ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ & ΣΥΣΤΗΜΑΤΩΝ

NETMODE - NETwork MANAGEMENT & OPTIMAL DESIGN LABORATORY

Σχεδιασμός και Ανάπτυξη Μηχανισμών Διαχείρισης και Παραμετροποίησης Εικονικών Συσκευών Δικτύωσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Μ. Γιώτης

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

Αθήνα, Ιούλιος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ & ΣΥΣΤΗΜΑΤΩΝ

Σχεδιασμός και Ανάπτυξη Μηχανισμών Διαχείρισης και Παραμετροποίησης Εικονικών Συσκευών Δικτύωσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Μ. Γιώτης

Επιβλέπων: Βασίλειος Μάγκλαρης

Καθηγητής Ε.Μ.Π.

NETMODE - NETwork MAnagement & OPtimal DEsign LABoratory

Αθήνα, Ιούλιος 2011

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ & ΣΥΣΤΗΜΑΤΩΝ

NETMODE - NETWORK MANAGEMENT & OPTIMAL DESIGN LABORATORY

Σχεδιασμός και Ανάπτυξη Μηχανισμών Διαχείρισης και Παραμετροποίησης Εικονικών Συσκευών Δικτύωσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Κωνσταντίνου Μ. Γιώτη

Επιβλέπων: Βασίλειος Μάγκλαρης

Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 18η Ιουλίου 2011.

.....	.....	.....
Βασίλειος Μάγκλαρης	Συμεών Παπαβασιλείου	Δημήτριος Καλογεράς
Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.	Ερευνητής ΕΠΙΣΕΥ

Αθήνα, Ιούλιος 2011

.....

ΓΙΩΤΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

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

Περίληψη

Το διαδίκτυο, όπως το γνωρίζουμε μέχρι σήμερα, εμφανίζει περιορισμούς λόγω της ευρείας και ταχύτατης επέκτασής του, περιορίζοντας έτσι το περιθώριο ανάπτυξης και εφαρμογής καινοτομιών. Τα "καθοριζόμενα από λογισμικό" δίκτυα (Software-Defined Networks) αποτελούν το μέλλον της δικτύωσης υπολογιστών. Η αρχιτεκτονική αυτή παρέχει στους ερευνητές μία ευκολότερη μέθοδο για τη δοκιμή νέων τεχνολογιών και πρωτοκόλλων. Το πιο σημαντικό όμως είναι πως μπορεί να αποτελέσει ένα κεντρικό υπόστρωμα δικτύωσης για τα Νέφη Υπολογιστών (Cloud Computing).

Σημαντικό ρόλο σε αυτήν την κατεύθυνση παίζει το πρωτόκολλο OpenFlow. Μέσω αυτού, μπορεί να επιτευχθεί ο διαχωρισμός μεταξύ του επιπέδου ελέγχου (control) και του επιπέδου προώθησης (forwarding) πακέτων σε ένα δίκτυο. Επιπροσθέτως, με τη χρήση του FlowVisor μπορεί να επιτευχθεί η εφαρμογή πολιτικών διαχωρισμού των δικτυακών πόρων σε απομονωμένα κομμάτια.

Για να μπορεί να χρησιμοποιηθεί ο FlowVisor σε πραγματικά παραγωγικά δίκτυα, είναι χρήσιμη η δημιουργία μιας διαδικτυακής εφαρμογής παρακολούθησης και διαχείρισής του. Η εφαρμογή αυτή αφενός θα καθιστά ευκολότερη τη χρήση του FlowVisor παρέχοντας ένα γραφικό περιβάλλον για τον χρήστη και αφετέρου θα βελτιώνει το κομμάτι της διαχείρισής του, υποστηρίζοντας πλέον πολλαπλούς χρήστες, καθώς και πολιτικές περιορισμού των χρηστών αυτών.

Abstract

Internet, as we know it until this day, appears to have limitations due to its wide and rapid growth, thus restricting the margin for developing and applying innovations. Software-Defined Networks constitute the future of networking. This architecture provides an easier approach for researchers to test new technologies and protocols. Most important, it can constitute a central networking substrate to Cloud Computing.

OpenFlow protocol holds a lead role in this direction. Through this protocol, we can achieve the separation between control layer and packet forwarding layer in a network. Moreover, using the FlowVisor we are able to apply policies in order to carve network resources into isolated slices.

In order to use FlowVisor in real production networks, it is useful to create a web application for monitoring and managing it. This application will allow for an easier use of FlowVisor by providing a graphical user interface and also, it will improve the management process by supporting multiple users, as well as restriction policies for these users.

Ευχαριστίες

Η διπλωματική εργασία αυτή, 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 ανάθεση της διπλωματικής εργασίας. Ακόμη, ευχαριστώ όλα τα μέλη του εργαστηρίου Netmode για τη βοήθειά τους, και ειδικότερα τον υπεύθυνο της διπλωματικής μου εργασίας Χρήστο Αργυρόπουλο, για την προθυμία του να με καθοδηγήσει όποτε αυτό χρειάστηκε.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου για την υποστήριξη που προσέφεραν και την κατανόηση που έδειξαν καθ' όλη τη διάρκεια των σπουδών μου.

Κωνσταντίνος

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κώδικες

1

Εισαγωγή

1.1 Αντικείμενο

Σκοπός της παρούσας διπλωματικής εργασίας είναι να γίνει περιγραφή της λειτουργίας του πρωτοκόλλου OpenFlow [1], καθώς και τα πλεονεκτήματα που μπορεί να προσφέρει η χρήση του. Η χρήση του πρωτοκόλλου αυτού διαδίδεται πλέον με ταχύτατους ρυθμούς και σημαντικό ρόλο σε αυτό παίζει η υποστήριξη που λαμβάνει αυτή τη στιγμή από μεγάλους κατασκευαστές δικτυακών συσκευών [2].

Πιο συγκεκριμένα αναλύεται η λειτουργία ενός OpenFlow μεταγωγέα και πώς μπορεί κάποιος να αποκτήσει ή να δημιουργήσει έναν τέτοιο. Κυρίως όμως η διπλωματική εργασία εστιάζεται στην διαχείριση, καθώς και στο ρόλο του Ελεγκτή στη διαχείριση του μεταγωγέα αυτού.

Όμως η χρησιμότητα οποιασδήποτε νέας τεχνολογίας καθορίζεται σε αρκετά μεγάλο βαθμό από τα εργαλεία και τις εφαρμογές που τη συνοδεύουν, που την υποστηρίζουν, και που την καθιστούν πιο χρήσιμη και παράλληλα, πιο εύχρηστη. Ένα τέτοιο εργαλείο-Ελεγκτής είναι ο FlowVisor, για τον οποίο αναλύεται ο τρόπος λειτουργίας του και οι δυνατότητες που κρύβει το συγκεκριμένο εργαλείο, καθώς και ο λόγος για τον οποίο η χρήση του αυξάνει σε σημαντικό βαθμό τις δυνατότητες ενός OpenFlow μεταγωγέα.

Με βάση τα παραπάνω, πρωταρχικός στόχος ήταν η βελτίωση της χρησιμότητας του FlowVisor δίνοντάς του τη δυνατότητα εφαρμογής πολιτικών για πολλαπλούς χρήστες, αλλά και της ευχρηστίας του. Έτσι δημιουργήθηκε στο πρακτικό σκέλος της παρούσας διπλωματικής εργασίας μία διεπαφή χρήστη μέσω του ιστού για τον FlowVisor, η οποία όχι

μόνο καθιστά πιο εύκολη τη χρήση του, αλλά προσθέτει μία νέα διάσταση στον τομέα της διαχείρισης του εργαλείου αυτού.

1.2 Οργάνωση

Η παρούσα διπλωματική εργασία μπορεί να διαχωριστεί σε δύο σκέλη. Το πρώτο σκέλος αποτελείται από την ανάλυση του πρωτοκόλλου OpenFlow καθώς και κάποιων βασικών εργαλείων. Το δεύτερο σκέλος αποτελεί και το πρακτικό κομμάτι της διπλωματικής, και περιγράφει τη δημιουργία της διεπαφής χρήστη, καθώς και τη συνεισφορά της στη χρήση του FlowVisor και γενικότερα του OpenFlow.

Πιο συγκεκριμένα, το πρώτο σκέλος ταυτίζεται με το Κεφάλαιο 2. Στο κεφάλαιο αυτό αναλύεται το πρωτόκολλο OpenFlow, ο τρόπος λειτουργίας του, ο διαχωρισμός των μεταγωγέων που χρησιμοποιούν την τεχνολογία OpenFlow, σε *OpenFlow-only* και *OpenFlow-enabled*, καθώς η μέθοδος που χρησιμοποιείται για τη λήψη αποφάσεων. Ακόμη, αναλύεται ο τρόπος λειτουργίας των Ελεγκτών και πιο συγκεκριμένα του Ελεγκτή NOX. Τέλος αναλύεται το εργαλείο-Ελεγκτής FlowVisor καθώς και οι δυνατότητές του.

Το δεύτερο σκέλος περιέχει τέσσερα κεφάλαια.

- Το Κεφάλαιο 3 ασχολείται με τις μεθόδους που χρησιμοποιήθηκαν για την ανάπτυξη της διεπαφής χρήστη. Γενικά, το κεφάλαιο αυτό επικεντρώνεται κατά κύριο λόγο στις εφαρμογές *fvctl* και *fvconfig* που συνοδεύουν τον FlowVisor, αλλά και στο Django το οποίο προσφέρει ένα πλαίσιο για την ανάπτυξη εφαρμογών ιστού.
- Το Κεφάλαιο 4 επικεντρώνεται στον τρόπο διασύνδεσης των στοιχείων που αναφέρθηκαν στο προηγούμενο κεφάλαιο, και πιο συγκεκριμένα στην ανάλυση των αρχείων κώδικα που δημιουργήθηκαν για την υλοποίηση της διεπαφής χρήστη του FlowVisor.
- Στο Κεφάλαιο 5 αναλύεται ο τρόπος εγκατάστασης και χρήσης της διεπαφής χρήστη. Αυτό πραγματοποιείται μέσω ενός παραδείγματος χρήσης της εφαρμογής ιστού που δημιουργήθηκε. Ακόμη, αναλύεται η προσφορά της εφαρμογής αυτής συγκρίνοντας τις ενέργειες στις οποίες καλείται να προβεί ο διαχειριστής ενός δικτύου, με ή χωρίς χρήση της συγκεκριμένης εφαρμογής.
- Τέλος στο Κεφάλαιο 6 συνοψίζονται τα πλεονεκτήματα που συνοδεύουν τη χρήση της διεπαφής χρήστη του FlowVisor, και προτείνονται τρόποι περαιτέρω βελτίωσής της.

2

Θεωρητικό Υπόβαθρο

Σήμερα, καθένας από τους κατασκευαστές συσκευών δικτύωσης υπολογιστών επιτρέπει ένα διαφορετικό βαθμό προγραμματισμού και ελέγχου των δρομολογητών και των μεταγωγέων του από τους διαχειριστές. Το γεγονός αυτό πολλές φορές οδηγεί σε περιορισμό της χρηστικότητας των συσκευών αυτών, καθώς και σε ανομοιογένεια στους μηχανισμούς διαχείρισης της δικτυακής κίνησης από συσκευές διαφορετικών κατασκευαστών. Ακόμη, η διαχείριση των δικτύων υπολογιστών απαιτεί την παραμετροποίηση, ξεχωριστά κάθε μίας συσκευής που χρησιμοποιείται.

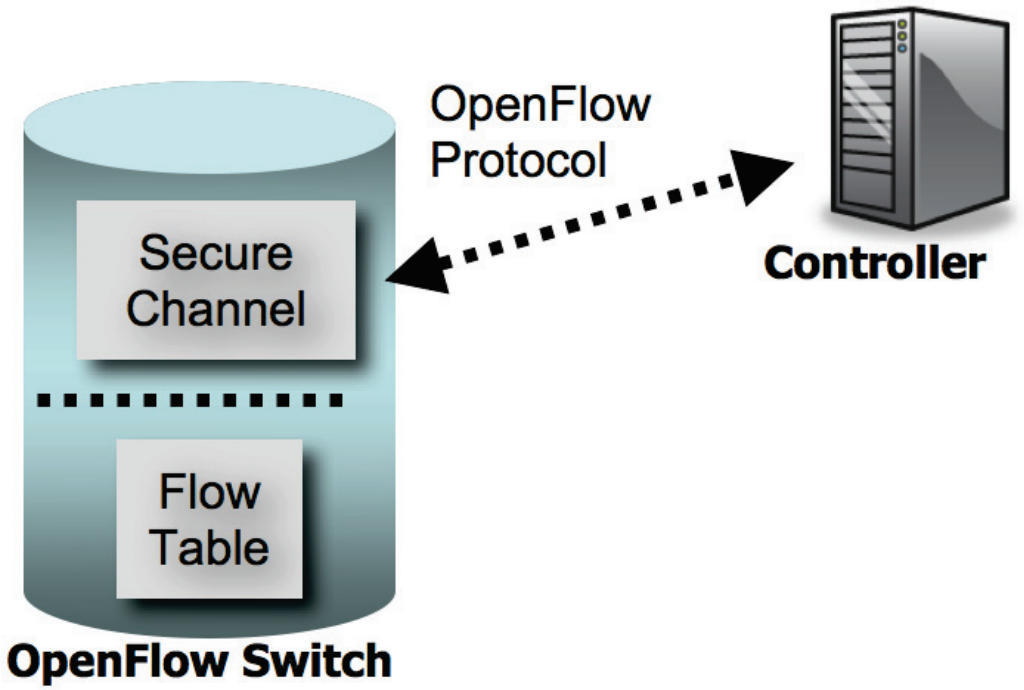
Η βασική ιδέα είναι ότι μπορούμε να εκμεταλλευτούμε το γεγονός πως οι περισσότεροι Ethernet μεταγωγείς (switches) πλέον περιέχουν πίνακες εγγραφών ροών δεδομένων[1] (flow-tables) ώστε να υλοποιούνται υπηρεσίες Network Address Translation (NAT), Quality of Service (QoS), Firewall κ.α. [3]. Το OpenFlow παρέχει ένα ελεύθερο πρωτόκολλο για τον προγραμματισμό αυτών των flow-tables. Κάθε OpenFlow switch ελέγχεται από έναν ερευνητή ή από τον διαχειριστή του δικτύου, μέσω ενός Ελεγκτή (Controller). Βασικό χαρακτηριστικό του Controller είναι το γεγονός ότι μπορεί να προσθέτει ή να αφαιρεί ροές δεδομένων (flows) στο flow-table του OpenFlow switch. Τέλος, με τη χρήση του FlowVisor μπορούμε να δημιουργήσουμε απομονωμένα τμήματα δικτυακών πόρων (slices), κάθε ένα από τα οποία θα ελέγχεται από έναν συγκεκριμένο Controller.

[1] Οι πίνακες αυτοί υλοποιούνται με τη χρήση πολλαπλών Ternary Content Addressable Memories (TCAMs), που περιέχουν access-lists για το φιλτράρισμα των πακέτων που λαμβάνει ένας τέτοιος μεταγωγέας με βάση τη διεύθυνση MAC, καθώς και QoS access-lists για την προτεραιότητα της δικτυακής κίνησης [4].

Με αυτές τις μεθόδους επιτρέπουμε στους ερευνητές να διεξάγουν πειράματα σε ετερογενή switches ή δρομολογητές (routers). Σημαντικό όμως είναι το γεγονός ότι αυτό επιτυγχάνεται χωρίς να απαιτείται από τους κατασκευαστές να εκθέσουν τις εσωτερικές λειτουργίες των προϊόντων τους. Στις επόμενες παραγράφους εξηγούνται αναλυτικά αυτές οι μέθοδοι, με βάση την έκδοση 1.0 του πρωτοκόλλου OpenFlow

2.1 Το Πρωτόκολλο OpenFlow

Ένα OpenFlow switch αποτελείται από ένα flow-table, το οποίο χρησιμοποιείται για την αντιστοίχιση και προώθηση των πακέτων, καθώς και έναν ασφαλή δίαυλο επικοινωνίας (secure channel) προς έναν Controller. Τέλος ο Controller διαχειρίζεται το OpenFlow switch μέσω του secure channel χρησιμοποιώντας το πρωτόκολλο OpenFlow (Σχήμα 2.1).



Σχήμα 2.1 : Ένα OpenFlow switch επικοινωνεί με έναν Controller μέσω ασφαλούς σύνδεσης (secure channel) χρησιμοποιώντας το πρωτόκολλο OpenFlow[Πηγή : OpenFlow Switch Specification Version 1.0.0]

Το flow-table περιέχει εγγραφές για τα flows, μετρητές (counters) και ενέργειες (actions) για την κάθε εγγραφή. Κάθε πακέτο που εισέρχεται στο OpenFlow switch

αντιπαραβάλλεται με τις εγγραφές του flow-table. Σε περίπτωση αντιστοίχισης με κάποια εγγραφή, εφαρμόζονται τα actions που συνοδεύουν τη συγκεκριμένη εγγραφή, αλλιώς το πακέτο προωθείται στον Controller μέσω του secure channel. Ο Controller είναι πλέον υπεύθυνος να λάβει μία απόφαση για τη διαδρομή που θα ακολουθήσει το πακέτο, προσθέτοντας ή αφαιρώντας εγγραφές από το flow-table του switch [5].

2.1.1 Flow Table

Κάθε εγγραφή του flow-table περιέχει πεδία κεφαλίδων (header fields) τα οποία αντιπαραβάλλονται με τα αντίστοιχα πεδία κάθε πακέτου που εισέρχεται στο OpenFlow switch. Ακόμη, περιέχει μετρητές που ανανεώνονται κάθε φορά που ένα πακέτο αντιστοιχίζεται με μία συγκεκριμένη εγγραφή. Τέλος περιέχει ενέργειες οι οποίες θα εφαρμοστούν σε περίπτωση αντιστοίχισης του πακέτου με κάποια εγγραφή (Πίνακας 2.1).

Πεδία Κεφαλίδων	Μετρητές	Ενέργειες

Πίνακας 2.1: Ένα σύνολο πεδίων κεφαλίδων, μετρητών και actions αποτελούν μια εγγραφή του flow-table

Στον Πινάκα 2.2 φαίνονται και τα δώδεκα πεδία κεφαλίδων, που μπορούν να αποθηκευτούν σε κάθε εγγραφή του flow-table. Αυτά τα δώδεκα πεδία θα συγκριθούν με τα αντίστοιχα πεδία κάθε πακέτου που εισέρχεται στο OpenFlow switch. Αξίζει να σημειωθεί πως κάθε πεδίο μπορεί να έχει είτε μια συγκεκριμένη τιμή, είτε την τιμή "ANY"[2] οπότε και η σύγκρισή του με το αντίστοιχο πεδίο ενός πακέτου θα είναι πάντα αληθής.

Ingress Port	Ether src	Ether dst	Ether type	VLAN id	VLAN Priority	IP src	IP dst	IP proto	IP ToS Bits	TCP/ UDP Src Port	TCP/ UDP Dst Port

Πίνακας 2.2: Πληροφορίες που αποτελούν κριτήριο για την αντιστοίχιση ενός πακέτου με ένα συγκεκριμένο flow και περιλαμβάνονται στις εγγραφές του flow-table

Σε κάθε OpenFlow switch διατηρούνται μετρητές για κάθε πίνακα, για τα flows, και για τις πόρτες ενός switch (switch port), καθώς και για τις ουρές αναμονής. Στον πίνακα 2.3 φαίνεται το σύνολο των μετρητών που διατηρούνται από ένα OpenFlow switch.

[2] Η τιμή "ANY" στο εξής θα αναφέρεται ως "wildcard"

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 2.3: Λίστα με τους διαθέσιμους μετρητές [Πηγή : OpenFlow Switch Specification Version 1.0.0]

Κάθε εγγραφή flow είναι συσχετισμένη με μία λίστα actions (η λίστα μπορεί να περιέχει από μηδέν έως οσαδήποτε actions). Τα actions που περιέχονται σε μία τέτοια εγγραφή υποδεικνύουν στο switch πώς να διαχειριστεί τα πακέτα που θα αντιστοιχιθούν με την εγγραφή αυτή. Αν δεν υπάρχει κάποιο action προώθησης (forward) στη λίστα, τότε το πακέτο απορρίπτεται (drop).

Ένα switch μπορεί να απορρίψει τη δημιουργία μίας καινούριας εγγραφής flow σε περίπτωση που δεν μπορεί να επεξεργαστεί τη λίστα actions που αυτή συμπεριλαμβάνει. Θεωρώντας αυτό ως δεδομένο, ένα OpenFlow switch δεν είναι απαραίτητο να υποστηρίζει όλα τα είδη actions που υποστηρίζει το OpenFlow πρωτόκολλο (πίνακας 2.4), αλλά πρέπει να υποστηρίζει τουλάχιστον αυτά με την ένδειξη "REQUIRED", τα οποία θεωρούνται

απαραίτητα για την υλοποίηση των βασικών λειτουργιών του. Όταν ένα switch συνδεθεί με τον Controller του, τότε τον ενημερώνει σχετικά με το ποια από τα προαιρετικά (ένδειξη "OPTIONAL" στον πίνακα 2.4) actions υποστηρίζει.

Το κύριο σύνολο actions ενός OpenFlow switch, με βάση το οποίο γίνεται και η προώθηση των πακέτων είναι το FORWARD. Κάθε switch πρέπει υποχρεωτικά να μπορεί να προωθήσει ένα πακέτο προς οποιοδήποτε φυσικό port, καθώς και προς τα παρακάτω εικονικά ports:

- ALL: Προώθηση ενός πακέτου προς όλες τις διεπαφές (interfaces) του switch, εκτός από την διεπαφή εισόδου.
- CONTROLLER: Αποστολή 128 bits (ή και ολόκληρου) του πακέτου προς τον Controller
- LOCAL: Αποστολή του πακέτου στο networking stack του ίδιου του switch
- TABLE: Πραγματοποιεί κάποιες ενέργειες στο flow-table του ίδιου του switch. Το action αυτό αφόρα μόνο packet-out μηνύματα
- IN_PORT: Προωθεί το πακέτο από την πόρτα εισόδου

Εκτός από αυτές τις πέντε εικονικές πόρτες, ένα switch μπορεί προαιρετικά να υποστηρίζει και τις επόμενες δύο:

- NORMAL: προώθηση πακέτων με τον “κανονικό” τρόπο προώθησης πακέτων (traditional forwarding path) που υποστηρίζει το εκάστοτε switch
- FLOOD: προώθηση του πακέτου με βάση το ελάχιστο Spanning Tree, χωρίς να συμπεριλαμβάνεται το port εισόδου του πακέτου

Γενικά, τα OpenFlow switch διακρίνονται στις παρακάτω δύο κατηγορίες:

- OpenFlow-only ή Dedicated OpenFlow switch: Ένα "άβουλο" datapath[3] το οποίο προωθεί πακέτα σύμφωνα με τις υποδείξεις του Controller που το ελέγχει.
- OpenFlow-enabled switch: Συγκεκριμένα εμπορικά switch, routers και access points που έχουν τροποποιηθεί με την προσθήκη του OpenFlow firmware. Το firmware αυτό αποτελείται από το Flow Table, το Secure Channel, καθώς και το πρωτοκόλλο OpenFlow.

[3] Ο όρος datapath αντιστοιχεί στην ουσία με τον όρο OpenFlow switch, σημειώνοντας ότι στην περίπτωση ενός OpenFlow-enabled switch, σαν datapath μπορούμε να ορίσουμε ένα συγκεκριμένο σύνολο των ports του switch αυτού.

Μία κρίσιμη διαφορά μεταξύ των δύο, είναι ότι τα Dedicated OpenFlow switch υποστηρίζουν μόνο τα FORWARD actions του πίνακα 2.4 με την ένδειξη REQUIRED. Αντίθετα, τα OpenFlow-enabled μπορεί να υποστηρίζουν και το FORWARD action *NORMAL*. Και τα δύο είδη όμως, μπορεί να υποστηρίζουν το action FLOOD.

Το σημαντικότερο ίσως σύνολο από τα "προαιρετικά" actions είναι το "Modify-Field" καθώς μπορεί να αλλάξει τις τιμές των κεφαλίδων ενός πακέτου, αυξάνοντας σημαντικά τη χρησιμότητα, και βελτιώνοντας τη λειτουργία του OpenFlow switch. Στον πίνακα 2.5 φαίνονται συνολικά τα "Modify-Field" actions.

Ενέργεια	**Λειτουργία**	**Αναγκαιότητα**
FORWARD	Προώθηση πακέτων προς οποιοδήποτε φυσικό port, καθώς και προς τα παρακάτω εικονικά: • ALL • CONTROLLER • LOCAL • TABLE • IN_PORT	*REQUIRED*
FORWARD	Προώθηση πακέτων τα παρακάτω εικονικά ports: • NORMAL: • FLOOD	*OPTIONAL*
ENQUEUE	Προώθηση του πακέτου στην ουρά ενός port	*OPTIONAL*
DROP	Μία εγγραφή flow χωρίς κανένα action ορισμένο, υποδεικνύει ότι το πακέτο πρέπει να απορριφθεί	*REQUIRED*
MODIFY-FIELD	Δίνει τη δυνατότητα σε ένα OpenFlow switch να μεταβάλλει τις τιμές των κεφαλίδων ενός πακέτου	*OPTIONAL*

Πίνακας 2.4: Σύνολο των actions που μπορεί να υποστηρίζονται από ένα OpenFlow switch

Action	Associated Data	Description
Set VLAN ID	12 bits	If no VLAN is present, a new header is added with the specified VLAN ID and priority of zero. If a VLAN header already exists, the VLAN ID is replaced with the specified value.
Set VLAN priority	3 bits	If no VLAN is present, a new header is added with the specified priority and a VLAN ID of zero. If a VLAN header already exists, the priority field is replaced with the specified value.
Strip VLAN header	-	Strip VLAN header if present.
Modify Ethernet source MAC address	48 bits: Value with which to replace existing source MAC address	Replace the existing Ethernet source MAC address with the new value
Modify Ethernet destination MAC address	48 bits: Value with which to replace existing destination MAC address	Replace the existing Ethernet destination MAC ad-dress with the new value.
Modify IPv4 source address	32 bits: Value with which to replace existing IPv4 source address	Replace the existing IP source address with new value and update the IP checksum (and TCP/UDP checksum if applicable). This action is only applica-ble to IPv4 packets.
Modify IPv4 destination address	32 bits: Value with which to replace existing IPv4 desti-nation address	Replace the existing IP des-tination address with new value and update the IP checksum (and TCP/UDP checksum if applicable). This action is only applied to IPv4 packets.
Modify IPv4 ToS bits	6 bits: Value with which to replace existing IPv4 ToS field	Replace the existing IP ToS field. This action is only ap-plied to IPv4 packets.
Modify transport source port	16 bits: Value with which to replace existing TCP or UDP source port	Replace the existing TCP/UDP source port with new value and update the TCP/UDP checksum. This action is only applicable to TCP and UDP packets.
Modify transport destination port	16 bits: Value with which to replace existing TCP or UDP destination port	Replace the existing TCP/UDP destination port with new value and update the TCP/UDP checksum This action is only applied to TCP and UDP packets.

Πίνακας 2.5 : Field-Modify actions [Πηγή : OpenFlow Switch Specification Version 1.0.0]

2.1.2 Secure Channel

Το Secure Channel είναι η διεπαφή εκείνη (interface) που συνδέει κάθε OpenFlow switch με έναν Controller. Μέσω αυτού του interface, ο Controller ρυθμίζει και διαχειρίζεται το switch, ενημερώνεται για γεγονότα (events) μέσω του switch και στέλνει πακέτα μέσα από αυτό. Το interface μπορεί να διαφέρει, ανάλογα με την υλοποίηση του OpenFlow switch, όμως κάθε μήνυμα που προορίζεται να σταλεί μέσω του secure-channel πρέπει να είναι τυποποιημένο σύμφωνα με το πρωτόκολλο OpenFlow.

2.1.3 Πρωτόκολλο OpenFlow

Γενικά, το πρωτόκολλο OpenFlow δημιουργήθηκε ώστε να παρέχει έναν τυποποιημένο τρόπο επικοινωνίας ενός OpenFlow switch με τον Controller που το ελέγχει. Το πρωτόκολλο αυτό υποστηρίζει τριών ειδών μηνύματα, Controller-to-switch, ασύγχρονα (asynchronous), και συμμετρικά (symmetric). Ακόμα, για κάθε είδος μηνύματος μπορούμε να διακρίνουμε πολλαπλές υποκατηγορίες, οι οποίες αναλύονται παρακάτω. Τα μηνύματα Controller-to-switch έχουν ως πηγή τον Controller και του δίνουν τη δυνατότητα να διαχειρίζεται απευθείας, ή να επιβλέπει, την κατάσταση ενός switch. Τα ασύγχρονα μηνύματα ξεκινούν από το switch και σκοπός τους είναι να ενημερώνουν τον Controller είτε για γεγονότα (events) που συμβαίνουν στο δίκτυο, είτε για αλλαγές στην κατάσταση του switch. Τέλος, τα συμμετρικά μηνύματα προέρχονται είτε από τον Controller είτε από το switch και αποστέλλονται χωρίς πριν να έχει υπάρξει κάποιο σχετικό αίτημα. Παρακάτω περιγράφονται αναλυτικότερα οι τρεις αυτές κατηγορίες μηνυμάτων.

2.1.3.1 Controller-to-switch

Τα μηνύματα του τύπου Controller-to-switch ξεκινούν από τον Controller προς το switch, χωρίς αυτό να υποχρεούται να αποστείλει κάποιο μήνυμα ως απάντηση. Τα μηνύματα αυτά διαχωρίζονται στις παρακάτω υποκατηγορίες:

i. **Features:** Από τη στιγμή έναρξης της συνεδρίας ανάμεσα στον Controller και το switch μέσω Transport Layer Security (TLS), ο Controller στέλνει ένα μήνυμα στο switch ζητώντας πληροφορίες για τις δυνατότητές του (features request), και περιμένει από εκείνο μία σχετική απάντηση (features reply).
ii. **Configuration:** Ο Controller μπορεί να παραμετροποιήσει τις ρυθμίσεις του switch που ελέγχει, ή να ζητήσει πληροφορίες για αυτές. Στην περίπτωση αυτή το switch είναι υποχρεωμένο να απαντήσει με σχετικό μήνυμα.

iii. **Modify-State:** Τα μηνύματα αυτά χρησιμοποιούνται από τον Controller κυρίως για προσθήκη, κατάργηση, ή τροποποίηση του flow-table που υπάρχει στο switch, ή για να ρυθμίσει τις ιδιότητες των ports του.

iv. **Read-State:** Χρησιμοποιούνται από τον Controller για να μαζέψει στατιστικά στοιχεία σχετικά με τον πίνακα των flows, τα ports του, καθώς και για κάθε εγγραφή flow ξεχωριστά.

v. **Send-Packet:** Τα μηνύματα send-packet χρησιμεύουν ώστε να υποδείξει ο Controller στο switch μέσω ποιου συγκεκριμένου port να προωθήσει ένα πακέτο.

vi. **Barrier:** Τα μηνύματα Barrier request/reply χρησιμοποιούνται ώστε να επιβεβαιώνει ο Controller ότι ισχύουν οι προϋποθέσεις για κάποιο συγκεκριμένο μήνυμα. Ακόμη χρησιμοποιούνται για να πληροφορηθεί ο Controller για την ολοκλήρωση κάποιας διεργασίας.

2.1.3.2 Ασύγχρονα (asynchronous)

Τα ασύγχρονα μηνύματα στέλνονται από το switch, χωρίς πρώτα να έχει υπάρξει σχετικό αίτημα από τον Controller. Σκοπός τους είναι να τον ενημερώσουν για αφίξεις πακέτων, για αλλαγές στην κατάσταση του switch, ή για κάποιο σφάλμα που έχει προκύψει. Οι τέσσερις βασικές υποκατηγορίες ασύγχρονων μηνυμάτων είναι:

i. **Packet-in:** Κάθε νέο πακέτο που εισέρχεται στο switch και δεν αντιστοιχίζεται με καμία από τις υπάρχουσες εγγραφές flow, προκαλεί την δημιουργία και αποστολή ενός μηνύματος Packet-in προς τον Controller (packet-in event). Αν το switch έχει αρκετή διαθέσιμη μνήμη ώστε να αποθηκεύσει προσωρινά (buffer) το πακέτο αυτό, τότε το μήνυμα που θα σταλεί θα περιλαμβάνει 128 bytes με τις απαραίτητες πληροφορίες που χρειάζεται ο Controller. Οι πληροφορίες αυτές αφορούν τις τιμές των κεφαλίδων του πακέτου που εισήλθε, καθώς και ένα μία τιμή αναγνώρισης (buffer ID) του πακέτου αυτού. Σε περίπτωση που το switch δεν υποστηρίζει την προσωρινή αποθήκευση πακέτων, ή δεν έχει αρκετή διαθέσιμη μνήμη, τότε το μήνυμα που θα αποσταλεί στον Controller θα περιλαμβάνει ολόκληρο το αρχικό πακέτο.

ii. **Flow-removed:** Όταν μία εγγραφή flow προστεθεί στο switch από τον Controller μέσω ενός flow-modify μηνύματος, υπαγορεύεται στο switch μετά από πόσο χρόνο αδράνειας πρέπει να σβήσει την εγγραφή αυτή. Ακόμη υπογορεύεται το πότε πρέπει να την σβήσει γενικώς, ανεξαρτήτως της δραστηριότητας που σχετίζεται με την συγκεκριμένη εγγραφή. Ταυτόχρονα υπαγορεύεται στο switch αν θα πρέπει να ενημερώσει τον Controller μετά από μια τέτοια διαγραφή, πράγμα το οποίο γίνεται με ένα μήνυμα τύπου flow-removed.

iii. **Port-status:** Το switch χρησιμοποιεί αυτά τα μηνύματα σε περιπτώσεις αλλαγής της κατάστασης ενός port, όπως για παράδειγμα σε περίπτωση που ένας χρήστης του switch απενεργοποιήσει ένα συγκεκριμένο port. Επιπροσθέτως, χρησιμοποιείται και σε περιπτώσεις αλλαγής της κατάστασης ενός port όπως αυτή ορίζεται από το πρωτόκολλο 802.1D.

iv. **Error:** Με τα μηνύματα αυτά, το switch μπορεί να ενημερώσει τον Controller για προβλήματα, ή σφάλματα που μπορεί να προκύψουν.

2.1.3.3 Συμμετρικά (symmetric)

Τα συμμετρικά μηνύματα μπορεί να αποστέλλονται είτε από ένα switch, είτε από έναν Controller, χωρίς η άλλη πλευρά να έχει ζητήσει μια τέτοια ενέργεια, και διαχωρίζονται στις παρακάτω τρεις κατηγορίες:

i. **Hello:** Μηνύματα αυτού του τύπου ανταλλάσσονται μεταξύ του switch και του Controller την στιγμή που θα ολοκληρωθεί η μεταξύ τους σύνδεσης.

ii. **Echo:** Μηνύματα του τύπου echo request/reply μπορεί να αποστείλει οποιαδήποτε από τις δύο πλευρές και χρησιμοποιείται για μετρήσεις καθυστέρησης (latency) ή φάσματος συχνότητας (bandwidth). Ακόμη, χρησιμοποιείται για να επιβεβαιωθεί αν η μεταξύ τους σύνδεση είναι ενεργή.

iii. **Vendor:** Ο σκοπός αυτών των μηνυμάτων είναι να παρέχουν ένα περιθώριο για περαιτέρω λειτουργικότητα, όσων αφορά τους τύπους των OpenFlow μηνυμάτων. Υλοποιήθηκαν κυρίως για στοιχεία μελλοντικών εκδόσεων του OpenFlow.

2.1.4 Σύνδεση switch - Controller και Κρυπτογράφηση

Το switch πρέπει να μπορεί να εγκαθιδρύσει την επικοινωνία με τον Controller, μέσω μιας διεύθυνσης IP και ενός port, τα οποία καθορίζονται από τον χρήστη του switch και παραμένουν σταθερά. Η κίνηση μέσω του secure channel δεν αντιπαραβάλλεται με τις εγγραφές του πίνακα των flows, και για αυτόν το λόγο το switch πρέπει να αναγνωρίζει την υπόλοιπη δικτυακή κίνηση ως τοπική (local), προκειμένου να την συγκρίνει με τις εγγραφές του flow-table.

Η επικοινωνία μεταξύ switch και Controller πραγματοποιείται μέσω μίας TLS σύνδεσης. Τη σύνδεση αυτή εκκινεί το switch προς τον Controller, ο οποίος αναμένει για αυτή συνήθως στο TCP port 6633. Στη συνέχεια, πραγματοποιείται η μεταξύ τους πιστοποίηση (authentication) μέσω ανταλλαγής πιστοποιητικών με ιδιωτικό κλειδί. Για το

λόγο αυτό, κάθε switch πρέπει να έχει ένα πιστοποιητικό για την πιστοποίηση του Controller και άλλο ένα για την δική του πιστοποίηση προς τον Controller.

Αν μετά την πάροδο κάποιου χρονικού διαστήματος η σύνδεση διακοπεί, τότε το switch προσπαθεί να συνδεθεί με τον ίδιο, ή εφεδρικούς Controllers οι οποίοι έχουν οριστεί. Σε περίπτωση που η σύνδεση αυτή αποτύχει μετά από έναν καθορισμένο αριθμό προσπαθειών, τότε το switch μπαίνει αυτομάτως σε κατάσταση ασφαλείας (emergency state) και επαναφέρει τον πίνακα των flows στην αρχική του κατάσταση (reset). Από τη στιγμή εκείνη, και μέχρι να επιτύχει ξανά τη σύνδεσή του με έναν Controller, η διαδικασία αντιστοίχισης και προώθησης των πακέτων που εισέρχονται στο switch, υπαγορεύεται από ένα flow table ασφαλείας (emergency), ο οποίος θα έχει καθοριστεί από τον χρήστη. Έτσι, μόλις ανακτηθεί η σύνδεση με κάποιον Controller υπάρχει η δυνατότητα να διατηρηθούν ή να καταργηθούν αυτές οι εγγραφές ασφαλείας.

