ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

NETMODE - NETWORK MANAGEMENT & OPTIMAL DESIGN LABORATORY

Σχεδιασμός και ανάπτυξη μηχανισμών συλλογής δεδομένων σε πειραματικές πλατφόρμες για το Internet του μέλλοντος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αδάμ Ι. Παυλίδης

Επιβλέπων : Βασίλειος Μάγκλαρης

Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

NETMODE - NETWORK MANAGEMENT & OPTIMAL DESIGN LABORATORY

Σχεδιασμός και ανάπτυξη μηχανισμών συλλογής δεδομένων σε πειραματικές πλατφόρμες για το Internet του μέλλοντος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αδάμ Ι. Παυλίδης

Επιβλέπων : Βασίλειος Μάγκλαρης

Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 17η Ιουλίου 2015.

.....	.....	.....
Βασίλειος Μάγκλαρης	Νεκτάριος Κοζύρης	Δημήτριος Καλογεράς
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Ερευνητής ΕΠΣΕΥ 'Β

Αθήνα, Ιούλιος 2015

.....

Αδάμ Παυλίδης

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

Ευχαριστίες

Η παρούσα διπλωματική εργασία αποτελεί το τελευταίο στάδιο των προπτυχιακών σπουδών μου στο Εθνικό Μετσόβιο Πολυτεχνείο.

Αρχικά θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Βασίλειο Μάγκλαρη για την εμπιστοσύνη που μου έδειξε, την ευκαιρία που μου έδωσε καθώς και για τις πολύτιμες συμβουλές του. Ακόμη, ευχαριστώ όλα τα μέλη του εργαστηρίου NETMODE για την βοήθεια τους, και ιδιαιτέρως τον υπεύθυνο της διπλωματικής μου εργασίας Κωνσταντίνο Γιώτη, για την άριστη συνεργασία μας καθώς και για την πολύτιμη καθοδήγηση του σε όλα τα στάδια εκπόνησης της.

Τέλος θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου για την ανεκτίμητη υποστήριξη και συμπαράσταση τους, στη διάρκεια των σπουδών μου.

Περίληψη

Σκοπός της διπλωματικής εργασίας είναι η ανάπτυξη μηχανισμών για την συλλογή δεδομένων δικτυακής κίνησης, λαμβάνοντας υπ όψιν τα ετερογενή περιβάλλοντα που συναντούμε στην σύγχρονη εποχή. Δίνεται ιδιαίτερη έμφαση στην ικανότητα προσαρμογής του κάθε μηχανισμού σε αυτές τις υποδομές χωρίς οι επιμέρους ιδιαιτερότητες τους να αποτελούν τροχοπέδη. Τα δεδομένα που συλλέγονται είναι ιδιαίτερα χρήσιμα στους διαχειριστές της κάθε υποδομής όπως επίσης και στους χρήστες της σε περίπτωση που πρόκειται για πλατφόρμα πολλαπλών ενοίκων. Επομένως, το επιθυμητό αποτέλεσμα είναι η κατανομή των δεδομένων ανάλογα με τους πόρους που αντιστοιχούν στον κάθε ένοικο.

Ειδικότερα, αναπτύχθηκε ένας μηχανισμός για αρχιτεκτονικές Δικτύων Οριζόμενων από Λογισμικό, βασισμένων στο πρωτόκολλο OpenFlow. Οι υποδομές που μελετούνται μέσω της υλοποίησης είναι ενός ή πολλαπλών χρηστών – ενοίκων. Ο μηχανισμός βασίζεται στην κίνηση σηματοδοσίας που ανταλλάσουν εγγενώς τα στοιχεία της SDN υποδομής και συμπληρώνεται κάνοντας δειγματοληψία της κίνησης με χρήση του sFlow. Στόχος είναι η πληροφορία που συλλέγεται να είναι προσβάσιμη από τον εκάστοτε ενδιαφερόμενο. Τέλος αναλύονται εναλλακτικές αρχιτεκτονικές, οι οποίες βασίζονται σε αναπροσαρμογές της αρχικής, κάνοντας χρήση του εικονικού μεταγωγέα Open vSwitch. Με αυτό τον τρόπο επιδιώκουμε την προσαρμογή του μηχανισμού τόσο σε υποδομές υπολογιστικών νεφών όσο και σε ασύρματες υποδομές.

Ο μηχανισμός αυτός μπορεί να αποτελέσει βάση για την δημιουργία ενός ενοποιημένου πλαισίου για την άντληση δικτυακών δεδομένων και αυτόματης παρουσίασης των δεδομένων αυτών στους χρήστες των διαφόρων πειραματικών υποδομών μεγάλης κλίμακας σαν μέρος της λειτουργικότητας που προσφέρει το περιβάλλον.

Λέξεις Κλειδιά: <<Δίκτυα Οριζόμενα από Λογισμικό, OpenFlow, Παρακολούθηση Δικτύων, Περιβάλλοντα Πολλαπλών Ενοίκων, Ετερογενή Περιβάλλοντα, Εικονική Δικτύωση >>

Abstract

The scope of this thesis was the development of a mechanism for the collection of network monitoring data, taking into account the heterogeneous environments encountered nowadays. Considering the variety of these environments, we strive to make the mechanism as adaptable as possible to the individual characteristics of each facility. The importance of data collected is significant not only for the administrators of each platform, but also for each user – tenant (when multi – tenant environments are concerned). In the case of multi – tenant environments, network monitoring data have to be accessible only from the corresponding tenant, enforcing isolation.

Specifically, the mechanism developed was based in Software Defined Networks architecture, namely OpenFlow based SDN, and targets both single and multi – tenant infrastructures. The aforementioned mechanism is based on the signaling traffic natively exchanged between elements in SDN architectures and is integrated with a sampling extension using sFlow. Our goal is to make the monitoring data gathered accessible on demand, in a personalized fashion. Finally, as mentioned we strive to make the proposed implementation adaptable. As a part of this effort, adaptations based on Open vSwitch are introduced, in order to allow our mechanism to be compatible with both cloud and wireless facilities – research platforms.

This mechanism can be used as the foundation for the creation of a unified framework for collecting network monitoring data, and automatically presenting them to the corresponding tenants of various experimental infrastructures as a part of the environment functionality using an automated and abstracted procedure.

Keywords: <<Software Defined Networks, OpenFlow, Network Monitoring, Multi – tenant environments, Heterogeneous environments, Network Virtualization>>

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

1

Εισαγωγή

1.1 Αντικείμενο

Η ανάπτυξη του Internet και των εικονικών υποδομών (virtual infrastructures) προσέφερε τη δυνατότητα διαμοιρασμού των δικτυακών πόρων στους χρήστες-ενοικιαστές (tenants) μεγάλης κλίμακας πειραματικών υποδομών (testbeds). Τέτοιες πλατφόρμες χρησιμοποιούνται για τον πειραματισμό, την ενσωμάτωση και την επικύρωση της επόμενης γενιάς των τεχνολογιών δικτύων (Future Internet Research and Experimentation – FIRE).

Στα πλαίσια της παρούσας διπλωματικής εργασίας μελετώνται μηχανισμοί και εργαλεία για την ανάπτυξη μεθόδων συλλογής δικτυακών δεδομένων (network monitoring data) από ετερογενή περιβάλλοντα. Λαμβάνοντας υπ όψιν την ποικιλομορφία των δικτυακών υποδομών που συναντούμε στις μέρες μας, βαρύνουσα σημασία έχει η ικανότητα προσαρμογής του μηχανισμού σε αυτές. Αν πρόκειται για υποδομές πολλαπλών ενοίκων, ο καθένας πρέπει να έχει πρόσβαση στα δεδομένα που συλλέγονται από πόρους που του έχουν εκμισθωθεί. Επιπροσθέτως, είναι επιθυμητό τα δεδομένα που αντλούνται να παρουσιάζονται στον εκάστοτε ενδιαφερόμενο απομονωμένα από αυτά που δεν τον αφορούν. Τέτοια δεδομένα είναι εξαιρετικά χρήσιμα για τους χρήστες των πειραματικών υποδομών, παρέχοντάς τους δυνατότητες όπως: (1) αξιολόγηση νέων μεθόδων και πρωτοκόλλων διαχείρισης της δικτυακής κίνησης, και (2) ανάπτυξη μηχανισμών ανίχνευσης και αντιμετώπισης περιστατικών δικτυακών επιθέσεων (Anomaly Detection and Attack Mitigation).

Πιο συγκεκριμένα, ο τρόπος λειτουργίας του μηχανισμού στηρίζεται στην αρχιτεκτονική Δικτύων Οριζόμενων από Λογισμικό (Software Defined Networks – SDN) βασισμένων στο πρωτόκολλο OpenFlow και στην πραγματοποίηση δειγματοληψίας με χρήση του προτύπου sFlow. Η υποδομή μπορεί να υποστηρίζει έναν ή πολλαπλούς χρήστες. Το δεύτερο επιτυγχάνεται χρησιμοποιώντας την πλατφόρμας εικονικοποίησης FlowVisor. Σημειώνεται πως, τα δεδομένα που συλλέγονται από τον μηχανισμό πρέπει να είναι προσβάσιμα από κάθε χρήστη – ένοικο, αλλά μόνο για δεδομένα που αντιστοιχούν σε πόρους που του έχουν εκμισθωθεί.

Επιπρόσθετα, παρουσιάζεται μια επέκταση της βασικής αρχιτεκτονικής μέσω του Open vSwitch. Χρησιμοποιώντας τις δυνατότητες του Open vSwitch (και ειδικά την δημιουργία OpenFlow – Enabled, λογικών switches) καταφέρνουμε να ενσωματώσουμε τον μηχανισμό που υλοποιήθηκε σε διάφορες ετερογενείς υποδομές οι οποίες δεν είναι βασισμένες στην αρχιτεκτονική SDN, προσφέροντας έναν ενιαίο τρόπο για την άντληση δικτυακών δεδομένων από αυτές.

1.2 Οργάνωση

Η παρούσα διπλωματική εργασία μπορεί να διαχωριστεί σε δύο τμήματα. Το πρώτο τμήμα συνίσταται στην ανάλυση της αρχιτεκτονικής SDN, του πρωτοκόλλου OpenFlow και άλλων προτύπων ή πρωτοκόλλων χρήσιμων ως υπόβαθρο για την κατανόηση της διπλωματικής εργασίας. Το δεύτερο τμήμα αποτελεί και το πρακτικό κομμάτι της εργασίας, όπου περιλαμβάνονται η υλοποίηση, οι βασικές αρχές πάνω στις οποίες στηρίχθηκε, η αξιολόγηση της καθώς και ο επίλογος.

Πιο συγκεκριμένα, το πρώτο τμήμα απαρτίζεται εξ ολοκλήρου από το **Κεφάλαιο 2**. Σε αυτό το κεφάλαιο αναλύεται η SDN αρχιτεκτονική, το πρωτόκολλο OpenFlow, ο Controller POX, οι πλατφόρμες εικονικοποίησης FlowVisor και OpenVirtex, καθώς και όσον αφορά την παρακολούθηση – συλλογή δικτυακών δεδομένων (1) το πρότυπο sFlow και (2) υπάρχουσες προσεγγίσεις βασισμένες σε SDN αρχιτεκτονικές.

Το δεύτερο τμήμα περιέχει τα υπόλοιπα τέσσερα κεφάλαια:

- **Κεφάλαιο 3:** Παρουσιάζεται η βασική αρχιτεκτονική δομή του μηχανισμού συλλογής δεδομένων καθώς και ο γενικός τρόπος λειτουργίας.
- **Κεφάλαιο 4:** Αρχικά, παρουσιάζεται το πρωτόκολλο JSON – RPC στο οποίο είναι βασισμένη η υλοποίηση του μηχανισμού. Επίσης, παρατίθεται ακριβής ανάλυση της υλοποίησης προσδιορίζοντας με ακρίβεια τις ενέργειες που επιτελεί κάθε αρχείο στα πλαίσια του μηχανισμού.
- **Κεφάλαιο 5:** Αναλύεται η πειραματική διαδικασία. Παρουσιάζεται η διάταξη και τα αποτελέσματα για διάφορα σενάρια, επίσης προτείνονται προσαρμογές του μηχανισμού για ετερογενείς υποδομές.
- **Κεφάλαιο 6:** Συνοψίζονται τα συμπεράσματα που εξήχθησαν στα πλαίσια της παρούσας διπλωματικής εργασίας και προτείνονται επιπλέον βελτιώσεις και επεκτάσεις.

2

Θεωρητικό Υπόβαθρο

Στο συγκεκριμένο κεφάλαιο αναλύεται το θεωρητικό υπόβαθρο (πρωτόκολλα, πρότυπα, μηχανισμοί και εφαρμογές), πάνω στο οποίο στηρίζεται η παρούσα διπλωματική εργασία.

2.1 Δικτύωση Υπολογιστών στην Σύγχρονη Εποχή

Η σύγχρονη εποχή έχει αναδείξει και συνεχίζει διαρκώς να αναδεικνύει διαφορετικές ανάγκες όσον αφορά τον τρόπος δικτύωσης των υπολογιστικών συστημάτων. Η ραγδαία αύξηση στην χρήση των φορητών συσκευών και στις υπηρεσίες που παρέχονται από τα υπολογιστικά νέφη (Cloud Services), η ανάλυση μεγάλου συνόλου δεδομένων (big data), καθώς και ο όγκος περιεχομένου που μεταδίδεται στις δικτυακές υποδομές οδηγούν την βιομηχανία στην επανεξέταση των παραδοσιακών αρχιτεκτονικών δικτύωσης, με στόχο την επιτυχή διαχείριση:

- *Διαρκώς εναλλασσόμενων μοτίβων κίνησης*: Η κίνηση εντός των data centers παρουσιάζει διάφορες διακυμάνσεις, σε αντίθεση με το μοντέλο client – server όπου ο κύριος όγκος της επικοινωνίας παρατηρούνταν μεταξύ ενός «πελάτη» και ενός εξυπηρετητή.
- *Διάδοση Cloud Based υπηρεσιών*: Παρατηρείται ιδιαίτερη αύξηση στην χρήση υπηρεσιών βασισμένων σε υπολογιστικά νέφη (τόσο δημόσια όσο και ιδιωτικά), από επιχειρήσεις και οργανισμούς. Έτσι απαιτείται η πρόσβαση κατά παραγγελία στις υποδομές και τις εφαρμογές που οι υποδομές αυτές προσφέρουν, με ευελιξία και ταυτόχρονα ασφάλεια.
- *Big Data συνεπάγεται αύξηση του Bandwidth*: Η επεξεργασία και ανάλυση μεγάλου συνόλου δεδομένων απαιτεί παράλληλη επεξεργασία σε πλήθος μηχανημάτων. Συνεπώς, επιτακτική είναι και η ανάγκη για αύξηση του εύρους ζώνης στις γραμμές μεταξύ των μηχανημάτων.

Επισημαίνεται πως στις υπάρχουσες αρχιτεκτονικές δικτύων, κάθε δικτυακή συσκευή αποτελεί ένα αυτόνομο στοιχείο. Στο εσωτερικό της κάθε συσκευής σχεδιάζονται και υλοποιούνται οι απαραίτητες λειτουργίες προώθησης, ελέγχου και διαχείρισης σε διακριτές ομάδες. Οι ομάδες αυτές είναι οι **διαστάσεις μετάδοσης δεδομένων**, **ελέγχου** και **διαχείρισης** (**data**, **control** και **management plane** αντιστοίχως).

Data Plane

Με τον όρο data ή forwarding plane αναφερόμαστε στα στοιχεία της συσκευής που είναι υπεύθυνα για την προώθηση και επεξεργασία της συσκευής με βάση κάποια καθορισμένη διαδικασία.

Control Plane

Το control plane αποτελεί το κομμάτι του δικτύου που μεταφέρει την σηματοδοσία και είναι υπεύθυνο για την διαμόρφωση και διαχείριση του τρόπου λειτουργίας του Data Plane (αρμόδιο δηλαδή για την δημιουργία της λογικής προώθησης – forwarding logic)

Management Plane

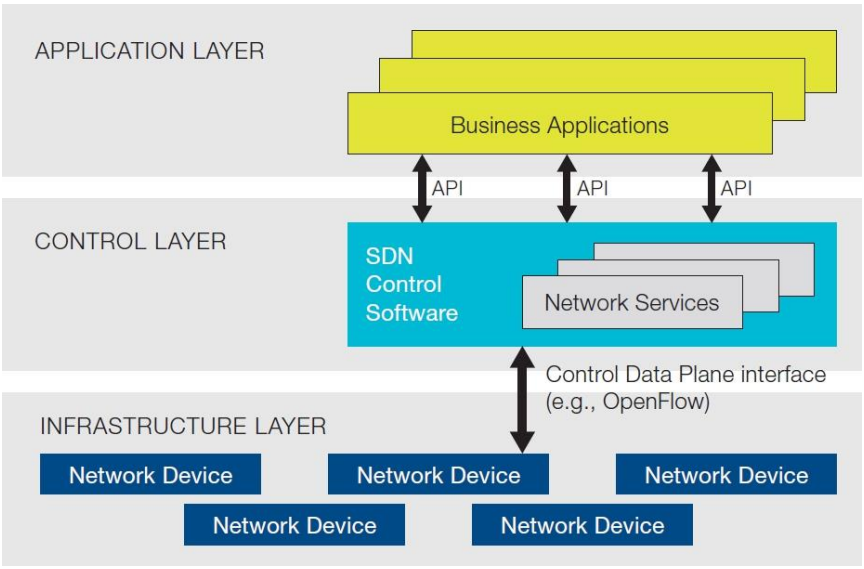
Μέρος του management plane, είναι οι ενέργειες που αφορούν την διαχείριση ενός δικτύου δεδομένων και επηρεάζουν τον τρόπο λειτουργίας των άλλων διαστάσεων. Το κύριο μοντέλο αναφοράς για την διαχείριση είναι το **FCAPS** (Fault, Configuration, Accounting, Performance, Security).

Επιπρόσθετα, σε αντιπαραβολή με τις τάσεις και τις ανάγκες που δημιουργούνται, οι εν ενεργεία τεχνολογίες δικτύωσης δεν έχουν σχεδιασθεί ώστε να μπορούν να ανταπεξέλθουν αποδοτικά σε αυτές. Κάποιοι περιορισμοί που συναντούμε στις σημερινές υποδομές είναι:

- *Πολυπλοκότητα που οδηγεί σε στασιμότητα*: Οι τεχνολογίες δικτύωσης απαρτίζονται κυρίως από συγκεκριμένα σύνολα πρωτοκόλλων, τα οποία έχουν στόχο να συνδέουν μηχανήματα με αξιοπιστία, ανεξαρτήτως απόστασης, ταχύτητας ζεύξης ή μορφής της τοπολογίας. Τα πρωτόκολλα ορίζονται ανεξάρτητα και κατά κύριο λόγο καθένα επιλύει ένα συγκεκριμένο πρόβλημα, χωρίς να παρέχεται κάποιο θεμελιώδες επίπεδο αφαίρεσης για λόγους ευκολίας. Έτσι οποιαδήποτε τροποποίηση στο δίκτυο απαιτεί μια σειρά πολύπλοκων αλλαγών στις ρυθμίσεις του δικτύου, κάτι που είναι σε πλήρη αντιδιαστολή με την δυναμική φύση που χαρακτηρίζει τις σύγχρονες υποδομές.
- *Εξάρτηση από τον κατασκευαστή*: Καθώς οι δικτυακές συσκευές προέρχονται από διάφορους κατασκευαστές, η ορθή παραμετροποίηση τους δεν είναι απλή υπόθεση. Η εκάστοτε συσκευή έχει εκ φύσεως συγκεκριμένο τρόπο ρύθμισης. Επίσης, αναλόγως τον κατασκευαστή επιτρέπεται έλεγχος με διαφορετικό τρόπο σε κάθε στοιχείο που την απαρτίζει (απουσία κάποιας ανοιχτής, κοινώς αποδεκτής διεπαφής – Open Interface). Συνεπώς ένας διαχειριστής πρέπει να ξέρει τον τρόπο παραμετροποίησης συσκευών για κάθε κατασκευαστή.
- *Αδυναμία κλιμακωσιμότητας*: Όσο μεγαλώνουν οι απαιτήσεις στις σύγχρονες υποδομές, αντίστοιχα μεγαλώνει και το μέγεθος του δικτύου. Ως αποτέλεσμα, οι διαδικασίες που φαίνονται απλές είναι δύσκολο να υλοποιηθούν σε μεγάλα δίκτυα. Επίσης , η παραμετροποίηση γίνεται για την κάθε συσκευή αυτόνομα καθώς το Control Plane είναι διανεμημένο σε κάθε συσκευή (distributed Control Plane). Κάτι τέτοιο δυσχεραίνει την όλη διαδικασία για μεγάλα δίκτυα με πολλές συσκευές.

2.2 SDN Architectures

Αυτή η αναντιστοιχία ανάμεσα στις τρέχουσες ανάγκες και τις δυνατότητες των δικτυακών συσκευών έχει ως αποτέλεσμα την δημιουργία μιας νέας αρχιτεκτονικής και την ανάπτυξη των σχετικών προτύπων. Η αρχιτεκτονική αυτή ονομάζεται **«Δικτύωση Οριζόμενη από Λογισμικό»** (**Software Defined Networking**) και βασίζεται στην αποσύνδεση του έλεγχου από την διαδικασία της προώθησης, καθώς επίσης στην προγραμματισημότητα του ελέγχου κατά άμεσο τρόπο. Συμπερασματικά, η μεταφορά του ελέγχου σε προσβάσιμες υπολογιστικές συσκευές, διαδικασία που στο παρελθόν πραγματοποιούνταν αυτόνομα σε κάθε δικτυακή συσκευή, δίνει την δυνατότητα σε εφαρμογές και υπηρεσίες δικτύου να έχουν μια αφαιρετική εποπτεία για την υποκείμενη δικτυακή υποδομή, αντιμετωπίζοντας την σαν μια λογική οντότητα.

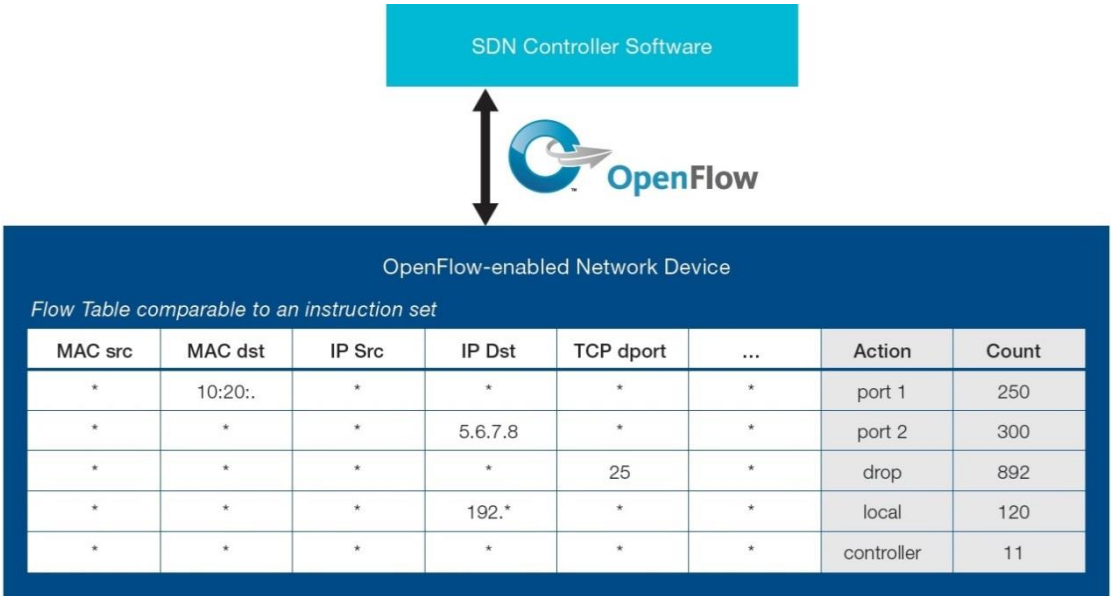


Σχήμα 2.1 Software – Defined Network Architecture, Πηγή [1]

Στο Σχήμα 2.1 παρουσιάζεται η δομή της SDN αρχιτεκτονικής. Η ευφυΐα του δικτύου (έλεγχος) είναι κεντρικοποιημένη με χρήση controllers βασισμένων σε λογισμικό οι οποίοι διατηρούν μια ολική όψη του δικτύου. Ως αποτέλεσμα μέσω του controller API το δίκτυο είναι αντιληπτό από τις εφαρμογές σαν ένα μοναδικό λογικό switch (μεταγωγέας). Η χρήση αυτής της αρχιτεκτονικής δίνει τη δυνατότητα ελέγχου του δικτύου από ένα συγκεκριμένο λογικό σημείο ανεξαρτήτως του κατασκευαστή, κάτι που απλοποιεί τον σχεδιασμό και την λειτουργία του δικτύου. Επίσης απλοποιείται ο τρόπος λειτουργίας των δικτυακών συσκευών, καθώς δεν απαιτείται πλέον από αυτές η επεξεργασία και κατανόηση των δικτυακών πρωτοκόλλων. Απλώς χρειάζεται, η αποδοχή οδηγιών για την επεξεργασία και προώθηση της κίνησης από το λογισμικό ελέγχου SDN.

Υπάρχουν διάφορες προσεγγίσεις για την υλοποίηση Software Defined Networks. Η επικρατέστερη, ευρέως διαδεδομένη υλοποίηση, η οποία επίσης χρησιμοποιήθηκε στα πλαίσια της εργασίας είναι το πρωτόκολλο OpenFlow ([2]). Το πρωτόκολλο OpenFlow

αποτελεί ένα ανοικτό πρότυπο για την διεπαφή επικοινωνίας μεταξύ του στρώματος «ελέγχου» και «προώθησης» της αρχιτεκτονικής SDN, επιτρέποντας άμεση πρόσβαση και διαχείριση του forwarding plane των δικτυακών συσκευών. Όπως φαίνεται και στο ακόλουθο σχήμα (Σχήμα 2.2), ο τρόπος λειτουργίας του πρωτοκόλλου μπορεί να συγκριθεί με το σύνολο εντολών (instruction set). Διατίθενται κάποιες βασικές πρωτογενείς εντολές, οι οποίες μπορούν να χρησιμοποιηθούν από εξωτερικές προς το πρωτόκολλο εφαρμογές για τον προγραμματισμό του data plane.



Σχήμα 2.2 Παράδειγμα του Instruction Set του OpenFlow, Πηγή [1]

Επιπρόσθετα, για την υλοποίηση του βασικού τρόπου λειτουργίας του πρωτοκόλλου χρησιμοποιείται η έννοια των flows (ροές πακέτων), με βάση την οποία πραγματοποιείται η αναγνώριση και διαχείριση της κίνησης εφαρμόζοντας προκαθορισμένους κανόνες (flow rules) οι οποίοι εγκαθίστανται στατικά ή δυναμικά. Καθώς το OpenFlow επιτρέπει τον προγραμματισμό του δικτύου κατά flows, μια αρχιτεκτονική SDN βασισμένη σε αυτό επιτρέπει έναν εξαιρετικά ευέλικτό τρόπο διαχείρισης της κίνησης. Ενδεικτικό παράδειγμα είναι ο κυρίαρχος τρόπος δρομολόγησης, ο οποίος βασίζεται στην διεύθυνση IP προορισμού. Δεν υπάρχει η δυνατότητα διαφορετικά flows μεταξύ δυο κόμβων, να χαίρουν διαφορετικής μεταχείρισης ως προς την δρομολόγηση. Πάντα θα ακολουθείται η ίδια διαδρομή ανεξαρτήτως διαφορετικών απαιτήσεων.

Συγκεντρωτικά, τα πλεονεκτήματα που προσφέρει η χρήση αρχιτεκτονικής SDN βασιζόμενων στο OpenFlow περιλαμβάνουν:

- **Κεντρικοποιημένο έλεγχο:** Το λογισμικό ελέγχου είναι σε θέση να προσφέρει έλεγχο του συνόλου του δικτύου επιτρέποντας την μαζική – κεντρική διαχείριση του δικτύου ανεξαρτήτως κατασκευαστή της κάθε συσκευής.
- **Μειωμένη πολυπλοκότητα:** Προσφέρεται ένα ευέλικτος τρόπος για την διαχείριση του δικτύου, ο οποίος σε συνδυασμό με την δημιουργία εργαλείων/εφαρμογών, επιτρέπει την αυτοματοποίηση και απλοποίηση πολλών διεργασιών οι οποίες πραγματοποιούνται χειροκίνητα, μειώνοντας το κόστος και μεγιστοποιώντας την ικανότητα προσαρμογής του δικτύου.

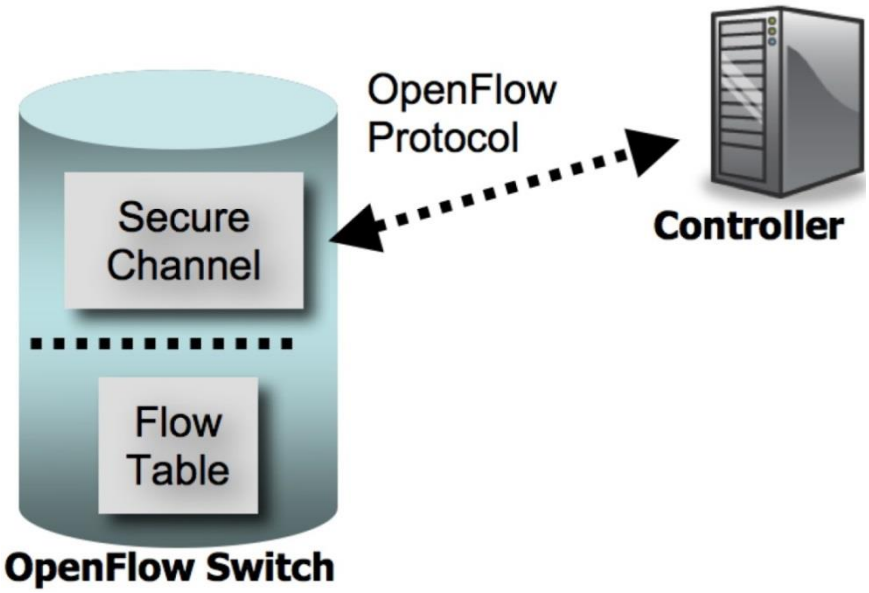
- **Αυξημένη αξιοπιστία και ασφάλεια:** Δίνεται η δυνατότητα στους διαχειριστές να διαμορφώσουν πολιτικές ή να πραγματοποιήσουν ρυθμίσεις υψηλού επιπέδου. Κατ επέκταση μπορούν να δημιουργηθούν προηγμένες εφαρμογές για την ασφάλεια και αξιοπιστία του δικτύου (ανίχνευση/αντιμετώπιση επιθέσεων, εφεδρικές διαδρομές).
- **Ευέλικτός (Granular) έλεγχος:** Ο τρόπος ελέγχου στο πρωτόκολλο OpenFlow, βασισμένος σε ροές πακέτων, δίνει τη δυνατότητα για εφαρμογή ιδιαίτερα ευέλικτων πολιτικών , διευκολύνοντας τους διαχειριστές να εξυπηρετήσουν τις σύγχρονες ανάγκες κατά το βέλτιστο τρόπο.
- **Ευκολία στην δημιουργία καινοτομιών:** Ο τρόπος λειτουργίας του πρωτοκόλλου δίνει σε κάθε διαχειριστή την δυνατότητα να αναπτύξει δικά του εξαρτήματα/ολοκληρωμένες εφαρμογές για τον έλεγχο του δικτύου διευκολύνοντας την δημιουργία καινοτόμων συστημάτων ελέγχου.
- **Βέλτιστη εμπειρία χρήστη:** Συσσωρεύοντας τον έλεγχο του δικτύου και επιτρέποντας πρόσβαση σε υψηλότερου επιπέδου εφαρμογές, σε πληροφορίες σχετικά με την κατάσταση του δικτύου μια υποδομή βασισμένη στην αρχιτεκτονική SDN μπορεί να ανταποκριθεί με τον βέλτιστο τρόπο στις ανάγκες του κάθε χρήστη.

2.3 OpenFlow Protocol

Αφού έχουμε προσδιορίσει την γενικότερη αρχιτεκτονική δομή, στη συνέχεια θα αναλυθεί ενδελεχώς ο τρόπος λειτουργίας του πρωτοκόλλου OpenFlow σε τεχνικό επίπεδο. Σημειώνεται πως αρχικά θα αναφερθούμε στην έκδοση 1.0 του πρωτοκόλλου.

2.3.1 *Switch*

Ένα OpenFlow Switch αποτελείται από ένα *flow table*, με βάση το οποίο γίνεται η προώθηση, καθώς και ενός ασφαλούς διαύλου επικοινωνίας (*Secure Channel*). Αρμόδιος για την διαχείριση του switch είναι ο Controller και η επικοινωνία γίνεται με χρήση του OpenFlow πρωτοκόλλου.



Σχήμα 2.3 Ένα OpenFlow switch επικοινωνεί με τον Controller μέσω δίαυλου ασφαλής επικοινωνίας, Πηγή [3]

Το flow table περιέχει εγγραφές σχετικές με τιμές κεφαλίδων, μετρητές και το σύνολο το ενεργειών που θα εφαρμοστούν στην περίπτωση αντιστοίχισης. Τα πακέτα επεξεργάζονται από το Switch και αντιπαραβάλλονται με το flow table. Σε περίπτωση που αντιστοιχισθεί κάποια εγγραφή, εφαρμόζεται το σύνολο των ενεργειών σε αυτό το πακέτο. Αντίθετα, εάν δεν υπάρχει αντιστοίχιση το πακέτο προωθείται στον Controller μέσω του Secure Channel, και στη συνέχεια εμπλέκεται ενεργά στην διαδικασία προώθησης, μεταβάλλοντας το flow table ή/και πραγματοποιώντας κάποιες ενέργειες.

2.3.2 *Flow Table*

Περισσότερες λεπτομέρειες για τα δομικά του flow table, καθώς και της διαδικασίας αντιστοίχισης (Πίνακας 2.1):

Header Fields	Counters	Actions

Πίνακας 2.1 Εγγραφή Flow Table (απαρτίζεται από Header Fields, Counters, Actions), Πηγή [3]

- Header fields: (πεδία κεφαλίδων) αντιπαραβολή εισερχομένων πακέτων.
- Counters: ανανέωση για κάθε πακέτο που αντιστοιχίζεται.
- Actions: εφαρμογή ενεργειών, στα πακέτα που αντιστοιχίζονται.

2.3.2.1 Header Fields

Παρατίθενται (Πίνακας 2.2) τα Header Fields των εγγραφών του Flow Table, με τα οποία συγκρίνονται τα ομότιμα πεδία του κάθε εισερχόμενου πακέτου. Δίνεται έμφαση στο γεγονός ότι όλα τα πεδία περιέχουν μια συγκεκριμένη τιμή ή εναλλακτικά τιμή **ANY** που αντιστοιχίζεται με οποιαδήποτε τιμή πακέτου (wildcard).

Ingress Port	Ether source	Ether dst	Ether type	VLAN id	VLAN Priority	IP src	IP dst	IP proto	IP ToS	TCP src	TCP dst

Πίνακας 2.2 Πεδία του πακέτου που χρησιμοποιούνται για την αντιστοίχιση, Πηγή [3]

Περαιτέρω αναλύονται τα Header Fields (Πίνακας 2.3).

Field	Bits	Applicable	Notes
Ingress Port	(Εξαρτάται από την υλοποίηση)	All Packets	Αριθμητική αναπαράσταση από το 1
Ethernet Source Address	48	All packets on enabled ports	
Ethernet destination	48	All packets on	

address		enabled ports	
Ethernet Type	16	All packets on enabled ports	
VLAN id	12	Ethernet type 0x8100	
VLAN Priority	3	Ethernet type 0x8100	VLAN PCP field
IP source address	32	IP and ARP	Υποστηρίζονται υποδίκτυα
IP destination address	32	IP and ARP	Υποστηρίζονται υποδίκτυα
IP Protocol	8	IP and ARP	Χρήση των 8 χαμηλότερων bits του ARP opcode
IP ToS	6	IP	
Transport source port / ICMP Type	16	TCP, UDP, ICMP	Χρήση των 8 χαμηλότερων bits για το ICMP Type
Transport destination port / ICMP Code	16	TCP, UDP, ICMP	Χρήση των 8 χαμηλότερων bits για το ICMP Code

Πίνακας 2.3 Μήκος πεδίων και πότε είναι εφαρμόσιμα, Πηγή [3]

2.3.2.2 Counters

Οι μετρητές διατηρούνται και ανανεώνονται για κάθε Flow Table, κάθε Flow, κάθε port του switch και κάθε ουρά αναμονής. Αναλυτικά φαίνονται οι διαθέσιμοι μετρητές στον Πίνακα 2.4.

