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

Συλλογή Πληροφοριών σε Δίκτυα Οριζόμενα από Λογισμικό

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Του

Μηνάς Κ Τράττου

Επιβλέπων : Συμεών Παπαβασιλείου

Καθηγητής Ε.Μ.Π

Αθήνα, Ιανουάριος, 2021

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

Συλλογή Πληροφοριών σε Δίκτυα Οριζόμενα από Λογισμικό

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Του

Μηνάς Κ Τράττου

Επιβλέπων : Συμεών Παπαβασιλείου

Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 12η Ιανουαρίου 2021.

.....	.....	.....
Συμεών Παπαβασιλείου Καθηγητής Ε.Μ.Π	Θεοδώρα Βαρβαρίγου Καθηγήτρια Ε.Μ.Π	Ιωάννα Ρουσσάκη Καθηγήτρια Ε.Μ.Π

Αθήνα, Ιανουάριος, 2021

.....

Μηνάς Τράττου

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

Περίληψη

Η παρακολούθηση δικτύων τη σύγχρονη εποχή, είναι ένας σημαντικός τομέας για την υγεία ενός δικτύου καθώς με την ανάπτυξη της τεχνολογίας περισσότεροι παροχείς υπηρεσιών διαδικτύου (Internet Service Providers) εξελίσσονται ώστε να μεταδίδουν εικόνα ή ακόμα και βίντεο σε πραγματικό χρόνο. Γι' αυτό το λόγο οι χειριστές των δικτύων χρειάζονται πλέον να έχουν ανά πάσα στιγμή μια ενημερωμένη εικόνα του συνολικού δικτύου ώστε να εξασφαλίσουν την ποιότητα εξυπηρέτησης για τέτοιες εφαρμογές. Είναι δηλαδή απαραίτητο να συλλέγουν δεδομένα όπως εύρος ζώνης, ποσοστό απώλειας πακέτων ή καθυστέρηση ζεύξης για να υπολογίζουν την κατάσταση του δικτύου προκειμένου να είναι σε θέση να επιλύσουν άμεσα τυχών προβλήματα που προκύπτουν ή ακόμα να τα χρησιμοποιήσουν για να βελτιστοποιήσουν τις ζεύξεις του δικτύου. Επομένως είναι αναγκαία η εξέλιξη της παρακολούθησης δικτύων ώστε να παρέχεται μια πιο ακριβείς εικόνα για την κυκλοφορία δεδομένων σε αυτά.

Στην παρούσα διπλωματική χρησιμοποιούμε δίκτυα οριζόμενα από λογισμικό (Software Defined Networks, SDN) ως λύση στο πιο πάνω πρόβλημα για την συλλογή των απαραίτητων μεγεθών σε τοπολογίες δικτύων. Ενώ υπάρχουν ήδη μερικές μέθοδοι συλλογής των πιο πάνω στατιστικών κάποιες από αυτές δεν είναι σε θέση να υπολογίσουν όλα τα στοιχεία που χρειάζεται με την μέθοδο που αναλύουν, ενώ άλλες επιβάλλουν επιπλέον εγκαταστάσεις στα δίκτυα για την λειτουργία τους έχοντας επιπρόσθετα έξοδα. Επιπρόσθετα πολλές από αυτές δεν έχουν την απαιτούμενη ακρίβεια στις μετρήσεις τους, δεν μπορούν να εφαρμοστούν σε όλων των ειδών δίκτυα ή ακόμα επιβαρύνουν τα δίκτυα με μεγάλο φόρτο προκειμένου να πετύχουν τον σκοπό τους.

Η παρακολούθηση δικτύων οριζόμενων από λογισμικό είναι ένα θέμα που δεν έχει αναπτυχθεί σε μεγάλο βαθμό ακόμα κυρίως λόγω του ότι η ιδέα των δικτύων οριζόμενων από λογισμικό συνεχίζει ακόμα να εξελίσσεται. Με την χρήση όμως των SDN έχει γίνει δημοφιλής και η χρήση του πρωτοκόλλου Openflow καθώς με τα χαρακτηριστικά του καθιστά ευκολότερη και ακριβέστερη την μέτρηση των εν λόγω δεδομένων στα δίκτυα. Στην παρούσα διπλωματική αναπτύσσουμε τις μεθόδους παρακολούθησης δικτύων που υπάρχουν για τα παραδοσιακά καθώς και για τα SDN δίκτυα. Στην συνέχεια προτείνουμε μία δική μας μέθοδο για τον υπολογισμό του ποσοστού απώλειας πακέτων, εύρους ζώνης και καθυστέρησης χρησιμοποιώντας τα χαρακτηριστικά του πρωτόκολλου Openflow και τέλος σχολιάζουμε την ακρίβεια των αποτελεσμάτων μας σε σχέση με τα αποτελέσματα γνωστών μεθόδων υπολογισμού τους.

Λέξεις κλειδιά

Δίκτυα οριζόμενα από λογισμικό, Πρωτόκολλο Openflow, Παρακολούθηση δικτύων, Συλλογή δεδομένων, Διαχειριστής Openflow, Delay, Bandwidth, Packet Loss

Abstract

Network monitoring in modern times has become more and more important as with the development of technology more internet service providers are evolving to transmit image or even video in real time. Thus, network operators now need to have an up-to-date view of the entire network at all times to ensure the quality of service for such applications. That is, it is necessary to collect data such as throughput, packet loss or delay to calculate the state of the network in order to be able to immediately solve any problems that arise or even use them to optimize the network's connections. It is therefore vital for network monitoring to evolve in order to provide a more accurate picture of data traffic in networks.

In this thesis we use software defined networks as a solution to the above problem for collecting the necessary metrics in network topologies. While there are already some methods of collecting the above statistics, some of them are not able to calculate all the data needed with the method they analyze, while others impose additional installations on the network for their operation needing additional costs. Moreover, many of them do not have the required accuracy with their measurements, they cannot be applied to all types of networks or they even burden the networks with a heavy load in order to achieve their purpose.

Network monitoring with Software Defined Networking architecture is an issue that has not yet been greatly developed mainly due to the fact that the idea of software defined networks is still in the early stages of development. However, with the use of SDN, the use of the Openflow protocol has also become popular, as its features make the measurement of the necessary data easier and more accurate. In this thesis we discuss the network monitoring methods that exist for both traditional and SDN networks. Then we propose our own method for calculating packet loss, throughput and delay using the features of the Openflow protocol and finally we comment on the accuracy of our results in relation to the results of traditional non-SDN methods.

Keywords

Software Defined Networks, Openflow Protocol, Network Monitoring, Data Collection, Openflow Controller, Delay, Bandwidth, Packet Loss

Ευχαριστίες

Η πραγματοποίηση της παρούσας διπλωματικής δεν θα ήταν εφικτή χωρίς τη βοήθεια και την επίβλεψη του καθηγητή ΕΜΠ, κύριου Συμεών Παπαβασιλείου, ιδιαίτερα κατά την διάρκεια της δύσκολης περιόδου στην οποία αυτή πραγματοποιήθηκε. Ακόμη, οφείλω οπωσδήποτε να ευχαριστήσω θερμά την Δρ. Αδαμαντία Στάμου, μεταδιδακτορική ερευνήτρια στο εργαστήριο NETMODE (NETwork Management and Optimal Design Laboratory) και Γρηγόρη Κακκάβα, υποψήφιο διδάκτορα και ερευνητικό συνεργάτη στο εργαστήριο NETMODE, για το θέμα της διπλωματικής που μου πρότειναν και την υποστήριξή τους σε όλη τη διάρκεια της διπλωματικής εργασίας. Αφού η διπλωματική αυτή σηματοδοτεί το τέλος της πενταετούς φοίτησης μου στο Εθνικό Μετσόβιο Πολυτεχνείο θα ήθελα να ευχαριστήσω την οικογένεια μου για την στήριξη και την αγάπη που μου δείχνουν όλα αυτά τα χρόνια. Τέλος, ένα μεγάλο ευχαριστώ στους φίλους μου Νεόφυτο, Σταύρο, Ειρήνη, Σάββα, Λένο, Μαρία που ήταν πάντα στο πλάι μου καθ' όλη την διάρκεια των φοιτητικών μου χρόνων αφού τους έκαναν μια από τις ομορφότερες περιόδους της ζωής μου.

Πίνακας περιεχομένων

Πίνακας εικόνων

Κατάλογος πινάκων

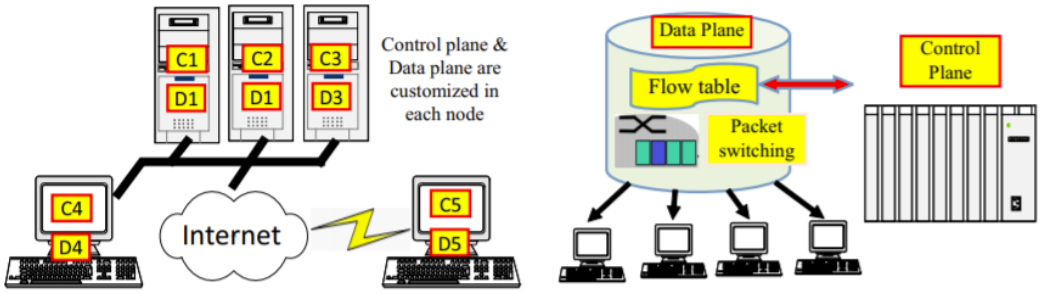
Κεφάλαιο 1 Εισαγωγή

1.1 Πρόλογος

Η παρακολούθηση δικτύων είναι ένα πολύ σημαντικό κομμάτι όσο αφορά την υγεία ενός δικτύου εφόσον παρέχει πληροφορίες για το δίκτυο συνεχώς ενώ βελτιώνει την ροή δεδομένων σε αυτό. Με την παρακολούθηση δικτύων εξασφαλίζουμε την διαθεσιμότητα και την βελτίωση της απόδοσης του εφόσον συγκεντρώνουμε σημαντικά στοιχεία για να την ανάπτυξή του στο μέλλον. Όπως αναφέρετε και στο άρθρο [1] η παρακολούθηση δικτύων μπορεί να συγκριθεί με μια επίσκεψη σε έναν καρδιολόγο όπου με την χρήση σύγχρονης τεχνολογίας εξάγουμε δεδομένα που μας πληροφορούν για την λειτουργικότητα του συστήματος. Όπως η διάγνωση ενός καρδιολόγου πρέπει να ληφθεί υπόψη έγκαιρα ώστε να αποφευχθούν σοβαρά προβλήματα έτσι και με τα δίκτυα χρειάζεται η 24ωρη παρακολούθηση τους ούτως ώστε να εξασφαλιστεί η παροχή υπηρεσιών τους. Όμως η ραγδαία ανάπτυξη της τεχνολογίας φέρει εις πέρας και πολλά εμπόδια που καθιστούν το έργο παρακολούθησης δικτύων ακόμα πιο δύσκολο.

1.2 Κίνητρο (motivation)

Μία λύση που μπορεί να βελτιώσει σημαντικά την παρακολούθηση δικτύων είναι η καινούρια τεχνολογία SDN (Software Defined Networking), μια αρχιτεκτονική δικτύων που αλλάζει ριζικά τη μορφή των δικτύων που γνωρίζαμε έως τώρα. Η αλλαγή που παρουσιάζει η αρχιτεκτονική αυτή είναι η έννοια του λογικού κεντρικού διαχειριστή δικτύου ο οποίος έχει την δυνατότητα να κρατά πληροφορίες για την κατάσταση του δικτύου και να επεξεργάζεται την κατάσταση αυτή. Ο κεντρικός διαχειριστής χρησιμοποιεί διάφορα πρωτόκολλα με το πιο δημοφιλή να είναι το Openflow με τα οποία προγραμματίζει τους μεταγωγείς του δικτύου ώστε να ακολουθούν συγκεκριμένα κριτήρια για την επεξεργασία και προώθηση της κυκλοφορίας. Έτσι οι κανόνες μετάδοσης δεδομένων σε ένα δίκτυο καθορίζονται από λογισμικό και όχι από υλικό, δίνοντας την δυνατότητα στον διαχειριστή να έχει περισσότερο έλεγχο της κυκλοφορίας στο δίκτυο αυξάνοντας ενδεχομένως σε σημαντικό βαθμό την απόδοση του. Η χρήση λοιπόν της καινούριας αυτής τεχνολογίας είναι σημαντικό να υιοθετηθεί από την σημερινή βιομηχανία εφόσον με τις εκσυγχρονισμένες μεθόδους διαχείρισης των δεδομένων ενός δικτύου το SDN μπορεί να βελτιώσει την απόδοση σύγχρονων τεχνολογιών όπως είναι για παράδειγμα το υπολογιστικό νέφος (cloud computing).



Εικόνα 1: Σύγκριση παραδοσιακών και SDN δικτύων [2]

1.3 Συνεισφορά (contribution)

Σκοπός της παρούσας αναφοράς είναι να χρησιμοποιήσουμε την καινούρια τεχνολογία που αποτελεί η αρχιτεκτονική SDN προκειμένου να πετύχουμε πιο αποτελεσματικές και αξιόπιστες μεθόδους παρακολούθησης δικτύων. Με τις ιδιότητες που προσφέρει η αρχιτεκτονική SDN, επιλύει πολλά προβλήματα που υπάρχουν στον τομέα της παρακολούθησης δικτύων όπως την αδυναμία άμεσης συλλογής δεδομένων με ικανοποιητική ακρίβεια ή την παρακολούθηση δικτύων χωρίς την μεγάλη επιβάρυνσή τους με όγκο δεδομένων που απαιτείται για την συλλογή των δεδομένων αυτών. Μέχρι τώρα αναπτύχθηκαν αρκετά εργαλεία που προσπαθούν να εκμεταλλευτούν τις ιδιότητες αυτές των SDN δικτύων μερικά από τα οποία θα αναφέρουμε και στην συνέχεια της παρούσας διπλωματικής. Χρησιμοποιώντας λοιπόν την αρχιτεκτονική SDN θα αναπτύξουμε και εμείς μια προτεινόμενη λύση σκοπός της οποίας είναι να παρέχει στον χρήστη με ευκολότερο και απλούστερο τρόπο συγκριτικά με τα εργαλεία που θα αναφέρουμε, τα στοιχεία που χρειάζεται για την ολοκληρωμένη παρακολούθηση δικτύου. Συγκεκριμένα θα αναπτύξουμε έναν κεντρικό διαχειριστή χρησιμοποιώντας την πλατφόρμα ανοιχτού κώδικα POX ο οποίος εκμεταλλευόμενος τις ιδιότητες που παρέχει το πρωτόκολλο Openflow καταγράφει και υπολογίζει σημαντικά μεγέθη για την παρακολούθηση ενός δικτύου. Η υλοποίησή μας δηλαδή είναι σε θέση να υπολογίσει με εύκολο τρόπο στατιστικά όπως είναι η καθυστέρηση, το ποσοστό χαμένων πακέτων και το εύρος ζώνης μιας ζεύξης, μετρήσεις που είναι πολύ σημαντικές για την ανάλυση τις κατάστασης ενός δικτύου. Τα πειράματα που διεξήγαμε πραγματοποιήθηκαν χρησιμοποιώντας την πλατφόρμα προσομοίωσης δικτύων mininet κατασκευάζοντας διάφορες τοπολογίες παρόμοιες με αυτές που θα χρησιμοποιούνταν σε ένα κέντρο δεδομένων.

Κεφάλαιο 2 Παρακολούθηση Δικτύων

Ένας σημαντικός τομέας στην διαχείριση δικτύων είναι η συλλογή δεδομένων (network monitoring), η συλλογή δηλαδή στατιστικών που αφορούν τις συνδέσεις των συσκευών σε όλο το δίκτυο. Σε ένα δίκτυο είναι απαραίτητη η γνώση της κατάστασης κάθε σύνδεσης μεταξύ κόμβων ανά πάσα στιγμή ούτως ώστε να ανιχνεύονται και να λύνονται άμεσα τυχόντα προβλήματα επικοινωνίας μεταξύ χρηστών. Στο παρελθόν οι διαχειριστές των δικτύων χρειάζονταν να παρακολουθούν ένα μικρό αριθμό υπολογιστών και είχαν να κάνουν με συνδέσεις που δεν ξεπερνούσαν τα 100 Mbps σε bandwidth. Στην σύγχρονη εποχή όμως η πολυπλοκότητα και οι διαστάσεις των δικτύων έχουν εξελιχτεί σε μεγάλο βαθμό τόσο με τις υψηλότερες ενσύρματες συνδέσεις όσο και με τις ασύρματες επικοινωνίες. Επομένως η παρακολούθηση δεδομένων στα δίκτυα είναι πλέον απαραίτητο να γίνεται με τον βέλτιστο τρόπο ούτως ώστε να διατηρείται σταθερή η απόδοσή τους ή για να εξασφαλιστεί η ασφάλεια του δικτύου.

Η παρακολούθηση δικτύων εμπλέκει την χρήση πολλαπλών μεθόδων με σκοπό την διατήρηση της ασφάλειας και ακεραιότητας στο εσωτερικό του δικτύου. Περιλαμβάνει τον έλεγχο λογισμικού, υλικού ή άλλων αδυναμιών που μπορεί να θέσουν σε κίνδυνο την «υγεία» του δικτύου. Η παρακολούθηση δικτύων είναι δύσκολο και απαιτητικό έργο που είναι ζωτικής σημασίας για έναν διαχειριστή δικτύου εφόσον οι διαχειριστές προσπαθούν συνεχώς να κρατήσουν ομαλή των λειτουργία του δικτύου καθώς η κατάρρευση του θα ήταν καταστροφική για την οποιαδήποτε επιχείρηση. Προκειμένου λοιπόν να είναι προληπτικοί και όχι αντιδραστικοί, οι διαχειριστές πρέπει να παρακολουθούν την κίνηση και την απόδοση της κυκλοφορίας σε όλο το δίκτυο και να επαληθεύουν ότι δεν υπάρχουν παραβιάσεις ασφαλείας εντός του δικτύου. Μέχρι τώρα υπάρχουν πολλοί μέθοδοι προσέγγισης της παρακολούθησης δικτύων με διαφορετικές αρχιτεκτονικές και ιδιότητες.

2.1 Βασικοί Ορισμοί

Πρωτόκολλο ICMP

Το ICMP (Internet Control Message Protocol, ή Πρωτόκολλο Ελέγχου Μηνυμάτων Διαδικτύου) [3] είναι ένα σημαντικό στοιχείο που χρησιμοποιείται από το πρωτόκολλο IP (Internet Protocol) για σκοπούς διαχείρισης δικτύου. Ονομάζεται και πρωτόκολλο ελέγχου (Control Protocol) αφού δεν χρησιμοποιείται μόνο για μεταφορά δεδομένων αλλά μεταφέρει επίσης στοιχεία για την κατάσταση του δικτύου. Το πρωτόκολλο ICMP χρησιμοποιείται κυρίως από την εντολή ping για τα πακέτα τύπου Echo Request και Echo Reply.

Πρωτόκολλα TCP και UDP

Τα πρωτόκολλα TCP (Transmission Control Protocol, Πρωτόκολλο Ελέγχου Μεταφοράς) και UDP (User Datagram Protocol, Πρωτόκολλο Δεδομενογράμματος Χρήστη) είναι πρωτόκολλα που χρησιμοποιούνται για μεταφορά δεδομένων. Το TCP είναι το κύριο πρωτόκολλο στα δίκτυα TCP/IP με το IP πρωτόκολλο να επεξεργάζεται πακέτα δεδομένων ενώ το TCP να επιτρέπει σε δύο κεντρικούς υπολογιστές να ανταλλάζουν ροές πακέτων δεδομένων δημιουργώντας παράλληλα σύνδεση μεταξύ τους. Ακόμα το πρωτόκολλο TCP εγγυείται την παραλαβή πακέτων με την ίδια σειρά με την οποία τα έχει στείλει ο αποστολέας αφού χρησιμοποιεί χειραψίες (handshaking) μεταξύ των υπολογιστών για να δημιουργήσει επικοινωνία μεταξύ τους.

Αντίθετα το πρωτόκολλο UDP θυσιάζει αξιοπιστία για ταχύτητα. Προσφέρει αναξιόπιστη, «ελάχιστη προσπάθεια» μεταφοράς δεδομένων σε πρωτόκολλα και εφαρμογές ανώτερου επιπέδου. Σε αντίθεση με το TCP δεν δημιουργεί μια μόνιμη σύνδεση μεταξύ αποστολέα παραλήπτη αλλά απλά συσκευάζει και μεταφέρει τα δεδομένα που χρειάζεται να μεταφερθούν.

2.2 Ενεργητικές και παθητικές μέθοδοι

Υπάρχουν αρκετοί τρόποι παρακολούθησης δικτύων. Οι μέθοδοι αυτοί χωρίζονται κυρίως σε ενεργητικές (active) και παθητικές (passive).

Με τις ενεργητικές μεθόδους συλλογής δεδομένων παράγονται και στέλνονται επιπλέον πακέτα στο δίκτυο προκειμένου να υπολογιστούν τα στατιστικά των ζεύξεων που χρειάζονται σε κάθε περίπτωση. Ένα απλό παράδειγμα μιας ενεργητικής μεθόδου συλλογής δεδομένων είναι η γνωστή εντολή *ping,* η οποία χρησιμοποιώντας ICMP πακέτα, μπορεί να υπολογίσει αποτελεσματικά την κατάσταση της σύνδεσης σημείων από άκρη σε άκρη (end-to-end) στο δίκτυο καθώς και τον χρόνο μετ' επιστροφής (round-trip time) πακέτων συγκεκριμένης διαδρομής σ' αυτό. Ενώ οι ενεργητικές μέθοδοι είναι χρήσιμες για την συλλογή δεδομένων για ένα δίκτυο, με τον επιπλέον όγκο πακέτων που παράγουν είναι πιθανό να επιβαρύνουν το δίκτυο και ως αποτέλεσμα να επηρεάσουν αρνητικά την ακρίβεια των μετρήσεών του.

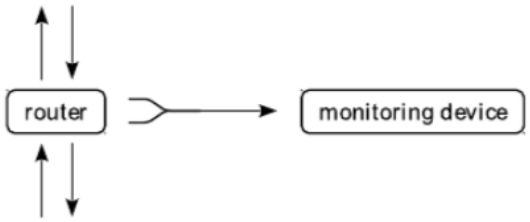
Αντίθετα οι παθητικές μέθοδοι συλλογής δεδομένων στα δίκτυα βασίζονται στην εγκατάσταση ξένων εργαλείων μέσα στο δίκτυο, τα οποία υπολογίζουν τα απαραίτητα στατιστικά που χρειάζονται σε κάθε περίπτωση παρακολουθώντας παθητικά την κίνηση πακέτων μέσα στο δίκτυο. Το πλεονέκτημα των παθητικών μεθόδων σε σχέση με τις ενεργητικές είναι ότι δεν παράγουν επιπλέον όγκο πακέτων που μπορεί να επηρεάσει την ακρίβεια των μετρήσεών τους. Όμως η εγκατάσταση των εργαλείων στο δίκτυο συνήθως απαιτεί μεγάλη επένδυση γεγονός που καθιστά τις παθητικές μεθόδους μη εφικτές για όλα τα δίκτυα.

2.3 Τεχνικές παρακολούθησης δικτύων

2.3.1 Αντιγραφή κυκλοφορίας

Μια κοινή ιδιότητα όλων των ειδών παρακολούθησης δικτύων είναι η αντιγραφή κυκλοφορίας (Traffic duplication), η δημιουργία δηλαδή αντιγράφου των διερχόμενων πακέτων μίας ακμής με σκοπό την ανάλυση του. Η μέθοδος αυτή μπορεί να πραγματοποιηθεί με δύο τρόπους, είτε με συσκευή αντιγραφής στην γραμμή (in line) της κυκλοφορίας είτε με τη μέθοδο κατοπτρισμού (mirroring). Υπάρχουν διάφοροι τρόποι εφαρμογής της μεθόδου αντιγραφής κυκλοφορίας, ο κατοπτρισμός θυρών, η χρήση συσκευής TAP (Test Access Port) ή η χρήση κάρτας διεπαφής δικτύου (Network Interface Card, NIC).

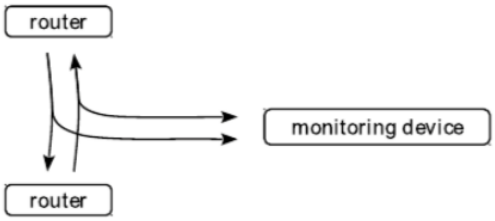
Ο κατοπτρισμός θυρών είναι μια λειτουργία συνήθως διαθέσιμη στους μεταγωγείς ενός δικτύου. Με την μέθοδο αυτή η κυκλοφορία μέσα από συγκεκριμένες θύρες μεταγωγέων αντιγράφεται σε μία δεύτερη θύρα που ονομάζεται θύρα κατοπτρισμού με σκοπό να μεταφερθεί στην συσκευή ανάλυσης δεδομένων για το δίκτυο. Και οι δύο κατεύθυνσης μετάδοσης του μεταγωγέα αντιγράφονται σε μία μόνο θύρα για την ανάλυση τους. Υπάρχουν όμως μερικά μειονεκτήματα στη χρήση της συγκεκριμένης μεθόδου. Σε περίπτωση που ο όγκος κυκλοφορίας είναι μεγαλύτερος από το μέγεθος που μπορεί να μεταδώσει η θύρα κατοπτρισμού τότε η θύρα κατοπτρισμού συνωστίζεται με αποτέλεσμα να απορρίπτει πακέτα. Επιπρόσθετα οι περισσότεροι μεταγωγείς δεν έχουν την υπολογιστική ισχύ ώστε να μπορούν να αντέξουν τον κατοπτρισμό και την μετάδοση πακέτων. Μιας και η μετάδοση πακέτων είναι ο κύριος ρόλος τους, σε περίπτωση μεγάλου όγκου κυκλοφορίας ο κατοπτρισμός ίσως δεν εφαρμόζεται σωστά.



Εικόνα 2: Εφαρμογή κατοπτρισμού θύρας [3]

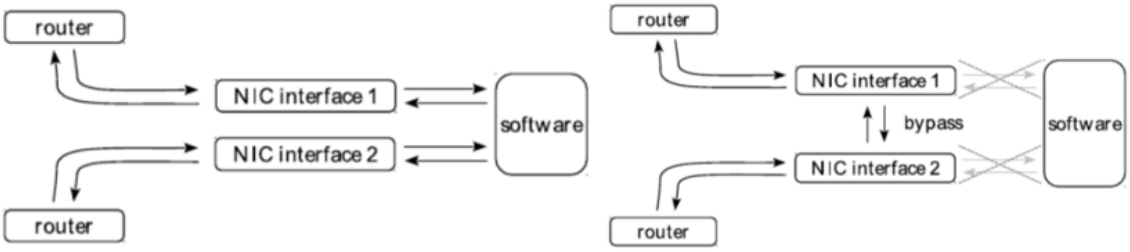
Η συσκευή TAP είναι μια συσκευή συλλογής πακέτων που τοποθετείται πάνω στη γραμμή κυκλοφορίας χωρίζοντάς την στα δύο για την συλλογή των δεδομένων από την συσκευή παρακολούθησης. Στο σημείο του διαχωρισμού της γραμμής η κυκλοφορία αντιγράφεται και σε αντίθεση με την προηγούμενη μέθοδο, σε αυτή τη περίπτωση οι δύο κατευθύνσεις κυκλοφορίας του μεταγωγέα μεταφέρονται στην συσκευή παρακολούθησης ξεχωριστά. Υπάρχουν διάφορα είδη συσκευών TAP που χωρίζονται κυρίως σε ενεργητικές και παθητικές. Κύριο πλεονέκτημα των παθητικών συσκευών σε σχέση με τις ενεργητικές είναι το γεγονός ότι σε περίπτωση οποιασδήποτε

διακοπής ρεύματος στην περίπτωση των παθητικών TAP δεν επηρεάζεται η γραμμή κυκλοφορίας του δικτύου ενώ στις ενεργητικές συσκευές TAP διακόπτεται η λειτουργία ολόκληρου του μεταγωγέα. Από την άλλη όμως η χρήση των παθητικών συσκευών περιορίζει σημαντικά τον ρυθμό μετάδοσης των αντιγραμμένων δεδομένων σε σχέση με τις ενεργητικές συσκευές.



