ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

NETMODE - NETWORK MANAGEMENT & OPTIMAL DESIGN LABORATORY

Ανίχνευση Επιθέσεων DDoS σε Προγραμματιζόμενο Επίπεδο Προώθησης Δεδομένων μέσω της Γλώσσας P4

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος-Χρήστος Π. Μητρόπουλος

Επιβλέπων : Βασίλειος Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2019

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

NETMODE - NETWORK MANAGEMENT & OPTIMAL DESIGN LABORATORY

Ανίχνευση Επιθέσεων DDoS σε Προγραμματιζόμενο Επίπεδο Προώθησης Δεδομένων μέσω της Γλώσσας P4

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος-Χρήστος Π. Μητρόπουλος

Επιβλέπων : Βασίλειος Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή επιτροπή την 22η Οκτωβρίου 2019.

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

Αθήνα, Οκτώβριος 2019

.....

Κωνσταντίνος-Χρήστος Μητρόπουλος

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

Περίληψη

Ο προγραμματισμός των συσκευών στο επίπεδο των δεδομένων (Data Plane Programmability) αποτελεί μια νέα τάση στο πλαίσιο των δικτύων Οριζομένων από Λογισμικό (SDN). Μέσω αυτού ορίζονται οι ενέργειες προώθησης των συσκευών, και προσφέρεται περισσότερη ευελιξία σε ένα μεγάλο εύρος λειτουργιών δικτύου, όπως είναι και η ανίχνευση ανωμαλιών. Κάτι τέτοιο είναι ιδιαίτερα κρίσιμο για την έγκαιρη αντιμετώπιση κατανεμημένων επιθέσεων άρνησης παροχής υπηρεσίας (Distributed Denial of Service Attacks - DDoS). Τέτοιες επιθέσεις εξελίσσονται διαρκώς σε επίπεδα κλίμακας και πολυπλοκότητας, καθιστώντας τες ένα από τα δυσκολότερα προβλήματα του Διαδικτύου.

Στο πλαίσιο της παρούσας διπλωματικής εργασίας επιδιώκεται η ανάπτυξη ενός μηχανισμού με στόχο την ανίχνευση πιθανών δικτυακών συμβάντων απευθείας στο επίπεδο δεδομένων μέσω της γλώσσας P4. Πρόκειται για μία γλώσσα ειδικού σκοπού που παρέχει ένα στρώμα αφαίρεσης στον διαχειριστή ώστε να γράφει προγράμματα που περιγράφουν τις λειτουργίες προώθησης, χωρίς να εξαρτάται από την εκάστοτε αρχιτεκτονική κάθε συσκευής.

Πιο συγκεκριμένα, ο προτεινόμενος μηχανισμός διενεργεί στατιστικούς υπολογισμούς για την δικτυακή κίνηση απευθείας στην συσκευή και υπό συνθήκες ειδοποιεί συστήματα χωρίς την παρέμβαση κάποιου εξωτερικού επιπέδου ελέγχου. Ειδικότερα, βασίζεται σε τρεις επιμέρους μετρικές, συνήθεις για την ανίχνευση επιθέσεων DDoS. Οι πρώτες δύο εκτιμούν το σύνολο των μοναδικών δικτυακών ροών (srcIP, dstIP, IP Protocol, srcPort, dstPort) συνολικά και ανά δίκτυο αντίστοιχα, με χρήση πιθανοτικών δομών. Η τρίτη αναλύει την ασυμμετρία της κίνησης, υπολογίζοντας τον λόγο εισερχόμενης προς εξερχόμενης κίνησης για tcp/udp κάθε υποδικτύου. Κάθε μετρική εφόσον παραβιάσει ένα προκαθορισμένο κατώφλι ενεργοποιεί έναν δείκτη. Σε περίπτωση που ενεργοποιηθούν και οι τρεις, αποστέλλονται κατάλληλες ειδοποιήσεις που αντιστοιχούν στην ανίχνευση επίθεσης.

Ο μηχανισμός αυτός θεωρητικά αποτελεί ένα ολοκληρωμένο πρώτο στάδιο ανίχνευσης ανωμαλιών, χωρίς να χρειάζεται κάποιου είδους πρωτόκολλο για την εξαγωγή ροών δεδομένων ή δειγματοληψίας που είναι οι επικρατέστεροι μηχανισμοί συλλογής δεδομένων. Παράλληλα θα μπορούσε να αποτελέσει μέρος ενός συνολικού συστήματος αντιμετώπισης επιθέσεων DDoS.

Λέξεις Κλειδιά: προγραμματισμός επιπέδου δεδομένων, επιθέσεις κατανεμημένης άρνησης υπηρεσιών, ανίχνευση ανωμαλιών δικτυακής κίνησης, P4

Abstract

Data Plane Programmability comprises an emerging tendency within the Software Defined Networking paradigm. It allows for the definition of forwarding actions on the devices, offering more flexibility in a wide aspect of network functions, such as anomaly detection. This is considerably critical for in time handling of Distributed Denial of Service Attacks (DDoS). Suck attacks are constantly evolving both in volume and sophistication, and compose a major threat in the Internet.

The purpose of this diploma thesis is the development of a mechanism, aimed at network traffic anomaly detection in the dataplane with the use of the P4 language. P4 is a domain-specific language that offers an abstraction layer on the way network devices process packets, without any dependence on their specific architecture.

Specifically, the suggested mechanism considers statistical attributes of network traffic and notify, on anomaly detection, external systems without the intervention of the control plane. In particular, it is based on three features, commonly used for DDoS attack detection. The first two provide an estimation of the amount of unique network flows (srcIP, dstIP, IP Protocol, srcPort, dstPort) as whole and per network domain respectively, with the use of probabilistic structures. The third analyzes the traffic asymmetry by computing the ration of incoming and outgoing traffic for tcp/up protocols within subnets. Based on predetermined thresholds each mechanism may raise a flag on threshold violation. In case all three flags are raised, appropriate notifications are sent indicating the existence of ongoing DDoS attacks.

The aforementioned mechanism constitutes a theoretically complete first stage of anomaly detection, with no usage of any kind of protocol for the extraction of data flows, or sampling, which are the dominant data collection mechanisms. At the same time, it could be part of a well rounded DDoS mitigation system.

Keywords: Data Plane Programmability, DDoS, anomaly detection, P4

Ευχαριστίες

Με την παρούσα διπλωματική εργασία ολοκληρώνεται το υπέροχο αυτό κεφάλαιο των προπτυχιακών σπουδών μου στο Εθνικό Μετσόβιο Πολυτεχνείο. Αρχικά θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Βασίλη Μάγκλαρη καθώς με εμπιστεύτηκε και μου έδωσε την δυνατότητα να ασχοληθώ με το συγκεκριμένο θέμα και να αποτελέσω μέλος του εργαστηρίου κατά το διάστημα αυτό. Επιπλέον, θα ήθελα να ευχαριστήσω ιδιαιτέρως τους υποψήφιους διδάκτορες Αδάμ Παυλίδη και Μαρίνο Δημολιάνη, καθώς και τον Δρα Δημήτρη Καλογερά για την συνεργασία, την πολύτιμη καθοδήγηση τους και τις συμβουλές τους καθ' όλη την διάρκεια της διπλωματικής εργασίας. Τέλος ένα μεγάλο ευχαριστώ στην οικογένεια και τους φίλους μου για την συμπαράσταση και την στήριξή τους όλα αυτά τα χρόνια.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κεφάλαιο 1 - Εισαγωγή

1.1 Περιγραφή Προβλήματος

Οι επιθέσεις κατανεμημένης άρνησης παροχής υπηρεσίας (DDoS) αποτελούν ένα από σημαντικότερα προβλήματα που καλούνται να αντιμετωπίσουν οι διαχειριστές δικτύων. Πρόκειται για επιθέσεις που έχουν ως σκοπό να παραλύσουν υπηρεσίες εντός ενός δικτύου, καθιστώντας αυτές απρόσιτες σε νόμιμους χρήστες. Τέτοιες επιθέσεις εξελίσσονται διαρκώς τόσο σε επίπεδο κλίμακας αλλά και πολυπλοκότητας, αξιοποιώντας ευφυείς τεχνικές και αναδυόμενα κενά ασφαλείας, καθιστώντας με αυτόν τον τρόπο δύσκολη την αντιμετώπισή τους.

Στο πλαίσιο αντιμετώπισης αυτού του τύπου επιθέσεων, απαιτείται η ταχύτατη ανίχνευση τους. Οι συμβατικοί μηχανισμοί ανίχνευσης βασίζονται σε τεχνικές δειγματοληψίας ή και αντιγραφής όλων των πακέτων κίνησης (καθρέπτισμα), για να συλλέξουν κίνηση, την οποία και θα στείλουν σε ειδικά συστήματα για την ανάλυσή της. Ωστόσο τέτοιοι μηχανισμοί παρουσιάζουν πολλές φορές αρκετά μεγάλες καθυστερήσεις καθώς και προβλήματα κλιμακωσιμότητας.

Η εφαρμογή των Οριζόμενων από Λογισμικό δικτύων (SDN) έφερε ένα μεγάλο σύνολο νέων λειτουργιών για την διαχείριση και τον έλεγχο των δικτύων. Κεντρικοί ελεγκτές περικλείουν πλέον την λογική της προώθησης της κίνησης και είναι υπεύθυνοι για την ενημέρωση των κανόνων των συσκευών προώθησης, αλλά και για την λήψη στατιστικών από αυτές για την περαιτέρω ανάλυσή τους. Με βάση αυτές τις ενέργειες αναπτύχθηκαν νέοι μηχανισμοί ανίχνευσης επιθέσεων. Αυτοί όμως βασιζόμενοι στην επικοινωνία μεταξύ των ελεγκτών και των συσκευών, έχουν ως αποτέλεσμα την δημιουργία μεγάλου όγκου κίνησης από και προς τους ελεγκτές, οι οποίοι πολλές φορές αδυνατούν να την επεξεργαστούν και να αποφανθούν έγκαιρα.

Η νέα τάση του προγραμματισμού του επιπέδου δεδομένων των δικτυακών συσκευών έρχεται να δώσει ενθαρρυντικές προοπτικές για την ανάπτυξη μηχανισμών ανίχνευσης. Μέσω αυτής είναι δυνατή η τροποποίηση των λειτουργιών προώθησης των δικτυακών συσκευών ώστε να ενσωματωθούν ενέργειες ανάλυσης της κίνησης για την εξαγωγή μετρικών ταυτόχρονα με την προώθηση των πακέτων. Εντούτοις η ανάλυση της κίνησης κατά την προώθησή της, κρύβει τον κίνδυνο της εισαγωγής καθυστερήσεων στην προώθηση μιας και δαπανώνται εξίσου πόροι για την ανάλυση.

Στο πλαίσιο της παρούσας διπλωματικής εργασίας λοιπόν, επιδιώκεται η ανάπτυξη ενός μηχανισμού ανίχνευσης επιθέσεων εξ' ολοκλήρου στο επίπεδο δεδομένων. Ο μηχανισμός αποσκοπεί στον υπολογισμό μετρικών ροών πακέτων και ασυμμετρίας κίνησης, αλλά και στην αποστολή ειδοποιήσεων εφόσον ανιχνευθεί πιθανή επίθεση. Η ανάπτυξή του βασίζεται στην γλώσσα P4, μια γλώσσα ειδικού σκοπού που αποτελεί κύριο εκφραστή του προγραμματισμού του επιπέδου δεδομένων την δεδομένη χρονική στιγμή.

1.2 Δομή Εργασίας

Η παρούσα διπλωματική εργασία αποτελείται από 6 κεφάλαια. Ακολουθεί μια συνοπτική αναφορά στο περιεχόμενο των υπολοίπων κεφαλαίων εργασίας:

- **Κεφάλαιο 2**: Παρουσιάζεται το θεωρητικό υπόβαθρο που αφορά τον προγραμματισμό επιπέδου δεδομένων και τις επιθέσεις DDoS, ενώ γίνεται μια εκτενής ανάλυση των βασικών στοιχείων της γλώσσας P4.
- **Κεφάλαιο 3**: Γίνεται αναλύονται σχετικές εργασίες που αποτέλεσαν πηγή έμπνευσης, και παρουσιάζεται επιγραμματικά ο προτεινόμενος μηχανισμός.
- **Κεφάλαιο 4**: Περιγράφονται με λεπτομέρεια τα στάδια του προτεινόμενου μηχανισμού, ενώ γίνεται αναφορά σε προγραμματιστικά περιβάλλοντα και πρόσθετα εργαλεία που χρησιμοποιήθηκαν.
- **Κεφάλαιο 5**: Αναφέρονται τα προγραμματιστικά περιβάλλοντα και πρόσθετα εργαλεία που χρησιμοποιήθηκαν. Γίνεται ανάλυση της πειραματικής διαδικασίας, αναφέρεται η πειραματική διάταξη και παρουσιάζονται τα αποτελέσματα των πειραμάτων που διεξήχθησαν.
- **Κεφάλαιο 6**: Συνοψίζονται τα συμπεράσματα της παρούσας διπλωματικής και προτείνονται επόμενα βήματα.

Κεφάλαιο 2 - Θεωρητικό υπόβαθρο

2.1 Επίπεδα Λειτουργιών και Αρχιτεκτονική SDN

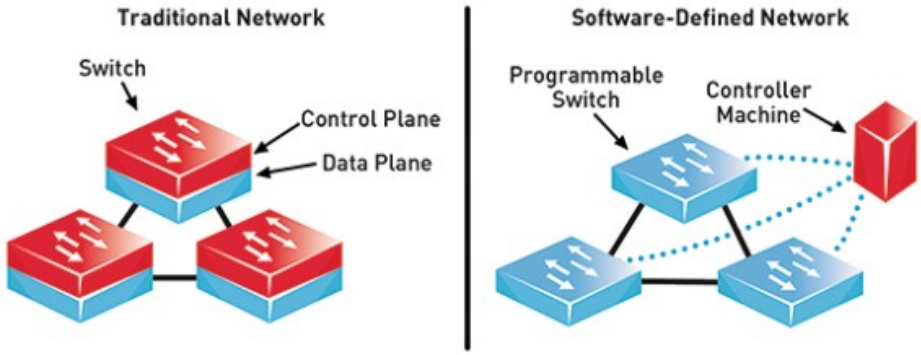
Στο πλαίσιο ενός δικτύου, οι δικτυακές συσκευές μεταφέρουν δεδομένα που παράγονται και αξιοποιούνται από ένα μεγάλο σύνολο εφαρμογών, ώστε να βελτιώνονται διάφοροι τομείς της καθημερινότητας. Οι συσκευές, λοιπόν, οφείλουν να εκτελέσουν λειτουργίες που στοχεύουν για την σωστή και αποδοτική διακίνηση δεδομένων, τις οποίες μπορούμε να τις κατατάξουμε σε τρία επίπεδα:

Επίπεδο Δεδομένων - Data Plane
Το συγκεκριμένο επίπεδο περιλαμβάνει τις λειτουργίες προώθησης των δεδομένων που εισέρχονται στην συσκευή με βάση προκαθορισμένους κανόνες.

Επίπεδο Ελέγχου - Control Plane
Στο επίπεδο ελέγχου εντάσσονται οι λειτουργίες που αφορούν την σηματοδοσία και τον καθορισμό κανόνων με βάση τους οποίους το επίπεδο δεδομένων προωθεί τα μηνύματα που εισέρχονται στην συσκευή.

Επίπεδο Διαχείρισης - Management Plane
Λειτουργίες που αφορούν την διαχείριση των δικτυακών συσκευών και κατ' επέκταση του δικτύου ανήκουν στο επίπεδο διαχείρισης, με κύριο μοντέλο αναφοράς το FCAPS : fault - βλαβών, configuration - διάρθρωσης, accounting - λογιστικής, performance - επιδόσεων, security - ασφαλείας.

Στα παραδοσιακά δίκτυα οι διαδικασίες προώθησης και ελέγχου είναι συνδεδεμένες. Η σηματοδοσία προέρχεται από εισερχόμενα πακέτα στην συσκευή, τα οποία η ίδια επεξεργάζεται με βάση ένα ευρύ σύνολο πρωτοκόλλων και παράγει τους κατάλληλους κανόνες για την προώθηση των δεδομένων. Ωστόσο, όπως αναφέρεται και στο [1], η ανάγκη για κεντρική διαχείριση και παραμετροποίηση των συσκευών δικτύου, η πολυπλοκότητα που δημιουργείται από το πλήθος των πρωτοκόλλων και των συσκευών, και η επιθυμία για προγραμματιζόμενες λειτουργίες που θα καθιστούσαν ευκολότερη την καινοτομία, αποτέλεσαν λόγους ανάπτυξης της αρχιτεκτονικής των Δικτύων Οριζομένων από το Λογισμικό (SDN - Software Defined Networking).



***Εικόνα 2.1**: Αρχιτεκτονική SDN*

Στην αρχιτεκτονική SDN, το επίπεδο ελέγχου διαχωρίζεται πλήρως από το επίπεδο δεδομένων, αποσπάται από τις δικτυακές συσκευές και μεταφέρεται σε συσκευές ελέγχου (controllers - χειριστές). Οι συσκευές ελέγχου, που βασίζονται σε λογισμικό, λειτουργούν ως ένα επίπεδο αφαίρεσης που επιτρέπει τον έλεγχο του δικτύου από ένα κεντρικό σημείο. Με αυτόν τον τρόπο αναλαμβάνουν αυτές πλέον να παρέχουν τους κανόνες προώθησης σε δικτυακές συσκευές όπως μεταγωγείς και δρομολογητές. Ταυτόχρονα παρέχουν δυνατότητες προγραμματισμού των λειτουργιών τους, διευκολύνοντας έτσι την διαχείριση του δικτύου.

Μια από τις πλέον διαδεδομένες υλοποιήσεις SDN δικτύων είναι το πρωτόκολλο Openflow [2], το οποίο αποτελεί ένα πρότυπο επικοινωνίας μεταξύ ενός κεντρικού ελεγκτή και μεταγωγών Openflow. Το Openflow βασίζεται στην έννοια των ροών διαδικτυακής κίνησης, δηλαδή στην ομαδοποίηση των πακέτων δεδομένων, σύμφωνα με ένα σύνολο κανόνων που βασίζονται σε θύρες εισόδου, πεδία των επικεφαλίδων και αποτελέσματα επεξεργασίας των πακέτων, για να κάνουν την αντιστοίχιση σε ροές. Οι μεταγωγείς περιέχουν πίνακες κανόνων (Flow tables), στους οποίους ο χειριστής εισάγει, διαγράφει ή τροποποιεί εγγραφές. Οι εγγραφές αυτές περιγράφουν συγκεκριμένες ροές και περιλαμβάνουν ενέργειες (instructions), από ένα σύνολο δυνατών ενεργειών που ορίζεται από το πρωτόκολλο και υλοποιείται από την συσκευή, οι οποίες θα εκτελεστούν στο πακέτο, εφόσον αυτό αντιστοιχεί στην συγκεκριμένη εγγραφή. Μέσω αυτών των κανόνων και ενεργειών θα αποφασιστεί τελικά αν και προς τα που θα προωθηθεί το κάθε πακέτο. Ορίζοντας, λοιπόν, και τροποποιώντας δυναμικά αυτούς τους πίνακες, ο χειριστής επηρεάζει την λειτουργία προώθησης των μεταγωγών και κατ' επέκταση καθορίζει τα μονοπάτια κίνησης των πακέτων μέσα στο δίκτυο.

2.2 Προγραμματισμός Επιπέδου Δεδομένων

Πρωτόκολλα, ωστόσο, όπως το OpenFlow, λειτουργούν κατά κύριο λόγω πάνω σε συσκευές με προκαθορισμένες λειτουργίες, δηλαδή τις λεγόμενες "fixed-functions" συσκευές. Για να καταλάβουμε τον όρο "fixed-functions" αξίζει να πάρουμε μια ιδέα από το πως μια συσκευή επεξεργάζεται εισερχόμενα πακέτα στα πλαίσια αυτών των πρωτοκόλλων. Αρχικά η συσκευή διαβάζει ένα σύνολο επικεφαλίδων από το πακέτο, διαδικασία γνωστή ως "parsing". Στην συνέχεια, έχοντας εξάγει τα απαραίτητα δεδομένα από τις επικεφαλίδες, ελέγχει όπως

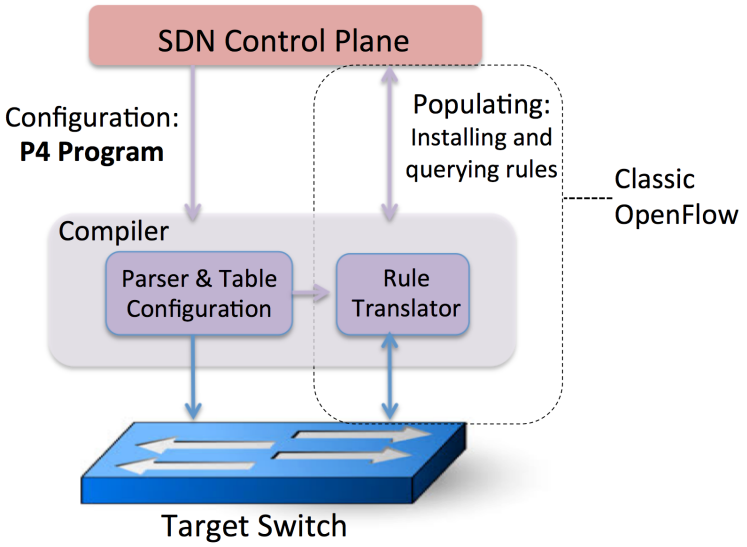
αναφέραμε παραπάνω ένα σύνολο πινάκων και εκτελεί τις λειτουργίες που αντιστοιχούν είτε σε κάποιον υπολογισμό, είτε σε τροποποίηση των επικεφαλίδων του πακέτου, είτε στην ενημέρωση τιμών που σχετίζονται με την εποπτεία του δικτύου. Στο τέλος, ανάλογα με τα αποτελέσματα των λειτουργιών, το πακέτο εξέρχεται από κάποια θύρα της συσκευής ή απορρίπτεται.

Στο πλαίσιο, λοιπόν, των "fixed-function" συσκευών, τόσο οι επικεφαλίδες που μπορούν να διαβαστούν και να χρησιμοποιηθούν σαν δεδομένα για την αντιστοίχιση πακέτου με κανόνες, και συνεπώς τα δικτυακά πρωτόκολλα που είναι κατανοητά από την συσκευή, όσο και οι λειτουργίες που μπορούν να αποτελέσουν αποτέλεσμα αντιστοίχισης με κανόνες, είναι προκαθορισμένες από την κατασκευή της συσκευής. Υπό αυτήν την οπτική, οποιαδήποτε καινούρια απαίτηση για υποστήριξη νέων πρωτοκόλλων και λειτουργιών, θα απαιτούσε χρονοβόρες εργασίες και δοκιμές από τους κατασκευαστές ώστε τελικά να παρέχουν τις επιθυμητές βελτιώσεις, δυσκολεύοντας με αυτόν τον τρόπο τις προσπάθειες διαχειριστών και ερευνητών για ανάπτυξη και καινοτομία πάνω σε δικτυακές υπηρεσίες. Χαρακτηριστικό παράδειγμα είναι οι επεκτάσεις του πρωτοκόλλου OpenFlow 1.x.. Καθώς στις επεκτάσεις υπήρχαν εισαγωγές καινούργιων πρωτοκόλλων που θα όφειλε να υποστηρίζει ένας μεταγωγέας, οι διαχειριστές δεν μπορούσαν άμεσα να τροποποιήσουν τις λειτουργίες της συσκευής, αλλά ο κατασκευαστής έπρεπε να προσθέσει τις ζητούμενες αλλαγές [3].

Εδώ εμφανίζεται η τάση του προγραμματισμού του επιπέδου δεδομένων (data plane programming) των δικτυακών συσκευών [4]. Σύμφωνα με αυτήν, ο προγραμματισμός του επιπέδου δεδομένων έγκειται στην δυνατότητα του ελεγκτή να επαναπροσδιορίζει γρήγορα και κατά βούληση τις μεθόδους ανάλυσης επικεφαλίδων (parsing) και επεξεργασίας (processing) των πακέτων. Ο διαχειριστής θα έχει, μέσω του ελεγκτή, την δυνατότητα να τροποποιεί το σύνολο των αποδεκτών επικεφαλίδων, να επιτρέπει την αντιστοίχιση κανόνων με βάση αυθαίρετες επικεφαλίδες, να εισάγει νέες και να τροποποιεί υπάρχουσες λειτουργίες επεξεργασίας των πακέτων. Με αυτόν τον τρόπο, θα μπορεί να δοκιμάσει, αναπτύξει και εφαρμόσει, ευκολότερα και γρηγορότερα, νέες τεχνικές και να τις εισάγει άμεσα στο δίκτυο κατά την λειτουργία του, μιας και πλέον δεν εξαρτάται από τον κατασκευαστή και τις λειτουργίες που αυτός παρέχει.

2.3 Γλώσσα P4

Για τον σκοπό του προγραμματισμού των λειτουργιών προώθησης των δικτυακών συσκευών αναπτύχθηκε η γλώσσα ειδικού σκοπού (domain specific language) P4. Η γλώσσα P4 παίρνει το όνομά της από τα αρχικά των λέξεων Programming Protocol-independent Packet Processors [3] (Προγραμματισμός, ανεξάρτητων από πρωτόκολλα, επεξεργαστών πακέτων) Αποτελεί μια γλώσσα υψηλού επιπέδου που στοχεύει στην περιγραφή των λειτουργιών του επιπέδου δεδομένων των προγραμματιζόμενων δικτυακών συσκευών, οι οποίες σε όρους P4 ονομάζονται συσκευές-στόχοι (targets).



