Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακή Διατριβή στα Πληροφοριακά και Επικοινωνιακά Συστήματα

Μελέτη SDN Δικτύων και Προτύπου Openflow, Εξομοιώσεις - Use cases- με το Mininet

Παπασταματάκη Μαρία Μαρκέλλα

Επιβλέπων Καθηγητής
Κυριάκος Βλάχος

Μάιος 2015

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μελέτη SDN Δικτύων και Προτύπου Openflow, Εξομοιώσεις - Use cases- με το Mininet

Παπασταματάκη Μαρία Μαρκέλλα

Επιβλέπων Καθηγητής
Κυριάκος Βλάχος

Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε
προς μερική εκπλήρωση των απαιτήσεων για απόκτηση

μεταπτυχιακού τίτλου σπουδών
στα Πληροφοριακά Συστήματα

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

Μάιος 2015

Περίληψη

Η ιδέα των προγραμματιζόμενων δικτύων έχει πρόσφατα αποκτήσει σημαντική δυναμική, λόγω της εμφάνισης της αρχιτεκτονικής SDN η οποία υπόσχεται να απλοποιήσει δραματικά τη διαχείριση των δικτύων και να δώσει ώθηση στην καινοτομία. Σε αυτήν τη μεταπτυχιακή διατριβή μελετάμε την αρχιτεκτονική SDN και το πρότυπο OpenFlow, το οποίο αποτελεί το πιο ευρέως αποδεκτό πρωτόκολλο επικοινωνίας για την ανωτέρω αρχιτεκτονική. Πέραν του σκοπού της θεωρητικής εξοικείωσης με τις έννοιες SDN και Openflow, η παρούσα μεταπτυχιακή διατριβή αφιερώνει ένα κεφάλαιο στη μελέτη του Mininet που αποτελεί το πλέον χρησιμοποιούμενο εργαλείο εξομοίωσης, τόσο από την ακαδημαϊκή όσο και από την ερευνητική κοινότητα, για την εμβάθυνση σε αυτές τις τεχνολογίες. Δημιουργήθηκε επίσης ένας οδηγός εγκατάστασης του Mininet, που προστέθηκε ως παράρτημα της διατριβής, για να διευκολύνει την μελλοντική ενασχόληση οποιουδήποτε το επιθυμεί με αυτό. Ακόμα, μέσω αναλυτικών βημάτων περιγράφθηκε ένας ικανός αριθμός πειραμάτων με στόχο την εξοικείωση με την αρχιτεκτονική και με τις λειτουργίες της. Στόχος ήταν να συλλεχθεί και να ομαδοποιηθεί αρκετό υλικό ώστε να διαπιστωθούν τα μειονεκτήματα και τα προτερήματα της επαναστατικής αυτής μορφής δικτύων.

Summary

Recently, the idea of programmable networks has gained a strong dynamic, further empowered by the emergence of SDN architecture, which promises to dramatically simplify network administration and give an extra boost to innovation. This thesis studies the SDN architecture as well as the OpenFlow protocol, which is the most widely used communication protocol for the aforementioned architecture. After becoming acquainted with the ideas behind SDN and OpenFlow, the next chapter's goal is to get familiar with Mininet, the most acceptable simulation tool by the academic community, for anyone who wants to thoroughly study these technologies.

Also included in this thesis is an installation guide for Mininet, which is included as an appendix, so as to facilitate and ease anyone who might wish to experiment with it in the future.

Furthermore, a series of experiments was meticulously analyzed in steps, so as to get familiarized with the architecture and it's features and functions. The final goal was to collect and categorize enough experiment data, in order to identify the pros and cons of this revolutionary type of networks.

Ευχαριστίες

Θα ήθελα στο σημείο αυτό να ευχαριστήσω ιδιαίτερα τον Κο Βλάχο για τις αξιόλογες παρεμβάσεις του στην διατριβή, αλλά και για την επιλογή ενός τόσο ενδιαφέροντος θέματος, καθώς και την οικογένεια μου για την πολύτιμη βοήθεια τους και στήριξη κατά την διάρκεια των μεταπτυχιακών μου σπουδών.

Περιεχόμενα

Κεφάλαιο 1
Εισαγωγή

Τα δίκτυα σήμερα είναι ζωτικής σημασίας και αποτελούν αναπόσπαστο κομμάτι της υποδομής των επιχειρήσεων, των σχολείων, των πανεπιστήμιων αλλά και των σπιτιών μας. Το διαδίκτυο όμως στην σημερινή του μορφή, εμφανίζει "ακαμψία" και περιορισμούς λόγω της αθρόας ανάπτυξης και επέκτασής του, περιορίζοντας τη δυνατότητα ανάπτυξης και εφαρμογής καινοτομιών. Η μαζική αυτή αποδοχή και επιτυχία του διαδικτύου στην τρέχουσα του δομή και σύνθεση, αποτελεί ταυτόχρονα ευλογία και κατάρα για τους ερευνητές που ασχολούνται με θέματα δικτύωσης και επιθυμούν να διερευνήσουν νέα πρωτόκολλα και μηχανισμούς, καθιστώντας την εργασία τους σημαντική μεν αλλά περιορίζοντας ταυτόχρονα, την δυνατότητα εφαρμογής νέων πρωτοκόλλων και την εισαγωγή νέων ιδεών. Παράλληλα η εκρηκτική αύξηση των χρηστών αλλά και των απαιτήσεων τους, οδηγούν τις συσκευές που απαρτίζουν το συμβατικό δίκτυο, στα όρια τους.

Όπως αναφέρθηκε προηγουμένως, η λεγόμενη "ακαμψία" του διαδικτύου οφείλεται σε μεγάλο βαθμό στη στενή σύνδεση μεταξύ του επιπέδου δεδομένων και του επιπέδου ελέγχου που σημαίνει ότι οι αποφάσεις σχετικά με τη ροή δεδομένων στο δίκτυο γίνονται σε αυτό κάθε αυτό το στοιχείο του δικτύου. Πολλά προβλήματα που οδηγούν στην "ακαμψία" του διαδικτύου οφείλονται στην έλλειψη μιας κοινής διεπαφής ελέγχου στις διαφορετικές συσκευές [15]. Στο παρελθόν προτάθηκαν λύσεις όπως τα Content Delivery Networks (CDNs) δίκτυα για να μειωθεί αυτό το φαινόμενο.

Αυτήν την στιγμή τόσο η επιστημονική κοινότητα όσο και κομμάτι των κατασκευαστών στρέφεται στο SDN σαν μια λύση που θα διευκολύνει την εξέλιξη του δικτύου και της καινοτομίας επιτρέποντας την ταχεία ανάπτυξη νέων υπηρεσιών και πρωτόκολλων έτσι το SDN σήμερα προσελκύει ιδιαίτερη προσοχή τόσο από την ακαδημαϊκή κοινότητα όσο και από τη βιομηχανία. Μια ομάδα φορέων εκμετάλλευσης δικτύων, παροχών υπηρεσιών και προμηθευτών έχουν πλέον δημιουργήσει το Open Network Foundation ως φορείς έρευνας στην βιομηχανία. Από την άλλη η ακαδημαϊκή κοινότητα έχει ιδρύσει το OpenFlow Network Research Center. Επίσης οργανισμοί προτυποποίησης όπως οι IRTF & IETF συμμετέχουν στις προσπάθειες για έρευνα και εξέλιξη στον νέο αυτό τομέα.

Τα "καθοριζόμενα από λογισμικό δίκτυα ή αλλιώς η δικτύωση οριζόμενη από το λογισμικό" (SDN-Software Defined Networks) αποτελούν το κλειδί για την λύση των προβλημάτων που αντιμετωπίζει το διαδίκτυο. Η αρχιτεκτονική αυτή, παρέχει στους ερευνητές, έναν νέο τρόπο για τη δοκιμή καινοτόμων τεχνολογιών και πρωτοκόλλων. Αυτός είναι και ο λόγος για τον οποίο η συγκεκριμένη αρχιτεκτονική αναμένεται να αποτελέσει το μέλλον της δικτύωσης υπολογιστών [15].

Τα καθοριζόμενα από λογισμικό δίκτυα - (SDN δίκτυα), εισάγουν ένα νέο τρόπο για να χειριστούν τη μεγάλη ποσότητα των πακέτων που διέρχονται από το δίκτυο. Χειρίζονται τα πακέτα ανά ροή, μέσω κεντρικού ελέγχου και επιτυγχάνουν έτσι την γρήγορη διαχείριση τους επιταχύνοντας τους χρόνους διαβίβασης και επεξεργασίας τους. Με το SDN, το επίπεδο ελέγχου και το επίπεδο προώθησης διαχωρίζονται και η λογική ελέγχου τίθεται σε μια κεντρική τοποθεσία. Ένα εξειδικευμένο πρωτόκολλο επικοινωνίας χρησιμοποιείται μεταξύ των δυο αυτών επιπέδων. Το πιο ευρέως διαδεδομένο και αποδεκτό πρωτόκολλο είναι το Openflow που αποτελεί ερευνητικό προϊόν του Πανεπιστημίου Stanford και το οποίο θα μελετηθεί στην παρούσα διατριβή.

Με την χρήση των δικτύων αυτών μπορούμε να έχουμε αρκετά οφέλη όπως την απλοποίηση αλγορίθμων, να οδηγηθούμε σε αλλαγές που θα επηρεάσουν το

υλικό - hardware των δικτύων και να έχουμε εξάλειψη των συσκευών που διαχειρίζονται την κίνηση στο δίκτυο κυρίως για σκοπούς ασφάλειας, όπως τα firewalls ή οι network address translators. Ακόμα το SDN μπορεί να αποτελέσει το κύριο υπόβαθρο για τα δίκτυα cloud.

1.1 Διάρθρωση Μεταπτυχιακής Διατριβής

Η παρούσα πτυχιακή εργασία, αποτελείται από έξι κεφάλαια και τέσσερα παραρτήματα.

Το πρώτο από αυτά, περιέχει μια μικρή εισαγωγή πάνω στα θέματα τα οποία πρόκειται να αναλυθούν.

Στο δεύτερο, μελετάται η SDN τεχνολογία. Αναλυτικότερα, παρουσιάζεται σε βάθος η αρχιτεκτονική του πρωτοκόλλου, τα είδη υποστηριζόμενων υπηρεσιών και ο συσχετισμός τους με τις τεχνικές παραμέτρους.

Στο τρίτο κεφάλαιο, πραγματοποιείται μελέτη του πρωτοκόλλου Openflow, σύμφωνα με τα specifications που έχουν εκδοθεί ως τώρα.

Στο τέταρτο κεφάλαιο, μελετάται το εργαλείο με το οποίο πραγματοποιήθηκαν οι εξομοιώσεις των δικτύων. Εξηγείται ο λόγος που επιλέχθηκε έναντι των άλλων προγραμμάτων και γίνεται σύντομη αναφορά στα πλεονεκτήματα του, που το καθιστούν απαραίτητο για την μελέτη δικτύων.

Στο πέμπτο κεφάλαιο, ακολουθεί το πειραματικό μέρος, οι εντολές με τις οποίες δημιουργήθηκαν τα πειράματα, τα αποτελέσματα και οι γραφικές παραστάσεις που συλλέχθηκαν με τη βοήθεια του wireshark. Επιπροσθέτως, γίνεται ανάλυση των αποτελεσμάτων.

Στο έκτο κεφάλαιο γίνεται αναφορά στην σημασία του SDN στα σύγχρονα δίκτυα και τους τρόπους χρήσης του. Κλείνοντας αναφερόμαστε σε μελλοντικά ενδιαφέροντα που ανακύπτουν με εναρκτήριο την παρούσα μεταπτυχιακή διατριβή.

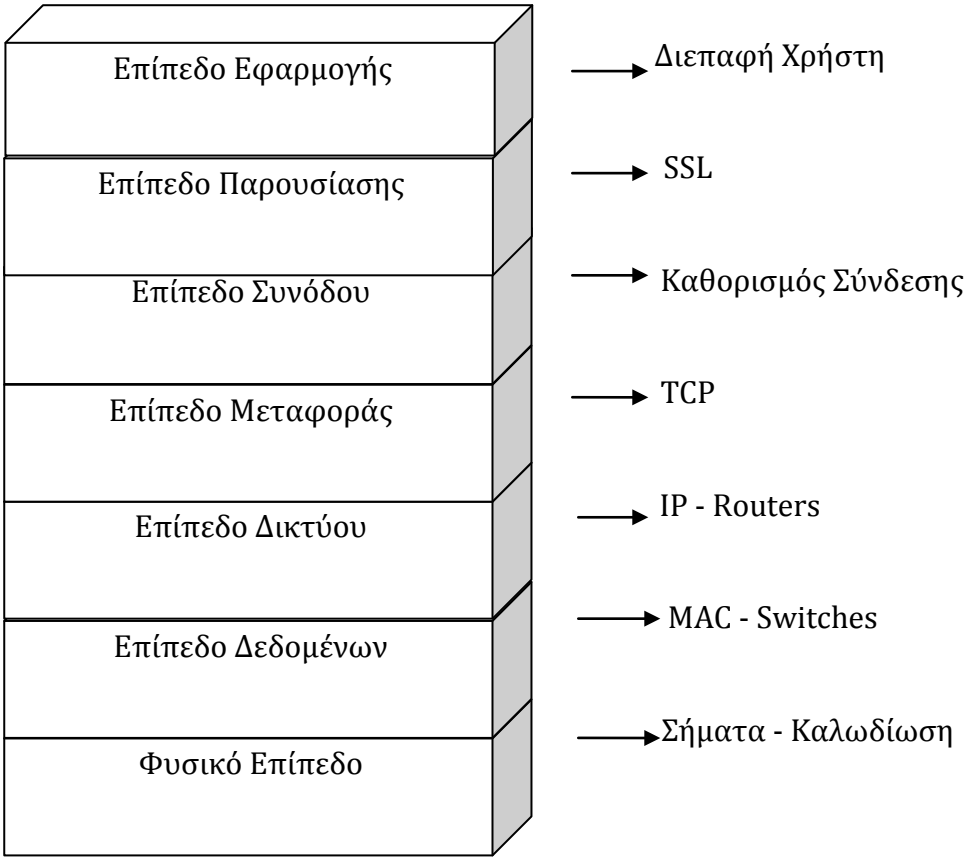
Τέλος, αναφέρονται οι πηγές πληροφοριών που λειτούργησαν ως σημεία αναφοράς γι' αυτή την εργασία, ομαδοποιημένες ανά papers, βιβλία και ιστοσελίδες, καθώς και τέσσερα παραρτήματα. Το πρώτο παράρτημα παρουσιάζει τα πιο συχνά χρησιμοποιούμενα ακρωνύμιά της SDN τεχνολογίας, το δεύτερο τον κώδικα με τον οποίο τρέξαμε τα πειράματα, το τρίτο κατάλογο των εικόνων και των γραφημάτων του κύριου μέρους της μεταπτυχιακής διατριβής και το τρίτο αναφέρει λεπτομερώς τον τρόπο εγκατάστασης του Mininet.

Κεφάλαιο 2
SDN

Στο παρόν κεφάλαιο θα περιγράψουμε τα υπάρχοντα δίκτυα, τα προβλήματα τους και την ανάγκη δημιουργίας μιας νέας μορφής δικτύων. Στην συνέχεια θα αναφερθούμε στα SDN δίκτυα, τις βασικές τους αρχές, την αρχιτεκτονική τους. Ακόμα θα αναφέρουμε στα πλεονεκτήματα των SDN δικτύων καθώς και τα πρωτόκολλα επικοινωνίας που χρησιμοποιούν.

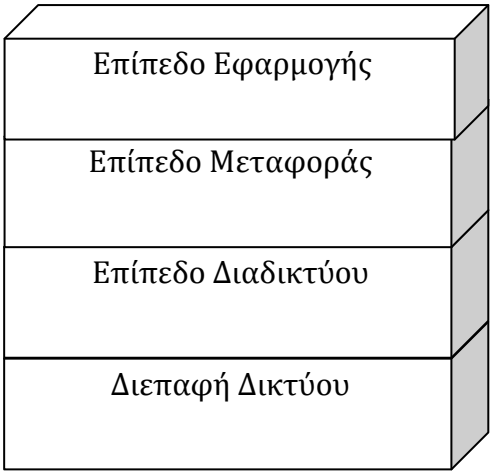
2.1 Υπάρχοντα Δίκτυα

Το μοντέλο της Διασύνδεσης Ανοιχτών Συστημάτων ή αλλιώς μοντέλο αναφοράς (OSI, Open Systems Interconnection) είναι το θεωρητικό μοντέλο που περιγράφει τον τρόπο με τον οποίο μπορούν να επικοινωνήσουν μεταξύ τους δύο οποιαδήποτε διαφορετικά συστήματα στα υφιστάμενα δίκτυα. Το μοντέλο αναπτύχθηκε από τον Διεθνή Οργανισμό Τυποποίησης (International Standards Organisation, ISO). Στο παρακάτω σχήμα φαίνεται η αρχιτεκτονική του μοντέλου OSI καθώς και κάποια χαρακτηριστικά παραδείγματα χρήσης του κάθε επιπέδου. Στην αρχιτεκτονική αυτή μια δέσμη ενεργειών λαμβάνει χώρα σε κάθε επίπεδο έτσι ώστε να καταστήσει ικανή την επικοινωνία μεταξύ δύο οποιανδήποτε υπολογιστών σε ένα κοινό δίκτυο.



Εικόνα 2. 1 Αρχιτεκτονική μοντέλου OSI - Παραδείγματα χρήσης ανά επίπεδο.

Παράλληλα το TCP/IP είναι το πρωτόκολλο που χρησιμοποιείται για τις διαδικασίες διευθυνσιοδότησης στο διαδίκτυο [29]. Το TCP/IP έχει την μορφή που περιγράφεται στο σχήμα 2.2 και αντιστοιχίζεται με τα επίπεδα του μοντέλου OSI. Ουσιαστικά η διεπαφή δικτύου ανταποκρίνεται στα δύο κατώτατα επίπεδα του μοντέλου OSI (επίπεδο δεδομένων και φυσικό επίπεδο), το επίπεδο διαδικτύου αντιστοιχεί με το επίπεδο δικτύου, το επίπεδο μεταφοράς με το επίπεδο μεταφοράς του επιπέδου OSI, ενώ το επίπεδο εφαρμογής του TCP αντιστοιχίζεται με τα επίπεδα εφαρμογής, παρουσίασης και συνόδου, δηλαδή τα τρία ανώτερα επίπεδα του OSI μοντέλου .



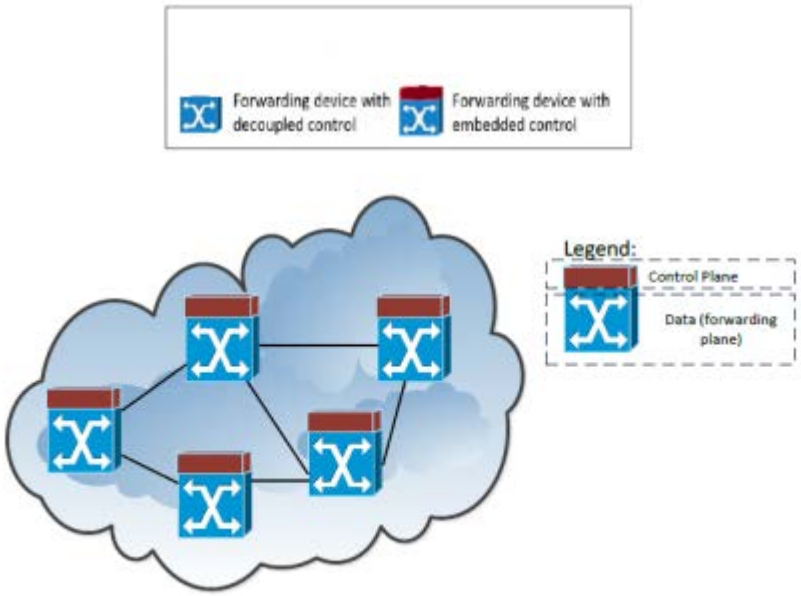
Εικόνα 2. 2 Το TCP/IP μοντέλο

Και στα δύο προαναφερθέντα μοντέλα κάθε επίπεδο προσφέρει πληροφορίες και δεδομένα ελέγχου στο επόμενο επίπεδο.

Η μεταγωγή πακέτων στα δίκτυα καθιστούσε την κάθε συσκευή αυτόνομη και αυτοτελή συσκευή ελέγχου ακόμα και όταν εφάρμοζαν ένα συγκεκριμένο πρωτόκολλο δρομολόγησης ή πολιτική διαχείρισης. Κάθε πακέτο δεδομένων περνά την ίδια ανάλυση και επεξεργασία παρόλο που ανήκει σε μια συγκεκριμένη ροή. Αυτή η συμβατική και παλαιωμένη αρχιτεκτονική δικτύων μπορεί να προκαλέσει προβλήματα επειδή δεν θα μπορεί πλέον να υποστηρίξει την εντυπωσιακή αύξηση των χρηστών και τη χρήση τόσο απαιτητικών δικτυακών εφαρμογών. Ο διαχωρισμός του επιπέδου ελέγχου και του επίπεδου διαχείρισης από το επίπεδο προώθησης δεδομένων στις δικτυακές συσκευές αποτελεί την κεντρική ιδέα του SDN. Ουσιαστικά ένας κεντρικός ελεγκτής είναι υπεύθυνος για την διαχείριση πολλών συσκευών προώθησης και αυτό έχει ως αποτέλεσμα πιο επαρκή, καινοτόμα και πιο ευέλικτα δίκτυα που ανταποκρίνονται στις απαιτήσεις των χρηστών. Τα SDN δίκτυα διαχειρίζονται μέσω ενός NOS - δικτυακού λειτουργικού συστήματος που βρίσκεται σε κεντρικούς ελεγκτές ώστε όλοι οι μεταγωγείς να λειτουργούν ομαλά, αρμονικά και ευέλικτα. Οι μεταγωγείς αυτοί δεν απαιτείται να είναι στην ίδια γεωγραφική περιοχή. Η διαχείριση πολλών κατανεμημένων κέντρων έλεγχου, που μπορεί να ανήκουν σε έναν πάροχο υπηρεσιών διαδικτύου, αποτελεί ένα παράδειγμα δικτύου SDN. Το Openflow αποτελεί το πιο ευρέως διαδεδομένο πρωτόκολλο επικοινωνίας αυτής της αρχιτεκτονικής.

Στα σύγχρονα δίκτυα οι μεταγωγείς - switches είναι συσκευές που η κύρια τους λειτουργία είναι η μετακίνηση των δεδομένων από τη θύρα εισόδου στη θύρα εξόδου, μια διαδικασία που καθορίζεται από τους κανόνες δρομολόγησης στην επικεφαλίδα του κάθε πακέτου. Οι μεταγωγείς όπως γνωρίζουμε δουλεύουν στο επίπεδο 2, επίπεδο ζεύξης δεδομένων - data link layer του μοντέλου αναφοράς OSI. Κατά την διαδικασία της μεταγωγής οι συσκευές προώθησης κάνουν χρήση των πινάκων προώθησης με την βοήθεια των οποίων αποφασίζεται η επιθυμητή θύρα εξόδου. Οι δρομολογητές - routers είναι ανώτερες συσκευές από τους απλούς μεταγωγείς, συμβουλεύονται πίνακες προώθησης αλλά μπορούν να αποφασίζουν την βέλτιστη δυνατή διαδρομή ώστε να έχουμε την πιο σωστή δρομολόγηση του πακέτου. Ταυτόχρονα με βάση την επικεφαλίδα IP του κάθε πακέτου βρίσκουν τον τελικό του προορισμό. Οι δρομολογητές λειτουργούν στο 3ο επίπεδο του OSI επίπεδο δικτύου – network layer.

Στα υπάρχοντα δίκτυα λοιπόν κάθε συσκευή δικτύου, όπως μεταγωγέας ή δρομολογητής, αποτελεί μια αυτόνομη οντότητα προώθησης, ελέγχου και διαχείρισης της κίνησης του δικτύου. Όμως, κάθε εταιρία κατασκευής διαδικτυακών συσκευών υλοποιεί με διαφορετικό τρόπο αυτές τις λειτουργίες και οι προδιαγραφές τους ανταποκρίνονται σε συγκεκριμένες ανάγκες. Ο χειριστής του δικτύου θα πρέπει να επέμβει σε κάθε μια από αυτές τις συσκευές με κάποια κατάλληλη για το μηχάνημα γλώσσα διαμόρφωσης - configuration language και να ορίσει τον τρόπο που θα προωθούν τοπικά το κάθε πακέτο. Αυτή η τακτική είναι χρονοβόρα ενώ οδηγεί σε δυσλειτουργίες και ασυμβατότητες μεταξύ συσκευών από διαφορετικούς κατασκευαστές. Παράλληλα, οι διεπαφές των δρομολογητών και των μεταγωγέων είναι κλειστές σε τρίτους και συνεπώς δεν μπορεί κάποιος να επέμβει σε αυτές αλλάζοντας για παράδειγμα τα πρωτόκολλα δρομολόγησης που χρησιμοποιεί η εκάστοτε συσκευή. Ακόμα, οι κατασκευαστές δεν θέλουν να μοιραστούν το πως λειτουργεί το εσωτερικό των μεταγωγέων. Αυτό σημαίνει ότι για να επέμβουμε στα μηχανήματα αυτά θα πρέπει να πραγματοποιήσουμε ανάλογες αλλαγές στο υλικό τους.



Εικόνα 2. 3 Υπάρχον δίκτυο. (Κάθε συσκευή ενσωματώνει τόσο το επίπεδο ελέγχου όσο και το επίπεδο δεδομένων - επίπεδο προώθησης.) [16]

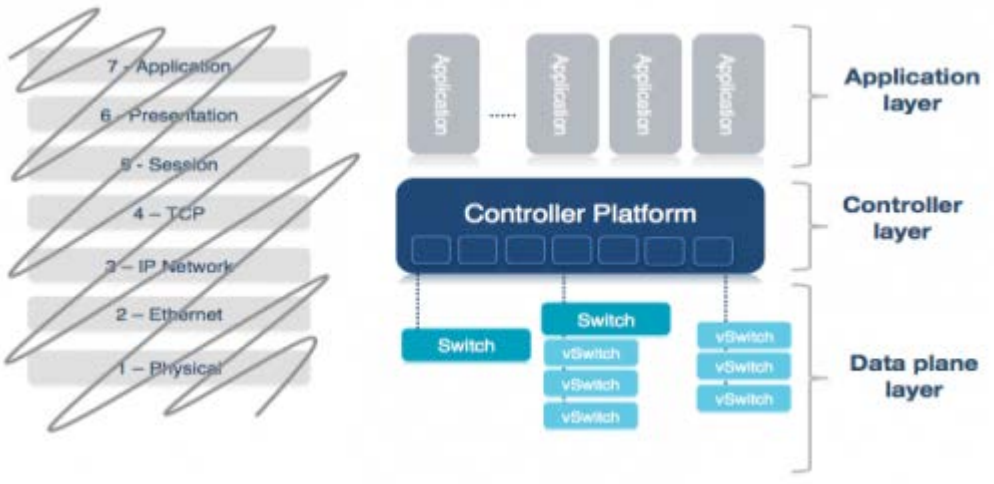
2.2 Δικτύωση Καθορισμένη από Λογισμικό (SDN)

Μια λύση για να υπερβούμε τα προβλήματα και τις δυσλειτουργίες αυτές, είναι να χρησιμοποιήσουμε το SDN. Οι πρώτες προσπάθειες για προγραμματιζόμενα δίκτυα και συνεπώς προπομποί του SDN ήταν το Open Signaling το 1995, το Active Networking την ίδια περίοδο, το DCAN 4D Project το 2004 ενώ πιο πρόσφατα το NETCONF και το Ethane το 2006. [15]

Το SDN σύμφωνα με το Ίδρυμα για την Ανοικτή Δικτύωση - Open Networking Foundation είναι η αρχιτεκτονική με την οποία επιτυγχάνεται ο φυσικός διαχωρισμός του επιπέδου ελέγχου δικτύου από το επίπεδο προώθησης και ο τρόπος με τον οποίο το επίπεδο ελέγχου μπορεί να ελέγχει πολλές διαφορετικές συσκευές. Με το SDN δίνεται η δυνατότητα στους ερευνητές να κάνουν πειράματα για την δημιουργία νέων πρωτοκόλλων ή αλγόριθμων δρομολόγησης σε ετερογενής μεταγωγείς σε line rate και με πραγματική κίνηση. [2] Παράλληλα με την βοήθεια του μπορούμε να αναπτύξουμε νέα πρωτόκολλα, πιο οικονομικά και πιο ευέλικτα.

2.2.1 Αρχιτεκτονική SDN

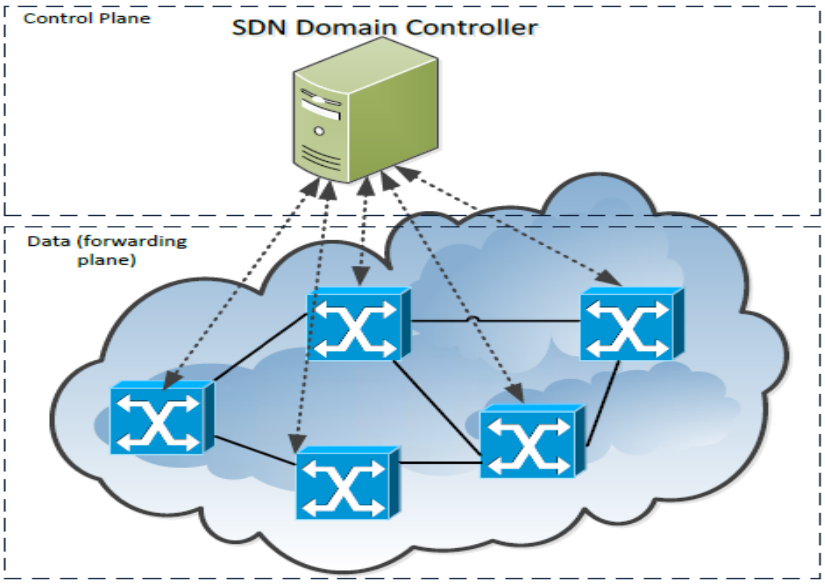
Το SDN ως νέα αρχιτεκτονική βασίζεται στην δημιουργία τριών "νέων" επιπέδων πάνω στα οποία στηρίζεται η διασύνδεση των συσκευών. Τα επίπεδα αυτά είναι το επίπεδο δεδομένων, το επίπεδο του ελεγκτή και το επίπεδο εφαρμογής. Σχηματικά το νέο "μοντέλο" επικοινωνίας αναπαρίσταται με το σχήμα 2.3. Το επίπεδο ελέγχου και το επίπεδο δεδομένων διαχωρίζονται και δημιουργείται μιας μορφής διεπαφή μεταξύ των. Το επίπεδο ελέγχου λειτουργεί έξω από τους δρομολογητές πάνω σε ένα network operating system (NOS) με την βοήθεια του οποίου χειριζόμαστε τους πίνακες προώθησης των μεταγωγέων και των δρομολογητών δηλαδή των συσκευών προώθησης του δικτύου. Με το νέο αυτό μοντέλο μπορεί πλέον να γίνεται εύκολα, απλά με εγκατάσταση νέου λογισμικού στο NOS, αλλαγή κεντρικά στον τρόπο διαχείρισης της κίνησης του δικτύου, χωρίς να επεμβαίνουμε ατομικά στις συσκευές προώθησης. Αυτή η νέα δυνατότητα απλοποιεί και διευκολύνει την γρήγορη διαχείριση του δικτύου. [13]



Εικόνα 2. 4 Το πρωτόκολλο OSI και η αφηρημένη νέα μορφή πρωτοκόλλου που υλοποιείται με το SDN [23]

Η δικτύωση καθορισμένη από λογισμικό - SDN αναπτύχθηκε για να διευκολύνει την καινοτομία και να επιτρέπει τον έλεγχο, μέσω του προγραμματισμού, της μεταφοράς δεδομένων στα δίκτυα. Μέσω αυτής της αρχιτεκτονικής έχουμε δυναμική διαχείριση των δικτύων, πιο αποδοτική χρήση τους ενώ ταυτόχρονα μειώνουμε το κόστος λειτουργίας τους και τον χρόνο απόκρισης των συσκευών κατά την χρήση απαιτητικών εφαρμογών που απαιτούν μεγάλο εύρος ζώνης για να λειτουργήσουν βέλτιστα. Ουσιαστικά επιτυγχάνεται ο διαχωρισμός του επιπέδου ελέγχου του δικτύου και των λειτουργιών προώθησης που λαμβάνουν χώρα στις συσκευές του δικτύου το οποίο έχει σαν αποτέλεσμα ο έλεγχος δικτύου να μπορεί να γίνει μέσω προγραμματισμού. Επιπροσθέτως, οι διαχειριστές του δικτύου μπορούν να προσαρμόσουν δυναμικά την κίνηση που λαμβάνει χώρα στο δίκτυο ώστε να ανταποκρίνεται στις εκάστοτε ανάγκες. [13]

Στο SDN, η ευφυΐα του δικτύου εντοπίζεται κεντρικά σε ελεγκτές καθοριζόμενους από λογισμικό (επίπεδο ελέγχου) που έχουν μια συνολική πανοραμική εικόνα του δικτύου, και σε δικτυακές συσκευές που γίνονται απλές συσκευές προώθησης πακέτων (επίπεδο δεδομένων) που μπορεί να προγραμματιστούν μέσω ενός λογισμικού ανοικτού κώδικα όπως το OpenFlow. Έτσι υπάρχει ένα κεντροποιημένο δίκτυο που βασίζει την λειτουργία του σε ένα ή περισσότερους κεντρικούς ελεγκτές, αντί σε επιμέρους συσκευές προώθησης. Με αυτόν τον τρόπο οι εφαρμογές έχουν μια ενιαία αντιμετώπιση σαν το δίκτυο να αποτελείται για παράδειγμα από ένα και μόνο λογικό μεταγωγέα. [15]

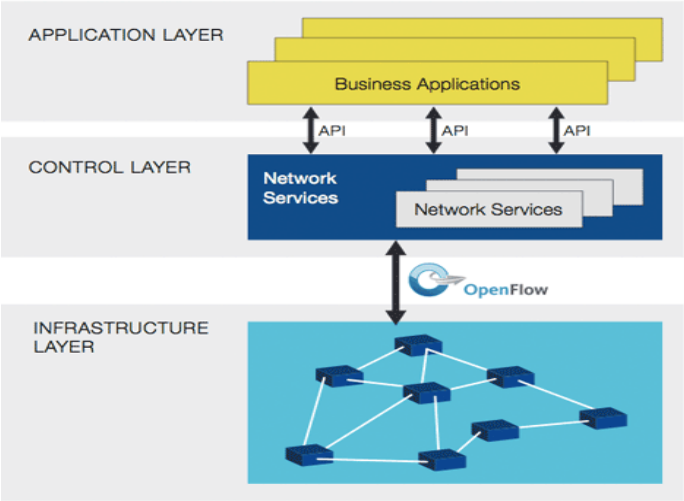


Εικόνα 2. 5 Το SDN δίκτυο. [16]

Λόγω του ότι το SDN απομονώνει το επίπεδο ελέγχου από το επίπεδο προώθησης (δεδομένων) το επίπεδο ελέγχου πλέον μετακινείται σε μια κεντρική συσκευή, έναν ελεγκτή καθοριζόμενο από λογισμικό (controller). Ενώ το επίπεδο δεδομένων παραμένει στις συσκευές προώθησης του δικτύου. Εάν ο ελεγκτής τοποθετηθεί σε έναν υψηλών προδιαγραφών εξυπηρετητή τότε έχουμε μεγαλύτερη επεξεργαστική ισχύ σε σχέση με τις συσκευές που είχαν ενσωματωμένο το επίπεδο ελέγχου του παρελθόντος, παράλληλα μπορούμε πολύ πιο εύκολα να έχουμε αναβαθμίσεις ή αλλαγές στην λειτουργία του δικτύου εάν αυτό απαιτείται, μέσω λογισμικού. Το SDN επιτρέπει στους διαχειριστές των δικτύων να ρυθμίσουν με ασφάλεια το δίκτυο και να βελτιστοποιήσουν τους πόρους του με την βοήθεια αυτοματοποιημένων προγραμμάτων λογισμικού τα οποία μπορούν να γράφουν και οι ίδιοι επειδή είναι ανεξάρτητα πλέον από τις εμπορικές συσκευές και το ιδιόκτητο τους λογισμικό.

Το επίπεδο ελέγχου απεικονίζεται ως ένας διακομιστής ή συσκευή που λαμβάνει την υποχρέωση λογοδοσίας για την επικοινωνία μεταξύ των επαγγελματικών εφαρμογών και το επίπεδο των δεδομένων. Το επίπεδο δεδομένων είναι

εκπροσωπείται από το δίκτυο υποδομής, όπου δεν υπάρχουν ουσιαστικές διαφορές μεταξύ πραγματικού υλικού και εικονικών συσκευών δικτύου. Έτσι, το επίπεδο ελέγχου πρέπει να δημιουργήσει μια αφηρημένη έννοια του δικτύου στον διαχειριστή και από τις δύο πλευρές, τόσο από την εφαρμογή όσο και από την πλευρά της υποδομής.



Εικόνα 2. 6 Η αρχιτεκτονική του SDN [36]

2.2.2 Πλεονεκτήματα SDN

Το SDN συνεπώς έχει προταθεί για να διευκολύνει την εξέλιξη του δικτύου και την καινοτομία επιτρέποντας την ταχεία ανάπτυξη νέων υπηρεσιών και την δημιουργία νέων πρωτοκόλλων. Υλοποιείται μέσω ανοικτών προτύπων, έτσι απλοποιεί το σχεδιασμό των δικτύων ενώ οι εντολές δίνονται από κεντρικούς ελεγκτές αντίθετα με το παρελθόν, όπου οι συσκευές ήταν προκαθορισμένες όσον αφορά τον τρόπο λειτουργίας τους, από τους κατασκευαστές τους ή από κάποιο εξειδικευμένο διαχειριστή. Αυτοί είναι και ο λόγοι για τους οποίους χρησιμοποιείται ως βασικό πλέον μάθημα από πολλά Πανεπιστήμια. Η μόνη απαίτηση είναι ότι χρειάζεται συμβατές συσκευές για να λειτουργήσει.

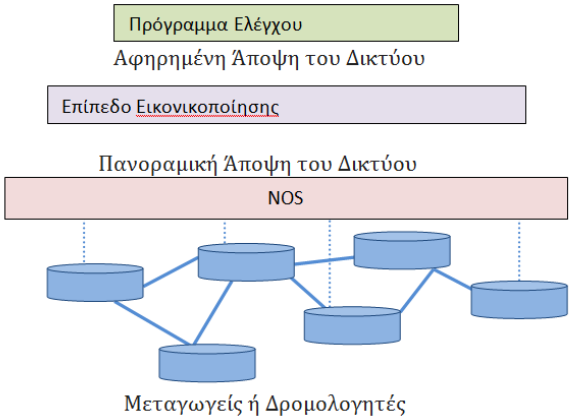
Τέσσερα σημεία κλειδιά της τεχνολογίας SDN, που την καθιστούν ευρέως πια αποδεκτή τόσο από την ακαδημαϊκή - ερευνητική κοινότητα όσο κανι από τους κατασκευαστές, είναι τα παρακάτω:

- Το SDN είναι ένα σύνολο αφηρημένων εννοιών για το επίπεδο ελέγχου, δεν αποτελείται από πολύπλοκους διαφορετικούς μηχανισμούς.

- Το SDN βασίζεται στο NOS - Network Operating System όπως το NOX που επιβλέπει και ελέγχει τους μεταγωγείς κεντρικά

- Το SDN δημιουργεί μια αφηρημένη έννοια του υπάρχοντος δικτύου γεγονός που καθιστά πιο εύκολο τον έλεγχο των λειτουργιών προώθησης και ελέγχου

- Η μεγάλη διαφορά που καθιστά την τεχνολογία SDN τόσο σημαντική σε σχέση με τις άλλες είναι ότι επιτρέπει την εικονικοποίηση - virtualization των δικτύων.



Εικόνα 2. 7 Τα βασικά στοιχεία της SDN τεχνολογίας

Τα κύρια πλεονεκτήματα του SDN και κατά συνέπεια των SDN δικτύων είναι ότι είναι δυνατό να προγραμματιστούν άμεσα, ο έλεγχος του δικτύου είναι δυνατός λόγω του ότι είναι διαχωρισμένος από το επίπεδο προώθησης και τις διαδικασίες του. Το δίκτυο πλέον είναι ευέλικτο. Συγκεκριμένα διαχωρίζοντας τα επίπεδα ελέγχου και προώθησης επιτρέπεται στους διαχειριστές να προσαρμόζουν δυναμικά τη ροή της κίνησης σε ολόκληρο το δίκτυο ώστε να ανταποκρίνεται στις μεταβαλλόμενες ανάγκες του και στις μεγάλες απαιτήσεις των εφαρμογών για διαθέσιμο εύρος ζώνης.

Ο τρόπος με τον οποίο προσφέρει κεντρική διαχείριση του δικτύου το SDN είναι το ότι η νοημοσύνη του Δικτύου πλέον είναι συγκεντρωμένη σε ελεγκτές SDN

που λειτουργούν με βάση το λογισμικό και διατηρούν μια σφαιρική εικόνα του δικτύου, η οποία δίνει την εικόνα στις εφαρμογές που τρέχουν, σαν το δίκτυο να αποτελείται από ένα και μόνο μεταγωγέα. Το δίκτυο μπορεί να ελεγχθεί και να ρυθμιστεί μέσω προγραμματισμού μέσω του SDN. Οι διαχειριστές του δικτύων μπορούν να ρυθμίσουν, να διαχειριστούν, να εξασφαλίσουν και να βελτιστοποιήσουν τους διαθέσιμους δικτυακούς πόρους γρήγορα μέσω δυναμικών αυτόματων προγραμμάτων SDN που μπορούν να γράψουν οι ίδιοι μιας και δεν εξαρτώνται πλέον από τη κάθε εταιρεία και το μηχάνημα της. Η τεχνολογία SDN βασίζεται σε ανοικτά πρότυπα ουδέτερα όσον αφορά τα μηχανήματα, δηλαδή η τεχνολογία που εφαρμόζεται στο δίκτυο δεν εξαρτάται πλέον από το κάθε μηχάνημα προώθησης όπως στα παραδοσιακά δίκτυα και τον κατασκευαστή του, απλοποιώντας το σχεδιασμό των δικτύων και διευκολύνοντας την διαχείριση του λόγω του ότι είναι πολύ εύκολο να καθορίσεις την κίνηση στο δίκτυο κεντρικά μέσω οδηγιών προς τους ελεγκτές SDN. [36]

2.2.3 Πρωτόκολλα Επικοινωνίας και Οντότητες SDN

Αυτή τη στιγμή δύο είναι οι κύριες υπάρχουσες αρχιτεκτονικές SDN: ForCES - ForCES Forwarding and Control Element Separation [29] και Openflow με την τελευταία να είναι η σπουδαιότερη και πιο ευρέως εφαρμόσιμη τεχνολογία.

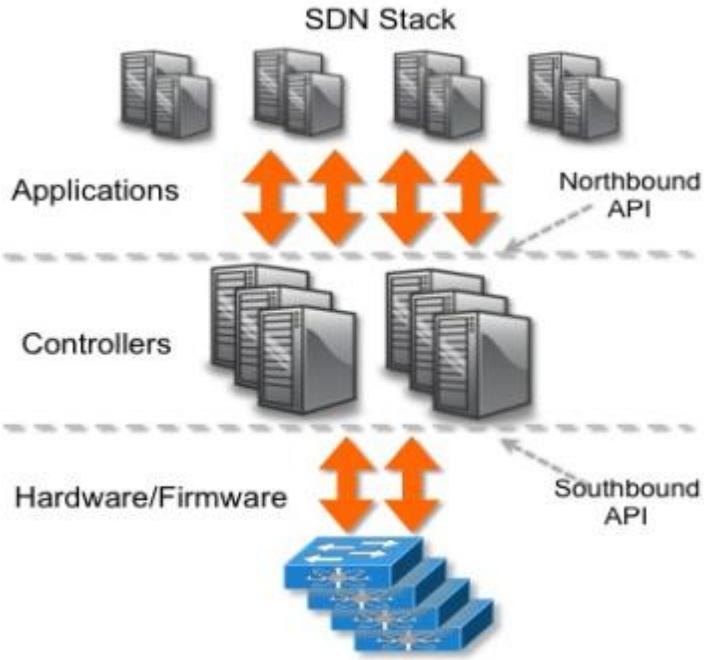
Το πρωτόκολλο ForCES ορίζει δύο λογικές οντότητες. Η πρώτη οντότητα ονομάζεται Forwarding Στοιχείο (FE) στοιχείο προώθησης και το δεύτερο στοιχείο ελέγχου (CE), τα οποία αμφότερα εφαρμόζουν το πρωτόκολλο ForCES για να επικοινωνούν. Τα FE είναι υπεύθυνα για τη χρήση του υποκείμενου υλικού και τον χειρισμό ανά πακέτο. Τα CE πραγματοποιούν τον έλεγχο και τις λειτουργίες σηματοδότησης και χρησιμοποιούν το πρωτόκολλο ForCES για να ορίσουν τον τρόπο που οι συσκευές προώθησης θα χειρίζονται τα πακέτα. Το πρωτόκολλο λειτουργεί με βάση ένα μοντέλο αφέντη σκλάβου, όπου τα στοιχεία προώθησης FES είναι σκλάβοι και τα στοιχεία ελέγχου CE είναι αφέντες.

Επί του παρόντος, όμως έχει επικρατήσει μια προδιαγραφή του SDN και οι σχετικές με αυτό υλοποιήσεις του τρόπου επικοινωνίας μεταξύ του επιπέδου δεδομένων και των εφαρμογών που ονομάζεται OpenFlow. Το OpenFlow δεν προσδιορίζει τον τρόπο που το επίπεδο ελέγχου θα εφαρμοσθεί τεχνικά ούτε το πώς η υποδομή του δικτύου θα χτιστεί αλλά είναι υπεύθυνο για την επικοινωνία τους. Η τυποποίηση των στοιχείων σε SDN έγιναν από το Ίδρυμα ανοικτού δικτύου Open Network Foundation (ONF), η οποία είναι μια κοινοπραξία της βιομηχανίας μη κερδοσκοπικού χαρακτήρα, που εργάζονται σε στενή συνεργασία για το OpenFlow. Αυτή η κατάσταση οδηγεί στην γενική γνώμη ότι το OpenFlow είναι ισοδύναμο με το SDN και ότι δεν υπάρχει κανένας περιορισμός σε ποια τεχνολογία μπορεί να χρησιμοποιηθεί σε μία υποδομή που βασίζεται στο SDN.

Το OpenFlow συνεπώς είναι ένα πρωτόκολλο επικοινωνίας που παρέχει πρόσβαση στο επίπεδο προώθησης ενός διακόπτη δικτύου (μεταγωγέα) ή δρομολογητή μέσω του δικτύου. Μέσω αυτού, μπορεί να επιτευχθεί ο διαχωρισμός μεταξύ του επιπέδου ελέγχου (control) και του επιπέδου προώθησης (forwarding) πακέτων σε ένα δίκτυο. Η βασική ιδέα είναι ότι μπορούμε να εκμεταλλευτούμε το γεγονός πως οι περισσότεροι Ethernet μεταγωγείς (switches) πλέον περιέχουν πίνακες εγγραφών ροών δεδομένων (flow-tables) ώστε να υλοποιούνται υπηρεσίες όπως Network Address Translation (NAT), Quality of Service (QoS), Firewall κ.α. [3]. Το OpenFlow παρέχει ένα ελεύθερο πρωτόκολλο για τον προγραμματισμό αυτών των flow-tables. Το Openflow καθορίζει ένα κανόνα για κάθε ροή, εάν ένα πακέτο ταιριάζει σε ένα συγκεκριμένο κανόνα τότε οι ανάλογες ενέργειες πραγματοποιούνται, τέτοιες ενέργειες μπορεί να είναι η προώθηση, η αλλαγή, η αποστολή του πακέτου σε κάποια ουρά κτλ.

Στο SDN, το επίπεδο ελέγχου του δικτύου είναι αποσυνδεδεμένο από την υποκείμενη υποδομή. Σε πολλά σενάρια, το επίπεδο ελέγχου είναι ενοποιημένο με ένα κεντρικό υπολογιστή - ελεγκτή που χρησιμοποιεί το πρωτόκολλο OpenFlow, ως μέθοδο επικοινωνίας, για τον έλεγχο κάθε κόμβου και τη ροή της κίνησης στο δίκτυο.

Πιο αναλυτικά εφαρμογές - που ονομάζονται northbound applications - κάθονται στην κορυφή του ελεγκτή. Το βόρειο API προσφέρει μια αφηρημένη διασύνδεση δικτύου στις εφαρμογές και τα συστήματα διαχείρισης που βρίσκονται στην κορυφή της στοίβας αρχιτεκτονικής του SDN [44]. Οι πληροφορίες από αυτές τις εφαρμογές περνούν μαζί μέσω μιας νότια διεπαφής - southbound application. Η νότια διεπαφή επιτρέπει στον ελεγκτή να καθορίσει τη συμπεριφορά των μεταγωγέων που βρίσκονται στο κάτω μέρος της SDN στοίβας. Ουσιαστικά, η βόρεια διεπαφή όσο και η νότια διεπαφή επιτρέπει σε συγκεκριμένη συνιστώσα - μέρος του δικτύου να επικοινωνεί με ένα συστατικό του δικτύου που βρίσκεται σε υψηλότερο ή χαμηλότερο επίπεδο, αντίστοιχα.



Εικόνα 2. 8 Διεπαφές SDN [32]

Κεφάλαιο 3
Openflow

Το Openflow αποτελεί βασικό κομμάτι του SDN. Η σύλληψη της ιδέας του Openflow έγινε στο Πανεπιστήμιο του Stanford το 2008. Τον Δεκέμβριο του 2009 κυκλοφόρησε η πρώτη έκδοση του πρωτοκόλλου version1.0. Το Open Networking Foundation - Ίδρυμα ανοιχτής δικτύωσης, είναι ο οργανισμός που διαχειρίζεται το πρωτόκολλο. Με την βοήθεια του μπορούμε να ανταποκριθούμε στις απαιτήσεις και στα νέα δεδομένα που εμφανίζουν τα δίκτυα την σημερινή εποχή. Ακόμα προσφέρει την επιθυμητή προσαρμοστικότητα στον έλεγχο ροής της πληροφορίας στα σύγχρονα δίκτυα, ενώ παράλληλα επιτρέπει την μείωση της πολυπλοκότητας της διαχείρισης και λειτουργίας του δικτύου.

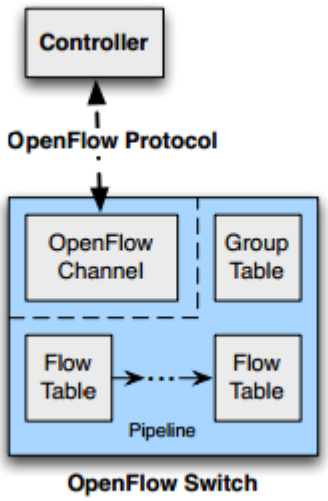
Πλέον είναι πολύ εύκολο για τις υφιστάμενες επιχειρήσεις αλλά και για τους φορείς να υιοθετήσουν τεχνολογίες SDN ακόμα και σε δίκτυα που αποτελούνται από συσκευές από διαφορετικούς κατασκευαστές. Το OpenFlow λύνει ένα βασικό πρόβλημα με την εφαρμογή του, βοηθά στο να δημιουργηθούν και να δοκιμαστούν νέες μέθοδοι δικτύωσης, νέα πρωτόκολλα και αλγόριθμοι πάνω σε ήδη ενεργά λειτουργικά δίκτυα ενώ τα υπάρχοντα πρωτοκόλλα δρομολόγησης και ασφάλειας παραμένουν ως έχουν. Έτσι δίνεται η δυνατότητα έρευνας και ανάπτυξης νέων τεχνολογιών χωρίς να διακυβεύεται η ορθή λειτουργία των προυπάρχοντων δικτύων και χωρίς να γίνονται αλλαγές στις υποδομές των δικτύων. [18]

Το OpenFlow συνεπώς μπορεί να τρέχει μαζί με τα παραδοσιακά πρωτοκόλλα στο ίδιο μηχάνημα με την χρήση της εικονοποίησης - virtualization, δηλαδή οι συσκευές μπορούν να τρέχουν ταυτόχρονα τα παραδοσιακά πρωτόκολλα και αλγόριθμους ενώ παράλληλα έχουν την δυνατότητα να διαχειρίζονται τα πακέτα ροής σύμφωνα με το πρωτόκολλο Openflow [49]. Αυτό έχει σαν αποτέλεσμα να μπορούν να γίνουν πειράματα και έρευνες πάνω στα ήδη υπάρχοντα δίκτυα παράλληλα με την εξυπηρέτηση της κλασικής κίνησης

του δικτύου χωρίς να δημιουργούνται προβλήματα είτε στην ροή της κίνησης του υπάρχοντος δικτύου είτε στη ερευνητική δραστηριότητα. Το δίκτυο παραμένει στην παλαιά του μορφή ενώ ο ερευνητής δοκιμάζει με πραγματική κίνηση νέες μεθόδους επικοινωνίας που σκοπό έχουν να ενισχύουν την βέλτιστη λειτουργικότητα του δικτύου.

Με την βοήθεια του Openflow ο έλεγχος δρομολόγησης των πακέτων περνάει σε ένα ή περισσότερους κεντρικούς ελεγκτές, οι οποίοι τρέχουν σε υψηλών δυνατοτήτων εξυπηρετητές και είναι σε διαρκή επικοινωνία με τις άλλες συσκευές προώθησης, που απαρτίζουν το δίκτυο και υποστηρίζουν το πρωτόκολλο, ώστε τα πακέτα να προωθούνται σωστά μέσα στο δίκτυο. Ουσιαστικά το Openflow μας βοηθά να χειραγωγούμε και να ελέγχουμε το επίπεδο προώθησης των συσκευών προώθησης στα δίκτυα όπως είναι οι δρομολογητές και οι μεταγωγείς.

Οι αποφάσεις δρομολόγησης λαμβάνουν χώρα σύμφωνα με τις flow entries - καταχωρήσεις ροής που αποστέλλουν οι ελεγκτές στις συσκευές προώθησης και οι οποίες καταγράφονται στους πίνακες προώθησης των συσκευών. Αυτή η διαδικασία δίνει την δυνατότητα για ευελιξία και προσαρμοστικότητα στο εκάστοτε τύπο ροής που τρέχει στο δίκτυο και προσαρμόζει την λειτουργία του δικτύου στον τύπο των εφαρμογών που τρέχουν.



Εικόνα 3. 1 Ο Openflow μεταγωγέας [33][20]

Η βασική ιδέα είναι απλή το Openflow εκμεταλλεύεται το γεγονός ότι οι περισσότεροι σύγχρονοι μεταγωγείς Ethernet και δρομολογητές περιέχουν πίνακες ροής. Παρόλο που οι πίνακες ροής του κάθε προμηθευτή είναι διαφορετικοί , έχει προσδιοριστεί ένα κοινό σύνολο λειτουργιών που τρέχουν στην πλειοψηφία των μεταγωγέων και δρομολογητών. Το OpenFlow εκμεταλλεύεται αυτό το κοινό σύνολο των λειτουργιών.

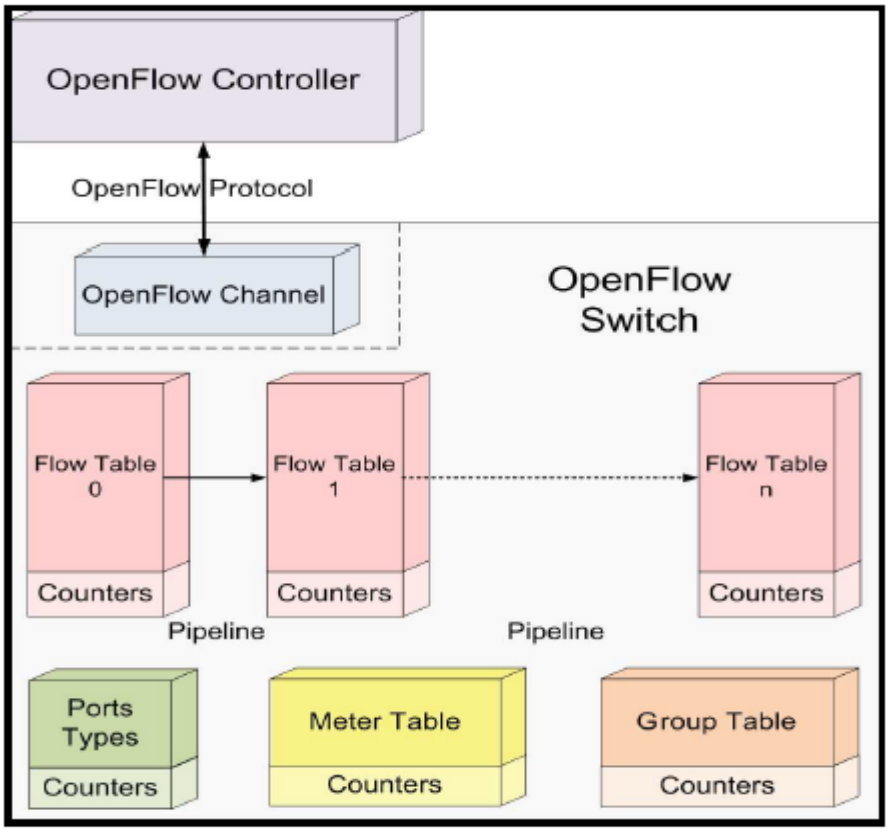
Το OpenFlow παρέχει ένα ανοικτό πρωτόκολλο με το οποίο μπορεί να προγραμματιστούν οι πίνακες ροής σε διαφορετικούς διακόπτες και δρομολογητές. Ένας διαχειριστής δικτύου μπορεί να διαχωρίσει την κίνηση σε ροή πραγματική και σε ροή για έρευνα. Οι ερευνητές μπορούν να ελέγχουν τις ροές που επιθυμούν επιλέγοντας τις διαδρομές που θα ακολουθήσουν τα πακέτα καθώς και την επεξεργασία θα υποστούν. Με τον τρόπο αυτό, οι ερευνητές μπορούν να δοκιμάσουν νέα πρωτόκολλα δρομολόγησης, νέα πρότυπα ασφαλείας, νέα σχήματα επικοινωνίας, ακόμη και εναλλακτικές λύσεις από το IP πρωτόκολλο. Ταυτόχρονα στο ίδιο δίκτυο, η κίνηση στην κλασική της μορφή είναι απομονωμένη και υποβάλλεται σε επεξεργασία κατά τον ίδιο τρόπο όπως σήμερα.

3.1 Ο Openflow Μεταγωγέας

Ένας μεταγωγέας OpenFlow αποτελείται από τουλάχιστον τρία μέρη: Έναν ή περισσότερους πίνακες ροής, που περιέχουν λίστα από ενέργειες που αντιστοιχίζονται στην κάθε μια ροή και ορίζουν στον μεταγωγέα πως να διαχειριστεί την συγκεκριμένη ροή, ένα πίνακα ομαδοποίησης - group table, ένα κανάλι που συνδέει το μεταγωγέα με τον ελεγκτή και επιτρέπει την επικοινωνία των δύο και το πρωτόκολλο Openflow που παρέχει ένα πρότυπο ώστε να επικοινωνεί ο ελεγκτής με τον μεταγωγέα. Ουσιαστικά μέσω του Openflow ο ελεγκτής καθορίζει την λειτουργία του μεταγωγέα. [4]

3.2 Πίνακες Ροής και Καταχωρίσεις Ροής

Χρησιμοποιώντας το Openflow ο ελεγκτής μπορεί να προσθέσει, να ενημερώσει και να διαγράψει καταχωρίσεις ροής -flow entries από τους πίνακες ροής - flow tables τόσο ανταποκρινόμενος στον εκάστοτε τύπο των πακέτων όσο και προκαθορισμένα.



Εικόνα 3. 2 Πίνακες ροής στον Openflow μεταγωγέα. [2]

Κάθε πίνακας ροής στον μεταγωγέα αποτελείται από ένα σύνολο καταχωρίσεων ροής - flow entries και κάθε καταχώρηση ροής, αποτελείται από τα πεδία των καταμετρητών - counters fields, συσχετιζόμενα πεδία ή πεδία αντιστοίχησης - match fields και τα πεδία εντολών - instructions field που περιέχουν εντολές ανάλογες του τύπου των πακέτων.

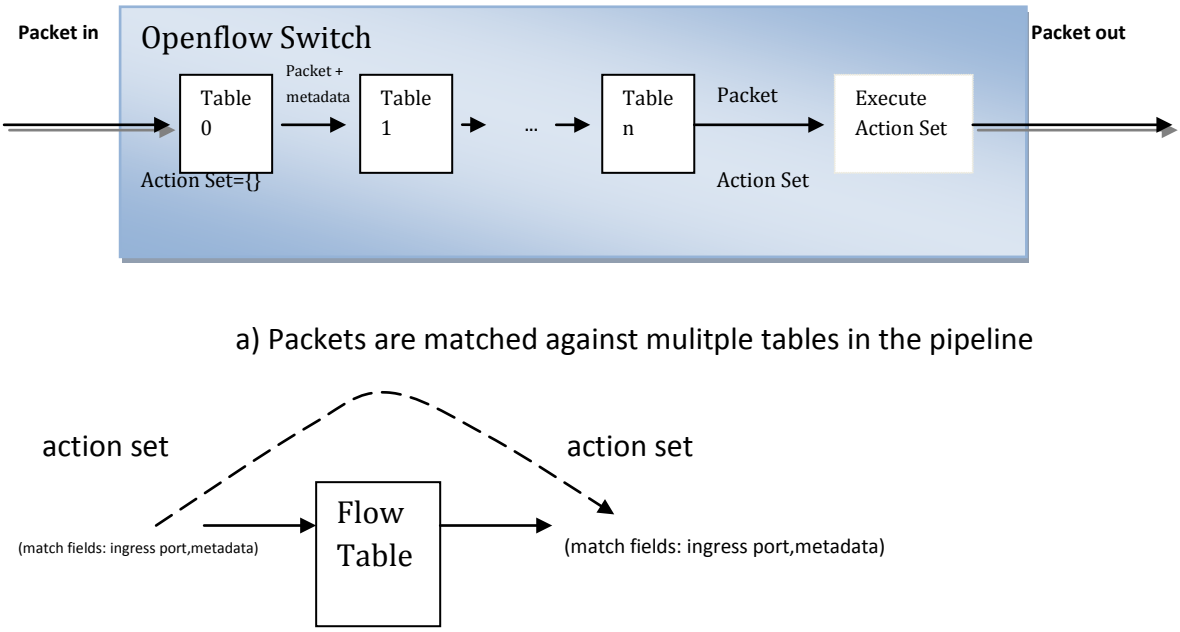
Ingress Port	Ether source	Ether dst	Ether type	VLAN id	VLAN priority	IP src	IP dst	IP proto	IP ToS bits	TCP/UDP src port	TCP/UDP dst port

Εικόνα 3. 3 Τα πεδία που ελέγχονται για να γίνει η διαδικασία αντιστοίχησης ή ταυτοποίησης σε σχέση με τις καταχωρίσεις ροής που εμπεριέχονται σε ένα πίνακα ροής. [33]

Από τον μεταγωγέα γίνεται η προσπάθεια αντιστοίχησης της ληφθείσας ροής με τους πίνακες ροής που είναι εγγεγραμμένοι σε αυτόν. Εάν μια αντιστοίχιση εντοπισθεί, τότε οι οδηγίες που σχετίζονται με την συγκεκριμένη καταχώρηση εκτελούνται. Εάν δεν εντοπισθεί αντιστοίχηση στον πίνακα ροής η επεξεργασία της ροής εξαρτάται από τον πίνακα αστοχίας καταχωρίσεων ροής - table miss flow entry του μεταγωγέα, για παράδειγμα τα πακέτα μπορεί να προωθούνται στον ελεγκτή μέσω του Openflow καναλιού, να απορρίπτονται είτε να μετακινούνται για αντιστοίχηση στον επόμενο διαθέσιμο πίνακα ροής του μεταγωγέα.

Τα πακέτα μετακινούνται με βάση την εκάστοτε καταχώρηση ροής σε μια θύρα εξόδου του μεταγωγέα από μια θύρα εισόδου. Οι θύρες στους μεταγωγείς Openflow μπορούν να είναι τριών τύπων: φυσικές θύρες του μεταγωγέα (θύρες που αντιστοιχούν στο υλικό - hardware του μεταγωγέα), εικονικές θύρες (θύρες που δεν αντιστοιχίζονται απευθείας με το hardware του μεταγωγέα) και δεσμευμένες θύρες. Το τελευταίο είδος θύρας μπορεί να χρησιμοποιείται για συγκεκριμένες λειτουργίες προώθησης, όπως αποστολή πακέτων στον ελεγκτή, μαζική προώθηση πακέτων σε όλες τις συσκευές (πλημμύρα) ή και για την χρήση κλασικών μεθόδων προώθησης που δεν συσχετίζονται με το Openflow.

Ακόμα, ο μεταγωγέας μπορεί να χρησιμοποιήσει επεξεργασία καναλιού - pipeline processing για πιο γρήγορη και μαζικού χαρακτήρα επεξεργασία της κίνησης στο δίκτυο. Μια συγκεκριμένη καταχώρηση ροής μπορεί να αντιστοιχηθεί με μια δέσμη ενεργειών με βάση τον πίνακα ροής - flow table. Αυτές οι ενέργειες μπορούν να κατευθύνουν το πακέτο σε ένα άλλο πίνακα ροής όπου η ίδια διαδικασία μπορεί να επαναληφθεί ξανά και ξανά. Πάντως οι πίνακες σε ένα Openflow switch ξεκινάνε από το 0 και σε περίπτωση pipeline processing - επεξεργασίας καναλιού το πακέτο μπορεί να μετακινηθεί μόνο από μικρότερο σε μεγαλύτερο πίνακα ροής συνεπώς η διαδικασία pipeline processing μπορεί να πάει μόνο προς τα εμπρός και όχι προς τα πίσω.



Εικόνα 3. 4 Έλεγχος πακέτων σε σχέση με τις εγγραφές που βρίσκονται στους πίνακες του μεταγωγέα. [33]

3.1.1 Πίνακες Ροής

Ένας πίνακας ροής αποτελείται από καταχωρίσεις ροής. Κάθε καταχώρηση ροής περιέχει:

- match fields - πεδία αντιστοίχισης για να μπορεί να αντιστοιχίσει τα εισερχόμενα πακέτα. Περιέχουν τη θύρα εισόδου, επικεφαλίδες πακέτων αλλά και μεταδεδομένα στην περίπτωση που έχουν προέλθει από ένα προηγούμενο πίνακα ροής.

- priority - προτεραιότητα, μας δίνει πληροφορίες για την προτεραιότητα με την οποία θα πρέπει ο μεταγωγέας να επεξεργαστεί την ροή.

- counters - μετρητές που ενημερώνονται όταν τα πακέτα ταιριάζουν με τις εγγραφές στου πίνακες ροής

- instructions - οδηγίες για να αλλάξουν την δέσμη ενεργειώνου λαμβάνουν χώρα ανά πακέτο

- timeouts - χρονικό όριο πάνω στο οποίο η κίνηση έχει λήξει στον μεταγωγέα.

- cookie - δεδομένα που θέτονται από τον ελεγκτή μπορεί να αφορούν στατιστικά στοιχεία κίνησης, αλλαγές στην κίνηση της ροής ή και διαγραφές ροής.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Εικόνα 3. 5 Κύρια πεδία μιας καταχώρησης ροής σε έναν πίνακα ροής [33]

3.1.2 Ταυτοποίηση

Κατά την είσοδο ενός πακέτου ο μεταγωγέας Openflow εφαρμόζει τις παρακάτω διεργασίες που φαίνονται στο 1ο διάγραμμα ροής . Ο μεταγωγέας ξεκινά κάνοντας ένα έλεγχο του πρώτου πίνακα ροής και με βάση την επεξεργασία καναλιού - pipeline processing μπορεί να κάνει έλεγχο των καταχωρίσεων ροής σε πολλούς διαφορετικούς πίνακες ροής. Ουσιαστικά μέσω αυτής της διαδικασίας αναζητούνται πεδία στις καταχωρίσεις ροής flow entries των πινάκων ροής - flow tables που ταυτίζονται με τα δεδομένα της επικεφαλίδας του πακέτου που εισήρθε στην συσκευή προώθησης, επίσης μεταδεδομένα μπορούν να χρησιμοποιηθούν ώστε να μεταφερθεί πληροφορία μεταξύ των πινάκων του μεταγωγέα. Εάν ο πίνακας ροής υπό εξέταση έχει πεδίο που έχει την τιμή ΑΝΥ τότε αυτό ταυτίζεται με όλες τις πιθανές τιμές στην επικεφαλίδα του πακέτου.

Όταν το πακέτο ταυτιστεί με κάποιο συγκεκριμένο πίνακα ροής μόνο η υψηλότερη καταχώρηση ροής - flow entry επιλέγεται. Οι καταμετρητές - counters που σχετίζονται με την συγκεκριμένη ροή πρέπει να προσαρμοστούν εκ νέου, δηλαδή να ενημερωθούν, και το σύνολο των ενεργειών για την συγκεκριμένη ροή πρέπει να εφαρμοστεί. Συγκεκριμένα δυο είναι οι κύριες ενέργειες η προώθηση και η απόρριψη του πακέτου.

Αναλυτικά η προώθηση περιλαμβάνει τις παρακάτω επιλογές:

1. All, Προώθηση του πακέτου σε όλες τις διεπαφές εκτός από την διεπαφή από την οποία εισήχθη το πακέτο στον μεταγωγέα

2. Controller, Ενθυλάκωση και αποστολή στον ελεγκτή

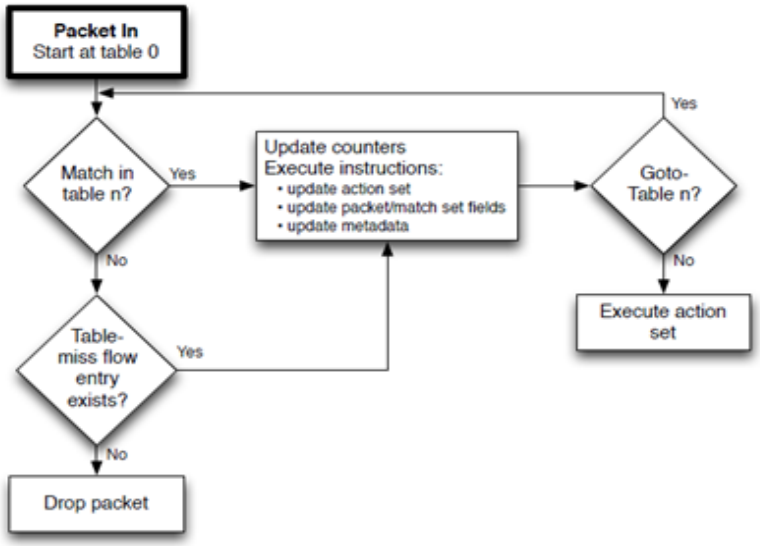
3. Local, Αποστολή του πακέτου στην στοίβα του μεταγωγέα

4. Table, Εκτέλεση των ενεργειών που υπάρχουν στον πίνακα ροής.

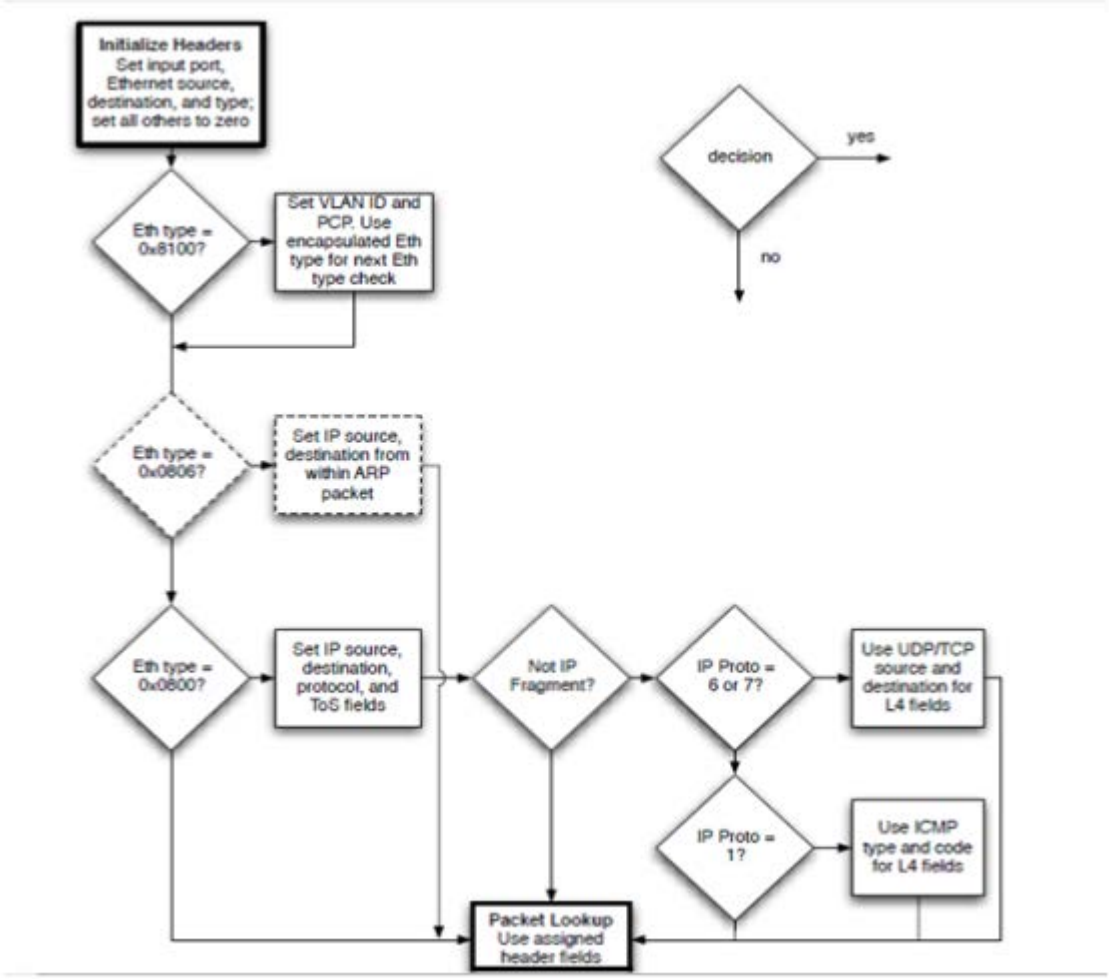
5. In Port, Αποστολή του πακέτου πίσω στην θύρα από την οποία εισήρθε

Εάν το πακέτο δεν αντιστοιχηθεί με καμία καταχώρηση ροής στους υπό σύγκριση πίνακες τότε το πακέτο είτε μεταφέρεται στον ελεγκτή όπως καθορίζει η προδιαγραφή μεταγωγέα Openflow 1.0.0 switch specification- controller , είτε απορρίπτεται όπως καθορίζει η προδιαγραφή 1.0.4 ή σε περίπτωση που είναι επιθυμητό από τον διαχειριστή του δικτύου ορίζεται εκ νέου η συμπεριφορά του μεταγωγέα χρησιμοποιώντας το πρωτόκολλο ρύθμισης του Openflow - Openflow Configuration Protocol. [33]

Υπάρχουν και οι προαιρετικές ενέργειες όπως η αλλαγή σε δεδομένα που υπάρχουν στην επικεφαλίδα του πακέτου όπως είναι τα δεδομένα για τη προτεραιότητα ή για την IP διεύθυνση προορισμού. Η τελευταία περίπτωση είναι πολύ χρήσιμη σε περιπτώσεις μεγάλων δικτύων στα οποία θέλουμε να έχουμε ίσο κατά το δυνατό επιμερισμό του φόρτου κίνησης. Μια άλλη προαιρετική ενέργεια είναι η αποστολή του πακέτου σε μια ουρά που βρίσκεται σε μια συγκεκριμένη θύρα. [33]



Εικόνα 3. 6 Διάγραμμα ροής που δείχνει την ροή των πακέτων σε ένα μεταγωγέα Openflow 1.0.4 [33]



Εικόνα 3. 7 Διάγραμμα ροής που παρουσιάζει την ανάλυση των επικεφαλίδων για την αντιστοίχηση με τους πινάκες ροής. [34]

Διαγραφή καταχωρίσεων ροής - flow entry removal από πίνακες ροής μπορούν να συμβούν με τρείς διαφορετικούς τρόπους. Ο πρώτος τρόπος είναι να στείλει αίτημα για την διαγραφή της καταχώρησης ο ελεγκτής. Ο δεύτερος τρόπος διαγραφής είναι ο μηχανισμός λήξης που έχει ο μεταγωγέας. Ο τρίτος τρόπος είναι η προαιρετική επιλογή για απόρριψη καταχώρησης του ίδιου του μεταγωγέα. Ο μηχανισμός λήξης του μεταγωγέα λειτουργεί άσχετα από τον μεταγωγέα και βασίζεται στην κατάσταση και τις ρυθμίσεις των καταχωρίσεων ροής. Κάθε καταχώρηση ροής έχει τα πεδία idle_timeout και hard_timeout συσχετισμένα με αυτή. Εάν το hard _timeout πεδίο είναι μη μηδενικό ο μεταγωγέας πρέπει να σημειώσει την ώρα άφιξης της καταχώρησης ροής - flow entry καθώς θα πρέπει να απορρίψει την καταχώρηση στην συνέχεια. Αυτό που θα συμβεί είναι ότι θα αφαιρεθεί μετά από το πέρας του συγκεκριμένου χρονικού ορίου άσχετα το με πόσα πακέτα έχει αντιστοιχηθεί. Εάν το idle_timeout είναι μη μηδενικό τότε ο μεταγωγέας θα πρέπει να σημειώσει τη χρονική στιγμή κατά την οποία έλαβε το τελευταίο πακέτο από την συγκεκριμένη ροή μιας και θα πρέπει να διαγράψει την καταχώρηση ροής λόγω μη λήψης νέου πακέτου του ίδιου τύπου ροής στο προκαθορισμένο χρόνο. Επίσης ο ελεγκτής μπορεί να αφαιρέσει καταχωρίσεις ροής από πίνακες ροής αποστέλλοντας εντολές διαγραφής με τα μηνύματα OFPFC_DELETE ή OFPFC_DELETE_STRICT. Τέλος διαγραφή καταχωρίσεων ροής μπορεί να συμβεί λόγω του ότι ο μεταγωγέας θέλει να ανακτήσει κάποιους πόρους του ή όταν αφαιρείται μια ολόκληρη ομάδα καταχωρίσεων ροής. Ο μεταγωγέας θα πρέπει να ελέγξει την OFPFF_SEND_FLOW_REM σημαία - flag της κάταχώρησης ροής και στέλνει μήνυμα διαγραφής της ροής στον ελεγκτή, το μήνυμα περιέχει πλήρη περιγραφή της καταχώρησης ροής, τον λόγο της διαγραφής, την διάρκεια ζωής της καταχώρησης ροής κατά την στιγμή της διαγραφής της και στατιστικά στοιχεία. [33]

3.1.3 Group Table - Πίνακες ομαδοποίησης

Οι πίνακες ομαδοποίησης αποτελούνται από καταχωρίσεις ομάδων - group entries. Η ικανότητα που έχει μια καταχώρηση ροής να δείχνει προς μια ομάδα ουσιαστικά επιτρέπει στο Openflow να προωθεί τα πακέτα στις συσκευές με νέες μεθόδους.

Αναλυτικά τα πεδία του πίνακα ομαδοποίησης - group entry είναι:

1. Το αναγνωριστικό ομάδας - group identifier, είναι ένας μη προσημασμένος ακέραιος αριθμός μεγέθους 32bit που καθορίζει μονοσήμαντα τη συγκεκριμένη καταχώρηση - entry.

2. Ο τύπος της ομάδας - group type, που καθορίζει την σημασιολογία της ομάδας δηλαδή δείχνει ποιες εντολές θα εκτελεστούν για τα πακέτα που ανήκουν σε αυτήν.

3. Οι μετρητές – counters, αριθμός που ενημερώνεται όταν πακέτα επεξεργάζονται με βάση μια ομάδα. Οι ομάδες εντολών - action buckets, μια λίστα εντολών στην οποία κάθε εντολή περιέχει ένα σύνολο ενεργειών που θα εφαρμοστεί στο προς επεξεργασία πακέτο.

Group Identifier	Group Type	Counters	Action Buckets

Εικόνα 3. 8 Κύρια πεδία μιας καταχώρησης ροής σε ένα πίνακα ροής. [33]

3.1.4 Τύποι Ομάδων - Group types

Κάθε μεταγωγέας δεν είναι απαραίτητο να υποστηρίζει όλους τους τύπους ομάδων που αναφέρονται παρακάτω. Απαραίτητο είναι να υποστηρίζει μόνο αυτούς τους τύπους που έχουν την σήμανση "Required".

1. Required - all: Εκτελούνται όλες οι εντολές των ομάδων, χρησιμοποιείται συνήθως για ευρυεκπομπή (broadcasting) και πολυεκπομπή (multicasting).

2. Optional - select: Εκτελείται μόνο μια εντολή από τη ομάδα εντολών.

3. Required - indirect: Εκτελείται μια μόνο προκαθορισμένη εντολή.

4. Optional - fast failover: Εκτελείται η πρώτη εντολή σε ισχύ.

3.2 Openflow κανάλι - Openflow Channel

Το Openflow κανάλι είναι η διεπαφή που συνδέει κάθε Openflow μεταγωγέα με ελεγκτή. Μέσω αυτής της διεπαφής ο ελεγκτής ρυθμίζει, διαχειρίζεται και αποστέλλει πακέτα προς τον μεταγωγέα. Το κανάλι συνήθως έχει κρυπτογράφηση TLS αλλά κάποιες φορές το κανάλι τρέχει πάνω από απλό TCP.

3.2.1 Openflow επισκόπηση

Το πρωτόκολλο OpenFlow υποστηρίζει τρεις τύπους μηνυμάτων: μηνύματα από τον ελεγκτή στον μεταγωγέα, ασύγχρονα μηνύματα και συμμετρική επικοινωνία, το καθένα αποτελείται από πολλαπλά υπό-είδη. Τα μηνύματα που αποστέλλονται από τον ελεγκτή προς τον μεταγωγέα χρησιμοποιούνται για να διαχειριστεί ο ελεγκτής άμεσα ή να επιθεωρήσει την κατάσταση του μεταγωγέα. Τα ασύγχρονα μηνύματα ξεκινούν από τον μεταγωγέα και χρησιμοποιούνται για να ενημερώσει τον ελεγκτή για γεγονότα που λαμβάνουν χώρα στο δίκτυο και για αλλαγές στην κατάσταση του μεταγωγέα. Τα συμμετρικά μηνύματα ξεκινούν είτε από το διακόπτη ή τον ελεγκτή και αποστέλλονται χωρίς αίτημα. Οι τύποι που χρησιμοποιούνται από OpenFlow περιγράφεται κατωτέρω.

Μηνύματα από τον ελεγκτή προς τον μεταγωγέα

Τα μηνύματα από τον ελεγκτή προς τον μεταγωγέα ξεκινάνε από τον ελεγκτή και μπορεί να απαιτούν απόκριση από τον μεταγωγέα.

1. Features: Ο ελεγκτής μπορεί να ζητήσει την ταυτότητα, τα χαρακτηριστικά και τις βασικές δυνατότητες του μεταγωγέα στέλνοντας ένα μήνυμα features request και ο μεταγωγέας μπορεί να απαντά στο αίτημα με features response. Αυτή η διαδικασία λαμβάνει χώρα κατά τη στιγμή της εγκαθίδρυσης της σύνδεσης του Openflow καναλιού.

2. Configuration: Ο ελεγκτής ορίζει και να ζητά τις παραμέτρους λειτουργίας του μεταγωγέα. Ο μεταγωγέας αποκρίνεται μόνο στα ερωτήματα του ελεγκτή
3. Modify-State: Τα μηνύματα αυτού του τύπου στέλνονται από τον ελεγκτή για να διαχειρίζεται την κατάσταση του μεταγωγέα. Ο βασικός τους σκοπός είναι να

προσθέτουν, να διαγράφουν και να διαφοροποιούν τις group entries στους Openflow πίνακες και να καθορίζουν τις ιδιότητες των θυρών του μεταγωγέα.

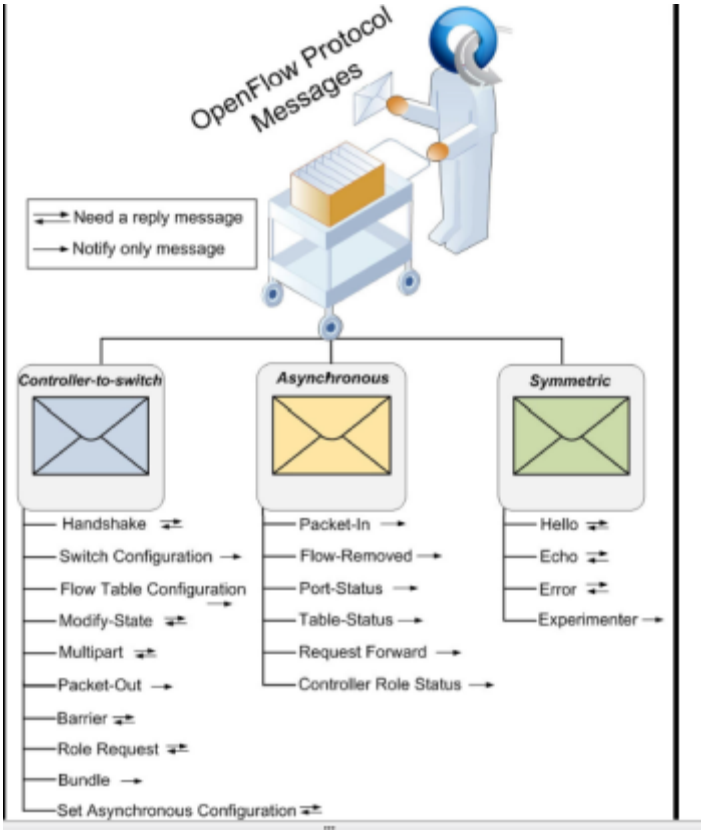
4. Read-State: Τα μηνύματα αυτού του τύπου χρησιμοποιούνται για να συγκεντρώσει ο ελεγκτής από τον μεταγωγέα διάφορες πληροφορίες όπως την τρέχουσα ρύθμιση του μεταγωγέα, στατιστικά στοιχεία καθώς και ιδιότητες του μεταγωγέα.

5. Packet-out: Αυτά χρησιμοποιούνται από τον ελεγκτή για να στέλνει πακέτα μέσω καθορισμένης θύρας του μεταγωγέα καθώς και να προωθεί πακέτα που έχουν ληφθεί μέσω μηνυμάτων Packet in. Έπίσης το μήνυμα περιέχει μια λίστα από ενέργειες που θα εφαρμοστούν με την σειρά που ορίζεται, μια άδεια λίστα ενεργειών απορρίπτει το πακέτο.

6. Barrier: Τα Barrier request/reply messages χρησιμοποιούνται από τον ελεγκτή για να διασφαλίσει ότι οι παράμετροι των πακέτων έχουν τηρηθεί και για να λάβει ειδοποιήσεις για ολοκληρωμένες λειτουργίες.

7. Role-Request: Τα Role-Request μηνύματα χρησιμοποιούνται από τον ελεγκτή για να καθορίσουν τον ρόλο του Openflow καναλιού. Αυτό είναι πολύ χρήσιμο όταν ο μεταγωγέας συνδέεται με πολλαπλούς ελεγκτές.

8. Asynchronous-Configuration: Τα Asynchronous-Configuration messages χρησιμεύουν στον ελεγκτή για να καθορίσουν, να προσθέσουν δηλαδή, ένα επιπλέον φίλτρο στα ασύγχρονα μηνύματα που θέλει να λαμβάνει από το Openflow κανάλι. Αυτό είναι πολύ χρήσιμο όταν ο μεταγωγέας συνδέεται με πολλαπλούς ελεγκτές. [33]



Εικόνα 3. 9 Τύποι μηνυμάτων Openflow. [2]

Ασύγχρονα μηνύματα

1. Packet in συμβάντα : μεταφέρουν τον έλεγχο ενός πακέτου στον ελεγκτή. Για όλα τα πακέτα που προωθούνται σε συγκεκριμένη θύρα του ελεγκτή χρησιμοποιώντας μια καταχώρηση ροής ή την αστοχία καταχώρησης ροής - table miss entry ένα packet in συμβάν πάντα αποστέλλεται στον ελεγκτή από τον μεταγωγέα. Στις περιπτώσεις που ο μεταγωγέας έχει διαθέσιμη προσωρινή μνήμη - buffer και το packet in συμβάν είναι ρυθμισμένο ώστε να αποθηκεύει προσωρινά πακέτα τότε τα packet in συμβάντα περιλαμβάνουν μόνο ένα κομμάτι της κεφαλίδας του πακέτου και ένα νούμερο αναγνώρισης - buffer ID χρησιμοποιείται από τον ελεγκτή όταν ο μεταγωγέας είναι έτοιμος να προωθήσει το πακέτο.

2. Packet out μηνύματα από τον ελεγκτή αυτόματα λήγουν μετά την πάροδο ενός συγκεκριμένου χρονικού ορίου.

3. Flow removed: Τα μηνύματα αυτά ενημερώνουν για την αφαίρεση μιας καταχώρησης ροής από ένα πίνακα ροής. Αυτού του είδους τα μηνύματα αποστέλλονται για καταχωρήσεις ροής με OFPFF_SEND_FLOW_REM flag set. Δημιουργούνται σαν αποτέλεσμα αιτήσεων για διαγραφή ροής ή σε περιπτώσεις που ο μεταγωγέας λήγει τη ροή με βάση για παράδειγμα το χρονικό όριο στο οποίο λήγει η συγκεκριμένη ροή - flow timeout.

4. Port status : Τα μηνύματα αυτά ενημερώνουν τον ελεγκτή για την αλλαγή μιας θύρας για παράδειγμα όταν μια σύνδεση έχει πέσει - link down ή όταν ορίζεται έτσι από τον διαχειριστή του δικτύου. [33]

Συμμετρικά μηνύματα

Τα μηνύματα symmetric είναι αμφίδρομα και λαμβάνουν χώρα χωρίς αιτήματα.

1. Hello: Τα μηνύματα αυτά ανταλλάσσονται μεταξύ του μεταγωγέα και του ελεγκτή κατά την εγκαθίδρυση της σύνδεσης.

2. Echo : Τα μηνύματα αυτά μπορούν να σταλούν τόσο από ελεγκτή όσο και από τον μεταγωγέα και πρέπει να περιέχουν echo reply. Χρησιμεύουν για να δείξουν την ικανότητα σύνδεσης μεταξύ μεταγωγέα και ελεγκτή και μπορούν να χρησιμοποιηθούν για να μετρηθεί η καθυστέρηση και το εύρος ζώνης.

3. Error: Τα μηνύματα αυτά χρησιμοποιούνται από τον μεταγωγέα ή τον ελεγκτή για να ειδοποιήσουν την άλλη πλευρά της σύνδεσης για προβλήματα. Χρησιμοποιούνται από τον ελεγκτή για να φανερώσει την αποτυχία ενός αιτήματος που ξεκίνησε από τον ελεγκτή.

4. Experimenter: Τα μηνύματα αυτά προσφέρουν ένα συγκεκριμένο τρόπο για τους μεταγωγείς Openflow ώστε να προσφέρουν πρόσθετη λειτουργικότητα, περισσότερες δυνατότητες όσον αφορά την δημιουργία και εξέλιξη πειραμάτων με χρήση των μηνυμάτων Openflow. [33]

3.2.2 Διαχείριση μηνυμάτων

Το Openflow πρωτόκολλο παρέχει αξιόπιστη παράδοση και επεξεργασία μηνυμάτων αλλά δεν παρέχει αναγνωρίσεις - aknowledgemnets ή την εξασφαλίση της επεξεργασίας του μηνύματος -ordered message processing.

Παράδοση μηνύματος : Τα μηνύματα έχουν εγγύηση παράδοσης εκτός από την περίπτωση που το Openflow κανάλι αποτύχει εντελώς στην οποία περίπτωση ο ελεγκτής δεν πρέπει να υποθέσει τίποτα σε σχέση με την κατάσταση του μεταγωγέα.

Επεξεργασία μηνύματος: Οι μεταγωγείς πρέπει να επεξεργάζονται κάθε μήνυμα που λαμβάνουν από τον ελεγκτή και πιθανότατα να δημιουργούν μια απάντηση. Εάν ο μεταγωγέας δεν μπορεί να επεξεργαστεί εξολοκλήρου ένα μήνυμα που έχει λάβει από τον ελεγκτή πρέπει να στείλει ένα μήνυμα σφάλματος.

Διάταξη ή Ταξινόμηση μηνυμάτων: Η ομαλή διάταξη μπορεί να διαπιστωθεί με την χρήση μηνυμάτων φράγματος -barrier. Σε περίπτωση έλλειψης τέτοιων μηνυμάτων ο μεταγωγέας μπορεί να αναδιοργανώσει τα μηνύματα. ώστε να μεγενθύνει την απόδοση του.