Εικόνα 3: Εγκατάσταση συσκευής TAP [3]

Με την εγκατάσταση κάρτας διεπαφής δικτύου, συνδέονται οι μεταγωγείς μίας γραμμής με λογισμικό που τοποθετείται πάνω στη γραμμή με σκοπό να αναλύει τα πακέτα κυκλοφορίας που περνάνε μέσω αυτού. Η χρήση της συγκεκριμένης μεθόδου όμως συστήνει και ακόμα ένα σενάριο αποτυχίας κυκλοφορίας εφόσον σε περίπτωση βλάβης του λογισμικού διακόπτεται ολόκληρη η σύνδεση των μεταγωγέων. Υπάρχουν όμως λύσεις του συγκεκριμένου προβλήματος που επιτρέπουν την μεταφορά των πακέτων διαμέσου των καρτών διεπαφών δικτύου (by pass) σε περίπτωση τέτοιας βλάβης.



Εικόνα 4: Χρήση κάρτας διεπαφής δικτύου με λειτουργία by pass και χωρίς [3]

2.3.2 Παρακολούθηση ροών

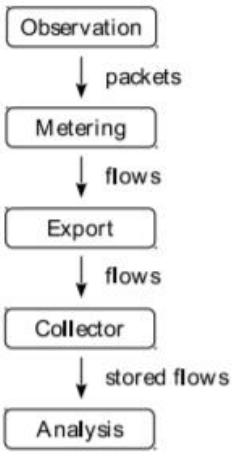
Με την μέθοδο παρακολούθησης ροών τα δεδομένα των πακέτων δεν αναλύονται σε επίπεδο κάτω από τις επικεφαλίδες τους αλλά η πληροφορία συγκεντρώνεται συνολικά για κάθε ροή. Ο ορισμός μιας ροής όπως αναφέρει και η πηγή **[4]** είναι «ένα σύνολο πακέτων που περνούν από ένα συγκεκριμένο σημείο παρακολούθησης ενός δικτύου σε κάποιο χρονικό περιθώριο». Συγκεκριμένα τα στοιχεία που χρησιμοποιούνται προκειμένου να ξεχωρίζουν τα πακέτα μεταξύ ροών είναι η

διευθύνσεις και θύρες του προορισμού και της πηγής καθώς και το πρωτόκολλο επίπεδου 4 (επίπεδο μεταφοράς) ενός πακέτου.

Με την μέθοδο παρακολούθησης ροών η συσκευή παρακολούθησης της κυκλοφορίας καταγράφει τα πιο πάνω δεδομένα που ορίζουν μια ροή καθώς και τον συνολικό αριθμό των bytes και πακέτων που μεταφέρθηκαν για κάθε ροή χωρίς όμως να αναλύουν ή να αποθηκεύουν το περιεχόμενο των μεταδιδόμενων πακέτων. Ως εκ τούτου, η ανάλυση ροών έχει μερικά πλεονεκτήματα. Αρχικά εφόσον το περιεχόμενο των πακέτων δεν αναλύεται, η παρακολούθηση ροών είναι σημαντικά γρηγορότερη από άλλες μεθόδους παρακολούθησης δεδομένων σε κάποιο υλικό ίδιας υπολογιστικής ισχύς. Επιπρόσθετα η προστασία προσωπικών δεδομένων δεν είναι ανησυχητικό θέμα στην περίπτωση παρακολούθησης ροών συγκριτικά με άλλες μεθόδους μιας και δεν αποθηκεύονται τα δεδομένα των πακέτων.

Η διαδικασία παρακολούθησης ροών είναι η εξής:

- Πρώτα τα πακέτα στέλνονται στην διαδικασία καταμέτρησης (metering process) όπου αναγνωρίζονται οι ροές των πακέτων και καταμετρώνται τα στατιστικά τους. Τα πακέτα δεν αναλύονται περισσότερο μετά από αυτό το σημείο.
- Έπειτα η τα στοιχεία των ροών στέλνονται σε κάποια διαδικασία εξαγωγής (export process) μετά από κάποιο χρονικό περιθώριο και από εκεί στέλνονται στην διαδικασία συλλογής (collector) όπου αποθηκεύονται για περεταίρω ανάλυση.



Εικόνα 5: Αρχιτεκτονική παρακολούθησης ροών [3]

2.4 Συμβατικά εργαλεία δοκιμών δικτύων

2.4.1 Ping

Υπάρχει μεγάλος αριθμός εργαλείων για την εκτέλεση δοκιμών σε δίκτυα το κάθε ένα να εξυπηρετεί διαφορετικούς σκοπούς. Ένα από τα πιο ευρέως χρησιμοποιούμενα εργαλεία είναι το Ping [5] που χρησιμοποιείται κυρίως για τον έλεγχο της διαθεσιμότητας και προσιτότητας ενός κόμβου. Το Ping λειτουργεί στέλνοντας πακέτα τύπου ICMP προς τον προορισμό και αναμένει για απάντηση τύπου ICMP response. Κατά την διάρκεια αποστολής και παραλαβής πακέτων καταγράφεται ο χρόνος εκτέλεσης της εντολής ως χρόνος μετ' επιστροφής και το packet loss καθώς το ping θεωρεί χαμένο ένα πακέτο εάν δεν έχει παραλάβει κάποια απάντηση ή εάν έχει λήξη (timed-out). Μετά την ολοκλήρωση της εντολής συγκεντρώνονται στατιστικά και απεικονίζονται ο αριθμός των πακέτων που στάλθηκαν, που παραλήφθηκαν, το ποσοστό των χαμένων πακέτων και ο χρόνος μετ' επιστροφής.

Καθώς υπάρχουν πολλά δοκιμάστηκα εργαλεία που χρησιμοποιούνται για τις πιο πάνω μετρήσεις [6] το ping είναι το πιο δημοφιλής μιας και είναι το πιο συνηθισμένο εργαλείο. Υποστηρίζεται και είναι ενσωματωμένο στις περισσότερες συσκευές επικοινωνίας και λειτουργικά συστήματα και ως εκ τούτου μπορεί να χρησιμοποιηθεί χωρίς κάποια επιπλέον εγκατάσταση. Έπειτα για την χρήση του δεν χρειάζεται η εγκατάστασή του και στα δύο άκρα της ζεύξης πράγμα που εξοικονομεί χρόνο και εργασία στην διάγνωση και αντιμετώπιση τυχών προβλημάτων.

2.4.2 Iperf

Ένα άλλο δοκιμαστικό εργαλείο είναι το Iperf [7] . Σε αντίθεση με το Ping το Iperf είναι ένα από τα εργαλεία που χρησιμοποιούνται για τον υπολογισμό του bandwidth και κατ' επέκταση της ποιότητας μίας ακμής σε ένα δίκτυο. Για την λειτουργία του, το Iperf χρειάζεται να εγκατασταθεί και στα δύο άκρα μιας ζεύξης εφόσον χρησιμοποιεί υπηρεσία πελάτη – εξυπηρετητή. Ακόμα το συγκεκριμένο εργαλείο υποστηρίζει το TCP καθώς και το UDP πρωτόκολλο μεταφοράς καθιστώντας κατάλληλο για τις περισσότερες μορφές δικτύων. Ως εκ τούτου έχει την δυνατότητα να υπολογίσει και άλλα χαρακτηριστικά όπως delay, jitter και packet loss.

Το δοκιμαστικό εργαλείο Iperf είναι απλό στην χρήση σε αντίθεση με άλλα εργαλεία υπολογισμού bandwidth όπως Netperf [8] ή Nuttcp [9] τα οποία συμπεριλαμβάνουν επιπρόσθετα χαρακτηριστικά που πολλές φορές δεν είναι χρήσιμα σε δοκιμές για δίκτυα. Ως εκ τούτου πρόκειται να προτιμήσουμε την χρήση του εργαλείου Iperf στην παρούσα διπλωματική για την μέτρηση του bandwidth, packet loss και του delay μιας ακμής στα πειράματα που θα διεξάγουμε.

2.4.3 Traceroute

Το Traceroute [10] είναι άλλο ένα συμβατικό εργαλείο δοκιμών δικτύου που δείχνει την διαδρομή μεταξύ αποστολέα και παραλήπτη σε μία τοπολογία δικτύου ενώ παράλληλα υπολογίζει το Delay των πακέτων της σύνδεσης τους. Το Traceroute στέλνει IP πακέτα σε κάθε κόμβο που ανήκει στην διαδρομή αποστολέα - παραλήπτη χρησιμοποιώντας το πεδίο time-to-live (TTL) του πακέτου. Με το TTL ο αποστολέας εξετάζει εάν κάθε βήμα (hop) είναι προσβάσιμο στέλνοντας του πακέτα τύπου UDP (προκαθορισμένο πρωτόκολλο, μπορεί να είναι και TCP, ICMP). Όταν η τιμή του πεδίου TTL είναι μεγαλύτερη από 1 τότε μειώνεται κατά 1 και προωθείται στον επόμενο κόμβο της διαδρομής, ενώ όταν η τιμή του είναι πλέον ίση με 1, τότε ο παραλήπτης απορρίπτει το πακέτο και στέλνει στον αποστολέα πακέτο τύπου *ICMP Time Exceeded*. Έτσι ο αποστολέας ενημερώνεται για τους ενδιάμεσους κόμβους μεταξύ αυτού και του παραλήπτη και για την απόσταση που έχει από τον καθένα από αυτούς. Τέλος το εργαλείο τυπώνει τον χρόνο της Round – Trip διαδρομής των πακέτων για κάθε βήμα μέχρι τον παραλήπτη.

2.5 Μέθοδοι παρακολούθησης δικτύων

Ένα από τα πιο χρησιμοποιημένα εργαλεία για monitoring σε δίκτυα είναι το Simple Network Management Protocol (SNMP) [11]. Δημιουργημένο από το 1988 το SNMP είναι πλέον εγκατεστημένο στα περισσότερα δίκτυα σήμερα. Όπως πολλά άλλα εργαλεία, το SNMP έχει την δυνατότητα να συλλέγει δεδομένα για κάθε θύρα, διεπαφή ή γενικά στατιστικά για κόμβους από κάθε μεταγωγέα. Το SNMP βασίζεται κυρίως στην ταχτική συλλογή δεδομένων από τους μεταγωγείς πράγμα που ίσως επηρεάσει την απόδοση τους λόγω περιορισμών του υλικού.

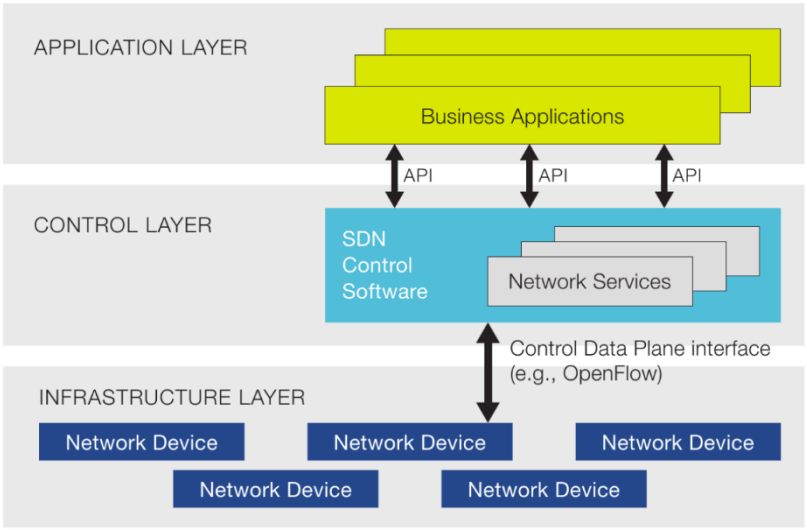
Το NetFlow [12] είναι ένα άλλο παθητικής μορφής εργαλείο που χρησιμοποιείται για συλλογή δεδομένων σε δίκτυα. Το NetFlow συλλέγει πακέτα δείγματα για κάθε ροή και βάσει εκείνων εξάγει γενικά στατιστικά για την κάθε μια. Συγκεκριμένα χρησιμοποιεί αλγόριθμο επιλογής 1-από-n τυχαία πακέτα, αποθηκεύει δηλαδή ένα πακέτο ανά n-άδα και θεωρεί πως αντιπροσωπεύει όλη την κίνηση της ροής. Ενώ η τεχνική με τα δείγματα θεωρείται πως έχει αρκετά καλή ακρίβεια, έχει εντούτοις και σημαντικά προβλήματα. Μερικά από αυτά είναι η πιθανότητα αποφυγής παραλαβής δειγμάτων από πιο μικρές ροές κατά τις μετρήσεις ή η αποθήκευση και ανάλυση του ίδιου πακέτου δείγματος από πολλούς κόμβους σε κάποιο μονοπάτι, επηρεάζοντας έτσι την ακρίβεια των μετρήσεων για ορισμένες ροές. Τα προβλήματα αυτά όμως επιλύει το εργαλείο cSamp [13] το οποίο αποθηκεύει δείγματα βάσει των διαφορετικών ροών και χρησιμοποιεί μεθόδους hashing προκειμένου να αποφύγει την διπλή αποθήκευση ίδιου δείγματος πακέτου.

Ένα παράδειγμα εργαλείου με ενεργητική μέθοδο συλλογής δεδομένων είναι το Skitter [14]. Το Skitter τοποθετεί φάρους (beacons) σε διάφορα σημεία του δικτύου τα οποία βοηθούν στην καταμέτρηση μεγεθών όπως Round-Trip Time (RTT) και delay μεταξύ συνδέσεων στο δίκτυο. Συγκεκριμένα δημιουργεί και στέλνει πακέτα που διαθέτουν χρονοσφραγίδα (timestamp) τα οποία σε συνδυασμό με τα beacons υπολογίζουν το γενικό delay του δικτύου. Για πιο ακριβείς μετρήσεις που αφορούν όλες τις ροές (flows) όμως, είναι απαραίτητη η εγκατάσταση μεγάλου όγκου από beacons πράγμα που δεν είναι βέλτιστο.

Κεφάλαιο 3: Software Defined Networking

3.1 Βασική ιδέα

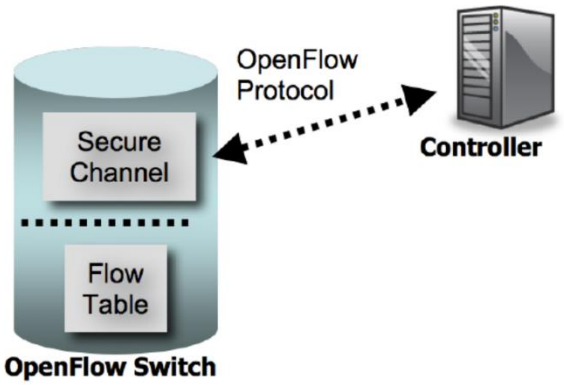
Η αρχιτεκτονική SDN είναι μία καινοτόμα προσέγγιση στην σχεδίαση, ανάπτυξη και διαχείριση των δικτύων που χωρίζει το επιπέδου ελέγχου (control plane) και επιπέδου δεδομένων (data plane) για την ευκολότερη προς τον χρήστη εμπειρία. Ενώ η ιδέα των δικτύων οριζόμενων από λογισμικό έχει έρθει στο φως τα τελευταία χρόνια, η ιδέα αυτής της προσέγγισης εξελισσόταν από την δεκαετία του 1900 με το Openflow και το Ethane **[15]** να είναι τα πρώτα που ασχολήθηκαν με την ιδέα ενός διαχειριστή (controller) να επικοινωνεί και να διαχειρίζεται κίνηση μεταξύ μεταγωγέων (switches)Το επίπεδο δεδομένων που είναι επίσης γνωστό και ως επίπεδο προώθησης και λήψης φροντίζει την σωστή προώθησης πακέτων μέσα στο δίκτυο, ενώ το επίπεδο ελέγχου αφορά την λογική πίσω από τις προωθήσεις πακέτων. Με την αρχιτεκτονική SDN το επίπεδο δεδομένων αναλαμβάνουν μεταγωγείς του δικτύου ενώ το επίπεδο ελέγχου τοποθετείται σε έναν λογικό συγκεντρωτικό (centralized) λειτουργικό σύστημα δικτύου (Network Operating System, NOS) γνωστό και ως διαχειριστής. Αυτή η τμηματοποίηση δικτύου προσφέρει πολλά οφέλη όσον αφορά την ευελιξία και τη δυνατότητα ελέγχου του δικτύου. Ένα από τα πλεονεκτήματα του διαχωρισμού των επιπέδων ελέγχου και δεδομένων είναι ότι η διαχείριση του δικτύου γίνεται απλούστερη λόγω του ότι πλέον το επίπεδο ελέγχου ασχολείται μόνο με τις πληροφορίες που σχετίζονται με τη λογική τοπολογία δικτύου, τη δρομολόγηση της κίνησης και ούτω καθεξής. Σε αντίθεση, το επίπεδο δεδομένων τακτοποιεί την κυκλοφορία του δικτύου σύμφωνα με την καθιερωμένη διαμόρφωση στο επίπεδο ελέγχου.



Εικόνα 6: Βασική αρχιτεκτονική των Software Defined Networks [16]

3.2 OpenFlow Protocol

Με τον διαχωρισμό μεταξύ επιπέδου ελέγχου και επιπέδου δεδομένων που προσφέρει στα δίκτυα η αρχιτεκτονική SDN είναι απαραίτητη μια διεπαφή που να φροντίζει την σωστή αλληλεπίδραση μεταξύ των δύο επιπέδων. Το πρωτόκολλο OpenFlow που προτείνεται από το Open Network foundation (ONF) [17] είναι μια τέτοια διεπαφή. Με το OpenFlow ο SDN διαχειριστής μπορεί να επικοινωνεί με ασφάλεια με τους OpenFlow μεταγωγείς επιτρέποντάς του να εγκαταστήσει, ενημερώσει ή να διαγράψει κανόνες σε αυτούς. Συγκεκριμένα κάθε OpenFlow μεταγωγέας στην τοπολογία έχει πίνακες ροών, πίνακες δηλαδή με κανόνες που λαμβάνει από τον διαχειριστή SDN με κάθε νέα παραλαβή πακέτου που βοηθούν στην βέλτιστη προώθηση αυτών ανάμεσα στους κόμβους του δικτύου.

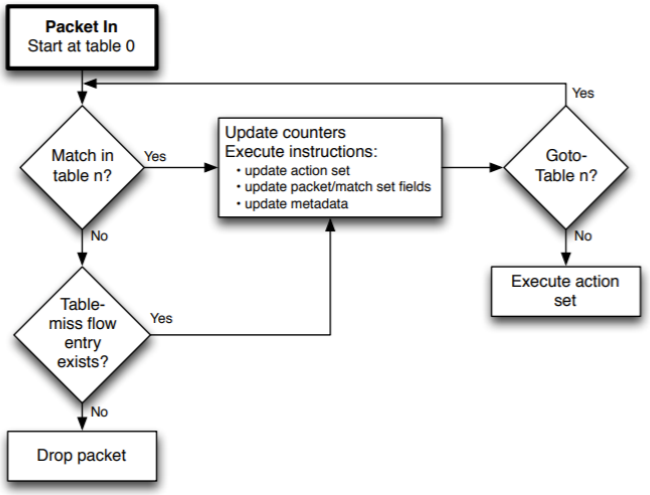


Εικόνα 7: Αρχιτεκτονική του OpenFlow μεταγωγέα [18]

Κατά την παραλαβή ενός πακέτου σε ένα Openflow μεταγωγέα, αυτός ελέγχει τον πίνακα ροών του για κάποια ροή που να ταιριάζει με την διεύθυνση προέλευσης και προορισμού ή ακόμα και με την θύρα του πακέτου. Εφόσον ο πίνακας δεν έχει κάποιο κανόνα που να ταιριάζει με το εισερχόμενο πακέτο, ο μεταγωγέας στέλνει πακέτο τύπου Packet-In στο SDN διαχειριστή [19]. Το πακέτο αυτό έχει σκοπό να ενημερώσει τον διαχειριστή για το παραληφθέν πακέτο ο οποίος με την σειρά του θα απαντήσει με έναν εκ των 2 τρόπων ανάλογα με τις ρυθμίσεις του:

- Πακέτο Packet-Out: Ο διαχειριστής μπορεί να ρυθμιστεί ώστε κάθε πακέτο που φτάνει σε κάποιον μεταγωγέα να έχει διαφορετική μεταχείριση. Αυτό επιτυγχάνεται με το Packet-Out πακέτο το οποίο συμπεριλαμβάνει ένα σύνολο εντολών (actions) που δίνει ο διαχειριστής στον μεταγωγέα προκειμένου να διαχειριστεί το συγκεκριμένο πακέτο.
- Πακέτο Flow-MOD: Στην περίπτωση εγκατάστασης επικοινωνίας μεταξύ δύο κόμβων, ανταλλάσσονται πολλά πακέτα ίδιας μορφής με παρόμοια χαρακτηριστικά. Σε τέτοια περίπτωση ο διαχειριστής μπορεί να ρυθμιστεί ώστε να στείλει ένα πακέτο τύπου Flow-MOD πίσω στον μεταγωγέα το οποίο θα εγκαταστήσει μια καινούρια ροή στον πίνακα ροών του.

Έτσι ο μεταγωγέας θα γνωρίζει πώς να διαχειρίζεται και μελλοντικά πακέτα με παρόμοια στοιχεία χωρίς να χρειαστεί άλλη επικοινωνία με τον διαχειριστή.



Εικόνα 8: Σχεδιάγραμμα που περιγράφει την ροή ενός πακέτου στον OpenFlow μεταγωγέα

Κάθε ροή στους πίνακες ροών των μεταγωγέων χαρακτηρίζεται από μερικά στοιχεία που την καθιστούν μοναδική Μερικά από τα πιο βασικά χαρακτηριστικά μίας ροής είναι τα εξής:

- Buffer ID: Ένας αριθμός που αναφέρεται σε ένα τύπο εισερχόμενων πακέτων που αποσκοπεί στην διευκόλυνση της αντιστοιχίας τους με κάποια ροή.
- Idle Timeout: Το χρονικό διάστημα μέχρι να αποσυρθεί η συγκεκριμένη ροή από τον πίνακα ροών του μεταγωγέα εφόσον δεν υπήρξαν πακέτα που αντιστοιχήθηκαν με την συγκεκριμένη ροή.
- Hard Timeout: Το χρονικό διάστημα μέχρι να αποσυρθεί η συγκεκριμένη ροή από τον πίνακα ροών του μεταγωγέα ακόμα και αν υπήρχε διακίνηση πακέτων που αντιστοιχίζονταν με την ροή αυτή. Η λήξη του συγκεκριμένου μετρητή προκαλεί την δημιουργία πακέτου τύπου FlowRemoved, που ενημερώνει την διαχειριστή για την απόσυρση της ροής από τον μεταγωγέα.
- Priority: Ένας αριθμός που χρησιμοποιείται για την ταξινόμηση των ροών σε ένα πίνακα. Εφόσον ένα πακέτο αντιστοιχιστεί με περισσότερες από μία ροές θα υπερισχύσει αυτή με το μεγαλύτερο priority.
- Actions: Ένα σύνολο εντολών με τις οποίες πρέπει να διαχειρίζεται ο μεταγωγέας τα πακέτα που αντιστοιχούν στη συγκεκριμένη ροή. Παραδείγματα τέτοιων εντολών μπορεί να είναι η επεξεργασία, απόρριψη ή προώθηση πακέτου σε συγκεκριμένη διεύθυνση.

3.3 Συλλογή δεδομένων σε SDN δίκτυα (Related Work)

Με την ευκολία στην διαχείριση δικτύου που προσφέρει η τεχνολογία SDN σε συνδυασμό με το πρωτόκολλο Openflow, η προσέγγιση καλύτερων QoS μετρήσεων είναι ακόμα πιο εφικτή. Η πρόσβαση που έχει ο διαχειριστής σε όλο το δίκτυο στα SDN εγγυάται σταθερό και έμπιστο υπολογισμό σημαντικών παραμέτρων μεταξύ κόμβων όπως το bandwidth packet loss ή latency η διαχείριση των οποίων είναι απαραίτητη για τα σύγχρονα δίκτυα. Χρησιμοποιώντας την προγραμματιστική φύση που παρέχει το Openflow πρωτόκολλο τα SDN προσφέρουν ευελιξία και πιο σημαντικά, δυναμικό υπολογισμό των προαναφερθέντων παραμέτρων που εγγυόνται υψηλή ποιότητα επικοινωνίας μεταξύ endpoints σε όλο το δίκτυο ανεξαρτήτως της κατάστασης του.

Έχουν πραγματοποιηθεί πολλές μέθοδοι για συλλογή δεδομένων σε SDN δίκτυα. Στην παρούσα διπλωματική θα αναφερθούν μερικές από αυτές:

OpenTM [20]: Ένα από τα πιο δημοφιλή εργαλεία συλλογής δεδομένων για τα δίκτυα που χρησιμοποιούν το πρωτόκολλο Openflow είναι το OpenTM. Χρησιμοποιώντας τις μοναδικές ιδιότητες του πρωτοκόλλου, το OpenTM υπολογίζει τον όγκο δεδομένων κυκλοφορίας (Traffic Matrix) από άκρη σε άκρη σε ένα δίκτυο συλλέγοντας τους μετρητές των byte και των πακέτων που αποθηκεύουν οι Openflow μεταγωγείς για τις ενεργές ροές τους. Έτσι επιφέρει ελάχιστη επιβάρυνση στο δίκτυο ενώ παράλληλα προσφέρει υψηλή ακρίβεια στις μετρήσεις του εφόσον ο όγκος κυκλοφορίας υπολογίζεται άμεσα χωρίς την χρήση απλοποιήσεων και μαθηματικών σχέσεων.

Για την βέλτιστη απόδοση του εργαλείου δοκιμάστηκαν διάφορες μέθοδοι συλλογής δεδομένων εφόσον η θέση ενός μεταγωγέα σε μια διαδρομή μπορεί να παίζει σημαντικό ρόλο στην ακρίβεια των αποτελεσμάτων. Συγκεκριμένα συλλέγοντας αποτελέσματα από τον τελευταίο μεταγωγέα ενός μονοπατιού επιφέρει τις πιο ακριβείς μετρήσεις θυσιάζοντας όμως την ποιότητα της ζεύξης στο σημείο αυτό εφόσον επιβαρύνει τον μεταγωγέα με μεγάλο όγκο δεδομένων. Έτσι χρησιμοποιούνται διάφοροι αλγόριθμοι για την συλλογή δεδομένων από τους μεταγωγείς για την ισορροπία ακρίβειας μετρήσεων και επιβάρυνσής του δικτύου. Μερικοί από αυτούς είναι επιλογή τυχαίου μεταγωγέα στο μονοπάτι, επιλογή τυχαίου μεταγωγέα με μεγαλύτερη πιθανότητα επιλογής κάποιου πλησιέστερου στον προορισμό ή επιλογή με την μέθοδο Round-Robin.

