ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη λογισμικού με χρήση ελεγκτή OpenFlow (OF controller) και υλοποίηση μηχανισμών δρομολόγησης πακέτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΡΕΝΤΙΦΗ ΧΡΙΣΤΟΦΟΡΟΥ

Επιβλέπων : Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη λογισμικού με χρήση ελεγκτή OpenFlow (OF controller) και υλοποίηση μηχανισμών δρομολόγησης πακέτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΡΕΝΤΙΦΗ ΧΡΙΣΤΟΦΟΡΟΥ

Επιβλέπων : Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την η Σεμπτεβρίου 2014.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Ευστάθιος Συκάς	Μιχαήλ Θεολόγου	Γεώργιος Στασινόπουλος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2014

(Υπογραφή)

.....

ΡΕΝΤΙΦΗΣ ΧΡΙΣΤΟΦΟΡΟΣ

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

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη και παρουσίαση ενός προγράμματος λογισμικού (εφαρμογής) που εφαρμόζεται σε ένα ελεγκτή δικτύου υπολογιστών, με στόχο την δρομολόγηση των δεδομένων στο δίκτυο με το ελάχιστο ενεργειακό κόστος και την συνεχή εναλλαγή της λειτουργίας (ενεργοποίηση/απενεργοποίηση) των συσκευών του δικτύου. Η επιλογή της κατάλληλης διαδρομής γίνεται αναλόγως του ενεργειακού προφίλ και το φορτίο των συσκευών που δραστηριοποιούνται στο δίκτυο αλλά και των αποφάσεων του διαχειριστή του δικτύου.

Για την ανάπτυξη της εφαρμογής αυτής χρησιμοποιείται ένα ερευνητικό πρωτόκολλο επικοινωνιών που ακούει στην ονομασία OpenFlow. Η χρήση του OpenFlow είναι σημαντική, διότι προσφέρει την δυνατότητα απόζευξης του πεδίου δεδομένων από το πεδίο ελέγχου του δικτύου. Έτσι υπάρχει ένα κεντροποιημένο δίκτυο που βασίζει την λειτουργία του σε ένα κεντρικό ελεγκτή, αντί σε επιμέρους συσκευές. Η ανάπτυξη της εφαρμογής γίνεται με την χρήση του πλέον βασικού ελεγκτή του πρωτοκόλλου OpenFlow, του ελεγκτή Beacon.

Επιπλέον χρησιμοποιήθηκε το λογισμικό Mininet.Το λογισμικό αυτό αποτελεί έναν εξομοιωτή δικτύου. Έχει τη δυνατότητα να εκτελεί ταυτόχρονα ένα σύνολο από τερματικά, δρομολογητές, μεταγωγείς Ethernet άλλα και των αντίστοιχων συνδέσμων σε ένα ενιαίο Linux Kernel(πυρήνα).

Τέλος για την εύρεση της ελάχιστης δυνατής ενεργειακής διαδρομής χρησιμοποιήθηκε το λογισμικό IBM ILOG CPLEX Optimization Studio (συχνά αναφέρεται ως CPLEX). Πρόκειται για ένα πακέτο λογισμικού βελτιστοποίησης που υποστηρίζει γλώσσα προγραμματισμού βελτιστοποίησης (OPL) η οποία έχει σχεδιαστεί ειδικά για τη βελτιστοποίηση συνδυαστικών προβλημάτων.

Λέξεις Κλειδιά

Πρωτόκολλο OpenFlow, Beacon controller, ροές, πίνακας ροών, Ενεργειακή δρομολόγηση, Mininet , CPLEX

Abstract

The purpose of this thesis is the development and presentation of a software program (application) that is applied to a controller computer network aimed at routing the data to the network with minimum energy costs and continuous operation switch (on / off) on the network. Choosing the right path is depending on the energy profile and the load of the devices operating on the network, but also the decisions of the network administrator.

For the development of this application using a research protocol communications hears the name OpenFlow. Using OpenFlow is important because it offers the possibility of decoupling the data field by field control network. So there is a centralized network bases its operation on a centralized controller, rather than individual devices. The development of the application is using the most basic controller protocol OpenFlow, controller Beacon.

Additional software used Mininet. To software is a network simulator. It has the capability to simultaneously perform a set of terminals, routers, Ethernet switches and other relevant links on a single Linux Kernel (kernel).

Finally finding the minimum possible energy path software used IBM ILOG CPLEX Optimization Studio (often referred to as CPLEX). This is a software package that supports optimization programming language (OPL) that has been designed specifically for combinatorial optimization problems.

Key Words

OpenFlow Protocol, Beacon controller, Flows, flow table, Eneergy aware routing, Mininet, CPLEX

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε κατά το ακαδημαϊκό έτος 2013-2014 στο Εθνικό Μετσόβιο Πολυτεχνείο. Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κ. Ευστάθιο Συκά για την εμπιστοσύνη που μου έδειξε αναθέτοντάς μου την παρούσα εργασία, αλλά και τη δυνατότητα που μου έδωσε να ασχοληθώ με την συγκεκριμένη θεματολογία. Επίσης, ευχαριστώ θερμά τον κ. Πάρι Χαραλάμπου, υποψήφιο διδάκτορα , για την καθοδήγησή του και την εξαιρετική συνεργασία που είχαμε.

Πίνακας περιεχομένων

ΣΧΗΜΑΤΑ

1 Εισαγωγή

1.1 Σκοπός διπλωματικής εργασίας

Στην σύγχρονη εποχή η χρήση και αξιοποίηση του διαδικτύου αυξάνεται ραγδαίως. Η δικτυακή χρήση των υπολογιστών και των τεχνολογικών συσκευών γενικότερα έχει γίνει αναπόσπαστο κομμάτι στην ζωή του ανθρώπου. Το γεγονός αυτό έχει ως αποτέλεσμα την εκπληκτική ανάπτυξη της δικτυακής υποδομής. Το γεγονός αυτό έχει ως αποτέλεσμα η σπατάλη σε ενέργεια που καταναλώνουν οι δικτυακές συσκευές να αυξάνεται. Η ανάγκη για εύρεση λύσεων εξοικονόμησης ενέργειας είναι ένας προβληματισμός που απασχολεί την κοινωνία όλο και περισσότερο στις μέρες μας.

Οι λύσεις και οι μέθοδοι αυτοί μπορούν να αναζητηθούν σε διαφορετικά επίπεδα δομής του δικτύου, αλλά το σημαντικότερο από αυτά παραμένει η ίδια η υποδομή του, η οποία αποτελείται από το σύνολο των συσκευών που το απαρτίζουν και συνδέονται μεταξύ τους, όπως οι δρομολογητές (routers), μεταγωγείς (switches) , επαναλήπτες (hub) , εξυπηρετητές (servers) και πελάτες (clients) .

Ο τρόπος δρομολόγησης των δικτυακών συσκευών μέχρι σήμερα παραμένει στατικός. Οι δρομολογητές και οι μεταγωγείς δημιουργούν διαδρομές στο δίκτυο με χρήση διάφορων αλγορίθμων (πχ ελάχιστης απόστασης – Dijkstra) για επικοινωνία των διαφόρων συσκευών μεταξύ τους. Οι αλγόριθμοι αυτοί κατασκευάστηκαν έτσι ώστε να επιλέγουν το συντομότερο μονοπάτι. Στις μέρες μας που υπάρχει ποικιλία στις ταχύτητες σύνδεσης αλλά και εφαρμογές με διαφορετικές ανάγκες ταχύτητας οι τρόποι αυτοί κρίνονται ανορθόδοξοι καθώς μπορεί να οδηγήσει στην μεγάλη αύξηση του φορτίου που επεξεργάζεται μια συσκευή ενώ ταυτόχρονα κάποια άλλη παραμένει

αδρανείς. Η αδράνεια μιας δικτυακής συσκευής είναι μια κατάσταση ανεπιθύμητη καθώς εμφανίζεται το φαινόμενο άσκοπης σπατάλης ενέργειας.

Η παρούσα διπλωματική εργασία έχει ως στόχο την ανάπτυξη ενός μηχανισμού που θα ελέγχει δυναμικά την δρομολόγηση των πακέτων υπολογίζοντας ανά τακτά χρονικά διαστήματα την οικονομικότερη οικονομική διαδρομή. Παράλληλα στόχος είναι η καταστολή της λειτουργίας (sleep mode) των συσκευών που είναι σε αδράνη κατάσταση αλλά και η ενεργοποίηση τους όταν αυτό κριθεί απαραίτητο.

Για την ανάπτυξη του μηχανισμού γίνεται η χρήση μιας νέας αρχιτεκτονικής, του OpenFlow, που παρέχει την δυνατότητα δημιουργίας εικονικών δικτυακών υποδομών σε δρομολογητές και δημιουργίας πολλαπλών διαδρομών σε ένα φυσικό δίκτυο χωρίς να είναι αναγκαία η επανεκτέλεση αλγορίθμου δημιουργίας συνδέσεων για κάθε διαδρομή από τον δρομολογητή.

1.2 Οργάνωση κειμένου

Η διπλωματική εργασία αποτελείται από συνολικά πέντε (5) κεφάλαια. Στο πρώτο κεφάλαιο αναπτύχθηκαν οι προβληματισμοί και οι ανάγκες που οδήγησαν στην δημιουργία της διπλωματικής αυτής εργασίας.

Στο κεφάλαιο 2, γίνεται η παρουσίαση της τεχνολογίας OpenFlow. Αναφέρεται η δομή της αρχιτεκτονικής και οι λόγοι για τους οποίους αναπτύχθηκε. Στην συνέχεια γίνεται μια παρουσίαση μιας ποικιλίας ελεγκτών που έχουν αναπτυχθεί, ενώ στο τέλος αναλύθηκαν οι δυνατότητες των OpenFlow δρομολογητών.

Στο κεφάλαιο 3 γίνεται μια εκτενής περιγραφή του ελεγκτή Beacon που χρησιμοποιήθηκε στην παρούσα εργασία. Αναφέρονται τα πλεονεκτήματα του και ο λόγος που επιλέχθηκε έναντι των υπολοίπων ελεγκτών που υπάρχουν στο εμπόριο σήμερα. Τέλος αναλύεται η δομή και η αρχιτεκτονική του Beacon.

Στο 4 κεφάλαιο περιγράφεται το λογισμικό Mininet που χρησιμοποιήθηκε για την εικονική δημιουργία της τοπολογίας μας. Παρουσιάζονται οι δυνατότητες και τα πλεονεκτήματα του. Επιπλέον γίνεται παρουσίαση και επεξήγηση της τοπολογίας που χρησιμοποιήθηκε στην παρούσα εργασία.

Τέλος στο 5 κεφάλαιο αναλύεται ο προβληματισμός που αναπτύχθηκε για την δρομολόγηση των πακέτων και την καταστολή των αδρανών συσκευών με στόχο τον περιορισμό άσκοπης σπατάλης ενέργειας. Γίνεται ανάλυση της διαδικασίας που ακολουθήσαμε για την ανάπτυξη του μηχανισμού. Επιπλέον, παρουσιάζεται το λογισμικό CPLEX και οι δυνατότητες που προσφέρουν τα λογισμικά που χρησιμοποιούν γλώσσα προγραμματισμού βελτιστοποίησης. Αναλύονται επίσης τα οφέλη και τα μειονεκτήματα που μπορεί να έχει μια αρχιτεκτονική SDN για την δρομολόγηση δεδομένων και την ασφάλεια των δικτύων υπολογιστών στα οποία αναπτύσσεται.

2 *Η τεχνολογία OpenFlow*

2.1 Το πρωτόκολλο openflow

Κοιτάζοντας πίσω στο παρελθόν τον τρόπο εφαρμογής των δικτύων πριν από 20 χρόνια και συγκρίνοντας τα με την σημερινή εικόνα της δικτύωσης, είναι προφανές ότι τα δίκτυα έχουν εξελιχθεί δραματικά σε ότι αναφορά την ταχύτητα, την αξιοπιστία αλλά και την ασφάλεια. Οι τεχνολογίες σε επίπεδο φυσικού στρώματος έχουν εξελιχθεί παρέχοντας συνδέσεις υψηλής ταχύτητας, η υπολογιστική ισχύς των δικτυακών συσκευών έχει βελτιωθεί αισθητά και μια τεράστια ποικιλία δικτυακών εφαρμογών έχει δημιουργηθεί. Παρολαυτά δεν παρουσιάζεται μεγάλη αλλαγή στη δομή του δικτύου από την εγκατάσταση του εως σήμερα. Στην υφιστάμενη υποδομή, οι σύνθετες εργασίες που συμβάλλουν στην λειτουργικότητα του δικτύου, όπως η δρομολόγηση ή οι εργασίες που αφορούν την πρόσβαση στο δίκτυο,είναι αποφάσεις που ανατίθενται στις δικτυακές συσκευές από διαφορετικούς παρόχους που όλοι τους τρέχουν διαφορετικό υλικολογισμικό (firmware). Αυτή η καθιερωμένη ιδιόκτητη βάση εξοπλισμού απαρτίζει ολόκληρη την υποδομή του δικτύου και δεν δίνει αρκετό χώρο για νέες ερευνητικές ιδέες όπως νέα πρωτόκολλα δρομολόγησης τα οποία πρέπει να δοκιμαστούν σε ευρεία κλίμακα, σε πραγματικά δίκτυα.

Επιπλέον η είσοδος του διαδικτύου αλλά και γενικότερα των τεχνολογιών που βασίζονται στην δικτύωση,αλλά και στην καθημερινότητα του ανθρώπου , αποτελεί έναν αποτρεπτικό παράγοντα για οποιαδήποτε προσπάθεια πειραματισμού και προσπάθεια αλλαγών στο δίκτυο. Αυτός ουσιαστικά είναι και ο πλέον πιο σημαντικός λόγος για τον οποίο η υποδομή του δικτύου έχει παραμείνει σε συμβατικές τεχνολογίες καθώς κάποιος λανθασμένος χειρισμός ενδεχομένως να καταστρέψει ένα

μεγάλο μέρος του δικτύου με αποτέλεσμα να αποτελέσει σημαντικό πρόβλημα στις εργασίες που επιτελούνται με την χρήση του.

Η ανάγκη να αντιμετωπιστούν αυτές οι θεμελιώδεις ελλείψεις στην δικτύωση που παρουσιάζονται σήμερα, είχε ως αποτέλεσμα μια ερευνητική συνεργασία έξι ετών μεταξύ του Πανεπιστημίου Stanford και του Πανεπιστήμιο της Καλιφόρνια στο Μπέρκλεϊ. Η συνεργασία αυτή ξεκίνησε το 2008 και οδήγησε στην ανάπτυξη της ιδέας του OpenFlow.

Το OpenFlow αποτελεί ένα προγραμματιζόμενο πρωτόκολλο δικτύου που δημιουργήθηκε με σκοπό την διαχείριση και την κατεύθυνση της κίνησης των δεδομένων μεταξύ των δρομολογητών (router) ,των μεταγωγέων (switch) και των επαναληπτών (hub). Βασικός στόχος της έρευνας ήταν η δημιουργία ενός προγραμματιζόμενου και ανοικτού περιβάλλοντος δικτύωσης για δοκιμή και εφαρμογή νέων τεχνολογιών, μεθόδων και αλγορίθμων δρομολόγησης και ασφάλειας δικτύου.

Οι δικτυακές συσκευές μέσω τον οποίων γίνεται προώθηση των πακέτων ονομάζονται δρομολογητές ή μεταγωγείς. Οι μεταγωγείς (switches) αποτελούν συσκευές με δυνατότητες προώθησης πακέτων από την θύρα εισόδου στην θύρα εξόδου σύμφωνα με την επικεφαλίδα του πακέτου και δουλεύουν στο επίπεδο 2 (ζεύξης δεδομένων – data link layer) του συστήματος OSI. Η λειτουργία της μεταγωγής πραγματοποιείται βάση των πινάκων προώθησης μέσω των οποίων λαμβάνεται η απόφαση για την θύρα εξόδου. Αντίθετα Οι δρομολογητές (routers) αποτελούν πιο πολύπλοκες συσκευές οι οποίες έχουν την δυνατότητα απόφασης δρομολόγησης του πακέτου και μπορούν να ελέγχουν την επικεφαλίδα IP για τον τελικό προορισμό του πακέτου. Αυτό συμβαίνει καθώς ο δρομολογητής λειτουργεί στο επίπεδο 3 του OSI (επίπεδο δικτύου – network layer) και έχουν την δυνατότητα δημιουργίας και διαχείρισης πολλαπλών δικτύων καθώς και λειτουργίας σαν πύλης εξόδου (gateway) και διαθέτουν πίνακες προώθησης, όπως οι μεταγωγείς. Τέλος η πλήμνη (hub) (concentrator, χρησιμοποιούνται και οι όροι επαναλήπτης , συγκεντρωτής, διανομέας) είναι μια συσκευή στην οποία συνδέονται δικτυακοί κόμβοι μέσω καλωδίων

συνεστραμμένων ζεύγων ή οπτικής ίνας ώστε να δρουν ως ενιαίο τμήμα. Κυρίως χρησιμοποιείται σε τοπικά δίκτυα ethernet. Οι αναμεταδότες λειτουργούν στο φυσικό επίπεδο (στρώμα 1) του μοντέλου OSI. Η συσκευή είναι μια μορφή αναμεταδότη πολλαπλών θυρών. Οι πλήμνες ethernet είναι επίσης υπεύθυνες για την προώθηση ενός σήματος συμφόρησης σε όλες τις θύρες, εφόσον εντοπιστεί κάποια σύγκρουση.

Το OpenFlow αξιοποιεί την ύπαρξη των πινάκων αντιστοιχίας στους σύγχρονους δρομολογητές και τους μεταγωγείς. Κύριο χαρακτηριστικό της τεχνολογίας αυτής είναι οι πίνακες ροής (flow tables) που λειτουργούν σε ταχύτητες ανάλογες με των γραμμών μεταφοράς και βρίσκουν εφαρμογή για firewall, NAT(Network address translation), QoS(Quality of Service) ή ακόμα και στο να συλλέγουν διάφορα στατιστικά, και ποικίλουν ανάλογα τον εκάστοτε πάροχο. Ωστόσο η ομάδα που δημιούργησε την νέα αυτή τεχνολογία, εντόπισε ένα κοινό σύνολο λειτουργιών που υποστηρίζονται από τους περισσότερους μεταγωγείς και δρομολογητές. Ο εντοπισμός αυτός δίνει την δυνατότητα δημιουργίας ενός τυποποιημένου τρόπου χειρισμού των πινάκων ροών και μπορούν να χρησιμοποιηθούν σε όλες τις συσκευές του δικτύου ανεξάρτητα των κατασκευαστή. Με αυτόν τον τρόπο η κίνηση δεδομένων στο δίκτυο μπορεί να οργανωθεί σε διάφορες ροές που μπορούν να ομαδοποιηθούν και να απομονωθούν προκειμένου να δρομολογούνται, να επεξεργάζονται ή να ελέγχονται με οποιονδήποτε επιθυμητό τρόπο. Το OpenFlow πρωτόκολλο συνέβαλλε στην διαμόρφωση και τον έλεγχο νέων μεθόδων δικτύωσης σε ήδη υπάρχοντα και λειτουργικά δίκτυα χωρίς να παρεμβαίνει στην λειτουργία των ήδη ενεργών πρωτοκόλλων δρομολόγησης και ασφάλειας.

Το OpenFlow μπορεί να εφαρμοστεί σε πανεπιστημιακά δίκτυα όπου η απομόνωση της έρευνας και η δημιουργία κίνησης δεδομένων από τους χρήστες των ακαδημαϊκών δικτύων ,αποτελούν λειτουργίες ζωτικής σημασίας. Ροές μπορούν να δημιουργηθούν και να διατηρηθούν από μια κεντρική οντότητα που ονομάζεται Ελεγκτής (controller).Ο Ελεγκτής μπορεί να επεκταθεί προκειμένου να εκτελέσει πρόσθετες λειτουργίες όπως η δρομολόγηση και οι αποφάσεις για την πρόσβαση στο δίκτυο. Καταργώντας την εκτέλεση αυτών των λειτουργιών από τις δικτυακές συσκευές που βρίσκονται διασκορπισμένες κατά μήκος του δικτύου μπορεί κανείς να

το ελέγχει κεντρικοποιημένα χρησιμοποιώντας τους αντίστοιχους ελεγκτές που αναλαμβάνουν την εκτέλεση τους. Οι μόνες απαιτήσεις προκειμένου να εφαρμοστεί η τροποποίηση αυτή είναι οι δρομολογητές που μπορούν να υποστηρίξουν το OpenFlow και η ύπαρξη ενός κεντρικού ελεγκτή (controller) που. Με το τρόπο αυτόν ο έλεγχος και το επίπεδο των δεδομένων δεν είναι πλέον συνδυασμένα σε μία μόνο συσκευή δικτύου, άλλα διαχωρίζονται και συνδέονται δυναμικά το ένα με το άλλο. Ο διαχωρισμός των λειτουργιών ελέγχου και το επίπεδο των δεδομένων και η υιοθέτηση ενός κεντρικά ελεγχόμενου μοντέλου δικτύου, είναι έννοιες που έχουν συζητηθεί και προσεγγίζονται από τους ερευνητές πριν. Οι προσπάθειες σαν τα ForCES και SoftRouter, έχουν προτείνει αρχιτεκτονικές για να επιτρέπεται η αποσύνδεση του ελέγχου και η λειτουργικότητα του επίπεδου δεδομένων των συσκευών δικτύου, στοχεύοντας έτσι στην παροχή μιας πιο αποτελεσματικής προώθησης και μεγαλύτερης ευελιξίας στις λειτουργίες ελέγχου. Το OpenFlow μοιράζεται πολλά κοινά σημεία με αυτές τις αρχιτεκτονικές ,έχοντας όμως σαν πλεονέκτημα την εισαγωγή της έννοιας των ροών και η αξιοποίηση της ύπαρξης των πινάκων ροής στους εμπορικούς δρομολογητές

Η νέα αυτή μέθοδος δικτύωσης βρίσκει εφαρμογή σε ήδη υπάρχοντα δίκτυα χωρίς να παρεμβαίνει στην λειτουργία των ενεργών πρωτοκόλλων δρομολόγησης και ασφάλειας. Το γεγονός αυτό διασφαλίζεται με την χρήση της έννοιας της εικονικοποίησης (virtualization),δηλαδή την δημιουργία εικονικής μηχανής σε μία δικτυακή συσκευή όπως ο μεταγωγέας ή ο δρομολογητής, όπου το λειτουργικό σύστημα της συσκευής αυτής του OpenFlow, θα λειτουργεί ταυτοχρόνως με την μηχανή αυτή. Έτσι όλα τα πακέτα που λαμβάνει η συσκευή, διαχειρίζονται από το ίδιο το λειτουργικό σύστημα της συσκευής και σε περίπτωση που κάποιο πακέτο ανήκει στο περιβάλλον OpenFlow, προωθείται στην εικονική μηχανή.

Αναλυτικότερα το OpenFlow αποτελεί το κυριότερο εκπρόσωπο της τεχνολογίας software defined networking (SDN) , αποδεκτή από αρκετούς ερευνητές λόγω των πλεονεκτημάτων της, και χρησιμοποιείται από κατασκευαστές συσκευών δικτύου. Σημαντικό πλεονέκτημα της τεχνολογίας αποτελεί το γεγονός οτι λειτουργεί ανεξάρτητα από την ήδη υπάρχουσα δομή της συσκευής με αποτέλεσμα να μην

απαιτείται οποιαδήποτε παρέμβαση στις εσωτερικές λειτουργίες, ακόμα και στην αρχιτεκτονική της. Έτσι εκλείπουν οι κίνδυνοι λανθασμένης λειτουργίας της εκάστοτε συσκευής, γεγονός που θα μπορούσε να αποτελέσει σημαντικό κίνδυνο για την ακεραιότητα (integrity) και ανθεκτικότητα (robustness) του δικτύου στο οποίο εγκαθίσταται η εν λόγω συσκευή.

2.2 Το OpenFlow Δίκτυο

Οι κύριες μονάδες ενός δικτύου που υποστηρίζει OpenFlow τεχνολογία είναι οι εξής:

- Δρομολογητές που υποστηρίζουν OpenFlow
- Εξυπηρετητές που εκτελούν τις διεργασίες του ελεγκτή
- Βάση δεδομένων που περιέχουν την τοπολογία του δικτύου

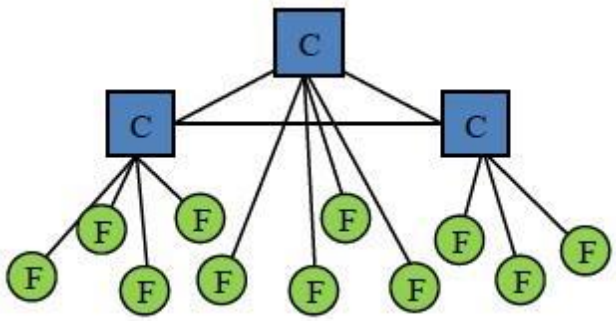
Οι δρομολογητές OpenFlow, αποτελούνται από πίνακες ροής που περιέχουν καταχωρήσεις ροής, οι οποίες χρησιμοποιούνται για την εκτέλεση αναζήτησης και προώθησης του πακέτου, και από ένα ασφαλές κανάλι που συνδέει την συσκευή με τον ελεγκτή OpenFlow για την ασφαλή ανταλλαγή μηνυμάτων.

Ο ελεγκτής αποτελεί μία βασική, κεντρική οντότητα η οποία εκτελεί όλες τις διεργασίες ελέγχου του OpenFlow δικτύου. Μέχρι σήμερα, έχουν αναπτυχθεί αρκετές πλατφόρμες ελεγκτών της τεχνολογίας OpenFlow όπως είναι η NOX, POX, Floodlight, η Beacon αλλά και αρκετές ακόμα. Στα πλαίσια της παρούσας διπλωματικής εργασία έγινε χρήση της πλατφόρμας Beacon όπως θα αναφερθεί και παρακάτω.

Επιπλέον ένας ελεγκτής παρέχει διεπαφή για την δημιουργία, την τροποποίηση και τον έλεγχο των πινάκων ροών των συσκευών που ελέγχει. Λειτουργεί συνήθως σε κάποιον διακομιστή που είναι συνδεδεμένος στο δίκτυο και ελέγχει είτε το σύνολο των συσκευών είτε μια ομάδα αυτών. Συνεπώς η λειτουργία ελέγχου του δικτύου μπορεί να γίνεται από έναν ή και περισσότερους ελεγκτές.

Η αρχιτεκτονική του πρωτοκόλλου OpenFlow βασίζεται στις αρχές λειτουργίας του Software defined networking (SDN). Η έννοια SDN αποτελεί μια νέα προσέγγιση στον σχεδιασμό, την κατασκευή και την διαχείριση των δικτύων. Η βασική ιδέα της έννοιας εντοπίζεται στο ότι διαχωρίζεται ο έλεγχος του δικτύου από το επίπεδο προώθησης των πακέτων, γεγονός που καθιστά ευκολότερο την βελτιστοποίηση του κάθε τομέα ξεχωριστά. Στο περιβάλλον αυτό ο ελεγκτής (Controller) ενεργεί ως "εγκέφαλος" παρέχοντας μια γενικότερη εικόνα του δικτύου. Μέσω του ελεγκτή, οι διαχειριστές του δικτύου μπορούν γρήγορα και εύκολα να πάρουν αποφάσεις για το πώς τα συστήματα που βασίζεται η προώθηση των πακέτων (δρομολογητές και μεταγωγείς) θα διαχειριστούν την κίνηση. Το OpenFlow αποτελεί το πιο γνωστό πρωτόκολλο που χρησιμοποιείται στα δίκτυα SDN και διευκολύνει την επικοινωνία μεταξύ του ελεγκτή (που ονομάζεται Southbound API) και των συσκευών δρομολόγησης. Αυτό μπορεί να επιτευχθεί εφαρμόζοντας το πεδίο ελέγχου σε κατάλληλο λογισμικό σε εξυπηρετητές και το πεδίο δεδομένων σε συσκευές του δικτύου όπως είναι οι μεταγωγείς και οι δρομολογητές.

Αυτό ακριβώς το μοντέλο υιοθετείται και από το OpenFlow, που είναι και ο κυριότερος εκπρόσωπος του SDN, αναθέτοντας τον έλεγχο της δρομολόγησης σε ένα αυτόνομο ελεγκτή (stand-alone controller) ο οποίος τρέχει στον εξυπηρετητή και επικοινωνεί άμεσα με τις υπόλοιπες συσκευές του δικτύου (προώθησης δεδομένων – data forwarding devices) που έχουν την δυνατότητα προώθησης πακέτων συμβατή με την αρχιτεκτονική του OpenFlow. Σχηματικά αυτό παρουσιάζεται πιο κάτω:



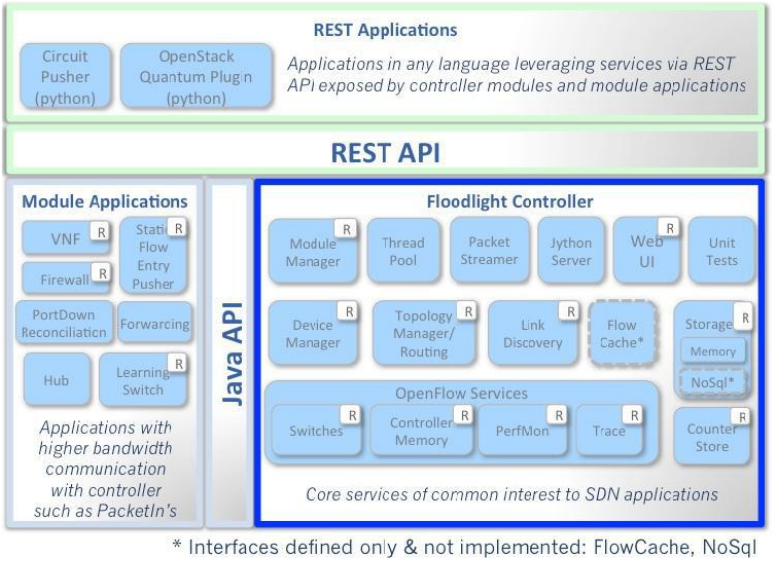
Σχήμα 2-1 Controller and Forwarding Layers

Όπως φαίνεται και στο σχήμα 2-1 είναι δυνατό να υπάρχουν πολλαπλοί OpenFlow ελεγκτές, οι οποίοι να επικοινωνούν αμφιμονοσήμαντα μεταξύ τους και ο κάθε controller να επικοινωνεί με τις δικές του συσκευές προώθησης πακέτων.

Οι διαδικασίες προώθησης και δρομολόγησης δεδομένων επιτελούνται από τις συσκευές προώθησης βάση των καταχωρίσεων ροής (flow entries) όπως αυτές αποστέλλονται στην συσκευή από τον ελεγκτή. Αυτό προσφέρει σημαντική ευελιξία ως προς την δυναμική ανάθεση πόρων από τον ελεγκτή καθώς και την δυνατότητα δημιουργίας ροών δεδομένων στο δίκτυο ανάλογα με την ζήτηση και την εφαρμογή που εκτελείται ανά δεδομένη χρονική στιγμή.

2.2.1 Ο ελεγκτής Floodlight

Ο Floodlight αποτελεί ένα λειτουργικό σύστημα που έχει ως στόχο τον κεντρικό έλεγχο εφαρμογών του δικτύου. Ο ελεγκτής αποτελείται από μια αυτόνομη συλλογή λειτουργικών μονάδων που επιτελούν τις κύριες λειτουργίες του Floodlight ως ελεγκτής OpenFlow καθώς και εφαρμογές που αναπτύσσονται με τρόπο ώστε να υπερτίθενται (on-top) της βασικής μονάδας του ελεγκτή (REST applications) ή να λειτουργούν μαζί με τον ελεγκτή (Module applications) , όπως παρουσιάζεται στο ακόλουθο σχήμα:



Σχήμα 2-2 Floodlight controller και REST applications

Όπως παρουσιάζεται πιο πάνω, ο Floodlight αποτελείται από τις λειτουργικές μονάδες τοπολογίας και ζεύξης (Topology Manager, Link Discovery), ελέγχου συσκευών και μονάδων (Device Manager, Module Manager), υπηρεσιών OpenFlow (OpenFlow Services), αποθήκευσης και χειρισμού μονάδας (Storage, Counter Store, Flow Cache, Packet Streamer, Thread Pool).

Οι εφαρμογές που χρησιμοποιούν τον Floodlight ως υποκείμενο στρώμα αναπτύσσονται ως ανεξάρτητες μονάδες Java και χρησιμοποιούν την προγραμματιστική διεπαφή REST του ελεγκτή (REST API – application programming interface). Μέσω του REST API οι εφαρμογές που αναπτύσσονται επικοινωνούν με τον Floodlight ελεγκτή και μπορούν να χρησιμοποιήσουν το δίκτυο για την όποια λειτουργία τους.

Επίσης οι εφαρμογές μπορούν να αναπτυχθούν και να ρυθμιστούν στο επίπεδο του Floodlight ελεγκτή, φτάνει να αναπτυχθούν και να εφαρμοστούν στον ελεγκτή ως υπομονάδες Java (Java modules). Αυτός ο τρόπος, παρότι είναι πιο απαιτητικός και δύσκολος από ότι η χρήση των REST API's, έχει σημαντικά οφέλη όσο αφορά την ταχύτητα εκτέλεσης της εφαρμογής καθώς και προσφοράς μεγαλύτερου εύρους ζώνης επικοινωνίας με τον Floodlight, μιας και η σύζευξη εφαρμογής – ελεγκτή γίνεται πιο άμεσα με την χρήση των Java API's.

2.2.2 Ο ελεγκτής NOX – NOX controller

Ο NOX ελεγκτής είναι μια πλατφόρμα λογισμικού η οποία προσφέρει την δυνατότητα κατασκευής, διαχείρισης και ελέγχου υπολογιστικών δικτύων με χρήση του πρωτοκόλλου OpenFlow. Ο NOX μάλιστα ήταν ο πρώτος ελεγκτής που αναπτύχθηκε αποκλειστικά για το πρωτόκολλο OpenFlow και συνεπώς είναι από τους πιο διαδεδομένους ελεγκτές για συσκευές OpenFlow.

Ο NOX αρχικά αναπτύχθηκε από την εταιρεία Nicira Networks, ταυτόχρονα με την ανάπτυξη του πρωτοκόλλου OpenFlow . Δόθηκε στην ερευνητική κοινότητα το

2008 και από τότε συνεχίστηκε η εξέλιξη του και υπήρξε η βάση για διάφορα ερευνητικά έργα που αφορούν το SDN (Software defined networking), έννοια η οποία ξεκίνησε από ακαδημαϊκές εργασίες για δημιουργία αρχιτεκτονικών για εταιρικά δίκτυα όπως είναι το SANE (Secure Architecture for the Networked Enterprise) και το Ethane.

Γενικά ο ελεγκτής NOX προσφέρει μια πλήρη προγραμματιστική διεπαφή (API – application programming interface) του OpenFlow 1.0 με χρήση των γλωσσών C++ καθώς και Python, χρήση ασύγχρονων εισόδων – εξόδων (I/O) και είναι προσανατολισμένος για λειτουργία σε συστήματα Linux, συγκεκριμένα για τις εκδόσεις Ubuntu 11.10 και 12.04, καθώς και για Debian.

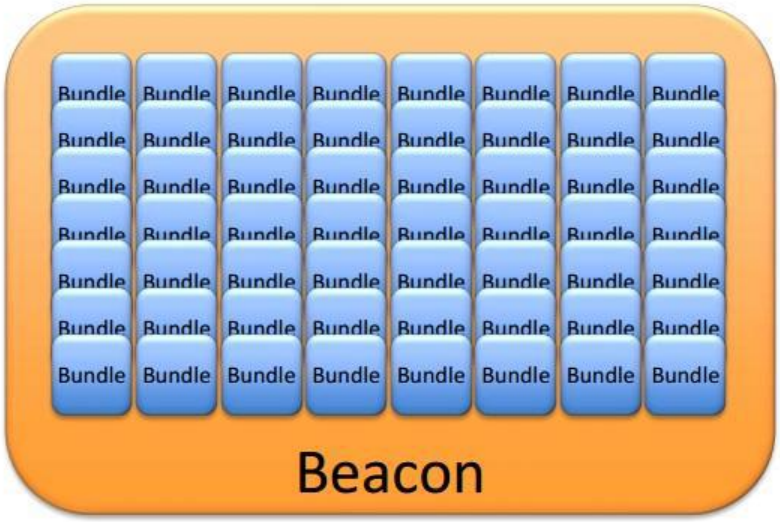
Για τον έλεγχο του δικτύου στο οποίο αναπτύσσεται, ο NOX χρησιμοποιείται ως πλατφόρμα στην οποία αναπτύσσονται και τρέχουν εφαρμογές οι οποίες εξειδικεύονται για να επιτελούν τις επιθυμητές ενέργειες στο δίκτυο από τον χρήστη. Οι εφαρμογές αυτές χρησιμοποιούν το OpenFlow API του NOX και έτοιμα υποπρογράμματα (components) από τον ελεγκτή για άμεση χρήση σε συνήθεις λειτουργίες όπως η δρομολόγηση των πακέτων (routing component) ή η ανίχνευση τοπολογίας του δικτύου (topology component).

2.2.3 Ο ελεγκτής Beacon

Ο Beacon είναι ένας ελεγκτής OpenFlow που αναπτύχθηκε σε περιβάλλον Java και διατέθηκε για χρήση το 2011. Μπορεί να τρέξει σε αρκετές πλατφόρμες (Windows, Linux, Android OS) και υποστηρίζει πολυνηματική (multithreaded) λειτουργία. Βασίζεται σε τμηματική λογική (modular) οπότε είναι εύκολα επεκτάσιμος με χρήση νέων τμημάτων (modules) για υποστήριξη επιπροσθέτων λειτουργιών.

Με την χρήση δεσμών κώδικα (code bundles) ο Beacon μπορεί να εκτελεί με δυναμικό τρόπο εφαρμογές που δεν είναι αλληλεξαρτώμενες ξεκινώντας,

σταματώντας ή θέτοντας σε παύση δέσμες κώδικα σύμφωνα με τις επιλογές του χρήστη, χωρίς να είναι αναγκαία η επανεκκίνηση του ελεγκτή για κάθε αλλαγή. Η αρχιτεκτονική αυτή του Beacon παρουσιάζεται στο σχήμα 2-3.



Σχήμα 2-3 Beacon controller με δέσμες κώδικα (Bundles)

Οι δέσμες κώδικα ανωτέρω μπορούν να δουλεύουν ταυτόχρονα, να διαμοιράζονται τα πακέτα κώδικα και εντολών τους (Java packages) με άλλες δέσμες, να επεκτείνουν το ρεπερτόριο λειτουργιών τους με άλλο κώδικα (extend) ή και να έχουν πολλαπλές εκδόσεις σε λειτουργία ταυτόχρονα (versioning).

Ο Beacon παρέχεται με όλες τις βασικές δέσμες κώδικα, όπως είναι η OpenFlow (OF 1.0 Protocol) για το πρωτόκολλο OpenFlow, οι δέσμες για κωδικοποίηση και αποκωδικοποίηση πακέτων (Ethernet, ARP, IPv4, LLDP, TCP, UDP), δέσμες για βασικές λειτουργίες μεταγωγέα (Core, Learning Switch, Hub, Device Manager) καθώς και δέσμες αλγορίθμων και εύρεσης τοπολογίας (Topology, Layer 2 Shortest Path Routing).

2.3 To OpenFlow Switch

Το OF switch μπορεί να είναι οποιαδήποτε συσκευή προώθησης πακέτων (δρομολογητής – router και μεταγωγέας – switch) που να διαθέτει ένα η

περισσότερους πίνακες ροής (flow tables) , ένα πίνακα ομαδοποίησης (group table) και δυνατότητα χρήσης ενός ξεχωριστού καναλιού για άμεση επικοινωνία με τον OF ελεγκτή. Μέσω αυτού του καναλιού γίνεται η διαχείριση του μεταγωγέα από τον ελεγκτή και μπορούν να προστεθούν, ενημερωθούν ή να αφαιρεθούν καταχωρήσεις ροής (flow entries) από τα flow tables.

Ένας δρομολογητής OpenFlow αποτελείται από έναν ή περισσότερους πίνακες ροής (flow tables) και πίνακες ομάδας (group tables), που εκτελούν λειτουργίες όπως αναζήτηση και προώθηση πακέτων , και από ένα κανάλι OpenFlow που συνδέεται στον ελεγκτή. Ο ελεγκτής διαχειρίζεται τον μεταγωγέα μέσω του πρωτοκόλλου OpenFlow. Χρησιμοποιώντας αυτό το πρωτόκολλο, ο ελεγκτής μπορεί να προσθέσει, να ενημερώσει και να διαγράψει καταχωρήσεις που βρίσκονται στον πίνακα ροών , είτε διαδραστικά (απαντώντας σε αιτήσεις πακέτων) είτε προληπτικά.

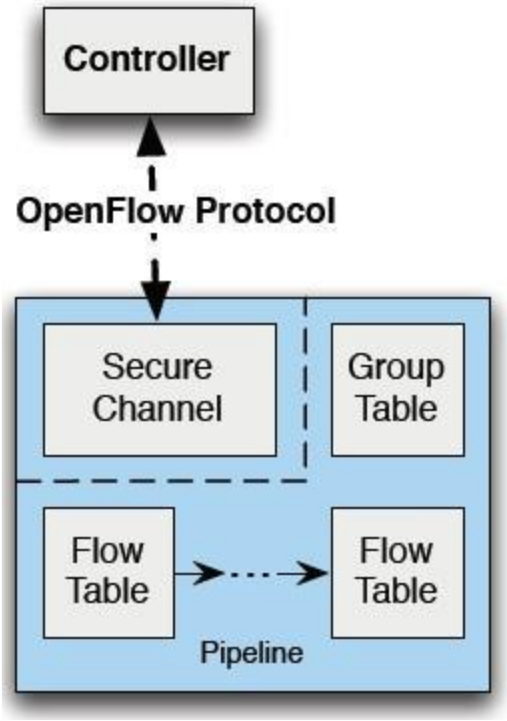
Κάθε πίνακας ροής στον διακόπτη περιέχει ένα σύνολο καταχωρήσεων όπου κάθε μία αποτελείται από τα πεδία match tables,μετρητές, καθώς και μια σειρά από εντολές για την εφαρμογή σε πακέτα αντιστοίχισης .

Η αντιστοίχιση ξεκινάει από τον πρώτο πίνακα ροής και μπορεί να συνεχίσει και στους υπόλοιπους πίνακες ροής. Οι καταχωρήσεις ροής (flow entries) αντιστοιχίζουν τα πακέτα με σειρά προτεραιότητας. Στην περίπτωση που βρεθεί αντιστοίχιση , οι οδηγίες που σχετίζονται με τη συγκεκριμένη καταχώρηση εκτελούνται. Εάν δεν βρεθεί καμία αντιστοιχία στον πίνακα ροής, το αποτέλεσμα εξαρτάται από τη διαμόρφωση του μεταγωγέα: το πακέτο μπορεί να διαβιβάζεται στον ελεγκτή μέσω του καναλιού OpenFlow, να απορριφτεί , ή μπορεί να συνεχίσει στον επόμενο πίνακα ροής.

Οδηγίες που σχετίζονται με κάθε καταχώρηση, περιγράφουν την προώθηση και την τροποποίηση των πακέτων, την επεξεργασία του πίνακα ομάδας (group table) , και την διαδικασία της διοχέτευσης. Οι οδηγίες που αναφέρονται στην διεργασία της διοχέτευσης, επιτρέπουν στα πακέτα να αποστέλλονται σε επόμενους πίνακες για

περαιτέρω επεξεργασία και επιτρέπουν την ανταλλαγή πληροφοριών, με τη μορφή των μεταδεδομένων, μεταξύ των πινάκων.

Καταχωρήσεις ροής μπορούν να διαβιβαστούν σε μια θύρα. Αυτή είναι συνήθως μια φυσική θύρα, αλλά μπορεί επίσης να είναι μια εικονική θύρα που ορίζεται από το μεταγωγέα. Οι εικονικές θύρες μπορούν να εκτελέσουν διαδικασίες προώθησης, όπως η αποστολή στον ελεγκτή, ή υπερχείλιση, ή προώθηση χρησιμοποιώντας μεθόδους χωρίς την χρήση OpenFlow, όπως οι συμβατικές λειτουργίες των δικτυακών συσκευών.



Σχήμα 2-4 Δομή του OpenFlow switch

Πέρα της διαχείρισης κάθε πακέτου ξεχωριστά, ο μεταγωγέας μπορεί να χρησιμοποιήσει το group table για πιο μαζική επεξεργασία της κίνησης. Ένα flow entry μπορεί να αντιστοιχεί σε ένα group table action (ενέργεια που ορίζεται στο group table) για πακέτα που αντιστοιχούν στο εν λόγω flow entry. Τέτοιες ενέργειες που επιτελούνται βάση και με χρήση του group table και των group entries που αυτό

περιλαμβάνει, μπορούν να είναι πιο πολύπλοκες, όπως η πολυδιόδευση πακέτων (multipath forwarding), γρήγορη επαναδρομολόγηση (fast reroute) ή η συσσωμάτωση ζεύξεων (link aggregation). Αναλυτικότερα το group table περιγράφεται στο κεφάλαιο 2.4.2 .

Όπως αναφέρθηκε και προηγουμένως τα flow tables σε ένα switch βρίσκονται σε διασωληνωμένη μορφή (pipelined) και ανάλογα με την μορφή της διασωλήνωσης οι μεταγωγείς διακρίνονται σε δύο είδη:

A. Οι OpenFlow-only μεταγωγείς, οι οποίοι υποστηρίζουν μόνο λειτουργίες OpenFlow και όλα τα πακέτα επεξεργάζονται με την OF διασωλήνωση του μεταγωγέα.

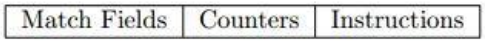
B. Οι OpenFlow-hybrid μεταγωγείς, οι οποίοι υποστηρίζουν και λειτουργία OpenFlow καθώς και λειτουργία Ethernet (κλασσική λειτουργία OSI layer 2 και 3 δρομολόγησης). Αυτοί οι μεταγωγείς περιλαμβάνουν ένα μηχανισμό κατηγοριοποίησης που ανακατευθύνει τα ληφθέντα πακέτα είτε στην διασωλήνωση OF είτε στη κανονική διασωλήνωση Ethernet. Ο μηχανισμός αυτός μπορεί να βασίζεται στην θύρα εισόδου των πακέτων ή πιθανά σε κάποια ετικέτα (tag) στην επικεφαλίδα (header) των πακέτων.