Counter	**Bits**
Per Table	
Active Entries	32
Packet Lookups	64
Packet Matches	64
Per Flow	
Received Packets	64
Received Bytes	64
Duration (seconds)	32
Duration (nanoseconds)	32

Per Port	
Received Packets	64
Transmitted Packets	64
Received Bytes	64
Transmitted Bytes	64
Receive Drops	64
Transmit Drops	64
Receive Errors	64
Transmit Errors	64
Receive Frame Alignment Errors	64
Receive Overrun Errors	64
Receive CRC Errors	64
Collisions	64
Per Queue	
Transmit Packets	64
Transmit Bytes	64
Transmit Overrun Errors	64

Πίνακας 2.4 Πίνακας με τους Counters, Πηγή [3]

2.3.2.3 Actions

Κάθε εγγραφή όπως αναφέρθηκε σχετίζεται με ένα σύνολο ενεργειών. Το σύνολο μπορεί να περιέχει καμία ενέργεια ή περισσότερες, οι οποίες εφαρμόζονται πάνω στα αντίστοιχα πακέτα. Εάν καμία από τις ενέργειες δεν αφορά την προώθηση (FORWARD) το πακέτο απορρίπτεται (DROP). Σημειώνεται πως οι ενέργειες πρέπει να εφαρμοσθούν με την αλληλουχία που προβλέπεται.

Ένα OpenFlow switch, ενδέχεται να απορρίψει μια εγγραφή, εάν δεν είναι σε θέση να επεξεργαστεί το σύνολο των ενεργειών της. Γενικότερα ένα switch δεν είναι απαραίτητο να υποστηρίζει όλες τους τύπους ενεργειών. Οφείλει ωστόσο να υποστηρίζει αυτούς που χαρακτηρίζονται "Required Actions", οι οποίοι είναι αναγκαίοι για τις βασικές λειτουργίες. Κατά την σύνδεση με τον Controller, το switch υποδηλώνει ποιές εκ των προαιρετικών – "Optional Actions" υποστηρίζει.

Τα συμβατά Switches με το πρωτόκολλο OpenFlow, χωρίζονται σε δυο κατηγορίες:

- OpenFlow – only: περιορίζονται στην υποστήριξη "Required Actions".
- OpenFlow – enabled: συσκευές που ενδέχεται να υποστηρίζουν επίσης τις ενέργειες "NORMAL".

Επιπρόσθετα και οι δυο κατηγορίες μπορούν να υποστηρίξουν την ενέργεια "FLOOD".

Required Action: Forward

Προώθηση πακέτων τόσο στις φυσικές πόρτες (physical ports) όσο και στις ακόλουθες εικονικές:

- **ALL:** Προώθηση του πακέτου σε όλες τις διεπαφές, εξαιρούμενη αυτής της οποίας προήλθε.
- **CONTROLLER:** Ενθυλάκωση του πακέτου και προώθηση στον Controller.
- **LOCAL:** Προώθηση του πακέτου στο local networking stack του ίδιου του switch.
- **TABLE:** Ενέργειες πάνω στο flow table, μόνο για packet – out μηνύματα.
- **IN_PORT:** Προώθηση του πακέτου από την πόρτα που εισήχθη.

Required Action: Drop

Μια εγγραφή χωρίς ενέργειες, υποδηλώνει πως όλα τα αντιστοιχιζόμενα πακέτα πρέπει να απορριφθούν.

Optional Actions: Forward

Προαιρετικά το switch μπορεί να υποστηρίζει την προώθηση πακέτων και στις ακόλουθες εικονικές πόρτες:

- **NORMAL:** Επεξεργασία των πακέτων με χρήση του βασικού τρόπου προώθησης που υποστηρίζεται από το switch.
- **FLOOD:** Προώθηση του πακέτου με βάση το ελάχιστο διατρέχων δέντρο (minimum spanning tree), εξαιρώντας το interface εισόδου.

Optional Actions: Enqueue

Η ενέργεια αυτή προωθεί το πακέτο μέσω μιας ουράς αναμονής που λειτουργεί σε μια πόρτα. Η προώθηση εξαρτάται από την παραμετροποίηση της ουράς.

Optional Actions: Modify – Field

Παρότι η τροποποίηση των πεδίων δεν είναι ενέργεια που υποστηρίζεται αναγκαστικά, παρουσιάζει ιδιαίτερο ενδιαφέρον καθώς βελτιώνει σε μεγάλο βαθμό την χρησιμότητα και τις δυνατότητες του OpenFlow switch. Οι ενέργειες παρατίθενται στον Πίνακα 2.5.

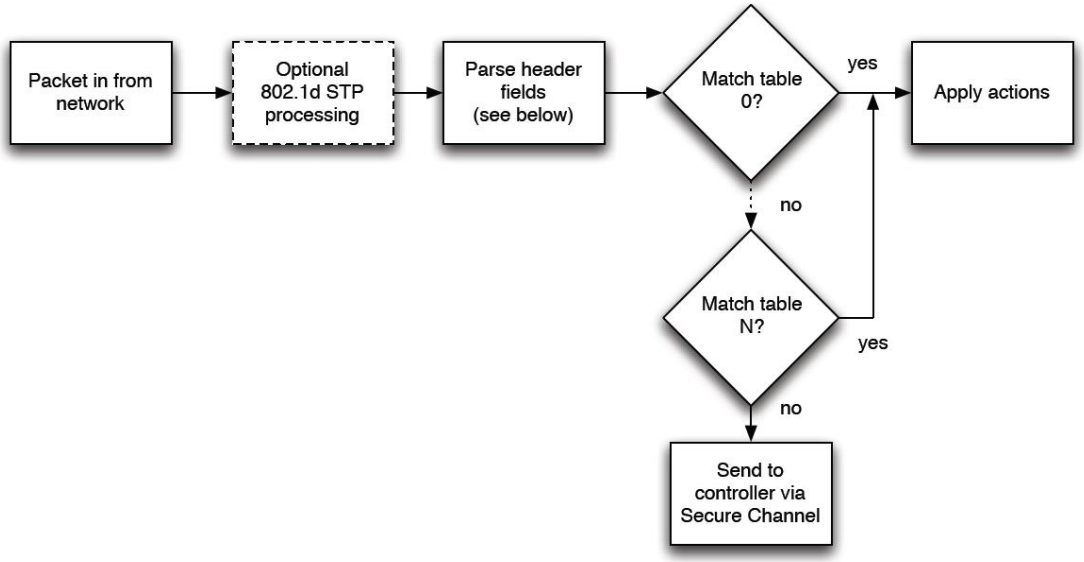
Action	**Associated Data**	**Description**
Set VLAN ID	12 bits	Εάν δεν υπάρχει VLAN, προστίθεται μια νέα κεφαλίδα με το VLAN ID του ορίσματος και προτεραιότητα 0. Εάν υπάρχει κεφαλίδα VLAN, αντικαθίσταται το VLAN ID με αυτό του ορίσματος

Set VLAN priority	3 bits	Εάν δεν υπάρχει VLAN, προστίθεται μια νέα κεφαλίδα με την προτεραιότητα του ορίσματος και VLAN ID 0. Εάν υπάρχει κεφαλίδα VLAN, αντικαθίσταται η προτεραιότητα με αυτή του ορίσματος
Strip VLAN header	-	Αφαίρεση της κεφαλίδας VLAN
Modify Ethernet source MAC address	48 bits: τιμή για την αντικατάσταση της MAC address	Αντικατάσταση της υπάρχουσας MAC πηγής με τη νέα τιμή
Modify Ethernet destination MAC address	48 bits: τιμή για την αντικατάσταση της MAC address	Αντικατάσταση της υπάρχουσας MAC προορισμού με τη νέα τιμή
Modify IPv4 source address	32 bits: τιμή για την αντικατάσταση της υπάρχουσας διεύθυνσης IPv4	Αντικατάσταση της υπάρχουσας IP πηγής με τη νέα τιμή και ανανέωση του IP checksum (και TCP/UDP checksum)
Modify IPv4 destination address	32 bits: τιμή για την αντικατάσταση της υπάρχουσας διεύθυνσης IPv4	Αντικατάσταση της υπάρχουσας IP πηγής με τη νέα τιμή και ανανέωση του IP checksum (και TCP/UDP checksum)
Modify IPv4 ToS bits	6 bits: τιμή για την αντικατάσταση του πεδίου ToS	Αντικατάσταση του πεδίου ToS με τη νέα τιμή
Modify transport source port	16 bits: τιμή για την αντικατάστασης της υπάρχουσας πόρτας TCP/UDP	Αντικατάσταση της υπάρχουσας TCP/UDP port με τη νέα τιμή και ανανέωση του checksum
Modify transport destination port	16 bits: τιμή για την αντικατάστασης της υπάρχουσας πόρτας TCP/UDP	Αντικατάσταση της υπάρχουσας TCP/UDP port με τη νέα τιμή και ανανέωση του checksum

Πίνακας 2.5 Modify Fields Actions, Πηγή [3]

2.3.3 *Matching*

Κατά την λήψη ενός πακέτου, ένα OpenFlow switch πραγματοποιεί τις ενέργειες που φαίνονται στο ακόλουθο διάγραμμα ροής (Σχήμα 2.4).



Σχήμα 2.4 Επεξεργασία ενός πακέτου, Πηγή [3]

- Προαιρετική επεξεργασία για το πρωτόκολλο 802.1d Spanning Tree Protocol.
- Σάρωση των κεφαλίδων.
- Αναζήτηση εγγραφής αντιστοίχησης.
- Εάν βρεθεί κάποια εφαρμόζονται οι ενέργειες στο σύνολο τους.
- Διαφορετικά, αποστολή του πακέτου στον Controller.

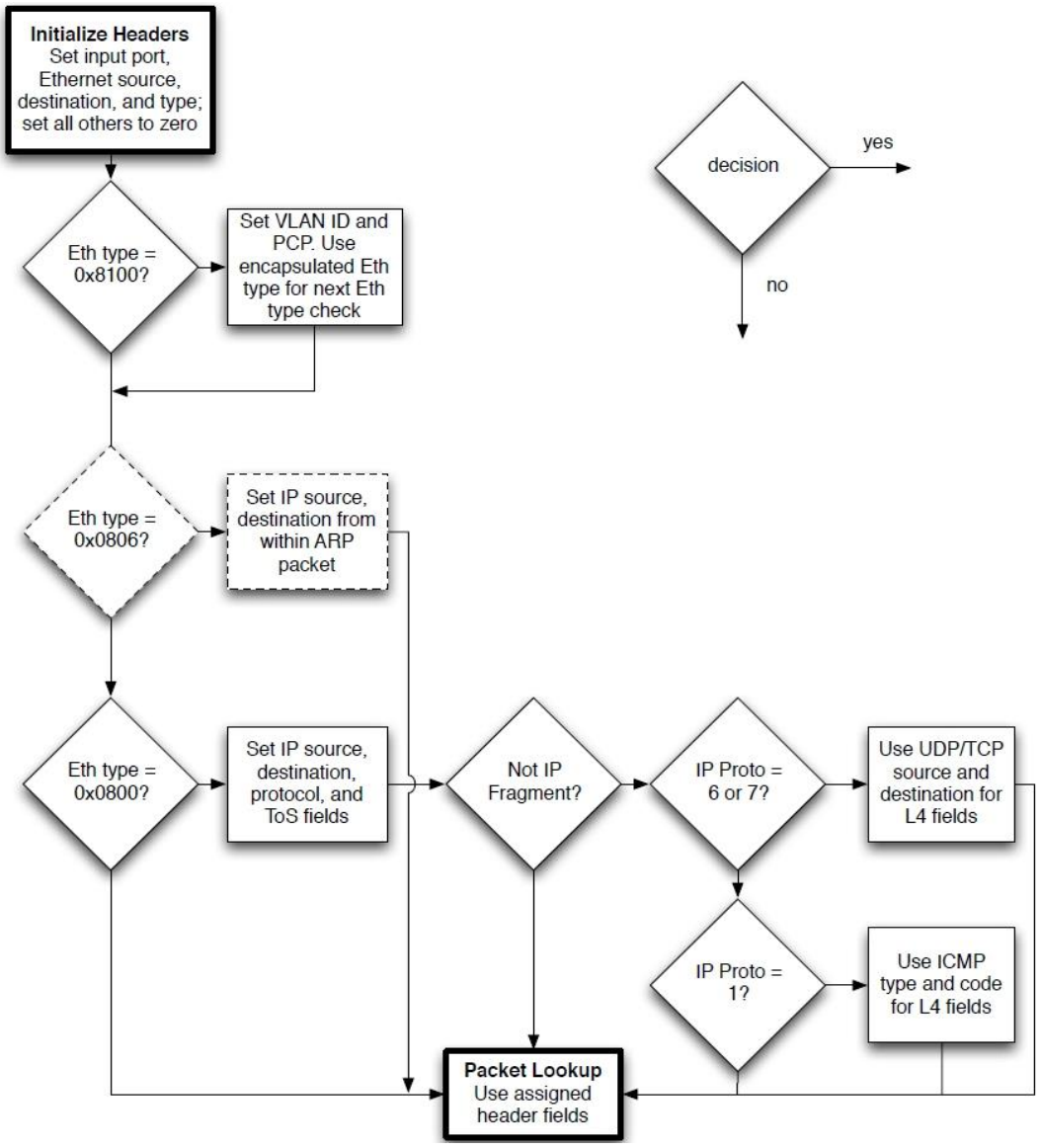
Ακολουθούν τα διαφορετικά πεδία που χρησιμοποιούνται για την διαδικασία της αναζήτησης στον πίνακα.

- Κανόνες που ορίζουν ένα Ingress port, συγκρίνονται με το αντίστοιχο port από το οποίο εισήλθε το πακέτο.
- Οι κεφαλίδες Ethernet, χρησιμοποιούνται για όλα τα πακέτα.
- Στην περίπτωση που το πακέτο έχει VLAN tag (Ethernet type 0x8100), χρησιμοποιείται το VLAN ID & VLAN PCP.
- Προαιρετικά για πακέτα ARP (Ethernet Type 0x0806), χρησιμοποιούνται επιπρόσθετα οι διευθύνσεις IP.
- Πακέτα IP (Ethernet type 0x0800), χρησιμοποιούνται επίσης και τα πεδία της επικεφαλίδας IP.
- Πακέτα IP που φέρουν TCP/UDP στο ανώτερο στρώμα (IP protocol 6/17 αντίστοιχα), η αναζήτηση περιλαμβάνει και τις πόρτες του στρώματος μεταφοράς.
- Πακέτα ICMP (IP Protocol 1), χρήση των πεδίων Type / Code.
- Πακέτα IP που έχον υποστεί θρυμματισμό, δεν χρησιμοποιούν πόρτες του επιπέδου μεταφοράς.

Ένα πακέτο αντιστοιχίζεται με μια εγγραφή του flow table αν οι διάφορες τιμές των κεφαλίδων του που χρησιμοποιούνται για την αναζήτηση αντιστοιχούν σε αυτές της εγγραφής. Εάν κάποιο πεδίο της εγγραφής έχει τιμή ANY, ταιριάζει με οποιαδήποτε τιμή κεφαλίδας. Σε περίπτωση αντιστοίχησης πραγματοποιούνται οι προβλεπόμενες ενέργειες

και ανανεώνονται οι αντίστοιχοι μετρητές για την συγκεκριμένη εγγραφή. Διαφορετικά το πακέτο αποστέλλεται στον controller.

Αξίζει να σημειωθεί πως τα πακέτα αντιστοιχίζονται σε εγγραφές με βάση κάποια προτεραιότητα. Ένα πακέτο μπορεί να αντιστοιχίζεται σε πολλές εγγραφές, αλλά η τελική αντιστοίχηση καθορίζεται από την προτεραιότητα. Μια εγγραφή με όλες τις τιμές προσδιορισμένες (χωρίς wildcards) έχει πάντα την υψηλότερη προτεραιότητα. Εγγραφές με wildcards διαθέτουν κάποια προτεραιότητα, με τις υψηλότερες προτεραιότητες να προηγούνται. Στην περίπτωση που πολλαπλές εγγραφές έχουν ίδια προτεραιότητα το switch έχει την ευχέρεια να επιλέξει την διάταξη τους.



Σχήμα 2.5 Διαδικασία αντιστοίχισης και ποιες κεφαλίδες χρησιμοποιούνται, Πηγή [3]

2.3.4 *Ανταλλαγή μηνυμάτων εντός του Openflow – Επισκόπηση του πρωτοκόλλου*

Όπως αναφέρθηκε, ο ασφαλής δίαυλος επικοινωνίας είναι η διεπαφή που συνδέει κάθε OpenFlow switch με τον controller. Μέσα από αυτή τη διεπαφή, ο controller ρυθμίζει και διαχειρίζεται το switch, δέχεται events από αυτό και στέλνει πακέτα μέσω αυτού.

Το switch πρέπει να αρχικοποιήσει την επικοινωνία μέσω του Secure Channel, ξεκινώντας μια σύνοδο (session) προς μια IP διεύθυνση και port. Η επικοινωνία αυτή δεν υπόκειται σε διαδικασία της αντιστοίχισης με τις εγγραφές του switch. Έτσι, ένα switch πρέπει να είναι σε θέση να ξεχωρίζει την υπόλοιπη δικτυακή κίνηση προκειμένου να ξεκινήσει την αντιπαραβολή με το flow table.

Ο δίαυλος υλοποιείται μέσω μιας συνόδου Transport Layer Security, προσφέροντας κρυπτογραφημένη επικοινωνία. Όπως αναφέρθηκε, η σύνοδος εγκαθιδρύεται από το switch και πραγματοποιείται η σύνδεση στην TCP port που αναμένει ο controller (συνήθως στην well known 6633). Εν συνεχεία, η κάθε πλευρά μπαίνει στην διαδικασίας πιστοποίησης του εταίρου συμβαλλόμενου μέσω της ανταλλαγής πιστοποιητικών υπογεγραμμένων από μια αρχή πιστοποίησης.

Όσον αφορά τα μηνύματα που ανταλλάσσονται, το πρωτόκολλο υποστηρίζει τρεις διαφορετικούς τύπους μηνυμάτων, όπου κάθε τύπος έχει υποκατηγορίες που θα αναλυθούν περαιτέρω:

- **Controller – to – switch:** Στέλνονται από τον controller και χρησιμοποιούνται για να διαχειριστούν ή επιθεωρήσουν το switch κατά άμεσο τρόπο.
- **Asynchronous:** Ξεκινούν από το switch με στόχο την ενημέρωση του controller είτε για γεγονότα του δικτύου, είτε για αλλαγές στην κατάσταση του δικτύου.
- **Symmetric:** Προέρχονται είτε από τον Controller είτε από το switch, χωρίς να τα ζητήσει συγκεκριμένα η άλλη πλευρά.

2.3.4.1 Controller-to-switch

Τα μηνύματα αποστέλλονται από τον controller και μπορεί να περιμένουν ή όχι απάντηση από το switch.

- **Features:** Κατά την εγκαθίδρυση της συνόδου, ο controller στέλνει στο switch ένα μήνυμα που ζητεί τα χαρακτηριστικά που υποστηρίζε (features request), στο οποίο απαντάει εν συνεχεία το switch παρέχοντας την λίστα με αυτά (features reply).
- **Configuration:** Ο Controller έχει την δυνατότητα να προβεί σε ρύθμιση των παραμέτρων του switch, ή να ζητήσει πληροφορίες σχετικά με τις ήδη υπάρχουσες ρυθμίσεις.
- **Modify – State:** Χρησιμοποιούνται για την διαχείριση της κατάστασης των switch. Κύριος στόχος τους είναι η προσθήκη, διαγραφή ή τροποποίηση των εγγραφών του flow table, καθώς και η ρύθμιση ιδιοτήτων των ports.
- **Read – State:** Χρησιμοποιούνται για την συλλογή στατιστικών δεδομένων από τα flow – tables, τις ports και συγκεκριμένα flow entries.

- **Send – Packet:** Χρησιμοποιούνται για την αποστολή πακέτων έξω από μια συγκεκριμένη port του switch, χωρίς αναγκαστικά να εγκαθίσταται κάποιος κανόνας στο table του switch.
- **Barrier:** Χρησιμοποιούνται είτε για να βεβαιωθεί πως οι εξαρτήσεις προηγούμενων μηνυμάτων έχουν ολοκληρωθεί πριν από την επεξεργασία επόμενων, είτε για την λήψη ειδοποιήσεων για ολοκληρωμένες ενέργειες

2.3.4.2 Asynchronous

Τα μηνύματα αποστέλλονται από τα switches χωρίς να ζητά ρητά ο controller. Κυρίως χωρίζονται στους ακόλουθους τύπους:

- **Packet – In:** Όλα τα πακέτα τα οποία δεν μπορούν να αντιστοιχισθούν σε κάποια εγγραφή, προκαλούν την αποστολή ενός γεγονότος Packet-in. Εάν το switch διαθέτει επαρκή διαθέσιμη μνήμη στον buffer ώστε να κρατήσει προσωρινά το πακέτο, τότε το μήνυμα περιλαμβάνει ένα τμήμα του πακέτου (από προεπιλογή τα πρώτα 128 bytes) καθώς και μια αναγνωριστική τιμή στον buffer. Σε περίπτωση μην υποστήριξης της προσωρινής αποθήκευσης ή μη επαρκούς μνήμης, στέλνεται ολόκληρο το πακέτο.
- **Flow – Removed:** Κατά την προσθήκη μιας εγγραφής στο flow table του switch από τον Controller, υπάρχουν δυο πεδία σχετικά με την αυτόματη αφαίρεση της. Όσον αφορά το πρώτο, μετά από ποιο χρονικό διάστημα αδράνειας (soft timeout), δηλαδή καμίας αντιστοίχιση πακέτων με αυτή, η εγγραφή αφαιρείται. Το δεύτερο προσδιορίζει μετά από ποιο χρονικό διάστημα θα αφαιρεθεί χωρίς να χρησιμοποιηθεί κάποιο άλλο κριτήριο (hard timeout). Στην εγγραφή επίσης υπάρχει μια σημαία η οποία ορίζει εάν η αφαίρεση πρέπει να προκαλέσει την αποστολή μηνύματος Flow–Removed στον controller.
- **Port – status:** Είναι αναμενόμενο από το switch να στέλνει τέτοια μηνύματα κατά την αλλαγή της κατάστασης σε ένα port. Παραδείγματος χάριν λόγω ενεργοποίησης/απενεργοποίησης από έναν χρήστη, ή αλλαγών όπως αυτές ορίζονται στο spanning tree protocol.
- **Error:** Χρησιμοποιούνται για την ενημέρωση του controller για τυχόν σφάλματα που αντιμετωπίσθηκαν.

2.3.4.3 Symmetric

Μηνύματα που στέλνονται και από τις δυο πλευρές.

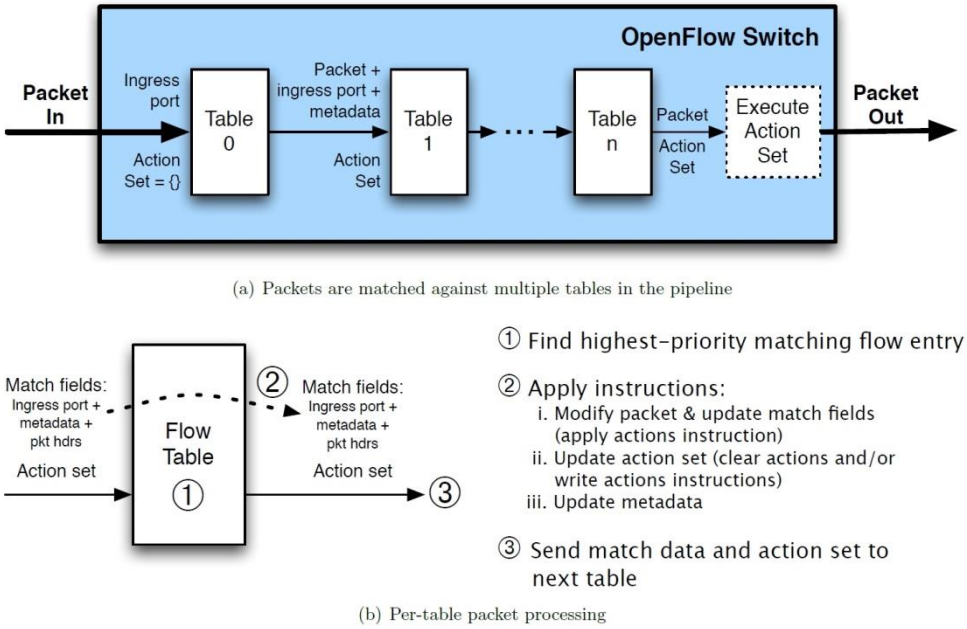
- **Hello:** Ανταλλάσσονται ανάμεσα στο switch και τον controller κατά την εκκίνηση της σύνδεσης τους.
- **Echo:** Μηνύματα echo request που στέλνονται είτε από το switch είτε από τον controller και περιμένουν ένα echo reply σε απάντηση. Χρησιμοποιούνται σαν δείκτης για latency/bandwidth, όπως επίσης και για επιβεβαίωση της ακεραιότητας της σύνδεσης.

- **Vendor:** Αποτελούν ένα τρόπο υλοποίησης περαιτέρω χρηστικότητας (αναλόγως τον vendor) εντός του OpenFlow πρωτοκόλλου. Σε επόμενες εκδόσεις του πρωτοκόλλου τα μηνύματα καλούνται "Experimenter".

2.3.5 *OpenFlow newer editions*

Παρότι για τις ανάγκες της παρούσας εργασίας έγινε χρήση της έκδοσης 1.0 του πρωτοκόλλου, οι σύγχρονες εκδόσεις διαθέτουν ενδιαφέροντα χαρακτηριστικά και δυνατότητες. Μια από τις ενδιαφέρουσες είναι η αλλαγή του τρόπου επεξεργασίας των πακέτων, λόγω της ύπαρξη περισσότερων του ενός flow table (σε ιεραρχική διάταξη).

- Τα Actions που υπήρχαν στις εγγραφές πλέον υφίστανται εντός ενός action set.
- Κατά την έναρξη της επεξεργασίας του πακέτου αυτό το set είναι κενό.
- Εκκίνηση της διαδικασίας από το πρώτο table.
- Εύρεση εγγραφής με την μέγιστη προτεραιότητα.
- Εφαρμογή των Instructions (που αντικατέστησαν τα actions), τα instructions έχουν σκοπό:
  - ✓ Εφαρμογή Actions χωρίς τροποποίηση του set
  - ✓ Τροποποίηση του Action Set
  - ✓ Τροποποίηση της επεξεργασίας του πακέτου εντός του pipeline των flow tables (Goto – table instruction, δυνατότητα μόνο για μεταγενέστερο table, δεν υπάρχει στο τελευταίο table του)
- Εάν το Instruction set δεν περιέχει Goto – table instruction η επεξεργασία στο pipeline σταματά και εκτελούνται τα Actions εντός του Set.



Σχήμα 2.6 Αντιστοίχιση και Επεξεργασία σε νεότερες εκδόσεις, Πηγή [4]

Η ύπαρξη πολλαπλών πινάκων δίνει περισσότερες δυνατότητες, όσον αφορά την διαδικασία του χειρισμού των πακέτων σε διαφορετικά βήματα, επιτρέποντας την εξοικονόμηση πόρων

στο switch κατά την διαδικασία του matching όσο και επιτρέποντας την προσθήκη πολλαπλών action από κάθε flow table.

Επίσης σημαντικές δυνατότητες στις επόμενες εκδόσεις είναι οι προσθήκες των Groups, Meters (για κάθε flow, υλοποιώντας την έννοια του περιορισμού της ροής), και υποστήριξη του πρωτοκόλλου IPv6.

2.3.5.1 Groups

Κάθε switch διαθέτει ένα πίνακα «Ομάδων» (group table), ο οποίος απαρτίζεται από εγγραφές διαφορετικών ομάδων (groups). Κάθε τέτοια εγγραφή απαρτίζεται από τα ακόλουθα πεδία (Πίνακας 2.6):

Group Identifier	Group Type	Counters	Action Buckets

Πίνακας 2.6 Group Entries, Πηγή [4]

- **Group Identifier:** Ένας μη προσημασμένος ακέραιος αριθμός μήκους 32 bits, ο οποίος προσδιορίζει το κάθε group με μοναδικό τρόπο.
- **Group Type:** Προσδιορίζει τον τύπο του group. Θα αναλυθούν αναλυτικότερα περεταίρω**.**
- **Counters:** Μετρητές, οι οποίοι ανανεώνονται κάθε φορά που ένα πακέτο επεξεργάζεται από το αντίστοιχο group.
- **Action Buckets:** Μια διατεταγμένη λίστα, από δοχεία με ενέργειες (action buckets). Το κάθε δοχείο περιέχει ένα σύνολο ενεργειών που εκτελούνται κατά την επεξεργασία, καθώς και τις σχετικές με τις ενέργειες αυτές παραμέτρους.

Στη συνέχεια ορίζονται οι διαφορετικοί τύποι Groups:

- **all:** Εκτέλεση όλων των Action Buckets που ορίζονται στο αντίστοιχο πεδίο. Αυτό το group χρησιμοποιείται για προώθηση πακέτων, multicast ή broadcast. Το πακέτο πρακτικά αντιγράφεται για κάθε bucket και τα αντίγραφα επεξεργάζονται από το αντίστοιχο bucket.
- **select:** Εκτέλεση ενός Action Bucket. Τα πακέτα κατευθύνονται σε ένα μοναδικό bucket του Group, βάση ενός αλγορίθμου επιλογής που υπολογίζεται από το ίδιο το switch. Ενδεικτικά παραδείγματα είναι η χρήση συνάρτησης κατακερματισμού σε στοιχεία του OpenFlow match που προσδιορίζεται από τον χρήστη, ή απλή εφαρμογή αλγόριθμου round robin. Σημειώνεται πως όλες οι ρυθμίσεις είναι εξωτερικές ως προς το πρωτόκολλο OpenFlow.
- **indirect:** Εκτέλεση του μοναδικού Action Bucket που είναι ορισμένο σε αυτό το Group. Το παραπάνω επιτρέπει σε πολλαπλές εγγραφές flow να χρησιμοποιούν ένα κοινό αναγνωριστικό (Group Identifier), για την εκτέλεση ενός Action Set (παράδειγμα επόμενο βήμα σε προώθηση IP). Αυτός ο τύπος είναι στην πράξη πανομοιότυπος με έναν **all**, στον οποίο είναι ορισμένο μόνο ένα Action bucket.

- **fast failover:** Εκτέλεση του πρώτου *ενεργού* Bucket. Κάθε action bucket είναι συνδεδεμένο με ένα συγκεκριμένο port ή/και ένα group που ελέγχει την κατάσταση του bucket. Αυτός ο τύπος groups, επιτρέπει στα switch να αλλάζει τον τρόπο προώθησης χωρίς να απαιτείται η επίκληση του controller

2.3.5.2 Meters

Στις νεότερες εκδόσεις του πρωτοκόλλου, υπάρχει επίσης και ένας πίνακας με «Ενδείκτες» - μετρητές ροής (Meter Table). Ένα τέτοιος πίνακας, απαρτίζει από αντίστοιχες εγγραφές (meter entries) οι οποίες ορίζουν μετρητές για κάθε flow, δίνοντας την δυνατότητα υλοποίησης απλών ή συνθετότερων, ενεργειών διασφάλισης ποιότητας (Quality of Service), όπως για παράδειγμα ο περιορισμός της ροής κίνησης (rate limiting). Κάθε τέτοιος δείκτης, μετρά το ρυθμό μετάδοσης πακέτων που έχουν ανατεθεί σε αυτόν, επιτρέποντας τον έλεγχο της ταχύτητας. Επισημαίνεται πως οι meters, είναι συνδεδεμένοι απευθείας με flow εγγραφές, σε αντίθεση με τις ουρές (Queues) που είναι συνδεδεμένες με πόρτες. Οποιαδήποτε flow εγγραφή μπορεί να ορίσει σύνδεση με κάποιο meter, μέσω του instruction set της. Επίσης, αξίζει να σημειωθεί πως ένας meter ελέγχει την συνολική ροή, από όλες τις flow εγγραφές που είναι συνδεδεμένες με αυτόν. Όσον αφορά την χρήση πολλαπλών μετρητών στο ίδιο flow table, κάτι τέτοιο είναι εφικτό αλλά όχι για διαφορετικούς meters στις ίδιες flow εγγραφές. Ωστόσο μπορούν να εφαρμοσθούν πολλαπλοί μετρητές, στις ίδιες ροές πακέτων σε διαφορετικά διαδοχικά flow tables.

Κάθε εγγραφή του πίνακα Μετρητών (Πίνακας 2.7) απαρτίζεται από:

Meter Identifier	Meter Bands	Counters

Πίνακας 2.7 Meter Entries, Πηγή [4]

- **Meter identifier:** Ένας μη προσημασμένος ακέραιος αριθμός μήκους 32 bits, ο οποίος προσδιορίζει μοναδικά την κάθε εγγραφή.
- **Meter bands:** Μία μη διατεταγμένη λίστα από μπάντες μετρητών (meter bands), όπου η καθεμία ορίζει το όριο της μπάντας και τον τρόπο επεξεργασίας των πακέτων που εμπίπτουν σε αυτές.
- **Counters:** Μετρητές (counters όχι meters), που ανανεώνονται όταν επεξεργάζονται πακέτα από τον αντίστοιχο μετρητή.

Meter Bands

Κάθε Μετρητής μπορεί να έχει μια ή περισσότερες μπάντες. Σημειώνεται πως τα πακέτα επεξεργάζονται από μια μόνο μπάντα, βάση της τρέχουσας τιμής της ροής. Ο μετρητής εφαρμόζει την μπάντα με την υψηλότερη τιμή ροής πακέτων η οποία τιμή όμως είναι χαμηλότερη από την τρέχουσα ροή. Σε περίπτωση που δεν υπάρχει μπάντα με αυτές τις προϋποθέσεις, δεν εφαρμόζεται καμία.

Κάθε μπάντα έχει ως αναγνωριστικό την ροή στην οποία εντάσσεται και απαρτίζεται από:

Band Type	Rate	Counters	Type specific arguments

Πίνακας 2.8 Meter Bands, Πηγή [4]

- Band type: Ορίζει τον τρόπο επεξεργασίας του πακέτου από αυτή τη μπάντα (drop, dscp remark).
- Rate**:** Χρησιμοποιείται από τον μετρητή για να επιλέξει την αντίστοιχη μπάντα, ορίζει το χαμηλότερο τρέχων ρυθμός μετάδοσης πάνω από τον οποίο μπορεί να εφαρμόζεται η αντίστοιχη μπάντα.
- Counters**:** Ανανεώνονται κατά την επεξεργασία ενός πακέτου από την μπάντα.
- Type specific arguments: Πρόσθετοι παράμετροι του band type.

2.4 OpenFlow Controller – POX

Η διαχείριση των δικτύων υπολογιστών, πραγματοποιείται μέσω παραμετροποίησης χαμηλού επιπέδου της κάθε συσκευής ξεχωριστά. Κατά την διάρκεια αυτής της διαδικασίας που είναι επίπονη και χρονοβόρα αναλογικά με το μέγεθος του δικτύου, ενδεχομένως να υπάρξουν σφάλματα ανθρώπινου παράγοντα. Ο διαχωρισμός του Control από το Data plane επιτρέπει τον έλεγχο των συσκευών μέσω του πρωτοκόλλου OpenFlow, παρέχοντας ένα αδιάσπαστο, κεντρικοποιημένο περιβάλλον προγραμματισμού και ελέγχου των δικτυακών συσκευών. Ο Controller είναι κατά τρόπον τινά ένα Λειτουργικό Σύστημα για το δίκτυο όπου η επικοινωνία μεταξύ των συσκευών και των εφαρμογών διέρχονται από εκείνον, χρησιμοποιώντας το υπερκείμενο (Northbound) Interface.

Υπάρχουν αρκετοί διαθέσιμοι Controller, σε διαφορετικές γλώσσες προγραμματισμού, ενδεικτικά κάποιοι:

- Open Daylight
- ONOS
- Flood light
- Ryu
- NOX
- POX

Ωστόσο θα επικεντρωθούμε στον Controller POX ([5]), έναν από τους πλέον διαδεδομένου, ο οποίος είναι βασισμένος στην γλώσσα Python ([6]).

