.ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής

Εξομοίωση δικτύων υπολογιστών με τη χρήση Mininet

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΙΑΝΟΥ ΠΑΠΑΘΕΟΔΩΡΟΥ

Επιβλέπων : Ευστάθιος Συκάς
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την.....Ιουνίου 2015.

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

Αθήνα, Ιούνιος 2015

(Υπογραφή)

.....

ΙΑΝΟΣ ΠΑΠΑΘΕΟΔΩΡΟΥ

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

Περίληψη

Η εξομοίωση δικτύων αποτελεί χωρίς καμία αμφιβολία την επικρατέστερη μεθοδολογία εξαγωγής συμπερασμάτων στην περιοχή των δικτύων υπολογιστών. Χρησιμοποιείται ευρέως για τη δημιουργία τοπολογιών και διαδικτυακών πρωτοκόλλων. Οι καλούμενοι εξομοιωτές δικτύων υπολογιστών μας επιτρέπουν να μοντελοποιούμε ένα δίκτυο υπολογιστών, καθορίζοντας τη συμπεριφορά του κόμβων και των συνδέσεων μεταξύ τους. Στη παρούσα διπλωματική εργασία θα ασχοληθούμε με την εικονικοποίηση δικτύων και πώς αυτή πραγματοποιείται. Στη συνέχεια θα δούμε πώς χρησιμοποιείται η εικονικοποίηση δικτύων για την εξομοίωση αυτών μέσω των κατάλληλων προγραμμάτων και τέλος θα αναπτύξουμε τον κατάλληλο κώδικα για τη δημιουργία διαφορετικών τοπολογιών και θα δώσουμε ένα εκτενές παράδειγμα πώς μπορεί να χρησιμοποιηθεί σε εργαστηριακές ασκήσεις,τέλος θα αναφερθούμε σε μελλοντικές προοπτικές του mininet όσον αφορά το πρωτόκολλο openflow. Είναι σημαντικό να καταστήσουμε σαφές ότι προσπαθούμε να φτιάξουμε ένα περιβάλλον όπου ο σπουδαστής-ερευνητής θα μπορεί να μελετά/εξομοιώνει πολύπλοκες τοπολογίες με διαφορετικούς αλγορίθμους δρομολόγησης χωρίς μεγάλη επιβάρυνση σε υπολογιστικούς πόρους στο πλαίσιο ενός/μίας εργαστηριακού μαθήματος/έρευνας

Λέξεις Κλειδιά: <<Mininet,Εικονικοποίηση,Openflow,δρομολόγηση,>>

Η σελίδα αυτή είναι σκόπιμα λευκή.

Πίνακας περιεχομένων

1 *ΕΙΣΑΓΩΓΗ*

Στο σύγχρονο κοινωνικό χώρο που χαρακτηρίζεται από ραγδαίες εξελίξεις σε ποικίλους τομείς, σημαντικός καθίσταται και ο ρόλος των δικτύων υπολογιστών στην καθημερινότητα και στον επιχειρησιακό τομέα. Αξίζει να αναλογιστεί κανείς την τεράστια αξία του διαδικτύου στην σημερινή εποχή. Η επικοινωνία έχει γίνει πιο απλή από ποτέ (skype, viber, facebook, κλπ), οι χρηματικές συναλλαγές έχουν επιταχυνθεί (paypal, κλπ), η ανταλλαγή μεγάλου όγκου πληροφορίας μπορεί να πραγματοποιείται πλέον εύκολα και γρήγορα (online cloud storage, κλπ).

Συνεπώς η μελέτη της συμπεριφοράς ενός υπολογιστικού δικτύου αλλά και η κατανόηση των πρωτοκόλλων που διέπουν την επικοινωνία των κόμβων – υπολογιστών που το αποτελούν είναι πολύ σημαντική γιατί προσφέρει γόνιμο έδαφος για βελτιώσεις επί των ήδη υπαρχόντων τεχνολογιών αλλά και βαθύτερη κατανόηση του αντικειμένου.

Ακόμα και στη πανεπιστημιακή εκπαίδευση, στο επίπεδο ενός προπτυχιακού μαθήματος δικτύων υπολογιστών, οι σπουδαστές μέσω της εποπτείας ενός αληθινού δικτύου και τις παρατήρησεις των πακέτων που ανταλλάσσονται αποκτούν μία πιο εμπεριστατωμένη εικόνα των εννοιών που έχουν διδαχθεί.

Πολλές φορές βεβαίως η παρατήρηση ενός αληθινού δικτύου in vivo μπορεί να μην είναι δυνατή είτε για λόγους ασφαλείας, που είναι και το πιο σημαντικό, είτε γιατί πολλές φορές η αναζήτηση πρόσθετης πληροφορίας για διαχειριστικούς σκοπούς και μόνο δημιουργεί επιπρόσθετο φορτίο κίνησης (SNMP). Επομένως κρίνεται αναγκαία η δημιουργία – ανεύρεση μεθόδων για την δημιουργία υποδομών που θα έχουν την ίδια συμπεριφορά με ένα αληθινό δίκτυο αλλά θα μπορούν να μεταβληθούν εύκολα και με μικρό κόστος.

Παρακάτω θα ασχοληθούμε με μία τεχνική που μας επιτρέπει να επιτύχουμε τα παραπάνω και λέγεται εικονικοποίηση δικτύου.

1.1 Εικονικοποίηση Δικτύου

Η εικονικοποιήση δικτύου είναι η αφαιρετική προσέγγιση ενός πραγματικού δικτύου και επιτρέπει την υποστήριξη πολλαπλών λογικών δικτύων πάνω στο ίδιο υλικό. Με άλλα λόγια μπορούμε να έχουμε δρομολογητές, μεταγωγείς και υπολογιστές να τρέχουν πάνω στην ίδια υποδομή. Για να γίνει αυτό πρέπει πρώτα να εικονικοποιηθούν οι κόμβοι του δικτύου. Μια από τις τεχνικές που χρησιμοποιούνται είναι η χρήση εικονικών μηχανημάτων και δεν είναι αρκετά αποδοτική. Μία άλλη τεχνική με την οποία θα ασχοληθούμε εκτενέστερα είναι η ελαφριά εικονικοποιήση είτε με τη χρήση Linux containers, ή FreeBSD jails. Η βασική ιδέα είναι ότι κάθε κόμβος του δικτύου αποκτά ξεχωριστή οπτική για αυτό μέσω της απομόνωσης.

Στη θεωρία των δικτύων υπολογιστών μαθαίνουμε για πακέτα που ταξιδεύουν από υπολογιστή σε υπολογιστή και συμβάλουν από την απλή επιβεβαίωση ότι ένας κόμβος του δικτύου βρίσκεται σε λειτουργία, στο συντονισμό ομάδων κόμβων για τη ταχύτερη αποστολή δυαδικής πληροφορίας στον προορισμό της. Επομένως η παρατήρηση αυτών των πακέτων μπορεί να μας δώσει μία καλύτερη εικόνα για το πως λειτουργούν αυτά τα δίκτυα. Οι πλατφόρμες ανάπτυξης που χρησιμοποιούνται συχνά για αυτό το σκοπό είναι χρήση υλικού για την πλήρη ανοικοδόμηση ενός δικτύου, χρήση προσομοιωτή, χρήση εξομοιωτή. Θα αναλύσουμε την καθεμιά παρακάτω παραθέτοντας μειονεκτήματα και πλεονεκτήματα.

Δημιουργία αληθινού δικτύου. Με τη χρήση αληθινών μηχανημάτων και καλωδίων για τις συνδέσεις δημιουργούμε ένα δίκτυο υπολογιστών και αφού αρχικοποιήσουμε τις παραμέτρους του δικτύου, μετά εγκαθιστώντας έπειτα ένα packet sniffer, φερ' ειπείν Wireshark , σε κάθε μηχάνημα μπορούμε να δούμε την κίνηση των πακέτων. Η μέθοδος αυτή είναι απόλυτα ρεαλιστική και γρήγορη. Μπορεί να χρησιμοποιηθεί από πολλούς. Μειονέκτημα σε αυτό είναι ότι κάποιος χρειάζεται να μεταβεί στη τοποθεσία εγκατάστασης των μηχανημάτων για να μπορέσει να ωφεληθεί από αυτά. Δεν είναι όμως φθηνή καθώς μπορεί να πρέπει να αγοραστεί ο απαραίτητος εξοπλισμός(υπολογιστές, μεταγωγείς, δρομολογητές) αν δεν είναι διαθέσιμος καθώς και να ακολουθήσει συντήρηση αυτού. Απόρροια του προηγούμενου είναι η αδυναμία κλιμάκωσης καθώς ο αριθμός των κόμβων σε ένα δίκτυο περιορίζεται σημαντικά από το κόστος. Τέλος η αλλαγή της εκάστοτε τοπολογίας προς μελέτη μπορεί να απαιτεί αναδιάταξη των καλωδίων στη καλύτερη και αλλαγή ενός μεγάλου τμήματος του εξοπλισμού στη χειρότερη περίπτωση.

Χρήση προσομοίωσης σε υπολογιστή. Με τη χρήση μαθηματικών μοντέλων είναι δυνατόν να εξαχθούν αποτελέσματα για τη συμπεριφορά και την απόδοση ενός δικτύου. Η μέθοδός αυτή είναι φθηνή, εύκολη στο να χρησιμοποιηθεί από τρίτους καθώς είναι πρόγραμμα εκτελέσιμο , ταχύτερη μερικές φορές και από τη πραγματικότητα καθώς είναι αρκετά αφαιρετική. Το πρόβλημα είναι πως δεν είναι ελαστική καθώς η αλλαγή της τοπολογίας μελέτης απαιτεί πολλές φορές την αλλαγή του κώδικα, μαθηματικού μοντέλου που χρησιμοποιείται. Επίσης οι εφαρμογές αυτές δεν τρέχουν κώδικα λειτουργικού συστήματος και δεν είναι διαδραστικές που σημαίνει ότι δεν μπορεί ο χρήστης να προκαλέσει αλλαγές σε πραγματικό χρόνο.

Χρήση εξομοιωτή σε υπολογιστή. Δηλαδή συγγραφή προγραμμάτων που δημιουργούν εικονικά αντίγραφα αληθινών μηχανημάτων με παρεμφερή συμπεριφορά και μπορούν να τρέχουν κώδικα λειτουργικού συστήματος. Επίσης τυχόν αλλαγές στο σύστημα μπορούν υλοποιηθούν με επέκταση του υπάρχοντος κώδικα. Επιπρόσθετα το εικονικό υλικό που δημιουργούν κοστίζει ελάχιστα καθώς μοναδικός μετρητής κόστους είναι το ποσό μνήμης που απαιτείται για την αποθήκευση κάθε μηχανήματος. Επιπλέον μπορούν πολύ εύκολα να πακεταριστούν με όλο τους τον κώδικα, ρυθμίσεις και δεδομένα σε μία εικόνα σκληρού δίσκου που μπορεί να αποθηκευτεί σε κάποια τοποθεσία στο διαδίκτυο και να ληφθεί, τροποποιηθεί, εκτελεστεί από υπολογιστές τρίτων. Τέλος είναι διαδραστική που σημαίνει ότι μπορούμε να επιφέρουμε αλλαγές σε πραγματικό χρόνο, μοναδικό μειονέκτημα ότι είναι πιο αργή σε σχέση με τον αληθινό εξοπλισμό και υστερεί στο συγχρονισμό.

Ανακεφαλαιώνοντας

Πλατφόρμα ανάπτυξης	Πλεονεκτήματα	Μειονεκτήματα
Χρήση υλικού εξοπλισμού	Γρήγορη, Πραγματική Μπορεί να χρησιμοποιηθεί από πολλούς	Ακριβή, δύσκολο να αλλαχθεί, αδύνατο να ληφθεί
Χρήση προσομοίωσης	Φτηνή, εύκολη να ληφθεί, λεπτομερής, Ταχύτερη από τη πραγματικότητα	Απαιτεί αλλαγή κώδικα κατά την τροποποίηση, λεπτομέρεια δεν συνεπάγεται ακρίβεια, δεν είναι διαδραστική
Χρήση εξομοιωτή	Φτηνή, εύκολη να ληφθεί, επεκτάσιμη, Διαδραστική	Αργή σε σχέση με το υλικό, χάνει σε συγχρονισμό

Καταλήγουμε λοιπόν στο συμπέρασμα ότι στη περίπτωση μας, που ενδιαφέρει να παρατηρήσουμε τα πακέτα που ανταλλάσσονται μέσα στο δίκτυο και όχι τόσο να μετρήσουμε τις παραμέτρους του και να κάνουμε γραφικές παραστάσεις η χρήση προγράμματος εξομοίωσης είναι η καλύτερη λύση.

1.2 Αντικείμενο της Διπλωματικής

Με γνώμονα πλέον τη χρήση εξομοίωσης προκύπτει ο εξής προβληματισμός. Βρισκόμαστε στο γραφείο και αναζητούμε μία μέθοδο για τη δημιουργία δικτύων υπολογιστών για εκπαιδευτικούς-ερευνητικούς σκοπούς χωρίς να χρησιμοποιήσουμε αληθινά μηχανήματα. Οι πόροι που διαθέτουμε περιορίζονται στους πόρους του φορητού μας υπολογιστή.

Αναζητούμε λοιπόν ένα πρόγραμμα-μέθοδο εξομοίωσης που θα μας παρέχει τα παρακάτω χαρακτηριστικά.

- Ελαστικότητα: Νέες τοπολογίες και λειτουργικότητες μπορούν να οριστούν μέσω κατάλληλου κώδικα ή παρεμφερούς λειτουργικού συστήματος.
- Αναπτυξιμότητα: Προσθήκη νέων λειτουργιών δεν θα απαιτεί αλλαγή της ήδη υπάρχουσας δομής ή κώδικα.
- Διαδραστικότητα: Η διαχείριση του δικτύου πρέπει να μπορεί να γίνεται σε πραγματικό χρόνο.
- Κλιμακωσιμότητα: Το περιβάλλον πρέπει να μπορεί να δημιουργεί δίκτυα εκατοντάδων κόμβων.
- Ρεαλιστικότητα: Η συμπεριφορά του δικτύου πρέπει να είναι ίδια με τη συμπεριφορά ενός πραγματικού αντίστοιχου δικτύου.
- Κοινοχρησία: Πρωτότυπες δομές που έχουμε φτιάξει να μπορούν να εκτελεστούν αλλά και τροποποιηθούν από τρίτους(συνεργάτες, ερευνητές, κλπ)
- Ελευθερία πρόσβασης – opensource: Να μπορεί να χρησιμοποιηθεί από όλους ανεξαρτήτως οικονομικής δυνατότητας.
- Εκπαιδευτική αξία: Να υπάρχει επαφή με κάποιο είδος κώδικα για το σπουδαστή-ερευνητή και εποπτεία διδαγμένης θεωρίας [1].

Με βάση τα προηγούμενα κριτήρια εξετάσαμε τρεις προσεγγίσεις:

<u>Εξομοίωση πλήρους συστήματος</u>

Σαν μια πρώτη ιδέα η χρήση ενός δικτύου που θα αποτελείται από εικονικά μηχανήματα ακούγεται ιδιαίτερα ελκυστική. Με τη χρήση ενός εικονικού μηχανήματος για κάθε υπολογιστή, μεταγωγέα, δρομολογητή μπορούμε εύκολα και γρήγορα να χτίσουμε μία τοπολογία ενώνοντας τα παραπάνω δομικά στοιχεία με εικονικές διεπαφές. Δοκιμάζοντας αυτή τη μέθοδο βλέπουμε ότι κάθε εικονικό μηχάνημα δεσμεύει αρκετή μνήμη επομένως η μεταβλητότητα της δομής μας είναι πολύ περιορισμένη λόγω περιορισμού στους αποθηκευτικούς μας πόρους.

<u>Εξομοίωση από απόσταση</u>

Μία δεύτερη ιδέα είναι να συμμετέχουμε σε κάποιο διεθνές πρόγραμμα(GENI) που έχει δημιουργήσει διαδικτυακές πλατφόρμες για αυτό το σκοπό που υποστηρίζουν δημιουργία μεγάλων εικονικών δικτύων. Το πρόβλημα εδώ είναι ότι δεν μπορούμε να έχουμε αναπτυξιμότητα καθώς τα μηχανήματα που μας παρέχονται καθορίζονται από το φορέα που διαχειρίζεται την όλη προσπάθεια και επίσης η ελευθερία πρόσβασης και τροποποίησης είναι περιορισμένη. Τέλος καθώς η ζήτηση τέτοιων πόρων είναι μεγάλη ο χρόνος πρόσβασης στις δομές αυτές είναι μικρός.

Container based εξομοίωση

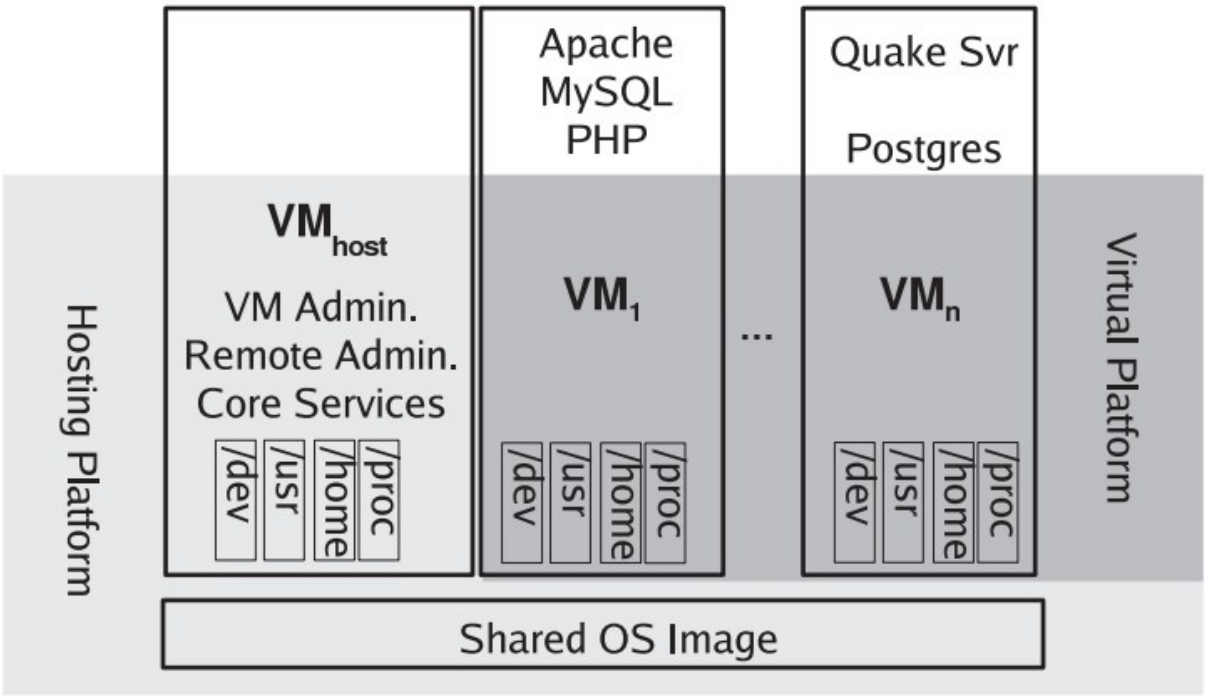
Η τελευταία μας επιλογή είναι η χρήση κατάλληλων προγραμμάτων που συγκεντρώνουν τα χαρακτηριστικά που έχουμε ήδη αναφέρει. Τα επικρατέστερα είναι το GNS3, CORE και το mininet. Θα αναφερθούμε εκτενέστερα στα δύο τελευταία καθώς το πρώτο είναι μεν ανοιχτό λογισμικό απαιτεί δε την χρήση εικόνων δρομολογητών και μεταγωγέων οι οποίες διατίθενται από την εταιρεία CISCO έναντι χρημάτων. Πριν περάσουμε σε μία συνοπτική περιγραφή των προγραμμάτων θα κάνουμε μια συνοπτική αναφορά στο πως δομείται ένα container based σύστημα.

2 *ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ*

Στο προηγούμενο κεφάλαιο ασχοληθήκαμε με την εικονικοποίηση δικτύου και δώσαμε μια σύντομη περιγραφή της. Αναφερθήκαμε στις μεθόδους δημιουργίας δικτύων υπολογιστών παραθέτοντας τα θετικά, αρνητικά στοιχεία της καθεμιάς. Στο παρόν κεφάλαιο θα ασχοληθούμε με την αρχιτεκτονική πάνω στην οποία πρέπει να είναι δομημένο ένα σύστημα για να υποστηρίζει εικονικοποίηση. Επίσης θα κάνουμε μία μικρή επισκόπηση προγραμμάτων τα οποία υλοποιούν την εικονικοποίηση πάνω σε ίδιες η παρεμφερείς αρχιτεκτονικές. Τέλος θα αναφερθούμε εκτενώς σε ένα από αυτά και θα περιγράψουμε αναλυτικά την εικονικοποίηση σε επίπεδο εντολών λειτουργικού συστήματος

2.1 Αρχιτεκτονική Συστήματος

Ένα container based σύστημα περιέχει μία διαμοιρασμένη εικόνα λειτουργικού που αποτελείται από ένα διαχειριστικό σύστημα αρχείων, και ένα σύνολο κοινόχρηστων βιβλιοθηκών και εκτελέσιμων αρχείων συστήματος .Κάθε εικονικό μηχάνημα μπορεί να εκκινηθεί, τερματιστεί σαν ένας φυσιολογικός υπολογιστής .Πόροι όπως χωρητικότητα δίσκου, υπολογιστική ισχύ, μνήμη εκχωρούνται σε κάθε μηχάνημα όταν δημιουργείται, και μπορούν να τροποποιηθούν σε πραγματικό χρόνο. Υπό τη σκοπιά άλλων εφαρμογών και του χρήστη οι εικόνες αυτές φαίνονται σαν ξεχωριστά μηχανήματα. Στην εικόνα έχουμε μία γραφική αναπαράσταση της παραπάνω περιγραφής

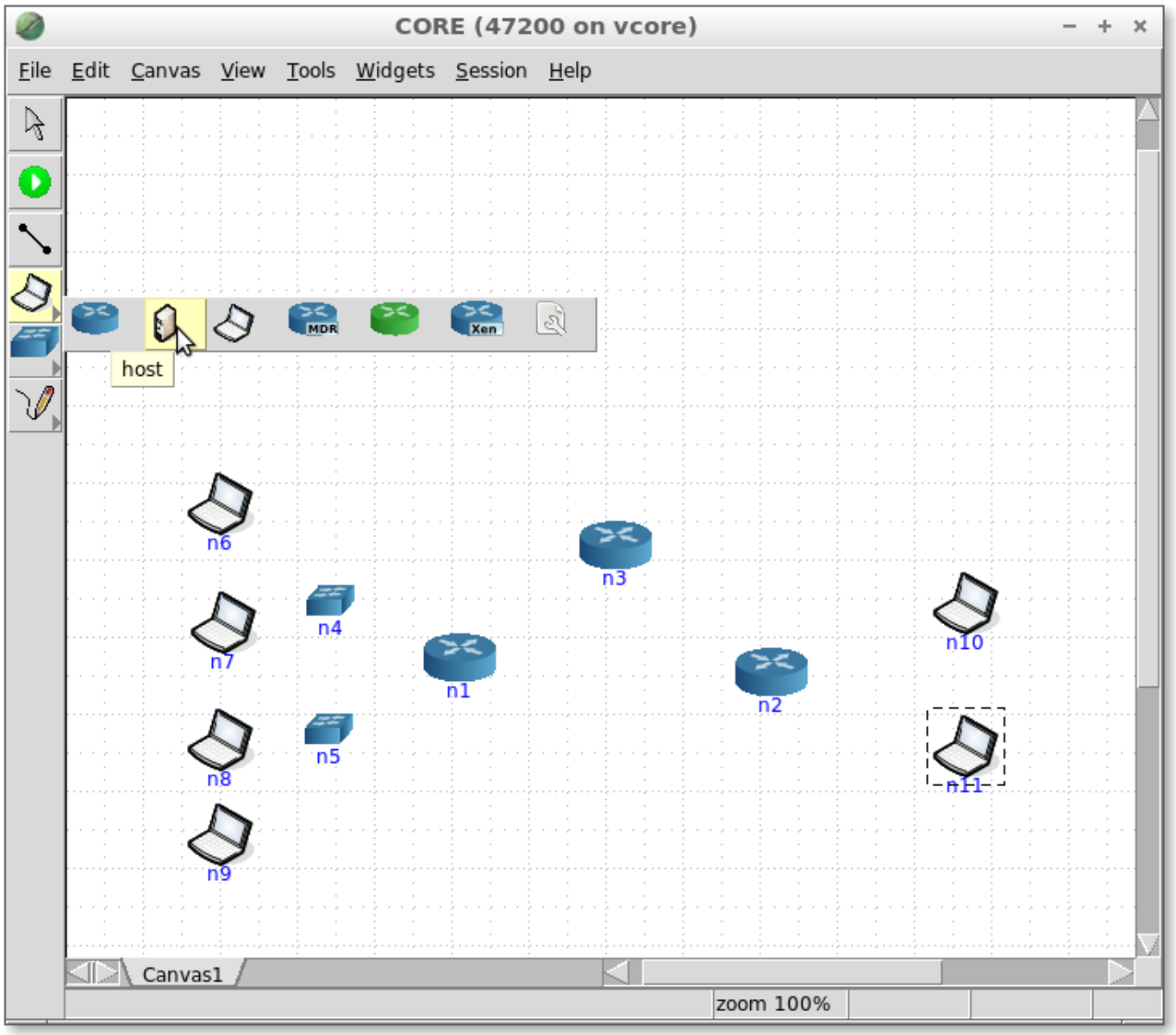


Έχουμε τρεις ομαδοποιήσεις .Το φιλοξενούν μηχάνημα αποτελείται από την κοινόχρηστη εικόνα του λειτουργικού και την εικόνα του διαχειριστή που ουσιαστικά ελέγχει τα υπόλοιπα εικονικά μηχανήματα. Το σημαντικότερο χαρακτηριστικό που προσφέρει η δομή αυτή είναι η απομόνωση. Διακρίνουμε την απομόνωση σε 3 κατηγορίες:
Απομόνωση βλάβης: Κάθε βλάβη εικονικού μηχανήματος παραμένει περιορισμένη στο δικό της τόπο ονομάτων και δεν μπορεί να επηρεάσει και τα υπόλοιπα μηχανήματα.
Απομόνωση πόρων: Αποφυγή παρεμβολής και ανεπιθύμητων αλληλεπιδράσεων μεταξύ των μηχανημάτων. Στους πόρους συγκαταλέγονται ο επεξεργαστής, μνήμη, εύρος ζώνης δικτύου
Απομόνωση ασφαλείας: Κάθε εικονικό μηχάνημα διατηρεί δικές τους ρυθμίσεις και ονοματοδοσία ανεξάρτητα από τα υπόλοιπα μηχανήματα.[2]

2.2 Προγράμματα που ακολουθούν την ίδια η παρεμφερή αρχιτεκτονική

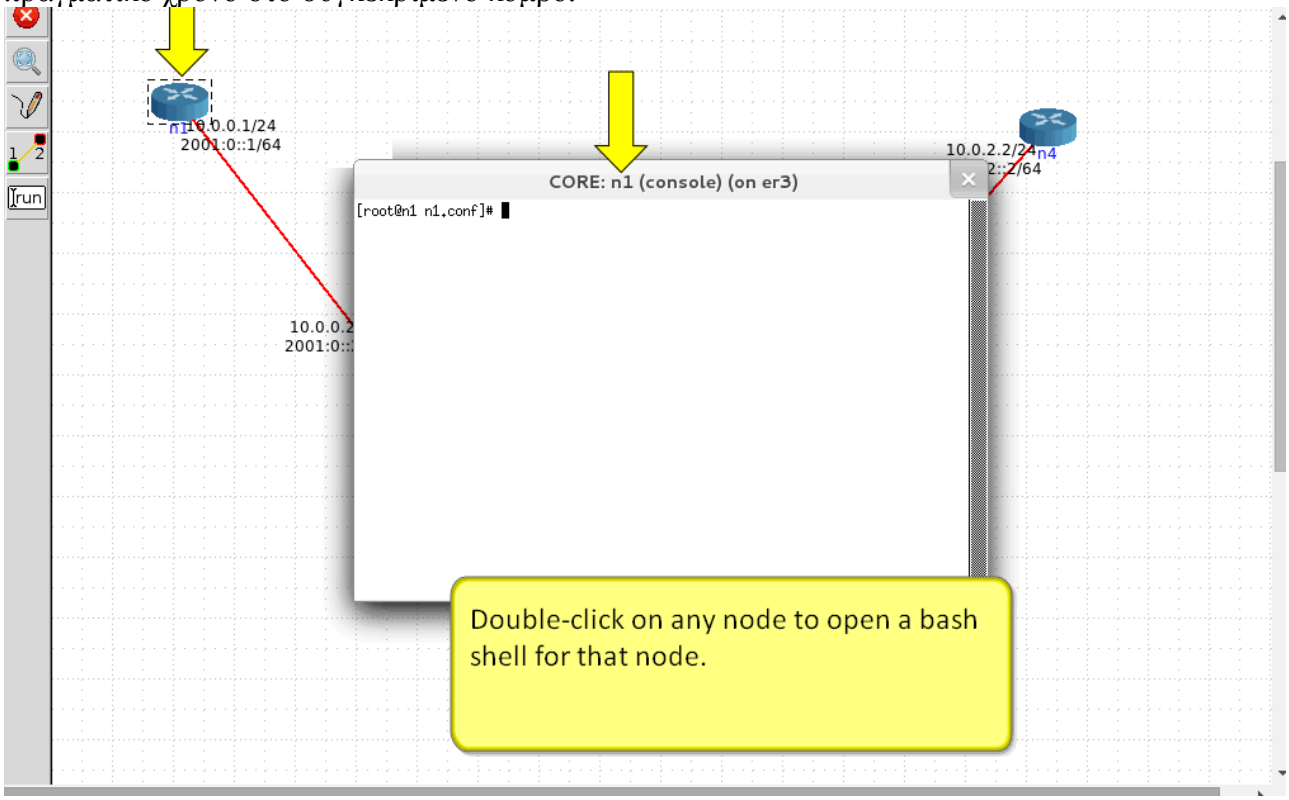
2.2.1 CORE

Το CORE είναι ένα πρόγραμμα εξομοίωσης δικτύων. Διαθέτει ένα πλούσιο γραφικό περιβάλλον πάνω στο οποίο υπάρχουν γραμμές εργαλείων, στις οποίες ο χρήστης μπορεί να βρεί έτοιμα τα



κύρια συστατικά ενός δικτύου(μεταγωγείς, δρομολογητές, υπολογιστές) τα οποία μπορεί να τοποθετήσει με το ποντίκι στον καμβά σχεδίασης και να τα συνδέσει μεταξύ τους. Επιπρόσθετες ιδιότητες όπως φερ' ειπείν κάποιο κατώφλι εύρους ζώνης, καθυστέρηση, απώλειες ανά ζεύξη μπορούν να καθοριστούν δυναμικά από το χρήστη.

Η τοπολογία μπορεί μπορεί να αρχικοποιηθεί και ενώ εκτελείται. Επιλέγοντας ένα κόμβο του δικτύου ανοίγει ένα παράθυρο στην οθόνη σε κέλυφος Unix για να δοθούν εντολές σε πραγματικό χρόνο στο συγκεκριμένο κόμβο.



Για να δημιουργήσει κάθε ξεχωριστό εικονικό μηχάνημα το CORE εκχωρεί σε κάθε στοίβα το δικό της χώρο αποθήκευσης με τη χρήση της εντολής jail του λειτουργικού συστήματος FreeBSD. Σε αντίθεση με τα παραδοσιακά εικονικά μηχανήματα, οι εικόνες αυτές δεν έχουν ένα ολοκληρωμένο λειτουργικό σύστημα που τρέχει πάνω στο φιλοξενούν μηχάνημα. Αντίθετα τρέχουν πάνω στον ίδιο πυρήνα και μοιράζονται το ίδιο σύστημα αρχείων, επεξεργαστή , μνήμη, ρολόι και άλλους πόρους. Με αυτό το τρόπο καταφέρνει να δημιουργεί τοπολογίες με εκατοντάδες εικονικά μηχανήματα. Δεν θα επεκταθούμε παρακάτω γιατί όλα τα υπόλοιπα χαρακτηριστικά του CORE εμπεριέχονται και στο mininet.

Κλείνουμε λέγοντας ότι το CORE διαθέτει μία άψογη πλατφόρμα εξομοίωσης που όμως δεν δίνεται καθόλου ευκαιρία στον σπουδαστή-ερευνητή να έρθει σε επαφή με κάποιο είδος κώδικα αφού όλα είναι απλοποιημένα στο επίπεδο μίας γραφικής εφαρμογής. Επίσης δεν υποστηρίζει καθόλου το πρωτόκολλο openflow στο οποίο θα αναφερθούμε παρακάτω εκτενώς [3].

2.2.2 Mininet

Το mininet, που αποτελεί τελικά και το πρόγραμμα με το οποίο θα ασχοληθούμε στη συνέχεια, είναι ένα τοπικό περιβάλλον εξομοίωσης που είναι γραμμένο στη γλώσσα Python και τρέχει σε περιβάλλον Linux. Το mininet αποτελεί ένα σύστημα ταχείας προτυποποίησης μεγάλων δικτύων υπολογιστών στους περιορισμένους πόρους ενός λάπτοπ. Χρησιμοποιώντας τα χαρακτηριστικά της ελαφριάς εικονικοποίησης σε επίπεδο λειτουργικού συστήματος, συμπεριλαμβανομένων διεργασιών και δικτυακών τόπων ονομάτων μπορεί να δημιουργεί εκατοντάδες κόμβους. Με τη χρήση της γλώσσας Python ο χρήστης μπορεί να ορίσει νέες δομές μέσω κληρονομικότητας και πολύπλοκες τοπολογίες μέσω απλών εντολών. Το mininet μπορεί να τρέξει σε ένα απλό λάπτοπ, κατανέμοντας διεργασίες και εικονικά ζεύγη ethernet μέσα σε δικτυακούς τόπους ονομάτων, και να δημιουργήσει δίκτυα μεγάλου εύρους ζώνης και εκατοντάδων κόμβων. Τέλος ολόκληρο το δίκτυο μπορεί να πακεταριστεί σε ένα εικονικό μηχάνημα έτσι ώστε να μπορεί να εκτελεστεί, τροποποιηθεί από τρίτους.

Το mininet μπορεί να δημιουργεί μεγάλες δικτυακές τοπολογίες εκμεταλλευόμενο την τεχνική της ελαφριάς εικονικοποίησης(light weight virtualization) .Ως αποτέλεσμα μεγάλες τοπολογίες, που η δημιουργία τους μέσω φόρτωσης εικονικών μηχανημάτων σε έναν υπολογιστή καθίσταται πρακτικά αδύνατη λόγω της περιορισμένης μνήμης μπορούν πολύ εύκολα να δημιουργηθούν από αυτό. Πρακτικά το mininet τα καταφέρνει παράγοντας εικονικά μηχανήματα που μοιράζονται ένα μεγάλο κομμάτι του αρχικού συστήματος όπως το σύστημα αρχείων χρήστη, τις κοινές βιβλιοθήκες και άλλα κομμάτια κοινού κώδικα. Κάθε μηχάνημα επομένως μπορεί να περιοριστεί σε μία απλή διεργασία κελύφους μέσα στο δικό της δικτυακό τόπο ονομάτων(διεύθυνση IP, πίνακας δρομολόγησης, , κλπ) .Με αυτό το τρόπο ο αποθηκευτικός χώρος κάθε εικονικού μηχανήματος περιορίζεται σε 1 MB σε αντίθεση με τα 70MB που θα καταλάμβανε στην πλήρη του μορφή.