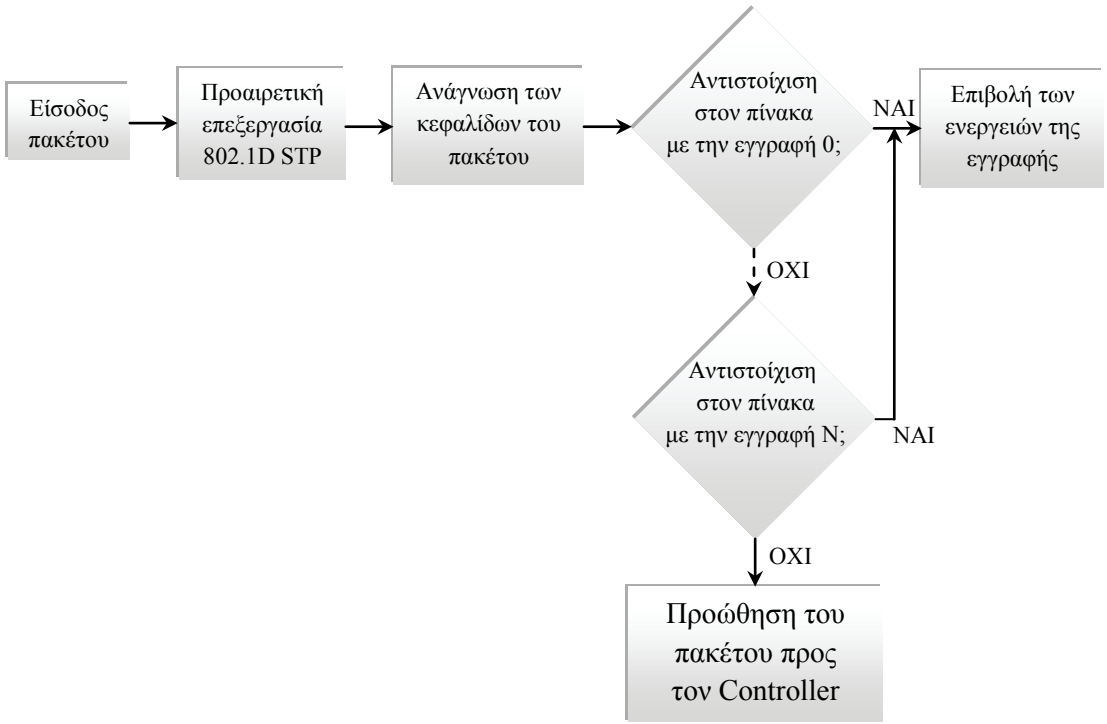
2.1.5 Αντιστοίχιση Πακέτων - Εγγραφών Flow

Για κάθε νέο πακέτο που θα λάβει, το OpenFlow switch εκτελεί τη διαδικασία που φαίνεται στην Σχήμα 2.2, ώστε να αποφασίσει πώς θα το διαχειριστεί και πού θα το προωθήσει. Βέβαια, η διαδικασία ελέγχου/σύγκρισης των κεφαλίδων του νέου πακέτου (Σχήμα 2.3) εξαρτάται από τον τύπο του πακέτου, όπως περιγράφεται παρακάτω.

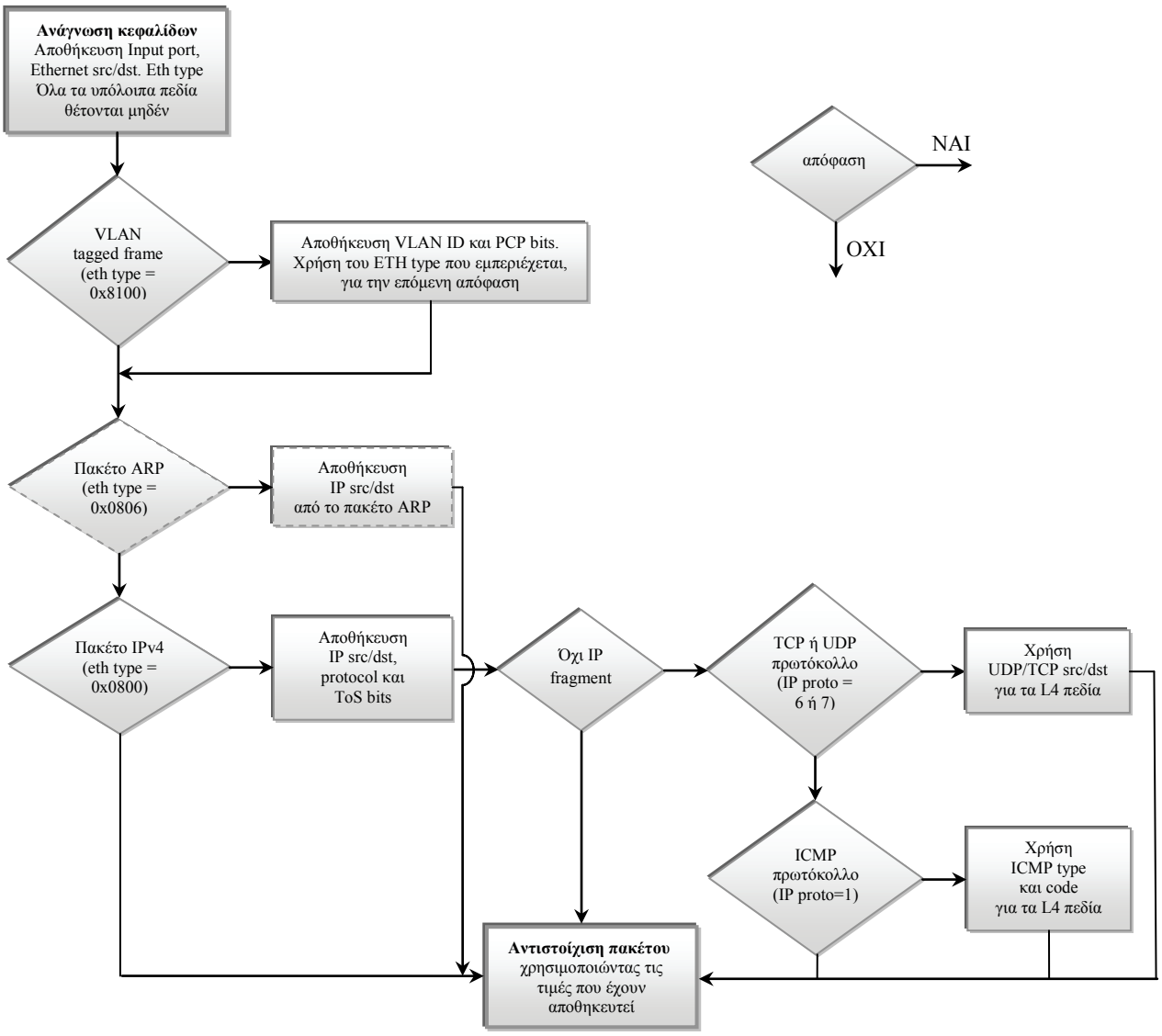
- Κανόνες που χαρακτηρίζονται από ένα συγκεκριμένο port εισόδου, αντιπαραβάλλονται με το φυσικό port που δέχτηκε το πακέτο.
- Οι κεφαλίδες Ethernet, όπως φαίνονται στον πίνακα 2.2, χρησιμοποιούνται για όλα τα πακέτα.
- Αν το πακέτο έχει ετικέτα VLAN (είναι δηλαδή τύπου 0x8100), τότε τα πεδία VLAN ID και PCP λαμβάνουν μέρος στην αντιστοίχιση.
- Για πακέτα ARP (Ethernet type 0x8606), στην διαδικασία αντιστοίχισης μπορεί να χρησιμοποιούνται και οι διευθύνσεις IP πηγής και προορισμού.
- Για πακέτα IP (Ethernet type 0x8000), στα πεδία αντιστοίχισης περιλαμβάνονται και αυτά της IP κεφαλίδας.
- Για πακέτα IP που χρησιμοποιούν πρωτόκολλο TCP ή UDP (IP πρωτόκολλο 6 ή 17), στην αντιστοίχιση χρησιμοποιούνται και οι πόρτες μεταφοράς (transport ports).
- Για IP πακέτα που χρησιμοποιούν πρωτόκολλο ICMP (IP πρωτόκολλο 1), στην αντιστοίχιση χρησιμοποιούνται και τα πεδία Type και Code.
- Για πακέτα IP με μη μηδενική μετατόπιση κατακερματισμού (fragment offset), ή More Fragments bit set, τα transport ports θεωρούνται ως μηδενικά για τη διαδικασία της αντιστοίχισης.

Στην περίπτωση που οι τιμές των κεφαλίδων ενός πακέτου ταιριάζουν με αυτές που ορίζονται σε κάποια εγγραφή flow, τότε το πακέτο αντιστοιχίζεται με τη συγκεκριμένη εγγραφή. Στη συνέχεια, αν το πακέτο μετά την αντιπαραβολή του με τις εγγραφές του flow-table αντιστοιχηθεί με κάποια, τότε ανανεώνονται και οι μετρητές που περιλαμβάνει η συγκεκριμένη εγγραφή. Αν το πακέτο δεν αντιστοιχηθεί με καμία εγγραφή, τότε αυτό προωθείται προς τον Controller μέσω του secure channel.

Τέλος, είναι σημαντικό να αναφερθεί, ότι τα πακέτα αντιστοιχίζονται με τις εγγραφές flow με βάση κάποια προτεραιότητα. Κάθε πακέτο μπορεί να αντιστοιχηθεί με πολλαπλές εγγραφές, όμως η σειρά με την οποία θα γίνει η αντιστοίχιση υπαγορεύεται από την προτεραιότητα της εγγραφής. Έτσι, μια εγγραφή που ταιριάζει επακριβώς με τις κεφαλίδες ενός πακέτου (δηλαδή για παράδειγμα δεν περιέχει wildcards), έχει πάντα τη μέγιστη προτεραιότητα. Αν για ένα πακέτο, συμβαίνει πολλαπλές εγγραφές του πίνακα των flows να αντιστοιχίζονται με αυτό και να έχουν την ίδια προτεραιότητα , τότε το switch είναι ελεύθερο να επιλέξει οποιαδήποτε σειρά για την αντιστοίχιση του πακέτου.



Σχήμα 2.2 : Ροή πακέτων σε ένα OpenFlow switch, σύμφωνα με την έκδοση 1.0 του πρωτοκόλλου OpenFlow



Σχήμα 2.3 : Διαδικασία αντιστοίχισης πακέτου με τις εγγραφές του πίνακα των Flows

2.2 Controller

Μέχρι πρόσφατα, η διαχείριση των δικτύων υπολογιστών πραγματοποιούνταν αποκλειστικά μέσω ρυθμίσεων χαμηλού επιπέδου, ξεχωριστά για το κάθε στοιχείο του δικτύου (π.χ. switch, router κ.α.). Η πρακτική αυτή εξαρτιόνταν άμεσα από την φυσική τοπολογία του κάθε δικτύου. Πλέον μέσω του διαχωρισμού του Data Plane[4] από το Control Plane[5] που μας προσφέρει το πρωτόκολλο OpenFlow μπορούμε να μιλήσουμε για ένα Λειτουργικό Σύστημα Δικτύων Υπολογιστών το οποίο παρέχει ένα ενιαίο και κεντρικό προγραμματιστικό περιβάλλον για τον έλεγχο ενός ολόκληρου δικτύου [6].

Ένα τέτοιο λειτουργικό σύστημα δεν διαχειρίζεται το δίκτυο καθ' αυτό, αλλά παρέχει τη δυνατότητα να παρακολουθούμε και να ελέγχουμε τις διεργασίες που γίνονται, μέσω του προγραμματιστικού του περιβάλλοντος. Έτσι η διαχείριση του δικτύου μπορεί πλέον να γίνει από εφαρμογές που έχουν δημιουργηθεί και "τρέχουν" στο λειτουργικό σύστημα. Παρουσιάζονται λοιπόν δύο καινοτομίες στον τομέα της διαχείρισης δικτύων που αλλάζουν τον τρόπο που αυτή γινόταν μέχρι στιγμής. Πρώτον το σύστημα αυτό υποστηρίζει εφαρμογές μέσω ενός κεντρικού προγραμματιστικού μοντέλου, οι οποίες μπορούν να διαχειριστούν ένα ολόκληρο δίκτυο υπολογιστών. Δεύτερον, οι εφαρμογές αυτές μπορούν να γραφούν στα πλαίσια μιας γενίκευσης υψηλότερου επιπέδου (π.χ. χρήστες και ονόματα host) και όχι στα πλαίσια παραμετροποιήσεων χαμηλού επιπέδου (π.χ. διευθύνσεις IP ή MAC). Το μόνο που χρειάζεται, είναι το λειτουργικό σύστημα αυτό να διατηρεί τις αντιστοιχήσεις μεταξύ των δύο αυτών επιπέδων.

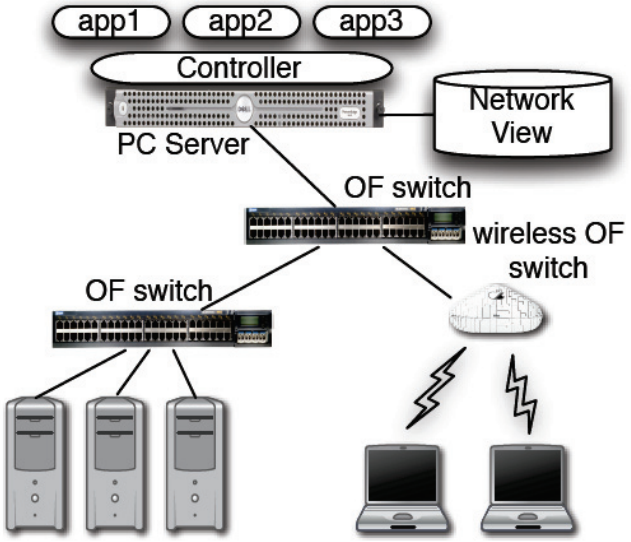
Έτσι δημιουργήθηκε η έννοια του Controller, ο οποίος δεν είναι τίποτα άλλο από ένα Λειτουργικό Σύστημα Δικτύων Υπολογιστών, υπεύθυνο για τη διαχείρισή τους. Όπως έχει ήδη αναφερθεί, κάθε OpenFlow switch διαχειρίζεται από έναν Controller. Ο Controller είναι υπεύθυνος για τη λήψη αποφάσεων σχετικών με την κατάσταση του switch καθώς και με την προώθηση νέων πακέτων που εισέρχονται σε αυτό. Έτσι ένας Controller ο οποίος μπορεί να λειτουργεί σε έναν απλό Ηλεκτρονικό Υπολογιστή έχει τη δυνατότητα να διαχειρίζεται πολλαπλά OpenFlow switches καθώς και να έχει μία οπτική ολόκληρου του δικτύου που ελέγχει (Σχήμα 2.4).

Υπάρχει αυτή τη στιγμή ένας περιορισμένος αλλά συνεχώς αυξανόμενος αριθμός Controllers που μπορούμε να χρησιμοποιήσουμε, αλλά εδώ θα επικεντρωθούμε στον Controller NOX [7], που είναι και ο πιο διαδεδομένος.

---

[4] Το Data Plane (ή αλλιώς Forwarding Plane) είναι υπεύθυνο για την ανα-πακέτο επεξεργασία η οποία είναι απαραίτητη για την προώθηση των πακέτων

[5] Το Control Plane αναλαμβάνει τις πολύπλοκες υπολογιστικές διαδικασίες που είναι συνηθισμένες στα πρωτόκολλα ελέγχου και σηματοδοσίας [8].



Σχήμα 2.4 : Ένας Controller διαχειρίζεται πολλαπλά OpenFlow switches [Πηγή : "NOX: Towards an Operating System for Networks"]

2.2.1 Γενική επισκόπηση του Nox Controller

Ο NOX Controller αποτελεί μία πλατφόρμα ελέγχου ενός δικτύου υπολογιστών και παρέχει ένα προγραμματιστικό περιβάλλον υψηλού επιπέδου. Με αυτό τον τρόπο μπορούν να δημιουργηθούν εφαρμογές (applications) που θα χρησιμοποιούνται από τον NOX ώστε να ληφθούν αποφάσεις για τη διαχείριση και παρακολούθηση του δικτύου. Ο κύριος σκοπός αυτών των εφαρμογών είναι να αποφασίσουν πώς και αν θα δρομολογηθεί κάθε πακέτο σε ένα δίκτυο υπολογιστών, κάτι το οποίο επιτυγχάνεται με τη χρήση των flows.

Αναλύοντας το δίκτυο σε επίπεδο flows μπορούμε, εφόσον έχει ληφθεί κάποια απόφαση για ένα πακέτο, τα πακέτα που θα ακολουθήσουν και θα έχουν τις ίδιες τιμές κεφαλίδων, να αντιμετωπιστούν με τον ίδιο τρόπο χωρίς να χρειάζεται πλέον η παρέμβαση του Controller. Έτσι ο NOX μπορεί να διαχειρίζεται από ένα μικρό δίκτυο μερικών hosts, μέχρι ένα μεγάλο δίκτυο εκατοντάδων switches, παρέχοντας και στις δύο περιπτώσεις έναν εύκολα τροποποιήσιμο τρόπο ελέγχου των δικτύων αυτών.

Το μόνο που απαιτεί ο NOX για την λειτουργία του είναι να έχει συνδεθεί με τουλάχιστον ένα OpenFlow switch, χρησιμοποιώντας το πρωτόκολλο OpenFlow. Όπως έχει ήδη αναφερθεί σε προηγούμενη παράγραφο, στην περίπτωση που κάποιο νέο πακέτο εισέλθει στο switch και δεν συσχετιστεί με κάποια από τις υπάρχουσες εγγραφές του πίνακα των flows, τότε θα αποσταλεί στον NOX. Από αυτό το σημείο και μετά, είναι υπεύθυνες οι

εφαρμογές που τρέχουν στον NOX για να αποφασίσουν τη διαδρομή που θα ακολουθήσει. Ένα τέτοιο πακέτο συνήθως αποτελεί την αιτία για την δημιουργία ενός νέου flow (flow-initiation). Όμως μπορεί μέσω των εφαρμογών που "τρέχουν" στον NOX, να αποφασιστεί να ληφθούν όλα τα πακέτα που αντιστοιχούν για παράδειγμα στο ίδιο πρωτόκολλο, πράγμα που σημαίνει ότι δεν θα δημιουργηθεί ποτέ ένα νέο flow. Γενικά ο NOX χρησιμοποιεί τις εισαγωγές νέων flows καθώς και όλη την υπόλοιπη δικτυακή κίνηση ώστε να ενημερώνεται κάθε στιγμή για την κατάσταση του δικτύου, καθώς και για να αποφασίσει αν και από ποια διαδρομή θα προωθήσει την κίνηση.

Αυτή τη στιγμή ο NOX μπορεί να λειτουργεί σε περιβάλλον user-space σε ένα συνηθισμένο PC ή Server και μπορεί να δημιουργεί περίπου 100.000 νέα flows ανά δευτερόλεπτο, υπεραρκετά δηλαδή ώστε να διαχειρίζεται την κίνηση μιας ολόκληρης πανεπιστημιακής κοινότητας. Οι εφαρμογές του NOX δημιουργούνται με χρήση είτε της γλώσσας Python είτε της C++, και φορτώνονται δυναμικά, ενώ η βασική υποδομή του, καθώς και συναρτήσεις που θεωρούνται κρίσιμες για την ταχύτητά του έχουν υλοποιηθεί σε C++.

2.2.2 Το προγραμματιστικό περιβάλλον του NOX

Το προγραμματιστικό περιβάλλον του NOX είναι κατά βάση αρκετά απλό, αφού περιστρέφεται γύρω από γεγονότα (events), την κατάσταση του δικτύου, καθώς και την αντιστοίχιση "ονομάτων" υψηλού επιπέδου με λειτουργίες χαμηλού επιπέδου.

2.2.2.1 Εφαρμογές

Ο NOX στον πυρήνα του, παρέχει μόνο μεθόδους χαμηλού επιπέδου για την αλληλεπίδρασή του με το υπόλοιπο δίκτυο. Όλες οι μέθοδοι υψηλού επιπέδου και τα γεγονότα, δημιουργούνται και υποστηρίζονται από τις εφαρμογές του (applications). Στην πραγματικότητα, μία εφαρμογή δεν είναι τίποτα άλλο παρά ένα άθροισμα μεθόδων. Επιπροσθέτως όμως, έχει τη δυνατότητα ορισμού συγκεκριμένων μεθόδων που μπορεί να χρησιμοποιηθούν από μία άλλη εφαρμογή. Έτσι για παράδειγμα η διαδικασία της δρομολόγησης (routing) ενός πακέτου μπορεί να υλοποιηθεί σε μία εφαρμογή και κάποια άλλη εφαρμογή που πιθανόν να χρειάζεται τη διαδικασία αυτή, μπορεί να τη δηλώσει σαν εξάρτηση και να τη χρησιμοποιήσει ελεύθερα. Στον Πίνακα 2.6 που ακολουθεί, φαίνονται κάποιες από τις βασικές εφαρμογές που παρέχει ο NOX. Όπως φαίνεται και στον πίνακα, οι εφαρμογές κατά βάση χωρίζονται σε τρεις βασικές κατηγορίες ανάλογα με το κομμάτι διαχείρισης του δικτύου το οποίο αφορούν. Οι κατηγορίες αυτές είναι:

i. **Core apps:** Παρέχουν ένα σύνολο λειτουργιών που χρησιμοποιούνται από τις άλλες δύο κατηγορίες εφαρμογών για τη διαχείριση του δικτύου.
ii. **Network apps:** Είναι οι εφαρμογές που διαχειρίζονται στην πραγματικότητα το δίκτυο και είναι υπεύθυνες για τη λήψη αποφάσεων.
iii. **Web apps:** Χρησιμοποιούνται από τον NOX για την παροχή υπηρεσιών διαδικτύου (Web services) στους χρήστες του.

ΕΦΑΡΜΟΓΗ	ΛΕΙΤΟΥΡΓΙΑ	ΚΑΤΗΓΟΡΙΑ
messenger	Παρέχει TCP/SSL server sockets για την επικοινωνία με άλλες συσκευές όπως οι hosts	*Core Apps*
snmp	Διαχειρίζεται snmp traps χρησιμοποιώντας ένα Python script ως trap handler μέσω NetSNMP	*Core Apps*
Packetdump	Λειτουργεί όπως το γνωστό tcpdump, εμφανίζοντας πληροφορίες για την κίνηση που διέρχεται μέσω του NOX	*Core Apps*
Discovery	Μαθαίνει και αποθηκεύει τους συνδέσμους (links) μεταξύ των switch που ελέγχει ο NOX, με τη χρήση LLDP πακέτων. Ακόμα μπορεί να δημιουργήσει Link_event.	*Network apps*
Topology	Κρατάει τους συνδέσμους που είναι ενεργοί τη δεδομένη στιγμή χωρίς όμως να μπορεί να δημιουργήσει events. Παρέχει όμως μεθόδους για χρήση σε άλλες εφαρμογές	*Network Apps*
Authenticator	Αποθηκεύει στοιχεία σχετικά με τη θέση των switches και hosts του δικτύου	*Network apps*
Routing	Είναι υπεύθυνο για τον υπολογισμό της διαδρομής (shortest-path) ανάμεσα σε κάθε ζευγάρι συνδεδεμένων switches	*Network apps*
Switch	Layer 2 switching σε όλο το δίκτυο	*Network apps*
Webservice	Παρέχει τη διεπαφή για τις υπηρεσίες web των εφαρμογών	*Web apps*
Webserver	Φιλοξενεί τη διεπαφή ελέγχου	*Web apps*

Πίνακας 2.6 : Κάποιες από τις βασικές εφαρμογές που παρέχει ο NOX

2.2.2.2 Γεγονότα

Τα δίκτυα μεγάλων επιχειρήσεων ή πανεπιστημιακών κοινοτήτων δεν είναι στατικά αφού συνεχώς υπάρχει η ανάγκη δημιουργίας ή διαγραφής flows, η ανάγκη προσθήκης και αφαίρεσης χρηστών, καθώς και σύνδεσμοι (links) των οποίων η κατάσταση μπορεί να μεταβάλλεται (up/down). Σαν γεγονός (event) χαρακτηρίζεται κάθε τι που θα συμβεί στο δίκτυο που διαχειρίζεται ο NOX, και που μπορεί να έχει ενδιαφέρον για κάποια από τις εφαρμογές του. Για να μπορέσουν οι εφαρμογές του NOX να αντιμετωπίσουν αυτή την πληθώρα γεγονότων και πληροφοριών, χρησιμοποιούν ένα σύνολο "διαχειριστών γεγονότων" (event handlers). Τα διάφορα γεγονότα συσχετίζονται με τους event handlers, ώστε σε κάθε περίπτωση να εκτελείται κάποια ενέργεια. Οι event-handlers χρησιμοποιούνται σύμφωνα με τη σειρά που έχουν δηλωθεί κατά την εκκίνηση του NOX, και η τιμή που θα επιστρέψουν υποδεικνύει αν οι event-handlers που έπονται θα συνεχίσουν την επεξεργασία του γεγονότος ή όχι. Κάποια γεγονότα παράγονται απευθείας μέσω συγκεκριμένων OpenFlow μηνυμάτων που μπορεί να λάβει ο NOX, ενώ άλλα γεγονότα μπορεί να παραχθούν από τις ίδιες τις εφαρμογές, σαν αποτέλεσμα συγκεκριμένων γεγονότων, όπως η σύνδεση ή αποσύνδεση ενός switch και η λήψη πακέτου ή στατιστικών μετρήσεων από το switch. Στους Πίνακες 2.7 και 2.8 φαίνονται κάποια από τα βασικά γεγονότα που μπορούν να χρησιμοποιηθούν από τις εφαρμογές του NOX. Αξίζει δε να σημειωθεί ότι ο διαχειριστής του NOX έχει τη δυνατότητα να δημιουργήσει πρόσθετες εφαρμογές καθώς και γεγονότα, συνοδευόμενα από τους αντίστοιχους event handlers, ώστε να διαχειρίζεται ο NOX το δίκτυο με τον επιθυμητό τρόπο.

Αν λοιπόν παρατηρήσουμε τις εφαρμογές του NOX από μια πιο γενική οπτική γωνία, δεν είναι τίποτα άλλο από ένα σύνολο event handlers. Έτσι τα γεγονότα είναι αυτά που καθορίζουν όλη τη λειτουργία του NOX.

ΓΕΓΟΝΟΣ	**ΑΙΤΙΑ ΔΗΜΙΟΥΡΓΙΑΣ**
Datapath_join_event	Σύνδεση νέου switch στο δίκτυο
Datapath_leave_event	Αποσύνδεση ενός switch από το δίκτυο
Packet_in_event	Παραλαβή νέου πακέτου από τον NOX
Flow_mod_event	Προσθήκη ή τροποποίηση ενός Flow από τον NOX
Flow_removed_event	Διαγραφή ή λήξη ενός flow
Port_status_event	Αλλαγή στην κατάσταση ενός port
Port_stats_in	Παραλαβή ενός Port_stats μηνύματος από τον Controller

Πίνακας 2.7 : Γεγονότα λόγω OpenFlow μηνυμάτων που λαμβάνονται από τον NOX

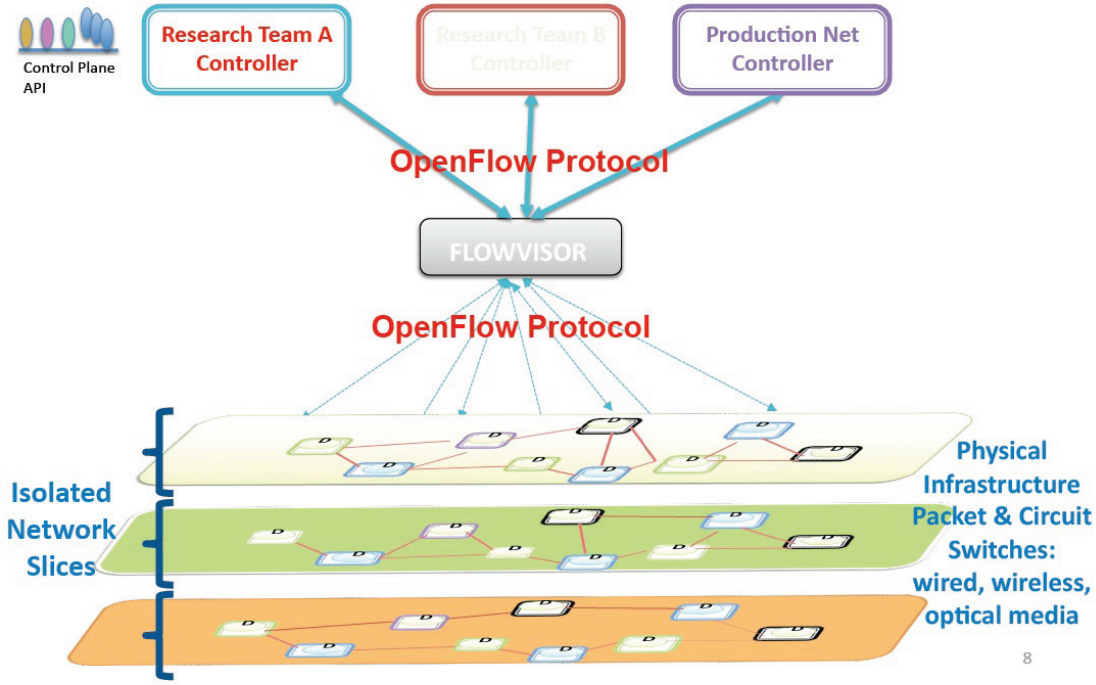
ΓΕΓΟΝΟΣ	ΑΙΤΙΑ ΔΗΜΙΟΥΓΙΑΣ
Host_event	Δημιουργείται από την εφαρμογή Authenticator κάθε φορά ένας καινούριος host συνδέεται στο δίκτυο
Flow_in_event	Δημιουργείται από την εφαρμογή Authenticator κάθε φορά που ο Controller λαμβάνει ένα Packet_in_event
Link_event	Δημιουργείται από την εφαρμογή Discovery για κάθε προσθήκη ή αλλαγή ενός συνδέσμου (link) στο δίκτυο

Πίνακας 2.8 : Γεγονότα που δημιουργούνται από εφαρμογές του NOX

2.3 FlowVisor

Ένα σημαντικό πρόβλημα της έρευνας πάνω στα δίκτυα υπολογιστών αποτελεί η τεκμηρίωση. Οι ερευνητές, στην προσπάθειά τους να αξιολογήσουν κάτι νέο, συναντούν συνήθως το πρόβλημα του ρεαλισμού και του κόστους. Και αυτό γιατί η δημιουργία ενός ρεαλιστικού περιβάλλοντος δοκιμών (testbed) θα στηριχτεί σε εμπορικά switches ή routers τα οποία αφενός έχουν "κλειστές" τις εσωτερικές τους λειτουργίες από τον κατασκευαστή τους και αφετέρου κοστίζουν. Ακόμη, η αξιολόγηση με χρήση software switches, αν και θα περιορίσει σημαντικά τα προηγούμενα προβλήματα, μειονεκτεί αφού η λειτουργία τους είναι αρκετές τάξεις μεγέθους πιο αργή [9].

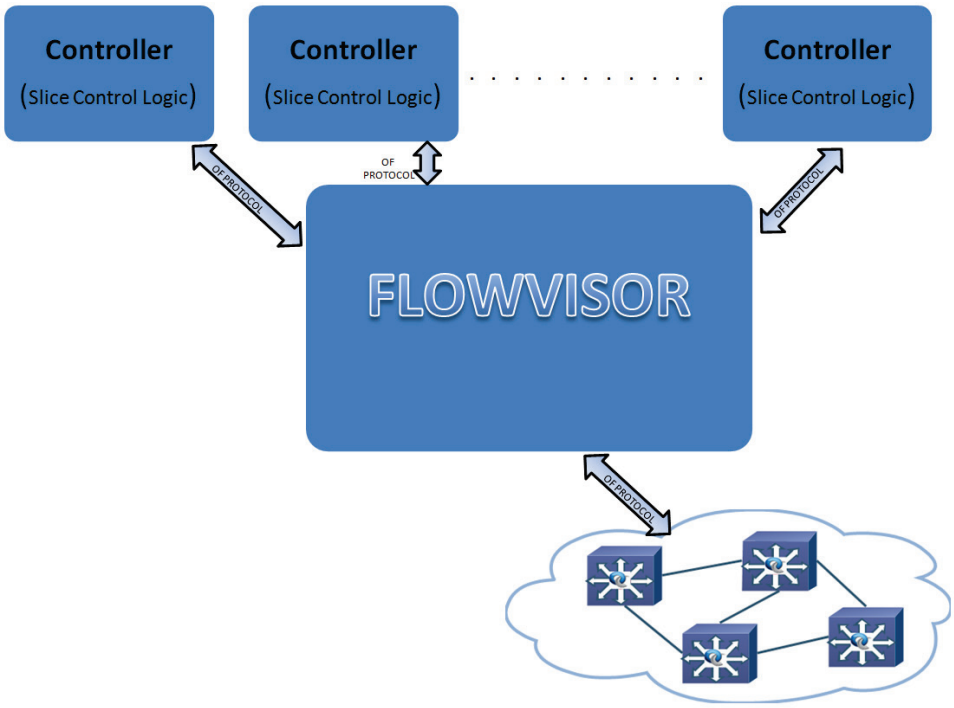
Μία λύση του προβλήματος αυτού για τους ερευνητές είναι η δημιουργία ενός “testbed” το οποίο είναι ενσωματωμένο σε ένα πραγματικό δίκτυο υπολογιστών, και συνεπώς λειτουργεί παράλληλα με αυτό. Όπως είδαμε στις προηγούμενες παραγράφους, με τη χρήση του πρωτοκόλλου OpenFlow και την αξιοποίησή του σε switches και Controllers είναι δυνατόν να διαχωριστεί πλήρως το Control Plane από το Data Plane. Μπορούμε τώρα να δημιουργήσουμε ένα ενδιάμεσο επίπεδο το οποίο θα "τεμαχίζει" το σύνολο των δικτυακών πόρων όπως φαίνεται στο Σχήμα 2.5, ούτως ώστε κάθε μία από τις "φέτες" (slices) που θα προκύψουν να μπορεί να χρησιμοποιηθεί σαν testbed για τις επιθυμητές δοκιμές και αξιολογήσεις. Εδώ πρέπει να επισημανθεί ότι το slice αυτό θα είναι πλήρως απομονωμένο από την παραγωγική δικτυακή κίνηση. Αυτή η τεχνική πραγματοποιείται με τη χρήση του FlowVisor [10].



Σχήμα 2.5 : Δικτυακοί πόροι διαχωρισμένοι σε slices και απομονωμένοι μεταξύ τους με χρήση του FlowVisor [Πηγή: [11] Reinvent Internet Infrastructure with OpenFlow and Software Defined Networking, 2010]

2.3.1 Γενική επισκόπηση του FlowVisor

Ο FlowVisor λειτουργεί σαν ένας "διαφανής" πληρεξούσιος (transparent proxy) ανάμεσα σε OpenFlow-enabled συσκευές και πολλαπλούς Controllers όπως φαίνεται στο Σχήμα 2.6. Κάθε slice που θα δημιουργηθεί, ελέγχεται από έναν και μόνο Controller ο οποίος αποτελεί την προγραμματιζόμενη λογική ελέγχου (control logic) του slice. Όλα τα μηνύματα OpenFlow ανάμεσα στα switches και τους Controllers στέλνονται μέσω του FlowVisor. Ταυτόχρονα, εκείνος χρησιμοποιεί το πρωτόκολλο OpenFlow για να επικοινωνήσει από τη μία μεριά με τους Controllers και από την άλλη με τα switches. Ακόμη, επειδή ο FlowVisor είναι "διαφανής", δεν χρειάζεται να γίνει καμία παραμετροποίηση στους Controllers, αφού νομίζουν ότι επικοινωνούν απευθείας με τα switches, ενώ παράλληλα το κάθε switch νομίζει ότι ελέγχεται από ένα και μόνο Controller.



Σχήμα 2.6 : Ο FlowVisor παρεμβάλλεται μεταξύ των Controllers που αποτελούν την λογική ελέγχου των slices, και πολλαπλών OpenFlow switches

Μπορούμε να συνοψίσουμε τη λειτουργία του FlowVisor στα παρακάτω τέσσερα σημαντικά σημεία:

- Ο FlowVisor ορίζει ένα slice σαν ένα σύνολο από flows που υπάρχουν σε μία δικτυακή τοπολογία
- Παρεμβάλλεται μεταξύ κάθε Controller και κάθε OpenFlow switch, έτσι ώστε κάθε Controller να μπορεί να παρατηρεί και να ελέγχει μόνο τα switches ή τμήματα των πόρων τους που πρέπει
- Διαχωρίζει το link bandwidth θέτοντας έναν ελάχιστο ρυθμό μεταφοράς δεδομένων για το σύνολο των flows που απαρτίζουν ένα slice
- Διαχωρίζει το flow-table σε κάθε switch, παρακολουθώντας ποιες εγγραφές flow ανήκουν σε κάθε slice.

Με βάση τα παραπάνω, χρησιμοποιώντας τον FlowVisor μπορούμε να τεμαχίσουμε στοιχεία του δικτύου, όπως το bandwidth, η τοπολογία, οι εγγραφές του flow-table, καθώς και η επεξεργαστική ισχύς των OpenFlow-enabled συσκευών. Έτσι κάθε χρήστης μπορεί να καθορίσει τις απαιτήσεις του, καθώς και την δικτυακή κίνηση που επιθυμεί να ελέγχεται από κάποιο συγκεκριμένο slice. Η τελευταία ορίζεται από ένα σύνολο από flows το οποίο

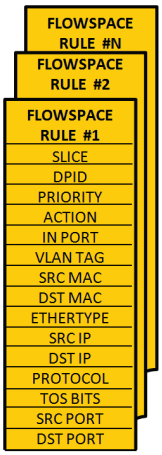
ονομάζεται flowspace και μαζί με τα παραπάνω στοιχεία χρησιμοποιείται για να καθοριστεί η πολιτική του κάθε slice. Εδώ πρέπει να σημειωθεί πως η τρέχουσα έκδοση του FlowVisor (version 0.7.1) υποστηρίζει την εικονικοποίηση (virtualization) και συνεπώς τον "τεμαχισμό" (slicing) της λογικής ελέγχου και προώθησης του κάθε switch, του flowspace και των εμπλεκομένων hardware πόρων του συστήματος (όπως είναι το bandwidth, οι ουρές, η τοπολογία και τα ports των switches).

2.3.2 Καθορισμός των Slices και της πολιτικής τους

Η πολιτική που επιβάλλει ο FlowVisor στα slices του ορίζεται από το σύνολο των δικτυακών πόρων, το flowspace και τον Controller που έχουν εκχωρηθεί σε κάθε slice. Όσων αφορά τους πόρους, μέσω της πολιτικής καθορίζεται ένα κομμάτι του συνολικού link bandwidth που είναι διαθέσιμο για το κάθε slice καθώς και ένα τμήμα της CPU και του flow-table ενός switch. Η πολιτική σχετικά με την τοπολογία καθορίζεται σαν μία λίστα κόμβων και ports που θα εκχωρηθούν στο κάθε slice. Αυτό όμως που θα μας απασχολήσει είναι η πολιτική που ορίζει το flowspace που ανήκει σε κάθε slice.