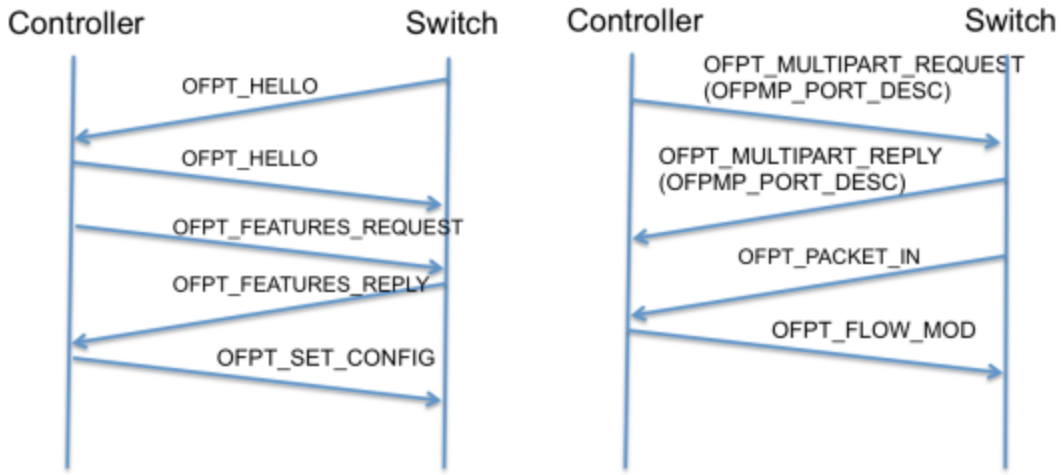
3.2.3 Συνδέσεις Openflow καναλιού - channel

Το Openflow κανάλι χρησιμοποιείται για να ανταλλάσονται μυνήματα μεταξύ του Openlfow μεταγωγέα και του Openflow ελεγκτή. Ένας τυπικός Openflow ελεγκτής μπορεί να οργανώνει και να διαχειρίζεται πολλαπλά κανάλια Openlfow καθένα από τα οποία μπορεί να ανήκει σε άλλον μεταγωγέα. Ο Openflow ελεγκτής διαχειρίζεται τον μεταγωγέα απομακρυσμένα πάνω σε ένα ή περισσότερα δίκτυα. Το Openflow κανάλι συνήθως αρχικοποιείται σαν μια απλή μοναδική σύνδεση δικτύου μεταξύ μεταγωγέα και του ελεγκτή χρησιμοποιώντας TLS ή απλά TCP.

Εγκαθίδρυση σύνδεσης

Ο μεταγωγέας πρέπει να είναι ικανός να δημιουργήσει και να ιδρύσει μια σύνδεση με τον ελεγκτή σε μια καθορισμένη από τον χρήστη αλλά σταθερή IP χρησιμοποιώντας μια

θύρα καθορισμένη από τον χρήστη ή την προκαθορισμένη του θύρα. Εάν ο μεταγωγέας ρυθμιστεί ώστε να επικοινωνεί με την IP διεύθυνση του ελεγκτή, ο μεταγωγέας ξεκινά μια σύνδεση TLS ή TCP μεταξύ τους. Ο μεταγωγέας θα πρέπει να αναγνωρίσει την κίνηση ως τοπική ώστε να αρχίσει να την συγκρίνει με τους πίνακες ροής που διαθέτει. Η πρώτη κίνηση μόλις η σύνδεση επιτευχθεί είναι η αποστολή από τις δυο άκρες ενός Hello μηνύματος - μήνυμα καλωσορίσματος. Στο παρακάτω σχήμα φαίνεται η ακολουθία των μηνυμάτων κατά την εγκατάσταση της σύνδεσης.



Εικόνα 3. 10 Σχηματική αναπαράσταση των μηνυμάτων που ανταλλάσονται μεταξύ ελεγκτή και μεταγωγέα. Τα μηνύματα αφορούν το πρωτόκολλο Openflow 1.3 που είναι το πιο πρόσφατο πρωτόκολλο που υποστηρίζεται από τους κατασκευαστές μεταγωγέων. [41]

Κεφάλαιο 4
Mininet

Οι περισσότερο χρησιμοποιούμενες πλατφόρμες για έρευνα στον χώρο των δικτύων είναι οι προσομοιωτές, οι εξομοιωτές και τα testbeds. Το πρώτο ζητούμενο από ένα τέτοιο σύστημα είναι να μπορεί να παράγει όσο το δυνατόν πιο ρεαλιστικά αποτελέσματα βασισμένο σε πραγματικά δεδομένα. Δηλαδή το επιθυμητό είναι τα αποτελέσματα της πλατφόρμας να μην απέχουν από τα αποτελέσματα τα οποία θα έδινε το δίκτυο αν υπήρχε στην φυσική του μορφή. Το δεύτερο ζητούμενο είναι αυτά τα αποτελέσματα να μπορούν να αναπαραχθούν οποιαδήποτε στιγμή από οποιανδήποτε ερευνητή επιθυμεί, με οποιαδήποτε υπολογιστή και το ανάλογο - κατάλληλο λογισμικό και να καταλήγουν στα ίδια ακριβώς συμπεράσματα.

4.1 Προσομοιωτές

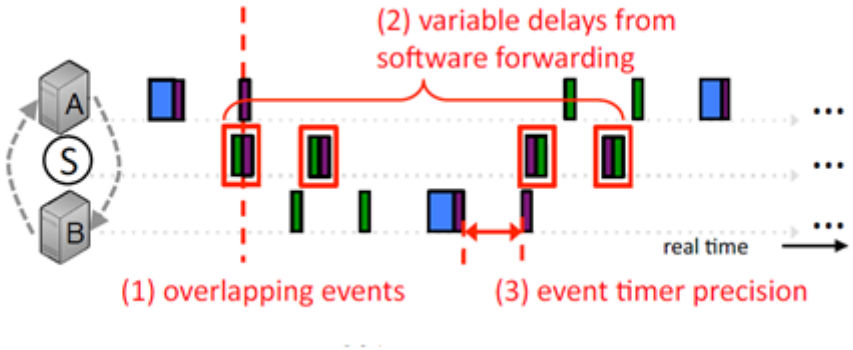
Οι προσομοιωτές (simulators) επιλέγονται από πολλούς για την διεξαγωγή πειραμάτων γιατί με την βοήθεια τους είναι εύκολο να αναπαραχθούν τα αποτελέσματα του κάθε σεναρίου ανεξάρτητα από το μηχάνημα στο οποίο το σενάριο τρέχει. Το πρόβλημα όμως των προσομοιωτών είναι το εξής, τα αποτελέσματα τους πολλές φορές δεν αντιστοιχούν στα αποτελέσματα των πειραμάτων που γίνονται με φυσικό υλικό ενώ ο κώδικας δεν είναι ο ίδιος με αυτόν που τρέχει για παράδειγμα σε ένα πραγματικό εξυπηρετητή. Επίσης οι προσομοιωτές καθορίζονται από διακριτά γεγονότα γεγονός που σημαίνει ότι το σύστημα αναπαρίσταται από μία αλληλουχία διακριτών συμβάντων, τα οποία γίνονται σε συγκεκριμένες χρονικές στιγμές και προκαλούν μεταβολές στο σύστημα.

4.2 Testbeds

Με τα testbeds το θέμα των ρεαλιστικών αποτελεσμάτων εξαλείφεται. Υπάρχουν ακόμα, διαθέσιμα testbeds τα οποία προσφέρουν απομόνωση των πόρων και έτσι το ένα πείραμα δεν επηρεάζει το άλλο. Το πρόβλημα όμως είναι ότι είναι δύσκολο αν όχι αδύνατο να αναπαραχθούν τα αποτελέσματα ενός πειράματος. Επίσης πολύ σημαντικό είναι ότι ο ερευνητής δεν μπορεί να αλλάξει εύκολα για παράδειγμα το λογισμικό σε ένα μεταγωγέα και έτσι νέα πρωτόκολλα ή νέες ιδέες δεν μπορούν να εξετασθούν. Ακόμα κοστίζει αρκετά να γίνει αναπαράσταση των επιθυμητών σεναρίων σε μεγάλη κλίμακα. Τέλος μιας και τα υλικά για την δημιουργία των πειραμάτων μπορεί να μην διαθέσιμα στο μέλλον τα αποτελέσματα δεν μπορούν πάντα να αναπαραχθούν.

4.3 Εξομοιωτές

Οι εξομοιωτές από την άλλη τρέχουν πραγματικό κώδικα με πραγματική κίνηση δικτύου και τα γεγονότα δεν είναι διακριτά. Τα γεγονότα συμβαίνουν σε συνεχής χρόνο ενώ οι εικονικές συσκευές που χρησιμοποιούνται για την δημιουργία του επιθυμητού εικονικού δικτύου κοστίζουν ελάχιστα. Το μειονέκτημα στους εξομοιωτές είναι ότι τα αποτελέσματα τους είναι πολλές φορές εξαρτώμενα ή ανάλογα του φόρτου δικτύου. Δεν προσφέρουν δηλαδή την επιθυμητή απομόνωση κατά την διάρκεια του πειράματος ώστε τα αποτελέσματα που θα συλλεχθούν από αυτό να είναι ανεξάρτητα της κίνησης στο δίκτυο και να ανταποκρίνονται στην πραγματικότητα.



Εικόνα 4. 1 Αιτίες έλλειψης πιστότητας εξομοιωτών: Η αλληλεπικάλυψη γεγονότων, οι καθυστερήσεις και η έλλειψη ακριβείας του χρονομετρητή.

4.4 Χαρακτηριστικά Ιδανικής Πλατφόρμας

Παρακάτω αναφέρονται τα απαραίτητα χαρακτηριστικά μιας ιδανικής πλατφόρμας για έρευνα στα δίκτυα. Συγκεκριμένα η πλατφόρμα θα πρέπει να λειτουργεί ρεαλιστικά, το υπό εξομοίωση σύστημα πρέπει να έχει την ίδια συμπεριφορά με το πραγματικό σύστημα αν αυτό μελετιόταν και κατασκευαζόταν από υλικό, καθώς και να τρέχει τον ίδιο κώδικα. Η συμπεριφορά του συστήματος χρονικά (χρονικές αποκρίσεις κτλ) δεν πρέπει να διαφέρει ή πρέπει να αποκλίνει ελάχιστα από το πραγματικό.

Ταυτόχρονα το υπό εξέταση σύστημα θα πρέπει να μπορεί να δημιουργήσει και να λάβει πραγματική κίνηση δικτύου από και προς το διαδίκτυο ή από χρήστες ή συστήματα στο τοπικό δίκτυο. Ακόμα θα πρέπει να είναι ευέλικτο ώστε να μπορεί να υποστηρίξει μεγάλο εύρος πειραμάτων. Συγκεκριμένα θα πρέπει να υποστηρίζει με ευκολία την δημιουργία πειραμάτων μεγάλης κλίμακας με πολλούς υπολογιστές, δρομολογητές και μεταγωγείς καθώς και την δημιουργία οποιασδήποτε τοπολογίας. Τέλος θα πρέπει να είναι εύκολο να αναπαράγεις το πείραμα στην πλατφόρμα ενώ ταυτόχρονα θα πρέπει να γίνονται όλα τα παραπάνω, με χαμηλό κόστος ώστε να διευκολυνθεί η εκπαιδευτική και ερευνητική διαδικασία.

4.5 Συμπεράσματα Σύγκρισης Πλατφορμών

Συνοψίζοντας τα παραπάνω, οι προσομοιωτές δεν έχουν ρεαλιστικά αποτελέσματα όσον αφορά τις λειτουργικές διαδικασίες που λαμβάνουν χώρα σε ένα δίκτυο, τα testbed έχουν μειονεκτήματα όσον αφορά την δημιουργία νέων τοπολογιών, ενώ οι εξομοιωτές δεν παρουσιάζουν με ρεαλισμό τις χρονικές αποκρίσεις των συστημάτων. Αυτήν την στιγμή υπάρχουν διάφορα διαθέσιμα προγράμματα για έρευνα στο SDN με το Mininet και το NS να υπερισχύουν έναντι των άλλων τις περισσότερες φορές και να προτιμούνται στην ερευνητική διαδικασία. Στο παρακάτω σχήμα αναφέρονται τα βασικά χαρακτηριστικά των διαθέσιμων προγραμμάτων.

Plaforfm	Mininet	EstiNet	Ns-3	Trema
Last Version	2.1.0+	8.0	3.19	0.4.6
Vendor	Stanford Univeristy. ON Lab	EstiNet Techonologies Inc.	Ns-3 Porject	NEC Corporation
Web site	www.mininet.org	www.estinet.com	www.nsnam.org	Io/trema/trema.github
Operating System	Ubuntu, Fedora	Linux, Fedora(14,17)	GNU/Linux,Windows,Free BSD,Mac,OSX	GNU/Debian, Ubuntu, Fedora
Openflow Versions	1.0-1.3	1.0,1.1,1.3, 1.3.2	0.89	1.0,1.3,1.3.1
GUI	VND, Miniedit	EstiNet GUI	VND	VND
Emulation mode	Yes	Yes	No	Yes
Simulation mode	No	Yes	Yes	No
Free or Proprietary	Free	Proprietary	Free	Free

Εικόνα 4. 2 Σύγκριση Πλατφορμών [2]

Το Mininet Hi Fi σύμφωνα με τον Heller [8] αίρει όλες τις προαναφερθείσες αντιξοότητες των εξομοιωτών και προσφέρει το ιδανικό μέσο για την διεξαγωγή πειραμάτων από

ερευνητές μιας και δίνει ρεαλιστικά αποτελέσματα, προσφέρει πιστότητα όσον αφορά τις χρονικές αποκρίσεις του υπό εξέταση δικτύου, είναι πρόγραμμα ανοικτού κώδικα, τρέχει τον ίδιο κώδικα με αυτόν που θα έτρεχε το πραγματικό υλικό ενώ δίνεται μέσω αυτού η δυνατότητα να συνεχίσει κάποιος την δουλειά κάποιου άλλου ερευνητή ή να ξανατρέξει εύκολα πειράματα αλλάζοντας παραμέτρους ή μη. [8]

4.6 Ο Εξομοιωτής Mininet

Το Mininet Hifi (High fidelity) ή απλά Mininet είναι ένας εξομοιωτής δικτύων που επιτρέπει να δημιουργούμε ένα εικονικό δίκτυο με μεταγωγείς, δρομολογητές, υπολογιστές και ελεγκτές SDN με μια μοναδική εντολή σε ένα και μόνο υπολογιστή, ουσιαστικά στο kernel ενός linux. Χρησιμοποιεί εικονικοποίηση η οποία όμως είναι πολύ ελαφριά για το σύστημα που την φιλοξενεί και με την βοήθεια της, εξομοιώνει ένα ολόκληρο δίκτυο στον ίδιο kernel. [8]

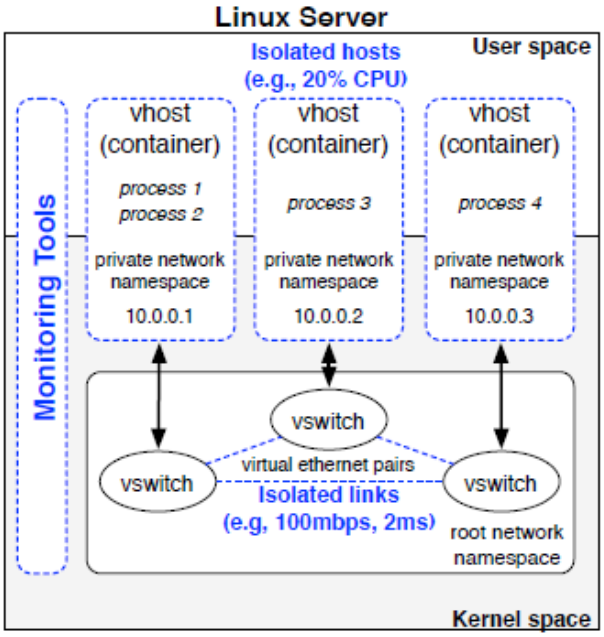
Το Mininet επιτρέπει στον χρήστη του να εξοικειωθεί με νέες έννοιες όπως το SDN και το Openflow καθώς και να δημιουργήσει ή να κάνει δοκιμές σε ελεγκτές SDN ή εφαρμογές τους. Αυτά τα χαρακτηριστικά το καθιστούν ιδανικό για πειραματισμούς και για εξοικείωση με νέες τεχνολογίες. Με την βοήθεια κάποιων συγκεκριμένων εντολών μπορεί κάποιος να επιτύχει επικοινωνία μεταξύ των υπολογιστών του δικτύου που έχει σχεδιάσει, να προσαρμόσει τις ταχύτητες των συνδέσεων στις επιθυμητές για το πείραμα, να διασταυρώσει την ανταλλαγή των πακέτων του Openflow μεταξύ μεταγωγέων και ελεγκτών, να ορίσει τους πίνακες ροής των μεταγωγέων και γενικά να εξετάσει την λειτουργία και την απόδοση του υπό εξέταση δικτύου.

4.6.1 Πλεονεκτήματα Mininet

Το Mininet Hifi (High fidelity) υποστηρίζεται από το mininet.org το οποίο καθοδηγεί τους χρήστες καθ'όλη την διαδικασία που ξεκινά από την εγκατάσταση του εξομοιωτή και φτάνει μέχρι στην δοκιμή ή την δημιουργία διαφορετικών πειραμάτων ώστε ο χρήστης να εξοικειωθεί με το προϊόν και τις τεχνολογίες που επιθυμεί. Ο εξομοιωτής αυτός παρουσιάζει τα παρακάτω πλεονεκτήματα:

- Είναι αρκετά γρήγορο και εύκολο να δημιουργήσεις ένα δίκτυο,
- Η πλατφόρμα υποστηρίζει την δημιουργία προσαρμοσμένων τοπολογιών
- Ουσιαστικά τρέχει σε Linux και άρα όποιο πρόγραμμα τρέχει σε Linux μπορεί να τρέξει και στο Mininet
- Υποστηρίζει τον προγραμματισμό Opeflow μεταγωγέων και ελεγκτών
- Είναι σχετικά εύκολο να το χειριστούν άτομα με εμπειρία στον προγραμματισμό
- Προσφέρει εικονικοποίηση χωρίς να βαραίνει τον υπολογιστή
- Είναι ανοικτού κώδικα [8]
- Δεν κοστίζει
- Μετά το Mininet 2.0 όλες οι επόμενες εκδόσεις του συγκεκριμένου προϊόντος είναι εξομοιωτές υψηλής πιστότητας όσον αφορά τα αποτελέσματα των εξομοιώσεων που προσφέρουν, σύμφωνα με τη διδακτορική διατριβή του Heller.
- Η TCP κίνηση μπορεί να εξομοιωθεί παρέχοντας ίδια αποτελέσματα με τα πραγματικά δίκτυα. [6]
- Προσφέρει απομόνωση των πόρων καθώς και μηχανισμούς επίβλεψης και ελέγχου των χρονικών αποκρίσεων των συσκευών. [4]

Η μαγεία πίσω από το Mininet είναι το ότι έχει ένα σύνολο χαρακτηριστικών που ενσωματώνονται στο Linux και επιτρέπουν σε ένα μόνο σύστημα να χωριστεί σε ένα αριθμό μικρότερων στοιχείων, το καθένα με σταθερό μερίδιο της επεξεργαστικής ισχύς του υπολογιστή, που σε συνδυασμό με τις δημιουργηθείσες εικονικές συνδέσεις επιτρέπει τον καθορισμό με ακρίβεια των καθυστερήσεων που θα εισαχθούν στο δίκτυο αλλά και των ταχυτήτων των συνδέσεων. Το Mininet χρησιμοποιεί εξαιρετικά ελαφρύ virtualization στον πυρήνα του Linux και απομονώνει αποτελεσματικά το εύρος ζώνης της CPU κάνοντας εξαιρετικά εύκολη την διαδικασία ελέγχου και παραμετροποίησης των υπό δημιουργία δικτύων. [7]



Εικόνα 4. 3 Πως λειτουργεί το Mininet χρησιμοποιώντας CPU & Kernel. [7]

4.6.2 Μειονεκτήματα Mininet

Το Mininet έχει βελτιώσει την μελέτη των δικτύων και έχει βοηθήσει πολλές φορές τους ερευνητές στην ανάπτυξη και την δοκιμή νέων εφαρμογών σε ελεγκτές. Έχει ήδη δει ευρεία χρήση, καθώς προσφέρει καλό ρεαλισμό και απρόσκοπτη μετάβαση από την ανάπτυξη έως την εγκατάσταση και την φυσική λειτουργία του δικτύου, αλλά δεν αποτελεί ιδανική λύση για δίκτυα μεγάλης κλίμακας. [5], [12]

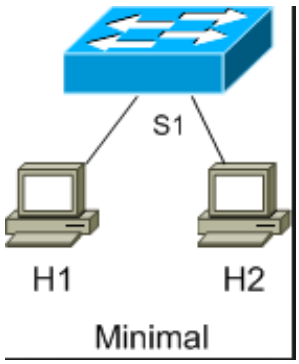
Κεφάλαιο 5
Πειράματα με το Mininet

Στο κεφάλαιο αυτό θα εξετάσουμε ένα σημαντικό αριθμό πειραμάτων με τα οποία θα μπορέσουμε να μελετήσουμε διαφορετικές μορφές δικτύων SDN και θα εκτελέσουμε βασικές εντολές που εμπεριέχει το Mininet. Συνεπώς, σε αυτό το κεφαλαίο θα μάθουμε να χρησιμοποιούμε το Mininet ως εργαλείο για τη δημιουργία εικονικών δικτύων και θα μπορέσουμε να εξάγουμε συμπεράσματα για τη συμπεριφορά, των υπό εξομοίωση δικτύων και τη λειτουργία τους. Παράλληλα, θα συλλέξουμε και θα παραθέσουμε τα γραφήματα του Wireshark που λήφθησαν κατά την διάρκεια αυτών των πειραμάτων ώστε να μελετήσουμε στη συνέχεια τα SDN δίκτυα και τη συμπεριφορά τους.

5.1 Δημιουργία Ελάχιστης Τοπολογίας

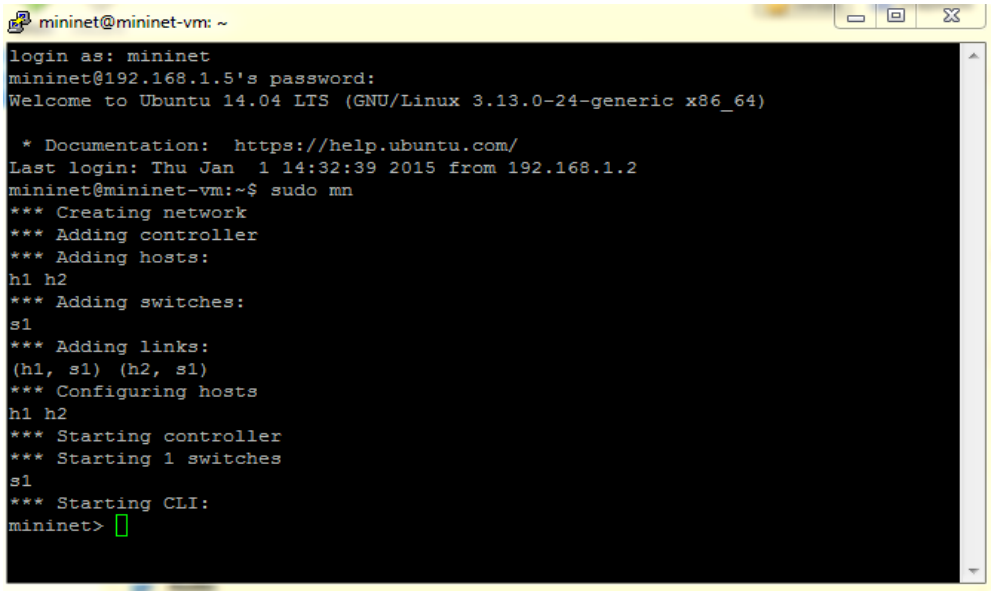
Το Mininet όπως είδαμε και στη προηγούμενη ενότητα δημιουργεί ένα ρεαλιστικό εικονικό δίκτυο, με πραγματικό κώδικα σε ένα μοναδικό μηχάνημα (VM) μέσα σε δευτερόλεπτα, με μία μόνο εντολή. Στο παρακάτω πείραμα θα δημιουργήσουμε την πιο βασική τοπολογία στο Mininet που είναι δυο κόμβοι συνδεδεμένοι με έναν μεταγωγέα για να εξοικειωθούμε με το πρόγραμμα εξομοίωσης και να διαπιστώσουμε τον τρόπο λειτουργίας του. Επίσης θα συλλέξουμε τα πακέτα που ανταλλάσουν τα μέλη του δικτύου με την βοήθεια του προγράμματος wireshark ώστε να κατασκευάσουμε κατόπιν τα γραφήματα που φανερώνουν ποιοτικά χαρακτηριστικά του δικτύου.

Για την δημιουργία ενός δικτύου στο Mininet αρκεί μια εντολή. Για παράδειγμα η εντολή **$ sudo mn** δημιουργεί την ελάχιστη minimal τοπολογία η οποία είναι ένας OpenFlow Kernel μεταγωγέας που συνδέετε με δυο υπολογιστές καθώς και ένας Openflow ελεγκτής [27]. Συγκεκριμένα το δίκτυο συνδέεται με τον προκαθορισμένο τον OpenFlow reference controller.



Εικόνα 5. 1 Ελάχιστη τοπολογία δικτύου [38]

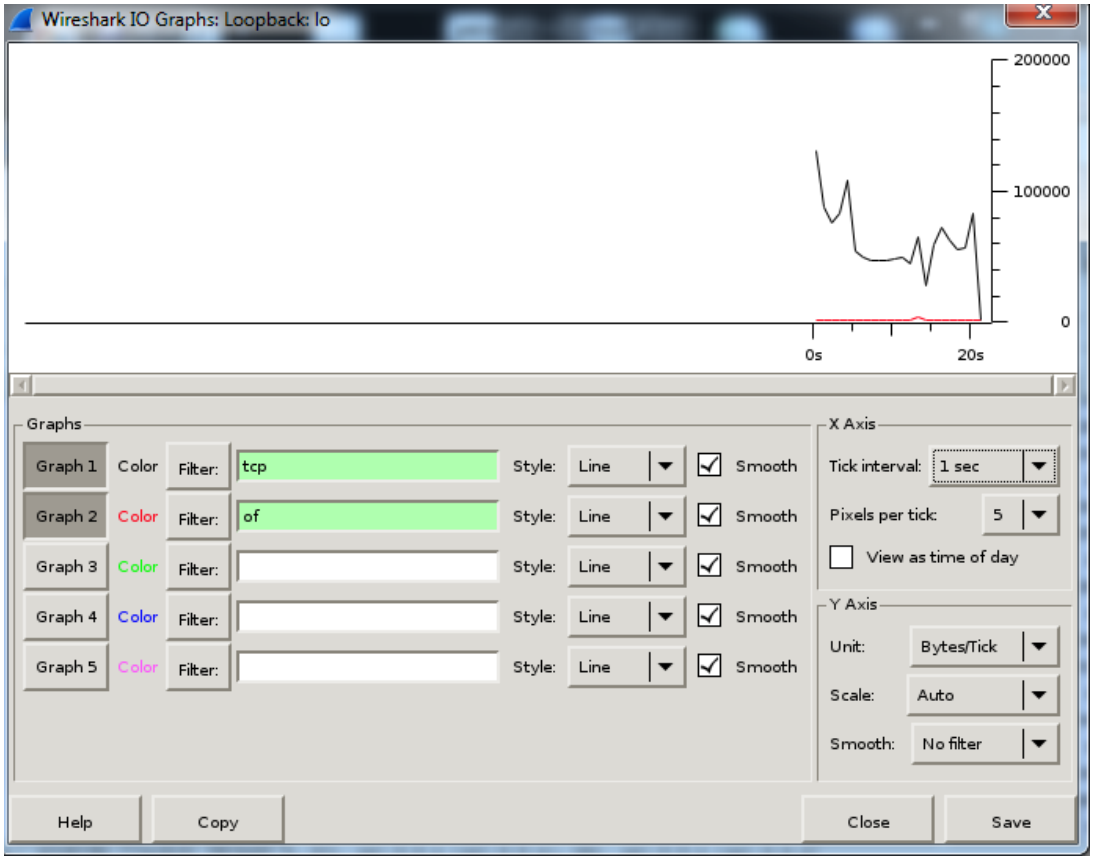
Στο παρακάτω σχήμα φαίνεται το CLI του Mininet. Παρατηρούμε ότι με την απλή εντολή **$ sudo mn** ο εξομοιωτής, σε κλάσματα δευτερολέπτου δημιούργησε το προαναφερθέν δίκτυο του σχήματος, πρόσθεσε τους υπολογιστές, τον μεταγωγέα, τις συνδέσεις μεταξύ των και ξεκίνησε την λειτουργία του ελεγκτή και του μεταγωγέα.



Εικόνα 5. 2 Δημιουργία Δικτύου στο Mininet όταν ορίσουμε την ελάχιστη τοπολογία με την εντολή $ sudo mn .

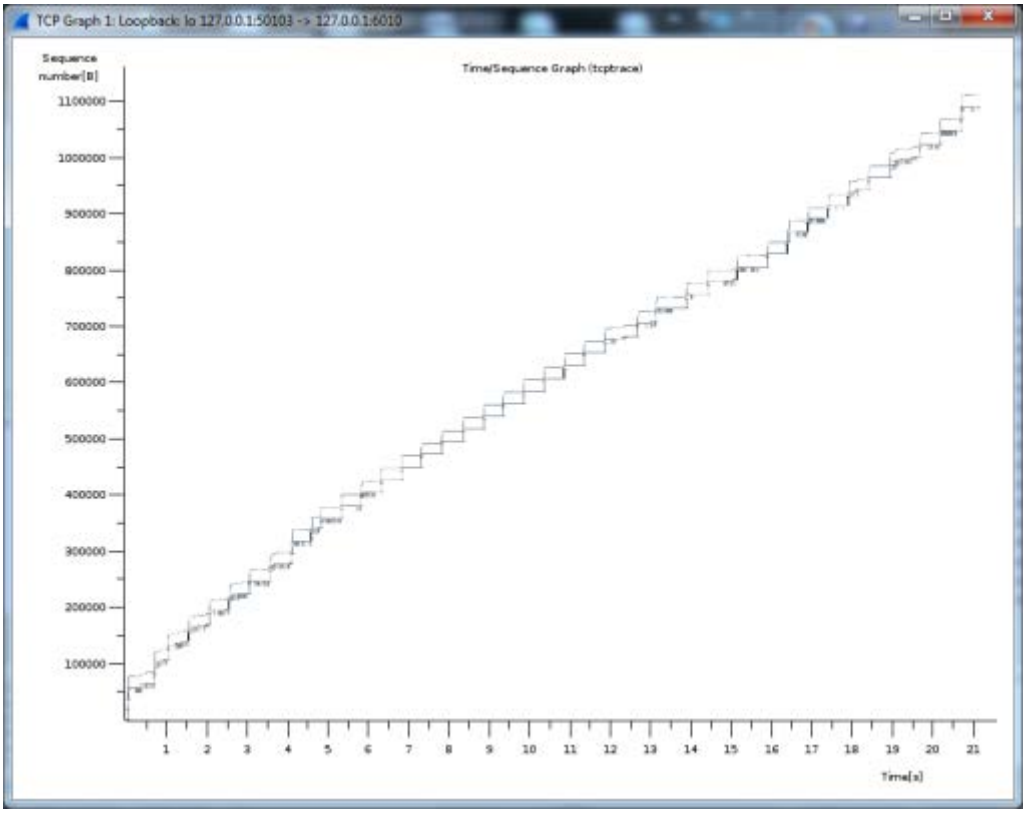
Για να ανοίξουμε το πρόγραμμα Wireshark και να δούμε τα πακέτα που ανταλλάσσονται μεταξύ μεταγωγέα και ελεγκτή στο Mininet πρέπει να

εκτελέσουμε την εντολή: $ **sudo wireshark &** πριν τη δημιουργία του υπό μελέτη δικτύου [21].



Γράφημα 5. 1 TCP ροή και OF μηνύματα κατά την δημιουργία της βασικής τοπολογίας στο Mininet.

Στα γραφήματα που ακολουθούν και αφορούν το πρώτο πείραμα φαίνονται τρείς διαφορετικές κατηγορίες γραφημάτων. Αρχικά το TCP trace γράφημα, φανερώνει στο κάθετο άξονα το Sequence Number των πακέτων και στο οριζόντιο άξονα το χρόνο [31]. Τα γραφήματα που αφορούν το Throughput έχουν στο κάθετο άξονα τα byte/sec ενώ στο οριζόντιο άξονα βρίσκονται τα δευτερόλεπτα, ουσιαστικά δείχνουν το εύρος ζώνης του δικτύου. Τα γραφήματα που παρουσιάζουν το Round Trip Time έχουν στο κάθετο άξονα το χρόνο δηλαδή τα δευτερόλεπτα και στον οριζόντιο άξονα το Sequence Number. Φανερώνουν δηλαδή τον χρόνο Ping του δικτύου. Όλα τα γραφήματα καταγράφηκαν κατά την διάρκεια της δημιουργίας των δικτύων και μετά την εκτέλεση των εντολών που σχετίζονται με το εκάστοτε πείραμα.



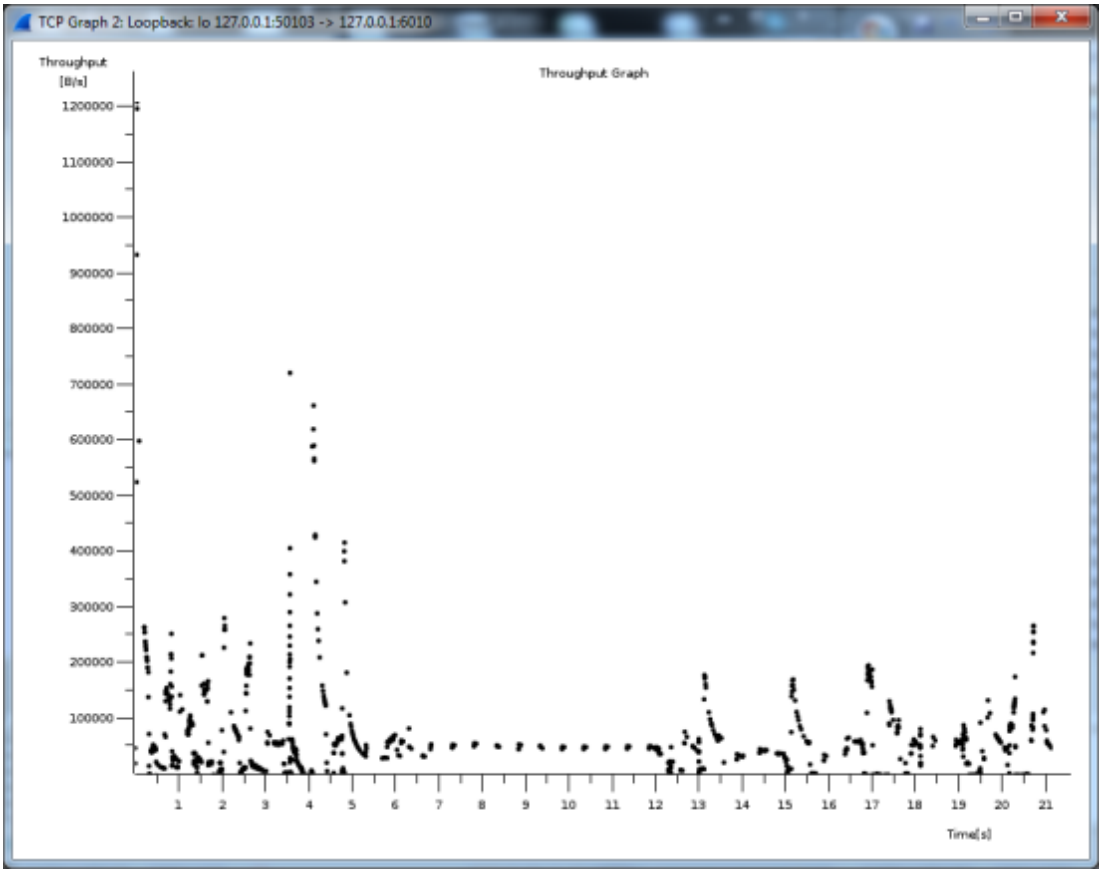
Γράφημα 5. 2 Μέτρηση TCP trace σε δίκτυο με ελάχιστη τοπολογία.

Στο Γράφημα Tcptrace φαίνεται μια χρονική ακολουθία που απεικονίζει την "δραστηριότητα" της σύνδεσης. Περισσότερες λεπτομέρειες είναι διαθέσιμες, όταν κάνουμε "zoom in" στον x άξονα [31]. Ο άξονας Υ όπως αναφέραμε δείχνει το Sequence Number. Όπως γνωρίζουμε από την μελέτη του TCP ο κάθε κόμβος διατηρεί ένα 32-bit αριθμό ακολουθίας που χρησιμοποιεί για να παρακολουθεί πόσα δεδομένα έχει στείλει και στις δύο πλευρές μιας συνόδου TCP. Αυτή η αλληλουχία αριθμών περιλαμβάνεται σε κάθε πακέτο που μεταφέρεται, και αναγνωρίζεται από τον ξενιστή αποστέλλοντας παράλληλα ένα αριθμό επιβεβαίωσης για να ενημερώσει τον κόμβο αποστολέα για την σωστή υποδοχή του πακέτου ή των δεδομένων που έλαβε με επιτυχία. Από την άλλη ο άξονας Χ δείχνει την ώρα που έκαναν τα δεδομένα να αποσταλούν.

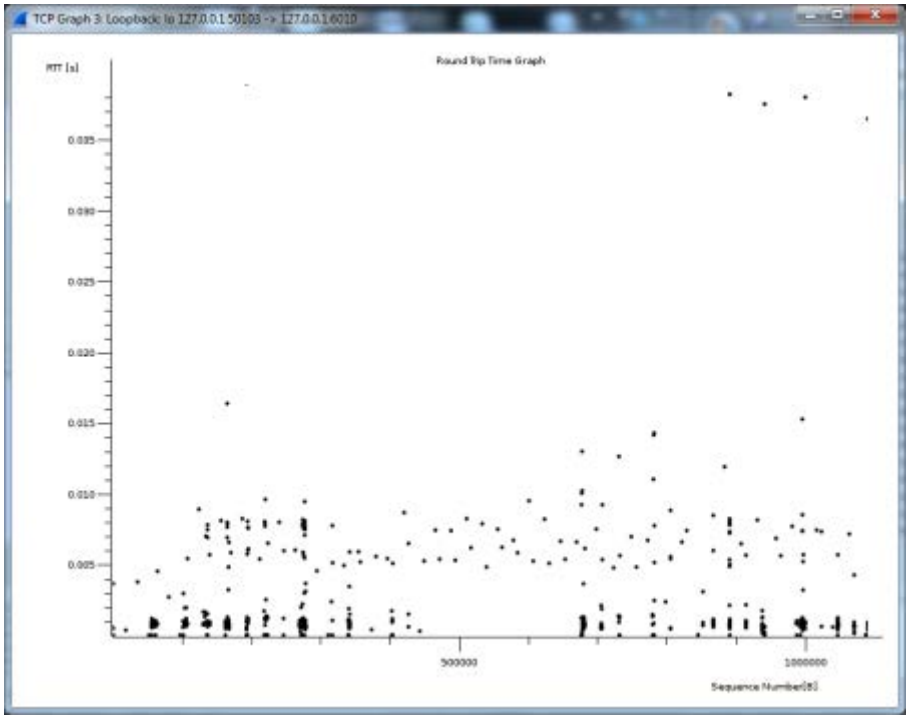
Στο Wireshark το Sequence Number είναι ένας σχετικός αριθμός που αποδίδει το πρόγραμμα ώστε να είναι πιο εύκολο για τον χρήστη να αναγνωρίσει τα πακέτα. Ο αριθμός αυτός στην πραγματικότητα είναι εύρους 32-bit και μπορεί

να είναι οποιοσδήποτε αριθμός μεταξύ του 0 και του 4294967295. Τα γραφήματα δεν μεγεθυνθήκαν περαιτέρω για λόγους οικονομίας χώρου, παρόλο που θα ήταν πιο εύκολο να παρατηρηθούν περισσότερα χαρακτηριστικά της κίνησης στα δίκτυα των πειραμάτων. Από την εικόνα που έχουμε παρατηρούμε όμως, ότι είναι εύκολα αντιληπτό ότι η κλίση στο συγκεκριμένο γράφημα δείχνει την απόδοση κατά την πάροδο του χρόνου.

Με το παρακάτω γράφημα Throughput μπορούμε να δούμε τη ποσότητα των δεδομένων που μεταδίδεται ανά χρονική στιγμή στο δίκτυο μας. Παρατηρούμε την ανταλλαγή των πακέτων για την εγκαθίδρυση της σύνδεσης στα πρώτα 5-6 δευτερόλεπτα αλλά και την κίνηση που προκαλούν οι εντολές nodes, dump, pingall κατά τα χρονικά διαστήματα 13-21 δευτερόλεπτα του άξονα χρόνου του γραφήματος (περισσότερες λεπτομέρειες για τις εντολές αυτές δίνονται στο παράρτημα).



Γράφημα 5. 3 Μέτρηση Throughput σε δίκτυο με ελάχιστη τοπολογία



Γράφημα 5. 4 Μέτρηση RTT χρόνου σε δίκτυο με ελάχιστη τοπολογία

Στο γράφημα που δείχνει τον χρόνο RTT στο δίκτυο της ελάχιστης τοπολογίας παρατηρούμε τον χρόνο μετ 'επιστροφής (RTT), ή αλλιώς τη λεγόμενη καθυστέρηση μετ' επιστροφής. Ο χρόνος αυτός δείχνει το διάστημα που απαιτείται από έναν παλμό σήματος ή πακέτο ώστε να ταξιδέψει από μια συγκεκριμένη πηγή προς ένα συγκεκριμένο προορισμό και πάλι πίσω. Στο πλαίσιο αυτό, η πηγή είναι ο υπολογιστής που πραγματοποιεί την έναρξη του σήματος και ο προορισμός είναι ένας απομακρυσμένος υπολογιστής. Παρατηρούμε ότι ο χρόνος καθυστέρησης είναι σε φυσιολογικές τιμές και αυτό είναι κάτι αναμενόμενο, δεδομένου ότι δεν υπάρχει ιδιαίτερη κίνηση στο δίκτυο, δεν έχει καθοριστεί να επικοινωνούν οι κόμβοι με καθυστέρηση κτλ.

5.1.4 Δημιουργία Εξυπηρετητή Δικτύου

Στο παρακάτω πείραμα θα εξετάσουμε την περίπτωση της προσομοίωσης στο Mininet ενός απλού εξυπηρετητή δικτύου (HTTP server) και ενός πελάτη - client. Θα πρέπει

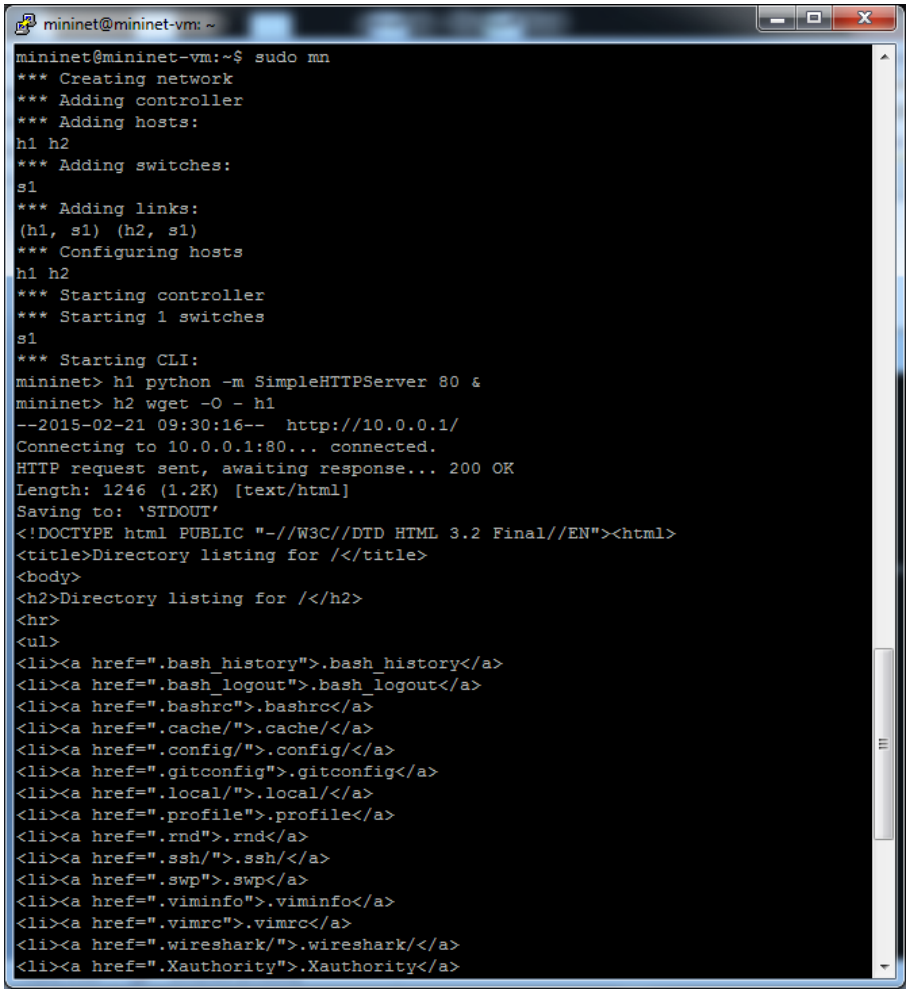
αφού λοιπόν έχουμε εγκαταστήσει για παράδειγμα την ελάχιστη τοπολογία -minimal να δώσουμε τις παρακάτω εντολές στο Mininet [47]:

a. mininet> h1 python -m SimpleHTTPServer 80 &

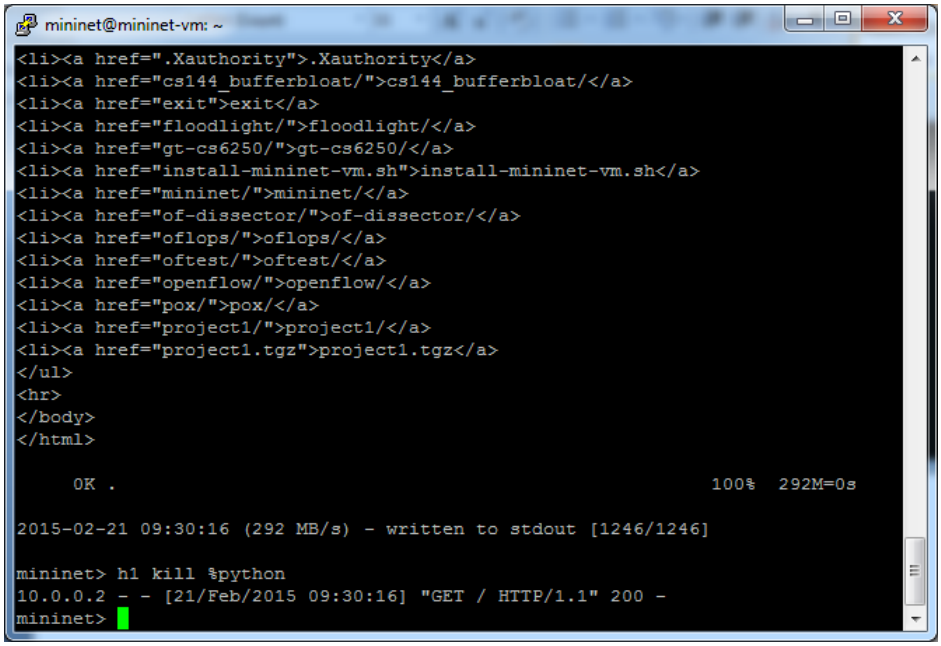
b. mininet> h2 wget -O - h1

....

c. mininet> h1 kill %python



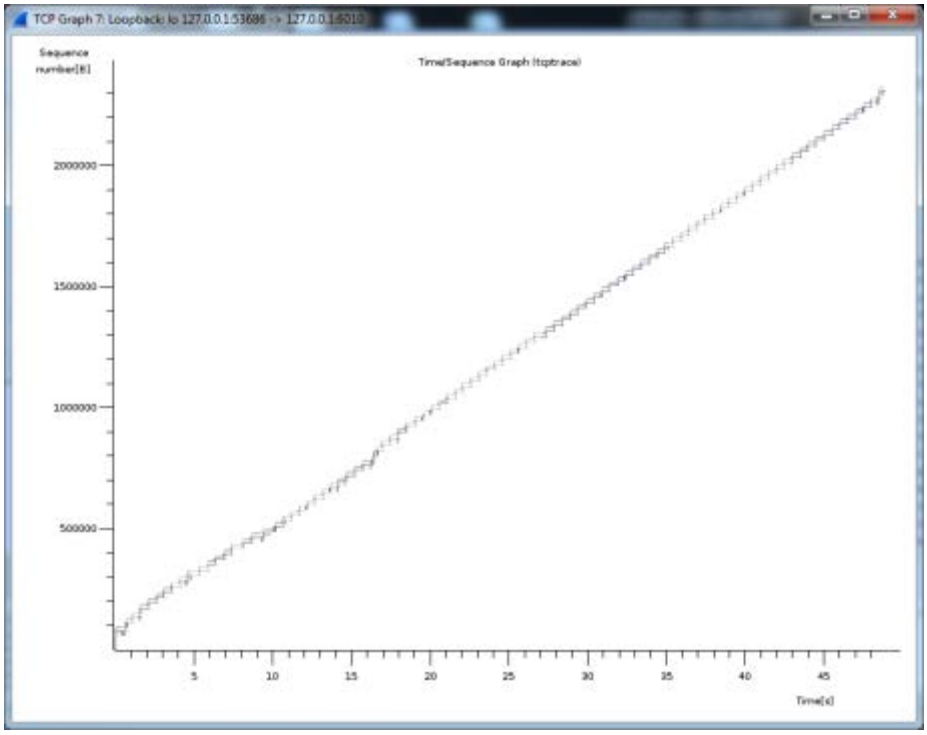
Εικόνα 5. 3 Δημιουργία ελάχιστης τοπολογίας και ενός απλού εξυπηρετητή δικτύου στο Mininet I



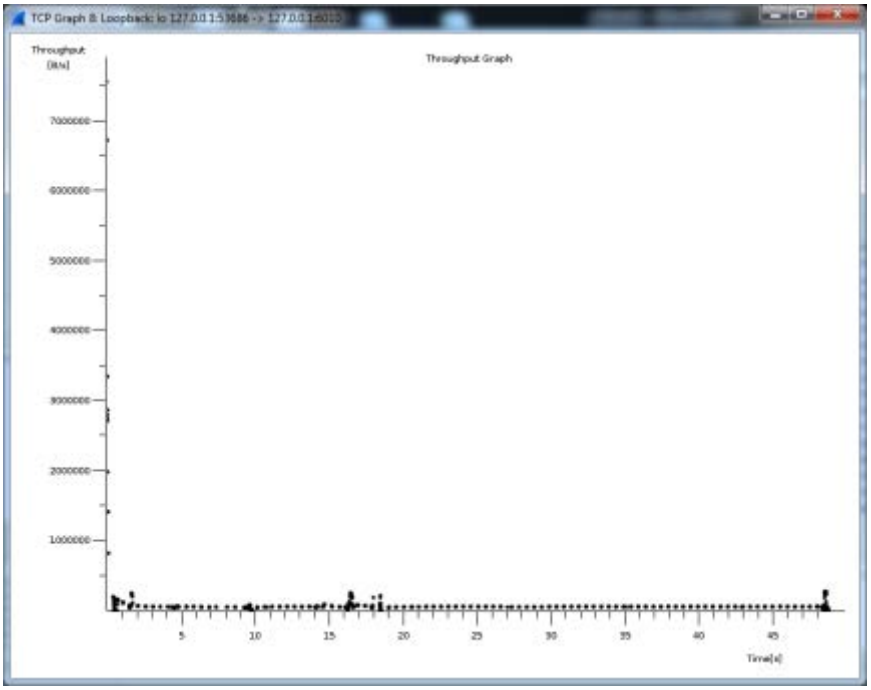
Εικόνα 5. 4 Δημιουργία ελάχιστης τοπολογίας και ενός απλού εξυπηρετητή δικτύου στο Mininet II

Στο παραπάνω πείραμα δημιουργήσαμε λοιπόν πάλι την ελάχιστη τοπολογία καθώς και ένα εξυπηρετητή στο δίκτυο και ζητήσαμε από τον άλλο κόμβο να τραβήξει την πληροφορία από αυτόν με την εντολή **mininet> h2 wget -O - h1**. Παρατηρούμε ότι το πείραμα εξελίχθηκε ομαλά. Είναι σημαντικό να μπορούμε να δημιουργούμε εξυπηρετητές στο Mininet μιας και είναι κομμάτι πολλών πειραμάτων που βρίσκονται στο διαδίκτυο για την εξοικείωση με το πρόγραμμα. Με αυτό το πείραμα συνεπώς αξιοποιούμε άλλη μια δυνατότητα του Mininet. Στο παρακάτω γράφημα παρατηρούμε ότι η απόδοση κατά την διάρκεια του χρόνου είναι ικανοποιητική.

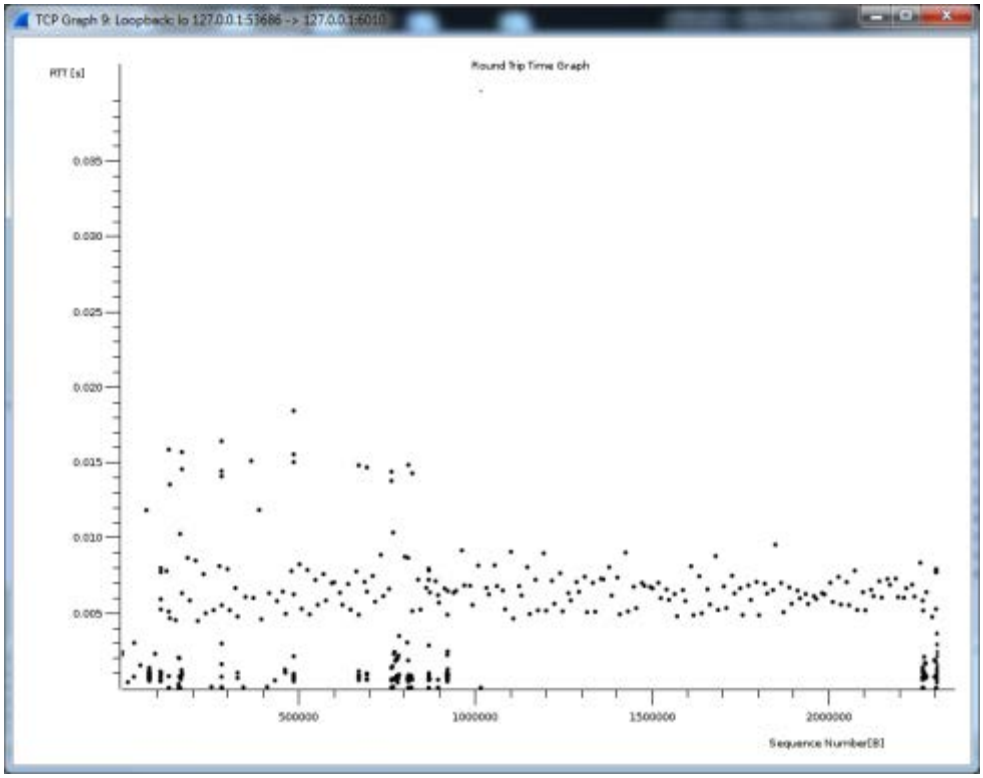
Στο γράφημα Throughput που ακολουθεί παρατηρούμε ότι η κίνηση που δημιουργείται στο δίκτυο είναι μηδαμινή όπως αναμενόταν για το συγκεκριμένο πείραμα. Η μόνη στιγμή που φανερώνεται στα γραφήματα κάποια ιδιαίτερη κίνηση, πέρα από την διακίνηση των πακέτων για την διατήρηση της σύνδεσης, είναι τα 16 δευτερόλεπτα που ήταν ο χρόνος κατά τον οποίο ζητήθηκε η σελίδα από τον εξυπηρετητή με την εντολή **mininet> h2 wget -O - h1**.



Γράφημα 5. 5 Μέτρηση TCP trace σε δίκτυο με ελάχιστη τοπολογία και κατά την δημιουργία HTTP Server.



Γράφημα 5. 6 Μέτρηση Throughput σε δίκτυο με ελάχιστη τοπολογία και κατά την δημιουργία ενός HTTP Server.



Γράφημα 5. 7 Μέτρηση RTT χρόνου σε δίκτυο με ελάχιστη τοπολογία και κατά την δημιουργία HTTP Server.

Οι χρόνοι καθυστέρησης στο παραπάνω γράφημα δεν εμφανίζουν διαφοροποιήσεις με το RTT γράφημα της αρχικής τοπολογίας παρά μόνο κατά την στιγμή που δημιουργείται το δίκτυο αλλά και κατά το χρονικό παράθυρο κατά το οποίο ζητάμε την σελίδα με την εντολή **mininet> h2 wget -O - h1**. Ο χρόνος RTT κατά αυτές τις χρονικές στιγμές φαίνεται γύρω στα 0,015 δευτερόλεπτα.

5.1.5 Παραμετροποίηση Ελεγκτή στο Mininet

Στην ενότητα αυτή θα επικεντρωθούμε στα διαφορετικά είδη ελεγκτών που υποστηρίζει το Mininet. Όπως γνωρίζουμε διαθέτει ενσωματωμένους ελεγκτές για τη δημιουργία παραμετροποιημένων δικτύων με διαφορετικού τύπου ελεγκτές και συγκεκριμένα διαθέτει τον reference controller (controller), τον ovs-controller και τον λιγότερο χρησιμοποιούμενο NOX Classic. Επίσης υπάρχει διαθέσιμος και ο Floodlight ο οποίος είναι ένας Java - based OpenFlow controller και είναι εύκολο να προστεθεί ως μέρος ενός δικτύου στην θέση του

προκαθορισμένου reference controller. Τέλος ο OpendayLight Controller αποτελεί ένα open source project και περιλαμβάνει ένα ελεγκτή μαζί με διάφορα plug in και applications.

Η παραμετροποίηση του ελεγκτή στο Mininet γίνεται με τις παρακάτω εντολές αντίστοιχα (δεν θα αναφερθούμε στους δυο τελευταίους ελεγκτές σε αυτή την μεταπτυχιακή διατριβή):

- # mn --controller ref
- # mn --controller ovsc
- # mn --controller nox [10]

TABLE 1 DIFFERENT SDN CONTROLLERS

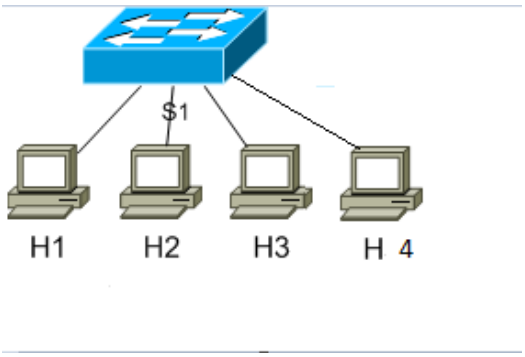
	POX	Ryu	Trema	Floodlight	Open Day Light
Language Support	Python	Python	C Ruby	Java	Java
OpenFlow Support	v1.0	v1.0 v1.2 v1.3	v1.0	v1.0	v1.0
OpenSource	Yes	Yes	Yes	Yes	Yes
GUI	Yes	Yes	No	Web GUI	Yes
REST API	No	Yes	No	Yes	Yes
Platform Support	Linux Mac Windows	Linux	Linux	Linux	Linux Mac Windows

Εικόνα 5. 5 SDN ελεγκτές και χαρακτηριστικά τους. [10]

5.2 Δημιουργία Δικτύου 4 κόμβων

Σαν δεύτερο παράδειγμα για δημιουργία δικτύου στο Mininet μπορούμε να δώσουμε την εντολή

$ sudo mn --topo single,4

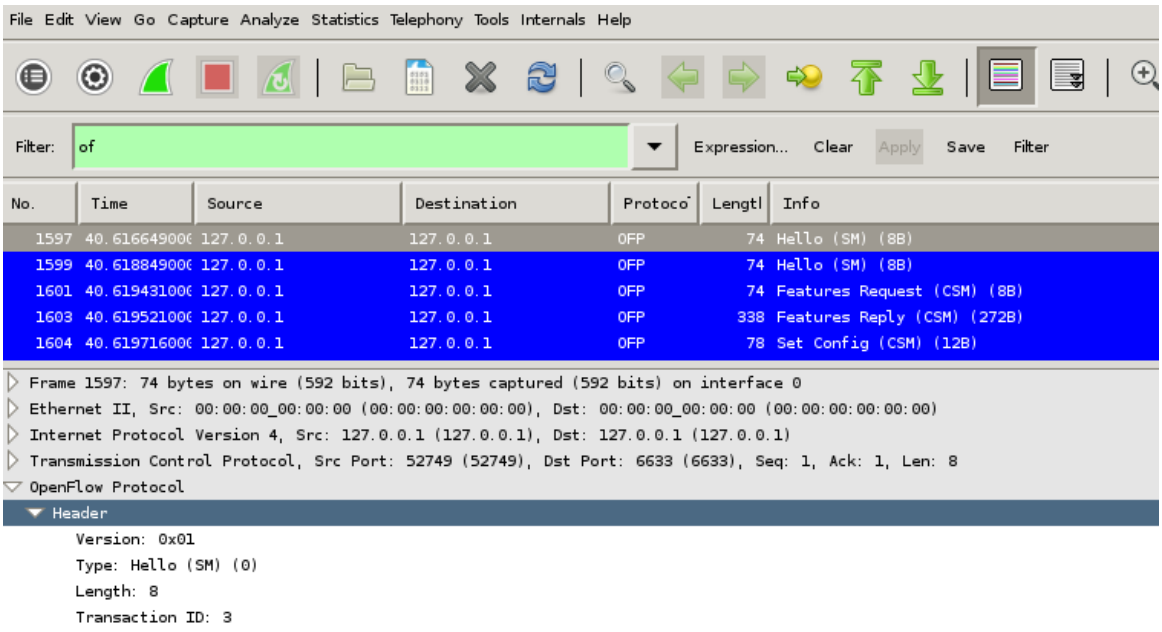


Εικόνα 5. 6 Δίκτυο με ένα μεταγωγέα και 4 κόμβους [38]

Η εντολή αυτή δημιουργεί ένα δίκτυο με ένα μεταγωγέα και τέσσερις κόμβους. Δηλαδή το Mininet με αυτή την εντολή δημιουργεί 4 εικονικούς κόμβους κάθε ένα με μια ξεχωριστή διεύθυνση IP, ένα μεταγωγέα Openflow με 4 θύρες ενώ συνδέει το κάθε κόμβο με τον μεταγωγέα με ένα εικονικό ethernet. Ορίζει τέλος την σύνδεση του μεταγωγέα με ένα ελεγκτή. Στο σενάριο δεν συγκεκριμενοποιήσαμε τον τύπο του ελεγκτή που επιθυμούμε να έχει το δίκτυο μας, συνεπώς θα χρησιμοποιήσουμε τον προκαθορισμένο ελεγκτή του Mininet που είναι ο OVS Controller. Ο ελεγκτής αυτός υλοποιεί την λειτουργικότητα ενός layer 2 mac learning switch. Δηλαδή ο ελεγκτής μαθαίνει τις MAC διευθύνσεις των κόμβων που είναι συνδεδεμένοι στο δίκτυο και προγραμματίζει έτσι τους πίνακες ροής του μεταγωγέα ώστε να εγκαθιδρύσει συνδέσεις μεταξύ εκείνων των κόμβων που επικοινωνούν. Στο σχήμα που ακολουθεί φαίνεται η εικόνα του Mininet κατά την διάρκεια δημιουργίας του δικτύου. Παρατηρούμε την προσθήκη συνδέσεων, του ελεγκτή καθώς και των κόμβων.

Εφόσον είχαμε ανοικτό, κατά την διάρκεια της δημιουργίας του δικτύου στο Mininet, το Wireshark, όπως είχαμε πει σε προηγούμενα πειράματα, ήδη θα έχουμε συλλάβει τα πακέτα που αντάλλαξαν οι συσκευές του εικονικού μας δικτύου. Το πείραμα αυτό έχει σαν στόχο να μας βοηθήσει μαζί με την θεωρία του Openflow να κατανοήσουμε το πρωτόκολλο επικοινωνίας πάνω στο οποίο βασίζεται το SDN.

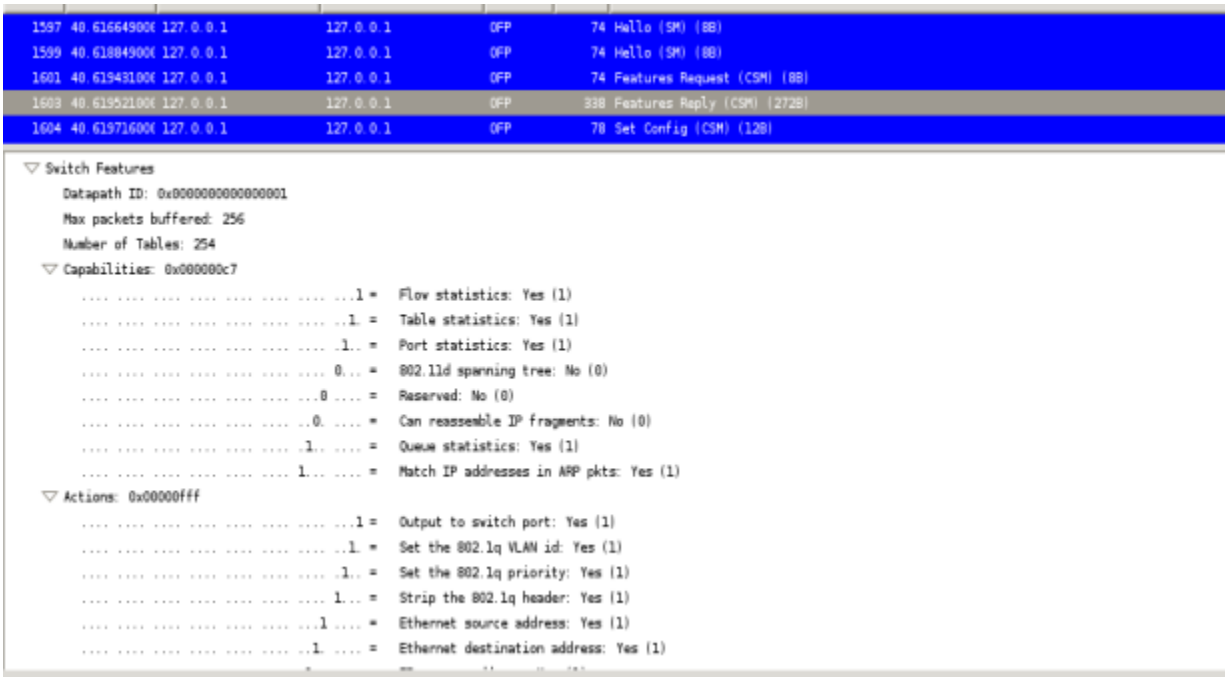
Συγκεκριμένα παρατηρούμε από την παρακάτω εικόνα του Wireshark τα μηνύματα Hello μεταξύ ελεγκτή και μεταγωγέα που ακολουθούν το TCP Handshake κατά τα οποία ο ελεγκτής στέλνει στον μεταγωγέα το νούμερο της έκδοσης του πρωτοκόλλου με το οποίο θα επικοινωνήσουν στην συνέχεια. Ο μεταγωγέας απαντά ότι υποστηρίζει τη συγκεκριμένη έκδοση.



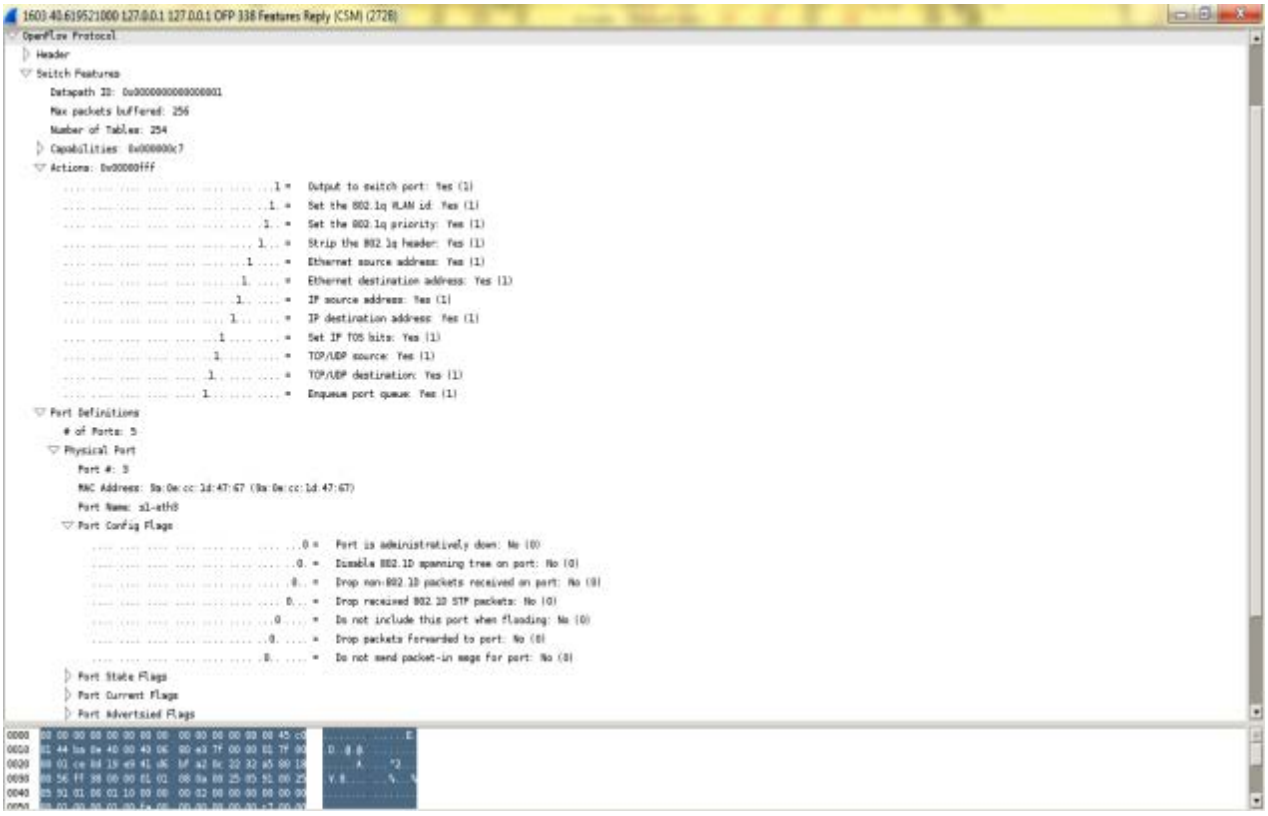
Εικόνα 5. 7 Εικόνα Wireshark κατά την διάρκεια δημιουργίας του δικτύου 4 κόμβων.

Μετά παρατηρούμε ένα Features Request μήνυμα με το οποίο ο ελεγκτής ρωτά να δει ποιες θύρες είναι διαθέσιμες. Το μήνυμα αποτελείται από μια Openflow επικεφαλίδα και δεν έχει σώμα. [41]

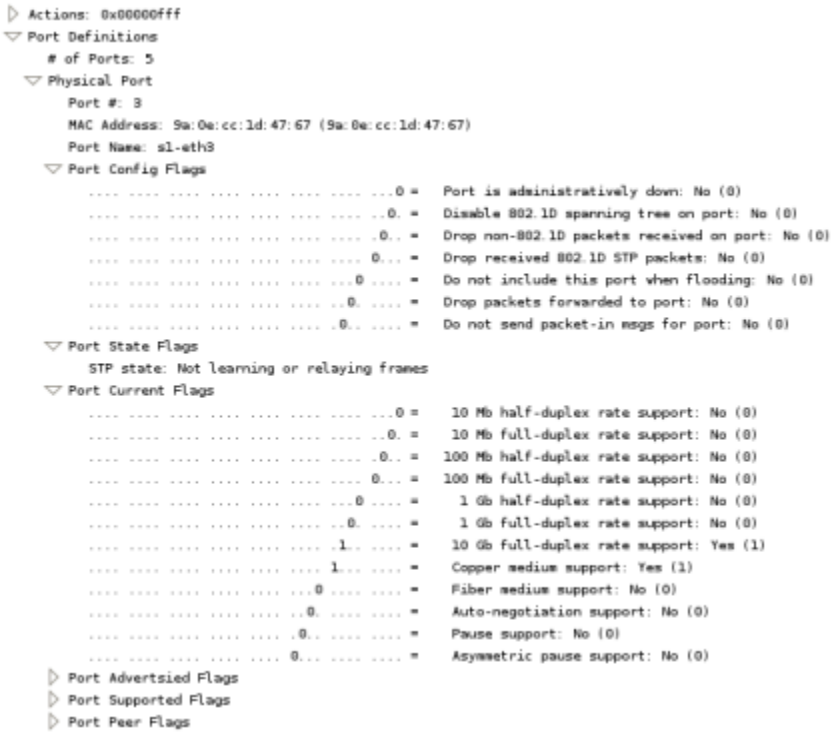
Στη συνέχεια παρατηρούμε ένα Features Reply μήνυμα στο οποίο ο μεταγωγέας απαντά με την λίστα των θυρών του, τις ταχύτητες τους και τους πίνακες μαζί με τις ενέργειες που περιέχουν.



Εικόνα 5. 8 Features Reply μήνυμα

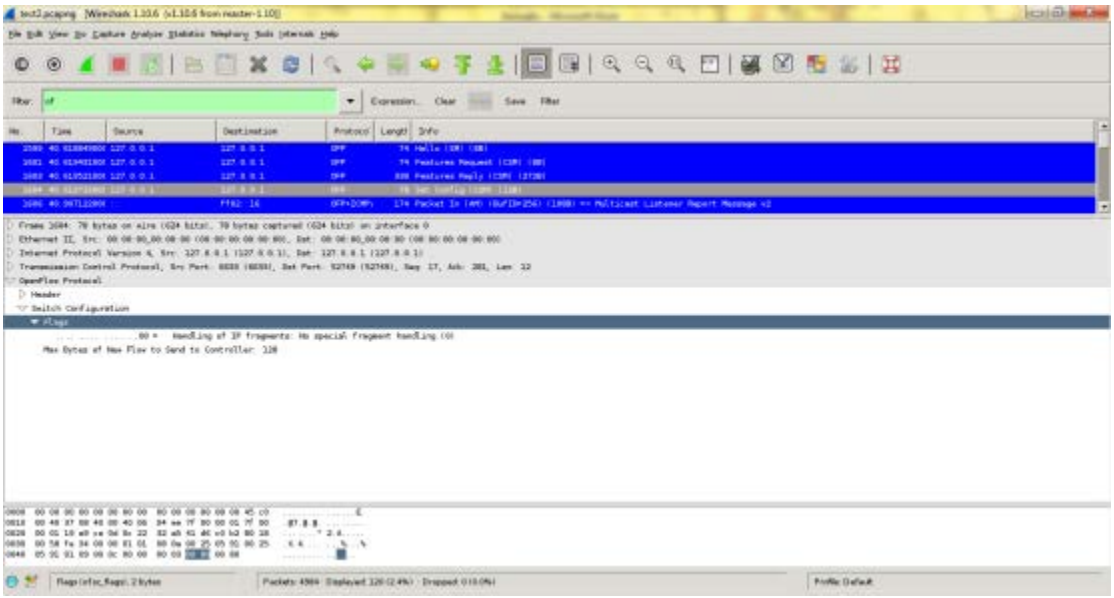


Εικόνα 5. 9 Features Reply μήνυμα



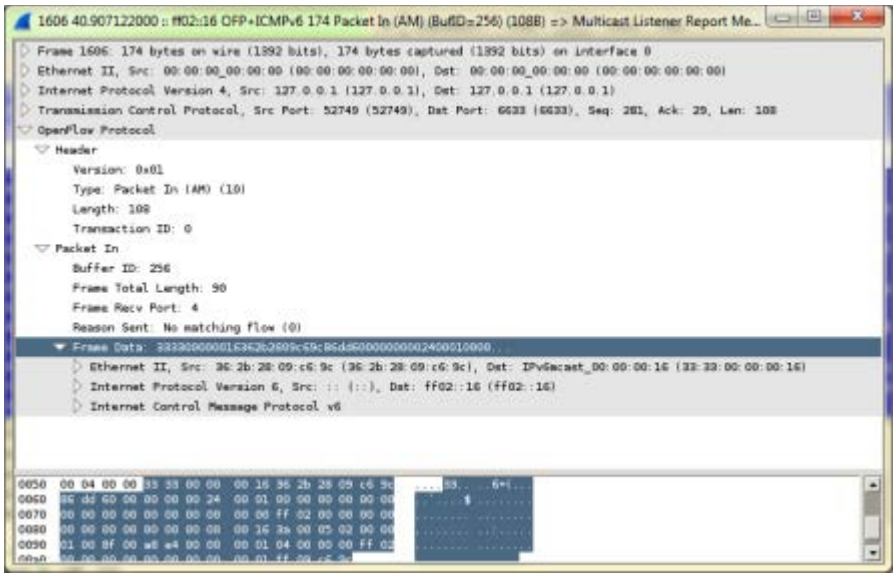
Εικόνα 5. 10 Features Reply μήνυμα - φαίνονται οι λεπτομέρειες της κάθε πόρτας του μεταγωγέα.

Το Set Config μήνυμα που ακολουθεί ξεκινά από τον ελεγκτή και ζητά από τον μεταγωγέα να στείλει τις λήξεις των ροών - flow expirations. Επίσης περιέχει το μέγιστο αριθμό bytes του πακέτου που μπορεί να αποσταλεί στον ελεγκτή.



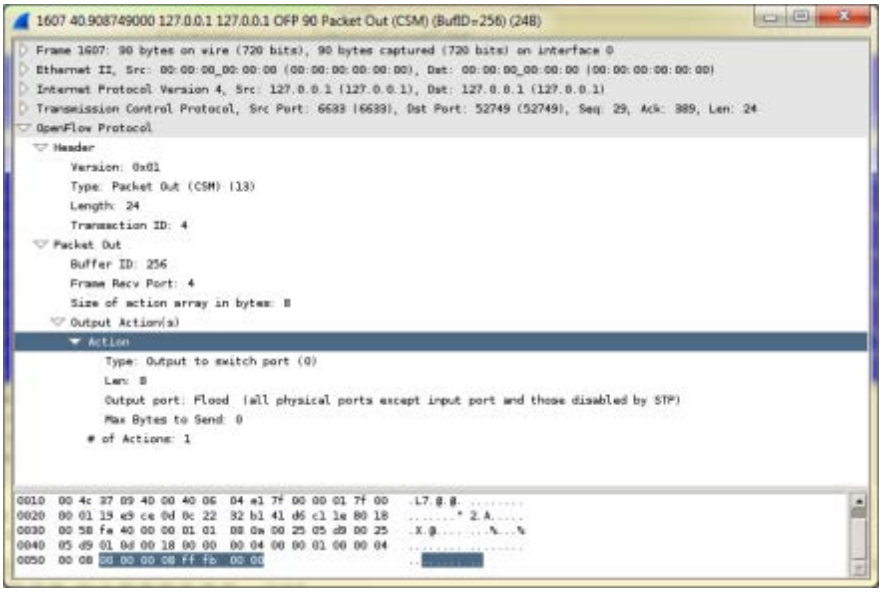
Εικόνα 5. 11 Set Config μήνυμα

Ακολουθεί το Packet in μήνυμα το οποίο είναι ένα πακέτο που παρελήφθη από τον μεταγωγέα και δεν ταιριάζει με καμία καταχώρηση στους πίνακες ροής οπότε αυτό προκαλεί την αποστολή του πακέτου στον ελεγκτή. [41]



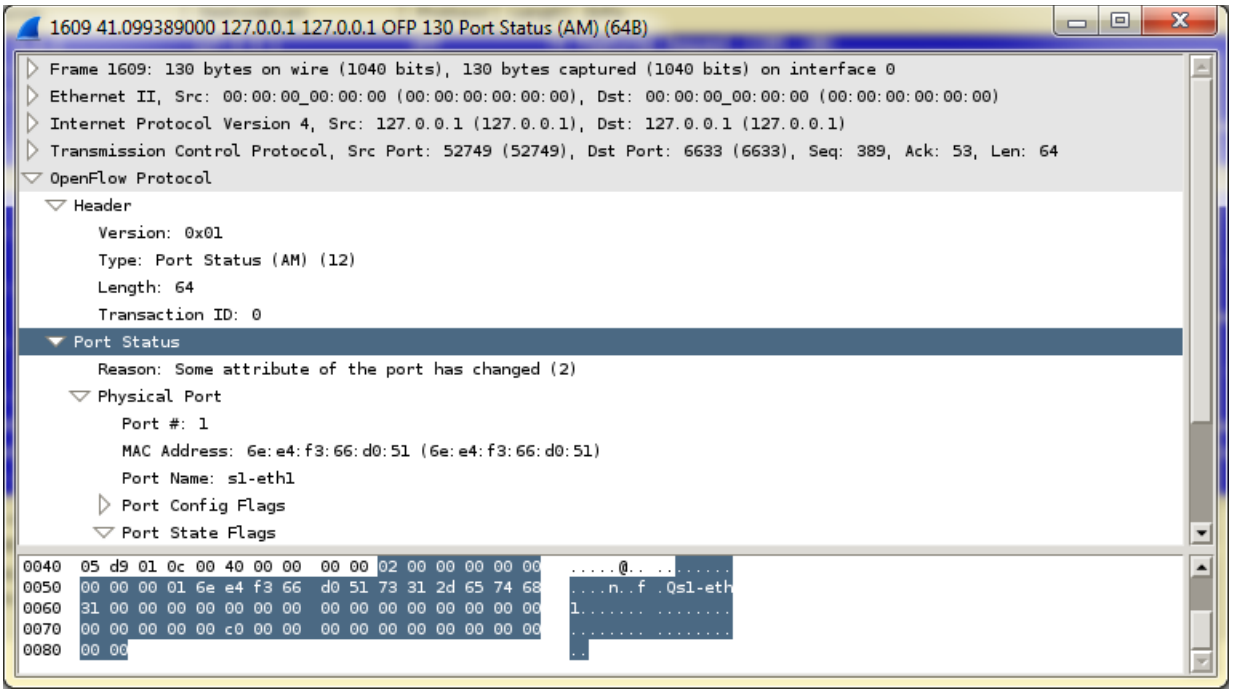
Εικόνα 5. 12 Λεπτομέρειες Packet in μηνύματος

Στην συνέχεια παρατηρούμε το μήνυμα Packet out με το οποίο ο ελεγκτής στέλνει ένα πακέτο σε μια ή περισσότερες θύρες του μεταγωγέα.



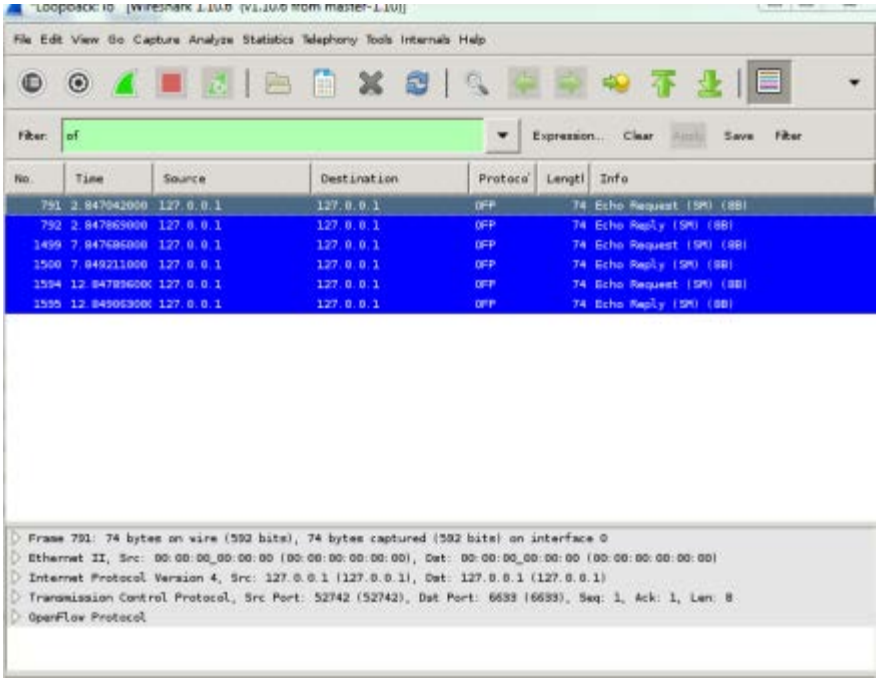
Εικόνα 5. 13 Packet Out μήνυμα

Το Port status μήνυμα δίνει τη δυνατότητα στον μεταγωγέα να πληροφορήσει τον ελεγκτή για τις αλλαγές στις ταχύτητες των θυρών ή στην συνδεσιμότητα τους. [41]



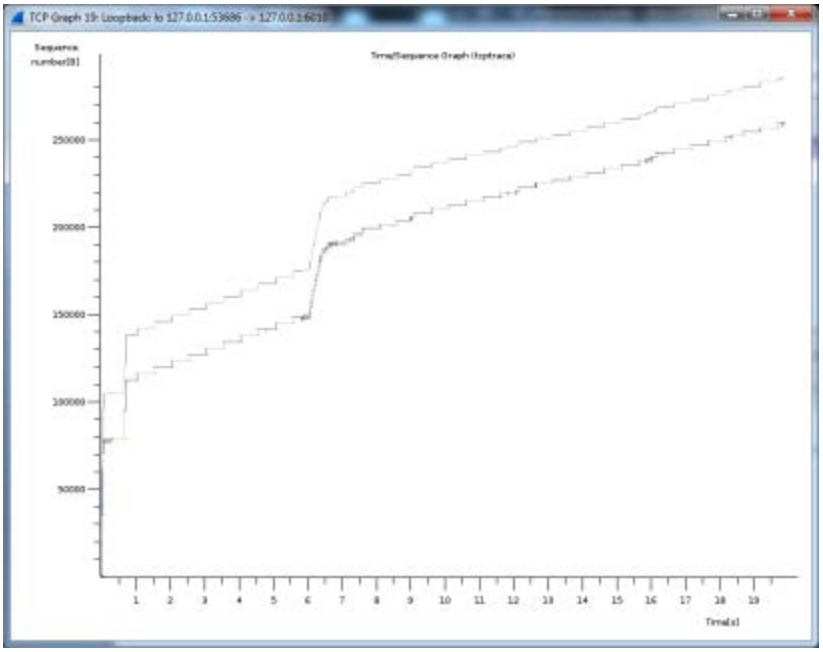
Εικόνα 5. 14 Port Status μήνυμα

Μετά την ίδρυση ενός δικτύου και την παρέλευση μικρού χρονικού διαστήματος παρατηρούμε τα μηνύματα echo μεταξύ μεταγωγέα και ελεγκτή. Τα μηνύματα αυτά αποστέλλονται μεταξύ των δυο συσκευών για να κρατήσουν την σύνδεση "ζωντανή". Από την εικόνα παρακάτω παρατηρούμε ότι τα μηνύματα echo & request είναι κομμάτι του Openflow πρωτοκόλλου όπως γνωρίζουμε.



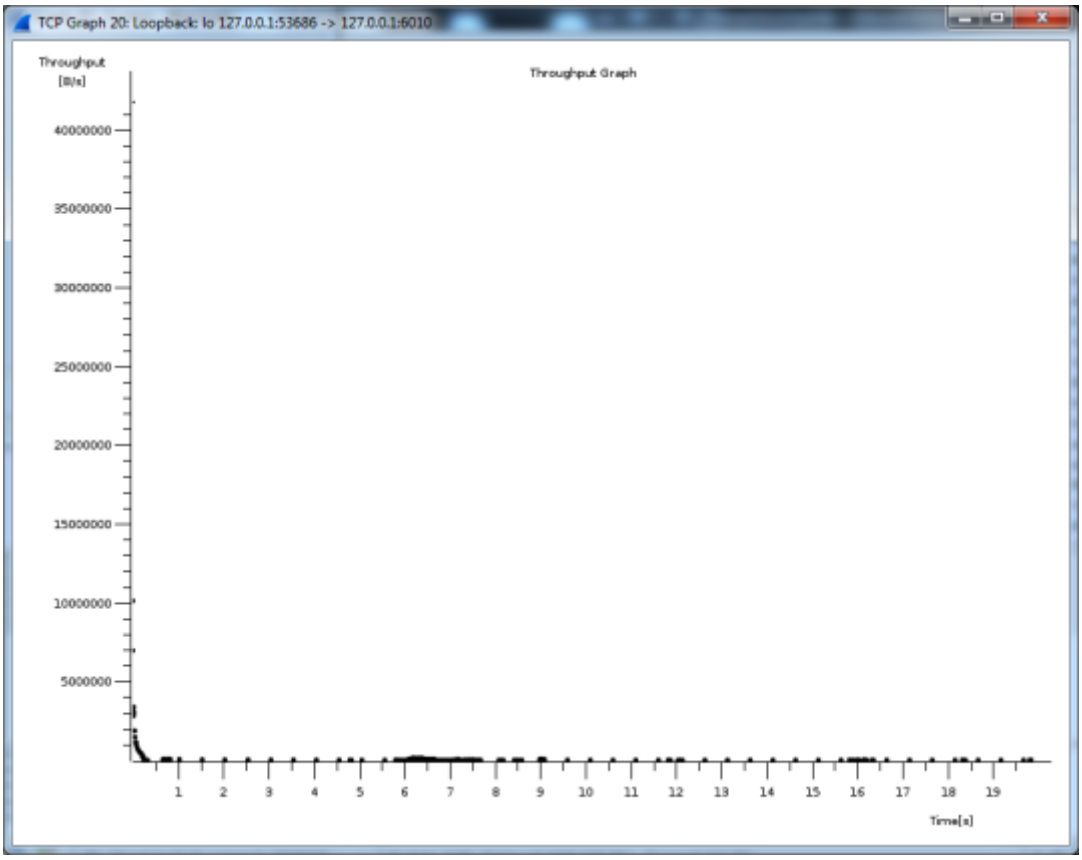
Εικόνα 5. 15 Echo Request & Echo Reply μηνύματα

Κατά την διάρκεια του παραπάνω πειράματος τα παρακάτω γραφήματα καταγράφηκαν ώστε να διαπιστώσουμε βασικά χαρακτηριστικά του δικτύου. Τα γραφήματα αυτά αφορούν χαρακτηριστικά όπως το Throughput μιας σύνδεσης, το γράφημα του Tcptrace και το Round Trip Time γράφημα.



Γράφημα 5. 8 Time Sequence Graph για το δίκτυο 4 κόμβων

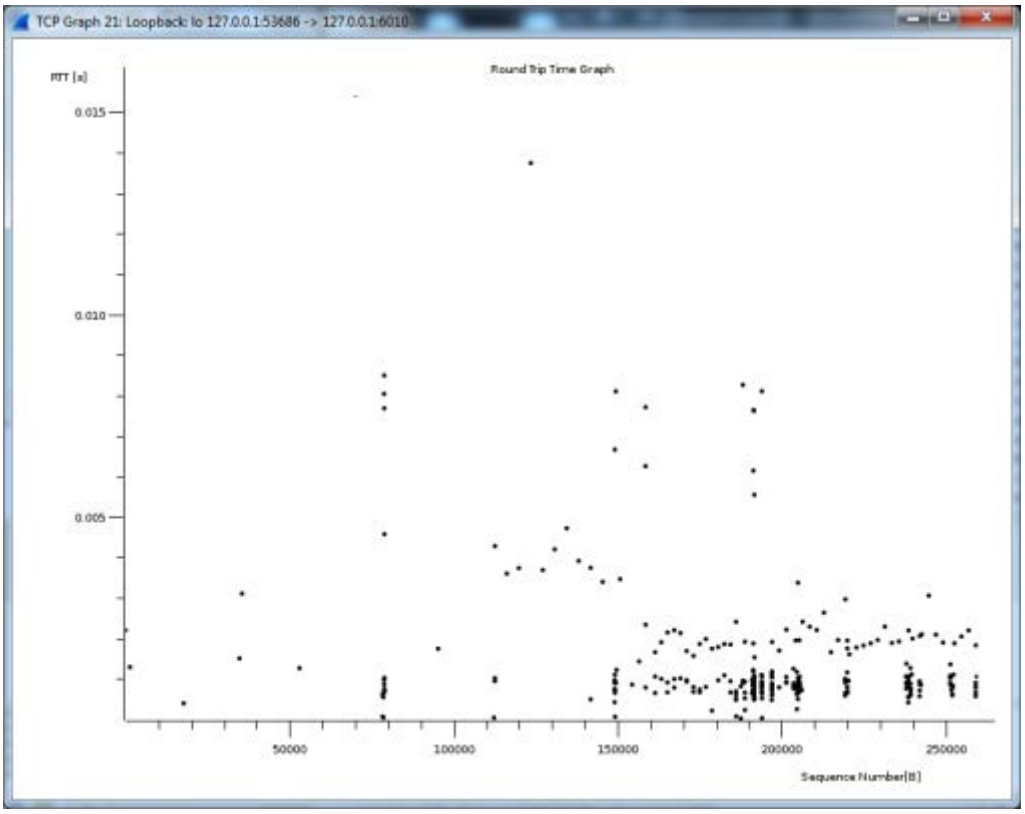
Στο Tcptrace γράφημα φαίνεται ο χ-άξονας που είναι ο χρόνος. Ο Υ-άξονας είναι οι αριθμοί ακολουθίας TCP. Ο αριθμός του κάθετου άξονα είναι αντιπροσωπευτικός των bytes που έχουν αποσταλεί. Ο αύξων αριθμός αυξάνεται κατά 1 για κάθε 1 byte δεδομένων TCP που έχει σταλεί. Ιδανικά θα βλέπαμε μια ομαλή γραμμή να ανεβαίνει προς τα δεξιά. Η κλίση της γραμμής είναι το θεωρητικό εύρος ζώνης της σύνδεσης. Όσο πιο απότομη είναι η γραμμή, τόσο υψηλότερη είναι η απόδοση. Τα μικρά μαύρα τμήματα αντιπροσωπεύουν τμήματα δεδομένων TCP. Όσο μεγαλύτερο είναι το μέγεθος των τμημάτων, τόσο περισσότερα δεδομένα έχουμε ανά πακέτο.