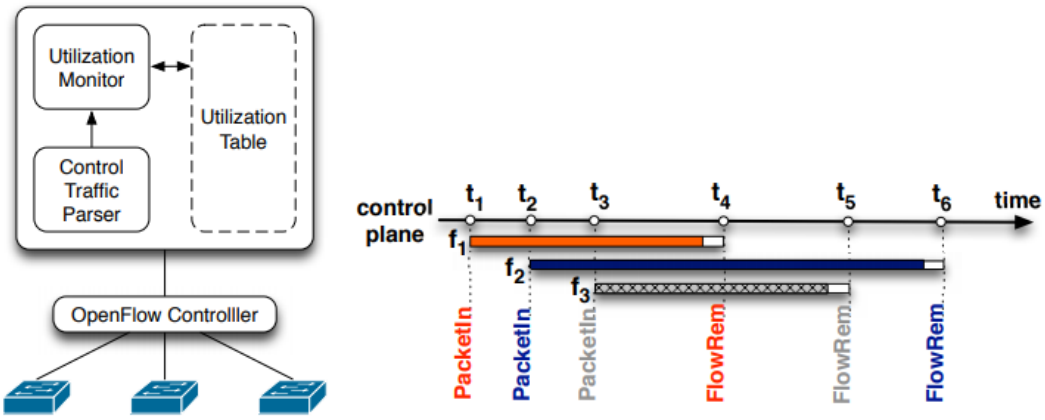
Το OpenTM συλλέγει ανά τακτά χρονικά διαστήματα δεδομένα από κάθε ενεργό ζεύγος IP διευθύνσεων (i,j) και υπολογίζει τον όγκο δεδομένων κυκλοφορίας αθροίζοντας τα στατιστικά κάθε ροής που προήλθαν από τον μεταγωγέα i και είχαν προορισμό τον μεταγωγέα j. Ξεκινά την καταγραφή για ένα ζεύγος διευθύνσεων όταν εμφανιστεί η πρώτη ροή δεδομένων μεταξύ των δύο και σταματά την καταγραφή όταν όλες οι ροές μεταξύ τους έχουν λήξη. Το OpenTM ενημερώνεται για τον αριθμό των ενεργών ροών ο οποίος αυξάνεται η μειώνεται αντίστοιχα κατά την παραλαβή *FlowRemoved* και *PacketIn* πακέτων. Όταν ο αριθμός των ενεργών ροών φτάσει το 1 το OpenTM στέλνει πακέτο τύπου *aggregate flow statistics* σε κάποιον από τους μεταγωγείς του μονοπατιού και ενημερώνει τον όγκο δεδομένων κυκλοφορίας όταν παραλάβει την απάντηση από αυτό. Εφόσον οι ενεργές ροές δεν έχουν φτάσει ακόμα το 0 τότε θα επαναλάβει την διαδικασία στέλνοντας σε

κάποιον άλλο μεταγωγέα πακέτο τύπου *aggregate flow statistics* η επιλογή του οποίου βασίζεται στην επεξεργαστική του ισχύς.

FlowSense **[21]**: Το FlowSense είναι ένα εργαλείο που βασίζεται στον ρυθμό μετάδοσης *PacketIn* και *FlowRemoved* πακέτων μεταξύ διαχειριστή και μεταγωγέων προκειμένου να συλλέξει και να υπολογίσει τα βασικά στατιστικά για το δίκτυο κατά την προσθήκη ή λήξη ροών αντίστοιχα. Χρησιμοποιεί χρονοσφραγίδες για να υπολογίζει ανά πάσα στιγμή τον αριθμό των ροών που είναι ενεργά και βάσει αυτού να μετρήσει το συνολικό bandwidth σε κάθε ακμή του δικτύου αθροίζοντας κάθε φορά το utilization των ενεργών ροών ξεχωριστά. Αυτό επιτυγχάνεται εξάγοντας από τα *FlowRemoved* πακέτα που θα παραχθούν με την λήξη των ροών τα εξής δεδομένα:

- Η διάρκεια ζωής της ροής, ο χρόνος δηλαδή που ήταν ενεργή.
- Το μέγεθος του όγκου δεδομένων που αντιστοιχήθηκε με αυτήν σε bytes.
- Η θύρα του μεταγωγέα από όπου προέρχονται τα πακέτα που αντιστοιχήθηκαν με την συγκεκριμένη ροή.

Με αυτά τα δεδομένα μπορεί εύκολα να υπολογιστεί πόσο η συγκεκριμένη ροή συνεισέφερε στο utilization της ακμής που συνδέεται με την συγκεκριμένη θύρα. Παράλληλα προκειμένου να υπολογιστεί η συνεισφορά όλων των ενεργών ροών κάθε φορά, χρησιμοποιούνται τα δεδομένα των *PacketIn* πακέτων τα οποία ενημερώνουν πότε κάποια καινούρια ροή φτάνει στον μεταγωγέα. Επομένως προκειμένου να υπολογιστεί το utilization που προκύπτει από κάποια ροή για μία ακμή υπάρχει μια καθυστέρηση μέχρι την λήξη της συγκεκριμένης ροής, όταν δηλαδή παραχθούν τα *FlowRemoved* πακέτα που χρειάζονται. Έτσι σε κάθε σημείο ελέγχου (checkpoint) είναι γνωστό ποιες ροές είναι ενεργές και συνεισέφεραν στην αντίστοιχη μέτρηση του utilization (εικόνα 1).



Εικόνα 9: Αλγόριθμος λειτουργίας του FlowSense [22].

PayLess [22]: Το PayLess είναι ένα άλλο εργαλείο χαμηλού κόστους που χρησιμοποιείται επίσης για συλλογή στατιστικών σε δίκτυα. Παρέχει ένα εύχρηστο περιβάλλον εργασίας που είναι εύκολα προσβάσιμο από διάφορες γλώσσες προγραμματισμού. Με το Payless ο χρήστης μπορεί να δώσει εντολή για την συλλογή συγκεκριμένων χαρακτηριστικών από τους μεταγωγείς και στην συνέχεια το εργαλείο αποφασίζει και συλλέγει τα απαραίτητα δεδομένα από τους κατάλληλους μεταγωγείς βάσει διαφόρων ευριστικών συναρτήσεων.

Μερικές από τις επιλογές που έχει ο χρήστης για να προσαρμόσει την συλλογή δεδομένων στο δίκτυο είναι οι εξής:

- Type: Υπάρχει δυνατότητα επιλογής του είδους των δεδομένων που πρόκειται να συλλεγούν. Όσον αφορά τα στατιστικά των ακμών στο δίκτυο το Payless προσφέρει την επιλογή «Performance».
- Metrics: Με την πιο πάνω επιλογή το Payless προσφέρει τον υπολογισμό στατιστικών όπως latency", "jitter", "throughput", "packet-drop".
- AggregationLevel: Μπορεί να ρυθμιστεί ακόμα και το επίπεδο συλλογής δεδομένων, εάν δηλαδή πρόκειται να συλλεγούν στατιστικά ανά ροή, θύρα, μεταγωγέα και άλλα.

Το Payless χρησιμοποιεί έναν προσαρμοστικό αλγόριθμο για την συλλογή δεδομένων στον δίκτυο. Ομοίως με το FlowSense, χρησιμοποιεί τα πακέτα τύπου *PacketIn* και *FlowRemoved* για να μεταξύ του διαχειριστή και των μεταγωγέων για να υπολογίσει τα στατιστικά των ακμών του δικτύου. Το Payless όμως χρησιμοποιεί και πακέτα τύπου *FlowStatisticsRequest* προκειμένου να ζητήσει στατιστικά των ροών από τους μεταγωγείς χωρίς να περιμένει πάντα τα *FlowRemoved* πακέτα που παράγονται κατά την λήξη των ροών.

Συγκεκριμένα μετά από κάποιο σταθερό χρόνο, εάν δεν έχει παραλάβει *FlowRemoved* πακέτο, ο διαχειριστής στέλνει *FlowStatisticsRequest*. Επίσης για να αποφευχθεί η συμφόρηση πακέτων σε ακμές που συνεισφέρουν σημαντικά στο δίκτυο ή για να μην υπάρχει μεγάλη συχνότητα *FlowStatisticsRequest* πακέτων σε ακμές που δεν χρειάζεται, το PayLess χρησιμοποιεί διάφορες μεταβλητές για να προσαρμόζεται άμεσα στον όγκο της κίνηση πακέτων στις ακμές του δικτύου. Συγκεκριμένα, εφόσον η διαφορά στις μετρήσεις που εξάγονται για μία ροή δεν έχει ξεπεράσει κάποια τιμή *Δ1* τότε αυξάνεται ο χρόνος timeout για την ροής κατά κάποια τιμή *α*. Με αυτό τον τρόπο αποφεύγεται η παραγωγή μεγάλου αριθμού *FlowStatisticsRequest* πακέτων εφόσον δεν είναι απαραίτητο. Αντίστοιχα στην αντίθετη περίπτωση που η διαφορά στις μετρήσεις για μία ροή ξεπεράσει κάποια τιμή *Δ2* τότε μειώνεται ο χρόνος timeout για τη ροή κατά κάποια σταθερά *β*. Έτσι για πιο εύχρηστες ροές στο δίκτυο που χρειάζονται πιο συχνή παρακολούθηση εξάγονται με μεγαλύτερο ρυθμό πακέτα τύπου *FlowStatisticsRequest* .

OpenNetMon [23]: Όπως και τα πιο πάνω, το OpenNetMon είναι επίσης ένα εργαλείο που χρησιμοποιείται για τις μετρήσεις σημαντικών μεγεθών σε δίκτυα με αρχιτεκτονική SDN χρησιμοποιώντας τις ιδιότητες του OpenFlow πρωτοκόλλου. Με παρόμοιο τρόπο και το OpenNetMon αποθηκεύει στατιστικά από τις ροές του δικτύου τα οποία συλλέγει ανά τακτά χρονικά διαστήματα προκειμένου να υπολογίσει throughput, packet loss και delay.

Για τον υπολογισμό του throughput το OpenNetMon ζητάει τακτικά από τους μεταγωγείς της τοπολογίας τα στατιστικά των ροών που έχουν εγκατεστημένα χρησιμοποιώντας τα πακέτα τύπου *PacketIn* και *FlowRemoved* εφόσον με τα δεδομένα τους μπορεί να γνωρίζει πότε πρόκειται να εγκατασταθούν καινούριες ροές ή να λήξουν ήδη υπάρχουσες. Έτσι το εργαλείο συλλέγει στοιχεία των ροών όπως το πλήθος των byte που αντιστοιχήθηκαν με κάποια ροή και τον χρόνο που η κάθε ροή ήταν ενεργή επιτρέποντας του έτσι να υπολογίσει το throughput κάθε ροής. Για τον υπολογισμό του packet loss το OpenNetMon ζητάει περιοδικά από τον πρώτο και τον τελευταίο μεταγωγέα ενός μονοπατιού τον αριθμό των πακέτων που αντιστοιχήθηκαν με την κάθε ροή. Συγκρίνοντας τα δύο μεγέθη προκύπτει το packet loss. Εφόσον τα δεδομένα που παίρνουν από τους δύο μεταγωγείς κάθε μονοπατιού επιστρέφουν και το πλήθος των bytes καθώς και τον χρόνο ζωής κάθε ροής, το εργαλείο συνδυάζει τον υπολογισμό του throughput και του packet loss αντλώντας τελικά πληροφορίες για το throughput μόνο από τον τελευταίο μεταγωγέα μιας διαδρομής.

Για τον υπολογισμό του delay μιας διαδρομής, το OpenNetMon προωθεί ανά τακτά χρονικά διαστήματα ένα επιπλέον πακέτο στον πρώτο μεταγωγέα μιας διαδρομής το οποίο ακολουθώντας την διαδρομή των υπολοίπων πακέτων, επιστρέφει πίσω στον διαχειριστή του δικτύου αφού το προωθήσει ο τελευταίος μεταγωγέας. Επομένως το delay υπολογίζεται με την διαφορά του χρόνου εκκίνησης και του χρόνου παραλαβής του πακέτου αφαιρώντας ακόμα δύο φορές τον χρόνο που χρειάζεται ένα πακέτο να μεταφερθεί από τον διαχειριστή στον μεταγωγέα. Το τελευταίο μέγεθος μπορεί εύκολα να υπολογιστεί διαιρώντας διά δύο τον RTT χρόνο πακέτων που απορρίπτονται από έναν μεταγωγέα και επιστρέφουν απευθείας πίσω στον διαχειριστή.

3.4 Προτεινόμενη Λύση

Έχοντας υπόψη τον τρόπο λειτουργείας των διαφόρων εργαλείων που αναπτύχθηκαν μέχρι τώρα φτιάξαμε και εμείς σε αυτή την αναφορά τον δικό μας διαχειριστή χρησιμοποιώντας τον διαχειριστή POX, οποίος με παρόμοιο τρόπο εξάγει στατιστικά για τις συνδέσεις στα SDN δίκτυα χρησιμοποιώντας μόνο τις ιδιότητες του πρωτοκόλλου OpenFlow.

Στην δική μας περίπτωση ο διαχειριστής στέλνει πακέτα τύπου *FlowStatsRequest* σε έναν μεταγωγέα μιας διαδρομής ανά τακτά χρονικά διαστήματα με παρόμοιο τρόπο δηλαδή όπως και το εργαλείο Payless και χρησιμοποιώντας τον χρόνο ζωής των ροών καθώς και τον αριθμό των bytes που αντιστοιχήθηκαν με την κάθε ροή, που παίρνουμε από τις απαντήσεις των μεταγωγέων, υπολογίζει εύκολα το throughput της ζεύξης. Σε αντίθεση με το εργαλείο FlowSense η δική μας λύση δεν βασίζεται πάνω στην δημιουργία ή λήξη των ροών της ζεύξης, αφού ζητώντας στατιστικά από τις ροές ανά διαστήματα δεν χρειάζεται να περιμένουμε την δημιουργία των *FlowRemoved* ή των *PacketIn* πακέτων για να προσδιορίσουμε την συνεισφορά των ροών στον υπολογισμό του packet loss ή του throughout μιας ακμής.

Κατόπιν για τον υπολογισμό του packet loss παρομοίως με το OpenNetMon,α αντλούμε πληροφορίες από το πρώτο και τελευταίο μεταγωγέα μιας ζεύξης, και πάλι ανά τακτά χρονικά διαστήματα, όμως εμείς στέλνουμε πακέτα τύπου *PortStatsRequest* από τα οποία μπορούμε να υπολογίσουμε και να συγκρίνουμε το συνολικό πλήθος των πακέτων που στάλθηκαν από την πηγή παράλληλα με το συνολικό πλήθος των πακέτων που παρέλαβε ο προορισμός, εξάγοντας έτσι το packet loss. Με αυτό το τρόπο το packet loss υπολογίζεται με άμεσο τρόπο χωρίς να χρειάζεται να υπολογιστεί η συνεισφορά κάθε ροής ξεχωριστά όπως το υπολογίζει δηλαδή το OpenNetMon.

Τέλος για τον υπολογισμό της καθυστέρησης (delay) ενός μονοπατιού με παρόμοιο τρόπο με το OpenNetMon δημιουργούμε και στέλνουμε ένα πακέτο τύπου PacketIn στον πρώτο μεταγωγέα ενός μονοπατιού με τελικό προορισμό του τον τελευταίο μεταγωγέα του μονοπατιού. Ξεκινάμε χρονομετρητή όταν το στείλουμε τον σταματάμε όταν το πακέτο φτάσει στον προορισμό του. Έπειτα για τον υπολογισμό του delay αφαιρούμε από την διαφορά των δύο χρόνων δύο φορές τον χρόνο μέχρι το πακέτο να φτάσει στους μεταγωγείς από τον διαχειριστή. Στην δική μας περίπτωση όμως σε αντίθεση με το OpenNetMon βρίσκουμε τον χρόνο αυτό μετρώντας τον χρόνο αποστολής και παραλαβής των πακέτων *PortStatsRequest* που χρησιμοποιούμε ήδη για τον υπολογισμό του packet loss. Έτσι με τον τρόπο μας δεν επιβαρύνουμε περισσότερο το δίκτυο για τον υπολογισμό του delay.

Η πιο πάνω μέθοδος προσφέρει καλή ακρίβεια στις μετρήσεις εφόσον η επιπλέον κίνηση που φορτώνουμε στο δίκτυο είναι μηδαμινή μιας και επιλέγουμε μικρό αριθμό μεταγωγέων στους οποίους στέλνουμε τα προαναφερθέν πακέτα. Έπειτα η ανάπτυξη της συγκεκριμένης μορφής διαχειριστή αποτελεί σημαντικό επίτευγμα διότι ο υπολογισμός των συγκεκριμένων στατιστικών γίνεται με εύκολο και παθητικό τρόπο. Λεπτομέρειες για τον τρόπο λειτουργίας καθώς και για τα αποτελέσματά του διαχειριστή φαίνονται στο κεφάλαιο 4 της αναφοράς.

Συνοψίζοντας λοιπόν φτιάξαμε τον ακόλουθο πίνακα, περιγράφοντας μερικά από τα προτερήματα των πιο πάνω γνωστών μεθόδων και της δικής μας υλοποίησης καθώς και του τομείς που υπερτερεί η μία της άλλης . Αναφέρουμε τα μοναδικά τους στοιχεία και τον τρόπο προσέγγισης

του κάθε εργαλείου όσον αφορά την συλλογή δεδομένων για τον υπολογισμό των βασικών στατιστικών (throughput, packet loss, delay) στα δίκτυα.

	Θετικά στοιχεία	Αρνητικά στοιχεία
FlowSense	Δεν επιβαρύνει το δίκτυο με επιπλέον πακέτα δεδομένων. Ευκολότερη χρήση εφόσον συλλέγει παθητικά δεδομένα από τους μεταγωγείς. Μεγάλη ακρίβεια εφόσον υπάρχουν ροές μικρής διάρκειας ζωής στο δίκτυο.	Περιορισμένος χρόνος παραλαβής μετρήσεων λόγω της καθυστέρησης μέχρι την λήξη των ροών. Σε δίκτυο με ροές με μεγάλη διάρκεια ζωής υπάρχει σημαντική καθυστέρηση μεταξύ μετρήσεων οπότε το FlowSense δεν είναι βέλτιστο, επομένως δεν μπορεί να ανταπεξέλθει στις απαιτήσεις ορισμένων πραγματικών δικτύων και δεν μπορεί να έχει εγγυημένη ακρίβεια στους υπολογισμούς του.
PayLess	Με το περιβάλλον χρήστη που προσφέρει, το Payless καλύπτει σχεδόν όλους τους τομείς της συλλογής δεδομένων σε δίκτυα. Επίσης με τον προσαρμοστικό του αλγόριθμο είναι ιδανικό για τοπολογίες με διάφορες αιχμές (spikes) στους ρυθμούς μετάδοσης δεδομένων.	Με τον προσαρμοστικό του αλγόριθμο το Payless χρειάζεται να επιβαρύνει περισσότερο το δίκτυο στέλνοντας με μεγαλύτερο ρυθμό πακέτα τύπου FlowStatisticsRequest σε ορισμένους μεταγωγείς που κρίνει απαραίτητο.
OpenTM	Το OpenTM επιφέρει ελάχιστη επιβάρυνση στο δίκτυο ζητώντας στατιστικά από μικρό αριθμό μεταγωγέων και χρησιμοποιώντας τους κατάλληλους αλγόριθμους επιλέγει τους μεταγωγείς που θα επηρεάσουν το λιγότερο την ποιότητα επικοινωνίας του ζεύγους των IP διευθύνσεων.	Το OpenTM χρησιμοποιείται μόνο για τον υπολογισμό του όγκου δεδομένων μεταξύ ζεύγους κόμβων (Traffic Matrix). Επίσης συλλέγοντας στατιστικά από τον ίδιο μεταγωγέα μπορεί επιφέρει μεγάλη επιβάρυνση στο δίκτυο. Ενώ η επιλογή ενός μικρού αριθμού μεταγωγέων για συλλογή δεδομένων δεν έχει μεγάλο κόστος στο

		δίκτυο, οι μεταγωγείς πρέπει να επιλεγούν πολύ προσεκτικά.
OpenNetMon	Το OpenNetMon επιφέρει επίσης μικρή επιβάρυνση στο δίκτυο αντλώντας πληροφορίας μόνο από ένα ή δύο μεταγωγείς για την μέτρηση packet loss ή throughput αντίστοιχα. Επίσης έχει προσαρμοστική φύση, αλλάζοντας τον ρυθμό συλλογής δεδομένων από τους μεταγωγείς ανάλογα με το πόσο σταθερές παραμένουν οι μετρήσεις που παίρνει κάθε φορά.	Ενώ συλλέγει δεδομένα μόνο από τους δύο μεταγωγείς στην αρχή και στο τέλος μίας διαδρομής, η επιπρόσθετη επιβάρυνση των ακρινών μεταγωγέων μπορεί να επηρεάσει περισσότερο την απόδοση της επικοινωνίας σε σχέση με την επιβάρυνση σε έναν ενδιάμεσο μεταγωγέα. Αυτό ισχύει διότι οι μεταγωγείς στα άκρα μιας τοπολογίας έχουν συνήθως μεγαλύτερο αριθμό ροών που πρέπει να συντηρήσουν καθιστώντας την επιβάρυνσή τους ακόμα πιο δαπανηρή.
Προτεινόμενη Λύση	Η λύση που προτείνεται στην παρούσα διπλωματική έχει επίσης μικρή επιβάρυνση αφού για τον υπολογισμό του throughput και του packet loss αντλούμε πληροφορίες από ένα ή δύο μεταγωγείς αντίστοιχα ενώ για το delay στέλνουμε μόνο ένα επιπλέον πακέτο στο δίκτυο κάθε φορά. Έπειτα η εξαγωγή των δεδομένων γίνεται απευθείας από τα δεδομένα των απαντήσεων των μεταγωγέων χωρίς την χρήση πολύπλοκων μαθηματικών σχέσεων.	Όπως και στην περίπτωση του OpenNetMon σε κάποιες περιπτώσεις η συλλογή δεδομένων από ακρινούς μεταγωγείς μιας ζεύξης ενδέχεται να επιφέρει περισσότερη επιβάρυνση στην ζεύξη του δικτύου μιας και οι μεταγωγείς αυτοί συντηρούν συνήθως περισσότερες ροές.

Πίνακας 1: Συνοπτική σύγκριση των αναφερθέντων μεθόδων παρακολούθησης SDN δικτύων

Κεφάλαιο 4: Πειράματα και μετρήσεις

4.1 Εισαγωγή στο Mininet

4.1.1 Mininet και η πλατφόρμα POX

Στην ενότητα αυτή θα χρησιμοποιήσουμε το εικονικό μηχάνημα **mininet [24]**, ένα προσομοιωτή δικτύων οριζόμενα από λογισμικό, προκειμένου να δημιουργήσουμε και να επεξεργαστούμε τα δεδομένα των δικτύων αυτής της αρχιτεκτονικής. Για τον προγραμματισμό του διαχειριστή θα εγκατασταθεί η πλατφόρμα δικτύωσης POX **[25]** που χρησιμοποιεί την γλώσσα προγραμματισμού Python. Η πλατφόρμα POX καθιστά ευκολότερη την διεξαγωγή πειραμάτων σε δίκτυα που χρησιμοποιούν την αρχιτεκτονική SDN και το OpenFlow πρωτόκολλο αφού επιτρέποντας την επεξεργασία διάφορων παραμέτρων προσφέρει μια ρεαλιστική εικόνα για τα δεδομένα σε τέτοιου είδους τοπολογίες.

Για την ευκολότερη κατανόηση και επεξεργασία της μορφής των τοπολογιών που θα φτιάξουμε θα χρησιμοποιήσουμε και τα πιο κάτω εργαλεία:

- **Gephi [26]:** Το Gephi είναι ένα εργαλείο ανοικτού κώδικα που χρησιμοποιείται κυρίως για εύκολη ανάλυση δικτύων και γράφων με ένα φιλικό προς τον χρήστη περιβάλλον. Μερικές από τις λειτουργίες που προσφέρει είναι η 3D προσομοίωση τοπολογιών, ρύθμιση του μεγέθους των κόμβων, εξαγωγή των δικτύων σε PDF ή SVG αρχεία ή ακόμα και η παράλληλη επεξεργασία ομάδων κόμβων για μεγάλες τοπολογίες στο δίκτυο χρησιμοποιώντας filtering systems **[27]**. Υπάρχουν περισσότερες πληροφορίες στην υποενότητα 7.2 του παραρτήματος όπου αναλύουμε το εργαλείο Gephi, πως να εγκατασταθεί και να χρησιμοποιηθεί για την δημιουργία διαφόρων τοπολογιών.
- **Miniedit [28]:** Το Miniedit είναι το γραφικό περιβάλλον που προσφέρει το mininet για την εύκολη προς τον χρήστη δημιουργία δικτύων. Υπάρχουν περισσότερες πληροφορίες στην υποενότητα 7.3 του παραρτήματος όπου περιγράφουμε πως μπορούμε να χρησιμοποιήσουμε το Miniedit με περισσότερες λεπτομέρειες.
- **Wireshark:** Το Wireshark είναι ένα εργαλείο με το οποίο μπορούμε να μελετήσουμε την κίνηση των πακέτων ανάμεσα στους κόμβους της τοπολογίας μας. Μια πιο λεπτομερείς περιγραφή του εργαλείου Wireshark υπάρχει στην υποενότητα 7.5.
- **SDN Narmox Spear:** Το SDN Narmox Spear είναι ένα online εργαλείο που λαμβάνει ως είσοδο τις πληροφορίες κάθε κόμβου και κάθε ακμής ενός δικτύου και παρουσιάζει με ευανάγνωστο τρόπο τη μορφή του εν λόγω δικτύου. Περισσότερες πληροφορίες για το εργαλείο SDN Narmox Spear υπάρχουν στην υποενότητα 7.4 του παραρτήματος.

4.1.2 Εντολές διαχείρισης του Mininet (Command-line Interface, CLI)

Με το Mininet μπορούμε να δημιουργήσουμε στοιχεία των SDN δικτύων όπως κόμβους, ενδιάμεσους κόμβους, ακμές ή διαχειριστές δικτύων, να τα επεξεργαστούμε ή να τα χρησιμοποιήσουμε για αλληλεπίδραση μεταξύ διαφόρων δικτύων. Μερικές σημαντικές εντολές για την βασική χρήση του mininet είναι οι εξής:

- mn --topo single,3 --mac --switch ovsk --controller remote. Ένα παράδειγμα δημιουργίας μιας τοπολογίας. Με την εντολή αυτή δημιουργούνται 3 εικονικοί κόμβοι οι οποίοι συνδέονται με έναν μεταγωγέα. Ακόμα η εντολή αυτή θέτει τις Mac και IP διευθύνσεις κάθε κόμβου και ρυθμίζει τον μεταγωγέα ώστε να επικοινωνεί με έναν απομακρυσμένο διαχειριστή δικτύου ο οποίος μπορεί και να τρέχει τοπικά στην ίδια προσομοίωση του Mininet.
- nodes: Εμφανίζει μια λίστα με όλους τους κόμβους του δικτύου.
- h1 ifconfig: Εμφανίζει πληροφορίες για την IP διεύθυνση του κόμβου, στην προκειμένη περίπτωση του h1.
- h1 ping h2: Στέλνει πακέτο (ICMP REQUEST) από τον κόμβο h1 στον κόμβο h2.