Οι σχεδιαστές των μεταγωγέων μπορούν να σχεδιάζουν την εσωτερική δομή των συσκευών κατά βούληση, με τον περιορισμό ότι για να πληρούν τις προϋποθέσεις για το πρωτόκολλο OpenFlow οι συσκευές πρέπει να διαθέτουν την βασική αρχιτεκτονική που αναλύθηκε προηγουμένως. Φυσικά μπορούν να γίνουν διαφοροποιήσεις που να είναι συμβατές με το OF, για παράδειγμα η χρήση μιας μάσκας bit (bitmask) για προώθηση πακέτων σε πολλαπλές θύρες σε ένα flow entry, αντί να γίνει χρήση του group table.

2.4 Flow Table και Group Table

2.4.1 Flow Table

Η δομή ενός πίνακα ροών (Flow Table) αποτελείται από τα αντίστοιχα Flow Entries όπως φαίνεται και στην κάτωθι εικόνα:



Σχήμα 2-5 Δομή Flow entry

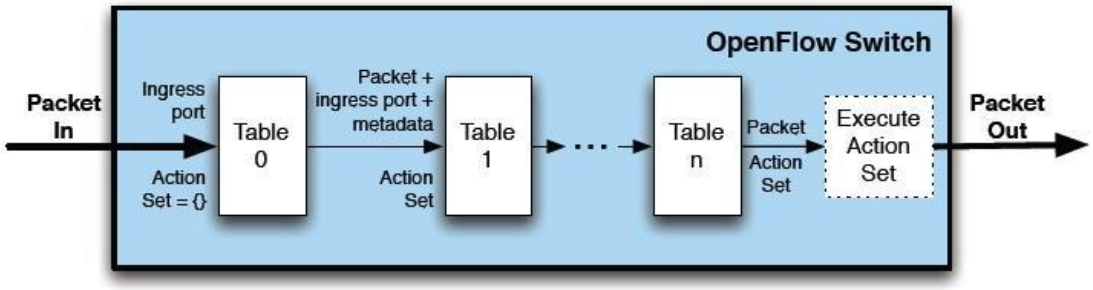
Πολλά τέτοια entries είναι καταχωρημένα σε ένα πίνακα που λέγεται flow table. Βάση αυτών των καταχωρίσεων γίνεται η δρομολόγηση των πακέτων από το πρωτόκολλο OF. Αναλυτικότερα η δομή ενός flow entry είναι:

- match fields: τα πεδία αντιστοίχισης για τα ληφθέντα πακέτα. Αυτά περιέχουν την θύρα εισόδου (ingress port), τις επικεφαλίδες και θύρες πακέτων (headers,ports) και προαιρετικά κάποια μεταδεδομένα (metadata).

- Counters: μετρητές που αφορούν τα ληφθέντα πακέτα, για παράδειγμα πόσα αντιστοιχήθηκαν επιτυχώς ή πόσα απορρίφθηκαν.

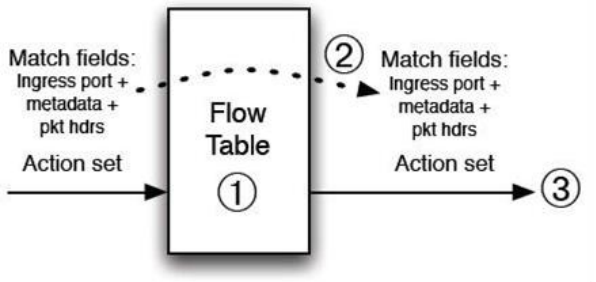
- Instructions: οδηγίες για την επιτέλεση ή τροποποίηση κάποιων ενεργειών που γίνονται κατά την διάρκεια της επεξεργασίας του πακέτου.

Για την διαδοχική σύγκριση στα flow tables το OF ακολουθεί την διαδικασία της διοχέτευσης (pipeline) δημιουργώντας έτσι μια σταθερή ροή πακέτων μέσω των flow tables, όπως μπορεί κανείς να διαπιστώσει και από το ακόλουθο σχήμα:



Σχήμα 2-6 Διαδρομή πακέτου μέσω των flow tables

Όπως γίνεται αντιληπτό και από το σχήμα οι πίνακες ροής ενός OpenFlow δρομολογητή είναι αριθμημένοι ακολουθιακά, ξεκινώντας από το 0. Η επεξεργασία αρχίζει σειριακά. Η διαδικασία της διοχέτευσης ξεκινάει πάντα από τον πρώτο πίνακα ροής. Για να γίνει η αντιστοίχιση κατά την λήψη ενός πακέτου γίνεται σύγκριση των δεδομένων της επικεφαλίδας του πακέτου, της θύρας εισόδου του πακέτου και τυχόν μεταδεδομένων διαδοχικά με το πρώτο flow entry του πρώτου flow table και σταδιακά όλων των επόμενων entries στα υπόλοιπα flow tables. Η αντιστοίχιση γίνεται κατά σειρά προτεραιότητας, δηλαδή η πρώτη επιτυχής αντιστοίχιση flow entry είναι αυτή που ακολουθείται, με τα επόμενα entries στα υπόλοιπα flow tables να χρησιμοποιούνται ανάλογα με τις οδηγίες του matched entry .



①Αντιστοίχιση κατά σειρά προτεραιότητας

②εκτέλεση οποιασδήποτε εντολής:

I. *Με την τροποποίηση του πακέτου ανάλογα με την εντολή που εκτελείται και ενημέρωση των πεδίων αντιστοίχισης*
II. *Ενημέρωση του πεδίου εντολών του πακέτου*
III. *Ενημέρωση μεταδεδομένων πακέτου*

③Αποστολή του αντίστοιχου πακέτου και των σετ εντολών στο επόμενο Flow Table

Σχήμα 2-7 Επεξεργασία πακέτου σε flow table

Όταν επιτύχει η αντιστοίχιση ενός πακέτου με ένα flow entry ενός flow table, εκτελείται το αντίστοιχο σετ εντολών (Instructions set) που περιλαμβάνει το entry. Οι εντολές αυτές μπορεί να ανακατευθύνουν το πακέτο σε άλλο flow table όπου η διαδικασία προώθησης επαναλαμβάνεται (εντολές Goto). Αξιοσημείωτο είναι το γεγονός οτι ένα flow entry μπορεί να κατευθύνει το πακέτο σε flow table με αριθμό

μεγαλύτερο από το τρέχον (στο οποίο βρίσκεται το entry), καθώς η διαδικασία της διοχέτευσης λειτουργεί με προώθηση του πακέτου προς τα εμπρός και όχι προς τα πίσω. Όπως είναι προφανές ότι τα flow entries του τελευταίου πίνακα ροής δεν μπορεί να περιλαμβάνει εντολές που αφορούν την διαδικασία προώθησης (εντολές Goto) .

Όταν το πακέτο φτάσει στο τέλος της διασωλήνωσης και δεν υπάρχει άλλη εντολή προώθησης τότε το πακέτο επεξεργάζεται από τις εντολές του flow entry και προωθείται στην ανάλογη θύρα εξόδου.

Στην περίπτωση που ένα πακέτο δεν αντιστοιχηθεί σε κάποιο flow entry ενός flow table τότε έχουμε αστοχία πίνακα – table miss. Η διαδικασία που ακολουθείται σε τέτοιες περιπτώσεις εξαρτάται από την πολιτική του εκάστοτε δικτύου OpenFlow. Ως προεπιλογή στην περίπτωση αυτή τα αντίστοιχα πακέτα προωθούνται μέσω του καναλιού ελέγχου (control channel) στον ελεγκτή. Βέβαια σε ορισμένες περιπτώσεις ακολουθείται και η διαδικασία απόρριψης του πακέτου (drop).

2.4.2 Group Table

Κάθε καταχώρηση ομάδας (group entry) περιέχει:

Group Identifier	Group Type	Counters	Action Buckets

Σχήμα 2-8 Δομή Group entry

- group identifier: αναγνωριστικό ομάδας, που είναι ένας μη προσημασμένος ακέραιος αριθμός των 32bit και χρησιμεύει για την ταυτοποίηση της ομάδας πακέτων που αφορούν το συγκεκριμένο entry.

- **group type:** ο τύπος της ομάδας, που δείχνει ποιες εντολές από την ομάδα εντολών (action bucket) θα εκτελεστούν για τα πακέτα που ανήκουν στο entry.

- **counters:** μετρητές, που χρησιμοποιούνται για διάφορες πληροφορίες, όπως αριθμός πακέτων που ανήκουν στο group entry.

- **action buckets:** Διατεταγμένη λίστα εντολών στην οποία κάθε εντολή περιέχει ένα σύνολο ενεργειών και τις σχετικές παραμέτρους για να εφαρμοστούν στο προς επεξεργασία πακέτο.

Όσο αφορά τα group types, υπάρχουν τέσσερις κατηγορίες:

- **Κατηγορία all :** Εκτελούνται όλες οι εντολές στις ομάδες εντολών, με το πακέτο να κλωνοποιείται σε αντίγραφα και να επεξεργάζεται από κάθε εντολή στο action bucket. Αυτή η κατηγορία χρησιμοποιείται συνήθως για ευρυεκπομπή (broadcasting) και πολυεκπομπή (multicasting).

- **Κατηγορία select:** Εκτελείται μόνο μια εντολή από το σύνολο της ομάδας εντολών, βάση των παραμέτρων του πακέτου και των αλγορίθμων επιλογής της εκάστοτε συσκευής.

- **Κατηγορία indirect:** Εκτελείται μια μόνο προκαθορισμένη εντολή στο group table, πράγμα που επιτρέπει πολλαπλές ροές πακέτων να δείχνουν σε ένα αναγνωριστικό ομάδας (group identifier) και να υποστηρίζεται γρηγορότερη και αποτελεσματικότερη προώθηση ομάδων πακέτων σε συγκεκριμένο προορισμό.

- Κατηγορία fast failover: Εκτελείται η πρώτη εν ενεργεία ομάδα εντολών, δηλαδή το σύνολο εντολών που είναι συσχετισμένο με μια ενεργή θύρα εξόδου. Με αυτή την μέθοδο κάθε δέσμη ενεργειών στο action bucket συσχετίζεται με μια θύρα εξόδου για να μπορεί η συσκευή δρομολόγησης σε περίπτωση απώλειας σύνδεσης από μια θύρα εξόδου να επαναδρομολογήσει πακέτα σε εφεδρικές θύρες χωρίς να χρειαστεί ξανά επικοινωνία με τον controller.

2.4.3 Μετρητές - Counters

Η OpenFlow τεχνολογία υποστηρίζει την δημιουργία μετρητών που μπορούν να τοποθετηθούν σε κάθε πίνακα, ροή ,θύρα, ουρά, group και σε στοίβες. Στον παρακάτω πίνακα συγκεντρώνονται όλα τα δυνατά σετ των μετρητών που υποστηρίζονται από τις προδιαγραφές του OpenFlow.

Counter	Bits
Per Table	
Reference count (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
Per Group	
Reference Count (flow entries)	32
Packet Count	64
Byte Count	64
Per Bucket	
Packet Count	64
Byte Count	64

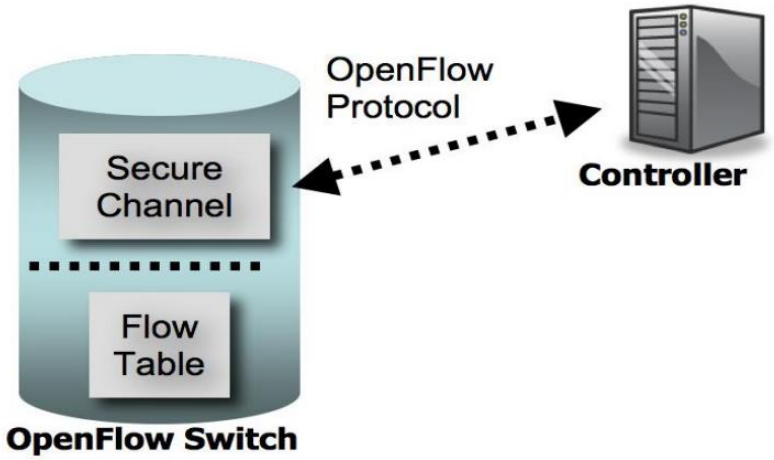
Σχήμα 2-9 Δομή Counter

Ο όρος Duration αναφέρεται στο συνολικό χρόνο που η ροή είναι εγκατεστημένη στον μεταγωγέα. Επιπλέον το πεδίο Receive Errors αναφέρεται στο σύνολο όλων των σφαλμάτων λήψης και συγκρούσεων.

Στην περίπτωση που η τιμή του μετρητή δεν είναι στις διαθέσιμες τιμές που παρέχονται από το σύστημα, η τιμή του ορίζεται στο -1 .

2.5 *Ασφαλές Κανάλι (Secure Chanel)*

Το ασφαλές κανάλι (secure channel) είναι η διεπαφή που συνδέει κάθε μεταγωγέα OpenFlow σε έναν ελεγκτή. Μέσω της διεπαφής αυτής ο ελεγκτής ανταλλάσσει μηνύματα με τις δικτυακές συσκευές προκειμένου να τις ρυθμίσει και να τις διαχειριστεί.



Σχήμα 2-10 Εξιδανικευμένος OpenFlow μεταγωγέας. Το Flow Table ελέγχεται από έναν απομακρυσμένο ελεγκτή μέσω του ασφαλούς καναλιού

Το OpenFlow παρέχει ένα πρωτόκολλο για την επικοινωνία μεταξύ της διαδικασίας του ελεγκτή και τους OpenFlow-μεταγωγείς. Υπάρχουν τρεις τύποι μηνυμάτων που υποστηρίζονται από το πρωτόκολλο OpenFlow. Τα μηνύματα τύπου controller-to-switch,asynchronous και symmetric. Θα περιγράψουμε εν συντομία αυτά τα τρία είδη μηνυμάτων.

Τα μηνύματα τύπου Controller-to-switch αποστέλλονται από τον ελεγκτή και δεν απαιτούν πάντα απάντηση από τον δρομολογητή. Μέσα από αυτά τα μηνύματα ο ελεγκτής ρυθμίζει τις δικτυακές συσκευές, διαχειρίζεται τον πίνακα ροής (flow table)

και αποκτά πληροφορίες για την κατάσταση του πίνακα ροής και τις δυνατότητες που υποστηρίζονται από τον δρομολογητή κάθε δεδομένη στιγμή.

Τα **ασύγχρονα** μηνύματα αποστέλλονται χωρίς αίτηση από τον μεταγωγέα στον ελεγκτή και δηλώνουν μια αλλαγή στο μεταγωγέα ή στην κατάσταση του δικτύου. Ένα από τα πιο σημαντικά γεγονότα είναι το packet-in, το οποίο συμβαίνει όταν ένα πακέτο που δεν έχει αντιστοίχιση με κάποια ροή, φτάνει στον δρομολογητή. Όταν συμβαίνει αυτό, ένα μήνυμα packet-in στέλνεται στον ελεγκτή, που περιέχει το πακέτο ή ένα μέρος του πακέτου, προκειμένου για τον ελεγκτή να εξετάσει και να προσδιορίσει ποιο είδος ροής θα πρέπει να συσταθεί για αυτό. Άλλα γεγονότα περιλαμβάνουν λήξη καταχωρήσεων ροής, αλλαγή κατάστασης της θύρας ή άλλα συμβάντα σφάλματος.

Τέλος, η τρίτη κατηγορία των μηνυμάτων OpenFlow είναι τα **συμμετρικά**(symmetriconous) μηνύματα που αποστέλλονται χωρίς αίτηση προς τις δύο κατευθύνσεις. Αυτοί μπορούν να χρησιμοποιηθούν για να βοηθήσουν ή να διαγνώσουν προβλήματα στην σύνδεση του ελεγκτή-διακόπτη.

2.5.1 Μηνύματα τύπου Controller-to-Switch

Τα μηνύματα τύπου Controller-to-Switch εκκινούν από τον ελεγκτή και απαιτούν ή όχι απάντηση από τον δρομολογητή. Παρακάτω περιγράφονται οι διαφορετικές λειτουργίες που μπορούν να εκτελεστούν με χρήση μηνυμάτων τύπου Controller-to-Switch.

Features: Ο ελεγκτής μπορεί να κάνει αίτημα στον τον δρομολογητή σχετικά με τις δυνατότητες του. Η αίτηση αυτή γίνεται στέλνοντας features request.Ο δρομολογητής στην συνέχεια στέλνει μια απάντηση τύπου features reply που περιλαμβάνει όλες τις δυνατότητες της συσκευής. Η λειτουργία αυτή πραγματοποιείται συνήθως και την στιγμή που γίνεται η εγκατάσταση του OpenFlow καναλιού.

Configurations: Ο ελεγκτής μπορεί να κάνει αιτήματα που αφορούν τις παραμέτρους ρύθμισης του δρομολογητή και μπορεί να τις ρυθμίσει. Ο δρομολογητής ανταποκρίνεται μόνο στα αιτήματα του ελεγκτή

Modify-State: Μηνύματα τύπου Modify-State αποστέλλονται από τον ελεγκτή στους δρομολογητές με σκοπό την διαχείριση της κατάστασης τους. Ο πρωταρχικός τους στόχος είναι να προσθέσουν/αφαιρέσουν και να τροποποιήσουν flows/groups στους OpenFlow πίνακες αλλά και να ρυθμίσουν ιδιότητες των θυρών των δικτυακών συσκευών.

Read-State: Μηνύματα τύπου Read-State, χρησιμοποιούνται από τον ελεγκτή για τη συλλογή στατιστικών στοιχείων από τον δρομολογητή. Τα μηνύματα αυτού του τύπου θα αναλυθούν περαιτέρω για λόγους που αναλύονται στην συνέχεια.

Packet-out: Χρησιμοποιούνται από τον ελεγκτή για την αποστολή πακέτων από μία συγκεκριμένη θύρα του μεταγωγέα αλλά και για να προωθεί τα πακέτα που λαμβάνονται μέσω packet-in μηνυμάτων. Μηνύματα packet-out πρέπει να περιλαμβάνουν ένα πλήρες πακέτο ή έστω κάποια ειδική ταμπέλα (ID) που αναφέρεται στο πακέτο που αποθηκεύεται στον μεταγωγέα. Το μήνυμα πρέπει επίσης να περιέχει μια λίστα ενεργειών που πρέπει να εκτελούνται με την σειρά που έχει καθοριστεί. Μια κενή λίστα θα έχει ως αποτέλεσμα την απόρριψη του πακέτου.

Barrier: Μηνύματα τύπου Barrier χρησιμοποιούνται από τον ελεγκτή κυρίως για λειτουργίες ελέγχου, όπως το να λαμβάνουν ενημερώσεις για πράξεις που έχουν ολοκληρωθεί.

2.6 Read State Μηνύματα

Τα μηνύματα τύπου Read-State, χρησιμοποιούνται από τον ελεγκτή για τη συλλογή στατιστικών στοιχείων από τον δρομολογητή. Όπως θα αναλυθεί στο πέμπτο κεφάλαιο για την εύρεση του επιθυμητού μονοπατιού στον αλγόριθμο δρομολόγησης

που δημιουργήθηκε, απαραίτητο ήταν η συλλογή στατιστικών στοιχείων σχετικά με διάφορα στοιχεία του δικτύου και η αποθήκευση των στοιχείων αυτών σε κάποια βάση δεδομένων που θα ενημερωνόταν συχνά.

Το πρωτόκολλο OpenFlow παρέχει την δυνατότητα συλλογής τέτοιων στοιχείων μέσω μηνυμάτων Read-State.Πιο συγκεκριμένα ανά τακτά χρονικά διαστήματα ο ελεγκτής αποστέλλει στις δικτυακές συσκευές μηνύματα OFPT_STATS_REQUEST.

```
struct ofp_stats_request {
        struct ofp_header header;
        uint16_t type;                  /* One of the OFPST_* constants. */
        uint16_t flags;                 /* OFPSF_REQ_* flags (none yet defined). */
        uint8_t pad[4];
        uint8_t body[0];                        /* Body of the request. */
};
OFP_ASSERT(sizeof(struct ofp_stats_request) == 16);
```

Οι δρομολογητές απαντούν με ένα ή και περισσότερα μηνύματα τύπου OFPT_STATS_REPLY.

```
struct ofp_stats_reply {
        struct ofp_header header;
        uint16_t type;            /* One of the OFPST_* constants. */
        uint16_t flags;          /* OFPSF_REPLY_* flags. */
        uint8_t pad[4];
        uint8_t body[0];                /* Body of the reply. */
};
OFP_ASSERT(sizeof(struct ofp_stats_reply) == 16);
```

Η μόνη τιμή που ορίζεται για σημαία (flag) σε μία απάντηση μηνύματος είναι η τιμή 0x0001, και ορίζεται μόνο στην περίπτωση που θα ακολουθήσουν περισσότερες από μία απαντήσεις. Για την διευκόλυνση εφαρμογής, οι δρομολογητές επιτρέπεται να στείλουν απαντήσεις (replies) χωρίς επιπλέον εγγραφές. Ωστόσο, πρέπει πάντα να αποστέλλει μετά ένα μήνυμα που θα περιέχει τιμές του πεδίου flag. Οι ταυτότητες των συναλλαγών (xid) των απαντήσεων πρέπει να ταιριάζει πάντα με την αίτηση που τους έχει ζητηθεί.

Αμφότερα και στις αιτήσεις και στις απαντήσεις, ο τύπος πεδίου προσδιορίζει το είδος πληροφοριών που δόθηκαν και καθορίζουν τον τρόπο που ερμηνεύεται το πεδίο:

```
enum ofp_stats_types {
        /* Description of this OpenFlow switch.
        * The request body is empty.
        * The reply body is struct ofp_desc_stats. */
        OFPST_DESC,

        /* Individual flow statistics.
        * The request body is struct ofp_flow_stats_request.
        * The reply body is an array of struct ofp_flow_stats. */
        OFPST_FLOW,

        /* Aggregate flow statistics.
        * The request body is struct ofp_aggregate_stats_request.
        * The reply body is struct ofp_aggregate_stats_reply. */
        OFPST_AGGREGATE,

        /* Flow table statistics.
        * The request body is empty.
        * The reply body is an array of struct ofp_table_stats. */
        OFPST_TABLE,

        /* Port statistics.
        * The request body is struct ofp_port_stats_request.
        * The reply body is an array of struct ofp_port_stats. */
        OFPST_PORT,

        /* Queue statistics for a port
        * The request body defines the port
        * The reply body is an array of struct ofp_queue_stats */
        OFPST_QUEUE,

        /* Group counter statistics.
        * The request body is empty.
        * The reply is struct ofp_group_stats. */
        OFPST_GROUP,

        /* Group description statistics.
        * The request body is empty.
        * The reply body is struct ofp_group_desc_stats. */
        OFPST_GROUP_DESC,

        /* Experimenter extension.
        * The request and reply bodies begin with a 32-bit experimenter ID,
        * which takes the same form as in "struct ofp_experimenter_header".
        * The request and reply bodies are otherwise experimenter-defined. */
        OFPST_EXPERIMENTER = 0xffff
};
```

Σε όλους τους τύπους των στατιστικών , εάν κάποια τιμή του μετρητή (counter) δεν είναι διαθέσιμη στον δρομολογητή, η τιμή του τίθεται στο -1.

2.6.1 Περιγραφή στατιστικών

Πληροφορίες σχετικά με τον κατασκευαστή του δρομολογητή, την έκδοση του υλικού (hardware),την έκδοση του λογισμικού, τον σειριακό αριθμό, και μια περιγραφή γενικότερα είναι διαθέσιμη από το τύπο στατιστικών OFPST_DESC:

```
/* Body of reply to OFPST_DESC request. Each entry is a NULL-terminated
* ASCII string. */
struct ofp_desc_stats {
        char mfr_desc[DESC_STR_LEN]; /* Manufacturer description. */
        char hw_desc[DESC_STR_LEN]; /* Hardware description. */
        char sw_desc[DESC_STR_LEN]; /* Software description. */
        char serial_num[SERIAL_NUM_LEN]; /* Serial number. */
        char dp_desc[DESC_STR_LEN]; /* Human readable description of datapath. */
};

OFP_ASSERT(sizeof(struct ofp_desc_stats) == 1056);
```

Όλες οι εγγραφές έχουν διαμορφωθεί με κωδικοποίηση τύπου ASCII και παραγεμισμένες από τα δεξιά με null bytes.(/0). Το πεδίο DESC_STR_LEN έχει μέγεθος 256 Bytes και το SERIAL_NUM_LEN 32 bytes.

2.6.2 επιμέρους ροές στατιστικού τύπου

Πληροφορίες για τις επιμέρους ροές ζητείται με τα στατιστικά τύπου OFPST_FLOW :

```
/* Body for ofp_stats_request of type OFPST_FLOW. */
struct ofp_flow_stats_request {
        uint8_t table_id;               /* ID of table to read (from ofp_table_stats),
                                        0xff for all tables. */
        uint8_t pad[3];          /* Align to 64 bits. */
        uint32_t out_port;              /* Require matching entries to include this
                                        as an output port. A value of OFPP_ANY
                                        indicates no restriction. */
        uint32_t out_group;             /* Require matching entries to include this
                                        as an output group. A value of OFPG_ANY
                                        indicates no restriction. */
        uint8_t pad2[4];                /* Align to 64 bits. */
        uint64_t cookie;                /* Require matching entries to contain this
                                        cookie value */
        uint64_t cookie_mask;    /* Mask used to restrict the cookie bits that
                                        must match. A value of 0 indicates
                                        no restriction. */
        struct ofp_match match;  /* Fields to match. */
};
OFP_ASSERT(sizeof(struct ofp_flow_stats_request) == 120);
```

Το Match field περιέχει μια περιγραφή των ροών που θα πρέπει να συνδυάζεται και επίσης μπορεί να περιέχει wildcard (χαρακτήρας αναπλήρωσης).

Το πεδίο του Table_id δείχνει το δείκτη ενός μεμονωμένου πίνακα που μπορεί να διαβαστεί ή το 0xff για όλους τους πίνακες.

Τα πεδία out_port και out_group προαιρετικά φιλτράρονται από την θύρα εξόδου και την ομάδα(group).Εάν ούτε το out_port,ούτε το out_group περιέχει μια τιμή

διαφορετική από αυτήν του OFPP_ANY και του OFPG_ANY αντίστοιχα, εισάγει έναν περιορισμό κατά την διαδικασία της αντιστοιχίας. Αυτός ο περιορισμός είναι ότι ο κανόνας πρέπει να περιέχει μια ενέργεια εξόδου κατευθυνόμενη στην εν λόγω θύρα ή ομάδα. Αξίζει να σημειωθεί ότι για την απενεργοποίηση του φιλτραρίσματος εξόδου, τόσο το πεδίο out_port όσο και το out_group πρέπει να ρυθμιστούν σε OFPP_ANY και OFPG_ANY αντίστοιχα.

Ο κορμός της απάντησης αποτελείται από μία σειρά από τα ακόλουθα:

```
/* Body of reply to OFPST_FLOW request. */
struct ofp_flow_stats {
        uint16_t length;        /* Length of this entry. */
        uint8_t table_id;                /* ID of table flow came from. */
        uint8_t pad;
        uint32_t duration_sec;  /* Time flow has been alive in seconds. */
        uint32_t duration_nsec;  /* Time flow has been alive in nanoseconds beyond
                                    duration_sec. */
        uint16_t priority;              /* Priority of the entry. Only meaningful
                                    when this is not an exact-match entry. */
        uint16_t idle_timeout;           /* Number of seconds idle before expiration. */
        uint16_t hard_timeout;    /* Number of seconds before expiration. */
        uint8_t pad2[6];                /* Align to 64-bits. */
        uint64_t cookie;                /* Opaque controller-issued identifier. */
        uint64_t packet_count;   /* Number of packets in flow. */
        uint64_t byte_count;            /* Number of bytes in flow. */
        struct ofp_match match;  /* Description of fields. */
        struct ofp_instruction instructions[0];  /* Instruction set. */
};
OFP_ASSERT(sizeof(struct ofp_flow_stats) == 136);
```

Τα πεδία αποτελούνται από εκείνα που προβλέπονται στο flow_mod που τα δημιούργησε, καθώς και από τον πίνακα στον οποίο έγινε η εγγραφή, τον αριθμός των πακέτων, και τον αριθμό των byte.

Τα πεδία duration_sec και duration_nsec, δείχνουν το παρελθόντα χρόνο που χρειάστηκε η ροή για να εγκατασταθεί στον μεταγωγέα. Η συνολική διάρκεια σε νανοδευτερόλεπτα μπορεί να υπολογιστεί ως duration_sec * 109 + duration_nsec. Οι υλοποιήσεις υποχρεούνται να παρέχουν ακρίβεια χιλιοστού του δευτερολέπτου. Μεγαλύτερη ακρίβεια ενθαρρύνεται όπου αυτή είναι διαθέσιμη

2.6.3 Aggregate Flow Statistics(Συγκεντρωτικά στατιστικά ροών)

Συγκεντρωτικές πληροφορίες για πολλαπλές ροές ζητούνται με τα στατιστικά τύπου OFPST_AGGREGATE:

```
/* Body for ofp_stats_request of type OFPST_AGGREGATE. */
struct ofp_aggregate_stats_request {
        uint8_t table_id;                /* ID of table to read (from ofp_table_stats)
0xff for all tables. */
        uint8_t pad[3];          /* Align to 64 bits. */
        uint32_t out_port;               /* Require matching entries to include this
                                     as an output port. A value of OFPP_ANY
                                     indicates no restriction. */
        uint32_t out_group;              /* Require matching entries to include this
                                    as an output group. A value of OFPG_ANY
                                    indicates no restriction. */
        uint8_t pad2[4];                 /* Align to 64 bits. */
        uint64_t cookie;                 /* Require matching entries to contain this
                                     cookie value */
        uint64_t cookie_mask;    /* Mask used to restrict the cookie bits that
                                     must match. A value of 0 indicates
                                     no restriction. */
        struct ofp_match match;  /* Fields to match. */
};
OFP_ASSERT(sizeof(struct ofp_aggregate_stats_request) == 120);
```

Τα πεδία σε αυτό το μήνυμα έχουν τις ίδιες σημασίες όπως στα επιμέρους στατιστικά ροής ζητούμενου τύπου (OFPST_FLOW).

Ο κορμός της απάντησης αποτελείται από τα ακόλουθα:

```
/* Body of reply to OFPST_AGGREGATE request. */
struct ofp_aggregate_stats_reply {
        uint64_t packet_count;          /* Number of packets in flows. */
        uint64_t byte_count;            /* Number of bytes in flows. */
        uint32_t flow_count;            /* Number of flows. */
        uint8_t pad[4];                 /* Align to 64 bits. */
};
OFP_ASSERT(sizeof(struct ofp_aggregate_stats_reply) == 24);
```

2.6.4 Table Statistics(Στατιστικά πίνακα)

Οι πληροφορίες σχετικά με τους πίνακες ζητούνται από τα στατιστικά ζητούμενου τύπου OFPST_TABLE.Η αίτηση δεν περιέχει κανένα δεδομένο του κορμού.

Ο κορμός της απάντησης αποτελείται από μια σειρά από τα ακόλουθα:

```
/* Body of reply to OFPST_TABLE request. */
struct ofp_table_stats {
        uint8_t table_id;               /* Identifier of table. Lower numbered tables
```

```
                                    are consulted first. */
        uint8_t pad[7];           /* Align to 64-bits. */
char name[OFP_MAX_TABLE_NAME_LEN];
        uint32_t wildcards;              /* Bitmap of OFPFMF_* wildcards that are
                                    supported by the table. */
        uint32_t match;          /* Bitmap of OFPFMF_* that indicate the fields
                                    the table can match on. */
        uint32_t instructions;           /* Bitmap of OFPIT_* values supported. */
        uint32_t write_actions;  /* Bitmap of OFPAT_* that are supported
                                    by the table with OFPIT_WRITE_ACTIONS. */
        uint32_t apply_actions;  /* Bitmap of OFPAT_* that are supported
                                    by the table with OFPIT_APPLY_ACTIONS. */
        uint32_t config;                 /* Bitmap of OFPTC_* values */
        uint32_t max_entries;            /* Max number of entries supported. */
        uint32_t active_count;   /* Number of active entries. */
        uint64_t lookup_count;   /* Number of packets looked up in table. */
        uint64_t matched_count;  /* Number of packets that hit table. */
};
OFP_ASSERT(sizeof(struct ofp_table_stats) == 88);
```

Ο κορμός περιέχει ένα πεδίο wildcard,το οποίο δείχνει τους τομείς για τους οποίους ο συγκεκριμένος πίνακας υποστηρίζει την ύπαρξη των wildcards. Για παράδειγμα, μια άμεση αναζήτηση πίνακα κατακερματισμού θα έχει αυτό το σύνολο πεδίων στο μηδέν, ενώ διαδοχικά η αναζήτηση πίνακα θα πρέπει να οριστεί σε OFPFW_ALL. Οι είσοδοι επιστρέφονται με τη σειρά που τα πακέτα διασχίζουν τους πίνακες.

Το πεδίο write_actions είναι μια δυαδική μάσκα(bitmask) των δράσεων που υποστηρίζονται από τον πίνακα χρησιμοποιώντας την εντολή OFPIT_WRITE_ACTIONS, ενώ το πεδίο apply_actions αναφέρεται στην εντολή OFPIT_APPLY_ACTIONS. Πειραματιζόμενες δράσεις δεν θα πρέπει να αναφερθούν μέσω αυτής της δυαδικής μάσκας(bitmask).Αυτή η μάσκα χρησιμοποιεί τις τιμές από ofp_action_type ως τον αριθμό των δυαδικών ψηφίων που θα μετατοπίσει αριστερά σε μια αντίστοιχη ενέργεια. Για παράδειγμα το OFPAT_SET_DL_VLAN θα χρησιμοποιούσε την σημαία 0x00000002.

Το μήκος του πεδίου OFP_MAX_TABLE_NAME_LEN είναι 32 Bytes.

2.6.5 Port statistics (στατιστικά θύρας)

Οι πληροφορίες σχετικά με τις θύρες ζητούνται από τα ζητούμενα στατιστικά τύπου OFPST_PORT:

```
/* Body for ofp_stats_request of type OFPST_PORT. */
struct ofp_port_stats_request {
        uint32_t port_no;               /* OFPST_PORT message must request statistics
                                 * either for a single port (specified in
                                 * port_no) or for all ports (if port_no ==
                                 * OFPP_ANY). */
        uint8_t pad[4];
};
OFP_ASSERT(sizeof(struct ofp_port_stats_request) == 8);
```

Το πεδίο port_no προαιρετικά φιλτράρει τα ζητούμενα στατιστικά στη δοσμένη θύρα. Για να ζητηθούν όλα τα στατιστικά θύρας, το port_no θα πρέπει να ρυθμιστεί σε OFF_ANY.

Ο κορμός της απάντησης αποτελείται από μια σειρά από τα ακόλουθα:

```
/* Body of reply to OFPST_PORT request. If a counter is unsupported, set
* the field to all ones. */
struct ofp_port_stats {
        uint32_t port_no;
        uint8_t pad[4];           /* Align to 64-bits. */
        uint64_t rx_packets;             /* Number of received packets. */
        uint64_t tx_packets;             /* Number of transmitted packets. */
        uint64_t rx_bytes;               /* Number of received bytes. */
        uint64_t tx_bytes;               /* Number of transmitted bytes. */
        uint64_t rx_dropped;              /* Number of packets dropped by RX. */
        uint64_t tx_dropped;             /* Number of packets dropped by TX. */
        uint64_t rx_errors;               /* Number of receive errors. This is a super-set
                                    of more specific receive errors and should be
                                    greater than or equal to the sum of all
                                    rx_*_err values. */
        uint64_t tx_errors;              /* Number of transmit errors. This is a super-set
                                   of more specific transmit errors and should be
                                   greater than or equal to the sum of all
                                   tx_*_err values (none currently defined.) */
        uint64_t rx_frame_err;    /* Number of frame alignment errors. */
        uint64_t rx_over_err;             /* Number of packets with RX overrun. */
        uint64_t rx_crc_err;             /* Number of CRC errors. */
        uint64_t collisions;              /* Number of collisions. */
};
OFP_ASSERT(sizeof(struct ofp_port_stats) == 104);
```

2.6.6 Σειρά αναμονής στατιστικών(Queue Statistics)

Το μήνυμα αίτησης στατιστικών OFPST_QUEUE παρέχει στατιστικά για μία ή περισσότερες θύρες αλλά και μία ή περισσότερες σειρές αναμονής. Ο ζητούμενος κορμός περιέχει ένα πεδίο port_no που αναγνωρίζει την θύρα του OpenFlow για την οποία ζητούνται στατιστικά, ή το OFPP_ANY που αναφέρεται σ όλες τις θύρες. Το

πεδίο queue_id αναγνωρίζει μια από τις πρωταρχικές σειρές αναμονής, ή το OFPQ_ALL που αναφέρεται σε όλες τις σειρές αναμονής που έχουν διαμορφωθεί για μια συγκεκριμένη θύρα.

```
struct ofp_queue_stats_request {
        uint32_t port_no;       /* All ports if OFPP_ANY. */
        uint32_t queue_id;      /* All queues if OFPQ_ALL. */
};
OFP_ASSERT(sizeof(struct ofp_queue_stats_request) == 8);
```

Ο κορμός της απάντησης αποτελείται από έναν πίνακα με την ακόλουθη δομή:

```
struct ofp_queue_stats {
        uint32_t port_no;
        uint32_t queue_id;      /* Queue i.d */
        uint64_t tx_bytes;      /* Number of transmitted bytes. */
        uint64_t tx_packets;    /* Number of transmitted packets. */
        uint64_t tx_errors;     /* Number of packets dropped due to overrun. */
};
OFP_ASSERT(sizeof(struct ofp_queue_stats) == 32);
```

2.6.7 Στατιστικά ομάδας(group statistics)

Το μήνυμα αίτησης στατιστικών OFPST_QUEUE παρέχει στατιστικά για μία ή περισσότερες ομάδες. Ο ζητούμενος κορμός αποτελείται από ένα πεδίο group_id,που μπορεί να ρυθμιστεί σε OFPG_ALL ώστε να αναφέρεται σε όλες τις ομάδες στον μεταγωγέα.

```
/* Body of OFPST_GROUP request. */
struct ofp_group_stats_request {
        uint32_t group_id;      /* All groups if OFPG_ALL. */
        uint8_t pad[4];  /* Align to 64 bits. */
};
        OFP_ASSERT(sizeof(struct ofp_group_stats_request) == 8);
```

Ο κορμός της απάντησης αποτελείται από έναν πίνακα με την ακόλουθη δομή:

```
/* Body of reply to OFPST_GROUP request. */
struct ofp_group_stats {
        uint16_t length;                /* Length of this entry. */
        uint8_t pad[2];         /* Align to 64 bits. */
        uint32_t group_id;              /* Group identifier. */
        uint32_t ref_count;             /* Number of flows or groups that directly forward
        to this group. */
        uint8_t pad2[4];                /* Align to 64 bits. */
        uint64_t packet_count;  /* Number of packets processed by group. */
        uint64_t byte_count;            /* Number of bytes processed by group. */
        struct ofp_bucket_counter bucket_stats[0];
};
OFP_ASSERT(sizeof(struct ofp_group_stats) == 32)
```

Το πεδίο bucket_stats _field αποτελείται από έναν πίνακα από δομές τύπου ofp_bucket_counter

```
/* Used in group stats replies. */
struct ofp_bucket_counter {
        uint64_t packet_count;   /* Number of packets processed by bucket. */
        uint64_t byte_count;              /* Number of bytes processed by bucket. */
};
        OFP_ASSERT(sizeof(struct ofp_bucket_counter) == 16);
```

2.6.8 Ομάδα περιγραφής στατιστικών(Group Description Statistics)

Το μήνυμα αίτησης στατιστικών OFPST_GROUP_DESC παρέχει έναν τρόπο για την καταγραφή των ρυθμίσεων των ομάδων σε έναν μεταγωγέα, μαζί με τις αντίστοιχες ενέργειες του κάδου (bucket) . Ο κορμός του μηνύματος αίτησης είναι άδειος, ενώ ο κορμός απάντησης είναι μια σειρά από την ακόλουθη δομή:

```
/* Body of reply to OFPST_GROUP_DESC request. */
struct ofp_group_desc_stats {
        uint16_t length;        /* Length of this entry. */
        uint8_t type;    /* One of OFPGT_*. */
        uint8_t pad;     /* Pad to 64 bits. */
        uint32_t group_id;       /* Group identifier. */
struct ofp_bucket buckets[0];
};
OFP_ASSERT(sizeof(struct ofp_group_desc_stats) == 8);
```

Πεδία για την ομάδα περιγραφής στατιστικών είναι τα ίδια όπως αυτά που χρησιμοποιούνται με το ofp_group_mod struct.

2.6.9 Πειραματιζόμενα στατιστικά(Experimenter Statistics)

Τα στατιστικά μηνύματα των ειδικών-πειραματιστών ζητούνται με τον στατιστικό τύπο OFPST_EXPERIMENTER.Τα πρώτα τέσσερα byte του μηνύματος είναι τα αναγνωριστικά των πειραματιστών. Το υπόλοιπο του κορμού είναι πειραματικά καθορισμένο.

Το πειραματικό πεδίο είναι ένα 32-bit ,αξία που μοναδικά αναγνωρίζει τον πειραματιστή. Εάν το πιο σημαντικό byte είναι το μηδέν, τα επόμενα τρία bytes θα είναι του πειραματιστή IEEE OUI. Εάν ο πειραματιστής δεν έχει(ή δεν επιθυμεί να χρησιμοποιήσει)τον OUI τους, θα πρέπει να επικοινωνήσουν με την κοινοπραξία του OpenFlow για να εξασφαλίσουν ένα.

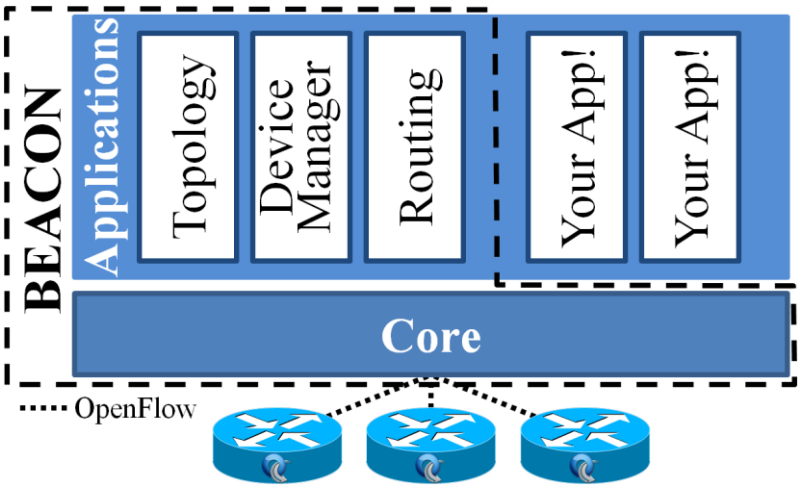
3 *Beacon Controller*

3.1 Ο ελεγκτής Beacon

Ο Beacon αποτελεί έναν από τους πιο διαδεδομένους ελεγκτές ανοιχτού κώδικα τύπου χρησιμοποιώντας την τεχνολογία OpenFlow.Δημιουργήθηκε στο πανεπιστήμιο του Stanford το 2010 και αναπτύχθηκε σε περιβάλλον Java.Χρησιμοποιείται ευρέως για διδακτικούς και για ερευνητικούς σκοπούς και βασίζεται στις αρχές των "Δικτύων Καθοριζόμενα από Λογισμικό" (Software Defined Networking (SDN).

Η πρώτη πλατφόρμα ελέγχου ανοιχτού κώδικα, για τις πρώτες εφαρμογές της τεχνολογίας ήταν ο NOX.Ο NOX έδωσε την δυνατότητα στους ερευνητές να αναπτύξουν δικτυακές εφαρμογές με χρήση προγραμματιστικών τεχνικών, χρησιμοποιώντας Python , αλλά και την προγραμματιστική γλώσσα C++ . Με την πάροδο του χρόνου δημιουργήθηκε η ανάγκη ανάπτυξης μιας πλατφόρμας που θα μπορούσε να καλύψει περισσότερες ανάγκες σε εφαρμογές αλλά και να παρέχει υψηλότερες αποδόσεις.

Η γλώσσα προγραμματισμού και το περιβάλλον ανάπτυξης της έχουν σημαντικό αντίκτυπο στην παραγωγικότητα των προγραμματιστών, και μπορεί επίσης να είναι ένας περιοριστικός παράγοντας στην απόδοση των εφαρμογών. Υπάρχουν πολλές γλώσσες φιλικές προς τον προγραμματιστή, αλλά η απόδοσή τους όταν χρησιμοποιούνται σε έναν ελεγκτή OpenFlow ήταν άγνωστη.



Σχήμα 3-1 Ελεγκτής Beacon

3.2 Αποδοτικότητα Προγραμματισμού

Στην ενότητα αυτή περιγράφονται οι επιλογές σχεδιασμού του ελεγκτή Beacon που προορίζονται για τη βελτίωση της παραγωγικότητας του έργου. Οι σχεδιαστικές επιλογές που παρουσιάζονται στους ακόλουθους τομείς : γλώσσα προγραμματισμού, βιβλιοθήκες και API.

3.2.1 Γλώσσα Προγραμματισμού

Πριν την ανάπτυξη του ελεγκτή Beacon οι κύριες γλώσσες προγραμματισμού των ελεγκτών ήταν η γλώσσα C και C++. Οι γλώσσες αυτές χρησιμοποιούνται για να παράγουν εφαρμογές πολύ υψηλής απόδοσης, αλλά επίσης επιβαρύνουν την εκτέλεση του έργου. Με την χρήση των γλωσσών αυτών παρουσιάζονται συνήθως προβλήματα όπως μεγάλος χρόνος μεταγλώττισης (> 10 λεπτών), συχνά σφάλματα στην διαδικασία της μεταγλώττισής, προβλήματα διαχείρισης μνήμης που οδηγούν σε κατακερματισμό σφαλμάτων (segmentation faults), διαρροές μνήμης και άλλα που δεν αναλύονται περαιτέρω.