Γράφημα 5. 9 Throughput Γράφημα για το δίκτυο 4 κόμβων

Παραπάνω έχουμε δημιουργήσει ένα γράφημα που αφορά τις ταχύτητες διέλευσης των δεδομένων που ονομάζεται Throughput graph. Σημείωση: Οι ταχύτητες διέλευσης μετριούνται σε bytes / δευτερόλεπτο. Παρατηρούμε κίνηση μόνο κατά την εγκαθίδρυση της σύνδεσης. Στο γράφημα Round Trip Time Graph φαίνεται ο χρόνος καθυστέρησης μετ 'επιστροφής ή (RTT). RTT όπως αναφέραμε είναι ο χρόνος που χρειάζεται ένα σήμα

για να αποσταλεί συν ο χρόνος που χρειάζεται για την αναγνώριση του σήματος που πρόκειται να λάβει. Ως εκ τούτου, αυτή η χρονική καθυστέρηση αποτελείται από τους χρόνους μεταδόσεως μεταξύ δύο σημείων σήματος. Το σήμα είναι γενικά, ένα πακέτο δεδομένων, και ο RTT χρόνος είναι επίσης γνωστός ως χρόνος ping. Ένας χρήστης μπορεί να καθορίσει το χρόνο RTT και χρησιμοποιώντας την εντολή ping.



Γράφημα 5. 10 Χρόνοι rtt για το δίκτυο 4 κόμβων

5.2.1 POX Ελεγκτής

Εάν θέλουμε να τρέξουμε ένα διαφορετικό παράδειγμα στο οποίο θα χρησιμοποιούμε ένα άλλον ελεγκτή εκτός από τον προκαθορισμένο του Mininet , για παράδειγμα τον POX ελεγκτή, μπορούμε να κάνουμε το παρακάτω πείραμα ώστε να διαπιστώσουμε εάν υπάρχουν διαφορές στις τιμές των μεταβλητών που δείχνουν τα χαρακτηριστικά του δικτύου.

Καταρχήν καθαρίζουμε τον εξομοιωτή ώστε να μην έχουμε πρόβλημα - conflict με προηγούμενα πειράματα με την εντολή **mininet>exit** και την εντολή **$ sudo mn -c**. Θα πρέπει να δώσουμε μια εντολή σαν την παρακάτω ώστε να τρέχει ο ταχύτερος μεταγωγέας:

$ sudo mn --topo single,4 --mac --switch ovsk --controller remote [42]

Στην συνέχεια αν εκτελέσουμε τις παρακάτω εντολές θα είμαστε πλέον μέσα στο POX [22]

$ git clone http://github.com/noxrepo/pox

$ cd pox

Εάν θέλουμε να έχουμε τον μεταγωγέα να λειτουργεί σαν απλό hub

$./pox.py log.level --DEBUG misc.of_tutorial

Η παραπάνω γραμμή ζητάει καταγραφή των συνομιλιών μεταξύ των μεταγωγέα και του ελεγκτή και την έναρξη του of _tutorial στοιχείου.

Εάν δώσουμε την εντολή:

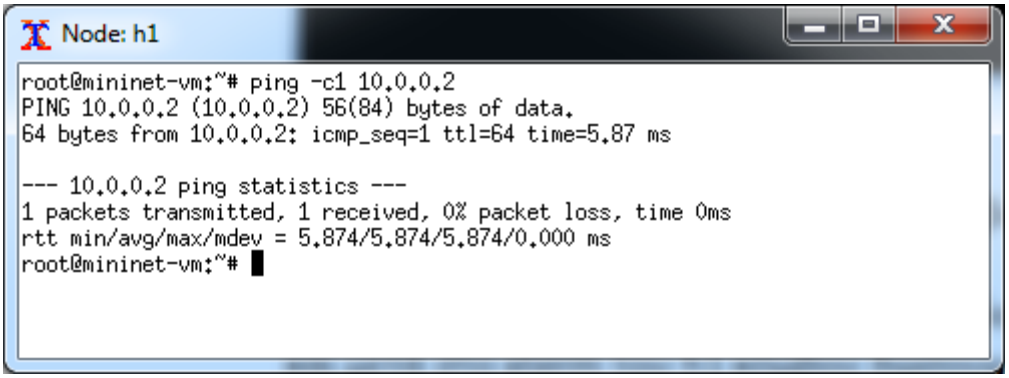
mininet> xterm h1 h2 h3 h4

Ανοίγουν σε αντίστοιχα παράθυρα οι κόμβοι μέσω του Xterm

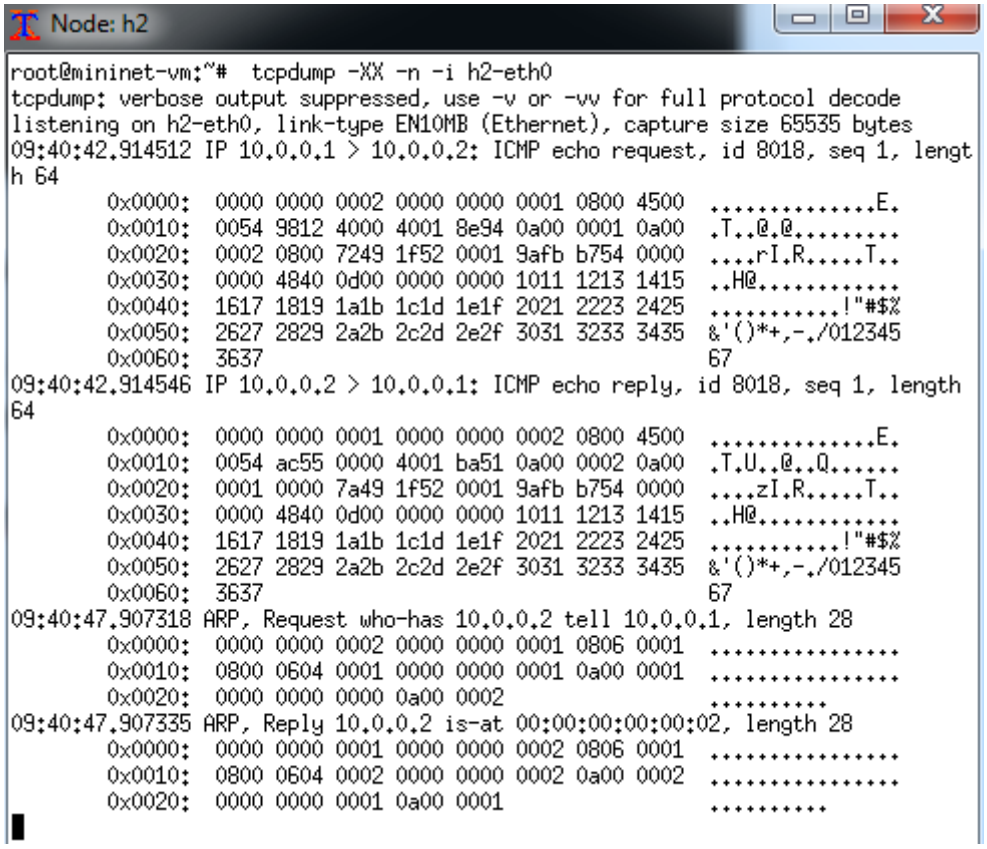
Εάν δώσουμε σε ένα από αυτά για παράδειγμα στο h2 **# tcpdump -XX -n -i h2-eth0** [22]

και μετά στο xterm του h1 κόμβου δώσουμε την εντολή

 # ping -c1 10.0.0.2 θα έχουμε την παρακάτω εικόνα στο term του h2 κόμβου που δείχνει ότι ο μεταγωγέας λειτουργεί ως hub.



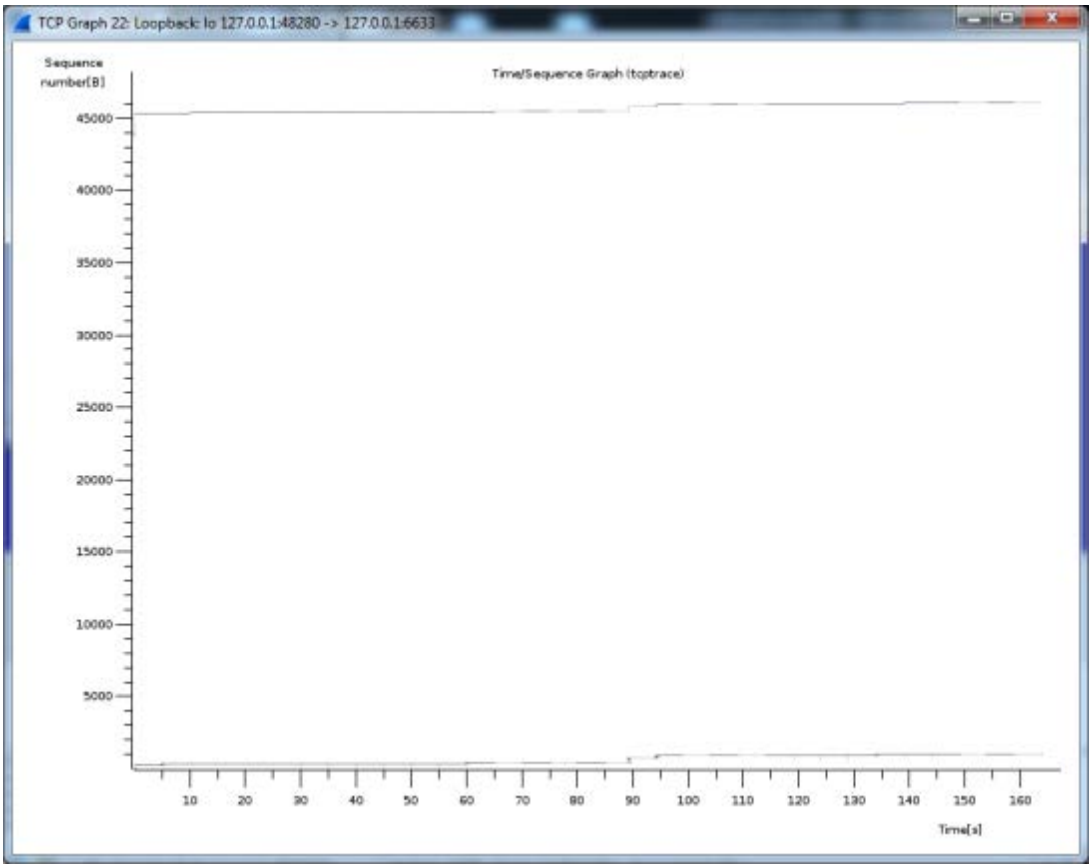
Εικόνα 5. 16 Αποτελέσματα ping -c1 10.0.0.2 στο δίκτυο 4 κόμβων.



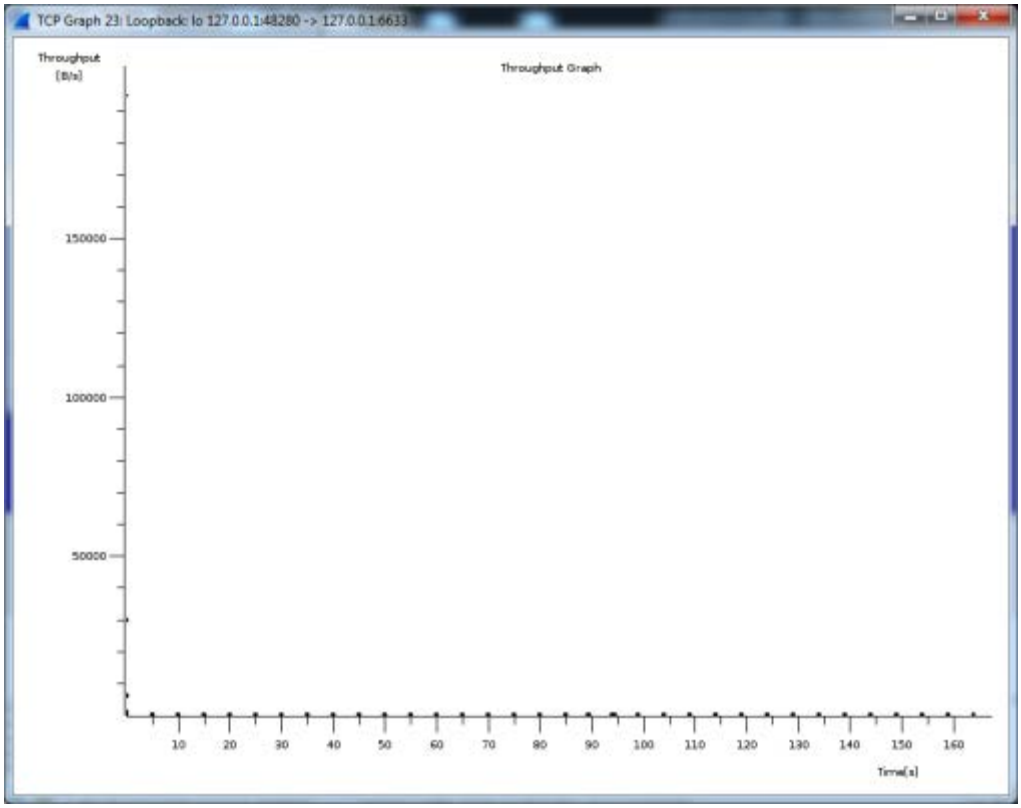
Εικόνα 5. 17 Αποτελέσματα # tcpdump -XX -n -i h2-eth0

Στην παραπάνω εικόνα βλέπουμε τα αποτελέσματα των echo request & echo reply μηνυμάτων καθώς και ερωτήσεις του ARP πρωτοκόλλου για το που βρίσκεται ο κόμβος 10.0.0.2 και την απάντηση του ARP πρωτοκόλλου. Με την εντολή **# tcpdump -XX -n -i h2-eth0** βλέπουμε ότι παίρνουμε πληροφορίες για τα πακέτα σε τιμές δεκαεξαδικού. Το tcpdump παρέχει έναν τρόπο ανάλογα την

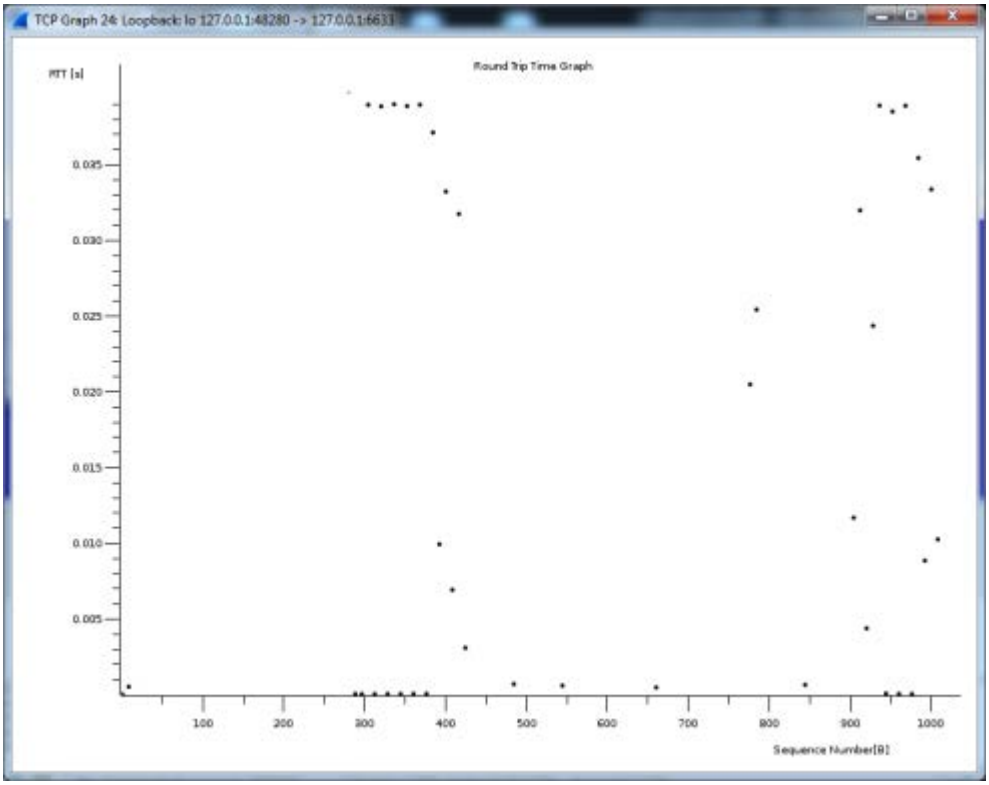
παράμετρο που θέτουμε ώστε να εκτυπώσουμε τα πακέτα τόσο σε ASCII όσο και σε HEX μορφή. Στην εντολή αυτή η παράμετρος -XX καθορίζει την δεκαεξαδική μορφή των στοιχείων που θα πάρουμε, το -n καθορίζει ότι θα συγκεντρώσουμε τα πακέτα που έχουν IP διευθύνσεις, το -i καθορίζει ότι θα ληφθούν τα πακέτα της συγκεκριμένης διεπαφής δηλαδή στην περίπτωση μας του eth0.



Γράφημα 5. 11 Time Sequence Number για το πείραμα με το POX



Γράφημα 5. 12 Throughput για το πείραμα με το POX



Γράφημα 5. 13 RTT χρόνοι για το POX πείραμα.

5.3 Δημιουργία Δικτύου με 3 Μεταγωγείς

Το πείραμα αυτό θα μας βοηθήσει να εξοικειωθούμε με το Mininet καθώς και να διαφοροποιήσουμε αρχεία κώδικα python ώστε να δημιουργούμε μόνοι μας τα επιθυμητά δίκτυα πέρα από τα προβλεπόμενα πειράματα που εμπεριέχονται στο Mininet. Για να τρέξουμε το παρακάτω πείραμα αντιγράφουμε τον κώδικα python από την διεύθυνση git clone *https://github.com/OMS6250/gt-cs6250.git* στην εικονική μηχανή του Mininet και κάνουμε update με τις παρακάτω εντολές:

$ sudo sed -i "s/security.ubuntu.com/old-releases.ubuntu.com/g" /etc/apt/sources.list

$ sudo sed -i "s/mirrors.kernel.org/old-releases.ubuntu.com/g" /etc/apt/sources.list

$ sudo apt-get update

Εγκαθιστούμε δυο βιβλιοθήκες με τις οποίες έχουμε δημιουργία γραφημάτων [52]

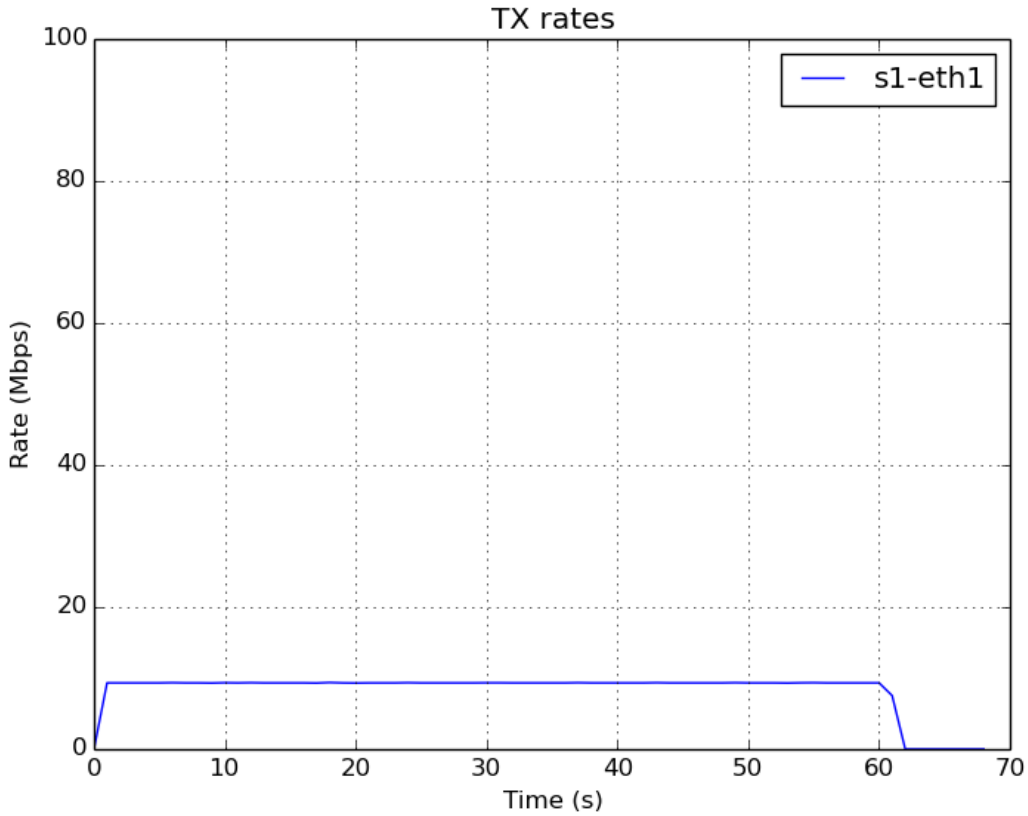
$ sudo apt-get install bwm-ng python-matplotlib

Αλλάζουμε το directory στο οποίο βρισκόμαστε με την παρακάτω εντολή [52]:

$ cd gt-cs6250/assignment-2

Έπειτα τρέχουμε την τοπολογία **$ sudo** ./**topology.sh** που υπάρχει διαθέσιμη στο φάκελο. Το script παράγει αποτελέσματα με χρονική σφραγίδα καθώς και κάποια γραφήματα για το TCP congestion window και ένα για το bandwidth σε Megabits per second. Για να δούμε τα γραφήματα απλά θα πρέπει να ξεκινήσουμε ένα απλό web server και να πλοηγηθούμε στην IP διεύθυνση της εικονικής μηχανής. Την IP διεύθυνση την βλέπουμε συσχετιζόμενη με την διεπαφή eth0 δίνοντας την εντολή **$ sudo ifconfig**. Στην συνέχεια εκκινούμε τον

εξυπηρετητή HTTP με την εντολή **$ python -m SimpleHTTPServer**. Στον υπολογιστή που φιλοξενεί την εικονική μας μηχανή πηγαίνουμε στην ηλεκτρονική διεύθυνση *http://ip_address:8000* με την βοήθεια ενός φυλλομετρητή που διαθέτουμε, εκεί μπορούμε να δούμε τους φακέλους που έχει δημιουργήσει το script και να παρατηρήσουμε τα σχετικά γραφήματα πέρα από την καταγραφή των μετρήσεων σε αρχεία.

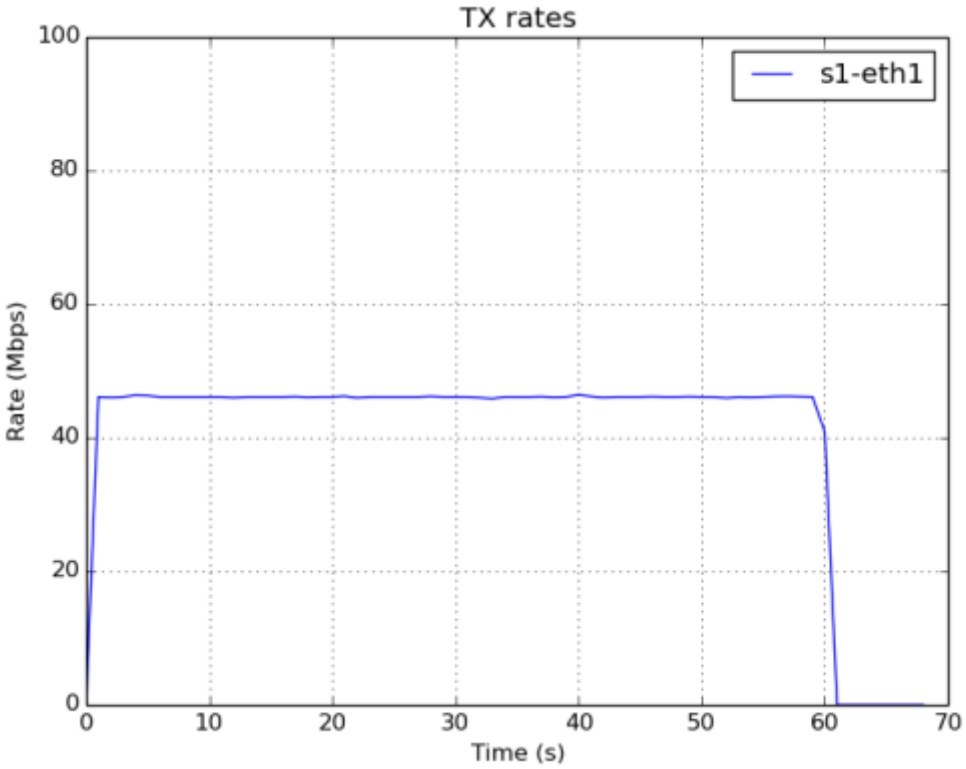


Γράφημα 5. 14 Μέτρηση της σύνδεσης μεταξύ s1-eth1

Εναλλακτικά, από τα παραπάνω βήματα μπορούμε να κατεβάσουμε τον κώδικα καθώς και τα *.sh αρχεία από την παρακάτω διεύθυνση: *https://github.com/OMS6250/gtcs6250/tree/56782b16b227fb251dd40016b1c80ff2a0d3e1b1/assignment-2*

Μπορούμε να επέμβουμε στον κώδικα ώστε να αλλάξουμε το εύρος ζώνης στο script ορίζοντας οι συνδέσεις να είναι 50 Mbps και να έχουν μια καθυστέρηση 10 msec. Για να δούμε ότι τρέχει σωστά παρατηρούμε την εικόνα του Mininet για

αναφορά λαθών καθώς και την αναφορά για την ταχύτητα των συνδέσεων. Επίσης ελέγχουμε τα αποτελέσματα του ping ώστε να επικοινωνούν σωστά οι κόμβοι και να μην απορρίπτουν πακέτα.

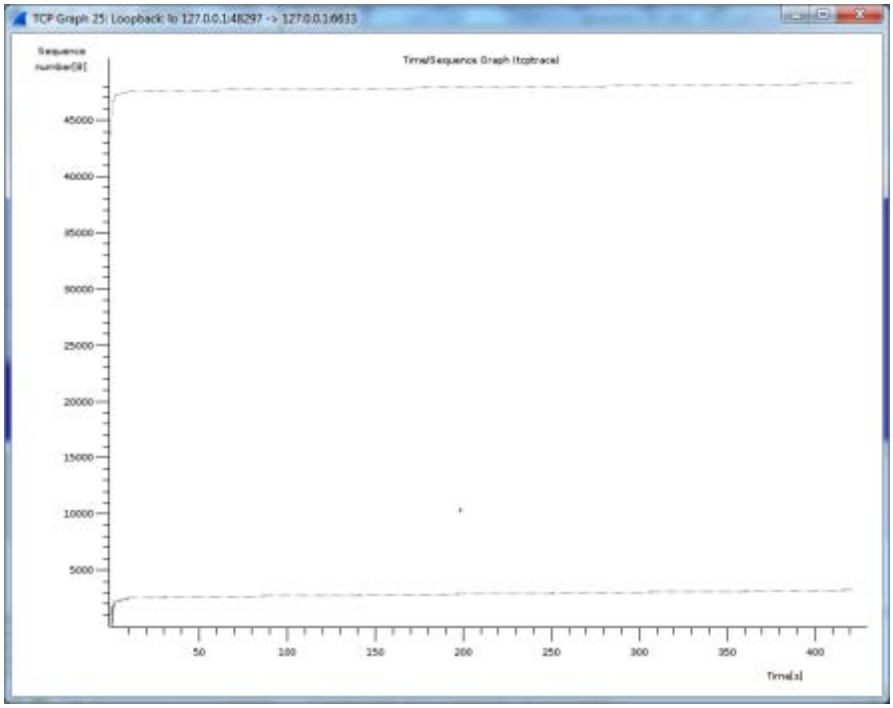


Γράφημα 5. 15 Ταχύτητα σύνδεσης μεταξύ κόμβων δικτύου 50 Mbps

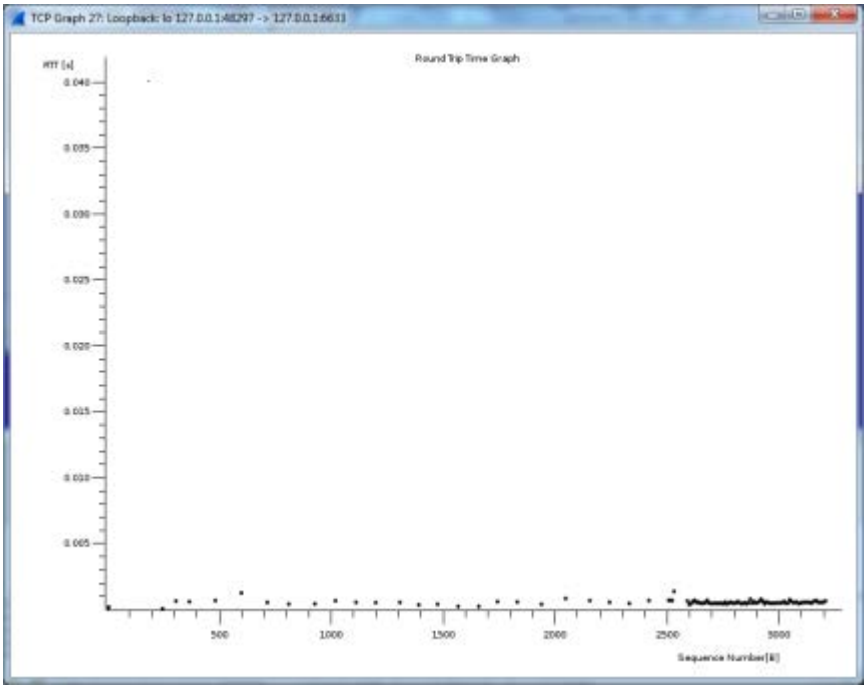
Παρατηρούμε από τα εξαγόμενα αποτελέσματα και ιδίως από το παραπάνω γράφημα 5.15 ότι η ταχύτητα πλέον της σύνδεσης μεταξύ κόμβου και μεταγωγέα είναι πλέον στα 50 Mbps. Προσθέσαμε λοιπόν δυο ακόμα μεταγωγείς και τις συνδέσεις τους και παρατηρούμε ότι το δίκτυο λειτουργεί ομαλά παρά τις αλλαγές που επιχειρήσαμε.

Τα παρακάτω γραφήματα συλλέχθηκαν με την βοήθεια του Wireshark. Παρατηρούμε ότι η απόδοση του δικτύου δεν είναι ικανοποιητική. Η αύξηση της απόδοσης του δικτύου καθώς και ο έλεγχος του πως θα μπορούσαμε να εξετάσουμε χαρακτηριστικά δυσλειτουργίας του δικτύου, μπορεί να γίνει σε κάποιο επόμενο στάδιο. Δυστυχώς ο χρόνος που διαθέτουμε για την

συγκεκριμένη διατριβή δεν είναι ικανός ώστε να εντρυφήσουμε σε βάθος στα θέματα αυτά.



Γράφημα 5. 16 Time Sequence δίκτυο με 3 Μεταγωγείς



Γράφημα 5. 17 RTT χρόνοι στο δίκτυο με 3 Μεταγωγείς

5.4 Μελέτη Απόδοσης Δικτύων

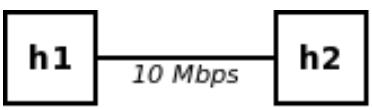
Στο πείραμα αυτό θα μετρήσουμε και θα αναλύσουμε την απόδοση μικρών δικτύων με την βοήθεια της εντολής --iperf [37]. Θα αναλύσουμε διαφορετικές τοπολογίες με την βοήθεια του Mininet. Τα διαθέσιμα αρχεία που αφορούν την τοπολογία υπό εξέταση βρίσκονται στο παρακάτω *.tar αρχείο *http://cs.wisc.edu/~agember/cs640/s14/files/project1.tgz.*

Για να κατεβάσουμε τα αρχεία στο Mininet πρέπει να εκτελέσουμε τις παρακάτω εντολές

$ wget http://cs.wisc.edu/~agember/cs640/s14/files/project1.tgz
$ tar xzvf project1.tgz

5.4.1 Δίκτυο Δυο Κόμβων Απευθείας Συνδεδεμένων.

Στο παρακάτω πείραμα θα εξετάσουμε μια σύνδεση μεταξύ δυο κόμβων. Στο σχήμα 5.18 φαίνεται ένα σχεδιάγραμμα της εν λόγω σύνδεσης. Ο στόχος είναι να δούμε την λειτουργία του δικτύου χωρίς τον μεταγωγέα καθώς και να εξοικειωθούμε με τις εντολές δημιουργίας κίνησης του Mininet. Στην συγκεκριμένη περίπτωση θα εξακριβωθεί αν οι συνδέσεις είναι «πραγματικά διπλές» και κατά πόσον η ροή της κυκλοφορίας σε κάθε κατεύθυνση είναι απομονωμένη. Το δίκτυο αποτελείται από δύο ζεύγη. Αυτό το πείραμα είναι μια από τις περιπτώσεις, όπου το Mininet-HiFi σύμφωνα με τον Heller [08] δίνει τα ίδια αποτελέσματα και έχουμε την ίδια ακριβώς κατανομή εύρους ζώνης σε σχέση με τη πάροδο του χρόνου, σε σύγκριση με την πραγματική υλοποίηση του δικτύου σε φυσική μορφή, σύμφωνα με την εν λόγω διδακτορική διατριβή.

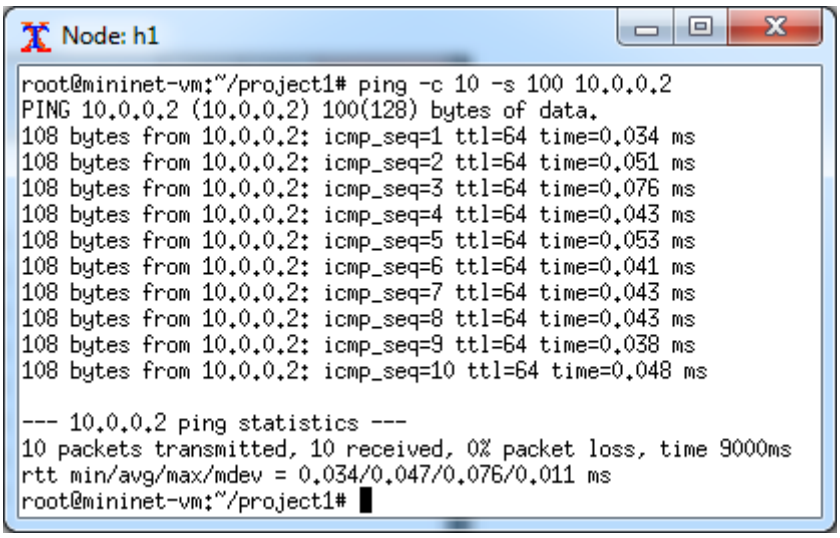


Εικόνα 5. 18 Δίκτυο από 2 κόμβους απευθείας συνδεδεμένους [37]

Για να δημιουργήσουμε την συγκεκριμένη τοπολογία θα πρέπει καταρχήν να είμαστε στον κατάλογο που έχουμε τοποθετήσει το project1 και στην συνέχεια να δώσουμε την εντολή **$ sudo python project1_point.py** με αυτόν τον τρόπο θα τρέξει το αρχείο που αποτελείται από κώδικα python.

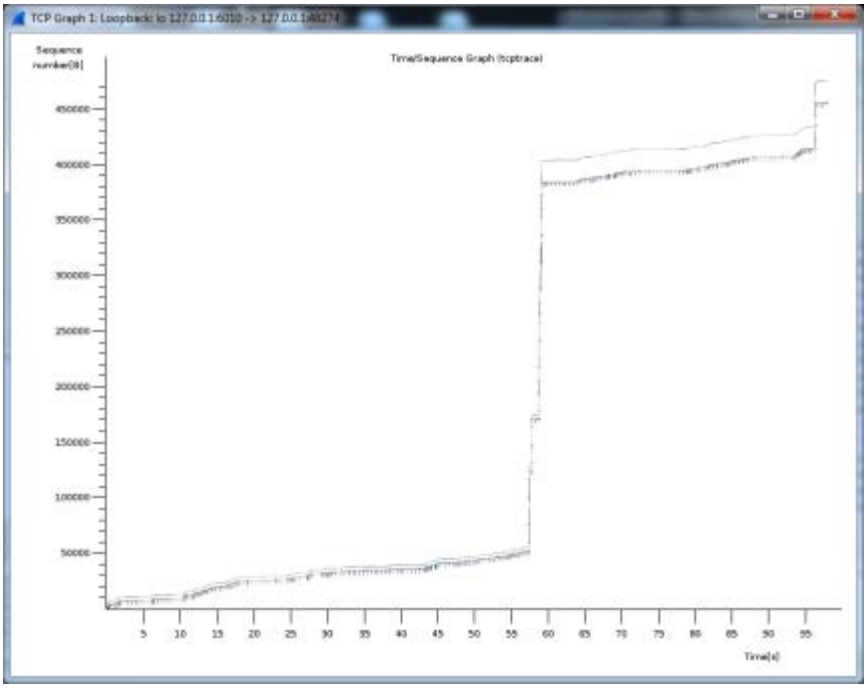
Για να εμφανιστούν οι κόμβοι μέσω X11 και να μπορέσουμε να δώσουμε εντολές εύκολα σε αυτούς δίνουμε την εντολή mininet> **xterm h1 h2**. Εμφανίζεται η παρακάτω εικόνα και θα ανοίξουν δύο παράθυρα για τον κάθε κόμβο αντίστοιχα.

Παρακάτω μπορούμε να συγκρίνουμε από τα παράθυρα xterm, τα αποτελέσματα που πήραμε από το πείραμα με τους δύο κόμβους που είναι απευθείας συνδεδεμένοι μεταξύ τους, δίνοντας την εντολή ping.

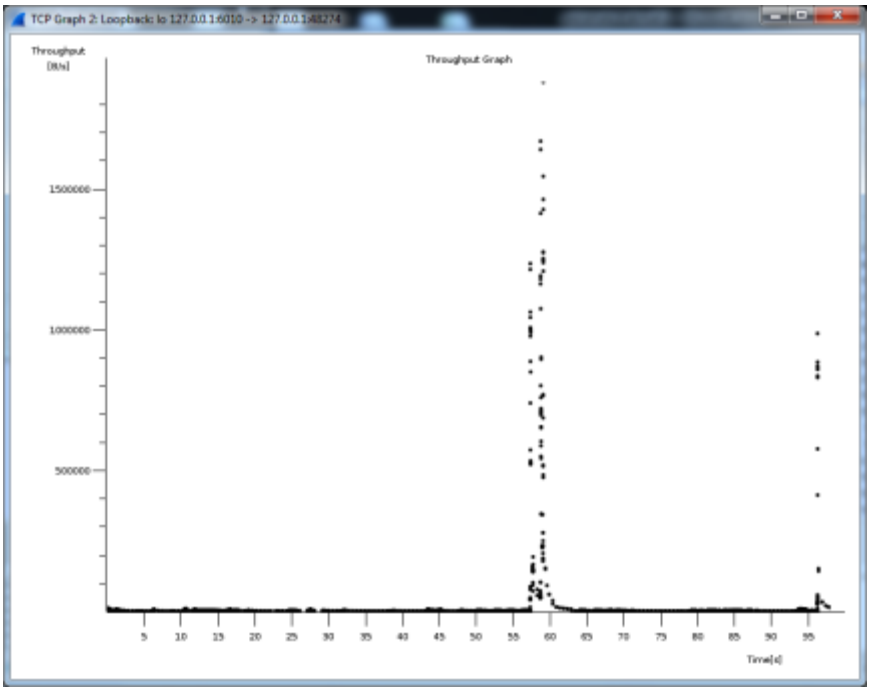


Εικόνα 5. 19 Ping αποτελέσματα στο δίκτυο με τους άμεσα συδεδεμένους κόμβους

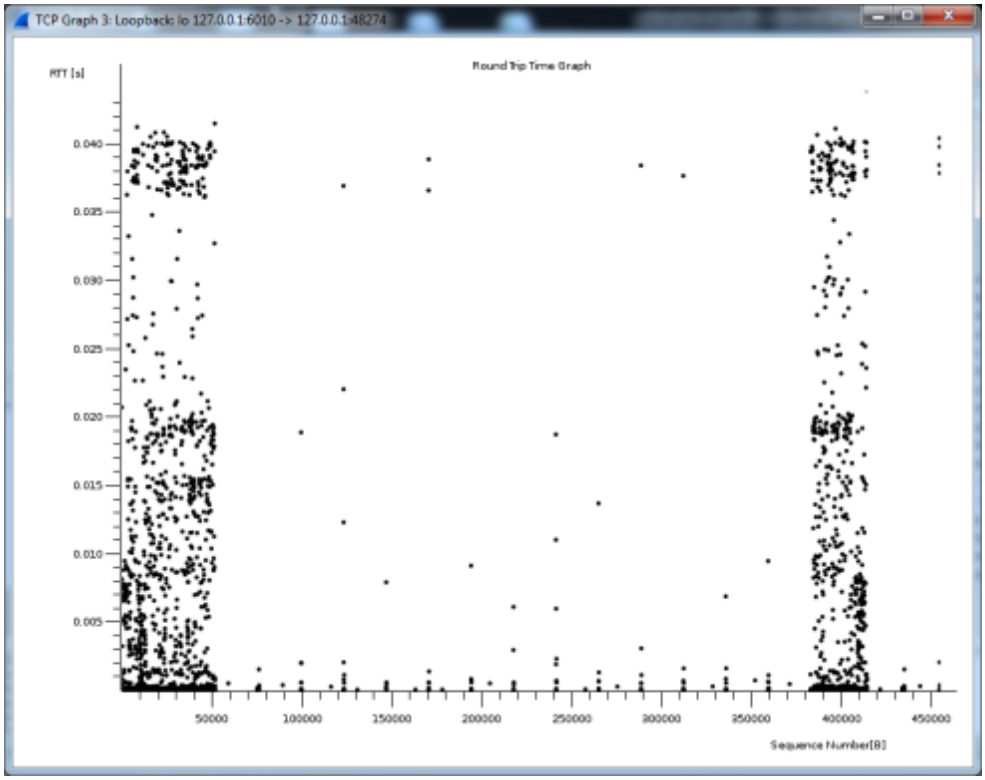
Κατά την διάρκεια της δημιουργίας του δικτύου έγινε λήψη των παρακάτω γραφημάτων με την βοήθεια του Wireshark. Από το παρακάτω γράφημα παρατηρούμε ότι η απόδοση του δικτύου μπορεί να μην είναι ιδανική αλλά διατηρεί αυξητική πορεία. Το γράφημα προέκυψε κατά την δημιουργία του δικτύου και για αυτό εμφανίζει τα σκαλοπάτια. Η αυξητική πορεία της γραμμής δηλώνει όπως γνωρίζουμε σταθερά αυξητική απόδοση γεγονός που δείχνει ότι το δίκτυο είναι σε καλή κατάσταση.



Γράφημα 5. 18 Time /Sequence Graph για το δίκτυο 2 κόμβων απευθείας συνδεδεμένων.



Γράφημα 5. 19 Throughput στο δίκτυο με τους άμεσα συνδεδεμένους κόμβους.



Γράφημα 5. 20 RTT χρόνοι στο δίκτυο με τους άμεσα συνδεδεμένους κόμβους.

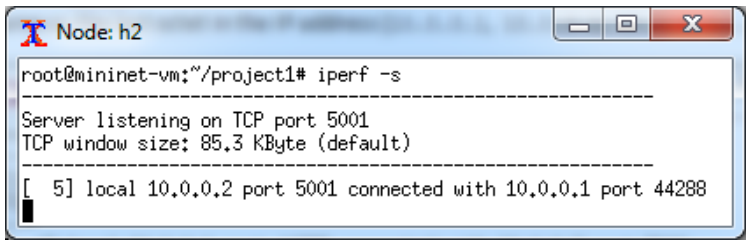
Παρακάτω θα μετρήσουμε το εύρος ζώνης μεταξύ δύο κόμβων χρησιμοποιώντας την εντολή iperf. Η εντολή αυτή αποστέλλει TCP ή UDP πακέτα μεταξύ δυο κόμβων. Ο ένας κόμβος θα πρέπει να παίζει τον ρόλο του εξυπηρετητή και ο άλλος τον ρόλο του πελάτη. Η εντολή που δίνουμε για να δημιουργήσουμε τον εξυπηρετητή είναι η παρακάτω: **$ iperf -s**

Η εντολή που δίνουμε στον πελάτη είναι η εξής: **iperf -c <ip>** όπου στην θέση της IP βάζουμε την διεύθυνση του εξυπηρετητή. Επίσης όσον αφορά γενικά την παραμετροποίηση των εντολών μπορούμε να δώσουμε την παράμετρο **-i** <δευτερόλεπτα> για να καθορίσουμε κάθε πότε θα πρέπει να μας επιστρέφει το εύρος ζώνης και την παράμετρο **-t** <δευτερόλεπτα> για να καθορίσουμε για πόσο χρονικό διάστημα ο πελάτης να μεταδίδει δεδομένα. Για να δώσουμε αυτές τις εντολές αρκεί να ανοίξουμε κάθε κόμβο σε διαφορετικό παράθυρο με την εντολή mininet>**xterm**

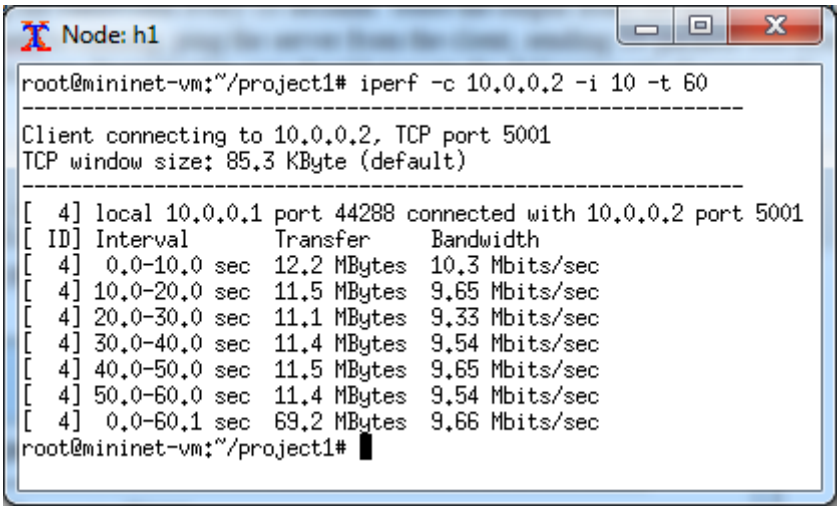
Έτσι για παράδειγμα μπορούμε να δώσουμε τις δυο παρακάτω εντολές ώστε να δημιουργηθεί ένα ζεύγος που μεταδίδει πληροφορία και μέσω της εντολής iperf να εξάγουμε συμπεράσματα για την γραμμή και την λειτουργία του δικτύου. Δημιουργούμε εξυπηρετητή τον h2 και πελάτη τον h1. Ο πελάτης δημιουργεί κίνηση και εξάγει το εύρος ζώνης της γραμμής κάθε δέκα δευτερόλεπτα ενώ η διάρκεια αυτής της κίνησης είναι συνολικά εξήντα δευτερόλεπτα.

h2$ iperf -s και **h1$ iperf -c 10.0.0.2 -i 10 -t 60**

Σημείωση: Μέσα στο xterm δεν μπορεί να δοθεί εντολή ως **iperf -c h2**. Η εντολή πρέπει να έχει την μορφή που αναφέρθηκε προτύτερα. Η εικόνα που θα έχουμε στα παράθυρα xterm είναι η παρακάτω:

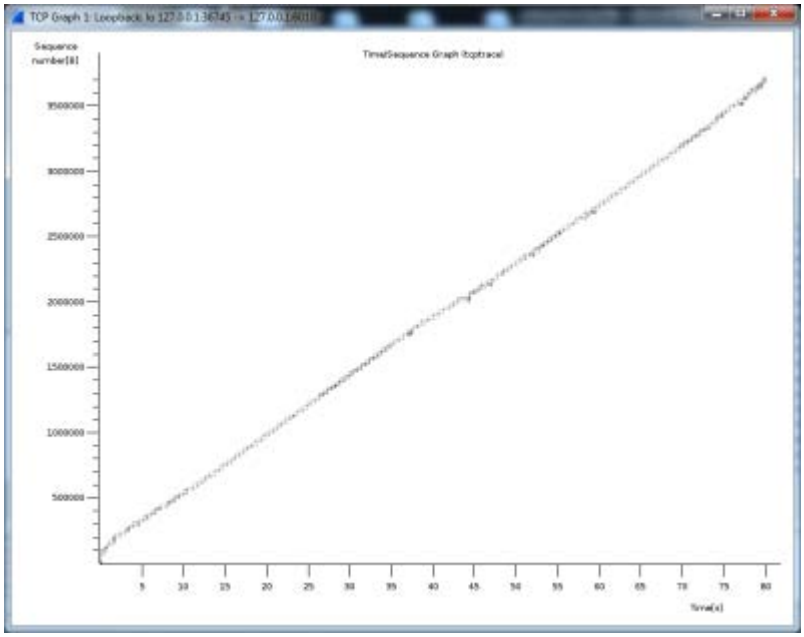


Εικόνα 5. 20 Εντολή δημιουργίας εξυπηρετητή στον h2 κόμβο.



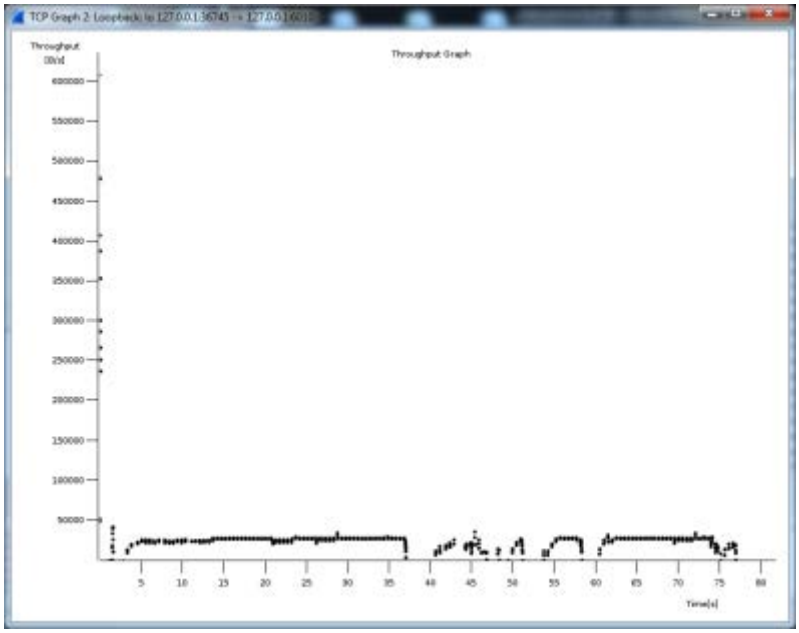
Εικόνα 5. 21 Εντολή δημιουργίας πελάτη στον h1 κόμβο. Φαίνεται το εύρος ζώνης της σύνδεσης.

Τα γραφήματα που συλλέχθηκαν κατά την διάρκεια του iperf ακολουθούν.



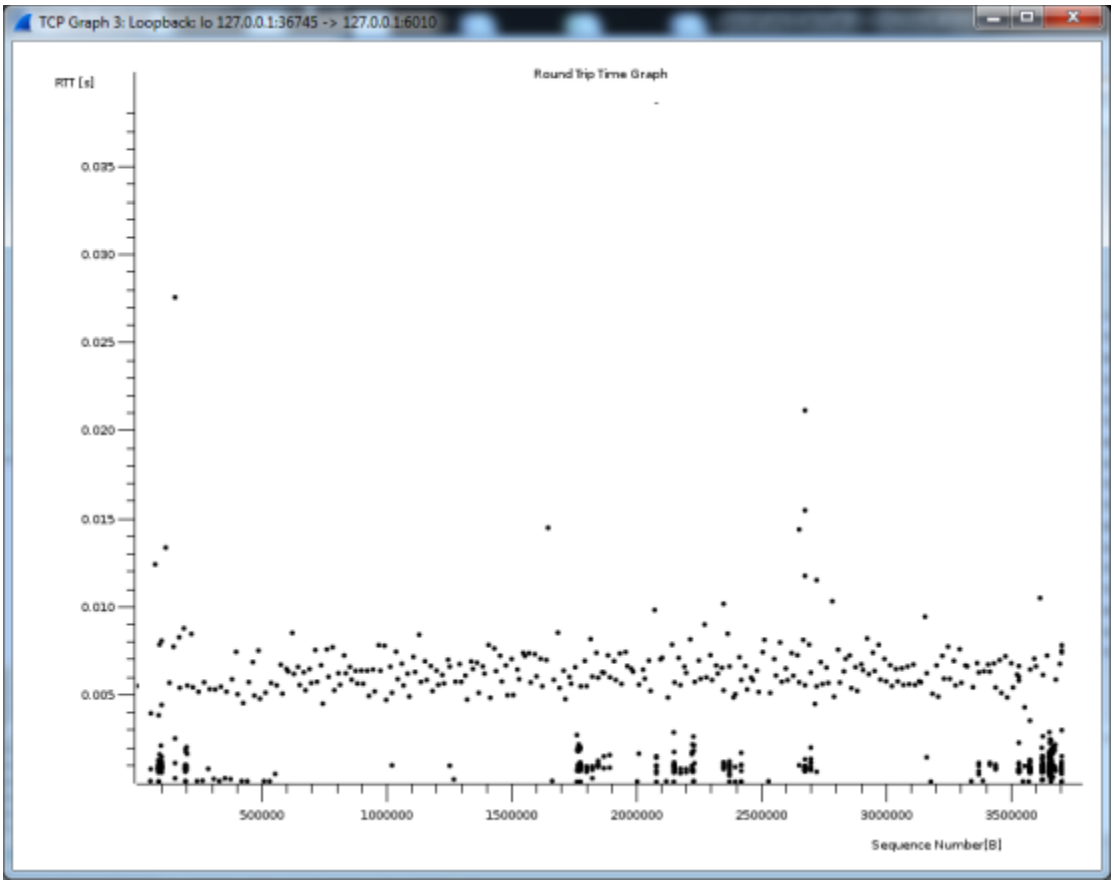
Γράφημα 5. 21 Sequence Graph κατά την διάρκεια εκτέλεσης της εντολής iperf.

Παρατηρούμε ότι το Time Sequence γράφημα παρουσιάζει την ιδανική εικόνα που είχαμε αναφέρει σε προηγούμενες παραγράφους. Η απόδοση του δικτύου είναι σε ιδανικά επίπεδα και δεν υπάρχει φυσικά πρόβλημα κατά την λειτουργία του.



Γράφημα 5. 22 Throughput κατά την διάρκεια του iperf.

Από τα παραπάνω γραφήματα (5.21-5.22) παρατηρούμε την ποσότητα της πληροφορίας που ανταλλάχθηκε μεταξύ των κόμβων του δικτύου κατά την εκτέλεση της εντολής iperf.

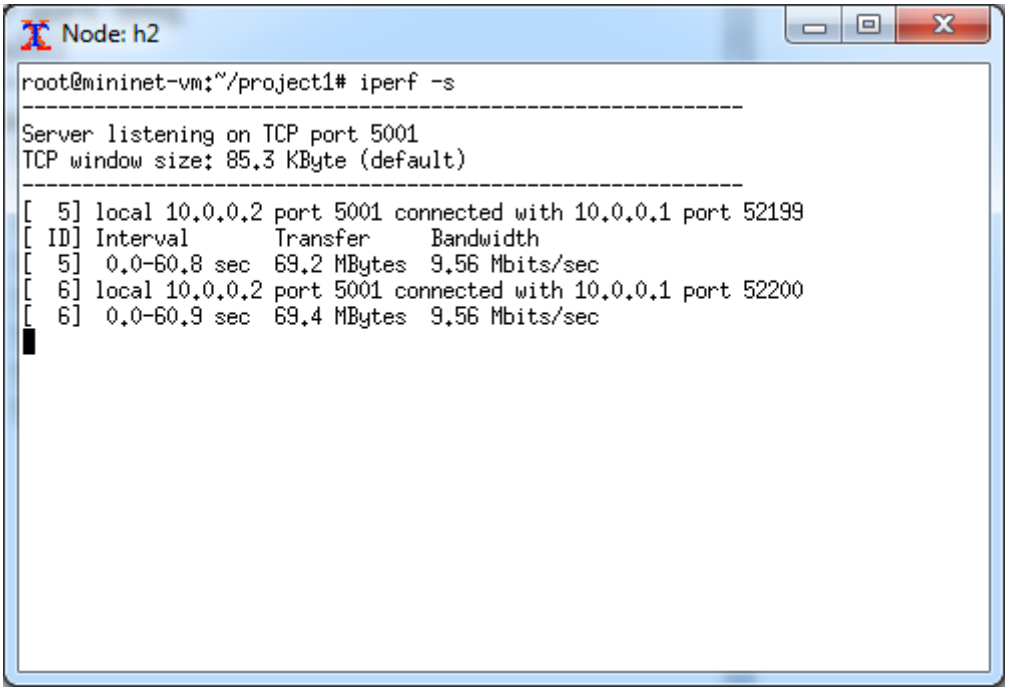


Γράφημα 5. 23 RTT χρόνοι κατά την διάρκεια του iperf.

Μπορούμε τα αποτελέσματα που εξάγονται από το iperf να τα αποθηκεύσουμε σε αρχείο καταγραφής άμεσα, για παράδειγμα, με την εντολή **h1$ iperf -c 10.0.0.2 -i 10 -t 60 > iperf.log.** Εάν δεν θέλουμε κάτι τέτοιο τα αποτελέσματα εμφανίζονται όπως είδαμε και παραπάνω μέσα στο παράθυρο xterm. Στην συγκεκριμένη διατριβή τα παρουσιάζω σαν εικόνες ώστε να είναι πιο εύκολο να εξαχθούν συμπεράσματα, από το να επισυνάπτονταν σε αυτήν ως συνοδευτικά αρχεία σε παράρτημα.

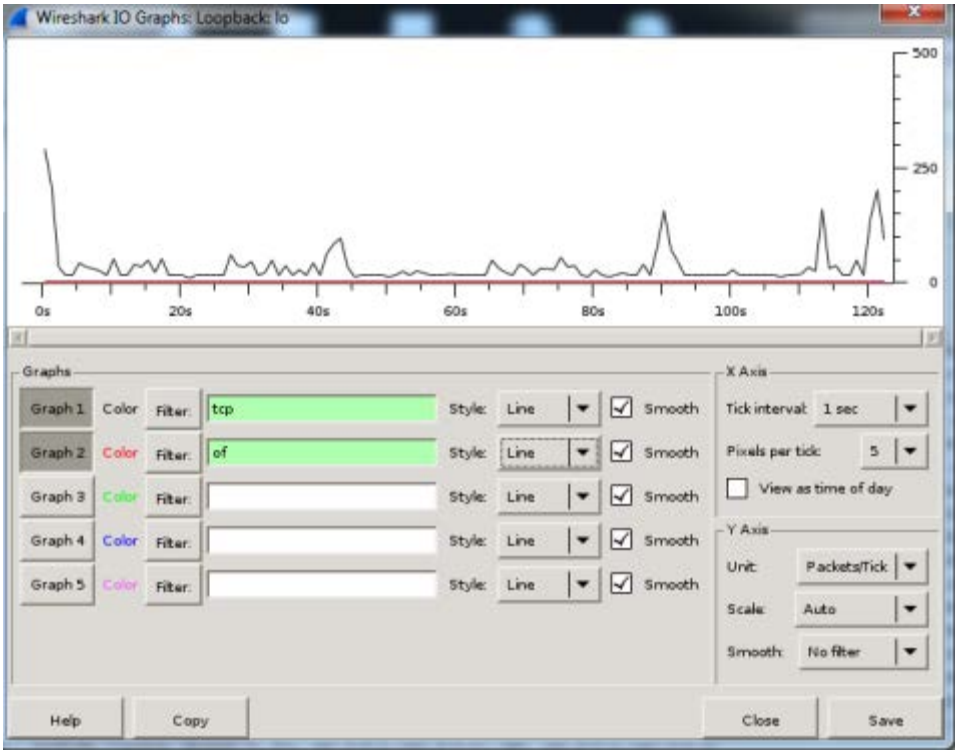
Εάν θέλουμε να δούμε πως αλλάζουν τα συλληφθέντα στοιχεία, μπορούμε να προσθέσουμε μια εντολή ping ώστε να δημιουργηθεί επιπλέον κίνηση μεταξύ των κόμβων. Έτσι για παράδειγμα μπορούμε να δώσουμε την εντολή:

h1$ ping -c 10 -s 100 10.0.0.2



Εικόνα 5. 22 Εικόνα του εξυπηρετητή κατά την διάρκεια εκτέλεσης των εντολών στο δίκτυο.

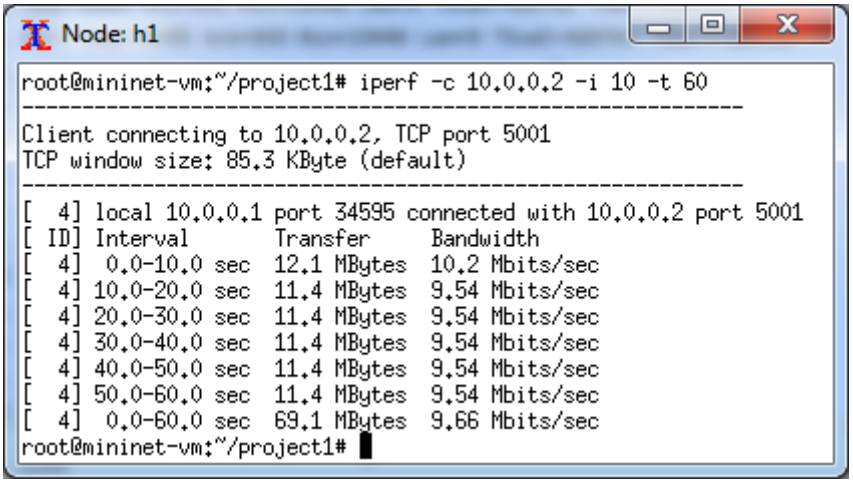
Στην παραπάνω εικόνα βλέπουμε την πόρτα του εξυπηρετητή, στην οποία "ακούει" που είναι η 5001. Αυτό φαίνεται και στο παραπάνω παράθυρο που φαίνονται οι συνδέσεις που δημιουργήθηκαν στον εξυπηρετητή κατά την διάρκεια του πειράματος. Στην εικόνα που ακολουθεί παρατηρούμε το γραφήμα που λήφθηκε μέσω wireshark για την περίπτωση που δώσουμε την iperf χωρίς την εντολή ping.



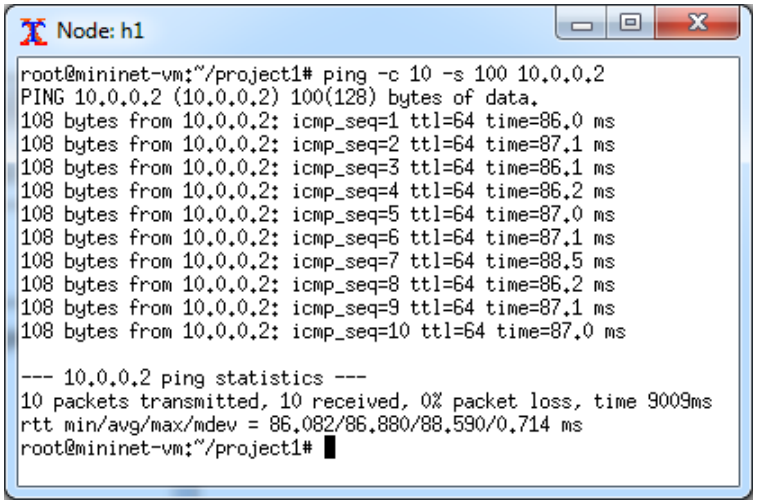
Γράφημα 5. 24 Γράφημα ροής TCP και OF κατά την εκτέλεση του πειράματος.

Στα παρακάτω σχήματα βλέπουμε τα δεδομένα που συλλέχτηκαν όταν δώσαμε την εντολή ping μαζί με το iperf, σχεδόν ταυτόχρονα.

h1#ping -c 10 -s 100 10.0.0.2

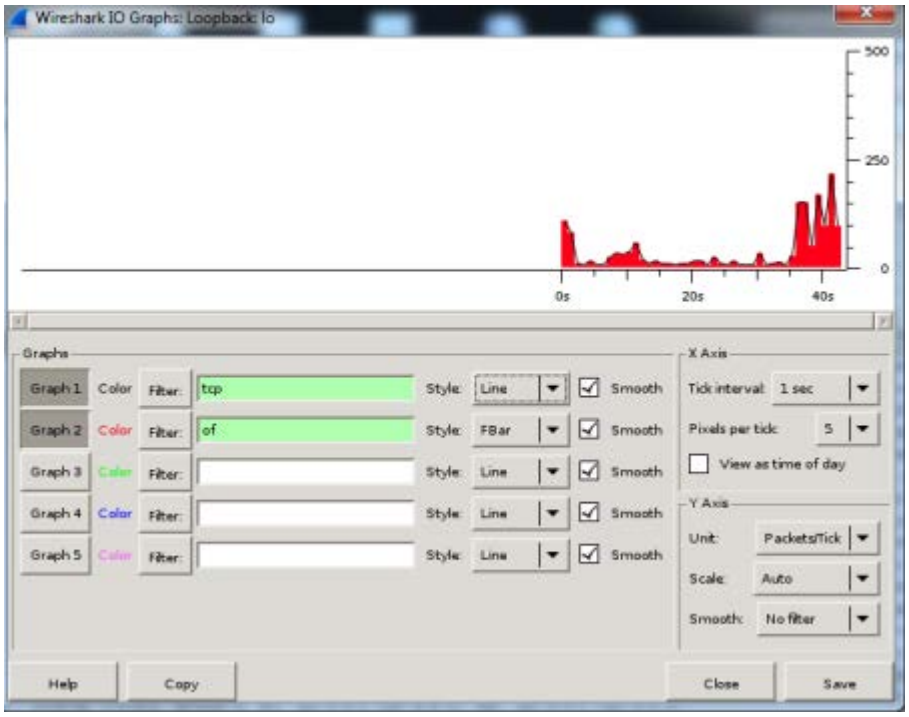


Εικόνα 5. 23 Δεδομένα iperf κατά την διάρκεια της ping εντολής στο δίκτυο.

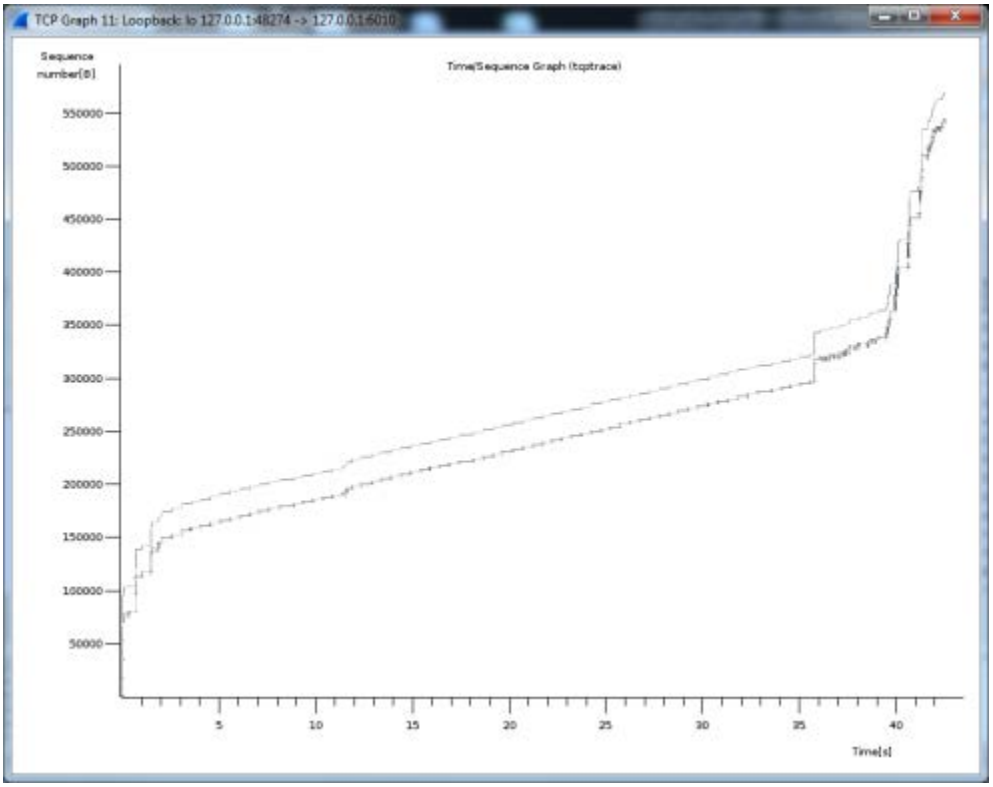


Εικόνα 5. 24 Αποτελέσματα της ping εντολής ενώ εκτελούσαμε και την iperf εντολή στο δίκτυο.

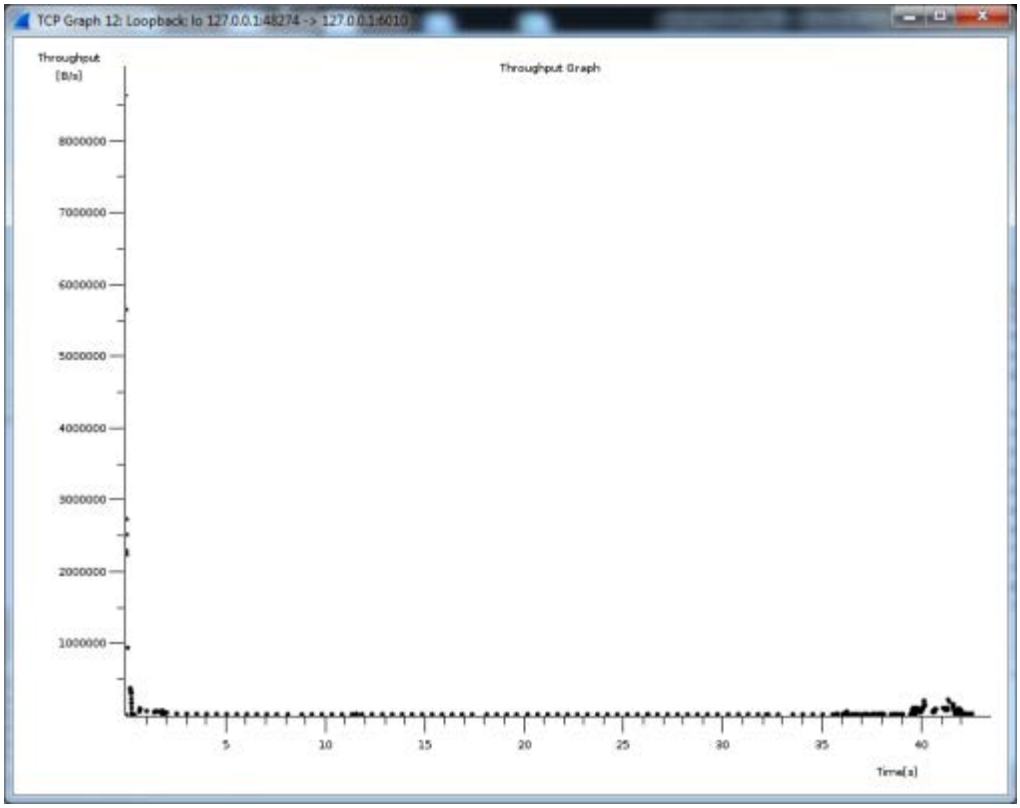
Οι εικόνες που παρουσιάζονται παραπάνω είναι με εκτέλεση των δυο εντολών iperf και ping για δημιουργία κίνησης. Η σύνταξη των εντολών φαίνεται στα αντίστοιχα παράθυρα. Τα γραφήματα δημιουργήθηκαν με το Wireshark κατά την διάρκεια του πειράματος.



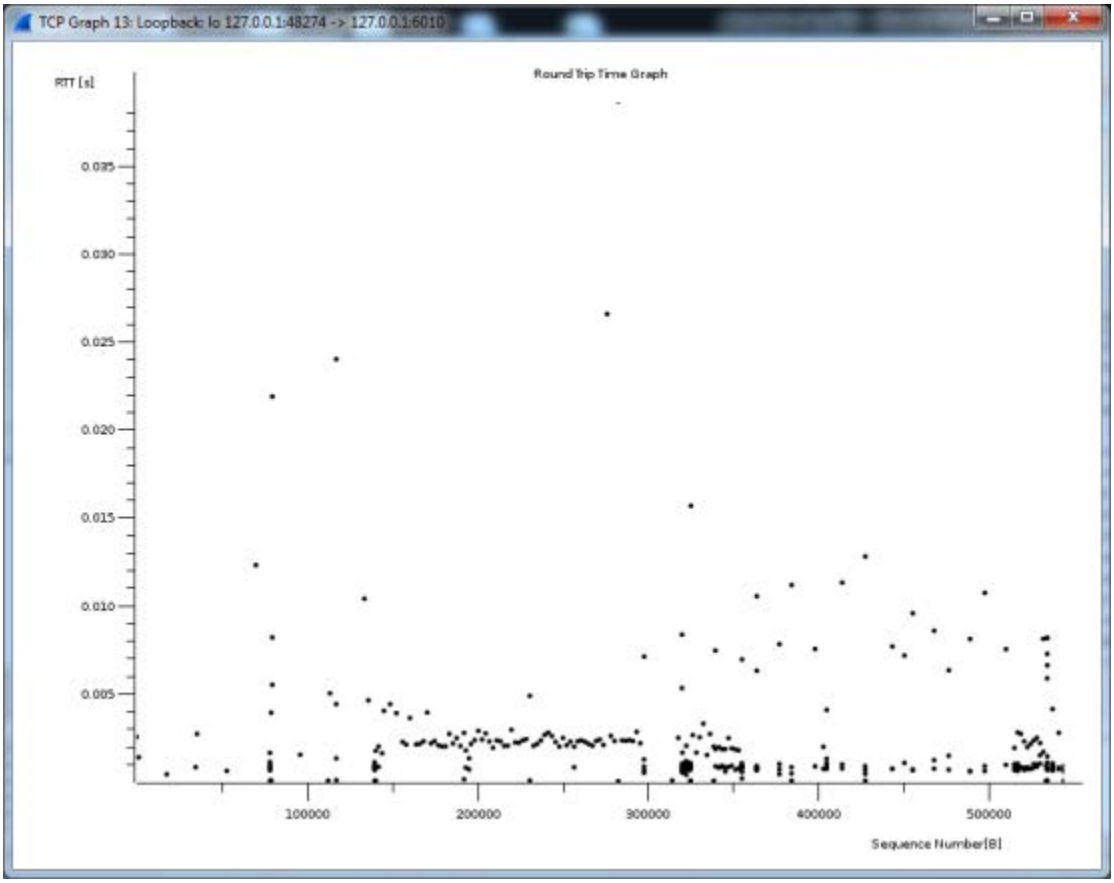
Γράφημα 5. 25 TCP & OF ροή στο δίκτυο



Γράφημα 5. 26 Time Sequence όταν εκτελούμε ping & iperf.



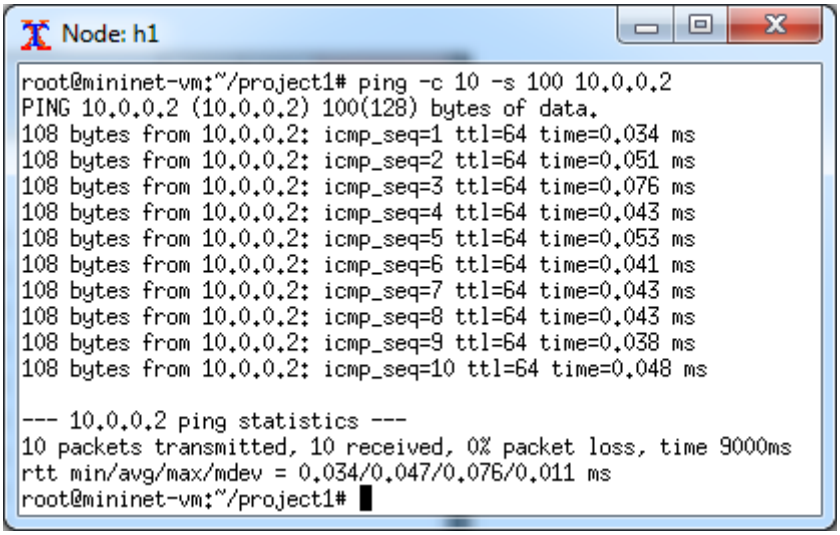
Γράφημα 5. 27 Throughput όταν εκτελούμε iperf & ping



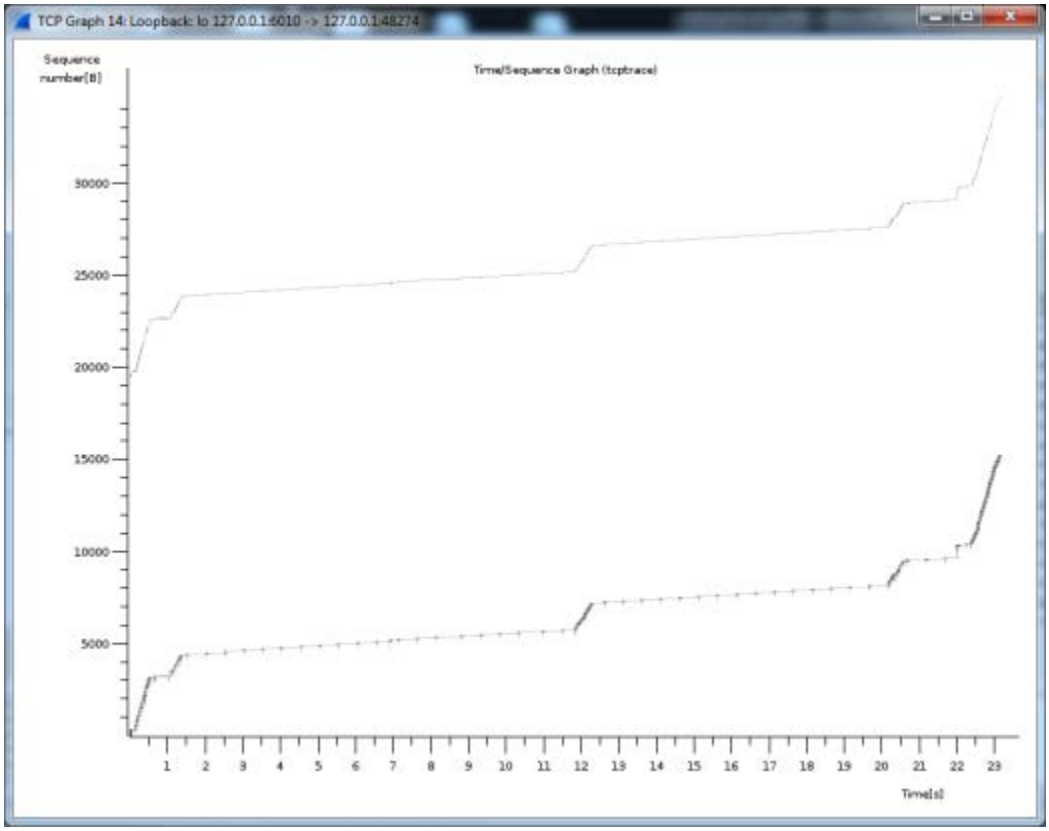
Γράφημα 5. 28 RTT όταν εκτελούμε ping & iperf

Παρατηρούμε ότι στα γραφήματα του wireshark δεν φαίνονται ξεκάθαρα οι επιπτώσεις στο δίκτυο της ταυτόχρονης εκτέλεσης των εντολών iperf & ping. Στην περίπτωση αυτή είναι πιο ξεκάθαρες οι μεταβολές που παρουσιάζονται στην κίνηση του δικτύου από τα αποτελέσματα που παίρνουμε από την εκτέλεση των εντολών μέσα στο παραθυρικό περιβάλλον.

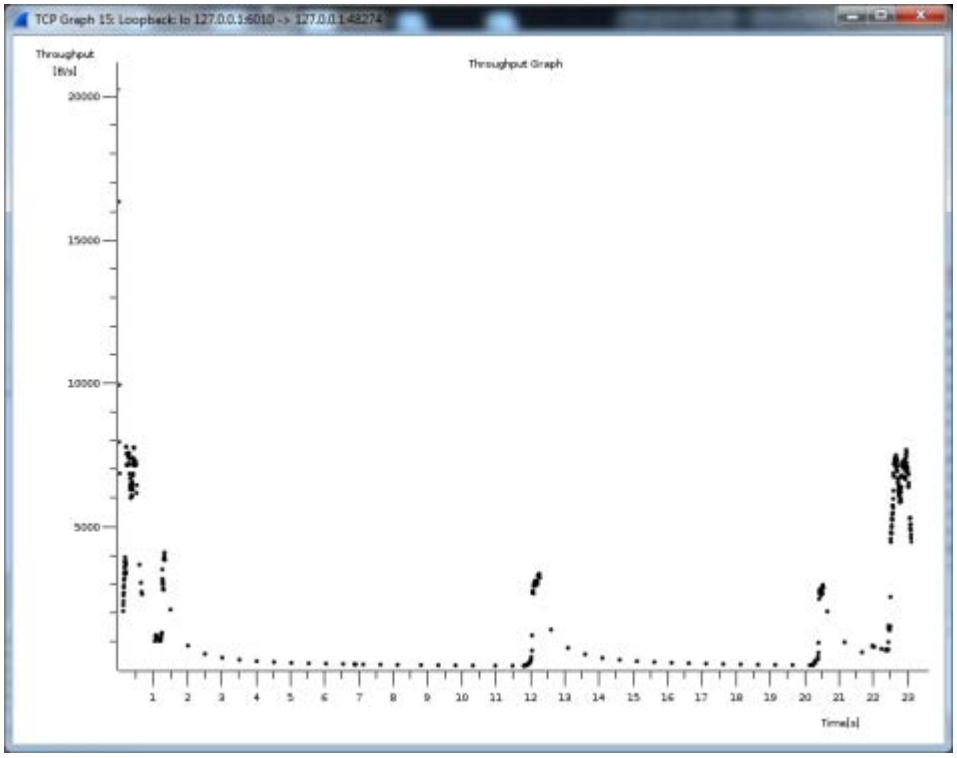
Στο παρακάτω παράθυρο φαίνονται τα δεδομένα που λαμβάνονται όταν δίνουμε την εντολή ping μόνη της χωρίς την εντολή iperf. Παρατηρούμε ότι οι χρόνοι ping είναι ικανοποιητικοί και δεν δείχνουν κάποιο πρόβλημα στην λειτουργία του δικτύου. Ακολουθούν και τα σχετικά γραφήματα τα οποία παράγαμε με την βοήθεια του προγράμματος Wireshark.



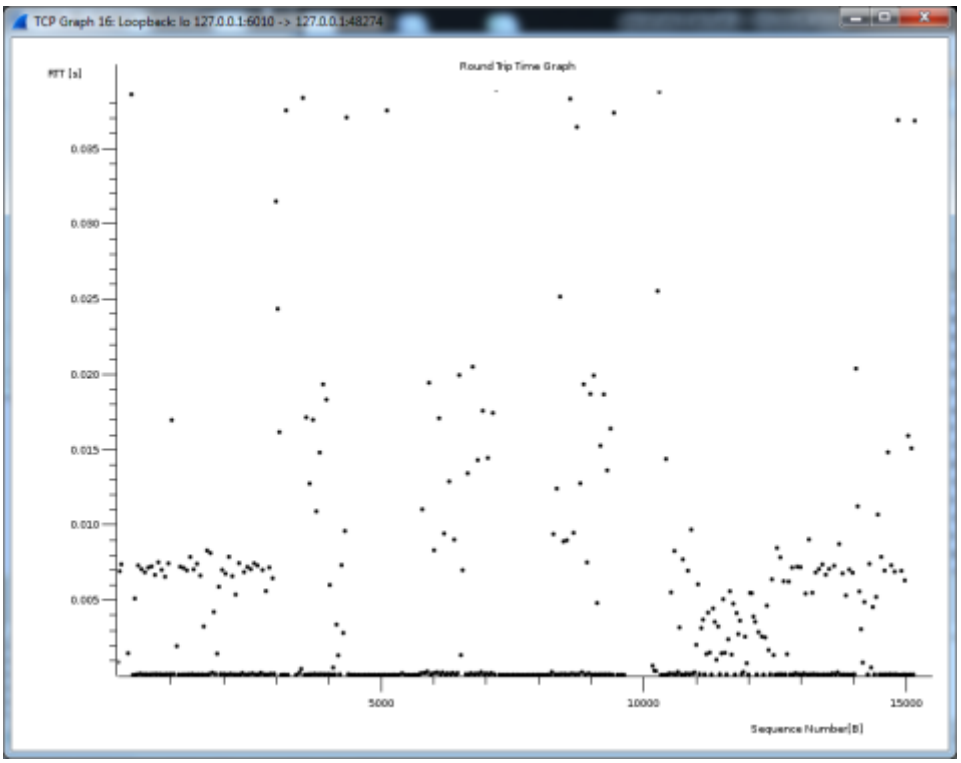
Εικόνα 5. 25 Αποτελέσματα ping εντολής στο δίκτυο.



Γράφημα 5. 29 Time Sequence όταν εκτελούμε μόνο την ping εντολή.

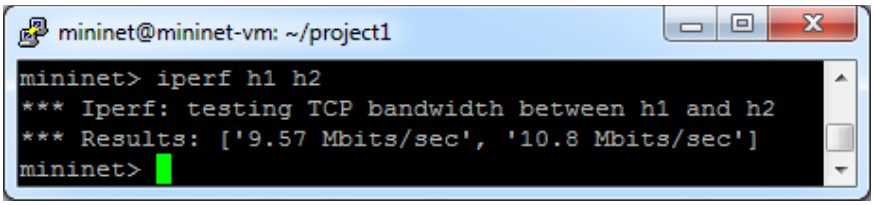


Γράφημα 5. 30 Throughput όταν εκτελώ την εντολή ping.

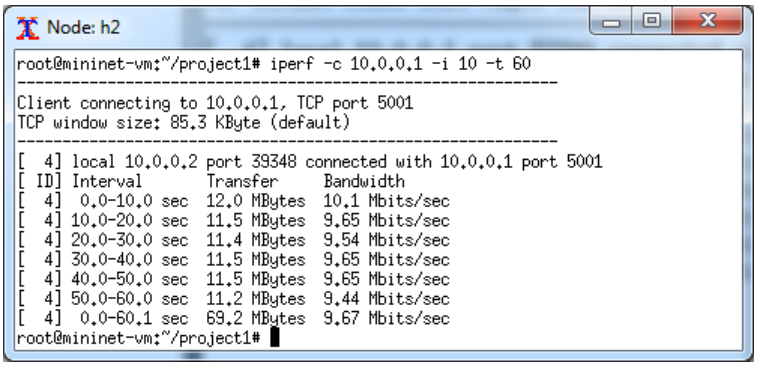


Γράφημα 5. 31 RTT χρόνοι κατά την εκτέλεση της εντολής ping

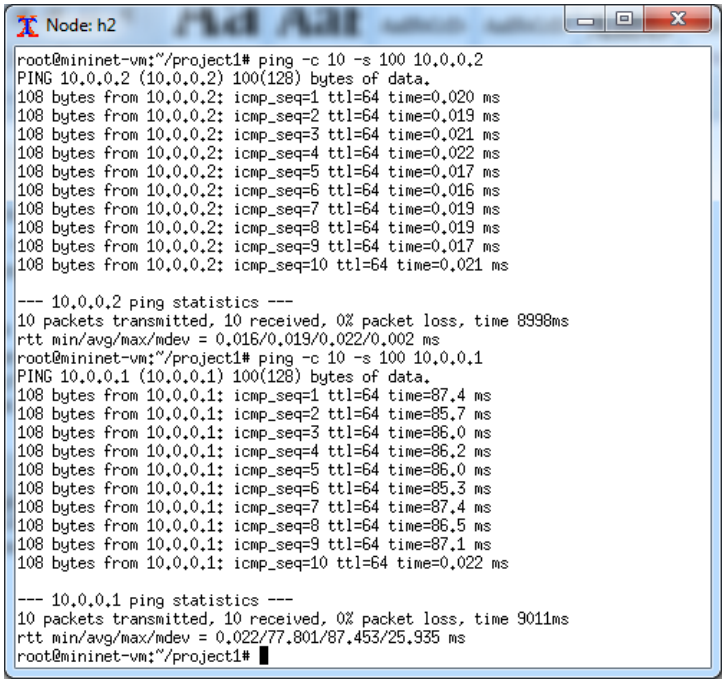
Εάν δώσουμε την εντολή mininet>**iperf h1 h2** στο cli παίρνουμε την παρακάτω εικόνα που μας εμφανίζει στο περιβάλλον CLI το TCP εύρος ζώνης της σύνδεσης μεταξύ των δύο κόμβων h1 και h2.



Εικόνα 5. 26 Αποτελέσματα iperf. Η εικόνα φανερώνει το BW μεταξύ των 2 κόμβων.



Εικόνα 5. 27 Στατιστικά για Iperf μαζί με ping

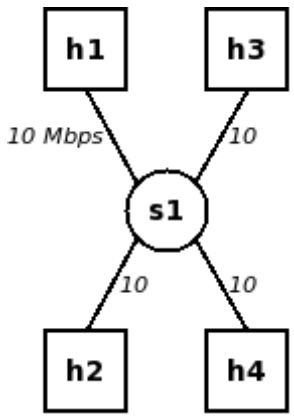


Εικόνα 5. 28 Αποτελέσματα ping χωρίς iperf (α μέρος) και με iperf (β μέρος).

Στην παραπάνω εικόνα 5.28 παρατηρούμε ευδιάκριτα ότι όταν κάνουμε ping χωρίς να τρέχει η εντολή iperf έχουμε τελείως διαφορετικούς χρόνους απόκρισης. Έτσι ενώ στην πρώτη περίπτωση οι χρόνοι είναι εξαιρετικά ικανοποιητικοί για όλες τις πιθανές εφαρμογές που μπορεί να τρέχουν σε έναν εξυπηρετητή, στην δεύτερη περίπτωση εφαρμογές όπως live video streaming ή on line gaming εμφανίζεται ότι ενδεχομένως θα εμφάνιζαν πρόβλημα ιδίως με την προσθήκη επιπλέον κίνησης στο δίκτυο μας, σε τέτοια περίπτωση μπορεί να είχαμε χρόνους ping μεγαλύτερους από 100ms. Αν πάλι οι εφαρμογές που τρέχουν είναι ελαφριές όπως επεξεργαστές αρχείων ή websites με σταθερές εικόνες τότε ακόμα και χρόνος ping γύρω στα 500 msec είναι αρκετός για να λειτουργούν ομαλά. Καταλαβαίνουμε λοιπόν ότι μπορούμε με την βοήθεια του Mininet να διαπιστώσουμε την απόδοση των δικτύων που επιθυμούμε να δημιουργήσουμε χωρίς κόστος εξετάζοντας την υλοποίηση νέων τεχνολογιών που δεν έχουν εφαρμοστεί ευρέως. Εδώ να σημειώσουμε ότι στοιχεία για την εντολή iperf καθώς και για την παραμετροποίηση αυτής μπορεί να βρει στο διαδίκτυο.

5.4.2 Δίκτυο Ενός Μεταγωγέα και Τεσσάρων Κόμβων.

