ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)

στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

"Διαχείριση Τοπολογίας Προγραμματιζόμενου Δικτύου, για Αρχιτεκτονικές του Διαδικτύου των Πραγμάτων"

Καββαδίας Γεώργιος

MM4160005

ΑΘΗΝΑ, ΙΟΥΝΙΟΣ 2018

Περίληψη

Το IoT (Internet of Things) είναι μια τεχνολογία που ολοένα και εξαπλώνεται στον τομέα των επικοινωνιών, και σύμφωνα με έρευνες, προβλέπεται ότι μέχρι το 2020 θα υπάρχουν πάνω από 50 δισεκατομμύρια συσκευές συνδεδεμένες στο Διαδίκτυο [1], [2], [3].

Στο άμεσο μέλλον -αν και ήδη υπάρχουν τέτοιου είδους εφαρμογές [4] - σχεδόν όλα (συσκευές & αντικείμενα) θα συνδέονται μεταξύ τους. Οι επικοινωνίες που λαμβάνουν χώρα ανάμεσα σε θεωρητικά «χαζές» συσκευές, αποτελούν το αντικείμενο του IoT, το οποίο όμως δεν περιορίζεται μονάχα εκεί. Αλλά επεκτείνεται και στο κομμάτι του πώς συσκευές διαφορετικές μεταξύ τους, οι οποίες θα παράγουν δεδομένα σε διαφορετικές δομές, και θα χρησιμοποιούν πιθανόν διαφορετικά πρωτόκολλα, θα μπορούν να ανταλλάξουν πληροφορίες [5]. Αν θέλαμε να δώσουμε κάποια παραδείγματα τέτοιων συσκευών, η λίστα θα μπορούσε να είναι τεράστια, περιλαμβάνοντας: IP κάμερες, οικιακές συσκευές όπως τα πλυντήρια, τα ψυγεία, οι πόρτες των γκαράζ, τα γνωστά σε όλους RFID, αισθητήρες κάθε είδους, ενεργοποιητές (actuators), smartphones, αεροπλάνα, αυτοκίνητα, ακόμη και πυρηνικούς αντιδραστήρες [6].

Συνειδητοποιώντας το μέγεθος αυτής της εξέλιξης όσον αφορά το κομμάτι της δικτύωσης (ενσύρματης & ασύρματης), γεννώνται, μεταξύ άλλων, ερωτήματα που σχετίζονται με το κομμάτι της διευθυνσιοδότησης, τον αριθμό των διαφορετικών συσκευών που μπορούν να υποστηριχτούν από ένα σημερινό δίκτυο, η δυνατότητα επέκτασης και προσαρμογής ενός δικτύου για να φιλοξενήσει νέες συσκευές, η δυνατότητα εύκολης διαχείρισης ενός τέτοιου δικτύου.

Εδώ έρχεται να δώσει την απάντηση η τεχνολογία των προγραμματιζόμενων δικτύων (SDN, Software-Defined Networking). Πρόκειται για μια τεχνολογία που δεν μετράει πάνω από δεκαετία αυτή τη στιγμή, και η οποία σαν σκοπό έχει να αλλάξει τον τρόπο σχεδίασης και διαχείρισης των δικτύων [7].

Το σημαντικό πλεονέκτημα ενός τέτοιου δικτύου, είναι πως λόγω του ότι είναι προγραμματιζόμενο, μπορεί να υποστηρίξει ένα αρκετά μεγάλο πλήθος συσκευών, και λόγω της δομής του, να ρυθμίζει σε πραγματικό χρόνο τη δρομολόγηση των δεδομένων, λαμβάνοντας υπόψη διάφορες παραμέτρους, και φυσικά να είναι ανθεκτικό και «ανοιχτό» στην προσθήκη νέων συσκευών και «πραγμάτων». Συν τοις

άλλοις, δίνει τη δυνατότητα καθολικής διαχείρισης των δικτυακών συσκευών τις οποίες ελέγχει.

Τα προγραμματιζόμενα δίκτυα (SDN) υπόσχονται νέες δυνατότητες, μετατρέποντας κάθε δίκτυο από ένα απλό σύστημα προώθησης πακέτων, σε ένα έξυπνο μέσο διανομής. Στην πρόσφατη εργασία με τίτλο «Edge-assisted Traffic Engineering and applications in the IoT» [8], επεκτείνεται η λειτουργικότητα του βασικού στοιχείου των SDN, του ελεγκτή (controller), ώστε να εκτελεί υπολογισμούς διαδρομής σε δοσμένες τοπολογίες.

Συγκεκριμένα, οι Νίκος Φωτίου et al. [8], εκμεταλλεύονται τη χρήση tags (ετικετών), οι οποίες περιγράφουν τις ιδιότητες και τις δυνατότητες των κόμβων του δικτύου, ενεργοποιώντας έναν νέο τύπο εφαρμογών ελέγχου δικτύου, και συνδέοντας έτσι τις εφαρμογές των χρηστών και τις ροές κυκλοφορίας με τις αποφάσεις και τη διαχείριση των δικτύων.

Σε αυτή την διπλωματική εργασία, επεκτείνουμε την λύση που προτείνεται στην προαναφερθείσα εργασία, δημιουργώντας μία εφαρμογή ανίχνευσης τοπολογίας δικτύου, σε πραγματικό χρόνο, με τη χρήση του Ελεγκτή POX, συμπεριλαμβανομένων και των link failures. Η πληροφορία που συγκεντρώνεται για το δίκτυο εξάγεται στην μορφή tags (ετικετών) με την οποία τροφοδοτείται το πρόγραμμα των προηγουμένων.

Για την επικύρωση της λύσης αυτής, θα χρησιμοποιηθεί ο εξομοιωτής Mininet. Στα επόμενα κεφάλαια, θα εξηγήσουμε στο βάθος που χρειάζεται, τις έννοιες του IoT, των SDN, του πρωτοκόλλου Openflow που θα χρησιμοποιήσουμε, αλλά και των εργαλείων που θα μας χρειαστούν στο πρακτικό κομμάτι. Τέλος, θα επεξηγήσουμε ακριβώς τον κώδικα υλοποίησης.

Abstract

IoT (Internet of Things) is a technology that is increasingly spreading in communications, and research predicts that by 2020 there will be over 50 billion Internet-connected devices [1], [2], [3].

In the near future - although there are already such applications [4] - almost everything (devices & objects) will be linked to each other. Communications that take place between theoretically "stupid" devices are the subject of IoT, which is not limited to it. But it also extends on the question, “how devices which are different from each other and which are going to produce data in different structures using (most probably) different protocols, will be able to exchange information?” [5]. If we wanted to give some examples of such devices, the list could be enormous, including: IP cameras, home appliances such as washing machines, refrigerators, garage doors, RFIDs, all sensors, actuators, smartphones, airplanes, cars, and even nuclear reactors [6].

Realizing the magnitude of this evolution with regard to the part of networking (wired & wireless), there are some questions that arise, Such as the addressing segment, the number of different devices that can be supported by a current network, the ability to scale out and adapt a network to host new devices, the ability to easily manage such a network.

This is where SDN gives the answer. It is a technology that does not count for more than a decade right now, and which is designed to change the way networks are designed and managed [7].

The important advantage of such a network is that because it is programmable, it can support a fairly large number of devices and because of its structure, it can adjust in real-time the routing of the data, taking into account various parameters, and of course it is durable and "open" in adding new devices and "things". In addition, it enables and allows the global management of the network devices it controls.

SDNs promise new capabilities, transforming each network from a simple packet forwarding system into an intelligent distribution medium. In the recent work entitled "Edge-Assisted Traffic Engineering and Applications in the IoT" [8], the functionality of the basic element of SDN, the controller, is extended in order to perform route calculations on given topologies.

In particular, Nikos Fotiou et al. [8], exploit the use of tags that describe the properties and capabilities of network nodes by enabling a new type of network control applications and thus connecting user applications and traffic flows with network decisions and management.

In this thesis, we extend the solution proposed in the above-mentioned paper, creating a real-time network topology discovery application using POX controller. In more details, we are taking advantage of LLDP (Link Layer Discovery Protocol), and two specific components of OpenFlow: openflow.discovery and host_tracker.

LLDP is a standard which was designated in May 2005 as IEEE 802.1AB-2005. It is a non-vendor specific L2 protocol, which can be used by a station attached to a LAN in order to advertise its capabilities and identity, and to receive relevant information from its neighboring stations.

Openflow.discovery component sends LLDP messages to all OpenFlow switches so that it detects the network topology. When a link is created or "dropped", an event is generated on the network. This event includes information about the two points that this link connects (datapath ID, port etc.).

Host_tracker component tries to control hosts on the network. So if something changes, it creates an event. In short, it learns the MACs of the hosts and sends regular ARP messages at regular intervals to see if the host is still alive. Those events include information about the link that connects the host to a switch.

Putting the above components together, gave us the ability to track changes, failures and new entries in our topology. The information gathered from our controller about the network is extracted in the form of tags. In a future work, this solution can be merged and evolve together with the solution of "Edge-Assisted Traffic Engineering and Applications in the IoT" [8], so they can work together.

To validate this solution, the Mininet emulator will be used. In the following chapters, we will explain in the needed depth the concepts of IoT, SDN, OpenFlow protocol that we will use, and the tools we will need in the practical part. Finally, we'll explain the implementation code.

Ευχαριστίες

……………….

Πίνακας περιεχομένων

Κατάλογος Εικόνων

Κατάλογος Αποσπασμάτων Κώδικα

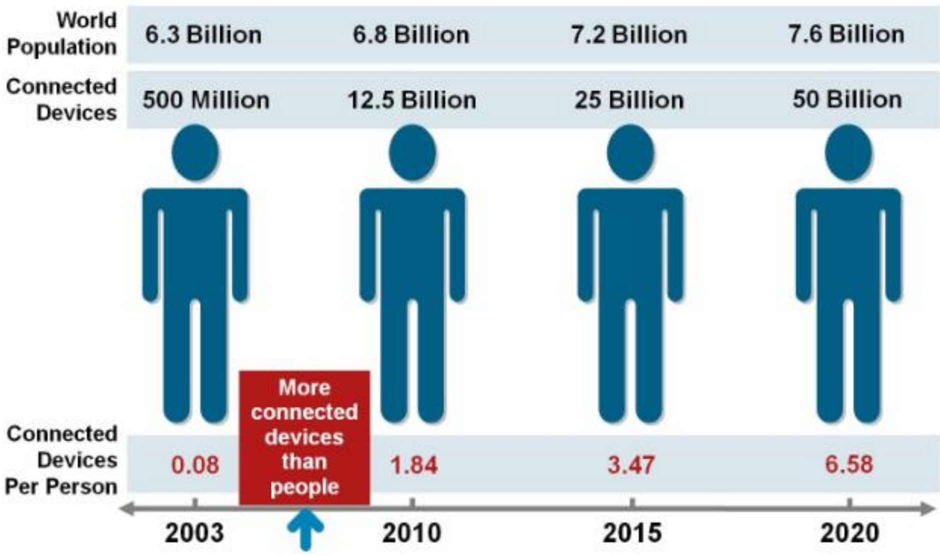
Κατάλογος Πινάκων

1. Το διαδίκτυο των πραγμάτων (IoT)

Το Internet of Things είναι μία έννοια που αφορά τα αντικείμενα της καθημερινότητας μας, από βιομηχανικές μηχανές μέχρι wearable συσκευές που χρησιμοποιούν ενσωματωμένους αισθητήρες για τη συλλογή δεδομένων και την ανάληψη κάποιας δράσης σε αυτά μέσα σε ένα δίκτυο.

Η εξέλιξη και περαιτέρω υλοποίηση του IoT, όπως φαίνεται, θα αλλάξει τα πάντα – ακόμη και τους εαυτούς μας. Παρ' όλο που ακούγεται υπερβολικό, αρκεί κάποιος να συλλογιστεί την επιρροή που έχει ήδη το διαδίκτυο στην εκπαίδευση, την επικοινωνία, την επιχειρηματική δραστηριότητα, την επιστήμη, την διακυβέρνηση και την ανθρωπότητα. Τώρα σκεφτείτε πως το IoT είναι η επόμενη γενιά εξέλιξης του διαδικτύου, δίνοντας τη δυνατότητα σε αυτό να συλλέγει δεδομένα, να τα αναλύει και να τα διαμοιράζει, με σκοπό αυτά να μετατραπούν σε πληροφορίες, γνώση και ίσως σοφία. Ωστόσο αυτό ελλοχεύει και κινδύνους, όπως όλες οι τεχνολογίες άλλωστε, αν δε χρησιμοποιηθεί σωστά, μέσα σε κάποιο νομικό και ηθικό πλαίσιο προστασίας των υποκειμένων [1].

Η ιδέα πίσω από το Internet of Things, είναι η σύνδεση όλων των ηλεκτρονικών συσκευών μεταξύ τους ή/και με το Internet [9]. Η Cisco σε μία μελέτη της [1] προβλέπει ότι μέχρι το 2020 θα υπάρχουν πάνω από 50 δισεκατομμύρια συσκευές συνδεδεμένες στο Διαδίκτυο (Εικόνα 1).



Εικόνα 1: Το IoT «γεννήθηκε» ανάμεσα στο 2008 και το 2009 και εξελίσσεται [1]

Ο όρος που θα μπορούσε να χρησιμοποιηθεί καλύτερα για το IoT είναι «σύνδεση των στοιχείων που μας ενδιαφέρουν στο διαδίκτυο». Με λίγα λόγια το IoT εφαρμόζεται σε ένα εξάρτημα που αποτελείται από αισθητήρες και μία σύνδεση δικτύου. Με αυτόν τον τρόπο μπορεί κάποιος να ελέγξει, να παρακολουθήσει, να διαχειριστεί ή και να επιβλέψει από την εργασία, το σπίτι, ή το αυτοκίνητο, αντικείμενα που τον ενδιαφέρουν.

Η αρχική ιδέα του IoT εμφανίστηκε στις αρχές της δεκαετίας του '80 όπου και δημιουργήθηκε ο πρώτος αυτόματος πωλητής αναψυκτικών που είχε τη δυνατότητα να παρακολουθεί και να αναφέρει στον προμηθευτή την ποσότητα των αναψυκτικών που βρίσκονταν εκείνη τη στιγμή στο ψυγείο αλλά και τη θερμοκρασία των προϊόντων που βρίσκονταν σε αυτό. Η ονομασία και ο ορισμός του IoT δόθηκε το 1985 από τον Peter T. Lewis σύμφωνα με τον οποίο «IoT είναι η ενσωμάτωση ανθρώπων, διαδικασιών, συσκευών και της τεχνολογίας, σε ένα κοινό δίκτυο, για την απομακρυσμένη παρακολούθηση, χειρισμό και αξιολόγηση των τάσεων των συσκευών» [10].

Ωστόσο, ο όρος έγινε διάσημος στα τέλη της δεκαετίας του 1990 από τον επιχειρηματία Kevin Ashton, ο οποίος είναι ένας από τους ιδρυτές του Auto-ID Center στο MIT. Ο Ashton ήταν μέρος μιας ομάδας που ανακάλυψε τον τρόπο να συνδέσει τα αντικείμενα με το διαδίκτυο μέσω RFID, Near Field Communication, Barcodes, QR codes κλπ.. Έχει δηλώσει ότι χρησιμοποίησε πρώτη φορά τη φράση «Internet of Things» σε μια παρουσίαση που έκανε το 1999 και ο όρος αυτός καθιερώθηκε από τότε [11].

Το Internet όπως το γνωρίζουμε αυτή τη στιγμή αποτελεί τη ραχοκοκαλιά του Internet of Things, ωστόσο δεν είναι απαραίτητο οι συσκευές να έχουν απευθείας πρόσβαση σε αυτό. Για παράδειγμα, ένα fitness band συλλέγει αμέτρητα δεδομένα για τη φυσική σου κατάσταση και την υγεία σου, τα μεταδίδει στο smartphone ή το tablet σου μέσω Bluetooth και στη συνέχεια αυτά περνάνε online, στην cloud υπηρεσία που χρησιμοποιείς για την καταγραφή τους. Πρακτικά, δηλαδή, μιλάμε για ένα περιβάλλον συλλογής δεδομένων από οποιαδήποτε ηλεκτρονική συσκευή ή μικροσκοπικό αισθητήρα υπάρχει γύρω μας.

Κάπως έτσι λειτουργεί ένα κτίριο που χρησιμοποιεί αισθητήρες (sensors) για την αυτόματη ρύθμιση της θέρμανσης ή του φωτισμού. Άλλο παράδειγμα είναι ένας εξοπλισμός παραγωγής που προειδοποιεί το προσωπικό συντήρησης για μία επικείμενη βλάβη.

1.1 Οφέλη του IoT

Είναι πολλά τα οφέλη που μπορούμε να αποκομίσουμε από την ανάλυση των data streams μεταξύ διαφόρων συσκευών. Εδώ είναι μερικά παραδείγματα των επιπτώσεων του Internet of Things σε διάφορους κλάδους:

- Έξυπνες λύσεις μεταφοράς επιταχύνουν την ροή της κυκλοφορίας, μειώνουν την κατανάλωση καυσίμων.
- Έξυπνα ηλεκτρικά δίκτυα (smart electric grids) συνδέουν πιο αποτελεσματικά ανανεώσιμες πηγές ενέργειας, βελτιώνουν την αξιοπιστία του συστήματος και χρεώνουν τους καταναλωτές με βάση μικρότερες προσαυξήσεις.
- Μηχανές αισθητήρων παρακολούθησης κάνουν διαγνώσεις και προβλέπουν θέματα συντήρησης που εκκρεμούν, βραχυπρόθεσμα stock-out αποθεμάτων, και θέτουν ακόμα και προτεραιότητες στα προγράμματα του προσωπικού που είναι υπεύθυνο για τις επισκευές για να καλύψουν αποτελεσματικότερα τις ανάγκες επισκευής εξοπλισμού.
- Data-driven συστήματα, χτισμένα στις υποδομές των «έξυπνων πόλεων» καθιστούν ευκολότερο για τους δήμους να «τρέχουν» τις διαδικασίες διαχείρισης αποθεμάτων, την επιβολή του νόμου και άλλα προγράμματα πιο αποτελεσματικά.

Υπάρχουν όμως και πολλά οφέλη από τη χρήση του IoT και σε προσωπικό επίπεδο. Συνδεδεμένες συσκευές χαράζουν τη δική τους πορεία τόσο στον κόσμο των επιχειρήσεων όσο και στη μαζική αγορά:

- Τελειώνει το γάλα στο ψυγείο. Αυτόματη λήψη υπενθύμισης στο κινητό από το ψυγείο για αγορά ενώ ο χρήστης είναι εκτός σπιτιού.
- Το σύστημα ασφαλείας του σπιτιού, επιτρέπει τον απομακρυσμένο έλεγχο από απόσταση των κλειδαριών και του θερμοστάτη, ή ρυθμίζει το κλιματιστικό ώστε να δροσίσει το σπίτι ή να ανοίξει τα παράθυρα.

1.2 Κίνδυνοι και προκλήσεις του IoT

Όπως αναφέρθηκε και παραπάνω, η ασφάλεια αποτελεί μια από τις μεγαλύτερες προκλήσεις που θα κληθούν να αντιμετωπίσουν οι εταιρείες. Κι αυτό γιατί θα πρέπει να προστατευτεί, τόσο το τεράστιο δίκτυο συνδεδεμένων «πραγμάτων» που, όπως όλα δείχνουν, θα δημιουργηθεί μέσα στα επόμενα χρόνια, όσο και του όγκου δεδομένων που θα συγκεντρώνεται από αυτά. Όταν, για παράδειγμα, αισθητήρες συλλέγουν δεδομένα για την κατάσταση της υγείας ενός ανθρώπου, πρέπει να διασφαλιστεί ότι αυτά τα δεδομένα θα παραμένουν ασφαλή και δεν πρόκειται ποτέ να πέσουν στα χέρια των λάθος ανθρώπων. Επιπλέον, με δισεκατομμύρια συνδεδεμένες συσκευές, θα μπορούσε κάποιος να εισβάλει σε ένα δίκτυο μέσω ενός έξυπνου πλυντηρίου που είναι συνδεδεμένο σε αυτό, και το οποίο μπορεί να έχει κάποιο κενό ασφαλείας.

Μια άλλη μεγάλη πρόκληση για τις εταιρείες, είναι η εύρεση αξιόπιστων και ενεργειακά αποδοτικών τρόπων αποθήκευσης και ανάλυσης των δεδομένων που θα παράγουν ταυτόχρονα δισεκατομμύρια συσκευές.

Παράλληλα όμως με την ανάπτυξη του IoT αυξάνεται και η ανάγκη για δυνατότητα διαχείρισης σε πραγματικό χρόνο αυξημένων απαιτήσεων κίνησης δεδομένων. Αυτό γίνεται εύκολα αντιληπτό καθώς θα πρέπει να παρέχεται επαρκές εύρος ζώνης για να καλύπτει από έναν αισθητήρα τοποθετημένο σε μία πόρτα, μέχρι υψηλής ευκρίνειας βίντεο που θα προέρχεται από μία κάμερα ασφαλείας. Ανάλογες θα είναι φυσικά και οι απαιτήσεις σε επίπεδο κρυπτογράφησης και ασφάλειας των δεδομένων.

Συνολικά, τα επόμενα χρόνια αναμένεται μία έξαρση του αριθμού των συνδεδεμένων συσκευών, των τοποθεσιών που αυτές βρίσκονται και φυσικά των λειτουργιών που αυτές θα επιτελούν. Ενδεικτικά μπορούμε να αναφέρουμε τα μελλοντικά νοσοκομεία: πέρα από τις standalone συνδεδεμένες συσκευές θα υπάρχει πληθώρα συσκευών οι οποίες θα βρίσκονται συνδεδεμένες με τους σταθμούς παρακολούθησης ασθενών του νοσηλευτικού προσωπικού.

1.2.1 Προκλήσεις για τις Διευθύνσεις Πληροφορικής

Παρακάτω παρουσιάζονται ενδεικτικά κάποιες από τις προκλήσεις που αντιμετωπίζουν ήδη ή θα αντιμετωπίσουν οι Διευθύνσεις Πληροφορικής, οι οποίες συνήθως καλούνται να επιτύχουν μια ασφαλή υλοποίηση της διασύνδεσης όλων των συσκευών:

- Η αποτελεσματική αυθεντικοποίηση και η διαχείριση δικαιωμάτων πρόσβασης των χρηστών.

- Απαιτήσεις κανονιστικής συμμόρφωσης (π.χ. ΓΚΠΔ).
- Το άγνωστο πολλές φορές κόστος διαχείρισης και αποθήκευσης του τεράστιου όγκου δεδομένων που συλλέγεται, αλλά και της συντήρησης της αναγκαίας δικτυακής υποδομής.
- Έλλειψη εξειδικευμένων γνώσεων και δεξιοτήτων.
- Η ιδιοκτησία των δεδομένων που συλλέγονται και που πολλές φορές ανήκουν σε κάποια άλλη Διεύθυνση εκτός Πληροφορικής (π.χ. Marketing, Ανθρωπίνων Πόρων, Εμπορική κτλ.).

1.3 Διασύνδεση των IoT συσκευών

Το IoT, όπως αναφέρθηκε, έχει ως βάση την σύνδεση διαφόρων μικρών ή μεγάλων συσκευών με ενσωματωμένους αισθητήρες και εξοπλισμό διασύνδεσης, τόσο μεταξύ τους όσο και με τον κατασκευαστή, για να λαμβάνουν και να μεταδίδουν σχετικά δεδομένα με στόχο να προσφέρουν περισσότερες και καλύτερες υπηρεσίες.

Με λίγα λόγια το IoT απεικονίζεται ως μια σειρά από νέα ανεξάρτητα ενσωματωμένα συστήματα, που συνεργάζονται με άλλα συστήματα συγκέντρωσης πληροφοριών σε μεγάλες βάσεις δεδομένων, και τα οποία όλα μαζί λειτουργούν με δικές τους υποδομές και χρησιμοποιούν το διαδίκτυο για τη διασύνδεση τους.

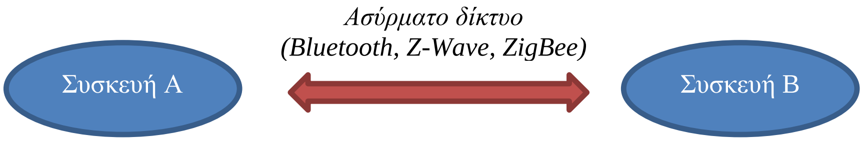
Τα τρία κύρια μέρη ενός IoT είναι:

1. τα «πράγματα», όπου συλλέγουν πληροφορίες οπουδήποτε και οποιαδήποτε στιγμή χρησιμοποιώντας RFID τεχνολογία, αισθητήρες και κώδικα
2. τα δίκτυα επικοινωνιών που συνδέουν τα «πράγματα»
3. τα υπολογιστικά συστήματα και οι εφαρμογές που επεξεργάζονται όσα δεδομένα ρέουν από και προς τα «πράγματα» όπως ένα cloud computing σύστημα.

Η συνδεσιμότητα των τριών αυτών μερών του IoT πραγματοποιείται με τέσσερις τρόπους δικτύωσης (σύνδεσης και επικοινωνίας) όπως περιγράφονται σε έγγραφο του Internet Architecture Board - IAB (RFC 7452) [14], και παρουσιάζονται παρακάτω.

1.3.1 Σύνδεση συσκευή-προς-συσκευή (device-to-device communication)

Το μοντέλο αυτό επικοινωνίας της συσκευής προς συσκευή αναπαρίσταται από δύο ή περισσότερες συσκευές που συνδέονται άμεσα και επικοινωνούν μεταξύ τους χωρίς ενδιάμεσο server. Αυτές οι συσκευές συνδέονται με πολλούς τύπους δικτύων, συμπεριλαμβανομένων των δικτύων IP ή το Internet, χρησιμοποιώντας πρωτόκολλα όπως το Bluetooth, Z-Wave, ή ZigBee.



Εικόνα 2: Σύνδεση συσκευή-προς-συσκευή

Το **Bluetooth** είναι για μια ασύρματη τηλεπικοινωνιακή τεχνολογία μικρών αποστάσεων, ένα πρότυπο για ασύρματα προσωπικά δίκτυα υπολογιστών (Wireless Personal Area Networks, WPAN).

Το **Z-Wave** είναι ένα πρωτόκολλο ασύρματων επικοινωνιών για εφαρμογές οικιακού αυτοματισμού. Χρησιμοποιεί χαμηλής ισχύος ραδιοκύματα.

Ένα πιο εξελιγμένο μέσο δικτύωσης από το Bluetooth αποτελεί το **ZigBee**. Πρόκειται για ένα τυποποιημένο πρωτόκολλο χαμηλής κατανάλωσης ισχύος σε Wireless Personal Area Networks (WPANs).

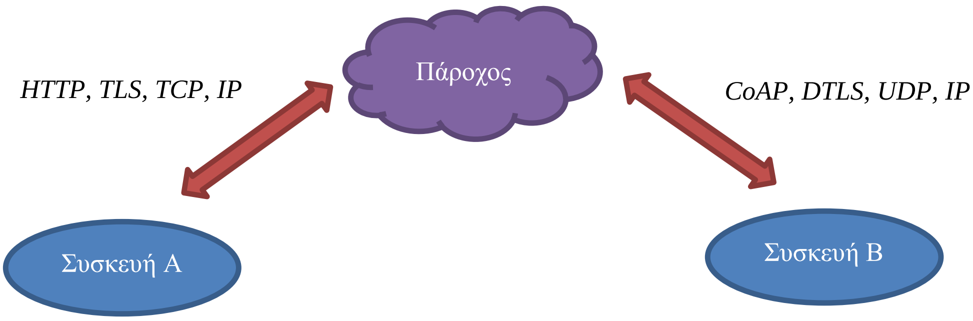
Η device-to-device communication χρησιμοποιείται σε εφαρμογές όπως συστήματα οικιακού αυτοματισμού, που συνήθως χρησιμοποιούν μικρά πακέτα δεδομένων για επικοινωνία μεταξύ των συσκευών και με απαίτηση σχετικά χαμηλού ρυθμού μετάδοσης δεδομένων.

1.3.2 Σύνδεση συσκευής-προς-cloud (device-to-cloud communication)

Η διασύνδεση στο IoT γίνεται εφαρμόζοντας τεχνολογίες, όπως το RFID και ασύρματους αισθητήρες, οι οποίες συλλέγουν τα δεδομένα που στη συνέχεια αξιοποιούνται από τα υπολογιστικά συστήματα. Αυτό έχει ως αποτέλεσμα την δημιουργία τεράστιων ποσοτήτων δεδομένων που θα πρέπει να αποθηκευτούν, να επεξεργαστούν και να παρουσιαστούν. Το cloud computing προσφέρει την υποδομή για τη συλλογή, ανάλυση, αποθήκευση και αποστολή πληροφοριών στον πελάτη.

Έτσι επιτυγχάνεται η παροχή υπηρεσιών προς τους χρήστες που επιθυμούν την πρόσβαση σε εφαρμογές σε οποιοδήποτε χρόνο και μέρος.

Όπως φαίνεται στο σχεδιάγραμμα σε αυτό το μοντέλο, η συσκευή IoT συνδέεται άμεσα με μια υπηρεσία cloud, όπως για παράδειγμα ένα πάροχο υπηρεσίας, για την ανταλλαγή δεδομένων και τον έλεγχο ροής των πληροφοριών.