Αν και έχουν γίνει προσπάθειες για την ελαχιστοποίηση προβλημάτων τέτοιου είδους , απαιτούνται τεχνικές δύσχρηστες αλλά και χρήση των περιφερειακών συσκευών , γεγονός που απαιτεί συσκευές με αυξημένες δυνατότητες. Αυτό είχε σαν αποτέλεσμα

την ανάγκη ανάπτυξης πλατφόρμας που θα βασίζεται σε φιλική γλώσσα προγραμματισμού και θα προσφέρει υψηλές αποδόσεις ενώ θα μπορεί να εγκατασταθεί σε βασικές υλικολογισμικές συσκευές (hardware), όπου η CPU και η μνήμη RAM μπορεί να επεκταθεί εύκολα με το χαμηλότερο δυνατό κόστος.

Τα βασικά χαρακτηριστικά που λήφθηκαν υπόψιν για την γλώσσα προγραμματισμού που θα επιλεχθεί για την ανάπτυξη της πλατφόρμας αυτής ήταν: η διαχείριση της μνήμης, πολλαπλή πλατφόρμα, και υψηλή απόδοσή.

Η αυτόματη διαχείριση της μνήμης (ονομάζεται και συλλογή σκουπιδιών) μπορεί να εξαλείψει τα περισσότερα σφάλματα προγραμματισμού που σχετίζονται με την μνήμη. Γλώσσες με την ικανότητα αυτή έχουν συνήθως καθόλου ή ελάχιστη σύνταξη, εξαλείφοντας τον χρόνο αναμονής για την μεταγλώττιση του προγράμματος. Επίσης οι γλώσσες του είδους αυτού παρέχουν αναφορά σφάλματος υποδεικνύοντας το ακριβές σημείο του λάθους. Οι γλώσσες που ικανοποιούν τις ικανότητες αυτές είναι η C# , Java και η Python.

Αν και η πλατφόρμα του Beacon σχεδιάστηκε για να λειτουργεί σε λειτουργικό σύστημα Linux, οι δημιουργοί του επιθυμούσαν να μπορεί να εκτελείται και στα υπόλοιπα δύο πιο σημαντικά λειτουργικά συστήματα, Windows και Mac OSX, χωρίς σημαντική προσπάθεια φορητότητας. Το πρόβλημα αυτό ήταν και από τα πιο σημαντικά που εμπόδιζε άλλες πλατφόρμες να μπορούν να εκτελούνται σε λειτουργικά συστήματα εκτός Linux.Αν και όλες οι προγραμματιστικές γλώσσες έχουν την δυνατότητα να εκτελεστούν σε όλες τις πλατφόρμες, η C# αδυνατεί να εκτελεστεί σε άλλη πλατφόρμα εκτός αυτή των Windows καθώς η CLR (Common Language Runtime) ,που αποτελεί τον επίσημο διερμηνέα της, υποστηρίζει μόνο τα Windows.Ως εκ τούτου η C# αποκλείστηκε από την επιλογή.

Η έννοια της υψηλής απόδοσης είναι υποκειμενική. Σε αυτό το πλαίσιο ένα χαρακτηριστικό του ορισμού της είναι η ικανότητα να αυξάνεται η αποδοτικότητα ανάλογα με την χρήση και των αριθμό των πυρήνων επεξεργασίας. Η έλλειψη της

δυνατότητας πολυνημάτωσης (multi-threading) από τον επίσημο διερμηνέα της Python εξαλείφει την γλώσσα αυτή ως υποψήφιο. Η υπολειπόμενη βασική επιλογή είναι η γλώσσα Java, για την οποία παραμένει άγνωστη η απόδοση της όταν χρησιμοποιείται σε ένα ελεγκτή OpenFlow.Τελικά επιλέχθηκε η Java βασιζόμενη στις υψηλές αποδόσεις που παρουσιάστηκαν από παρόμοια προγράμματα που χρησιμοποιούν αυτή την γλώσσα προγραμματισμού.

3.3 Βιβλιοθήκες

Ο ελεγκτής Beacon χρησιμοποιεί πολλαπλές βιβλιοθήκες σε μία προσπάθεια να μεγιστοποιήσει την επαναχρησιμοποίηση του κώδικα και να ελαφρύνει τον φόρτο του ίδιου του ελεγκτή αλλά και των εφαρμογών που τον χρησιμοποιούν. Η πιο σημαντική βιβλιοθήκη είναι η Spring. Το Spring, προϊόν της εταιρείας Springsource, είναι ένα ελαφρύ Java/J2EE πλαίσιο εφαρμογών που βασίζεται σε κώδικα που δημοσιεύθηκε στο βιβλίο “Expert One-on-One J2EE Design and Development” από τον βασικό συντελεστή του framework, Rod Johnson.

Κυρίως στη πλατφόρμα του Beacon χρησιμοποιούνται δυο στοιχεία της βιβλιοθήκης αυτής: Τον Container βασισμένο στο Inversion of Control (IoC) ή Dependency Injection κατά άλλους και το Web Framework.Μία κοινή εργασία της Java είναι η δημιουργία αντικειμένων και η σύνδεση μεταξύ τους με την ανάθεση τους ως ιδιότητα κάποιου άλλου. Η χρήση του πλαισίου loC επιτρέπει στους προγραμματιστές δημιουργήσουν μία λίστα σε XML, που περιλαμβάνει πληροφορίες σχετικά με τα αντικείμενα τα οποία δημιουργήθηκαν και πως αυτά συνδέονται και στην συνέχεια παρέχει μεθόδους για τον τρόπο ανάκτησης των αντικειμένων αυτών. Χρησιμοποιώντας ένα πλαίσιο loC εξοικονομείται σημαντικός χρόνος για την ανάπτυξη της πλατφόρμας σε σχέση με την κοινή εναλλακτική λύση που προβλέπει την δημιουργία πολλών κλάσεων ταυτόχρονα. Ο ελεγκτής Beacon χρησιμοποιεί το πλαίσιο loC για την διασύνδεση εντός και μεταξύ των εφαρμογών. Το πλαίσιο Web χρησιμοποιείται για την αντιστοίχιση αιτημάτων τύπου Web και REST σε απλές κλήσεις μεθόδων, και εκτελεί αυτόματη μετατροπή του αιτήματος και των δεδομένων απόκρισης σε αντικείμενα Java.

3.4 Beacon API

Η Διεπαφή Προγραμματισμού Εφαρμογών (API) για Beacon έχει σχεδιαστεί ώστε να είναι απλή και αποτελεσματική χωρίς περιορισμούς. Οι προγραμματιστές είναι ελεύθεροι να χρησιμοποιήσουν κάθε διαθέσιμη κατασκευή Java, όπως είναι τα νήματα(Threads), χρονόμετρα(Timers), υποδοχές (Sockets), κλπ. Το API περιλαμβάνει και πληροφορίες σχετικά με την αλληλεπίδραση με τους δρομολογητές OpenFlow. Ο Beacon χρησιμοποιεί το μοντέλο του παρατηρητή όπου οι ακροατές(listerners) συσχετίζονται για να ανταποκρίνονται σε συγκεκριμένα γεγονότα.

Ο ελεγκτής Beacon περιλαμβάνει την βιβλιοθήκη OpenFlowJ για την διενέργεια εργασιών σχετικά με μηνύματα τύπου OpenFlow. Η OpenFlowJ αποτελεί μία αντικειμενοστραφής εφαρμογή της Java και αναλύεται στις προδιαγραφές OpenFlow 1.0. Η OpenFlowJ περιέχει κώδικα που συμβάλλει στην αποσειριοποίηση των μηνυμάτων που προέρχονται από τα δίκτυο, σε αντικείμενα, και στην σειριοποίηση αντίστοιχων μηνυμάτων και την αποστολή στο δίκτυο.

Η αλληλεπίδραση με τους δρομολογητές OpenFlow γίνεται μέσω της κλάσης IBeaconProvider. Οι ακροατές συσχετίζονται για να ανταποκρίνονται σε γεγονότα όπως η σύνδεση ή η αφαίρεση ενός δρομολογητή (IOFSwitchListener) ,η αρχικοποίηση του δρομολογητή (IOFInitializerListener) και η λήψη συγκεκριμένων μηνυμάτων τύπου OpenFlow (IOFMessageListener). Ο κώδικας Java των κλάσεων αυτών αλλά και όλων των υπόλοιπων παρατίθεται στο παράρτημα Α.

Ο Beacon περιλαμβάνει επίσης εφαρμογές που βασίζονται σε λειτουργίες σχετικές με τον πυρήνα, προσθέτοντας επιπλέον API:

Device Manager. Παρακολουθεί τις συσκευές που έχουν συνδεθεί στο δίκτυο, συμπεριλαμβανομένων των: διευθύνσεων (Ethernet και IP), την τελευταία ημερομηνία σύνδεσης, και το δρομολογητή και πόρτα που συνδέθηκε η συσκευή την

τελευταία φορά. Το API Device Manager παρέχει μια διεπαφή (IDeviceManager) που αποσκοπεί στην αναζήτηση γνωστών συσκευών, καθώς και τη δυνατότητα ενημέρωσης για διάφορα γεγονότα όπως όταν προστίθενται νέες συσκευές, όταν πραγματοποιείται ενημέρωση, ή αφαίρεση των συσκευών.

Topology. Εντοπίζει τους συνδέσμους μεταξύ συνδεδεμένων δρομολογητών OpenFlow. Μέσω της διεπαφής του ITopology επιτρέπει την ανάκτηση ενός καταλόγου αυτών των συνδέσμων, και καταγράφει γεγονότα όπως η σύνδεση ή η αφαίρεση ενός συνδέσμου.

Routing. Προσφέρει την συντομότερη διαδρομή δρομολόγησης (σε επίπεδο 1-ζεύξης δεδομένων) μεταξύ των συσκευών του δικτύου. Αυτή η εφαρμογή χρησιμοποιεί το περιβάλλον IRoutingEngine, επιτρέποντας εναλλαγές στις ρυθμίσεις που ελέγχουν την δρομολόγηση. Οι εφαρμογές του τύπου αυτού χρησιμοποιούν όλες τις δυνατές μεθόδους υπολογισμού συντομότερης διαδρομής. Το API Routing εξαρτάται τόσο από τα αντίστοιχα Topology και Device Manager.

Web.Παρέχει ένα UI (User Interface-Διεπαφή χρήστη) μέσω τοπικού δικτύου (localhost: 8080) για τον ελεγκτή Beacon. Η δικτυακή αυτή εφαρμογή χρησιμοποιεί την διεπαφή IWebManageable, επιτρέποντας την πρόσθεση νέων στοιχείων στο UI.

3.5 Δομοστοιχείωση εκτέλεσης(Runtime Modularity)

Οι περισσότεροι ελεγκτές έχουν την δυνατότητα να επιλέγουν τις εφαρμογές που θα δημιουργήσουν (δομοστοιχείωση χρόνου μεταγλώττισης) και αυτές που θα εκτελέσουν κατά την εκκίνηση του ελεγκτή (δομοστοιχείωση χρόνου εκκίνησης).Ο ελεγκτής Beacon παρέχει την δυνατότητα όχι μόνο να προσθέσει και να αφαιρέσει εφαρμογές κατά την εκκίνηση της λειτουργίας του, αλλά και τη πρόσθεση και αφαίρεση αυτών χωρίς να διακοπεί η λειτουργία του ελεγκτή Beacon (δομοστοιχείωση εκτέλεσης).

Το γεγονός αυτό επιτρέπει στους προγραμματιστές την ανάπτυξη νέων τρόπων αλληλεπίδρασης με τον ελεγκτή. Πιθανές χρήσεις των χαρακτηριστικών αυτών αποτελούν:

- Η δημιουργία και την προσωρινή εγκατάσταση εφαρμογών με στόχο την αποσφαλμάτωση μέσω της συλλογή πληροφοριών.
- Διορθώσεις σφαλμάτων ή βελτίωση των ήδη υπάρχον εφαρμογών.
- εγκατάσταση νέων εφαρμογών κατά το χρόνο εκτέλεσης από ένα "κατάστημα εφαρμογών".
- Απενεργοποίηση εφαρμογών που παρουσιάζουν εσφαλμένη συμπεριφορά.

Για να ενεργοποιήσετε αυτήν τη λειτουργία, Beacon χρησιμοποιεί μια υλοποίηση της προδιαγραφής OSGi, την εφαρμογή Equinox. Το OSGi είναι μία αρχιτεκτονική βασισμένη σε συνθετικά (components), τα οποία στη γλώσσα του OSGi ονομάζονται bundles. ΄Ενα bundle διανείμετε ως ένα αρχείο jar και προσφέρει μία λειτουργικότητα στο συνολικό πλαίσιο. ΄Ενα από τα δυνατά στοιχεία του OSGi είναι η δυναμική διαχείριση του κύκλου ζωής (life-cycle) των bundles. ΄Ενα bundle μπορεί να εγκατασταθεί, εκκινηθεί, ανανεωθεί και σταματηθεί δυναμικά, χωρίς να σταματάει η λειτουργία του υπόλοιπου πλαισίου. Προφανώς αυτό είναι ιδιαίτερα χρήσιμο για συσκευές που χρειάζεται να λειτουργούν ασταμάτητα, γιατί μπορούν να αναβαθμιστούν χωρίς διακοπή της λειτουργίας τους.

Οι προγραμματιστές καθορίζουν τον τρόπο με τον οποίο διαμορφώνεται η εφαρμογή τους. Για παράδειγμα ένα bundle μπορεί να περιέχει εφαρμογές πολλαπλές ή και μία μόνο , είτε μπορεί να υπάρχει μία που να μπορεί να διαδοθεί σε πολλαπλά bundles. Αυτές οι αποφάσεις συνήθως γίνονται με βάση το βαθμό της δομοστοιχείας που μια εφαρμογή έχει. Για παράδειγμα αν ένα μέρος της εφαρμογής μπορεί να αντικατασταθεί κατά την έναρξη ή την διάρκεια της εκτέλεσης, όπως η μηχανή παραγωγής διαδρομών που χρησιμοποιείται από την εφαρμογή δρομολόγησης του Beacon.

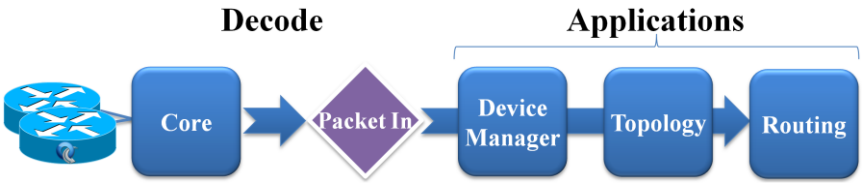
Ένα στοιχείο των προδιαγραφών OSGi είναι η καταχώρηση υπηρεσιών. Ενεργεί ως μεσολαβητής για υπηρεσίες και συμβάλλει στην καταχώρηση τους. Επιπλέον δίνουν την δυνατότητα στους καταναλωτές να ανακτήσουν μια συγκεκριμένη υπηρεσία ανάλογα τις ανάγκες τους. Ο Beacon χρησιμοποιεί σε μεγάλο βαθμό το μοντέλο αυτό δίνοντας στους πελάτες την δυνατότητα να κάνουν αιτήσεις και να λαμβάνουν εφαρμογές των υπηρεσιών αυτών. Η διάρκεια του κύκλου ζωής των υπηρεσιών είναι δυναμική: οι υπηρεσίες μπορούν να έρχονται και να φεύγουν με βάση τα bundles που είναι εγκατεστημένα και λειτουργούν ήδη.

3.6 Απόδοση

Η απόδοση ενός ελεγκτή OpenFlow συνήθως μετριέται με τον αριθμό των εισερχόμενων πακέτων που μπορεί ένας ελεγκτής να επεξεργαστεί και να απαντήσει σε ανά δευτερόλεπτο και από τον μέσο χρόνο που απαιτεί ο ελεγκτής για την επεξεργασία κάθε γεγονότος. Αυτή η ενότητα περιγράφει την αρχιτεκτονική Beacon για την επεξεργασία μηνυμάτων OpenFlow.

3.6.1 Χειρισμός γεγονότων

Οι εφαρμογές που υλοποιούν την διεπαφή IOFMessageListener δηλώνονται από την υπηρεσία IBeaconProvider η οποία λαμβάνει συγκεκριμένα μηνύματα τύπου OpenFlow που προέρχονται από τους δρομολογητές. Οι καταχωρημένοι ακροατές αποτελούν ένα σειριακό αγωγό επεξεργασίας για κάθε μήνυμα τύπου OpenFlow.Η διάταξη των ακροατών εντός κάθε αγωγού είναι ρυθμιζόμενη, και ο ακροατής μπορεί να επιλέξει αν θα διαδώσει κάθε γεγονός ή όχι. Ένα παράδειγμα του αγωγού παρουσιάζεται στο ακόλουθο σχήμα. Ο αγωγός αυτός έχει τρείς διαφορετικές εφαρμογές για την λήψη μηνυμάτων εισερχομένων πακέτων: Device Manager, Topology, και Routing.



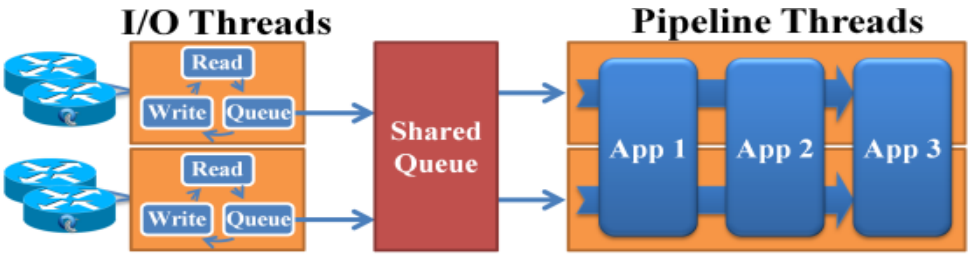
Σχήμα 3-2 : Ροή διαδικασιών του Thread IOFMessageListener

3.6.2 Ανάγνωση μηνυμάτων OpenFlow

Για την επίτευξη υψηλών επιδόσεων, ο ελεγκτής Beacon και όλες οι εφαρμογές του είναι πλήρως πολυνηματικές (multithrated). Δύο από πολυνηματικές εφαρμογές που χρησιμοποιούνται για την ανάγνωση και την επεξεργασία μηνυμάτων OpenFlow παρουσιάζονται στη συνέχεια.

3.6.2.1 Κοινόχρηστη ουρά αναμονής

Στο σχήμα 3-3 παρουσιάζεται η σχεδίαση του μοντέλου διαμοιραζόμενης ουράς αναμονής που αποτελείται από δύο σύνολα νημάτων εκτέλεσης(thread). Το πρώτο σύνολο, η ομάδα I/O, είναι υπεύθυνη για την ανάγνωση και την αποσειριοποίηση των μηνυμάτων OpenFlow που προέρχονται από τους δρομολογητές και στην συνέχεια τα αναγνωσμένα μηνύματα τοποθετούνται σε ουρά και προωθούνται στην κοινόχρηστη ουρά αναμονής όπως φαίνεται και στο σχήμα. Κάθε δρομολογητής είναι συνδεδεμένος μόνο σε ένα νήμα (εκτέλεσης) I/O, ενώ αντιθέτως πολλοί δρομολογητές μπορεί να είναι συνδεδεμένοι στο ίδιο νήμα. Το I/O νήμα μπορεί επίσης να καταγράψει τα εξερχόμενα μηνύματα του OpenFlow στους μεταγωγείς του.



Σχήμα 3-3 μοντέλο διαμοιραζόμενης ουράς

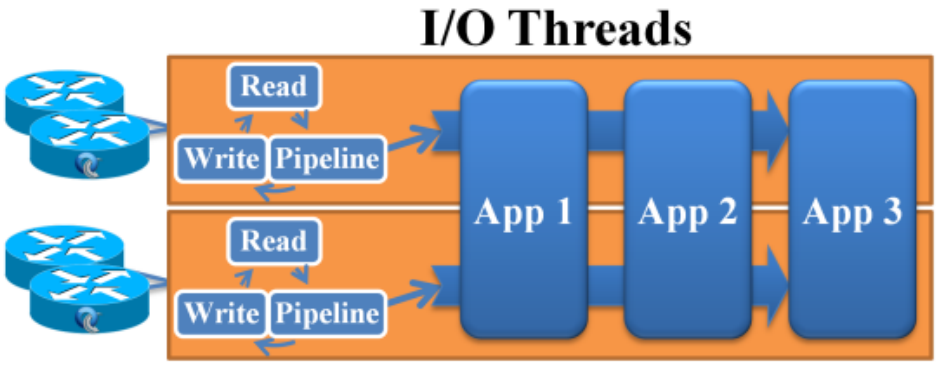
Το δεύτερο σύνολο των threads, τα λεγόμενα threads διασωλήνωσης, εξάγουν τα OpenFlow μηνύματα από την κοινόχρηστη ουρά αναμονής, και στην συνέχεια τα πακέτα προωθούνται μέσω του αγωγού IOFMessageListener που αντιστοιχίζονται ανάλογα με τον τύπο του κάθε μηνύματος. Αυτό είναι αποτελεσματικό για threads του αγωγού, κάθε φορά που υπάρχουν μηνύματα στην ουρά. Επιπλέον με τον τρόπο αυτό όλα τα threads της διοχέτευσης μπορεί να είναι απασχολημένα με την επεξεργασία των πακέτων. Ωστόσο, ο σχεδιασμός απαιτεί επίσης ασφάλεια στην κοινόχρηστη ουρά, αλλά και αντίστοιχη ασφάλεια μεταξύ των δύο συνόλων των νημάτων.

Μία σημαντική παραλλαγή του σχεδιασμού αυτού αποτελεί η ύπαρξη πολλαπλών ουρών αναμονής, ίσως μια ανά δρομολογητή, ή ακόμα περισσότερες, με την κάθε ουρά να έχει διαφορετικές προτεραιότητες. Αυτό θα μπορούσε να βελτιώσει τη δικαιοσύνη στην επεξεργασία μηνυμάτων μεταξύ των δρομολογητών, με χρονοπρογραμματισμό εξυπηρέτησης των ουρών εκ περιτροπής.

3.6.2.2 Ολοκληρωμένη εκτέλεση

Μια διεργασία παραδίδει τον έλεγχο μόνον όταν έχει ολοκληρώσει το έργο της (run to completion). Η τεχνική αυτή αποτελεί μια απλοποιημένη εκδοχή της κοινόχρηστης ουράς αναμονής, έχοντας μόλις ένα ενιαίο σύνολο νημάτων I/O . Κάθε νήμα είναι παρόμοια με τα νήματα I / O που εκτελούνται στην προηγούμενη τεχνική του κοινού σχεδιασμού ουράς. Ωστόσο αντί να τοποθετηθεί το μήνυμα σε ουρά για να επεξεργαστεί από ένα νήμα διασωλήνωσης, διοχετεύει απευθείας κάθε αναγνωσμένο

μήνυμα μέσω του αγωγού IOFMessageListener. Στο παρακάτω σχήμα απεικονίζεται η αρχιτεκτονική της τεχνικής κοινόχρηστης ουράς αναμονής.



Σχήμα 3-4 Run-to-completion

Η τεχνική αυτή δεν απαιτεί επιπλέον ασφάλεια στην διαδρομή κατά την διαδικασία της ανάγνωσης του μηνύματος, καθώς το νήμα το οποίο αποσειριακοποιεί το μήνυμα, το προωθεί μέσω του αγωγού. Περαιτέρω παρέχει την ακόλουθη εγγύηση στην διεπαφή IOFMessageListeners: Για κάθε δρομολογητή, μόνο ένα νήμα θα επεξεργάζεται μηνύματα από τον δρομολογητή αυτών την ίδια χρονική στιγμή.

Ένας σημαντικός περιορισμός της τεχνικής αυτής αποτελεί το γεγονός ότι ορισμένοι δρομολογητές που μπορεί να είναι απασχολημένοι, ενδέχεται να συνδεθούν με κάποιο υποσύνολο νημάτων, αφήνοντας άλλα νήματα σε αδράνεια. Για την αποφυγή παρόμοιων περιπτώσεων απαιτείται η περιοδική αναπροσαρμογή των δρομολογητών στα νήματα.

Ο ελεγκτής Beacon χρησιμοποιεί την τεχνική run-to-completion με ένα ρυθμιζόμενο αριθμό από I/O νήματα. Οι δρομολογητές OpenFlow κατά τη σύνδεση τους δρομολογούν τα πακέτα στα I/O νήματα κάνοντας χρήση του αλγορίθμου χρονοπρογραμματισμού Round-Robin και παραμένουν στατικά συνδεδεμένοι με το αντίστοιχο νήμα μέχρι να αποσυνδεθούν από το δίκτυο. Κάθε νήμα I/O λαμβάνει σειριακά τα δεδομένα που έχουν ήδη υποστεί επεξεργασία από κάθε δρομολογητή που είναι συνδεδεμένος με το νήμα.

3.6.2.3 Αποστολή μηνυμάτων OpenFlow

Ο τρόπος με τον οποίο τα μηνύματα αποστέλλονται από τον ελεγκτή στον δρομολογητή επηρεάζει σημαντικά την απόδοση του ελεγκτή. Ο ελεγκτής Beacon υποστηρίζει την πολυνηματική λειτουργία. Ως εκ τούτου, η αποστολή μηνυμάτων μπορεί να γίνει ταυτόχρονα από πολλαπλά threads. Έτσι είναι απαραίτητη η ανάπτυξη μεθόδων συγχρονισμού με στόχο την αποφυγή συγκρούσεων.

Στο σχήμα 55 περιλαμβάνεται ο αλγόριθμος σύμφωνα με τον οποίο σχεδιάζεται η εγγραφή τα μηνύματα που αποστέλλονται στους δρομολογητές OpenFlow. Οι εφαρμογές στην συνέχεια καλούν την μέθοδο εγγραφής, η οποία σειριοποιεί και προσαρτεί το μήνυμα σε μία ειδική προσωρινή μνήμη του δρομολογητή, και στην συνέχεια προωθεί τα δεδομένα στην υποδοχή.

```
1       function WRITE(OFMessage msg)
2               buf.append(msg)
3               flush()
4       end function
5       function FLUSH
6               socket.write(buf)
7       end function
```

Σχήμα 3-5: Μέθοδοι αρχικοποίησης δρομολογητή

Οι επιδόσεις κάνοντας χρήση του σχεδιασμού αυτού, ήταν χαμηλές. Η υπηρεσία ιχνηλάτισης έδειξε ότι η Java JVM εκτελούσε μία εγγραφή στον πυρήνα του συστήματος για κάθε εγγραφή υποδοχής, χωρίς ενδιάμεσο επεξεργασία. Σε ένα απασχολημένο σύστημα, ο χρόνος που δαπανάται σε μεταφορές δεδομένων μεταξύ χρήστη και πυρήνα ήταν σημαντική.

Για την επίλυση του προβλήματος αυτού αναπτύχθηκε μία νέα αρχιτεκτονική. Το πρώτο μέρος αυτής της αρχιτεκτονικής παρουσιάζεται στο σχήμα 3-6 , και αναφέρεται σε μια τροποποιημένη μέθοδο flush (εκτελεί αναγκαστική μεταφορά των δεδομένων

του output stream. Χρησιμοποιείται για το άδειασμα του buffer.) που περιέχει δύο boolean μεταβλητές: την write και την needSelect. Η μεταβλητή write έχει οριστεί σε αληθής όταν εμφανίζεται μια εγγραφή υποδοχής, προλαμβάνοντας τις επακόλουθες εγγραφές έως ότου η μεταβλητή τεθεί σε τιμή false. Η μεταβλητή needSelect ορίζεται όταν υπάρχουν δεδομένα ,που δεν αντιστοιχούν σε εγγραφή, μετά από κάθε εγγραφή υποδοχής, γεγονός που δείχνει οτι η ουρά TCP είναι γεμάτη, και τα υπόλοιπα δεδομένα θα που πρέπει να εγγραφούν στο μέλλον, όταν ο η προσωρινή μνήμη είναι γίνεται και πάλι διαθέσιμη.

```
1:      function FLUSH
2:              if !written && !needSelect then
3:                      socket.write(buf)
4:                      written true
5:                      if buf.remaining() > 0 then
6:                              needSelect true
7                       end if
8:              end if
9:      end function
```

Σχήμα 3-6 τροποποιημένη μέθοδο flush

Το δεύτερο μέρος του σχεδιασμού βασίζεται σε τροποποιήσεις στον βρόχο I /O όπως φαίνεται στο κάτωθι σχήμα 3-7 όπου παρουσιάζεται ο αντίστοιχος αλγόριθμος που χρησιμοποιήθηκε. Στον αλγόριθμο αυτόν προστέθηκαν οι γραμμές 3-9 και 15-18 με σκοπό της υποστήριξη της αρχιτεκτονικής αυτής. Πιο συγκεκριμένα οι γραμμές 3-9 εξασφαλίζουν ότι κάθε δρομολογητής θα δημιουργήσει εγγραφές για τα εξερχόμενα δεδομένα μία φορά ανά I/O βρόχο όσο οι εξερχόμενοι TCP buffers δεν είναι πλήρης. Οι γραμμές 15-18 εξασφαλίζουν ότι θα δημιουργηθεί εγγραφή μόνο όταν υπάρχει διαθέσιμος χώρος στους εξερχόμενους buffers.

Το αποτέλεσμα είναι ότι τα μηνύματα είτε εγγράφονται αμέσως είτε μία φορά ανά βρόχο I/O. Τα συστήματα που βρίσκονται υπο μεγάλο φόρτο εργασίας δημιουργείται ένας φυσικός μηχανισμός μεταξύ των βρόχων I/O, μειώνοντας έτσι τις κλήσεις εγγραφής και τις αλληλεπιδράσεις μεταξύ χρήστη-πυρήνα.

```
1       function IOLOOP
2               while true do
3                       for all Switch sw : switches do
4                               sw.written false
5                               sw.flush()
6                               if sw.needSelect then
7                                       sw.selectKey.addOp(WRITE)
8                               end if
9                       end for
10                      readySwitches = select(switches)
11                      for all Switch sw : readySwitches do
12                              if sw.selectKey.readable then
13                                      readAndProcessMessages(sw)
14                              end if
15                              if sw.selectKey.writeable then
16                              sw.needSelect false
17                              sw.flush()
18                              end if
19                      end for
20              end while
21      end function
```

Σχήμα 3-7 Αλγόριθμος τροποποίησης I/O Loop βρόχου

4 Mininet

4.1 Το Λογισμικό Mininet

Κατά την εκτέλεση του πειραματικού μέρους της διπλωματικής εργασίας απαραίτητο γεγονός αποτελεί η δημιουργία ενός δικτύου αποτελούμενο από δικτυακές συσκευές προώθησης και δρομολόγησης πακέτων, συσκευές που θα αποτελούν τους τελικούς χρήστες αλλά και από αυτές που θα εκτελούν τις εργασίες του ελεγκτή της τεχνολογίας OpenFlow. Όπως γίνεται αντιληπτό, η προσομοίωση ενός τέτοιου συστήματος αποτελεί τροχοπέδη καθώς απαιτούνται οικονομικοί πόροι αλλά και όχι μόνο καθώς η τοπολογία του συστήματος θα πρέπει αποτελείται από πολλές συσκευές και να είναι εύκολο τροποποιήσιμο ώστε τα αποτελέσματα να μπορούν να γενικευτούν. Επιπλέον κατά την εκτέλεση δικτυακών πειραμάτων και εργασιών είναι απαραίτητη η δημιουργία πολλαπλών τοπολογιών ώστε να μπορεί το σύστημα μας να δοκιμαστεί σε διαφορετικές συνθήκες. Όπως γίνεται αντιληπτό κάτι τέτοιο είναι πολύ δύσκολο στην περίπτωση που η προσομοίωση αυτή θα πρέπει να συμβεί με την χρήση πραγματικών συσκευών αλλά και εξίσου χρονοβόρο.

Για την διευκόλυνση όλων των παραπάνω, έγινε χρήση του προγράμματος mininet. Το πρόγραμμα Mininet αποτελεί έναν εξομοιωτή δικτύου. Έχει την δυνατότητα να εκτελεί ταυτόχρονα ένα σύνολο από τερματικά, δρομολογητές, μεταγωγείς ethernet άλλα και των αντίστοιχων συνδέσμων σε ένα ενιαίο Linux Kernel(πυρήνα). Χρησιμοποιεί την τεχνολογία της εικονοποίησης ώστε να μπορεί ένα ενιαίο σύστημα να προσομοιώνεται ως ένα πλήρες δίκτυο, χρησιμοποιώντας το ίδιο σύστημα πυρήνα και με τους ίδιους κωδικούς χρήστη. Το κάθε εικονικό τερματικό στο mininet λειτουργεί σαν ένα πραγματικό τερματικό. Επιπλέον παρέχεται η δυνατότητα ασφαλούς σύνδεσης (τύπου SSH) στο τερματικό, και να εκτελέσει οποιοδήποτε πρόγραμμα (με την προϋπόθεση ότι αυτό είναι εγκατεστημένο στο σύστημα Linux).Τα

προγράμματα που εκτελούνται μπορούν να αποστείλουν πακέτα μεταξύ των τερματικών καθώς αναγνωρίζει την σύνδεση μεταξύ τους ως διεπαφές τύπου Ethernet.Η αποστολή των πακέτων πραγματοποιείται με δεδομένη ταχύτητα σύνδεσης και την απαιτούμενη καθυστέρηση. Τα πακέτα επεξεργάζονται από συσκευές που λειτουργούν ως δρομολογητές (Ethernet swtiches, routers) με δεδομένο χρόνο σε ουρές αναμονής. Όταν δύο προγράμματα, όπως για παράδειγμα το iperf (το οποίο μετράει την χωρητικότητα της γραμμής μεταξύ δύο σημείων) μεταξύ ενός πελάτη(client) και ενός διακομιστή (server) επικοινωνούν μέσω Mininet, η μετρούμενη απόδοση θα πρέπει να είναι κοινή με αυτή των δύο φυσικών μηχανών.

Εν συντομία, στο Mininet, τα τερματικά, οι δρομολογητές, οι μεταγωγείς, οι ελεγκτές και οι συνδέσεις δημιουργούνται με τη χρήση λογισμικού (software) και όχι υλικού (hardware) . Είναι δυνατόν η δημιουργία ενός δικτύου Mininet παρόμοιο με ένα πραγματικό δίκτυο που βασίζεται σε hardware, ή η δημιουργία ενός δικτύου hardware παρόμοιο με αυτό του Mininet, τα οποία να εκτελούν τον ίδιο δυαδικό κώδικα και εφαρμογές στην καθεμία πλατφόρμα.

Όπως γίνεται αντιληπτό το mininet αποτελεί ένα εύχρηστο και αξιόπιστο εργαλείο στην προσομοίωση δικτύων συγκεντρώνοντας σημαντικά πλεονεκτήματα. Παρακάτω συνοψίζονται τα πιο αξιοσημείωτα πλεονεκτήματα.

- Η δημιουργία ενός απλουστευμένου δικτύου πραγματοποιείται σε ελάχιστο χρονικό διάστημα με αποτέλεσμα να μπορεί να πραγματοποιηθεί γρήγορα η διαδικασία της αποσφαλμάτωσης (debugging).
- Παρέχεται η δυνατότητα εκτέλεσης όλων των λογισμικών που υποστηρίζονται από το λειτουργικό σύστημα Linux
- Μπορεί να τροποποιηθεί η προώθηση των πακέτων: Οι δρομολογητές-Mininet μπορούν να προγραμματιστούν χρησιμοποιώντας OpenFlow πρωτόκολλο.

- Το Mininet μπορεί να εκτελεστεί σε οποιαδήποτε υπολογιστή, διακομιστή, εικονική μηχανή ή ακόμα και σε τεχνολογία τύπου cloud (νεφοϋπολογιστική).
- Τα αποτελέσματα του λογισμικού μπορούν να αναπαραχθούν από οποιοδήποτε χρήστη καθώς το μόνο που απαιτείται είναι η εκτέλεση του ιδίου κώδικα στο αντίστοιχο τερματικό.
- Το Mininet αποτελεί εύχρηστο λογισμικό. Για την δημιουργία και την εκτέλεση πειραμάτων απαιτείται προγραμματισμός σε γλώσσα Python.
- Αποτελεί έργο ανοιχτού κώδικα και βρίσκεται υπό ενεργή ανάπτυξη. Η κοινότητα του Mininet αποτελείται από χρήστες και προγραμματιστές και μπορούν να συμβάλλουν στην αντιμετώπιση οποιουδήποτε προβλήματος που μπορεί να αντιμετωπίσει ο εκάστοτε χρήστης.

4.2 *Δημιουργία τοπολογιών*

Το mininet υποστηρίζει την δημιουργία παραμετροποιήσιμων τοπολογιών. Με την δημιουργία του αντίστοιχου κώδικα Python, παρέχεται η δυνατότητα δημιουργίας ευέλικτης τοπολογίας η οποία μπορεί να διαμορφωθεί με βάση τα στοιχεία που εντάσσονται στον κώδικα, και μπορεί να επαναχρησιμοποιηθεί σε πολλαπλά πειράματα.

Για παράδειγμα, παρακάτω παρουσιάζεται μία τοπολογία του δικτύου η οποία αποτελείται από ένα καθορισμένο αριθμό χρηστών(hosts) και συνδέονται με ένα μεταγωγέα.

```
#!/usr/bin/python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def __init__(self, n=2, **opts):
        # Initialize topology and default options
        Topo.__init__(self, **opts)
```

```
            switch = self.addSwitch('s1')
            # Python's range(N) generates 0..N-1
            for h in range(n):
                host = self.addHost('h%s' % (h + 1))
                self.addLink(host, switch)

    def simpleTest():
        "Create and test a simple network"
        topo = SingleSwitchTopo(n=4)
        net = Mininet(topo)
        net.start()
        print "Dumping host connections"
        dumpNodeConnections(net.hosts)
        print "Testing network connectivity"
        net.pingAll()
        net.stop()

    if __name__ == '__main__':
        # Tell mininet to print useful information
        setLogLevel('info')
        simpleTest()
```

Οι κλάσεις και οι συναρτήσεις που χρησιμοποιήθηκαν επεξηγούνται περαιτερω:

Topo: Η βασική κλάση που χρησιμοποιείται στις τοπολογίες Mininet

addSwitch(): προσθέτει έναν δρομολογητή στην τοπολογία και επιστρέφει την ονομασία του δρομολογητή

addHost():προσθέτει τερματικό στην τοπολογία και επιστρέφει την ονομασία του

addLink():προσθέτει μια αμφίδρομη σύνδεση στην τοπολογία.(Οι σύνδεσεις στο Mininet είναι αμφίδρομες, εκτός αν αναφέρεται διαφορετικά.)

Mininet: κύρια κατηγορία για την δημιουργία και τη διαχείρηση του δικτύου

start(): Ενεργοποιεί την λειτουργία του δικτύου.

pingAll(): ελέγχει την συνδεσιμότητα των τερματικών εκτελώντας διαδοχικά αιτήσεις ping μεταξύ των κόμβων.

stop(): Τερματίζει την λειτουργία του δικτύου.

net.hosts: Επιστρέφει την ονομασία όλων των κόμβων

dumpNodeConnections(): Απορρίπτει συνδέσεις προς/από ένα σύνολο κόμβων

4.3 *Ρύθμιση παραμέτρων απόδοσης*

Εκτός από τις βασικές λειτουργίες της δικτύωσης, το mininet παρέχει δυνατότητα ρύθμισης της απόδοσης και απομόνωσης ορισμένων χαρακτηριστικών, μέσω των κλάσεων `CPULimitedHost` και `TCLink`.

```
#!/usr/bin/python

    from mininet.topo import Topo
    from mininet.net import Mininet
    from mininet.node import CPULimitedHost
    from mininet.link import TCLink
    from mininet.util import dumpNodeConnections
    from mininet.log import setLogLevel

    class SingleSwitchTopo(Topo):
        "Single switch connected to n hosts."
        def __init__(self, n=2, **opts):
            Topo.__init__(self, **opts)
            switch = self.addSwitch('s1')
            for h in range(n):
                # Each host gets 50%/n of system CPU
                host = self.addHost('h%s' % (h + 1),
                   cpu=.5/n)
                # 10 Mbps, 5ms delay, 10% loss, 1000 packet queue
                self.addLink(host, switch,
                   bw=10, delay='5ms', loss=10, max_queue_size=1000,
use_htb=True)

    def perfTest():
        "Create network and run simple performance test"
        topo = SingleSwitchTopo(n=4)
        net = Mininet(topo=topo,
                      host=CPULimitedHost, link=TCLink)
        net.start()
        print "Dumping host connections"
        dumpNodeConnections(net.hosts)
        print "Testing network connectivity"
        net.pingAll()
        print "Testing bandwidth between h1 and h4"
        h1, h4 = net.get('h1', 'h4')
        net.iperf((h1, h4))
        net.stop()

    if __name__ == '__main__':
        setLogLevel('info')
        perfTest()
```

Οι πιο σημαντικές μέθοδοι και παράμετροι που χρησιμοποιήθηκαν επεξηγούνται περαιτέρω:

`self.addHost(name, cpu=f)`: Με την χρήση της εντολής αυτής επίτρεπεται ο ορισμός του ποσοστού της συνολικής CPU του συστήματος που θα χρησιμοποιήσει ο εικονικός χρήστης.

`self.addLink( node1, node2, bw=10, delay='5ms', max_queue_size=1000, loss=10, use_htb=True)`: Δημιουργεί σύνδεση διπλής κατεύθυνσης μεταξύ δύο κόμβων με συγκεκριμένα χαρακτηριστικά όπως χωρητικότητα, καθυστέρηση, ανεκτικότητα απωλειών πακέτων, με μέγιστο μέγεθος της ουράς αναμονής τα 100 πακέτα. Η παράμετρος `bw` εκφράζεται σε Mb/s, ενώ η `delay` ακολουθείται με την αντίστοιχη μονάδα χρόνου (s,ms,us).Αντίθετα η παράμετρος `loos` εκφράζεται σε ποσοστό επι τοις εκατό.

4.4 *Εκτέλεση προγραμμάτων στα εικονικά τερματικά*

Η εκτέλεση προγραμμάτων στα τερματικά αποτελεί το πιο αξιοσημείωτο γεγονός κατά την εκτέλεση των πειραμάτων, έτσι ώστε να μπορούν να υποστηριχτούν επιπλέον εντολές από τις συνηθισμένες εντολές τύπου pingAll () και iperf (). Η διαδικασία αυτή υποστηρίζεται από το λογισμικό Mininet.

Κάθε τερματικό στο Mininet αποτελεί ουσιαστικά ένα κέλυφος τύπου bash που συνδέεται με μία ή και περισσότερες διεπαφές δικτύου με αποτέλεσμα να υποστηρίζει την εκτέλεση εντολών τύπου bash. Για τον λόγο αυτό για την επικοινωνία με κάθε τερματικό χρησιμοποιείται κυρίως μέθοδος τύπου CMD.

Για την εκτέλεση μίας εντολής από κάποιον ξενιστή και την αποτύπωση του αποτελέσματος, μέσω μεθόδου cmd, χρησιμοποιείται ο παρακάτω κώδικας

```
h1 = net.get('h1')
    result = h1.cmd('ifconfig')
    print result
```

Σε αρκετές περιπτώσεις απαιτείται η εκτέλεση μιας εντολής στο προσκήνιο για κάποιο διάστημα, η διακοπή της ή η αποθήκευση του αποτελέσματος σε κάποιο αρχείο.

```
    from time import sleep
    ...
    print "Starting test..."
```

```
    h1.cmd('while true; do date; sleep 1; done > /tmp/date.out &')
    sleep(10)
    print "Stopping test"
    h1.cmd('kill %while')
    print "Reading output"
    f = open('/tmp/date.out')
    lineno = 1
    for line in f.readlines():
        print "%d: %s" % ( lineno, line.strip() )
        lineno += 1
    f.close()
```

Εκτός από τη χρήση του μηχανισμού αναμονής του κελύφους, το Mininet παρέχει την δυνατότητα επιτρέπει την εκτέλεση ενός συνόλου εντολών χρησιμοποιώντας την εντολή sendCmd () και, στη συνέχεια, η οποία αναμένετε να ολοκληρωθεί σε κάποια μεταγενέστερη στιγμή, χρησιμοποιώντας την εντολή waitOutput ():

```
for h in hosts:
        h.sendCmd('sleep 20')
    …
    results = {}
    for h in hosts:
        results[h.name] = h.waitOutput()
```

4.5 *Σύστημα αρχείων Mininet*

Οι εικονικοί ξενιστές του mininet διαμοιράζονται από προεπιλογή τους φακέλων root του συστήματος του υποκείμενου διακομιστή. Αντίθετα η δημιουργία νέου ξεχωριστού συστήματος (filesystem) αποτελεί χρονοβόρα διαδικασία και ιδιαίτερα δύσκολη.

Η κοινή χρήση των αρχείων του συστήματος παρέχει το πλεονέκτημα ότι δεν θα χρειαστεί η αντιγραφή των δεδομένων μεταξύ των ξενιστών καθώς έχουν ήδη δημιουργηθεί.

Το γεγονός αυτό όμως, έχει και ένα σημαντικό μειονέκτημα. Σε περίπτωση που απαιτείται ειδική διαμόρφωση για κάποιο πρόγραμμα (πχ. Httpd), απαιτείται η δημιουργία νέων αρχείων ρύθμισης παραμέτρων για κάθε ξενιστή. Επιπλέον δημιουργείται και ο κίνδυνος συγκρούσεων αρχείων, σε περίπτωση που δημιουργηθεί στον ίδιο κατάλογο το ίδιο αρχείο.

4.6 *Μέθοδοι παραμετροποίησης των ξενιστών*

Οι ξενιστές στο mininet παρέχουν μια σειρά από μεθόδους που συμβάλλουν στην ευκολία της διαμόρφωση του δικτύου.

`IP()`:Επιστρέφει την διεύθυνση IP του τερματικού η κάποιας συγκεκριμένης διεπαφής.

`MAC()`:Επιστρέφουν την διεύθυνση MAC του τερματικού η κάποιας συγκεκριμένης διεπαφής.

`setARP()`: Δημιουργεί εγγραφή static ARP στην ARP cache του τερματικού.

`setIP()`: Ρυθμίσεις συγκεκριμένη διεύθυνση IP για κάποιο τερματικό η διεπαφή.

`setMAC()`: Ρυθμίσεις συγκεκριμένη διεύθυνση IP για κάποιο τερματικό η διεπαφή.

