Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Επικοινωνιων Ηλεκτρονικης και Συστηματων Πληροφορικης
Εργαστηριο Διαχειρισης και Βελτιστου Σχεδιασμου Δικτυων Τηλεματικης

Ανίχνευση και αναγνώριση κατανεμημένων επιθέσεων άρνησης παροχής υπηρεσίας στο επίπεδο δεδομένων με χρήση της γλώσσας P4

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Χρήστου Κ. Σεραφείδη

Επιβλέπων: Συμεών Παπαβασιλείου
Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2021

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΔΙΑΧΕΙΡΙΣΗΣ ΚΑΙ ΒΕΛΤΙΣΤΟΥ ΣΧΕΔΙΑΣΜΟΥ ΔΙΚΤΥΩΝ
ΤΗΛΕΜΑΤΙΚΗΣ (NETMODE)

Ανίχνευση και αναγνώριση κατανεμημένων επιθέσεων άρνησης παροχής υπηρεσίας στο επίπεδο δεδομένων με χρήση της γλώσσας P4

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Χρήστου Κ. Σεραφείδη

Επιβλέπων: Συμεών Παπαβασιλείου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 15η Νοεμβρίου 2021.

.....	.....	.....
Συμεών Παπαβασιλείου	Ευστάθιος Συκάς	Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Νοέμβριος 2021.

.....

Χρήστος Κ. Σεραφείδης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών

Σύνοψη

Οι κατανεμημένες επιθέσεις άρνησης παροχής υπηρεσίας (DDoS) αποτελούν μείζον πρόβλημα στην σύγχρονη εποχή του διαδικτύου καθώς στοχεύουν στην παράλυση υπηρεσιών και στην εξάντληση των πόρων που αυτές χρησιμοποιούν, καθιστώντας τις απροσπέλαστες από τους νόμιμους χρήστες τους. Οι επιθέσεις αυτές εξελίσσονται διαρκώς τόσο σε συχνότητα και όγκο όσο και σε ευφυία δυσχαιρένοντας ακόμη περισσότερο τις προσπάθειες ανίχνευσης και αντιμετώπισής τους επιφέροντας λειτουργικές αλλά και πολλές φορές οικονομικές επιπτώσεις στους πληττόμενους οργανισμούς.

Οι διαθέσιμοι μηχανισμοί ανίχνευσης εδράζονται κυρίως στο επίπεδο ελέγχου των δικτυακών υποδομών. Ωστόσο, παρά την ευφυία τέτοιων μηχανισμών, η ανάγκη ετεροχρονισμένης ανάλυσης μεγάλου όγκου κίνησης από εξωτερικούς ελεγκτές απαιτεί πληθώρα υπολογιστικών πόρων και οδηγεί σε καθυστερημένη ανίχνευση των επιθέσεων. Με την ανάπτυξη της δυνατότητας προγραμματισμού στο επίπεδο δεδομένων των δικτυακών συσκευών, δημιουργούνται νέες προοπτικές για τον σχεδιασμό και την υλοποιήση μεθόδων ανίχνευσης επιθέσεων DDoS σε πραγματικό χρόνο κατά τη διέλευση των πακέτων από τις δικτυακές συσκευές, χωρίς την ανάγκη επεξεργασίας από μηχανισμούς του επιπέδου ελέγχου.

Στο πλαίσιο αυτό, βασικός στόχος της παρούσας διπλωματικής εργασίας είναι αφενός η επέκταση ενός μηχανισμού ανίχνευσης επιθέσεων άρνησης παροχής υπηρεσίας με την προσθήκη λειτουργιών χρονικής παρακολούθησης των ανωμαλιών της κίνησης και της δυνατότητας ταυτοποίησης του στόχου αλλά και του τύπου της επίθεσης αφετέρου η βελτίωση της ταχύτητας επεξεργασίας πακέτων μέσα από τη χρήση δειγματοληψίας. Ο μηχανισμός ανίχνευσης καταμετρά τις μοναδικές ροές πακέτων (πλειάδες που αποτελούνται από τις IP διευθύνσεις πηγής και προορισμού, το πρωτόκολλο και τις πόρτες πηγής και προορισμού) συνολικά και ανά παρακολουθούμενο υποδίκτυο καθώς και την ασυμμετρία εισερχόμενων - εξερχόμενων πακέτων για κάθε υποδίκτυο. Με χρήση απλής εκθετικής εξομάλυνσης υπολογίζει κατώφλια που αν ξεπεραστούν δημιουργούνται ειδοποιήσεις πιθανής επίθεσης προς το επίπεδο ελέγχου. Διαμορφώνεται έτσι μια χρονοσειρά πλήθους συμβάντων η οποία είναι διαθέσιμη στο επίπεδο ελέγχου. Για την αναγνώριση των θυμάτων των επιθέσεων επιλέχθηκε και ενσωματώθηκε ο μηχανισμός HashPipe που ανιχενύει τις "top k" ροές κίνησης με βάση τον όγκο τους και περιέχει πληροφορία για την IP διεύθυνση των αποδεκτών της επίθεσης αλλά και για την πόρτα της υπηρεσίας που πλήττεται.

Ο μηχανισμός επιτυγχάνει να αναγνωρίσει άμεσα την έναρξη της επίθεσης καθώς και το μεγαλύτερο μέρος της διάρκειάς της ειδοποιώντας το επίπεδο ελέγχου ώστε να λάβει μέτρα αντιμετώπισης, χωρίς να παράγει αξιοσημείωτο ποσοστό λανθασμένων ειδοποιήσεων. Η εφαρμογή δειγματοληψίας ακόμη και με μεγάλο ρυθμό βελτιώνει αισθητά την ταχύτητα επεξεργασίας των πακέτων από τη δικτυακή συσκευή χωρίς να επιδρά σημαντικά στην ακρίβεια της ανίχνευσης και στο ποσοστό λανθασμένων ειδοποιήσεων.

Λέξεις Κλειδιά: τηλεπικοινωνιακά δίκτυα, ανίχνευση δικτυακών επιθέσεων, κατανεμημένες επιθέσεις άρνησης παροχής υπηρεσίας, προγραμματισμός επιπέδου δεδομένων, δειγματοληψία, P4, αναγνώριση αποδεκτών κακόβουλης κίνησης

Abstract

Distributed denial of service attacks (DDoS) pose a major threat for data networks. They intend to disrupt the normal operation of online services by flooding them with malicious traffic and consuming all available resources, causing these services to be unavailable to legitimate end-users. These attacks tend to use more advanced logic and grow both in frequency and volume, creating functional and economical loses on the organizations under attack.

Most available detection mechanisms reside on the network control plane. Despite their intelligence, the need for delayed traffic analysis requires a great amount of resources and usually fails to instantaneously detect the oncoming attack. The introduction of data plane programmability techniques creates new opportunities for the design and implementation of more robust, real time detection strategies during packet processing.

This thesis extends a DDoS detection mechanism by introducing new functionalities for observing the incidents timeseries and identifying the attack's type and victims. It also tries to enhance the speed of packet processing with the use of packet sampling. The detection mechanism tracks and counts total and per monitored subnet unique flows (tuples that consist of source and destination IP addresses, protocol and source and destination ports) and incoming - outgoing packet asymmetry per subnet. With the use of the simple exponential smoothing statistical technique, the mechanism calculates thresholds that when exceeded trigger the creation of notifications to the control plane indicating possible attacks. These events compose a timeseries that is available to the control plane. To point out the attack's victims, HashPipe, a mechanism that detects "top k" flows with regards to their volume, was implemented and embedded. It holds information about the victim's IP address and the network port of the service under attack.

The proposed mechanism succeeds in instantly detecting the beginning of the attack and also most of its duration, notifying the control plane in order to take mitigation actions, while keeping the percentage of false notifications significantly low. The use of packet sampling, even with high frequency, achieves the boost of the packet processing speed while affecting minimally the detection accuracy and the percentage of false notifications.

Keywords: data networks, network anomaly detection, DDoS, data plane programmability, sampling, P4, attack traffic receiver identification

Ευχαριστίες

Η παρούσα διπλωματική εργασία έρχεται να σφραγίσει την πορεία των προπτυχιακών σπουδών μου πάνω στην επιστήμη του Ηλεκτρολόγου Μηχανικού και Μηχανικού Ηλεκτρονικών Υπολογιστών μέσα από την οποία αποκόμισα πληθώρα γνώσεων και εμπειριών. Θέλω να ευχαριστήσω το εργαστήριο διαχείρισης και βέλτιστου σχεδιασμού δικτύων τηλεματικής και ιδαίτερα τον ομότιμο καθηγητή κύριο Βασίλειο Μάγκλαρη που μου έδωσε την ευκαιρία να ασχοληθώ με το συγκεκριμένο θέμα αλλά και τον υποψήφιο διδάκτορα Μαρίνο Δημολιάνη για την καθοδήγησή του και την άψογη συνεργασία μας στη διάρκεια εκπόνησης της διπλωματικής εργασίας. Τέλος οφείλω ένα μεγάλο ευχαριστώ και ευγνωμοσύνη στην οικογένειά μου, τους γονείς μου Κώστα και Αρχοντούλα και την αδερφή μου Ιωάννα, στους φίλους μου και σε δύο πολύ σημαντικούς για εμένα ανθρώπους, τη Δήμητρα και τον Γιώργο, όλοι εκ των οποίων στάθηκαν αρωγοί στην προσπάθειά μου όλα αυτά τα χρόνια και συνέβαλαν στη δημιουργία των ευχάριστων αναμνήσεων που θα τα χαρακτηρίζουν.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Τμημάτων Κώδικα

Κεφάλαιο 1

Εισαγωγή

1.1 Παρακίνηση - Το πρόβλημα των DDoS επιθέσεων

Οι επιθέσεις άρνησης παροχής υπηρεσιών αποτελούν μείζον πρόβλημα καθώς στοχεύουν στην παράλυση υπηρεσιών και την εξάντληση των πόρων που αυτές χρησιμοποιούν καθιστώντας τις απροσπέλαστες από νόμιμους χρήστες. Με την στρατολόγηση δικτύων botnets από την πλευρά των επιτιθέμενων και με την αξιοποίηση ευφυών τεχνικών, οι επιθέσεις αυτές εμφανίζονται με διαρκώς αυξανόμενη συχνότητα και με ιδιαίτερη πολυπλοκότητα. Η παραγκώνιση της λειτουργίας υπηρεσιών μπορεί να οδηγήσει και σε οικονομικές συνέπειες των πληττόμενων οργανισμών. Η έγκαιρη ανίχνευση και αντιμετώπισή των επιθέσεων αποτελεί δύσκολο έργο για τους διαχειριστές δικτύων και τους σχεδιαστές συστημάτων ασφαλείας.

Οι διαθέσιμοι μηχανισμοί ανίχνευσης και αντιμετώπισης εδράζονται κατά κύριο λόγο στο επίπεδο ελέγχου. Πραγματοποιούν αντιγραφή ή δειγματοληψία της κίνησης και ο έλεγχος πραγματοποιείται σε κεντρικοποιημένο σημείο εκτός των συσκευών πρόωθησης και δρομολόγησης της κίνησης. Παρά την ελευθερία που προσφέρει το επίπεδο ελέγχου για την ανάπτυξη σύνθετων ευφυών μηχανισμών ανίχνευσης, η επιβάρυνση από την ανάγκη ετεροχρονισμένης επεξεργασίας κίνησης μεγάλης κλίμακας οδηγεί σε αργή αντίδραση απέναντι σε εμφανιζόμενες επιθέσεις. Το πρόβλημα αυτό παραμένει και στην περίπτωση δικτύων οριζόμενων από λογισμικό, στα οποία οι λειτουργίες ελέγχου υλοποιούνται σε κεντρικό σημείο και παραμετροποιούν κατάλληλα τις συσκευές προώθησης.

Η εμφάνιση της δυνατότητας προγραμματισμού του επιπέδου δεδομένων δημιουργεί νέες προοπτικές για τον σχεδιασμό και υλοποίηση συστημάτων ανίχνευσης και αντιμετώπισης επιθέσεων απευθείας στο επίπεδο δεδομένων κατά τη διέλευση της κίνησης σε πραγματικό χρόνο και χωρίς ανάγκη παρέμβασης από εξωτερικούς ελεγκτές. Ωστόσο οι μηχανισμοί αυτοί θα πρέπει να μην καθυστερούν ή δυσχαιρένουν τη βασική λειτουργία του επιπέδου δεδομένων δηλαδή την προώθηση της κίνησης. Αυτός ο περιορισμός - σε συνδυασμό με την πρώιμη μορφή των δυνατοτήτων προγραμματισμού στο επίπεδο δεδομένων - δημιουργούν νέες προοπτικές για τον σχεδιασμό και την υλοποίηση ευφυών και ταυτόχρονα αποδοτικών μηχανισμών ανίχνευσης επιθέσεων.

1.2 Συνεισφορά

Βασικοί στόχοι της παρούσας διπλωματικής εργασίας αποτελούν οι εξής:

- Η βελτιστοποίηση και επέκταση ενός μηχανισμού [1] [2] ανίχνευσης κατανεμημένων επιθέσεων άρνησης υπηρεσίας (DDoS) με την προσθήκη επιπλέον διαδικασιών που θα επιτρέπουν την εγκυρότερη ανίχνευση και θα παρέχουν κατάλληλα δεδομένα που θα μπορούν να αξιοποιηθούν για την περαιτέρω ανάλυση της κίνησης ενός δικτύου.

- Η μελέτη της επίδρασης της εφαρμογής δειγματοληψίας στον ανωτέρω μηχανισμό προκειμένου να ενισχυθεί η ρυθμαπόδοσή του χωρίς παράλληλα να επηρεαστεί αρνητικά η εγκυρότητα της ανίχνευσης των επιθέσεων.

1.3 Δομή της εργασίας

Στο κεφάλαιο 2 γίνεται αναφορά στο θεωρητικό υπόβαθρο σχετικά με τα δίκτυα οριζόμενα από λογισμικό, τον προγραμματισμό στο επίπεδο δεδομένων του δικτύου και τις επιθέσεις κατανεμημένης άρνησης παροχής υπηρεσίας (DDoS). Επίσης αναλύονται τα βασικά χαρακτηριστικά της γλώσσας $P4$ που χρησιμοποιήθηκε για την ανάπτυξη του προτεινόμενου μηχανισμού. Στο κεφάλαιο 3 αναλύονται προηγούμενες εργασίες πάνω στον τομέα των δικτυακών επιθέσεων και του προγραμματισμού στο επίπεδο δεδομένων. Ο μηχανισμός που υλοποιήθηκε και οι τεχνικές λεπτομέριες της υλοποίησης παρουσιάζονται στο κεφάλαιο 4. Το κεφάλαιο 5 περιλαμβάνει τις πειραματικές διατάξεις και την ανάλυση των αποτελεσμάτων που προέκυψαν. Τέλος, στο κεφάλαιο 6 παρουσιάζονται τα συμπεράσματα της εργασίας καθώς επίσης και κάποιες μελλοντικές επεκτάσεις.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

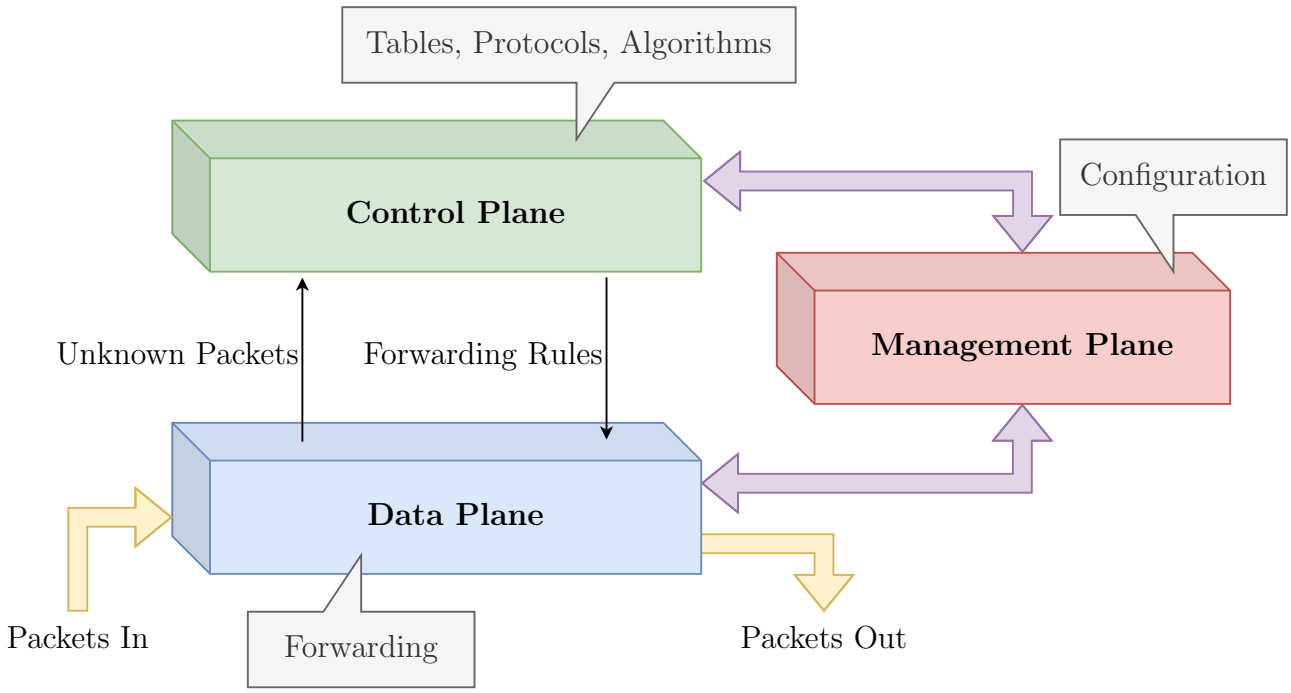
2.1 Δίκτυα οριζόμενα σε λογισμικό (SDN) & Προγραμματισμός επιπέδου δεδομένων

Οι δικτυακές συσκευές αναλαμβάνουν την διαχείριση και προώθηση πακέτων που παράγονται εντός ενός δικτύου δεδομένων. Οι λειτουργίες που σχετίζονται με αυτή τη διαδικασία κατατάσσονται σε τρία επίπεδα

- **Επίπεδο Δεδομένων - Data plane**: Είναι υπεύθυνο για την διαχείριση των πακέτων που φτάνουν στην συσκευή από κάποια διεπαφή εισόδου. Αποκαλείται και επίπεδο προώθησης καθώς χρησιμοποιεί πίνακες προώθησης ώστε να ορίσει τις διεπαφές εξόδου των πακέτων.
- **Επίπεδο Ελέγχου - Control plane**: Αφορά την συλλογή, επεξεργασία και διαχείριση πληροφοριών του δικτύου ώστε να καθορίσει τους κανόνες με τους οποίους θα γίνει η προώθηση των πακέτων από το επίπεδο δεδομένων. Η λειτουργία του βασίζεται σε σαφώς ορισμένα πρωτόκολλα .
- **Επίπεδο Διαχείρισης - Management Plane**: Περιλαμβάνει τις λειτουργίες που σχετίζονται με την αλληλεπίδραση και την παρακολούθηση των δικτυακών συσκευών με σκοπό τη διαχείριση των δικτυακών υποδομών.

Στα παραδοσιακά δίκτυα τα επίπεδα δεδομένων και ελέγχου είναι άρρηκτα συνδεδεμένα. Ο κατασκευαστής της συσκευής καθορίζει εκ των προτέρων την λειτουργία του επιπέδου δεδομένων σύμφωνα με τα πρωτόκολλα που θα υποστηρίζει η συσκευή. Οι κανόνες προώθησης διαμορφώνονται τοπικά στη συσκευή με βάση ειδικούς αλγορίθμους και ο διαχειριστής του δικτύου έχει περιορισμένες δυνατότητες παραμετροποίησης τους και μόνο στο πλαίσιο της εκάστοτε συσκευής. Ωστόσο με την διάδοση του διαδικτύου και τη μεγέθυνση των δικτύων δεδομένων που πλέον περιλαμβάνουν εκατονάδες δικτυακές συσκευές, αυτός ο τρόπος διαχείρισης καθίσταται εξαιρετικά δύσκολος και επιρρεπής σε πληθώρα σφαλμάτων.

Η ανάγκη ευρύτερης παραμετροποίησης και κεντρικού ελέγχου των δικτυακών υποδομών οδήγησε στον διαχωρισμό των δύο επιπέδων και στη δημιουργία των δικτύων οριζόμενων σε λογισμικό. Το επίπεδο ελέγχου αποσυνδέεται από το επίπεδο δεδομένων και μπορεί να αποσπαστεί από τις συσκευές προώθησης και να μεταφερθεί σε κεντρικούς ελεγκτές. Δίνεται έτσι η δυνατότητα για πιο σύνθετη και ευρεία παραμετροποίηση των κανόνων προώθησης αλλά και για μαζικό έλεγχο πολλαπλών δικτυακών συσκευών. Την πιο χαρακτηριστική υλοποίηση τέτοιων δικτύων αποτελεί το πρωτόκολλο OpenFlow [4] το οποίο ορίζει τον τρόπο επικοινωνίας ενός κεντρικού ελεγκτή με ειδικούς μεταγωγείς. Ομαδοποιεί τα πακέτα σε ροές σύμφωνα με ορίσμένα χαρακτηριστικά των επικεφαλίδων των πρωτοκόλλων και δίνει τη



Σχήμα 2.1: Επίπεδα δικτύου [3]

δυνατότητα στον διαχειριστή του δικτύου να ορίσει τις ενέργειες που θα εκτελούνται για κάθε ροή καθορίζοντας κατ' επέκταση και τη λειτουργία όλης της δικτυακής υποδομής. Οι κανόνες που προκύπτουν παραμετροποιούν ταυτόχρονα όλες τις δικτυακές συσκευές. Τόσο τα διαθέσιμα πρωτόκολλα όσο και οι διαθέσιμες ενέργειες είναι καθορισμένα από τον κατασκευαστή της συσκευής και ο διαχειριστής του δικτύου δεν μπορεί να παρέμβει με τροποποιήσεις ή προσθήκες.

Παρά τις ευκολίες που προσφέρει η κεντρική διαχείριση των δικτυακών υποδομών στα δίκτυα οριζόμενα από λογισμικό, η ανάγκη για παρέμβαση των κατασκευαστών συσκευών προκειμένου να υποστηριχθούν νέα πρωτόκολλα και λειτουργιες επεξεργασίας για τα πακέτα, οδήγησε στην δυνατότητα προγραμματισμού του επιπέδου δεδομένων [5]. Δίνεται η δυνατότητα, στο βαθμό που το επιτρέπει ο σχεδιασμός της εκάστοτε δικτυακής συσκευής, ο διαχειριστής του δικτύου να ορίζει αυθαίρετα νέες επικεφαλίδες καθώς και μεθόδους επεξεργασίας των πακέτων και να δημιουργεί νέα είδη κανόνων προώθησης. Αυτό ανοίγει το δρόμο για τον σχεδιασμό και την υλοποίηση νέων τεχνικών ανάλυσης της δικτυακής κίνησης αλλά και για την ανάπτυξη νέων μεθόδων ανίχνευσης επιθέσεων και διατήρησης της ασφάλειας των δικτύων άμεσα και χωρίς την ανάγκη παρέμβασης από τους κατασκευαστές των συσκευών.

2.2 Η γλώσσα P4

Η γλώσσα P4 (Programming Protocol-independent Packet Processors) [6] αποτελεί μια γλώσσα ειδικού σκοπού που αναπτύχθηκε με σκοπό τον προγραμματισμό των λειτουργιών προώθησης των δικτυακών συσκευών. Επιτρέπει τον καθορισμό του τρόπου κατά τον οποίο τα πακέτα θα επεξεργαστούν από το επίπεδο δεδομένων (dataplane) ενός προγραμματιζόμενου hardware ή software στοιχείου δικτύωσης (switch ή κάρτας δικτύου ή router). Αν και αρχικά προορίζονταν για προγραμματιζόμενα switches στη συνέχεια χρησιμοποιήθηκε σε ποικιλία δικτυακού εξοπλισμού.

Ένα τέτοιο στοιχείο δικτυακού εξοπλισμού υλοποιεί τόσο το επίπεδο ελέγχου (controlplane) όσο και το επίπεδο δεδομένων (dataplane). Η P4 προορίζεται για την υλοποίηση του επιπέδου δεδομένων. Επιπλέον, ένα P4 πρόγραμμα εν μέρει ορίζει τη διεπαφή επικοινωνίας μεταξύ των επιπέδων δεδομένων και ελέγχου, αλλά δεν μπορεί να χρησιμοποιηθεί για τη λειτουργία του επιπέδου ελέγχου.

Οι βασικοί παράγοντες που οδήγησαν στην ανάπτυξη της P4, σύφμωνα με τους δημιουργούς της, είναι οι εξής:

- Δυνατότητα τροποποίησης της λειτουργίας των δικτυακών συσκευών (Reconfigurability). Θα πρέπει να είναι δυνατή η αλλαγή του τρόπου επεξεργασία και διαχείρισης των πακέτων χωρίς την παρέμβαση του κατασκευαστή της συσκευής.
- Ανεξαρτησία από πρωτόκολλα (Protocol Independence). Η λειτουργία της συσκευής δεν θα πρέπει να δεσμεύεται από τη χρήση συγκεκριμένων πρωτοκόλλων.
- Ανεξαρτησία από συσκευές στόχους (Target Independence). Ένα πρόγραμμα γραμμένο σε P4 θα πρέπει να μπορεί να λειτουργεί σε κάθε συσκευή που την υποστηρίζει χωρίς ο προγραμματιστής να χρειάζεται να γωνρίζει λεπτομέρειες σχετικά με την κατασκευή της δικτυακής συσκευής και χωρίς να απαιτούνται τροποποίησεις για τη μεταφερσιμότητα του υλοποιημένου προγράμματος.

Ο πρώτος από αυτούς τους στόχους ικανοποιείται μιας και ο χρήστης της συσκευής μπορεί να ορίσει ανά πάσα στιγμή τον επιθυμητό μηχανισμό επεξεργασίας και διαχείρισης των πακέτων και να επαναπρογραμματίσει την συσκευή με βάση αυτόν. Ο δεύτερος στόχος ικανοποιείται από το γεγονός ότι η P4 επιτρέπει την δημιουργία αυθαίρετων επικεφαλίδων - άρα και πρωτοκόλλων - με τις οποίες θα ορίζεται η λειτουργία του δικτύου και η μορφή των πακέτων. Η ικανοποίηση του τρίτου στόχου εξαρτάται σε μεγάλο βαθμό από τους κατασκευαστές των συσκευών. Η P4 ορίζει δομές και βασικές λειτουργίες που θα πρέπει να υποστηρίζει η συσκευή αλλά χωρίς τον τρόπο με τον οποίο αυτές θα υλοποιούνται. Οι κατασκευαστές που επιθυμούν οι συσκευές τους να είναι συμβατές με την P4 οφείλουν να παρέχουν κατάλληλους μεταγλωττιστές που θα μετατρέπουν κατάλληλα τα P4 προγράμματα σε εκτελέσιμο κώδικα για την συγκεκριμένη συσκευή. Επιπλέον, επειδή η P4 δίνει την δυνατότητα υλοποίησης δομών και λειτουργιών πέραν των βασικών, δεν εξασφαλίζεται η συμβατότητα μεταξύ διαφορετικών κατασκευαστών καθώς κάθε συσκευή μπορεί να υποστηρίζει διαφορετικό σύνολο επιπλεόν λειτουργιών. Ακόμη, είναι δυνατόν το P4 πρόγραμμα να αιτήται πόρους που η συσκευή να μην μπορεί να παρέχει, περιορίζοντας έτσι τις διαθέσιμες δυνατότητες ανάπτυξης νέων λειτουργιών.

2.2.1 Αρχιτεκτονικό μοντέλο συσκευής και βιβλιοθήκες

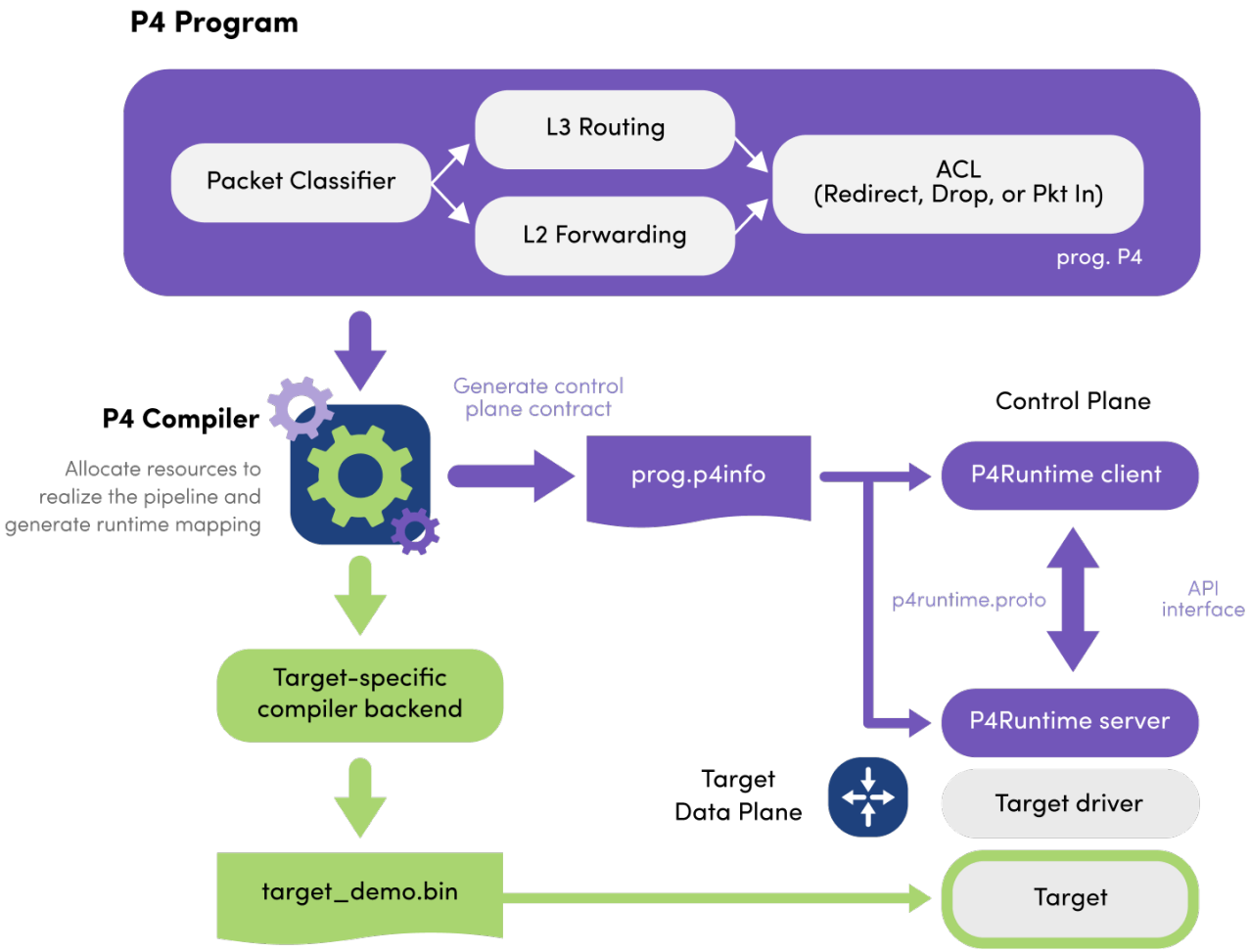
Οι κατασκευαστές συσκευών συμβατών για προγραμματισμό με χρήση της P4 οφείλουν πέραν του μεταγλωττιστή να παρέχουν και τον ορισμό σε P4 των διαθέσιμων δομών που μπορούν να προγραμματιστούν καθώς επίσης και όλων των επιπλέον λειτουργιών που υποστηρίζει η εκάστοτε συσκευή.