Το σύνολο των flows που ορίζει το flowspace ενός συγκεκριμένου slice μπορούμε να θεωρήσουμε ότι συνιστά έναν επακριβώς ορισμένο υποχώρο του συνολικού γεωμετρικού χώρου πιθανών κεφαλίδων των πακέτων. Έτσι, ορίζοντας τον υποχώρο αυτό, ουσιαστικά ορίζουμε την πολιτική ενός slice όσων αφορά τη δικτυακή κίνηση που το αφορά. Δεδομένων των κεφαλίδων ενός πακέτου ο FlowVisor μπορεί να αποφασίσει ποιο flowspace τις περιλαμβάνει και συνεπώς σε ποιο slice ανήκει. Μπορεί λοιπόν να απομονώσει δύο slices αφού επιβεβαιώσει ότι τα flowspaces δεν επικαλύπτονται ή να αποφασίσει ποιο switch πρέπει να χρησιμοποιηθεί για την επικοινωνία μεταξύ δύο διαφορετικών slices. Ακόμη μπορεί να επιτρέψει σε ένα πακέτο να ανήκει ταυτόχρονα σε δύο ή περισσότερα slices αφού κάποιο ή κάποια slices μπορεί να χρησιμοποιούνται για λόγους παρακολούθησης άλλων slices.

Το flowspace για κάθε slice ορίζεται όπως φαίνεται και στο Σχήμα 2.7 σαν ένα σύνολο κανόνων για τα πεδία του Πίνακα 2.2, με τρόπο παρόμοιο με αυτόν του ορισμού κανόνων firewall. Κάθε κανόνας περιλαμβάνει ένα action όπως allow, read-only ή deny και χρησιμοποιείται με βάση την προτεραιότητα του. Ακόμη, σε περιπτώσεις ίσης προτεραιότητας επικρατεί ο κανόνας που βρέθηκε πρώτος. Τέλος ο κάθε κανόνας συσχετίζεται και με ένα συγκεκριμένο OpenFlow switch (datapath) με βάση Datapath ID που θα οριστεί. Οι κανόνες συνδυάζονται ώστε να δημιουργήσουν ένα κομμάτι της δικτυακής κίνησης που θα ελέγχεται από ένα συγκεκριμένο slice.



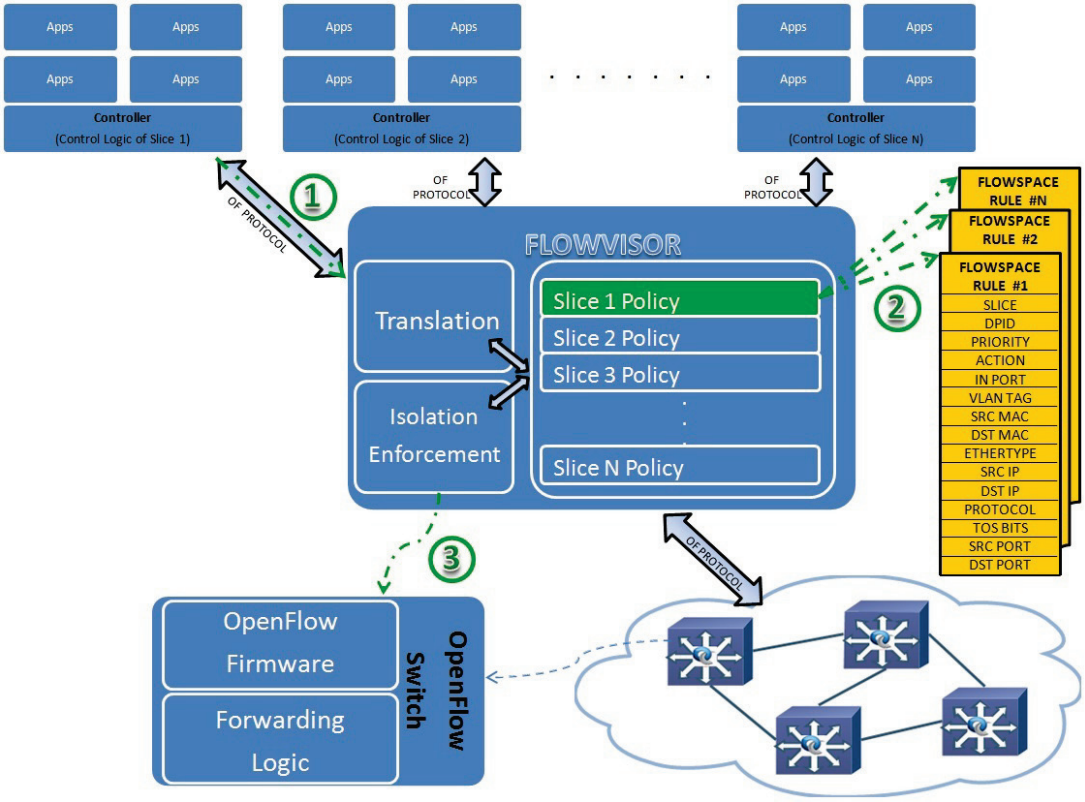
Σχήμα 2.7 : Το flowspace ενός slices ορίζεται από ένα σύνολο κανόνων όπως οι κανόνες του firewall

2.3.3 Λειτουργία του FlowVisor

Ο FlowVisor επιβάλλει τη διαφάνεια και την απομόνωση των slices επιθεωρώντας, επανεγγράφοντας και ελέγχοντας τα μηνύματα OpenFlow μεταξύ των switches και των Controllers. Χρησιμοποιώντας την πολιτική που έχει οριστεί για τους πόρους του κάθε slice (Resource Allocation Policy), τον τύπο του μηνύματος, τον προορισμό, την πηγή και γενικά το περιεχόμενο ενός μηνύματος OpenFlow, θα το προωθήσει αυτούσιο, θα το επανεγγράψει, ή θα το επιστρέψει στον αποστολέα του μαζί με ένα μήνυμα σφάλματος .

Για ένα μήνυμα που στάλθηκε από έναν Controller, με τις παραπάνω μεθόδους, ο FlowVisor θα επιβεβαιώσει ότι οι ενέργειες που αυτό περιλαμβάνει θα ισχύσουν μόνο για τους πόρους τους οποίους ελέγχει και διαχειρίζεται ο Controller. Τα πιο σημαντικά μηνύματα που μπορεί να στείλει ένας Controller αφορούν την προσθήκη, διαγραφή ή τροποποίηση εγγραφών του flow-table ενός switch. Όπως έχει αναφερθεί, οι κανόνες προώθησης πακέτων σε ένα switch αποτελούνται από ένα σύνολο κανόνων flow, καθώς και ένα σύνολο actions που θα ισχύσουν. Κατά τη διαδικασία λοιπόν του ορισμού των flows αυτών, και με σκοπό την διατήρηση της διαφάνειάς του και την απομόνωση των slices, ο FlowVisor επανεγγράφει τους κανόνες flow καθώς και τα actions που περιλαμβάνονται στο OpenFlow μήνυμα που θα στείλει ο Controller. Αυτό γίνεται ούτως ώστε να μην παραβιάζεται η πολιτική που διέπει το slice με το οποίο είναι συσχετισμένος ο Controller. Στο Σχήμα 2.8 φαίνεται ένα παράδειγμα αποστολής ενός πακέτου από το control plane (Controller) προς το forwarding plane (OpenFlow switch). Ο FlowVisor παρεμβάλλεται μεταξύ του OpenFlow switch και του Controller που διαχειρίζεται το Slice 1, λαμβάνοντας ένα OpenFlow μήνυμα που έστειλε ο τελευταίος και συλλέγοντας τις απαραίτητες πληροφορίες (Βήμα 1). Στη συνέχεια, ελέγχει τις πληροφορίες αυτές χρησιμοποιώντας το σύνολο των κανόνων που αποτελούν την πολιτική

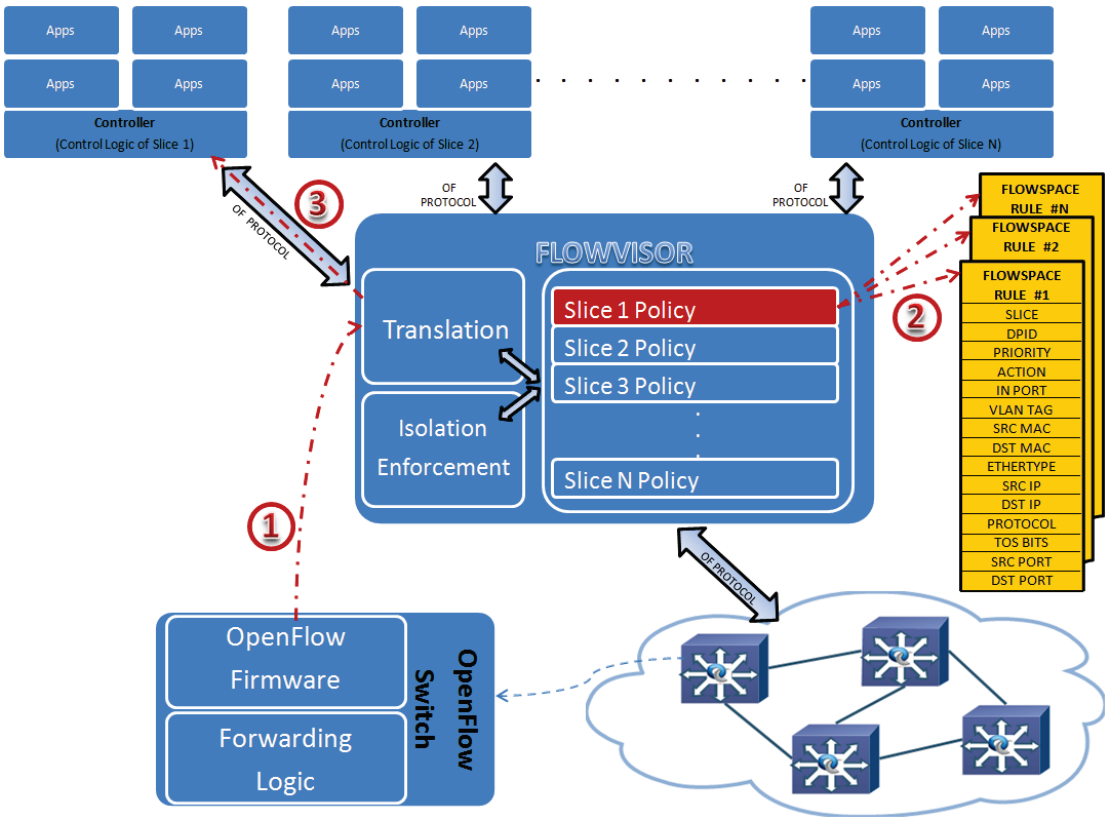
του Slice 1 (Βήμα 2). Τέλος επανεγγράφει το μήνυμα ώστε να συμμορφώνεται με την πολιτική αυτή (Isolation Enforcement) και το προωθεί στο OpenFlow switch που προοριζόταν από την αρχή (Βήμα 3).



Σχήμα 2.8 : Παράδειγμα της διαδικασίας αποστολής ενός OpenFlow μηνύματος από το Control Plane (Controller) προς το Forwarding Plane (OpenFlow switch), με τον FlowVisor να παρεμβάλλεται, επιβάλλοντας την εφαρμογή της πολιτικής του κάθε slice

Με έναν εντελώς ανάλογο τρόπο ελέγχεται και η δικτυακή κίνηση από το forwarding plane προς το data plane, όπως φαίνεται και στο Σχήμα 2.9. Έτσι, για ένα μήνυμα σταλμένο προς την αντίθετη κατεύθυνση σε σχέση με το προηγούμενο παράδειγμα, δηλαδή από ένα switch προς έναν Controller, ο FlowVisor θα εξετάσει αυτό το μήνυμα. Στη συνέχεια, ανάλογα με το περιεχόμενο του μηνύματος και την πολιτική του κάθε slice θα το προωθήσει μόνο στους Controllers εκείνων των slices που πρέπει. Γενικότερα, στην περίπτωση αυτή ο FlowVisor επανεγγράφει προσεκτικά τα μηνύματα από το OpenFlow switch ώστε να διατηρείται η διαφάνεια του ίδιου. Στοχεύοντας στην απομόνωση των slices μεταξύ τους, ο FlowVisor θα προωθήσει στον Controller ενός slice μηνύματα που αφορούν το control plane, μόνο αν το switch που έστειλε το μήνυμα ανήκει στην τοπολογία του συγκεκριμένου slice όπως ορίζεται από την πολιτική του. Ακόμη θα επανεγγράψει τα Feature Negotiation μηνύματα, έτσι ώστε ο Controller να μπορεί να δει μόνο τα ports εκείνα που επιτρέπεται να

γνωρίζει. Τέλος μηνύματα του τύπου port up/port down φιλτράρονται με τον ίδιο τρόπο και προωθούνται μόνο στα slices που πρέπει.



Σχήμα 2.9 : Διαδικασία αποστολής ενός OpenFlow μηνύματος από ένα OpenFlow switch προς τον Controller

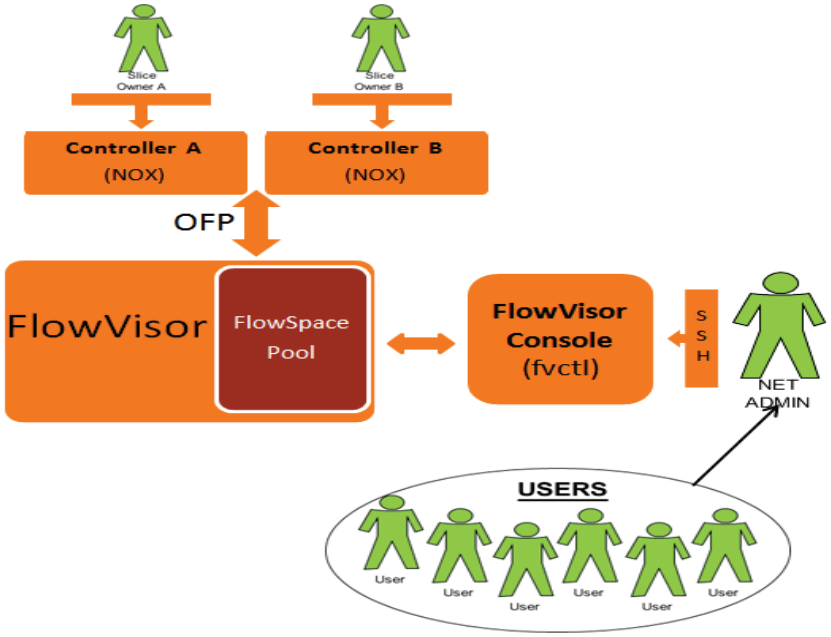
2.3.4 Διαχείριση του FlowVisor

Ο FlowVisor στην ουσία είναι ένας ακόμη Controller ο οποίος, όπως έχει ήδη αναφερθεί, δρα σαν ένας διαφανής διαμεσολαβητής ανάμεσα στα switches και στους Controllers που τα διαχειρίζονται. Οπότε, όπως και κάθε άλλος Controller, τρέχει σε ένα συμβατικό υπολογιστή ή server σαν μία εφαρμογή. Στην τρέχουσα υλοποίησή του, η διαχείριση του FlowVisor γίνεται απευθείας μέσω κονσόλας του υπολογιστή ή μέσω απομακρυσμένης σύνδεσης (SSH). Όμως η δυνατότητα αυτή μέχρι στιγμής περιορίζεται και απευθύνεται αποκλειστικά και μόνο στον διαχειριστή του δικτύου (network administrator).

Η αφαιρετικότητα που παρέχει ο FlowVisor με το διαχωρισμό της δικτυακής κίνησης σε flowspaces, δίνει τη δυνατότητα σε απλούς χρήστες να αναθέσουν τη διαχείριση της δικτυακής τους κίνησης, ή μέρος αυτής, σε όποιο από τα slices αυτοί επιθυμούν. Αυτό βέβαια

κατόπιν συνεννόησης με τον administrator, ο οποίος θα αναλάβει να πραγματοποιήσει τις απαραίτητες ρυθμίσεις μέσω της κονσόλας του FlowVisor.

Ένα παράδειγμα της παραπάνω διαδικασίας φαίνεται στο Σχήμα 2.9. Υποθέτουμε ότι στο "επίπεδο" κάτω από τον FlowVisor υπάρχει ένα σύνολο από OpenFlow switches. Ο διαχειριστής του δικτύου έχει δημιουργήσει δύο ξεχωριστά και απομονωμένα slices, τα A και B, τα οποία ελέγχονται από τους αντίστοιχους Controllers. Το slice A διαχειρίζεται την παραγωγική κίνηση του δικτύου, ενώ το slice B χρησιμοποιείται για τον διαμοιρασμό όλης της διαδικτυακής κίνησης (HTTP traffic) σε πολλαπλούς web servers. Ο διαχωρισμός αυτός αποσκοπεί στη βελτιστοποίησης της ποιότητας υπηρεσιών (Quality of Service). Ο τρόπος ελέγχου και προώθησης των πακέτων για κάθε slice, έχει ορίστει από τον ιδιοκτήτη του κάθε ενός (Slice Owner) μέσω των εφαρμογών που υπάρχουν στον Controller του κάθε slice. Υπάρχει τέλος ένας αριθμός χρηστών (Users) οι οποίοι επιθυμούν να εντάξουν όλη την HTTP κίνησή τους στο slice B. Οι χρήστες αυτοί πρέπει να επικοινωνήσουν με τον διαχειριστή του δικτύου, ο οποίος με τη σειρά του θα ρυθμίσει τον FlowVisor προσθέτοντας (ή αφαιρώντας) τα απαραίτητα τμήματα δικτυακών πόρων στο flowspace του κάθε slice ώστε να ικανοποιηθούν οι απαιτήσεις των χρηστών.



Σχήμα 2.10 : Ο Διαχειριστής του δικτύου (Net Admin) ελέγχει τον FlowVisor, δημιουργεί τα slices, τα οποία ελέγχουν οι ιδιοκτήτες τους μέσω των αντίστοιχων Controllers, και ικανοποιεί τις απαιτήσεις των χρηστών σχετικά με την δικτυακή τους κίνηση

3

Ανάλυση Συστήματος

Όπως είδαμε στην προηγούμενη παράγραφο, ο FlowVisor είναι ένα εργαλείο με το οποίο μπορούμε να "τεμαχίσουμε" τους δικτυακούς πόρους σε slices, απομονωμένα μεταξύ τους, το καθένα από τα οποία ελέγχεται από έναν διαφορετικό Controller. Όμως η διαχείριση του χρήσιμου αυτού εργαλείου είναι αυτή τη στιγμή ευθύνη του διαχειριστή του δικτύου και μόνο αυτού. Σε ένα ιδανικό σύστημα διαχείρισης δικτύων υπολογιστών ο χρήστης, είτε αυτός είναι ο ιδιοκτήτης ενός slice είτε είναι ένας απλός χρήστης ο οποίος θέλει να εντάξει την δικτυακή του κίνηση ή τμήμα αυτής σε ένα συγκεκριμένο slice, θα έπρεπε να έχει τη δυνατότητα να πραγματοποιεί μόνος του τις απαραίτητες ενέργειες στον FlowVisor. Πάντα όμως με την προϋπόθεση ότι πρώτα θα πιστοποιηθεί η ταυτότητά του και θα ελεγχθούν οι απαιτήσεις του σύμφωνα με μία πολιτική που θα έχει ορίσει ο ίδιος ο διαχειριστής του δικτύου.

Ένας τρόπος λύσης αυτού του προβλήματος προτείνεται στην παρούσα διπλωματική εργασία, δημιουργώντας μία Διεπαφή Χρήστη (Web User Interface - WebUI), η οποία θα χρησιμοποιεί τις λειτουργίες του FlowVisor αλλά ταυτόχρονα θα παρέχει στον διαχειριστή του δικτύου τη δυνατότητα καθορισμού πολιτικών και περιορισμών για τον κάθε χρήστη. Ταυτόχρονα θα δίνει τη δυνατότητα στους χρήστες, όπως αυτοί ορίστηκαν παραπάνω, να καθορίζουν οι ίδιοι τον έλεγχο της δικτυακής τους κίνησης, πάντα όμως υπό τους περιορισμούς που έχει θέσει ο διαχειριστής.

Στις επόμενες παραγράφους περιγράφεται η αρχιτεκτονική που χρησιμοποιήθηκε για τη δημιουργία αυτού του Web-UI καθώς και οι απαιτήσεις που υπάρχουν για την σωστή και ολοκληρωμένη λειτουργία του.

3.1 Αρχιτεκτονική του συστήματος

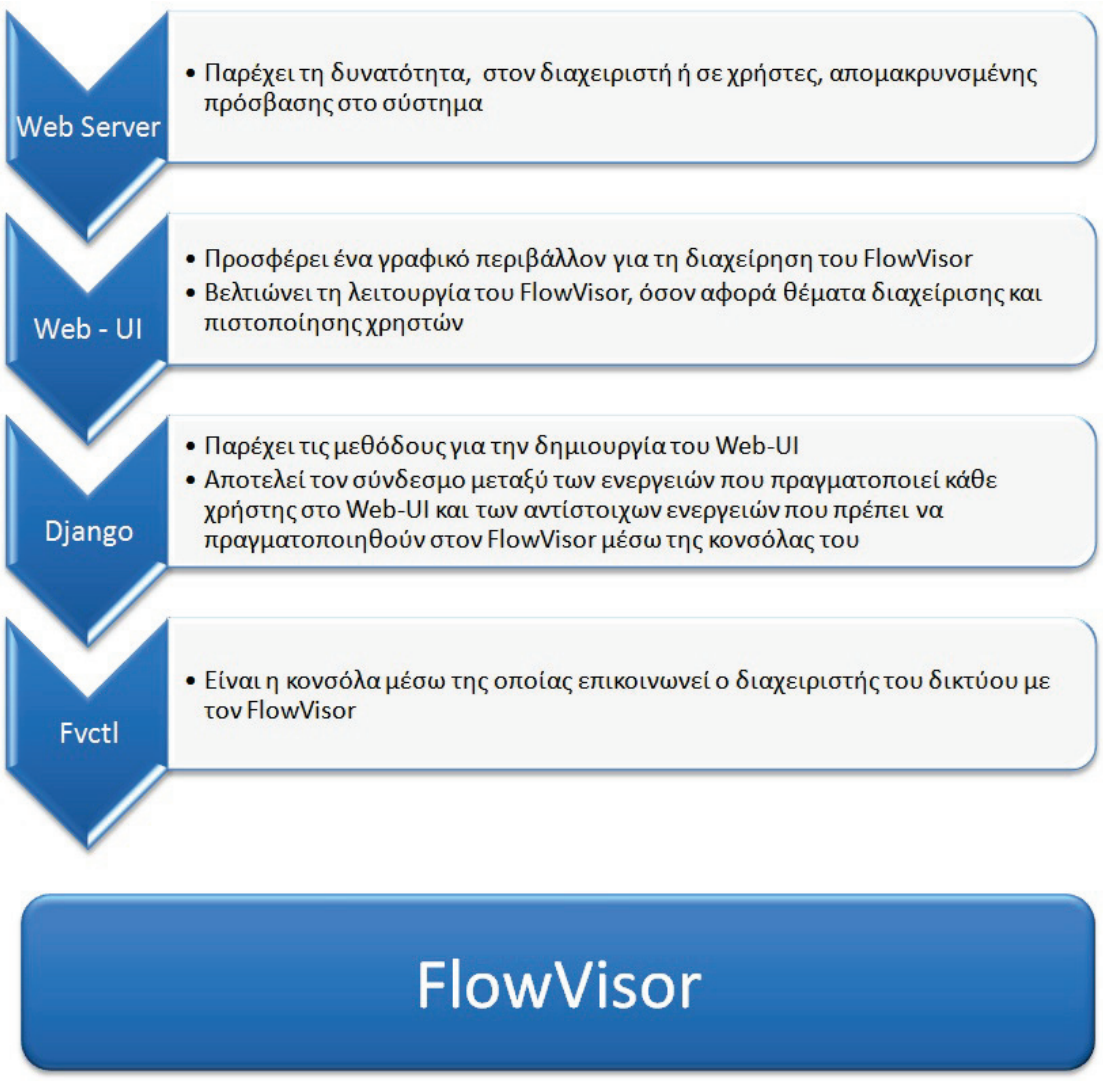
Το σύστημα που δημιουργήθηκε, μπορεί να διαχωριστεί σε τέσσερα υποσυστήματα, καθένα από τα οποία έχει έναν πολύ σημαντικό και συγκεκριμένο σκοπό. Αυτά είναι, ο FlowVisor, το Django, ο εξυπηρετητής ιστού (web server) και τέλος το Web-UI, που αποτελεί και τον τελικό στόχο.

Το πρώτο και ίσως σημαντικότερο υποσύστημα αποτελεί ο ίδιος ο FlowVisor και κατ'επέκταση η κονσόλα που παρέχει για τη δημιουργία, διαγραφή, παραμετροποίηση και γενικότερα για τη διαχείρηση των slices του δικτύου. Πρόσβαση σε αυτήν την κονσόλα μπορεί να έχει αποκλειστικά και μόνο ο διαχειρηστής του δικτύου ή κάποιος χρήστης ο οποίος έχει διαπιστευτήρια διαχειριστή (administrator credentials) του ηλεκτρονικού υπολογιστή όπου λειτουργεί ο FlowVisor.

Ακόμη, πολύ σημανατικό υποσύστημα αποτελεί το web framework Django [12] που χρησιμοποιήθηκε για την υλοποίηση του Web-UI και την σύνδεση των λειτουργιών του με την κονσόλα του FlowVisor. Το Django σχεδιάστηκε ώστε να διευκολύνει τη δημιουργία σύνθετων ιστοσελίδων που απαιτούν τη χρήση βάσεων δεδομένων, ενώ δίνει έμφαση στην επαναχρησιμοποίηση και διασύνδεση των στοιχείων του. Έτσι δίνει τη δυνατότητα δημιουργίας εφαρμογών Web υψηλών επιδόσεων, με τρόπο αρκετά κομψό, εύκολο και γρήγορο.

Ένα ακόμα απαραίτητο υποσύστημα αποτελεί ο Web Server που τρέχει στον ίδιο ηλεκτρονικό υπολογιστή με το Django. Αυτός δίνει τη δυνατότητα σε οποιονδήποτε, από τον διαχειριστή του δικτύου μέχρι τον απλό χρήστη, να κάνει χρήση του FlowVisor απομακρυνσμένα μέσω της εφαρμογής ιστού που δημιουργείται με το Django, χωρίς να απαιτείται η πρόσβασή του (φυσική ή απομακρυνσμένη μέσω ασφαλούς σύνδεσης) στην κονσόλα του υπολογιστή που φιλοξενεί τον FlowVisor.

Τέλος, μετά τα υποσυστήματα που αναφέρθηκαν παραπάνω, φτάνουμε στο Web-UI, δηλαδή την εφαρμογή ιστού που δημιουργήθηκε κάνοντας χρήση του Django και που ουσιαστικά, εκτός από την διευκόλυνση που προσφέρει στη χρήση του FlowVisor παρέχοντας ένα γραφικό περιβάλλον, βελτιώνει τη λειτουργία του παρέχοντας μία αποτελεσματική λύση στο πρόβλημα που αναφέρθηκε στην αρχή του κεφαλαίου. Στο Σχήμα 3.1 που ακολουθεί φαίνεται πώς συνεργάζονται αυτά τα υποσυστήματα, από τη στιγμή που ένας χρήστης μέσω του Web Server θα αποκτήσει πρόσβαση στο Web-UI ώστε να χρησιμοποιήσει απομακρυνσμένα τον FlowVisor, μέχρι τη στιγμή που οι ενέργειές του θα αποκτήσουν ισχύ στην δικτυακή κίνηση η οποία ελέγχεται από τον FlowVisor.



Σχήμα 3.1: Τα τέσσερα κύρια υποσυστήματα συνεργάζονται ώστε να δώσουν την δυνατότητα σε έναν χρήστη ή διαχειριστή να ελέγξει απομακρυνσμένα τον FlowVisor

Στις επόμενες παραγράφους θα παρουσιαστούν αναλυτικότερα οι δυνατότητες που παρέχει το κάθε ένα απο τα υποσυστήματα, πώς αυτά συνεργάζονται, καθώς και το μοντέλο Οντοτήτων - Συσχετίσεων της βάσης δεδομένων που χρησιμοποιείται.

3.2 Περιγραφή Λειτουργιών των υποσυστημάτων

3.2.1 Κονσόλα του FlowVisor

Μέσω της γραμμής εντολών του υπολογιστή όπου ζει ο FlowVisor, ο διαχειριστής ελέγχει την λειτουργία του, χρησιμοποιώντας δύο εργαλεία-εφαρμογές, τα *fvctl* και *fvconfig*. Ο FlowVisor λειτουργεί μέσω ενός web server ο οποίος μέσω του πρωτοκόλλου XML-RPC (eXtensible Markup Language - Remote Procedure Call) [13] δέχεται κλήσεις τύπου *HTTP request* από την διεπαφή προγράμματος εφαρμογής (Application Program Interface - API). Παρακάτω περιγράφονται αναλυτικότερα οι δυνατότητες που προσφέρουν οι δύο αυτές εφαρμογές στην έκδοση 0.7.1 του λογισμικού του FlowVisor.

3.2.1.1 Fvconfig

Η διάρθρωση (configuration) της εγκατάστασης του FlowVisor αποθηκεύεται σε ένα αρχείο XML, το οποίο όμως δεν είναι εύκολα αναγνώσιμο από τον άνθρωπο. Αυτός είναι και ο σκοπός της εφαρμογής fvconfig, δηλαδή να εμφανίζει τις πληροφορίες που είναι αποθηκευμένες σε ένα τέτοιο αρχείο με μορφή εύκολα αναγνώσιμη, και ταυτόχρονα να προσφέρει στον διαχειριστή τη δυνατότητα να "παραποιεί" τις πληροφορίες αυτές απευθείας και σύμφωνα με τις ανάγκες του, χωρίς να βρίσκεται σε λειτουργία ο FlowVisor. Στον Πίνακα 3.1 φαίνεται το σύνολο των διαθέσιμων εντολών της εφαρμογής καθώς και μια σύντομη περιγραφή τους. Στο συγκεκριμένο XML αρχείο είναι αποθηκευμένα:

- Στοιχεία που αφορούν την λειτουργία του FlowVisor, όπως το port όπου ακούει ο web-server για τις κλήσεις από το API, το port όπου περιμένει να συνδεθούν τα OpenFlow switches που αποτελούν το δίκτυο που διαχειρίζεται, κ.α.

- Τα στοιχεία του κάθε Controller που ελέγχει κάποιο slice μέσω του FlowVisor, δηλαδή η διεύθυνση IP του, το port όπου ακούει και αναμένει να συνδεθούν OpenFlow switches, το όνομα του slice που διαχειρίζεται και τέλος το e-mail του διαχειριστή-ιδιοκτήτη του συγκεκριμένου slice.

- Το σύνολο των κανόνων-πολιτικών που έχει θεσπίσει ο διαχειριστής για κάθε slice, και που αποτελούν το FlowSpace καθενός από αυτά.

Εντολή	Περιγραφή
Match <structure>	Δεδομένης μιας συγκεκριμένης δομής δεδομένων, επιστρέφει όλους τους κανόνες flowspace που αντιστοιχούν με αυτήν
Dump	Εμφανίζει τις πληροφορίες που περιέχονται στο XML αρχείο σε μορφή εύκολα αναγνώσιμη από τον άνθρωπο
Chpasswd <slicename>	Αλλάζει τον κωδικό διαχείρισης ενός slice
Query <dpid> <slicename>	Αν παραληφθεί η παράμετρος <slicename> εμφανίζει όλα τα slices στο flowspace των οποίων περιλαμβάνεται το συγκεκριμένο DPID, αλλιώς εμφανίζει όλα τα ports του συγκεκριμένου DPID που ανήκουν στο flowspace του συγκεκριμένου slice
generateCert	Δημιουργεί ένα πιστοποιητικό SSL για την πιστοποίηση του διαχειριστή
generate	Δημιουργεί ένα καινούργιο αρχείο XML για την αποθήκευση των ρυθμίσεων του FlowVisor

Πίνακας 3.1 : Το σύνολο των διαθέσιμων εντολών της εφαρμογής fvconfig

3.2.1.2 Fvctl

Η εφαρμογή Fvctl είναι το βασικότερο εργαλείο για τον έλεγχο του FlowVisor. Χρησιμοποιώντας την μέσω της γραμμής εντολών, ο διαχειριστής πραγματοποιεί τις κλήσεις προς τον web-server του FlowVisor μέσω του πρωτοκόλλου XML-RPC με σκοπό την ρύθμιση, την επίλυση προβλημάτων, την παρακολούθηση και τη διαχείρισή του. Η εφαρμογή αυτή είναι που δίνει τη δυνατότητα στο διαχειριστή να παρακολουθεί την τρέχουσα κατάσταση του FlowVisor, να δημιουργεί και να παραμετροποιεί τα slices, καθώς και να προσθέτει εγγραφές flowspace μεταβάλλοντας την πολιτική του κάθε slice. Αυτό φαίνεται και από το σύνολο των διαθέσιμων εντολών της συγκεκριμένης εφαρμογής στον Πίνακα 3.2. Ενώ με την εφαρμογή fvconfig ο FlowVisor πρέπει να μην είναι σε λειτουργία για να μπορέσει ο διαχειριστής να τον παραμετροποιήσει, το fvctl του παρέχει αυτή τη δυνατότητα σε πραγματικό χρόνο. Πιο συγκεκριμένα, οι βασικότερες δυνατότητες που του παρέχει αυτή η εφαρμογή είναι:

- Να δει το σύνολο των slices που υπάρχουν μία δεδομένη στιγμή.

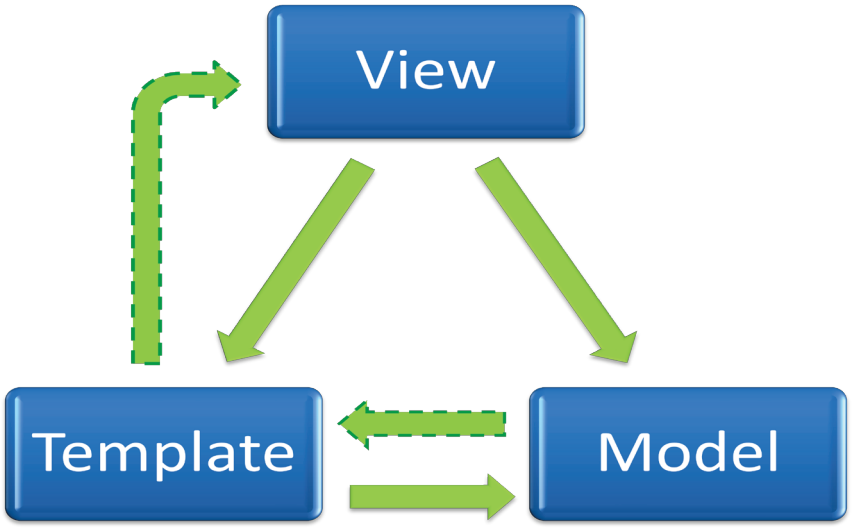
- Να δημιουργήσει ή να διαγράψει slices.
- Να δει το σύνολο των κανόνων flowspace όλων των slices.
- Να προσθέσει ή να αφαιρέσει κανόνες flowspace.
- Να δει όλα τα OpenFlow switches που είναι συνδεδεμένα με τον FlowVisor μία δεδομένη στιγμή, καθώς και τους συνδέσμους (links) μεταξύ τους.

Εντολή	**Περιγραφή**
listSlices	Εμφανίζει το σύνολο των slices που έχουν δημιουργηθεί
getSliceInfo	Εμφανίζει πληροφορίες σχετικά με τον Controller ενός slice (IP, port κ.α.)
createSlice	Δημιουργεί ένα καινούριο slice
deleteSlice	Διαγράφει ένα slice
changePasswd	Αλλάζει τον διαχειριστικό κωδικό ενός slice
listFlowSpace	Εμφανίζει όλους τους κανόνες που ορίζουν την πολιτική των slices
removeFlowSpace	Διαγράφει έναν κανόνα flowspace
addFlowSpace	Προσθέτει έναν κανόνα flowspace
listDevices	Εμφανίζει όλα τα OpenFlow switches που είναι συνδεδεμένα
getDeviceInfo	Εμφανίζει πληροφορίες ενός switch, όπως ports κ.α.
getLinks	Εμφανίζει τους συνδέσμους μεταξύ των OpenFlow switches
registerCallback	Εγκαθιστά ένα xmlrpc callback που θα ενημερώνει τον FlowVisor σε περίπτωση αλλαγής της τοπολογίας
unregisterCallback	Απεγκαθιστά ένα xmlrpc callback
setConfig	Ρυθμίζει παραμέτρους που αφορούν τη λειτουργία του FlowVisor
getConfig	Εμφανίζει όλες τις παραμέτρους για τη λειτουργία του FlowVisor
getSliceStats	Εμφανίζει στατιστικά για ένα slice, όπως απεσταλμένα μηνύματα κ.α.
getSwitchStats	Εμφανίζει στατιστικά για ένα switch, όπως απεσταλμένα μηνύματα κ.α.
getSwitchFlowDB	Εμφανίζει το flow-table ενός switch όπως το ξέρει ο FlowVisor
getSliceRewriteDB	Εμφανίζει τις αλλαγές που έχει πραγματοποιήσει ο FlowVisor στις εγγραφές flow ενός switch όσον αφορά ένα συγκεκριμένο slice
ping	Ελέγχει αν λειτουργούν ο XMLRPC server και η πιστοποίηση χρηστών

Πίνακας 3.2 : Το σύνολο των διαθέσιμων εντολών της εφαρμογής fvctl

3.2.2 Django web framework και Web Server

Το Django είναι ένα ελεύθερο πλαίσιο (framework) δημιουργίας εφαρμογών ιστού σε γλώσσα Python, το οποίο ακολουθεί την αρχιτεκτονική Model-View-Controller (MVC) [14]. Η αρχιτεκτονική αυτή απομονώνει το επιπίπεδο της λογικής της εφαρμογής από το επίπεδο της διεπαφής χρήστη, επιτρέποντας την ανεξάρτητη ανάπτυξη, συντήρηση και έλεγχο των δύο αυτών επιπέδων. Σύμφωνα βέβαια με την τεκμηρίωση του Django, η αρχιτεκτονική που χρησιμοποιεί ονομάζεται Model-Template-View (MTV - Σχήμα 3.2). Σύμφωνα λοιπόν με αυτήν την αρχιτεκτονική, Model ονομάζεται το επίπεδο πρόσβασης δεδομένων (data-access layer), όπου η εφαρμογή ιστού αλληλεπιδρά με στοιχεία βάσεων δεδομένων και άλλων πληροφοριών. Το επίπεδο Template είναι εκείνο που καθορίζει πώς εμφανίζονται τα δεδομένα στον χρήστη, κάτι που στην αρχιτεκτονική MVC αποτελεί το επίπεδο View. Τέλος το επίπεδο View περιγράφει ποιά δεδομένα πρέπει να εμφανιστούν σε έναν δεδομένο χρήστη. Δεν προσδιορίζει επακριβώς πώς θα εμφανιστούν, παρά αναθέτει την συγκεκριμένη εργασία στο επίπεδο Template. Το επίπεδο View της αρχιτεκτονικής MTV, αναφέρεται στο επίπεδο Controller της αρχιτεκτονικής MVC.