Για παράδειγμα:

```
print "Host", h1.name, "has IP address", h1.IP(), "and MAC address",
h1.MAC()
```

4.7 *Mininet CLI*

Το Mininet περιλαμβάνει περιβάλλον γραμμής εντολών (Command Line Interface) που μπορούν να λειτουργήσει και σε ένα δίκτυο. Παρέχει μια ποικιλία από χρήσιμες εντολές, καθώς και τη δυνατότητα εμφάνισης παραθύρων xterm για την εκτέλεση εντολών σε επιμέρους κόμβους του δικτύου σας.

```
from mininet.topo import SingleSwitchTopo
    from mininet.net import Mininet
    from mininet.cli import CLI
```

```
    net = Mininet(SingleSwitchTopo(2))
    net.start()
    CLI(net)
    net.stop()
```

Η χρήση της γραμμής εντολών συμβάλει και στην αποσφαλμάτωση του δικτύου, καθώς εμφανίζει την δικτυακή τοπολογία (με χρήση της εντολής net), τον έλεγχο την συνδεσιμότητας (με την εντολή pingall) και την αποστολή εντολών σε όλα τα τερματικά ανεξάρτητα.

```
*** Starting CLI:
mininet> net
c0
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (0/2 lost)
mininet> h1 ip link show
746: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
749: h1-eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
    link/ether d6:13:2d:6f:98:95 brd ff:ff:ff:ff:ff:ff
```

4.8 Mininet API

Καλούμε API ή Διεπαφή Προγραμματισμού Εφαρμογών γνωστή και ως Διασύνδεση Προγραμματισμού Εφαρμογών, τη διεπαφή των προγραμματιστικών διαδικασιών που ένα λειτουργικό σύστημα, βιβλιοθήκη ή εφαρμογή παρέχει προκειμένου να επιτρέπει να γίνονται προς αυτό αιτήσεις από άλλα προγράμματα ή / και ανταλλαγή δεδομένων.

Στις προηγούμενες παραγράφους παρουσιάστηκε μια σειρά από κλάσεις της Python που περιλαμβάνονται στο API του Mininet, συμπεριλαμβανομένου των κλάσεων `Topo`, `Mininet`, `Host`, `Switch`, `Link` και των υποκατηγοριών τους. Λόγω απλοποίησης της και διευκόλυνσης του προγραμματισμού οι κλάσεις χωρίζονται σε τρείς κατηγορίες-επίπεδα: υψηλού επιπέδου API, μεσαίου επιπέδου API και χαμηλού επιπέδου API.

- Χαμηλού επιπέδου API: Αποτελείται από τις κύριες κλάσεις που αφορούν τους κόμβους και τις συνδέσεις (όπως `Host`, `Switch`, και `Link` και τις υποκατηγορίες τους) οι οποίες χρησιμοποιούνται για την δημιουργία δικτύου. Η τρόπος δημιουργίας δικτύου μόνο με κλάσεις του επιπέδου αυτού αποτελεί διαδικασία ιδιαίτερα δύσχρηστη.

- Μεσαίου επιπέδου API: Προσθέτει αντικείμενα τύπου `Mininet` , τα οποία χρησιμεύουν ως επιπρόσθετα στοιχεία και ρυθμίσεις σε κόμβους και συνέσεις. Παρέχει μία σειρά από μεθόδους (όπως `addHost()`, `addSwitch()`, και `addLink()`) για την πρόσθεση κόμβων και συνδέσεων στον δίκτυο ,καθώς και για την διαμόρφωση του δικτύου, την εκκίνηση και τον τερματισμό του (`start()`,`stop()`) .

- Υψηλού επιπέδου API: Παρέχει πρόσθετες επιλογές ρύθμισης της τοπολογίας. Μέσω της κλάσης `topo` προσφέρει την δυνατότητα δημιουργίας υποδείγματος τοπολογίας που μπορεί να παραμετροποιηθεί και να επαναχρησιμοποιηθεί. Τα υποδείγματα του τύπου αυτού ορίζονται μέσω της εντολής `mn` (μέσω του ορίσματος `--custom option`) και εκτελούνται από την γραμμή εντολών.

Γενικότερα για τον απευθείας έλεγχο των κόμβων και των δρομολογητών, χρησιμοποιείται το API χαμηλού επιπέδου. Αντιθέτως για την διακοπή της λειτουργίας ενός δικτύου χρησιμοποιείται το API μεσαίου επιπέδου.

Η δημιουργία ενός πλήρους και λεπτομερέστατου δικτύου μπορεί να πραγματοποιηθεί με την χρήση οποιουδήποτε επιπέδου API, αλλά συνήθως επιλέγεται το μεσαίο και το υψηλό επίπεδο χάρη στις κλάσεις που περιέχουν και διευκολύνουν αρκετά την δημιουργία του. Παρακάτω παρουσιάζονται τρία παραδείγματα δημιουργίας ενός πλήρους δικτύου χρησιμοποιώντας ένα διαφορετικό επίπεδο API κάθε φορά.

Low-level API: κόμβοι και συνδέσεις

```
h1 = Host( 'h1' )
h2 = Host( 'h2' )
s1 = OVSSwitch( 's1', inNamespace=False )
c0 = Controller( 'c0', inNamespace=False )
Link( h1, s1 )
Link( h2, s1 )
h1.setIP( '10.1/8' )
h2.setIP( '10.2/8' )
c0.start()
s1.start( [ c0 ] )
print h1.cmd( 'ping -c1', h2.IP() )
s1.stop()
c0.stop()
```

Mid-level API: Δίκτυο

```
net = Mininet()
h1 = net.addHost( 'h1' )
h2 = net.addHost( 'h2' )
s1 = net.addSwitch( 's1' )
c0 = net.addController( 'c0' )
net.addLink( h1, s1 )
net.addLink( h2, s1 )
net.start()
print h1.cmd( 'ping -c1', h2.IP() )
CLI( net )
net.stop()
```

High-level API: Υπόδειγμα Τοπολογίας

```
class SingleSwitchTopo( Topo ):
    "Single Switch Topology"
    def __init__( self, count=1, **params ):
        Topo.__init__( self, **params )
        hosts = [ self.addHost( 'h%d' % i )
                  for i in range( 1, count + 1 ) ]
        s1 = self.addSwitch( 's1' )
        for h in hosts:
            self.addLink( h, s1 )

net = Mininet( topo=SingleSwitchTopo( 3 ) )
net.start()
CLI( net )
net.stop()
```

Το μεσαίο επίπεδο API αποτελείται από απλούστερη δόμηση καθώς όπως φαίνεται και στο παραπάνω παράδειγμα απαιτεί την δημιουργία κλάσης τοπολογίας. Αν το API χαμηλού και μεσαίου επιπέδου είναι ευέλικτα, έχουν το μειονέκτημα ότι είναι περισσότερο δύσχρηστη η επαναχρησιμοποίηση τους σε αντίθεση με το API υψηλού επιπέδου.

4.9 *Εργαλεία μέτρησης*

Το mininet περιέχει εντολές-εργαλεία που καταγράφουν μετρήσεις και συμβάλουν στον έλεγχο του δικτύου και στην προσπάθεια αποσφαλμάτωσης του. Παρακάτω παρουσιάζονται τα πιο σημαντικά εργαλεία μαζί με την αντίστοιχες εντολές τους.

- Εύρος ζώνης (`bmw-ng, ethstats`)
- Καθυστέρηση (μέσω της εντολής `ping`)
- Ουρές αναμονής (μέσω της εντολής `tc` που περιλαμβάνεται στην κλάση `monitor.py`)
- Στατιστικά TCP (`tcp_probe`)
- Χρήση CPU (`top,cpuacct`)

4.10 *OpenFlow και προσαρμοσμένη δρομολόγηση*

Ένα από τα πιο ισχυρά και χρήσιμα χαρακτηριστικά του mininet είναι ότι χρησιμοποιεί Δίκτυα Καθοριζόμενα από Λογισμικό. Με την χρήση του OpenFlow πρωτοκόλλου επιτρέπεται ο προγραμματισμός των δρομολογητών έτσι ώστε να μπορούν να λαμβάνουν αποφάσεις για τα πακέτα που εισέρχονται σε αυτούς. Η τεχνολογία OpenFlow καθιστά τους προσομοιωτές, όπως είναι το Mininet, περισσότερο χρήσιμους δεδομένου ότι ο σχεδιασμός των δικτυακών συστημάτων, συμπεριλαμβανομένου και την προσαρμοσμένη προώθηση πακέτων με χρήση OpenFlow,μπορεί να μεταφερθεί σε OpenFlow δρομολογητές για λειτουργίες σχετικά με το ρυθμό γραμμής.

4.10.1 OpenFlow Ελεγκτές

Κατά την διενέργεια ενός πειράματος το mininet χρησιμοποιεί ως προεπιλογή ελεγκτή τύπου `ovsc`. Η αντίστοιχη ισοδύναμη εντολή είναι:

```
$ sudo mn --controller ovsc
```

Ο ελεγκτής αυτού του τύπου υλοποιεί ένα απλό δρομολογητή μάθησης Ethernet,και υποστηρίζει εως και δεκαέξι επιμέρους δρομολογητές.

Κατά την κατασκευή ενός σεναρίου(Script) όταν εκτελεστεί η κλάση Mininet() θα πρέπει να έχει οριστεί και αντίστοιχη κλάση του ελεγκτή. Σε περίπτωση που δεν έχει δηλωθεί κάποια από τον χρήστη κάποια συγκεκριμένη κλάση ελεγκτή τότε καλείται η προεπιλεγμένη κλάση Controller()με αποτέλεσμα την δημιουργία ελεγκτών τύπου Stanford/OpenFlow.

Αντιθέτως παρέχεται και η δυνατότητα χρησιμοποιήσεις διαφορετικού τύπου ελεγκτή ανάλογα με τις ανάγκες της εφαρμογής. Ο χρήστης μπορεί να δημιουργήσει μία υποκατηγορία Controller() και να την μεταφέρει στα αρχεία του συστήματος του Mininet. Ένα παράδειγμα υλοποίησης ενός ελεγκτή τύπου POX παρατίθεται παρακάτω.

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller
from mininet.topo import SingleSwitchTopo
from mininet.log import setLogLevel

import os

class POXBridge( Controller ):
    "Custom Controller class to invoke POX forwarding.l2_learning"
    def start( self ):
        "Start POX learning switch"
        self.pox = '%s/pox/pox.py' % os.environ[ 'HOME' ]
        self.cmd( self.pox, 'forwarding.l2_learning &' )
    def stop( self ):
        "Stop POX"
        self.cmd( 'kill %' + self.pox )

controllers = { 'poxbridge': POXBridge }

if __name__ == '__main__':
    setLogLevel( 'info' )
    net = Mininet( topo=SingleSwitchTopo( 2 ), controller=POXBridge )
    net.start()
    net.pingAll()
    net.stop()
```

Αξιοσημείωτο είναι ότι το παραπάνω παράδειγμα είναι έτσι δομημένο ώστε να μπορεί να χρησιμοποιηθεί ως ο προεπιλεγμένος ελεγκτής στην εντολή `mn` για την διεκπεραίωση δοκιμών.

4.10.2 Εξωτερικοί ελεγκτές OpenFlow

Στην παρούσα διπλωματική εργασία γίνεται χρήση του ελεγκτή Beacon που έχει αναπτυχθεί σε περιβάλλον Java. Αυτό έχει ως αποτέλεσμα να μην συνίσταται να δημιουργηθεί στο Mininet κλάση που να αναφέρεται στον ελεγκτή Beacon καθώς το Mininet χρησιμοποιεί κλάσεις που έχουν αναπτυχθεί σε περιβάλλον Python. Για τον λόγο αυτό το Mininet παρέχει την δυνατότητα σύνδεσης με ελεγκτή που εκτελείται σε κάποιο τερματικό στο τοπικό δίκτυο σε εικονική μηχανή ή ακόμα και στον ίδιο τον υπολογιστή μας.

Η κλάση `RemoteController()` λειτουργεί ως διαμεσολαβητής για έναν ελεγκτή ο οποίος μπορεί να λειτουργεί σε οποιοδήποτε σημείο του δικτύου ελέγχου, με την διαφορά ότι η ένερξη και η διακοπη της λειτουργίας του θα πρέπει να γίνει με τρόπο χειροκίνητο ή με κάποιο μηχανισμό που δεν ελέγχεται από το Mininet.

Παρακάτω παρατίθενται ορισμένες χρήσεις της κλάσης `RemoteController()`

```
from functools import partial
net = Mininet( topo=topo, controller=partial( RemoteController,
ip='127.0.0.1', port=6633



net = Mininet( topo=topo, controller=lambda name: RemoteController( name,
ip='127.0.0.1' ) )



net = Mininet( topo=topo, controller=None)
net.addController( 'c0', controller=RemoteController, ip='127.0.0.1',
port=6633 )
```

Όπως φαίνεται και στα παραδείγματα ο ελεγκτής αποτελεί συνάρτηση δόμησης και όχι αντικείμενο. Παρέχεται η δυνατότητα δημιουργίας συνάρτησης δόμησης εν σειρά χρησιμοποιώντας το όρισμα `partial` ή `lambda` ή δημιουργώντας μία συνάρτηση που θα παίρνει ορίσματα και θα επιστρέφει τον ελεγκτή ως αντικείμενο. Τέλος δίνεται και η δυνατότητα εισαγωγής του ελεγκτή και ως κλάση (υποκατηγορία της κλάσης `RemoteController()`.

Μία επίσης χρήσιμη ιδιότητα αποτελεί οτι είναι εφικτή η δημιουργία πολλαπλών ελεγκτών και η δημιουργία μιας υποκατηγορίας της κλάσης Switch() που θα επιτρέπει την σύνδεση με διαφορετικούς ελεγκτές:

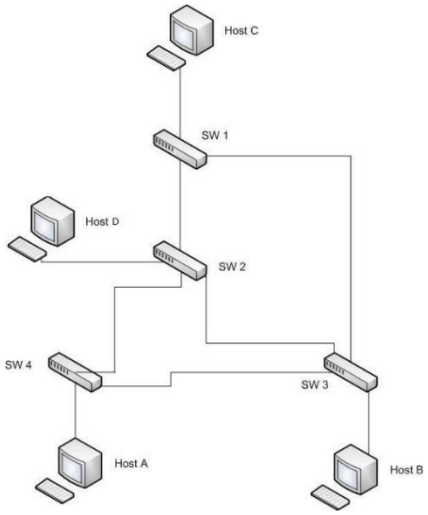
```
c0 = Controller( 'c0' )  # local controller
c1 = RemoteController( 'c1', ip='127.0.0.2' )  # external controller
cmap = { 's1': c0, 's2': c1, 's3': c1 }

class MultiSwitch( OVSSwitch ):
    "Custom Switch() subclass that connects to different controllers"
    def start( self, controllers ):
        return OVSSwitch.start( self, [ cmap[ self.name ] ] )
```

Τέλος επιτρέπεται ο ορισμός του εξωτερικού ελεγκτή μέσω της γραμμής εντολής:

```
$ sudo mn --controller remote,ip=192.168.51.101
```

4.11 Τοπολογία

Το API του Mininet επιτρέπει στον χρήστη την δημιουργία προσαρμοσμένων δικτύων ανάλογα με τις ανάγκες του, με την χρήση λίγων γραμμών κώδικα σε γλώσσα Python.Στην παρούσα εργασία επιλέχθηκε η παρακάτω τοπολογία.



Σχήμα 4-1 Τοπολογία

Σύμφωνα με το παραπάνω σχήμα παρατηρούμε ότι η τοπολογία μας αποτελείται από τρία τερματικά και τέσσερις δρομολογητές με τις αντίστοιχες

συνδέσεις που φαίνονται. Για την δημιουργία όμως της παραπάνω τοπολογίας απαιτείται όπως ειπώθηκε και παραπάνω η μετάφραση του σε γλώσσα Python.Στην παρακάτω εικόνα παρουσιάζεται ο κώδικας της τοπολογίας

```
"""Custom topology example

Topology consists of 4 switches plus 3 hosts :

Adding the 'topos' dict with a key/value pair to generate our newly
defined
topology enables one to pass in '--topo=mytopo' from the command line.
"""

from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        A = self.addHost( 'h1' )
        B = self.addHost( 'h2' )
        C = self.addHost( 'h3' )
        D = self.addHost( 'h4' )
        SW1 = self.addSwitch( 's1' )
        SW2 = self.addSwitch( 's2' )
        SW3 = self.addSwitch( 's3' )
        SW4 = self.addSwitch( 's4' )

        # Add links
        self.addLink( C, SW1 )
        self.addLink( SW1, SW2 )
        self.addLink( SW2, SW4 )
        self.addLink( SW2, SW3 )
        self.addLink( SW1, SW3 )
        self.addLink( SW4, A )
        self.addLink( SW3, B )
        self.addLink( SW3, SW4 )
        self.addLink( SW2, D )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

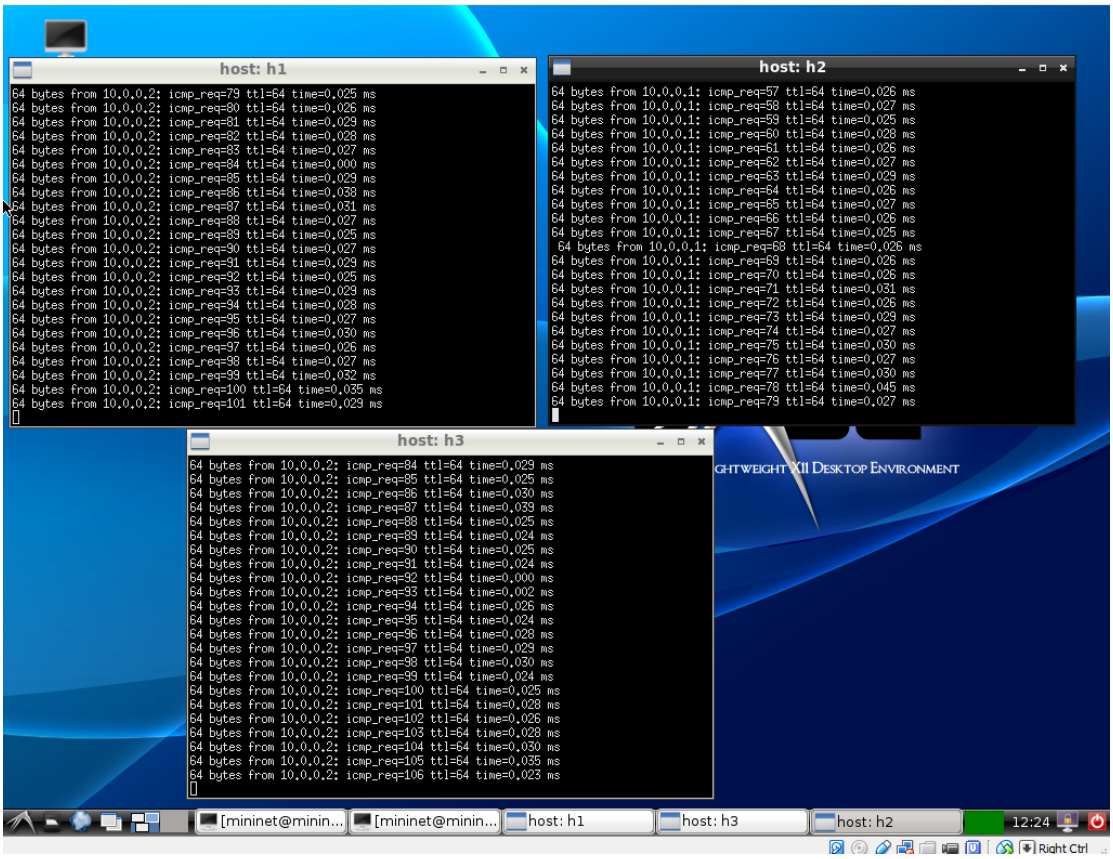
Σχήμα 4-2 2 Τοπολογία σε μορφή Python

Στο παραπάνω σχήμα φαίνεται ο κώδικας της τοπολογίας και τα στοιχεία που το αποτελούν. Με τον τρόπο αυτό μπορεί κανείς να φτιάξει την τοπολογία του που να καλύπτει τις ανάγκες του για να μπορεί να πραγματοποιείσαι τις προσομοιώσεις που επιθυμεί. Στην συνέχεια για την εκτέλεση της προσομοίωσης χρειάζεται να ανατρέξουμε στον mininet μέσω της παρακάτω εντολής.

```
sudo mn –custom topologia.py –topo mytopo – mac –controller=remote,
ip=192.168.1.88
```

Στο σημείο αυτό θα επεξηγήσουμε τις παραπάνω εντολές ώστε να γίνει κατανοητή η χρήση της κάθε εντολής. Με τον όρο sudo mn , ο χρήστης αποκτά δικαιώματα διαχειριστή στον mininet και δίνει οδηγίες στο mininet ναι εκτελέσει το αρχείο topologia.py που βρίσκεται στο φάκελο custom και να δημιουργήσει την τοπολογία mytopo με τις αντίστοιχες συνδέσεις που την διέπουν όπως αυτή ορίζεται στο αρχείο topologia.py. Ο λόγος που συμβαίνει αυτή η επισήμανση από το πρόγραμμα είναι καθώς στο αρχείο μπορεί να περιλαμβάνονται παραπάνω από μία τοπολογίες. Επιπλέον με το όρισμα –mac ρυθμίζεται η διεύθυνση MAC του κάθε τερματικού ίσο με την διεύθυνση IP του. Αυτό γίνεται για λόγους απλούστευσης κατά την διαδικασία ανάλυσης των αποτελεσμάτων. Επιπλέον για την δημιουργία κίνησης από τερματικό σε τερματικό και γενικότερα τον έλεγχο λειτουργιών των τερματικών χρησιμοποιείται το όρισμα –x πού έχει ως αποτέλεσμα την δημιουργία παραθύρων εκτέλεσης εντολών (command prompt) ένα για κάθε συσκευή. Τέλος το πιο σημαντικό όρισμα είναι το –controller=remote, ip=192.168.1.88.Με την εντολή αυτή ορίζουμε τον ελεγκτή της τοπολογίας (OpenFlow Controller) ότι βρίσκεται στην διεύθυνση με ip την 192.168.1.88 . Αυτό το γεγονός επιτρέπει στον ελεγκτή να βρίσκεται κάπου απομακρυσμένα όπως στην προκειμένη περίπτωση πού βρίσκεται εκτός της εικονικής μηχανής.

Στην παρακάτω εικόνα φαίνεται η δημιουργία κίνησης από τους host A,Host C προς τον Host B ο όποιος έχει διεύθυνση ip 10.0.0.2 και από τον Host B προς τον Host 1 με ip 10.0.0.1 .Η δημιουργία κίνησης γίνεται με την επαναλαμβανομένη κλήση αιτήσεων τύπου ping μέσω της εντολής ping 10.0.0.2 και με την αντίστοιχη για τον Host B.



Σχήμα 4-3 Δημιουργία Κίνησης μεταξύ των Host μέσω του λογισμικού Mininet

5 *Ενεργειακή Δρομολόγηση*

Η ενεργειακή απόδοση έχει πλέον καταστεί ζωτικής σημασίας για όλες τις βιομηχανίες και γενικότερα για ολόκληρη την κοινωνία, καθώς υπάρχει ισχυρό κίνητρο για την μείωση του κεφαλαίου και των δαπανών. Σύμφωνα με την παγκόσμια βιβλιογραφία, η ετήσια ηλεκτρική ενέργεια που καταναλώνεται από συσκευές του δικτύου στις Η.Π.Α. υπολογίζεται στα 6,06 Terra Watt ανα ώρα, το οποίο μεταφράζεται περίπου σε ένα δισεκατομμύριο δολάρια τον χρόνο, παρουσιάζοντας έτσι ένα ισχυρό επιχείρημα για την μείωση της ενέργειας που καταναλώνεται από συσκευές δικτύωσης, όπως συσκευές hubs, πρόσβασης δικτύου(access points), μεταγωγείς και δρομολογητές.

5.1 Ενεργειακό προφίλ δικτυακών συσκευών

Στις μέρες μας έχουν παρουσιαστεί αρκετές προσεγγίσεις για τη μείωση της ενεργειακής κατανάλωσης των δικτύων. Ορισμένες από αυτές αφορούν την χρήση της κατάστασης sleep mode στην οποία οι συσκευές εισέρχονται σε κατάσταση χαμηλής κατανάλωσης ενέργειας κατά τη διάρκεια των περιόδων αδράνειας, και την υιοθέτηση ανάλογων ενεργειακών μηχανισμών όπου ο αρχιτεκτονικός σχεδιασμός της συσκευής παρέχει την δυνατότητα να κάνει την κατανάλωση ενέργειας ανάλογη του πραγματικού φορτίου.

Η ενέργεια που αποθηκεύεται σ’ αυτές τις συσκευές πολλές φορές χάνεται λόγω επιβάρυνσης της από τα επιπλέον φορτίο. Για αυτό θα πρέπει να αλλάζει αναλόγως την αναλογικότητα του φορτίου. Ως εκ τούτου, δεν είναι απλό να προβλεφθεί ποια είναι τα σενάρια που κάνουν την κατάσταση αναστολής λειτουργίας αποδοτική ως προς την εξοικονόμηση ενέργειας. Στις προσεγγίσεις αυτού του τύπου θεωρούμε απλά μοντέλα των δικτύων με κύριο στόχο την συλλογή πληροφοριών σχετικά με την

ενέργεια που καταναλώνουν οι συσκευές αναλογικά με την κυκλοφορία που διαχειρίζονται και τον ποσοστό των κόμβων που έχουν τεθεί σε κατάσταση αναστολής λειτουργίας(sleep mode).Τα μοντέλα αυτά μας επιτρέπουν να προβλέψουμε πόση ενέργεια μπορεί να εξοικονομηθεί σε διαφορετικά σενάρια. Τα αποτελέσματα δείχνουν ότι κάνοντας χρήση του μοντέλου sleep mode μπορεί να επιτευχθεί σημαντική μείωση στην κατανάλωση ενέργειας. Ωστόσο εάν η κατανάλωσης της στατικής ενέργειας είναι κατά μία τάξη μεγέθους μικρότερη από το φορτίο αναλογικού στοιχείου, τότε η αναστολή λειτουργείας των συσκευών δεν αποτελεί συμφέρουσα κατάσταση.

5.2 Real-Time δρομολόγηση

Όπως γίνεται αντιληπτό η εξοικονόμηση ενέργειας των δικτύων αποτελεί μια νέα ανησυχία στους αρχιτέκτονες των δικτύων. Η άσκοπη σπατάλη των ενεργειακών πόρων έχουν σημαντική επίπτωση όχι μόνο στην οικολογία του περιβάλλοντος άλλα και στην οικονομία. Βέβαια όπως αναφέρθηκε και παραπάνω η αρχιτεκτονική των σημερινών δικτύων αποτελεί τροχοπέδη στην ανάπτυξη νέων εφαρμογών που απαιτούν μεγαλύτερη ταχύτητα και μετάδοση σε πραγματικό χρόνο. Για την αντιμετώπιση φαινομένων τέτοιων απαιτούνται μηχανισμοί που στηρίζονται σε μοντέλα και θα δρομολογούν την κίνηση του δικτύου με τέτοιο τρόπο που να λαμβάνονται υπόψιν διάφοροι παράγοντες, όπως η εξοικονόμηση ενέργειας αλλά και η προτεραιότητα των πακέτων.

Σημαντική ανησυχία για την ανάπτυξη τέτοιων μηχανισμών αποτέλεσε η υλοποίηση τους, καθώς η αποφάσεις για την επιλογή του καλύτερου μονοπατιού δρομολόγησης δεν μας επιτρέπει να γίνεται χειροκίνητα μιας και οι μεταβλητές που επηρεάζουν τους μηχανισμούς αυτούς αλλάζουν σε πολύ μικρά χρονικά διαστήματα. Συνεπώς για την ανάπτυξη μηχανισμών δρομολόγησης απαιτείται η χρήση λογισμικών που θα είναι έτσι προγραμματισμένα ώστε να μπορούν σε μηδενικό χρόνο να λαμβάνουν αποφάσεις για την δρομολόγηση των πακέτων και ταυτόχρονα να θέτουν σε

καταστολή (sleep mode) τις ανενεργές συσκευές. Η ανάπτυξη της τεχνολογίας Software defined networking (δικτύων που βασίζονται στα δίκτυα καθοριζόμενα από λογισμικό) συνέβαλλε σημαντικά στην ανάπτυξη της ιδέας αυτής.

Μέσω του πρωτοκόλλου OpenFlow παρέχεται η δυνατότητα επικοινωνίας των ελεγκτών με τις δικτυακές συσκευές δρομολόγησης. Μέσω του δίαυλου επικοινωνίας επιτρέπεται η ανταλλαγή μηνυμάτων διαχείρισης αλλά και συλλογής πληροφοριών , γεγονός που όπως θα αναλυθεί παρακάτω είναι απαραίτητο για την ανάπτυξη του μηχανισμού δρομολόγησης.

5.3 Λογισμικό Βελτιστοποίησης CPLEX

Όπως αναφέρθηκε προηγουμένως, η επιλογή της κατάλληλης δρομολόγησης της κίνησης των πακέτων, δεν μπορεί να γίνει χειροκίνητα από τον διαχειριστή του δικτύου για η συχνότητα της κίνησης και η ταχύτητα των πακέτων είναι τόσο μεγάλη που απαιτούνται αυτοματοποιημένες ενέργειες.

Για τον λόγο αυτόν οι διαχειριστές χρησιμοποιούν λογισμικά που υποστηρίζουν γλώσσα προγραμματισμού βελτιστοποίησης (OPL). Η γλώσσα προγραμματισμού OPL είναι μια γλώσσα που έχει σχεδιαστεί ειδικά για τη βελτιστοποίηση συνδυαστικών προβλημάτων. Ο σχεδιασμός του OPL παρακινήθηκε από την επιτυχή εφαρμογή γλωσσών προγραμματισμού, όπως AMPL και GAMS, στη μοντελοποίηση και επίλυση μαθηματικών προβλημάτων. Μέσω της γλώσσας OPL επιτρέπει στο χρήστη να μοντελοποιεί προβλήματα που παραδοσιακά εκφράζονται σε αλγεβρική γραφή με τη χρήση ισοδύναμων όρων υπολογιστή.

Με βάση την ιδέα της αξιοποίησης των χαρακτηριστικών τόσο στον μαθηματικό προγραμματισμό όσο και στην περιορισμένη προσέγγιση του προγραμματισμού στην επίλυση του προβλήματος στο χέρι, το OPL επιτρέπει στο χρήστη να διαμορφώσει το πρόβλημα σε μια υβριδική προσέγγιση. Το γεγονός αυτό, μαζί με άλλες νέες έννοιες,

όπως οι περιορισμοί ανώτερης τάξης και ο λογικός συνδυασμός βελτιώνει σημαντικά την αναγνωσιμότητα και την εκφραστικότητα του προβλήματος, αφαιρώντας ταυτόχρονα τυχόν περιττές λεπτομέρειες. Το OPL γεφυρώνει αποτελεσματικά το χάσμα μεταξύ μοντελοποίησης και επίλυσης προβλημάτων. Η γλώσσα OPL μέχρι στιγμής υποστηρίζεται μόνο από το λογισμικό IBM ILOG CPLEX Optimization Studio, το οποίο χρησιμοποιούμαι στην συνέχεια για την επίλυση των προβλημάτων.

Το λογισμικό IBM ILOG CPLEX Optimization Studio (συχνά αναφέρεται ως CPLEX) είναι ένα πακέτο λογισμικού βελτιστοποίησης. Το CPLEX Optimizer οφείλει την ονομασία του στην ονομασία της μεθόδου simplex, που εφαρμόζεται στη γλώσσα προγραμματισμού C, αν και σήμερα υποστηρίζει επίσης και άλλα είδη μαθηματικής βελτιστοποίησης και προσφέρει επιπλέον επιλογές εκτός από την χρήση της γλώσσας C. Το CPLEX αρχικά αναπτύχθηκε από τον Robert E. Bixby το 1988 από την εταιρία CPLEX Optimization Inc, η οποία εξαγοράστηκε από την ILOG το 1997. Στην συνέχεια η εταιρία ILOG εξαγοράστηκε από την IBM, τον Ιανουάριο του 2009. Το CPLEX συνεχίζει να αναπτύσσεται ενεργά από την εταιρία IBM.

Το IBM ILOG CPLEX Optimizer σχεδιάστηκε για να λύνει προβλήματα ακέραιου προγραμματισμού, μεγάλα προβλήματα γραμμικού προγραμματισμού με τη χρήση παραλλαγών της μεθόδου simplex ή της μεθόδου εμποδίων και εσωτερικών σημείων (Barrier and interior point methods), κυρτών και μη κυρτών συνόλου προβλημάτων τετραγωνικού προγραμματισμού, αλλά και κυρτών περιορισμένων τετραγωνικών προβλημάτων , τα οποία επιλύονται μέσω κωνικού προγραμματισμού δεύτερης τάξης, ή SOCP).

Το CPLEX Optimizer περιέχει ένα στρώμα που ονομάζεται Concert και δημιουργεί την συμβατότητα του λογισμικού με τις γλώσσες C ++, C # και Java. Παρέχεται επιπλέον μια διεπαφή σε γλώσσα Python που βασίζεται στη διασύνδεση της γλώσσας C. Επιπλέον, επιτρέπονται συνδέσεις στο Microsoft Excel και το MATLAB. Τέλος, παρέχεται ένα αυτόνομο διαδραστικό πρόγραμμα αποσφαλμάτωσης που έχει ως στόχο τον εντοπισμό σφαλμάτων. Το CPLEX Optimizer είναι προσβάσιμο μέσω

ανεξάρτητων συστημάτων μοντελοποίησης όπως τα AIMMS, AMPL, GAMS, MPL, OpenOpt, OptimJ και TOMLAB.

Το CPLEX προσφέρει επίσης λύσεις βελτιστοποίησης του δικτύου και στοχεύει σε μια ειδική κατηγορία των γραμμικών πρόβλημα που εμφανίζουν τις δομές του δικτύου. Τα προβλήματα αυτού του είδους θα μας απασχολήσουν στην συγκεκριμένη περίπτωση. Το CPLEX μπορεί να βελτιστοποιήσει τέτοια προβλήματα όπως τα συνήθη γραμμικά προγράμματα, αλλά αν το CPLEX μπορεί να εξαγάγει όλο το σύνολο ή ένα μέρος του προβλήματος ως ένα δίκτυο, τότε θα επιτευχθεί πιο αποτελεσματική βελτιστοποίηση του δικτύου σε αυτό το μέρος του προβλήματος και στην συνέχεια χρησιμοποιεί αυτή την μερική λύση που βρέθηκε για να κατασκευάσει ένα προχωρημένο σημείο εκκίνησης που θα συμβάλλει στην βελτιστοποίηση του υπολοίπου μέρους.

5.4 Δημιουργία Πληροφοριών

Για την βελτιστοποίηση του προβλήματος μας και την επίτευξη εξοικονόμησης ενέργειας απαραίτητη είναι η συλλογή πληροφοριών που όπως θα αναλυθεί παρακάτω θα χρησιμοποιηθούν ως δεδομένα. Ο αλγόριθμος που θα χρησιμοποιηθεί απαραίτητο είναι να αναγνωρίζει την τοπολογία του δικτύου αλλά και ορισμένες πληροφορίες σχετικά με την κατάσταση των δρομολογητών και την κίνηση του δικτύου. Αν και η τοπολογία του δικτύου αποτελεί κατά κύριο λόγο ένα στατικό στοιχείο, δηλαδή η χαρτογράφηση του δικτύου γίνεται συνήθως είτε μία φορά κατά την εκκίνηση του προγράμματος είτε ανανεώνεται σε μεγάλα χρονικά διαστήματα, οι στατιστικές πληροφορίες που απαιτούνται από τις δικτυακές συσκευές αποτελούν στοιχεία τα οποία μεταβάλλονται άμεσα.

Η κίνηση μέσα στο δίκτυο αποτελεί ένα στοιχείο το οποίο συνήθως μπορεί κάποιος να το παρακολουθήσει με δειγματοληψία σε τακτά χρονικά διαστήματα. Το γεγονός αυτό οφείλεται στο ότι οι χρήστες και οι δικτυακές συσκευές ανταλλάσσουν μηνύματα με μεγάλη συχνότητα και μεγάλους ρυθμούς. Επιπλέον σημαντικός παράγοντας που

επηρεάζει την κίνηση στο δίκτυο αποτελεί η καθυστέρηση των μηνυμάτων που οφείλεται σε παράγοντες όπως η συσσώρευση μηνυμάτων σε ουρές αναμονής αλλά και πολλούς άλλους που δεν θα αναφερθούν στο πλαίσιο της παρούσας διπλωματικής. Τέλος εξίσου σημαντικό γεγονός αποτελεί η διαδρομή των πακέτων κατά την αποστολή μηνυμάτων μεταξύ δύο κόμβων, καθώς δεν ακολουθείται πάντα η ίδια διαδρομή με αποτέλεσμα να μην υπάρχει κάποιο αναλογικό μοντέλο πρόβλεψης της κίνησης που επισκέπτεται κάθε συσκευή.

Όπως γίνεται άμεσα αντιληπτό, απαιτείται η συνεχής ανανέωση των στατικών πληροφοριών που χρησιμοποιεί το CPLEX για να βελτιστοποιήσουμε την δρομολόγηση και να πετύχουμε το επιθυμητό αποτέλεσμα.

5.4.1 Τοπολογία Δικτύου

Για να μπορέσει να λειτουργήσει ο αλγόριθμος βελτιστοποίησης στο CPLEX, απαιτείται ο καθορισμός της τοπολογίας του δικτύου, έτσι ώστε να ληφθεί υπόψιν ο τρόπος διασύνδεσης του δικτύου για τη εύρεση της ενναλακτικής δρομολόγησης των πακέτων σε περίπτωση που αυτό απαιτηθεί από τον λογισμικό. Πιο συγκεκριμένα για την μείωση της κατανάλωσης ενέργειας, μπορεί να προκύψει από το CPLEX σαν λύση, η αλλαγή της κατάστασης ενέργειας ενός δρομολογητή (είσοδος του σε κατάσταση sleep mode), και η εύρεση εναλλακτικής διαδρομής για τα νέα πακέτα. Όποτε η χαρτογράφηση του δικτύου αποτελεί πληροφορία απαραίτητη για το CPLEX.

Αν και θα μπορούσε αυτή η πληροφορία να δοθεί χειροκίνητα και στατικά στο λογισμικό κάτι τέτοιο αποφεύγεται. Ο σημαντικότερος λόγος είναι ότι θα πρέπει η λειτουργία του να υποστηρίζει την καθολικότητα, δηλαδή να μπορεί να χρησιμοποιείται γενικευμένα από όλους τους χρήστες και τα δίκτυα και να μην αποτελεί μία εξειδικευμένη περίπτωση. Επιπλέον η δομή του δικτύου αλλάζει δυναμικά καθώς οι δικτυακές συσκευές και τα τερματικά ενεργοποιούνται και απενεργοποιούνται συνεχώς με αποτέλεσμα να απαιτείται δυναμική ανανέωση των πληροφοριών.

Για την δημιουργία των πληροφοριών αυτών χρησιμοποιούμαι τον ελεγκτή Beacon μέσω του οποίου δημιουργούνται μηνύματα προς όλους τους δρομολογητές και επιστρέφουν απαντήσεις που αναφέρουν της ενεργές τους θύρες και τις γειτονικές συσκευές με τις οποίες συνδέονται.

Πιο συγκεκριμένα, όπως έχει ήδη αναλυθεί και σε προηγούμενο κεφάλαιο, ο ελεγκτής επικοινωνεί άμεσα με τους δρομολογητές και ανταλλάσσουν μηνύματα μέσω ασφαλούς καναλιού. Ο ελεγκτής Beacon υποστηρίζει την δημιουργία στατιστικών μηνυμάτων και μηνυμάτων τοπολογίας. Μέσω του thread Topology, αποστέλλονται μηνύματα στις δικτυακές συσκευές ώστε να εντοπιστούν οι σύνδεσμοι μεταξύ συνδεδεμένων δρομολογητών OpenFlow. Μέσω της διεπαφής του ITopology επιτρέπεται η ανάκτηση ενός καταλόγου αυτών των συνδέσμων, και την καταγραφή των γεγονότων όταν ένας σύνδεσμος προστεθεί ή αφαιρεθεί.

```
package net.beaconcontroller.topology;
import java.util.Map;
public interface ITopology {
    public boolean isInternal(SwitchPortTuple idPort);
    public Map<LinkTuple, Long> getLinks();
}
```

Σχήμα 5-1 Η ITopology κλάση

Μέσω της συνάρτησης *Map<LinkTuple, Long> getLinks()* καλείται η κλάση `LinkTuple` η οποία με την σειρά της καλεί τις απαραίτητες κλάσεις και στο τέλος επιστρέφει και δημιουργείται μήνυμα τύπου `OFStatisticsRequest` που στην συνέχεια αποστέλλεται στους δρομολογητές που καθορίζεται από το μήνυμα. Στην συνέχεια οι δρομολογητές αποστέλλουν μηνύματα αναγνωριστικά σε όλους τους γειτονικούς δρομολογητές και επιπλέον έχοντας συλλέξει τις πληροφορίες που απαιτούνται αποστέλλουν την απάντηση `OFStatisticsReply`, την οποία λαμβάνει ο ελεγκτής και επιστρέφει την διεύθυνση των δρομολογητών και τις πόρτες μέσω των οποίων συνδέονται οι δρομολογητές.

Παρακάτω συνοψίζεται η κλάση `OFStatisticsRequest` που αναφέρεται στην δημιουργία μηνυμάτων τύπου OpenFlow που αποστέλλονται στους δρομολογητές και επιστρέφουν στατιστικά δεδομένα.

```
package org.openflow.protocol;

import java.util.Collections;

import org.openflow.protocol.statistics.OFStatistics;
import org.openflow.util.U16;
public class OFStatisticsRequest extends OFStatisticsMessageBase {
    public OFStatisticsRequest() {
        super();
        this.type = OFType.STATS_REQUEST;
        this.length = U16.t(OFStatisticsMessageBase.MINIMUM_LENGTH);
    }
    public OFStatistics getStatistics() {
        if (statistics == null)
            return null;
        else if (statistics.size() == 0)
            return null;
        else
            return statistics.get(0);
    }
    public OFStatisticsRequest setStatistics(OFStatistics statistics) {
        this.statistics = Collections.singletonList(statistics);
        return this;
    }
}
```

Σχήμα 5-2 Δημιουργία αιτήματος OFStatisticsRequest

Στα πλαίσια της παρούσας διπλωματικής εργασίας, δημιουργούμε από τον ελεγκτή μηνύματα τύπου `OFStatisticsRequest` όπως αναφέρθηκε προηγουμένως, στους δρομολογητές και στην συνέχεια τοποθετούμε τα αποτελέσματα σε ένα αρχείο δεδομένων, topology.dat, η μορφή του οποίου θα αναλυθεί στην συνέχεια. Στο ακόλουθο σχήμα παρουσιάζετε η διαδικασία αποθήκευσης των στατιστικών δεδομένων της τοπολογίας του δικτύου.

```
ScheduledExecutorService ses = Executors.newSingleThreadScheduledExecutor();

ses.scheduleAtFixedRate(new Runnable() {
    @Override
    public void run() {

BufferedWriter writer = null;
        try {
            writer = new BufferedWriter(new FileWriter("topology.dat"));
            List<String> columnName = new ArrayList<String>();
```

```
            columnName = new ArrayList<String>();
            columnName.add("Src Id");
            columnName.add("Src Port");
            columnName.add("Dst Id");
            columnName.add("Dst Port");
            columnName.add("Capacity");
            columnName.add("Cost");
            int n = columnName.size();
            for(int i = 0; i < n ; i++){
              writer.write(columnName.get( i ) );
              writer.write("                    ");
            }
            writer.newLine();
            for (LinkTuple lt : topology.getLinks().keySet()) {
                List<String> row = new ArrayList<String>();
                row.add(HexString.toHexString(lt.getSrc().getSw().getId()));
                row.add(lt.getSrc().getPort().toString());
                row.add(HexString.toHexString(lt.getDst().getSw().getId()));
                row.add(lt.getDst().getPort().toString());
                row.add(findBandwidth(lt.getSrc().getSw()  ,  lt.getSrc().getPort()  ) +
"Mbps");
                row.add("10");
                for(int i = 0; i < n ; i++){
                    writer.write( row.get( i ) );
                    writer.write("          ");
                  }
                writer.newLine();
            }

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                // Close the writer regardless of what happens...
                writer.close();
            } catch (Exception e) {
            }
        }

    }
}, 0, 10, TimeUnit.MINUTES);

// when finished
ses.shutdown();
```

Σχήμα 5-3 Αποθήκευση στοιχείων τοπολογίας του δικτύου

Τα αρχείο αυτό αποθηκεύεται σε κάποιο συγκεκριμένο φάκελο που ορίζεται από τον προγραμματιστή (στην περίπτωση μας στην επιφάνεια εργασίας του υπολογιστή) και μπορεί να έχει πρόσβαση και το CPLEX ώστε να μπορεί να εισαγάγει το αρχείο αυτό.

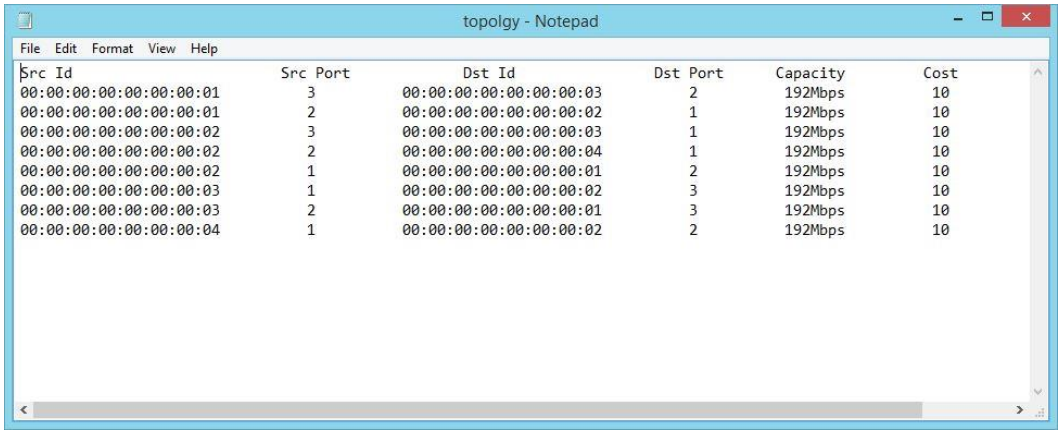
Το αρχείο που αποθηκεύει τα στοιχεία της τοπολογίας ανανεώνεται αυτόματα ανά δέκα λεπτά με αποτέλεσμα την σταδιακή ενημέρωση για τυχόν αλλαγές στα στοιχεία του αρχείου, που ενδεχομένως να προέκυψαν.