Η P4 παρέχει τον ορισμό μιας βασικής, ευρέως χρησιμοποιούμενης αρχιτεκτονικής, γνωστής ως v1model. Όπως φαίνεται στην εικόνα 2.4, αυτή αποτελείται από 5 διακριτά τμήματα: Το κομμάτι του parser, το ingress control, το σύστημα διαχείρισης της κίνησης, το egress control και τέλος το deparsing. Τα στάδια αυτά καθώς και οι λειτουργίες που επιτελεί το καθένα θα παρουσιαστούν στις επόμενες παραγράφους.

2.2.2 Βασικοί και σύνθετοι τύποι δεδομένων

Η P4 περιλαμβάνει τους εξής βασικούς τύπους:

- `int<N>`: Προσημασμένοι ακέραιοι μεγέθους N bits
- `bit<N>`: Μη προσημασμένοι ακέραιοι μεγέθους N bits
- `varbit<N>`: Συμβολοσειρές μεγέθους το πολύ N bits



Σχήμα 2.2: Προγραμματισμός δικτυακής συσκευής με P4 [7]

- `bool`: Λογικές τιμές true, false

Πάνω στους τύπους `bit<N>` και `int<N>` ορίζονται οι βασικές αριθμητικές πράξεις της πρόσθεσης, της αφαίρεσης και του πολλαπλασιασμού. Η πράξη της διαίρεσης δεν υποστηρίζεται. Επιτρέπεται επίσης η μετατροπή τύπων από προσημασμένο σε μή προσημασμένο σταθερού μεγέθους και αντίστροφα (`int<N>` $\longleftrightarrow$ `bit<N>`) και η μετατροπή του μεγέθους του αριθμού για σταθερό τύπο προσήμου (`int<N>` $\longleftrightarrow$ `int<M>`) και (`bit<N>` $\longleftrightarrow$ `bit<M>`).

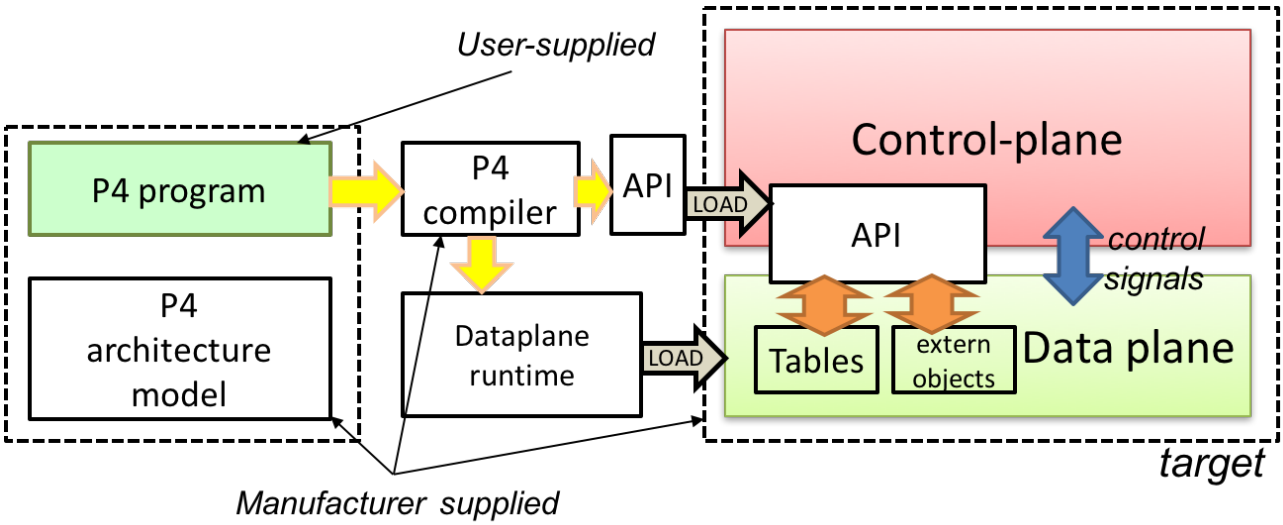
Με χρήση των παραπάνω βασικών τύπων μπορούν να οριστθούν σύνθετοι τύποι όπως:

- **structs**. Δομές που περιέχουν ένα σύνολο πεδίων. Ο τύπος αυτών μπορεί να είναι κάποιος από τους βασικούς ή κάποια άλλη δομή struct.
- **headers**. Δομές όπως τα structs που περιέχουν επιπλέον ένα validity bit το οποίο είναι προσπελάσιμο με τις συναρτήσεις `isValid` και `setValid, setInvalid` και υποδηλώνει την εγκυρότητα των δεδομένων της δομής

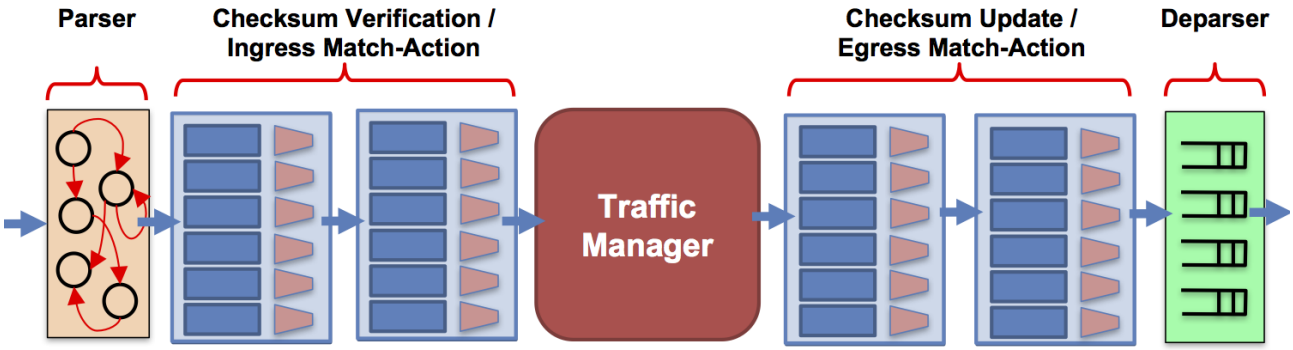
Οι δομές structs μπορούν να χρησιμοποιηθούν για τον ορισμό των μεταδεδομένων στα πακέτα και οι δομές headers για τον ορισμό επικεφαλίδων.

2.2.3 Parsers

Οι parsers είναι μηχανές καταστάσεων στις οποίες ο προγραμματιστής ορίζει τόσο τη λειτουργία κάθε κατάστασης όσο και τις μεταβάσεις μεταξύ διαφορετικών καταστάσεων. Βασική λειτουργία κάθε



Σχήμα 2.3: Προγραμματισμός δικτυακής συσκευής με P4 [8]



Σχήμα 2.4: v1model - Το βασικό μοντέλο αρχιτεκτονικής της P4 [9]

κατάστασης αποτελεί η ανάγνωση και εξαγωγή των επικεφαλίδων από τα πακέτα. Τα εξαγόμενα δεδομένα τοποθετούνται σε δομές headers. Σε αυτές περιέχονται με τη σειρά τα πεδία που θα εξαχθούν απο το πακέτο καθώς επίσης και ένα πεδίο που υποδηλώνει την εγκυρότητα των δεδομένων που περιέχει η δομή (validity bit). Αν μια επικεφαλίδα εξαχθεί επιτυχώς τότε είναι έγκυρη, αλλιώς είναι άκυρη.

Η λειτουργία ενός parser ξεκινά πάντα από την κατάσταση `start` και καταλήγει σε μία εκ των `accept` ή `reject`. Η εξαγωγή των δεδομένων της επικεφαλίδας γίνεται με την συνάρτηση `extract()` η οποία δέχεται ως όρισμα μια δομή header. Αν όλα τα πεδία της δομής εξαχθούν επιτυχώς τότε τίθεται επιπλέον και το validity bit. Η μετάβαση μεταξύ καταστάσεων γίνεται μέσω της εντολής `transition`. Για τη διαχείριση περιπτώσεων υπάρχει η εντολή `select()` η οποία επιτρέπει την μετάβαση σε διαφορετική κατάσταση ανάλογα με την τιμή κάποιου πεδίου.

2.2.4 Deparsers

Οι deparsers αναλαμβάνουν την επανατοποθέτηση των - επεξεργασμένων ή μη - επικεφαλίδων πίσω στο πακέτο ώστε αυτό να εξαχθεί από την δικτυακή συσκευή. Η τοποθέτηση των επικεφαλίδων γίνεται με χρήση της συνάρτησης `emit()` η οποία θα τοποθετήσει την επικεφαλίδα στο πακέτο μόνο αν αυτή είναι έγκυρη (έχει τεθεί το validity bit). Για πολλαπλές επικεφαλίδες, η κλήση της συνάρτησης emit θα πρέπει να γίνει για κάθε επικεφαλίδα και με την ίδια σειρά με την οποία οι επικεφαλίδες εξήχθησαν από το πακέτο.

2.2.5 Actions

Τα actions είναι συναρτήσεις που τροποποιούν δεδομένα των επικεφαλίδων και των μεταδεδομένων των πακέτων. Επίσης, συγκεντρώνουν υπολογισμούς που επαναλαμβάνονται σε πολλά σημεία του υπόλοιπου κώδικα. Δέχονται δύο τύπους παραμέτρων, τις κατευθυνόμενες (directional) και τις μη κατευθυνόμενες (directionless). Οι πρώτες μπορούν να χαρακτηριστούν ως παράμετροι εισόδου (in) εισόδου - εξόδου (inout) ή μόνο εξόδου (out). Οι δεύτερες μπορούν να χρησιμοποιηθούν μόνο ως παράμετροι εισόδου. Στο κυρίως σώμα των συναρτήσεων αυτών επιτρέπεται μόνο η εκτέλεση αριθμητικών και λογικών πράξεων και η χρήση δομών if-else. Οι επαναληπτικές δομές δεν υποστηρίζονται.

2.2.6 Controls

Στα control blocks πραγματοποιείται όλη η επεξεργασία μεταδεδομένων και επικεφαλίδων των πακέτων, οι επιθυμητές λειτουργίες υπολογισμών και εξαγωγών στατιστικών δεδομένων καθώς και η διαδικασία της προώθησης των πακέτων. Το κυρίως σώμα τους περιλαμβάνεται εντός του apply block. Σε αυτό συνυπάρχουν αυτούσια κομμάτια κώδικα, δομές if-else, κλήσεις συναρτήσων (actions), κλήσεις άλλων control blocks, εφαρμογή δομών αντιστοίχισης (tables) και λειτουργίες ανάγνωσης και εγγραφής σε δομές registers και counters. Και πάλι, οι επαναληπτικές δομές δεν υποστηρίζονται.

Σύμφωνα με το v1model (εικόνα 2.4), οι διαδικασίες ingress και egress καθώς επίσης και ο έλεγχος και η ανανεώση των checksums αλλά και η διαδικασία του deparsing πραγματοποιούνται από control blocks.

2.2.7 Tables

Τα tables αποτελούν δομές αντιστοίχισης. Οι εγγραφές του πίνακα είναι της μορφής $key \Rightarrow action, data$. Κατά τον ορισμό ενός πίνακα καθορίζεται το κλειδί (key) καθώς και ο τρόπος αντιστοίχισης με εγγραφές της μορφής field: match_action. Καθορίζονται επίσης το μέγεθος του πίνακα (size), οι πιθανές ενέργειες (actions) που μπορεί να εκτελέσει και προαιρετικά η προκαθορισμένη ενέργεια (default_action). Οι ενέργειες (§2.2.5) για κάθε κλειδί και τα αντίστοιχα δεδομένα μπορούν να ορισθούν από το control plane. Τα δεδομένα αυτά θα αντιστοιχιθούν στις directionless παραμέτρους των συναρτήσεων.

Κατά την εφαρμογή ενός πίνακα σε μια τιμή, αυτή συγκρίνεται με το κλειδί της εκάστοτε εγγραφής του πίνακα με τον τρόπο που έχει ορισθεί. Αν υπάρξει αντιστοίχιση τότε εκτελείται η ενέργεια που αντιστοιχεί στη συγκεκριμένη εγγραφή χρησιμοποιώντας τα αντίστοιχα δεδομένα. Η εφαρμογή του πίνακα εκτελείται με την συνάρτηση `apply()`. Αυτή επιστρέφει δύο τιμές: την τιμή `hit` η οποία αποτιμάται σε true αν υπήρξε αντιστοίχιση με κάποιον κανόνα, αλλιώς αποτιμάται σε false, και την την τιμή `action_run` που επιστρέφει την συάρτηση που εκτελέστηκε.

2.2.8 Registers & Counters

Στη βασική αρχιτεκτονική περιγράφονται επίσης και οι εξωετερικές δομές registers και counters. Εξωτερικές υπό την έννοια ότι διατηρούν τις τιμές τους μεταξύ των πακέτων. Αυτές χρησιμεύουν για την αποθήκευση δεδομένων που είναι απαραίτητα καθόλη τη διάρκεια εκτέλεσης της προγραμματισμένης διαδικασίας καθώς και για την καταμέτρηση γεγονότων αντίστοιχα.

Οι registers υποστηρίζουν την ανάγνωση και την εγγραφή τιμών με τις συναρτήσεις `read` και `write` αντίστοιχα και οι τιμές τους είναι προσπελάσιμες τόσο από το data plane όσο και από το control plane. Οι counters υποστηρίζουν μόνο τη λειτουργία της καταμέτρησης με τη συνάρτηση `count` και οι τιμές τους είναι προσπελάσιμες μόνο από το control plane.

2.2.9 Digests

Τα digests αποτελούν μηνύματα μικρού μεγέθους που μπορεί να στείλει ασύγχρονα το data plane προς το control plane χωρίς να υπάρχει ανάγκη το δεύτερο να ελέγχει ανά τακτά χρονικά διαστήματα την ύπαρξή τους. Το περιεχόμενο των μηνυμάτων μπορεί να είναι ελεύθερα ορισμένο από τον προγραμματιστή ανάλογα με το είδος της επιθυμητής ειδοποίησης. Η αποστολή τους γίνεται με χρήση της συνάρτησης `digest()`.

2.2.10 Packages

Ο τύπος package αποτελεί τον τρόπο με τον οποίο εκφράζεται το αρχιτεκτονικό μοντέλο της συσκευής όπως αυτό έχει οριστεί από τον κατασκευαστή. Αποτελεί την διεπαφή με την οποία όλα τα επιμέρους τμήματα της υλοποιήσης συνδέονται μεταξύ τους για να αποτελέσουν την πλήρη περιγραφή της λειτουργίας της δικτυακής συσκευής.

2.3 Επιθέσεις DDoS

Οι επιθέσεις άρνησης παροχής υπηρεσίας ή DoS αποτελούν μια υποκατηγορία κακόβουλης κίνησης που υπονομεύει την ασφάλεια και την καλή λειτουργία των δομών και των υπηρεσιών του διαδικτύου. Πρόκειται για επιθέσεις που στοχεύουν στον κατακλυσμό των δομών που φιλοξενούν κάποια υπηρεσία με μεγάλο όγκο κίνησης ώστε η υπηρεσία να αδυνατεί να επεξεργαστεί τα αιτήματα των νόμιμων χρηστών της [10].

Οι επιθέσεις αυτές παρουσιάζουν κάποια συγκεκριμένα τεχνικά χαρακτηριστικά. Οι κακόβουλοι χρήστες στρατολογούν πλήθος συσκευών με πρόσβαση στο διαδίκτυο (υπολογιστές, routers, κινητά τηλέφωνα μέχρι και IoT συσκευές [11]) οι οποίες έχουν προσβληθεί σε προηγούμενο χρόνο με κακόβουλο κώδικα. Αυτές οργανώνονται ως ένα δίκτυο (botnet) και κατακλύζουν το θύμα με μεγάλους όγκους κακόβουλης κίνησης από πολλές διαφορετικές πηγές και με πακέτα μεταβλητής μορφής. Είναι μάλιστα πολύ συνηθισμένο να αποκρύπτονται οι πραγματικές IP διευθύνσεις των επιτιθέμενων συσκευών και να χρησιμοποιούνται τυχαίες (address spoofing) ώστε να είναι αδύνατος ο εντοπισμός των πηγών της επίθεσης [12]. Οι επιθέσεις αυτές συχνά εκμεταλλεύονται αδυναμίες των λειτουργικών συστημάτων αλλά και κάποια χαρακτηριστικά πρωτοκόλλων όπως στην περίπτωση των επιθέσεων SYN flooding την τριπλή χειραψία του πρωτοκόλλου TCP [13]. Άλλοτε οι επιτιθέμενες συσκευές προσποιούμενες το θύμα, πραγματοποιούν αιτήματα προς άλλες νόμιμες υπηρεσίες οι οποίες κατακλύζουν με τις απαντήσεις τους τον αποδέκτη της επίθεσης όπως στην περίπτωση του DNS amplification [14]. Στο [15] γίνεται μια εκτενής ανάλυση και κατηγοριοποίηση των επιθέσεων DDoS με βάση πληθώρα χαρακτηριστικών.

Σύμφωνα με στατιστικά [16] [17] [18] η εμφάνιση τέτοιων επιθέσεων αυξάνεται τόσο σε συχνότητα και όγκο όσο και σε πολυπλοκότητα. Κρίνεται επιτακτική η ανάγκη σχεδιασμού μηχανισμών που θα μπορούν έγκαιρα να τις ανιχνεύσουν και να τις αντιμετωπίσουν. Το πεδίο των προγραμματιζόμενων διαδικασιών στο επίπεδο δεδομένων προσφέρει νέες προοπτικές για τον σχεδιασμό τέτοιων συστημάτων.

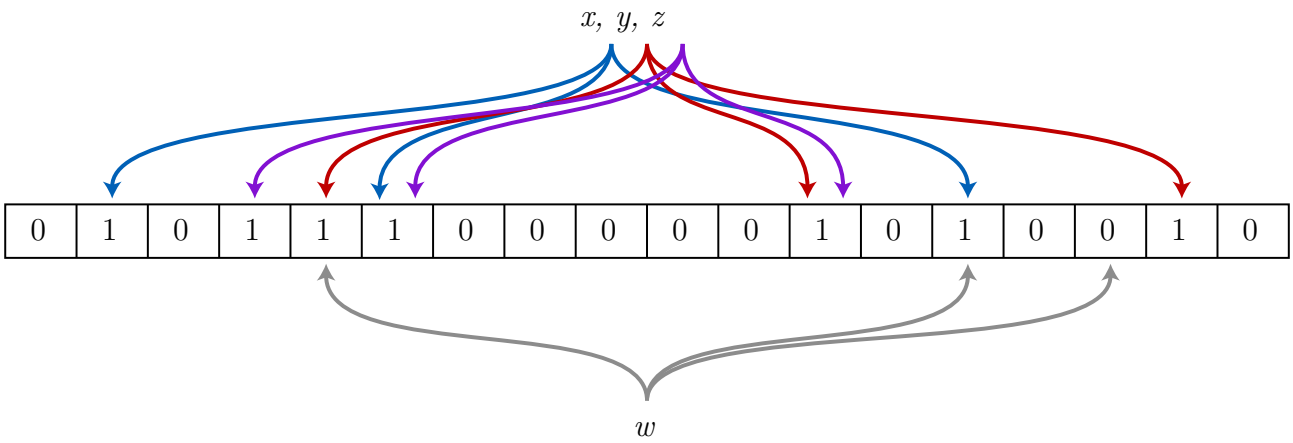
2.4 Bloom Filters & Sketches

2.4.1 Bloom Filters

Τα Bloom Filters [19] αποτελούν πιθανοτικές δομές για τον έλεγχο της ύπαρξης ενός στοιχείου σε ένα σύνολο με τρόπο αποδοτικό τόσο σε απαιτήσεις χρόνου εκτέλεσης όσο και απαιτούμενης μνήμης. Αποτελούνται από έναν πίνακα μεγέθους m bits και από k συναρτήσεις κατακερματισμού οι οποίες αντιστοιχίζουν κάθε στοιχείο του συνόλου σε κάποια από τις m θέσεις του πίνακα. Η εισαγωγή ενός στοιχείο στο bloom filter πραγματοποιείται τροφοδοτώντας το στοιχείο στις k συναρτήσεις κατακερματισμού και θέτοντας σε 1 τις αντίστοιχες θέσεις του πίνακα που θα προκύψουν ως οι τιμές εξόδου των συναρτήσεων. Για τον έλεγχο ύπαρξης ενός στοιχείου στο σύνολο, αυτό τροφοδοτείται στις k συναρτήσεις κατακερματισμού και στη συνέχεια ελέγχεται η τιμή των αντίστοιχων θέσεων του πίνακα. Αν έστω και μία από αυτές τις θέσεις έχει τιμή 0 τότε το στοιχείο *σίγουρα* δεν υπάρχει μέσα στο σύνολο. Αν όλες οι θέσεις έχουν την τιμή 1 τότε *πθανώς* το στοιχείο να υπάρχει στο σύνολο. Η πιθανότητα λανθασμένης εκτίμησης προσεγγίζεται από τον τύπο $P = (1 - e^{-kn/m})^k$ όπου k το πλήθος των συναρτήσεων κατακερματισμού, m το μέγεθος του πίνακα και n το πλήθος των στοιχείων που έχουν ήδη εισαχθεί στην δομή.

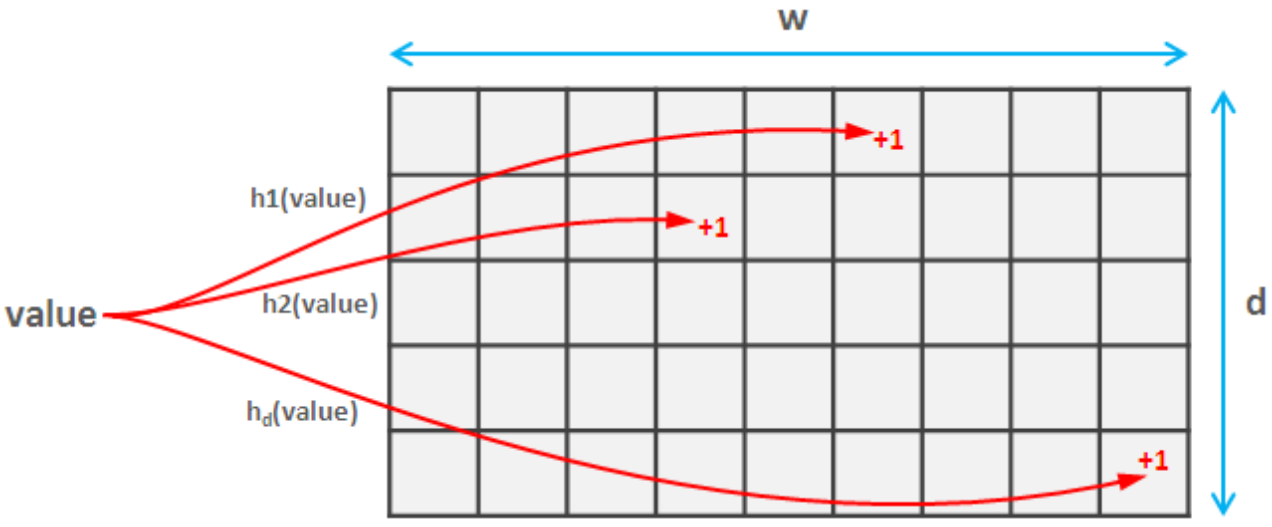
2.4.2 Count-Min Sketches

Οι δομές sketch [21] βασίζονται στον αλγόριθμο count sketch με τον οποίο μπορεί να βρεθεί αποδοτικά η συχνότητα εμφάνισης ενός στοιχείου σε ένα σύνολο. Μία παραλλαγή τους η οποία χρησιμοποιείται



Σχήμα 2.5: Παράδειγμα Bloom Filter [20]

στο πλαίσιο αυτής της διπλωματικής εργασίας είναι τα count min sketches [22]. Πρόκειται για έναν πίνακα w στηλών και d γραμμών και d συναρτήσεις κατακερματισμού $h_1 \dots h_d$ οι οποίες αντιστοιχίζουν στοιχεία του συνόλου σε θέσεις επί των γραμμών του πίνακα, $h_i : [n] \rightarrow [w]$. Κάθε φορά που εισάγεται ένα στοιχείο, η θέση κάθε γραμμής που υποδεικνύεται από την αντίστοιχη συνάρτηση κατακερματισμού αυξάνεται κατά 1. Η συχνότητα εμφάνισης κάθε στοιχείου προκύπτει από την εύρεση των τιμών των θέσεων που υποδεικνύουν στον πίνακα οι συναρτήσεις κατακερματισμού και την επιλογή της μικρότερης από αυτές. Ο αλγόριθμος παρέχει μια εκτίμηση της συχνότητας $\widehat{a_i}$ η οποία είναι ίση ή μεγαλύτερη της πραγματικής συχνότητας εμφάνισης του στοιχείου a_i Οι παράμετροι w και d επηρεάζουν το σφάλμα της εκτίμησης συχνότητας. Συγκεκριμένα για $w = \lceil e/\varepsilon \rceil$ και $d = \lceil ln1/\delta \rceil$ για την εκτίμηση συχνότητας ισχύει $a_i \leq \widehat{a_i} \leq a_i + \varepsilon N$ με πιθανότητα $1 - \delta$ όπου N το μέγεθος του συνόλου.



Σχήμα 2.6: Count-Min Sketch [23]

Κεφάλαιο 3

Συναφείς Εργασίες

3.1 Ανίχνευση επιθέσεων DDoS μέσω υπολογισμού της εντροπίας

Ο υπολογισμός της εντροπίας ως μέσο για τον εντοπισμό DDoS επιθέσεων προτείνεται στην δημοσίευση των Li, Zhou και Xiao [12]. Οι συγγραφείς αναγνωρίζουν ότι είναι δύσκολο να διακριθεί μία περίοδος ιδιαίτερα αυξημένης κίνησης από μία από μία οργανωμένη επίθεση. Η μέθοδος της εντροπίας αξιοποιεί τα στατιστικά χαρακτηριστικά από συγκεκριμένα πεδία στις επικεφαλίδες των πακέτων, όπως οι διευθύνσεις IP και η τιμή του TTL. Προτείνουν μία μέθοδο που βασίζεται στην αθροιστική εντροπία και μία που βασίζεται στον χρόνο.

Η προτεινόμενη μέθοδος της αθροιστικής εντροπίας χρησιμοποιεί τον αλγόριθμο CUMSUM (Cumulative Sum). Ο αλγόριθμος αυτός προέρχεται από το πεδίο του ελέγχου στοχαστικών διαδικασιών και βασίζεται στο γεγονός ότι οποιαδήποτε αλλαγή στην στοχαστική διαδικασία επηρεάζει την κατανομή της τυχαίας διαδικασίας [24] [25]. Στην συγκεκριμένη περίπτωση, οι συγγραφείς υπολογίζουν την εντροπία X_n της χρονοσειράς των εισερχόμενων πακέτων και κάνουν την παραδοχή ότι η διακύμανση είναι ίση με την μέση τιμή, δηλαδή $var(X_n) = E(X_n) = \alpha$. Στη συνέχεια μετατοπίζουν την χρονοσειρά κατά $\beta = 2\alpha$, ώστε να προκύψει η $Z_n = X_n - \beta$. Η Z_n έχει αρνητικές τιμές όταν η κίνηση είναι φυσιολογική και όταν η τιμή της γίνει θετική, τότε θεωρούν ότι υπάρχει επίθεση.

Ακολούθως επεκτείνουν την μέθοδο, ώστε να είναι πιο ανθεκτική σε απότομες αλλά σύντομης διάρκειας αυξήσεις τις κίνησης. Πιο αναλυτικά, μία μοναδική θετική τιμή δεν είναι αρκετή για να διαπιστωθεί με ασφάλεια μία επίθεση. Ως εκ τούτου, οι συγγραφείς προτείνουν την δημιουργία ενός διανύσματος V μεγέθους m, στο οποίο θα αποθηκεύονται οι τελευταίες m τιμές της Z_n. Οι τιμές αυτές λαμβάνονται με σταθερή χρονική απόσταση. Στην περίπτωση που όλες οι τιμές του διανύσματος είναι θετικές, τότε μπορεί να εξαχθεί με ασφάλεια το συμπέρασμα ότι το παρακολουθούμενο δίκτυο δέχεται επίθεση.