Στο παρακάτω πείραμα δημιουργούμε ένα απλό δίκτυο με ένα μεταγωγέα και τέσσερις κόμβους με συνδέσεις 10Mbps μεταξύ τους. Η δομή του δικτύου γίνεται φανερή στο παρακάτω σχήμα:

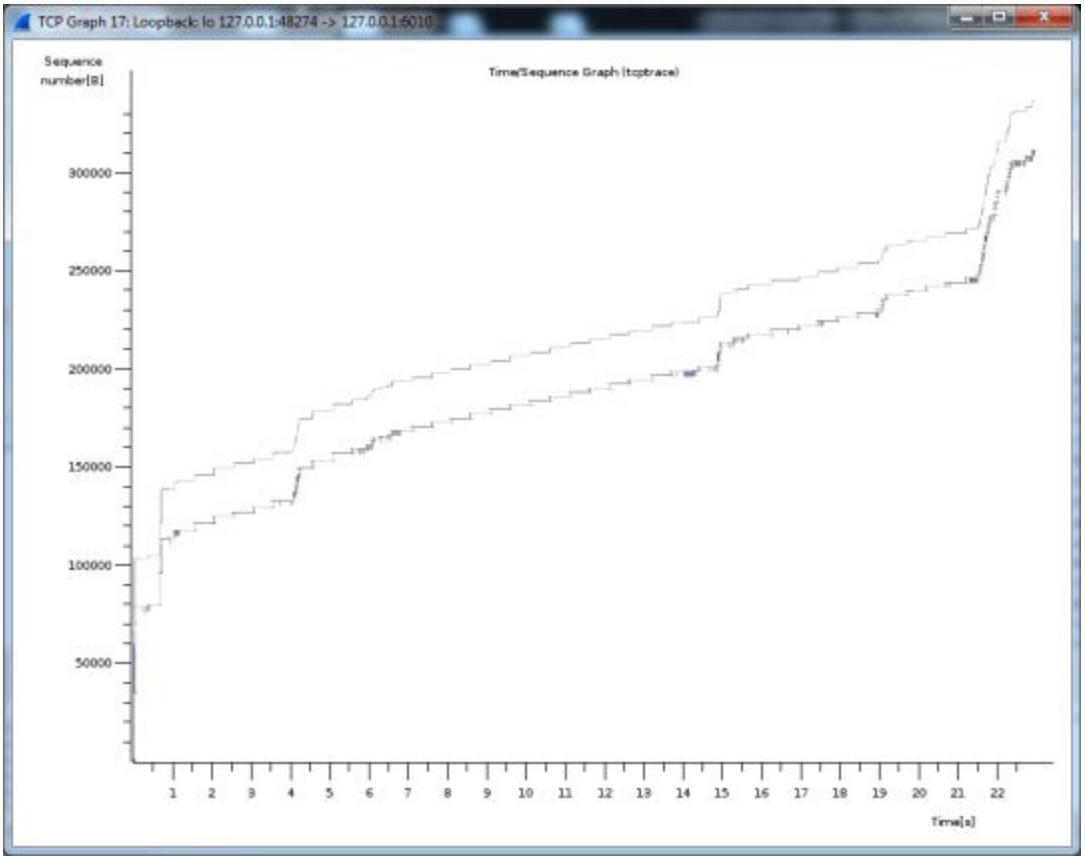


Εικόνα 5. 29 Δίκτυο ενός μεταγωγέα συνδεδεμένων με 4 κόμβους. [37]

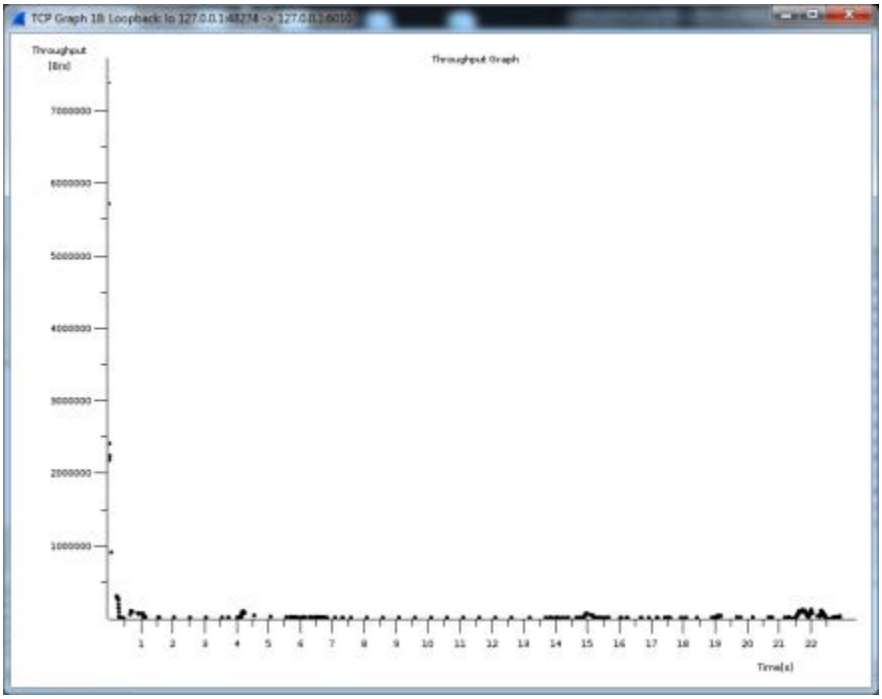
Για να δημιουργήσουμε το παραπάνω δίκτυο τρέχουμε το script με κώδικα Python project1_switch.py , αφού βρεθούμε στον κατάλογο project1 όπως έχουμε αναφέρει σε προηγούμενο παράδειγμα χρησιμοποιώντας sudo. Η εντολή έχει ως εξής:

$ sudo python project1_switch.py

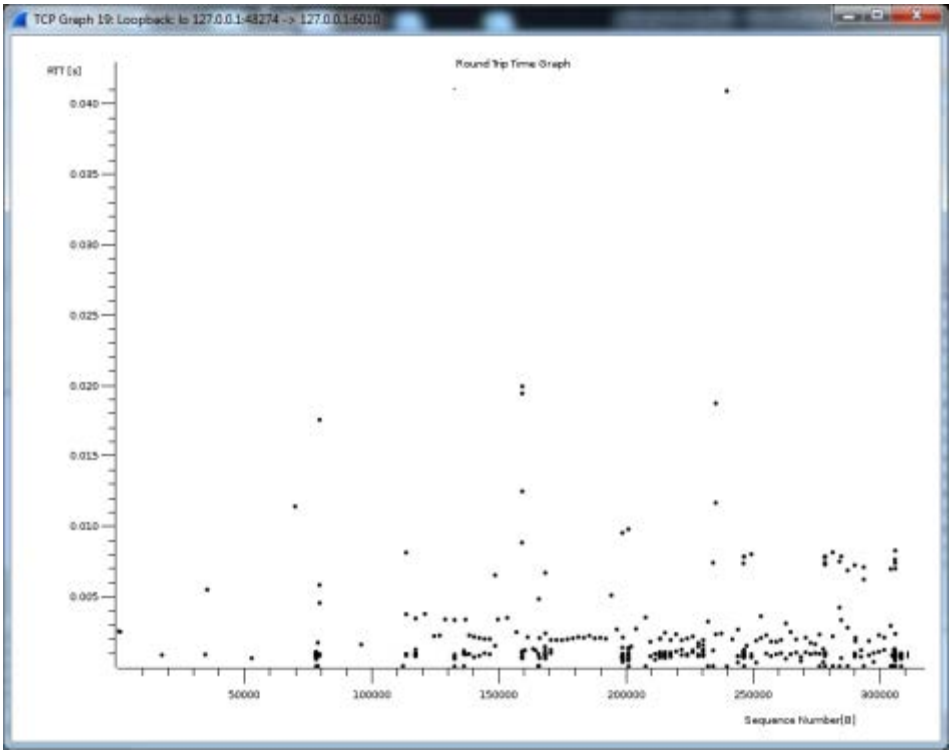
Παρατηρούμε ότι με την εντολή μας το Mininet δημιούργησε το δίκτυο, πρόσθεσε τον μεταγωγέα και τον προκαθορισμένο ελεγκτή ενώ όρισε ότι κάθε σύνδεση θα έχει εύρος ζώνης 10 Mbps. Τα αποτελέσματα που παίρνουμε με την βοήθεια του Wireshark είναι τα ακόλουθα:



Γράφημα 5. 32 Time Sequence κατά την διάρκεια δημιουργίας του δικτύου.

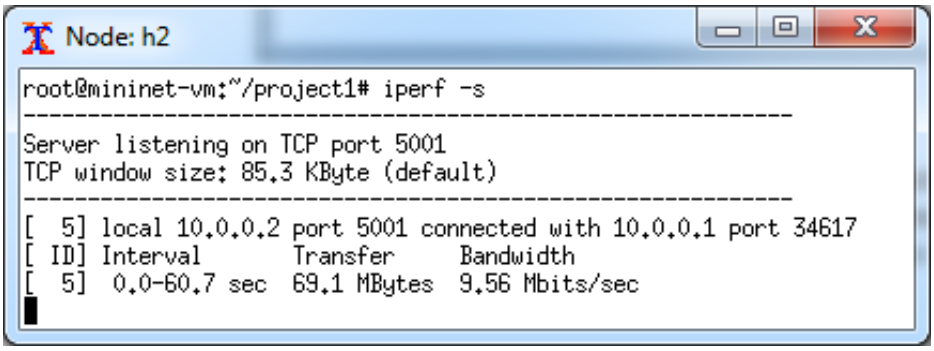


Γράφημα 5. 33 Throughput κατά την διάρκεια δημιουργίας του δικτύου.

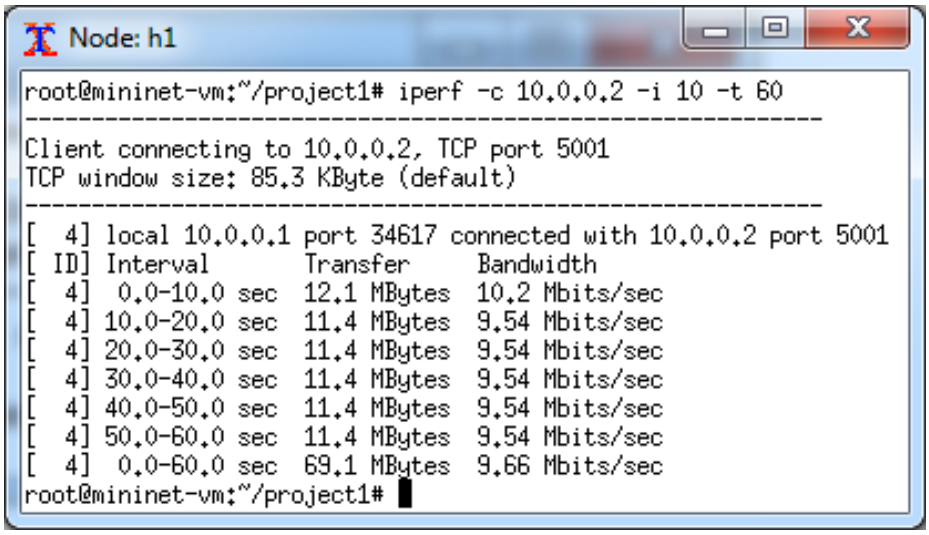


Γράφημα 5. 34 RTT χρόνοι κατά την δημιουργία του δικτύου.

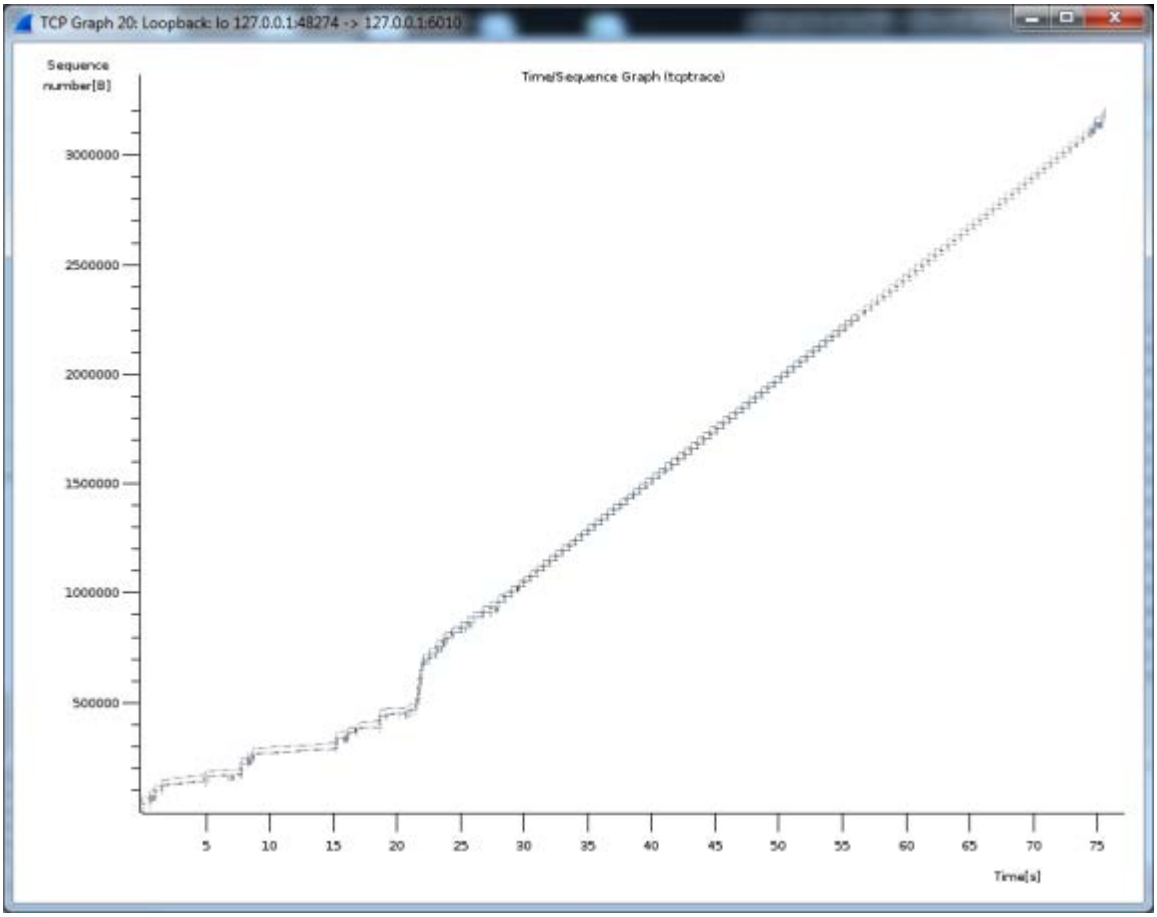
Σε αυτό το πείραμα δίνουμε την εντολή iperf για εξήντα δευτερόλεπτα εξάγοντας το εύρος ζώνης κάθε δέκα δευτερόλεπτα. Ο κόμβος h1 είναι ο πελάτης ενώ ο κόμβος h2 είναι ο εξυπηρετητής. Παρακάτω εμφανίζονται οι εικόνες των xterm καθώς και οι εντολές που δόθηκαν στους αντίστοιχους κόμβους.



Εικόνα 5. 30 Ορισμός του h2 κόμβου ως εξυπηρετητή.

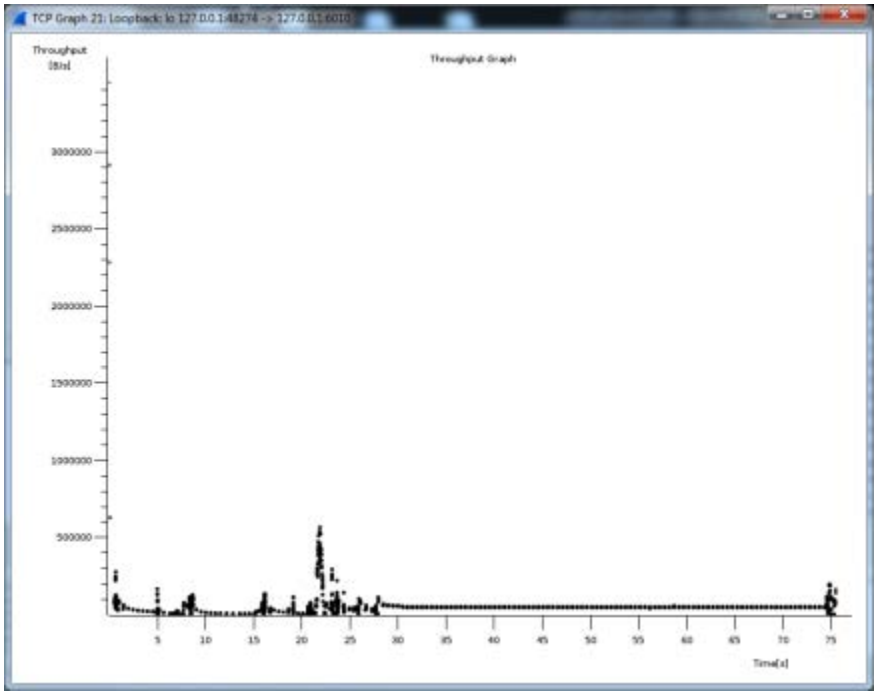


Εικόνα 5. 31 Iperf αποτελέσματα στο δίκτυο με 1 μεταγωγέα και 4 κόμβους.

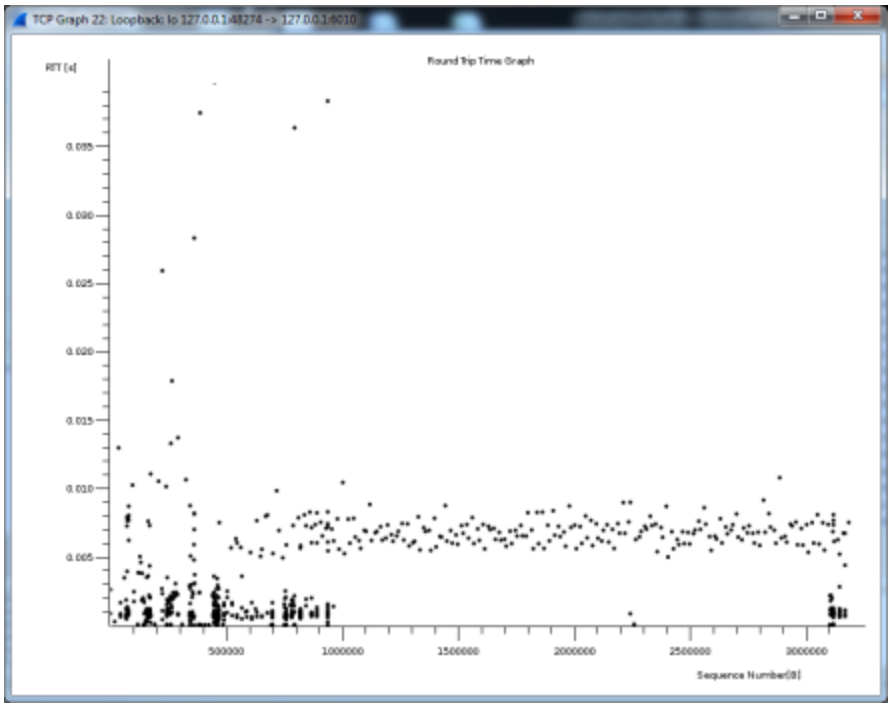


Γράφημα 5. 35 Time Sequence στο δίκτυο για iperf εξήντα δευτερόλεπτων.

Στο παραπάνω γράφημα Time Sequence παρατηρούμε τις αυξομειώσεις στην αρχή του λόγω της εγκαθίδρυσης του δικτύου. Μετά από τα 20 δευτερόλεπτα παρατηρούμε ένα φυσιολογικό διάγραμμα Time Sequence που δείχνει μια αυξανόμενη απόδοση στο δίκτυο. Η γραμμή αυτή δημιουργήθηκε κατά την εκτέλεση της εντολής iperf.



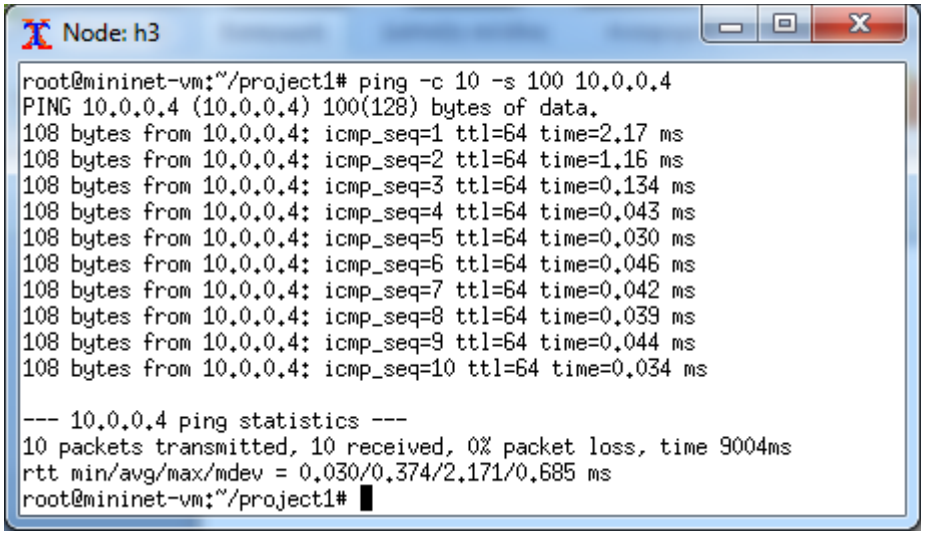
Γράφημα 5. 36 Throughput στο οποίο φαίνεται η ροή της κίνησης κατά την διάρκειατου iperf .



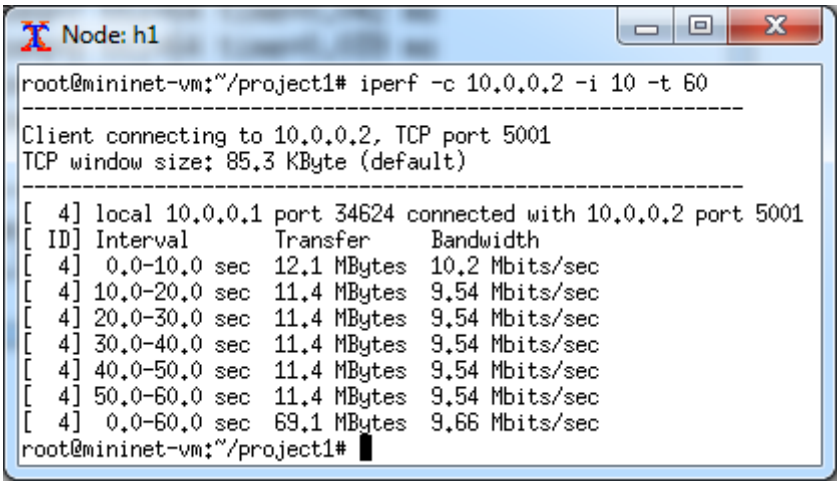
Γράφημα 5. 37 RTT χρόνοι στο δίκτυο κατά την διάρκεια του iperf.

Παρακάτω επαναλαμβάνουμε το iperf από τον h1 ενώ ταυτόχρονα στέλνουμε κίνηση και με την εντολή ping από τον h3 στον h4.

Η εντολή που δίνουμε είναι: # **ping -c 10 -s 100 10.0.0.4** από τον h3 στον h4.

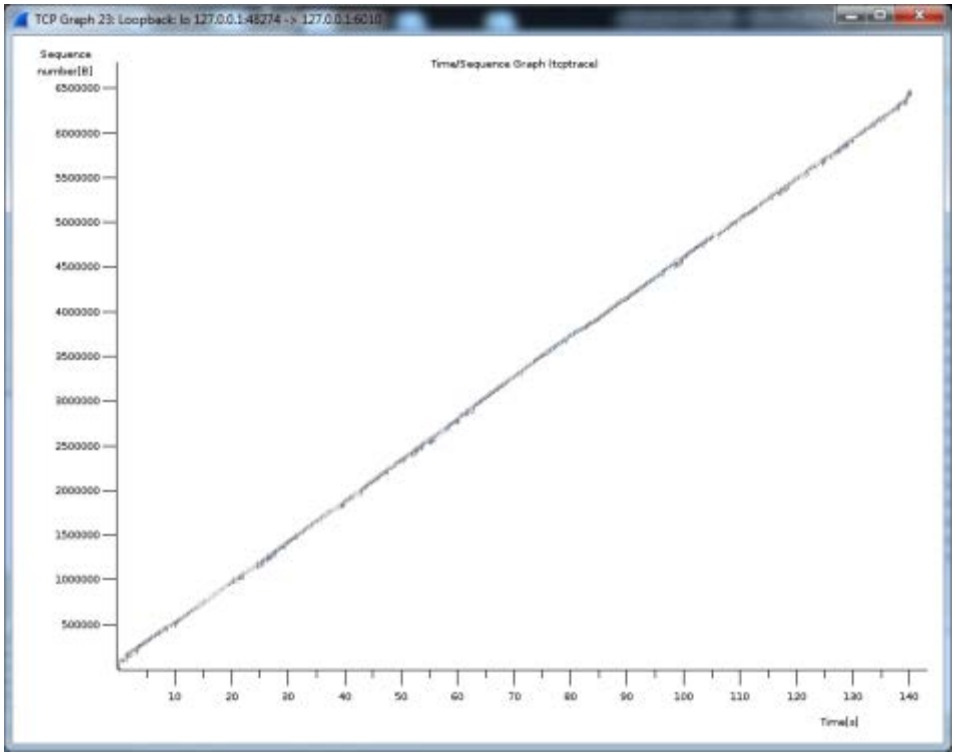


Εικόνα 5. 32 Ping από τον h3 στον h4 μαζί με iperf σε ένα ζεύγος κόμβων.

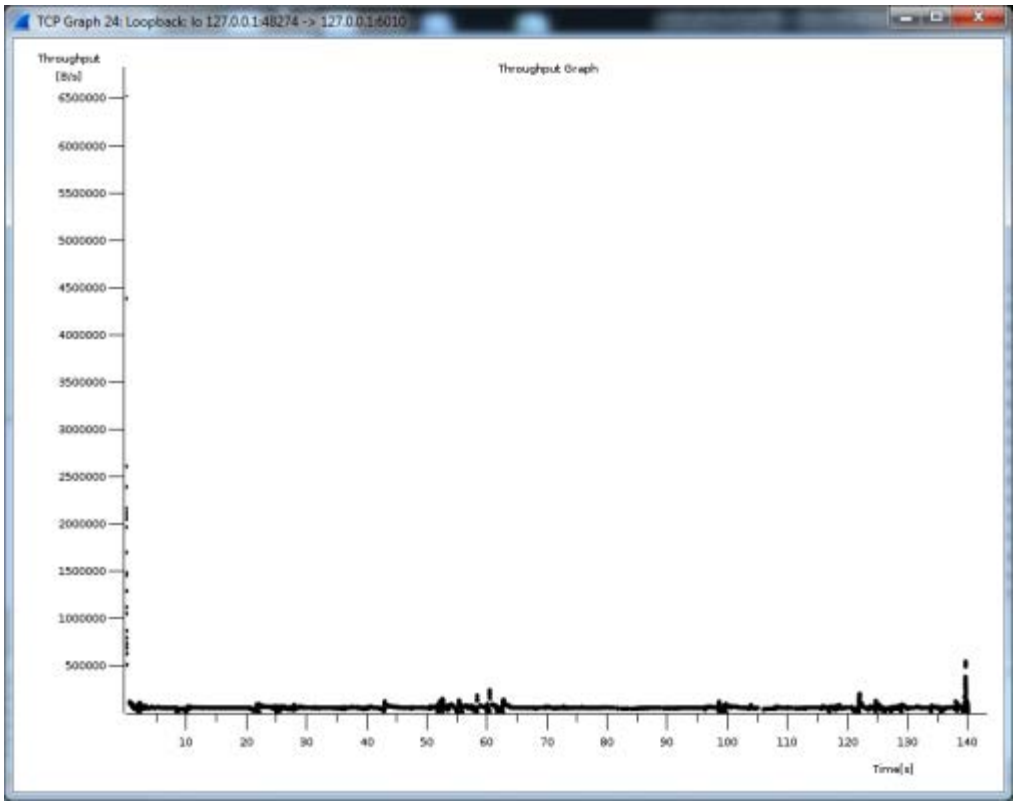


Εικόνα 5. 33 Iperf αποτελέσματα μαζί με ping εντολή από τον h3 στον h4.

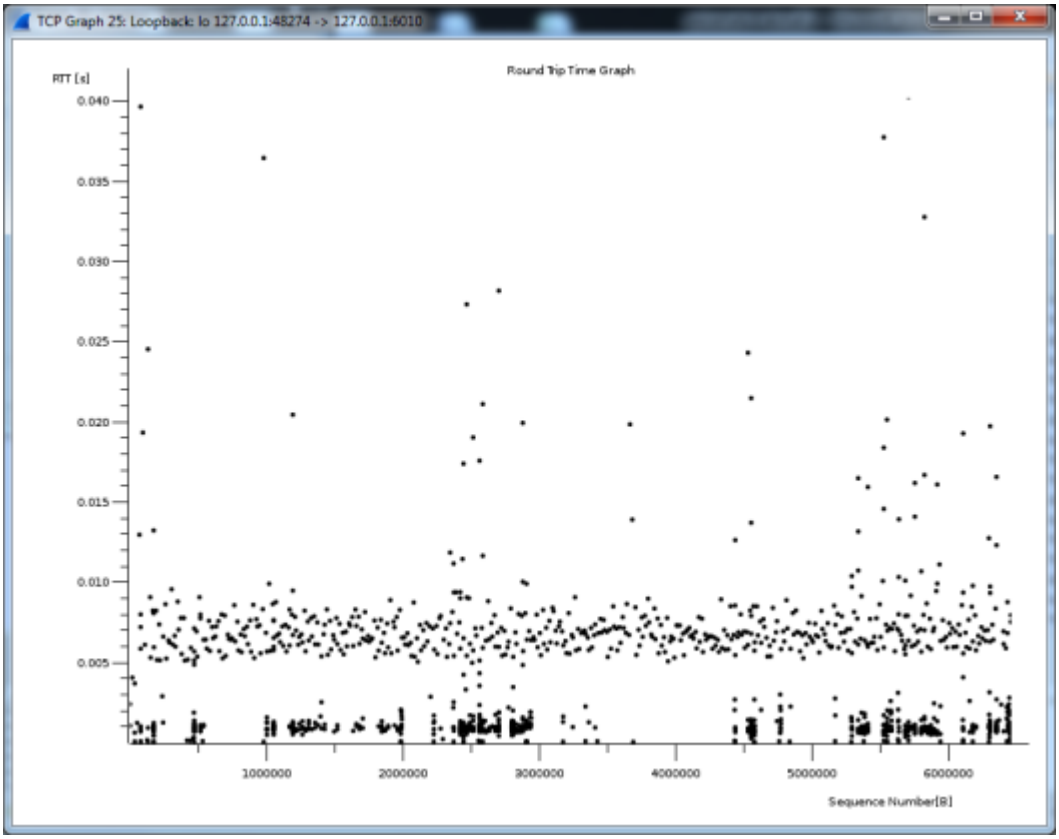
Τα γραφήματα που συλλέχτηκαν κατά την διάρκεια του πειράματος είναι τα παρακάτω:



Γράφημα 5. 38 Time Sequence ενώ υπάρχει ένα ζεύγος με iperf κίνηση και ένα που εκτελεί ping.

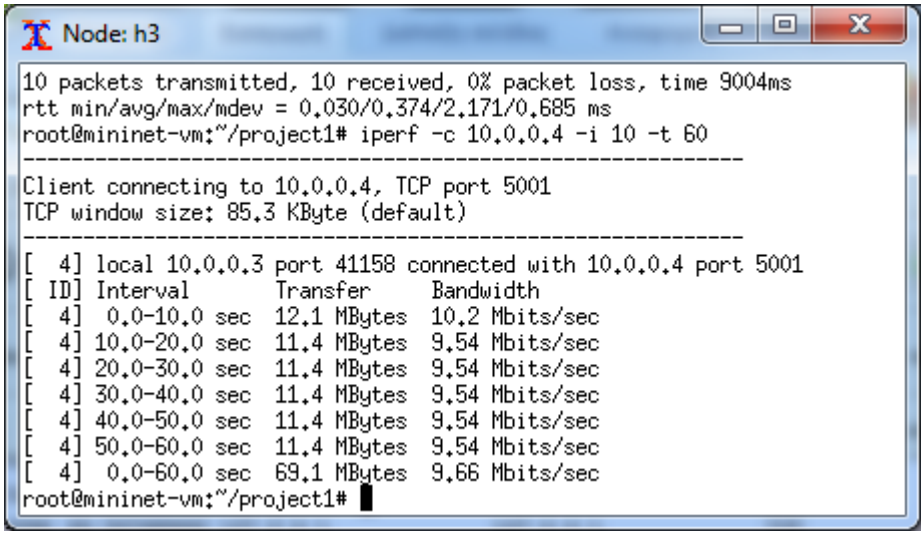


Γράφημα 5. 39 Throughput ενώ υπάρχει ένα ζεύγος με iperf κίνηση και ένα με ping.

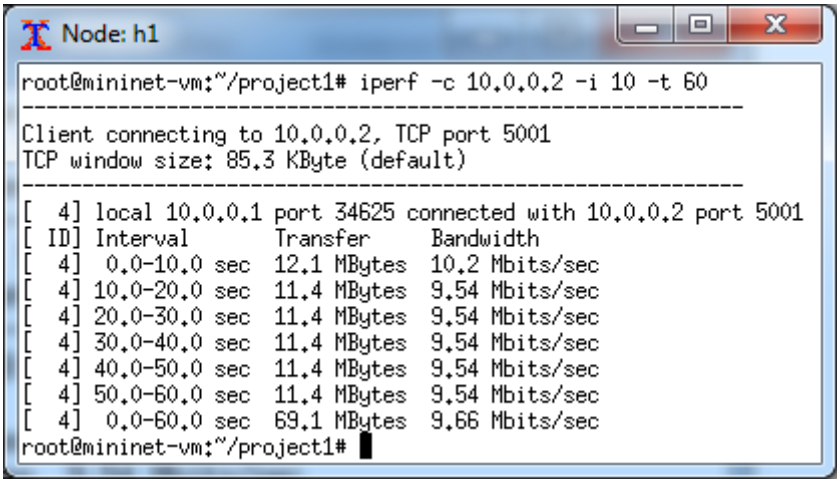


Γράφημα 5. 40 RTT χρόνοι ενώ υπάρχει ένα ζεύγος με iperf κίνηση και ένα με ping.

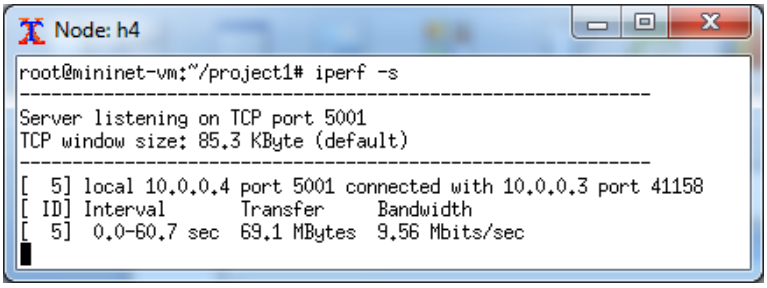
Μια παραλλαγή του παραπάνω πειράματος που θα μας βοηθήσει να δούμε τις μεταβολές στο δίκτυο στην περίπτωση που έχουμε αύξηση της κίνησης σε αυτό είναι να προσπαθήσουμε σχεδόν ταυτόχρονα να τρέξουμε την iperf εντολή μεταξύ δύο ζευγών κόμβων. Οι κόμβοι h1 και h3 είναι πελάτες ενώ οι h2 και h4 είναι εξυπηρετητές. Ο h1 χρησιμοποιεί τον h2 ως server και ο h3 χρησιμοποιεί τον h4 ως server. Όπως στα προηγούμενα πειράματα τρέχουμε την εντολή iperf για 60 δευτερόλεπτα και το εύρος ζώνης παρουσιάζεται κάθε 10 δευτερόλεπτα. Παρακάτω εμφανίζονται τα αποτελέσματα του πειράματος μέσα από τα xterm παράθυρα.



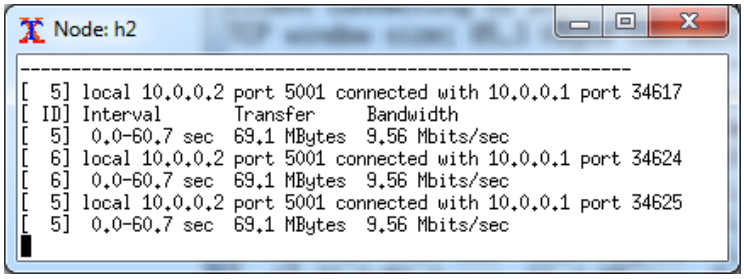
Εικόνα 5. 34 Αποτελέσματα Iperf για 2 ζεύγη κόμβων στο δίκτυο.



Εικόνα 5. 35 Αποτελέσματα Iperf για 2 ζεύγη κόμβων στο δίκτυο.

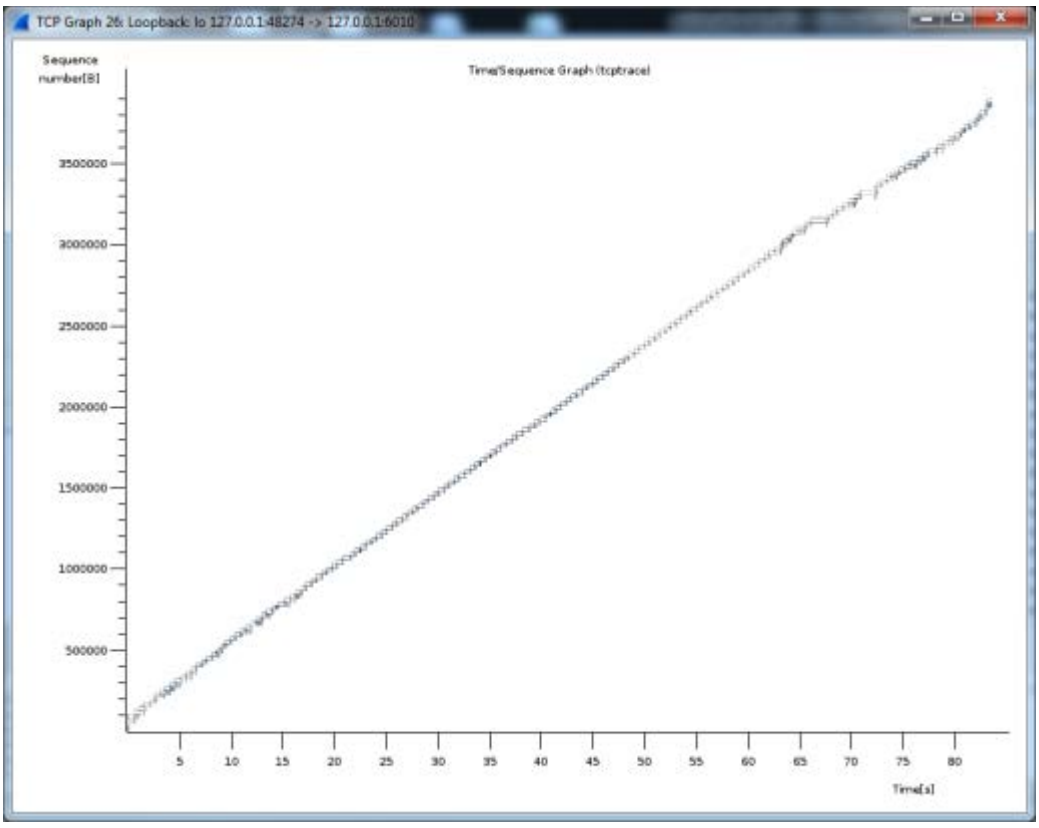


Εικόνα 5. 36 Ορισμός του h4 ως εξυπηρετητή.

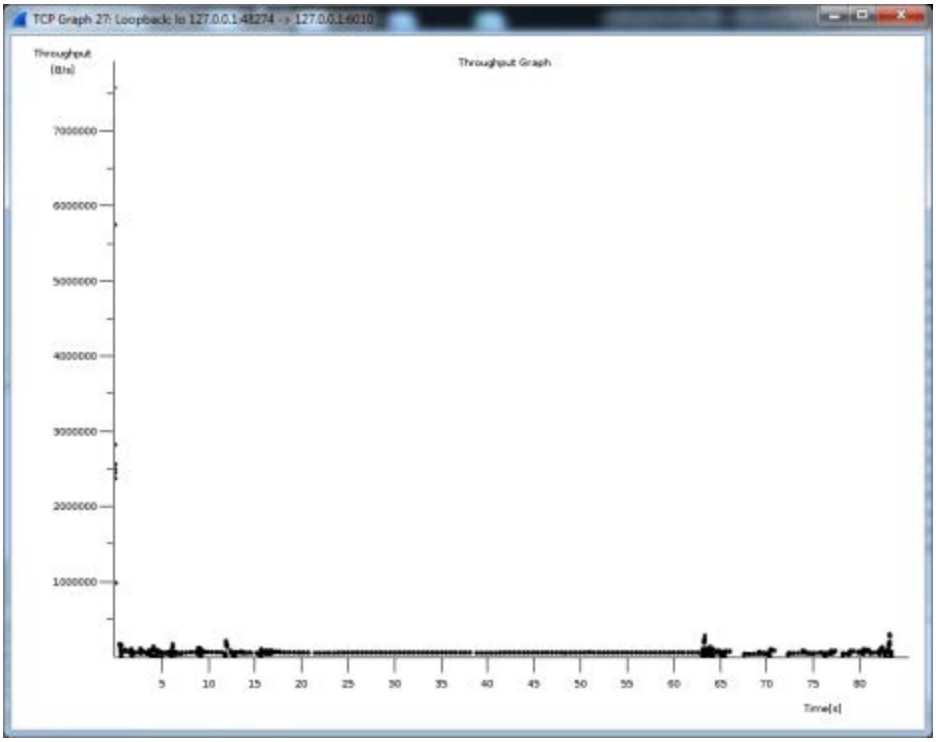


Εικόνα 5. 37 Ορισμός του h2 ως εξυπηρετητή.

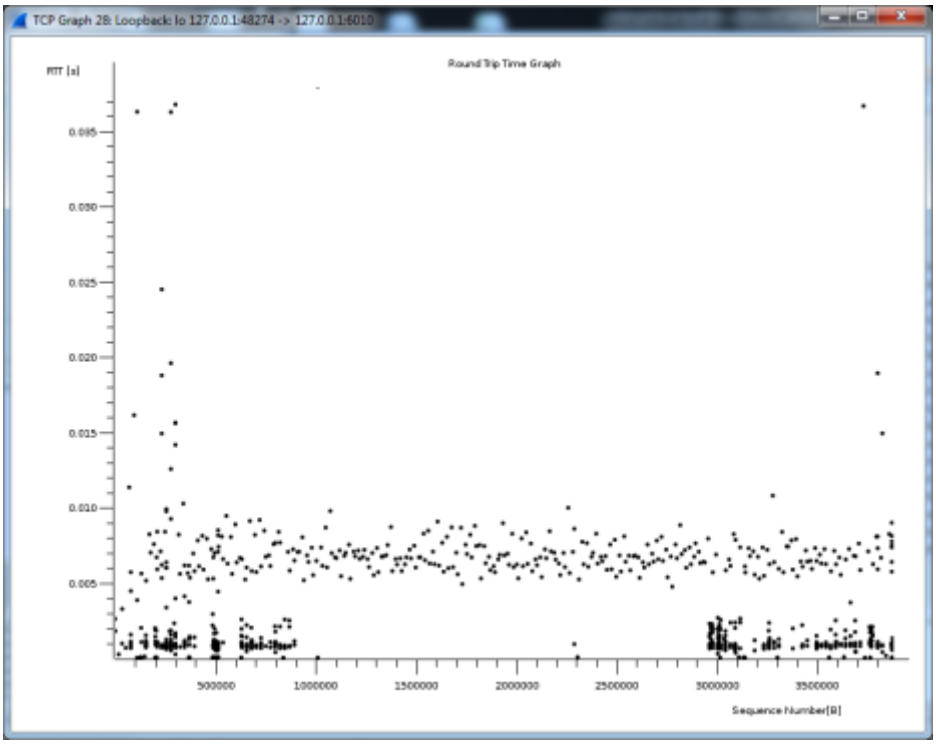
Οι γραφικές παραστάσεις που καταγράφηκαν και δίνουν τα κύρια χαρακτηριστικά του δικτύου ακολουθούν:



Γράφημα 5. 41 Time Sequence όταν 2 ζεύγη κόμβων εκτελούν την εντολή iperf.

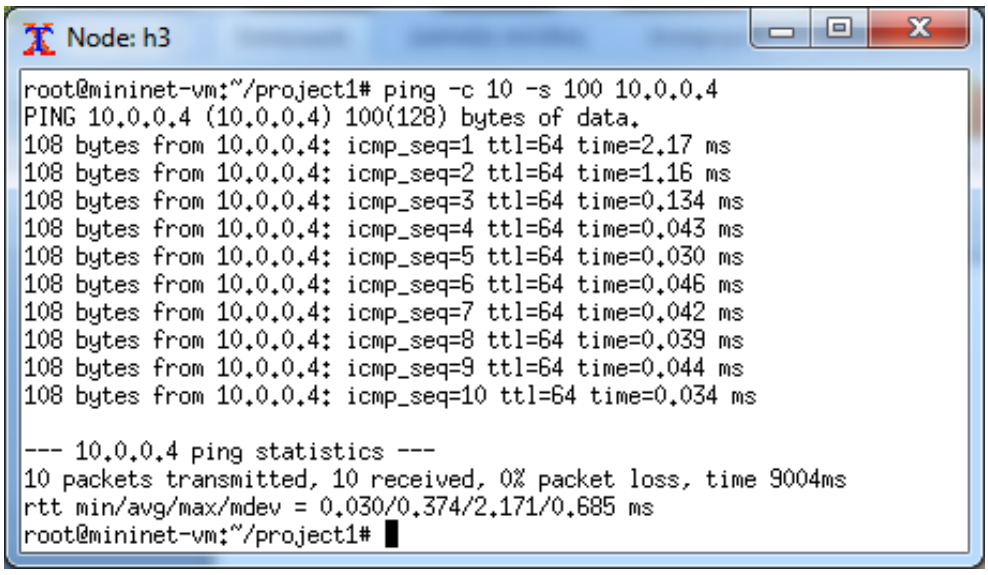


Γράφημα 5. 42 Throughtput όταν 2 ζεύγη κόμβων εκτελούν την εντολή iperf.



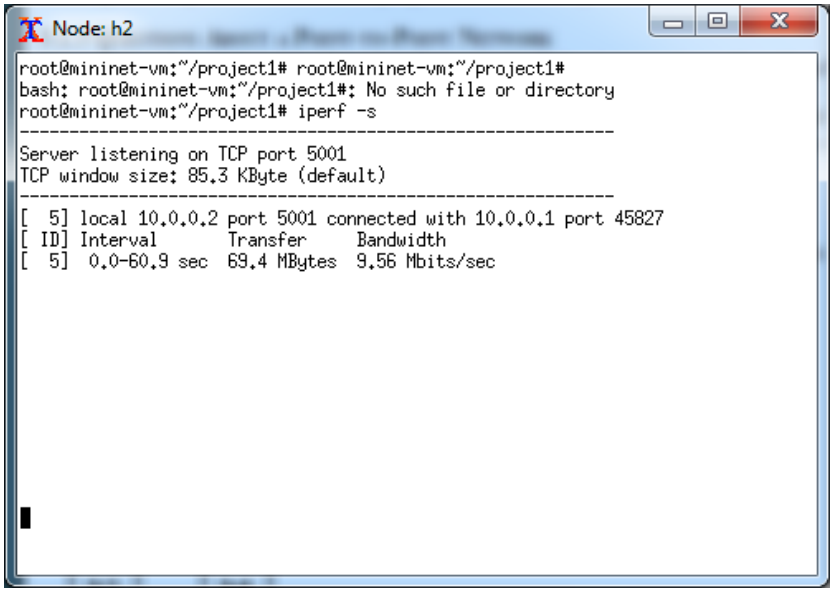
Γράφημα 5. 43 RTT χρόνοι όταν 2 ζεύγη κόμβων που εκτελούν την εντολή iperf.

Εδώ στην παρακάτω εικόνα βλέπουμε τα συλληφθέντα στοιχεία με το iperf και το ping στην περίπτωση του δικτύου με ένα μεταγωγέα και τέσσερις κόμβους από τους οποίους ανά δυο επικοινωνούν ως ζεύγος δηλαδή ως εξυπηρετητής και ως πελάτης. Αν συγκρίνουμε τις δυο αυτές εικόνες τα στοιχεία που έχουμε παρατηρούμε ότι ο ελάχιστος RTT χρόνος αυξήθηκε, το ίδιο και ο μέσος χρόνος RTT, ο μέγιστος χρόνος RTT αυξήθηκε επίσης πολύ και ουσιαστικά πολλαπλασιάστηκε επί ενός παράγοντα κοντά στο 100, ενώ η απόκλιση πενταπλασιάστηκε. Επιπροσθέτως υπάρχει μια μικρή απόκλιση του εύρους ζώνης της σύνδεσης εξ ' ορισμού με την μέτρηση του εύρους ζώνης που δίνει η εντολή iperf. Παρατηρούμε μια διαφορά κατά 0,5 Mbits/sec.



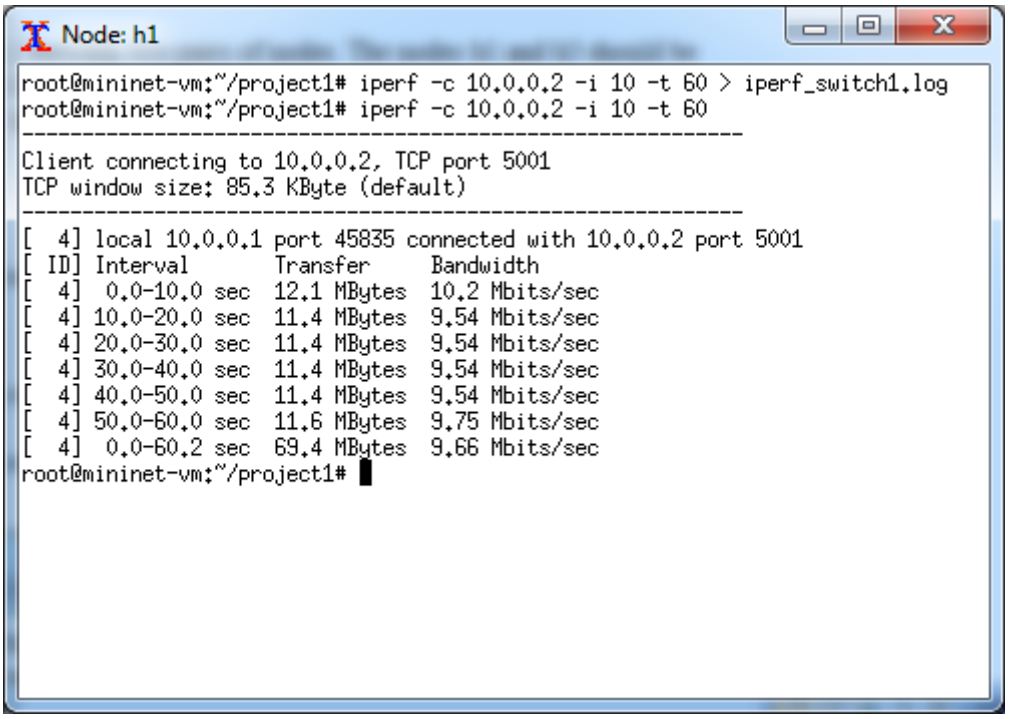
Εικόνα 5. 38 Ping αποτελέσματα όταν 2 ζεύγη κόμβων που εκτελούν την εντολή iperf.

Παρακάτω επαναλαμβάνουμε το προηγούμενο πείραμα δηλαδή δίνουμε την εντολή **$ sudo python project1_switch.py** στην συνέχεια ανοίγουμε τα επιθυμητά xterm παράθυρα ώστε να δώσουμε απευθείας εντολές στους κόμβους. Ορίζουμε ως εξυπηρετητή τον κόμβο h2. Δίνουμε δηλαδή την εντολή **iperf -s** στον h2 όπως φανερώνεται στην παρακάτω εικόνα.



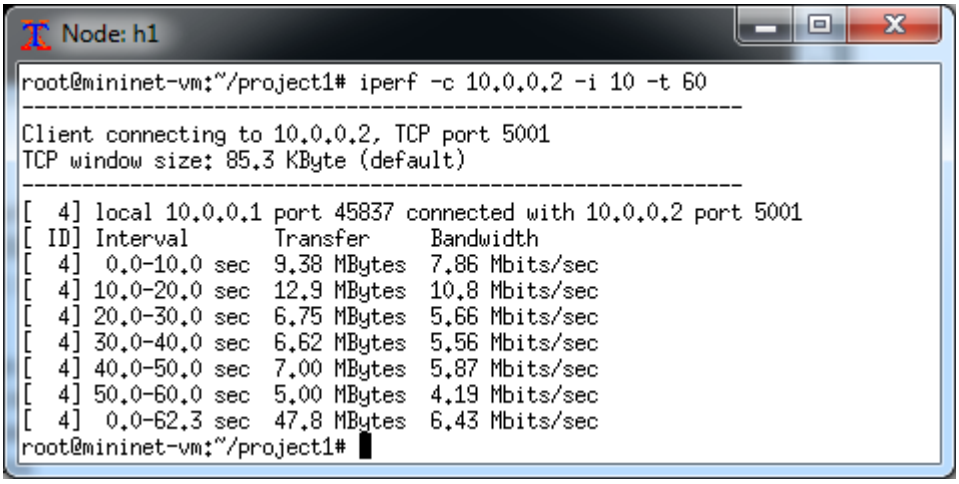
Εικόνα 5. 39 Ορισμός του h2 ως εξυπηρετητή.

Στην συνέχεια πραγματοποιούμε iperf μεταξύ του κόμβου h1 και h2 με την εντολή **iperf -c 10.0.0.2 -i 10 -t 60** που δίνουμε στον h1. Παρατηρούμε τα αποτελέσματα που παρουσιάζονται στην εικόνα που ακολουθεί:

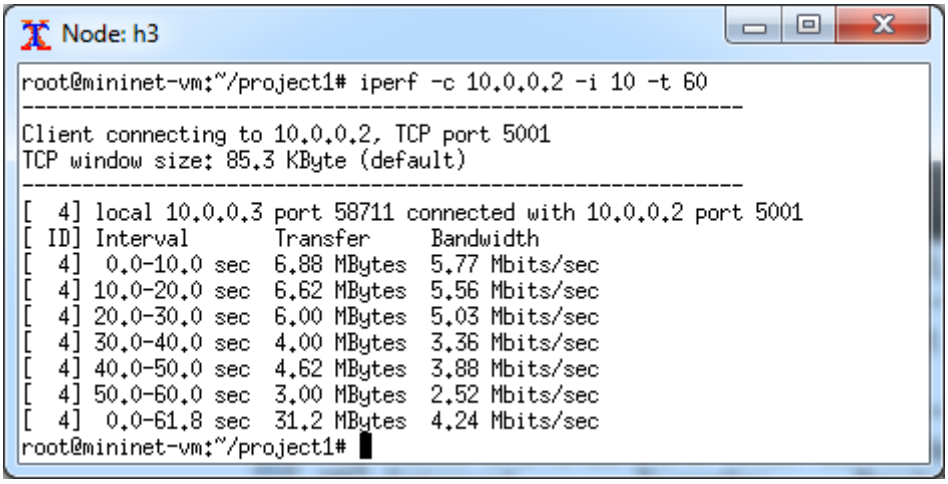


Εικόνα 5. 40 Iperf μεταξύ h1 & h2

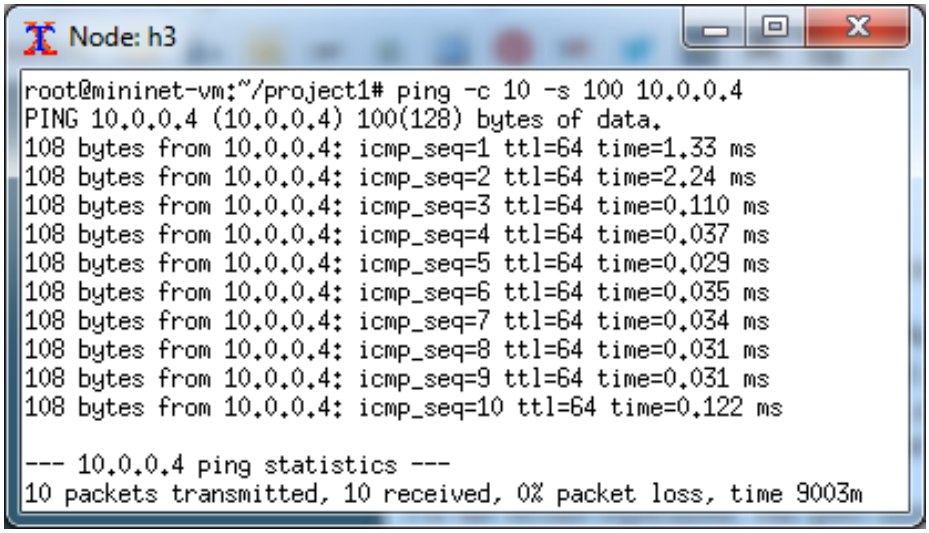
Εάν εκτός από την εντολή iperf μεταξύ του κόμβου h1 και h2 με την εντολή **iperf -c 10.0.0.2 -i 10 -t 60** που δίνουμε στον h1, δώσουμε και την εντολή **iperf -c 10.0.0.2 -i 10 -t 60** στον h3, μαζί με την εντολή **ping -c 10 -s 100 10.0.0.4** που δίνουμε στον h3 συλλέγουμε τα αποτελέσματα που φαίνονται στις τρείς παρακάτω εικόνες. Παρατηρούμε μέσω αυτών μεγάλες διαφορές στο εύρος ζώνης που δίνει η εντολή iperf καθώς και στον χρόνο RTT. Όσο λοιπόν περισσότεροι κόμβοι δημιουργούν κίνηση στο δίκτυο τόσο μειώνεται το διαθέσιμο εύρος ζώνης και αυξάνεται ο χρόνος RTT.



Εικόνα 5. 41 Αποτελέσματα Iperf όταν δυο κόμβοι δημιουργούν κίνηση προς στον h2 μέσω της εντολής iperf ενώ εκτελείται και ping στον h4.



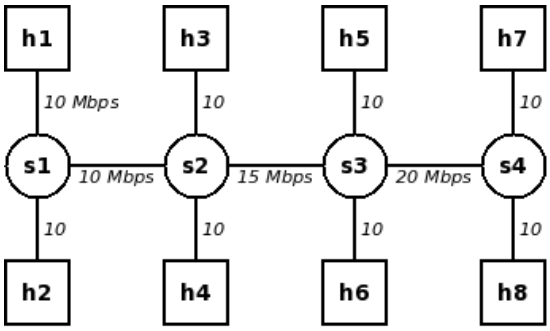
Εικόνα 5. 42 Αποτελέσματα Iperf όταν δυο κόμβοι δημιουργούν κίνηση προς στον h2 μέσω της εντολής iperf ενώ εκτελείται και ping στον h4.



Εικόνα 5. 43 Αποτελέσματα Ping όταν δυο κόμβοι δημιουργούν κίνηση προς στον h2 μέσω της εντολής iperf ενώ εκτελείται και ping στον h4.

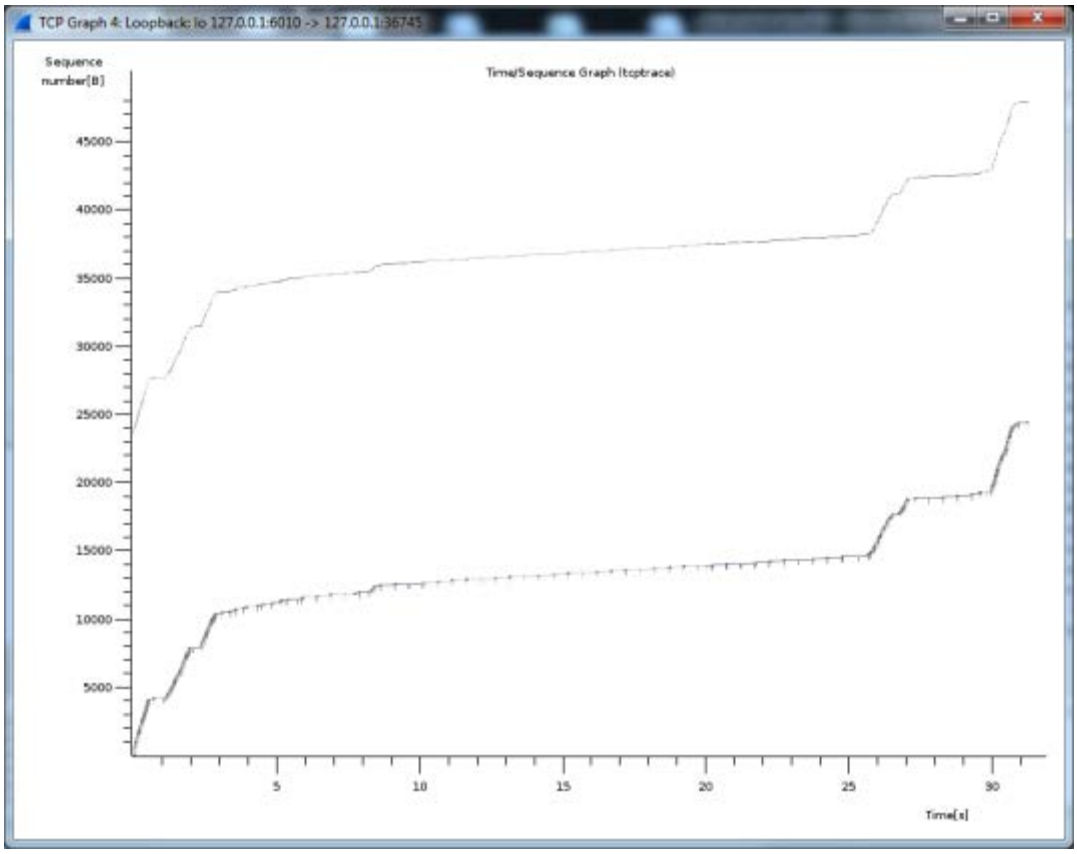
5.4.3 Δίκτυο με 4 Μεταγωγείς και 8 Κόμβους.

Στο παρακάτω πείραμα που θα μελετήσουμε το δίκτυο αποτελείται από τέσσερις μεταγωγείς και οχτώ κόμβους. Οι μεταγωγείς s1 και s2 είναι συνδεδεμένοι μεταξύ τους με 10 Mbps εύρος ζώνης σύνδεση, οι μεταγωγείς s2 και s3 είναι συνδεδεμένοι μεταξύ τους με 15 Mbps εύρος ζώνης σύνδεση και οι μεταγωγείς s3 και s4 είναι συνδεδεμένοι μεταξύ τους με 20Mbps εύρος ζώνης σύνδεση. Κάθε κόμβος συνδέεται με μεταγωγέα με σύνδεση 10Mbps αντίστοιχα. Έτσι οι κόμβοι h1, h2 συνδέονται με τον κόμβο s1, οι κόμβοι h3, h4 με τον s2, οι κόμβοι h5, h6 με τον s3 και οι h7, h8 με τον s4. Ακολουθεί σχηματική αναπαράσταση του δικτύου.



Εικόνα 5. 44 Δίκτυο με 4 μεταγωγείς και 8 κόμβους. [37]

Για να τρέξουμε στο Mininet το αρχείο με αυτήν την τοπολογία, τρέχουμε το Python script **project1_network.py** χρησιμοποιώντας την εντολή **sudo**. [37] Δίνουμε δηλαδή την εντολή $ sudo python project1_network.py αφού είμαστε στο φάκελο με τους κώδικες του project1.

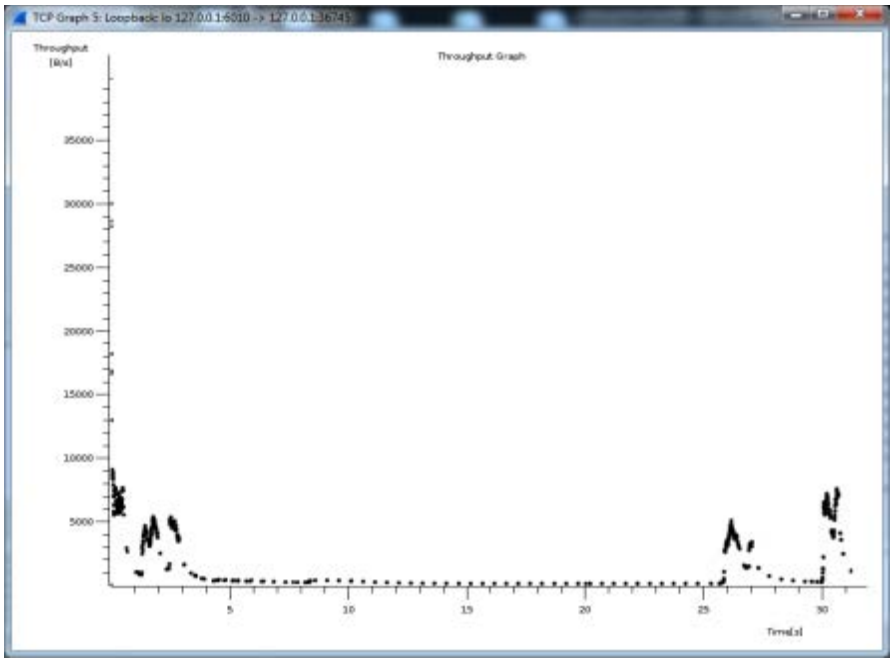


Γράφημα 5. 44 Time Sequence κατά την δημιουργία του δικτύου με τους 4 μεταγωγείς και τους 8 κόμβους.

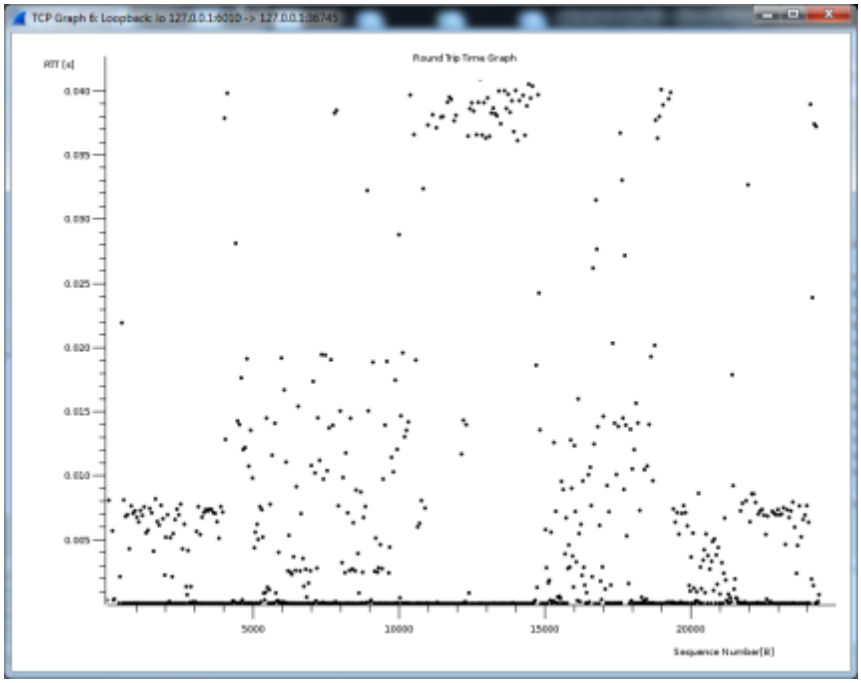
Στα γραφήματα 5.42-5.44 παρατηρούμε χαρακτηριστικά του δικτύου όπως το εύρος ζώνης, ο χρόνος μετ΄επιστροφής και η ακολουθία των πακέτων που ανταλλάσει το δίκτυο που δημιουργήσαμε με την εντολή:

$ sudo python project1_network.py

στο Mininet. Αποτελούν δηλαδή αποτελέσματα της εγκαθίδρυσης των συνδέσεων και των στοιχείων από τα οποία αποτελείται. Οι πληροφορίες συλλέχτηκαν με την βοήθεια του Wireshark.

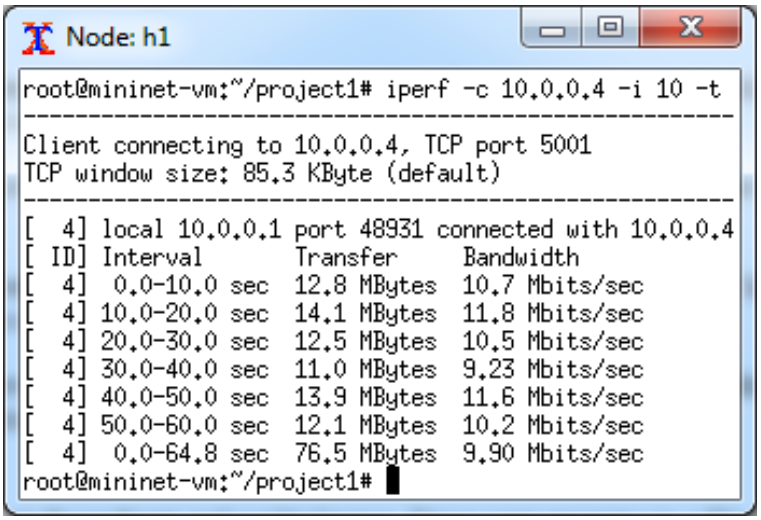


Γράφημα 5. 45 Throughput κατά την διάρκεια δημιουργίας του δικτύου των 4 μεταγωγέων και 8 κόμβων.

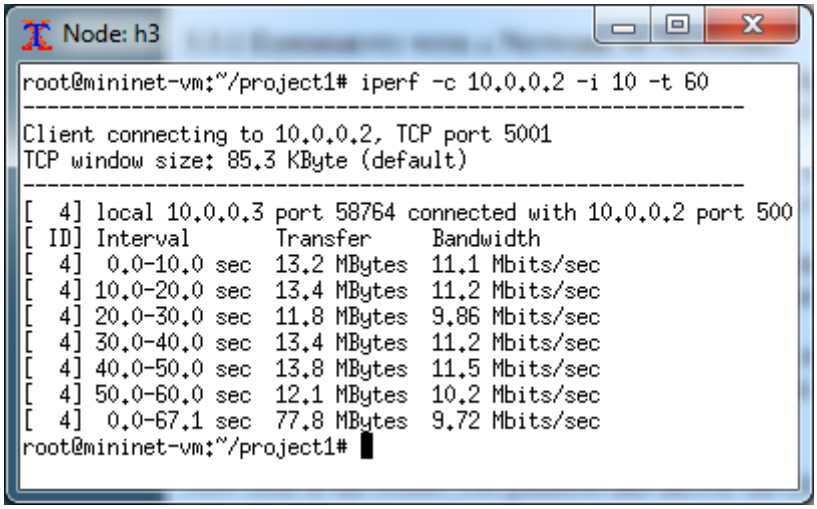


Γράφημα 5. 46 RTT κατά την διάρκεια δημιουργίας του δικτύου των 4 μεταγωγέων και 8 κόμβων.

Στο συγκεκριμένο πείραμα τρέχουμε την εντολή iperf για εξήντα δευτερόλεπτα εξάγοντας το εύρος ζώνης κάθε δέκα δευτερόλεπτα μεταξύ δυο ζευγών κόμβων ταυτόχρονα. Δημιουργούμε δηλαδή κίνηση μεταξύ 2 ζευγών κόμβων του δικτύου. Οι γραφικές παραστάσεις που εξάγονται κατά την διάρκεια του πειράματος ακολουθούν.

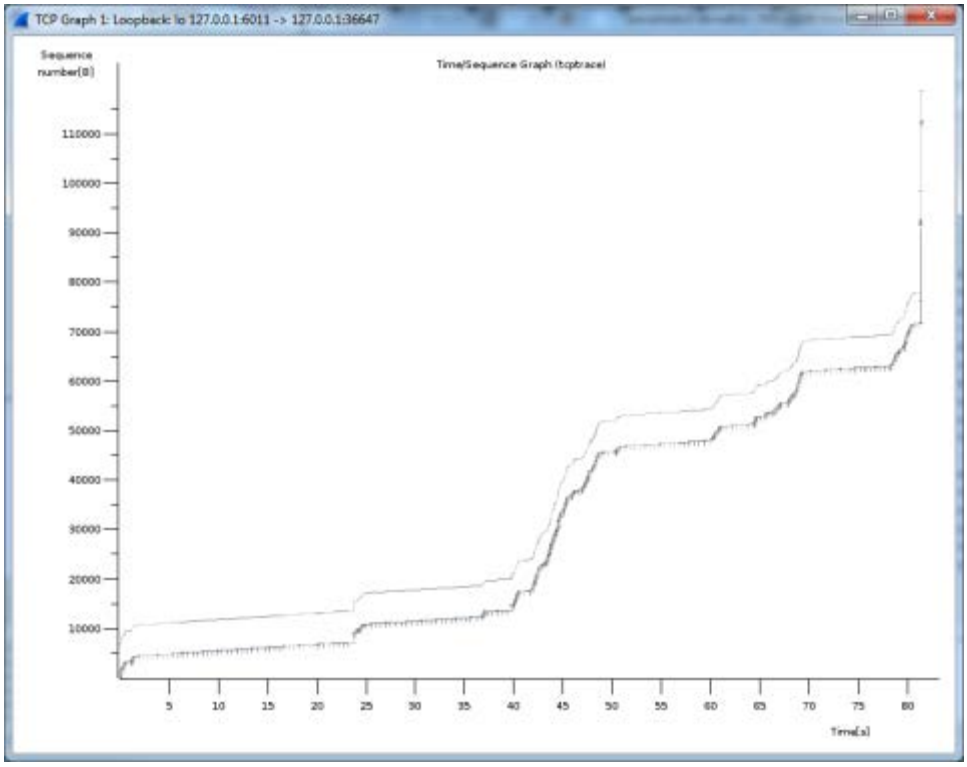


Εικόνα 5. 45 Iperf μεταξύ h1 & h4 κόμβου.

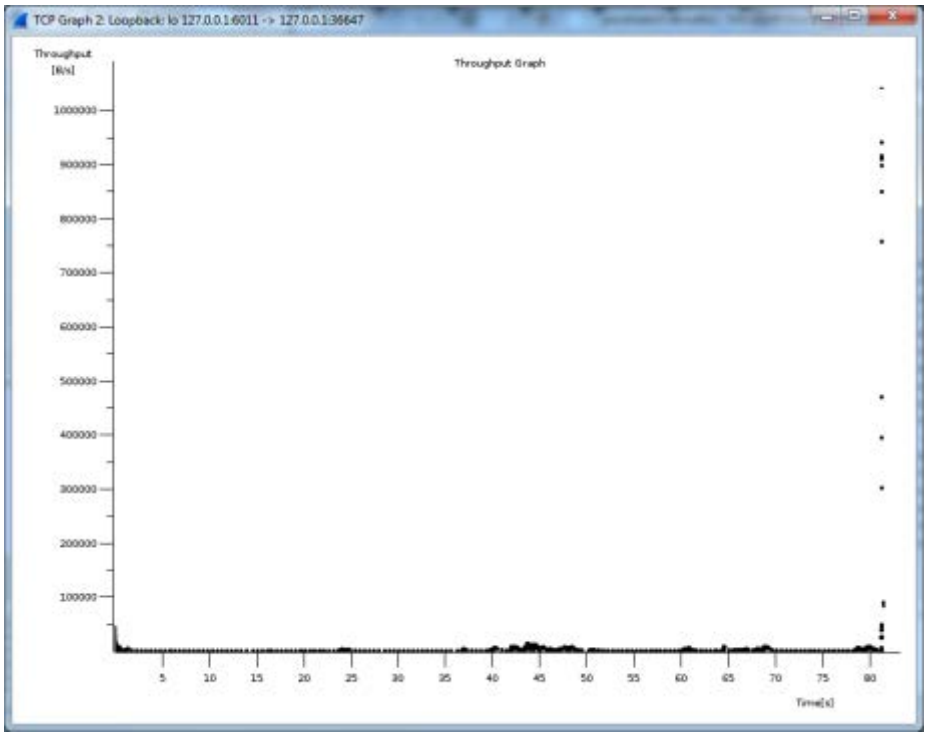


Εικόνα 5. 46 Iperf μεταξύ h3 & h2 κόμβου.

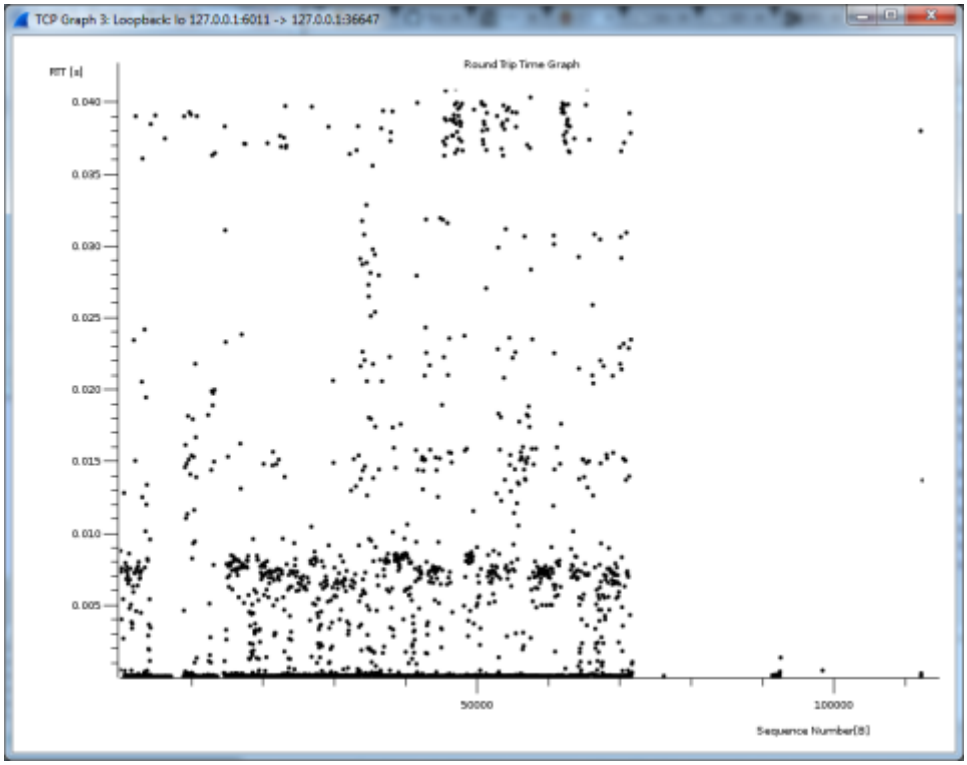
Παρατηρούμε ότι το εύρος ζώνης είναι όπως το ορίσαμε κατά την δημιουργία του δικτύου δηλαδή περίπου στα 10 Mbits/sec (μέσος όρος 9,72-9,90 Mbits/sec).



Γράφημα 5. 47 Time Sequence κατά την εκτέλεση iperf μεταξύ 2 ζευγών του δικτύου.

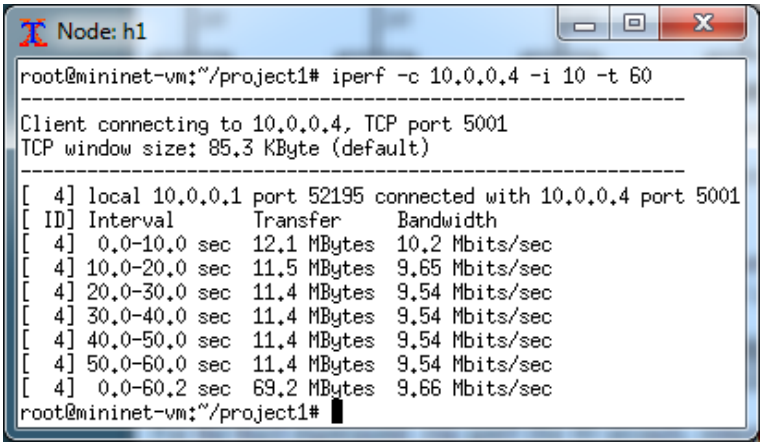


Γράφημα 5. 48 Throughput κατά την εκτέλεση iperf μεταξύ 2 ζευγών του δικτύου.



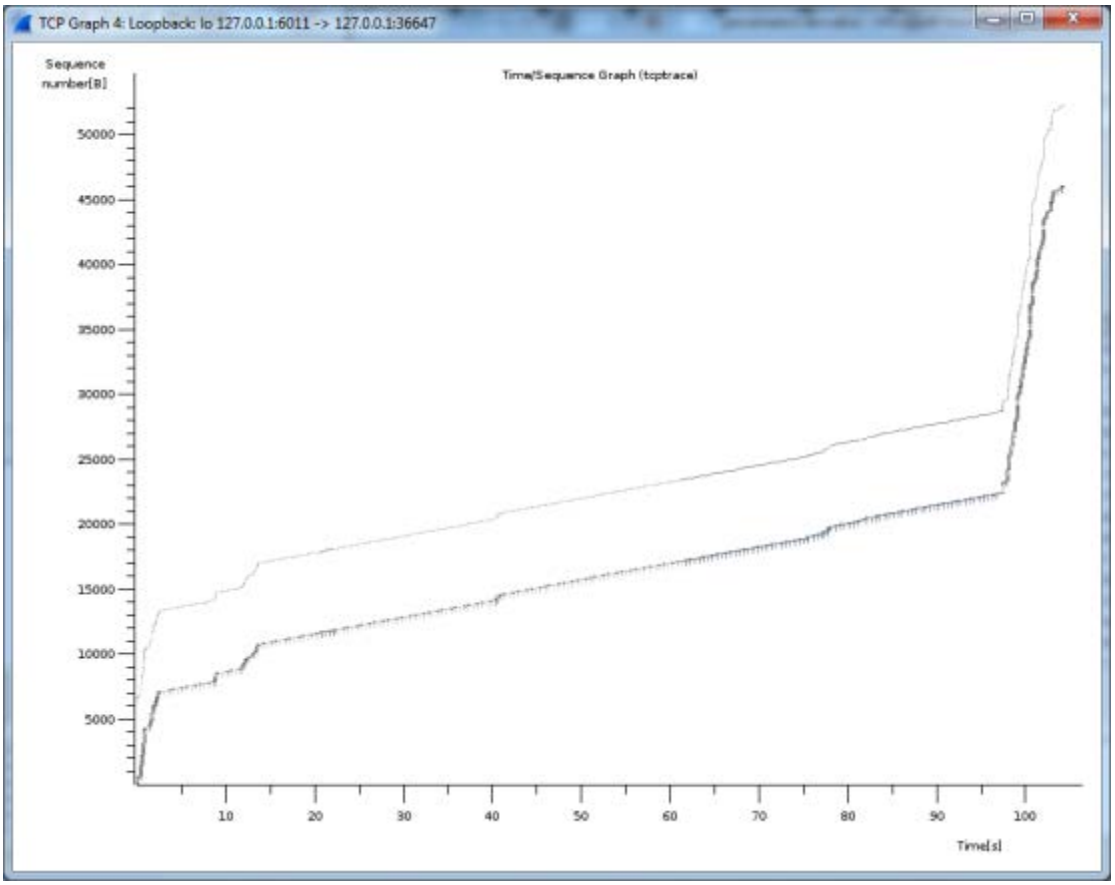
Γράφημα 5. 49 RTT Times κατά την εκτέλεση iperf μεταξύ 2 ζευγών του δικτύου.

Για το δεύτερο πείραμα πάνω στην συγκεκριμένη τοπολογία τρέχουμε την εντολή iperf για εξήντα δευτερόλεπτα εξάγοντας το εύρος ζώνης κάθε δέκα δευτερόλεπτα μεταξύ όμως ενός ζεύγους κόμβων και συγκεκριμένα των κόμβων h1 -> h4. Σκοπός του πειράματος είναι να δούμε αν υπάρχουν βασικές διαφορές στις τιμές του εύρους ζώνης που μας δίνει το Mininet ως αποτέλεσμα της εκτέλεσης συγκεκριμένων εντολών. Παρατηρούμε ότι το εύρος ζώνης κυμαίνεται στα 9,66 Mbits/sec για την περίπτωση αυτή.



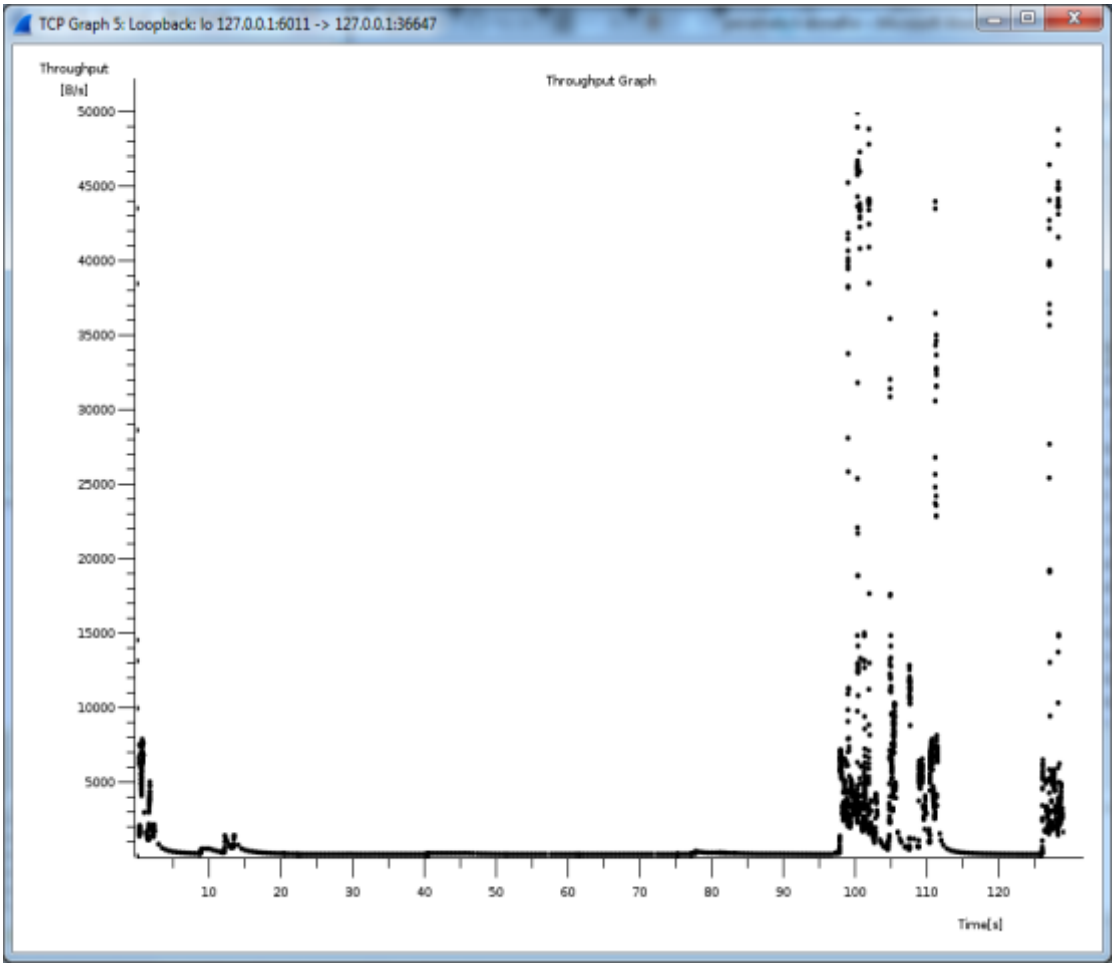
Εικόνα 5. 47 Iperf μεταξύ h1 & h4 κόμβου.

Παράλληλα με τα αποτελέσματα που έχουμε μέσω των xterm παραθύρων εξάγουμε τις παρακάτω γραφικές παραστάσεις με την βοήθεια του Wireshark.

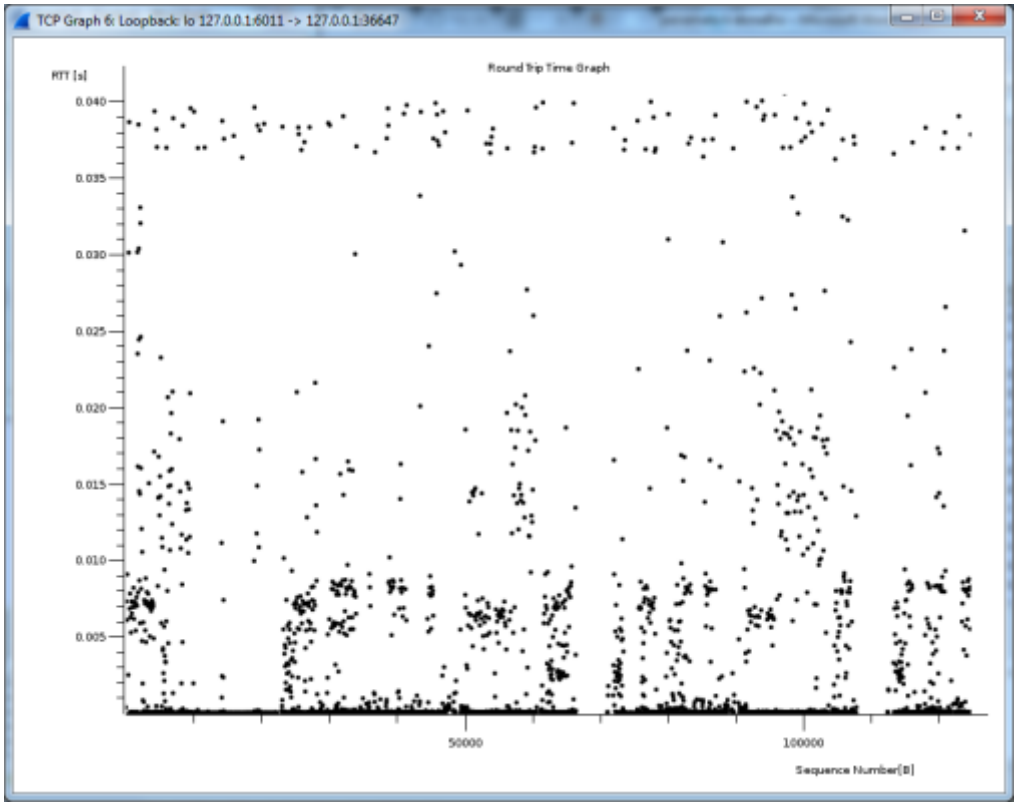


Γράφημα 5. 50 Time Sequence κατά την εκτέλεση iperf μεταξύ 1 ζεύγους του δικτύου.

Από το παραπάνω γράφημα καταλαβαίνουμε ότι η απόδοση του δικτύου αυξάνεται όσο περνάει ο χρόνος, μιας και η ιδανική λειτουργία ενός δικτύου αναπαριστάται στο είδος αυτό του γραφήματος ως μια διαγώνιο μεταξύ των δυο αξόνων.



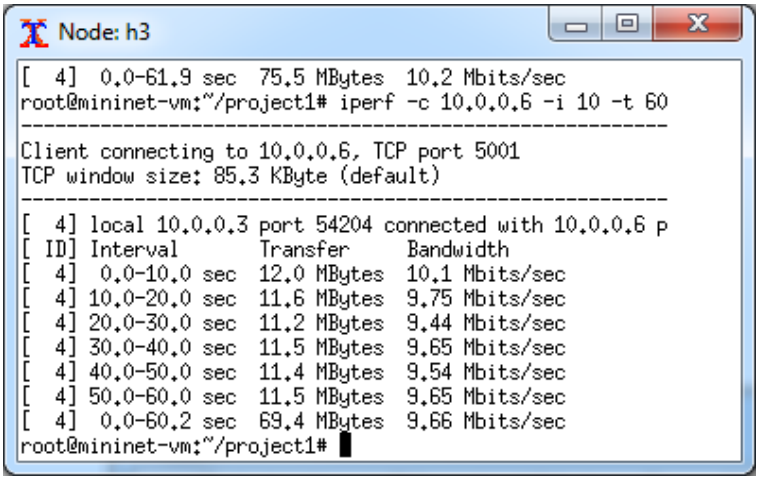
Γράφημα 5. 51 Throughput κατά την εκτέλεση iperf μεταξύ 1 ζεύγους του δικτύου.



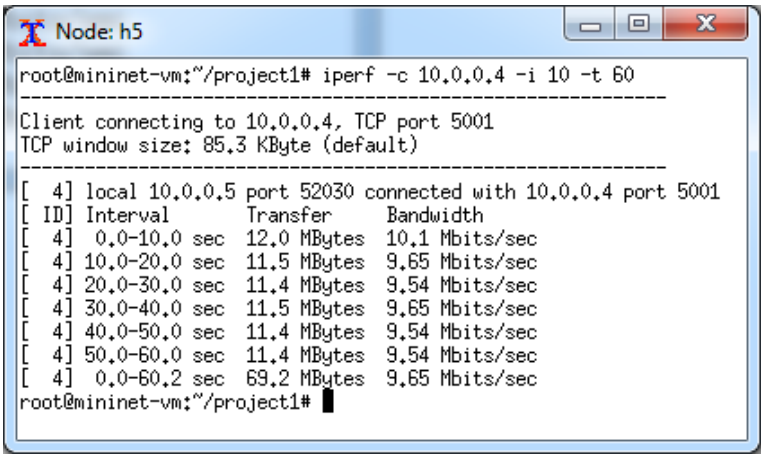
Γράφημα 5. 52 RTT Times κατά την εκτέλεση iperf μεταξύ 1 ζεύγους του δικτύου.

Παρατηρούμε ότι ο χρόνος καθυστέρησης μετά επιστροφής είναι λίγο μικρότερος στην περίπτωση της εκτέλεσης της εντολής iperf μεταξύ 1 ζεύγους του δικτύου σε σχέση με το γράφημα που παρουσίασαμε για την αντίστοιχη μεταβλητή στην περίπτωση της εκτέλεσης της εντολής iperf μεταξύ 2 ζεύγων του δικτύου.

Στο συγκεκριμένο πείραμα που αποτελεί την τρίτη εκδοχή για τη συγκεκριμένη τοπολογία δικτύου τρέχουμε την εντολή iperf για εξήντα δευτερόλεπτα ταυτόχρονα εξάγοντας το εύρος ζώνης κάθε δέκα δευτερόλεπτα μεταξύ δυο ζευγών κόμβων και συγκεκριμένα παράγουμε κίνηση μέσω της εντολής iperf από τον h3 στον h6 και από τον h5 στον h4 . Το πείραμα μοιάζει με αυτό που είχαμε εκτέλεση εντολής iperf μεταξύ 2 ζευγών κόμβων του δικτύου στην περίπτωση όμως αυτή έχουμε διαφορετικό εύρος ζώνης καθορισμένο ως σύνδεση όπως αποδεικνύει και το σχήμα της εικόνας 5.44.