Τα στοιχεία που συγκεντρώθηκαν από το `OFStatisticsReply` , στην συνέχεια τοποθετούνται σε τέσσερις στήλες πίνακα ώστε να μπορεί το αρχείο να είναι εύκολα διαχειρίσιμο και ευανάγνωστο και από τον CPLEX και από τον προγραμματιστή καθώς σε περίπτωση σφάλματος μπορεί να βοηθήσει στην διαδικασία αποσφαλμάτωσης. Στην συνέχεια απεικονίζεται ο κώδικας μέσω του οποίου τα δεδομένα εκχωρούνται στις τέσσερις στήλες

```
row.add(HexString.toHexString(lt.getSrc().getSw().getId()));
                row.add(lt.getSrc().getPort().toString());
                row.add(HexString.toHexString(lt.getDst().getSw().getId()));
                row.add(lt.getDst().getPort().toString());
                row.add(findBandwidth(lt.getSrc().getSw() , lt.getSrc().getPort() ) +
"Mbps");
                row.add("10");
```

Οι στήλες αυτές περιλαμβάνουν ,με την σειρά που αναλύονται, την διεύθυνση του δρομολογητή που έστειλε το αναγνωριστικό μήνυμα (όπως αναφέρθηκε προηγουμένως) , την θύρα του δρομολογητή-αποστολέα μέσω της οποίας έγινε η σύνδεση, τη διεύθυνση του δρομολογητή-λήπτη και στην τελευταία στήλη περιλαμβάνεται η θύρα του δρομολογητή-λήπτη.

Στην παρακάτω εικόνα απεικονίζεται το περιεχόμενο του αρχείου topology.dat που αναφέρεται στην τοπολογία που αναπτύχθηκε στο mininet όπως αναλύθηκε και στο κεφάλαιο 4.



Σχήμα 5-4 Topology.dat

Src ID: Στο πεδίο αυτό εγγράφονται η διεύθυνση IPv6 των δρομολογητών από τους οποίους στάλθηκε μήνυμα τύπου OpenFlow στους γειτονικούς δρομολογητές.

Src Port: Στο πεδίο αυτό εγγράφονται η πόρτα των δρομολογητών από την οποία αποστέλλεται το μήνυμα τύπου OpenFlow στους γειτονικούς δρομολογητές.

Dst ID: Στο πεδίο αυτό εγγράφονται η διεύθυνση IPv6 των δρομολογητών οι οποίοι λαμβάνουν το μήνυμα τύπου OpenFlow που στάλθηκε από τους γειτονικούς δρομολογητές.

Dst Port: Στο πεδίο αυτό εγγράφονται η πόρτα των δρομολογητών οι οποίοι λαμβάνουν το μήνυμα τύπου OpenFlow που στάλθηκε από τους γειτονικούς δρομολογητές.

Capacity: Η χωρητικότητα της γραμμής του δικτύου.

Cost: Το ενεργειακό κόστος του κάθε κόμβου. Απαραίτητο πεδίο στον υπολογισμό της βέλτιστης λύσης που υπολογίζει το λογισμικό CPLEX.

5.4.2 Δημιουργία στατιστικών στοιχείων

Για να μπορέσουμε να αποφανθούμε για την βέλτιστη δρομολόγηση με σκοπό την ελαχιστοποίηση της κατανάλωσης ενέργειας, σημαντικό χαρακτηριστικό αποτελεί ο φόρτος εργασίας κάθε δικτυακής συσκευής. Ο αλγόριθμος του CPLEX που χρησιμοποιείται εφαρμόζει κατάλληλα στατιστικά στοιχεία βασισμένα στο μοντέλο που θα αναλυθεί στην συνέχεια με σκοπό να υπολογισθεί εάν η κάθε συσκευή μπορεί να ενταχθεί σε κατάσταση sleep mode.

Μέσω του ελεγκτή Beacon δημιουργούμε συνεχώς αιτήματα τύπου `OFStatisticsRequest` προς όλους τους δρομολογητές με στόχο την δυναμική αποτύπωση της κατάστασης κίνησης των συσκευών. Οι δικτυακές συσκευές στην συνέχεια αποστέλλουν μηνύματα τύπου `OFStatisticsReply` που περιέχουν τις αντίστοιχες απαντήσεις.

Ο ελεγκτής Beacon δημιουργεί αιτήσεις αυτού του τύπου ανά πέντε λεπτά ώστε να μπορεί να αποτυπώνεται συνεχώς η κατάσταση του δικτύου, γεγονός που θα βελτιώσει την κατάσταση της αναδρομολόγησης.

Όπως αναφέρθηκε και στο 2ο κεφάλαιο η τεχνολογία OpenFlow επιτρέπει στις δικτυακές συσκευές δημιουργία πολλών, τύπου στατιστικών, μηνυμάτων. Στην περίπτωση που εξετάζουμε επιλέξαμε να συλλέγουμε ορισμένα στοιχεία. Πιο συγκεκριμένα οι δρομολογητές στις απαντήσεις τύπου `OFStatisticsReply` που αναφέρθηκαν προηγουμένως συλλέγουν πληροφορίες που αφορούν τα μηνύματα που διαχειρίζονται την στιγμή εκείνη στο δίκτυο. Έτσι επιλέγουμε να αποθηκεύσουμε πληροφορίες όπως την θύρα που εισήχθη το πακέτο, την θύρα από την οποία θα εξέλθει, την διεύθυνση IPv4 του αποστολέα και του παραλήπτη, τα πακέτα και τα Byte που ανταλλάχθηκαν αλλά και το χρόνο επεξεργασίας και μεταφοράς των πακέτων.

Στην συνέχεια παρουσιάζουμε τον κώδικα που εκτελείται στον ελεγκτή Beacon, μέσω του οποίου δημιουργούνται τα αιτήματα προς τους δρομολογητές και αποθηκεύονται στην συνέχεια σε αρχείο τύπου .dat . Ο ελεγκτής είναι έτσι προγραμματισμένος ώστε τα μηνύματα που λαμβάνει από κάθε δρομολογητή αποθηκεύονται σε χωριστό αρχείο με όνομα την διεύθυνση IPv6 της δικτυακής συσκευής.

```
ScheduledExecutorService ses = Executors.newSingleThreadScheduledExecutor();


ses.scheduleAtFixedRate(new Runnable() {
    @Override
    public void run() {


/* Update flow file*/
        System.out.println("---->switchId="+switchId);
        String sw_id =switchId.replace(":", "_");
        System.out.println("---->switchId="+sw_id);
        FileWriter writer = new FileWriter(sw_id+".dat");
        BufferedWriter bufferWritter = new BufferedWriter(writer);
        for(int i=0; i<data.size(); i++){


        bufferWritter.write(U16.f(data.get(i).getMatch().getInputPort())+" ");

bufferWritter.write(HexString.toHexString(data.get(i).getMatch().getDataLayerSource())
+" ");
```

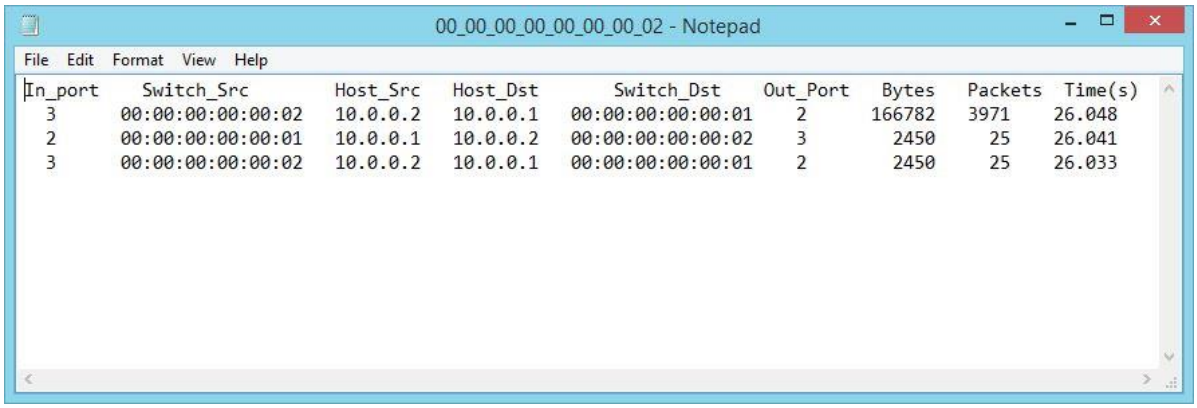
```
bufferWritter.write(IPv4.fromIPv4Address(data.get(i).getMatch().getNetworkSource())+"
");

bufferWritter.write(IPv4.fromIPv4Address(data.get(i).getMatch().getNetworkDestination(
))+ " ");
        StringBuffer outPorts = new StringBuffer();
        for (OFAction action : data.get(i).getActions()) {
            if (action instanceof OFActionOutput) {
                OFActionOutput ao = (OFActionOutput)action;
                if (outPorts.length() > 0)
                    outPorts.append(" ");
                outPorts.append(U16.f(ao.getPort()));
            }
        }


bufferWritter.write(HexString.toHexString(data.get(i).getMatch().getDataLayerDestinati
on())+" ");

        bufferWritter.write(outPorts.toString()+" ");
        bufferWritter.write(U64.f(data.get(i).getByteCount())+" ");
        bufferWritter.write(U64.f(data.get(i).getPacketCount())+" ");
        double duration = (double)U32.f(data.get(i).getDurationSeconds())+((double)
data.get(i).getDurationNanoseconds()) / 1000000000d;
        bufferWritter.write(Double.toString(duration));
        bufferWritter.newLine();
        }
        bufferWritter.close();


        return view;
    }

     }


}, 0, 10, TimeUnit.MINUTES);

// when finished
ses.shutdown();
```

Σχήμα 5-5 flows.dat

Όπω φαίνεται και απο το σχήμα τα στατιστικά στοιχεία εγγράφονται σε μορφή πίνακα,γεγονός που κάνει το αρχείο ευανάγνωστο. Επιπλέον, όπως ειπώθηκε και προηγουμένως τα αρχεία έχουν ως όνομα την διεύθυνση των δρομολογητών.Αυτό συμβαίνει απο την εντολή `sw_id+".dat"` .

Η αποθήκευση των στατιστικών σε διαφορετικά αρχεία διευκολύνει τον μηχανικό του δικτύου αλλά και το λογισμικό να αναγνωρίζει της κίνηση του κάθε δρομολογητή χωριστά. Έτσι, μπορεί να αντιλαμβάνεται τυχόν δυσλειτουργίες που πιθανών να έχουν προκύψει αλλά και να συμβάλλει στην αποσφαλμάτωση του δικτύου εφόσον κάτι τέτοιο κριθεί απαραίτητο.

Στην παρακάτω εικόνα απεικονίζεται το περιεχόμενο του αρχείου 00_00_00_00_00_01.dat που αναφέρεται στα στατιστικά στοιχεία (flows) που συλλέγονται σε μία χρονική στιγμή απο τον δρομολογητή με την αντίστοιχη διεύθυνση.



Σχήμα 5-6 00_00_00_00_00_01.dat

5.4.3 Βέλτιστη δρομολόγηση μέσω CPLEX

Τα στατιστικά στοιχεία και η τοπολογία του δικτύου μας διαβάζονται και καταγράφονται με συγκεκριμένη δομή σε ένα αρχείο τύπου .dat .Στην παρούσα εργασία το αρχείο αυτό έχει την ονομασία green_stoixeia.dat . Το CPLEX στην συνέχεια είναι έτσι προγραμματισμένο ώστε να αντλεί όλα τα απαιτούμενα στοιχεία από το αρχείο.

Στην συνέχεια απεικονίζεται η μορφή του αρχείου green_stoixeia.dat

```
// Set of router nodes
 V = {
// nodeid, baseline consumption
  <  1,    0 >, //SW1
  <  2,    0 >, //SW2
  <  3,    0 >, //SW3
  <  4,    0 >, //SW4
};


// Set of router edges
// from, to, alpha, beta, C, link_id
E = {
  <  <  1,    0 >,    <  2,    0 >,  18.75,    22.5,   26.25,  30, 10, 1 >,
  <  <  2,    0 >,    <  1,    0 >,  18.75,    22.5,   26.25,  30, 10, 1 >,
  <  <  1,    0 >,    <  3,    0 >,  18.75,    22.5,   26.25,  30, 20, 2 >,
  <  <  3,    0 >,    <  1,    0 >,  18.75,    22.5,   26.25,  30, 20, 2 >,
  <  <  2,    0 >,    <  3,    0 >,  18.75,    22.5,   26.25,  30, 20, 3 >,
  <  <  3,    0 >,    <  2,    0 >,  18.75,    22.5,   26.25,  30, 20, 3 >,
  <  <  2,    0 >,    <  4,    0 >,  18.75,    22.5,   26.25,  30, 20, 4 >,
  <  <  4,    0 >,    <  2,    0 >,  18.75,    22.5,   26.25,  30, 20, 4 >,
  <  <  3,    0 >,    <  4,    0 >,  18.75,    22.5,   26.25,  30, 10, 5 >,
  <  <  4,    0 >,    <  3,    0 >,  18.75,    22.5,   26.25,  30, 10, 5 >,
};



// Set of flows
F = {
// from, to, rate(se bit per sec)
  <  <  1,    0 >,    <  2,    0 >,  0,13742394 >,
  <  <  1,    0 >,    <  3,    0 >,  0,12882111 >,
  <  <  1,    0 >,    <  4,    0 >,  0,15000484 >,
  <  <  2,    0 >,    <  1,    0 >,  0,13742394 >,
  <  <  2,    0 >,    <  3,    0 >,  0,13324865 >,
  <  <  2,    0 >,    <  4,    0 >,  0,16862925 >,
  <  <  3,    0 >,    <  1,    0 >,  0,12882111 >,
  <  <  3,    0 >,    <  2,    0 >,  0,13324865 >,
  <  <  3,    0 >,    <  4,    0 >,  0,15492165 >,
  <  <  4,    0 >,    <  1,    0 >,  0,15000484 >,
  <  <  4,    0 >,    <  2,    0 >,  0,16862925 >,
  <  <  4,    0 >,    <  3,    0 >,  0,15492165 >,

};
```

Σχήμα 5-7 data_stoixeia.dat

Όπως να αναλυθεί και παρακάτω, για τον υπολογισμό της κατάλληλης διαδρομής ορίζονται τρία ορίσματα το V , E, F . Η μεταβλητή V περιλαμβάνει τον αριθμό και την ονομασία όλων των δρομολογητών του δικτύου.

Στο όρισμα E περιλαμβάνονται όλες οι υπάρχουσες συνδέσεις μεταξύ των δρομολογητών και πέντε αριθμητικές ποσότητες που υποδεικνύουν κάποια ενεργειακό κόστος του δρομολογητή σε διαφορετικές καταστάσεις ανάλογα με τον βαθμό χρησιμοποίησης της γραμμής. Η πρώτη ποσότητα αναφέρεται στο 0% δηλαδή οι δρομολογητές της σύνδεσης βρίσκονται σε κατάσταση αδράνειας. Στην περίπτωση αυτή οι δρομολογητές καταναλώνουν ένα σημαντικό ποσοστό ενέργειας γεγονός που επιθυμούμε την ελαχιστοποίηση της κατάστασης αυτής και την είσοδο των ανενεργών συσκευών σε κατάσταση sleep mode. Τέλος η τελευταία κατάσταση, αναφέρεται στην περίπτωση που ο βαθμός χρησιμοποίησης της γραμμής είναι σε ποσοστό 100%. Τέλος

οι υπόλοιπες καταστάσεις αναφέρονται σε ενδιάμεσες. Τα στοιχεία που περιέχουν όλες τις υπάρχουσες συνδέσεις (E) εισάγονται από το αρχείο δεδομένων topology.dat

Η μεταβλητή F περιλαμβάνει τα στατιστικά της κίνησης του δικτύου. Πιο συγκεκριμένα περιλαμβάνει το ρυθμό μετάδοσης για όλες τις περιπτώσεις αποστολής μηνυμάτων μεταξύ των χρηστών. Στην περίπτωση μας έχουμε 4 χρήστες με αποτέλεσμα να εμφανίζονται 12 διαφορετικές περιπτώσεις. Όπως αναλύθηκε και παραπάνω μέσω του ελεγκτή δημιουργούμε αρχεία στατιστικών δεδομένων που περιέχουν την κίνηση των δρομολογητών αναγράφοντας τον προορισμό και την κατεύθυνση του πακέτου αλλά και τα πακέτα και τον χρόνο που κάνουν για να μεταδοθούν. Έτσι με την δημιουργία ενός αυτοματοποιημένος προγράμματος κατασκευάστηκε ο ρυθμός που απαιτεί το CPLEX σαν δεδομένο για όλες τις δυνατές διαδρομές. Το γεγονός ότι τα στατιστικά στοιχεία ανανεώνονται ανά τακτά χρονικά διαστήματα έχει ως αποτέλεσμα τα στοιχεία του ρυθμού μετάδοσης στο CPLEX να βασίζονται σε ρεαλιστικά αποτελέσματα.

Στην συνέχεια υπολογίζεται η ελάχιστη δυνατή διαδρομή με στόχο την ελαχιστοποίηση της κατανάλωσης ενέργειας σύμφωνα με το παρακάτω αλγοριθμικό μοντέλο.

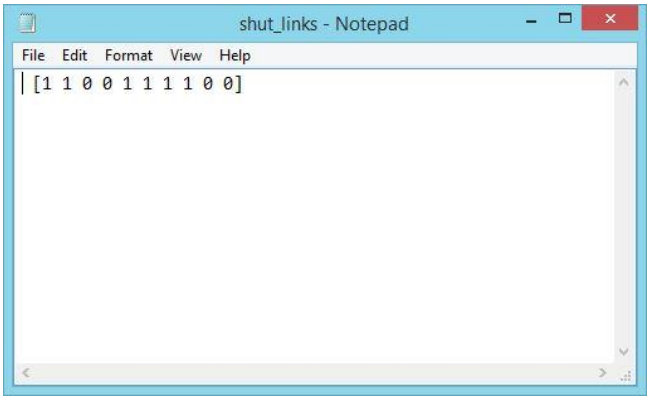
```
dexpr float TotalEnergy = sum (v in V) v.B * Y[v] + sum (e in E) (
S1[e]*e.alpha + S2[e]*e.beta +S3[e]*e.gama +S4[e]*e.delta);

minimize TotalEnergy;
```

Σχήμα 5-8 Αλγόριθμος ελαχιστοποίησης της κατανάλωσης ενέργειας

Οι μεταβλητές Y[V],Z[E],S1[E],S2[E],S3[E],S4[E], παίρνουν τιμές 0 και 1 με αποτέλεσμα να λειτουργούν ως διακόπτες ή μεταβλητές αποφάσεων. Επιπλέον οι τιμές της μεταβλητής V όπως αναφέρθηκε και προηγουμένως αποτελεί τους κόμβους του δικτύου ενώ η μεταβλητή E περιλαμβάνει πίνακα με πληροφορίες σχετικά με την

ενεργειακή κατανάλωση του δικτύου από τερματικό σε τερματικό ανάλογα τον βαθμό χρησιμότητας της γραμμής.

Στην συνέχεια αφού το λογισμικό κάνει τους απαραίτητους υπολογισμούς, εκτυπώνει σε ένα αρχείο τύπου txt με όνομα shut_links.txt ένα πίνακα-γραμμή που περιέχει τόσες στήλες όσες και οι συνδέσεις μεταξύ των δρομολογητών. Το πεδίο ορισμού των τιμών του πίνακα είναι το μοναδιαίο σύστημα. Όταν ένα κελί έχει την τιμή 1 σημαίνει πως η σύνδεση αυτή θα παραμείνει ανοιχτή, ενώ αντιθέτως αν έχει την τιμή 0 θα πρέπει να τερματιστεί η λειτουργεία της σύνδεσης.



Σχήμα 5-9 Shut_links.txt

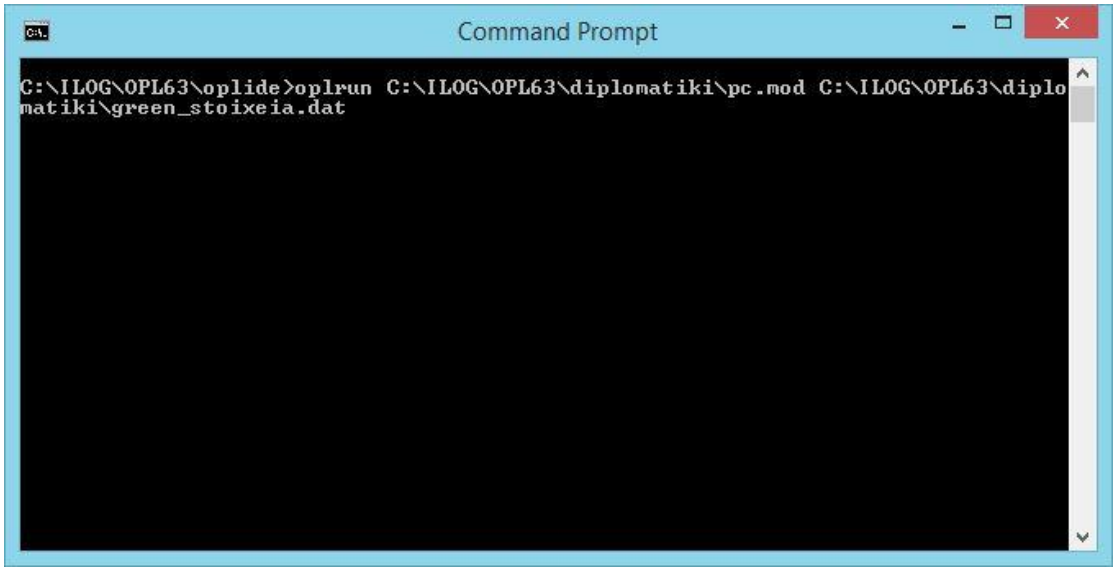
Τα στοιχεία του πίνακα αναφέρονται στις συνδέσεις των δικτυακών συσκευών όπως αυτές αναγράφονται στο αρχείο green_stoixeia.dat στο πεδίο E , αλλά και στο topology.dat . Αυτό έχει ως συνέπεια να γνωρίζει και ο προγραμματιστής του δικτύου αλλά και ο ελεγκτής πως το κάθε στοιχείο του πίνακα αναφέρεται σε συγκεκριμένη διασύνδεση και δεν αλλάζει δυναμικά.

Στην συνέχεια τα αποτελέσματα αυτά πρέπει να αναλυθούν από τον controller ώστε να ενημερώσει τις αντίστοιχες συσκευές να απενεργοποιήσουν την κατάλληλη θύρα και κατά συνέπεια την σύνδεση μεταξύ τους.

5.4.4 *Απενεργοποίηση διασυνδέσεων μέσω του ελεγκτή*

Ο ελεγκτής είναι προγραμματισμένος να εκτελεί κάθε πέντε λεπτά το λογισμικό CPLEX ώστε να εκτελεί τους απαιτούμενους υπολογισμούς. Η εντολή μέσω της οποίας πραγματοποιείται αυτό το γεγονός είναι η κάτωθι:

- `oplrun [options] model-file [data-file ...]`



Σχήμα 5-10 Εκτέλεση του CPLEX μέσω CMD

Η διαδικασία αυτή γίνεται με την χρήση χρονοπρογραμματισμένης εκτέλεσης, μία δυνατότητα που παρέχει η java μέσω της κλάσης timer(). Στην συνέχεια εκτελείται μέσω του ελεγκτή η εντολή που αναφέρθηκε προηγουμένως καθώς η java επιτρέπεται την εκτέλεση εντολών μέσω CMD. Παρακάτω παρατίθεται ο κώδικας μέσω του οποίου πραγματοποιείται το γεγονός αυτό.

```
try {
    // Execute command
    String command = "cmd /c start cmd.exe";
    Process child = Runtime.getRuntime().exec(command);

    // Get output stream to write from it
    OutputStream out = child.getOutputStream();

          out.write("cd ../../ILOG/OPL63/oplide /r/n".getBytes());

           out.flush();
           out.write("oplrun \\ILOG\\OPL63\\diplomatiki\\pc.mod
          \\ILOG\\PL63\\iplomatiki\\reen_stoixeia.dat  /r/0n".getBytes());
           out.close();

} catch (IOException e) {

}
```

Σχήμα 5-11 Εκτέλεση εντολής σε java μέσω Command Prompt

Κατά την εκτέλεση της εντολής το CPLEX εκτελείται και δημιουργεί το αρχείο με τα αποτελέσματα σε συγκεκριμένο χώρο του υπολογιστή, που προβλέπεται από τον προγραμματιστή. Ο ελεγκτής κάνει χρήση μιας μεταβλητής τύπου Boolean και δεν συνεχίζει την εκτέλεση του υπολοίπου κώδικα εως ότου δημιουργηθεί το απαραίτητο αρχείο, το shut_links.txt .

Ο Beacon πρέπει να αναλύσει τα αποτελέσματα του CPLEX και να αποστείλει OpenFlow μηνύματα στους δρομολογητές. Για τον σκοπό αυτό διαβάζει κάθε σειρά ξεχωριστά του αρχείο topology.txt και αποθηκεύει σε κάποιον πίνακα την διεύθυνση των δρομολογητών της κάθε διασύνδεσης και τις αντίστοιχες θύρες μέσω των οποίων γίνεται. Στην συνέχεια διαβάζει το πρώτο στοιχείο του shut_links.txt και ανάλογα με την τιμή του κελιού, αποστέλλει στους δρομολογητές που υπάρχουν στον πίνακα την αντίστοιχη εντολή, όπως αυτή θα αναλυθεί στην συνέχεια. Μόλις ολοκληρωθεί η ενέργεια αυτή ο ελεγκτής διαβάζει την επόμενη γραμμή του αρχείου topology.txt που περιέχει το επόμενο link. Η διαδικασία που αναφέρθηκε προηγουμένως επαναλαμβάνεται μέχρι να ολοκληρωθεί για όλες τις υπάρχουσες συνδέσεις.

Στην συνέχεια ο ελεγκτής στέλνει στους δρομολογητές τα αντίστοιχα μηνύματα. Τα μηνύματα αυτά όπως είναι αναμενόμενο πρέπει να έχουν συγκεκριμένη δομή για να μπορούν να αναγνωριστούν οι πληροφορίες από τις συσκευές. Για τον λόγο αυτό το πρωτόκολλο OpenFlow έχει ορίσει συγκεκριμένη δομή μηνυμάτων που αποστέλλονται και λαμβάνονται από τις δικτυακές συσκευές και τον ελεγκτή. Στην συγκεκριμένη περίπτωση ο ελεγκτής χρησιμοποιεί μηνύματα τύπου OFPT_PORT_MOD για να τροποποιήσει την συμπεριφορά μιας συγκεκριμένης θύρας. Παρακάτω παρατίθεται η δομή του μηνύματος, το οποίο χρησιμοποιήθηκε και στον ελεγκτή Beacon.

```
/* Modify behavior of the physical port */
struct ofp_port_mod {
struct ofp_header header;
uint32_t port_no;
uint8_t pad[4];
uint8_t hw_addr[OFP_ETH_ALEN]; /* The hardware address is not
configurable. This is used to
sanity-check the request, so it must
be the same as returned in an
ofp_port struct. */
```

```
uint8_t pad2[2]; /* Pad to 64 bits. */
uint32_t config; /* Bitmap of OFPPC_* flags. */
uint32_t mask; /* Bitmap of OFPPC_* flags to be changed. */
/* Port mod property list - 0 or more properties */
struct ofp_port_mod_prop_header properties[0];
};
OFP_ASSERT(sizeof(struct ofp_port_mod) == 32);
```

Σχήμα 5-12 μήνυμα τύπου OFPT_PORT_MOD

Το πεδίο config περιγράφει ρυθμίσεις διαχείρισης των θυρών και έχει την ακόλουθη δομή

```
/* Flags to indicate behavior of the physical port. These flags are
* used in ofp_port to describe the current configuration. They are
* used in the ofp_port_mod message to configure the port's behavior.
*/
enum ofp_port_config {
OFPPC_PORT_DOWN = 1 << 0, /* Port is administratively down. */
OFPPC_NO_RECV = 1 << 2, /* Drop all packets received by port. */
OFPPC_NO_FWD = 1 << 5, /* Drop packets forwarded to port. */
OFPPC_NO_PACKET_IN = 1 << 6 /* Do not send packet-in msgs for port. */
                                    };
```

Σχήμα 5-13 Μήνυμα τύπου ofp_port_config

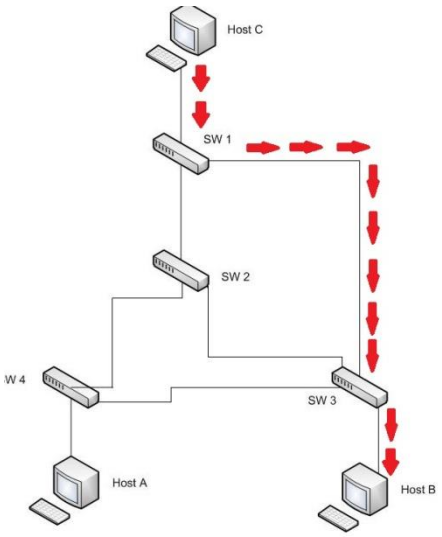
Το bit OFPPC_PORT_DOWN υποδεικνύει ότι η θύρα είναι κλειστή και δεν πρέπει να χρησιμοποιείται. Όπως γίνεται αντιληπτό η τιμή αυτή χρησιμοποιείται σε περίπτωση που τα αποτελέσματα που προέκυψαν από το CPLEX απαιτούν το κλείσιμο κάποιας συγκεκριμένης θύρας μια συσκευής. Αντίθετα για την ενεργοποίηση της θύρας χρησιμοποιείται η τιμή 1.

Το πεδίο mask χρησιμοποιείται για την επιλογή των bits στο πεδίο config τα οποία θα τροποποιηθούν.

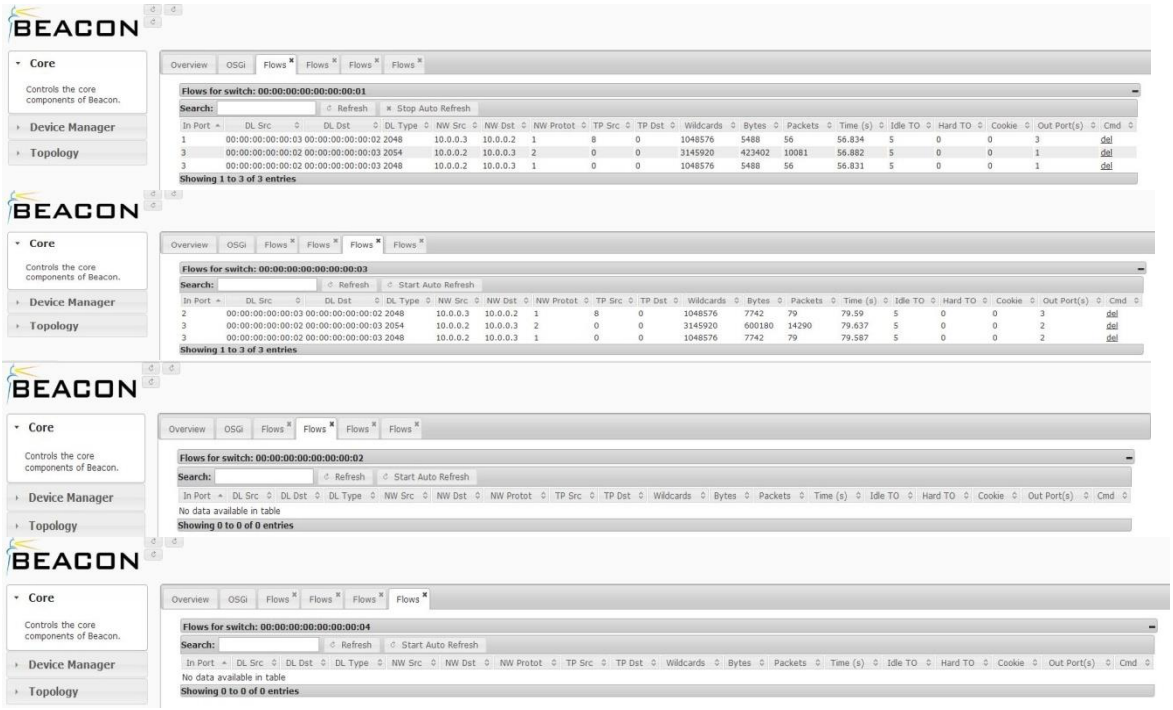
Κατά την ολοκλήρωση της αποστολής των μηνυμάτων οι συσκευές τροποποιούν κατάλληλα την κατάσταση των θυρών τους και η κίνηση των δικτύου αναδρομολογείται ανάλογα την υφιστάμενη κατάσταση του δικτύου. Στην συνέχεια οι ελεγκτές συλλέγουν νέα στατιστικά στοιχεία από τις συσκευές και με την πάροδο του

χρόνου, εκτελείται εκ νέου, όπως προαναφέρθηκε, το λογισμικό CPLEX και ανακυκλώνεται η ίδια διαδικασία που αναλύθηκε.

Στις παρακάτω εικόνες παριστάνεται μία απεικόνιση της διαδικασίας αυτής. Στην συγκεκριμένη περίπτωση δημιουργείται κίνηση από τον υπολογιστή C στον υπολογιστή B. Όπως φαίνεται και από την παρακάτω εικόνα η κίνηση πηγαίνει από την διαδρομή μεταξύ του δρομολογητή S1 και S3 καθώς από τον δρομολογητή S2 δεν περνάει κίνηση ενώ όλη η κίνηση πηγαίνει από τους δρομολογητές που αναφέρθηκαν προηγουμένως.



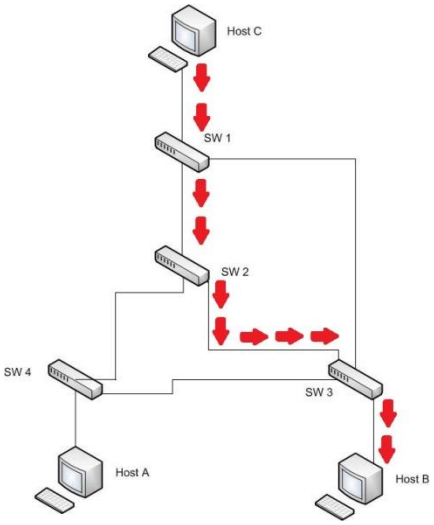
Σχήμα 5-14 Κατεύθυνση πακέτων κατά την αποστολή μηνύματός μεταξύ των τερματικών B και C– Πριν την εκτέλεση του CPLEX



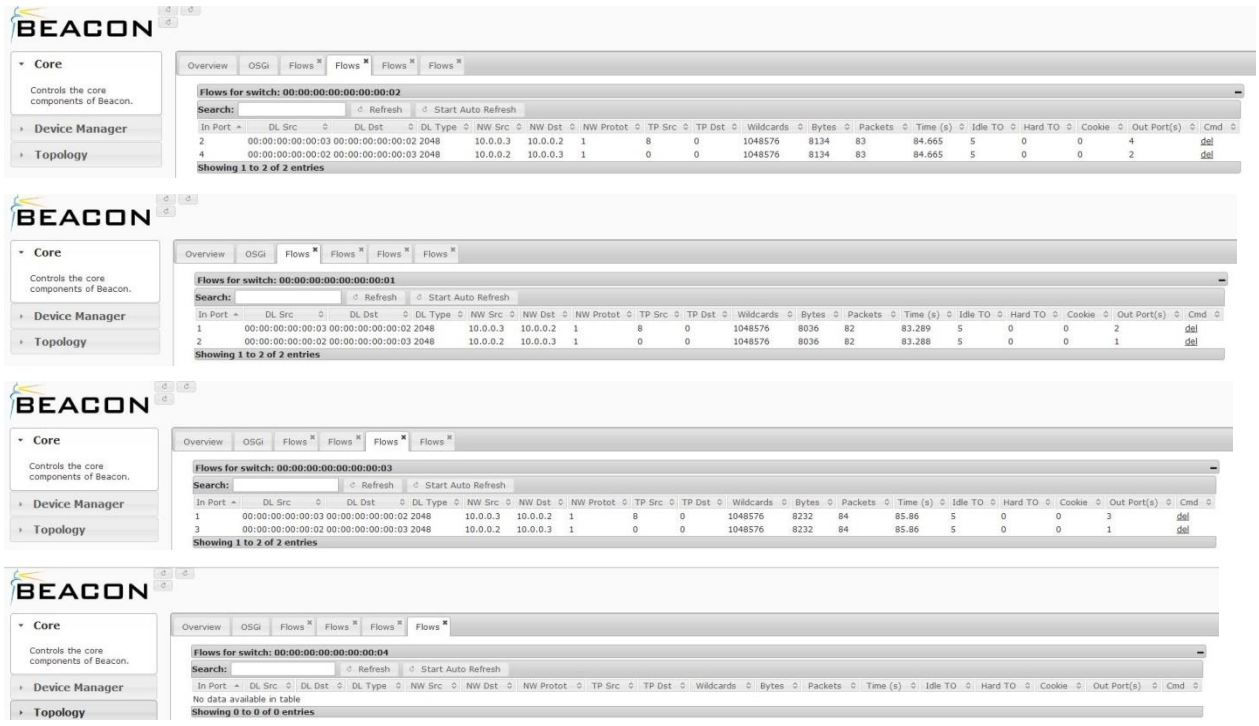
Σχήμα 5-15 7 Κατανομή κίνησης στους δρομολογητές κατά την αποστολή πακέτων από το τερματικό C στο B – Πριν την εκτέλεση του CPLEX

Στην συνέχεια το CPLEX όπως φαίνεται και στο σχήμα 5.6 , δίνει ως αποτέλεσμα την κατάργηση της σύνδεσης μεταξύ S1-S3 καθώς υπολογίστηκε πως η διαδρομή S1-S2-S3 είναι ενεργειακά πιο αποδοτική. Έτσι ο ελεγκτής αναλαμβάνει την απενεργοποίηση της θύρας 3 του δρομολογητή S1 και τη θύρα 2 του S3.

Άμεση συνέπεια η αναδρομολόγηση των πακέτων του μηνύματός που αποστέλλεται μεταξύ των δύο υπολογιστών. Η νέα διαδρομή των πακέτων είναι αυτή μεταξύ των δρομολογητών S1,S2,S3 γεγονός που αποδεικνύεται και από τις αντίστοιχες εικόνες καθώς στην περίπτωση αυτή δημιουργείται κίνηση και στον δρομολογητή S2 με DPID: 00:00:00:00:00:00:00:02



Σχήμα 5-16 Κατεύθυνση πακέτων κατά την αποστολή μηνύματος μεταξύ των τερματικών B και C– Μετά την εκτέλεση του CPLEX



Σχήμα 5-17 Κατανομή κίνησης στους δρομολογητές κατά την αποστολή πακέτων από το τερματικό C στο B – Μετά την εκτέλεση του CPLEX

5.5 *Ανακεφαλαίωση και τελικά συμπεράσματα*

Η εξοικονόμηση ενέργειας στις μέρες μας αποτελεί τον σημαντικότερο προβληματισμό σε όλες τις κοινωνίες. Όπως γίνεται αντιληπτό, οι προβληματισμοί και η προσπάθεια αντιμετώπισης της κατανάλωση της ενέργειας στο διαδίκτυο και στις εφαρμογές του αποτελεί σημαντικό γεγονός, μιας και η χρήση του έχει γίνει σημαντικό κομμάτι στην καθημερινότητα του ανθρώπου.

Στην προσπάθεια αντιμετώπισης του φαινομένου αυτού, στα πλαίσια της παρούσας διπλωματικής εργασίας, αναπτύχθηκε ένας μηχανισμός μέσω του ελεγκτή Beacon, ο οποίος επικοινωνεί αμφίδρομα με τις δικτυακές συσκευές και μπορεί να ανταλλάσσει συνεχώς μηνύματα.Ο ελεγκτής κάνει χαρτογράφηση της τοπολογίας του δικτύου και συλλέγει στατιστικά στοιχεία σχετικά με το φορτίο που περνάει από τους δρομολογητές. Στην συνέχεια τα στοιχεία συλλέγονται από ένα λογισμικό βελτιστοποίησης που υποστηρίζει γλώσσα προγραμματισμού βελτιστοποίησης (OPL) , με σκοπό την εύρεση της ενεργειακά οικονομικότερης δρομολόγησης. Το Beacon στην συνέχεια έχει ως σκοπό να αναλύσει τα αποτελέσματα του CPLEX, και να επικοινωνήσει με τους αντίστοιχους δρομολογητές για να καταστείλουν την λειτουργία τους ή να εισέλθουν σε κατάσταση κανονικής λειτουργίας.

Η χρήση του μηχανισμού αυτού μπορεί να εφαρμοστεί και σε μεγάλης έκτασης δίκτυα με την χρήση ενός κεντρικοποιημένου ελεγκτή, παρέχοντας μια πιο αποτελεσματική και οικονομική χρήση των πόρων ενός δικτύου OpenFlow . Ο μηχανισμός που αναπτύχθηκε θα πρέπει να συνδυάζεται και με άλλους μηχανισμούς που θα πρέπει να ελέγχουν την υπηρεσία και της ανάγκες του κάθε μηνύματος. Φυσικά οι απαιτήσεις σε κάθε διασύνδεση μεταξύ των κόμβων του δικτύου εξαρτώνται άμεσα από την λειτουργία που επιτελείται από κάθε εφαρμογή. Σε μια κλήση και επικοινωνία φωνής με χρήση τεχνολογιών όπως VoIP (Voice over IP) είναι αναγκαία η γρήγορη και άμεση μεταφορά των δεδομένων στο δίκτυο, με ανοχές σε περίπτωση απώλειας κάποιων πακέτων δεδομένων. Σε άλλη περίπτωση όπως είναι η μεταφορά δεδομένων πληροφορίας όπως κειμένου ή πολυμεσικών δεδομένων (audio-video) υπάρχουν διαφορικές απαιτήσεις , όπως υψηλότερο εύρος ζώνης, ανοχή

σε περίπτωση χρονικής καθυστέρησης αλλά μηδενική ανοχή σε περίπτωση απώλειας πακέτου δεδομένων. Συνεπώς κατά την επιλογή του σημαντικότερης διαδρομής θα πρέπει να λαμβάνονται ως κριτήρια και οι ανάγκες τις εφαρμογής, ώστε να μην υποβαθμίζεται η ποιότητα της υπηρεσίας και να συμβάλλει στην πραγματική διευκόλυνση της λειτουργίας του δικτύου, προσθέτοντας παράλληλα τον χαρακτήρα του “πράσινου δικτύου” .

Γενικά, μπορούμε να δούμε με την χρήση του ελεγκτή Beacon και του πρωτοκόλλου OpenFlow το κύριο όφελος της SDN (Software Defined Networking) αρχιτεκτονικής, συγκριτικά με άλλες αρχιτεκτονικές δικτύου . Με την χρήση ενός κεντρικού ελεγκτή οι διαχειριστές ενός δικτύου SDN μπορούν να επέμβουν άμεσα και να τροποποιήσουν με όποιο τρόπο επιθυμούν τις υπηρεσίες που παρέχει το δίκτυο, μέσω του ελεγκτή, χωρίς να είναι αναγκαία η χειροκίνητη ρύθμιση των παραμέτρων των συσκευών του δικτύου (μεταγωγέων, δρομολογητών ή τερματικών). Επιτρέπεται ο αυτόματος έλεγχος της λειτουργίας των συσκευών , και αλλαγή της κατάστασης τους σε πραγματικό χρόνο , παρέχοντας εξοικονόμηση ενέργειας χωρίς παράλληλα να υποβαθμίζεται η ποιότητα λειτουργίας του δικτύου. Τέλος αναβαθμίζεται και η ασφάλεια του δικτύου, καθώς με ένα κεντρικό τρόπο ελέγχου μπορεί πιο εύκολα να ανιχνευθούν κακόβουλες ενέργειες ή λειτουργίες στο δίκτυο και να αντιμετωπιστούν άμεσα από τον διαχειριστή του δικτύου.

Η SDN αρχιτεκτονική είναι ακόμα μια νέα μέθοδος δημιουργίας και διαχείρισης δικτύων υπολογιστών, που ακόμη αναπτύσσεται και δοκιμάζεται. Με την πάροδο όμως του χρόνου τα πρωτόκολλα που την υποστηρίζουν (όπως το OpenFlow) γίνονται πιο ώριμα και αξιόπιστα, ενώ αναπτύσσονται όλο και περισσότεροι ελεγκτές σε διάφορες γλώσσες προγραμματισμού, πέραν του NOX, όπως ο Beacon ή ο Floodlight, που προσφέρουν όλο και περισσότερες δυνατότητες για την δημιουργία κεντροποιημένων, λειτουργικών και αξιόπιστων δικτύων υπολογιστών.

Βιβλιογραφία

[1] OpenFlow Specifications Sheet v1.4.0 2013

[2] Rob Sherwood, Michael Chan, Glen Gibby, Nikhil Handigol, Te-Yuan Huang, Peyman Kazemian, Masayoshi Kobayashi, David Underhill, Kok-Kiong Yapy, Guido Appenzellery, Nick McKeowny, *Carving Research Slices Out of Your Production Networks with OpenFlow* 2009

[3] https://openflow.stanford.edu/display/Beacon/Home , www.openflow.org

[4] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, Scott Shenker *Ethane: Taking Control of the Enterprise* 2007

[5] Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller, Nick McKeown *Implementing an OpenFlow Switch on the NetFPGA platform 2008*

[6] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, David Walker, *Composing Software-Defined Networks 2013*

[7] Dmitry Drutskoy, Eric Keller, Jennifer Rexford *Scalable Network Virtualization in Software-Defined-Networks 2013*

[8] Marco Canini, Daniele Venzano, Peter Peresini, Dejan Kostic, Jennifer Rexfrod *A NICE Way to Test OpenFlow Applications 2012*

[9] Luca Chiaraviglio, Delia Ciullo, Marco Mellia, Michela Meo *Modeling sleep mode gains in energy-aware networks 2013*

[10] Slavica Aleksic *Analysis of Power Consumption in Future High-Capacity Network Nodes 2009*

[11] R. Bolla, R. Bruschi, L. D'Agostino, P. Lago, C. Lombardo, S. Mangialardi, F. Podda, *EEDROP: An Energy-Aware Router Prototype 2013*

[12] David Erickson *The Beacon OpenFlow Controller 2010*

[13] https://www.mininet.org

[14] IBM ILOG CPLEX V12.1 *User's Manual for CPLEX* 2009

Παράρτημα

Παρακάτω επισυνάπτονται οι πιο σημαντικές προγραμματιστικές διεπαφές (API) του ελεγκτή Beacon, που χρησιμοποιήθηκε.