3.2 Χαρακτηρισμός δικτυακών ανωμαλιών με την μέθοδο των υποχώρων

Η μέθοδος των υποχώρων (subspace method) έχει προταθεί ως λύση για το πρόβλημα του εντοπισμού και του χαρακτηρισμού ανωμαλιών στην δικτυακή κίνηση από τους Lakhina, Crovella και Diot [26]. Η προτεινομένη λύση αξιοποιεί τις πληροφορίες που προκύπτουν από την παρακολούθηση των ροών σε ένα δίκτυο. Για κάθε ροή καταμετρούνται τα διακινούμενα bytes, πακέτα καθώς και οι συνδέσεις. Το

σύνολο των δεδομένων που προκύπτει είναι πολύ υψηλής τάξης και γι' αυτό οι συγγραφείς επιλέγουν να εφαρμόσουν σε αυτά την μέθοδο των υποχώρων.

Η μέθοδος των υποχώρων είναι μία μέθοδος μάθησης που επεξεργάζεται ταυτόχρονα δεδομένα από το σύνολο των ροών. Σκοπός της είναι να διαχωρίσει τα χαρακτηριστικά της κίνησης σε φυσιολογικά και μη. Φυσιολογικά είναι τα χαρακτηριστικά που δίνουν τα πιο συνηθισμένα χρονικά μοτίβα της δικτυακής κίνησης. Για τον υπολογισμό των χρονικών αυτών μοτίβων, οι συγγραφείς προτείνουν την μέθοδο PCA (Principal Component Analysis), η οποία αποσυνθέτει το σύνολο των δεδομένων σε ιδιοδιανύσματα. Τα χαρακτηριστικά που συναντώνται στα πρώτα ιδιοδιανύσματα αντιστοιχούν στην φυσιολογική κίνηση, ενώ τα υπόλοιπα στην μη φυσιολογική.

Η πειραματική αξιολόγηση δείχνει ότι η μέθοδος των υποχώρων μπορεί να εντοπίσει πολλά διαφορετικά είδη ανωμαλιών στην δικτυακή κίνηση. Στις ανωμαλίες αυτές περιλαμβάνονται σύμφωνα με τους συγγραφείς οι επιθέσεις άρνησης επίθεσης (από μία πηγή ή κατανεμημένες), η σάρωσης θυρών (port scanning), η απότομη αύξηση ζήτησης μίας υπηρεσίας (flash crowds), η διάδοση σκουληκιών υπολογιστών (worm propagation) και οι διακοπές δικτυακής κίνησης (outage). Για την αναγνώριση των ανωμαλιών είναι απαραίτητες και οι τρεις διαφορετικές χρονοσειρές (bytes, πακέτα και συνδέσεις), καθώς οι χρονοσειρές περιέχουν συμπληρωματικά δεδομένα.

3.3 Ανίχνευση και χαρακτηρισμός δικτυακών ανωμαλιών μέσω επιλεκτικής δειγματοληψίας

Λόγω των πολύ υψηλών ταχυτήτων μετάδοσης δεδομένων, η καταγραφή του συνόλου των πακέτων που εισέρχονται και εξέρχονται από ένα δίκτυο μπορεί να εισάγει σημαντικές καθυστερήσεις. Για τον λόγο αυτό, είναι απαραίτητο να υπάρχει κάποια δειγματοληψία. Ωστόσο, η τυχαία δειγματοληψία αλλοιώνει τα δεδομένα και ως εκ τούτου μειώνει την απόδοση των αλγορίθμων ανίχνευσης δικτυακών ανωμαλιών [27] [28]. Οι αλγόριθμοι που βασίζονται στον υπολογισμό της εντροπίας επηρεάζονται λιγότερο σε σχέση με αυτούς που βασίζονται στην καταμέτρηση του όγκου της κίνησης [29].

Λαμβάνοντας υπόψη τα παραπάνω, οι Ανδρουλάκης, Χατζηγιαννάκης και Παπαβασιλείου [30] αξιολογούν την αποτελεσματικότητα δύο μηχανισμών επιλεκτικής δειγματοληψίας σε διαφορετικά είδη δικτυακών ανωμαλιών. Η επιλεκτική δειγματοληψία υπερέχει της τυχαίας, καθώς οι συγγραφείς παρατηρούν ότι ένα μεγάλο μέρος της συνολικής πληροφορίας βρίσκεται σε λίγες μόνο ροές. Ο πρώτος μηχανισμός που προτείνουν διαλέγει πιο συχνά πακέτα από τις μικρές ροές, ενώ ο δεύτερος από τις μεγαλύτερες ροές.

Στην συνέχεια, οι συγγραφείς δοκιμάζουν τους παραπάνω μηχανισμούς σε δικτυακές κινήσεις με διαφορετικές ανωμαλίες. Η μετρική που επιλέγουν για την ανίχνευση των ανωμαλιών είναι η κανονικοποιημένη εντροπία και τα μεγέθη που παρακολουθούν είναι η IP διεύθυνση πηγής και προορισμού, η θύρα πηγής και προορισμού και το μέγεθος κάθε ροής. Η πειραματική αξιολόγηση δείχνει ότι η συχνότερη δειγματοληψία των μικρών ροών είναι κατάλληλη για την ανίχνευση DDoS επιθέσεων, διάδοσης σκουληκιών υπολογιστών (worm propagation) και σάρωσης θυρών (port scan). Αντιθέτως, η συχνότερη δειγματοληψία μεγαλύτερων ροών είναι καταλληλότερη για την ανίχνευση απότομης αύξησης κίνησης από έναν (alpha flow) ή περισσότερους (flash crowd) χρήστες.

Ο χαρακτηρισμός των ανωμαλιών γίνεται μέσω των μεγεθών, για τα οποία η εντροπία αλλάζει περισσότερο. Στην περίπτωση των DDoS επιθέσεων αλλάζει περισσότερο η εντροπία της IP και της θύρας προορισμού, καθώς υπάρχει έντονη αύξηση της κίνησης προς ένα συγκεκριμένο θύμα. Στην περίπτωση της διάδοσης σκουληκιών αλλάζει η εντροπία των IP προορισμού και της θύρας προορισμού, ενώ στην σάρωση θυρών αλλάζει η εντροπία των διευθύνσεων IP και της θύρας προέλευσης. Τέλος, στην περίπτωση της απότομης αύξησης κίνησης από χρήστες αλλάζει η εντροπία της IP προέλευσης, της θύρας προέλευσης και πιο έντονα η εντροπία του μεγέθους ροής.

Είδος ανωμαλίας κίνησης	Περιγραφή	Αλλαγή στην εντροπία
Επιθέσεις άρνησης υπηρεσίας (DDoS)	Επιθέσεις σε συγκεκριμένη υπηρεσία που την καθιστούν απροσπέλαστη στους χρήστες της	Αξιοσημείωτη πτώση στην εντροπία IP διεύθυνσης και πόρτας προορισμού. Σχεδόν καμία αλλαγή στις εντροπίες IP διεύθυνσης πηγής, πόρτας πηγής και μεγέθους ροών.
Διάδωση σκουληκιών υπολογιστών	Ένα αυτοαναπαραγόμενο πρόγραμμα που προσπαθεί να μολύνει άλλους υπολογιστές αξιοποιώντας κάποια συγκεκριμένη αδυναμία	Αξιοσημείωτη πτώση στην εντροπία IP διεύθυνσης πηγής και πόρτας προορισμού. Μικρή αύξηση εντροπίας IP διεύθυνσης προορισμού και πόρτας πηγής. Μικρή μείωση στην εντροπία μεγέθους ροών.
Σάρωση Θυρών	Αποστολή πακέτων ελέγχου σε ένα μεγάλο εύρος θυρών σε κάποιον συγκεκριμένη διεπαφή για τον έλεγχο διαθέσιμων υπηρεσιών	Αξιοσημείωτη πτώση στην εντροπία IP διεύθυνσης πηγής, IP διεύθυνσης προορισμού και πόρτας πηγής. Μικρή αύξηση εντροπίας πόρτας προορισμού. Μικρή μείωση στην εντροπία μεγέθους ροών.
Απότομη αύξηση ζήτησης από πολλούς χρήστες	Αύξηση ζήτησης κάποιας υπηρεσίας	Μικρή μείωση στην εντροπία όλων των χαρακτηριστικών.
Απότομη άυξηση ζήτησης από έναν χρήστη	Μικρός αριθμός ροών με μεγάλο όγκο πακέτων	Μικρή μείωση στην εντροπία IP διεύθυνσης πηγής και προορισμού. Σχεδόν καμία αλλαγή στην εντροπία πόρτας πηγής και προορισμού και μεγέθους ροών.

Πίνακας 3.1: Κατηγοριοποίηση ανωμαλιών της κίνησης με βάση αλλαγές στην εντροπία [30]

3.4 Μηχανισμός ανίχνευσης DDoS επιθέσεων στο επίπεδο δεδομένων με χρήση της P4

Ο προγραμματισμός στο επίπεδο δεδομένων αποτελεί μια πολλά υποσχόμενη τεχνολογία που επιτρέπει την υλοποιήση μηχανισμών γρήγορης ανίχνευσης και αντιμετώπισης δικτυακών επιθέσεων. Οι επικρατούντες μηχανισμοί εξαρτώνται από την δειγματοληψία ή την αντιγραφή ροών της κίνησης από συσκευές του δικτύου και την ανάλυση αυτής από εξωτερικούς ελεγκτές σε κεντρικό επίπεδο. Σε αυτό το πλαίσιο, στη δημοσίευση των Δημολιάνη, Παυλίδη και Μάγκλαρη [1] προτείνεται ένας μηχανισμός ανίχνευσης επιθέσεων DDoS που λειτουργεί εξ' ολοκλήρου στο επίπεδο δεδομένων. Ο μηχανισμός αυτός εξετάζει την δικτυακή κίνηση και υπολογίζει μετρικές για την παρακολούθησή της, αναλύει τις μετρικές αυτές ώστε να ανιχνεύσει πιθανές απειλές και ειδοποιεί εξωτερικά συστήματα αντιμετώπισης σε περίπτωση επίθεσης. Διατηρεί μια λίστα από παρακολουθούμενα υποδίκτυα για τα οποία είναι επιθυμητή η ανάλυση της κίνησης και η ανίχευση επιθέσεων. Για κάθε ένα από αυτά υπολογίζονται τρεις μετρικές: ο συνολικός αριθμός ροών, οι ροές ανά παρακολουθούμενο υποδίκτυο και η ασυμμετρία εισερχόμενων και εξερχόμενων πακέτων.

Για τον υπολογισμό των παραπάνω μετρικών, η παρακολούθηση της κίνησης χωρίζεται σε χρονοπαράθυρα (εποχές). Αυτός ο διαχωρισμός πραγματοποιείται προκειμένου να εξαχθούν τα ποσοτικά χαρακτηριστικά της κανονικής κίνησης, ώστε στη συνέχεια να μελετηθούν οι χρονικές μεταβολές τους και να εντοπιστούν οι μεγάλες αποκλίσεις από τις αναμενόμενες τιμές. Σε πρώτο στάδιο γίνεται η διαλογή των TCP και UDP πακέτων τα οποία θα προχωρήσουν στην επεξεργασία από τον μηχανισμό. Σύμφωνα με το [31], η πλειονότητα των επιθέσεων DDoS γίνεται με πακέτα που χρησιμοποιούν τα δύο αυτά πρωτόκολλα μεταφοράς. Στη συνέχεια γίνεται η επεξεργασία των πακέτων αυτών και οι υπολογισμοί των τριών μετρικών.

Η πρώτη μετρική βασίζεται στην καταμέτρηση των μοναδικών ροών της κίνησης. Ως ροή ορίζεται η πλειάδα (διεύθυνση IP πηγής, διεύθυνση IP προορισμού, πρωτόκολλο, πόρτα πηγής, πόρτα προορισμού). Σύμφωνα με το [32], οι επιθέσεις DDoS χαρακτηρίζονται από πολύ μεγάλη αύξηση του πλήθους των ροών. Για τον χαρακτηρισμό μιας ροής ως μοναδικής χρησιμοποιούνται πιθανοτικές δομές Bloom Filters [19]. Ο μηχανισμός χρησιμοποιεί τις μετρήσεις για το πλήθος μοναδικών ροών από τις προηγούμενες εποχές ως δείγμα εκπαίδευσης και προσαρμογής στις συνθήκες κίνησης. Με βάση αυτές συμπεραίνεται το κατώφλι το οποίο κατηγοριοποιεί τις νέες μετρήσεις για το πλήθος των μοναδικών ροών σε στατιστικά συμβατές με το δείγμα μάθησης, και άρα καλόβουλες, και σε outliers οπότε υποδεικνύει την ύπαρξη επίθεσης. Για την προσαρμογή στις συνθήκες κανονικής κίνησης και τον καθορισμό του αναμενόμενου πλήθους μοναδικών ροών, ο μηχανισμός χρησιμοποιεί την απλή εκθετική εξομάλυνση ή αλλιώς κινητό μέσο όρο εκθετικού βάρους (EWMA) [33],

$$M_n \;=\; a * TIF_n \;+\; (1-a) * M_{n-1} \tag{3.1}$$

με $M_1 = TIF_1$ και όπου TIF_n το πλήθος των μοναδικών ροών (total incoming flows) για την εποχή n. Ο συντελεστής a ονομάζεται συντελεστής εξομάλυνσης και ελέγχει την επίδραση της τρέχουσας μέτρησης του πλήθος των ροών στην αναμενόμενη τιμή για την επόμενη εποχή. Η επιλογή της συγκεκριμένης μεθόδου γίνεται αφενός για την εναρμόνιση με τις διακυμάνσεις των ροών κανονικής κίνησης, αφετέρου διότι υλοποιείται γρήγορα και αποδοτικά με χρήση απλών πράξεων. Το επίπεδο δεδομένων παρέχει πολύ περιορισμένες δυνατότητες ως προς τις αριθμητικές πράξεις [34], καθώς βασικός του ρόλος είναι η μεταγωγή πακέτων και όχι η εκτέλεση σύνθετων στατιστικών μεθόδων ανάλυσης της κίνησης. Η έλλειψη υποστήριξης για αριθμούς κινητής υποδιαστολής καθώς και για την πράξη της διαίρεσης, υποχρεώνει τον προγραμματιστή είτε στη χρήση απλούστερων μεθόδων ανάλυσης είτε στην αναγωγή των σύνθετων αριθμητικών πράξεων στις βασικές πράξεις της άθροισης, του πολλαπλασιασμού και της κύλισης (shifting). Για τον ορισμό του κατωφλίου αποδεκτών διακυμάνσεων των ροών της

κίνησης, είναι απαραίτητος και το υπολογισμός του κινητού μέσου όρου εκθετικού βάρους της διαφοράς αναμενόμενης και πραγματικής τιμής του πλήθους μοναδικών ροών,

$$D_n \ = \ a * |M_n \ - \ TIF_n| \ + \ (1 - a) * D_{n-1} \tag{3.2}$$

με $D_1 = 0$. Έτσι, το κατώφλι διαμορφώνεται ως $T_n = M_{n-1} + k{*}D_{n-1}$, όπου ο συντελεστής k ονομάζεται παράγοντας ευαισθησίας καθώς διευρύνει ή συρρικνώνει το εύρος των αποδεκτών διακυμάνσεων γύρω από την αναμενόμενη τιμή του πλήθους μοναδικών ροών. Αν το καταμετρούμενο πλήθος ροών ξεπεράσει το κατώφλι $TIF_n > T_n$, τότε ο μηχανισμός θεωρεί πως υφίσταται κάποια ανωμαλία στην κίνηση.

Για την περαιτέρω αποσαφήνιση του προορισμού των ροών της κίνησης αλλά και για την παροχή πιο συγκεκριμένης πληροφορίας για το πληττόμενο υποδίκτυο σε περίπτωση επίθεσης, υπολογίζεται η δεύτερη μετρική - το πλήθος μοναδικών ροών ανά υποδίκτυο. Σε κάθε εποχή καταμετρώνται οι ροές ανάλογα με το παρακολουθούμενο δίκτυο προορισμού στο οποίο ανήκουν (SIF_n^i - ith subnet's incoming flows). Για την απόφαση περί ύπαρξης ανωμαλίας, εξετάζεται το ποσοστό ροών κάθε παρακολουθούμενου υποδικτύου επί των συνολικών ροών και συγκρίνεται με ένα αναμενόμενο προϋπολογισμένο ποσοστό. Ο υπολογισμός αυτών των ποσοστών γίνεται σε προγενέστερο χρόνο αναλύοντας τα χαρακτηριστικά της κανονικής κίνησης.

$$\frac{SIF_n^i}{TIF_n} > f \tag{3.3}$$

Προκειμένου να αποφευχθεί η κατηγοριοποίηση ενός υποδικτύου ως υπό επίθεση ενώ δέχεται μεγάλη κίνηση στην οποία ανταποκρίνεται επιτυχώς με απαντήσεις, υπολογίζεται η τρίτη μετρική - η ασυμμετρία εισερχόμενων και εξερχόμενων πακέτων. Σε κάθε εποχή καταμετρούνται τα πακέτα και προς τις δύο κατευθύνσεις και υπολογίζεται ο δείκτης CR_n^i (current packet symmetry ratio of subnet i during epoch n) των εισερχόμενων προς τα εξερχόμενα πακέτα. Στη συνέχεια, ελέγχεται ότι ο δείκτης αυτός δεν υπερβαίνει τον δείκτη κανονικής συμμετρίας NR^i (normal packet symmetry ratio of subnet i) περισσότερο από ένα κατώφλι r. Το κατώφλι αυτό καθώς και ο δείκτης κανονικής συμμετρίας έχουν προϋπολογιστεί με βάση την εμπειρία αλλά και την ανάλυση της κανονικής κίνησης.

$$\frac{CR_n^i}{NR^i} > r \tag{3.4}$$

Οι συγγραφείς υλοποίησαν τον προτεινόμενο μηχανισμό με χρήση της γλώσσας P4 [7]. Ο έλεγχος της εγκυρότητας ανίχνευσης πραγματοποιήθηκε σε μία κάρτα Netronome Agilo CX SmartNIC [35] ταχύτητας 10GbE. Ως πηγή και προορισμό της κίνησης ελέγχου του μηχανισμού χρησιμοποίησαν δύο εικονικά μηχανήματα τα οποία συνδέονται με την κάρτα Netronome. Η καλόβουλη κίνηση αναπαράγεται από αρχεία καταγραφών του WIDE project [36], από τα οποία έχουν ορισθεί ως παρακολουθούμενα τα 255 /24 υποδίκτυα με την μεγαλύτερη κίνηση. Για την κακόβουλη κίνηση χρησιμοποιήσαν αρχεία καταγραφών από την μελέτη "Booters" [37]. Η διάρκεια κάθε εποχής ορίστηκε στο 1 δευτερόλεπτο.

Τα πειράματα που εκτελέστηκαν αφορούν διάφορους ρυθμούς αναπαραγωγής τόσο της καλόβουλης όσο και της κακόβουλης κίνησης προκειμένου να αναλυθεί η ακρίβεια ανίχνευσης καθώς και η ταχύτητα προώθησης πακέτων από τον μηχανισμό. Χρησιμοποιώντας παραλλαγές της προτεινόμενης υλοποιήσης, οι συγγραφείς επιτυγχάνουν σε όλες τις περιπτώσεις πειραμάτων άμεση ανίχνευση της επίθεσης και του πληττόμενου υποδικτύου εντός μίας εποχής, με τα ποσοστά ακρίβειας να κυμαίνονται από 85% μέχρι 100%. Επίσης, αποδεικνύουν ότι ο πρόσθετος επεξεργαστικός φόρτος που προσθέτει ο μηχανισμός ανίχνευσης επιδρά στην ταχύτητα προώθησης μόνο μετά τα 5 Mpps, κίνηση η οποία ξεπερνάει κατά πολύ τη συνηθισμένη κίνηση των 1 - 2 Mpps που παρουσιάζουν δίκτυα με συνδέσεις ταχύτητας 10 Gbit [38].

Κεφάλαιο 4

Ο μηχανισμός ανίχνευσης και αναγνώρισης επιθέσεων

Οι επεκτάσεις του μηχανισμού ανίχνευσης που έχει προταθεί στο [1] αφορούν την προσθήκη λειτουργίας καταμέτρησης των παραγόμενων ειδοποιήσεων, την ενσωμάτωση του μηχανισμού HashPipe [39], ο οποίος αναδεικνύει τις top k ροές κίνησης με βάση τον όγκο τους, και την εφαρμογή δειγματοληψίας για την επιλογή των πακέτων που θα επεξεργασθούν από τους δύο μηχανισμούς.

Ο πλήρης κώδικας του μηχανισμού είναι διαθέσιμος στο GitHub [40] και στο παράρτημα Β΄.

4.1 Επικεφαλίδες και πρωτόκολλο μεταφοράς

Κατά την είσοδο των πακέτων στη δικτυακή συσκευή, εκτελείται η εξαγωγή και ανάλυση των επικεφαλίδων η οποία αποτελεί βασική προϋπόθεση για όλες τις λειτουργίες (προώθησης και ανίχνευσης) που θα αναφερθούν στη συνέχεια. Για τον μηχανισμό απαιτείται η επιλογή πακέτων που χρησιμοποιούν πρωτόκολλο μεταφοράς **TCP** ή **UDP**. Ο μηχανισμός, ξεκινώντας πάντα από την αρχική κατάσταση `start`, εξάγει από το πακέτο την επικεφαλίδα Ethernet (**parse_ethernet**). Στη συνέχεια, ανάλογα με τον τύπο του πεδίου *type* προχωράει είτε στην εξαγωγή της επικεφαλίδας IP (**parse_ipv4, type = 0x0800**) είτε στην κατάσταση αποδοχής (`accept`). Σε περίπτωση που η επικεφαλίδα IP περιέχει options (πεδίο $ihl > 5$), αυτά εξάγονται σε άλλη επικεφαλίδα (**parse_ipv4_options**) προκειμένου στο επόμενο βήμα να ερμηνευθούν σωστά οι πόρτες πηγής και προορισμού την επικεφαλίδας του πρωτοκόλλου μεταφοράς. Αυτές εξάγονται όταν το πεδίο protocol της επικεφαλίδας IP έχει τιμή 6 (TCP) ή 17 (UDP). Στο σημείο αυτό πρέπει να σημειωθεί ότι κανένα πακέτο δεν απορρίπτεται σε κανένα βήμα της διαδικασίας εξαγωγής των επικεφαλίδων. Τα πακέτα που ενδιαφέρουν τον μηχανισμό είναι όσα έχουν έγκυρη την επικεφαλίδα πρωτοκόλλου μεταφοράς άρα κατ' επέκταση και τις επικεφαλίδες Ethernet και IP. Η διαδικασία αυτή φαίνεται στον κώδικα 4.1

```
1  parser HeaderParser(packet_in packet, out headers hdr, inout metadata meta, inout
       standard_metadata_t standard_metadata) {
2
3      state start {
4          transition parse_ethernet;
5      }
6
7      state parse_ethernet {
8          packet.extract(hdr.ethernet);
9          transition select(hdr.ethernet.etherType) {
10             EtherType.IPV4: parse_ipv4;
11             default: accept;
12         }
13     }
14
15     state parse_ipv4 {
```

```
16          packet.extract(hdr.ipv4);
17          transition select(hdr.ipv4.ihl) {
18              5: check_ports;
19              _: parse_ipv4_options;
20          }
21      }
22
23      state parse_ipv4_options {
24          packet.extract(hdr.ipv4_options, (bit<32>)(((bit<16>)hdr.ipv4.ihl - 5) * 32) );
25          transition check_ports;
26      }
27
28      state check_ports {
29          transition select(hdr.ipv4.protocol) {
30              ProtoType.TCP: parse_tcp_udp_port;
31              ProtoType.UDP: parse_tcp_udp_port;
32              default: accept;
33          }
34      }
35
36      state parse_tcp_udp_port {
37          packet.extract(hdr.ports);
38          transition accept;
39      }
40
41  }
```

Κώδικας 4.1: Parsing - Εξαγωγή επικεφαλίδων

4.2 Προώθηση/Δρομολόγηση των πακέτων

Η βασική λειτουργία του επιπέδου δεδομένων μιας δικτυακής συσκευής είναι η προώθηση (switching) ή δρομολόγηση (routing) των πακέτων. Δεδομένου ότι ο σχεδιασμός και η υλοποίηση της συγκεκριμένης λειτουργίας εξαρτώνται σε μεγάλο βαθμό από το σημείο τοποθέτησης σε μια δικτυακή υποδομή αλλά και από τους στόχους του οργανισμού που τη χρησιμοποιεί, επιβάλλεται η ανεξαρτησία της υλοποίησής της από τον υπόλοιπο μηχανισμό ανίχνευσης. Για το λόγο αυτό, στο πλαίσιο της παρούσας διπλωματικής εργασίας επιλέχθηκε η απλούστερη δυνατή υλοποίηση, αυτή της αμφίδρομης προώθησης μεταξύ δύο σημείων α και β. Τα σημεία αυτά αντιστοιχίζονται σε δύο θύρες της δικτυακής συσκευής ($\alpha \rightarrow 1$ και $\beta \rightarrow 2$). Τα μεταδεδομένα **standard_metadata** κάθε εισερχόμενου πακέτου περιέχουν πληροφορία για την θύρα εισόδου (**ingress_port**) και με χρήση του κώδικα 4.2 θέτουμε την κατάλληλη θύρα εξόδου (**egress_spec**).

```
1  control BasicDeviceRole(inout standard_metadata_t standard_metadata) {
2      action l2_forward() {
3          if (standard_metadata.ingress_port == 1) {
4              standard_metadata.egress_spec = 2;
5          }
6          else {
7              standard_metadata.egress_spec = 1;
8          }
9      }
10
11     apply {
12         l2_forward();
13     }
14 }
```

Κώδικας 4.2: Layer 2 Forwarding

4.3 Ταυτοποίηση παρακολουθούμενων υποδικτύων

Ο μηχανισμός, προκειμένου να προσαρμόζεται στις ανάγκες του εκάστοτε οργανισμού που τον χρησιμοποιεί, έχει σχεδιαστεί ώστε να επεξεργάζεται μόνο τα πακέτα που ανήκουν σε συγκεκριμένα υποδίκτυα. Τον ορισμό των παρακολουθούμενων αυτών υποδικτύων αναλαμβάνει ο διαχειριστής του δικτύου. Με αυτόν τον τρόπο μπορεί να διαχωριστεί η κίνηση σε εσωτερική (μεταξύ παρακολουθούμενων υποδικτύων), εισερχόμενη (από άγνωστα προς παρακολουθούμενα υποδίκτυα) και εξερχόμενη (από παρακολουθούμενα προς άγνωστα υποδίκτυα).

Τα παρακολουθούμενα δίκτυα αποθηκεύονται σε δύο **tables** ως κανόνες της μορφής `matched_key` ⇒ `action, data`. Μετά την εξαγωγή των επικεφαλίδων, τα TCP και UDP πακέτα (έγκυρες επικεφαλίδες ethernet, ipv4 & ports) και συγκεκριμένα οι διευθύνσεις IP πηγής και προορισμού ελέγχονται για ομοιότητα με τα κλειδιά των πινάκων με την τεχνική ternary matching. Με βάση αυτή την τεχνική οι δύο τιμές συγκρίνονται αφού πρώτα εφαρμοστεί σε αυτές μία μάσκα. Το κλειδί `matched_key` απαρτίζεται από το υποδίκτυο και την αντίστοιχη μάσκα. Αν υπάρξει αντιστοίχιση τότε εκτελείται η διαδικασία action που αναθέτει στο πεδίο **meta.src_subnet_code** ή **meta.dst_subnet_code** των μεταδεδομένων `meta` τον κωδικό αριθμό που υποδεικνύουν τα δεδομένα (data) της εγγραφής όπως φαίνεται στον κώδικα 4.3. Ο κωδικός αυτός αριθμός είναι μοναδικός για κάθε παρακολουθούμενο υποδίκτυο και θα χρησιμοποιηθεί σε διαδικασίες που θα παρουσιαστούν στη συνέχεια. Αν δεν υπάρξει αντιστοίχιση, τότε ενεργοποιείται ο default κανόνας που αναθέτει στο αντίστοιχο πεδίο των μεταδεδομένων μία ειδική τιμή `CODE_SUBNET_UNMONITORED` που υποδηλώνει ότι το το υποδίκτυο δεν παρακολουθείται. Ο λόγος που απαιτούνται δύο πίνακες με τα ίδια δεδομένα είναι η ανάγκη διαφοροποίησης μεταξύ του action που αναθέτει τον κωδικό του υποδικτύου πηγής και αντίστοιχα του υποδικτύου προορισμού.

```
1  action get_src_subnet_code(bit<32> subnet_code) {
2      meta.src_subnet_code = subnet_code;
3  }
4
5  action get_dst_subnet_code(bit<32> subnet_code) {
6      meta.dst_subnet_code = subnet_code;
7  }
8
9  table monitored_subnets_src {
10     key = {
11         hdr.ipv4.srcAddr : ternary;
12     }
13     actions = {
14         get_src_subnet_code;
15         NoAction;
16     }
17     size = MONITORING_CAPACITY + 2;
18     default_action = NoAction();
19 }
20
21 table monitored_subnets_dst {
22     key = {
23         hdr.ipv4.dstAddr : ternary;
24     }
25     actions = {
26         get_dst_subnet_code;
27         NoAction;
28     }
29     size = MONITORING_CAPACITY + 2;
30     default_action = NoAction();
31 }
32
33 apply {
34     monitored_subnets_src.apply();
35     monitored_subnets_dst.apply();
```

```
}
```

Κώδικας 4.3: Ταυτοποίηση παρακολουθούμενων υποδικτύων

4.4 Χρονοσήμανση

Η λειτουργία του μηχανισμού ανίχνευσης βασίζεται στην καταγραφή και ανάλυση των μεταβολών της κίνησης. Η κίνηση καταγράφεται μέσα σε χρονοπαράθυρα τα οποία στο εξής θα αποκαλούνται **εποχές**. Αφού τα πακέτα περάσουν από το στάδιο της ταυτοποίησης παρακολουθούμενων δικτύων, τότε αν τουλάχιστον ένας από του κωδικούς αριθμούς `meta.src_subnet_code` και `meta.dst_subnet_code` αντιστοιχεί σε παρακολουθούμενο δίκτυο, το πακέτο θα λάβει μια σήμανση εποχής στο πεδίο των μεταδεδομένων **meta.epoch**. Πρόκειται για έναν αύξοντα αριθμό που κατατάσσει το πακέτο σε ένα συγκεκριμένο χρονικό διάστημα παρακολούθησης. Κάθε φορά η τιμή της τρέχουσας εποχής καθώς επίσης και η χρονική στιγμή έναρξής της αποθηκεύονται σε καταχωρητές. Το εισερχόμενο πακέτο περιέχει πληροφορία για την χρονική στιγμή άφιξής του στη δικτυακή συσκευή. Την πληροφορία αυτή παρέχει η ίδια η συσκευή στο πεδίο μεταδεδομένων **standard_metadata.ingress_global_timestamp**. Αυτή στη συνέχεια συγκρίνεται με την αποθηκευμένη χρονική στιγμή έναρξης της τρέχουσας εποχής. Αν διαπιστωθεί ότι αυτές απέχουν παραπάνω από ενα προκαθορισμένο χρονικό διάστημα (interval), τότε ενημερώνονται οι καταχωρητές τρέχουσας εποχής και χρονικής στιγμής έναρξής της και το πακέτο λαμβάνει την τιμή της νέας εποχής. Σε αντίθετη περίπτωση το πακέτο λαμβάνει την τιμή της τρέχουσας εποχής.