Επιπρόσθετα το mininet αποτελεί ένα πολύ καλό εργαλείο ταχείας προτυποποίησης καθώς κάθε καινοτόμος ιδέα στο χώρο των δικτύων υπολογιστών μπορεί να δοκιμασθεί εύκολα και γρήγορα με δεδομένο ότι οι χρόνοι αρχικοποίησης είναι μικροί ακόμα και για μεγάλες τοπολογίες καθώς οι πόροι που απαιτούνται είναι λίγα MB αποθηκευτικού χώρου. Επιπλέον κάθε πρωτότυπο μπορεί να πακεταριστεί σε ένα εικονικό μηχάνημα και να διανεμηθεί ελεύθερα σε άλλους ερευνητές, σπουδαστές, κλπ.

Το γεγονός φυσικά ότι όλα τα μηχανήματα μοιράζονται το ίδιο σύστημα αρχείων, μας δημιουργεί πρόβλημα όταν θα χρειαστεί να τρέξουμε σε κάθε μηχάνημα έναν δαίμονα που απαιτεί ξεχωριστό αρχείο παραμετροποίησης. Το mininet υποστηρίζει επίσης όλες τις λειτουργίες του πρωτοκόλλου openflow καθώς οι μεταγωγείς που δημιουργεί μπορούν να εκτελέσουν όλες τις απαραίτητες ενέργεις που το πρωτόκολλο απαιτεί.

Παρακάτω θα δώσουμε μία συνοπτική περιγραφή του πρωτοκόλλου openflow και θα περιγράψουμε πως αυτό ενσωματώνεται στο mininet

2.2.2.1 Εισαγωγή στο πρωτόκολλο openflow

Η βασική ιδέα που οδήγησε στο πρωτόκολλο Openflow είναι πολύ απλή. Οι περισσότεροι σύγχρονοι μεταγωγείς και δρομολογητές διαθέτουν πίνακες ροής που χρησιμοποιούνται για τη δρομολόγηση, προώθηση πακέτων, συλλογή στατιστικών δεδομένων, κλπ. Ενώ οι περισσότεροι από αυτούς τους πίνακες είναι διαφορετικοί όλοι εμφανίζουν ένα κοινό σύνολο από λειτουργίες. Το εν λόγω πρωτόκολλο εκμεταλλεύεται αυτό το χαρακτηριστικό.

Συγκεκριμένα το openflow προσφέρει τη δυνατότητα προγραμματισμού των πινάκων ροής διαφορετικών μεταξύ τους δρομολογητών και μεταγωγέων. Για παράδειγμα ο διαχειριστής ενός δικτύου μπορεί να φροντίσει η κίνηση των πακέτων που προέρχονται από το δικό του τερματικό να ακολουθεί διαφορετικό δρομολόγιο και να χαίρει διαφορετικών υπηρεσιών επεξεργασίας από όλα τα υπόλοιπα πακέτα του δικτύου χωρίς να επηρεάζεται καθόλου η λειτουργία όλου του δικτύου το οποίο συνεχίζει να λειτουργεί όπως λειτουργούσε ανεπηρέαστο. Με αυτό το τρόπο μπορεί φερ' ειπείν ένας ερευνητής να δοκιμάσει διαφορετικά πρωτόκολλα δρομολόγησης, μοντέλα ασφαλείας ακόμα και εναλλακτικά πρωτόκολλα στο IP χωρίς να επηρεάζει καθόλου το δίκτυο που τον φιλοξενεί.

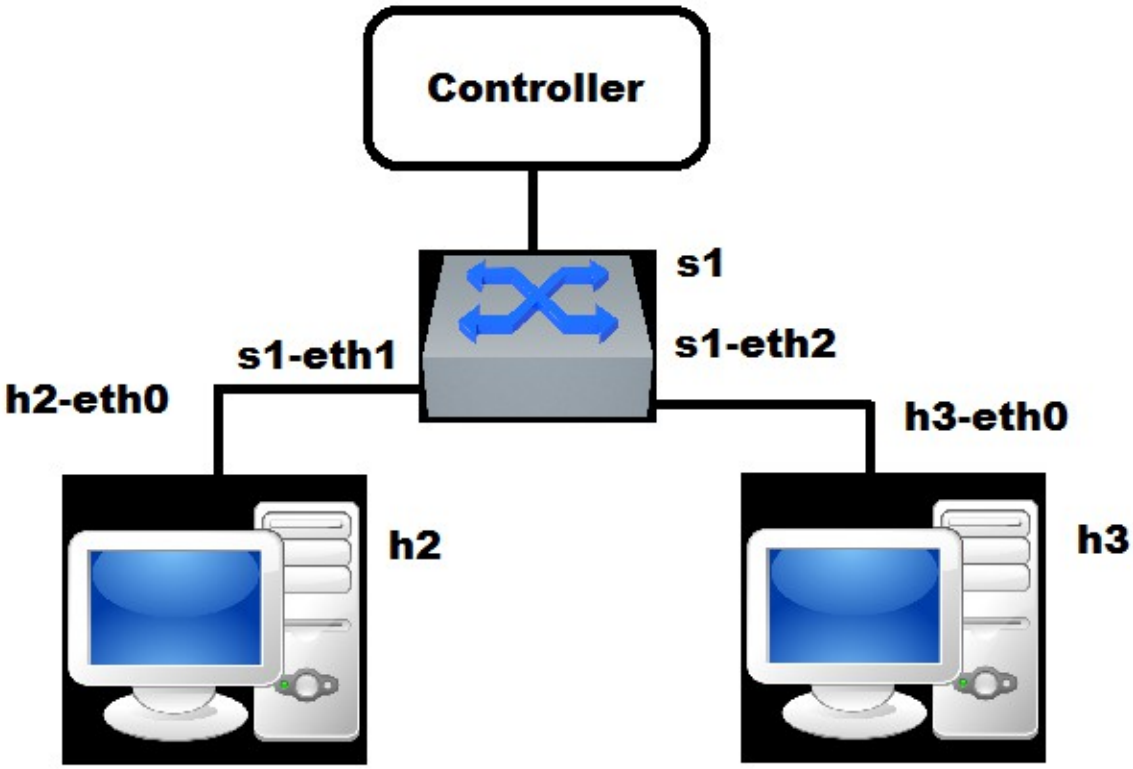
Για το υπόλοιπο της περιγραφής θα αναφερόμαστε σε μεταγωγείς και δρομολογητές ως ένα κοινό στοιχείο με το όνομα μεταγωγέας openflow καθώς παρουσία του συγκεκριμένου πρωτοκόλλου, αυτά τα δύο στοιχεία ισοδυναμούν. Ένας μεταγωγέας openflow αποτελείται από ένα πίνακα ροής και μια ενέργεια για κάθε καταχώρηση, η οποία πληροφορεί το μεταγωγέα πως θα επεξεργαστεί ένα πακέτο που ανήκει σε αυτή. Μία ασφαλής σύνδεση του μεταγωγέα με μία απομακρυσμένη διαχειριστική διεργασία την οποία θα αποκαλούμε από εδώ και πέρα controller. Η σύνδεση επιτρέπει επίσης την αποστολή πακέτων, εντολών από τον μεταγωγέα στον controller και αντίστροφα, με τη χρήση του πρωτοκόλλου openflow. Το εν λόγω πρωτόκολλο προσδιορίζει μία συγκεκριμένη διεπαφή πάνω στο μεταγωγέα μέσω τις όποιας εισάγονται οι καταχωρήσεις στον πίνακα ροής από εξωτερικό παράγοντα(controller) , απαλείφοντας την ανάγκη προγραμματισμού του μεταγωγέα από ειδικούς.

Θα επεκταθούμε περισσότερο στις λειτουργίες του πρωτοκόλλου openflow αναφερόμενοι στη συνέχεια στις λειτουργίες του μεταγωγέα openflow του mininet.

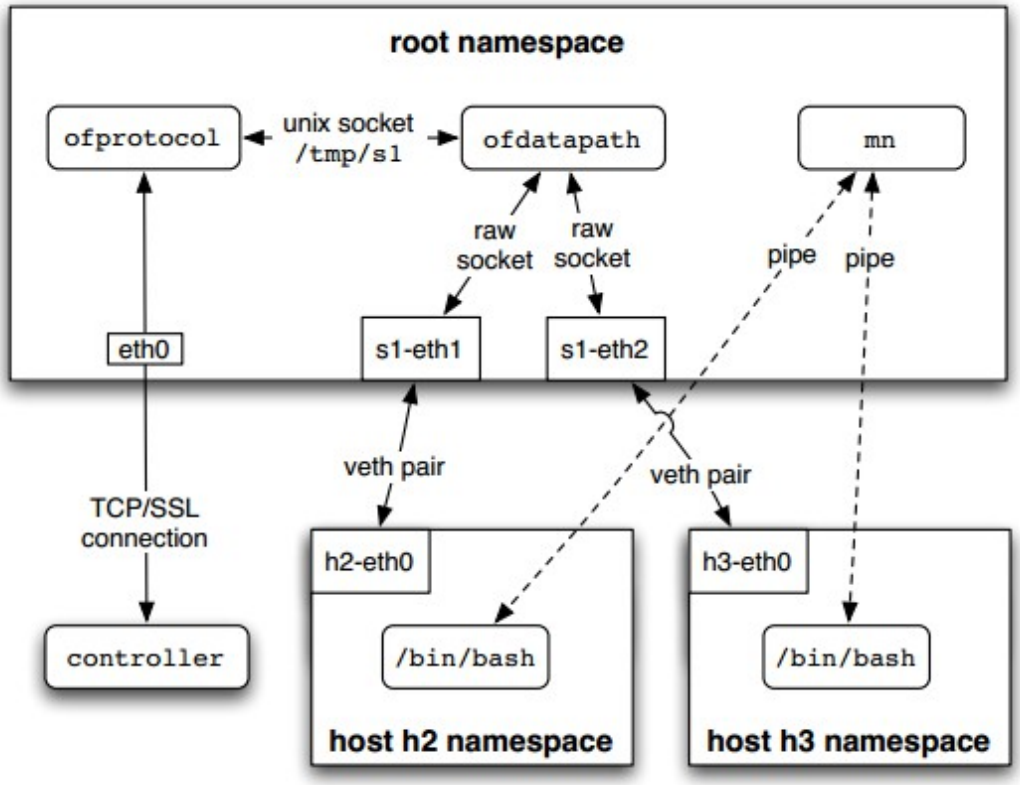
2.2.2.2 Mininet και openflow

Ο μεταγωγέας openflow του mininet είναι ένα εικονικό στοιχείο που προωθεί πακέτα από μία από τις θύρες του σε μία άλλη σύμφωνα με τις υποδείξεις του controller. Κάθε πακέτο κατηγοριοποιείται με βάση την καταχώρηση του πίνακα ροής στην οποία ανήκει. Κάθε καταχώρηση αντιπροσωπεύει μία ροή. Μία ροή μπορεί να είναι όλα τα πακέτα που ανήκουν σε μία σύνδεση TCP, όλα τα πακέτα που αποστέλλονται από μία συγκεκριμένη διεύθυνση MAC ή IP, όλα τα πακέτα που προέρχονται από ένα συγκεκριμένο υποδίκτυο, όλα τα πακέτα που κατευθύνονται προς μία συγκεκριμένη θύρα προορισμού, κλπ. Για πακέτα που δεν ακολουθούν το πρότυπο IPv4 μία ροή μπορεί να οριστεί μέσω συγκεκριμένων κεφαλίδων.
Κάθε καταχώρηση έχει και μία ενέργεια που σχετίζεται με αυτή. Θα αναφέρουμε τις 3 βασικότερες από αυτές.
Προώθηση των πακέτων μίας συγκεκριμένης ροής σε μία από τις θύρες του μεταγωγέα.
Προώθηση ενός πακέτου, που ανήκει σε μία νέα ροή για την οποία δεν υπάρχει καταχώρηση στο πίνακα, στον controller ο οποίος αποφασίζει αν θα προστεθεί μία νέα καταχώρηση στο πίνακα του μεταγωγέα ή θα απορρίψει το πακέτο και θα στείλει και εντολή για μελλοντική απόρριψη παρεμφερών πακέτων. Η ενέργεια αυτή μπορεί να χρησιμοποιηθεί και για προώθηση πακέτων στον controller που πρέπει να υποστούν περαιτέρω επεξεργασία.[1]
Απόρριψη όλων των πακέτων που ανήκουν σε μία συγκεκριμένη ροή. Μπορεί να χρησιμοποιηθεί για θέματα ασφαλείας.
Κάθε καταχώρηση έχει 3 πεδία.
Μία κεφαλίδα πακέτου που ορίζει τη ροή.
Την ενέργεια που θα εκτελείται για πακέτα αυτής της ροής.
Στατιστικά που αφορούν το αριθμό των πακέτων που έχουν περάσει που ανήκουν σε αυτή τη ροή, χρόνο ζωής της συγκεκριμένης καταχώρησης, κλπ.
Παρακάτω θα ασχοληθούμε με το πως το mininet δημιουργεί τερματικά, μεταγωγείς και δρομολογητές μέσω της εικονικοποίησης και θα δείξουμε πως λειτουργεί μέσα από ένα παράδειγμα.

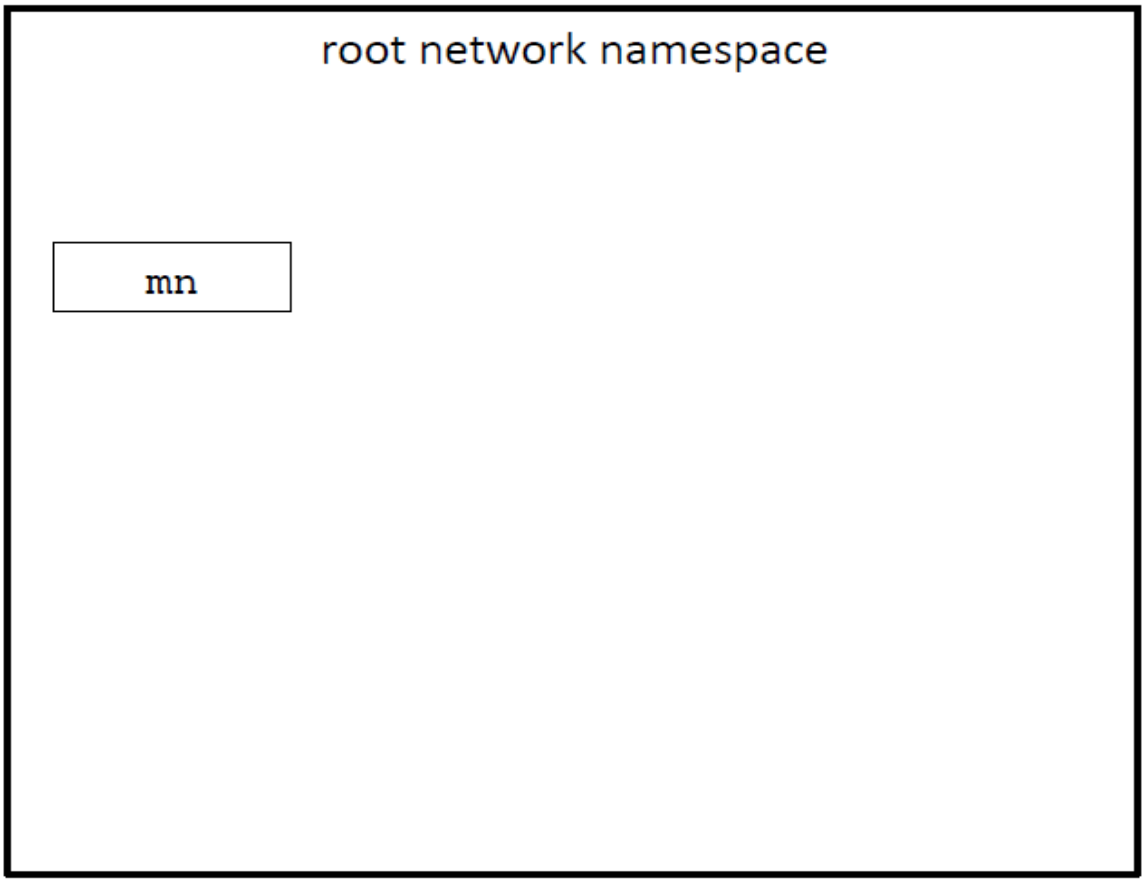
Θα δείξουμε πως λειτουργεί το mininet μέσα από ένα παράδειγμα.
Έστω ότι δημιουργούμε την παρακάτω τοπολογία, το πως θα το δούμε παρακάτω!



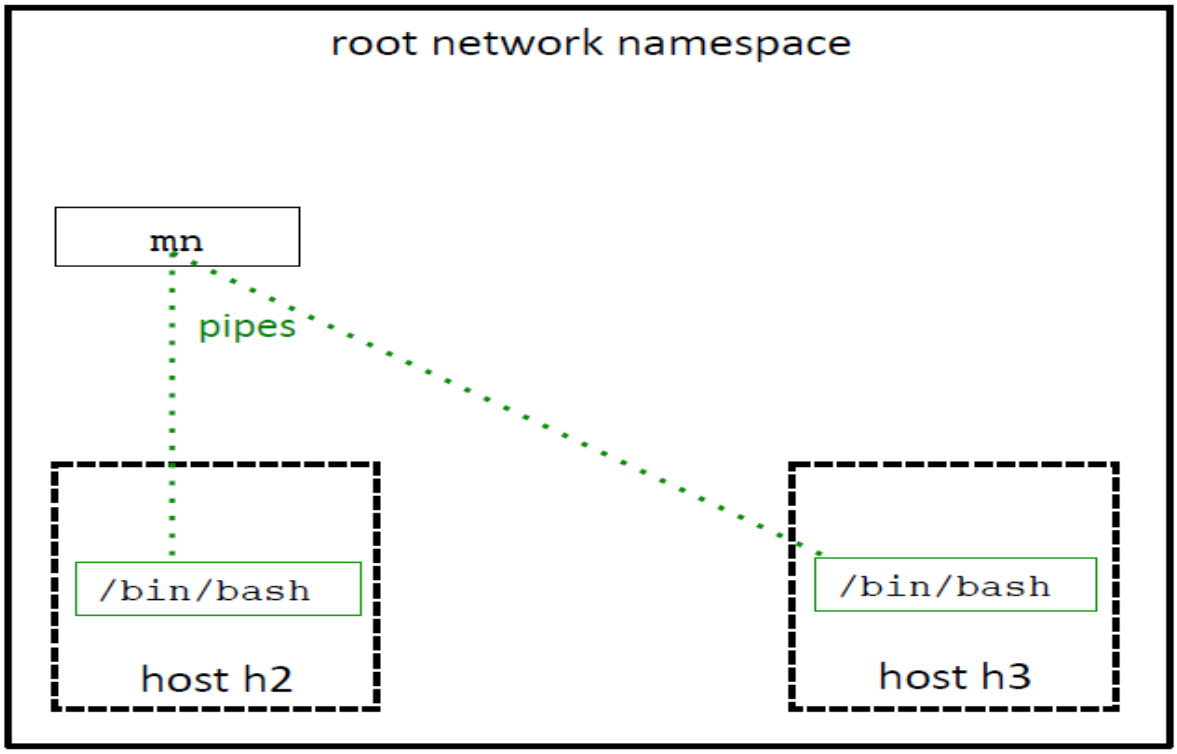
Η αρχιτεκτονική το mininet αναπαρίσταται σχηματικά στο ακόλουθο διάγραμμα



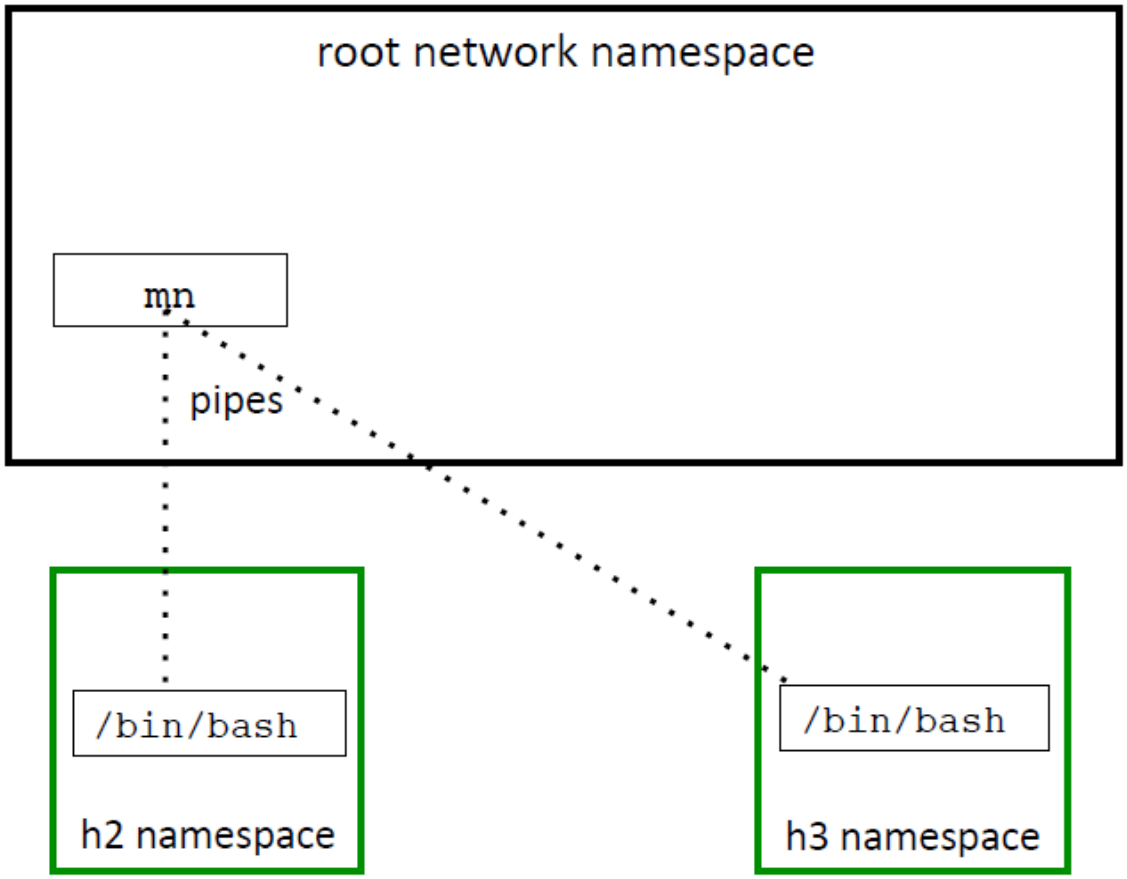
Θα δώσουμε μία σύντομη περιγραφή πως προέκυψε το παραπάνω σχήμα και μετά θα αναφερθούμε πιο αναλυτικά στα επιμέρους υλικά και στο πως λειτουργεί.
Αρχικά τρέχουμε το εκτελέσιμο πρόγραμμα mn μέσα στο διαχειριστικό τόπο ονομάτων



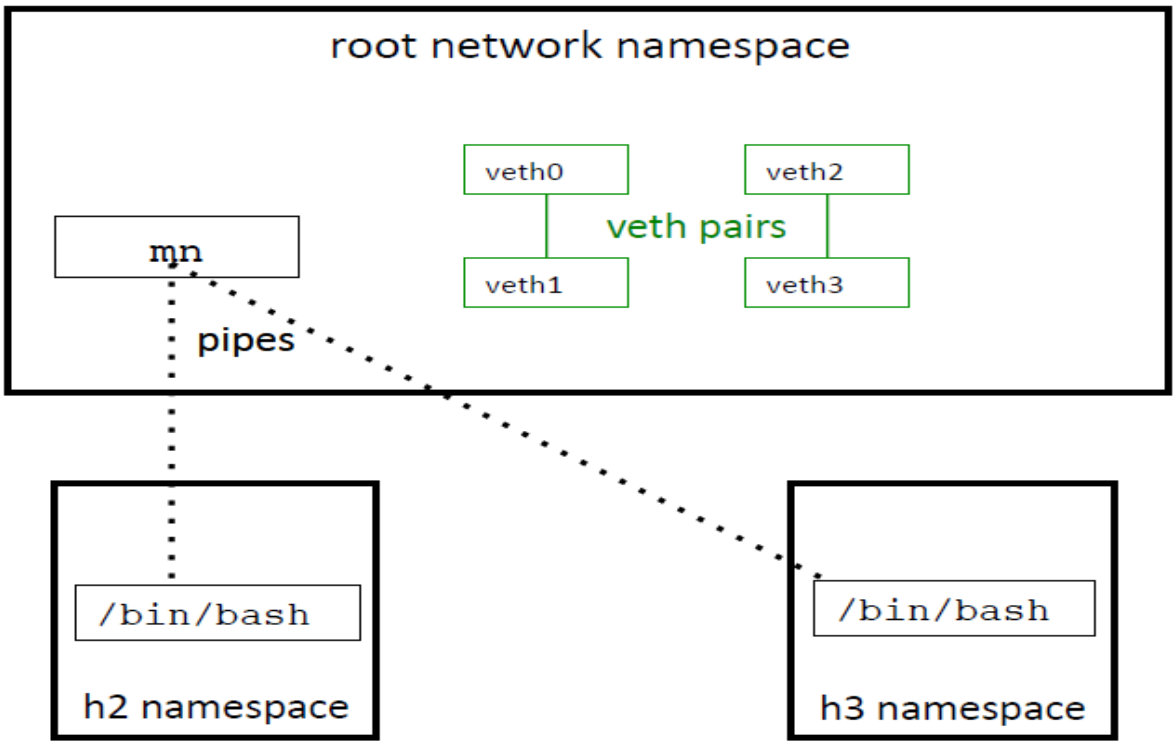
Στη συνέχεια το mn δημιουργεί δύο διεργασίες bash μία για κάθε τερματικό



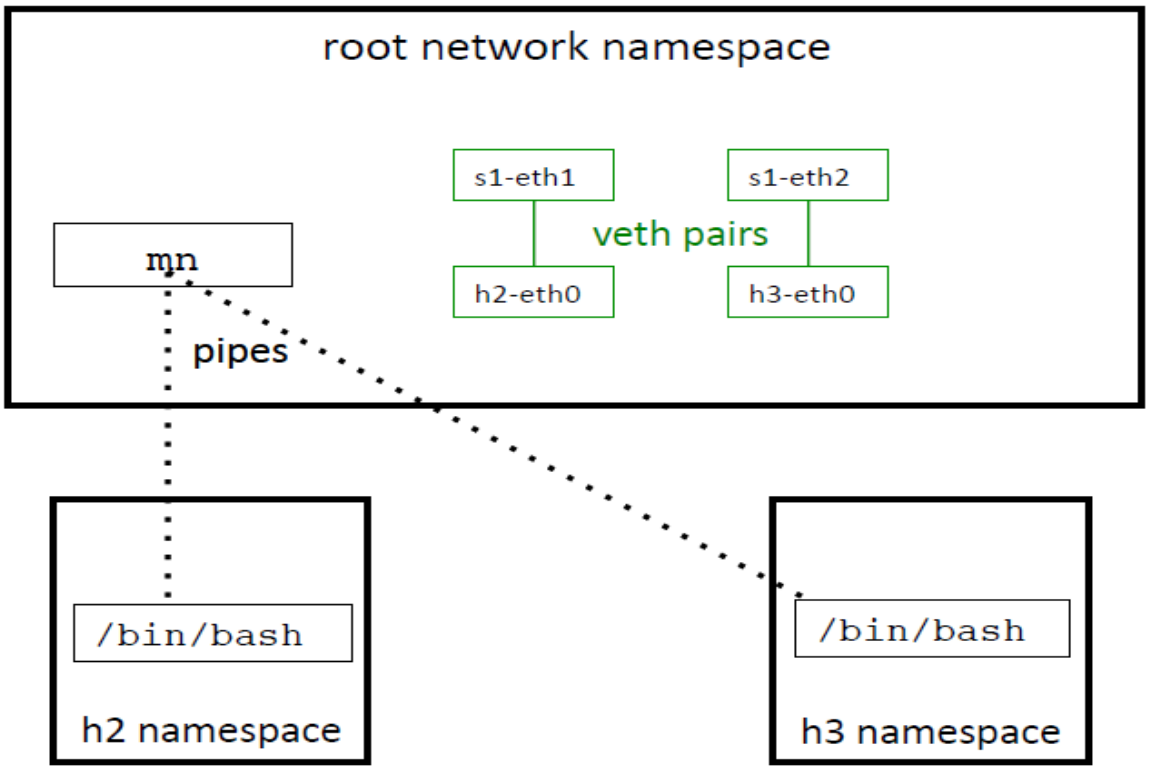
Στη συνέχεια με τη χρήση της εντολής unshare(CLONE_NEWNET) , τοποθετεί τις δύο διεργασίες σε δικούς τους ξεχωριστούς τόπους ονομάτων



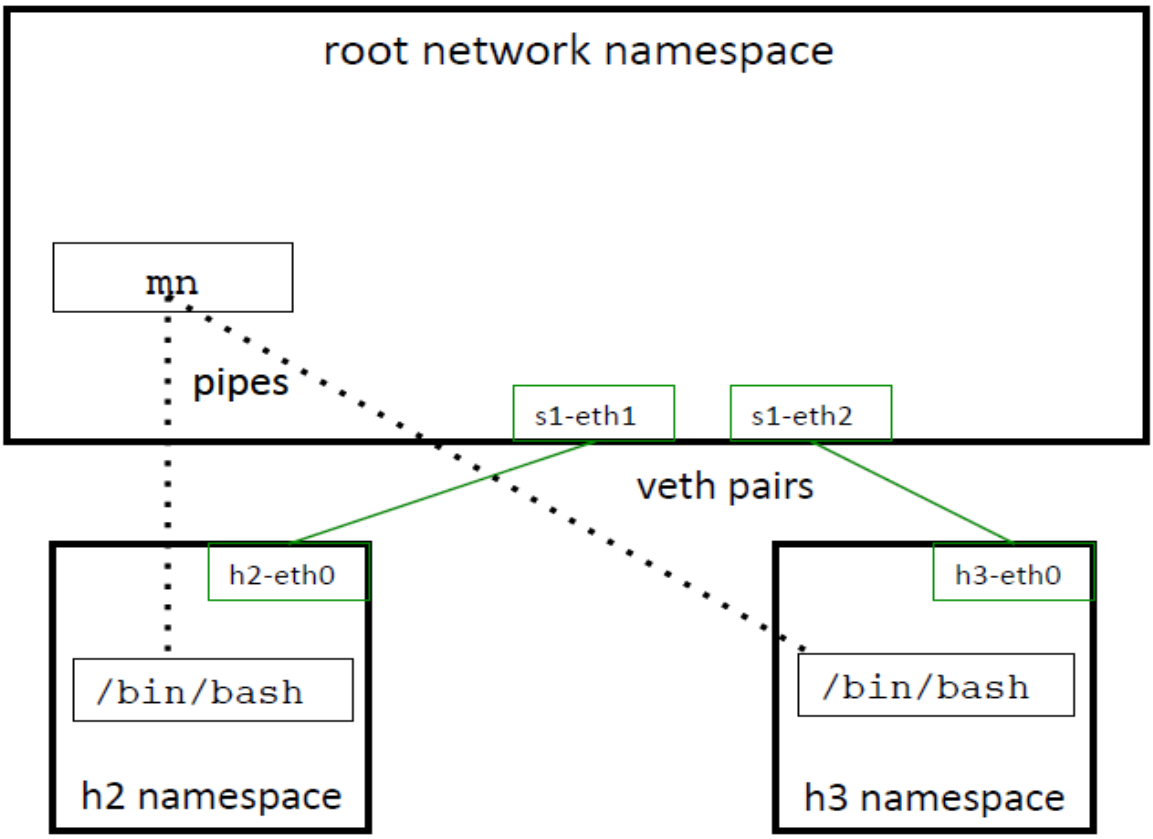
Δημιουργεί δύο ζεύγη εικονικών διεπαφών ethernet με την εντολή ip link add



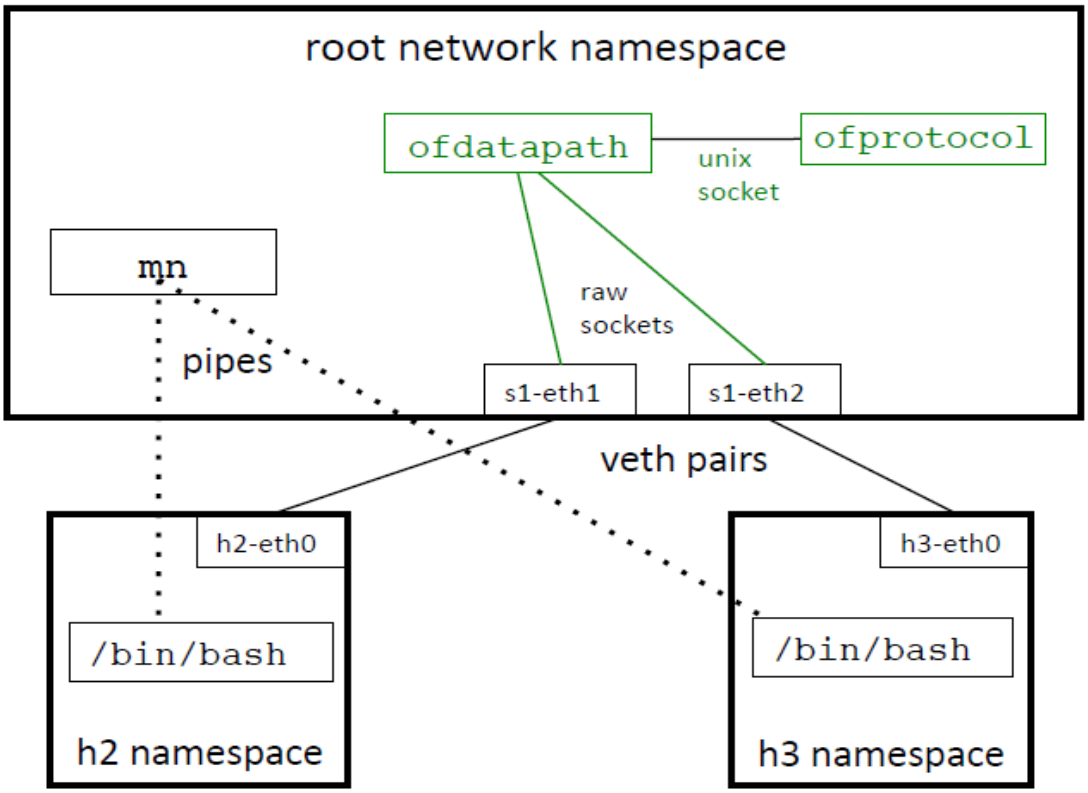
Τους δίνει ονόματα με την εντολή ip link set veth0 name s1-eth1