- Controller.java

```
package net.beaconcontroller.core.internal;

public class Controller implements IBeaconProvider, SelectListener {
    protected static Logger log = LoggerFactory.getLogger(Controller.class);
    protected static int LIVENESS_POLL_INTERVAL = 1000;
    protected static int LIVENESS_TIMEOUT = 5000;
    protected static String SWITCH_REQUIREMENTS_TIMER_KEY = "SW_REQ_TIMER";

    protected ConcurrentHashMap<Long, IOFSwitchExt> activeSwitches;
    protected CopyOnWriteArraySet<IOFSwitchExt> allSwitches;
    protected Map<String,String> callbackOrdering;
    protected boolean deletePreExistingFlows = true;
    protected ExecutorService es;
    protected BasicFactory factory;
    protected boolean immediate = false;
    protected ExecutorService initializerExecutorService;
    protected CopyOnWriteArrayList<IOFInitializerListener> initializerList;
    protected Object initializerLock;
    protected ConcurrentHashMap<IOFSwitchExt,
CopyOnWriteArrayList<IOFInitializerListener>> initializerMap;
    protected String initializerOrdering;
    protected String listenAddress;
    protected int listenPort = 6633;
    protected IOLoop listenerIOLoop;
    protected volatile boolean listenerStarted = false;
    protected ServerSocketChannel listenSock;
    protected Timer livenessTimer;
    protected ConcurrentMap<OFType, List<IOFMessageListener>> messageListeners;
    protected boolean noDelay = true;
    protected volatile boolean shuttingDown = false;
    protected Set<IOFSwitchListener> switchListeners;
    protected List<IOLoop> switchIOLoops;
    protected Integer threadCount;
    protected BlockingQueue<Update> updates;
    protected Thread updatesThread;

    protected class Update {
        public IOFSwitch sw;
        public boolean added;

        public Update(IOFSwitch sw, boolean added) {
            this.sw = sw;
            this.added = added;
        }
    }


    public Controller() {
        this.messageListeners =
            new ConcurrentHashMap<OFType, List<IOFMessageListener>>();
        this.switchListeners = new CopyOnWriteArraySet<IOFSwitchListener>();
        this.updates = new LinkedBlockingQueue<Update>();
        this.initializerLock = new Object();
        this.callbackOrdering = new HashMap<String, String>();
    }

    public void handleEvent(SelectionKey key, Object arg) throws IOException {
        if (arg instanceof ServerSocketChannel)
            handleListenEvent(key, (ServerSocketChannel)arg);
        else
            handleSwitchEvent(key, (IOFSwitchExt) arg);
    }

    protected void handleListenEvent(SelectionKey key, ServerSocketChannel ssc)
            throws IOException {
```

```
        SocketChannel sock = listenSock.accept();
        log.info("Switch connected from {}", sock.toString());
        sock.socket().setTcpNoDelay(this.noDelay);
        sock.configureBlocking(false);
        sock.socket().setSendBufferSize(1024*1024);
        OFSwitchImpl sw = new OFSwitchImpl();

        IOLoop sl = null;
        for (IOLoop loop : switchIOLoops) {
            if (sl == null || loop.getStreams().size() < sl.getStreams().size())
                sl = loop;
        }

        SelectionKey switchKey = sl.registerBlocking(sock, 0, sw);
        OFStream stream = new OFStream(sock, factory, switchKey, sl);
        stream.setImmediate(this.immediate);
        sw.setInputStream(stream);
        sw.setOutputStream(stream);
        sw.setSocketChannel(sock);
        sw.setBeaconProvider(this);
        sw.transitionToState(OFSwitchState.HELLO_SENT);

        addSwitch(sw);

        stream.write(factory.getMessage(OFType.HELLO));

        switchKey.interestOps(SelectionKey.OP_READ);
        sl.addStream(stream);
        log.info("Added switch {} to IOLoop {}", sw, sl);
        sl.wakeup();
    }

    protected void handleSwitchEvent(SelectionKey key, IOFSwitchExt sw) {
        OFStream out = ((OFStream)sw.getOutputStream());
        OFStream in = (OFStream) sw.getInputStream();
        try {

            if (!key.isValid())
                return;

            if (key.isReadable()) {
                List<OFMessage> msgs = in.read();
                if (msgs == null) {
                    // graceful disconnect
                    disconnectSwitch(key, sw);
                    return;
                }
                sw.setLastReceivedMessageTime(System.currentTimeMillis());
                handleMessages(sw, msgs);
            }

            if (key.isWritable()) {
                out.clearSelect();
                key.interestOps(SelectionKey.OP_READ);
            }

            if (out.getWriteFailure()) {
                disconnectSwitch(key, sw);
                return;
            }
        } catch (IOException e) {
            log.error("Exception during IOloop", e);
            disconnectSwitch(key, sw);
        } catch (CancelledKeyException e) {
            log.error("Exception during IOloop", e);
            disconnectSwitch(key, sw);
        }
    }

    protected void disconnectSwitch(SelectionKey key, IOFSwitchExt sw) {
        synchronized (sw) {
            key.cancel();
            OFStream stream = (OFStream) sw.getInputStream();
            stream.getIOLoop().removeStream(stream);
            removeSwitch(sw);
            try {
                sw.getSocketChannel().socket().close();
```

```
            } catch (IOException e1) {
            }
            this.initializerMap.remove(sw);
            if (!OFSwitchState.DISCONNECTED.equals(sw.getState())) {
                sw.transitionToState(OFSwitchState.DISCONNECTED);
            }
            log.info("Switch disconnected {}", sw);
        }
    }

    @SuppressWarnings("unchecked")
    protected void handleMessages(IOFSwitchExt sw, List<OFMessage> msgs)
            throws IOException {
        for (OFMessage m : msgs) {
            if (((OFStream)sw.getInputStream()).getWriteFailure()) {
                break;
            }
            switch (m.getType()) {
                case ECHO_REQUEST:
                    OFMessageInStream in = sw.getInputStream();
                    OFMessageOutStream out = sw.getOutputStream();
                    OFEchoReply reply = (OFEchoReply) in
                            .getMessageFactory().getMessage(
                                    OFType.ECHO_REPLY);
                    reply.setXid(m.getXid());
                    out.write(reply);
                    break;
                case ECHO_REPLY:
                    break;
                case ERROR:
                    logError(sw, (OFError)m);
                default:
                    switch (sw.getState()) {
                        case DISCONNECTED:
                            log.info("Switch {} in state DISCONNECTED, exiting message
processing loop", sw);
                            return;
                        case HELLO_SENT:
                            if (m.getType() == OFType.HELLO) {
                                log.debug("HELLO from {}", sw);

sw.transitionToState(OFSwitchState.FEATURES_REQUEST_SENT);

sw.getOutputStream().write(factory.getMessage(OFType.FEATURES_REQUEST));
                            }
                            break;
                        case FEATURES_REQUEST_SENT:
                            if (m.getType() == OFType.FEATURES_REPLY) {
                                log.debug("Features Reply from {}", sw);
                                sw.setFeaturesReply((OFFeaturesReply) m);

                                OFStatisticsRequest sr = new OFStatisticsRequest();
                                sr.setStatisticType(OFStatisticsType.DESC);
                                sw.getOutputStream().write(sr);

sw.transitionToState(OFSwitchState.DESCRIPTION_STATISTICS_REQUEST_SENT);
                            }
                            break;
                        case DESCRIPTION_STATISTICS_REQUEST_SENT:
                            if (m.getType() == OFType.STATS_REPLY) {
                                OFStatisticsReply sr = (OFStatisticsReply) m;
                                if (sr.getStatisticType() == OFStatisticsType.DESC &&
sr.getStatistics().size() > 0) {
                                    OFDescriptionStatistics desc =
(OFDescriptionStatistics) sr.getStatistics().get(0);
                                    sw.setDescriptionStatistics(desc);
                                    log.debug("Description Statistics Reply from {}:
{}", sw, desc);

                                    OFSetConfig config = (OFSetConfig) factory
                                            .getMessage(OFType.SET_CONFIG);
                                    config.setMissSendLength((short) 0xffff)
                                    .setLengthU(OFSetConfig.MINIMUM_LENGTH);
                                    sw.getOutputStream().write(config);

sw.getOutputStream().write(factory.getMessage(OFType.BARRIER_REQUEST));
```

```
sw.getOutputStream().write(factory.getMessage(OFType.GET_CONFIG_REQUEST));

sw.transitionToState(OFSwitchState.GET_CONFIG_REQUEST_SENT);
                                }
                            }
                            break;
                        case GET_CONFIG_REQUEST_SENT:
                            if (m.getType() == OFType.GET_CONFIG_REPLY) {
                                OFGetConfigReply cr = (OFGetConfigReply) m;
                                if (cr.getMissSendLength() == (short)0xffff) {
                                    log.debug("Config Reply from {} confirms miss
length set to 0xffff", sw);
                                    sw.transitionToState(OFSwitchState.INITIALIZING);

                                    CopyOnWriteArrayList<IOFInitializerListener>
initializers =

(CopyOnWriteArrayList<IOFInitializerListener>) initializerList.clone();
                                    this.initializerMap.put(sw, initializers);
                                    log.debug("Remaining initializers for switch {}:
{}", sw, this.initializerMap.get(sw));
                                    if (deletePreExistingFlows) {
                                        OFMatch match = new
OFMatch().setWildcards(OFMatch.OFPFW_ALL);
                                        OFMessage fm = ((OFFlowMod)
sw.getInputStream().getMessageFactory()
                                                .getMessage(OFType.FLOW_MOD))
                                                .setMatch(match)
                                                .setCommand(OFFlowMod.OFPFC_DELETE)
                                                .setOutPort(OFPort.OFPP_NONE)

.setLength(U16.t(OFFlowMod.MINIMUM_LENGTH));
                                        sw.getOutputStream().write(fm);

sw.getOutputStream().write(factory.getMessage(OFType.BARRIER_REQUEST));
                                    }

                                    if (initializers.size() > 0)
                                        queueInitializer(sw,
initializers.iterator().next());
                                    else
                                        advanceInitializers(sw);
                                } else {
                                    log.error("Switch {} refused to set miss send
length to 0xffff, disconnecting", sw);

disconnectSwitch(((OFStream)sw.getInputStream()).getKey(), sw);
                                    return;
                                }
                            }
                            break;
                        case INITIALIZING:
                            CopyOnWriteArrayList<IOFInitializerListener> initializers
=
                                initializerMap.get(sw);
                            Iterator<IOFInitializerListener> it =
initializers.iterator();
                            if (it.hasNext()) {
                                IOFInitializerListener listener = it.next();
                                try {
                                    listener.initializerReceive(sw, m);
                                } catch (Exception e) {
                                    log.error(
                                            "Error calling initializer listener: {} on
switch: {} for message: {}, removing listener",
                                            new Object[] { listener, sw, m });
                                    advanceInitializers(sw);
                                }
                            }
                            break;
                        case ACTIVE:
                            List<IOFMessageListener> listeners = messageListeners
                            .get(m.getType());
                            if (listeners != null) {
                                for (IOFMessageListener listener : listeners) {
                                    try {
```

```
                                    if (listener instanceof IOFSwitchFilter) {
                                        if
(!((IOFSwitchFilter)listener).isInterested(sw)) {
                                            continue;
                                        }
                                    }
                                    if (Command.STOP.equals(listener.receive(sw,
m))) {
                                        break;
                                    }
                                } catch (Exception e) {
                                    log.error("Failure calling listener ["+
                                            listener.toString()+
                                            "] with message ["+m.toString()+
                                            "]", e);
                                }
                            }
                        } else {
                            log.debug("Unhandled OF Message: {} from {}", m, sw);
                        }
                        break;
                    }
                }
            }
        }

    protected void logError(IOFSwitch sw, OFError error) {
        OFErrorType et = OFErrorType.values()[0xffff & error.getErrorType()];
        switch (et) {
            case OFPET_HELLO_FAILED:
                OFHelloFailedCode hfc = OFHelloFailedCode.values()[0xffff &
error.getErrorCode()];
                log.error("Error {} {} from {}", new Object[] {et, hfc, sw});
                break;
            case OFPET_BAD_REQUEST:
                OFBadRequestCode brc = OFBadRequestCode.values()[0xffff &
error.getErrorCode()];
                log.error("Error {} {} from {}", new Object[] {et, brc, sw});
                break;
            case OFPET_BAD_ACTION:
                OFBadActionCode bac = OFBadActionCode.values()[0xffff &
error.getErrorCode()];
                log.error("Error {} {} from {}", new Object[] {et, bac, sw});
                break;
            case OFPET_FLOW_MOD_FAILED:
                OFFlowModFailedCode fmfc = OFFlowModFailedCode.values()[0xffff &
error.getErrorCode()];
                log.error("Error {} {} from {}", new Object[] {et, fmfc, sw});
                break;
            case OFPET_PORT_MOD_FAILED:
                OFPortModFailedCode pmfc = OFPortModFailedCode.values()[0xffff &
error.getErrorCode()];
                log.error("Error {} {} from {}", new Object[] {et, pmfc, sw});
                break;
            case OFPET_QUEUE_OP_FAILED:
                OFQueueOpFailedCode qofc = OFQueueOpFailedCode.values()[0xffff &
error.getErrorCode()];
                log.error("Error {} {} from {}", new Object[] {et, qofc, sw});
                break;
            default:
                break;
        }
    }

    public synchronized void addOFMessageListener(OFType type, IOFMessageListener
listener) {
        List<IOFMessageListener> listeners = messageListeners.get(type);
        if (listeners == null) {
            messageListeners.putIfAbsent(type,
                    new CopyOnWriteArrayList<IOFMessageListener>());
            listeners = messageListeners.get(type);
        }

        if (callbackOrdering != null && callbackOrdering.containsKey(type.toString()))
{
            String order = callbackOrdering.get(type.toString());
```

```
            orderedInsert(order, listeners, listener, new
IOrderName<IOFMessageListener>() {
                @Override
                public String get(IOFMessageListener obj) {
                    return obj.getName();
                }
            });
        } else {
            listeners.add(listener);
        }
    }

       protected <T> void orderedInsert(String order, List<T> objects, T object,
IOrderName<T> orderName) {
        if (!order.contains(orderName.get(object))) {
            objects.add(object);
            return;
        }

        String[] orderArray = order.split(",");
        int myPos = 0;
        for (int i = 0; i < orderArray.length; ++i) {
            orderArray[i] = orderArray[i].trim();
            if (orderArray[i].equals(orderName.get(object)))
                myPos = i;
        }
        List<String> beforeList = Arrays.asList(Arrays.copyOfRange(orderArray, 0,
myPos));

        boolean added = false;
        if (objects.size() > 0) {
            for (int i = 0; i < objects.size(); ++i) {
                if (beforeList.contains(orderName.get(objects.get(i))))
                    continue;
                objects.add(i, object);
                added = true;
                break;
            }
        }
        if (!added) {
            objects.add(object);
        }
    }

    public synchronized void removeOFMessageListener(OFType type, IOFMessageListener
listener) {
        List<IOFMessageListener> listeners = messageListeners.get(type);
        if (listeners != null) {
            listeners.remove(listener);
        }
    }

    public void startUp() throws IOException {
        initializerList = new CopyOnWriteArrayList<IOFInitializerListener>();
        initializerMap = new ConcurrentHashMap<IOFSwitchExt,
CopyOnWriteArrayList<IOFInitializerListener>>();
        switchIOLoops = new ArrayList<IOLoop>();
        activeSwitches = new ConcurrentHashMap<Long, IOFSwitchExt>();
        allSwitches = new CopyOnWriteArraySet<IOFSwitchExt>();

        if (threadCount == null)
            threadCount = 1;

        this.factory = new BasicFactory();

        es = Executors.newFixedThreadPool(threadCount+1);
        initializerExecutorService = Executors.newFixedThreadPool(2);

        for (int i = 0; i < threadCount; ++i) {
            final IOLoop sl = new IOLoop(this, 500, i);
            switchIOLoops.add(sl);
            es.execute(new Runnable() {
                public void run() {
                    try {
                        log.info("Started thread {} for IOLoop {}",
Thread.currentThread(), sl);
                        sl.doLoop();
```

```
                    } catch (Exception e) {
                        log.error("Exception during worker loop, terminating thread",
e);
                    }
                }}
            );
        }

        updatesThread = new Thread(new Runnable () {
            @Override
            public void run() {
                while (true) {
                    try {
                        Update update = updates.take();
                        if (switchListeners != null) {
                            for (IOFSwitchListener listener : switchListeners) {
                                try {
                                    if (update.added)
                                        listener.addedSwitch(update.sw);
                                    else
                                        listener.removedSwitch(update.sw);
                                } catch (Exception e) {
                                    log.error("Error calling switch listener", e);
                                }
                            }
                        }
                    } catch (InterruptedException e) {
                        log.warn("Controller updates thread interupted", e);
                        if (shuttingDown)
                            return;
                    }
                }
            }}, "Controller Updates");
        updatesThread.start();

        livenessTimer = new Timer();
        livenessTimer.scheduleAtFixedRate(new TimerTask() {
            @Override
            public void run() {
                checkSwitchLiveness();
            }
        }, LIVENESS_POLL_INTERVAL, LIVENESS_POLL_INTERVAL);

        log.info("Beacon Core Started");
    }

    public synchronized void startListener() {
        if (listenerStarted)
            return;

        try {
            listenSock = ServerSocketChannel.open();
            listenSock.socket().setReceiveBufferSize(512*1024);
            listenSock.configureBlocking(false);
            if (listenAddress != null) {
                listenSock.socket().bind(
                        new java.net.InetSocketAddress(InetAddress
                                .getByAddress(IPv4
                                        .toIPv4AddressBytes(listenAddress)),
                                listenPort));
            } else {
                listenSock.socket().bind(new java.net.InetSocketAddress(listenPort));
            }
            listenSock.socket().setReuseAddress(true);

            listenerIOLoop = new IOLoop(this, -1);
            listenerIOLoop.register(listenSock, SelectionKey.OP_ACCEPT, listenSock);
        } catch (IOException e) {
            log.error("Failure opening listening socket", e);
            System.exit(-1);
        }

        log.info("Controller listening on {}:{}", listenAddress == null ? "*"
                : listenAddress, listenPort);


        es.execute(new Runnable() {
```

```
            public void run() {
                try {
                    listenerIOLoop.doLoop();
                } catch (Exception e) {
                    log.error("Exception during accept loop, terminating thread", e);
                }
            }}
        );

        listenerStarted = true;
    }

    public synchronized void stopListener() {
        if (!listenerStarted)
            return;

        try {
            listenerIOLoop.shutdown();
            listenSock.socket().close();
            listenSock.close();
        } catch (IOException e) {
            log.error("Failure shutting down listening socket", e);
        } finally {
            listenerStarted = false;
        }
    }

    public void shutDown() throws IOException {
        shuttingDown = true;
        livenessTimer.cancel();

        stopListener();

        for (Iterator<Entry<Long, IOFSwitchExt>> it =
activeSwitches.entrySet().iterator(); it.hasNext();) {
            Entry<Long, IOFSwitchExt> entry = it.next();
            entry.getValue().getSocketChannel().socket().close();
            it.remove();
        }

        for (IOLoop sl : switchIOLoops) {
            sl.shutdown();
        }

        es.shutdown();
        initializerExecutorService.shutdownNow();
        updatesThread.interrupt();
        log.info("Beacon Core Shutdown");
    }


    protected void checkSwitchLiveness() {
        long now = System.currentTimeMillis();
        log.trace("Liveness timer running");

        for (Iterator<IOFSwitchExt> it = allSwitches.iterator(); it.hasNext();) {
            IOFSwitchExt sw = it.next();
            long last = sw.getLastReceivedMessageTime();
            SelectionKey key = ((OFStream)sw.getInputStream()).getKey();

            if (now - last >= (2*LIVENESS_TIMEOUT)) {
                log.info("Switch liveness timeout detected {}ms, disconnecting {}",
now - last, sw);
                disconnectSwitch(key, sw);
            } else if (now - last >= LIVENESS_TIMEOUT) {
                OFEchoRequest echo = new OFEchoRequest();
                try {
                    sw.getOutputStream().write(echo);
                } catch (IOException e) {
                    log.error("Failure sending liveness probe, disconnecting switch "
+ sw.toString(), e);
                    disconnectSwitch(key, sw);
                }
            }
        }
    }
    public void setCallbackOrdering(Map<String, String> callbackOrdering) {
```

```
        this.callbackOrdering = callbackOrdering;
    }


    protected ConcurrentMap<OFType, List<IOFMessageListener>> getMessageListeners() {
        return messageListeners;
    }


    protected void setMessageListeners(
            ConcurrentMap<OFType, List<IOFMessageListener>> messageListeners) {
        this.messageListeners = messageListeners;
    }

    @Override
    public Map<Long, IOFSwitch> getSwitches() {
        return Collections.unmodifiableMap(new HashMap<Long,
IOFSwitch>(this.activeSwitches));
    }



    protected Set<IOFSwitchExt> getAllSwitches() {
        return Collections.unmodifiableSet(this.allSwitches);
    }

    @Override
    public void addOFSwitchListener(IOFSwitchListener listener) {
        this.switchListeners.add(listener);
    }

    @Override
    public void removeOFSwitchListener(IOFSwitchListener listener) {
        this.switchListeners.remove(listener);
    }


    protected void addSwitch(IOFSwitchExt sw) {
        this.allSwitches.add(sw);
    }
    protected void addActiveSwitch(IOFSwitchExt sw) {
        this.activeSwitches.put(sw.getId(), sw);
        Update update = new Update(sw, true);
        try {
            this.updates.put(update);
        } catch (InterruptedException e) {
            log.error("Failure adding update to queue", e);
        }
    }

    protected void removeSwitch(IOFSwitchExt sw) {
        this.allSwitches.remove(sw);
        if (OFSwitchState.ACTIVE == sw.getState()) {
            if (!this.activeSwitches.remove(sw.getId(), sw)) {
                log.warn("Removing switch {} has already been replaced", sw);
            }
            Update update = new Update(sw, false);
            try {
                this.updates.put(update);
            } catch (InterruptedException e) {
                log.error("Failure adding update to queue", e);
            }
        }
    }

    @Override
    public Map<OFType, List<IOFMessageListener>> getListeners() {
        return Collections.unmodifiableMap(this.messageListeners);
    }

    public void setListenAddress(String listenAddress) {
        this.listenAddress = listenAddress;
    }

    public void setListenPort(int listenPort) {
        this.listenPort = listenPort;
    }
```

```
    public void setThreadCount(Integer threadCount) {
        this.threadCount = threadCount;
    }



    public void setImmediate(boolean immediate) {
        this.immediate = immediate;
    }



    public void setNoDelay(boolean noDelay) {
        this.noDelay = noDelay;
    }


    public void setDeletePreExistingFlows(boolean deletePreExistingFlows) {
        this.deletePreExistingFlows = deletePreExistingFlows;
    }

    @Override
    public void addOFInitializerListener(IOFInitializerListener listener) {
        synchronized (this.initializerLock) {
            if (initializerOrdering != null) {
                String order = initializerOrdering;
                orderedInsert(order, this.initializerList, listener, new
IOrderName<IOFInitializerListener>() {
                    @Override
                    public String get(IOFInitializerListener obj) {
                        return obj.getInitializerName();
                    }
                });
            } else {
                this.initializerList.add(listener);
            }
        }
    }

    @Override
    public void removeOFInitializerListener(IOFInitializerListener listener) {
        synchronized (this.initializerLock) {
            this.initializerList.remove(listener);
            Iterator<Map.Entry<IOFSwitchExt,
CopyOnWriteArrayList<IOFInitializerListener>>> it =
                this.initializerMap.entrySet().iterator();
            while (it.hasNext()) {
                Map.Entry<IOFSwitchExt, CopyOnWriteArrayList<IOFInitializerListener>>
entry =
                    it.next();
                entry.getValue().remove(listener);
            }
        }
    }

    @Override
    public void initializerComplete(IOFSwitch sw, IOFInitializerListener initializer)
{
        log.debug("Initializer for switch {} has completed: {}", sw, initializer);
        IOFSwitchExt swExt = (IOFSwitchExt) sw;
        advanceInitializers(swExt);
    }

    protected void queueInitializer(final IOFSwitch sw, final IOFInitializerListener
initializer) {
        initializerExecutorService.submit(new Runnable() {
            @Override
            public void run() {
                initializer.initializerStart(sw);
            }});
    }

    protected void advanceInitializers(IOFSwitchExt sw) {
        CopyOnWriteArrayList<IOFInitializerListener> initializers =
initializerMap.get(sw);
```

```
        Iterator<IOFInitializerListener> it = initializers.iterator();
        if (it.hasNext()) {
            IOFInitializerListener initializer = it.next();
            initializers.remove(initializer);
            log.debug("Remaining initializers for switch {}: {}", sw, initializers);
        }

        if (it.hasNext()) {
            IOFInitializerListener initializer = it.next();
            queueInitializer(sw, initializer);
        } else {


            synchronized (sw) {
                if (!OFSwitchState.DISCONNECTED.equals(sw.getState())) {
                    sw.transitionToState(OFSwitchState.ACTIVE);
                    // Add switch to active list
                    addActiveSwitch(sw);
                }
                initializerMap.remove(sw);
            }
        }
    }



    public void setInitializerOrdering(String initializerOrdering) {
        this.initializerOrdering = initializerOrdering;
    }

    @Override
    public List<IOFInitializerListener> getInitializers() {
        return Collections.unmodifiableList(this.initializerList);
    }

    public Map<String, String> getCallbackOrdering() {
        return callbackOrdering;
    }

    @Override
    public InetAddress getListeningIPAddress() {
        return listenSock.socket().getInetAddress();
    }

    @Override
    public int getListeningPort() {
        return listenSock.socket().getLocalPort();
    }
}
```

- IOLoop.java

```
package net.beaconcontroller.core.io.internal;


public class IOLoop {
    protected SelectListener callback;
    protected boolean dontStop;
    protected int id;
    protected Object registrationLock;
    protected int registrationRequests = 0;
    protected Queue<Object[]> registrationQueue;
    protected Selector selector;
    protected List<OFStream> streams;
    protected long timeout;


    public IOLoop(SelectListener cb, int id) throws IOException {
        callback = cb;
        dontStop = true;
        this.id = id;
        selector = SelectorProvider.provider().openSelector();
        registrationLock = new Object();
        registrationQueue = new ConcurrentLinkedQueue<Object[]>();
```

```
        streams = new CopyOnWriteArrayList<OFStream>();
        timeout = 0;
    }


    public IOLoop(SelectListener cb, long timeout, int id) throws IOException {
        callback = cb;
        dontStop = true;
        this.id = id;
        selector = SelectorProvider.provider().openSelector();
        registrationLock = new Object();
        registrationQueue = new ConcurrentLinkedQueue<Object[]>();
        streams = new CopyOnWriteArrayList<OFStream>();
        this.timeout = timeout;
    }

    public void register(SelectableChannel ch, int ops, Object arg)
            throws ClosedChannelException {
        registrationQueue.add(new Object[] {ch, ops, arg});
    }


    public synchronized SelectionKey registerBlocking(SelectableChannel ch, int ops,
Object arg)
            throws ClosedChannelException {
        synchronized (registrationLock) {
            registrationRequests++;
        }
        selector.wakeup();
        SelectionKey key = ch.register(selector, ops, arg);
        synchronized (registrationLock) {
            registrationRequests--;
            registrationLock.notifyAll();
        }
        return key;
    }


    public void doLoop() throws IOException {
        int nEvents;
        processRegistrationQueue();

        while (dontStop) {
            for (OFStream stream : streams) {
                stream.clearWrote();
                if (stream.getNeedsSelect()) {
                    stream.getKey().interestOps(
                            stream.getKey().interestOps()
                                    | SelectionKey.OP_WRITE);
                }
            }
            nEvents = selector.select(timeout);
            if (nEvents > 0) {
                for (Iterator<SelectionKey> i = selector.selectedKeys()
                        .iterator(); i.hasNext();) {
                    SelectionKey sk = i.next();
                    i.remove();

                    if (!sk.isValid())
                        continue;

                    Object arg = sk.attachment();
                    callback.handleEvent(sk, arg);
                }
            }

            if (this.registrationQueue.size() > 0)
                processRegistrationQueue();

            if (registrationRequests > 0) {
                synchronized (registrationLock) {
                    while (registrationRequests > 0) {
                        try {
                            registrationLock.wait();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
```

```
                    }
                }
            }
        }
    }

    protected void processRegistrationQueue() {
        for (Iterator<Object[]> it = registrationQueue.iterator(); it.hasNext();) {
            Object[] args = it.next();
            SelectableChannel ch = (SelectableChannel) args[0];
            try {
                ch.register(selector, (Integer) args[1], args[2]);
            } catch (ClosedChannelException e) {
                e.printStackTrace();
            }
            it.remove();
        }
    }



    public void wakeup() {
       if (selector != null) {
           selector.wakeup();
       }
    }

    public void shutdown() {
        this.dontStop = false;
        wakeup();
    }

    public void addStream(OFStream stream) {
        this.streams.add(stream);
    }

    public void removeStream(OFStream stream) {
        this.streams.remove(stream);
    }



    public List<OFStream> getStreams() {
        return streams;
    }

    @Override
    public String toString() {
        return "IOLoop [id=" + id + " stream_count="+streams.size()+"]";
    }
}
```

- OFStream.java

```
package net.beaconcontroller.core.io.internal;

public class OFStream extends OFMessageAsyncStream implements OFMessageSafeOutStream {
    protected Logger log = LoggerFactory.getLogger(OFStream.class);

    protected SelectionKey key;
    protected IOLoop ioLoop;
    protected boolean writeFailure = false;
    protected boolean needsSelect = false;
    protected boolean wrote = false;
    protected boolean immediate = false;

    public OFStream(SocketChannel sock, OFMessageFactory messageFactory,
            SelectionKey key, IOLoop selectLoop) throws IOException {
        super(sock, messageFactory);
        this.key = key;
        this.ioLoop = selectLoop;
    }

    @Override
```

```
public void write(OFMessage m) throws IOException {
    synchronized (outBuf) {
        appendMessageToOutBuf(m);
        if (!wrote && !needsSelect) {
            flush();
        }
    }
  }

@Override
public void write(List<OFMessage> l) throws IOException {
    synchronized (outBuf) {
        for (OFMessage m : l) {
            appendMessageToOutBuf(m);
        }
        if (!wrote && !needsSelect) {
            flush();
        }
    }
}

public void flush() throws IOException {
    synchronized (outBuf) {
        if (wrote || needsSelect)
            return;
        outBuf.flip();
        try {
            sock.write(outBuf);
        } catch (IOException e) {
            log.info("Detected remote switch hangup {}", sock);
            this.writeFailure = true;
        }
        if (!immediate)
            wrote = true;
        outBuf.compact();
        if (outBuf.position() > 0) {
            needsSelect = true;
        }
    }
}

public boolean needsFlush() {
    synchronized (outBuf) {
        return outBuf.position() > 0;
    }
}


public boolean getWriteFailure() {
    return writeFailure;
}


public boolean getNeedsSelect() {
    synchronized (outBuf) {
        return needsSelect;
    }
}

public void clearWrote() throws IOException {
    synchronized (outBuf) {
        this.wrote = false;
        if (outBuf.position() > 0 && !needsSelect)
            flush();
    }
}

public void clearSelect() throws IOException {
    synchronized (outBuf) {
        this.wrote = false;
        this.needsSelect = false;
        if (outBuf.position() > 0)
            flush();
    }
}

public SelectionKey getKey() {
```

```
        return key;
    }


    public IOLoop getIOLoop() {
        return ioLoop;
    }


    public boolean getImmediate() {
        return immediate;
    }


    public void setImmediate(boolean immediate) {
        this.immediate = immediate;
    }
    }
```

- IPv4.java

```
package net.beaconcontroller.packet;

public class IPv4 extends BasePacket {
    public static byte PROTOCOL_ICMP = 0x1;
    public static byte PROTOCOL_UDP = 0x11;
    public static Map<Byte, Class<? extends IPacket>> protocolClassMap;

    static {
        protocolClassMap = new HashMap<Byte, Class<? extends IPacket>>();
        protocolClassMap.put(PROTOCOL_ICMP, ICMP.class);
        protocolClassMap.put(PROTOCOL_UDP, UDP.class);
    }

    protected byte version;
    protected byte headerLength;
    protected byte diffServ;
    protected short totalLength;
    protected short identification;
    protected byte flags;
    protected short fragmentOffset;
    protected byte ttl;
    protected byte protocol;
    protected short checksum;
    protected int sourceAddress;
    protected int destinationAddress;
    protected byte[] options;

    public IPv4() {
        super();
        this.version = 4;
    }


    public byte getVersion() {
        return version;
    }


    public IPv4 setVersion(byte version) {
        this.version = version;
        return this;
    }


    public byte getHeaderLength() {
        return headerLength;
    }

    public IPv4 setHeaderLength(byte headerLength) {
        this.headerLength = headerLength;
        return this;
    }
```

```
public byte getDiffServ() {
    return diffServ;
}

public IPv4 setDiffServ(byte diffServ) {
    this.diffServ = diffServ;
    return this;
}

public short getTotalLength() {
    return totalLength;
}

public IPv4 setTotalLength(short totalLength) {
    this.totalLength = totalLength;
    return this;
}

public short getIdentification() {
    return identification;
}

public IPv4 setIdentification(short identification) {
    this.identification = identification;
    return this;
}

public byte getFlags() {
    return flags;
}

public IPv4 setFlags(byte flags) {
    this.flags = flags;
    return this;
}

public short getFragmentOffset() {
    return fragmentOffset;
}

public IPv4 setFragmentOffset(short fragmentOffset) {
    this.fragmentOffset = fragmentOffset;
    return this;
}

public byte getTtl() {
    return ttl;
}

public IPv4 setTtl(byte ttl) {
    this.ttl = ttl;
    return this;
}

public byte getProtocol() {
    return protocol;
}

public IPv4 setProtocol(byte protocol) {
    this.protocol = protocol;
    return this;
}

public short getChecksum() {
    return checksum;
}


public IPv4 setChecksum(short checksum) {
    this.checksum = checksum;
    return this;
}

public int getSourceAddress() {
    return sourceAddress;
}
```

```
    public IPv4 setSourceAddress(int sourceAddress) {
        this.sourceAddress = sourceAddress;
        return this;
    }

    public IPv4 setSourceAddress(String sourceAddress) {
        this.sourceAddress = IPv4.toIPv4Address(sourceAddress);
        return this;
    }


    public int getDestinationAddress() {
        return destinationAddress;
    }

    public IPv4 setDestinationAddress(int destinationAddress) {
        this.destinationAddress = destinationAddress;
        return this;
    }

    public IPv4 setDestinationAddress(String destinationAddress) {
        this.destinationAddress = IPv4.toIPv4Address(destinationAddress);
        return this;
    }
    public byte[] getOptions() {
        return options;
    }

    public IPv4 setOptions(byte[] options) {
        if (options != null && (options.length % 4) > 0)
            throw new IllegalArgumentException(
                    "Options length must be a multiple of 4");
        this.options = options;
        return this;
    }

    public byte[] serialize() {
        byte[] payloadData = null;
        if (payload != null) {
            payload.setParent(this);
            payloadData = payload.serialize();
        }

        if (this.headerLength == 0) {
            int optionsLength = 0;
            if (this.options != null)
                optionsLength = this.options.length / 4;
            this.headerLength = (byte) (5 + optionsLength);
        }

        if (this.totalLength == 0) {
            this.totalLength = (short) (this.headerLength * 4 + ((payloadData == null)
? 0
                    : payloadData.length));
        }

        if (this.parent != null && this.parent instanceof Ethernet)
            ((Ethernet)this.parent).setEtherType(Ethernet.TYPE_IPv4);

        byte[] data = new byte[this.totalLength];
        ByteBuffer bb = ByteBuffer.wrap(data);

        bb.put((byte) (((this.version & 0xf) << 4) | (this.headerLength & 0xf)));
        bb.put(this.diffServ);
        bb.putShort(this.totalLength);
        bb.putShort(this.identification);
        bb.putShort((short) (((this.flags & 0x7) << 13) | (this.fragmentOffset &
0x1fff)));
        bb.put(this.ttl);
        bb.put(this.protocol);
        bb.putShort(this.checksum);
        bb.putInt(this.sourceAddress);
        bb.putInt(this.destinationAddress);
        if (this.options != null)
            bb.put(this.options);
        if (payloadData != null)
```

```
            bb.put(payloadData);

        if (this.checksum == 0) {
            bb.rewind();
            int accumulation = 0;
            for (int i = 0; i < this.headerLength * 2; ++i) {
                accumulation += 0xffff & bb.getShort();
            }
            accumulation = ((accumulation >> 16) & 0xffff)
                    + (accumulation & 0xffff);
            this.checksum = (short) (~accumulation & 0xffff);
            bb.putShort(10, this.checksum);
        }
        return data;
    }

    @Override
    public IPacket deserialize(byte[] data, int offset, int length) {
        ByteBuffer bb = ByteBuffer.wrap(data, offset, length);
        short sscratch;

        this.version = bb.get();
        this.headerLength = (byte) (this.version & 0xf);
        this.version = (byte) ((this.version >> 4) & 0xf);
        this.diffServ = bb.get();
        this.totalLength = bb.getShort();
        this.identification = bb.getShort();
        sscratch = bb.getShort();
        this.flags = (byte) ((sscratch >> 13) & 0x7);
        this.fragmentOffset = (short) (sscratch & 0x1fff);
        this.ttl = bb.get();
        this.protocol = bb.get();
        this.checksum = bb.getShort();
        this.sourceAddress = bb.getInt();
        this.destinationAddress = bb.getInt();

        if (this.headerLength > 5) {
            int optionsLength = (this.headerLength - 5) * 4;
            this.options = new byte[optionsLength];
            bb.get(this.options);
        }

        IPacket payload;
        if (IPv4.protocolClassMap.containsKey(this.protocol)) {
            Class<? extends IPacket> clazz = IPv4.protocolClassMap.get(this.protocol);
            try {
                payload = clazz.newInstance();
            } catch (Exception e) {
                throw new RuntimeException("Error parsing payload for IPv4 packet",
e);
            }
        } else {
            payload = new Data();
        }
        this.payload = payload.deserialize(data, bb.position(), bb.limit()-
bb.position());
        this.payload.setParent(this);
        return this;
    }

    public static int toIPv4Address(String ipAddress) {
        String[] octets = ipAddress.split("\\.");
        if (octets.length != 4)
            throw new IllegalArgumentException("Specified IPv4 address must" +
                "contain 4 sets of numerical digits separated by periods");

        int result = 0;
        for (int i = 0; i < 4; ++i) {
            result |= Integer.valueOf(octets[i]) << ((3-i)*8);
        }
        return result;
    }

    public static String fromIPv4Address(int ipAddress) {
        StringBuffer sb = new StringBuffer();
        int result = 0;
        for (int i = 0; i < 4; ++i) {
```

```
            result = (ipAddress >> ((3-i)*8)) & 0xff;
            sb.append(Integer.valueOf(result).toString());
            if (i != 3)
                sb.append(".");
        }
        return sb.toString();
    }


    public static String fromIPv4Address(byte[] ipAddress) {
        return fromIPv4Address(IPv4.bytesToInt(ipAddress));
    }


    public static int bytesToInt(byte[] addressByteArray) {
        int retval = 0;
        for (int i = 0; i < 4; ++i) {
            retval |= (addressByteArray[i] & 0xff) << 8*(3-i);
        }
        return retval;
    }


    public static byte[] intToBytes(int address) {
        byte[] bytes = new byte[4];
        for (int i = 0; i < 4; ++i) {
            bytes[i] = (byte) (address >>> (8*(3-i)));
        }
        return bytes;
    }

    public static String fromIPv4AddressCollection(Collection<Integer> ipAddresses) {
        if (ipAddresses == null)
            return "null";
        StringBuffer sb = new StringBuffer();
        sb.append("[");
        for (Integer ip : ipAddresses) {
            sb.append(fromIPv4Address(ip));
            sb.append(",");
        }
        sb.replace(sb.length()-1, sb.length(), "]");
        return sb.toString();
    }


    public static byte[] toIPv4AddressBytes(String ipAddress) {
        String[] octets = ipAddress.split("\\.");
        if (octets.length != 4)
            throw new IllegalArgumentException("Specified IPv4 address must" +
                "contain 4 sets of numerical digits separated by periods");

        byte[] result = new byte[4];
        for (int i = 0; i < 4; ++i) {
            result[i] = Integer.valueOf(octets[i]).byteValue();
        }
        return result;
    }

    public boolean payloadIsICMP() {
        return this.protocol == PROTOCOL_ICMP;
    }


    public boolean payloadIsUDP() {
        return this.protocol == PROTOCOL_UDP;
    }


    @Override
    public int hashCode() {
        final int prime = 2521;
        int result = super.hashCode();
        result = prime * result + checksum;
        result = prime * result + destinationAddress;
        result = prime * result + diffServ;
        result = prime * result + flags;
        result = prime * result + fragmentOffset;
```

```
        result = prime * result + headerLength;
        result = prime * result + identification;
        result = prime * result + Arrays.hashCode(options);
        result = prime * result + protocol;
        result = prime * result + sourceAddress;
        result = prime * result + totalLength;
        result = prime * result + ttl;
        result = prime * result + version;
        return result;
    }


    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (!super.equals(obj))
            return false;
        if (!(obj instanceof IPv4))
            return false;
        IPv4 other = (IPv4) obj;
        if (checksum != other.checksum)
            return false;
        if (destinationAddress != other.destinationAddress)
            return false;
        if (diffServ != other.diffServ)
            return false;
        if (flags != other.flags)
            return false;
        if (fragmentOffset != other.fragmentOffset)
            return false;
        if (headerLength != other.headerLength)
            return false;
        if (identification != other.identification)
            return false;
        if (!Arrays.equals(options, other.options))
            return false;
        if (protocol != other.protocol)
            return false;
        if (sourceAddress != other.sourceAddress)
            return false;
        if (totalLength != other.totalLength)
            return false;
        if (ttl != other.ttl)
            return false;
        if (version != other.version)
            return false;
        return true;
    }
        }
```

- OFEchoReply.java

```
package org.openflow.protocol;

public class OFEchoReply extends OFEchoRequest {
    public static int MINIMUM_LENGTH = 8;

    public OFEchoReply() {
        super();
        this.type = OFType.ECHO_REPLY;
        this.length = U16.t(MINIMUM_LENGTH);
    }
}
```

- OFEchoRequest.java

```
package org.openflow.protocol;

public class OFEchoRequest extends OFMessage {
    public static int MINIMUM_LENGTH = 8;
    byte[] payload;

    public OFEchoRequest() {
```

```
        super();
        this.type = OFType.ECHO_REQUEST;
    }

    @Override
    public void readFrom(ByteBuffer bb) {
        super.readFrom(bb);
        int datalen = this.getLengthU() - MINIMUM_LENGTH;
        if (datalen > 0) {
            this.payload = new byte[datalen];
            bb.get(payload);
        }
    }

    public byte[] getPayload() {
        return payload;
    }

    public OFEchoRequest setPayload(byte[] payload) {
        this.payload = payload;
        return this;
    }

    @Override
    public void writeTo(ByteBuffer bb) {
        super.writeTo(bb);
        if (payload != null)
            bb.put(payload);
    }

    @Override
    public void computeLength() {
        this.length = U16.t(MINIMUM_LENGTH + ((payload != null) ? payload.length :
0));
    }
}
```