Εικόνα 2.2: *Το P4 σε αντίθεση με το OpenFlow [3]*

Σύμφωνα με τους δημιουργούς της, η γλώσσα P4 αναπτύχθηκε με τρεις κύριους στόχους:

- Δυνατότητα τροποποίησης (Reconfigurability). Ο χειριστής θα μπορεί ενεργά να τροποποιήσει την λειτουργία ανάλυσης και επεξεργασίας πακέτων στις συσκευές στόχους.
- Ανεξαρτησία από πρωτόκολλα (Protocol Independence). Η λειτουργία της συσκευής δεν θα πρέπει να βασίζεται και να δεσμεύεται από την χρήση συγκεκριμένων δικτυακών πρωτοκόλλων.
- Ανεξαρτησία από συσκευές-στόχους (Target Independence). Ο χειριστής και ο κώδικας του προγράμματος δεν χρειάζεται να γνωρίζουν λεπτομέρειες σχετικές με την κατασκευή της συσκευής για να την προγραμματίσουν.

Ο πρώτος στόχος ικανοποιείται, μιας και οι συσκευές που υποστηρίζουν P4 μπορούν να επαναπρογραμματιστούν άμεσα, χωρίς ανάγκη μεσολάβησης του κατασκευαστή. Ο δεύτερος στόχος ικανοποιείται από το γεγονός πως μέσω των προγραμμάτων P4 μπορούμε να ορίσουμε αυθαίρετα επικεφαλίδες και πρωτόκολλα με βάση τα οποία θα γίνεται η λειτουργία της προώθησης. Ο τρίτος έγκειται στην υποστήριξη του P4 από τους κατασκευαστές. Η γλώσσα P4 ορίζει δομές που εκφράζουν βασικές λειτουργίες που θα πρέπει να εκτελεί η συσκευή, χωρίς ωστόσο να ορίζει πως αυτές οι λειτουργίες θα υλοποιούνται από την κάθε συσκευή. Έτσι εφόσον ένας κατασκευαστής θέλει οι συσκευές του να υποστηρίζουν την γλώσσα P4, τότε θα πρέπει να παρέχει κατάλληλους μεταγλωττιστές για την κάθε συσκευή, οι οποίοι θα μετατρέπουν τα υψηλού επιπέδου προγράμματα P4 σε χαμηλού επιπέδου εντολές κατάλληλες για τον εκάστοτε συσκευή. Με αυτόν τον τρόπο ένα αποδεκτό πρόγραμμα P4 θα μπορεί να εκτελεστεί για παράδειγμα τόσο σε έναν εικονικό μεταγωγέα όσο και σε μια φυσική κάρτα δικτύου, εφόσον υποστηρίζουν και οι δύο το P4.

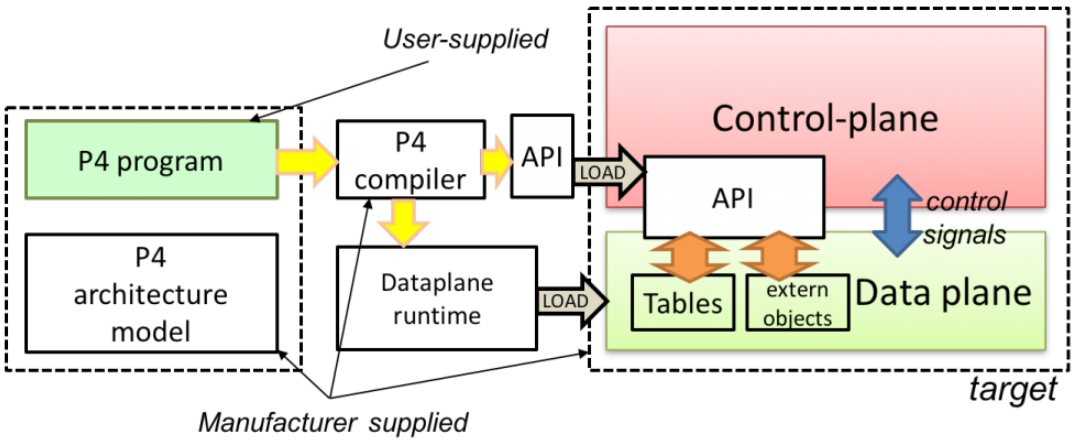
Βέβαια θα πρέπει να τονίσουμε ότι προγράμματα γραμμένα σε P4 δεν είναι πάντα μεταφέρσιμα από συσκευή σε συσκευή. Όπως θα δούμε παρακάτω, η γλώσσα P4 επιτρέπει

σε κατασκευαστές να παρέχουν επιπλέον λειτουργίες, εκτός από το σύνολο που αυτή περιγράφει, και επομένως αυτές είναι πιθανό να μην υλοποιούνται από άλλες συσκευές. Επίσης μπορεί η συσκευή να υποστηρίζει μέρος των λειτουργιών που περιγράφονται από το P4, ενώ ακόμα μπορεί το πρόγραμμα να ζητάει πόρους (π.χ. μνήμη) που κάποια συσκευή δεν μπορεί να παρέχει. Ως αποτέλεσμα, ενώ η γλώσσα σχεδιάστηκε με στόχο την μεταφερσιμότητα των προγραμμάτων της, εντούτοις είναι αρκετά πιθανό προγράμματα που γράφτηκαν για μια συσκευή, να μην μπορούν να εκτελεστούν σε άλλες ή και να εμφανίζουν διαφορές στην απόδοσή τους.

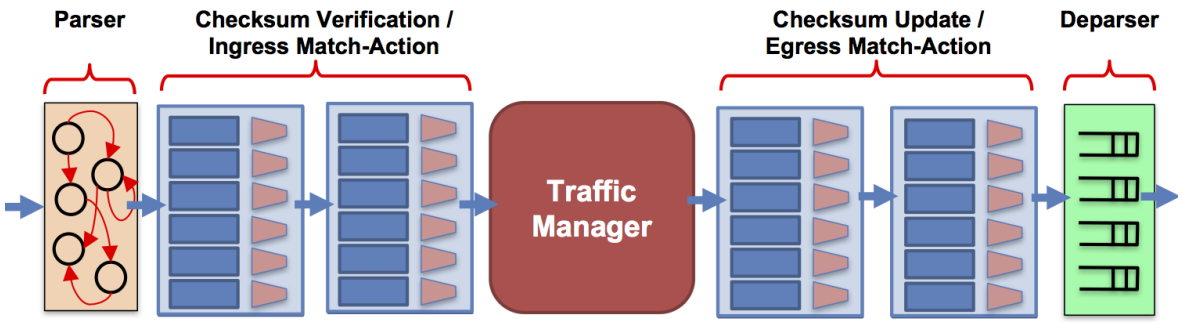
Την συγκεκριμένη στιγμή υπάρχουν δύο εκδόσεις της γλώσσας P4: η έκδοση $P4_{14}$ [5] και η νεότερη έκδοση $P4_{16}$ [6]. Η έκδοση $P4_{16}$ αλλάζει ως ένα βαθμό το λεξιλόγιο και την σύνταξη της γλώσσας, εμπλουτίζοντας τις λειτουργίες και την λογική της. Στην συνέχεια θα παρουσιάσουμε τα βασικά στοιχεία της γλώσσας P4 μέσω της $P4_{16}$, τα περισσότερα από τα οποία στην λογική τους είναι όμοια και στις δύο εκδόσεις αλλά όχι στην υλοποίησή τους. Επιπλέον θα αναφέρουμε και κάποιες βασικές διαφορές μεταξύ των δύο. Από εδώ και πέρα όταν αναφερόμαστε στην γλώσσα P4 θα εννοούμε την έκδοση $P4_{16}$, εκτός και αν αναφέρουμε συγκεκριμένες εκδόσεις.

2.3.1 Αρχιτεκτονική συσκευής - Βιβλιοθήκες

Για τον προγραμματισμό των συσκευών απαιτείται από τον κατασκευαστή, εκτός του μεταγλωττιστή, και ο ορισμός σε P4 του αρχιτεκτονικού μοντέλου της συσκευής. Πρόκειται για κώδικα σε P4 ο οποίος περιγράφει το σύνολο των κομματιών της συσκευής που μπορούν να προγραμματιστούν, όπως ο parser και και το κομμάτι ελέγχου εισόδου (ingress control block) που θα αναφέρουμε στην συνέχεια. Επιπλέον σε αυτό το αρχείο υπάρχουν πιθανώς και πρόσθετες λειτουργίες που ο κατασκευαστής παρέχει στον προγραμματιστή.



***Εικόνα 2.3**: Προγραμματισμός συσκευής με το P4 [6]*



Εικόνα 2.4: *Βασική αρχιτεκτονική (v1model) του P4*

Μια βασική αρχιτεκτονική που χρησιμοποιείται, εκφράζεται από την παραπάνω εικόνα. Στην αρχή έχουμε το κομμάτι του parser, που ακολουθείται από το block ελέγχου εισόδου. Όπως θα εξηγήσουμε και στην συνέχεια, στα control blocks γίνονται οι διάφορες λειτουργίες όπως αντιστοίχισης και υπολογισμού. Στην συνέχεια παρεμβάλλεται ένα σύστημα διαχείρισης της κίνησης (ουρές, buffers, χρονοπρογραμματισμός πακέτων), το οποίο βρίσκεται έξω από την σκοπιά του P4. Τέλος έχουμε το control block εξόδου , και τον deparser στον οποίο ορίζουμε με ποιες επικεφαλίδες θα βγει από την συσκευή το πακέτο.

Εκτός από το αρχείο αρχιτεκτονικής, πρόσθετος κώδικας P4 μπορεί να υπάρχει και σε αρχεία βιβλιοθήκες (libraries). Οι βιβλιοθήκες εισάγονται στον κώδικα με την χρήση της δεσμευμένης λέξης **#include**. Το ίδιο το P4 ορίζει την βασική βιβλιοθήκη **core.p4** στην οποία υπάρχουν οι ορισμοί packet_in , packet_out που αποτελούν αναφορές στο εισερχόμενο και εξερχόμενο πακέτο αντίστοιχα.

2.3.2 Βασικοί Τύποι

Η γλώσσα $P4_{16}$ αποτελείται από ένα σύνολο βασικών τύπων με τους οποίους ορίζονται πιο σύνθετες δομές. Οι κυριότεροι είναι οι εξής:

- **bit<N>**: πρόκειται για μη προσημασμένο ακέραιο μεγέθους N bits
- **int<N>**: πρόκειται για προσημασμένο ακέραιο μεγέθους N bits
- **varbit<N>**: πρόκειται για ακολουθία μεταβλητού μεγέθους σε bits με όριο τα N bits
- **bool**: λογικός τύπος με δυνατές τιμές true , false
- **match_kind**: ειδικός τύπος που περιέχει μεθόδους αντιστοίχισης (exact, ternary, lpm) και χρησιμοποιείται στους πίνακες

Πάνω στους τύπους **bit<N>** και **int<N>** ορίζονται βασικές αριθμητικές πράξεις (εκτός από διαίρεση), λογικές πράξεις και συγκρίσεις. Οι τύποι **varbit<N>** χρησιμοποιούνται κυρίως για πεδία επικεφαλίδων μεταβλητού μήκους (π.χ. IPv4 options) και επιτρέπονται μόνο οι έλεγχοι για ισότητα και ανισότητα. Υποστηρίζονται μετατροπές τύπων (casts) μεταξύ **int - bit**, και **bit<1> - bool**, καθώς και αλλαγή των μεγεθών των αριθμών **bit<M> - bit<N>** , **int<M> - int<N>**. Οι μετατροπές δεν γίνονται αυτόματα και θα πρέπει να αναφέρονται στο πρόγραμμα.

2.3.3 Header - Struct

Ο τύπος header αποτελεί δομή στην οποία αποθηκεύουμε τις επικεφαλίδες που διαβάζουμε από τα πακέτα. Ο τύπος struct χρησιμεύει κατά κύριο λόγο στην αποθήκευση μεταδεδομένων, χρήσιμων κατά την ανάλυση και επεξεργασία των πακέτων. Και οι δύο αυτές δομές είναι παρόμοιες, με την μόνη διαφορά να είναι πως ο τύπος header περιλαμβάνει ένα κρυφό bit, το οποίο ονομάζεται validity bit, και χρησιμοποιείται για τον έλεγχο εγκυρότητας του header. Τίθεται στο 1 κατά το parsing και μπορεί να προσπελαστεί με τις συναρτήσεις :

- **isValid()** : επιστρέφει **true** αν **validity bit = 1,** αλλιώς **false**
- **setValid()**, **setInvalid()**: αναθέτουν τιμές 1 και 0 αντίστοιχα στο **validity bit**

Πίνακας 2.1: Ορισμος επικεφαλίδας Ethernet και αυθαίρετων μεταδεδομένων

header ethernet_t {
  bit<32> dstAddr;
  bit<32> srcAddr;
  bit<16> etherType;
}
	struct metadata {
	  bit<32> field1;
	  bit<16> field2;
	  bit<8> field3;
	}

Δομές μεταδεδομένων μπορεί να ορίσει τόσο ο χρήστης στο πρόγραμμά του, όσο και ο κατασκευαστής στο αρχιτεκτονικό μοντέλο της συσκευής ή σε κάποια βιβλιοθήκη. Συνήθως παρέχονται από την αρχιτεκτονική δομές μεταδεδομένα που αφορούν την πόρτα εισόδου , την πόρτα εξόδου, το μέγεθος του πακέτου, τον χρόνο εισόδου και άλλα.

2.3.4 Parser

Ο Parser όπως έχουμε αναφέρει ορίζει τις πιθανές ακολουθίες με τις οποίες θα αναλυθούν οι επικεφαλίδες των πακέτων και πρακτικά περιγράφει μια μηχανή πεπερασμένων καταστάσεων (finite state machine). Στις καταστάσεις εξάγονται επικεφαλίδες από τα πακέτα και αποθηκεύονται σε δομές header μέσω της εντολής **extract()**, ενώ ορίζονται μεταβάσεις σε άλλες καταστάσεις μέσω της εντολής **transition**. Μπορούν να οριστούν μεταβάσεις υπό συνθήκη, με την βοήθεια της εντολής **select()** η οποία παίρνει σαν όρισμα το πεδίο του οποίου θα ελέγξουμε την τιμή, και ορίζει ένα σύνολο από πιθανές μεταβάσεις. Η αρχική κατάσταση ονομάζεται **start**, ενώ οι τελικές καταστάσεις είναι οι **accept** και **reject**, οι οποίες περιγράφουν επιτυχές ή ανεπιτυχές parsing αντίστοιχα. Όταν το parsing τελειώσει επιτυχώς, δηλαδή φτάσουμε στην κατάσταση **accept**, αρχίζει το κομμάτι του ingress control block.

Στον παρακάτω πίνακα φαίνεται ο ορισμός ενός parser για το διάβασμα επικεφαλίδων ethernet και IPv4. Κατά τον ορισμό του parser θα πρέπει να ορίσουμε σαν παραμέτρους την αναφορά στο πακέτο εισόδου, καθώς και τις δομές για τις επικεφαλίδες. Επιπλέον, μπορούμε να περάσουμε σαν παραμέτρους δομές μεταδεδομένων που πιθανώς θα χρειαστούμε ή θα θέλουμε να τροποποιήσουμε κατά το parsing.

Πίνακας 2.2: Ορισμός parser για Ethernet και IPv4 [7]

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }

}
```

2.3.5 Control Blocks

Στα κομμάτια ελέγχου (control blocks) γίνεται η επεξεργασία των επικεφαλίδων των πακέτων που διαβάστηκαν από τον parser, το σύνολο των επιθυμητών υπολογισμών για την εξαγωγή στατιστικών, και πρακτικά η υλοποίηση των επιθυμητών λειτουργιών προώθησης της συσκευής. Στο εσωτερικό του μπορούν να υπάρχουν αυτούσια κομμάτια κώδικα, κλήσεις συναρτήσεων και δομές αντιστοίχισης (match-action units). Στο πλαίσιο του P4, οι συναρτήσεις ονομάζονται **actions** και οι δομές αντιστοίχισης **tables**.

Πίνακας 2.3: Ορισμός control block [7]

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
                  ....
}
```

Κατά τον ορισμό του control block οφείλουμε να αναφέρουμε σαν παραμέτρους που θα μπορούμε να προσπελάσουμε, τόσο τις δομές επικεφαλίδων που διαβάσαμε από το πακέτο, όσο και δομές μεταδεδομένων που θα χρειαστούμε. Μέσα στο control block μπορούμε να

ορίσουμε αρχικά actions και tables που θα χρειαστούμε κατά την επεξεργασία. Στην συνέχεια υπάρχει το **apply** block. Στο μπλοκ αυτό γράφεται το σύνολο του κώδικα που θα εκτελείται για την προώθηση κάθε νέου πακέτου, ενώ υποστηρίζεται η χρήση **if-else** συνθηκών, όχι όμως και δομών επανάληψης όπως **for** και **while** loops.

Τα control blocks εισόδου (ingress) και εξόδου (egress) δεν διαφέρουν σχεδόν καθόλου στην λειτουργία τους. Ωστόσο θα πρέπει στο ingress control block να ορίσουμε την επιθυμητή πόρτα εξόδου του πακέτου, την οποία δεν μπορούμε να αλλάξουμε στο egress control block.

2.3.6 Tables - Actions

Τα tables περιγράφουν δομές αντιστοίχισης. Όταν ορίζουμε ένα table θα πρέπει να ορίσουμε και βασικές ιδιότητές του (properties):

- **key**: πρόκειται για μια λίστα με εγγραφές της μορφής **field: match_kind** που ορίζουν ποιο πεδίο (field) θα πρέπει να αντιστοιχιστεί και με ποια μορφή αντιστοίχισης (match_kind).
- **actions**: πρόκειται για μια λίστα με όλες τις δυνατές συναρτήσεις που μπορούν να εκτελεστούν ως αποτέλεσμα αντιστοίχισης με κάποιον κανόνα.

Μπορούμε να ορίσουμε επιπλέον και δύο προαιρετικές ιδιότητες:

- **default_action**: εισάγεται μετά την ιδιότητα actions και ορίζει την συνάρτηση που θα εκτελεστεί αν δεν υπάρξει αντιστοίχιση με κάποιον κανόνα.
- **size**: ορίζει το μέγιστο πλήθος κανόνων που μπορεί να έχει ο πίνακας.

Πίνακας 2.4: Ορισμός και κλήση table [7]

```
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = drop();
}
```

```
apply {
    if (hdr.ipv4.isValid()) {
        ipv4_lpm.apply();
    }
}
```

Οι κανόνες εισάγονται και τροποποιούνται κατά κύριο λόγο από το επίπεδο ελέγχου και δεν μπορεί το ίδιο το πρόγραμμα να τροποποιήσει δυναμικά το σύνολο των κανόνων σε έναν πίνακα. Στους κανόνες δίνονται οι τιμές σύγκρισης, μάσκες (masks) που πιθανώς θα χρειαστούν, και το όνομα της συνάρτησης που θα εκτελεστεί από τον συγκεκριμένο κανόνα. Από την βιβλιοθήκη **core.p4** παρέχονται τρεις μορφές αντιστοίχισης:

- **exact**: Η τιμή του πεδίου με την τιμή του κανόνα θα πρέπει να ταυτίζονται.

- **ternary**: Τα bits της τιμής του πεδίου θα πρέπει να ταυτίζονται με τα αντίστοιχα του κανόνα. Ο ορισμός των bits γίνεται με βάση μια μάσκα που παρέχει ο κανόνας.
- **lpm**: Longest Prefix Matching.

Για να χρησιμοποιήσουμε κάποιο table στο control block θα πρέπει να καλέσουμε την μέθοδό του **apply()**. Η κλήση αυτής της μεθόδου επιστρέφει μια δομή με δύο τιμές:
- **hit**: **true** αν κάποιος υπήρξε αντιστοίχιση με κάποιον κανόνα, αλλιώς **false**.
- **action_run**: επιστρέφει την συνάρτηση που εκτελέστηκε.

Τα actions, όπως είπαμε, αποτελούν συναρτήσεις και μπορούν να εκτελεστούν είτε σαν αποτέλεσμα αντιστοίχισης σε κάποιο table, είτε να κληθούν αυτόνομα στο εσωτερικό του **apply** block ενός control block.

Πίνακας 2.5: Ορισμός actions [7]

```
action drop() {
   mark_to_drop();
}

action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
   standard_metadata.egress_spec = port;
   hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
   hdr.ethernet.dstAddr = dstAddr;
   hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}
```

2.3.7 Deparser

Ο deparser εκτελεί την αντίστροφη διαδικασία από τον parser, δηλαδή ορίζει τις επικεφαλίδες που θα πρέπει να προστεθούν στο πακέτο πριν εξέλθει από την συσκευή. Ο ορισμός του deparser είναι όμοιος με αυτόν ενός control block, μόνο που θα πρέπει σαν όρισμα να υπάρχει και ο τύπος **packet_out** δηλαδή αναφορά στο εξερχόμενο πακέτο. Με την εντολή **emit()**, η οποία παίρνει σαν όρισμα τις επιθυμητές επικεφαλίδες, γίνεται η προσθήκη τους στο πακέτο, η οποία θα πρέπει να πραγματοποιηθεί με την σειρά που εξήχθησαν κατά την διαδικασία του parsing. Επιπλέον για να προστεθεί μια επικεφαλίδα στο πακέτο θα πρέπει να είναι έγκυρη, δηλαδή το validity bit να είναι ίσο με 1.

Πίνακας 2.6: Ορισμός deparser [7]

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

2.3.8 Package

Ο τύπος package είναι αυτός με τον οποίο ο κατασκευαστής ορίζει προγραμματιστικά τις δομές που εκφράζουν το αρχιτεκτονικό μοντέλο της συσκευής (pipeline), και αποτελεί διεπαφή με την οποία μπορούμε να συνδέσουμε τα δομικά κομμάτια που έχουμε δημιουργήσει (parsers, control blocks, deparsers) με την συσκευή, ώστε να ξεκινήσει η εκτέλεση των λειτουργιών.

Πίνακας 2.7: Σύνδεση προγραμματισμένων κομματιών με switch [7]
V1Switch(MyParser(), MyVerifyChecksum(), MyIngress(), MyEgress(), MyComputeChecksum(), MyDeparser()) main;

2.3.9 Σημαντικές διαφορές $P4_{14}$ - $P4_{16}$

- Counters, Meters, Registers: Πρόκειται για μετρητές και καταχωρητές, που αποτελούν δομές μνήμης η οποίες δεν είναι ξεχωριστές ανά πακέτο, αλλά κρατάνε μια συνολική κατάσταση της λειτουργίας προώθησης (stateful structs). Στην παλαιότερη έκδοση $P4_{14}$ αυτές οι δομές αποτελούσαν μέρος των προδιαγραφών της γλώσσας, ενώ στην έκδοση $P4_{16}$ θεωρούνται δομές τις οποίες κρίνει ο κατασκευαστής αν θα παρέχει η όχι (extern δομές).

- Digests (Ειδοποιήσεις): Τα digests είναι μηνύματα ειδοποιήσεων, τα οποία μπορεί να στείλει η συσκευή στον χειριστή στο επίπεδο ελέγχου. Ομοίως με τους καταχωρητές, δεν συμπεριλαμβάνεται ως βασικό στοιχείο της γλώσσας στην νεότερη έκδοση.

- Πράξεις και Τελεστές: Στην έκδοση $P4_{14}$ δεν υπάρχουν τελεστές αριθμητικών πράξεων, οι οποίες περιορίζονται, επιπλέον, μόνο στην πρόσθεση και στην αφαίρεση μέσω της κλήσης αντίστοιχων συναρτήσεων (add, subtract).

- Εκτέλεση Κώδικα και Μεταβλητές: Στην έκδοση $P4_{14}$ δεν υπάρχει η δυνατότητα να εκτελείται αυθαίρετος κώδικας μέσα σε control block που δεν αποτελεί μέρος action. Δηλαδή πρέπει όλη η λειτουργία που θέλουμε να εκτελεστεί να βρίσκεται μέσα σε συναρτήσεις, και το control block αυτό που κάνει είναι να καλεί tables και actions. Επιπλέον δεν επιτρέπεται να υπάρχει if-else δομή μέσα σε action, πράγμα που στο $P4_{16}$ δεν απαγορεύεται (αν και αρκετές συσκευές δεν το υλοποιούν ακόμα). Τέλος δεν υπάρχουν μεταβλητές στο $P4_{14}$, τον ρόλο των οποίων παίζουν οι δομές μεταδεδομένων (metadata), οι οποίες είναι σαφώς ορισμένες στο $P4_{14}$.

2.4 Network Monitoring

Ο λόγος για τον οποίο μας ενδιαφέρουν οι παραπάνω τεχνολογίες θα γίνει ευκολότερα κατανοητός αν εξετάσουμε και τον όρο Network monitoring. Με τον όρο αυτό αναφερόμαστε σε ένα σύνολο ενεργειών διαχείρισης και εποπτείας του δικτύου με διάφορους σκοπούς όπως:

- την διαπίστωση σωστής λειτουργίας,
- την επιδιόρθωση βλαβών,
- την διατήρηση της ασφάλειας,
- την ανάλυση του δικτύου για περαιτέρω εξέλιξή του.

Μερικές τεχνικές με της οποίες μπορούμε να πετύχουμε εποπτεία πάνω στο δίκτυο είναι:

- ανάλυση της δικτυακής κίνησης
- έλεγχος πόρων και στατιστικών των διαφόρων συσκευών
- έλεγχος των τελικών χρηστών που συνδέονται στο δίκτυο

Θα σταθούμε κυρίως στο κομμάτι της ανάλυσης κίνησης. Μελετώντας την κίνηση του δικτύου του, ο διαχειριστής μπορεί να εξάγει στατιστικά (όπως ώρες αιχμής, κύριους αποδέκτες κίνησης) για την κίνηση ώστε να τα χρησιμοποιήσει ως βάση για πρόσθετες εργασίες. Μπορεί επιπλέον να χρησιμοποιήσει τεχνικές ανάλυσης κίνησης ως μέθοδο ανίχνευσης ανωμαλιών και πιθανών επιθέσεων που έχουν ως στόχο είτε τελικούς χρήστες του δικτύου, είτε το ίδιο το δίκτυο.