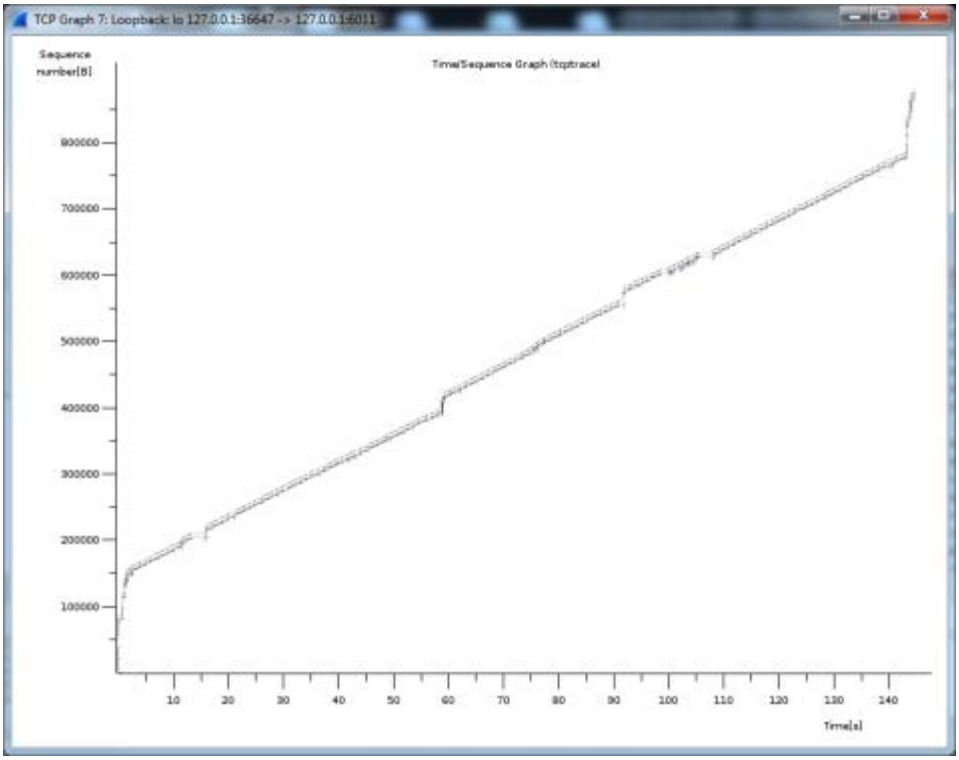


Εικόνα 5. 48 Iperf μεταξύ h3 & h6 κόμβου.

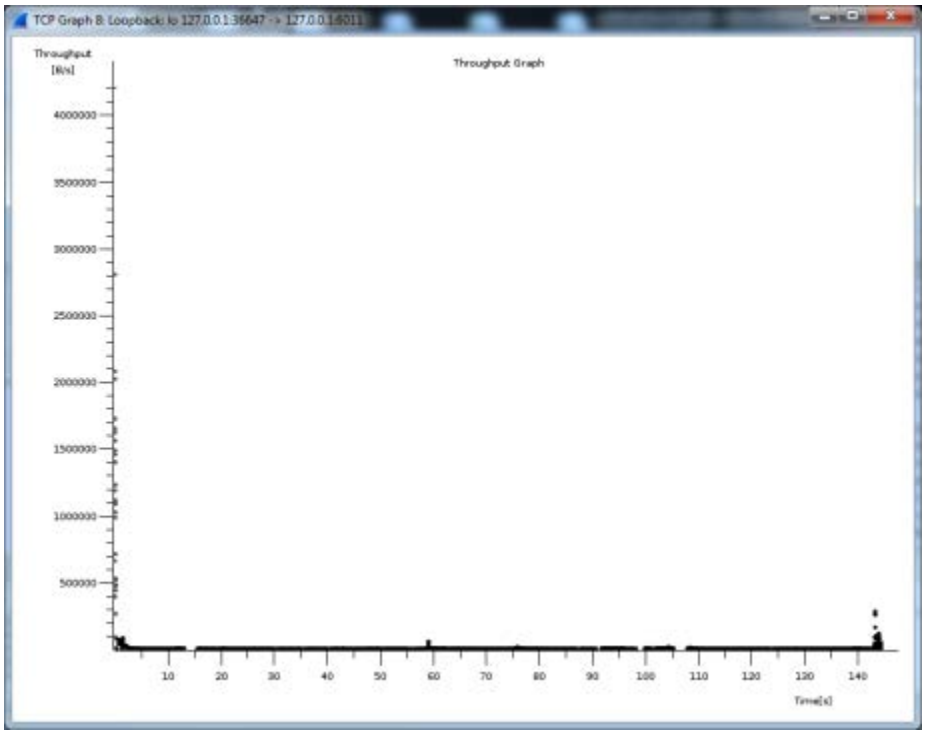


Εικόνα 5. 49 Iperf μεταξύ h5 & h4 κόμβου.

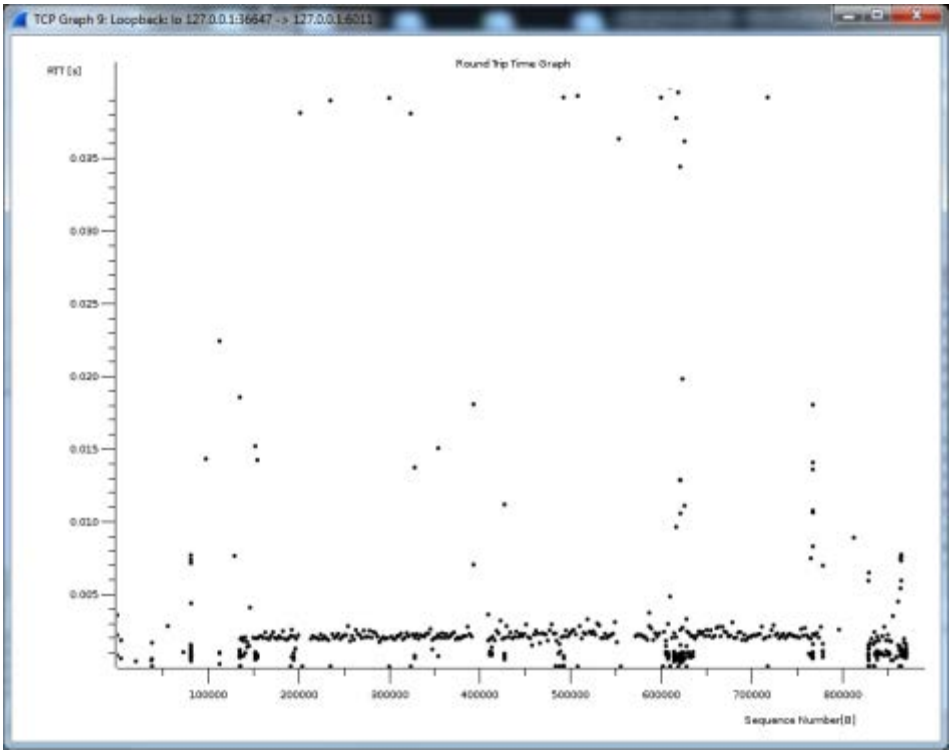
Παρατηρούμε ότι το εύρος ζώνης κυμαίνεται μεταξύ 9,65-9,66 Mbits/sec παρόλο που έχουμε καθορίσει την σύνδεση μεταξύ των κόμβων των μεταγωγέων να είναι 15 Mbps. Προφανώς το δίκτυο "δεσμεύεται" λόγω της σύνδεσης των 10 Mbps μεταξύ κόμβου και μεταγωγέα. Παρόλα αυτά στο γράφημα Time Sequence βλέπουμε μια άριστη απόδοση του δικτύου όπως θα περίμενε κανείς σε θεωρητικό επίπεδο.



Γράφημα 5. 53 Time Sequence κατά την εκτέλεση iperf μεταξύ 2 ζευγών του δικτύου (τρίτη εκδοχή).



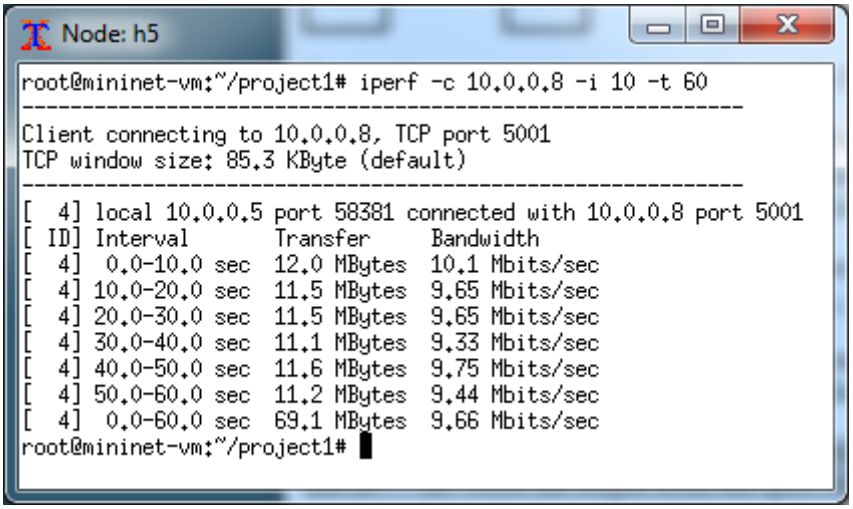
Γράφημα 5. 54 Throughput κατά την εκτέλεση iperf μεταξύ 2 ζευγών του δικτύου (τρίτη εκδοχή).



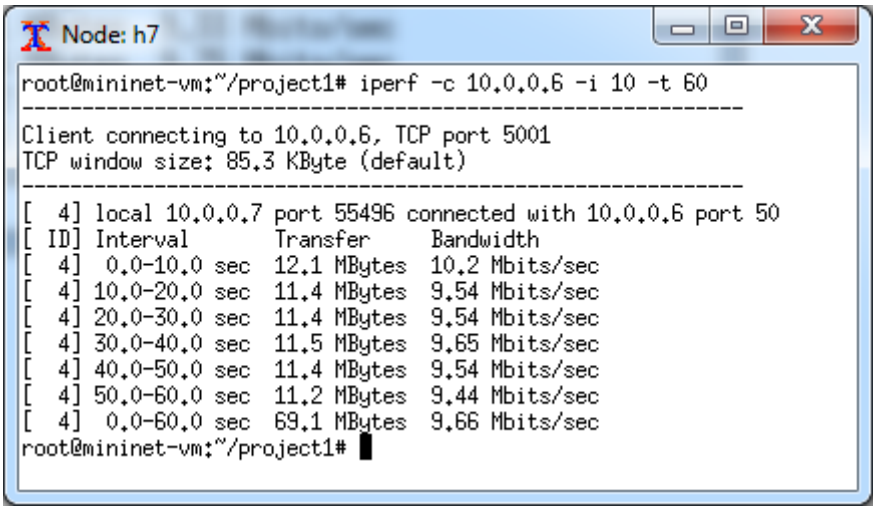
Γράφημα 5. 55 RTT χρόνοι κατά την εκτέλεση iperf μεταξύ 2 ζευγών του δικτύου (τρίτη εκδοχή).

Παρατηρούμε ότι οι χρόνοι RTT είναι πολύ ικανοποιητικοί. Αυτό αποτελεί και καθοριστικό παράγοντα για την σωστή λειτουργία και απόδοση του δικτύου. Η διακύμανση του RTT σε αυτήν την τρίτη εκδοχή του πειράματος για την εξεταζόμενη τοπολογία είναι πολύ μικρή. Το γεγονός αυτό είναι καθοριστικό για την απόδοση του δικτύου και έχει ως αποτέλεσμα την εικόνα που είχαμε στο γράφημα 5.53 του TimeSequence.

Στη τέταρτη εκδοχή θα τρέξουμε την εντολή iperf κάθε εξήντα δευτερόλεπτα εξάγοντας το εύρος ζώνης κάθε δέκα δευτερόλεπτα μεταξύ δυο ζευγών κόμβων ταυτόχρονα δηλαδή από τον h5 -> h8 και από τον h7 -> h6

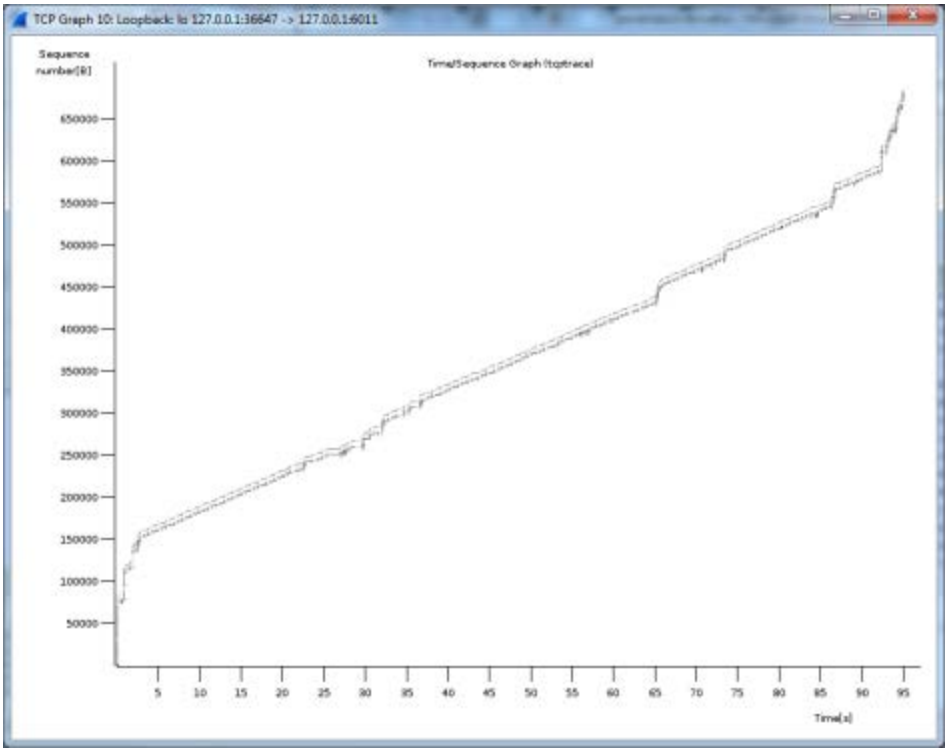


Εικόνα 5. 50 Iperf h5 -> h8.

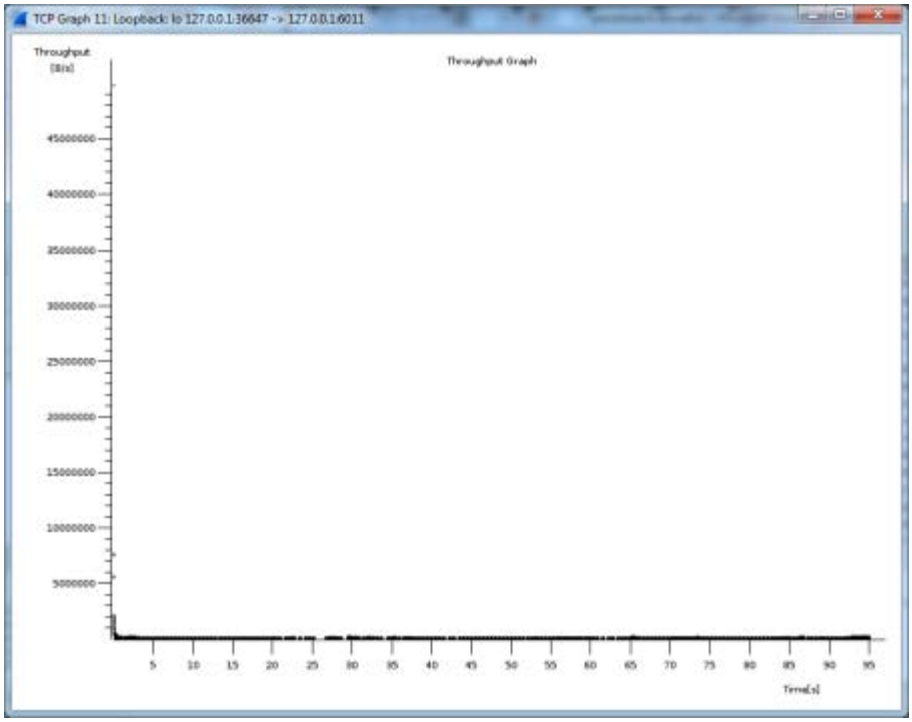


Εικόνα 5. 51 Iperf h7 -> h6.

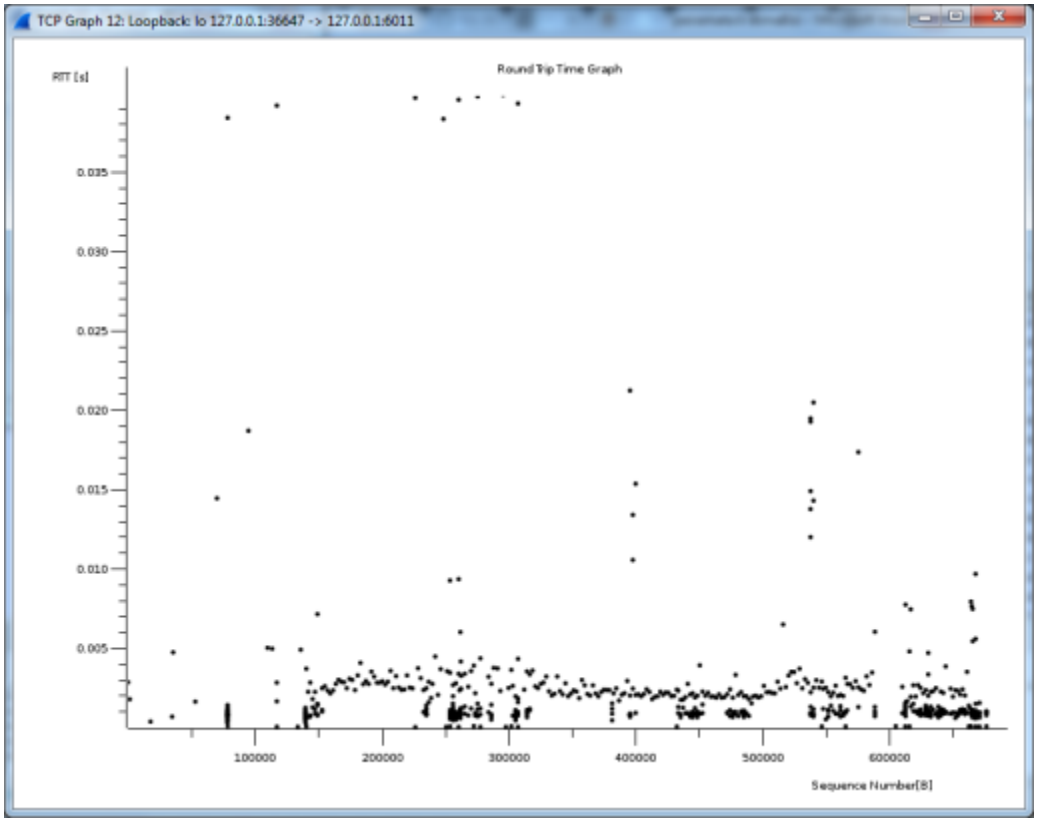
Τα γραφήματα που δημιουργήθησαν κατά την διάρκεια της τέταρτης εκδοχής του πειράματος ακολουθούν.



Γράφημα 5. 56 Time Sequence κατά την εκτέλεση iperf h5 -> h8 & h7 -> h6



Γράφημα 5. 57 Throughput κατά την εκτέλεση iperf h5 -> h8 & h7 -> h6



Γράφημα 5. 58 RTT χρόνοι κατά την εκτέλεση iperf h5 -> h8 & h7 -> h6

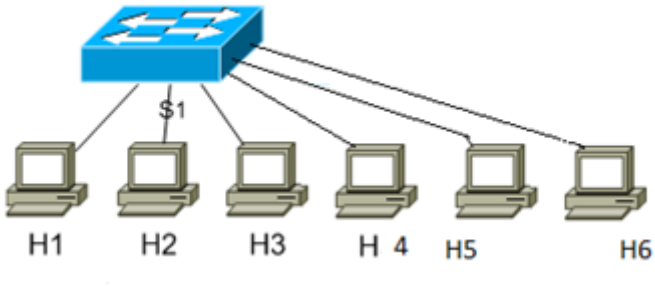
Παρατηρούμε από τα παραπάνω πειράματα ότι το πρώτο πείραμα εμφανίζει μικρότερο εύρος ζώνης από το δεύτερο, παρατήρηση λογική μιας και η κίνηση στο δίκτυο είναι διπλάσια στην πρώτη περίπτωση από την δεύτερη. Στο τρίτο πείραμα παρατηρούμε ότι έχουμε μια πιο σωστή απόδοση του δικτύου σε σχέση με το πρώτο πείραμα στο συγκεκριμένο δίκτυο, βέβαια αυτό δικαιολογείται και από το γεγονός, ότι στο τρίτο πείραμα παρατηρούμε τα δεδομένα που εξάγονται όταν μεταφέρονται μέσω μιας σύνδεσης αυξημένης κατά 5 Mbits/sec, σε σχέση με τη σύνδεση του πρώτου πειράματος που περιοριζόμαστε στα 10 Mbits/sec.

Αν συγκρίνουμε τα δεδομένα που λάβαμε όσον αφορά το εύρος ζώνης, του δεύτερου πειράματος στο συγκεκριμένο δίκτυο και του τέταρτου πειράματος μπορούμε να συμπεράνουμε ότι το εύρος ζώνης είναι και στις δύο περιπτώσεις γύρω στο 9,66 Mbits/sec, αλλά η απόδοση του δικτύου στο τέταρτο πείραμα είναι πολύ καλύτερη. Ακόμα, ο χρόνος RTT είναι πολύ μικρότερος στο τελευταίο

πείραμα γεγονός που δείχνει ότι είναι πιο εύκολο να τρέξουν πιο απαιτητικές εφαρμογές στον εξυπηρετητή - κόμβο 6 από ότι στον εξυπηρετητή κόμβο 4.

5.5 Επίδραση Καθυστέρησης & Απώλειας σε Δίκτυα

Στα παρακάτω πειράματα δημιουργούμε την παρακάτω συνδεσμολογία δικτύου



Εικόνα 5. 52 Απλή συνδεσμολογία 6 κόμβων και 1 μεταγωγέα.

Στην συνέχεια θα εκτελέσουμε διαφορετικές εντολές στο Mininet ώστε να εξετάσουμε την επίδραση της απώλειας (loss) πακέτων καθώς και την επίδραση της καθυστέρησης στο δίκτυο.

5.5.1 Έλεγχος Λειτουργίας Δικτύου

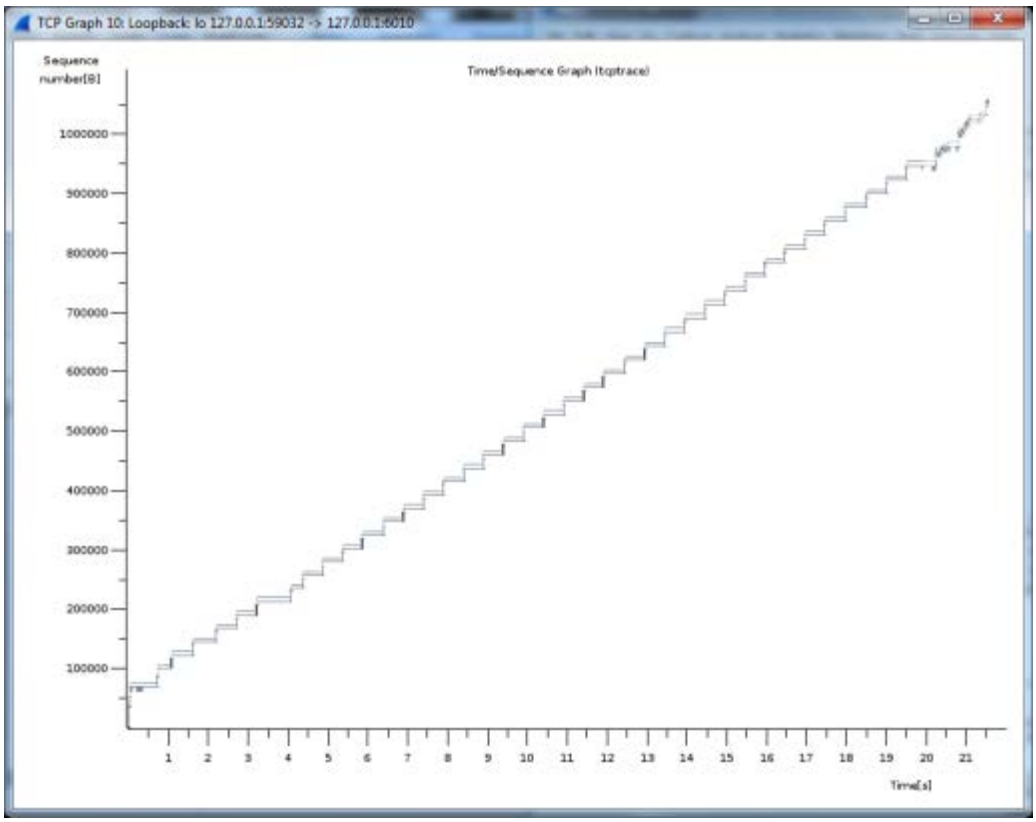
Η εντολή που δημιουργεί την παραπάνω τοπολογία στο Mininet είναι η παρακάτω:

$ sudo mn --test pingall --topo single,6 --link tc,bw=20

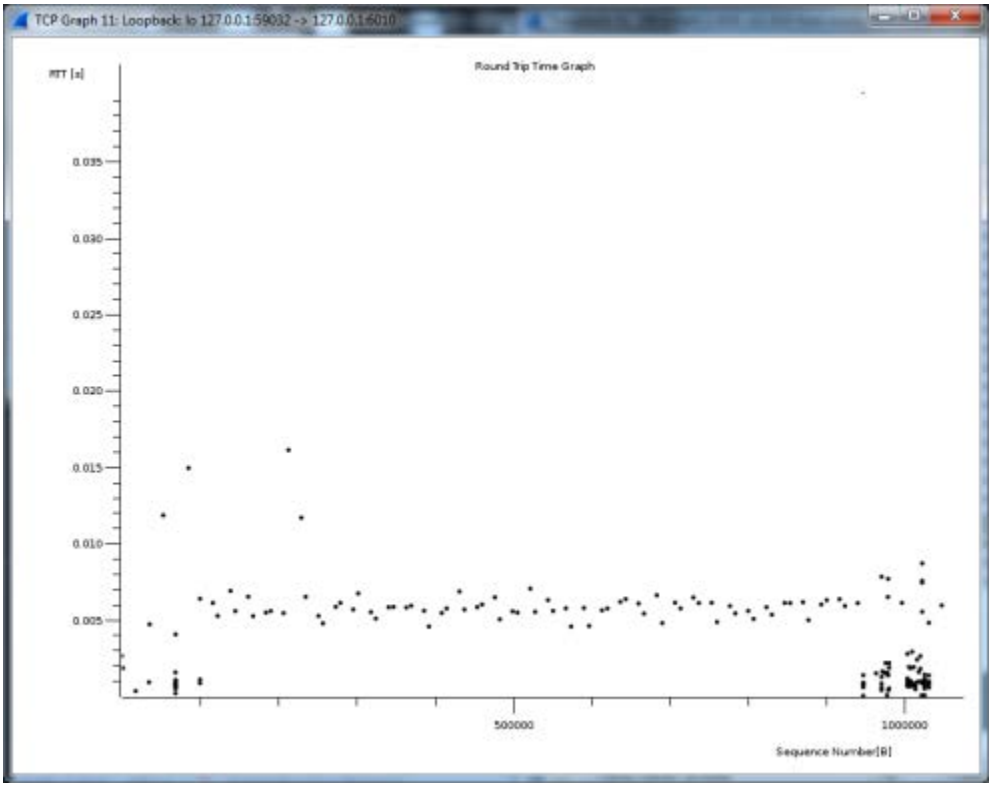
Με αυτήν την εντολή ταυτόχρονα με την δημιουργία του δικτύου επίσης ελέγχουμε τα αποτελέσματα της ping εντολής σε όλους τους κόμβους του δικτύου με την παράμετρο **--test pingall** και θέτουμε το εύρος ζώνης στα 20 Mbits/sec με την παράμετρο **--link tc,bw=20** [47]. Στην παρακάτω εικόνα του Mininet φαίνεται η δημιουργία και έναρξη λειτουργίας του δικτύου καθώς και

τα αποτελέσματα της εντολής pingall. Παρατηρούμε ότι οι κόμβοι όλοι επικοινωνούν μεταξύ τους και ότι το εύρος ζώνης είναι 20 Mbits/sec όπως ορίσαμε. Το πείραμα ολοκληρώνεται μέσα σε 2-3 δευτερόλεπτα. [10]

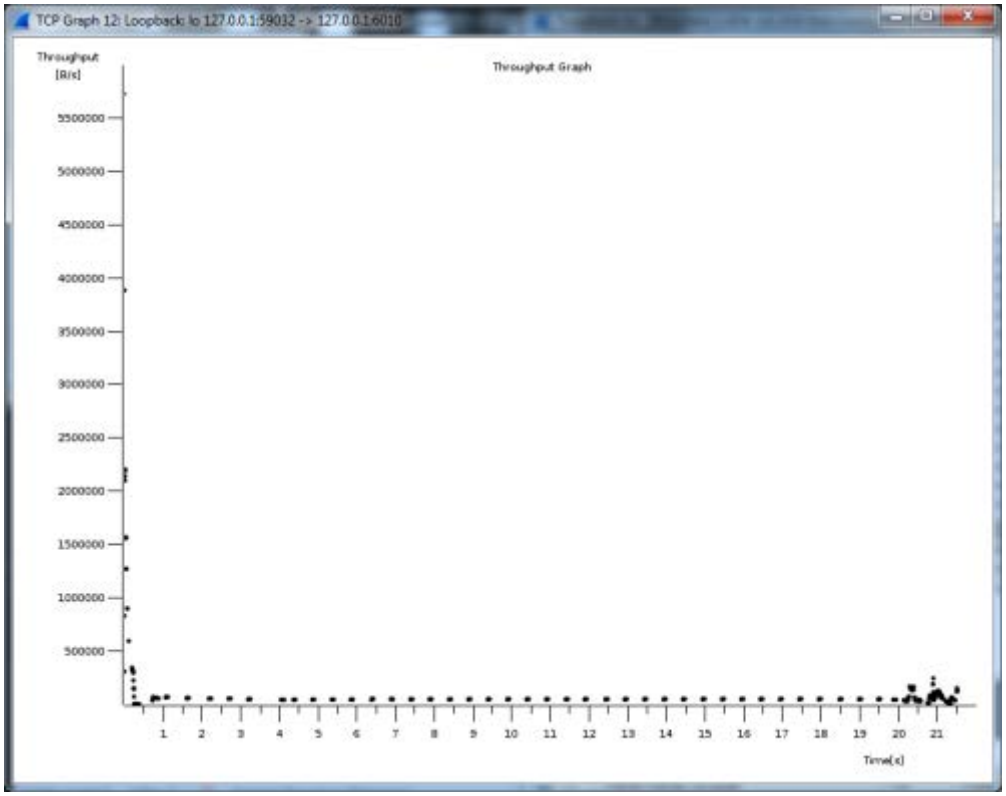
Οι γραφικές παραστάσεις που δημιουργήθηκαν κατά την διάρκεια δημιουργίας του δικτύου και του pingall test ακολουθούν. Παρατηρούμε πολύ μικρούς χρόνους RTT. Παράλληλα η απόδοση του δικτύου όπως αυτή φαίνεται στο tcp trace, Time Sequence Graph είναι ικανοποιητική. Παράλληλα στην αρχή του γραφήματος του throughput graph βλέπουμε και ότι το throughput του δικτύου είναι της τάξης των 20 Mbits/sec όπως ορίσαμε κατά την δημιουργία του.



Γράφημα 5. 59 Time Sequence απλού δικτύου 6 κόμβων και 1 μεταγωγέα.



Γράφημα 5. 60 Throughput απλού δικτύου 6 κόμβων και 1 μεταγωγέα.



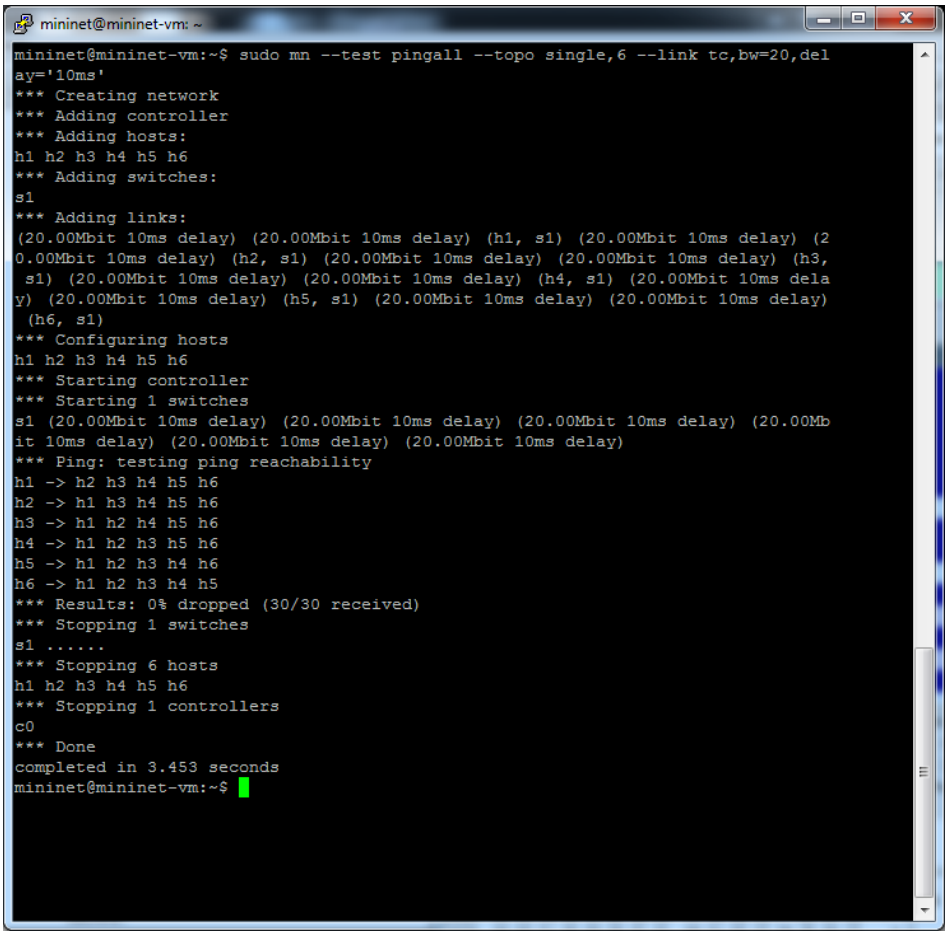
Γράφημα 5. 61 RTT απλού δικτύου 6 κόμβων και 1 μεταγωγέα.

5.5.2 Εισαγωγή Καθυστέρησης στο Δίκτυο

Στο παρακάτω πείραμα δημιουργούμε πάλι την τοπολογία μιας απλής σύνδεσης ενός μεταγωγέα και έξι κόμβων που ελέγχονται από τον προκαθορισμένο controller και οι μεταξύ τους συνδέσεις έχουν εύρος ζώνης 20 Mbits/sec αλλά προσθέτουμε μια καθυστέρηση μεταξύ τους της τάξης των 10ms. Η εντολή που δίνεται έχει την παρακάτω μορφή (έχουμε και πάλι προσθέσει ένα τεστ pingall στο δίκτυο):

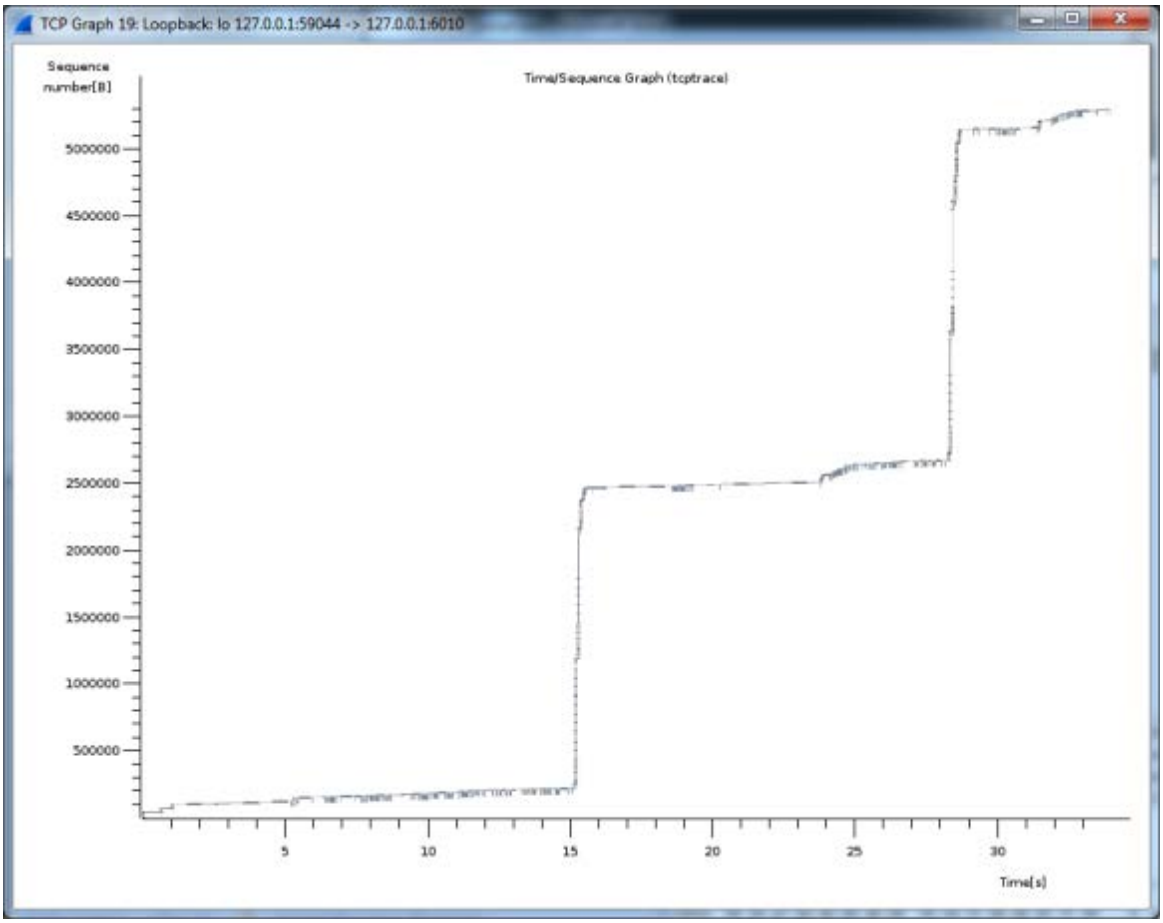
$ sudo mn --test pingall --topo single,6 --link tc,bw=20,delay='10ms' [48]

Η εικόνα που ακολουθεί δείχνει την απόκριση του Mininet στην παραπάνω εντολή. Παρατηρούμε την εισαγωγή καθυστέρησης μεταξύ όλων των συνδέσεων.

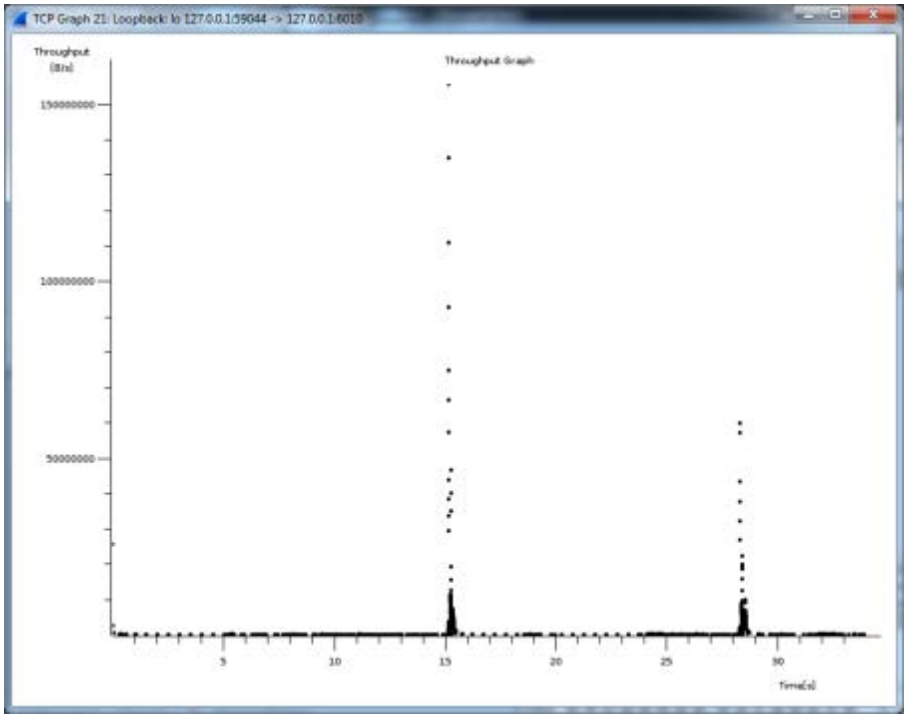


Εικόνα 5. 53 Εισαγωγή καθυστέρησης σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα.

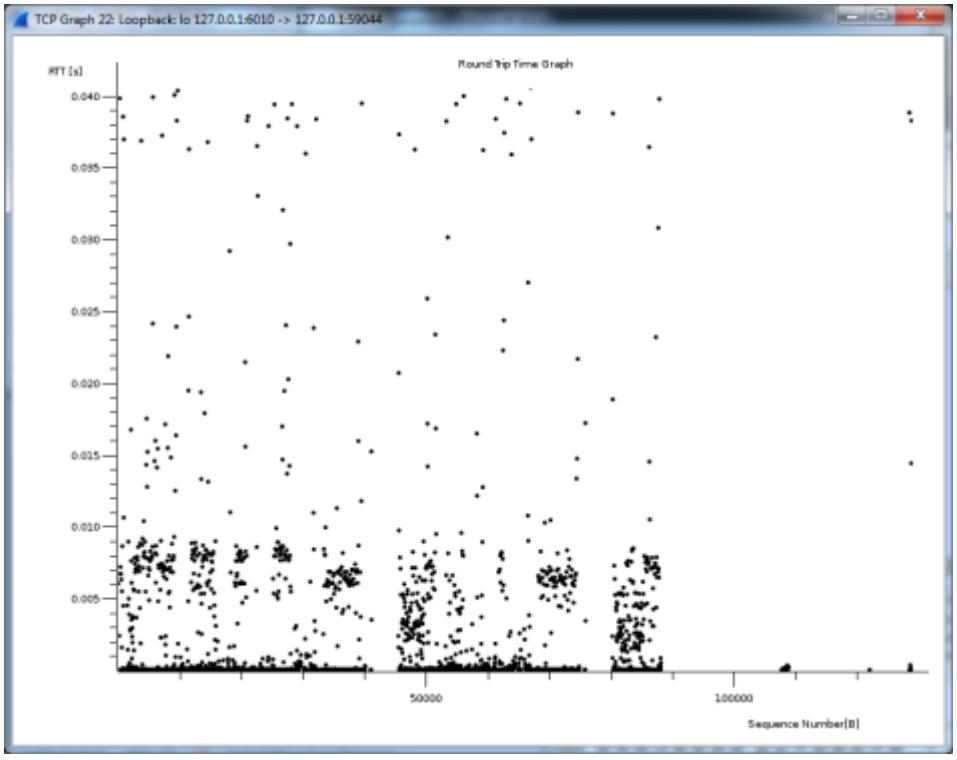
Τα γραφήματα που παίρνουμε κατά την διάρκεια της δημιουργίας του δικτύου ακολουθούν. Στο γράφημα Time Sequence (tcptrace) βλέπουμε το αποτέλεσμα της εισαγωγής καθυστέρησης σε όλες τις συνδέσεις του δικτύου. Παρατηρούμε ότι η απόδοση του δικτύου δεν έχει την μορφή του προηγούμενου πειράματος. Ουσιαστικά αύξηση έχουμε μετά την πάροδο σχεδόν 15 δευτερολέπτων παρόλο την εισαγωγή καθυστέρησης ίσης με δέκα δευτερόλεπτα. Φυσικά το παραπάνω έχει αντίκτυπο και στον χρόνο RTT όπως φαίνεται πλέον κυμαίνεται ως τα 0,040 sec.



Γράφημα 5. 62 Time Sequence σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση.



Γράφημα 5. 63 Throughput σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση.



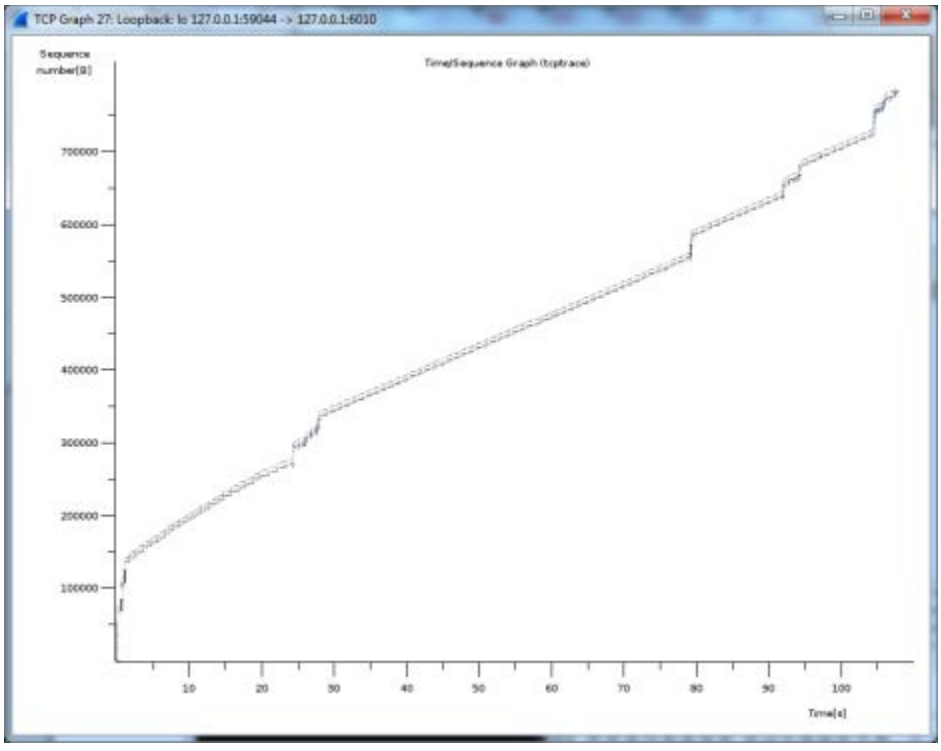
Γράφημα 5. 64 RTT σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση.

5.5.3 Εισαγωγή Απώλειας

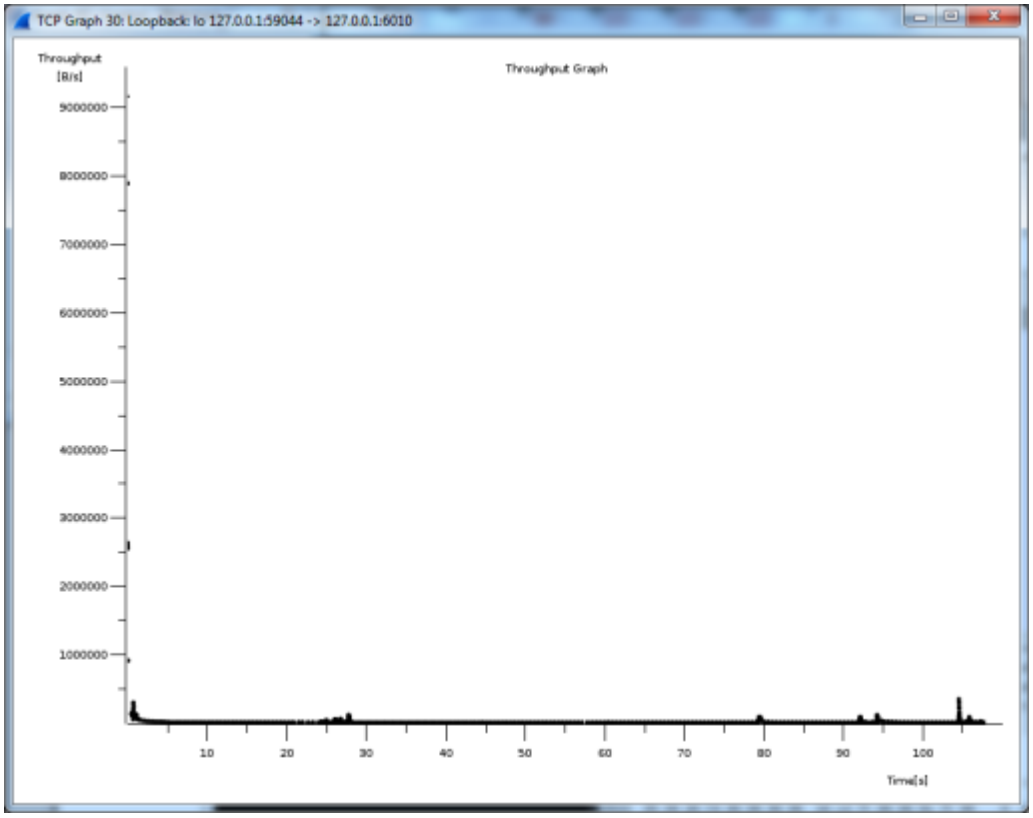
Παρακάτω κάνουμε μια δεύτερη παραλλαγή στο ακριβώς προηγούμενο πείραμα προσθέτοντας απώλεια 1% κατά την διάρκεια της επικοινωνίας των κόμβων. Η εντολή που δίνουμε στο Mininet για την δημιουργία ενός τέτοιου δικτύου είναι η παρακάτω (η παράμετρος που ορίζει την απώλεια τίθεται με κόμμα δίπλα στην καθυστέρηση με τον όρο **loss=1**)

$ sudo mn --test pingall --topo single,6 --link tc,bw=20,delay='10ms',loss=1

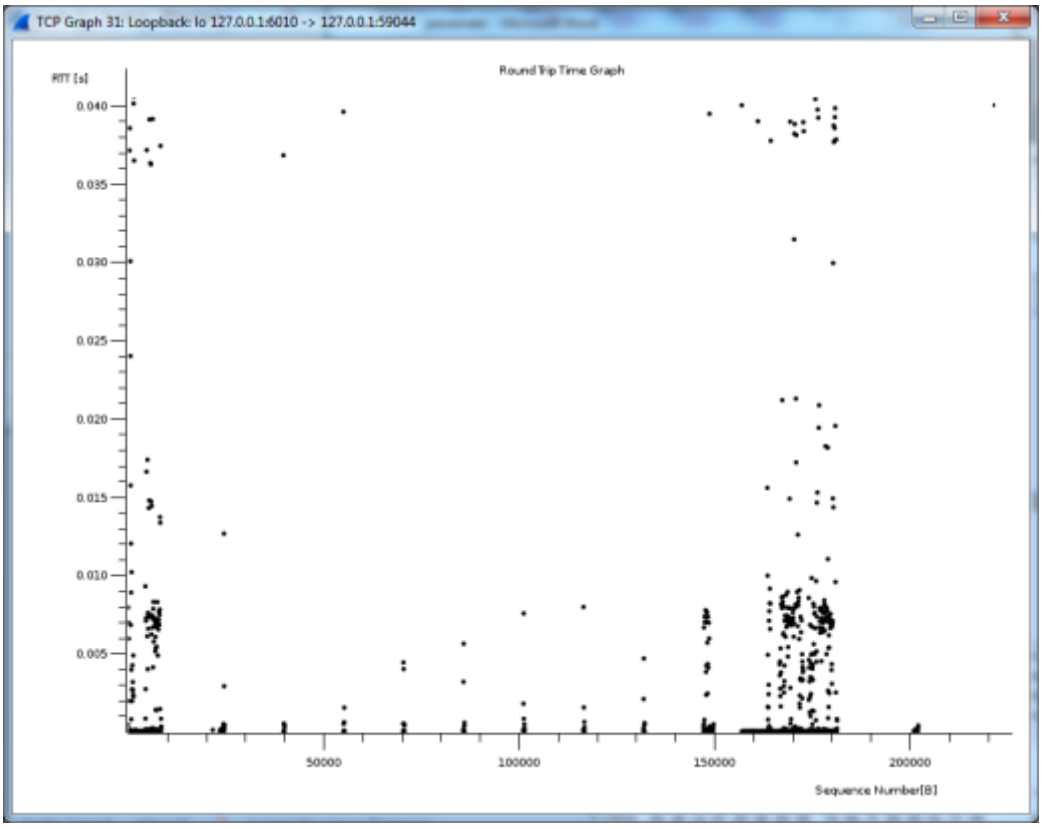
Από τα αποτελέσματα που έχουμε κατά τη δημιουργία του δικτύου, της εγκαθίδρυσης των συγκεκριμένων συνδέσεων με τα προαναφερόμενα χαρακτηριστικά καθώς και τα αποτελέσματα από το pingall στο δίκτυο, φαίνεται ξεκάθαρα η απώλεια κάποιων πακέτων κατά την διάρκεια του pingall γεγονός που δεν συνέβαινε σε κανένα προηγούμενο πείραμα. Παρατηρούμε επίσης ότι το 13% είναι dropped, έχουν δηλαδή χαθεί. Το πείραμα όπως ήταν αναμενόμενο έκανε παραπάνω χρόνο για να ολοκληρωθεί.



Γράφημα 5. 65 Time Sequence σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με απώλεια.



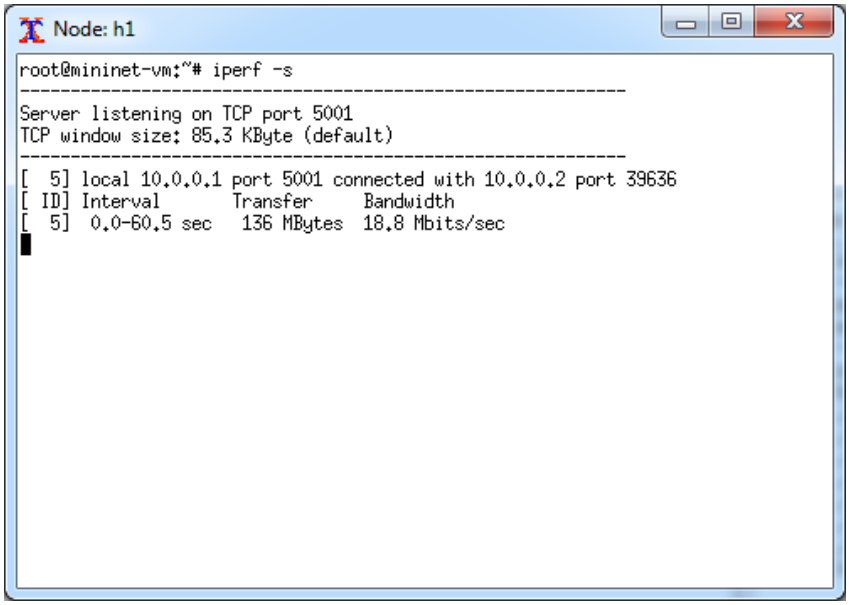
Γράφημα 5. 66 Throughput σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με απώλεια.



Γράφημα 5. 67 RTT σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με απώλεια.

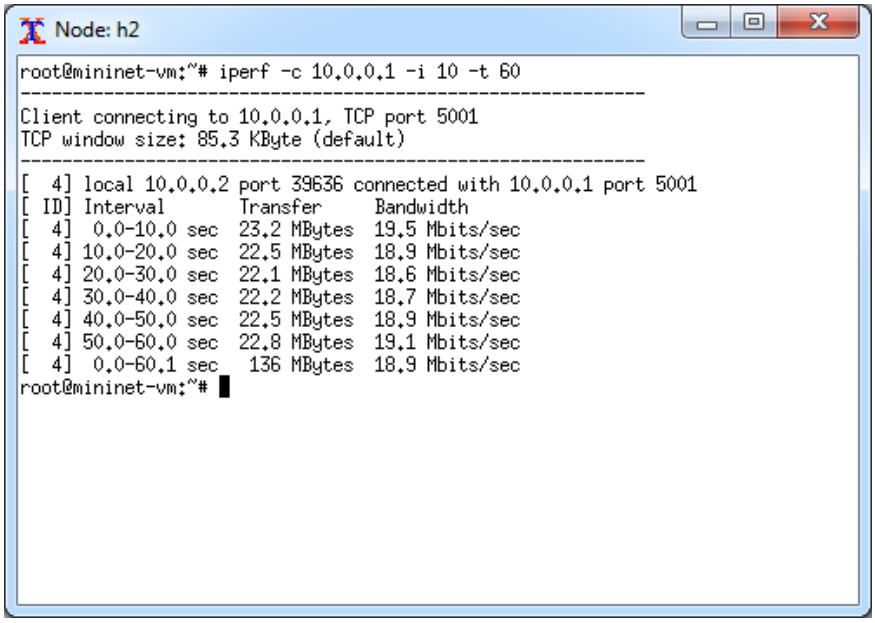
Εάν δώσουμε την εντολή: mininet> **xterm h1 h2 h3 h4 h5 h6**

Με αυτήν την εντολή ανοίγουν έξι παράθυρα xterm ώστε να μπορούμε να δώσουμε άμεσα εντολές στο κάθε κόμβο του δικτύου. Ορίζουμε ως εξυπηρετητή τον κόμβο h1 με την εντολή **h1# iperf -s** όπως φαίνεται και στην παρακάτω εικόνα.

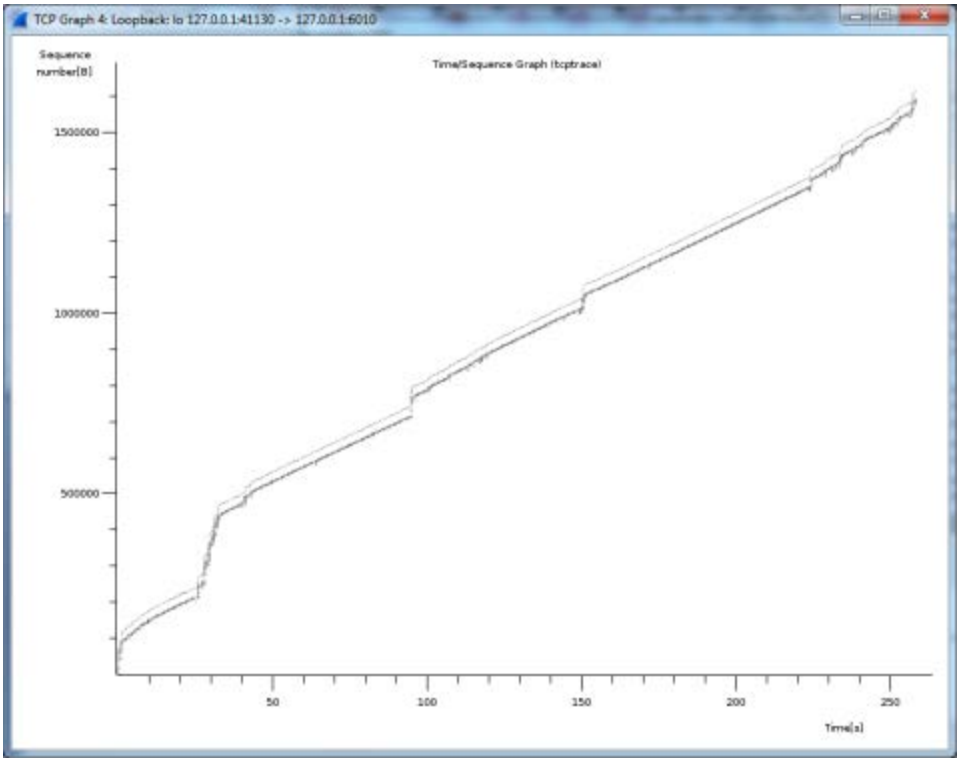


Εικόνα 5. 54 Δημιουργία του h1 ως εξυπηρετητή.

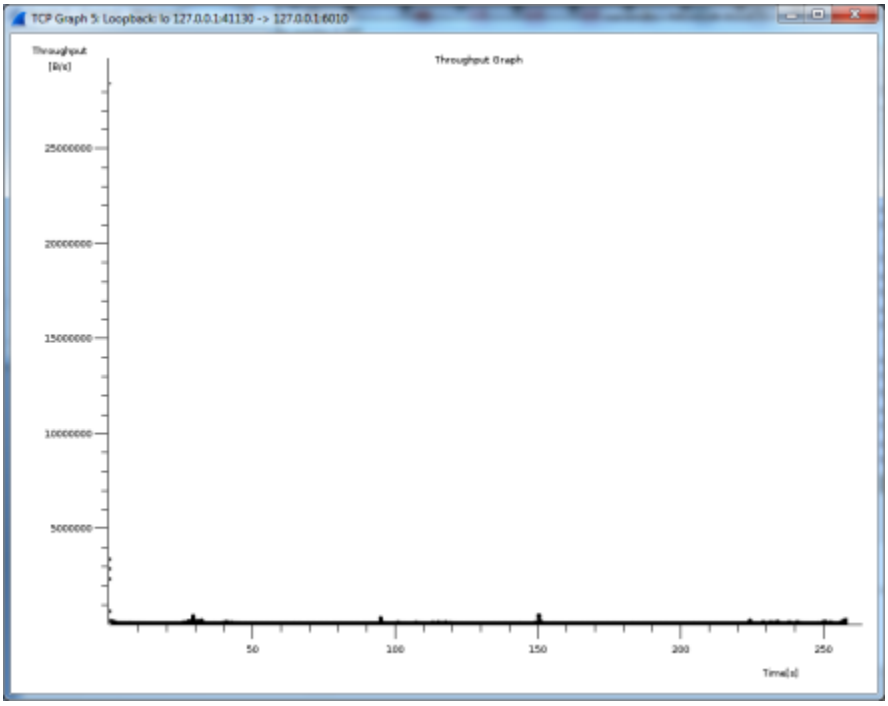
Στον κόμβο h2 δίνουμε την εντολή **h2#iperf -c 10.0.0.1 -i -t 60** ώστε να δημιουργήσουμε κίνηση μεταξύ εξυπηρετητή και πελάτη. Τα γραφήματα που εξάγαμε ακολουθούν.



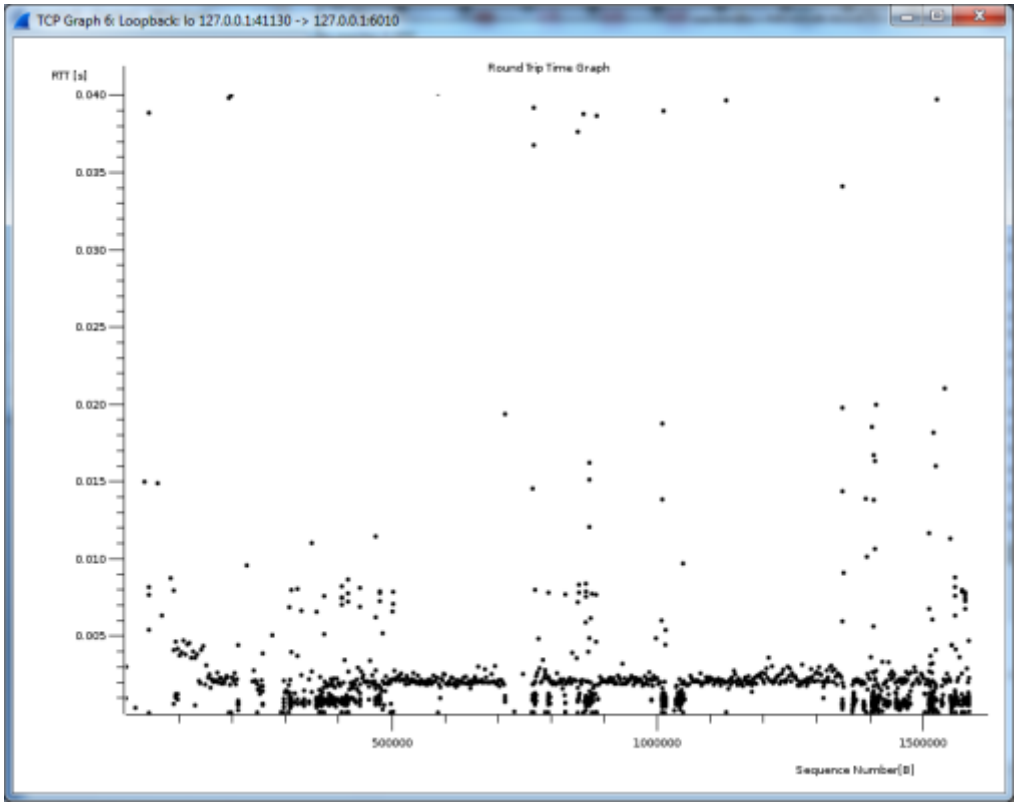
Εικόνα 5. 55 Δημιουργία του h2 ως πελάτη και εκτέλεση iperf στον h1.



Γράφημα 5. 68 Time Sequence σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση & με απώλεια ενώ εκτελείται iperf.

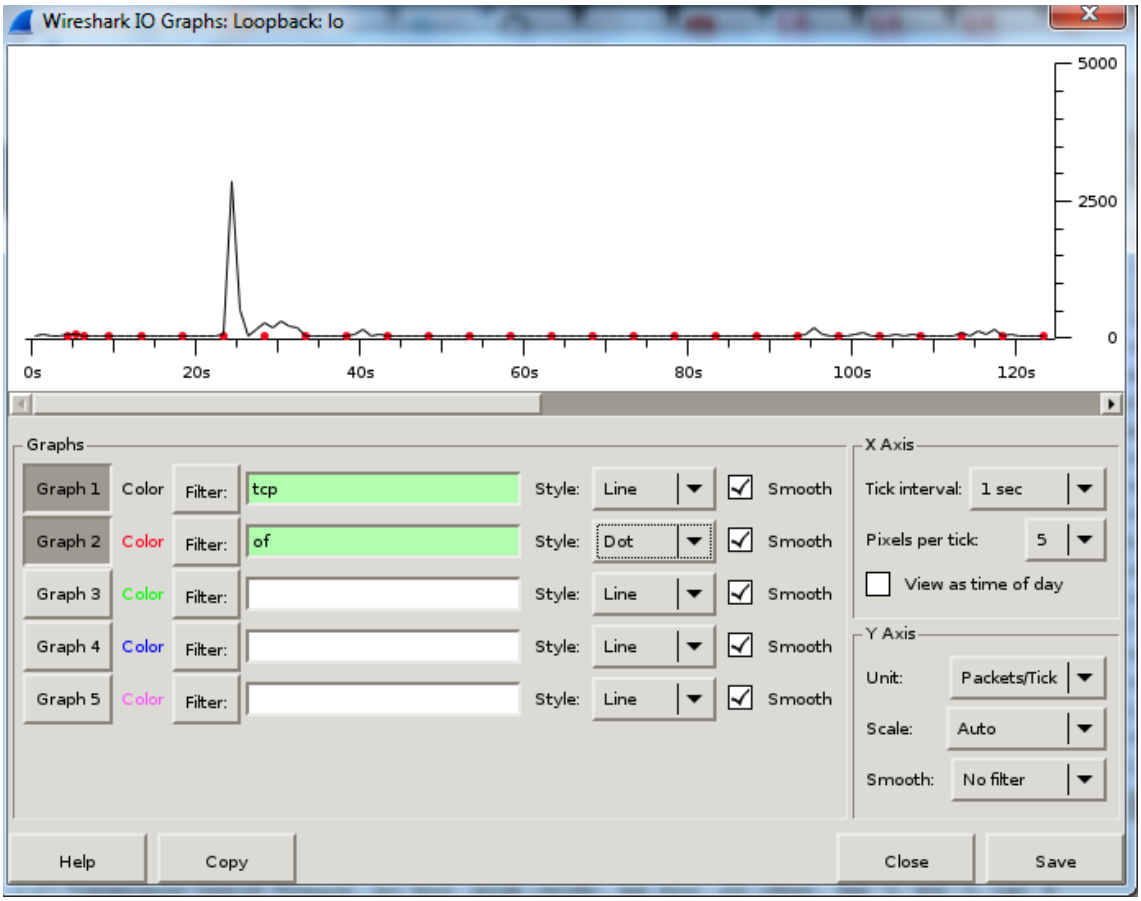


Γράφημα 5. 69 Throughput σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση & με απώλεια ενώ εκτελείται iperf.



Γράφημα 5. 70 RTT σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση & με απώλεια ενώ εκτελείται iperf.

Από τα παραπάνω γραφήματα είναι εξαιρετικά δύσκολο να αναγνωρίσουμε την απώλεια των πακέτων της τάξεως του 1% που ορίσαμε κατά την δημιουργία του δικτύου. Παρατηρούμε όμως στα γραφήματα τα αποτελέσματα της καθυστέρησης που είναι της τάξεως των 10 msec. Σαν συμπέρασμα καταλήγουμε ότι φαινόμενα όπως απώλεια πακέτων μπορούν να παρατηρηθούν πιο εύκολα (τουλάχιστον τέτοιας κλίμακας) από αποτελέσματα εντολών όπως η ping παρά από γραφήματα. Αντίθετα η καθυστέρηση απεικονίζεται ευκρινώς στα γραφήματα του Wireshark και είναι εύκολα αντιληπτή.



Γράφημα 5. 71 Γραφική παράσταση των πακέτων OF & TCP σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση & με απώλεια ενώ εκτελείται iperf.

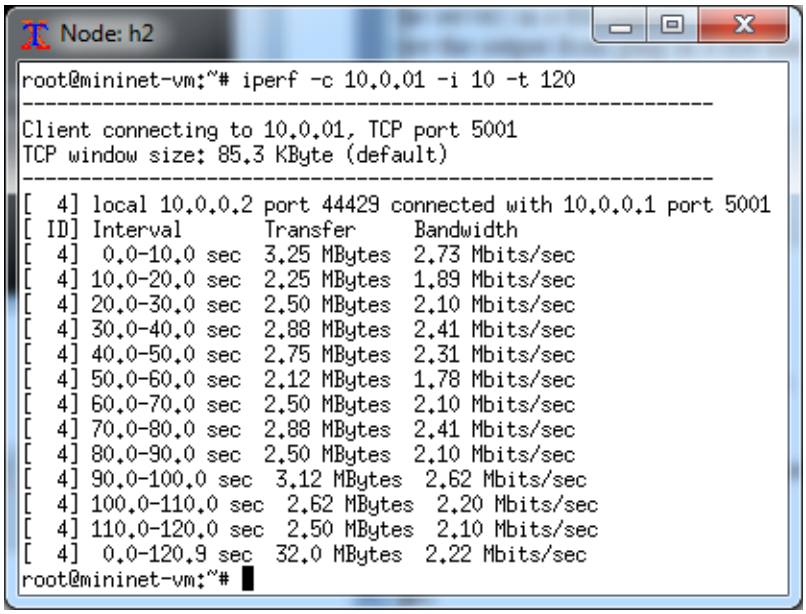
Παρακάτω αφού δώσουμε την εντολή **$ sudo mn -c** για να καθαρίσει τα δεδομένα της προηγούμενης εξομοίωσης δίνουμε την εντολή:

$ sudo mn --topo single,6 --link tc,bw=20,delay='10ms',loss=1

δηλαδή χωρίς την παράμετρο του pingall.

Επίσης δίνουμε την εντολή mininet>**xterm h1 h2 h3 h4 h5 h6** οπότε ανοίγουν τα 6 xterm παράθυρα αντίστοιχα για τον κάθε κόμβο. Στόχος αυτού του πειράματος είναι να εξάγουμε παρατηρήσεις για την κίνηση από το δίκτυο που έχουμε δημιουργήσει δημιουργώντας ροή πληροφορίας μεταξύ ζευγών κόμβων.

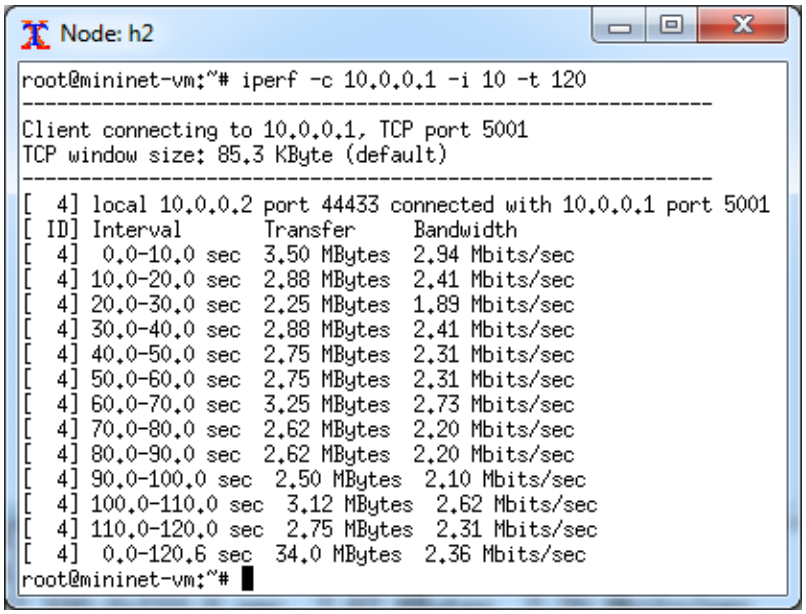
Για να δημιουργήσουμε κίνηση ορίσαμε μερικούς hosts ως servers (**iperf -s**) και μερικούς ως clients (**iperf -c**). Θέτουμε έτσι τον h1 ως server με την εντολή **h1#iperf -s** και τον h2 ως client δημιουργώντας κίνηση με την εντολή **h2# iperf -c 10.0.0.1 -i 10 -t 120**. Δημιουργώ δηλαδή ένα ζεύγος κόμβων που παρουσιάζει κίνηση στο δίκτυο.



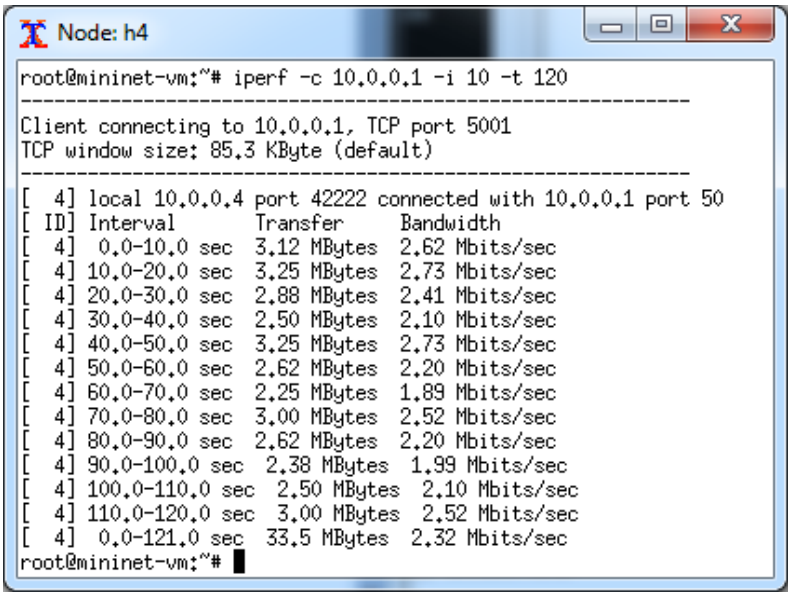
Εικόνα 5. 56 Αποτελέσματα iperf -c 10.0.0.1 -i 10 -t 120.

Παρατηρούμε από την παραπάνω εικόνα ότι το εύρος ζώνης έχει μειωθεί σημαντικά και ενώ είναι ορισμένο στα 20 Mbits/sec ως μέσος όρος εμφανίζεται

από τα αποτελέσματα του πειράματος να είναι στα 2,22 Mbits/sec. Εάν τώρα δημιουργήσουμε κίνηση από δυο κόμβους πχ τους h2 & h4 στον h1 κόμβο που λειτουργεί ως server με την εντολή **# iperf -c 10.0.0.1 -i 10 -t 120** λαμβάνουμε τα παρακάτω αποτελέσματα:



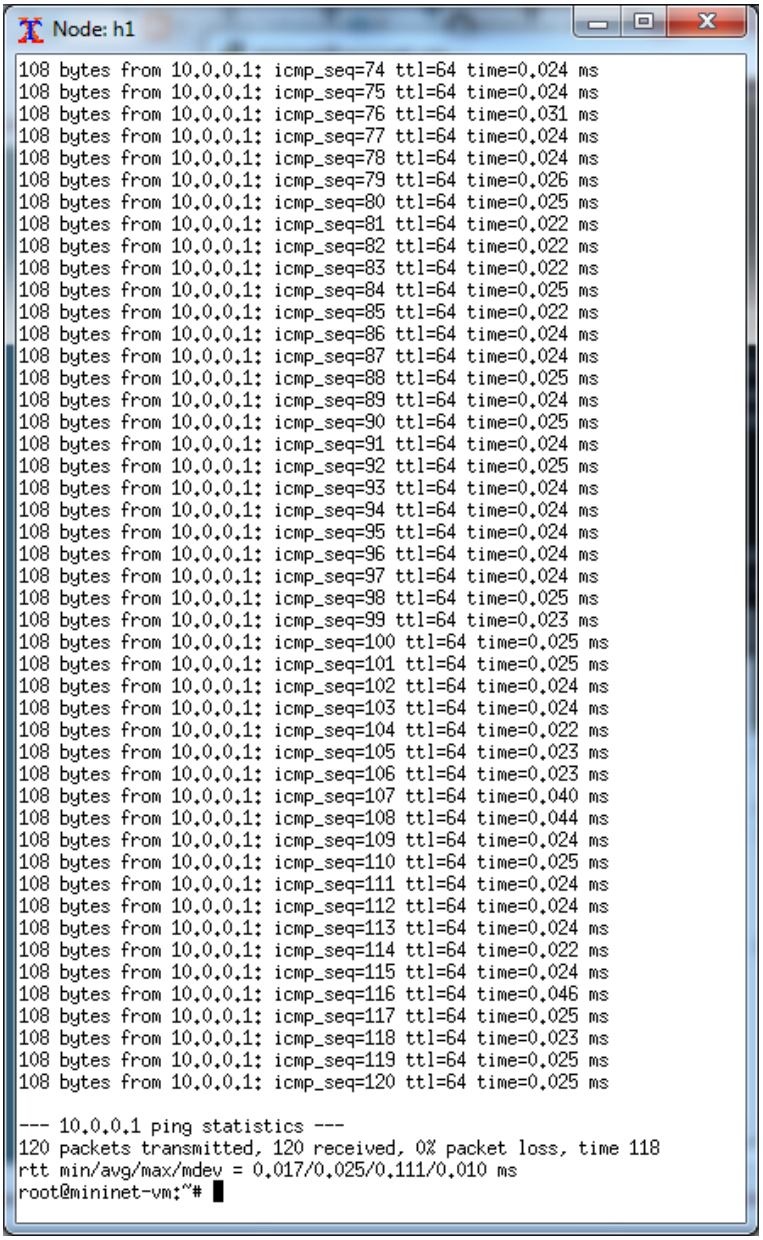
Εικόνα 5. 57 Αποτελέσματα iperf για κίνηση από h2->h1 & h4->h2. I



Εικόνα 5. 58 Αποτελέσματα iperf για κίνηση από h2->h1 & h4->h2. II

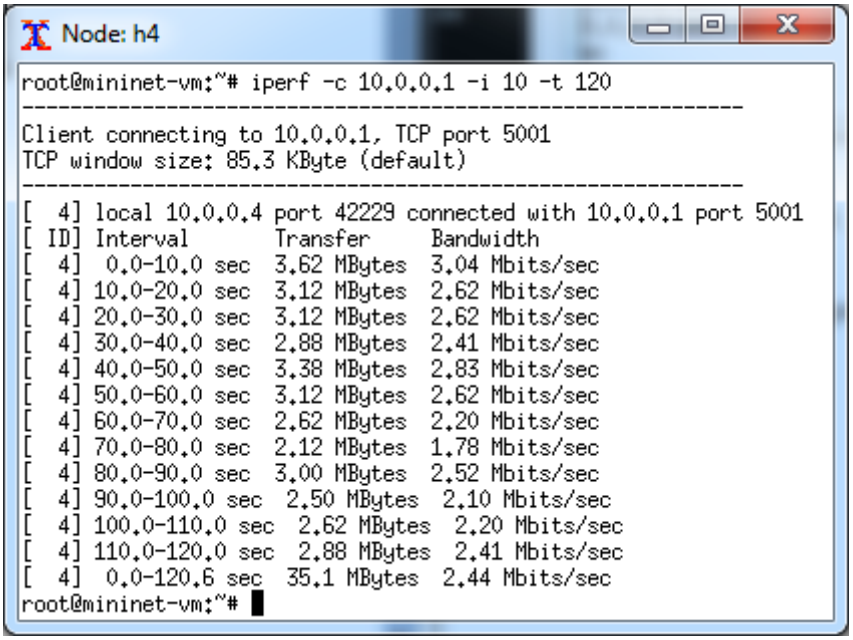
Παρατηρούμε από τις δυο παραπάνω εικόνες ότι δεν υπάρχει αξιόλογη μεταβολή στο εύρος ζώνης.

Εάν δώσουμε και την εντολή **# ping -c 120 -s 100 10.0.0.1** δηλαδή εκτελέσουμε 120 φορές ping και παράλληλα δημιουργούμε κίνηση με την εντολή iperf από 2 hosts ενώ έχω ένα host ως server λαμβάνουμε τα παρακάτω γραφήματα και συλλέγουμε τα παρακάτω στοιχεία:

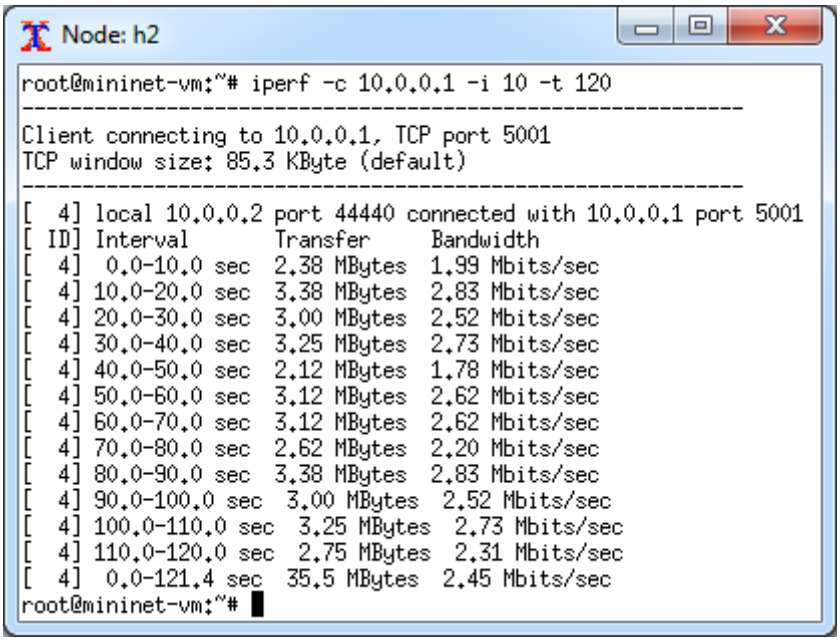


Εικόνα 5. 59 Αποτελέσματα ping ενώ έχουμε δώσει Iperf από 2 κόμβους προς 1 server.

Από την παραπάνω εικόνα 5.59 παρατηρούμε ότι ο χρόνος RTT είναι 10 msec ακριβώς όπως ορίσαμε στο Mininet κατά την δημιουργία του δικτύου.



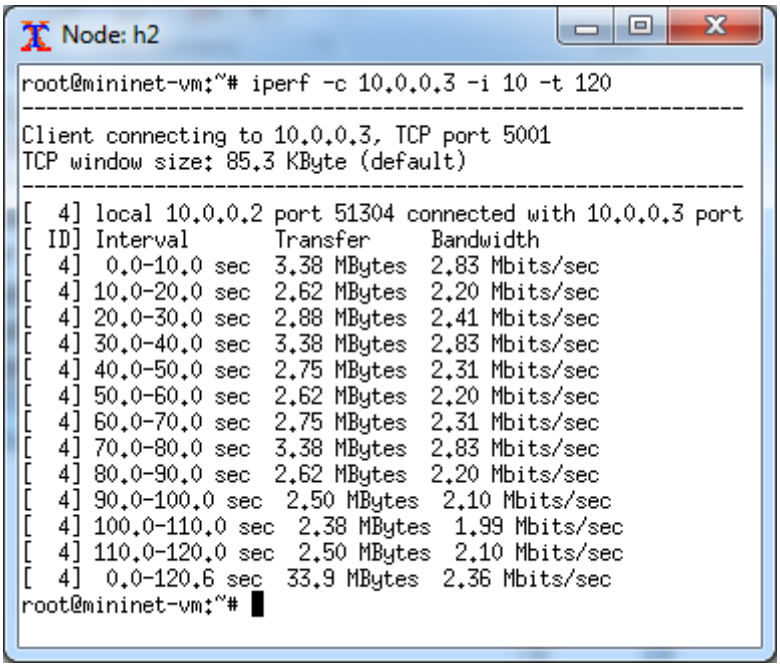
Εικόνα 5. 60 Αποτελέσματα iperf για κίνηση από h2->h1 & h4->h2.



Εικόνα 5. 61 Αποτελέσματα iperf για κίνηση από h2->h1 & h4->h2. II

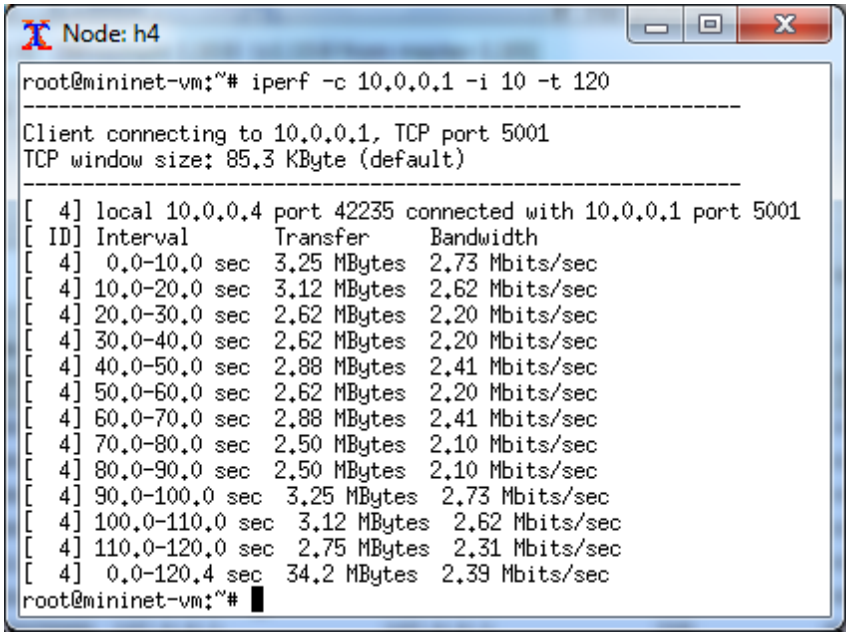
Παρατηρούμε ότι δεν υπάρχει αξιόλογη μεταβολή στο εύρος ζώνης του δικτύου και η καθυστέρηση του χρόνου RTT είναι η αναμενόμενη.

Παρακάτω εξετάζουμε μια παραλλαγή του προηγούμενου πειράματος στην οποία έχουμε ορίσει 2 servers. Για παράδειγμα εδώ ορίσαμε τον h3 και τον h1 ως server και 2 hosts: τον h2 και τον h4, και δημιουργούμε κίνηση μεταξύ τους. Παραθέτονται τα αποτελέσματα των εντολών δημιουργίας κίνησης καθώς και τα γραφήματα για το εύρος ζώνης και το RTT χρόνο που εξάγουμε με την βοήθεια του Wireshark.

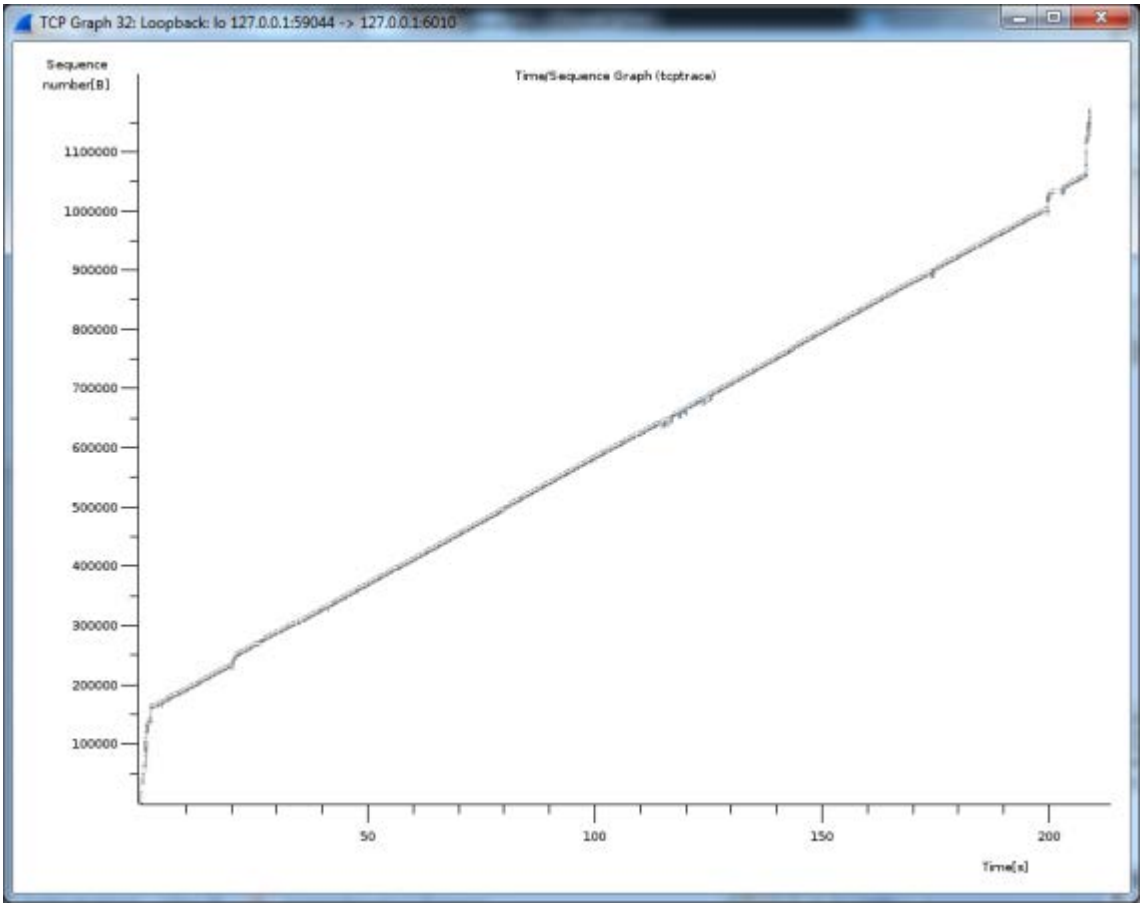


Εικόνα 5. 62 Αποτελέσματα iperf όταν δυο ζεύγη κόμβων δημιουργούν κίνηση στο δίκτυο.I

Από τα παραπάνω δεν διαπιστώνουμε κάποια ιδιαίτερη μεταβολή στη λειτουργία του δικτύου. Παρακάτω ακολουθούν τα γραφήματα που συλλέχθηκαν κατά την διάρκεια του πειράματος.