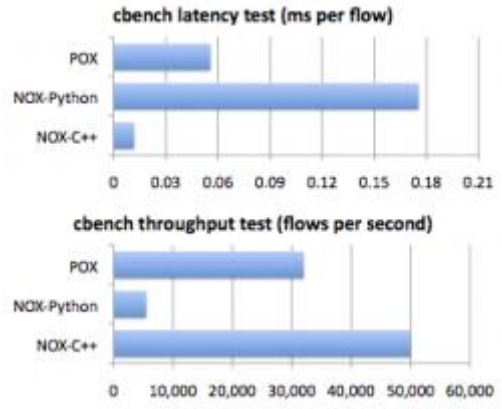
POX

Ο POX παρέχει μια πλατφόρμα ανάπτυξης και προτυποποίησης λογισμικού ελέγχου δικτύου, χρησιμοποιώντας την γλώσσα Python. Στην πραγματικότητα ο POX είναι η μεταφορά του NOX (ενός άλλου ιδιαίτερα διαδεδομένου Controller) σε Python. Ο POX λειτουργεί ως το γενικότερο υπόβαθρο επί του οποίου πραγματοποιείται, όλη η επικοινωνία με τα OpenFlow switches και αποτελεί ένα από τα πλέον δημοφιλή εργαλεία στον τομέα

του SDN. Βρίσκεται υπό διαρκή ενημέρωση και αποτελεί ιδανική λύση για εξοικείωση με το περιβάλλον των Software Defined Networks.

Κάποια από τα χαρακτηριστικά του POX:

- "Pythonic" OpenFlow interface.
- Επαναχρησιμοποίηση των διάφορων στοιχείων (επιλογή μονοπατιού, ανακάλυψη της τοπολογίας).
- "Runs anywhere" – Εκτέλεση σε οποιοδήποτε σύστημα, συγκεκριμένα στοχεύει σε Linux, Mac OS and Windows.
- Υποστηρίζει το ίδιο γραφικό περιβάλλον χρήστη και οπτικά εργαλεία με τον συγγενικό controller NOX.
- Έχει ικανοποιητική απόδοση σε σύγκριση με τις εφαρμογές του NOX γραμμένες σε Python (Σχήμα 2.7 Σύγκριση απόδοσης διαφορετικών Controller).



Σχήμα 2.7 Σύγκριση απόδοσης διαφορετικών Controller, Πηγή [5]

Η εκτέλεση του POX γίνεται μέσω του αρχείου pox.py . Εκτελώντας όμως αποκλειστικά το αρχείο pox.py δεν έχει κάποιο ιδιαίτερο αποτέλεσμα. Η κύρια λειτουργικότητα του POX παρέχεται από *εξαρτήματα* (components). Ο POX διαθέτει κάποια components (ωστόσο κυρίως απευθύνεται σε κοινό που έχει ως στόχο να αναπτύξει τα δικά του components).

2.4.1 *Components*

Όταν αναφερόμαστε σε components, εννοούμε αρχεία που μπορούμε να προσθέσουμε στην εντολή με την οποία επικαλούμαστε τον POX. Όπως αναφέρθηκε ο POX διαθέτει εκ κατασκευή components, κάποια παρέχουν τη βασική λειτουργικότητα, κάποια ορισμένα βολικά χαρακτηριστικά και κάποια είναι απλώς παραδείγματα. Στην συνέχεια παρατίθενται ενδεικτικά κάποια από τα διαθέσιμα components ([7],[8]).

Py

Εκκίνηση ενός διαδραστικού Python interpreter. Χρήση για αποσφαλμάτωση και πειραματισμό.

Forwarding.hub

Απλή εγκατάσταση FLOOD κανόνων σε κάθε switch, μετασχηματίζοντας τα σε hubs.

Forwarding.l2_learning

Παρόμοια λειτουργία με ένα L2 learning switch. Παρόλο που το εξάρτημα εστιάζει στις διάφορες διευθύνσεις του στρώματος 2, οι κανόνες που εγκαθίστανται είναι όσο το δυνατόν πιο συγκεκριμένοι (τα λιγότερα δυνατά wildcards).

Forwarding.l2_pairs

Όμοια λειτουργία με το l2_learning ωστόσο οι κανόνες βασίζονται κυρίως στις διευθύνσεις MAC.

forwarding.l3_learning

Το συγκεκριμένο εξάρτημα δεν αποτελεί ακριβώς ένα router, αλλά επίσης δεν είναι ένα L2 switch. Μαθαίνει και καταχωρεί διευθύνσεις IP, χωρίς να δίνει ιδιαίτερη σημασία σε έννοιες όπως τα διαφορετικά υποδίκτυα. Προκειμένου να εναρμονιστεί με τον τρόπο λειτουργίας των host, είναι δυνατό κατά την φόρτωση να ορίσουμε fake gateways.

forwarding.l2_multi

Συμπεριφέρεται σαν ένα learning switch. Η σημαντική του διαφορά με τα αντίστοιχα εξαρτήματα είναι πως χρησιμοποιεί το **openflow.discovery** για να ενημερωθεί για όλη την τοπολογία του δικτύου. Έτσι μόλις ένα switch καταχωρήσει μια διεύθυνση MAC όλα μοιράζονται αυτή την καταχώρηση. Σημειώνεται πως απαιτείται επίσης, η χρήση του εξαρτήματος openflow.discovery.

forwarding.topo_proactive

Εγκατάσταση κανόνων προκαταβολικά, με βάση μια διεύθυνση IP που έχει σημασία για την τοπολογία. Η μεταγωγή βασίζεται στο forwarding.l2_multi που αναφέρθηκε προηγουμένως.

openflow.spanning_tree

Χρησιμοποιεί το openflow.discovery για την δημιουργία μιας επισκόπησης της τοπολογίας του δικτύου, δημιουργεί ένα spanning tree και στη συνέχεια απενεργοποιεί το flooding για τις πόρτες που δεν είναι στο spanning tree.

openflow.webservice

Παρέχει ένα web service τύπου JSON-RPC (θα αναλυθεί εκτενώς στο 44) για την αλληλεπίδραση με το OpenFlow πρωτόκολλο. Προέρχεται από την υπηρεσία μηνυμάτων of_service και προϋποθέτει την χρήση του webcore εξαρτήματος. Η υπηρεσία είναι προσβάσιμη με αποστολή μηνυμάτων HTTP POST στο http://webcore_running/OF.

web.webcore
Εκκίνηση ενός web server εντός της διεργασίας του POX.

Messenger
Προσφέρει μια διεπαφή για τον POX ώστε να επικοινωνεί με εξωτερικές διαδικασίες μέσω αμφίδρομων μηνυμάτων βασισμένα σε JSON.

openflow.of_01
Επικοινωνία με OpenFlow 1.0 switches. Στην περίπτωση φόρτωσης διαφορετικών components που χρησιμοποιούν OpenFlow φορτώνεται και αυτό με τις προεπιλεγμένες τιμές. Υπάρχει δυνατότητα αλλαγής των τιμών στην χειροκίνητη εκτέλεση (port, address, private-key, certificate, certificate-authority)

openflow.discovery
Αποστολή ειδικά διαμορφωμένων LLDP μηνυμάτων ώστε ο controller να ανακαλύψει την τοπολογία του δικτύου

openflow.debug
Δημιουργία αρχείων pcap με τα OpenFlow μηνύματα που ανταλλάζονται.

misc.full_payload
Η προεπιλεγμένη συμπεριφορά όταν ένα πακέτο δεν μπορεί να αντιστοιχισθεί σε κάποια εγγραφή του flow table είναι η αποστολή ενός μέρους του πακέτου (τα πρώτα 128 bytes) στον controller. Αυτό το εξάρτημα τροποποιεί κατάλληλα τα switches που συνδέονται στον controller ώστε να στέλνουν ολόκληρο το πακέτο.

Log

Ο POX χρησιμοποιεί το σύστημα καταγραφής συμβάντων της Python. Το εξάρτημα log επιτρέπει την παραμετροποίηση του σε σημαντικό βαθμό μέσω της γραμμής εντολών. Ενδεικτικά:

- Απενεργοποίηση της καταγραφής
- Μορφοποίηση της καταγραφής, προσθήκη και μορφοποίηση timestamps
- Εκτύπωση σε stderr, stdout, αρχεία, αποστολή μέσω TCP Sockets, UDP Datagrams και HTTP GET/POST

log.level
Η καταγραφή ακολουθεί την ίδια δομή που ακολουθεί το σύστημα της Python. Διαφορετικά εξαρτήματα έχουν τους δικούς τους loggers το όνομα του καθενός φαίνεται στα μηνύματα καταγραφής. Στη πραγματικότητα οι καταγραφείς σχηματίζουν την δική τους

ιεραρχία και επιπρόσθετα κάθε ένας έχει ένα επίπεδο ("level"), το οποίο αντιστοιχεί πόσο σημαντικό ή όχι είναι ένα μήνυμα. Το εξάρτημα log.level επιτρέπει την παραμετροποίηση του επιπέδου των πληροφοριών που δείχνει το κάθε σύστημα καταγραφής.

Το πόσο σημαντικό είναι κάθε μήνυμα περιγράφεται από τις ακόλουθες κατηγορίες (από το πλέον στο λιγότερο σημαντικό):

- CRITICAL
- ERROR
- WARNING
- INFO
- DEBUG

Το προεπιλεγμένο επίπεδο καταγραφής για τον POX είναι το INFO.

2.4.2 *POX APIs*

Εκτός από τα εξαρτήματα στα οποία έγινε αναφορά, ο POX διαθέτει ένα σύνολο APIs προκειμένου να διευκολύνει την ανάπτυξη εφαρμογών δικτυακού ελέγχου. Στη συνέχεια παρουσιάζονται ενδεικτικά κάποια εξ αυτών:

2.4.2.1 POX Core

Υπάρχει το αντικείμενο "core", το οποίο λειτουργεί σαν κεντρικός κόμβος για μεγάλο μέρος του API του POX. Κάποιες από τις λειτουργίες που προσφέρει είναι απλώς wrappers (ρουτίνες και συναρτήσεις που κύριος στόχος τους είναι η κλήση δευτερευουσών ρουτινών και συναρτήσεων) ενώ άλλες είναι αυτόνομες. Ωστόσο, από τους βασικότερους λόγους ύπαρξης του αντικειμένου "core" είναι να χρησιμοποιείται σαν σημείο συνάντησης μεταξύ των εξαρτημάτων. Έτσι αντί να χρησιμοποιούνται οι εντολές "import" ώστε ένα εξάρτημα να εισάγει ένα άλλο προκειμένου να αλληλεπιδράσει μαζί του, τα εξαρτήματα εγγράφονται πλέον ("register") στο "core" και επικοινωνούν μεταξύ τους μέσω αυτού.

2.4.2.2 Σύστημα διαχείρισης γεγονότων (pox.lib.revent)

Ο χειρισμός των γεγονότων στον POX ακολουθεί το μοτίβο "publish – subscribe", στο οποίο οι αποστολείς μηνυμάτων (γνωστοί ως publishers) δεν προγραμματίζουν τα μηνύματα να στέλνονται απευθείας σε συγκεκριμένους παραλήπτες (γνωστοί ως subscribers). Κάποια αντικείμενα κάνουν publish events (χρησιμοποιείται συνήθως ο όρος "raise"), σε αυτά τα αντικείμενα συνέχεια κάποιο άλλο μπορεί να κάνει subscribe και να αναμένει για συγκεκριμένα events (αντίστοιχα χρησιμοποιείται ο όρος "listening to" / "handling"). Το ζητούμενο είναι όταν συμβαίνει ένα event (γίνεται raise) να ενεργοποιηθεί ένα συγκεκριμένο τμήμα κώδικα για τον χειρισμό του (γνωστό και ως "**event handler"**). Σημαντικό μέρος

του τρόπου λειτουργίας του POX βασίζεται στη δημοσίευση και στον χειρισμό των events. Κάτι τέτοιο διευκολύνει την ανάπτυξη εφαρμογών απλώς δημιουργώντας απλως, εξαρτήματα που περιέχουν handlers για τα διάφορα events (Packet In, Connection Up, Port Status).

2.4.2.3 Πακέτα (pox.lib.packet)

Πολλές εφαρμογές στον POX βασίζονται στην αλληλεπίδραση με πακέτα (για παράδειγμα κατασκευή πακέτων και αποστολή μέσω του switch ή προσπέλαση ενός πακέτου που ελήφθη από ένα packet in event). Προκείμενου να διευκολύνει την παραπάνω διαδικασία ο POX διαθέτει μια βιβλιοθήκη για την ανάλυση και κατασκευή πακέτων. Κάποιοι από τους τύπους των πακέτων που υποστηρίζει η βιβλιοθήκη του POX είναι Ethernet, ARP, IPv4, ICMP, TCP, UDP, DHCP, DNS. Κάθε τύπος υποστηρίζει συγκεκριμένα attributes.

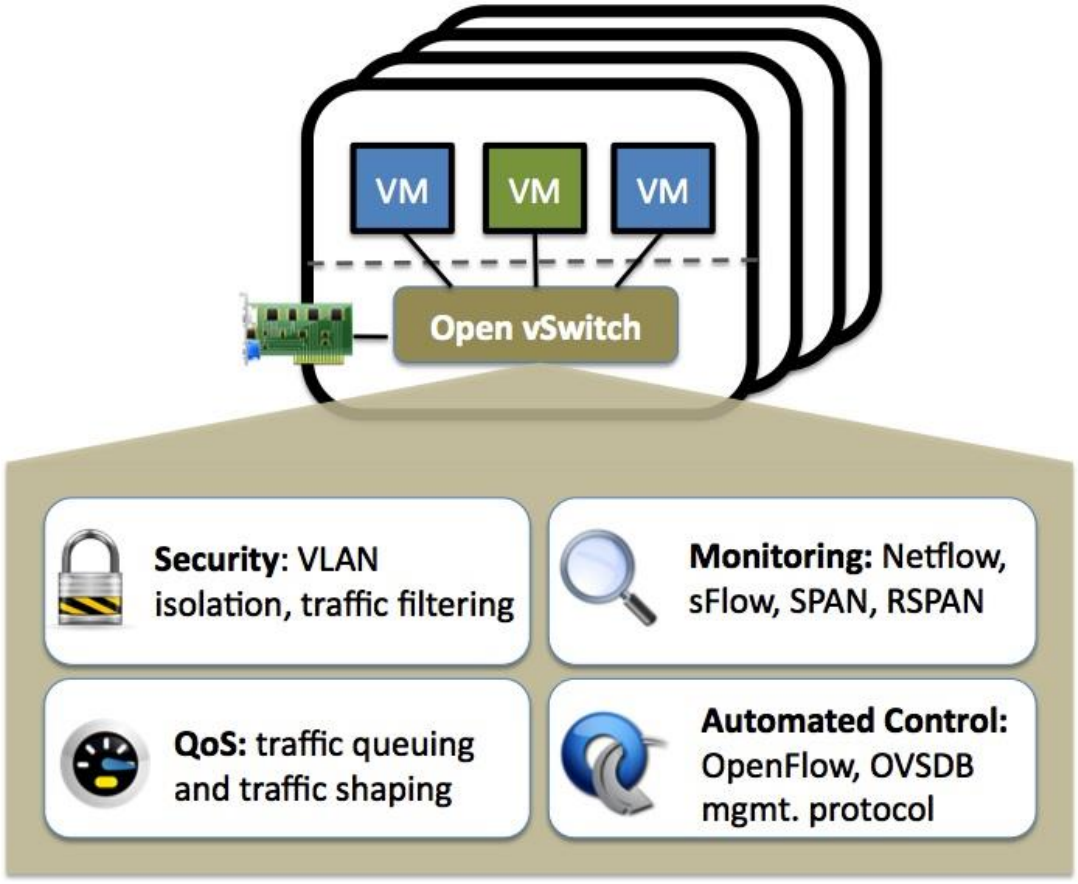
2.4.2.4 Νήματα, διεργασίες και χρονόμετρα (pox.lib.recoco)

Η βιβλιοθήκη recoco του POX χρησιμοποιείται για την υλοποίηση απλών συνεργατικών διεργασιών. Ίσως το βασικότερο πλεονέκτημα των διεργασιών αυτών είναι πως συνήθως δεν χρειάζεται να γίνει κάποια ειδική ενέργεια για τον συγχρονισμό τους. Όσον αφορά την δημιουργία διαφορετικών νημάτων μπορεί να χρησιμοποιηθεί ξανά η βιβλιοθήκη recoco ή συνηθισμένη βιβλιοθήκη "threading" της Python.

Ενδιαφέρον παρουσιάζει και η εκτέλεση διεργασιών μετά από κάποιο χρονικό μέσω χρονομέτρων. Ενδεικτικά παραδείγματα χρήσης η αποσύνδεση του controller από το switch αν δεν έχει ληφθεί για κάποιο διάστημα απάντηση σε μηνύματα echo, ή επίσης το αίτημα στο switch ανά περιόδους ώστε να στέλνει στατιστικά δεδομένα για τους κανόνες που περιέχει.

2.5 Open vSwitch

Το Open vSwitch είναι ένα εικονικό (virtual) switch πολλαπλών στρωμάτων, βιομηχανικής ποιότητας. Είναι σχεδιασμένο για να διευκολύνει στο μέγιστο τον αυτοματισμό στις λειτουργίες δικτύου μέσω προγραμματικών επεκτάσεων, ενώ ταυτόχρονα υποστηρίζει τα συνήθη διαχειριστικά interfaces και πρωτόκολλα (NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag). Επίσης, υποστηρίζει τον διαμοιρασμό (distribution) σε διαφορετικά φυσικά μηχανήματα. Ιδιαίτερο ενδιαφέρον παρουσιάζει η δυνατότητα δημιουργίας λογικών switches (OpenFlow Enabled) τα οποία μπορούν να ενταχθούν σε εικονικές τοπολογίες, γεγονός που προσφέρει σημαντικές ερευνητικές και εμπορικές δυνατότητες.



Σχήμα 2.8 Open vSwitch, Πηγή [9]

Το Open vSwitch χρησιμοποιείται σε μια πληθώρα προϊόντων και εκτελείται σε πολλά ιδιαίτερα μεγάλα βιομηχανικά περιβάλλοντα. Ενδεικτικά αναφέρεται πως είναι το προεπιλεγμένο switch στον XenServer6.0, στο Xen Cloud Platform και επίσης υποστηρίζει Xen, KVM, Proxmox VE και Virtual Box, επιπρόσθετα έχει ενσωματωθεί σε πολλά συστήματα διαχείρισης εικονικών πόρων (OpenStack, openQRM, OpenNebula, oVirt).

2.6 Network Virtualization

Οι θεμελιώδεις αρχές της εικονικοποίησης ήδη έχουν υλοποιηθεί σε αρκετά πρωτόκολλα δικτύων που χρησιμοποιούνται κατά κόρον. (Παράδειγμα VLANs – VPNs). Η ραγδαία ανάπτυξη των υπολογιστικών νεφών (Cloud Computing) ανέδειξε περαιτέρω την ανάγκη για εικονικοποίηση δικτυακών πόρων (Network Virtualization) καθώς χρειαζόταν ένας τρόπος ώστε πολλαπλοί χρήστες (ένοικοι) να μοιράζονται την ίδια δικτυακή υποδομή. Η αρχιτεκτονική του SDN, αποτελεί ένα στέρεο υπόβαθρο για την ανάπτυξη και διαχείριση εικονικού περιβάλλοντος. Σημειώνεται πως, με τον όρο Network Virtualization δεν συνεπάγεται απαραίτητα χρήση του SDN ούτε το αντίθετο. Αποτελούν διαφορετικές έννοιες, ωστόσο υπάρχει μια σχέση συμβίωσης ανάμεσα στα δυο:

- Το SDN αποτελεί μια τεχνολογία που διευκολύνει την ύπαρξη του Network Virtualization.

- Μπορεί να εκμεταλλευθεί το Network Virtualization στην διαδικασία της εκτίμησης – δοκιμής ενός SDN περιβάλλοντος. Η ικανότητα της αποσύνδεσης μια εφαρμογής ελέγχου από το υποκείμενο data plane, δίνοντας την δυνατότητα να εκτιμήσουμε και δοκιμάσουμε την εφαρμογή, σε οποιοδήποτε εικονικό περιβάλλον.

Υπάρχουν δυο βασικές προσεγγίσεις για το Network Virtualization:

a) Full Network Virtualization
b) Control plane "slicing"

2.6.1 *Full Network Virtualization*

Όπως αναφέρεται και στο [10], η εικονικοποίηση του δικτύου συνίσταται στην παρουσίαση ενός αφαιρετικού επιπέδου που είναι αποσυνδεδεμένο από την φυσική τοπολογία. Κάτι τέτοιο επιτρέπει σε πολλαπλά εικονικά δίκτυα να συνυπάρχουν πάνω σε μια κοινή υποδομή, όπου κάθε εικονικό δίκτυο μπορεί να έχει διαφορετική τοπολογία, χωρίς σχέση με το υπόβαθρο στο οποίο στηρίζονται.

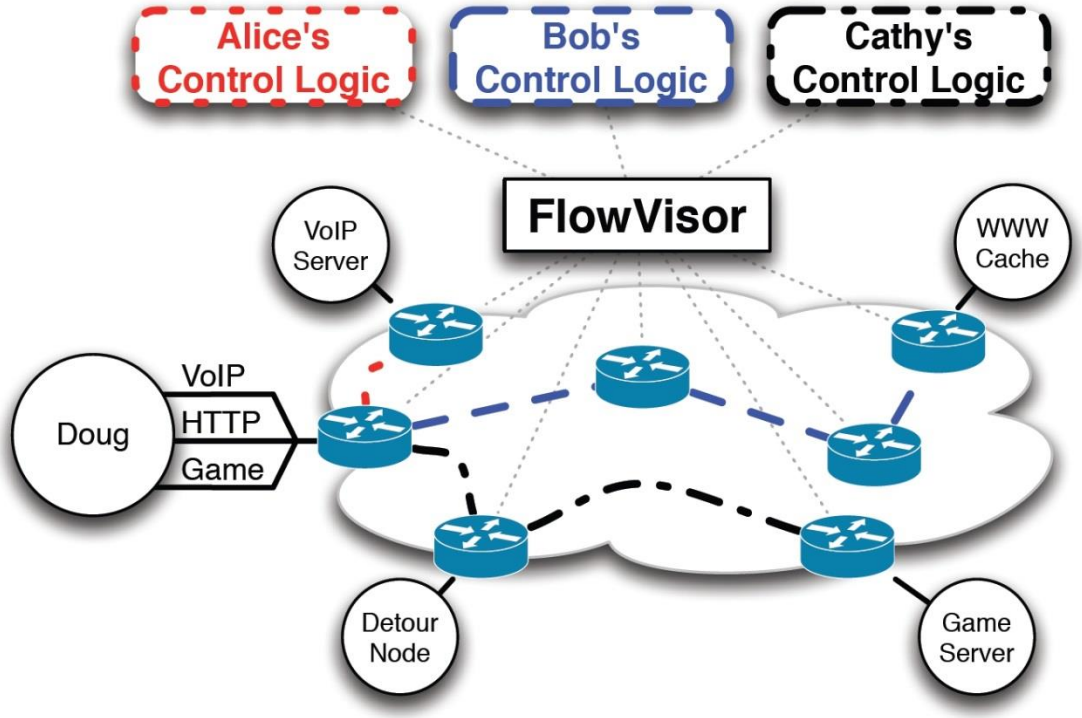
2.6.2 *Control Plane Slicing*

Η κύρια ιδέα είναι ο διαχωρισμός των δικτύων σε «τεμάχια» ("Slices" – μια ιδέα που αρχικά παρουσιάστηκε στο PlanetLab [11]), όπου κάθε Slice περιλαμβάνει συγκεκριμένους δικτυακούς πόρους και είναι υπό την εποπτεία διαφορετικού SDN Controller. Το παραπάνω είναι εφικτό χρησιμοποιώντας έναν ενδιάμεσο Controller ο οποίος λειτουργεί σαν διαφανής πληρεξούσιος ("proxy"), ο οποίος παρεμβάλλεται ανάμεσα στον Controller του κάθε Slice και τα OpenFlow switches. Λόγω της διαφάνειας ("transparency") του ενδιάμεσου οι Controllers έχουν την αίσθηση πως επικοινωνούν απευθείας με τα switch, και αντίστοιχα τα switch με τους Controllers.

2.7 FlowVisor

Ο FlowVisor [12], [13] είναι ένας πειραματικός SDN controller που υλοποιεί την εικονικοποίηση του δικτύου χωρίζοντας το φυσικό δίκτυο σε πολλαπλά λογικά (αυτό επιτυγχάνεται πρακτικά τεμαχίζοντας το Control Plane). Ο FlowVisor απομονώνει αυτά τα δίκτυα ώστε κάθε controller να λαμβάνει μηνύματα και να έχει πρόσβαση μόνο σε switches και πόρους που του αναλογούν. Η απομόνωση που προσφέρει ο FlowVisor είναι από τεχνικής πλευράς εφικτή όχι απλά προωθώντας ή φιλτράροντας την επικοινωνία controller – switch αλλά τροποποιώντας και το περιεχόμενο τους (π.χ. στα μηνύματα OpenFlow negotiation για την ανακάλυψη των φυσικών ports του switch). Σημειώνεται πως ο διαμοιρασμός των πόρων σε τεμάχια μπορεί να γίνει σε επίπεδο εύρους ζώνης, τοπολογίας, CPU και flow table των συσκευών και φυσικά κίνησης.

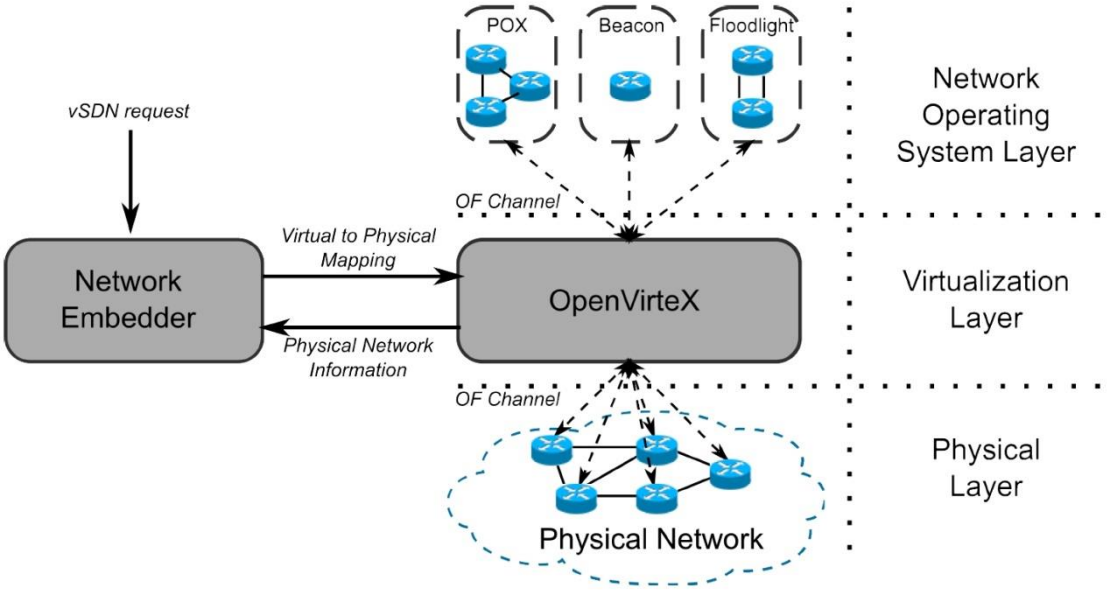
Παρόλο που γενικά θεωρείται μια πειραματική τεχνολογία το πανεπιστήμιο Stanford έχει ενσωματώσει τον FlowVisor στο πανεπιστημιακό δίκτυο του. Επίσης, ο FlowVisor επίσης αποτελεί λύση σε ένα από τα προβλήματα της έρευνας πάνω σε δίκτυα υπολογιστών, αυτό της ρεαλιστικής τεκμηρίωσης. Καθώς αφενός για να φτιαχτεί ένα περιβάλλον για δοκιμές (testbed) αποκλειστικά για τις ανάγκες του πειραματισμού έχει πολύ μεγάλο κόστος, αφετέρου πρέπει το περιβάλλον των δοκιμών να έχει τις ίδιες ιδιομορφίες (κίνηση, χρήστες κλπ) με το περιβάλλον το οποίο στοχεύουμε να εκτελέσουμε την έρευνα μας. Επιπροσθέτως, πρόκειται για κάτι πειραματικό που δοκιμάζεται σε πραγματικές συνθήκες, συνεπώς δεν θα ήταν ασφαλές να χρησιμοποιηθεί εξ ολοκλήρου ένα πραγματικό δίκτυο παραγωγής. Ο FlowVisor επιτρέπει τον τεμαχισμό του δικτύου παραγωγής ώστε να δίνει τη δυνατότητα στο περιβάλλον δοκιμών να συνυπάρξει πάνω στην δομή ενός πραγματικού δικτύου και να είναι μεταβλητό ανάλογα με τις ανάγκες της έρευνας, χωρίς να επηρεάζει τον κύριο όγκο της εμπορικής του κίνησης.



Σχήμα 2.9 Network Virtualization με χρήση FlowVisor, Πηγή [12],[13]

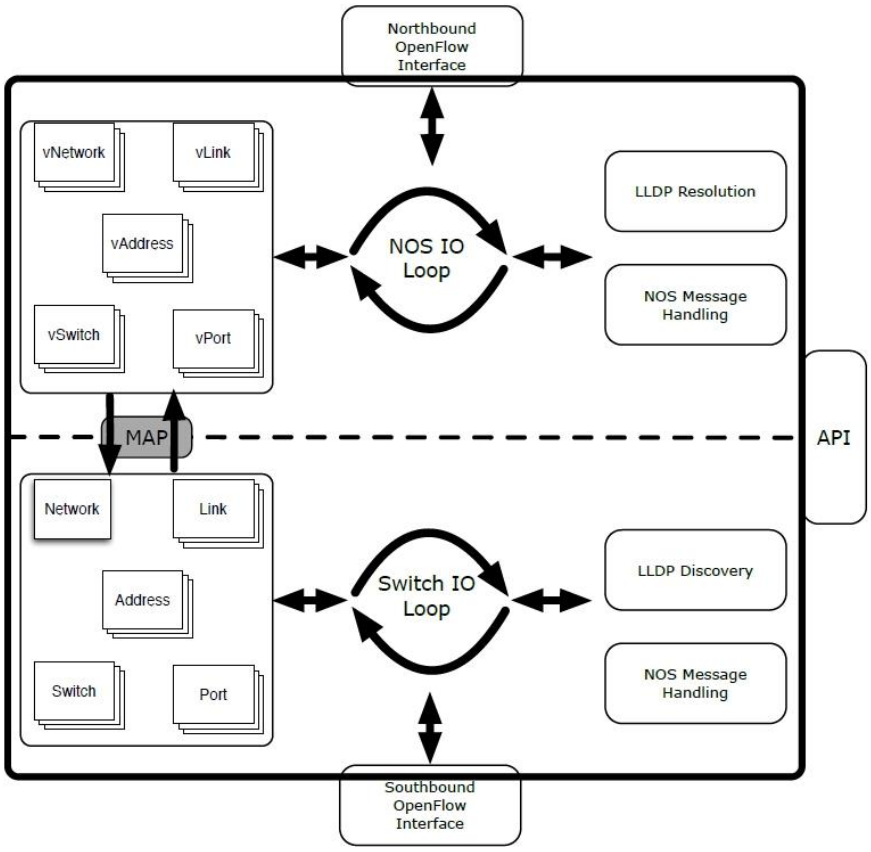
2.8 OpenVirtex

Το OpenVirtex [14] αποτελεί μια πλατφόρμα network virtualization που δίνει στους χρήστες την δυνατότητα να δημιουργούν και να διαχειρίζονται εικονικά Δίκτυα Οριζόμενα από Λογισμικό (vSDNs). Οι χρήστες – ένοικοι έχουν το ελεύθερο να ορίσουν την τοπολογία, τον τρόπο διευθυνσιοδότησης του εικονικού δικτύου καθώς και να διαθέτουν το δικό τους Λειτουργικό Σύστημα Δικτύων. Όπως φαίνεται και στο Σχήμα 2.10 OpenVirteX architecture, όπου περιγράφεται η αρχιτεκτονική, το εργαλείο που κάνει εφικτή την διαδικασία λέγεται Network Embedder. Το εργαλείο αυτό δέχεται αιτήματα για την δημιουργία νέων εικονικών δικτύων (προσδιορίζοντας πλήρως τοπολογία, ζεύξεις κλπ) και στην συνέχεια συγκεντρώνοντας πληροφορίες από την φυσική τοπολογία δημιουργεί μια αντιστοίχιση μεταξύ πραγματικής και εικονικής τοπολογίας.



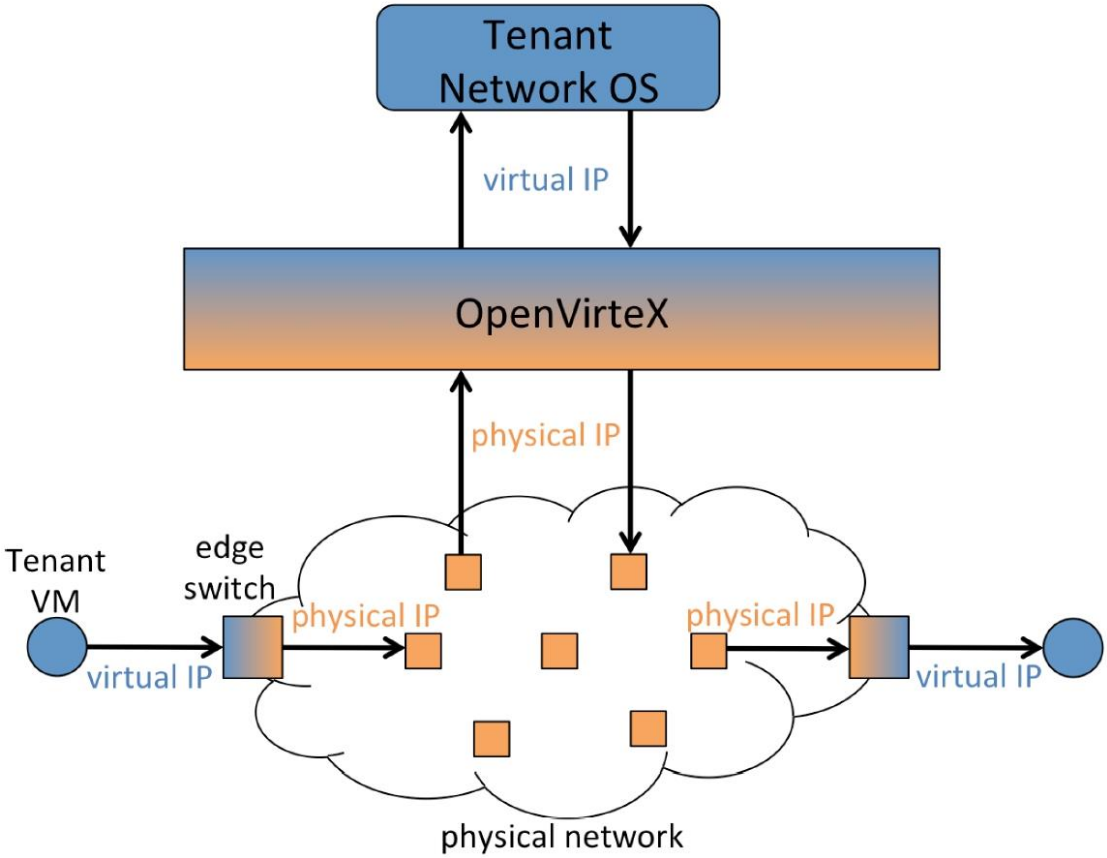
Σχήμα 2.10 OpenVirteX architecture, Πηγή [14]

Εσωτερικά, το OpenVirteX βασίζεται στην αποσύνδεση των εικονικών στοιχείων από τα αντίστοιχα πραγματικά (Σχήμα 2.11). Προκειμένου να επιτευχθεί αυτό το μοντελοποιεί όλα τα πραγματικά και εικονικά στοιχεία και διατηρεί μια αντιστοίχιση ανάμεσα τους που παρέχεται από τον Network Embedder όπως αναφέρθηκε. Η αντιστοίχιση δεν ορίζει τον τρόπο υλοποίησης της εικονικοποίησης.



Σχήμα 2.11 OpenVirteX Internal Architecture, Πηγή [14]

Συνεπώς η πλατφόρμα του OpenVirteX προσφέρει εικονικές τοπολογίες πλήρως παραμετροποιήσιμες, καθώς και σχήμα εικονικής διευθυνσιοδότησης ανεξάρτητο σε κάθε εικονικό δίκτυο. Όσον αφορά τον τρόπο με τον οποίο επιτυγχάνεται η εικονικοποί ηση πραγματοποιείται επανεγγραφή των διεύθυνσεων IP όταν το Λειτουργικό Σύστημα του Δικτύου αναφέρεται σε κανόνες που έχουν κεφαλίδες επιπέδου 3 και επανεγγραφή των διευθύνσεων MAC όταν έχουν κεφαλίδες επιπέδου 2 αντίστοιχα. Στο Σχήμα 2.12 παρατίθεται η διαδικασία της μετάφρασης.