Δύο από τις κυριότερες τεχνικές εξαγωγής δεδομένων από το δίκτυο για την ανάλυσή τους αποτελούν η δειγματοληψία (sampling) και το καθρέφτισμα (mirroring). Η πρώτη τεχνική αντιγράφει ένα ποσοστό πακέτων (1 ανά ν πακέτα) και τα προωθεί σε μια μονάδα κατάλληλη για την επεξεργασία τους, ενώ ο δεύτερος μηχανισμός αντιγράφει όλη την κίνηση για να την στείλει σε αντίστοιχη μονάδα. Ωστόσο, από την μία, με την δειγματοληψία υπάρχει πιθανότητα να μην έχουμε συνολική εικόνα της κίνησης καθώς δεν ελέγχεται ένα σημαντικό μέρος των πακέτων, από την άλλη η τεχνική του mirroring απαιτεί αρκετά αυξημένους δικτυακούς πόρους και υπολογιστική ισχύ, ειδικότερα όταν αφορά μεγάλες ροές κίνησης. Με την ανάπτυξη των SDN δικτύων, πρωτόκολλα όπως το OpenFlow έδωσαν την δυνατότητα στις δικτυακές συσκευές να συλλέγουν πιο σύνθετα δεδομένα [8] (πχ. δεδομένα με βάση ροές κίνησης), ωστόσο και αυτή η μέθοδος μπορεί να αποδειχθεί απαιτητική, καθώς πλέον οι ελεγκτές είναι υπεύθυνοι και για την συλλογή στατιστικών, εκτός από την παροχή οδηγιών προώθησης.

Με την εμφάνιση των προγραμματιζόμενων συσκευών δημιουργήθηκε πεδίο για δοκιμές και πειραματισμούς υπό νέες συνθήκες. Συγκεκριμένα η δυνατότητα προγραμματισμού των λειτουργιών προώθησης δίνει την δυνατότητα για την εφαρμογή αλγορίθμων ανάλυσης κίνησης ταυτόχρονα με την προώθησή της. Έτσι δεν χρειάζεται να αντιγραφεί η κίνηση φορτώνοντας ακόμα περισσότερο τις συνδέσεις και απαιτώντας ειδικούς μηχανισμούς, και επίσης δεν χάνεται μέρος κίνησης λόγω δειγματοληψίας. Βέβαια υπάρχει ένα καινούριο αντιστάθμισμα που θα πρέπει να μελετηθεί υπό αυτές τις συνθήκες, το οποίο είναι η ταχύτητα προώθησης των πακέτων. Είναι λογικό από την στιγμή που εισάγεται επιπλέον λειτουργία στις διαδικασίες προώθησης, να παρατηρηθεί μείωση στην απόδοση των συσκευών σχετικά με την ταχύτητά τους. Πλέον, λοιπόν, η ερευνητική κοινότητα προσπαθεί να διαπιστώσει ποια πρακτικά είναι η χρυσή τομή ανάμεσα στην πολυπλοκότητα των

αλγορίθμων ανάλυσης που μπορούν να εφαρμοστούν κατά την προώθηση των δεδομένων και στην μείωση της ταχύτητας προώθησης που αυτοί θα επιφέρουν.

2.5 Επιθέσεις DDoS

Όπως αναφέραμε παραπάνω, οι τεχνικές ανάλυσης της δικτυακής κίνησης συμβάλουν και στην διατήρηση της ασφάλειας του δικτύου, Υπό αυτό το πρίσμα, τέτοιες τεχνικές επιδιώκουν την ανίχνευση κακόβουλης κίνησης, κίνησης δηλαδή που αποτελεί μέρος επίθεσης και έχει ως στόχο να πλήξει πληροφοριακά συστήματα, τελικούς χρήστες, εξυπηρετητές, δίκτυα, για να προκαλέσει προβλήματα στην λειτουργία και στην απόδοσή τους. Υπάρχουν διάφορες κατηγορίες επιθέσεων που αποσκοπούν στο να βλάψουν υπηρεσίες στο διαδίκτυο και αξιοποιούν ένα ευρύ σύνολο μεθόδων και τεχνικών ώστε να πετύχουν τον σκοπό τους. Στην παρούσα διπλωματική θα δώσουμε έμφαση σε μία κατηγορία από αυτές, στις λεγόμενες επιθέσεις κατανεμημένης άρνησης υπηρεσιών, ή αλλιώς DDoS (Distributed Denial of Service) [9].

Οι επιθέσεις DDoS, όπως αναφέρει και το όνομά τους, έχουν ως στόχο να προκαλέσουν δυσλειτουργίες σε υπηρεσίες των δικτύων που θέλουν να βλάψουν, ώστε να τις καταστήσουν απρόσιτες σε νόμιμους χρήστες. Τέτοιες επιθέσεις αναφέρονται ως κατανεμημένες, γιατί η κακόβουλη κίνηση πηγάζει από ένα μεγάλο αριθμό μηχανημάτων, διάσπαρτων στο διαδίκτυο. Αυτά τα μηχανήματα, έχοντας μολυνθεί από κακόβουλο κώδικα, μετατρέπονται σε bots, δηλαδή μηχανήματα που μπορούν να εκτελέσουν αυτόματα εντολές που τους έχει ορίσει ο επιτιθέμενος. Δημιουργείται έτσι ένα δίκτυο από bots (botnet) [10], το οποίο χρησιμοποιεί ο επιτιθέμενος, ώστε να στείλει ταυτόχρονα έναν τεράστιο όγκο κακόβουλης κίνησης. Η κίνηση προέρχεται από ένα μεγάλο σύνολο διαφορετικών διευθύνσεων, διευθύνσεις που πολλές φορές είναι και τροποποιημένες (spoofed), για να δυσκολέψουν ακόμα περισσότερο την ανίχνευση του επιτιθέμενου αλλά και την αντιμετώπιση της επίθεσης. Ως συνέπεια, υπηρεσίες του δικτύου στόχου αναλώνονται στο να ανταποκριθούν στα πολυπληθή αιτήματα της κακόβουλης κίνησης, και οι νόμιμοι χρήστες αδυνατούν να αποκτήσουν πρόσβαση σε αυτές, ενώ και οι συσκευές δικτύου πολλές φορές αδυνατούν να ανταπεξέλθουν στην προώθηση τέτοιων ποσοτήτων κίνησης και αρχίζουν να απορρίπτουν πακέτα που μπορεί να αποτελούν και μέρος καλόβουλης κίνησης.

Για να αυξήσουν τα αποτελέσματα επιθέσεων, πολλές φορές χρησιμοποιούνται έμμεσα ως μέρος της επίθεσης και εξυπηρετητές που δεν έχουν μολυνθεί από κακόβουλο κώδικα με τεχνικές κατοπτρισμού(reflection) και ενίσχυσης (amplification). Στις τεχνικές αυτές, τα bots παριστάνουν τον στόχο και στέλνουν μαζικά αιτήματα σε διάφορους μη μολυσμένους εξυπηρετητές. Τα αιτήματα αυτά είναι κατάλληλα ώστε να έχουν μικρό μέγεθος, αλλά δημιουργούν αρκετά μεγαλύτερες απαντήσεις. Οι εξυπηρετητές, θεωρώντας πως πρόκειται για κανονικό αίτημα, στέλνουν τις απαντήσεις στον στόχο, με αποτέλεσμα αυτός να κατακλύζεται από τον όγκο τους.

Οι επιθέσεις DDoS είναι, σύμφωνα με στατιστικά [11], μια από τις σημαντικότερες διαδικτυακές απειλές την σημερινή εποχή, με μεγάλο αριθμό επιθέσεων, και αντίκτυπο τόσο στο κύρος όσο και στην οικονομία των θυμάτων. Επομένως η γρήγορη αντιμετώπιση τέτοιων επιθέσεων κρίνεται επιτακτική. Προφανώς το πρώτο βήμα για την αντιμετώπισή τους είναι η έγκαιρη ανίχνευση αυτών των επιθέσεων. Η ανίχνευση βασίζεται σε διάφορες μετρικές,

σχετικές με τις ροές πακέτων, την απόκλιση από το φυσιολογικό μοτίβο κίνησης, ασυμμετρίες που παρατηρούνται μεταξύ της εισερχόμενης και εξερχόμενης κίνησης. Τέτοιες μετρικές αναμένεται να αλλάξουν σημαντικά κατά την διάρκεια μιας επίθεσης, μιας και ένας μεγάλος αριθμός πακέτων θα παρατηρείται για πρώτη φορά και θα προέρχεται από πάρα πολλές διαφορετικές διευθύνσεις. Ως αποτέλεσμα ο παρατηρούμενος αριθμός ροών θα αυξηθεί και η συμμετρία της κίνησης του δικτύου θα αλλάξει σημαντικά υπέρ της εισερχόμενης κίνησης, στην οποία θα συμπεριλαμβάνεται και η κακόβουλη κίνηση. Επιπλέον μιας και έχει μεγάλη σημασία η ταχύτητα ανίχνευσης της επίθεσης, μια λύση ανίχνευσης στο επίπεδο δεδομένων προγραμματιζόμενων συσκευών, θα μπορούσε να οδηγήσει σε πιο σύντομη ανίχνευση συγκριτικά με τις παραδοσιακές μεθόδους, με αποτέλεσμα να μπορούν να ληφθούν πιο έγκαιρα μέτρα αντιμετώπισης.

Κεφάλαιο 3 – Συναφείς Εργασίες και Επισκόπηση Μηχανισμού

3.1 Σχετικές Εργασίες

Πριν από την παρουσίαση του προτεινόμενου μηχανισμού, είναι συνετό να αναφερθούμε σε κάποιες σχετικές εργασίες που αποτέλεσαν πηγή έμπνευσης και βάση της συγκεκριμένης διπλωματικής.

3.1.1 Hashpipe

Σκοπός της συγκεκριμένης εργασίας [12] είναι η ανίχνευση των k μεγαλύτερων "heavy hitter" ροών αποκλειστικά στο επίπεδο δεδομένων με βάση τον αλγόριθμο HashPipe. Ως "heavy hitters" χαρακτηρίζονται οι ροές με μεγάλο όγκο πακέτων, η ανίχνευση των οποίων συμβάλει σε ένα μεγάλο σύνολο δικτυακών λειτουργιών όπως στην ανίχνευση και αντιμετώπιση επιθέσεων και στην δυναμική διαχείριση και προώθηση κίνησης.

Ο αλγόριθμος HashPipe βασίζεται στον Space Saving Algorithm, ο οποίος χρησιμοποιεί έναν πίνακα για να αποθηκεύει ζευγάρια από ροές και τα αντίστοιχα μεγέθη τους. Από κάθε καινούριο πακέτο εξάγεται η αντίστοιχη ροή. Εφόσον αυτή υπάρχει στον πίνακα τότε το μέγεθός της αυξάνεται κατά 1. Αν δεν υπάρχει και ταυτόχρονα υπάρχουν άδειες θέσεις στον πίνακα τότε εισάγεται κανονικά, αλλιώς αντικαθιστά την ροή του πίνακα με το μικρότερο μέγεθος, το οποίο αυξάνεται κατά 1.

Για να καταστεί ικανή η εφαρμογή αυτής της λογικής πάνω σε προγραμματιζόμενους μεταγωγείς, έπρεπε να λυθούν κάποια εγγενή προβλήματα που αφορούσαν απαιτήσεις σε χρονική και χωρική απόδοση. Αρχικά αντιμετωπίστηκε το θέμα της εύρεσης του ελάχιστου στοιχείου, με την αναζήτηση του ελάχιστου όχι σε όλο τον πίνακα αλλά σε d τυχαίες θέσεις με την χρήση hash functions.

Στην συνέχεια, για να γίνεται αποδοτική πρόσβαση στην μνήμη, ο αρχικός πίνακας σπάει σε d επιμέρους πίνακες, όπου μπορούν να προσπελαστούν παράλληλα από πακέτα. Η διαδικασία πλέον γίνεται σε επίπεδα όπου σε κάθε ένα προσπελάζεται μια θέση ενός από τους d πίνακες.

Τέλος για να μην απαιτείται και δεύτερο πέρασμα πακέτων από τον μηχανισμό, ένα για να βρεθεί το ελάχιστο από όλα τα επίπεδα και δεύτερο για να ενημερωθεί, αρχικά εισάγονται πάντα οι καινούριες ροές στο πρώτο επίπεδο αντικαθιστώντας αν χρειαστεί υπάρχουσα ροή, η οποία και αποτελεί την νέα ροή σύγκρισης. Στα επόμενα στάδια, εφόσον υπάρχουν συγκρούσεις, εισάγουμε στον πίνακα την μεγαλύτερη ροή και κρατάμε την μικρότερη για να την συγκρίνουμε στην συνέχεια. Με αυτόν τον τρόπο μεταφέρουμε μεταφέρουμε πρακτικά τις μικρότερες ροές σε επόμενα επίπεδα μέχρι να εισαχθούν στον πίνακα κάποιου επόμενου επιπέδου ή μέχρι να απορριφθούν στο τελευταίο επίπεδο. Με αυτές τις μετατροπές ορίζεται ο αλγόριθμος HashPipe, όπως φαίνεται και στην εικόνα 3.2.

```
1                                  ▷ Insert in the first stage
2  l_1 ← h_1(iKey)
3  if key_{l_1} = iKey then
4  |   val_{l_1} ← val_{l_1} + 1
5  |   end processing
6  end
7  else if l_1 is an empty slot then
8  |   (key_{l_1}, val_{l_1}) ← (iKey, 1)
9  |   end processing
10 end
11 else
12 |   (cKey, cVal) ← (key_{l_1}, val_{l_1})
13 |   (key_{l_1}, val_{l_1}) ← (iKey, 1)
14 end
15                                 ▷ Track a rolling minimum
16 for i ← 2 to d do
17 |   l ← h_i(cKey)
18 |   if key_l = cKey then
19 |   |   val_l ← val_l + cVal
20 |   |   end processing
21 |   end
22 |   else if l is an empty slot then
23 |   |   (key_l, val_l) ← (cKey, CVal)
24 |   |   end processing
25 |   end
26 |   else if val_l < cVal then
27 |   |   swap (cKey, cVal) with (key_l, val_l)
28 |   end
29 end
```

***Εικόνα 3.2**: Αλγόριθμος HashPipe*

Η υλοποίηση του παραπάνω αλγορίθμου έγινε στην έκδοση $P4_{14}$. Για την εκτέλεση των ενεργειών χρησιμοποιήθηκε ένα table σε κάθε επίπεδο με ένα προκαθορισμένο action. Για τους πίνακες χρησιμοποιήθηκαν πίνακες από registers, ένας για τις ροές και ένας για το μέγεθός τους ανά επίπεδο. Για την μεταφορά του ελάχιστου στοιχείου στα διάφορα επίπεδα χρησιμοποιήθηκαν μεταδεδομένα.

Όσον αφορά τα πειραματικά αποτελέσματα, αρχικά έγινε μια μελέτη για την επιρροή του πλήθους d των επιπέδων του αλγορίθμου στην απόδοση του αλγορίθμου σχετικά. Κρατώντας πάντα σταθερό το συνολικό μέγεθος των πινάκων, έδειξαν πως αυξάνοντας τα επίπεδα αυξάνεται και η απόδοση του αλγορίθμου, μέχρι ένα σημείο όπου αύξηση στο d παρουσιάζει ελάχιστη βελτίωση. Έτσι κατέληξαν σε μια σταθερή τιμή του d για την συνέχεια των πειραμάτων. Στην συνέχεια έγινε μέτρηση της ακρίβειας του αλγορίθμου σχετικά με την μη εύρεση αληθινών heavy hitters (false negatives) και την εύρεση ψευδών (false positives), όπου έδειξαν ότι αυξάνοντας την μνήμη και μειώνοντας τον επιθυμητό αριθμό k των μέγιστων ροών, πετυχαίνουν πολύ μικρά ποσοστά λαθών (<10% false negatives και <3% false positives), χάνοντας κατά κύριο λόγο τις μικρότερες από τις k heavy-hitter ροές. Επιπλέον, έγινε σύγκριση με μεθόδους που βασίζονται σε sampling και σε count-min sketches, στην οποία φάνηκε να υπερέχει αισθητά ο αλγόριθμος HashPipe τόσο ως προς την ανίχνευση των μέγιστων ροών, όσο και ως προς το εύρος λάθους στον υπολογισμό των μεγεθών των μέγιστων ροών.

3.1.2 Παρακολούθηση Ροών μέσω Bloom-Filters

Στην συγκεκριμένη εργασία [13] μελετήθηκε η δυνατότητα ανάπτυξης δομών bloom-filters πάνω στην γλώσσα $P4_{14}$ για την παρακολούθηση ροών πακέτων, και έγινε μια αρχική δοκιμή αυτών των δομών για την ανίχνευση κακόβουλων ροών. Αν και θα αναλυθούν καλύτερα σε επόμενο κεφάλαιο, τα bloom-filters είναι πιθανοτικές δομές που χρησιμοποιούνται για να ελέγξουν την συμμετοχή ενός στοιχείου σε ένα σύνολο. Αρχικά γίνεται μια εισαγωγή στα bloom filters και μια εκτενής παρουσίαση για τον τρόπο δημιουργίας τους στο $P4_{14}$,

αναφερόμενοι ταυτόχρονα και σε περιορισμούς της γλώσσας $P4_{14}$. Στην συνέχεια, μέσω υλοποίησης πάλι σε $P4_{14}$, γίνεται μια ποιοτική ανάλυση της ακρίβειας των bloom-filters ώστε να διαπιστωθεί η ικανότητα ανίχνευσης διαφορετικών ροών. Έπειτα περιγράφονται και τα counting bloom-filters, μια τροποποίηση των bloom-filters ώστε να μπορούν να υποδείξουν και συχνότητα εμφάνισης του στοιχείου στο σύνολο. Γίνεται και πάλι αναφορά στην υλοποίηση των counting bloom-filters στο $P4_{14}$ και μελέτη της απόδοσής τους.

Με βάση τα counting bloom-filters υλοποιείται και ένας μηχανισμός ανίχνευσης επιθέσεων DDoS τύπου SYN flood. Οι επιθέσεις SYN flood βασίζονται στην λεγόμενη τριμερή χειραψία του πρωτοκόλλου TCP, κατά την οποία το πρώτο πακέτο έχει ενεργοποιημένη την σημαία SYN. Η επίθεση εκμεταλλεύεται αυτήν την σημαία, στέλνοντας σε εξυπηρετητές πολλά ψεύτικα πακέτα για σύναψη σύνδεσης. Αποτέλεσμα αυτού είναι να εξαντλούνται τα περιθώρια σύναψης νέας σύνδεσης από τον εξυπηρετητή και να απορρίπτονται νέα αιτήματα. Ο εν λόγω μηχανισμός χρησιμοποιεί δύο counting bloom-filters για να μετρήσει για κάθε ροή τον αριθμό των πακέτων SYN και τον αριθμό των υπόλοιπων πακέτων της ροής. Στην συνέχεια θα υπολογιστεί ο μεταξύ τους λόγος και αν υπερβεί κάποιο κατώφλι, τότε θα θεωρηθεί κακόβουλη ροή.

Για την μέτρηση της απόδοσης του συγκεκριμένου μηχανισμού έγιναν πειράματα σχετικά με την αδυναμία ανίχνευσης κακόβουλης ροής (ποσοστό false negatives) αναλογικά με το πλήθος των κακόβουλων ροών. Τα αποτελέσματα έδειξαν ότι όσο αυξάνεται ο αριθμός των κακόβουλων ροών τόσο δυσκολεύει η ανίχνευσή τους, μιας και τα πακέτα της κακόβουλης κίνησης μοιράζονται καλύτερα και δεν ξεπερνούν οι ροές το αντίστοιχο κατώφλι. Ωστόσο σε μικρό αριθμό ροών εμφανίζει καλή συμπεριφορά, καθιστώντας τους συγγραφείς αισιόδοξους για την χρησιμότητα των bloom-filters σε αλγορίθμους ανίχνευσης ροών στο επίπεδο δεδομένων.

Τέλος αξίζει να σημειώσουμε ότι στην συγκεκριμένη εργασία αναφέρεται πως η πλατφόρμα πειραματισμού βασίστηκε στο Bmv2 software switch [14], έναν εικονικό μεταγωγό ο οποίος χρησιμοποιείται κατά κύριο λόγο για την ανάπτυξη προγραμμάτων P4 και την δοκιμή της ορθότητάς τους. Ωστόσο, όπως παρατηρήθηκε στην εργασία και όπως θα δούμε και στην συνέχεια δεν παρουσιάζει αποδεκτές επιδόσεις ως προς τον ρυθμό επεξεργασίας και προώθησης των πακέτων.

3.1.3 Ανίχνευση DDoS επιθέσεων μέσω εντροπίας Shannon επί των ροών.

Στο πλαίσιο αυτής της έρευνας [15], αναπτύχθηκε ένας μηχανισμός ανίχνευσης DDoS επιθέσεων βασισμένος στην εντροπία Shannon και σε δομές sketch. Η εντροπία Shannon θεωρητικά αποτελεί ένα μέσο μέτρησης της τυχαιότητας των στοιχείων ενός δείγματος. Όσον αφορά το συγκεκριμένο πρόβλημα, όσο αυξάνεται ο αριθμός πακέτων με διαφορετικές διευθύνσεις τόσο θα μειώνεται η εντροπία και το αντίθετο, και με βάση αυτήν την τιμή και κάποια όρια παίρνεται η απόφαση αν υπάρχει επίθεση στο δίκτυο ή όχι. Τα sketches αποτελούν πιθανοτικές δομές που χρησιμεύουν στην μέτρηση της συχνότητας εμφάνισης ενός στοιχείου σε ένα σύνολο και χρησιμοποιούνται στην εργασία για να μετρηθούν οι διαφορετικές διευθύνσεις πηγής και προορισμού των πακέτων.

Πριν προχωρήσουμε στην ανάλυση του μηχανισμού θα πρέπει να αναφέρουμε τους τύπους στους οποίους βασίζεται ο μηχανισμός. Η εντροπία στο συγκεκριμένο πλαίσιο δίνεται από τον τύπο $H = \log(m) - \frac{1}{m}\sum f_x \log(f_x)$ όπου m το σύνολο των πακέτων και f_x η συχνότητα εμφάνισης κάθε διαφορετικής διεύθυνσης. Η νόρμα εντροπίας S ορίζεται ως $S = \sum f_x \log(f_x)$.

Ο μηχανισμός ξεκινάει με τον υπολογισμό των διαφορετικών διευθύνσεων της κίνησης. Υπάρχουν δύο διαφορετικά sketches για διευθύνσεις πηγής και προορισμού αντίστοιχα. Για κάθε πακέτο που εισέρχεται στην συσκευή, ενημερώνονται οι συχνότητες των αντίστοιχων διευθύνσεων και στην συνέχεια ενημερώνεται η τιμή της αντίστοιχης νόρμας της εντροπίας με το εξής τέχνασμα: $S \leftarrow S + f_x \log(f_x) - (f_x - 1)\log(f_x - 1)$. Πρακτικά δηλαδή ενημερώνεται ο βαθμός συνεισφοράς της συγκεκριμένης διεύθυνσης στην νόρμα, και ο υπολογισμός της μοιράζεται στο σύνολο των πακέτων. Μιας και δεν είναι δυνατός ο υπολογισμός λογαρίθμων στο πλαίσιο του P4, χρησιμοποιήθηκαν πίνακες με προ-υπολογισμένες τιμές της συνάρτησης $f_x \log(f_x) - (f_x - 1)\log(f_x - 1)$. Στον πίνακα αυτό, η αναζήτηση γίνεται με βάση την τωρινή συχνότητα f_x με την τεχνική lpm.

Μόλις τελειώσει το παράθυρο των m πακέτων τότε γίνεται ο υπολογισμός των εντροπιών διεύθυνσης πηγής και προορισμού και ο έλεγχος των ορίων για την ανίχνευση επίθεσης. Θέτοντας ως μέγεθος παραθύρου m κάποια n-οστή δύναμη του 2, ο παραπάνω τύπος μετατρέπεται στον $H = n - S \gg n$ όπου οι πράξεις που απαιτούνται πλέον είναι αφαίρεση και μετατόπιση (shifting) που υποστηρίζονται από το P4. Έπειτα μέσω των μεθόδων Exponentially Weighted Moving Average (EWMA) και Exponentially Weighted Moving Mean Difference (EWMMD), οι οποίες θα αναλυθούν στο επόμενο κεφάλαιο, υπολογίζονται χρονικές μέσες τιμές M και αποκλίσεις D και μέσω αυτών ορίζονται τα όρια ανίχνευσης ως: $H > M + k \cdot D$ για τις διευθύνσεις πηγής και $H < M - k \cdot D$ για διευθύνσεις προορισμού, όπου k ένας παράγοντας ευαισθησίας. Εφόσον κάποια από τις δύο συνθήκες ικανοποιηθεί τότε ο μηχανισμός αναγνωρίζει επίθεση.

Για την μέτρηση των επιδόσεων του αλγορίθμου χρησιμοποιήθηκε το Bmv2 software switch. Αρχικά μελετήθηκε η σχέση μεταξύ της μνήμης που χρησιμοποιείται από τα sketches και τα σφάλματα στον υπολογισμό της εντροπίας και αποδείχτηκε ότι αυξάνοντας τα μεγέθη των δομών, μπορούν να επιτευχθούν μικρότερα σφάλματα.