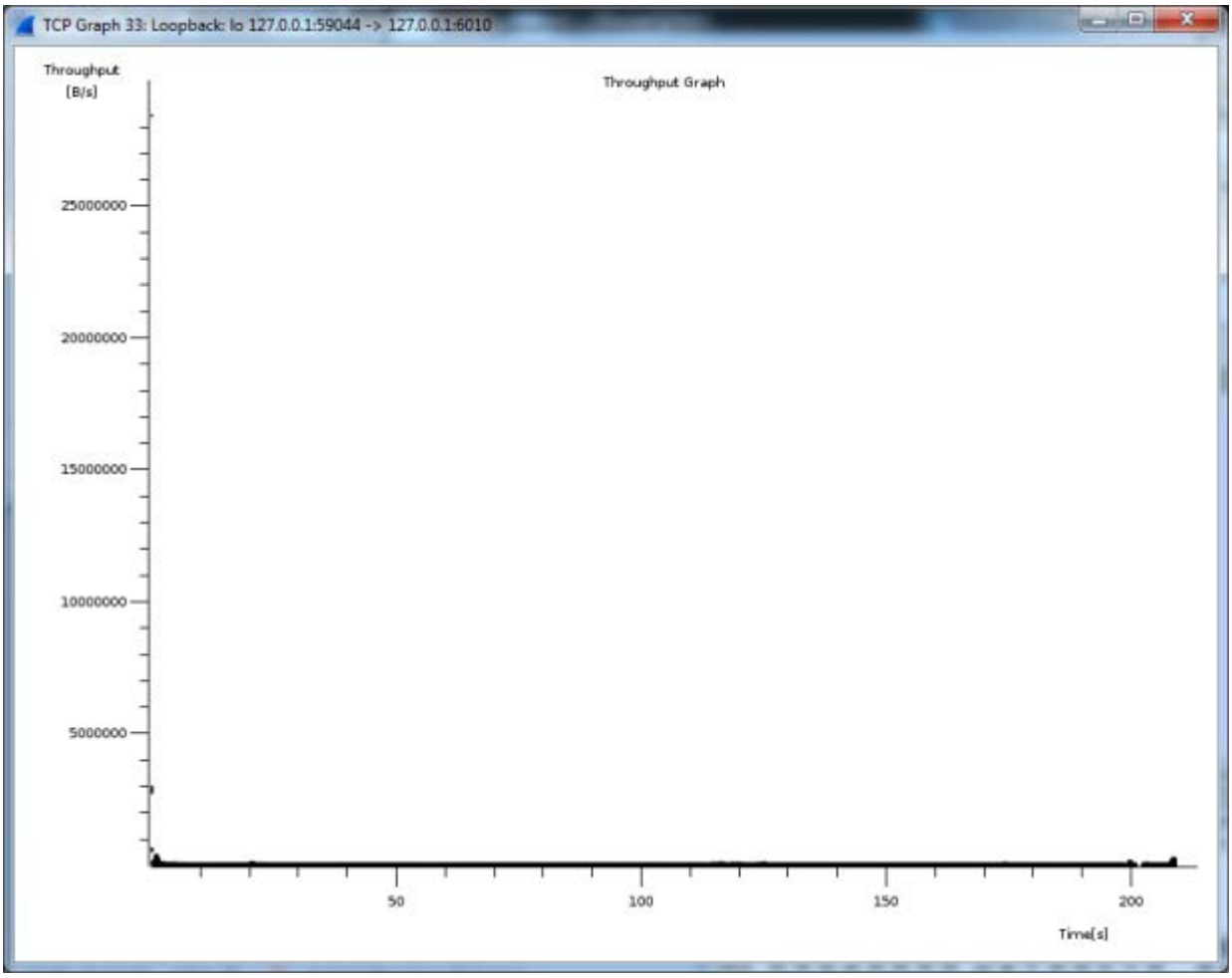


Εικόνα 5. 63 Αποτελέσματα iperf όταν δυο ζεύγη κόμβων δημιουργούν κίνηση στο δίκτυο.II

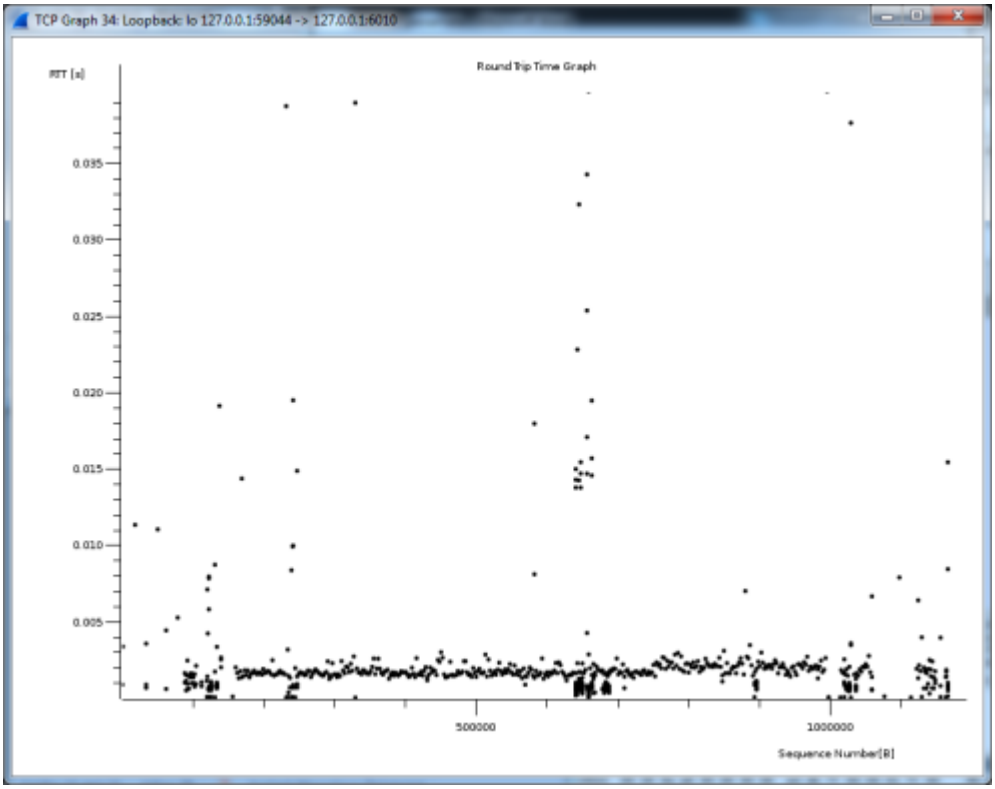


Γράφημα 5. 72 Time Sequence σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση & με απώλεια ενώ εκτελείται iperf σε δυο ζεύγη.

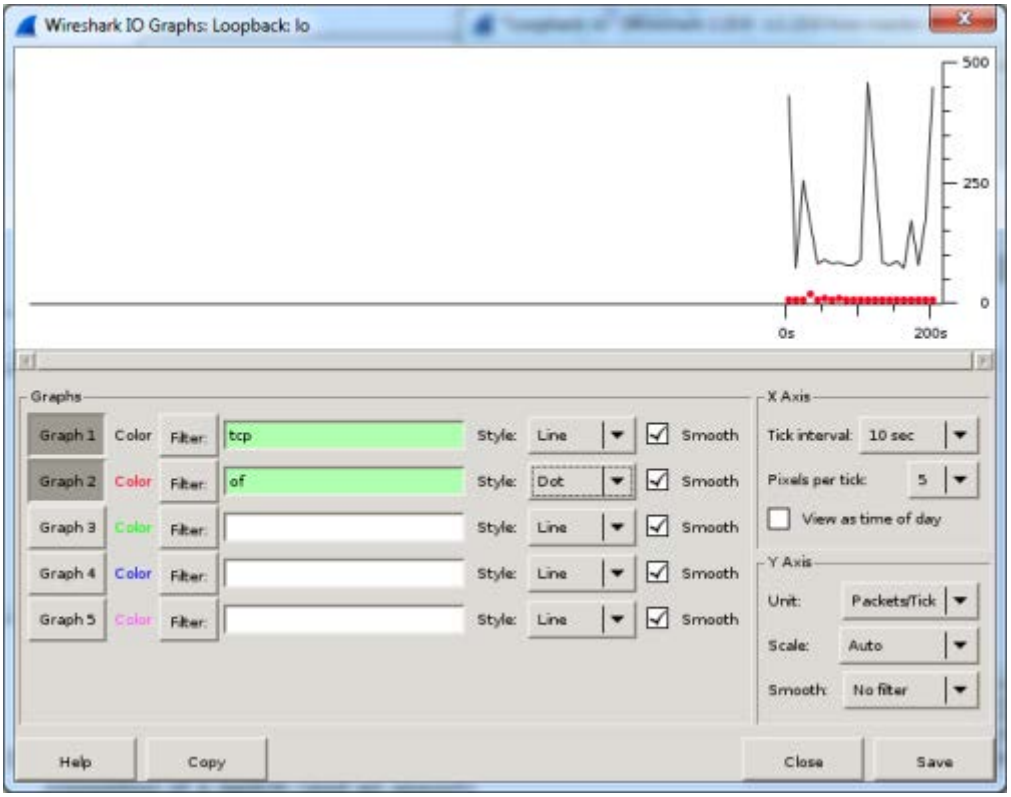
Η απόδοση του δικτύου είναι σε ικανοποιητικά επίπεδα. Το μόνο θέμα που εμφανίζει ως πρόβλημα το δίκτυο είναι η καθυστέρηση των 10 msec και η απώλεια των πακέτων που φαίνεται κυρίως όταν εκτελούμε την εντολή pingall στο δίκτυο.



Γράφημα 5. 73 Throughput σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση & με απώλεια ενώ εκτελείται iperf σε δυο ζεύγη.

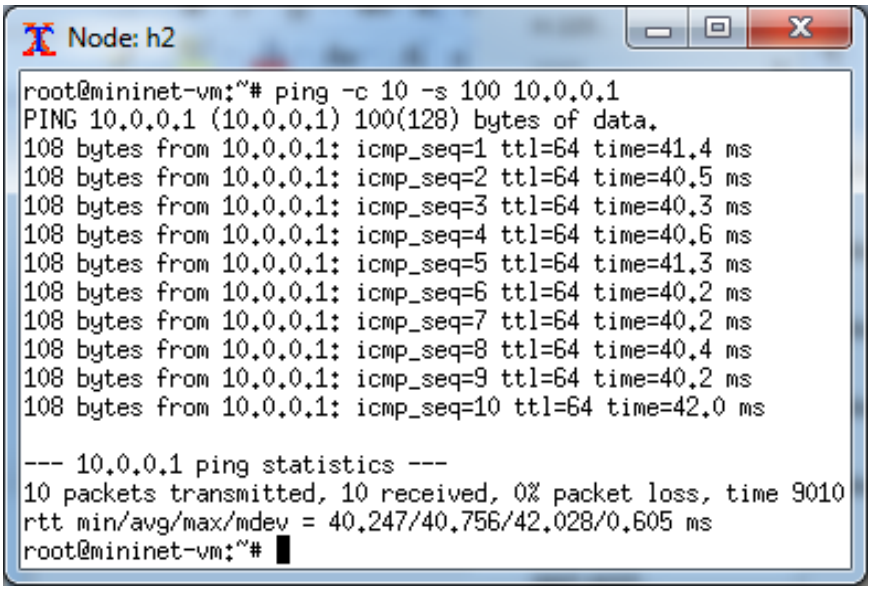


Γράφημα 5. 74 RTT σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση & με απώλεια ενώ εκτελείται iperf.

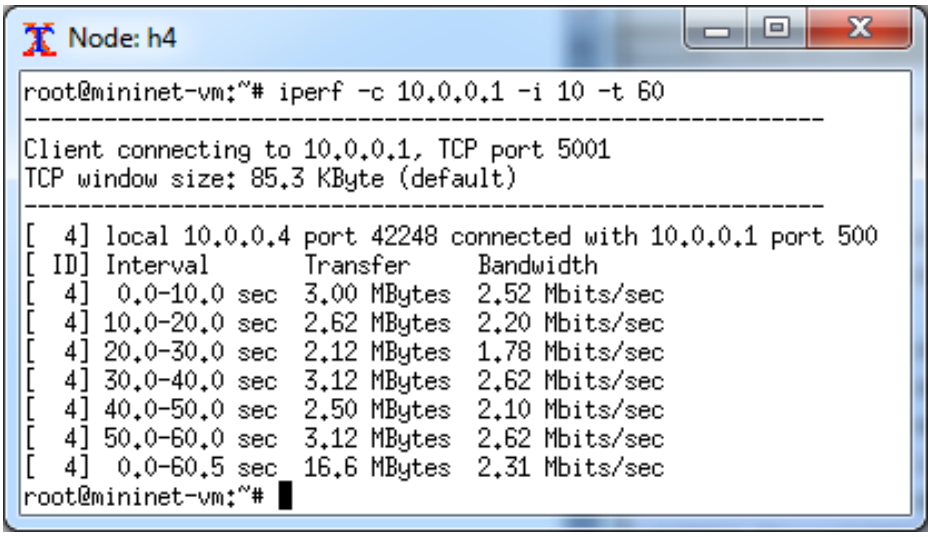


Γράφημα 5. 75 Γραφική παράσταση των πακέτων OF & TCP σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση & με απώλεια ενώ εκτελείται iperf σε 2 ζεύγη.

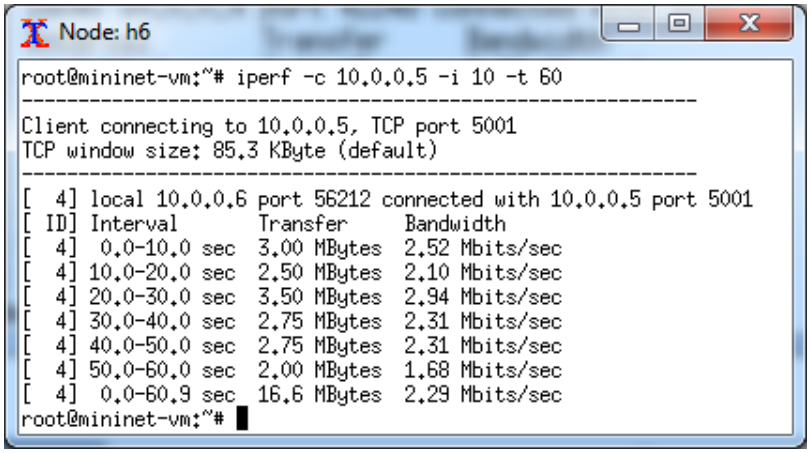
Το δίκτυο όπως φαίνεται και από τις προηγούμενες δεν εμφανίζει αξιόλογες μεταβολές. Μια εναλλαγή στο προηγούμενο πείραμα είναι εάν έχω 2 servers 2 hosts που δημιουργούν κίνηση και δώσω παράλληλα εντολή για 10 φορές ping **h2 # ping -c 10 -s 100 10.0.0.1**. Τα ζεύγη που δημιουργούμε είναι οι κόμβοι h4->h1 και h6 ->h5 ενώ η εντολή ping δίνεται στον κόμβο h2 όπως φανερώνεται στο σχήμα που ακολουθεί.



Εικόνα 5. 64 ελέσματα Ping όταν έχω κίνηση από h4->h1 και h6 ->h5 ενώ η εντολή ping δίνεται από τον κόμβο h2 ->h1.

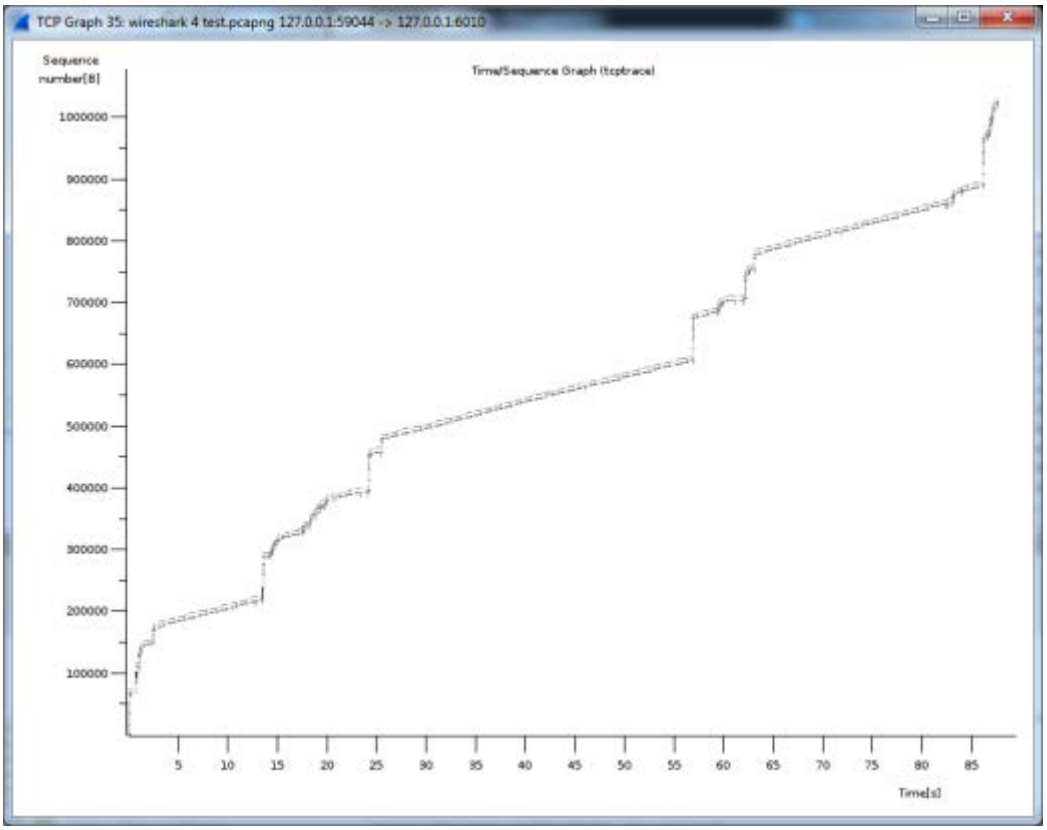


Εικόνα 5. 65 Αποτελέσματα Iperf όταν έχω κίνηση από h4->h1 και h6 ->h5 ενώ η εντολή ping δίνεται από τον κόμβο h2 ->h1. I

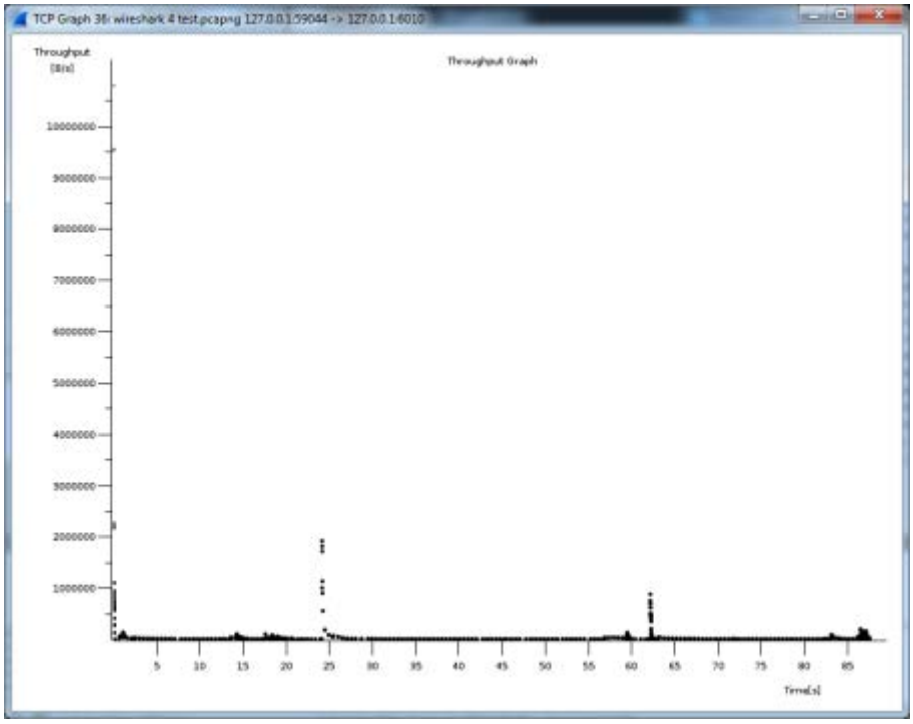


Εικόνα 5. 66 Αποτελέσματα Ping όταν έχω κίνηση από h4->h1 και h6 ->h5 ενώ η εντολή ping δίνεται από τον κόμβο h2 ->h1. II

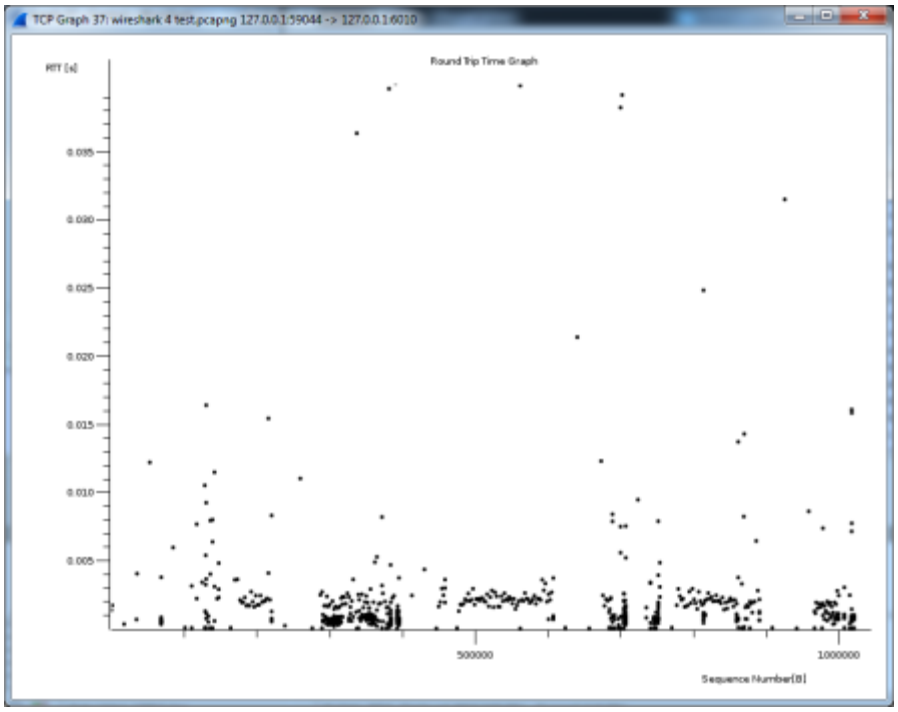
Τα γραφήματα που προκύπτουν από τα δεδομένα που συλλέχθηκαν κατά την διάρκεια του πειράματος ακολουθούν:



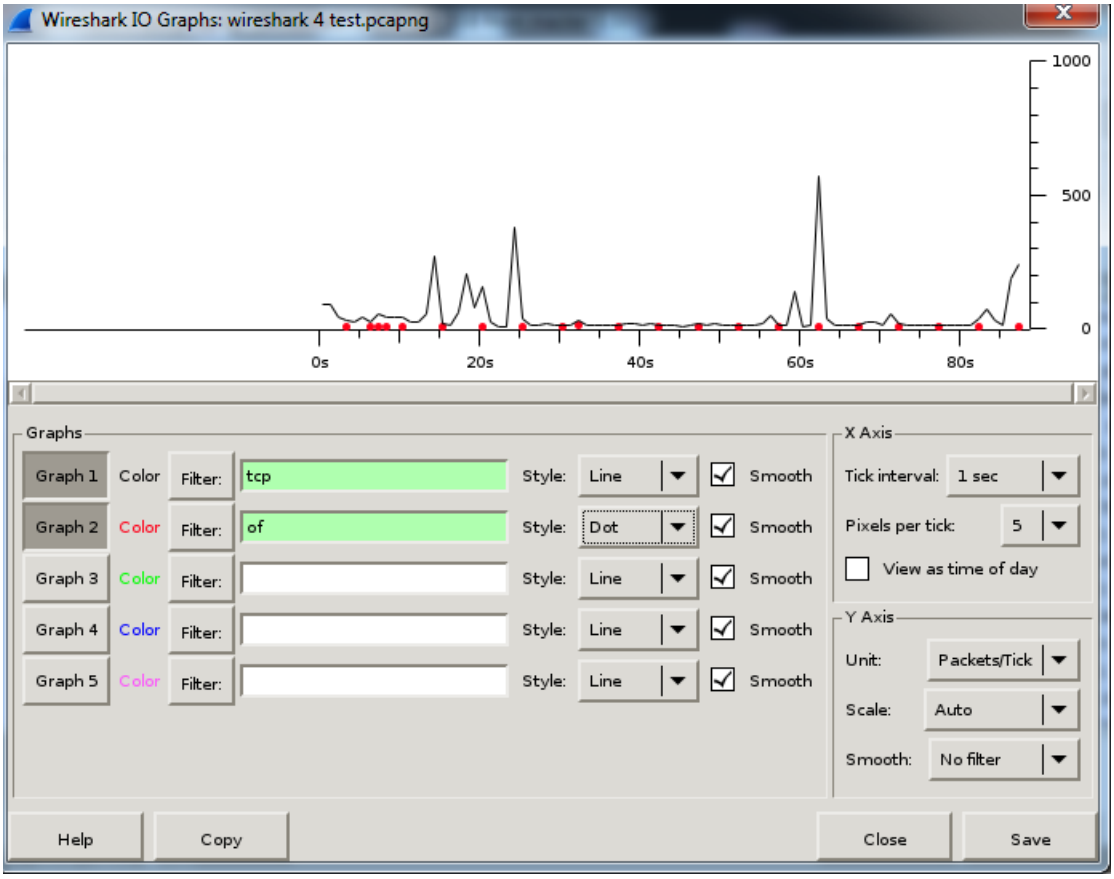
Γράφημα 5. 76 Time Sequence σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση & με απώλεια ενώ εκτελείται iperf και ping.



Γράφημα 5. 77 Throughput σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση & με απώλεια ενώ εκτελείται iperf και ping.



Γράφημα 5. 78 RTT σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση & με απώλεια ενώ εκτελείται iperf και ping.



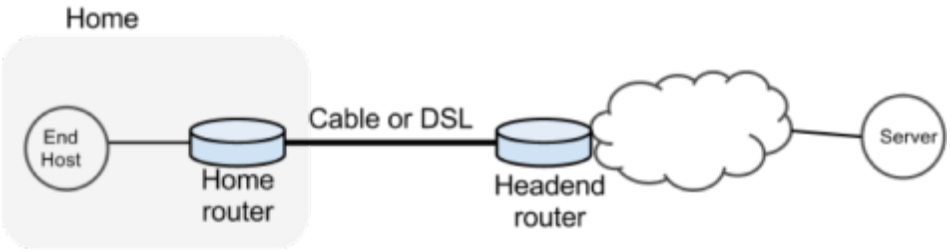
Γράφημα 5. 79 Γραφική παράσταση των πακέτων OF & TCP σε απλό δίκτυο 6 κόμβων και 1 μεταγωγέα με καθυστέρηση & με απώλεια ενώ εκτελείται iperf - ping .

5.6 Bufferbloat

Στο παρακάτω πείραμα θα εξετάσουμε την δυναμική του TCP σε οικιακά δίκτυα. Στο παρακάτω σχήμα παρουσιάζεται ένα "τυπικό" οικιακό δίκτυο με ένα δρομολογητή συνδεδεμένο στο τέλος της υποδοχής. Το Home Router συνδέεται μέσω καλωδίου ή DSL σε ένα router Headend στο γραφείο του παρόχου πρόσβασης Διαδικτύου. Εμείς θα μελετήσουμε τι συμβαίνει όταν έχουμε λήψη δεδομένων από έναν απομακρυσμένο διακομιστή στον τελικό κόμβο στο οικιακό δίκτυο. Σε ένα πραγματικό δίκτυο είναι δύσκολο να μετρηθεί το cwnd (επειδή είναι πληροφορία που κατέχει ο Server) και η κατάληψη προσωρινής μνήμης (επειδή είναι πληροφορία που κατέχει ο router). Για να γίνει πιο εύκολη η

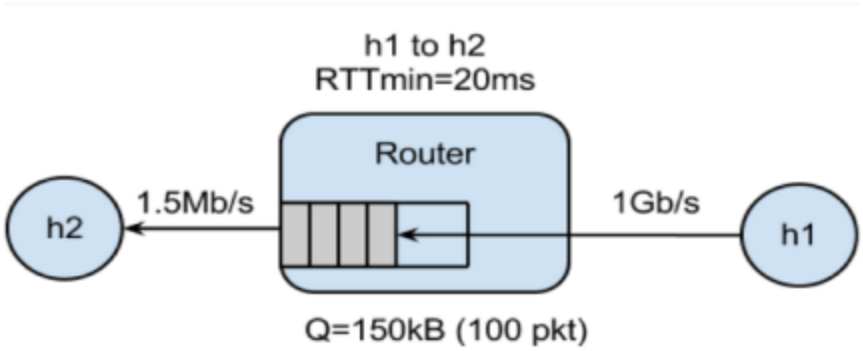
μέτρηση αυτών των πληροφοριών, θα εξομοιώσουμε το δίκτυο με την βοήθεια του Mininet. [50][30]

Μέσα από αυτήν την διαδικασία θα μάθουμε για τη δυναμική του cwnd και το βαθμό πληρότητας του buffer σε ένα «πραγματικό» δίκτυο. Επίσης θα κατανοήσουμε γιατί οι πολύ μεγάλοι buffers των δρομολογητών μπορούν να οδηγήσουν σε κακή απόδοση τα οικιακά δίκτυα. Το πρόβλημα αυτό συχνά αποκαλείται "Bufferbloat." [26]



Εικόνα 5. 67 Εικόνα του δικτύου που θα προσομοιώσουμε.

Το δίκτυο που θα δημιουργήσουμε στο Mininet περιγράφεται από τις δυο αυτές εικόνες. Στην δεύτερη εικόνα φαίνονται εκτός από την συνδεσιμότητα χαρακτηριστικά των συνδέσεων όπως το εύρος ζώνης, το μέγεθος του buffer καθώς και οι χρόνοι RTT.



Εικόνα 5. 68 Τεχνικά χαρακτηριστικά του δικτύου που θα προσομοιώσουμε

Εκκινούμε λοιπόν για άλλη μια φορά το Mininet και καθαρίζουμε με την εντολή **$ mn -c** ότι μπορεί να έχει αποθηκευτεί κατά την διάρκεια προηγούμενων πειραμάτων ώστε να αποκτήσουμε "καθαρά" πειραματικά αποτελέσματα. Δίνουμε την εντολή:

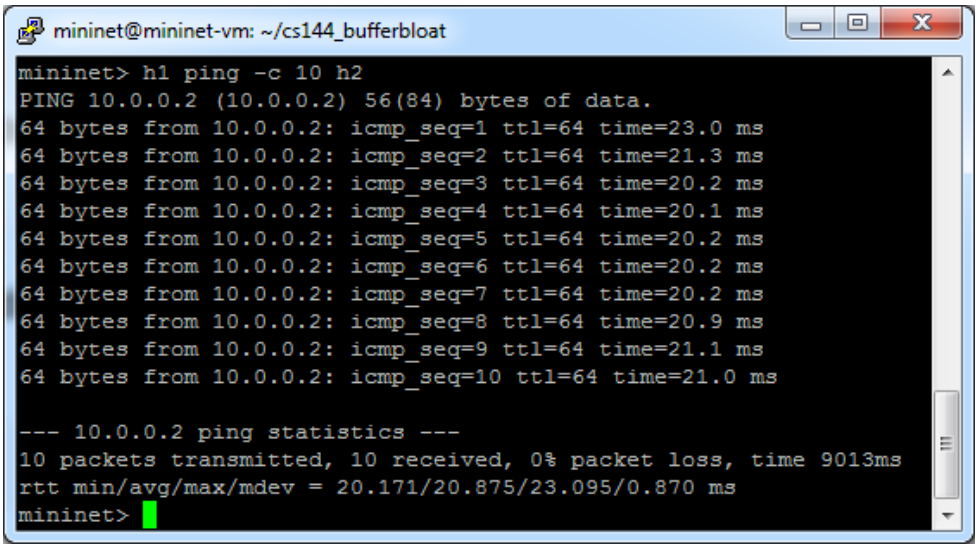
$ git clone https://bitbucket.org/huangty/cs144_bufferbloat.git

Ορίζουμε το directory στο οποίο επιθυμούμε να εργαστούμε με την εντολή

$ cd cs144_bufferbloat/ και στη συνέχεια τρέχουμε την τοπολογία με την εντολή **$ sudo ./run.sh**

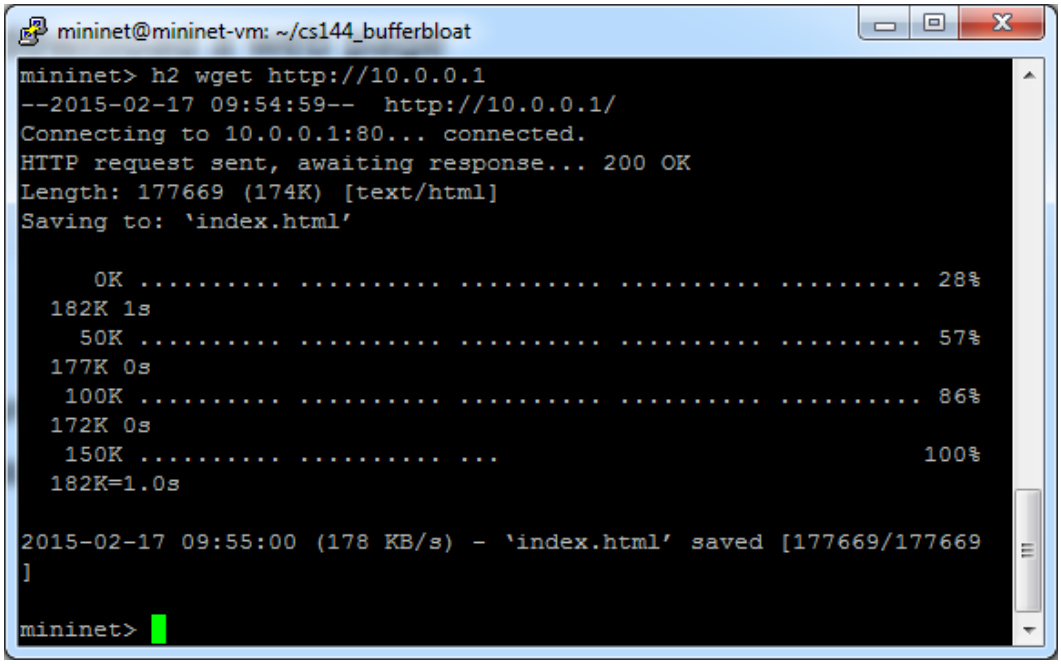
Παρατηρούμε από το ping τεστ που διεξάγεται μεταξύ των κόμβων του δικτύου υπάρχει επικοινωνία μεταξύ των.

Ουσιαστικά μαζί με την δημιουργία της συγκεκριμένης τοπολογίας στο Mininet διεξάγεται και ένα ping τεστ και παρατηρούμε ότι η αρχική καθυστέρηση μεταξύ δύο hosts του δικτύου είναι 20ms. Μετά από αυτό το βήμα είμαστε έτοιμοι να μετρήσουμε ξανά την καθυστέρηση μεταξύ H1 και H2 δίνοντας την εντολή mininet> **h1 ping -c 10 h2**



Εικόνα 5. 69 Αποτελέσματα του ping από τον h1 στον h2.

Παρακάτω δοκιμάζουμε τους χρόνους που απαιτούνται για να κατεβάσουμε μια ιστοσελίδα από τον κόμβο H1. Η εντολή που δίνουμε στο Mininet είναι mininet> **h2 wget http://10.0.0.1** Τα αποτελέσματα αυτής της εντολής παρουσιάζονται στην παρακάτω εικόνα:



Εικόνα 5. 70 Απόκριση του h2 κόμβου bufferbloat δικτύου στην εντολή wget.

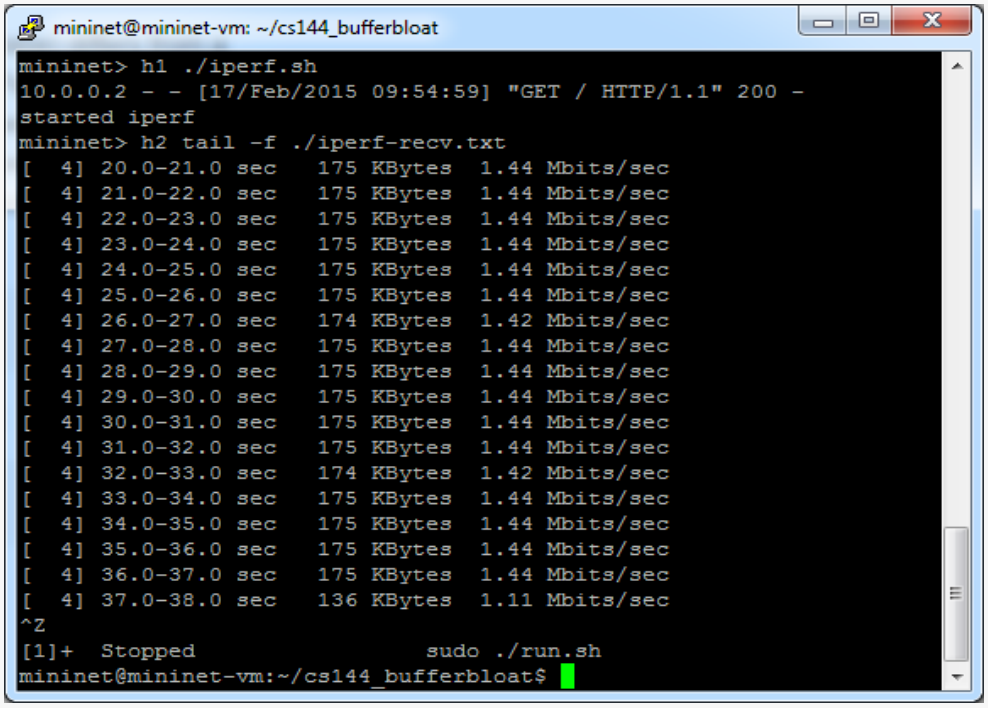
5.6.1 Δημιουργία ροής video

Για να δούμε πώς κινείται η δυναμική μιας μεγάλης ροής (το οποίο εισέρχεται στην AIMD φάση) διαφέρει από μια μικρή ροή (η οποία δεν εξέρχεται ποτέ από την αργή εκκίνηση), θα επαναλάβουμε το δεύτερο κομμάτι του πειράματος για μια "ροή βίντεο". Έτσι αντί να παρακολουθούμε πραγματικά βίντεο στον υπολογιστή μας, θα δημιουργήσουμε μια μεγάλης διάρκειας TCP σύνδεση υψηλής ταχύτητας ώστε να μιμηθούμε τα αποτελέσματα που θα είχαμε από μια μεγάλη ροή βίντεο. Μπορούμε να δημιουργήσουμε στον εξομοιωτή μεγάλες ροές

κίνησης, χρησιμοποιώντας την εντολή Iperf, και συγκεκριμένα θα πρέπει να εκτελέσουμε την εντολή : mininet> **h1 ./iperf.sh**

Μπορούμε να δούμε το throughput της TCP ροής από τον H1 στον H2 δίνοντας την εντολή: mininet> **h2 tail -f ./iperf-recv.txt**

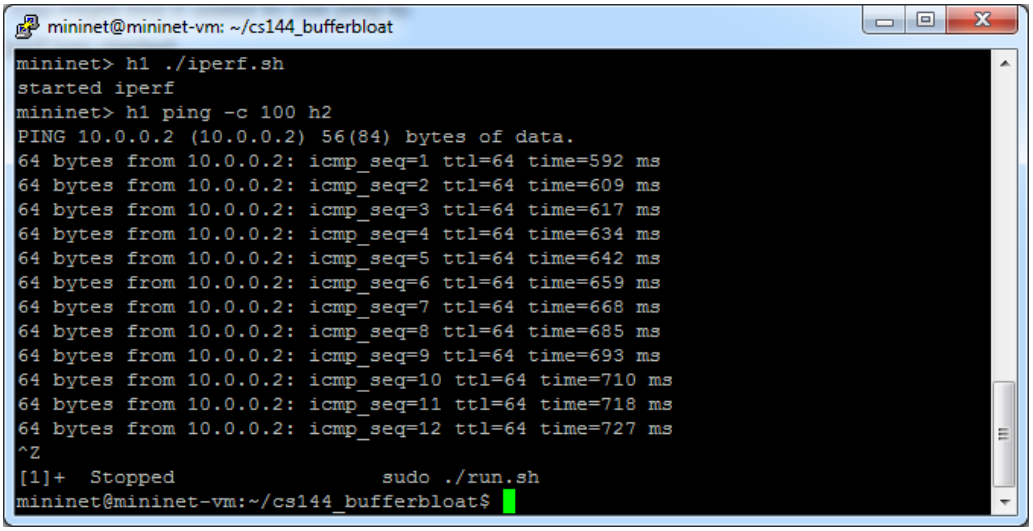
Μπορούμε να τερματίσουμε οποιαδήποτε στιγμή την παρατήρηση των αποτελεσμάτων του throughput πατώντας CTRL-C.



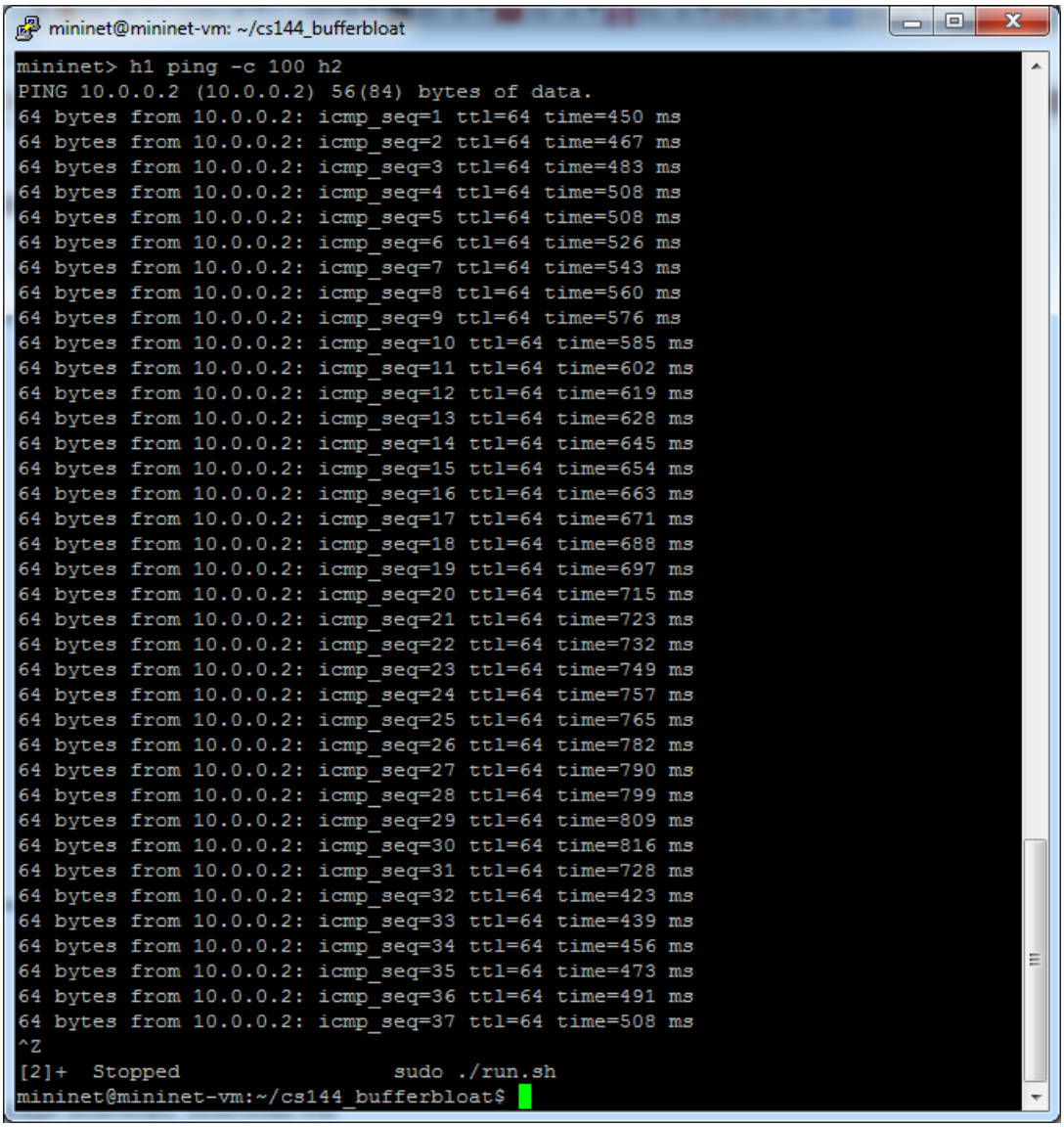
Εικόνα 5. 71 Δημιουργία ροής βίντεο στο Mininet.

Επίδραση της ροής βίντεο στο χρόνο μετ΄επιστροφή των πακέτων.

Εάν δώσουμε τώρα την εντολή: mininet> **h1 ping -c 100 h2** παρατηρούμε τα παρακάτω αποτελέσματα στην οθόνη του Mininet.



Εικόνα 5. 72 Επίδραση ροής βίντεο στα αποτελέσματα της ping εντολής. I

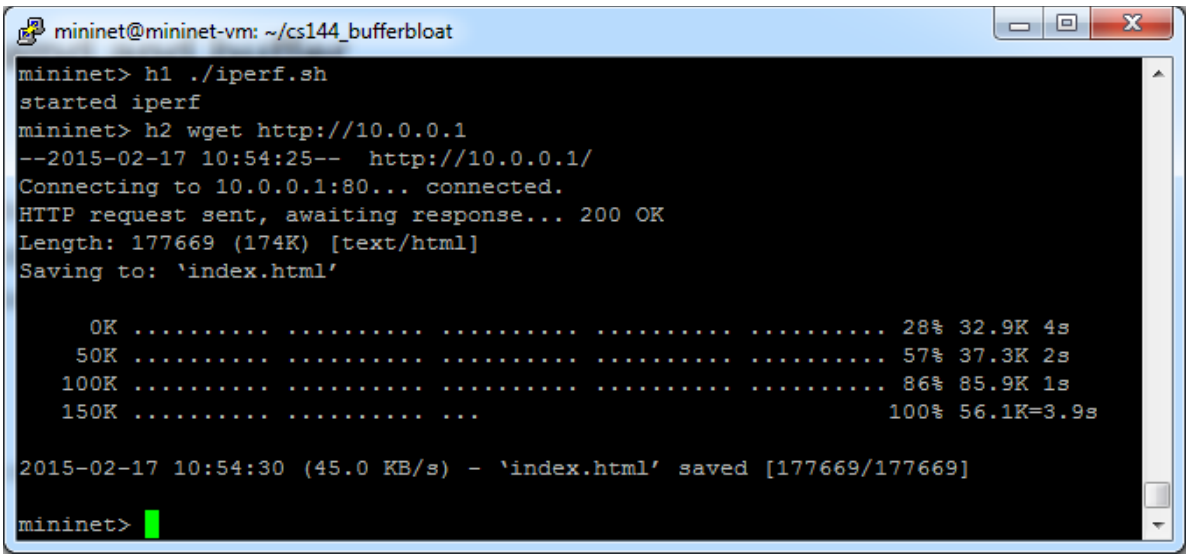


Εικόνα 5. 73 Επίδραση ροής βίντεο στα αποτελέσματα της ping εντολής. II

Επίδραση της ροής iperf (ροή βίντεο) στο κατέβασμα σελίδων web.

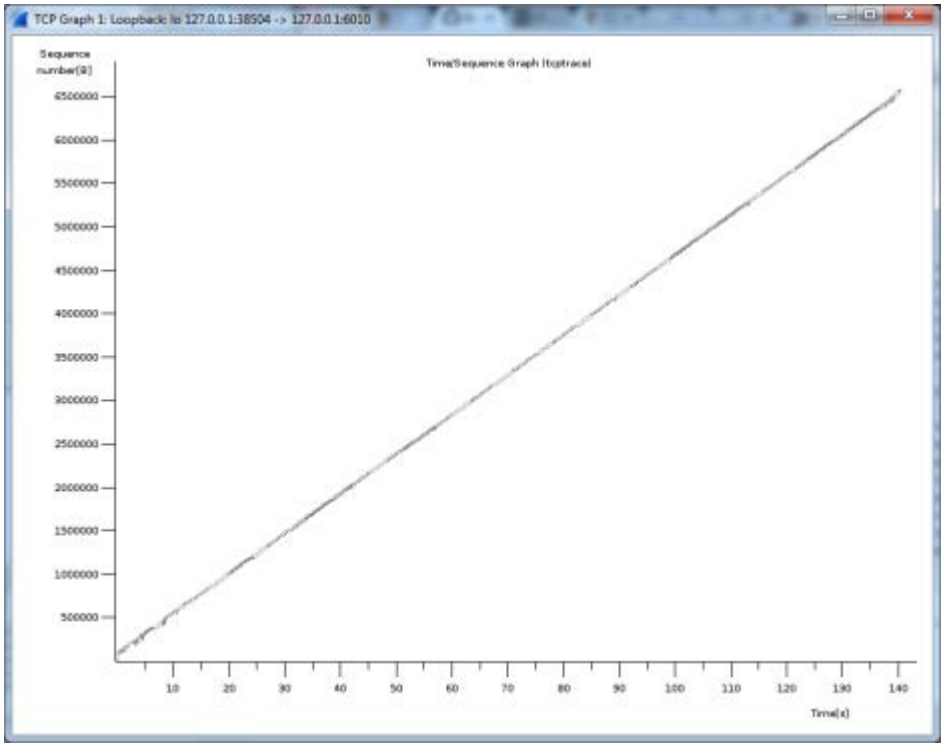
Για να διαπιστώσουμε πως επηρεάζει η iperf ροή το κατέβασμα της web σελίδας μπορούμε να κατεβάσουμε την σελίδα ξανά ενώ τρέχει η εντολή iperf.

```
mininet> h2 wget http://10.0.0.1
```

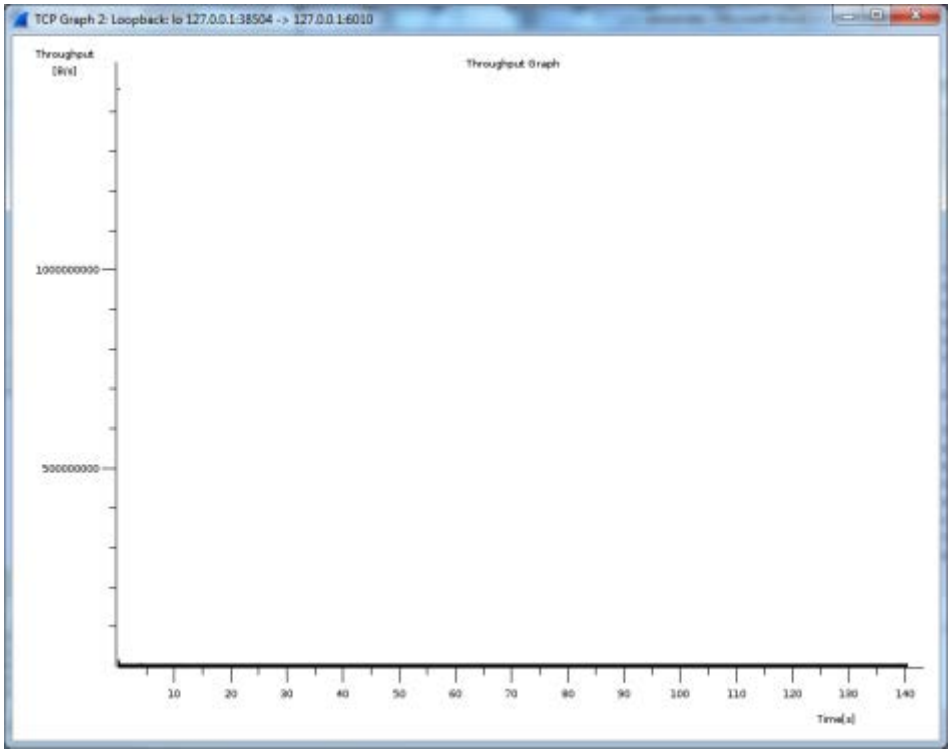


Εικόνα 5. 74 Αποτελέσματα wget όταν στο δίκτυο υπάρχει ροή βίντεο.

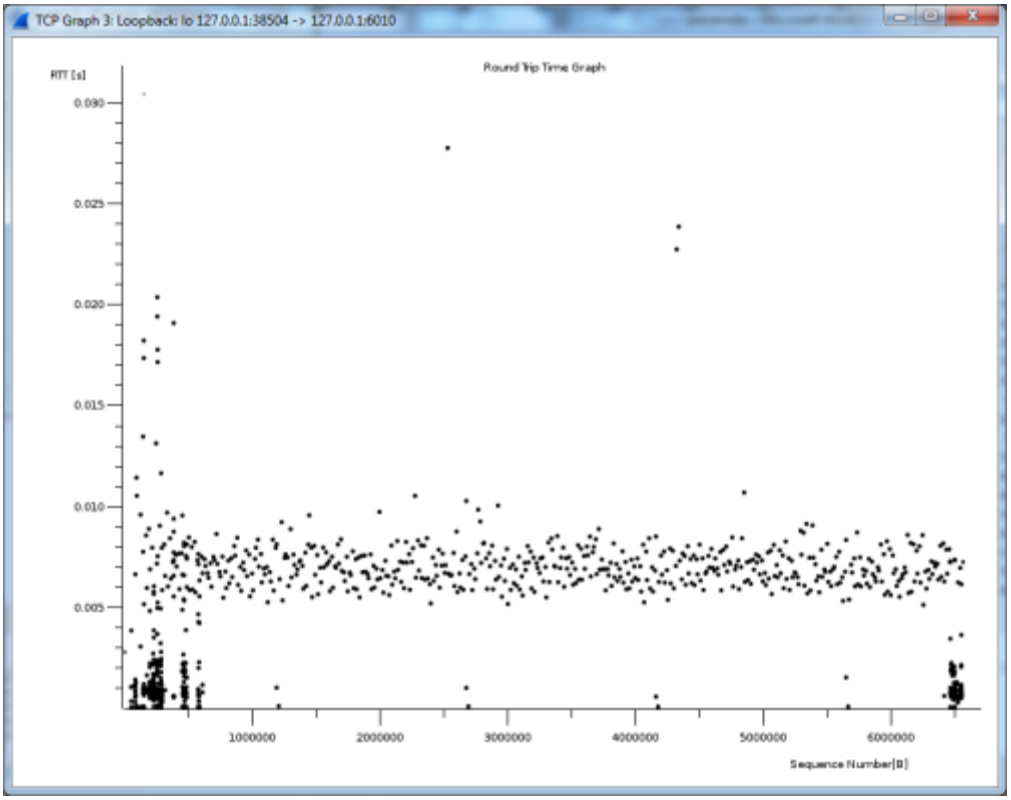
Παρατηρούμε ότι ο χρόνος που απαιτείται πλέον είναι κοντά στα 4 sec ενώ στην προηγούμενη περίπτωση που ζητήσαμε την σελίδα χωρίς να τρέχει η εντολή iperf ήταν μόνο 1 sec. Στα γραφήματα που συλλέξαμε βλέπουμε ότι το δίκτυο αντιδρά ομαλά ενώ δεν παρουσιάζει προβλήματα και ακόμα και οι χρόνοι RTT κυμαίνονται πλην ελάχιστων εξαιρέσεων γύρω στα 0,010sec.



Γράφημα 5. 80 Time Sequence στο δίκτυο ενώ εκτελείται ροή βίντεο



Γράφημα 5. 81 Throughput στο δίκτυο ενώ εκτελείται ροή βίντεο.



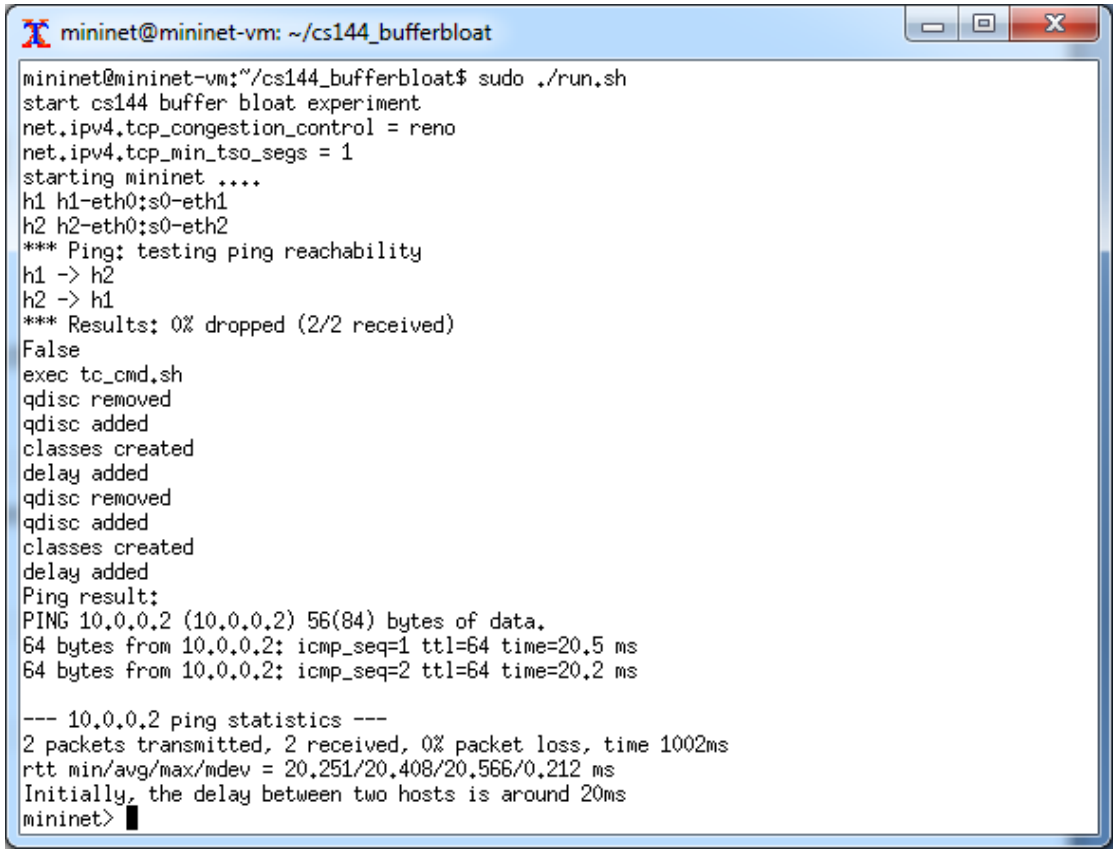
Γράφημα 5. 82 RTT στο δίκτυο ενώ εκτελείται ροή βίντεο

5.6.2 Μέτρηση του πραγματικού cwnd και του βαθμού χρήσης του buffer.

Με την βοήθεια του Mininet θα μετρήσουμε το cwnd και τον βαθμό χρήσης της χωρητικότητας του buffer. Χρησιμοποιώντας ένα script θα συλλέξουμε τις τιμές του cwnd και το βαθμό κατάληψης της προσωρινής μνήμης σε αρχεία. Θα τρέξουμε ξανά τα πειράματα και θα σχεδιάσουμε γραφήματα από τις πραγματικές τιμές.

Πραγματοποιούμε επανεκκίνηση του Mininet και ξανατρέχουμε τα προηγούμενα πειράματα. Δίνουμε λοιπόν τις εντολές :

```
mininet> exit

bash# sudo ./run.sh
```

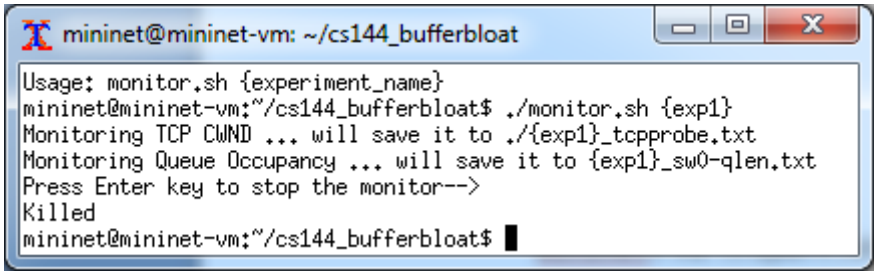


Εικόνα 5. 75 Δημιουργία του bufferbloat δικτύου (xterm).

Ξαναπηγαίνουμε στο **cs144_bufferbloat** directory σε ένα άλλο παράθυρο και πληκτρολογούμε την παρακάτω εντολή:

```
# ./monitor.sh <EXP_1>
```

Δεν ανησυχούμε αν δούμε κάποια αναφορά σφάλματος, έχει να κάνει με το γεγονός ότι το συγκεκριμένο στοιχείο δεν έχει εισαχθεί στο Mininet ξανά από την εγκατάσταση του. Το Mininet μας δίνει πληροφορίες για το που θα αποθηκευτεί η συλλογή των αποτελεσμάτων του πειράματος.



Εικόνα 5. 76 Εντολή καταγραφής.

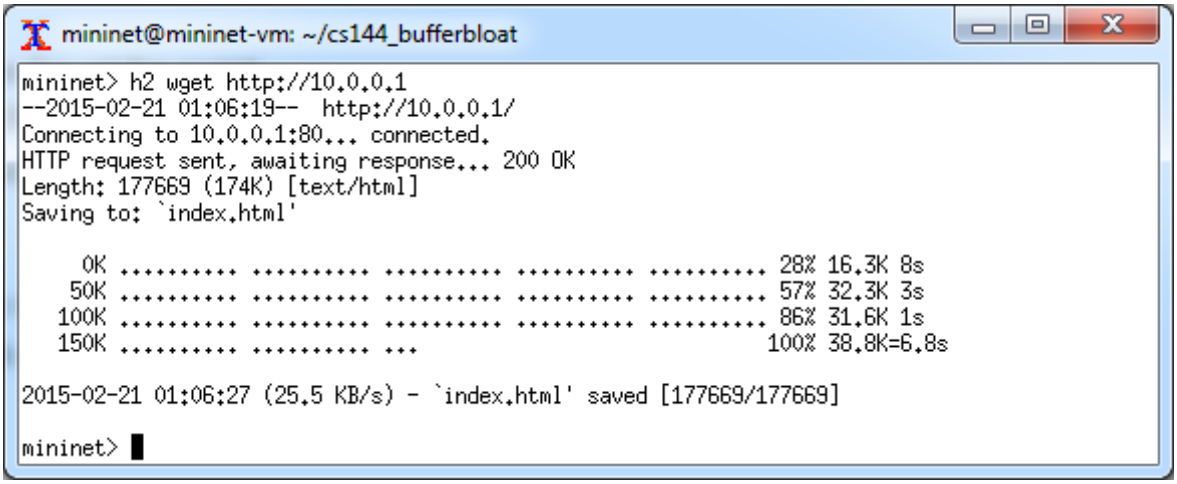
Για να σταματήσουμε το πείραμα απλά πατάμε enter.

Σε ένα άλλο παράθυρο δίνουμε την εντολή:

```
mininet> h1 ./iperf.sh
```

Την αφήνουμε να εκτελεσθεί για 70 περίπου δευτερόλεπτα. Δίνουμε ξανά την παρακάτω εντολή:

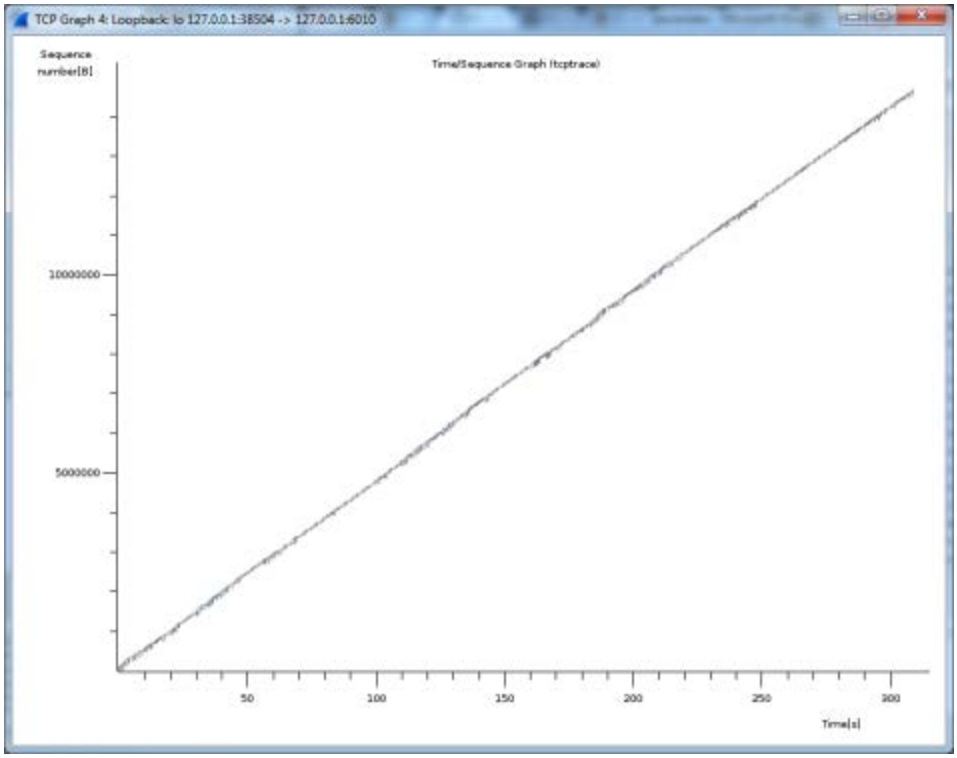
```
mininet> h2 wget http://10.0.0.1
```



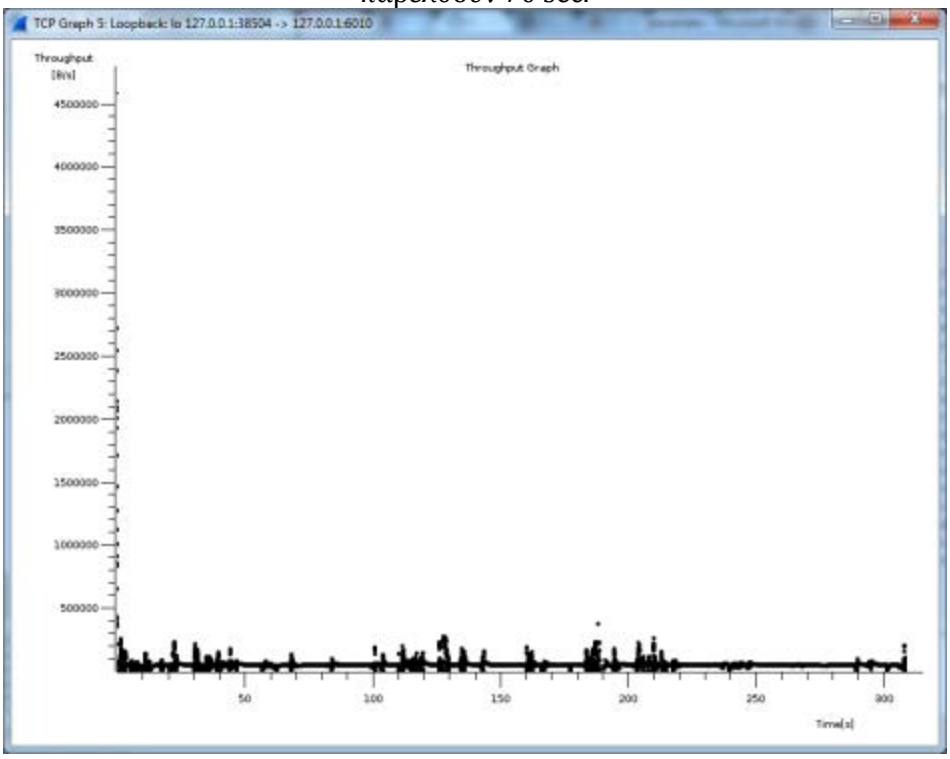
Εικόνα 5. 77 Αποτελέσματα wget όταν στο δίκτυο υπάρχει ροή βίντεο και παρέλθουν 70 sec.

Παρατηρούμε ότι πλέον ο χρόνος για να φορτωθεί η σελίδα είναι 6,8 δευτερόλεπτα. Στις γραφικές παραστάσεις που δημιουργήθηκαν κατά την διάρκεια του πειράματος βλέπουμε την αύξηση στο throughput καθώς και το ότι

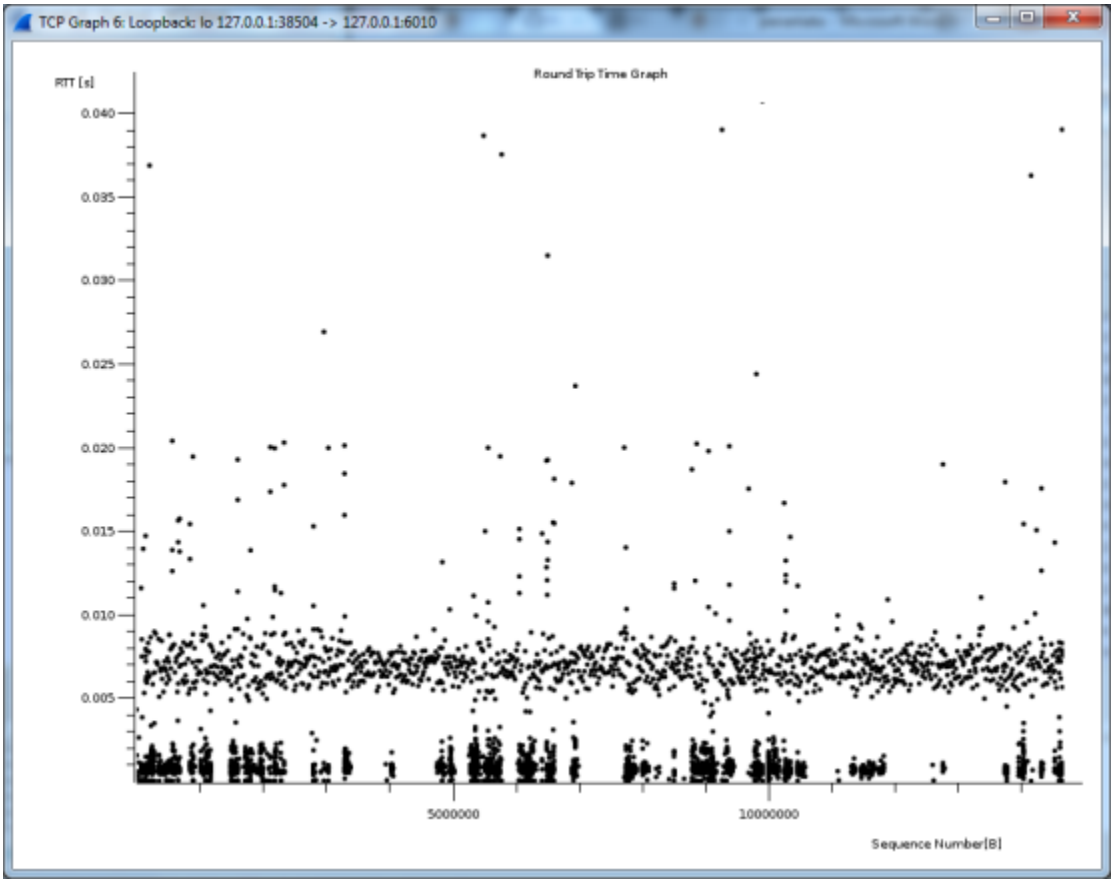
ο RTT χρόνος έχει λίγο αυξηθεί αν το συγκρίνουμε με τα αποτελέσματα του πρώτου πειράματος. Αυτό είναι αποτέλεσμα της εντολής που τρέξαμε για περισσότερο από 70 δευτερόλεπτα και δημιούργησε αυξημένη κίνηση στο δίκτυο μας.



Γράφημα 5. 83 Time Sequence στο δίκτυο bufferbloat με ροή βίντεο και αφού παρέλθουν 70 sec.



Γράφημα 5. 84 Throughput στο δίκτυο bufferbloat με ροή βίντεο και αφού παρέλθουν 70 sec



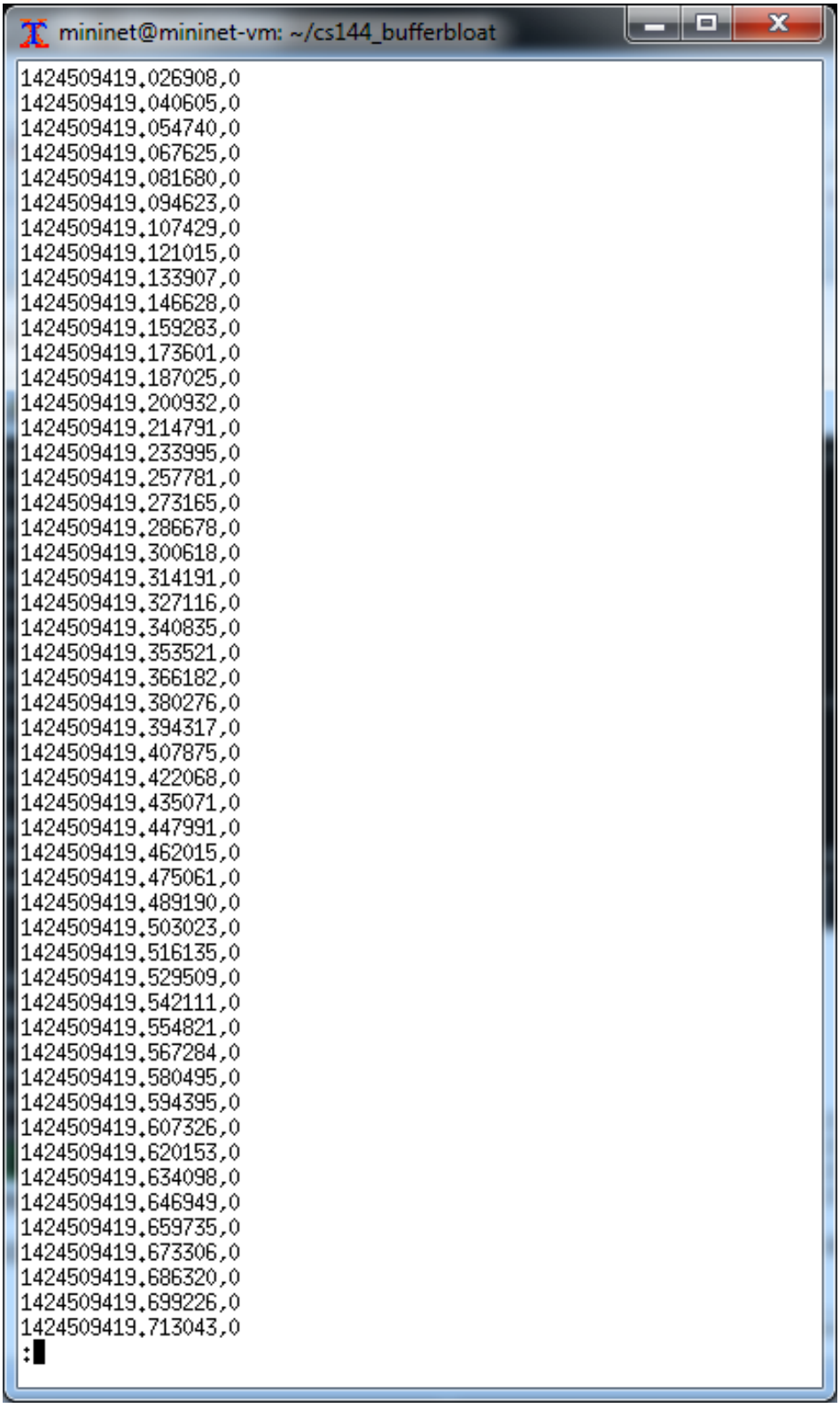
Γράφημα 5. 85 RTT στο δίκτυο bufferbloat με ροή βίντεο και αφού παρέλθουν 70 sec.

Για να δημιουργήσουμε τα σχεδιαγράμματα TCP cwnd και βαθμού χρήσης της ουράς από τα δεδομένα που συλλέχτηκαν κατά την διάρκεια του πειράματος, δίνουμε την εντολή:

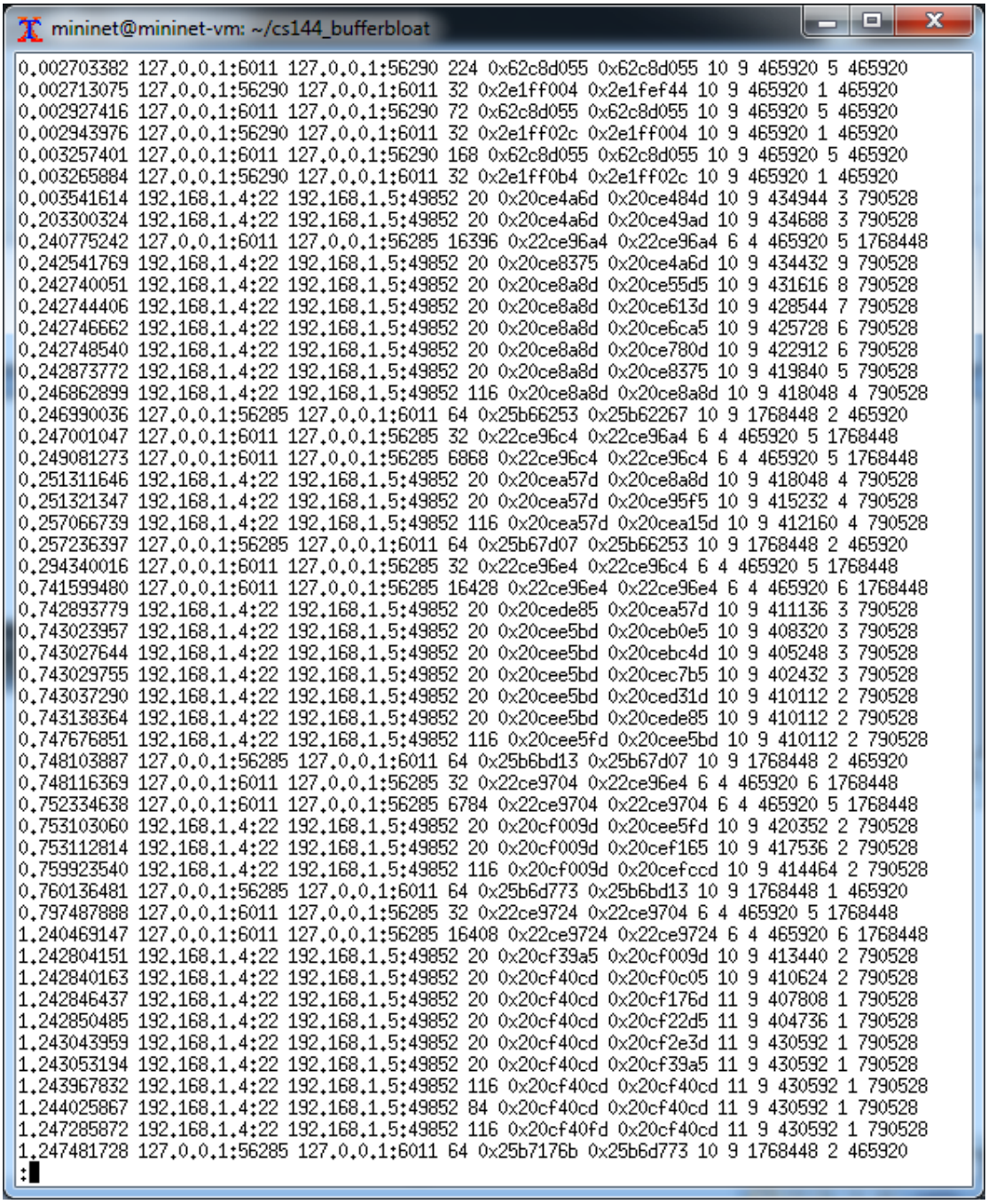
bash# ./plot_figures.sh <EXP_1>

Το script θα δημιουργήσει και ένα webserver στο μηχάνημα που φιλοξενεί τον εξομοιωτή ώστε να μπορούμε να δούμε τα αποτελέσματα μέσω μιας url διεύθυνσης.

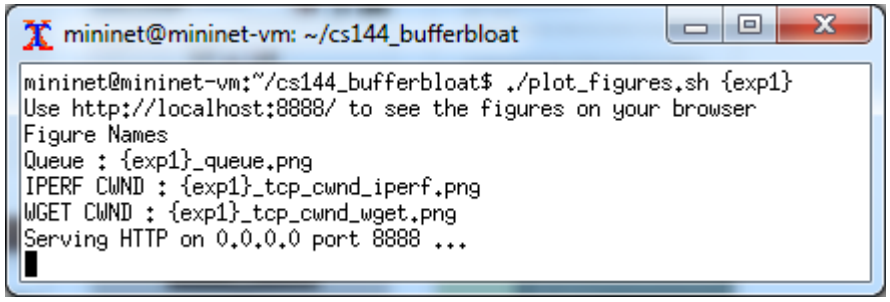
Μέχρι τώρα συνειδητοποιήσαμε ότι το buffer στο router είναι τόσο μεγάλο που όταν γεμίζει με πακέτα από την κίνηση iperf καθυστερεί την ροή του wget. Στην συνέχεια θα διαπιστώσουμε τρόπους ώστε να αρθεί το συγκεκριμένο πρόβλημα. Στις ακόλουθες εικόνες φαίνονται τα εξαγόμενα αποτελέσματα.



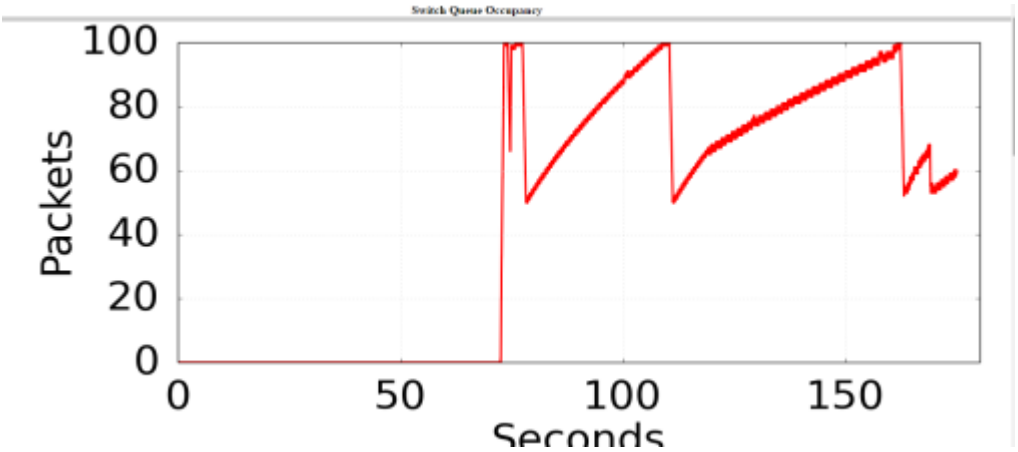
Εικόνα 5. 78 Αποτελέσματα εξομοίωσης από τα οποία θα δημιουργηθούν τα γραφήματα TCP cwnd II.



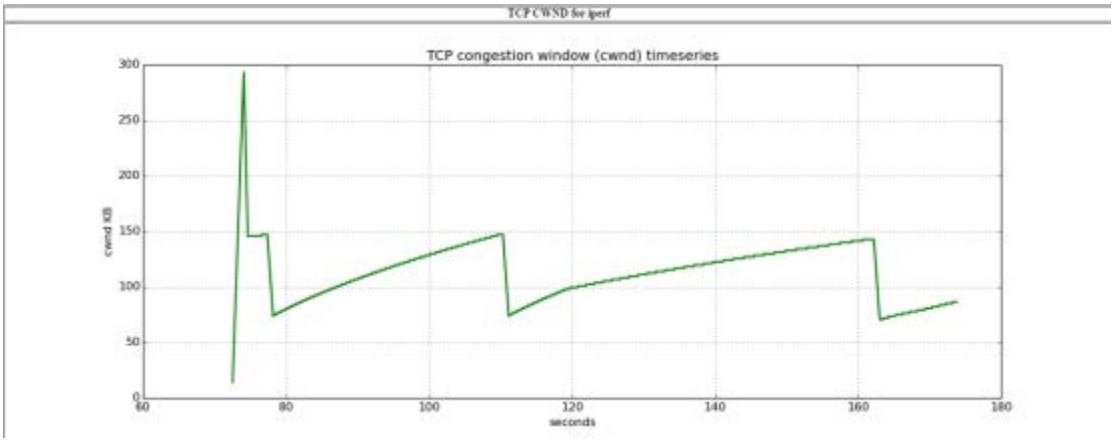
Εικόνα 5. 79 Αποτελέσματα εξομοίωσης από τα οποία θα δημιουργηθούν τα γραφήματα TCP cwnd II



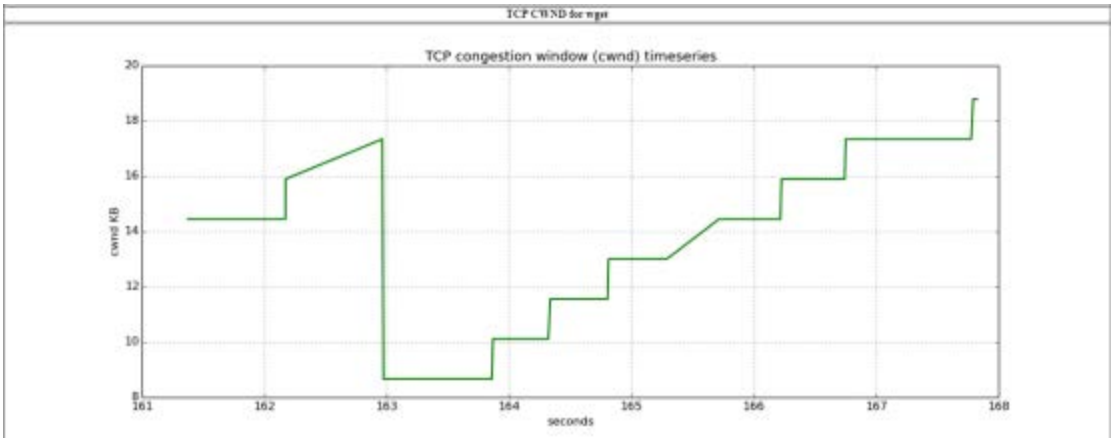
Εικόνα 5. 80 Εντολή δημιουργίας γραφημάτων στο Mininet.



Γράφημα 5. 86 Πληρότητα ουράς στον Μεταγωγέα.



Γράφημα 5. 87 TCP Congestion Window για το iperf.



Γράφημα 5. 88 TCP Congestion Window για το wget.

5.6.3 Ορισμός Μικρότερου Buffer

Σταματάμε το Mininet και το ξανά εκκινούμε αλλά αυτή την φορά δημιουργούμε buffers με μέγεθος 20 πακέτων από εκεί που στο πρώτο πείραμα ήταν μεγέθους 100 πακέτων.

prompt> sudo ./run-minq.sh

Δίνουμε πάλι την εντολή της καταγραφής των δεδομένων:

prompt> sudo ./monitor.sh <EXP_NAME>

Επαναλαμβάνουμε τα τελευταία βήματα του προηγούμενου πειράματος:

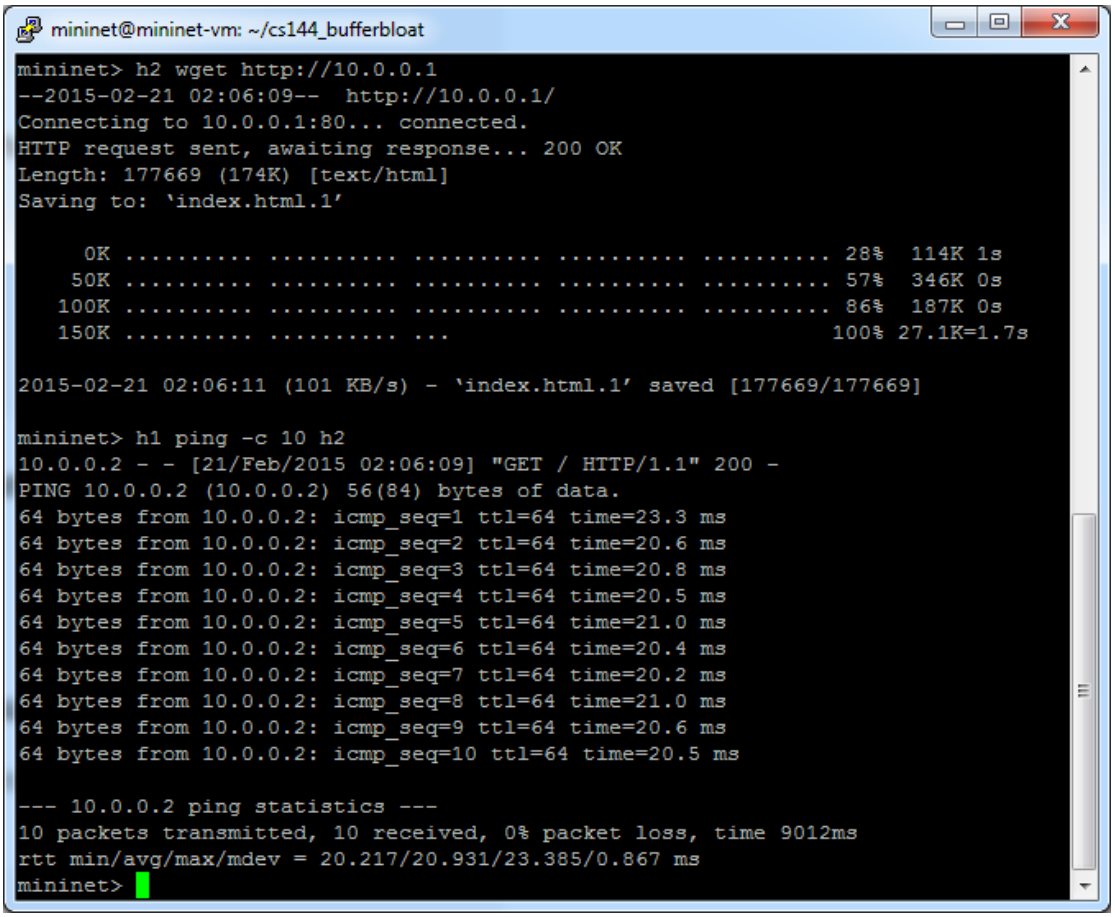
mininet> **h2 wget http://10.0.0.1**

mininet> **h1 ping -c 10 h2**

mininet> **h1 ./iperf.sh**

mininet> **h1 ping -c 30 h2**

mininet> **h2 wget http://10.0.0.1**

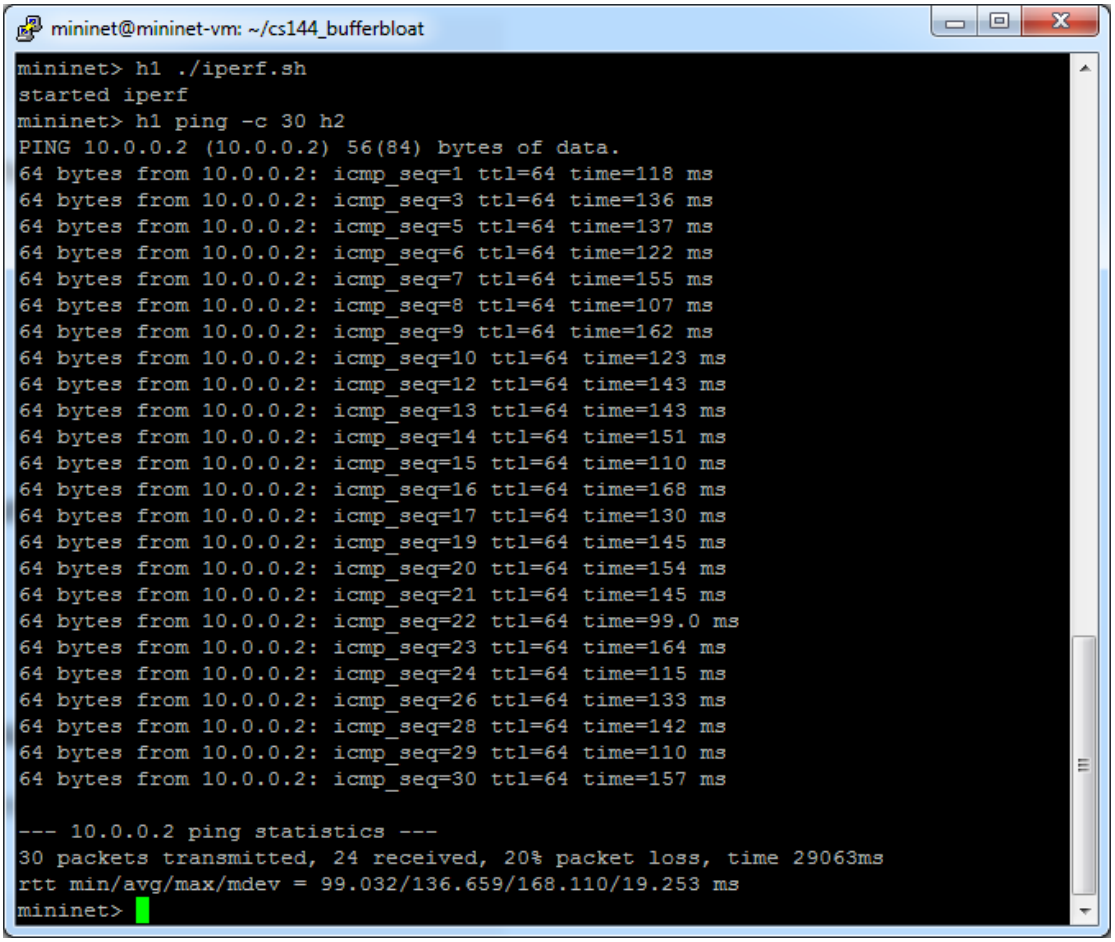


Εικόνα 5. 81 Wget & Ping στο Bufferbloat δικτύο με buffer 20 πακέτων.

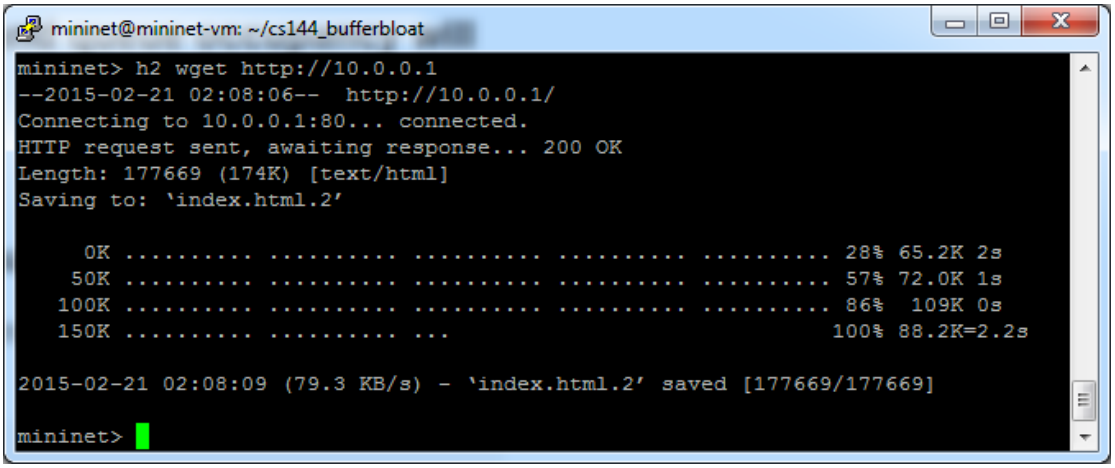
Τα αποτελέσματα που συλλέχτηκαν ακολουθούν. Παρατηρούμε ότι ο χρόνος που απαιτείται πλέον για να κατέβει η σελίδα μειώθηκε κατά 1/2 σχεδόν.