Εικόνα 3: Σύνδεση συσκευή-προς-cloud

Η επικοινωνία συσκευής-παρόχου γίνεται με δυο τρόπους:

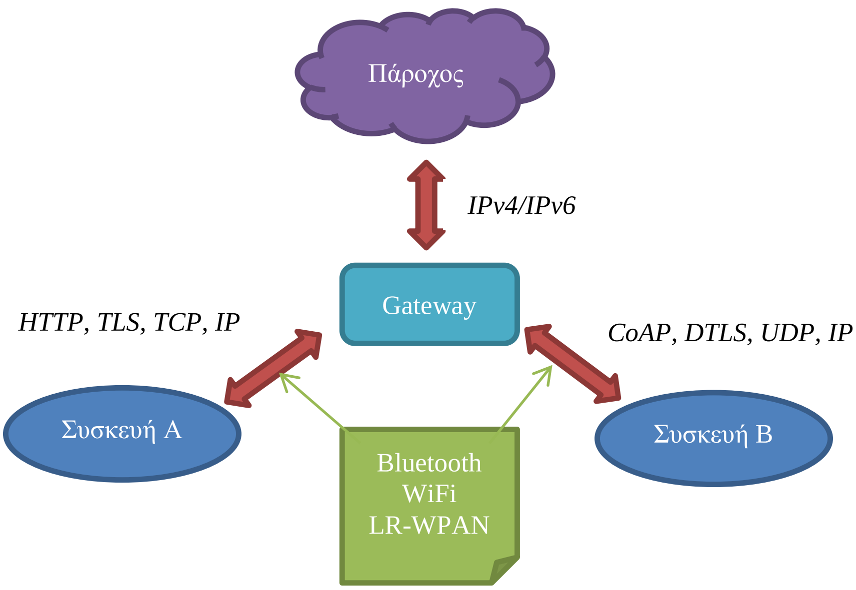
- <u>1ος τρόπος:</u>
  - **Πρωτόκολλο Μεταφοράς Υπερκειμένου (HyperText Transfer Protocol, HTTP):** παρέχεται η δυνατότητα στο πρόγραμμα-πελάτη να στέλνει δεδομένα στον εξυπηρετητή.
  - **TLS (Transport Layer Security):** είναι ένα πρωτόκολλο που εγγυάται ότι κατά την επικοινωνία εξυπηρετητή-πελάτη μέσω του διαδικτύου δεν πρόκειται να μεσολαβήσει κάποιος τρίτος που θα υποκλέψει το περιεχόμενο της επικοινωνίας.
  - **TCP (Transmission Control Protocol-Πρωτόκολλο Ελέγχου Μεταφοράς)**: επιβεβαιώνει την αξιόπιστη αποστολή και λήψη δεδομένων.
  - **Πρωτόκολλο διαδικτύου-IP:** είναι υπεύθυνο για τη δρομολόγηση των πακέτων δεδομένων.
- <u>2ος τρόπος:</u>
  - **Πρωτόκολλο Συσκευών Περιορισμένων Δυνατοτήτων (CoAP)**: Οι συσκευές στις οποίες έχει εισαχθεί το πρωτόκολλο αυτό μπορούν να λειτουργήσουν ταυτόχρονα ως εξυπηρετητές αλλά και ως πελάτες μέσα στο δίκτυο αισθητήρων. Η επικοινωνία ανάμεσα σε 2

οποιαδήποτε endpoints βασίζεται στην χρήση ενός απλού μοντέλου αιτήσεων/απαντήσεων βασισμένο στην λειτουργία του UDP. Σε αντίθεση με το HTTP το οποίο χρησιμοποιεί το TCP, για τις συσκευές αυτού του τύπου είναι προτιμότερη χρήση του UDP καθώς οι απαιτήσεις για την διατήρηση των συνδέσεων υπερβαίνουν συχνά τις δυνατότητες των συσκευών.

- **DTLS (Datagram Transport Layer Security)**: παρέχει ισοδύναμη προστασία με το TLS στα πρωτόκολλα του στρώματος εφαρμογής που χρησιμοποιούν το UDP ως πρωτόκολλο μεταφοράς.
- **User Datagram Protocol-UDP:** είναι υπεύθυνο για τη μεταφορά δεδομένων.
- **Πρωτόκολλο διαδικτύου – IP**: είναι υπεύθυνο για τη δρομολόγηση των πακέτων δεδομένων.

1.3.3 Σύνδεση συσκευής-προς-gateway (Device-to-Gateway Model)

Σε αυτή την σύνδεση η συσκευή και ο πάροχος έρχονται σε επικοινωνία μέσω ενός διαύλου (gateway). Μια gateway συσκευή μπορεί να απορρίπτει, να αθροίζει και να ελέγχει τη μορφή των δεδομένων από μια ομάδα απλών αισθητήρων πριν τα στείλει κάπου αλλού.

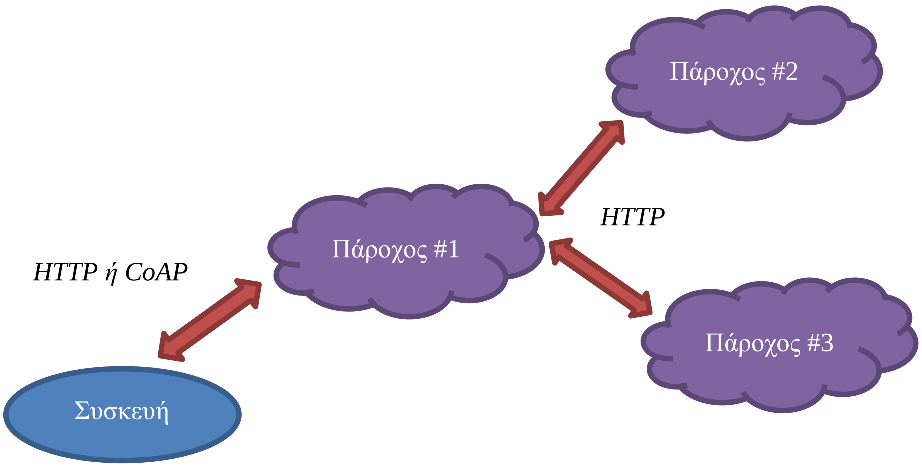


Εικόνα 4: Σύνδεση συσκευής-προς-gateway

Για τη σύνδεση «πράγματος»/συσκευής με το gateway ακολουθούνται οι τρόποι που περιγράφηκαν παραπάνω. Η συνδεσιμότητα gateway-cloud πραγματοποιείται με τα πρωτόκολλα IPv4/IPv6. Προτιμάται το IPv6 γιατί έχει καλύτερη δυνατότητα αυτορρύθμισης συσκευών, καλύτερη ποιότητα υπηρεσιών (QoS) και μεγαλύτερη ασφάλεια από το IPv4, καθώς και μεγαλύτερο πλήθος ταυτόχρονα υποστηριζόμενων συσκευών.

1.3.4 Back-End μοντέλο ανταλλαγής δεδομένων (Back-End Data-Sharing Model)

Το back-end μοντέλο ανταλλαγής δεδομένων αναφέρεται σε μια αρχιτεκτονική επικοινωνίας που επιτρέπει στους χρήστες να εξάγουν και να αναλύουν τα δεδομένα των «πραγμάτων» από μια υπηρεσία cloud, σε συνδυασμό με δεδομένα από άλλες πηγές.



Εικόνα 5: Back-End μοντέλο ανταλλαγής δεδομένων

Για παράδειγμα, ένας εταιρικός χρήστης, υπεύθυνος για ένα συγκρότημα γραφείων, θα πρέπει να συλλέγει και να αναλύει τα δεδομένα κατανάλωσης ενέργειας και κοινής ωφέλειας που παράγονται από όλους τους αισθητήρες IoT και τα Internet-enabled συστήματα κοινής ωφελείας στις εγκαταστάσεις. Η back-end ανταλλαγή δεδομένων επιτρέπει στην εταιρεία να έχει εύκολη πρόσβαση και ανάλυση όλων των δεδομένων στο cloud που παράγονται από όλες τις συσκευές στο κτίριο.

1.4 Πεδία Εφαρμογής

Ο συνδυασμός της τεχνολογίας IoT, με το Cloud Computing (υπηρεσίες πληροφορικής στο σύννεφο), τα Big Data (συλλογή και ανάλυση τεράστιου όγκου δεδομένων) και τις wearable συσκευές (ηλεκτρονικές συσκευές που φοριούνται από τους χρήστες) δημιουργούν ένα πολύ έξυπνο περιβάλλον υπερσυνδεσιμότητας που φέρνει νέες υπηρεσίες και δυνατότητες συνδυασμού με τεχνολογίες.

Συνεπώς το IoT απευθύνεται και μπορεί να εφαρμοστεί σε ένα πολύ ευρύ φάσμα επιχειρηματικών δραστηριοτήτων και όχι μόνο. Παρακάτω μπορείτε να δείτε μερικούς από αυτούς:

- Βιομηχανική παραγωγή [15]
- Γεωργία [16]
- Αυτοκινητοβιομηχανία [16]
- Ο τομέας της έξυπνης ενέργειας (SmartGrids) [17]
- Ιατρική & Υγεία [16]
- Οικιακοί αυτοματισμοί/έξυπνο σπίτι [16]

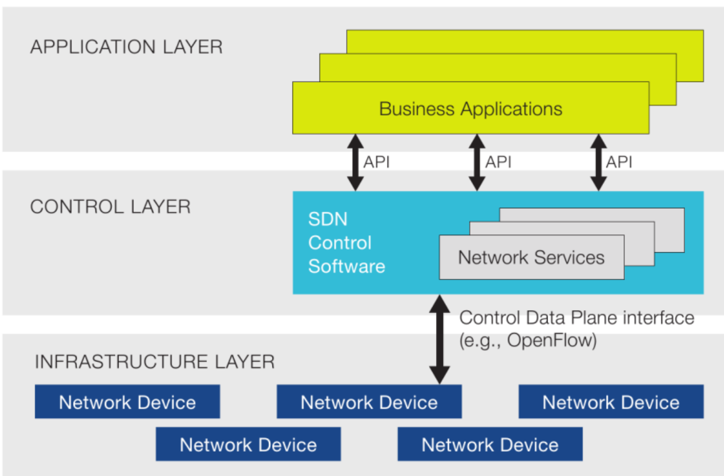
2. Προγραμματιζόμενα Δίκτυα (SDN)

Πρόκειται για μια τεχνολογία που δεν μετράει πάνω από δεκαετία αυτή τη στιγμή, και η οποία σαν σκοπό έχει να αλλάξει τον τρόπο σχεδίασης και διαχείρισης των δικτύων. Το SDN επιτρέπει στις εφαρμογές να έχουν γνώση του δικτύου προσεγγίζοντας με έναν καινούριο τρόπο την αρχιτεκτονική των δικτύων. Σε ένα παραδοσιακό δίκτυο, ο δικτυακός εξοπλισμός, όπως ένας μεταγωγέας ή ένας δρομολογητής, περιέχουν το επίπεδο ελέγχου και το επίπεδο δεδομένων. Το επίπεδο ελέγχου ορίζει την διαδρομή την οποία θα ακολουθήσουν τα πακέτα μέσα στο δίκτυο, ενώ το επίπεδο δεδομένων είναι το κομμάτι του δικτύου που φέρει τα πακέτα [18].

Η νοημοσύνη του δικτύου συγκεντρώνεται σε ελεγκτές οι οποίοι επί της ουσίας είναι λογισμικό που διατηρεί μια γενική άποψη του δικτύου. Με αυτόν τον τρόπο το δίκτυο εμφανίζεται στις εφαρμογές ως ένας λογικός δρομολογητής. Με το SDN, οι επιχειρήσεις και οι πάροχοι αποκτούν έλεγχο σε ολόκληρο το δίκτυο από ένα λογικό σημείο, το οποίο απλουστεύει σημαντικά τον σχεδιασμό του δικτύου και την λειτουργία του. Επίσης, το SDN απλουστεύει σημαντικά τις συσκευές δικτύου, καθώς δεν χρειάζεται πλέον να κατανοήσουν και να επεξεργαστούν χιλιάδες πρότυπα πρωτοκόλλου, αλλά να δεχτούν εντολές από τους ελεγκτές [19].

Η τεχνολογία του SDN διαχωρίζει το δίκτυο σε 3 επίπεδα:

- εφαρμογών,
- ελέγχου και
- δεδομένων ή υποδομών



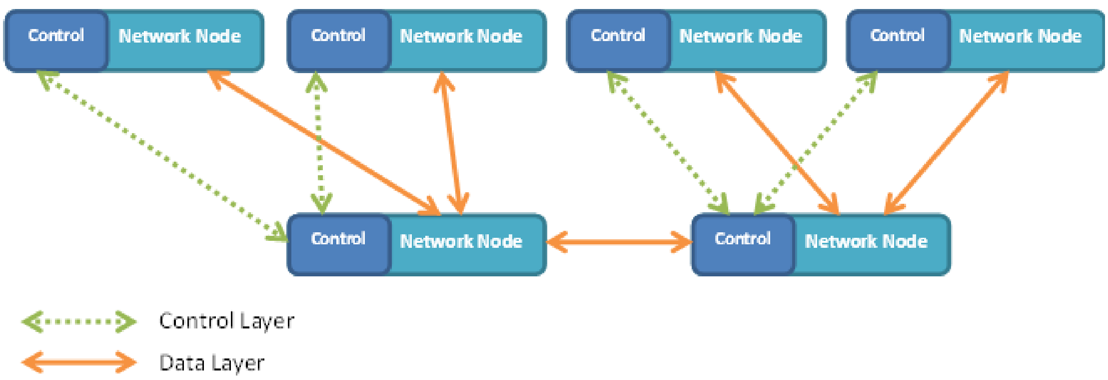
Εικόνα 6: Επίπεδα SDN [20]

Πιο συγκεκριμένα:

- **Επίπεδο εφαρμογών:** περιέχει εφαρμογές SDN οι οποίες επικοινωνούν μέσω διεπαφής προγραμματισμού εφαρμογών (Application Programming Interface, API). Οι εφαρμογές αυτές επικοινωνούν απευθείας τις ανάγκες δικτύου τους και την επιθυμητή δικτυακή συμπεριφορά με τον Controller του δικτύου.
- **Επίπεδο ελέγχου, ή ελεγκτής SDN:** είναι ουσιαστικά ένα λειτουργικό σύστημα δικτύου, το οποίο αποφασίζει για το routing της κυκλοφορίας και παρέχει έναν λογικό χάρτη του δικτύου στις εφαρμογές SDN που υλοποιούνται πάνω σε αυτό. Οι αποφάσεις λαμβάνονται με βάση μια γενικότερη άποψη ολόκληρου του δικτύου, και όχι με την περιορισμένη ορατότητα που λειτουργούν οι δρομολογητές σήμερα.
- **Επίπεδο δεδομένων:** προωθεί την κίνηση σύμφωνα με τις εντολές που λαμβάνει από το επίπεδο ελέγχου. Ως επίπεδο δεδομένων μπορούμε να έχουμε στο μυαλό μας διαφόρων ειδών switches και routers [21].

2.1 Παραδοσιακές αρχιτεκτονικές δικτύων

Τα παραδοσιακά δίκτυα που γνωρίζουμε σήμερα είναι απομονωμένα και διαχωρίζονται με φυσικό τρόπο για να καλύψουν τις ανάγκες της βιομηχανίας, των παρόχων υπηρεσιών, των οργανισμών και των τελικών χρηστών. Στα δίκτυα αυτά τόσο το επίπεδο ελέγχου (control plane) όσο και το επίπεδο δεδομένων (data plane) υλοποιούνται στην ίδια συσκευή, όπως φαίνεται στην Εικόνα 7.



Εικόνα 7: Παραδοσιακή αρχιτεκτονική δικτύου

Παρά το γεγονός ότι η παραδοσιακή αρχιτεκτονική δικτύου έχει ανταπεξέλθει με αξιοθαύμαστο τρόπο στην εξέλιξη της δικτυακής τεχνολογίας, είναι δύσκολο, αν όχι αδύνατο, να ανταποκριθεί το ίδιο ικανοποιητικά στον σημερινό κόσμο και τις νέες απαιτήσεις. Τα τμήματα πληροφορικής των επιχειρήσεων αναζητούν την εικονικοποίηση (virtualization) των περισσοτέρων από τους κεντρικούς υπολογιστές (servers) τους. Μια τέτοια διαδικασία είναι δύσκολη, δεδομένου ότι η ζήτηση τόσο για την εφαρμογή όσο και για την κινητικότητα των χρηστών αυξάνεται ραγδαία.

2.2 Όρια παραδοσιακών δικτύων

Στα πρώτα βήματα της τεχνολογίας δικτύων, είχαμε μόνο να ασχοληθούμε με απλούς κανόνες Ethernet ή IP, και το δίκτυο ήταν στατικό. Σήμερα όμως, οι απαιτήσεις στα δίκτυα καθίστανται όλο και πιο σύνθετες [23], [24], καθώς αυτά γίνονται όλο και πιο δυναμικά και νέες έννοιες για τον διαχωρισμό του ελέγχου και των δεδομένων αναπτύσσονται, ώστε να αντιμετωπίσουν αυτούς τους περιορισμούς [25], [26].

Πράγματι, τα δίκτυα πρέπει να ανταποκρίνονται και να είναι ελαστικά, καθώς πολλοί υπολογιστές συνδέονται, μετακινούνται ή αποσυνδέονται λόγω των πολιτικών BYOD [27], των ασύρματων δικτύων και της κατ' αίτηση υπηρεσίας Cloud computing για παράδειγμα. Επιπλέον, πρέπει να ρυθμιστούν οι αντίστοιχες πολύπλοκες πολιτικές, είτε σε θέματα ασφάλειας (π.χ. απομόνωση μεταξύ χρηστών, κανόνων τείχους προστασίας, λίστες πρόσβασης και διαχείρισης δικαιωμάτων) είτε διαχείρισης της κυκλοφορίας (Quality of Service, load balancing). Τα δίκτυα γίνονται επίσης μεγαλύτερα και τα διαφορετικά δίκτυα πρέπει να συνυπάρχουν (διαφορετικά επίπεδα ασφάλειας στις επιχειρήσεις, πολλαπλοί μισθωτές στο cloud computing) και να έχουν προσαρμόσιμες πολιτικές δικτύου.

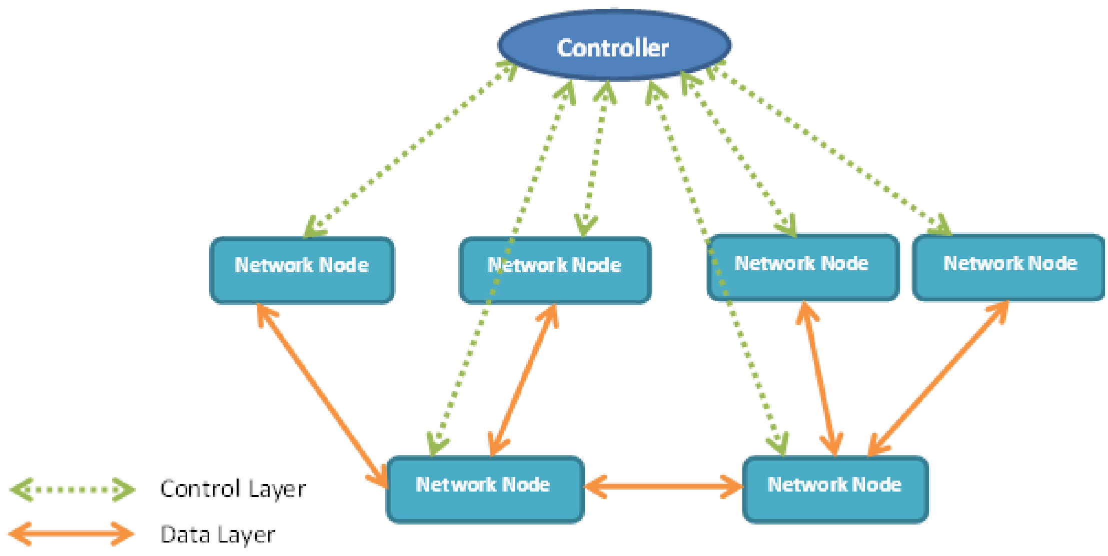
Ταυτόχρονα, χρειάζονται περισσότερες συσκευές υψηλού επιπέδου και δικτύου (switches, routers και middleboxes) και αυτές οι συσκευές πρέπει να διαμορφωθούν ξεχωριστά και συχνά προέρχονται από διαφορετικούς προμηθευτές (και συνεπώς χρησιμοποιούν εντολές που σχετίζονται με προμηθευτές). Τέλος, οι μηχανικοί δικτύων πρέπει να αντιμετωπίσουν όλο και περισσότερη πολυπλοκότητα όσον αφορά τη δρομολόγηση [23], [28] και τη διαχείριση των ροών, καθώς οι ροές κυκλοφορίας αλλάζουν και εξελίσσονται γρήγορα, αλλά και επειδή πρέπει να ληφθούν υπόψη νέες απειλές. Για να γίνει αυτό, μπορούν να βοηθηθούν από διάφορα middleboxes (IDS, Firewalls, NAT) που πρέπει να τοποθετηθούν επαρκώς σε αυτό το συνεχώς μεταβαλλόμενο δίκτυο.

Επομένως, είναι δύσκολο να διαχειριστούν δίκτυα και να εφαρμόσουν αποτελεσματικά νέες πολιτικές, δεδομένου ότι κάθε συσκευή λειτουργεί σε ένα δεδομένο πρωτόκολλο δικτύου και πρέπει να διαμορφωθεί κατάλληλα ώστε να επικοινωνεί με τις άλλες. Έτσι, δεν είναι εύκολο να προστεθεί κάποιος κανόνας, επειδή δεν υπάρχει τυποποιημένος τρόπος για να γίνει αυτό, αλλά και επειδή μπορεί να έρθει σε σύγκρουση με μία από τις πολλές συσκευές που βρίσκονται στη διαδρομή [24]. Επιπλέον, οι κανόνες προορίζονταν αρχικά να είναι στατικοί και συνεπώς το δίκτυο είναι στατικό και δυσπροσάρμοστο στις τρέχουσες ανάγκες και τεχνολογίες.

2.3 Εισαγωγή στην τεχνολογία SDN

Με βάση το RFC 7426 [29], SDN ορίζεται: Μια νέα προσέγγιση στον δικτυακό προγραμματισμό όπου δίνεται η δυνατότητα αρχικοποίησης, ελέγχου και δυναμικής διαχείρισης των δικτυακών συσκευών μέσω ανοιχτών διεπαφών, σε ένα πλαίσιο δημιουργίας δικτύων (framework) όπου υφίσταται φυσικός διαχωρισμός του επιπέδου ελέγχου - Control Plane από το επίπεδο προώθησης - Forwarding Plane.

Αυτός ο διαχωρισμός επιτρέπει τον αυτόνομο σχεδιασμό και την διαχείριση του δικτύου με μια διαφορετική προσέγγιση από την κλασσική. Το επίπεδο ελέγχου (control plane) είναι υπεύθυνο για να παρατηρεί το δίκτυο, να ελέγχει το επίπεδο προώθησης και ουσιαστικά συγκεντρώνει όλη την «δικτυακή ευφυΐα». Λειτουργίες λοιπόν όπως η δρομολόγηση υλοποιούνται σε αυτό το επίπεδο ενημερώνοντας κατάλληλα μέσω διεπαφών τα switches στο επίπεδο προώθησης. Η αρχιτεκτονική που περιγράφεται παρουσιάζεται στην Εικόνα 8.



Εικόνα 8: Αρχιτεκτονική δικτύου SDN

Αυτός ο διαχωρισμός κάνει απλές τις συσκευές προώθησης, καθώς η λογική ελέγχου υλοποιείται σε έναν κεντρικό ελεγκτή (controller), με αποτέλεσμα την εύκολη υλοποίηση πολιτικών, ρυθμίσεων και επέκτασης του δικτύου. Ο διαχωρισμός αυτός πραγματοποιείται από μια προγραμματιζόμενη διεπαφή (interface) μεταξύ των μεταγωγέων και του SDN ελεγκτή. Ο ελεγκτής επιτρέπει τον έλεγχο της κατάστασης των στοιχείων που αποτελούν το επίπεδο δεδομένων μέσω μίας προγραμματιζόμενης διεπαφής εφαρμογής (API), π.χ. του OpenFlow.

Η τεχνολογία του SDN και του Openflow ξεκίνησε αρχικά πειραματικά από την ακαδημαϊκή κοινότητα, αλλά υποστηρίχτηκε γρήγορα από μεγάλες επιχειρήσεις, όπως Google, Facebook, Yahoo, Microsoft, Verizon και Deutsche Telekom, ιδρύοντας έτσι το Open Networking Foundation (ONF) με μοναδικό σκοπό του την εξέλιξη, την προώθηση και την αποδοχή του SDN.

2.3.1 Ιστορική εξέλιξη της τεχνολογίας SDN

Το 1980 υπήρχε η έννοια του κεντρικοποιημένου ελέγχου του δικτύου καθώς τα δεδομένα και ο έλεγχος γίνονταν μέσα από το ίδιο κανάλι, υπήρχαν ειδικές συχνότητες για αρχικοποίηση της γραμμής ή για τη δρομολόγηση της τηλεφωνικής κλήσης. Αργότερα, το 1981 η AT&T διαχώρισε το επίπεδο Δεδομένων από το επίπεδο Ελέγχου μέσω μίας οντότητας που ονομαζόταν Network Point και από εκεί μπορούσαν εύκολα να καταλάβουν αν η γραμμή ήταν δεσμευμένη.

Το 1990 εμφανίστηκαν τα Active Networks σαν μία έννοια των προγραμματιζόμενων δικτύων (Programmable Networks), όπου αποτελούνταν από μεταγωγείς που έκαναν κάποιους υπολογισμούς στα πακέτα, όπως το ίχνος (trace) σε κάθε δρομολογητή, firewalls, proxies και κάποιες εφαρμογές. Έρευνα της DARPA το 1994-95 [30] βρίσκει προβλήματα στα δίκτυα αυτά, όπως δυσκολία προσαρμογής σε νέες τεχνολογίες, μικρή απόδοση λόγω των πολλών επιπέδων πρωτοκόλλων καθώς και δυσκολίες σε νέες υπηρεσίες. Ήταν κάτι καινοτόμο και πήρε 10 χρόνια για να γίνει πρότυπο, να υλοποιηθεί και να αναπτυχθεί. Αποτελείται από active nodes που δίνουν τη δυνατότητα στους δρομολογητές να δίνουν στην υποδομή του δικτύου νέες υπηρεσίες κάτι που το κάνει να μοιάζει στα Software Defined Networks (SDN's). Έχει δύο προσεγγίσεις, το Capsules, όπου το κάθε μήνυμα είναι πρόγραμμα και αξιολογείται από το Node που μεταφέρει τα πακέτα, και τα Programmable Switches όπου συναρτήσεις τρέχουν στους δρομολογητές και τα πακέτα δρομολογούνται μέσω των προγραμματιζόμενων κόμβων καθώς το πρόγραμμα που τρέχει για τη δρομολόγηση του πακέτου εξαρτάται από το περιεχόμενο στο header του πακέτου. Η τεχνολογία αυτή είχε κάποια μειονεκτήματα, όπως ότι δεν ήταν καθαρή η έννοια του χρόνου, είχε ακριβό υλικό, υπήρχαν προβλήματα στην ασφάλεια (τα πακέτα περιέχουν τον κώδικα του προγράμματος) και στη διαλειτουργικότητα του δικτύου, δηλαδή στην ικανότητα να στέλνονται και να λαμβάνονται πακέτα συνεχώς από το δίκτυο για QoS του χρήστη χωρίς να δημιουργούνται προβλήματα στο δίκτυο. Όλα αυτά τα προβλήματα οδήγησαν στο συμπέρασμα ότι έπρεπε να υπάρχει μία υλοποίηση, όπου θα ήταν συμβατή με την προς τα πίσω τεχνολογία και να λειτουργεί με την ίδια υποδομή. Τη λύση αυτή μας δίνει το πρωτόκολλο του Openflow.

Επίσης, το 1990 αναπτύσσεται η ιδέα του Network Virtualization, όπου μας δίνει την αναπαράσταση μίας ή περισσοτέρων δικτυακών τοπολογιών από την ίδια δικτυακή υποδομή. Σαν τέτοιες περιπτώσεις βρίσκουμε τα VLAN's, Tempest, Vini, Cabo, VMWare και Nicira [31]. Μας δίνουν πολλά πλεονεκτήματα, όπως διαμοιρασμό των πόρων καθώς πολλοί δρομολογητές λειτουργούν πάνω σε μία πλατφόρμα και χρησιμοποιούν τη CPU, RAM, Forwarding Tables και Bandwidth καθώς και μία δυνατότητα προσωποποίησης στην οποία μπορούμε να έχουμε το δικό μας λογισμικό για τις πολιτικές δρομολόγησης (routing) και προώθησης (forwarding) μέσω CPU γενικού σκοπού και Network processors & FPGA's για το επίπεδο Δεδομένων.

Όλα αυτά τα στοιχεία αποτελούν μία κληρονομιά για το SDN γιατί:

- Διαχωρίζονται οι υπηρεσίες από την υποδομή του δικτύου.
- Μπορούμε να έχουμε πολλαπλούς Controllers σε ένα Switch.
- Μπορούμε να έχουμε πολλές λογικές τοπολογίες.

Για να γίνει όμως ο έλεγχος σε ένα δίκτυο μεταγωγής πακέτων θα πρέπει να έχουμε τον διαχωρισμό του επιπέδου Ελέγχου (Control Plane) από το επίπεδο Δεδομένων (Data Plane).

Το επιπέδου Ελέγχου (Control Plane) αποτελεί τη λογική για τον έλεγχο της συμπεριφοράς προώθησης των πακέτων, π.χ. routing πρωτόκολλα, φίλτρα, firewalls, NAT, security, load balancer.

Το επίπεδο Δεδομένων (Data Plane) είναι υπεύθυνο για τη προώθηση της κίνησης σύμφωνα με τη λογική του επιπέδου Ελέγχου, π.χ. ip forwarding – Layer 2 Switching.

Ο διαχωρισμός των επιπέδων αυτών μας οδηγεί σε κάποια πλεονεκτήματα:

- **Ταχύτερη ανάπτυξη/εξέλιξη:** Η λογική του ελέγχου δεν είναι δεμένη με το hardware και έτσι έχουμε την ανεξάρτητη εξέλιξη και ανάπτυξη του λογισμικού του Controller του δικτύου σε σχέση με την υποδομή του δικτύου.
- **Ευρεία απεικόνιση του δικτύου:** Μπορούμε να συμπεράνουμε ευκολότερα τη συμπεριφορά του δικτύου, επιτρέποντας έτσι τον έλεγχο του δικτύου από ένα high-level πρόγραμμα που βλέπουμε το δίκτυο, την αποσφαλμάτωσή του και το τσεκάρισμα των λειτουργιών του.
- **Ευελιξία:** Παρουσίαση των νέων υπηρεσιών πιο εύκολα.