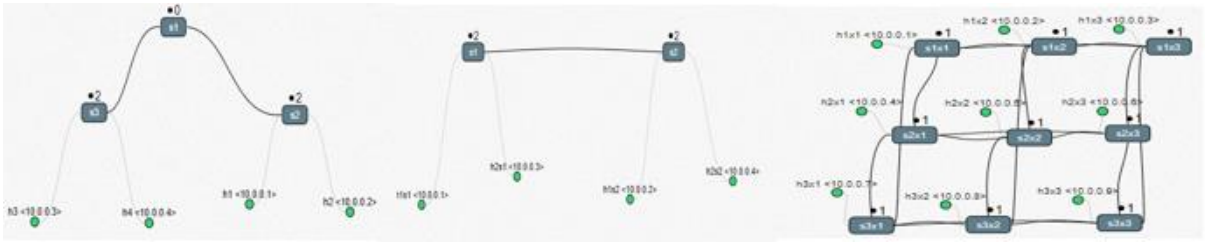
Την υποενότητα 7.6 του παραρτήματος περιγράφουμε με περισσότερες λεπτομέρειες μερικές από τις πιο χρήσιμες εντολές του εργαλείου Mininet.

4.1.3 Είδη μορφών τοπολογιών που υποστηρίζει το Mininet

Όπως προαναφέρθηκε, με το Mininet μπορούμε να φτιάξουμε εύκολα τοπολογίες [29] με την χρήση μίας μόνο εντολής όπως για παράδειγμα sudo mn --topo single,3 --mac --controller remote --switch ovsk. Η παράμετρος --topo καθορίζει την μορφή που θα έχει το δίκτυο που πρόκειται να δημιουργηθεί και μερικές από τις μορφές τοπολογιών που θα χρησιμοποιήσουμε είναι οι εξής:

- Single: Στην τοπολογία αυτή δημιουργείται μόνο ένα μεταγωγέας και κόμβοι ίσοι με το όρισμα που δίνεται.
- Linear: Στη τοπολογία αυτή δημιουργούνται μεταγωγείς που συνδέονται μεταξύ τους γραμμικά οι οποίοι θα έχουν ο καθένας κόμβους/παιδία. ο αριθμός των κόμβων και των μεταγωγέων καθορίζεται από όρισμα του χρήστη.
- Tree: Με την τοπολογία αυτή δημιουργείται δίκτυο με μορφή δέντρου του οποίου το ύψος και ο αριθμός κόμβων/φύλλων καθορίζονται από όρισμα του χρήστη.

- **Torus:** Με την τοπολογία αυτή δημιουργείται τοπολογία με ομάδες από μεταγωγείς ο καθένας από τους οποίους θα έχει έναν κόμβο/παιδί. Το πλήθος των ομάδων καθώς και ο αριθμός των μεταγωγέων που τις αποτελούν καθορίζονται από όρισμα του χρήστη.



Εικόνα 10: Παραδείγματα τοπολογιών δέντρου, γραμμικών και Torus

4.2 SDN Controller setup

4.2.1 Προγραμματισμός POX controller σε python

Ο POX είναι ένας διαχειριστής δικτύων ανοικτού κώδικα γραμμένος σε γλώσσα προγραμματισμού python που χρησιμοποιείται για κατασκευή SDN δικτύων. Παρέχει έναν αποτελεσματικό τρόπο υλοποίησής του openflow πρωτοκόλλου ανάμεσα στους μεταγωγείς και τους διαχειριστές δικτύων, χρησιμοποιώντας διάφορες συμπεριφορές για τους διαχειριστές όπως hub, switch και άλλα [30]. Αρχικά, για τον προγραμματισμό του διαχειριστή δημιουργούμε ένα python αρχείο στο directory pox/pox/misc (vi basic_controller.py). Δημιουργούμε πρώτα μια κλάση Switch με την οποία θα αντιστοιχεί κάθε μεταγωγέας που θα έχουν οι τοπολογίες μας.

Στον κατασκευαστή (constructor) της κλάσης μας θα ορίσουμε ένα αντικείμενο connection το οποίο θα χρησιμοποιεί ο διαχειριστής για να επικοινωνήσει με το συγκεκριμένο μεταγωγέα καθώς και έναν listener για να «ακούει» τα πακέτα που προέρχονται από το μεταγωγέα αυτό. Επιπροσθέτως δημιουργούμε και μία κενή λίστα την οποία θα χρησιμοποιούμε για να αντιστοιχίζουμε τις mac διευθύνσεις των κόμβων της τοπολογίας με τις θύρες του μεταγωγέα.

```
from pox.core import core
import pox.openflow.libopenflow_01 as of

log = core.getLogger()

class Switch (object):
  """
  A Switch object is created for each switch that connects.
  A Connection object for that switch is passed to the __init__ function.
  """
  def __init__ (self, connection):
    # Keep track of the connection to the switch so that we can
    # send it messages!
    self.connection = connection

    # This binds our PacketIn event listener
    connection.addListeners(self)

    # Use this table to keep track of which ethernet address is on
    # which switch port (keys are MACs, values are ports).
    self.mac_to_port = {}
```

Εικόνα 11: Default constructor της κλάσης Switch στον controller μας

Θα χρειαστεί επίσης να δημιουργήσουμε μεθόδους που να αναλαμβάνουν τα packet_in πακέτα και τις απαντήσεις του controller στα switch. Για την αντιμετώπιση των packet_in πακέτων δημιουργούμε την μέθοδο handel_PacketIn. Εδώ απλά θα καλούμαι τη μέθοδο που θέλουμε εμείς να αντιμετωπίζει το switch τα πακέτα κάθε φορά. Έπειτα προσθέτουμε και τη μέθοδο resend_packet η οποία δέχεται ως όρισμα το packet_in πακέτο καθώς και την θύρα στην οποία ο controller θα δώσει οδηγία στο switch να στείλει το πακέτο που έλαβε. Εδώ δημιουργούμε πακέτα τύπου packet_out που συμπεριλαμβάνουν το μήνυμα του packet_in που έλαβε ο controller καθώς και τη οδηγία προώθησης που δώσει ο controller στο switch.

```
def resend_packet (self, packet_in, out_port):
    """
    Instructs the switch to resend a packet that it had sent to us.
    "packet_in" is the ofp_packet_in object the switch had sent to the
    controller due to a table-miss.
    """
    msg = of.ofp_packet_out()
    msg.data = packet_in

    # Add an action to send to the specified port
    action = of.ofp_action_output(port = out_port)
    msg.actions.append(action)

    # Send message to switch
    self.connection.send(msg)
```

Εικόνα 12: Μέθοδος resend_packet της κλάσης του controller

Μία από τις μεθόδους που θα εξετάσουμε σε αυτή την ενότητα είναι η hub συμπεριφορά των switch, με την οποία να προωθούν τα πακέτα τους προς όλες τους τις θύρες εκτός από την θύρα προέλευσης του πακέτου. Δημιουργούμε λοιπόν την μέθοδο **act_like_hub**, με την οποία θέλουμε απλά ο controller να δίνει οδηγία να προωθούνται τα πακέτα προς όλες τις θύρες των switch. Εδώ απλά καλούμε την **resend_packet** μέθοδο με όρισμα για την θύρα προορισμού την λέξη κλειδί **OFPP_ALL** που μεταφράζεται «σε όλες τις θύρες».

```
def act_like_hub (self, packet, packet_in):
    """
    Implement hub-like behavior -- send all packets to all ports besides
    the input port.
    """

    # We want to output to all ports -- we do that using the special
    # OFPP_ALL port as the output port.  (We could have also used
    # OFPP_FLOOD.)
    self.resend_packet(packet_in, of.OFPP_ALL)
```

Εικόνα 13: Μέθοδος act_like_hub της κλάσης του controller

Η άλλη συμπεριφορά που θα μελετήσουμε είναι η συμπεριφορά **switch** με την οποία το switch προωθεί το πακέτο μόνο προς την θύρα προορισμού. Για τη συμπεριφορά αυτή Προσθέτουμε στην κλάση του controller μας την μέθοδο **act_like_switch.** Εδώ θα χρησιμοποιούμε την λίστα με τις διευθύνσεις των κόμβων που ορίσαμε προηγουμένως. Θα ελέγχουμε για κάθε εισερχόμενο πακέτο εάν η διεύθυνση προορισμού του είναι γνωστή, τότε θα προωθούμε το πακέτο μόνο προς αυτή ενώ εάν δεν είναι γνωστή την προωθούμε προς όλες τις θύρες.

```
def act_like_switch (self, packet, packet_in):

    # Implement switch-like behavior.
    # Here's some psuedocode to start you off implementing a learning
    # switch.  You'll need to rewrite it as real Python code.

    # Learn the port for the source MAC

    if (packet.src not in self.mac_to_port):
        #... <add or update entry>
      print ("Learning " + str(packet.src) + " who is listening to port " + str(packet_in.in_port))
      self.mac_to_port[packet.src] = packet_in.in_port
    #if the port associated with the destination MAC of the packet is known:
      # Send packet out the associated port

    if (packet.dst in self.mac_to_port):
      print(str(packet.dst) + " known destination, sending message just to it")
      self.resend_packet(packet_in, self.mac_to_port[packet.dst])
      #instead of resending the packet (comment out the above and
      #uncomment and complete the below.)
    else:
      print(str(packet.dst) + " unknown desetination, sending message to everyone")
      self.resend_packet(packet_in, of.OFPP_ALL)
```

Εικόνα 14: Μέθοδος act_like_switch της κλάσης του controller

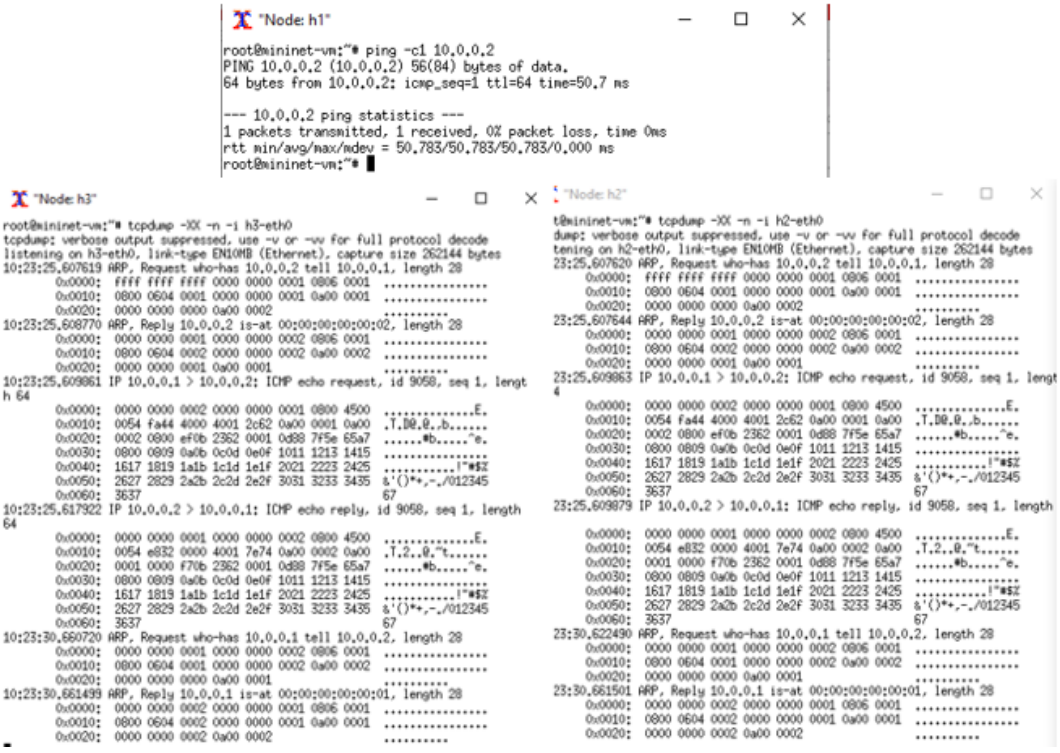
4.2.2 Controller με συμπεριφορά hub

Δημιουργούμε τοπολογία δικτύου με 3 κόμβους, 1 μεταγωγέα και 1 απομακρυσμένο διαχειριστή Αφού μεταβούμε στο directory που είναι εγκατεστημένη η pox (**cd pox**) τρέχουμε τον διαχειριστή με την εντολή **./pox.py log.level --DEBUG misc.basic_controller.**

```
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.of_tutorial
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.1.0 (betta) going up...
DEBUG:core:Running on CPython (2.7.6/Oct 26 2016 20:32:47)
DEBUG:core:Platform is Linux-4.2.0-27-generic-i686-with-Ubuntu-14.04-trusty
INFO:core:POX 0.1.0 (betta) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 1]
```

Εικόνα 15: Ενεργοποίηση διαχειριστή POX

Για να ελέγξουμε την hub συμπεριφορά του διαχειριστή ανοίγουμε ένα παράθυρο xterm για κάθε έναν από τους κόμβους (**xterm h1 h2 h3**) και εκτελούμε **tcmpdump** στους h2 h3 όπως φαίνεται στις εικόνες για να παρακολουθούμε τι διακίνηση πακέτων στο κάθε κόμβο του δικτύου. Στον κόμβο h1 κάνουμε ping στον h2 και βλέπουμε τα πιο κάτω (εικόνα 16) αποτελέσματα:



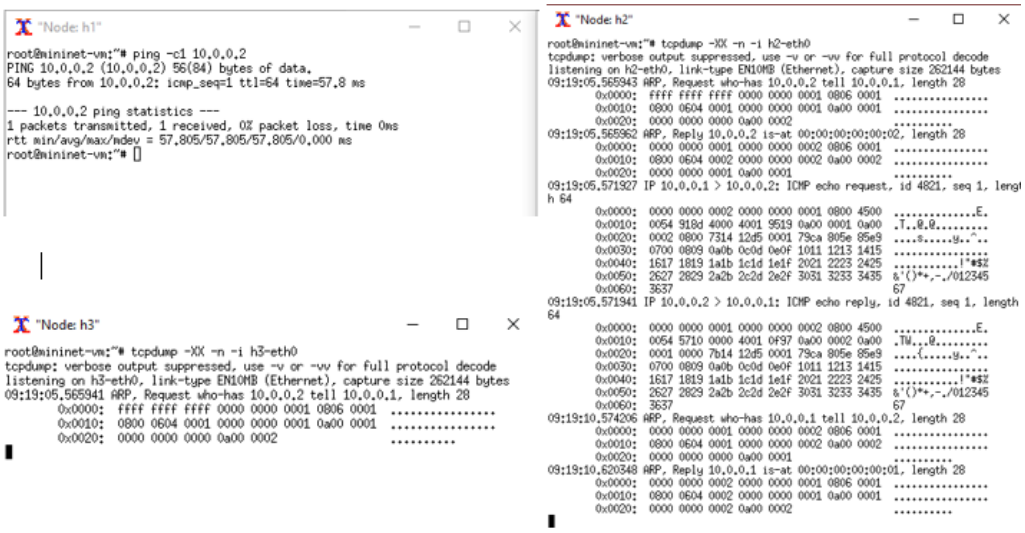
Εικόνα 16: Hub συμπεριφορά του διαχειριστή

Παρατηρούμε ότι τα πακέτα στάλθηκαν και στις 2 θύρες που “ακούνε” οι 2 hosts όπως είχαμε περιγράψει κατά την υλοποίηση της μεθόδου **act_like_hub()**.Ο controller με συμπεριφορά hub δίνει οδηγία να σταλούν τα πακέτα που λαμβάνει το switch προς όλες τις θύρες του. Είναι όμως εμφανές ότι η συμπεριφορά hub ενώ είναι εύκολη σε υλοποίηση, δεν είναι βέλτιστη για το δίκτυο μας εφόσον όλοι οι hosts λαμβάνουν πακέτα που δεν προορίζονται γι’ αυτούς.

4.2.3 Controller με συμπεριφορά switch

Μια πιο αποτελεσματική συμπεριφορά για τον controller είναι η συμπεριφορά switch. Όπως αναφέραμε με την συμπεριφορά αυτή θέλουμε να στέλνουμε τα κάθε πακέτο μόνο στην διεύθυνση ip για την οποία προορίζεται και όχι σε όλους τους hosts παρά μόνο εάν η διεύθυνση αυτή δεν είναι γνωστή. Καλούμε λοιπόν την μέθοδο **act_like_switch** και αφού αποθηκεύσουμε τις αλλαγές μας, τρέχουμε τον controller όπως προηγουμένως και στα 3 xterm παράθυρα για την ίδια τοπολογία με πριν, εκτελούμε πάλι τις εντολές.

tcpdump –XX –n –i h2-eth0 στον h2
tcpdump –XX –n –i h3-eth0 στον h3
ping –c1 10.0.0.2 στον h1



Εικόνα 17: Switch συμπεριφορά του διαχειριστή

Παρατηρούμε τώρα πως εκτός από το πακέτο arp-request που γίνεται broadcast σε όλους, ο host 3 δεν βλέπει τα πακέτα του ping αφού δεν προορίζονται γι’ αυτόν. Αν προσπαθήσουμε να κάνουμε ping σε μια διεύθυνση που δεν υπάρχει φαίνεται πιο ξεκάθαρα ότι και οι 2 hosts λαμβάνουν τα μηνύματα ARP-request.

Εικόνα 18: Συμπεριφορά switch όταν ο διαχειριστής δεν γνωρίζει τον προορισμό ενός πακέτου

4.2.4 Δημιουργία ροών μεταξύ κόμβων

Μέχρι τώρα ο controller στις μεθόδους **act_like_hub()** και **act_like_switch()** ήταν προγραμματισμένος ώστε να απαντά στο switch με πακέτα τύπου **Packet_out** δίνοντας του την εντολή να τα στέλνει είτε σε συγκεκριμένη διεύθυνση είτε να τα κάνει broadcast σε όλες τις γνωστές στο switch διευθύνσεις. Η επικοινωνία μεταξύ hosts όμως μπορεί να γίνει ακόμα πιο γρήγορη αν δημιουργούμε flows για τα εισερχόμενα στο switch πακέτα κάθε φορά **[31].** Τροποποιούμε τον κώδικα της **act_like_switch** ώστε να στέλνει flow_mod πακέτο στο switch που να δημιουργεί flow με timeout 40 δευτερολέπτων και να αντιστοιχεί πακέτα παρόμοια με αυτό που μόλις έλαβε το switch.

```
def act_like_switch (self, packet, packet_in):

    # Implement switch-like behavior.
    # Here's some psuedocode to start you off implementing a learning
    # switch.  You'll need to rewrite it as real Python code.

    # Learn the port for the source MAC

    if (packet.src not in self.mac_to_port):
        #... <add or update entry>
      print ("Learning " + str(packet.src) + " who is listening to port " + str(packet_in.in_port))
      self.mac_to_port[packet.src] = packet_in.in_port
    #if the port associated with the destination MAC of the packet is known:
      # Send packet out the associated port

    if (packet.dst in self.mac_to_port):
      out_port = self.mac_to_port[packet.dst]
      msg = of.ofp_flow_mod()
      msg.match = of.ofp_match.from_packet(packet)
      msg.idle_timeout = 30
      msg.hard_timeout = 40
      msg.buffer_id = packet_in.buffer_id
      action = of.ofp_action_output(port = out_port)
      msg.actions.append(action)
      self.connection.send(msg)

    else:
      print(str(packet.dst) + " unknown desetination, sending message to everyone")
      self.resend_packet(packet_in, of.OFPP_ALL)
```

Εικόνα 19: Κώδικας python για προσθήκη ροών στους μεταγωγείς

Δημιουργούμε την προηγούμενη τοπολογία και κάνουμε ping από τον h1 στον h2. Εάν εκτελέσουμε μετά την ολοκλήρωση του ping την εντολή sh ovs-ofctl dump-flows s1 παρατηρούμε ότι τώρα υπάρχει flow υπάρχουν flows στο flow table του switch.

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=2.332s, table=0, n_packets=1, n_bytes=98, idle_timeout=10, hard_timeout
=15, idle_age=2, icmp,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,nw_sr
c=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:2
 cookie=0x0, duration=2.33s, table=0, n_packets=1, n_bytes=98, idle_timeout=10, hard_timeout=
15, idle_age=2, icmp,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw_src
=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:1
mininet>
```

Εικόνα 20: Επιτυχής προσθήκη flow στο flow table μετά το ping μεταξύ κόμβων

Εκτελώντας επίσης την εντολή iperf παρατηρούμε ότι το αποτέλεσμα της τώρα σε σχέση με πριν έχει πολύ μεγαλύτερο bandwidth. Αυτό συμβαίνει επειδή πλέον με την υλοποίηση αυτή δεν στέλνονται OpenFlow packet-in μηνύματα από το switch στον controller για κάθε εισερχόμενο πακέτο.

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['11.9 Gbits/sec', '11.9 Gbits/sec']
```

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['7.24 Mbits/sec', '7.92 Mbits/sec']
```

Εικόνα 21:Σύγκριση bandwidth με χρήση ροών (αριστερά) και χωρίς (δεξιά)

Η εντολή iperf είναι ένα εργαλείο που χρησιμοποιείται για τον υπολογισμό του μέγιστου bandwidth που μπορεί να επιτευχθεί σε μια ζεύξη ενός δικτύου. Για να μετρήσει το bandwidth, η εντολή εκτελεί διάφορες TCP (Transmission Control Protocol) δοκιμές μεταξύ δύο κόμβων και έτσι κατά την εκτέλεσή του στέλνεται μεγάλος αριθμός πακέτων. Χωρίς τις ροές με την χρήση της εντολής

iperf για κάθε πακέτο που λαμβάνει ο μεταγωγέας παράγεται Packet In πακέτο που προορίζεται για τον διαχειριστή και ένα Packet Out πακέτο που παράγει ο διαχειριστής για τον μεταγωγέα ως απάντηση για το πού να στείλει το πακέτο που έλαβε. Αυτό έχει ως αποτέλεσμα την μεγάλη openflow κίνηση μέσα στο δίκτυο. Αντίθετα προσθέτοντας ροές στον μεταγωγέα, ώστε να γνωρίζει προς τα πού να προωθήσει τα πακέτα χωρίς να «ρωτήσει» τον διαχειριστή παρατηρούμε τεράστια διαφορά στον αριθμό των openflow πακέτων που παράγονται. Στην πιο κάτω (εικόνα 22) εικόνα φαίνονται οι ροές που δημιουργούνται κατά την εκτέλεση της εντολής iperf καθώς και ο μεγάλος αριθμός πακέτων που αντιστοιχήθηκαν με αυτές.

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=8.345s, table=0, n_packets=4, n_bytes=272, idle_timeout=30
, hard_timeout=40, idle_age=8, tcp,vlan_tci=0x0000,dl_src=0a:3e:bd:52:c8:20,dl_d
st=be:9a:1f:31:1e:e8,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw_tos=16,tp_src=52798,tp_d
st=5001 actions=output:3
 cookie=0x0, duration=8.319s, table=0, n_packets=78717, n_bytes=5195330, idle_ti
meout=30, hard_timeout=40, idle_age=3, tcp,vlan_tci=0x0000,dl_src=be:9a:1f:31:1e
:e8,dl_dst=0a:3e:bd:52:c8:20,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_tos=0,tp_src=500
1,tp_dst=52800 actions=output:1
 cookie=0x0, duration=8.321s, table=0, n_packets=144909, n_bytes=7212887930, idl
e_timeout=30, hard_timeout=40, idle_age=3, tcp,vlan_tci=0x0000,dl_src=0a:3e:bd:5
2:c8:20,dl_dst=be:9a:1f:31:1e:e8,nw_src=10.0.0.1,nw_dst=10.0.0.3,nw_tos=0,tp_src
=52800,tp_dst=5001 actions=output:3
 cookie=0x0, duration=8.343s, table=0, n_packets=3, n_bytes=206, idle_timeout=30
, hard_timeout=40, idle_age=8, tcp,vlan_tci=0x0000,dl_src=be:9a:1f:31:1e:e8,dl_d
st=0a:3e:bd:52:c8:20,nw_src=10.0.0.3,nw_dst=10.0.0.1,nw_tos=0,tp_src=5001,tp_dst
=52798 actions=output:1
mininet>
```

Εικόνα 22: Τα flows που δημιουργεί η χρήση της iperf.

4.3 Συγκρίσεις μετρήσεων

4.3.1 Ανάπτυξη διαχειριστή για μετρήσεις ποιότητας ζεύξεων σε SDN δίκτυο

Αφού έχουμε παρατηρήσει τους διάφορους τρόπους επικοινωνίας των μεταγωγέων με τους κόμβους σε ένα δίκτυο καλούμαστε τώρα να χρησιμοποιήσουμε τις λειτουργίες και τα προτερήματα του Openflow πρωτοκόλλου ώστε να μπορούμε να συλλέγουμε τις απαραίτητες πληροφορίες από τα δίκτυα αρχιτεκτονικής SDN με το μικρότερο κόστος. Φτιάξαμε λοιπόν μια υλοποίηση ενός διαχειριστή στην γλώσσα προγραμματισμού python χρησιμοποιώντας την πλατφόρμα pox προκειμένου να λαμβάνουμε μερικές σημαντικές μετρήσεις μεταξύ των ζεύξεων όπως τον αριθμό πακέτων που μεταφέρονται καθώς και τον αριθμό των bytes που στέλνονται σε συγκεκριμένα χρονικά πλαίσια. Έτσι μπορούμε να υπολογίσουμε μεγέθη όπως throughput και packet loss. Χρησιμοποιούμε επίσης διάφορες χρονοσφραγίδες προκειμένου να είμαστε σε θέση να υπολογίσουμε κα το delay.

Για την υλοποίηση της προτεινόμενης μας λύσης, επεκτείναμε την υλοποίηση των μαθησιακών μεταγωγέων (learning switches), μια από τις αρκετές έτοιμες εκδοχές Openflow διαχειριστών που μας παρέχει η πλατφόρμα POX. Συγκεκριμένα χρησιμοποιήθηκε το python αρχείο l2_learning.py [32] το

οποίο φροντίζει ώστε οι μεταγωγείς μιας τοπολογίας να μαθαίνουν και να αποθηκεύουν σε κάποιο πίνακα όλες τις IP διευθύνσεις των γνωστών σε αυτούς κόμβων καθώς και την αντίστοιχα θύρα τους από την οποία μπορούν να επικοινωνήσουν μαζί τους.

Ουσιαστικά ακολουθώντας την υλοποίηση της πηγής [33] προσθέσαμε τον κατάλληλο κώδικα στο προαναφερθέν αρχείο ώστε να στέλνονται πακέτα τύπου FlowStatsReceived στον πρώτο μεταγωγέα του μονοπατιού που εξετάζουμε ανά τακτά χρονικά διαστήματα, την συχνότητα των οποίων ρυθμίζουμε εμείς με την μέθοδο timer_func. Προσθέτουμε κατόπιν event listener που «ακούει» πότε ο μεταγωγέας θα απαντήσει με FlowStatsReceived, την απάντηση του οποίου επεξεργαζόμαστε στη μέθοδο _handle_flowstats_received. Για τις μετρήσεις μας χρησιμοποιούμε τις εξής μεταβλητές που περιέχει το πακέτο FlowStatsReceived:

- byte_count: Επιστρέφει τον αριθμό των bytes που αντιστοιχήθηκαν με τη συγκεκριμένη ροή μέχρι τώρα.
- packet_count: Επιστρέφει τον αριθμό των packets που αντιστοιχήθηκαν με τη συγκεκριμένη ροή μέχρι τώρα.
- duration_sec: Επιστρέφει τον χρόνο σε δευτερόλεπτα που η συγκεκριμένη ροή είναι ενεργή.
- duration_nsec: Επιστρέφει τον επιπλέον χρόνο σε nano-seconds που είναι ενεργή η συγκεκριμένη ροή.