Σχήμα 3.2 Έννοια της αρχιτεκτονικής MTV. Οι διακεκομένες γραμμές αντιπροσωπεύουν έναν έμμεσο συσχετισμό δύο στοιχείων, ενώ οι υπόλοιπες αντιπροσωπεύουν τον άμεσο συσχετισμό δύο στοιχείων

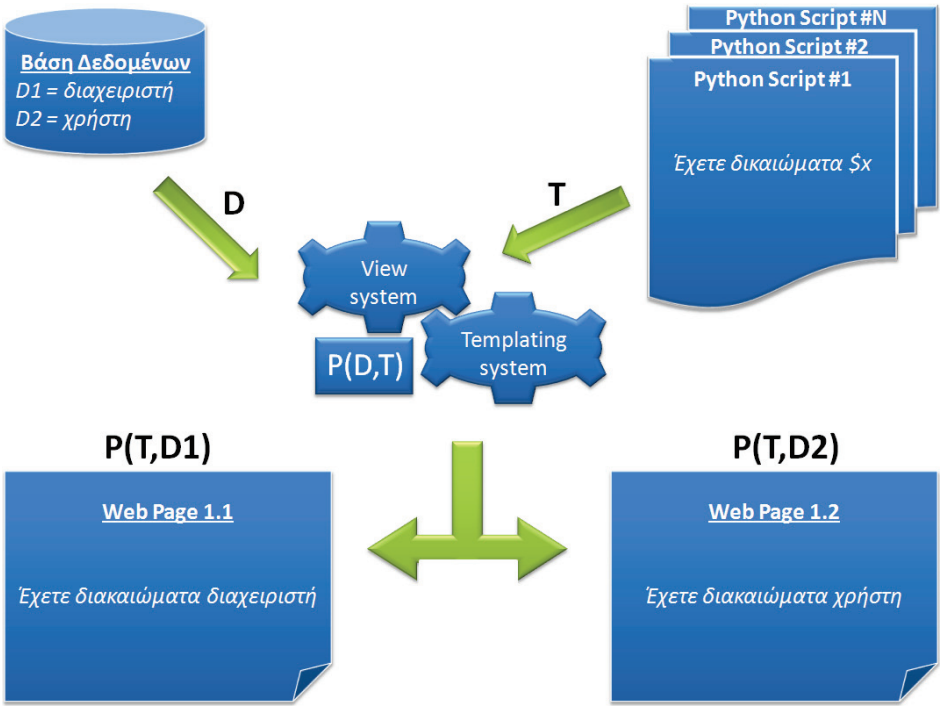
Ο πυρήνας του Django λοιπόν, αποτελείται κατά κύριο λόγο από τρεις μηχανισμούς [15]:

i. Έναν απεικονιστή (mapper) ο οποίος προσφέρει τη δυνατότητα συσχέτισης μοντέλων δεδομένων (που ορίζονται σαν κλάσεις Python) με σχεσιακές βάσεις δεδομένων (Model Layer).

ii. Ένα σύστημα επισκόπησης (View Layer) για την επεξεργασία των αιτημάτων (requests)

iii. Ένα σύστημα σχεδιοτύπων ιστού (web templating system) το οποίο δίνει στο χρήστη τη δυνατότητα δημιουργίας, επεξεργασίας και συντήρησης μαζικών αρχείων web, με σκοπό τη δημιουργία και συντήρηση ιστοσελίδων (Template Layer).

Ένα απλοϊκό παράδειγμα χρήσης του Django φαίνεται στο Σχήμα 3.3. Μέσω του Django έχει δημιουργηθεί μία βάση δεδομένων για ένα μοντέλο D, με τις εγγραφές D1 = "διαχειριστή" και D2 = "χρήστη". Παράλληλα έχουν δημιουργηθεί αρχεία κώδικα Python για την δυναμική δημιουργία ιστοσελίδων. Μέσω των λειτουργιών του Django συσχετίζονται οι εγγραφές της βάσης δεδομένων με την μεταβλητή x που ζητείται από το Script #1, και σαν αποτέλεσμα παράγεται μία διαφορετική ιστοσελίδα, ανάλογα με την τιμή των εγγραφών του μοντέλου D.



Σχήμα 3.3: Χρήση του Django για τη δημιουργία δυναμικών ιστοσελίδων

Με τη χρήση του Django μπορούμε πλέον να συνδέσουμε μεταξύ τους τα δύο επίπεδα του Σχήματος 3.1 που το "περιβάλλουν". Αυτό μπορεί να γίνει δημιουργώντας Python scripts, τα οποία όταν καλούνται θα τρέχουν τις ανάλογες εντολές στην κονσόλα του FlowVisor ώστε να πάρουν τις απαραίτητες πληροφορίες και να τις επεξεργαστούν.

Ακολούθως, οι πληροφορίες αυτές μπορούν να εγγραφούν στην βάση δεδομένων και πλέον να χρησιμοποιούνται κατά βούληση ώστε να παράγονται οι απαραίτητες ιστοσελίδες, το σύνολο των οποίων και αποτελεί το Web-UI του FlowVisor. Περίπου η ίδια διαδικασία μπορεί να γίνει και από την αντίθετη κατεύθυνση, ώστε μέσω του Web-UI να μπορούμε να τροποποιήσουμε τις ρυθμίσεις του FlowVisor. Λεπτομέρειες για τον τρόπο με τον οποίο υλοποιούνται αυτές οι διαδικασίες θα ακολουθήσουν στο επόμενο κεφάλαιο.

Το μόνο που χρειάζεται πλέον για την πρόσβαση του διαχειριστή του δικτύου, ή ενός χρήστη, στο Web-UI είναι η χρήση ενός web server ο οποίος θα αναλαμβάνει να αντιστοιχεί τα URL, όπως αυτά ζητούνται από το πρόγραμμα πλοήγησης του χρήστη, με τα αντίστοιχα αρχεία ιστοσελίδων, όπως αυτά παράγονται μέσω του Django. Αν και δεν έχει αναφερθεί στις βασικές του λειτουργίες, το Django περιλαμβάνει εκτός των άλλων και έναν web server για την ανάπτυξη ιστοσελίδων, ο οποίος είναι ιδανικός για την δοκιμή και ανεύρεση προβλημάτων της εφαρμογής ιστού που προσπαθεί κάποιος να αναπτύξει. Βέβαια ο συγκεκριμένος web server έχει σχεδιαστεί με σκοπό να λειτουργεί κυρίως τοπικά στο σύστημα όπου χρησιμοποιείται το Django και δεν θα μπορούσε να εξυπηρετήσει τις απαιτήσεις μιας εφαρμογής ιστού η οποία θα χρησιμοποιούνταν κάτω από πραγματικές συνθήκες, με πολλαπλούς χρήστες να ζητούν πρόσβαση σε αυτήν. Για αυτόν τον λόγο σε τέτοιες περιπτώσεις απαιτείται η χρήση κάποιου web server που μπορεί να ανταπεξέλθει κάτω από αυτές συνθήκες, όπως ο Apache, ή ο lighttpd. Παρόλα αυτά, για τις ανάγκες τις συγκεκριμένης διπλωματικής χρησιμοποιείται ο web-server που περιλαμβάνεται στη διανομή του Django, αφού σκοπός ήταν η ανάπτυξη μιας συγκεκριμένης εφαρμογής ιστού που θα αποτελεί μία Διεπαφή Χρήστη για τον FlowVisor.

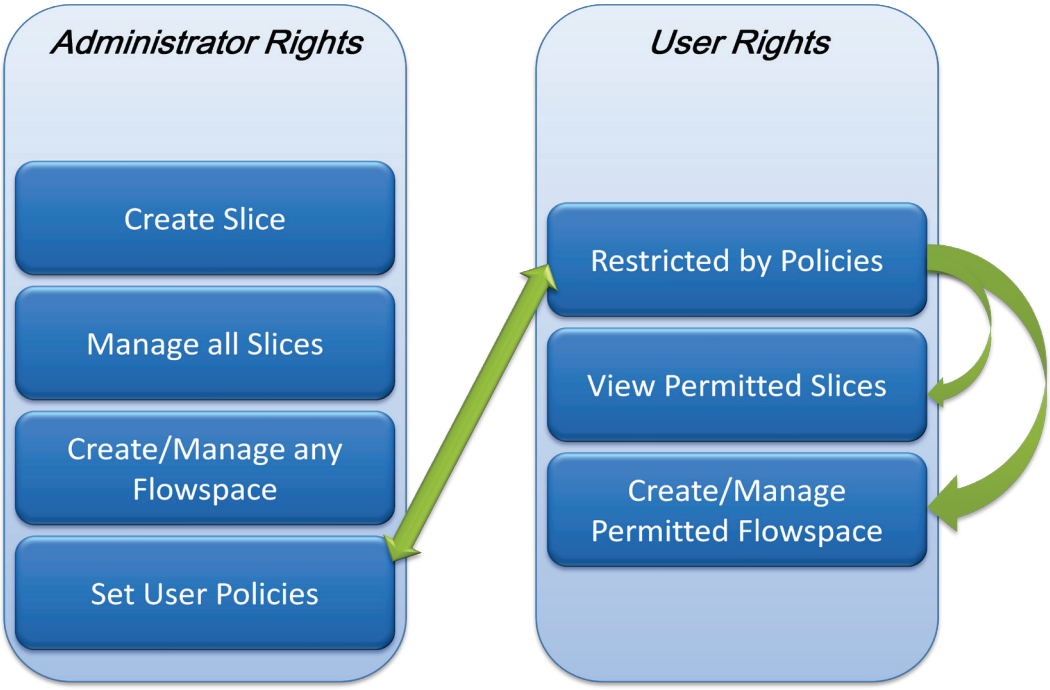
3.2.3 Διεπαφή Χρήστη (Web User Interface)

Σκοπός της δημιουργίας του Web-UI είναι να προσφέρει πρόσβαση στον διαχειριστή του δικτύου ή στους χρήστες που θέλουν να διαχειριστούν απομακρυσμένα τον FlowVisor. Παράλληλα βελτιώνει τη λειτουργία του, όσον αφορά θέματα διαχείρισης και πιστοποίησης χρηστών. Πιο συγκεκριμένα το Web-UI παρέχει στους χρήστες ένα γραφικό περιβάλλον για την εκτέλεση των βασικότερων εντολών του FlowVisor, δηλαδή:

- ✓ Δημιουργία και διαγραφή slices, ορίζοντας παράλληλα τον Controller που ελέγχει το καθένα από αυτά.
- ✓ Δημιουργία, διαγραφή και παραμετροποίηση των κανόνων flowspace για κάθε ένα από τα slices.
- ✓ Παρουσίαση των διαθέσιμων OpenFlow switches που είναι συνδεδεμένα στο δίκτυο.

Ο FlowVisor όμως, μέχρι στιγμής τουλάχιστον, δεν παρέχει κάποια μέθοδο για την δημιουργία χρηστών και τον καθορισμό πολιτικών που θα τους περιορίζουν. Έτσι, εκτός από τις ήδη υπάρχουσες λειτουργίες του FlowVisor, μέσω του Web-UI παρέχονται επιπλέον λειτουργίες και επιλογές για την διαχείριση του δικτύου, την πιστοποίηση χρηστών και τον καθορισμό ξεχωριστής πολιτικής για κάθε έναν από αυτούς. Οι λειτουργίες αυτές, όπως φαίνονται και στο Σχήμα 3.4, αναλύονται παρακάτω:

- ✓ Ο μόνος που μπορεί να δημιουργήσει καινούρια slices και να πραγματοποιήσει ενέργειες όπως εκκίνηση, επανεκκίνηση ή τερματισμό του FlowVisor είναι ο διαχειριστής του δικτύου, ή κάποιος χρήστης με δικαιώματα διαχειριστή.
- ✓ Ο διαχειριστής έχει τη δυνατότητα δημιουργίας λογαριασμών χρηστών. Έπειτα, για κάθε έναν από αυτούς τους χρήστες μπορεί να ορίσει την κατηγορία στην οποία ανήκει ο λογαριασμός του, δηλαδή *χρήστης* (στην εφαρμογή ονομάζεται staff user) ή *διαχειριστής* (super user στην εφαρμογή).
- ✓ Κατά τη δημιουργία ενός slice ο διαχειριστής ορίζει ποιοι χρήστες μπορούν να έχουν τον έλεγχο αυτού του slice.
- ✓ Εκτός του διαχειριστή, κάθε ένας από τους χρήστες βλέπει μόνο τα slices που του ανήκουν (αν είναι ιδιοκτήτης ενός ή περισσοτέρων) ή τα slices στα οποία συμμετέχει (αν είναι απλός χρήστης). Σύμφωνα με την ίδια λογική, βλέπει μόνο τους κανόνες που αποτελούν το flowspace το οποίο τον αφορά, ενώ τέλος βλέπει μόνο τα OpenFlow switches (datapaths) στα οποία του έχει δώσει πρόσβαση ο διαχειριστής.
- ✓ Μετά τη δημιουργία ενός καινούριου χρήστη, ο διαχειριστής μπορεί να ρυθμίσει τους κανόνες περιορισμού του χρήστη αυτού, οι οποίοι μαζί με την κατηγορία στην οποία έχει ενταχθεί ο χρήστης ορίζουν την πολιτική που τον περιορίζει. Οι κανόνες αυτοί ορίζονται έτσι ώστε για κάθε συνδυασμό ενός OpenFlow switch και ενός χρήστη να υπάρχει ένας μοναδικός κανόνας περιορισμού. Τέλος κάθε κανόνας ορίζεται από ένα σύνολο πεδίων, όπως αυτό φαίνεται από τον Πίνακα 3.3.
- ✓ Ο διαχειριστής μπορεί να δημιουργήσει καινούριους κανόνες flowspace για οποιοδήποτε slice (ακόμα και αν η διαχείρισή του έχει ανατεθεί σε κάποιον χρήστη), διατηρώντας έτσι την δικαιοδοσία που είχε στον FlowVisor εξαρχής.
- ✓ Κάθε χρήστης έχει τη δυνατότητα να διαχειριστεί το flowspace ενός slice, μόνο αν είναι εξουσιοδοτημένος από τον διαχειριστή για τον έλεγχο του συγκεκριμένου slice και πάντα υπό τον περιορισμό των κανόνων που ορίζουν την πολιτική του συγκεκριμένου χρήστη. Έτσι, κατά τη δημιουργία ή παραμετροποίηση ενός κανόνα flowspace από έναν χρήστη, τα διαθέσιμα slices και OpenFlow switches (datapaths) περιορίζονται σύμφωνα με τις ρυθμίσεις του διαχειριστή, ενώ τα υπόλοιπα πεδία ελέγχονται σύμφωνα με τους περιορισμούς της πολιτικής του χρήστη αυτού.

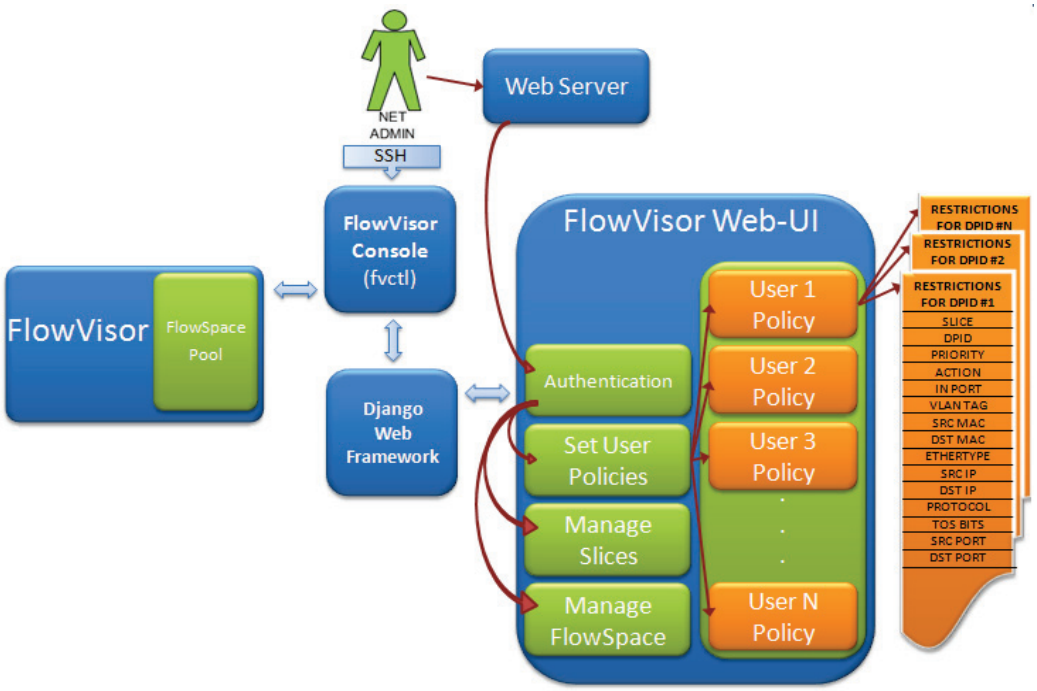


Σχήμα 3.4: Απεικόνιση των δικαιωμάτων που έχει η κάθε μία από τις δύο κατηγορίες χρηστών. Ως Administrator ορίζεται ο διαχειριστής ή κάποιος χρήστης με διαχειριστικά δικαιώματα, ενώ ως User ορίζεται κάθε ιδιοκτήτης ενός slice καθώς και κάθε απλός χρήστης

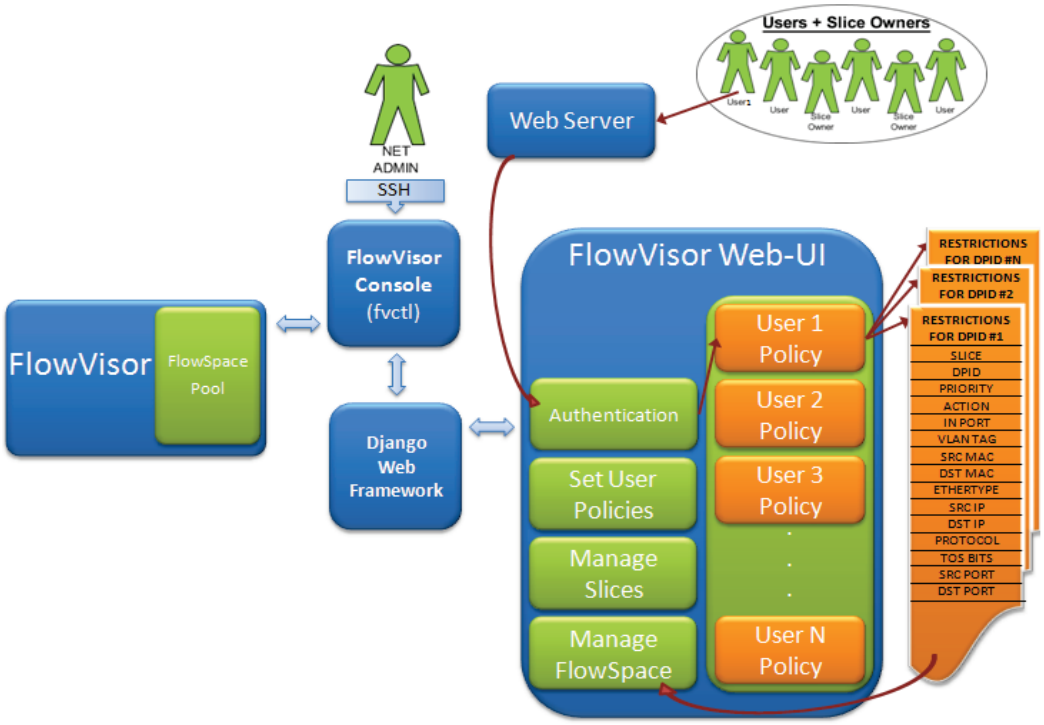
Field Category	Field Title
General Information	User
General Information	Datapath ID (dpid)
General Information	Priority
General Information	Action
Layer 2 Attributes	Ingress Port
Layer 2 Attributes	VLAN tag
Layer 2 Attributes	Source MAC
Layer 2 Attributes	Destination MAC
Layer 3 Attributes	Ethernet type
Layer 3 Attributes	Source IP
Layer 3 Attributes	Destination IP
Layer 4 Attributes	Protocol
Layer 4 Attributes	TOS bits
Layer 4 Attributes	Source Port
Layer 4 Attributes	Destination Port

Πίνακας 3.3 : Πεδία από τα οποία αποτελείται κάθε κανόνας για τον ορισμό των περιορισμών ενός χρήστη

Με βάση λοιπόν τα παραπάνω, και συνδυάζοντας το Web-UI του FlowVisor με την κονσόλα του, μέσω του Django και χρησιμοποιώντας έναν web server, πετυχαίνουμε το επιθυμητό αποτέλεσμα όπως φαίνεται και στα δύο παρακάτω σχήματα. Στο Σχήμα 3.5 φαίνονται οι δυνατότητες που παρέχει το Web-UI στον διαχειριστή, ο οποίος αφού φτάσει σε αυτό μέσω του web server και πιστοποιηθεί η ταυτότητά του, έχει τη δυνατότητα να καθορίσει την πολιτική που θα διέπει κάθε χρήστη, να δημιουργήσει ή να διαχειριστεί slices καθώς και flowspaces. Στο Σχήμα 3.6 φαίνονται οι δυνατότητες που παρέχει το Web-UI σε ένα χρήστη, είτε αυτός είναι ο ιδιοκτήτης/διαχειριστής ενός slice, είτε ένας απλός χρήστης ο οποίος χρησιμοποιεί ένα συγκεκριμένο slice για τον έλεγχο της δικτυακής του κίνησης, ή τμήματος αυτής. Αφού ο χρήστης πιστοποιηθεί (όπως και στο Σχήμα 3.5), τότε μπορεί να διαχειριστεί το FlowSpace των slices στα οποία έχει πρόσβαση και ίσως να το διευρύνει προσθέτοντας περαιτέρω κανόνες, πάντα όμως περιορισμένος από την πολιτική που έχει θέσει ο διαχειριστής για τον συγκεκριμένο χρήστη. Και στις δύο περιπτώσεις, οποιαδήποτε ενέργεια του εκάστοτε χρήστη, θα ενεργοποιήσει την αντίστοιχη διαδικασία η οποία δεν είναι τίποτα άλλο παρά ένα κομμάτι του επιπέδου View της αρχιτεκτονικής του Django. Ακολούθως, αφού επεξεργαστούν τα στοιχείων που θα δώσει ή θα ζητήσει ο χρήστης, θα πραγματοποιηθούν οι αντίστοιχες ενέργειες στον FlowVisor μέσω της κονσόλας του και παράλληλα θα εμφανιστεί η αντίστοιχη ιστοσελίδα στον σελιδομετρητή (browser) του χρήστη.



Σχήμα 3.5: Διαδικασία ελέγχου του FlowVisor από τον διαχειριστή, κάνοντας χρήση του Web-UI (έχει βέβαια και τη δυνατότητα να παραμετροποιήσει τον FlowVisor απευθείας από την κονσόλα του)

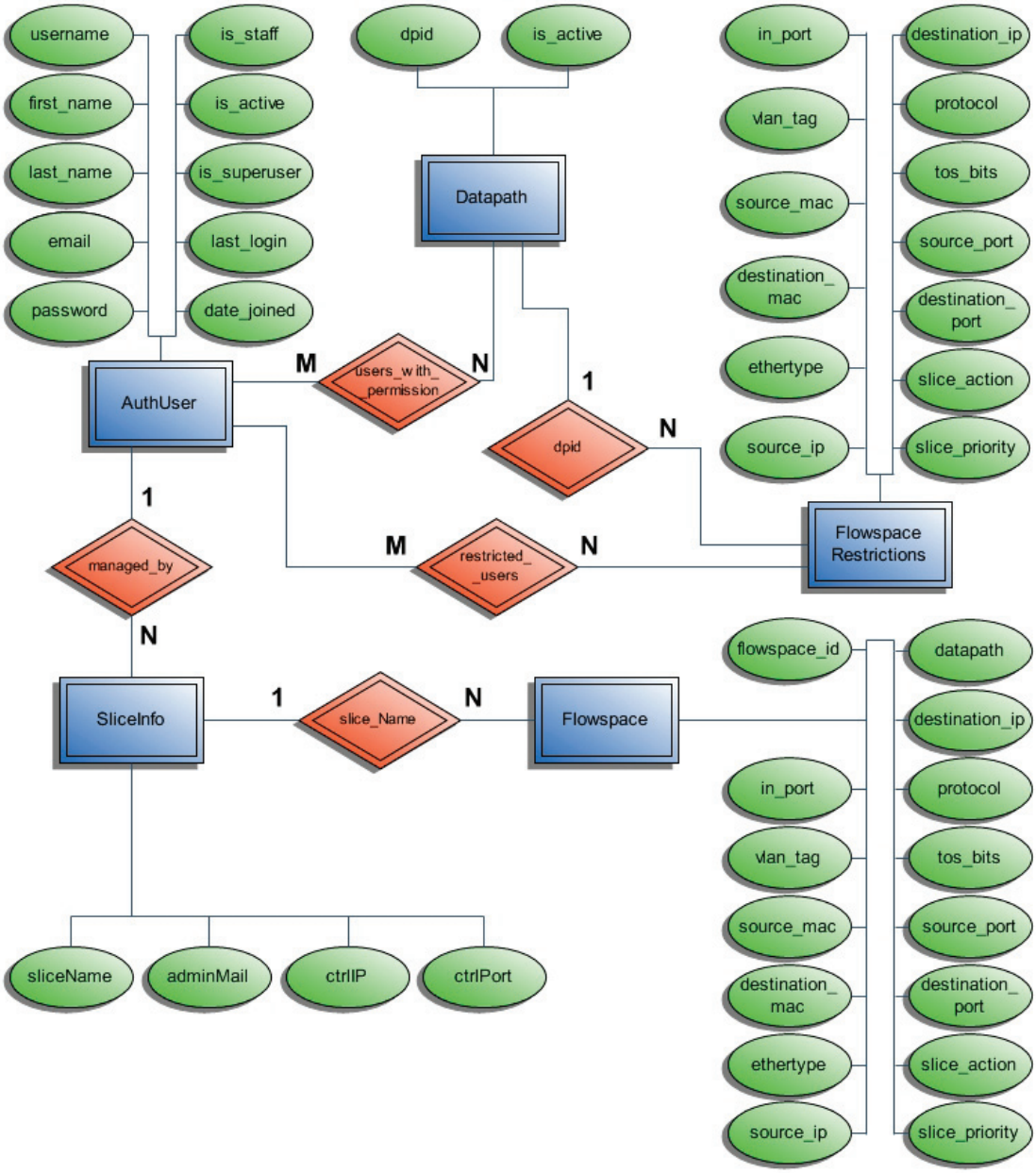


Σχήμα 3.6: Διαδικασία ελέγχου του FlowVisor από έναν χρήστη μέσω του Web-UI

3.3 Μοντέλο Οντοτήτων Συσχετίσεων

Για τη σωστή και ολοκληρωμένη λειτουργία του, το Web-UI έχει την ανάγκη μίας Βάσης Δεδομένων η οποία θα κρατάει αποθηκευμένες όλες τις απαραίτητες πληροφορίες και στην οποία θα έχει πρόσβαση το Django μέσω τον εφαρμογών του, για την άντληση των πληροφοριών αυτών, ή την τροποποίησή τους, σύμφωνα με τις επιταγές του εκάστοτε χρήστη. Στην παράγραφο αυτή παρουσιάζεται ένα σύστημα για την περιγραφή των πληροφοριών αυτών, το οποίο είναι γνωστό ως Μοντέλο Οντοτήτων-Συσχετίσεων (Entities-Relationships Model - ERM) και απεικονίζεται με το αντίστοιχο διάγραμμα (ERD) [16].

Ως Οντότητα ορίζεται ένα αντικείμενο το οποίο παρουσιάζει ενδιαφέρον για την εφαρμογή ιστού, ξεχωρίζει από τα υπόλοιπα αντικείμενα, και παράλληλα λειτουργεί αφαιρετικά σε έναν πολύπλοκο τομέα. Ο τύπος κάθε Οντότητας ορίζεται από μία συλλογή χαρακτηριστικών (attributes), τα οποία και την περιγράφουν. Τέλος, Συσχέτιση αποκαλείται η σύνδεση δύο ή και περισσότερων οντοτήτων μεταξύ τους, η οποία παρουσιάζει ενδιαφέρον για τον σχεδιασμό.



Σχήμα 3.7 : Διάγραμμα Οντοτήτων-Συσχετίσεων (ERD) της Βάσης Δεδομένων που χρησιμοποιείται μέσω του Django για την λειτουργία του Web-UI

Στο Σχήμα 3.7 φαίνεται το Διαγραμμα Οντοτήτων-Συσχετίσεων (ERD) που σχεδιάστηκε για την ανάλυση των απαιτήσεων του FlowVisor Web-UI. Γενικά σε ένα ERD οι Οντότητες παρουσιάζονται ως παραλληλόγραμμα, τα χαρακτηριστικά τους ως ελλείψεις και οι Συσχετίσεις ως ρόμβοι. Επιπλέον, στα άκρα κάθε σύνδεσης δύο Οντοτήτων αναφέρεται η πληθικότητα (cardinality), δηλαδή ο αριθμός στιγμιοτύπων ενός τύπου

Οντοτήτων που μπορούν να αντιστοιχίζονται με μία οντότητα ενός άλλου τύπου. Διακρίνονται δε, τρία είδη συσχετίσεων:

i. 1-1 (ένα-προς-ένα): Μία Οντότητα ενός τύπου αντιστοιχίζεται το πολύ με μία οντότητα ενός άλλου τύπου.
ii. 1-N (ένα-προς-πολλά): Μία Οντότητα ενός τύπου μπορεί να αντιστοιχιθεί με πολλά στιγμιότυπα μιας Οντότητας άλλου τύπου.
iii. M-N (πολλά-προς-πολλά): Κάθε στιγμιότυπο μίας Οντότητα ενός τύπου μπορεί να αντιστοιχηθεί με πολλά στιγμιότυπα μίας Οντότητας άλλου τύπου.

4

Σχεδίαση Συστήματος

Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, για την υλοποίηση του Web User Interface, απαιτείται η χρήση τριών βασικών υποσυστημάτων - εργαλείων, του FlowVisor, του Django και ενός Web Sever. Σε αυτό το κεφάλαιο θα δούμε τελικά πώς συνεργάζονται ακριβώς αυτά τα υποσυστήματα και ποιές διαδικασίες υλοποιεί το καθένα, ώστε να παρέχεται στον διαχειριστή ή τον χρήστη η επιθυμητή υπηρεσία.

4.1 Υλοποίηση Διεπαφής Χρήστη

Το Django, όντας ένα πλαίσιο ανάπτυξης διαδικτυακών εφαρμογών σε γλώσσα Python [17], ακολουθεί την αρχιτεκτονική MTV (Model - Template - View), όπως είδαμε στο προηγούμενο κεφάλαιο. Για τον λόγο αυτό, η υλοποίηση του επιθυμητού Web - UI, όπως και οποιασδήποτε άλλης διαδικτυακής εφαρμογής, στηρίζεται κυρίως σε τέσσερα αρχεία κώδικα Python και επιπλέον, σε κάποια σχεδιότυπα (template) HTML τα οποία καθορίζουν πώς θα εμφανίζονται κάποιες πληροφορίες στον τελικό χρήστη. Πιο συγκεκριμένα, ο κορμός της λειτουργίας της διαδικτυακής εφαρμογής αποτελείται από τα αρχεία:

i. **models.py :** Το αρχείο αυτό περιλαμβάνει την περιγραφή των πινάκων της βάσης δεδομένων, οι οποίοι αναπαρίστανται ως κλάσεις της Python. Κάθε τέτοια κλάση ορίζεται ως "μοντέλο" (model) και ουσιαστικά αντιπροσωπεύει μία Οντότητα, όπως αυτή ορίστηκε στο προηγούμενο κεφάλαιο. Μέσω των κλάσεων αυτών μπορούμε να δημιουργούμε, να ανασύρουμε, να ανανεώνουμε και να διαγράφουμε εγγραφές της βάσης δεδομένων χρησιμοποιώντας κώδικα Python και όχι δηλώσεις τύπου SQL.

Αξίζει δε να σημειωθεί πως το συγκεκριμένο αρχείο αντιπροσωπεύει το στοιχείο Model της αρχιτεκτονικής MTV που ακολουθεί το Django.

ii. **views.py :** Το αρχείο αυτό αποτελεί ουσιαστικά την λογική εργασίας για την παραγωγή κάθε ιστοσελίδας και αποτελείται από μεθόδους, κάθε μία από τις οποίες αποτελεί ένα διαφορετικό View. Κάθε τέτοια μέθοδος δέχεται ως παράμετρο ένα αντικείμενο που περιέχει όλες τις πληροφορίες σχετικά με την εντολή του χρήστη η οποία ενεργοποίησε τη συγκεκριμένη συνάρτηση, δηλαδή το συγκεκριμένο View. Το συγκεκριμένο αρχείο, όπως άλλωστε είναι φανερό και από το όνομά του, αντιπροσωπεύει το στοιχείο View της αρχιτεκτονικής MTV.

iii. **urls.py :** Το αρχείο αυτό περιλαμβάνει τις αντιστοιχίσεις μεταξύ των URL που θα ζητήσει ο χρήστης μέσω των επιλογών του στον browser, με τις αντίστοιχες συναρτήσεις του αρχείου views.py.

iv. **admin.py :** Μέσω του αρχείου αυτού υλοποιείται η αυτοματοποιημένη διεπαφή διαχειριστή (admin interface) της διαδικτυακής εφαρμογής, χρησιμοποιώντας κάποιες έτοιμες προς χρήση διαδικασίες που παρέχει το Django στους χρήστες του. Μέσω του αρχείου αυτού καθορίζουμε ποια μοντέλα από αυτά που έχουμε δημιουργήσει στο αρχείο models.py μπορούν να επεξεργαστούν μέσω της διεπαφής διαχειριστή. Ακόμη, για κάθε ένα από αυτά τα μοντέλα δημιουργούμε μία κλάση, η οποία περιλαμβάνει όλες τις επιτρεπτές λειτουργίες για το συγκεκριμένο μοντέλο. Τέλος, καθορίζει τον τρόπο με τον οποίο θα εμφανίζονται αυτές οι κλάσεις στον διαχειριστή και σε όσους έχουν πρόσβαση στο συγκεκριμένο interface της διαδικτυακής εφαρμογή.

Εκτός αυτών των αρχείων και των templates που δημιουργήθηκαν, υπάρχουν τρία ακόμη αρχεία κώδικα Python τα οποία στην ουσία παρέχουν τις μεθόδους διασύνδεσης του Web-UI με την κονσόλα του FlowVisor. Τα αρχεία αυτά είναι:

i. **utilities.py :** Σκοπός αυτού του αρχείου είναι η ανάγνωση των slices που έχουν ήδη δημιουργηθεί στον FlowVisor και η αποθήκευσή τους σε μια δομή δεδομένων τύπου λεξικού. Ακόμη, το συγκεκριμένο αρχείο παρέχει τις συναρτήσεις για τον έλεγχο των εγγραφών flowspace που θα προσπαθήσει κάποιος χρήστης να δημιουργήσει ή να τροποποιήσει ώστε να συμμορφώνονται με την πολιτική που έχει θεσπίσει ο διαχειριστής για τον συγκεκριμένο χρήστη.