Η ανάπτυξη του ελέγχου στα δίκτυα μεταγωγής πακέτων ξεκίνησε από το 2003. Εκεί το ForCES [32] χρησιμοποιεί πρωτόκολλα για πολλαπλά στοιχεία ελέγχου (Control Elements) και στοιχεία προώθησης της κίνησης (Forwarding Elements) δίνει μία ιδέα σαν το Openflow αλλά χρειάζεται να γίνει πρότυπο, να υιοθετηθεί και να δημιουργηθεί νέο υλικό.

Το 2004 υπήρξε η ιδέα να χρησιμοποιηθούν υπάρχοντα πρωτόκολλα για τον υπολογισμό των διαδρομών με σκοπό την προώθηση των πακέτων και ονομάστηκε Routing Control Platform [32]. Εδώ, ο υπολογισμός γίνεται σε ένα σημείο του

Αυτόνομου Συστήματος (Autonomous System) και όταν βρεθούν οι διαδρομές διαμοιράζονται μέσω του πρωτοκόλλου iBGP, αλλά ο έλεγχος περιορίζεται από την υποστήριξη που έχουν τα υπάρχοντα πρωτόκολλα.

Το 2007 δημιουργήθηκε το Ethane [32] και λειτούργησε ως αρχιτεκτονική δικτύου για επιχειρήσεις. Είχε ως βάση τις άμεσες εκτελούμενες πολιτικές δικτύου. Ο Domain Controller υπολογίζει τα flow tables με βάση τις πολιτικές που υπάρχουν αλλά χρειάζονται τροποποιημένα switches, όπως OpenWrt-NetFPGA και Linux που να υποστηρίζουν το Ethane.

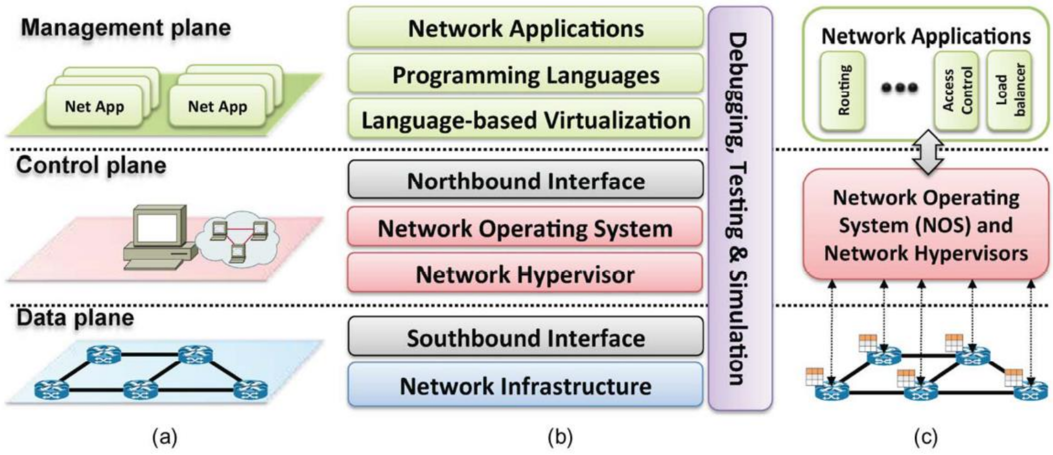
Το 2008 έχουμε την παρουσίαση του Openflow. Ο controller επικοινωνεί με τους forwarding tables του switch και αυτό που χρειάζεται είναι να ανοίξουν οι κατασκευαστές μία διεπαφή (interface) έτσι ώστε να δέχεται ο μεταγωγέας τις ροές κίνησης από τον ελεγκτή.

Συνοψίζοντας, ενδεικτικά, μερικά προβλήματα που μπορούμε να επέμβουμε μέσω του SDN είναι τα εξής:

- Δρομολόγηση βάση προτεραιοτήτων
- Προώθηση & δρομολόγηση πακέτων
- Πρόβλεψη και καταμερισμός κίνησης
- Έλεγχος συμφόρησης
- Ανακάλυψη και εξαγωγή της τοπολογίας του δικτύου

2.3.2 Αρχιτεκτονική SDN

Μια αρχιτεκτονική SDN μπορεί να περιγραφεί ως μια σύνθεση των διαφορετικών επιπέδων δικτύου, όπως φαίνεται στην Εικόνα 9. Υπάρχουν τρία βασικά επίπεδα, το επίπεδο εφαρμογών ή επίπεδο διαχείρισης, από το οποίο οι εφαρμογές μέσω του Northbound API επικοινωνούν με το επίπεδο ελέγχου, ώστε να εκφράσουν τις ανάγκες του δικτύου. Το επίπεδο ελέγχου, το οποίο αποτελεί την «ευφυΐα του δικτύου» όπως έχουμε ήδη εξηγήσει. Και το επίπεδο δεδομένων στο οποίο ανήκουν οι δικτυακές συσκευές, κι εκεί γίνεται η προώθηση των πακέτων με τρόπο που έχει ορίσει το επίπεδο ελέγχου μέσω του Southbound API [21], [33].

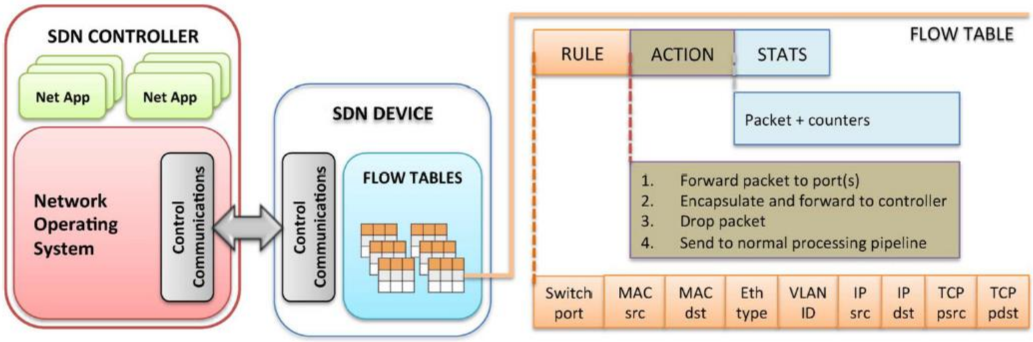


Εικόνα 9: Αρχιτεκτονική επιπέδων SDN [31]

Η Εικόνα 9(a) παρουσιάζει τη γενική αρχιτεκτονική του SDN. Τα στρώματα του SDN φαίνονται στην Εικόνα 9(b) και μία απεικόνιση της αρχιτεκτονικής σχεδιασμού συστήματος έχουμε στην Εικόνα 9(c).

2.3.2.1 Υποδομή SDN

Η υποδομή του SDN είναι σχεδόν ίδια με αυτή των παραδοσιακών δικτύων, δηλαδή αποτελείται από δικτυακό εξοπλισμό (δρομολογητές, μεταγωγείς, κλπ). Η κυριότερη διαφορά βρίσκεται στο γεγονός ότι οι φυσικές συσκευές είναι απλά στοιχεία προώθησης της κίνησης χωρίς να έχουν κάποιο ενσωματωμένο λογισμικό για τον έλεγχο, ούτε λαμβάνουν αποφάσεις προώθησης των πακέτων. Η «εξυπνάδα» του δικτύου βρίσκεται σε ένα κεντρικό σύστημα ελέγχου το οποίο ονομάζεται Network Operating System (NOS). Αυτά τα δίκτυα βρίσκονται πάνω από κάποιες διεπαφές (Openflow), οι οποίες επιτρέπουν τον δυναμικό προγραμματισμό ετερογενών στοιχείων προώθησης κάτι που ήταν δύσκολο μέχρι τώρα.



Εικόνα 10: Υποδομή SDN [31]

Σε μια αρχιτεκτονική SDN/OpenFlow υπάρχουν δύο κύρια στοιχεία, οι ελεγκτές (SDN Controller) και οι συσκευές προώθησης (SDN device), όπως φαίνεται στην Εικόνα 10. Μια συσκευή του επιπέδου δεδομένων είναι ένα υλικό ή στοιχείο λογισμικού που ειδικεύεται στην προώθηση των πακέτων, ενώ ελεγκτής είναι ένα λογισμικό (ο «εγκέφαλος του δικτύου») που τρέχει σε ένα μηχάνημα. Μια OpenFlow enabled συσκευή προώθησης βασίζεται σε έναν αγωγό που έχει πίνακες-ροής (flow tables), όπου κάθε εγγραφή ενός πίνακα-ροής εξυπηρετεί τρεις λειτουργίες:

1. Να αντιστοιχηθούν οι ροές με τους κανόνες
2. Ενέργειες που πρέπει να εκτελεστούν όταν έχουμε αντιστοιχία των πακέτων
3. Μετρητές που κρατούν στατιστικά στοιχεία των πακέτων.

Αυτό το υψηλού επιπέδου και απλοποιημένο μοντέλο που προέρχεται από το OpenFlow είναι σήμερα ο πιο διαδεδομένος σχεδιασμός του SDN για τις συσκευές στο επίπεδο δεδομένων. Παρ’ όλα αυτά, υπάρχουν και άλλες προδιαγραφές για τις συσκευές προώθησης που υποστηρίζουν SDN, όπως οι POF και Negotiable Datapath Models (NDMs) [33].

Μέσα σε μια συσκευή OpenFlow ένα μονοπάτι καθορίζεται μέσα από μια αλληλουχία πινάκων-ροής για το πώς πρέπει να αντιμετωπιστούν τα πακέτα. Όταν φθάνει ένα νέο πακέτο, η διαδικασία αναζήτησης ξεκινάει από τον πρώτο πίνακα και τελειώνει είτε με μία αντιστοίχιση σε έναν από τους πίνακες της αλληλουχίας ή με μια αστοχία (όταν δεν υπάρχει κανόνας για το πακέτο). Ένας κανόνας-ροής μπορεί να καθορίζεται από το συνδυασμό διαφορετικών πεδίων που ταιριάζουν, όπως απεικονίζεται στην Εικόνα 10. Εάν δεν υπάρχει κανένας κανόνας, το πακέτο θα απορριφθεί. Ωστόσο, η κοινή πρακτική είναι να εγκαταστήσουμε έναν προεπιλεγμένο κανόνα που λέει στον μεταγωγέα να στείλει το πακέτο στον ελεγκτή OpenFlow ή ακόμα και σε μεταγωγείς που δεν υποστηρίζουν OpenFlow. Οι προτεραιότητες των κανόνων ακολουθούν τη φυσική αλληλουχία του αριθμού των πινάκων και της κλάσης των σειρών σε έναν πίνακα-ροής. Οι δυνατές ενέργειες που μπορεί να δεχτεί ένα πακέτο είναι:

1. Να προωθηθεί το πακέτο στην εξερχόμενη πόρτα (ες)
2. Να ενθυλακωθεί το πακέτο και να προωθηθεί στον ελεγκτή
3. Να απορριφθεί το πακέτο
4. Να σταλεί με την κανονική αλληλουχία της επεξεργασίας

5. Να σταλεί σε έναν επόμενο πίνακα-ροής ή σε ειδικούς πίνακες, όπως ομάδες ή πίνακες-μέτρησης που εισάγονται στην τελευταία έκδοση του πρωτοκόλλου OpenFlow.

2.3.2.2 Southbound API

Τα Southbound Interfaces (ή Southbound APIs) είναι ένα είδους «γέφυρες» ανάμεσα στον έλεγχο και στα στοιχεία προώθησης που επιτρέπουν σε ένα SDN δίκτυο να επικοινωνήσει με συστατικά κατώτερων επιπέδων. Σαν ένα κεντρικό στοιχείο στο σχεδιασμό του SDN, τα Southbound APIs αποτελούν ένα φραγμό για την εισαγωγή και αποδοχή οποιασδήποτε νέας τεχνολογίας στη δικτύωση. Υπό αυτό το πρίσμα, η εμφάνιση των SDN προτάσεων για τα Southbound APIs, όπως το OpenFlow θεωρούνται ευπρόσδεκτες από πολλούς στον κλάδο.

Αυτά τα πρότυπα προώθησης της διαλειτουργικότητας, επιτρέπουν η ανάπτυξη των συσκευών δικτύου να γίνεται ανταγωνιστική. Αυτό έχει ήδη αποδειχθεί από την διαλειτουργικότητα μεταξύ των OpenFlow-enabled συσκευών από διαφορετικούς προμηθευτές.

Το OpenFlow είναι η πιο ευρέως αποδεκτή λύση και αναπτύσσεται σαν ανοικτό πρότυπο Southbound API για το SDN. Παρέχει μια κοινή προδιαγραφή για τις OpenFlow-enabled συσκευές προώθησης, καθώς και για την επικοινωνία μέσω ενός καναλιού μεταξύ των συσκευών του επιπέδου δεδομένων και του επίπεδου ελέγχου (π.χ. μεταγωγείς και ελεγκτές).

Το πρωτόκολλο OpenFlow παρέχει τις ακόλουθες τρεις πηγές πληροφόρησης στο NOS:

- Μηνύματα βασισμένα στα γεγονότα αποστέλλονται από συσκευές προώθησης στον ελεγκτή όταν μία σύνδεση ή μία πόρτα αλλάζει.
- Στατιστικά στοιχεία ροών που δημιουργούνται από τις συσκευές προώθησης στέλνονται και συλλέγονται από τον ελεγκτή.
- Μηνύματα τύπου packet-in αποστέλλονται από τις συσκευές προώθησης στον ελεγκτή όταν δεν γνωρίζουν τι πρέπει να κάνουν με τη νέα εισερχόμενη ροή ή επειδή υπάρχει ενέργεια τύπου «send to controller» αντιστοιχισμένη στην είσοδο του πίνακα-ροής.

Αυτές οι πληροφορίες είναι βασικές για να παρέχουν πληροφορίες του επιπέδου-ροής στο NOS. Παρ' όλα αυτά, υπάρχουν και άλλες προδιαγραφές για την υλοποίηση Southbound API's, όπως ForCES, Open vSwitch Database, POF, OpFLEX, OpenState, κλπ [33].

2.3.2.3 Επόπτες Δικτύου (Network Hypervisors)

Ένα από τα ενδιαφέροντα χαρακτηριστικά των τεχνολογιών εικονικοποίησης (virtualization) σήμερα, είναι το γεγονός ότι οι εικονικές μηχανές μπορούν εύκολα να μεταφερθούν από ένα φυσικό server στον άλλο και ότι μπορούν να δημιουργηθούν ή και να διαγραφούν κατά απαίτηση, επιτρέποντας έτσι την ευέλικτη και εύκολη διαχείρισή τους.

Παρά τη μεγάλη πρόοδο στην εικονικοποίηση υπολογιστών και των στοιχείων αποθήκευσης, το δίκτυο εξακολουθεί να παραμένει στατικό, ρυθμιζόμενο στην εικονικοποίηση με ένα τρόπο box-by-box, δηλαδή την κάθε μία δικτυακή συσκευή ανεξάρτητα από τις υπόλοιπες. Οι κύριες απαιτήσεις του δικτύου αφορούν την τοπολογία του δικτύου και τον χώρο διευθύνσεων. Διαφορετικό φόρτο εργασίας απαιτούν διαφορετικές τοπολογίες δικτύου και υπηρεσίες, όπως υπηρεσίες επιπέδου L2 ή L3, ή ακόμη περισσότερο σύνθετες υπηρεσίες επιπέδων L4-L7 για προηγμένη λειτουργικότητα. Επί του παρόντος, είναι πολύ δύσκολο για μια ενιαία φυσική τοπολογία να υποστηρίξει τις ποικίλες απαιτήσεις των εφαρμογών και υπηρεσιών.

Ομοίως, ο χώρος IP διευθύνσεων είναι δύσκολο να αλλάξει στα τρέχοντα δίκτυα. Ως εκ τούτου, είναι δύσκολο να κρατήσουμε τις αρχικές δικτυακές ρυθμίσεις ενός πελάτη, αφού οι εικονικές μηχανές δεν μπορούν να μεταφερθούν σε αυθαίρετες τοποθεσίες και το σχήμα των IP διευθύνσεων να είναι σταθερό και δύσκολο να αλλάξει. Για παράδειγμα, το IPv6 δεν μπορεί να χρησιμοποιηθεί από τις εικονικές μηχανές (VMs) του μισθωτή, εφόσον το φυσικό επίπεδο που αποτελείται από συσκευές προώθησης υποστηρίζει μόνο IPv4.

Για να παρέχεται πλήρης εικονικοποίηση, το δίκτυο θα πρέπει να παρέχει παρόμοιες ιδιότητες με το υπολογιστικό στρώμα. Η δικτυακή υποδομή θα πρέπει να είναι σε θέση να υποστηρίζει αυθαίρετα τις τοπολογίες δικτύου και τα σχήματα των διευθύνσεων. Κάθε πελάτης θα πρέπει να έχει τη δυνατότητα να ρυθμίζει τους υπολογιστικούς κόμβους και το δίκτυο ταυτόχρονα. Η μεταφορά των hosts θα πρέπει να ενεργοποιήσει αυτόματα τη τροποποίηση των αντίστοιχων εικονικών θυρών του δικτύου. Θα μπορούσε κανείς να σκεφτεί ότι τεχνικές virtualization, όπως τα VLANs (εικονοποίηση επιπέδου L2), το NAT (εικονοποίηση χώρου διευθύνσεων IP) και το MPLS (εικονοποίηση διαδρομής) είναι αρκετά για να παρέχουν πλήρη και αυτοματοποιημένη εικονικοποίηση δικτύου. Ωστόσο, αυτές οι τεχνολογίες είναι προσαρμοσμένες σε ρυθμίσεις box-by-box, δηλαδή, δεν υπάρχει ένα ενιαίο abstract interface ώστε να ρυθμιστεί (ή να επαναρυθμιστεί) το δίκτυο με ένα συνολικό τρόπο. Κατά συνέπεια, η δικτυακή παραμετροποίηση μπορεί να πάρει μήνες, ενώ η υπολογιστική διαρκεί μόνο λίγα λεπτά.

2.3.2.4 Λειτουργικά Συστήματα Δικτύων/Ελεγκτές

Τα παραδοσιακά λειτουργικά συστήματα παρέχουν αφαιρέσεις (π.χ., υψηλού επιπέδου APIs) για την πρόσβαση σε χαμηλότερου επίπεδου συσκευές, τη διαχείριση και την ταυτόχρονη πρόσβαση σε υποκειμενικούς πόρους (π.χ. μονάδα σκληρού δίσκου, προσαρμογέα δικτύου, CPU, μνήμη), και παρέχουν μηχανισμούς προστασίας για την ασφάλεια. Αυτές οι λειτουργίες και οι πόροι είναι οι κύριοι μοχλοί για την αύξηση της παραγωγικότητας, κάνοντας τη χρήση του συστήματος και των εφαρμογών πιο εύκολη για τους προγραμματιστές. Στην ευρεία χρήση τους έχει συμβάλει η εξέλιξη των διαφόρων συστημάτων (π.χ. γλώσσες προγραμματισμού) και η ανάπτυξη πολλών εφαρμογών. Σε αντίθεση, τα δίκτυα μέχρι στιγμής διαχειρίζονται και ρυθμίζονται χρησιμοποιώντας χαμηλού επιπέδου εντολές, προσαρμοσμένες στο υλικό από κλειστά λειτουργικά συστήματα NOSs (π.χ. Cisco IOS και Juniper Junos). Επιπλέον, η ιδέα των λειτουργικών συστημάτων να αφαιρεί χαρακτηριστικά συγκεκριμένων συσκευών και να παρέχει, με διαφανή τρόπο, κοινές λειτουργίες είναι κάτι που ακόμα απουσιάζει από τα δίκτυα. Για παράδειγμα, σήμερα οι σχεδιαστές των πρωτοκόλλων δρομολόγησης πρέπει να ασχοληθούν με πολύπλοκους κατανεμημένους αλγορίθμους κατά την επίλυση προβλημάτων δικτύωσης. Έτσι οι επαγγελματίες δικτύων λύνουν τα ίδια προβλήματα ξανά και ξανά.

Το SDN έχει υποσχεθεί να διευκολύνει τη διαχείριση του δικτύου και να ελαφρύνει το βάρος της επίλυσης των προβλημάτων διαχείρισης, μέσω της δικτύωσης του λογικά δοσμένου κεντρικού ελέγχου που προσφέρεται από ένα NOS. Όπως και με τα παραδοσιακά λειτουργικά συστήματα, η κρίσιμη αξία του NOS είναι να παρέχει αφαιρέσεις, βασικές υπηρεσίες και κοινά API για τους προγραμματιστές. Κάποιες υπηρεσίες που παρέχονται από το NOS είναι η γενική λειτουργία της κατάστασης του δικτύου και πληροφορίες σχετικές με τη δικτυακή τοπολογία, η αναζήτηση συσκευών και η κατανομή των δικτυακών ρυθμίσεων. Με το NOS για παράδειγμα, ο προγραμματιστής δεν θα χρειάζεται πλέον να νοιάζεται για τις λεπτομέρειες χαμηλού επιπέδου της διανομής δεδομένων μεταξύ των στοιχείων δρομολόγησης ώστε να καθορίσει τις πολιτικές του δικτύου. Τέτοια συστήματα μπορούν αναμφισβήτητα να δημιουργήσουν ένα νέο περιβάλλον που μπορεί να ενισχύσει την καινοτομία με ταχύτερο ρυθμό, μειώνοντας την εν γένει πολυπλοκότητα της δημιουργίας νέων πρωτοκόλλων δικτύου και δικτυακών εφαρμογών. Παρόμοια με ένα παραδοσιακό λειτουργικό σύστημα, η πλατφόρμα ελέγχου αφαιρεί το χαμηλότερο επίπεδο λεπτομερειών της σύνδεσης και αλληλεπιδρά με τις συσκευές προώθησης (δηλαδή, την υλοποίηση πολιτικών δικτύου).

2.3.2.5 Northbound API

Τα Northbound και Southbound APIs αποτελούν τις δύο βασικές αφαιρέσεις του συστήματος SDN, που επιτρέπουν την επικοινωνία με τα ανώτερα και τα κατώτερα επίπεδα δικτύου, αντίστοιχα.

Ενώ η διεπαφή Southbound έχει ήδη μία ευρέως αποδεκτή πρόταση, το OpenFlow, μια κοινή διεπαφή Northbound εξακολουθεί να είναι ένα ανοικτό ζήτημα. Αυτή τη στιγμή δεν μπορεί να οριστεί ένα πρότυπο για διεπαφή Northbound, καθώς οι περιπτώσεις χρήσης εξακολουθούν να είναι σε επεξεργασία. Εν πάση περιπτώσει, μια κοινή διεπαφή Northbound αναμένεται να προκύψει καθώς το SDN εξελίσσεται. Μια αφαίρεση που θα επιτρέπει στις εφαρμογές δικτύου να μην εξαρτώνται από τις συγκεκριμένες υλοποιήσεις, είναι σημαντικό να διερευνηθεί πλήρως για το SDN. Η διεπαφή Northbound είναι ως επί το πλείστον ένα σύστημα λογισμικού και δεν είναι υλικό, όπως είναι η περίπτωση του Southbound API. Η υλοποίηση γίνεται συνήθως με τη χρήση REST APIs σε συνδυασμό με http πρωτόκολλα και json εφαρμογές, τα οποία βοηθούν στην πρόσβαση και επικοινωνία με τον ελεγκτή μέσω browser.

2.3.2.6 Επικοινωνία East-West (Intercontroller)

Εκτός της Northbound και Southbound επικοινωνίας, οι ελεγκτές έχουν τη δυνατότητα για East-West ή αλλιώς Intercontroller επικοινωνία, δηλαδή την διασύνδεση μεταξύ τους. Αυτή η δυνατότητα αξιοποιείται σε Multicontroller αρχιτεκτονικές, όπου το δίκτυο εργάζεται με άνω του ενός ελεγκτών.

Η East-West επικοινωνία μπορεί να περιλαμβάνει τον συγχρονισμό των ελεγκτών μεταξύ τους ώστε σε περίπτωση βλάβης του ενός, τον ρόλο της διαχείρισης του δικτύου να αναλάβει ο επόμενος, ή για να υπάρξει συνεργασία όσον αφορά ελεγκτές που βρίσκονται σε διαφορετικά μέρη του δικτύου και λειτουργούν παράλληλα μοιράζοντας μεταξύ τους πληροφορίες για τα δίκτυά τους (π.χ. τοπολογίες και κατάσταση διαφόρων δικτυακών στοιχείων) δημιουργώντας έτσι μια πλήρη εικόνα των διασυνδεδεμένων δικτύων-ελεγκτών.

2.3.2.7 Γλώσσες Προγραμματισμού

Οι γλώσσες προγραμματισμού έχουν πολλαπλασιαστεί στο πέρασμα των δεκαετιών. Τόσο τα πανεπιστήμια όσο και η βιομηχανία έχουν συμμετάσχει στην εξέλιξη των χαμηλού επιπέδου γλωσσών μηχανής για συγκεκριμένα υλικά, όπως assembly x86, και των υψηλού επιπέδου ισχυρών γλωσσών προγραμματισμού, όπως οι Java και Python. Οι εξελίξεις προς κώδικα με χαρακτηριστικά μεταφερσιμότητας και επαναχρησιμοποίησης έχουν οδηγήσει σε σημαντική στροφή τη βιομηχανία ηλεκτρονικών υπολογιστών.

Ομοίως, η δυνατότητα προγραμματισμού στα δίκτυα έχει αρχίσει να μετακινείται από τις γλώσσες μηχανής χαμηλού επιπέδου, στις γλώσσες προγραμματισμού υψηλού επιπέδου. Οι γλώσσες μηχανής μιμούνται τη συμπεριφορά των συσκευών προώθησης, αναγκάζοντας τους προγραμματιστές να ξοδεύουν πάρα πολύ χρόνο σε χαμηλού επιπέδου λεπτομέρειες αντί να λύσουν το πρόβλημα.

Πρώτα τα προγράμματα πρέπει να ασχοληθούν με τις λεπτομέρειες της συμπεριφοράς του υλικού, όπως η επικάλυψη και η προτεραιότητα των κανόνων και τα flow rules που εγκαθίστανται στο επίπεδο δεδομένων. Η χρήση των γλωσσών χαμηλού επιπέδου καθιστά δύσκολη την επαναχρησιμοποίηση λογισμικού και οδηγεί σε μια διαδικασία περισσότερο επιρρεπή σε σφάλματα ανάπτυξης.

Οι αφαιρέσεις που παρέχονται από τις γλώσσες προγραμματισμού υψηλού επιπέδου μπορούν να βοηθήσουν σημαντικά στην αντιμετώπιση πολλών προκλήσεων. Στο SDN μπορούν να σχεδιαστούν και να χρησιμοποιηθούν γλώσσες προγραμματισμού υψηλού επιπέδου για:

1. Να δημιουργηθεί υψηλότερο επίπεδο αφαίρεσης για την απλούστευση των διεργασιών προγραμματισμού των συσκευών προώθησης.
2. Να επιτρέπει πιο παραγωγικά και εστιασμένα στο πρόβλημα περιβάλλοντα για τους προγραμματιστές λογισμικού του δικτύου.
3. Να προωθήσει αρθρωτό λογισμικό και επαναχρησιμοποίηση του κώδικα στο επίπεδο ελέγχου του δικτύου.
4. Τη συμβολή στην ανάπτυξη του virtualization για το δίκτυο.

2.3.2.8 Εφαρμογές Δικτύου

Οι εφαρμογές των δικτύων είναι εκείνες που μπορούν να εφαρμόσουν τη λογική ελέγχου που θα μεταφράζεται σε εντολές για να εγκατασταθούν στο επίπεδο δεδομένων, καθορίζοντας στη συνέχεια, την συμπεριφορά των συσκευών προώθησης.

Ως παράδειγμα μπορούμε να δούμε μία απλή εφαρμογή δρομολόγησης. Η λογική της εφαρμογής αυτής είναι να ορίσει τη διαδρομή μέσω της οποίας τα πακέτα θα μεταφέρονται από το σημείο Α στο σημείο Β. Για την επίτευξη αυτού του στόχου, η αίτηση δρομολόγησης βασίζεται στην τοπολογία, ώστε να αποφασίσει σχετικά με την διαδρομή (path) που θα χρησιμοποιηθεί, και να δώσει εντολή στον ελεγκτή να εγκαταστήσει τους κατάλληλους κανόνες προώθησης σε κάθε συσκευή προώθησης που περιλαμβάνεται στην διαδρομή από το Α στο Β.

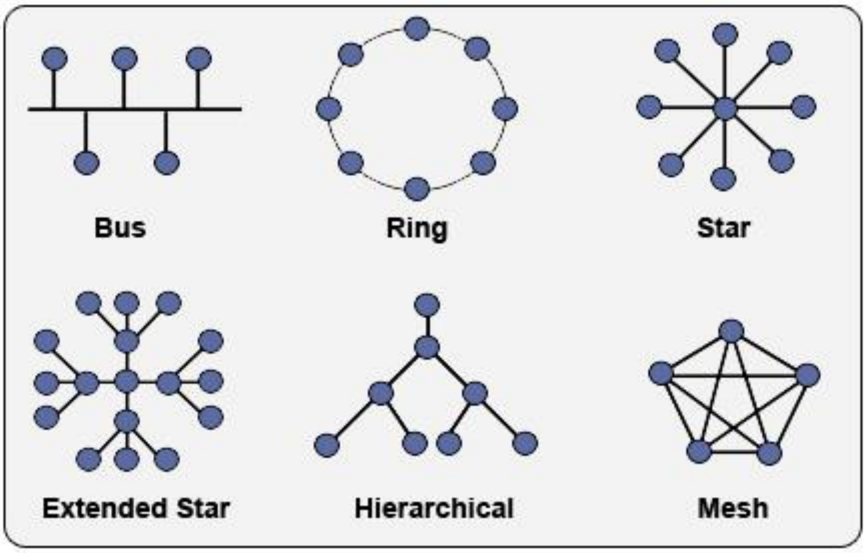
Το SDN μπορεί να αναπτυχθεί σε οποιοδήποτε παραδοσιακό περιβάλλον δικτύου, από το σπίτι μέχρι τα επιχειρησιακά δίκτυα των Data Centers και τα σημεία ανταλλαγής Διαδικτύου (Internet exchange points). Μια τέτοια ποικιλία περιβαλλόντων οδήγησε σε ένα ευρύ φάσμα εφαρμογών δικτύου. Οι υπάρχουσες εφαρμογές δικτύου επιτελούν παραδοσιακές λειτουργίες, όπως η δρομολόγηση, η εξισορρόπηση φορτίου (Load Balancing), οι πολιτικές ασφαλείας κ.α. Ωστόσο βοήθησε και στο να διερευνηθούν νέες προσεγγίσεις, όπως η μείωση κατανάλωσης ενέργειας (ενεργοαποδοτικά δίκτυα), μελέτη σεναρίων ανάκτησης από βλάβη (fail-over), αξιοπιστία λειτουργιών στο επίπεδο των δεδομένων, end-to-end QoS, virtualization του δικτύου και διαχείριση της κινητικότητας σε ασύρματα δίκτυα.