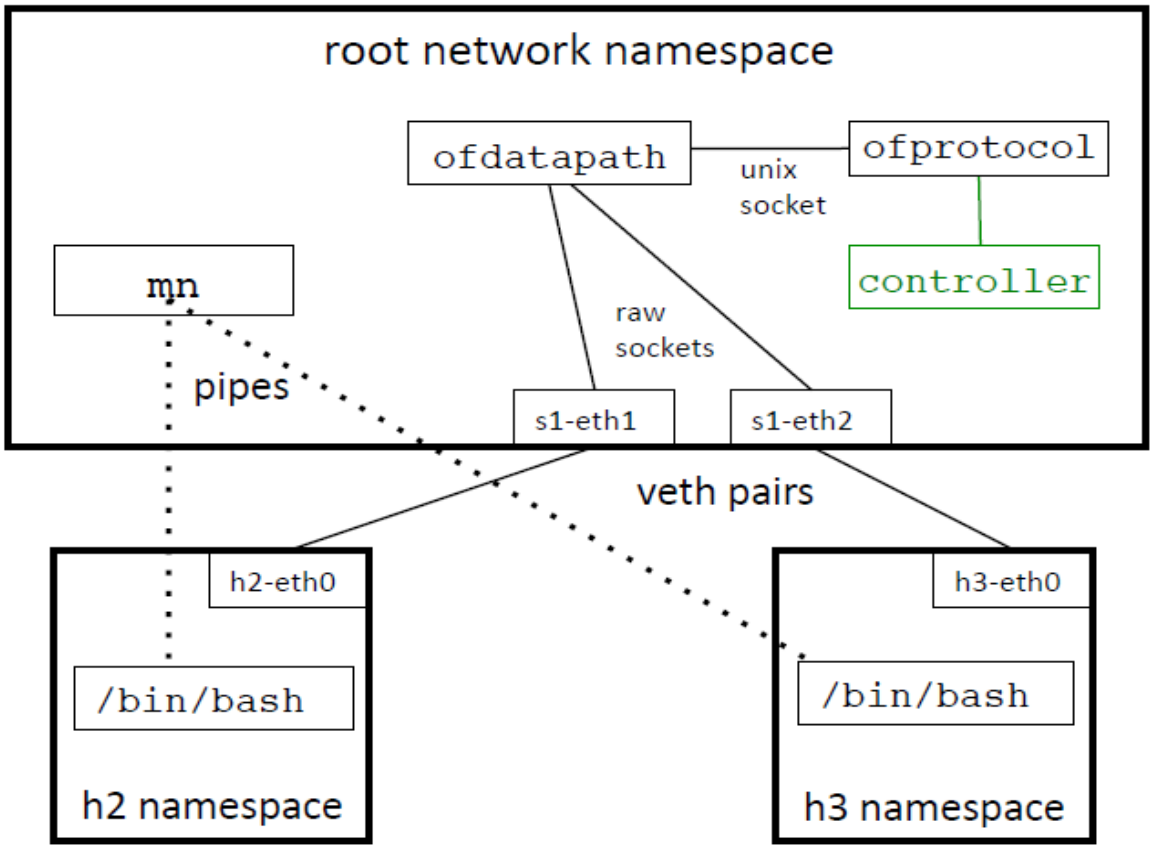
Τοποθετεί το δεύτερο κομμάτι κάθε ζεύγους στο δικό του τόπο ονομάτων με τη χρήση της εντολής ip link set h2-eth0 netns <h2 namespace>



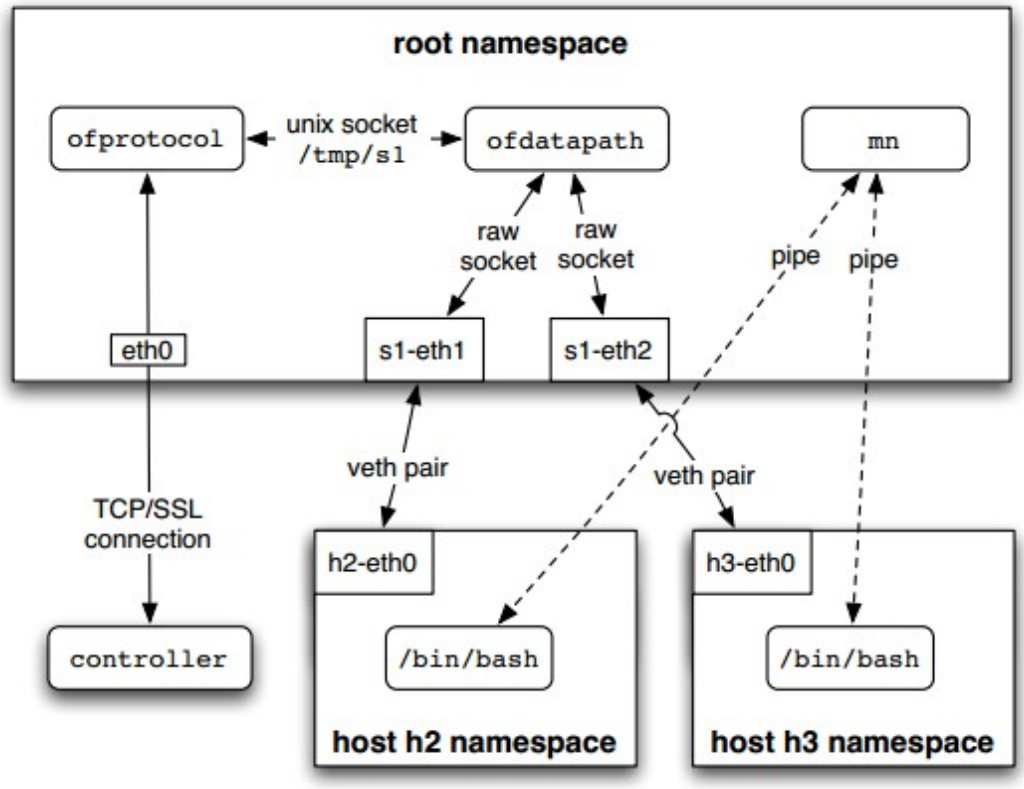
Δημιουργία ενός μεταγωγέα



Δημιουργία ενός controller για τον μεταγωγέα



Άρα το τελικό αποτέλεσμα διαμορφώθηκε όπως και παραπάνω



Veth pair: Ένα εικονικό ζεύγος ethernet (veth: virtual ethernet) συμπεριφέρεται σαν ένα αληθινό καλώδιο που συνδέει δύο εικονικές διεπαφές. Τα πακέτα που στέλνονται μέσα από τη μία διεπαφή παραδίδονται στην άλλη.

Host: Δικτυακοί τόποι ονομάτων που αποθηκεύουν καταστάσεις δικτύου. Παρέχουν διεργασίες και ομάδες διεργασιών με ιδιωτικές διεπαφές, θύρες και πίνακες δρομολόγησης, όπως πίνακες ARP και IP. Για παράδειγμα δύο υπολογιστές μπορούν να συνυπάρχουν στο ίδιο σύστημα σε διαφορετικούς δικτυακούς τόπους ονομάτων και να ακούνε και οι δύο στην δική τους διεπαφή eth0 στη θύρα 80. Στο mininet ένα τερματικό είναι σε όρους λειτουργικών συστημάτων μια διεργασία κελύφους, φερ' ειπείν bash, που μεταφέρεται στο δικό της τόπο ονομάτων με την εντολή συστήματος unshare(CLONE_NEWNET) .Κάθε τερματικό έχει τη δική του εικονική διεπαφή/ές ethernet και έναν αυλό με τη μητρική διεργασία mn που στέλνει εντολές και ελέγχει την έξοδο.

Switch(s1) : Συμπεριφέρεται ακριβώς όπως ένας αληθινός μεταγωγέας. Χρησιμοποιείται για το διαχωρισμό υποδικτύων και βρίσκεται στο διαχειριστικό τόπο ονομάτων (root namespace) .

Controller: Μπορεί να βρίσκεται οπουδήποτε είτε στο εικονικό είτε στο πραγματικό δίκτυο αρκεί το μηχάνημα στο οποία τρέχουν οι μεταγωγείς να έχει σύνδεση επιπέδου IP με τον controller. Στην περίπτωση μας που το mininet τρέχει μέσα σε ένα εικονικό μηχάνημα ο controller μπορεί να βρίσκεται είτε μέσα σε αυτό, είτε στο φιλοξενούν, είτε στο cloud.

Αφού λοιπόν σηκώσουμε το παραπάνω δίκτυο στο mininet μπορούμε να τρέξουμε εντολές σε κάθε τερματικό ξεχωριστά, να προκαλέσουμε βλάβη σε κάποιο καλώδιο. Για το σκοπό αυτό το mininet έχει ένα δικό του CLI (Command Line Interface) που μας επιτρέπει να διαχειριζόμαστε το δίκτυο που μόλις κατασκευάσαμε. Ένα πλεονέκτημα του παραπάνω είναι ότι γνωρίζει τα ονόματα των κόμβων και μπορεί αυτόματα να αντιστοιχίσει ονόματα με διευθύνσεις IP. Για παράδειγμα μπορούμε να τρέξουμε την εντολή h2 ping h3!

Αναφορικά με το παραπάνω σχήμα η εντολή αυτή στέλνεται στη διεργασία που εξομοιώνει το τερματικό2, παράγοντας ένα ICMP echo request που περνάει από την ιδιωτική διεπαφή h2-eth0 του h2 και μπαίνει στο πυρήνα του μηχανήματος μέσω ενός ζεύγους veth. Το μήνυμα λαμβάνεται από το μεταγωγέα s1 που βρίσκεται στο διαχειριστικό τόπο ονομάτων και μετά φτάνει στο h3 μέσω του άλλου ζεύγους veth.

3 *ΠΕΡΙΓΡΑΦΗ ΕΝΤΟΛΩΝ ΚΑΙ ΛΕΙΤΟΥΡΓΙΩΝ*

Στο παρών κεφάλαιο θα ασχοληθούμε με το mininet εκτενέστερα. Συγκεκριμένα αφού εφοδιάσουμε τον αναγνώστη με ένα σαφή και λεπτομερή οδηγό εγκατάστασης ώστε να μπορέσει και εκείνος να εκτελέσει τους κώδικες που θα παρέχουμε παρακάτω, θα προχωρήσουμε στην αναλυτική περιγραφή των εντολών που μπορεί να χρησιμοποιηθούν για να οριστούν νέες τοπολογίες .Τέλος θα δώσουμε ένα μικρό παράδειγμα χρήσης των εντολών για την κατασκευή μίας πολύ απλής τοπολογίας

3.1 Εισαγωγή στο mininet

Το mininet είναι ένα εργαλείο που χρησιμοποιείται για δικτυακές εξομείωσεις και τρέχει σε περιβάλλον Linux. Το mininet μπορεί να δημιουργήσει ένα ρεαλιστικό εικονικό δίκτυο μέσω κώδικα που τρέχει πάνω σε ένα και μόνο vm (virtual machine) .Συνολικά το mininet δημιουργεί μια δομή από τελικά τερματικά(PC) , μεταγωγείς(switches: χρησιμοποιούνται για να διαχωρίζουν διαφορετικά μεταξύ τους υποδίκτυα) , δρομολογητές(routers: όμοιοι με τα τελικά τερματικά μόνο που έχουν την ip_forwarding λειτουργία ενεργοποιημένη) πάνω στο ίδιο λειτουργικό σύστημα. Κάθε τερματικό στο mininet συμπεριφέρεται σαν ένα αληθινό μηχάνημα, μπορούμε να συνδεθούμε σε αυτό με ssh(secure shell protocol) και μπορούμε να τρέξουμε προγράμματα πάνω σε αυτό με την προϋπόθεση βέβαια ότι είναι εγκατεστημένα στο vm που τρέχει το mininet. Τα προγράμματα αυτά μπορούν να στέλνουν πακέτα μέσω εικονικών διεπαφών τα οποία επεξεργάζονται οι μεταγωγείς και οι δρομολογητές πριν φτάσουν στην τελική τοποθεσία. Συγκεφαλαιώνοντας το mininet υπερέχει γιατί δημιουργεί όλα τα κομμάτια ενός δικτύου με τη βοήθεια λογισμικού και χωρίς τη χρήση υλικού και η συμπεριφορά του δικτύου στις περισσότερες των περιπτώσεων είναι όμοια. Στη συνέχεια θα χρησιμοποιήσουμε το mininet για να ορίσουμε τις δικές μας τοπολογίες με τη χρήση της γλώσσας Python.[4]

3.2 Οδηγίες εγκατάστασης

Αν δουλεύετε πάνω σε μία πρόσφατη έκδοση Linux Ubuntu μπορείτε να εγκαταστήσετε το πακέτο του mininet, διαφορετικά θα χρειαστείτε ένα Ubuntu VM και μία πλατφόρμα για να το σηκώσετε. Για να δείτε ποιά είναι ή έκδοση του λειτουργικού σας πληκτρολογείστε
lsb_release -a

Μετά με βάση την έκδοση που σας έδωσε η παραπάνω εντολή επιλέξτε μόνο ένα από τα παρακάτω

Mininet 2.1.0 on Ubuntu 13.10: sudo apt-get install mininet
Mininet 2.0.0 on Ubuntu 13.04: sudo apt-get install mininet
Mininet 2.0.0 on Ubuntu 12.10: sudo apt-get install mininet/quantal-backports
Mininet 2.0.0 on Ubuntu 12.04: sudo apt-get install mininet/precise-backports

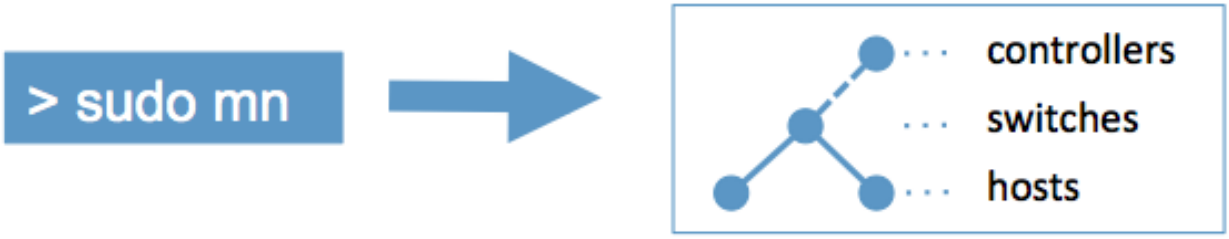
Μετά το τέλος της εγκατάστασης θα πρέπει να σταματήσετε την υπηρεσία openvswitch-controller αν συνεχίζει να τρέχει

sudo service openvswitch-controller stop
sudo update-rc.d openvswitch-controller disable

Για να ελέγξετε άν το mininet όντως εγκαταστάθηκε σωστά πληκτρολογήστε

sudo mn --test pingall

Στην παραπάνω εντολή το mininet δημιουργεί την απλούστερη τοπολογία(δύο τερματικά: h1, h2, ένα μεταγωγέα: s1, ένα controller: Θα αναφερθούμε αργότερα σε αυτό) που φαίνεται και στην εικόνα παρακάτω και κάνει εκατέρωθεν ping μεταξύ των δύο τερματικό για να τεστάρει την επικοινωνία.



Σημείωση: Αν το mininet εμφανίζει μύνημα σφάλματος ότι η υπηρεσία Open vSwitch δεν λειτουργεί θα χρειαστεί να τρέξετε την εντολή

sudo dpkg-reconfigure openvswitch-datapath-dkms
sudo service openflow-switch restart

Για να μπορέσετε να περατώσετε με επιτυχία την επίδειξη που θα ακολουθήσει θα χρειαστεί να τρέξετε τις παρακάτω εντολές που θα εγκαταστήσουν κάποια χρήσιμα εργαλεία όπως το περιβάλλον Wireshark για Ubuntu
git clone git: //github.com/mininet/mininet
mininet/util/install.sh -fw

Εναλλακτικά μπορείτε να κατεβάσετε το έτοιμο μηχάνημα Master-Thesis-VM από το σύνδεσμο.
https://pithos.okeanos.grnet.gr/public/HAeuvTGCFtdsJ1EU1JymU5
Περιέχει έτοιμους τους κώδικες καθώς και όλα τα αρχεία που είναι απαραίτητα για να δουλέψει κανείς.

Για να μπορείτε να δουλέψετε αποτελεσματικά θα πρέπει να μπορείτε να εμφανίζετε το wireshark, αλλά και τις οθόνες των τερματικών που δημιουργούνται στο mininet στο περιβάλλον windows του φιλοξενούντος μηχανήματος(Η σε όποιο άλλο περιβάλλον δουλεύετε) .

Για το σκοπό αυτό θα χρησιμοποιήσουμε προώθηση X11(X11 forwarding)
Δε θα επεκταθούμε περισσότερο σε αυτή τη τεχνολογία.
Για όποιον ενδιαφέρεται περισσότερο http: //en.wikipedia.org/wiki/X_Window_System

Κατεβάστε το Xming και τα fonts του
Xming http: //sourceforge.net/projects/xming/files/latest/download?source=files
Fonts : http: //sourceforge.net/projects/xming/files/Xming-fonts/7.5.0.70/Xming-fonts-7-5-0-70-setup.exe/download

Κατεβάστε το PuTTY (ssh client για windows)
http: //the.earth.li/~sgtatham/putty/latest/x86/putty.exe

Ρύθμιση του Xming
Σε περιβάλλον windows κάνετε κλικ στο Start και επιλέγετε Xming → XLaunch
Στο παράθυρο διαλόγου που θα εμφανιστεί επιλέγετε τη ρύθμιση Multiple windows, στη θέση Display number βάζετε 0 και κάνετε κλικ στο κουμπί Next.
Συνεχίζετε να πατάτε Next μέχρι να φτάσετε στο παράθυρο διαλόγου Finish configuration οπότε και πατάτε Finish

Στο virtual Box θα πρέπει να έχετε ένα interface σε δικτύωση host only για να μην αλλάζει η IP και να μένει σταθερή

Ρύθμιση του PuTTY

Ξεκινήστε το PuTTY
Στα αριστερά επιλέγετε Connection → SSH → X11
Στα δεξιά επιλέγετε Enable X11 forwarding κάνοντας τικ στο αντίστοιχο κουτάκι
Στα αριστερά επιλέγετε Session(μπορεί να χρειαστεί να χρησιμοποιήσετε τη μπάρα κύλισης καθώς είναι η πιο πάνω επιλογή)
Καταχωρήστε την διεύθυνση IP της διεπαφής που θα συνδέεστε σταθερά από δω και πέρα.
Καταχωρήστε ένα όνομα κάτω από το Saved Sessions
Κάντε κλικ στο Save
Από εδώ και πέρα θα μπορείτε να συνδέεστε κατευθείαν με το εικονικό μηχάνημα(εφόσον είναι σε λειτουργία) κάνοντας απλά κλικ στο όνομα που καταχωρήσατε στη συνέχεια κλικ στο Load και τέλος πατώντας Open

Για να δοκιμάσετε ότι το X11 forwarding λειτουργεί συνδεθείτε με ssh στο εικονικό μηχάνημα και πληκτρολογείστε
sudo wireshark
αν όλα πάνε καλά θα εμφανιστεί το παράθυρο φόρτωσης του wireshark στην οθόνη των Windows.

Τέλος για να μπορείτε στο εικονικό μηχάνημα να ανοίγετε πάνω από ένα παράθυρα θα εγκαταστήσετε θα χρησιμοποιήσετε το πρόγραμμα screen πάνω στο Ubuntu.
sudo apt-get install screen

Στη συνέχεια πληκτρολογείτε screen.Με [Ctrl: πατημένο]+A+C ανοίγετε δεύτερη κονσόλα
και με [Ctrl: πατημένο]+A+p πηγαίνετε πίσω και με [Ctrl: πατημένο]+A+n πηγαίνετε εμπρός!

Τέλος για να φύγετε από τη κατάσταση πολλαπλών παραθύρων πληκτρολογείτε
[Ctrl: πατημένο]+A+d

Για περισσότερες πληροφορίες σχετικά με το screen μπορείτε να ανατρέξετε στο
https: //help.ubuntu.com/community/Screen

3.3 Εντολές του mininet

Για να ξεκινήσετε το mininet και να φορτώσετε τη μικρότερη προεπιλεγμένη τοπολογία(Θα αναφερθούμε στις τοπολογίες παρακάτω και πως μπορούμε να φορτώνουμε δικές μας τοπολογίες) πληκρολογείστε

sudo mn

Σημείωση: Αν έχετε τρέξει ξανά το mininet πριν, επιβάλλετε προτού ξεκινήσετε μία νέα προσομοίωση να πληκτρολογείτε sudo mn -c ώστε να καθαρίζει ο χώρος εργασίας από εναπομείναντα σκουπίδια. Και μετά κανονικά sudo -mn ….

Μεταφέρεστε πλέον στο CLI (Command Line Interface) του MININET στο οποίο μπορείτε πλέον να τρέχετε όλες τις εντολές που αφορούν την τοπολογία και όχι μόνο.

Μερικές χρήσιμες εντολές για τη συνέχεια είναι

help: Εμφανίζει βοηθητική πληροφορία σχετικά με το MININET
nodes: Εμφανίζει όλους τους κόμβους της τοπολογίας με τα ονόματά τους
links: Εμφανίζει όλες τις συνδέσεις των κόμβων της τοπολογίας
dump: Εμφάνισε ολοκληρωμένη πληροφορία για τις συνδέσεις τον κόμβων

Αν πριν από μία εντολή προηγηθεί το όνομα κάποιου κόμβου της τοπολογίας τότε η εντολή θα εκτελεστεί αποκλειστικά και μόνο σε αυτό το κόμβο.
Πχ: h1 ifconfig
ΠΡΟΣΟΧΗ:Η εντολή ifconfig είναι bugged όταν χρησιμοποιείται για εισαγωγή IP,αντ'αυτού θα χρησιμοποιείται την py PC1.intf('PC1-eth0').setIP('192.168.0.1/24').

Με αυτό το τρόπο μπορείτε να ελέγξετε την επικοινωνία μεταξύ δύο τερματικών φερ' ειπείν.
Ο αριθμός μετά το -c αντιστοιχεί στα πόσα πακέτα ICMP ECHO REQUEST θα σταλούν.
Πχ: h1 ping -c 1 h2

Για να ελέγξετε την επικοινωνία όλων τον κόμβων
Πχ: pingall

Για να προσομοιώσετε βλάβη σύνδεσης μεταξύ δύο κόμβων: link h1 s1 down.
Αντίστοιχα για να επαναφέρετε τη σύνδεση πληκτρολογείτε link h1 s1 up

Για να ανοίξετε μία ξεχωριστή οθόνη για ένα κόμβο: xterm h1
και για να την κλείσετε πληκτρολογείτε exit

Για να ορίσετε κανόνες openflow χρησιμοποιείτε την εντολή sh ovs-ofctl add-flow <όνομα-μεταγ>
Περισσότερα για εντολές openflow θα δούμε στο κομμάτι των αναλυτικών παραδειγμάτων παρακάτω

3.4 Δημιουργία τοπολογίας στο mininet

Στο εικονικό μηχάνημα στο φάκελο /home/mininet/mininet/custom βρίσκεται το αρχείο topo-2sw-2host.py το οποίο είναι ένα python script

"""Παράδειγμα
τοπολογίας

Δύο μεταγωγείς με ένα τερματικό συνδεδεμένο στο καθένα

τερματικό --- μεταγωγέας --- μεταγωγέας --- τερματικό

"""

from mininet.topo import Topo

class MyTopo(Topo) :
"Ενα απλό παράδειγμα"

def __init__(self) :
"Μία δικιά μας δημιουργία"

# Αρχικοποίηση της τοπολογίας
Topo.__init__(self)

# Εδώ προσθέτετε ή αφαιρείτε κώδικα
leftHost = self.addHost('h1')
rightHost = self.addHost('h2')
leftSwitch = self.addSwitch('s3')
rightSwitch = self.addSwitch('s4')

# Εδώ προσθέτετε τις συνδέσεις σας
self.addLink(leftHost, leftSwitch)
self.addLink(leftSwitch, rightSwitch)
self.addLink(rightSwitch, rightHost)

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

Αντιγράφοντας το περιεχόμενο του σε ένα άλλο αρχείο θα μπορείτε στη συνέχεια να δημιουργήσετε τις δικές σας τοπολογίες αλλάζοντας απλά κάποιες γραμμές κώδικα στο νέο αρχείο.
Με την εντολή X=self.addHost(' κ ')
Προσθέτετε ένα τερματικό στη τοπολογία και επιστρέφετε το ονομά του σαν string(' κ ') στη μεταβλητή X

Με την εντολή X=self.addSwitch(' κ ')
Προσθέτετε ένα μεταγωγέα στη τοπολογία και επιστρέφετε το όνομά του σαν string στη μεταβλητή X

Με την εντολή self.addLink(X, Y) δηλώνεται ότι στην τοπολογία σας υπάρχει σύνδεση του X με τον Y και αντίστροφα
Αλλάζετε το αρχείο κατα το δοκούν, αποθηκεύετε και τρέχετε τη νέα σας τοπολογία με την εντολή sudo -mn --custom /home/mininet/mininet/<όνομα του αρχείου σας> --topo mytopo –controller=ovsc

Δεν θα επεκταθούμε σε αυτό το σημείο στο γιατί ο controller είναι ovsc. Πρακτικά θέλουμε τους μεταγωγείς μας να υλοποιούν MAC address forwarding.

Το Mininet διαθέτει επίσης και έτοιμες προκατασκευασμένες τοπολογίες στις οποίες όμως δεν θα αναφερθούμε.[5]

4 *ΠΡΟΒΛΗΜΑΤΑ ΠΟΥ ΑΝΤΙΜΕΤΩΠΙΣΑΜΕ*

Σε αυτό το κεφάλαιο θα αναφερθούμε στα προβλήματα που αντιμετωπίσαμε κατά τη διάρκεια συγγραφής της διπλωματικής εργασίας και πως τα επιλύσαμε. Σε κάποιες περιπτώσεις η λύση ήταν απλή και αρκούσε καθαρό μυαλό και ένα ευφυολόγημα σε επίπεδο εντολών. Σε άλλη περίπτωση όμως απαιτήθηκε ριζική αλλαγή του υπάρχων κώδικα σε κάποια σημεία ώστε να μπορέσει να υποστηρίξει με επιτυχία ανεξάρτητα συστήματα αρχείων για κάθε κόμβο.

4.1 Εισαγωγή στο miniNExT

Μέχρι στιγμής με τη χρήση του mininet είδαμε πως μπορούμε να κατασκευάζουμε απλές τοπολογίες με δρομολογητές και μεταγωγείς. Τι θα συμβεί όμως άμα πάμε να τρέξουμε ένα πρόγραμμα δικτυακής δρομολόγησης σε κάποιους από τα τερματικά. Στην αρχή θα νομίσουμε ότι όλα βαίνουν καλώς,στη συνέχεια όμως θα παρατηρήσουμε ότι όλα τα τερματικά τροποποιούν το ίδιο αρχείο παραμετροποίησης(που ανήκει στο εικονικό μηχάνημα που τρέχει το mininet) με αποτέλεσμα να μην μπορούμε να εισαγάγουμε ξεχωριστές ρυθμίσεις για τον καθένα. Αυτό συμβαίνει γιατί όλα τα μηχανήματα που δημιουργεί το mininet μοιράζονται το ίδιο σύστημα αρχείων, αυτό του φιλοξενούντος μηχανήματος. Πρακτικά δηλαδή δεν μπορούμε να τρέξουμε προγράμματα που απαιτούν ξεχωριστή ρύθμιση παραμέτρων σε αρχείο /etc για κάθε μηχάνημα. Χρειαζόμαστε λοιπόν επιπρόσθετη απομόνωση σε επίπεδο αρχείων (mount namespaces).Για το σκοπό αυτό θα χρησιμοποιήσουμε το miniNExT μία επέκταση του mininet που μας επιτρέπει ο κάθε τερματικό να έχει τα δικά του αρχεία παραμετροποίησης μέσα στο container που ορίζει το mininet.

Οδηγίες εγκατάστασης!

Θεωρούμε ότι έχετε ήδη εγκαταστήσει το περιβάλλον mininet

Μεταβείτε στο φάκελο /home/mininet
Πληκτρολογείστε git clone git://github.com/USC-NSL/miniNExT (Προσοχή για να λειτουργήσει η εντολή πρέπει το virtual box να είναι ρυθμισμένο έτσι ώστε το εικονικό μηχάνημά σας να έχει πρόσβαση στο διαδίκτυο)
στη συνέχεια make deps
μετά sudo apt-get install 'make deps'
και τέλος sudo make install

Κατασκευή τοπολογιών με miniNExT:

Η μέθοδος που θα ακολουθήσουμε εδώ διαφέρει απο όσα έχουμε δει μέχρι τώρα. Θα χρειαστεί να γράφουμε πιο αναλυτικό κώδικα για την τοπολογία μας καθώς και να ορίσουμε χειροκίνητα τους τόπους ονομάτων που θα "ζεί" το κάθε τερματικό.

Μεταβείτε στο φάκελο /home/mininet/miniNExT/examples/quagga-ixp/.Θα βρείτε ένα αρχείο μέσα με το όνομα topo.py

Παρακάτω θα αναλύσουμε τις εντολές που χρησιμοποιούμε στην τοπολογία μας και τι μας προσφέρει η κάθε μία!

```
"""
Παράδειγμα τοπολογίας με τη χρήση δρομολογητών Quagga
"""

import inspect
import os
from mininext.topo import Topo
from mininext.services.quagga import QuaggaService

net = None


class QuaggaTopo(Topo):

    "Κατασκευάζω και αρχικοποιώ την τοπολογία μου"

    def __init__(self):

        Topo.__init__(self)
```

Με τη χρηση της εντολής αυτής αποθηκεύουμε στη μεταβλητή selfPath τη διαδρομή για το αρχείο που βρίσκεται ο κωδικάς μας.

```
            selfPath = os.path.dirname(os.path.abspath(
            inspect.getfile(inspect.currentframe())))  # script directory
```

Δημιουργώ την υπηρεσία Quagga

```
        quaggaSvc = QuaggaService(autoStop=False)
```

Η μεταβλητή quaggaBaseConfigPath περιέχει τη διαδρομή που πρέπει να ακολουθήσει το mininet για να βρεί τα αρχεία παραμετροποίησης του κάθε τερματικό

```
        quaggaBaseConfigPath = selfPath + '/configs/'
```

Με την παρακάτω εντολή προσθέτω έναν νέο τερματικό στην τοπολογία μου και θέτωντας τα αναγνωριστικά true ορίζω στην ουσία επιπλέον απομόνωση(isolation) που χρειάζομαι για να μπορεί να έχει το κάθε τερματικό τις δικές του διεργασίες και τα δικά του αρχεία παραμετροποίησης.

```
        PC1=self.addHost(name='PC1',
                         hostname='PC1',
                         privateLogDir=True,
                         privateRunDir=True,
                         inMountNamespace=True,
                         inPIDNamespace=True,
                         inUTSNamespace=True)
```

Για να αρχικοποιήσουμε την υπηρεσία quagga σε αυτό τον host περνάμε στην μεταβλητή quaggaSvcConfig τη διαδρομή που ακολουθούμε για να φτάσουμε στα αρχεία που χρειάζεται το quagga για να τρέξει διάφορους δαίμονες δρομολόγησης. Κάνουμε παράθεση στο τέλος το όνομα του τερματικού που μας ενδιαφέρει

```
quaggaSvcConfig = \
    {'quaggaConfigPath': quaggaBaseConfigPath + 'PC1'}
```

Για να εισάγουμε σε ένα κόμβο τις τοπολογίας μας την παραπάνω πληροφορία καθώς και την υπηρεσία Quagga χρησιμοποιούμε σαν αναγνώριση το όνομα του κόμβου και εισάγουμε επίσης την υπηρεσία και την πληροφορία για τα αρχεία παραμετροποίησης

```
self.addNodeService(node='PC1', service=quaggaSvc,
                    nodeConfig=quaggaSvcConfig)
```

```
# Δημιουργώ ένα νέο τερματικό και το εισάγω στο δίκτυο μου
R1      = self.addHost(name='R1',
                       hostname='R1',
                       privateLogDir=True,
                       privateRunDir=True,
                       inMountNamespace=True,
                       inPIDNamespace=True,
                       inUTSNamespace=True)
```

```
# Αρχικοποιώ την υπηρεσία quagga σε αυτό το τερματικό και της παρέχω τη διαδρομή προς τα αρχεία παραμετροποίησης του
quaggaSvcConfig = \
    {'quaggaConfigPath': quaggaBaseConfigPath + 'R1'}
self.addNodeService(node='R1', service=quaggaSvc,
                    nodeConfig=quaggaSvcConfig)
```

```
# Δημιουργώ ένα νέο τερματικό και το εισάγω στο δίκτυο μου
PC2     = self.addHost(name='PC2',
                       hostname='PC2',
                       privateLogDir=True,
                       privateRunDir=True,
                       inMountNamespace=True,
                       inPIDNamespace=True,
                       inUTSNamespace=True)
```

```
        # Αρχικοποιώ την υπηρεσία quagga σε αυτό το τερματικό και της παρέχω τη διαδρομή προς τα αρχεία παραμετροποίησης του
        quaggaSvcConfig = \
                {'quaggaConfigPath': quaggaBaseConfigPath + 'PC2'}
        self.addNodeService(node='PC2', service=quaggaSvc,
                            nodeConfig=quaggaSvcConfig)
```

Τέλος εισάγω τις συνδέσεις που θα υπάρχουν στην τοπολογία μας με τις παρακάτω εντολές χρησιμοποιώντας σαν αναγνωριστικά τα ονόματα των κόμβων!

```
        self.addLink(PC1, R1)
        self.addLink(R1, PC2)
```

Τρέχετε το αρχείο με την εντολή

```
sudo  /home/mininet/miniNExT/examples/quagga-ixp/start.py
```

4.2 Πρόβλημα διπλής ηχούς

Ένα άλλο πρόβλημα που αντιμετωπίσαμε είναι πως όταν έχουμε σηκώσει την τοπολογία και θέλουμε να αποκτήσουμε πρόσβαση στο quagga ενός δρομολογητή μέσω της εντολής <όνομα δρομολογητή> vtysh η γραμμή εντολών που ανοίγει έχει πρόβλημα διπλής ηχούς.
Για την αποτελεσματική επίλυση αυτού του ζητήματος στη θέση της παραπάνω εντολής εκτελούμε sh /home/mininet/miniNExT/util/mx <όνομα δρομολογητή> vtysh
Πρακτικά σημαίνει ότι η εντολή εκτελείται μέσω του προθέματος sh, στο κέλυφος του φιλοξενούντος μηχανήματος. Το προγραμματάκι mx που καλούμε μας δίνει τη δυνατότητα να ξεκινήσουμε ένα bash script μέσα στο container που βρίσκεται ο εκάστοτε δρομολογητής στον οποίο θα εκκινήσει το quagga.

4.3 Πρόβλημα capturing λόγω του extra isolation

Κινούμενοι στο ίδιο μήκος κύματος παρατηρήσαμε ότι η επιπλέον απομόνωση που προσφέρει το miniNExT εμποδίζει το xterm να εκτελεστεί. Απόρροια του προηγούμενου είναι η αδυναμία να τρέξουμε το wireshark σε ένα κόμβο και παράλληλα να επιφέρουμε αλλαγές στη τοπολογία καθώς όλα γίνονται στο ίδιο παράθυρο. Επομένως το wireshark θα πρέπει να τρέξει πάνω στο λάπτοπ. Το πρόβλημα είναι πως μόνο οι μεταγωγείς βρίσκονται πάνω στο διαχειριστικό τόπο ονομάτων τα οποία χρησιμοποιεί το mininet για να συνδέει τους κόμβους μεταξύ τους και βρίσκονται στην εμβέλεια του wireshark. Επομένως για να κάνουμε capture ανάμεσα σε δύο κόμβους (δρομολογητής,τερματικό) θα πρέπει να παρεμβάλλουμε ένα μεταγωγέα ανάμεσα τους.

Επομένως αντί για την εντολή addLink θα χρησιμοποιούμε την εντολή addcustlink ο κώδικας της οποίας είναι ο εξής:

```
def addcustlink(self,pc1, pc2):
    sw=self.addSwitch(pc1+'-'+pc2)
    self.addLink(pc1,sw)
    self.addLink(sw,pc2)
```

Επομένως για να δούμε τα πακέτα που ανταλλάσσονται εκατέρωθεν του H1 και R1 αρκεί να κάνω capture στην αντίστοιχη θύρα του μεταγωγέα H1-R1.
Υπενθυμίζουμε ότι για να αποκτήσει αναλυτική πληροφορία ο χρήστης για τις συνδέσεις των κόμβων πληκτρολογεί dump. Έχοντας ολοκληρώσει τη περιγραφή των εργαλείων που θα χρησιμοποιήσουμε θα περάσουμε στη παρουσίαση των σχετικών τοπολογιών που δημιουργήσαμε, παραθέτοντας κάθε φορά τον αντίστοιχο κώδικα.