Σχήμα 2.12 Mapping Process, Πηγή [14]

Συνολικά, εκτός από τα προαναφερθέντα τα vSDN προσφέρουν:

- **Ανθεκτικότητα:** Μια εικονική ζεύξη μπορεί να αντιστοιχισθεί σε πολλαπλές φυσικές με στόχο να λειτουργούν ως εφεδρικές σε περίπτωση που κάποια ζεύξη καταρρεύσει.
- **Δυναμική Παραμετροποίηση:** Η ρύθμιση ενός δικτύου συνίσταται στον χειρισμό ζευγών key – value. Αυτή η διαδικασία είναι απλή, καθώς η αντιστοίχιση είναι κεντρικοποιημένη και έτσι είναι εφικτή η παραμετροποίηση ενός δικτύου που είναι ήδη ενεργό.
- **Δυνατότητα Αποθήκευσης:** Κάθε δίκτυο διατηρεί μια συλλογή από πληροφορίες για κάθε στοιχείο. Αυτό επιτρέπει στο OpenVirteX να διατηρήσει αυτές τις πληροφορίες σε έναν μόνιμο χώρο αποθήκευσης. Είναι λοιπόν δυνατό να αποθηκεύονται και να φορτώνονται οι ρυθμίσεις για το κάθε vSDN, κάτι που αποτελεί το πρώτο βήμα για την αποθήκευση Στιγμιότυπων (Snapshots).

Σημειώνεται πως ο FlowVisor λειτουργεί με διαφορετικό τρόπο, διαιρώντας το υποκείμενο δίκτυο σε υποσύνολα και όχι εξομοιώνοντας εικονικά δίκτυα. Με βάση αυτό τον τρόπο λειτουργίας εντοπίζονται οι ακόλουθοι περιορισμοί:

- Όλα τα τμήματα μοιράζονται ένα κοινό σύνολο διευθύνσεων, και έτσι δεν είναι δυνατή η ύπαρξη ανεξάρτητων και αυτόνομων διευθύνσεων.
- Σε συνέχεια του παραπάνω, δεν επιτρέπεται σε ένα κομμάτι να έχει μια αυθαίρετη (εικονική) τοπολογία. Μπορεί μόνο απαρτίζεται από ένα υποσύνολο της φυσικής τοπολογίας.
- Η απομόνωση ανάμεσα στο κάθε κομμάτι εξασφαλίζεται με την εφαρμογή κανόνων για το flowspace που αντιστοιχεί στο καθένα. Οι κανόνες δεν πρέπει σε καμία περίπτωση να παρουσιάζουν επικάλυψη, δυστυχώς λάθος παραμετροποίηση τους μπορεί να οδηγήσει στην παραβίαση αυτής της αρχής. Σε αυτή την περίπτωση δεν μπορεί να εφαρμοσθεί εγγυημένα η απομόνωση των κομματιών που επηρεάζονται.

2.9 Network Monitoring

Με τον όρο Network Monitoring συνήθως αναφερόμαστε σε μια σειρά ενεργειών που εντάσσονται στα Management Plane. Ενέργειες που πραγματοποιούνται από διαχειριστές δικτύων υπολογιστών με στόχο την παρακολούθηση και ανάλυση της δικτυακής κίνησης στο δίκτυο υπό την εποπτεία τους. Τα πλεονεκτήματα της παρακολούθηση του δικτύου και αποθήκευση των δεδομένων αυτών ("Network Monitoring Data") είναι πολύπλευρα:

- Ασφάλεια
- Ποιότητα υπηρεσιών
- Εντοπισμός προβλημάτων
- Πρόβλεψη μελλοντικών αναγκών
- Business Support Systems/Operations Support Systems
- Ευκολίες στην αποσφαλμάτωση (debugging)

Οι τρόποι παρακολούθησης ποικίλουν. Παραδείγματος χάριν μπορούμε να παρακολουθούμε τις δικτυακές συσκευές ή τους hosts που συνδέονται σε αυτές (ιδιαίτερα σύνηθες εάν δεν ενδιαφερόμαστε αποκλειστικά για την δικτυακή κίνηση αλλά για υπολογιστικές πληροφορίες, μνήμη, επεξεργαστής, αποθηκευτικός χώρος κλπ). Στα πλαίσια της παρούσας εργασίας θα επικεντρωθούμε στον τομέα της δικτυακής κίνησης. Η παρακολούθηση έχει διαφορετικές μορφές, με τις πλέον διαδεδομένες να είναι (1) το καθρέφτισμα (mirror) και (2) η δειγματοληψία (sampling) της κίνησης.

Στην περίπτωση του mirroring, αυτούσια η κίνηση που διέρχεται από το υπό παρακολούθηση στοιχείο αντιγράφεται («καθρεφτίζεται») και έχει μια συγκεκριμένη μεταχείριση. Όσον αφορά την δειγματοληψία, η υλοποίηση της επιλέγει πακέτα με κάποια συγκεκριμένη συχνότητα και επίσης τα μεταχειρίζεται κατάλληλα. Συνήθως αυτή η

μεταχείριση συνίσταται στην αποστολή της κίνησης (mirrored / sampled) σε ένα μηχάνημα που είναι αφιερωμένο στην συλλογή και στη συνέχεια, επεξεργασία των στοιχείων αυτών.

Υπάρχουν ποικίλοι τρόποι για να πετύχουμε την εξαγωγή και αποστολή των δεδομένων αυτών σε μηχανήματα για περαιτέρω επεξεργασία, καθένας βασισμένος σε διαφορετικά πρότυπα και πρωτόκολλα (π.χ. NetFlow, JFlow, sFlow). Ορισμένα μάλιστα είναι vendor specific, και παρέχουν ολοκληρωμένο περιβάλλον τόσο εξαγωγής όσο και ανάλυσης δεδομένων. Τα μηχανήματα που δέχονται τα δεδομένα συνήθως αποκαλούνται συλλέκτες (collectors) και υπάρχει πληθώρα διαφορετικών εφαρμογών που επιτελούν την διαδικασία της συλλογής και ανάλυσης (π.χ. ntop, Ganglia).

Όσον αφορά την σύγκριση των μεθόδων παρακολούθησης, μπορεί το mirror της κίνησης να αποδίδει με τον πιο πιστό τρόπο την κατάσταση, αλλά στα σύγχρονα περιβάλλοντα τα οποία χαρακτηρίζονται από κίνηση μεγάλου όγκου, είναι μια διαδικασία που απαιτεί πολλούς πόρους τόσο στα μηχανήματα που εκτελούν το mirroring (switches, hosts) όσο και στους collectors – analyzers. Ειδικότερα, όσον αφορά τα switches επιβαρύνονται βασικά στοιχεία όπως η CPU κάτι που μπορεί να μειώσει την μεταγωγική ικανότητα του ίδιου του switch. Επίσης καθώς η κίνηση από διάφορα ports αθροίζεται η ζεύξη στην οποία κατευθύνεται το mirror ενδεχομένως να υπερφορτωθεί προκαλώντας packet loss, delays, buffer overflow. Ωστόσο, ακόμα μεγαλύτερο πρόβλημα εντοπίζεται στα μηχανήματα που επεξεργάζονται την ροή της πληροφορίας καθώς αυτά συνήθως υλοποιούν πολυπλοκότερους αλγόριθμους για την επεξεργασία των πακέτων. Ρεαλιστική λύση λοιπόν από πλευρά κλιμακωσιμότητας ("scalability") σε περιβάλλοντα με υψηλό όγκο κίνησης (όπως τα δίκτυα κορμού ενός Internet Provider ή ένα Data Center) αποτελεί η δειγματοληψία πακέτων με κάποια αναλογία πακέτων.

2.9.1 ***sFlow***

Στα πλαίσια της παρούσας εργασίας, υλοποιήθηκε ένας μηχανισμός που χρησιμοποιεί δειγματοληψία για την εξαγωγή των δεδομένων, χρησιμοποιώντας το sFlow [15]. Το sFlow, συντομογραφία για το "sampled flow", αποτελεί ένα πρότυπο τρόπο για την βιομηχανία όσον αφορά την δειγματοληψία πακέτων. Δίνει τη δυνατότητα εξαγωγής περικομμένων ("truncated") πακέτων παράλληλα με μετρητές για τα interfaces. Όσον αφορά τις μεγάλες υποδομές, το sFlow χρησιμοποιεί δειγματοληψία και έτσι παρέχει μια λύση που είναι δελεαστική (λόγω scalability) στα σύγχρονα περιβάλλοντα.

Σχολιάζοντας την συμβατότητα του sFlow με διάφορα περιβάλλοντα, σημειώνεται πως υποστηρίζεται από δικτυακές συσκευές πολλαπλών κατασκευαστών. Επίσης, τα δεδομένα στην μορφή που εξάγονται είναι αναγνωρίσιμα από μια πληθώρα εργαλείων διαχείρισης και παρακολούθησης. Ένα sFlow σύστημα μπορεί να απαρτίζεται από πολλαπλές συσκευές οι οποίες πραγματοποιούν δειγματοληψία:

- Με βάση κάποια πιθανότητα (πακέτων και ενεργειών του επιπέδου εφαρμογής)
- Ανά χρονικές περιόδους (μετρητές)

Τα δείγματα των πακέτων/ενεργειών και μετρητών αποκαλούνται flow samples και counter samples αντίστοιχα. Τα δεδομένα αποστέλλονται μέσω δεδομενογραμμάτων ("datagrams")

σε ένα μηχάνημα που τα επεξεργάζεται κατάλληλα. Επίσης συνήθως παρουσιάζει αποτελέσματα για την δικτυακή κίνηση με βάση τα δεδομένα που συλλέχθηκαν. Αυτό αποκαλείται συνήθως στην ορολογία sFlow collector.

2.9.1.1 Flow Samples

Βασιζόμενο πάνω σε ένα ρυθμό δειγματοληψίας n, κατά μέσο όρο 1 από n πακέτα (ή άλλες πληροφορίες) επιλέγονται τυχαία. Παρόλο που δεν αποτυπώνονται αποτελέσματα με την μέγιστη ακρίβεια, η προσέγγιση είναι ικανοποιητική [16].

2.9.1.2 Counter Samples

Το διάστημα της μέτρησης ("polling interval") ορίζει την χρονική συχνότητα με την οποία η συσκευή θα στέλνει τους μετρητές για τα interfaces της. Αυτές οι μετρήσεις παρέχουν με άμεσο τρόπο το εύρος ζώνης της γραμμής σε αυτό το χρονικό διάστημα. Σημειώνεται πως σύμφωνα με το [17] τα sFlow counter samples είναι αποδοτικότερος τρόπος παρακολούθησης από το SNMP όταν παρακολουθείται μεγάλο πλήθος interfaces.

2.9.1.3 sFlow datagrams

Τα δεδομένα των δειγμάτων στέλνονται σαν UDP πακέτα στην IP και port που ορίζονται για τον collector (η well known port για τον sFlow collector είναι η 6343). Παρόλο που χρησιμοποιείται το πρωτόκολλο UDP (πιθανόν να χαθεί κάποιο πακέτο καθώς το UDP δεν είναι τόσο αξιόπιστο πρωτόκολλο), η απώλεια δεν έχει σημαντικό αντίκτυπο στην ακρίβεια των μετρήσεων που πραγματοποιεί ένας sFlow agent. Αυτό δικαιολογείται καθώς, στην περίπτωση απώλειας των counter samples θα ανανεωθούν κατάλληλα οι τιμές στο επόμενο χρονικό διάστημα ενώ στην περίπτωση απώλειας των packet samples επί της ουσίας έχουμε μείωση του ρυθμού δειγματοληψίας. Στο εσωτερικού του UDP πακέτου είναι ενθυλακωμένο το sFlow datagram, το καθένα από τα οποία παρέχει τις εξής πληροφορίες:

- Έκδοση sFlow
- Διεύθυνση IP προέλευσης
- Έναν αύξοντα αριθμό
- Το πλήθος των samples που περιέχονται στο sFlow datagram
- Τα ίδια τα δείγματα

2.9.2 *Monitoring in SDN*

Όσον αφορά την δικτυακή παρακολούθηση σε περιβάλλοντα SDN, εκτός από τις μεθόδους που έχουν αναφερθεί παραπάνω, όπως αναφέρθηκε και στην εισαγωγή για το πρωτόκολλο OpenFlow τα ίδια τα switches έχουν την δυνατότητα να αποθηκεύουν μετρητές και άλλες στατιστικές πληροφορίες ανά διαφορετικές κατηγορίες του switch (flows, groups, tables, switch παραπομπή στον πίνακα).

Ιδιαίτερο ενδιαφέρον υπάρχει στην συλλογή και χρήση των δεδομένων δικτυακής κίνησης σε τέτοια περιβάλλοντα, καθώς οι δυνατότητες που έχει μια εφαρμογή ελέγχου τέτοιων δικτύων είναι πολυπληθείς. Η ανάλυση των δεδομένων δίνει τη δυνατότητα στην εφαρμογή να αλλάζει τον τρόπο προώθησης ή απόρριψης των πακέτων δυναμικά και να τροποποιεί τα

πακέτα, ανταποκρινόμενη στην εκάστοτε κατάσταση του δικτύου και ζήτησης των πόρων. Ενδεικτικά κάποια σενάρια θα ήταν:

- Load Balancer – Traffic Shaping
- Ανίχνευση και Αντιμετώπιση επιθέσεων
- Firewall – NAT

Ο τρόπος με τον οποίο είναι διαθέσιμα τα στατιστικά μέσω του OpenFlow πρωτοκόλλου εκ κατασκευής είναι ιδιαίτερα δελεαστικός καθώς υπάρχουν δεδομένα ανά κατηγορίες και επίσης η συλλογή των στατιστικών και χρήση αυτών είναι αμφότερες διεργασίες που ενσωματώνονται στο Network Operating System.

Ωστόσο η παραπάνω διαδικασία ενδέχεται να επιφέρει σημαντικό φόρτο τόσο στα switches, αλλά κυρίως στους controllers καθώς πλέον δεν υπάρχει δυνατότητα, η συλλογή των στατιστικών να γίνεται από ένα μηχάνημα αφοσιωμένο σε αυτό το σκοπό. Ως αποτέλεσμα, η διαδικασία της συλλογής και επεξεργασίας των δεδομένων πρέπει να απορροφηθεί εξολοκλήρου από τον controller, ο οποίος πέραν της προγραμματιστικής λογικής που εφαρμόζει για την προώθηση και επεξεργασία των πακέτων, πρέπει επιπλέον να ενσωματώσει και την επεξεργασία/χρήση αυτών των δεδομένων. Σε περίπτωση που ο controller χρησιμοποιεί όλους τους διαθέσιμους πόρους για την διαδικασία της παρακολούθησης και ανάλυσης του δικτύου, ενδέχεται να δυσκολεύεται να επιτελέσει και τις απλούστερες λειτουργίες της μεταγωγής/δρομολόγησης των πακέτων.

Σύμφωνα με το [18] η επικοινωνία μεταξύ του OpenFlow Controller και του ολοκληρωμένου κυκλώματος του κάθε switch (ASIC), είναι τάξεις μεγέθους μικρότερη ως προς το εύρος ζώνης. Συνεπώς, αυτή η προσέγγιση είναι δύσκολο να εφαρμοστεί σε σύγχρονα περιβάλλοντα καθώς οι απαιτήσεις για την διαχείριση μεγάλου όγκου κίνησης ήδη δημιουργούν πρόβλημα στην επικοινωνία controller – switch.

Βαρύνουσα σημασία έχει το παραπάνω αν αναλογιστούμε πως μια από τις πλέον συνηθισμένες επιθέσεις που καλούνται να αντιμετωπίσουν οι διαχειριστές δικτύων και υπολογιστικών συστημάτων είναι η μαζική άρνηση πρόσβασης σε μια υπηρεσία ("Distributed Denial of Service"). Τέτοιες επιθέσεις χαρακτηρίζονται κυρίως από τεράστιο όγκο κίνησης, με στόχο την αδυναμία χειρισμού αυτής τόσο από τις δικτυακές συσκευές υπερφορτώνοντας το data – plane, αλλά κυρίως από τους ίδιους τους εξυπηρετητές.

Όπως επίσης έχει δειχθεί στο [19], τέτοιες επιθέσεις ενδέχεται να έχουν ως παράπλευρη απώλεια το control plane, καθώς πρακτικά μια τέτοια επίθεση εκτός από την περιττή κατανάλωση δικτυακών πόρων και τον ενδεχόμενο αποκλεισμό άλλων χρηστών από τις υπηρεσίες θα είχε ως παράπλευρη απώλεια την κατάρρευση του ίδιου του control plane. Συνεπώς, ενσωματώνοντας το συλλέκτη στο control plane, η κατάσταση ακόμα περισσότερο.

Ανάλογα με τις ανάγκες της υποδομής θα μπορούσε να χρησιμοποιηθεί κάποιο πρότυπο / πρωτόκολλο δειγματοληψίας. Το sFlow που περιγράψαμε αναλυτικά αν και είναι ιδιαίτερα αποτελεσματικό υλοποιεί sampling βασισμένο σε πακέτα. Όσον αφορά όμως τις υποδομές SDN η δικτυακή κίνηση εντάσσεται σε ροές πακέτων ανάλογα με την πολιτική που εφαρμόζει ο controller και ένας μηχανισμός packet sampling ενδεχομένως να μην ανιχνεύει

ικανοποιητικά (ή και καθόλου) ροές πακέτων οι οποίες αποτελούν πολύ μικρό ποσοστό της κίνησης. Αυτές οι ροές αποκαλούνται "mice flows", ενώ οι δεσπόζουσες ροές με το μεγαλύτερο ποσοστό κίνησης "elephant flows". Υπάρχουν εργαλεία που εξάγουν στατιστικά για κάθε ροή πακέτων ξεχωριστά, όπως το NetFlow, αλλά η περεταίρω επεξεργασία που χρειάζεται για την σάρωση και κατηγοριοποίηση των πακέτων το καθιστά, σημαντικά πιο απαιτητικό από την άποψη των πόρων, σε σύγκριση με το απλούστερο sFlow.

Ενδεικτικά παρατίθενται διαφορετικοί τρόποι για την συλλογή στατιστικών σε περιβάλλοντα SDN:

- FleXam
- FlowSense

2.9.2.1 FleXam

Το πρόβλημα που εντοπίζεται στο [20] είναι η περιορισμένη πρόσβαση σε πληροφορίες σχετικές με το κάθε πακέτο (το περιεχόμενο για παράδειγμα), όπως επίσης και η δυσκολία που εμπεριέχει η δειγματοληψία βασισμένη σε ροή πακέτων. Το FleXam αποτελεί μια ευέλικτη επέκταση του OpenFlow Firmware με στόχο την καλύτερη παροχή δεδομένων σε εφαρμογές παρακολούθησης και ασφάλειας. Ιδιαίτερο ενδιαφέρον παρουσιάζει η ικανότητα του μηχανισμού να πραγματοποιεί δειγματοληψία βασισμένη στην ροή πακέτων (flow–based sampling), καθώς όπως αναφέρθηκε η απλή δειγματοληψία πακέτων ενδεχομένως να μην εντοπίζει επιτυχώς τις μικρές σε όγκο κίνησης ροές. Όπως έχει αναφερθεί η flow–based δειγματοληψία είναι μια περίπλοκη διαδικασία και απαιτεί σημαντικές αλλαγές στον τρόπο επεξεργασίας των πακέτων, ενδεχομένως και στο υλικό των switches. Παρόλα αυτά, στο πρωτόκολλο OpenFlow ο τρόπος λειτουργίας βασίζεται σε κανόνες για συγκεκριμένες ροές πακέτων, κανόνες οι οποίοι είναι συνδεδεμένοι με διάφορες ενέργειες (actions). Συνεπώς η υλοποίηση της δειγματοληψίας σαν επιπλέον **Action** του πρωτοκόλλου OpenFlow, επιτρέπει την παρακολούθηση του δικτύου με τον επιθυμητό τρόπο χωρίς σημαντικό overhead λειτουργώντας παράλληλα με τον ρυθμό μετάδοσης της γραμμής (line rate).

Ο τρόπος δειγματοληψίας που παρέχει η επέκταση αυτή μπορεί να είναι:

- **Στοχαστικός:** βασισμένος σε μια προκαθορισμένη πιθανότητα. Σε κάθε πακέτο αντιστοιχίζεται ένας τυχαίος αριθμός, ομοιόμορφα ανάμεσα στο 0 και το 1. Εάν ο αριθμός είναι μικρότερος από την προκαθορισμένη πιθανότητα τότε το πακέτο επιλέγεται για δειγματοληψία.
- **Ντετερμινιστικός:** βασισμένος πάνω σε ένα μοτίβο (για κάθε **k** πακέτα διάλεξε **m** από αυτά αγνοώντας τα **δ** πρώτα). Αντίστοιχα σε αυτή την περίπτωση εάν ισχύει η ανίσωση *(((* ***Received_Packet_Counter*** *–* ***δ)*** *%* ***k)*** *<* ***m)*** τότε το πακέτο επιλέγεται για δειγματοληψία. Σημειώνεται πως τα OpenFlow switches εκ κατασκευής διαθέτουν μετρητές για τον αριθμό των πακέτων που έχουν αντιστοιχισθεί σε ένα flow (Received_Packet_Counter).

Σχετικά με αυτή την προσέγγιση παρατηρούνται τα εξής:

- **Τροποποίηση του OpenFlow firmware:** Δημιουργώντας ένα νέο action συνεπάγεται πως πρέπει να υπάρξει αναβάθμιση του firmware σε όλες τις OpenFlow συσκευές. Μια τέτοια ενέργεια μπορεί να μην είναι επιθυμητή για λόγους ασφάλειας, ή πλήθους συσκευών.
- **Απαραίτητη η ανάπτυξη αυτόνομης εφαρμογής:** Καθώς η εν λόγω επέκταση βασίζεται σε τροποποίηση του OpenFlow πρωτοκόλλου πρέπει να δημιουργηθεί ένας μηχανισμός μέσω του οποίου θα αντλούνται τα δεδομένα που συλλέγονται.

2.9.2.2 FlowSense

Το FlowSense [21] είναι ένας μηχανισμός ο οποίος στοχεύει στην παρακολούθηση δικτύων βασισμένα στην SDN αρχιτεκτονική. Στόχος είναι η διαδικασία να πραγματοποιείται με το ελάχιστο δυνατό κόστος. Αυτό επιτυγχάνεται με παθητικό τρόπο, αιχμαλωτίζοντας και αναλύοντας την κίνηση που αφορά την σηματοδοσία ανάμεσα στις δικτυακές συσκευές και το επίπεδο ελέγχου τους. Αναλύοντας περαιτέρω, χρησιμοποιούνται τα PacketIn και FlowRemoved μηνύματα που ανταλλάσσονται κατά την άφιξη ενός νέους ή κατά την εκπνοή, ενός flow. Κατά αυτόν τον τρόπο δίνεται η δυνατότητα υπολογισμού της χρησιμοποίησης στις ζεύξεις ανάμεσα στις συσκευές, χρησιμοποιώντας τον όγκο της κίνησης που έχει αντιστοιχιστεί με τον κάθε κανόνα στο διάστημα που ήταν ενεργός.

Παρατηρούνται τα εξής για τον μηχανισμό:

- Μετρήσεις πραγματοποιούνται σε διακριτά σημεία του χρόνου.
- Μέση χρησιμοποίηση όχι στιγμιαία.
- Πρόβλημα σε περίπτωση που η κίνηση σηματοδοσίας είναι μικρή (για λόγους scalability).

3

Σχεδιαστικές Αρχές

Οι σύγχρονες υποδομές είναι στην πλειοψηφία τους ανομοιογενείς. Κάθε μια από αυτές έχει επιμέρους χαρακτηριστικά και ιδιαιτερότητες, τόσο εκ φύσεως (για παράδειγμα ενσύρματες – ασύρματες υποδομές, μεγάλα – μικρά δίκτυα) όσο και εκ κατασκευής/διαμόρφωσης (virtualization ή όχι, SDN enabled ή όχι).

- SDN Infrastructures
- Datacenters
- Cloud Computing facilities
- Multi – tenant research environments
- Wireless testbeds
- Transit Networks
- Enterprise Networks
- Small Office Home Office

Στόχος είναι η δημιουργία ενός μηχανισμού – πλαισίου το οποίο θα επιτρέπει την παρακολούθηση ποικίλων υποδομών χωρίς να περιορίζεται από τις επιμέρους ιδιομορφίες και περιορισμούς, αλλά ταυτόχρονα χωρίς να απαιτεί συγκεκριμένες επεμβατικές ενέργειες σε συσκευές και πρωτόκολλα. Επιθυμητή είναι η παρακολούθηση τόσο της τρέχουσας κίνησης, αλλά και η διατήρηση μιας αποθηκευτικής δομής που θα περιέχει τα στοιχεία, λειτουργώντας σαν ιστορικό.

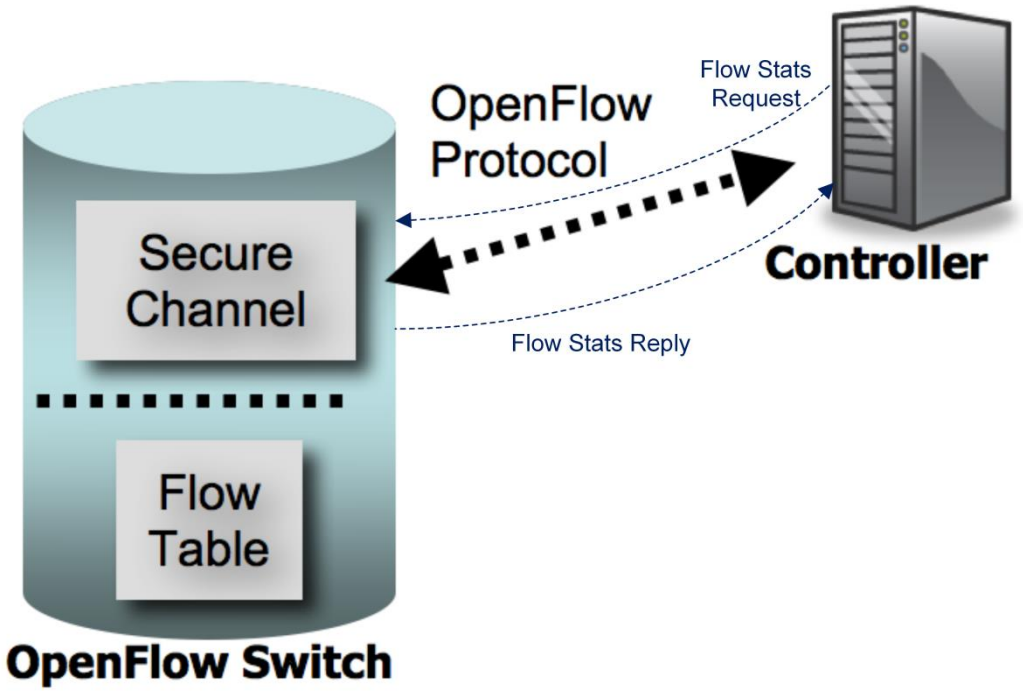
Ομαδοποιούμε τις υποδομές με βάση τα κύρια χαρακτηριστικά τους σε:

- SDN
- Cloud – Datacenters
- Wireless

Ο διαχωρισμός βασίστηκε στην ανάγκη αντιμετώπισης των τριών κατηγοριών αυτών με διαφορετικό τρόπο όσον αφορά τον τρόπο επικοινωνίας των διαφόρων κόμβων σε αυτές (π.χ. υποστήριξη OpenFlow στις δικτυακές συσκευές, χρήση δικτυακής συσκευής για την ενσύρματη επικοινωνία, ασύρματη επικοινωνία).

3.1 Παρουσίαση Αρχιτεκτονικής σε SDN

Αρχικά θα γίνει παρουσίαση της αρχιτεκτονικής και ανάλυση του τρόπου λειτουργίας του μηχανισμού για περιβάλλοντα SDN. Η ιδέα βασίστηκε στα μηνύματα που ανταλλάσουν εκ φύσεως, στοιχεία που επικοινωνούν μέσω του πρωτόκολλου OpenFlow. Όπως έχει αναφερθεί και κατά την περιγραφή του πρωτοκόλλου και των δυνατοτήτων του, υποστηρίζεται εκ κατασκευής ένας μηχανισμός όπου ο controller ζητά ρητά από τα switches την αποστολή δεδομένων δικτυακής κίνησης (Σχήμα 3.1). Αφού λάβει την αντίστοιχη απάντηση τα επεξεργάζεται καταλλήλως.



Σχήμα 3.1 Εγγενής Μέθοδος Συλλογής Στατιστικών μέσω του OpenFlow

Αυτό όπως έχει αναφερθεί στο κεφάλαιο 2 έχει κάποια μειονεκτήματα. Αφενός όπως θα αποδειχθεί στην πορεία (Κεφάλαιο 5, Αξιολόγηση Υλοποίησης) η συχνή χρήση του μηχανισμού είναι ιδιαίτερα ακριβή από άποψη υπολογιστικού κόστους. Αφ ετέρου (αναφερόμαστε στα per – flow στατιστικά) τα δεδομένα στα οποία έχει πρόσβαση ο controller, αφορούν ενεργά flow rules (βρίσκονται αυτή τη στιγμή στα flow tables των switches), έτσι δεν υπάρχουν δεδομένα για προηγούμενες καταστάσεις. Προκειμένου να κρατούνται ιστορικά στοιχεία πρέπει η εφαρμογή εκτός από την ήδη ακριβή υπολογιστικά διαδικασία να επιβαρύνει περισσότερο την κατάσταση αποθηκεύοντας τα δεδομένα κατάλληλα. Κάτι τέτοιο όμως προϋποθέτει σωστό προγραμματισμό των requests έχοντας ως κριτήριο τα χρονικά διαστήματα ανά τα οποία εκπνέουν τα flows.

Όσον αφορά την προτεινόμενη υλοποίηση, αρχικά αναπτύχθηκε ένας μηχανισμός αποθήκευσης των flow εγγραφών σε μια δομή. Σημείο κλειδί, για την λειτουργία του

μηχανισμού είναι η κίνηση σηματοδοσίας που ανταλλάσουν, στα πλαίσια του OpenFlow πρωτοκόλλου, controller και switches. Χρησιμοποιώντας αυτή την κίνηση δημιουργούνται καταχωρήσεις σε δυο δομές: (1) στην δομή "Active" κατά την στιγμή εγκατάστασης των κανόνων στα switches (2) στην δομή "Expired" όπου μεταφέρονται οι εγγραφές από την δομή "Active" κατά την εκπνοή του χρονικού ορίου που έχει τεθεί στον κάθε κανόνα (ιστορική παράθεση). Σημειώνεται πως, πανομοιότυποι κανόνες σε διαφορετικές χρονικές στιγμές διαχωρίζονται και αποθηκεύονται ξεχωριστά. Το παραπάνω σύστημα αποτελεί μια αποθηκευτική δομή για την δικτυακή κίνηση (**"Flow Repository"**).

Στη συνέχεια, ενσωματώθηκε μηχανισμός για την πραγματοποίηση δειγματοληψίας της κίνησης, χρησιμοποιώντας το sFlow και υλοποιήθηκε ο αντίστοιχος συλλέκτης δειγμάτων. Η διαδικασία που επιτελεί ο συλλέκτης των δειγμάτων, αποτελεί αναπόσπαστο κομμάτι του συνολικού μηχανισμού. Μπορεί να λειτουργήσει είτε ως εξάρτημα του OpenFlow controller είτε ως αυτόνομη εφαρμογή.

Ωστόσο όπως έχει αναφερθεί, το sFlow υλοποιεί δειγματοληψία κατά πακέτα. Συνεπώς, απαραίτητη ήταν και η ανάπτυξη μιας αλγοριθμικής διαδικασίας για την επεξεργασία των sFlow samples. Η επεξεργασία αυτή έχει ως στόχο, να εντοπίσει την ροή των πακέτων στην οποία ανήκει το κάθε δείγμα ώστε να ανανεωθούν καταλλήλως οι μετρητές. Σημειώνεται πως η παραπάνω διαδικασία μπορεί να φαίνεται ιδιαίτερα δαπανηρή από άποψη πόρων, ωστόσο πριν την προσπάθεια αναδημιουργίας του flow και εύρεσης του αντίστοιχου flow rule στο οποίο ανήκει το πακέτο, αναζητείται η ύπαρξη exact match flow rule. Το παραπάνω αποκτά νόημα, καθώς η προβλεπόμενη συμπεριφορά βάση του OpenFlow specification είναι οι νέοι κανόνες να είναι όσο το δυνατόν πιο συγκεκριμένοι. Πρακτικά, η αποθήκευση των flow rules στις δομές γίνεται με την βοήθεια συναρτήσεων κατακερματισμού οπότε η διαδικασία γίνεται υπολογιστικά ευκολότερη με την εφαρμογή της ίδιας συνάρτησης στα ομότιμα πεδία του sFlow sample, και αναζήτησης με βάση το αποτέλεσμα αυτό.

Η συλλογή είναι υλοποιημένη ξεχωριστά από την διαδικασία της αντιστοίχισης των δειγμάτων σε ενεργές flow εγγραφές. Αυτό προτιμήθηκε ώστε οι λειτουργίες να είναι διαχωρισμένες με απώτερο στόχο χαμηλότερο load στο Flow Repository, δυνατότητα μεταφοράς των διεργασιών σε ξεχωριστά μηχανήματα, δυνατότητα επιλογής διαφορετικού συλλέκτη sFlow (τα δεδομένα βεβαίως θα πρέπει να είναι σε συμφωνία με αυτά που αναμένει το Flow Repository προς επεξεργασία) χωρίς να αλλάζει η λειτουργία του Flow Repository.

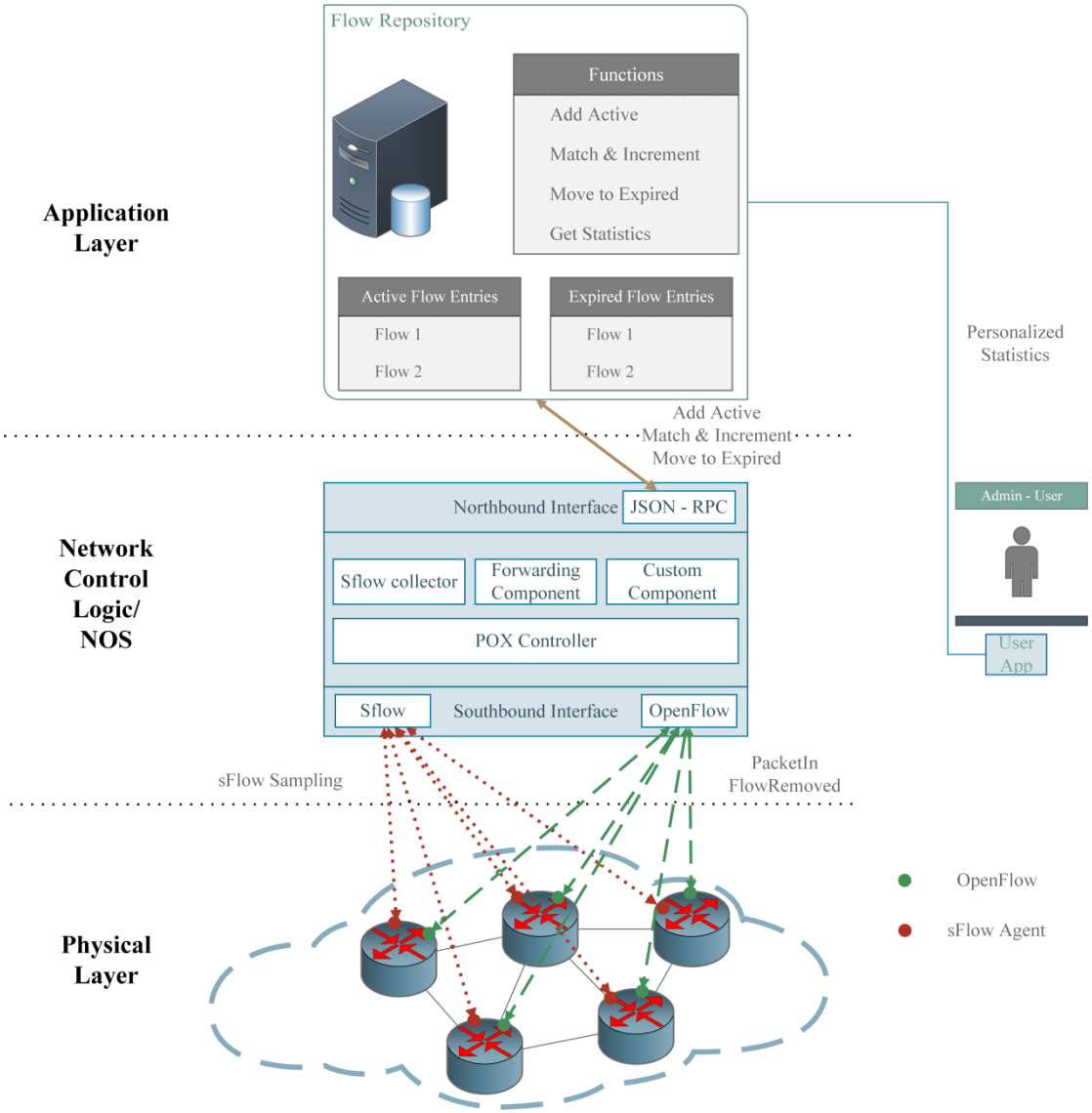
Αξίζει να σημειωθεί πως παρόλο που η δειγματοληψία αφορά αμιγώς τα πακέτα και όχι πακέτα που ανήκουν σε κάποια συγκεκριμένη ροή (flow), εμφανίζονται όλοι οι κανόνες που έχουν εγκατασταθεί, ανεξαρτήτως αν έχουν επεξεργαστεί δείγματα sFlow που κατηγοριοποιούνται στην αντίστοιχη ροή. Το παραπάνω είναι εφικτό χάρη στο τρόπο λειτουργίας του πρωτοκόλλου OpenFlow και πιο συγκεκριμένα των μηνυμάτων **PacketIn** και **FlowRemoved** που ανταλλάσσονται ανάμεσα στον controller και τα switches.