Σε θεωρητικό επίπεδο γνωρίζουμε ότι οι δρομολογητές του δικτύου περιέχουν buffers για να κρατήσουν τα πακέτα κατά τη διάρκεια των περιόδων συμφόρησης. Σήμερα, το μέγεθος των buffers προσδιορίζεται από τη δυναμική του αλγορίθμου ελέγχου συμφόρησης του TCP. Ειδικότερα, ο στόχος είναι όταν ένα σύνδεσμος είναι κορεσμένος, να είναι απασχολημένος στο 100% του χρόνου. Αυτό είναι ισοδύναμο με το να ειπωθεί ότι το buffer δεν είναι ποτέ άδειο. Ένα ευρέως μεταχειρισμένως κανόνας - rule-of-thumb ορίζει ότι κάθε link χρειάζεται ένα buffer του μέγεθος B = RTT × C, όπου RTT είναι ο μέσος χρόνος

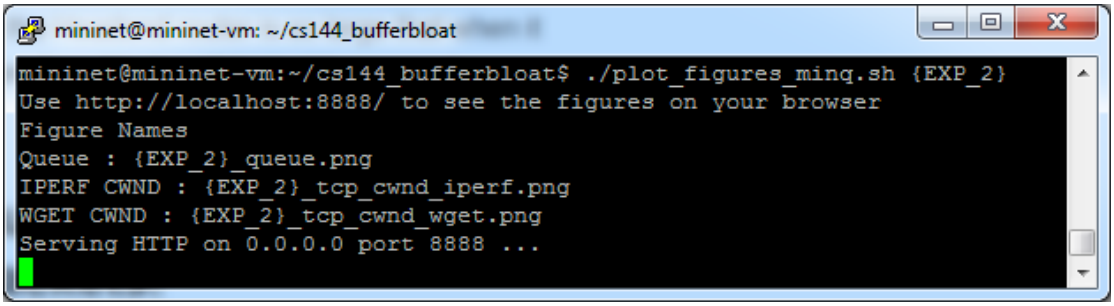
μετ 'επιστροφής της ροής που διέρχεται κατά μήκος της σύνδεσης, και C είναι ο ρυθμός δεδομένων. [1]



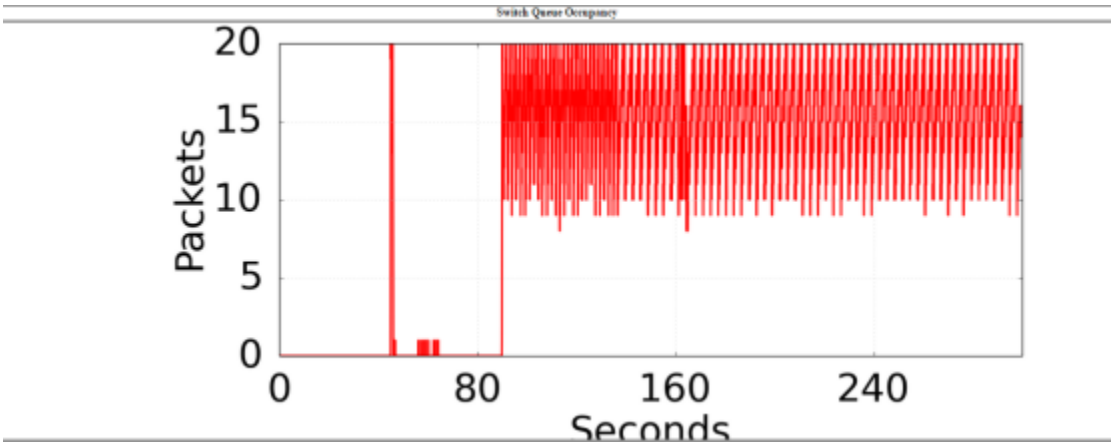
Εικόνα 5. 82 Ping & Iperf στο δίκτυο με buffer 20 πακέτων.



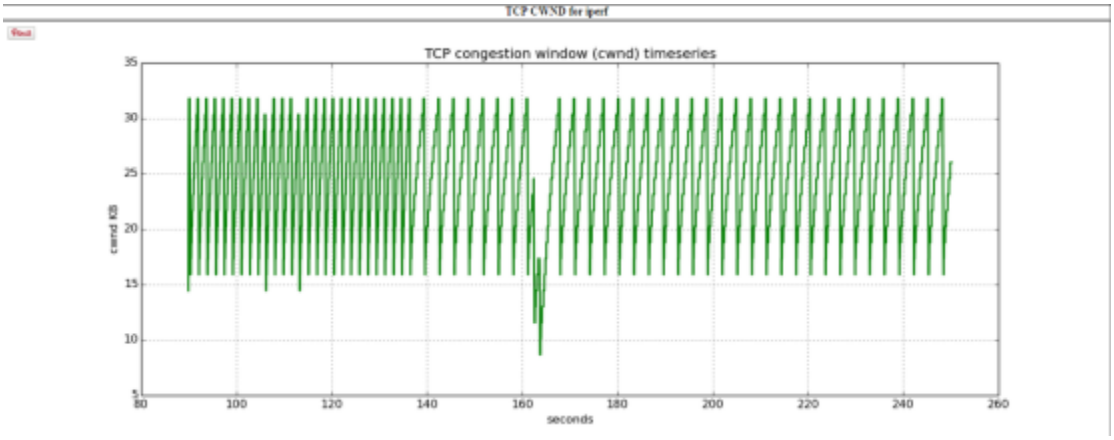
Εικόνα 5. 83 Wget στο Bufferbloat δικτύο με buffer 20 πακέτων



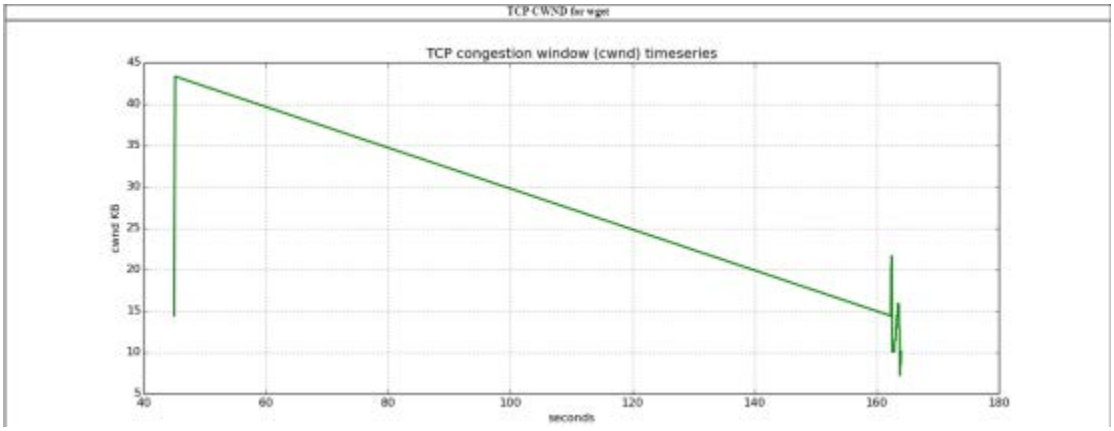
Εικόνα 5. 84 Εντολή δημιουργίας γραφημάτων για το Bufferbloat δικτύο με buffer 20 πακέτων



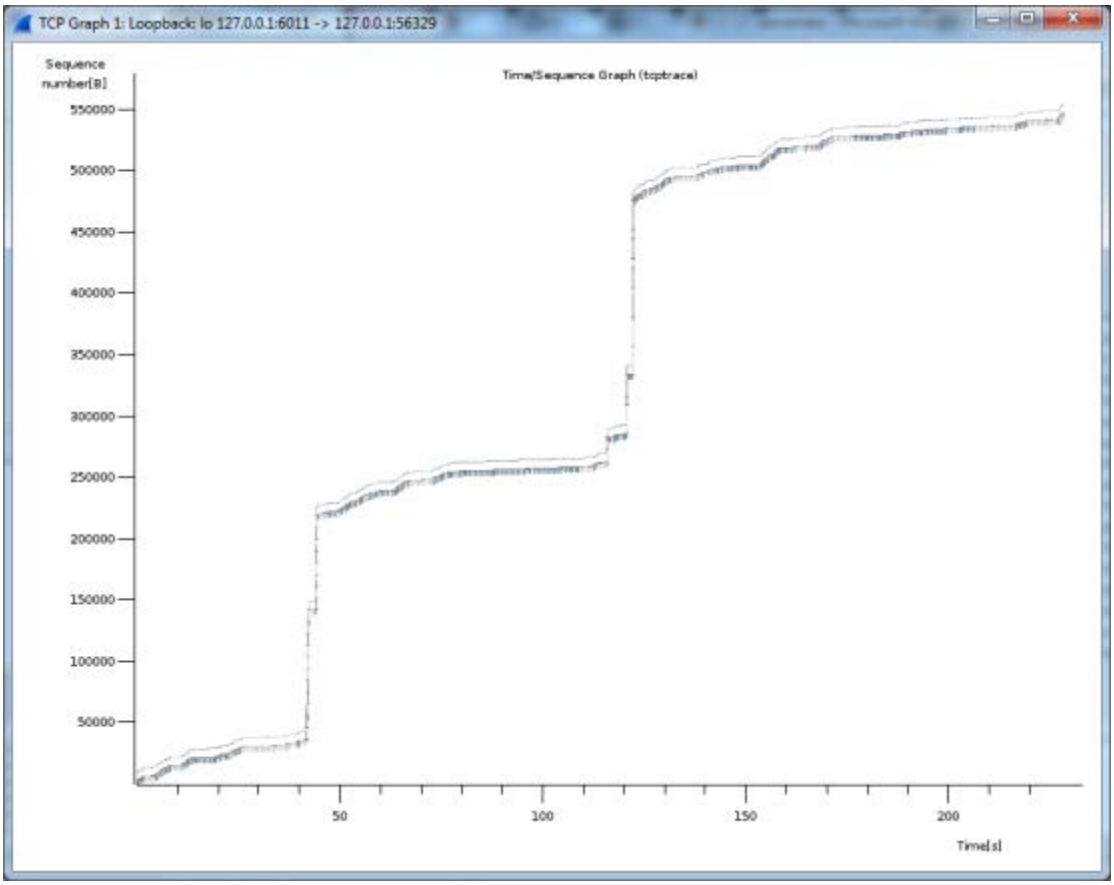
Γράφημα 5. 89 Πληρότητα ουράς για το μεταγωγέα 20 πακέτων



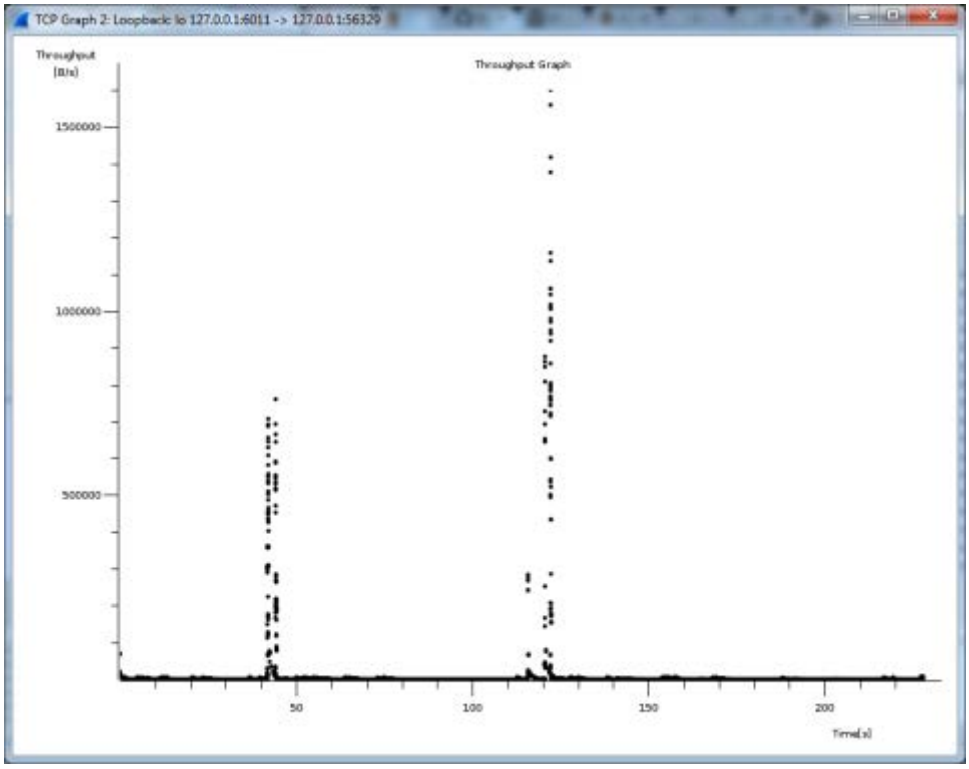
Γράφημα 5. 90 TCP congestion window όταν έχουμε iperf & buffer 20 πακέτων



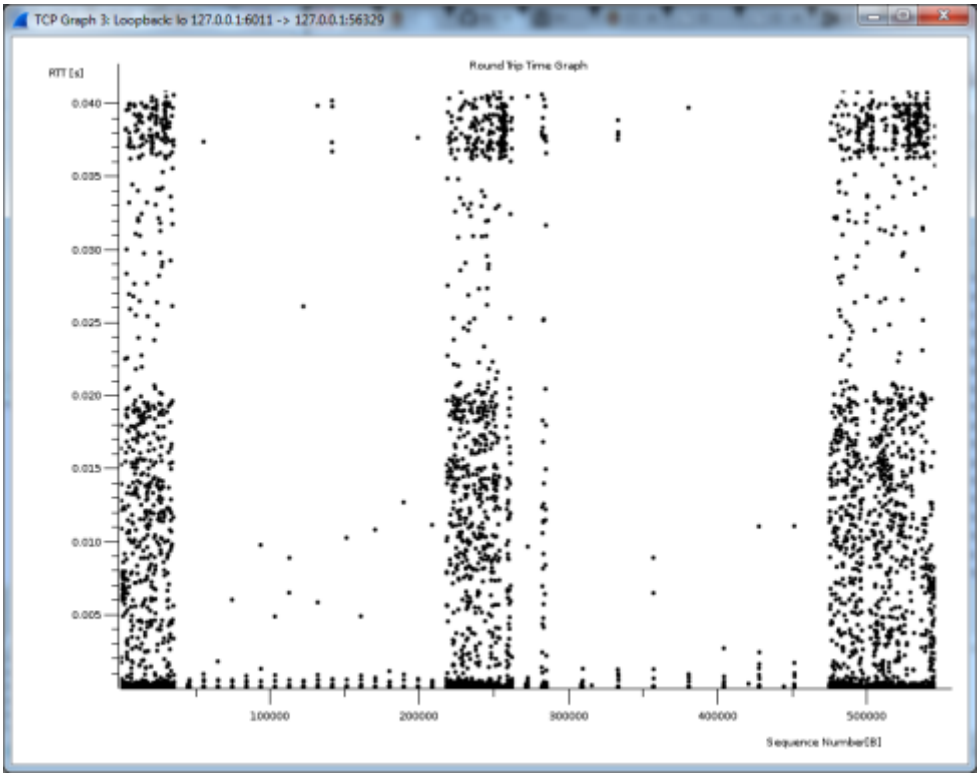
Γράφημα 5. 91 TCP congestion window όταν έχουμε wget & buffer 20 πακέτων.



Γράφημα 5. 92 Time Sequence κατά την διάρκεια του Iperf & wget με buffer 20 πακέτων



Γράφημα 5. 93 Throughput κατά την διάρκεια του Iperf & wget με buffer 20 πακέτων.



Γράφημα 5. 94 RTT κατά την διάρκεια του Iperf & wget με buffer 20 πακέτων

5.6.4 Υλοποίηση Buffer με 2 ουρές.

Εκκινούμε το Mininet ξανά αλλά αυτήν την φορά θα δημιουργήσουμε δυο διαφορετικές ουρές μια για κάθε τύπο κίνησης [26]. Δίνουμε λοιπόν την ακόλουθη εντολή στο Mininet.

$ **sudo ./run-diff.sh**

Επαναλαμβάνουμε τα τελευταία βήματα του πειράματος:

mininet> **h2 wget http://10.0.0.1**

mininet> **h1 ping -c 10 h2**

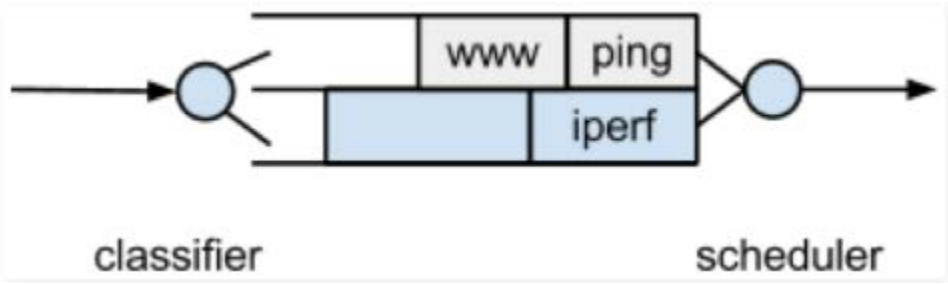
mininet> **h1 ./iperf.sh**

mininet> **h1 ping -c 30 h2**

mininet> **h2 wget http://10.0.0.1**

Παρατηρούμε ότι τα αποτελέσματα του ping καθώς και ο χρόνος κατεβάσματος της ιστοσελίδας με την εντολή wget δεν διαφοροποιείται σημαντικά πριν και μετά την εισαγωγή κίνησης στο δίκτυο με την εντολή iperf.

Το πρόβλημα φαίνεται να είναι ότι τα πακέτα της σύντομης ροής έχουν κολλήσει πίσω από πολλά πακέτα της μεγάλης ροής βρίσκονται δηλαδή στην ίδια ουρά. Τι θα συμβεί αν έχουμε διατηρήσει μια ξεχωριστή ουρά για κάθε ροή και στη συνέχεια να θέσουμε την Iperf και την wget σε κυκλοφορία σε διαφορετικές ουρές;



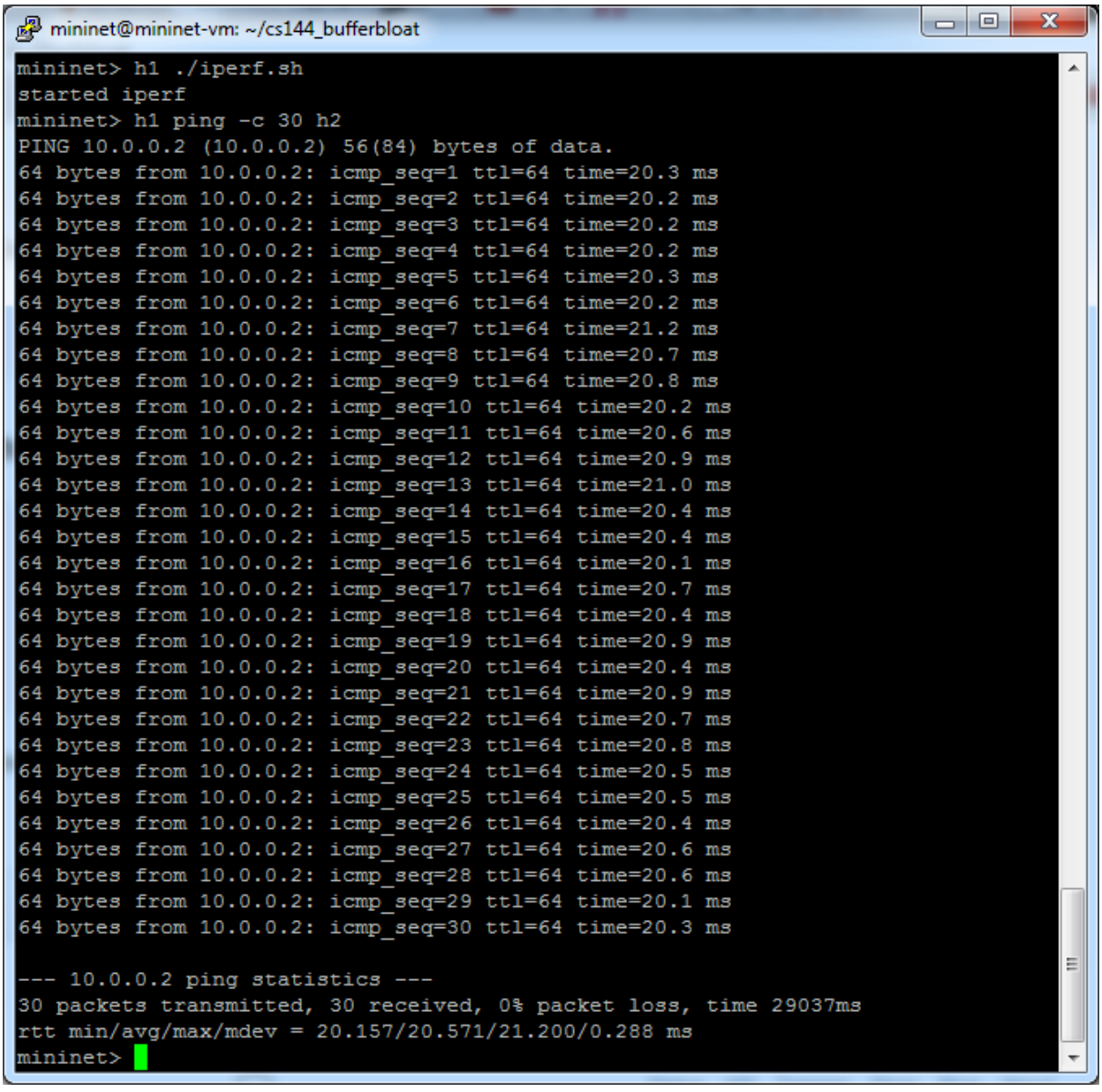
Εικόνα 5. 85 Σχηματική αναπαράσταση του buffer με δυο ουρές.

Για το πείραμα αυτό λοιπόν, βάζουμε το Iperf και πακέτα wget / ping σε ξεχωριστές ουρές στο δρομολογητή Headend [26]. Το script περιέχει υλοποίηση δίκαιης ουρά αναμονής, έτσι ώστε όταν οι δύο ουρές είναι απασχολημένες, κάθε ροή να λαμβάνει το ήμισυ του ποσοστού σύνδεσης συμφόρησης.

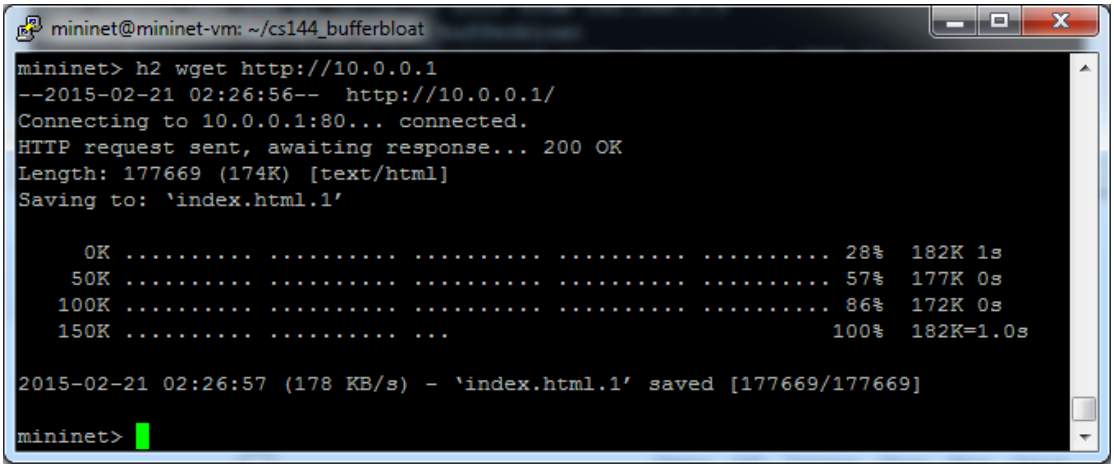
```
mininet@mininet-vm: ~/cs144_bufferbloat
mininet> h2 wget http://10.0.0.1
--2015-02-21 02:25:02--  http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177669 (174K) [text/html]
Saving to: 'index.html'

     0K .......... .......... .......... .......... .......... 28%  182K 1s
    50K .......... .......... .......... .......... .......... 57%  178K 0s
   100K .......... .......... .......... .......... .......... 86%  172K 0s
   150K .......... .......... ...                              100%  182K=1.0s

2015-02-21 02:25:03 (178 KB/s) - 'index.html' saved [177669/177669]

mininet> h1 ping -c 10 h2
10.0.0.2 - - [21/Feb/2015 02:25:02] "GET / HTTP/1.1" 200 -
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=22.2 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=20.4 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=20.3 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=20.3 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=20.2 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=20.2 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=20.3 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=20.8 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=20.8 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9012ms
rtt min/avg/max/mdev = 20.188/20.629/22.227/0.600 ms
mininet>
```

Εικόνα 5. 86 Wget & ping στο δίκτυο με buffer με 2 ξεχωριστές ουρές.

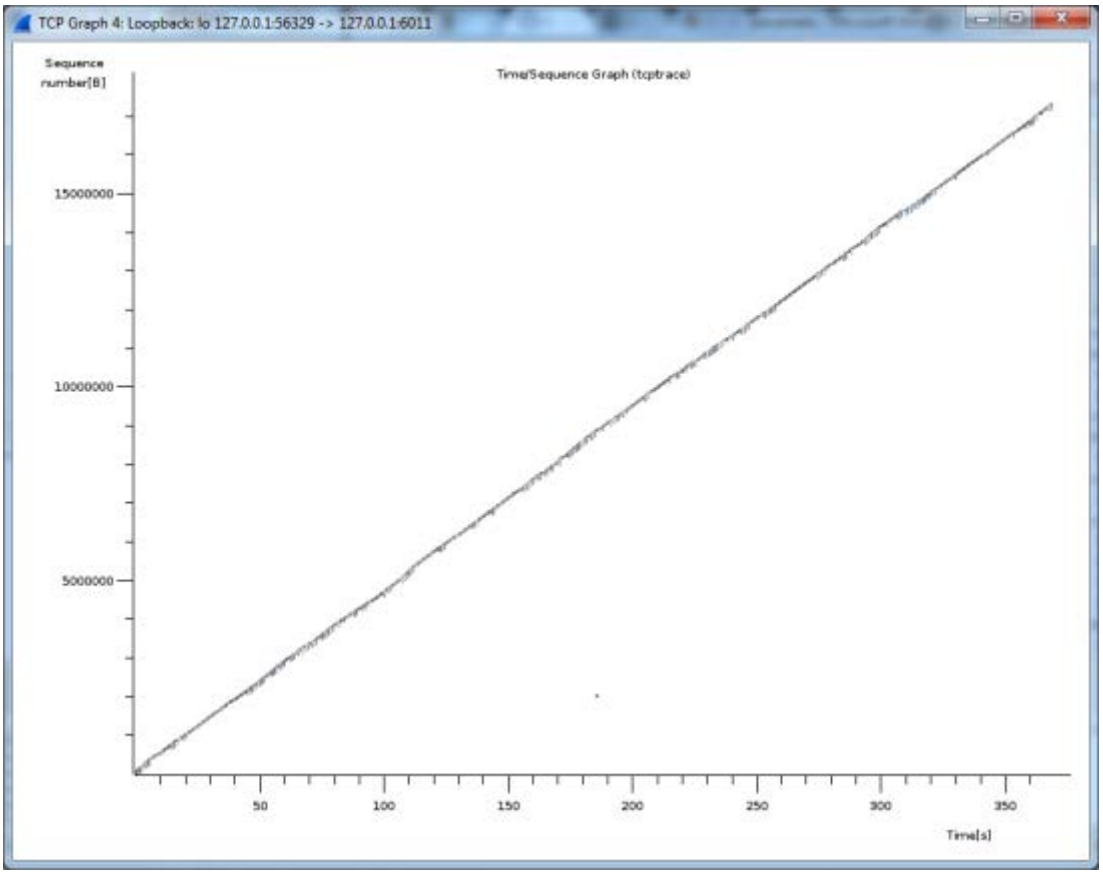
Παρατηρούμε ότι ο χρόνος που απαιτήθηκε για να κατέβει η ιστοσελίδα είναι πλέον 1 δευτερόλεπτο. Οι χρόνοι RTT είναι κοντά στα 0,010 δευτερόλεπτα στο μεγαλύτερο μέρος τους, η απόδοση του δικτύου είναι ικανοποιητική σύμφωνα με το tcptrace γράφημα και το throughput κυμαίνεται σε φυσιολογικά πλαίσια.



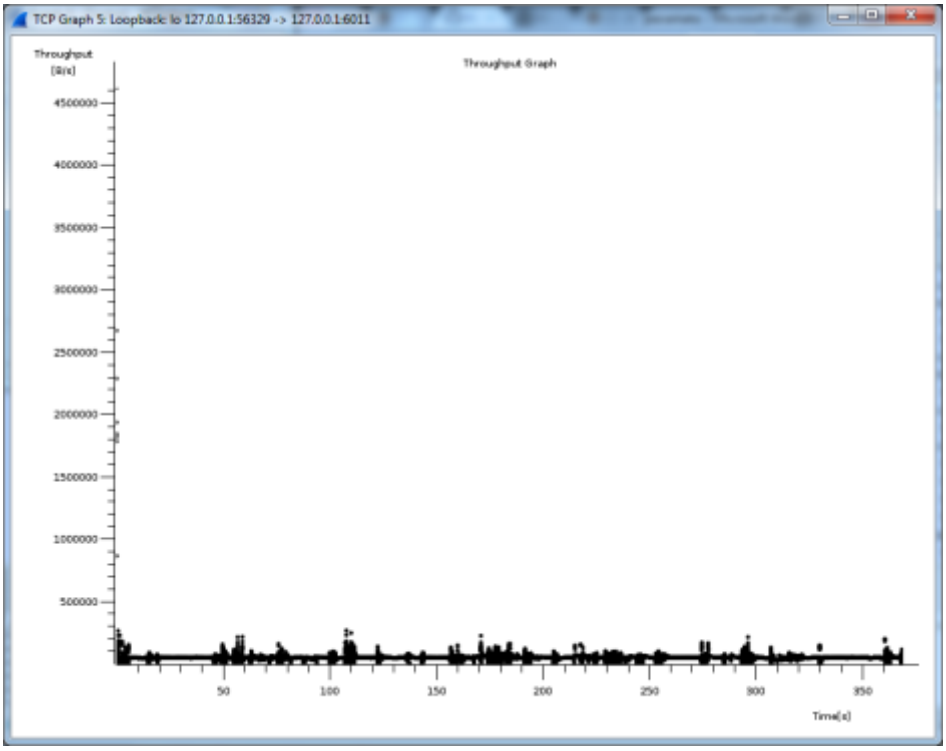
Εικόνα 5. 87 Ping μαζί με iperf στο δικτύο με το buffer με 2 ξεχωριστές ουρές.



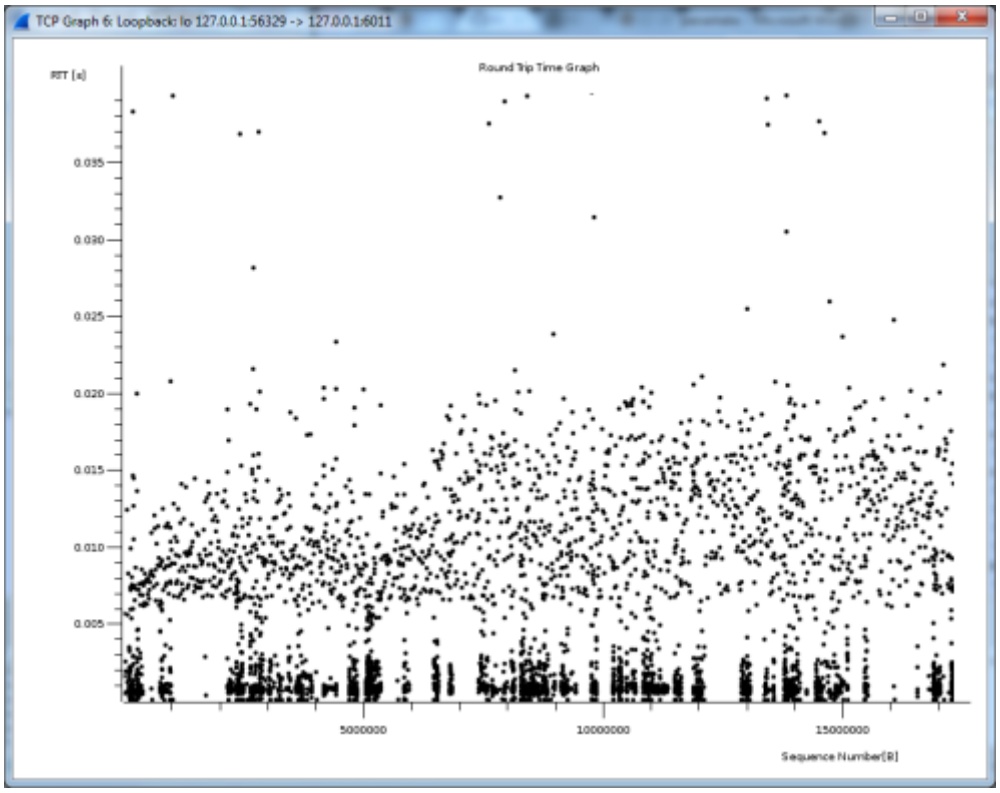
Εικόνα 5. 88 Αποτελέσματα wget στο δικτύο με το buffer με 2 ξεχωριστές ουρές.



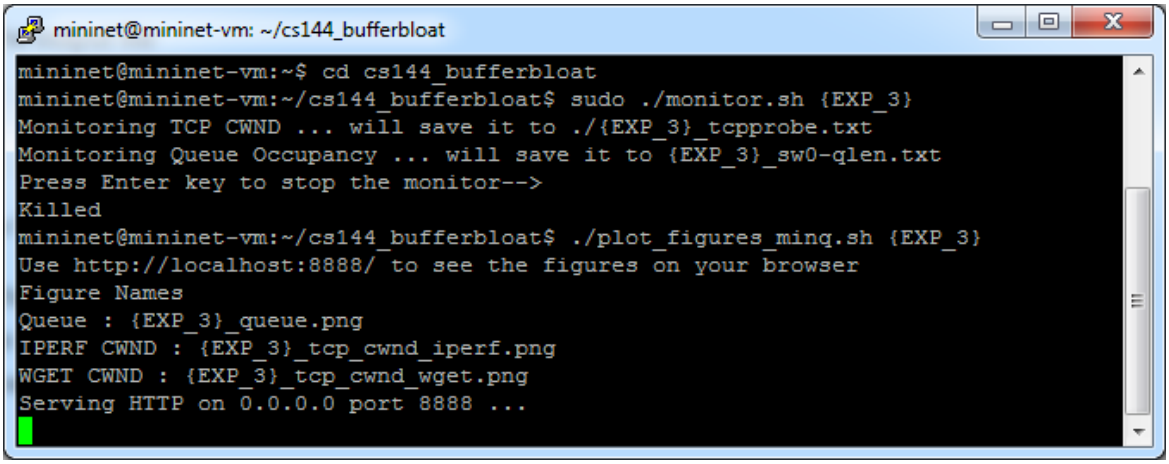
Γράφημα 5. 95 Time Sequence στο δίκτυο με το buffer με 2 ξεχωριστές ουρές.



Γράφημα 5. 96 Throughput στο δίκτυο με το buffer με 2 ξεχωριστές ουρές

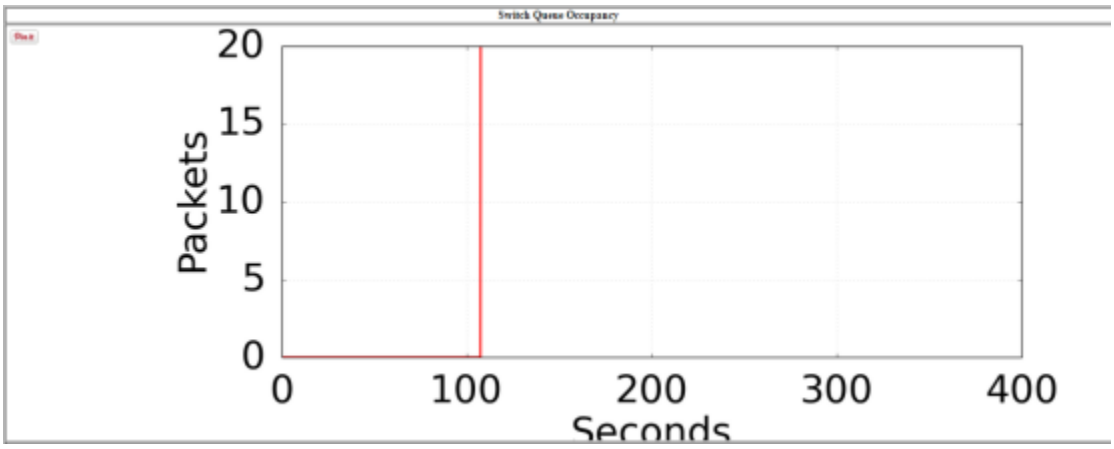


Γράφημα 5. 97 RTT στο δίκτυο με το buffer με 2 ξεχωριστές ουρές



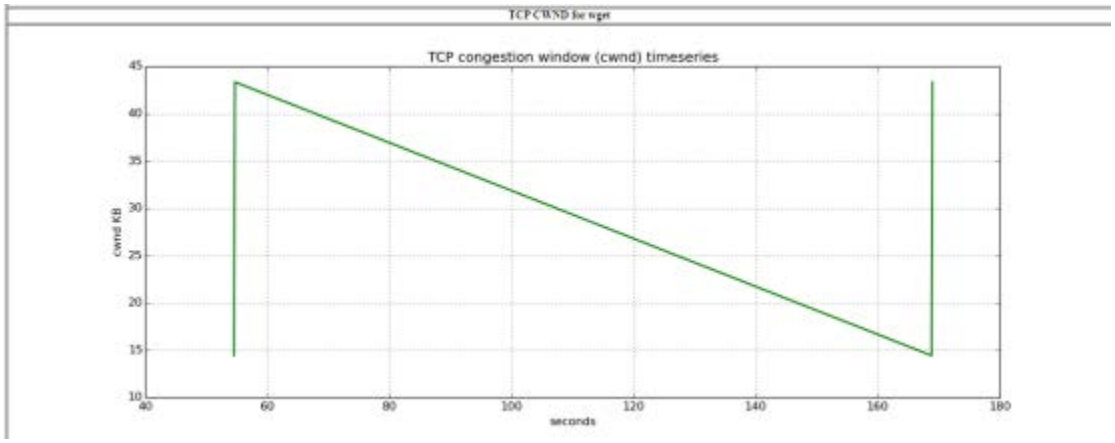
Εικόνα 5. 89 Εντολή καταγραφής αποτελεσμάτων στο δίκτυο με το buffer με 2 ξεχωριστές ουρές

Παρατηρούμε από τα παρακάτω γραφήματα ότι τόσο η καθυστέρηση του ping όσο και το κατέβασμα της σελίδας με την εντολή wget δεν διαφοροποιείται κατά την διάρκεια της εντολής iperf.



Γράφημα 5. 98 Πληρότητα ουράς στο δίκτυο με το buffer με 2 ξεχωριστές ουρές.

Γράφημα 5. 99 TCP Congestion Window για το iperf στο δίκτυο με το buffer με 2 ξεχωριστές ουρές



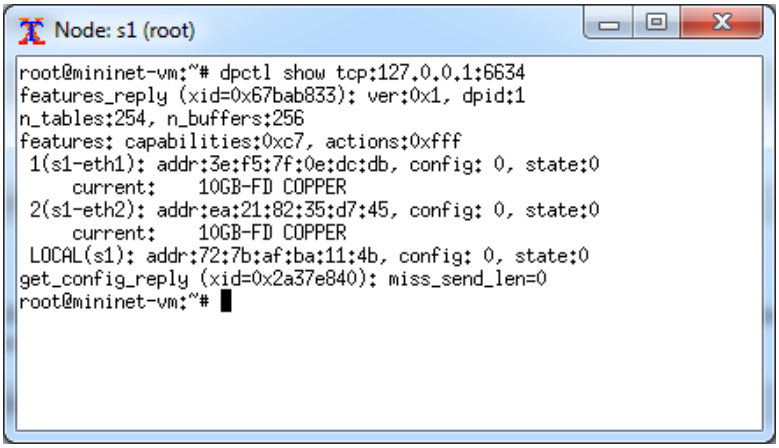
Γράφημα 5. 100 TCP Congestion Window για το wget στο δίκτυο με το buffer με 2 ξεχωριστές ουρές.

5.7 Τοποθέτηση Openflow Κανόνων.

Το dpctl είναι ένα βοηθητικό πρόγραμμα που συνοδεύει το OpenFlow και επιτρέπει την προβολή και τον έλεγχο του πίνακα ροών ενός μεταγωγέα. Είναι ιδιαίτερα χρήσιμο για τον εντοπισμό σφαλμάτων, διότι δίνει την δυνατότητα για προβολή της κατάσταση ροής και των μετρητών ροής [39]. Οι περισσότεροι μεταγωγείς OpenFlow μπορούν να ξεκινήσουν με μια παθητική θύρα ακρόασης (πολύ συχνά η θύρα είναι πιθανότατα η 6634), από την οποία μπορεί κάποιος να ελέγξει την κίνηση στον μεταγωγέα, χωρίς να χρειάζεται να προστεθεί κώδικας διόρθωσης σφαλμάτων στον ελεγκτή.

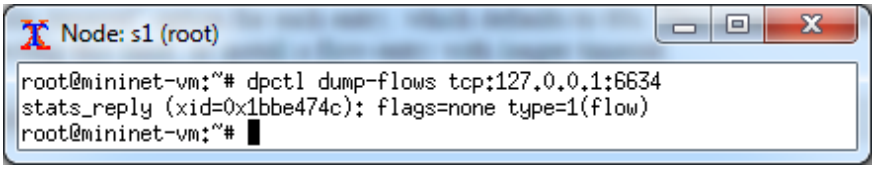
Η "show" εντολή συνδέει τον εντολέα με τον μεταγωγέα και επιστρέφει την κατάσταση των θυρών καθώς και τις ικανότητες τους. Για να δώσουμε την εντολή στον μεταγωγέα θα πρέπει να είμαστε στο αντίστοιχο παράθυρο xterm (root).

#dpctl show tcp:127.0.0.1:6634



Εικόνα 5. 90 Dpctl Show εντολή στον μεταγωγέα.

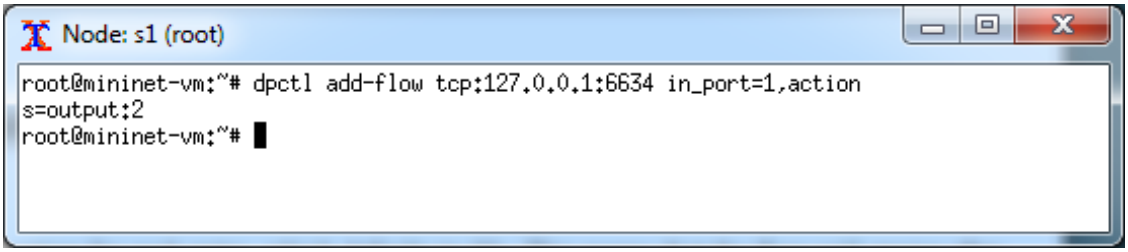
Η παρακάτω εντολή δείχνει τους τρέχοντες κανόνες: **# dpctl dump-flows tcp:127.0.0.1:6634**



Εικόνα 5. 91 Dpctl dump flows εντολή στον μεταγωγέα.

Για να εισάγουμε ένα νέο κανόνα μπορούμε να δώσουμε την εντολή:

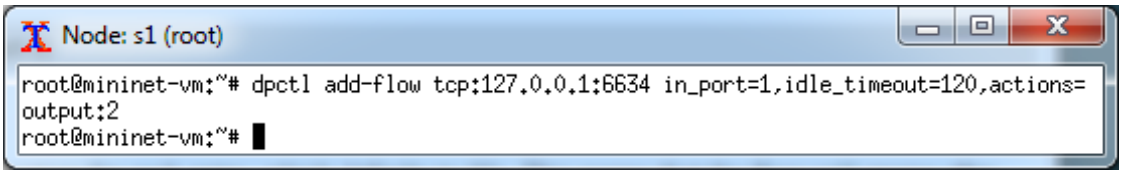
 # dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2



Εικόνα 5. 92 Dpctl add flow εντολή στον μεταγωγέα.

Όταν δίνουμε την εντολή "dpctl dump-flows"[38]μπορούμε να δούμε μια "idle_timeout" επιλογή για κάθε καταχώρηση, που προκαθορισμένα είναι στα 60sec. Αυτό σημαίνει ότι η ροή "λήγει" μετά από 60 δευτερόλεπτα όπου δεν έχουμε εισερχόμενη κίνηση. Εάν θέλουμε να εισάγουμε ένα κανόνα με μεγαλύτερο χρόνο αναμονής μπορούμε να δώσουμε μια εντολή σαν την παρακάτω:

#dpctl add-flow tcp:127.0.0.1:6634in_port=1,idle_timeout=120,actions=output:2



Εικόνα 5. 93 Dpctl add flow idle time out εντολή στον μεταγωγέα.

Κεφάλαιο 6
Επίλογος

Η απεριόριστη ανάπτυξη του δικτύου, η πληθώρα εφαρμογών και οι απαιτήσεις λόγω της εκρηκτικής αύξησης των χρηστών του δικτύου οδηγούν το συμβατικό δίκτυο και τις συσκευές του στα όριά τους [2]. Το SDN εισάγει ένα νέο τρόπο για να χειριστεί το τεράστιο ποσό των πακέτων που διέρχονται από το δίκτυο. Γνωρίζουμε ότι τα πακέτα μπορούν να ομαδοποιηθούν σε ροές και είναι δυνατός ο χειρισμός των πακέτων αυτών κεντρικά ανά ροή εφόσον ορίσουμε συγκεκριμένες δράσεις στους μεταγωγείς του δικτύου, ώστε να επιταχυνθεί η προώθηση τους στον κόμβο προορισμό. Αυτό είναι μόνο ένα από τα πολλά οφέλη ενός κεντρικού ελέγχου του δικτύου. [12]

Η πιο ευρέως διαδεδομένη αρχιτεκτονική SDN σήμερα είναι το OpenFlow πρωτόκολλο. Γιγαντιαίες εταιρείες, συμπεριλαμβανομένης της Google, έχουν αρχίσει να εφαρμόζουν το πρωτόκολλο OpenFlow στα δικά τους κέντρα δεδομένων [4][24]. Η σταδιακή εφαρμογή του SDN και του OpenFlow σε δικτυακές υποδομές υποθέτει αρχικά τη συνύπαρξη των δικτύων που βασίζονται στο OpenFlow με εκείνα των συμβατικών δικτύων. Αυτό απαιτεί εκτεταμένες μελέτες για να διερευνηθούν οι δυνατότητες και οι περιορισμοί του πρωτόκολλου. [3]

Πολλά πρωτόκολλα SDN είναι διαθέσιμα σήμερα, αλλά η χρήση του πρωτόκολλου OpenFlow συνιστάται ιδιαίτερα λόγω της φύσης του καθώς και της ευρείας και ταχείας του ανάπτυξης. Η σωστή χρήση του κατάλληλου λογισμικού προσομοίωσης στην μελέτη των OpenFlow και SDN καθώς και κατά την ανάπτυξη και δημιουργία έργων που βασίζονται σε νέες τεχνολογίες αιχμής εξοικονομεί πολύ χρόνο και χρήμα σε σύγκριση με την πρακτικές δοκιμές. Η άμεση πρακτική εφαρμογή των ιδεών μας είναι συχνά ιδιαίτερα ακριβή και οι

περισσότερες διαθέσιμες επιλογές στην αγορά προσφέρουν υποστήριξη σε παλαιότερες εκδόσεις του προτύπου OpenFlow. [12]

Σε άρθρο για το SDN που δημοσιεύτηκε στο The Econonist αναφέρεται ότι οι εταιρείες που χρησιμοποιούν λύσεις SDN μπορούν να εξοικονομήσουν μέχρι και 50% των δαπανών τους για τη δικτύωση - networking, μειώνοντας τις επενδύσεις σε συσκευές που χρησιμοποιούνται ως middleware και υιοθετώντας πιο προσιτές λύσεις. Το hardware μπορεί να γίνει πιο φτηνό, καθώς η ευφυΐα φεύγει από τις συσκευές και περνά στον κεντρικό controller του SDN. Διαδικασίες όπως η παρακολούθηση του δικτύου αλλά και έλεγχοι ασφάλειας μπορούν να γίνουν με εφαρμογές που τρέχουν σε SDN controller.

Όμως στο SDN, όπου τα στοιχεία των δικτύων ελέγχονται από απόσταση, είναι πολύ εύκολο να αυξηθεί το overhead, δηλαδή ο χρόνος επεξεργασίας που απαιτείται για τον έλεγχο των μεταδόσεων. Το πρόβλημα αυτό μηδενίζεται ελέγχοντας τις ροές κίνησης των πακέτων και επαναλαμβάνοντας την απόφαση που επιλέξαμε για το πρώτο πακέτο μιας συγκεκριμένης ροής σε όλα τα πακέτα της ίδιας ροής. Επίσης το overhead μπορεί να μειωθεί περαιτέρω, ομαδοποιώντας ροές, όπως τη συνολική κίνηση μεταξύ δύο κόμβων, και ορίζοντας αποφάσεις ελέγχου για το σύνολο αυτών των ροών. Η κεντρικοποιημένη διαχείριση κάνει εφικτή την εποπτεία του δικτύου και τη λήψη καλύτερων και πιο έξυπνων αποφάσεων κατά την πραγματοποίηση αλλαγών στο δίκτυο, από τη στιγμή που η αλλαγή συμβαίνει άπαξ. Όλα αυτά επίσης επιτρέπουν ευκολότερη διαχείριση της υποδομής και την μείωση του κόστους λειτουργίας της. [15]

Στην εποχή μας σημειώνεται τεράστια τεχνολογική πρόοδος, το φαινόμενο αυτό επιφέρει αύξηση της ζήτησης μεγαλύτερου εύρου ζώνης, καλύτερων υπηρεσιών, μεγαλύτερων ταχυτήτων, μεγαλύτερης αξιοπιστίας και με χαμηλότερο κόστος. Σύμφωνα με την Cisco, το SDN είναι μια καινοτόμα εξέλιξη [45]. Η φύση της αρχιτεκτονικής βοηθά να απλοποιηθούν διαδικασίες, αυτοματοποιώντας και κεντρικοποιώντας τον έλεγχο και τη διαχείριση. Το δίκτυο πλέον μπορεί να ανταποκρίνεται καλύτερα στην τρέχουσα δυναμική των επιχειρήσεων και των

σύγχρονων αναγκών και απαιτήσεων. Η Microsoft και η Google μιλάνε ήδη για το θετικό αντίκτυπο του SDN στη διαχείριση της κυκλοφορίας ευρείας περιοχής σε δίκτυα μεγάλης κλίμακας.

Το SDN επιτρέπει να δοθεί προτεραιότητα σε καίριες εφαρμογές και να ορισθεί η ποιότητα της υπηρεσίας (QoS) και οι παράμετροι καθυστέρησης. Τα νέα επίπεδα ευελιξίας, προγραμματισμού και διαχείρισης του δικτύου που προσφέρονται από τη χρήση αυτής της τεχνολογίας μπορούν να μας βοηθήσουν να αντιμετωπίσουμε τις ακόλουθες βασικές απαιτήσεις των WAN: το δυναμικό καθορισμό των τύπων κίνησης, τη μεταφορά δεδομένων μεταξύ απομακρυσμένων κόμβων, την αυξημένη ασφάλεια των συνδέσεων WAN, την ικανότητα για προγραμματισμό εφαρμογών μέσω API, την παρακολούθηση της κίνησης σε πραγματικό χρόνο και την ευέλικτη πρόσβαση με VPN, τη συνολική διαχείριση και εικόνα μεγάλων δικτύων. [11] [43]

Μια άλλη εφαρμογή του SDN, αφορά χρήστες ασύρματων δικτύων οι οποίοι θα μπορούν να κινούνται απρόσκοπτα εντός αυτών, για παράδειγμα, μέσω ενός ευέλικτου μηχανισμού μεταβίβασης handoff για τα WiFi-enabled τηλέφωνα από ένα σημείο πρόσβασης σε ένα άλλο. Επίσης, είναι πολύ σημαντικό ότι δίνεται η δυνατότητα να μιλάμε για μη IP δίκτυα, μιας και η συγκεκριμένη τεχνολογία δεν καθορίζει ότι τα πακέτα θα πρέπει να έχουν μια συγκεκριμένη μορφή. Ακόμα υπάρχει και η επιλογή της επεξεργασίας πακέτων αντί ροών, όπου ο ελεγκτής θα παίρνει αποφάσεις για το πότε αρχίζει η κάθε ροή. [11]

Για την εξέταση και υιοθέτηση των παραπάνω τεχνολογιών, απαιτείται η προηγούμενη προσομοίωση και εξομοίωση, των υπό έρευνα έργων δικτύωσης, Οι προσομοιώσεις ή εξομοιώσεις μπορούν να παρέχουν μια στερεή βάση για να καθοριστούν χωρίς κόστος τα πλεονεκτήματα και τα μειονεκτήματά τους. Η εξομοίωση είναι πιο ρεαλιστική από την προσομοίωση, δεδομένου ότι, πραγματοποιείται σε πραγματικό χρόνο και επιτρέπει σε πραγματικά μηχανήματα που τρέχουν κώδικα να επικοινωνούν με εξομοιωμένα συστήματα.

Σύμφωνα με την διδακτορική διατριβή του Brandon Heller, η υψηλής πιστότητας εξομοίωση αποτελεί έναν εύκολο τρόπο για τους ερευνητές να ελέγξουν τις δυνατότητες και τα μελανά σημεία των δικτύων υπό εξέταση και να αναπαράγουν με ευκολία δημοσιευμένα πειράματα άλλων ερευνητών. Το Mininet Hi Fi αποτελεί την κατάλληλη λύση για τέτοιου είδους εξομοιώσεις μιας και είναι εύκολο μέσω απλών εντολών ή κώδικα να αναπαρασταθούν ολόκληρα δίκτυα σχεδόν φτάνοντας τον ρεαλισμό των αποτελεσμάτων που παρέχουν τα testbeds. [17]

Μελλοντικά η παρούσα μεταπτυχιακή διατριβή θα μπορούσε να αποτελέσει εναρκτήριο βοήθημα για την ενασχόληση με την τεχνολογία SDN και το πρωτόκολλο Openflow. Επιπροσθέτως, με τη βοήθεια του εργαλείου εξομοίωσης Mininet θα μπορούσε οποιοδήποτε το επιθυμεί να ασχοληθεί με τη σχεδίαση δικτύων, να μελετήσει τα οφέλη του SDN αναπαράγοντας τα πειράματα που αναπτύχθηκαν, διαπιστώνοντας τις αλλαγές που προκαλεί σε ένα δίκτυο, για παράδειγμα, ο ορισμός του κατάλληλου μεγέθους buffer όπως φάνηκε και στο πείραμα με την ονομασία bufferbloat. Επίσης μπορεί ένας ερευνητής να μελετήσει τις κατάλληλες τοπολογίες δικτύων ανάλογα με τον τύπο των εφαρμογών που θα εκτελούνται σε αυτά, να παρατηρήσει χαρακτηριστικά των δικτύων υπό έρευνα, όπως η καθυστέρηση και το εύρος ζώνης, κάτι που έγινε σε όλα τα πειράματα αφού μελετήθηκαν τα γραφήματα που συλλέξαμε με την χρήση του Wireshark. Πέρα από τα προαναφερθέντα, ένα αντικείμενο πολύ ενδιαφέρον ως έρευνα θα ήταν η δημιουργία περισσότερων customised δικτύων στο Mininet και η μελέτη των ορίων αξιοπιστίας του Mininet, δημιουργώντας κίνηση της τάξεως των 1,5 Gbps σε αυτά, συγκρίνοντάς τα έτσι με πραγματικές υλοποιήσεις (testbeds). [14]

Βιβλιογραφία

[01] G. Appenzeller, I. Keslassy, and N. McKeown. 2004. Sizing router buffers. SIGCOMM Comput. Commun. *Rev.* 34, 4 (August 2004), 281-292

[02] M. Basheer Al-Somaidai, E. Bassam Yahya, (2014). Survey of software components to emulate OpenFlow protocol as an SDN implementation. American Journal of Software Engineering and Applications, [online] 1(3), pp.1-9. Available at: http://article.sciencepublishinggroup.com/pdf/10.11648.j.ajsea.20140306.12.pdf [Accessed 23 Mar. 2015].

[03] V. Gourov 'SDN-based concept for Network Monitoring', ICIST 2014 4th International Conference on Information Society and Technology Proceedings, 1(4 th.), Belgrade, Serbia, 2014,pp. 137-143.

[04] M.Großmann, S. Schuberth 2013. Auto Mininet: Assessing the Internet Topology Zoo in a Software-Defined Network Emulator In: 7 Workshop 2013 (MMBnet 2013), 05.-06.09.13, Hamburg, Germany.

[05] M. Gupta, J.Sommers, and P. Barford. 2013. Fast, accurate simulation for SDN prototyping. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (HotSDN '13). ACM, New York, NY, USA, 31-36.

[06] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, N. McKeown (2012) ' Mininet Performance Fidelity Benchmarks', , (CSTR 2012-2)

[07] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. 2012. "Reproducible network experiments using container-based emulation." In Proceedings of the 8th international conference on Emerging networking experiments and technologies (CoNEXT '12). ACM, New York, NY, USA, 253-264.

[08] B. Heller, (2013). Reproducible Network Research with High-Fidelity Emulation. PhD. Stanford.

[09] F. Hu, (2014). Network Innovation through OpenFlow and SDN. NW: Taylor & Francis Group, LLC, pp.3-13.

[10] S. Kaur , J. Singh and N. Singh Ghumman In Proceedings ICCCS 2014, Shaheed Bhadat Singh State Technical Campus,08.-09.8.2014 Punjab, India p. 134-138

[11] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J. (2008). "OpenFlow. OpenFlow: Enabling Innovation in Campus Networks" SIGCOMM Comput. Commun. *Rev.*, 38(2), p.69-74.

[12] B. Lantz B. Heller N. McKeown " A Network in a Laptop: Rapid Prototyping for Software-Defined Networks" Proceeding Hotnets-IX Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Article No. 19, 6, 2010

[13] A. Lara, A. Kolasani, B. Ramamurthy; "Network Innovation using OpenFlow: A Survey," IEEE Communications Surveys & Tutorials, vol.16, no.1, pp.493,512, First Quarter 2014

[14] Raja Rathnam Naidu K, (n.d.). Performance Analysis of Open vSwitch under Congestion. 1st ed.

[15] B. Nunes Astuto, M. Mendon¸ca, X. Nam Nguyen, K. Obraczka, T. Turletti. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. To appear in IEEE Communications Surveys & Tutorials. 2013, Submitted on 29 Oct 2013 (v3), last revised 19 Jan 2014 (v5)

[16] E. Sørensen (2014) SDN used for policy enforcement in a federated military network. Msc Norwegian University of Science and Technology

[17] H. Zeng. (2014) Automatic Data Plane Testing. [Doctor of Philosophy]. Stanford University.

[18] Α. Ιεροδιάκονος. Ανάπτυξη Application Aware Controller σε Περιβάλλον Openflow. (2013). Bachelor. Εθνικό Μετσόβιο Πολυτεχνείο.

[19] 1, h. (2015). *lost and found (for me ?): how to use Mininet part 1*. [online] Lost-and-found-narihiro.blogspot.gr. Available at: http://lost-and-found-narihiro.blogspot.gr/2012/12/how-to-use-mininet-part-1.html [Accessed 19 Mar. 2015].

[20] Archive.openflow.org, (2015). [online] OpenFlow Switch Specification Version 1.1.0 Implemented (Wire Protocol 0x02) Available at: http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf [Accessed 19 Mar. 2015]..

[21] Archive.openflow.org, (2015). HOTITutorial2010 - OpenFlow Wiki. [online] Available at: http://archive.openflow.org/wk/index.php/HOTITutorial2010 [Accessed 19 Mar. 2015].

[22] Archive.openflow.org, (2015). *OpenFlow Tutorial - OpenFlow Wiki*. [online] Available at: http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial [Accessed 19 Mar. 2015].

[23] Big Switch Networks, Inc., (2015). *Medium, Large and Extra, Extra Large*. [online] Available at: http://www.bigswitch.com/blog/2012/10/30/medium-large-and-extra-extra-large [Accessed 20 Mar. 2015].

[24] Biztech.gr, (2015). *Software Defined Networking (SDN): Η ανά-γέννηση του δικτύου*. [online] Available at: http://biztech.gr/software-defined-networking-sdn-h-ana-gennisi-tou-diktuou [Accessed 28 Mar. 2015].

[25] Cs.colostate.edu, (2015). *Basic vi Commands*. [online] Available at: http://www.cs.colostate.edu/helpdocs/vi.html [Accessed 29 Mar. 2015].

[26] GitHub, (2014). *mininet/mininet*. [online] Available at: https://github.com/mininet/mininet/wiki/Bufferbloat [Accessed 19 Mar. 2015].

[27] GitHub, (2015). *Mininet*. [online] Available at: https://github.com/mininet/mininet [Accessed 19 Mar. 2015].

[28] GitHub, (2015). mininet/mininet. [online] Available at: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet [Accessed 19 Mar. 2015].

[29] Hiemstra, J. (2015). *CCNA TechNotes: OSI model*. [online] Techexams.net. Available at: http://www.techexams.net/technotes/ccna/osimodel.shtml [Accessed 22 Mar. 2015].

[30] Iperf.fr, (2015). *Iperf - The TCP/UDP Bandwidth Measurement Tool*. [online] Available at: https://iperf.fr/ [Accessed 19 Mar. 2015].

[31] Irg.cs.ohiou.edu, (2015). *Tcptrace Time Sequence Graph*. [online] Available at: http://irg.cs.ohiou.edu/tcptrace/tcptrace_old/tsg.html [Accessed 22 Mar. 2015].

[32] NetworkStatic | Brent Salisbury's Blog, (2012). *The Northbound SDN API - A Big Little Problem*. [online] Available at: http://networkstatic.net/the-northbound-api-2/ [Accessed 20 Mar. 2015].

[33] OpenFlow Switch Specification Version 1.4.0 (Wire Protocol 0x05). (2013). 1st ed. [ebook] Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf [Accessed 19 Mar. 2015].

[34] OpenFlow Switch Specification. (2009). Version 1.0.0 (Wire Protocol 0x01)1st ed. [ebook] Available at: http://OpenFlowSwitch.org [Accessed 19 Mar. 2015].

[35] Open-Source Routing and Network Simulation, (2013). Set up the Mininet network simulator. [online] Available at: http://www.brianlinkletter.com/set-up-mininet/ [Accessed 19 Mar. 2015].

[36] Opennetworking.org, (2015). *Software-Defined Networking (SDN) Definition - Open Networking Foundation*. [online] Available at:

https://www.opennetworking.org/sdn-resources/sdn-definition [Accessed 1 Apr. 2015].

[37] Pages.cs.wisc.edu, (2015). *CS 640, Spring 2014 - Project1*. [online] Available at: http://pages.cs.wisc.edu/~agember/cs640/s14/project1 [Accessed 19 Mar. 2015].

[38] Route Reflector, (2013). *MiniNet as an SDN test platform*. [online] Available at: http://www.routereflector.com/2013/11/mininet-as-an-sdn-test-platform/ [Accessed 19 Mar. 2015].

[39] Sdn.cs.wisc.edu, (2015). *SDN Bootcamp - Session1*. [online] Available at: http://sdn.cs.wisc.edu/bootcamp/session1 [Accessed 19 Mar. 2015].

[40] Sdn.cs.wisc.edu, (2015). *SDN Bootcamp - Session2*. [online] Available at: http://sdn.cs.wisc.edu/bootcamp/session2 [Accessed 19 Mar. 2015].

[41] Sdnhub.org, (2015). *OpenFlow version 1.3 tutorial | SDN Hub*. [online] Available at: http://sdnhub.org/tutorials/openflow-1-3/ [Accessed 1 Apr. 2015].

[42] Sdnhub.org, (2015). *Useful mininet setups | SDN Hub*. [online] Available at: http://sdnhub.org/resources/useful-mininet-setups/ [Accessed 19 Mar. 2015].

[43] SearchSDN, (2015). *In the SDN WAN: Network programmability, provisioning and high availability*. [online] Available at: http://searchsdn.techtarget.com/tip/In-the-SDN-WAN-Network-programmability-provisioning-and-high-availability [Accessed 28 Mar. 2015].

[44] Searchsdn.techtarget.com, (2015). *Northbound API guide: The rise of the network applications*. [online] Available at: http://searchsdn.techtarget.com/guides/Northbound-API-guide-The-rise-of-the-network-applications [Accessed 20 Mar. 2015].

[45] Software-Defined Networking: Why We Like It and How We Are Building On It. (2013). 1st ed. [ebook] Cisco. Available at: http://www.cisco.com/web/strategy/docs/gov/cis13090_sdn_sled_white_paper.pdf [Accessed 28 Mar. 2015].

[46] Team, M. (2015). *Download/Get Started with Mininet - Mininet*. [online] Mininet.org. Available at: http://mininet.org/download/ [Accessed 19 Mar. 2015].

[47] Team, M. (2015). *Mininet Walkthrough - Mininet*. [online] Mininet.org. Available at: http://mininet.org/walkthrough/ [Accessed 19 Mar. 2015].

[48] Team, M. (2015). *Mininet Walkthrough - Mininet*. [online] Mininet.org. Available at: http://mininet.org/walkthrough/#link-variations [Accessed 19 Mar. 2015].

[49] Warfield, A. (2015). *SDN success in the real world*. [online] InfoWorld. Available at: http://www.infoworld.com/article/2610459/sdn/sdn-success-in-the-real-world.html [Accessed 28 Mar. 2015].

[50] Web.stanford.edu, (2015). *Programming Assignment 1: Bufferbloat*. [online] Available at: http://web.stanford.edu/class/cs244/pa1.html [Accessed 19 Mar. 2015].

[51] Wiki.utdallas.edu, (2015). *X11 Forwarding using Xming and PuTTY - Frequently Asked Questions - UTD Wiki*. [online] Available at: https://wiki.utdallas.edu/wiki/display/FAQ/X11+Forwarding+using+Xming+and+PuTTY [Accessed 19 Mar. 2015].

[52] Udacity.com, (2015). *Assignment 2 - Mininet Topology - Udacity*. [online] Available at: https://www.udacity.com/wiki/cn/assignment2-topology [Accessed 1 Apr. 20

Παράρτημα Α
Ακρωνύμια

AIMD	Additive-Increase/Multiplicative-Decrease algorithm
API	Application Programming Interface
ARP	Address Resolution Protoco
CLI	Command Line Interface
CWND	Congestion Window
ForCES	Forwarding and Control Element Separation
HTTP	HyperText Transfer Protocol
ISO	International Standards Organisation
MAC address	Media Access Control address
Mbps	Megabit per second
NAT	Network Address Translation
NOS	Network Operating System
OF	OpenFlow

OFC	OpenFlow Controller
ONF	Open Network Foundation
OSI	Open Systems Interconnection
OVS-Controller	Open Virtual Switch Controller
QoS	Quality of Service
RAM	Random access memory
RTT	Round-Trip Time
SDN	Software-Defined Networking
SSH	Secure Shell
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VM	Virtual Machine
VPN	Virtual Private Network
WAN	Wide Area Networks
X11	X Window System
XTERM	Standard TERMinal emulator for the X Window System

Παράρτημα Β

Κώδικες

Project1

Κώδικας project1_switch.py

```
#!/usr/bin/env python
"""
author: Aaron Gember (agember@cs.wisc.edu)

Custom topologies for CS640, Spring 2014, Project 1
"""

from mininet.cli import CLI
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.topo import Topo
from mininet.log import setLogLevel

class SwitchedNetwork(Topo):
    def __init__(self, **opts):
        Topo.__init__(self, **opts)
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')
        s1 = self.addSwitch('s1')
        self.addLink(h1, s1, bw=10)
        self.addLink(h2, s1, bw=10)
        self.addLink(h3, s1, bw=10)
        self.addLink(h4, s1, bw=10)

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

    # Create data network
    topo = SwitchedNetwork()
    net = Mininet(topo=topo, link=TCLink, autoSetMacs=True,
            autoStaticArp=True)

    # Run network
    net.start()
    CLI( net )
    net.stop()
```

Project1_point.py

```
#!/usr/bin/env python
"""
author: Aaron Gember (agember@cs.wisc.edu)

Custom topologies for CS640, Spring 2014, Project 1
"""

from mininet.cli import CLI
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.topo import Topo
from mininet.log import setLogLevel

class PointToPoint(Topo):
    def __init__(self, **opts):
        Topo.__init__(self, **opts)
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        self.addLink(h1, h2, bw=10)

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

    # Create data network
    topo = PointToPoint()
    net = Mininet(topo=topo, link=TCLink, autoSetMacs=True,
            autoStaticArp=True)

    # Run network
    net.start()
    CLI( net )
    net.stop()
```

Project1_network.py

```
#!/usr/bin/env python
"""
author: Aaron Gember (agember@cs.wisc.edu)

Custom topologies for CS640, Spring 2014, Project 1
"""

from mininet.cli import CLI
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.topo import Topo
from mininet.log import setLogLevel

class NetworkOfNetworks(Topo):
```

```
    def __init__(self, **opts):
        Topo.__init__(self, **opts)
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')
        h5 = self.addHost('h5')
        h6 = self.addHost('h6')
        h7 = self.addHost('h7')
        h8 = self.addHost('h8')
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        self.addLink(h1, s1, bw=10)
        self.addLink(h2, s1, bw=10)
        self.addLink(h3, s2, bw=10)
        self.addLink(h4, s2, bw=10)
        self.addLink(h5, s3, bw=10)
        self.addLink(h6, s3, bw=10)
        self.addLink(h7, s4, bw=10)
        self.addLink(h8, s4, bw=10)
        self.addLink(s1, s2, bw=10)
        self.addLink(s2, s3, bw=15)
        self.addLink(s3, s4, bw=20)

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

    # Create data network
    topo = NetworkOfNetworks()
    net = Mininet(topo=topo, link=TCLink, autoSetMacs=True,
           autoStaticArp=True)

    # Run network
    net.start()
    CLI( net )
    net.stop()
```

Pox.py

```
#!/bin/sh -

# Copyright 2011-2012 James McCauley
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at:
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

```
# If you have PyPy 1.6+ in a directory called pypy alongside pox.py,
we
# use it.
# Otherwise, we try to use a Python interpreter called python2.7,
which
# is a good idea if you're using Python from MacPorts, for example.
# We fall back to just "python" and hope that works.

''''true

#export OPT="-u -O"

export OPT="-u"

export FLG=""

if [ "$(basename $0)" = "debug-pox.py" ]; then

  export OPT=""

  export FLG="--debug"

fi


if [ -x pypy/bin/pypy ]; then

  exec pypy/bin/pypy $OPT "$0" $FLG "$@"

fi


if type python2.7 > /dev/null 2> /dev/null; then

  exec python2.7 $OPT "$0" $FLG "$@"

fi

exec python $OPT "$0" $FLG "$@"

'''


from pox.boot import boot


if __name__ == '__main__':

  boot()
```

Network Assignment 2

measure.py

```
#!/usr/bin/python

"Networking Assignment 2"

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.log import lg, output
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import irange, custom, quietRun, dumpNetConnections
from mininet.cli import CLI

from time import sleep, time
from multiprocessing import Process
from subprocess import Popen
import argparse

import sys
import os
from util.monitor import monitor_devs_ng

from mntopo import MNTopo

parser = argparse.ArgumentParser(description="Topology bandwith and
TCP tests")

parser.add_argument('--dir', '-d',
                    help="Directory to store outputs",
                    default="results")

parser.add_argument('--cli', '-c',
                    action='store_true',
                    help='Run CLI for topology debugging purposes')

parser.add_argument('--time', '-t',
                    dest="time",
                    type=int,
                    help="Duration of the experiment.",
                    default=60)

parser.add_argument('--buffer', '-s',
                    dest="size",
                    type=int,
                    help="Size of buffer.")

# Expt parameters
args = parser.parse_args()

if not os.path.exists(args.dir):
    os.makedirs(args.dir)

lg.setLogLevel('info')
```

```
def waitListening(client, server, port):
    "Wait until server is listening on port"
    if not 'telnet' in client.cmd('which telnet'):
        raise Exception('Could not find telnet')
    cmd = ('sh -c "echo A | telnet -e A %s %s"' %
           (server.IP(), port))
    while 'Connected' not in client.cmd(cmd):
        output('waiting for', server,
               'to listen on port', port, '\n')
        sleep(.5)

def progress(t):
       # Begin: Template code
    while t > 0:
        print '  %3d seconds left  \r' % (t)
        t -= 1
        sys.stdout.flush()
        sleep(1)

def start_tcpprobe():
    os.system("rmmod tcp_probe 1>/dev/null 2>&1; modprobe tcp_probe")
    Popen("cat /proc/net/tcpprobe > %s/tcp_probe.txt" % args.dir,
shell=True)

def stop_tcpprobe():
    os.system("killall -9 cat; rmmod tcp_probe")

def get_txbytes(iface):
    f = open('/proc/net/dev', 'r')
    lines = f.readlines()
    for line in lines:
        if iface in line:
            break
    f.close()
    if not line:
        raise Exception("could not find iface %s in /proc/net/dev:%s"
%
                        (iface, lines))
    return float(line.split()[9])

def get_rates(iface,nsamples=1,period=30,
          wait=10):
    """Returns rate in Mbps"""
    # Returning nsamples requires one extra to start the timer.
    nsamples += 1
    last_time = 0
    last_txbytes1 = 0
    last_txbytes2 = 0
    last_txbytes3 = 0
    ret = []
    sleep(wait)
    iface1 = 's1-eth1'
    iface2 = 's2-eth1'
    iface3 = 's3-eth1'
    while nsamples:
        nsamples -= 1

        txbytes1 = get_txbytes(iface1)
        txbytes2 = get_txbytes(iface2)
        txbytes3 = get_txbytes(iface3)
```

```
            now = time()
            elapsed = now - last_time

            last_time = now

            rate1 = (txbytes1 - last_txbytes1) * 8.0 / 1e6 / elapsed
            rate2 = (txbytes2 - last_txbytes2) * 8.0 / 1e6 / elapsed
            rate3 = (txbytes3 - last_txbytes3) * 8.0 / 1e6 / elapsed
            if last_txbytes1 != 0:
                ret.append(rate1)
                ret.append(rate2)
            ret.append(rate3)
                ret.append(rate1+rate2+rate3)
            last_txbytes1 = txbytes1
            last_txbytes2 = txbytes2
            last_txbytes3 = txbytes3
            sys.stdout.flush()
            sleep(period)
        return ret

    def run_topology_experiment(net):
        "Run experiement"

        seconds = args.time

        # Start the bandwidth and cwnd monitors in the background
        monitor = Process(target=monitor_devs_ng,
                args=('%s/bwm.txt' % args.dir, 1.0))
        monitor.start()
        start_tcpprobe()

        # Get receiver and clients
        recvr = net.getNodeByName('receiver')
        sender = net.getNodeByName('sender')

        s1 = net.getNodeByName('s1')

        # Start the receiver
        port = 5001
        recvr.cmd('iperf -s -p', port,
                  '> %s/iperf_server.txt' % args.dir, '&')

        waitListening(sender, recvr, port)

        sender.sendCmd('iperf -c %s -p %s -t %d -i 1 -yc >
    %s/iperf_%s.txt' %
            (recvr.IP(), 5001, seconds, args.dir, recvr))

        rates = get_rates(iface='s1-eth1')
        print rates

        sender.waitOutput()

        recvr.cmd('kill %iperf')

        # Shut down monitors
        monitor.terminate()
        stop_tcpprobe()

    def check_prereqs():
```

```
    "Check for necessary programs"
    prereqs = ['telnet', 'bwm-ng', 'iperf', 'ping']
    for p in prereqs:
        if not quietRun('which ' + p):
            raise Exception((
                'Could not find %s - make sure that it is '
                'installed and in your $PATH') % p)

def main():
    "Create and run experiment"
    start = time()

    topo = MNTopo()

    host = custom(CPULimitedHost, cpu=.15)  # 15% of system bandwidth
    link = custom(TCLink, max_queue_size=200)

    net = Mininet(topo=topo, host=host, link=link)

    net.start()

    print "*** Dumping network connections:"
    dumpNetConnections(net)

    print "*** Testing connectivity"

    net.pingAll()

    if args.cli:
        # Run CLI instead of experiment
        CLI(net)
    else:
        print "*** Running experiment"
        run_topology_experiment(net)

    net.stop()
    end = time()
    os.system("killall -9 bwm-ng")
    print "Experiment took %.3f seconds" % (end - start)

if __name__ == '__main__':
    check_prereqs()
    main()

#!/usr/bin/python

"Networking Assignment 2"

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.log import lg, output
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import irange, custom, quietRun, dumpNetConnections
from mininet.cli import CLI

from time import sleep, time
from multiprocessing import Process
from subprocess import Popen
import argparse
```

```
import sys
import os
from util.monitor import monitor_devs_ng

from mntopo import MNTopo

parser = argparse.ArgumentParser(description="Topology bandwith and
TCP tests")

parser.add_argument('--dir', '-d',
                    help="Directory to store outputs",
                    default="results")

parser.add_argument('--cli', '-c',
                    action='store_true',
                    help='Run CLI for topology debugging purposes')

parser.add_argument('--time', '-t',
                    dest="time",
                    type=int,
                    help="Duration of the experiment.",
                    default=60)

parser.add_argument('--buffer', '-s',
                    dest="size",
                    type=int,
                    help="Size of buffer.")

# Expt parameters
args = parser.parse_args()

if not os.path.exists(args.dir):
    os.makedirs(args.dir)

lg.setLogLevel('info')
```

mntopo.py

```
from mininet.topo import Topo

from mininet.net import Mininet

from mininet.link import TCLink

from mininet.util import custom


# Topology to be instantiated in Mininet

class MNTopo(Topo):

    "Mininet test topology"
```

```
    def __init__(self, cpu=.1, max_queue_size=None, **params):

        # Initialize topo
        Topo.__init__(self, **params)

        # Host and link configuration
        hostConfig = {'cpu': cpu}
        linkConfig = {'bw': 10, 'delay': '1ms', 'loss': 0,
                      'max_queue_size': max_queue_size }

        # Hosts and switches
        s1 = self.addSwitch('s1')
        sender = self.addHost('sender', **hostConfig)
        receiver = self.addHost('receiver', **hostConfig)

        # Wire receiver
        self.addLink(receiver, s1, **linkConfig)

        # Wire sender
        self.addLink(sender, s1, **linkConfig)
```

ping.py

```
#!/usr/bin/python

"Networking Assignment 2 Latency Script"

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

from mntopo import MNTopo
```

```
def latencyTest():
    "Create network and run latency test"
    topo = MNTopo()
    net = Mininet(topo=topo, host=CPULimitedHost, link=TCLink)
    net.start()
    #print "Testing network latency"

    # Get hosts
    sender = net.get('sender')
    receiver = net.get('receiver')

    result = sender.cmd('ping -c 5 ', receiver.IP())

    print result

    #net.pingAll()
    net.stop()

if __name__ == '__main__':
    setLogLevel('output')
    latencyTest()
```

topology.sh

```
#!/bin/bash


# Exit on any failure

set -e


# Check for uninitialized variables

set -o nounset


ctrlc() {

    killall -9 python

    mn -c

    exit

}


trap ctrlc SIGINT
```

```
start=`date`
exptid=`date +%b%d-%H:%M`
rootdir=topology-$exptid

# Note: you need to make sure you report the results
# for the correct port!
# In this example, we are assuming that each
# client is connected to port 2 on its switch.

for n in 1; do
    dir=$rootdir
    python measure.py \
        --dir $dir
    python util/plot_rate.py \
        --xlabel 'Time (s)' \
        --ylabel 'Rate (Mbps)' \
        -i 's.*-eth1' \
        -f $dir/bwm.txt \
        -o $dir/rate.png
    python util/plot_tcpprobe.py \
        -f $dir/tcp_probe.txt \
        -o $dir/cwnd.png
done

echo "Started at" $start
echo "Ended at" `date`
echo "Output saved to $rootdir"
```

plot-rate.sh

```
#!/bin/bash

# Creates a graph of the bandwidth from a bandwidth log

# Usage: ./plot-rate.sh ./student_folder/bwm.txt
# To use this script pass the path of the bandwidth log as the first
argument.

python util/plot_rate.py --rx \
        --maxy 10 \
        --xlabel 'Time (s)' \
        --ylabel 'Rate (Mbps)' \
        -i 's.*-eth2' \
        -f $1 \
        -o ./rate.png
```

Bufferbloat

bufferbloat.py

```
#!/usr/bin/python


"CS144 In-class exercise: Buffer Bloat"


from mininet.topo import Topo

from mininet.node import CPULimitedHost

from mininet.link import TCLink

from mininet.net import Mininet

from mininet.log import lg, info

from mininet.util import dumpNodeConnections

from mininet.cli import CLI

from monitor import monitor_qlen


from subprocess import Popen, PIPE

from time import sleep, time

from multiprocessing import Process
```

```
from argparse import ArgumentParser

import sys
import os

# Parse arguments

parser = ArgumentParser(description="BufferBloat tests")
parser.add_argument('--bw-host', '-B',
                    dest="bw_host",
                    type=float,
                    action="store",
                    help="Bandwidth of host links",
                    required=True)

parser.add_argument('--bw-net', '-b',
                    dest="bw_net",
                    type=float,
                    action="store",
                    help="Bandwidth of network link",
                    required=True)
parser.add_argument('--delay',
                    dest="delay",
                    type=float,
                    help="Delay in milliseconds of host links",
                    default=10)

parser.add_argument('--dir', '-d',
                    dest="dir",
                    action="store",
```

```
                    help="Directory to store outputs",
                    default="results",
                    required=True)

parser.add_argument('-n',
                    dest="n",
                    type=int,
                    action="store",
                    help="Number of nodes in star.",
                    required=True)

parser.add_argument('--nflows',
                    dest="nflows",
                    action="store",
                    type=int,
                    help="Number of flows per host (for TCP)",
                    required=True)

parser.add_argument('--maxq',
                    dest="maxq",
                    action="store",
                    help="Max buffer size of network interface in
packets",
                    default=500)

parser.add_argument('--cong',
                    dest="cong",
                    help="Congestion control algorithm to use",
                    default="reno")
parser.add_argument('--diff',
                    help="Enabled differential service",
```

```
                    action='store_true',
                    dest="diff",
                    default=False)

# Expt parameters
args = parser.parse_args()


class StarTopo(Topo):
    "Star topology for Buffer Bloat experiment"

    def __init__(self, n=2, cpu=None, bw_host=1000, bw_net=1.5,
                 delay=10, maxq=None, diff=False):
        # Add default members to class.
        super(StarTopo, self ).__init__()

        # Create switch and host nodes
        for i in xrange(n):
            self.addHost( 'h%d' % (i+1), cpu=cpu )


        self.addSwitch('s0', fail_mode='open')


        self.addLink('h1', 's0', bw=bw_host,
                     max_queue_size=int(maxq) )

        for i in xrange(1, n):
            self.addLink('h%d' % (i+1), 's0', bw=bw_host)
```

```
def ping_latency(net):
    "(Incomplete) verify link latency"
    h1 = net.getNodeByName('h1')
    h1.sendCmd('ping -c 2 10.0.0.2')
    result = h1.waitOutput()
    print "Ping result:"
    print result.strip()


def bbnet():
    "Create network and run Buffer Bloat experiment"
    print "starting mininet ...."
    # Seconds to run iperf; keep this very high
    seconds = 3600
    start = time()
    # Reset to known state
    topo = StarTopo(n=args.n, bw_host=args.bw_host,
                    delay='%sms' % args.delay,
                    bw_net=args.bw_net, maxq=args.maxq,
diff=args.diff)
    net = Mininet(topo=topo, host=CPULimitedHost, link=TCLink,
                  autoPinCpus=True)
    net.start()
    dumpNodeConnections(net.hosts)
    net.pingAll()
    print args.diff
    if args.diff:
        print "Differentiate Traffic Between iperf and wget"
        os.system("bash tc_cmd_diff.sh")
    else:
        print "exec tc_cmd.sh"
        os.system("bash tc_cmd.sh %s" % args.maxq)
```

```
    sleep(2)

    ping_latency(net)

    print "Initially, the delay between two hosts is around %dms" %
(int(args.delay)*2)

    h2 = net.getNodeByName('h2')

    h1 = net.getNodeByName('h1')

    h1.cmd('cd ./http/; nohup python2.7 ./webserver.py &')

    h1.cmd('cd ../')

    h2.cmd('iperf -s -w 16m -p 5001 -i 1 > iperf-recv.txt &')

    CLI( net )

    h1.cmd("sudo pkill -9 -f webserver.py")

    h2.cmd("rm -f index.html*")

    Popen("killall -9 cat", shell=True).wait()


if __name__ == '__main__':

    bbnet()
```

exp_monitor.py

```
from monitor import monitor_qlen


from subprocess import Popen, PIPE

from time import sleep, time

from multiprocessing import Process

from argparse import ArgumentParser


import sys

import os


parser = ArgumentParser(description="CWND/Queue Monitor")
```

```
parser.add_argument('--exp', '-e',
                    dest="exp",
                    action="store",
                    help="Name of the Experiment",
                    required=True)
# Expt parameters
args = parser.parse_args()


def start_tcpprobe():
    "Install tcp_pobe module and dump to file"
    os.system("(rmmod tcp_probe >/dev/null 2>&1); modprobe tcp_probe
full=1;")
    print "Monitoring TCP CWND ... will save it to ./%s_tcpprobe.txt
" % args.exp
    Popen("cat /proc/net/tcpprobe > ./%s_tcpprobe.txt" %
          args.exp, shell=True)


def qmon():
    monitor = Process(target=monitor_qlen,args=('s0-eth2', 0.01,
'%s_sw0-qlen.txt' % args.exp ))
    monitor.start()
    print "Monitoring Queue Occupancy ... will save it to %s_sw0-
qlen.txt " % args.exp
    raw_input('Press Enter key to stop the monitor--> ')
    monitor.terminate()


if __name__ == '__main__':
    start_tcpprobe()
    qmon()
    Popen("killall -9 cat", shell=True).wait()
```

monitor.py

```
from time import sleep, time
from subprocess import *
import re


default_dir = '.'


def monitor_qlen(iface, interval_sec = 0.01, fname='%s/qlen.txt' %
default_dir):
    pat_queued = re.compile(r'backlog\s[^\s]+\s([\d]+)p')
    cmd = "tc -s qdisc show dev %s" % (iface)
    ret = []
    open(fname, 'w').write('')
    while 1:
        p = Popen(cmd, shell=True, stdout=PIPE)
        output = p.stdout.read()
        # Not quite right, but will do for now
        matches = pat_queued.findall(output)
        if matches and len(matches) > 1:
            ret.append(matches[1])
            t = "%f" % time()
            open(fname, 'a').write(t + ',' + matches[1] + '\n')
        sleep(interval_sec)
    #open('qlen.txt', 'w').write('\n'.join(ret))
    return


def monitor_count(ipt_args="--src 10.0.0.0/8",
                  interval_sec=0.01, fname='%s/bytes_sent.txt'
                  % default_dir, chain="OUTPUT"):
    cmd = "iptables -I %(chain)s 1 %(filter)s -j RETURN" % {
```

```
            "filter": ipt_args,
            "chain": chain,
            }
        # We always erase the first rule; will fix this later
        Popen("iptables -D %s 1" % chain, shell=True).wait()
        # Add our rule
        Popen(cmd, shell=True).wait()
        open(fname, 'w').write('')
        cmd = "iptables -vnL %s 1 -Z" % (chain)
        while 1:
            p = Popen(cmd, shell=True, stdout=PIPE)
            output = p.stdout.read().strip()
            values = output.split(' ')
            if len(values) > 2:
                t = "%f" % time()
                pkts, bytes = values[0], values[1]
                open(fname, 'a').write(','.join([t, pkts, bytes]) + '\n')
            sleep(interval_sec)
        return

def monitor_devs(dev_pattern='^s', fname="%s/bytes_sent.txt" %
                    default_dir, interval_sec=0.01):

        """Aggregates (sums) all txed bytes and rate (in Mbps) from
           devices whose name matches @dev_pattern and writes to
    @fname"""
        pat = re.compile(dev_pattern)
        spaces = re.compile('\s+')
        open(fname, 'w').write('')
        prev_tx = {}
        while 1:
```

```
            lines = open('/proc/net/dev').read().split('\n')
            t = str(time())
            total = 0
            for line in lines:
                line = spaces.split(line.strip())
                iface = line[0]
                if pat.match(iface) and len(line) > 9:
                    tx_bytes = int(line[9])
                    total += tx_bytes - prev_tx.get(iface, tx_bytes)
                    prev_tx[iface] = tx_bytes
            open(fname, 'a').write(','.join([t,
                 str(total * 8 / interval_sec / 1e6), str(total)]) +
"\n")
            sleep(interval_sec)
        return


def monitor_devs_ng(fname="%s/txrate.txt" % default_dir,
interval_sec=0.01):
    """Uses bwm-ng tool to collect iface tx rate stats.  Very
reliable."""
    cmd = ("sleep 1; bwm-ng -t %s -o csv "
           "-u bits -T rate -C ',' > %s" %
           (interval_sec * 1000, fname))
    Popen(cmd, shell=True).wait()


def monitor_cpu(fname="%s/cpu.txt" % default_dir):
    cmd = "(top -b -p 1 -d 1 | grep --line-buffered \"^Cpu\") > %s" %
fname
    # BL: Disabling until we reinstantiate attachment using setns.
    #if container is not None:
    #    cmd = ("(top -b -p 1 -d 1 | "
    #           "grep --line-buffered \\\"^Cpu\\\") > %s" % fname)
```

```
    #    cmd = "lxc-execute -n %s -- bash -c \"%s\"" % (container,
cmd)
    Popen(cmd, shell=True).wait()
```

plot_defaults.py

```
#!/usr/bin/python

"CS144 In-class exercise: Buffer Bloat"

from mininet.topo import Topo
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.net import Mininet
from mininet.log import lg, info
from mininet.util import dumpNodeConnections
from mininet.cli import CLI
from monitor import monitor_qlen

from subprocess import Popen, PIPE
from time import sleep, time
from multiprocessing import Process
from argparse import ArgumentParser

import sys
import os

# Parse arguments

parser = ArgumentParser(description="BufferBloat tests")
```

```
parser.add_argument('--bw-host', '-B',
                    dest="bw_host",
                    type=float,
                    action="store",
                    help="Bandwidth of host links",
                    required=True)

parser.add_argument('--bw-net', '-b',
                    dest="bw_net",
                    type=float,
                    action="store",
                    help="Bandwidth of network link",
                    required=True)
parser.add_argument('--delay',
                    dest="delay",
                    type=float,
                    help="Delay in milliseconds of host links",
                    default=10)

parser.add_argument('--dir', '-d',
                    dest="dir",
                    action="store",
                    help="Directory to store outputs",
                    default="results",
                    required=True)

parser.add_argument('-n',
                    dest="n",
                    type=int,
                    action="store",
```

```
                    help="Number of nodes in star.",
                    required=True)

parser.add_argument('--nflows',
                    dest="nflows",
                    action="store",
                    type=int,
                    help="Number of flows per host (for TCP)",
                    required=True)

parser.add_argument('--maxq',
                    dest="maxq",
                    action="store",
                    help="Max buffer size of network interface in
packets",
                    default=500)

parser.add_argument('--cong',
                    dest="cong",
                    help="Congestion control algorithm to use",
                    default="reno")
parser.add_argument('--diff',
                    help="Enabled differential service",
                    action='store_true',
                    dest="diff",
                    default=False)

# Expt parameters
args = parser.parse_args()
```

```
class StarTopo(Topo):
    "Star topology for Buffer Bloat experiment"

    def __init__(self, n=2, cpu=None, bw_host=1000, bw_net=1.5,
                 delay=10, maxq=None, diff=False):
        # Add default members to class.
        super(StarTopo, self ).__init__()

        # Create switch and host nodes
        for i in xrange(n):
            self.addHost( 'h%d' % (i+1), cpu=cpu )



        self.addSwitch('s0', fail_mode='open')



        self.addLink('h1', 's0', bw=bw_host,
                     max_queue_size=int(maxq) )

        for i in xrange(1, n):
            self.addLink('h%d' % (i+1), 's0', bw=bw_host)

def ping_latency(net):
    "(Incomplete) verify link latency"
    h1 = net.getNodeByName('h1')
    h1.sendCmd('ping -c 2 10.0.0.2')
    result = h1.waitOutput()
    print "Ping result:"
    print result.strip()
```