Στην συνέχεια, έγινε μία μελέτη της συμπεριφοράς του αλγορίθμου σχετικά με τον παράγοντα ευαισθησίας k. Σε αυτήν δείχτηκε ότι υπάρχει μια περιοχή τιμών που διατηρεί ταυτόχρονα υψηλό ποσοστό σωστών εκτιμήσεων επίθεσης (true positive ratio - TPR) και χαμηλό ποσοστό λανθασμένων (false positive ratio - FPR). Πριν από αυτή την περιοχή παρατηρείται αυξημένο FPR, δηλαδή διακυμάνσεις στην κανονική κίνηση προκαλούν λανθασμένη αναγνώριση. Αντίθετα μετά την περιοχή αυτή παρατηρείται μειωμένο TPR, και συνεπώς δεν μπορεί ο μηχανισμός να ανιχνεύσει μικρότερης έντασης επιθέσεις.

Τέλος, έγινε σύγκριση μεταξύ της εφαρμογής του μηχανισμού στο επίπεδο δεδομένων και την χρήση του ίδιου μηχανισμού συνεργατικά με δειγματοληψία, τα αποτελέσματα της οποίας έδειξαν υπεροχή του μηχανισμού όταν αυτός τρέχει πάνω σε προγραμματιζόμενη συσκευή.

3.1.4 Πρόσθετες Αναφορές

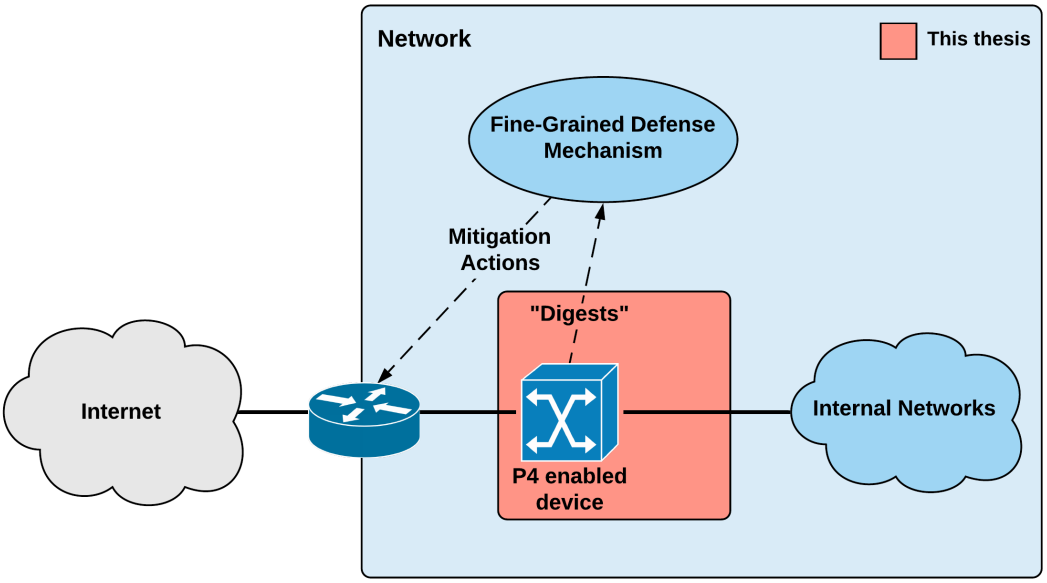
Πριν προχωρήσουμε στο επόμενο κεφάλαιο και στην διεξοδική ανάλυση του μηχανισμού, κρίνεται σκόπιμο να απαριθμήσουμε μερικές εργασίες που χρησιμοποιούν την δυνατότητα του προγραμματισμού του επιπέδου δεδομένων για σκοπούς διαφορετικούς από την ανίχνευση επιθέσεων. Με αυτό θέλουμε να τονίσουμε ότι η σημασία της νέας αυτής τάσης των προγραμματιζόμενων συσκευών δεν εμφανίζεται μόνο στον συγκεκριμένο τομέα, αλλά και σε ένα μεγάλο εύρος άλλων λειτουργιών του διαδικτύου:

- **DC.p4** [16]: Αποτελεί μια από τις πρώτες εργασίες πάνω στο P4, το οποίο και χρησιμοποιήθηκε στο πλαίσιο της εργασίας για να εκφράσει τις λειτουργίες προώθησης ενός data-center switch.
- **P4CEP** [17]: Μελετά την ανάλυση σύνθετων γεγονότων (Complex Event Processing) και την μεταφορά τέτοιας λογικής στο επίπεδο δεδομένων δικτυακών συσκευών.
- **HULA** [18]: Ανάπτυξη του μηχανισμού HULA και υλοποίησή του στο P4, με σκοπό την εξισορρόπηση του φόρτου (load balancing) της δικτυακής κίνησης.

3.2 Παρουσίαση Προτεινόμενου Μηχανισμού

Ο μηχανισμός που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής στοχεύει στην διερεύνηση των δυνατοτήτων του προγραμματισμού επιπέδου δεδομένων, που αφορούν στην ανάλυση της κίνησης και στην ανίχνευση επιθέσεων, σύμφωνα με τα στοιχεία που αναφέρθηκαν στο κεφάλαιο 2. Συγκεκριμένα κατά την διαδικασία προώθησης των πακέτων, συλλέγονται στοιχεία σχετικά με εισερχόμενες ροές πακέτων και ασυμμετρίας της κίνησης, ώστε να μπορεί ενεργά η προγραμματιζόμενη συσκευή να ενημερώσει το επίπεδο ελέγχου και κατ' επέκταση επόμενα στάδια του αμυντικού μηχανισμού για τυχόν επίθεση που έχει ανιχνευθεί.

Από μια υψηλή σκοπιά, ο εν λόγω μηχανισμός ανήκει στην περίμετρο του δικτύου διαχείρισης, το εσωτερικό του οποίου θέλουμε να προστατέψουμε. Πρόκειται για προγραμματιζόμενες συσκευές οι οποίες υποστηρίζουν το πρωτόκολλο P4 και έχουν αναλάβει την προώθηση της κίνησης των υποδικτύων που θέλουμε να ελέγχουμε ενεργά, ώστε να μπορούν παράλληλα να κάνουν και την απαιτούμενη ανάλυση. Με βάση τα αποτελέσματα ανάλυσης θα μπορούν να στείλουν ειδοποιήσεις, ή αλλιώς "digests" όπως αναφέρεται στο πλαίσιο του P4, σε επόμενο στάδιο ενός εξειδικευμένου αμυντικού μηχανισμού αντιμετώπισης επιθέσεων. Αυτός με την σειρά του θα είναι υπεύθυνος για την λεπτομερέστερη ανάλυση της κακόβουλης κίνησης και τον περιορισμό της επίθεσης. Επομένως οι προγραμματιζόμενες συσκευές και ο προτεινόμενος μηχανισμός της παρούσας διπλωματικής θα αποτελούν το πρώτο στάδιο ενός ολοκληρωμένου μηχανισμού αντιμετώπισης επιθέσεων. Η σχηματική αναπαράσταση μιας τέτοιας αρχιτεκτονικής παρουσιάζεται στο επόμενο σχήμα:



***Εικόνα 3.1**: Προτεινόμενη αρχιτεκτονική βασισμένη σε προγραμματιζόμενες συσκευές.*

Επιγραμματικά τα βασικά στοιχεία που καθιστούν χρήσιμο έναν τέτοιο μηχανισμό είναι τα εξής:

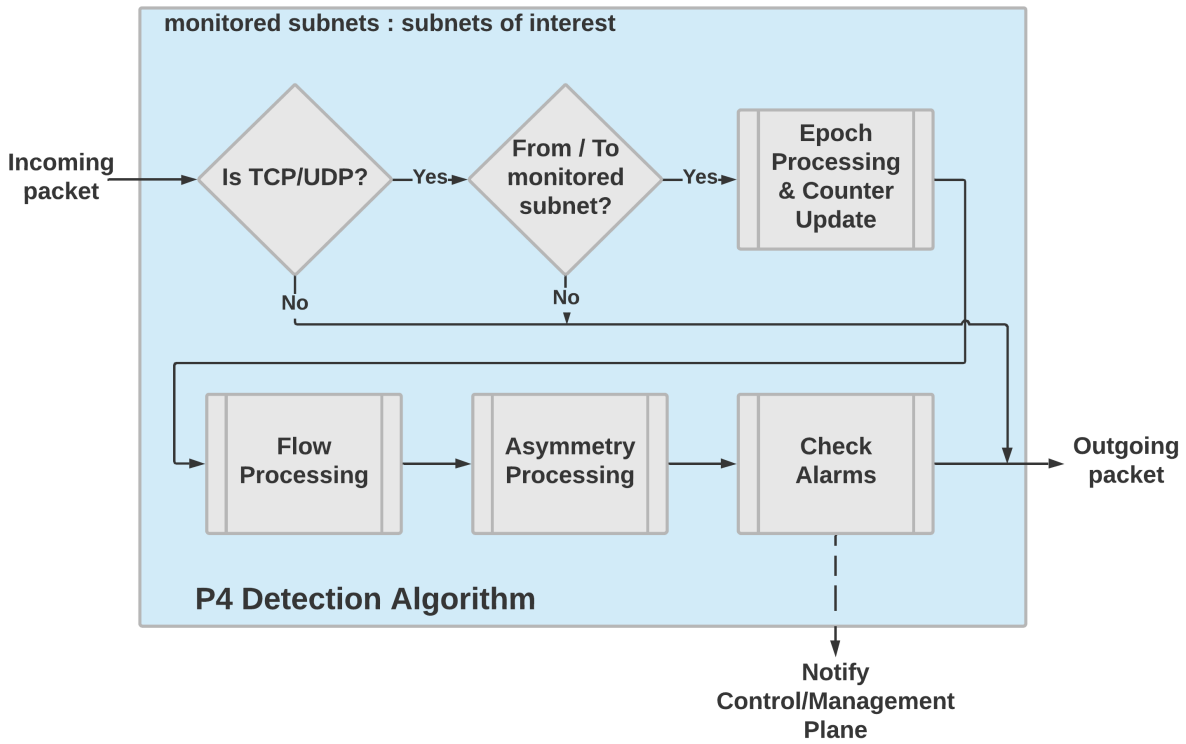
- **In-line ανάλυση**: Επωφελούμαστε από την δυνατότητα προγραμματισμού των λειτουργιών προώθησης των συσκευών ώστε να εξάγουμε τα επιθυμητά στατιστικά παράλληλα με την προώθηση της κίνησης. Με αυτόν τον τρόπο δεν απαιτείται η χρήση πρόσθετων τεχνικών συλλογής δεδομένων, κερδίζοντας πολύτιμο χρόνο στην προσπάθεια ανίχνευσης των επιθέσεων.
- **Συνδυασμός μετρικών**: Ο μηχανισμός βασίζεται στον υπολογισμό μετρικών που αφορούν τις ροές πακέτων συνολικά και ανά υποδίκτυο, καθώς και την ασυμμετρία της κίνησης ανά υποδίκτυο. Χρησιμοποιώντας ταυτόχρονα τις μετρικές αυτές μπορούμε να αυξήσουμε το ποσοστό επιτυχίας του μηχανισμού και παράλληλα να παρέχουμε μια πιο λεπτομερή ενημέρωση σχετικά με τον πιθανό στόχο.
- **Ορισμός ελεγχόμενων υποδικτύων**: Μπορούμε ενεργά να τροποποιούμε το σύνολο των υποδικτύων που θα ελέγχει ο μηχανισμός. Έτσι μπορούμε να σπάσουμε ένα υποδίκτυο που θεωρούμε πως αποτελεί πιθανό στόχο σε επιμέρους ώστε να πετύχουμε λεπτομερέστερη ανίχνευση, ενώ παράλληλα να συνενώσουμε κάποια δίκτυα που θα ήταν λιγότερο πιθανό να υποστούν επίθεση.
- **Αποστολή ειδοποιήσεων**: Σε περίπτωση ανίχνευσης πιθανής επίθεσης, γίνεται άμεση αποστολή αντίστοιχης ειδοποίησης από τον μηχανισμό, χωρίς να απαιτείται αίτημα από κάποιον ελεγκτή. Συνεπώς απαιτούνται λιγότεροι πόροι για την επικοινωνία ελεγκτή και συσκευής και γίνεται γρηγορότερη ενημέρωση των επόμενων σταδίων αντιμετώπισης.
- **Μεταφερσιμότητα**: Ο μηχανισμός βασίζεται εξ' ολοκλήρου στις προδιαγραφές της γλώσσας P4, και δεν εξαρτάται από συγκεκριμένες λειτουργίες της εκάστοτε συσκευής. Επομένως μπορεί αυτούσιος να εφαρμοστεί σε πλήθος συσκευών χωρίς να απαιτείται τροποποίησή του, εφόσον οι συσκευές υποστηρίζουν το P4.

Κεφάλαιο 4 - Ανάλυση Υλοποίησης

4.1 Επεξήγηση Μηχανισμού

Ο μηχανισμός που αναπτύχθηκε στην παρούσα διπλωματική εργασία στοχεύει στην ανίχνευση ανωμαλιών που προκαλούνται από επιθέσεις DDoS στην δικτυακή κίνηση. Σε συνέχεια της ανίχνευσης καλείται να δημιουργήσει ειδοποιήσεις από το επίπεδο δεδομένων, ώστε να χρησιμοποιηθούν από επόμενα στάδια ενός συνολικού αμυντικού μηχανισμού για την αντιμετώπιση της επίθεσης.

Η ανάπτυξη του μηχανισμού έγινε στην γλώσσα P4 και στην έκδοση $P4_{16}$. Στο παρακάτω διάγραμμα αναπαριστάται συνολικά η λειτουργία του.



***Εικόνα 4.1**: Συνολική Αναπαράσταση του Μηχανισμού*

Η λογική στην οποία στηρίζεται ο μηχανισμός είναι η ακόλουθη. Θεωρούμε ότι υπάρχει ένα σύνολο υποδικτύων (monitored subnets) που θέλουμε να προστατέψουμε από επιθέσεις DDoS που προέρχονται από το διαδίκτυο. Έτσι εισάγουμε αυτόν τον μηχανισμό σε κατάλληλο σημείο, ώστε να μπορεί να εξυπηρετεί όλη την κίνηση από και προς αυτά τα υποδίκτυα, πραγματοποιώντας παράλληλα και την ανάλυσή της.

Η ανάλυση της κίνησης βασίζεται σε τρεις επιμέρους μηχανισμούς. Οι δύο πρώτοι μηχανισμοί βασίζονται στην μέτρηση των εισερχόμενων ροών, ο πρώτος συνολικά για όλα τα δίκτυα και ο δεύτερος για το συγκεκριμένο υποδίκτυο το οποίο αφορά η κάθε ροή. Πρέπει να αναφέρουμε πως με τον όρο ροή εννοούμε κάθε πεντάδα της μορφής:
(Διεύθυνση IP πηγής, Διεύθυνση IP προορισμού, πρωτόκολλο μεταφοράς, πόρτα πηγής, πόρτα προορισμού)
ή αλλιώς **(src IP, dst IP, proto, src port, dst port)**. Ο τρίτος μηχανισμός βασίζεται στην μέτρηση των εισερχόμενων και εξερχόμενων πακέτων και στην εξαγωγή αποτελέσματος για ασυμμετρία για κάθε υποδίκτυο. Κάθε ένας από τους τρεις μηχανισμούς, εφόσον διαπιστώσει ανωμαλία σηκώνει μια "σημαία" (flag - alarm). Εφόσον και οι τρεις σημαίες σηκωθούν τότε ο μηχανισμός συμπεραίνει ότι υπάρχει κάποια επίθεση και δημιουργεί ειδοποίηση (digest) για να ενημερώσει επόμενα επίπεδα.

Στην συνέχεια θα περιγράψουμε αναλυτικά κάθε στάδιο του μηχανισμού, αναλύοντας ταυτόχρονα και την υλοποίησή του στο $P4_{16}$.

4.2 Ανίχνευση πρωτοκόλλου μεταφοράς

Ο μηχανισμός ξεκινάει με τον έλεγχο του πρωτοκόλλου μεταφοράς του κάθε πακέτου, και πιο συγκεκριμένα αν πρωτόκολλο μεταφοράς είναι είτε TCP είτε UDP. Ο έλεγχος αυτός χρησιμοποιείται για τον ορισμό της ροής του πακέτου, αλλά και για τον υπολογισμό της ασυμμετρίας κίνησης, μιας και η τελευταία υπολογίζεται ξεχωριστά για κάθε πρωτόκολλο. Επιλέγουμε αυτά τα δύο πρωτόκολλα αφενός για να μπορέσουμε να ορίσουμε σωστά την ροή για το κάθε πακέτο με βάση τον παραπάνω ορισμό, αφετέρου διότι μεγάλος αριθμός των επιθέσεων DDoS βασίζεται σε υπηρεσίες που χρησιμοποιούν τα δύο αυτά πρωτόκολλα.

Η αναγνώριση του πρωτοκόλλου γίνεται με την βοήθεια του parser. Στον ορισμό του, υπάρχει η κατάσταση **parse_ipv4** κατά την οποία διαβάζεται η IP επικεφαλίδα του πακέτου και εισάγεται σε δομή τύπου **header ipv4_t** και συγκεκριμένα στην επικεφαλίδα **ipv4**. Σε αυτήν συμπεριλαμβάνεται και το πεδίο protocol, η τιμή του οποίου περιγράφει τον τύπο της επόμενης επικεφαλίδας. Με βάση αυτό το πεδίο γίνεται και υπό συνθήκη μετάβαση σε επόμενη κατάσταση. Αν αυτό το πεδίο έχει την τιμή 6 (δεκαεξαδική 0x06) όπου και πρόκειται για το πρωτόκολλο TCP, ή την τιμή 17 (δεκαεξαδική 0x11) όπου αντιστοιχεί στο UDP, τότε η επόμενη κατάσταση είναι η **parse_tcp_udp_ports**. Σε αυτήν την κατάσταση διαβάζονται οι θύρες που χρησιμοποιεί η συγκεκριμένη ροή πακέτων. Μιας και στα δύο πρωτόκολλα τα bits για τις θύρες βρίσκονται στην αρχή της επικεφαλίδας και είναι με την ίδια σειρά, και δεν χρειαζόμαστε επιπλέον πληροφορίες, μπορούμε να χρησιμοποιήσουμε την ίδια κατάσταση και για τα δύο. Οι τιμές αυτές αποθηκεύονται στην επικεφαλίδα **ports** τύπου **header tcp_udp_port_t**. Μόλις τελειώσει η κατάσταση **parse_tcp_udp_ports** ή αν το πεδίο protocol δεν έχει καμία από τις δύο αυτές τιμές, μεταφερόμαστε στην κατάσταση **accept** όπου και τελειώνει το parsing και αρχίζει το κύριο μέρος του μηχανισμού. Εκεί ελέγχουμε την εγκυρότητα της επικεφαλίδας **ports**. Αν αυτή είναι έγκυρη σημαίνει ότι η κατάσταση **parse_tcp_udp_ports** ολοκληρώθηκε και επομένως όντως το πακέτο χρησιμοποιεί ένα από τα δύο πρωτόκολλα. Σε αυτήν την περίπτωση επιτρέπουμε στο πακέτο να συνεχίσει σε επόμενες λειτουργίες του μηχανισμού. Αν η επικεφαλίδα είναι άκυρη τότε δεν χρησιμοποιείται κανένα από τα δύο πρωτόκολλα και το πακέτο προωθείται χωρίς επιπλέον επεξεργασία.

Πίνακας 4.1: Parsing και επικεφαλίδες του μηχανισμού

```
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.protocol) {
            TCP_PROTO: parse_tcp_udp_port;
            UDP_PROTO: parse_tcp_udp_port;
            default: accept;
        }
    }

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

```
header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}

header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<8>    diffserv;
    bit<16>   totalLen;
    bit<16>   identification;
    bit<3>    flags;
    bit<13>   fragOffset;
    bit<8>    ttl;
    bit<8>    protocol;
    bit<16>   hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}

header tcp_udp_port_t {
    bit<16> srcPort;
    bit<16> dstPort;
}

struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
    tcp_udp_port_t  ports;
}
```

4.3 Αναγνώριση Υποδικτύου

Εφόσον το πακέτο περάσει επιτυχώς από το στάδιο ανίχνευσης του πρωτοκόλλου, γίνεται η αναγνώριση των υποδικτύων που αφορά. Αυτό γίνεται καθώς μπορεί να υπάρχει κίνηση που δεν σχετίζεται με κάποιο από τα ελεγχόμενα υποδίκτυα, όπου η κίνηση αυτή θα προωθηθεί χωρίς περαιτέρω επεξεργασία. Μας ενδιαφέρουν δηλαδή μόνο τα πακέτα που είτε πηγάζουν είτε προορίζονται για κάποιο από τα ελεγχόμενα δίκτυα.

Για την αναγνώριση των υποδικτύων χρησιμοποιούνται δύο πίνακες (tables), ο **subnet_src** για την διεύθυνση πηγής, και ο **subnet_dst** για την διεύθυνση προορισμού. Σε αυτούς τους δύο πίνακες παρέχουμε από το control plane, με τρόπο που θα αναλύσουμε αργότερα, τους κανόνες αντιστοίχισης που συσχετίζουν κάθε υποδίκτυο με ένα μοναδικό αριθμό, ο οποίος θα

είναι υπεύθυνος για την πρόσβαση στις σωστές θέσεις μνήμης των δομών που θα ακολουθήσουν. Η μορφή των κανόνων αυτών είναι η εξής:
IP address, Mask => Identifier, Action.
Ο τρόπος με τον οποίο γίνεται η αντιστοίχιση ονομάζεται ternary. Όπως έχουμε αναφέρει κατά το ternary matching, οι τιμές ελέγχονται για ομοιότητα μετά την λογική σύζευξή τους με μία μάσκα. Στην συνέχεια μόλις βρεθεί ο κανόνας αντιστοίχισης τότε ενεργοποιείται το αντίστοιχο action, **get_src_subnet** ή **get_dst_subnet** ανάλογα τον πίνακα, μέσω των οποίων οι αριθμοί - αναγνωριστικά των δικτύων εισάγονται σαν μεταδεδομένα στην δομή **meta** τύπου **struct metadata**, στα πεδία **meta.s_subnet** και **meta.d_subnet** αντίστοιχα.

Πίνακας 4.2: Πίνακες αναγνώρισης υποδικτύων και αντίστοιχες συναρτήσεις

```
table subnet_src {
    key = {
        hdr.ipv4.srcAddr : ternary;
    }
    actions = {
        get_src_subnet;
        NoAction;
    }
    size = 258;
          default_action = NoAction();
}

table subnet_dst {
    key = {
        hdr.ipv4.dstAddr : ternary;
    }
    actions = {
        get_dst_subnet;
        NoAction;
    }
    size = 258;
          default_action = NoAction();
}
```