Βέβαια είναι διαδεδομένο σε περιβάλλοντα υψηλής κίνησης να είναι εγκατεστημένοι κάποιοι default κανόνες προληπτικά οι οποίοι δεν εκπνέουν ποτέ ώστε να μην εμπλέκεται συνεχώς ο controller [18] (και αντίστοιχα δεν στέλνουν FlowRemoved events). Όμως αυτοί οι κανόνες απλώς θα παραμείνουν στην δομή "Active" και θα ανανεώνονται μέσω της δειγματοληψίας.

Κατά την άφιξη ενός πακέτου για το οποίο δεν υπάρχει κανόνας στο switch δημιουργείται ένα PacketIn event και πλέον αναγνωρίζεται το αντίστοιχο flow που θα δημιουργηθεί. Έτσι

είναι δυνατός ο εντοπισμός και των "mice" flows που ενδεχομένως να μην ανιχνεύονταν απλώς με την χρήση του sFlow.

Ακολουθεί η αρχιτεκτονική διάταξη της υλοποίησης (Σχήμα 3.2 Προτεινόμενη Αρχιτεκτονική βασισμένη σε SDN υποδομές).



Σχήμα 3.2 Προτεινόμενη Αρχιτεκτονική βασισμένη σε SDN υποδομές

Αναλύοντας συνολικά την αρχιτεκτονική:

- Κατά την άφιξη νέων πακέτων, τα switches μέσω του Southbound API αποστέλλουν PacketIn Events προς τον controller.
- Ο Controller αποφασίζει τον τρόπο προώθησης τους με βάση το Forwarding Component. Στη συνέχεια εάν πρόκειται να εγκατασταθεί κάποιο flow, ενημερώνεται το Custom Component το οποίο με την σειρά του χρησιμοποιεί το Northbound interface για να ενημερώσει το Flow Repository όπου δημιουργούνται νέες καταχωρήσεις στην δομή "Active".

- Στις δικτυακές συσκευές της διάταξης πραγματοποιείται δειγματοληψία. Τα δείγματα αυτά αποστέλλονται στον συλλέκτη μέσω του Southbound interface ο οποίος τα στέλνει στο Flow Repository για περεταίρω επεξεργασία.
- Όταν εκπνέουν χρονικά στέλνονται FlowRemoved events από τα αντίστοιχα switches. Τα events αυτά χειρίζονται από το Custom Component μέσω του οποίου ξεκινάει η διαδικασία για την μεταφορά των αντίστοιχων καταχωρήσεων στον πίνακα με τα Expired Flows, μαζί με τις μετρήσεις που συλλέχθηκαν στο ενδιάμεσο από τον sFlow collector.
- Τέλος τα δεδομένα είναι διαθέσιμα στον χρήστη από το Flow Repository για κάθε μελλοντική χρήση.

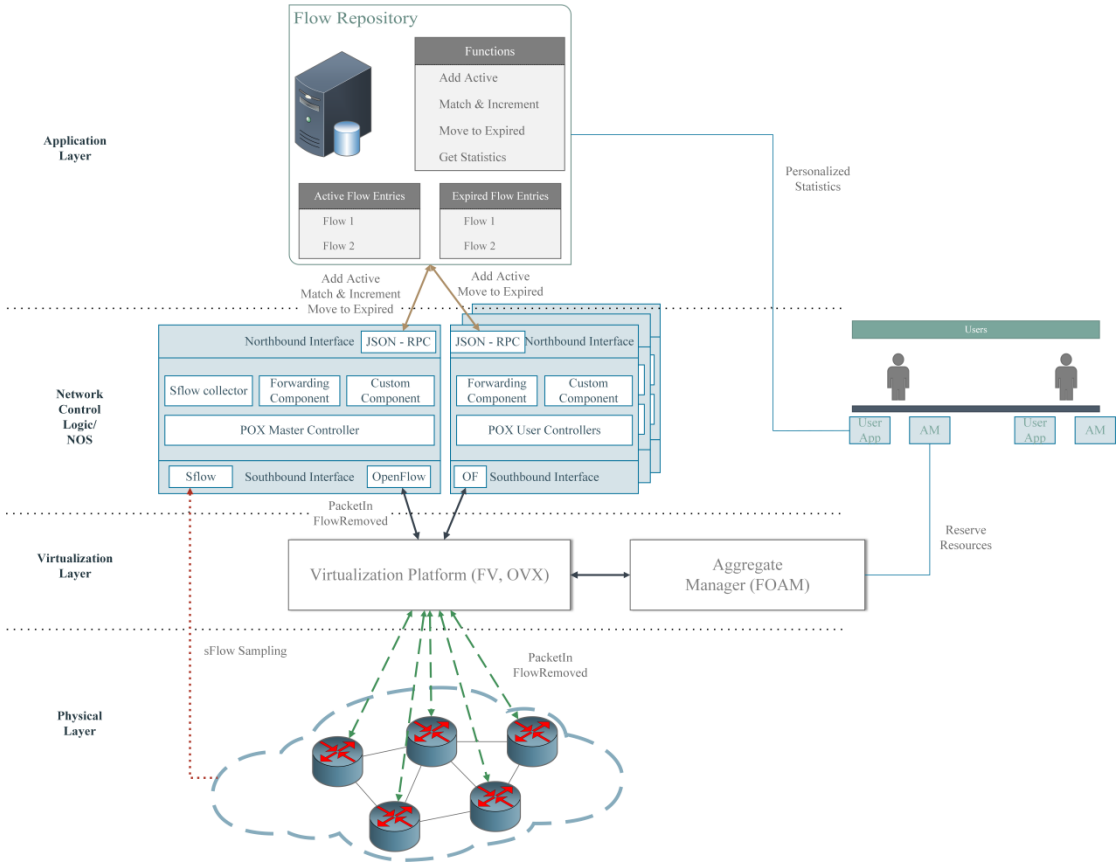
3.2 SDN Multitenant

Σε αυτή την ενότητα στόχος είναι να παρουσιάσουμε μια αναπροσαρμοσμένη αρχιτεκτονική για υποδομές Δικτύων Καθοριζόμενων από Λογισμικό στις οποίες πραγματοποιείται διαμοιρασμός πόρων σε πολλαπλούς χρήστες. Ενδεικτικό παράδειγμα τέτοιων υποδομών είναι η πλατφόρμα GENI [22] (Global Environment for Network Innovations), η οποία προσφέρει ένα εικονικό εργαστήριο για έρευνα και εκπαίδευση πάνω σε δικτύωση υπολογιστών και κατανεμημένα συστήματα. Οι υποδομές του GENI απαρτίζονται από ομόσπονδους εταίρους οι οποίοι προσφέρουν πόρους στους διάφορους χρήστες συνεργατικά. Συνήθως, για την εικονικοποίηση πόρων, χρησιμοποιείται ο FlowVisor. Γενικότερα σε αντίστοιχες υποδομές υπεύθυνος για τον διαμοιρασμό των εικονικών πόρων είναι ο "Aggregate Manager". Η διαδικασία του διαμοιρασμού συνεπάγεται τόσο την άμεση αλληλεπίδραση με το επίπεδο που πραγματοποιεί την εικονικοποίηση, το Virtualization Layer, όσο και την αλληλεπίδραση με τους χρήστες (συνήθως έμμεσα χρησιμοποιώντας κάποιο ενδιάμεσο API) προκειμένου να δέχεται αιτήματα για την εκμίσθωση πόρων κλπ.

Ειδικότερα, όσον αφορά τις OpenFlow υποδομές της πειραματικής πλατφόρμας GENI, ο Aggregate Manager καλείται FOAM [23] (Flowvisor OpenFlow Aggregate Manager), και μέσω αυτού επιτρέπεται στους χρήστες η πρόσβαση σε OpenFlow πόρους (μέσω του FlowVisor). Ο FOAM χρησιμοποιεί RSpecs για την δέσμευση πόρων. Τα RSpecs [23] είναι κατάλληλα διαμορφωμένα xml με τα οποία χρήστες ζητούν πρόσβαση σε συγκεκριμένους πόρους. Ο FOAM επικοινωνεί με τον FlowVisor και τον ενημερώνει για τους πόρους που αντιστοιχούν στον κάθε χρήστη, ώστε να ανανεωθεί καταλλήλως το flowspace του τεμαχίου κάθε χρήστη.

Ωστόσο, στα πλαίσια της παρούσας διπλωματικής εργασίας δεν θα μελετηθεί ο τρόπος λειτουργίας του FOAM. Επικεντρωνόμαστε: (1) στην δυνατότητα του μηχανισμού να διαχωρίζει την κίνηση με βάση το flowspace του FlowVisor, και (2) στην δυνατότητα πρόσβασης των χρηστών στην κίνηση που συλλέχθηκε για το δικό τους flowspace.

Ακολούθως παρουσιάζεται η αναπροσαρμοσμένη αρχιτεκτονική για περιβάλλοντα πολλαπλών ενοίκων (Σχήμα 3.3 Αρχιτεκτονική SDN Multitenant).



Σχήμα 3.3 Αρχιτεκτονική SDN Multitenant

Σε σύγκριση με την προηγούμενη προσέγγιση, δεν παρουσιάζονται σημαντικές διαφορές στον τρόπο λειτουργίας του μηχανισμού. Πλέον η κίνηση σηματοδοσίας κατανέμεται μέσω του FlowVisor, ο οποίος ως ενδιάμεσος Controller προωθεί με βάση το flowspace τα PacketIn και FlowRemoved messages από τα διάφορα switches, στους αρμόδιους Controllers. Όσον αφορά το flowspace, αναπτύσσεται μηχανισμός ο οποίος ενημερώνει το Flow Repository για το flowspace των διάφορων χρηστών.

Όπως αναλύθηκε και στην προηγούμενη ενότητα οι Controllers μέσω του Northbound interface αλληλεπιδρούν με το Flow Repository. Πλέον όμως, το Flow Repository κατά την δημιουργία μιας νέας εγγραφής προσπαθεί να την κατατάξει στο Slice κάποιου χρήστη, με βάση το συνολικό flowspace του FlowVisor. Με αυτόν τον τρόπο, δίνεται η δυνατότητα στους χρήστες να ζητήσουν όλες τις πληροφορίες που έχουν συλλεχθεί για το δικό τους slice.

Σημειώνεται πως συνήθως σε τέτοιες υποδομές υπάρχει ένας κύριος (master) controller ο οποίος τρέχει και το εξάρτημα του συλλέκτη, και οι controllers των επιμέρους χρηστών που επικοινωνούν με την υποδομή μέσω του FlowVisor.

4

Ανάλυση Υλοποίησης

Όπως αναφέρθηκε και στο κεφάλαιο 3, η αρχιτεκτονική βασίζεται στην υλοποίηση του Flow Repository, των εξαρτημάτων του OpenFlow Controller και του συλλέκτη για τα δείγματα sFlow καθώς και σε μια εφαρμογή χρήστη για την επικοινωνία με το Flow Repository. Στην συνέχεια θα αναλυθεί εκτενώς ο τρόπος υλοποίησης του μηχανισμού. Αρχικά παρουσιάζονται ορισμένα στοιχεία βασικά για την υλοποίηση.

4.1 Πρότυπο JSON

Το ακρώνυμο αναλύεται σε JavaScript Object Notation [24], και αποτελεί ένα ανοικτό πρότυπο που χρησιμοποιεί κείμενο αναγνώσιμο από τον άνθρωπο για την κωδικοποίηση δεδομένων (με απώτερο στόχο την μετάδοση τους) που έχουν την δομή "χαρακτηριστικό" – "τιμή" ("attribute" – "value"). Κυρίως χρησιμοποιείται για την μετάδοση δεδομένων ανάμεσα σε server και web application, σαν εναλλακτικό του XML. Παρόλο που προέρχεται, όπως φαίνεται και από την ονομασία του, από την γλώσσα προγραμματισμού JavaScript το JSON είναι μια μορφή δεδομένων ευρέως διαδεδομένη και πλήρως ανεξάρτητη από την γλώσσα προγραμματισμού.

4.2 Βιβλιοθήκη Pickle

Το εξάρτημα αυτό υλοποιεί έναν θεμελιώδη αλλά ισχυρό αλγόριθμο για την μετατροπή της δομής των αντικειμένων της Python σε μια αλληλουχία από bytes, ώστε να μπορούν να περάσουν από μια εφαρμογή σε μια άλλη, να αποθηκευτούν σε αρχείο κλπ. Η διαδικασία αυτή είναι γνωστή ως "pickling" ενώ αυτή της επαναφοράς στην αρχική δομή των αντικειμένων "unpickling".

4.3 Πρωτόκολλο JSON – RPC

Αποτελεί ένα RPC (remote procedure call – δυνατότητα κλήσης μιας μεθόδου η οποία είναι υλοποιημένη σε ένα απομακρυσμένο πρόγραμμα) πρωτόκολλο κωδικοποιημένο στο μορφή JSON [25]. Στα πλαίσια του πρωτοκόλλου επιτρέπεται σε ένα σύστημα να στείλει ενημερώσεις (πληροφορίες στις οποίες ο server δεν χρειάζεται να απαντήσει) όπως επίσης και να πραγματοποιεί κλήσεις στον server (οι οποίες αναγκαστικά πρέπει να απαντηθούν). Ο server έχει την δυνατότητα να απαντά σε αυτές με σειρά της επιλογής του.

Παρατίθενται οι διαφορετικές κατηγορίες μηνυμάτων του πρωτοκόλλου:

- **Request (*κλήση μεθόδου*)**
  Μια απομακρυσμένη μέθοδος καλείται αποστέλλοντας κατάλληλο αίτημα στον server. Το αίτημα αυτό είναι ένα αντικείμενο το οποίο σειριοποιείται χρησιμοποιώντας την δομή του JSON. Η δομή αυτή απαρτίζεται από ένα διατεταγμένο σύνολο ζευγών ονομάτων/τιμών (name/value). Τα ζεύγη του συγκεκριμένου μηνύματος είναι:
  - **jsonrpc:** Τιμή είναι η έκδοση του JSON – RPC.
  - **params:** Τιμή είναι ένας πίνακας με αντικείμενα τα οποία αποτελούν τις παραμέτρους που θα περαστούν κατά την κλήση της μεθόδου.
  - **id:** Τιμή είναι ένα αναγνωριστικό. Μπορεί να είναι οποιουδήποτε τύπου και χρησιμοποιείται για την αντιστοίχιση της απόκρισης (**Response**) με το αντίστοιχο αιτήματος στο οποίο απαντά.
  - **method:** Τιμή είναι μια συμβολοσειρά που περιέχει το όνομα της μεθόδου που επιθυμεί ο αποστολέας να επικαλεστεί.
- **Response**
  Όταν η μέθοδος που κλήθηκε ολοκληρωθεί, τότε η υπηρεσία πρέπει να απαντήσει στο αίτημα με ένα Response. Η απόκριση είναι ένα αντικείμενο το οποίο επίσης σειριοποιείται χρησιμοποιώντας τη δομή του JSON. Τα ζεύγη για τον συγκεκριμένο τύπο μηνύματος είναι:
  - **jsonrpc:** Έκδοση του JSON – RPC.
  - **result:** Τιμή είναι το αντικείμενο που επιστρέφεται από την μέθοδο που επικαλέστηκε μέσω του αντίστοιχου Request. Πρέπει να είναι null στην περίπτωση που προέκυψε κάποιο σφάλμα κατά την κλήση της μεθόδου.
  - **id:** Τιμή είναι ένα αναγνωριστικό. Όπως αναφέρθηκε η τιμή πρέπει να είναι ίδια με την τιμή του αντίστοιχου Request.
  - **error:** Τιμή είναι ένα αντικείμενο Error (με πληροφορίες όπως τον κωδικό του σφάλματος, μήνυμα σφάλματος, δεδομένα πάνω σε αυτό) σε περίπτωση που προέκυψε κάποιο σφάλμα κατά την κλήση της μεθόδου, null σε οποιαδήποτε άλλη περίπτωση.
- **Notification**
  Τα μηνύματα Notification αποτελούν μια ειδική έκδοση των μηνυμάτων Request, για τα οποία δεν απαιτείται απάντηση με Response. Όπως και οι δυο τύποι

μηνυμάτων που αναφέρθηκαν παραπάνω, το Notification έχει ένα αντικείμενο που σειριοποιείται με χρήση της δομής JSON. Τα ζεύγη name/value είναι όμοια με αυτά που έχουν περιγραφεί για το μήνυμα Request, με μια εξαίρεση. Το πεδίο id πρέπει να έχει τιμή null (δεν χρειάζεται αναγνωριστικό καθώς δεν προβλέπεται απόκριση του server).

Ένα ενδεικτικό παράδειγμα επικοινωνίας:

- **-->** Αντικείμενο που στέλνεται στον server
- **<--** Αντικείμενο που λαμβάνεται από τον server
- Εκτέλεση της εντολής **server.echo("Hello JSON RPC")**
  --> {"jsonrpc": "2.0", "method": "echo", "params": ["Hello JSON-RPC"], "id": 1}
  <-- {"jsonrpc": "2.0", "result": "Hello JSON-RPC", "error": null, "id": 1}

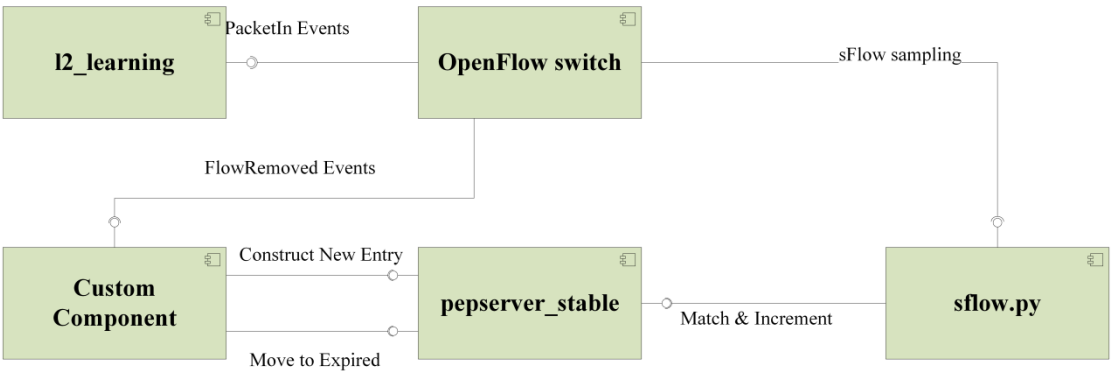
Η επικοινωνία που προβλέπεται στα πλαίσια του πρωτοκόλλου JSON – RPC μπορεί να μεταφερθεί μέσω μηνυμάτων HTTP, HTTPS ή σαν ροή δεδομένων μέσω Sockets. Συγκεκριμένα στην προκειμένη περίπτωση, η επικοινωνία του Flow Repository και των λειτουργιών που προσφέρει, με τα διαφορετικά στοιχεία που τις χρησιμοποιούν επιτυγχάνεται με την χρήση HTTP μηνυμάτων και ειδικότερα όσον αφορά τα HTTP Requests, μηνύματα POST.

Όσον αφορά το κομμάτι της υλοποίησης, έχοντας υπ όψιν τις διάφορες δυνατότητες που είναι επιθυμητό να παρέχονται, δημιουργήθηκαν οι απαραίτητες συναρτήσεις που θα είναι διαθέσιμες προς κλήση. Οι συναρτήσεις αυτές πρέπει να γίνουν όλες register στον JSON – RPC server ώστε να μπορεί να εξυπηρετήσει Requests με τιμές του πεδίου "Method" το όνομα αυτών των συναρτήσεων. Σημειώνεται πως τα ορίσματα που χρησιμοποιούνται για την απομακρυσμένη κλήση δεν μπορούν να μεταφερθούν στην αρχική τους μορφή και πρέπει να κωδικοποιηθούν (για παράδειγμα σειριοποίηση με χρήση των βιβλιοθηκών JSON ή Pickle).

4.4 Ανάλυση του αρχείου pepserver_stable.py

Όπως έχει αναφερθεί, μεγάλο μέρος του Flow Repository είναι βασισμένο στην λειτουργικότητα που προσφέρει το API του JSON – RPC. Στο συγκεκριμένο αρχείο ορίζονται οι μέθοδοι/συναρτήσεις που είναι διαθέσιμες προς χρήση μέσω του API, όπως επίσης και οι υπόλοιπες που χρησιμοποιούνται εσωτερικά για διάφορες λειτουργίες που είναι απαραίτητες στις προηγούμενες.

Αρχικά παρατίθεται ένα Component Diagram (Σχήμα 4.1 Component Diagram για τον μηχανισμό), ώστε να φαίνεται προγραμματιστικά ο τρόπος λειτουργίας του μηχανισμού (πως χρησιμοποιούνται τα Interfaces).



Σχήμα 4.1 Component Diagram για τον μηχανισμό

Περιγράφονται οι δομές δεδομένων που χρησιμοποιούνται ευρέως από διάφορες συναρτήσεις:

- Dictionaries: Δομή δεδομένων ενσωματωμένη στην Python, απαρτίζεται από ένα μη διατεταγμένο ζεύγος "κλειδιού" – "τιμής". Τα Dictionaries αναπαρίστανται συμβολικά ως {key1: value1, key2: value2 …}. Ακολουθούν οι κύριες δομές dictionaries που χρησιμοποιούνται
  - ✓ Active, αποθήκευση των Active flow entries
  - ✓ Expired, αποθήκευση των Expired flow entries
  - ✓ Mac_table, χρησιμοποιείται για την αντιστοίχιση Source MAC address με Ingress port
  - ✓ Mapper, χρησιμοποιείται για την αντιστοίχιση των πεδίων του sflow, με αυτά του OpenFlow
- Lists:
  - ✓ Flowspace, περιέχει το flowspace που έχει ορισθεί στον FlowVisor

Συναρτήσεις που ορίζονται:

- **Construct_new_entry(serialized_match)**
  Η συνάρτηση δέχεται σαν όρισμα μια σειριοποιημένη τουπλα (tuple) της μορφής (*OpenFlow Match*, *Datapath ID του OpenFlow switch*, *timestamp*)

  Χρησιμοποιείται η συνάρτηση **construct_hashed_key** με όρισμα το αντικείμενο match για την δημιουργία μιας hashed τιμής με βάση το όρισμα. Εάν δεν υπάρχει άλλη εγγραφή για το συγκεκριμένο dpid, δημιουργείται καταχώρηση με κλειδί το dpid και τιμή ένα κενό dictionary.

  Κατόπιν, στο Active, δημιουργείται μια νέα εγγραφή (αν δεν υπάρχει ήδη) με κλειδιά το dpid και την τιμή της συνάρτησης κατακερματισμού, και τιμή ένα νέο dictionary με την δομή που φαίνεται στον παρακάτω πίνακα (Πίνακας 4.1 Ζεύγη Key – Value για κάθε καταχώρηση).

Τέλος στο dictionary mac_table, δημιουργείται μια αντιστοίχιση μεταξύ του ingress port ενός OpenFlow switch με συγκεκριμένο dpid και source MAC address.

Keys	Values
"Counters"	Dictionary με τους μετρητές που χρησιμοποιούνται
"Match"	Dictionary που έχεις ως keys τα πεδία του OpenFlow Match και ως values τις τιμές των πεδίων αυτών
"Timestamps"	Dictionary με keys "start", "end" και τις αντίστοιχες τιμές. Σημειώνεται πως το "end" αποκτά τιμή κατά την αφαίρεση της εγγραφής
"Slice_Owner"	Slice στο οποίο εντάσσεται αυτή η εγγραφή, κατηγοριοποίηση με χρήση της συνάρτησης assign_flowspace

Πίνακας 4.1 Ζεύγη Key – Value για κάθε καταχώρηση

- **Move_to_expired(serialized_match)**
  Η συνάρτηση δέχεται πανομοιότυπο όρισμα με αυτό της προηγούμενης. Η λειτουργία είναι ελαφρώς διαφορετική. Εκ νέου, εφαρμόζεται στο match (το match πλέον αφορά το flow που εξέπνευσε) η ίδια συνάρτηση κατακερματισμού.

  Στόχος αυτή τη φορά είναι να εξετάσουμε αν υπάρχει στο dictionary Active για το συγκεκριμένο dpid (Active[dpid]), η εγγραφή με κλειδί το hashed value που υπολογίσαμε πριν. Αν δεν υπάρχει (κάτι τέτοιο δεν είναι αναμενόμενο αλλά μπορεί να συμβεί), δεν γίνεται κάποια περαιτέρω διαδικασία.

  Στην αντίθετη περίπτωση, η τιμή που αντιστοιχεί σε αυτό, αφαιρείται από το Active[dpid] dictionary. Με χρήση του timestamp που περάστηκε ως παράμετρος, ανανεώνεται η τιμή στο timestamps. Επίσης, προκειμένου ίδια flows σε διαφορετικές στιγμές να είναι διαχωρισμένα στην δομή Expired, καλείται η συνάρτηση που χρησιμοποιούμε για τον κατακερματισμό, με 2 επιπλέον ορίσματα, ένα για το timestamp που πλέον συνυπολογίζεται ώστε να διαχωρίζονται χρονικά και άλλο ένα που λειτουργεί ως σημαία για την συνάρτηση (προσδιορίζει τον τρόπο λειτουργίας της)

- **collect_sflow(flow)**
  Δέχεται ένα σειριοποιημένο Dictionary για ένα sFlow sample. Αυτό μετατρέπεται και αποθηκεύεται κατάλληλα.

  Στη συνέχεια, ακολουθείται μια διαδικασία προκειμένου τα πεδία του sFlow sample, να αντιστοιχισθούν με τα αντίστοιχα του OpenFlow match (Πίνακας 4.2). συγκεκριμένα δίνοντας έμφαση στον τρόπο που διαχειρίζεται κάποια από αυτά ο POX (όπως το πεδίο vlan). Περαιτέρω επεξεργασία απαιτείται για την

ανακατασκευή του Ingress port (χρήση της δομής Mac_table), καθώς ενδέχεται να μην είναι ίσα μεταξύ τους (για παράδειγμα παρατηρείται κατά την χρήση logical bridges του OpenVswitch), λόγω του τρόπου λειτουργίας του sFlow agent στα switch.

OpenFlow	**sFlow**
in_port	inputPort
dl_src	SrcMAC
dl_dst	DstMAC
dl_vlan	invlan
dl_vlan_pcp	Από τις κεφαλίδες
dl_type	Από τις κεφαλίδες – ή μορφή csv
nw_tos	IPTOS
nw_proto	IPProtocol
nw_src	srcIP
nw_dst	dstIP
tp_src	TCP/UDPSrcPort - ICMPType
tp_dst	TCP/UDPDstPort - ICMPCode

Πίνακας 4.2 Πίνακας αντιστοίχησης πεδίων OpenFlow – sFlow

Έπειτα, εφαρμόζεται η συνάρτηση **construct_hashed_sflow.** Η συνάρτηση έχει παρόμοιο τρόπο λειτουργίας όπως η **construct_hashed_key** που χρησιμοποιήθηκε στις δυο προηγούμενες μεθόδους. Με βάση το αποτέλεσμα που επιστρέφει, αναζητείται εγγραφή με τέτοιο κλειδί στο dictionary Active[dpid]. Σε περίπτωση που βρεθεί τέτοια ανανεώνονται κατάλληλα οι μετρητές. Σημειώνεται πως το παραπάνω συνήθως είναι εφικτό όταν υπάρχει exact match. Αντίθετα, εάν η παραπάνω διαδικασία δεν καρποφορήσει, γίνεται προσπάθεια κατηγοριοποίησης του sFlow sample στους κανόνες που υπάρχουν στο Active[dpid].

Παρουσιάζεται ο αλγόριθμος αυτής της διαδικασίας. Αρχικά απαριθμούμε όλες τις εγγραφές για το δεδομένο dpid (Active[dpid]). Ξεκινάμε από την πρώτη (η διάταξη είναι τυχαία).

i. για την τρέχουσα εγγραφή απαριθμούμε , τα πεδία του dictionary match. Ξεκινάμε από το πρώτο (η διάταξη είναι επίσης τυχαία).

ii. Εάν δεν υπάρχει τιμή για το συγκεκριμένο πεδίο (**None**) τότε προχωράμε σε επόμενο πεδίο, καθώς θεωρείται wildcard και ταιριάζει με οτιδήποτε.
iii. Διαφορετικά, εάν η αντίστοιχη τιμή του sFlow δεν υπάρχει τότε προχωράμε στην επόμενη εγγραφή καθώς εδώ υπάρχει αναντιστοιχία ανάμεσα σε sFlow και πεδίο OpenFlow.
iv. Διαφορετικά, εάν τα πεδία του sFlow και το αντίστοιχο της τρέχουσας εγγραφής είναι ίσα, ομοίως με το ii συνεχίζουμε με το επόμενο πεδίο.
v. Διαφορετικά:
    a) Εάν πρόκειται για το πεδίο των διευθύνσεων IP, ελέγχουμε την περίπτωση όπου το πεδίο του OpenFlow αναφέρεται σε ένα υποδίκτυο διευθύνσεων και η διεύθυνση του sFlow sample ανήκει στο υποδίκτυο αυτό.
    b) Σε άλλη περίπτωση, υπάρχει εκ νέου αναντιστοιχία και επιλέγουμε την επόμενη Flow εγγραφή και συνεχίζουμε από το βήμα i.

Εάν εξαντληθούν όλα τα πεδία χωρίς να έχουμε προχωρήσει σε άλλη εγγραφή, η κατηγοριοποίηση ήταν επιτυχής και ανανεώνονται οι μετρητές της αντίστοιχης εγγραφής.
Εάν εξαντληθούν όλες οι εγγραφές χωρίς να έχουμε καταφέρει το παραπάνω η κατηγοριοποίηση ήταν ανεπιτυχής.

- **check_flowspace()**
  Η συνάρτηση δεν δέχεται κάποιο όρισμα. Σειριοποιεί τη λίστα flowspace και επιστρέφει την τιμή.

- **check2()**
  Η συνάρτηση δεν δέχεται κάποιο όρισμα. Δημιουργεί ένα tuple με τα active και expired dictionaries. Τα σειριοποιεί και επιστρέφει την τιμή.

- **update_flowspace(serial)**
  Η συνάρτηση δέχεται σαν όρισμα μια σειροποιημένη λίστα. Μετά την μετατροπή της σειριοποιημένης λίστας στην αρχική της μορφή, κάθε στοιχείο αναμένεται να είναι ένας κανόνας για το flow space του FlowVisor, σε μορφή dictionary. Τα στοιχεία αυτά ενσωματώνονται στην λίστα flowspace.

- **assign_flowspace(hash_val, dpid)**
  Η συνάρτηση δέχεται σαν όρισμα, μια hashed τιμή και το datapath id. Ως στόχο έχει την ένταξη μιας flow εγγραφής στον αντίστοιχο κανόνα του flow space και κατ επέκταση στο Slice, που ανήκει αυτός ο κανόνας.

  Αυτό το επιτυγχάνει χρησιμοποιώντας τη λίστα flowspace που έχει ανανεωθεί από την συνάρτηση update_flowspace. Απαριθμώντας λοιπόν τα στοιχεία της λίστας, για κάθε ένα από αυτά εφαρμόζεται ο ίδιος αλγόριθμός που περιγράφηκε στην

collect_sflow για την κατηγοριοποίηση του sFlow sample σε κάποιο flow entry. Μόνο που πλέον εννοιολογικά στην θέση του sFlow βρίσκεται το flow entry, και στη θέση του flow entry βρίσκεται ο κανόνας για το flowspace. Επίσης άλλες μικρές τροποποιήσεις αφορούν την εξέταση κανόνων μόνο για το συγκεκριμένο datapath id, καθώς και την περίπτωση που είτε οι διευθύνσεις του, είτε του flowspace είτε του flow entry είτε και των δυο δίνονται σε μορφή A.B.C.D/M (CIDR).

- **Construct_hashed_key(match, time_s=None, hash_f=1)**
  Η συνάρτηση σε κάθε περίπτωση αναμένει σαν πρώτο όρισμα ένα match αντικείμενο. Προαιρετικά ορίσματα είναι το timestamp, και το hash_f. Το timestamp όπως είδαμε το παρέχει σαν όρισμα μια κλήση της συνάρτησης **move_to_expired**, όπως επίσης και το flag hash_f το οποίο αλλάζει σε ποια πεδία θα εφαρμοστεί ο κατακερματισμός.

  Σε κάθε περίπτωση δημιουργείται ένα tuple, με τιμές των πεδίων του OpenFlow (εκτός από το VLAN_PCP εξαιτίας ενός προβλήματος σχετικά με τον τρόπο επεξεργασίας του συγκεκριμένου πεδίου από τον POX) καθώς την τιμή της μεταβλητής time_s (Η προεπιλεγμένη τιμή για την μεταβλητή time_s είναι **None** εκτός και αν οριστεί ρητά).

  Σε περίπτωση που η τιμή του hash_f είναι 1 εφαρμόζεται η βασική συνάρτηση κατακερματισμού της Python, **hash** μόνο στις τιμές των πεδίων του OpenFlow και επιστρέφεται η τιμή. Σε κάθε άλλη περίπτωση συνυπολογίζεται και η τιμή του time_s, και επιστρέφεται η τιμή. Προεπιλεγμένη τιμή είναι 1, συνεπώς η βασική συμπεριφορά είναι να μην υπολογίζεται η τιμή της μεταβλητής time_s.

- **Construct_hashed_sflow(match)**
  Η συνάρτηση αναμένει σαν όρισμα ένα dictionary που περιέχει εγγραφές με keys τα ονόματα των πεδίων του match, και values τις αντίστοιχες τιμές. Πανομοιότυπος τρόπος λειτουργίας με την **construct_hashed_key**.

- **Construct_dict(Match, dpid)**
  Η συνάρτηση αναμένει δυο ορίσματα, ένα match αντικείμενο, και το αντίστοιχο datapath id. Κατασκευάζεται ένα dictionary με βάση τα πεδία του αντικειμένου match (keys) και τις αντίστοιχες τιμές (values) καθώς και το αντίστοιχο dpid. Επιστρέφει το dictionary που κατασκευάστηκε.

Επίσης, στον κώδικα ορίζεται (1) σε ποια IP / port θα εκκινήσει ο server και (2) ποίες από τις παραπάνω συναρτήσεις γίνονται expose μέσω του API του RPC, με κλήση της μεθόδου register_function("function Name") της κλάσης SimpleJSONRPCServer. Στη συνέχεια ξεκινάει η λειτουργία του server.

Τα επόμενα αρχεία τρία αναφέρονται σε εξαρτήματα του POX controller, που υλοποιήθηκαν από την αρχή ή υπάρχουσες υλοποιήσεις που τροποποιήθηκαν.

4.5 Ανάλυση του αρχείου flowrem.py

Το συγκεκριμένο εξάρτημα αναπτύχθηκε από την αρχή, με κύριο στόχο την παροχή δυο συναρτήσεων ως handlers για τον χειρισμό δυο διαφορετικών τύπων Events.

- CustomEvent με τον αντίστοιχο handler "**_handle_CustomEvent**".
- FlowRemoved με τον αντίστοιχο handler "**_handle_FlowRemoved**".