Η ποικιλία των εφαρμογών δικτύου σε συνδυασμό με την ανάπτυξη πραγματικών περιπτώσεων χρήσης, αναμένονται να είναι ο μοχλός για την προώθηση και την ευρεία υιοθέτηση των SDNs. Παρά τη μεγάλη ποικιλία των περιπτώσεων χρήσης, οι περισσότερες εφαρμογές SDN μπορούν να ομαδοποιηθούν σε μία από τις παρακάτω έξι κατηγορίες:

1. Μηχανισμός δικτυακής κίνησης
2. Κινητικότητα χρηστών και Ασύρματη σύνδεση
3. Μετρήσεις και παρακολούθηση δικτύου
4. Ασφάλεια και αξιοπιστία δικτύου
5. Δικτύωση Κέντρου Δεδομένων
6. Κατασκευή online καταστημάτων που προσφέρουν έτοιμες εφαρμογές SDN

2.3.3 Ορισμός τοπολογίας δικτύου

Τοπολογία ορίζουμε τις συνδέσεις (φυσικές και λογικές) μεταξύ των στοιχείων ενός δικτύου. Σε τελικό επίπεδο είναι η σχέση μεταξύ δυο βασικών δομικών στοιχείων, αυτή των κόμβων και των ζεύξεων (nodes & links). Οι κόμβοι μπορεί να είναι είτε συνδετικοί (connecting), είτε τερματικοί (terminal). Μερικές απλές και συνηθισμένες αναπαραστάσεις τοπολογιών από τον συνδυασμό των οποίων προκύπτουν και πολυπλοκότερες φαίνονται στην Εικόνα 11:



Εικόνα 11: Τοπολογίες δικτύου [34]

Αναφορικά οι βασικές αυτές τοπολογίες είναι οι εξής έξι:

- Διαύλου
- Δακτυλίου
- Αστέρα

- Επεκταμένου αστέρα
- Ιεραρχική – δέντρου
- Πλέγματος

Η γνώση της τοπολογίας ενός δικτύου, είναι μια πολύ σημαντική πληροφορία που εξυπηρετεί τους σκοπούς μας όσον αφορά τις υπηρεσίες που δημιουργούμε πάνω από αυτό. Αυτή η πληροφορία είναι ικανή να επιτρέψει την πλήρη εποπτεία του δικτύου ανά πάσα στιγμή, μειώνοντας παράλληλα τον χρόνο που χρειάζεται ένας διαχειριστής για να συλλέξει πληροφορίες για τα υπάρχοντα συστήματα. Αυτές μπορούν έπειτα να χρησιμοποιηθούν ώστε να επιλυθεί κάποιο πρόβλημα που μπορεί να προκύψει σε περίπτωση βλάβης, διακοπής δικτύου κλπ. (troubleshooting) ή και σε περίπτωση αλλαγής διαχειριστή όπου δεν θα υπάρχει καταγεγραμμένη καμία γνώση των υποδομών του δικτύου.

2.3.3.1 Μέθοδοι & πρωτόκολλα κατασκευής τοπολογίας

Οι μεθοδολογίες που ακολουθήθηκαν έως τώρα για την κατασκευή τοπολογιών διακρίνονται σε 2 βασικές κατηγορίες:

1. Αυτές που χρησιμοποιούν πρωτόκολλα διαχείρισης για την συλλογή πληροφοριών
2. Αυτές που συμμετέχουν στην διαδικασία της δρομολόγησης ενεργά και παρακολουθούν την ροή των πακέτων για την κατανόηση των συνδέσεων.

Ανεξαρτήτου μεθοδολογίας όμως, η αναπαράσταση και εκμετάλλευση των κόμβων είναι πολύ σημαντική διαδικασία, ειδικά σε ένα πολύπλοκο αφηρημένο/εικονικοποιημένο περιβάλλον με πολλαπλά φυσικά μηχανήματα και VMs (Virtual Machines ή εικονικές μηχανές). Παρ' όλα αυτά, εφόσον πρόκειται για L2 δίκτυο, η διαδικασία ανακάλυψης της τοπολογίας θα είναι όμοια για κάθε κλίμακα.

Όσον αφορά τις διαθέσιμες εφαρμογές που είναι ικανές να προσδιορίζουν την τοπολογία του δικτύου υπάρχουν διάφορες, ανοιχτού κώδικα και μη. Αναφορικά η ανοιχτού κώδικα πλατφόρμα OpenNMS υποστηρίζει τα δημοφιλέστερα πρωτόκολλα και μπορεί να δημιουργήσει ένα γράφημα της τοπολογίας του δικτύου βασιζόμενη στα πακέτα που ανταλλάσσονται στο δίκτυο. Το κύριο μειονέκτημα αυτών των «παραδοσιακών» εφαρμογών και μεθόδων είναι ότι δεν είναι σαφώς καθορισμένη η διαδικασία για την συλλογή πληροφοριών του δικτύου. Απαιτείται συνήθως, ακόμα ένας σταθμός διαχείρισης από όπου πρέπει να περνάει πρώτα η κίνηση ώστε να υπόκειται σε επεξεργασία. Μόνο αφού γίνει αυτό μπορούν αυτές οι πληροφορίες να διατεθούν στις όποιες εφαρμογές.

Λύσεις όπως οι προηγούμενες όμως, υστερούν της δυναμικότητας και προσαρμοστικότητας που χρειάζεται ένα σύγχρονο δίκτυο και ειδικά ένα SDN δίκτυο για να λειτουργήσει. Χρειαζόμαστε δηλαδή ενεργά στοιχεία ικανά να αλληλεπιδρούν στο δίκτυο.

Τα πρωτόκολλα που χρησιμοποιήθηκαν στο παρελθόν για την απόκτηση πληροφοριών σε δίκτυα 2ου επιπέδου ήταν κυρίως κλειστού κώδικα (proprietary) ή vendor specific (π.χ. Nortel DP, ExtremeDP, Microsoft LLTD) εξαρτώμενα πλήρως από το υλικό του κατασκευαστή που έτρεχαν (hardware specific) όπως το γνωστό πρωτόκολλο της Cisco, Cisco Discovery Protocol. Για την κάλυψη λοιπόν αυτής της ανάγκης όσον αφορά την κατασκευή τοπολογιών συχνά γινόταν χρήση πρωτοκόλλων τα οποία δεν ήταν προσανατολισμένα για αυτό τον σκοπό.

Ένα παράδειγμα είναι η χρησιμοποίηση του ARP (Address Resolution Protocol) που χρησιμοποιούνταν από εφαρμογές οι οποίες μέσω κάποιων αλγοριθμικών διαδικασιών, και σε συνδυασμό με πρωτόκολλα διαχείρισης (SNMP) εξήγαγαν το επιθυμητό αποτέλεσμα. Την λύση σε αυτό το πρόβλημα έδωσε η IEEE με το standard IEEE 802.1AB ή αλλιώς Link Layer Discovery Protocol (LLDP).

To LLDP είναι ένα vendor-neutral πρωτόκολλο που λειτουργεί στο επίπεδο 2 της OSI stack και παρέχει μια ευρεία γκάμα πληροφοριών. Μεταξύ αυτών είναι οι γείτονες, οι δυνατότητες και η ταυτότητα της συσκευής. Παρακάτω αναλύουμε το πρωτόκολλο αυτό και αναφερόμαστε ειδικότερα στα πλεονεκτήματα και τις ιδιότητες του που το καθιστούν κατάλληλο για την δημιουργία της τοπολογίας του δικτύου σε μια SDN αρχιτεκτονική.

2.4 Πλεονεκτήματα

2.4.1 Προγραμματιζόμενα Δίκτυα

Με το SDN, είναι απλούστερη και λιγότερο επιρρεπής σε σφάλματα η αλλαγή των πολιτικών δικτύου, αφού απλά πρέπει να αλλάξουμε μια πολιτική υψηλού επιπέδου και όχι πολλαπλούς κανόνες σε όλο τον εξοπλισμό του δικτύου.

Επιπλέον, η αντίδραση σε μια τροποποίηση δικτύου (μια νέα διασύνδεση ή μια νέα συσκευή για παράδειγμα) ή ένα συμβάν δικτύου (όπως μια επίθεση ή μια υποψία εισβολής) είναι εύκολη και αποτελεσματική αφού μπορούμε εύκολα να αλλάξουμε ολόκληρο το δίκτυο για να διαχειριστούμε το συμβάν. Συν τοις άλλοις, αυτό μπορεί να γίνει όσο το δυνατόν πιο κοντά στην πηγή και να αποφευχθεί η δημιουργία άσκοπου φόρτου στο εσωτερικό μας δίκτυο. Πριν από το SDN, θα έπρεπε να αλλάξουμε με μη αυτόματο τρόπο κανόνες που θα μπορούσαν να οδηγήσουν σε σφάλματα ή αργούς χρόνους αντίδρασης.

Τέλος, η συγκέντρωση της λογικής σε ένα τέτοιο πλήρως προσαρμοσμένο ελεγκτή με καθολική γνώση και υψηλή υπολογιστική ισχύ απλοποιεί την ανάπτυξη πιο εξελιγμένων λειτουργιών.

Αυτή η δυνατότητα προγραμματισμού του δικτύου είναι το βασικό στοιχείο του SDN.

2.4.2 Ευελιξία

Το SDN προσφέρει επίσης υψηλή ευελιξία στη διαχείριση δικτύου. Καθίσταται εύκολη η επαναδρομολόγηση της κυκλοφορίας, η επιθεώρηση συγκεκριμένων ροών, η δοκιμή νέων πολιτικών ή η ανεύρεση απροσδόκητων ροών, για παράδειγμα, επειδή ο ελεγκτής θα προσαρμόσει τους κανόνες χαμηλού επιπέδου σε συγκεκριμένες απαιτήσεις. Έτσι, το SDN είναι ένα εξαιρετικό εργαλείο για τον έλεγχο του νέου αλγορίθμου δρομολόγησης ή του πρωτοκόλλου, και για την απλοποίηση της ασφάλειας του δικτύου καθώς και της ανίχνευσης κακόβουλων προγραμμάτων.

2.4.3 Ενοποιημένες Πολιτικές

Με τον ελεγκτή του, το SDN εγγυάται επίσης μια ενιαία και επικαιροποιημένη πολιτική για το δίκτυο: δεδομένου ότι ο ελεγκτής είναι υπεύθυνος για την προσθήκη κανόνων στα switches, δεν υπάρχει κίνδυνος ένας διαχειριστής δικτύου να ξεχάσει έναν μεταγωγέα ή να εγκαταστήσει ασυνάρτητους κανόνες μεταξύ συσκευών. Πράγματι, ο χειριστής θα καθορίσει απλώς έναν νέο κανόνα και ο ελεγκτής θα αναλάβει να προσαρμόσει και να διαμορφώσει την αποστολή συνεκτικών κανόνων σε κάθε σχετική συσκευή. Επιπλέον, εάν μετακινηθεί μια συσκευή, οι παλιοί κανόνες μπορούν να καταργηθούν και οι νέοι να υπολογιστούν και να προστεθούν δυναμικά. Έτσι, είναι εύκολο να αναπτυχθεί μια ενοποιημένη πολιτική αλλά και να προστεθεί έξυπνη αυτοματοποίηση στο δίκτυο.

2.4.4 Δρομολόγηση

Το SDN μπορεί επίσης να χρησιμοποιηθεί για τη διαχείριση των πληροφοριών δρομολόγησης με κεντρικοποιημένο τρόπο [28], [35], [36] μεταβιβάζοντας τη δρομολόγηση και χρησιμοποιώντας μια διεπαφή για τον ελεγκτή. Ορισμένες λύσεις βρίσκονται υπό μελέτη: το RouteFlow [37], [38] ή η ομάδα εργασίας της IETF για το i2rs [39] για παράδειγμα.

Η διαχείριση της δρομολόγησης μέσω του SDN μπορεί να χρησιμοποιηθεί για τη καλύτερη διαχείριση της χρήσης των συνδέσμων και, ως εκ τούτου, για τη βελτίωση της απόδοσης αυξάνοντας τη μέση χρήση κάθε συνδέσμου χωρίς τον πλήρη κορεσμό του, καθώς αυτό θα ήταν αντιπαραγωγικό.

2.4.5 Διαχείριση Cloud

Το SDN επιτρέπει επίσης την εύκολη διαχείριση μιας πλατφόρμας cloud. Πράγματι, σε ένα τέτοιο περιβάλλον, ο διαχωρισμός του επιπέδου ελέγχου, με το οποίο οι πελάτες και τα συστήματα αλληλεπιδρούν μέσω του API, και του επιπέδου προώθησης είναι απαραίτητο. Η δυναμική που έχει το SDN αντιμετωπίζει προβλήματα που υπάρχουν στο cloud όπως η επέκταση, η προσαρμογή, οι μετατροπές ή μετακινήσεις εικονικών μηχανών και η απομόνωση [40]. Για παράδειγμα, το OpenStack Neutron [41] ήδη παρέχει «δικτύωση ως υπηρεσία» (networking as a service), και σχεδιάζεται να ενσωματωθεί με έναν ελεγκτή SDN, OpenDaylight [42], ώστε να δουλέψει σε ένα SDN περιβάλλον. Το Neutron θα χρησιμοποιήσει το Northbound API του OpenDaylight για τη διαχείριση του δικτύου.

Μια άλλη εργασία, το FlowN [43] παρέχει ένα πλαίσιο για την απομόνωση των χρηστών/ενοικιαστών, παρέχοντας σε κάθε έναν τη δυνατότητα προγραμματισμού του δικτύου: κάθε ενοικιαστής θα μπορεί να τρέχει τον ελεγκτή του -επιτρέποντας στον χρήστη να τον προσαρμόσει εάν χρειαστεί, ή απλά να χρησιμοποιήσει τον προεπιλεγμένο ελεγκτή ο οποίος θα παρέχει συνολική σύνδεση (παρόμοια με την Amazon EC2)- και να διαχειρίζεται το δίκτυό του. Για να γίνει αυτό, το FlowN παρακολουθεί κάθε αίτημα δικτύου και το διανέμει στον ελεγκτή του σωστού ενοικιαστή με βάση τη θύρα εισόδου του πακέτου.

Για να εξασφαλιστεί η απομόνωση δικτύου για χιλιάδες ενοικιαστές και διακομιστές, υπάρχουν λύσεις όπως οι:

- NVGRE (Microsoft: μεταφορά μέσω tunneling πακέτων L2 πάνω από L3),
- VXLAN [44] (μια τεχνολογία της Cisco που υποστηρίζεται από το Open vSwitch: ενσωμάτωση Ethernet frames σε πακέτα UDP),
- Nicira NVP (L2 μέσω IP διαχειριζόμενα από ένα επίπεδο ελέγχου)
- ή το Amazon EC2 (IP πάνω από IP διαχειριζόμενα από ένα επίπεδο ελέγχου).

Τα παραπάνω αντιμετωπίζουν αυτό το πρόβλημα, αλλά μια κοινή παραδοχή είναι ότι δεν μπορούμε να πετύχουμε επέκταση χωρίς επίπεδο ελέγχου [45].

2.4.6 Απλοποίηση Υλικού

Το SDN τείνει να χρησιμοποιεί βασικές τεχνολογίες για τον έλεγχο του εξοπλισμού δικτύου, ενώ η υπολογιστική ισχύς απαιτείται μόνο στον ελεγκτή. Έτσι, οι μεταγωγείς/δρομολογητές δεν χρειάζεται να παρέχουν πολλές δυνατότητες. Πρέπει απλά να επιβάλλουν άμεσα τις εντολές που καθορίζει ο ελεγκτής.

Αυτές οι λειτουργίες μπορούν να εκτελεστούν από εξειδικευμένο εξοπλισμό που προσφέρει μόνο μια διασύνδεση ανεξάρτητη από τον κατασκευαστή. Δεδομένου ότι όλοι οι μεγαγωγείς θα έχουν τους ίδιους σκοπούς, θα υπάρξει λιγότερη διάκριση ανάμεσα στο υλικό, μιας και αυτό θα προσφέρει λιγότερα εξειδικευμένα χαρακτηριστικά (τα οποία θα παρέχονται από τον ελεγκτή). Έτσι, ο εξοπλισμός δικτύων θα προσφέρει χαμηλού επιπέδου διασυνδέσεις, και κατ' επέκταση θα χαμηλώσει το κόστος του [46]. Με ένα τέτοιο υλικό, θα ήταν επίσης εύκολο να προστεθούν νέες συσκευές -δεδομένου ότι δεν είναι εξειδικευμένες-, να συνδεθούν στο δίκτυο και να αναλάβει ο ελεγκτής να τις διαχειριστεί σύμφωνα με την καθορισμένη πολιτική. Έτσι, το δίκτυο θα μπορεί να επεκταθεί εύκολα, μόλις και ο ελεγκτής γίνει επεκτάσιμος.

2.4.7 Υβριδική Ανάπτυξη (Hybrid Deployment)

Το SDN είναι πλέον μια ώριμη τεχνολογία που χρησιμοποιείται σε πραγματικά δίκτυα [46] με μεγάλη επιτυχία και σημαντικά κέρδη απόδοσης.

Δεδομένου ότι το SDN απαιτεί την πλήρη χρήση μεταγωγέων SDN, δεν μπορεί να εφαρμοστεί εύκολα σε όλα τα δίκτυα επιχειρήσεων, τα οποία εξακολουθούν να έχουν κλασικούς μεταγωγείς και δεν μπορούν να αναβαθμίσουν ταυτόχρονα όλο τον εξοπλισμού τους. Ωστόσο, είναι δυνατή μια μεταβατική ανάπτυξη [47] η οποία επωφελείται από τις δυνατότητες SDN.

Σε αντίθεση με τα τυπικά μοτίβα ανάπτυξης -το SDN τοποθετείται παράλληλα με το υπάρχον δίκτυο ή το δίκτυο χωρίζεται σε κομμάτια που θα μοιράζονται μεταξύ του κλασικού δικτύου και των νέων δικτύων SDN [48]. Το Panopticon [47] παρέχει έναν τρόπο για την ανάπτυξη του SDN κάνοντας τις δύο τεχνολογίες να συνυπάρχουν: Εγκαθιστά μεταγωγείς SDN στις διαδρομές δεδομένων και ακόμη και αν ένα πακέτο δεν μπορεί να χρησιμοποιηθεί από ένα μεταγωγέα SDN, θα μεταδοθεί μέσα από αυτόν.

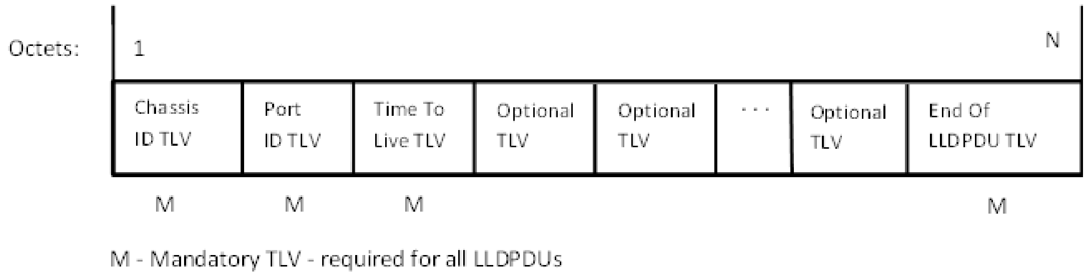
Τα μειονεκτήματα εδώ είναι ότι είναι δύσκολο να φιλτράρουμε ένα πακέτο ανάλογα με την προέλευσή του, δεν μπορούμε να δράσουμε στην πηγή και δεν μπορούμε να εφαρμόσουμε καθολικές πολιτικές.

3. Link Layer Discovery Protocol

Το πρωτόκολλο LLDP ως standard ορίστηκε τον Μάιο του 2005 με την ονομασία IEEE 802.1AB-2005 [49]. Είναι ένα πρωτόκολλο L2 ανεξαρτήτου κατασκευαστή, το οποίο μπορεί να χρησιμοποιηθεί από έναν σταθμό προσαρτημένο σε ένα LAN για να διαφημίσει (advertise) τις δυνατότητές (capabilities) & την ταυτότητά του (identity), αλλά και να λάβει αντίστοιχες πληροφορίες από τους γειτονικούς του σταθμούς.

3.1 Δομή Frame

Οι πληροφορίες που μας δίνει το συγκεκριμένο πρωτόκολλο ανταλλάσσονται μεταξύ των συσκευών χρησιμοποιώντας την πλαισίωση Ethernet. Κάθε ένα από αυτά τα πλαίσια (Εικόνα 12) περιέχει το LLDPU (LLDP Data Unit), το οποίο αποτελεί μια ακολουθία τιμών ακολουθώντας την κωδικοποίηση TLV (Type Length Value).



Εικόνα 12: LLDP Frame

Είναι ουσιαστικά ένας τρόπος αναπαράστασης προαιρετικών πληροφοριών που τον χρησιμοποιούμε όταν θέλουμε να μεταδοθούν δεδομένα μέσα σε ένα επικοινωνιακό κανάλι μέσω κάποιου πρωτοκόλλου. Η αναπαράσταση δεδομένων με αυτήν την μορφή έχει κάποια πλεονεκτήματα όπως:

- Τυχαία σειρά των στοιχείων μέσα στο σώμα του μηνύματος.
- Ευκολία δημιουργίας xml κώδικα για περιγραφή των δεδομένων σε human-readable μορφή.
- Ευκολία αναζήτησης και χρήσης των πληροφοριών (λόγω του Binary format) από εργαλεία ανάλυσης.

3.2 Δομή TLV

Η δομή του κάθε TLV αποτελείται από τα εξής δομικά στοιχεία:

- **Type:** 7bits (Ένας δυαδικός κωδικός ο οποίος υποδεικνύει τον τύπο του μηνύματος που ακολουθεί)
- **Length:** 9 bits (Το μήκος του πεδίου τιμών – συνήθως σε bytes)
- **Value:** 0-511octets (Μεταβλητού μήκους σειριακά δεδομένα από byte τα οποία περιέχουν ουσιαστικά τα δεδομένα του μέρους του μηνύματος που αποστέλλεται)

Ένα απλό παράδειγμα μιας τέτοιας κωδικοποίησης μπορεί να είναι το παρακάτω:

command_c/4/makeCall_c/phoneNumberToCall_c/8/"722-4246"

Σε αυτό το παράδειγμα οι λέξεις ***command_c, makeCall_c***, ***phoneNumberToCall_c*** αναπαριστούν τον τύπο (type), και τα 4, 8 το μήκος των τιμών (value) αντίστοιχα.

Όπως μπορούμε να παρατηρήσουμε και στην Εικόνα 13 που περιγράφει την δομή του πλαισίου, το αναγνωριστικό του LLDP στην κεφαλίδα είναι 0x88cc. Για την αποστολή των πακέτων χρησιμοποιείται μια ειδική διεύθυνση πολλαπλής διανομής (multicast address) όπου δεν προωθείται από συσκευές οι οποίες είναι συμβατές με το πρότυπο 802.1D, αποφεύγοντας έτσι την δημιουργία κλειστών βρόχων. Κάθε πλαίσιο LLDP λοιπόν ξεκινάει με κάποια υποχρεωτικά TLV που περιγράφουν στοιχεία όπως τα:

- Chassis ID
- Port ID
- Time-to-Live

Μετά από αυτά ακολουθούν προαιρετικά πεδία και στο τέλος του πλαισίου ένα ειδικό TLV που ονομάζεται *endofLLDPDU* με τα στοιχεία type & length να είναι 0 και ουσιαστικά δηλώνει το τέλος αυτού.

LLDP Ethernet frame structure

Preamble	Destination MAC	Source MAC	Ethertype	Chassis ID TLV	Port ID TLV	Time to Live TLV	Optional TLVs	End of LLDPDU TLV	Frame check sequence
	01:80:c2:00:00:0e, or 01:80:c2:00:00:03, or 01:80:c2:00:00:00	Station's address	0x88CC	Type=1	Type=2	Type=3	Zero or more complete TLVs	Type=0, Length=0	

Εικόνα 13: Δομή LLDP frame

3.3 Πληροφορίες που μπορούμε να συλλέξουμε

Με το LLDP μπορούν να συλλεχθούν κάποιες πολύ σημαντικές και χρήσιμες πληροφορίες που είναι αποθηκευμένες στην συσκευή προς διαχείριση. Σε μια τέτοια βάση στηρίχτηκε και η εφαρμογή που χρησιμοποιήθηκε στην παρούσα εργασία για την κατασκευή της τοπολογίας του δικτύου. Αναλυτικότερα η υλοποίηση θα παρουσιαστεί σε επόμενο κεφάλαιο. Ειδικότερα τώρα, όσον αφορά τις πληροφορίες που μπορούν να συλλεχθούν είναι οι εξής:

- Όνομα και περιγραφή συστήματος
- Όνομα και περιγραφή πόρτας
- Όνομα εικονικού τοπικού δικτύου VLAN
- Διεύθυνση IP για την διαχείριση του συστήματος
- Ιδιότητες/Δυνατότητες συστήματος, όπως δρομολόγηση, προώθηση κτλ.
- Όνομα φυσικής διεύθυνσης – MAC address
- Λειτουργία PoE (Power over Ethernet)
- Link aggregation (μέθοδος που επιτρέπει τον συνδυασμό πολλαπλών δικτυακών συνδέσεων παράλληλα ώστε να αυξηθεί η δυνατότητα διεκπεραίωσης κίνησης ή throughput)

Μπορεί ακόμα να χρησιμοποιηθεί ένα πρόσθετο που δημιουργήθηκε και ενσωματώθηκε το 2006, το οποίο ονομάζεται Media Endpoint Discovery (LLDP-MDED) και επιτρέπει:

- Αυτόματη αναγνώριση των κανόνων που διέπουν ένα τοπικό δικτύου, όπως προτεραιότητες VLAN, διαφοροποιημένων υπηρεσιών κλπ.
- Ανακάλυψη συσκευών για την δημιουργία τοπικών βάσεων δεδομένων σε περιπτώσεις VoIP κτλ.
- Αυτόματη διαχείριση ενέργειας σε περιπτώσεις PoE
- Διαχείριση των διαθέσιμων συσκευών που επιτρέπει τον εντοπισμό των συστημάτων και των καθορισμό των χαρακτηριστικών τους. (κατασκευαστής, έκδοση λογισμικού και υλικού κ.α)

3.4 Εφαρμογές του LLDP

Το LLDP είναι ένα one-hop πρωτόκολλο. Αυτό σημαίνει ότι τα πακέτα που στέλνει μεταξύ των δικτυακών συσκευών μπορούν να φτάσουν μόνο μέχρι τον διπλανό τους γείτονα, δηλαδή το μηχάνημα που απέχει μόνο ένα hop απόσταση στην τοπολογία ή απλά είναι directly connected. Αν εμείς λοιπόν έχουμε μια συσκευή (switch) η οποία είναι άμεσα συνδεδεμένη με 3 άλλες συσκευές (pc, switch κλπ) και θελήσουμε να δούμε τις συνδέσεις τοπικά από την πρώτη συσκευή χρησιμοποιώντας το LLDP, θα δούμε όλες τις διαθέσιμες πληροφορίες και από τις 3 συνδεδεμένες συσκευές. Θέλοντας λοιπόν για τις ανάγκες τις διπλωματικής να μπορούμε να κατασκευάζουμε μια τοπολογία δικτύου, χρειαστήκαμε έναν τρόπο να συλλέξουμε πληροφορίες από τις διαθέσιμες συσκευές οι οποίες όμως να μας επιτρέπουν και την δημιουργία του «χάρτη» του δικτύου μας.

Προσεγγίζοντας το πρόβλημα με μια SDN λογική όπου η ευφυΐα του δικτύου είναι συγκεντρωμένη στον controller μπορούμε με διαδοχικά query από αυτόν προς τις ελεγχόμενες συσκευές να μάθουμε τους γείτονες κάθε συσκευής και ύστερα να κατασκευάσουμε τον χάρτη του δικτύου κάνοντας τους απαραίτητους συσχετισμούς των διευθύνσεων και πορτών. Για να επιτύχουμε κάτι τέτοιο χρειαζόμαστε από κάθε συσκευή το ζεύγος πληροφοριών mac-address και connected port, οι οποίες μπορούν να προσδιοριστούν από το LLDP. Συλλέγοντας τελικά αυτές τις πληροφορίες τοπικά στον controller από κάθε forwardIng element είμαστε σε θέση να συμπεραίνουμε την τοπολογία δικτύου.

Το επικρατέστερο πρωτόκολλο όσον αφορά το SDN αλλά και που έχει χρησιμοποιηθεί έως σήμερα επί των πλείστων σε οργανισμούς είναι το OpenFlow. Για τον λόγο αυτό είναι θεμιτό να αναλύσουμε τον τρόπο που το OpenFlow λειτουργεί και εκτελεί τις διαδικασίες ανακάλυψης τοπολογίας.

4. Το Πρωτόκολλο OpenFlow

Το OpenFlow είναι το πρώτο προτυποποιημένο πρωτόκολλο για την επικοινωνία του επιπέδου ελέγχου με το επίπεδο δεδομένων σε SDN αρχιτεκτονικές (Southbound API). Επιτρέπει την άμεση πρόσβαση και διαχείριση της πλατφόρμας προώθησης των δεδομένων των δικτυακών συσκευών, δηλαδή των εικονικών και φυσικών μεταγωγέων και δρομολογητών. Το OpenFlow είναι απαραίτητο για τη μετακίνηση του ελέγχου του δικτύου από τις δικτυακές συσκευές σε ένα λογικά κατανεμημένο λογισμικό ελέγχου και εφαρμόζεται και στις δυο πλευρές της διεπαφής μεταξύ του λογισμικού ελέγχου του SDN και της δικτυακής υποδομής.