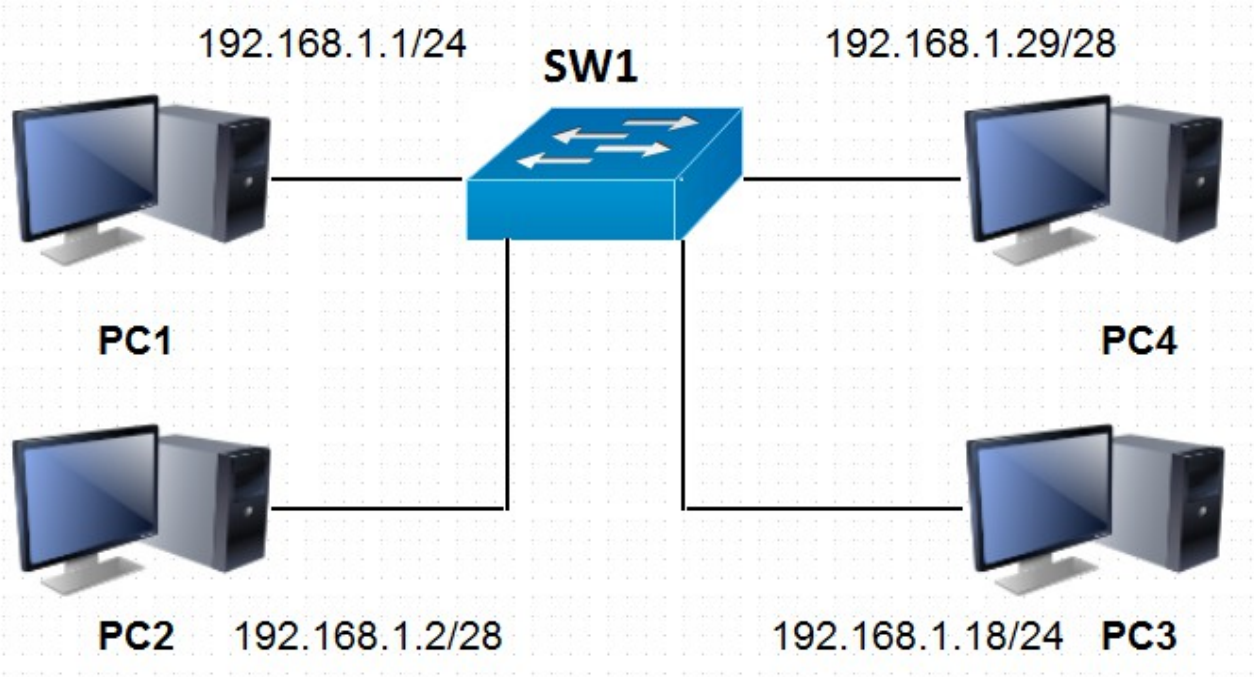
5 *ΥΛΟΠΟΙΗΣΗ-ΚΩΔΙΚΑΣ*

Στο παρόν κεφάλαιο θα ασχοληθούμε αποκλειστικά και μόνο με τον κώδικα της διπλωματικής. Δεν θα παραθέσουμε όλα τα αρχεία καθώς είναι μεγάλα σε αριθμό

Ακολουθούν μερικά παραδείγματα κώδικα που συνοδεύονται από μία γραφική απεικόνιση του δικτύου που κατασκευάζουμε.

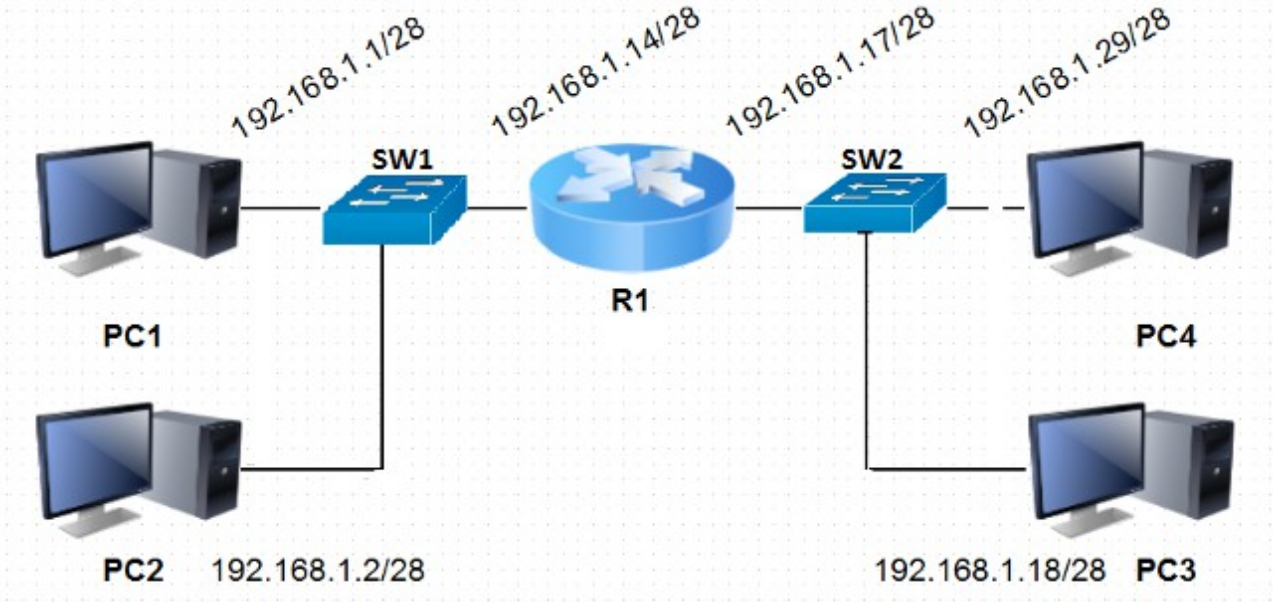
Σημείωση: Για να δουλέψουν οι πολύπλοκες τοπολογίες με το miniNExT απαιτούνται τα παρακάτω. Θα πρέπει να δημιουργήσετε στο φάκελο"/home/mininet/miniNExT/examples/quagga-ixp/configs/" ένα φάκελο με το όνομα του κάθε host-router, φερ ειπείν "/home/mininet/miniNExT/examples/quagga-ixp/configs/R1", και σε αυτόν τα αρχεία debian.conf, zebra.conf, daemons, ripd.conf, ospfd.conf, bgpd.conf και κάθε φορά θα αντικαθιστάτε το περιεχόμενο του αρχείου topo.py με το κώδικα της τοπολογίας που θέλετε να τρέξετε κάθε φορά (κώδικας κάτω από τη τοπολογία).Επίσης για κάθε host που λειτουργεί σαν δρομολογητής θα πρέπει να δώσετε την εντολή <host> sysctl -w net.ipv4.ip_forward=1

Τοπολογία για τη μελέτη της συμπεριφοράς 4 μηχανημάτων, με διαφορετικές μάσκες υποδικτύου, όταν συνδέονται στο ίδιο τοπικό δίκτυο



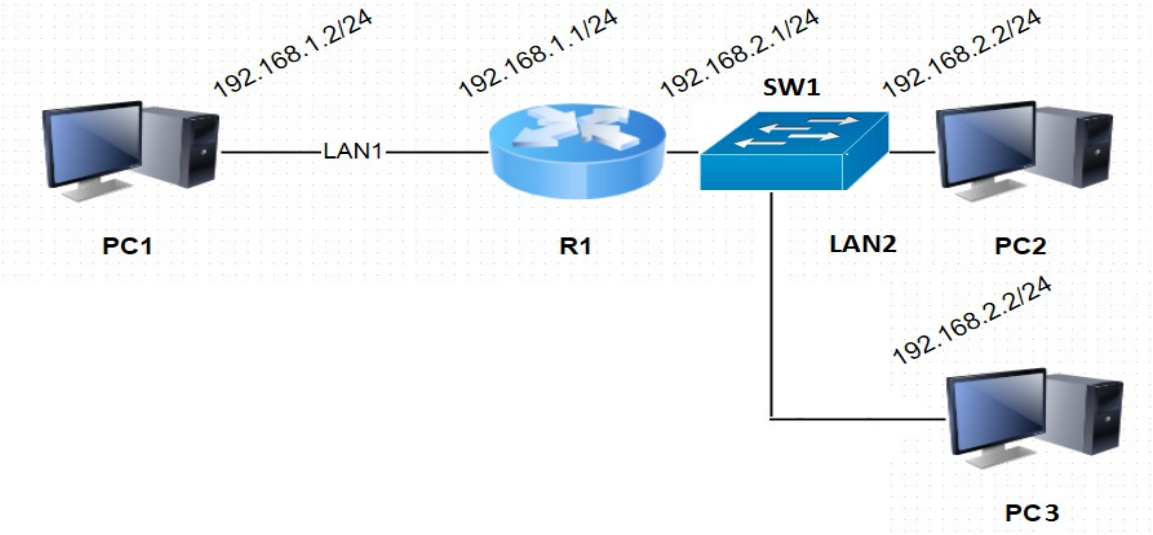

```
from mininet.topo import Topo

class MyTopo( Topo ) :
    "Ένα απλό παράδειγμα"

    def __init__( self ) :


        # Αρχικοποίηση της τοπολογίας
        Topo.__init__( self )

        # Προσθέτω υπολογιστές και δρομολογητές
        PC1 = self.addHost( 'PC1' )
        PC2 = self.addHost( 'PC2' )
        PC3 = self.addHost( 'PC3' )
        PC4 = self.addHost( 'PC4' )
        SW1 = self.addSwitch( 'SW1')

        # Προσθέτω τις συνδέσεις
        self.addLink( PC1,  SW1 )
        self.addLink( PC2,  SW1 )
        self.addLink( PC3,  SW1 )
        self.addLink( PC4,  SW1 )

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

Τοπολογία για τη μελέτη της συμπεριφοράς ενός απλού δικτύου με δρομολογητή.



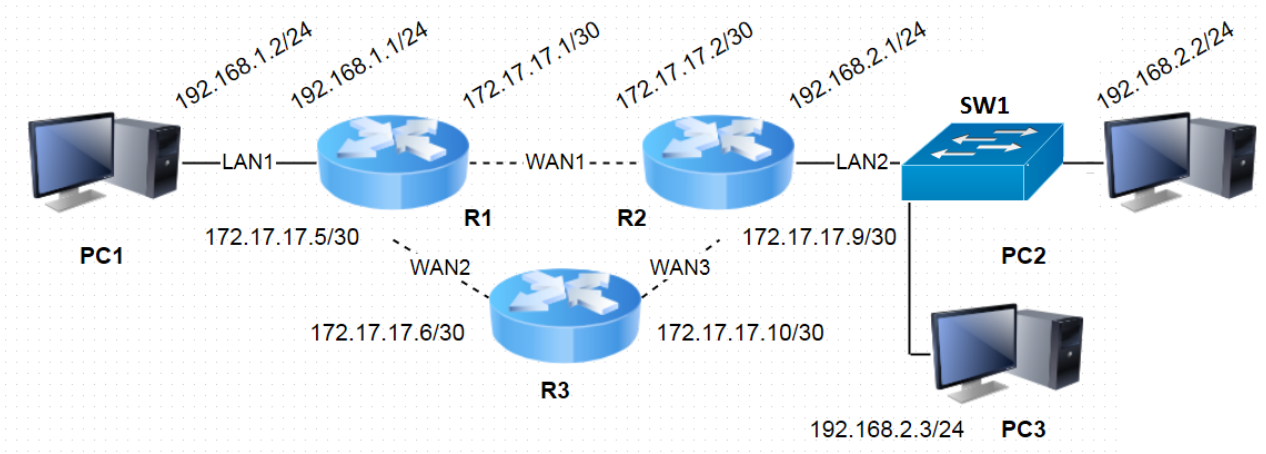

```
from mininet.topo import Topo

class MyTopo( Topo ) :
    "Ενα απλό παράδειγμα"

    def __init__( self ) :
        "Δημιουργία δικής μας τοπολογίας."

        # Αρχικοποίηση της τοπολογίας
        Topo.__init__( self )

        # Προσθέτω υπολογιστές και δρομολογητές
        PC1 = self.addHost( 'PC1' )
        PC2 = self.addHost( 'PC2' )
        PC3 = self.addHost( 'PC3' )
        PC4 = self.addHost( 'PC4' )
        SW1 = self.addSwitch( 'SW1')
        SW2 = self.addSwitch( 'SW2')
        R1  = self.addHost( 'R1')

        # Προσθέτω τις συνδέσεις
        self.addLink( PC1,  SW1 )
        self.addLink( PC2,  SW1 )
        self.addLink( R1,  SW1 )
        self.addLink( PC3,  SW2 )
        self.addLink( PC4,  SW2 )
        self.addLink( R1,  SW2 )
topos = { 'mytopo': ( lambda: MyTopo()  )  }
```

Τοπολογία για τη μελέτη τις συμπεριφοράς ενός δικτύου σε συνθήκες στατικής δρομολόγησης




```
from mininet.topo import Topo

class MyTopo( Topo ) :
   "Ενα απλό παράδειγμα."

   def __init__( self ) :
      "Δημιουργία δικής μας τοπολογίας."

      # Αρχικοποίηση της τοπολογίας
      Topo.__init__( self )

      # Προσθέτω υπολογιστές και δρομολογητές
      PC1 = self.addHost( 'PC1' )
      PC2 = self.addHost( 'PC2' )
      PC3 = self.addHost( 'PC3' )
      SW1 = self.addSwitch( 'SW1' )
      R1 = self.addHost( 'R1' )

      #Προσθέτω τις συνδέσεις
      self.addLink( PC1,  R1  )
      self.addLink( PC2,  SW1 )
      self.addLink( PC3,  SW1 )
      self.addLink( SW1,  R1 )



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

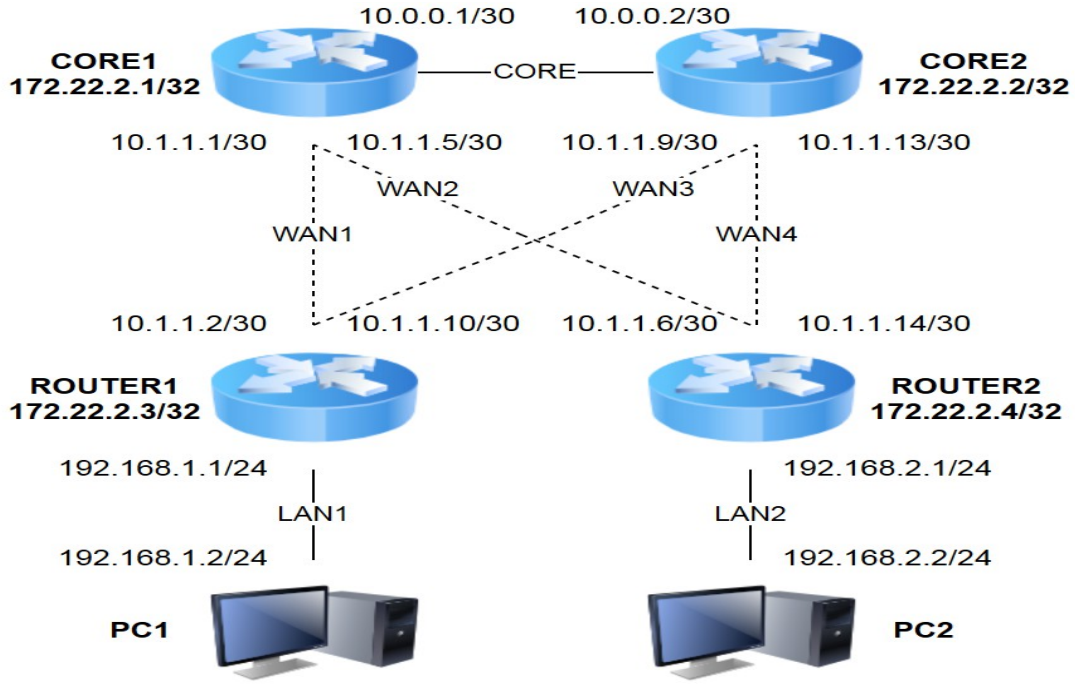
Τοπολογία για τη μελέτη ενός δικτύου σε συνθήκες στατικής δρομολόγησης




```
from mininet.topo import Topo

class MyTopo( Topo ) :
   "Ενα απλό παράδειγμα τοπολογίας."

   def __init__( self ) :
      "Δημιουργία δικής μας τοπολογίας."

      # Αρχικοποίηση τοπολογίας
      Topo.__init__( self )

      # Προσθέτω υπολογιστές και μεταγωγείς
      PC1 = self.addHost( 'PC1' )
      PC2 = self.addHost( 'PC2' )
      PC3 = self.addHost( 'PC3' )
      R1 = self.addHost( 'R1' )
      R2 = self.addHost( 'R2' )
      R3 = self.addHost( 'R3' )
      SW1=self.addSwitch( 'SW1' )

      # Προσθέτω συνδέσεις
      self.addLink( PC1,  R1  )
      self.addLink( R1,  R2 )
      self.addLink( R1,  R3 )
      self.addLink( R3,  R2 )
      self.addLink( R2,  SW1 )
      self.addLink( SW1,  PC2 )
      self.addLink( SW1,  PC3 )


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

ΔΥΝΑΜΙΚΗ ΔΡΟΜΟΛΟΓΗΣΗ ΜΕ QUAGGA ΚΑΙ miniNeXT

Τοπολογία για τη μελέτη της συμπεριφοράς του πρωτοκόλλου RIP σε ένα εταιρικό δίκτυο
Στο κώδικα αυτό έχουμε χρησιμοποιήσει και μία διεύθυνση επιστροφής(loopback) για κάθε δρομολογητή. Ο λόγος είναι ότι αυτή η διεύθυνση χρησιμοποιείται από τα διάφορα λογισμικά διαχείρισης δικτύων, ώστε ακόμα και μετά τη βλάβη κάποιας φυσικής διασύνδεσης της συσκευής να μπορεί ο εκάστοτε δρομολογητής να παραμένει προσβάσιμος μέσω κάποιας άλλης φυσικής διεπαφής. Το χαρακτηριστικό που την κάνει να ξεχωρίζει είναι ότι οι διεργασίες που τη χρησιμοποιούν στέλνουν η λαμβάνουν τη πληροφορία χρησιμοποιώντας τη διεύθυνσή της αντί για τη φυσική διεπαφή μέσω της οποίας διέρχεται η κίνηση. Επίσης η τοποθέτηση των δρομολογητών κορμού είναι τέτοια που εξασφαλίζει τη μη διακοπή της επικοινωνίας σε περίπτωση βλάβης.
Σε περίπτωση σφάλματος κάποιου καλωδίου οι διπλές διαδρομές εξασφαλίζουν την ισχυρή συνεκτικότητα του δικτύου και αποτρέπουν πιθανή κατάρρευσή του
Σημείωση:Δεν φαίνεται στην εικόνα αλλά στον κώδικα έχουμε παρεμβάλει ανάμεσα σε κάθε σύνδεση ένα switch για να μπορούμε να παρατηρήσουμε τα πακέτα που ανταλλάσσονται.



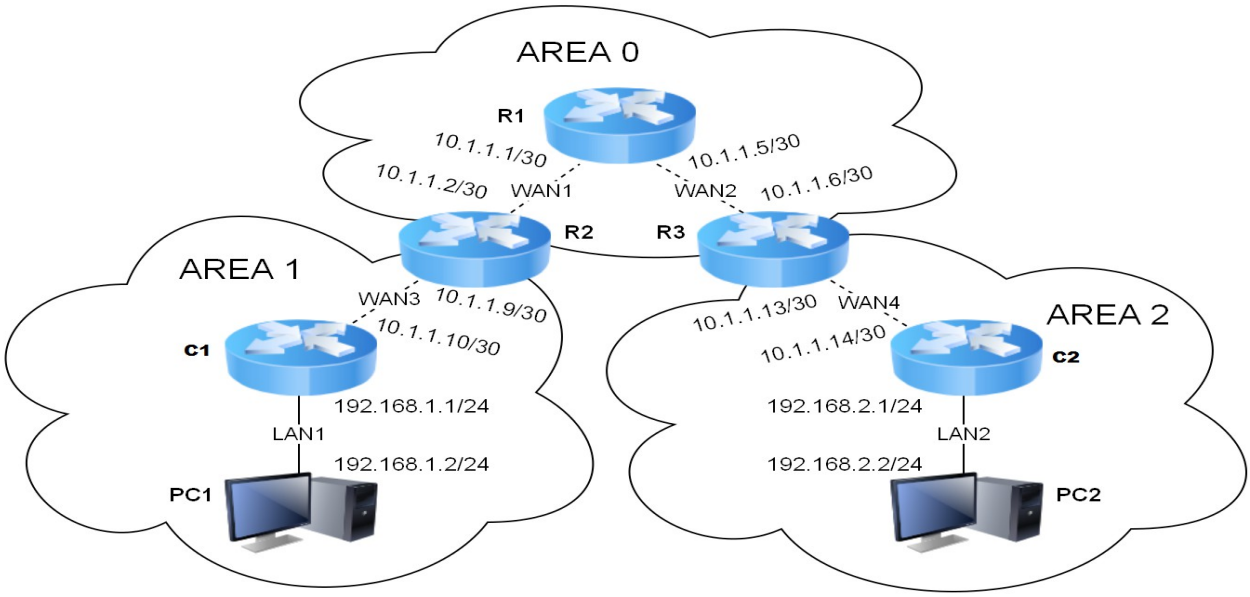

```
"""
Example topology of Quagga routers
"""

import inspect
import os
from mininext.topo import Topo
from mininext.services.quagga import QuaggaService

net = None


class QuaggaTopo(Topo) :


    def __init__(self) :


        Topo.__init__(self)
```

```
# Φάκελος στον οποίο βρίσκεται το αρχείο μας
selfPath = os.path.dirname(os.path.abspath(
   inspect.getfile(inspect.currentframe() ) ) )   # script directory



# Η υπηρεσία quagga
quaggaSvc = QuaggaService(autoStop=False)


#Η τοποθεσία που βρίσκονται τα αρχεία παραμετροποίησης για κάθε κόμβο
quaggaBaseConfigPath = selfPath + '/configs/'

# Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
PC1=self.addHost(name='PC1',
                 hostname='PC1',
                 privateLogDir=True,
                 privateRunDir=True,
                 inMountNamespace=True,
                 inPIDNamespace=True,
                 inUTSNamespace=True)

   # Αρχικοποιω την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
quaggaSvcConfig = \
      {'quaggaConfigPath': quaggaBaseConfigPath + 'PC1'}
self.addNodeService(node='PC1',  service=quaggaSvc,
                    nodeConfig=quaggaSvcConfig)


  # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
R1   = self.addHost(name='R1',
                            hostname='R1',
                            privateLogDir=True,
                            privateRunDir=True,
                            inMountNamespace=True,
                            inPIDNamespace=True,
                            inUTSNamespace=True)

                  # Προσθέτω μία “καρφωτή” διεύθυνση επιστροφής στο κόμβο R1
self.addNodeLoopbackIntf(node='R1',  ip='172.22.2.3/32')

   # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
quaggaSvcConfig = \
      {'quaggaConfigPath': quaggaBaseConfigPath + 'R1'}
self.addNodeService(node='R1',  service=quaggaSvc,
                    nodeConfig=quaggaSvcConfig)

   # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
R2   = self.addHost(name='R2',
                            hostname='R2',
                            privateLogDir=True,
                            privateRunDir=True,
                            inMountNamespace=True,
                            inPIDNamespace=True,
                            inUTSNamespace=True)

# Προσθέτω μία “καρφωτή” διεύθυνση επιστροφής στο κόμβο R2
self.addNodeLoopbackIntf(node='R2',  ip='172.22.2.4/32')


   #
quaggaSvcConfig = \
      {'quaggaConfigPath': quaggaBaseConfigPath + 'R2'}
self.addNodeService(node='R2',  service=quaggaSvc,
                    nodeConfig=quaggaSvcConfig)
```

```
                # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
C1   = self.addHost(name='C1',
                    hostname='C1',
                    privateLogDir=True,
                    privateRunDir=True,
                    inMountNamespace=True,
                    inPIDNamespace=True,
                    inUTSNamespace=True)


# Προσθέτω μία “καρφωτή” διεύθυνση επιστροφής στο κόμβο C1
self.addNodeLoopbackIntf(node='C1',  ip='172.22.2.1/32')



   # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
quaggaSvcConfig = \
      {'quaggaConfigPath': quaggaBaseConfigPath + 'C1'}
self.addNodeService(node='C1',  service=quaggaSvc,
                nodeConfig=quaggaSvcConfig)


                # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
C2   = self.addHost(name='C2',
                    hostname='C2',
                    privateLogDir=True,
                    privateRunDir=True,
                    inMountNamespace=True,
                    inPIDNamespace=True,
                    inUTSNamespace=True)

#  Προσθέτω μία “καρφωτή” διεύθυνση επιστροφής στο κόμβο C1
self.addNodeLoopbackIntf(node='C2',  ip='172.22.2.2/32')


   # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
quaggaSvcConfig = \
      {'quaggaConfigPath': quaggaBaseConfigPath + 'C2'}
self.addNodeService(node='C2',  service=quaggaSvc,
                nodeConfig=quaggaSvcConfig)

  # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
PC2 = self.addHost(name='PC2',
                    hostname='PC2',
                    privateLogDir=True,
                    privateRunDir=True,
                    inMountNamespace=True,
                    inPIDNamespace=True,
                    inUTSNamespace=True)

   # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
quaggaSvcConfig = \
      {'quaggaConfigPath': quaggaBaseConfigPath + 'PC2'}
self.addNodeService(node='PC2',  service=quaggaSvc,
                nodeConfig=quaggaSvcConfig)


   # Δημιουργώ τις συνδέσεις
self.addcustlink(PC1,  R1)
self.addcustlink(R1,  C1)
self.addcustlink(R1,  C2)
self.addcustlink(C1,  C2)
self.addcustlink(C2,  R2)
self.addcustlink(C1,  R2)
self.addcustlink(PC2,  R2)
```

```
def addcustlink(self, pc1, pc2) :
    sw=self.addSwitch(pc1+'-'+pc2)
    self.addLink(pc1, sw)
    self.addLink(sw, pc2)
```

Τοπολογία για τη μελέτη της συμπεριφοράς του πρωτοκόλλου OSPF σε ένα δίκτυο 3 διαφορετικών περιοχών καθώς και για τη μελέτη της διανομής πληροφορίας άλλων πρωτοκόλλων (RIP,BGP) μέσω του OSPF.



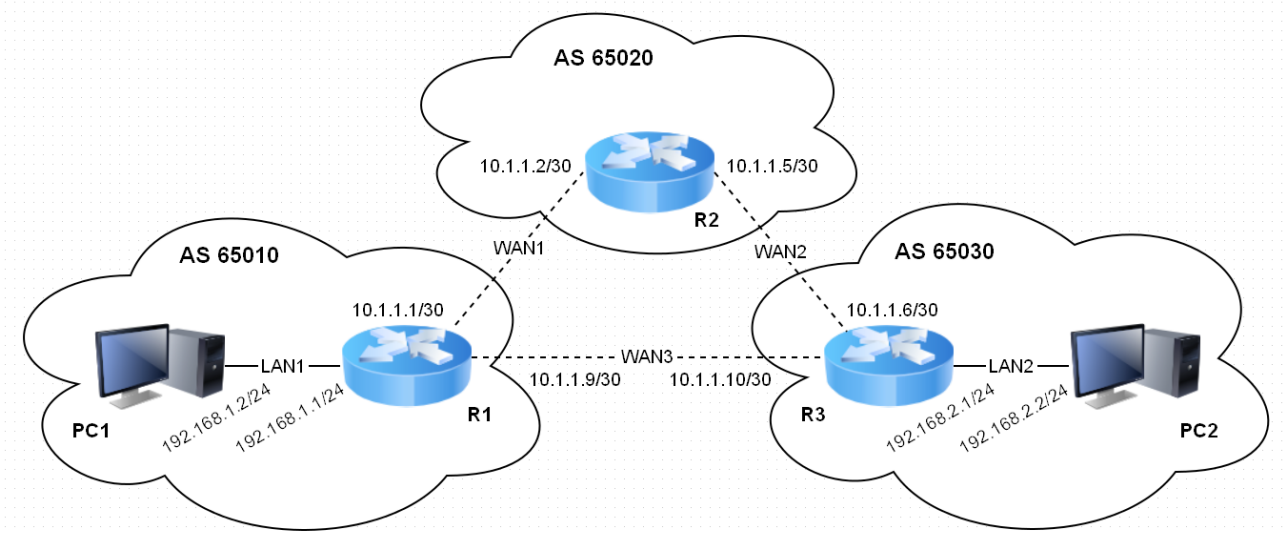

```
"""
Example topology of Quagga routers
"""

import inspect
import os
from mininext.topo import Topo
from mininext.services.quagga import QuaggaService

net = None


class QuaggaTopo(Topo) :

    "Creates a topology of Quagga routers"

    def __init__(self) :


        Topo.__init__(self)
```

```
    # Φάκελος στον οποίο βρίσκεται το αρχείο μας
    selfPath = os.path.dirname(os.path.abspath(
       inspect.getfile(inspect.currentframe() ) ) )   # script directory

    #Η υπηρεσία quagga
    quaggaSvc = QuaggaService(autoStop=False)

    #  #Η τοποθεσία που βρίσκονται τα αρχεία παραμετροποίησης για κάθε κόμβο
    quaggaBaseConfigPath = selfPath + '/configs/'




# Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    PC1=self.addHost(name='PC1',
                     hostname='PC1',
                     privateLogDir=True,
                     privateRunDir=True,
                     inMountNamespace=True,
                     inPIDNamespace=True,
                     inUTSNamespace=True)

       # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
          {'quaggaConfigPath': quaggaBaseConfigPath + 'PC1'}
    self.addNodeService(node='PC1',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)

    # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    R1  = self.addHost(name='R1',
                               hostname='R1',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)
       #Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
          {'quaggaConfigPath': quaggaBaseConfigPath + 'R1'}
    self.addNodeService(node='R1',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)

    # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    R2  = self.addHost(name='R2',
                               hostname='R2',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)

       # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
          {'quaggaConfigPath': quaggaBaseConfigPath + 'R2'}
    self.addNodeService(node='R2',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)
```

```
    # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    R3  = self.addHost(name='R3',
                               hostname='R3',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)


# Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
          {'quaggaConfigPath': quaggaBaseConfigPath + 'R3'}
    self.addNodeService(node='R3',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)

                    # Dimiourgo ena neo host kai ton eisago sto diktio mou
    C1  = self.addHost(name='C1',
                               hostname='C1',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)

       # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
          {'quaggaConfigPath': quaggaBaseConfigPath + 'C1'}
    self.addNodeService(node='C1',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)

                    # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    C2  = self.addHost(name='C2',
                               hostname='C2',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)

       # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
          {'quaggaConfigPath': quaggaBaseConfigPath + 'C2'}
    self.addNodeService(node='C2',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)

       # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    PC2         = self.addHost(name='PC2',
                               hostname='PC2',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)

       # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
          {'quaggaConfigPath': quaggaBaseConfigPath + 'PC2'}
    self.addNodeService(node='PC2',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)
```

```
        # Δημιουργώ τις συνδέσεις
      self.addcustlink(PC1,  C1)
      self.addcustlink(C1,  R2)
      self.addcustlink(R2,  R1)
      self.addcustlink(R1,  R3)
      self.addcustlink(R3,  C2)
      self.addcustlink(C2,  PC2)

   def addcustlink(self,  pc1,  pc2) :
      sw=self.addSwitch(pc1+'-'+pc2)
      self.addLink(pc1,  sw)
      self.addLink(sw,  pc2)
```

Τοπολογία για τη μελέτη της συμπεριφοράς του πρωτοκόλλου BGP σε ένα δίκτυο 3 αυτόνομων συστημάτων. Ειδικότερα συνίσταται η χρήση της τοπολογίας για την εποπτεία της λειτουργίας της διαφήμισης δικτύων από το ένα αυτόνομο σύστημα σε ένα άλλο καθώς και την ανάγκη αποδοχής και των δύο να προωθούν πακέτα το ένα στο άλλο για να μπορέσει να υπάρξει στο τέλος επικοινωνία.

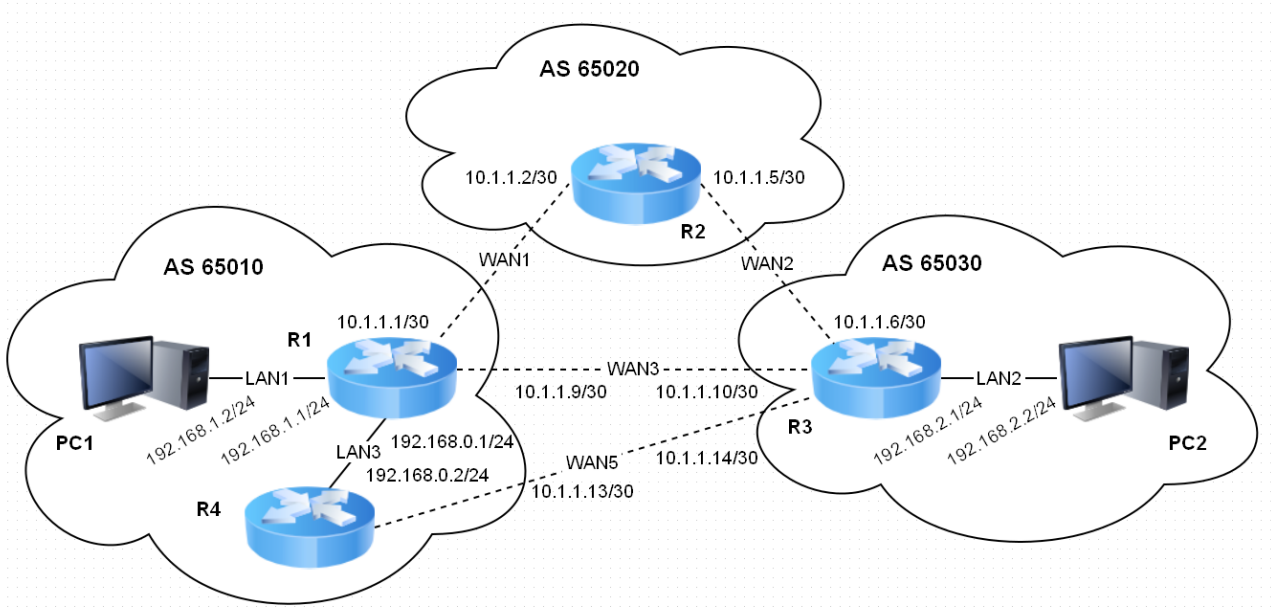



```
"""
Example topology of Quagga routers
"""

import inspect
import os
from mininext.topo import Topo
from mininext.services.quagga import QuaggaService

net = None


class QuaggaTopo(Topo) :

   "Creates a topology of Quagga routers"

   def __init__(self) :


      Topo.__init__(self)
```

```
# Φάκελος στον οποίο βρίσκεται το αρχείο μας
selfPath = os.path.dirname(os.path.abspath(
   inspect.getfile(inspect.currentframe() ) ) )   # script directory


# Η υπηρεσία quagga
quaggaSvc = QuaggaService(autoStop=False)

#  #Η τοποθεσία που βρίσκονται τα αρχεία παραμετροποίησης για κάθε κόμβο
quaggaBaseConfigPath = selfPath + '/configs/'

#Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
PC1=self.addHost(name='PC1',
                 hostname='PC1',
                 privateLogDir=True,
                 privateRunDir=True,
                 inMountNamespace=True,
                 inPIDNamespace=True,
                 inUTSNamespace=True)


   # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
quaggaSvcConfig = \
       {'quaggaConfigPath': quaggaBaseConfigPath + 'PC1'}
self.addNodeService(node='PC1',  service=quaggaSvc,
                    nodeConfig=quaggaSvcConfig)




# Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
R1     = self.addHost(name='R1',
                          hostname='R1',
                          privateLogDir=True,
                          privateRunDir=True,
                          inMountNamespace=True,
                          inPIDNamespace=True,
                          inUTSNamespace=True)
   # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
quaggaSvcConfig = \
       {'quaggaConfigPath': quaggaBaseConfigPath + 'R1'}
self.addNodeService(node='R1',  service=quaggaSvc,
                    nodeConfig=quaggaSvcConfig)

# Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
R2     = self.addHost(name='R2',
                          hostname='R2',
                          privateLogDir=True,
                          privateRunDir=True,
                          inMountNamespace=True,
                          inPIDNamespace=True,
                          inUTSNamespace=True)

   # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
quaggaSvcConfig = \
       {'quaggaConfigPath': quaggaBaseConfigPath + 'R2'}
self.addNodeService(node='R2',  service=quaggaSvc,
                    nodeConfig=quaggaSvcConfig)

                      # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
R3     = self.addHost(name='R3',
                          hostname='R3',
                          privateLogDir=True,
                          privateRunDir=True,
                          inMountNamespace=True,
                          inPIDNamespace=True,
                          inUTSNamespace=True)

   # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
quaggaSvcConfig = \
       {'quaggaConfigPath': quaggaBaseConfigPath + 'R3'}
self.addNodeService(node='R3',  service=quaggaSvc,
                    nodeConfig=quaggaSvcConfig)
```

```
    # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    PC2   = self.addHost(name='PC2',
                         hostname='PC2',
                         privateLogDir=True,
                         privateRunDir=True,
                         inMountNamespace=True,
                         inPIDNamespace=True,
                         inUTSNamespace=True)


       # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
            {'quaggaConfigPath': quaggaBaseConfigPath + 'PC2'}
    self.addNodeService(node='PC2',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)

       # Δημιουργώ τις συνδέσεις
    self.addLink(PC1,  R1)
    self.addLink(R1,  R2)
    self.addLink(R2,  R3)
    self.addLink(R1,  R3)
    self.addLink(R3,  PC2)
```

Τοπολογία για τη μελέτη θεμελίωσης εναλλακτικών διαδρομών μέσω BGP και πως επιλέγεται τελικά από ένα σύνολο διαθέσιμων διαδρομών με τη χρήση κάποιων κριτηρίων




```
"""
Example topology of Quagga routers
"""

import inspect
import os
from mininext.topo import Topo
from mininext.services.quagga import QuaggaService

net = None



class QuaggaTopo(Topo) :
```

```
"Creates a topology of Quagga routers"

def __init__(self) :


    Topo.__init__(self)

    # Διαδρομή που βρίσκεται το αρχείο"
    selfPath = os.path.dirname(os.path.abspath(
        inspect.getfile(inspect.currentframe() ) ) )   # script directory

    # Η υπηρεσία quagga
    quaggaSvc = QuaggaService(autoStop=False)

    #  #Η τοποθεσία που βρίσκονται τα αρχεία παραμετροποίησης για κάθε κόμβο
    quaggaBaseConfigPath = selfPath + '/configs/'

    # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    PC1=self.addHost(name='PC1',
                     hostname='PC1',
                     privateLogDir=True,
                     privateRunDir=True,
                     inMountNamespace=True,
                     inPIDNamespace=True,
                     inUTSNamespace=True)
# Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
          {'quaggaConfigPath': quaggaBaseConfigPath + 'PC1'}
    self.addNodeService(node='PC1',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)

    # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    R1  = self.addHost(name='R1',
                               hostname='R1',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)
       # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
          {'quaggaConfigPath': quaggaBaseConfigPath + 'R1'}
    self.addNodeService(node='R1',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)

       # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    R2  = self.addHost(name='R2',
                               hostname='R2',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)
       #Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
          {'quaggaConfigPath': quaggaBaseConfigPath + 'R2'}
    self.addNodeService(node='R2',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)
```

```
    # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    R3  = self.addHost(name='R3',
                               hostname='R3',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)

       # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
           {'quaggaConfigPath': quaggaBaseConfigPath + 'R3'}
    self.addNodeService(node='R3',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)

                    # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    C1  = self.addHost(name='C1',
                               hostname='C1',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)


       # Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
           {'quaggaConfigPath': quaggaBaseConfigPath + 'C1'}
    self.addNodeService(node='C1',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)

                # Δημιουργώ ένα νέο κόμβο και τον εισάγω στο δίκτυο μου
    PC2 = self.addHost(name='PC2',
                               hostname='PC2',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)

       #Αρχικοποιώ την υπηρεσία guagga σε αυτό το κόμβο και της λέω που να πάει να βρει τα αρχεία παραμετροποίησης του
    quaggaSvcConfig = \
           {'quaggaConfigPath': quaggaBaseConfigPath + 'PC2'}
    self.addNodeService(node='PC2',  service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)

           # Δημιουργώ τις συνδέσεις
    self.addcustlink(PC1,  R1)
    self.addcustlink(R1,  C1)
    self.addcustlink(R1,  R2)
    self.addcustlink(R1,  R3)
    self.addcustlink(C1,  R3)
    self.addcustlink(R2,  R3)
    self.addcustlink(R3,  PC2)

def addcustlink(self,  pc1,  pc2) :
    sw=self.addSwitch(pc1+'-'+pc2)
    self.addLink(pc1,  sw)
    self.addLink(sw,  pc2)
```

6 *ΕΛΕΓΧΟΣ*

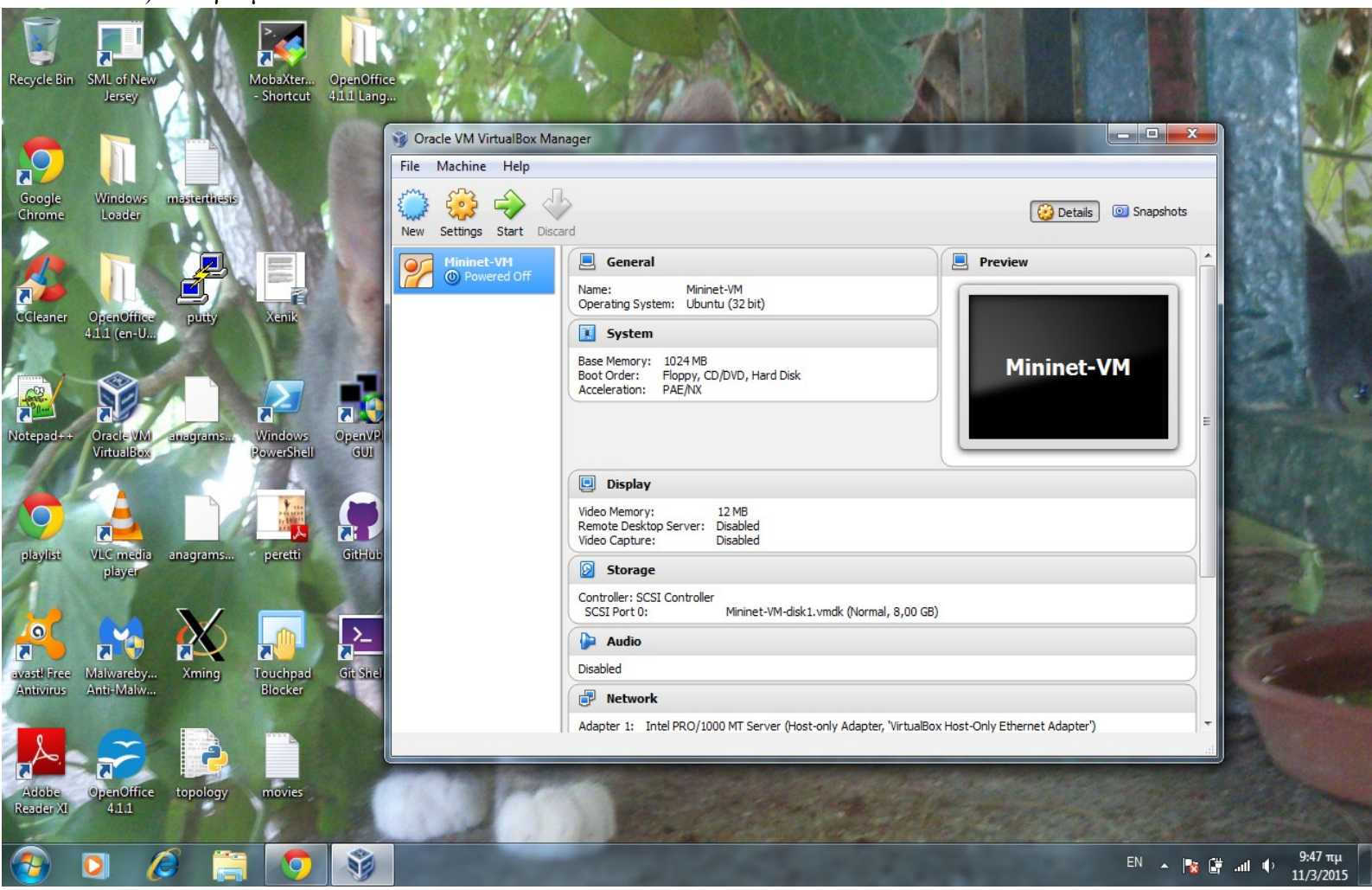
Σε αυτό το κεφάλαιο εξετάζουμε την ορθή λειτουργία του κώδικα μας μέσω ενός σεναρίου χρήσης. Η ορθότητα επιβεβαιώνεται μέσω της παρατήρησης των φυσιολογικών διεργασιών που λαμβάνουν χώρα στο δίκτυο που δημιουργούμε. Επίσης το κεφάλαιο αυτό λειτουργεί και σαν ένα πλήρες παράδειγμα χρήσης του κώδικα και μπορεί να εμπλουτιστεί, επεκταθεί ανάλογα με τις ανάγκες του αναγνώστη. Θα χρησιμοποιήσουμε το πρωτόκολλο RIP. Την περιγραφή της εκτέλεσης θα πλαισιώνουν αναλυτικές φωτογραφίες για καλύτερη εποπτεία αλλά και να μπορέσουμε να εξηγήσουμε κάποια στοιχειώδη κομμάτια που κρίνεται απαραίτητο να καταλάβει ο αναγνώστης. Η εκτέλεση θα ξεκινήσει στο mininet όπου και θα καταδειχθεί η αναποτελεσματική περάτωση του σεναρίου λόγω έλλειψης extra isolation, συνεχίζοντας με τη χρήση miniNexT που προσφέρει τα επιπλέον χαρακτηριστικά που έλειπαν. Τέλος θα δώσουμε έμφαση στο πρωτόκολλο openflow μέσω μίας πληθώρας εντολών σε όλα τα επίπεδα ενός δικτύου, παρέχοντας παραδείγματα για την σύνταξη των εκάστοτε εντολών.Όλοι οι κώδικες που απαιτούνται βρίσκονται στο παράρτημα με τη σειρά αναφοράς.

6.1 Μεθοδολογία έλεγχου

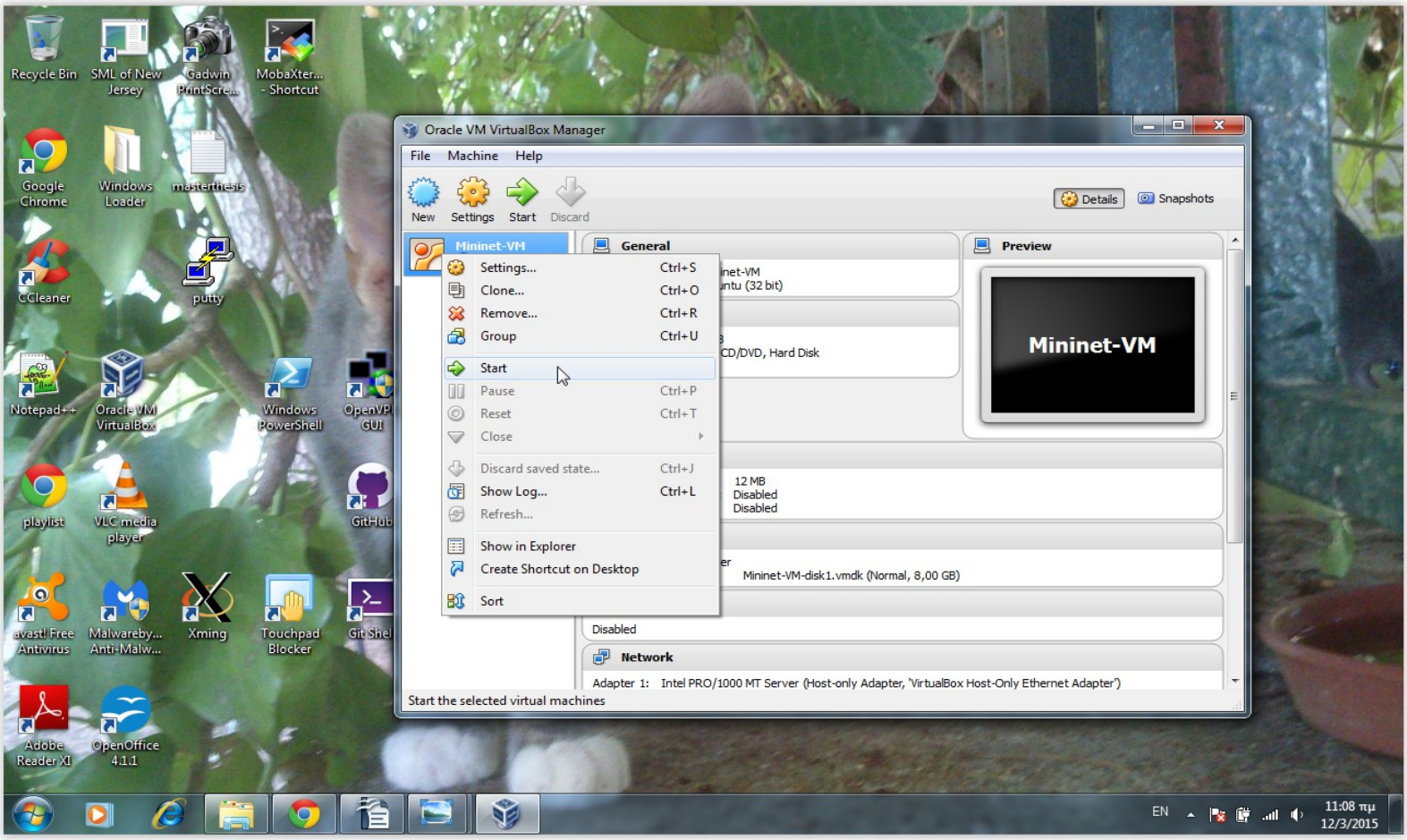
Το σενάριο χρήσης έχει ως εξής: Επιθυμούμε να εξετάσουμε τις πιθανές ενέργειες που πρέπει να προηγηθούν σε μία τοπολογία ώστε να υπάρξει επικοινωνία κάθε υπολογιστή με όλους τους υπόλοιπους. Το πρωτόκολλο δρομολόγησης που θα χρησιμοποιηθεί είναι το RIP.
Στις ενέργειες αυτές συμπεριλαμβάνεται και ρύθμιση των δρομολογητών ώστε να ακολουθούν τους κανόνες λειτουργίας του πρωτοκόλλου. Εν συνεχεία θα αναζητήσουμε το σύνολο των κανόνων που πρέπει να εισαχθούν σε ένα μεταγωγέα openflow για να επιτραπεί μία ορισμένη κίνηση πακέτων σε ένα δίκτυο υπολογιστών και να εμποδιστούν άλλες.

6.2 Αναλυτική παρουσίαση ελέγχου

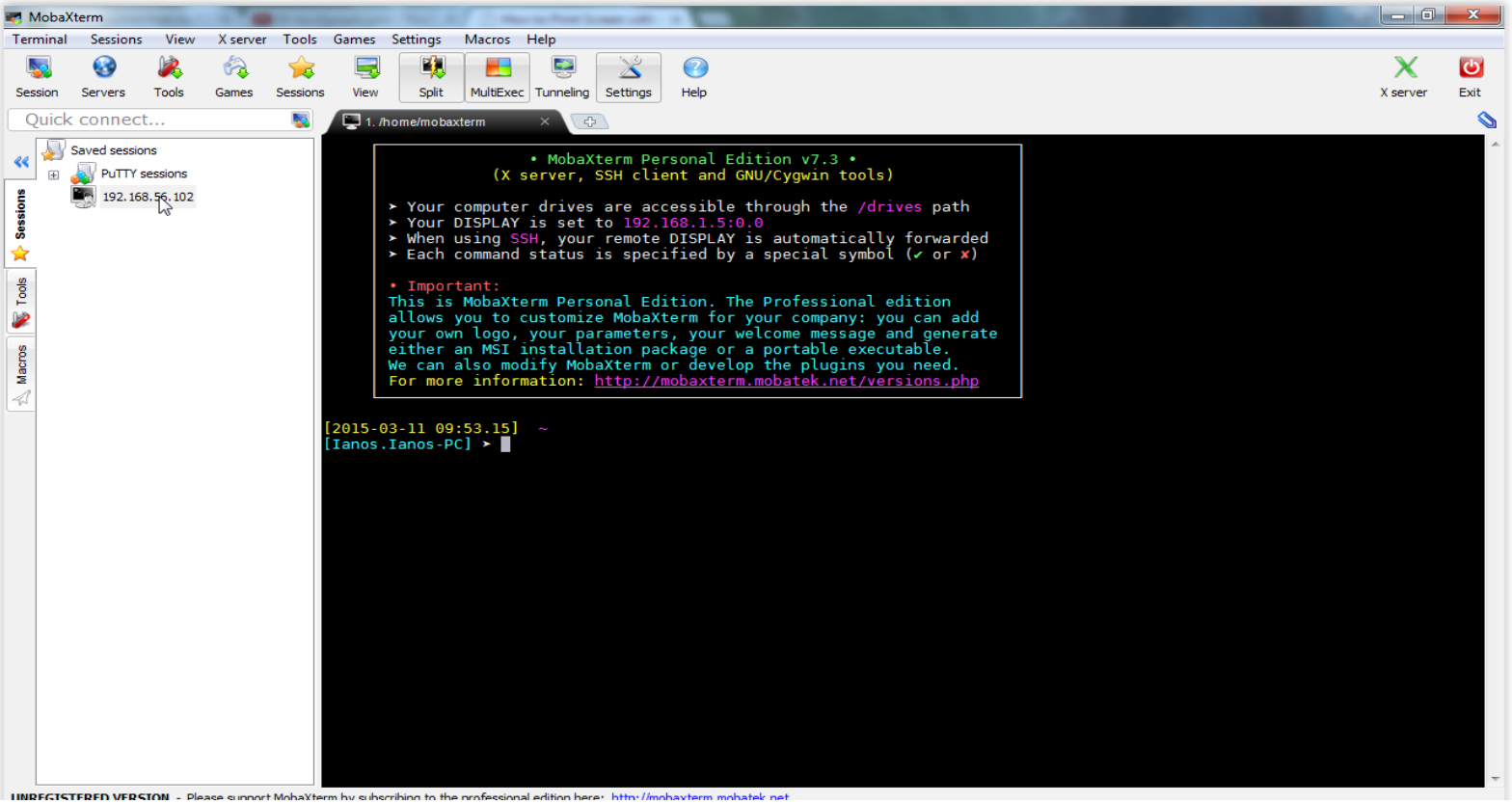
1) Ανοίγουμε το Virtual Box



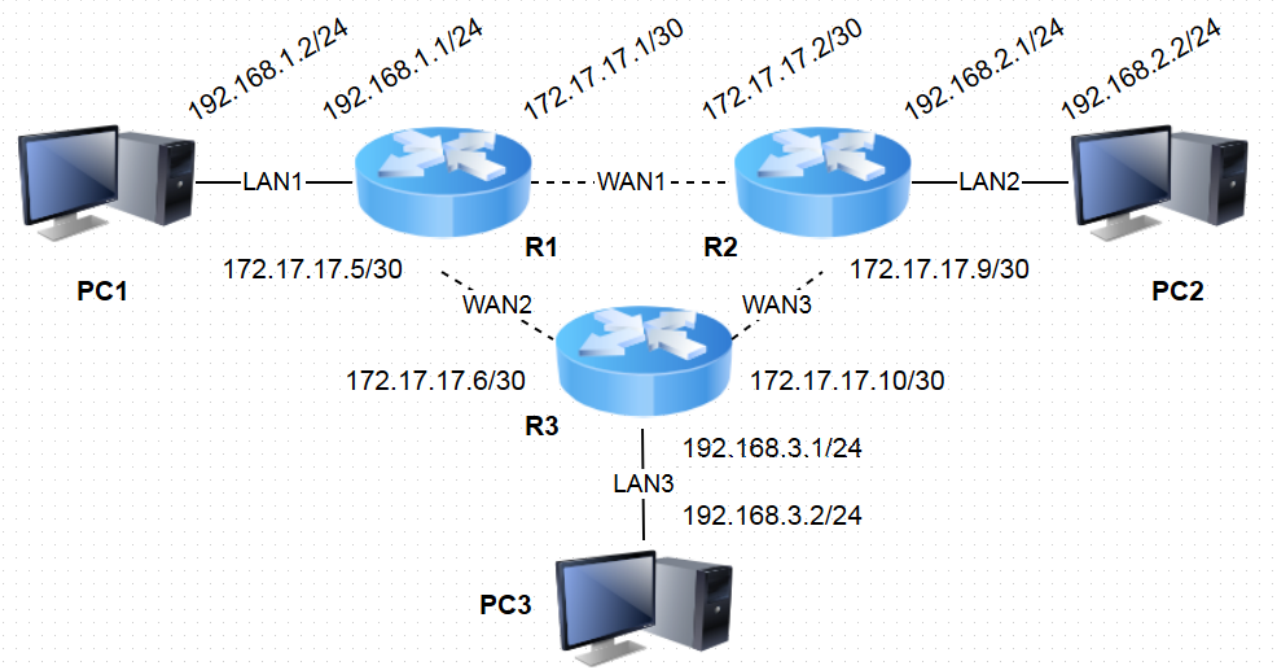
Στη συνέχεια σηκώνουμε το εικονικό μηχάνημα που έχει εγκατεστημένο το MININET.
Συνδεόμαστε σε αυτό με κάποιο SSH client (Putty ή κάποιο άλλο, πχ MobaXterm) .
Σε αυτή τη παρουσίαση χρησιμοποιήσαμε το MobaXterm γιατί προσφέρει γραφική απεικόνιση του συστήματος αρχείων και έτσι διευκολύνεται η αντιγραφή και αντικατάσταση.
Υπενθυμίζουμε πως αυτό έγινε καθαρά και μόνο για την οικονομία του χρόνου, σε κάθε περίπτωση συστήνουμε στον αναγνώστη να ακολουθήσει την πεπατημένη για να εξοικειωθεί και με τις εντολές του λειτουργικού συστήματος Ubuntu.



Συνδεόμαστε με SSH στο μηχάνημα, του οποίου η IP είναι 192.168.56.102.



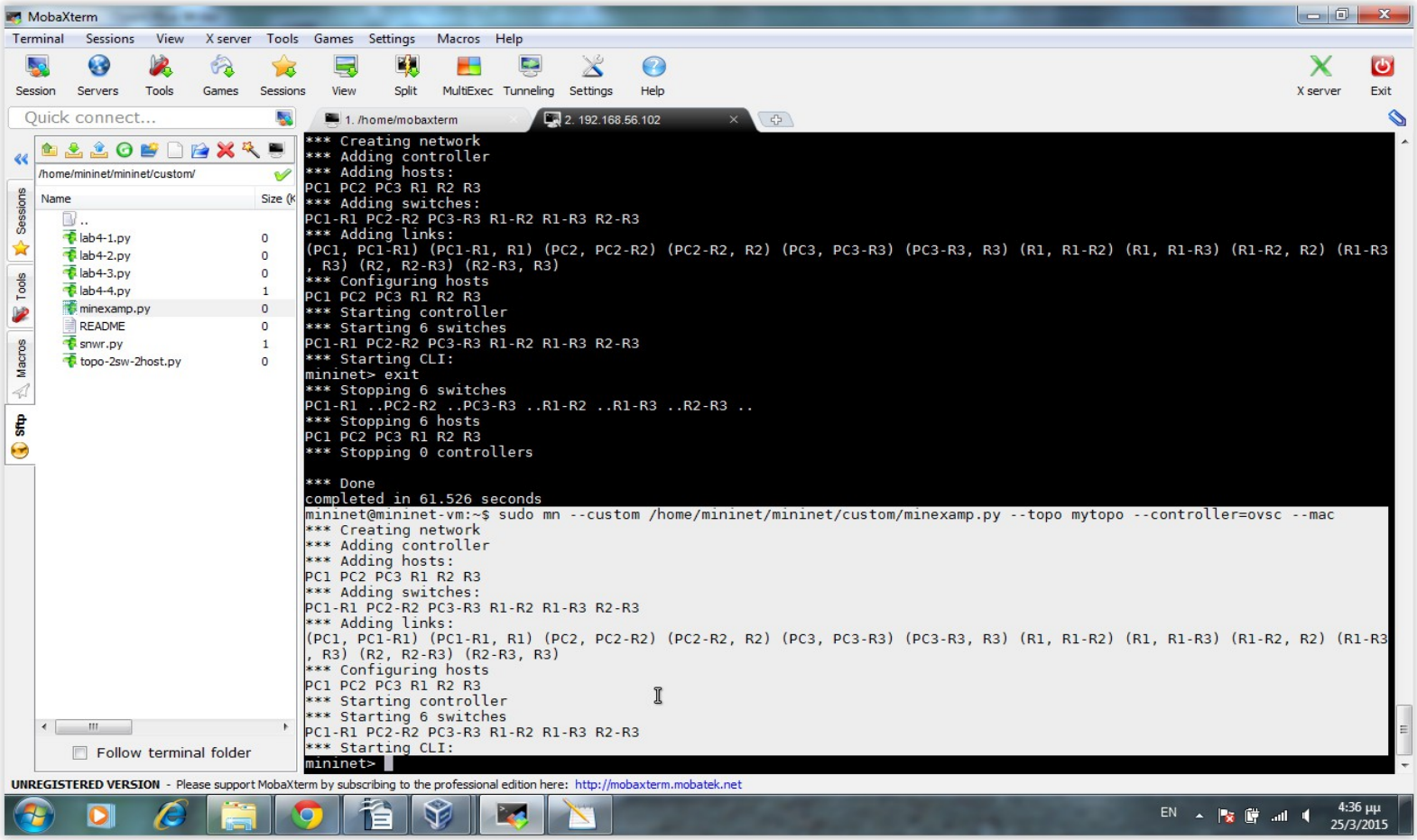
Η τοπολογία που θα δημιουργήσουμε είναι η παρακάτω:



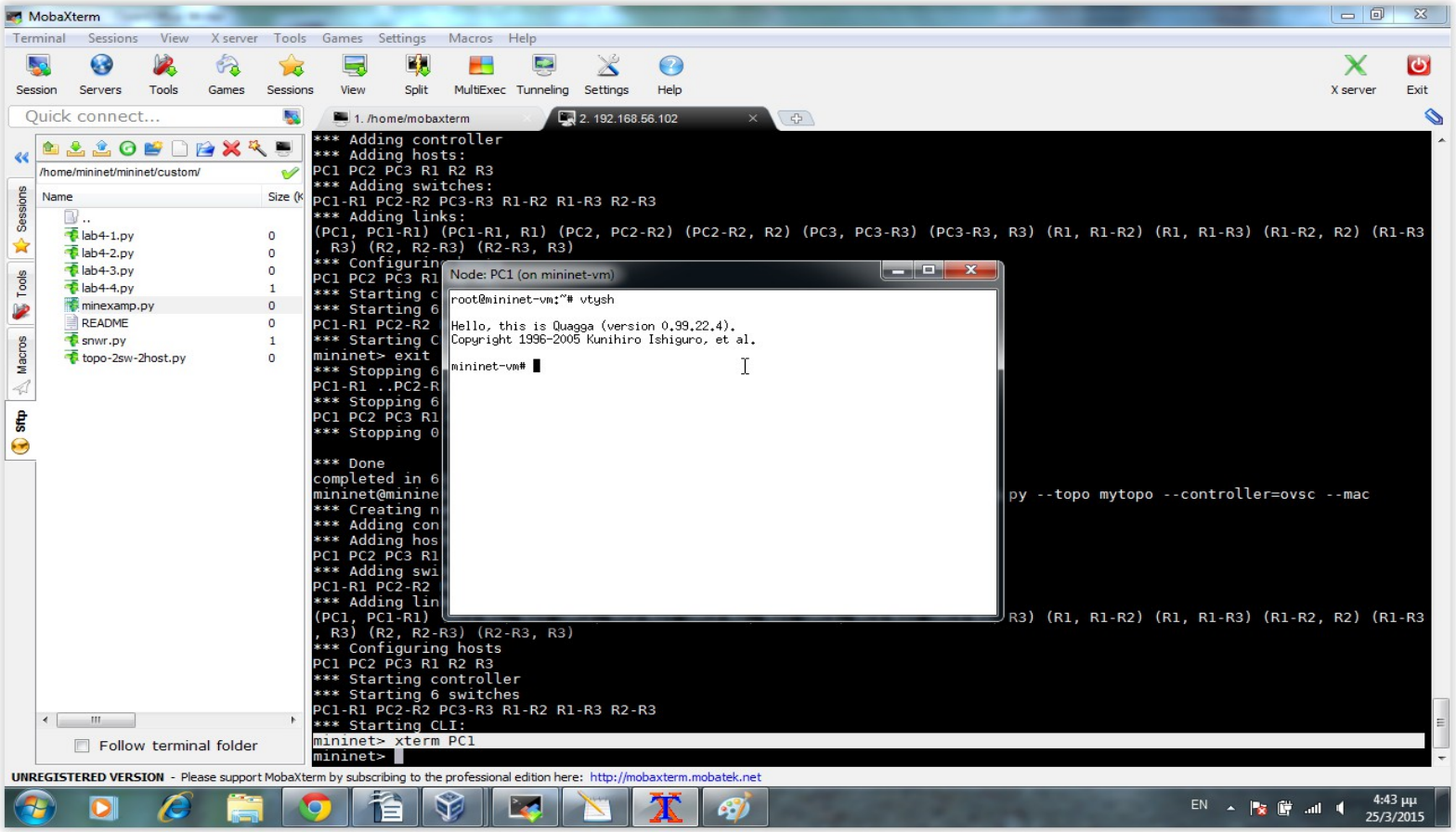
Ο κώδικας της τοπολογίας βρίσκεται στο αρχείο minexamp.py και τον τρέχουμε με την παρακάτω εντολή

sudo mn --custom /home/mininet/mininet/custom/minexamp.py --topo mytopo --controller=ovsc --mac

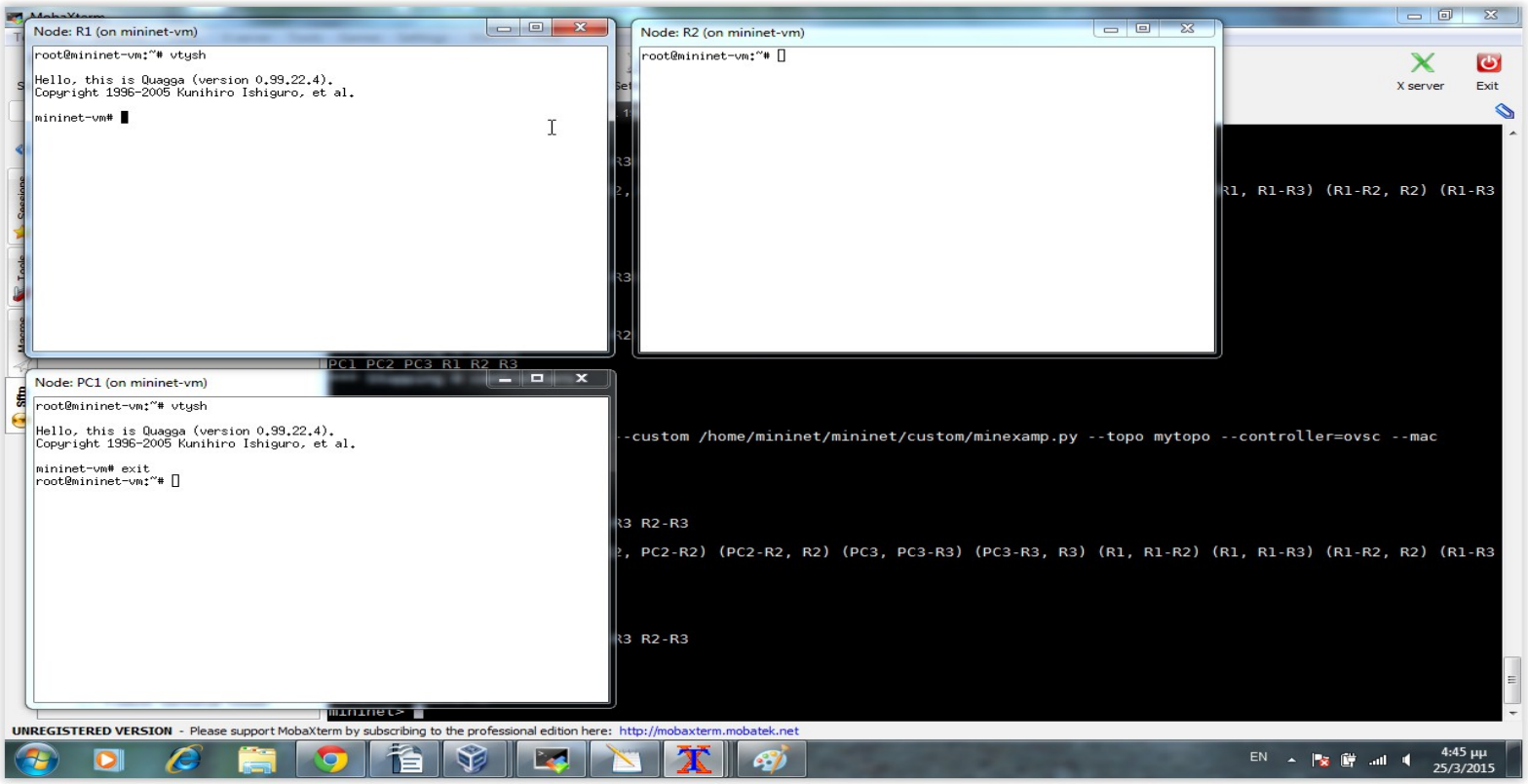
Και παρατηρούμε



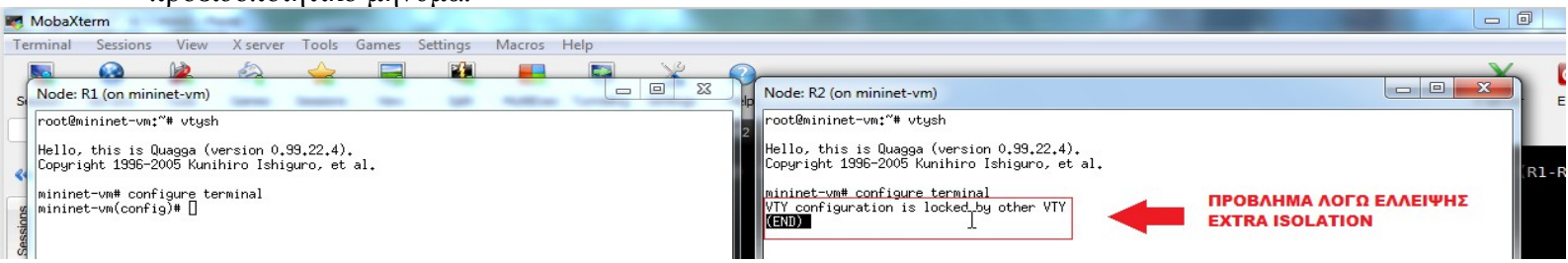
Δοκιμάζουμε το xterm(Προσοχή να έχουμε σηκώσει το Xming προτού προχωρήσουμε) .
Εκτελούμε την εντολή xterm PC1 και βλέπουμε ότι ανοίγει ένα νέο παράθυρο για το τερματικό PC1



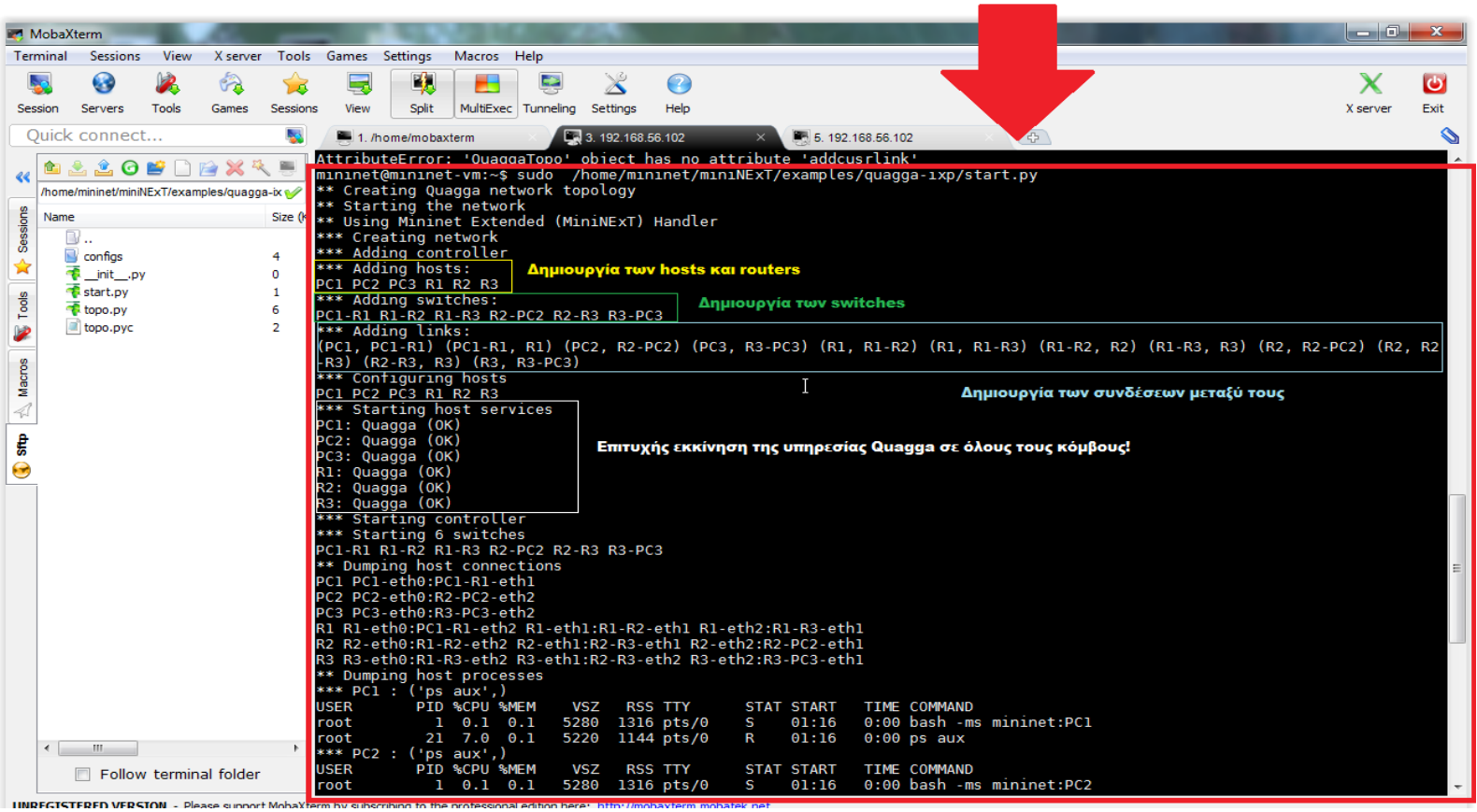
Κάνω το ίδιο με τα δρομολογητές R1 και R2 και εισέρχομαι στο quaqqa με την εντολή vtysh



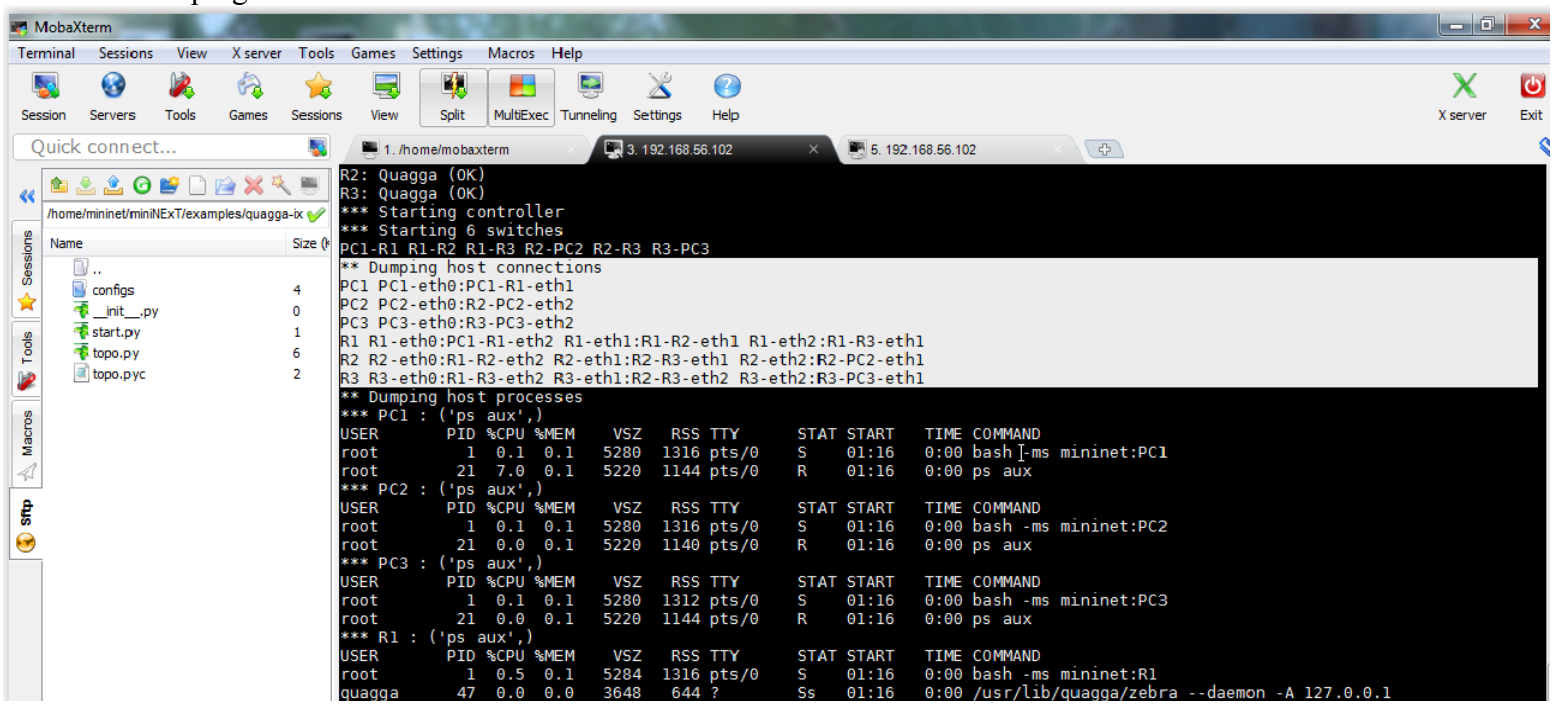
Στη συνέχεια εισέρχομαι στο διαχειριστικό περιβάλλον quagga των R1 και R2 με την εντολή configure terminal. Κατά την εκτέλεση της εντολής στο δεύτερο παρατηρώ το εξής προειδοποιητικό μήνυμα.



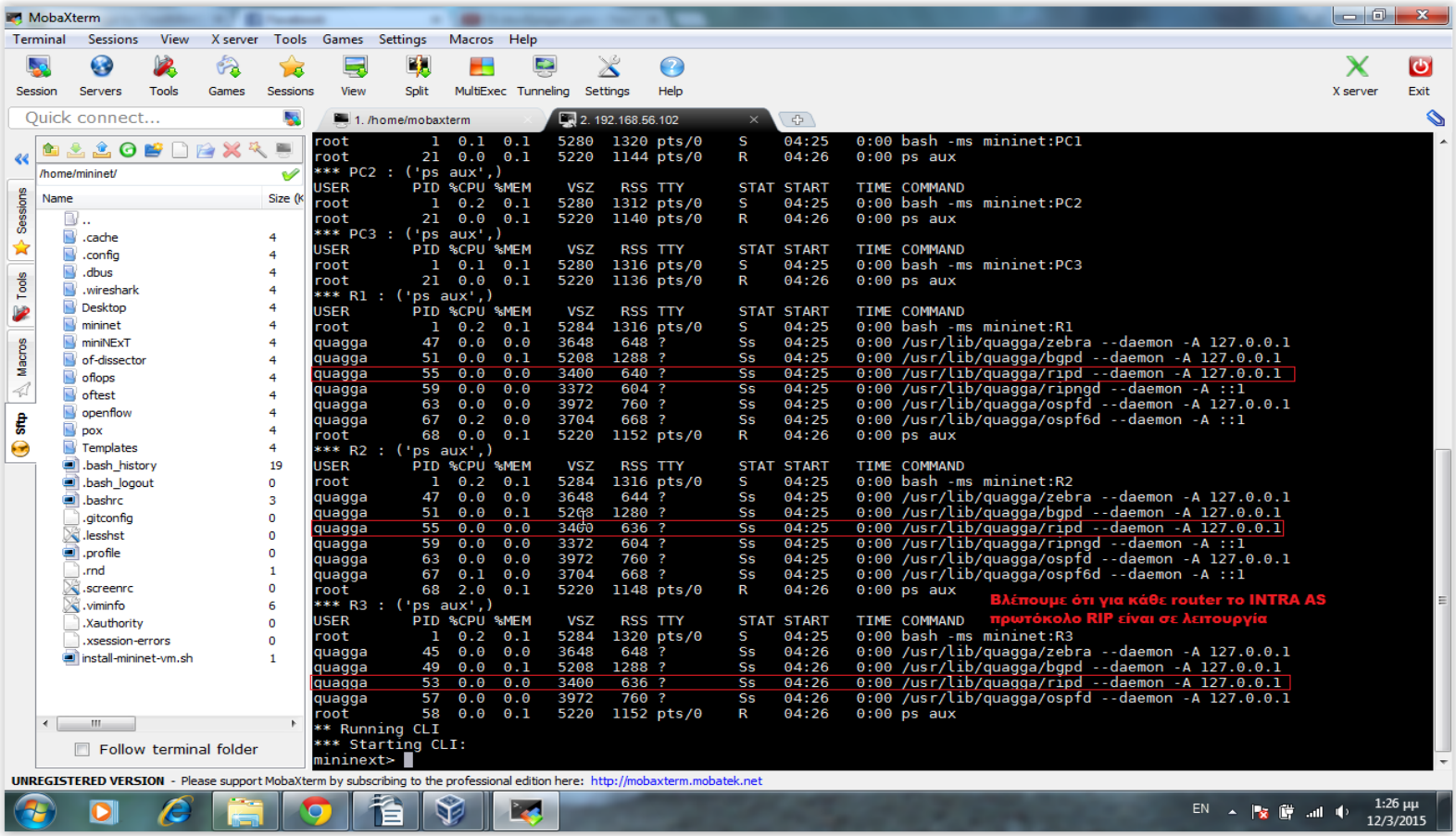
Το μήνυμα αυτό καταδεικνύει με απλό και σαφή τρόπο γιατί το σενάριο εκτέλεσης μας δεν μπορεί να προχωρήσει. Όπως φαίνεται και στην εικόνα τα δύο quagga σε διαφορετικούς hosts βλέπουνε το ίδια σύστημα αρχείων οπότε και όταν το ένα επιχειρεί να τροποποιήσει το αρχείο που ήδη τροποποιεί το άλλο προκύπτει αυτό το conflict. Επομένως κάτω από αυτές τις συνθήκες η περάτωση του σεναρίου μας είναι αδύνατη χωρίς την προσθήκη extra isolation ώστε το κάθε τερματικό να έχει το δικό του σύστημα αρχείων για το quagga. Το επιπλέον χαρακτηριστικό που αναζητούμε μας προσφέρει το miniNExT. Τερματίζουμε την εκτέλεση του mininet εν συνεχεία επιλέγουμε τον κώδικα της τοπολογίας για miniNExT ο οποίος βρίσκεται στο αρχείο lab6-3.py. Ανάμεσα σε κάθε σύνδεση παρεμβάλλουμε ένα switch για τις ανάγκες του capture με Wireshark. Μεταβαίνουμε στη φάκελο "/home/mininet/miniNExT/examples/quagga-ixp/topo.py" και αντικαθιστούμε τον κώδικα του topo.py με τον επιθυμητό (ΠΑΡΑΡΤΗΜΑ Α) .
Εν συνεχεία τρέχουμε την εντολή sudo /home/mininet/miniNExT/examples/quagga-ixp/start.py και βλέπουμε το παρακάτω



Αναγνωρίζουμε τα άκρα (ποία είναι η R1-eth2 κλπ) με βάση τη πληροφορία που περιέχεται στο dumping host connections



Βλέπουμε πχ ότι ο R1 συνδέεται με το PC1 μέσω της διεπαφής (R1-eth0) που συνδέεται με το switch PC1-R1 που έχει δύο εισόδους(eth1 για το PC1, eth2 για το PC2) .
Επίσης βλέπουμε ότι το πρωτόκολλο RIP έχει ενεργοποιηθεί στους δρομολογητές R1, R2, R3.Σε περίπτωση που ο ορισμός των αρχείων debian.conf, zebra.conf, ripd.conf δεν έχει προηγηθεί ή δεν είναι σωστός οι γραμμές αυτές θα απουσιάζουν



Στη συνέχεια αρχίζουμε να χτίζουμε εκ του μηδενός την διευθυνσιοδότηση της τοπολογίας με βάση το σχήμα μας. Αναλυτικά εκτελούμε τις παρακάτω εντολές.
Για το PC1:
py PC1.intf('PC1-eth0').setIP('192.168.1.2/24')
Για το PC2:
py PC2.intf('PC2-eth0').setIP('192.168.2.2/24')
Για το PC3:
py PC3.intf('PC3-eth0').setIP('192.168.3.2/24')
Για το R1
py R1.intf('R1-eth0').setIP('192.168.1.1/24')
py R1.intf('R1-eth1').setIP('172.17.17.1/30')
py R1.intf('R1-eth2').setIP('172.17.17.5/30')
Για το R2
py R2.intf('R2-eth0').setIP('172.17.17.2/30')
py R2.intf('R2-eth1').setIP('172.17.17.9/30')
py R2.intf('R2-eth2').setIP('192.168.2.1/24')
Για το R3
py R3.intf('R3-eth0').setIP('172.17.17.6/30')
py R3.intf('R3-eth1').setIP('172.17.17.10/30')
py R3.intf('R3-eth2').setIP('192.168.3.1/24')
Συνίσταται στον αναγνώστη να επαληθεύσει την όντως σωστή ανάθεση διευθύνσεων με την εντολή <node-name> ifconfig για την αποφυγή προβλημάτων στη συνέχεια.
Έπειτα θα ορίσουμε προεπιλεγμένη πύλη σε κάθε PC.
PC1 route add default gw 192.168.1.1
PC2 route add default gw 192.168.2.1
PC3 route add default gw 192.168.3.1

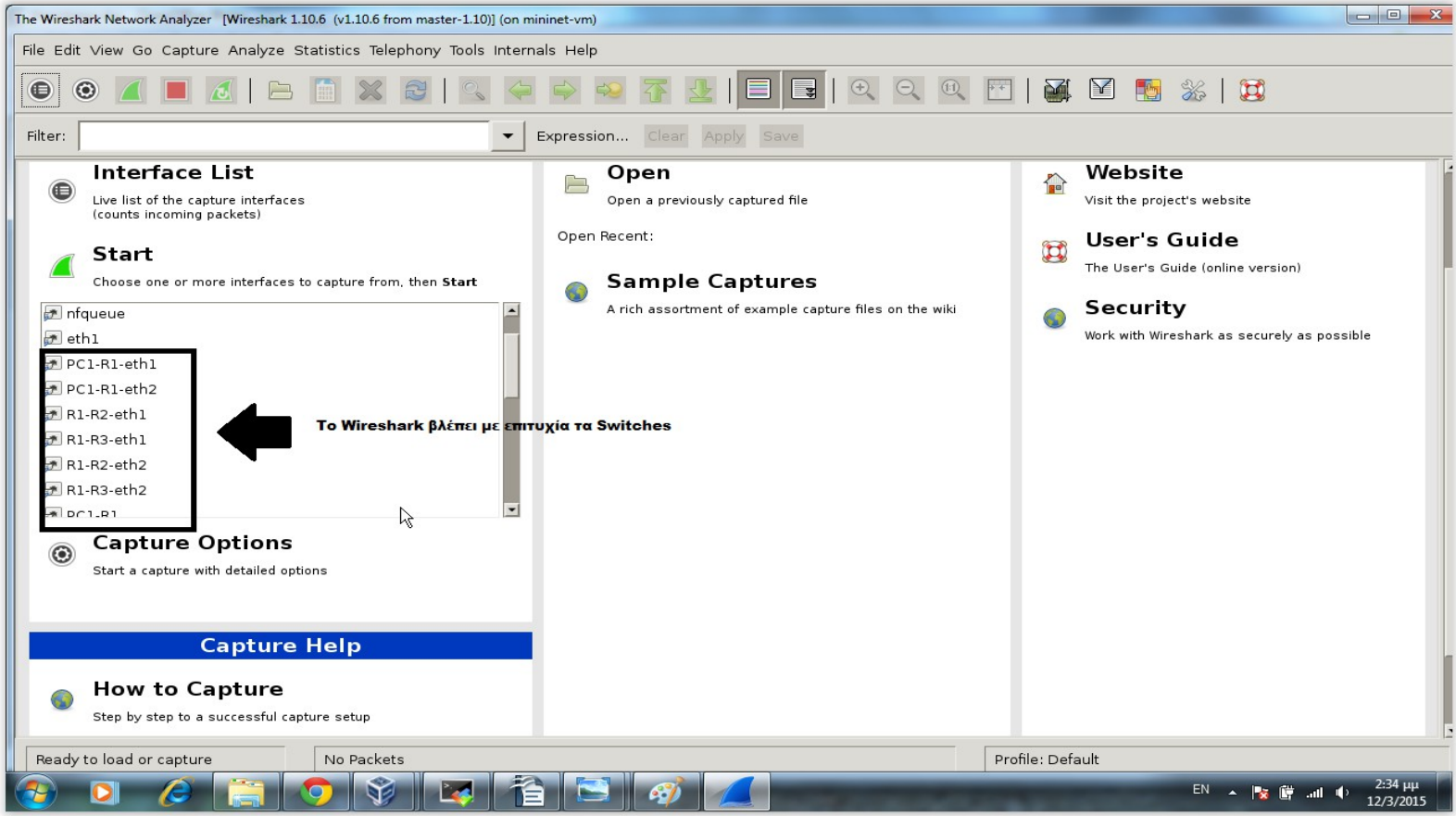
Για να υπάρχει αποτελεσματική επικοινωνία μεταξύ των PC πρέπει όπως γνωρίζουμε από τη θεωρία κάθε υπολογιστής(R1, R2, R3) να λειτουργήσει ως δρομολογητής. Για το σκοπό αυτό πρέπει να ενεργοποιήσουμε τη λειτουργία της προώθησης πακέτων.

R1 sysctl -w net.ipv4.ip_forward=1
R2 sysctl -w net.ipv4.ip_forward=1
R3 sysctl -w net.ipv4.ip_forward=1

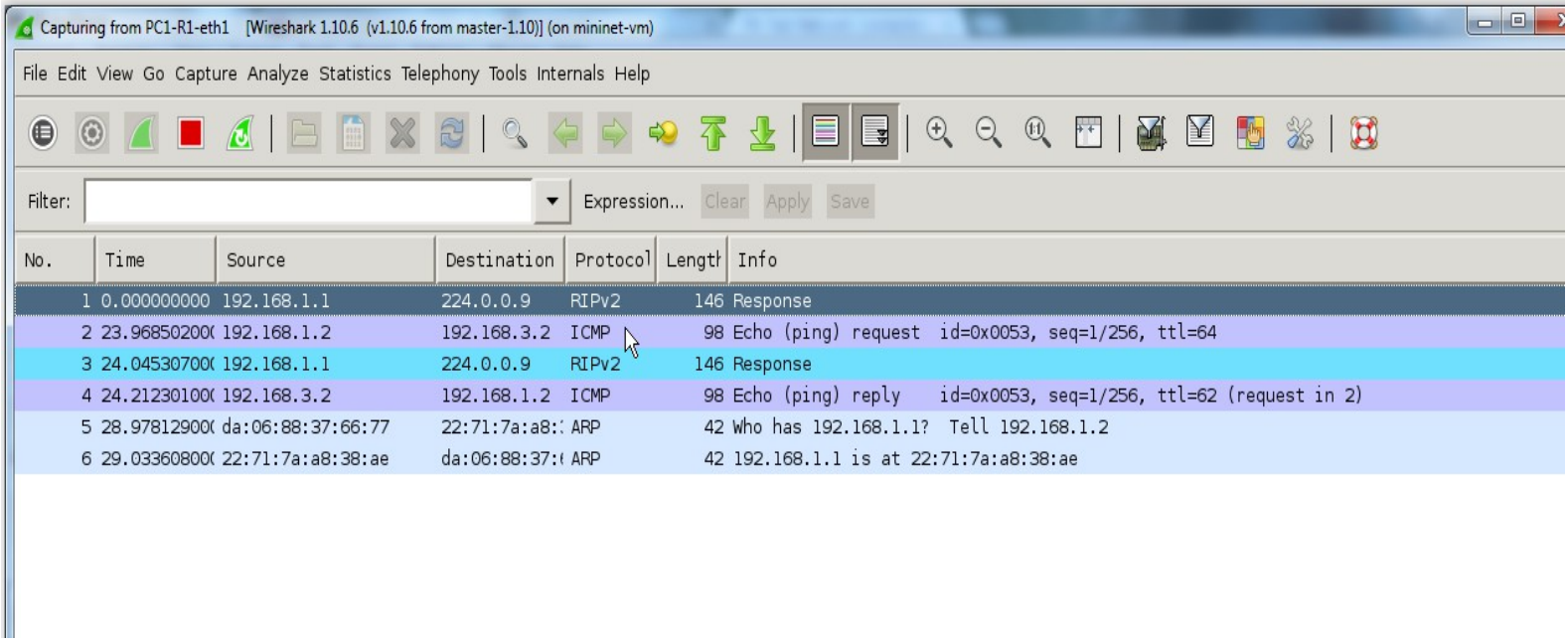
Τέλος για να μπορέσει ένας δρομολογητής να στείλει ένα πακέτο που δέχεται στη σωστή έξοδο, διεπαφή πρέπει να έχει στους πίνακες δρομολόγησής του καταχώρηση που να αφορά το υποδίκτυο που ανήκει η διεύθυνση προορισμού του πακέτου.
Το πρωτόκολλο RIP συμβάλλει σε αυτό διαφημίζοντας τα υποδίκτυα στα οποία συνδέεται ένας κόμβος, στους γειτονικούς κ.ο.κ. Εμπλουτίζονται με αυτό το τρόπο οι πίνακες δρομολόγησης των κόμβων του δικτύου.
Εκτελούμε τις παρακάτω εντολές
sh /home/mininet/miniNExT/util/mx R1 vtysh
configure terminal
router rip
network 192.168.1.0/24
network 172.17.17.4/30
network 172.17.17.0/30
exit
exit
exit
sh /home/mininet/miniNExT/util/mx R2 vtysh
configure terminal
router rip
network 192.168.2.0/24
network 172.17.17.0/30
network 172.17.17.8/30
exit
exit
exit
sh /home/mininet/miniNExT/util/mx R3 vtysh
configure terminal
router rip
network 192.168.3.0/24
network 172.17.17.4/30
network 172.17.17.8/30
exit
exit
exit
Εμφανίζοντας τον πίνακα δρομολόγησης ενός εκ των τριών δρομολογητών μπορούμε να δούμε ό, τι μαθαίνουν δυναμικά τα άλλα υποδίκτυα
Οι εγγραφές με το R είναι οι δυναμικές!