Με τα πιο πάνω δεδομένα που συγκεντρώνει ο διαχειριστής μέσω τον πακέτων FlowStatsReceived, μαθαίνει τον αριθμό των bytes που αντιστοιχήθηκαν για κάθε ροή με την μεταβλητή byte_count και το χρονικό διάστημα κατά τον οποίο η κάθε ροή ήταν ενεργή με τις μεταβλητές duration_sec και duration_nsec. Οπότε το μέγεθος Throughput για κάθε ενεργή ροή προκύπτει από το πηλίκο των αριθμό των bytes διά την συνολική διάρκεια χρόνου που ήταν ενεργή κάθε ροή (δηλαδή $duration_{\sec[i]} + \frac{duration_{nsec[i]}}{10^9}$ εφόσον η τιμή της μεταβλητής duration_nsec είναι σε nano seconds). Επομένως προκειμένου να έχουμε τη στιγμιαία τιμή του Throughput με μορφή τιμών (i, i-1) προκύπτει και η σχέση της πηγής [33]:

$$Throughput = \frac{byte\_counts[i] - byte\_counts[i-1]}{duration_{\sec[i]} + \frac{duration_{nsec[i]}}{10^9} - (duration_{\sec[i-1]} + \frac{duration_{nsec[i-1]}}{10^9})}$$

Με παρόμοιο τρόπο με τον πιο πάνω ρυθμίσαμε τον διαχειριστή μας ώστε να στέλνει και πακέτα τύπου PortStatsRequest, αυτή τη φορά όμως στον πρώτο και τελευταίο μεταγωγέα της διαδρομής. Αφού προσθέσαμε έναν event listener μπορούμε να «ακούσουμε» τις απαντήσεις των μεταγωγέων και με τα πακέτα PortStatsReceived τις οποίες επεξεργαζόμαστε στην μέθοδο _handle_portstats_received. Στην προαναφερθείσα μέθοδο χρησιμοποιούμε τις πιο κάτω μεταβλητές:

- port_no: Επιστρέφει τον αριθμό που αντιστοιχεί στην συγκεκριμένη διεπαφή του μεταγωγέα.
- tx_packets: Επιστρέφει τον αριθμό των πακέτων που στάλθηκαν από την συγκεκριμένη διεπαφή.
- rx_packets: Επιστρέφει τον αριθμό των πακέτων που παρέλαβε ο μεταγωγέας μέσω της συγκεκριμένης διεπαφής του.

Εφόσον μέσω της μεθόδου επεξεργασίας των στοιχείων κάθε διεπαφής έχουμε πρόσβαση στον αριθμό των πακέτων που στάλθηκαν και παραλήφθηκαν από κάθε διεπαφή κάθε μεταγωγέα μπορούμε εύκολα να υπολογίσουμε το **packet loss** μιας επικοινωνίας στο δίκτυο χρησιμοποιώντας τις μετρήσεις από τους δύο απομακρυσμένους μεταγωγείς τις ζεύξης. Δηλαδή με την τιμή των μεταβλητών **tx_packets** και **rx_packets** γνωρίζουμε ανά πάσα στιγμή τον αριθμό των πακέτων (σε bytes) που στάλθηκαν από τον αποστολέα και τον αριθμό των πακέτων που παρέλαβε ο παραλήπτης αντίστοιχα. Επομένως το μέγεθος packet loss προκύπτει από τον λόγο των απορριφθέντων πακέτων προς τον συνολικό αριθμό πακέτων που έστειλε η πηγή. Έτσι προκύπτει ο εξής τύπος από την πηγή **[33]:**

$$packet\_loss = \frac{tx\_packets(source) - rx\_packets(destination)}{tx\_packets(source)}$$

Τέλος για τον υπολογισμό του Delay χρησιμοποιήσαμε και πάλι τη μέθοδο υπολογισμού Delay της πηγής **[33].** Σύμφωνα με την πηγή ρυθμίσαμε τον διαχειριστή μας ώστε να στέλνει ένα πακέτο τύπου **PacketIn** στον πρώτο μεταγωγέα του μονοπατιού που ελέγχουμε, με προορισμό τον τελευταίο μεταγωγέα του μονοπατιού. Χρησιμοποιούμε χρονοσφραγίδες και μετράμε τον χρόνο αποστολής καθώς και τον χρόνο παραλαβής του πακέτου αυτού. Για τον υπολογισμό του delay αρκεί να βρούμε την διαφορά των δύο χρόνων και έπειτα από αυτή να αφαιρέσουμε τον χρόνο που χρειάζεται το πακέτο για να φτάσει από τον διαχειριστή στον πρώτο μεταγωγέα και τον αντίστοιχο χρόνο για τον τελευταίο μεταγωγέα. Τον χρόνο αυτό υπολογίζουμε χρησιμοποιώντας χρονοσφραγίδες για τον χρόνο αποστολής των πακέτων **PortStatsRequest** και τον χρόνο παραλαβής των πακέτων **PortStatsReceived**. Η τιμή που ψάχνουμε ισούται προφανώς με το μισό της διαφοράς των δύο χρόνων. Συνοψίζοντας, η σχέση για τον υπολογισμό του Delay της πηγής **[33]** προκύπτει ως εξής:

$$Delay = PacketIn_{arrival\_time} - PacketIn_{send\_time} - T_1 - T_2$$

Όπου T_1 και T_2 οι δύο χρόνοι που περιγράψαμε πιο πάνω για τον πρώτο και τελευταίο μεταγωγέα της διαδρομής αντίστοιχα.

Ο κώδικας της προτεινόμενης μας λύσης φαίνεται στην υποενότητα 7.7 του παραρτήματος.

4.4.2 Σύγκριση bandwidth

4.4.2.1 Χρήση της υπηρεσίας πελάτη εξυπηρετητή της εντολής iperf

Χρησιμοποιώντας λοιπόν τον κώδικα του διαχειριστή που φτιάξαμε στην προηγούμενη ενότητα θα συγκρίνουμε τώρα το bandwidth των ζεύξεων ενός δικτύου για να παρατηρήσουμε την ακρίβεια που θα έχουν οι μετρήσεις που θα εξάγουμε. Προκειμένου να εξακριβώσουμε τον βαθμό ακρίβειας των μετρήσεών μας θα δημιουργήσουμε μια ελεγχόμενη τοπολογία με 3 κόμβους και έναν μεταγωγέα της οποίας όλες οι ακμές θα έχουν bandwidth = 10Mbps. Στη συνέχεια θα συγκρίνουμε τις μετρήσεις μας

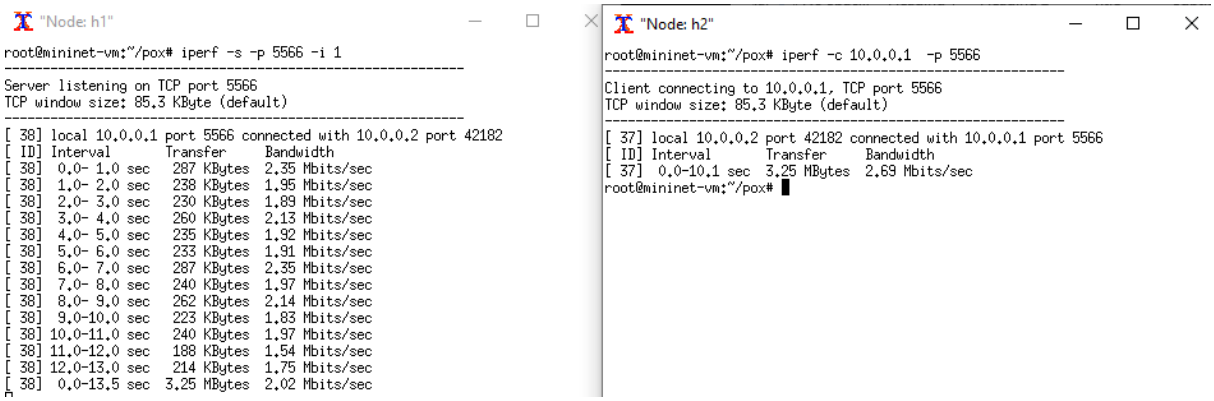
με το αποτέλεσμα της εντολής **iperf** χρησιμοποιώντας την υπηρεσία πελάτη – εξυπηρετητή **[34]** όπως φαίνεται πιο κάτω:

- Ανοίγουμε δύο παράθυρα xterm για τους δύο κόμβους.

  xterm h1 h2
- Στον πρώτο κόμβο ξεκινούμε τον iperf εξυπηρετητή, διαλέγουμε θύρα στην οποία ακούει και δηλώνουμε interval = 1 δευτερόλεπτο.

  iperf –s –p 5566 –i 1
- Στον δεύτερο κόμβο ξεκινούμε τον iperf πελάτη και περνάμε σαν όρισμα την διεύθυνση του εξυπηρετητή και την θύρα στην οποία ακούει.

  iperf –c 10.0.0.1 -p 5566

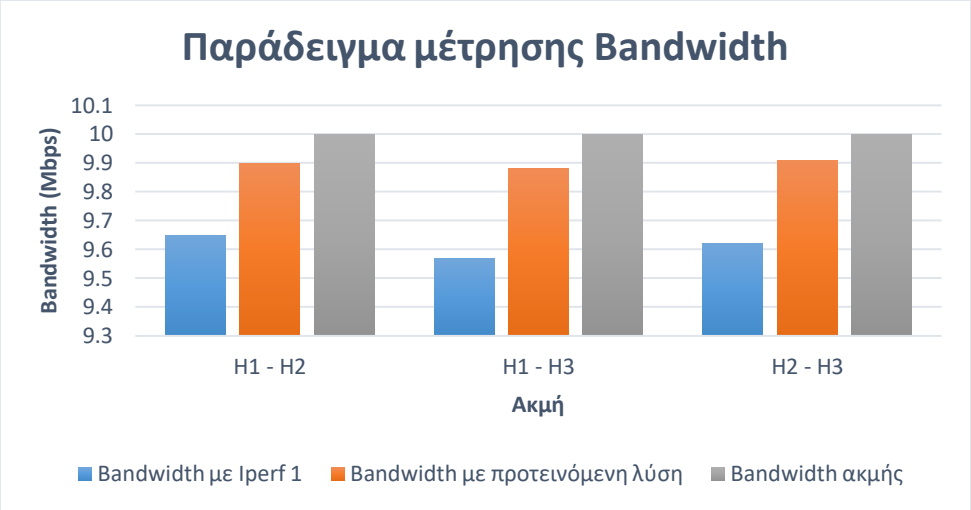


Εικόνα 23: Παράδειγμα χρήσης του iperf πελάτη – εξυπηρετητή

Για την επίδειξη της λειτουργίας του εργαλείου Iperf καθώς και της προτεινόμενής μας λύσης συγκεντρώσαμε στον πίνακα 2 το bandwidth κάθε ακμής στην τοπολογία τριών κόμβων συνδεδεμένων με έναν μεταγωγέα:

	Μετρήσεις εντολής iperf (Mbps)	Μετρήσεις bandwidth με την προτεινόμενη λύση (Mbps)
H1-H2	9.65	9.90
H1-H3	9.57	9.88
H2-H3	9.62	9.91

Πίνακας 2: Πίνακας σύγκρισης bandwidth μεταξύ της εντολής iperf και προτεινόμενης λύσης



Εικόνα 24: Παράδειγμα σύγκρισης τιμών bandwidth μεταξύ Iperf και προτεινόμενης λύσης

Στον πίνακα 2 παρουσιάσαμε τις μετρήσεις που πήραμε για Bandwidth με τη χρήση της υπηρεσίας πελάτη-εξυπηρετητή του εργαλείου Iperf και την προτεινόμενη λύση και στο γράφημα της εικόνας 24 συγκρίνουμε την ακρίβεια των δύο εργαλείων με το ρυθμισμένο Bandwidth των ακμών. Από την γραφική παράσταση φαίνεται πως σε κάθε ακμής της τοπολογίας η μέτρηση της προτεινόμενης λύσης ήταν ακριβέστερη από την αντίστοιχη του εργαλείου Iperf. Η σύγκριση αυτή αποτελεί παράδειγμα αξιολόγησης της προτεινόμενης μας λύσης εφόσον πρόκειται να διεξάγουμε μεγαλύτερα και πιο πολύπλοκα πειράματα για τον έλεγχο της ακρίβειάς της στις παρακάτω ενότητες.

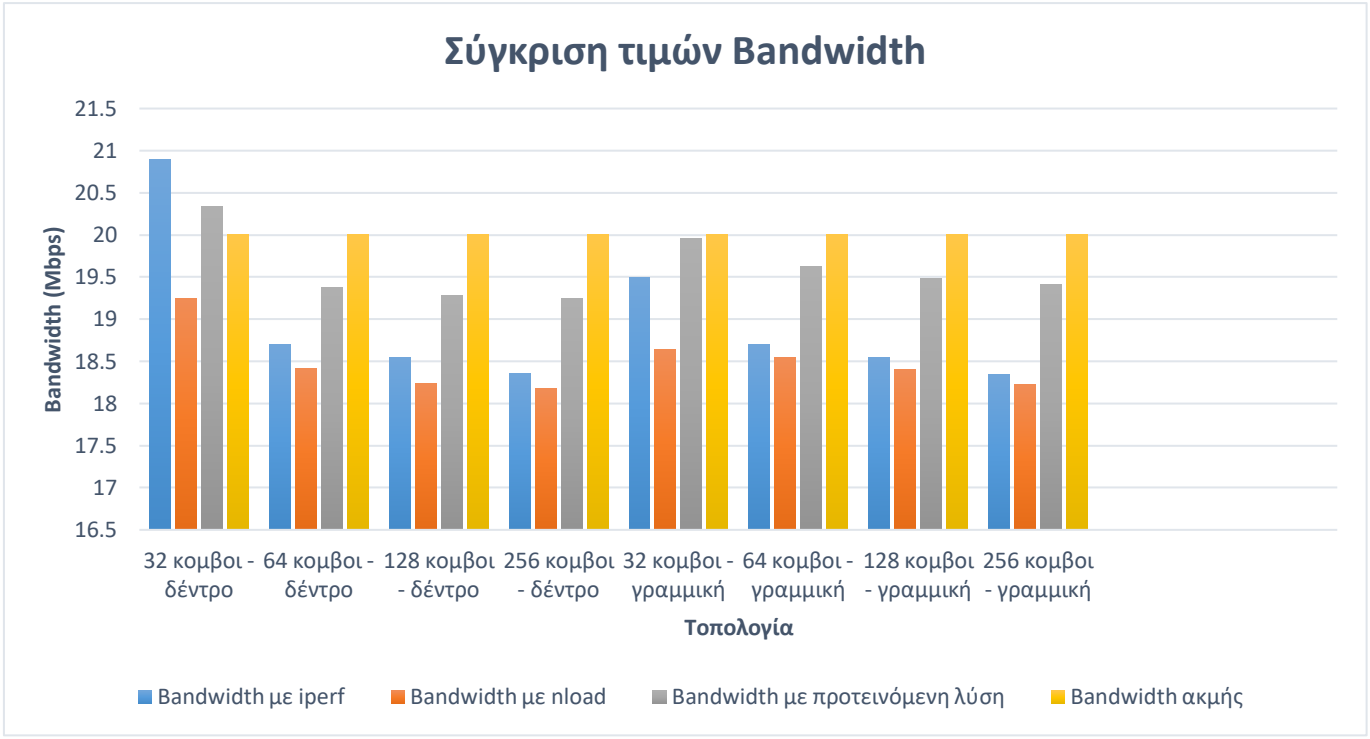
4.4.2.2 Συγκρίσεις Bandwidth με εργαλείο Iperf και nload

Σε αυτή την ενότητα θα συγκρίνουμε τις μετρήσεις για το Bandwidth της προτεινόμενής μας λύσης με ορισμένα εργαλεία που χρησιμοποιούνται συγκεκριμένα για υπολογισμό Bandwidth σε διάφορες μορφές τοπολογιών. Ακολουθώντας την μεθοδολογία της πηγής [33] φτιάχνουμε ελεγχόμενες τοπολογίες με συγκεκριμένα χαρακτηριστικά ακμών ώστε να συγκρίνουμε το αποτέλεσμα της προτεινόμενης λύσης με κάποια αναμενόμενη τιμή. Φτιάχνουμε λοιπόν γραμμικές τοπολογίες και τοπολογίες μορφής δέντρου διαφόρων μεγεθών οι ακμές των οποίων θα είναι προγραμματισμένες ώστε να έχουν ρυθμισμένο Bandwidth = 20 Mbps . Για τις μετρήσεις μας θα εξετάσουμε το Bandwidth μεταξύ των πλέον απομακρυσμένων κόμβων κάθε τοπολογίας. Για σύγκριση τιμών θα χρησιμοποιηθεί το εργαλείο Iperf καθώς και το linux εργαλείο nload [35], μια εφαρμογή κονσόλας που παρακολουθεί την κίνηση του δικτύου και το εύρος ζώνης σε πραγματικό χρόνο, οπτικοποιεί την εισερχόμενη και εξερχόμενη κίνηση χρησιμοποιώντας δύο γραφήματα και παρέχει πρόσθετες πληροφορίες όπως

συνολικό ποσό μεταφερόμενων δεδομένων και ελάχιστη ή μέγιστη χρήση δικτύου. Στον πίνακα 3 καταγράψαμε τον μέσο όρο 10 μετρήσεων Bandwidth για κάθε περίπτωση τοπολογίας για τα 3 εργαλεία.

Μορφή τοπολογίας (2 παιδιά ανά μεταγωγέα)	Μετρήσεις bandwidth μέσω εντολής iperf (Mbps)	Μετρήσεις bandwidth μέσω εργαλείου nload (Mbps)	Μετρήσεις bandwidth με την προτεινόμενη λύση (Mbps)
Τοπολογία δυαδικού δέντρου ύψους 5	20.9	19.25	20.34
Τοπολογία δυαδικού δέντρου ύψους 6	18.7	18.41	19.38
Τοπολογία δυαδικού δέντρου ύψους 7	18.55	18.24	19.28
Τοπολογία δυαδικού δέντρου ύψους 8	18.36	18.18	19.25
Γραμμική τοπολογία με 16 μεταγωγείς	19.5	18.64	19.96
Γραμμική τοπολογία με 32 μεταγωγείς	18.7	18.54	19.62
Γραμμική τοπολογία με 64 μεταγωγείς	18.55	18.40	19.48
Γραμμική τοπολογία με 128 μεταγωγείς	18.34	18.22	19.41

Πίνακας 3: Πίνακας σύγκρισης τιμών bandwidth μεταξύ εργαλείου iperf, nload και προτεινόμενης λύσης



Εικόνα 25: Σύγκριση μετρήσεων για Bandwidth μεταξύ Iperf, nload και προτεινόμενης λύσης

4.4.2.3 Σχόλια αποτελεσμάτων της ενότητας 4.4.2.2

Με τις μετρήσεις του πίνακα 3 συγκρίναμε τα αποτελέσματα της προτεινόμενής μας λύσης με τα αποτελέσματα της εντολής iperf και nload για γραμμικές τοπολογίες και για τοπολογίες δέντρα. Ρυθμίσαμε τις ακμές των τοπολογιών έτσι ώστε να έχουν σταθερό bandwidth στα 20 Mbps σε κάθε περίπτωση και στην εικόνα 25 παρουσιάσαμε τις μετρήσεις μας σε γράφημα για να τις συγκρίνουμε με την πραγματική τιμή του bandwidth. Από την γραφική παράσταση παρατηρούμε πως η προτεινόμενη λύση διατηρεί τις τιμές τις πιο κοντά στην πραγματική τιμή του bandwidth σε σχέση με τα αποτελέσματα της εντολής iperf και του nload σε κάθε περίπτωση τοπολογίας που εξετάσαμε. Συγκεκριμένα τα αποτελέσματα και των τριών εργαλείων φαίνεται να έχουν μεγαλύτερη απόκλισή όσο μεγαλύτερες είναι οι τοπολογίες πράγμα αναμενόμενο λόγο της μεγαλύτερης διαδρομής που ταξιδεύουν τα πακέτα οπότε υπάρχει μεγαλύτερο περιθώριο σφαλμάτων. Παρατηρώντας τις μεγαλύτερες σε μέγεθος τοπολογίες φαίνεται πως στην περίπτωση του εργαλείου Iperf τα αποτελέσματα του Bandwidth συγκλίνουν γύρω στα 18.35 Mbps με σφάλμα δηλαδή μέχρι και 8.25%. Ομοίως το εργαλείο nload φαίνεται να φτάνει μέχρι την τιμή 18.20 Mbps σε απόκλιση δηλαδή 9% από την θεωρητική τιμή των 20 Mbps. Τέλος στην περίπτωση της προτεινόμενης λύσης φαίνεται ότι τα

αποτελέσματα δεν ξεπερνούν την απόκλιση 4.25% με την περίπτωση των 256 κόμβων της τοπολογίας δέντρου να φτάνει την τιμή 19.25Mbps. Προφανώς η διαφορά στην ακρίβεια των μετρήσεων της προτεινόμενης λύσης με τα εργαλεία nload και Iperf οφείλεται στον μεγάλο όγκο πακέτων που προσθέτουν αυτά στο δίκτυο, ειδικά στην περίπτωση του Iperf που εκτελούνται διάφορες δοκιμές TCP για να υπολογιστεί το Bandwidth της ζεύξης. Αντίθετα η προτεινόμενη λύση προσθέτει ελαφριά επιπρόσθετη πληροφορία στην ζεύξη του δικτύου.

4.4.3 Σύγκριση Packet Loss

4.4.3.1 Ρυθμίσεις τοπολογιών

Με παρόμοιο τρόπο θα συγκρίνουμε τώρα το αναμενόμενο packet loss που προκύπτει από κάθε μορφή τοπολογίας με τα αποτελέσματα που παράγει ο διαχειριστής που φτιάξαμε. Προκειμένου να γίνει η σύγκριση φτιάχνουμε και πάλι ελεγχόμενες τοπολογίες όπως και στην πηγή [33] χρησιμοποιώντας το Python API του Mininet. Ρυθμίσαμε λοιπόν τις τοπολογίες μας ώστε μόνο οι ακμές που συνδέουν τους μεταγωγείς μεταξύ τους να έχουν **packet loss rate = 2%**. Όλες οι ακμές που συνδέουν κόμβο με μεταγωγέα είναι ρυθμισμένες ώστε να έχουν **μηδενικό packet loss rate**, ενώ όλες οι ακμές έχουν ρυθμισμένο **bandwidth = 20Mbps/Sec**. Επιλέξαμε τις ρυθμίσεις αυτές εφόσον σύμφωνα με τα δεδομένα που έχουμε στην διάθεσή μας όπως φαίνεται από την ενότητα **4.3.1**, γνωρίζουμε τον αριθμό των πακέτων που μεταφέρονται ανά πάσα στιγμή μόνο διαμέσων των μεταγωγέων. Έτσι οι μετρήσεις μας θα είναι ακόμα πιο κοντά στα αναμενόμενα ποσοστά του packet loss. Στους πίνακες 4 και 5 που ακολουθούν συγκεντρώσαμε τις μετρήσεις του Packet Loss rate σε διάφορες γραμμικές τοπολογίες και τοπολογίες δέντρα. Συγκεκριμένα θα συγκρίνουμε την αναμενόμενη τιμή του Packet Loss με το αποτέλεσμα της προτεινόμενης λύσης και με τα εργαλεία ping και iperf. Σε κάθε περίπτωση καταγράψαμε τον μέσο όρο 10 μετρήσεων Packet Loss για κάθε περίπτωση τοπολογίας για τα 3 εργαλεία.

4.4.3.2 Συγκρίσεις Packet Loss με εργαλείο Iperf

Όπως προαναφέρθηκε και στο κεφάλαιο 2.4, εργαλείο iperf λειτουργεί με υπηρεσία πελάτη-εξυπηρετητή. Εντούτοις οι μετρήσεις που εξάγει θα είναι για την διαδρομή αποστολέα - παραλήπτη. Έτσι ανάλογα με την μορφή τοπολογίας σε κάθε περίπτωση προκύπτει η πιο κάτω εξίσωση υπολογισμού του αναμενόμενου Packet Loss:

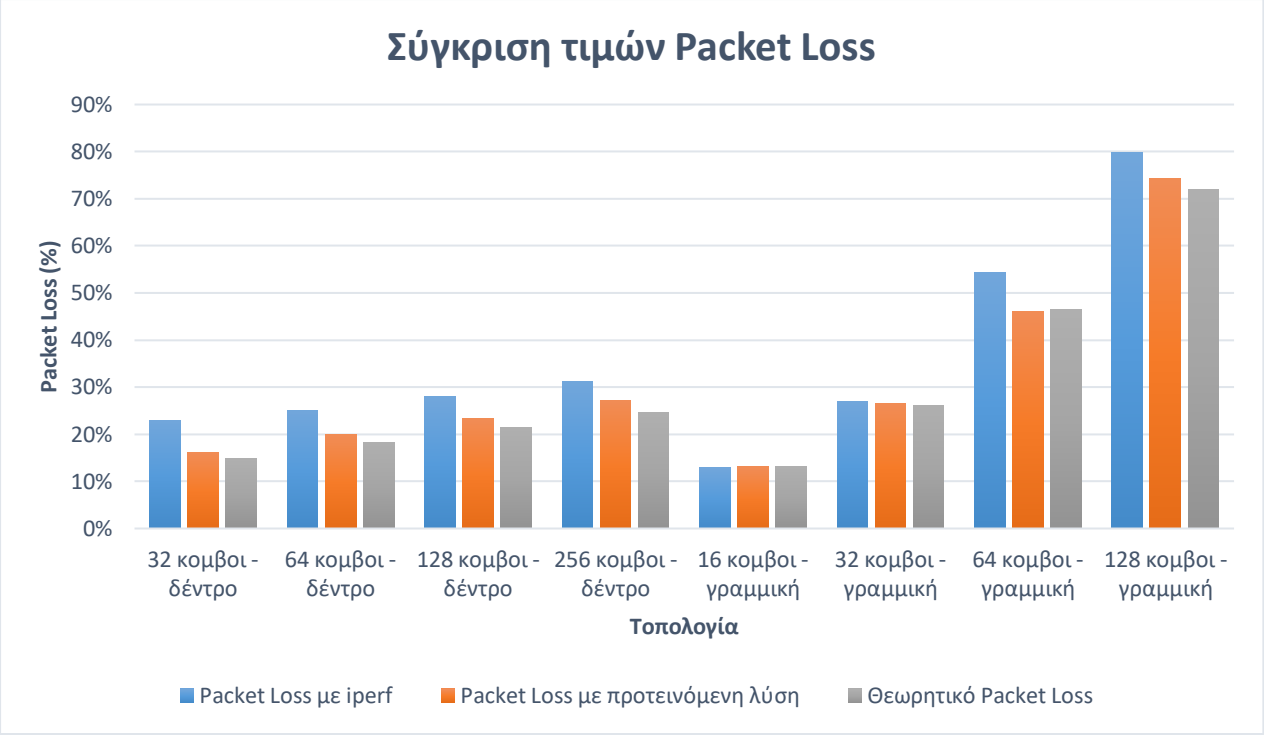
$$Expected\ loss\ rate\ [\mathbf{33}] = 1 - (1 - link\ loss\ rate)^{links}$$

Όπου *link loss rate* είναι το ρυθμισμένο Packet Loss που έχουν οι ακμές και *links* είναι το πλήθος των ακμών με το loss rate αυτό. Εφόσον ο όρος $(1 - link\ loss\ rate)^{links}$ εκφράζει την πιθανότητα κανένα πακέτο να μην έχει χαθεί μέσα σε μια ζεύξη, προκύπτει λοιπόν η πιο πάνω σχέση που εκφράζει το

συμπληρωματικό αυτού το ενδεχομένου δηλαδή την πιθανότητα να απορριφθούν ένα η περισσότερα πακέτα. Βάσει των ρυθμίσεών που ορίσαμε στις ακμές, για τις γραμμικές τοπολογίες ο όρος *links* παίρνει τιμή ίση με $\textit{πλήθος μεταγωγέων} - 1$ ενώ για τοπολογίες δέντρα $(\textit{ύψος δέντρου} - 1) \times 2$. Για να συγκρίνουμε το Packet Loss ενεργοποιήσαμε το πρωτόκολλο μεταφοράς UDP με το εργαλείο Iperf εφόσον μόνο με τη χρήση UDP το Iperf υπολογίζει το μέγεθος αυτό.