Το OpenFlow χρησιμοποιεί την έννοια της ροής (flow) για να αναγνωρίσει τη δικτυακή κίνηση και η οποία βασίζεται σε προκαθορισμένους κανόνες αντιστοίχισης που μπορούν στατικά ή δυναμικά να οριστούν από τον ελεγκτή. Επίσης επιτρέπει στον διαχειριστή να ορίσει πως θα κατανεμηθεί η κίνηση στις δικτυακές συσκευές ανάλογα τους πόρους του δικτύου, τις ανάγκες των εφαρμογών και τα πρότυπα της συνηθισμένης κίνησης στο δίκτυο. Ενώ στα παραδοσιακά IP δίκτυα η δρομολόγηση στο διαδίκτυο γίνεται βάσει των διευθύνσεων IP και δύο ροές με τα ίδια εναρκτήρια και τερματικά σημεία θα ακολουθήσουν την ίδια διαδρομή λόγω των ίδιων διευθύνσεων IP που έχουν, στην αρχιτεκτονική SDN δεν συμβαίνει το ίδιο. Πλέον δύο ροές μπορεί να έχουν ίδια σημεία πηγής και προορισμού αλλά να ακολουθούν άλλη διαδρομή και με διαφορετική προτεραιότητα διότι έχουν διαφορετικές απαιτήσεις, οπότε ο πάροχος να επιθυμεί διαφορετική πολιτική δρομολόγησης για την καθεμία.

Δίκτυα αρχιτεκτονικής SDN που υποστηρίζουν OpenFlow μπορούν να αναπτυχθούν σε ήδη υπάρχοντα δίκτυα. Οι συσκευές δικτύωσης μπορούν να υποστηρίξουν προώθηση βάσει του OpenFlow όσο και βάσει της συμβατικής δρομολόγησης TCP/IP. Υπάρχει η δυνατότητα υβριδικών μεταγωγέων (όπως αναφέρθηκε στο προηγούμενο κεφάλαιο) που λειτουργούν σε κάθε περιβάλλον (SDN ή μη) ή εφαρμόζουν κεντρικοποιημένο έλεγχο στις εφαρμογές (παρακολούθηση και διαχείριση του δικτύου), οπότε δεν απαιτείται να αντικατασταθεί κάθε δικτυακή συσκευή με SDN-μεταγωγείς [50]. Αυτό είναι μεγάλο πλεονέκτημα για τους παρόχους που μπορούν να οδηγηθούν στην αρχιτεκτονική SDN σταδιακά, βασιζόμενοι στα παραδοσιακά δίκτυα τους μετατρέποντάς τα σε SDN.

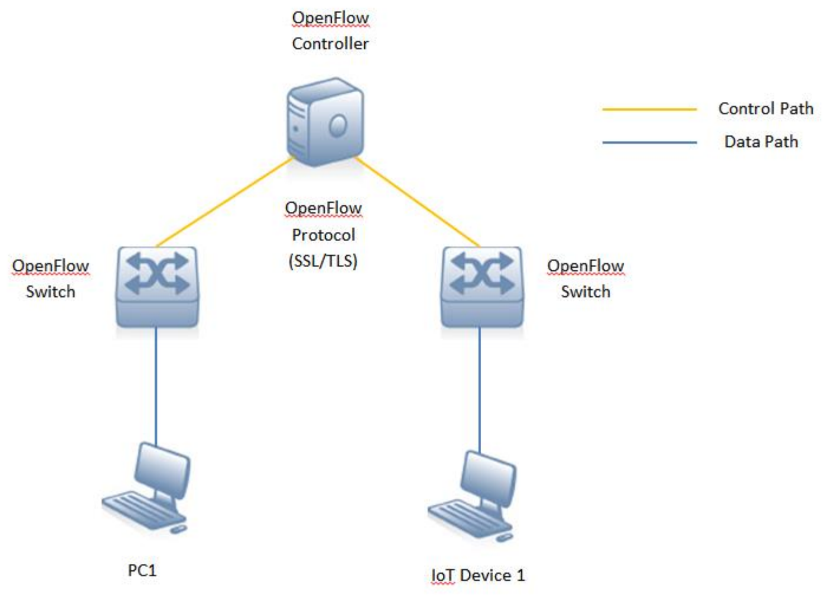
4.1 Αρχιτεκτονική OpenFlow

Η δικτυακή αρχιτεκτονική του OpenFlow αποτελείται από τρεις βασικές έννοιες:

1. OpenFlow-συμβατούς μεταγωγείς που συνθέτουν το επίπεδο δεδομένων.
2. Το επίπεδο ελέγχου αποτελείται από έναν ή περισσότερους OpenFlow ελεγκτές.
3. Ένα ασφαλές κανάλι ελέγχου συνδέει τους μεταγωγείς με το επίπεδο ελέγχου.

4.1.1 Μεταγωγέας OpenFlow (OpenFlow Switch)

Οι μεταγωγείς επικοινωνούν μεταξύ τους αλλά και με τους τελικούς χρήστες (hosts) χρησιμοποιώντας το λογισμικό διαδρομής δεδομένων (data path) που παρέχεται και ο ελεγκτής επικοινωνεί με τους μεταγωγείς χρησιμοποιώντας την διαδρομή ελέγχου όπως φαίνεται παρακάτω, Εικόνα 14.



Εικόνα 14: Αρχιτεκτονική OpenFlow

Ο μεταγωγέας OpenFlow αποτελείται από πίνακες ροής, οι οποίοι χρησιμοποιούνται για την αναζήτηση και προώθηση πακέτων. Μια εικονική υλοποίηση δρομολογητή είναι ο OpenVSwitch (OVS).

Η σύνδεση μεταξύ του ελεγκτή OpenFlow και του μεταγωγέα εξασφαλίζεται μέσω ενός Ασφαλούς Καναλιού (Secure Channel) χρησιμοποιώντας κρυπτογραφικά πρωτόκολλα SSL ή TLS, όπου ο μεταγωγέας και ο ελεγκτής επικυρώνονται αμοιβαία με την ανταλλαγή πιστοποιητικών που υπογράφονται με ιδιωτικό κλειδί και από τις δύο πλευρές. Αν και αυτός είναι ένας πολύ ισχυρός αλγόριθμος ασφάλειας, ο ελεγκτής μπορεί να είναι τρωτός σε επιθέσεις άρνησης υπηρεσιών (DoS), ή σε επίθεση Man in the middle, επομένως πρέπει να εφαρμοστούν κατάλληλες πρακτικές ασφάλειας για να αποτραπούν τέτοιου είδους επιθέσεις.

4.1.2 Πίνακας Ροής (Flow Table)

Εδώ μπορούμε να δούμε μια απλή και γρήγορη επισκόπηση των δυνατοτήτων του OpenFlow. Λεπτομερέστερες πληροφορίες σχετικά με τα χαρακτηριστικά που παρέχονται από τις διάφορες εκδόσεις OpenFlow μπορούν να βρεθούν στην προδιαγραφή του πρωτοκόλλου κάθε έκδοσης (v.1.5 [51]).

Μια καταχώρηση στον πίνακα ροής αποτελείται από πολλαπλά πεδία:

- Τα πεδία αντιστοίχισης (match fields): η θύρα εισόδου και μια κεφαλίδα πακέτων που περιγράφουν το πακέτο: μπορεί να χρησιμοποιεί πληροφορίες από τα επίπεδα OSI 2 έως 4 (Ethernet, IP και TCP / UDP) καθώς και wildcards, αν χρειαστεί, ώστε να παρέχει προσαρμοστικότητα [52]. Τα διαθέσιμα πεδία εξαρτώνται από την έκδοση του OpenFlow και περιγράφονται λεπτομερώς στην προδιαγραφή της κάθε έκδοσης (v.1.5 [51]).
- Η προτεραιότητα ροής (flow priority): το ταίριασμα με την υψηλότερη προτεραιότητα θα χρησιμοποιηθεί για να προσδιοριστεί η ενέργεια.
- Μετρητές (counters) που θα αποθηκεύουν στατιστικά στοιχεία σχετικά με τη ροή (πόσα πακέτα/bytes αντιστοιχίστηκαν ήδη με αυτήν τη ροή). Μπορούν να χρησιμοποιηθούν για να ανιχνεύσουν, μεταξύ άλλων, άχρηστες ροές ή ανώμαλη κίνηση. Υπάρχουν επίσης μετρητές που σχετίζονται με τις φυσικές πόρτες, τον πίνακα και την ουρά.
- Τα χρονικά όρια (timeouts) της ροής που καθορίζουν πότε αυτή αφαιρείται από τον πίνακα. Υπάρχουν δύο τύποι:

    - το χρονικό όριο αναμονής (idle timeout), το οποίο ορίζει το αποδεκτό χρονικό διάστημα αδράνειας της ροής (μετά από αυτή την περίοδο, η ροή θα αφαιρεθεί από τον πίνακα)
    - το συνολικό χρονικό όριο (hard timeout), το οποίο καθορίζει τη μέγιστη διάρκεια ζωής της ροής, ανεξαρτήτως εάν εξακολουθεί να χρησιμοποιείται ή όχι.
- Η ενέργεια (action):
    - Προώθηση ενός πακέτου σε μια δεδομένη πόρτα - ή αντιγραφή του και προώθηση σε πολλές πόρτες - ή πλημμύρησε το δίκτυο.
    - Ενσωμάτωση του πακέτου σε ένα πακέτο OpenFlow, και προώθηση αυτού στον ελεγκτή για περαιτέρω ανάλυση. Πιθανόν να προστεθεί στον μεταγωγέα μια αντίστοιχη ροή από τον ελεγκτή για να μπορεί να χειριστεί τα επόμενα παρόμοια πακέτα.
    - Απέρριψε το πακέτο.
    - Πρόσθεσε το πακέτο στην ουρά. Χρησιμοποιείται στις περιπτώσεις βασικού QoS.
    - Τροποποίηση των πεδίων του πακέτου. Μπορεί, για παράδειγμα, να χρησιμοποιηθεί για την αντικατάσταση των VLAN ID, MAC και IP διευθύνσεων.
- Ένα cookie (Αδιαφανές αναγνωριστικό που εκδίδεται από τον ελεγκτή)

4.2 Ελεγκτές OpenFlow (Controllers)

Ο ελεγκτής είναι ο πυρήνας και το κύριο μέρος του λειτουργικού συστήματος του δικτύου (NOS) στο SDN. Είναι υπεύθυνος για το χειρισμό του πίνακα-ροών του μεταγωγέα, καθώς και για την επικοινωνία μεταξύ των εφαρμογών και των συσκευών του δικτύου χρησιμοποιώντας το πρωτόκολλο OpenFlow. Ακόμα, στέλνει στις συσκευές προώθησης καταχωρήσεις ροής (flow entries), με βάση τις οποίες γίνεται η δρομολόγηση και η προώθηση δεδομένων. Οι ροές δεδομένων δημιουργούνται δηλαδή ανάλογα με την ζήτηση την κάθε χρονική στιγμή, ενώ ο ελεγκτής προσφέρει δυναμική ανάθεση πόρων. Σε ένα δίκτυο, μπορεί να υπάρχουν ένας ή περισσότεροι ελεγκτές. Οι ελεγκτές μπορούν να ταξινομηθούν σε δύο κύριες κατηγορίες:

1. Ανοικτού κώδικα, μεμονωμένοι ελεγκτές.

2. Εμπορικοί, κλειστού κώδικα, κατανεμημένοι ελεγκτές.

Οι ελεγκτές ανοικτού κώδικα είναι διαθέσιμοι για την έρευνα και την ανάπτυξη, αναπαριστώνται σαν μεμονωμένοι ελεγκτές με τη δυνατότητα ανάπτυξης διάφορων APIs για την υλοποίηση συγκεκριμένων διεργασιών. Υπάρχουν πολλοί Open source OpenFlow ελεγκτές, με κύρια μεταξύ τους διαφορά τη γλώσσα προγραμματισμού που είναι γραμμένοι. Παρακάτω είναι μια λίστα για τους ελεγκτές ανοικτού κώδικα βασισμένη στη γλώσσα προγραμματισμού τους [53]:

- C++: NOX (επίσης σε Python)
- Java: Beacon, Floodlight και OpenDaylight [42]
- Python: POX και RYU.

Οι κατανεμημένοι ελεγκτές είναι σε θέση να λειτουργούν και να ελέγχουν το δίκτυο μέσω πολλαπλών ελεγκτών. Με την υλοποίηση αυτή, τα οφέλη που έχουν είναι επιπλέον αφαίρεση των επιπέδων στο επίπεδο ελέγχου και στην ανοχή σε σφάλματα. Μερικοί τέτοιοι ελεγκτές είναι: Onix [54], από Nicira Networks, IRIS [31], από την ερευνητική ομάδα του ETRI, Big Network Controller από Big Switch Networks και Programmable Flow από την NEC. Οι ελεγκτές Onix και IRIS έχουν την πρόσθετη δυνατότητα επέκτασης απόδοσης με την προσθήκη επιπλέον ελεγκτών μέσα σε ένα Cluster ελεγκτών (Controller Cluster).

4.2.1 Ελεγκτής NOX

Η ανάπτυξή του ξεκίνησε στην Nicira, και είναι ο πρώτος Openflow controller [55], [56]. Είναι ανοιχτού κώδικα και είναι προγραμματισμένος σε Python και C++. Η σημερινή εφαρμογή είναι μόνο σε C++ και προσφέρει μια υλοποίηση ενός API του OpenFlow 1.0. Χρησιμοποιήθηκε από πολλούς ερευνητές, και γι' αυτό ένα μεγάλο μέρος των εφαρμογών που αναφέρονται στα papers, βασίζονται σε αυτόν. Παρέχει ένα φιλικό περιβάλλον προς τον προγραμματιστή, το οποίο περιλαμβάνει έτοιμα υποπρογράμματα (components) για λειτουργίες όπως η δρομολόγηση πακέτων (routing) και η ανίχνευση τοπολογίας δικτύου (topology discovery). Επίσης, είναι εύκολο να τροποποιήσει κανείς τα υποπρογράμματα που παρέχονται, ή να δημιουργήσει νέα. Η χρήση του αυτή την περίοδο βρίσκεται σε πτώση, μιας και η τελευταία του ενημέρωση έγινε κάπου στα μέσα του 2012 [56].

4.2.2 Ελεγκτής POX

Ο ελεγκτής POX είναι η βελτιωμένη εξέλιξη του NOX που είχε γραφτεί σε python, και έπειτα αποσύρθηκε. Εκτός από την υλοποίηση του OpenFlow API σε Python, ο POX προσφέρει έτοιμα components όπως και ο NOX: λειτουργεί σε πολλές πλατφόρμες όπως Linux, Mac OS, Windows και υποστηρίζει το ίδιο γραφικό περιβάλλον χρήστη και τα ίδια εργαλεία οπτικοποίησης όπως ο NOX.

Λόγω της χρήσης της Python, είναι εύκολο να προστεθούν και να αναπτυχθούν νέα χαρακτηριστικά, αλλά η επίδοση του POX -αν και μπορεί να βελτιωθεί με τη χρήση εργαλείων όπως το PyPy- είναι σημαντικά χαμηλότερη [57], [58] από αυτή των άλλων ελεγκτών, και επομένως δεν είναι κατάλληλος για την χρήση σε επιχειρησιακό περιβάλλον.

4.2.3 Ελεγκτής RYU

Ο ελεγκτής RYU είναι μια υλοποίηση OpenFlow σε Python, θεωρείται ευέλικτος για τις συνήθεις λειτουργίες δικτύου. Αποτελείται από συλλογή υποπρογραμμάτων και βιβλιοθηκών όπως Netconf, OF-conf και υποστηρίζει τα πρωτόκολλα OpenFlow 1.0, 1.2, 1.3, 1.4, 1.5 [59].

4.2.4 Ελεγκτής Onix

Ο Onix [60] είναι ένας controller που είναι εγκατεστημένος και λειτουργεί σε ένα σύνολο από φυσικούς server όπου σε κάθε server μπορεί να τρέχει ταυτόχρονα ο controller. Το βασικό του πλεονέκτημα είναι ότι το control plane που διαχειρίζεται το δίκτυο αποτελεί ένα κατανεμημένο σύστημα. Σε σχέση με άλλους controllers, είναι πιο αξιόπιστος και πιο επεκτάσιμος καθώς πολλαπλές του διεργασίες εκτελούνται ταυτόχρονα σε διαφορετικά φυσικά μηχανήματα, σε αντίθεση με άλλους, οι οποίοι προσφέρουν μονάχα ένα κεντρικό σημείο διαχείρισης. Διαφορετικές διεργασίες του εκτελούνται για την διαχείριση του δικτύου και τον έλεγχό του, οι οποίες τα συλλέγουν και τα διαμοιράζουν, και διαφορετικές για την ενημέρωση του εξοπλισμού και την δημιουργία των κατάλληλων πινάκων. Παράλληλα προσφέρει μία πιο φιλική προγραμματιστική διεπαφή ενώ δεν υποστηρίζει μονάχα το OpenFlow ως πρωτόκολλο επικοινωνίας μεταξύ controller και εξοπλισμού.

4.2.5 Ελεγκτής Beacon

Ο ελεγκτής Beacon [61], [58] είναι ένας ελεγκτής OpenFlow σε Java, ο οποίος δημιουργήθηκε το 2010. Είναι ευρέως γνωστός για διδασκαλία, έρευνα, καθώς και η βάση για τον ελεγκτή Floodlight. Μπορεί να λειτουργήσει σε πολλές πλατφόρμες, όπως Windows, Linux, Android OS, υποστηρίζει πολυνηματική (multithreaded) λειτουργία και βασίζεται σε τμηματική λογική (modular), οπότε είναι εύκολα επεκτάσιμος. Ένα από τα μεγάλα του πλεονεκτήματα είναι η δυναμική του φύση. Έχει την δυνατότητα να εκτελεί, σταματάει και να μπορεί να εγκαταστήσει εφαρμογές ακόμη και την στιγμή που ο controller λειτουργεί σε αντίθεση με τους προαναφερόμενους. Εξ' αιτίας της επίδοσής του [58], [62], είναι μια πολύ καλή λύση για χρήση σε πραγματικές συνθήκες.

4.2.6 Ελεγκτής Floodlight

Ο Floodlight [63] είναι βασισμένος στη Java και προήλθε από τον Beacon, αλλά υιοθετεί μια διαφορετική αρχιτεκτονική και τρόπο λειτουργίας [64]. Είναι ανοιχτού κώδικα και έχει άδεια από την Apache, ενώ στην ανάπτυξή του έχει συνεισφέρει σημαντικά η εταιρεία Big Switch Networks. Είναι εύκολο να εγκατασταθεί και επίσης επιτυγχάνει πολύ καλές επιδόσεις. Για την ώρα, αποτελεί τον πιο ώριμο OpenFlow controller. Με όλες τις λειτουργίες του, ο Floodlight είναι περισσότερο μια συνολική λύση, παρά ένας απλός ελεγκτής.

4.2.7 Ελεγκτής Opendaylight

Το Opendaylight [42] είναι ένα σύνολο από ερευνητικά έργα που σχετίζονται με τα SDN δίκτυα και αναπτύσσονται υπό την επίβλεψη του Linux foundation με την συμμετοχή των μεγαλύτερων εταιρειών δικτύων στον κόσμο.

Πρόκειται για ένα πλαίσιο ανοιχτού κώδικα για τη διευκόλυνση της πρόσβασης στο SDN και το NFV (Εικονικοποίηση λειτουργιών δικτύου). Όπως ο Floodlight έτσι και ο Opendaylight μπορεί να θεωρηθεί μια ολοκληρωμένη λύση.

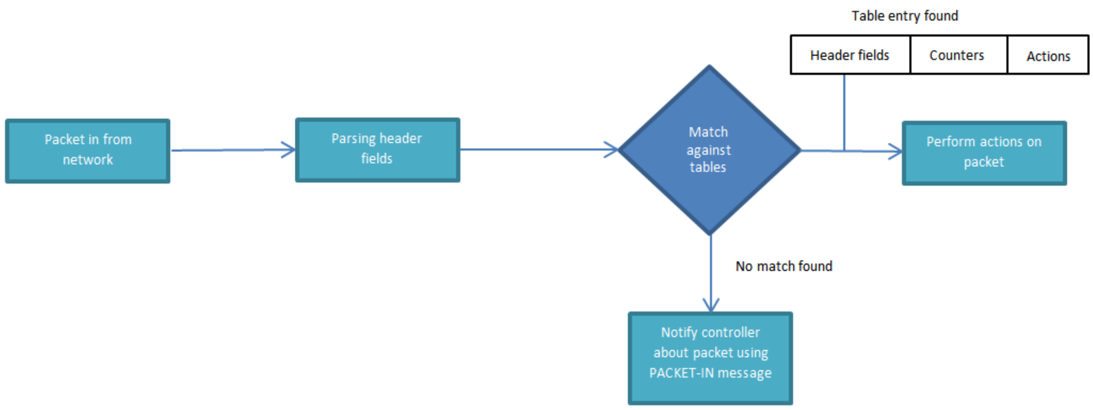
4.5 Επικοινωνίες

Υπάρχουν τρεις κατηγορίες επικοινωνίας στο πρωτόκολλο OpenFlow [65]:

- **Controller-to-Switch:** η επικοινωνία αυτή είναι υπεύθυνη για την ανίχνευση χαρακτηριστικών, ρυθμίσεων, προγραμματισμό του μεταγωγέα και ανάκτηση πληροφοριών (τύποι μηνυμάτων: Features, Configuration, Modify-State, Read-State, Send-Packet, Barrier).
- **Asynchronous:** επικοινωνία με πρωτοβουλία του συμβατού OpenFlow μεταγωγέα χωρίς καμία παρακίνηση από τον ελεγκτή. Χρησιμοποιείται για να ενημερώνει τον ελεγκτή για αφίξεις πακέτων, αλλαγές κατάστασης του μεταγωγέα και λάθη (τύποι μηνυμάτων: Packet-in, Flow-Removed, Portstatus, Error).
- **Symmetric:** επικοινωνία για να δούμε αν το κανάλι ελέγχου είναι ενεργό και διαθέσιμο (τύποι μηνυμάτων: Hello, Echo, Vendor).

4.6 Μηχανισμός Προώθησης Πακέτων

Σε ένα OpenFlow δίκτυο όταν ο μεταγωγέας λαμβάνει ένα πακέτο, αναλύει το πεδίο κεφαλίδας και ελέγχει αν ταιριάζει στους κανόνες του πίνακα-ροής. Αν υπάρχει ένα ταίριασμα, τότε υλοποιείται η ενέργεια από τον πίνακα-ροής. Αν τα πακέτα ταιριάζουν σε περισσότερους από έναν κανόνες, τότε τα πακέτα αντιπαραβάλλονται με μια συγκεκριμένη εγγραφή-ροής βάσει προτεραιοτήτων, δηλαδή επιλέγεται η εγγραφή-ροής με την υψηλότερη προτεραιότητα. Στη συνέχεια, ο μεταγωγέας ενημερώνει τους μετρητές του εν λόγω πίνακα-ροής. Τέλος, ο μεταγωγέας προωθεί το πακέτο σε μια πόρτα εξόδου. Αν το εισερχόμενο πακέτο δεν ταιριάζει με καμία εγγραφή-ροής, ο μεταγωγέας θα προωθήσει το πακέτο στον ελεγκτή για να υπολογίσει την διαδρομή που θα πρέπει να ακολουθηθεί για αυτό και άλλα παρόμοια μελλοντικά πακέτα. Η διαδικασία μηχανισμού της προώθηση των πακέτων απεικονίζεται στο διάγραμμα ροής της Εικόνα 15.

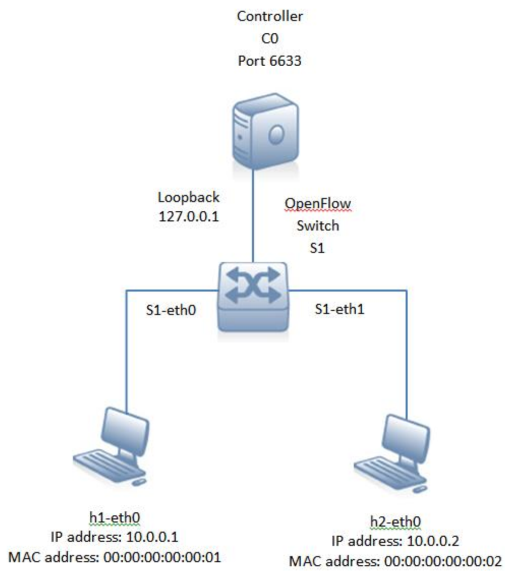


Εικόνα 15: Διάγραμμα ροής προώθησης πακέτων

4.7 Επίδειξη μηνυμάτων που ανταλλάσσονται στο OpenFlow δίκτυο

Για να δείξουμε τα μηνύματα που ανταλλάσσονται σε ένα πραγματικό δίκτυο OpenFlow [67], μπορούμε να χρησιμοποιήσουμε τον δικτυακό εξομοιωτή Mininet [68]. Με αυτόν θα εξομοιώσουμε δύο τελικούς χρήστες (hosts) συνδεδεμένους με ένα μεταγωγέα που συνδέεται σε έναν ελεγκτή, όπως φαίνεται και στην Εικόνα 16.

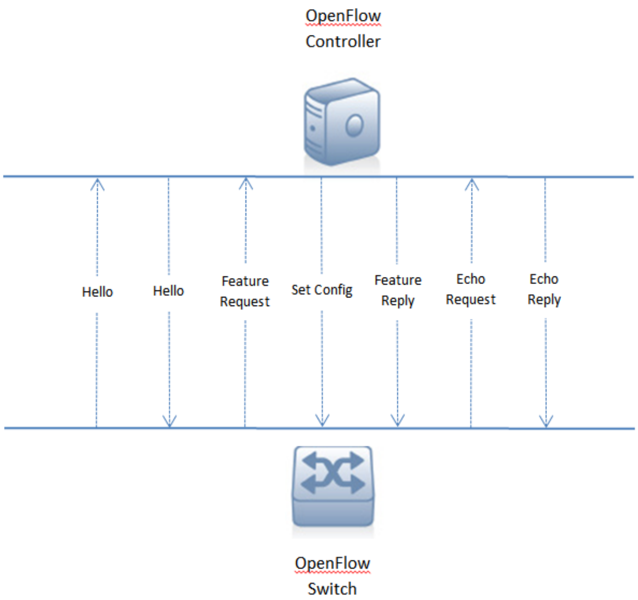
Γι' αυτή την επίδειξη πρέπει να εξηγήσουμε την εγκατάσταση της σύνδεσης μεταξύ Μεταγωγέα - Ελεγκτή, και την επικοινωνία host-to-host μέσω του μεταγωγέα και ελεγκτή OpenFlow.



Εικόνα 16: Δικτυακή τοπολογία από το Mininet

4.7.1 Δημιουργία μηνυμάτων μεταξύ μεταγωγέα και ελεγκτή

Όταν ένας μεταγωγέας συνδέεται σε ένα δίκτυο OpenFlow εγκαθιδρύει μια σύνδεση TCP με τη διεύθυνση IP του ελεγκτή (Loopback interface 127.0.0.1), σε μια προεπιλεγμένη πόρτα με αριθμό 6633. Μετά τη διαδικασία αυτή οι δύο πλευρές αρχίζουν να ανταλλάσσουν μηνύματα «Hello», που περιλαμβάνουν τον μεγαλύτερο αριθμό έκδοσης του OpenFlow που υποστηρίζουν. Ακολουθεί το μήνυμα «Feature request» το οποίο αποστέλλεται από τον ελεγκτή για να μάθει ποιες πόρτες είναι διαθέσιμες στον μεταγωγέα, ο οποίος με τη σειρά του απαντά με μήνυμα «Feature reply» που περιέχει μια λίστα με τις πόρτες, την ταχύτητα των πορτών, και τους υποστηριζόμενους πίνακες και ενέργειες. Το μήνυμα «SET config» αποστέλλεται στη συνέχεια από τον ελεγκτή στον μεταγωγέα για να ρωτήσει αν θα λήξει τη ροή. Τέλος, μηνύματα «echo request» και «echo reply» αποστέλλονται συχνά μεταξύ του ελεγκτή και του μεταγωγέα για να ανταλλάξουν πληροφορίες σχετικά με το εύρος ζώνης, τις καθυστερήσεις και για να κρατάνε «ζωντανή» τη σύνδεσή τους. Η διαδικασία αυτή απεικονίζεται παρακάτω στην Εικόνα 17.