```
mininet-vm(config)# exit
mininet-vm# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

C>* 10.0.0.0/8 is directly connected, R3-eth0
C>* 127.0.0.0/8 is directly connected, lo
R>* 172.17.17.0/30 [120/2] via 172.17.17.5, R3-eth0, 00:00:41
C>* 172.17.17.4/30 is directly connected, R3-eth0
C>* 172.17.17.8/30 is directly connected, R3-eth1
R>* 192.168.1.0/24 [120/2] via 172.17.17.5, R3-eth0, 00:00:41
R>* 192.168.2.0/24 [120/2] via 172.17.17.9, R3-eth1, 00:00:38
C>* 192.168.3.0/24 is directly connected, R3-eth2
mininet-vm# exit
```

Τέλος για να δοκιμάσουμε και το Wireshark ανοίγουμε ένα δεύτερο ssh session στο μηχάνημα που είναι εγκατεστημένο το Mininet. Στη συνέχεια εκτελούμε την εντολή sudo wireshark.
Προσοχή, πρέπει πρώτα να σηκώσουμε τον Xming Server αλλιώς η προσπάθεια θα αποτύχει.
Στο παράθυρο του Wireshark βλέπουμε το παρακάτω.

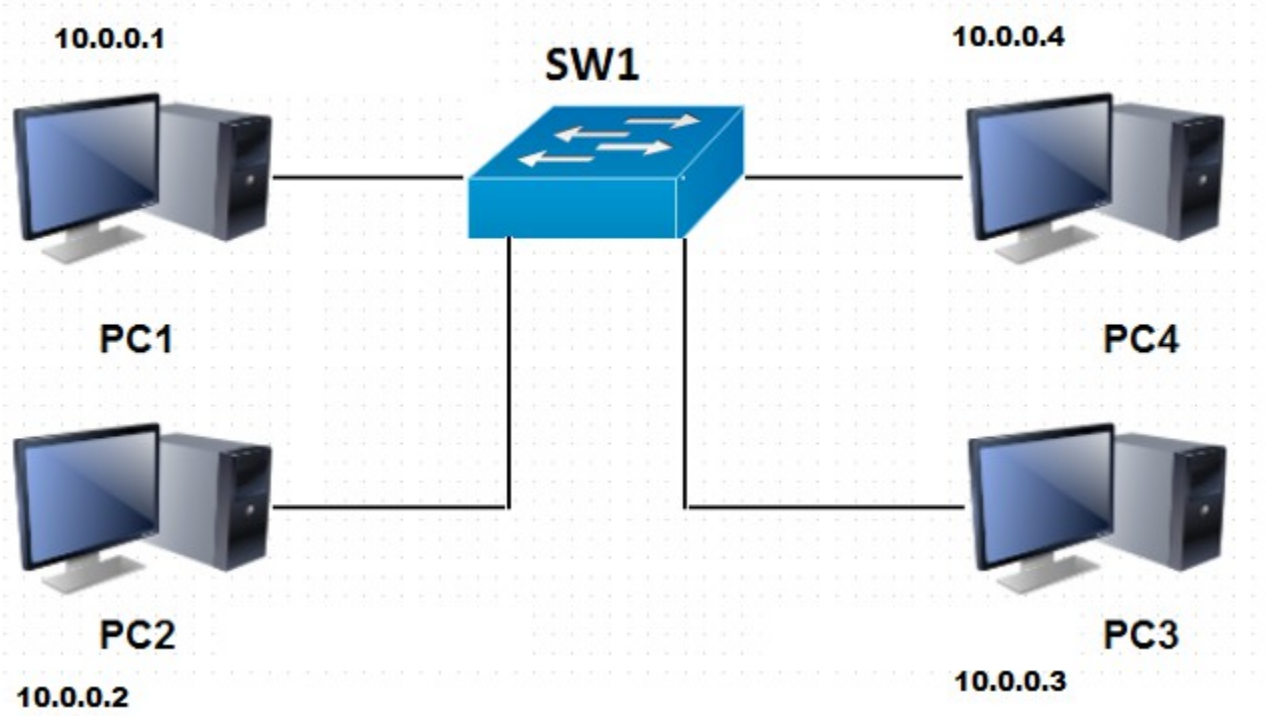


Ξεκινάμε μία καταγραφή στο PC1-R1 switch (PC1-R1-eth1 ή PC1-R1-eth2 δεν έχει σημασία)
Και στο αρχικό session μας που έχουμε το Mininet να τρέχει πληκτρολογούμε
PC1 ping -c 1 PC3
Στο Wireshark παρατηρούμε



Βλέπουμε λοιπόν πως ο κώδικάς μας λειτουργεί σωστά και δίνει τα επιθυμητά αποτελέσματα.

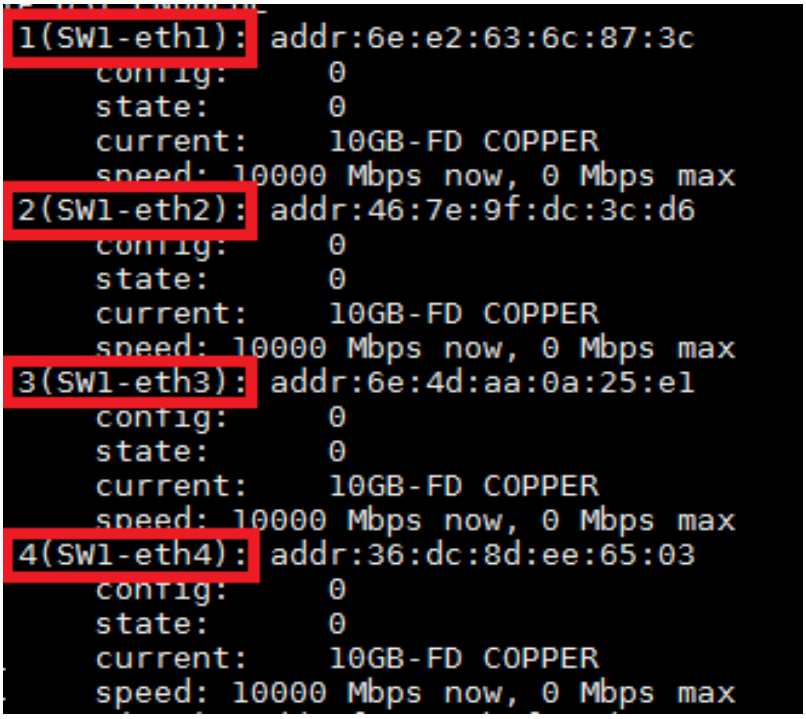
Στη συνέχεια θα δώσουμε μικρά παραδείγματα χρήσης του πρωτοκόλλου Openflow που υλοποιεί το mininet και αποτελεί το βασικό λόγο που επιλέξαμε το πρόγραμμα αυτό. Ο μεταγωγείς του mininet μπορούν να δεχτούν εντολές μέσω της εντολής ovs-ofctl που χρησιμοποιείται για τη χειροκίνητη εισαγωγή καταχωρήσεων στον πίνακα ροής τους. Η εισαγωγή των κανόνων αυτών μπορεί να γίνει και δυναμικά με τη χρήση controllers αλλά στο συγκεκριμένο παράδειγμα θα επικεντρωθούμε στην στατική εισαγωγή καθαρά και μόνο λόγω θέματος εποπτείας και καλύτερης κατανόησης.
Δημιουργούμε την παρακάτω τοπολογία που βρίσκεται αποθηκευμένη στο ΠΑΡΑΡΤΗΜΑ Β



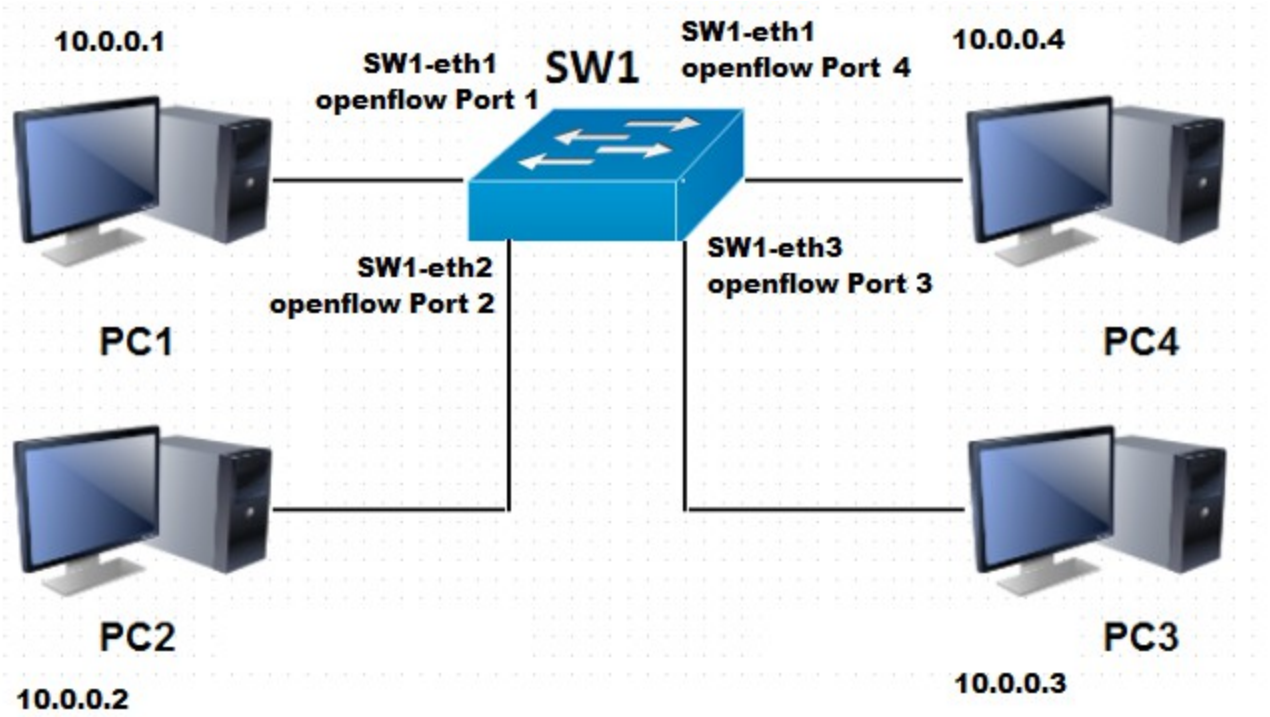
Με την εντολή sudo mn –custom/home/mininet/mininet/custom/snwr.py --topo mytopo --controller=none --mac, φορτώνουμε την τοπολογία στο mininet.
Η επιλογή --mac χρησιμεύει στο να αποδίδονται στα τερματικά εύκολες στη χρήση διευθύνσεις MAC.Τσεκάρουμε ότι η τοπολογία μας δημιουργήθηκε σωστά ελέγχοντας διευθύνσεις αλλά και συνδέσεις μέσω των εντολών dump, net και παρατηρούμε το παρακάτω