Μορφή τοπολογίας (2 παιδιά ανά μεταγωγέα)	Θεωρητική τιμή αναμενόμενου Loss Rate ζεύξης	Μέτρηση Loss Rate μέσω εντολής iperf	Μετρήσεις Loss Rate με την προτεινόμενη λύση
Τοπολογία δυαδικού δέντρου ύψους 5	14.92%	23%	16.2%
Τοπολογία δυαδικού δέντρου ύψους 6	18.29%	25%	20%
Τοπολογία δυαδικού δέντρου ύψους 7	21.5%	28%	23.4%
Τοπολογία δυαδικού δέντρου ύψους 8	24.6%	31.2%	27.2%
Γραμμική τοπολογία με 8 μεταγωγείς	13.2%	13%	13.1%
Γραμμική τοπολογία με 16 μεταγωγείς	26.14%	27%	26.6%
Γραμμική τοπολογία με 32 μεταγωγείς	46.54%	54.3%	46%
Γραμμική τοπολογία με 64 μεταγωγείς	72%	79.8%	74.4%

Πίνακας 4: Πίνακας σύγκρισης τιμών loss rate μεταξύ εντολής iperf και προτεινόμενης λύσης



Εικόνα 26: Σύγκριση μετρήσεων για Packet Loss μεταξύ εντολής iperf και προτεινόμενης λύσης

4.4.3.3 Συγκρίσεις Packet Loss με εργαλείο Ping

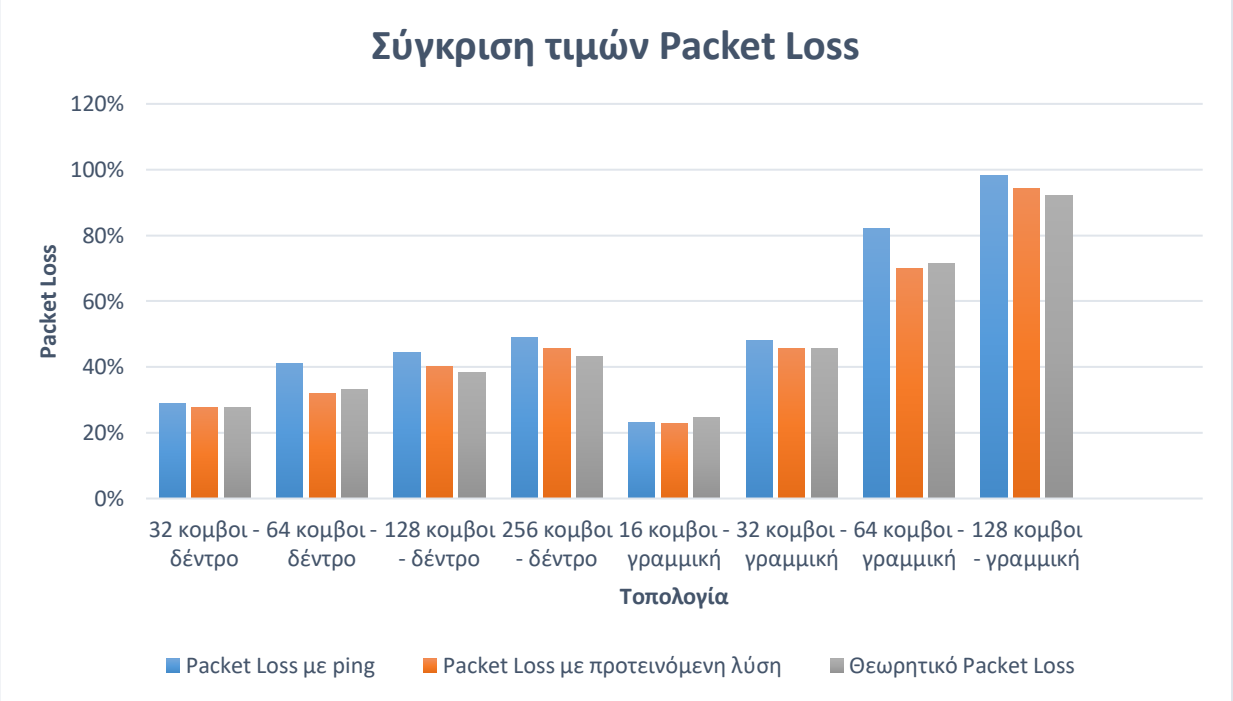
Θα συγκρίνουμε τώρα τα αποτελέσματα της προτεινόμενης λύσης και με το εργαλείο Ping. Αντίθετα με την περίπτωση του εργαλείου Iperf, όπως αναφέρει και η πηγή **[33]** το Ping αναμένεται να έχει διαφορετικά αποτελέσματα εφόσον όπως προαναφέραμε και στο κεφάλαιο 2.4 το Ping χρησιμοποιείται μόνο από το ένα άκρο της διαδρομής και εξάγει αποτελέσματα που αφορούν την Round - Trip διαδρομή δηλαδή αποστολέα – παραλήπτη – αποστολέα. Έτσι στην εξίσωση που χρησιμοποιήσαμε προηγουμένως:

$$Expected\ loss\ rate\ [\mathbf{33}] = 1 - (1 - link\ loss\ rate)^{links}$$

η παράμετρος links θα πάρει την διπλάσια τιμή για κάθε τοπολογία που ελέγξαμε στην ενότητα 4.4.3.2 αφού τα πακέτα πρόκειται να περάσουν δύο φορές από τις ίδιες ακμές της τοπολογίας. Επομένως ο όρος *links* παίρνει τιμή ίση με $(\pi\lambda\acute{\eta}\theta o\varsigma\ \mu\epsilon\tau\alpha\gamma\omega\gamma\acute{\epsilon}\omega\nu\ -1)\ \times 2$ για γραμμικές τοπολογίες ενώ για τοπολογίες δέντρα $(\acute{\upsilon}\psi o\varsigma\ \delta\acute{\epsilon}\nu\tau\rho o\upsilon - 1)\ \times 4$.

Μορφή τοπολογίας (2 παιδιά ανά μεταγωγέα)	Θεωρητική τιμή αναμενόμενου Loss Rate ζεύξης	Μετρήσεις Loss Rate μέσω εντολής ping	Μετρήσεις Loss Rate με την προτεινόμενη λύση
Τοπολογία δυαδικού δέντρου ύψους 5	27.6%	29%	27.6%
Τοπολογία δυαδικού δέντρου ύψους 6	33.2%	41.2%	32%
Τοπολογία δυαδικού δέντρου ύψους 7	38.4%	44.5%	40.2%
Τοπολογία δυαδικού δέντρου ύψους 8	43.2%	49%	45.7%
Γραμμική τοπολογία με 8 μεταγωγείς	24.6%	23%	22.9%
Γραμμική τοπολογία με 16 μεταγωγείς	45.5%	48%	45.7%
Γραμμική τοπολογία με 32 μεταγωγείς	71.4%	82.2%	69.8%
Γραμμική τοπολογία με 64 μεταγωγείς	92.16%	98.2%	94.4%

Πίνακας 5: Πίνακας σύγκρισης τιμών loss rate μεταξύ εντολής ping και προτεινόμενης λύσης



Εικόνα 27: Σύγκριση μετρήσεων για Packet Loss μεταξύ εντολής ping και προτεινόμενης λύσης

4.4.3.4 Σχόλια αποτελεσμάτων της ενότητας 4.4.3.2 και 4.4.3.3

Στους πίνακες 4 και 5 παρουσιάσαμε τις μετρήσεις του μεγέθους Packet Loss που πήραμε με διάφορες τοπολογίες δέντρα και γραμμικές για τα εργαλεία Iperf, Ping και την προτεινόμενη μας λύση και έπειτα συγκρίνουμε την ακρίβεια των τριών στις γραφικές παραστάσεις των εικόνων 26 και 27.

Ξεκινώντας από την σύγκριση Iperf με την προτεινόμενη λύση παρατηρούμε πως σε κάθε περίπτωση τοπολογίας η προτεινόμενη λύση και πάλι διατηρούσε μικρότερη διαφορά από το αναμενόμενο Packet Loss σε σχέση το εργαλείο Iperf. Φαίνεται επίσης πως ενώ σε κάποιες περιπτώσεις τα 2 εργαλεία παρουσίασαν μεγάλη ακρίβεια στις μετρήσεις τους, εντούτοις σε περιπτώσεις όπως την τοπολογία δέντρου ύψους 5 το εργαλείο Ping σημείωσε μεγάλου μεγέθους απόκλιση από την αναμενόμενη τιμή (53.7%) ενώ παράλληλα η προτεινόμενη λύση διατήρησε μικρή απόκλιση (8.6%) στην ίδια περίπτωση. Παράλληλα παρατηρούμε πως στις μεγαλύτερες τοπολογίες υπάρχει αυξημένη ανακρίβεια από τη μεριά του Iperf ενώ η προτεινόμενη λύση διατηρεί μια σχετικά ακριβής τιμή χωρίς να ξεπεράσει την απόκλιση 8.6% που προαναφέρθηκε

Στην περίπτωση του Ping, τα δύο εργαλεία φαίνεται να διατηρούν επιτυχώς την ακρίβεια τους σε κάθε περίπτωση τοπολογίας. Από το διάγραμμα της εικόνας 27 παρατηρούμε πως σε καμία περίπτωση δεν υπήρξε μεγάλη απόκλιση από κανένα εργαλείο ενώ παράλληλα σε κάθε μορφή τοπολογίας η προτεινόμενη λύση είχε μεγαλύτερη ακρίβεια από την αντίστοιχη μέτρηση του εργαλείου Ping με μοναδική εξαίρεση την περίπτωση της γραμμικής τοπολογίας 8 μεταγωγέων όπου η προτεινόμενη λύση παρουσίασε απόκλιση 6.9% ενώ το εργαλείο Ping 6.5%. Εν τέλει η περίπτωση αυτή αποτελούσε την μεγαλύτερη απόκλιση για την προτεινόμενη λύση ενώ για το Ping είχαμε απόκλιση μέχρι και 15% (στην περίπτωση γραμμικής τοπολογίας 32 μεταγωγέων). Αξίζει επίσης να σημειωθεί ότι και για το Ping παρουσιάζεται μεγαλύτερη ανακρίβεια στις μεγαλύτερες τοπολογίες συγκριτικά με την προτεινόμενη λύση.

Στη περίπτωση του εργαλείου Ping η διαφορά ης ακρίβειας των μετρήσεων οφείλεται στην συνεχή αποστολή των πακέτων τύπου ICMP στο δίκτυο επηρεάζοντας την ακρίβεια των μετρήσεων του εργαλείου. Το εργαλείο Iperf όπως προαναφέρθηκε στην ενότητα 4.4.3.4 βασίζεται στην αποστολή μεγάλου όγκου πληροφορίας για τις μετρήσεις του πράγμα που επίσης δικαιολογεί την μεγαλύτερη ανακρίβεια σε σχέση με την προτεινόμενη λύση.

4.4.4 Σύγκριση delay

4.4.4.1 Ρυθμίσεις τοπολογιών

Σε αυτή την υποενότητα θα συγκρίνουμε τις μετρήσεις του Delay που παράγει η προτεινόμενη λύση για διάφορες μορφές τοπολογιών με το αναμενόμενο Delay κάθε τοπολογίας προκειμένου να παρατηρήσουμε πόσο η μορφή της τοπολογίας επηρεάζει την ακρίβεια των μετρήσεων για το Delay. Προκειμένου να γίνει η σύγκριση φτιάχνουμε και πάλι ελεγχόμενες τοπολογίες χρησιμοποιώντας το Python API του Mininet σύμφωνα με την μέθοδο που ακολουθεί και η πηγή [33]. Ρυθμίσαμε λοιπόν τις τοπολογίες μας ώστε μόνο οι ακμές που συνδέουν τους μεταγωγείς μεταξύ τους να έχουν καθυστέρηση Delay = 50ms ενώ όλες οι ακμές της τοπολογίας είναι ρυθμισμένες ώστε να έχουν bandwidth = 20 Mbps/Sec. Όπως αναφέρεται και στην πηγή [33] με αυτό το τρόπο οι ενδείξεις των εργαλείων με τα οποία θα συγκρίνουμε τα αποτελέσματα της προτεινόμενης λύσης θα αναφέρονται αντιστοιχούν μόνο στο delay των ακμών ενδιάμεσα των μεταγωγέων. Έτσι θα έχει νόημα η σύγκρισή τους με την προτεινόμενη λύση εφόσον με αυτή μπορούμε να γνωρίζουμε μόνο τα δεδομένα των ακμών ενδιάμεσα των μεταγωγέων.

Πρόκειται να συγκρίνουμε τις μετρήσεις της προτεινόμενης μας λύσης με τα εργαλεία Ping Traceroute. Και τα δύο εργαλεία υπολογίζουν μετά την λήξη τους το Delay της Round – Trip διαδρομής σε αντίθεση με την προτεινόμενη λύση που υπολογίζει το Delay για την διαδρομή αποστολέα - παραλήπτη μόνο, οπότε αφού οι διαδρομές από και προς τον παραλήπτη είναι συμμετρικές όσον αφορά την ποιότητα της ζεύξης, διαιρούμε τις μετρήσεις που πήραμε για τα δύο εργαλεία διά 2 και έχουμε μία σχετικά ακριβής προσέγγιση της διαδρομής αποστολέα – παραλήπτη για τα δύο εργαλεία. Επομένως παρουσιάζουμε στους πίνακες 6 και 7 τις μετρήσεις για την διαδρομή αποστολέα – παραλήπτη σε όλες τις περιπτώσεις.

Για τη σύγκριση του Delay πρόκειται ακόμα να συνδυάσουμε την μέθοδο των ελεγχόμενων τοπολογιών με την μέθοδο που ακολουθεί και η πηγή [36]. Πρόκειται δηλαδή να τρέξουμε ελεγχόμενες τοπολογίες με διαφορετικές τιμές ρυθμισμένων Delay στις ακμές ενδιάμεσα των μεταγωγέων. Στην περίπτωσή μας θα κατασκευάσουμε τοπολογίες με Delay = 0ms και Delay = 50ms στις ακμές και θα υπολογίσουμε το συνολικό Delay για τα τρία εργαλεία και στις δύο περιπτώσεις. Σύμφωνα με την πηγή [36] η διαφορά στις δύο μετρήσεις που θα πάρουμε θα είναι γνωστό μέγεθος λόγω των γνωστών ρυθμίσεων σε κάθε περίπτωση και γι' αυτό θα χρησιμοποιήσουμε την διαφορά τους σαν αναμενόμενο Delay για κάθε τοπολογία. Επομένως στους πίνακες 6 και 7 που ακολουθούν καταγράφουμε την διαφορά του συνολικού Delay που υπολογίστηκε στην περίπτωση Delay = 50ms για κάθε ακμή και Delay = 0ms για κάθε ακμή και την συγκρίνουμε με την τιμή του αναμενόμενου Delay κάθε φορά. Η τιμή του αναμενόμενου Delay στην περίπτωσή μας προκύπτει από την σχέση:

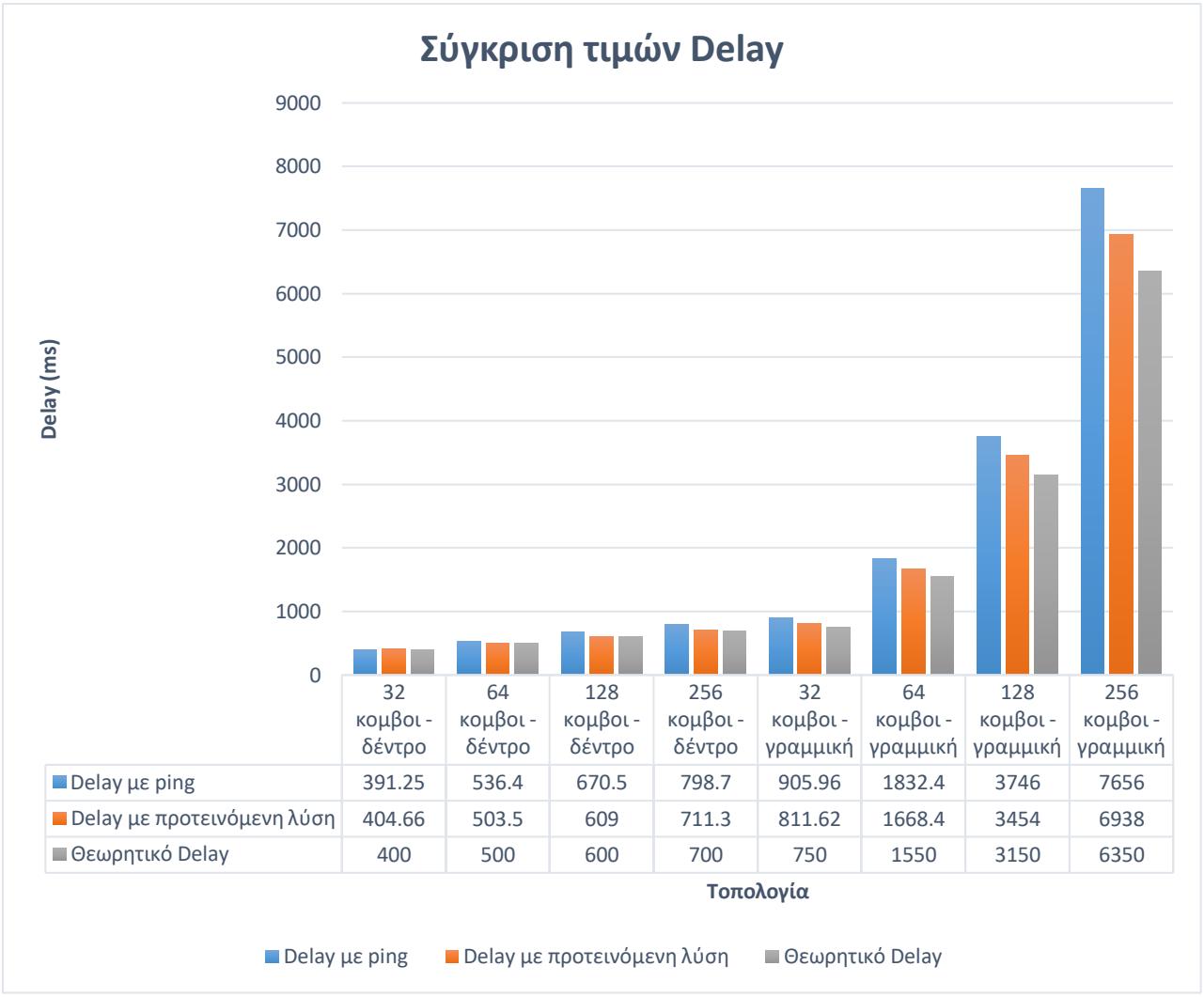
$$Expected\ delay = links\ \times 50$$

Ο όρος *links* είναι ο αριθμός των ακμών που συνδέουν τον πρώτο και τελευταίο μεταγωγέα της διαδρομής αφού η συνολική καθυστέρηση ισούται με το άθροισμα των καθυστερήσεων των ακμών. Στους πίνακες 6 και 7 καταγράψαμε τον μέσο όρο 10 μετρήσεων του Delay για κάθε περίπτωση τοπολογίας.

4.4.4.2 Συγκρίσεις Delay με εργαλείο Ping

Μορφή τοπολογίας (2 παιδιά ανά μεταγωγέα)	Τιμή αναμενόμενου Delay (ms)	Μετρήσεις Delay με ping (ms)	Μετρήσεις Delay με προτεινόμενη (ms)
Τοπολογία δυαδικού δέντρου ύψους 5	400	391.25	404.66
Τοπολογία δυαδικού δέντρου ύψους 6	500	536.4	503.5
Τοπολογία δυαδικού δέντρου ύψους 7	600	670.5	609
Τοπολογία δυαδικού δέντρου ύψους 8	700	798.7	711.3
Γραμμική τοπολογία με 16 μεταγωγείς	750	905.96	811.62
Γραμμική τοπολογία με 32 μεταγωγείς	1550	1832.4	1668.4
Γραμμική τοπολογία με 64 μεταγωγείς	3150	3746	3454
Γραμμική τοπολογία με 128 μεταγωγείς	6350	7656	6938

Πίνακας 6: Πίνακας σύγκρισης τιμών Delay μεταξύ εντολής ping και προτεινόμενης λύσης

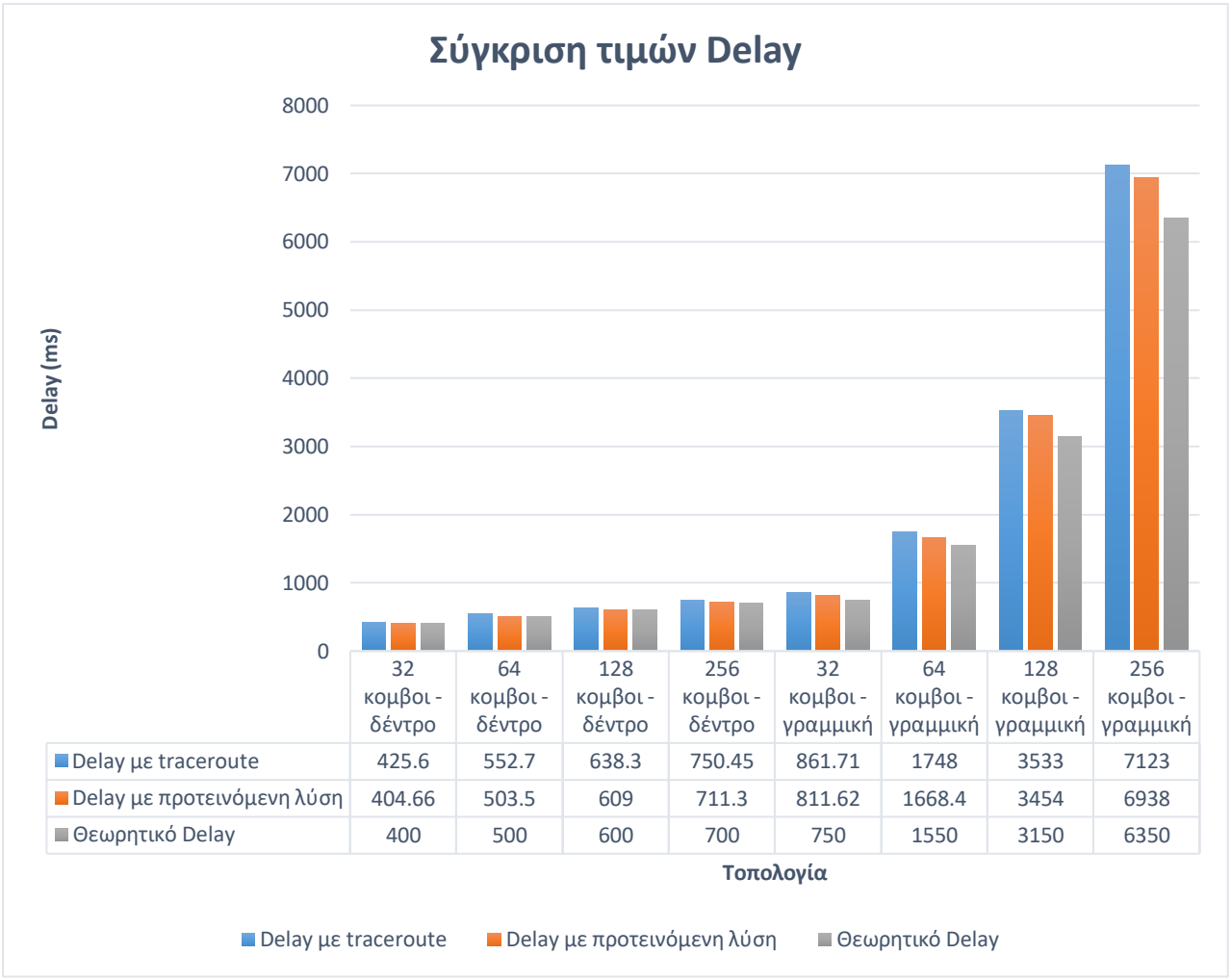


Εικόνα 28: Σύγκριση μετρήσεων για Delay μεταξύ εντολής ping και προτεινόμενης λύσης

4.4.4.3 Συγκρίσεις Delay με εργαλείο Traceroute

Μορφή τοπολογίας (2 παιδιά ανά μεταγωγέα)	Τιμή αναμενόμενου Delay (ms)	Μετρήσεις Delay με Traceroute (ms)	Μετρήσεις Delay με προτεινόμενη (ms)
Τοπολογία δυαδικού δέντρου ύψους 5	400	425.6	404.66
Τοπολογία δυαδικού δέντρου ύψους 6	500	552.7	503.5
Τοπολογία δυαδικού δέντρου ύψους 7	600	638.3	609
Τοπολογία δυαδικού δέντρου ύψους 8	700	750.45	711.3
Γραμμική τοπολογία με 16 μεταγωγείς	750	861.71	811.62
Γραμμική τοπολογία με 32 μεταγωγείς	1550	1748	1668.4
Γραμμική τοπολογία με 64 μεταγωγείς	3150	3533	3454
Γραμμική τοπολογία με 128 μεταγωγείς	6350	7123	6938

Πίνακας 7: Πίνακας σύγκρισης τιμών Delay μεταξύ εντολής traceroute και προτεινόμενης λύσης



	32 κομβοι - δέντρο	64 κομβοι - δέντρο	128 κομβοι - δέντρο	256 κομβοι - δέντρο	32 κομβοι - γραμμική	64 κομβοι - γραμμική	128 κομβοι - γραμμική	256 κομβοι - γραμμική
Delay με traceroute	425.6	552.7	638.3	750.45	861.71	1748	3533	7123
Delay με προτεινόμενη λύση	404.66	503.5	609	711.3	811.62	1668.4	3454	6938
Θεωρητικό Delay	400	500	600	700	750	1550	3150	6350

Εικόνα 29: Σύγκριση μετρήσεων για Delay μεταξύ εντολής traceroute και προτεινόμενης λύσης

4.4.4.4 Σχόλια αποτελεσμάτων της ενότητας 4.4.4.2 και 4.4.4.3

Στους πίνακες 6 και 7 παρουσιάσαμε τις μετρήσεις του μεγέθους Delay που πήραμε με διάφορες τοπολογίες δέντρα και γραμμικές για τα εργαλεία Ping, Traceroute και την προτεινόμενη μας λύση και έπειτα συγκρίνουμε την ακρίβεια των τριών στις γραφικές παραστάσεις των εικόνων 28 και 29.

Παρατηρώντας τις μετρήσεις και των τριών εργαλείων φαίνεται πως είχαμε μεγαλύτερη ακρίβεια στα αποτελέσματα που σημειώθηκαν για τις τοπολογίες των δέντρων από ότι στις γραμμικές τοπολογίες. Συγκεκριμένα όσον αφορά την σύγκριση με το Ping, η προτεινόμενη λύση είχε σε κάθε περίπτωση μεγαλύτερη ακρίβεια από το εργαλείο Ping και στις δύο μορφές τοπολογιών. Για τις τοπολογίες δέντρα η προτεινόμενη λύση παρουσίασε απόκλιση μέχρι μόλις 1.34% ενώ για τις γραμμικές τοπολογίες παρουσίασε απόκλιση μέχρι 12.4%. Αντίθετα το εργαλείο Ping παρουσίασε απόκλιση μέχρι 14.1% στις τοπολογίες δέντρα και απόκλιση 20.6% στις γραμμικές τοπολογίες. Εδώ παρατηρούμε μεγάλη απόκλιση και από τις δύο μεριές όσο μεγαλώνει το μήκος της διαδρομής εφόσον όσο μεγαλύτερη τοπολογία είχαμε τόσο μεγαλύτερη ήταν και η απόκλιση των εργαλείων στην περίπτωση των γραμμικών τοπολογιών.