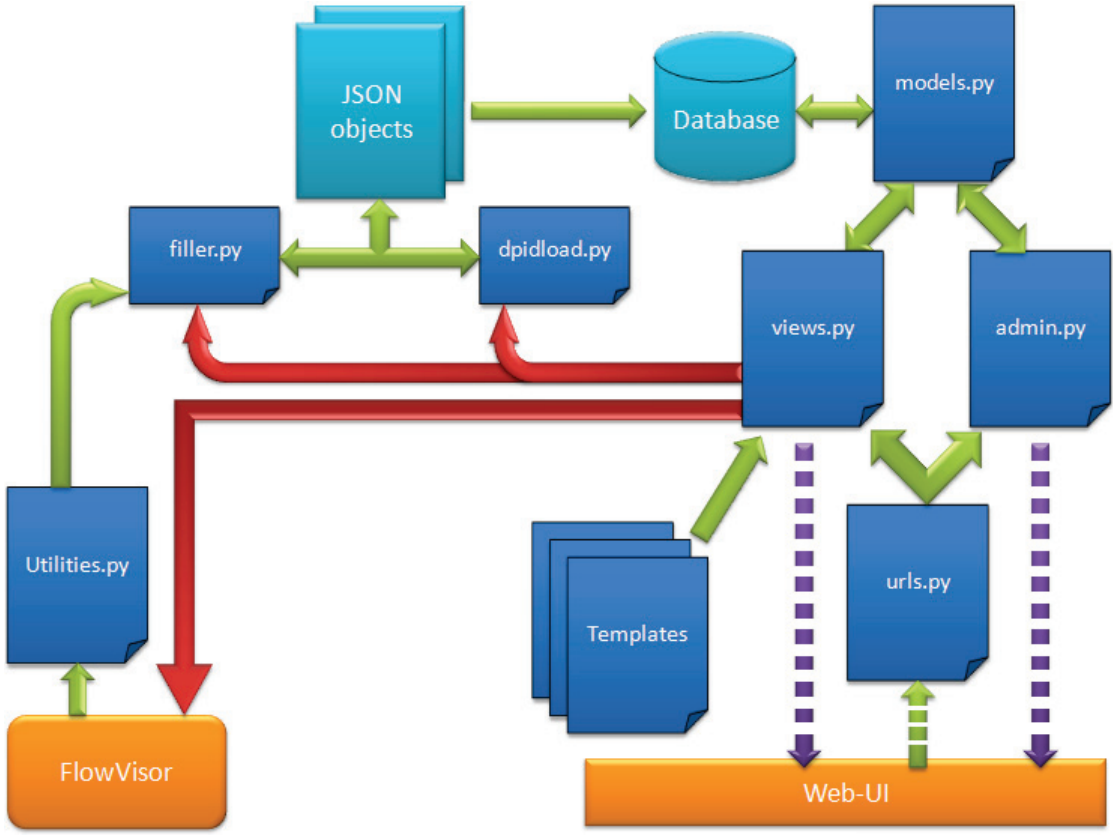
ii. **filler.py :** Αυτό το αρχείο, αφού χρησιμοποιήσει ως δεδομένο το λεξικό (dictionary) που δημιουργήθηκε μέσω του utilities.py, διαβάζει από τον FlowVisor τους κανόνες flowspace που έχουν δημιουργηθεί για κάθε ένα από τα slices που υπάρχουν. Ακολούθως, δημιουργεί μία δομή τύπου JSON (JavaScript Object Notation) [18] για όλα τα slices και άλλη μία για όλα τα flowspaces. Τέλος, αποθηκεύει κάθε μία από

αυτές τις δομές σε ένα αρχείο JSON. Το μορφότυπο (format) JSON είναι ένας ελαφρύς και εύχρηστος τρόπος διασύνδεση δεδομένων και παράλληλα εύκολα αναγνώσιμος από τον άνθρωπο. Οι δομές JSON είναι ενιαίες και ανεξάρτητες γλωσσών προγραμματισμού, ενώ υποστηρίζονται από όλες τις σύγχρονες γλώσσες, όπως η Python. Ένα αντικείμενο δομής JSON είναι ένα μη διατεταγμένο σύνολο ζευγαριών ονομάτων-τιμών, το οποίο διατηρεί μία συγκεκριμένη δομή. Χρησιμοποιώντας αυτές τις δομές JSON, μπορούμε να μεταφέρουμε τα δεδομένα που έχουμε συλλέξει από τον FlowVisor στην βάση δεδομένων, πράγμα που είναι και ο τελικός στόχος του συγκεκριμένου αρχείου.

iii. **dpidload.py :** Μέσω αυτού του αρχείου γίνεται η ανάγνωση των datapaths (OpenFlow switches) που είναι συνδεδεμένα με τον FlowVisor και ενημερώνεται σχετικά με αυτά η βάση δεδομένων

Στο Σχήμα 4.1 φαίνεται συνολικά ο τρόπος με το οποίο συνεργάζονται τα αρχεία (μέσω των μεθόδων τους) που αναφέρθηκαν παραπάνω, με τα υπόλοιπα υποσυστήματα για την παροχή της επιθυμητής υπηρεσίας στον τελικό χρήστη. Αρχικά, μέσω του αρχείου utilities.py αντλούνται τα απαραίτητα δεδομένα από τον FlowVisor, και αφού γίνουν οι απαραίτητες διαδικασίες μέσω του αρχείου filler.py και αντληθούν οι πληροφορίες για τα flowspaces, δημιουργούνται τα αντίστοιχα αρχεία JSON τα οποία θα χρησιμοποιηθούν για την συμπλήρωση των αντίστοιχων πεδίων των πινάκων της βάσης δεδομένων. Το ίδιο γίνεται και για τις πληροφορίες σχετικά με τα datapaths μέσω του αρχείου dpidload.py. Οι πίνακες της βάσης δεδομένων αντιστοιχίζονται με συγκεκριμένες κλάσεις Python μέσω του αρχείου models.py, και στο εξής η οποιαδήποτε αλληλεπίδραση χρηστών με τις εγγραφές της βάσης δεδομένων γίνεται μέσω αυτών των κλάσεων και κατά συνέπεια μέσω των αρχείων views.py και admin.py τα οποία τις χρησιμοποιούν. Τη στιγμή που κάποιος χρήστης θα ζητήσει κάποια πληροφορία ή θα επιχειρήσει κάποια παραμετροποίηση του FlowVisor, αυτό αρχικά θα σημαίνει ότι ο χρήστης αυτός θα ζητήσει, μέσω του browser του, ένα συγκεκριμένο URL. Αυτό το URL θα επεξεργαστεί μέσω του αρχείου urls.py και είτε θα εμφανιστεί η κατάλληλη σελίδα της διεπαφής διαχειριστή μέσω του αρχείου admin.py, είτε θα συσχετιστεί η απαίτηση του χρήστη με κάποια από τις μεθόδους του αρχείου views.py. Στην δεύτερη περίπτωση, αρχικά θα εξεταστεί η πολιτική που περιορίζει το χρήστη μέσω του αρχείου views.py, και η οποία εξαρτάται από τις σχετικές πληροφορίες που είναι αποθηκευμένες στην βάση δεδομένων και από την επεξεργασία τους μέσω των μεθόδων του αρχείου utilities.py. Στη συνέχεια, αφού χρησιμοποιηθεί το κατάλληλο template, θα πραγματοποιηθεί η ανάλογη τροποποίηση του FlowVisor ή η εμφάνιση των ζητούμενων πληροφοριών στο Web-UI. Σε οποιαδήποτε από τις δύο περιπτώσεις, τα αρχεία views.py και admin.py είναι αυτά που θα

καθορίσουν το περιεχόμενο και την μορφοποίηση της τελικής ιστοσελίδας που θα παρουσιαστεί στον χρήστη.



Σχήμα 4.1 : Διασύνδεση των διαφόρων αντικειμένων που χρησιμοποιούνται για την υλοποίηση της αρχιτεκτονικής, και την αλληλεπίδραση μεταξύ του FlowVisor και του Web-UI. Με τα πράσινα βέλη συμβολίζεται η τροφοδότηση πληροφοριών, ενώ με τα κόκκινα συμβολίζεται η κλήση της αντίστοιχης μεθόδου. Τέλος με τα μωβ βέλη συμβολίζεται το αποτέλεσμα ενός request.

4.2 Αναλυτική περιγραφή των αρχείων υλοποίησης

4.2.1 Ανάλυση του αρχείου models.py

Όπως αναφέρθηκε παραπάνω, μέσω αυτού του αρχείου γίνεται η συσχέτιση των πινάκων της βάσης δεδομένων με κλάσεις της Python, με αντιστοιχία ένα προς ένα. Κάθε μία από τις κλάσεις έχει ένα σύνολο πεδίων, κάθε ένα από τα οποία αντιστοιχεί σε μία στήλη του αντίστοιχου πίνακα της βάσης δεδομένων, και καθορίζει τον τύπο των τιμών που αποθηκεύονται εκεί, δηλαδή αν οι εγγραφές της στήλης αυτής θα είναι αριθμοί, χαρακτήρες, λογικά αντικείμενα τύπου true/false, κλειδιά σύνδεσης του πίνακα αυτού με το πεδίο ενός ή πολλών άλλων πινάκων, κ.α. Ακόμη, μέσα στην κλάση ορίζεται μία συνάρτηση η οποία καθορίζει ποιό από τα πεδία του μοντέλου αυτού θα είναι το αντιπροσωπευτικό και κατά συνέπεια αυτό που θα εμφανίζεται και θα υποδηλώνει μία γενίκευση αυτού του μοντέλου.

Μία όμως ακόμη σημαντική λειτουργία που υλοποιείται μέσω αυτού του αρχείου, είναι η χρήση σημάτων (signals) [19]. Τα σήματα αυτά δίνουν τη δυνατότητα σε συγκεκριμένες διαδικασίες να ενημερώσουν άλλες διαδικασίες για το τι πρόκειται να συμβεί, πριν προβούν σε κάποια ενέργεια.

Στον Κώδικα 4.1 φαίνεται η μοντελοποίηση του πίνακα της βάσης δεδομένων που διατηρεί τις πληροφορίες όλων των slices. Οι στήλες που απαρτίζουν τον πίνακα αυτόν είναι:

- sliceName : Είναι εγγραφές τύπου χαρακτήρα όπως ορίζεται μέσω της υποκλάσης *CharField* της κλάσης models και διατηρούν την ονομασία κάθε ενός από τα slices
- adminMail : Είναι εγγραφές τύπου χαρακτήρα και διατηρούν το e-mail του διαχειριστή - ιδιοκτήτη κάθε ενός από τα slices
- ctrlIp : Είναι και αυτές εγγραφές χαρακτήρων και αποθηκεύονται σε αυτές οι διευθύνσεις IP κάθε ενός Controller που ελέγχει ένα συγκεκριμένο slice
- ctrlPort : Είναι πεδία ακέραιων αριθμών όπως φαίνεται και από την υποκλάση IntegerField που χρησιμοποιείται, και αποθηκεύουν το port στο οποίο "ακούει" ο Controller κάθε ενός από τα slices.
- managed_by : Είναι πεδία που συσχετίζουν κάθε σειρά του πίνακα της βάσης δεδομένων με τον λογαριασμό που έχει δημιουργήσει ο διαχειριστής του δικτύου για τον διαχειριστή - ιδιοκτήτη κάθε ενός slices. Ο τύπος αυτού του πεδίου ορίζεται μέσω της υποκλάσης *ForeignKey*, ενώ το μοντέλο με το οποίο συσχετίζεται είναι το *User*.

Τέλος, στον κώδικα φαίνεται η συνάρτηση η οποία καθορίζει την στήλη που θα "αντιπροσωπεύει" το συγκεκριμένο μοντέλο, και κυρίως η συνάρτηση αυτή χρησιμοποιείται από το αρχείο admin.py για τον καθορισμό του τρόπου παρουσίασης του μοντέλου αυτού.

```
class SliceInfo(models.Model):

    sliceName = models.CharField(max_length=200, verbose_name='Slice Name')
    adminMail = models.CharField(max_length=200, verbose_name='Administrator\'s e-Mail')
    ctrlIp = models.CharField(max_length=200, verbose_name='Controller\'s IP')
    ctrlPort = models.IntegerField(verbose_name='Controller\'s Port')
    managed_by = models.ForeignKey(User, null=True, blank=True)

    def __unicode__(self):
        return self.sliceName
```

Κώδικας 4.1 : Μοντελοποίηση του πίνακα της βάσης δεδομένων που περιέχει τα στοιχεία των slices

Με την ίδια διαδικασία ορίζονται και τα υπόλοιπα μοντέλα που χρειάζονται για τον συσχετισμό με τους αντίστοιχους πίνακες της βάσης δεδομένων, τα οποία είναι τα εξής:

- FlowSpace
- Datapath
- FlowspaceRestriction

Περαιτέρω ανάλυση των παραπάνω μοντέλων θα γίνει στην ενότητα 4.3 του κεφαλαίου αυτού. Είναι σημαντικό ωστόσο να αναφερθεί ότι στο συγκεκριμένο αρχείο δεν εμφανίζεται πουθενά η υλοποίηση του μοντέλου User το οποίο αντιστοιχεί με έναν πίνακα της βάσης δεδομένων και διατηρεί τα στοιχεία όλων των χρηστών και διαχειριστών του Web-UI. Αυτό συμβαίνει γιατί το μοντέλο αυτό είναι καθορισμένο και υλοποιημένο εξαρχής από το ίδιο το Django, μιας και παρέχει μια υπηρεσία απαραίτητη για σχεδόν οποιαδήποτε εφαρμογή ιστού.

Η χρήση των σημάτων (signals) παίζει καθοριστικό ρόλο στη μοντελοποίηση της βάσης δεδομένων και τη διατήρηση της ταύτισης των πληροφοριών της βάσης δεδομένων, με τις πληροφορίες που διατηρεί ο FlowVisor. Στον Κώδικα 4.2 φαίνεται η διαδικασία που πραγματοποιείται λόγω του σήματος που παράγεται όταν κάποιος χρήστης επιλέξει να διαγράψει κάποιο slice μέσω του Web-UI. Το σήμα αυτό ονομάζεται pre-delete, και αυτό γιατί ενημερώνει την συνάρτηση που εμφανίζεται στον Κώδικα 4.2 για διαγραφή κάποιων πληροφοριών από τη βάση δεδομένων, πριν αυτή πραγματοποιηθεί. Σε μία τέτοια περίπτωση, τα βήματα που πραγματοποιούνται είναι τα εξής:

1. Αναγνώριση του/των slice/slices που επέλεξε ο χρήστης προς διαγραφή.

2. Διαγραφή του/των αντίστοιχων slice/slices από τον FlowVisor (εδώ πρέπει να σημειωθεί πως όταν διαγραφεί ένα slice από τον FlowVisor, διαγράφονται ταυτόχρονα και οι αντίστοιχες εγγραφές κανόνων flowspace, ενώ το ίδιο θα συμβεί και στην βάση δεδομένων μας στο τελευταίο βήμα, αφού το μοντέλο FlowSpace είναι συσχετισμένο με το μοντέλο sliceInfo).
3. Διαγραφή της γραμμής που αφορά το συγκεκριμένο slice από το αρχείο *record* το οποίο αποθηκεύει εξωτερικά κάποιες κωδικοποιημένες πληροφορίες, και το οποίο θα αναλυθεί στην ενότητα 4.4 του κεφαλαίου.
4. Πλέον η απαίτηση του χρήση για διαγραφή κάποιου slice μπορεί να ολοκληρωθεί με τη διαγραφή του αντίστοιχου slice από τη βάση δεδομένων

```
def delete_sl(sender, **kwargs):
  name=kwargs['instance'].sliceName
  cmd='fvctl --passwd-file=/home/flowvisor07/fv-ui/fv_site/slices/rootutil deleteSlice '+str(name)
 (command_output,exitstatus)=run (cmd, withexitstatus=1)
  time.sleep(1)
  f = open('/home/flowvisor07/fv-ui/fv_site/slices/record','r')
  newdata=''
  for line in f:
    finder=line.rfind('=')
    if line[0:finder]<>name:
      newdata += line
  f.close()
  f = open('/home/flowvisor07/fv-ui/fv_site/slices/record','w')
  f.write(newdata)
  f.close()
pre_delete.connect(delete_sl, sender=SliceInfo)
```

Κώδικας 4.2 : Καθορισμός σήματος για την διαγραφή των slices και των αντίστοιχων flowspaces από το σύστημα του FlowVisor, πριν αυτά διαγραφούν οριστικά από τη βάση δεδομένων του Web-UI

Στο αρχείο models.py υλοποιείται με τον ίδιο τρόπο ένα ακόμη σήμα, αποκλειστικά για την διαγραφή εγγραφών flowspace από τον FlowVisor όταν ένας χρήστης ζητήσει την διαγραφή ενός η περισσοτέρων κανόνων flowspace μέσω του Web-UI.

4.2.2 Ανάλυση του αρχείου views.py

Μέσω των μεθόδων του αρχείου views.py γίνεται η προσθήκη ή η τροποποίηση ενός flowspace ή ενός slice, η ανανέωση της βάσης δεδομένων, καθώς και κάποιες διαχειριστικές

λειτουργίες του FlowVisor όπως εκκίνηση ή παύση της λειτουργίας του. Οι παραπάνω λειτουργίες πραγματοποιούνται μέσω έντεκα μεθόδων, οι οποίες περιγράφονται παρακάτω.

Μέθοδοι *createsl* και *cslice*

Μέσω της μεθόδου createsl γίνεται ο έλεγχος της πολιτικής του χρήστη ο οποίος επιθυμεί να δημιουργήσει ένα καινούριο slice. Αν ο χρήστης αυτό είναι ο διαχειριστής ή έχει διαχειριστικά δικαιώματα, τότε μπορεί να προχωρήσει στην διαδικασία δημιουργίας του slice, μέσω του html template *create_slice.html* που θα εμφανιστεί. Σε αντίθετη περίπτωση απλά ενημερώνεται ότι δεν έχει τα απαραίτητα δικαιώματα για την συγκεκριμένη ενέργεια.

Στην περίπτωση ο χρήστης περάσει τον έλεγχο και αφού δώσει τα στοιχεία που θα χαρακτηρίζουν το καινούριο slice, τότε αναλαμβάνει η μέθοδος cslice την υλοποίηση της απαίτησης του χρήστη. Η μέθοδος αυτή, όπως φαίνεται και στον Κώδικα 4.3 πραγματοποιεί τις ακόλουθες ενέργειες:

1. Ανάγνωση των στοιχείων του slice που πρόκειται να δημιουργηθεί, όπως τα έδωσε ήδη ο διαχειριστής.
2. Προσθήκη νέας γραμμής στο αρχείο record που αναφέρει ποιος χρήστης θα έχει δικαίωμα ελέγχου του slice που πρόκειται να δημιουργηθεί.
3. Δημιουργία του καινούριου slice μέσω της εφαρμογής fvctl του FlowVisor και ανάγνωση αποτελέσματος.
4. Αν η δημιουργία του καινούριου slice ήταν επιτυχής, τότε ενημερώνεται η βάση δεδομένων του Web-UI με την προσθήκη νέας εγγραφής στο μοντέλο sliceInfo, η οποία θα περιέχει όλα τα στοιχεία του συγκεκριμένου slice, δηλαδή όνομα, διεύθυνση IP, port, e-mail του χρήστη που θα το διαχειρίζεται και το κλειδί αντιστοίχισης του χρήστη αυτού με το μοντέλο User. Σε περίπτωση αποτυχίας ο διαχειριστής ενημερώνεται για την αποτυχία της ενέργειάς του.
5. Το καινούριο slice είναι πλέον έτοιμο προς χρήση και μπορεί στο εξής θα ελέγχεται μόνο από τον διαχειριστή του δικτύου και από τον χρήστη στον οποίο ανέθεσε ο διαχειριστής τον έλεγχο του slice αυτού.

```
def cslice(request):
  name = request.GET['slicename']
  ip = request.GET['ip']
  port = request.GET['port']
  mail = request.GET['mail']
  userer = request.GET['username']
  f = open('/home/flowvisor07/fv-ui/fv_site/slices/record','a')
  record=name+'='+userer+'\n'
  f.write(record)
  fullip='tcp:'+ip+':'+port
  cmd='fvctl '+'--passwd-file=/home/flowvisor07/fv-ui/fv_site/slices/rootutil '+'createSlice '+name+'
'+fullip+' '+mail
  (command_output,exitstatus)=run (cmd, events={'(?i)New password:':'pass\n'}, withexitstatus=1)
  outpt=command_output.splitlines()
  result=outpt[1]
  if result=='success!':
    p=User.objects.get(username=userer)
    user_id=p.id
    a=User.objects.get(pk=user_id)
    b=a.sliceinfo_set.create(sliceName=name,adminMail=mail, ctrlIp=ip, ctrlPort=port)
    b.save()
    b=SliceInfo.objects.get(sliceName=name)
    b=b.id
    return HttpResponseRedirect('/admin/slices/sliceinfo/%s/' % str(b))
  else:
    return HttpResponse("Creation Failed")
```

Κώδικας 4.3 : Μέθοδος για τη δημιουργία ενός slice στον FlowVisor από τον διαχειριστή το δικτύου

Μέθοδοι *createfs* και *cflowspace*

Οι δύο αυτές μέθοδοι λειτουργούν κατ' αντιστοιχία με τις μεθόδους createsl και cslise, με τη διαφορά όμως ότι τελικός στόχος είναι η δημιουργία ενός κανόνα flowspace και όχι ενός slice, όπως άλλωστε φαίνεται και από την ονομασία τους. Μέσω της μεθόδου createfs γίνεται ο έλεγχος της πολιτικής του χρήστη που επιθυμεί να δημιουργήσει έναν νέο κανόνα flowspace. Σε κάθε κανόνα πρέπει να οριστεί, όπως έχει αναφερθεί και σε

προηγούμενο κεφάλαιο, το όνομα του slice στο οποίο θα "ανήκει" ο συγκεκριμένος κανόνας, καθώς και το datapath που θα αφορά. Έτσι μέσω της μεθόδου createfs ελέγχεται αν ο χρήστης είναι διαχειριστής ή όχι. Αν το αποτέλεσμα είναι θετικό, τότε έχει τη δυνατότητα να δημιουργήσει έναν καινούριο κανόνα σε οποιοδήποτε slice και χρησιμοποιώντας οποιοδήποτε datapath, ενώ αν το αποτέλεσμα είναι αρνητικό ο καινούριος κανόνας θα πρέπει να συμμορφώνεται με την πολιτική που έχει ορίσει ο διαχειριστής για τον χρήστη αυτόν. Για να υλοποιηθεί αυτός ο έλεγχος, δημιουργείται μία δομή δεδομένων τύπου dictionary, η οποία αναλόγως με την κατηγορία στην οποία ανήκει ο χρήστης (δηλαδή superuser ή staff user) περιέχει τα slices και datapaths που μπορεί να χρησιμοποιήσει στον κανόνα αυτόν. Στη συνέχεια εμφανίζεται στον browser του χρήστη η ιστοσελίδα για τη δημιουργία του καινούριου κανόνα flowspace, μέσω του template *create_flowspace.html*, και το οποίο χρησιμοποιεί σαν δεδομένο το dictionary που έχει ήδη δημιουργηθεί.

Στη συνέχεια, και αφού ο χρήστης έχει συμπληρώσει τα στοιχεία που θα χαρακτηρίζουν τον καινούριο κανόνα flowspace, αναλαμβάνει η μέθοδος cflowspace για τον έλεγχο των τιμών που έχει δώσει ο χρήστης, καθώς και για την τελική υλοποίηση της απαίτησής του. Αξίζει να σημειωθεί ότι μέσω αυτής της μεθόδου γίνεται εκτός από την δημιουργία καινούριου flowspace και η τροποποίηση ενός flowspace που έχει ήδη δημιουργηθεί. Πιο συγκεκριμένα, τα βήματα που ακολουθούνται είναι τα εξής:

1. Ανάγνωση των τιμών που έδωσε ο χρήστης για την δημιουργία του καινούριου κανόνα flowspace και έλεγχος των τιμών αυτών σύμφωνα με τους περιορισμούς που έχει καθορίσει ο διαχειριστής για τον συγκεκριμένο χρήστη. Ακόμη ελέγχεται αν οι τιμές που έδωσε ο χρήστης είναι σε λογικά πλαίσια. Ο έλεγχος αυτός γίνεται χρησιμοποιώντας τις μεθόδους Check_num και Check_ip του αρχείου utilities.py, ενώ ένα παράδειγμα τέτοιου ελέγχου για το πεδίο source_IP φαίνεται στον Κώδικα 4.4.
2. Έλεγχος αν η μέθοδος αυτή θα χρησιμοποιηθεί για δημιουργία ή τροποποίηση ενός κανόνα flowspace και εκτέλεση της αντίστοιχης εντολής, μέσω της εφαρμογής fvctl του FlowVisor.
3. Έλεγχος για την επιτυχία ή αποτυχία της ενέργειας.
4. Σε περίπτωση επιτυχίας, ανανεώνεται η βάση δεδομένων του Web-UI τροποποιώντας τα αντίστοιχα πεδία μιας εγγραφής αν ο χρήστης επιχείρησε κάτι τέτοιο, ή αλλιώς προσθέτοντας μία καινούρια εγγραφή. Σε περίπτωση αποτυχίας επιστρέφεται ένα σχετικό μήνυμα στον χρήστη, όπου αναφέρεται ποιό πρόβλημα υπήρξε.
5. Σε περίπτωση επιτυχίας εμφανίζεται στον browser του χρήστη μία νέα ιστοσελίδα που περιέχει τον κανόνα που μόλις δημιούργησε ή τροποποίησε.

```
flow[6] = request.GET['source_IP']
    if not request.user.is_superuser and not rest=="free":
     result = Check_ip(str(b.source_IP),str(flow[6]))
     if result=="decline":
       return HttpResponse('Selected source IP not permitted!')
```

Κώδικας 4.4 : Έλεγχος του πεδίου source_IP μιας εγγραφής κανόνα flowspace που πρόκειται να δημιουργηθεί ή να τροποποιηθεί

Μέθοδος *changefs* και *changefsfinal*

Μέσω της μεθόδου και χρησιμοποιώντας το template *change_flowspace.html* δημιουργείται μία ιστοσελίδα την οποία βλέπει ο χρήστης μέσω του browser του, και από εκεί μπορεί να επιλέξει ποιον κανόνα flowspace επιθυμεί να αλλάξει. Όπως φαίνεται και στον Κώδικα 4.5, η διαδικασία που υλοποιείται είναι:

1. Δημιουργείται ενός dictionary με όλα τα slices που υπάρχουν στην βάση δεδομένων
2. Έλεγχος και επιλογή των slices στα οποία έχει δικαιοδοσία ο συγκεκριμένος χρήστης
3. Δημιουργία νέου λεξικού το οποίο περιέχει το αναγνωριστικό (flowspace_id) κάθε εγγραφής flowspace, το datapath το οποίο αφορά και το όνομα του slice στο οποίο ανήκει.
4. Τελικά μέσω του template που αναφέρθηκε παραπάνω, και του τελευταίου dictionary που δημιουργήθηκε, παράγεται η επιθυμητή ιστοσελίδα.

Πλέον, αφού επιλέξει ο χρήστης ποιον κανόνα επιθυμεί να τροποποιήσει, αναλαμβάνει η μέθοδος changefsfinal, χρησιμοποιώντας το template *flowspace_change_form.html* να δημιουργήσει μια φόρμα ή οποία θα έχει ήδη συμπληρωμένες τις τιμές των πεδίων του κανόνα flowspace που έχει επιλεγεί, και μέσω της οποίας ο χρήστης μπορεί να πραγματοποιήσει την τροποποίηση που επιθυμεί. Στην φόρμα αυτή, όπως και κατά την δημιουργία ενός flowspace, τα διαθέσιμα πεδία που αφορούν το όνομα του slice στο οποίο θα ανήκει ο συγκεκριμένος κανόνας καθώς και το datapath, είναι περιορισμένα με τέτοιον τρόπο ώστε να επιβάλλεται η πολιτική που έχει ορίσει ο διαχειριστής για τον χρήστη που προσπαθεί να τροποποιήσει τον κανόνα. Στον Κώδικα 4.6 φαίνεται πως γίνεται ο έλεγχος της πολιτικής του χρήστη. Από την στιγμή που θα ολοκληρώσει τις ενέργειές της η μέθοδος αυτή, θα αναλάβει η μέθοδος cflowspace για τον περεταίρω έλεγχο, αλλά και την τελική υλοποίηση της τροποποίησης του κανόνα flowspace που επιλέχτηκε.

```
def changefs(request):
  userer=request.user.username
  flow_dict={}
  slice_dict=SliceInfo.objects.all()
  message = {}
  for i in range (0,len(slice_dict)):


    a=SliceInfo.objects.get(sliceName=str(slice_dict[i]))
    if a.managed_by == request.user:
       b=FlowSpace.objects.filter(slice_Info=a.id)
       for j in range(0, len(b)):
          z=FlowSpace.objects.get(flow_space_id=b[j], slice_Info=a.id)
          message[b[j]]='SLICE : '+str(a)+'|'*(30-len(str(a)))+' DATAPATH : '+str(z.datapath)
  return
render_to_response('change_flowspace.html',{'flowspaces':message,'user':userer,'total':len(message)})
```

Κώδικας 4.5 : Μέθοδος για την επιλογή και παρουσίαση των κανόνων flowspace που μπορεί να τροποποιήσει ένας συγκεκριμένος χρήστης

```
if request.user.is_superuser:
     slice_dict=SliceInfo.objects.all()
     b=Datapath.objects.all()
     dpid_dict=b.filter(is_active='up')
   elif request.user.is_staff:
     slice_dict=SliceInfo.objects.filter(managed_by=request.user)
     b=Datapath.objects.filter(users_with_permission=request.user)
     dpid_dict=b.filter(is_active='up')
   else:
     return HttpResponse('You don\'t have permission to do that')
```

Κώδικας 4.6 : Έλεγχος μέσω της μεθόδου changefsfinal για τα slices και datapaths που είναι διαθέσιμα στον εκάστοτε χρήστη, σύμφωνα με την πολιτική που έχει οριστεί για αυτόν

Μέθοδοι *power* και *power_do*

Σκοπός των μεθόδων power και power_do είναι να παρέχουν στον διαχειριστή του δικτύου τη δυνατότητα παύσης, εκκίνησης ή επανεκκίνησης του FlowVisor μέσω του Web-UI. Μέσω της μεθόδου power γίνεται έλεγχος αν ο χρήστης είναι ο διαχειριστής του δικτύου, ή αν έχει αντίστοιχα δικαιώματα, και στην περίπτωση που το αποτέλεσμα είναι θετικό, χρησιμοποιείται το template *power_choose.html* ώστε να μπορέσει ο χρήστης να επιλέξει μια ενέργεια εκ των τριών επιθυμεί να πραγματοποιήσει.

Στη συνέχεια αναλαμβάνει η μέθοδος power_do να πραγματοποιήσει την ενέργεια που επέλεξε ο χρήστης, όπως φαίνεται και στον Κώδικα 4.7. Αφού πραγματοποιηθεί η ενέργεια αυτή, ο χρήστης μεταφέρεται στην αρχική σελίδα (" .../admin/index.html ").

```
def power_do(request):
 if not request.user.is_superuser:
  return HttpResponseRedirect("http://localhost:8000/admin/auth/user/")
 else:
  act=request.GET['selected_action']
  cmdstop="/etc/init.d/flowvisor stop"
  cmdstart="/etc/init.d/flowvisor start"
  if act=='on':
    run (cmdstart)
    message="FLOWVISOR WAS POWERED ON!"
  elif act=='off':
    run (cmdstop)
    message="FLOWVISOR WAS POWERED OFF!"
  elif act=='restart':
    run (cmdstop)
    time.sleep(2)
    run (cmdstart)
    message="FLOWVISOR WAS RESTARTED!"
  else:
    message='NO ACTION SELECTED'
  apps=applist(request)
  url='../../'
  return render_to_response('admin/index.html',{'msg':message, 'user':request.user, 'app_list':apps,'urls':url})
```

Κώδικας 4.7 : Υλοποίηση της επιλογής του διαχειριστή για εκκίνηση, παύση ή επανεκκίνηση της λειτουργίας του FlowVisor

Μέθοδος *filldb*

Όπως έχει ήδη αναφερθεί, επιθυμούμε να μην χάσει ο διαχειριστής την δυνατότητα ελέγχου του FlowVisor απευθείας μέσω της κονσόλας του. Παράλληλα όμως πρέπει οι ρυθμίσεις που είναι αποθηκευμένες στη βάση δεδομένων του Web-UI να συμβαδίζουν πλήρως με τις ρυθμίσεις οι οποίες ισχύουν στην πραγματικότητα στον FlowVisor. Οι δύο παραπάνω απαιτήσεις υλοποιούνται μέσω της μεθόδου filldb. Η μέθοδος αυτή επιτρέπει στον διαχειριστή να ανανεώνει, όποτε αυτός κρίνει σκόπιμο, τη βάση δεδομένων του Web-UI έτσι ώστε να συμβαδίζει πλήρως με τις τροποποιήσεις που έχουν ήδη πραγματοποιηθεί στον FlowVisor, είτε αυτές έγιναν μέσω του Web-UI, είτε απευθείας μέσω της κονσόλας. Πιο συγκεκριμένα οι ενέργειες που πραγματοποιούνται είναι:

1. Έλεγχος ώστε να επιβεβαιωθεί ότι ο χρήστης ο οποίος προσπαθεί να ανανεώσει τη βάση δεδομένων έχει το δικαίωμα να εκτελέσει μία τέτοια ενέργεια.
2. Αποθήκευση κάποιων πληροφοριών στο αρχείο *record2*. Ο σκοπός και οι πληροφορίες αυτού του αρχείου θα αναλυθούν στην τελευταία ενότητα του κεφαλαίου.
3. Εξαγωγή των δεδομένων των πινάκων Datapath και FlowspaceRestrictions, και αποθήκευσή τους υπό την μορφή αντικειμένων JSON στα αρχεία datapath.json και restrictions.json αντίστοιχα. Αυτό γίνεται γιατί αν και ο σκοπός αυτής της μεθόδου είναι η ανανέωση της βάσης δεδομένων όσον αφορά τα slices και τους κανόνες flowspace, στην πραγματικότητα η βάση δεδομένων θα αδειάσει και θα προστεθούν όλες οι εγγραφές εξαρχής. Κατά συνέπεια, πρέπει να υπάρχουν κάπου αποθηκευμένα τα στοιχεία που αφορούν τα μοντέλα Datapath και FlowspaceRestrictions.
4. Εκτέλεση του κώδικα Python που είναι αποθηκευμένος στο αρχείο filler.py το οποίο θα αναλυθεί παρακάτω.
5. Μετά την ολοκλήρωση εκτέλεσης του αρχείου filler.py, η βάση δεδομένων έχει πλέον ανανεωθεί και είναι πλήρως ενημερωμένη για τις υπάρχουσες ρυθμίσεις στον FlowVisor και ο χρήστης μεταφέρεται στην αρχική σελίδα λαμβάνοντας ένα μήνυμα που τον ενημερώνει για την ολοκλήρωση της διαδικασίας.

Μέθοδος *loaddpid*

Ο σκοπός αυτής της μεθόδου είναι παρόμοιος με αυτόν της μεθόδου filldb, αλλά αντί για τα slices και τους κανόνες flowspace, ανανεώνονται τα datapaths. Έτσι, γίνεται έλεγχος για το ποιά datapaths είναι συνδεδεμένα τη δεδομένη χρονική στιγμή με τον FlowVisor και αποθηκεύονται στην βάση δεδομένων. Όποια datapaths από αυτά που προϋπήρχαν στην βάση δεδομένων δεν εμφανιστούν στον έλεγχο, σημαίνει ότι έχουν αποσυνδεθεί, οπότε η

κατάστασή τους τροποποιείται αναλόγως. Πιο συγκεκριμένα τα βήματα που ακολουθούνται είναι τα εξής:

1. Αποθήκευση των πληροφοριών που είναι αποθηκευμένες στον πίνακα Datapath της βάσης δεδομένων στο αρχείο datapath.json, ώστε να διατηρηθούν ανέπαφες οι σχετικές με τα datapath πολιτικές που έχει ήδη ορίσει ο διαχειριστής
2. Εκτέλεση του αρχείου κώδικα Python dpidload.py ώστε να γίνει ανάγνωση των συνδεδεμένων με τον FlowVisor datapaths. Το αρχείο αυτό θα αναλυθεί παρακάτω.
3. Εφόσον η εκτέλεση του dpidload.py έχει ολοκληρωθεί, ενημερώνεται η βάση δεδομένων διαβάζοντας το τροποποιημένο πλέον αρχείο datapath.json.
4. Η βάση δεδομένων έχει πλέον ενημερωθεί, οπότε ο χρήστης μεταφέρεται στην αρχική σελίδα, λαμβάνοντας ένα μήνυμα για την ολοκλήρωση της διαδικασίας.

Μέθοδος *ping*

Μέσω της μεθόδου ping δίνεται η δυνατότητα σε οποιονδήποτε χρήστη να ελέγξει αν ο FlowVisor βρίσκεται σε λειτουργία ή όχι, και ταυτόχρονα να μπορεί να δει ποιά έκδοση του FlowVisor χρησιμοποιείται. Μετά την εκτέλεση αυτής της μεθόδου, ο χρήστης μεταφέρεται στην αρχική σελίδα, λαμβάνοντας ένα μήνυμα που περιέχει τις παραπάνω πληροφορίες που τον ενδιαφέρουν.