```
action get_src_subnet(bit<32> subnet) {
    meta.s_subnet = subnet;
}

action get_dst_subnet(bit<32> subnet) {
    meta.d_subnet = subnet;
}
```

Μιας και εργαζόμαστε πάνω σε υποδίκτυα θα ήταν πιο λογικό να εφαρμόσουμε αντιστοίχιση lpm. Ο λόγος που δεν το κάναμε είναι πως η τεχνική αυτή ενώ υπάρχει στις προδιαγραφές της γλώσσας $P4_{16}$ και υποστηρίζεται από το εικονικό switch bmv2 [14], δεν υποστηρίζεται από την κάρτα Netronome SmartNIC [19]. Αποφασίσαμε λοιπόν να χρησιμοποιήσουμε ternary matching και κατ' επέκταση στοιχεία που εμπεριέχονται αρχικά στην βασική αρχιτεκτονική του $P4_{16}$, αλλά ταυτόχρονα υποστηρίζονται και από τις δύο πλατφόρμες. Οι λόγοι για την συγκεκριμένη απόφαση είναι αφενός πως ο πειραματισμός θα γίνει και στις δύο πλατφόρμες οπότε είναι δόκιμο να χρησιμοποιούν τον ίδιο κώδικα, αφετέρου και κυριότερα πρόκειται για μια προσπάθεια να διαπιστώσουμε την έκταση της μεταφερσιμότητας του $P4_{16}$, το εύρος των λειτουργιών δηλαδή που μπορούμε να εκτελέσουμε χωρίς να χρειαστεί να

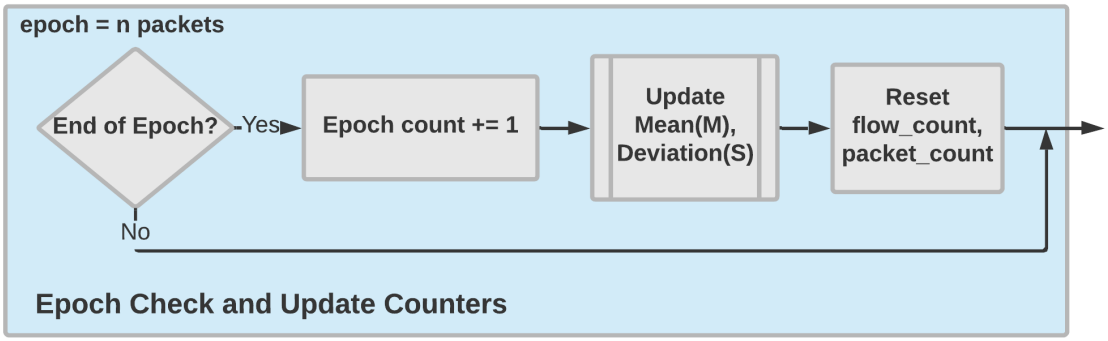
καταφύγουμε σε ενέργειες τροποποίησης των δυνατοτήτων της κάθε συσκευής. Παρόμοιες αποφάσεις θα παρθούν και για επόμενα στοιχεία του μηχανισμού.

Στην συγκεκριμένη περίπτωση, στην τεχνική ternary θα πρέπει να δοθεί προσοχή στην σειρά με την οποία θα εισάγουμε τους κανόνες στους πίνακες και στην προτεραιότητα που θα τους δώσουμε ώστε να εξασφαλίσουμε σωστή αντιστοίχιση. Αν υπάρχουν υποδίκτυα που μοιράζονται ένα εύρος διευθύνσεων θα πρέπει είτε να δώσουμε προτεραιότητα στους ειδικότερους κανόνες είτε να τους εισάγουμε νωρίτερα. Παράδειγμα στον μηχανισμό αποτελεί ο γενικός κανόνας αντιστοίχισης κάθε διεύθυνσης. Αντί να αποτυγχάνει η αντιστοίχιση για δίκτυα που δεν μας ενδιαφέρουν, ενεργοποιείται αυτός ο κανόνας παρέχοντας μια προκαθορισμένη τιμή. Η διαφορά είναι ότι ο έλεγχος θα γίνει στις τιμές που θα επιστραφούν και όχι στο αν πέτυχε ο πίνακας (υπήρξε hit).

Μόλις εκτελεστούν αυτοί οι δύο πίνακες λοιπόν ελέγχουμε τις τιμές **meta.s_subnet**, **meta.d_subnet**. Αν κάποια από αυτές δεν αντιστοιχεί στην προκαθορισμένη τιμή του γενικού κανόνα, τότε το πακέτο αφορά κάποιο από τα ελεγχόμενα υποδίκτυα, και συνεχίζεται ο μηχανισμός. Σε αντίθετη περίπτωση το πακέτο προωθείται άμεσα.

4.4 Έλεγχος Τέλους Εποχής

Μετά τον έλεγχο των υποδικτύων το πρώτο πράγμα που γίνεται είναι ο έλεγχος για το τέλος εποχής. Οι εποχές αποτελούν διαστήματα με βάση τα οποία εκτελούνται εργασίες ενημέρωσης για τις λειτουργίες του συνολικού μηχανισμού. Για την οριοθέτηση των εποχών μπορούν να χρησιμοποιηθούν πακετοπαράθυρα αλλά και χρονικά διαστήματα.



***Εικόνα 4.2**: Λογικό Διάγραμμα Αρχής Εποχής*

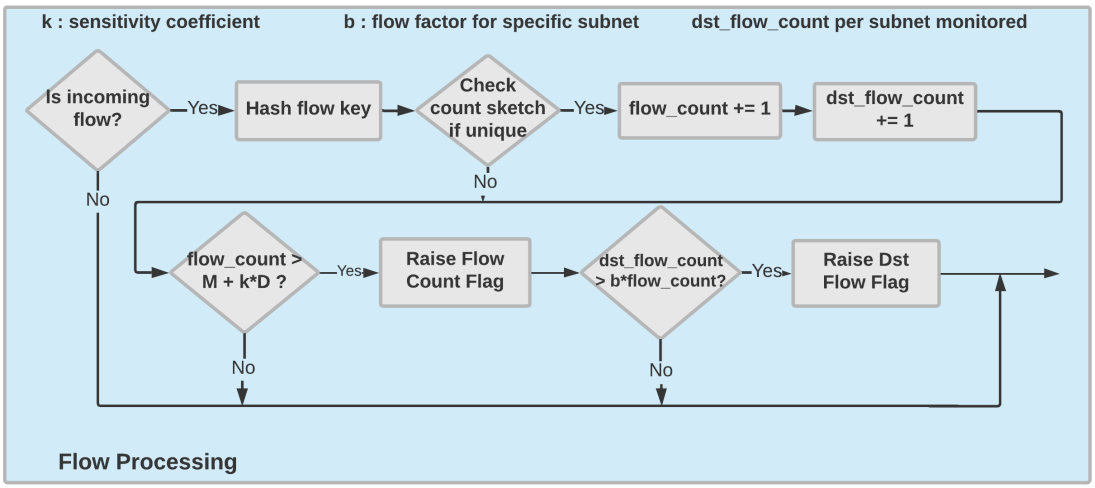
Για την χρήση πακετοπαραθύρων μπορεί να χρησιμοποιηθεί ένας register ως μετρητής πακέτων. Έχοντας θέσει ένα μέγεθος N πακέτων για τις εποχές, ο μετρητής κατά την εισαγωγή κάθε επιθυμητού πακέτου αυξάνεται, και ελέγχεται με αυτό το μέγεθος. Εφόσον ξεπεραστεί τότε εκτελούνται λειτουργίες ενημέρωσης των επιμέρους μηχανισμών και μηδενίζεται ο μετρητής, αποτελώντας την αρχή μιας καινούριας εποχής. Η επιλογή εποχών με βάση τον αριθμό πακέτων μπορεί να γίνει με σκοπό να ελέγξουμε την απόδοση των

πιθανοτικών δομών που θα χρησιμοποιήσουμε για την εύρεση του αριθμού των εισερχόμενων ροών. Μιας και οι δομές αυτές αυξάνουν όπως θα δούμε την πιθανότητα λάθους εκτίμησης όσο αυξάνεται το πλήθος των διαφορετικών ροών που αναγνωρίζουν και κατ' επέκταση το πλήθος πακέτων που ελέγχουν, τα πακετοπαράθυρα αποτελούν μία εγγύηση για την πιθανότητα σφαλμάτων αυτών των δομών.

Ωστόσο τα πακετοπαράθυρα δεν μπορούν να αναγνωρίσουν αύξηση της ποσότητας της κίνησης σε χρονικό πλαίσιο. Σε περίπτωση επίθεσης κάποια από τα πακέτα κακόβουλης κίνησης αντικαθιστούν αντίστοιχα καλόβουλης ώστε να διατηρείται ο αριθμός N σταθερός. Έτσι δεν μπορούμε να βγάλουμε συμπέρασμα για την χρονική εξέλιξη της κίνησης αλλά και πιθανής επίθεσης, μιας και ο χρόνος που καλύπτει η κάθε εποχή μεταβάλλεται. Έτσι εξίσου εφικτή είναι η χρήση χρονικών διαστημάτων για την μέτρηση των εποχών. Αν και δεν γίνεται στο $P4_{16}$ ο μηχανισμός να διαβάσει χρόνο από κάποιο ρολόι, κατά την είσοδο του κάθε πακέτου παρέχεται σαν μεταδεδομένα και μια χρονοσήμανση (**timestamp**) με τον χρόνο εισαγωγής του, που μπορεί να χρησιμοποιηθεί για τον ορισμό χρονικών εποχών. Ορίζοντας ένα επιθυμητό χρονικό διάστημα διάρκειας κάθε εποχής (**interval**) και κρατώντας της αρχή κάθε μιας σε κάποιο καταχωρητή , μπορούμε με βάση τον χρόνο εισαγωγής του κάθε πακέτου να επιτύχουμε τον διαχωρισμό των εποχών. Με αυτήν την υλοποίηση μπορούμε να έχουμε εικόνα για την χρονική εξέλιξη μιας επίθεσης καθώς οι κακόβουλες ροές απλά προστίθενται στις φυσιολογικές. Ωστόσο θα πρέπει να εξασφαλίσουμε ότι έχουμε δώσει αρκετό μέγεθος στις πιθανοτικές δομές ώστε να επιτυγχάνεται ικανοποιητική απόδοση ακόμα και σε περιπτώσεις αυξημένης κίνησης.

4.5 Ανίχνευση Μοναδικών Ροών

Πρόκειται για τον πρώτο από τους τρεις επιμέρους μηχανισμούς που είναι υπεύθυνοι για την ανίχνευση επιθέσεων. Αυτός χρησιμοποιώντας πιθανοτικές δομές που μοιάζουν με bloom-filters και sketches, ανιχνεύει το πλήθος των μοναδικώς εισερχόμενων ροών κάθε εποχή για τα δίκτυα που μας ενδιαφέρουν και συγκρίνει το πλήθος αυτό με κατάλληλο κατώφλι ώστε να ανιχνεύσει ανωμαλία στον αριθμό των ροών από εποχή σε εποχή. Το κατώφλι είναι μεταβλητό για κάθε εποχή και υπολογίζεται από τον χρονικό μέσο και διακύμανση των ροών ανά εποχή, οι οποίοι προκύπτουν με τις τεχνικές EWMA και EWMD παρόμοια με το [15].

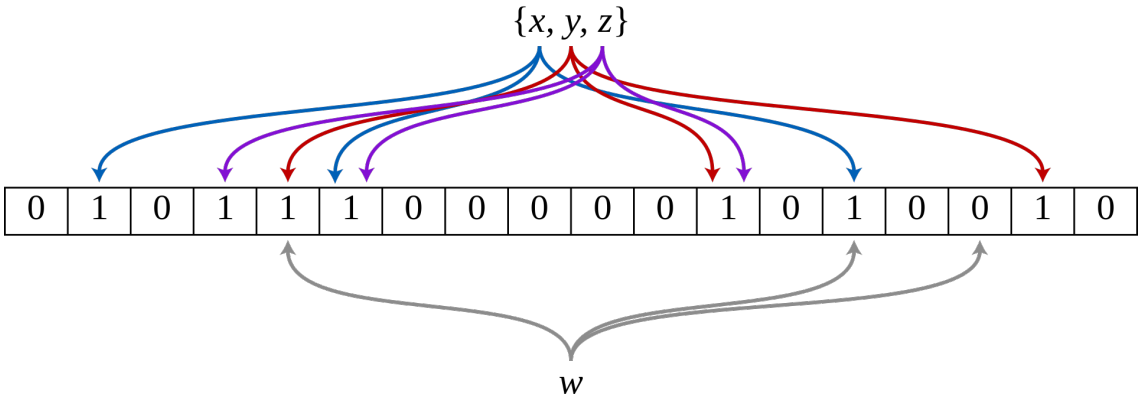


***Εικόνα 4.3**: Λογικό διάγραμμα Μηχανισμού Ανάλυσης Ροών*

Στην αρχή ελέγχουμε αν πρόκειται για πιθανή εισερχόμενη ροή, δηλαδή αν το πακέτο έχει σαν δίκτυο προορισμού κάποιο από τα ελεγχόμενα υποδίκτυα και ταυτόχρονα δεν πηγάζει από κάποιο από αυτά. Αυτό γίνεται καθώς δεν μας ενδιαφέρουν οι εξερχόμενες ροές από τα υποδίκτυα μιας και ο μηχανισμός ανιχνεύει επιθέσεις από το διαδίκτυο. Σε περίπτωση λοιπόν που πρόκειται για εξερχόμενη ροή από κάποιο από τα ελεγχόμενα υποδίκτυα δεν προσπερνάμε τον μηχανισμό αυτό και συνεχίζουμε στον μηχανισμό ασυμμετρίας.
Αντίθετα εφόσον πρόκειται για εισερχόμενη ροή, ελέγχουμε αν πρόκειται για καινούρια με την βοήθεια κατάλληλων πιθανοτικών δομών. Κρίνεται σκόπιμο να κάνουμε αρχικά μια αναφορά στα bloom-filters και στα sketches που αποτελούν βάση για το κομμάτι αυτό.

Bloom-Filters

Τα bloom-filters [20, 21], όπως έχουμε αναφέρει, αποτελούν πιθανοτικές δομές δεδομένων με σκοπό τον έλεγχο για την ύπαρξη ενός στοιχείου σε ένα σύνολο χρησιμοποιώντας σταθερό πλήθος θέσεων μνήμης. Αποτελούνται από έναν πίνακα με m bits και από ένα σύνολο k συναρτήσεων κατακερματισμού (hash functions) κάθε μία από τις οποίες αντιστοιχίζει στοιχεία σε κάποια από τις m θέσεις του πίνακα. Για την εισαγωγή ενός στοιχείου στο bloom-filter παρέχουμε το στοιχείο σαν όρισμα στις συναρτήσεις κατακερματισμού και θέτουμε τιμή 1 στις θέσεις που μας υποδεικνύουν. Για να ελέγξουμε αν ένα στοιχείο υπάρχει ήδη, το περνάμε από τις συναρτήσεις και ελέγχουμε τις αντίστοιχες θέσεις του πίνακα. Αν έστω και μία από αυτές έχει την τιμή 0 τότε το στοιχείο σίγουρα δεν υπάρχει στο bloom-filter, δηλαδή πρόκειται για καινούριο στοιχείο. Αντίθετα αν και οι τρεις θέσεις έχουν τιμή 1 τότε **πιθανώς** το στοιχείο αυτό υπάρχει ήδη στο σύνολο του bloom-filter.



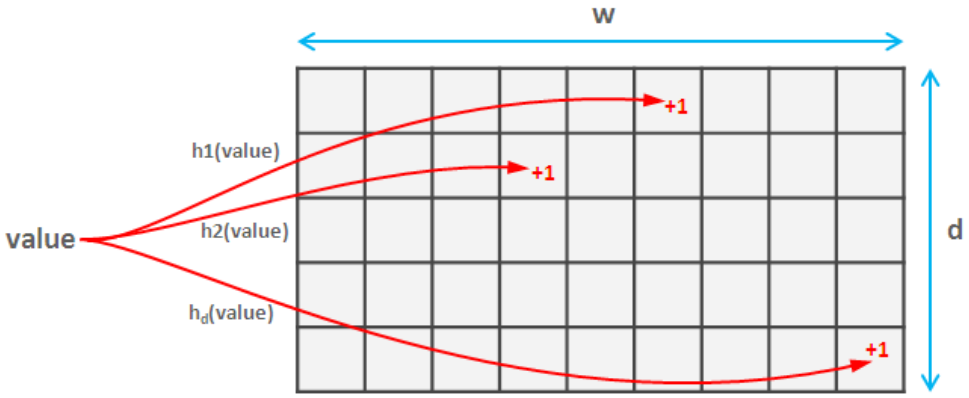
***Εικόνα 4.4**: Παράδειγμα Bloom Filter. [19]*
Το w δεν ανήκει στο σύνολο μιας και αντιστοιχεί σε θέση μνήμης με τιμή 0.

Η πιθανότητα λάθους εκτίμησης συνδέεται άμεσα με το μέγεθος m του πίνακα, το πλήθος n των εισαχθέντων στοιχείων στο bloom-filter και το πλήθος k των συναρτήσεων κατακερματισμού. Συγκεκριμένα για σταθερά m και k, και πλήθος n ήδη εισαχθέντων στοιχείων η πιθανότητα λανθασμένης ανίχνευσης ενός στοιχείου εκτός συνόλου προσεγγίζεται από τον τύπο $(1-e^{(-kn/m)})^k$, από τον οποίο μπορούμε να συμπεράνουμε ότι αυξάνοντας το μέγεθος του bloom-filter μπορούμε να μικρύνουμε την πιθανότητα λάθους, ενώ αυξάνοντας τα στοιχεία που υπάρχουν σε αυτό, η πιθανότητα μεγαλώνει αντίστοιχα.

Sketches

Τα sketches [22, 23, 24] αποτελούν πιθανοτικές δομές στοχεύοντας κατά κύριο λόγο στο στην εύρεση της συχνότητας του κάθε στοιχείου στο σύνολο. Συνήθως αναφέρονται και ως count-min sketches. Αποτελούνται από έναν δισδιάστατο πίνακα w στηλών και d γραμμών. Σε κάθε γραμμή αντιστοιχεί μια συνάρτηση κατακερματισμού. Υπό μια έννοια πρόκειται για d διακριτά bloom-filters με μια συνάρτηση κατακερματισμού στο κάθε ένα. Ωστόσο δεν έχουν την ίδια λειτουργία. Κατά την εισαγωγή του, κάθε στοιχείο περνάει από τις d διαφορετικές hashes για να βρεθεί η θέση του σε κάθε γραμμή του πίνακα, και στην συνέχεια αυτές οι θέσεις αυξάνονται κατά 1. Για την εύρεση της συχνότητας του στοιχείου, ελέγχονται οι τιμές των ίδιων θέσεων και ως εκτίμηση συχνότητας κρατείται η μικρότερη από αυτές.

Η εκτίμηση συχνότητας που επιστρέφουν τα count-min sketches είναι πάντα μεγαλύτερη ή ίση με την πραγματική συχνότητα εμφάνισης των στοιχείων στο σύνολο. Οι παράμετροι w και d επηρεάζουν την απόδοση αυτής της εκτίμησης, και μπορούν να ρυθμιστούν κατάλληλα ώστε να επιτευχθούν αποδεχτές πιθανότητες για το εύρος λάθους. Συγκεκριμένα θέτοντας $w=\lceil e/\varepsilon \rceil$ και $d=\lceil \ln(1/\delta) \rceil$, η εκτίμηση συχνότητας f' και η πραγματική συχνότητα f θα έχουν την σχέση $f' \leq f+\varepsilon N$ με πιθανότητα 1-δ , όπου N το μέγεθος του συνόλου.



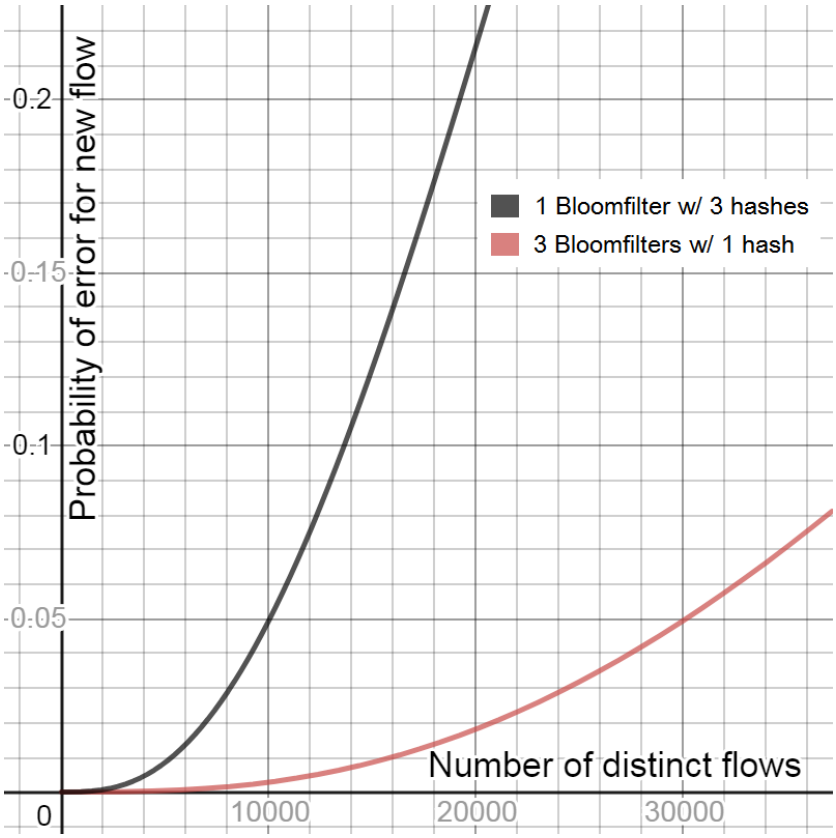
***Εικόνα 4.5**: Παράδειγμα Count Min Sketch [21]*

Επιστρέφοντας στον μηχανισμό μας, η δομή που χρησιμοποιούμε μοιάζει με sketch αλλά λειτουργεί ως bloom-filter. Πρακτικά υπάρχουν 3 πίνακες από registers (**sketch_1**,**sketch_2**, **sketch_3**) σε κάθε έναν από τους οποίους αντιστοιχεί μια συνάρτηση κατακερματισμού (**crc32**, **crc16**, **csum16** αντίστοιχα). Ο λόγος που έγινε αυτή η επιλογή είναι η επιθυμία να μεγιστοποιήσουμε των αριθμό των διαφορετικών ροών που μπορούν να ανιχνευθούν, με ικανοποιητική πιθανότητα. Αρχικά η υλοποίησή μας περιορίζεται από την χρήση μόνο των τριών συναρτήσεων κατακερματισμού που αναφέραμε παραπάνω, μιας και αυτές υποστηρίζονται ταυτόχρονα από τις δύο πλατφόρμες bmv2 και Netronome SmartNIC, σκεπτόμενοι πρακτικά τους ίδιους λόγους μεταφερσιμότητας όπως και στην περίπτωση του ternary matching στα tables. Επιπλέον οι δύο συναρτήσεις (**crc16** και **csum16**) έχουν περιορισμένο σύνολο τιμών (65536) και μέχρι τόσες διευθύνσεις μπορούμε να αποκτήσουμε με αυτές. Αν υλοποιούσαμε ένα μόνο bloom-filter με τρεις συναρτήσεις κατακερματισμού θα χρησιμοποιούσαμε μεν το ένα τρίτο από την μνήμη, αλλά θα μπορούσαμε να ανιχνεύσουμε αρκετά λιγότερες μοναδικές ροές, όπως φαίνεται και στο διάγραμμα της εικόνας 4.6.

Πίνακας 4.3: Υλοποίηση πιθανοτικής δομής και συνάρτηση υπολογισμού hash των ροών