Κάτι αντίστοιχο βλέπουμε να συμβαίνει και στην σύγκριση της προτεινόμενης μας λύσης με το εργαλείο Traceroute. Ενώ παρατηρούμε από την προτεινόμενη λύση απόκλιση μέχρι 1.16% και από το Traceroute μέχρι 7.2% για τις τοπολογίες δέντρα, στις γραμμικές τοπολογίες το ποσοστό απόκλισης φαίνεται να φτάνει μέχρι 9.26% για την προτεινόμενη λύση και 14.9% για το εργαλείο Traceroute.

Η διαφορά των μετρήσεων και πάλι οφείλεται στον μεγάλο όγκο πληροφορίας με την οποία επιβαρύνουν το δίκτυο τα εργαλεία Traceroute και Ping, σε αντίθεση με την μικρή πληροφορία που προσθέτει η προτεινόμενη λύση. Εν τέλει φαίνεται ότι η προτεινόμενη λύση που κάνει χρήση της τεχνολογίας SDN, εμφανίζει μεγαλύτερη αξιοπιστία στα αποτελέσματα και είναι πιο κοντά στην θεωρητική τιμή των μετρήσεων. Αντίθετα, φαίνεται ότι με τα συμβατικά εργαλεία traceroute και ping, υπάρχει μεγαλύτερη απόκλιση από τις θεωρητικές τιμές και άρα μικρότερη αξιοπιστία μετρήσεων, η οποία απόκλιση γίνεται πιο έντονη για μεγαλύτερες τοπολογίες. Συνεπώς επιβεβαιώνουμε τα θεωρητικά πλεονεκτήματα της παρατήρησης δικτύων μέσω SDN λύσης, με τα πειραματικά αποτελέσματα που παρήχθησαν, συμπεραίνοντας ότι για μεγάλες τοπολογίες δικτύων (όπως Data Centers), τα συμβατικά εργαλεία μετρήσεων δεν είναι αρκετά αξιόπιστα και χρειάζονται νέα εργαλεία μετρήσεων, όπως η προτεινόμενη λύση

Κεφάλαιο 5: Συμπεράσματα

5.1 Σύνοψη

Στην αρχή της παρούσας διπλωματικής αναλύσαμε πόσο σημαντική είναι η παρακολούθηση δικτύων στη σύγχρονη εποχή και αναφερθήκαμε στο πρόβλημα που παρουσιάζει η παραδοσιακή αρχιτεκτονική δικτύων να ανταπεξέλθει στην ανάπτυξη της τεχνολογίας και του διαδικτύου. Συγκεκριμένα τονίσαμε τον μεγάλο ρόλο που έχει η παρακολούθηση δικτύων στην σύγχρονη εποχή με τα δεδομένα που μπορούν να στείλουν πλέον οι χρήστες να εξελίσσονται σε τέτοιο βαθμό ώστε οι διαχειριστές δικτύων να χρειάζονται ανά πάσα στιγμή μια ολοκληρωμένη εικόνα του δικτύου προκειμένου να μπορούν να προσφέρουν τις νέες αυτές υπηρεσίες.

Στη συνέχεια διερευνήσαμε μερικές από τις πιο βασικές τεχνικές που χρησιμοποιούν τα δίκτυα για παρακολούθηση και συλλογή δεδομένων σε μία τοπολογία δικτύου. Πρώτιστα αναλύσαμε ότι οι μέθοδοι συλλογής δεδομένων σε ένα δίκτυο χωρίζονται κυρίως σε παθητικές και ενεργητικές. Όπως είχαμε αναφέρει οι παθητικές μέθοδοι βασίζονται στην εγκατάσταση ξένων εργαλείων στις ακμές ενός δικτύου που παρακολουθούν την κυκλοφορία σε αυτό με παθητικό τρόπο χωρίς δηλαδή να επιφέρουν επιπλέον όγκο κυκλοφορίας αλλά κοστίζουν περισσότερο. Από την άλλη οι ενεργητικές μέθοδοι βασίζουν τους υπολογισμούς τους σε πακέτα που στέλνουν στο δίκτυο προκειμένου να μετρήσουν βασικά μεγέθη. Έπειτα αναφέραμε κάποιες γνωστές μεθόδους συλλογής δεδομένων που χρησιμοποιούνται όπως η αντιγραφή κυκλοφορίας και η παρακολούθηση ροών με την αντιγραφή κυκλοφορίας να χρησιμοποιεί διάφορα εργαλεία ώστε να διακλαδώνει μία ακμή και να δημιουργεί αντίγραφα των πακέτων που την διαπερνούν προκειμένου να τα αναλύσει και την παρακολούθηση ροών να χωρίζει τα πακέτα σε ομάδες (ροές) αναλύοντας μόνο συγκεκριμένα στοιχεία για αυτά όπως στοιχεία για τον προορισμό και την πηγή τους. Τέλος αναφερθήκαμε εν συντομία σε μερικά ήδη υπάρχοντα γνωστά εργαλεία που χρησιμοποιούνται για παρακολούθηση δικτύων όπως Simple Network Management Protocol (SNMP), NetFlow, cSamp και το Skitter.

Αφού αναλύσαμε τις μεθόδους που χρησιμοποιούνται ήδη για την παρακολούθηση δικτύων εισαγάγαμε την έννοια του Software Defined Networking δίκτυα δηλαδή οριζόμενα από λογισμικό. Πρώτα αναφέραμε σύντομα τι προσφέρει η καινοτόμα αυτή ιδέα, δηλαδή το γεγονός ότι χωρίζει το επίπεδο ελέγχου με το επίπεδο δεδομένων σε ένα δίκτυο χρησιμοποιώντας έναν κεντρικό διαχειριστή δικτύου, ο οποίος με την χρήση λογισμικού είναι υπεύθυνος για τους κανόνες προώθησης των δεδομένων στο δίκτυο. Όπως επισημάνθηκε αυτό αποτελεί μεγάλο προτέρημα για τον διαχειριστή του δικτύου αφού καθιστά απλούστερη την διαχείριση του, προσφέροντας μεγαλύτερη ευελιξία και δυνατότητα ελέγχου του δικτύου.

Μετά την σύντομη αναφορά στην αρχιτεκτονική SDN, αναλύουμε το πρωτόκολλο Openflow που χρησιμοποιείται σε συνδυασμό με την αρχιτεκτονική αυτή. Το πρωτόκολλο Openflow αποτελεί την διεπαφή που φροντίζει την σωστή αλληλεπίδραση μεταξύ του επιπέδου ελέγχου και δεδομένων. Συγκεκριμένα το Openflow εγκαθιστά στους μεταγωγείς του δικτύου πίνακες ροών που χρησιμοποιεί

ο Openflow διαχειριστής προκειμένου να εγκαταστήσει, ενημερώσει ή να διαγράψει κανόνες σε αυτούς για την βέλτιστη επικοινωνία μεταξύ των κόμβων του δικτύου.

Με την ανάπτυξη της λειτουργίας του Openflow πρωτοκόλλου αναφέραμε μερικούς δημοφιλείς τρόπους που χρησιμοποιούνται για την παρακολούθηση στα δίκτυα SDN μιας και τα περισσότερα χρησιμοποιούν τις ιδιότητες του πρωτοκόλλου αυτού. Αναφερθήκαμε με λεπτομέρεια στα εργαλεία OpenTM, FlowSense, Payless και OpenNetMon, τα οποία με αποκτούν δεδομένα για το δίκτυο είτε στέλνοντας πακέτα τύπου *FlowStatsRequest* στους μεταγωγείς σε συγκεκριμένα χρονικά διαστήματα που καθορίζουνε με ξεχωριστό αλγόριθμο, είτε περιμένουν την δημιουργία και την λήξη των ροών του δικτύου ώστε να παραχθούν πακέτα τύπου *PacketIn* και *FlowRemoved* αντίστοιχα και με τα δεδομένα των συγκεκριμένων πακέτων να υπολογίσουν τα απαραίτητα μεγέθη. Λαμβάνοντας υπόψη τον τρόπο λειτουργίας των προαναφερθέντων μεθόδων εισαγάγαμε στην συνέχεια την δική μας προτεινόμενη λύση που συνδυάζει τις πιο πάνω μεθόδους συλλογής δεδομένων αλλά φροντίζοντας παράλληλα να επιφέρει όσο το δυνατό λιγότερο περιττό όγκο κυκλοφορίας δεδομένων στο δίκτυο.

Προκειμένου να δοκιμαστεί η προτεινόμενή μας λύση εκτελέσαμε στην επόμενη ενότητα πολλαπλά πειράματα χρησιμοποιώντας τον προσομοιωτή δικτύων mininet. Έπειτα από μια μικρή εισαγωγή για την ανάπτυξη και λειτουργία ενός Openflow διαχειριστή χρησιμοποιώντας την πλατφόρμα ανοιχτού κώδικα POX φτιάξαμε διάφορες ελεγχόμενες τοπολογίες γραμμικής μορφής και μορφής δέντρου και χρησιμοποιώντας την προτεινόμενή μας λύση και τα εργαλεία Iperf, Ping, και Traceroute πήραμε και συγκρίναμε μετρήσεις για bandwidth packet loss και delay για τις τοπολογίες αυτές. Συγκεκριμένα ελέγξαμε την συμπεριφορά των αποτελεσμάτων μας σε τοπολογίες διαφορετικών μεγεθών, και σε ακμές με διαφορετικές ρυθμίσεις για να εξασφαλίσουμε την ακρίβεια των μετρήσεων μας.

Σκοπός της παρούσας διπλωματικής ήταν να τονίσει το πρόβλημα που υπάρχει όσον αφορά την παρακολούθηση δικτύων στην σημερινή εποχή και χρησιμοποιώντας την τεχνολογία SDN να προσπαθήσουμε να το αντιμετωπίσουμε . Η συνεισφορά μας σε αυτό το θέμα είναι λοιπόν η παρουσίαση της δικής μας λύσης για το παρόν πρόβλημα χρησιμοποιώντας τις μοναδικές ιδιότητες που προσφέρει το πρωτόκολλο Openflow ώστε οι διαχειριστές δικτύων να έχουν έναν εύκολο έμπιστο και οικονομικό τρόπο παρακολούθησης των δικτύων τους.

5.2 Συγκρίσεις - Παρατηρήσεις

Με τα πειράματα που εκτελέσαμε και τις μετρήσεις που πήραμε καταλήγουμε στο συμπέρασμα ότι η ακρίβεια των μετρήσεων της προτεινόμενης λύσης είναι μεγαλύτερη από την ακρίβεια των δημοφιλών συμβατικών εργαλείων δοκιμών που χρησιμοποιήσαμε. Η διαφορά οφείλεται κυρίως στις δυνατότητες του Openflow πρωτοκόλλου που εκμεταλλεύεται η προτεινόμενη λύση, σε συνδυασμό με την αρχιτεκτονική SDN. Στην περίπτωση των συμβατικών εργαλείων δοκιμών, οι μετρήσεις των μεγεθών υπολογίζονται με την δημιουργία κίνησης στο δίκτυο όπως για παράδειγμα τα μηνύματα τύπου ICMP

που στέλνονται κατά την διάρκεια εκτέλεσης του εργαλείου Ping ή Traceroute, από αποστολέα σε παραλήπτη περιμένοντας απάντηση σε μορφή άλλων μηνυμάτων τύπου ICMP reply, ή ακόμα στην περίπτωση του εργαλείου Iperf όπου εκτελούνται διάφορες TCP δοκιμές μεταξύ δύο κόμβων δημιουργώντας έτσι μεγάλο όγκο δεδομένων προκειμένου να υπολογιστεί η μέγιστη τιμή του bandwidth μιας ζεύξης. Έτσι ο επιπλέον φόρτος δεδομένων επηρεάζει σημαντικά τα αποτελέσματα των πιο πάνω εργαλείων. Αντίθετα η προτεινόμενη λύση συλλέγει παθητικά δεδομένα από την κίνηση όπως αριθμό πακέτων και αριθμό bytes ανά ζεύξη, τα οποία χρησιμοποιεί για την καταμέτρηση των αντίστοιχων μεγεθών χωρίς την επιπλέον επιβάρυνση του δικτύου. Η διαφορά των εργαλείων με την προτεινόμενη λύση γίνεται πιο ξεκάθαρη όταν συγκρίνουμε την απόδοση τους στην ενότητα 4.3 όπου αποδείξαμε πως σε κάθε μορφή τοπολογίας γραμμική ή δέντρου οι μετρήσεις της λύσης μας είχαν ακριβέστερη τιμή συγκριτικά με τα εργαλεία που χρησιμοποιήθηκαν σε κάθε περίπτωση. Αρχικά οι υπολογισμοί για το bandwidth είχαν αρκετά μικρή απόκλιση από τις θεωρητικές τιμές που χρησιμοποιήθηκαν για σύγκριση. Έπειτα προχωρώντας στις μετρήσεις για το ποσοστό του Packet Loss συμπεράναμε ότι η προτεινόμενη λύση παρουσιάζει μεγαλύτερη ακρίβεια σε σχέση με τα εργαλεία που χρησιμοποιήσαμε για μέτρο σύγκρισης (Iperf και Ping). Τέλος για την σύγκριση του delay παρατηρούμε πως η ακρίβεια της προτεινόμενης λύσης έχει αρκετή διαφορά από την ακρίβεια των υπολοίπων εργαλείων στις περισσότερες περιπτώσεις, ενώ σε κάποιες τοπολογίες έχουν παρόμοια απόκλιση, με την προτεινόμενη λύση όμως να έχει ακριβέστερες μετρήσεις κάθε φορά.

5.3 Μελλοντική Εργασία

Ενώ με την προτεινόμενη μας λύση πετύχαμε την δημιουργία ενός σχετικά απλού διαχειριστή για την παρακολούθηση SDN δικτύων, εξακολουθεί να υπάρχει περιθώριο βελτίωσης ανάπτυξης της υλοποίησης μας στο μέλλον. Μία ενδιαφέρουσα βελτίωση στην προτεινόμενη λύση μπορεί να προσφέρει η ο συνδυασμός της με την τεχνική Network Tomography [37]. Το Network Tomography στηρίζεται στην συλλογή δεδομένων για την απόδοση ενός δικτύου βασιζόμενο πάνω στην κίνηση ανάμεσα σε ένα υποσύνολο κόμβων του δικτύου. Η χρήση ενός υποσυνόλου από τους κόμβους μιας τοπολογίας για την εξαγωγή στατιστικών είναι πολύ χρήσιμη τεχνική εφόσον για τον υπολογισμών των απαραίτητων μεγεθών θα επιβαρύνονται κάθε φορά οι επιλεγμένοι κόμβοι βελτιώνοντας έτσι σημαντικά την συνολική επίδοση του δικτύου. Επιπρόσθετα μία ενδιαφέρουσα μέθοδος με την οποία το Network Tomography προσεγγίζει τον σκοπό του και που θα μπορούσε να συνδυαστεί επίσης με την προτεινόμενή μας λύση είναι το Network Coding (NC). Η γενική ιδέα του NC είναι η ομαδοποίηση των πακέτων που λαμβάνουν οι ενδιάμεσοι μεταγωγείς μίας διαδρομής χρησιμοποιώντας απλές πράξεις πρόσθεσης και πολλαπλασιασμού. Τα ομαδοποιημένα πακέτα καταλήγουν εν τέλει στον τελικό κόμβο της διαδρομής όπου καταγράφονται ως ένας γραμμικός συνδυασμός των πακέτων που έχει στείλει ο κόμβος πηγή. Η προσέγγιση αυτή του Network Tomography μπορεί να χρησιμοποιηθεί

μαζί με την προτεινόμενη λύση βελτιώνοντας σημαντικά τις μετρήσεις της εφόσον όχι μόνο λαμβάνει πιο άμεσα τις επιθυμητές μετρήσεις που χρειάζεται για μια ζεύξη, αλλά περιορίζει την επιβάρυνση του δικτύου μόνο στο υποσύνολο κόμβων τους οποίους εξετάζει.

Άλλος πιθανός τομέας επέκτασης της προτεινόμενης μας λύσης είναι η συμβατότητα της με κατανεμημένες πλατφόρμες ελεγκτών, με δίκτυα δηλαδή που αποτελούνται με περισσότερους από ένα κεντρικό διαχειριστή. Σύμφωνα με την πηγή [38] ένα από τα μειονεκτήματα των δικτύων οριζόμενων από λογισμικό είναι ο περιορισμός του δικτύου σε απόδοση λόγω της προσέγγισης τους με έναν κεντρικό ελεγκτή, ειδικότερα σε τοπολογίες μεγάλου μεγέθους. Όπως αναφέρεται στο πιο πάνω άρθρο αυτός ο περιορισμός μπορεί να αντιμετωπιστεί με το πρόβλημα παροχής δυναμικού ελεγκτή (Dynamic Controller Provisioning Problem, DCPP). Με το DCPP καθορίζονται δυναμικά ο αριθμός των διαχειριστών ενός δικτύου καθώς και η τοποθέτησή τους στο δίκτυο βάσει της ανά πάσα στιγμής κατάστασης του δικτύου με τέτοιο τρόπο ώστε να περιοριστούν όσο το δυνατό περισσότερο ο χρόνος ρύθμισης των απαραίτητων ροών καθώς και ο όγκος των δεδομένων κυκλοφορίας στο δίκτυο. Ο συνδυασμός της τεχνικής DCPP με την προτεινόμενη λύση της διπλωματικής μας θα ήταν πολύ ενδιαφέρον εφόσον με τον περιορισμένο όγκο κυκλοφορίας που θα έχει ως αποτέλεσμα η χρήση πολλαπλών διαχειριστών σε μία τοπολογία η ακρίβεια των μετρήσεων μας για Packet Loss Bandwidth Delay θα ήταν ακόμα μεγαλύτερη. Το κατανεμημένο σύστημα μοιράζει την κυκλοφορία που θα υπήρχε μεταξύ μεταγωγέων – διαχειριστή μεταξύ των πολλαπλών διαχειριστών που χρησιμοποιεί. Έτσι το δίκτυο θα υφίσταται κατανεμημένο και άρα λιγότερο φόρτο σε όλες του της ζεύξης του, δίνοντας έτσι περισσότερη ακρίβεια στις μετρήσεις που θα εξάγει η προτεινόμενη λύση.

Κεφάλαιο 6: Βιβλιογραφία

[1] K. S. Nash, A. Behr Network Monitoring Definition and Solutions. (2007). http://www.cio.com/article/133700/Network_Monitoring_Definition_and_Solutions

[2] F. Hu, Q. Hao and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," in *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181-2206, Fourthquarter 2014, doi: 10.1109/COMST.2014.2326417.

[3] P. Arote and K. V. Arya, "Detection and Prevention against ARP Poisoning Attack Using Modified ICMP and Voting," 2015 International Conference on Computational Intelligence and Networks, Bhubaneshwar, 2015, pp. 136-141, doi: 10.1109/CINE.2015.34.

[4] Claise, B.: Cisco Systems NetFlow Services Export Version 9 [online]. [cit. 2015-04-21]. URL, https://tools.ietf.org/html/rfc3954

[5] Wikipedia: The Free Encyclopedia, “Ping (networking utility),” retrieved on 17 March 2012, available at http://en.wikipedia.org/wiki/Ping_(networking_utility)

[6] N. Q. Do, H. S. Ong, L. C. Lai, Y. X. Che and X. J. Ong, "Open-source testing tools for smart grid communication network," 2013 IEEE Conference on Open Systems (ICOS), Kuching, 2013, pp. 156-161, doi: 10.1109/ICOS.2013.6735066.

[7] Iperf, retrieved on 20 February 2013, available at http://iperf.sourceforge.net/

[8] R. Jones, “Netperf,” retrieved on 20 February 2013, available at http://www.netperf.org/netperf/

[9] Nuttcp Development Team, “Nuttcp,” retrieved on 20 February 2013, available at http://www.nuttcp.net/nuttcp/Welcome%20Page.html

[10] Fabien Viger, Brice Augustin, Xavier Cuvellier, Clémence Magnien, Matthieu Latapy, Timur Friedman, Renata Teixeira, Detection, understanding, and prevention of traceroute measurement artifacts, Computer Networks, Volume 52, Issue 5, 2008, Pages 998-1018, ISSN 1389-1286

[11] J. D. Case, M. S. Fedor, M. L. Schoffstall και J. R. Davin, «Simple Network Management Protocol (SNMP),» pp. 1-36, 1990

[12] B. Claise και Ed, «Cisco Systems NetFlow Services Export Version 9,» pp. 1-33, 2004

[13] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella και D. G. Andersen, «CSAMP: A System for Network-Wide Flow Monitoring,» pp. 233-246, 2008.

[14] B. Huffaker, D. Plummer, D. Moore και K. Claffy, «Topology discovery by active probing,» σε Proceedings 2002 Symposium on Applications and the Internet (SAINT) Workshops, 2002, pp. 90-96.

[15] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007. Ethane: taking control of the enterprise. SIGCOMM Comput. Commun. Rev. 37, 4 (October 2007), 1–12. DOI:https://doi.org/10.1145/1282427.1282382

[16] Zurich University of Applied Sciences, 2020. An Introduction to Software-Defined Networking (SDN). [Blog] SERVICE ENGINEERING (ICCLAB & SPLAB), Available at: <https://blog.zhaw.ch/icclab/an-introduction-to-software-defined-networking-sdn/#comments> [Accessed 21 September 2020].

[17] Open Networking Foundation. 2020. [online] Available at: <https://www.opennetworking.org/> .

[18]Sarai, R., 2019. Software Defined Network And The Openflow Protocol Live

[19]Khatri, Vikramajeet. (2013). M.Sc. Thesis: Analysis of OpenFlow Protocol in Local Area Networks.

[20] Tootoonchian A., Ghobadi M., Ganjali Y. (2010) OpenTM: Traffic Matrix Estimator for OpenFlow Networks. In: Krishnamurthy A., Plattner B. (eds) Passive and Active Measurement. PAM 2010. Lecture Notes in Computer Science, vol 6032. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-12334-4_21

[21] Yu, Curtis & Lumezanu, Cristian & Zhang, Yueping & Singh, Vishal & Jiang, Guofei & Madhyastha, Harsha. (2013). FlowSense: Monitoring Network Utilization with Zero Measurement Cost. Passive and active measurement (PAM). 7799. 31-41. 10.1007/978-3-642-36516-4_4.

[22] S. R. Chowdhury, M. F. Bari, R. Ahmed and R. Boutaba, "PayLess: A low cost network monitoring framework for Software Defined Networks" 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, 2014, pp. 1-9, doi: 10.1109/NOMS.2014.6838227

[23] N. L. M. van Adrichem, C. Doerr and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks," *2014 IEEE Network Operations and Management Symposium (NOMS)*, Krakow, 2014, pp. 1-8, doi: 10.1109/NOMS.2014.6838228.

[24] Casimer DeCusatis, Aparicio Carranza and Jean Delgado-Caceres, "Modeling Software Defined Networks using Mininet", *Proceedings of the 2nd International Conference on Computer and Information Science and Technology (CISTI6)*, no. 133, May 11 12,2016

[25] Kaur, Sukhveer & Singh, Japinder & Ghumman, Navtej. (2014). Network Programmability Using POX Controller. 10.13140/RG.2.1.1950.6961.

[26] Linkletter, B., 2015, Install Gephi on the Mininet VM, Linkletter, B., http://www.brianlinkletter.com/install-gephi-on-the-mininet-vm/

[27] Bastian M, Heymann S, Jacomy M (2009) Gephi: an open source software for exploring and manipulating networks. In: International AAAI Conference on Weblogs and Social Media. Association for the Advancement of Artificial Intelligence.

[28] Linkletter, B., 2015, Using the POX SDN Controller, Linkletter, B., http://www.brianlinkletter.com/using-the-pox-sdn-controller/

[29] K. Alrashedy, B. Kimmett and T. A. Gulliver, "Performance of software-defined networking controllers for different network topologies," 2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), Victoria, BC, 2017, pp. 1-4, doi: 10.1109/PACRIM.2017.8121925.

[30] Kaur S, Singh J, Ghumman NS (2014) Network programmability using POX controller. International conference on communication, computing & systems, at SBS Staten technical campus, Ferozepur, Punjab, India, volume: 1

[31] Kadam A., 2016, Implementing a Learning Switch and adding a flow entry to the switch using POX, Kadam A., https://ajinkyakadam.bitbucket.io/blogpost/2016/08/14/lswitch/

[32] Stanford University, Openflow, *POX Wiki* https://openflow.stanford.edu/display/ONL/POX+Wiki.html#POXWiki-forwarding.l2_learning,

[33] Santos, Renato & Ribeiro, Thiago & Cesar, Cecilia. (2015). A network monitor and controller using only OpenFlow. 9-16. 10.1109/LANOMS.2015.7332663.

[34] Goyal M.K., Verma Y.K., Bassi P., Misra P.K. (2012) Performance Analysis of TCP & UDP in Co-located Variable Bandwidth Environment Sharing Same Transmission Links. In: Meghanathan N., Nagamalai D., Chaki N. (eds) Advances in Computing and Information Technology. Advances in Intelligent Systems and Computing, vol 176. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-31513-8_31

[35] Kili, A., 2020. Nload - Monitor Linux Network Bandwidth Usage In Real Time. [online] Tecmint.com. Available at: https://www.tecmint.com/nload-monitor-linux-network-traffic-bandwidth-usage/

[36] Alghadhban, Amer & Shihada, Basem. (2018). Delay Analysis of New-Flow Setup Time in Software Defined Networks. 10.1109/NOMS.2018.8406231.

[37] Kakkavas, Grigorios & Gkatzioura, Despina & Karyotis, Vasileios & Papavassiliou, Symeon. (2020). A Review of Advanced Algebraic Approaches Enabling Network Tomography for Future Network Infrastructures. Future Internet. 12. 20. 10.3390/fi12020020.

[38] M. F. Bari *et al.*, "Dynamic Controller Provisioning in Software Defined Networks," *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, Zurich, 2013, pp. 18-25, doi: 10.1109/CNSM.2013.6727805.