```
mininet> dump
<Host PC1: PC1-eth0:10.0.0.1 pid=1413>
<Host PC2: PC2-eth0:10.0.0.2 pid=1414>
<Host PC3: PC3-eth0:10.0.0.3 pid=1415>
<Host PC4: PC4-eth0:10.0.0.4 pid=1418>
<OVSSwitch SW1: lo:127.0.0.1,SW1-eth1:None,SW1-eth2:None,SW1-eth3:None,SW1-eth4:None pid=1421>
mininet> net
PC1 PC1-eth0:SW1-eth1
PC2 PC2-eth0:SW1-eth2
PC3 PC3-eth0:SW1-eth3
PC4 PC4-eth0:SW1-eth4
SW1 lo:  SW1-eth1:PC1-eth0 SW1-eth2:PC2-eth0 SW1-eth3:PC3-eth0 SW1-eth4:PC4-eth0
```

Στην παραπάνω τοπολογία βλέπουμε πως το σύστημα μας αναγνωρίζει τις διεπαφές ethernet του μεταγωγέα αλλά μας ενδιαφέρει επίσης πως εκείνες αντιστοιχίζονται στις θύρες openflow. Χρησιμοποιούμε για το σκοπό αυτό την εντολή sh ovs-ofctl show SW1



Το πρόθεμα sh μπαίνει στις εντολές που πρέπει να εκτελεστούν πάνω στο κέλυφος του φιλοξενούντος μηχανήματος και μας διευκολύνει στο να μη χρειάζεται να μεταπηδούμε κάθε φορά από το mininet στο διαχειριστικό περιβάλλον εντολών.
Και στην οθόνη μας βλέπουμε τις αντιστοιχίες. Επομένως η τοπολογία μας παίρνει την μορφή

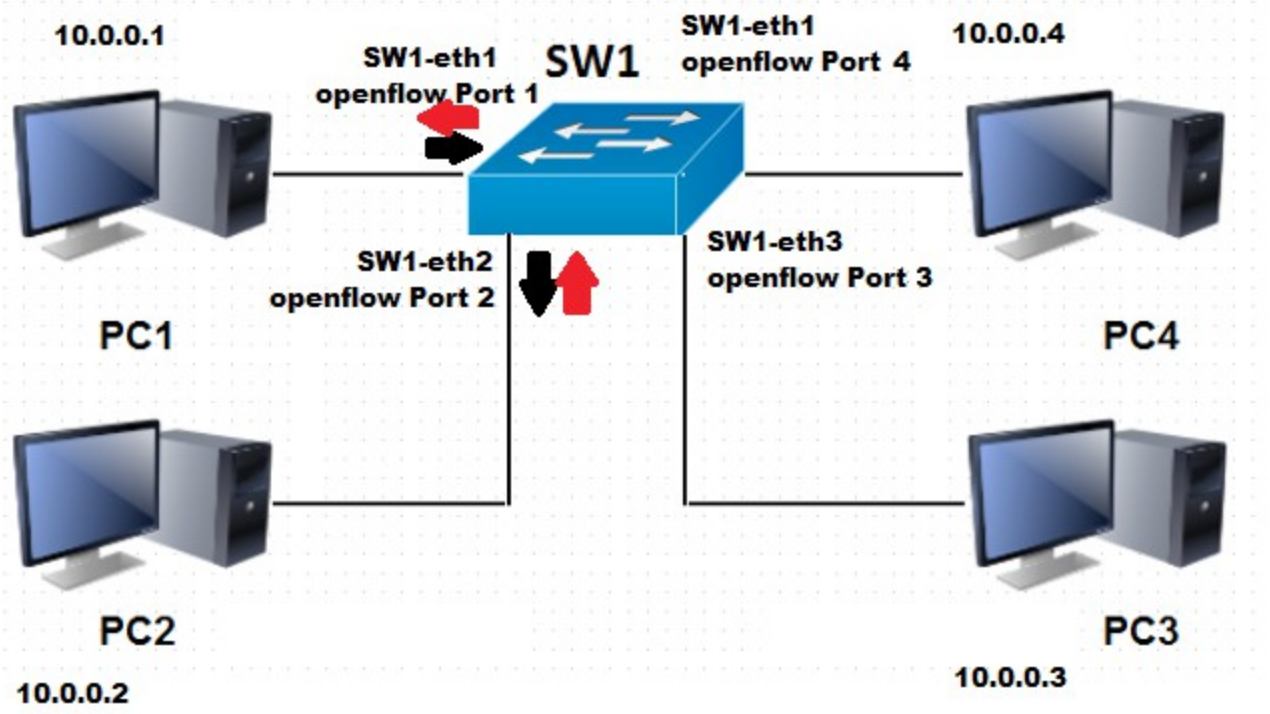


Πληκτρολογώντας pingall βλέπουμε πως δεν υπάρχει επικοινωνία και αυτό είναι απόλυτα φυσιολογικό.
Ξεκινάμε προσθέτοντας τη πρώτη μας καταχώρηση στο μεταγωγέα openflow
sh ovs-ofctl add-flow SW1 action=normal
Η επιλογή action=normal χρησιμοποιείται για να δώσουμε στο μεταγωγέα την φυσιολογική λειτουργία της προώθησης πακέτων σε επίπεδο διευθύνσεων MAC
Να σημειωθεί πως δεν έχουμε ορίσει κάποια συγκεκριμένα πεδία αναγνώρισης και έτσι όλα τα πακέτα γίνονται δεκτά.
Τεστάρουμε εν συνεχεία με pingall και βλέπουμε ότι υπάρχει πλήρης επικοινωνία των τερματικών
Για να δούμε οποιαδήποτε στιγμή τις καταχωρήσεις μας στο μεταγωγέα πληκτρολογούμε
sh ovs-ofctl dump-flows SW1

```
mininet> sh ovs-ofctl dump-flows SW1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=184.8s, table=0, n_packets=48, n_bytes=3360, idle_age=175, actions=NORMAL
mininet>
```

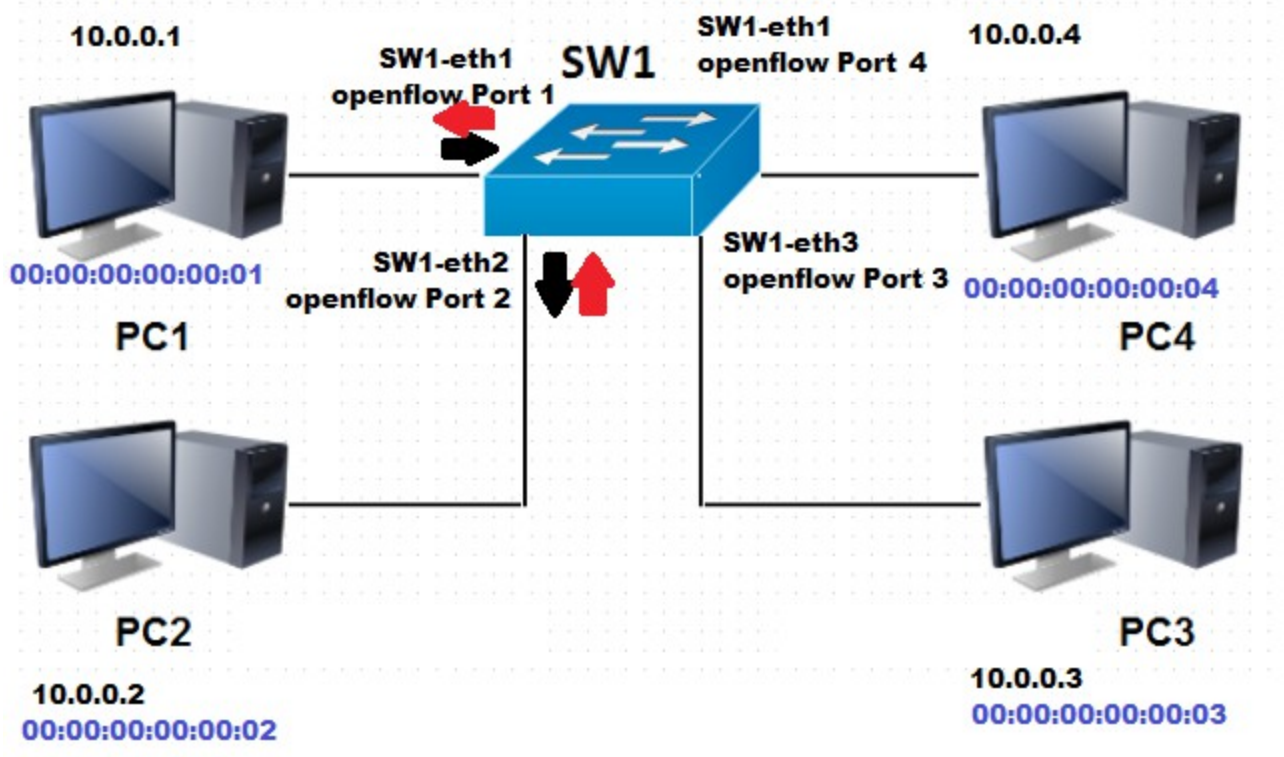
Ο μετρητής n_packets αφορά τον αριθμό των πακέτων που έχουν ενεργοποιήσει αυτόν το κανόνα.
Το duration αφορά τον χρόνο που έχει παρέλθει από τότε που δημιουργήθηκε αυτός ο κανόνας.
Το idle_age αφορά το χρόνο που έχει περάσει από τότε που ήρθε το τελευταία πακέτο που ενεργοποίησε το κανόνα. Ο μετρητής αυτός χρησιμοποιείται και για το καθαρισμό του πίνακα ροής απο καταχωρήσεις που έχουν παραμείνει ανενεργές για μεγάλο χρονικό διάστημα.
Εν συνεχεία καθαρίζουμε το πίνακα ροής του μεταγωγέα με την εντολή
sh ovs-ofctl del-flows SW1

Προχωρούμε τώρα σε ένα ταίριασμα επιπέδου 1 (Θυρών μεταγωγέα)
Στο παράδειγμά μας θα δώσουμε εντολή στο μεταγωγέα μας όποιο πακέτο λαμβάνει στη θύρα openflow 1 να το προωθεί στη θύρα openflow 2 και αντίστροφα όπως φαίνεται και στην παρακάτω εικόνα



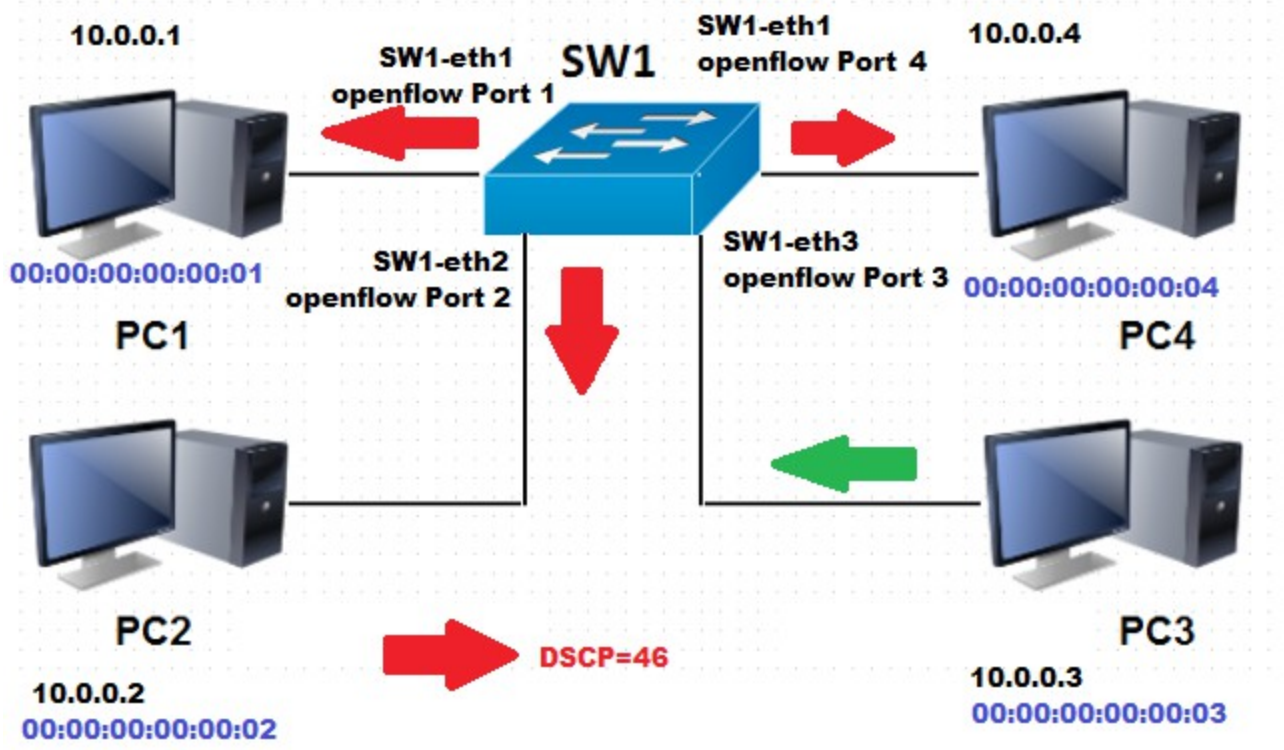
Για το σκοπό αυτό προσθέτουμε τις εξής καταχωρήσεις
sh ovs-ofctl add-flow SW1 priority=500, in_port=1, actions=output: 2
sh ovs-ofctl add-flow SW1 priority=500, in_port=2, actions=output: 1
Παρατηρούμε ότι σε κάθε κανόνα το αντίστοιχο κριτήριο ταιριάσματος είναι η θύρα openflow εισόδου και η αντίστοιχη ενέργεια που εκτελείται είναι η προώθηση του πακέτου στην επιλεγμένη θύρα εξόδου.
Μπορούμε να ελέγξουμε την επικοινωνία μεταξύ των PC1 και PC2 αλλά και την αδυναμία των PC3 και PC4 να επικοινωνήσουν με όλους τους άλλους αφού δεν υπάρχει κάποιος κανόνας και τα πακέτα τους απορρίπτονται μετά την άφιξή τους και τον έλεγχο στο μεταγωγέα
Η επιλογή priority χρησιμοποιείται σε περίπτωση που ένα πακέτο ενεργοποιεί πάνω από έναν κανόνες λαμβάνεται υπόψιν και εκτελείται ο κανόνας με τη μεγαλύτερη προτεραιότητα.
Μπορούμε να τσεκάρουμε αυτό βάζοντας έναν κανόνα απόρριψης όλων των πακέτων με προτεραιότητα μεγαλύτερη του 500
sh ovs-ofctl add-flow SW1 priority=32768, actions=drop
Ο αριθμός 32768 έχει αξία γιατί όταν το συγκεκριμένο πεδίο δεν προσδιορίζεται σε ένα κανόνα είναι η default τιμή που χρησιμοποιείται.
Κάνοντας pingall βλέπουμε ότι δεν υπάρχει πλέον επικοινωνία καθώς ενεργοποιείται πλέον ο κανόνας της απόρριψης.
Αν εμφανίσουμε και πάλι τις καταχωρήσεις βλέπουμε ότι ο νέος κανόνας βρίσκεται κάτω κάτω και ότι η τιμή της προτεραιότητας του δεν αναγράφεται επειδή είναι η default τιμή. Διαγράφω τη συγκεκριμένη καταχώρηση με sh ovs-ofctl del-flows SW1 --strict και βλέπω πως επανέρχεται κανονικά η επικοινωνία. Στη συνέχεια σβήνουμε όλες τις καταχωρήσεις στο πίνακα ροής του μεταγωγέα.

Προχωρούμε σε ταίριασμα επιπέδου 2(Διευθύνσεις MAC)
Στο παράδειγμα μας θα υλοποιήσουμε κάτι αντίστοιχο με παραπάνω μόνο που αυτή τη φορά ο μεταγωγέας όλα τα πακέτα που έχουν διεύθυνση MAC αφετηρίας 00: 00: 00: 00: 00: 01 και διεύθυνση MAC προορισμού 00: 00: 00: 00: 00: 02 θα προωθούνται στη θύρα openflow 2 του μεταγωγέα και το αντίστροφο στη θύρα 1.

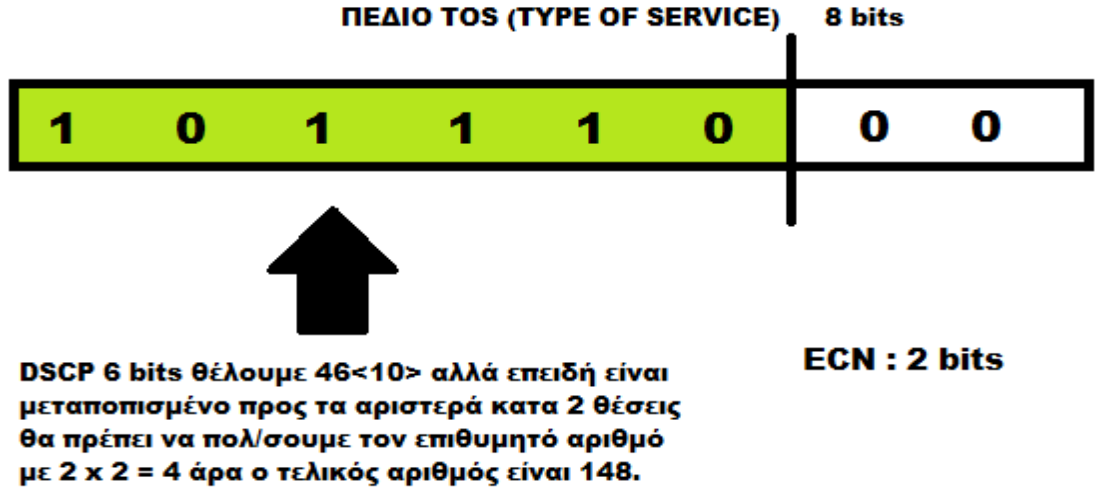


Εισάγω λοιπόν τις καταχωρήσεις
sh ovs-ofctl add-flow SW1 dl_src=00: 00: 00: 00: 00: 01, dl_dst=00: 00: 00: 00: 00: 02, actions=output: 2
sh ovs-ofctl add-flow SW1 dl_src=00: 00: 00: 00: 00: 02, dl_dst=00: 00: 00: 00: 00: 01, actions=output: 1
Επιχειρούμε την επικοινωνία και βλέπουμε ότι αποτυγχάνει. Ο λόγος είναι ότι τα δύο τερματικά δεν γνωρίζουν τη διεύθυνση MAC το ένα του άλλου.
Πρέπει να εισάγουμε επομένως μία καταχώρηση που επιτρέπει τη διέλευση των πακέτων ARP.
sh ovs-ofctl add-flow SW1 dl_type=0x806, nw_proto=1, actions=flood
Η επιλογή dl_type=0x806 είναι η τιμή που αναφέρεται στο πρωτόκολλο ARP. Το nw_proto=1 αναφέρεται σε πακέτα τύπου request και το actions=flood σημαίνει ότι ότι ο μεταγωγέας θα προωθήσει σε όλες τις θύρες openflow το πακέτο που μόλις έλαβε εξαιρουμένης της ληφθείσας θύρας
Προχωρούμε σε έλεγχο της επικοινωνίας και παρατηρούμε ότι τα τερματικά PC1, PC2 επικοινωνούν κανονικά μεταξύ τους
Εν συνεχεία διαγράφουμε όλες τις καταχωρήσεις στο πίνακα ροής του μεταγωγέα.

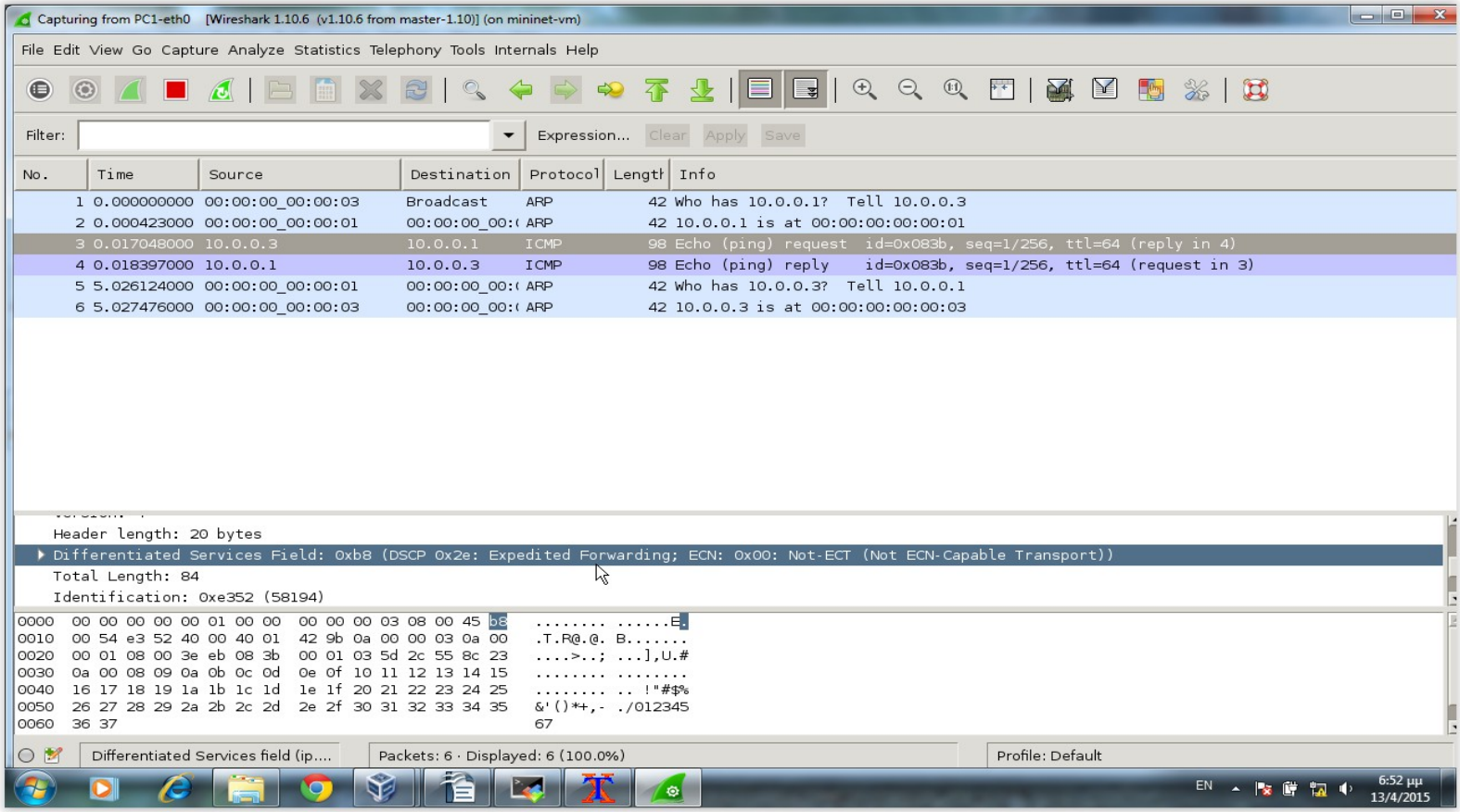
Προχωρούμε σε ταίριασμα επιπέδου 3(Διευθύνσεις IP)
Στο παράδειγμά μας θα επιτρέψουμε επικοινωνία μεταξύ όλων των υπολογιστών και θα αλλάξουμε όλες τις τιμές της παραμέτρου DSCP όλων των πακέτων που προέρχονται από το PC3 σε 46 για να καταδείξουμε ότι χαίρουν ειδικής προνομιακής προώθησης.



Για να επιτρέψουμε την πλήρη επικοινωνία όλων των τερματικών εισάγουμε την καταχώρηση
sh ovs-ofctl add-flow SW1 priority=500, dl_type=0x800, nw_src=10.0.0.0/24, nw_dst=10.0.0.0/24, actions=normal
Το πεδίο dl_type=0x800 αναφέρεται στο πρωτόκολλο IP. Και τα nw_src και nw_dst αναφέρονται στα υποδίκτυα που ανήκουν οι διευθύνσεις IP των τεσσάρων τερματικών.
Για να μεταβάλλουμε τις τιμές DCSP των πακέτων που προέρχονται από το PC3 εισάγουμε την καταχώρηση
sh ovs-ofctl add-flow SW1 priority=800, ip, nw_src=10.0.0.3, actions=mod_nw_tos: 184, normal

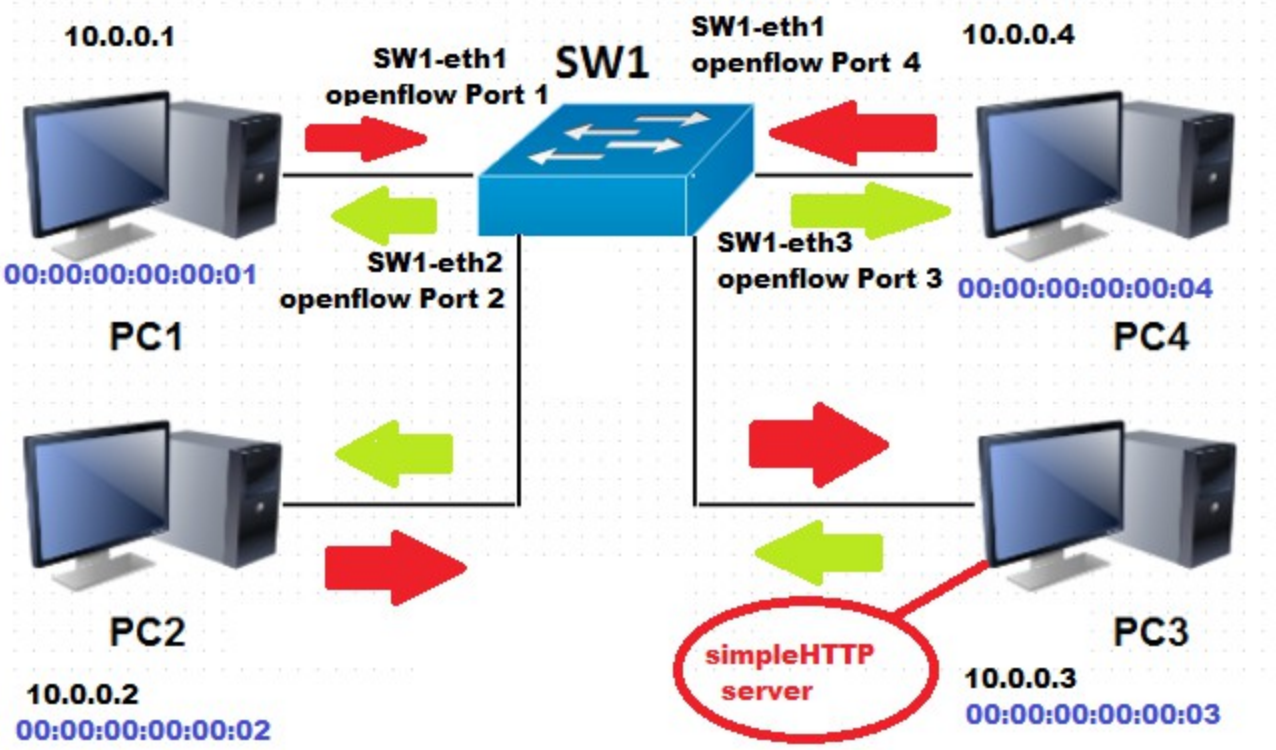


Δεν πρέπει να ξεχνάμε πως για να δουλέψει η επικοινωνία πρέπει να εισάγουμε κανόνες για το πρωτόκολλο ARP
sh ovs-ofctl add-flow SW1 arp, nw_dst=10.0.0.1, actions=output: 1
sh ovs-ofctl add-flow SW1 arp, nw_dst=10.0.0.2, actions=output: 2
sh ovs-ofctl add-flow SW1 arp, nw_dst=10.0.0.3, actions=output: 3
sh ovs-ofctl add-flow SW1 arp, nw_dst=10.0.0.4, actions=output: 4
Για να δούμε ότι δουλεύει η μετατροπή που επιθυμούμε ξεκινάμε ένα Wireshark στο PC1 και έπειτα πληκτρολογούμε
PC3 ping -c 1 PC1



Βλέπουμε ότι το αντίστοιχο πεδίο έχει μεταβληθεί κατά το δοκούν.
Εν συνεχεία διαγράφουμε όλους τους κανόνες από τον μεταγωγέα.

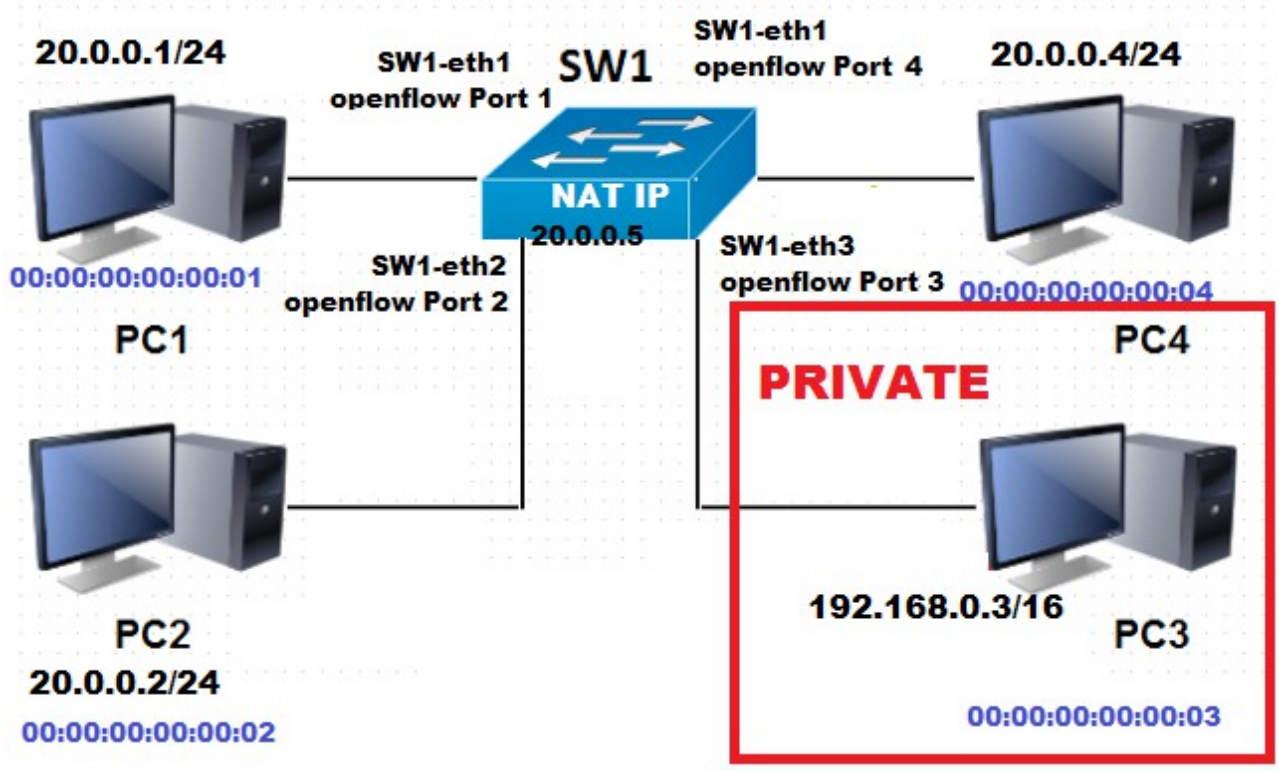
Προχωρούμε σε ταίριασμα επιπέδου 4(Θύρες)
Σε αυτό το παράδειγμα θα ξεκινήσουμε έναν απλό Python Web server στο PC3 και θα επιτρέψουμε την επικοινωνία όλων των άλλων τερματικών μόνο με το Web Server.



Για να εκκινήσουμε ένα απλό web server στο PC3
PC3 python -m SimpleHTTPServer 80 &
Θα επιτρέψουμε το ARP για όλα τα τερματικά για να διευκολυνθούμε.
sh ovs-ofctl add-flow SW1 arp,actions=normal
Για να επιτρέψουμε την επικοινωνία με το server μόνο θα πρέπει να χρησιμοποιήσουμε σαν κριτήριο ταιριάσματος στο κανόνα μας τη θύρα προορισμού του κάθε πακέτου.
sh ovs-ofctl add-flow SW1 priority=500, dl_type=0x800, nw_proto=6, tp_dst=80, actions=output: 3
όπου με το nw_proto=6 προσδιορίζουμε πακέτα τύπου tcp.
Για να ολοκληρωθεί η επικοινωνία πρέπει να εισάγουμε ένα κανόνα που να επιτρέπει και την επιστροφή
sh ovs-ofctl add-flow SW1 priority=800, ip, nw_src=10.0.0.3, actions=normal

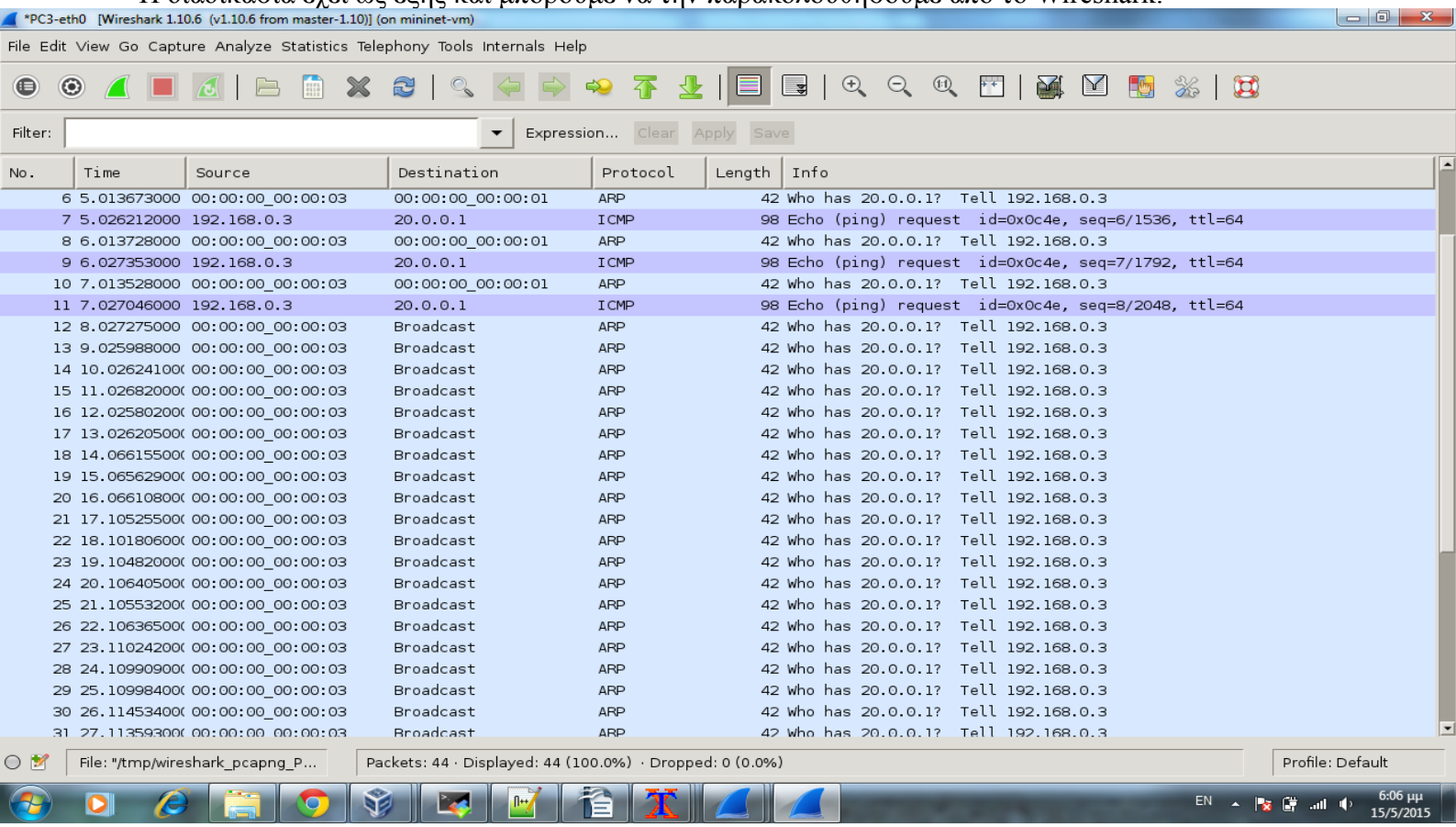
Για να συνδεθούμε στον εξυπηρετητή και να εμφανίσουμε την αρχική σελίδα πληκτρολογούμε
PC1 curl PC3
Σημείωση:
Σε περίπτωση που που το πρόγραμμα curl δεν είναι εγκατεστημένο
sudo apt-get install curl

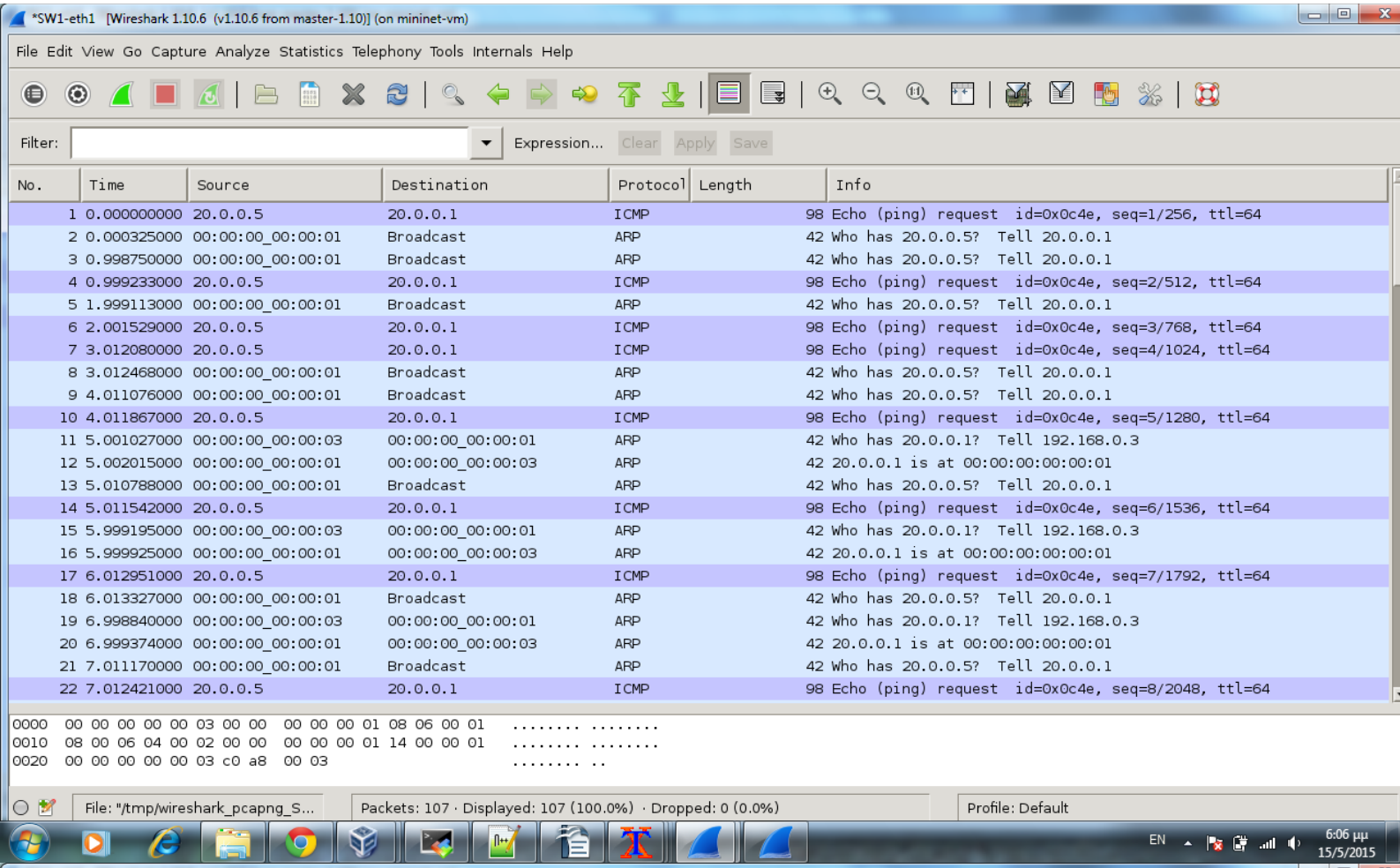
Ένα ενδιαφέρον συμπέρασμα που προκύπτει από το τελευταίο παράδειγμα είναι πως ένας μεταγωγέας openflow μπορεί να χρησιμοποιηθεί σαν firewall αν εισαγάγουμε τις κατάλληλες ρυθμίσεις στο πίνακα ροής του. Συγκεκριμένα εισάγοντας ένα κανόνα όπου το ταίριασμα γίνεται σε επίπεδο θύρας προορισμού(80) τα μόνα πακέτα που θα περάσουν προς το τερματικό PC3 είναι αυτά που αφορούν την επικοινωνία με τον εξυπηρετητή που λειτουργεί εκεί.
Τέλος θα προσπαθήσουμε να δημιουργήσουμε με τη βοήθεια των κανόνων openflow ένα σενάριο NAT.
Η τοπολογία που θα χρησιμοποιήσουμε είναι η ίδια με μερικές διαφοροποιήσεις.

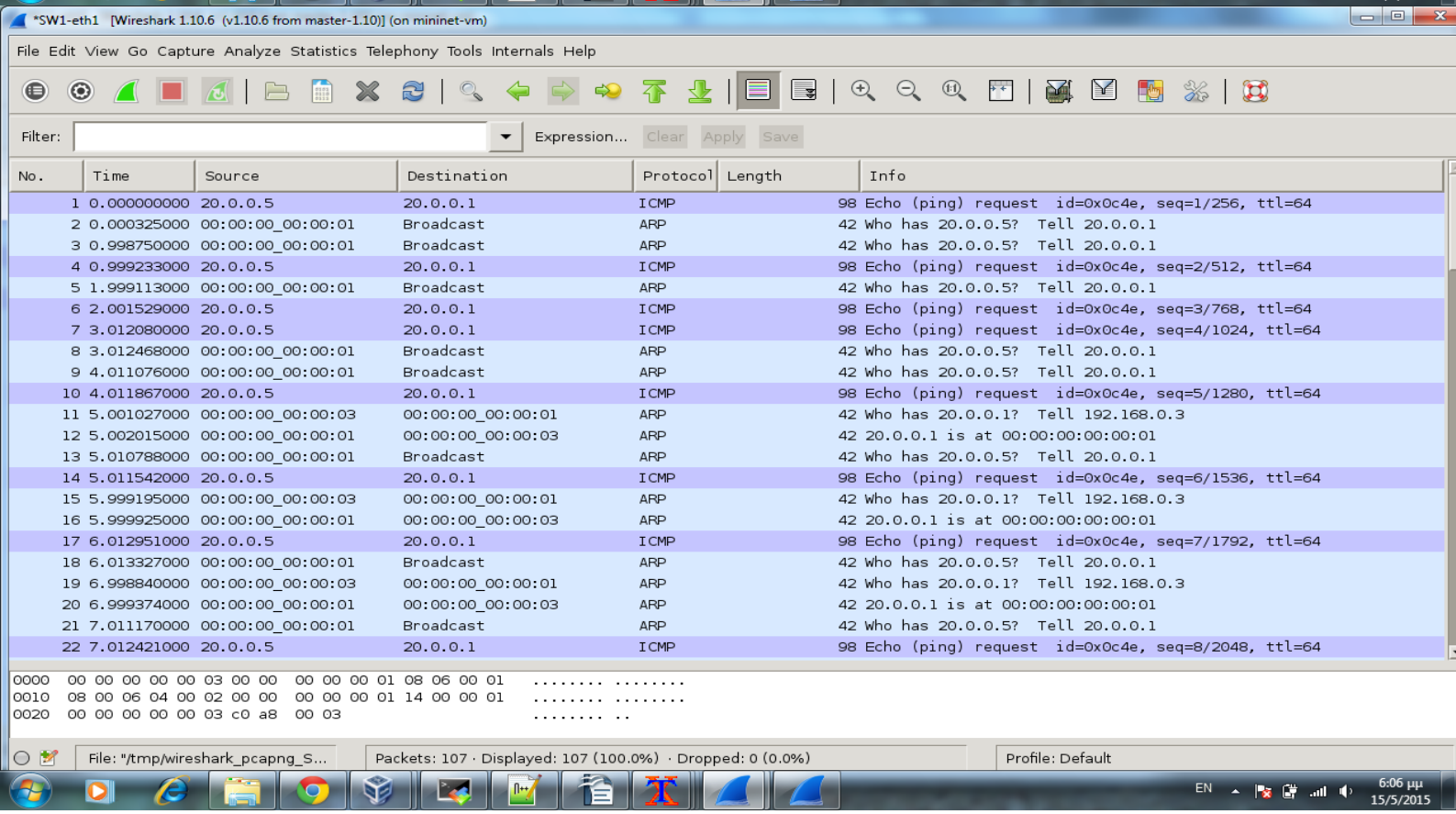


Στο σενάριο μας ο υπολογιστής PC3 βρίσκεται πίσω από το NAT και δεν είναι προσβάσιμος από τους γύρω υπολογιστές. Μπορεί να επικοινωνήσει μαζί τους αλλά μόνο όταν εκείνος το θελήσει. Στην απλούστερη μορφή εκείνος μπορεί να κάνει PING ενώ οι άλλοι όχι.
Θα ορίσουμε αρχικά ως προεπιλεγμένη πύλη τη διεπαφή που συνδέεται με το εκάστοτε υποδίκτυο για κάθε υπολογιστή.
PC1 ip route add default via 20.0.0.1
PC2 ip route add default via 20.0.0.2
PC3 ip route add default via 192.168.0.3
PC4 ip route add default via 20.0.0.4
Στη συνέχεια θα επιτρέψουμε την ανταλλαγή των πακέτων ARP
sh ovs-ofctl add-flow SW1 priority=500,arp,actions=flood
Κάθε ερώτημα ARP που φτάνει στο μεταγωγέα προωθείται σε όλες τις υπόλοιπες θύρες του.

Εισάγουμε τώρα τους κανόνες που θα επιτρέψουν την επικοινωνία του PC3 με τον έξω κόσμο.
sh ovs-ofctl add-flow SW1
priority=500,dl_type=0x800,nw_src=192.168.0.3,nw_dst=20.0.0.1,actions = mod_nw_src :
20.0.0.5,output:1
sh ovs-ofctl add-flow SW1
priority=500,dl_type=0x800,nw_src=192.168.0.3,nw_dst=20.0.0.2,actions = mod_nw_src :
20.0.0.5,output:2
sh ovs-ofctl add-flow SW1
priority=500,dl_type=0x800,nw_src=192.168.0.3,nw_dst=20.0.0.4,actions = mod_nw_src :
20.0.0.5,output:4
Στη συνέχεια εισάγουμε τους κανόνες που επιτρέπουν την απάντηση από τον έξω κόσμο.
sh ovs-ofctl add-flow SW1
priority=500,dl_type=0x800,nw_src=20.0.0.0/24,nw_dst=20.0.0.5,actions =mod_nw_dst:
192.168.1.3,output:3
Τέλος εισάγουμε το κανόνα αλλαγής της διεύθυνσης IP του PC3 με την NAT IP.
sh ovs-ofctl add-flow SW1
arp,nw_proto=1,dl_src=00:00:00:00:00:01,actions=mod_dl_src:86:bd:55:f9:45:f5,mod_dl_dst:00:0
0:00:00:00:0 1,output:1
Ο οποίος σημαίνει. Όταν φτάσει ένα πακέτο ARP(arp) request(nw_proto=1) με διεύθυνση αφετηρίας MAC(dl_src=00:00:00:00:00:01) άλλαξε τη διεύθυνση αφετηρίας MAC σε 86:bd:55:f9:45:f5(mod_dl_src:86:bd:55:f9:45:f5) και προορισμού σε 00:00:00:00:00:01(mod_dl_dst:00:00:00:00:00:0 1) και προώθησε το στην openflow θύρα 1.
στη συνέχεια σηκώνουμε από ένα Wireshark στα PC1,PC3,SW1-eth1 και κάνουμε ping από το PC3 στο PC1.
Η διαδικασία έχει ως εξής και μπορούμε να την παρακολουθήσουμε από το Wireshark.




*SW1-eth1 [Wireshark 1.10.6 (v1.10.6 from master-1.10)] (on mininet-vm)
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply Save
No. Time Source Destination Protocol Length Info
1 0.000000000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=1/256, ttl=64
2 0.000325000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
3 0.998750000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
4 0.999233000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=2/512, ttl=64
5 1.999113000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
6 2.001529000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=3/768, ttl=64
7 3.012080000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=4/1024, ttl=64
8 3.012468000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
9 4.011076000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
10 4.011867000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=5/1280, ttl=64
11 5.001027000 00:00:00_00:00:03 00:00:00_00:00:01 ARP 42 Who has 20.0.0.1? Tell 192.168.0.3
12 5.002015000 00:00:00_00:00:01 00:00:00_00:00:03 ARP 42 20.0.0.1 is at 00:00:00:00:00:01
13 5.010788000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
14 5.011542000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=6/1536, ttl=64
15 5.999195000 00:00:00_00:00:03 00:00:00_00:00:01 ARP 42 Who has 20.0.0.1? Tell 192.168.0.3
16 5.999925000 00:00:00_00:00:01 00:00:00_00:00:03 ARP 42 20.0.0.1 is at 00:00:00:00:00:01
17 6.012951000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=7/1792, ttl=64
18 6.013327000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
19 6.998840000 00:00:00_00:00:03 00:00:00_00:00:01 ARP 42 Who has 20.0.0.1? Tell 192.168.0.3
20 6.999374000 00:00:00_00:00:01 00:00:00_00:00:03 ARP 42 20.0.0.1 is at 00:00:00:00:00:01
21 7.011170000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
22 7.012421000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=8/2048, ttl=64
0000 00 00 00 00 00 03 00 00 00 00 00 01 08 06 00 01 ....
0010 08 00 06 04 00 02 00 00 00 00 00 01 14 00 00 01 ....
0020 00 00 00 00 00 03 c0 a8 00 03 ....
File: "/tmp/wireshark_pcapng_S... Packets: 107 · Displayed: 107 (100.0%) · Dropped: 0 (0.0%) Profile: Default
EN 6:06 μμ 15/5/2015


*SW1-eth1 [Wireshark 1.10.6 (v1.10.6 from master-1.10)] (on mininet-vm)
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply Save
No. Time Source Destination Protocol Length Info
1 0.000000000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=1/256, ttl=64
2 0.000325000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
3 0.998750000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
4 0.999233000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=2/512, ttl=64
5 1.999113000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
6 2.001529000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=3/768, ttl=64
7 3.012080000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=4/1024, ttl=64
8 3.012468000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
9 4.011076000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
10 4.011867000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=5/1280, ttl=64
11 5.001027000 00:00:00_00:00:03 00:00:00_00:00:01 ARP 42 Who has 20.0.0.1? Tell 192.168.0.3
12 5.002015000 00:00:00_00:00:01 00:00:00_00:00:03 ARP 42 20.0.0.1 is at 00:00:00:00:00:01
13 5.010788000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
14 5.011542000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=6/1536, ttl=64
15 5.999195000 00:00:00_00:00:03 00:00:00_00:00:01 ARP 42 Who has 20.0.0.1? Tell 192.168.0.3
16 5.999925000 00:00:00_00:00:01 00:00:00_00:00:03 ARP 42 20.0.0.1 is at 00:00:00:00:00:01
17 6.012951000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=7/1792, ttl=64
18 6.013327000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
19 6.998840000 00:00:00_00:00:03 00:00:00_00:00:01 ARP 42 Who has 20.0.0.1? Tell 192.168.0.3
20 6.999374000 00:00:00_00:00:01 00:00:00_00:00:03 ARP 42 20.0.0.1 is at 00:00:00:00:00:01
21 7.011170000 00:00:00_00:00:01 Broadcast ARP 42 Who has 20.0.0.5? Tell 20.0.0.1
22 7.012421000 20.0.0.5 20.0.0.1 ICMP 98 Echo (ping) request id=0x0c4e, seq=8/2048, ttl=64
0000 00 00 00 00 00 03 00 00 00 00 00 01 08 06 00 01 ....
0010 08 00 06 04 00 02 00 00 00 00 00 01 14 00 00 01 ....
0020 00 00 00 00 00 03 c0 a8 00 03 ....
File: "/tmp/wireshark_pcapng_S... Packets: 107 · Displayed: 107 (100.0%) · Dropped: 0 (0.0%) Profile: Default
EN 6:06 μμ 15/5/2015

Αρχικά το PC3 κάνει ARP request για την MAC του υπολογιστή με IP 20.0.0.1(PC1).Το πακέτο φτάνει στο μεταγωγέα και προωθείται στο PC1, που στέλνει ένα ARP reply με την διεύθυνση MAC του (00:00:00:00:00:01).Το πακέτο φτάνει στο μεταγωγέα που το προωθεί πίσω στο PC3.Εκείνο με τη σειρά του στέλνει ένα ICMP request στο PC1
Η έλλειψη συγχρονισμού που αναφέραμε σαν πρόβλημα είναι εμφανής στη καταγραφή καθώς το ping αρχίζει και μερικά ARP replies καταγράφονται αργότερα.
Παρατηρούμε όμως ότι το ping δεν ολοκληρώνεται ποτέ και ο λόγος είναι ότι στην ερώτηση του PC1 για τη διεύθυνσή MAC της NAT IP 20.0.0.5 δεν υπάρχει καμία απάντηση. Επομένως ο τελευταίος κανόνας που εισάγαμε προφανώς και δεν λειτουργεί.
Μετά από έρευνα που διεξάγαμε καταλήξαμε στο συμπέρασμα ότι ο μεταγωγέας δεν μπορεί να τροποποιήσει το πακέτο. Επομένως πρέπει να καταφύγουμε σε τροποποίηση των πινάκων ροής και στην ουσία να δημιουργήσουμε το πακέτο απάντησης.
Βέβαια το σύνολο των κανόνων για να το πετύχουμε αυτό είναι αρκετά πιο πολύπλοκο,της μορφής