```
sketch_1.read(lep, meta.hash1);
if (lep != gep) {
    new_flow = 1;
    sketch_1.write(meta.hash1, gep);
}
sketch_2.read(lep, meta.hash2);
if (lep != gep) {
    new_flow = 1;
    sketch_2.write(meta.hash2, gep);
}
sketch_3.read(lep, meta.hash3);
if (lep != gep) {
    new_flow = 1;
    sketch_3.write(meta.hash3, gep);
}
```

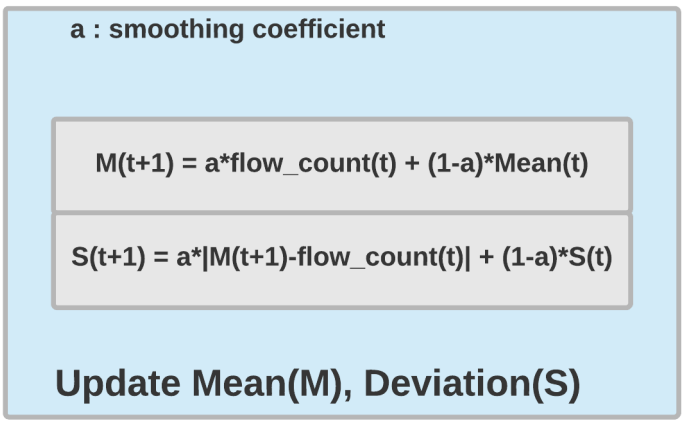
```
action get_hashes(){
        hash(meta.hash1, HashAlgorithm.crc32,
(bit<32>) 0,
{ hdr.ipv4.srcAddr,hdr.ipv4.dstAddr,hdr.ipv4.
protocol,hdr.ports.srcPort,hdr.ports.dstPort },
(bit<32>)65536);
        hash(meta.hash2, HashAlgorithm.crc16,
(bit<32>) 0,
{ hdr.ipv4.srcAddr,hdr.ipv4.dstAddr,hdr.ipv4.
protocol,hdr.ports.srcPort,hdr.ports.dstPort },
(bit<32>)65536);
        hash(meta.hash3,
HashAlgorithm.csum16, (bit<32>) 0,
{ hdr.ipv4.srcAddr,hdr.ipv4.dstAddr,hdr.ipv4.
protocol,hdr.ports.srcPort,hdr.ports.dstPort },
(bit<32>)65536);
    }
```



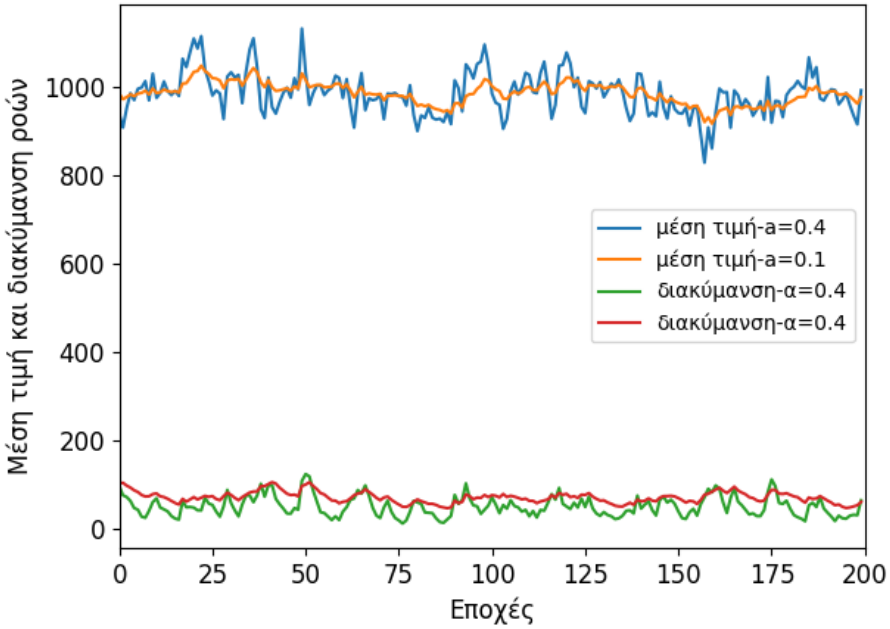
***Εικόνα 4.6**: Γράφημα πιθανότητας λάθους για 1 bloom filter με 3 hashes και 3 bloom filters με 1 hash*

Μέσω της συνάρτησης **get_hashes()** περνάμε την ροή από τις συναρτήσεις κατακερματισμού βρίσκοντας τις θέσεις της στους πίνακες, και αποθηκεύουμε τις τιμές αυτές σαν μεταδεδομένα (**meta.hash1**, **meta.hash2**, **meta.hash3**). Στην συνέχεια ελέγχουμε τους πίνακες. Εδώ είναι μια επιπλέον αλλαγή σε σχέση με τα συμβατικά bloom-filters. Σε αυτά, η κάθε θέση έχει δύο δυνατές τιμές 0 και 1. Εδώ όμως εισάγουμε τον αριθμό της εποχής που προσπελάστηκε τελευταία φορά η κάθε θέση. Ο λόγος που γίνεται αυτό είναι πως δεν υπάρχει η δυνατότητα για μαζικό μηδενισμό όλων των θέσεων των registers. Έτσι κατά τον έλεγχο συγκρίνουμε την τωρινή εποχή με την εποχή που περιέχει η κάθε θέση. Αν είναι ίδιες τότε θεωρούμε πως βρίκαμε 1 στον πίνακα, αν όχι 0. Έτσι αν κάποια από τις 3 θέσεις έχει διαφορετική εποχή από την τωρινή, την ενημερώνουμε και θεωρούμε ότι πρόκειται για καινούρια ροή.

Εφόσον βρήκαμε καινούρια ροή αυξάνουμε τον μετρητή ροών **flow_counter**, και ελέγχουμε αν οι τωρινές ροές υπερβαίνουν την μέση τιμή συν έναν παράγοντα k επί την διακύμανση. Η τεχνική είναι παρόμοια με το [15] και βασίζεται στις EWMA και EWMD. Η μέση τιμή **flow_m** και η διακύμανση **flow_s** ενημερώνονται στην αρχή κάθε εποχής με βάση τους τύπους που φαίνονται στην εικόνα 4.7. Ο όρος a ονομάζεται συντελεστής εξομάλυνσης. Μεγάλη τιμή του a έχει ως συνέπεια να δίνεται βάρος στην καινούρια μέτρηση και μέσω αυτής να προκύπτουν η νέα μέση τιμή και διακύμανση. Πρακτικά με μεγάλο a, οι τιμές αυτές ακολουθούν πιστά τις μεταβολές της κίνησης, με την καμπύλη που θα σχημάτιζαν να παρουσιάζει απότομες μεταβολές και αρκετά τοπικά μέγιστα και ελάχιστα. Αντίθετα με μικρό a δίνουμε λιγότερη έμφαση στις καινούριες μετρήσεις. Έτσι τυχόν αποκλίσεις της κανονικής κίνησης δεν επηρεάζουν σημαντικά αυτές τις τιμές, εξομαλύνοντας με αυτόν τον τρόπο τις τιμές αυτών των μετρικών κατά την πάροδο των εποχών. Από την άλλη ο παράγοντας k ορίζει ένα διάστημα αποδεκτών αποκλίσεων από την μέση τιμή της κίνησης που δεν θα θεωρηθούν αφύσικες. Αν το σύνολο των τωρινών ροών ξεπεράσει το όριο αυτό, τότε ο μηχανισμός ενεργοποιεί την σημαία **meta.flow_count_flag** και την αποθηκεύει σαν μεταδεδομένα του πακέτου.



***Εικόνα 4.7**: Σχέσεις EWMA, EWMD*



Εικόνα 4.8: *Παράδειγμα τεχνικών EWMA EWMD για α=0.1 και α=0.4 Οι τιμές παράχθηκαν τυχαία με βάση κανονική κατανομή μέσης τιμής 1000 και διασποράς 100. Βλέπουμε ότι με μεγαλύτερο α έχουμε απότομες μεταβολές. Αντίθετα μικρότερο α έχουμε πιο ομαλή καμπύλη.*

Για τον υπολογισμό των παραπάνω τιμών βλέπουμε ότι χρησιμοποιούνται πράξεις όπως πολλαπλασιασμός και shifting. Αυτό προκύπτει από το γεγονός πως δεν υποστηρίζεται από το $P4_{16}$ η πράξη της διαίρεσης και αριθμοί κινητής υποδιαστολής. Για να λυθεί αυτό θεωρούμε fixed point αναπαράσταση για τους αριθμούς a, k, δηλαδή πως τα τελευταία bits των αριθμών αυτών αντιπροσωπεύουν κλάσματα δυνάμεων του 2. Συγκεκριμένα για το a χρησιμοποιούμε 8 bits ως αναπαράσταση κλασμάτων και για το k 4 bits. Αυτό σημαίνει πως το λιγότερης σημασίας bit για το a αντιστοιχεί στην τιμή 2^{-8} και για το κ στην τιμή 2^{-4}. Πολλαπλασιάζοντας τώρα με αυτές τις τιμές και μετατοπίζοντας κατά τις αντίστοιχες θέσεις δεξιά (right shift) το αποτέλεσμα, προσομοιώνουμε πολλαπλασιασμό με δεκαδικές τιμές.

Η σημασία αυτού του μηχανισμού είναι μεγάλη. Όπως είδαμε πολλά συστήματα [13,15] χρησιμοποιούν με τον έναν ή τον άλλον τρόπο την μέτρηση των μοναδικών ροών της κίνησης για να βγάλουν συμπεράσματα για το είδος αυτής. Ιδιαίτερα για τις επιθέσεις DDoS αποτελεί μια καλή μετρική μιας και κατά την επίδρασή τους εμφανίζεται καινούρια κίνηση από μεγάλο πλήθος μηχανημάτων. Ιδιαίτερα αν η επίθεση βασίζεται σε μεγάλο δίκτυο botnet τότε ο αριθμός ροών εκτοξεύεται.

Ωστόσο η διακύμανση των συνολικών ροών δεν αποτελεί πάντα σίγουρη ένδειξη για επίθεση. Κατά την λειτουργία του δικτύου υπάρχουν μοτίβα κίνησης που μεταβάλλονται. Για παράδειγμα είναι λογικό τις βραδυνές ώρες η κίνηση ενός εταιρικού δικτύου να παρουσιάζεται αρκετά μειωμένη, και να αυξάνεται κατά τις ώρες αιχμής. Επίσης πάντα είναι πιθανή η εμφάνιση μικρών αιχμών κίνησης. Τέτοιες καταστάσεις θα μπορούσαν να

ενεργοποιήσουν έναν μηχανισμό που λαμβάνει υπόψη του μόνο τις ροές. Για τον λόγο αυτό χρησιμοποιούμε και τους επόμενους επιμέρους μηχανισμούς, ώστε να αυξήσουμε το ποσοστό σιγουριάς σε περίπτωση ανίχνευσης πιθανής επίθεσης, αλλά και να υποδείξουμε το υποδίκτυο που αποτελεί στόχο αυτής.

Πίνακας 4.4: Ενημέρωση μέσης τιμής και διακύμανσης

```
flow_s.read(s,0);
flow_m.read(m,0);
kappa.read(k,0);
alpha.read(a,0);
m = ((a*flow) + ((256 - a)*m))>>8;
if (m < flow){
   s = (a*(flow-m)) + ((256 - a)*s);
   s = s >> 8;
}
else {
   s = (a*(m-flow)) + ((256 - a)*s);
   s = s >> 8
}
flow_s.write(0,s);
flow_m.write(0,m);
```

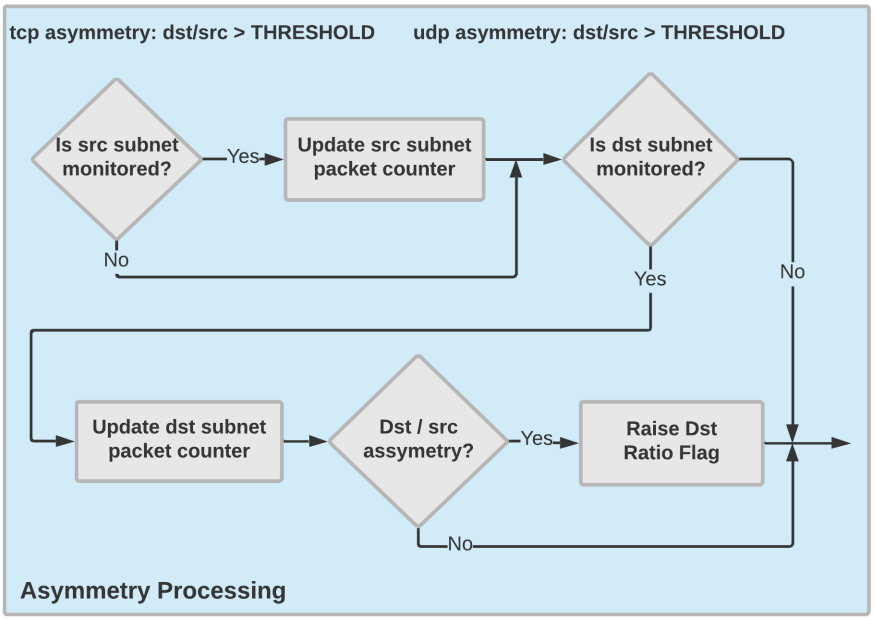
4.6 Ροές ανά δίκτυο

Ο δεύτερος μηχανισμός χρησιμοποιεί και πάλι την έννοια των ροών, αλλά προσπαθεί να προσδώσει μια πιο στοχευμένη οπτική ελέγχοντας τις ροές ανά υποδίκτυο. Με την ανίχνευση της ροής ως μοναδική αυξάνουμε κατά 1 και το πλήθος των ροών για το συγκεκριμένο υποδίκτυο. Τις τιμές αυτές τις αποθηκεύουμε στον πίνακα (register) **flow_dst**, και για την θέση χρησιμοποιούμε την τιμή **meta.d_subnet**, δηλαδή τον δείκτη που μας είχε επιστρέψει η αντιστοίχιση με το table **subnet_dst**. Και πάλι λόγω του ότι δεν είναι δυνατός ο μαζικός μηδενισμός του πίνακα από εποχή σε εποχή, πρέπει να βρούμε τρόπο να ελέγχουμε ότι δεν περιέχονται παλαιότερες τιμές στον πίνακα. Γι' αυτό χρησιμοποιούμε τον πίνακα **flow_epoch_dest** στον οποίο αποθηκεύουμε την τελευταία εποχή που γράφτηκε η αντίστοιχη θέση μνήμης του **flow_dst**. Έτσι μπορούμε να καταλάβουμε αν πρόκειται για σύγχρονα δεδομένα ή παλιά που πρέπει να μηδενιστούν.

Έπειτα γίνεται έλεγχος με ένα κατώφλι που ορίζεται με βάση τις συνολικές τωρινές ροές και έναν παράγοντα **factor** σχετικό με το ποσοστό ροών για τα υποδίκτυα. Ο παράγοντας αυτός προκύπτει από πρότερη μελέτη της κίνησης και αντιστοιχεί σε ένα ποσοστό ορίου κίνησης για τα υποδίκτυα. Εφόσον οι ροές υπερβούν αυτό το όριο, τότε θεωρούμε πως το συγκεκριμένο υποδίκτυο δέχεται σημαντικό μέρος της συνολικής κίνησης και επομένως αποτελεί έναν πιθανό στόχο επίθεσης. Με αυτόν τον τρόπο ο συνολικός μηχανισμός παρέχει μια πιο λεπτομερή εικόνα κατά την ενημέρωση περί ανίχνευσης επίθεσης, καθώς ασχολείται με υποδίκτυα που παρουσιάζουν σημαντική κίνηση, η οποία μπορεί να έχει προκληθεί από κάποια επίθεση.

4.7 Ασυμμετρία κίνησης

Η ανίχνευση ασυμμετρίας κίνησης αποτελεί την τρίτη λειτουργία του προτεινόμενου μηχανισμού. Σε αντίθεση με τους προηγούμενους 2 βασίζεται μόνο στις διευθύνσεις IP πηγής και προορισμού για να αναγνωρίσει ασυμμετρία στην κίνηση ανά υποδίκτυο και ανά πρωτόκολλο (TCP / UDP). Με βάση αυτές ενημερώνονται αρχικά οι αντίστοιχοι μετρητές και στην συνέχεια, εφόσον πρόκειται για πακέτο με διεύθυνση προορισμού κάποιο από τα ελεγχόμενα υποδίκτυα, ελέγχεται η συνθήκη ασυμμετρίας για το συγκεκριμένο υποδίκτυο και πρωτόκολλο. Τέλος, αν χρειάζεται σηκώνεται η τρίτη σημαία.



***Εικόνα 4.9**: Λογικό Διάγραμμα Ανίχνευσης Ασυμμετρίας*

Ο πίνακας (register) που αποθηκεύει τις μετρήσεις των πακέτων ονομάζεται **stats**. Αυτός κρατάει τις μετρήσεις και για τα δύο πρωτόκολλα αλλά και για τα πακέτα εισόδου και εξόδου των δικτύων. Συγκεκριμένα στο πρώτο τέταρτο του πίνακα αποθηκεύονται τα δεδομένα εισόδου των δικτύων για το πρωτόκολλο TCP. Στο δεύτερο τέταρτο, αποθηκεύονται τα δεδομένα εξόδου για το TCP, ενώ στο τρίτο και τέταρτο τέταρτο αποθηκεύονται τα δεδομένα του πρωτοκόλλου UDP, εισόδου και εξόδου αντίστοιχα. Ο λόγος που χρησιμοποιούμε μια δομή για να αποθηκεύσουμε όλα τα στοιχεία είναι η εξοικονόμηση εντολών του προγράμματος. Στο μυαλό ενός προγραμματιστή θα ήταν λογικό να περιγράψουμε την διαδικασία με μια συνάρτηση, η οποία ανάλογα την κατάσταση θα επιλέγει και θα γράφει στον κατάλληλο πίνακα. Ωστόσο μιας και δεν υποστηρίζεται (τόσο στο bmv2 όσο και στην κάρτα Netronome) η υπό συνθήκη εκτέλεση εντολών, και κατ' επέκταση πρόσβαση στην μνήμη, μέσα στο σώμα ενός action, αναγκαζόμαστε να χρησιμοποιήσουμε έναν ενιαίο πίνακα.

Και εδώ χρησιμοποιούμε έναν βοηθητικό πίνακα **epoch** για να ελέγξουμε ότι ο **stats** δεν περιέχει παλιές μετρίσεις. Συγκεκριμένα ο **epoch** διατηρεί για κάθε υποδίκτυο την τελευταία

εποχή που συναντήσαμε πακέτο και επομένως αυξήσαμε τον αντίστοιχο μετρητή του **stats**. Συγκρίνοντας την τιμή αυτή με την τωρινή εποχή μπορούμε να συμπεράνουμε αν απλά χρειάζεται προσαύξηση του μετρητή των πακέτων, ή και μηδενισμός του.

Θα μπορούσαμε για κάθε δίκτυο να κρατούμε περισσότερες από μια τελευταίες εποχές ωστε να αποκτήσουμε μια καλύτερη αξιολόγηση της συμμετρίας τις κίνησης. Ωστόσο για την διαχείριση και την ενημέρωσή τους χρειάζεται να τις περιγράψουμε με εμφωλευμένα if-else statements, μιας και δεν υποστηρίζονται επαναληπτικές δομές από το P4. Αυτό όμως αυξάνει σημαντικά τις γραμμές κώδικα του προγράμματος, την πολυπλοκότητα και τις προσπελάσεις στην μνήμη σε σημείο που κρίνεται πως δεν έχουμε πρακτικό όφελος από την χρήση του.

Πίνακας 4.5: Ενημέρωση μνήμης για εισερχόμενο πακέτο

```
proto = (bit<32>)hdr.ipv4.protocol & 1;
if (meta.s_subnet < 256){
   meta.s_eff_addr = (proto << 11) + meta.s_subnet;
   ep_addr = (proto << 9) + meta.s_subnet;
   epoch.read(lep, ep_addr);
   if(gep != lep){
      stats.write(meta.s_eff_addr,0);
      }
      epoch.write(ep_addr, gep);
}
stats.read(pc, meta.s_eff_addr);
stats.write(meta.s_eff_addr, pc+1);

if (meta.d_subnet < 256){
   meta.d_eff_addr = (proto << 11) +  1024 + meta.d_subnet;
   ep_addr = (proto << 9) + 256 +  meta.d_subnet;
   epoch.read(lep, ep_addr);
   if(gep != lep){
      stats.write(meta.d_eff_addr,0);
      }
      epoch.write(ep_addr, gep);
}
stats.read(pc, meta.d_eff_addr);
pc = pc + 1
stats.write(meta.d_eff_addr, pc);
```

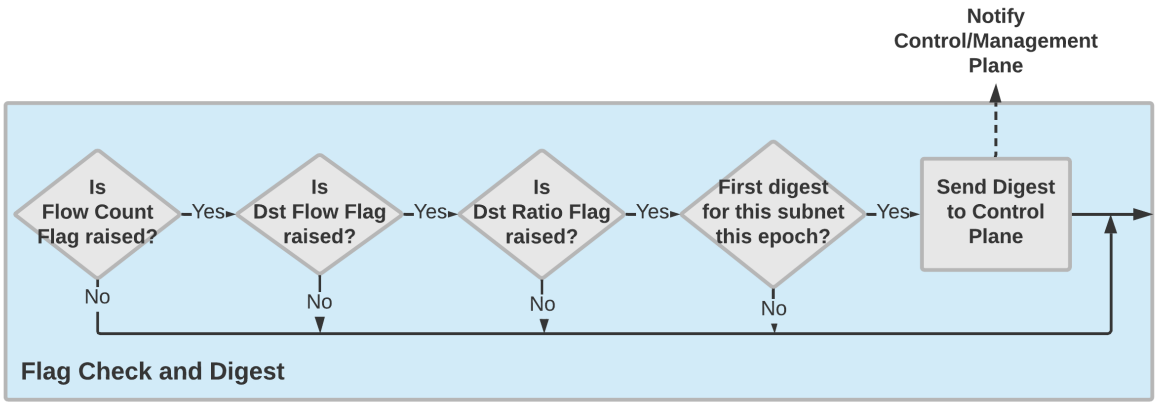
Τέλος εφόσον το πακέτο προορίζεται για κάποιο από τα ελεγχόμενα υποδίκτυα, πραγματοποιούμε και τον αντίστοιχο έλεγχο περί ασυμμετρίας. Ο λόγος που γίνεται μόνο σε αυτά τα πακέτα, είναι πως μόνο τέτοια πακέτα μπορούν να οδηγήσουν σε υπέρβαση του ορίου του μηχανισμού. Αν υπάρξει τέτοια υπέρβαση ενεργοποιείται η τρίτη σημαία **meta.dst_ratio_flag**.

Η ασυμμετρία κίνησης αποτελεί μια βασική μετρική [25,26] για την ανίχνευση ανωμαλιών στην δικτυακή κίνηση και αποσκοπεί στην βελτίωση της απόδοσης του παρόντος μηχανισμού. Με βάση αυτήν μπορούμε να παρέχουμε έναν διαχωρισμό για την περίπτωση αφύσικης και πιθανότατα κακόβουλης κίνησης, και για την περίπτωση της εμφάνισης αυξημένων ροών κανονικής κίνησης. Τα όρια ασυμμετρίας αποτελούν έναν σημαντικό παράγοντα για την ορθή απόδοση αυτής, και ενώ για το πρωτόκολλο TCP φαίνεται να υπάρχει ένα παρατηρούμενο όριο (4.5 : 1 σύμφωνα με το [25]), για το UDP δεν ισχύει το ίδιο, λόγω και της διαφοράς λειτουργίας των δύο πρωτοκόλλων. Στο πλαίσιο της διπλωματικής τα όρια και για τα δύο πρωτόκολλα υπολογίζονται με βάση πρότερη ανάλυση της κανονικής κίνησης. Σε κάθε υποδίκτυο αντιστοιχεί ένας δείκτης ασυμμετρίας ο οποίος συγκρίνεται με την τωρινή παρατηρούμενη. Εφόσον αυτοί διαφέρουν κατά ένα ποσοστό, υπολογισμένο πάλι με βάση την φυσιολογική κίνηση, τότε θεωρούμε πως υπάρχει ασυμμετρία στο συγκεκριμένο υποδίκτυο, και ενεργοποιούμε την σημαία. Παράλληλα η ασυμμετρία παρέχει πληροφορία και για το πλήθος των πακέτων της κίνησης, η οποία δεν παρέχεται στις προηγούμενες μετρικές. Επομένως προσθέτοντάς την στον μηχανισμό μπορούμε να ανιχνεύσουμε και επιθέσεις που σχετίζονται με μεγαλύτερο πλήθος πακέτων και δημιουργούν λιγότερες ροές.

Πίνακας 4.6: Έλεγχος ασυμμετρίας

```
if(proto == 0) {
   asymmetry_factor_tcp.read(asymmetry_factor, meta.d_subnet);
   dst = dst*asymmetry_factor;
}
else {
   asymmetry_factor_udp.read(asymmetry_factor, meta.d_subnet);
   dst = dst*asymmetry_factor;
}
src = src + 1;
dst = dst + 1;
if (20*src < 10*dst) meta.dst_ratio_flag = 1;
```

4.8 Έλεγχος Σημαιών και Δημιουργία Digest



***Εικόνα 4.10**: Λογικό Διάγραμμα Ελέγχου Σημαιών*

Τελευταίο στάδιο του μηχανισμού είναι ο έλεγχος των σημαιών. Εφόσον και οι τρεις επιμέρους μηχανισμοί έχουν θέσει τις αντίστοιχες σημαίες και ταυτόχρονα δεν έχει δημιουργηθεί πάλι ειδοποίηση για το συγκεκριμένο υποδίκτυο την συγκεκριμένη εποχή, δημιουργείται μήνυμα ειδοποίησης (digest) και στέλνεται στο επίπεδο ελέγχου και σε επόμενους μηχανισμούς για περαιτέρω ανάλυση.

Οι σημαίες όπως έχουμε αναφέρει είναι αποθηκευμένες σαν μεταδεδομένα. Επιπλέον ο πίνακας **digest_ckeck** περιέχει την τελευταία εποχή που έγινε αποστολή digest για κάθε υποδίκτυο. Ο συγκεκριμένος πίνακας υπάρχει για να μην παράγεται υπερβολικός αριθμός ειδοποιήσεων, ειδικά σε περίπτωση επίθεσης. Το μήνυμα digest μεταφέρει πληροφορία για το δίκτυο που δημιούργησε το μήνυμα και πιθανόν δέχεται επίθεση, για το πρωτόκολλο που περιείχε το πακέτο και για την εποχή που παράχθηκε.

Πίνακας 4.7: Έλεγχος και δημιουργία digest