[39] Linkletter, B., 2015, Visualizing software defined network topologies using POX and Gephi, Linkletter, B., https://www.brianlinkletter.com/visualizing-software-defined-network-topologies-using-pox-and-gephi/#3

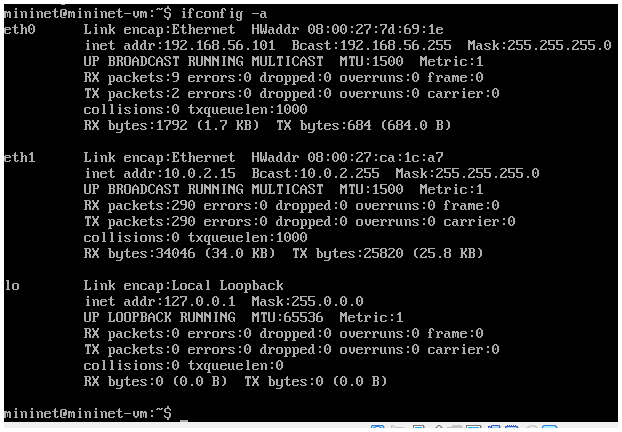
Κεφάλαιο 7: Παράρτημα

7.1 Εγκατάσταση και εισαγωγή στο εικονικό μηχάνημα

Για να στήσουμε το εικονικό μηχάνημα Mininet θα χρειαστεί πρώτα η εγκατάσταση των VirtualBox Xming και PuTTy καθώς και του image file που αντιστοιχεί στο εργαλείο mininet. Φορτώνουμε το VirtualBox και κάνουμε import το αρχείο .ovf όπως αναφέρεται στον οδηγό εγκατάστασης. Μετά την εγκατάσταση των πιο πάνω φορτώνουμε το εικονικό μηχάνημα και δίνουμε username και password "mininet" όπου ζητηθούν. Εκτελούμε την εντολή **sudo dhclient eth1** για να αποδώσουμε ip διεύθυνση και στη δεύτερη θύρα (adapter). Μπορούμε να μορφοποιήσουμε το αρχείο /etc/network/interfaces ούτως ώστε αυτό να γίνεται αυτόματα μετά την φόρτωση του μηχανήματος προσθέτοντας του τις ακόλουθες εντολές

```
auto eth1
iface eth1 inet dhcp
```

Μπορούμε να δούμε τις IP διευθύνσεις των 2 θυρών του μηχανήματος εκτελώντας την εντολή **ifconfig –a**. Η θύρα που μας ενδιαφέρει είναι η Host-Only με διεύθυνση IP 192.168.56 εφόσον αυτή θα χρησιμοποιούμε για να συνδεθούμε με SSH (Secure Shell) με την πλατφόρμα PuTTy που εγκαταστήσαμε



Εικόνα 30: Αποτέλεσμα της εντολής ifconfig -a

Για ευκολία λοιπόν, κατά την πρόσβαση και επεξεργασίας δεδομένων στο mininet θα χρησιμοποιήσουμε το **PuTTy** για να συνδεθούμε μέσω ssh στο εικονικό μας μηχάνημα καθώς και τον εξυπηρετητή **Xming.** Φορτώνοντας το PuTTy, και πληκτρολογώντας την IP διεύθυνση της θύρας «Host-Only» που είδαμε προηγούμενος συνδεόμαστε με επιτυχία στο μηχάνημα και δίνουμε και πάλι . Αφού «username» και «password» όπως θα τα δίναμε αν το φορτώναμε κανονικά.

7.2 Εργαλείο Gephi

Ένα άλλο εργαλείο που θα φανεί χρήσιμο στην παρούσα διπλωματική είναι το Gephi. Για την εγκατάσταση του Gephi είναι απαραίτητη η εγκατάσταση της Java στο μηχάνημά μας. Αφού εγκατασταθούν μπορούμε να το χρησιμοποιήσουμε για δημιουργία διαφόρων τοπολογιών. Για να τρέξουμε τον SDN διαχειριστή για το Gephi ανοίγουμε νέο παράθυρο mininet και εκτελούμε την πιο κάτω εντολή:

```
sudo ~/pox/pox.py forwarding.l2_learning \
openflow.discovery misc.gephi_topo \
openflow.spanning_tree --no-flood --hold-down \
host_tracker info.packet_dump \
samples.pretty_log log.level --DEBUG
```

Έπειτα σε καινούριο παράθυρο εκτελούμε την εντολή sudo ./gephi για να φορτώσουμε το Gephi. Με τις κατάλληλες ρυθμίσεις προσαρμόζουμε το Gephi ώστε να συνδεθεί με τον διαχειριστή που τρέχουμε και στη συνέχεια σε νέο παράθυρο του Mininet τρέχουμε μια τοπολογία δικτύου

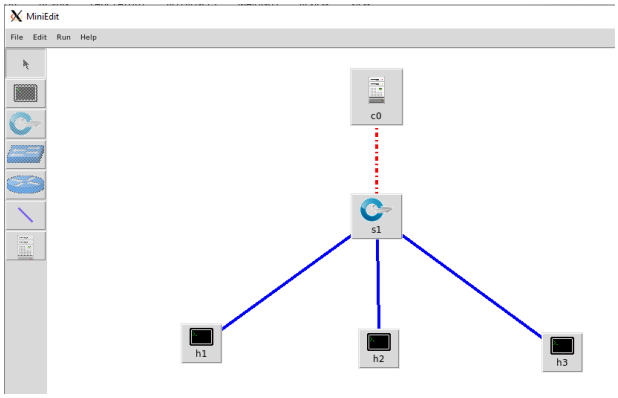
```
sudo mn --topo torus,3,3 --controller remote
```

Θα δούμε στην οθόνη να εμφανίζονται πολλές κουκκίδες που αντιπροσωπεύουν τους κόμβους της τοπολογίας. Με τις κατάλληλες ρυθμίσεις τις και με τη βοήθεια του οδηγού [39] τις συγκεντρώνουμε για να είναι πιο ευδιάκριτες. Στο παράθυρο που τρέχει το mininet εκτελούμε την εντολή pingall για να ανιχνεύσει ο διαχειριστής τους κόμβους του δικτύου και να εμφανίσει τις κουκκίδες που τους αντιπροσωπεύουν.

Εικόνα 31: Τοπολογία τύπου torus με 9 switch και 9 hosts στο Gephi

7.3 Miniedit

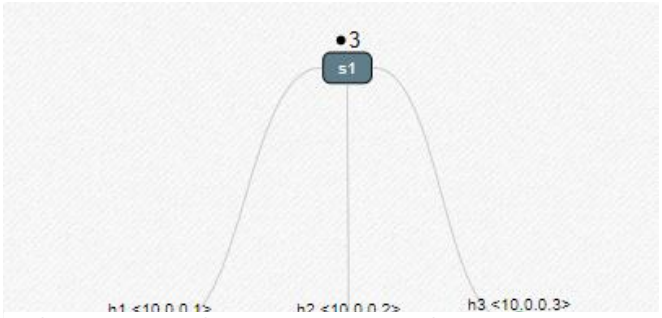
Μπορούμε εναλλακτικά να χρησιμοποιήσουμε το εργαλείο MiniEdit, ένα γραφικό περιβάλλον που προσφέρει το mininet για την εύκολη προς τον χρήστη δημιουργία τοπολογιών. Φορτώνουμε το MiniEdit με την εντολή **sudo ~/mininet/examples/miniedit.py** Στο νέο περιβάλλον εργασίας που εμφανίζεται μπορούμε να χρησιμοποιήσουμε τα εικονίδια στο αριστερό μέρος για να προσθέσουμε στην τοπολογία μας κόμβους, μεταγωγείς ή τον διαχειριστή του δικτύου. Στην επιλογή «Properties» για κάθε κόμβο μπορούμε να τροποποιήσουμε τα στοιχεία όπως για παράδειγμα να ρυθμίσουμε την IP διεύθυνσή του. Μετά τις απαραίτητες ρυθμίσεις τρέχουμε την τοπολογία με την επιλογή «Run»



Εικόνα 32: Τοπολογία με 1 switch και 3 hosts στο minedit

7.4 SDN Narmox Spear

Για μια πιο ξεκάθαρη εικόνα του δικτύου που δημιουργήσαμε μπορούμε να χρησιμοποιήσουμε το online εργαλείο **SDN Narmox Spear** που είναι διαθέσιμο στον παρακάτω σύνδεσμο http://mininet.spear.narmox.com/. Το εργαλείο αυτό παίρνει ως είσοδο το αποτέλεσμα της εντολής **links** και **dump** και απεικονίζει στη συνέχεια το δίκτυο που δημιουργήσαμε στο mininet με τους κόμβους και τις συνδέσεις τους. Παραδείγματος χάρη για την απλή τοπολογία με ένα μεταγωγέα που συνδέεται με τρεις κόμβους εφόσον μεταφέρουμε τα απαραίτητα στοιχεία στο **SDN Narmox Spear**, εξάγει το εξής αποτέλεσμα:

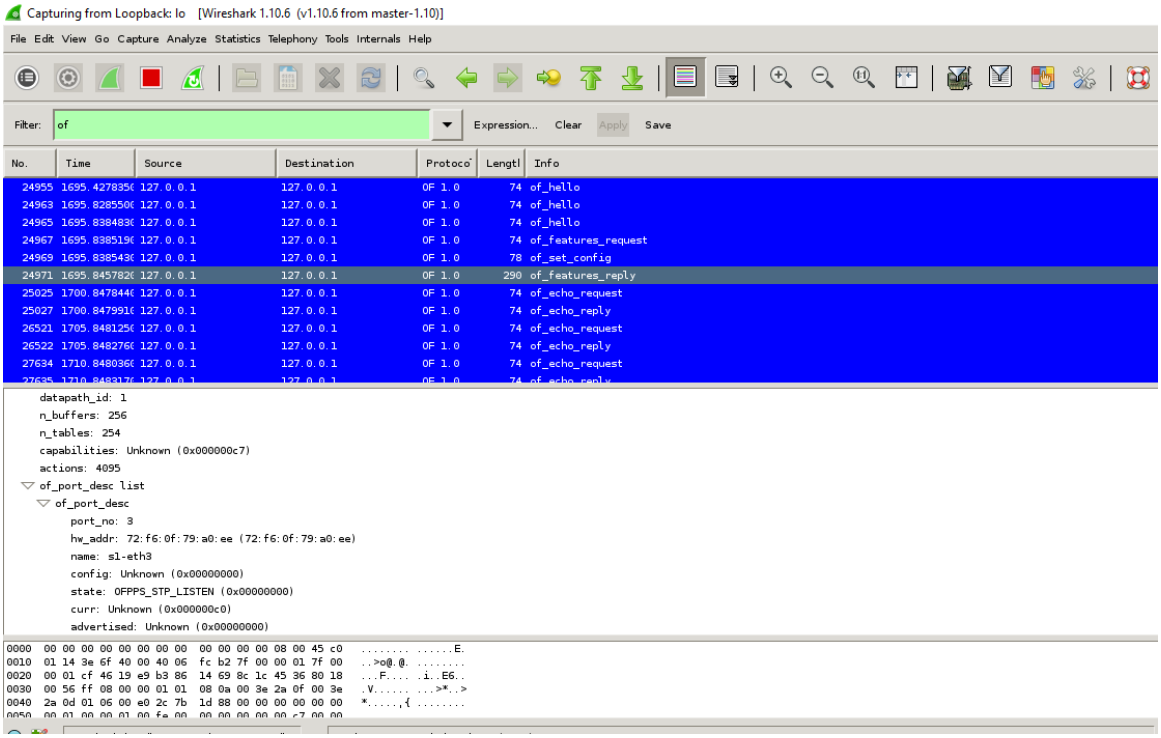


Εικόνα 33: Αναπαράσταση single τοπολογίας με 3 hosts στο SDN Narmox Spear

7.5 Wireshark

Προκειμένου να φορτώσουμε το wireshark, ανοίγουμε νέο ssh παράθυρο με το PyTTy, με επιλεγμένη την επιλογή X11 forwarding και φορτώνουμε το wireshark με την εντολή **sudo wireshark&.** Επιλέγουμε να παρακολουθούμε την loopback διεπαφή (interface) και ξεκινούμε την καταγραφή. Για την καταγραφή συγκεκριμένης κίνησης στο δίκτυο φιλτράρουμε τα πακέτα που μας ενδιαφέρουν εισάγοντας φίλτρο στο πεδίο «filter». Για να καταγράψουμε για παράδειγμα την openflow κίνηση μεταξύ μεταγωγέων και του διαχειριστή, δίνουμε σαν παράμετρο το φίλτρο “of”. Τρέχουμε κα πάλι την απλή τοπολογία με τον μεταγωγέα συνδεδεμένο με τρεις κόμβους. Παρατηρούμε ότι στο wireshark εμφανίζονται OF πακέτα :

Στην αρχή παρατηρούμε **Hello** πακέτα από controller σε switch και αντίστροφα. Μετά έχουμε τα πακέτα **features request** και **set config** από τον διαχειριστή στον μεταγωγέα όπου ζητά να μάθει τις διαθέσιμες θύρες και στοιχεία για τις ροές αντίστοιχα και τέλος έχουμε την απάντηση του μεταγωγέα με το **features reply**



Εικόνα 34: Καταγραφή openflow πακέτων στο wireshark

7.6 Διαχείριση εντολών στο Mininet (Command-Line Interface)

Όπως έχουμε αναφέρει με το mininet μπορούμε να δημιουργήσουμε τοπολογίες δικτύων με μία εντολή. Για παράδειγμα με την εντολή **sudo mn --topo single,3 --mac --switch ovsk --controller remote** δημιουργείται μια απλή τοπολογία με τρεις κόμβους που συνδέονται με έναν μεταγωγέα. Συγκεκριμένα με το όρισμα –topo επιλέγουμε την μορφή της τοπολογίας, με το όρισμα –switch καθορίζουμε τον τύπο του μεταγωγέα που χρησιμοποιείται στην τοπολογία και ομοίως με το όρισμα –controller καθορίζουμε τον τύπο του διαχειριστή. Τέλος το όρισμα –mac ανέθεσε και 3 ip διευθύνσεις στους τρεις κόμβους και έθεσε τις mac διευθύνσεις τους ίσες με τις ip τους.

Για να εμφανίσουμε όλους τους κόμβους μιας τοπολογίας εκτελούμε την εντολή **nodes.** Επίσης για να δούμε τον τρόπο με τον οποίο οι κόμβοι του δικτύου είναι συνδεδεμένοι μεταξύ τους εκτελούμε την εντολή **net**, η οποία εμφανίζει όλα τις ακμές της τοπολογίας. Για να εκτελέσουμε οποιαδήποτε εντολή για έναν συγκεκριμένο κόμβο απλά προσθέτουμε το όνομα του κόμβου μπροστά από την εντολή. Για παράδειγμα με την εντολή **h1 ifconfig** μπορούμε να δούμε πληροφορίες που αφορούν μόνο τον κόμβο h1 Για να εμφανίσουμε τις ροές ενός μεταγωγέα εκτελούμε την εντολή **sudo ovs-ofctl dump-flows s1** η οποία εμφανίζει τον πίνακα με τις ροές ενός μεταγωγέα εφόσον υπάρχουν.

Στήνοντας την πιο πάνω τοπολογία μπορούμε να ελέγξουμε την επικοινωνία τον κόμβων της με την χρήση της εντολής **ping**. Με την εντολή αυτή στέλνουμε πακέτο ICMP request προς τον κόμβο που ορίζουμε και αναμένουμε απάντηση τύπου ICMP reply. Όμως επειδή στην προκειμένη περίπτωση δεν πήραμε καμία απάντηση πράγμα που οφείλεται δεν έχουμε ακόμα ροές εγκατεστημένες στον μεταγωγέα του δικτύου. Μπορούμε να προσθέσουμε ροές χωρίς την χρήση για να γίνει επιτυχές το ping μεταξύ των κόμβων με τις εντολές

 # ovs-ofctl add-flow s1 in_port=1,actions=output:2

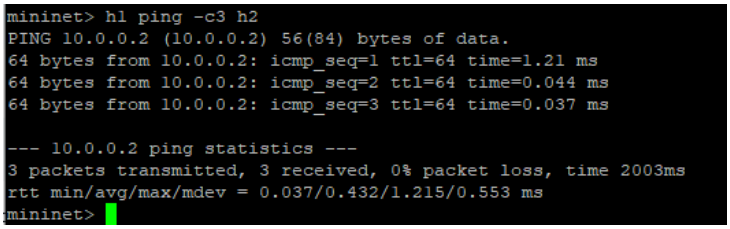
 # ovs-ofctl add-flow s1 in_port=2,actions=output:1

οι οποίες δίνουν εντολή στον μεταγωγέα να προωθεί πακέτα που προέρχονται από την θύρα 1 προς την θύρα 2 και αντίστροφα. Εάν εμφανίσουμε τώρα τον πίνακα ροών του μεταγωγέα θα παρατηρήσουμε τις δύο καταχωρήσεις μας και εκτελώντας την εντολή ping η επικοινωνία μεταξύ των κόμβων που βρίσκονται στις θύρες 1 και 2 θ

α είναι επιτυχής.

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=16.964s, table=0, n_packets=0, n_bytes=0, idle_age=16, in_
port=1 actions=output:2
 cookie=0x0, duration=3.822s, table=0, n_packets=0, n_bytes=0, idle_age=3, in_po
rt=2 actions=output:1
mininet@mininet-vm:~$
```

Εικόνα 35: Το flow table του switch μετά την προσθήκη των 2 flow



Εικόνα 36: Η εντολή ping ολοκληρώθηκε με επιτυχία

7.7 Πηγαίος κώδικας του διαχειριστή ενότητας 4.3.1

```
.
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str, str_to_dpid
from pox.lib.util import str_to_bool
import time
from pox.lib.recoco import Timer
from pox.lib.addresses import IPAddr, EthAddr
import pox.lib.packet as pkt
from pox.openflow.of_json import *
from pox.lib.packet.packet_base import packet_base
from pox.lib.packet.packet_utils import *
import struct
from pox.openflow.of_json import *
_flood_delay = 0
log = core.getLogger()
counts=[0 for i in range (6)]
flow_stats=[]
flow_src_dst=[0 for i in range (2)]
timercount=0
switchcount=0
start_time = 0.0
sent_time1=0.0
sent_time2=0.0
received_time1 = 0.0
received_time2 = 0.0
src_dpid=0
dst_dpid=0
mytimer = 0
OWD1=0.0
OWD2=0.0

class myproto(packet_base):
  "My Protocol packet struct"
  def __init__(self):
    packet_base.__init__(self)
    self.timestamp=0
```

```
  def hdr(self, payload):
    return struct.pack('!I', self.timestamp)
#switchID=0
## handler for timer function that sends the requests to all the
## switches connected to the controller.
def _timer_func ():
  global start_time, sent_time1, sent_time2, src_dpid, dst_dpid
  if src_dpid <>0:
    sent_time1=time.time() * 1000 - start_time
    #send out port_stats_request packet through src_dpid
    core.openflow.getConnection(src_dpid).send(of.ofp_stats_request(body=of.ofp_port_stats_request()))
    core.openflow.getConnection(src_dpid).send(of.ofp_stats_request(body=of.ofp_flow_stats_request()))
  if dst_dpid <>0:
    sent_time2=time.time() * 1000 - start_time
    #send out port_stats_request packet through src_dpid
    core.openflow.getConnection(dst_dpid).send(of.ofp_stats_request(body=of.ofp_port_stats_request()))
  #for connection in core.openflow._connections.values():
    #connection.send(of.ofp_stats_request(body=of.ofp_flow_stats_request()))
    #connection.send(of.ofp_stats_request(body=of.ofp_port_stats_request()))
  #log.debug("Sent %i flow/port stats request(s)", len(core.openflow._connections))
  f = myproto()
  f.timestamp = int(time.time()*1000 - start_time)
  e = pkt.ethernet()
  e.src=EthAddr("0:0:0:0:0:2")
  e.dst=EthAddr("0:1:0:0:0:1")
  e.type=0x5577
  e.payload = f
  msg = of.ofp_packet_out()
  msg.data = e.pack()
  msg.actions.append(of.ofp_action_output(port=3))
  core.openflow.getConnection(src_dpid).send(msg)
def _handle_ConnectionDown (event):
  global mytimer
  print "ConnectionDown: ", dpidToStr(event.connection.dpid)
  #mytimer.cancel()
def _handle_ConnectionUp (event):
  global src_dpid, dst_dpid, mytimer
  print "ConnectionUp: ", dpidToStr(event.connection.dpid)
  #remember the connection dpid for switch to controller (src_dpid) and switch1 to controller(dst_dpid)
  for m in event.connection.features.ports:
    if m.name == "s1-eth1":
      src_dpid = event.connection.dpid
    elif m.name == "s8-eth2":
      dst_dpid = event.connection.dpid
  # when the controller knows both src_dpid and dst_dpid, the probe packet is sent out every 2 seconds
  if src_dpid<>0 and dst_dpid<>0:
    Timer(2, _timer_func, recurring=True)
    #Timer(5, _timer_func, recurring=True)
# handler to display flow statistics received in JSON format
# structure of event.stats is defined by ofp_flow_stats()
def _handle_flowstats_received (event):
  stats = flow_stats_to_list(event.stats)
  log.debug("FlowStatsReceived from %s: %s",
    dpidToStr(event.connection.dpid), stats)
```

```
#  # Get number of bytes/packets in flows for web traffic only
  global flow_stats
  global flow_src_dst
  global timercount
  global counts, src_dpid, dst_dpid
  currbytes = 0
  flows = 0
  packet = 0
  dur = 0
  throughput = 0
  for f in event.stats:
    counts[0] = event.stats[0].match.nw_src
    counts[1] = event.stats[0].match.nw_dst
    currbytes +=f.byte_count
    packet +=f.packet_count
    dur = f.duration_sec + (f.duration_nsec/1000000000)
  if ((packet - counts[2] >0) and (currbytes - counts[3]>0) and (dur - counts[4]>0)):
    counts[2] = packet - counts[2]
    counts[3] = currbytes - counts[3]
    counts[4] = dur - counts[4]
    if counts[4] != 0:
      counts[5] = counts[3] / counts[4]
  if event.stats:
    log.info("Traffic from %s to %s: %s bytes (%s packets) in %s seconds. Current thoughput %s",
      counts[0],counts[1], counts[3], counts[2],counts[4],counts[5])
  if not event.stats:
    timercount+=1
  if timercount==3:
    flow_src_dst=[]
    flow_stats=[]
    counts = [0 for i in range (6)]
    timercount=0
    print("\nReinitialisiong flow lists")
# handler to display port statistics received in JSON format
def _handle_portstats_received (event):
  global start_time, sent_time1, sent_time2, received_time1, received_time2, src_dpid, dst_dpid,OWD1,OWD2
  received_time = time.time() * 1000 - start_time
   #measure T1
  if event.connection.dpid == src_dpid:
    OWD1=0.5*(received_time - sent_time1)
   #measure T2
  elif event.connection.dpid == dst_dpid:
    OWD2=0.5*(received_time - sent_time2)
    #print "OWD2: ", OWD2, "ms"
  stats = flow_stats_to_list(event.stats)
  ##if ((dpidToStr(event.connection.dpid) ==src_dpid) or (dpidToStr(event.connection.dpid) ==dst_dpid)):
  print('switch '+str(event.connection.dpid))
  if event.stats:
    for f in event.stats:
      print("port number: ", f.port_no,"transmitted packets: ", f.tx_packets, " received packets: ", f.rx_packets)
  log.debug("PortStatsReceived from %s: %s",
    dpidToStr(event.connection.dpid), stats)
class LearningSwitch (object):
 """
```

```
  The learning switch "brain" associated with a single OpenFlow switch.
  When we see a packet, we'd like to output it on a port which will
  eventually lead to the destination.  To accomplish this, we build a
  table that maps addresses to ports.
  We populate the table by observing traffic.  When we see a packet
  from some source coming from some port, we know that source is out
  that port.
  When we want to forward traffic, we look up the desintation in our
  table.  If we don't know the port, we simply send the message out
  all ports except the one it came in on.  (In the presence of loops,
  this is bad!).
  In short, our algorithm looks like this:
  For each packet from the switch:
  1) Use source address and switch port to update address/port table
  2) Is transparent = False and either Ethertype is LLDP or the packet's
     destination address is a Bridge Filtered address?
     Yes:
        2a) Drop packet -- don't forward link-local traffic (LLDP, 802.1x)
            DONE
  3) Is destination multicast?
     Yes:
        3a) Flood the packet
            DONE
  4) Port for destination address in our address/port table?
     No:
        4a) Flood the packet
            DONE
  5) Is output port the same as input port?
     Yes:
        5a) Drop packet and similar ones for a while
  6) Install flow table entry in the switch so that this
     flow goes out the appopriate port
     6a) Send the packet out appropriate port
  """
  def __init__ (self, connection, transparent):
    global switchcount
    switchcount+=1
    # Switch we'll be adding L2 learning switch capabilities to
    self.connection = connection
    self.transparent = transparent
    # Our table
    self.macToPort = {}
    # We want to hear PacketIn messages, so we listen
    # to the connection
    connection.addListeners(self)
    # We just use this to know when to log a helpful message
    self.hold_down_expired = _flood_delay == 0
    #log.debug("Initializing LearningSwitch, transparent=%s",
    #          str(self.transparent))
  def _handle_PacketIn (self, event):
    global start_time,OWD1,OWD2
    packet = event.parsed
    received_time = time.time() * 1000 - start_time
    if packet.type==0x5577 and event.connection.dpid==dst_dpid:
```

```
  c=packet.find('ethernet').payload
  d,=struct.unpack('!I', c)
  print "delay:", received_time - d - OWD1-OWD2, "ms"
def flood (message = None):
  msg = of.ofp_packet_out()
  if time.time() - self.connection.connect_time >= _flood_delay:
    # Only flood if we've been connected for a little while...
    if self.hold_down_expired is False:
      # Oh yes it is!
      self.hold_down_expired = True
      log.info("%s: Flood hold-down expired -- flooding",
          dpid_to_str(event.dpid))
    if message is not None: log.debug(message)
    #log.debug("%i: flood %s -> %s", event.dpid,packet.src,packet.dst)
    # OFPP_FLOOD is optional; on some switches you may need to change
    # this to OFPP_ALL.
    msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
  else:
    pass
    #log.info("Holding down flood for %s", dpid_to_str(event.dpid))
  msg.data = event.ofp
  msg.in_port = event.port
  self.connection.send(msg)
def drop (duration = None):
  if duration is not None:
    if not isinstance(duration, tuple):
      duration = (duration,duration)
    msg = of.ofp_flow_mod()
    msg.match = of.ofp_match.from_packet(packet)
    msg.idle_timeout = duration[0]
    msg.hard_timeout = duration[1]
    msg.buffer_id = event.ofp.buffer_id
    self.connection.send(msg)
  elif event.ofp.buffer_id is not None:
    msg = of.ofp_packet_out()
    msg.buffer_id = event.ofp.buffer_id
    msg.in_port = event.port
    self.connection.send(msg)
self.macToPort[packet.src] = event.port # 1
if not self.transparent: # 2
  if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
    drop() # 2a
    return
if packet.dst.is_multicast:
  flood() # 3a
else:
  if packet.dst not in self.macToPort: # 4
    flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
  else:
    port = self.macToPort[packet.dst]
    if port == event.port: # 5
      log.warning("Same port for packet from %s -> %s on %s.%s.  Drop."          # 5a
          % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
      drop(10)
```

```
        return
      log.debug("installing flow for %s.%i -> %s.%i" %        # 6
            (packet.src, event.port, packet.dst, port))
      msg = of.ofp_flow_mod()
      msg.match = of.ofp_match.from_packet(packet, event.port)
      msg.idle_timeout = 10
      msg.hard_timeout = 30
      msg.actions.append(of.ofp_action_output(port = port))
      msg.data = event.ofp # 6a
      self.connection.send(msg)
class l2_learning (object):
  def __init__ (self, transparent, ignore = None ):
    core.openflow.addListeners(self)
    self.transparent = transparent
    self.ignore = set(ignore) if ignore else ()
  def _handle_ConnectionUp (self, event):
    if event.dpid in self.ignore:
      log.debug("Ignoring connection %s" % (event.connection,))
      return
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)
def launch (transparent=False, hold_down=_flood_delay, ignore = None):
  from pox.lib.recoco import Timer
  global switchID
  try:
    global _flood_delay
    _flood_delay = int(str(hold_down), 10)
    assert _flood_delay >= 0
  except:
    raise RuntimeError("Expected hold-down to be a number")
  if ignore:
    ignore = ignore.replace(',', ' ').split()
    ignore = set(str_to_dpid(dpid) for dpid in ignore)
  global start_time
  start_time = time.time() * 1000
  print "start_time:", start_time
  core.registerNew(l2_learning, str_to_bool(transparent), ignore)
# attach handsers to listners
  core.openflow.addListenerByName("FlowStatsReceived",
    _handle_flowstats_received)
  core.openflow.addListenerByName("PortStatsReceived",
    _handle_portstats_received)
  core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)
  core.openflow.addListenerByName("ConnectionDown", _handle_ConnectionDown)
```