Μέθοδος *applist*

Όπως αναφέρθηκε στις τελευταίες τέσσερις μεθόδους, μετά την ολοκλήρωση των ενεργειών τους, μεταφέρουν τον χρήστη στην αρχική σελίδα ώστε να μπορεί να συνεχίσει με οποιαδήποτε τροποποίηση ή ενημέρωση επιθυμεί και παράλληλα του μεταφέρουν κάποια πληροφορία μέσω ενός μηνύματος. Η αρχική σελίδα όμως αυτή, είναι υλοποιημένη με τρόπο αυτόματο και τυποποιημένο μέσω του Django. Έτσι για να υλοποιηθεί η ιστοσελίδα αυτή "χειροκίνητα" μέσω μίας από τις παραπάνω μεθόδους, είναι απαραίτητη η χρήση του template index.html το οποίο έχει επίσης δημιουργηθεί αυτόματα από το Django, και λαμβάνει ως δεδομένο ένα dictionary που περιέχει όλες τις πληροφορίες σχετικά με τα μοντέλα που έχουν δημιουργηθεί και έχουν συσχετιστεί με πίνακες της βάσης δεδομένων. Ο σκοπός λοιπόν της μεθόδου applist είναι η δημιουργία αυτού του dictionary, όπως φαίνεται και στον Κώδικα 4.8.

```
def applist(request):
        app_dict = {}
        user = request.user
        for model, model_admin in site._registry.items():
                app_label = model._meta.app_label
                has_module_perms = user.has_module_perms(app_label)

                if has_module_perms:
                        perms = model_admin.get_model_perms(request)

                        if True in perms.values():
                                model_dict = {
                                        'name': capfirst(model._meta.verbose_name_plural),
                                        'admin_url':    mark_safe('/admin/%s/%s/'    %
(app_label, model.__name__.lower())),
                                        'perms': perms,
                                }
                                if app_label in app_dict:
                                        app_dict[app_label]['models'].append(model_dict)
                                else:
                                        app_dict[app_label] = {
                                                'name': app_label.title(),
                                                'app_url': app_label + '/',
                                                'has_module_perms': has_module_perms,
                                                'models': [model_dict],
                                        }

        app_list = app_dict.values()
        app_list.sort(lambda x, y: cmp(x['name'], y['name']))
        return (app_list)
```

Κώδικας 4.8 : Μέθοδος applist δημιουργεί την δομή δεδομένων app_list, την οποία δέχεται ως είσοδο το template index.html για την υλοποίηση της αρχικής σελίδας

4.2.3 Ανάλυση του αρχείου urls.py

Μέσω του αρχείου urls.py γίνεται, όπως έχει ήδη αναφερθεί, η αντιστοίχιση των url που ζητάει κάποιος χρήστης μέσω του browser του, με συγκεκριμένες μεθόδους του αρχείου views.py, με κλάσεις του αρχείου admin.py. Ακόμη, ορίζεται η διεύθυνση του φακέλου ο οποίος θα περιέχει όλα τα στατικά αρχεία εικόνας. Τέλος ενεργοποιείται η λειτουργία ελέγχου και πιστοποίησης των χρηστών (" .../accounts/login/ "), η οποία έχει δημιουργηθεί αυτόματα από το Django. Στον Κώδικα 4.9 φαίνεται η υλοποίηση αυτών των αντιστοιχίσεων.

```
urlpatterns = patterns('',
    (r'^slices/filldb/$', 'slices.views.filldb'),
    (r'^slices/view', 'slices.views.mainslices'),
    (r'^slices/cslice/$', 'slices.views.cslice'),
    (r'^slices/cflowspace/$', 'slices.views.cflowspace'),
    (r'^slices/flowspace/fschange/$', 'slices.views.changefs'),
    (r'^slices/flowspace/change/$', 'slices.views.changefsfinal'),
    (r'^admin/checkup/$', 'slices.views.ping'),
    (r'^admin/power/$', 'slices.views.power'),
    (r'^admin/power/do/$', 'slices.views.power_do'),
    (r'^admin/loaddpid/$', 'slices.views.loaddpid'),
    (r'^admin/slices/sliceinfo/add/$', 'slices.views.createsl'),
    (r'^admin/slices/flowspace/add/$', 'slices.views.createfs'),
    (r'^media/(?P<path>.*)$', 'django.views.static.serve',
 {'document_root': settings.MEDIA_ROOT}),
     (r'^accounts/login/$', login),
     (r'^admin/', include(admin.site.urls)),
)
```

Κώδικας 4.9 : Τμήμα του κώδικα του αρχείου urls.py, όπου υλοποιείται η αντιστοίχιση των urls που μπορεί να ζητήσει ο χρήστης, με μεθόδους άλλων αρχείων όπως το views.py

4.2.4 Ανάλυση του αρχείου admin.py

Μέσω των κλάσεων και των μεθόδων του αρχείου admin.py καθορίζεται ο τρόπος παρουσίασης των μοντέλων που αντιπροσωπεύουν τους πίνακες της βάσης δεδομένων. Έτσι, δημιουργείται μία κλάση για κάθε μοντέλο η οποία καθορίζει τις λεπτομέρειες για την παρουσίαση του κάθε μοντέλου και όλων των πεδίων που αυτό περιλαμβάνει, και ταυτόχρονα καθορίζονται μέθοδοι οι οποίες εκτελούν κάποια συγκεκριμένη λειτουργία για κάθε δεδομένο μοντέλο. Αν και ο καθορισμός της εμφάνισης των μοντέλων είναι ελαχίστης σημασίας και συνεπώς δεν χρήζει περαιτέρω ανάλυσης, είναι πολύ σημαντική μία μέθοδος που χρησιμοποιείται σε κάθε μοντέλο, η μέθοδος *queryset*, η οποία φαίνεται και στον Κώδικα 4.10 μαζί με την κλάση που έχει δημιουργηθεί για το μοντέλο SliceInfo. Μέσω της μεθόδου αυτής, γίνεται η επιλογή των εγγραφών ενός πίνακα της βάσης δεδομένων, με κριτήριο την πολιτική που έχει καθοριστεί για τον κάθε χρήστη ο οποίος ζητά να δει και ίσως να χρησιμοποιήσει τις εγγραφές αυτές. Έτσι, μέσω της μεθόδου αυτής στον Κώδικα 4.10 καθορίζεται ότι αν ο χρήστης είναι ο διαχειριστής του δικτύου, ή τουλάχιστον έχει δικαιώματα διαχειριστή, μπορεί να δει όλες τις εγγραφές που αντιστοιχούν στο μοντέλο το οποίο έχει ζητήσει. Σε αντίθετη περίπτωση, φιλτράρονται όλες οι εγγραφές και επιλέγονται εκείνες μόνο που επιτρέπεται να δει, όπως αυτό καθορίζεται από το πεδίο "managed_by" του μοντέλου SliceInfo. Με τον ίδιο ακριβώς τρόπο, αλλά με τη χρήση διαφορετικού πεδίου αναλόγως με το μοντέλο, υλοποιείται η λειτουργία αυτή και για τα υπόλοιπα μοντέλα έτσι ώστε τελικά κάθε χρήστης να βλέπει και να χρησιμοποιεί μόνο τις εγγραφές εκείνες που του επιτρέπει ο διαχειριστής.

```
class SliceInfoAdmin(admin.ModelAdmin):
    fieldsets = [
        ('Slice Name and Slice user\'s mail', {'fields': ['sliceName','adminMail']}),
        ('TCP connection INFO: ', {'fields': ['ctrlIp','ctrlPort'], 'classes':
['collapse']}), ]
    inlines = [FlowSpaceInline]
    list_display = ('sliceName','adminMail','managed_by')
    def queryset(self, request):
        qs = super(SliceInfoAdmin, self).queryset(request)
        # If super-user, show all comments
        if request.user.is_superuser:
            return qs
        return qs.filter(managed_by=request.user)
```

Κώδικας 4.10 : Κλάση του αρχείου admin.py για τον καθορισμό του τρόπου εμφανίσεις των πληροφοριών του πίνακα SliceInfo της βάσης δεδομένων

4.2.5 Ανάλυση του αρχείου utilities.py

Το αρχείο utilities.py περιλαμβάνει τέσσερις μεθόδους, μέσω τον οποίων μπορεί να γίνει ανάγνωση πληροφοριών του FlowVisor σχετικών με τα slices, καθώς και έλεγχος των κανόνων flowspace που μπορεί να δημιουργήσει ή να τροποποιήσει κάποιος χρήστης.

Μέθοδος *listSlices*

Με την μέθοδο αυτή γίνεται η ανάγνωση των ονομάτων των slices από τον FlowVisor, χρησιμοποιώντας την εντολή "listslices" της εφαρμογής fvctl. Στη συνέχεια δημιουργείται ένα dictionary που περιέχει τα ονόματα αυτά, καθώς και μία μεταβλητή η οποία κρατάει αποθηκευμένο τον συνολικό αριθμό των slices.

Μέθοδος *SliceInfo*

Η μέθοδος αυτή δέχεται ως είσοδο το dictionary και την μεταβλητή, τα οποία δημιουργήθηκαν μέσω της μεθόδου listslices, και για κάθε ένα slice αντλεί από τον FlowVisor τις σχετικές με αυτό πληροφορίες μέσω της εντολής getSliceInfo της εφαρμογής fvctl. Τελικά δημιουργεί και επιστρέφει τρία dictionaries, ένα για τα διαχειριστικά e-mails των slices, ένα για τις IP των Controllers και τέλος ένα για τα Ports στα οποία αναμένουν οι Controller αυτοί να συνδεθεί κάποιο OpenFlow switch.

Μέθοδος *Check_num*

Η μέθοδος αυτή δέχεται ως δεδομένα τον περιορισμό που έχει θέσει ο διαχειριστής για ένα συγκεκριμένο πεδίο ενός κανόνα περιορισμού κάποιου χρήστη, την μέγιστη τιμή που μπορεί να λάβει το πεδίο αυτό, την ελάχιστη, και τέλος την τιμή που επιθυμεί να δώσει ο χρήστης σε αυτό το πεδίο προκειμένου να δημιουργήσει έναν καινούριο κανόνα flowspace. Σημαντικό είναι πως η τιμή περιορισμού που μπορεί να δώσει ο διαχειριστής μπορεί να μην είναι μία συγκεκριμένη τιμή, αλλά μπορεί να είναι ένα σύνολο το οποίο περιλαμβάνει μεμονωμένες τιμές και εύρη τιμών, τα οποία διαχωρίζονται μεταξύ τους με ένα ελληνικό ερωτηματικό. Έτσι λοιπόν η μέθοδος αυτή εκτελεί τα παρακάτω βήματα:

1. Διαχωρισμός του συνόλου των τιμών που έχει ορίσει ο διαχειριστής ως περιορισμούς για το συγκεκριμένο πεδίο.
2. Έλεγχος αν έχει δώσει ο διαχειριστής κάποια εύρη τιμών, και αν ναι τότε έλεγχος αν η τιμή που θέλει να χρησιμοποιήσει ο χρήστης είναι δεκτή. Αν είναι δεκτή, τότε επιστρέφεται θετικό αποτέλεσμα.

3. Έλεγχος αν οποιαδήποτε από τις μεμονωμένες τιμές που μπορεί να έχει δώσει ο διαχειριστής ταυτίζεται με την τιμή που θέλει να χρησιμοποιήσει ο χρήστης και σε περίπτωση ταύτισης επιστρέφεται θετικό αποτέλεσμα.
4. Έλεγχος αν η τιμή που θέλει να χρησιμοποιήσει ο χρήστης στέκει και δεν έχει δώσει για παράδειγμα στο πεδίο slice_action την τιμή "8" ενώ οι αποδεκτές τιμές είναι "1","2","4" ή κάποιο άθροισμά τους.
5. Αν ο διαχειριστής έχει δώσει στον περιορισμό την τιμή "*" που σημαίνει πως ο χρήστης μπορεί να χρησιμοποιήσει οποιαδήποτε τιμή για το συγκεκριμένο πεδίο, τότε παραλείπονται τα βήματα 2 και 3, και επιστρέφει θετικό αποτέλεσμα.
6. Αν οποιοδήποτε εκ των βημάτων 2, 3, 5 επιστρέψει θετικό αποτέλεσμα και το ίδιο συμβεί και στο βήμα 4, τότε η μέθοδος επιστρέφει την τιμή "accept", ενώ αλλιώς την τιμή "decline".

Η τιμή που θα επιστραφεί από την συγκεκριμένη μέθοδο χρησιμοποιείται από την μέθοδο cflowspace του αρχείου views.py για να κρίνει αν πρέπει να δεχτεί την τιμή που θέλει να δώσει ο χρήστης σε ένα συγκεκριμένο πεδίο ή όχι. Αξίζει δε να σημειωθεί ότι η μέθοδος ελέγχου αυτή, χρησιμοποιείται μόνο για τα πεδία priority, action, in_port, source_MAC, destination_MAC, source_port και destination_port μιας εγγραφής flowspace, ενώ για τα πεδία source_IP και destination_IP χρησιμοποιείται η μέθοδος Check_ip, η οποία είναι και η τελευταία μέθοδος του αρχείου utilities.py και λειτουργεί με εντελώς ανάλογο τρόπο προκειμένου να ελέγξει την εγκυρότητα των διευθύνσεων IP πηγής και προορισμού που επιθυμεί να δώσει ο χρήστης σε έναν κανόνα flowspace.

4.2.6 Ανάλυση του αρχείου filler.py

Το αρχείο κώδικα Python filler.py είναι υπεύθυνο να επανεκκινήσει την βάση δεδομένων, συμπληρώνοντας τους πίνακες που έχουν δημιουργηθεί εκεί, με τα πιο πρόσφατα στοιχεία που μπορεί να συλλέξει από τον FlowVisor όσων αφορά τα slices και τα flowspaces. Το αρχείο αυτό δεν περιλαμβάνει καμία ξεχωριστή μέθοδο ή κλάση, και η διαδικασία που ακολουθείται, αφού δεχτεί σαν είσοδο τα δεδομένα που επιστρέφουν οι μέθοδοι listSlices και SliceInfo του αρχείου utilies.py, είναι η εξής:

1. Ανάγνωση του χρήστη που είχε ορίσει ο διαχειριστής για τον έλεγχο κάθε slice πριν την επανεκκίνηση της βάσης δεδομένων και του αναγνωριστικού κλειδιού που χαρακτήριζε κάθε slice, από τα αρχεία record και record2.
2. Δημιουργία μιας δομής JSON για τα slices, όπως αυτή φαίνεται στον Κώδικα 4.11.
3. Αποθήκευση της δομής αυτής στο αρχείο slices_init_db.json και από εκεί, μεταφορά των πληροφοριών στη βάση δεδομένων.
4. Ανάγνωση όλων των πεδίων, όλων των κανόνων flowspace που υπάρχουν ήδη στον FlowVisor, μέσω της εντολής *dump* της εφαρμογής fvconfig.
5. Δημιουργία δομής δεδομένων JSON για τους κανόνες flowspace, όπως αυτή φαίνεται στον Κώδικα 4.12.
6. Αποθήκευση της δομής αυτής στον αρχείο flowspace_init_db.json και από εκεί, μεταφορά των πληροφοριών αυτών στη βάση δεδομένων.

Αφού ολοκληρωθεί η εκτέλεση του κώδικα του αρχείου filler.py το οποίο είχε κληθεί από την μέθοδο filldb του αρχείου views.py, η βάση δεδομένων έχει ενημερωθεί και συμβαδίζει πλήρως με τις ρυθμίσεις που έχουν εφαρμοστεί στον FlowVisor.

```
my_structure=[ {
 "model": "slices.SliceInfo",
 "pk": j+1,
 "fields": {
  "sliceName": dictls[j],
  "adminMail": d1[j],
  "ctrlIp": d2[j],
  "ctrlPort": d3[j],
  "managed_by": user_id[j]
 } }, ]
```

Κώδικας 4.11 : Δομή δεδομένων JSON για μία γραμμή του πίνακα SliceInfo της βάσης δεδομένων

```
flow_structure=[ {
          "model": "slices.FlowSpace",
          "pk": g,
          "fields": {
            "slice_Info" : pk_num,
            "flow_space_id" : sl_id,
            "datapath" : dpid,
            "in_port" : inport,
            "vlan_tag" : vlan,
            "source_MAC" : s_mac,
            "destination_MAC" : d_mac,
            "ethertype" : ethtype,
            "source_IP" : src_ip,
            "destination_IP" : dest_ip,
            "protocol" : net_proto,
            "tos_bits" : tos,
            "source_port" : src_prt,
            "destination_port" : dst_prt,
            "slice_action" : action,
            "slice_priority" : prio,
          } }, ]
```

Κώδικας 4.12 : Δομή δεδομένων JSON για μία γραμμή του πίνακα FlowSpace της βάσης δεδομένων

4.2.7 Ανάλυση του αρχείου dpidload.py

Σκοπός του αρχείου αυτού είναι η ανανέωση της βάσης δεδομένων, όσον αφορά τα datapaths που είναι συνδεδεμένα μία δεδομένη χρονική στιγμή με τον FlowVisor, και παράλληλα η διατήρηση των ρυθμίσεων που έχει κάνει ο διαχειριστής του δικτύου για τα ήδη υπάρχοντα datapaths. Έτσι, η διαδικασία που ακολουθείται είναι η εξής:

1. Ανάγνωση από τον FlowVisor, όλων των datapaths τα οποία είναι συνδεδεμένα με αυτόν την δεδομένη χρονική στιγμή, μέσω της εντολής listDevices της εφαρμογής fvctl.
2. Ανάγνωση των σχετικών με τα datapaths πληροφοριών που έχουν αποθηκευτεί ήδη στο αρχείο datapath.json μέσω της μεθόδου loaddpid του αρχείου views.py, από όπου και κλήθηκε ο κώδικας του αρχείου dpidload.py.
3. Σύγκριση των πληροφοριών που συλλέχθηκαν στα βήματα 1 και 2. Η τιμή του πεδίου is_active για τα datapaths που βρέθηκαν συνδεδεμένα στον FlowVisor παίρνει την τιμή "up", ενώ οποιοδήποτε datapath βρέθηκε μέσω του αρχείου datapath.json και δεν βρέθηκε συνδεδεμένο στον FlowVisor παίρνει την τιμή down, αλλά παραμένει αποθηκευμένη η τιμή του πεδίου users_with_permission, όπως την έχει ορίσει ο διαχειριστής.
4. Δημιουργία μίας δομής JSON για τα datapaths, όπως αυτή φαίνεται στον Κώδικα 4.13.
5. Διαγραφή όλων των πληροφοριών που είναι ήδη αποθηκευμένες στο αρχείο datapath.json και αποθήκευση εκεί της δομής που μόλις δημιουργήθηκε.
6. Μεταφορά των πληροφοριών του αρχείου datapath.json στην βάση δεδομένων για την ολοκλήρωση της διαδικασίας.

```
my_structure=[ {
 "model": "slices.datapath",
 "pk": j+1,
"fields": {
  "users_with_permission": usr[dpid[j]],
 "is_active": act[dpid[j]],
  "dpid": dpid[j],
 } }, ]
```

Κώδικας 4.13 : Δομή δεδομένων JSON για μία γραμμή του πίνακα datapath της βάσης δεδομένων

4.3 Βάση Δεδομένων

Στην ενότητα 3.3 του προηγούμενου κεφαλαίου παρουσιάστηκε το Μοντέλο Οντοτήτων-Συσχετίσεων της βάσης δεδομένων που δημιουργήθηκε και χρησιμοποιείται για την αποθήκευση πληροφοριών απαραίτητων για την λειτουργία του Web-UI. Στην ενότητα αυτή λοιπόν θα αναλυθεί αυτή η βάση δεδομένων, καθώς και η μεταφορά των οντοτήτων και των στοιχείων τους, από το διάγραμμα ERD σε πίνακες της βάσης δεδομένων.

Σκοπός της βάσης δεδομένων είναι η αποθήκευση πληροφοριών που αφορούν τα slices, τα datapaths, τους χρήστες, καθώς και τους περιορισμούς των χρηστών αυτών. Έτσι ήταν αναγκαία η δημιουργία των παρακάτω Οντοτήτων, όπως φαίνεται και στο Σχήμα 3.7 :

- User
- SliceInfo
- Flowspace
- FlowspaceRestrictions
- Datapath

Μέσω των πινάκων που δημιουργήθηκαν στη βάση δεδομένων για κάθε μία από τις παραπάνω Οντότητες γίνεται δυνατή η αποθήκευση, χρήση και ταξινόμηση όλων των απαραίτητων πληροφοριών. Όπως είναι πλέον φανερό κάθε Οντότητα ταυτίζεται με έναν πίνακα και ταυτόχρονα κάθε πίνακας αντιπροσωπεύεται από μία κλάση Python για το Django, άρα θα μπορούσαμε να πούμε ότι οι κλάσεις αυτές ουσιαστικά μοντελοποιούν σε γλώσσα Python τις Οντότητες, όπως αυτές φαίνονται και στο Σχήμα 3.7.

Πίνακας για την Οντότητα *User*

Ο πίνακας αυτός περιέχει όλες τις απαραίτητες πληροφορίες για τους λογαριασμούς χρηστών και τον λογαριασμό του διαχειριστή του Web-UI. Κάθε γραμμή του πίνακα αντιστοιχεί σε ένα και μόνο συγκεκριμένο χρήστη, ενώ τα στοιχεία που αποθηκεύονται στον πίνακα αυτόν, ένα σε κάθε στήλη, καθώς και το είδος πληροφορίας που κρατούν, είναι τα εξής:

- **id :** Εσωτερικό κλειδί, μοναδικό για κάθε γραμμή, το οποίο χρησιμοποιείται για την αναγνώριση κάθε εγγραφής.
- **username :** Όνομα-Ψευδώνυμο του χρήστη
- **first_name:** Όνομα του χρήστη
- **last_name:** Επίθετο χρήστη
- **email :** E-mail επικοινωνίας με τον χρήστη
- **password :** Κρυπτογραφημένος κωδικός πρόσβασης του χρήστη

- **is_staff :** Κατηγορία η οποία χαρακτηρίζει και την πολιτική που διέπει κάθε χρήστη. Στην συγκεκριμένη κατηγορία ανήκουν οι απλοί χρήστες, καθώς και οι ιδιοκτήτες συγκεκριμένων slices
- **is_active :** Το στοιχείο αυτό καθορίζει αν ο χρήστης θεωρείται ενεργός ή όχι
- **is_superuser :** Κατηγορία χρηστών, όπως η κατηγορία is_staff. Εδώ ανήκει ο διαχειριστής του δικτύου/FlowVisor/Web-UI, καθώς και οποιοσδήποτε άλλος χρήστης πρέπει να έχει αντίστοιχα δικαιώματα
- **last_login :** Πιο πρόσφατη ημερομηνία εισόδου του χρήστη στο Web-UI
- **date_joined :** Ημερομηνία εγγραφής του χρήστη στο Web-UI

Πίνακας για την Οντότητα *Datapath*

Στον πίνακα αυτόν αποθηκεύονται πληροφορίες σχετικές με τα Datapaths που υπάρχουν συνδεδεμένα, ή έχουν υπάρξει συνδεδεμένα, με τον FlowVisor. Κάθε γραμμή του πίνακα αυτού αντιπροσωπεύει ένα και μόνο datapath, ενώ τα στοιχεία, κάθε ένα από τα οποία αντιπροσωπεύει και μια στήλη του πίνακα αυτού, είναι τα εξής:

- **id :** Εσωτερικό κλειδί για την αναγνώριση κάθε γραμμής.
- **dpid :** Αναγνωριστικό για κάθε ένα datapath, έτσι όπως κοινοποιείται στον FlowVisor από το ίδιο το OpenFlow switch.
- **is_active :** Ένα πεδίο λογικής μεταβλητής όπου αποθηκεύεται η πληροφορία για το αν είναι συνδεδεμένο ένα datapath με τον FlowVisor μια συγκεκριμένη χρονική στιγμή, ή όχι.
- **users_with_permission :** Αυτό το πεδίο συσχετίζει πολλαπλές γραμμές του πίνακα Datapath, με πολλαπλές γραμμές του πίνακα User, κρατώντας για κάθε γραμμή του πρώτου πίνακα τα κλειδιά των αντίστοιχων γραμμών του δεύτερου πίνακα.

Πίνακας για την Οντότητα SliceInfo

Στον πίνακα αυτόν αποθηκεύονται πληροφορίες σχετικά με τα slices που έχουν δημιουργηθεί, καθώς και τους Controller που τα ελέγχουν, με κάθε γραμμή του να αντιστοιχεί σε ένα και μόνο slice. Κάθε στήλη του πίνακα αντιπροσωπεύει και ένα από τα παρακάτω στοιχεία:

- **id :** Εσωτερικό κλειδί για την αναγνώριση κάθε γραμμής.
- **sliceName :** Όνομα του κάθε slice.
- **adminMail :** E-mail του χρήστη στον οποίο ανήκει το συγκεκριμένο slice και ο οποίος ελέγχει μέσω ενός Controller τη λογική προώθησης πακέτων για το slice αυτό.

- **ctrlIp :** Διεύθυνση IP στην οποία "ακούει" ο Controller και αναμένει για σύνδεση των OpenFlow switches που θα ελέγχονται από αυτόν
- **ctrlPort :** Port στο οποίο ακούει ο Controller
- **managed_by :** Κλειδί που χρησιμοποιείται για τον συσχετισμό κάθε εγγραφής του πίνακα SliceInfo, με μία εγγραφή του πίνακα User

Πίνακας για την Οντότητα Flowspace

Στον πίνακα αυτό αποθηκεύονται πληροφορίες σχετικά με τους κανόνες flowspace που έχουν οριστεί στον FlowVisor. Κάθε γραμμή αντιστοιχεί σε έναν και μόνο κανόνα, ενώ τα πεδία, κάθε ένα από τα οποία αντιστοιχεί και με μία στήλη του πίνακα, είναι τα εξής:

- **id : Ε**σωτερικό κλειδί για την αναγνώριση κάθε γραμμής
- **flowspace_id :** Εσωτερικό κλειδί του FlowVisor για την αναγνώριση κάθε εγγραφής κανόνα flowspace.
- **in_port :** Αριθμός του φυσικού port του ενός datapath από όπου εισήλθε το πακέτο πριν σταλεί στον Controller
- **vlan_tag :** Αριθμός του VLAN για το οποίο ισχύει ο συγκεκριμένος κανόνας
- **source_mac :** Διεύθυνση MAC πηγής
- **destination_mac :** Διεύθυνση MAC προορισμού
- **ethertype :** Πρωτόκολλα ethernet για τα οποία ισχύει ο κανόνας
- **source_ip :** Διεύθυνση IP πηγής
- **destination_ip :** Διεύθυνση IP προορισμού
- **protocol :** Αριθμός που αντιπροσωπεύει το πρωτόκολλο IP για το οποίο ισχύει ο κανόνας flowspace
- **tos_bits :** ToS (Type Of Service) bits της IPv4 κεφαλίδας
- **source_port :** Port πηγής ενός πακέτου
- **destination_port :** Port προορισμού ενός πακέτου
- **slice_action :** Δικαιώματα σχετικά με εγγραφή ή ανάγνωση πακέτων που έχει το συγκεκριμένο slice το οποίο αφορά ο συγκεκριμένος κανόνας flowspace
- **slice_priority :** Προτεραιότητα που έχει ο κανόνας αυτός σε σχέση με άλλους, για πακέτα που θα αντιστοιχηθούν με αυτούς τους κανόνες
- **slice_Name :** Αυτό το πεδίο συσχετίζει πολλαπλές τιμές του πίνακα Flowspace, με μία τιμή του πίνακα SliceInfo. Έτσι κρατάει για κάθε γραμμή του πρώτου πίνακα, την τιμή του πεδίου id του δεύτερου πίνακα
- **datapath :** Datapath για το οποίο ισχύει ο συγκεκριμένος κανόνας.

Πίνακας για την Οντότητα FlowspaceRestrictions

Ο πίνακας αυτός αποθηκεύει τους περιορισμούς που έχει θέσει ο διαχειριστής για τους χρήστες του Web-UI. Έτσι σε κάθε γραμμή αποθηκεύεται ένας περιορισμός που αφορά πολλαπλούς χρήστες και ένα συγκεκριμένο datapath. Τα στοιχεία που αντιστοιχούν στις στήλες του πίνακα είναι ίδια με αυτά του πίνακα Flowspace, με τρεις διαφορές όμως. Αρχικά δεν περιλαμβάνονται τα στοιχεία slice_Name και datapath του πίνακα Flowspace. Ακόμη, τα στοιχεία που ταυτίζονται με αυτά του πίνακα Flowspace αποθηκεύουν κωδικοποιημένη, όχι μία μόνο τιμή, αλλά ένα σύνολο μεμονωμένων τιμών και ένα σύνολο εύρων τιμών. Τέλος ο πίνακας FlowspaceRestrictions περιέχει δύο ακόμα στοιχεία-στήλες:

- dpid : Συσχετίζει πολλαπλές εγγραφές του πίνακα FlowspaceRestriction με μία τιμή του πίνακα datapath, αποθηκεύοντας την τιμή του στοιχείου id του τελευταίου
- restricted_users : Συσχετίζει πολλαπλές εγγραφές του πίνακα FlowspaceRestriction με πολλαπλές τιμές του πίνακα User.

4.4 Κωδικοποίηση αρχείων

Για την υλοποίηση συγκεκριμένων λειτουργιών όπως έχει αναφερθεί στην ενότητα 4.3 χρησιμοποιούνται δύο αρχεία καταγραφών (log files) που δημιουργούνται μέσω μεθόδων του views.py και οι πληροφορίες που περιέχουν τροποποιούνται από αυτό αλλά και άλλα αρχεία κώδικα Python όπως έχουμε δει.

Αρχείο *Record*

Μία βασική δυνατότητα του Web-UI είναι ο συγχρονισμός της βάσης δεδομένων μας με τον FlowVisor, όπως έχει ήδη αναλυθεί παραπάνω. Λόγω του ότι είναι απαραίτητο, κάθε φορά που θα συγχρονίζεται η βάση δεδομένων μέσω του filler.py, να διατηρούνται οι συσχετίσεις που έχει ορίσει ο διαχειριστής μεταξύ των slices και των χρηστών οι οποίοι μπορούν να τα διαχειρίζονται ή να τα χρησιμοποιούν, δημιουργήθηκε το συγκεκριμένο αρχείο. Για κάθε ένα slice που υπήρχε πριν γίνει ο συγχρονισμός, αντιστοιχίζεται μία γραμμή του αρχείου record, όπου είναι αποθηκευμένο το όνομα του slice, καθώς και ο χρήστης στον οποίο είχε ανατεθεί ο έλεγχος τους slice αυτού, υπό την μορφή:

"<SliceName> = <UserName>"

Κατά την εκτέλεση του κώδικα που περιέχεται στο filler.py, θα χρησιμοποιηθούν οι εγγραφές του αρχείου καταγραφής Record, ώστε να αντιστοιχηθούν σωστά τα slices με τους χρήστες οι οποίοι τα διαχειρίζονται, ενώ αν βρεθεί κάποιο slice το οποίο δεν έχει αντιστοιχιστεί με κάποιο χρήστη, θα ανατεθεί ο έλεγχός του στον διαχειριστή του δικτύου.

Αρχείο *Record2*

Για τον ίδιο λόγο με το αρχείο Record, δημιουργήθηκε και το αρχείο Record2. Στο αρχείο αυτό αποθηκεύονται οι τιμές των πεδίων id του πίνακα User. Έτσι, κάθε γραμμή του αρχείου αυτού περιέχει την τιμή του πεδίου username ενός χρήστη και την τιμή του πεδίου id, υπό την μορφή:

" <UserName> = <User_id> "

Έτσι, κατά τη δημιουργία της Δομής Δεδομένων JSON (η οποία θα αποθηκευτεί στο αρχείο "slices_init_db.json") μέσω του κώδικα του αρχείου filler.py, θα χρησιμοποιηθούν οι εγγραφές των αρχείων καταγραφής Record και Record2, για την εύρεση του κλειδιού που πρέπει να τοποθετηθεί στο πεδίο "managed_by" κάθε εγγραφής του πίνακα SliceInfo.

5

Ανάπτυξη και Έλεγχος του Web-UI

5.1 Χαρακτηριστικά υλοποίησης του Web-UI

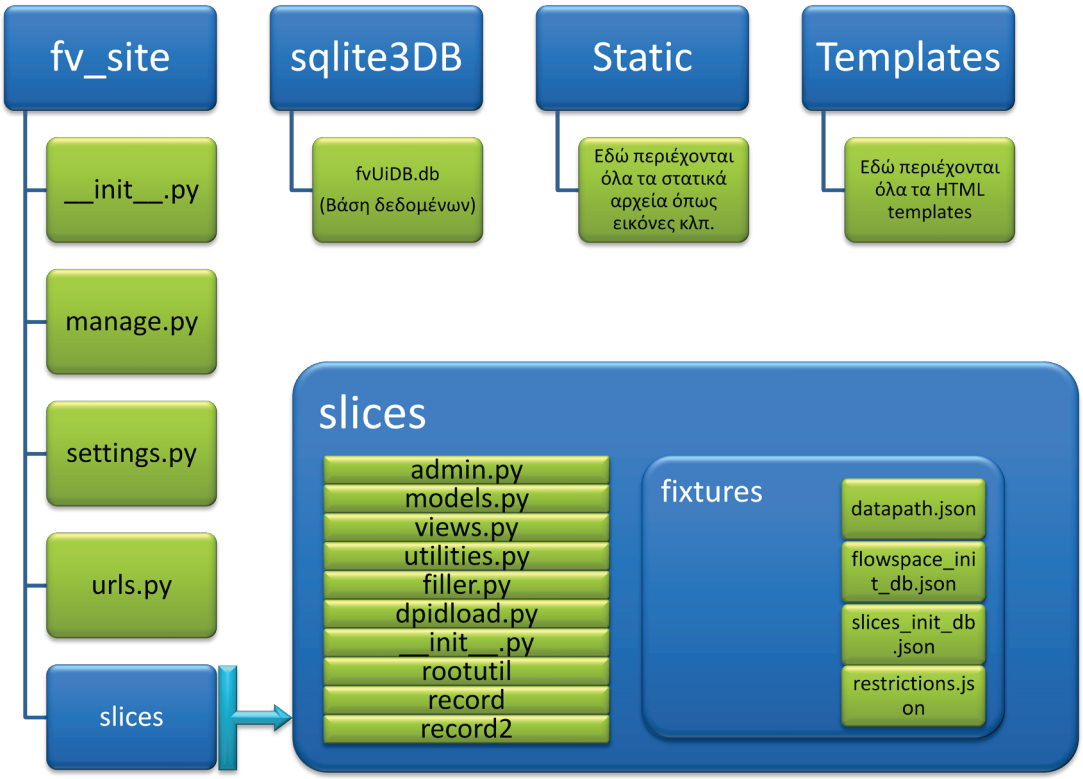
Η ανάπτυξη και λειτουργία του Web-UI για το FlowVisor έγινε σε περιβάλλον Linux Ubuntu 9.10 [20], χρησιμοποιώντας το Web Framework Django όπως έχει αναλυθεί σε προηγούμενο κεφάλαιο. Παρακάτω περιγράφονται οι απαραίτητες εντολές για την εγκατάσταση των απαραίτητων στοιχείων του Web-UI σε ένα τέτοιο σύστημα.

1. Εγκατάσταση του FlowVisor 0.7.1 [21]
   a) ↵ apt-get install ant sun-java6-jdk
   b) ↵ apt-get install git-core
   c) ↵ gedit /etc/apt/sources.list
   d) Στο τέλος του αρχείου "sources.list" προσθέτουμε τη γραμμή "deb http://openflow.org/downloads/GENI/DEB unstable /binary-$(ARCH)"
   e) ↵ apt-get update
   f) ↵ apt-get install flowvisor
2. Εγκατάσταση του Django v1.2.5
   a) Κατεβάζουμε το πακέτο με τα binaries του Django από το: *http://www.djangoproject.com/download/1.2.5/tarball/*
   b) Για γίνει η εγκατάσταση πρέπει να επαληθεύσουμε ότι διαθέτουμε εγκατεστημένο το πακέτο Python v2.4 (και πάνω)
   c) Στη συνέχεια τρέχουμε το αρχείο setup.py που υπάρχει στον φάκελο της διανομής του Django, με την εντολή:

      ↵ pyhton setup.py install

3. Εκτέλεση και αρχικές ρυθμίσεις του FlowVisor
    d) ↵ fvconfig generate /usr/etc/flowvisor/flowvisor-config.xml
    e) ↵ flowvisor /usr/etc/flowvisor/flowvisor-config.xml

Πλέον στο σύστημά μας έχει εγκατασταθεί το Django, καθώς και ο FlowVisor, ο οποίος έχει ήδη τεθεί σε λειτουργία. Το μόνο που απομένει πλέον είναι η τοποθέτηση των απαραίτητων αρχείων στο σωστό μονομάτι (Path), καθώς και η εκκίνηση του Web Server. Τα αρχεία τοποθετούνται στον φάκελο "/home/<username>/fv-ui" όπου στην συγκεκριμένη υλοποίηση το username είναι "flowvisor07". Η οργάνωση των αρχείων στη μνήμη, μέσα στον φάκελο "fv-ui" φαίνεται στο Σχήμα 5.1, όπου με μπλε χρώμα εμφανίζονται οι φάκελοι και υποφάκελοι, ενώ με πράσινο εμφανίζονται τα αρχεία που βρίσκονται μέσα σε κάθε φάκελο.



Σχήμα 5.1 : Οργάνωση των αρχείων στη μνήμη του υπολογιστή, μέσα στον φάκελο "/home/<username>/fv-ui"

Τα αρχεία "__init__.py", "manage.py" και "settings.py" είναι αρχεία που παρέχονται από το Django από τη στιγμή που θα ξεκινήσει η ανάπτυξη μιας διαδικτυακής εφαρμογής. Εφόσον έχουν τοποθετηθεί σωστά όλα τα αρχεία, μπορούμε πλέον να εκκινήσουμε τον Web Server που παρέχει το Django. Αυτό γίνεται χρησιμοποιώντας το αρχείο κώδικα "manage.py" μέσω

της εντολής: ↵ *python manage.py runserver*. Πλέον το Web-UI του FlowVisor βρίσκεται σε λειτουργία και στην επόμενη ενότητα θα ακολουθήσει ο έλεγχος της λειτουργίας του, με τη χρήση ενός σεναρίου λειτουργίας.

5.2 Εγκατάσταση OpenFlow switch και NOX Controller

Για την ανάπτυξη του Web-UI ήταν απαραίτητη η χρήση OpenFlow switches καθώς και NOX Controllers για τον έλεγχο και την αξιολόγηση της λειτουργίας του. Όσον αφορά τα OpenFlow switches χρησιμοποιήθηκε η υλοποίηση software OpenFlow switches που παρέχεται μέσω του πανεπιστημίου Stanford. Για την εγκατάσταση ενός τέτοιου OpenFlow switch σε έναν ηλεκτρονικό υπολογιστή και ενός NOX Controller στον ίδιο ή σε έναν διαφορετικό υπολογιστή, ακολουθούνται τα παρακάτω βήματα:

1. Εγκατάσταση των βιβλιοθηκών απο τις οποίες εξαρτάται ο NOX Controller
   a) ↵ cd /etc/apt/sources.list.d
   b) ↵ wget http://openflowswitch.org/downloads/debian/nox.list
   c) ↵ apt-get update
   d) ↵ apt-get install nox-dependencies git-core

2. Μεταφόρτωση του λογισμικού του NOX Controller (version Zaku) και μεταγλώττιση:
   a) ↵ git clone git://noxrepo.org/nox
   b) ↵ cd nox
   c) ↵ git checkout -b nox-zaku origin/zaku
   d) ↵ ./boot.sh
   e) ↵ mkdir build
   f) ↵ cd build
   g) ↵ ../configure
   h) ↵ make -j 5 (απαιτείται τουλάχιστον 1GB RAM στον υπολογιστή)

3. Εκκίνηση του Controller:
   a) ↵ cd nox/build/src/
   b) ↵ ./nox_core -v -i ptcp:<*Controller's port*>

4. Μεταφόρτωση του λογισμικού του OpenFlow reference switch v1.0.0 και των βιβλιοθηκών από τις οποίες εξαρτάται:
   a) ↲ apt-get install automake pkg-config autoconf libtool
   b) ↲ git clone git://openflowswitch.org/openflow.git
   c) ↲ cd openflow
   d) ↲ git checkout -b openflow.v1.0 origin/release/1.0.0

5. Μεταγλώττιση των αρχείων:
   a) ↲ ./boot.sh
   b) ↲ ./configure
   c) ↲ make
   d) ↲ make install

6. Εκκίνηση λειτουργίας του OpenFlow switch:

   ↲ ./udatapath/ofdatapath punix:/var/run/dp0 -i eth1,eth2,eth3 ...

   Με αυτή την εντολή δημιουργείται το datapath dp0 το οποίο χρησιμοποιεί ως ports του OpenFlow switch τα interfaces eth1, eth2 και eth3.