Σημειώνεται πως ονομάζοντας την συνάρτηση **_handle_"EventName"** αυτόματα ορίζεται αυτή σαν χειριστής για γεγονότα με όνομα **"EventName"**. Όσον αφορά τους handlers, ο πρώτος στοχεύει στον χειρισμό γεγονότων CustomEvent (δημιουργήθηκε επίσης), το οποίο γίνεται raise από οποιοδήποτε εξάρτημα του controller επιθυμεί να εγκαταστήσει έναν κανόνα στο switch. Αντίστοιχα ο δεύτερος handler είναι επιφορτισμένος με το χειρισμό γεγονότων FlowRemoved, τα οποία γίνονται raise για κάθε flow που εκπνέει στο OpenFlow switch (σε περίπτωση που έχει οριστεί το κατάλληλο flag). Οι παραπάνω συναρτήσεις εντάσσονται ως μέθοδοι, σε μια κλάση που αποτελεί τον κορμό του εξαρτήματος.

Το CustomEvent που αναφέρθηκε προηγουμένως, υλοποιείται δημιουργώντας την ομώνυμη κλάση. Επίσης δημιουργείται μια κλάση EventSourcer μέσω της οποίας γίνεται raise το προηγούμενο Event. Τέλος δημιουργείται μια συνάρτηση (invoker) που απλά λειτουργεί ως wrapper. Σκοπός της είναι να κάνει raise το CustomEvent , μέσω της κλάσης EventSourcer που αναφέραμε πιο πάνω.

Επικεντρωνόμαστε εκ νέου στους handlers οι οποίοι έχουν πανομοιότυπη λειτουργία. Αρχικά, δημιουργούν ένα tuple, με την δομή που περιγράφεται στην **construct_new_entry** (OpenFlow Match αντικείμενο, DPID, και timestamp). Το timestamp δημιουργείται με χρήση της συνάρτησης time, της βιβλιοθήκης time (time.time(), Δευτερόλεπτα με σημείο αναφοράς το UNIX Epoch). Στη συνέχεια το tuple σειριοποιείται μέσω της βιβλιοθήκης Pickle, και με όρισμα αυτό πραγματοποιείται RPC στον server. Συγκεκριμένα, ο handler του CustomEvent πραγματοποιεί RPC στην μέθοδο **construct_new_entry**, ενώ ο handler για τα FlowRemoved στην μέθοδο **move_to_expired**.

4.6 Ανάλυση του αρχείου l2_learning.py

Το συγκεκριμένο εξάρτημα είναι ένα από τα βασικά του POX, και περιγράφηκε στην αντίστοιχη ενότητα. Το μεγαλύτερο μέρος της λειτουργικότητας του εξαρτήματος μένει ίδιο. Οι αλλαγές που έγιναν εντοπίζονται, (1) στην προσθήκη του κατάλληλου flag για τα

FlowRemoved (OFPFF_SEND_FLOW_REM), ώστε να γίνονται raise τα αντίστοιχα events κατά την εκπνοή των flows και (2) στην χρήση της συνάρτησης **invoker** με ορίσματα δυο αντικείμενα, ένα που χαρακτηρίζει την σύνδεση με το switch και ένα που περιλαμβάνει το μήνυμα που στέλνεται στο switch (εντός του οποίου περιέχεται και το match που ενδιαφέρει τον Handler του CustomEvent). Σημειώνεται πως δεν είναι απαραίτητη η επιλογή του συγκεκριμένου εξαρτήματος για την λειτουργία του Flow Repository. Οποιοδήποτε controller application εγκαθιστά κανόνες είναι αποδεκτό, αρκεί να προσθέσει τις αλλαγές που αναφέρονται προκειμένου να καταγράφονται αυτοί οι κανόνες στο Flow Repository.

4.7 Ανάλυση του αρχείου flow_stats.py

Το συγκεκριμένο εξάρτημα παρουσιάζει έναν ενδεικτικό τρόπου άντλησης στατιστικών στοιχείων από τα OpenFlow switches στον controller χρησιμοποιώντας τις δυνατότητες που προσφέρει το πρωτόκολλο εγγενώς. Στην αρχική του μορφή το εξάρτημα είχε ως στόχο την συλλογή στατιστικών που αφορούν web κίνηση αλλά τελικά έγιναν ορισμένες τροποποιήσεις με στόχο την κατηγοριοποίηση όλης της κίνησης με τρόπο που θα περιγραφεί στην συνέχεια. Η κίνηση εντάσσεται σε μια δομή dictionary, όμοιες με αυτές που έχουμε αναφέρει προηγουμένως (Ανάλυση του αρχείου pepserver_stable.py).

Συναρτήσεις που ορίζονται:

- **Construct_hashed_key(match)**
  Χρησιμοποιείται η ίδια συνάρτηση με αυτή που ορίζεται στο αρχείο **pepserver_stable.py** για την εφαρμογή συνάρτησης κατακερματισμού στα πεδία του αντικειμένου OpenFlow match.

- **_timer_func()**
  Η συγκεκριμένη συνάρτηση δεν δέχεται κάποιο όρισμα. Η λειτουργία της είναι να ενεργοποιεί μέσω χρονομέτρων μια άλλη συνάρτηση που στέλνει ανά διαστήματα σε κάθε συνδεδεμένο με τον controller switch, μήνυμα με το οποίο ζητούνται τα στατιστικά του.
- **_handle_flowstats_received(event)**
  Η συνάρτηση αποτελεί handler για γεγονότα τύπου **flowstats_received**. Δέχεται ένα όρισμα τύπου event και στη συνέχεια για κάθε διαφορετικό στοιχείο των στατιστικών εφαρμόζεται η συνάρτηση **construct_hashed_key**. Κατόπιν ελέγχεται εάν η τιμή αυτή υπάρχει ήδη σαν κλειδί στην δομή dictionary. Εάν δεν υπάρχει, δημιουργείται εκ νέου μια καταχώρηση με τιμή επίσης ένα dictionary για πληροφορίες όπως το συνολικό αριθμό bytes, πακέτων και τον αριθμό των φορών που έχει εντοπιστεί η κάθε εγγραφή (αρχικοποιημένη στο 1 την πρώτη φορά). Αντίθετα στην περίπτωση που υπάρχει τιμή, ανανεώνονται οι τρεις παραπάνω μετρητές.

- **_handle_portstats_received(event)**
  Η συνάρτηση αποτελεί handler για γεγονότα τύπου **portstats_received**. Δέχεται επίσης ένα όρισμα τύπου event. Σημειώνεται πως δεν τροποποιήθηκε η συγκεκριμένη συνάρτηση, καθώς κυρίως χρησιμοποιήθηκαν στατιστικά δεδομένα που αφορούν εγγραφές flows.

Τέλος η συνάρτηση που ενεργοποιεί το εξάρτημα εκτός από την εγκαθίδρυση listeners για την σωστή λειτουργία των handler, χρησιμοποιεί την συνάρτηση Timer που παρουσιάστηκε στην αντίστοιχη ενότητα για τον POX(υποκεφάλαιο 2.4.2.4). Μέσω της συνάρτησης Timer καλείται η συνάρτηση **_timer_func** ανά τακτά χρονικά διαστήματα.

4.8 Ανάλυση του αρχείου sflow.py

Στο συγκεκριμένο αρχείο, υλοποιείται η διαδικασία της συλλογής των sFlow samples και αποστολής αυτών στο Flow Repository. Κατά την εκκίνηση το πρόγραμμα περιμένει τουλάχιστον 1 όρισμα από το standard input (την πόρτα του επιπέδου μεταφοράς στην οποία θα λαμβάνονται τα sFlow samples). Προαιρετικά, δέχεται και το όνομα ενός αρχείου που περιλαμβάνει αντιστοιχίσεις μεταξύ του dpid ενός switch με το αντίστοιχο sourceId πεδίο του sFlow sample που στέλνει ο sFlow agent. Σημειώνεται πως χρησιμοποιείται ο collector ανοιχτού κώδικα sflowtool ([26]) υλοποιημένος σε γλώσσα C. Προκειμένου να είναι συμβατός με το περιβάλλον των υπολοίπων εφαρμογών, υλοποιήθηκε ένας "wrapper" μέσω της βιβλιοθήκης subprocess. Ακολουθούν οι διαφορετικές συναρτήσεις που υλοποιούνται:

- **call_and_peek_output**
  Δέχεται σαν όρισμα την εντολή προς εκτέλεση, αυτή θα δινόταν σε περιβάλλον φλοιού (Shell). Προαιρετικά δέχεται, μια Boolean τιμή για την δημιουργία ή όχι καινούργιου Shell (προεπιλεγμένη False). Χρησιμοποιείται η βιβλιοθήκη pseudo terminal και δημιουργεί ένα καινούργιο ζεύγος master, slave. Στη συνέχεια με χρήση του subprocess εκτελείται αυτή την εντολή, στην προκειμένη περίπτωση επικαλείται το πρόγραμμα sflowtool με stdout τον File Descriptor slave. Όσο η εντολή εκτελείται αυτόνομα, η συνάρτηση διαβάζει μέσω του άκρου master ότι γράφεται στο άκρο slave από το process του collector και το επιστρέφει ανά γραμμές (μέχρι να βρεθεί ο χαρακτήρας "\n").

- **separate_fields**
  Δέχεται σαν όρισμα μια συμβολοσειρά. Στην συμβολοσειρά αυτή αναζητείται ο χαρακτήρας του κενού. Με βάση το σημείο που βρίσκεται ο χαρακτήρας αυτός, η συμβολοσειρά διαχωρίζεται σε δυο επιμέρους, το πεδίο του sFlow sample και την αντίστοιχη τιμή (εφαρμόζεται στα sFlow fields όπου field και value διαχωρίζονται από ένα χαρακτήρα κενού).

- **sflowParser**
  Η συνάρτηση δεν δέχεται κάποιο όρισμα. Αρχικά δημιουργείται μια λίστα με τα πεδία του sFlow sample, που είναι επιθυμητό να ελέγχονται (συνάρτηση **separate_fields**). Χρησιμοποιείται επίσης μια μεταβλητή σαν flag (αρχικοποιημένη στο 0) για τον διαχωρισμό των διαφορετικών samples, καθώς ένα sFlow datagram μπορεί να περιλαμβάνει περισσότερα του ενός samples. Στη συνέχεια καλείται η συνάρτηση **call_and_peek_output** με παράμετρο το επιθυμητό port που δόθηκε κατά την εκτέλεση της εφαρμογής και ελέγχονται τα εξής για κάθε αυτόνομη συμβολοσειρά l που αναλύεται:
  - ✓ Εάν εντός της γραμμής βρίσκεται η συμβολοσειρά "FLOWSAMPLE", το flag τίθεται στην τιμή 1 (θα ακολουθήσει καινούργιο δείγμα).
  - ✓ Εάν το flag έχει τιμή 1, τότε εφαρμόζεται η συνάρτηση **separate_fields** με όρισμα την τρέχουσα γραμμή.
  - ✓ Εάν εντός της γραμμής βρίσκεται η συμβολοσειρά "endSample" και η τιμή του flag είναι 1, το flag γίνεται 0 (τέλος δείγματος). Στην συνέχεια ακολουθεί μια διαδικασία ανακατασκευής κάποιων πεδίων που είναι απαραίτητα για την αντιστοίχιση του sample με τα flow entries (Ethernet Type για όλα τα πακέτα και IP addresses/ARP Opcode όσον αφορά τα ARP). Η διαδικασία χρησιμοποιεί τις κεφαλίδες που περιλαμβάνονται στο sample. Επίσης προστίθεται το dpid με βάση την αντιστοίχιση που δόθηκε στο αρχείο. Τέλος, τα sFlow samples στην τελική τους μορφή σειριοποιούνται και αποστέλλονται στο Flow Repository.

4.9 Ανάλυση του αρχείου get_flowspace.py

Κύριος στόχος της εφαρμογής είναι η αποστολή στο Flow Repository, των κανόνων του flowspace με τους οποίους έχει ρυθμιστεί ο FlowVisor. Αυτή η λειτουργικότητα υλοποιείται με την βοήθεια των εξής συναρτήσεων:

- **call_and_peek_output**
  Χρησιμοποιείται η ίδια συνάρτηση που ορίζεται και στο **sflow.py**. Στόχος είναι η σάρωση των αποτελεσμάτων που προκύπτουν από την εκτέλεση μιας εντολής σε περιβάλλον φλοιού (θυμίζουμε πως η συνάρτηση δέχεται την εντολή αυτή ως παράμετρο, με την μορφή συμβολοσειράς)

- **flowvisor_parser**
  Καλείται από το κυρίως σώμα (main), της εφαρμογής. Στη συνέχεια καλείται η **call_and_peek_output**, και για κάθε γραμμή που επιστρέφει γίνεται προσπάθεια κατασκευής dictionary (λόγω της δομής key – value που έχουν οι κανόνες του flowspace). Σημειώνεται πως κάθε γραμμή αντιστοιχεί σε ένα κανόνα του flowspace. Κάθε dictionary τοποθετείται (γίνεται append) σε κατάλληλη λίστα.

Τέλος η λίστα σειριοποιείται και με όρισμα αυτή τη τιμή καλείται η μέθοδος **update_flowspace** που ορίζεται στο **pepserver_stable.py**.

4.10 Ανάλυση του αρχείου client.py

Στο συγκεκριμένο αρχείο έχει υλοποιηθεί μια εφαρμογή που δίνει τη δυνατότητα επικοινωνίας με το Flow Repository, με στόχο την προσπέλαση των στατιστικών που περιέχονται για κάθε flow. Η εκτέλεση της εφαρμογής δεν αναμένει κάποιο όρισμα. Κατά την εκτέλεση δίνει τη δυνατότητα στο χρήστη να επιλέξει, μέσω αριθμημένων επιλογών, διαφορετικές λειτουργίες. Αυτές είναι:

- **Active:** Εκτύπωση όλων των εγγραφών που είναι ενεργές.
- **Expired:** Εκτύπωση όλων των εγγραφών που έχουν εκπνεύσει.
- **Aggregates:** Εκτύπωση όλων των εγγραφών ανεξαρτήτως της κατάστασης στην οποία βρίσκονται, αθροίζοντας πρώτα όσες είναι ίδιες αλλά εντοπίζονται σε διαφορετικές χρονικές στιγμές.
- **Select timestamp**: Εκτύπωση όλων των εγγραφών που είναι ενεργές/έχουν εκπνεύσει και χρονικά εντάσσονται σε ένα χρονικό πλαίσιο.
- **Exit**: Έξοδος από το τρέχον πρόγραμμα.

Τα εκάστοτε αποτελέσματα δεν εκτυπώνονται στην οθόνη αλλά γράφονται σε κάποιο αρχείο, για ευκολία στην προσπέλαση και στον μετέπειτα χειρισμό.

Οι παραπάνω επιλογές είναι διαθέσιμες μέσω κάποιων συναρτήσεων οι οποίες περιγράφονται στη συνέχεια:

- **construct_hashed_flow**
  Πανομοιότυπη συνάρτηση με την συνάρτηση που χρησιμοποιείται για τα sflow samples και ορίζεται στο αρχείο **pepserver_stable.py**. Εφαρμόζει συνάρτηση κατακερματισμού στα πεδία του κάθε flow entry το οποίο δέχεται σαν όρισμα με την μορφή dictionary.

- **grab**
  Δεν δέχεται κάποια παράμετρο σαν όρισμα, εκτελεί ένα RPC (συγκεκριμένα καλεί την απομακρυσμένη μέθοδο **check2()**) και την τιμή που επιστρέφεται από αυτό την αποσειριοποιεί και την επιστρέφει με την σειρά της (πρόκειται για την δομή tuple στην οποία περιέχονται οι δομές Active, Expired).

- **flowt**
  Δέχεται σε κάθε περίπτωση μια παράμετρο σαν όρισμα η οποία προσδιορίζει εάν θα προσπελασθούν τα Active ή Expired flows. Προαιρετικά δέχεται ακόμα δυο παραμέτρους που προσδιορίζουν ένα χρονικό πλαίσιο. Οι προεπιλεγμένες τιμές είναι τέτοιες ώστε εάν δεν δοθεί κάποια παράμετρος, να μένει το άκρο του

αντίστοιχου περιορισμού ελεύθερο. Εκτελείται μια κλήση της συνάρτησης **grab** και η δομή που επιστρέφεται από αυτήν προσπελαύνεται. Τα δεδομένα της δομής που εμπίπτουν στο χρονικό πλαίσιο που ορίζεται (προεπιλεγμένο ή όχι) γράφονται με μια συγκεκριμένη δομή (csv) σε κατάλληλο αρχείο.

- **active**
  Δεν δέχεται κάποιο όρισμα, δουλεύει σαν wrapper για την κλήση της συνάρτησης flowt, με όρισμα 0 (ώστε να προσπελασθεί η δομή με τα Active flows).

- **expired**
  Δεν δέχεται κάποιο όρισμα, δουλεύει σαν wrapper για την κλήση της συνάρτησης flowt, με όρισμα 1 (ώστε να προσπελασθεί η δομή με τα Expired flows).

- **aggregate**
  Δεν δέχεται κάποιο όρισμα. Αρχικά εκτελείται η συνάρτηση **grab** ώστε να έχουμε ένα αντίγραφό των δομών Active/Expired. Έχοντας ως βάση τα περιεχόμενα της δομής Active, προσπελαύνεται ολόκληρη η δομή Expired. Σε κάθε εγγραφή εφαρμόζεται η συνάρτηση **construct_hashed_flow** με όρισμα το "match" της (πεδία του OpenFlow match) και προκύπτει μια τιμή για αυτή την εγγραφή.
  Στη συνέχεια συναθροίζονται οι εγγραφές, δηλαδή εάν η αυτή τιμή υπάρχει στην δομή Active προσθέτουμε τις τιμές των μετρητών της συγκεκριμένης εγγραφής στην ήδη υπάρχουσα και επιπρόσθετα αυξάνουμε και έναν μετρητή που κρατάει πόσες εγγραφές έχουν αθροιστεί συνολικά. Σε διαφορετική περίπτωση, αφαιρούμε την εγγραφή από την δομή Expired και την τοποθετούμε αυτούσια, στην δομή Active. Τέλος τα αθροισμένα στοιχεία γράφονται σε κατάλληλο αρχείο (δομή csv).

- **timewindows**
  Η συνάρτηση δεν δέχεται κάποιο όρισμα. Αρχικά τυπώνει την τρέχουσα ημερομηνία και ώρα(timestamp). Έπειτα αναμένει σαν είσοδο από το χρήστη, μια τιμή που προσδιορίζει εάν θέλουμε Active/Expired εγγραφές και δυο timestamps για το χρονικό πλαίσιο, ένα για το κάτω και ένα για το άνω άκρο. Το timestamp πρέπει να δίνεται σε μια συγκεκριμένη μορφή. Στη συνέχεια καλείται η συνάρτηση **flowt** με ορίσματα τις τιμές που εισήγαγε ο χρήστης. Σημειώνεται πως εάν έχει δοθεί το κενό σαν timestamp θεωρείται πως το αντίστοιχο άκρο δεν πρέπει να έχει περιορισμό (εάν και για τα δυο άκρα δοθεί σαν τιμή το κενό η κλήση, ισοδυναμεί με αυτή των συναρτήσεων **active**/**expired**).

5

Αξιολόγηση Υλοποίησης

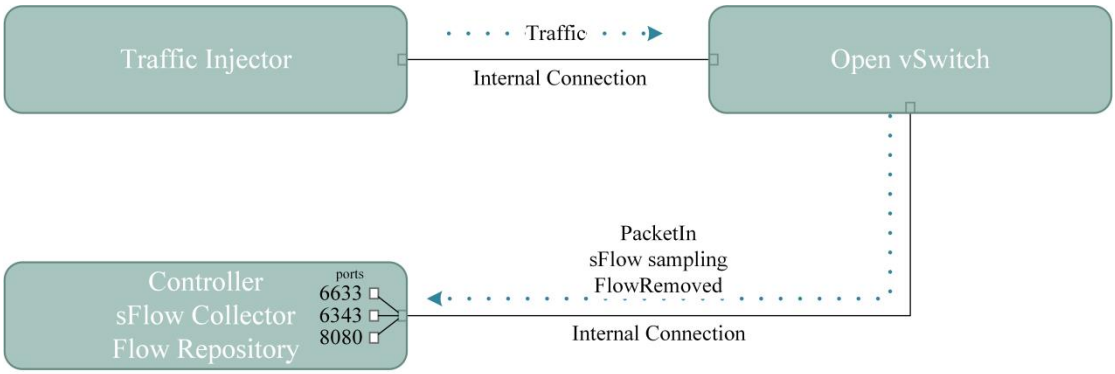
Στο κεφάλαιο αυτό, αρχικά αναλύεται η πειραματική διαδικασία που ακολουθήθηκε. Επίσης, παρουσιάζονται τα δεδομένα τα οποία εξάγονται από την άνω διαδικασία και στην συνέχεια αξιολογούνται. Τέλος, προτείνονται τροποποιήσεις ώστε ο μηχανισμός να είναι συμβατός με διάφορες ετερογενείς υποδομές.

5.1 Πειραματική διάταξη

Εκτός από τα αρχεία κώδικα της υλοποίησης που περιγράφονται εκτενώς στο κεφάλαιο 4, κατά την πειραματική διαδικασία χρησιμοποιούνται τα ακόλουθα προγράμματα:

- Top [27]
- Tcpreplay [28]
- OpenVswitch
- FlowVisor

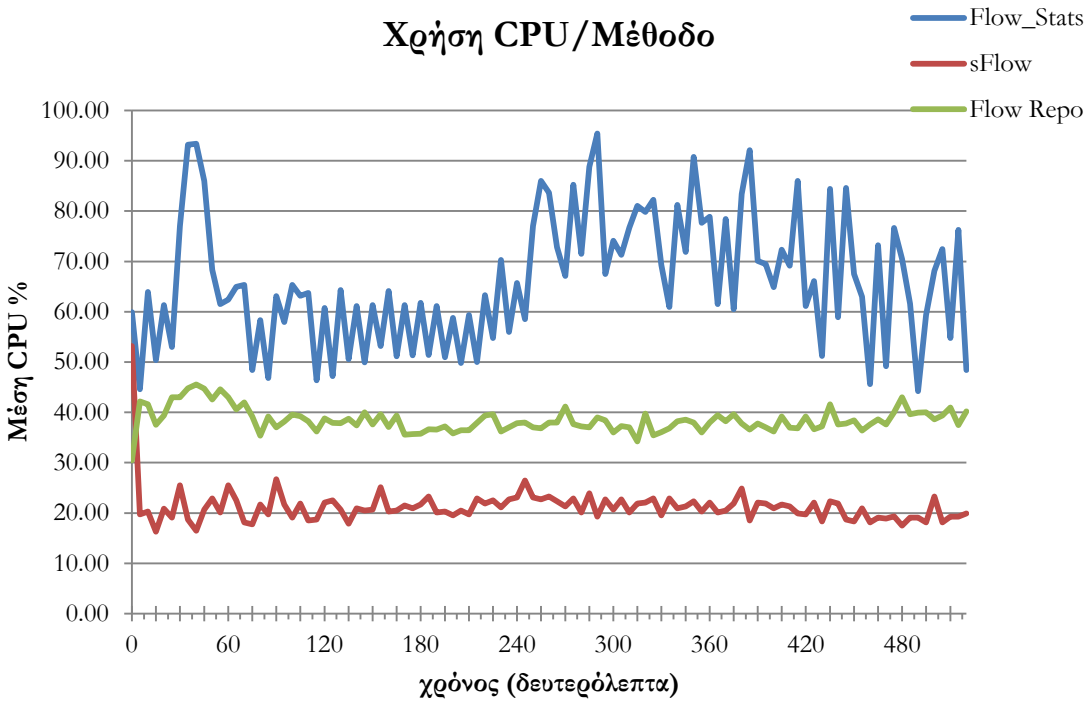
Όσον αφορά την τοπολογία, χρησιμοποιήθηκαν τρεις διαφορετικές εικονικές μηχανές (λειτουργικό σύστημα Ubuntu 14.04 [29]) για τις ανάγκες των πειραμάτων. Ακολουθεί η διάταξη (Σχήμα 5.1 Πειραματική Διάταξη).



Σχήμα 5.1 Πειραματική Διάταξη

- **host**
  Σκοπός αυτού του μηχανήματος είναι η αναπαραγωγής κίνησης (αποθηκευμένη σε κατάλληλο αρχείο pcap με χρήση του tcpdump [30]) προς το **openvswitch** μέσω του εργαλείου tcpreplay.
- **openvswitch**
  Σε αυτό το μηχάνημα λειτουργεί το **openvswitch**, μέσα σε αυτό δημιουργείται ένα logical bridge στο οποίο προστίθεται η κατάλληλη διεπαφή. Το bridge αυτό συνδέεται με τον OpenFlow controller και επίσης ρυθμίζεται ο sFlow agent για την αποστολή δειγμάτων.
- **FlowRepository**
  Σε αυτό το μηχάνημα βρίσκεται το μεγαλύτερο μέρος της υλοποίησης. Πιο συγκεκριμένα ο OpenFlow controller, το Flow Repository και ο sFlow collector. Επίσης στο συγκεκριμένο μηχάνημα εκτελείται η εφαρμογή client, και εξάγονται τα δεδομένα που καταγράφονται από το Flow Repository.

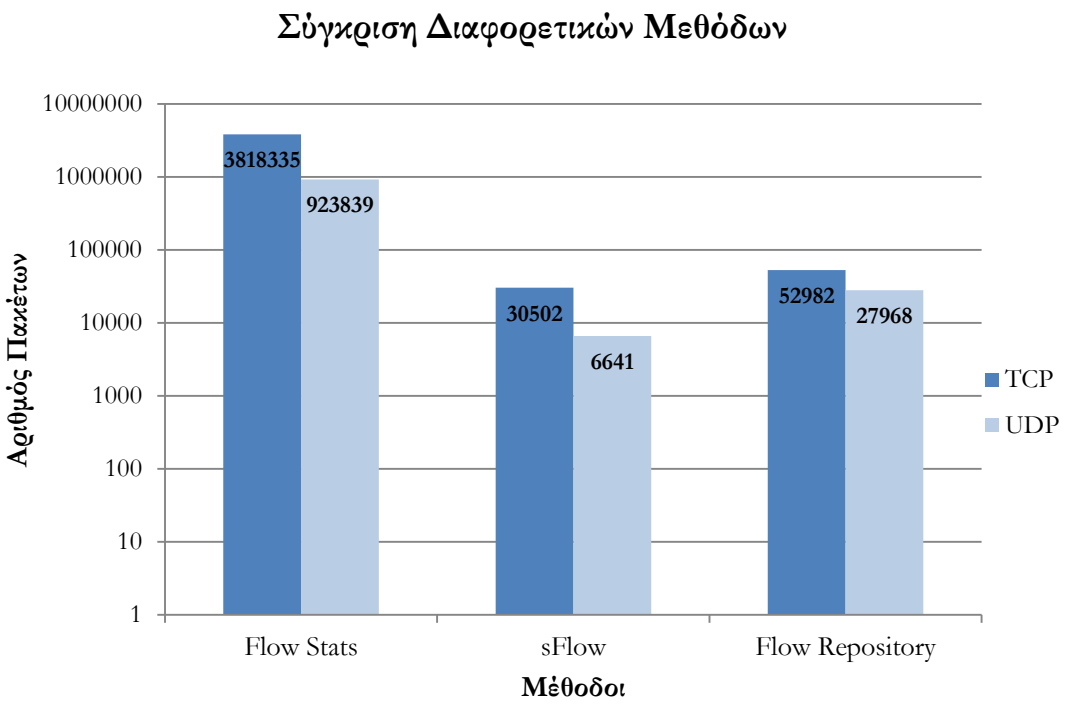
Αρχικά εξετάζεται το υπολογιστικό κόστος που επιφέρει η προτεινόμενη υλοποίηση στον controller, μετρώντας την χρησιμοποίηση του επεξεργαστή από την αντίστοιχη εφαρμογή, σε σύγκριση με την μέθοδο Flow Stats και χρησιμοποιώντας αμιγώς το πρωτόκολλο sFlow. Οι μετρήσεις λαμβάνονται με χρήση του προγράμματος "top", για τις διαφορετικές μεθόδους συγκέντρωσης στατιστικών στοιχείων και παρατίθενται στο ακόλουθο διάγραμμα (Σχήμα 5.2).



Σχήμα 5.2 Διάγραμμα υπολογιστικού κόστους για τις μεθόδους Flow Stats, sFlow, Flow Repository

Παρατηρούμε πως όπως ήταν αναμενόμενο, η συχνή πραγματοποίηση αιτημάτων Flow Stats προς το OpenFlow switch δημιουργεί πράγματι μεγάλο φόρτο στον Controller. Σημειώνεται πως αυτό αναμένεται να κλιμακωθεί περαιτέρω για περισσότερα του ενός OpenFlow swiches, αυξημένη ροή κίνησης ή πολυπλοκότερη επεξεργασία των stats reply μηνυμάτων.

Στη συνέχεια συγκρίνεται η ικανότητα των διαφορετικών μεθόδων, με βάση την ακρίβεια κάθε μιας. Η κίνηση κατηγοριοποιείται σε TCP/UDP (κάτι το οποίο αποτελεί τον μεγαλύτερο όγκο της κίνησης, σε ποσοστό 99.6%) και στην συνέχεια αντιπαραβάλλεται για τις διάφορες μεθόδους (Σχήμα 5.3).

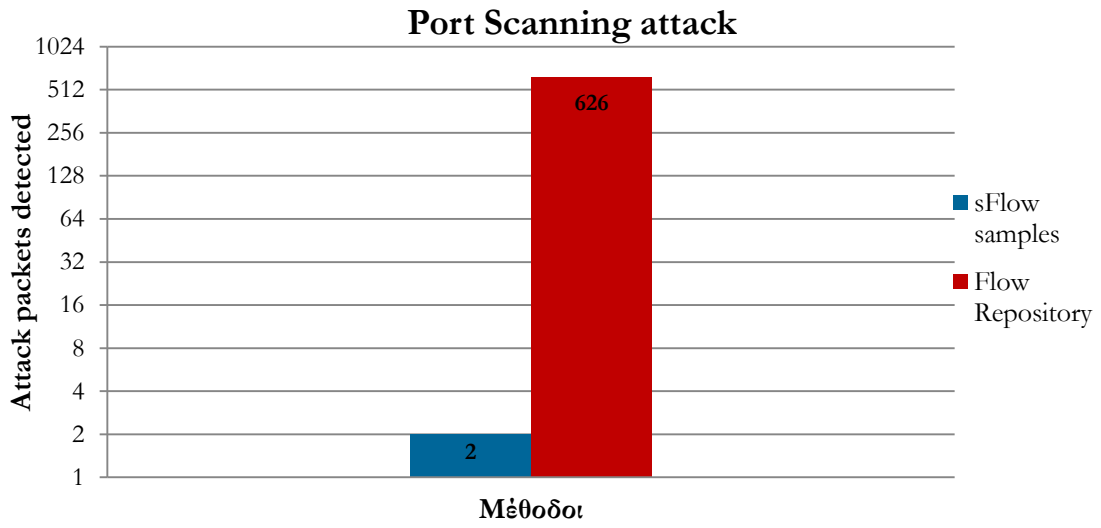


Σχήμα 5.3 Διάγραμμα ακρίβειας των 3 μεθόδων, κατηγοριοποίηση της κίνησης σε TCP / UDP

Παρατηρούμε πως ο αριθμός των πακέτων που εντοπίζει η μέθοδος Flow Stats είναι σημαντικά μεγαλύτερος από τις αντίστοιχες μεθόδους. Αυτό συμβαίνει εξαιτίας του ρυθμού δειγματοληψίας (1/128) περίπου σε αυτό τον λόγο εντοπίζεται και η διαφορά Flow Stats και sFlow. Επίσης φαίνεται πως το Flow Repository ανιχνεύει επιπλέον πακέτα σε σχέση με την απλή χρήση του sFlow (λόγω της χρήσης σηματοδοσίας).

Σκοπός του επόμενου πειράματος είναι η περαιτέρω ανάδειξη των πλεονεκτημάτων του Flow Repository έναντι της απλής χρήσης του sFlow sampling. Ακολούθως, πραγματοποιείται αυτόνομα σάρωση ενός /24 δικτύου για open ports (port scanning attack), με χρήση του εργαλείου nmap [31]. Το σενάριο είναι ιδανικό για τον σκοπό του πειράματος, καθώς πρόκειται για ένα είδος επίθεσης που δημιουργεί "mice" flows. Η σάρωση αυτή καταγράφεται σε ένα αρχείο pcap, και ενσωματώνεται στο βασικό αρχείο που

χρησιμοποιείται γενικότερα στα πλαίσια των μετρήσεων, μέσω του προγράμματος mergecap [32] . Κατά την διάρκεια της σάρωσης στάλθηκαν 626 διαφορετικά TCP SYN μηνύματα. Στο ακόλουθο διάγραμμα (Σχήμα 5.4) παρουσιάζεται το πλήθος των μηνυμάτων της επίθεσης που ανιχνεύει η κάθε μέθοδος.



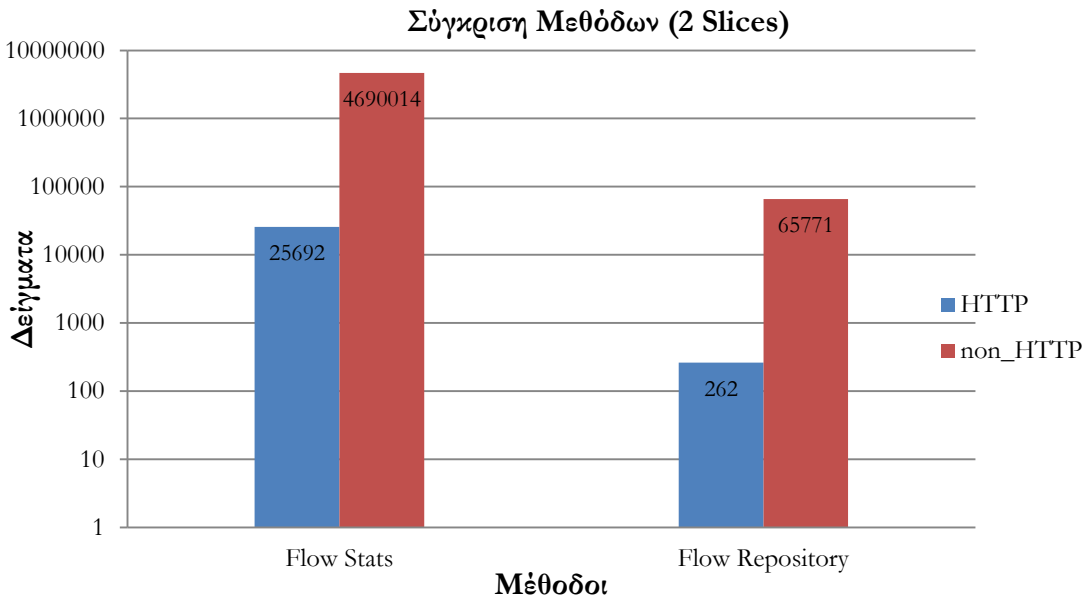
Σχήμα 5.4 Ενδεικτικό παράδειγμα χρήσης για ανίχνευση επίθεσης Port Scan

Όπως παρατηρείται, το sFlow sampling, ανιχνεύει ένα πολύ μικρό ποσοστό της επίθεσης, μόλις 2 μηνύματα, ενώ αντίθετα το Flow Repository καταφέρνει να εντοπίσει επιτυχώς ολόκληρη την ροή των πακέτων που σχετίζονται με την επίθεση, χρησιμοποιώντας τα μηνύματα του OpenFlow.