- OFError.java

```
package org.openflow.protocol;

public class OFError extends OFMessage implements OFMessageFactoryAware {
    public static int MINIMUM_LENGTH = 12;

    public enum OFErrorType {
        OFPET_HELLO_FAILED, OFPET_BAD_REQUEST, OFPET_BAD_ACTION,
OFPET_FLOW_MOD_FAILED, OFPET_PORT_MOD_FAILED, OFPET_QUEUE_OP_FAILED
    }

    public enum OFHelloFailedCode {
        OFPHFC_INCOMPATIBLE, OFPHFC_EPERM
    }

    public enum OFBadRequestCode {
        OFPBRC_BAD_VERSION, OFPBRC_BAD_TYPE, OFPBRC_BAD_STAT, OFPBRC_BAD_VENDOR,
OFPBRC_BAD_SUBTYPE, OFPBRC_EPERM, OFPBRC_BAD_LEN, OFPBRC_BUFFER_EMPTY,
OFPBRC_BUFFER_UNKNOWN
    }

    public enum OFBadActionCode {
        OFPBAC_BAD_TYPE, OFPBAC_BAD_LEN, OFPBAC_BAD_VENDOR, OFPBAC_BAD_VENDOR_TYPE,
OFPBAC_BAD_OUT_PORT, OFPBAC_BAD_ARGUMENT, OFPBAC_EPERM, OFPBAC_TOO_MANY,
OFPBAC_BAD_QUEUE
    }

    public enum OFFlowModFailedCode {
        OFPFMFC_ALL_TABLES_FULL, OFPFMFC_OVERLAP, OFPFMFC_EPERM,
OFPFMFC_BAD_EMERG_TIMEOUT, OFPFMFC_BAD_COMMAND, OFPFMFC_UNSUPPORTED
    }

    public enum OFPortModFailedCode {
        OFPPMFC_BAD_PORT, OFPPMFC_BAD_HW_ADDR
    }

    public enum OFQueueOpFailedCode {
```

```
    OFPQOFC_BAD_PORT, OFPQOFC_BAD_QUEUE, OFPQOFC_EPERM
}

protected short errorType;
protected short errorCode;
protected OFMessageFactory factory;
protected byte[] error;
protected boolean errorIsAscii;

public OFError() {
    super();
    this.type = OFType.ERROR;
    this.length = U16.t(MINIMUM_LENGTH);
}

public short getErrorType() {
    return errorType;
}

public OFError setErrorType(short errorType) {
    this.errorType = errorType;
    return this;
}

public OFError setErrorType(OFErrorType type) {
    this.errorType = (short) type.ordinal();
    return this;
}

public short getErrorCode() {
    return errorCode;
}

public OFError setErrorCode(OFHelloFailedCode code) {
    this.errorCode = (short) code.ordinal();
    return this;
}

public OFError setErrorCode(short errorCode) {
    this.errorCode = errorCode;
    return this;
}

public OFError setErrorCode(OFBadRequestCode code) {
    this.errorCode = (short) code.ordinal();
    return this;
}

public OFError setErrorCode(OFBadActionCode code) {
    this.errorCode = (short) code.ordinal();
    return this;
}

public OFError setErrorCode(OFFlowModFailedCode code) {
    this.errorCode = (short) code.ordinal();
    return this;
}

public OFError setErrorCode(OFPortModFailedCode code) {
    this.errorCode = (short) code.ordinal();
    return this;
}

public OFError setErrorCode(OFQueueOpFailedCode code) {
    this.errorCode = (short) code.ordinal();
    return this;
}

public OFMessage getOffendingMsg() {
    if (this.error == null)
        return null;
    ByteBuffer errorMsg = ByteBuffer.wrap(this.error);
    if (factory == null)
        throw new RuntimeException("MessageFactory not set");
    List<OFMessage> messages = this.factory.parseMessages(errorMsg,
            error.length);
    if (messages.size() > 0)
```

```
            return messages.get(0);
        else
            return null;
    }

    public OFError setOffendingMsg(OFMessage offendingMsg) {
        if (offendingMsg == null) {
            super.setLengthU(MINIMUM_LENGTH);
        } else {
            this.error = new byte[offendingMsg.getLengthU()];
            ByteBuffer data = ByteBuffer.wrap(this.error);
            offendingMsg.writeTo(data);
            super.setLengthU(MINIMUM_LENGTH + offendingMsg.getLengthU());
        }
        return this;
    }

    public OFMessageFactory getFactory() {
        return factory;
    }

    @Override
    public void setMessageFactory(OFMessageFactory factory) {
        this.factory = factory;
    }

    public byte[] getError() {
        return error;
    }

    public OFError setError(byte[] error) {
        this.error = error;
        return this;
    }

    public boolean isErrorIsAscii() {
        return errorIsAscii;
    }

    public OFError setErrorIsAscii(boolean errorIsAscii) {
        this.errorIsAscii = errorIsAscii;
        return this;
    }

    @Override
    public void readFrom(ByteBuffer data) {
        super.readFrom(data);
        this.errorType = data.getShort();
        this.errorCode = data.getShort();
        int dataLength = this.getLengthU() - MINIMUM_LENGTH;
        if (dataLength > 0) {
            this.error = new byte[dataLength];
            data.get(this.error);
            if (this.errorType == OFErrorType.OFPET_HELLO_FAILED.ordinal())
                this.errorIsAscii = true;
        }
    }

    @Override
    public void writeTo(ByteBuffer data) {
        super.writeTo(data);
        data.putShort(errorType);
        data.putShort(errorCode);
        if (error != null)
            data.put(error);
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = super.hashCode();
        result = prime * result + Arrays.hashCode(error);
        result = prime * result + errorCode;
        result = prime * result + (errorIsAscii ? 1231 : 1237);
        result = prime * result + errorType;
        return result;
    }
```

```
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (!super.equals(obj))
            return false;
        if (getClass() != obj.getClass())
            return false;
        OFError other = (OFError) obj;
        if (!Arrays.equals(error, other.error))
            return false;
        if (errorCode != other.errorCode)
            return false;
        if (errorIsAscii != other.errorIsAscii)
            return false;
        if (errorType != other.errorType)
            return false;
        return true;
    }

    @Override
    public void computeLength() {
        this.length = U16.t(MINIMUM_LENGTH + ((error != null) ? error.length : 0));
    }
    }
```

- OFFeaturesReply.java

```
package org.openflow.protocol;

public class OFFeaturesReply extends OFMessage {
    public static int MINIMUM_LENGTH = 32;

    public enum OFCapabilities {
        OFPC_FLOW_STATS     (1 << 0),
        OFPC_TABLE_STATS    (1 << 1),
        OFPC_PORT_STATS     (1 << 2),
        OFPC_STP            (1 << 3),
        OFPC_RESERVED       (1 << 4),
        OFPC_IP_REASM       (1 << 5),
        OFPC_QUEUE_STATS    (1 << 6),
        OFPC_ARP_MATCH_IP   (1 << 7);

        protected int value;

        private OFCapabilities(int value) {
            this.value = value;
        }

        public int getValue() {
            return value;
        }
    }

    protected long datapathId;
    protected int buffers;
    protected byte tables;
    protected int capabilities;
    protected int actions;
    protected List<OFPhysicalPort> ports;

    public OFFeaturesReply() {
        super();
        this.type = OFType.FEATURES_REPLY;
        this.length = U16.t(MINIMUM_LENGTH);
    }

    public long getDatapathId() {
        return datapathId;
    }
    public OFFeaturesReply setDatapathId(long datapathId) {
        this.datapathId = datapathId;
        return this;
```

```
}

public int getBuffers() {
    return buffers;
}

public OFFeaturesReply setBuffers(int buffers) {
    this.buffers = buffers;
    return this;
}

public byte getTables() {
    return tables;
}

public OFFeaturesReply setTables(byte tables) {
    this.tables = tables;
    return this;
}

public int getCapabilities() {
    return capabilities;
}

public OFFeaturesReply setCapabilities(int capabilities) {
    this.capabilities = capabilities;
    return this;
}

public int getActions() {
    return actions;
}

public OFFeaturesReply setActions(int actions) {
    this.actions = actions;
    return this;
}


public List<OFPhysicalPort> getPorts() {
    return ports;
}

public Map<Short, OFPhysicalPort> getPortMap() {
    Map<Short, OFPhysicalPort> map = new HashMap<Short, OFPhysicalPort>();
    if (this.ports != null) {
        for (OFPhysicalPort port : this.ports) {
            map.put(port.getPortNumber(), port);
        }
    }
    return map;
}

public OFFeaturesReply setPorts(List<OFPhysicalPort> ports) {
    this.ports = ports;
    if (ports == null) {
        this.setLengthU(MINIMUM_LENGTH);
    } else {
        this.setLengthU(MINIMUM_LENGTH + ports.size()
                * OFPhysicalPort.MINIMUM_LENGTH);
    }
    return this;
}

@Override
public void readFrom(ByteBuffer data) {
    super.readFrom(data);
    this.datapathId = data.getLong();
    this.buffers = data.getInt();
    this.tables = data.get();
    data.position(data.position() + 3); // pad
    this.capabilities = data.getInt();
    this.actions = data.getInt();
    if (this.ports == null) {
        this.ports = new ArrayList<OFPhysicalPort>();
    } else {
        this.ports.clear();
```

```
        }
        int portCount = (super.getLengthU() - 32)
                / OFPhysicalPort.MINIMUM_LENGTH;
        OFPhysicalPort port;
        for (int i = 0; i < portCount; ++i) {
            port = new OFPhysicalPort();
            port.readFrom(data);
            this.ports.add(port);
        }
    }

    @Override
    public void writeTo(ByteBuffer data) {
        super.writeTo(data);
        data.putLong(this.datapathId);
        data.putInt(this.buffers);
        data.put(this.tables);
        data.putShort((short) 0); // pad
        data.put((byte) 0); // pad
        data.putInt(this.capabilities);
        data.putInt(this.actions);
        if (this.ports != null)
            for (OFPhysicalPort port : this.ports) {
                port.writeTo(data);
            }
    }

    @Override
    public int hashCode() {
        final int prime = 139;
        int result = super.hashCode();
        result = prime * result + actions;
        result = prime * result + buffers;
        result = prime * result + capabilities;
        result = prime * result + (int) (datapathId ^ (datapathId >>> 32));
        result = prime * result + ((ports == null) ? 0 : ports.hashCode());
        result = prime * result + tables;
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (!super.equals(obj)) {
            return false;
        }
        if (!(obj instanceof OFFeaturesReply)) {
            return false;
        }
        OFFeaturesReply other = (OFFeaturesReply) obj;
        if (actions != other.actions) {
            return false;
        }
        if (buffers != other.buffers) {
            return false;
        }
        if (capabilities != other.capabilities) {
            return false;
        }
        if (datapathId != other.datapathId) {
            return false;
        }
        if (ports == null) {
            if (other.ports != null) {
                return false;
            }
        } else if (!ports.equals(other.ports)) {
            return false;
        }
        if (tables != other.tables) {
            return false;
        }
        return true;
    }
```

```
    @Override
    public void computeLength() {
        this.length = U16.t(MINIMUM_LENGTH + ((ports != null) ? (ports.size() *
OFPhysicalPort.MINIMUM_LENGTH) : 0));
    }
}
```

- OFGetConfigReply.java

```
package org.openflow.protocol;

public class OFGetConfigReply extends OFSwitchConfig {
    public OFGetConfigReply() {
        super();
        this.type = OFType.GET_CONFIG_REPLY;
    }
}
```

- OFMatch.java

```
package org.openflow.protocol;

public class OFMatch implements Cloneable, Serializable {

    private static final long serialVersionUID = 1L;
    public static int MINIMUM_LENGTH = 40;

    final public static int OFPFW_ALL = ((1 << 22) - 1);

    final public static int OFPFW_IN_PORT = 1 << 0;
    final public static int OFPFW_DL_VLAN = 1 << 1;
    final public static int OFPFW_DL_SRC = 1 << 2;
    final public static int OFPFW_DL_DST = 1 << 3;

    final public static int OFPFW_DL_TYPE = 1 << 4;
    final public static int OFPFW_NW_PROTO = 1 << 5;
    final public static int OFPFW_TP_SRC = 1 << 6;
    final public static int OFPFW_TP_DST = 1 << 7;

    final public static int OFPFW_NW_SRC_SHIFT = 8;
    final public static int OFPFW_NW_SRC_BITS = 6;
    final public static int OFPFW_NW_SRC_MASK = ((1 << OFPFW_NW_SRC_BITS) - 1) <<
OFPFW_NW_SRC_SHIFT;
    final public static int OFPFW_NW_SRC_ALL = 32 << OFPFW_NW_SRC_SHIFT;

    final public static int OFPFW_NW_DST_SHIFT = 14;
    final public static int OFPFW_NW_DST_BITS = 6;
    final public static int OFPFW_NW_DST_MASK = ((1 << OFPFW_NW_DST_BITS) - 1) <<
OFPFW_NW_DST_SHIFT;
    final public static int OFPFW_NW_DST_ALL = 32 << OFPFW_NW_DST_SHIFT;

    final public static int OFPFW_DL_VLAN_PCP = 1 << 20;
    final public static int OFPFW_NW_TOS = 1 << 21;

    public static final short OFP_VLAN_NONE = (short) 0xffff;

    final public static String STR_IN_PORT = "in_port";
    final public static String STR_DL_DST = "dl_dst";
    final public static String STR_DL_SRC = "dl_src";
    final public static String STR_DL_TYPE = "dl_type";
    final public static String STR_DL_VLAN = "dl_vlan";
    final public static String STR_DL_VLAN_PCP = "dl_vpcp";
    final public static String STR_NW_DST = "nw_dst";
    final public static String STR_NW_SRC = "nw_src";
    final public static String STR_NW_PROTO = "nw_proto";
    final public static String STR_NW_TOS = "nw_tos";
    final public static String STR_TP_DST = "tp_dst";
    final public static String STR_TP_SRC = "tp_src";

    protected int wildcards;
    protected short inputPort;
    protected byte[] dataLayerSource;
    protected byte[] dataLayerDestination;
    protected short dataLayerVirtualLan;
    protected byte dataLayerVirtualLanPriorityCodePoint;
```

```
protected short dataLayerType;
protected byte networkTypeOfService;
protected byte networkProtocol;
protected int networkSource;
protected int networkDestination;
protected short transportSource;
protected short transportDestination;

public OFMatch() {
    this.wildcards = OFPFW_ALL;
    this.dataLayerDestination = new byte[6];
    this.dataLayerSource = new byte[6];
}

public byte[] getDataLayerDestination() {
    return this.dataLayerDestination;
}

public OFMatch setDataLayerDestination(byte[] dataLayerDestination) {
    this.dataLayerDestination = dataLayerDestination;
    return this;
}

public OFMatch setDataLayerDestination(String mac) {
    byte bytes[] = HexString.fromHexString(mac);
    if (bytes.length != 6)
        throw new IllegalArgumentException(
                "expected string with 6 octets, got '" + mac + "'");
    this.dataLayerDestination = bytes;
    return this;
}

public byte[] getDataLayerSource() {
    return this.dataLayerSource;
}

public OFMatch setDataLayerSource(byte[] dataLayerSource) {
    this.dataLayerSource = dataLayerSource;
    return this;
}

public OFMatch setDataLayerSource(String mac) {
    byte bytes[] = HexString.fromHexString(mac);
    if (bytes.length != 6)
        throw new IllegalArgumentException(
                "expected string with 6 octets, got '" + mac + "'");
    this.dataLayerSource = bytes;
    return this;
}

public short getDataLayerType() {
    return this.dataLayerType;
}

public OFMatch setDataLayerType(short dataLayerType) {
    this.dataLayerType = dataLayerType;
    return this;
}

public short getDataLayerVirtualLan() {
    return this.dataLayerVirtualLan;
}

public OFMatch setDataLayerVirtualLan(short dataLayerVirtualLan) {
    this.dataLayerVirtualLan = dataLayerVirtualLan;
    return this;
}

public byte getDataLayerVirtualLanPriorityCodePoint() {
    return this.dataLayerVirtualLanPriorityCodePoint;
}

public OFMatch setDataLayerVirtualLanPriorityCodePoint(byte pcp) {
    this.dataLayerVirtualLanPriorityCodePoint = pcp;
    return this;
}
```

```
public short getInputPort() {
    return this.inputPort;
}

public OFMatch setInputPort(short inputPort) {
    this.inputPort = inputPort;
    return this;
}

public int getNetworkDestination() {
    return this.networkDestination;
}

public OFMatch setNetworkDestination(int networkDestination) {
    this.networkDestination = networkDestination;
    return this;
}

public int getNetworkDestinationMaskLen() {
    return Math
            .max(32 - ((wildcards & OFPFW_NW_DST_MASK) >> OFPFW_NW_DST_SHIFT),
                    0);
}

public int getNetworkSourceMaskLen() {
    return Math
            .max(32 - ((wildcards & OFPFW_NW_SRC_MASK) >> OFPFW_NW_SRC_SHIFT),
                    0);
}

public byte getNetworkProtocol() {
    return this.networkProtocol;
}

public OFMatch setNetworkProtocol(byte networkProtocol) {
    this.networkProtocol = networkProtocol;
    return this;
}

public int getNetworkSource() {
    return this.networkSource;
}

public OFMatch setNetworkSource(int networkSource) {
    this.networkSource = networkSource;
    return this;
}

public byte getNetworkTypeOfService() {
    return this.networkTypeOfService;
}

public OFMatch setNetworkTypeOfService(byte networkTypeOfService) {
    this.networkTypeOfService = networkTypeOfService;
    return this;
}

public short getTransportDestination() {
    return this.transportDestination;
}

public OFMatch setTransportDestination(short transportDestination) {
    this.transportDestination = transportDestination;
    return this;
}

public short getTransportSource() {
    return this.transportSource;
}

public OFMatch setTransportSource(short transportSource) {
    this.transportSource = transportSource;
    return this;
}

public int getWildcards() {
    return this.wildcards;
```

```
    }

    public OFMatch setWildcards(int wildcards) {
        this.wildcards = wildcards;
        return this;
    }

    public static OFMatch load(byte[] packetData, short inputPort) {
        OFMatch ofm = new OFMatch();
        return ofm.loadFromPacket(packetData, inputPort);
    }

    public OFMatch loadFromPacket(byte[] packetData, short inputPort) {
        short scratch;
        int transportOffset = 34;
        ByteBuffer packetDataBB = ByteBuffer.wrap(packetData);
        int limit = packetDataBB.limit();

        this.wildcards = 0; // all fields have explicit entries

        this.inputPort = inputPort;

        if (inputPort == OFPort.OFPP_ALL.getValue())
            this.wildcards |= OFPFW_IN_PORT;

        assert (limit >= 14);
        this.dataLayerDestination = new byte[6];
        packetDataBB.get(this.dataLayerDestination);
        this.dataLayerSource = new byte[6];
        packetDataBB.get(this.dataLayerSource);
        this.dataLayerType = packetDataBB.getShort();

        if (getDataLayerType() != (short) 0x8100) {
            setDataLayerVirtualLan((short) 0xffff);
            setDataLayerVirtualLanPriorityCodePoint((byte) 0);
        } else {

            scratch = packetDataBB.getShort();
            setDataLayerVirtualLan((short) (0xfff & scratch));
            setDataLayerVirtualLanPriorityCodePoint((byte) ((0xe000 & scratch) >>
13));
            this.dataLayerType = packetDataBB.getShort();
        }

        switch (getDataLayerType()) {
        case 0x0800:

            scratch = packetDataBB.get();
            scratch = (short) (0xf & scratch);
            transportOffset = (packetDataBB.position() - 1) + (scratch * 4);
            scratch = packetDataBB.get();
            setNetworkTypeOfService((byte) ((0xfc & scratch) >> 2));
            packetDataBB.position(packetDataBB.position() + 7);
            this.networkProtocol = packetDataBB.get();
            packetDataBB.position(packetDataBB.position() + 2);
            this.networkSource = packetDataBB.getInt();
            this.networkDestination = packetDataBB.getInt();
            packetDataBB.position(transportOffset);
            break;
        case 0x0806:

           int arpPos = packetDataBB.position();
            scratch = packetDataBB.getShort(arpPos + 6);
            setNetworkProtocol((byte) (0xff & scratch));

            scratch = packetDataBB.getShort(arpPos + 2);
            if (scratch == 0x800 && packetDataBB.get(arpPos + 5) == 4) {
                this.networkSource = packetDataBB.getInt(arpPos + 14);
                this.networkDestination = packetDataBB.getInt(arpPos + 24);
            } else {
                setNetworkSource(0);
                setNetworkDestination(0);
            }
            break;
        default:
            setNetworkTypeOfService((byte) 0);
            setNetworkProtocol((byte) 0);
```

```
            setNetworkSource(0);
            setNetworkDestination(0);
            break;
        }

        switch (getNetworkProtocol()) {
        case 0x01:
            this.transportSource = U8.f(packetDataBB.get());
            this.transportDestination = U8.f(packetDataBB.get());
            break;
        case 0x06:
            this.transportSource = packetDataBB.getShort();
            this.transportDestination = packetDataBB.getShort();
            break;
        case 0x11:
            this.transportSource = packetDataBB.getShort();
            this.transportDestination = packetDataBB.getShort();
            break;
        default:
            setTransportDestination((short) 0);
            setTransportSource((short) 0);
            break;
        }
        return this;
    }

    public void readFrom(ByteBuffer data) {
        this.wildcards = data.getInt();
        this.inputPort = data.getShort();
        this.dataLayerSource = new byte[6];
        data.get(this.dataLayerSource);
        this.dataLayerDestination = new byte[6];
        data.get(this.dataLayerDestination);
        this.dataLayerVirtualLan = data.getShort();
        this.dataLayerVirtualLanPriorityCodePoint = data.get();
        data.get(); // pad
        this.dataLayerType = data.getShort();
        this.networkTypeOfService = data.get();
        this.networkProtocol = data.get();
        data.get(); // pad
        data.get(); // pad
        this.networkSource = data.getInt();
        this.networkDestination = data.getInt();
        this.transportSource = data.getShort();
        this.transportDestination = data.getShort();
    }

    public void writeTo(ByteBuffer data) {
        data.putInt(wildcards);
        data.putShort(inputPort);
        data.put(this.dataLayerSource);
        data.put(this.dataLayerDestination);
        data.putShort(dataLayerVirtualLan);
        data.put(dataLayerVirtualLanPriorityCodePoint);
        data.put((byte) 0x0); // pad
        data.putShort(dataLayerType);
        data.put(networkTypeOfService);
        data.put(networkProtocol);
        data.put((byte) 0x0); // pad
        data.put((byte) 0x0); // pad
        data.putInt(networkSource);
        data.putInt(networkDestination);
        data.putShort(transportSource);
        data.putShort(transportDestination);
    }

    @Override
    public int hashCode() {
        final int prime = 131;
        int result = 1;
        result = prime * result + Arrays.hashCode(dataLayerDestination);
        result = prime * result + Arrays.hashCode(dataLayerSource);
        result = prime * result + dataLayerType;
        result = prime * result + dataLayerVirtualLan;
        result = prime * result + dataLayerVirtualLanPriorityCodePoint;
        result = prime * result + inputPort;
        result = prime * result + networkDestination;
```

```
        result = prime * result + networkProtocol;
        result = prime * result + networkSource;
        result = prime * result + networkTypeOfService;
        result = prime * result + transportDestination;
        result = prime * result + transportSource;
        result = prime * result + canonicalizeWildcards(wildcards);
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (!(obj instanceof OFMatch)) {
            return false;
        }
        OFMatch other = (OFMatch) obj;
        if (!Arrays.equals(dataLayerDestination, other.dataLayerDestination)) {
            return false;
        }
        if (!Arrays.equals(dataLayerSource, other.dataLayerSource)) {
            return false;
        }
        if (dataLayerType != other.dataLayerType) {
            return false;
        }
        if (dataLayerVirtualLan != other.dataLayerVirtualLan) {
            return false;
        }
        if (dataLayerVirtualLanPriorityCodePoint !=
other.dataLayerVirtualLanPriorityCodePoint) {
            return false;
        }
        if (inputPort != other.inputPort) {
            return false;
        }
        if (networkDestination != other.networkDestination) {
            return false;
        }
        if (networkProtocol != other.networkProtocol) {
            return false;
        }
        if (networkSource != other.networkSource) {
            return false;
        }
        if (networkTypeOfService != other.networkTypeOfService) {
            return false;
        }
        if (transportDestination != other.transportDestination) {
            return false;
        }
        if (transportSource != other.transportSource) {
            return false;
        }
        if (canonicalizeWildcards((wildcards & OFMatch.OFPFW_ALL)) !=
                canonicalizeWildcards((other.wildcards & OFPFW_ALL))) { // only
            return false;
        }
        return true;
    }

    @Override
    public OFMatch clone() {
        try {
            OFMatch ret = (OFMatch) super.clone();
            ret.dataLayerDestination = this.dataLayerDestination.clone();
            ret.dataLayerSource = this.dataLayerSource.clone();
            return ret;
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException(e);
        }
    }
```

```
@Override
public String toString() {
    String str = "";

    if ((wildcards & OFPFW_IN_PORT) == 0)
        str += "," + STR_IN_PORT + "=" + U16.f(this.inputPort);

    if ((wildcards & OFPFW_DL_DST) == 0)
        str += "," + STR_DL_DST + "="
                + HexString.toHexString(this.dataLayerDestination);
    if ((wildcards & OFPFW_DL_SRC) == 0)
        str += "," + STR_DL_SRC + "="
                + HexString.toHexString(this.dataLayerSource);
    if ((wildcards & OFPFW_DL_TYPE) == 0)
        str += "," + STR_DL_TYPE + "=0x"
                + Integer.toHexString(U16.f(this.dataLayerType));
    if ((wildcards & OFPFW_DL_VLAN) == 0)
        str += "," + STR_DL_VLAN + "=0x"
                + Integer.toHexString(U16.f(this.dataLayerVirtualLan));
    if ((wildcards & OFPFW_DL_VLAN_PCP) == 0)
        str += ","
                + STR_DL_VLAN_PCP
                + "="
                + Integer.toHexString(U8
                        .f(this.dataLayerVirtualLanPriorityCodePoint));

    if (getNetworkDestinationMaskLen() > 0)
        str += ","
                + STR_NW_DST
                + "="
                + cidrToString(networkDestination,
                        getNetworkDestinationMaskLen());
    if (getNetworkSourceMaskLen() > 0)
        str += "," + STR_NW_SRC + "="
                + cidrToString(networkSource, getNetworkSourceMaskLen());
    if ((wildcards & OFPFW_NW_PROTO) == 0)
        str += "," + STR_NW_PROTO + "=" + U8.f(this.networkProtocol);
    if ((wildcards & OFPFW_NW_TOS) == 0)
        str += "," + STR_NW_TOS + "=" + U8.f(this.networkTypeOfService);

    if ((wildcards & OFPFW_TP_DST) == 0)
        str += "," + STR_TP_DST + "=" + U16.f(this.transportDestination);
    if ((wildcards & OFPFW_TP_SRC) == 0)
        str += "," + STR_TP_SRC + "=" + U16.f(this.transportSource);
    if ((str.length() > 0) && (str.charAt(0) == ','))
        str = str.substring(1);
    return "OFMatch[" + str + "]";
}

private String cidrToString(int ip, int prefix) {
    String str;
    if (prefix >= 32) {
        str = ipToString(ip);
    } else {
        int mask = ~((1 << (32 - prefix)) - 1);
        str = ipToString(ip & mask) + "/" + prefix;
    }

    return str;
}


public void fromString(String match) throws IllegalArgumentException {
    if (match.equals("") || match.equalsIgnoreCase("any")
            || match.equalsIgnoreCase("all") || match.equals("[]"))
        match = "OFMatch[]";
    String[] tokens = match.split("[\\[,\\]]");
    String[] values;
    int initArg = 0;
    if (tokens[0].equals("OFMatch"))
        initArg = 1;
    this.wildcards = OFPFW_ALL;
    int i;
    for (i = initArg; i < tokens.length; i++) {
        values = tokens[i].split("=");
        if (values.length != 2)
            throw new IllegalArgumentException("Token " + tokens[i]
```

```
                    + " does not have form 'key=value' parsing " + match);
            values[0] = values[0].toLowerCase(); // try to make this case insens
            if (values[0].equals(STR_IN_PORT) || values[0].equals("input_port")) {
                this.inputPort = U16.t(Integer.valueOf(values[1]));
                this.wildcards &= ~OFPFW_IN_PORT;
            } else if (values[0].equals(STR_DL_DST)
                    || values[0].equals("eth_dst")) {
                this.dataLayerDestination = HexString.fromHexString(values[1]);
                this.wildcards &= ~OFPFW_DL_DST;
            } else if (values[0].equals(STR_DL_SRC)
                    || values[0].equals("eth_src")) {
                this.dataLayerSource = HexString.fromHexString(values[1]);
                this.wildcards &= ~OFPFW_DL_SRC;
            } else if (values[0].equals(STR_DL_TYPE)
                    || values[0].equals("eth_type")) {
                if (values[1].startsWith("0x"))
                    this.dataLayerType = U16.t(Integer.valueOf(
                            values[1].replaceFirst("0x", ""), 16));
                else
                    this.dataLayerType = U16.t(Integer.valueOf(values[1]));
                this.wildcards &= ~OFPFW_DL_TYPE;
            } else if (values[0].equals(STR_DL_VLAN)) {
                if (values[1].contains("0x")) {
                    this.dataLayerVirtualLan = U16.t(Integer.valueOf(
                            values[1].replaceFirst("0x", ""), 16));
                } else {
                    this.dataLayerVirtualLan = U16.t(Integer.valueOf(values[1]));
                }
                this.wildcards &= ~OFPFW_DL_VLAN;
            } else if (values[0].equals(STR_DL_VLAN_PCP)) {
                this.dataLayerVirtualLanPriorityCodePoint = U8.t(Short
                        .valueOf(values[1]));
                this.wildcards &= ~OFPFW_DL_VLAN_PCP;
            } else if (values[0].equals(STR_NW_DST)
                    || values[0].equals("ip_dst"))
                setFromCIDR(values[1], STR_NW_DST);
            else if (values[0].equals(STR_NW_SRC) || values[0].equals("ip_src"))
                setFromCIDR(values[1], STR_NW_SRC);
            else if (values[0].equals(STR_NW_PROTO)) {
                this.networkProtocol = U8.t(Short.valueOf(values[1]));
                this.wildcards &= ~OFPFW_NW_PROTO;
            } else if (values[0].equals(STR_NW_TOS)) {
                this.networkTypeOfService = U8.t(Short.valueOf(values[1]));
                this.wildcards &= ~OFPFW_NW_TOS;
            } else if (values[0].equals(STR_TP_DST)) {
                this.transportDestination = U16.t(Integer.valueOf(values[1]));
                this.wildcards &= ~OFPFW_TP_DST;
            } else if (values[0].equals(STR_TP_SRC)) {
                this.transportSource = U16.t(Integer.valueOf(values[1]));
                this.wildcards &= ~OFPFW_TP_SRC;
            } else
                throw new IllegalArgumentException("unknown token " + tokens[i]
                        + " parsing " + match);
        }
    }


    private void setFromCIDR(String cidr, String which)
            throws IllegalArgumentException {
        String values[] = cidr.split("/");
        String[] ip_str = values[0].split("\\.");
        int ip = 0;
        ip += Integer.valueOf(ip_str[0]) << 24;
        ip += Integer.valueOf(ip_str[1]) << 16;
        ip += Integer.valueOf(ip_str[2]) << 8;
        ip += Integer.valueOf(ip_str[3]);
        int prefix = 32;

        if (values.length >= 2)
            prefix = Integer.valueOf(values[1]);
        int mask = 32 - prefix;
        if (which.equals(STR_NW_DST)) {
            this.networkDestination = ip;
            this.wildcards = (wildcards & ~OFPFW_NW_DST_MASK)
                    | (mask << OFPFW_NW_DST_SHIFT);
        } else if (which.equals(STR_NW_SRC)) {
            this.networkSource = ip;
```

```
            this.wildcards = (wildcards & ~OFPFW_NW_SRC_MASK)
                    | (mask << OFPFW_NW_SRC_SHIFT);
        }
    }

    public static String ipToString(int ip) {
        return Integer.toString(U8.f((byte) ((ip & 0xff000000) >> 24))) + "."
                + Integer.toString((ip & 0x00ff0000) >> 16) + "."
                + Integer.toString((ip & 0x0000ff00) >> 8) + "."
                + Integer.toString(ip & 0x000000ff);
    }

    /**
    public OFMatch reverse(short inputPort, boolean wildcardInputPort) {
        OFMatch ret = this.clone();

        // Set the input port
        if (wildcardInputPort) {
            ret.inputPort = 0;
            ret.wildcards |= OFPFW_IN_PORT;
        } else {
            ret.inputPort = inputPort;
            ret.wildcards &= ~OFPFW_IN_PORT;
        }

        ret.dataLayerDestination = this.dataLayerSource.clone();
        ret.dataLayerSource = this.dataLayerDestination.clone();

        ret.networkDestination = this.networkSource;
        ret.networkSource = this.networkDestination;

        ret.transportDestination = this.transportSource;
        ret.transportSource = this.transportDestination;

        ret.wildcards &= ~(OFPFW_DL_DST | OFPFW_DL_SRC |
                OFPFW_NW_DST_MASK | OFPFW_NW_SRC_MASK |
                OFPFW_TP_DST | OFPFW_TP_SRC);
        ret.wildcards |= ((this.wildcards & OFPFW_DL_DST) != 0 ) ? OFPFW_DL_SRC : 0;
        ret.wildcards |= ((this.wildcards & OFPFW_DL_SRC) != 0 ) ? OFPFW_DL_DST : 0;
        ret.wildcards |= (((this.wildcards & OFPFW_NW_DST_MASK) >> OFPFW_NW_DST_SHIFT)
<< OFPFW_NW_SRC_SHIFT);
        ret.wildcards |= (((this.wildcards & OFPFW_NW_SRC_MASK) >> OFPFW_NW_SRC_SHIFT)
<< OFPFW_NW_DST_SHIFT);
        ret.wildcards |= ((this.wildcards & OFPFW_TP_DST) != 0 ) ? OFPFW_TP_SRC : 0;
        ret.wildcards |= ((this.wildcards & OFPFW_TP_SRC) != 0 ) ? OFPFW_TP_DST : 0;

        return ret;
    }

    public boolean subsumes(OFMatch match) {
        if ((wildcards & OFPFW_IN_PORT) == 0) {
            if (inputPort != match.inputPort) {
                return false;
            }
        }

        if ((wildcards & OFPFW_DL_DST) == 0) {
            if (!Arrays.equals(dataLayerDestination, match.dataLayerDestination)) {
                return false;
            }
        }
        if ((wildcards & OFPFW_DL_SRC) == 0) {
            if (!Arrays.equals(dataLayerSource, match.dataLayerSource)) {
                return false;
            }
        }
        if ((wildcards & OFPFW_DL_TYPE) == 0) {
            if (dataLayerType != match.dataLayerType) {
                return false;
            }
        }
        if ((wildcards & OFPFW_DL_VLAN) == 0) {
            if (dataLayerVirtualLan!= match.dataLayerVirtualLan) {
                return false;
            }
        }
        if ((wildcards & OFPFW_DL_VLAN_PCP) == 0) {
```

```
            if (dataLayerVirtualLanPriorityCodePoint !=
match.dataLayerVirtualLanPriorityCodePoint) {
                return false;
            }
        }

        int maskLen = getNetworkDestinationMaskLen();
        if (maskLen > match.getNetworkDestinationMaskLen()) {
            return false;
        }
        int mask = (maskLen == 0) ? 0 : (0xffffffff << (32 - maskLen));
        if ((networkDestination & mask) != (match.networkDestination & mask)) {
            return false;
        }
        maskLen = getNetworkSourceMaskLen();
        if (maskLen > match.getNetworkSourceMaskLen()) {
            return false;
        }
        mask = (maskLen == 0) ? 0 : (0xffffffff << (32 - maskLen));
        if ((networkSource & mask) != (match.networkSource & mask)) {
            return false;
        }
        if ((wildcards & OFPFW_NW_PROTO) == 0) {
            if (networkProtocol != match.networkProtocol) {
                return false;
            }
        }
        if ((wildcards & OFPFW_NW_TOS) == 0) {
            if (networkTypeOfService != match.networkTypeOfService) {
                return false;
            }
        }

        if ((wildcards & OFPFW_TP_DST) == 0) {
            if (transportDestination != match.transportDestination) {
                return false;
            }
        }
        if ((wildcards & OFPFW_TP_SRC) == 0) {
            if (transportSource != match.transportSource) {
                return false;
            }
        }

        return true;
    }

    protected int canonicalizeWildcards(int wc) {
        if ((wc & OFPFW_NW_SRC_ALL) != 0)
            wc = (wc & ~OFPFW_NW_SRC_MASK) | OFPFW_NW_SRC_ALL;
        if ((wc & OFPFW_NW_DST_ALL) != 0)
            wc = (wc & ~OFPFW_NW_DST_MASK) | OFPFW_NW_DST_ALL;
        return wc;
    }
}
```

- OFPort.java

```
package org.openflow.protocol;

public enum OFPort {
    OFPP_MAX                ((short)0xff00),
    OFPP_IN_PORT            ((short)0xfff8),
    OFPP_TABLE              ((short)0xfff9),
    OFPP_NORMAL             ((short)0xfffa),
    OFPP_FLOOD              ((short)0xfffb),
    OFPP_ALL                ((short)0xfffc),
    OFPP_CONTROLLER         ((short)0xfffd),
    OFPP_LOCAL              ((short)0xfffe),
    OFPP_NONE               ((short)0xffff);

    protected short value;
```

```
    private OFPort(short value) {
        this.value = value;
    }

    public short getValue() {
        return value;
    }
}
```

- OFSetConfig.java

```
package org.openflow.protocol;

public class OFSetConfig extends OFSwitchConfig {
    public OFSetConfig() {
        super();
        this.type = OFType.SET_CONFIG;
    }
}
```

- OFStatisticsReply.java

```
package org.openflow.protocol;

import java.util.List;

import org.openflow.protocol.statistics.OFStatistics;
import org.openflow.util.U16;

public class OFStatisticsReply extends OFStatisticsMessageBase {
    public enum OFStatisticsReplyFlags {
        REPLY_MORE       (1 << 0);

        protected short type;

        OFStatisticsReplyFlags(int type) {
            this.type = (short) type;
        }

        public short getTypeValue() {
            return type;
        }
    }

    public List<OFStatistics> getStatistics() {
        return statistics;
    }

    public OFStatisticsMessageBase setStatistics(List<OFStatistics> statistics) {
        this.statistics = statistics;
        return this;
    }

    public OFStatisticsReply() {
        super();
        this.type = OFType.STATS_REPLY;
        this.length = U16.t(OFStatisticsMessageBase.MINIMUM_LENGTH);
    }
}
```

- OFStatisticsRequest.java

```
package org.openflow.protocol;

public class OFStatisticsRequest extends OFStatisticsMessageBase {
    public OFStatisticsRequest() {
        super();
        this.type = OFType.STATS_REQUEST;
        this.length = U16.t(OFStatisticsMessageBase.MINIMUM_LENGTH);
    }

    public OFStatistics getStatistics() {
```

```
        if (statistics == null)
            return null;
        else if (statistics.size() == 0)
            return null;
        else
            return statistics.get(0);
    }

    public OFStatisticsRequest setStatistics(OFStatistics statistics) {
        this.statistics = Collections.singletonList(statistics);
        return this;
    }
}
```

- OFType.java

```
package org.openflow.protocol;

public enum OFType {
    HELLO                (0, OFHello.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFHello();
                            }}),
    ERROR                (1, OFError.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFError();
                            }}),
    ECHO_REQUEST         (2, OFEchoRequest.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFEchoRequest();
                            }}),
    ECHO_REPLY           (3, OFEchoReply.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFEchoReply();
                            }}),
    VENDOR               (4, OFVendor.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFVendor();
                            }}),
    FEATURES_REQUEST     (5, OFFeaturesRequest.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFFeaturesRequest();
                            }}),
    FEATURES_REPLY       (6, OFFeaturesReply.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFFeaturesReply();
                            }}),
    GET_CONFIG_REQUEST   (7, OFGetConfigRequest.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFGetConfigRequest();
                            }}),
    GET_CONFIG_REPLY     (8, OFGetConfigReply.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFGetConfigReply();
                            }}),
    SET_CONFIG           (9, OFSetConfig.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFSetConfig();
                            }}),
    PACKET_IN            (10, OFPacketIn.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFPacketIn();
                            }}),
    FLOW_REMOVED         (11, OFFlowRemoved.class, new Instantiable<OFMessage>() {
```

```
                            @Override
                            public OFMessage instantiate() {
                                return new OFFlowRemoved();
                            }}),
    PORT_STATUS             (12, OFPortStatus.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFPortStatus();
                            }}),
    PACKET_OUT              (13, OFPacketOut.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFPacketOut();
                            }}),
    FLOW_MOD                (14, OFFlowMod.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFFlowMod();
                            }}),
    PORT_MOD                (15, OFPortMod.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFPortMod();
                            }}),
    STATS_REQUEST           (16, OFStatisticsRequest.class, new Instantiable<OFMessage>()
{
                            @Override
                            public OFMessage instantiate() {
                                return new OFStatisticsRequest();
                            }}),
    STATS_REPLY             (17, OFStatisticsReply.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFStatisticsReply();
                            }}),
    BARRIER_REQUEST         (18, OFBarrierRequest.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFBarrierRequest();
                            }}),
    BARRIER_REPLY           (19, OFBarrierReply.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFBarrierReply();
                            }}),
    QUEUE_CONFIG_REQUEST    (20, OFMessage.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFQueueConfigRequest();
                            }}),
    QUEUE_CONFIG_REPLY      (21, OFMessage.class, new Instantiable<OFMessage>() {
                            @Override
                            public OFMessage instantiate() {
                                return new OFQueueConfigReply();
                            }}),
    UNKNOWN                 (0xff, OFUnknownMessage.class, new Instantiable<OFMessage>() {
                                @Override
                                public OFMessage instantiate() {
                                    return new OFUnknownMessage();
                            }});

    protected static Logger log = LoggerFactory.getLogger(OFType.class);

    static OFType[] mapping;

    protected Class<? extends OFMessage> clazz;
    protected Constructor<? extends OFMessage> constructor;
    protected Instantiable<OFMessage> instantiable;
    protected byte type;

    OFType(int type, Class<? extends OFMessage> clazz, Instantiable<OFMessage>
instantiator) {
        this.type = (byte) type;
        this.clazz = clazz;
        this.instantiable = instantiator;
        try {
            this.constructor = clazz.getConstructor(new Class[]{});
```

```
        } catch (Exception e) {
            throw new RuntimeException(
                    "Failure getting constructor for class: " + clazz, e);
        }
        OFType.addMapping(this.type, this);
    }

    static public void addMapping(byte i, OFType t) {
        if (mapping == null)
            mapping = new OFType[32];
        if (i >= 0 && i < 32)
            OFType.mapping[i] = t;
    }

    static public void removeMapping(byte i) {
        if (i >= 0 && i < 32)
            OFType.mapping[i] = null;
    }

    static public OFType valueOf(Byte i) {
        try {
            return OFType.mapping[i];
        } catch (ArrayIndexOutOfBoundsException e) {
            log.warn("Unknown message type requested: {}", U8.f(i));
            return UNKNOWN;
        }
    }

    public byte getTypeValue() {
        return this.type;
    }

    public Class<? extends OFMessage> toClass() {
        return clazz;
    }

    public Constructor<? extends OFMessage> getConstructor() {
        return constructor;
    }

    public OFMessage newInstance() {
        return instantiable.instantiate();
    }

    public Instantiable<OFMessage> getInstantiable() {
        return instantiable;
    }

    public void setInstantiable(Instantiable<OFMessage> instantiable) {
        this.instantiable = instantiable;
    }
}
```

Device.java

```
package net.beaconcontroller.devicemanager;

public class Device {
    protected byte[] dataLayerAddress;
    protected long lastSeen;
    protected Set<Integer> networkAddresses;
    protected IOFSwitch sw;
    protected Short swPort;

    public Device() {
        this.networkAddresses = new ConcurrentSkipListSet<Integer>();
    }

    public byte[] getDataLayerAddress() {
        return dataLayerAddress;
    }

    public void setDataLayerAddress(byte[] dataLayerAddress) {
        this.dataLayerAddress = dataLayerAddress;
```

```
    }

    public Short getSwPort() {
        return swPort;
    }

    public void setSwPort(Short swPort) {
        this.swPort = swPort;
    }

    public IOFSwitch getSw() {
        return sw;
    }

    public void setSw(IOFSwitch sw) {
        this.sw = sw;
    }

    public Set<Integer> getNetworkAddresses() {
        return networkAddresses;
    }

    public void setNetworkAddresses(Set<Integer> networkAddresses) {
        this.networkAddresses = networkAddresses;
    }

    @Override
    public int hashCode() {
        final int prime = 2633;
        int result = 1;
        result = prime * result + Arrays.hashCode(dataLayerAddress);
        result = prime
                * result
                + ((networkAddresses == null) ? 0 : networkAddresses.hashCode());
        result = prime * result + ((sw == null) ? 0 : sw.hashCode());
        result = prime * result + ((swPort == null) ? 0 : swPort.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (!(obj instanceof Device))
            return false;
        Device other = (Device) obj;
        if (!Arrays.equals(dataLayerAddress, other.dataLayerAddress))
            return false;
        if (networkAddresses == null) {
            if (other.networkAddresses != null)
                return false;
        } else if (!networkAddresses.equals(other.networkAddresses))
            return false;
        if (sw == null) {
            if (other.sw != null)
                return false;
        } else if (!sw.equals(other.sw))
            return false;
        if (swPort == null) {
            if (other.swPort != null)
                return false;
        } else if (!swPort.equals(other.swPort))
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "Device [dataLayerAddress=" +
                ((dataLayerAddress == null) ? "null" :
HexString.toHexString(dataLayerAddress)) +
                ", swId=" + ((sw == null) ? "null" :
HexString.toHexString(sw.getId())) +
                ", swPort=" + ((swPort == null) ? "null" : (0xffff & swPort)) +
                ", networkAddresses="
```

```
                + IPv4.fromIPv4AddressCollection(networkAddresses) +
                ", lastSeen=" + lastSeen +
                "]";
    }

    public long getLastSeen() {
        return lastSeen;
    }

    public synchronized void setLastSeen(long lastSeen) {
        this.lastSeen = lastSeen;
    }
}
```