```
def bbnet():
    "Create network and run Buffer Bloat experiment"
    print "starting mininet ...."
    # Seconds to run iperf; keep this very high
    seconds = 3600
    start = time()
    # Reset to known state
    topo = StarTopo(n=args.n, bw_host=args.bw_host,
                    delay='%sms' % args.delay,
                    bw_net=args.bw_net, maxq=args.maxq,
diff=args.diff)
    net = Mininet(topo=topo, host=CPULimitedHost, link=TCLink,
                  autoPinCpus=True)
    net.start()
    dumpNodeConnections(net.hosts)
    net.pingAll()
    print args.diff
    if args.diff:
        print "Differentiate Traffic Between iperf and wget"
        os.system("bash tc_cmd_diff.sh")
    else:
        print "exec tc_cmd.sh"
        os.system("bash tc_cmd.sh %s" % args.maxq)
    sleep(2)
    ping_latency(net)
    print "Initially, the delay between two hosts is around %dms" %
(int(args.delay)*2)
    h2 = net.getNodeByName('h2')
    h1 = net.getNodeByName('h1')
    h1.cmd('cd ./http/; nohup python2.7 ./webserver.py &')
    h1.cmd('cd ../')
```

```
    h2.cmd('iperf -s -w 16m -p 5001 -i 1 > iperf-recv.txt &')
    CLI( net )
    h1.cmd("sudo pkill -9 -f webserver.py")
    h2.cmd("rm -f index.html*")
    Popen("killall -9 cat", shell=True).wait()

if __name__ == '__main__':
    bbnet()
```

plot_queue.py

```
'''
Plot queue occupancy over time
'''
from helper import *
import plot_defaults

plot_defaults.quarter_size()

from matplotlib.ticker import MaxNLocator
from pylab import figure


parser = argparse.ArgumentParser()
parser.add_argument('--files', '-f',
                    help="Queue timeseries output to one plot",
                    required=True,
                    action="store",
                    nargs='+',
                    dest="files")
```

```
parser.add_argument('--maxy',
                    help="Max mbps on y-axis..",
            type=int,
                    default=500,
                    action="store",
                    dest="maxy")

parser.add_argument('--miny',
                    help="Min mbps on y-axis..",
            type=int,
                    default=0,
                    action="store",
                    dest="miny")

parser.add_argument('--legend', '-l',
                    help="Legend to use if there are multiple plots.
File names used as default.",
                    action="store",
                    nargs="+",
                    default=None,
                    dest="legend")

parser.add_argument('--out', '-o',
                    help="Output png file for the plot.",
                    default=None, # Will show the plot
                    dest="out")

parser.add_argument('-s', '--summarise',
                    help="Summarise the time series plot (boxplot).
First 10 and last 10 values are ignored.",
```

```
                    default=False,
                    dest="summarise",
                    action="store_true")

parser.add_argument('--cdf',
                    help="Plot CDF of queue timeseries (first 10 and
last 10 values are ignored)",
                    default=False,
                    dest="cdf",
                    action="store_true")

parser.add_argument('--labels',
                    help="Labels for x-axis if summarising; defaults
to file names",
                    required=False,
                    default=[],
                    nargs="+",
                    dest="labels")

parser.add_argument('--every',
                    help="If the plot has a lot of data points, plot
one every EVERY (x,y) point (default 1).",
                    default=1,
                    type=int)

args = parser.parse_args()
if args.labels is None:
    args.labels = args.files

if args.legend is None:
    args.legend = []
    for file in args.files:
```

```
        args.legend.append(file)


to_plot=[]
def get_style(i):
    if i == 0:
        return {'color': 'red'}
    else:
        return {'color': 'black', 'ls': '-.'}


print args.files
m.rc('figure', figsize=(16, 6))
fig = figure()
ax = fig.add_subplot(111)
for i, f in enumerate(args.files):
    data = read_list(f)
    xaxis = map(float, col(0, data))
    start_time = xaxis[0]
    xaxis = map(lambda x: x - start_time, xaxis)
    qlens = map(float, col(1, data))


    if args.summarise or args.cdf:
        to_plot.append(qlens[10:-10])
    else:
        xaxis = xaxis[::args.every]
        qlens = qlens[::args.every]
        ax.plot(xaxis, qlens, label=args.legend[i], lw=2,
**get_style(i))


    ax.xaxis.set_major_locator(MaxNLocator(4))
```

```
#plt.title("Queue sizes")
plt.title("")
plt.ylabel("Packets")
plt.grid(True)
#yaxis = range(0, 1101, 50)
#ylabels = map(lambda y: str(y) if y%100==0 else '', yaxis)
#plt.yticks(yaxis, ylabels)
#plt.ylim((0,1100))
plt.ylim((args.miny,args.maxy))

if args.summarise:
    plt.xlabel("Link Rates")
    plt.boxplot(to_plot)
    xaxis = range(1, 1+len(args.files))
    plt.xticks(xaxis, args.labels)
    for x in xaxis:
        y = pc99(to_plot[x-1])
        print x, y
        if x == 1:
            s = '99pc: %d' % y
            offset = (-20,20)
        else:
            s = str(y)
            offset = (-10, 20)
        plt.annotate(s, (x,y+1), xycoords='data',
                xytext=offset, textcoords='offset points',
                arrowprops=dict(arrowstyle="->"))
```

```
elif args.cdf:
    fig = figure()
    ax = fig.add_subplot(111)
    for i,data in enumerate(to_plot):
        xs, ys = cdf(map(int, data))
        ax.plot(xs, ys, label=args.legend[i], lw=2, **get_style(i))
        plt.ylabel("Fraction")
        plt.xlabel("Packets")
        plt.ylim((0, 1.0))
        #plt.legend(args.legend, loc="upper left")
        plt.title("")
        ax.xaxis.set_major_locator(MaxNLocator(4))
else:
    plt.xlabel("Seconds")
    #if args.legend:
    #    plt.legend(args.legend, loc="upper left")
    #else:
    #    plt.legend(args.files)

if args.out:
    plt.savefig(args.out)
else:
    plt.show()
```

plot_tcpprobe.py

```
from helper import *
from collections import defaultdict
import argparse

parser = argparse.ArgumentParser()
```

```
parser.add_argument('--sport', help="Enable the source port filter
(Default is dest port)", action='store_true', dest="sport",
default=False)

parser.add_argument('-p', '--port', dest="port", default='5001')

parser.add_argument('-f', dest="files", nargs='+', required=True)

parser.add_argument('-o', '--out', dest="out", default=None)

parser.add_argument('-H', '--histogram', dest="histogram",

                    help="Plot histogram of sum(cwnd_i)",

                    action="store_true",

                    default=False)


args = parser.parse_args()


def first(lst):

    return map(lambda e: e[0], lst)


def second(lst):

    return map(lambda e: e[1], lst)


"""

Sample line:

2.221032535 10.0.0.2:39815 10.0.0.1:5001 32 0x1a2a710c 0x1a2a387c 11
2147483647 14592 85

"""

def parse_file(f):

    times = defaultdict(list)

    cwnd = defaultdict(list)

    srtt = []

    for l in open(f).xreadlines():

        fields = l.strip().split(' ')

        #print len(fields)

        if len(fields) < 10:
```

```
            break
        #print "using sport? %s" % args.sport
        if not args.sport:
            #print "using dport %s (compare with %s)" % (args.port,
fields[1].split(':')[1])
            if fields[2].split(':')[1] != args.port:
                continue
        else:
            #print "using sport %s (compare with %s)" % (args.port,
fields[1].split(':')[1])
            if fields[1].split(':')[1] != args.port:
                continue
        sport = int(fields[1].split(':')[1])
        times[sport].append(float(fields[0]))

        c = int(fields[6])
        cwnd[sport].append(c * 1480 / 1024.0)
        srtt.append(int(fields[-1]))
    return times, cwnd

added = defaultdict(int)
events = []

def plot_cwnds(ax):
    global events
    for f in args.files:
        times, cwnds = parse_file(f)
        for port in sorted(cwnds.keys()):
            t = times[port]
            cwnd = cwnds[port]
```

```
            events += zip(t, [port]*len(t), cwnd)
            ax.plot(t, cwnd)

    events.sort()
total_cwnd = 0
cwnd_time = []

min_total_cwnd = 10**10
max_total_cwnd = 0
totalcwnds = []

m.rc('figure', figsize=(16, 6))
fig = plt.figure()
plots = 1
if args.histogram:
    plots = 2

axPlot = fig.add_subplot(1, plots, 1)
plot_cwnds(axPlot)

for (t,p,c) in events:
    if added[p]:
        total_cwnd -= added[p]
    total_cwnd += c
    cwnd_time.append((t, total_cwnd))
    added[p] = c
    totalcwnds.append(total_cwnd)

axPlot.plot(first(cwnd_time), second(cwnd_time), lw=2, label="$\sum_i
W_i$")
axPlot.grid(True)
```

```
#axPlot.legend()

axPlot.set_xlabel("seconds")

axPlot.set_ylabel("cwnd KB")

axPlot.set_title("TCP congestion window (cwnd) timeseries")


if args.histogram:

    axHist = fig.add_subplot(1, 2, 2)

    n, bins, patches = axHist.hist(totalcwnds, 50, normed=1,
facecolor='green', alpha=0.75)


    axHist.set_xlabel("bins (KB)")

    axHist.set_ylabel("Fraction")

    axHist.set_title("Histogram of sum(cwnd_i)")


if args.out:

    print 'saving to', args.out

    plt.savefig(args.out)

else:

    plt.show()
```

iperf.sh

```
#!/bin/bash

#qsize=$1

iperf -c 10.0.0.2 -p 5001 -t 3600 -i 1 -w 16m -Z reno > iperf.txt &

echo started iperf
```

tc_cmd_diff.sh

```
#!/bin/bash


rate=1.5Mbit
```

```
rate2=0.75Mbit


function add_qdisc {
    dev=$1
    tc qdisc del dev $dev root
    echo qdisc removed


    tc qdisc add dev $dev root handle 1: htb default 1
    echo qdisc added


    tc class add dev $dev classid 1:1 parent 1: htb rate $rate
    tc class add dev $dev classid 1:10 parent 1:1 htb rate $rate ceil
$rate
    tc class add dev $dev classid 1:11 parent 1:1 htb rate $rate ceil
$rate
    echo classes created


    tc qdisc add dev $dev parent 1:10 handle 10: netem delay 10ms
limit 100
    tc qdisc add dev $dev parent 1:11 handle 11: netem delay 10ms
limit 100


    # Direct iperf traffic to classid 10:1
    tc filter add dev $dev protocol ip parent 1: prio 1 u32 match ip
dport 5001 0xffff flowid 1:10
    tc filter add dev $dev protocol ip parent 1: prio 1 u32 match ip
sport 5001 0xffff flowid 1:10
    tc filter add dev $dev protocol ip parent 1: prio 1 u32 match ip
protocol 1 0xff flowid 1:11
    tc filter add dev $dev protocol ip parent 1: prio 1 u32 match ip
dport 80 0xffff flowid 1:11
    tc filter add dev $dev protocol ip parent 1: prio 1 u32 match ip
sport 80 0xffff flowid 1:11
    echo filters added
```

```
}

add_qdisc s0-eth1
add_qdisc s0-eth2
```

tc_cmd.sh

```
#!/bin/bash

qlen=$1
rate=1.5Mbit
rate2=1.5Mbit

function add_qdisc {
    dev=$1
    tc qdisc del dev $dev root
    echo qdisc removed

    tc qdisc add dev $dev root handle 1:0 htb default 1
    echo qdisc added

    tc class add dev $dev parent 1:0 classid 1:1 htb rate $rate ceil
$rate
    echo classes created

    tc qdisc add dev $dev parent 1:1 handle 10: netem delay 10ms
limit $qlen

    echo delay added
}

add_qdisc s0-eth1
add_qdisc s0-eth2
```

Παράρτημα Γ

Εικόνες & Γραφήματα

Γ.1 Κατάλογος εικόνων

Γ.2 Κατάλογος Γραφημάτων

Παράρτημα Δ
Βοηθός Εγκατάστασης - Mininet

Ανάλογα με το λειτουργικό σύστημα που έχει ο υπολογιστή μας θα πρέπει να εγκαταστήσουμε τις αντίστοιχες εφαρμογές που προτείνει ο παρακάτω πίνακας ώστε να επιτύχουμε σωστή λειτουργία του Mininet εξομοιωτή:

OS Type	OS Version	Virtualization Software	X Server	Terminal
Windows	7+	VirtualBox	Xming	PuTTY
Windows	XP	VirtualBox	Xming	PuTTY
Mac	OS X 10.7-10.8 Lion/Mountain Lion	VirtualBox	download and install XQuartz	Terminal.app (built in)
Mac	OS X 10.5-10.6 Leopard/Snow Leopard	VirtualBox	X11 (install from OS X main system DVD, preferred), or download XQuartz	Terminal.app (built in)
Linux	Ubuntu 10.04+	VirtualBox	X server already installed	gnome terminal + SSH built in

Εικόνα Δ. 94 Απαιτούμενες εφαρμογές ανά λειτουργικό σύστημα ώστε να λειτουργεί σωστά το Mininet. [22]

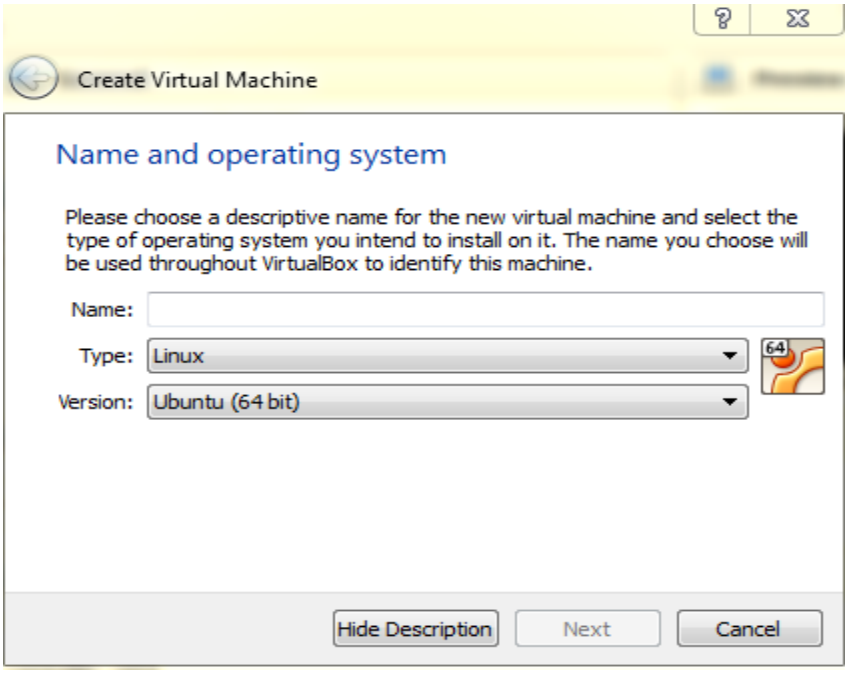
Δ.1 Βήματα Εγκατάστασης Mininet

Τα βήματα για την εγκατάσταση του Mininet που ακολουθήθηκαν σε αυτήν την διατριβή δεδομένου ότι ο υπολογιστής που έχουμε στην κατοχή μας έχει λειτουργικό Windows 7 είναι τα παρακάτω:

1. Σαν αρχή επισκεπτόμαστε την σελίδα:
   http://mininet.org/download/ [46]
   και ακολουθούμε τα βήματα της προτεινόμενης εγκατάστασης, τα βήματα αυτά αναλύονται με λεπτομέρεια στην συνέχεια.
2. Πρώτα κατεβάζουμε την εικόνα μιας εικονικής μηχανής η οποία θα περιέχει και τον εξομοιωτή Mininet. Εικόνα του Mininet μπορούμε να αποκτήσουμε μέσω του github στην σελίδα:
   https://github.com/mininet/mininet/wiki/Mininet-VM-Images
3. Στην παραπάνω σελίδα επιλέχθηκε η έκδοση: Mininet 2.2. 0 Ubuntu 14.04 - 64 bit
4. Στην συνέχεια κατεβάζουμε το winrar αρχείο με το σύστημα "mininet-2.2.0-141209-ubuntu-14.04-server-amd64.zip" το οποίο ουσιαστικά προέρχεται από την σελίδα http://onlab.vicci.org/mininet-vm/mininet-2.2.0-141209-ubuntu-14.04-server-amd64.zip και κάνουμε extract το αρχείο στην επιθυμητή τοποθεσία του σκληρού μας.
5. Εγκαθιστούμε ένα σύστημα εικονοποίησης, συγκεκριμένα επιλέχθηκε το Virtualbox 4.3.18 επιλέγοντας x86/AMD64 που τον Οκτώβρη του 2014 ήταν η πιο πρόσφατη έκδοση, κατά την στιγμή συγγραφής της διατριβής υπάρχει ήδη διαθέσιμη η έκδοση 4.3.20, άρα για να εγκαταστήσει κάποιος την παλιά έκδοση πρέπει να επισκεφτεί τον σύνδεσμο
   https://www.virtualbox.org/wiki/Download_Old_Builds
   και να επιλέξει την έκδοση που επιθυμεί. Ακολουθούμε τον οδηγό - wizard και δεν αλλάζουμε καμία από τις προκαθορισμένες ρυθμίσεις. Μπορεί να μας βγάλει μήνυμα για εγκατάσταση λοιπών στοιχείων όπως προσαρμογέα δικτύου, ελεγκτές και την υπηρεσία δικτύου της Oracle corporation στην περίπτωση αυτή επιλέγουμε το checkbox που αναφέρει "Να θεωρείτε πάντα αξιόπιστο το λογισμικό από Oracle Corporation" και συνεχίζουμε την εγκατάσταση. Επιλέγουμε "Τέλος" στην τελευταία εικόνα του οδηγού.
6. Μετά την εγκατάσταση του Virtualbox επιλέγουμε Oracle Vm - VirtualBox από την λίστα προγραμμάτων του υπολογιστή. (χρήσιμες πληροφορίες για την εγκατάσταση και λειτουργία του Virtualbox υπάρχουν στον παρακάτω σύνδεσμο: *http://dlc-cdn.sun.com/virtualbox/4.3.20/UserManual.pdf*)
7. Εμφανίζεται η παρακάτω εικόνα

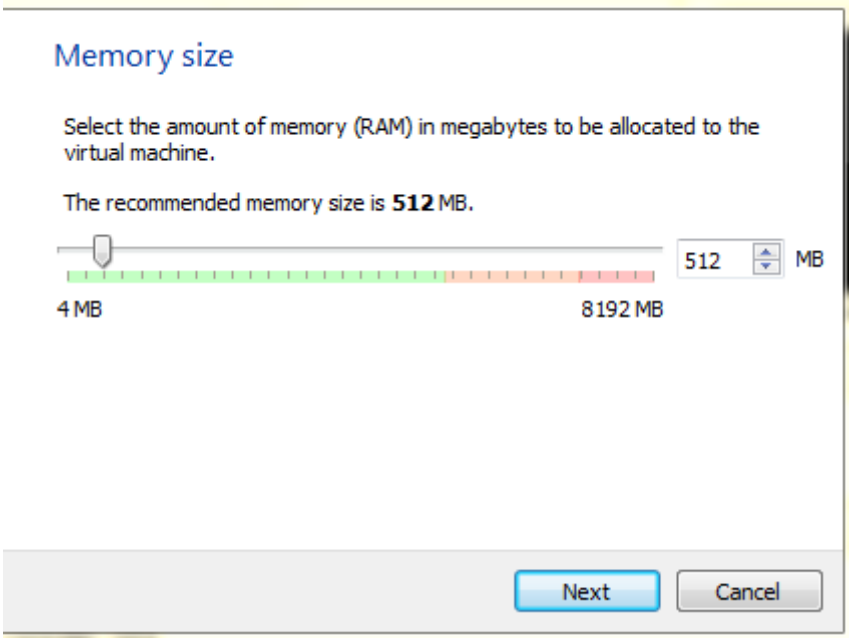
Εικόνα Δ. 95 Οδηγός Εγκατάστασης: Πρώτη οθόνη του VirtualBox

8. Επιλέγουμε New και εμφανίζεται ο οδηγός που μας προτρέπει να δημιουργήσουμε μια νέα εικονική μηχανή επιλέγουμε Version: Ubuntu 64 bit Type: Linux και δίνουμε ένα χαρακτηριστικό όνομα στην νέα εικονική μηχανή. Πατάμε "Next".



Εικόνα Δ. 96 Οδηγός Εγκατάστασης: Επιλογή λειτουργικού

9. Κατόπιν επιλέγουμε την ποσότητα της μνήμης RAM του συστήματος μας που θα αποδοθεί στην εικονική μηχανή και έτσι δημιουργούμε ένα εικονικό σκληρό δίσκο με τη RAM που επιθυμούμε.

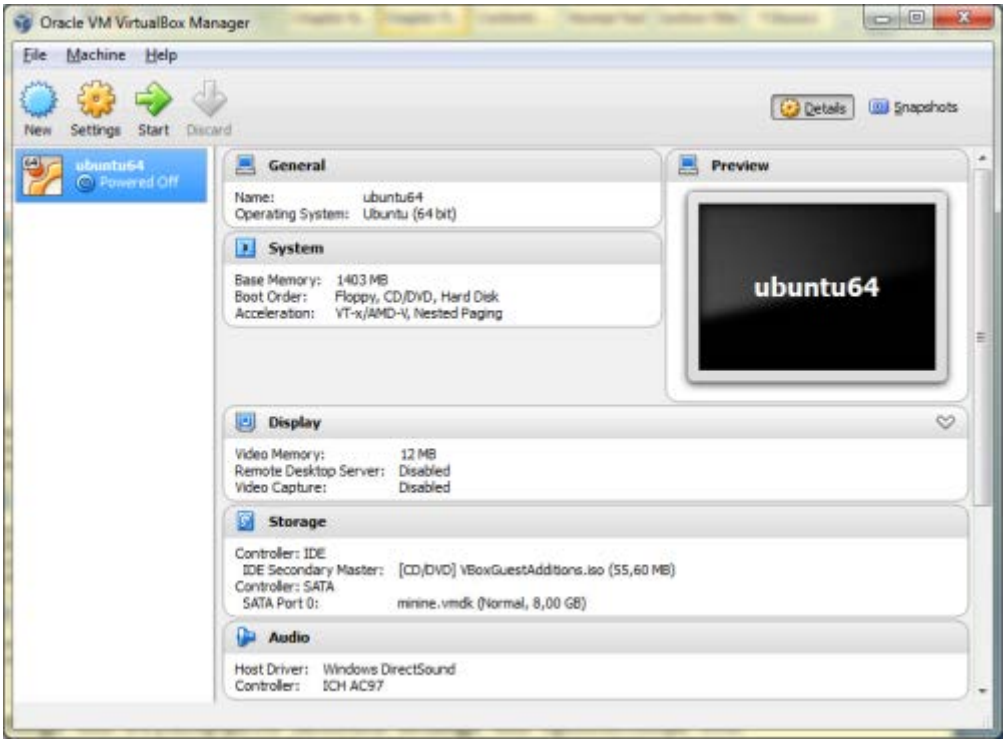


Εικόνα Δ. 97 Οδηγός Εγκατάστασης: Επιλογή μεγέθους μνήμης

10. Στο επόμενο βήμα επιλέγουμε το radio button: Use an existing virtual hard drive file.

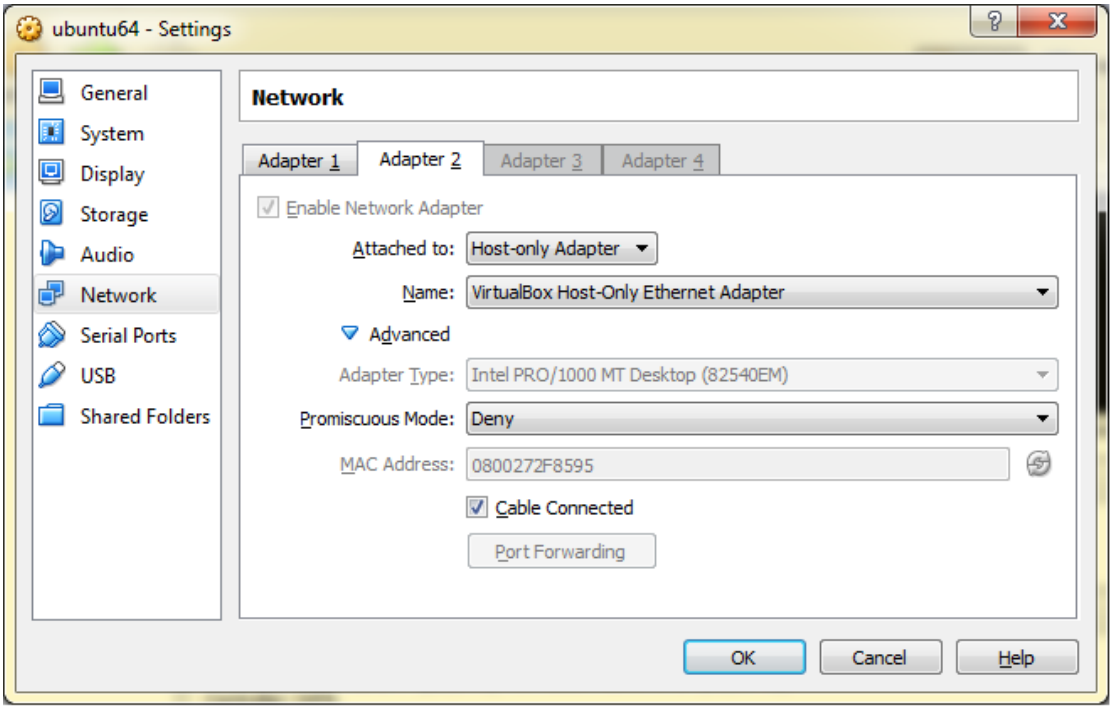
Εικόνα Δ. 98 Οδηγός Εγκατάστασης: Επιλογή σκληρού δίσκου

11. Επιλέγουμε το *.vmdk αρχείο που είχαμε κατεβάσει στο βήμα 4 και μετά πατάμε το πλήκτρο "Create". Εμφανίζεται ο Διαχειριστής του εικονικού συστήματος της Oracle.

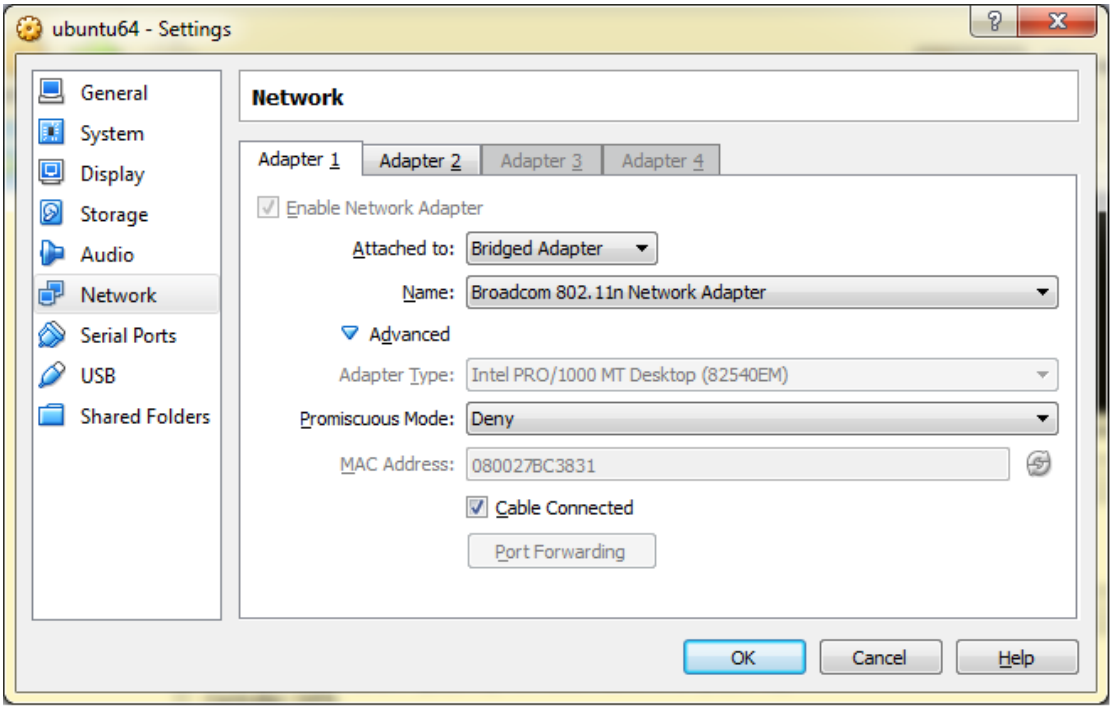


Εικόνα Δ. 99 Οδηγός Εγκατάστασης: Διαχειριστής VirtualBox

12. Επιλέγουμε "Settings" και συγκεκριμένα "Network settings" και προσθέτουμε ένα "Host only adapter" για να γίνεται εύκολα η επικοινωνία μεταξύ της εικονικής μηχανής και του υπολογιστή μας. Αν δεν φαίνεται η επιλογή "Host only adapter" ακολουθούμε την διαδρομή File menu/Preferences/Network και "Add host-only network" button με τις default ρυθμίσεις.



Εικόνα Δ. 100 Οδηγός Εγκατάστασης: Ρύθμιση Διεπαφής Δικτύου I



Εικόνα Δ. 101 Οδηγός Εγκατάστασης: Ρύθμιση Διεπαφής Δικτύου II

13. Στον Adapter 1 θα πρέπει να έχουμε μια εικόνα σαν την παραπάνω. Καθορίζουμε δηλαδή έναν "Bridged Adapter". Αφήνουμε τις προκαθορισμένες ρυθμίσεις.

14. Στην συνέχεια θα πρέπει εφόσον έχουμε λειτουργικό σύστημα Windows να κατεβάσουμε το "Putty.exe" από την παρακάτω διεύθυνση

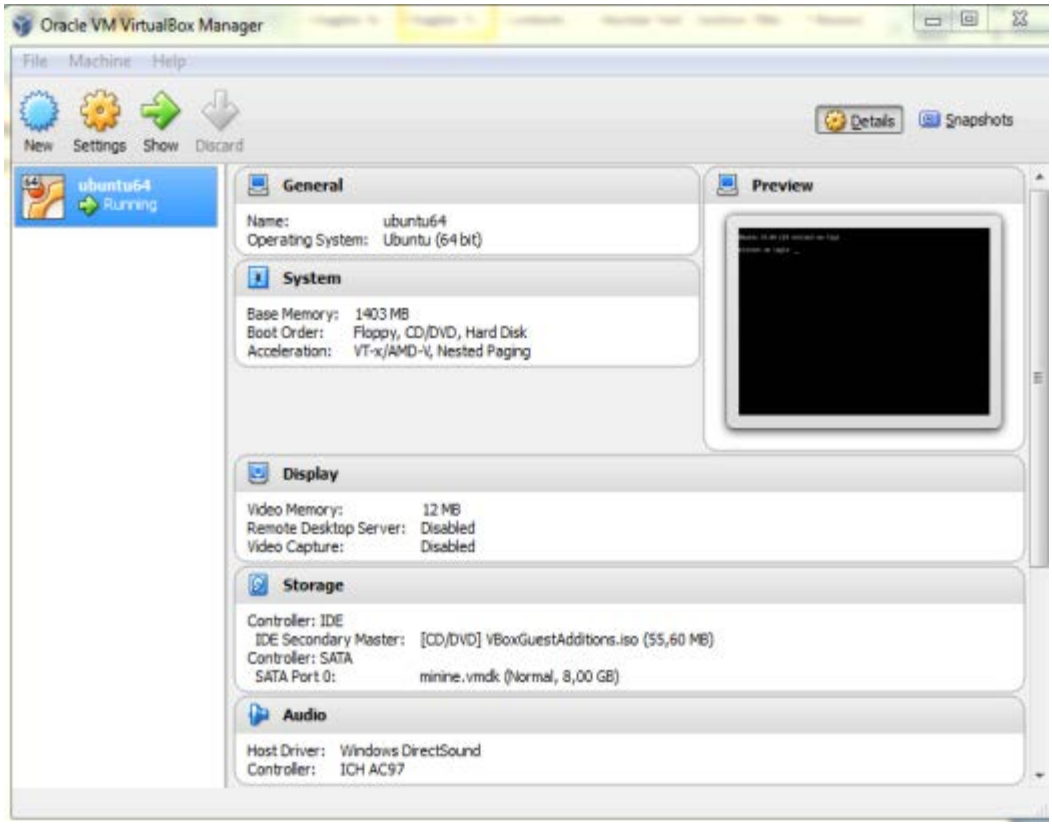
    http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

    καθώς και τον "Xming server" αντίστοιχα από

    http://sourceforge.net/projects/xming/ για να μπορέσει να λειτουργήσει σωστά το Mininet και να είναι πιο εύκολη η δημιουργία πειραμάτων.

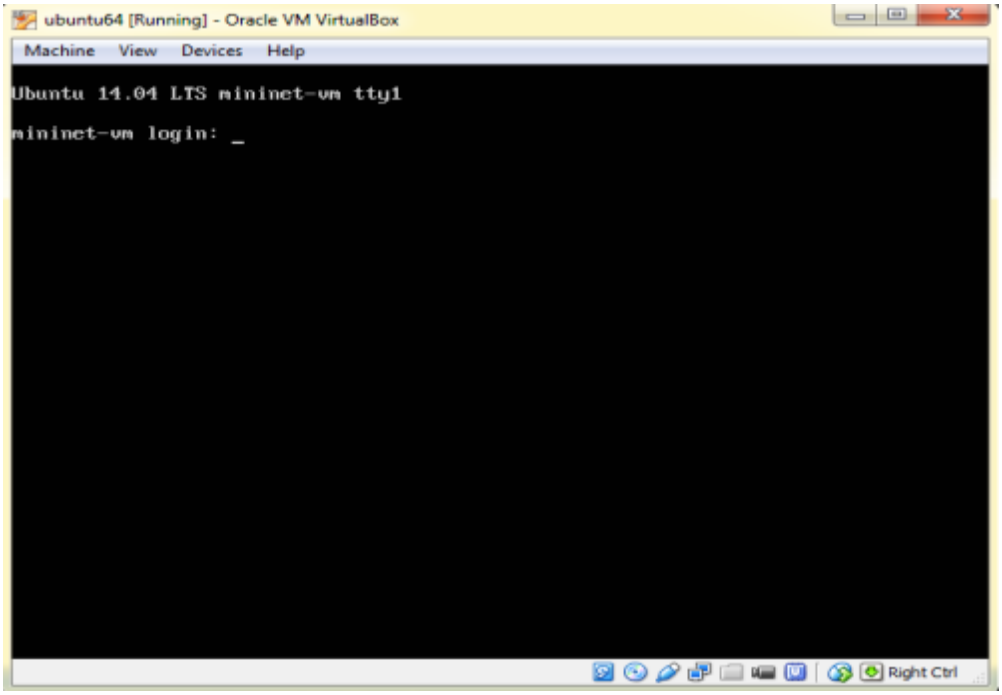
15. Εγκαθιστούμε το "Putty" και το "Xming Server". (Το δεύτερο φαίνεται ότι τρέχει στο σύστημα αν επιλέξουμε εμφάνιση κρυφών εικονιδίων.)

16. Κάνουμε "Start" την εικονική μηχανή που έχουμε δημιουργήσει. Αρχίζει το λειτουργικό σύστημα Ubuntu να εμφανίζεται στην οθόνη μας.



Εικόνα Δ. 102 Οδηγός Εγκατάστασης: Διαχειριστής VirtualBox

17. Στο επόμενο βήμα έχουμε την οθόνη login του Mininet στο περιβάλλον των Ubuntu.

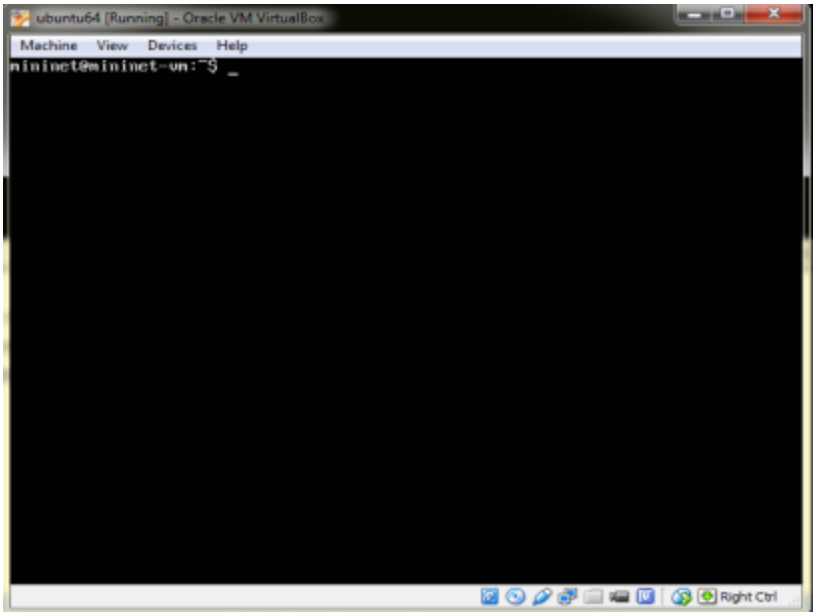


Εικόνα Δ. 103 Οδηγός Εγκατάστασης: Οθόνη login Mininet

Δ.2 Εικονική Μηχανή Mininet

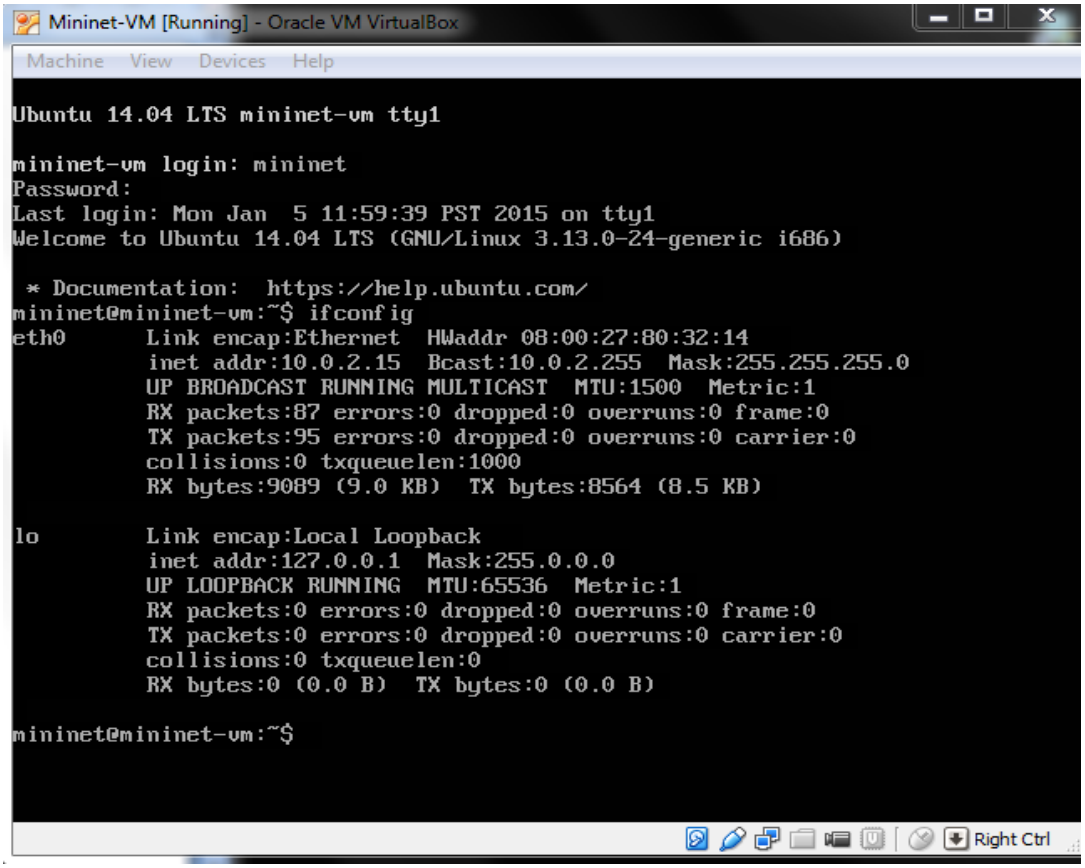
1. Εισάγω το username: mininet και τον κωδικό: mininet (Να σημειώσω εδώ ότι η εισαγωγή του κωδικού δεν φαίνεται για λόγους ασφάλειας, οπότε απλά πατάμε μετά την εισαγωγή, enter).

2. Είμαστε πλέον στην εικονική μηχανή του Mininet και θα πρέπει να έχουμε την παρακάτω εικόνα.



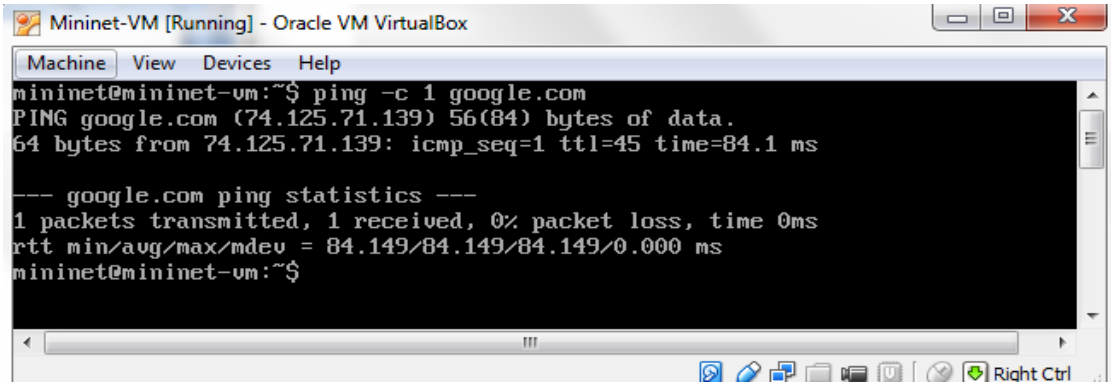
Εικόνα Δ. 104 Οδηγός Εγκατάστασης: Οθόνη Mininet

3. Δίνουμε την εντολή **ifconfig** για να δούμε τις διεπαφές που είναι διαθέσιμες στην εικονική μηχανή του Mininet. Όπως παρατηρούμε και στο επόμενο σχήμα βλέπουμε δυο διεπαφές eht0 και lo (loopback interface). Έχουμε από προηγουμένως καθορίσει στην διεπαφή της εικονικής μηχανής και στη διεπαφή του υπολογιστή μας να επικοινωνούν μεταξύ τους, στα βήματα 12 και 13. Συνεπώς έχει αποδοθεί μια IP διεύθυνση που τρέχει στο LAN στο οποίο συνδέεται το μηχάνημα μας. Όπως βλέπουμε η eth0 έχει την IP διεύθυνση: 10.0.2.15/24. Ενώ η loopback διεύθυνση είναι η 127.0.0.1. [35]



Εικόνα Δ. 105 Οδηγός Εγκατάστασης: Mininet εντολή ifconfig.

4. Με αυτόν τον τρόπο μπορεί η εικονική μηχανή να συνδεθεί στο διαδίκτυο. Μπορούμε να επιβεβαιώσουμε την σύνδεση αυτή κάνοντας ping σε ένα external server όπως την google.com. Δίνουμε λοιπόν την παρακάτω εντολή: mininet@mininet-vm**:~$ ping -c 1 google.com**



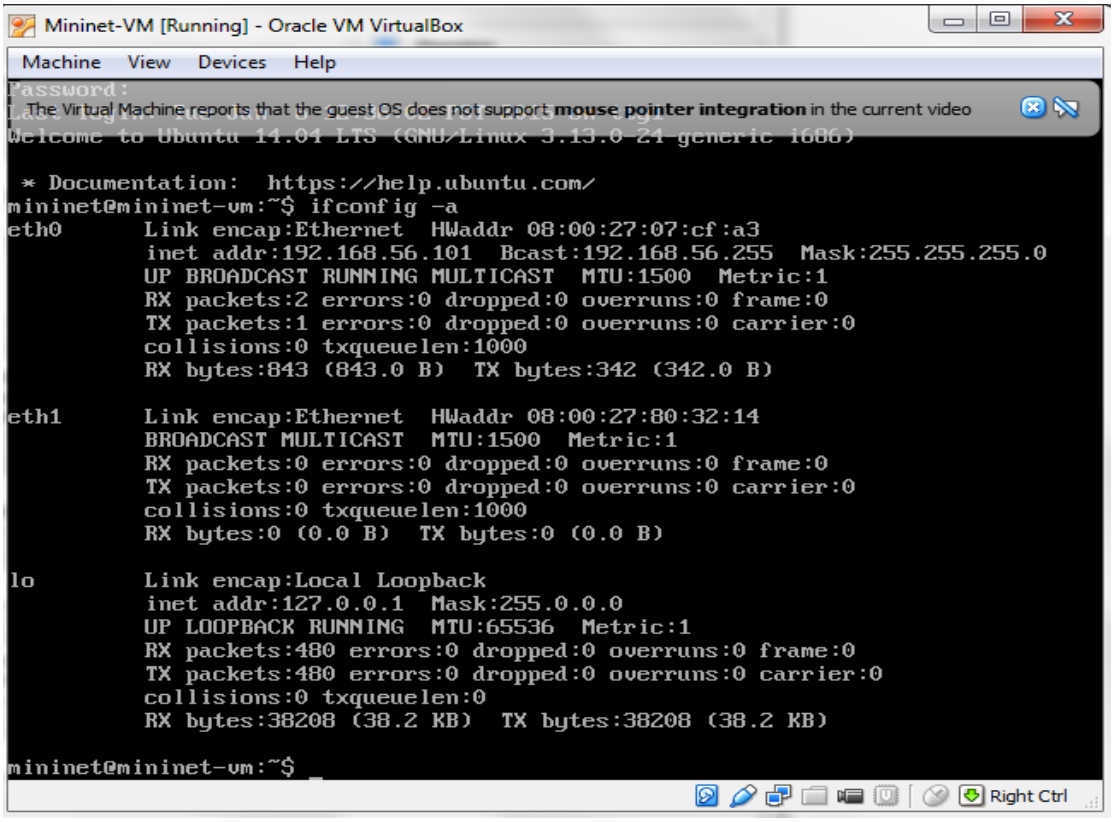
Εικόνα Δ. 106 Οδηγός Εγκατάστασης: Mininet εντολή ping.

Παρατηρούμε ότι τα πακέτα λαμβάνονται κανονικά.

5. Εάν ξαναδώσουμε **ifconfig** παρατηρούμε και πάλι ότι δεν βλέπουμε την διεπαφή eth1. Δηλαδή την διεπαφή που είναι συνδεδεμένη στο host only interface.

6. Για να δούμε όλες τις διεπαφές ακόμα και αυτές που δεν είναι up δίνουμε την εξής εντολή: mininet@mininet-vm:~$ **ifconfig -a** Παρατηρούμε τρείς διεπαφές lo, eth0, and eth1. Αλλά η διεπαφή eht1 δεν λειτουργεί και δεν έχει αντιστοιχηθεί σε μια IP.

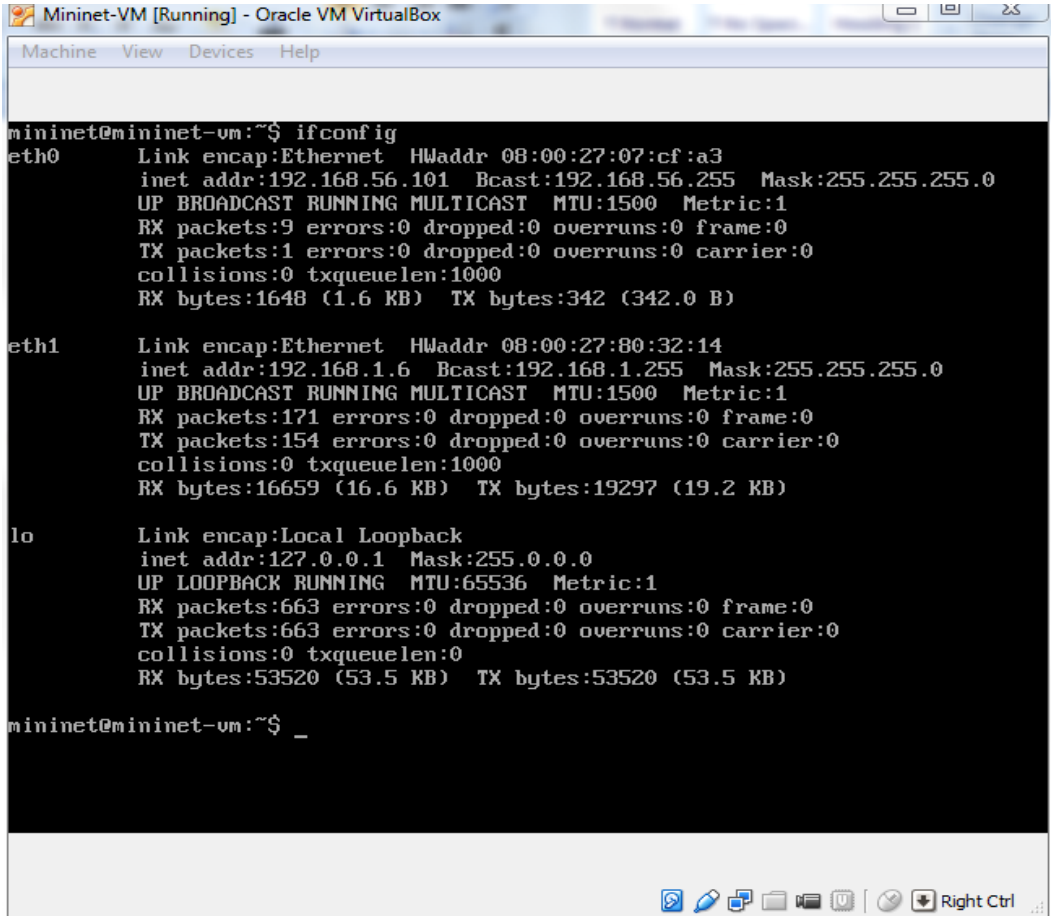
7. Μπορούμε να διορθώσουμε αυτό το σφάλμα αν ξεκινήσουμε ένα DHCP client στην διεπαφή eth1 ώστε να ζητήσει μια IP διεύθυνση από τον DHCP server που λειτουργεί στο host-only interface που συνδέεται με το eth1. [35]



Εικόνα Δ. 107 Οδηγός Εγκατάστασης: Mininet εντολή ifconfig -a

8. Δίνουμε την εντολή **mininet@mininet-vm:~$ sudo dhclient eth1**

9. Στην συνέχεια τρέχουμε πάλι την εντολή **ifconfig**. Παρατηρούμε ότι η διεπαφή eth1 έχει πλέον την IP διεύθυνση 192.168.1.6. [35]

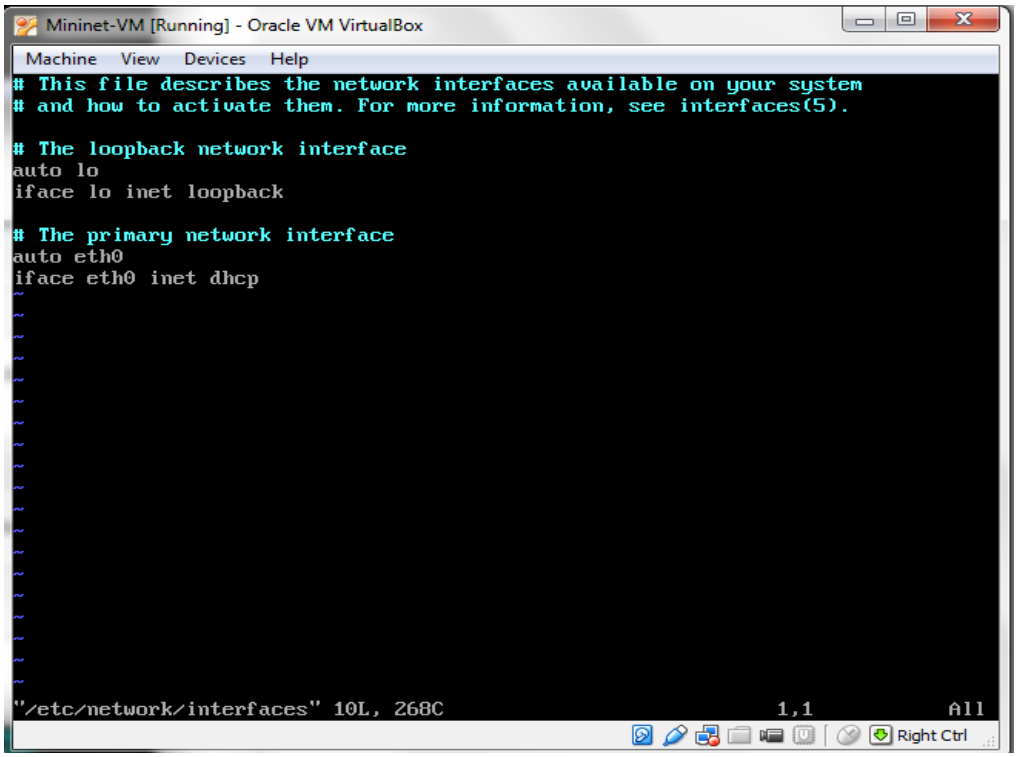


Εικόνα Δ. 108 Οδηγός Εγκατάστασης: Mininet εντολή ifconfig

10. Αν θέλουμε όμως να κάνουμε τα αποτελέσματα μόνιμα ώστε να μην κάνουμε αυτήν την διαδικασία κάθε φορά που τρέχουμε εκ νέου την εικονική μηχανή τότε θα πρέπει να αλλάξουμε το αρχείο των διεπαφών με το παρακάτω τρόπο και να προσθέσουμε σε αυτό την διεπαφή με όνομα eth1. Γράφουμε

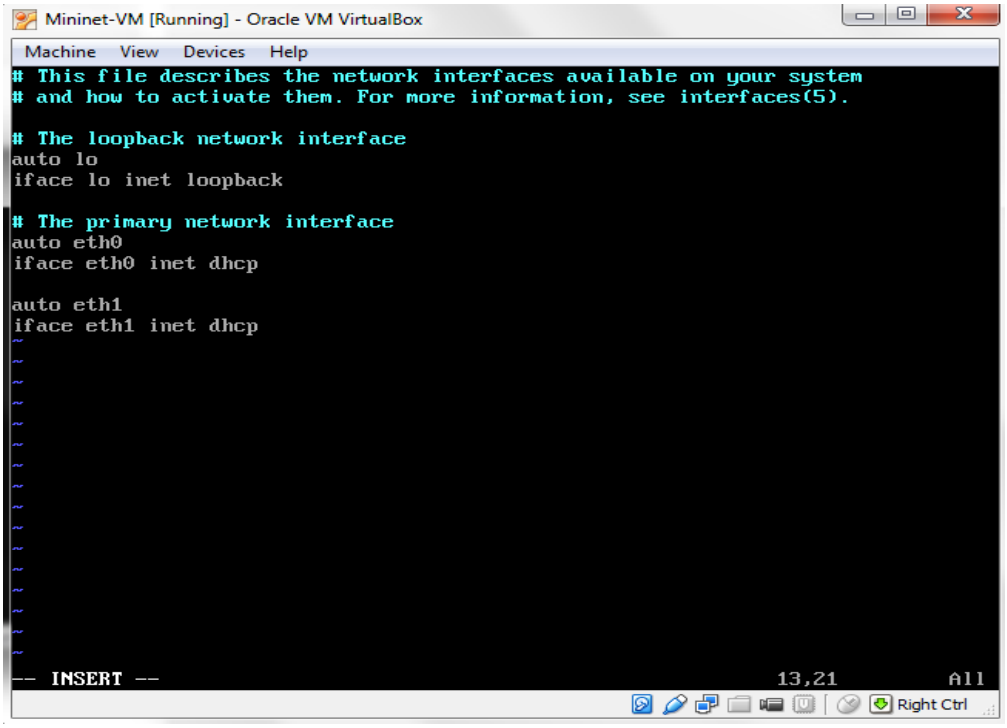
mininet@mininet-vm:~$ **sudo vi /etc/network/interfaces**

Ουσιαστικά αυτό τρέχει τον "Vi editor" που υπάρχει προκαθορισμένα στα Ubuntu και μετά την εντολή και το enter έχουμε την παρακάτω εικόνα:



Εικόνα Δ. 109 Οδηγός Εγκατάστασης: Mininet network interfaces

11. Για να εισάγουμε κείμενο στο αρχείο θα πρέπει να πατήσουμε το γράμμα "i" πλέον ο editor θα είναι σε mode "insert" όπως παρατηρούμε στην παρακάτω εικόνα. Περισσότερες πληροφορίες για τον vi editor μπορεί κανείς να αποκτήσει για παράδειγμα από σελίδες πανεπιστημίων. [25]



Εικόνα Δ. 110 Οδηγός Εγκατάστασης: Mininet διορθωμένες διεπαφές

12. Εισάγουμε τις παρακάτω δυο γραμμές στο αρχείο στην περιοχή των πρωτευόντων διεπαφών

"auto eth1

iface eth1 inet dhcp" [35]

πλέον το αρχείο έχει το παρακάτω κείμενο όπως φαίνεται και στην ακόλουθη εικόνα

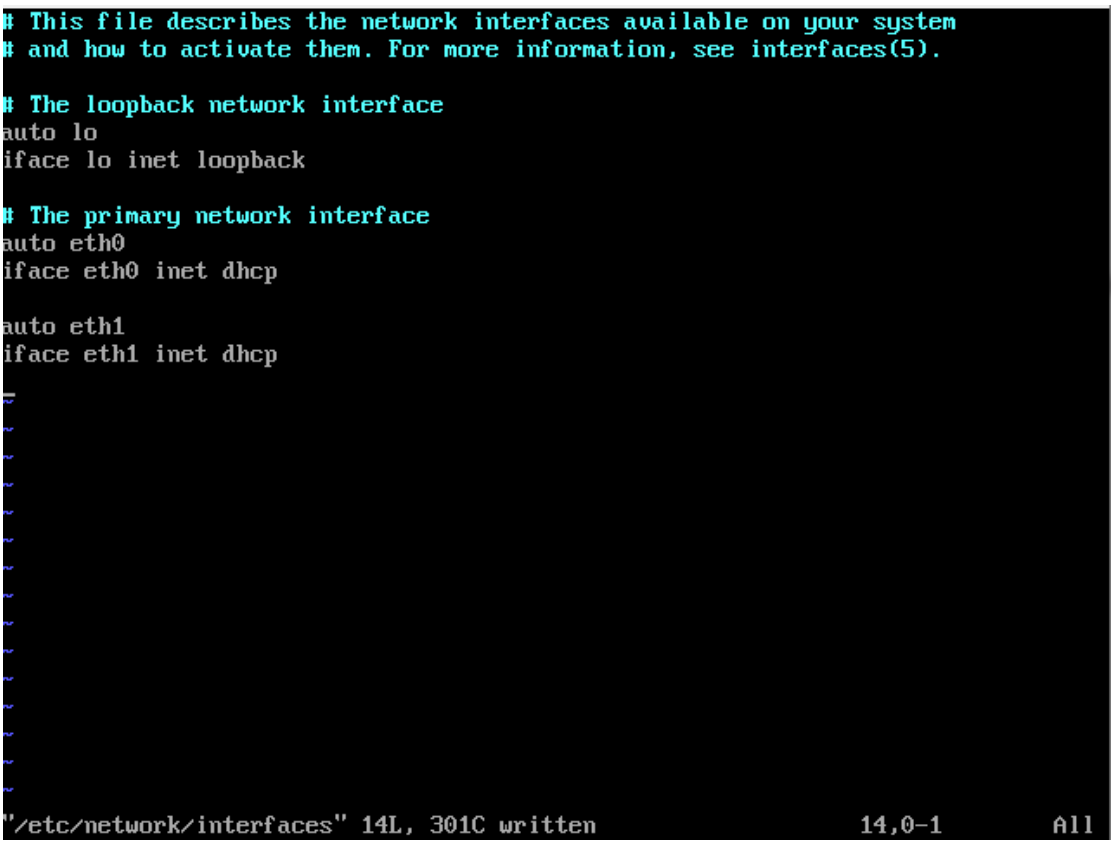
13. Αν θέλουμε να παραμείνει ανοικτός ο Vi editor για να δούμε τις αλλαγές που κάναμε πατάμε:

"escape

:w

και στην συνέχεια enter".

14. Παρατηρούμε την παρακάτω εικόνα



Εικόνα Δ. 111 Οδηγός Εγκατάστασης: Αποθήκευση διορθωμένων διεπαφών δικτύου

15. Κλείνουμε αν θέλουμε την εικονική μηχανή και την ξανά ανοίγουμε για να επιβεβαιώσουμε ότι οι αλλαγές που κάναμε είναι μόνιμου χαρακτήρα.

16. Δίνω ifcongig -a και βλέπω ότι παραμένει η IP 192.168.1.6 στο eth1

Η εγκατάσταση του Mininet έχει πλέον επιτευχθεί.

Δ.3 SSH Client

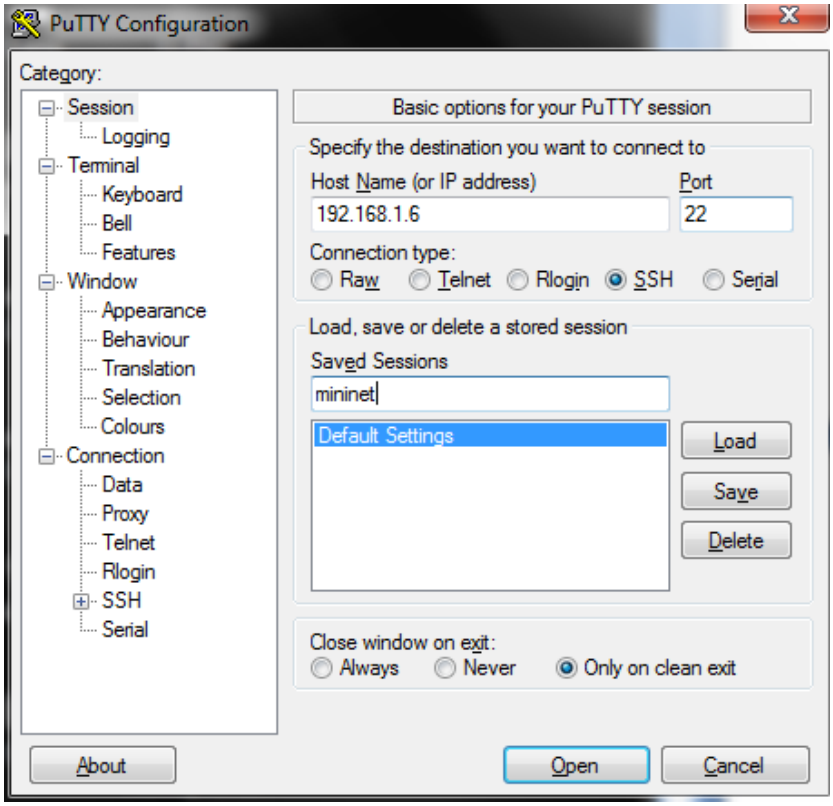
Στα επόμενα βήματα θα χρησιμοποιήσουμε ένα λογισμικό SSH client που θα συνδέεται με την εικονική μηχανή του Mininet. Αυτό το κάνουμε για τους εξής δυο λόγους:

I. Από τον υπολογιστή μας (τον υπολογιστή δηλαδή που φιλοξενεί την εικονική μηχανή) μπορούμε με αυτόν τον τρόπο να συνδεθούμε με εφαρμογές απομακρυσμένες - remote applications που τρέχουν στην εικονική μηχανή του Mininet. Τέτοιες εφαρμογές είναι το "Xterm" & το "Wireshark".

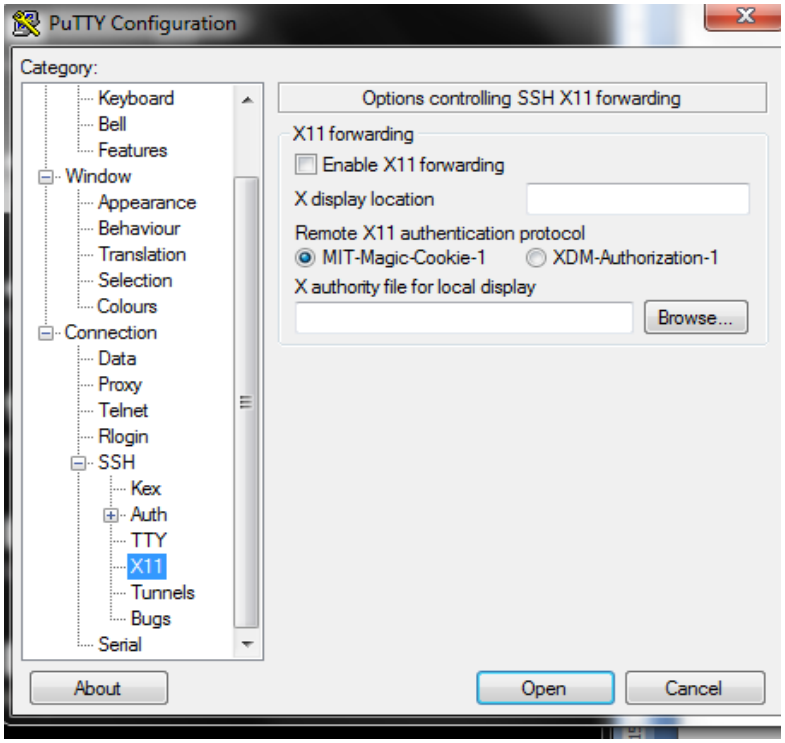
II. Μπορούμε να χρησιμοποιήσουμε ένα τερματικό παράθυρό είτε μέσα από το "Putty" καθαυτό είτε μέσα από το "Xterm - Xming server" ώστε να μπορούμε εύκολα να κάνουμε λειτουργίες όπως αντιγραφή και επικόλληση αλλά και να έχουμε άμεση πρόσβαση στο υπολογιστή μας γεγονός που δεν συμβαίνει στην περίπτωση που τρέχουμε κανονικά την εικονική μηχανή μέσα στο περιβάλλον του Virtualbox.

Εάν δημιουργήσουμε και εγκαταστήσουμε μια SSH σύνδεση στο virtual machine με το X11 forwarding enabled ώστε να μπορούμε να τρέχουμε X εφαρμογές στο Mininet virtual machine οι οποίες να εμφανίζονται στον "X Server" που τρέχει στον υπολογιστή μας και χρησιμοποιήσουμε για να γίνει αυτό, το "Xterm" και το "Putty", τότε θα μπορούμε να επιτύχουμε τα παραπάνω και να έχουμε πιο εύκολη για τον χρήστη λειτουργία της εικονικής μηχανής και του Mininet γενικότερα. Για να γίνουν τα παραπάνω μετά την εγκατάσταση του "Putty" και του "Xming server" που έλαβε χώρα στο βήμα 15 της 5. 1 ενότητας πρέπει να βεβαιωθούμε ότι τρέχουν τα δύο αυτά προγράμματα ώστε να ρυθμίσουμε τις παρακάτω παραμέτρους και το X11 να λειτουργεί ορθά.

1. Συγκεκριμένα όσον αφορά το "Putty" για να το ρυθμίσουμε ώστε να λειτουργεί σωστά θα πρέπει στο πεδίο host name να εισάγω την IP του eth1 δηλαδή 192.168.1.6 port 22 με connection type ssh. Επίσης στο πεδίο saved sessions εισάγω το όνομα mininet, έπειτα πηγαίνω στο SSH connection και επιλέγω X11

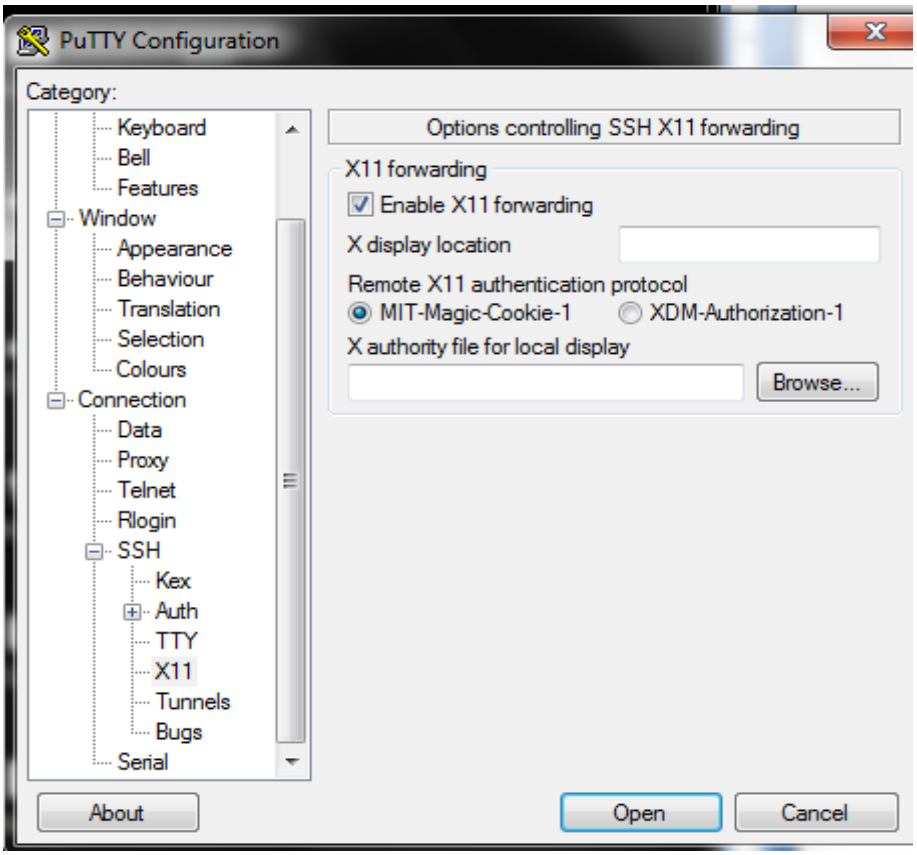


Εικόνα Δ. 112 Οδηγός Εγκατάστασης: Putty setup



Εικόνα Δ. 113 Οδηγός Εγκατάστασης: Putty X11 ρυθμίσεις [51] [21]

2. Κάνω check στο checkbox "Enable X11 Forwarding"

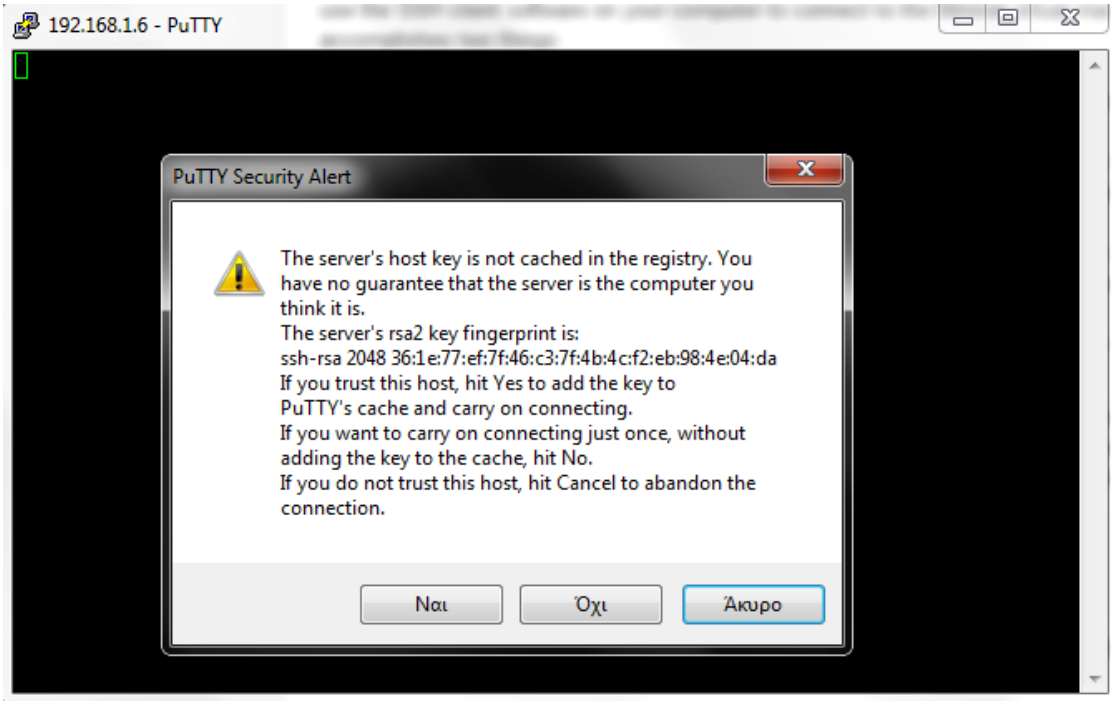


Εικόνα Δ. 114 Οδηγός Εγκατάστασης: Putty ρυθμίσεις

3. Πηγαίνω πίσω στο session και πατάω το πλήκτρο save.

4. Στο επόμενο βήμα επιλέγω την saved session με το όνομα mininet που έχουμε δημιουργήσει κάνοντας double click σε αυτήν.

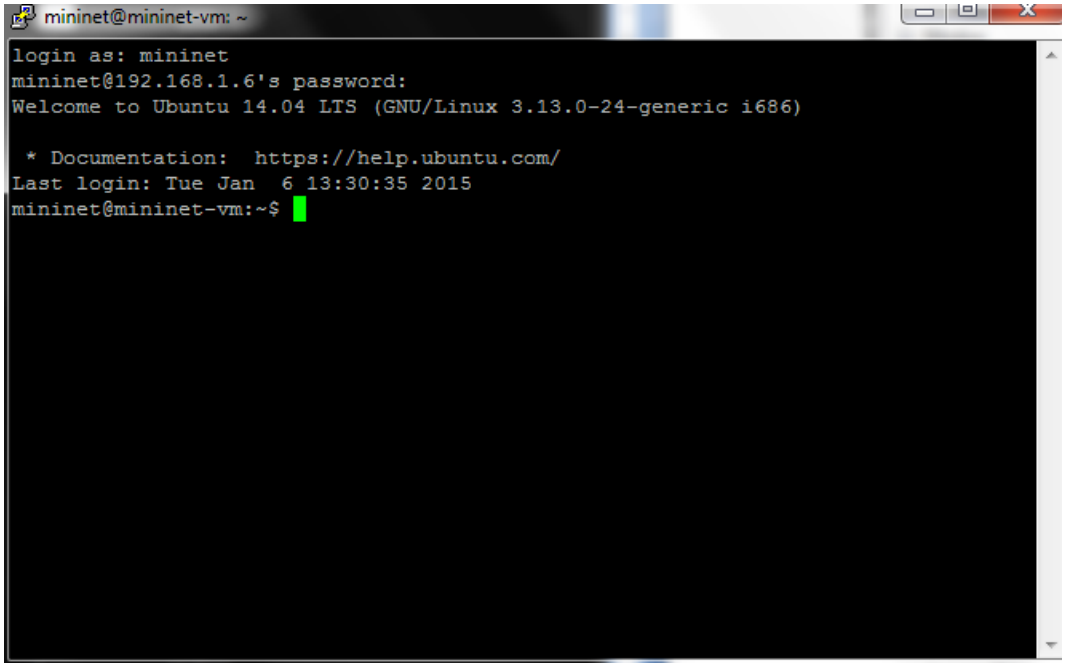
5. Εμφανίζεται το παρακάτω μήνυμα στο οποίο επιλέγουμε "Ναι" ώστε να θυμάται το "Putty" ότι είναι μια αξιόπιστη σύνδεση για εμάς.



Εικόνα Δ. 115 Οδηγός Εγκατάστασης: Putty ρυθμίσεις σύνδεσης

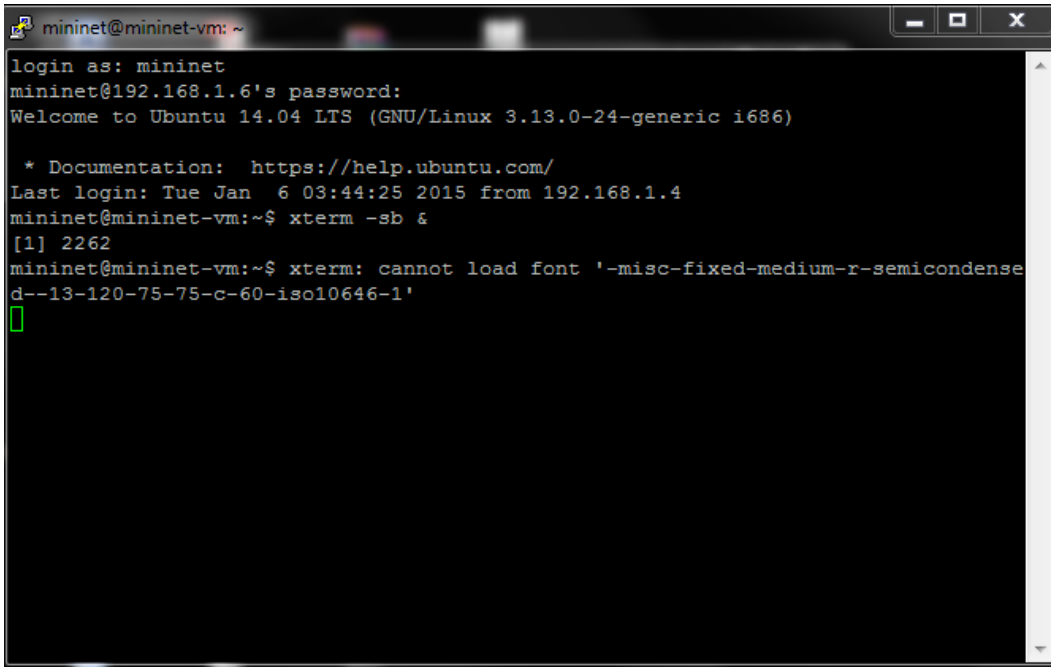
6. Εμφανίζεται η οθόνη login as και password του Mininet μέσα από το "Putty" εισάγω κανονικά mininet & mininet αντίστοιχα. Πλέον μέσω του putty έχω άμεση πρόσβαση στο Mininet. Έχουμε καταφέρει δηλαδή να έχουμε μια ασφαλή σύνδεση μέσω SSH με την εικονική μηχανή και συγκεκριμένα με το Mininet εξομοιωτή.

7. Πλέον μπορούμε να συνδεθούμε στο Mininet μέσω μιας σύνδεσης SSH.



Εικόνα Δ. 116 Οδηγός Εγκατάστασης: Mininet μέσω Putty

8. Αν τρέξουμε την εντολή: mininet@mininet-vm**:~$ xterm -sb &**



Εικόνα Δ. 117 Οδηγός Εγκατάστασης: Xterm & Putty

Τότε θα τρέξει μέσω του "Xming server" το "Xterm", εδώ να σημειώσουμε ότι το sb στο τέλος της εντολής το προσθέσαμε ώστε να έχουμε scroll back feature στο παράθυρο μας.

9. Οπότε με την παραπάνω εντολή ανοίγει το παρακάτω παράθυρο

Εικόνα Δ. 118 Οδηγός Εγκατάστασης: Xterm παράθυρο

10. Μπορούμε πλέον να τρέξουμε προγράμματα όπως το Wireshark στο Mininet Xterm window. Δίνοντας την εντολή:

mininet@mininet-vm:~$ **sudo apt-get wireshark**

μετά την εγκατάσταση του προγράμματος wireshark δίνουμε την εντολή:

mininet@mininet-vm:~$ **sudo wireshark &**

θα εκτελεσθεί το πρόγραμμα "Wireshark" [21] στην εικονική μηχανή του Mininet αλλά θα εμφανίζεται σε ένα X παράθυρο στον υπολογιστή μας.

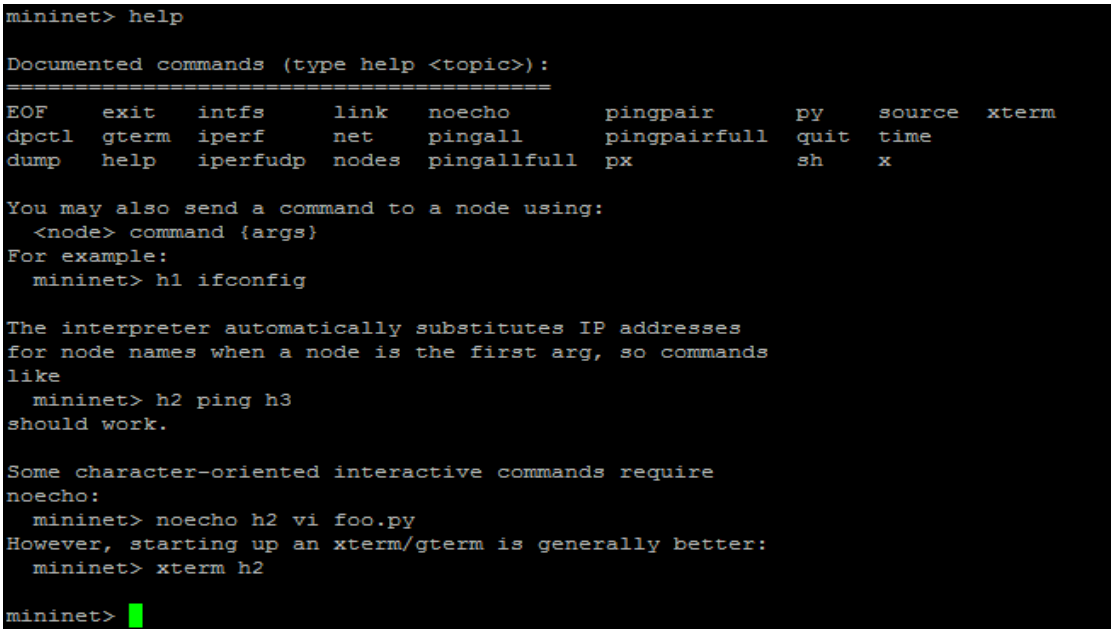
Έχουμε πλέον τελειώσει τα βήματα εγκατάστασης του Mininet και έχουμε επιτύχει την επικοινωνία μεταξύ της εικονικής μηχανής, του εξομοιωτή και του υπολογιστή μας μέσω ασφαλής σύνδεσης SSH. Τέλος έχουμε διαπιστώσει ότι μπορούμε να τρέξουμε προγράμματα όπως το "Wireshark" μέσω παραθυρικού περιβάλλοντος γεγονός το οποίο θα μας εξυπηρετήσει κατά την διάρκεια των προσομοιώσεων και των πειραμάτων που θα διεξάγουμε στην συνέχεια της διατριβής ενώ θα μας βοηθήσει να εξάγουμε συμπεράσματα ή ακόμα και γραφήματα.

Στο κείμενο της μεταπτυχιακής διατριβής όταν γράφουμε **mininet>** εννοούμε την διεπαφή του Mininet, εάν δηλώνουμε **$** εννοούμε το SSH τερματικό δηλαδή την χρήση της διεπαφής putty ως κανονικός χρήστης, ενώ αν γράφουμε # εννοούμε την χρήση του SSH τερματικού ως διαχειριστής.

Για την δημιουργία ενός δικτύου στο Mininet αρκεί μια εντολή. Για παράδειγμα η εντολή **$ sudo mn** δημιουργεί την ελάχιστη minimal τοπολογία η οποία είναι ένας OpenFlow Kernel μεταγωγέας που συνδέετε με δυο υπολογιστές καθώς και ένας Openflow ελεγκτής

Δ .4 Βασικές εντολές στο Mininet

Σε αυτό το σημείο θα αναφερθούμε σε βασικές εντολές του Mininet γεγονός που πραγματοποιείται αν πληκτρολογήσουμε "**Help**" στο CLI και παρατηρήσουμε την εμφανιζόμενη λίστα από διαθέσιμες εντολές στο Mininet.



Εικόνα Δ.26 Βασικές εντολές Mininet

Οι πιο σημαντικές από αυτές τις εντολές είναι οι παρακάτω, τις οποίες εξηγούμε περιληπτικά παρακάτω:

mininet>**nodes**: εμφανίζει τους κόμβους του δικτύου [39] [19]

mininet>**net**: εμφανίζει τις συνδέσεις μεταξύ των κόμβων [28]

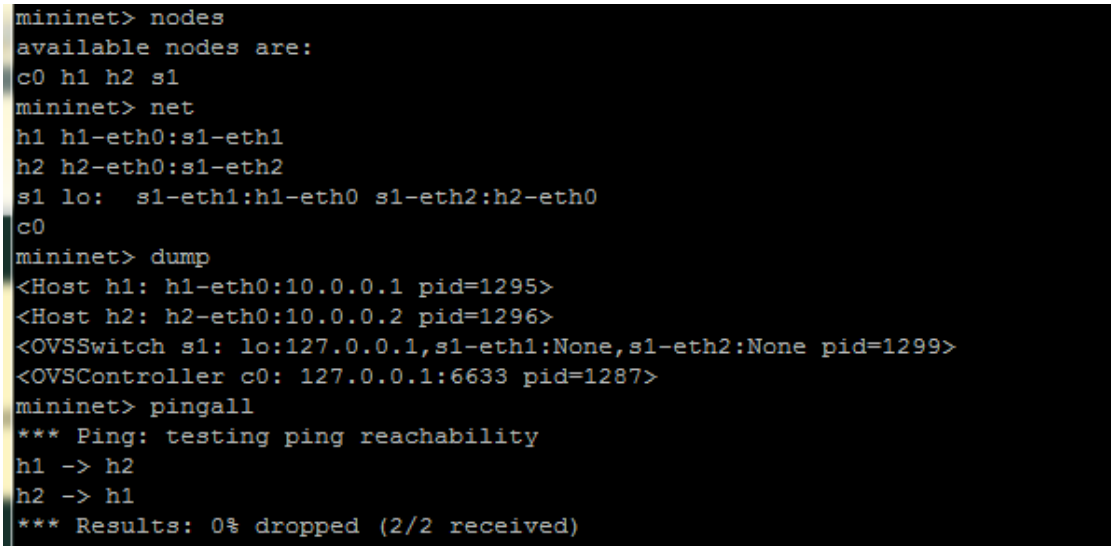
mininet>**dump** εμφανίζει dump πληροφορίες για όλους τους κόμβους

mininet>**pingall** κάνει ping σε όλους τους κόμβους και εμφανίζει τα αποτελέσματα του, δείχνει δηλαδή το εάν υπάρχει όντως επικοινωνία μεταξύ των κόμβων του δικτύου και τους χρόνους RTT.

mininet**>exit** Εάν θέλουμε να βγούμε από το Mininet και να σταματήσουμε το δίκτυο που έχουμε δημιουργήσει, τότε θα πρέπει να κάνουμε έξοδο από το CLI δίνοντας την παραπάνω εντολή**.**

mininet>**intfs**: εμφανίζει μια λίστα με τις διεπαφές που υπάρχουν στο δίκτυο.

mininet>**iperf**: Πραγματοποιεί ένα iperf TCP τέστ μεταξύ δύο κόμβων που καθορίζει ο χρήστης. Δημιουργεί δηλαδή κίνηση στο δίκτυο. Αποτελεί μια από τις σημαντικότερες εντολές. [40] [30]



Εικόνα Δ.27 Nodes, Dump & Pingall εντολές στο περιβάλλον Mininet.

Στο παραπάνω σχήμα παρατηρούμε τα αποτελέσματα των προαναφερθέντων εντολών στο περιβάλλον του Mininet για την συγκεκριμένη τοπολογία που δημιουργήθηκε. Όπως παρατηρούμε υπάρχει επικοινωνία μεταξύ των κόμβων του δικτύου όπως φανερώνει η εντολή **"Pingall"**. Συνεπώς με μια πρόχειρη ματιά, το δίκτυο λειτουργεί ομαλά.

Εάν θέλουμε να σταματήσουμε το δίκτυο μπορούμε όπως αναφέραμε προηγούμενα να οποιαδήποτε στιγμή να δώσουμε την εντολή "**exit**". Στη συνέχεια για να ολοκληρωθεί η διαδικασία και να διαγραφεί το δίκτυο από την μνήμη του Mininet που είχαμε δημιουργήσει δίνουμε την εντολή **$ sudo mn -c** [47]. Με αυτόν τον τρόπο σβήσαμε από την μνήμη του εξομοιωτή το δίκτυο και σταματήσαμε τον ελεγκτή και τον μεταγωγέα έτσι ώστε να μπορούμε να δημιουργήσουμε νέα πειράματα σε καθαρό περιβάλλον. Η παραπάνω διαδικασία επίσης είναι σημαντική και συστήνεται για τις περιπτώσεις στις οποίες το Mininet εμφανίζει σφάλματα ή κολλάει.

Δημιουργία Γραφημάτων στο Mininet

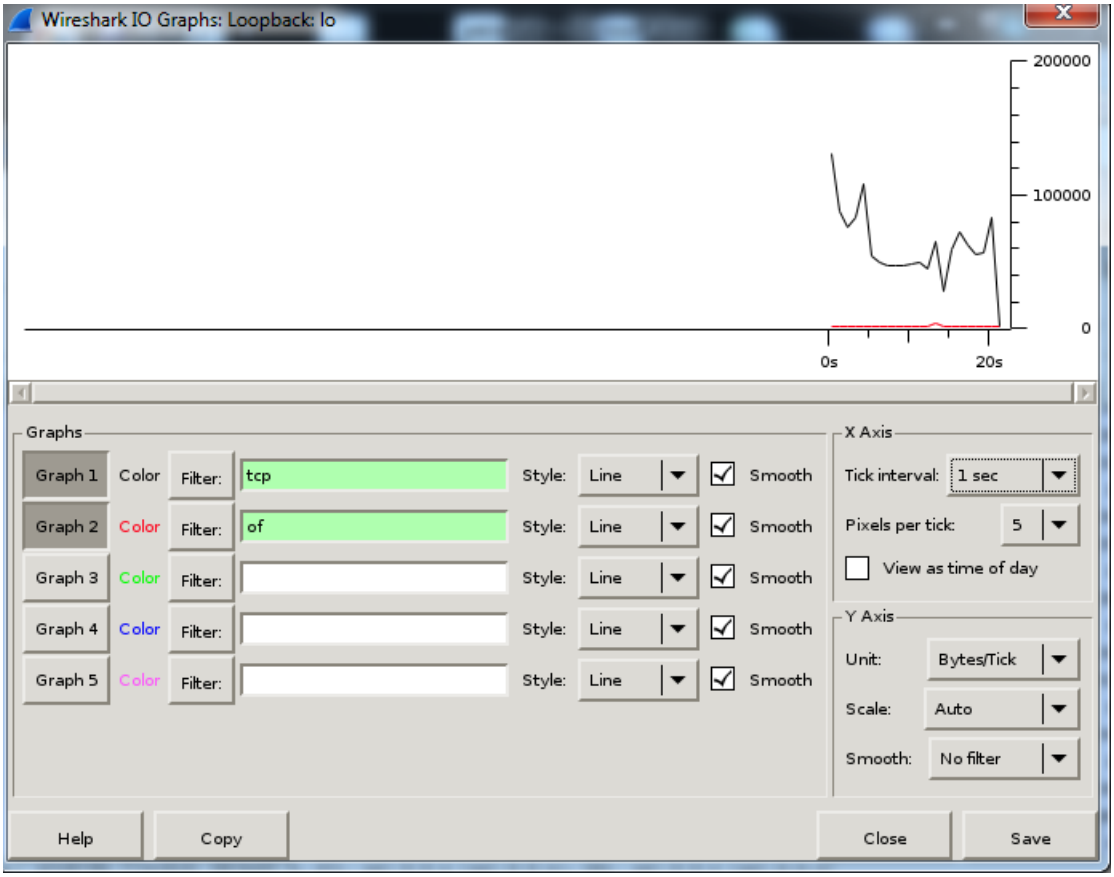
Για να ανοίξουμε το πρόγραμμα Wireshark και να δούμε τα πακέτα που ανταλλάσσονται μεταξύ μεταγωγέα και ελεγκτή στο Mininet πρέπει να εκτελέσουμε την εντολή:

$ **sudo wireshark &** [21]

όπως παρατηρούμε αυτή η εντολή δίνεται στην εικονική μηχανή και όχι στο CLI του Mininet ώστε να τρέξει το πρόγραμμα. Τα βήματα που ακολουθήθηκαν για να δεσμευτούν τα πακέτα που ανταλλάσσονται στους κόμβους του δικτύου αναλύονται παρακάτω:

1. Μετά την εντολή "sudo wireshark [22] &" θα έχουμε μια ειδοποίηση ότι εκτελούμε το πρόγραμμα Wireshark με καθολική - root πρόσβαση στην οποία επιλέγουμε "Οκ".
2. Αφού ξεκινήσει το Wireshark επιλέγουμε από την γραμμή μενού, "Capture" και στην συνέχεια "Interfaces", τσεκάρουμε τη διεπαφή "lo" (loopback interface) και πατάμε στο κουμπί "Start".
3. Σαν φίλτρο στην ανάλογη θέση θέτουμε το "of" που αντιστοιχεί στα πακέτα που ανήκουν στο πρωτόκολλο Openflow και στην συνέχεια πατάμε το κουμπί "Apply". Μπορούμε να ορίσουμε ως δεύτερο φίλτρο "TCP" ώστε να βλέπουμε την πληροφορία που ανταλλάσσεται στο δίκτυο και αφορά το συγκεκριμένο πρωτόκολλο.
4. Αν επιθυμούμε αφήνουμε ανοικτό το Wireshark κατά την δημιουργία του δικτύου και κατά την εκτέλεση εντολών κίνησης για να συλλάβουμε τα πακέτα που ανταλλάσουν ο μεταγωγέας και ο ελεγκτής κατά την εγκαθίδρυση της σύνδεσης τους δηλαδή τη στιγμή δημιουργίας του δικτύου στο Mininet.

Αν για παράδειγμα συλλάβουμε τα πακέτα κατά την διάρκεια δημιουργίας της ελάχιστης τοπολογίας στο Mininet αλλά και των εντολών που δόθηκαν στη προηγούμενη παράγραφο (nodes, dump, pingall) θα έχουμε το ακόλουθο IO γραφήμα με την βοήθεια του Wireshark:



Γράφημα Δ. 1 Wireshark IO Γράφημα, κατά την διάρκεια δημιουργίας της ελάχιστης τοπολογίας. Φαίνονται τα φίλτρα TCP & OF.