Με τη χρήση χρονοπαραθύρων επιτυγχάνουμε τη σκιαγράφηση των ποσοτικών μεταβολών της κίνησης οι οποίες θα χρησιμοποιηθούν αργότερα. Στον κώδικα 4.4 φαίνεται η διαδικασία χρονοσήμανσης η οποία μαζί με τη διαδικασία ταυτοποίησης των παρακολουθούμενων υποδικτυών αποτελούν το πρώτο βήμα του μηχανισμού.

```
control PacketTagging(inout headers hdr, inout metadata meta, inout standard_metadata_t
    standard_metadata, in bool should_reset_epoch, out bool will_initiate_epoch_reset) {
    register<bit<32>>(1) r_global_epoch;
    register<bit<64>>(1) r_last_ingress_timestamp;

    apply {
        will_initiate_epoch_reset = false;

        if (meta.src_subnet_code != CODE_SUBNET_UNMONITORED || meta.dst_subnet_code !=
    CODE_SUBNET_UNMONITORED) {
            bit<32> current_epoch;
            bit<64> last_timestamp;
            bit<64> current_timestamp;

            r_global_epoch.read(current_epoch, 0);

            r_last_ingress_timestamp.read(last_timestamp, 0);
            current_timestamp = (bit<64>) standard_metadata.ingress_global_timestamp;

            if (last_timestamp == 0) {
                r_last_ingress_timestamp.write(0, current_timestamp);
                last_timestamp = current_timestamp;
            }

            if ((current_timestamp - last_timestamp > EPOCH_DURATION_MICROSEC) &&
    should_reset_epoch) {
                r_last_ingress_timestamp.write(0, last_timestamp + EPOCH_DURATION_MICROSEC);
                r_global_epoch.write(0, current_epoch + 1);

                meta.epoch = current_epoch + 1;
                will_initiate_epoch_reset = true;
            }
```

```
30              else {
31                  meta.epoch = current_epoch;
32              }
33          }
34      }
35  }
```

Κώδικας 4.4: Χρονοσήμανση πακέτων

4.5 Μηχανισμός ανίχνευσης

4.5.1 Μοναδικές ροές

Ο πρώτος επιμέρους μηχανισμός ανίχνευσης βασίζεται στην καταμέτρηση των μοναδικών ροών της κίνησης στις οποίες εμπλέκονται τα παρακολουθούμενα δίκτυα. Υπενθυμίζεται ότι ως ροή ορίζεται η πλειάδα **(IP πηγής, IP προορισμού, πρωτόκολλο, πόρτα πηγής, πόρτα προορισμού)**. Επειδή ο αριθμός τους είναι αυθαίρετα μεγάλος, ειδικά στις περιπτώσεις όπου υπάρχει κάποια επίθεση, δεν είναι δυνατή η χρήση κάποιας δομής πίνακα ή set. Επιπλέον, εφόσον η P4 δεν επιτρέπει επαναληπτικούς βρόχους, οποιαδήποτε δομή που βασίζεται στην σειριακή αναζήτηση για την εύρεση και προσθήκη στοιχείων δεν μπορεί να χρησιμοποιηθεί. Λύση σε αυτό το πρόβλημα αποτελούν οι πιθανοτικές δομές Bloom Filters και Sketches που παρουσιάστηκαν στο κεφάλαιο 2. Αυτές επιτρέπουν άμεσα με χρήση συναρτήσεων κατακερματισμού την λήψη απόφασης για το αν κάποια ροή είναι καινούρια, οπότε θα καταμετρηθεί ως μοναδική, ή έχει εμφανιστεί ξανά σε προηγούμενα πακέτα.

Ο μηχανισμός αρχικά ελέγχει αν η ροή κατευθύνεται προς κάποιο παρακολουθούμενο δίκτυο αλλά δεν πηγάζει από κάποιο από αυτά καθώς ενδιαφέρει μόνο η εισερχόμενη κίνηση. Σε αυτό το στάδιο τόσο η εσωτερική όσο και εξερχόμενη κίνηση θεωρούνται καλόβουλες. Για τον έλεγχο μοναδικότητας χρησιμοποιούνται τρεις πίνακες από registers (**r_uniqueness_sketch_{1,2,3}**) οι οποίοι μοιάζουν με δομή count-min sketch (§2.4.2) αλλά λειτουργούν συνεργατικά ως μία δομή bloom filter (§2.4.1). Τρεις συναρτήσεις κατακερματισμού (crc32, crc16, csum16) αντιστοιχίζουν την εισερχόμενη ροή σε θέσεις των πινάκων. Σε αυτές αποθηκεύονται οι τιμές των εποχών κατά τις οποίες προσπαλάστηκαν τελευταία φορά. Αν έστω και μία από τις θέσεις αντιστοιχεί σε παλαιότερη εποχή (η σύγκριση $register\_value == meta.epoch$ αποδίδει τιμή 0) τότε η ροή είναι καινούρια οπότε καταμετράται στο σύνολο των μοναδικών ροών (**r_epoch_flow_counter**) και οι τρείς πίνακες ενημερώνονται κατάλληλα, γράφοντας στις θέσεις των πινάκων την τιμή της τρέχουσας εποχής [2]. Γίνεται χρήση του αύξοντα αριθμού των εποχών αντί τιμών 0 και 1 στους registers καθώς δεν υποστηρίζεται ο μαζικός μηδενισμός πινάκων που απαιτείται στην αρχή κάθε νέας εποχής.

Το συνολικό πλήθος των μοναδικών ροών συγκρίνεται στη συνέχεια με ένα κατώφλι. Αν η τρέχουσα τιμή του πλήθους των μοναδικών ροών ξεπεράσει το κατώφλι $T(t)$ τότε ενεργοποιείται η πρώτη σημαία **meta.flag_total_flow_count** η οποία σηματοδοτεί ανωμαλία της συνολικής κίνησης για την τρέχουσα εποχή.

Για τον υπολογισμό του κατωφλίου χρησιμοποιούνται οι τεχνικές **EWMA** και **EWMD** [33]. Προκύπτει ως η προβλεπόμενη τιμή του πλήθους των ροών με χρήση της μεθόδου εκθετικού κινητού μέσου όρου προσαυξημένο κατά έναν παράγοντα απόκλισης. Αυτός υπολογίζεται πολλαπλασιάζοντας την τιμή του εκθετικού κινητού μέσου όρου μέσων διαφορών με έναν παράγοντα ευαισθησίας k [41]. Οι εξισώσεις 4.1, 4.2 και 4.3 περιγράφουν τον υπολογισμό των μεθόδων EWMA και EWMD καθώς επίσης και του κατωφλίου. Οι υπολογισμοί αυτοί πραγματοποιούνται κατά την έναρξη μιας νέας εποχής και ισχύουν καθόλη τη διάρκειά της.

$$M(t) = a * total\_unique\_flows(t-1) + (1-a) * M(t-1) \tag{4.1}$$

$$D(t) = a * |M(t) - total\_unique\_flows(t-1)| + (1-a) * D(t-1) \tag{4.2}$$

$$T(t) = M(t) + k * D(t) \tag{4.3}$$

Ο παράγοντας a ονομάζεται συντελεστής εξομάλυνσης και λαμβάνει τιμές $0 < a < 1$. Όσο μεγαλύτερη η τιμή του a τόσο περισσότερο επηρεάζει η τρέχουσα τιμή του πλήθους συνολικών ροών την προβλεπόμενη τιμή και άρα τόσο πιο γρήγορα προσαρμόζεται ο μηχανισμός στις απότομες μεταβολές της κίνησης. Γι'αυτό το λόγο επιλέγονται κυρίως μικρές τιμές για τον συντελεστή a. Ο παράγοντας k ονομάζεται συντελεστής ευαισθησίας και εκφράζει την ανοχή του μηχανισμού σε αποκλίσεις από την προβλεπόμενη τιμή. Όσο μεγαλύτερη η τιμή του συντελεστή, τόσο μειώνεται η ευαισθησία καθώς το όριο των αποδεκτών αποκλίσεων αυξάνεται. Εν γένει, και οι δύο συντελεστές είναι δεκαδικοί αριθμοί. Ωστόσο η P4 δεν υποστηρίζει δεκαδικούς αριθμούς και πράξεις κινητής υποδιαστολής (εν προκειμένω διαίρεση). Αυτό το πρόβλημα μπορεί να παρακαμφθεί ορίζοντας για τους συντελεστές τιμές κλασμάτων δυνάμεων του 2. Έτσι ο πολλαπλασιασμός με δεκαδικό αριθμό αντικαθίσταται από πολλαπλασιασμό ακεραίων και shifting. Οι σχέσεις 4.1, 4.2 και 4.3 μετασχηματίζονται στις 4.4, 4.5 και 4.6 αντίστοιχα. Το πρόβλημα υπολογισμού μαθηματικών εκφράσεων που περιέχουν διαιρέσεις και λογαρίθμους έχει αναλυθεί και σε σχετικές μελέτες [34]

$$M(t) = [a' * total\_unique\_flows(t-1) + (2^n - a') * M(t-1)] >> n \tag{4.4}$$

$$D(t) = \{a' * |M(t) - total\_unique\_flows(t-1)| + (2^n - a') * D(t-1)\} >> n \tag{4.5}$$

$$T(t) = M(t) + [k' * D(t)] >> m \tag{4.6}$$

```
bit<32> f;
bit<32> a;
bit<32> m;
bit<32> s;

r_epoch_flow_counter.read(f, 0);
r_threshold_a.read(a, 0);
r_threshold_m.read(m, 0);
r_threshold_s.read(s, 0);

m = ((a * f) + ((256 - a) * m)) >> 8;
s = (m < f) ? ((a * (f - m)) + ((256 - a) * s)) : ((a * (m - f)) + ((256 - a) * s));
s = s >> 8;

r_threshold_m.write(0, m);
r_threshold_s.write(0, s);

r_epoch_flow_counter.write(0, 0);
```

Κώδικας 4.5: Υπολογισμός κατωφλίου στην αρχή της εποχής

```
if (meta.src_subnet_code == CODE_SUBNET_UNMONITORED && meta.dst_subnet_code !=
    CODE_SUBNET_UNMONITORED) {
    bit<32> hash_1;
    bit<32> hash_2;
    bit<32> hash_3;

    hash(hash_1, HashAlgorithm.crc32, (bit<32>) 0, { hdr.ipv4.srcAddr,hdr.ipv4.dstAddr,hdr.ipv
    4.protocol,hdr.ports.srcPort,hdr.ports.dstPort }, (bit<32>)65536);
    hash(hash_2, HashAlgorithm.crc16, (bit<32>) 0, { hdr.ipv4.srcAddr,hdr.ipv4.dstAddr,hdr.ipv
    4.protocol,hdr.ports.srcPort,hdr.ports.dstPort }, (bit<32>)65536);
    hash(hash_3, HashAlgorithm.csum16, (bit<32>) 0, { hdr.ipv4.srcAddr,hdr.ipv4.dstAddr,hdr.
    ipv4.protocol,hdr.ports.srcPort,hdr.ports.dstPort }, (bit<32>)65536);
```

```
9
10      bit<32> last_epoch;
11      bit<1> is_new_flow;
12
13      is_new_flow = 0;
14
15      r_uniqueness_sketch_1.read(last_epoch, hash_1);
16      if (last_epoch != meta.epoch) {
17          is_new_flow = 1;
18          r_uniqueness_sketch_1.write(hash_1, meta.epoch);
19      }
20
21      r_uniqueness_sketch_2.read(last_epoch, hash_2);
22      if (last_epoch != meta.epoch) {
23          is_new_flow = 1;
24          r_uniqueness_sketch_2.write(hash_2, meta.epoch);
25      }
26
27      r_uniqueness_sketch_3.read(last_epoch, hash_3);
28      if (last_epoch != meta.epoch) {
29          is_new_flow = 1;
30          r_uniqueness_sketch_3.write(hash_3, meta.epoch);
31      }
32
33      bit<32> total_unique_flows;
34      bit<16> subnet_total_flows;
35
36      r_epoch_flow_counter.read(total_unique_flows, 0);
37      total_unique_flows = total_unique_flows + (bit<32>)is_new_flow;
38      r_epoch_flow_counter.write(0, total_unique_flows);
39
40      if (is_new_flow == 1) {
41
42          r_last_flow_epoch_per_subnet.read(last_epoch, meta.dst_subnet_code);
43
44          if (last_epoch == meta.epoch) {
45              r_total_flows_per_subnet.read(subnet_total_flows, meta.dst_subnet_code);
46              subnet_total_flows = subnet_total_flows + 1;
47          }
48          else {
49              r_last_flow_epoch_per_subnet.write(meta.dst_subnet_code, meta.epoch);
50              subnet_total_flows = 1;
51          }
52
53          r_total_flows_per_subnet.write(meta.dst_subnet_code, subnet_total_flows);
54
55      }
56      else {
57          r_total_flows_per_subnet.read(subnet_total_flows, meta.dst_subnet_code);
58      }
59
60      bit<32> k;
61      bit<32> m;
62      bit<32> s;
63
64      r_threshold_m.read(m, 0);
65      r_threshold_s.read(s, 0);
66      r_threshold_k.read(k, 0);
67
68      if ( total_unique_flows > m + ((k * s) >> 4) ) {
69          meta.flag_total_flow_count = 1;
70      }
71  }
```

Κώδικας 4.6: Υπολογισμός μοναδικών ροών και έλεγχος κατωφλίου

Η καταμέτρηση και ο έλεγχος της διακύμανσης των μοναδικών ροών αποτελεί πολύ σημαντική μετρική, ειδικά στην περίπτωση των επιθέσεων DDoS που η συγκεντρωμένη κίνηση προς κάποιον προορισμό από πληθώρα πηγών αναμένεται να αυξήσει κατακόρυφα το πλήθος των ροών. Ωστόσο η μετρική αυτή είναι

πολύ ευαίσθητη και στις ριπές καλόβουλης κίνησης που μπορεί να συμβούν κατά την κανονική λειτουργία του δικτύου, οδηγώντας έτσι σε άσκοπη ενεργοποίηση του μηχανισμού.

4.5.2 Καταμέτρηση ροών ανά υποδίκτυο

Για να υπάρξει μια πιο στοχευμένη ανάλυση των μοναδικών ροών κίνησης, ο δεύτερος επιμέρους μηχανισμός ανίχνευσης εστιάζει στον καταμερισμό της κίνησης στα παρακολουθούμενα υποδίκτυα. Έτσι, όταν μια ροή προσμετράται στις συνολικές ως μοναδική, αυξάνεται ταυτόχρονα και ο μετρητής πλήθους ροών για το υποδίκτυο προορισμού της. Για τον σκοπό αυτό χρησιμοποιείται ένας πίνακας μετρητών **r_total_flows_per_subnet**, μεγέθους όσο και το πλήθος των παρακολουθούμενων δικτύων. Ο κωδικός αριθμός που έχει λάβει το πακέτο για το υποδίκτυο προορισμού του (§4.3) **meta.dst_subnet_code** λειτουργεί ως δείκτης για τον πίνακα αυτόν. Κάθε φορά η νέα ροή αυξάνει τον μετρητή της θέσης του πίνακα που υποδεικνύεται από αυτόν τον κωδικό. Ένας δεύτερος βοηθητικός πίνακας (**r_last_flow_epoch_per_subnet**) αποθηκεύει τον αύξοντα αριθμό της εποχής κατά την οποία προσπελάστηκε το αντίστοιχο πεδίο του πίνακα r_total_flows_per_subnet. Χρησιμοποιείται ώστε ο μηχανισμός να γνωρίζει αν τα δεδομένα που περιέχει το πεδίο είναι έγκυρα (ανοίκουν στην τρέχουσα εποχή) ή πρέπει να αρχικοποιηθούν (ανήκουν σε παλαιότερη εποχή). Η χρήση του είναι απαραίτητη καθώς η P4 δεν υποστηρίζει μαζικό μηδενισμό δομών πινάκων καταχωρητών.

Στη συνέχεια, το πλήθος ροών που καταμετρήθηκε για το υποδίκτυο προορισμού του πακέτου συγκρίνεται με ένα κατώφλι το οποίο προκύπτει ως ένα ποσοστό **factor** του συνολικού πλήθους των μοναδικών ροών. Αν η τιμή ξεπεράσει το κατώφλι τότε ενεργοποιείται και η δεύτερη σημαία **meta.flag_subnet_flow_count** η οποία σηματοδοτεί ανωμαλία της κίνησης ενός συγκεκριμένου υποδικτύου για την τρέχουσα εποχή. Ο συντελεστής factor αποτελεί το ποσοστό επί της συνολικής κίνησης που αντιστοιχεί στο συγκεκριμένο υποδίκτυο και υπάρχει ένας για κάθε παρακολουθούμενο υποδίκτυο. Οι τιμές τους προκύπτουν από προγενέστερη ανάλυση της κίνησης έχοντας γνώση του συνόλου των επιθυμητών παρακολουθούμενων δικτύων.

Με τη χρήση αυτής της μετρικής επιτυγχάνεται καλύτερη εποπτεία της συμπεριφοράς της κίνησης του δικτύου. Ο καταμερισμός της συνολικής κίνησης βοηθάει στην εξομάλυνση του φαινομένου ενεργοποίησης του μηχανισμού από ριπές καλόβουλης κίνησης με μεγαλύτερη επιτυχία σε σχέση με τον πρώτο μηχανισμό. Ωστόσο δεν πετυχαίνει ικανοποιητικά αποτελέσματα στην περίπτωση που οι ριπές κατευθύνονται προς συγκεκριμένο μόνο υποδίκτυο αλλά και σε περιπτώσεις που η κίνηση επίθεσης είναι κατάλληλα καταμερισμένη σε όλα τα παρακολουθούμενα υποδίκτυα.

```
1 bit<32> subnet_ratio_factor;
2 r_ratio_factor_per_subnet.read(subnet_ratio_factor, meta.dst_subnet_code);
3
4 if ( (bit<32>)subnet_total_flows > ((subnet_ratio_factor * total_unique_flows) >> 8) ) {
5     meta.flag_subnet_flow_count = 1;
6 }
```

Κώδικας 4.7: Ροές ανά υποδίκτυο και έλεγχος κατωφλίου

4.5.3 Ασυμμετρία εισερχόμενης - εξερχόμενης κίνησης

Ο τρίτος επιμέρους μηχανισμός βασίζεται στην καταμέτρηση και ανάλυση του πλήθους των εισερχόμενων και εξερχόμενων πακέτων ανά υποδίκτυο και ανα πρωτόκολλο. Κάθε πακέτο που εισέρχεται στον μηχανισμό προσμετράται ανάλογα με τους κωδικούς αριθμούς meta.src_subnet_code και meta.dst_subnet_code που έχει λάβει κατά το στάδιο της ταυτοποίησης αλλά και με το πρωτόκολλο μεταφοράς (hdr.ipv4.protocol). Οι τιμές αποθηκεύονται στον πίνακα **r_assymetry_packet_counter** ο οποίος χωρίζεται σε τέσσερα μέρη (TCP διεύθυνση πηγής - προορισμού και UDP διεύθυνση πηγής - προορισμού) καθένα μεγέθους όσο και το πλήθος των παρακολουθούμενων δικτύων. Και πάλι

χρησιμοποιείται ο βοηθητικός πίνακα **r_assymetry_last_epoch** για τον έλεγχο της εποχής κατά την οποία προσπελάστηκε τελευταία φορά το εκάστοτε πεδίο.

Μετά την καταμέτρηση συγκρίνονται οι τιμές του πλήθους εισερχόμενων και εξερχόμενων πακέτων για το υποδίκτυο προορισμού του πακέτου και αν η τιμή τους υπερβαίνει κάποιον συντελεστή τότε ενεργοποιείται η τρίτη σημαία **meta.flag_subnet_flow_assymetry**. Υπάρχει ένας συντελεστής ανά υποδίκτυο και ανά πρωτόκολλο και αυτοί έχουν υπολογστοί όπως και στον προηγούμενο μηχανισμό με προγενέστερη ανάλυση της δικτυακής κίνησης και εξαρτώνται σε μεγάλο βαθμό από τα χαρακτηριστικά και την τοπολογία του δικτύου.

Ο μηχανισμός αυτός συμπληρώνει τους δύο προηγούμενους αντιμετωπίζοντας το πρόβλημα της ανίχνευσης ως επιθέσεων των ριπών κανονικής κίνησης προς συγκεκριμένο υποδίκτυο. Σε αυτές τις περιπτώσεις η κίνηση παρουσιάζει συμμετρία σε αντίθεση με κίνηση που οφείλεται σε κάποια επίθεση με αποτέλεσμα να μην ενεργοποιείται άσκοπα ο μηχανισμός.

```
meta.flag_subnet_flow_assymetry = 0;

bit<32> inner_register_shift = hdr.ipv4.protocol == ProtoType.TCP ? (bit<32>) 0 :
    MONITORING_CAPACITY * 2;

meta.src_stats_index = inner_register_shift + meta.src_subnet_code;
meta.dst_stats_index = inner_register_shift + MONITORING_CAPACITY + meta.dst_subnet_code;

bit<16> stats_packet_count;
bit<32> last_epoch;

if (meta.src_subnet_code != CODE_SUBNET_UNMONITORED) {
    r_assymetry_last_epoch.read(last_epoch, meta.src_stats_index);

    if (last_epoch != meta.epoch) {
        r_assymetry_last_epoch.write(meta.src_stats_index, meta.epoch);
        r_assymetry_packet_counter.write(meta.src_stats_index, 1);
    }
    else {
        r_assymetry_packet_counter.read(stats_packet_count, meta.src_stats_index);
        r_assymetry_packet_counter.write(meta.src_stats_index, stats_packet_count + 1);
    }
}


if (meta.dst_subnet_code != CODE_SUBNET_UNMONITORED) {
    r_assymetry_last_epoch.read(last_epoch, meta.dst_stats_index);

    if (last_epoch != meta.epoch) {
        r_assymetry_last_epoch.write(meta.dst_stats_index, meta.epoch);
        r_assymetry_packet_counter.write(meta.dst_stats_index, 1);
        stats_packet_count = 1;
    }
    else {
        r_assymetry_packet_counter.read(stats_packet_count, meta.dst_stats_index);
        r_assymetry_packet_counter.write(meta.dst_stats_index, stats_packet_count + 1);
    }

    bit<32> stats_src_packet_count;
    bit<32> stats_dst_packet_count;

    stats_dst_packet_count = (bit<32>) (stats_packet_count + 1);

    r_assymetry_packet_counter.read(stats_packet_count, meta.src_stats_index);
    stats_src_packet_count = (bit<32>) stats_packet_count;

    bit<32> asymmetry_factor;

    if (hdr.ipv4.protocol == ProtoType.TCP) {
        r_assymetry_factor_tcp.read(asymmetry_factor, meta.dst_subnet_code);
    }
    else {
```

```
52          r_assymetry_factor_udp.read(asymmetry_factor, meta.dst_subnet_code);
53      }
54
55      stats_dst_packet_count = stats_dst_packet_count * asymmetry_factor;
56
57      stats_src_packet_count = stats_src_packet_count + 1;
58      stats_dst_packet_count = stats_dst_packet_count + 1;
59
60      if (ASSYMETRY_SRC_FACTOR * stats_src_packet_count < ASSYMETRY_DST_FACTOR *
        stats_dst_packet_count) {
61          meta.flag_subnet_flow_assymetry = 1;
62      }
63  }
```

Κώδικας 4.8: Ασυμμετρία εισερχόμενων & εξερχόμενων πακέτων

4.5.4 Digests - Ειδοποιήσεις προς το control plane

Ένα πολύ σημαντικό χαρακτηριστικό της γλώσσας P4 είναι η δυνατότητα παραγωγής ειδοποιήσεων προς το επίπεδο ελέγχου. Αυτές ονομάζονται digests και μπορούν να συμπεριλαμβάνουν πληθώρα πληροφοριών ανάλογα με την εφαρμογή. Με τον τρόπο αυτό δεν απαιτείται το control plane να ζητά συνεχώς την ύπαρξη πληροφοριών από το data plane επιβαρρύνοντας το με περιττό φόρτο.

Ο μηχανισμός ανίχνευσης εκμεταλλεύται τα digests ώστε να ειδοποίησει το control plane σε περίπτωση επίθεσης ώστε το τελευταίο να μπορέσει άμεσα να κινήσει διαδικασίες αντιμετώπισης. Όταν ενεργοποιηθούν και οι τρεις σημαίες **meta.flag_total_flow_count**, **meta.flag_subnet_flow_count** και **meta.flag_subnet_flow_assymetry** ο μηχανισμός δημιουργεί μια ειδοποίηση προς το control plane η οποία περιέχει πληροφορία για την συγκεκριμένη IP διεύθυνση του παρακολουθούμενου δικτύου που δέχεται την επίθεση, τους κωδικούς αριθμούς υποδικτύου πηγής και προορισμού και τον αύξοντα αριθμό της τρέχουσας εποχής καθώς επίσης και το κλειδί του μηχανισμού HashPipe που αναλύεται στην ενότητα 4.6. Σε έναν επιπλέον καταχωρητή (**r_flags_epoch**) αποθηκεύεται ο αύξων αριθμός της εποχής κατά την οποία στάλθηκε τελευταία φορά digest. Αν κατά την τρέχουσα εποχή έχει ξανασταλεί τότε αποφεύγεται η επαναποστολή του προκειμένου να μην υπάρξει κατακλισμός του control plane απο ειδοποιήσεις, γεγονός πολύ πιθανό κατά την διάρκεια κάποιας επίθεσης στην οποία ο μηχανισμός θα ενεργοποιείται για κάθε πακέτο.

```
1  if (meta.dst_subnet_code != CODE_SUBNET_UNMONITORED) {
2      r_flags.write(0, meta.flag_total_flow_count);
3      r_flags.write(1, meta.flag_subnet_flow_count);
4      r_flags.write(2, meta.flag_subnet_flow_assymetry);
5
6      if (meta.flag_total_flow_count == 1 && meta.flag_subnet_flow_count == 1 && meta.
       flag_subnet_flow_assymetry == 1) {
7          digest(
8              (bit<32>) 1024,
9              {
10                 hdr.ipv4.dstAddr,
11                 meta.hashPipeKey,
12                 meta.epoch,
13                 meta.src_subnet_code,
14                 meta.dst_subnet_code
15             }
16         );
17     }
18 }
```

Κώδικας 4.9: Ειδοποιήσεις προς το control plane

4.6 Heavy-Hitters: HashPipe 4 σταδίων

Επιδιώκοντας την επέκταση της διαδικασίας ανίχνευσης και των πληροφοριών που είναι διαθέσιμες στο control plane, υλοποιήθηκε ο μηχανισμός **HashPipe** [39]. Η βασική λειτουργία του μηχανισμού αυτού είναι ο εντοπισμός των "top k heavy hitters" δηλαδή των k ροών που παρουσιάζουν πολύ μεγάλους όγκους κίνησης.

4.6.1 Ο αλγόριθμος

Ο αλγόριθμος του HashPipe βασίζεται στον Space Saving Algorithm, χρησιμοποιώντας περιορισμένο αριθμό καταχωρητών ανάλογο του πλήθους των επιθυμητών heavy hitters $O(k)$ και όχι του πλήθους των μοναδικών ροών που μπορεί να ξεπερνούν τις πολλές εκατοντάδες χιλιάδες. Κάθε καταχωρητής αποθηκεύει ένα ζεύγος κλειδιού και πλήθους εμφανίσεών του μέχρι στιγμής. Για κάθε πακέτο ο αλγόριθμος ελέγχει αν το αντίστοιχο κλεδί υπάρχει στον πίνακα καταχωρητών και αν ναι τότε αυξάνει το πλήθος εμφανίσεων κατά 1. Αν δεν υπάρχει αλλά υπάρχει κενός χώρος τότε εισάγεται στον πίνακα με πληθικότητα 1 αλλιώς αντικαθιστά το ζεύγος με την μικρότερη πληθικότητα, η οποία αυξάνεται κατά 1.