7. Επικοινωνία του OpenFlow switch me ton Controller

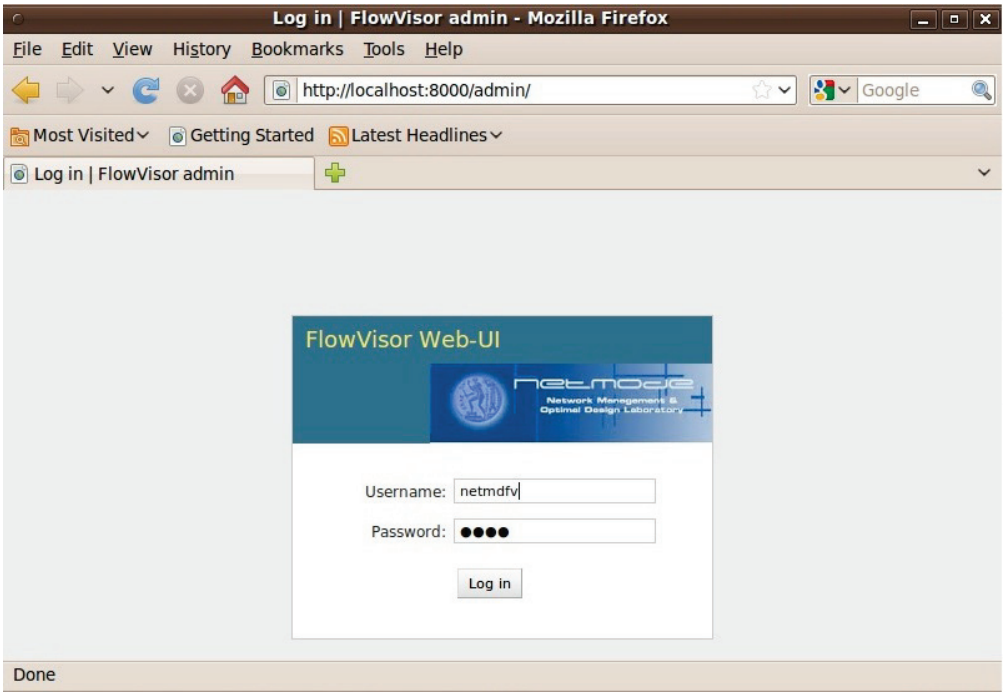
   ↲ ./secchan/ofprotocol unix:/var/run/dp0 tcp:<*Controller's IP*>:<*Controller's Port*>

Στην περίπτωση που κάποιος επιθυμούσε απλά να θέσει ένα OpenFlow switch υπό τον έλεγχο κάποιου Controller αρκεί να ακολουθήσει την παραπάνω διαδικασία. Ο Controller αναμένει μέσω του port που ορίζεται στο βήμα 3.b να επικοινωνήσει με κάποιο OpenFlow switch, ενώ το switch από την άλλη πλευρά αναζητά τον Controller μέσω της διεύθυνσης IP και του port που ορίζονται στο βήμα 7. Πρέπει όμως να τονιστεί, πως στην περίπτωση που επιθυμούμε να παρεμβάλλεται ο FlowVisor μεταξύ πολλαπλών OpenFlow switches και Controllers, τότε στο βήμα 7 θα οριστούν σαν IP και port του Controller, εκείνα τα FlowVisor.

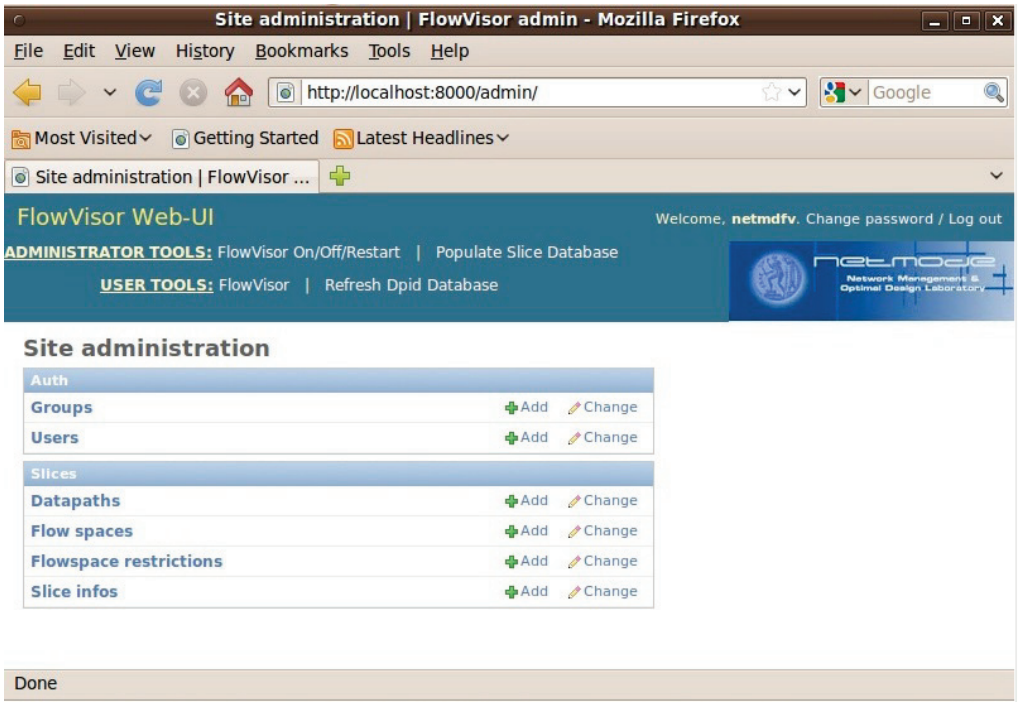
5.3 Έλεγχος λειτουργίας του FlowVisor Web-UI

Για τον έλεγχο της διαδικτυακής εφαρμογής που δημιουργήθηκε, θα χρησιμοποιήσουμε ένα απλό παράδειγμα. Υποθέτουμε λοιπόν πως σε ένα δίκτυο, ο διαχειριστής του επιθυμεί να ορίσει σαν προεπιλογή ότι όλη η δικτυακή κίνηση θα ελέγχεται από έναν Controller που θα ονομάσουμε Controller1. Ακόμη υποθέτουμε πως υπάρχει ένας ερευνητής που χρησιμοποιεί το συγκεκριμένο δίκτυο και επιθυμεί όλη η HTTP κίνησή του να ελέγχεται από τον Controller2, μέσω του οποίου θα μπορέσει να δοκιμάσει έναν αλγόριθμο εξισορρόπησης του φορτίου (Load Balancing Algorithm). Τέλος όμως είναι πολύ πιθανό, σε περίπτωση που ο αλγόριθμος αυτός αποδειχθεί αποδοτικός, να επιθυμούν περισσότεροι χρήστες να αναθέσουν τον έλεγχο της HTTP κίνησής τους στον Controller2.

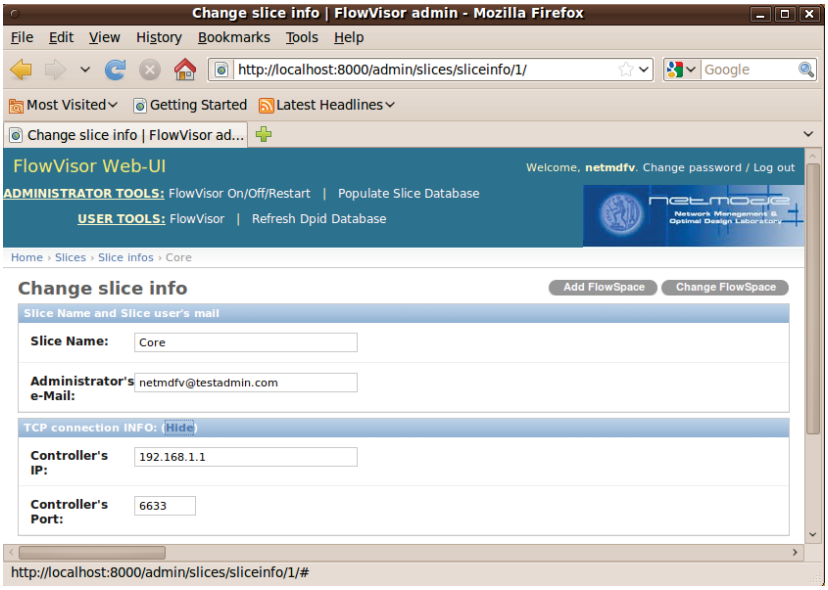
Ο διαχειριστής εμφανίζεται στο Web-UI ως ο χρήστης "netmdf", και έχει τον πλήρη έλεγχό του. Αρχικά, πρέπει να πιστοποιηθεί η ταυτότητα του διαχειριστή όπως φαίνεται και στο Σχήμα 5.2. Στη συνέχεια εισέρχεται στην αρχική επιφάνεια του Web-UI (Σχήμα 5.3) από όπου μπορεί πλέον να παραμετροποιήσει τον FlowVisor σύμφωνα με τις ανάγκες του. Τώρα πρέπει να δημιουργήσει ένα slice το οποίο θα διαχειρίζεται όλη τη δικτυακή κίνηση, μέσω του Controller1. Αυτό γίνεται επιλέγοντας το πλήκτρο "Add" δίπλα από τον σύνδεσμο "Slice infos", ενώ στη συνέχεια πρέπει να δώσει τα στοιχεία που θα χαρακτηρίζουν το slice αυτό, δηλαδή όνομα, διεύθυνση IP του Controller1, port του Controller1 και διαχειριστικό e-mail το οποίο προφανώς θα είναι το e-mail του ίδιου του διαχειριστή. Το slice που έχει πλέον δημιουργηθεί ονομάζεται "Core" και φαίνεται στο Σχήμα 5.4. Τέλος πρέπει να ορίσει το FlowSpace του συγκεκριμένου slice προσθέτοντας τους αντίστοιχους κανόνες. Στη συγκεκριμένη περίπτωση που θέλουμε να θέσουμε ολόκληρο το δίκτυο υπό τον έλεγχο του συγκεκριμένου slice, είναι αρκετός ένας και μόνο κανόνας ο οποίος θα έχει όλα του τα πεδία σαν wildcard, με δικαίωμα ανάγνωσης και εγγραφής όλων των OpenFlow πακέτων ενώ σαν datapath ορίζουμε την τιμή "any". Τέλος στον κανόνα θα ορίσουμε ως προτεραιότητα την τιμή 100 (τυχαία τιμή), και ο κανόνας αυτός φαίνεται στο Σχήμα 5.5.



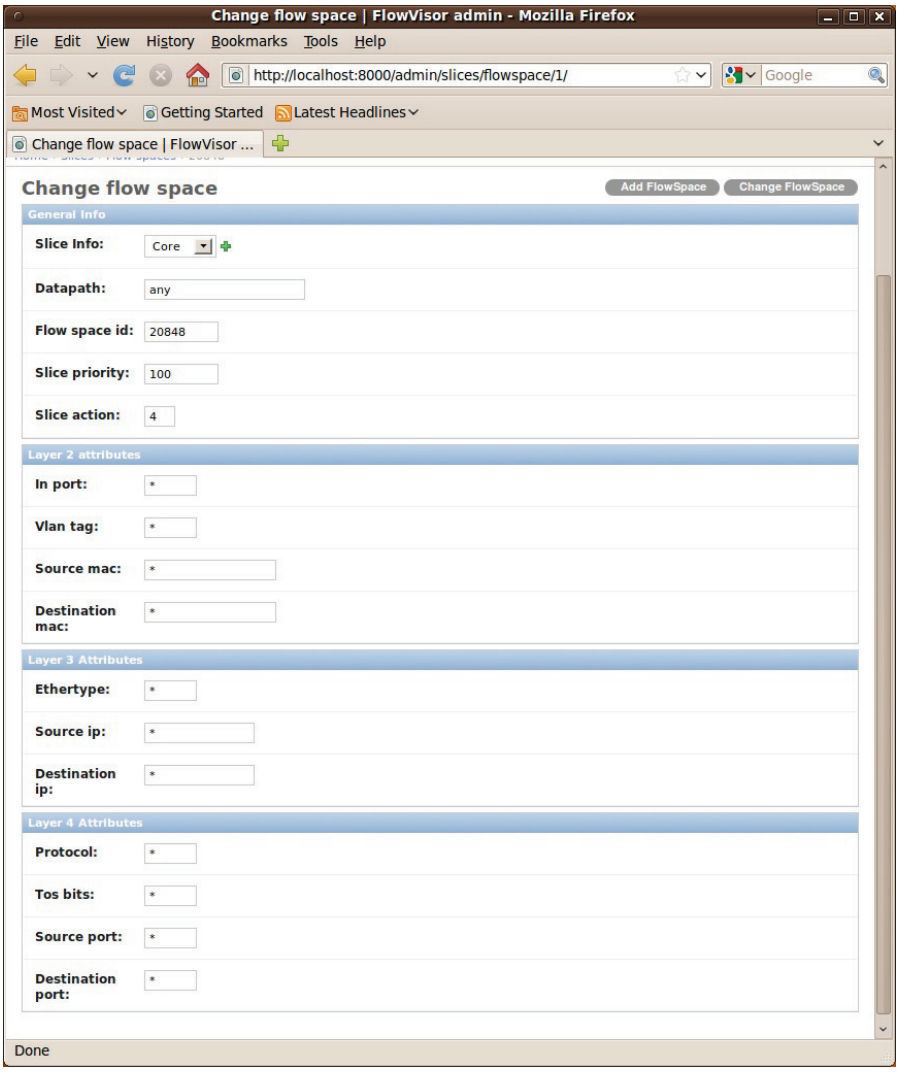
Σχήμα 5.2 : Επιφάνεια επαλήθευσης ταυτότητας χρηστών



Σχήμα 5.3 : Αρχική επιφάνεια για τον διαχειριστή του FlowVisor



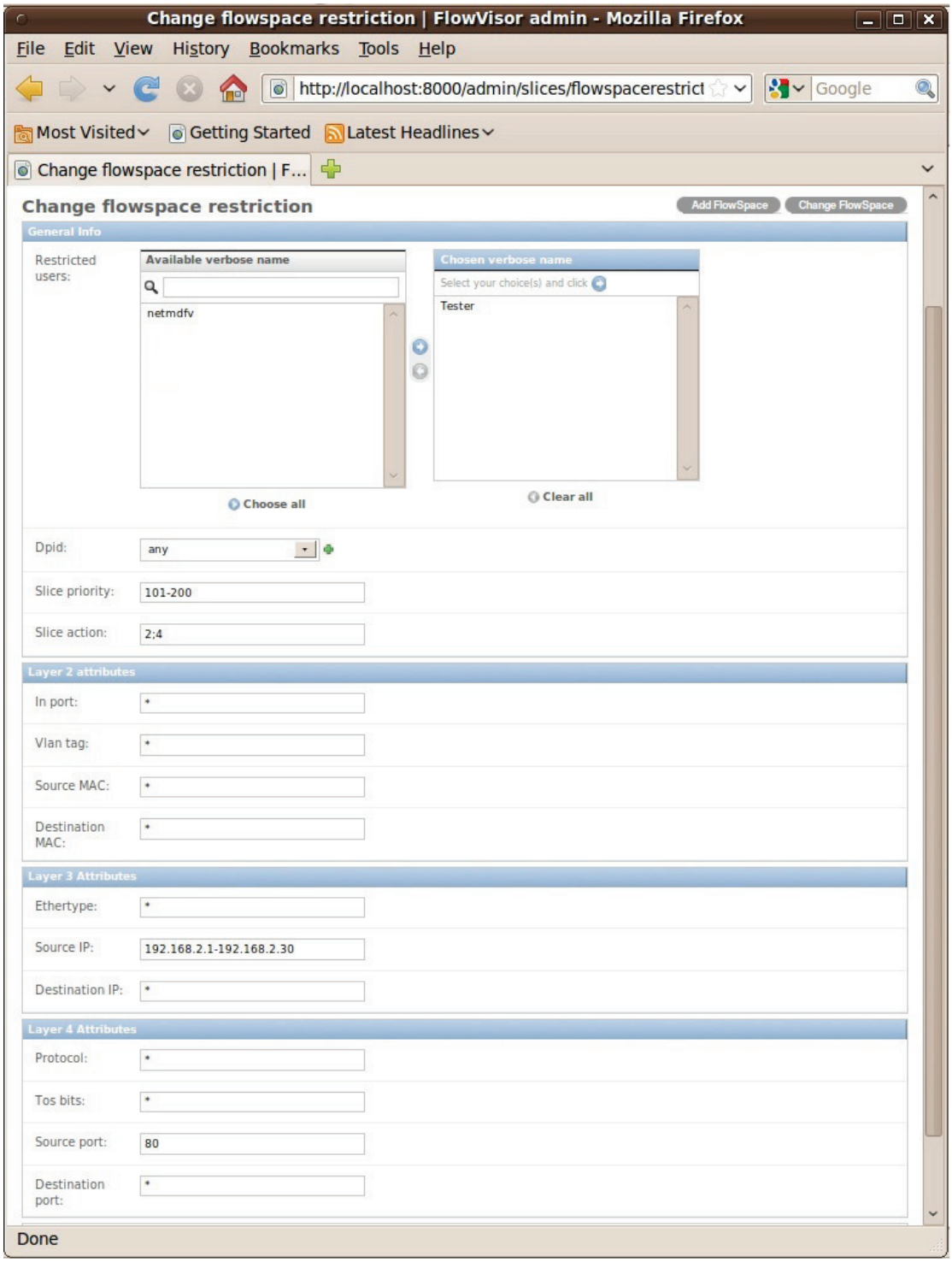
Σχήμα 5.4 : Επιφάνεια όπου εμφανίζονται πληροφορίες σχετικές με το slice και τον Controller, μέσω του οποίου ελέγχεται η δικτυακή κίνηση



Σχήμα 5.5 : Επιφάνεια όπου εμφανίζεται ο κανόνας που ορίζει το flowspace το οποίο ανήκει στο slice Core

Πλέον ο FlowVisor έχει παραμετροποιηθεί έτσι, ώστε ο Controller1 να είναι υπεύθυνος για ολόκληρη την δικτυακή κίνηση. Εξαιρώντας το γραφικό περιβάλλον που προσφέρει η χρήση του Web-UI, η διαδικασία μέχρι στιγμής θα μπορούσε να γίνει εύκολα χρησιμοποιώντας απλά και μόνο την κονσόλα του FlowVisor. Όμως σειρά πλέον έχουν οι απαραίτητες ρυθμίσεις, ώστε να δοθεί η δυνατότητα στον ερευνητή να πραγματοποιήσει το πείραμά του, και εδώ φαίνεται η πρόσθετη λειτουργικότητα που προσφέρει το Web-UI έναντι της κονσόλας του FlowVisor. Για να γίνει λοιπόν αυτό χρησιμοποιώντας την κονσόλα του FlowVisor, θα έπρεπε αρχικά ο διαχειριστής να δημιουργήσει ένα καινούριο slice που θα ελέγχεται από τον Controller2 ο οποίος ανήκει στον ερευνητή. Στη συνέχεια θα έπρεπε να προσθέσει χειροκίνητα ο διαχειριστής τούς κανόνες flowspace που θα ικανοποιούν τις ανάγκες του ερευνητή τη δεδομένη χρονική στιγμή. Στο εξής όμως, ο ερευνητής θα είχε υπό τον έλεγχό του αποκλειστικά και μόνο τον Controller2, ενώ για οποιαδήποτε περαιτέρω τροποποίηση χρειαζόταν, θα έπρεπε να ζητήσει από τον διαχειριστή να προβεί στις απαραίτητες ενέργειες, και ακριβώς αυτό είναι το σημείο όπου το Web-UI προσφέρει τη λύση.

Χρησιμοποιώντας το Web-UI του FlowVisor, αρχικά ο διαχειριστής πρέπει να δημιουργήσει ένα λογαριασμό για τον ερευνητή, όπου και θα του δώσει δικαιώματα "staff user", δηλαδή ενός χρήστη με προκαθορισμένα περιθώρια ελευθερίας για την παραμετροποίηση του FlowVisor. Στη συνέχεια πρέπει να ορίσει κανόνες περιορισμού για τον χρήστη αυτόν, ορίζοντας κατά συνέπεια και την πολιτική του. Πιο συγκεκριμένα, αφού ο ερευνητής επιθυμεί να ελέγχει μέσω του Controller2 μόνο την HTTP κίνησή του, ο διαχειριστής θα δημιουργήσει δύο κανόνες περιορισμού του flowspace (Flowspace Restrictions), έναν για την περίπτωση που ο ερευνητής χρησιμοποιεί τον υπολογιστή του ως server και έναν για την περίπτωση που ο υπολογιστής του λειτουργεί ως client. Το μόνο που πρέπει να εξασφαλίσει ο διαχειριστής με τους κανόνες αυτούς είναι πως ο Controller2 θα ελέγχει οποιοδήποτε OpenFlow πακέτο έχει σαν πηγή ή σαν προορισμό την διεύθυνση IP του ερευνητή και το port 80. Ακόμη, στην περίπτωση που ο ερευνητής επιθυμεί να ελέγχει την HTTP κίνηση περισσότερων από έναν υπολογιστών (και κατά συνέπεια περισσότερων από μία διευθύνσεων IP) μπορεί ο διαχειριστής να ορίσει στους δύο κανόνες που αναφέρθηκαν, όλες τις απαιτούμενες διευθύνσεις IP των υπολογιστών. Αν υποθέσουμε λοιπόν πως ο ερευνητής επιθυμεί ο Controller2 να ελέγχει την HTTP κίνηση όλων των υπολογιστών που ανήκουν στο υποδίκτυο 192.168.2.0/27, τότε ο κανόνας περιορισμού που θα ορίσει ο διαχειριστής, για την περίπτωση που οι υπολογιστές που ανήκουν στο υποδίκτυο αυτό λειτουργούν ως clients, φαίνεται στο Σχήμα 5.6. Με εντελώς ανάλογο τρόπο θα δημιουργηθεί και ο κανόνας για την εξερχόμενη HTTP κίνηση.



Σχήμα 5.6 : Κανόνας περιορισμού του flowspace που μπορεί να ελέγξει ο Controller2

Μετά τη δημιουργία των περιορισμών για τον ερευνητή, είναι εκείνος υπεύθυνος για την δημιουργία του flowspace που θα ανήκει στο slice που ελέγχει ο Controller2, ενώ σε περίπτωση που κριθεί αναγκαία κάποια τροποποίηση του flowspace μπορεί ο ίδιος να προβεί στις απαραίτητες ρυθμίσεις, χωρίς την παρέμβαση του διαχειριστή. Στον κανόνα περιορισμού του Σχήματος 5.6 πρέπει να τονιστούν τα ακόλουθα:

- Η τιμή του πεδίου "Slice Priority" που μπορεί να δώσει ο ερευνητής στους κανόνες flowspace που θα δημιουργήσει, πρέπει να είναι μεγαλύτερη από την τιμή 100, ώστε να ξεπερνούν τον κανόνα που έχει ορίσει ο διαχειριστής για το flowspace που ανήκει στον Controller1.
- Στο πεδίο "Slice action" έχουν οριστεί οι τιμές "2" και "4", έτσι ώστε ο ερευνητής να μπορεί να επιλέξει ανάμεσα στα δικαιώματα ανάγνωσης (για την τιμή "2") ή εγγραφής (για την τιμή "3").
- Όλα τα πεδία που έχουν οριστεί ως wildcard (δηλαδή έχουν την τιμή "*"), μπορούν να πάρουν οποιαδήποτε τιμή επιθυμεί ο ερευνητής, αρκεί οι τιμές που θα δώσει στα υπόλοιπα πεδία να είναι μέσα στα επιτρεπτά όρια.

Τέλος, πολύ πιθανό είναι σε ένα δίκτυο όπως αυτό του παραδείγματος, να βρεθούν χρήστες οι οποίοι επιθυμούν να αναθέσουν τη διαχείριση της δικής τους διαδικτυακής κίνησης στον Controller2. Ο διαχειριστής μπορεί με μία διαδικασία εντελώς ανάλογη με την προηγούμενη, να δημιουργήσει λογαριασμό στο Web-UI του FlowVisor για κάθε έναν από αυτούς τους χρήστες, και αφού ορίσει τους ανάλογους περιορισμούς, να τους επιτρέψει να ελέγξουν ποιος Controller θα ελέγχει την διαδικτυακή τους κίνηση, όπως αυτοί θεωρούν σκόπιμο.

6

Επίλογος

6.1 Σύνοψη και συμπεράσματα

Το πρωτόκολλο OpenFlow παρουσιάζεται ως ένας δικτυακός μηχανισμός βασισμένος σε flows, ούτως ώστε οι ερευνητές να μπορούν να εκτελούν δικτυακά πειράματα σε ένα πραγματικό δίκτυο υπολογιστών, από όπου διέρχεται η καθημερινή δικτυακή τους κίνηση [22]. Εκτός από αυτό όμως, τα Καθορισμένα Μέσω Εφαρμογών δίκτυα (Software-Defined Networks - SDN) αναδεικνύονται ως μία νέα αρχιτεκτονική των δικτύων, και το OpenFlow αποτελεί για την αρχιτεκτονική αυτή ένα API ανοιχτό και ανεξάρτητο από κατασκευαστές [23].

Σημαντικό ρόλο στην αρχιτεκτονική SDN μπορεί να έχει ο FlowVisor, ως ένας Controller για OpenFlow switches, που παρέχει τη δυνατότητα τεμαχισμού των δικτυακών πόρων σε slices, απομονωμένα μεταξύ τους. Έτσι, σαν μία προσπάθεια βελτίωσης της λειτουργίας του FlowVisor, καθώς και της ευχρηστίας του, παρουσιάστηκε στη συγκεκριμένη διπλωματική εργασία ένα Web-UI για τον FlowVisor.

Η συνεισφορά του Web-UI στην λειτουργικότητα του FlowVisor που λειτουργεί σε ένα δίκτυο που σχεδιάστηκε με βάση την αρχιτεκτονική SDN συνοψίζεται στα παρακάτω σημεία:

- Μέσω του γραφικού περιβάλλοντος που προσφέρει το Web-UI, γίνεται πιο εύκολη η χρήστη του FlowVisor, αφού οι πληροφορίες που ζητούνται από τον χρήστη του FlowVisor, καθώς και οι πληροφορίες που παρέχονται από αυτόν, εμφανίζονται με τρόπο πιο κατανοητό και πιο οικείο προς τον άνθρωπο.

- Δίνει τη δυνατότητα στον διαχειριστή ενός δικτύου να δημιουργήσει λογαριασμούς χρηστών, ούτως ώστε να μπορούν να παρακολουθούν ή να ελέγχουν συγκεκριμένα slices του δικτύου αυτού.
- Ο διαχειριστής μπορεί, και επιβάλλεται, να ορίσει την πολιτική που θα περιορίζει τον κάθε έναν από τους χρήστες που θα δημιουργήσει, ώστε οποιοσδήποτε χρήστης να μπορεί να παρακολουθεί ή να ελέγχει μόνο τα slices που επιτρέπονται.
- Στη βάση δεδομένων, εκτός των άλλων, διατηρούνται και εγγραφές για datapaths που μπορεί να έχουν αποσυνδεθεί από τον FlowVisor, έτσι ώστε ο διαχειριστής να έχει μια πιο σαφή εικόνα του "ιστορικού" τους.

Όμως η χρήση του Web-UI έρχεται με ένα αντάλλαγμα, αφού πλέον γίνεται πιο περίπλοκος ο ρόλος του διαχειριστή. Αυτό γιατί εκτός από την ευθύνη για τα slices και τους κανόνες flowspace, είναι υπεύθυνος και για τη δημιουργία των πολιτικών των χρηστών, κάτι που σε μεγάλα δίκτυα υπολογιστών, όπως πανεπιστημίων ή μεγάλων επιχειρήσεων, μπορεί να είναι μία δύσκολη διαδικασία, αναλόγως με τις απαιτήσεις των χρηστών.

6.2 Βελτιώσεις και μελλοντικές επεκτάσεις

Μέσω του Web-UI υλοποιούνται οι βασικότερες των λειτουργιών του FlowVisor, ενώ βελτιώνεται η πλευρά της διαχείρισης των slices. Η υλοποίηση που παρουσιάστηκε ήταν μία προσπάθεια στα πλαίσια της διπλωματικής εργασίας και βέβαια υπάρχει περιθώριο περαιτέρω βελτίωσής της.

Ένα σημείο το οποίο χρήζει βελτίωσης, αφορά τις πολιτικές των χρηστών και τον τρόπο ορισμού αυτών. Σημαντική βελτίωση στον τομέα αυτό θα αποτελούσε η τροποποίηση του τρόπου λειτουργίας των κανόνων περιορισμού των χρηστών, ίσως με τη χρήση ψευδοκώδικα αντί για πεδία τιμών, αφού όσο πιο περίπλοκες είναι οι απαιτήσεις των χρηστών, τόσο περισσότεροι κανόνες μπορεί να απαιτούνται από τον διαχειριστή. Ακόμη πιο σωστή και ίσως πιο πρακτική προσέγγιση θα ήταν ο ορισμός μίας γενικής πολιτικής κατά τη δημιουργία του λογαριασμού κάθε χρήστη, πράγμα το οποίο χρήζει περαιτέρω ανάλυσης, αφού οι πολιτικές αυτές πρέπει από τη μία να είναι αρκετά γενικές, αλλά από την άλλη δεν πρέπει να δίνουν υπερβολική ελευθερία στους χρήστες.

Μία σημαντική βελτίωση στη λειτουργία του Web-UI, θα ήταν η δυνατότητα να παρέχει στατιστικές πληροφορίες σχετικά με τα datapaths και τα flows που υπάρχουν στο δίκτυο, και αυτό είναι ένα σημείο όπου υστερεί το Web-UI έναντι της κονσόλας του FlowVisor.

Τέλος μία ακόμη βελτίωση και ίσως η πιο σημαντική θα ήταν να ανεξαρτητοποιηθεί εντελώς το Web-UI από τις εφαρμογές fvctl και fvconfig, και να "συνδεθεί" απευθείας με το API του FlowVisor. Με αυτόν τον τρόπο όχι μόνο θα ήταν εύκολη η υλοποίηση της βελτίωσης που αναφέρθηκε στην προηγούμενη παράγραφο, αλλά ταυτόχρονα θα πετυχαίναμε σημαντική μείωση στο χρόνο αναμονής κατά τη διάρκεια δημιουργίας, διαγραφής ή αλλαγής των slices και των κανόνων flowspace, καθώς και κατά τον συγχρονισμό της βάσης δεδομένων με τον FlowVisor.

Βιβλιογραφία

[1]. "Open Networking Foundation. *http://www.opennetworkingfoundation.org* "

[2]. "OpenFlow. *http://www.openflow.org*"

[3]. McKeown, N., T. Anderson, et al. (2008). "OpenFlow: Enable Innovation in Campus Networks."

[4]. Balchunas, A. (2007). "Switching Tables v1.01."

[5]. Brandon, H. (2009). "OpenFlow Switch Specification
Version 1.0.0 (Wire Protocol 0x01)."

[6]. Gude, N., T. Koponen, et al. (2008). "NOX: Towards an Operating System for Networks."

[7]. "NOX | An OpenFlow Controller. *http://noxrepo.org/wp/"*

[8]. Yang, L., R. Dantu, et al. (2004). "Forwarding and Control Element Separation (ForCES) Framework."

[9]. Sherwood, R., G. Gibb, et al. (2010). "Can The Production Network Be the Testbed?".

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

[11]. McKeown, N. and G. Parulkar "Reinvent Internet Infrastructure with OpenFlow and Software Defined Networking."

[12]. "Django web framework. *http://www.djangoproject.com*"

[13]. "XML-RPC. *http://www.xmlrpc.com/spec*"

[14]. Burbeck, S. (1992). "Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)."

[15]. Holovaty, A. and J. Kaplan-Moss "The Django Book
The Definitive Guide to Django: Web Development Done Right."

[16]. Chen, P. P.-S. (1976). "The Entity-Relationship Model: Toward a Unified View of Data."

[17]. "Python Programming Language. *http://www.python.org*"

[18]. Crockford, D. (2009). "Introducing JSON."

[19]. Sternberg, E. (2010). "Django Signals enable Aspect-Oriented programming."

[20]. "Ubuntu Linux. *http://www.ubuntu.com*"

[21]. "FlowVisor Deploy. *http://openflow.stanford.edu/display/flowvisor/fv_deploy*"

[22]. Sherwood, R., M. Chan, et al. (2009). "Carving Research Slices Out of Your Production Networks with OpenFlow."

[23]. Yap, K.-K., T.-Y. Huang, et al. (2009). "Towards Software-Friendly Networks."

Παράρτημα

Το Παράρτημα περιέχει τον κώδικα Python που υπάρχει σε κάθε ένα από τα αρχεία που αναλύθηκαν στο Κεφάλαιο 4.

Α. Κώδικας αρχείου models.py

```
from django.db import models
from django.db.models.signals import pre_delete
from django import forms
from django.contrib.auth.models import User
from django.forms import ModelForm
from pexpect import *
import time

class SliceInfo(models.Model):
    sliceName = models.CharField(max_length=200, verbose_name='Slice Name')
    adminMail = models.CharField(max_length=200, verbose_name='Administrator\'s e-Mail')
    ctrlIp = models.CharField(max_length=200, verbose_name='Controller\'s IP')
    ctrlPort = models.IntegerField(verbose_name='Controller\'s Port')
    managed_by = models.ForeignKey(User, null=True, blank=True)

    def __unicode__(self):
        return self.sliceName

def delete_sl(sender, **kwargs):
   name=kwargs['instance'].sliceName
   cmd='fvctl --passwd-file=/home/flowvisor07/fv-ui/fv_site/slices/rootutil deleteSlice '+str(name)
   (command_output,exitstatus)=run (cmd, withexitstatus=1)
   time.sleep(1)
   f = open('/home/flowvisor07/fv-ui/fv_site/slices/record','r')
   newdata=''
   for line in f:
      finder=line.rfind('=')
      if line[0:finder]<>name:
         newdata += line
   f.close()
   f = open('/home/flowvisor07/fv-ui/fv_site/slices/record','w')
   f.write(newdata)
   f.close()
pre_delete.connect(delete_sl, sender=SliceInfo)


class FlowSpace(models.Model):
    slice_Info = models.ForeignKey(SliceInfo)
    flow_space_id = models.CharField(max_length=20)
    datapath = models.CharField(max_length=25)
    in_port = models.CharField(max_length=10)
    vlan_tag = models.CharField(max_length=10)
    source_MAC = models.CharField(max_length=25)
    destination_MAC = models.CharField(max_length=25)
    ethertype = models.CharField(max_length=10)
    source_IP = models.CharField(max_length=30)
```

```
    destination_IP = models.CharField(max_length=30)
    protocol = models.CharField(max_length=10)
    tos_bits = models.CharField(max_length=10)
    source_port = models.CharField(max_length=10)
    destination_port = models.CharField(max_length=10)
    slice_action = models.CharField(max_length=10)
    slice_priority = models.CharField(max_length=10)

    def __unicode__(self):
        return self.flow_space_id
        super(FlowSpace, self).delete()

def delete_fv(sender, **kwargs):
    ids=kwargs['instance'].flow_space_id
    cmd='fvctl --passwd-file=/home/flowvisor07/fv-ui/fv_site/slices/rootutil removeFlowSpace '+ids
    (command_output,exitstatus)=run (cmd, withexitstatus=1)
    time.sleep(1)
pre_delete.connect(delete_fv, sender=FlowSpace)

class Datapath(models.Model):
    ACTIVE_CHOICES = (
        ('up', 'Connected'),
        ('dc', 'Disconnected'),
    )
    dpid = models.CharField(max_length=25)
    users_with_permission = models.ManyToManyField(User, null=True, blank=True)
    is_active = models.CharField(max_length=2, choices=ACTIVE_CHOICES, blank=True)

    def __unicode__(self):
        return self.dpid

class FlowspaceRestriction(models.Model):
    restricted_users = models.ManyToManyField(User, null=True, blank=True)
    dpid = models.ForeignKey(Datapath, null=True, blank=True)
    in_port = models.CharField(max_length=100, null=True, blank=True)
    vlan_tag = models.CharField(max_length=100, null=True, blank=True)
    source_MAC = models.CharField(max_length=250, null=True, blank=True)
    destination_MAC = models.CharField(max_length=250, null=True, blank=True)
    ethertype = models.CharField(max_length=100, null=True, blank=True)
    source_IP = models.CharField(max_length=300, null=True, blank=True)
    destination_IP = models.CharField(max_length=300, null=True, blank=True)
    protocol = models.CharField(max_length=100, null=True, blank=True)
    tos_bits = models.CharField(max_length=100, null=True, blank=True)
    source_port = models.CharField(max_length=100, null=True, blank=True)
    destination_port = models.CharField(max_length=100, null=True, blank=True)
    slice_action = models.CharField(max_length=100, null=True, blank=True)
    slice_priority = models.CharField(max_length=100, null=True, blank=True)

    def __unicode__(self):
        return self.dpid.dpid
```