Στη συνέχεια, αξιολογούμε την προτεινόμενη υλοποίηση για περιβάλλοντα πολλαπλών χρηστών, χρησιμοποιώντας την πλατφόρμα FlowVisor. Όπως έχει αναφερθεί ο FlowVisor, λειτουργεί σαν proxy controller, και διαχωρίζει την κίνηση προθέτοντας κανόνες σε δυο διαφορετικά κομμάτια (“slices”), ώστε να διαχειρίζεται από δυο πανομοιότυπους controllers. Η κίνηση χωρίζεται σε:

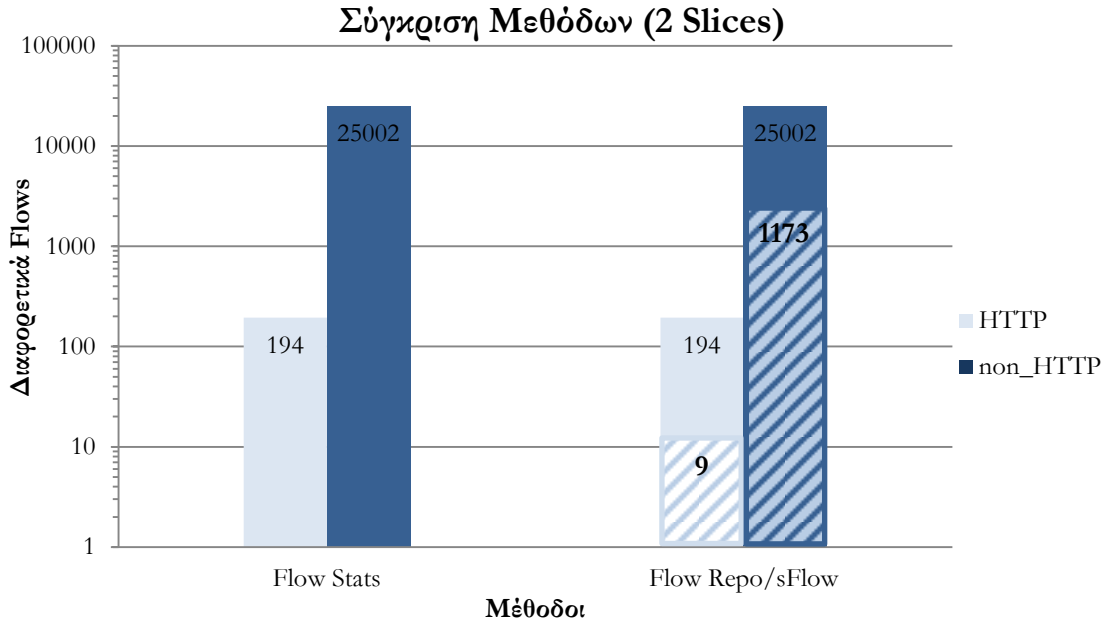
- **HTTP**
- **non HTTP**

Ακολουθεί το διάγραμμα για τον αριθμό των πακέτων που έχουν ανιχνευθεί εκτελώντας Flow Stats Request και χρησιμοποιώντας το Flow Repository (Σχήμα 5.5).



Σχήμα 5.5 Sliced Τοπολογία με χρήση FlowVisor, HTTP και non – HTTP κίνηση

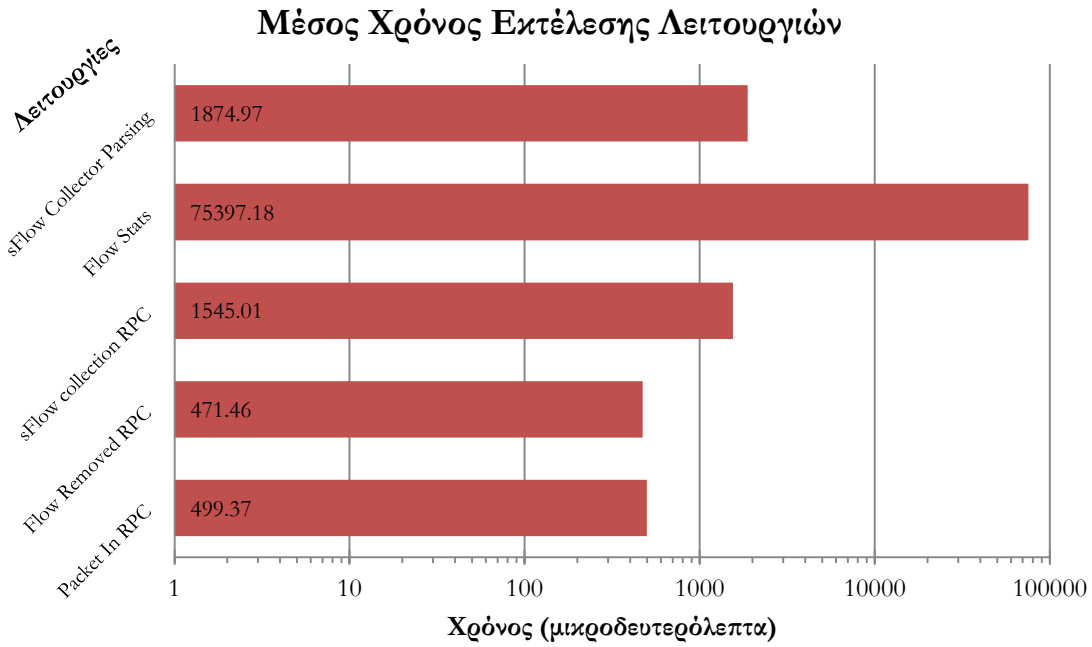
Στο επόμενο διάγραμμα (Σχήμα 5.6), παρατίθεται επίσης το πλήθος των διαφορετικών flows που έχει την δυνατότητα να αντιληφθεί κάθε μέθοδος, για τα διαφορετικά Slices. Επισημαίνεται πως στο διάγραμμα, εντός των στηλών που αφορούν την μέθοδο Flow Repository φαίνεται με πλάγιες γραμμές το πλήθος των διαφορετικών flows που θα εντόπιζε η μέθοδος sFlow από μόνη της χωρίς την χρήση των μηνυμάτων του Control Plane.



Σχήμα 5.6 Sliced Τοπολογία με χρήση FlowVisor, HTTP και non – HTTP κίνηση, σύγκριση της προτεινόμενης μεθόδου με το sFlow

Παρατηρούμε ξανά πως, η υλοποίηση του Flow Repository αναβαθμίζει σημαντικά τις δυνατότητες της δειγματοληψίας με sFlow επιτυγχάνοντας την ανίχνευσης των "mice" flows, ανιχνεύοντας όσα flows θα καταγράφονταν και κατά την άντληση των δεδομένων απευθείας από το switch.

Τέλος, πραγματοποιήθηκαν μετρήσεις όσον αφορά την χρονική διάρκεια που χρειάζεται για να ολοκληρωθούν οι διάφορες διαδικασίες που επιτελεί το Flow Repository, καθώς και η αντίστοιχη διαδικασία για την επεξεργασία δεδομένων μέσω Flow Stats request (Σχήμα 5.7).



Σχήμα 5.7 Χρόνος εκτέλεσης των διάφορων διεργασιών που επιτελούνται

Είναι εμφανής ακόμα μια φορά η διαφορά στο κόστος της επεξεργασίας στατιστικών μέσω Flow Stats, σε σύγκριση με τις προτεινόμενες μεθόδους. Επίσης παρατηρούμε πως από τις διαδικασίες του Flow Repository η συλλογή των στατιστικών και η προσπάθεια ένταξης τους στα ενεργά flows είναι οι πλέον δαπανηρές, κάτι το οποίο ήταν αναμενόμενο με βάση την ανάλυση της υλοποίησης (Κεφάλαιο 4).

5.2 Προσαρμογή σε διαφορετικά περιβάλλοντα

Κατά την παρουσίαση της αρχιτεκτονικής στο Σχεδιαστικές Αρχές, είχε αναφερθεί πως η προτεινόμενη παραδοχή (προϋπόθεση SDN υποδομής) αποτελεί μια συγκεκριμένη περίπτωση από τις διαφορετικές που συναντάμε στις σύγχρονες υποδομές. Στο συγκεκριμένο υποκεφάλαιο θα παρουσιαστούν διάφοροι μηχανισμοί ώστε η προτεινόμενη υλοποίηση να αποτελεί βιώσιμη λύση για Cloud και ασύρματες εγκαταστάσεις, προσφέροντας όλες τις δυνατότητες που έχουν παρατεθεί τόσο σε θεωρητικό επίπεδο όσο και μέσω της πειραματικής διαδικασίας, χωρίς να επηρεάζεται από επιμέρους διαφορές.

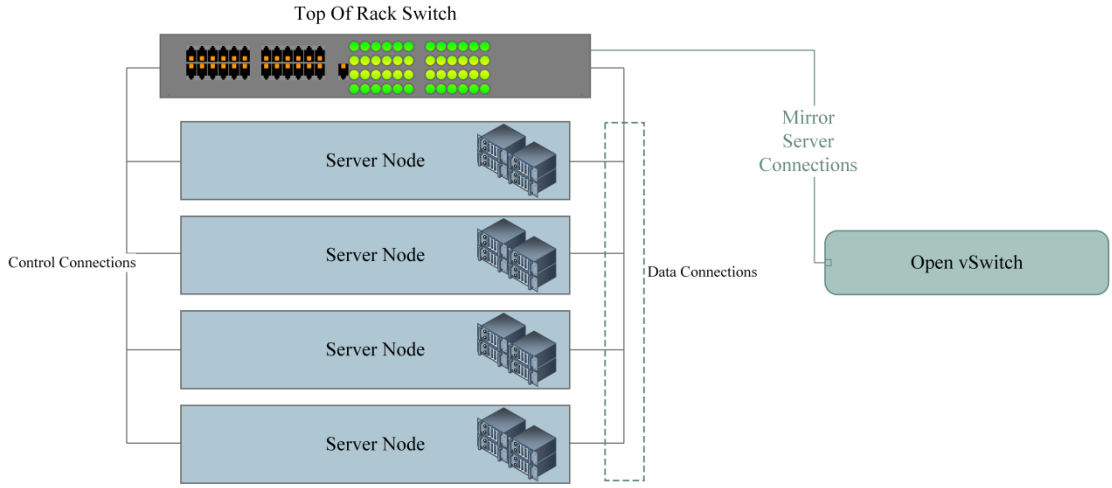
5.2.1 *Cloud Facility*

Η κυρίαρχη αρχιτεκτονική δομή που παρατηρείται σε υποδομές που δεν χρησιμοποιούν Δικτύωση καθοριζόμενη από λογισμικό αποτελείται από διάφορους κόμβους συνδεδεμένους με ένα μεταγωγέα. Ειδικότερα όσον αφορά την πλειοψηφία των σύγχρονων υποδομών οι κόμβοι αυτοί είναι οργανωμένοι σε συστοιχίες (racks), όπου κάθε συστοιχία έχει και ένα κεντρικό switch (Top Of Rack switch – TOR Switch), καθώς και άλλες δευτερεύουσες συσκευές δικτύωσης. Μια τέτοια υποδομή αποτελεί τυπική περίπτωση ενός Cloud Facility.

Προκειμένου λοιπόν να ενσωματωθεί η διαδικασία ανίχνευσης των "mice" flows, προτείνεται το mirror (Σχήμα 5.8) της κίνησης του κάθε server. Αυτή η κίνηση κατευθύνεται σε μια διάταξη που λειτουργεί αυτόνομα και έχει ως μοναδικό στόχο την δημιουργία flows με ένα προκαθορισμένο, χωρίς ιδιαίτερη σημασία action (π.χ. εγκαθίδρυση drop rules) απλά και μόνο για την καταγραφή των διαφορετικών flows που εγκαθίστανται. Η διάταξη απαρτίζεται από ένα Open vSwitch, έναν OpenFlow controller και το Flow Repository (Σχήμα 5.9).

Η παραπάνω διάταξη μας επιτρέπει να χρησιμοποιήσουμε τις δυνατότητες του πρωτοκόλλου OpenFlow χωρίς να αλλάζει ο προκαθορισμένος τρόπος δικτύωσης, ούτε να επιβαρύνονται οι τρέχουσες διεργασίες της εκάστοτε συστοιχίας. Αρχικά, δημιουργείται ένα logical bridge στο Open vSwitch, σε αυτό το bridge συνδέεται μέσω μιας δευτερεύουσας διεπαφής ο OpenFlow controller. Τέλος ενεργοποιείται ο sFlow agent στο logical bridge.

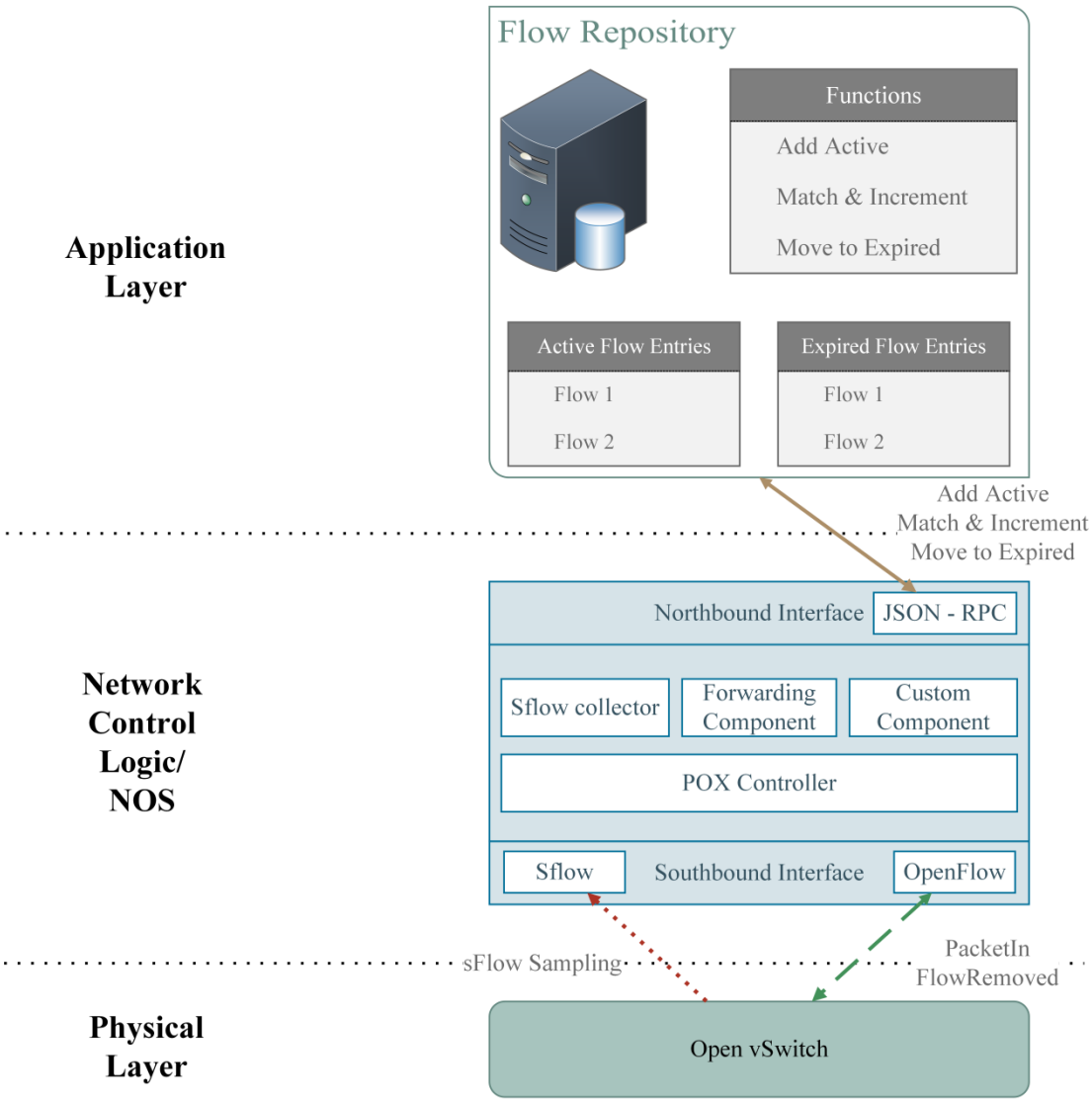
Στη συνέχεια το σύστημα που απαρτίζουν bridge, controller, sFlow agent, Flow Repository, λειτουργεί με τον τρόπο που έχουμε αναλύσει, καταγράφοντας την εγκατάσταση όλων των κανόνων, πραγματοποιώντας δειγματοληψία της κίνησης του bridge και αποστολή των δειγμάτων στο Flow Repository.



Σχήμα 5.8 Ενδεικτική διάταξη υποδομής Cloud. Προσθήκη Open vSwitch

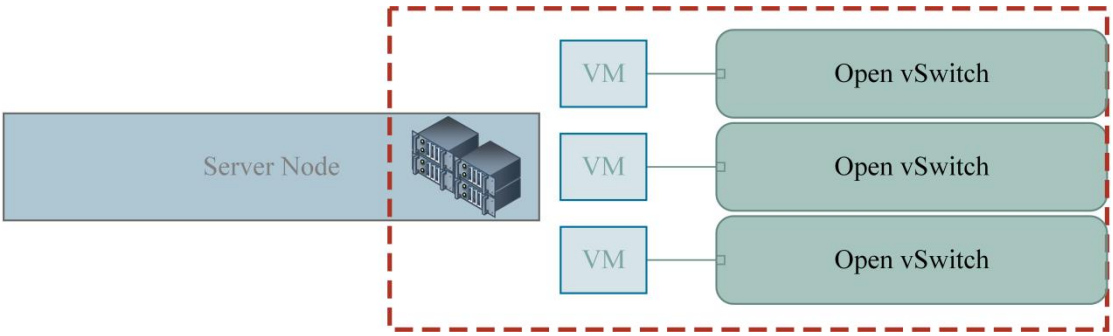
Σημειώνεται πως σε τέτοιες υποδομές λόγω του πυκνού όγκου κίνησης είναι ιδιαίτερα συνηθισμένη η χρήση κάποιας μεθόδου δειγματοληψίας, με στόχο την παρακολούθηση της υποδομής και έτσι θα μπορούσε να χρησιμοποιηθεί εναλλακτικά ο sFlow agent του TOR switch. Σημειώνεται πως σε περίπτωση που το καθρέφτισμα της κίνησης αξιολογείται ως μια δαπανηρή διεργασία για το switch που το επιτελεί, μπορεί να γίνει χρήση Network Taps

(Layer 1 συσκευές που χρησιμοποιούν την καλωδίωση για να καθρεφτίζουν και να παρακολουθούν την κίνηση χωρίς την δαπάνη υπολογιστικών πόρων από τα switches).



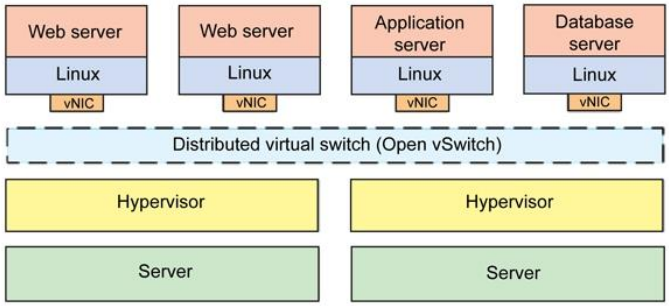
Σχήμα 5.9 Προσαρμογή της Αρχιτεκτονικής για Cloud υποδομές

Ένα μειονέκτημα που παρουσιάζει αυτή η προσαρμογή είναι το mirroring αγνοεί την κίνηση στο εσωτερικό του κάθε server, καθώς είναι ιδιαίτερα συνηθισμένο μέσα σε κάθε server να συνυπάρχουν πολλές εικονικές μηχανές. Σε περίπτωση που θέλουμε να μην έχουμε τέτοιες απώλειες μπορούμε εναλλακτικά να τοποθετήσουμε το Open vSwitch εντός της κάθε εικονικής μηχανής, αλλά διατηρώντας controller και Flow Repository εξωτερικά (Σχήμα 5.10).



Σχήμα 5.10 Εναλλακτική προσέγγιση με τοποθέτηση του Open vSwitch σε κάθε VM ή Server.

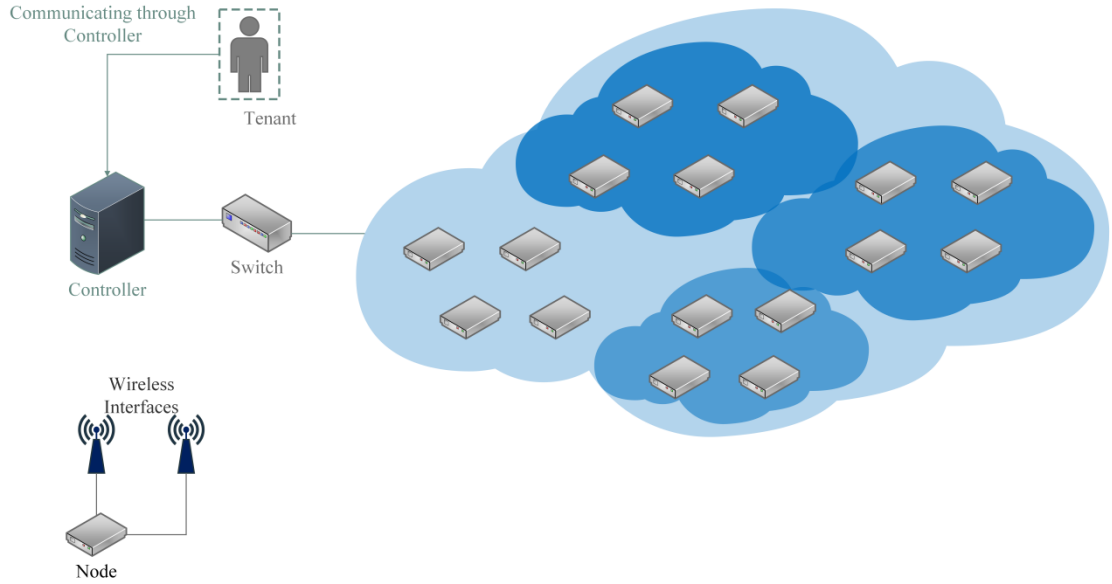
Επισημαίνεται επίσης πως, το Open vSwitch αποτελεί τον κύριο τρόπο δικτύωσης σε κάποιους Hypervisors (Xen, KVM). Εκμεταλλευόμενοι αυτό το γεγονός μπορούμε να δημιουργήσουμε στο ήδη υπάρχον Open vSwitch κατάλληλα logical bridges, τα οποία θα είναι συνδεδεμένα με τον OpenFlow Controller και θα στέλνουν δείγματα μέσω του sFlow agent, προκειμένου να εξασφαλισθεί η λειτουργία του μηχανισμού (Σχήμα 5.11), χωρίς να επεμβαίνουμε στις εικονικές μηχανές.



Σχήμα 5.11 Open vSwitch και Hypervisor

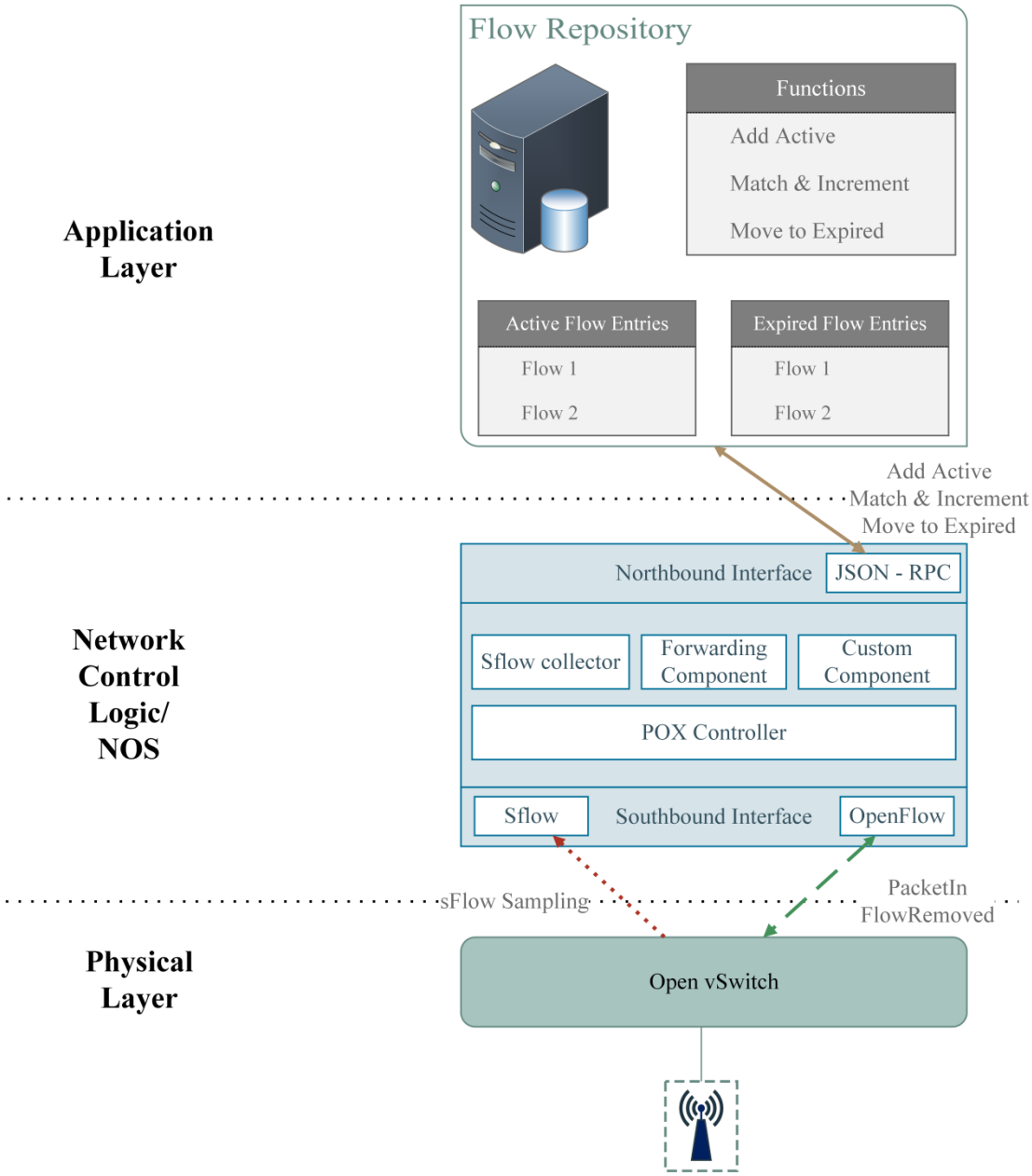
5.2.2 *Wireless Facility*

Μέχρι στιγμής έχουμε αναφερθεί μόνο σε διατάξεις βασισμένες σε ενσύρματη δικτύωση. Με στόχο λοιπόν να δίνεται η δυνατότητα παρακολούθησης της κίνησης σε περιβάλλοντα όπου οι κόμβοι επικοινωνούν ασύρματα (Σχήμα 5.12), προτείνεται μια προσαρμογή της αρχιτεκτονικής παρόμοια με αυτής που περιγράφηκε στην ενότητα 5.2.1 για τις εγκαταστάσεις υπολογιστικών νεφών.



Σχήμα 5.12 Παράδειγμα αρχιτεκτονικής Wireless Facility

Προϋπόθεση για την ορθή λειτουργία της προτεινόμενης προσαρμογής, είναι η συμβατότητα των κόμβων που συμμετέχουν στην ασύρματη τοπολογία με το Open vSwitch. Κάτι τέτοιο δεν αποτελεί ιδιαίτερο περιορισμό καθώς όπως έχει αναφερθεί στο κεφάλαιο 2, το Open vSwitch μπορεί να εγκατασταθεί και να λειτουργήσει επιτυχώς σε μια ευρεία γκάμα διαφορετικών συστημάτων. Σε σύγκριση με την αρχιτεκτονική που παρουσιάστηκε προηγουμένως, δεν υπάρχει η δυνατότητα να γίνει mirror της δικτυακής κίνησης. Έτσι, χρησιμοποιείται το Open vSwitch για αυτή την διαδικασία, όπως προτάθηκε και στο εναλλακτικό σενάριο εντός του κάθε VM. Στην προκειμένη περίπτωση, δημιουργείται σε κάθε κόμβο ένα logical bridge, στο οποίο προστίθεται η ασύρματη διεπαφή (ή εναλλακτικά οποιαδήποτε άλλη διεπαφή αυτού του κόμβου είναι επιθυμητό να παρακολουθείται). Πραγματοποιούνται όλες οι ενέργειες που περιγράφηκαν στην ενότητα Cloud Facility, και στην συνέχεια η διάταξη λειτουργεί κατά τον προβλεπόμενο τρόπο. Παρατίθεται η αρχιτεκτονική διάταξη (Σχήμα 5.13).



Σχήμα 5.13 Ενσωμάτωση της υλοποίησης σε Wireless Facilities

6

Επίλογος

Σε αυτό το κεφάλαιο παρουσιάζεται μια σύνοψη της παρούσας διπλωματικής εργασίας και συμπεράσματα πάνω στην πειραματική διαδικασία. Επιπροσθέτως, παρατίθενται διάφορες προτάσεις για βελτίωση και περαιτέρω επέκταση στο μέλλον.

6.1 Σύνοψη και συμπεράσματα

Στην παρούσα διπλωματική εργασία προτάθηκε ένας μηχανισμός με στόχο την συλλογή δικτυακών δεδομένων από ετερογενείς υποδομές. Αρχικά, εκμεταλλευόμαστε την κίνηση σηματοδοσίας που ανταλλάσει εγγενώς, στα πλαίσια του πρωτοκόλλου OpenFlow, το λογισμικό ελέγχου με τους μεταγωγείς, και συμπληρωματικά χρησιμοποιώντας το πρότυπο sFlow αναπτύχθηκε ένας μηχανισμός παρακολούθησης. Ο μηχανισμός αυτός, επιτυγχάνει τον εντοπισμό όλων των flows (ιδιαίτερο ενδιαφέρον παρουσιάζει η ικανότητα του μηχανισμού να εντοπίζει τα mice flows, κάτι που αποδείχθηκε στο Κεφάλαιο 5 σε σύγκριση με την αντίστοιχη ικανότητα του sFlow) και επίσης αντιστοιχίζει sFlow samples τα οποία λαμβάνονται από τα switches, με τα flow rules έχοντας ως στόχο την πιο πιστή απεικόνιση της τρέχουσας κίνησης χωρίς να χρειάζεται να εκπνεύσουν οι κανόνες. Τα δεδομένα που εξάγονται είναι προσβάσιμα για οποιαδήποτε μελλοντική χρήση (είσοδος σε εφαρμογές ασφαλείας, Πειραματικές διαδικασίες κλπ). Ειδικά στην περίπτωση των υποδομών πολλαπλών ενοίκων προσφέρονται στον κάθε ενδιαφερόμενο στοιχεία που αφορούν τα resources που του αντιστοιχούν.

Προκειμένου αυτός ο μηχανισμός να είναι λειτουργικός για διάφορες υποδομές στο υποκεφάλαιο 5.2 προτάθηκαν διάφορες αναπροσαρμογές της αρχικής του δομής. Αυτές βασίζονται στο Open vSwitch και έχουν ως στόχο να ενσωματώσουν τη χρήση του πρωτοκόλλου OpenFlow στις διάφορες διατάξεις. Το παραπάνω δεν στοχεύει τόσο στον έλεγχο του data plane όσο στην εκμετάλλευση της λειτουργίας του OpenFlow βασισμένης σε ροές πακέτων ώστε να υπάρχουν τα ίδια πλεονεκτήματα που προσφέρει η αρχική πρόταση.

6.2 Βελτιώσεις και Μελλοντικές επεκτάσεις

Ο μηχανισμός που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας, δίνει τη δυνατότητα παρακολούθησης δικτυακών υποδομών με διαφορετικά χαρακτηριστικά. Η υλοποίηση συγκεντρώνει πληροφορίες για τα διαφορετικά flows της κίνησης και εντάσσει σε αυτά δείγματα πακέτων με χρήση του προτύπου sFlow.

Αρχικά, μια σημαντική βελτίωση θα μπορούσε να αποτελέσει η προσθήκη μιας βάσης δεδομένων για την καλύτερη αποθήκευση και προσπέλαση των δεδομένων. Προτιμούνται οι NoSQL βάσεις ώστε να βελτιστοποιείται η αποδοτικότητα του μηχανισμού σε σύγκριση με τον όγκο των δεδομένων, τόσο από την πλευρά του Flow Repository όσο και από την πλευρά της εφαρμογής που πρόκειται να αντλήσει τα δεδομένα για διάφορες χρήσεις.

Επίσης, επιπλέον βελτιώσεις θα μπορούσαν να γίνουν στο επίπεδο εικονικοποίησης. Όσον αφορά τον FlowVisor, στην τρέχουσα υλοποίηση χρησιμοποιείται μια εξωτερική εφαρμογή η οποία, αναλαμβάνει να ενημερώνεται ανά διαστήματα για το τρέχον flowspace με το οποίο έχει παραμετροποιηθεί ο FlowVisor και με την σειρά της ενημερώνει επίσης το Flow Repository. Θα ήταν αποδοτικότερο αν η διαδικασία γινόταν αυτόματα μέσω της εφαρμογής του FlowVisor όταν μεταβάλλεται το flowspace, κάτι που συνεπάγεται όμως την τροποποίηση του πηγαίου κώδικα του FlowVisor. Θα ήταν επίσης ιδιαίτερα ενδιαφέρουσα η προσπάθεια ενοποίησης του μηχανισμού με την πλατφόρμα OpenVirteX λαμβάνοντας υπ όψιν τις δυνατότητες που προσφέρει και το διαρκώς αυξανόμενο βαθμό αποδοχής του. Σε αντιπαραβολή με τον FlowVisor, το OpenVirtex δημιουργεί εικονικούς μεταγωγείς και ζεύξεις. Ακριβολογώντας δεν δημιουργεί ακριβώς, απλά χρησιμοποιεί την υποκείμενη δικτυακή υποδομή και κατασκευάζει αντιστοιχίες μεταξύ εικονικών και πραγματικών δικτύων. Έτσι παρουσιάζονται δυσκολίες κατάταξης των δειγμάτων που θα λαμβάνονται από την κάθε πραγματική δικτυακή συσκευή στον αντίστοιχο εικονικό μεταγωγέα που αντιλαμβάνεται ο controller του χρήστη.

Απώτερος στόχος θα ήταν η ομαδοποίηση όλων των διεργασιών που χρειάζονται για την ορθή λειτουργία του μηχανισμού και η αυτοματοποίηση τους για τις διάφορες υποδομές, προσφέροντας, την παρακολούθηση του δικτύου και την άντληση των στοιχείων, σε ένα αφαιρετικό επίπεδο σαν ένα Virtual Network Function [33] διαθέσιμο σε οποιοδήποτε χρήστη – ένοικο του δικτύου.

Βιβλιογραφία

[1] Open Networking Foundation, "Software-Defined Networking : The New Norm for Networks," *ONF White Paper,* 2012.

[2] OpenFlow protocol, http://archive.openflow.org.

[3] Open Networking Foundation, "OpenFlow Switch Specification Version 1.0". *https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf.*

[4] Open Networking Foundation, "OpenFlow Switch Specification Version 1.4". *https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.*

[5] About POX | NOXRepo, http://www.noxrepo.org/pox/about-pox/.

[6] Python Programming Language, http://www.python.org.

[7] POX Wiki, https://openflow.stanford.edu/display/ONL/POX+Wiki.

[8] The POX Controller | Github, https://github.com/noxrepo/pox.

[9] Open vSwitch, http://www.openvswitch.org/.

[10] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," 2013.

[11] PlanetLab, http://www.planet-lab.org.

[12] R. Sherwood, G. Gibb, et al, "FlowVisor: A Network Virtualization Layer," 2009.

[13] R. Sherwood, G. Gibb, et al, "Can the Production Network Be the Testbed?," 2010.

[14] A. Al-shabibi, M. Leenheer, B. Snow, "OpenVirteX : Make Your Virtual SDNs Programmable," 2014.

[15] sFlow, http://www.sflow.org/.

[16] Peter Phaal and Sonia Panchen sflow.org, Packet Sampling Basics http://www.sflow.org/packetSamplingBasics/index.htm.

[17] G. Liu, N. Neufeld, CERN, "Management of the LHCb network based on SCADA system," 2009.

[18] A. Curtis, J. Mogul. J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," 2011.

[19] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Computer Networks,* 2014.

[20] S. Shirali-shahreza, "FleXam: Flexible Sampling Extension for Monitoring and Security Applications in OpenFlow," 2013.

[21] C. Yu, C. Lumezanu, Y. Zhang et al., "FlowSense: Monitoring Network Utilization with Zero Measurement Cost".

[22] GENI, https://www.geni.net/.

[23] FlowVisor OpenFlow Aggregate Manager (FOAM), https://openflow.stanford.edu/display/FOAM/Home.

[24] Introducing JSON, http://json.org.

[25] JSON-RPC, http://json-rpc.org.

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

[27] top - display Linux tasks, http://linux.die.net/man/1/top.

[28] tcpreplay, http://linux.die.net/man/1/tcpreplay.

[29] ubuntu, http://www.ubuntu.com.

[30] tcpdump, http://www.tcpdump.org/.

[31] nmap - Network exploration tool and security / port scanner, http://linux.die.net/man/1/nmap.

[32] mergecap - Merges two or more capture files into one, http://linux.die.net/man/1/mergecap.

[33] ETSI GS NFV 002 V1.1.1 (2013-10).

[34] N. McKeown, T. Anderson, et al., "OpenFlow: Enable Innovation in Campus," 2008.

[35] Mininet An Instant Virtual Network on your Laptop (or other PC), http://mininet.org/.

[36] Programmable Virtual Networks (OpenVirteX), http://ovx.onlab.us.

[37] FlowVisor, https://openflow.stanford.edu/display/ONL/Flowvisor.

[38] FlowVisor Tutorial, https://wiki.onlab.us/display/PW/FlowVisor+Tutorial#FlowVisorTutorial-Introduction.