Εφόσον δεν είναι αποδοτική η αναζήτηση του ελαχίστου ολόκληρου του πίνακα και η P4 δεν υποστηρίζει επαναληπτικές δομές, ο αλγόριθμος αναζητεί το ελάχιστο μέσα σε ένα σύνολο d θέσεων οι οποίες προκύπτουν από d συναρτήσεις κατακερματισμού. Προκειμένου να γίνεται αποδοτικότερα η πρόσβαση στη μνήμη, ο αρχικός πίνακας χωρίζεται σε d υποπίνακες χωρίζοντας έτσι την υλοποίηση σε στάδια. Στο πρώτο στάδιο γίνεται πάντα εισαγωγή της νέας πλειάδας ενώ στα υπόλοιπα εφαρμόζεται ο αλγόριθμος όπως περιγράφεται ανωτέρω.

4.6.2 Υλοποίηση

Στην συγκεκριμένη υλοποίηση ως κλειδί χρησιμοποίηθηκε η πλειάδα **(destination ip, source port, destination port)**. Ο αριθμός των σταδίων επιλέχθηκε να είναι ίσος με τέσσερα καθώς περισσότερα στάδια θα αύξαναν πολύ το μέγεθος του εκτελέσιμου κώδικα χωρίς να προσφέρουν αντίστοιχα μεγάλη βελτίωση της ακρίβειας. Σε κάθε στάδιο τα κλειδιά και οι τιμές αποθηκεύονται σε δύο ξεχωριστούς πίνακες καταχωρητών τους **r_stage_{i}_keys** και **r_stage_{i}_values** αντίστοιχα. Ένας επιπλέον βοηθητικός πίνακας **r_stage_{i}_epoch** αποθηκεύει τον αύξοντα αριθμό της εποχής που προσπαλάστηκε τελευταία φορά η αντίστοιχη πλειάδα ώστε να ορίσει την εγκυρότητα ή μή των δεδομένων της ανάλογα με την τρέχουσα εποχή. Ο πίνακας **r_stage_{i}_coeff** κάθε σταδίου αποθηκεύει τις σταθερές a και b οι οποίες χρησιμοποιούνται για την υλοποίηση των συναρτήσεων κατακερματισμού που είναι της μορφής $h_i(x) = (a_i * x + b_i)\ mod\ p_i$ όπου p_i το μέγεθος του πίνακα κάθε σταδίου. Για την αποφυγή overflows από τον πολλαπλασιασμό μεγάλων αριθμών, οι συναρτήσεις κατακερματισμού μετατράπηκαν με χρήση των ιδιοτήτων της αριθμητικής modulo και των δυαδικών αριθμών σε πράξεις πολλαπλασιασμού, πρόσθεσης και λογικής σύζευξης με μικρούς αριθμούς.

```
action calculate_hash_index(in bit<32> lower_key, in bit<32> a, in bit<32> b, out bit<32>
    index) {
    bit<32> _MOD = HASHPIPE_STAGE_CAPACITY - 1;
    index = (((a * (lower_key & _MOD)) & _MOD) + b) & _MOD;
}

control HashPipe_Stage_1(inout metadata meta) {
    register<hashPipeKey_t>(HASHPIPE_STAGE_CAPACITY)    r_stage_1_keys;
    register<hashPipeValue_t>(HASHPIPE_STAGE_CAPACITY)  r_stage_1_values;
    register<bit<32>>(HASHPIPE_STAGE_CAPACITY)          r_stage_1_epoch;
    register<bit<32>>(2)                                r_stage_1_coeff;

    apply {
        bit<32>         hashTableIndex;
```

```
        hashPipeKey_t   prev_key;
        hashPipeValue_t prev_value;
        bit<32>         prev_epoch;
        hashPipeValue_t value_to_save;
        bool            willCarryKey;
        bit<32>         a;
        bit<32>         b;

        r_stage_1_coeff.read(a, 0);
        r_stage_1_coeff.read(b, 1);
        calculate_hash_index(meta.hashPipeKey, a, b, hashTableIndex);

        r_stage_1_keys.read(prev_key, hashTableIndex);
        r_stage_1_values.read(prev_value, hashTableIndex);
        r_stage_1_epoch.read(prev_epoch, hashTableIndex);

        willCarryKey = (prev_epoch == meta.epoch) && (prev_key != meta.hashPipeKey);
        value_to_save = (prev_epoch == meta.epoch) && (prev_key == meta.hashPipeKey) ?
    prev_value + 1 : 1;

        r_stage_1_keys.write(hashTableIndex, meta.hashPipeKey);
        r_stage_1_values.write(hashTableIndex, value_to_save);
        r_stage_1_epoch.write(hashTableIndex, meta.epoch);

        meta.hashPipeKey = willCarryKey ? prev_key : 0;
        meta.hashPipeValue = willCarryKey ? prev_value : 0;
    }
}
```

Κώδικας 4.10: HashPipe στάδιο 1

```
control HashPipe_Stage_i(inout metadata meta) {
    register<hashPipeKey_t>(HASHPIPE_STAGE_CAPACITY)    r_stage_i_keys;
    register<hashPipeValue_t>(HASHPIPE_STAGE_CAPACITY)  r_stage_i_values;
    register<bit<32>>(HASHPIPE_STAGE_CAPACITY)          r_stage_i_epoch;
    register<bit<32>>(2)                                r_stage_i_coeff;

    apply {
        bit<32>         hashTableIndex;
        hashPipeKey_t   prev_key;
        hashPipeValue_t prev_value;
        bit<32>         prev_epoch;
        hashPipeKey_t   key_to_save;
        hashPipeValue_t value_to_save;
        bool            willCarryKey;
        bit<32>         a;
        bit<32>         b;

        r_stage_i_coeff.read(a, 0);
        r_stage_i_coeff.read(b, 1);
        calculate_hash_index(meta.hashPipeKey, a, b, hashTableIndex);

        r_stage_i_keys.read(prev_key, hashTableIndex);
        r_stage_i_values.read(prev_value, hashTableIndex);
        r_stage_i_epoch.read(prev_epoch, hashTableIndex);

        if (prev_epoch != meta.epoch || prev_key == meta.hashPipeKey) {
            key_to_save = meta.hashPipeKey;
            value_to_save = (prev_epoch != meta.epoch) ? meta.hashPipeValue : prev_value +
    meta.hashPipeValue;
            meta.hashPipeKey = 0;
            meta.hashPipeValue = 0;
        }
        else {
            key_to_save         = (prev_value < meta.hashPipeValue) ? meta.hashPipeKey :
    prev_key;
            value_to_save       = (prev_value < meta.hashPipeValue) ? meta.hashPipeValue :
    prev_value;
```

```
35              meta.hashPipeKey    = (prev_value < meta.hashPipeValue) ? prev_key : meta.
    hashPipeKey;
36              meta.hashPipeValue  = (prev_value < meta.hashPipeValue) ? prev_value : meta.
    hashPipeValue;
37          }
38
39          r_stage_i_keys.write(hashTableIndex, key_to_save);
40          r_stage_i_values.write(hashTableIndex, value_to_save);
41          r_stage_i_epoch.write(hashTableIndex, meta.epoch);
42      }
43  }
```

Κώδικας 4.11: HashPipe στάδια 2, 3, 4

4.7 Συνεργασία μηχανισμών και Δειγματοληψία

Ο μηχανισμός ανίχνευσης και το hashpipe λειτουργούν συνεργατικά. Όταν η διαδικασία ανίχνευσης παράξει κάποια ειδοποίηση για πιθανή επίθεση προς το control plane, τα δεδομένα του hashpipe μπορούν να ανακληθούν και να χρησιμοποιηθούν συμπληρωματικά για την εξαγωγή συμπερασμάτων σχετικά με το αν υπάρχει όντως κάποια επίθεση, το σχετικό της μέγεθος, τον αποδέκτη της καθώς και τον τύπο της ο οποίος μπορεί να προσδιοριστεί από την πόρτα προορισμού των πακέτων [42] [43]. Αυτός είναι και ο λόγος που επιλέχθηκε ως κλειδί η πλειάδα **(destination ip, source port, destination port)**, ώστε να είναι δυνατή η άμεση εξαγωγή πληροφοριών σχετικά με τον αποδέκτη και τον τύπο της επίθεσης.

Ωστόσο η επιβάρυνση στο μέγεθος της επεξεργασίας που υφίστανται τα πακέτα με την προσθήκη του hashpipe είναι αρκετά μεγάλη αυξάνοντας τον κίνδυνο μείωσης της επίδοσης του δικτύου. Προκειμένου να αποφευχθεί η εκτεταμένη καθυστέρηση, τα πακέτα δειγματοληπτούνται με ρυθμό $1/n$ και στη συνέχεια προχωρούν στην επεξεργασία από τους δύο μηχανισμούς. Οι ρυθμοί δειγματοληψίας, ξεχωριστά για κάθε μηχανισμό, αποθηκεύονται στον καταχωρητή **r_sample_interval**. Ένας μετρητής, επίσης ξεχωριστός για κάθε μηχανισμό που αποθηκεύεται στον καταχωρητή **r_sample_count**, καταμετρά τα εισερχόμενα πακέτα και προωθεί στον μηχανισμό μόνο όταν η τιμή του μετρητή εξισωθεί με τον ρυθμό δειγματοληψίας. Έπειτα ο μετρητής μηδενίζεται και η διαδικασία επαναλαμβάνεται.

```
1  control IngressLogic(inout headers hdr, inout metadata meta, inout standard_metadata_t
       standard_metadata) {
2
3      BasicDeviceRole()       ctrl_device_role;
4      PacketTagging()         ctrl_packet_tagging;
5      DetectionMechanism()    ctrl_ddos_protection;
6      HashPipe()              ctrl_hash_pipe;
7
8      register<bit<16>>(2) r_sample_interval;
9      register<bit<16>>(2) r_sample_count;
10
11     apply {
12         ctrl_device_role.apply(standard_metadata);
13         if (hdr.ports.isValid()) {
14             bit<16> interval;
15             bit<16> count_detection;
16             bit<16> count_hashpipe;
17             bool epoch_reset;
18             bool willApplyDetection;
19             bool willApplyHashPipe;
20
21             // Separate sampling intervals
22             // Check detection
23             r_sample_interval.read(interval, 0);
24             r_sample_count.read(count_detection, 0);
25             count_detection = count_detection + 1;
26             willApplyDetection = (count_detection == interval);
```

```
27
28              // Check hashpipe
29              r_sample_interval.read(interval, 1);
30              r_sample_count.read(count_hashpipe, 1);
31              count_hashpipe = count_hashpipe + 1;
32              willApplyHashPipe = (count_hashpipe == interval);
33
34              if (willApplyDetection) {
35                  // STEP 1: Tag packet with epoch info. New epoch is allowed. May initiate
    epoch resetting in detection pipeline.
36                  ctrl_packet_tagging.apply(hdr, meta, standard_metadata, true, epoch_reset);
37                  // STEP 2: Run detection pipeline
38                  ctrl_ddos_protection.apply(hdr, meta, standard_metadata, epoch_reset);
39                  // STEP 3: Reset counter
40                  count_detection = 0;
41              }
42
43              if (willApplyHashPipe) {
44                  // STEP 1: Tag packet with epoch info. New epoch is NOT allowed. epoch_reset
    will be false.
45                  // Not needed if detection has already run. Avoid unnecessary control calls.
46                  if (!willApplyDetection) {
47                      ctrl_packet_tagging.apply(hdr, meta, standard_metadata, false, epoch_reset
    );
48                  }
49                  // STEP 2: Run hashpipe
50                  ctrl_hash_pipe.apply(hdr, meta);
51                  // STEP 3: Reset counter
52                  count_hashpipe = 0;
53              }
54
55              r_sample_count.write(0, count_detection);
56              r_sample_count.write(1, count_hashpipe);
57
58          }
59      }
60
61  }
```

Κώδικας 4.12: Δειγματοληψία

Κεφάλαιο 5

Αποτελέσματα και Πειραματική Αξιολόγηση

5.1 Εργαλεία ανάπτυξης και διατάξεις ελέγχου

5.1.1 P4 Behavioral Model (BMv2)

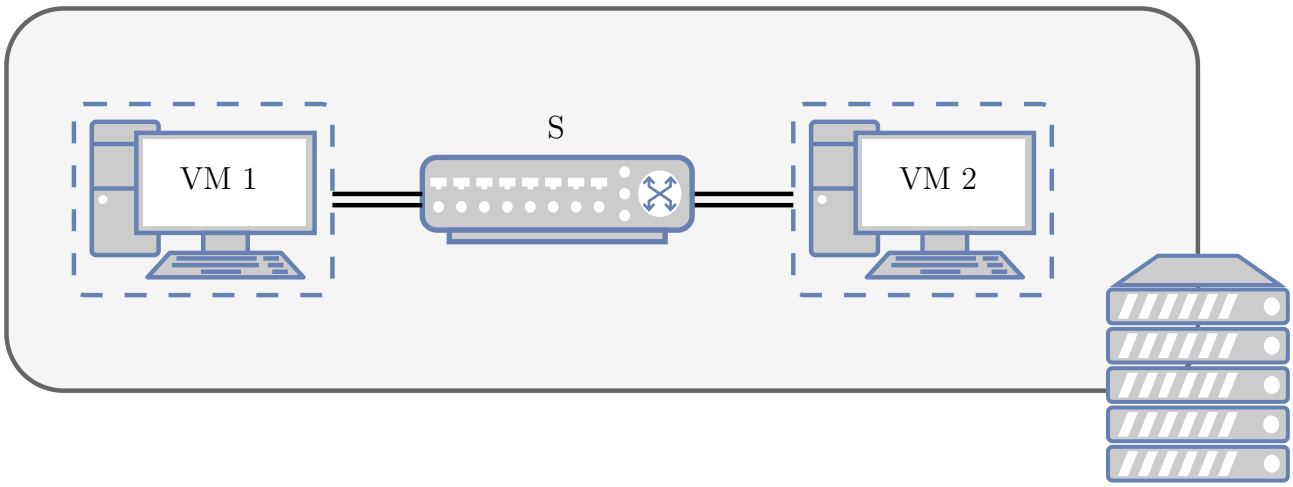
Το BMv2 Software Switch [44] αποτελεί ένα περιβάλλον ανάπτυξης και ελέγχου για προγράμματα γραμμένα σε P4. Το ίδιο έχει υλοποιηθεί με την C++11 και παρέχει τη δυνατότητα ορισμού εικονικών switches συγκεκριμένης αρχιτεκτονικής από τον προγραμματιστή. Το περιβάλλον ανάπτυξης παρέχει ήδη υλοποιημένες κάποιες παραλλαγές του βασικού μοντέλου αρχιτεκτονικής της P4 (εικόνα 2.4).

Η μορφή του κώδικα βασίστηκε στα πρότυπα των επίσημων παραδειγμάτων της P4 [45]. Για την ανάπτυξη και τον έλεγχο της υλοποίησης χρησιμοποιήθηκε το **simple_switch** το οποίο δέχεται ως είσοδο τον ορισμό ενός P4 προγράμματος σε μορφή JSON και προσομοιώνει τη λειτουργία του. Για την παραγωγή αυτής της περιγραφής σε JSON χρησιμοποιήθηκε ο επίσημος μεταγλωττιστής **p4c** [46]. Για την παραμετροποίηση και επικοινωνία με το εικονικό switch παρέχεται επίσης το εργαλείο **simple_switch_CLI** [47] το οποίο επιτρέπει την ανάγνωση και τροποποίηση των τιμών καταχωρητών, μετρητών και εγγραφών πινάκων.

Για τον έλεγχο εγκυρότητας του προγράμματος δημιουργήθηκε μια διάταξη τριών εικονικών μηχανημάτων πάνω σε ένα φυσικό μηχάνημα όπως φαίνεται στην εικόνα 5.1. Η κίνηση παράγεται στο εικονικό μηχάνημα 1, στη συνέχεια διέρχεται από το εικονικό μηχάνημα `S` που χρησιμοποιεί το simple_switch για την προσομοίωση του προγράμματος και καταλήγει στο εικονικό μηχάνημα 2. Επειδή το simple_switch_CLI δεν δίνει τη δυνατότητα ανάγνωσης digests, δημιουργήθηκε με χρήση της Python ένα πρόγραμμα (κώδικας A.1) που διαβάζει τα digests από ένα UNIX socket στο οποίο τα τοποθετεί το simple_switch και στη συνέχεια τα αποθηκεύει σε ένα αρχείο σε ευανάγνωστη μορφή για περαιτέρω ανάλυση. Επιπλεόν, για την αποσφαλμάτωση του κώδικα χρησιμοποιήθηκε και ο debugger **p4dbg** [48] που παρέχεται στο περιβάλλον του BMv2. Πρέπει να σημειωθεί ότι το BMv2 δεν ενδείκνυται για την πραγματοποίηση πειραμάτων και εξαγωγή αποτελεσμάτων καθώς δεν είναι σχεδιασμένο με γνώμονα τις επιδόσεις αλλά την διευκόλυνση της ανάπτυξης προγραμμάτων.

5.1.2 Netronome SmartNIC Tools

Όλα τα πειράματα για την αξιολόγηση του μηχανισμού πραγματοποιήθηκαν σε πραγματικό εξοπλισμό και συγκεκριμένα σε μία κάρτα Netronome SmartNIC. Οι Netronome SmartNICs [35] είναι προγραμματιζόμενες δικτυακές κάρτες υψηλών επιδόσεων που υποστηρίζουν μεγάλο εύρος εφαρμογών, από απλές λειτουργίες δικτύωσης μέχρι σύνθετες διαδικασίες διαχείρισης και παρακολούθησης της



Σχήμα 5.1: Διάταξη ελέγχου του μηχανισμού

δικτυακής κίνησης. Βασίζονται στους επεξεργαστές ειδικού σκοπού Netronome Flow Processors οι οποίοι επιτυγχάνουν πολύ υψηλές ταχύτητες επεξεργασίας με χαμηλή ενεργειακή κατανάλωση και επιτρέπουν την ταχεία προσαρμογή σε νέες δικτυακές λειτουργίες και δυνατότητες.

Για τον προγραμματισμό της κάρτας χρησιμοποιήθηκε η σουίτα εργαλείων NFP Software Development Kit (NFP-SDK). Το περιβάλλον ανάπτυξης **Programmer Studio** δίνει την δυνατότητα συγγραφής προγραμμάτων σε P4 ή σε C. Ενσωματώνει όλα τα απαραίτητα στάδια (preprocessing, compiling, assembling, linking) για την μετατροπή των προγραμμάτων σε εκτελέσιμα αρχεία κατάλληλης μορφής (**nffw**) τα οποία μπορούν στη συνέχεια να εκτελεστούν στις κάρτες. Για την αποστολή του εκτελέσιμου αρχείου στην κάρτα αλλά και για την επικοινωνία με αυτή, χρησιμοποιούνται τα προγράμματα **pif_rte** και **rtecli** αντίστοιχα. Το pif_rte αναλαμβάνει την αποστολή του εκτελέσιμου στην κάρτα και την αρχική παραμετροποίηση των δομών που χρησιμοποιούνται από αυτό και λειτουργεί σαν server που αποδέχεται αιτήματα που ανακαλούν ή τροποποιούν δεδομένα των δομών του προγράμματος όσο αυτό εκτελείται στην κάρτα. Το rtecli αποτελεί ενδιάμεσο πρόγραμμα διαχείρισης που με τις κατάλληλες εντολές δημιουργεί και αποστέλει τέτοια αιτήματα προς το pif_rte και παρουσιάζει σε ευανάγνωστη μορφή τα αποτελέσματα.

5.2 Προσομοίωση δικτυακής κίνησης

Για την προσομοίωση της δικτυακής κίνησης που θα διέρχεται από τον μηχανισμό χρησιμοποιήθηκαν καταγραφές από το ιαπωνικό project WIDE [36]. Επιπλέον, για την προσομοίωση των επιθέσεων χρησιμοποιήθηκαν καταγραφές από τη μελέτη των "Booters" [37].

Η αναπαραγωγή των καταγραφών έγινε με χρήση του εργαλείου **tcpreplay** [49]. Τα αρχεία καταγραφών pcap αρχικά επεξεργάζονται ώστε οι MAC διευθύνσεις των πακέτων να αντιστοιχούν στις διεθύνσεις MAC των πραγματικών δικτυακών διεπαφών της διάταξης και να αποφευχθεί η απόρριψή τους. Η επεξεργασία αυτή έγινε με το εργαλείο tcprewrite αλλά και με ένα πρόγραμμα σε Python που δημιουργήθηκε ειδικά για αυτόν τον σκοπό (κώδικας A.2). Στη συνέχεια τα τροποιημένα αρχεία αναπαράγονται με το tcpreplay με διάφορους ρυθμούς αναπαραγωγής (σε πακέτα ανά δευτερόλεπτο).

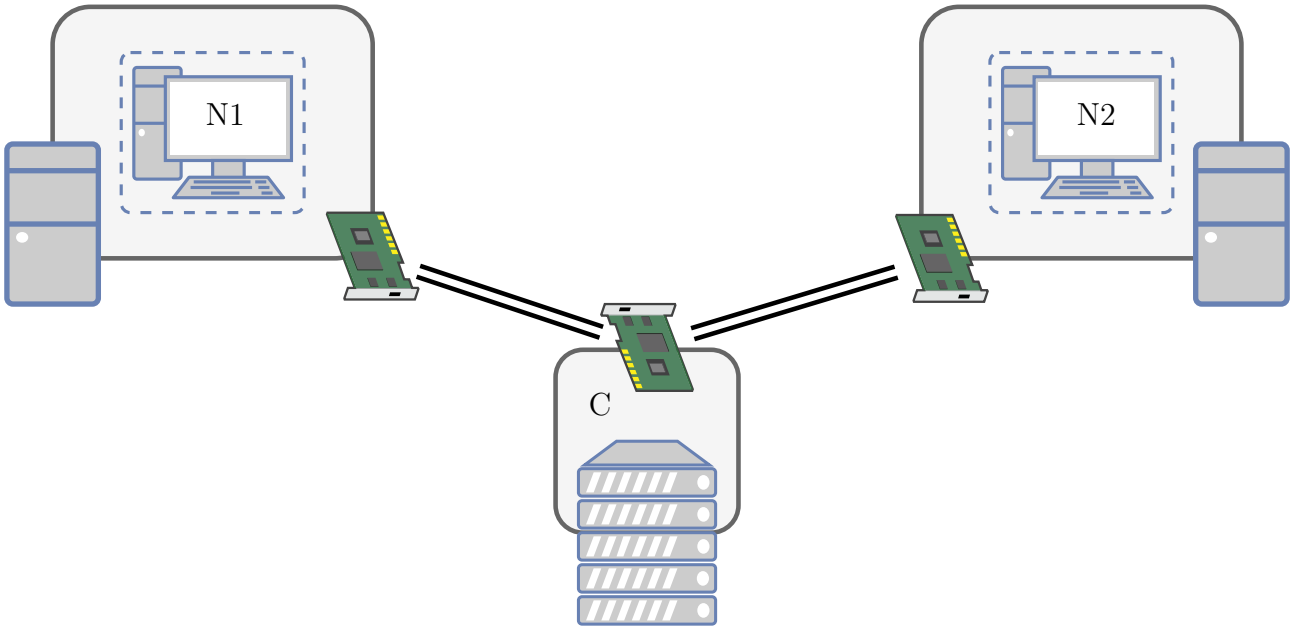
Κατά τη φάση ελέγχου του μηχανισμού χρησιμοποιήθηκε επίσης το εργαλείο - βιβλιοθήκη **scapy** [50] για την δημιουργία και αποστολή ειδικά κατασκευασμένων πακέτων με πολύ χαμηλούς ρυθμούς αποστολής.

Κατά τη φάση του ελέγχου της ρυθμαπόδοσης χρησιμοποιήθηκε το εργαλείο **pfsend** [51] που επιτρέπει πολύ υψηλούς ρυθμούς αναπαραγωγής της κίνησης προκειμένου να προσομοιωθεί ο όγκος κίνησης μίας επίθεσης DDoS. Για την καταμέτρηση των εξερχόμενων από τον μηχανισμό πακέτων ανά δευτερόλεπτο

χρησιμοποιήθηκε το πρόγραμμα **watch_ethtool** (κώδικας Α.3) το οποίο δημιουργήθηκε για τους σκοπούς της συγκεκριμένης μέτρησης και βασίστηκε στο εργαλείο `mmwatch` [52].

5.3 Πειραματικές διατάξεις - μεθοδολογία μετρήσεων

Για την εκτέλεση των πειραμάτων χρησιμοποιήθηκε διάταξη παρόμοια με την διάταξη ελέγχου. Δύο φυσικά μηχανήματα N_1 και N_2 αποτελούν την πηγή και τον προορισμό της αναπαραγόμενης κίνησης. Ένα τρίτο φυσικό μηχάνημα C που διαθέτει μία κάρτα Netronome SmartNIC διασυνδέει τα άλλα δύο στην τοπολογία που φαίνεται στο σχήμα 5.2.



Σχήμα 5.2: Διάταξη πειραμάτων

Μετά από εξέταση των αρχείων καταγραφής από το WIDE επιλέχθηκαν τα 255 /24 υποδίκτυα με τη μεγαλύτερη κίνηση τα οποία θα αποτελέσουν τα παρακολουθούμενα δίκτυα του μηχανισμού. Με βάση αυτά προϋπολογίζονται τα ποσοστά κίνησης ανά υποδίκτυο και οι παράγοντες ασυμμετρίας που θα αποτελέσουν την αρχική παραμετροποίηση του μηχανισμού. Για την αρχική παραμετροποίηση των δομών στην κάρτα απαιτείται μαζί με τον εκτελέσιμο κώδικα και ένα αρχείο επέκτασης **p4cfg** με δομή JSON που περιέχει τις αρχικές τιμές των registers και τις εγγραφές και τα δεδομένα των match-action tables. Αυτό μπορεί να δημιουργηθεί με τη βοήθεια του Programmer Studio και σύμφωνα με τις τιμές που υπολογίστηκαν από την ανάλυση των αρχείων καταγραφής.

Για την πραγματοποίηση κάθε σεναρίου μετρήσεων χρησιμοποιήθηκαν τα εξής προγράμματα:

1. Στον κόμβο N_1 το πρόγραμμα **run_traffic** (κώδικας Α.5) που αναπαράγει την βασική κίνηση με συγκεκριμένο ρυθμό και για συγκεκριμένο χρονικό διάστημα και εμφυτεύει σε αυτήν την κίνηση της επίθεσης σε συγκεκριμένες χρονικές στιγμές και για επιλεγμένη χρονική διάρκεια και ρυθμό.

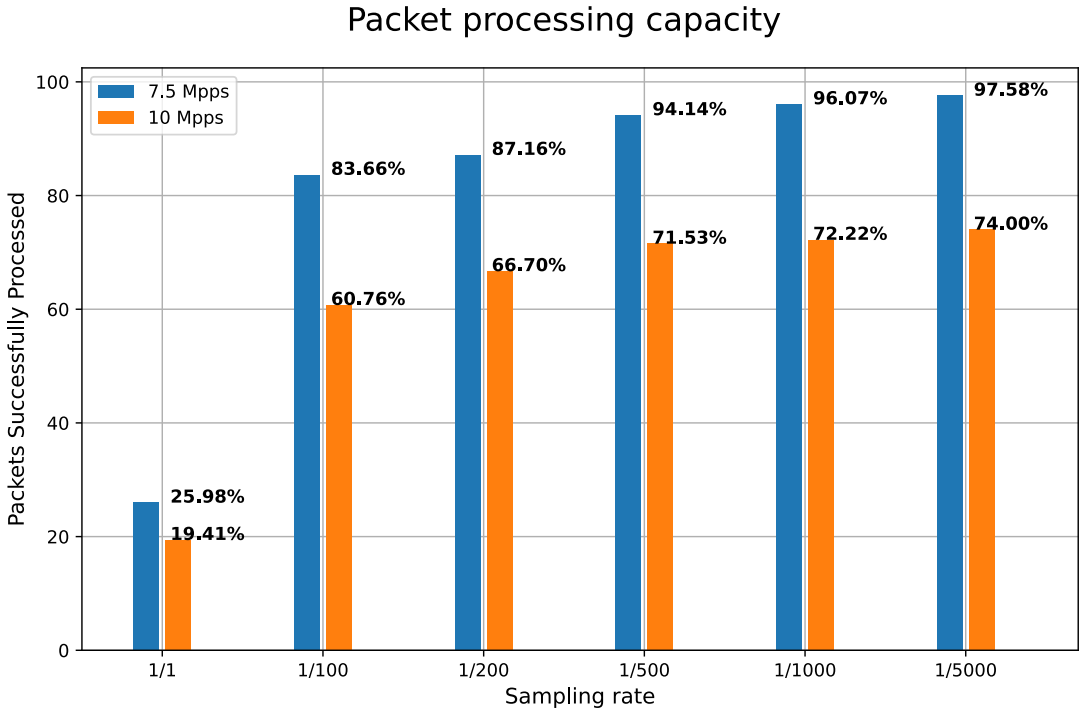
2. Στον κόμβο C το πρόγραμμα **run_testbed** (κώδικας Α.4) που με χρήση του rtecli αρχικοποιεί τις δομές του προγράμματος, καταγράφει τα digests καθ'όλη τη διάρκεια του πειράματος και αποθηκεύει τις καταστάσεις των δομών του hashpipe ανά δευτερόλεπτο.

3. Στον κόμβο N_2 το πρόγραμμα **watch_ethtool** (κώδικας Α.3) που καταγράφει το ρυθμό άφιξης πακέτων ώστε να υπάρχει εποπτεία για το αν υπήρξε απώλεια πακέτων κατά τη διάρκεια των πειραμάτων.

5.4 Επίδραση δειγματοληψίας στην ρυθμαπόδοση

Στις σύγχρονες δικτυακές υποδομές, η ταχύτητα με την οποία φτάνουν πακέτα από το δίκτυο στους μεταγωγείς είναι πολύ μεγάλη, ειδικά στις περιπτώσεις που οι δικτυακές συνδέσεις επιτρέπουν διέλευση κίνησης πολλών Gbps. Για το λόγο αυτό, η επεξεργασία των πακέτων στις δικτυακές συσκευές πρέπει να πραγματοποιείται όσο το δυνατόν ταχύτερα προκειμένου να μην υπάρξει συμφόρηση και τελικά απώλεια πακέτων. Οποιαδήποτε επιπλέον λειτουργία πέραν της απλής μεταγωγής αυξάνει αρκετά τον χρόνο επεξεργασίας κάθε πακέτου. Ο μηχανισμός που υλοποιήθηκε περιλαμβάνει αρκετές επιπλέον λειτουργίες οι οποίες, όταν πραγματοποιούνται για κάθε πακέτο, ενδέχεται να περιορίσουν πολύ την ταχύτητα διέλευσης τους από την κάρτα.