Β. Κώδικας αρχείου views.py

```
from django.http import HttpResponse
from subprocess import call, PIPE
from django.contrib.auth.models import User
from django.http import HttpResponseRedirect
from django.shortcuts import render_to_response
from slices.models import SliceInfo, FlowSpace, Datapath, FlowspaceRestriction
from pexpect import *
from django.core import serializers
from utilities import Check_ip, Check_num
from django.conf import settings
from django.contrib import admin
from django.utils.safestring import mark_safe
from django.utils.text import capfirst
from django.contrib.sites.models import Site
import subprocess
import time

def createsl(request):
  user_dict=User.objects.all()
  if request.user.is_superuser:
      return render_to_response('create_slice.html',{'users':user_dict, 'user':request.user.username})
  else:
      message = 'You are not authorized. Only Super-Users can create Slices'
      return HttpResponse(message)

def createfs(request):
  if request.user.is_staff:
    slice_dict=SliceInfo.objects.filter(managed_by=request.user)
    b=Datapath.objects.filter(users_with_permission=request.user)
    dpid_dict=b.filter(is_active='up')
    if request.user.is_superuser:
     slice_dict=SliceInfo.objects.all()
     b=Datapath.objects.all()
     dpid_dict=b.filter(is_active='up')
    return render_to_response('create_flowspace.html',{'user':request.user.username,'slices':slice_dict,'dpid':dpid_dict})
  else:
      message = 'Plz log in'
      return HttpResponse(message)

def cslice(request):
 name = request.GET['slicename']
 ip = request.GET['ip']
 port = request.GET['port']
 mail = request.GET['mail']
 userer = request.GET['username']
 f = open('/home/flowvisor07/fv-ui/fv_site/slices/record','a')
```

```
 record=name+'='+userer+'\n'
 f.write(record)
 fullip='tcp:'+ip+':'+port
 cmd='fvctl '+'--passwd-file=/home/flowvisor07/fv-ui/fv_site/slices/rootutil '+'createSlice '+name+' '+fullip+' '+mail
 (command_output,exitstatus)=run (cmd, events={'(?i)New password:':'pass\n'}, withexitstatus=1)
 outpt=command_output.splitlines()
 result=outpt[1]
 if result=='success!':
  p=User.objects.get(username=userer)
  user_id=p.id
  a=User.objects.get(pk=user_id)
  b=a.sliceinfo_set.create(sliceName=name,adminMail=mail, ctrlIp=ip, ctrlPort=port)
  b.save()
  b=SliceInfo.objects.get(sliceName=name)
  b=b.id
  return HttpResponseRedirect('/admin/slices/sliceinfo/%s/' % str(b))
 else:
  return HttpResponse("Creation Failed")

def cflowspace(request):
  userer=request.user.username
  p=User.objects.get(username=userer)
  user_id=p.id
  slice_dict=SliceInfo.objects.all()

  dpidG='any'
  if 'datapath' in request.GET:
   dpidG = request.GET['datapath']
  elif not request.user.is_superuser:
   return HttpResponse('You are not a super-user. You can\'t add Flowspace for \'any\' dpid')
  if not request.user.is_superuser:
   if Datapath.objects.filter(dpid=dpidG).exists():
    d=Datapath.objects.get(dpid=dpidG)
    if FlowspaceRestriction.objects.filter(restricted_users=user_id).exists():
     a=FlowspaceRestriction.objects.filter(restricted_users=user_id)
     rest="asd"
     if a.filter(dpid=d.id).exists():
      rest="asd"
      b=a.get(dpid=d.id)
     else:
      rest="free"
    else:
     rest="free"

  slice_prio = request.GET['slice_priority']
  if not request.user.is_superuser and not rest=="free":
   result = Check_num(str(b.slice_priority),1,100000,str(slice_prio))
   if result=="decline":
    return HttpResponse('Slice priority number not permitted!')
```

```
flow={}

flow[1] = request.GET['in_port']
if not request.user.is_superuser and not rest=="free":
 result = Check_num(str(b.in_port),1,100,str(flow[1]))
 if result=="decline":
   return HttpResponse('Selected in-port not permitted!')

flow[2] = request.GET['vlan_tag']
check=str(flow[2])
if check[0:2]=='0x' :
  flow[2]=str(int(check, 16))
if not request.user.is_superuser and not rest=="free":
 result = Check_num(str(b.vlan_tag),0,4095,str(flow[2]))            ##min-max = 0-4095
 if result=="decline":
   return HttpResponse('Selected vlan-tag not permitted!')



flow[3] = request.GET['source_MAC']
if not request.user.is_superuser and not rest=="free":
 result = Check_num(str(b.source_MAC),1,1,str(flow[3]))           ##min-max = random
 if result=="decline":
   return HttpResponse('Selected source MAC not permitted!')

flow[4] = request.GET['destination_MAC']
if not request.user.is_superuser and not rest=="free":
 result = Check_num(str(b.destination_MAC),1,1,str(flow[4]))           ##min-max = random
 if result=="decline":
   return HttpResponse('Selected destination MAC not permitted!')

flow[5] = request.GET['ethertype']
check=str(flow[5])
if check[0:2]=='0x' :
  flow[5]=str(int(check, 16))
if not request.user.is_superuser and not rest=="free":
 result = Check_num(str(b.ethertype),0,65535,str(flow[5]))           ##min-max = 0-65535
 if result=="decline":
   return HttpResponse('Selected ethertype not permitted!')

flow[6] = request.GET['source_IP']
if not request.user.is_superuser and not rest=="free":
 result = Check_ip(str(b.source_IP),str(flow[6]))
 if result=="decline":
   return HttpResponse('Selected source IP not permitted!')

flow[7] = request.GET['destination_IP']
if not request.user.is_superuser and not rest=="free":
 result = Check_ip(str(b.destination_IP),str(flow[7]))
 if result=="decline":
   return HttpResponse('Selected destination IP not permitted!')
```

```
flow[8] = request.GET['protocol']
if not request.user.is_superuser and not rest=="free":
 result = Check_num(str(b.protocol),0,255,str(flow[8]))            ##min-max = 0-255
 if result=="decline":
   return HttpResponse('Selected protocol not permitted!')

flow[9] = request.GET['tos_bits']
if not request.user.is_superuser and not rest=="free":
 result = Check_num(str(b.tos_bits),0,255,str(flow[9]))            ##min-max = 0-255
 if result=="decline":
   return HttpResponse('Selected tos_bits not permitted!')

flow[10] = request.GET['source_port']
if not request.user.is_superuser and not rest=="free":
 result = Check_num(str(b.source_port),0,65535,str(flow[10]))            ##min-max = 0-65535
 if result=="decline":
   return HttpResponse('Selected source port not permitted!')

flow[11] = request.GET['destination_port']
if not request.user.is_superuser and not rest=="free":
 result = Check_num(str(b.destination_port),0,65535,str(flow[11]))            ##min-max = 0-65535
 if result=="decline":
   return HttpResponse('Selected destination port not permitted!')


slice_name = request.GET['slice_name']

slice_act = request.GET['slice_action']
if not request.user.is_superuser and not rest=="free":
 result = Check_num(str(b.slice_action),2,5,str(slice_act))            ##can be 2,4,5
 if result=="decline" or int(slice_act)==3:
   return HttpResponse('Selected slice action not permitted!')

flowname={}
flowname[1]='in_port'
flowname[2]='dl_vlan'
flowname[3]='dl_src'
flowname[4]='dl_dst'
flowname[5]='dl_type'
flowname[6]='nw_src'
flowname[7]='nw_dst'
flowname[8]='nw_proto'
flowname[9]='nw_tos'
flowname[10]='tp_src'
flowname[11]='tp_dst'


if 'id' in request.GET:
  fs_id=request.GET['id']
```

```
    cmd='fvctl '+'--passwd-file=/home/flowvisor07/fv-ui/fv_site/slices/rootutil '+'changeFlowSpace '+fs_id+' '+dpidG+'
'+slice_prio+' '
   else:
    cmd='fvctl '+'--passwd-file=/home/flowvisor07/fv-ui/fv_site/slices/rootutil '+'addFlowSpace '+dpidG+' '+slice_prio+' '

   flowcmd=''
   for i in range (1,len(flow)+1):
     if flow[i]=='':
       flow[i]='*'
     if flow[i]<>'*':
      flowcmd += flowname[i]+'='+flow[i]+","

   if flowcmd=='':
     flowcmd='any,'

   cmd += flowcmd[0:-1]+' '+'Slice:'+slice_name+'='+slice_act

   (command_output,exitstatus)=run (cmd, withexitstatus=1)
   z=command_output.splitlines()
   result=z[0]
   if result[0:7]=='success':
    flow_id=result[9:]
    for i in range (0,len(slice_dict)):
      if str(slice_dict[i])==str(slice_name):
        name_id=i+1
    if 'id' in request.GET:
      a=SliceInfo.objects.get(pk=name_id)
      b=FlowSpace.objects.get(flow_space_id=fs_id)
      b.slice_Info=a
      b.datapath=dpidG
      b.in_port=flow[1]
      b.vlan_tag=flow[2]
      b.source_MAC=flow[3]
      b.destination_MAC=flow[4]
      b.ethertype=flow[5]
      b.source_IP=flow[6]
      b.destination_IP=flow[7]
      b.protocol=flow[8]
      b.tos_bits=flow[9]
      b.source_port=flow[10]
      b.destination_port=flow[11]
      b.slice_action=slice_act
      b.slice_priority=slice_prio
      b.save()
      flow_id=fs_id
    else:
      a=SliceInfo.objects.get(pk=name_id)

b=a.flowspace_set.create(flow_space_id=flow_id,datapath=dpidG,in_port=flow[1],vlan_tag=flow[2],source_MAC=flow[3],desti
```

```
nation_MAC=flow[4],ethertype=flow[5],source_IP=flow[6],destination_IP=flow[7],protocol=flow[8],tos_bits=flow[9],source_p
ort=flow[10],destination_port=flow[11],slice_action=slice_act,slice_priority=slice_prio)
        b.save()
      b=FlowSpace.objects.get(flow_space_id=flow_id)
      b=b.id

      return HttpResponseRedirect('/admin/slices/sliceinfo/%s/' % str(name_id))
    else:
      return HttpResponse('Flowspace creation/change failed with error %s' %str(command_output))



def mainslices(request):
   a=FlowspaceRestriction.objects.get(pk=3)
   b=a.restricted_users.values('username')
   #name=b[1]['username']
   name=''
   for i in range (0,len(b)):
     name += b[i]['username']+', '
   return HttpResponse(name[0:-2])

def changefs(request):
  userer=request.user.username
  flow_dict={}
  slice_dict=SliceInfo.objects.all()
  message = {}
  for i in range (0,len(slice_dict)):

    a=SliceInfo.objects.get(sliceName=str(slice_dict[i]))
    if a.managed_by == request.user:
      b=FlowSpace.objects.filter(slice_Info=a.id)
      for j in range(0, len(b)):
        z=FlowSpace.objects.get(flow_space_id=b[j], slice_Info=a.id)
        message[b[j]]='SLICE : '+str(a)+'|'*(30-len(str(a)))+' DATAPATH : '+str(z.datapath)
  return render_to_response('change_flowspace.html',{'flowspaces':message,'user':userer,'total':len(message)})

def changefsfinal(request):
  if request.user.is_superuser:
    slice_dict=SliceInfo.objects.all()
    b=Datapath.objects.all()
    dpid_dict=b.filter(is_active='up')
  elif request.user.is_staff:
    slice_dict=SliceInfo.objects.filter(managed_by=request.user)
    b=Datapath.objects.filter(users_with_permission=request.user)
    dpid_dict=b.filter(is_active='up')
  else:
    return HttpResponse('You don\'t have permission to do that')
  fs_id = request.GET['selected_action']
  a=FlowSpace.objects.get(flow_space_id=fs_id)
  g=a.slice_Info
  name=g.sliceName
```

```
 return render_to_response('flowspace_change_form.html',{'user':request.user.username,'slices':slice_dict, 'dpid':a.datapath, 'prio':a.slice_priority, 'inport':a.in_port, 'act':a.slice_action, 'vlan':a.vlan_tag, 'src_mac':a.source_MAC, 'dst_mac':a.destination_MAC, 'ethertype':a.ethertype, 'src_ip':a.source_IP, 'dst_ip':a.destination_IP, 'proto':a.protocol, 'tos':a.tos_bits, 'src_port':a.source_port, 'dst_port':a.destination_port, 'id':fs_id, 'dpids':dpid_dict, 'slname':str(name) })

def power(request):
 if not request.user.is_superuser:
    # return Http404
  return HttpResponseRedirect("http://localhost:8000/admin/auth/user/")
 else:
  return render_to_response('power_choose.html',{'user':request.user.username})

site = admin.site
def applist(request):
        app_dict = {}
        user = request.user
        for model, model_admin in site._registry.items():
                app_label = model._meta.app_label
                has_module_perms = user.has_module_perms(app_label)

                if has_module_perms:
                        perms = model_admin.get_model_perms(request)

                        if True in perms.values():
                                model_dict = {
                                        'name': capfirst(model._meta.verbose_name_plural),
                                        'admin_url': mark_safe('/admin/%s/%s/' % (app_label, model.__name__.lower())),
                                        'perms': perms,
                                }
                                if app_label in app_dict:
                                        app_dict[app_label]['models'].append(model_dict)
                                else:
                                        app_dict[app_label] = {
                                                'name': app_label.title(),
                                                'app_url': app_label + '/',
                                                'has_module_perms': has_module_perms,
                                                'models': [model_dict],
                                        }

        app_list = app_dict.values()
        app_list.sort(lambda x, y: cmp(x['name'], y['name']))
        return (app_list)


def loaddpid(request):
 if not request.user.is_staff:
    # return Http404
 return HttpResponseRedirect("http://localhost:8000/admin/auth/user/")
 else:
```

```
 JSONserializer = serializers.get_serializer("json")
 json_serializer = JSONserializer()
 out = open("/home/flowvisor07/fv-ui/fv_site/slices/fixtures/datapath.json", "w")
 json_serializer.serialize(Datapath.objects.all(), stream=out, indent=2)
 out.close()
 cmdnewdata=("python /home/flowvisor07/fv-ui/fv_site/slices/dpidload.py")
 run (cmdnewdata)
 cmdjson = 'python /home/flowvisor07/fv-ui/fv_site/manage.py loaddata datapath.json'
 run (cmdjson)
 message='Datapaths Database is up-to-date'
 apps=applist(request)
 url='/admin/'
 return render_to_response('admin/index.html',{'msg':message, 'user':request.user, 'app_list':apps, 'urls':url})

def power_do(request):
 if not request.user.is_superuser:
     # return Http404
 return HttpResponseRedirect("http://localhost:8000/admin/auth/user/")
 else:
 act=request.GET['selected_action']
 cmdstop="/etc/init.d/flowvisor stop"
 cmdstart="/etc/init.d/flowvisor start"
 if act=='on':
  run (cmdstart)
  message="FLOWVISOR WAS POWERED ON!"
 elif act=='off':
  run (cmdstop)
  message="FLOWVISOR WAS POWERED OFF!"
 elif act=='restart':
  run (cmdstop)
  time.sleep(2)
  run (cmdstart)
  message="FLOWVISOR WAS RESTARTED!"
 else:
  message='NO ACTION SELECTED'
 apps=applist(request)
 url='../../'
 return render_to_response('admin/index.html',{'msg':message, 'user':request.user, 'app_list':apps, 'urls':url})

def ping(request):
  cmdping='fvctl --passwd-file=/home/flowvisor07/fv-ui/fv_site/slices/rootutil ping ok'
  (command_output,exitstatus)=run (cmdping, withexitstatus=1)

  if command_output[0:5]=='error':
   message = 'Flowvisor is down'
  else:
   out=command_output.splitlines()
   p1=out[1].rfind('::')
   ver=out[1][23:p1]
   message='FlowVisor is up and running! \n FlowVisor version:  '+ver
```

```
  apps=applist(request)
  url='/admin/'
  return render_to_response('admin/index.html',{'msg':message, 'user':request.user, 'app_list':apps, 'urls':url})

def filldb(request):
   if not request.user.is_superuser:
     # return Http404
     return HttpResponseRedirect("http://localhost:8000/admin/auth/user/")
   else:
    user_dict=User.objects.all()
    f = open('/home/flowvisor07/fv-ui/fv_site/slices/record2','w')
    for i in range (0,len(user_dict)):
     record=str(user_dict[i])+'='+str(i+1)+'\n'
     f.write(record)
    JSONserializer = serializers.get_serializer("json")
    json_serializer = JSONserializer()
    out = open("/home/flowvisor07/fv-ui/fv_site/slices/fixtures/datapath.json", "w")
    json_serializer.serialize(Datapath.objects.all(), stream=out, indent=2)
    out.close()
    out = open("/home/flowvisor07/fv-ui/fv_site/slices/fixtures/restrictions.json", "w")
    json_serializer.serialize(FlowspaceRestriction.objects.all(), stream=out, indent=2)
    out.close()
    subprocess.Popen(['python','/home/flowvisor07/fv-ui/fv_site/slices/filler.py'], stdout=PIPE)
    message='Slices Database was (re)populated!'
    apps=applist(request)
    url='/admin/'
    return render_to_response('admin/index.html',{'msg':message, 'user':request.user, 'app_list':apps, 'urls':url})
```

Γ. Κώδικας αρχείου urls.py

```
from django.conf.urls.defaults import *
from django.contrib.auth.views import login
from django.conf import settings

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    (r'^slices/filldb/$', 'slices.views.filldb'),
    (r'^slices/view', 'slices.views.mainslices'),
    (r'^slices/cslice/$', 'slices.views.cslice'),
    (r'^slices/cflowspace/$', 'slices.views.cflowspace'),
    (r'^slices/flowspace/fschange/$', 'slices.views.changefs'),
    (r'^slices/flowspace/change/$', 'slices.views.changefsfinal'),
    (r'^admin/checkup/$', 'slices.views.ping'),
    (r'^admin/power/$', 'slices.views.power'),
    (r'^admin/power/do/$', 'slices.views.power_do'),
    (r'^admin/loaddpid/$', 'slices.views.loaddpid'),
    (r'^admin/slices/sliceinfo/add/$', 'slices.views.createsl'),
    (r'^admin/slices/flowspace/add/$', 'slices.views.createfs'),
    (r'^media/(?P<path>.*)$', 'django.views.static.serve',
 {'document_root': settings.MEDIA_ROOT}),

     (r'^accounts/login/$', login),
     (r'^admin/', include(admin.site.urls)),
)
```

Δ. Κώδικας αρχείου admin.py

```
from slices.models import SliceInfo, FlowSpace, Datapath, FlowspaceRestriction
from django.contrib.auth.models import User
from django.db import models
from django.contrib import admin
from django.contrib.contenttypes.models import ContentType
from subprocess import call, PIPE
from django import forms
from pexpect import *
from django import forms
from django.contrib.admin.widgets import FilteredSelectMultiple
import subprocess

class FlowSpaceInline(admin.TabularInline):
    model = FlowSpace
    extra = 0

class FlowSpaceForm(forms.ModelForm):
    datapath = forms.CharField(widget=forms.TextInput(attrs={'size':'19','READONLY':'true'}))
    flow_space_id = forms.CharField(widget=forms.TextInput(attrs={'size':'7','READONLY':'true'}))
    in_port = forms.CharField(widget=forms.TextInput(attrs={'size':'4','READONLY':'true'}))
    vlan_tag = forms.CharField(widget=forms.TextInput(attrs={'size':'4','READONLY':'true'}))
    source_MAC = forms.CharField(widget=forms.TextInput(attrs={'size':'15','READONLY':'true'}))
    destination_MAC = forms.CharField(widget=forms.TextInput(attrs={'size':'15','READONLY':'true'}))
    ethertype = forms.CharField(widget=forms.TextInput(attrs={'size':'4','READONLY':'true'}))
    source_IP = forms.CharField(widget=forms.TextInput(attrs={'size':'12','READONLY':'true'}))
    destination_IP = forms.CharField(widget=forms.TextInput(attrs={'size':'12','READONLY':'true'}))
    protocol = forms.CharField(widget=forms.TextInput(attrs={'size':'4','READONLY':'true'}))
    tos_bits = forms.CharField(widget=forms.TextInput(attrs={'size':'4','READONLY':'true'}))
    source_port = forms.CharField(widget=forms.TextInput(attrs={'size':'4','READONLY':'true'}))
    destination_port = forms.CharField(widget=forms.TextInput(attrs={'size':'4','READONLY':'true'}))
    slice_action = forms.CharField(widget=forms.TextInput(attrs={'size':'1','READONLY':'true'}))
    slice_priority = forms.CharField(widget=forms.TextInput(attrs={'size':'7','READONLY':'true'}))
    class Meta:
        model = FlowSpace

class SliceInfoAdmin(admin.ModelAdmin):
    fieldsets = [
      ('Slice Name and Slice user\'s mail', {'fields': ['sliceName','adminMail']}),
      ('TCP connection INFO: ',          {'fields': ['ctrlIp','ctrlPort'], 'classes': ['collapse']}),
    ]
    inlines = [FlowSpaceInline]
    list_display = ('sliceName','adminMail','managed_by')
    def queryset(self, request):
      qs = super(SliceInfoAdmin, self).queryset(request)
      if request.user.is_superuser:
        return qs
```

```
        return qs.filter(managed_by=request.user)

class FlowSpaceAdmin(admin.ModelAdmin):
    fieldsets = [
        ('General Info',{'fields': ['slice_Info','datapath','flow_space_id','slice_priority','slice_action']}),
        ('Layer 2 attributes', {'fields': ['in_port', 'vlan_tag', 'source_MAC', 'destination_MAC']}),
        ('Layer 3 Attributes', {'fields': ['ethertype', 'source_IP', 'destination_IP']}),
        ('Layer 4 Attributes', {'fields': ['protocol', 'tos_bits', 'source_port', 'destination_port']}),
    ]
    def queryset(self, request):
        qs = super(FlowSpaceAdmin, self).queryset(request)
        if request.user.is_superuser:
            return qs
        a=SliceInfo.objects.filter(managed_by=request.user)
        return qs.filter(slice_Info=a)
    form = FlowSpaceForm
    list_display = ('slice_Info','datapath','slice_action','slice_priority','flow_space_id')

class DatapathForm(forms.ModelForm):
    dpid = forms.CharField(widget=forms.TextInput(attrs={'size':'4','READONLY':'true'}))
    users_with_permission = forms.ModelMultipleChoiceField(queryset=User.objects.all(),
    required=False, widget=FilteredSelectMultiple("verbose name", is_stacked=False))
    class Meta:
        model = Datapath

class DatapathAdmin(admin.ModelAdmin):
    def queryset(self, request):
        qs = super(DatapathAdmin, self).queryset(request)
        if request.user.is_superuser:
            return qs
        return qs.filter(users_with_permission=request.user)
    filter_horizontal = ('users_with_permission',)
    list_display = ('dpid', 'is_active')

    def get_form(self, request, obj=None, **kwargs):
                            if not request.user.is_superuser:
                self.fieldsets = [
                    ('Datapath ID', {'fields': ['dpid', 'is_active']}),
                ]
                self.readonly_fields='dpid'
            else:
                self.fieldsets = [
                ('Datapath ID', {'fields': ['dpid', 'users_with_permission', 'is_active']}),
                ]
                            return super(DatapathAdmin, self).get_form(request, obj, **kwargs)

class FlowspaceRestrictionForm(forms.ModelForm):
    restricted_users = forms.ModelMultipleChoiceField(queryset=User.objects.all(),
    required=False, widget=FilteredSelectMultiple("verbose name", is_stacked=False))
    class Meta:
```

```
        model = FlowspaceRestriction

class FlowspaceRestrictionAdmin(admin.ModelAdmin):
    form = FlowspaceRestrictionForm
    filter_horizontal = ('restricted_users',)
    list_display = ('dpid','upper_case_name','slice_action','slice_priority')

     def queryset(self, request):
        qs = super(FlowspaceRestrictionAdmin, self).queryset(request)
        if request.user.is_superuser:
            return qs
        return qs.filter(restricted_users=request.user)

    def upper_case_name(self, obj):
     ids=obj.pk
     a=FlowspaceRestriction.objects.get(pk=ids)
     b=a.restricted_users.values('username')
     name=''
     for i in range (0,len(b)):
        name += b[i]['username']+', '
     if name =='':
        name='Not assigned to any users yet  '
     return (name[0:-2])
    upper_case_name.short_description = 'Restricted users'

    def get_form(self, request, obj=None, **kwargs):
                            if not request.user.is_superuser:
                self.fieldsets = [
                  ('General Info',{'fields': ['dpid','slice_priority','slice_action']}),
                  ('Layer 2 attributes', {'fields': ['in_port', 'vlan_tag', 'source_MAC', 'destination_MAC']}),
                  ('Layer 3 Attributes', {'fields': ['ethertype', 'source_IP', 'destination_IP']}),
                  ('Layer 4 Attributes', {'fields': ['protocol', 'tos_bits', 'source_port', 'destination_port']}),
                ]
                self.readonly_fields='dpid'
            else:
                self.fieldsets = [
                  ('General Info',{'fields': ['restricted_users','dpid','slice_priority','slice_action']}),
                  ('Layer 2 attributes', {'fields': ['in_port', 'vlan_tag', 'source_MAC', 'destination_MAC']}),
                  ('Layer 3 Attributes', {'fields': ['ethertype', 'source_IP', 'destination_IP']}),
                  ('Layer 4 Attributes', {'fields': ['protocol', 'tos_bits', 'source_port', 'destination_port']}),
                ]
                            return super(FlowspaceRestrictionAdmin, self).get_form(request, obj, **kwargs)

admin.site.register(SliceInfo, SliceInfoAdmin)
admin.site.register(FlowSpace, FlowSpaceAdmin)
admin.site.register(Datapath, DatapathAdmin)
admin.site.register(FlowspaceRestriction, FlowspaceRestrictionAdmin)
```

Ε. Κώδικας αρχείου utilities.py

```
from subprocess import Popen, PIPE

def listSlices():
   stdout, stderr = Popen(['fvctl','--passwd-file=/home/flowvisor07/fv-ui/fv_site/slices/rootutil','listSlices'], shell=False, stdout=PIPE).communicate()
   slicedict={}
   l=0
   for i in range(0, len(stdout)):
      if stdout[i]==':':
        j=i+1
        while stdout[j]!='\n':
           j += 1
        if stdout[i+2:j] != 'root':
          slicedict[l] = stdout[i+2:j]
          l=l+1
   l -= 1
   return (slicedict,l)
dictls,l = listSlices()

def SliceInfo(dictls,l):
   maildict={}
   ctrl_ip_dict={}
   ctrl_port_dict = {}
   z = 0
   for i in range(0, l+1):
    stdout, stderr = Popen(['fvctl','--passwd-file=/home/flowvisor07/fv-ui/fv_site/slices/rootutil','getSliceInfo',dictls[i]], shell=False, stdout=PIPE).communicate()
     s=stdout.splitlines()
     for j in range(0, len(s)):
      e=s[j]
      m=e.rfind('email=')
      ip=e.rfind('hostname=')
      p=e.rfind('port=')
      if m<>-1:
        maildict[i]=e[m+6:]
      if ip<>-1:
        ctrl_ip_dict[i]=e[ip+9:]
      if p<>-1:
        ctrl_port_dict[i]=e[p+5:]
   return (maildict, ctrl_ip_dict, ctrl_port_dict)
```

```
def Check_num(rest, rest_min, rest_max, to_check):

#####   Input   ##########
## rest : restrictions ranges and values (INT)
## to_check : flowspace value to check against restrictions (STR)
## rest_min/max : min/max values for flowvisor

   if to_check[0:2]=='0x' :        #check for hex
     to_check=str(int(check, 16))
#####Brake rest rule##########

   check={}
   end=rest.rfind(';')
   i=0
   while end <> -1:
    start=rest[0:end-1].rfind(';')
    if start==-1:
     check[i]=rest[start+1:end]
     break
    else:
     check[i]=rest[start+1:end]
    end=start
    i += 1
   rest_range={}
   rest_solo={}
   j=0
   z=0
   for i in range (0, len(check)):
     point=check[i].rfind('-')
     if point == -1:
       rest_solo[z]=check[i]
       z +=1
     else:
       rest_range[j]=check[i][0:point]
       j += 1
       rest_range[j]=check[i][point+1:]
       j += 1
   result='decline'
   if rest=='*' or rest=='':
    result='accept'

   i=0
   if to_check != '*' and to_check !='':
    while i in range(0,len(rest_range)):
      if int(to_check) in range (int(rest_range[i]), int(rest_range[i+1])+1):
        result='accept'
      i += 2

   if to_check in rest_solo.values():
     result='accept'
```

```
    if rest != '*' and rest !='' and to_check != '*' and to_check !='':
      if to_check.rfind(':') == -1:  #####no min/max for MAC
        if int(to_check) < rest_min:
          result='decline'
        if int(to_check) > rest_max:
          result='decline'

    return (result)


def Check_ip(rest, to_check):
    check={}
    end=rest.rfind(';')
    i=0
    while end <> -1:
      start=rest[0:end-1].rfind(';')
      if start==-1:
        check[i]=rest[start+1:end]
        break
      else:
        check[i]=rest[start+1:end]
      end=start
      i += 1
    rest_range={}
    rest_solo={}
    j=0
    z=0
    for i in range (0, len(check)):
      point=check[i].rfind('-')
      if point == -1:
        rest_solo[z]=check[i]
        z +=1
      else:
        rest_range[j]=check[i][0:point]
        j += 1
        rest_range[j]=check[i][point+1:]
        j += 1

    result='decline'

    if rest=='*' or rest=='':
      result='accept'


    parts = to_check.split(".")
    if rest!='*' and rest !='':
      if len(parts) != 4 :
          result = 'decline'
          return (result)
```

```
    for item in parts:
        if not 0 <= int(item) <= 255:
            result = 'decline'
            return (result)

    i=0
    if len(parts) == 4:
      while i in range(0,len(rest_range)):
        parts_s=rest_range[i].split(".")
        parts_e=rest_range[i+1].split(".")
        if len(parts_s)==4 and len(parts_e)==4:
          p=0
          for q in range(0, 4):
            if parts[q] != '':
             if int(parts[q]) in range (int(parts_s[q]), int(parts_e[q])+1):
                p += 1
                if p==4:
                  result='accept'
        i += 2

    if to_check in rest_solo.values():
      result='accept'

    return (result)
```

ΣΤ. Κώδικας αρχείου filler.py

```
from utilities import SliceInfo, listSlices
from subprocess import Popen, PIPE, call
from pexpect import *
import json
import subprocess

dictls,l = listSlices()
d1, d2, d3 = SliceInfo(dictls,l)
j=0
test = {}
test2 = {}
f = open('/home/flowvisor07/fv-ui/fv_site/slices/record','r')
for line in f:
    finder=line.rfind('=')
    test[line[0:finder]]=line[finder+1:-1]
f.close()
f = open('/home/flowvisor07/fv-ui/fv_site/slices/record2','r')
for line in f:
    finder=line.rfind('=')
    test2[line[0:finder]]=line[finder+1:-1]
f.close()
user_id={}
for r in range (0, l+1):
  if dictls[r] in test:
     user_id[r]=test2[test[dictls[r]]]
  else:
     user_id[r]=1

my_structure=[]
if 0 in dictls:

 my_structure=[

 {
  "model": "slices.SliceInfo",
  "pk": j+1,
  "fields": {
    "sliceName": dictls[j],
    "adminMail": d1[j],
    "ctrlIp": d2[j],
    "ctrlPort": d3[j],
    "managed_by": user_id[j]
   }
 },
 ]
 j +=1
```

```
 while j <= l :
      my_structure.append(
       {
        "model": "slices.SliceInfo",
        "pk": j+1,
        "fields": {
          "sliceName": dictls[j],
          "adminMail": d1[j],
          "ctrlIp": d2[j],
          "ctrlPort": d3[j],
          "managed_by": user_id[j]
         }
       },
      )
      j += 1

with open('/home/flowvisor07/fv-ui/fv_site/slices/fixtures/slices_init_db.json', mode='w') as f:
    json.dump(my_structure,f, indent=2)

cmdreset='python /home/flowvisor07/fv-ui/fv_site/manage.py reset slices'
(command_output,exitstatus)=run (cmdreset, events={'Type \'yes\' to continue, or \'no\' to cancel:':'yes\n'}, withexitstatus=1)
subprocess.call(['python','/home/flowvisor07/fv-
ui/fv_site/manage.py','loaddata','slices_init_db.json','datapath.json','restrictions.json'], stdout=PIPE)
stdout, stderr = Popen(['fvconfig','dump','/usr/etc/flowvisor/flowvisor-config.xml'], shell=False, stdout=PIPE).communicate()
j=0
z=5
linelist = stdout.splitlines()
counter = 0
flow_structure =[]
if len(linelist) != 0:
 for i in range(0, len(linelist)):
  z=linelist[i]
  if z[0:22] == '!flowspace::FLOWMAP : ':
    EndDpid = z.rfind("dpid=[") + 6
    StartOf = z.rfind("],ruleMatch=[OFMatch[")
    EndOf = StartOf + 21
    StartAction = z.rfind("]],actionsList=[Slice:")
    EndAction = StartAction + 22
    StartID = z.rfind("],id=[")
    EndID = StartID + 6
    StartPrio = z.rfind("],priority=[")
    EndPrio = StartPrio + 12
    EndLine = z.rfind("],]")
    if z[EndDpid:EndDpid+3] in ("any","all","ALL"):
      dpid = 'any'
    elif z[EndDpid:StartOf] == "00:ff:ff:ff:ff:ff:ff:ff" :
      dpid = 'any'
    else:
      dpid = z[EndDpid:StartOf]
    act=z[EndOf:StartAction]
```

```
sinp=act.rfind("in_port=")
einp=act.rfind("in_port=")+8
if sinp == -1 :
  inport = "*"
else :
  c=einp+1
  while (c < len(act)) and (act[c] <> ',') :
      c +=1
  inport = act[einp:c]
sdmac=act.rfind("dl_dst=")
edmac=act.rfind("dl_dst=")+8
if sdmac == -1 :
  d_mac = "*"
else :
  c=edmac+1
  while (c < len(act)) and (act[c] <> ',') :
      c +=1
  if act[edmac] == '0' :
    d_mac = '0'+act[edmac:c]
  else:
    d_mac = act[edmac:c]
ssmac=act.rfind("dl_dst=")
esmac=act.rfind("dl_dst=")+8
if ssmac == -1 :
  s_mac = "*"
else :
  c=esmac+1
  while (c < len(act)) and (act[c] <> ',') :
      c +=1
  if act[esmac] == '0' :
    s_mac = '0'+act[esmac:c]
  else:
    s_mac = act[esmac:c]
setht=act.rfind("dl_type=")
eetht=act.rfind("dl_type=")+8
if setht == -1 :
  ethtype = "*"
else :
  c=eetht+1
  while (c < len(act)) and (act[c] <> ',') :
      c +=1
  ethtype = act[eetht:c]
svlan=act.rfind("dl_vlan=")
evlan=act.rfind("dl_vlan=")+8
if svlan == -1 :
  vlan = "*"
else :
  c=evlan+1
  while (c < len(act)) and (act[c] <> ',') :
      c +=1
```

```
  vlan = act[evlan:c]
sdip=act.rfind("nw_dst=")
edip=act.rfind("nw_dst=")+7
if sdip == -1 :
  dest_ip = "*"
else :
  c=edip+1
  while (c < len(act)) and (act[c] <> ',') :
      c +=1
  dest_ip = act[edip:c]
ssip=act.rfind("nw_src=")
esip=act.rfind("nw_src=")+7
if ssip == -1 :
  src_ip = "*"
else :
  c=esip+1
  while (c < len(act)) and (act[c] <> ',') :
      c +=1
  src_ip = act[esip:c]
snetp=act.rfind("nw_proto=")
enetp=act.rfind("nw_proto=")+9
if snetp == -1 :
  net_proto = "*"
else :
  c=enetp+1
  while (c < len(act)) and (act[c] <> ',') :
      c +=1
  net_proto = act[enetp:c]
stos=act.rfind("nw_tos=")
etos=act.rfind("nw_tos=")+7
if stos == -1 :
  tos = "*"
else :
  c=etos+1
  while (c < len(act)) and (act[c] <> ',') :
      c +=1
  tos = act[etos:c]
sdpt=act.rfind("tp_dst=")
edpt=act.rfind("tp_dst=")+7
if sdpt == -1 :
  dst_prt = "*"
else :
  c=edpt+1
  while (c < len(act)) and (act[c] <> ',') :
      c +=1
  dst_prt = act[edpt:c]
sspt=act.rfind("tp_src=")
espt=act.rfind("tp_src=")+7
if sspt == -1 :
  src_prt = "*"
```

```
else :
  c=espt+1
  while (c < len(act)) and (act[c] <> ',') :
      c +=1
  src_prt = act[espt:c]
prio=z[EndPrio:EndLine]
sl_id=z[EndID:StartPrio]
slist=z[EndAction:StartID]
n=slist.count('=')
slice_action_dict = {}
start=0
c=1
end=len(slist)
for sl in range(0,n):
  while slist[c] <> '=' :
     c +=1
  slice_action_dict[slist[start:c]] = slist[c+1]
  start= c +9
  c += 10
for h in range(0,l+1):
   counter2 = 0
   if dictls[h] in slice_action_dict.keys():
      action = slice_action_dict[dictls[h]]
      pk_num = h + 1
      print dictls[h],pk_num,action
      counter2 += 1
      if counter == 0 and counter2 == 1 :
       g=1
       flow_structure=[

        {
         "model": "slices.FlowSpace",
         "pk": g,
         "fields": {
            "slice_Info" : pk_num,
            "flow_space_id" : sl_id,
            "datapath" : dpid,
            "in_port" : inport,
            "vlan_tag" : vlan,
            "source_MAC" : s_mac,
            "destination_MAC" : d_mac,
            "ethertype" : ethtype,
            "source_IP" : src_ip,
            "destination_IP" : dest_ip,
            "protocol" : net_proto,
            "tos_bits" : tos,
            "source_port" : src_prt,
            "destination_port" : dst_prt,
            "slice_action" : action,
            "slice_priority" : prio,
```

```
            }
           },
          ]
         else:

          flow_structure.append(

           {
            "model": "slices.FlowSpace",
            "pk": g,
            "fields": {
              "slice_Info" : pk_num,
              "flow_space_id" : sl_id,
              "datapath" : dpid,
              "in_port" : inport,
              "vlan_tag" : vlan,
              "source_MAC" : s_mac,
              "destination_MAC" : d_mac,
              "ethertype" : ethtype,
              "source_IP" : src_ip,
              "destination_IP" : dest_ip,
              "protocol" : net_proto,
              "tos_bits" : tos,
              "source_port" : src_prt,
              "destination_port" : dst_prt,
              "slice_action" : action,
              "slice_priority" : prio,
            }
           },
          )
         g += 1


    counter += 1

with open('/home/flowvisor07/fv-ui/fv_site/slices/fixtures/flowspace_init_db.json', mode='w') as f:
    json.dump(flow_structure,f, indent=2)
subprocess.call(['python','/home/flowvisor07/fv-ui/fv_site/manage.py','loaddata','flowspace_init_db.json'], stdout=PIPE)
```

Ζ. Κώδικας αρχείου dpidload.py

```
from pexpect import*
import json
cmdlistdpid='fvctl --passwd-file=/home/flowvisor07/fv-ui/fv_site/slices/rootutil listDevices'
(command_output,exitstatus)=run (cmdlistdpid, withexitstatus=1)
dpid={}
act={}
usr={}
if command_output==":
  print 'no dpid'
else:
  out=command_output.splitlines()
  for i in range(0,len(out)):
    p=out[i].find(':')
    dpid[i]=out[i][p+2:]
    act[dpid[i]]="up"
    usr[dpid[i]]=[]
total = len(dpid)
output_json=json.load(open('/home/flowvisor07/fv-ui/fv_site/slices/fixtures/datapath.json'))

for i in range(0, len(output_json)):
  dp=output_json[i]["fields"]["dpid"]
  if dp in dpid.values():
    act[dp]="up"
  else:
    dpid[total]=dp
    if str(dp)=='any':
      act[dp]="up"
    else:
      act[dp]="dc"
    total += 1
  usr[dp]=output_json[i]["fields"]["users_with_permission"]

j=0
my_structure=[]
if 0 in dpid:
 my_structure=[

 {
  "model": "slices.datapath",
  "pk": j+1,
 "fields": {
    "users_with_permission": usr[dpid[j]],
   "is_active": act[dpid[j]],
    "dpid": dpid[j],
   }
 },
```

```
]
j +=1
 while j <= total-1 :
      my_structure.append(
       {
        "model": "slices.datapath",
        "pk": j+1,
        "fields": {
          "users_with_permission": usr[dpid[j]],
          "is_active": act[dpid[j]],
          "dpid": dpid[j],
         }
      },
      )
      j += 1

with open('/home/flowvisor07/fv-ui/fv_site/slices/fixtures/datapath.json', mode='w') as f:
    json.dump(my_structure,f, indent=2)
```