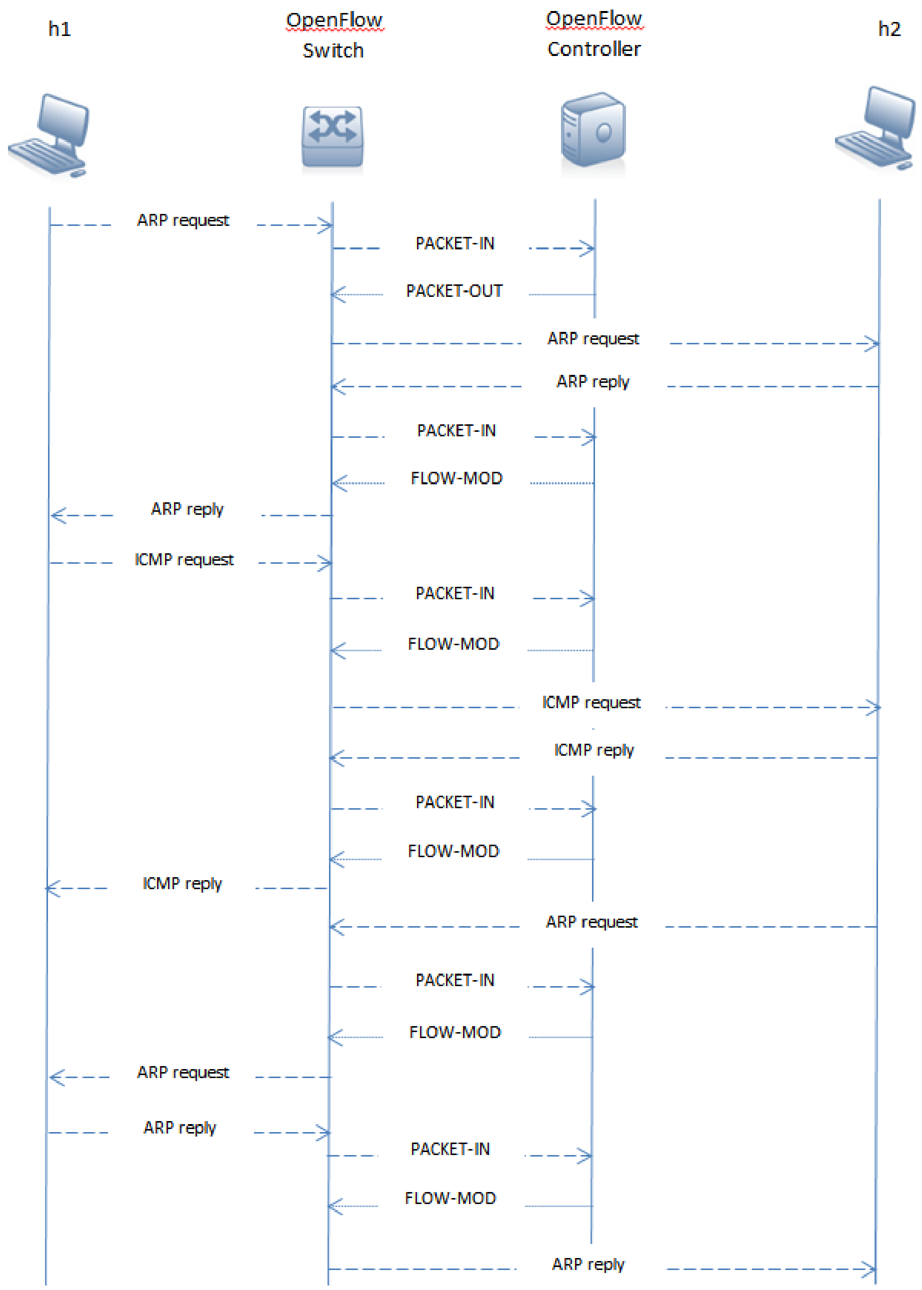


Εικόνα 17: Μηνύματα επικοινωνίας μεταξύ του OpenFlow μεταγωγέα και του OpenFlow ελεγκτή

4.7.2 Μηνύματα που ανταλλάσσονται μεταξύ δύο Hosts

Για να δείξουμε πως γίνεται η σύνδεση host-to-host σε ένα OpenFlow δίκτυο, θα χρησιμοποιήσουμε το εργαλείο του Ping για να στείλουμε ICMP πακέτα από τον host1 (h1) στον host2 (h2) και το αντίστροφο.

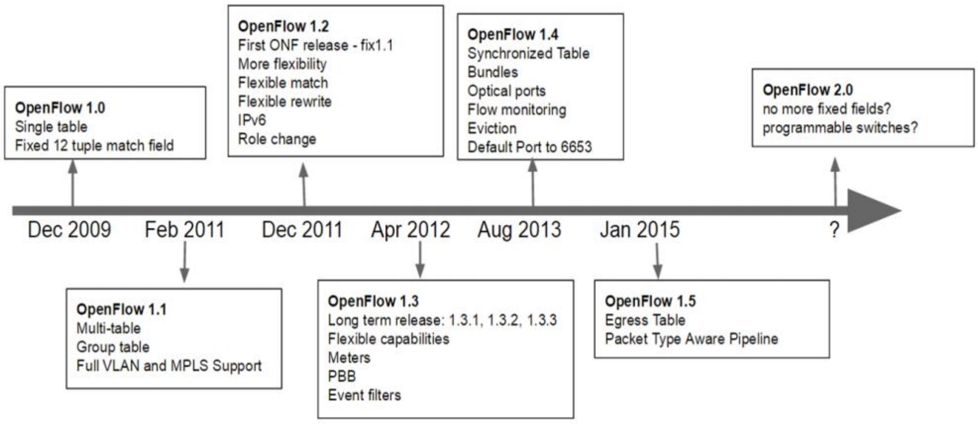
Η διαδικασία ξεκινά όταν ο h1 στέλνει ένα «αίτημα ARP» στον μεταγωγέα, ζητώντας να μάθει τη MAC διεύθυνση του h2. Ο μεταγωγέας δεν γνωρίζει πώς να διαχειριστεί το πακέτο και έτσι στέλνει το πακέτο ως μήνυμα «PACKET-IN» στον ελεγκτή. Ο ελεγκτής απαντά με ένα μήνυμα «PACKET-OUT», που έχει μια ενέργεια η οποία οδηγεί τον μεταγωγέα να στείλει το πακέτο σε όλες τις πόρτες του εκτός από τη πόρτα εισόδου, περιμένοντας απάντηση στο αίτημα του. Όταν ο h2 απαντήσει στο αίτημα, ο μεταγωγέας στέλνει την απάντηση στον ελεγκτή επειδή δεν έχει καμία γνώση για το που θα προωθήσει το πακέτο. Όταν ο ελεγκτής λάβει την «απάντηση ARP», στέλνει μήνυμα «FLOW-MOD» για να εγκαταστήσει ο μεταγωγέας μια νέα εγγραφή-ροής, η οποία θα χρησιμοποιηθεί στο μέλλον για τις «ARP απαντήσεις» από τον h2 και οι οποίες προωθούνται απευθείας από τον μεταγωγέα, χωρίς να κοινοποιούνται στον ελεγκτή. Η ίδια διαδικασία συμβαίνει όταν ο h1 στέλνει «ICMP» αίτηση/απάντηση και όταν ο h2 στέλνει ένα «αίτημα ARP» για να μάθει τη MAC διεύθυνση του h1 έχοντας ως συνέπεια την «απάντηση ARP». Στο τέλος, πέντε νέες εγγραφές-ροής θα εγκατασταθούν στον πίνακα-ροής του μεταγωγέα από τον ελεγκτή OpenFlow, όπως φαίνεται και στην Εικόνα 18.



Εικόνα 18: Διαδικασία Ping μεταξύ h1 και h2

4.8 Προδιαγραφές εκδόσεων OpenFlow

Το OpenFlow κυκλοφόρησε τον Δεκέμβρη του 2009. Από τότε έχουν κυκλοφορήσει άλλες πέντε εκδόσεις, οι οποίες προσέθεταν συνεχώς νέες δυνατότητες και λειτουργίες στο πρωτόκολλο. Στην Εικόνα 19 φαίνεται το roadmap αυτών των αλλαγών. Παρακάτω θα περιγράψουμε συνοπτικά τις αλλαγές της κάθε έκδοσης, σε σχέση με την πρώτη (1.0 [65]).



Εικόνα 19: Roadmap εκδόσεων OpenFlow [69]

4.8.1 OpenFlow 1.1

Το OpenFlow 1.1 [70] κυκλοφόρησε τον Φεβρουάριο του 2011. Περιέχει σημαντικές αλλαγές σε σύγκριση με το OpenFlow 1.0. Για παράδειγμα, η επεξεργασία πακέτων λειτουργεί διαφορετικά. Στην έκδοση 1.1 τα πακέτα διεκπεραιώνονται μέσω αγωγού (pipeline) πολλαπλών πινάκων-ροής. Οι δύο κύριες αλλαγές είναι ένας αγωγός (pipeline) αποτελούμενος από πολλαπλούς πίνακες-ροής και έναν πίνακα-ομάδας (group table).

4.8.2 OpenFlow 1.2

Το OpenFlow 1.2 [71] κυκλοφόρησε τον Δεκέμβριο του 2011. Έρχεται με εκτεταμένη υποστήριξη του πρωτοκόλλου σε σχέση με το IPv6. Το OpenFlow 1.2 μπορεί να ταιριάξει τις IPv6 διευθύνσεις πηγής και προορισμού, αριθμό πρωτοκόλλου, flow label, την κλάση της κίνησης στα διάφορα πεδία του ICMPv6. Επίσης, μπορεί ένας μεταγωγέας να συνδεθεί σε περισσότερους από έναν ελεγκτές.

4.8.3 OpenFlow 1.3

Το OpenFlow 1.3 [72], εισάγει νέες δυνατότητες για την παρακολούθηση, τις υπηρεσίες και τη διαχείριση (Monitoring Operations, Management-OAM). Για το σκοπό αυτό προστίθεται ένας πίνακας-μετρητής (Meter-table) στην αρχιτεκτονική του μεταγωγέα. Ο μετρητής συνδέεται άμεσα με μία εγγραφή του πίνακα-ροής από το αναγνωριστικό του μετρητή και μετρά το ποσοστό των πακέτων που έχουν ανατεθεί. Μια ζώνη-μετρητών (Meter-band) μπορεί να χρησιμοποιηθεί για να περιορίσει το σχετικό πακέτο ή το ρυθμό δεδομένων απορρίψεων των πακέτων όταν γίνεται υπέρβαση από ένα συγκεκριμένο ποσοστό. Αντί να απορρίπτει τα πακέτα, μια ζώνη-μετρητών μπορεί προαιρετικά να επαναχρωματίσει τα πακέτα με τροποποίηση του πεδίου των διαφοροποιημένων υπηρεσιών (DS field). Έτσι, απλά ή πολύπλοκα πλαίσια QoS μπορούν να υλοποιηθούν με το OpenFlow 1.3 και στις επόμενες εκδόσεις του.

4.8.4 OpenFlow 1.4

Το OpenFlow 1.4 [73] κυκλοφόρησε τον Οκτώβριο του 2013. Η ONF βελτίωσε την υποστήριξη για το OpenFlow Extensible Match (OXM). Προστέθηκαν TLV δομές για τις πόρτες, τους πίνακες και τις ουρές στο πρωτόκολλο, και δύσκολα κωδικοποιημένα τμήματα των προηγούμενων εκδόσεων έχουν πλέον αντικατασταθεί από νέες δομές TLV. Η ρύθμιση των οπτικών θυρών είναι πλέον δυνατή. Επιπλέον, οι ελεγκτές μπορούν να στείλουν μηνύματα ελέγχου σε μια ενιαία δέσμη μηνυμάτων στους μεταγωγείς. Συμπεριλαμβάνονται επίσης, μικρές βελτιώσεις στους πίνακες-ομάδας και δυνατότητες παρακολούθησης.

4.8.5 OpenFlow 1.5

Το OpenFlow 1.5 [51] επεκτείνει περαιτέρω την δέσμη μηνυμάτων με το «Scheduled bundle» συμπεριλαμβάνοντας μια ιδιότητα χρόνου εκτέλεσης. Ένας μεταγωγέας που έλαβε προγραμματισμένη δέσμη θα στείλει τα μηνύματα όσο το δυνατόν πιο κοντά στον χρόνο εκτέλεσης. Αυτό ενισχύει περαιτέρω τον συγχρονισμό μεταξύ των πολλαπλών μεταγωγέων. Μέχρι τώρα, ο μεταγωγέας αντιστοιχούσε και επεξεργαζόταν μόνο τα εισερχόμενα πακέτα. Με άλλα λόγια, ήταν δύσκολο να επεξεργαστείς τα πακέτα στην πόρτα εξόδου. Αυτό επιλύθηκε στο OpenFlow 1.5 με τον νέο πίνακα «Egress», επιτρέποντας την αντιστοίχιση πακέτων με βάση τη θύρα εξόδου τους.

5. Ο δικτυακός εξομοιωτής Mininet

5.1 Το λογισμικό Mininet

Το Mininet είναι ένας εξομοιωτής δικτύων που εκτελείται σε Linux, και μπορεί να προσομοιάζει τη λειτουργία εικονικών τερματικών (terminals), μεταγωγέων (switches), ελεγκτών (controllers) και γραμμών (links/lines).

Τα τερματικά στο Mininet τρέχουν επίσης Linux λειτουργικό σύστημα και συμπεριφέρονται σαν πραγματικές συσκευές στις οποίες παρέχεται η δυνατότητα ασφαλούς σύνδεσης (ssh) και εκτέλεσης προγραμμάτων που είναι εγκατεστημένα στο ΛΣ Linux.

Τα πακέτα που μεταφέρονται για την εκτέλεση των προγραμμάτων περνούν από εικονικές-διεπαφές Ethernet με χωρητικότητα γραμμής που μπορεί να παραμετροποιηθεί [68]. Οι μεταγωγείς υποστηρίζουν το πρωτόκολλο OpenFlow για υψηλή ευελιξία παραμετροποίησης σε SDN αρχιτεκτονικές [74].

Για παράδειγμα, όταν δύο προγράμματα, όπως το iperf (μετρητής χωρητικότητας γραμμής δύο σημείων) μεταξύ πελάτη και διακομιστή επικοινωνούν μέσω Mininet, η απόδοση θα μπορούσε να ταιριάξει με αυτή των δύο φυσικών μηχανών.

5.2 Πλεονεκτήματα του Mininet

Το Mininet αποτελεί ένα εύχρηστο και εύκολο εργαλείο στην προσομοίωση δικτύων, το οποίο έχει ευρεία χρήση από τους μηχανικούς δικτύων για την ανάπτυξη προγραμμάτων και μελέτη των αποτελεσμάτων τους. Κάποια από τα πιο σημαντικά του πλεονεκτήματα είναι:

- Η δημιουργία ενός απλού δικτύου πραγματοποιείται σε ελάχιστο χρονικό διάστημα. Αυτό έχει ως συνέπεια την γρήγορη εκτέλεση, τροποποίηση και αποσφαλμάτωσή του.
- Η δημιουργία πολλαπλών τοπολογιών με ένα απλό Python script. Από ένα απλουστευμένο δίκτυο, μέχρι μεγαλύτερες τοπολογίες, όπως το Internet, κάποιο Data Center και άλλα.

- Οποιοδήποτε πρόγραμμα, λογισμικό είναι εγκατεστημένο στο λειτουργικό σύστημα Linux μπορεί να εκτελεστεί, και να μεταφερθεί αργότερα σε πραγματικές συσκευές.
- Υποστηρίζει το OpenFlow πρωτόκολλο κι αποτελεί απλό και οικονομικό τρόπο για πειράματα σε SDN. Συνεπώς, μπορεί να τροποποιηθεί η προώθηση πακέτων, καθώς οι μεταγωγείς του Mininet είναι προγραμματιζόμενοι χρησιμοποιώντας το πρωτόκολλο OpenFlow.
- Η ευέλικτη εκτέλεση του Mininet σε πολλαπλές πλατφόρμες, υπολογιστές, εικονικά μηχανήματα, ακόμα και σε cloud περιβάλλον.
- Είναι έργο Ανοικτού Λογισμικού και υπό συνεχή ανάπτυξη, που σημαίνει πως μπορεί ο καθένας να συνεισφέρει σε αυτό αλλά και να το παραμετροποιήσει όπως θέλει. Συν τοις άλλοις, υπάρχει μια ενεργή κοινότητα αποτελούμενη από χρήστες και προγραμματιστές, η οποία μπορεί να συμβάλλει στην αντιμετώπιση οποιουδήποτε προβλήματος.
- Δίνει τη δυνατότητα CLI (Command Line Interface) για debugging και εντολές για πειραματισμό στο δίκτυο.
- Το βασικότερο πλεονέκτημα είναι πως ο κώδικας που αναπτύσσεται για τους OpenFlow μεταγωγείς και τις υπόλοιπες δικτυακές συσκευές μπορεί να εκτελεστεί σε πραγματικό σύστημα με ελάχιστες αλλαγές.

5.3 Δημιουργία τοπολογιών

Το Mininet δίνει τη δυνατότητα δημιουργίας παραμετροποιήσιμων τοπολογιών με την χρήση της γλώσσας προγραμματισμού Python. Μια τοπολογία μπορεί να δημιουργηθεί μια φορά από τον χρήστη με τις παραμέτρους που εκείνος επιθυμεί, και να επαναχρησιμοποιηθεί πολλές φορές.
Θα εξηγήσουμε σε επόμενο κεφάλαιο με λεπτομέρεια τον κώδικα μιας τέτοιας τοπολογίας, κατά τη διάρκεια επεξήγησης της υλοποίησης που έγινε στα πλαίσια αυτής της εργασίας.

5.4 Ρύθμιση Παραμέτρων

Είναι σημαντικό να αναφερθεί επίσης πως το Mininet μέσω του API του δίνει τη δυνατότητα στον χρήστη να:

- περιορίσει τους πόρους του συστήματος που θα χρησιμοποιεί το εικονικό τερματικό
- να εκτελέσει προγράμματα παράλληλα στα τερματικά
- παραμετροποιήσει το δίκτυο ορίζοντας:
    - IP
    - MAC
    - Να προσθέσει μια στατική εγγραφή ARP σε κάποιο τερματικό

Για περισσότερες πληροφορίες, μπορείτε να ανατρέξετε στην πλήρη τεκμηρίωση του Mininet [75].

5.5 Διεπαφή Προγραμματισμού Εφαρμογών Mininet (API)

Το API του Mininet αποτελείται από αρκετές κλάσεις Python. Κάποια παραδείγματα αυτών είναι: οι Topo, Mininet, Host, Switch, Link και οι υποκλάσεις αυτών. Είναι σκόπιμο αυτές οι κλάσεις να χωριστούν σε επίπεδα, μιας και συνήθως εκείνες που βρίσκονται σε υψηλότερο επίπεδο, χρησιμοποιούν εκείνες που είναι σε χαμηλότερο.

Το API λοιπόν χωρίζεται σε τρία επίπεδα:

- **API Χαμηλού επιπέδου:** αποτελείται από τον κόμβο βάσης και τις κλάσεις συνδέσμων. Τέτοιες είναι οι Host, Switch, Link και οι υποκλάσεις τους. Αυτές μπορούν να χρησιμοποιηθούν και ανεξάρτητα για τη δημιουργία ενός δικτύου, αλλά είναι λίγο δύσχρηστες.
- **API Μεσαίου επιπέδου:** προσθέτει το αντικείμενο Mininet, το οποίο λειτουργεί ως container για κόμβους και συνδέσμους. Παρέχει μια σειρά μεθόδων (όπως οι addHost(), addSwitch() και addLink()) για την προσθήκη κόμβων και συνδέσεων σε ένα δίκτυο, καθώς επίσης και δυνατότητα διαμόρφωσης, εκκίνησης και τερματισμού δικτύου.
- **API Υψηλού επιπέδου:** προσθέτει ένα αφαιρετικό πρότυπο τοπολογίας, την κλάση Topo, η οποία παρέχει τη δυνατότητα δημιουργίας

επαναχρησιμοποιήσιμων, παραμετροποιήσιμων προτύπων τοπολογίας. Αυτά τα πρότυπα μπορούν να περαστούν ως παράμετροι στην εντολή mn (μέσω της επιλογής --custom) και να χρησιμοποιηθούν από τη γραμμή εντολών.

5.6 Μέτρηση Απόδοσης

Τα πιο βασικά εργαλεία μέτρησης απόδοσης στο Mininet είναι:

- Εύρος ζώνης (bwm-ng, ethstats)
- Καθυστέρηση (ping)
- Ουρές Αναμονής (tc, περιλαμβάνεται στο monitor.py)
- Στατιστικά TCP (tcp_probe)
- Χρήση CPU (top, cpuacct)

5.7 Πρωτόκολλο OpenFlow και Προσαρμοσμένη Δρομολόγηση

Ένα από τα πιο σημαντικά χαρακτηριστικά του Mininet είναι ότι χρησιμοποιεί το SDN. Χρησιμοποιώντας το πρωτόκολλο OpenFlow και σχετικά εργαλεία, δίνεται η δυνατότητα να προγραμματίσεις μεταγωγείς (switches) έτσι ώστε να κάνουν σχεδόν τα πάντα με τα πακέτα που λαμβάνουν. Το OpenFlow καθιστά εξομοιωτές όπως το Mininet πολύ χρήσιμους, καθώς τα μοντέλα δικτύων, συμπεριλαμβανομένης της προσαρμοσμένης προώθησης πακέτων με την χρήση του OpenFlow, μπορούν εύκολα να μεταφερθούν σε φυσικά OpenFlow switches.

5.7.1 Ελεγκτές OpenFlow

Εάν εκκινηθεί το Mininet χωρίς να διευκρινιστεί ο ελεγκτής (controller), χρησιμοποιείται από προεπιλογή ο ελεγκτής τύπου ovsc.
Η ισοδύναμη εντολή είναι

sudo mn --controller default

Ο συγκεκριμένος τύπος ελεγκτή υλοποιεί έναν απλό μεταγωγέα εκμάθησης (learning switch) Ethernet, και μπορεί να υποστηρίξει μέχρι 16 επιπλέον μεταγωγείς.

Αν κληθεί ο δομητής Mininet() στο script χωρίς διευκρίνιση κλάσης ελεγκτή, χρησιμοποιείται η προεπιλεγμένη κλάση Controller(), η οποία θα δημιουργήσει έναν ελεγκτή τύπου Stanford/OpenFlow.

Ωστόσο, παρέχεται η δυνατότητα στον χρήστη να δημιουργήσει και να χρησιμοποιήσει έναν διαφορετικό ελεγκτή. Αυτό μπορεί να γίνει δημιουργώντας μια υποκλάση της Controller() και μεταφέροντας την στα αρχεία συστήματος του Mininet.

5.7.2 Εξωτερικοί Ελεγκτές OpenFlow

Είναι πιθανό, να χρειαστεί ή να είναι σκόπιμο να συνδεθεί το Mininet σε κάποιον υπάρχοντα ελεγκτή, ο οποίος όμως να εκτελείται κάπου αλλού, για παράδειγμα σε κάποιο άλλο σύστημα μέσα στο LAN, ή σε κάποιο σύστημα στο Internet.

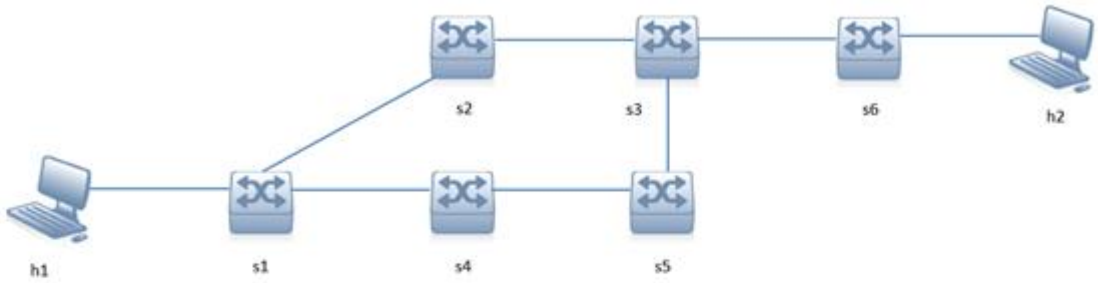
Η κλάση RemoteController ενεργεί ως διακομιστής μεσολάβησης (proxy) για κάποιον ελεγκτή ο οποίος μπορεί να εκτελείται οπουδήποτε στο δίκτυο ελέγχου, και ο οποίος πρέπει να ξεκινήσει και να τερματιστεί χειροκίνητα ή με κάποιο άλλο μηχανισμό εκτός του άμεσου ελέγχου του Mininet.

6. Κατασκευή τοπολογίας δικτύου SDN & Ανίχνευση της (Επεξήγηση κώδικα)

Στα παρακάτω κεφάλαια θα παρουσιάσουμε τον κώδικα που αναπτύξαμε και χρησιμοποιήσαμε για το σκοπό αυτής της διπλωματικής εργασίας. Θα περιγραφούν τα κυριότερα μέρη αυτού, ενώ ολόκληρο τον κώδικα μπορείτε να τον βρείτε στο BitBucket[1] (https://bitbucket.org/nyxteridas/sdn-topology-discovery-using-pox/).

6.1 Κώδικας τοπολογίας (topology.py)

Σε αυτό το κεφάλαιο θα εξηγήσουμε τον κώδικα που χρησιμοποιήθηκε για την δημιουργία της τοπολογίας που φαίνεται στην Εικόνα 20.



Εικόνα 20: Τοπολογία δικτύου παραδείγματος

Μέσα στο script ορίζουμε τη μέθοδο myNetwork() η οποία θα αναλάβει όλη τη διαδικασία ορισμού της τοπολογίας.

Αρχικά δημιουργούμε ένα αντικείμενο τύπου Mininet, το οποίο θα αντιπροσωπεύει την τοπολογία μας. Δεν της δίνουμε ως όρισμα κάποιο συγκεκριμένο αντικείμενο τοπολογίας, μιας και θα την δημιουργήσουμε εμείς (topo=None, build=False).

Ορίζουμε:

- τη βάση των IP διευθύνσεων που θα λάβουν οι hosts, και το subnet αυτής (ipBase='10.0.0.0/8'),

[1] Το BitBucket είναι μια online υπηρεσία version control συστήματος όπως το GitHub. Ανήκει στην Atlassian και σου δίνει τη δυνατότητα να χρησιμοποιήσεις είτε Git, είτε Mercurial για τον έλεγχο και την αναθεώρηση του κώδικά σου. (https://bitbucket.org/)

- την βασική κλάση που θα χρησιμοποιηθεί για τη δημιουργία των συνδέσμων (link=TCLink),
- την επιλογή να έχουν στατικές MAC διευθύνσεις τα ζεύγη (autoStaticArp=True) ώστε να αποφύγουμε τα broadcast μηνύματα
- και τέλος αφήνουμε κενό τον ελεγκτή. Θα τον δημιουργήσουμε αργότερα.

```
def myNetwork():

    net = Mininet( topo=None,
                   build=False,
                   ipBase='10.0.0.0/8',
                   link=TCLink, controller=None, autoStaticArp=True)
….
```

Απόσπασμα Κώδικα 1: Ορισμός δικτύου στο Mininet

Αμέσως μετά ορίζουμε τον ελεγκτή μας, και τον προσθέτουμε στο δίκτυό μας (net.addController(…)). Η μέθοδος «addController» επί της ουσίας, προσθέτει ένα αντικείμενο ελεγκτή μέσα σε μια λίστα ελεγκτών που περιέχει το αντικείμενο «net». Αυτός ο ελεγκτής, δημιουργείται βάσει των παραμέτρων που δίνουμε ως ορίσματα κατά την κλήση της μεθόδου:

- το όνομά του (name='c0'),
- τον τύπο του ελεγκτή (controller=RemoteController),
- την διεύθυνση στην οποία βρίσκεται (ip='127.0.0.1'),
- το πρωτόκολλο στο οποίο «ακούει» (protocol='tcp')
- και την πόρτα στην οποία περιμένει τα μηνύματα (port=6633)

Στην περίπτωσή μας ορίζουμε τον ελεγκτή ως Remote Controller, δηλαδή ως έναν ελεγκτή που βρίσκεται εκτός του ελέγχου του Mininet. Αυτό θα μας χρησιμεύσει στη δεύτερη φάση όπου θα χρειαστεί να εκτελέσουμε τον Controller με το script ανίχνευσης τοπολογίας.

```
….
    info( '*** Adding controller\n' )
    c0=net.addController(name='c0',
                      controller=RemoteController,
                      ip='127.0.0.1',
                      protocol='tcp',
                      port=6633)
….
```

Απόσπασμα Κώδικα 2: Ορισμός Ελεγκτή στο Mininet

Σειρά έχει η προσθήκη των switches (μεταγωγέων) για την τοπολογία μας (net.addSwitch(…)). Η μέθοδος «addSwitch», παρομοίως με την «addController», προσθέτει ένα αντικείμενο μεταγωγέα μέσα σε μια λίστα μεταγωγέων που περιέχει το αντικείμενο «net».

Δημιουργούμε έξι switches με:

- ονόματα της μορφής «s_x"», όπου x το νούμερο του εν λόγω μεταγωγέα ('s1', 's2', 's3' κ.ο.κ) ,
- το Datapath ID για το κάθε switch (dpid='1', dpid='2', dpid='3' κ.ο.κ)
- και τέλος, τον τύπο του switch που θα χρησιμοποιήσουμε. Στην περίπτωσή μας είναι το OpenVSwitch (cls=OVSKernelSwitch)

```
….
   info( '*** Add switches\n')
   s4 = net.addSwitch('s4', cls=OVSKernelSwitch, dpid='4')
   s6 = net.addSwitch('s6', cls=OVSKernelSwitch, dpid='6')
   s5 = net.addSwitch('s5', cls=OVSKernelSwitch, dpid='5')
   s1 = net.addSwitch('s1', cls=OVSKernelSwitch, dpid='1')
   s2 = net.addSwitch('s2', cls=OVSKernelSwitch, dpid='2')
   s3 = net.addSwitch('s3', cls=OVSKernelSwitch, dpid='3')
….
```

Απόσπασμα Κώδικα 3: Ορισμός Μεταγωγέων στο Mininet

Οι κόμβοι που έχουν μείνει να ορίσουμε, είναι οι hosts μας (net.addHost(…)). Η μέθοδος «addHost», προσθέτει ένα αντικείμενο host μέσα στη λίστα των hosts που περιέχει το αντικείμενο «net».