Στα πειράματα που ακολουθούν εφαρμόστηκε δειγματοληψία στα πακέτα που διέρχονται από τον μηχανισμό με σκοπό την μελέτη της επίδρασής της στον ρυθμό επεξεργασίας των πακέτων. Η περίπτωση $s = 1$ αποτελεί την βάση των μετρήσεων και πρόκειται για το σενάριο όπου όλα τα πακέτα επιλέγονται για να επεξεργαστούν από τον μηχανισμό. Η αναπαραγωγή της κίνησης πραγματοποιείται από τον κόμβο πηγής ($N_1 \rightarrow N_2$) με όσο το δυνατόν μεγαλύτερο ρυθμό. Για την επίτευξή της χρησιμοποιήθηκε το εργαλείο `pfsend` [51] με ρυθμούς αποστολής **7.5 Mpps** και **10 Mpps**.



Σχήμα 5.3: Επίδοση επεξεργασίας πακέτων από τον μηχανισμό

Στο διάγραμμα 5.3 φαίνεται το ποσοστό των συνολικών πακέτων που ο μηχανισμός επεξεργάστηκε με επιτυχία. Για τόσο υψηλό όγκο κίνησης, η πρόσθετη επεξεργασία που πραγματοποιείται οδηγεί σε πολύ μειωμένη δυνατότητα διαχείρισης των πακέτων με μόλις ένα στα τέσσερα πακέτα να επεξεργάζεται πλήρως στην περίπτωση που δεν χρησιμοποιείται δειγματοληψία. Η καθυστέρηση αυτή οδηγεί στην υπερβολική αύξηση του μεγέθους των ουρών εισόδου πακέτων με αποτέλεσμα να εμφανίζονται μεγάλες απώλειες. Ωστόσο ακόμη και με χρήση πυκνής δειγματοληψίας παρατηρείται πολύ μεγάλη βελτίωση στη δυνατότητα διαχείρισης πακέτων. Όσα πακέτα δεν επιλέγονται να επεξεργασθούν από τον μηχανισμό δεν συμβάλουν στην αύξηση μεγέθους της ουράς αναμονής και έτσι περιορίζονται οι απώλειες. Η περαιτέρω μείωση του ρυθμού δειγματοληψίας δεν προκαλεί εξίσου ραγδαία αύξηση της απόδοσης του μηχανισμού.

5.5 Αποτελεσματικότητα μηχανισμού ανίχνευσης

Για την αξιολόγηση της εγκυρότητας του μηχανισμού ανίχνευσης πραγματοποιήθηκε προσομοίωση κανονικής και κακόβουλης κίνησης. Πραγματοποιήθηκαν πειράματα διάρκειας 600 δευτερολέπτων στα οποία αναπαράχθηκε με πραγματικό ρυθμό η κανονική κίνηση. Ταυτόχρονα, στις χρονικές στιγμες $t_1 = 280\ sec$ και $t_2 = 450\ sec$ παρεμβλήθηκε επίσης με κανονικό ρυθμό κακόβουλη κίνηση διάρκειας $d = 24\ sec$. Ως αποδέκτες της επίθεσης ορίστηκαν τυχαίες διευθύνσεις από 5 τυχαία επιλεγμένα παρακολουθούμενα υποδίκτυα που παρουσιάζουν μεγάλα ποσοστά αναμενόμενης κανονικής κίνησης. Η διάρκεια των εποχών ορίστηκε σε $T = 2\ sec$.

Στο τέλος κάθε εποχής, ο μηχανισμός δημιουργεί ένα digest message με δεδομένα τον αύξοντα αριθμό της εποχής και το πλήθος ειδοποιήσεων για πιθανή επίθεση που δημιουργήθηκαν. Τα μηνύματα αυτά συλλέχθηκαν, και με γνώση της χρονικής στιγμής και διάρκειας των επιθέσεων αξιολογήθηκε η ικανότητα του μηχανισμού να παράγει μηνύματα αναγνώρισης της επίθεσης.

Ως μετρικές αξιολόγησης χρησιμοποιήθηκαν τα **True Positive Rate** και **False Positive Rate**. Για να ορισθούν πρέπει να χαρακτηρίσουμε τις πιθανές καταστάσεις των αποτελεσμάτων:

- **True Positive (TP)**: Εποχές με ενεργή επίθεση που ο μηχανισμός δημιούργησε ειδοποιήσεις
- **True Negative (TN)**: Εποχές χωρίς επίθεση που ο μηχανισμός δεν δημιούργησε ειδοποιήσεις
- **False Positive (FP)**: Εποχές χωρίς επίθεση που όμως ο μηχανισμός δημιούργησε ειδοποιήσεις
- **False Negative (FN)**: Εποχές με επίθεση που ο μηχανισμός δεν δημιούργησε ειδοποιήσεις

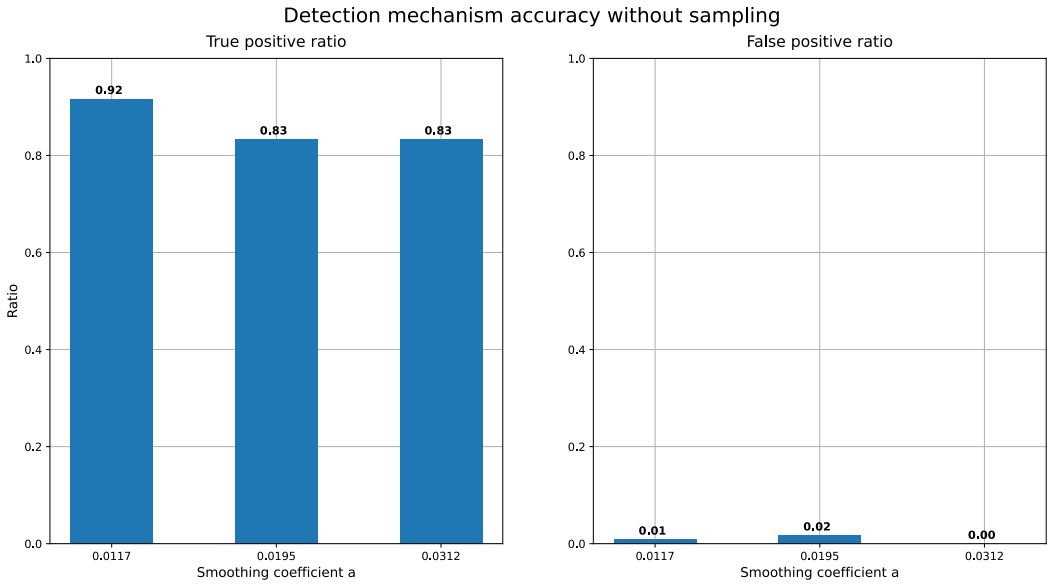
Οι δύο μετρικές μπορούν με βάση τα παραπάνω να ορισθούν ως:

- Το ποσοστό των εποχών κατά τις οποίες εξελισσόταν η επίθεση και ο μηχανισμός δημιούργησε σωστές ειδοποίησεις
$$TPR = \frac{TP}{TP + FN} \tag{5.1}$$
- Το ποσοστό των εποχών κατά τις οποίες δεν εξελισσόταν επίθεση αλλά ο μηχανισμός δημιούργησε λανθασμένες ειδοποιήσεις
$$FPR = \frac{FP}{FP + TN} \tag{5.2}$$

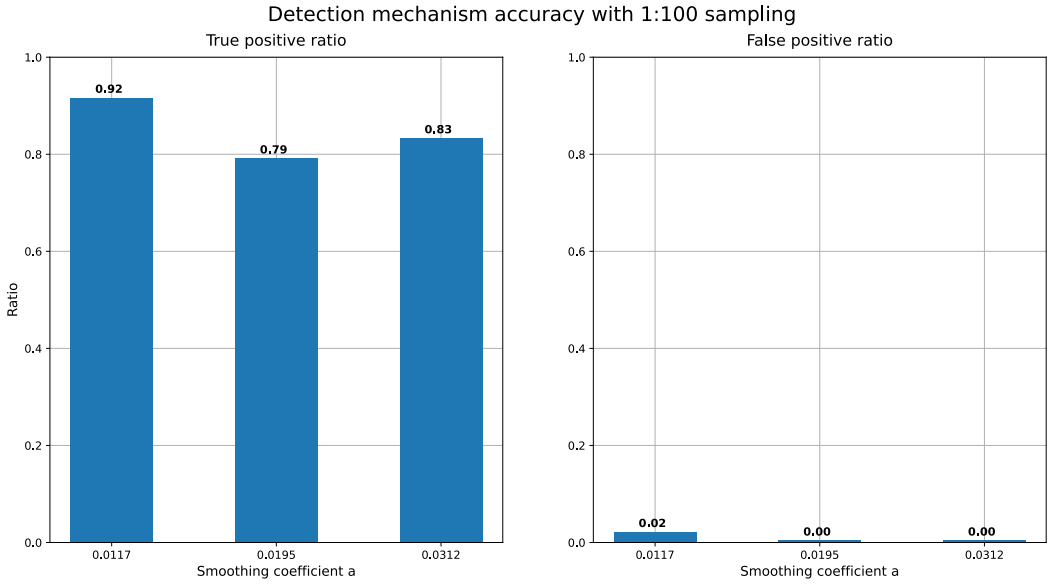
Η βασική περίπτωση, στην οποία δεν εφαρμόζεται δειγματοληψία, επιτυγχάνει πολύ υψηλά ποσοστά ανίχνευσης των επιθέσεων καθ' όλη τη διάρκειά τους εμφανίζοντας ταυτόχρονα πολύ μικρό αριθμό λανθασμένων ειδοποιήσεων. Στο διάγραμμα 5.4 παρουσιάζονται οι δείκτες `TPR` και `FPR` για τρεις διαφορετικές τιμές του συντελεστή εξομάλυνσης a. Ο συντελεστής αυτός χρησιμοποιείται στην εκθετική εξομάλυνση για τον υπολογισμό των κατωφλιών για τις δύο πρώτες μετρικές του μηχανισμού ανίχνευσης όπως αυτές περιγράφονται στην ενότητα 4.5. Σε όλα τα πειράματα οι μετρήσεις λήφθηκαν μετά τις πρώτες 30 εποχές ώστε ο μηχανισμός να προσαρμοστεί στα χαρακτηριστικά της κανονικής κίνησης.

Σε όλες τις περιπτώσεις ο μηχανισμός έχει επιτυχώς αναγνωρίσει την επίθεση. Οι αποκλίσεις στα ποσοστά οφείλονται στο γεγονός ότι ο μηχανισμός μετά από ορισμένο χρονικό διάστημα προσαρμόζεται στην κίνηση της επίθεσης. Όσο μεγαλύτερος ο συντελεστής εξομάλυνσης a τόσο ταχύτερη η προσαρμογή του μηχανισμού. Η εκρηκτικότητα που παρουσιάζει το πλήθος των μοναδικών ροών κατά τη διάρκεια των επιθέσεων επηρεάζει και αυτή την ταχύτητα προσαρμογής του μηχανισμού. Ωστόσο σε όλες τις περιπτώσεις, ο μηχανισμός δεν παρήγαγε παρά ελάχιστες λανθασμένες ειδοποιήσεις σε εποχές που δεν υπήρχε σε εξέλιξη κάποια επίθεση.

Με την εφαρμογή της δειγματοληψίας αρχικά με αυξημένο ρυθμό ($f_s = \frac{1}{100}$) παρατηρείται ελάχιστη μείωση της ακρίβειας ανίχνευσης και μόνο για μία περίπτωση του συντελεστή εξομάλυνσης a, όπως



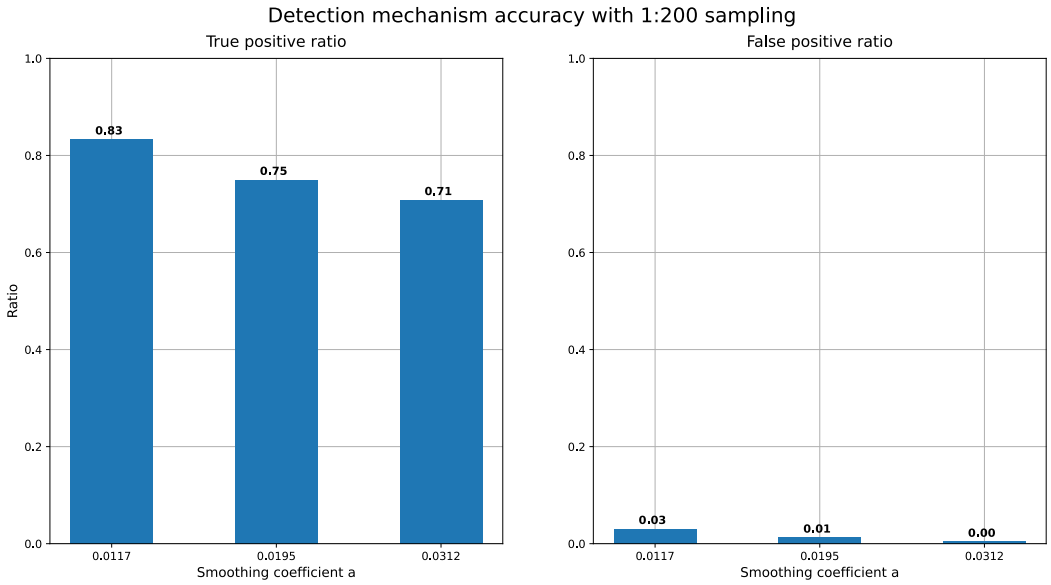
Σχήμα 5.4: Απόδοση μηχανισμού ανίχνευσης χωρίς δειγματοληψία



Σχήμα 5.5: Απόδοση μηχανισμού ανίχνευσης με ρυθμό δειγματοληψίας 1/100

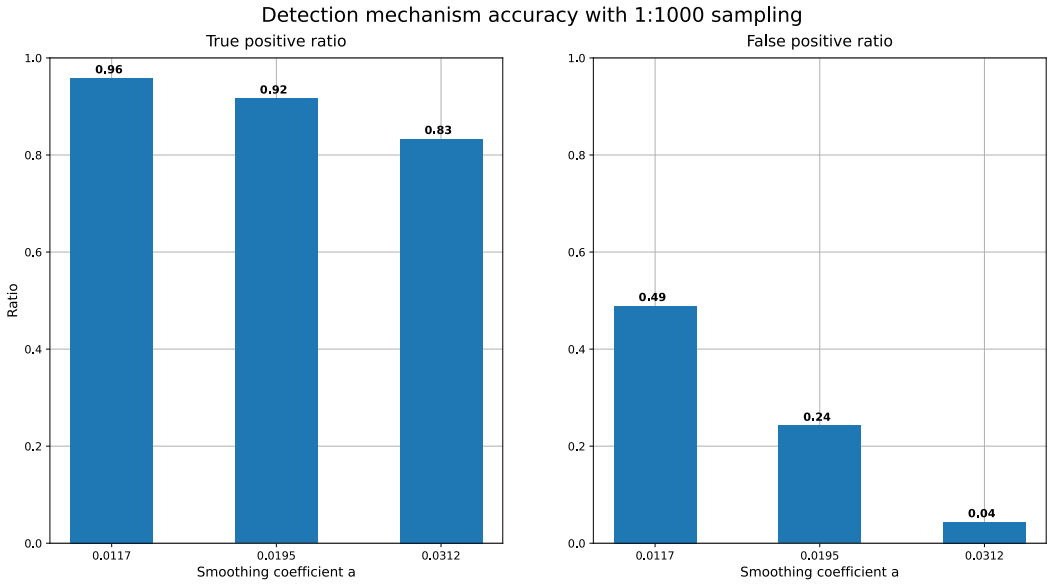
παρουσιάζεται στο διάγραμμα 5.5. Οι αποκλίσεις από την πλήρη ανίχνευση της επίθεσης οφείλονται και πάλι στο γεγονός ότι ο μηχανισμός μετά από ορισμένο χρονικό διάστημα οδηγείται σε προσαρμογή, με αποτέλεσμα οι ειδοποιήσεις να ελαττώνονται προς το τέλος της επίθεσης. Πρέπει να τονιστεί ότι με την εφαρμογή της δειγματοληψίας, μειώνεται η διαθέσιμη πληροφορία με την οποία ο μηχανισμός αποφαίνεται για την ύπαρξη ή μη της επίθεσης. Ωστόσο με τον συγκεκριμένο ρυθμό δειγματοληψίας ο μηχανισμός επιτυγχάνει να αναγνωρίσει πλήρως την επίθση χωρίς να παράγει σημαντικό ποσοστό λανθασμένων ειδοποιήσεων.

Ελλαττώνοντας πεταιτέρω τον ρυθμό δειγματοληψίας αρχίζει να παρατηρείται μείωση στην ικανότητα του μηχανισμού να αντιλαμβάνεται το σύνολο της επίθεσης. Όπως φαίνεται στο διάγραμμα 5.6, η προσαρμογή στην μειωμένη πληροφορία για τις ροές της κίνησης είναι ταχύτερη ειδικά για μεγαλύτερες τιμές του συντελεστή εξομάλυνσης a. Ταυτόχρονα, οι μικρές διακυμάνσεις οδηγούν σε περισσότερες



Σχήμα 5.6: Απόδοση μηχανισμού ανίχνευσης με ρυθμό δειγματοληψίας 1/200

λανθασμένες ενεργοποιήσεις του μηχανισμού, δημιουργώντας έτσι μία αυξητική τάση στις λανθασμένες ειδοποιήσεις οι οποίες ωστόσο εξακολουθούν να παραμένουν σε ιδιαίτερα χαμηλά ποσοστά.



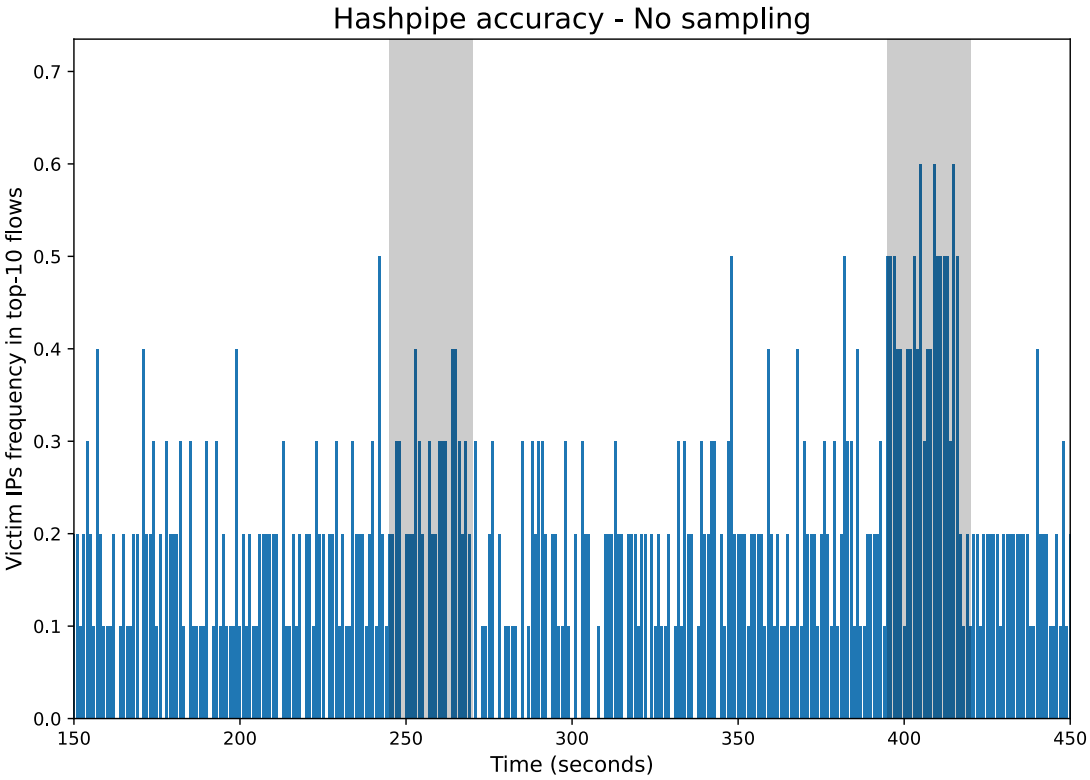
Σχήμα 5.7: Απόδοση μηχανισμού ανίχνευσης με ρυθμό δειγματοληψίας 1/1000

Τα οφέλη της δειγματοληψίας σταματούν έπειτα από έναν ορισμένο ρυθμό. Στην περίπτωση $f_s = \frac{1}{1000}$ που παρουσιάζεται στο διάγραμμα 5.7 ο μηχανισμός δίνει την ψευδαίσθηση της ικανότητας πλήρους αναγνώρισης της επίθεσης με αποτελέσματα καλύτερα ακόμη και από την βασική περίπτωση χωρίς δειγματοληψία. Ωστόσο τα ιδιαίτερα αυξημένα ποσοστά λανθασμένων ειδοποιήσεων, που αγγίζουν μέχρι και το 50% στην περίπτωση μικρού συντελεστή εξομάλυνσης, καταδεικνύουν ότι ο μηχανισμός ουδέποτε κατάφερε να προσαρμοστεί στις συνθήκες κίνησης λόγω της ιδιαίτερα μειωμένης διαθέσιμης πληροφορίας. Κατά το μεγαλύτερο μέρος της διάρκειας των πειραμάτων, άρα και κατά τη διάρκεια των επιθέσεων, παράγονται ειδοποιήσεις που υποδεικνύουν επίθεση και σε αυτό οφείλεται το εξαιρετικά

αυξημένο ποσοστό έγκυρης ανίχνευσης.

5.6 Ακρίβεια μηχανισμού HashPipe

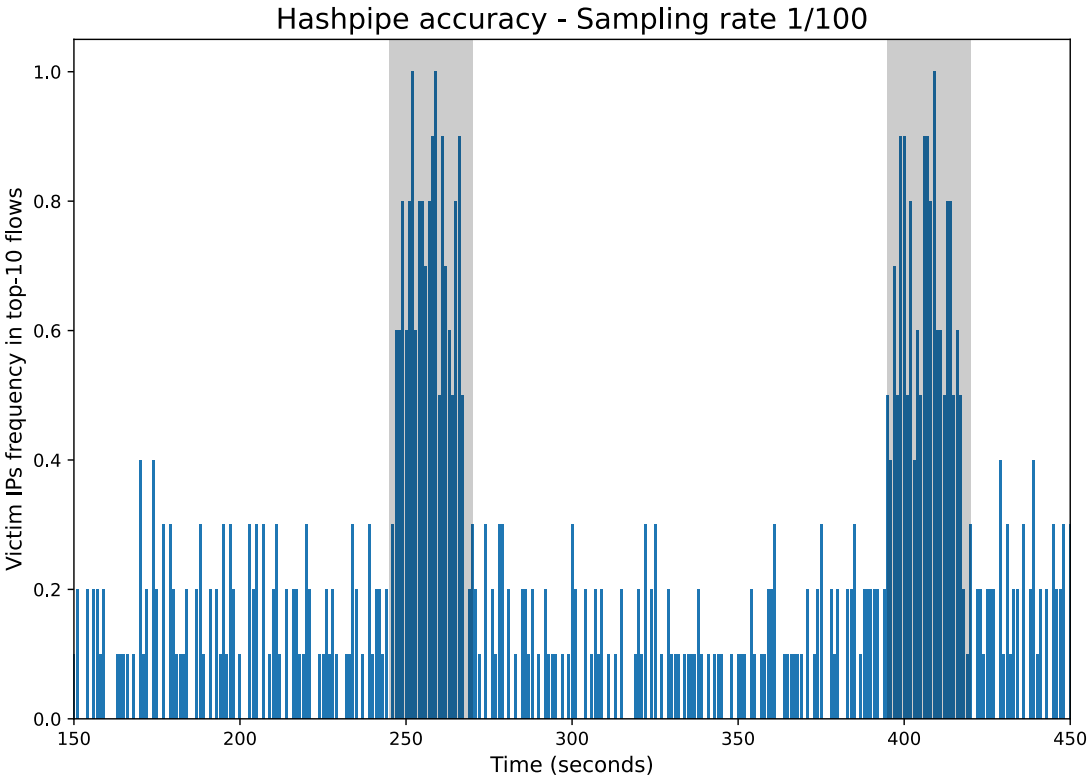
Κατά τη διάρκεια εκτέλεσης των πειραμάτων αποτελεσματικότητας του μηχανισμού ανίχνευσης συλλέγονται ανά δευτερόλεπτο και οι τιμές των δομών του hashpipe. Σκοπός είναι ο εντοπισμός των top flows οι οποίες εμφανίζονται στην κίνηση κατά τη διάρκεια της επίθεσης, ώστε το control plane να έχει καλύτερη εποπτεία για τους στόχους της επίθεσης. Εφόσον η κίνηση της επίθεσης έχει τροποποιηθεί ώστε να κατευθύνεται προς στόχους εντός των παρακολουθούμενων υποδικτύων, η εγκυρότητα του μηχανισμού βασίζεται στο κατά πόσον αυτοί οι στόχοι εμφανίζονται στην κορυφή του hashpipe. Η χωρητικότητα των δομών αποθήκευσης του μηχανισμού έχει οριστεί στους 128 καταχωρητές για κάθε ένα από τα τέσσερα επίπεδα της υλοποίησης.



Σχήμα 5.8: Απόδοση μηχανισμού hashpipe χωρίς δειγματοληψία

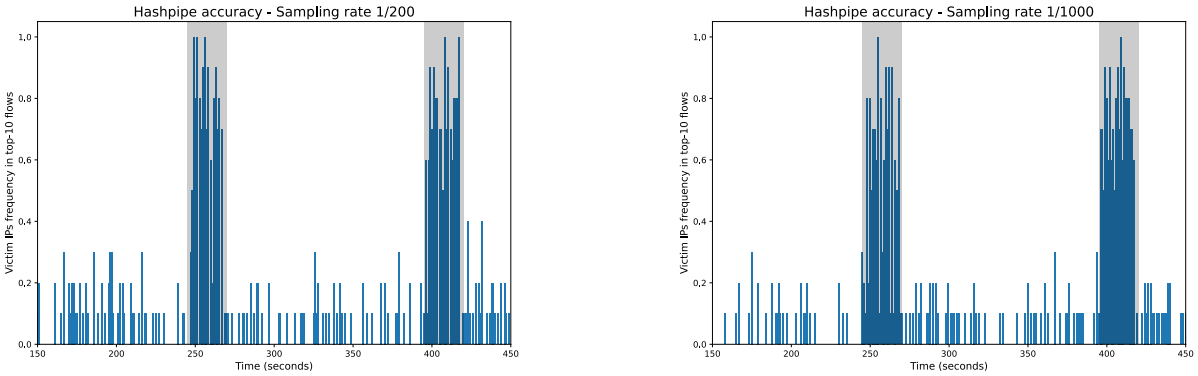
Στο διάγραμμα 5.8 παρουσιάζονται τα αποτελέσματα από τη συλλογή και επεξεργασία των αποθηκευμένων δεδομένων των τεσσάρων σταδίων του μηχανισμού hashpipe. Κατά τη διάρκεια της πρώτης επίθεσης, ο μηχανισμός δεν κατάφερε να συγκεντρώσει τις ροές προς τους αποδέκτες της. Κατά τη διάρκεια της δεύτερης επίθεσης φαίνεται πως υπήρξε αρκετά μεγαλύτερη συγκέντρωση των συγκεκριμένων ροών, ωστόσο όχι αρκετή ώστε να μπορούν ξεκάθαρα να διαχωριστούν από τις ροές της κανονικής κίνησης. Το αποτέλεσμα αυτό οφείλεται στο επιλεχθέν μέγεθος των δομών αποθήκευσης του hashpipe. Λόγω του ιδιαίτερα αυξημένου πλήθους ροών τόσο κατά τη διάρκεια της κανονικής κίνησης όσο και κατά τη διάρκεια της επίθεσης, προκαλούνται πολύ συχνές συγκρούσεις εντός των δομών με αποτέλεσμα να εκτοπίζονται πολύ συχνά ροές με μικρή κίνηση, που μπορεί να αναφέρονται είτε σε

κανονική είτε σε κακόβουλη κίνηση, εις βάρος άλλων με μεγαλύτερη κίνηση. Για το λόγο αυτό τα δεδομένα που συλλέγονται από τον μηχανισμό είναι αναξιόπιστα.



Σχήμα 5.9: Απόδοση μηχανισμού hashpipe με ρυθμό δειγματοληψίας 1/100

Με την εφαρμογή δειγματοληψίας ακόμη και με σχετικά υψηλό ρυθμό ($f_s = \frac{1}{100}$) η αποτελεσματικότητα του μηχανισμού βελτιώνεται σε πολύ μεγάλο βαθμό. Όπως παρουσιάζεται στο διάγραμμα 5.9, οι ροές προς τους αποδέκτες της επίθεσης κυριαρχούν στις δομές του hashpipe. Οι συγκρούσεις είναι πιο περιορισμένες και έτσι μπορεί να δοθεί έμφαση στις ροές κακόβουλης κίνησης.



(a) $f_s = 1/200$ (b) $f_s = 1/1000$

Σχήμα 5.10: Απόδοση μηχανισμού hashpipe με ρυθμούς δειγματοληψίας 1/200 και 1/1000

Οι μικρότεροι ρυθμοί δειγματοληψίας, όπως φαίνεται στο διάγραμμα 5.10, ευνοούν ακόμη περισσότερο τον μηχανισμό του HashPipe, ωστόσο όπως και στην περίπτωση της ανίχνευσης της επίθεσης, η πολύ

μικρή δειγματοληψία μπορεί να οδηγήσει σε λανθασμένη σήμανση μιας ροής ως κακόβουλης λόγω έλλειψης πληροφορίας από τον χαμηλό αριθμό εισαρχόμενων πακέτων.