[39] FlowVisor - A network hypervisor, https://github.com/opennetworkinglab/flowvisor.

Παράρτημα

A. Πηγαίος κώδικας του αρχείου pepserver_stable.py

```
#!/usr/bin/python

from jsonrpclib.SimpleJSONRPCServer import SimpleJSONRPCServer
import pickle
import hashlib
from struct import unpack
from socket import inet_aton


# Global Dictionaries

# Active Flows
active = {}

# Expired Flows
expired = {}

# Used as a MAC Table, matching OF ports to MAC addresses
mac_table = {}

# List to save flowspace
flowspace = []

# Structure to match sflow fields to OpenFlow fields
mapper = {'inputPort': 'in_port',
        'srcMAC': 'dl_src',
        'dstMAC': 'dl_dst',
        'IPTOS': 'nw_tos',
        'IPProtocol': 'nw_proto',
        'srcIP': 'nw_src',
        'dstIP': 'nw_dst'}


def check_flowspace():
    """
            :params None

            :rtype : Serialized version of flowspace
    """
            # print flowspace
    a = pickle.dumps(flowspace)
    return a


def check2():
```

```
    """
            :params None

            :rtype : Serialized tuple containing active and expired dictionaries
    """
    args = (active,expired)
    b = pickle.dumps(args)
    return b


def update_flowspace(serial):
    """
            :params serialized version of flowspace

            :rtype :
    """
    # TODO delete previous flowspace (or keep a diff)
    # Update Flowspace
    print 'Updated'
    space = pickle.loads(serial)
    for l in range(0, len(space)):
        flowspace.append(space[l])


def assign_flowspace(hash_val, dpid):
    """
            :params hashed value of our flow rule and dpid

            :rtype
    """

    flow_fields = ['in_port', 'dl_src', 'dl_dst', 'dl_type', 'dl_vlan',
                   'nw_proto', 'nw_src', 'nw_dst', 'nw_tos', 'tp_src', 'tp_dst']

            # List throught all flowspace rules
    for a in range(0, len(flowspace)):
                    # if we examine a flowspace rule that refers to another dpid, continue
        if flowspace[a]['dpid'][-1] != dpid:
            continue
                    # for every match field
        for k in flow_fields:
            if k not in flowspace[a]['match']:
                # wildcard in flow space so we dont care
                # print '1'
                continue
            elif active[dpid][hash_val]['match'][k] is None:
                # flowspace field is not a wildcard
                # if flow space does not  have the same value, break
                # print '2'
                break
            elif flowspace[a]['match'][k] == active[dpid][hash_val]['match'][k]:
                # if both above conditions are not met, check if their values are identical
                                        #print '3'
                continue
            else:
                                        # special treatment for ip subnets
                if (k == 'nw_src') or (k == 'nw_dst'):
                    # split flowspace and openflow rules to A.B.C.D/M format

                    fsp = flowspace[a]['match'][k].split('/')
                    frl = active[dpid][hash_val]['match'][k].split('/')
```

```
                if len(fsp) == 1:
                                                        # if length is equal to 1 its just an IP ( A.B.C.D )
not a subnet
                    break
                else:
                    # A.B.C.D/M
                    tmp = int (fsp[1])
                                                        # convert equivalent subnet mask (M)
                    fsp_mask = int ((tmp* '1' + (32 - tmp) * '0'), 2)

                    if len(frl) == 1:
                                                                # check if flow rule is a a.b.c.d
                                                                # Then Check if (A.B.C.D && M ==
a.b.c.d)
                        if (unpack("!L", inet_aton(frl[0]))[0] & fsp_mask) == unpack("!L",
inet_aton(fsp[0]))[0]:
                            continue
                        else:
                            break
                    else:
                                                                # flow rule is a.b.c.d/m
                                                                # do the same but this time check if flow
rule refers to a subset of flowspace
                        tmp = int (frl[1])
                        frl_mask = int ((tmp * '1' + (32 - tmp) * '0'), 2)

                        if frl_mask > fsp_mask:
                            break
                        elif (unpack("!L", inet_aton(frl[0]))[0] & fsp_mask) == unpack("!L",
inet_aton(fsp[0]))[0]:
                            continue
                        else:
                            break

                else:
                    # not same value, (or something else)
                    # print '4'
                    break

            else:
                # exhausted flow rule, so flowspace is found
                active[dpid][hash_val]['slice_Owner'] = flowspace[a]['slice-action'][0]['slice-name']
                break
        else:
                        # not part of any flowspace
            print 'Not part of any flowspace'
            active[dpid][hash_val]['slice_Owner'] = None


def return_active():
    pass


def return_expired():
    pass


def construct_hashed_key(match, time_s=None, hash_f=1):
    """
            :params (OpenFlow match, timestamp, flag)
```

```
    Depending on the value of hash_f returns different
    hashed objects

    :rtype : hashed value
    """
    key = (match.in_port,
           match.dl_src.toStr(),
           match.dl_dst.toStr(),
           match.dl_type,
           match.dl_vlan,
           # match.dl_vlan_pcp,
           match.nw_proto,
           match.nw_src.toStr(),
           match.nw_dst.toStr(),
           match.nw_tos,
           match.tp_src,
           match.tp_dst,
           time_s)
    # print 'printing the values of our soon to be hashed key'
    # print key

    if hash_f == 1:
                        #if hash_f == 1 dont include the timestamp parameter
        print hash(key[:len(key) -1])
        return hash(key[:len(key)-1])
    else:
                        # else include it
        print hash(key[:len(key) -1])
        return hash(key)


def construct_hashed_sflow(match):
    """
    :param match: OpenFlow match object
    :return: hashed value
    """
    # Constructing key
    key = (match['in_port'],
           match['dl_src'],
           match['dl_dst'],
           match['dl_type'],
           match['dl_vlan'],
           # match['dl_vlan_pcp'],
           match['nw_proto'],
           match['nw_src'],
           match['nw_dst'],
           match['nw_tos'],
           match['tp_src'],
           match['tp_dst']
           )

    # print key
    print hash(key)
    return hash(key)


def construct_dict(match, dpid):
    """
    :param match: OpenFlow match object, dpid
    :return: dictionary with those values
```

```
    """
    di = {}
    di['in_port'] = match.in_port
    di['dl_src'] = match.dl_src.toStr()
    di['dl_dst'] = match.dl_dst.toStr()
    di['dl_type'] = match.dl_type
    di['dl_vlan'] = match.dl_vlan
    # di['dl_vlan_pcp'] = match.dl_vlan_pcp
    di['nw_proto'] = match.nw_proto
    di['nw_src'] = match.nw_src.toStr()
    di['nw_dst'] = match.nw_dst.toStr()
    di['nw_tos'] = match.nw_tos
    di['tp_src'] = match.tp_src
    di['tp_dst'] = match.tp_dst
    di['dpid'] = dpid
    return di


def construct_new_entry(serialized_match):
            """
    :param match: serialized OpenFlow match
    :return:
    """
    args = pickle.loads(serialized_match)
    match = args[0]
    dpid = args[1]
    time = float(args[2])
    # print '\n\n___Installing New Entry___'

            # Construct hashed value based on match
    d = construct_hashed_key(match)

    # print ("Switch: %s,\t Flow(Hash): %s" % (dpid, d))

    # Check if the dpid entry has been initialized
    if dpid not in active:
        active[dpid] = {}

    # Creating the Flow Entry
            # if the flow rule is not already in the Active structure
    if d not in active[dpid]:
        active[dpid][d] = {'counters': {'counterX': 1, 'Packet_In': 1, 'mult_Packet_in': 1},
                           'match': construct_dict(match, dpid),
                           'timestamps': {'start': time, 'end': None},
                           'headers': {},
                           'slice_Owner': None
                           }
        hlp = active[dpid][d]['match']

        if dpid not in mac_table:
            mac_table[dpid] = {}
            mac_table[dpid][hlp['dl_src']] = hlp['in_port']
        else:
            # should i do nothing? or overwrite in any case
            mac_table[dpid][hlp['dl_src']] = hlp['in_port']

        # assign_flowspace(d, dpid)
            # else increment a counter measuring multiple packetIns
    else:
        active[dpid][d]['counters']['mult_Packet_in'] += 1
```

```
            # MAC Table
    if dpid not in mac_table:
        mac_table[dpid] = {}
        mac_table[dpid][hlp['dl_src']] = hlp['in_port']
    else:
        mac_table[dpid][hlp['dl_src']] = hlp['in_port']

    # assign_flowspace(d, dpid)



def move_to_expired(serialized_match):
            """
    :param match: serialized OpenFlow match
    :return:
    """
    args = pickle.loads(serialized_match)
    match = args[0]
    dpid = args[1]
    time = float(args[2])
    # print "\n\n___Moving Expired Entry___"

            # create a hash value for the expired flow (not including timestamp)
    e = construct_hashed_key(match)

            # if there is such a flow
    if e in active[dpid]:
                # remove the flow
        found = active[dpid].pop(e)
        found['timestamps']['end'] = time

                        # now create another hashed value, including timestamp
        d = construct_hashed_key(match, found['timestamps']['start'], 0)

        if dpid not in expired:
            expired[dpid] = {}

        expired[dpid][d] = found
        if active[dpid] == {}:
            del active[dpid]


def collect_sflow(flow):
            """
    :param match: serialized sflow sample
    :return:
    """

    # print '\n\n___Incrementing Counter Entry___'
    sflow = {}

    b = pickle.loads(flow)

            # TODO will try with sflow = b
    for kk, vv in b.iteritems():
        sflow[kk] = vv

    # transition from sflow fields {} to, openflow match {}
    match = {}
```

```
# convert dl_type
dpid = sflow.pop('dpid')
match['dl_type'] = sflow.pop('dl_type')

# manipulate VLAN tag
if sflow['in_vlan'] == '0':
   match['dl_vlan'] = 65535
   del sflow['in_vlan']
elif sflow['in_vlan'] is not None:
   match['dl_vlan'] = int(sflow.pop('in_vlan'))
else:
   match['dl_vlan'] = None

# unifying UDP/TCP/ICMP or wildcarding
if 'SrcPort' in sflow:
   # no need to check for Dst Port, since this is a sample and bound to have DST aswell
   match['tp_src'] = int(sflow['SrcPort'])
   match['tp_dst'] = int(sflow['DstPort'])
elif 'ICMPType' in sflow:
   match['tp_src'] = int(sflow['ICMPType'])
   match['tp_dst'] = int(sflow['ICMPCode'])
else:
   match['tp_src'] = None
   match['tp_dst'] = None

fields = ['inputPort', 'srcMAC', 'dstMAC',
        'IPProtocol', 'srcIP', 'dstIP', 'IPTOS']

intfields = ['inputPort', 'IPProtocol', 'IPTOS']

        # translate rest of the fields using the structure mapper{}
for a in fields:
   if a not in sflow:
      match[mapper[a]] = None
   elif a in intfields:
      match[mapper[a]] = int(sflow.pop(a))
   else:
      match[mapper[a]] = sflow.pop(a)

# modify ingress port using mac_table
match['in_port'] = mac_table[dpid][match['dl_src']]

# print 'OpenFlow Match:'
# print match

try:
   d = construct_hashed_sflow(match)

   print 'Printing Hash: %d\n\n' % d
   # print 'Printing of all flows of the DPID: %s' % dpid
   # print active[dpid]

   if d in active[dpid]:
      print 'Hash Found'
      active[dpid][d]['counters']['counterX'] += 1
      # adding headers of packet

      # f = match['headerBytes']
      # e = hash(f)
      # if e not in active[dpid][d]['headers']:
      #    active[dpid][d]['headers'][e] = f
```

```
            # print active
        else:
            print 'Hash not found'
                                    # same functionality as in assign_flowspace()
            for kk, vv in active[dpid].iteritems():
                for ll, ww in vv['match'].iteritems():
                    if ww is None:
                        # can continue with next iteration
                        continue
                    elif (ll not in match) or (match[ll] is None):
                        # key error will not be raised, because if key not present second 'or' will not be evaluated
                        break
                    elif match[ll] == ww:
                        # if sample attribute is equal with the match then we have no difference
                        continue
                    else:
                        if (ll == 'nw_src') or (ll == 'nw_dst'):
                            # special support for ip
                            frl = ww.split('/')
                            if len(frl) == 1:
                                # A.B.C.D
                                # should have been caught by outer elif
                                break

                            # A.B.C.D/M
                            tmp = int(frl[1])
                            frl_mask = int((tmp * '1' + (32 - tmp) * '0'), 2)
                            # sflow: a.b.c.d
                            if (unpack("!L", inet_aton(match[ll]))[0] & frl_mask) == unpack("!L",
inet_aton(frl[0]))[0]:
                                continue
                            else:
                                break
                        else:
                            # not same value, (or something else)
                            # print '4'
                            break
                else:
                    # if loop is completed with no breaks this value under investigation should be incremented
                    active[dpid][kk]['counters']['counterX'] += 1
                    print 'Flow Reconstructed'
                    break
            else:
                print 'Hash Reconstrution failed.\n'
    except:
        print 'Error caught.\nPrinting match field'
        print sflow
        print match
        print 'This should not have happened'

            # print ("Switch: %s,\t sFlow(Hash): %s" % (dpid, d))


if __name__ == "__main__":
    # binding server to port
    server = SimpleJSONRPCServer(('localhost', 8080))

    # register functions for usage
    server.register_function(construct_new_entry)
    server.register_function(move_to_expired)
```

```
server.register_function(collect_sflow)
server.register_function(check_flowspace)
server.register_function(check2)
server.register_function(update_flowspace)

# start server
server.serve_forever()
```

Β. Πηγαίος κώδικας του αρχείου flowrem.py

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
from pox.lib.util import dpidToStr
from pox.lib.addresses import EthAddr
from collections import namedtuple
from pox.openflow import *
from pox.forwarding import l2_learning
import os
import pickle
import jsonrpclib
import time

# Create Loggers
log = core.getLogger()
log1 = core.getLogger("Flow Installed")
log2 = core.getLogger("Flow Removed")
# Create a new jsonrpclib.Server object, bound to an IP
server = jsonrpclib.Server('http://10.0.0.2:8080')


class CustomEvent(Event):
    """
    Our own custom event
    """

    def __init__(self, connection, ofp):
        Event.__init__(self)
        self.connection = connection
        self.ofp = ofp
        self.dpid = connection.dpid


class EventSourcer (EventMixin):
    """
    Our event raise class. Can only raise CustomEvent
    """
    _eventMixin_events = set([
    CustomEvent,
    ])

# Create a new object of type EventSourcer
handler = EventSourcer()
```

```
class FlowRemovalHandler (EventMixin):
    """
    The core functionality of our module.
    Key methods are the handlers for Custom Events and Flow Removed Events
    Each one serializes the ofp.match and performs an RPC to Flow Repository
    """
    def __init__(self):
        self.listenTo(core.openflow)
        self.listenTo(handler)
        log.debug("...Enabling Flow Removal Module...")

    def _handle_CustomEvent(self, event):
        # log1.debug('Flow Installed on switch: %s', event.connection)
        # log1.debug(event.ofp.match)

        args = ()
        connction = event.connection
                    # create tuple, serialize it and perform an RPC
        args = (event.ofp.match, str(connction.dpid), str(time.time()))
        b = pickle.dumps(args)

        server.construct_new_entry(b)
        # a = server.construct_new_entry(b)
        # c = pickle.loads(a)

    def _handle_FlowRemoved(self, event):

        args = ()
        connection = event.connection
                    # create tuple, serialize it and perform an RPC
        args = (event.ofp.match, str(connection.dpid), str(time.time()))
        b = pickle.dumps(args)
        a = server.move_to_expired(b)
        c = pickle.loads(a)
        log2.debug('Flow Removed from switch: %s', event.connection)
        # log2.debug(c)


def launch():
    """
    Starting the module
    """
    core.registerNew(FlowRemovalHandler)


def invoker(con, msg):
    """
    When called raises CustomEvent
    """
    handler.raiseEvent(CustomEvent(con, msg))
```

C. Πηγαίος κώδικας του αρχείου l2_learning.py

```
# Copyright 2011 James McCauley
#
# This file is part of POX.
#
# POX is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# POX is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with POX.  If not, see <http://www.gnu.org/licenses/>.

"""
An L2 learning switch.

It is derived from one written live for an SDN crash course.
It is somwhat similar to NOX's pyswitch in that it installs
exact-match rules for each flow.
"""

from pox.openflow import *
from pox.lib.revent import *
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
import time

log = core.getLogger()

# We don't want to flood immediately when a switch connects.
# Can be overriden on commandline.
_flood_delay = 0

class LearningSwitch (object):
  """
  The learning switch "brain" associated with a single OpenFlow switch.

  When we see a packet, we'd like to output it on a port which will
  eventually lead to the destination.  To accomplish this, we build a
  table that maps addresses to ports.

  We populate the table by observing traffic.  When we see a packet
  from some source coming from some port, we know that source is out
  that port.

  When we want to forward traffic, we look up the desintation in our
  table.  If we don't know the port, we simply send the message out
  all ports except the one it came in on.  (In the presence of loops,
  this is bad!).
```

```
  In short, our algorithm looks like this:

  For each packet from the switch:
  1) Use source address and switch port to update address/port table
  2) Is transparent = False and either Ethertype is LLDP or the packet's
     destination address is a Bridge Filtered address?
     Yes:
        2a) Drop packet -- don't forward link-local traffic (LLDP, 802.1x)
            DONE
  3) Is destination multicast?
     Yes:
        3a) Flood the packet
            DONE
  4) Port for destination address in our address/port table?
     No:
        4a) Flood the packet
            DONE
  5) Is output port the same as input port?
     Yes:
        5a) Drop packet and similar ones for a while
  6) Install flow table entry in the switch so that this
     flow goes out the appopriate port
     6a) Send the packet out appropriate port
  """
  def __init__ (self, connection, transparent):
    # Switch we'll be adding L2 learning switch capabilities to
    self.connection = connection
    self.transparent = transparent

    # Our table
    self.macToPort = {}

    # We want to hear PacketIn messages, so we listen
    # to the connection
    connection.addListeners(self)

    # We just use this to know when to log a helpful message
    self.hold_down_expired = _flood_delay == 0

    #log.debug("Initializing LearningSwitch, transparent=%s",
    #          str(self.transparent))

  def _handle_PacketIn (self, event):
    """
    Handle packet in messages from the switch to implement above algorithm.
    """

    packet = event.parsed

    def flood (message = None):
      """ Floods the packet """
      msg = of.ofp_packet_out()
      if time.time() - self.connection.connect_time >= _flood_delay:
        # Only flood if we've been connected for a little while...

        if self.hold_down_expired is False:
          # Oh yes it is!
          self.hold_down_expired = True
```

```
      log.info("%s: Flood hold-down expired -- flooding",
          dpid_to_str(event.dpid))

    if message is not None: log.debug(message)
    #log.debug("%i: flood %s -> %s", event.dpid,packet.src,packet.dst)
    # OFPP_FLOOD is optional; on some switches you may need to change
    # this to OFPP_ALL.
    msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
  else:
    pass
    #log.info("Holding down flood for %s", dpid_to_str(event.dpid))
  msg.data = event.ofp
  msg.in_port = event.port
  self.connection.send(msg)

def drop (duration = None):
  """
  Drops this packet and optionally installs a flow to continue
  dropping similar ones for a while
  """
  if duration is not None:
    if not isinstance(duration, tuple):
      duration = (duration,duration)
    msg = of.ofp_flow_mod()
    msg.match = of.ofp_match.from_packet(packet)
    msg.idle_timeout = duration[0]
    msg.hard_timeout = duration[1]
    msg.buffer_id = event.ofp.buffer_id
    self.connection.send(msg)
  elif event.ofp.buffer_id is not None:
    msg = of.ofp_packet_out()
    msg.buffer_id = event.ofp.buffer_id
    msg.in_port = event.port
    self.connection.send(msg)

self.macToPort[packet.src] = event.port # 1

if not self.transparent: # 2
  if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
    drop() # 2a
    return

if packet.dst.is_multicast:
  flood() # 3a
else:
  if packet.dst not in self.macToPort: # 4
    flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
  else:
    port = self.macToPort[packet.dst]
    if port == event.port: # 5
      # 5a
      log.warning("Same port for packet from %s -> %s on %s.%s.  Drop."
          % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
      drop(10)
      return
    # 6
    log.debug("installing flow for %s.%i -> %s.%i" %
          (packet.src, event.port, packet.dst, port))
```

```
        msg = of.ofp_flow_mod()
        # Adding Flag in order to receive Flow Removed Events
        msg.flags = of.OFPFF_SEND_FLOW_REM
        msg.match = of.ofp_match.from_packet(packet, event.port)
        msg.idle_timeout = 10
        msg.hard_timeout = 30
        msg.actions.append(of.ofp_action_output(port = port))
        msg.data = event.ofp # 6a
        self.connection.send(msg)

        # Importing our custom module in order to raise a custom event and create
        # a new entry for the flow installed
        import pox.misc.flowrem as docker

        docker.invoker(event.connection, msg)

class l2_learning (object):
  """
  Waits for OpenFlow switches to connect and makes them learning switches.
  """
  def __init__ (self, transparent):
    core.openflow.addListeners(self)
    self.transparent = transparent

  def _handle_ConnectionUp (self, event):
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)


def launch (transparent=False, hold_down=_flood_delay):
  """
  Starts an L2 learning switch.
  """
  try:
    global _flood_delay
    _flood_delay = int(str(hold_down), 10)
    assert _flood_delay >= 0
  except:
    raise RuntimeError("Expected hold-down to be a number")

  core.registerNew(l2_learning, str_to_bool(transparent))
```

D. Πηγαίος κώδικας του αρχείου flow_stats.py

```
#!/usr/bin/python
# Copyright 2012 William Yu
# wyu@ateneo.edu
#
# This file is part of POX.
#
# POX is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
```

```
# POX is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with POX. If not, see <http://www.gnu.org/licenses/>.
#

"""
This is a demonstration file created to show how to obtain flow
and port statistics from OpenFlow 1.0-enabled switches. The flow
statistics handler contains a summary of web-only traffic.
"""

# standard imports
import time
from datetime import datetime
from pox.core import core
from pox.lib.util import dpidToStr
import pox.openflow.libopenflow_01 as of

# include as part of the betta branch
from pox.openflow.of_json import *

log = core.getLogger()
dict_stats = {}
counters_fstats = 0
counters_value = 0


def construct_hashed_key(match):
    """
    Depending on the value of hash_f returns different
    hashed objects
    :rtype : hash
    """
    key = (match.in_port,
           match.dl_src.toStr(),
           match.dl_dst.toStr(),
           match.dl_type,
           match.dl_vlan,
           # match.dl_vlan_pcp,
           match.nw_proto,
           match.nw_src.toStr(),
           match.nw_dst.toStr(),
           match.nw_tos,
           match.tp_src,
           match.tp_dst)

    return hash(key)



# handler for timer function that sends the requests to all the
# switches connected to the controller.
def _timer_func ():
  for connection in core.openflow._connections.values():
```

```
    connection.send(of.ofp_stats_request(body=of.ofp_flow_stats_request()))
    connection.send(of.ofp_stats_request(body=of.ofp_port_stats_request()))
  log.debug("Sent %i flow/port stats request(s)", len(core.openflow._connections))

# handler to display flow statistics received in JSON format
# structure of event.stats is defined by ofp_flow_stats()
def _handle_flowstats_received (event):
  stats = flow_stats_to_list(event.stats)
  log.debug("FlowStatsReceived from %s: %s",
    dpidToStr(event.connection.dpid), stats)

  # Get number of bytes/packets in flows for web traffic only
  start = datetime.now()
  global counters_fstats
  global counters_value
  counters_fstats += 1

  for f in event.stats:
    # Hash each match object
    hk = construct_hashed_key(f.match)

            # Simple manipulation of statistics
    if hk not in dict_stats:
      dict_stats[hk] = {'bytes': f.byte_count, 'packets': f.packet_count, 'appeared': 1, 'time': time.time()}
    else:
      dict_stats[hk]['bytes'] += f.byte_count
      dict_stats[hk]['packets'] += f.packet_count
      dict_stats[hk]['appeared'] += 1

  end = datetime.now()
  delta = end - start
  counters_value += float(delta.total_seconds())
  print counters_value/counters_fstats



# handler to display port statistics received in JSON format
def _handle_portstats_received (event):
  stats = flow_stats_to_list(event.stats)
  log.debug("PortStatsReceived from %s: %s",
    dpidToStr(event.connection.dpid), stats)

# main functiont to launch the module
def launch ():
  from pox.lib.recoco import Timer

  # attach handsers to listners
  core.openflow.addListenerByName("FlowStatsReceived",
    _handle_flowstats_received)
  core.openflow.addListenerByName("PortStatsReceived",
    _handle_portstats_received)

  # timer set to execute every five seconds
  Timer(5, _timer_func, recurring=True)
```

Ε. Πηγαίος κώδικας του αρχείου sflow.py

```
#!/usr/bin/python

import time
import datetime
import os
import sys
import pickle
import jsonrpclib
import ast

# Create a mapping between sflow agent ID and DPID
sflow_dpid = {}

# Crate a new object of class jsonrpclib.Server
server = jsonrpclib.Server('http://localhost:8080')


def call_and_peek_output(cmd, shell=False):
    import pty
    import subprocess
    # Create a new pair master, slave
    master, slave = pty.openpty()
    print cmd
    p = subprocess.Popen(cmd, shell=shell, stdin=None, stdout=slave, close_fds=True)
    os.close(slave)
    line = ""
    while True:
        try:
            ch = os.read(master, 1)
        except OSError:
            # We get this exception when the spawn process closes all references to the
            # pty descriptor which we passed him to use for stdout
            # (typically when it and its childs exit)
            break
        line += ch
        if ch == '\n':
            yield line
            line = ""
    if line:
        yield line

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


def separate_fields(mystr):
    """
            :params: string

    separating mystr into two parts, field - value pairs.

            :returns: pair of field and its value
    """
    p = mystr.rfind(' ')
    field = mystr[:p]
```

```
    value = mystr[p+1:-2]
    if field == "TCPDstPort" or field == "UDPDstPort":
        field = "DstPort"
    elif field == "TCPSrcPort" or field == "UDPSrcPort":
        field = "SrcPort"

    return(field,value)


def sflowParser():
    print sflow_dpid

    # Fields we are interested in keeping from an sflow sample
    tuples = ['sourceId', 'inputPort', 'outputPort', 'srcMAC', 'dstMAC',
              'in_vlan', 'IPProtocol', 'srcIP', 'dstIP', 'IPTOS', 'headerBytes',
              'SrcPort', 'DstPort', 'ICMPType', 'ICMPCode']
    # flag to identify sample start - end
    switch = 0

    for l in call_and_peek_output(['sflowtool -p %s' % sys.argv[1]], shell=True):
        # ---- Condition to grab the 1st line where a new flowsample starts
        if 'FLOWSAMPLE' in l:
            switch = 1
            flow = {}

        # ---- Condition to identify where the flowsample ends
        if switch == 1 and "endSample" in l:
                                # set flag to 0
            switch = 0

                                # process samples

            # support for field, ethernet type protocol using header values
            b = flow['headerBytes'].split('-')
            ether_type = b[12] + b[13]
            flow['dl_type'] = int(ether_type, 16)

            if ether_type == '0806':
                flow['srcIP'] = str(int(b[28], 16)) + '.' + str(int(b[29], 16)) + '.' \
                            + str(int(b[30], 16)) + '.' + str(int(b[31], 16))

                flow['dstIP'] = str(int(b[38], 16)) + '.' + str(int(b[39], 16)) + '.' \
                            + str(int(b[40], 16)) + '.' + str(int(b[41], 16))

                flow['IPProtocol'] = int(b[20] + b[21], 16)

            l = ''
            m = ''

            #  construct mac address with colons
            for k in (0, 2, 4, 6, 8, 10):
                l = l + flow['srcMAC'][k:k+2] + ':'
                m = m + flow['dstMAC'][k:k+2] + ':'

            # Remove last colon
            flow['srcMAC'] = l[:17]
            flow['dstMAC'] = m[:17]
```

```
            # added dictionary mapping, or default situational mininet support
            if not sflow_dpid:
                flow['sourceId'] = flow['sourceId'][3:]
                flow['dpid'] = str(int(flow['sourceId']) + 1)
            else:
                flow['dpid'] = sflow_dpid[flow['sourceId']]

            print 'flow:'
            print flow

            b = pickle.dumps(flow)
            #if 'outputPort' in flow:
             #   server.collect_sflow(b)
            #else:
             #   print 'packet in Sampled\n'
            # The above is used not to double-count the flows, since counter is initialized in one
            # The above has to be modified since in our case we drop traffic

            try:
                server.collect_sflow(b)
            except:

                print 'Unknown Error'

        # ---- Condition to split the fields of flowsamples
        if switch == 1:
            field, value = separate_fields(l)
            if field in tuples:
                flow[field] = value


if __name__ == "__main__":
    a = len(sys.argv)
    if a == 2:
        sflowParser()
    elif a == 3:
        file_name = sys.argv[2]
        try:
                                # construct sflow_dpid mapping based on a file
            with open(file_name) as f:
                for line in f:
                    (key, val) = line.split()
                    sflow_dpid[key] = val

            sflowParser()
        except IOError:
            print "\nNo such file: \t%s\n" %file_name
            print "Provide EXACTLY two arguments <target_port> <filename> (JSON Format)\n"
    else:
        print "Wrong Usage: Provide EXACTLY two arguments <target_port> <filename> (JSON
Format)"
        exit()
```

F. Πηγαίος κώδικας του αρχείου get_flowspace.py

```
#!/usr/bin/python

import time
import datetime
import os  # threading, re
import sys
import ast
import jsonrpclib
import pickle

server = jsonrpclib.Server('http://10.0.0.2:8080')

a = list()

def call_and_peek_output(cmd, shell=False):
    import pty
    import subprocess
    master, slave = pty.openpty()
    print cmd
    p = subprocess.Popen(cmd, shell=shell, stdin=None, stdout=slave, close_fds=True)
    os.close(slave)
    line = ""
    while True:
        try:
            ch = os.read(master, 1)
        except OSError:
            # We get this exception when the spawn process closes all references to the
            # pty descriptor which we passed him to use for stdout
            # (typically when it and its childs exit)
            break
        line += ch
        if ch == '\n':
            yield line
            line = ""
    if line:
        yield line

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


def flowvisor_parser():
    for l in call_and_peek_output('fvctl -n list-flowspace', shell=True):
        try:
            k = ast.literal_eval(l)
            a.append(k)
        except SyntaxError:
                                # this ignores the first line
            print "First Line"


    fe = tuple(a)
    # print fe
    c = pickle.dumps(fe)
    server.update_flowspace(c)
```

```
if __name__ == "__main__":
    if len(sys.argv) == 1:
        flowvisor_parser()
    else:
        print "Wrong Usage: Provide NO arguments"
        exit()
```

G. Πηγαίος κώδικας του αρχείου client.py

```
#!/usr/bin/python

import os
import sys
import pickle
import csv
import time

flow_match = ['in_port', 'dl_src', 'dl_dst', 'dl_type',
              'dl_vlan', 'nw_proto', 'nw_src', 'nw_dst',
              'nw_tos', 'tp_src', 'tp_dst']

import jsonrpclib
server = jsonrpclib.Server('http://localhost:8080')


def construct_hashed_sflow(match):
    """
    :params: OpenFlow match object
    :return: hashed value computed on that object
    """
    # Constructing key
    key = (match['in_port'],
           match['dl_src'],
           match['dl_dst'],
           match['dl_type'],
           match['dl_vlan'],
           # match['dl_vlan_pcp'],
           match['nw_proto'],
           match['nw_src'],
           match['nw_dst'],
           match['nw_tos'],
           match['tp_src'],
           match['tp_dst']
           )

    return hash(key)


def grab():
    #
    b = server.check2()
    a = pickle.loads(b)
    return a
```

```
def flowt(table, start=0, end=float("inf")):
    a = grab()
    # slice_name = raw_input("Please Enter a slice to print flows for:\t")

            with open('flowt.csv', 'w') as csvfile:
        fieldnames = ['dpid', 'hash', 'in_port', 'dl_src', 'dl_dst', 'dl_type', 'dl_vlan', 'nw_proto', 'nw_src',
                      'nw_dst', 'nw_tos', 'tp_src', 'tp_dst', 'Packet_Counter', 'Packet_In','Slice_Owner']

        writer = csv.DictWriter(csvfile, fieldnames = fieldnames)
        writer.writeheader()

                        for kk, vv in a[table].iteritems():
                                for l, w in vv.iteritems():
                                        if w['timestamps']['start'] > start and w['timestamps']['end'] <
end and w['slice_Owner'] == slice_name:

                b = w['match']
                b['dpid'] = kk
                b['hash'] = l
                b['Packet_Counter'] = w['counters']['counterX']
                b['Packet_In'] = w['counters']['PacketIn']
                b['Slice_Owner'] = w['slice_Owner']
                writer.writerow(b)


def active():
    print 'Printing Active Counters\n\n'
    flowt(0)


def expired():
    print 'Printing Expired Counters\n\n'
    flowt(1)


def aggregate():
    print 'Printing Aggregate Counters\n\n'
    a = grab()
    aggr = a[0]
    for dpid, rest in a[1].iteritems():
        for hashes, dicts in rest.iteritems():
            hk = construct_hashed_sflow(dicts['match'])


            print 'Hash Key: %s' % hk
            if dpid not in aggr:
                aggr[dpid] = {}
                aggr[dpid][hk] = dicts
                aggr[dpid][hk]['counters']['PacketIn'] = 1
            elif hk not in aggr[dpid]:
                aggr[dpid][hk] = dicts
                aggr[dpid][hk]['counters']['PacketIn'] = 1
            else:
                aggr[dpid][hk]['counters']['counterX'] += dicts['counters']['counterX']
                try:
                    aggr[dpid][hk]['counters']['PacketIn'] += 1
```

```
            except KeyError:
                aggr[dpid][hk]['counters']['PacketIn'] = 1

    print '\n'

    # the following is used for the Accuracy part of the experiment
    with open('aggregates.csv', 'w') as csvfile:
        fieldnames = ['dpid', 'hash', 'in_port', 'dl_src', 'dl_dst', 'dl_type', 'dl_vlan', 'nw_proto', 'nw_src',
                      'nw_dst', 'nw_tos', 'tp_src', 'tp_dst', 'Packet_Counter', 'Packet_In','Slice_Owner']

        writer = csv.DictWriter(csvfile, fieldnames = fieldnames)
        writer.writeheader()

        for kk, vv in aggr.iteritems():
            for l, w in vv.iteritems():
#               if w['slice_Owner'] == slice_name:
                                                # choice = raw_input("Please Enter the desired slice:\t")

                    b = w['match']
                    b['dpid'] = kk
                    b['hash'] = l
                    b['Packet_Counter'] = w['counters']['counterX']
                    b['Packet_In'] = w['counters']['PacketIn']
                    b['Slice_Owner'] = w['slice_Owner']

                    writer.writerow(b)


def timewindows():
    print "Current time is: %s" % time.strftime('%d.%m.%Y %H:%M:%S',
time.localtime(time.time()))

    # Enter local time
    # Look up conversion of date in time.time format
    print "Please use ONLY the following format: d.m.Y H:M:S"
    start = raw_input("Please enter from where to start:\t")
    if not start:
        start = 0
    else:
        start = int(time.mktime(time.strptime(start, "%d.%m.%Y %H:%M:%S")))

    end = raw_input("Please enter from where to end:  \t")

    if not end:
        end = float("inf")
    else:
        end = int(time.mktime(time.strptime(end, "%d.%m.%Y %H:%M:%S")))

            # read which structure should be parsed
    table = raw_input("Select:\n0: \t Active\n1: \t Expired\n\n")

            # make sure the right ones are chosen
    while table not in ['0', '1']:
        print "Select 0 or 1"
        table = raw_input("Select:\n0: \t Active\n1: \t Expired\n\n")

    print 'Printing Counters for timewindow %s %s \n\n' % (start, end)
```

```
    if table == 0:
        print 'Printing Active Counters\n\n'
    else:
        print 'Printing Expired Counters\n\n'

    flowt(int(table), start, end)

def check4():
    b = server.check_flowspace()
    a = pickle.loads(b)
    print a
    return a



if __name__ == '__main__':
    print 'Enter:'
    print '1:\t Active'
    print '2:\t Expired'
    print '3:\t Aggregates'
    print '4:\t Select timewindows'
    print '5:\t to exit\n\n'

    choice = raw_input("Please Enter a valid option:\t")
    print "\n\n"
    choices = ['1', '2', '3', '4', '5']
    option = {'1': active,
              '2': expired,
              '3': aggregate,
              '4': timewindows,
              '5': sys.exit}

    while True:
        while choice not in choices:
            choice = raw_input("Please Enter a valid option:\t")
            print "\n\n"
        option[choice]()
        choice = None
```