paper/blob/672dc5ad9022229ead875c605e1ff030dee02bee/benchmarks/bench01_baseline.org
[25] "bpf.h « linux « uapi « include - kernel/git/stable/linux.git - Linux kernel stable tree." https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/include/uapi/linux/bpf.h?h=v5.10.123 (accessed Jun. 20, 2022).
[26] "presentation-lpc2018-xdp-tutorial.pdf." Accessed: Jun. 14, 2022. [Online]. Available: https://indico.lpc.events/event/2/contributions/71/attachments/17/9/presentation-lpc2018-xdp-tutorial.pdf
[27] "Home - DPDK." https://www.dpdk.org/ (accessed Jun. 13, 2022).
[28] P. Bosshart *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Comput. Commun. Rev.*, Jul. 2014, doi: 10.1145/2656877.2656890.
[29] "[GIT] Networking - David Miller." https://lore.kernel.org/lkml/20160727.010753.2221383279830501569.davem@davemloft.net/ (accessed Jun. 07, 2022).
[30] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970, doi: 10.1145/362686.362692.
[31] D. Katsaros, "Bloom Filter's False Positives Rate," p. 4.
[32] D. Guo, Y. Liu, X. Li, and P. Yang, "False Negative Problem of Counting Bloom Filter," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 5, pp. 651–664, Feb. 2010, doi: 10.1109/TKDE.2009.209.
[33] M. Mitzenmacher, "Compressed Bloom filters," *IEEEACM Trans. Netw.*, vol. 10, no. 5, pp. 604–612, Jul. 2002, doi: 10.1109/TNET.2002.803864.
[34] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal, "The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables," p. 10.
[35] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, "Optimizing Bloom Filter: Challenges, Solutions, and Comparisons." arXiv, Jan. 06, 2019. Accessed: Jun. 17, 2022. [Online]. Available: http://arxiv.org/abs/1804.04777
[36] F. Chang *et al.*, "Bigtable: A Distributed Storage System for Structured Data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 1–26, Jun. 2008, doi: 10.1145/1365815.1365816.
[37] "Bloom Filters | Apache Cassandra Documentation." https://cassandra.apache.org/doc/trunk/cassandra/operating/bloom_filters.html (accessed Jun. 17, 2022).
[38] J. Talbot, "What are Bloom filters?," *3 min read*, Jul. 15, 2015.

https://blog.medium.com/what-are-bloom-filters-1ec2a50c68ff (accessed Jun. 17, 2022).
[39] B. M. Maggs and R. K. Sitaraman, “Algorithmic Nuggets in Content Delivery,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 3, pp. 52–66, Jul. 2015, doi: 10.1145/2805789.2805800.
[40] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, “Cuckoo Filter: Practically Better Than Bloom,” in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, Sydney Australia, Dec. 2014, pp. 75–88. doi: 10.1145/2674005.2674994.
[41] R. Pagh and F. F. Rodler, “Cuckoo hashing,” *J. Algorithms*, vol. 51, no. 2, pp. 122–144, Feb. 2004, doi: 10.1016/j.jalgor.2003.12.002.
[42] “Cuckoo hashing,” *Wikipedia*. Jun. 17, 2022. Accessed: Jun. 17, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cuckoo_hashing&oldid=1093534813
[43] A. D. Breslow and N. S. Jayasena, “Morton filters: Faster, spaceefficient cuckoo filters via biasing, compression, and decoupled logical sparsity,” 2018, vol. 11, no. 9, pp. 1041–1055. doi: 10.14778/3213880.3213884.
[44] “MurmurHash,” *Wikipedia*. Apr. 21, 2022. Accessed: Jun. 21, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=MurmurHash&oldid=1083913936
[45] “Internetstiftelsen Zone Data.” https://zonedata.iis.se/ (accessed Jun. 19, 2022).
[46] “Bloom filter calculator.” https://hur.st/bloomfilter/ (accessed Jun. 19, 2022).
[47] A. Kirsch and M. Mitzenmacher, “Less hashing, same performance: Building a better Bloom filter,” *Random Struct. Algorithms*, vol. 33, no. 2, pp. 187–218, Sep. 2008, doi: 10.1002/rsa.20208.
[48] *Morton Filter Description*. AMD Compute Libraries, 2022. Accessed: Jun. 20, 2022. [Online]. Available: https://github.com/AMDComputeLibraries/morton_filter
[49] *XDP Programming Hands-On Tutorial*. XDP-project, 2022. Accessed: Jun. 20, 2022. [Online]. Available: https://github.com/xdp-project/xdp-tutorial/blob/a24d425b67789276c4a92396811b73d2f4f0ab21/basic-solutions/xdp_loader.c
[50] “Netronome NFP-4000 Flow Processor - Product Brief.” https://www.netronome.com/media/documents/PB_NFP-4000-7-20.pdf (accessed Jul. 06, 2022).
[51] “Debian -- Το Οικουμενικό Λειτουργικό Σύστημα.” https://www.debian.org/ (accessed Jun. 21, 2022).
[52] I. S. Consortium, “BIND 9.” https://www.isc.org/bind/ (accessed Jun. 21, 2022).
[53] “Netdev | Netdev 0x13 - Session.” https://legacy.netdevconf.info/0x13/session.html?xdp-offload-with-virtio-net (accessed Jun. 27, 2022).
[54] “tcprewrite man page.” https://tcpreplay.appneta.com/wiki/tcprewrite-man.html (accessed Jun. 21, 2022).
[55] “Tcpreplay - Pcap editing and replaying utilities.” https://tcpreplay.appneta.com/ (accessed Jun. 21, 2022).
[56] “DNS Statistics - Switch,” *https://www.nic.ch/statistics/dns/*, [Online]. Available: https://www.nic.ch/statistics/dns/
[57] “Home | TCPDUMP & LIBPCAP.” https://www.tcpdump.org/ (accessed Jun. 21, 2022).
[58] “Wireshark · Go Deep.” https://www.wireshark.org/ (accessed Jun. 21, 2022).
[59] P. C. Dillinger and S. Walzer, “Ribbon filter: practically smaller than Bloom and Xor,” *Arxiv Prepr.*, Mar. 2021, [Online]. Available: http://arxiv.org/abs/2103.02515
[60] M. Wang, M. Zhou, S. Shi, and C. Qian, “Vacuum filters: more space-efficient and faster replacement for bloom and cuckoo filters,” *Proc. VLDB Endow.*, vol. 13, no. 2, pp. 197–210, Oct. 2019, doi: 10.14778/3364324.3364333.
[61] T. M. Graf and D. Lemire, “Xor Filters: Faster and Smaller Than Bloom and Cuckoo Filters,” Dec. 2019, doi: 10.1145/3376122.
[62] “PF_RING ZC (Zero Copy),” *ntop*, Apr. 13, 2014.

https://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/ (accessed Jun. 13, 2022).
[63] “Accelerating with XDP over Mellanox ConnectX NICs,” *NVIDIA Technical Blog*, Jun. 18, 2020. https://developer.nvidia.com/blog/accelerating-with-xdp-over-mellanox-connectx-nics/ (accessed Jun. 28, 2022).
[64] “Agilio eBPF Software - Netronome.” https://www.netronome.com/products/agilio-software/agilio-ebpf-software/ (accessed Jun. 28, 2022).

Παράρτημα

Κώδικας

Ο κώδικας του αμυντικού μηχανισμού που υλοποιήθηκε σε αυτή τη διπλωματική εργασία είναι διαθέσιμος στο: ***https://github.com/anadhm/dns-wt-mitigation-xdp-asmds.***