```
digest_check.read(df,meta.d_subnet);
if(meta.flow_count_flag == 1 && meta.flow_dst_flag == 1 && meta.dst_ratio_flag ==1
   && df < gep){
   digest_check.write(meta.d_subnet, gep);
   digest((bit<32>)1024, { (ip4Addr_t)hdr.ipv4.dstAddr, (bit<32>)meta.d_subnet,
                           (bit<8>)hdr.ipv4.protocol, (bit<32>)gep });
}
```

Εξίσου σημαντικό στοιχείο του μηχανισμού είναι και η δυνατότητα να δημιουργεί από μόνο του ειδοποιήσεις και να τις προωθεί σε ανώτερα επίπεδα. Το γεγονός πως δεν χρειάζεται το επίπεδο ελέγχου να ζητά συνεχώς πληροφορίες μειώνει την σημαντικά την χρήση πόρων των

δύο συσκευών, ενώ παράλληλα το επίπεδο ελέγχου με αυτές τις ειδοποιήσεις μπορεί έγκαιρα να ενημερώσει επιπλέον μηχανισμούς ώστε να παρθούν δράσεις για την αντιμετώπιση της επίθεσης το γρηγορότερο δυνατόν.

4.9 Προώθηση Πακέτων

Οφείλουμε να αναφέρουμε και τον τρόπο προώθησης πακέτων, καθώς αποτελεί τον πρώτο και κυριότερο στόχο του επιπέδου δεδομένων. Είτε πρόκειται για πακέτο που θα επεξεργαστεί ο μηχανισμός, είτε όχι, η προώθηση στον συγκεκριμένο μηχανισμό γίνεται με βάση την συνάρτηση l2_forward(). Μέσω της συνάρτησης αυτής θέτουμε τον αριθμό της επιθυμητής πόρτας εξόδου στο πεδίο **egress_spec** των μεταδεδομένων **standard_metadata** που παρέχονται από την αρχιτεκτονική των συσκευών. Στην περίπτωση της παρούσας διπλωματικής ο μηχανισμός ανίχνευσης είναι ανεξάρτητος από αυτόν της προώθησης. Έτσι θεωρούμε πως η συσκευή αποτελείται από 2 πόρτες, οπότε η λειτουργία προώθησης απλουστεύεται στην προώθηση του πακέτου από την αντίθετη πόρτα από την εισερχόμενη.

Πίνακας 4.8: Συνάρτηση προώθησης πακέτου

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

Κεφάλαιο 5 - Πειραματικά Αποτελέσματα

5.1 Πλατφόρμες ανάπτυξης

5.1.1 Bmv2 Software Switch

Αν και έχει επικρατήσει η ονομασία Bmv2 Software Switch, το Bmv2 ή ολογράφως Behavioral Model Version 2, αποτελεί πρακτικά ένα framework γραμμένο στην γλώσσα προγραμματισμού C++11, πάνω στο οποίο μπορεί κάποιος να ορίσει ένα εικονικό switch με δεδομένη αρχιτεκτονική και να μέσω αυτού να αναπτύξει και να ελέγξει προγράμματα γραμμένα σε P4. Το Bmv2 παρέχει παράλληλα μια βασική υλοποίηση ενός τέτοιου εικονικού switch (target) που ονομάζεται **simple_switch** [27] και στηρίζεται στην βασική αρχιτεκτονική του P4 (v1model - εικόνα 2.4). Συνήθως όταν γίνεται αναφορά στο Bmv2 εννοείται η υλοποίηση **simple_switch** ή κάποια τροποποίηση αυτής. Τονίζεται ότι το Bmv2 αποτελεί περιβάλλον ανάπτυξης και όχι εικονική συσκευή υψηλής απόδοσης.

Για την ανάπτυξη P4 προγραμμάτων στο Bmv2 απαιτείται η μετατροπή τους από P4 σε πρότυπο JSON μέσω του μεταγλωττιστή **p4c** [28]. Το παραγόμενο JSON αρχείο χρησιμοποιείται στην συνέχεια ως όρισμα για την εκκίνηση του εικονικού switch. Παράλληλα για την διαχείριση του switch και την ενημέρωση των δομών του P4 προγράμματος κατά την λειτουργία του, χρησιμοποιείται το πρόγραμμα-ελεγκτής **simple_switch_CLI** [29]. Πρόκειται για τον βασικό ελεγκτή που παρέχει το Bmv2 για την επικοινωνία με το **simple_switch**. Είναι γραμμένος σε γλώσσα Python 2 και μέσω αυτού εκτελούνται εντολές που έχουν να κάνουν με την συλλογή στατιστικών από το switch και την ενημέρωση πινάκων και δομών. Χρησιμοποιώντας το **simple_switch_CLI** και ένα αρχείο κειμένου με εντολές προς αυτό μπορούμε να προωθήσουμε μαζικά ρυθμίσεις στο switch, όπως για παράδειγμα τις αρχικές ρυθμίσεις λειτουργίας του που περιέχουν τους κανόνες των tables και τις αρχικές τιμές των registers. Ωστόσο δεν μπορούμε μέσω αυτού να διαβάσουμε τα digests που δημιουργεί το switch κατά την λειτουργία του. Γι' αυτό τον σκοπό δημιουργήθηκε από εμάς ένα πρόγραμμα Python, το οποίο συνδέεται στο UNIX socket που ανοίγει το Bmv2 και διαβάζει τα digests.

5.1.2 Netronome SmartNIC

Οι Netronome SmartNICs [19] αποτελούν κάρτες δικτύου υψηλών αποδόσεων που υποστηρίζουν ένα μεγάλο εύρος ενεργειών από κλασσική δικτύωση μέχρι προγραμματιζόμενες διαδικασίες. Βασίζονται σε ειδικούς επεξεργαστές Netronome Flow Processors (NFP) και σε κατάλληλο λογισμικό ώστε να πετυχαίνουν μεγάλες ταχύτητες προώθησης αλλά και ευελιξία σε ανάπτυξη και εφαρμογή νέων λειτουργιών.

Για τον προγραμματισμό των καρτών αυτών παρέχεται το εργαλείο NFP Software Development Kit (NFP-SDK). Αυτό περιλαμβάνει ένα σύνολο λειτουργιών που είναι υπεύθυνα για την ανάπτυξη των επιθυμητών προγραμμάτων, την επικοινωνία της κάρτας με

το φυσικό ή εικονικό μηχάνημα στο οποίο συνδέεται αλλά και την διαχείριση και τον έλεγχό της από τον προγραμματιστή.

Για την υλοποίηση προγραμμάτων παρέχεται το περιβάλλον ανάπτυξης **Programmer Studio**. Μέσω αυτού είναι δυνατή η ανάπτυξη προγραμμάτων με βάση την γλώσσα P4 αλλά και την γλώσσα C (δεν χρησιμοποιήθηκε στην παρούσα διπλωματική) είτε παράλληλα είτε μεμονωμένα. Αρχικά γίνεται η συγγραφή του P4 προγράμματος και στην συνέχεια μέσω μεταγλωττιστών η μετατροπή της καταρχάς σε γλώσσα C και κατά δεύτερον σε μικροεντολές τις οποίες και θα εκτελέσει η κάρτα. Το τελικό παραγόμενο αρχείο έχει επέκταση **nffw** και είναι αυτό που θα σταλεί στην κάρτα για να εκτελεστεί. Για να γίνει η αποστολή, αλλά και η μετέπειτα επικοινωνία με την κάρτα, χρησιμοποιούνται δύο επιπλέον προγράμματα, τα **pif_rte** και **rte_cli**. Το **pif_rte** τρέχει στο μηχάνημα στο οποίο ανήκει η κάρτα και απαιτείται ώστε να μπορούν να φορτωθούν σε αυτή τα επιθυμητά προγράμματα. Λειτουργεί σαν server που λαμβάνει τα διάφορα αιτήματα και αναλαμβάνει να επικοινωνήσει με την κάρτα ώστε να αλλάξει την λειτουργία και τις ρυθμίσεις της. Από την άλλη, το **rte_cli** αποτελεί πρόγραμμα-ελεγκτή που δημιουργεί αιτήματα προς το **pif_rte**. Μέσω του **rte_cli** με τις κατάλληλες υποεντολές μπορούμε να ορίσουμε και να αλλάξουμε το πρόγραμμα λειτουργίας της κάρτας, να προωθήσουμε ρυθμίσεις τόσο κατά την εκκίνηση όσο και κατά την λειτουργία της κάρτας και να διαβάσουμε digests που θα παράγει η κάρτα. Για τις αρχικές ρυθμίσεις ειδικότερα χρησιμοποιείται αρχείο με επέκταση **p4cfg**. Αυτό μπορεί να δημιουργηθεί και μέσω του **Programmer Studio**, περιλαμβάνει την αρχική παραμετροποίηση του προγράμματος και στέλνεται παράλληλα με αυτό στην κάρτα.

Κρίνεται σκόπιμο να παρουσιάσουμε κάποιες δυσκολίες που συναντήσαμε κατά την εφαρμογή και την χρήση των παραπάνω λειτουργιών και αντίστοιχες λύσεις, καθώς μπορούν να αποτελέσουν χρήσιμη πληροφορία για μελλοντικές εργασίες πάνω στις κάρτες. Αρχικά για την χρήση του NFP-SDK και τον προγραμματισμό των καρτών απαιτείται η εγκατάσταση διαφορετικού προγράμματος οδήγησης (driver) στο λειτουργικό του μηχανήματος διαχείρισης της κάρτας, απ' ότι χρειάζεται για την λειτουργία της με παραδοσιακές μεθόδους. Η εισαγωγή αυτού του driver γίνεται κατά την εγκατάσταση του NFP-SDK, ωστόσο θα πρέπει να έχει διαγραφεί ο προηγούμενος, αλλιώς η διαδικασία θα αποτύχει.

Στην συνέχεια, κατά την χρήση του συγκεκριμένου driver και του NFP-SDK, οι φυσικές θύρες της κάρτας δεν είναι ορατές από το λειτουργικό σύστημα. Εφόσον παρέχονται δυνατότητες εικονικοποίησης (SR-IOV support [30]) από το λειτουργικό αλλά και τα BIOS του συστήματος στο οποίο συνδέεται η κάρτα, τότε μπορούν να γίνουν ορατές εικονικές θύρες που αντιστοιχούν στις φυσικές. Ωστόσο το μηχάνημα που χρησιμοποιήθηκε για την παρούσα διπλωματική δεν υποστήριζε την συγκεκριμένη τεχνολογία, και επομένως δεν ήταν δυνατή η διαχείριση τους και η λήψη στατιστικών με την χρήση συνηθισμένων εργαλείων (ifconfig, ethtool). Το NFP-SDK διαθέτει κάποια σχετικά προγράμματα για τον έλεγχο των θυρών, που όμως δεν παρέχουν κάποιο ολοκληρωμένο εγχειρίδιο χρήσης με αποτέλεσμα να γίνεται αρκετά περίπλοκη η χρήση τους. Στην περίπτωσή μας, έγινε συλλογή των σχετικών στατιστικών από τα μηχανήματα πηγής και προορισμού της κίνησης και όχι από τις κάρτες.

Παρατηρήθηκαν επίσης διάφορα προβλήματα στην διασύνδεση της κάρτας με μεταγωγείς. Συγκεκριμένα, συχνά οι θύρες των μεταγωγών στις οποίες συνδέονταν οι θύρες τις κάρτας

έπεφταν. Δεν καταφέραμε να λύσουμε το συγκεκριμένο πρόβλημα πέρα από το να επαναφέρουμε χειροκίνητα τις θύρες που έπεφταν.

Τέλος όσον αφορά τη σχέση του P4 με τις κάρτες, αυτές μέχρι στιγμής υποστηρίζουν σχεδόν όλες τις προδιαγραφές της γλώσσας $P4_{14}$ και μόνο ένα σύνολο από την γλώσσα $P4_{16}$. Παράλληλα παρουσιάζουν περιορισμούς στο μέγεθος προγραμμάτων (εντολών) που μπορούν να εκτελέσουν. Το όριο ορίζεται στις παραγόμενες μικροεντολές και όχι στον κώδικα P4, επομένως ο μέγιστος αριθμός γραμμών P4 ορίζεται ανάλογα με το πρόγραμμα και την αντιστοίχισή του σε μικροεντολές και δεν είναι γνωστός.

5.2 Πρόσθετα Εργαλεία Ανάπτυξης

5.2.1 Jinja2

Το Jinja2 [31] αποτελεί μια μηχανή προτύπων βασισμένη στην γλώσσα προγραμματισμού Python. Μέσω αυτού μπορούμε να ορίσουμε να δημιουργήσουμε γρήγορα και αποδοτικά αρχεία μεγάλου μεγέθους που οι επιμέρους γραμμές παρουσιάζουν ένα επαναλαμβανόμενο μοτίβο. Στην παρούσα διπλωματική χρησιμοποιήθηκε για την δημιουργία αρχείων που περιέχουν τις αρχικές ρυθμίσεις του μηχανισμού. Τα αρχεία αυτά περιέχουν εκατοντάδες γραμμές από παρόμοιες εντολές και η συμβατική συγγραφή τους θα ήταν εξαιρετικά χρονοβόρα. Με βάση το Jinja2 framework ορίζουμε τον βασικό σκελετό που επαναλαμβάνεται και τα σημεία που αλλάζουν εισάγονται σαν μεταβλητές μέσω εντολών python. Έτσι μπορούμε να παράγουμε εκατοντάδες γραμμές ρυθμίσεων έχοντας γράψει μόλις μερικές δεκάδες.

5.2.2 Scapy

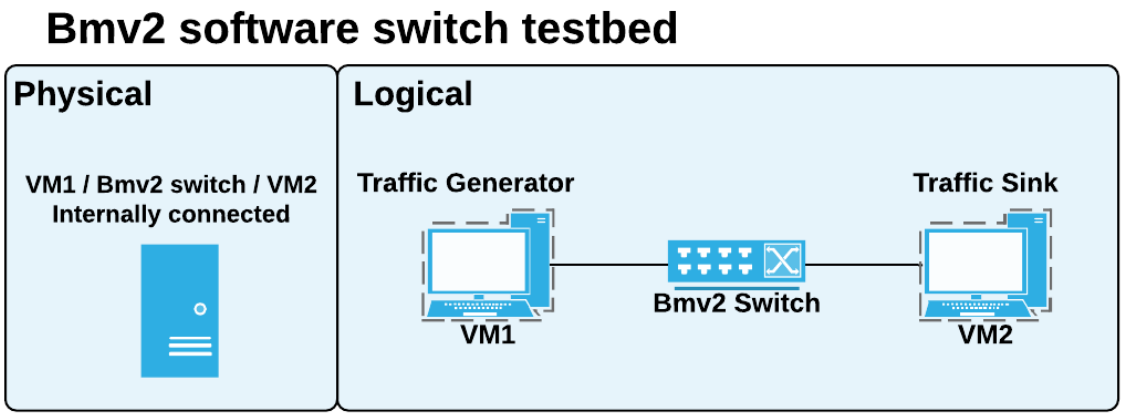
Το Scapy [32] αποτελεί βιβλιοθήκη της γλώσσας Python με σκοπό την δημιουργία, ανίχνευση και ανάλυση πακέτων δικτυακής κίνησης. Μέσω αυτής μπορούμε να κατασκευάσουμε και να στείλουμε πακέτα στον μηχανισμό μας, ώστε να ελέγξουμε την ορθή λειτουργία αυτού.

5.3 Πειραματική Διάταξη και Εργαλεία

Για την μέτρηση της επίδοσης του παρόντος μηχανισμού χρησιμοποιούμε το Bmv2 software switch με την διάταξη που φαίνεται στο σχήμα 5.1. Αυτή αποτελείται από τρία διαφορετικά virtual machines που βρίσκονται στον ίδιο server. Το πρώτο vm (VM1) αποτελεί την πηγή κίνησης ενώ το τρίτο (VM2) είναι ο δέκτης. Στο ενδιάμεσο βρίσκεται το vm που φιλοξενεί το Bmv2 και προωθεί αποκλειστικά μέσω αυτού την κίνηση που παράγεται από το VM1 προς το VM2. Επιπλέον στο vm του Bmv2 υπάρχει και ο μηχανισμός συλλογής των digests που δημιουργεί κατά την λειτουργία του ο μηχανισμός. Οι συνδέσεις μεταξύ των vms είναι απομονωμένες από το υπόλοιπο δίκτυο και δεν διέρχεται μέσω αυτών άλλη κίνηση.

Επιπροσθέτως χρησιμοποιήθηκαν τα ακόλουθα εργαλεία:

- **Tcpreplay** [33]: Πρόκειται για ένα σύνολο εργαλείων για την τροποποίηση και την αποστολή πακέτων που περιέχονται σε αρχεία pcap.
- **Mmwatch** [34]: Πρόγραμμα σε python που δέχεται ως όρισμα εντολές και τις εκτελεί περιοδικά, εμφανίζοντας αριθμητικά αποτελέσματα ως ρυθμούς. Στην παρούσα διπλωματική χρησιμοποιήθηκε με την εντολή ifconfig για την εξαγωγή ρυθμών αποστολής και λήψης πακέτων.



***Εικόνα 5.1**: Τοπολογία Πειραμάτων*

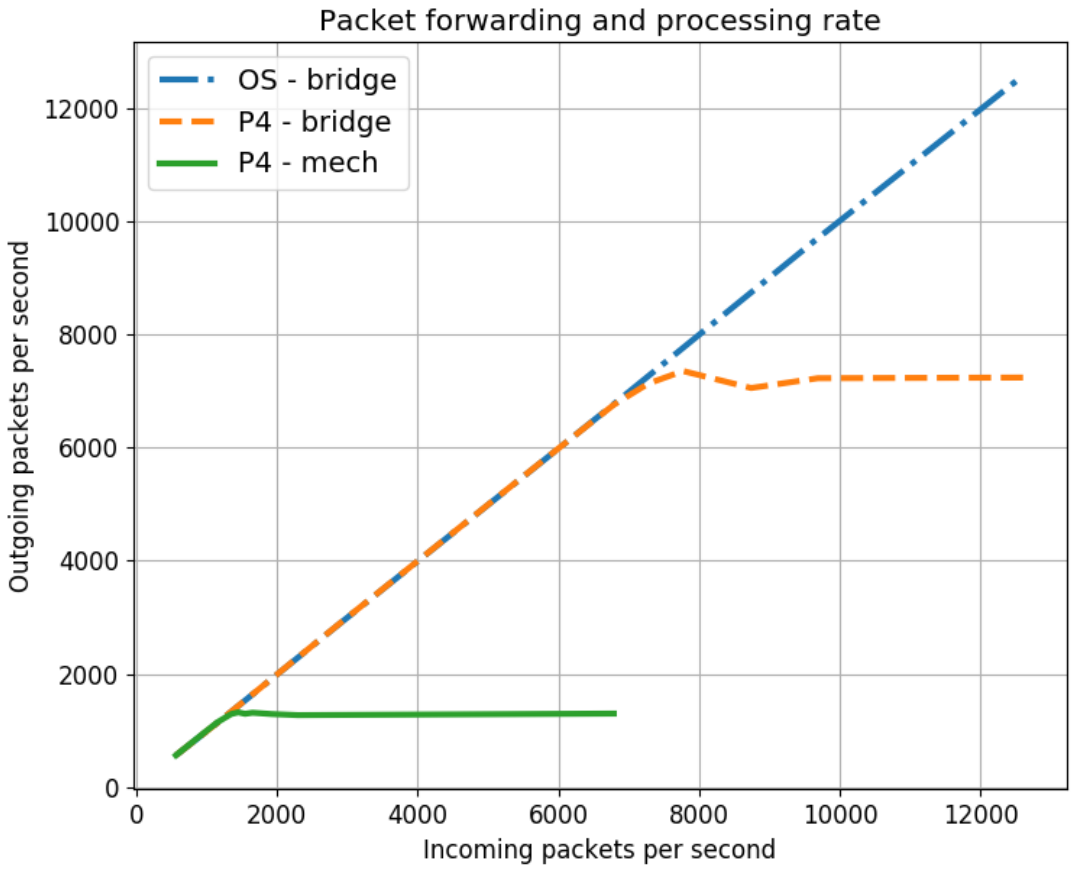
5.4 Χρονική απόδοση

Αρχικά κρίθηκε σκόπιμο να ελέγξουμε την απόδοση στον ρυθμό επεξεργασίας πακέτων του μηχανισμού όταν εφαρμόζεται στο Bmv2. Με αυτόν τον τρόπο θα πάρουμε μια εικόνα για το πρόσθετο φόρτο που επιφέρει ο εν λόγω μηχανισμός, αλλά και θα ορίσουμε ένα όριο για τον ρυθμό της κίνησης που θα πρέπει να παράγουμε σε επόμενα πειράματα.

Όπως αναφέραμε και πριν, η κίνηση ξεκινάει από το VM1, όπου με την βοήθεια του εργαλείου tcpreplay παρέχουμε σταθερό ρυθμό αποστολής πακέτων κατά την διάρκεια της κάθε μέτρησης, και που θα μεταβάλλουμε μεταξύ μετρήσεων. Η κίνηση αυτή προωθείται στο ενδιάμεσο VM του Bmv2, γίνεται η επεξεργασία της και αποστέλεται προς το VM2. Στο VM2 μέσω του εργαλείου mmwatch και της εντολής ifconfig μετράμε τον ρυθμό άφιξης των εισερχόμενων πακέτων.

Οι μετρήσεις έγιναν για τρεις τρόπους διασύνδεσης των VM1 και VM2:

- **OS - bridge**: Εδώ δεν λειτουργεί το Bmv2 αλλά τα VM1 και VM2 συνδέονται μέσω μιας εικονικής γέφυρας από το λειτουργικό του ενδιάμεσου VM.
- **P4 - bridge**: Μεταξύ των δύο VMs παρεμβάλουμε το Bmv2, το οποίο τρέχει πρόγραμμα P4 που προσομοιάζει μια γέφυρα.
- **P4 - mech**: Σε αυτήν την περίπτωση λειτουργεί στο Bmv2 ο μηχανισμός ανίχνευσης που αναπτύξαμε.



***Εικόνα 5.2**: Ρυθμός Επεξεργασίας και Προώθησης Πακέτων*

Τα αποτελέσματα των μετρήσεων φαίνονται στην εικόνα 5.2. Αρχικά βλέπουμε πως η χρήση και μόνο του εικονικού μεταγωγέα Bmv2 εισάγει μεγάλο κόστος στην λειτουργία της μεταφοράς πακέτων καθώς ο ρυθμός προώθησης πέφτει στα μόλις 7200 περίπου πακέτα το δευτερόλεπτο. Επομένως διαπιστώνουμε πειραματικά αυτό που αναφέραμε και προηγουμένως, δηλαδή πως το Bmv2 αποτελεί κυρίως περιβάλλον ανάπτυξης και εξομοίωσης λειτουργιών και όχι συσκευή προώθησης υψηλής απόδοσης. Στην συνέχεια παρατηρούμε ότι και η λειτουργία του μηχανισμού παρουσιάζει σημαντική μείωση στην απόδοση της τάξης του 81% (1300 πακέτα το δευτερόλεπτο). Αυτό αποτελεί μια σημαντική υπενθύμιση πως οι μηχανισμοί που αναπτύσσονται στο επίπεδο δεδομένων με σκοπό την μελέτη της κίνησης δεν έχουν αμελητέα επίδραση στον χρόνο προώθησης και συνεπώς αυτό θα πρέπει να λαμβάνεται υπόψη κατά την δημιουργία και εφαρμογή τους.

5.5 Αποτελεσματικότητα Μηχανισμού

Στην συνέχεια μελετήθηκε η αποτελεσματικότητα του προτεινόμενου μηχανισμού. Για την εξαγωγή ορθότερων αποτελεσμάτων από τον έλεγχο του μηχανισμού κρίθηκε σημαντική η χρήση πραγματικής κίνησης. Έτσι η μελέτη έγινε χρησιμοποιώντας κίνηση που προέρχεται

από το ιαπωνικό project WIDE και συγκεκριμένα αποτελεί κίνηση από το δίκτυο κορμού του WIDE προς ISP [35]. Επιπλέον ως κίνηση επίθεσης χρησιμοποιήθηκαν αρχεία κίνησης από την μελέτη των "Booters" [36].

5.5.1 Μεθοδολογία μετρήσεων

Μετά από εξέταση του αρχείου κανονικής κίνησης επιλέχθηκε ένα σύνολο 255 /24 δικτύων, τα οποία θα αποτελούν τα ελεγχόμενα δίκτυα για τον μηχανισμό μας. Η επιλογή αυτή έγινε με βάση τα ποσά της εισερχόμενης κίνησης προς αυτά τα δίκτυα, και επιλέχθηκαν αυτά με την μεγαλύτερη κίνηση ώστε να γίνει ουσιαστικός έλεγχος του μηχανισμού. Ωστόσο σε γενικότερο πλαίσιο ο μηχανισμός θα μπορούσε να αποδεχτεί οποιοδήποτε συνδυασμό διευθύνσεων και μεγέθους δικτύου. Βέβαια όσο πιο μικρά είναι τα επιλεγμένα δίκτυα τόσο καλύτερα αποτελέσματα θα εξάγουμε σχετικά με τον στόχο των επιθέσεων, επομένως το μέγεθος των δικτύων εξαρτάται από την απαίτηση για στοχευμένη ή μη ανίχνευση.

Επιπλέον λαμβάνοντας υπόψη την μειωμένη ταχύτητα επεξεργασίας των πακέτων από το Bmv2 και τις τιμές που βρήκαμε παραπάνω μειώνουμε την ταχύτητα δημιουργίας τους από το VM1 ώστε να μην τα χάσουμε από τον αυξημένο φόρτο. Παράλληλα ορίζουμε εποχές των 10 δευτερολέπτων στον μηχανισμό μας. Με τα παραπάνω πρακτικά προσομοιώνουμε εποχές που κάθε μία αντιστοιχεί σε 200 ms δευτερόλεπτα κανονικής κίνησης. Τις πρώτες 20 εποχές τις θεωρούμε μεταβατικές ώστε να μπορεί ο μηχανισμός να προσαρμοστεί στα δεδομένα τις κίνησης με βάση τις τεχνικές EWMA και EWMD, και επομένως στέλνουμε μόνο καλόβουλη κίνηση. Στην συνέχεια ξεκινάμε και την κίνηση επίθεσης με διάρκεια 20 εποχών, με στόχο το δίκτυο με την περισσότερη εισερχόμενη κίνηση.

Τέλος για τις τεχνικές EWMA και EWMD ορίζουμε τιμή του συντελεστή εξομάλυνσης a ίση με 1/256. Όπως αναφέραμε και στο προηγούμενο κεφάλαιο, όσο μεγαλύτερη τιμή δίνουμε στο a τόσο μεγαλύτερη σημασία δίνουμε στις καινούριες τιμές σε σχέση με τις παλιές. Θέλουμε λοιπόν η τιμή αυτή να είναι όσο μικρή γίνεται, ώστε ναι μεν ο μηχανισμός μας να διατηρεί μια εικόνα από τις μεταβολές τις κίνησης, αλλά να μην προσαρμόζεται γρήγορα σε απότομες διακυμάνσεις της κανονικής κίνησης αλλά και στην κίνηση επίθεσης. Γρήγορη προσαρμογή σε τέτοιες περιπτώσεις θα είχε ως αποτέλεσμα την ραγδαία αύξηση του μέσου και της διακύμανσης και έτσι και του ορίου των συνολικών ροών. Με αυτόν τον τρόπο ο μηχανισμός μετά από μικρό χρονικό διάστημα θα θεωρούσε την συνέχεια της επίθεσης ως φυσιολογική, αποτυγχάνοντας έτσι να την εντοπίσει και να στείλει ειδοποιήσεις.