Θα χρειαστούμε δύο hosts με:

- ονόματα της μορφής «h_x», όπου x το νούμερο του εν λόγω host ('h1', 'h2') ,
- την IP διεύθυνση του (ip='10.0.0.1', ip='10.0.0.2')
- τον τύπο του (cls=Host)
- και το defaultRoute, το οποίο χρησιμοποιείται για να ενημερώσει τον host με ποιο router θα πρέπει να μιλήσει ώστε να έχει πρόσβαση στο internet. Στην περίπτωσή μας δε χρειάζεται κάτι τέτοιο (defaultRoute=None).

```
….
   info( '*** Add hosts\n')
   h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
   h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
….
```

Απόσπασμα Κώδικα 4: Ορισμός Host στο Mininet

Μερικοί διάσπαρτοι κόμβοι δεν αποτελούν δίκτυο αν δεν ενωθούν μεταξύ τους. Έτσι, το τελευταίο στάδιο σχεδίασης είναι να ορίσουμε τους συνδέσμους μεταξύ των διαφόρων κόμβων (net.addLink(…)). Η μέθοδος «addLink», προσθέτει ένα αντικείμενο συνδέσμου μέσα σε μια λίστα με συνδέσμους που περιέχει το αντικείμενο «net». Αποτελεί ίσως το ευκολότερο βήμα ορισμού, μιας και τα μόνα που έχουμε να δώσουμε ως ορίσματα στη μέθοδο «addLink» είναι οι δύο άκρες του κάθε συνδέσμου (net.addLink(h1, s1), net.addLink(s2, s1) κ.ο.κ).

```
….
   info( '*** Add links\n')
   net.addLink(h1, s1)
   net.addLink(s2, s1)
   net.addLink(s1, s4)
   net.addLink(s4, s5)
   net.addLink(s5, s3)
   net.addLink(s2, s3)
   net.addLink(s3, s6)
   net.addLink(s6, h2)
….
```

Απόσπασμα Κώδικα 5: Ορισμός Συνδέσμων στο Mininet

Αφού ολοκληρώσαμε το όρισμα των αντικειμένων που αποτελούν την τοπολογία μας, σειρά έχει η έναρξη του δικτύου. Αυτό θα γίνει σε τέσσερα επιμέρους βήματα:

1. Έναρξη δικτύου (net.build()). Με την κλήση της μεθόδου «build» μέσω του αντικειμένου «net», λέμε στο Mininet να «χτίσει» το δίκτυό μας, με βάση τα αντικείμενα που ορίσαμε μέσα στο αντικείμενο «net».

```
….
info( '*** Starting network\n')
   net.build()
….
```

Απόσπασμα Κώδικα 6: Εκκίνηση Δικτύου

2. Το δίκτυό μας έχει ξεκινήσει, συνεπώς το επίπεδο του data plane είναι έτοιμο, όχι όμως και το επίπεδο του control plane. Επόμενο στάδιο είναι να ξεκινήσουμε τον/τους ελεγκτές μας (controller.start()). Επειδή όπως έχουμε αναφέρει και στο κομμάτι της θεωρίας, ένα SDN δίκτυο μπορεί να έχει παραπάνω από έναν ελεγκτή, στην προσέγγισή μας χρησιμοποιούμε μια for loop ώστε να προσπελάσουμε όλους τους ελεγκτές που έχουν οριστεί στη

σχετική λίστα του αντικειμένου «net», και να τους εκκινήσουμε.

```
….
  info( '*** Starting controllers\n')
  for controller in net.controllers:
     controller.start()
….
```

Απόσπασμα Κώδικα 7: Εκκίνηση Ελεγκτών

3. Προηγουμένως εκκινήσαμε τους κόμβους μέσω της εντολής `net.build()`, αλλά δεν ορίσαμε ποιος ελεγκτής θα διαχειρίζεται τους μεταγωγείς μας. Με τις εντολές που φαίνονται στο Απόσπασμα Κώδικα 8, παίρνουμε από το αντικείμενο «net» τον κάθε μεταγωγέα μέσω του ονόματός του, και του ορίζουμε σε ποιον ελεγκτή θα ακούει. Στην περίπτωσή μας έχουμε μόνο έναν ελεγκτή (c0).

```
….
  info( '*** Starting switches\n')
  net.get('s4').start([c0])
  net.get('s6').start([c0])
  net.get('s5').start([c0])
  net.get('s1').start([c0])
  net.get('s2').start([c0])
  net.get('s3').start([c0])
….
```

Απόσπασμα Κώδικα 8: Ορισμός controller για κάθε switch

4. Επιπλέον, αφού έχουμε ήδη ξεκινήσει τα switches μας, πρέπει να θέσουμε σε αυτά και συγκεκριμένες IP διευθύνσεις, ώστε να γνωρίζουμε ποιο βρίσκεται πού. Αυτό γίνεται με την εκτέλεση μιας command μέσω cmd όπως φαίνεται παρακάτω:

```
….
  info( '*** Post configure switches and hosts\n')
  s4.cmd('ifconfig s4 10.0.0.6')
  s6.cmd('ifconfig s6 10.0.0.8')
  s5.cmd('ifconfig s5 10.0.0.7')
  s1.cmd('ifconfig s1 10.0.0.3')
  s2.cmd('ifconfig s2 10.0.0.4')
  s3.cmd('ifconfig s3 10.0.0.5')
….
```

Απόσπασμα Κώδικα 9: Ορισμός IP για τα switches

Το τελευταίο κομμάτι κώδικα, επί της ουσίας μας ανοίγει ένα Command Line Interface στο δίκτυο «net» που μόλις εκκινήσαμε.

```
….
   CLI(net)
….
```

Απόσπασμα Κώδικα 10: Άνοιγμα CLI στο δίκτυο

Για την εκτέλεση του παραπάνω script, θα χρειαστεί μέσω cmd να εκτελεστεί η εντολή:

sudo python <path-to-file>/topology.py

6.2 Ανίχνευση τοπολογίας

6.2.1 Περιγραφή κώδικα ανίχνευσης (topoDiscovery.py)

Σε αυτό το κεφάλαιο θα εξηγήσουμε τον κώδικα που χρησιμοποιήσαμε για να επεκτείνουμε τη λειτουργικότητα του POX με τη χρήση του OpenFlow, ώστε να ανιχνεύει και να μας δίνει πληροφορίες σχετικές με την τοπολογία του δικτύου.

Τα βασικά μέρη αυτού του script είναι:

1. **Κλάση topoDiscovery:** πρόκειται για την κλάση που υλοποιεί τη λειτουργία που θέλουμε και περιέχει όλες τις μεθόδους που θα χρειαστούμε. Η υλοποίησή μας βασίζεται στα events που δημιουργούνται μέσα στο δίκτυο. Από τη στιγμή που θέλουμε ο POX να «ακούει» αυτά τα events και αναλόγως να δημιουργεί κάποιο output, πρέπει η κλαση μας να κληρονομεί από την κλάση EventMixin (class topoDiscovery(EventMixin)).
2. **Global μεταβλητή linkCounter:** τη χρησιμοποιούμε για να κρατάμε τον αριθμό των links που έχουν δημιουργηθεί μέσα στο δίκτυό μας. Την χρειαζόμαστε κατά τη διάρκεια εμπλουτισμού του JSON αρχείου που κρατά την πληροφορία της τοπολογίας μας.
3. **Μέθοδος __init__:** πρόκειται για τον δομητή (constructor) της κλάσης μας «topoDiscovery». Καλείται μια φορά κατά την έναρξη του controller. Μέσα σε αυτή τη μέθοδο ορίζονται οι listeners που θα χρειαστούμε για τα events που πρέπει να παρακολουθήσουμε. Τα events αυτά παράγονται από τα παρακάτω components που χρησιμοποιούμε συνδυαστικά:

- **openflow.discovery (LinkEvent):** το component αυτό στέλνει LLDP μηνύματα σε όλα τα OpenFlow switches, έτσι ώστε να ανιχνεύσει την τοπολογία του δικτύου. Όταν κάποιο link δημιουργείται ή «πέφτει», παράγεται στο δίκτυο ένα event του τύπου LinkEvent. Μέσα σε αυτά τα events περιλαμβάνονται οι παρακάτω πληροφορίες:

Όνομα	Τιμή
dpid1	Το DataPath ID του ενός από τα switches που συμμετέχει στο link
port1	Το port dpid1 που συμμετέχει στο link
dpid2	Το DataPath ID του άλλου switch που συμμετέχει στο link
port2	Το port dpid2 που συμμετέχει στο link
uni	Μία μοναδική αναπαράσταση του link. Αυτό κανονικοποιεί τη σειρά των DPIDs ώστε να μπορούμε να συγκρίνουμε δύο links μεταξύ τους
end[0 or 1]	Οι άκρες ενός link σαν tuple. Π.χ. end[0] = (dpid1,port1)

Πίνακας 1: Πληροφορίες που περιλαμβάνονται στα LinkEvents

- **host_tracker (Host Events):** προσπαθεί να ελέγξει τους hosts στο δίκτυο. Έτσι αν κάτι αλλάξει, δημιουργεί ένα host event. Εν ολίγοις, μαθαίνει τις MAC των hosts και ανά τακτά χρονικά διαστήματα στέλνει μηνύματα ARP ώστε να δει αν ο host είναι ακόμα εκεί. Βασίζεται στα πακέτα που έρχονται στον controller, άρα θα πρέπει να χρησιμοποιηθεί ταυτόχρονα με κάποιο component όπως το forwarding.l2_learning. Περισσότερα για αυτό θα δούμε στη συνέχεια. Μέσα σε αυτά τα events περιλαμβάνονται και οι πληροφορίες για το link που συνδέει τον host με κάποιο switch.

4. **Μέθοδος handle_LinkEvent:** εκτελείται κάθε φορά που δημιουργείται ένα LinkEvent. Ο ρόλος της είναι να ενημερώσει την τοπολογία σχετικά με τα switches και την κατάσταση των links μεταξύ τους.
5. **Μέθοδος handle_HostEvent:** εκτελείται κάθε φορά που δημιουργείται ένα HostEvent. Ο ρόλος της είναι να ενημερώσει την τοπολογία σχετικά με τους hosts και τα links αυτών με τα switches.

6. **Μέθοδος launch:** εκτελείται μια φορά στην έναρξη του script και ως σκοπό έχει να ενεργοποιήσει το component host_tracker, να αρχικοποιήσει το JSON αρχείο στο οποίο αποθηκεύουμε την πληροφορία για την τοπολογία μας, και να εκκινήσει την εκτέλεση του script μας δημιουργώντας ένα νέο αντικείμενο της κλάσης topoDiscovery.

Παρακάτω θα περιγράψουμε τις βασικές μεθόδους του script μας:

- **handle_LinkEvent:** Ως όρισμα δέχεται το event το οποίο την τρίγκαρε και το αντικείμενο τύπου topoDiscovery μέσα στο οποίο κλήθηκε. Από το event που δημιουργήθηκε, κρατάμε το link και εξάγουμε από αυτό τα dpid των switches, και τις πόρτες μέσω των οποίων συνδέονται. Τέλος, κρατάμε σε μια μεταβλητή την μοναδική αναπαράσταση του συγκεκριμένου link.

```
....
"""
Hold the link which triggered the event
"""
l = event.link

"""
Find out between which switches and ports was this link
"""
sw1 = l.dpid1
sw2 = l.dpid2
pt1 = l.port1
pt2 = l.port2

"""
Take a unique link representation in order to avoid duplicates
"""
uniqueLinlRepr = l.uni
....
```

Απόσπασμα Κώδικα 11: Ανάγνωση πληροφοριών ενός LinkEvent

Το επόμενο στάδιο αφορά την ανανέωση των πληροφοριών που υπάρχουν μέσα στο JSON αρχείο που κρατά την πληροφορία της τοπολογίας μας. Συνεπώς:

1. Ανοίγουμε το αρχείο μας για διάβασμα, και την πληροφορία που περιέχει την εξάγουμε σε μορφή python dictionary για να την χειριστούμε ευκολότερα.
2. Διαμορφώνουμε τα ονόματα των μεταγωγέων ώστε να έχουν τη μορφή «s_x», όπου x το μοναδικό ID του DataPath (dpid).
3. Διαμορφώνουμε την MAC διεύθυνση των switches ώστε να ταιριάζει με την επίσημη αναπαράσταση αυτών των διευθύνσεων (διαχωρισμός με «:» αντί για «-»)
4. Διαβάζουμε από το dictionary που δημιουργήσαμε από το αρχείο, το tag «switches», και σε αυτό προσθέτουμε τις πληροφορίες για τα switches που συμμετέχουν στο event μας. Αν αυτά τα δύο switches υπάρχουν ήδη μέσα στο tag αυτό, τότε θα ενημερωθούν με τα νέα στοιχεία τους.

```
....
"""
open configuration json for reading
and save it as dictionary to the variable jsonToPython.
Update the info according to what you are reading from network.
Write it back to the configuration json.
"""
with open('configuration.json', 'r') as myfile:
    jsonData=myfile.read().replace('\n', ' ')
jsonToPython = json.loads(jsonData)

"""
create the names of the switches
"""
s1Num = 's%d' %l.dpid1
s2Num = 's%d' %l.dpid2

"""
read the part that contains the switches from the dictionary which
was read previously from file, and store it
"""
switches = jsonToPython['switches']

"""
Format the MAC address of switches and store it
"""
switches[s1Num] = '%s' %dpid_to_str(l.dpid1).replace('-',':')
switches[s2Num] = '%s' %dpid_to_str(l.dpid2).replace('-',':')

"""
update the element "switches" in the dictionary
which was read from the configuration JSON
```

```
"""
jsonToPython['switches'] = switches
....
```

Απόσπασμα Κώδικα 12: Ενημέρωση πληροφοριών Μεταγωγέων στο JSON

5. Διαβάζουμε από το dictionary που δημιουργήσαμε από το αρχείο, το tag «link».
6. Διαμορφώνουμε το όνομα των switches βασιζόμενοι στην μοναδική αναπαράσταση του link που δημιουργήσαμε προηγουμένως.
7. Στην περίπτωση που το event μας ενημερώνει πως ένα link προστέθηκε στο δίκτυό μας, τότε σκανάρουμε το dictionary που σχετίζεται με τα links και ελέγχουμε αν ανάμεσα στα δύο switches μας υπάρχει ήδη δηλωμένη κάποια σύνδεση. Σε αυτή την περίπτωση δεν γίνεται καμία ενημέρωση. Αν όμως δεν υπάρχει κάποια ήδη δηλωμένη σύνδεση ανάμεσα στους μεταγωγείς μας, τότε το script δημιουργεί μια νέα εγγραφή για το JSON, στην οποία περιλαμβάνει έναν μοναδικό αριθμό για το link (χρησιμοποιώντας την global μεταβλητή linkCounter), τις πόρτες μέσω των οποίων συνδέονται οι μεταγωγείς, και τα ονόματα των μεταγωγέων.
Αν πρόκειται για πτώση συνδέσμου, σκανάρουμε το dictionary που σχετίζεται με τα links και ελέγχουμε αν ανάμεσα στα δύο switches μας υπάρχει ήδη δηλωμένη κάποια σύνδεση. Αν υπάρχει, τότε αυτή αφαιρείται.
8. Προσθέτουμε στο dictionary τις ανανεωμένες πληροφορίες για το link που συμμετέχει στο event μας.
9. Γράφουμε όλες τις αλλαγές πίσω στο JSON αρχείο.

```
....
"""
read the part that contains the links from the dictionary which was
read previously from file, and store it
"""
links = jsonToPython['link']

"""
create the names of the switches based on the unique link
representation
"""
s1Num = 's%d' %uniqueLinlRepr.dpid1
s2Num = 's%d' %uniqueLinlRepr.dpid2
```

```
"""
If an event is added or removed, means that a link is up or down.
Update all the relevant info in the configuration JSON
"""
if event.added:

    """
    Search in the existing list of link's if this one already exists
    If it exists, do nothing.
    """
    linkExistFlag = False

    for key,val in links.items():
        for key2,val2 in val.items():
            if key2 == "connection":
                if val2 == [s1Num,s2Num]:
                    linkExistFlag = True
    """
    if the link does not exist, give it a unique number
    and insert it in the list
    """
    if not linkExistFlag:
        linkCounter += 1
        links[format(linkCounter, 'b').zfill(10)] = {'connection':
[s1Num,s2Num], 'ports': [uniqueLinlRepr.port1,uniqueLinlRepr.port2]}

elif event.removed:

    """
    Search in the existing list of link's if this one already exists
    If it exists, remove it.
    """

    for key,val in links.items():
        for key2,val2 in val.items():
            if key2 == "connection":
                if val2 == [s1Num,s2Num]:
                    del links[key]

"""
update the element "link" in the dictionary
which was read from the configuration JSON
"""
jsonToPython['link'] = links


""" Write back to the configuration json the updates. """
with open('configuration.json', 'w') as outfile:
   json.dump(jsonToPython, outfile, indent=4, separators=(',', ': '),
sort_keys=True)
....
```

Απόσπασμα Κώδικα 13: Ενημέρωση πληροφοριών Συνδέσμων μεταξύ Μεταγωγέων στο JSON

- **handle_HostEvent:** Ως όρισμα δέχεται το event το οποίο την τρίγκαρε και το αντικείμενο τύπου topoDiscovery μέσα στο οποίο κλήθηκε. Από το event που δημιουργήθηκε, εξάγουμε την πληροφορία που αφορά την MAC διεύθυνση του host, την πόρτα του switch στην οποία συνδέεται και το DataPath ID του switch στο οποίο συνδέεται.

```
....
"""
In our case we need to know which host is with us,
which is its MAC, in which switch it is connected,
and in which port
"""
macaddr = event.entry.macaddr.toStr()
port = event.entry.port
dp = event.entry.dpid

....
```

Απόσπασμα Κώδικα 14: Ανάγνωση πληροφοριών ενός HostEvent

Το επόμενο στάδιο αφορά την ανανέωση των πληροφοριών που υπάρχουν μέσα στο JSON αρχείο που κρατά την πληροφορία της τοπολογίας μας. Συνεπώς:

1. Ανοίγουμε το αρχείο μας για διάβασμα, και την πληροφορία που περιέχει την εξάγουμε σε μορφή python dictionary για να την χειριστούμε ευκολότερα.
2. Διαβάζουμε από το dictionary που δημιουργήσαμε από το αρχείο, το tag «hosts».
3. Διαμορφώνουμε το όνομα του host ώστε να έχει τη μορφή «h_x», όπου x το μοναδικό ID του host.
4. Διαμορφώνουμε το όνομα του switch ώστε να έχει τη μορφή «s_x», όπου x το DataPath ID του switch.
5. Ανανεώνουμε τις πληροφορίες για τον host στο dictionary.

```
....
"""
open configuration json for reading
and save it as dictionary to the variable jsonToPython.
Update the info according to what you are reading from network.
Write it back to the configuration json.
"""
with open('configuration.json', 'r') as myfile:
    jsonData=myfile.read().replace('\n', ' ')
```

```
jsonToPython = json.loads(jsonData)

"""
read the part that contains the hosts from the dictionary which was
read previously from file, and store it
"""
hosts = jsonToPython['hosts']

"""
create the names of the hosts
"""
hNum = 'h%d' %port

"""
create the names of the switches
"""
sNum = 's%d' %dp

"""
Format the entry of a host
"""
hosts[hNum] = {"mac": macaddr, "ip": "0.0.0.0", "color": "red"}

"""
update the element "hosts" in the dictionary
which was read from the configuration JSON
"""
jsonToPython['hosts'] = hosts
....
```

Απόσπασμα Κώδικα 15: Ενημέρωση πληροφοριών hosts στο JSON

6. Διαβάζουμε από το dictionary που δημιουργήσαμε από το αρχείο, το tag «link».
7. Σκανάρουμε το dictionary που σχετίζεται με τα links και ελέγχουμε αν ανάμεσα στο συγκεκριμένο ζεύγος host-switch υπάρχει ήδη δηλωμένη κάποια σύνδεση. Σε αυτή την περίπτωση δεν γίνεται καμία ενημέρωση. Αν όμως δεν υπάρχει κάποια ήδη δηλωμένη σύνδεση, τότε το script δημιουργεί μια νέα εγγραφή για το JSON, στην οποία περιλαμβάνει έναν μοναδικό αριθμό για το link (χρησιμοποιώντας την global μεταβλητή linkCounter), τις πόρτες μέσω των οποίων συνδέονται host και switch, και τα ονόματα τους.
8. Προσθέτουμε στο dictionary τις ανανεωμένες πληροφορίες για το link που συμμετέχει στο event μας.
9. Γράφουμε όλες τις αλλαγές πίσω στο JSON αρχείο.

```
....
"""
```

```
update the list of links with those from hosts
read the part that contains the links from the dictionary which was
read previously from file, and store it
"""
links = jsonToPython['link']

"""
Search in the existing list of link's if this one already exists
If it exists, do nothing.
"""
linkExistFlag = False

for key,val in links.items():
    for key2,val2 in val.items():
        if key2 == "connection":
            if val2 == [hNum,sNum]:
                linkExistFlag = True

"""
if the link does not exist, give it a unique number
and insert it in the list
"""
if not linkExistFlag:
    linkCounter += 1
    links[format(linkCounter, 'b').zfill(10)] = {'connection':
[hNum,sNum], 'ports': [1,port]}

"""
Write back to the configuration json the updates.
"""
with open('configuration.json', 'w') as outfile:
   json.dump(jsonToPython, outfile, indent=4, separators=(',', ': '),
sort_keys=True)
....
```

Απόσπασμα Κώδικα 16: Ενημέρωση πληροφοριών Συνδέσμων μεταξύ Host-Μεταγωγέα στο JSON

Για την εκτέλεση του παραπάνω script, θα χρειαστεί να μεταφέρουμε το script μας στην τοποθεσία */home/mininet/pox/ext/*, και μέσω cmd να εκτελεστεί ο controller POX, και να δοθούν ως παράμετροι:

- openflow.discovery: για να μπορούμε να ανιχνεύσουμε την τοπολογία του δικτύο όπως περιγράψαμε παραπάνω
- forwarding.l2_learning: για να μπορούμε να προωθούμε την κίνηση στον controller
- και την κλάση μας topoDiscovery, για να μπορέσουμε να επεκτείνουμε τη λειτουργικότητα όπως σχεδιάζαμε.

Η εντολή θα μοιάζει κάπως έτσι:

../pox.py topoDiscovery openflow.discovery forwarding.l2_learning

6.2.2 Περιγραφή αρχείου JSON (configuration.json)

Το JSON αρχείο που κρατά την πληροφορία της τοπολογίας μας, έχει την παρακάτω μορφή:

```
{
  "hosts": {
    "h1": {
      "mac": "00:00:00:00:00:01",
      "ip": "10.0.0.1",
      "color": "red"
    },
    "h2": {
      "mac": "00:00:00:00:00:02",
      "ip": "10.0.0.2",
      "color": "red"
    }
  },

  "switches":{
      "s1": "00:00:00:00:00:01",
      "s2" : "00:00:00:00:00:02"
  },

  "link":
  {
    "0000000001":{
      "connection":["h1","s1"],
      "ports":[1,1]
    },
    "0000000010":{
      "connection":["s1","s2"],
     "ports":[2,1]
    },  "0000000100":{
     "connection":["s2","h2"],
     "ports":[2,1]
    }
  }
}
```

Απόσπασμα Κώδικα 17: Μορφή JSON τοπολογίας

Η πληροφορία χωρίζεται σε τρεις κατηγορίες:

1. **Hosts:** Περιέχει πληροφορίες για όλους τους hosts. Επί της ουσίας πρόκειται για εμφωλευμένα tuples της μορφής «key:value», τα οποία ως key χρησιμοποιούν το όνομα του host, και ως value περιέχουν την MAC διεύθυνση, την IP του και ένα αναγνωριστικό (color).

2. **Link:** Περιέχει πληροφορίες για όλα τα links. Επί της ουσίας πρόκειται για εμφωλευμένα tuples της μορφής «key:value», τα οποία ως key χρησιμοποιούν ένα μοναδικό δυαδικό αριθμό για κάθε link, και ως value περιέχουν έναν πίνακα με τα ονόματα των κόμβων τους οποίους συνδέουν, και έναν πίνακα πορτών.
3. **Switches:** Περιέχει πληροφορίες για όλα τα switches. Οι τιμές του έχουν τη μορφή «key:value», όπου key είναι το όνομα του switch, και value η MAC διεύθυνσή του.

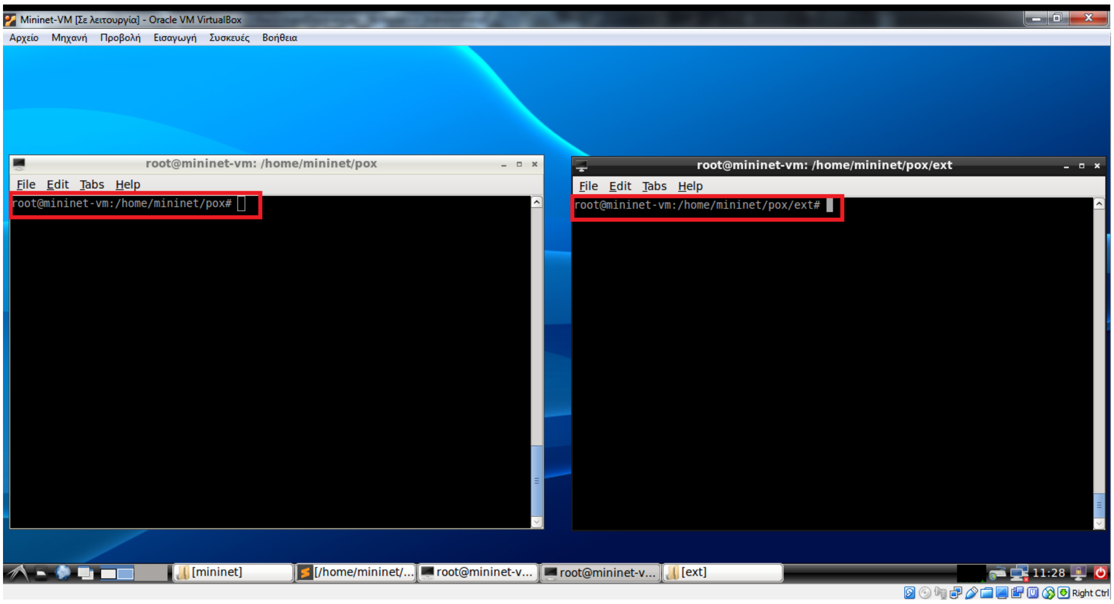
6.3 Παράδειγμα Εκτέλεσης

Σε αυτό το κεφάλαιο θα δούμε ένα παράδειγμα εκτέλεσης των παραπάνω script. Η εκτέλεση έγινε στο περιβάλλον Mininet.

Τα paths στα οποία έχουμε τα scripts κατά τη διάρκεια αυτής της παρουσίασης είναι:

- topology.py: /home/mininet/pox/
- topoDiscovery.py: /home/mininet/pox/ext/

Ανοίγουμε δύο ξεχωριστά terminals στο περιβάλλον του Mininet και μεταφέρουμε το καθένα σε ένα από τα παραπάνω paths:

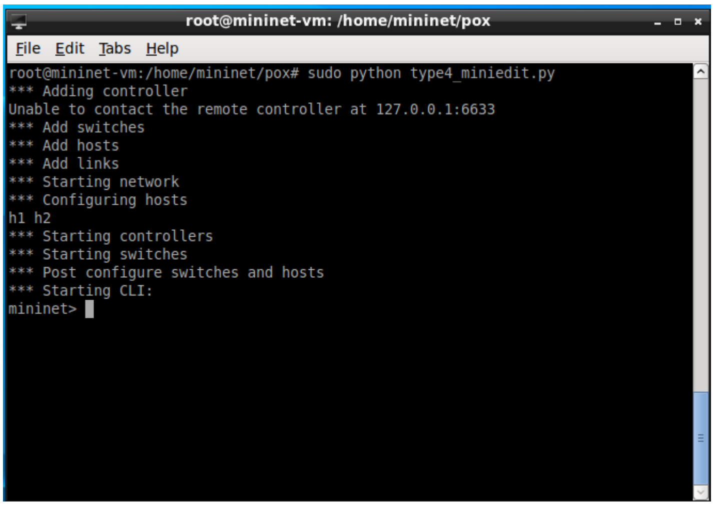


Εικόνα 21: Περιβάλλον Mininet-Παράδειγμα εκτέλεσης

Στο πρώτο εκτελούμε το script δημιουργίας της τοπολογίας με την εντολή:

sudo python <path-to-file>/topology.py

και περιμένουμε να ολοκληρωθεί.