- DeviceManagerImpl.java

```
package net.beaconcontroller.devicemanager.internal;

public class DeviceManagerImpl implements IDeviceManager, IOFMessageListener,
        IOFSwitchListener, ITopologyAware {
    protected static Logger log = LoggerFactory.getLogger(DeviceManagerImpl.class);

    protected IBeaconProvider beaconProvider;
    protected Map<Long, Device> dataLayerAddressDeviceMap;
    protected Set<IDeviceManagerAware> deviceManagerAware;
    protected ReentrantReadWriteLock lock;
    protected Map<Integer, Device> networkLayerAddressDeviceMap;
    protected volatile boolean shuttingDown = false;
    protected Map<IOFSwitch, Set<Device>> switchDeviceMap;
    protected Map<SwitchPortTuple, Set<Device>> switchPortDeviceMap;
    protected ITopology topology;
    protected BlockingQueue<Update> updates;
    protected Thread updatesThread;

    protected enum UpdateType {
        ADDED, REMOVED, MOVED, NW_ADDED, NW_REMOVED
    }

    protected class Update {
        public Device device;
        public Integer networkAddress;
        public Set<Integer> networkAddresses;
        public Set<Integer> oldNetworkAddresses;
        public IOFSwitch oldSw;
        public Short oldSwPort;
        public IOFSwitch sw;
        public Short swPort;
        public UpdateType updateType;

        public Update(UpdateType type) {
            this.updateType = type;
        }
    }

    public DeviceManagerImpl() {
        this.dataLayerAddressDeviceMap = new ConcurrentHashMap<Long, Device>();
        this.lock = new ReentrantReadWriteLock();
        this.networkLayerAddressDeviceMap = new ConcurrentHashMap<Integer, Device>();
        this.switchDeviceMap = new ConcurrentHashMap<IOFSwitch, Set<Device>>();
        this.switchPortDeviceMap = new ConcurrentHashMap<SwitchPortTuple,
Set<Device>>();
        this.updates = new LinkedBlockingQueue<Update>();
    }

    public void startUp() {
        beaconProvider.addOFMessageListener(OFType.PACKET_IN, this);
        beaconProvider.addOFMessageListener(OFType.PORT_STATUS, this);
        beaconProvider.addOFSwitchListener(this);

        updatesThread = new Thread(new Runnable () {
            @Override
            public void run() {
                while (true) {
                    try {
                        Update update = updates.take();
```

```
                        if (deviceManagerAware != null) {
                            for (IDeviceManagerAware dma : deviceManagerAware) {
                                try {
                                    switch (update.updateType) {
                                        case ADDED:
                                            dma.deviceAdded(update.device);
                                            break;
                                        case REMOVED:
                                            dma.deviceRemoved(update.device);
                                            break;
                                        case MOVED:
                                            dma.deviceMoved(update.device,
                                                    update.oldSw,
                                                    update.oldSwPort,
                                                    update.sw, update.swPort);
                                            break;
                                        case NW_ADDED:
                                            dma.deviceNetworkAddressAdded(
                                                    update.device,
                                                    update.networkAddresses,
                                                    update.networkAddress);
                                            break;
                                        case NW_REMOVED:
                                            dma.deviceNetworkAddressRemoved(
                                                    update.device,
                                                    update.networkAddresses,
                                                    update.networkAddress);
                                            break;
                                    }
                                } catch (Exception e) {
                                    log.error("Exception in callback", e);
                                }
                            }
                        }
                    } catch (InterruptedException e) {
                        log.warn("DeviceManager Updates thread interupted", e);
                        if (shuttingDown)
                            return;
                    }
                }
            }}, "DeviceManager Updates");
        updatesThread.start();
    }

    public void shutDown() {
        shuttingDown = true;
        beaconProvider.removeOFMessageListener(OFType.PACKET_IN, this);
        beaconProvider.removeOFMessageListener(OFType.PORT_STATUS, this);
        beaconProvider.removeOFSwitchListener(this);
        updatesThread.interrupt();
    }

    @Override
    public String getName() {
        return "devicemanager";
    }

    public Command handlePortStatus(IOFSwitch sw, OFPortStatus ps) {
        if ((byte)OFPortReason.OFPPR_DELETE.ordinal() == ps.getReason() ||
            ((byte)OFPortReason.OFPPR_MODIFY.ordinal() == ps.getReason() &&
                    (((OFPortConfig.OFPPC_PORT_DOWN.getValue() &
ps.getDesc().getConfig()) > 0) ||
                    ((OFPortState.OFPPS_LINK_DOWN.getValue() &
ps.getDesc().getState()) > 0)))) {
            SwitchPortTuple id = new SwitchPortTuple(sw,
ps.getDesc().getPortNumber());
            lock.writeLock().lock();
            try {
                if (switchPortDeviceMap.containsKey(id)) {

                    Set<Device> switchPortDevices = switchPortDeviceMap.remove(id);

                    for (Device device : switchPortDevices) {
                        switchDeviceMap.get(id.getSw()).remove(device);
                        delDevice(device);
                    }
                }
```

```
            } finally {
                lock.writeLock().unlock();
            }
        }
        return Command.CONTINUE;
    }
    protected void delDevice(Device device) {

dataLayerAddressDeviceMap.remove(Ethernet.toLong(device.getDataLayerAddress()));
        if (!device.getNetworkAddresses().isEmpty()) {
            for (Integer nwAddress : device.getNetworkAddresses()) {
                networkLayerAddressDeviceMap.remove(nwAddress);
            }
        }
        updateStatus(device, false);
        if (log.isDebugEnabled()) {
            log.debug("Removed device {}", device);
        }
    }

    @Override
    public Command receive(IOFSwitch sw, OFMessage msg) throws IOException {
        if (msg instanceof OFPortStatus) {
            return handlePortStatus(sw, (OFPortStatus) msg);
        }
        OFPacketIn pi = (OFPacketIn) msg;
        OFMatch match = new OFMatch();
        match.loadFromPacket(pi.getPacketData(), pi.getInPort());

        if ((match.getDataLayerSource()[0] & 0x1) != 0)
            return Command.CONTINUE;

        Long dlAddr = Ethernet.toLong(match.getDataLayerSource());
        Integer nwSrc = match.getNetworkSource();
        Device device = null;
        Device nwDevice = null;
        lock.readLock().lock();
        try {
            device = dataLayerAddressDeviceMap.get(dlAddr);
            nwDevice = networkLayerAddressDeviceMap.get(nwSrc);
        } finally {
            lock.readLock().unlock();
        }
        SwitchPortTuple ipt = new SwitchPortTuple(sw, pi.getInPort());
        if (!topology.isInternal(ipt)) {
            if (device != null) {
                device.setLastSeen(System.currentTimeMillis());

                boolean updateNeeded = false;
                boolean movedLocation = false;
                boolean addedNW = false;
                boolean nwChanged = false;

                if ((!sw.equals(device.getSw()))
                        || (pi.getInPort() != device.getSwPort().shortValue())) {
                    movedLocation = true;
                }
                if (nwDevice == null && nwSrc != 0) {
                    addedNW = true;
                } else if (nwDevice != null && !device.equals(nwDevice)) {
                    nwChanged = true;
                }

                if (movedLocation || addedNW || nwChanged) {
                    updateNeeded = true;
                }

                if (updateNeeded) {
                    lock.writeLock().lock();
                    try {
                        if (movedLocation) {
                            IOFSwitch oldSw = device.getSw();
                            Short oldPort = device.getSwPort();
                            delSwitchDeviceMapping(device.getSw(), device);
                            delSwitchPortDeviceMapping(
                                    new SwitchPortTuple(device.getSw(),
                                            device.getSwPort()), device);
```

```
                            device.setSw(sw);
                            device.setSwPort(pi.getInPort());
                            addSwitchDeviceMapping(device.getSw(), device);
                            addSwitchPortDeviceMapping(
                                    new SwitchPortTuple(device.getSw(),
                                            device.getSwPort()), device);
                            updateMoved(device, oldSw, oldPort, sw,
device.getSwPort());
                            if (log.isDebugEnabled()) {
                                log.debug("Device {} moved from switch: {} port: {} to
switch: {} port: {}",
                                        new Object[] {
                                                device,
                                                HexString.toHexString(oldSw.getId()),
                                                0xffff & oldPort.shortValue(),
                                                HexString.toHexString(sw.getId()),
                                                0xffff & pi.getInPort()});
                            }
                        }
                        if (addedNW) {
                            device.getNetworkAddresses().add(nwSrc);
                            this.networkLayerAddressDeviceMap.put(nwSrc, device);
                            updateNetwork(device, device.getNetworkAddresses(), nwSrc,
true);
                            if (log.isDebugEnabled()) {
                                log.debug("Added IP {} to MAC {}",
                                        IPv4.fromIPv4Address(nwSrc),

HexString.toHexString(device.getDataLayerAddress()));
                            }
                        } else if (nwChanged) {
                            nwDevice.getNetworkAddresses().remove(nwSrc);
                            updateNetwork(nwDevice, nwDevice.getNetworkAddresses(),
nwSrc, false);
                            device.getNetworkAddresses().add(nwSrc);
                            this.networkLayerAddressDeviceMap.put(nwSrc, device);
                            updateNetwork(device, device.getNetworkAddresses(), nwSrc,
true);
                            if (log.isWarnEnabled()) {
                                log.warn(
                                        "IP Address {} changed from MAC {} to {}",
                                        new Object[] {
                                                IPv4.fromIPv4Address(nwSrc),
                                                HexString.toHexString(nwDevice
                                                        .getDataLayerAddress()),
                                                HexString.toHexString(device
                                                        .getDataLayerAddress()) });
                            }
                        }
                    } finally {
                        lock.writeLock().unlock();
                    }
                }
            } else {
                device = new Device();
                device.setDataLayerAddress(match.getDataLayerSource());
                device.setSw(sw);
                device.setSwPort(pi.getInPort());
                device.setLastSeen(System.currentTimeMillis());
                lock.writeLock().lock();
                try {
                    this.dataLayerAddressDeviceMap.put(dlAddr, device);
                    if (nwSrc != 0) {
                        device.getNetworkAddresses().add(nwSrc);
                        this.networkLayerAddressDeviceMap.put(nwSrc, device);
                    }
                    addSwitchDeviceMapping(device.getSw(), device);
                    addSwitchPortDeviceMapping(new SwitchPortTuple(
                            sw, device.getSwPort()), device);
                    if (nwDevice != null) {
                        nwDevice.getNetworkAddresses().remove(nwSrc);
                        updateNetwork(nwDevice, nwDevice.getNetworkAddresses(), nwSrc,
false);
                        if (log.isWarnEnabled()) {
                            log.warn(
                                    "IP Address {} changed from MAC {} to {}",
                                    new Object[] {
```

```
                                                IPv4.fromIPv4Address(nwSrc),
                                                HexString.toHexString(nwDevice
                                                        .getDataLayerAddress()),
                                                HexString.toHexString(device
                                                        .getDataLayerAddress())
});
                            }
                        }
                        updateStatus(device, true);
                        log.debug("New Device: {}", device);
                    } finally {
                        lock.writeLock().unlock();
                    }
                }
            }

            return Command.CONTINUE;
        }

        protected void addSwitchDeviceMapping(IOFSwitch sw, Device device) {
            if (switchDeviceMap.get(sw) == null) {
                switchDeviceMap.put(sw, new HashSet<Device>());
            }
            switchDeviceMap.get(sw).add(device);
        }

        protected void delSwitchDeviceMapping(IOFSwitch sw, Device device) {
            switchDeviceMap.get(sw).remove(device);
            if (switchDeviceMap.get(sw).isEmpty()) {
                switchDeviceMap.remove(sw);
            }
        }

        protected void addSwitchPortDeviceMapping(SwitchPortTuple id, Device device) {
            if (switchPortDeviceMap.get(id) == null) {
                switchPortDeviceMap.put(id, new HashSet<Device>());
            }
            switchPortDeviceMap.get(id).add(device);
        }

        protected void delSwitchPortDeviceMapping(SwitchPortTuple id, Device device) {
            switchPortDeviceMap.get(id).remove(device);
            if (switchPortDeviceMap.get(id).isEmpty()) {
                switchPortDeviceMap.remove(id);
            }
        }

        @Override
        public Device getDeviceByNetworkLayerAddress(Integer address) {
            lock.readLock().lock();
            try {
                return this.networkLayerAddressDeviceMap.get(address);
            } finally {
                lock.readLock().unlock();
            }
        }



        public void setBeaconProvider(IBeaconProvider beaconProvider) {
            this.beaconProvider = beaconProvider;
        }

        public void setTopology(ITopology topology) {
            this.topology = topology;
        }

        @Override
        public Device getDeviceByDataLayerAddress(byte[] address) {
            lock.readLock().lock();
            try {
                return this.dataLayerAddressDeviceMap.get(Ethernet.toLong(address));
            } finally {
                lock.readLock().unlock();
            }
        }
```

```
    @Override
    public List<Device> getDevices() {
        lock.readLock().lock();
        try {
            return new ArrayList<Device>(this.dataLayerAddressDeviceMap.values());
        } finally {
            lock.readLock().unlock();
        }
    }

    @Override
    public void addedSwitch(IOFSwitch sw) {
    }

    @Override
    public void removedSwitch(IOFSwitch sw) {
        lock.writeLock().lock();
        try {
            if (switchDeviceMap.get(sw) != null) {
                for (Iterator<Map.Entry<SwitchPortTuple, Set<Device>>> it =
switchPortDeviceMap
                        .entrySet().iterator(); it.hasNext();) {
                    Map.Entry<SwitchPortTuple, Set<Device>> entry = it.next();
                    if (entry.getKey().getSw().equals(sw)) {
                        it.remove();
                    }
                }

                Set<Device> devices = switchDeviceMap.remove(sw);
                for (Device device : devices) {
                    delDevice(device);
                }
            }
        } finally {
            lock.writeLock().unlock();
        }
    }

    @Override
    public void linkUpdate(IOFSwitch src, short srcPort, IOFSwitch dst,
            short dstPort, boolean added) {
        if (added) {
            SwitchPortTuple id = new SwitchPortTuple(dst, dstPort);
            lock.writeLock().lock();
            try {
                if (switchPortDeviceMap.containsKey(id)) {
                    Set<Device> devices = switchPortDeviceMap.remove(id);
                    for (Device device : devices) {
                        switchDeviceMap.get(id.getSw()).remove(device);
                        delDevice(device);
                    }
                }
            } finally {
                lock.writeLock().unlock();
            }
        }
    }


    public void setDeviceManagerAware(Set<IDeviceManagerAware> deviceManagerAware) {
        this.deviceManagerAware = deviceManagerAware;
    }


    protected void updateStatus(Device device, boolean added) {
        Update update;
        if (added) {
            update = new Update(UpdateType.ADDED);
        } else {
            update = new Update(UpdateType.REMOVED);
        }
        update.device = device;
        this.updates.add(update);
    }

    protected void updateMoved(Device device, IOFSwitch oldSw, Short oldPort,
            IOFSwitch sw, Short port) {
```

```
        Update update = new Update(UpdateType.MOVED);
        update.device = device;
        update.oldSw = oldSw;
        update.oldSwPort = oldPort;
        update.sw = sw;
        update.swPort = port;
        this.updates.add(update);
    }

    protected void updateNetwork(Device device, Set<Integer> networkAddresses,
            Integer networkAddress, boolean added) {
        Update update;
        if (added) {
            update = new Update(UpdateType.NW_ADDED);
        } else {
            update = new Update(UpdateType.NW_REMOVED);
        }
        update.device = device;
        update.networkAddress = networkAddress;
        update.networkAddresses = Collections
                .unmodifiableSet(new HashSet<Integer>(networkAddresses));
        this.updates.add(update);
    }
}
```

- SwitchPortTuple.java

```
package net.beaconcontroller.topology;

public class SwitchPortTuple {
    protected IOFSwitch sw;
    protected Short port;

    public SwitchPortTuple(IOFSwitch sw, Short port) {
        super();
        this.sw = sw;
        this.port = port;
    }

    public SwitchPortTuple(IOFSwitch sw, Integer port) {
        super();
        this.sw = sw;
        this.port = port.shortValue();
    }

    public IOFSwitch getSw() {
        return sw;
    }

    public Short getPort() {
        return port;
    }

    @Override
    public int hashCode() {
        final int prime = 5557;
        int result = 1;
        result = prime * result + ((sw == null) ? 0 : sw.hashCode());
        result = prime * result + ((port == null) ? 0 : port.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (!(obj instanceof SwitchPortTuple))
            return false;
        SwitchPortTuple other = (SwitchPortTuple) obj;
        if (sw == null) {
            if (other.sw != null)
                return false;
        } else if (!sw.equals(other.sw))
```

```
            return false;
        if (port == null) {
            if (other.port != null)
                return false;
        } else if (!port.equals(other.port))
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "SwitchPortTuple [id="
                + ((sw == null) ? "null" : HexString.toHexString(sw.getId()))
                + ", port=" + ((port == null) ? "null" : U16.f(port)) + "]";
    }
}
```

- OFPhysicalPort.java

```
package org.openflow.protocol;

public class OFPhysicalPort {
    public static int MINIMUM_LENGTH = 48;
    public static int OFP_ETH_ALEN = 6;

    public enum OFPortConfig {
        OFPPC_PORT_DOWN    (1 << 0),
        OFPPC_NO_STP       (1 << 1),
        OFPPC_NO_RECV      (1 << 2),
        OFPPC_NO_RECV_STP  (1 << 3),
        OFPPC_NO_FLOOD     (1 << 4),
        OFPPC_NO_FWD       (1 << 5),
        OFPPC_NO_PACKET_IN (1 << 6);

        protected int value;

        private OFPortConfig(int value) {
            this.value = value;
        }

        public int getValue() {
            return value;
        }
    }

    public enum OFPortState {
        OFPPS_LINK_DOWN   (1 << 0),
        OFPPS_STP_LISTEN  (0 << 8),
        OFPPS_STP_LEARN   (1 << 8),
        OFPPS_STP_FORWARD (2 << 8),
        OFPPS_STP_BLOCK   (3 << 8),
        OFPPS_STP_MASK    (3 << 8);

        protected int value;

        private OFPortState(int value) {
            this.value = value;
        }

        public int getValue() {
            return value;
        }
    }

    public enum OFPortFeatures {
        OFPPF_10MB_HD    (1 << 0),
        OFPPF_10MB_FD    (1 << 1),
        OFPPF_100MB_HD   (1 << 2),
        OFPPF_100MB_FD   (1 << 3),
        OFPPF_1GB_HD     (1 << 4),
        OFPPF_1GB_FD     (1 << 5),
        OFPPF_10GB_FD    (1 << 6),
        OFPPF_COPPER     (1 << 7),
        OFPPF_FIBER      (1 << 8),
        OFPPF_AUTONEG    (1 << 9),
```

```
        OFPPF_PAUSE       (1 << 10),
        OFPPF_PAUSE_ASYM  (1 << 11);

        protected int value;

        private OFPortFeatures(int value) {
            this.value = value;
        }


        public int getValue() {
            return value;
        }
    }

    protected short portNumber;
    protected byte[] hardwareAddress;
    protected String name;
    protected int config;
    protected int state;
    protected int currentFeatures;
    protected int advertisedFeatures;
    protected int supportedFeatures;
    protected int peerFeatures;

    public short getPortNumber() {
        return portNumber;
    }

    public OFPhysicalPort setPortNumber(short portNumber) {
        this.portNumber = portNumber;
        return this;
    }

    public byte[] getHardwareAddress() {
        return hardwareAddress;
    }

    public OFPhysicalPort setHardwareAddress(byte[] hardwareAddress) {
        if (hardwareAddress.length != OFP_ETH_ALEN)
            throw new RuntimeException("Hardware address must have length "
                    + OFP_ETH_ALEN);
        this.hardwareAddress = hardwareAddress;
        return this;
    }

    public String getName() {
        return name;
    }

    public OFPhysicalPort setName(String name) {
        this.name = name;
        return this;
    }

    public int getConfig() {
        return config;
    }

    public OFPhysicalPort setConfig(int config) {
        this.config = config;
        return this;
    }

    public int getState() {
        return state;
    }

    public OFPhysicalPort setState(int state) {
        this.state = state;
        return this;
    }

    public int getCurrentFeatures() {
        return currentFeatures;
    }
```

```
public OFPhysicalPort setCurrentFeatures(int currentFeatures) {
    this.currentFeatures = currentFeatures;
    return this;
}

public int getAdvertisedFeatures() {
    return advertisedFeatures;
}

public OFPhysicalPort setAdvertisedFeatures(int advertisedFeatures) {
    this.advertisedFeatures = advertisedFeatures;
    return this;
}

public int getSupportedFeatures() {
    return supportedFeatures;
}

public OFPhysicalPort setSupportedFeatures(int supportedFeatures) {
    this.supportedFeatures = supportedFeatures;
    return this;
}

public int getPeerFeatures() {
    return peerFeatures;
}

public OFPhysicalPort setPeerFeatures(int peerFeatures) {
    this.peerFeatures = peerFeatures;
    return this;
}

public void readFrom(ByteBuffer data) {
    this.portNumber = data.getShort();
    if (this.hardwareAddress == null)
        this.hardwareAddress = new byte[OFP_ETH_ALEN];
    data.get(this.hardwareAddress);
    byte[] name = new byte[16];
    data.get(name);
    int index = 0;
    for (byte b : name) {
        if (0 == b)
            break;
        ++index;
    }
    this.name = new String(Arrays.copyOf(name, index),
            Charset.forName("ascii"));
    this.config = data.getInt();
    this.state = data.getInt();
    this.currentFeatures = data.getInt();
    this.advertisedFeatures = data.getInt();
    this.supportedFeatures = data.getInt();
    this.peerFeatures = data.getInt();
}

public void writeTo(ByteBuffer data) {
    data.putShort(this.portNumber);
    data.put(hardwareAddress);
    try {
        byte[] name = this.name.getBytes("ASCII");
        if (name.length < 16) {
            data.put(name);
            for (int i = name.length; i < 16; ++i) {
                data.put((byte) 0);
            }
        } else {
            data.put(name, 0, 15);
            data.put((byte) 0);
        }
    } catch (UnsupportedEncodingException e) {
        throw new RuntimeException(e);
    }
    data.putInt(this.config);
    data.putInt(this.state);
    data.putInt(this.currentFeatures);
    data.putInt(this.advertisedFeatures);
    data.putInt(this.supportedFeatures);
```

```
        data.putInt(this.peerFeatures);
    }

    @Override
    public int hashCode() {
        final int prime = 307;
        int result = 1;
        result = prime * result + portNumber;
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (!(obj instanceof OFPhysicalPort)) {
            return false;
        }
        OFPhysicalPort other = (OFPhysicalPort) obj;
        if (portNumber != other.portNumber) {
            return false;
        }
        return true;
    }
}
```

- TopologyWebManageable.java

```
@Controller
@RequestMapping("/topology")
public class TopologyWebManageable implements IWebManageable {
    protected static Logger log =
LoggerFactory.getLogger(TopologyWebManageable.class);
    protected List<Tab> tabs;
    protected ITopology topology;

    public TopologyWebManageable() {
        tabs = new ArrayList<Tab>();
        tabs.add(new Tab("Overview", "/wm/topology/overview.do"));
    }

    @Override
    public String getName() {
        return "Topology";
    }

    @Override
    public String getDescription() {
        return "View the discovered topology.";
    }

    @Override
    public List<Tab> getTabs() {
        return tabs;
    }

    public int findBandwidth(IOFSwitch sw,Short port){
        Map<Short,OFPhysicalPort> ports = sw.getFeaturesReply().getPortMap();
        return ports.get(port).getCurrentFeatures();
    }

    @RequestMapping("/overview")
    public String overview(Map<String, Object> model) {
        Layout layout = new TwoColumnLayout();
        model.put("layout", layout);

        // Listener List Table
        List<String> columnNames = new ArrayList<String>();
        List<List<String>> cells = new ArrayList<List<String>>();
        columnNames = new ArrayList<String>();
```

```
        columnNames.add("Src Id");
        columnNames.add("Src Port");
        columnNames.add("Dst Id");
        columnNames.add("Dst Port");
        cells = new ArrayList<List<String>>();
        for (LinkTuple lt : topology.getLinks().keySet()) {
            List<String> row = new ArrayList<String>();
            row.add(HexString.toHexString(lt.getSrc().getSw().getId()));
            row.add(lt.getSrc().getPort().toString());
            row.add(HexString.toHexString(lt.getDst().getSw().getId()));
            row.add(lt.getDst().getPort().toString());
            cells.add(row);
        }

        BufferedWriter writer = null;
        try {
            writer = new BufferedWriter(new FileWriter("topolgy.txt"));
            List<String> columnName = new ArrayList<String>();
            columnName = new ArrayList<String>();
            columnName.add("Src Id");
            columnName.add("Src Port");
            columnName.add("Dst Id");
            columnName.add("Dst Port");
            columnName.add("Capacity");
            columnName.add("Cost");
            int n = columnName.size();
            for(int i = 0; i < n ; i++){
              writer.write(columnName.get( i ) );
              writer.write("                    ");
            }
            writer.newLine();
            for (LinkTuple lt : topology.getLinks().keySet()) {
                List<String> row = new ArrayList<String>();
                row.add(HexString.toHexString(lt.getSrc().getSw().getId()));
                row.add(lt.getSrc().getPort().toString());
                row.add(HexString.toHexString(lt.getDst().getSw().getId()));
                row.add(lt.getDst().getPort().toString());
                row.add(findBandwidth(lt.getSrc().getSw() , lt.getSrc().getPort() ) +
"Mbps");
                row.add("10");
                for(int i = 0; i < n ; i++){
                    writer.write( row.get( i ) );
                    writer.write("          ");
                  }
                writer.newLine();
            }

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                // Close the writer regardless of what happens...
                writer.close();
            } catch (Exception e) {
            }
        }

        Map<String,String> tableOptions = new HashMap<String, String>();
        tableOptions.put("\"bFilter\"", "true");
        TableSection tableSection = new TableSection("Discovered Links", columnNames,
cells, "table-links", tableOptions);
        layout.addSection(tableSection, TwoColumnLayout.COLUMN1);

        return BeaconViewResolver.SIMPLE_VIEW;
    }


    /**
     * @param topology the topology to set
     */
    @Autowired
    public void setTopology(ITopology topology) {
        this.topology = topology;
    }
}
```

- CoreWebManagable.java

```
@Controller
@RequestMapping("/core")
public class CoreWebManageable implements BundleContextAware, IWebManageable {

    protected interface OFSRCallback {
        OFStatisticsRequest getRequest();
    }

    protected static Logger log = LoggerFactory.getLogger(CoreWebManageable.class);
    protected IBeaconProvider beaconProvider;
    protected BundleContext bundleContext;
    protected PackageAdmin packageAdmin;
    protected List<Tab> tabs;

    public CoreWebManageable() {
        tabs = new ArrayList<Tab>();
        tabs.add(new Tab("Overview", "/wm/core/overview.do"));
        tabs.add(new Tab("OSGi", "/wm/core/osgi.do"));
    }

    @Autowired
    @Override
    public void setBundleContext(BundleContext bundleContext) {
        this.bundleContext = bundleContext;
    }

    @Autowired
    public void setBeaconProvider(IBeaconProvider beaconProvider) {
        this.beaconProvider = beaconProvider;
    }

    @Override
    public String getName() {
        return "Core";
    }

    @Override
    public String getDescription() {
        return "Controls the core components of Beacon.";
    }

    @Override
    public List<Tab> getTabs() {
        return tabs;
    }

    @RequestMapping("/overview")
    public String overview(Map<String, Object> model) {
        Layout layout = new TwoColumnLayout();
        model.put("layout", layout);

        layout.addSection(
                new StringSection("Welcome",
                        "Thanks for using Beacon! Beacon is listening for connections
at " +
                (beaconProvider.getListeningIPAddress().isAnyLocalAddress() ? ""
                        : beaconProvider.getListeningIPAddress().getHostAddress()) +
                " port " + beaconProvider.getListeningPort() + "."),
                TwoColumnLayout.COLUMN1);

        model.put("title", "Switches");
        model.put("switches", beaconProvider.getSwitches().values());
        layout.addSection(new JspSection("switches.jsp", model),
TwoColumnLayout.COLUMN2);

        List<String> columnNames = new ArrayList<String>();
        List<List<String>> cells = new ArrayList<List<String>>();
        columnNames = new ArrayList<String>();
        columnNames.add("OpenFlow Packet Type");
        columnNames.add("Listeners");
        cells = new ArrayList<List<String>>();
        for (Entry<OFType, List<IOFMessageListener>> entry :
beaconProvider.getListeners().entrySet()) {
            List<String> row = new ArrayList<String>();
            row.add(entry.getKey().toString());
```

```
            StringBuffer sb = new StringBuffer();
            for (IOFMessageListener listener : entry.getValue()) {
                sb.append(listener.getName() + " ");
            }
            row.add(sb.toString());
            cells.add(row);
        }
        layout.addSection(new TableSection("OpenFlow Packet Listeners", columnNames,
cells, "table-listeners"), TwoColumnLayout.COLUMN1);

        return BeaconViewResolver.SIMPLE_VIEW;
    }

    @RequestMapping("/osgi")
    public String osgi(Map<String, Object> model) {
        Layout layout = new OneColumnLayout();
        model.put("layout", layout);

        model.put("title", "Add Bundle");
        layout.addSection(new JspSection("addBundle.jsp", new HashMap<String,
Object>(model)), TwoColumnLayout.COLUMN1);

        model.put("bundles", Arrays.asList(this.bundleContext.getBundles()));
        model.put("title", "OSGi Bundles");
        layout.addSection(new JspSection("bundles.jsp", model),
TwoColumnLayout.COLUMN1);

        return BeaconViewResolver.SIMPLE_VIEW;
    }

    @RequestMapping("/bundle/{bundleId}/{action}")
    @ResponseBody
    public String osgiAction(@PathVariable Long bundleId, @PathVariable String action)
{
        final Bundle bundle = this.bundleContext.getBundle(bundleId);
        if (action != null) {
            try {
                if (BundleAction.START.toString().equals(action)) {
                    bundle.start();
                } else if (BundleAction.STOP.toString().equals(action)) {
                    bundle.stop();
                } else if (BundleAction.UNINSTALL.toString().equals(action)) {
                    bundle.uninstall();
                } else if (BundleAction.REFRESH.toString().equals(action)) {
                    packageAdmin.refreshPackages(new Bundle[] {bundle});
                }
            } catch (BundleException e) {
                log.error("Failure performing action " + action + " on bundle " +
bundle.getSymbolicName(), e);
            }
        }
        return "";
    }

    @RequestMapping(value = "/bundle/add", method = RequestMethod.POST)
    public View osgiBundleAdd(@RequestParam("file") MultipartFile file, Map<String,
Object> model) throws Exception {
        BeaconJsonView view = new BeaconJsonView();

        File tempFile = null;
        Bundle newBundle = null;
        try {
            tempFile = File.createTempFile("beacon", ".jar");
            file.transferTo(tempFile);
            tempFile.deleteOnExit();
            newBundle =
bundleContext.installBundle("file:"+tempFile.getCanonicalPath());
            model.put(BeaconJsonView.ROOT_OBJECT_KEY,
                    "Successfully installed: " + newBundle.getSymbolicName()
                            + "_" + newBundle.getVersion());
        } catch (IOException e) {
            log.error("Failure to create temporary file", e);
            model.put(BeaconJsonView.ROOT_OBJECT_KEY, "Failed to install bundle.");
        } catch (BundleException e) {
            log.error("Failure installing bundle", e);
            model.put(BeaconJsonView.ROOT_OBJECT_KEY, "Failed to install bundle.");
        }
```

```
        view.setContentType("text/javascript");
        return view;
    }

    @RequestMapping("/refreshWeb")
    @ResponseBody
    public String refreshWebBundle() {
        for (Bundle bundle : this.bundleContext.getBundles()) {
            if (bundle.getSymbolicName().equalsIgnoreCase("net.beaconcontroller.web"))
{
                packageAdmin.refreshPackages(new Bundle[] {bundle});
                try {
                    Thread.sleep(1000);
                    while (bundle.getState() != Bundle.ACTIVE) {
                        Thread.sleep(100);
                    }
                } catch (InterruptedException e) {
                    log.error("Interupted waiting for refresh", e);
                }
            }
        }
        return "";
    }

    protected List<OFStatistics> getSwitchStats(OFSRCallback f, String switchId,
String statsType) {
        IOFSwitch sw = beaconProvider.getSwitches().get(HexString.toLong(switchId));
        Future<List<OFStatistics>> future;
        List<OFStatistics> values = null;
        if (sw != null) {
            try {
                future = sw.getStatistics(f.getRequest());
                values = future.get(10, TimeUnit.SECONDS);
            } catch (Exception e) {
                log.error("Failure retrieving " + statsType, e);
            }
        }
        return values;
    }

    protected class makeFlowStatsRequest implements OFSRCallback {
        public OFStatisticsRequest getRequest() {
            OFStatisticsRequest req = new OFStatisticsRequest();
            OFFlowStatisticsRequest fsr = new OFFlowStatisticsRequest();
            OFMatch match = new OFMatch();
            match.setWildcards(0xffffffff);
            fsr.setMatch(match);
            fsr.setOutPort(OFPort.OFPP_NONE.getValue());
            fsr.setTableId((byte) 0xff);
            req.setStatisticType(OFStatisticsType.FLOW);
            req.setStatistics(fsr);
            req.setLengthU(req.getLengthU() + fsr.getLength());
            return req;
        };
    }

    public String addStatsSection(String statsType, @PathVariable String switchId,
Map<String,Object> model, List<OFStatistics> stats) {
        OneColumnLayout layout = new OneColumnLayout();
        model.put("title", statsType + " for switch: " + switchId);
        model.put("layout", layout);
        model.put(statsType, stats);
        layout.addSection(new JspSection(statsType + ".jsp", model), null);
        return BeaconViewResolver.SIMPLE_VIEW;
    }

    protected List<OFStatistics> getSwitchFlows(String switchId) {
        return getSwitchStats(new makeFlowStatsRequest(), switchId, "flows");
    }

    @RequestMapping("/switch/{switchId}/flows/json")
    public View getSwitchFlowsJson(@PathVariable String switchId, Map<String,Object>
model) {
        BeaconJsonView view = new BeaconJsonView();
        model.put(BeaconJsonView.ROOT_OBJECT_KEY, getSwitchFlows(switchId));
        return view;
    }
```

```
    @SuppressWarnings("unchecked")
    @RequestMapping("/switch/{switchId}/flows/dataTable")
    public View getSwitchFlowsDataTable(@PathVariable String switchId,
Map<String,Object> model) throws IOException {
        List<OFFlowStatisticsReply> data = new ArrayList<OFFlowStatisticsReply>();
        List<OFStatistics> stats = getSwitchFlows(switchId);
        if (stats != null) {
            data.addAll((Collection<? extends OFFlowStatisticsReply>) stats);
        }
        DataTableJsonView<OFFlowStatisticsReply> view = new
DataTableJsonView<OFFlowStatisticsReply>(
                data, new OFFlowStatisticsReplyDataTableFormatCallback(switchId));

        System.out.println("---->switchId="+switchId);
        String sw_id =switchId.replace(":", "_");
        System.out.println("---->switchId="+sw_id);
        FileWriter writer = new FileWriter(sw_id+".txt");
        BufferedWriter bufferWritter = new BufferedWriter(writer);
        for(int i=0; i<data.size(); i++){
        bufferWritter.write(U16.f(data.get(i).getMatch().getInputPort())+" ");

bufferWritter.write(HexString.toHexString(data.get(i).getMatch().getDataLayerSource())
+" ");

bufferWritter.write(IPv4.fromIPv4Address(data.get(i).getMatch().getNetworkSource())+"
");

bufferWritter.write(IPv4.fromIPv4Address(data.get(i).getMatch().getNetworkDestination(
))+ " ");
        StringBuffer outPorts = new StringBuffer();
        for (OFAction action : data.get(i).getActions()) {
            if (action instanceof OFActionOutput) {
                OFActionOutput ao = (OFActionOutput)action;
                if (outPorts.length() > 0)
                    outPorts.append(" ");
                outPorts.append(U16.f(ao.getPort()));
            }
        }

bufferWritter.write(HexString.toHexString(data.get(i).getMatch().getDataLayerDestinati
on())+" ");
        bufferWritter.write(outPorts.toString()+" ");
        bufferWritter.write(U64.f(data.get(i).getByteCount())+" ");
        bufferWritter.write(U64.f(data.get(i).getPacketCount())+" ");
        double duration = (double)U32.f(data.get(i).getDurationSeconds())+((double)
data.get(i).getDurationNanoseconds()) / 1000000000d;
        bufferWritter.write(Double.toString(duration));
        bufferWritter.newLine();
        }
        bufferWritter.close();

        return view;
    }

    @RequestMapping("/switch/{switchId}/flows")
    public String getSwitchFlows(@PathVariable String switchId, Map<String,Object>
model) {
        OneColumnLayout layout = new OneColumnLayout();
        model.put("title", "Flows for switch: " + switchId);
        model.put("layout", layout);
        model.put("switchId", switchId);
        model.put("switchIdEsc", switchId.replaceAll(":", ""));
        layout.addSection(new JspSection("flows.jsp", model), null);
        return BeaconViewResolver.SIMPLE_VIEW;
    }

    @RequestMapping("/switch/{switchId}/flow/{flowId}/del")
    public String delFlow(@PathVariable String switchId,
            @PathVariable String flowId, Map<String, Object> model) {
        OFMatch match = new OFMatch();
        match.fromString(flowId);
        OFFlowMod mod = new OFFlowMod();
        mod.setCommand(OFFlowMod.OFPFC_DELETE)
            .setMatch(match)
            .setOutPort(OFPort.OFPP_NONE);
```

```
        IOFSwitch sw = beaconProvider.getSwitches().get(
                HexString.toLong(switchId));
        try {
            sw.getOutputStream().write(mod);
        } catch (IOException e) {
            log.error("Failure writing delete flowmod", e);
        }

        return BeaconViewResolver.SIMPLE_VIEW;
    }

    protected List<OFStatistics> getSwitchTables(String switchId) {
        return getSwitchStats(new OFSRCallback() {
            @Override
            public OFStatisticsRequest getRequest() {
                OFStatisticsRequest req = new OFStatisticsRequest();
                req.setStatisticType(OFStatisticsType.TABLE);
                req.setLengthU(req.getLengthU());
                return req;
            }
        }, switchId, "tables");
    }


    @RequestMapping("/switch/{switchId}/tables/json")
    public View getSwitchTablesJson(@PathVariable String switchId, Map<String,Object>
model) {
        BeaconJsonView view = new BeaconJsonView();
        model.put(BeaconJsonView.ROOT_OBJECT_KEY, getSwitchTables(switchId));
        return view;
    }

    @RequestMapping("/switch/{switchId}/tables")
    public String getSwitchTables(@PathVariable String switchId, Map<String,Object>
model) {
        List<OFStatistics> ports = getSwitchTables(switchId);
        return addStatsSection("tables", switchId, model, ports);
    }

    protected List<OFStatistics> getSwitchPorts(String switchId) {
        return getSwitchStats(new OFSRCallback() {
            @Override
            public OFStatisticsRequest getRequest() {
                OFStatisticsRequest req = new OFStatisticsRequest();
                OFPortStatisticsRequest psr = new OFPortStatisticsRequest();
                psr.setPortNumber(OFPort.OFPP_NONE.getValue());
                req.setStatisticType(OFStatisticsType.PORT);
                req.setStatistics(psr);
                req.setLengthU(req.getLengthU() + psr.getLength());
                return req;
            }
        }, switchId, "ports");
    }

    @RequestMapping("/switch/{switchId}/ports/json")
    public View getSwitchPortsJson(@PathVariable String switchId, Map<String,Object>
model) {
        BeaconJsonView view = new BeaconJsonView();
        model.put(BeaconJsonView.ROOT_OBJECT_KEY, getSwitchPorts(switchId));
        return view;
    }

    @RequestMapping("/switch/{switchId}/ports")
    public String getSwitchPorts(@PathVariable String switchId, Map<String,Object>
model) {
        List<OFStatistics> ports = getSwitchPorts(switchId);
        return addStatsSection("ports", switchId, model, ports);
    }

    @Autowired
    public void setPackageAdmin(PackageAdmin packageAdmin) {
        this.packageAdmin = packageAdmin;
    }
}
```

- OFPortChangeState.java

```
package net.beaconcontroller.core.io.internal;

public class OFPortChangeState {

    protected IOFSwitch sw;

    public void run()
    {

        try {
            while (true) {

        File f = null;
        File d = null;
        boolean bool = false;
        boolean bool1 = false;
        String[] str ;
        int OFPPC_PORT_UP_CONFIG = 0;
        int OFPPC_PORT_UP_MASK = 1;


        f = new File("c://Users/Pano/Desktop/shut_links.txt");
        d = new File("topolgy.txt");
        bool = f.exists();
        bool1= d.exists();

       if(bool == true && bool1 == true)
       {
           BufferedReader reader1 = null;
           BufferedReader reader2 = null;

           try {
                reader1 = new BufferedReader(new FileReader("shut_links.txt"));
                reader2 = new BufferedReader(new FileReader("topolgy.txt"));

            }
           catch (FileNotFoundException e) {
                e.printStackTrace();
            }

           try {
                reader1.readLine();
                reader2.readLine();
            }
           catch (IOException e) {
                e.printStackTrace();
            }
           String line1 ;
           String line2 ;
           String str1 = null;
           String str2 = null;
           String str3 = null;
           String str4 = null;
           String str5 = null;
           String str6 = null;
           String str7 = null;
           String str8 = null;
           String str9 = null;
           String str10= null;

           try {
               while ((line1 = reader1.readLine()) != null) {
                       str = line1.split(" ");
                       str1  = str[0];
                       str2  = str[1];
                       str3  = str[2];
                       str4  = str[3];
                       str5  = str[4];
                       str6  = str[5];
                       str7  = str[6];
                       str8  = str[7];
                       str9  = str[8];
                       str10 = str[9];
```

```
            }
        }
        catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
                int iter = 0 ;
        try {
            while ((line2 = reader2.readLine()) != null) {
                String[] parts = line2.split("          ");
                String part1 = parts[0];
                String part2 = parts[1];
                String part3 = parts[2];
                String part4 = parts[3];
                short s1 = Short.parseShort(part2);
                short s2 = Short.parseShort(part4);

                switch(iter){

                    case 0:
                        if(str1 == "0"){

                            OFPortMod pm1 = new OFPortMod();
                            pm1.setPortNumber((short)s1);
                            pm1.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x01});
                            pm1.setConfig(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                            pm1.setMask(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                            sw.getOutputStream().write(pm1);

                            OFPortMod pm2 = new OFPortMod();
                            pm2.setPortNumber((short)s2);
                            pm2.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x03});
                            pm2.setConfig(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                            pm2.setMask(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                            sw.getOutputStream().write(pm2);

                        }
                        else if(str1=="1"){


                            OFPortMod pm1 = new OFPortMod();
                            pm1.setPortNumber((short)s1);
                            pm1.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x01});
                            pm1.setConfig(OFPPC_PORT_UP_CONFIG);
                            pm1.setMask(OFPPC_PORT_UP_MASK);
                            sw.getOutputStream().write(pm1);

                            OFPortMod pm2 = new OFPortMod();
                            pm2.setPortNumber((short)s2);
                            pm2.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x03});
                            pm2.setConfig(OFPPC_PORT_UP_CONFIG);
                            pm2.setMask(OFPPC_PORT_UP_MASK);
                            sw.getOutputStream().write(pm2);


                        }
                        break;

                    case 2:
                        if(str3 == "0"){

                            OFPortMod pm1 = new OFPortMod();
                            pm1.setPortNumber((short)s1);
                            pm1.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x01});
                            pm1.setConfig(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                            pm1.setMask(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                            sw.getOutputStream().write(pm1);

                            OFPortMod pm2 = new OFPortMod();
                            pm2.setPortNumber((short)s2);
                            pm2.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x03});
```

```
                                pm2.setConfig(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                pm2.setMask(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                sw.getOutputStream().write(pm2);

                            }
                            else if(str3=="1"){


                                OFPortMod pm1 = new OFPortMod();
                                pm1.setPortNumber((short)s1);
                                pm1.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x01});
                                pm1.setConfig(OFPPC_PORT_UP_CONFIG);
                                pm1.setMask(OFPPC_PORT_UP_MASK);
                                sw.getOutputStream().write(pm1);

                                OFPortMod pm2 = new OFPortMod();
                                pm2.setPortNumber((short)s2);
                                pm2.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x03});
                                pm2.setConfig(OFPPC_PORT_UP_CONFIG);
                                pm2.setMask(OFPPC_PORT_UP_MASK);
                                sw.getOutputStream().write(pm2);


                            }
                            break;
                        case 4:
                            if(str1 == "0"){

                                OFPortMod pm1 = new OFPortMod();
                                pm1.setPortNumber((short)s1);
                                pm1.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x01});
                                pm1.setConfig(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                pm1.setMask(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                sw.getOutputStream().write(pm1);

                                OFPortMod pm2 = new OFPortMod();
                                pm2.setPortNumber((short)s2);
                                pm2.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x03});
                                pm2.setConfig(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                pm2.setMask(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                sw.getOutputStream().write(pm2);

                            }
                            else if(str1=="1"){


                                OFPortMod pm1 = new OFPortMod();
                                pm1.setPortNumber((short)s1);
                                pm1.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x01});
                                pm1.setConfig(OFPPC_PORT_UP_CONFIG);
                                pm1.setMask(OFPPC_PORT_UP_MASK);
                                sw.getOutputStream().write(pm1);

                                OFPortMod pm2 = new OFPortMod();
                                pm2.setPortNumber((short)s2);
                                pm2.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x03});
                                pm2.setConfig(OFPPC_PORT_UP_CONFIG);
                                pm2.setMask(OFPPC_PORT_UP_MASK);
                                sw.getOutputStream().write(pm2);


                            }
                            break;

                        case 6:
                            if(str1 == "0"){

                                OFPortMod pm1 = new OFPortMod();
                                pm1.setPortNumber((short)s1);
                                pm1.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x01});
```

```
                                pm1.setConfig(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                pm1.setMask(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                sw.getOutputStream().write(pm1);

                                OFPortMod pm2 = new OFPortMod();
                                pm2.setPortNumber((short)s2);
                                pm2.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x03});
                                pm2.setConfig(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                pm2.setMask(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                sw.getOutputStream().write(pm2);

                            }
                            else if(str1=="1"){


                                OFPortMod pm1 = new OFPortMod();
                                pm1.setPortNumber((short)s1);
                                pm1.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x01});
                                pm1.setConfig(OFPPC_PORT_UP_CONFIG);
                                pm1.setMask(OFPPC_PORT_UP_MASK);
                                sw.getOutputStream().write(pm1);

                                OFPortMod pm2 = new OFPortMod();
                                pm2.setPortNumber((short)s2);
                                pm2.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x03});
                                pm2.setConfig(OFPPC_PORT_UP_CONFIG);
                                pm2.setMask(OFPPC_PORT_UP_MASK);
                                sw.getOutputStream().write(pm2);


                            }
                            break;
                        case 8:
                            if(str1 == "0"){

                                OFPortMod pm1 = new OFPortMod();
                                pm1.setPortNumber((short)s1);
                                pm1.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x01});
                                pm1.setConfig(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                pm1.setMask(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                sw.getOutputStream().write(pm1);

                                OFPortMod pm2 = new OFPortMod();
                                pm2.setPortNumber((short)s2);
                                pm2.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x03});
                                pm2.setConfig(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                pm2.setMask(OFPortConfig.OFPPC_PORT_DOWN.getValue());
                                sw.getOutputStream().write(pm2);

                            }
                            else if(str1=="1"){


                                OFPortMod pm1 = new OFPortMod();
                                pm1.setPortNumber((short)s1);
                                pm1.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x01});
                                pm1.setConfig(OFPPC_PORT_UP_CONFIG);
                                pm1.setMask(OFPPC_PORT_UP_MASK);
                                sw.getOutputStream().write(pm1);

                                OFPortMod pm2 = new OFPortMod();
                                pm2.setPortNumber((short)s2);
                                pm2.setHardwareAddress(new byte[]
{0x00,0x00,0x00,0x00,0x00,0x03});
                                pm2.setConfig(OFPPC_PORT_UP_CONFIG);
                                pm2.setMask(OFPPC_PORT_UP_MASK);
                                sw.getOutputStream().write(pm2);


                            }
                            break;
```

```
                    default:
                        break;

                }
                iter ++ ;
            }
        }
        catch (NumberFormatException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    try {
        String command = "cmd /c start cmd.exe";
        Process child = Runtime.getRuntime().exec(command);

        OutputStream out = child.getOutputStream();

        out.write("cd ../../ILOG/OPL63/oplide /r/n".getBytes());

        out.flush();
        out.write("oplrun \\ILOG\\OPL63\\diplomatiki\\pc.mod
\\ILOG\\PL63\\iplomatiki\\reen_stoixeia.dat  /r/0n".getBytes());
        out.close();
    }
    catch (IOException e) {
    }
    Thread.sleep(300 * 1000);
         }
     } catch (InterruptedException e) {
         e.printStackTrace();
     }
}

     }
```