Οι βασικές μετρικές απόδοσης που θα χρησιμοποιηθούν είναι τα ποσοστά σωστής και λανθασμένης ανίχνευσης επιθέσεων, δηλαδή True Positive Rate και False Positive Rate αντίστοιχα. Για να τα ορίσουμε θα πρέπει να εκφράσουμε κάθε πιθανή κατάσταση ενός αποτελέσματος:

- **True Positive (TP)**: Πρόκειται για την σωστή αποστολή digest για υποδίκτυο που την δεδομένη εποχή δέχεται επίθεση.
- **False Positive (FP)**: Πρόκειται για την αποστολή digest για υποδίκτυο που δεν δέχεται επίθεση την συγκεκριμένη εποχή. Σημαίνει λανθασμένη ανίχνευση.
- **True Negative (TN)**: Δεν γίνεται αποστολή digest για υποδίκτυο που δεν δέχεται επίθεση.

- **False Negative (FN)**: Δεν γίνεται αποστολή digest για υποδίκτυο που δέχεται επίθεση την δεδομένη εποχή. Σημαίνει αδυναμία ανίχνευσης.

Με βάση τα παραπάνω ορίζονται και οι μετρικές απόδοσης:

- **True Positive Rate**: Εκφράζει το ποσοστό των σωστών digests, δηλαδή των έγκυρων ειδοποιήσεων σχετικά με επίθεση, σε σχέση με το σύνολο όλων των ειδοποιήσεων που εν γένει θα έπρεπε να έχουν δημιουργηθεί και θα αντιστοιχούσαν στην πλήρη αναγνώριση της επίθεσης. Ορίζεται δηλαδή από τον τύπο:

$$TPR = \frac{TP}{TP + FN}$$

- **False Positive Rate**: Εκφράζει το ποσοστό των λανθασμένων digest, δηλαδή ειδοποιήσεων που δημιουργήθηκαν ενώ δεν υπάρχει επίθεση, σε σχέση με το σύνολο που αντιστοιχεί στην μη ύπαρξη επίθεσης. Ορίζεται από τον τύπο:

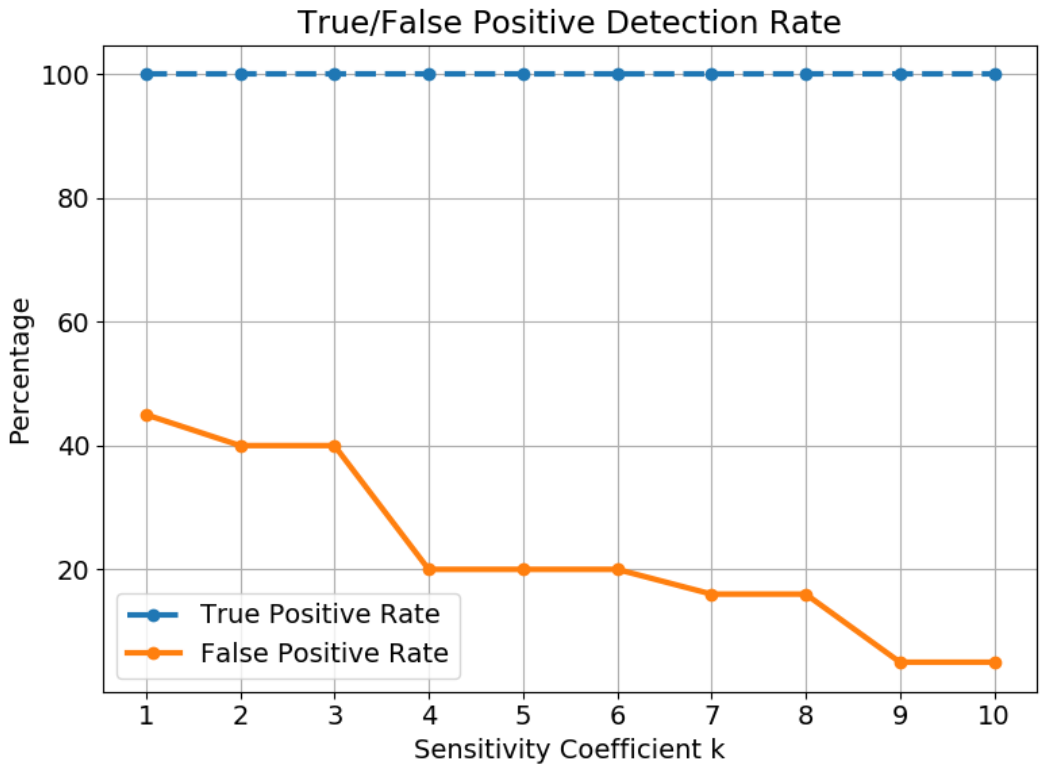
$$FPR = \frac{FP}{FP + TN}$$

5.5.2 Απόδοση ανίχνευσης ανωμαλιών συνολικών ροών.

Συνεχίζοντας θα μελετήσουμε την απόδοση του συνολικού μηχανισμού, θέλοντας παράλληλα να διαπιστώσουμε την σημασία της ταυτόχρονης χρήσης των επιμέρους μηχανισμών. Αρχικά ελέγχουμε μόνο τον πρώτο επιμέρους μηχανισμό που αφορά τις συνολικές ροές χρησιμοποιώντας τις τεχνικές EWMA και EWMD. Πιο συγκεκριμένα θα ελέγξουμε τις μετρικές TPR και FPR για διαφορετικές τιμές του συντελεστή ευαισθησίας k. Θυμίσουμε ότι ο συντελεστής k επιδρά στο όριο ανίχνευσης και ορίζει ένα εύρος τιμών συνολικών ροών που θεωρούνται αποδεκτές από τον μηχανισμό.

Στην εικόνα 5.3 φαίνονται τα αποτελέσματα των μετρήσεων. Αρχικά βλέπουμε ότι καταφέρνουμε να αναγνωρίσουμε πλήρως τις επιθέσεις, σε όλες τις εποχές που διαρκούν και για όλες τις τιμές του συντελεστή k. Αυτό σημαίνει ότι οι συνολικές ροές αποτελούν μια καλή ένδειξη επίθεσης. Ωστόσο παρατηρούμε ότι εμφανίζονται αρκετά μεγάλα ποσοστά λανθασμένων ανιχνεύσεων, που σημαίνει ότι σε πολλές περιπτώσεις εμφανίζονται κάποιες διακυμάνσεις σε κανονική κίνηση που αναγνωρίζονται λανθασμένα ως επίθεση. Όσο αυξάνουμε τον συντελεστή k παρατηρούμε μείωση στα ποσοστά λάθους.

Ωστόσο δεν είναι συνετό να αυξάνουμε αυθαίρετα τον συντελεστή k, γιατί με αυτόν τον τρόπο αυξάνουμε πρακτικά και το όριο ανίχνευσης. Έτσι θα μπορούσαν επιθέσεις με λιγότερες μοναδικές ροές να περάσουν απαρατήρητες από τον μηχανισμό μας. Επιπλέον δεν μπορούμε να αναγνωρίσουμε ποιο είναι το υποδίκτυο που δέχεται επίθεση μιας και για κάθε καινούρια ροή πάνω από το όριο ενεργοποιείται ο μηχανισμός ανεξαρτήτως υποδικτύου. Παρατηρούμε λοιπόν ότι αν και οι συνολικές ροές είναι μια καλή μετρική, εντούτοις δεν είναι αρκετή.

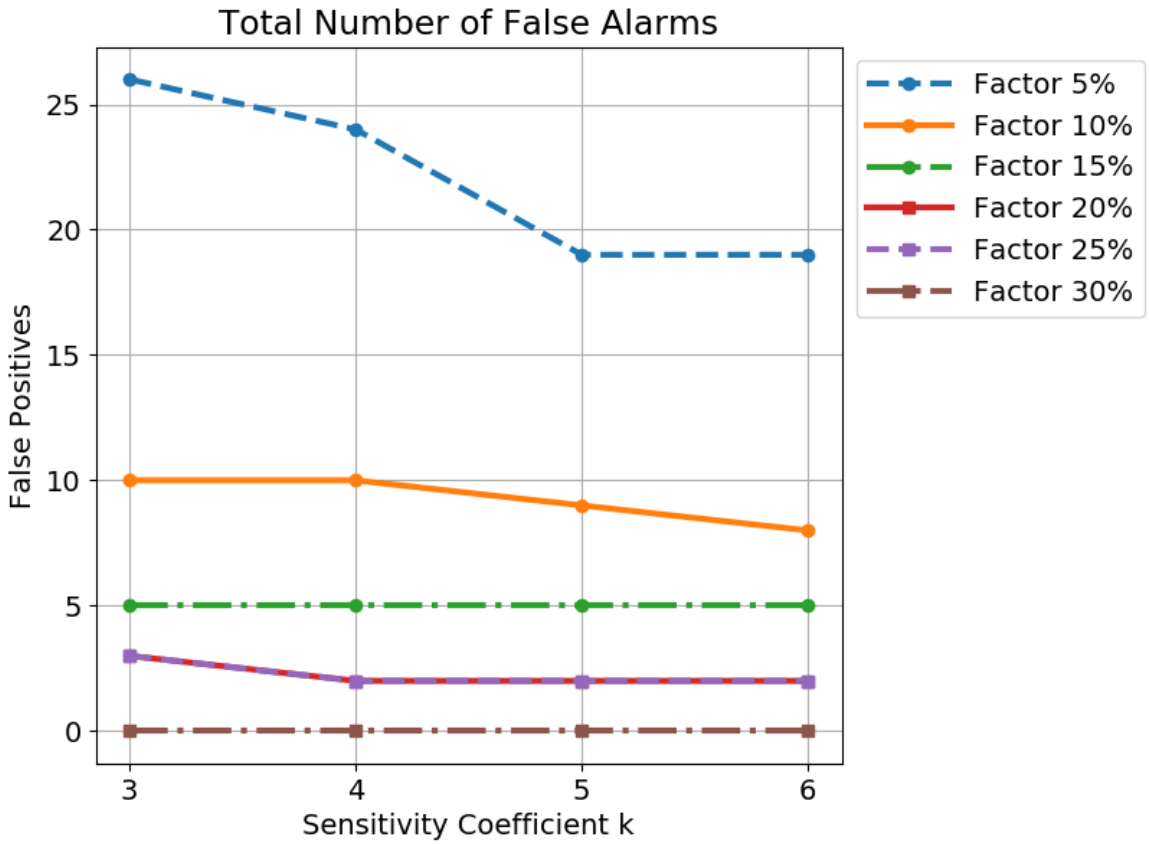


***Εικόνα 5.3**: Ποσοστά TPR/FPR του Επιμέρους Μηχανισμού Συνολικών Ροών για διάφορα k*

5.5.3 Συνδυασμός συνολικών ροών με ροές ανά δίκτυο

Έπειτα ενισχύουμε τον συνολικό μηχανισμό μας προσθέτοντας την δυνατότητα να μετράμε ροές ανά υποδίκτυο και να δίνουμε σημασία σε αυτά που εμφανίζουν ένα σημαντικό ποσοστό **f** εισερχόμενων ροών σε σχέση με τις συνολικές. Θα δοκιμάσουμε επομένως να συνδυάσουμε το ποσοστό f με τον συντελεστή k για να δούμε αν μπορούμε να πάρουμε ικανοποιητικά αποτελέσματα σε μικρότερες τιμές του k.

Αρχικά πρέπει να τονιστεί ότι σε όλες τις μετρήσεις οι δύο επιμέρους μηχανισμοί κατάφεραν να αναγνωρίσουν πλήρως τις επιθέσεις όπως και πριν. Στην εικόνα 5.4 παρουσιάζονται τα συνολικά False Positives που δίνουν οι δύο μηχανισμοί, για διάφορες τιμές των f και k. Παρατηρούμε ότι αυξάνοντας το ποσοστό f πετυχαίνουμε λιγότερα λάθη. Αυτό σημαίνει ότι τα υποδίκτυα με μικρότερη κίνηση που μπορεί να παρουσιάσουν διακυμάνσεις δεν δημιουργούν λανθασμένες ειδοποιήσεις, αλλά και πως σε περίπτωση επίθεσης ροές που κατευθύνονται σε υποδίκτυα μικρότερης κίνησης δεν ενεργοποιούν τον μηχανισμό. Επιπλέον βλέπουμε ότι με τους δύο μηχανισμούς πετυχαίνουμε ελάχιστα λάθη ακόμα και σε μικρές τιμές του k, γεγονός που δεν ίσχυε όταν λειτουργούσε μόνο ο πρώτος μηχανισμός. Συγκεκριμένα βλέπουμε ότι για ποσοστό f ίσο με 30% επιτυγχάνουμε μηδενικά λάθη.



***Εικόνα 5.4**: Λανθασμένες Ειδοποιήσεις από Συνολικές Ροές και Ροές ανά Υποδίκτυο*
Για factors 20 και 25 οι μετρήσεις ταυτίζονται.

5.5.4 Προσθήκη Μηχανισμού Ανίχνευσης Ασυμμετρίας

Τέλος προθέτουμε τον μηχανισμό ανίχνευσης ασυμμετρίας κίνησης στους προηγούμενους δύο και μετράμε την απόδοση του συνολικού προτεινόμενου μηχανισμού. Έχοντας διεξάγει από πριν ανάλυση στην κανονική κίνηση, έχουμε προσδιορίζει για κάθε υποδίκτυο και πρωτόκολλο ένα αντιπροσωπευτικό ποσοστό ασυμμετρίας NR. Κάθε φορά λοιπόν θα συγκρίνουμε το παρατηρούμενο ποσοστό CR με το αντιπροσωπευτικό NR, και εφόσον ο λόγος τους υπερβεί την τιμή r = 1.2 (ένα περιθώριο διαφοράς των δύο τιμών ασυμμετρίας), θα θεωρήσουμε ότι έχουμε ύπαρξη ασύμμετρης κίνησης στο συγκεκριμένο υποδίκτυο. Επιπλέον θα διατηρήσουμε σταθερό συντελεστή k = 3 και θα μετρήσουμε την απόδοση του συνολικού μηχανισμού για διάφορες τιμές του ποσοστού f της σημασίας του υποδικτύου.

Στον πίνακα 5.1 έχουμε τα αποτελέσματα των μετρήσεων για τις διάφορες τιμές του f. Σε όλα τα πειράματα έχουμε και πάλι πλήρη ανίχνευση επιθέσεων, δηλαδή TPR = 100%. Στον πίνακα παρουσιάζουμε τον αριθμό των λανθασμένων ειδοποιήσεων που επιστρέφει ο μηχανισμός. Βλέπουμε ότι με την εφαρμογή και του τρίτου μηχανισμού παίρνουμε αισθητά λιγότερα false positives απ' ότι πριν, και επιτυγχάνουμε μηδενικά λάθη σε ακόμα μικρότερες τιμές των k και f , συγκεκριμένα για k = 3 και f = 25. Αυτό είναι σπουδαίο, ειδικά αν

σκεφτούμε ότι από μόνος του ο μηχανισμός συνολικών ροών εμφάνιζε ποσοστά λάθους της τάξης του 45% σε ανάλογη τιμή. Έτσι διατηρώντας αυτές τις τιμές χαμηλά, μπορούμε να επιτύχουμε αναγνώριση μικρότερων DDoS επιθέσεων που σε άλλη περίπτωση θα θεωρούνταν κανονική κίνηση. Θεωρούμε λοιπόν πως η ύπαρξη πολλαπλών μετρικών συμβάλει σημαντικά στην λεπτομερέστερη αναγνώριση επιθέσεων.

Πίνακας 5.1: Λάθη μηχανισμού με και χωρίς την Ανίχνευση Ασυμμετρίας

Network Significance (factor f)	Two Features	All Features
5%	26	12
10%	10	2
15%	5	1
20%	3	1
25%	3	0
30%	0	0

Κεφάλαιο 6 - Συμπεράσματα και Επεκτάσεις

Στο πλαίσιο της παρούσας διπλωματικής εργασίας αναπτύχθηκε ένας μηχανισμός με στόχο την ανίχνευση ανωμαλιών στην δικτυακή κίνηση που προέρχονται από επιθέσεις DDoS, απευθείας στο επίπεδο δεδομένων μέσω της γλώσσας P4.

Όσον αφορά το P4, αυτό παρέχει αρκετές δομές ικανές να περιγράψουν λειτουργίες ανάλυσης και προώθησης κίνησης. Παρατηρήθηκαν διάφοροι περιορισμοί από την ίδια την γλώσσα, αλλά και από τις συσκευές που την υποστηρίζουν, οι οποίοι οφείλονται κυρίως στην φύση της γλώσσας, και στο γεγονός πως πρόκειται για μια καινούρια σχετικά γλώσσα που ακόμα αναπτύσσεται και που οι κατασκευαστές ακόμα προσαρμόζονται σε αυτήν. Παρά τους περιορισμούς, καταφέραμε να δημιουργήσουμε έναν μηχανισμό μεταφέρσιμο μεταξύ των δύο περιβαλλόντων ανάπτυξης και εφαρμόσιμο εξ΄ ολοκλήρου στο επίπεδο δεδομένων των συσκευών. Δοκιμάζοντας τον μηχανισμό με βάση αρχεία πραγματικής κίνησης, μελετήσαμε την χρήση μετρικών εισερχόμενων ροών και ασυμμετρίας κίνησης και διαπιστώσαμε ότι η ταυτόχρονη χρήση τους ενισχύει την αποτελεσματικότητα του μηχανισμού, πετυχαίνοντας ακριβέστερη και λεπτομερέστερη ανίχνευση επιθέσεων. Επιπλέον δώσαμε σε αυτόν την δυνατότητα να δημιουργεί μηνύματα ειδοποιήσεων όταν αναγνωρίσει κάποια ανωμαλία. Πλέον δεν απαιτούνται τακτικά αιτήματα από κάποιον ελεγκτή, αλλά τα μηνύματα μπορούν να χρησιμοποιηθούν άμεσα από επόμενα στάδια για γρηγορότερη αντιμετώπιση επιθέσεων.

Ένα άμεσο επόμενο βήμα θα ήταν να δοκιμαστεί ο μηχανισμός σε συσκευές υψηλής απόδοσης. Όπως παρατηρήσαμε μέσω του Bmv2, η χρήση του μηχανισμού επιφέρει χρονική καθυστέρηση στην προώθηση των πακέτων. Επομένως θα πρέπει να εξεταστεί αν η εφαρμογή του σε πραγματική ροή κίνησης επιτρέπει την διατήρηση αποδεκτών ρυθμών προώθησης, μιας και ο κύριος στόχος του επιπέδου δεδομένων είναι η προώθηση των πακέτων. Εδώ υπάρχει ταυτόχρονα ερευνητικό ενδιαφέρον για την ανάπτυξη νέων και την προσαρμογή παλαιότερων αλγορίθμων, ώστε να είναι ικανοί να εφαρμοστούν στο πλαίσιο προώθησης κίνησης σε προγραμματιζόμενες συσκευές, απαιτώντας μειωμένους πόρους και χρόνο επεξεργασίας από αυτές.

Μια επέκταση του μηχανισμού θα ήταν ο συνδυασμός της λειτουργίας του με έναν ελεγκτή, ώστε να γίνεται δυναμική τροποποίηση των τιμών των διαφόρων σταθερών ορίων που εφαρμόζει ο μηχανισμός. Όπως έχουμε αναφέρει τα όρια των επιμέρους μηχανισμών παρέχονται στην αρχή της λειτουργίας του. Ωστόσο θα ήταν ενδιαφέρον να γίνει προσπάθεια ώστε να αναπτυχθεί ένα σύστημα που θα μπορεί αυτόνομα και δυναμικά, με βάση την κίνηση του δικτύου και τις αναμενόμενες τιμές της, να ρυθμίζει περιοδικά σε μεγαλύτερα διαστήματα τα όρια αυτά, ώστε να επιτυγχάνεται η καλύτερη προσαρμογή του μηχανισμού στην κίνηση του δικτύου κατά την διάρκεια της λειτουργίας του.

Θα μπορούσε επιπλέον να μελετηθεί η υλοποίηση ενός επόμενου επιπέδου αντιμετώπισης επιθέσεων, το οποίο αφού λάμβανε ειδοποιήσεις παρόμοιες με του παρόντος μηχανισμού, θα προσπαθούσε να τις ανακόψει. Εδώ είναι εξίσου ενδιαφέρον να διαπιστωθεί αν κάτι τέτοιο θα μπορούσε να γίνει και πάλι εξ' ολοκλήρου στο επίπεδο δεδομένων ή αν τελικά χρειάζεται ένας τρίτος μηχανισμός αφοσιωμένος στην ανακοπή των επιθέσεων.

Μια ιδέα για μελλοντικές έρευνες θα ήταν και η προσπάθεια συνδυασμού του προγραμματισμού επιπέδου δεδομένων με τεχνικές μηχανικής μάθησης. Θα μπορούσαν να γίνουν προσπάθειες υλοποίησης μηχανισμών, στο πλαίσιο των οποίων οι συσκευές, είτε αυτόνομα είτε συνεργατικά (πράγμα που φαίνεται και πιο λογικό), θα μπορούν να διεξάγουν πιο σύνθετες αναλύσεις και να εξάγουν πιο σύνθετες μετρικές, αξιοποιώντας πρωτόκολλα βασισμένα σε αλγορίθμους μηχανικής μάθησης.

Τέλος μιας και ο προγραμματισμός επιπέδου δεδομένων βρίσκεται ακόμα σε αρχικά στάδια, παρουσιάζεται έντονο επιστημονικό ενδιαφέρον για την δυνατότητα συμβολής του σε διαφόρους τομείς των δικτυακών λειτουργιών, όχι μόνο της αντιμετώπισης επιθέσεων.

Βιβλιογραφία

[1] N.Feamster, J.Rexford, E.Eegura, "The Road to SDN: An Intellectual History of Programmable Networks"
link: https://www.cs.princeton.edu/courses/archive/fall13/cos597E/papers/sdnhistory.pdf

[2] "OpenFlow Switch Specification version 1.5.1"
link: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[3] P.Bosshart, D.Daly, G.Gibb, M.Izzard, N.McKeown, J.Rexford, C.Schlesinger, D.Talayco, A.Vahdat, G.Varghese, D.Walker, "P4: Programming Protocol-Independent Packet Processors"
link: https://www.sigcomm.org/sites/default/files/ccr/papers/2014/July/0000000-0000004.pdf

[4] R.Bifulco, G. Retvari, "A Survey on the Programmable Data Plane: Abstractions, Architectures, and Open Problems"
link: https://5gtango.eu/papers/2018/2018_HPSR.pdf

[5] "The P4 Language Specification Version 1.0.5 (Nov 2018)"
link: https://p4.org/p4-spec/p4-14/v1.0.5/tex/p4.pdf

[6] "$P4_{16}$ Language Specification Version 1.1.0 (Nov 2018)"
link: https://p4.org/p4-spec/docs/P4-16-v1.1.0-spec.pdf

[7] "P4 tutorials"
link: https://github.com/p4lang/tutorials/blob/master/exercises/basic/solution/basic.p4

[8] K.Giotis, C.Argyropoulos, G.Androulakis, D.Kalogeras, V.Maglaris, "Combining OpenFlow and sFlow for an effective and scalable Anomaly Detection and Mitigation mechanism on SDN environments"
link:
https://www.researchgate.net/publication/260838090_Combining_OpenFlow_and_sFlow_for_an_effective_and_scalable_anomaly_detection_and_mitigation_mechanism_on_SDN_environments

[9] “What is a DDoS Attack?”
link: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/

[10] “What is a DDoS Botnet?”
link: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-botnet/

[11] "Jaw-Dropping DDoS Statistics to Keep in Mind for 2019"
link: https://hostingtribunal.com/blog/ddos-statistics/

[12] V.Sivaraman, S.Narayana, O.Rottenstreich, S.Muthukrishnan, J.Rexford, "Heavy-Hitter Detection Entirely in the Data Plane"

link: https://www.cs.princeton.edu/~jrex/papers/hashpipe17.pdf

[13] J.Hill, M.Aloserij, P.Grosso, "Tracking network flows with P4"
link: https://pdfs.semanticscholar.org/a5c6/8a0596c3829649439cdc0bb7a0eb5187a41a.pdf

[14] "Bmv2 Software Switch"
link: https://github.com/p4lang/behavioral-model

[15] A.C.Lapoli, J.A.Marques, L.Paschoal, "Offloading Real-time DDoS Attack Detection to Programmable Data Planes"
link: http://dl.ifip.org/db/conf/im/im2019/189394.pdf

[16] A.Sivaraman, C.Kim, R.Krishnamoorthy, A.Dixit, M.Budiu, "DC.p4: Programming the Forwarding Plane of a Data-Center Switch"
link: https://dl.acm.org/citation.cfm?id=2775007

[17] T.Kohler, R.Mayer, F.Durr, M.Maab, S.Bhowmik, K.Rothermel, "P4CEP: Towards In-Network Complex Event Processing"
link: https://arxiv.org/abs/1806.04385

[18] N.Katta, M.Hira, C.Kim, A.Sivaraman, J.Rexford, "HULA: Scalable Load Balancing Using Programmable Data Planes"
link: https://dl.acm.org/citation.cfm?id=2890968

[19] "Netronome SmartNICs"
https://www.netronome.com/products/smartnic/overview/

[20] M.Mitzenmacher, E.Upfal, "Probability and Computing: Randomized Algorithms and Probabilistic Analysis"
link: https://books.google.gr/books?id=0bAYl6d7hvkC&pg=PA110#v=onepage&q&f=false

[21] "Bloom filter"
link: https://en.wikipedia.org/wiki/Bloom_filter

[22] N.Wein, M.Charikar, "Lecture 7: Heavy Hitters: the Count-Min Sketch"
link: http://web.stanford.edu/class/cs369g/files/lectures/lec7.pdf

[23] "Explaining The Count Sketch Algorithm"
link: https://stackoverflow.com/questions/6811351/explaining-the-count-sketch-algorithm

[24] M.Savi D.Parniewicz, D.Ding, D.Siracusa, "Sketch-based DDoS detection using P4 data plane programming"
link: https://geant.app.box.com/s/eufhet10f3l9rb0wbcijp0wcrfcdtbes

[25] C.Kreibich, A.Warfield, J.Crowcroft, S.Hand, I.Pratt, "Using Packet Symmetry to Curtail Malicious Traffic"
link: https://pdfs.semanticscholar.org/929a/a495746b7c177e537d839f3a824bc143e060.pdf

[26] K.Giotis, G.Androulidakis, V.Maglaris, "A scalable anomaly detection and mitigation architecture for legacy networks via an OpenFlow middlebox"
link: https://onlinelibrary.wiley.com/doi/epdf/10.1002/sec.1368

[27] "The BMv2 Simple Switch target"
link: https://github.com/p4lang/behavioral-model/blob/master/docs/simple_switch.md

[28] "p4c compiler"
link: https://github.com/p4lang/p4c

[29] simple_switch_CLI
link: https://github.com/p4lang/behavioral-model/blob/master/tools/runtime_CLI.py

[30] "What is SR-IOV?"
link: https://blog.scottlowe.org/2009/12/02/what-is-sr-iov/

[31] Jinja
link: https://jinja.palletsprojects.com/en/2.10.x/

[32] Scapy
link: https://scapy.net/

[33] "Tcpreplay"
link: https://tcpreplay.appneta.com

[34] "Mmwatch"
link: https://github.com/cloudflare/cloudflare-blog/tree/master/2017-06-29-ssdp

[35] "Packet traces from WIDE backbone"
link: https://mawi.wide.ad.jp/mawi/

[36] J.J.Santanna, R.Rijswijk-Deij, R.Hofstede, A.Sperotto, M.Wierbosch, L.Z.Granville, A.Pras, "Booters - An Analysis of DDoS-as-a-Service Attacks"
link: https://www.researchgate.net/publication/283883204_Booters_-_An_analysis_of_DDoS-as-a-service_attacks