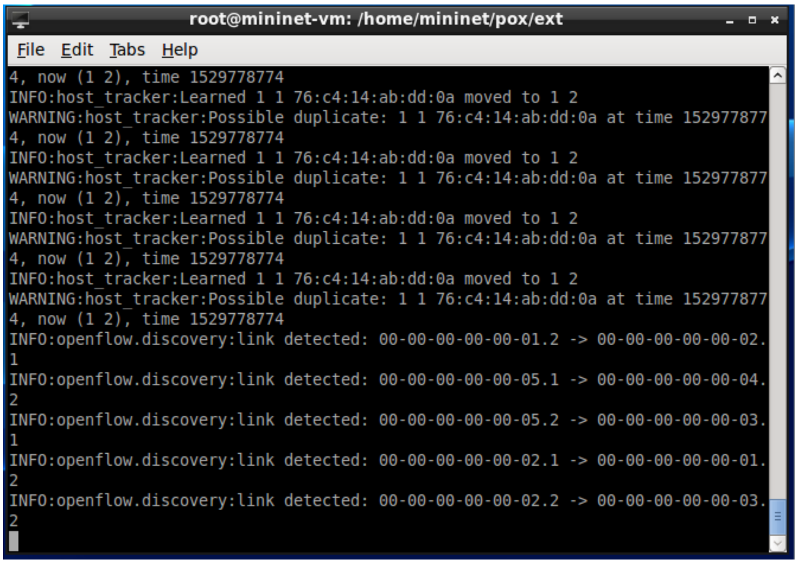


Εικόνα 22: Περιβάλλον Mininet-Εκτέλεση topology.py

Έπειτα στο δεύτερο terminal εκτελούμε το script ανίχνευσης της τοπολογίας με την εντολή:

../pox.py topoDiscovery openflow.discovery forwarding.l2_learning

Ο κώδικας εκτελείται:



Εικόνα 23: Περιβάλλον Mininet-Εκτέλεση topoDiscovery.py

Στο παράθυρο που εκτελείται η τοπολογία μας, δίνουμε την εντολή:

pingall

Ώστε να δημιουργήσουμε κίνηση στο δίκτυό μας, και να ανιχνευθούν όλοι οι hosts από το component host_tracker.

```
root@mininet-vm:/home/mininet/pox# sudo python type4_miniedit.py
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
h1 h2
*** Starting controllers
*** Starting switches
*** Post configure switches and hosts
*** Starting CLI:
mininet> link s1 s4 down
mininet> link s1 s4 up
mininet> link s1 s4 down
mininet> link s1 s4 up
mininet> link s1 s4 down
mininet> link s1 s4 up
mininet> pingall
*** Ping: testing ping reachability
```

Εικόνα 24: Δημιουργία κίνησης στο δίκτυο - pingall

Ανοίγοντας το JSON που έχει ενημερωθεί στο path /home/mininet/pox/ext/, θα δούμε τα παρακάτω αποτελέσματα:


```
{
    "hosts": {
        "h1": {
            "color": "red",
            "ip": "0.0.0.0",
            "mac": "be:02:a9:eb:40:2c"
        },
        "h2": {
            "color": "red",
            "ip": "0.0.0.0",
            "mac": "ea:59:3f:67:91:d7"
        }
    },
    "link": {
        "0000000001": {
            "connection": ["s3","s6"],
            "ports": [3, 1]
        },
        "0000000010": {
            "connection": ["s3","s5"],
            "ports": [1, 2]
        },
        "0000000011": {
            "connection": ["s2","s3"],
            "ports": [2, 2]
        },
        "0000000100": {
            "connection": ["s4","s5"],
            "ports": [2, 1]
        },
        "0000000101": {
```

```
            "connection": ["s1","s4"],
            "ports": [3, 1]
        },
        "0000000110": {
            "connection": ["s1","s2"],
            "ports": [2, 1 ]
        },
        "0000000111": {
            "connection": ["h1","s1"],
            "ports": [1, 1]
        },
        "0000001000": {
            "connection": ["h2","s6"],
            "ports": [1, 2]
        }
    },
    "switches": {
        "s1": "00:00:00:00:00:01",
        "s2": "00:00:00:00:00:02",
        "s3": "00:00:00:00:00:03",
        "s4": "00:00:00:00:00:04",
        "s5": "00:00:00:00:00:05",
        "s6": "00:00:00:00:00:06"
    }
}
```

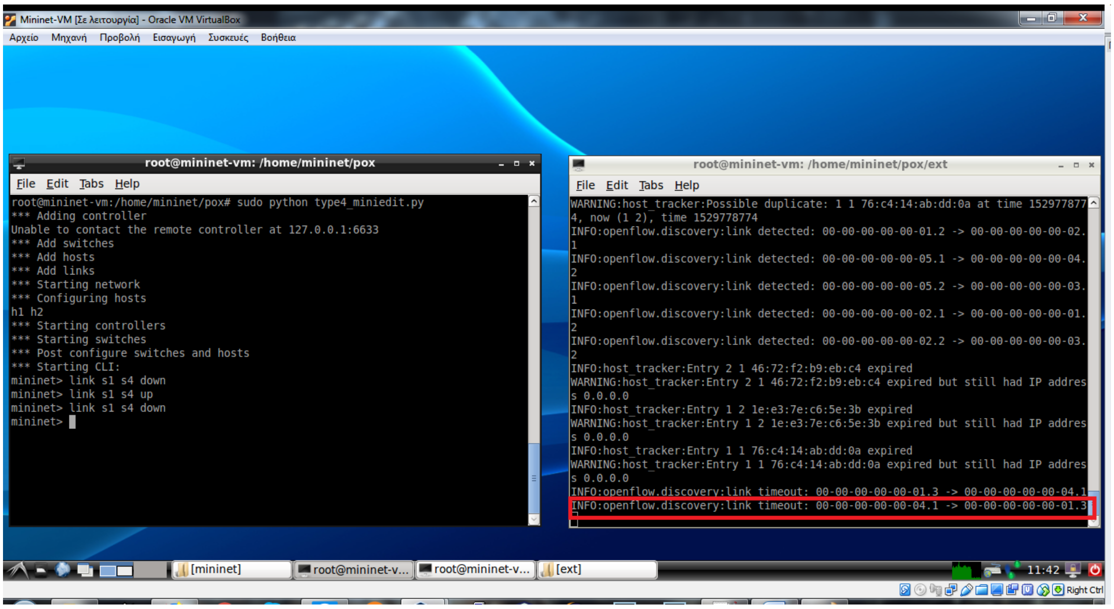
Απόσπασμα Κώδικα 18: JSON πλήρους τοπολογίας

Μπορούμε εύκολα να ελέγξουμε την ορθότητά τους, κοιτάζοντας απλά το script δημιουργίας της τοπολογίας μας.

Τώρα ας δοκιμάσουμε να ρίξουμε έναν σύνδεσμο μεταξύ δύο switches, και να δούμε αν το JSON θα ανανεωθεί σωστά, κι έπειτα να το ξαναπροσθέσουμε.

Η εντολή απενεργοποίησης ενός συνδέσμου, π.χ. ανάμεσα στο s1 και το s4, είναι:

link s1 s4 down



Εικόνα 25: Περιβάλλον Mininet-Πτώση συνδέσμου

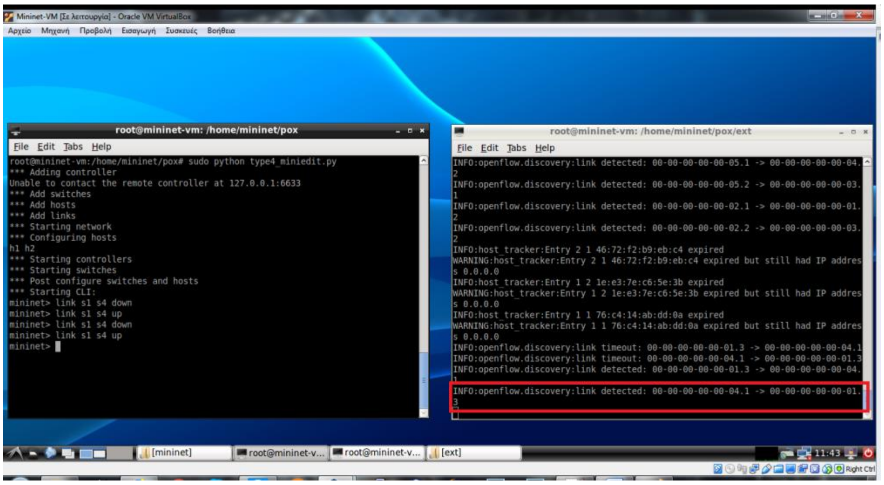
Μετά την πτώση του συνδέσμου ανάμεσα στα switches s1 και s4 το JSON ανανεώνεται:


```
"link": {
        "0000000001": {
            "connection": ["s3","s6"],
            "ports": [3, 1]
        },
        "0000000010": {
            "connection": ["s3","s5"],
            "ports": [1, 2]
        },
        "0000000011": {
            "connection": ["s2","s3"
            ],
            "ports": [2, 2]
        },
        "0000000100": {
            "connection": ["s4","s5"],
            "ports": [2, 1]
        },
        "0000000110": {
            "connection": ["s1","s2"],
            "ports": [2, 1]
        },
        "0000000111": {
            "connection": ["h1","s1"],
            "ports": [1, 1]
        },
        "0000001000": {
            "connection": ["h2","s6"],
            "ports": [1, 2]
        }
```

```
    }
```
Απόσπασμα Κώδικα 19: Πεδίο Link του JSON, μετά την πτώση συνδέσμου

Ενώ η εντολή ενεργοποίησης:

link s1 s4 up



Εικόνα 26: Περιβάλλον Mininet-Επαναφορά συνδέσμου

Μετά την επαναφορά του συνδέσμου ανάμεσα στα switches s1 και s4 το JSON ανανεώνεται:

```
    "link": {
        "0000000001": {
            "connection": ["s3","s6"],
            "ports": [3, 1]
        },
        "0000000010": {
            "connection": ["s3","s5"],
            "ports": [1, 2]
        },
        "0000000011": {
            "connection": ["s2","s3"],
            "ports": [2, 2]
        },
        "0000000100": {
            "connection": ["s4","s5"],
            "ports": [2, 1]
        },
        "0000000110": {
            "connection": ["s1","s2"],
            "ports": [2, 1]
        },
        "0000000111": {
            "connection": ["h1","s1"],
            "ports": [1, 1]
        },
        "0000001000": {
```

```
            "connection": ["h2","s6"],
            "ports": [1, 2]
        },
        "0000001001": {
            "connection": ["s1","s4"],
            "ports": [3, 1]
        }
    }
```
Απόσπασμα Κώδικα 20: Πεδίο Link του JSON, μετά την επαναφορά συνδέσμου

Συνεπώς, βλέπουμε πως αναλόγως με το αν κάποιο link «πέφτει» ή επανέρχεται, το JSON που κρατάει την τοπολογία μας ενημερώνεται άμεσα και σωστά.

6.4 Περιορισμοί Λύσης

Μπορεί η λύση αυτή να δουλεύει για τον αρχικό σκοπό που υλοποιήθηκε, ωστόσο περιέχει και κάποιους περιορισμούς που ίσως να βγουν στο προσκήνιο αν χρειαστεί να επεκταθεί περαιτέρω. Παρακάτω αναφέρουμε όσους γνωρίζαμε την ώρα που γραφόταν αυτή η εργασία:

- Σε περίπτωση πτώσης του link μεταξύ κάποιου host και κάποιου μεταγωγέα, η εφαρμογή δεν ενημερώνεται. Αυτό συμβαίνει γιατί σε τέτοια περίπτωση ούτε το component host_tracker, το οποίο χρησιμοποιείται για την ανίχνευση των hosts, αλλά ούτε και το openflow.discovery, το οποίο ανιχνεύει events σχετικά με τα links, καταφέρνουν να την ανιχνεύσουν. Το host_tracker περιορίζεται στο να ελέγχει αν κάποιος host που έχει ήδη καταγράψει συνεχίζει να είναι ενεργοποιημένος, ενώ το openflow.discovery ασχολείται με links που βρίσκονται μόνο ανάμεσα σε switches.
- Οι πληροφορίες που μπορεί να μάθει ο controller για τους hosts περιορίζονται στην διεύθυνση MAC, στην πόρτα σύνδεσης του host στο switch και στο DPID του switch αυτού. Αυτό έγκειται πάλι στους περιορισμούς που θέτει το host_tracker.
- Γίνεται να οριστούν link attributes (π.χ. bandwidth, type, name) μέσω του script ορισμού της τοπολογίας, αλλά το openflow.discovery δεν μπορεί να τα ανακτήσει. Όπως αναφέρθηκε και παραπάνω, οι τιμές που μπορεί να φέρει το openflow.discovery, περιορίζονται σε αυτά που αναγράφονται στον Πίνακας 1.

7. Σχετικές εργασίες

Δεδομένου πως το πρόβλημα που προσπάθησε να λύσει η συγκεκριμένη διπλωματική εργασία δεν είναι νέο, κρίνουμε σκόπιμο να αναφερθούμε σε κάποιες από τις εργασίες εκείνες που μελετήσαμε κατά τη διάρκεια προετοιμασίας αυτής της αναφοράς. Η αναφορά μας σε αυτές τις εργασίες, θα γίνει κατά χρονολογική σειρά, από την παλιότερη στην πιο πρόσφατη.

Το 2014, ο κος. Thomas Paradis, από το KTH Royal Institute of Technology της Στοκχόλμης, παρουσίασε την εργασία του με τίτλο «Sotware-Defined Networking» [76]. Μέσα από αυτή την εργασία, δημιούργησε ένα prototype το οποίο μεταξύ άλλων αναλάμβανε να ανιχνεύσει την τοπολογία ενός δικτύου και τις σχέσεις μεταξύ των κόμβων του, ώστε αργότερα να εφαρμόσει εύκολα κάποια πολιτική QoS. Για την υλοποίησή του, χρησιμοποίησε τον ελεγκτή POX.

Την ίδια χρονιά, ο κος. Χριστόφορος Ρεντίφης από το ΕΜΠ, παρουσίασε την εργασία του με τίτλο «Ανάπτυξη λογισμικού με χρήση ελεγκτή OpenFlow (OF controller) και υλοποίηση μηχανισμών δρομολόγησης πακέτων» [54]. Στα πλαίσια της εργασίας του, ανέπτυξε ένα λογισμικό το οποίο εφαρμόστηκε σε έναν ελεγκτή Beacon με σκοπό να επιτύχει 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 του φορτίου των συσκευών που δραστηριοποιούνται στο δίκτυο αλλά και των αποφάσεων του διαχειριστή του δικτύου. Για την υλοποίηση της, χρησιμοποιήθηκε το πρωτόκολλο OpenFlow. 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.

Το 2015, ο κος. Γιώργος Ταρναράς από το Πανεπιστήμιο Πατρών, στη διπλωματική εργασία του με τίτλο «Υλοποίηση και εκτέλεση αλγορίθμου βέλτιστης δρομολόγησης σε L2 δίκτυο με τεχνικές προγραμματιζόμενων δικτύων - SDN» [77], υλοποίησε έναν βέλτιστο αλγόριθμο για την ανακάλυψη της τοπολογίας ενός δικτύου, με τη χρήση του πρωτοκόλλου LLDP μέσω του OpenFlow. Η εφαρμογή του, ενημερώνεται από τα switches για τους γείτονές τους και δημιουργεί μία τοπολογία του δικτύου την οποία χρησιμοποιούν οι όποιοι αλγόριθμοι δρομολόγησης, χωρίς την ανάγκη να αποκόπτονται ζεύξεις για την αποφυγή κλειστών βρόχων. Τέλος, στην υλοποίησή του, βασίστηκε στο ForCES [32] Framework.

Την ίδια χρονιά, η κα. Ουρανία-Στέλλα Βουκελάτου από το ΕΜΠ, στην εργασία της με τίτλο «Ελεγκτής OpenFlow για Υπηρεσίες με Επίγνωση Περιβάλλοντος» [78], ανέπτυξε και παρουσίασε ένα μηχανισμό ο οποίος επιτύγχανε τη βέλτιστη δρομολόγηση δεδομένων στο δίκτυο με τη συνεχή εναλλαγή της λειτουργίας των συσκευών του δικτύου και την επίγνωση των στοιχείων περιβάλλοντος. Εδώ, η επιλογή της κατάλληλης διαδρομής γίνεται με βάση την τοπολογία και τις αποφάσεις του διαχειριστή του δικτύου. Για την υλοποίηση χρησιμοποιήθηκε το πρωτόκολλο OpenFlow και ο ελεγκτής Beacon. Τέλος, χρησιμοποιήθηκε η μέθοδος επικοινωνίας Anycast. Πρόκειται για μία μέθοδο δρομολόγησης πακέτων βασισμένη στην εύρεση της συντομότερης διαδρομής όταν πρόκειται για δρομολόγηση προς μία διεύθυνση IP, που την μοιράζεται μία συγκεκριμένη ομάδα συσκευών δικτύου.

Ένα χρόνο αργότερα, η κα. Δέσποινα Παληού από το ΕΜΠ, σε εργασία της με τίτλο «Εφαρμογή SDN για δυναμική δρομολόγηση σε Software Defined Networks» [79], αναπτύσσει μια εφαρμογή η οποία ελέγχει εξωτερικά τον ελεγκτή ενός SDN δικτύου και επαναπροσδιορίζει τη δρομολόγηση στο δίκτυο προκειμένου να επιτευχθεί δρομολόγηση με ενεργειακά βέλτιστο τρόπο. Για την επιλογή της δρομολόγησης λαμβάνονται υπόψη ενεργειακά μοντέλα των συσκευών καθώς και στατιστικά της κίνησης στις ροές του δικτύου, που προκύπτουν μετά από κατάλληλη παραμετροποίησή του μέσω του ελεγκτή. Ο ελεγκτής που επέλεξε για την υλοποίηση της λύσης της είναι ο OpenDaylight. Η τοπολογία που επιλέχθηκε προσομοιάζει τις τοπολογίες που επιλέγονται για τα Data Centers λόγω του μεγάλου ποσοστού ενέργειας που καταναλώνεται σ’ αυτά. Για την εκμάθηση της τοπολογίας του δικτύου καθώς και για την παραμετροποίησή του χρησιμοποιήθηκε το REST API του OpenDaylight, ενώ για την εύρεση της βέλτιστης δρομολόγησης χρησιμοποιήθηκε το λογισμικό CPLEX.

Λίγα χρόνια αργότερα, τον Μάιο του 2017, οι A. Azzouni, N. Thi Mai Trang, R. Boutaba και G. Pujolle από το Πανεπιστήμιο του Waterloo του Καναδά, δημοσιεύουν μια εργασία τους στο LIP6 / UPMC στο Παρίσι με τίτλο, «Limitations of OpenFlow Topology Discovery Protocol» [80]. Μέσα από αυτή την εργασία, φέρνουν στην επιφάνεια τα προβλήματα απόδοσης, τα κενά ασφαλείας και τους περιορισμούς που έχει το OFDP (OpenFlow Discovery Protocol), και παρουσιάζουν συνοπτικά ένα νέο πρωτόκολλο, το sOFDP, το οποίο υποστηρίζουν πως είναι πιο αποδοτικό και ασφαλές.

Βιβλιογραφία

[1] Dave Evans, "The Internet of Things-How the Next Evolution of the Internet," Cisco Internet Business Solutions Group (IBSG), White Paper 2011.

[2] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," IDC iView: IDC Anal. Future, 2007, 2012.

[3] S. Taylor, "The next generation of the Internet revolutionizing the way we work, live, play, and learn," CISCO, CISCO Point of View, 2013.

[4] L. Atzori, A. Iera, G. Morabito, "The Internet of Things: A survey," *Computer Networks*, 2010.

[5] Y. Li, X. Su, J. Riekki, T. Kanter, R. Rahmani, "A SDN-based architecture for horizontal Internet of Things services," Proc. IEEE International Conference on Communications, 2016.

[6] O. Flauzac, C. Gonzalez, A. Hachani, F. Nolot, "SDN based architecture for IoT and improvement of the security," University of Reims Champagne-Ardenne, CReSTIC Reims, 2015.

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

[8] Nikos Fotiou, Dimitrios Mendrinos, George C. Polyzos, "Edge-assisted Traffic Engineering and applications in the IoT," Mobile Multimedia Laboratory, Athens University of Economics and Business, Athens, 2018.

[9] i-TECH4u.gr. (2018, June) "Internet of Things σε απλά ελληνικά". [Online]. http://www.itech4u.gr/tech/hands-on/item/7262-internet-of-things-se-apla-ellinika/7262-internet-of-things-se-apla-ellinika

[10] Friedemann Mattern, Christian Floerkemeier, "From the Internet of Computers to the Internet of Things," Distributed Systems Group, Institute for Pervasive Computing, ETH, 2010.

[11] Γεωργία Ντοά, Μεταπτυχιακή Διπλωματική Εργασία: «Blockchain και η εφαρμογή του στο Internet of Things (IoT)», Τμήμα Ψηφιακών Συστημάτων, Πανεπιστήμιο Πειραιώς, 2017.

[12] OTS. (2018, June) "Μια λύση στο πλαίσιο υλοποίησης των Smart Cities". [Online]. http://blog.ots.gr/tag/internet-of-things/

[13] Tobby Simon, "Critical Infrastructure and the Internet of Things," Global Commission on Internet Governance, 2017.

[14] Internet Architecture Board (IAB) (2018, June) RFC7452 - "Architectural Considerations in Smart Object Networking". [Online]. https://tools.ietf.org/html/rfc7452

[15] i-scoop. (2018, June) "Industry 4.0: the fourth industrial revolution". [Online]. https://www.i-scoop.eu/industry-4-0/#Industrial_Data_Space_linking_IoT_and_smart_services_in_Industry_40_and_beyond

[16] Ankur Pipara. (2018, June) "Internet of things (IoT)". [Online].

https://www.slideshare.net/AnkurPipara/internet-of-things-iot-2014

[17] Yasir Saleem, Noel Crespi, Mubashir Husain Rehmani, Rebecca Copeland, "Internet of Things-aided Smart Grid: Technologies, Architectures, Applications, Prototypes, and Future Research Directions," IEEE, 2017.

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

[19] ONF, "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation, 2012.

[20] SDX Central. (2018, June) "Understanding the SDN Architecture – SDN Control Plane & SDN Data Plane". [Online]. https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/

[21] ONF, "SDN Architecture Overview," Open Networking Foundation, Version 1.0, 2013.

[22] Cozlink. (2018, June) "Basic Information about Software Defined Network (SDN) Architecture". [Online]. https://www.cozlink.com/modules-a272-275-273/article-73747.html

[23] David A Maltz et al., "Routing design in operational networks: A look from," ACM SIGCOMM Computer Communication Review, 2004.

[24] Avishai Wool, "A quantitative study of firewall configuration errors," *Computer* , 2004.

[25] Martin Casado et al., "Ethane: Taking control of the enterprise," ACM SIGCOMM Computer Communication Review, 2007.

[26] Martin Casado et al., "SANE: A protection architecture for enterprise networks," in USENIX Security Symposium, 2006.

[27] Bradley Mitchell. (2018, Apr.) "An Introduction to BYOD for IT Networks". [Online]. https://www.lifewire.com/an-introduction-to-byod-for-it-networks-817819

[28] Matthew Caesar et al., "Design and implementation of a routing control," in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, 2005.

[29] Evangelos Haleplidis, Spyros Denazis, Kostas Pentikousis, Jamal Hadi Salim, David Meyer and Odysseas Koufopavlou, "SDN Layers and Architectures Terminology," RFC7426, IRTF SDNRG research group, 2015.

[30] N. Feamster. (2014, July) "M1.2 SDN History: Programmable Networks". [Online]. https://www.youtube.com/watch?v=CFNTzni1FhA&list=PLpherdrLyny8YN4M24iRJBM

[31] N. Feamster. (2014, Sep.) "M1.3 SDN History: Network Virtualization". [Online]. https://www.youtube.com/watch?v=vsbyyNFg5BM&list=PLpherdrLyny8YN4M24iRJB

[32] N. Feamster. (2014, Sep.) "M1.4: SDN History: Control of Packet-Switched Networks". [Online]. https://www.youtube.com/watch?v=AFj-ZIuAGwo&list=PLpherdrLyny8YN4M24iRJBMCXkLcGbmhY&index=5

[33] Diego Kreutz, Fernando M. V. Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak

Azodolmolky and Steve Uhlig, "Software-Defined Networking: A Comprehensive Survey," in Proceedings of the IEEE, 2015.

[34] Learn Networking. (2018, June) "A Guide to Network Topology". [Online]. http://learn-networking.com/network-design/a-guide-to-network-topology

[35] Nick Feamster et al., "The case for separating routing from routers," in Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture, ACM, 2004.

[36] J Van der Merwe et al., "Dynamic connectivity management with an intelligent route service control point," in Proceedings of the 2006 SIGCOMM workshop on Internet network management. ACM, 2006.

[37] Marcelo R Nascimento et al., "Virtual routers as a service: the routeflow approach leveraging software-defined networks," in Proceedings of the 6th International Conference on Future Internet Technologies. ACM, 2011.

[38] Christian Esteve Rothenberg et al., "Revisiting routing control platforms with the eyes and muscles of software-defined networking," in Proceedings of the first workshop on Hot topics in software defined networks. ACM, 2012.

[39] Internet Engineering Task. (2018, June) "Interface to the Routing System (i2rs) Workgroup". [Online]. https://datatracker.ietf.org/wg/i2rs/documents/

[40] Siamak Azodolmolky, Philipp Wieder, and Ramin Yahyapour, "SDN-based cloud computing networking," in Transparent Optical Networks (ICTON), 2013 15th International Conference on. IEEE, 2013.

[41] OpenStack. (2018, June) "OpenStack Neutron project". [Online]. https://wiki.openstack.org/wiki/Neutron

[42] OpenDaylight. (2018, June) "OpenDaylight website". [Online]. https://www.opendaylight.org/

[43] Dmitry Drutskoy, Eric Keller, and Jennifer Rexford., "Scalable network virtualization in software-defined networks," IEEE Internet Computing, 2013.

[44] M Mahalingam et al. IETF. (2018, June) "VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks". [Online]. tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-06

[45] Ivan Pepelnjak. (2018, June) "Cloud Networking – From Theory to Practice, ipSpace.net / NIL Data Communications". [Online]. https://ripe64.ripe.net/presentations/21-Data_Center_Fabrics_-_What_Really_Matters_(RIPE).pdf

[46] Sushant Jain et al., "B4: Experience with a globally-deployed software defined WAN," in Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM. ACM., 2013.

[47] Dan Levin et al, "Toward Transitional SDN Deployment in Enterprise Networks," in Open Networking Summit, 2013.

[48] Nick McKeown et al., "OpenFlow: enabling innovation in campus networks," in ACM SIGCOMM Computer Communication Review, 2008.

[49] IEEE Station and Media Access Control Connectivity Discovery, "IEEE Standard 802.1 AB (LLDP)", 2009.

[50] Rivka Gewirtz Little. (2018, June) "Hybrid SDN is the gateway drug to the new network". [Online]. https://searchsdn.techtarget.com/feature/Hybrid-SDN-is-the-gateway-drug-to-the-new-network

[51] ONF. (2018, June) "Openflow Specification v.1.5.1". [Online]. http://www.opennetworking.org/images//openflow-switch-v1.5.1.pdf

[52] Saurav Das, Guru Parulkar, and Nick McKeown, "Why OpenFlow/SDN can succeed where GMPLS failed," in European Conference and Exhibition on Optical Communication, 2012.

[53] Martìn Casado. (2018, June) "List of OpenFlow Software Projects". [Online]. https://yuba.stanford.edu/~casado/of-sw.html

[54] Χριστόφορος Ρεντίφης, "Ανάπτυξη λογισμικού με χρήση ελεγκτή OpenFlow (OF controller)," Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα, 2014.

[55] Natasha Gude et al., "NOX: towards an operating system for networks," in ACM SIGCOMM Computer Communication Review, 2008.

[56] NOX. (2018, June) "NOX website". [Online]. https://github.com/noxrepo/

[57] Ruslan L Smeliansky and Alexander V Shalimov, "Advanced Study of SDN/OpenFlow controllers," Conference: Proceedings of the 9th Central & Eastern European Software, 2013.

[58] David Erickson, "The Beacon OpenFlow Controller," HotSDN. ACM, 2013.

[59] OSRG. (2018, June) "Component- Based Software Defined Networking Framework: Build SDN Agilely". [Online]. http://osrg.github.io/ryu/

[60] Koponen, T., M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R.Ramanathan et al., "Onix: A distributed control platform for largescale production networks Design and Implementation (OSDI ’10)," in Proceedings of the 9th USENIX Conferenceon Operating Systems , Berkeley, 2010.

[61] Beacon Wiki. (2018, June) "Beacon webpage". [Online]. https://openflow.stanford.edu/display/Beacon/Home

[62] David Erickson et al., "Optimizing a virtualized data center," ACM SIGCOMM Computer Communication Review, 2011.

[63] Floodlight Project. (2018, June) "Floodlight webpage". [Online]. www.projectfloodlight.org/floodlight/

[64] Sridhar Rao. (2018, June) "SDN Series Part Five: Floodlight, an OpenFlow Controller". [Online]. https://thenewstack.io/sdn-series-part-v-floodlight/

[65] ONF. (2018, June) "OpenFlow Switch Specification v1.0.0". [Online]. https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf

[66] W. Braun και M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," Future Internet, 2014.

[67] A. Sonba και H. Abdalkreim, "Performance Comparison Of the state of the art Openflow Controllers," School of Information Science, Computer and Electrical Engineering, Halmstad, 2014.

[68] Mininet. (2018, June) "Mininet Overview". [Online]. http://mininet.org/overview/

[69] Kspviswa. (2018, June) "OpenFlow Version RoadMap". [Online]. http://kspviswa.github.io/OpenFlow_Version_Roadmap.html

[70] ONF. (2018, June) "OpenFlow Switch Specification v1.1.0". [Online]. https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-spec-v1.1.0.pdf

[71] ONF. (2018, June) "OpenFlow Switch Specification v1.2". [Online]. https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-spec-v1.2.pdf

[72] ONF. (2018, June) "OpenFlow Switch Specification v1.3.0". [Online]. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[73] ONF. (2018, June) "OpenFlow Switch Specification". [Online]. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[74] Mininet Wiki. (2018, June) "Introduction to Mininet". [Online]. https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[75] Mininet Wiki. (2018, June) "Mininet Documentation". [Online]. https://github.com/mininet/mininet/wiki/Documentation

[76] Thomas Paradis, Master Thesis: "Software-Defined Networking," KTH Information and Communication Technology, Stockholm, 2014.

[77] Ταρναράς Γεώργιος, Διπλωματική Εργασία: "Υλοποίηση και εκτέλεση αλγορίθμου βέλτιστης δρομολόγησης σε L2 δίκτυο με τεχνικές προγραμματιζόμενων δικτύων – SDN," Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών, Πανεπιστήμιο Πατρών, 2015.

[78] Ουρανία Στέλλα Βουκελάτου, Διπλωματική Εργασία: "Ελεγκτής OpenFlow για Υπηρεσίες με Επίγνωση Περιβάλλοντος," Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Εθνικό Μετσόβιο Πολυτεχνείο, 2015.

[79] Παληού Δέσποινα, Διπλωματική Εργασία: "Εφαρμογή SDN για δυναμική δρομολόγηση σε Software Defined Networks," Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Εθνικό Μετσόβιο Πολυτεχνείο, 2016.

[80] Abdelhadi Azzouni, Nguyen Thi Mai Trang, Raouf Boutaba, and Guy Pujolle, "Limitations of OpenFlow Topology Discovery Protocol," 1LIP6 / UPMC, Paris, 2017.

[81] Xuyansen. (2018, June) "Discovery Topology in POX". [Online]. http://xuyansen.work/discovery-

topology-in-mininet/

[82] Γκρέμος Βασίλειος, Μπενέκος Σπυρίδων, Διπλωματική Εργασία: "Internet of Things: Constrained Application Protocol over Information-Centric Networking," Οικονομικό Πανεπιστήμιο Αθηνών, Αθήνα, 2017.

[83] Παπαγρίβας Ελευθέριος & Φραγκουλάκης Γεώργιος, Πτυχιακή Εργασία: "To Internet των Πραγμάτων - Internet of Things," Τμήμα Μηχανικών Πληροφορικής, ΤΕΙ Δυτικής Ελλάδος, 2016.

[84] Ιωάννης Χριστοδουλόπουλος, Πτυχιακή Εργασία: "Software Defined Networking," Τμήμα Μηχανικών Πληροφορικής, ΤΕΙ Δυτικής Ελλάδος, 2016.

[85] Ηλίας Κ. Σχισμένος, Διπλωματική Εργασία: "SDN τεχνολογία, τα δίκτυα στην νέα εποχή," Χαροκόπειο Πανεπιστήμιο Αθηνών, Τμήμα Πληροφορικής και Τηλεματικής, 2016.

[86] Λούνγκου Βασίλε, Πτυχιακή Εργασία: "Software Defined Networking," Τμήμα Μηχανικών Πληροφορικής, ΤΕΙ Δυτικής Ελλάδος, 2016.

[87] OpenFlow Wiki-Standford. (2018, June) "OpenFlow Wiki (Standford)". [Online]. https://openflow.stanford.edu/display/ONL/POX+Wiki

[88] OpenFlow Wiki-GitHub. (2018, June) "OpenFlow Wiki (GitHub)". [Online]. https://noxrepo.github.io/pox-doc/html/