Κεφάλαιο 6

Συμπεράσματα και Μελλοντικές Επεκτάσεις

6.1 Σύνοψη

Στο πλαίσιο της παρούσας διπλωματικής εργασίας υλοποιήθηκε η επέκταση ενός μηχανισμού ανίχνευσης κατανεμημένων επιθέσεων άρνησης παροχής υπηρεσίας (Destributed Denial of Service attacks - DDoS) στο επίπεδο δεδομένων με την προσθήκη ενός μηχανισμού υπόδειξης του αποδέκτη της κακόβουλης κίνησης αλλά και του τύπου της. Επιπλέον, εφαρμόστηκε δειγματοληψία πακέτων με σκοπό τον περιορισμό της ανάγκης ανάλυσης του συνόλου της διερχόμενης κίνησης με ταυτόχρονη διατήρηση της ακρίβειας ανίχνευσης των επιθέσεων.

Αρχικά παρουσιάστηκαν περιληπτικά σχετικές εργασίες που πραγμαγματοποιούν ανάλυση της κίνησης και ανίχνευση επιθέσεων στο επίπεδο ελέγχου καθώς και σχετικές δημοσιεύσεις. Για την υλοποίηση του προτεινόμενου μηχανισμού χρησιμοποιήθηκε η γλώσσα προγραμματισμού P4, μια γλώσσα ειδικού σκοπού που επιτρέπει τον σχεδιασμό και υλοποίηση διαδικασιών διαχείρησης και ανάλυσης της δικτυακής κίνησης οι οποίες θα εκτελούνται σε συμβατές δικτυακές συσκευές (μεταγωγείς ή κάρτες).

Η ανίχνευση της επίθεσης επιτυγχάνεται με την ανάλυση της κίνησης με παραμετρικές στατιστικές μεθόδους (απλή εκθετική εξομάλυνση) και τον ορισμό κατωφλιών για το πλήθος των μοναδικών ροών της κίνησης αλλά και για την ασυμμετρία εισερχόμενων και εξερχόμενων πακέτων. Η πληροφορία για τον αποδέκτη της επίθεσης συλλέγεται από τον μηχανισμό HashPipe ο οποίος διατηρεί τις ροές που εμφανίζουν τον μεγαλύτερο όγκο κίνησης.

Η πειραματική αξιολόγηση του μηχανισμού πραγματοποιήθηκε σε πραγματικό εξοπλισμό του εργαστηρίου και συγκεκριμένα σε κάρτα Netronome SmartNIC. Για τα πειράματα χρησιμοποιήθηκαν καταγραφές κίνησης από το WIDE project και την μελέτη των Booters. Η εφαρμογή δειγματοληψίας στα πακέτα που πρόκειται να επεξεργασθούν από τον μηχανισμό οδήγησε στη βελτίωση της ορθότητας ανίχνευσης με τον μηχανισμό ανίχνευσης να παρουσιάζει αμελητέα μείωση της ακρίβειάς του για σχετικά μεγάλους ρυθμούς δειγματοληψίας. Ο μηχανισμός HashPipe ωφελείται εξίσου καθώς περιορίζονται οι συγκρούσεις μεταξύ ροών μεγάλου όγκου καλόβουλης και κακόβουλης κίνησης με τις δεύτερες να επικρατούν σημαντικά κατά τη διάρκεια της επίθεσης. Ωστόσο ο αυθαίρετα μικρός ρυθμός δειγματοληψίας οδηγεί στην μεγάλη αύξηση των λανθασμένων ανιχνεύσεων λόγω της έλλειψης πληροφορίας από τον μικρό αριθμό επεξεργαζόμενων πακέτων. Ρυθμοί δειγματοληψίας $f_s = \frac{1}{100}$ και $f_s = \frac{1}{200}$ δείχνουν να βελτιώνουν την επεξεργαστική ικανότητα του μηχανισμού χωρίς να επηρεάζεται η ακρίβεια της ανίχνευσης, διατηρώντας ταυτόχρονα σε πολύ χαμηλά επίπεδα τις λανθασμένες ανιχνεύσεις.

6.2 Μελλοντικές εργασίες

Ο προτεινόμενος μηχανισμός μπορεί να επεκταθεί περαιτέρω με την χρήση διαφορετικών στατιστικών μεθόδων (μη παραμετρικών) ή με χρήση αλγορίθμων μηχανικής μάθησης για τον υπολογισμό των κατωφλιών και των χαρακτηριστικών της κίνησης.

Ιδιαίτερο ερευνητικό ενδιαφέρον έχει η μελέτη της δυνατότητας εφαρμογής πιο σύνθετων μεθόδων και αλγορίθμων στο επίπεδο δεδομένων λαμβάνοντας υπόψη τους περιορισμούς της γλώσσας P4 και διατηρώντας χαμηλό τον αντίκτυπο της επιπλεόν αυτής επεξεργασίας στην λειτουργία και την ταχύτητα προώθησης των πακέτων από τη δικτυακή συσκευή.

Τέλος, μία ιδέα για την πρακτική εφαρμογή αυτού του μηχανισμού θα ήταν η ενσωμάτωσή του σε κάποιο επαγγελματικό, ερευνητκό ή ακαδημαϊκό δίκτυο προκειμένου να μελετηθεί η συνεισφορά του στην αντιμετώπιση επιθέσεων DDoS σε πραγματικές συνθήκες.

Παράρτημα Α΄

Προγράμματα ανάλυσης και μετρήσεων σε Python

A.1 Ανάγνωση Digests από το BMv2

```python
#!/usr/bin/python3

import argparse
import json
from struct import unpack as _u, pack as _p

class Preamble:
    def __init__(self, obj):
        self.count = obj["count"]
        self.id = int(obj["desc"]["id"])
        self.app_id = int(obj["desc"]["app_id"])
        self.name = obj["desc"]["name"]
        self.field_list_name = obj["desc"]["field_list_name"]
        self.fields = dict(
                        map(
                            lambda item: (item["name"], int(item["width"])),
                            obj["desc"]["fields"]
                        )
                    )

class DigestBase:
    def __init__(self, item):
        self.preamble = Preamble(item[0])
        self.raw_message = item[1]

class InfoMessage:
    def __init__(self, raw_message):
        self.epoch = int(raw_message[0][1], 16)
        self.count = int(raw_message[1][1], 16)

    def __str__(self):
        return f"Epoch {self.epoch}: {self.count} events"

class AttackMessage:
    def __init__(self, raw_message):
        self.dstAddr = "{0}.{1}.{2}.{3}".format(*_u("!BBBB", _p("!L", int(raw_message[0][1], 16))))
        self.hashPipeKey = raw_message[1][1]
        self.epoch = int(raw_message[2][1], 16)
        self.dst_subnet_code = int(raw_message[3][1], 16)

    def __str__(self):
        return f"{self.dstAddr} generated attack message during epoch {self.epoch}. Code = {self.dst_subnet_code}, Key = {self.hashPipeKey}"
```

```
43
44 class Digest(DigestBase):
45     def __init__(self, item):
46         super().__init__(item)
47         if self.preamble.id == 0:
48             self.type = "info"
49             self.message = InfoMessage(self.raw_message)
50         elif self.preamble.id == 1:
51             self.type = "attack"
52             self.message = AttackMessage(self.raw_message)
53         else:
54             raise ValueError(f"Unknown digest type id: {self.preamble.id}")
55
56     def __str__(self):
57         return self.message.__str__()
58
59 def main():
60     parser = argparse.ArgumentParser()
61
62     parser.add_argument("-f", metavar="<digest file>", dest="digestfile", required=True)
63
64     args = parser.parse_args()
65
66     with open(args.digestfile, 'r') as f:
67         digests = json.loads(f.read())
68
69     for d in digests:
70         print(Digest(d))
71
72 if __name__ == '__main__':
73     main()
```

Κώδικας A.1: Ανάγνωση digests από το BMv2

A.2 Μετατροπή διευθύνσεων MAC σε pcap αρχεία

```
1  import sys
2  from struct import unpack as _u
3  from binascii import unhexlify as _uh
4
5  if __name__ == '__main__':
6      fin = open(sys.argv[1], 'rb')
7      fout = open(sys.argv[2], 'wb')
8
9      dmac = _uh("xx:xx:xx:xx:xx:xx".replace(':', ''))
10     smac = _uh("yy:yy:yy:yy:yy:yy".replace(':', ''))
11
12     # preamble
13     fout.write(fin.read(24))
14
15     while True:
16         # read captured item header
17         hdr = fin.read(16)
18         # break if no more items
19         if len(hdr) < 16:
20             break
21         # write item header
22         fout.write(hdr)
23         # extract packet length
24         pl = _u("<8xI4x", hdr)[0]
25         # read packet
26         pkt = fin.read(pl)
27         # write new dmac
28         fout.write(dmac)
29         # write new smac
30         fout.write(smac)
31         # write rest of Packets
```

```
        fout.write(pkt[12:])

    fin.close()
    fout.close()
```

Κώδικας Α.2: Μετατροπή MAC διευθύνσεων καταγραφής

Α.3 Παρακολούθηση ρυθμού εξερχόμενων πακέτων

```
#!/usr/bin/python3

import os
import sys
import time
import json
import subprocess
from datetime import datetime
from argparse import ArgumentParser

def read_ethtool_statistics(outfile, sleeptime=1, verbose=False):
    cmd = '/sbin/ethtool -S enp6s16 | grep -E "^\s*(rx_packets|tx_packets)"'
    values = {'rx_packets:': [], 'tx_packets:': []}

    prev_rx = None
    prev_tx = None
    max_len = 0

    rx = None
    tx = None

    while True:
        try:
            out = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE).communicate()[0]
            result = out.decode().split()

            if len(result) != 4:
                raise ValueError("ethtool results not in correct format")

            values[result[0]].append(int(result[1]))
            values[result[2]].append(int(result[3]))

            if verbose:
                rx = f"{float(int(result[1]) - prev_rx) / sleeptime} pps" if prev_rx else f"{int(result[1])} packets"
                tx = f"{float(int(result[3]) - prev_tx) / sleeptime} pps" if prev_tx else f"{int(result[3])} packets"
                prev_rx = int(result[1])
                prev_tx = int(result[3])

                infoline = f"rx_packets: {rx} || tx_packets: {tx}"
                max_len = max(max_len, len(infoline))

                print(" " * max_len, end='\r')
                print(infoline, end='\r')

            time.sleep(sleeptime)

        except KeyboardInterrupt:
            print(" " * max_len, end='\r')
            print("DONE")
            break

    #with open(outfile, 'a') as f:
     #   f.write(json.dumps(values))

def generate_parser():
    parser = ArgumentParser(
```

```
57          prog="read_ethtool",
58          description="Read ethtool statistics and store data in json format"
59      )
60
61      parser.add_argument(
62          "-C",
63          metavar="<dir>",
64          dest="home",
65          required=False,
66          default='./',
67          help="If provided, paths will be relative to this"
68      )
69
70      parser.add_argument(
71          "-t",
72          metavar="n",
73          dest="threadstatstime",
74          required=False,
75          type=float,
76          default=1,
77          help="Retrieve statistics every n seconds"
78      )
79
80      parser.add_argument(
81          "-o",
82          metavar="<filename>",
83          dest="outfile",
84          required=False,
85          default=f"results_ethtool_{datetime.now().strftime('%Y_%m_%d_%H_%M')}",
86          help="Output filename"
87      )
88
89      parser.add_argument(
90          "-q",
91          dest="quiet",
92          required=False,
93          action="store_true",
94          default=False,
95          help="Don't output progress"
96      )
97
98      return parser
99
100 def main():
101     parser = generate_parser()
102     args = parser.parse_args()
103     params = vars(args)
104
105     if os.path.isdir(params['home']):
106         params['outfile'] = os.path.join(params['home'], params['outfile'])
107     else:
108         raise ArgumentTypeError(f"{params['home']} is not a directory")
109
110     read_ethtool_statistics(params['outfile'], params['threadstatstime'], not params['quiet'])
111
112 if __name__ == '__main__':
113     try:
114         main()
115     except KeyboardInterrupt:
116         print('Interrupted')
117         sys.exit(0)
118     except SystemExit:
119         os._exit(0)
```

Κώδικας Α.3: Παρακολούθηση ρυθμού εξερχόμενων πακέτων

A.4 Αρχικοποίηση δομών κάρτας και καταγραφή ειδοποιήσεων

```
#!/usr/bin/python3

import os
import sys
import json
import argparse
from datetime import datetime
import subprocess
import signal
import time
import threading

def reset_nic(s, a, k):
    cmd = f"/home/christos/scripts/reset_nic.py {s} {a} {k} 2>/dev/null"
    subprocess.run(cmd, shell=True)
    return

def poll_digests(polltime, codedir, filename):
    cmd = f"rtecli -j digests poll -t {polltime} 2>/dev/null"
    basedir = f"/home/christos/results/digests/{codedir}"
    os.makedirs(basedir, exist_ok=True)
    resfile = f"{basedir}/{filename}"

    try:
        fres = open(resfile, 'w')

        out = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE).communicate()[0]

        fres.write(json.dumps(out.decode()))

    except:
        pass

    finally:
        fres.close()
        os.chown(basedir, 1001, 1001)

    return

def poll_hashpipe(codedir, resultsdir):
    cmd = f"/home/christos/scripts/read_hashpipe.py -c {codedir} -d {resultsdir} -z 2>/dev/
    null"
    subprocess.run(cmd, shell=True)
    return

def main():
    parser = argparse.ArgumentParser(
            prog="run_testbed",
            description="Run experiments"
    )

    parser.add_argument(
            "-c",
            metavar="codedir",
            dest="codedir",
            required=True,
            help="Codedir Name"
    )

    parser.add_argument(
            "-t",
            metavar="polltime",
            dest="polltime",
```

```
63              required=True,
64              type=int,
65              help="Polltime"
66      )
67
68      parser.add_argument(
69              "-s",
70              metavar="sampling",
71              dest="sampling",
72              required=True,
73              type=int,
74              help="Sampling"
75      )
76
77      parser.add_argument(
78              "-a",
79              metavar="threshold_a",
80              dest="threshold_a",
81              required=True,
82              type=int,
83              help="Threshold a"
84      )
85
86      parser.add_argument(
87              "-k",
88              metavar="threshold_k",
89              dest="threshold_k",
90              required=True,
91              type=int,
92              help="Threshold k"
93      )
94
95      args = parser.parse_args()
96
97      nic = threading.Thread(target=reset_nic, name="reset_nic", args=(args.sampling, args.
        threshold_a, args.threshold_k))
98      nic.start()
99      nic.join()
100
101     digests = threading.Thread(target=poll_digests, name="poll_digests", args=(args.polltime,
        args.codedir, f"digests_s{args.sampling}_a{args.threshold_a}_k{args.threshold_k}.json"))
102     hashpipe = threading.Thread(target=poll_hashpipe, name="poll_hashpipe", args=(args.codedir
        , f"hashpipe_s{args.sampling}_a{args.threshold_a}_k{args.threshold_k}"))
103
104     input("\033[0;30;46mPress any key to start polling...\033[0;0m")
105     digests.start()
106     hashpipe.start()
107
108     digests.join()
109
110     if hashpipe.is_alive():
111         signal.pthread_kill(hashpipe.ident, signal.SIGKILL)
112
113     hashpipe.join()
114
115     try:
116         while True:
117             time.sleep(120)
118
119     except KeyboardInterrupt:
120         if digests.is_alive():
121             signal.pthread_kill(digests.ident, signal.SIGKILL)
122         if hashpipe.is_alive():
123             signal.pthread_kill(hashpipe.ident, signal.SIGKILL)
124
125         digests.join()
126         hashpipe.join()
127
128     except SystemExit:
```

```
        os._exit(0)

if __name__ == '__main__':
    main()
```

Κώδικας Α.4: Αρχικοποίηση δομών κάρτας και καταγραφή ειδοποιήσεων

Α.5 Αναπαραγωγή δικτυακής κίνησης πειραμάτων

```
#!/usr/bin/python3

import os
import sys
from argparse import ArgumentParser, ArgumentTypeError
import subprocess
import threading
import time

def run_normal(duration: int, pps: int) -> None:
    print("\033[0;37;44mReplaying normal traffic\033[0;0m")
    cmd = f'tcpreplay --intf1="ens16" --quiet --duration={duration} --pps={pps} normal.pcap 2
    >/dev/null'
    subprocess.run(cmd, shell=True)
    print("\033[0;37;44mNormal traffic completed\033[0;0m")

def run_attack(duration: int, delay: int, pps:int) -> None:
    time.sleep(delay)
    print("\033[0;35;41mReplaying attack traffic\033[0;0m")
    cmd = f'tcpreplay --intf1="ens16" --quiet --duration={duration} --pps={pps} attack.pcap 2
    >/dev/null'
    subprocess.run(cmd, shell=True)
    print("\033[0;30;43mAttack traffic completed\033[0;0m")

def generate_parser():
    parser = ArgumentParser(
        prog="run_traffic",
        description="Replay normal & attack pcap files"
    )

    parser.add_argument(
        "-tn", "--time-normal",
        metavar="N",
        dest="time_normal",
        required=True,
        type=int,
        help="Replay time for normal file in seconds"
    )

    parser.add_argument(
        "-pn", "--pps-normal",
        metavar="N",
        dest="pps_normal",
        required=False,
        default=5000,
        type=int,
        help="Packets per second to send from normal file"
    )

    parser.add_argument(
        "-pa", "--pps-attack",
        metavar="N",
        dest="pps_attack",
        required=False,
        default=50000,
        type=int,
        help="Packets per second to send from normal file"
    )
```

```
57
58      def time_delay_attack_t(s):
59          try:
60              ta, da = map(int, s.split(','))
61              return ta, da
62          except:
63              raise ArgumentTypeError("Duration and delay of attack traffic must be given in T,D
        format. Provide multiple tuples for multiple attacks")
64
65      parser.add_argument(
66          "-tada", "--time-delay-attack",
67          metavar="T,D",
68          dest="time_delay_attack",
69          required=True,
70          type=time_delay_attack_t,
71          nargs='+',
72          help="Duration - Delay tuple of attack traffic"
73      )
74
75      return parser
76
77
78  def main():
79      parser = generate_parser()
80      args = parser.parse_args()
81
82      normal = threading.Thread(target=run_normal, name="Normal Traffic", args=(args.time_normal
        , args.pps_normal))
83
84      attacks = []
85      for t,d in args.time_delay_attack:
86          attack = threading.Thread(target=run_attack, name="Attack Traffic", args=(t, d, args.
        pps_attack))
87          attacks.append(attack)
88
89      normal.start()
90      for attack in attacks:
91          attack.start()
92
93      for attack in attacks:
94          attack.join()
95      normal.join()
96
97      print("Traffic replay completed!")
98
99
100 if __name__ == '__main__':
101     try:
102         while True:
103             main()
104             q = input("\033[0;36;40mRestart? Press n for no, any other key for yes\033[0;0m")
105             if q == 'n':
106                 break
107     except KeyboardInterrupt:
108         print("Interrupted")
109         sys.exit(0)
110     except SystemExit:
111         os._exit(0)
```

Κώδικας Α.5: Αναπαραγωγή δικτυακής κίνησης πειραμάτων

Α.6 Καταγραφή δεδομένων δομής HashPipe

```
1  #!/usr/bin/python3
2
3  import os
4  import sys
```

```
import json
import time
from datetime import datetime
from argparse import ArgumentParser, ArgumentTypeError
import subprocess
import threading

def exec_rtecli_command(stopevent, registername, outfile, sleeptime=1):
    cmd = f"/usr/local/bin/rtecli -j registers -r {registername} get"
    data = threading.local()
    data.values = []
    while not stopevent.is_set():
        try:
            out = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE).communicate()[0]
            data.values.append({"timestamp": time.time(), "register_data": json.loads(out.
    decode())})
        except:
            pass
        time.sleep(sleeptime)
    with open(outfile, 'a') as f:
        f.write(json.dumps(data.values))

def generate_parser():
    parser = ArgumentParser(
        prog="read_hashpipe",
        description="Read hashpipe register values and store data in json format"
    )

    parser.add_argument(
        "-c",
        metavar="<dir>",
        dest="codedir",
        required=True,
        help="Code directory"
    )

    parser.add_argument(
        "-d",
        metavar="<dir>",
        dest="results",
        required=True,
        help="Results directory"
    )

    parser.add_argument(
        "-z",
        dest="zip",
        required=False,
        action="store_true",
        default=False,
        help="Create zip file with results"
    )

    return parser

def main():
    parser = generate_parser()
    args = parser.parse_args()

    stop = threading.Event()
    threads = []

    basedir = f"/home/christos/results/hashpipe/{args.codedir}"
    resultsdir = os.path.join(basedir, args.results)
    os.makedirs(resultsdir, exist_ok=True)

    for stage in range(1,5):
        for type in ["keys", "values", "epoch"]:
            registername = f"r_stage_{stage}_{type}"
```

```
            outfile = os.path.join(resultsdir, registername)
            thread = threading.Thread(target=exec_rtecli_command, name=registername, args=(
    stop, registername, outfile))
            threads.append(thread)

    thread = threading.Thread(target=exec_rtecli_command, name="r_global_epoch", args=(stop, "
    r_global_epoch", os.path.join(resultsdir, "r_global_epoch")))
    threads.append(thread)

    for thread in threads:
        thread.start()

    while True:
        try:
            time.sleep(60)
        except KeyboardInterrupt:
            break

    stop.set()

    for thread in threads:
        thread.join()

    if args.zip:
        subprocess.run(f"tar -czf {basedir}/{args.results}.tar.gz {resultsdir}/", shell=True)

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print("Interrupted")
        sys.exit(0)
    except SystemExit:
        os._exit(0)
```

Κώδικας Α.6: Καταγραφή δεδομένων δομής HashPipe

Παράρτημα Β΄

Πλήρης κώδικας μηχανισμού

```
#include <core.p4>
#include <v1model.p4>

#include "definitions.p4"

#include "parsers.p4"
#include "checksums.p4"
#include "ingress.p4"
#include "egress.p4"

V1Switch(
    HeaderParser(),
    VerifyIPv4Checksum(),
    IngressLogic(),
    EgressLogic(),
    UpdateIPv4Checksum(),
    HeaderDeparser()
) main;
```

Κώδικας B.1: code.p4

```
const bit<64> EPOCH_DURATION_NANOSEC = 2_000_000_000;

const bit<32> MONITORING_CAPACITY = 256;
const bit<32> CODE_SUBNET_UNMONITORED = 256;

const bit<32> ASSYMETRY_SRC_FACTOR = 20;
const bit<32> ASSYMETRY_DST_FACTOR = 10;

const bit<32> HASHPIPE_STAGE_CAPACITY = 128;

typedef bit<64> hashPipeKey_t;
typedef bit<16> hashPipeValue_t;

typedef bit<9>  egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ipv4Addr_t;

typedef bit<16> EtherType;
typedef bit<8> ProtoType;

const EtherType EtherType_IPv4 = 0x0800;
const ProtoType ProtoType_TCP = 0x06;
const ProtoType ProtoType_UDP = 0x11;

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    EtherType   etherType;
}
```

```
30
31 header ipv4_t {
32     bit<4>    version;
33     bit<4>    ihl;
34     bit<8>    diffServ;
35     bit<16>   totalLen;
36     bit<16>   identification;
37     bit<3>    flags;
38     bit<13>   fragOffset;
39     bit<8>    ttl;
40     ProtoType protocol;
41     bit<16>   hdrChecksum;
42     ipv4Addr_t srcAddr;
43     ipv4Addr_t dstAddr;
44 }
45
46 header ipv4_options_t {
47     varbit<320> options;
48 }
49
50 header tcp_udp_port_t {
51     bit<16> srcPort;
52     bit<16> dstPort;
53 }
54
55 header intrinsic_metadata_t {
56     bit<64> ingress_global_timestamp;
57     bit<64> current_global_timestamp;
58 }
59
60 struct metadata {
61     bit<32> epoch;
62     hashPipeKey_t hashPipeKey;
63     hashPipeValue_t hashPipeValue;
64     bit<32> src_subnet_code;       // SOURCE SUBNET INDEX
65     bit<32> dst_subnet_code;       // DESTINATION SUBNET INDEX
66     bit<32> src_stats_index;     // STATS REG INDEX: SRC
67     bit<32> dst_stats_index;     // STATS REG INDEX: DST
68     bit<1> flag_total_flow_count; // FLAG 1: TOTAL FLOW COUNT EXCEEDED THRESHOLD
69     bit<1> flag_subnet_flow_count;   // FLAG 2: PER DESTINATION SUBNET FLAG EXCEEDED THRESHOLD
70     bit<1> flag_subnet_flow_assymetry;  // FLAG 3:
71 }
72
73 struct headers {
74     ethernet_t   ethernet;
75     ipv4_t       ipv4;
76     ipv4_options_t ipv4_options;
77     tcp_udp_port_t  ports;
78     intrinsic_metadata_t intrinsic_metadata;
79 }
```

Κώδικας B.2: definitions.p4

```
1 parser HeaderParser(packet_in packet, out headers hdr, inout metadata meta, inout
      standard_metadata_t standard_metadata) {
2
3     state start {
4         transition parse_ethernet;
5     }
6
7     state parse_ethernet {
8         packet.extract(hdr.ethernet);
9         transition select(hdr.ethernet.etherType) {
10            EtherType_IPv4: parse_ipv4;
11            default: accept;
12        }
13    }
14
15    state parse_ipv4 {
```

```
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.ihl) {
            5: check_ports;
            _: parse_ipv4_options;
        }
    }

    state parse_ipv4_options {
        packet.extract(hdr.ipv4_options, (bit<32>)(((bit<16>)hdr.ipv4.ihl - 5) * 32) );
        transition check_ports;
    }

    state check_ports {
        transition select(hdr.ipv4.protocol) {
            ProtoType_TCP: parse_tcp_udp_port;
            ProtoType_UDP: parse_tcp_udp_port;
            default: accept;
        }
    }

    state parse_tcp_udp_port {
        packet.extract(hdr.ports);
        transition accept;
    }

}

control HeaderDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
        packet.emit(hdr.ipv4_options);
        packet.emit(hdr.ports);
    }
}
```

Κώδικας B.3: parsers.p4

```
control VerifyIPv4Checksum(inout headers hdr, inout metadata meta) {
    apply {
        verify_checksum(
            hdr.ipv4.isValid(),
            {
                hdr.ipv4.version,
                hdr.ipv4.ihl,
                hdr.ipv4.diffServ,
                hdr.ipv4.totalLen,
                hdr.ipv4.identification,
                hdr.ipv4.flags,
                hdr.ipv4.fragOffset,
                hdr.ipv4.ttl,
                hdr.ipv4.protocol,
                hdr.ipv4.srcAddr,
                hdr.ipv4.dstAddr
            },
            hdr.ipv4.hdrChecksum,
            HashAlgorithm.csum16
        );
    }
}

control UpdateIPv4Checksum(inout headers  hdr, inout metadata meta) {
    apply {
        update_checksum(
            hdr.ipv4.isValid(),
            {
                hdr.ipv4.version,
                hdr.ipv4.ihl,
                hdr.ipv4.diffServ,
```

```
32                  hdr.ipv4.totalLen,
33                  hdr.ipv4.identification,
34                  hdr.ipv4.flags,
35                  hdr.ipv4.fragOffset,
36                  hdr.ipv4.ttl,
37                  hdr.ipv4.protocol,
38                  hdr.ipv4.srcAddr,
39                  hdr.ipv4.dstAddr
40              },
41              hdr.ipv4.hdrChecksum,
42              HashAlgorithm.csum16
43          );
44      }
45  }
```