```
sh ovs-vsctl set Bridge s1 "protocols=OpenFlow13"
sh ovs-ofctl add-group -OOpenFlow13 s1 group_id=1,type=all,bucket=output:1
sh ovs-ofctl add-group -OOpenFlow13 s1 group_id=2,type=all,bucket=output:2,4
sh ovs-ofctl add-group -OOpenFlow13 s1 group_id=3,type=all,bucket=output:3
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=0, priority=1000, dl_type=0x0806, actions=goto_table=105"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=0, priority=100, dl_dst=00:00:5E:00:02:01, action=goto_table=5"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=0, priority=100, dl_dst=00:00:5E:00:02:02, action=goto_table=5"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=0, priority=100, dl_dst=00:00:5E:00:02:03, action=goto_table=5"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=0, priority=0, action=goto_table=20"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=5, priority=65535, dl_type=0x0800, nw_dst=10.10.10.0/24 actions=goto_table=10"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=5, priority=65535, dl_type=0x0800, nw_dst=10.10.20.0/24 actions=goto_table=10"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=5, priority=100, dl_type=0x0800,  nw_dst=172.16.1.10
actions=mod_nw_dst=10.10.10.2, goto_table=10"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=5, priority=100, dl_type=0x0800,  nw_src=10.10.10.2,
actions=mod_nw_src=172.16.1.10,  goto_table=10"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=5, priority=0, actions=goto_table=10"
sh  ovs-ofctl add-flow -OOpenFlow13 s1 "table=10, dl_type=0x0800, nw_dst=10.10.10.0/24,
actions=mod_dl_src=00:00:5E:00:02:01, dec_ttl, goto_table=15"
sh  ovs-ofctl add-flow -OOpenFlow13 s1 "table=10, dl_type=0x0800, nw_dst=10.10.20.0/24,
actions=mod_dl_src=00:00:5E:00:02:02, dec_ttl, goto_table=15"
sh  ovs-ofctl add-flow -OOpenFlow13 s1 "table=10, dl_type=0x0800, nw_dst=172.16.1.0/24,
actions=mod_dl_src=00:00:5E:00:02:03, dec_ttl, goto_table=15"
sh  ovs-ofctl add-flow -OOpenFlow13 s1 "table=10, priority=0, actions=output:0"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=15, dl_type=0x0800, nw_dst=10.10.10.2, actions=mod_dl_dst:00:00:00:00:00:01,
goto_table=20"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=15, dl_type=0x0800, nw_dst=10.10.20.2, actions=mod_dl_dst:00:00:00:00:00:02,
goto_table=20"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=15, dl_type=0x0800, nw_dst=10.10.20.4, actions=mod_dl_dst:00:00:00:00:00:04,
goto_table=20"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=15, dl_type=0x0800, nw_dst=172.16.1.2, actions=mod_dl_dst:00:00:00:00:00:03,
goto_table=20"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=15, priority=0, actions=goto_table=20"
sh  ovs-ofctl add-flow -OOpenFlow13 s1 "table=20, priority=0, actions=goto_table=25"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=25, in_port=1, dl_dst=01:00:00:00:00:00/01:00:00:00:00:00, actions=group=1"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=25, in_port=2, dl_dst=01:00:00:00:00:00/01:00:00:00:00:00, actions=group=2"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=25, in_port=3, dl_dst=01:00:00:00:00:00/01:00:00:00:00:00, actions=group=3"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=25, in_port=4, dl_dst=01:00:00:00:00:00/01:00:00:00:00:00, actions=group=2"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=25, dl_dst=00:00:00:00:00:01,actions=output=1"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=25, dl_dst=00:00:00:00:00:02,actions=output=2"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=25, dl_dst=00:00:00:00:00:03,actions=output=3"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=25, dl_dst=00:00:00:00:00:04,actions=output=4"
sh ovs-ofctl add-flow -OOpenFlow13 s1 "table=105, dl_type=0x0806, nw_dst=10.10.10.1, actions=move:NXM_OF_ETH_SRC[]-
.....
```

και ξεφεύγει αρκετά σε πολυπλοκότητα.
Μπορούμε να παρακάμψουμε αυτό το εμπόδιο συνδέοντας το μεταγωγέα με έναν controller.

Έχοντας επομένως καταδείξει τη μέθοδο με την οποία τροποποιούμε τμήματα ενός πακέτου ολοκληρώσαμε τη παρουσίαση του openflow. Ελπίζουμε να αποτελέσει έναυσμα για τη συστηματικότερη μελέτη των εν λόγω μεθοδολογιών, ώστε στο μέλλον να ενταχθούν στη καθημερινότητα ενισχύοντας την έρευνα και τον διαχωρισμό των δικτύων.

7 *ΕΠΙΛΟΓΟΣ*

Στο κεφάλαιο του επιλόγου θα αναφερθούμε εν τάχυ στα βασικά σημεία που καλύψαμε. Τέλος θα επισημάνουμε μελλοντικές προοπτικές που υπάρχουν καθώς και θα προτείνουμε ιδέες για την αποτελεσματική ενσωμάτωση του πρωτοκόλλου openflow στη καθημερινότητα και στον τομέα των επιχειρήσεων.

7.1 Σύνοψη και συμπεράσματα

Στη παρούσα διπλωματική εργασία ασχοληθήκαμε με μεθόδους που μας επιτρέπουν να κατασκευάζουμε πειραματικά δίκτυα υπολογιστών. Αφού αναλύσαμε πλεονεκτήματα και μειονεκτήματα όλες, καταλήξαμε στη μέθοδο της εξομοίωσης που είχε συγκριτικά χαμηλότερο κόστος. Το πλαίσιο που μας εξυπηρετούσε ήταν εξομοίωση μέσω εικονικοποίησης, που μας επέτρεπε να κατασκευάσουμε πολύπλοκες τοπολογίες.

Στη συνέχεια αναζητήσαμε προγραμματιστικές πλατφόρμες που να πλαισιώνουν αυτό το σκεπτικό, και αφού καταλήξαμε στις πιο χρήσιμες, εκλέξαμε εξ αυτών το mininet που υποστήριζε επιπλέον το πρωτόκολλο openflow. Αναφερθήκαμε επιπρόσθετα στη τεχνική της ελαφριάς εικονικοποίησης, που χρησιμοποιεί το mininet για να μειώνει δραστικά τους αποθηκευτικούς πόρους που απαιτούνται για τη δημιουργία των εικονικών μηχανημάτων. Εστιάσαμε στο πρωτόκολλο openflow και πως υλοποιείται από το mininet. Περιγράψαμε με σαφήνεια και πληρότητα ένα ευρύτατο φάσμα εντολών και λειτουργιών ακολουθούμενο από διαφωτιστικά παραδείγματα όπου αυτό κρινόταν αναγκαίο.

Στο κομμάτι των προβλημάτων που αντιμετωπίσαμε, εισαγάγαμε το miniNExT που αποτελεί μία επέκταση του mininet και λύνει το πρόβλημα της πρόσθετης απομόνωσης που απαιτείται για να μπορούμε να τρέξουμε το quagga στα εικονικά μας μηχανήματα.
Στη συνέχεια στο κομμάτι του ελέγχου παρουσιάσαμε αναλυτικά παραδείγματα λειτουργίας του mininet όπου και καταδείχθηκε η ορθότητα των προτάσεων που παρουσιάσαμε. Στο τέλος των παραδειγμάτων συμπεριλάβαμε και αναλυτική παρουσίαση του πρωτοκόλλου openflow και ποικίλων τρόπων που μπορεί κανείς να το χρησιμοποιήσει καταλήγοντας στο συμπέρασμα ότι ένας μεταγωγέας openflow μπορεί εύκολα και γρήγορα να χρησιμοποιηθεί σαν firewall διαχωρίζοντας τη κίνηση πακέτων από και προς ένα τερματικό.

7.2 Μελλοντικές προοπτικές

Τα δίκτυα υπολογιστών έχουν αποκτήσει σημαντική θέση στις επιχειρήσεις, στα σπίτια και στα πανεπιστήμια. Η μεγάλη αυτή επιτυχία, αποτέλεσε ταυτόγχρονα ευλογία και κατάρα για τους ερευνητές. Η δουλειά τους είναι πλέον σχετική με τις εξελίξεις, αλλά η πιθανότητα τους να δημιουργήσουν κάτι επαναστατικό απέχει πολύ από τη πραγματικότητα. Η μείωση της αξίας κάθε νέου επιστημονικού ευρήματος, προέρχεται από την ήδη εγκατεστημένη, αχανή συλλογή πρωτοκόλλων και εξοπλισμού, και τη διστακτική στάση της επιχειρηματικής κοινότητας να επιτρέψει στους ερευνητές να πειραματιστούν με την αληθινή κίνηση δεδομένων.
Ως αποτέλεσμα η θέσπιση νέων τεχνολογιών στο ήδη υπάρχων σύνολο είναι πολύ δύσκολη αφού ο πήχης έχει τεθεί πλέον πολύ ψηλά. Απόρροια των προηγουμένων είναι οι επιστήμονες να μη μπορούν να δοκιμάσουν τις μελέτες τους σε ρεαλιστικά περιβάλλοντα με αποτέλεσμα να μη μπορούν να διαθέτουν πειστικά αποτελέσματα και επομένως να μη λαμβάνουν υποστήριξη.
Στο τέλος οι περισσότερες ιδέες που προέρχονται από τον ερευνητικό τομέα δεν δοκιμάζονται καν και περνούν απαρατήρητες.

Χαρακτηριστικό όλων των παραπάνω είναι η άποψη που έχει επικρατήσει, πως η παγκόσμια διαδικτυακή υποδομή έχει πλέον οριστικοποιηθεί.
Στην εργασία μας αναφερθήκαμε πως θα μπορούσε να λυθεί το πρόβλημα των ερευνητών. Για να μπορεί κανείς να δοκιμάσει νέα πρωτόκολλα μέσα σε ένα καθημερινό δίκτυο αρκεί να εξασφαλίσει πως όλοι οι μεταγωγείς, δρομολογητές υλοποιούν τις βασικές ενέργειες του πρωτοκόλλου openflow. Έτσι θα μπορεί φερ' ειπείν τα πακέτα που προέρχονται από το δικό του υπολογιστή να υπόκεινται στο νέο πρωτόκολλο προς δοκιμή ενώ όλα τα υπόλοιπα πακέτα θα υπόκεινται στο προκαθορισμένο πρωτόκολλο.

Αν και η λύση που προτείνουμε δεν έχει εφαρμοστεί ακόμα, ελπίζουμε στο να βοηθήσει τους κατασκευαστές να ξεπεράσουν το σκεπτικισμό τους όσον αφορά να παρέχουν στην ακαδημαϊκή κοινότητα το λογισμικό των μεταγωγέων, δρομολογητών καθώς η δική μας λύση ξεπερνάει αυτό το εμπόδιο.
Συγκεκριμένα:
1) έχει χαμηλό κόστος
2) μπορεί να υποστηρίξει ένα μεγάλο έυρος ερευνητικών συστημάτων
3) Διασφαλίζει το διαχωρισμό της κίνησης που προέρχεται από τα τερματικά των ερευνητών, από όλα τα υπόλοιπα.
4) είναι σύμφωνη με την απαίτηση των κατασκευαστών να μην “ανοίξουν” τα μηχανήματά τους
Κλείνοντας να αναφέρουμε τη πεποίθησή μας ότι θα αποτελέσει μία κερδοφόρα επένδυση για τους κατασκευαστές εξοπλισμού δικτύων υπολογιστών η ενσωμάτωση του λειτουργιών του openflow στα προϊόντα τους καθώς θα αποτελέσει πόλο έλξης της ερευνητικής κοινότητας, πανεπιστημίων, επιχειρήσεων κλπ.

7.3 Ευχαριστίες

Θέλω να ευχαριστήσω το καθηγητή επιβλέπων κύριο Ευστάθιο Συκά , και το μεταδιδακτορικό ερευνητή κύριο Δημήτριο Καλογερά για τη πολύτιμη βοήθεια και συμπαράσταση σε όλη τη διάρκεια συγγραφής της διπλωματικής. Τέλος να ευχαριστήσω τον υποψήφιο διδάκτορα του πανεπιστημίου της νότιας καλιφόρνιας κύριο Brandon Schlinker για τη πολύτιμη βοήθειά του σε θέματα code debugging και fault handling

8 *ΒΙΒΛΙΟΓΡΑΦΙΑ*

[1]Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, Nick McKeown
Reproducible Network Experiments Using Container Based Emulation, ACM, 2012

[2]Stephen Soltesz, Herbert Potzl, Marc E. Fiuczynski, Andy Bavler, Larry Peterson
Container-based Operating System Virtualization:A Scalable, High-performance Alternative to Hypervisors, ACM, 2012

[3]Jeff Ahrenholz, Claudiu Danilov, Thomas R. Henderson, Jae H. Kim
Core: A Real-Time Network Emulator
IEEE, 2008

[4]Bob Lantz, Brandon Heller, Nick Mc Keown
A Network in a Laptop:Rapid Prototyping for Software-Defined-Networks, ACM, 2010

[5]Karamjeet Kaur, Japinder Singh, Navtej Singh Ghumman
Mininet as a Software Defined Networking Testing Platform
ICCCS, 2014

[6]G.Calarco, M.Casoni
On the effectiveness of Linux containers for network virtualization, Elsevier, 2013

ΠΑΡΑΡΤΗΜΑ

Κώδικες για το τμήμα του ελέγχου
Α)

```
"""
Example topology of Quagga routers
"""
import inspect
import os
from mininext.topo import Topo
from mininext.services.quagga import QuaggaService
net = None
class QuaggaTopo(Topo):
   "Creates a topology of Quagga routers"
   def __init__(self):
      Topo.__init__(self)
      selfPath = os.path.dirname(os.path.abspath(
         inspect.getfile(inspect.currentframe())))  # script directory
      quaggaSvc = QuaggaService(autoStop=False)
      quaggaBaseConfigPath = selfPath + '/configs/'
      PC1=self.addHost(name='PC1',
                       hostname='PC1',
                       privateLogDir=True,
                       privateRunDir=True,
                       inMountNamespace=True,
                       inPIDNamespace=True,
                       inUTSNamespace=True)
      quaggaSvcConfig = \
            {'quaggaConfigPath': quaggaBaseConfigPath + 'PC1'}
      self.addNodeService(node='PC1', service=quaggaSvc,
                          nodeConfig=quaggaSvcConfig)
      R1          = self.addHost(name='R1',
                                 hostname='R1',
                                 privateLogDir=True,
                                 privateRunDir=True,
                                 inMountNamespace=True,
                                 inPIDNamespace=True,
                                 inUTSNamespace=True)
      quaggaSvcConfig = \
            {'quaggaConfigPath': quaggaBaseConfigPath + 'R1'}
      self.addNodeService(node='R1', service=quaggaSvc,
                          nodeConfig=quaggaSvcConfig)
      R2          = self.addHost(name='R2',
                                 hostname='R2',
                                 privateLogDir=True,
                                 privateRunDir=True,
                                 inMountNamespace=True,
                                 inPIDNamespace=True,
                                 inUTSNamespace=True)
```

```
    quaggaSvcConfig = \
        {'quaggaConfigPath': quaggaBaseConfigPath + 'R2'}
    self.addNodeService(node='R2', service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)
    R3          = self.addHost(name='R3',
                               hostname='R3',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)
    quaggaSvcConfig = \
        {'quaggaConfigPath': quaggaBaseConfigPath + 'R3'}
    self.addNodeService(node='R3', service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)
    PC2         = self.addHost(name='PC2',
                               hostname='PC2',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)
    quaggaSvcConfig = \
        {'quaggaConfigPath': quaggaBaseConfigPath + 'PC2'}
    self.addNodeService(node='PC2', service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)
    PC3         = self.addHost(name='PC3',
                               hostname='PC3',
                               privateLogDir=True,
                               privateRunDir=True,
                               inMountNamespace=True,
                               inPIDNamespace=True,
                               inUTSNamespace=True)
    quaggaSvcConfig = \
        {'quaggaConfigPath': quaggaBaseConfigPath + 'PC3'}
    self.addNodeService(node='PC3', service=quaggaSvc,
                        nodeConfig=quaggaSvcConfig)
    self.addcustlink(PC1, R1)
    self.addcustlink(R1,R2)
    self.addcustlink(R1,R3)
    self.addcustlink(R2,R3)
    self.addcustlink(R2, PC2)
    self.addcustlink(R3,PC3)
def addcustlink(self,pc1, pc2):
    sw=self.addSwitch(pc1+'-'+pc2)
    self.addLink(pc1,sw)
    self.addLink(sw,pc2)
```

B)

```
from mininet.topo import Topo

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

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


      Topo.__init__( self )


      PC1 = self.addHost( 'PC1' )
      PC2 = self.addHost( 'PC2' )
      PC3 = self.addHost( 'PC3' )
      PC4 = self.addHost( 'PC4' )
      SW1 = self.addSwitch( 'SW1' )


      self.addLink( PC1, SW1 )
      self.addLink( PC2, SW1 )
      self.addLink( PC3, SW1 )
      self.addLink( PC4, SW1 )


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