Κώδικας B.4: checksums.p4

```
control IngressLogic(inout headers hdr, inout metadata meta, inout standard_metadata_t
    standard_metadata) {

    register<bit<16>>(2) r_sample_interval;
    register<bit<16>>(2) r_sample_count;
    register<bit<64>>(1) r_epoch_duration;

    register<bit<32>>(1) r_global_epoch;
    register<bit<64>>(1) r_last_ingress_timestamp;

    register<bit<32>>(1) r_global_packet_counter;

    register<bit<32>>(1) r_epoch_flow_counter;
    register<bit<32>>(1) r_threshold_a;
    register<bit<32>>(1) r_threshold_b;
    register<bit<32>>(1) r_threshold_k;
    register<bit<32>>(1) r_threshold_m;
    register<bit<32>>(1) r_threshold_s;

    register<bit<32>>(65536) r_uniqueness_sketch_1;
    register<bit<32>>(65536) r_uniqueness_sketch_2;
    register<bit<32>>(65536) r_uniqueness_sketch_3;

    register<bit<16>>(MONITORING_CAPACITY) r_total_flows_per_subnet;
    register<bit<32>>(MONITORING_CAPACITY) r_last_flow_epoch_per_subnet;
    register<bit<32>>(MONITORING_CAPACITY) r_ratio_factor_per_subnet;

    register<bit<32>>(MONITORING_CAPACITY * 4) r_assymetry_packet_counter;
    register<bit<32>>(MONITORING_CAPACITY * 4) r_assymetry_last_epoch;
    register<bit<32>>(MONITORING_CAPACITY) r_assymetry_factor_tcp;
    register<bit<32>>(MONITORING_CAPACITY) r_assymetry_factor_udp;

    register<bit<1>>(3) r_flags;
    register<bit<32>>(1) r_flags_epoch;
    register<bit<32>>(1) r_current_epoch_digests;

    register<hashPipeKey_t>(HASHPIPE_STAGE_CAPACITY)    r_stage_1_keys;
    register<hashPipeValue_t>(HASHPIPE_STAGE_CAPACITY)  r_stage_1_values;
    register<bit<32>>(HASHPIPE_STAGE_CAPACITY)          r_stage_1_epoch;
    register<bit<32>>(2)                                r_stage_1_coeff;

    register<hashPipeKey_t>(HASHPIPE_STAGE_CAPACITY)    r_stage_2_keys;
    register<hashPipeValue_t>(HASHPIPE_STAGE_CAPACITY)  r_stage_2_values;
    register<bit<32>>(HASHPIPE_STAGE_CAPACITY)          r_stage_2_epoch;
    register<bit<32>>(2)                                r_stage_2_coeff;

    register<hashPipeKey_t>(HASHPIPE_STAGE_CAPACITY)    r_stage_3_keys;
    register<hashPipeValue_t>(HASHPIPE_STAGE_CAPACITY)  r_stage_3_values;
    register<bit<32>>(HASHPIPE_STAGE_CAPACITY)          r_stage_3_epoch;
    register<bit<32>>(2)                                r_stage_3_coeff;

    register<hashPipeKey_t>(HASHPIPE_STAGE_CAPACITY)    r_stage_4_keys;
    register<hashPipeValue_t>(HASHPIPE_STAGE_CAPACITY)  r_stage_4_values;
    register<bit<32>>(HASHPIPE_STAGE_CAPACITY)          r_stage_4_epoch;
    register<bit<32>>(2)                                r_stage_4_coeff;

    counter(1, CounterType.packets) outer;
    counter(1, CounterType.packets) inner;

    action l2_forward() {
        if (standard_metadata.ingress_port == 0) {
            standard_metadata.egress_spec = 1;
        }
        else {
            standard_metadata.egress_spec = 0;
        }
    }

    action get_src_subnet_code(bit<32> subnet_code) {
```

```
69          meta.src_subnet_code = subnet_code;
70      }
71
72      action get_dst_subnet_code(bit<32> subnet_code) {
73          meta.dst_subnet_code = subnet_code;
74      }
75
76      table monitored_subnets_src {
77          key = {
78              hdr.ipv4.srcAddr : ternary;
79          }
80          actions = {
81              get_src_subnet_code;
82              NoAction;
83          }
84          size = MONITORING_CAPACITY + 2;
85          default_action = NoAction();
86      }
87
88      table monitored_subnets_dst {
89          key = {
90              hdr.ipv4.dstAddr : ternary;
91          }
92          actions = {
93              get_dst_subnet_code;
94              NoAction;
95          }
96          size = MONITORING_CAPACITY + 2;
97          default_action = NoAction();
98      }
99
100     action calculate_hash_index(in bit<32> lower_key, in bit<32> a, in bit<32> b, out bit<32>
    index) {
101         bit<32> _MOD = HASHPIPE_STAGE_CAPACITY - 1;
102         index = (((a * (lower_key & _MOD)) & _MOD) + b) & _MOD;
103     }
104
105     apply {
106         l2_forward();
107
108         outer.count(0);
109
110         if (hdr.ports.isValid()) {
111
112             monitored_subnets_src.apply();
113             monitored_subnets_dst.apply();
114
115             if (meta.src_subnet_code == CODE_SUBNET_UNMONITORED && meta.dst_subnet_code ==
    CODE_SUBNET_UNMONITORED) {
116                 return;
117             }
118
119             if (hdr.ports.srcPort < hdr.ports.dstPort) {
120                 meta.hashPipeKey = hdr.ipv4.dstAddr ++ hdr.ports.srcPort ++ 16w0;
121             }
122             else {
123                 meta.hashPipeKey = hdr.ipv4.dstAddr ++ 16w0 ++ hdr.ports.dstPort;
124             }
125
126             bit<16> interval_detection;
127             bit<16> interval_hashpipe;
128             bit<16> count_detection;
129             bit<16> count_hashpipe;
130             bit<1> epoch_reset;
131             bool willApplyDetection;
132             bool willApplyHashpipe;
133
134             r_sample_interval.read(interval_detection, 0);
135             r_sample_count.read(count_detection, 0);
```

```
136             count_detection = count_detection + 1;
137             willApplyDetection = (count_detection == interval_detection);
138 
139             r_sample_interval.read(interval_hashpipe, 1);
140             r_sample_count.read(count_hashpipe, 1);
141             count_hashpipe = count_hashpipe + 1;
142             willApplyHashpipe = (count_hashpipe == interval_hashpipe);
143 
144             epoch_reset = 0;
145 
146             if (willApplyDetection || willApplyHashpipe) {
147 
148                 inner.count(0);
149 
150                 bit<32> current_epoch;
151                 bit<64> last_timestamp;
152                 bit<64> current_timestamp;
153 
154                 bit<64> epoch_duration_nanosec;
155                 r_epoch_duration.read(epoch_duration_nanosec, 0);
156 
157                 r_global_epoch.read(current_epoch, 0);
158 
159                 r_last_ingress_timestamp.read(last_timestamp, 0);
160                 // intrinsic_metadata.ingress_global_timestamp
161                 // 64----------------------32----------------------0
162                 // | ------ SECONDS ------ | ---- NANOSECONDS ---- |
163                 current_timestamp = hdr.intrinsic_metadata.ingress_global_timestamp;
164 
165                 if ((current_timestamp - last_timestamp > epoch_duration_nanosec) &&
    willApplyDetection) {
166                     r_last_ingress_timestamp.write(0, current_timestamp);
167                     r_global_epoch.write(0, current_epoch + 1);
168 
169                     meta.epoch = current_epoch + 1;
170                     epoch_reset = 1;
171 
172                     bit<32> ced;
173                     r_current_epoch_digests.read(ced, 0);
174                     digest(
175                         32w2048,
176                         {
177                             current_epoch,
178                             ced
179                         }
180                     );
181                     r_current_epoch_digests.write(0, 0);
182                 }
183                 else {
184                     meta.epoch = current_epoch;
185                 }
186             }
187 
188             if (willApplyDetection) {
189                 bit<32> packet_count;
190 
191                 r_global_packet_counter.read(packet_count, 0);
192 
193                 if (epoch_reset == 1) {
194                     r_global_packet_counter.write(0, 1);
195 
196                     bit<32> f;
197                     bit<32> a1;
198                     bit<32> b1;
199                     bit<32> m;
200                     bit<32> s;
201 
202                     r_epoch_flow_counter.read(f, 0);
203                     r_threshold_a.read(a1, 0);
```

```
                    r_threshold_b.read(b1, 0);
                    r_threshold_m.read(m, 0);
                    r_threshold_s.read(s, 0);

                    if (meta.epoch == 1) {
                        m = f;
                        s = 0;
                    }
                    else {
                        m = ((a1 * f) + ((256 - a1) * m)) >> 8;
                        s = (m < f) ? ((b1 * (f - m)) + ((256 - b1) * s)) : ((b1 * (m - f)) + ((256 - b1) * s));
                        s = s >> 8;
                    }

                    r_threshold_m.write(0, m);
                    r_threshold_s.write(0, s);

                    r_epoch_flow_counter.write(0, 0);
                }
                else {
                    r_global_packet_counter.write(0, packet_count + 1);
                }

                meta.flag_total_flow_count = 0;
                meta.flag_subnet_flow_count = 0;

                if (meta.src_subnet_code == CODE_SUBNET_UNMONITORED && meta.dst_subnet_code != CODE_SUBNET_UNMONITORED) {
                    bit<32> hash_1;
                    bit<32> hash_2;
                    bit<32> hash_3;

                    hash(hash_1, HashAlgorithm.crc32, (bit<32>) 0, { hdr.ipv4.srcAddr,hdr.ipv4.dstAddr,hdr.ipv4.protocol,hdr.ports.srcPort,hdr.ports.dstPort }, (bit<32>)65536);
                    hash(hash_2, HashAlgorithm.crc16, (bit<32>) 0, { hdr.ipv4.srcAddr,hdr.ipv4.dstAddr,hdr.ipv4.protocol,hdr.ports.srcPort,hdr.ports.dstPort }, (bit<32>)65536);
                    hash(hash_3, HashAlgorithm.csum16, (bit<32>) 0, { hdr.ipv4.srcAddr,hdr.ipv4.dstAddr,hdr.ipv4.protocol,hdr.ports.srcPort,hdr.ports.dstPort }, (bit<32>)65536);

                    bit<32> last_epoch;
                    bit<1> is_new_flow;

                    is_new_flow = 0;

                    r_uniqueness_sketch_1.read(last_epoch, hash_1);
                    if (last_epoch != meta.epoch) {
                        is_new_flow = 1;
                        r_uniqueness_sketch_1.write(hash_1, meta.epoch);
                    }

                    r_uniqueness_sketch_2.read(last_epoch, hash_2);
                    if (last_epoch != meta.epoch) {
                        is_new_flow = 1;
                        r_uniqueness_sketch_2.write(hash_2, meta.epoch);
                    }

                    r_uniqueness_sketch_3.read(last_epoch, hash_3);
                    if (last_epoch != meta.epoch) {
                        is_new_flow = 1;
                        r_uniqueness_sketch_3.write(hash_3, meta.epoch);
                    }

                    bit<32> total_unique_flows;
                    bit<16> subnet_total_flows;

                    r_epoch_flow_counter.read(total_unique_flows, 0);
                    total_unique_flows = total_unique_flows + (bit<32>)is_new_flow;
                    r_epoch_flow_counter.write(0, total_unique_flows);
```

```
268
269                 if (is_new_flow == 1) {
270
271                     r_last_flow_epoch_per_subnet.read(last_epoch, meta.dst_subnet_code);
272
273                     if (last_epoch == meta.epoch) {
274                         r_total_flows_per_subnet.read(subnet_total_flows, meta.
    dst_subnet_code);
275                         subnet_total_flows = subnet_total_flows + 1;
276                     }
277                     else {
278                         r_last_flow_epoch_per_subnet.write(meta.dst_subnet_code, meta.
    epoch);
279                         subnet_total_flows = 1;
280                     }
281
282                     r_total_flows_per_subnet.write(meta.dst_subnet_code,
    subnet_total_flows);
283
284                 }
285                 else {
286                     r_total_flows_per_subnet.read(subnet_total_flows, meta.dst_subnet_code
    );
287                 }
288
289                 bit<32> k;
290                 bit<32> m;
291                 bit<32> s;
292
293                 r_threshold_m.read(m, 0);
294                 r_threshold_s.read(s, 0);
295                 r_threshold_k.read(k, 0);
296
297                 if ( total_unique_flows > m + ((k * s) >> 4) ) {
298                     meta.flag_total_flow_count = 1;
299                 }
300
301                 bit<32> subnet_ratio_factor;
302                 r_ratio_factor_per_subnet.read(subnet_ratio_factor, meta.dst_subnet_code);
303
304                 if ( (bit<32>)subnet_total_flows > ((subnet_ratio_factor *
    total_unique_flows) >> 8) ) {
305                     meta.flag_subnet_flow_count = 1;
306                 }
307             }
308
309             meta.flag_subnet_flow_assymetry = 0;
310
311             bit<32> inner_register_shift = hdr.ipv4.protocol == ProtoType_TCP ? (bit<32>)
    0 : MONITORING_CAPACITY * 2;
312
313             meta.src_stats_index = inner_register_shift + meta.src_subnet_code;
314             meta.dst_stats_index = inner_register_shift + MONITORING_CAPACITY + meta.
    dst_subnet_code;
315
316             bit<32> stats_packet_count;
317             bit<32> last_epoch;
318
319             if (meta.src_subnet_code != CODE_SUBNET_UNMONITORED) {
320                 r_assymetry_last_epoch.read(last_epoch, meta.src_stats_index);
321
322                 if (last_epoch != meta.epoch) {
323                     r_assymetry_last_epoch.write(meta.src_stats_index, meta.epoch);
324                     r_assymetry_packet_counter.write(meta.src_stats_index, 1);
325                 }
326                 else {
327                     r_assymetry_packet_counter.read(stats_packet_count, meta.
    src_stats_index);
328                     r_assymetry_packet_counter.write(meta.src_stats_index,
```

```
    stats_packet_count + 1);
                }
            }


            if (meta.dst_subnet_code != CODE_SUBNET_UNMONITORED) {
                r_assymetry_last_epoch.read(last_epoch, meta.dst_stats_index);

                if (last_epoch != meta.epoch) {
                    r_assymetry_last_epoch.write(meta.dst_stats_index, meta.epoch);
                    r_assymetry_packet_counter.write(meta.dst_stats_index, 1);
                    stats_packet_count = 0;
                }
                else {
                    r_assymetry_packet_counter.read(stats_packet_count, meta.
    dst_stats_index);
                    r_assymetry_packet_counter.write(meta.dst_stats_index,
    stats_packet_count + 1);
                }

                bit<32> stats_src_packet_count;
                bit<32> stats_dst_packet_count;

                stats_dst_packet_count = (bit<32>) (stats_packet_count + 1);

                r_assymetry_packet_counter.read(stats_packet_count, meta.src_stats_index);
                stats_src_packet_count = (bit<32>) stats_packet_count;

                bit<32> asymmetry_factor;

                if (hdr.ipv4.protocol == ProtoType_TCP) {
                    r_assymetry_factor_tcp.read(asymmetry_factor, meta.dst_subnet_code);
                }
                else {
                    r_assymetry_factor_udp.read(asymmetry_factor, meta.dst_subnet_code);
                }

                stats_dst_packet_count = stats_dst_packet_count * asymmetry_factor;

                stats_src_packet_count = stats_src_packet_count + 1;
                stats_dst_packet_count = stats_dst_packet_count + 1;

                if (ASSYMETRY_SRC_FACTOR * stats_src_packet_count < ASSYMETRY_DST_FACTOR *
     stats_dst_packet_count) {
                    meta.flag_subnet_flow_assymetry = 1;
                }
            }

            if (meta.dst_subnet_code != CODE_SUBNET_UNMONITORED) {
                r_flags.write(0, meta.flag_total_flow_count);
                r_flags.write(1, meta.flag_subnet_flow_count);
                r_flags.write(2, meta.flag_subnet_flow_assymetry);

                bit<32> current_epoch_digests;
                bit<32> flags_last_epoch;
                r_flags_epoch.read(flags_last_epoch, 0);

                if (meta.flag_total_flow_count == 1 && meta.flag_subnet_flow_count == 1 &&
     meta.flag_subnet_flow_assymetry == 1) {
                    if (flags_last_epoch != meta.epoch) {
                        digest(
                            (bit<32>) 1024,
                            {
                                hdr.ipv4.dstAddr,
                                meta.hashPipeKey,
                                meta.epoch,
                                meta.dst_subnet_code
                            }
                        );
```

```
                        r_flags_epoch.write(0, meta.epoch);
                    }
                    r_current_epoch_digests.read(current_epoch_digests, 0);
                    r_current_epoch_digests.write(0, current_epoch_digests + 1);
                }
            }

            count_detection = 0;
        }

        if (willApplyHashpipe) {

            if (meta.dst_subnet_code != CODE_SUBNET_UNMONITORED) {

                // meta.hashPipeKey = hdr.ipv4.dstAddr ++ hdr.ports.srcPort ++ hdr.ports.dstPort;

                bit<32> lower_key;
                lower_key = meta.hashPipeKey[31:0];

                bit<32>         hashTableIndex;
                hashPipeKey_t   prev_key;
                hashPipeValue_t prev_value;
                bit<32>         prev_epoch;
                hashPipeKey_t   key_to_save;
                hashPipeValue_t value_to_save;
                bool            willCarryKey;
                bit<32>         a;
                bit<32>         b;

                r_stage_1_coeff.read(a, 0);
                r_stage_1_coeff.read(b, 1);

                calculate_hash_index(lower_key, a, b, hashTableIndex);

                r_stage_1_keys.read(prev_key, hashTableIndex);
                r_stage_1_values.read(prev_value, hashTableIndex);
                r_stage_1_epoch.read(prev_epoch, hashTableIndex);

                willCarryKey = (prev_epoch == meta.epoch) && (prev_key != meta.hashPipeKey);
                value_to_save = (prev_epoch == meta.epoch) && (prev_key == meta.hashPipeKey) ? prev_value + 1 : 1;

                r_stage_1_keys.write(hashTableIndex, meta.hashPipeKey);
                r_stage_1_values.write(hashTableIndex, value_to_save);
                r_stage_1_epoch.write(hashTableIndex, meta.epoch);

                meta.hashPipeKey = willCarryKey ? prev_key : 0;
                meta.hashPipeValue = willCarryKey ? prev_value : 0;

                if (meta.hashPipeKey != 0) {
                    r_stage_2_coeff.read(a, 0);
                    r_stage_2_coeff.read(b, 1);

                    lower_key = meta.hashPipeKey[31:0];
                    calculate_hash_index(lower_key, a, b, hashTableIndex);

                    r_stage_2_keys.read(prev_key, hashTableIndex);
                    r_stage_2_values.read(prev_value, hashTableIndex);
                    r_stage_2_epoch.read(prev_epoch, hashTableIndex);

                    if (prev_epoch != meta.epoch || prev_key == meta.hashPipeKey) {
                        key_to_save = meta.hashPipeKey;
                        value_to_save = (prev_epoch != meta.epoch) ? meta.hashPipeValue : prev_value + meta.hashPipeValue;
                        meta.hashPipeKey = 0;
                        meta.hashPipeValue = 0;
                    }
```

```
458                         else {
459                             key_to_save         = (prev_value < meta.hashPipeValue) ? meta.
    hashPipeKey : prev_key;
460                             value_to_save       = (prev_value < meta.hashPipeValue) ? meta.
    hashPipeValue : prev_value;
461                             meta.hashPipeKey    = (prev_value < meta.hashPipeValue) ? prev_key
     : meta.hashPipeKey;
462                             meta.hashPipeValue  = (prev_value < meta.hashPipeValue) ?
    prev_value : meta.hashPipeValue;
463                         }
464 
465                         r_stage_2_keys.write(hashTableIndex, key_to_save);
466                         r_stage_2_values.write(hashTableIndex, value_to_save);
467                         r_stage_2_epoch.write(hashTableIndex, meta.epoch);
468                     }
469 
470                     if (meta.hashPipeKey != 0) {
471                         r_stage_3_coeff.read(a, 0);
472                         r_stage_3_coeff.read(b, 1);
473 
474                         lower_key = meta.hashPipeKey[31:0];
475                         calculate_hash_index(lower_key, a, b, hashTableIndex);
476 
477                         r_stage_3_keys.read(prev_key, hashTableIndex);
478                         r_stage_3_values.read(prev_value, hashTableIndex);
479                         r_stage_3_epoch.read(prev_epoch, hashTableIndex);
480 
481                         if (prev_epoch != meta.epoch || prev_key == meta.hashPipeKey) {
482                             key_to_save = meta.hashPipeKey;
483                             value_to_save = (prev_epoch != meta.epoch) ? meta.hashPipeValue :
    prev_value + meta.hashPipeValue;
484                             meta.hashPipeKey = 0;
485                             meta.hashPipeValue = 0;
486                         }
487                         else {
488                             key_to_save         = (prev_value < meta.hashPipeValue) ? meta.
    hashPipeKey : prev_key;
489                             value_to_save       = (prev_value < meta.hashPipeValue) ? meta.
    hashPipeValue : prev_value;
490                             meta.hashPipeKey    = (prev_value < meta.hashPipeValue) ? prev_key
     : meta.hashPipeKey;
491                             meta.hashPipeValue  = (prev_value < meta.hashPipeValue) ?
    prev_value : meta.hashPipeValue;
492                         }
493 
494                         r_stage_3_keys.write(hashTableIndex, key_to_save);
495                         r_stage_3_values.write(hashTableIndex, value_to_save);
496                         r_stage_3_epoch.write(hashTableIndex, meta.epoch);
497                     }
498 
499                     if (meta.hashPipeKey != 0) {
500                         r_stage_4_coeff.read(a, 0);
501                         r_stage_4_coeff.read(b, 1);
502 
503                         lower_key = meta.hashPipeKey[31:0];
504                         calculate_hash_index(lower_key, a, b, hashTableIndex);
505 
506                         r_stage_4_keys.read(prev_key, hashTableIndex);
507                         r_stage_4_values.read(prev_value, hashTableIndex);
508                         r_stage_4_epoch.read(prev_epoch, hashTableIndex);
509 
510                         if (prev_epoch != meta.epoch || prev_key == meta.hashPipeKey) {
511                             key_to_save = meta.hashPipeKey;
512                             value_to_save = (prev_epoch != meta.epoch) ? meta.hashPipeValue :
    prev_value + meta.hashPipeValue;
513                             meta.hashPipeKey = 0;
514                             meta.hashPipeValue = 0;
515                         }
516                         else {
```

```
517                             key_to_save         = (prev_value < meta.hashPipeValue) ? meta.
    hashPipeKey : prev_key;
518                             value_to_save       = (prev_value < meta.hashPipeValue) ? meta.
    hashPipeValue : prev_value;
519                         }
520
521                         r_stage_4_keys.write(hashTableIndex, key_to_save);
522                         r_stage_4_values.write(hashTableIndex, value_to_save);
523                         r_stage_4_epoch.write(hashTableIndex, meta.epoch);
524                     }
525                 }
526
527                 count_hashpipe = 0;
528             }
529
530             r_sample_count.write(0, count_detection);
531             r_sample_count.write(1, count_hashpipe);
532
533         }
534     }
535
536 }
```

Κώδικας B.5: ingress.p4

```
1 control EgressLogic(inout headers hdr, inout metadata meta, inout standard_metadata_t
    standard_metadata) {
2     apply {
3
4     }
5 }
```

Κώδικας B.6: egress.p4

Βιβλιογραφία

[1] Marinos Dimolianis, Adam Pavlidis, and Vasilis Maglaris. "A Multi-Feature DDoS Detection Schema on P4 Network Hardware". In: *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. 2020, pp. 1–6. DOI: 10.1109/ICIN48450.2020.9059327.

[2] Κωνσταντίνος-Χρήστος Μητρόπουλος and Βασίλειος Μάγκλαρης. "Ανίχνευση Επιθέσεων DDoS σε Προγραμματιζόμενο Επίπεδο Προώθησης Δεδομένων μέσω της Γλώσσας P4". National Technical University of Athens, 2019. URL: http://artemis.cslab.ece.ntua.gr:8080/jspui/handle/123456789/17397.

[3] *Defining Software Defined Networking*. URL: https://thenewstack.io/defining-software-defined-networking-part-1/.

[4] *OpenFlow Specification*. Mar. 2015. URL: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[5] Wenfeng Xia et al. "A Survey on Software-Defined Networking". In: *IEEE Communications Surveys Tutorials* 17.1 (2015), pp. 27–51. DOI: 10.1109/COMST.2014.2330903.

[6] Pat Bosshart et al. "P4: Programming Protocol-Independent Packet Processors". In: *SIGCOMM Comput. Commun. Rev.* 44.3 (July 2014), pp. 87–95. ISSN: 0146-4833. DOI: 10.1145/2656877.2656890. URL: https://doi.org/10.1145/2656877.2656890.

[7] *P4 - Language Consortium*. URL: https://p4.org/.

[8] *$P4_{16}$ Language Specification*. May 2021. URL: https://p4.org/p4-spec/docs/P4-16-v1.2.2.html.

[9] *P4 v1model*. URL: https://github.com/p4lang/tutorials/blob/master/P4_tutorial.pdf.

[10] *What is a DDoS attack?* URL: https://www.cloudflare.com/en-gb/learning/ddos/what-is-a-ddos-attack/.

[11] *What is IoT*. URL: https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/.

[12] Liying Li, Jianying Zhou, and Ning Xiao. "DDoS Attack Detection Algorithms Based on Entropy Computing". In: *Information and Communications Security*. Ed. by Sihan Qing, Hideki Imai, and Guilin Wang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 452–466. ISBN: 978-3-540-77048-0.

[13] *What is TCP SYN flood?* URL: https://www.cloudflare.com/en-gb/learning/ddos/syn-flood-ddos-attack/.

[14] *What are DNS amplification attacks?* URL: https://www.cloudflare.com/en-gb/learning/ddos/dns-amplification-ddos-attack/.

[15] Jelena Mirkovic and Peter Reiher. "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms". In: *SIGCOMM Comput. Commun. Rev.* 34.2 (Apr. 2004), pp. 39–53. ISSN: 0146-4833. DOI: 10.1145/997150.997156. URL: https://doi.org/10.1145/997150.997156.

[16] *DDoS attack trends for Q3 2021.* URL: https://radar.cloudflare.com/notebooks/ddos-2021-q3.

[17] *DDoS attacks in Q3 2021.* URL: https://securelist.com/ddos-attacks-in-q3-2021/104796/.

[18] *Global ddos summary - October 2021.* URL: https://horizon.netscout.com/?atlas=summary.

[19] *Bloom Filter.* URL: https://en.wikipedia.org/wiki/Bloom_filter.

[20] David Eppstein. *Bloom Filter Example.* By David Eppstein - self-made, originally for a talk at WADS 2007, Public Domain. 2007. URL: https://commons.wikimedia.org/w/index.php?curid=2609777.

[21] Moses Charikar, Kevin Chen, and Martin Farach-Colton. "Finding frequent items in data streams". In: *Theoretical Computer Science* 312.1 (2004). Automata, Languages and Programming, pp. 3–15. ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(03)00400-6. URL: https://www.sciencedirect.com/science/article/pii/S0304397503004006.

[22] Graham Cormode and S. Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications". In: *Journal of Algorithms* 55.1 (2005), pp. 58–75. ISSN: 0196-6774. DOI: https://doi.org/10.1016/j.jalgor.2003.12.001. URL: https://www.sciencedirect.com/science/article/pii/S0196677403001913.

[23] *Explaining count min sketches.* URL: https://stackoverflow.com/questions/6811351/explaining-the-count-sketch-algorithm.

[24] J Lu et al. "DDoS Attack Detection based on Non-parameter CUSUM". In: 2004.

[25] B Lin, O Li, and Q Liu. "DDoS attacks detection based on sequential change detection". In: *Computer Engineering* 31.9 (2005), pp. 135–137.

[26] Anukool Lakhina, Mark Crovella, and Christiphe Diot. "Characterization of Network-Wide Anomalies in Traffic Flows". In: *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement.* IMC '04. Taormina, Sicily, Italy: Association for Computing Machinery, 2004, pp. 201–206. ISBN: 1581138210. DOI: 10.1145/1028788.1028813. URL: https://doi.org/10.1145/1028788.1028813.

[27] Jianning Mai et al. "Impact of packet sampling on portscan detection". In: *IEEE Journal on Selected Areas in Communications* 24.12 (2006), pp. 2285–2298.

[28] Jianning Mai et al. "Is sampled data sufficient for anomaly detection?" In: *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement.* 2006, pp. 165–176.

[29] Daniela Brauckhoff et al. “Impact of packet sampling on anomaly detection metrics”. In: *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. 2006, pp. 159–164.

[30] Georgios Androulidakis, Vassilis Chatzigiannakis, and Symeon Papavassiliou. “Network anomaly detection and classification via opportunistic sampling”. In: *IEEE Network* 23.1 (2009), pp. 6–12. DOI: 10.1109/MNET.2009.4804318.

[31] *DDoS Attack Glossary: Top 12 Attack Vectors*. URL: https://www.cpomagazine.com/cyber-security/ddos-attack-glossary-top-12-attack-vectors/.

[32] Ramin Sadre, Anna Sperotto, and Aiko Pras. “The effects of DDoS attacks on flow monitoring applications”. In: *2012 IEEE Network Operations and Management Symposium*. 2012, pp. 269–277. DOI: 10.1109/NOMS.2012.6211908.

[33] Ângelo Cardoso Lapolli, Jonatas Adilson Marques, and Luciano Paschoal Gaspary. “Offloading Real-time DDoS Attack Detection to Programmable Data Planes”. In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2019, pp. 19–27.

[34] Damu Ding, Marco Savi, and Domenico Siracusa. “Estimating Logarithmic and Exponential Functions to Track Network Traffic Entropy in P4”. In: Apr. 2020. DOI: 10.1109/NOMS47738.2020.9110257.

[35] *Netronome SmartNICs*. URL: https://www.netronome.com/products/smartnic/overview/.

[36] *WIDE project packet traces*. URL: https://mawi.wide.ad.jp/mawi/.

[37] José Jair Santanna et al. “Booters — An analysis of DDoS-as-a-service attacks”. In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2015, pp. 243–251. DOI: 10.1109/INM.2015.7140298.

[38] *GRNET Monitoring Tools*. URL: https://mon.grnet.gr.

[39] Vibhaalakshmi Sivaraman et al. “Heavy-Hitter Detection Entirely in the Data Plane”. In: SOSR ’17. Santa Clara, CA, USA: Association for Computing Machinery, 2017, pp. 164–176. ISBN: 9781450349475. DOI: 10.1145/3050220.3063772. URL: https://doi.org/10.1145/3050220.3063772.

[40] *Christos Serafeidis - Diploma Thesis - Github*. URL: https://github.com/chriserafi/diploma-thesis.

[41] Eva Ostertagova and Oskar Ostertag. “Forecasting Using Simple Exponential Smoothing Method”. In: *Acta Electrotechnica et Informatica* 12 (Dec. 2012), pp. 62–66. DOI: 10.2478/v10198-012-0034-2.

[42] *Types of DDoS Attacks*. URL: https://cybersecurity.att.com/blogs/security-essentials/types-of-ddos-attacks-explained.

[43] *Types of DDoS Attacks*. URL: https://www.esecurityplanet.com/networks/types-of-ddos-attacks/.

[44] *Behavioral Model (BMv2) - The reference P4 software switch*. URL: https://github.com/p4lang/behavioral-model.

[45] *P4 Tutorials - Github*. URL: https://github.com/p4lang/tutorials.

[46] *$P4_{16}$ Reference Compiler*. URL: https://github.com/p4lang/p4c.

[47] *BMv2 Simple Swtich CLI*. URL: https://github.com/p4lang/behavioral-model/blob/main/tools/runtime_CLI.py.

[48] *BMv2 Debugger*. URL: https://github.com/p4lang/behavioral-model/blob/main/tools/p4dbg.py.

[49] *Tcpreplay*. URL: https://tcpreplay.appneta.com/.

[50] *Scapy*. URL: https://scapy.readthedocs.io/en/latest/.

[51] *pfsend*. URL: https://www.ntop.org/solutions/wire-speed-traffic-generation.

[52] *Cloudflare mmwatch*. URL: https://github.com/cloudflare/cloudflare-blog/blob/master/2017-06-29-ssdp/mmwatch.